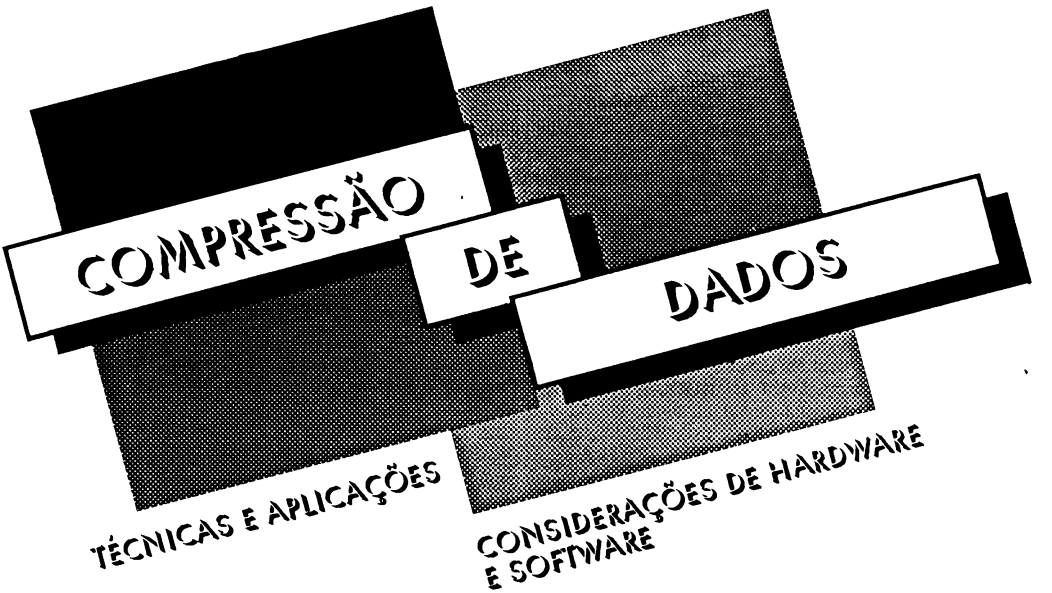
Gilbert Held

COMPRESSÃO DE DADOS

Técnicas e Aplicações
Considerações de Hardware e software

ÉRICA

WILEY


COMPRESSÃO
DE
DADOS
TÉCNICAS E APLICAÇÕES
CONSIDERAÇÕES DE HARDWARE
E SOFTWARE

Título do original em Inglês: Data Compression Techniques and Applications
Hardware and Software Considerations

Tradutora: Andréa Dell'Amore Santos
Revisores Técnico: Carlos Augusto P. Gomes
Antonio Carlos Barbosa

Dados Internacionais de Catalogação na Publicação (CIP)
(Câmara Brasileira do Livro, SP, Brasil)

Held, Gilbert, 1943-
Compressão de dados : técnicas e aplicações, considerações de hardware e software / Gilbert Held e Thomas R. Marshall ; tradução Andréa Dell' Amore Santos. -- São Paulo : Érica, 1992.

ISBN 85-7194-110-6

1. Dados - Compressão (Ciência da computação) I. Marshall, Thomas R. II. Título.

92-0087 CDD-005.764

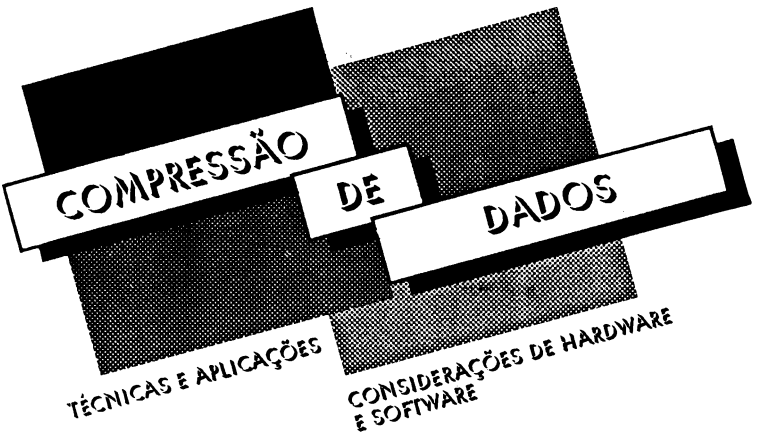
Indices para catálogo sistemático:

1. Compressão de dados : Ciência da computação 005.764

ISBN (edição original): 0 471 92941 7

ÉRICA **LIVROS ÉRICA EDITORA LTDA**
Rua Jarinú, 594 - Tatuapé - SP
Fone: (011) 294-8686

Gilbert Held
(4-Degree Consulting, Macon, Georgia, USA)
e
Thomas R. Marshall
(Autor do Software)



Tradução:
Andréa Dell'Amore Santos

Tradução Integral

Ano: 1998 97 96 95 94 93 92
Edição: 10 9 8 7 6 5 4 3 2 1

LIVROS ÉRICA EDITORA LTDA

ÉRICA

LIVROS ÉRICA EDITORA LTDA

Rua Jarinú, 594 - Tatuapé - São Paulo - S. P.
Telefone: (011) 294 - 8686
Cx. Postal 15617 - CEP: 03306
C.G.C. 50.268.838/0001-39

DEDICATÓRIA

À Beverly, Jonathan e Jessica por sua paciência e compreensão. Ao Dr. Alexander Ioffe e sua família, que encontrei em Moscou; possamos, um dia, nos encontrar de novo. À Harry R. Karp por sua orientação e instruções e ao meu pai Milton, que me ensinou a determinar como, quando os outros me perguntavam porque.

A Livros Érica Editora agradece ao **CARLOS AUGUSTO PEREIRA GOMES E ANTONIO CARLOS BARBOSA** *pela gentileza de terem revisado tecnicamente e complementado esta tradução com rotinas e programas em linguagem C, as quais se encontram no apêndice C.*

SOBRE O AUTOR

Gilbert Held é bacharel em Engenharia Eletrônica pela Faculdade Militar da Pensilvânia. Ele obteve seu mestrado em Engenharia Eletrônica, com ênfase em Ciência da Computação, pela New York University. Ele também possui os graus de mestrado em Administração de Empresas e de M.S.T.M. pela American University.

Gilbert Held é chefe do Departamento de Comunicações de Dados do Departamento de Gerenciamento de Pessoal do Governo dos Estados Unidos. Ele presta consultoria para diversas empresas e ministra vários cursos sobre Computadores e Teoria da Decisão, Gerenciamento de Sistemas de Informação e Comunicações de Dados, em diversas faculdades. Ele escreveu diversos livros e artigos.

PREFÁCIO

O objetivo original da primeira edição deste livro foi de proporcionar aos leitores um conhecimento profundo das técnicas, práticas e de fácil implementação, de compressão de dados. Ao escrever a segunda edição de Compressão de Dados, o objetivo original foi mantido. Contudo a abrangência e a profundidade dos assuntos abordados foram expandidos de maneira a incluir técnicas adicionais de compressão de dados, bem como exemplos de codificação desenvolvidos em linguagem de programação BASIC para software de compressão e descompressão de dados usando diferentes técnicas de compressão. Baseado na segunda edição, esta terceira edição representa uma expansão considerável das edições anteriores deste livro. As informações à respeito da estrutura e operação de vários algoritmos populares de compressão que chegaram ao mercado, desde a publicação da segunda edição deste livro, foram acrescentadas. As novas técnicas de compressão incluem Microcom Networking Protocol (MNP) compressão de dados Classe 5, MNP Classe 7 Compressão de Dados Avançada e o método de compressão V.42bis CCITT, baseado na técnica Lempel-Ziv. Além disso, diversos métodos para aumentar a eficiência das técnicas de compressão estatística e de compressão orientada a caractere, com base na experiência do autor nos últimos três anos, são, agora, compartilhados pelos leitores. No tocante à compressão orientada a caractere, adicionou-se um novo capítulo que discorre sobre os três métodos que podem ser usados pelos leitores para obter o caractere especial de indicação de compressão, que é a chave para implementar este tipo de compressão.

O desenvolvimento deste livro é resultado de uma série de seminários que o autor conduziu a respeito das características, operação e aplicação de 25 dispositivos de comunicações de dados. Durante estes seminários, uma pequena parte do tempo foi dedicada ao exame de alguns métodos de compressão de dados e como estes métodos poderiam aumentar a eficiência da transmissão. Na conclusão de cada seminário, a habitual série de perguntas dos participantes diziam respeito às fontes de informações sobre algoritmos adicionais de compressão de dados, técnicas de implementação e considerações de hardware e software. Tendo conduzido anteriormente uma pesquisa sobre documentação para desenvolver diversos módulos de software de compressão basea-

dos em microprocessador, percebi que existia uma lacuna nas literaturas, com relação aos métodos práticos de compressão de dados que poderiam ser aplicados aos problemas típicos de transmissão de dados. Com base no sucesso da primeira e segunda edições, parece que Compressão de Dados vêm contribuindo de forma significante para preencher esta lacuna. Espero que esta edição esclareça as dúvidas dos muitos leitores que me contataram durante os últimos anos para obter informações adicionais com relação à compressão de dados.

O primeiro capítulo deste livro aborda o fundamento lógico, a utilização das técnicas de compressão de dados e serve como introdução geral a um assunto chave inter-relacionado no livro - a compressão de dados e a eficiência da transmissão. Uma vez que a chave para uma implementação com sucesso das técnicas de compressão orientada a caractere é a seleção de um caractere apropriado de indicação de compressão, o capítulo 2 está centrado neste tópico. Devido a relação dentre os caracteres de indicação de compressão e os códigos de dados, o capítulo 2 revisa, primeiramente os quatro códigos de dados mais conhecidos e, então, examina os três métodos que os leitores podem usar para obter os caracteres de indicação de compressão. No capítulo 3, oito algoritmos distintos de compressão de dados são abordados, enquanto o capítulo 4 examina as técnicas de codificação estatística. O capítulo 5 focaliza o esforço da análise de dados exigida antes de alguém poder selecionar um método de compressão de dados ótimo ou quase ótimo. A interligação por software das rotinas de compressão de dados e outras considerações a respeito da compressão são apresentadas no capítulo 6. Os tópicos inclusos neste capítulo dizem respeito aos requisitos de memória, técnicas de posicionamento de módulo de software em buffers e considerações de temporização.

No capítulo 7, diversas categorias de dispositivos para compressão são examinadas, fornecendo ao leitor uma alternativa para desenvolver software de compressão. Outro tópico abordado neste capítulo diz respeito à economia na utilização e na operação do hardware e do software que executam a compressão.

Há dois apêndices neste livro. No apêndice A encontra-se uma listagem de programa em FORTRAN e em BASIC de um programa cuja execução permite a análise de freqüência de diversos tipos de arquivos de dados. Este programa pode ser usado pelos leitores para analisar seus arquivos de dados e determinar o método de compressão de dados que traz maior benefício ao se implementar. Além disso, o programa permite que se determine a susceptibilida-

de dos arquivos de dados a diferentes técnicas de compressão, antes mesmo de se implementar estas técnicas.

No apêndice B, os exemplos de codificação para comprimir e descomprimir dados apresentados em todo o livro, foram reunidos em um exemplo global cujo nome de programa mais apropriado foi: SHRINK (encolher). Os leitores são encorajados a revisar a listagem do programa e a usar o programa como ele aparece, ou a modificá-lo para satisfazer suas exigências particulares desde que não seja usado com fins comerciais.

Com a extinção da tarifa com desconto de volume conhecida como TELPAK, em conjunto com os aumentos de tarifa de linha privada, grandes usuários de comunicações de dados, infelizmente, se tornaram vítimas de uma taxa de inflação nas comunicações de 75 por cento. A compressão de dados representa um método de ter este custo sob controle. Como descrito neste livro, a maioria dos métodos de compressão é fácil de se implementar e o retorno de investimento na forma de redução de custos pode vir rapidamente. O efeito da compressão de dados sobre o tempo e o custo da transmissão pode ser resumido em uma palavra - ENCOLHA-O.

Gilbert Held

Macon, Georgia

NOTA: Foi incluso um novo apêndice, dito C, com Descrições e Listagens do programa em C.

SUMÁRIO

1 RACIONALIZAÇÃO E UTILIZAÇÃO 01

1.1 Compressão Lógica 02
1.2 Compressão Física 04
1.3 Os Benefícios da Compressão 05
1.4 Terminologia 06
1.5 Aplicações em Comunicações 08
1.6 Aplicação em Armazenamento de Dados 11
1.7 A Compressão de Dados e a Transferência de Informações 13

2 CÓDIGOS DE DADOS E CARACTERES INDICANDO COMPRESSÃO 31

2.1 Códigos de Dados 31
2.2 Selecionando Caracteres de Indicação de Compressão 37

3 TÉCNICAS DE COMPRESSÃO DE DADOS . 43

3.1 Supressão de Caracteres Nulos 44
3.2 Mapeamento de Bit 49
3.3 Comprimento de Fileira 59
3.4 Compactação de Meio Byte 81
3.5 Codificação Diatômica 112
3.6 Substituição de Padrões 131
3.7 Codificação Relativa 135
3.8 Operação no Modo de Formulários 142

4 CODIFICAÇÃO ESTATÍSTICA 151

4.1 Teoria da Informação 152
4.2 Codificação Huffman 158
4.3 Códigos Huffman Modificados 169
4.4 Codificação Shannon-Fano 176
4.5 Códigos de Vírgula 186
4.6 Compressão Auto-Adaptável 188
4.7 Métodos Modernos de Compressão Estatística 199

5 CONSIDERAÇÕES DO SISTEMA E ANÁLISE DE DADOS 217

5.1 Considerações do Sistema 217
5.2 Análise de Dados 220

6 CONSIDERAÇÕES DE LIGAÇÃO DE SOFTWARE 237

6.1 Posicionamento da Rotina de Compressão 237
6.2 Considerações de Temporização 242

7 USANDO PRODUTOS DE HARDWARE E SOFTWARE PARA EXECUTAR A COMPRESSÃO 247

7.1 Produtos de Hardware 248
7.2 Produtos de Software 256

APÊNDICE A - DESCRIÇÃO E LISTAGENS DO PROGRAMA DATANALYSIS 277

A.1 Programa FORTRAN: Descrição Operacional 277
A.2 Rotina Principal 277
A.3 Transferabilidade do Programa 283
A.4 Atribuições às Variáveis 284
A.5 Programa em BASIC: Descrição Operacional 286
A.6 Rotinal Principal 287
A.7 Atribuições às Variáveis 292
A.8 Listagem do Programa DATANALYSIS em FORTRAN . 294
A.9 Listagem do Programa DATANALYSIS em BASIC . . . 303

APÊNDICE B - DESCRIÇÃO E LISTAGENS DO PROGRAMA SHRINK 311

B.1 MERGEC.BAS e MERGED.BAS 312
B.2 Operações de Programa 319

APÊNDICE C - DESCRIÇÕES E LISTAGENS DO PROGRAMA EM C 323

Escrito por:

Carlos Augusto P. Gomes
Antonio Carlos Barbosa

CAP. 01

RACIONALIZAÇÃO E UTILIZAÇÃO

Na cronologia do desenvolvimento do computador, a transferência de informações em grande escala pela computação remota, e o desenvolvimento de armazenamento e de sistemas de recuperação de informações em massa vêm testemunhando um crescimento enorme. Em compasso com este crescimento, diversas áreas de problema desenvolveram-se, o que pode resultar em altos, mas desnecessários gastos.

Um problema é o chamado "banco de dados descontrolados". Aqui, o tamanho do banco de dados usado por uma organização para o armazenamento de informações e programas de recuperação torna-se cada vez maior, necessitando de drives de disco adicionais para sistemas on-line e rolos de fitas magnéticas para aqueles sistemas que podem ser processados em um ambiente "batch".

Aliado ao crescimento do tamanho dos bancos de dados, há um grande aumento no número de usuários e no período de uso por funcionário em locais remotos. Estes fatores resultam em quantias enormes de dados, sendo transferidos entre computadores e terminais remotos. De maneira a fornecer melhorias para as transferências de dados solicitadas, as linhas de comunicação e os dispositivos auxiliares, tais como: modems e multiplexadores, são continuamente atualizados por muitas organizações para permitir uma maior capacidade de transferência de dados.

Embora as soluções óbvias a estes problemas de armazenamento de dados e de transferência de informações sejam instalar dispositivos adicionais de armazenamento e expandir os recursos de comunicações existentes, ocorre que haveria, também, a neces-

sidade de um aumento adicional de equipamento e custos operacionais da empresa. Um método que pode ser empregado para aliviar uma parte do problema de armazenamento de dados e de transferência de informações, é através da representação de dados por códigos mais eficientes. Se você examinar um banco de dados de uma empresa ou monitorar uma linha de transmissão, haverá uma grande chance de os caracteres individuais, que compõem o banco de dados e a seqüência de transmissão, poderem ser codificados de forma mais eficaz. As duas técnicas que podem resultar em representações mais eficazes dos dados codificados são a compressão de dados lógica e física.

1.1 COMPRESSÃO LÓGICA

Quando um banco de dados é projetado, um dos primeiros passos do analista é obter a máxima redução dos dados. Esta redução dos dados resulta da eliminação dos campos redundantes de informação e da representação dos elementos de dados nos campos restantes, com menos indicadores lógicos quanto possível.

Embora a compressão lógica seja dependente dos dados e o método empregado possa variar de acordo com a percepção do analista, os dois exemplos seguintes ilustrarão a facilidade da implementação e os benefícios desta técnica de compressão.

Um exemplo simples de compressão lógica de dados é o campo de função no banco de dados de um funcionário. Vamos supor que 30 posições alfanuméricas são alocadas para este campo. Se o campo for fixo, as funções como, por exemplo, a descrição da função com 17 caracteres 'LAVADOR DE PRATOS' teria 13 espaços em branco inseridos no restante do campo. Assim, 30 milhões de caracteres de armazenamento seriam necessários para o campo de função de um milhão de empregados. Suponha que há 32.768 cargos diferentes. Em vez de indicar o título do cargo, você poderia designar um código equivalente de dado de cinco dígitos, eliminando 25 posições de caractere por campo. O tamanho do campo poderia ser reduzido ainda mais ao alocar o valor binário referente a 1 ou mais caracteres, no código de função. Como exemplo, um caractere de 8 bits poderia representar 2^8 - 1 ou 255 valores ou códigos de função distintos, ligando dois caracteres de 8 bits através de software apropriado, forneceria 2^{16} - 1 ou 65.535 códigos distintos. Isto reduziria o tamanho do campo de 30 para 2 caracteres, economizando 28 milhões de caracteres de armazenamento. Se nossa contagem começar em zero, em vez do início convencional que é 1, um caractere de 8 bits poderia representar 256 códigos,

enquanto um caractere de 16 bits poderia ser empregado para representar 65.536 valores diferentes.

O segundo exemplo de compressão lógica fica em um campo de data. Este tipo de campo ocorre freqüentemente em bancos de dados. Normalmente, os valores numéricos equivalentes dos subcampos de representação do dia, mês e anos são usado no lugar da anotação por extenso. Assim, 01 04 81 representaria 1 de abril de 1981. Enquanto esta compressão lógica resulta em 6 caracteres numéricos de armazenamento, uma redução de dados adicional pode resultar do armazenamento da data com valor binário. Uma vez que o dia nunca excederá 31, 5 bits são suficientes para representar o campo de dados. De forma semelhante, 4 bits seriam usados para representar o valor do mês, enquanto 7 bits poderiam representar 127 anos, permitindo um ano relativo se estendendo de 1900 a 2027.

A compressão lógica, usando a representação binária e numérica, está ilustrada na Figura 1.1 para o exemplo anterior de campo de data. É interessante observar que o emprego da representação binária reduz o campo de data para 16 dígitos binários ou dois caracteres concatenados de 8 bits de armazenamento. Como discutido, muitos métodos de compressão lógica podem ser considerados por um analista durante o processo de projeto do banco de dados. Cada método pode resultar em um grau distinto de redução de armazenagem de dados. De forma correspondente, quando bancos de dados ou partes destes bancos de dados comprimidos logicamente são transmitidos entre locais, o tempo de transmissão é reduzido, uma vez que poucos caracteres de dados são transmitidos.

Exemplo por	DIA	MÊS	ANO
extenso	1	ABRIL	1981
Compressão lógica usando representação numérica			
Exemplo	01	04	81
Compressão lógica usando representação binária			
Exemplo	00001	0100	1010001

Figura 1.1 - Métodos de compressão lógica. A compressão lógica pode resultar da representação binária, numérica ou alfanumérica dos dados de uma notação abreviada.

Enquanto a compressão lógica puder ser uma ferramenta eficiente para minimizar o tamanho de um banco de dados, ela somente poderá reduzir o tempo de transmissão, quando os dados transmitidos forem comprimidos logicamente. Assim, a transmissão dos dados de pergunta e resposta, não é normalmente afetada, uma vez que estes dados são codificados tipicamente como entidades distintas e separadas com representação apropriada de bits na codificação de cada caractere. De forma semelhante, a ocorrência de repetição de padrões e grupos de caracteres, que são normalmente apresentados em relatórios transmitidos dos sistemas de computação aos terminais, não são afetados. Para estas situações, uma redução do tempo de transmissão de dados depende da compressão física dos dados que são encontrados.

1.2 COMPRESSÃO FÍSICA

A compressão física pode ser vista como o processo de reduzir a quantidade de dados antes de entrarem em um meio de transmissão e como o processo de expandir estes dados, em seu formato original, para serem recebidos em local distante. Embora a compressão lógica e a compressão física possam resultar em redução no tempo de transmissão, existem aplicações distintas e diferentes para as duas técnicas. A compressão lógica é, normalmente, usada para representar os bancos de dados de maneira mais eficiente e não considera a freqüência de ocorrência de caracteres ou grupos de caracteres.

A compressão física tira vantagem do fato que quando os dados são codificados como entidades distintas e separadas, as probabilidades de ocorrência de caracteres e de grupos de caracteres diferem umas das outras, uma vez que os caracteres de ocorrência freqüente são codificados em muitos bits da mesma forma que aqueles caracteres que ocorrem raramente, a redução dos dados se torna possível ao codificar os caracteres de ocorrência freqüente em códigos curtos de bits e de maneira inversa, representar os caracteres de ocorrência rara por códigos mais longos de bits. Como na compressão lógica, existem muitas técnicas de compressão física. Algumas técnicas substituem cadeias repetidas de caracteres por um caractere especial indicador de compressão e um caractere de contagem de quantidade. Outras técnicas substituem os caracteres de ocorrência freqüente por um código binário curto, enquanto os caracteres encontrados raramente são substituídos por códigos binários mais longos. No capítulo 3, oito métodos distintos de compressão física são abordados em detalhes. No

restante deste livro, focalizaremos nossa atenção na compressão física de dados.

1.3 OS BENEFÍCIOS DA COMPRESSÃO

Quando a compressão de dados é usada para reduzir o volume de dados armazenados, o tempo global de execução do programa pode ser reduzido. Isto ocorre porque a redução no armazenamento causará uma redução no número de acessos ao disco, enquanto a codificação e a decodificação necessárias pela técnica empregada de compressão, resultarão em instruções adicionais de programa sendo executadas. Já que o tempo de execução de um conjunto de instruções de programa é, normalmente, e de forma significativa, menor do que o tempo necessário para se acessar e transferir os dados para o dispositivo periférico, o tempo global de execução do programa pode ser reduzido.

Com relação à transmissão de dados, a compressão fornece diversos benefícios ao planejador de rede, além de fornecer economia potencial de custo, associada ao envio de menos dados pela rede telefônica chaveada, onde o custo da chamada baseia-se, geralmente, em sua duração. Em primeiro lugar, a compressão pode reduzir a probabilidade de ocorrência de erros de transmissão, já que poucos caracteres são transmitidos, quando os dados estão comprimidos e a probabilidade de ocorrência de erro permanece constante. Em segundo lugar, já que a compressão aumenta a eficiência, ela pode reduzir ou mesmo eliminar jornadas extras de trabalho. Finalmente, ao converter texto, que é representado por um código convencional, como, por exemplo, o padrão ASCII, em um código diferente, os algoritmos de compressão podem fornecer um nível de segurança contra o monitoramento ilícito.

Em comunicações de dados, a transferência de dados comprimidos através de um meio, resulta em um aumento na taxa efetiva de transferência de informações; embora a taxa real de transferência de dados, expressa em bits por segundo, se mantenha a mesma. A compressão de dados pode ser implementada na maioria dos hardwares existentes, através do software ou através do uso de um dispositivo especial de hardware que incorpore uma ou mais técnicas de compressão.

Um diagrama de blocos básico de compressão de dados encontra-se ilustrado na Figura 1.2. Apresentado como uma caixa preta, a compressão e a descompressão podem ocorrer dentro do processador do usuário, incluindo computadores pessoais, terminais inteligentes ou em um dispositivo externo ao processador, tal como

um componente especializado de comunicações. Os primeiros, entre estes componentes, foram os concentradores de dados e os multiplexadores estatísticos, por volta dos anos 70 e início dos anos 80. A partir da metade dos anos 80, aproximadamente, aconteceu uma revolução na utilização de produtos que executavam a compressão de dados nas áreas de modems de rede chaveada, armazenamento em disco usado em microcomputadores e mainframes. Literalmente, centenas de fabricantes de modem para rede chaveada incorporaram uma variedade de algoritmos de compressão de dados em seus produtos, enquanto diversos projetistas de software comercializaram produtos que aumentavam a capacidade de armazenamento de discos ao comprimir os dados antes de armazená-los.

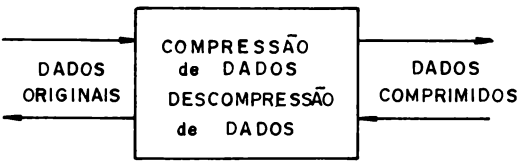


Figura 1.2 - Diagrama de blocos básico de compressão de dados. Um fluxo de dados originais operando de acordo com um ou mais algoritmos de compressão, resulta na geração de um fluxo de dados comprimidos.

Para examinar, detalhadamente, uma parte dos benefícios resultantes do emprego de uma ou mais técnicas de compressão, é necessária uma revisão de algumas terminologias fundamentais de compressão.

1.4 TERMINOLOGIA

Como ilustra a figura 1.2, um fluxo de dados originais é manipulado de acordo com um determinado algoritmo para gerar um fluxo de dados comprimidos. Esta compressão do fluxo de dados originais é, às vezes, chamada de processo de codificação. Como resultado, o fluxo de dados comprimidos é também chamado de fluxo de dados codificados. Ao reverter o processo, o fluxo de dados comprimidos é descomprimido para se reproduzir o fluxo de dados originais. Uma vez que o processo de descompressão resulta na decodificação do fluxo de dados comprimidos, o resultado é, às vezes, chamado de fluxo de dados decodificados. Usaremos os termos fluxo de dados originais e fluxo de dados decodificados

como sinônimos, bem como os termos fluxo de dados comprimidos e fluxo de dados codificados.

O grau de redução de dados obtido como o resultado do processo de compressão é conhecido como razão de compressão. Esta razão mede a quantidade de dados comprimidos em comparação com a quantidade de dados originais, de maneira que (Ruth e Krentzler, 1972):

$$\text{Razão de compressão} = \frac{\text{Comprimento da cadeia de dados originais}}{\text{Comprimento da cadeia de dados comprimidos}}$$

Da equação acima, é óbvio que quanto maior a razão de compressão, mais eficaz é a técnica de compressão empregada. Outro termo usado ao se falar de compressão é a figura de mérito, onde:

$$\text{Figura de mérito} = \frac{\text{Comprimento da cadeia de dados comprimidos}}{\text{Comprimento da cadeia de dados originais}}$$

A figura de mérito é a recíproca da razão de compressão e deve sempre ser menor que 1 para que o processo de compressão seja efetivo. A fração da redução de dados é: um menos a figura de mérito. Assim, uma técnica de compressão, que resulte em um caractere de dado comprimido para cada três caracteres do fluxo de dados originais, teria uma taxa de compressão de 3, uma figura de mérito de 0,33 e uma fração de redução de dados de 0,66.

Os dados randômicos puros não deveriam, por definição, ter qualquer redundância. Enquanto a taxa de compressão para estes tipos de dados deve ser unitária, em muitos casos, o projeto ou aplicação inadequados de um algoritmo de compressão pode resultar em um grau de expansão de dados, resultando na queda da taxa de compressão para um valor menor que o unitário.

Quando os dados são comprimidos, a taxa de compressão variará, proporcionalmente, à susceptibilidade aos dados ao algoritmo ou algoritmos usados. Assim, você deve voltar sua atenção para a taxa de compressão média e não para a taxa alcançada em uma situação particular. Em geral, pode-se esperar que bons algoritmos alcancem uma taxa média de compressão de dados de 1,5 , enquanto excelentes algoritmos, baseados em técnicas sofis-

ticadas de processamento, alcançarão uma taxa média de compressão que ultrapassa 2,0 .

1.5 APLICAÇÕES EM COMUNICAÇÕES

Para obter uma visão global de alguns dos benefícios disponíveis através da incorporação da compressão de dados, podemos levar em conta uma aplicação de comunicação de dados típica. Como ilustrado na parte superior da figura 1.3, um terminal batch remoto está conectado ao computador central com a transmissão ocorrendo a uma taxa de 9,6 kbps. Vamos supor que os dados a serem transmitidos não fossem comprimidos. Se através da programação de um ou mais algoritmos de compressão ou da instalação de um dispositivo de compressão em hardware, fosse obtida uma taxa de compressão de 2, diversas alternativas podem ser obtidas com respeito a sua metodologia de comunicações de dados. Primeiro, nosso tempo de transmissão de dados é reduzido uma vez que a taxa efetiva de transferência de informação aumentou para, aproximadamente 19,2 kbps como mostra a parte central da figura 1.3. Ignorando a sobrecarga de processamento do software de comunicações, o tempo de transmissão de dados é dividido em dois. Assim, você pode, agora, considerar que o uso do terminal batch remoto em outro processamento de aplicativos remoto ou, talvez, uma cara jornada extra ou parte dela possa ser aliviada. Na parte inferior da figura 1.3 está ilustrada outra opção do usuário. Aqui, a taxa de transmissão pode ser reduzida para 4800 bps. Com uma taxa de compressão de 2, isto equivale a uma taxa de transferência de informações de 9600 bps. Ao diminuir a taxa de transmissão de dados, os modems de 9600 bps, que são mais caros, podem ser substituídos por modems de 4800 bps e o condicionamento de linha, normalmente exigido ao se transmitir dados a 9600 bps, pode ser eliminado, resultando em uma redução adicional de custo.

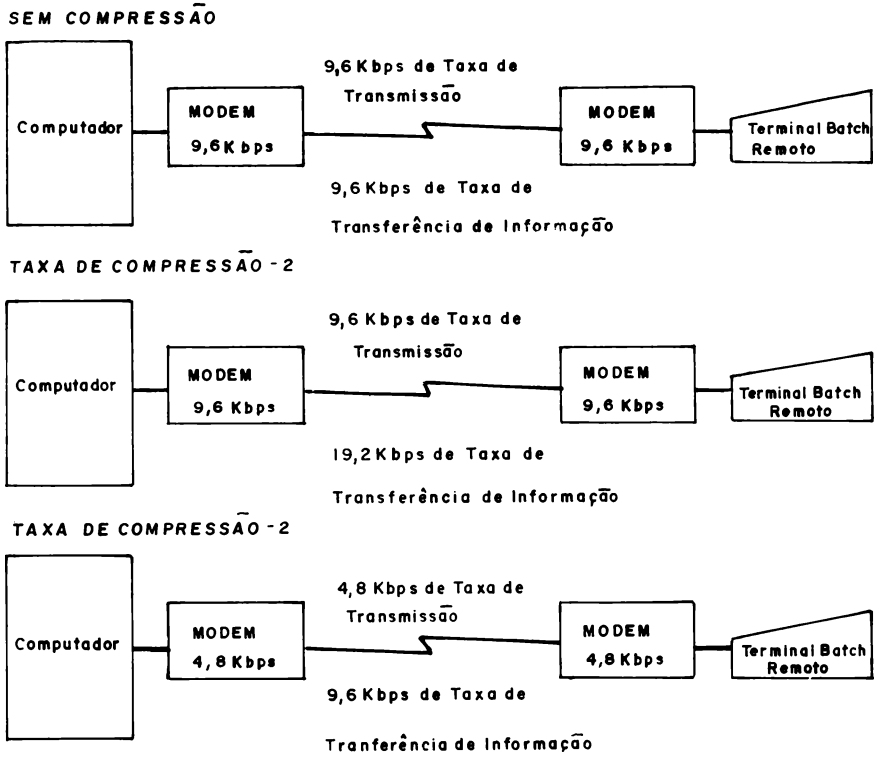


Figura 1.3 - A compressão de dados afeta a taxa de transferência de informação (ITR); através do uso da compressão de dados, a metodologia e estrutura da instalação de comunicações de dados podem ser alteradas.

Um segundo tipo de aplicação em comunicações que pode se beneficiar da utilização da compressão de dados, encontra-se ilustrado na figura 1.4. Uma rede multiponto típica é apresentada na parte superior da figura 1.4, conectando terminais de diversas localizações geográficas ao local do computador, via uma linha de comunicação comum. Tipicamente, a atividade de transmissão dos terminais é o fator preponderante que limita a linha multiponto em um certo número máximo de pontos. Na parte inferior da Figura 1.4, supõem-se que os modems com execução de compressão foram substituídos por modems convencionais usados na configuração original multiponto. Já que a compressão de dados em uma linha multiponto reduz o fluxo de dados na linha, sua utilização habilitará normalmente pontos adicionais a serem acrescentados à linha com

um limitante que é o desempenho da linha, onde os retardos afetam o tempo de resposta dos terminais ligados a cada ponto. Neste exemplo em particular, assume-se que o uso de modems com execução de compressão permite um aumento de 4 para 6 no número de ponto de linha.

Para os modems de rede chaveada, a compressão de dados fornece a possibilidade de se obter uma capacidade de transferência de informações por uma fração do custo de modems de maior velocidade. Isto ocorre devido aos complexos esquemas de modulação, usados pelos modems operando a 9600 bps se comparados àqueles com esquemas menos complexos, usados pelos modems operando a 2400 bps. Ao incorporar um algoritmo de compressão em um modem de 2400 bps, fica possível alcançar um desempenho de transferência de dados entre 4800 e 9600 bps com o custo adicional de um chip ROM (memória somente de leitura), que custa, aproximadamente, US$ 5,00 no atacado. Em comparação, o conjunto de circuitos analógicos necessários para que o modem opere a 9600 bps, custa em torno de US$ 100 a mais do que um conjunto de circuitos necessários para um modem de 2400 bps. Já que a maioria dos fabricantes vende seus produtos de três a cinco vezes

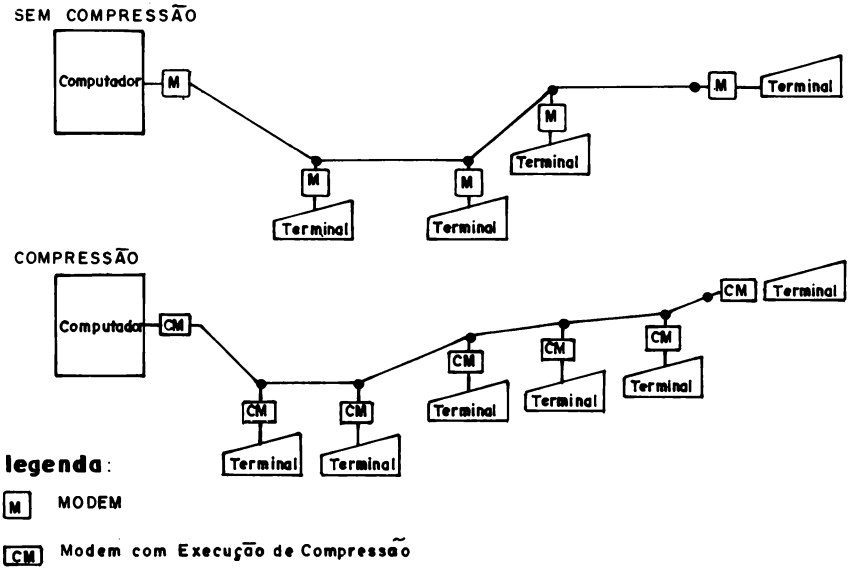


Figura 1.4 - A compressão de dados em uma linha multiponto reduz o fluxo de dados na linha, permitindo a operação de terminais adicionais.

o preço de custo, o uso da compressão em modems de baixa velocidade pode ser traduzido em economia significativa no preço de varejo, em comparação com o custo de modems operando a taxas mais elevadas. Além disso, a incorporação da compressão de dados em modems de alta velocidade para rede chaveada, fornece aos usuários a capacidade de transferência de informação, além da capacidade sustentada pela atual tecnologia de modem.

1.6 APLICAÇÃO EM ARMAZENAMENTO DE DADOS

Para ilustrar a aplicabilidade da compressão de dados em aplicações de armazenamento de dados, vamos examinar como os dados são armazenados em um disquete. Durante o processo de formatação, o computador divide o espaço de gravação em trilhas, que são círculos concêntricos. Os disquetes usados com os microcomputadores IBM PC e PC XT possuem 40 trilhas, numeradas de 0 na borda externa a 39 na borda interna. Cada trilha é subdividida em setores, sendo que o número de setores por trilha é função da capacidade de armazenamento do disquete.

A figura 1.5 ilustra a relação entre trilha e setor para os disquetes de 5,25 polegadas usados com os microcomputadores IBM PC e PC XT. Os disquetes, usados com estes computadores, têm 40 trilhas e oito ou nove setores por trilha, com o número de setores por trilha baseado no parâmetro especificado pelo comando FORMAT. Cada setor pode armazenar até 512 caracteres e representa a quantidade mínima de dados que podem ser armazenados ou recuperados em uma operação de leitura/escrita do disco. Se você escrever um caractere em um arquivo, você usará um setor de 512 caracteres, enquanto salvar um arquivo de 513 caracteres exigiria o uso de dois setores.

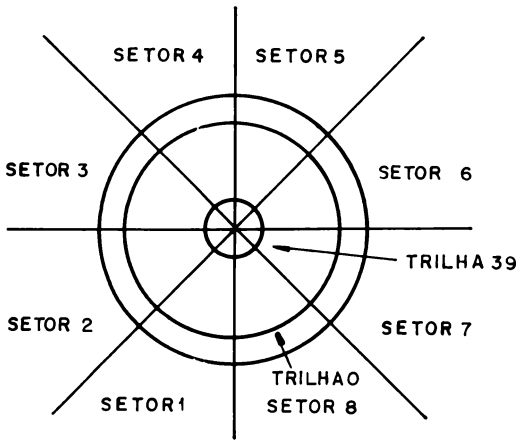


Figura 1.5 - A relação entre a trilha e o setor (observação: os disquetes podem ser formatados para ficarem com oito ou nove setores por trilha).

A capacidade total de armazenamento dos disquetes de dupla face de 5,25 polegadas, formatados para ficarem com oito setores por trilha é:

2 faces x 40 trilhas/face x 8 setores/trilha x 512 caracteres/setor =

= 327.680 caracteres.

Vamos supor que você deseja transferir um arquivo que contenha 350.000 caracteres. Sem um programa de compressão você não pode armazenar este arquivo em um disquete. Além disso, se o arquivo não for capaz de ser subdividido, como um arquivo binário, talvez você tenha que levar em consideração uma expansão na capacidade de armazenamento de seu computador, adicionando um disco rígido ou um drive de disco flexível de maior capacidade. Como alternativa, você pode transferir uma versão comprimida do arquivo. Hoje, muitos bulletin-boards usam programas de domínio público para arquivar arquivos no formato comprimido, tanto para a redução do tempo de transferência, como para a redução da necessidade de armazenamento. Se assumirmos que uma taxa de compressão 2:1 é alcançada, o arquivo de 350.000 caracteres seria comprimido para 175.000 caracteres. O arquivo não só caberia no disquete, mas, além disso, poderia ser armazenado usando 342 setores. Isto deixaria 298 setores disponíveis para outros propósitos, como armazenar um programa de descompressão. Uma vez

transferido, você pode expandir o programa na memória ou, se tiver um disco rígido, expandir o programa nesta unidade de armazenamento. Se você quiser distribuir o programa para outros usuários de microcomputador, poderia considerar a distribuição da versão comprimida do programa. Isto permitiria que você distribuísse o programa em um único disquete. Em comparação, a distribuição da versão normal do arquivo necessitaria de dois disquetes se o arquivo pudesse ser separado logicamente.

Para maiores informações a respeito da operação e utilização dos dois programas desenvolvidos para comprimir arquivos de microcomputadores IBM PC e compatíveis, leia o capítulo 7. Além disso, os leitores podem levar em consideração a compra do disco de conveniência que acompanha este livro e que contém estes programas em seu formato original.

1.7 A COMPRESSÃO DE DADOS E A TRANSFERÊNCIA DE INFORMAÇÕES

Quando os dados são transmitidos entre terminais, entre terminal e computador ou entre dois computadores, diversos fatores de retardo podem ser encontrados e que, acumulativamente, afetam a taxa de transferência de informações. Os dados transmitidos através de transmissão devem ser convertidos em um formato aceitável para aquele meio. Quando dados digitais são transmitidos por linhas telefônicas analógicas, os modems devem ser empregados para converter os pulsos digitais da máquina em um sinal modulado aceitável para a transmissão por um circuito telefônico analógico. O tempo entre o primeiro bit que entra no modem e o primeiro sinal modulado, gerado pelo dispositivo, é conhecido como tempo de retardo interno do modem. Já que dois dispositivos são necessários para um circuito ponto-a-ponto, o tempo de retardo interno total, encontrado durante a seqüência de transmissão, equivale a duas vezes o tempo de retardo interno do modem. Estes tempos podem variar de alguns a 10 ou mais milissegundos (ms). O segundo retardo encontrado em um circuito é em função da distância entre os pontos, conhecida como tempo de retardo de propagação ou de circuito. Este é o tempo necessário para que o sinal seja propagado ou transferido pela linha até o fim desta. O tempo de retardo de propagação pode chegar a, aproximadamente, 1 milissegundo para cada 241 km de circuito, acrescido de 12 milissegundos, para se obter o tempo total.

Uma vez que os dados são recebidos na extremidade, estes devem ser trabalhados, resultando em um retardo de processamento que se refere à função do computador ou do terminal empregado. O mesmo ocorre com a quantidade de dados transmitidos que devem ser trabalhados. O tempo de retardo de processamento pode variar de alguns milissegundos, onde uma simples verificação de erro é executada para determinar se os dados transmitidos foram recebidos corretamente, até muitos segundos, onde uma busca de um banco de dados deve ocorrer em resposta a uma consulta transmitida. Cada vez que o sentido da transmissão muda em um protocolo half-duplex típico, os sinais de controle mudam no modem associado ao computador e no modem associado às interfaces de terminal. O tempo solicitado para chavear os sinais de controle para se mudar o sentido da transmissão, é conhecido como tempo de retorno de linha e pode resultar em retardos de até 250 milissegundos ou mais, dependendo do protocolo de transmissão empregado. Podemos denotar o efeito da compressão de dados ao examinar o protocolo de transmissão, geralmente conhecido como comunicações BISYNC e algumas de suas derivações.

As Comunicações BISYNC

Um dos protocolos de transmissão mais freqüentemente empregado é a estrutura de controle de comunicações BISYNC (Binary Synchronous Communications). Esta estrutura de controle de linha foi introduzida em 1966, pela International Business Machine Corporation (IBM) e é usada para a transmissão por muitos dispositivos de média e alta velocidade, incluindo terminais e sistemas de computação. O BISYNC fornece um conjunto de regras que regem a transmissão síncrona de dados codificados binariamente. Enquanto este protocolo pode ser usado com uma variedade de códigos de transmissão, está limitado para o modo de transmissão half-duplex e requer a confirmação de recebimento de cada bloco de dados transmitidos. No processo de evolução, vários protocolos síncronos foram desenvolvidos para suplementar ou servir como substituto ao BISYNC. O mais proeminente foi o protocolo HDLC (high-level data link control) definido pela International Standard Organization (ISO).

A diferença chave entre os protocolos BISYNC e HDLC é que o BISYNC é um half-duplex com estrutura de controle de transmissão orientada a caractere, enquanto o HDLC é orientado a bit com a estrutura de controle de transmissão full duplex. Podemos investigar a eficiência destas estruturas básicas de controle de transmissão e o efeito da compressão de dados sobre sua eficiência de

transferência de informações. Para assim fazê-lo, um exame de alguns procedimentos típicos de controle de erro é solicitado em primeiro lugar.

Controle de Erro

O procedimento de controle de erro, mais freqüentemente empregado, é conhecido como solicitação automática para repetição (ARQ). Neste tipo de procedimento de controle, na detecção de um erro, uma solicitação é feita pela estação de recebimento para a estação de envio para retransmitir a mensagem. Dois tipos de procedimentos ARQ foram desenvolvidos: 'stop and wait ARQ' e 'go back *n* ARQ', que é, às vezes, chamado de ARQ contínuo.

'Stop and wait ARQ' é um tipo simples de procedimento de controle de erro. Aqui, a estação de transmissão pára no final de cada bloco e espera por uma resposta do terminal de recebimento, com referência à exatidão do bloco (ACK) ou erro (NAK), antes de transmitir o próximo bloco. Este tipo de procedimento de controle de erro encontra-se ilustrado na figura 1.6. Aqui, o tempo entre os blocos transmitidos é chamado de tempo morto, que age para reduzir a taxa efetiva de dados no circuito.

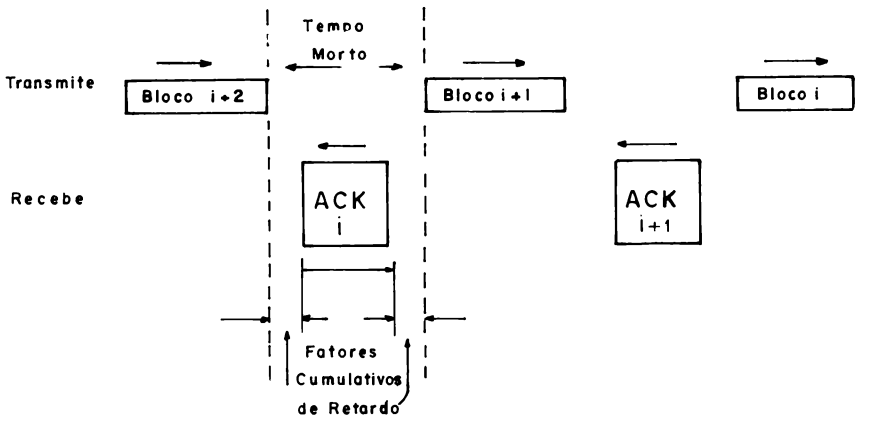


Figura 1.6 - Stop and wait ARQ. Neste tipo de procedimento de controle de erro, o receptor transmite uma confirmação de recebimento depois de cada bloco. Isto pode resultar em uma quantia significativo de tempo de retardo cumulativo entre os blocos de dados.

Quando o modo de transmissão é half-duplex, o circuito deve mudar o sentido duas vezes para cada bloco transmitido; uma vez para receber a resposta (ACK ou NAK) e outra vez para reassumir a transmissão. Estas mudanças de sentido da linha, bem como os fatores de tempo de retardo de propagação, de tempo de processamento de mensagem da estação e de tempo de retardo interno do modem, contribuem para aquilo que é apresentado como fatores cumulativos de retardo.

Quando o tipo 'go back *n* ARQ' de procedimento de controle de erro é empregado, o tempo morto pode ser reduzido substancialmente a ponto de ser insignificante. Um modo de implementar este tipo de procedimento de controle de erro, é através da utilização de um canal inverso simultâneo para a sinalização da aceitação de recebimento como ilustrado na figura 1.7. Neste tipo de modo de operação, a estação receptora envia de volta a resposta ACK ou NAK pelo canal inverso para cada bloco transmitido. Se o canal primário operar a uma taxa de dados muito mais alta do que o canal inverso, muitos blocos podem ser recebidos antes da estação de transmissão receber o NAK, em resposta a um bloco que na estação receptora apresentou erro. O número de blocos que você pode voltar atrás para solicitar uma retransmissão, *n*, é uma função do tamanho do bloco e da área de buffer disponível nas máquinas e terminais, que servem como estações de recepção e transmissão, e da razão entre as taxas de transferência de dados dos canais primários e inversos, e o tempo de processamento necessário para computar o caractere de verificação de bloco e transmitir a aceitação de recebimento. Para o último, este tempo é mostrado na figura 1.7, como pequenas brechas entre os blocos ACK e NAK.

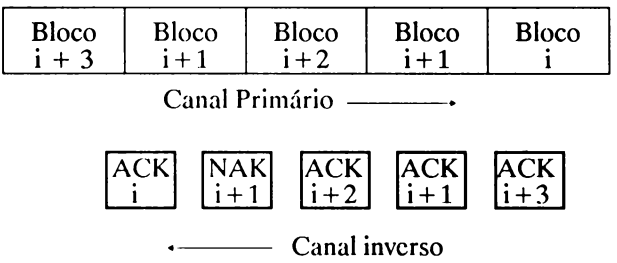


Figura 1.7 - Go back n ARQ. No procedimento de controle de erro 'go back n ARQ', o transmissor envia, continuamente, mensagens até que o receptor detecte um erro. O receptor então transmite uma aceitação negativa (NAK) no canal inverso e o transmissor retransmite o bloco recebido com erro. Algumas versões desta técnica requerem que os blocos enviados, antes que a indicação de erro fosse encontrada, sejam retransmitidos, juntamente com os blocos recebidos com erro.

Modelo half-duplex completo

Quando um bloco de mensagem é transmitido na estrutura de controle BISYNC, vários caracteres de controle estão contidos naquele bloco, juntamente com o texto da mensagem. Se a variável C é atribuída para representar o número de caracteres de controle por bloco e a variável D é usada para representar o número de caracteres de dados, então o comprimento total do bloco é C + D. Se a taxa de transferência de dados, expressa em bps, for denominada como T_R e o número de bits por caractere for denominado como B_c, então o tempo de transmissão para um caractere é igual a B_C/T_R, que pode ser denominado como Tc. Já que os caracteres D + C estão contidos em um bloco de mensagem, o tempo necessário para transmitir o bloco será T_c* (D + C). Uma vez que o bloco é recebido, este deve ser aceito. Para assim fazê-lo, a estação receptora é solicitada para, primeiramente, computar o caractere de verificação de bloco (BCC) e compará-lo com o caractere BCC transmitido, que se encontra agregado no fim do bloco transmitido. Embora o caractere BCC seja computado à medida em que os dados são recebidos, a comparação é executada depois que o bloco for inteiramente recebido e somente aí uma aceitação de recebimento pode ser transmitida. O tempo para verificar os caracteres BCC computados e transmitidos e formar e transmitir a aceitação de recebimento, é conhecido como tempo de processamento e de aceitação de recebimento (T_{PA}).

Quando a transmissão é half-duplex, o tempo de retorno (mudança de sentido) da linha (T_L) exigido para reverter o sentido de transmissão da linha deve ser acrescentado. Normalmente, este tempo inclui o tempo de retardo do modem para se obter RTS/CTS (solicitação-para-envio/liberado-para-envio), bem como cada um dos tempos de retardo interno dos modems. Para a aceitação de recebimento alcançar seu destino, esta deve ser propagada através do circuito de tempo de retardo de propagação, denominado como T_p. Se a mensagem de aceitação de recebimento contiver A caracteres então, quando transmitida no canal primário, A*B_C/T_R segundos são necessários para enviar a aceitação de recebimento.

Uma vez que a estação de transmissão de origem recebe a aceitação de recebimento, ela deve determinar se é exigida para retransmitir o bloco de mensagem enviado previamente. Este tempo é semelhante ao tempo de aceitação e de processamento discutido anteriormente. Para transmitir um novo bloco de mensagem ou repetir o bloco de mensagem enviado anteriormente, a linha deve mudar de sentido novamente e o bloco de mensagem necessitará tempo para propagar através da linha e chegar na estação recep-

tora. Assim, o tempo total para transmitir um bloco de mensagem e receber a aceitação de recebimento, conhecido como T_B, é:

$$T_B = T_C * (D + C) + 2* (T_{PA} + T_L + T_p) + (A*B_C / T_R). \quad (1.1)$$

Já que eficiência é: taxa de transferência de dados dividida pela taxa teórica de transferência de dados, a eficiência da estrutura de controle de transmissão (Etcs), torna-se:

$$E_{TCS} = \frac{B_C * D * (1 - P)}{T_R * T_B}. \quad (1.2)$$

Na equação (1.2), o numerador representa o número real de bits de dados que são recebidos corretamente. Aqui, B_C é o número de bits por caractere e D é o número de caracteres de dados no bloco. Assim, B_C*D representa o número de bits contidos nos caracteres de dados que formam o bloco. Já que há uma probabilidade, P, que um ou mais bits no bloco estejam com erro, multiplicando B_C*D(1 - P) resulta no número médio de bits por bloco que não contém erro.

Voltando nossa atenção ao denominador, T_R representa a taxa de transferência de dados, enquanto T_B representa o tempo total para transmitir um bloco de mensagem e receber uma aceitação de recebimento. Isto resulta no denominador contendo o número teórico de bits que poderia ser transferido e recebido. Ao dividir o número de bits recebidos corretamente (numerador) pelo número teórico de bits que poderiam ser transferidos (denominador), temos, como resultado, a eficiência da estrutura de controle da transmissão.

Embora o que foi exposto seja uma medida da eficiência da estrutura de controle da transmissão, esta não considera a eficiência do código de dados, ou seja, a relação entre os bits de informação e os bits totais por caractere. Quando a eficiência do código de dados é incluída, obtemos uma medida da eficiência da transferência de informação. Podemos chamar esta relação de razão de transferência de informação (ITR), que nos proporcionará uma medida da eficiência do protocolo com referência à transferência de informações do protocolo. Isto resulta em:

$$ITR = \frac{B_{IC} * E_{TCS}}{B_C} \quad (1.3)$$

onde:

ITR	=	Razão de transferência de informação
B_{IC}	=	Bits de informação por caractere
B_C	=	Bits totais por caractere
D	=	Caracteres de dados por bloco de mensagem
A	=	Caracteres na mensagem de aceitação de recebimento
C	=	Caracteres de controle por bloco de mensagem
T_R	=	Taxa de transferência de dados (bps)
T_C	=	Tempo de transmissão por caractere (Bc/Tr)
T_{PA}	=	Tempo de processamento e de aceitação
TI	=	Tempo de mudança de sentido da linha
T_P	=	Tempo de retardo de propagação
P	=	Probabilidade de haver um ou mais erros no bloco.

A partir do que foi visto anteriormente, a razão de transferência de informação nos fornece uma medida da eficiência da estrutura de controle da transmissão, sem considerar o efeito da compressão. Quando se considera a compressão, obtemos um novo termo que chamaremos de razão efetiva de transferência de informações (EITR).

Quando os dados são comprimidos, o fluxo de dados originais será reduzido de tamanho antes da transmissão. A redução real é dependente dos algoritmos de compressão empregados, bem como da composição dos dados. Em geral, podemos supor que a razão de compressão considera o número de caracteres no fluxo de dados comprimidos, para incluir caracteres especiais de controle necessários para indicar um ou mais algoritmos de compressão. Esta suposição razoável simplifica o efeito de se considerar a compressão de dados, ao examinar um protocolo particular. Como exemplo, considere um bloco de dados com 160 caracteres comprimidos em 78 caracteres de dados, mais dois caracteres indicadores de compressão. Aqui, a razão de compressão seria de 160/(78 + 2) ou 2. O efeito sobre a equação, previamente desenvolvida para computar a razão de transferência de informação, seria a de substituir o D do numerador pelo comprimento da cadeia não comprimida de 160 caracteres, enquanto D no denominador, seria os 78 caracteres reais de dados comprimidos, mais os dois caracteres especiais necessários para indicar a compressão de dados, resultando em um total de 80 caracteres. Se o número total de caracteres de controle, que envolve o bloco de dados, for relativamente pequeno, a razão efetiva de transferência de informação pode ser aproximada à multiplicação da taxa de transferência de informação pela taxa de compressão.

Assumiremos que nossa taxa de transmissão de dados seja de 4800 bps e que transmitiremos as informações, usando a estrutura de controle de transmissão BISYNC, empregando o procedimento de controle de erro 'stop and wait ARQ. Além disso, vamos assumir os seguintes parâmetros:

A	= 4 caracteres por aceitação de recebimento
B_{IC}	= 8 bits por caractere
B_C	= 8 bits por caractere
D	= 80 caracteres de dados por bloco
C	= 10 caracteres de controle por bloco
T_R	= 4800 bps
T_C	= 8/4800 = 0,001 66 segundos (s) por caractere
T_{pa}	= 20 ms = 0,02 s
T_I	= 100 ms = 0,10 s
T_P	= 30 ms = 0,03 s
P	= 0,01.

Assim:

$$ITR = \frac{8*80*(1 - 0.01)}{4800*[0.001\ 66*(80+10) + 2*(0.02+0.03+0.1) + 4*8/4800]}$$

$$= 0.2894$$

Já que a taxa de transferência de informações em bits (TRIB) é igual ao produto da razão de transferência de dados e da razão de transferência de informações, obtemos:

$$TRIB = ITR*T_R = 0.2894*4800 = 1389\ bps$$

Para o exemplo anterior, aproximadamente 29 por cento da razão de transferência de dados é efetivamente usada.

Vamos, agora, examinar o efeito de dobrar o tamanho do texto para 160 caracteres, enquanto os parâmetros anteriores exceto P, continuam como antes. Já que o tamanho do bloco dobrou, P aproximadamente dobra, resultando no ITR:

$$ITR = \frac{8*160*(1 - 0.02)}{4800*[0.001\ 66*(1600+10) + 2*(0.02+0.03+0.1) + 4*8/4800]}$$

$$= 0.4438$$

Com um ITR de 0,4438 o TRIB fica agora:

$$TRIB = ITR * T_R = 0.4438 * 4800 = 2130 \text{ bps.}$$

Aqui, ao dobrar o tamanho do bloco, a porcentagem da razão de transferência de dados efetivamente usada, aumenta para 44,38 por cento.

Efeito de Compressão

Vamos supor que um ou mais algoritmos de compressão de dados sejam empregados, resultando na razão de compressão de 2. Qual o efeito que isto teria sobre a razão efetiva de transferência de informação?

A razão efetiva de transferência de informação (EITR) pode ser obtida ao se modificar a equação (1.2), como a seguir:

$$EITR = \frac{B_{IC} * D_1 * (1-P)}{T_R * [T_C * (D_2+C) + 2*(T_{PA}+T_L+T_P) + (A*B_C/T_R)]} \qquad (1.4)$$

onde:

D_1 = tamanho do bloco de dados originais, em caracteres antes da compressão

D_2 = tamanho do bloco de dados comprimidos, em caracteres incluindo os caracteres especiais de indicação de compressão

Se, na média, 160 caracteres de dados são transmitidos em um formato comprimido de 80 caracteres, obtemos:

$$EITR = \frac{8*160*(1-0.01)}{4800*[0.001\ 66*(80+10)+2*(0.02+0.03+0.1)+(4*8/4800)]}$$

$$= 0.5788$$

Como discutido anteriormente, a razão efetiva de transferência de informação pode ser aproximada a:

$$EITR \cong ITR * CR \tag{1.5}$$

Ao substituir, obtemos:

$$EITR \cong 0.2861 * 2 \cong 0.5722$$

A partir do que foi visto acima, observaremos que a diferença entre o EITR computado (0,5788) e o EITR aproximado (0,5722) é basicamente insignificante. Assim, se você conhece o ITR, pode simplesmente multiplicá-lo pela razão de compressão para obter uma aproximação aceitável à razão efetiva de transferência de informação.

Já que a razão de transferência de informações em bits (TRIB) é produto da razão efetiva de transferência de informação e da taxa operacional de dados, obtemos:

$$TRIB = 0.5788 * 4800 = 2778\ bps.$$

Na tabela 1.1, o leitor encontrará uma comparação das variações no ITR, EITR e TRIB, quando, para dois tamanhos diferentes de bloco, os dados comprimidos e não comprimidos são transmitidos.

Tabela 1.1 Comparação do Efeito da compressão

	Dados não Comprimidos		Dados Comprimidos	
Tamanho do bloco (caracteres)	80	160	80	160
ITR (sem dimensão)	0,2894	0,4438	N/A	N/A
EITR (sem dimensão)	N/A	N/A	0,5788	0,8678
TRIB (bps)	1389	2130	2778	4165

Com base na tabela 1.1, está claro que os dois métodos podem ser empregados para aumentar a eficiência da transmissão. Primeiro, você pode alterar o protocolo ou a seqüência de controle da transmissão, variando o tamanho dos blocos de dados transmitidos. Alternativamente, você pode comprimir os dados antes de transmitir o bloco de informações. Os dois métodos podem resultar em mais informações por unidade de tempo, sendo passadas pela linha de transmissão.

Para facilitar a computação do ITR com base nos tamanhos variados de blocos, um programa em linguagem BASIC foi desenvolvido. Este programa, listado na tabela 1.2, computa o ITR para tamanhos de blocos que variam de 40 a 4000 caracteres, em incrementos de 40 caracteres. Ao examinar a listagem do programa, na tabela 1.2, observe que a probabilidade de um bloco estar com erro é de 0,005. Isto corresponde à metade de 0,01, já que agora, estamos variando os tamanhos de blocos a cada 40 caracteres, em vez de 80 caracteres. Observe, também, que como cada bloco é incrementado de 40 caracteres, a probabilidade de um bloco conter um ou mais bits com erros aumentou 0,005, já que a probabilidade de um bloco apresentar um ou mais bits com erro é proporcional ao seu tamanho.

Na tabela 1.3, o leitor encontrará uma tabulação da execução do programa que calculou o ITR, à medida que o tamanho do bloco variou de 40 a 4000 caracteres em incrementos de 40. Ao examinar esta tabela, você deve observar que o ITR máximo de 0,7358 é obtido quando o tamanho do bloco chegar a 1080

Tabela 1.2 - A listagem do programa BASIC para computar o ITR em função do tamanho do bloco

```
A = 4
BIC = 8
BC = 8
D = 80
C = 10
TR = 4800
TC = 8 / 4800
TPA = 0.02
TL = 0.1
TP = 0.03
P = 0.005
FOR BLOCK = 40 TO 4000 STEP 40
NUMERATOR = BIC * BLOCK * (1 - P)
DENOM = (TR * (O,00166 * (BLOCK + 10) + 2 * (TPA + TL + TP) + A * BIC / TR))
ITR = NUMERATOR / DENOM
PRINT USING ' #.#### ####';ITR;BLOCK
P = P + 0.005
NEXT BLOCK
```

Tabela 1.3 - A razão de transferência de informação e o tamanho do bloco. Probabilidade de erro no bloco = 0,01 por 80 caracteres

ITR	Tamanho do Bloco	ITR	Tamanho do Bloco
0,1702	40	0,6828	2040
0,2894	80	0,6794	2080
0,3771	120	0,6759	2120
0,44381	60	0,6723	2160
0,4960	200	0,6687	2200
0,5376	240	0,6651	2240
0,5714	280	0,6614	2280
0,5992	320	0,6576	2320
0,6222	360	0,6539	2360
0,6415	400	0,6501	2400
0,6577	440	0,6462	2440
0,6714	480	0,6423	2480
0,6830	520	0,6384	2520
0,6928	560	0,6345	2560
0,7011	600	0,6305	2600
0,7082	640	0,6265	2640
0,7142	680	0,6224	2680
0,7191	720	0,6184	2720
0,7233	760	0,6143	2760
0,7267	800	0,6102	2800
0,7295	840	0,6060	2840
0,7317	880	0,6019	2880
0,7333	920	0,5977	2920
0,7345	960	0,5935	2960
		0,5893	3000
0,7353	1000	0,5850	3040
0,7357	1040	0,5808	3080
0,7358	1080	0,5765	3120
0,7356	1120	0,5722	3160
0,7350	1160	0,5679	3200
0,7343	1200	0,5635	3240
0,7332	1240	0,5592	3280
0,7320	1280	0,5548	3320
0,7306	1320	0,5504	3360
0,7290	1360	0,5460	3400
0,7272	1400	0,5416	3440

Tabela 1.3 (continuação)

ITR	Tamanho do Bloco	ITR	Tamanho do Bloco
0,7252	1440	0,5372	3480
0,7231	1480	0,5328	3520
0,7209	1520	0,5283	3560
0,7185	1560	0,5239	3600
0,7161	1600	0,5194	3640
0,7135	1640	0,5149	3680
0,7108	1680	0,5104	3720
0,7080	1720		
0,7051	1760	0,5059	3760
0,7021	1800	0,5014	3800
0,6991	1840	0,4969	3840
0,6960	1880	0,4924	3880
0,6928	1920	0,4878	3920
0,6895	1960	0,4833	3960
0,6862	2000	0,4787	4000

caracteres. Isto indica que, à medida que o tamanho do bloco aumenta com uma taxa de erro constante, um certo ponto é alcançado, onde o tempo para retransmitir um bloco grande invalida, ocasionalmente, o aumento do tamanho do bloco. Para os parâmetros considerados, o tamanho ideal de bloco é de 1080 caracteres. Somente para a situação ideal, onde P = 0, um contínuo aumento no tamanho do bloco geraria eficiências adicionais.

Na figura 1.8, o ITR é plotado em função do tamanho do bloco para a condição livre de erro e para a condição de erro com probabilidade 0,01. A condição de erro com probabilidade 0,01 por bloco de 80 caracteres, foi mantida constante ao se incrementar a taxa de erro na proporção do aumento no tamanho do bloco. Uma vez que uma linha livre de erro não é algo que um engenheiro de transmissão possa sensatamente esperar, existirá um tamanho máximo de bloco, o qual diminuirá a eficiência da linha. Neste ponto, somente a compressão de dados resultará em eficiências adicionais na transmissão. Além disso, do ponto de vista físico, a área de buffer de alguns dispositivos pode impedir que os tamanhos de blocos excedam um determinado número de caracteres. Mais uma vez, a compressão de dados pode se tornar um mecanismo eficaz

para aumentar a eficiência da transmissão, enquanto se mantêm as exigências de buffer de dados dentro de um nível aceitável.

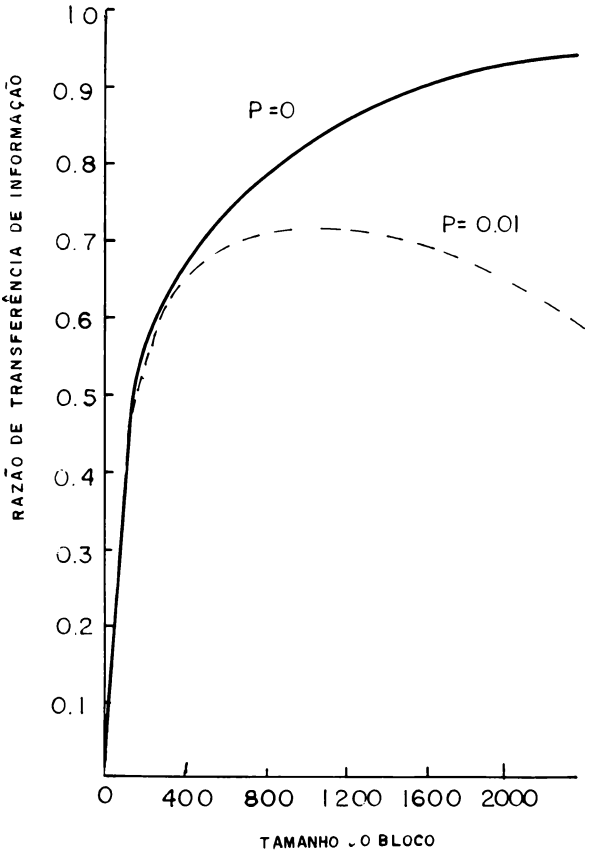


Figura 1.8 - ITR e taxa de erro.

Modelo de canal de retorno

Considere um procedimento de controle de erro 'stop and wait ARQ', onde um canal de retorno fica disponível para a transmissão de aceitação de recebimento. O uso deste canal de retorno elimina a necessidade de mudança de sentido da linha; contudo, a transmissão ainda fica sendo half-duplex, já que a aceitação de recebimento somente é transmitida depois que cada bloco de mensagem recebido for processado.

Quando o bloco de mensagem é enviado à estação receptora, são encontrados o retardo da propagação e o retardo do processamento. Quando a aceitação dos recebimentos retorna, ocorre mais um retardo de propagação e outro retardo de processamento. Além desses retardos, você também deve considerar o tempo necessário para transmitir a mensagem de aceitação do recebimento. Se A indica o comprimento em caracteres da mensagem de aceitação de recebimento e T_S é a taxa de dados do canal inverso em bps, então o tempo de transmissão para a aceitação de recebimento torna-se $(A*B_C)/T_S$.

O tempo total de retardo devido à propagação e ao processamento, bem como ao tempo de transmissão da aceitação de recebimento torna-se:

$$2*(T_{PA} + T_p) + \frac{A*B_C}{T_S}$$

Assim, a razão de transferência de informações torna-se:

$$ITR = \frac{B_{IC} * D_1(1-p)}{T_R * [T_C * (D_2+C)+2*(T_{PA}+T_p)+A * B_C / T_s]} \quad (1.6)$$

Vamos examinar o efeito deste procedimento modificado de transmissão do exemplo anterior, onde os dados foram empacotados com 80 caracteres por bloco. Vamos assumir que um canal inverso de 75 bps esteja disponível e nossa mensagem de aceitação de recebimento seja composta de quatro caracteres de oito bits.

Assim:

$$ITR = \frac{8*80*(1 - 0.01)}{4800*[0.001\ 66*(80+10)+2*(0.02+0.03)+4*8/75]} = 0.1953$$

Observe que o ITR realmente diminuiu. Isto foi provocado pela lentidão do canal inverso, onde ele levou 0,4266 (4*8/75) segundos para transmitir uma aceitação de recebimento. Em comparação, as mudanças de sentido das duas linhas que foram eliminadas, exigiam somente 0,2 s quando a aceitação de recebimento era enviada a 4800 bps no canal primário. Este procedimento modificado é

basicamente eficaz, quando o tempo de retorno da linha excede o tempo de transmissão da aceitação de recebimento no canal de retorno. Esta situação normalmente ocorre quando a taxa primária de transferência de dados é de 2400 bps ou menos. Se os dados são comprimidos antes da transmissão e a razão de compressão de 2 resulta em 160 caracteres de dados transmitidos como um bloco de 80 caracteres comprimidos, o EITR pode ser computado como a seguir:

$$EITR = \frac{8*160*(1 - 0.01)}{4800*[0.001\ 66*(80+10)+2*(0.02+0.03)+4*8/75]} = 0.39$$

Ao comparar o efeito de compressão, observe que a razão de transferência de informação em bits (TRIB) aumenta de 0,1953*4800 ou 937 para 0,39*4800 ou 1872 bps. Assim, a razão de compressão de 2 pode ser esperada para dobrar aproximadamente tudo.

O modelo full duplex

Uma eficiência completa muito melhor com o procedimento de controle de erro 'stop and wait ARQ' pode ser obtida quando o modo full duplex de transmissão é empregado. Embora isto requeira o uso de uma linha de quatro fios ou modems que dividam um circuito de dois fios em bandas de freqüências independentes, os modems e a linha não têm de ser invertidos. Isto permite que uma aceitação de recebimento seja transmitida na mesma taxa de dados que o bloco de mensagem, mas na direção inversa, sem a mudança da linha. Assim, a razão de transferência de informação fica:

$$ITR = \frac{B_{IC} * D_1\ 8\ (1 - P)}{T_R * [T_C * (D_2+C)+2*(T_{PA} + T_p)]} \qquad (1.7)$$

Novamente, voltando ao exemplo original do bloco de 80 caracteres, obtemos:

$$ITR = \frac{8*80*(1 - 0.01)}{4800*[0.001\ 66*(80+10)+2*(0.02+0.03)]} = 0.5293$$

Quando a compressão de dados resulta em uma razão de compressão de 2, obtemos:

$$EITR = \frac{8*160*(1 - 0.01)}{4800*[0.001\ 66*(80+10)+2*(0.02+0.03)]} = 1.06$$

Com um EITR maior que a unidade, isto significa que os bits das informações por unidade de tempo (no formato comprimido) excedem a taxa de transmissão de dados do equipamento conectado à linha. Isto ilustra o valor da compressão de dados, permitindo que você obtenha taxas de transferência de dados altamente eficazes, sem necessitar de instalações de comunicações adicionais.

Ocorre uma segunda variação do modelo full duplex quando o procedimento de controle de erro 'go back n ARQ' é empregado. Nesta situação, só é retransmitido o bloco recebido com erro. Aqui, a razão de transferência de informação torna-se:

$$ITR = \frac{B_{IC} * D_1 * (1 - P)}{T_R * [T_C * (D_2 + C)]} \qquad (1.8)$$

Novamente, substituindo os valores do exemplo original, obtemos:

$$ITR = \frac{8*80*(1 - 0.01)}{4800*[0.001\ 66*(80+10)]} = 0.8835$$

Esta é, obviamente, a técnica mais eficiente, já que a mudança de sentido da linha é eliminada e o tempo de processamento e de aceitação de recebimento (Tpa) e o tempo de retardo de propagação (Tp) em cada sentido são anulados, devido à transmissão simultânea do bloco de mensagem e da resposta de aceitação de recebimento. Se considerarmos o efeito da razão de compressão de 2, a razão efetiva de transferência de informações pode ser computada como a seguir:

$$EITR = \frac{8*160*(1 - 0.01)}{4800*[0.001\ 66*(80+10)]} = 1.767$$

Como uma alternativa para o cálculo anterior, você poderia estimar o EITR ao multiplicar o ITR pela razão de compressão.

Procedendo desta forma nesta situação, o resultado seria exatamente o mesmo. Para o exemplo anterior, o TRIB torna-se 1.767*4800 bps ou 8482 bps.

Aqui, a estrutura de compressão e de protocolo permite uma transferência efetiva de informações a 8482 bps em um via de dados de 4800 bps. De fato, a seleção de um protocolo apropriado em conjunto com algoritmos eficientes de compressão de dados, pode resultar em uma transferência de dados muito eficiente. Isto resultará em transferências de dados, normalmente associadas às instalações em banda larga, através das instalações convencionais de transmissão de voz.

CAP. 02

CÓDIGOS DE DADOS E CARACTERES INDICANDO COMPRESSÃO

Antes de discorrermos sobre as técnicas de compressão de dados, é importante estar familiarizado com os diversos códigos de dados e a representação binária dos caracteres destes códigos. Esta familiaridade é necessária, uma vez que as técnicas de compressão orientadas a caractere estão fundamentadas no emprego de um ou mais caracteres em um determinado conjunto de caracteres. Estes caracteres são usados para indicar a ocorrência de um algoritmo de compressão, sendo que a compressão e descompressão não podem ocorrer sem o uso destes caracteres.

Neste capítulo, examinaremos, primeiro, a composição de quatro códigos de dados populares. Uma vez feita esta análise, utilizaremos o conhecimento anterior para discutir os diversos métodos que podem ser usados para desenvolver caracteres de indicação de compressão, além das vantagens e desvantagens associadas ao uso de cada método.

2.1 CÓDIGOS DE DADOS

Código BAUDOT

O código Baudot de cinco níveis foi projetado para permitir que as teleimpressoras operassem com maior rapidez e precisão, do que os relés usados para transmitir informações via telégrafo. O código Baudot de cinco níveis permite 32 combinações únicas de bits, ou seja 32 caracteres, observe que o número de diferentes

caracteres que podem ser gerados a partir de um código com dois estados diferentes (binários) é 2^m, onde m é o número de posições no código. Embora 32 caracteres possam ser normalmente representados com este código, a necessidade de transmitir dígitos, letras do alfabeto e sinais de pontuação torna necessário projetar um mecanismo para estender a capacidade do código para incluir representações adicionais de caracteres.

O mecanismo de extensão foi realizado, usando-se dois caracteres 'shift' - 'shift' letras e 'shift' figuras. A transmissão de um caractere shift informa o receptor que os caracteres seguintes ao caractere shift devem ser interpretados como caracteres de um símbolo e conjunto numérico ou de um conjunto alfabético de caracteres. O código Baudot de cinco dígitos está ilustrado na Figura 2.1 para um determinado arranjo de palheta do terminal. Uma transmissão de 1s em todas as posições de bit de 1 a 5 indica um 'shift' letras e os caracteres seguintes à transmissão daquele caractere são interpretados como letras. De forma análoga, a transmissão de 1s nas posições de bit 1, 2, 4 e 5 indicaria um 'shift' figuras e os caracteres seguintes seriam interpretados como sendo numéricos ou símbolos baseados em sua estrutura de código.

Códigos BCD e EBCDIC

O desenvolvimento de sistemas de computação exigiu a implementação de sistemas de codificação para converter caracteres alfanuméricos em notação binária. Um dos códigos mais antigos usados para converter dados em uma forma aceitável para o computador foi o sistema decimal codificado em binário (BCD). Esta técnica de codificação permite que as informações numéricas sejam representadas por 4 bits binários e que o conjunto de caracteres alfanuméricos seja representado através do uso de 6 bits de informações. Este código é apresentado na Figura 2.2. Uma vantagem deste código é que 2 dígitos decimais podem ser armazenados em uma palavra de computador de 8 bits e manipulados com instruções apropriadas do computador. Embora somente 36 caracteres sejam mostrados, o código BCD é capaz de conter um conjunto de 2^6 ou 64 caracteres diferentes.

Caracteres		Seleção de bits				
Letras	Figuras	1	2	3	4	5
	-	1				
A	?	1				1
B	:	.				1
C	$	1	1	1		
D	3	1			1	
E	!	1	1		1	
F	¢			1		1
G					1	1
H	8			1	1	
I	'	1	1	1		
J	(	1				
K	)		1	1	1	1
L	.		1		1	1
M	,		1	1		
N	9		1	1	1	1
O	0				1	1
P	1	1		1	1	1
Q	4			1		
R		1	1			
S	5		1	1	1	
T	7	1	1			
U	;			1	1	1
V	2	1		1		1
W	/	1	1		1	1
X	6	1	1	1		1
Z	"	1	1	1	1	1
Funções						
Retorno de carro						
Alimentação de linha		1		1		
Espaço						
Letras shift	<	1	1	1	1	1
Figuras shift	=	1	1	1	1	1

Figura 2.1 - Representação de dados do código Baudot.

Além de transmitir letras, numéricos e sinais de pontuação, um número considerável de caracteres de controle pode ser exigido para promover o controle de linha. Estes caracteres de controle podem ser usados para ligar e desligar os dispositivos que são conectados à linha de comunicação, controlar a transmissão real de dados, manipular formatos de mensagens e executar funções adicionais. Assim, um conjunto estendido de caracteres é geralmente exigido para comunicação de dados. Um conjunto de carac-

Posição de bit						
b_6	b_5	b_4	b_3	b_2	b_1	Caractere
0	0	0	0	0	1	A
0	0	0	0	1	0	B
0	0	0	0	1	1	C
0	0	0	1	0	0	D
0	0	0	1	0	1	E
0	0	0	1	1	0	F
0	0	0	1	1	1	G
0	0	1	0	0	0	H
0	0	1	0	0	1	I
0	1	0	0	0	1	J
0	1	0	0	1	0	K
0	1	0	0	1	1	L
0	1	0	1	0	0	M
0	1	0	1	0	1	N
0	1	0	1	1	0	O
0	1	0	1	1	1	P
0	1	1	0	0	0	Q
0	1	1	0	0	1	R
1	0	0	0	1	0	S
1	0	0	0	1	1	T
1	0	0	1	0	0	U
1	0	0	1	0	1	V
1	0	0	1	1	0	W
1	0	0	1	1	1	X
1	0	1	0	0	0	Y
1	0	1	0	0	1	Z
1	1	0	0	0	0	0
1	1	0	0	0	1	1
1	1	0	0	1	0	2
1	1	0	0	1	1	3
1	1	0	1	0	0	4
1	1	0	1	0	1	5
1	1	0	1	1	0	6
1	1	0	1	1	1	7
1	1	1	0	0	0	8
1	1	1	0	0	1	9

Figura 2.2 - Representação de dados do decimal codificado em binário (BCD).

teres deste tipo é o código estendido de caracteres decimais codificados em binário para intercâmbio de informações (EBCDIC), apresentado na Figura 2.3. Este código é uma extensão do sistema decimal codificado em binário e utiliza 8 bits para a representação de caracteres. Este código permite 2^8 ou 256 caracteres únicos para serem representados embora, atualmente, se utilize um número menor de caracteres. Este código é usado, essencialmente, para a transmissão por computadores orientados a byte. O uso do EBC-

DIC por computadores pode aliviar a necessidade de se ter o computador executando a conversão de código se os terminais conectados transmitirem dados naquele código. Caso contrário, o computador executará a conversão de código antes da operação dos dados. O conjunto de caracteres EBCDIC está ilustrado na Figura 2.3. Observe que os caracteres indefinidos neste código, ou em qualquer código para aquele propósito, podem ser usados para significar operações especiais predefinidas ou representações de dados. Assim, uma ou mais técnicas de compressão de dados poderiam ser indicadas por uma seqüência de 8 bits que represente um caractere, definido ou indefinido, de um conjunto de caracteres EBCDIC. Normalmente, é melhor empregar um caractere indefinido, uma vez que caracteres delineados, além de serem numéricos, gráficos ou alfabéticos, incluem caracteres de controle que orientam a operação do terminal e a seqüência de comunicações.

Quando os dados são comprimidos de acordo com um certo algoritmo, uma seqüência de 8 bits é tipicamente inserida no fluxo de dados para sinalizar que a compressão ocorreu e o que tipo de algoritmo foi empregado. Neste livro, referimo-nos ao uso destes caracteres como caracteres especiais indicadores de compressão ou caracteres especiais para brevidade. Quando os dados comprimidos são recebidos, a mesma seqüência de 8 bits indicaria a presença de dados comprimidos. Além disso, informaria a estação de recepção sobre o algoritmo de compressão de dados empregado, desta forma capacitaria o receptor para descomprimir o fluxo de dados comprimidos.

	00			01					10				11			
	00	01	10	11	00	01	10	11	00	01	10	11	00	01	10	11
0000	NUL	DEL	DS		SP	&										0
0001	SOH	DCI	SOS				/		a	j			A	J		1
0010	STX	DC2	FS	SYN					b	k	s		B	K	S	2
0011	ETX	TM							c	l	°t		C	L	T	3
0010	PF	RES	BYP	PN					d	m	u		D	M	U	4
0101	HT	NL	LF	RS					e	n	v		E	N	V	5
0110	LC	BS	ETB	UC					f	o	w		F	O	W	6
0111	DEL	IL	ESC	EOT					g	p	x		G	P	X	7
1000		CAN					h	q	y		H	Q	Y	8		
1001		EM						i	r	z		I	R	Z	9	
1010	SMM	CC	SM		c	!		:								
1001	VT	CU1	CU2	CU3	.	$	,	•								
1100	FF	IFS		DC4	<	*	%	®								
1101	CR	IGS	ENAQ	NAK	(	)	–	'								
1110	SP	IPS	ACK		+	;	>	=								
1111	SI	IUS	BEL	SUB	|		?	"								

ACK	Acknowledge	EOT	End of transmission	PN	Punch on
BEL	Bell	ESC	Escape	RES	Restore
BS	Backspace	ETB	End of transmission block	RS	Reader stop
BYP	By pass	ETX	End of text	SI	Shift in
CAN	Cancel	FF	Form feed	SM	Set mode
CC	Cursor control	FS	Field separator	SMM	Start of manual message
CR	Carriage return	HT	Horizontal tab	SO	Shift out
CU1	Customer use 1	IFS	Interchange file separator	SOH	Start of heading
CU2	Customer use 2	IGS	Interchange group separator	SOS	Start of significance
CU3	Customer use 3	IL	Idle	SP	Space
DC1	Device control 1	IRS	Interchange record separator	STX	Start of text
DC2	Device control 2	IUS	Interchange unit separator	SUB	Substitute
DC4	Device control 4	LC	Lower case	SYN	Synchronous idle
DEL	Delete	LF	Line feed	TM	Tape mark
DLE	Data link escape	NAK	Negative acknowledge	UC	Upper case
DS	Digit select	NL	New line	VT	Vertical tab
EDM	End of medium	NUL	Null		
ENQ	Enquiry	PF	Punch off		

Special graphic characters

¢	Cent sign	–	Minus sign, hyphen
.	Period, decimal point	/	Slash
<	Less-than sign	,	Comma
(	Left parenthesis	%	Per-cent
+	Plus sign	—	Underscore
|	Logical OR	>	Greater-than sign
&	Ampersand	?	Question mark
!	Exclamation mark	:	Colon
$	Dollar sign	#	Number sign
*	Asterisk	@	At sign
)	Right parenthesis	'	Prime, apostrophe
;	Semicolon	=	Equal sign
¬	Logical NOT	"	Quotation mark

Figura 2.3 - Conjunto de caracteres EBCDIC.

Código ASCII

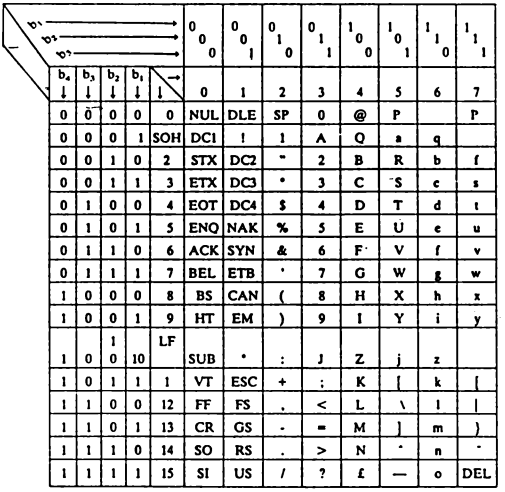
Devido à proliferação do número de códigos de transmissão de dados, surgiram tentativas de padronizar estes códigos. Esta busca pela padronização resultou no desenvolvimento do Código Padrão para Intercâmbio de Informação (ASCII). Este código de sete níveis encontra-se ilustrado na Figura 2.4 e baseia-se no código de 7 bits desenvolvido pela Organização Internacional para Padronização (ISO).

Os caracteres ASCII estão codificados em 7 bits, enquanto o oitavo bit fica disponível para uso como bit de paridade. O ASCII de oito níveis é comumente usado para transmissão por dispositivos de teletipo assíncronos interativos, onde o bit de paridade pode representar paridade par ou ímpar.

Outra versão popular do ASCII é representada pelo conjunto de caracteres usados pelo IBM PC e computadores pessoais compatíveis. Conhecido como ASCII estendido, este conjunto de caracteres usa 8 bits, em vez de 7, usados no ASCII convencional. Os primeiros 128 caracteres ASCII, cujos códigos são de 0 a 127 são exatamente os mesmos que aqueles listados na Figura 2.4. Os outros 128 caracteres são formados pelo estabelecimento do oitavo bit e define gráficos de bloco e outros caracteres especiais.

2.2 SELECIONANDO CARACTERES DE INDICAÇÃO DE COMPRESSÃO

A chave para a compressão orientada à caractere é a seleção e utilização de um ou mais caracteres de indicação de compressão. O caractere de indicação de compressão servirá como um flag, informando ao dispositivo de recepção que ocorreu compressão e que o tipo de compressão foi executado. O dispositivo de recepção pode, então, aplicar um algoritmo de descompressão apropriado para restaurar os dados em sua forma original.



b_4	b_3	b_2	b_1	→	0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
↓	↓	↓	↓	↓	0	1	2	3	4	5	6	7
0	0	0	0	0	NUL	DLE	SP	0	@	P		P
0	0	0	1	SOH	DC1	!	1	A	Q	a	q	
0	0	1	0	2	STX	DC2	"	2	B	R	b	f
0	0	1	1	3	ETX	DC3	•	3	C	S	c	s
0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
1	0	0	0	8	BS	CAN	(	8	H	X	h	x
1	0	0	1	9	HT	EM	)	9	I	Y	i	y
1	0	1 0	10	LF	SUB	•	:	J	Z	j	z	
1	0	1	1	1	VT	ESC	+	;	K	[	k	[
1	1	0	0	12	FF	FS	,	<	L	\	l	|
1	1	0	1	13	CR	GS	-	=	M	]	m	)
1	1	1	0	14	SO	RS	.	>	N	^	n	~
1	1	1	1	15	SI	US	/	?	£	—	o	DEL

Control characters

NUL	Null/idle
SOH	Start of heading (CC)
STX	Start of text (CC)
ETX	End of text (CC)
EOT	End of transmission (CC)
ENQ	Enquiry (CC)
ACK	Acknowledge (CC)
BEL	Audible or attention signal
BS	Backspace (FE)
HT	Horizontal tabulation (punch card skip) (FE)
LF	Line feed
VT	Vertical tabulation (FE)
FF	Form feed (FE)
CR	Carriage return (FE)
SO	Shift out
SI	Shift in
DLE	Data link escape (CC)
DC1	
DC2	Device controls
DC3	
DC4	Device control (stop)
NAK	Negative acknowledge (CC)
SYN	Synchronous idle (CC)
ETB	End of transmission block (CC)
CAN	Cancel
EM	End of medium
ESC	Escape
FS	File separator (IS)
GS	Group separator (IS)
RS	Record separator (IS)
US	Unit separator (IS)
DEL	Delete

Note: (CC) Communication control
(FE) Formal effector
(IS) Information separator

Special graphic characters

SP	Space
!	Exclamation mark
|	Logical OR
"	Quotation marks
#	Number signs
$	Dollar sign
%	Per cent
&	Ampersand
'	Semicolon
<	Less than
=	Equals
>	Greater than
?	Question mark
@	Commercial at
[	Opening bracket
\	Reverse slant
(	Opening parenthesis
)	Closing parenthesis
*	Asterisk
+	Plus
,	Comma
-	Hyphen
.	Period
/	Slant
:	Colon
]	Closing bracket
—	Underline
^	Grave accent
{	Opening brace
|	Vertical line
}	Closing brace
~	Tilde

Figura 2.4 - O código de dados ASCII.

Para ilustrar o uso de um caractere de indicação de compressão, suponha que uma cadeia de caracteres repetidos ocorreu, como, por exemplo, XXXXXX. Um possível método para comprimir este grupo seria codificar um caractere especial de indicação de compressão, seguido de uma contagem e o caractere real que foi comprimido. Esta técnica de codificação é conhecida como codificação do run-length e será discutida mais detalhadamente no capítulo 3, quando examinaremos uma variedade de técnicas de compressão orientadas a caractere. Sob esta técnica de codificação, qualquer cadeia repetida de caracteres com mais de três pode ser codificada como ilustrado na Figura 2.5.

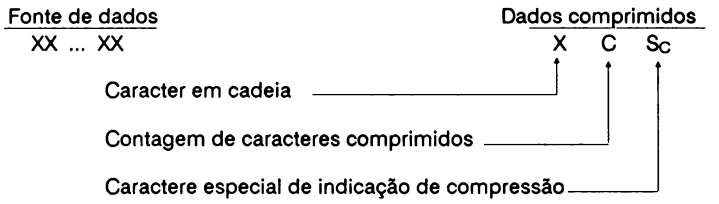


Figura 2.5 - Codificação run-length.

A chave para a seleção de caracteres de indicação de compressão apropriados depende de diversos fatores, incluindo os códigos de dados empregados, o número de mecanismos de compressão orientados a caractere que você usará e, quando for necessário, conhecimento anterior dos dados a serem transmitidos ou armazenados. Destes três fatores, sem dúvida o mais importante é o código de dados empregado que terá um peso mais relevante sobre os outros fatores.

Alguns códigos de dados, tal como EBCDIC, têm caracteres não atribuídos no conjunto de caracteres. Nestas situações, a seleção de caracteres especiais de indicação de compressão envolve a seleção de caracteres não atribuídos de um conjunto de caracteres que você está usando.

À medida em que incorporar uma mistura de técnicas de compressão orientadas à caractere, você pode ficar sem caracteres não atribuídos. Além disso, se você usa Baudot, ASCII de sete níveis, ou outros códigos que não tenham caracteres não atribuídos no conjunto de caracteres você deve usar uma técnica diferente para obter caracteres de indicação de compressão. Nestas situações pode considerar o uso da capacidade de código estendido de certos códigos de dados ou a técnica de inserir e deletar.

A utilização de código estendido

Alguns conjuntos de caracteres, incluindo ASCII e EBCDIC, têm a capacidade embutida de redefinir caracteres através da utilização de caracteres Shift Out (SO) e Shift In (SI). O caractere Shift Out, de fato, indica que todos os caracteres seguintes a ele (até que um caractere Shift In (SI) seja encontrado) tenham novos significados representados por um novo conjunto de caracteres que você definiu. A Figura 2.6 ilustra o uso de um código estendido para implementar a compressão run-length. Observe que o formato de dados comprimidos requer, agora, cinco caracteres. Em compensação, o uso de um caractere existente no conjunto de caracteres, como sendo o caractere especial de indicação de compressão requer somente três caracteres. Assim, um código estendido para a compressão run-length somente seria benéfico quando um grupo de caracteres repetidos ultrapassasse cinco.

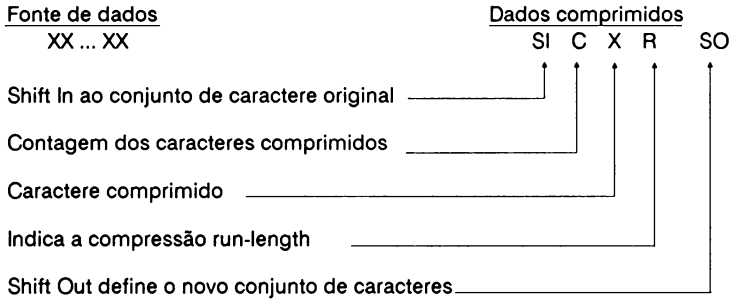


Figura 2.6 - A codificação run-length usando um código estendido.

A técnica de inserção e deleção

A técnica de inserção e deleção pode ser considerada semelhante às técnicas usadas em muitos mecanismos de transmissão de dados para efetuar a transparência de dados e evitar que as seqüências de execuções de definição dos bits sejam mal interpretadas como um flag no protocolo HDLC. Sob a técnica de inserção e deleção, você deve selecionar caracteres de indicação de compressão que não são freqüentemente encontrados nos dados a serem comprimidos. Na língua inglesa você pode selecionar tais caracteres como o Q e o Z, uma vez que sua freqüência de ocorrência é muito baixa em comparação com os outros caracteres.

Fonte de dados	Dados comprimidos
JOHN Q. PUBLIC	JOHN QQ. PUBLIC
$XXXXXX 5.15	$QX65.15

Figura 2.7 - A codificação run-length usando a técnica de inserção e deleção para formar um caractere especial de indicação de compressão.

Uma vez que seus caracteres de indicação de compressão estejam selecionados, seu algoritmo de compressão deve também examinar os dados para detectar a ocorrência natural de caracteres de indicação de compressão. Se estes caracteres aparecerem naturalmente nos dados, o processo de codificação acrescentará um segundo caractere, enquanto o processo de decodificação deve então deletar o caractere previamente inserido, daí o termo inserção e deleção. A Figura 2.7 ilustra o uso do caractere Q como um caractere especial de indicação de compressão, quando duas cadeias são manipuladas pelo uso da técnica de inserção e deleção para formar o caractere de indicação de compressão. Observe que no primeiro exemplo o fluxo de dados comprimidos realmente representa a expansão de dados, já que a compressão run-length não poderia ser usada para comprimir a fonte de dados e o caractere Q surgiu naturalmente na cadeia de dados. No segundo exemplo, a cadeia de seis Xs foi comprimido com o Q servindo de caractere de indicação de compressão.

CAP. 03

TÉCNICAS DE COMPRESSÃO DE DADOS

A enorme expansão da computação remota, na última década, concentrou o interesse das equipes de comunicações nas técnicas de compressão de dados. Originalmente, na década de 60, chamou a atenção do usuário de processamento de dados como um mecanismo para aumentar a capacidade dos dispositivos de armazenamento de massa; hoje, a compressão está sendo aplicada no campo das comunicações de dados. Aqui, a compressão tem, como conseqüência, a transferência de dados em períodos de tempo menores do que se estes fossem transmitidos sem o emprego de uma técnica de compressão.

Neste capítulo, abordamos oito métodos distintos que podem ser empregados para comprimir dados. Cada um dos métodos, descritos aqui, é implementado com base no uso de um caractere especial de indicação de compressão, e pode ser classificado como uma técnica de compressão de dados orientada a caractere. Embora cada uma das técnicas possa ser implementada individualmente, do ponto de vista prático é muito comum implementar duas ou mais técnicas de compressão orientadas a caractere, já que os dados a serem comprimidos serão, normalmente, mais susceptíveis à mistura de métodos de compressão.

A compressão orientada a caractere, embora ofereça o potencial de reduzir, de forma significativa, o armazenamento de dados ou os requisitos de transmissão, não deve ser vista como um meio único de compressão. Em muitos casos, a compressão orientada a caractere pode ser o primeiro nível de um esquema de compressão multinível, com outros níveis empregando o método de compressão baseado em estatística, que codifica os dados com base na freqüência de ocorrência de todos os caracteres, incluindo os

caracteres especiais de indicação de compressão. Consulte o capítulo 4 para obter informações específicas a respeito dos métodos de compressão de dados baseados em estatística.

Além de examinar os métodos de compressão orientados a caractere, diversas combinações de técnicas são discutidas com ênfase em sua utilização e em sua eficiência. Algumas das técnicas abordadas neste capítulo, exigem, para serem eficazes, uma análise cuidadosa do tráfego de dados atual ou projetado. No desenvolvimento do software, que irá realizar os algoritmos de codificação e decodificação, nenhuma das técnicas apresentadas requer mais que um nível moderado de dificuldade. De fato, a maioria das técnicas, neste capítulo, deve ser de fácil implementação para os usuários finais, sendo que sua implementação pode resultar em um alto grau de redução de dados com uma quantidade mínima de esforço.

Ao se aplicar uma ou mais técnicas de compressão orientadas a caractere, as eficiências operacionais podem ser aumentadas ou os custos de transmissão reduzidos. Para o primeiro caso, a compressão de dados permitirá um aumento nas informações transferidas através da ligação de dados por intervalo de unidade de tempo. Com relação ao segundo caso, reduzir a quantia de dados a serem transferidos fisicamente, pode-se tornar possível o emprego de ligações de dados de menor velocidade, resultando em uma redução de custo, se comparado com a despesa de uma ligação de dados que opera a uma taxa de dados mais alta.

3.1 SUPRESSÃO DE CARACTERES NULOS

A supressão de caracteres em branco ou nulos foi uma das primeiras técnicas de compressão de dados desenvolvida. Hoje, esta técnica simplista é empregada no protocolo de transmissão IBM 3780 BISYNC freqüentemente usado. Além disso, a supressão de caracteres nulos é freqüentemente usada com a mistura de outras técnicas de compressão orientadas a caractere, para reduzir o armazenamento e a transmissão de dados.

Visão geral da técnica

Como o próprio nome indica, a supressão de caracteres nulos é uma técnica de compressão de dados que varre um fluxo de dados a procura de repetidos caracteres em branco ou nulos. Ao encontrar esta seqüência, os caracteres em branco ou nulos são

substituídos por um par especial de caracteres cujo formato encontra-se ilustrado na figura 3.1. Primeiro, o caractere indicador de compressão é empregado para indicar que ocorreu a supressão de caracteres nulos. O segundo caractere é usado para indicar a quantidade de caracteres nulos que foram encontrados e substituídos pela seqüência de dois caracteres (Aronson, 1977; Ruth e Kreutzer, 1972).

A. Formato de compressão

CONTAGEM DE NULO	CARACTERE INDICADOR DE COMPRESSÃO

B. Exemplo de compressão de dados

Fluxo original de dados XYZbbbbbQRX
Fluxo de dados comprimidos XYZSc5QRX
onde: Sc = caractere especial indicador de compressão

C. Processo de varredura de dados

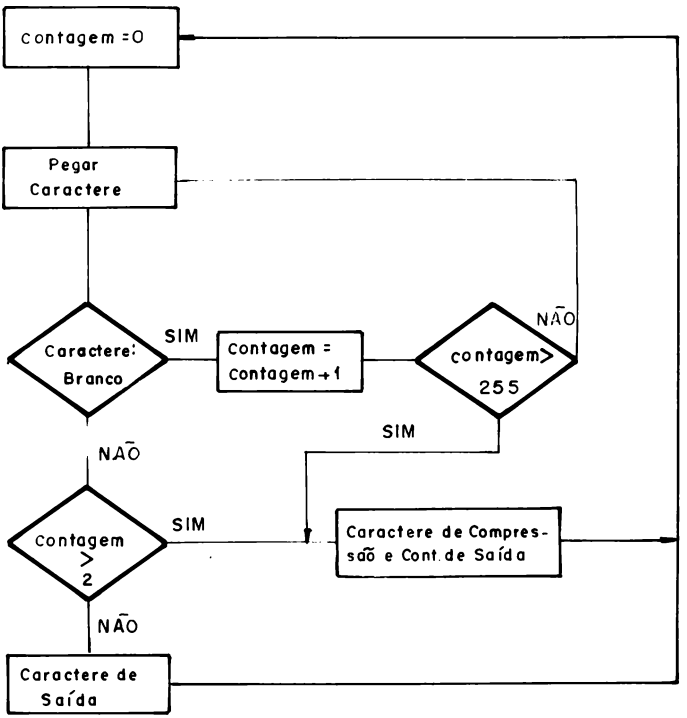


Figura 3.1 - Supressão de caracteres nulos.

Quando a seqüência de dois caracteres é transmitida dentro de um fluxo de dados, o dispositivo de recebimento executa uma busca do caractere especial, usado para indicar a supressão de caracteres nulos. Ao 'encontrar' o caractere, o receptor sabe que o próximo caractere contém a contagem do número de nulos que foram comprimidos. A partir desta informação, o fluxo original de dados pode ser reconstruído.

Na parte central da figura 3.1, encontra-se um exemplo da aplicação de supressão de caracteres nulos em um fluxo de dados. Aqui, o caractere S_c indica um caractere especial de indicação de compressão, avisando que a supressão de caracteres nulos ocorreu.

Na parte inferior da figura 3.1, está ilustrado o fluxograma do processo de varredura de supressão de caracteres nulos. Se presumirmos um formato de 8 bits para os caracteres de dados, então o contador de caracteres pode armazenar valores de até 255 caracteres nulos, encontrados seqüencialmente antes do estouro do valor do contador, isto se começarmos pelo número 1. Se começarmos pelo número 0, representando o valor 1, teremos 256.

Limitações

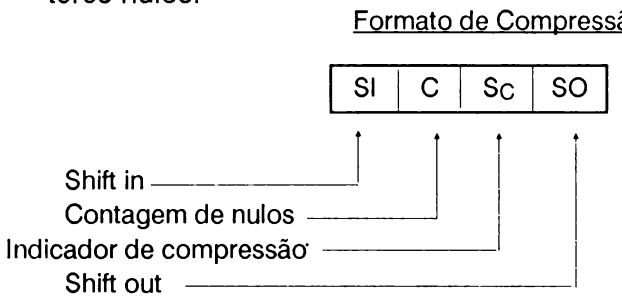
Já que a compressão de até 255 caracteres nulos, encontrados seqüencialmente, sempre resulta em uma seqüência de compressão a dois caracteres, não há possibilidade de haver economia, a não ser que três ou mais caracteres nulos sejam encontrados em seqüência. Assim, uma seqüência de dois caracteres nulos não deve ser trocada pelo formato comprimido de supressão de nulos. Isto porque não resultaria em uma economia e porque o processo de compressão e descompressão exige uma parte do tempo do processador. Além disso, se você estiver empregando diversas técnicas de compressão de dados, veremos que os dois caracteres nulos, encontrados seqüencialmente, podem ser efetivamente comprimidos pelo processo de codificação Diatômica. Esta técnica de compressão resulta em uma redução de dados de 100 por cento para a seqüência de dois caracteres nulos, em contraste com a técnica de supressão de caracteres nulos que é ineficaz.

Uma limitação chave associada ao uso da supressão de caracteres nulos se refere à seleção do caractere especial de indicação de compressão. Se um caractere indefinido, do conjunto de caracteres que você está usando, estiver disponível, então as economias começam a se acumular, quando dois ou mais caracteres nulos seqüenciais são encontrados. Se você tem de usar um conjunto de

caracteres estendidos definido primeiramente pelo caractere shift out (SO), você precisará de uma seqüência de quatro caracteres para codificar uma fileira de nulos. A figura 3.2 ilustra o formato de compressão e um exemplo de supressão de caracteres nulos, usando um código estendido.

Como se pode observar no exemplo da figura 3.2, a supressão de caracteres nulos somente é viável para uma seqüência de quatro ou mais nulos, quando um código estendido tem de ser usado para formar um caractere especial de indicação de compressão.

Para considerar a técnica de inserção e deleção, quando se usa o caractere de indicação de compressão, você precisa ter conhecimento da freqüência de ocorrência do caractere selecionado para inserção e deleção, em comparação à economia potencial que pode advir da supressão de caracteres nulos. Se a freqüência de ocorrência for maior que a porcentagem potencial de redução de dados, seu algoritmo resultará na expansão de dados, embora realmente suprimisse todas as ocorrências de três ou mais caracteres nulos.



Exemplo de compressão de dados

Fluxo original de dados	XYZ⌀⌀⌀⌀⌀QRX
Fluxo de dados comprimidos	XYZSOSC5SIQRX

Figura 3.2 - Supressão de caracteres nulos usando um código estendido.

Embora a supressão de caracteres nulos seja encarada como uma técnica elementar de compressão de dados, ela é muito fácil de se implementar; os resultados obtidos com esta técnica podem ser substanciais. Foram mostrados ganhos de rendimento de 30 a 50 por cento, em várias instalações de computadores que mudaram do 2780 com a seqüência de controle de transmissão bissíncrona

e que não comprimia dados para o 3780 com seqüência que executava a supressão de caracteres nulos.

Variações da técnica

Duas variações da técnica de supressão de caracteres nulos podem ser usadas, para comprimir partes de documentos contendo campos variáveis ou predefinidos de margem esquerda. Em uma situação particular, pode ser útil reservar um grupo dentro do conjunto de caracteres para representar diversos números predefinidos de espaços ou de caracteres nulos. Assim, um caractere pode, então, representar o campo de margem esquerda com 5 espaços, enquanto um segundo caractere poderia ser usado para representar os 20 espaços necessários, para tabular até o início da coluna dentro de um documento.

Já que os campos de margem esquerda predefinidos representam posições de parada de tabulação, uma segunda variação da técnica de supressão de caracteres nulos é obtida pelo emprego do caractere de tabulação. Se as paradas de tabulação forem predefinidas, você só tem de substituir a seqüência de espaços ou de caracteres nulos por um caractere de tabulação, onde este indicará que o próximo caractere começa em uma determinada coluna daquela linha, e todas as colunas entre o último caractere e a localização do começo do próximo caractere, são compostas de espaços ou de caracteres nulos. Para ilustrar este conceito, vamos supor que uma parte do documento, que desejamos transmitir, seja como a seguir:

Vamos, agora, analisar o reflexo na economia com os gastos com defesa:

Ano	Armas	Manteiga
xxxx	yyyy	zzzz

Observe que há quatro localizações distintas de parada de tabulação neste documento - o campo de margem esquerda de um parágrafo e das três posições de coluna. Assim, uma parada de tabulação seguida pelo caractere 'N' poderia ser usada para posicionar o início do parágrafo na posição apropriada. Já que o campo de margem esquerda ocorre antes da primeira posição de coluna, para se posicionar 'A' na coluna ano, seriam necessárias duas paradas de tabulação. De modo semelhante, o 'A' em armas teria de ser precedido de três caracteres de parada de tabulação e, etc.

À medida que o campo de margem esquerda e as posições de localização de coluna exclusivos aumentam em um documento ou entre documentos diferentes, o número de caracteres de parada de tabulação, que poderiam ser usados para representar uma localização predefinida, resultariam na expansão de dados em vez de em sua compressão. Para evitar que estas situações ocorram, assim como para eliminar a necessidade de se ter conhecimento anterior sobre o campo de margem esquerda e localizações de coluna, pode ser empregado um procedimento de parada de tabulação variável. Ao usar paradas de tabulação variável, você simplesmente substituir o caractere de parada de tabulação e a posição da coluna para tabular, por espacejamento entre as colunas. Voltando ao exemplo anterior, se 'ano' começa na coluna 15, enquanto 'armas' e 'manteiga' começam nas colunas 30 e 45 respectivamente, a linha de nomes das colunas poderia ser substituída pela seqüência Ts15 Ano Ts30 Armas Ts45 Manteiga, onde Ts representa o caractere de parada de tabulação.

Uma das aplicações chave para a substituição de cadeias de caracteres nulos por paradas de tabulação, são as transmissões CICS do mainframe IBM. Diversas empresas de software comercializam produtos que interceptam as transmissões CICS externas para terminais e, entre muitas funções, substituem cadeias de nulos por seqüências de tabulação. Ao transmitir uma quantidade menor de dados, tanto a eficiência do sistema de transmissão, quanto os tempos de resposta do usuário melhoram.

3.2 MAPEAMENTO DE BIT

Esta técnica de compressão se torna efetiva quando os dados a serem manipulados forem compostos por uma alta proporção de tipos de dados específicos, tais como numéricos, ou de uma proporção maior de um caractere específico, tais como espaços em branco. Como o próprio nome diz, um mapa de bits é empregado para indicar a presença ou a ausência de caracteres de dados, ou o fato de que certos caracteres de dados foram manipulados anteriormente e devem ser manipulados novamente, para que os dados voltem ao seu formato original.

Processo de codificação

Para examinar a técnica de mapeamento de bit e sua aplicação, veremos, inicialmente, como ela pode ser empregada para implementar um dos tipos de supressão de caracteres nulos. Na parte

esquerda da figura 3.3, está ilustrada uma parte do fluxo de dados, composto por 3 caracteres de dados e 5 caracteres nulos. Aqui, os 5 nulos representam 62,5 por cento do conteúdo da cadeia e estão espalhados por todo o fluxo de dados em uma seqüência randômica. Já que a supressão de caracteres nulos é somente efetiva quando 3 ou mais espaços em branco seqüenciais são encontrados, seu uso somente reduziria a cadeia de 8 para 7 caracteres de comprimento.

Através do uso de um mapa de bit anexado na frente da cadeia, podemos indicar a presença ou ausência de nulos e, conseqüentemente, reduzir o tamanho da cadeia de dados. Na parte inferior da figura 3.3, encontra-se ilustrada a utilização de um caractere de mapa de bits, onde todos os nulos são soltos da cadeia de dados e o bit que corresponde à posição de nulo é estabelecido em zero, enquanto a posição de bit no mapa, que corresponde ao caractere de dados válido é definida em um.

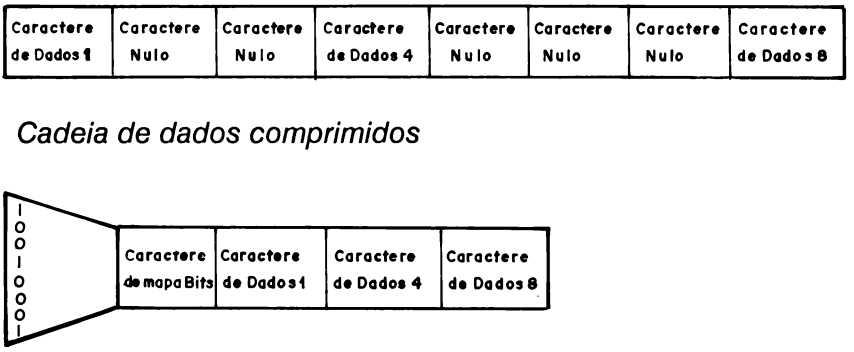


Figura 3.3 - O processo de mapeamento de bit. Em um fluxo de dados típico, há uma alta probabilidade que um ou mais caracteres sejam repetidos. Usando um caractere para servir como um mapa de bit, ele pode eliminar a alta freqüência de ocorrência de caracteres de um fluxo de dados.

Ao comparar a cadeia de dados comprimidos com a cadeia original de dados, os 8 caracteres de dados, incluindo os nulos, foram reduzidos para 4 caracteres; 3 caracteres de dados e o caractere de mapa de bits. Isto resulta na razão de compressão de 2:1 para esta determinada aplicação.

Considerações de Hardware

O caractere de mapa de bits, ilustrado na figura 3.3, indica as posições de caracteres de dados não nulos pela localização, da esquerda para a direita. Ao inverter a ordem do mapa de bits, as posições dos elementos de dados podem ser indicadas da direita para a esquerda. A figura 3.4 mostra dois métodos diferentes para se formar o mapa de bits que irá representar a cadeia de dados comprimidos. Usando a técnica de posicionamento do elemento de dados por mapa de bits, ilustrada na parte inferior da figura 3.4, o caractere de mapa de bits resultante da fluxo original de dados, conforme ilustrado na figura 3.3, ficaria 10001001. O conjunto de instruções do dispositivo de hardware para a execução da técnica de supressão por mapa de bits em estudo, regerá o método de posicionamento do elemento no mapa de bits a ser empregado. Isto pode ser facilmente explicado ao examinar, primeiramente, o fluxograma das funções que têm de ser executadas na cadeia original de dados, de maneira a construir o mapa de bits e a cadeia de dados comprimidos.

O processo de supressão por mapa de bits está ilustrado funcionalmente na figura 3.5. A rotina de software para comprimir os dados deve, em primeiro lugar, iniciar o contador de posição do mapa de bits (1), o mapa de bits (2) e o contador de caracteres (3).

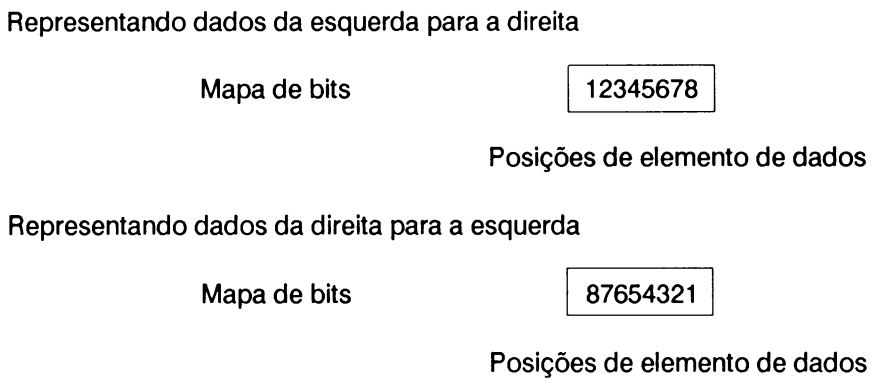


Figura 3.4 - O posicionamento de elemento do mapa de bits. Os dois métodos podem ser empregados para representar a cadeia de dados comprimidos no mapa de bits - dados representados da esquerda para a direita e da direita para a esquerda.

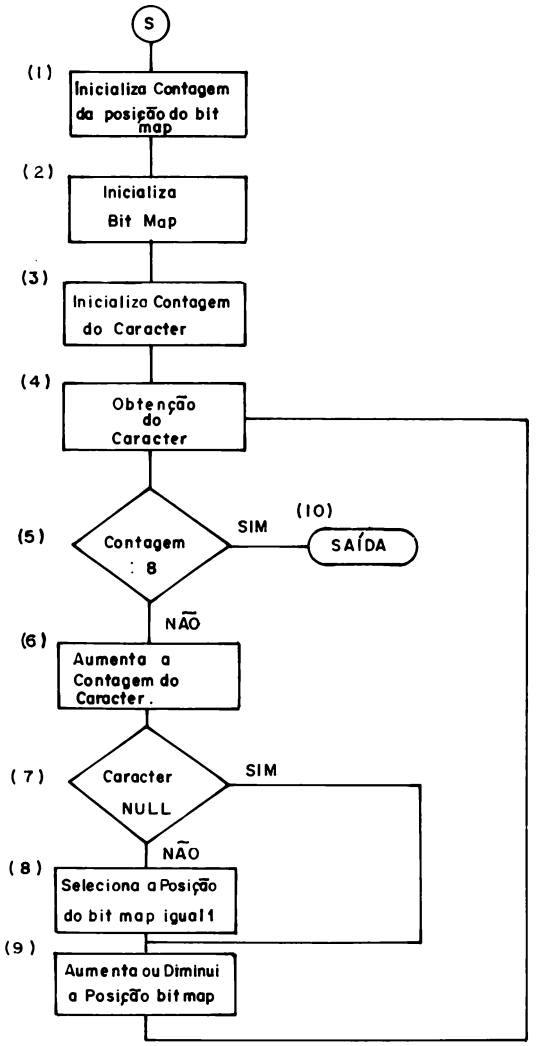


Figura 3.5 - Fluxograma da função de supressão por mapa de bits.

Depois que um caractere é obtido (4), o contador de caracteres é comparado com oito (5). Se forem iguais, indica que os oito caracteres de entrada foram processados e, assim, podemos sair da rotina (10). Se não forem iguais, o contador de caracteres é incrementado (6) e o caractere em análise é comparado com o caractere nulo (7). Se o caractere em análise não for nulo, a posição

do mapa de bits é definida como sendo o binário 1 (8). Se o caractere for nulo, esta função (8) é pulada. A seguir, a posição do mapa de bits é incrementada ou decrementada (9), de maneira que o mapa de bits seja preparado para ser definido como zero na localização de bit seguinte, se o próximo caractere examinado for nulo. Finalmente, depois dos oito caracteres serem processados, o contador se iguala a oito (5) e a ramificação de saída da rotina é seguida (10).

Do ponto de vista do hardware, o método usado para executar as funções indicadas nos blocos (8) e (9), da figura 3.5, depende das instruções de shift e lógicas disponíveis para serem utilizadas pelo programador. Esta inter-relação pode ser vista ao indicar o efeito no caractere de mapa de bits, à medida que os caracteres de dados subseqüentes são examinados. Na figura 3.6, está ilustrado o efeito sobre o mapa de bits e sobre a 'máscara', à medida que uma progressão de caracteres de dados é examinada. Aqui, a máscara é obtida simplesmente deslocando o binário 1 através das posições no mapa de 8 bits e, feita uma função lógica "OU" com o mapa de bits, quando o caractere de dados não for nulo. Ao examinar a máscara, podemos observar que uma operação aritmética ou lógica de deslocamento à esquerda é necessária, se desejarmos posicionar nosso mapa de bits de maneira que o bit à direita indique a presença ou ausência de um caractere nulo no primeiro elemento da cadeia original de dados. Assim, do ponto de vista do hardware, a instrução de deslocamento disponível será um fator decisivo para se determinar a maneira que os elementos do mapa de bits são posicionados. Embora a maioria dos microprocessadores, minicomputadores e certamente, computadores de grande porte tenha funções de deslocamento à esquerda e à direita, alguns microprocessadores podem ter capacidade limitada de deslocamento. Esta capacidade deve ser examinada antes de tentar implementar esta técnica.

Caractere de dados	Mapa de bits inicial	Máscara	Mapa de bits ⊕ máscara(ou mapa de bit se nulo)
Dados	00000000	00000001	00000001
Nulo	00000001	00000010	00000001
Dados	00000001	00000100	00000101
Dados	00000101	00001000	00001101
Nulo	00001101	00010000	00001101
Nulo	00001101	00100000	00001101
Nulo	00001101	01000000	00001101
Dados	00001101	10000000	10001101

Figura 3.6 - O processo de mascaramento do mapa de bits. O caractere de máscara é obtido deslocando-se o binário 1 por todas as posições de bit e feita uma função lógica "OU" com o mapa de bits, quando o caractere de dados não for nulo.

A eficiência da supressão

No exemplo anterior, o caractere do mapa de bits continha 8 bits. Embora o exemplo mostrasse uma redução, da cadeia original de dados à cadeia de dados comprimidos, de 50 por cento dos caracteres, devemos considerar o que acontece com a eficiência de compressão quando a porcentagem de nulos na cadeia de dados diminui. A tabela 3.1 mostra a razão de compressão com base na porcentagem de nulos contidos na cadeia, para um mapa de 8 bits. Quando não há caracteres nulos, a cadeia resultante de dados aumenta em mais 1 caractere, resultado da adição do caractere de mapa de bits, gerando, asssim, uma razão de compressão de 0,888. Isto significa que no pior caso, onde não haja caracteres nulos a serem suprimidos, teremos 12,5 por cento de dados adicionais como resultado do emprego desta técnica de compressão.

Podemos desenvolver um modelo matemático da eficiência de supressão, como a seguir. Se p for a probabilidade de que qualquer caractere seja nulo, o número estimado de nulos em uma cadeia de comprimento S é denominado S*p*.

Tabela 3.1 Eficiência de compressão e porcentagem de nulos

Porcentagem de nulos	Tamanho da cadeia resultante	Razão de compressão
0,0	9	0,888
12,5	8	1,000
25,0	7	1,143
37,5	6	1,334
50,0	5	1,600
62,5	4	2,000
75,0	3	2,667
87,5	2	4,000
100,0	1	8,000

Usando a compressão de caracteres nulos, este será codificado como uma cadeia de comprimento

$$S * (1 - p) + \left[\frac{S}{8}\right]$$

e, então, a razão de compressão é

$$\left(\frac{1}{S}\left\{S\,(1-p) + \left[\frac{S}{8}\right]\right\}\right)^{-1} \cong \left((1-p) + \frac{1}{8}\right)^{-1}$$

para valores grandes de S.

Variações do mapeamento de bits

Na abordagem anterior, sobre o procedimento de mapeamento de bits, supomos que um caractere nulo ou qualquer outro caractere, que aparecesse em grande proporção se comparado ao restante dos dados, tenha de ser suprimido. Para algumas aplicações, não há um determinado caractere que seja encontrado com maior freqüência do que outros; contudo, em certos casos, você pode encontrar uma situação onde um tipo específico de dados,

como os numéricos, apareçam freqüentemente. Uma aplicação onde esta situação poderia existir é na área de controle de processo, onde as leituras numéricas de vários equipamentos são transmitidas à central para processamento e os sinais de controle voltam aos dispositivos com base em determinados critérios predefinidos. Dependendo do código de transmissão empregado, alguma economia pode ser obtida pelo uso da técnica de mapeamento de bit. Se os dados a serem transmitidos estiverem no EBCDIC, código ampliado de caracteres decimais codificados em binário para intercâmbio de informações, então, as quatro primeiras posições de bits de cada caractere numérico estarão todas com 1. Assim, o caractere de mapeamento de bits poderia ser empregado para indicar o número de caracteres compactados na cadeia comprimida, onde cada caractere conteria dois dígitos com as quatro primeiras posições precedentes arrancadas. Esta técnica está ilustrada na figura 3.7 e é muito semelhante à técnica de compactação de meio byte que abordaremos, mais tarde, neste capítulo.

As limitações da técnica

Uma limitação chave da técnica de mapeamento de bits é que ela é aplicável aos dados que têm unidades de tamanho fixas, tais como: caracteres, bytes ou palavras.

Cadeia original de dados

1
8
4
6
2
7
9
8
3
2

Cadeia de dados comprimidos

00001010 Mapeamento de Bits	
8	1
6	4
7	2
8	9
2	3

Figura 3.7 - Supressão de meio dígito. Na técnica de supressão de meio dígito, o conteúdo do mapa de bits especifica o número de dígitos que o segue, compactado na forma de dois por caractere.

Quando usada para suprimir um determinado caractere, tal como um nulo, a razão de compressão desta técnica é diretamente proporcional à porcentagem de ocorrência daquele caractere no fluxo original de dados. Assim, se determinado um caractere em uma cadeia de dados ocorrer 30 por cento do tempo, enquanto um segundo caractere encontrado com maior freqüência ocorre, vamos dizer, 25 por cento do tempo, esta técnica ignora a alta porcentagem de ocorrência do segundo caractere, ou quaisquer outros caracteres. Como veremos a seguir, uma técnica conhecida como codificação de comprimento de fileira pode ser empregada, para tirar vantagem da redundância de ocorrência adjacente de todos os caracteres em um fluxo de dados.

Existem outras duas limitações associadas à técnica de mapeamento de bits, que são: a exigência de se operar com somente um tipo de caractere, tais como nulo ou dígito, e se desenvolver um método para reconhecer que um certo caractere representa um mapa de bits, em vez de dados. Vamos examinar como podemos superar a segunda limitação, quando se usam dados ASCII de 7 níveis e como podemos desenvolver um método para superar esta limitação, quando se usam outros conjuntos de caracteres.

Se estivermos usando ASCII de 7 níveis, podemos tirar o bit de paridade e usar a oitava posição de bit como o indicador de mapa de bits. Embora esta técnica reduza a

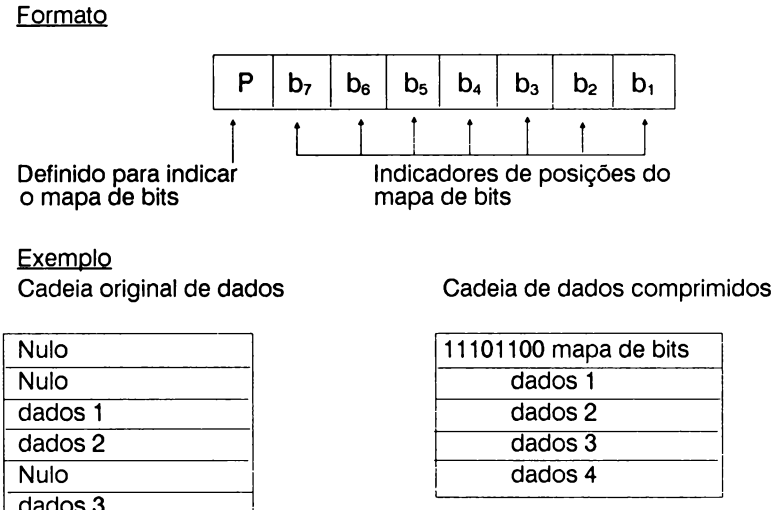


Figura 3.8 - Usando o bit de paridade com um indicador de mapeamento de bit.

razão de compressão máxima que pode ser alcançada através do uso da técnica de mapeamento de bits, ela também elimina a possibilidade de expansão de dados. Para ilustrar como podemos evitar a expansão de dados, vamos examinar como esta técnica revisada trabalha.

A figura 3.8 ilustra a formação de um indicador de mapa de bits, através da ativação do bit de posição 8, que no ASCII de 7 níveis é usado como um bit de paridade. Na parte inferior da figura 3.8, o fluxo de dados comprimidos resultante é mostrado com base no fluxo original de dados com sete caracteres. Agora, vamos supor que a composição dos próximos sete caracteres mude, de maneira que não se encontrem caracteres nulos. Sob esta técnica revisada, os caracteres são simplesmente colocados no fluxo de dados comprimidos, sem a necessidade de acrescentar um mapa de bits que resultaria na expansão de dados. Aqui, os dados são reconhecidos como dados, já que o bit na posição de paridade não está ativado. Outra vantagem associada a esta técnica de compressão revisada é o fato de que a ativação do bit na posição, que anteriormente era de paridade, resulta no reconhecimento automático do caractere como o mapa de bits de 7 posições. Isto lhe permite eliminar a codificação necessária para acompanhar os mapas de bits verso os dados.

Se você estiver usando todas as posições de bit dentro do caractere, tal como o EBCDIC de 8 níveis, você deve, então, acompanhar a relação entre os caracteres do mapa de bits e os caracteres de dados. Para proceder desta forma, você pode

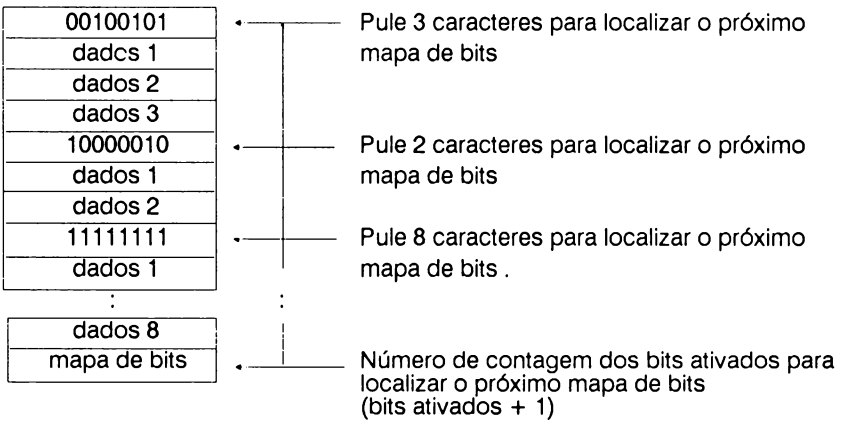


Figura 3.9 - Localizando os mapas de bits. Sucessivos mapas de bits podem ser colocados no fluxo de dados ao acrescentar 1 à contagem do número de bits ativados no mapa de bits anterior

simplesmente, contar o número de bits ativados no primeiro mapa de bits que, então, lhe diz o número de caracteres para pular antes de se ler o próximo mapa de bits. A figura 3.9 ilustra o uso da contagem dos bits ativados no mapa de bits, para determinar a localização do próximo mapa de bits no fluxo de dados.

3.3 COMPRIMENTO DE FILEIRA

A codificação de comprimento de fileira é um método de compressão de dados que reduzirá, fisicamente, qualquer tipo de seqüência de caracteres repetidos, desde que a seqüência de caracteres alcance um nível predefinido de ocorrência. Para a situação especial onde o caractere nulo é o caractere repetido, a compressão de comprimento de fileira pode ser vista como um processo avançado de supressão de nulos (Rubin, 1976; Ruth e Kreutzer, 1972).

Operação

De maneira semelhante ao método usado para efetuar a supressão de nulos, o emprego da codificação de comprimento de fileira normalmente requer o uso de um caractere especial para indicar que ocorreu este tipo de compressão. Uma exceção ao uso de um caractere especial de indicação de compressão é o método de compressão Classe 5 Microcom Networking Protocol (MNP), classes, descrito posteriormente nesta seção. Quando um caractere indicador de compressão é usado, ele é normalmente seguido por um dos caracteres repetidos, que estava na cadeia de caracteres repetidos, que foi encontrada. Finalmente, um caractere de contagem indicará o número de vezes que o caractere repetido apareceu na seqüência.

Quando códigos como ASCII ou EBCDIC são empregados, uma boa escolha para o caractere especial é aquela que não ocorrerá na cadeia de dados. Para cada um destes códigos há diversos caracteres não designados com representações únicas de bits que podem ser usados. Para as situações onde o conjunto de caracteres não contenha caracteres não utilizados, tal como o código Baudot de 5 níveis (de bits), esta técnica ainda pode ser usada desde que se selecione um caractere que não possa ser usado duas vezes em sucessão, tais como shift letra e shift número, para indicar que ocorreu compressão. O leitor deve consultar o capítulo 2 para obter informações adicionais com relação à seleção e à

utilização de caracteres de indicação de compressão a partir de diferentes códigos de caractere.

Processo de codificação

O processo de compressão de comprimento de fileira se dá como uma cadeia de caracteres repetidos, sendo convertida em uma cadeia de dados comprimidos, como mostra a figura 3.10 (Aronso, 1977). Já que são necessários três caracteres para indicar compressão, a codificação de comprimento de fileira é somente efetiva, quando uma cadeia de dados contém uma seqüência de quatro ou mais caracteres repetidos de dados.

Três exemplos da aplicação da codificação de comprimento de fileira nas seqüências de caracteres repetidos estão apresentados na Tabela 3.2. Observe que Sc representa o caractere especial usado para indicar a ocorrência da codificação de comprimento de fileira, enquanto o símbolo 'b' é usado para indicar a presença de um caractere em branco.

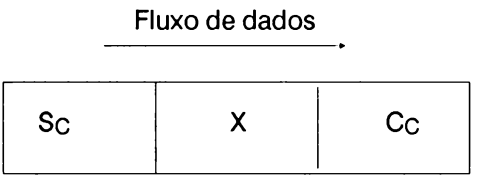


Figura 3.10 - Codificação de comprimento de fileira, formato geral de compressão. Na codificação de comprimento de fileira, um caractere especial, um caractere de dados repetidos e um caractere de contagem de caractere são necessários para indicar os parâmetros de compressão.

S_C = Caractere especial indicando a existência da compressão.

X = Qualquer caractere repetido de dados.

C_C = Contagem de caracteres. Esta contagem corresponde ao número de vezes que o caractere comprimido tem de ser repetido.

Tabela 3.2 Aplicando a codificação de comprimento de fileira

Cadeia original de dados	Cadeia de dados codificados
$******55.72	$Sc*655.72
-------	Sc-9
Gunsb̸b̸b̸b̸b̸b̸b̸b̸b̸b̸b̸b̸b̸b̸b̸b̸b̸b̸b̸b̸b̸b̸Butter	GunsScb̸1XButter

Uma vez que o formato de supressão de nulos exige dois caracteres, o emprego da compressão de comprimento de fileira para suprimir nulos, sempre resulta em um caractere adicional gerado no fluxo de dados comprimidos. Embora isto não seja significativo quando grandes cadeias de nulos são comprimidas, diversas cadeias pequenas de nulos poderiam resultar em uma quantidade excessiva de dados comprimidos. Isto sugere a você que se deve levar em conta o uso de uma mistura de diversos algoritmos para executar a compressão de dados.

As principais etapas do processo de codificação de comprimento de fileira estão apresentadas na figura 3.11, através do uso de um fluxograma de sistemas. Inicialmente, um contador de caracteres (1) e um contador de repetição de caracteres (2) são iniciados com zero. Depois que um caractere da cadeia original de dados for obtido (3), o contador de caracteres é incrementado (4) de um. Esta contagem de caractere é, então, comparada com um (5). No primeiro ciclo, esta comparação sempre é verdadeira e o caractere é, então, colocado na área do buffer (armazenamento temporário) (6) para processamento posterior se for descoberto que a cadeia original de dados contém quatro ou mais caracteres de dados repetidos. Para o segundo e para os ciclos subseqüentes, o caractere obtido a partir da cadeia original de dados (3) é comparado com o caractere colocado no buffer (7). Se o caractere atual for igual ao caractere do buffer, a compressão pode ser possível se quatro ou mais caracteres idênticos forem encontrados em seqüência. Assim, quando o caractere obtido se iguala ao caractere armazenado, o contador de repetição (8) é incrementado por um e outro caractere é obtido a partir da cadeia original de dados (3). Se o caractere atual em exame não se iguala ao caractere armazenado (7), o contador de repetição é comparado com quatro (9). Se for menor que quatro, a compressão não vale a pena, já que três caracteres devem ser usados para codificar os dados comprimidos. Quando o contador de repetição for igual ou maior que quatro (9), o formato de compressão (10) pode, agora, ser estabelecido.

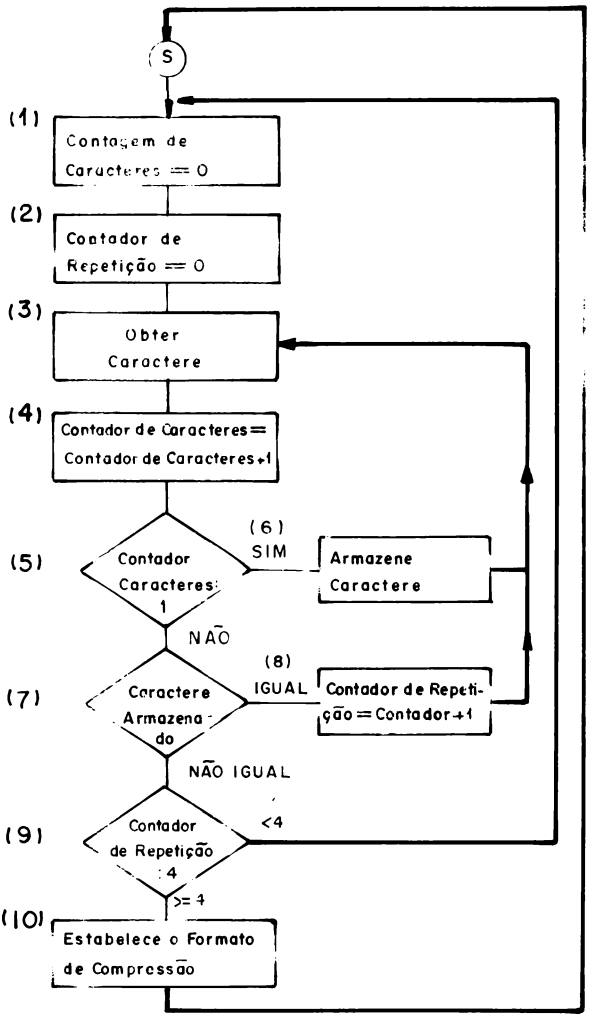


Figura 3.11 - Processo básico de codificação de comprimento de fileira .

Considerações especiais

No fluxograma básico de codificação, ilustrado na figura 3.11, presumiu-se que o contador de repetição era capaz de ter uma faixa ilimitada de valores. Na realidade, o valor máximo que o contador de caracteres repetidos pode ter é uma função do nível de código de caractere empregado. Para um código de caractere de 8 níveis (8 bits por caractere), um máximo entre 255 e 260 caracteres

repetidos podem ser representados pelo contador de caracteres. O valor exato dependerá de como o contador de caracteres é empregado. Na maioria das situações, o valor real do contador de caracteres é usado como o número de caracteres repetitivos. Deste modo, o valor máximo do contador é 2^8 - 1 ou 255. Já que o formato de compressão, ilustrado na Figura 3.10, ocorre somente quando quatro ou mais caracteres repetitivos são encontrados, a presença de um caractere de contagem de caracteres em si só implica na existência de quatro ou mais caracteres repetitivos. Assim, o contador de caracteres com todos os bits em zero pode ser usado para indicar quatro caracteres repetitivos, enquanto o contador de caracteres com todos os bits ativados em 1 indicaria, então, 260 caracteres repetitivos. Uma vez que o método de empregar o contador de repetição estiver determinado, o fluxograma da figura 3.11 deve ser modificado para aceitar uma comparação adicional do contador de repetição para testar se o valor armazenado no contador de caracteres não ultrapassa o máximo valor permitido.

Decodificação

As funções necessárias para descomprimir os dados comprimidos, de acordo com o processo de codificação de comprimento de fileira, encontram-se ilustradas na Figura 3.12 no formato de fluxograma. No começo do procedimento de descompressão, um flag de compressão é desligado (1) e um caractere é obtido da cadeia de dados comprimidos (2). A seguir, se o flag de compressão estiver desligado (3), o caractere é comparado com o caractere especial indicador de compressão de comprimento de fileira (4) para determinar se ocorreu a compressão de comprimento de fileira. Se o caractere não for o caractere especial, o próximo caractere é obtido (2). Se for o caractere indicador de compressão de comprimento de fileira, o flag de compressão é ligado (5) e o próximo caractere é obtido (2). No próximo passo, já que o flag de compressão está ligado (3), o seguinte caractere obtido (6) será o caractere repetido de dados, enquanto o próximo caractere (7) contém a contagem de caracteres. Uma vez que estes caracteres foram obtidos, o formato de descompressão pode ser iniciado (8).

Utilização

Uma das utilizações mais populares da codificação de comprimento de fileira é o subconjunto conhecido como supressão de caractere nulo. Esta técnica de compressão foi primariamente adotada no protocolo BISYNC do IBM 3780. A compressão de

espaços é uma característica padrão deste protocolo, quando o dispositivo 3780 está operando no modo linha com os dados não transparentes. Aqui, cada grupo de dois ou mais caracteres consecutivos de espaço, até 63, é substituído por um caractere IGS se o código de transmissão for EBCDIC ou por um caractere GS se o código for ASCII. Os dois caracteres são seguidos de um caractere de contagem de espaços que define o número de espaços removidos. Para a situação onde ocorrem 64 caracteres consecutivos de espaço ou mais, se torna necessária a inserção de um caractere IGS ou GS e um caractere de contagem de espaços adicionais.

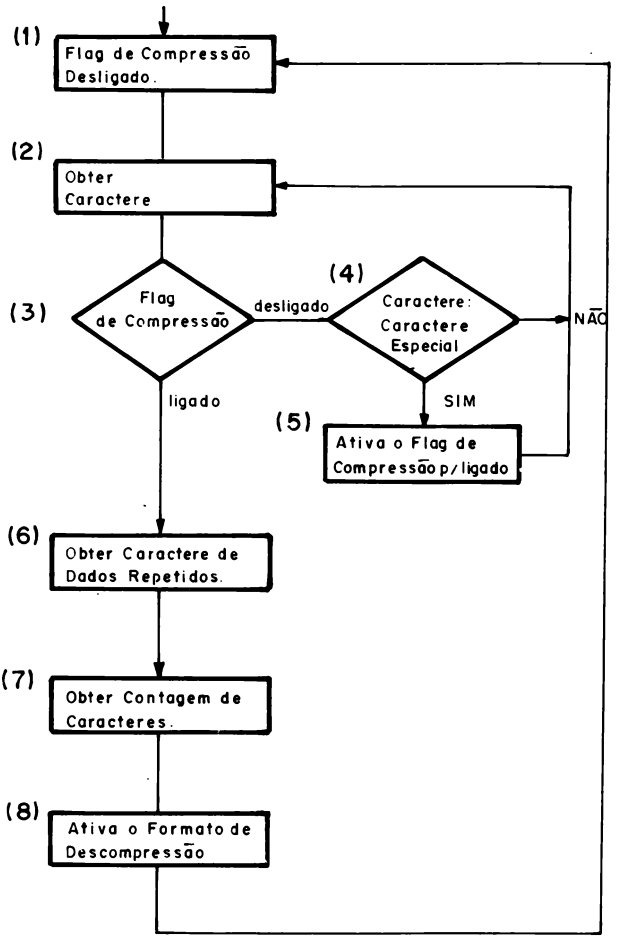


Figura 3.12 - Processo de decodificação de comprimento de fileira.

Nos sistemas Honeywell, uma versão de compressão de comprimento de fileira é usada em seu software de sistema de terminal remoto (GRTS,) rodando em processadores auxiliares que se comunicam com os terminais remotos sob o protocolo de computador remoto (RC). Além disso, o mesmo tipo de compressão é usado no sistema autônomo de fita-a-fita (SATTS) da Honeywell para a transmissão de bobinas de fitas magnéticas entre as várias localidades. Nesta versão de codificação de comprimento de fileira, um registro é examinado com o intuito de se achar uma série de três ou mais ocorrências do mesmo caractere de dados. Quando este tipo de situação ocorre, a série é comprimida e a cadeia de caracteres repetidos é convertida em uma seqüência de três caracteres, como ilustrado na figura 3.13.

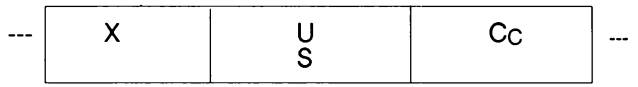


Figura 3.13 - A versão da Honeywell para a codificação de comprimento de fileira. A codificação de comprimento de fileira, como implementada nos sistemas Honeywell, difere ligeiramente da maioria dos outros fabricantes de computadores.

X = Qualquer caractere de dados repetido.
US = O caractere ASCII (0011111).
Cc = Contagem binária de 6 bits. O caractere BCD representado pela contagem binária deve ser traduzido para ASCII quando da transmissão ao subsistema de comunicações. Esta contagem é o número de vezes que o caractere comprimido deve ser repetido (máximo de 63)

A implementação mais popular da codificação de comprimento de fileira realmente apresenta uma ligeira variação na técnica de compressão de dados previamente descrita. A variação da codificação de comprimento de fileira, que alcançou uma base instalada de aproximadamente um milhão de produtos, se deu na técnica de codificação de comprimento de fileira empregada pelo Microcom Networking Protocol (MNP) Classe 5, que é licenciado para uso por mais de 50 fabricantes de modems de rede chaveada.

A compressão de dados MNP Classe 5 usa a combinação de duas técnicas de compressão de dados - condificação de comprimento de fileira e codificação de freqüência adaptativa. A técnica de codificação de freqüência adaptativa pode ser classificada como

um método de compressão baseado em estatística, e examinaremos esta técnica no capítulo 4.

No método de codificação de comprimento de fileira MNP Classe 5, uma contagem de repetição é inserida no fluxo de dados para representar o número de octetos repetidos, onde este segue as três primeiras ocorrências da seqüência. Uma vez que a versão MNP de codificação de comprimento de fileira envia os três primeiros octetos repetidos, estes caracteres repetidos servem como indicador do início da codificação de comprimento de fileira e, de fato, elimina a necessidade de utilizar um caractere especial de indicação de compressão.

A parte superior da Figura 3.14 ilustra o formato da codificação de comprimento de fileira MNP Classe 5.

A parte inferior da Figura 3.14 contém cinco exemplos da execução da codificação de comprimento de fileira MNP Classe 5 em diferentes cadeias.

Formato

onde:

X = caractere repetido
C = contagem de repetição

Exemplos

Fluxo original de dados	Fluxo de dados comprimidos
A	A
BB	BB
CCC	0CCC
DDDD	4DDD
EEEE	5EEE

Figura 3.14 - Codificação de comprimento de fileira MNP Classe 5. Uma vez que os três caracteres de repetição indicam a presença da codificação, o formato de compressão de comprimento de fileira MNP Classe 5 não requer o uso de um caractere especial indicador de compressão.

Observe que sob esta versão de codificação de comprimento de fileira, uma seqüência repetida de precisamente três caracteres resulta que uma contagem de repetição com zero será inserida no fluxo de dados comprimidos. Além disso, a contagem máxima de repetição permitida pelo MNP é limitada a 250 decimal. Assim, esta parte da compressão de dados Classe 5 pode resultar na expansão de dados, quando várias seqüências de três caracteres repetidos são encontradas no fluxo original de dados. Além disso, a versão MNP de codificação de comprimento de fileira requer quatro caracteres, enquanto que o método de codificação de comprimento de fileira, descrito anteriormente, usa três caracteres. Por último, a codificação convencional de comprimento de fileira pode ser implementada para comprimir até 256 caracteres que se repetem, enquanto que o MNP Classe 5 limita-se a 250. Estas desvantagens devem ser ponderadas, juntamente com o fato de a versão MNP Classe 5 de codificação de comprimento de fileira eliminar a necessidade de uso de um caractere especial de indicação de compressão, já que os três caracteres repetidos indicam que ocorreu a compressão e, também, identificam o caractere que foi comprimido.

Eficiência

A eficiência da codificação de comprimento de fileira depende do número de ocorrências de caracteres repetidos nos dados a serem comprimidos, do comprimento médio do caractere repetido e da técnica empregada para executar a compressão. Na tabela 3.3, o leitor encontrará uma listagem com os resultados da execução de um programa de computador, escrito para computar a razão de compressão global, baseada em um número variado de ocorrência de caracteres repetidos em uma cadeia de 1000 caracteres de dados. Aqui, o número de ocorrência de caracteres repetidos variou de 10 a 50, enquanto o comprimento médio dos caracteres repetidos variou de 4 a 10. Presumiu-se que foram usados três caracteres para o formato de dados comprimidos. As razões de compressão computadas, listadas na tabela 3.3, vão de 1,0101 (baixo) a 1,5384 (alto). A tabela é uma representação sintética, devido o texto real ser muito amplo. Já que esta tabela cobre a maioria das ocorrências mais comuns para compressão, ela fornece uma referência tabular prática aos leitores para determinar o efeito da codificação de comprimento de fileira.

Tabela 3.3 - A eficiência da codificação de comprimento de fileira com base na cadeia original de dados de 1000 caracteres

Número de ocorrências de caracteres repetidos	Comprimento médio de caracteres repetidos	Relação de Compressão
10	4	1.010
10	5	1.020
10	6	1.031
10	7	1.042
10	8	1.053
10	9	1.064
10	10	1.075
20	4	1.020
20	5	1.042
20	6	1.064
20	7	1.087
20	8	1.111
20	9	1.136
20	10	1.163
30	4	1.031
30	5	1.064
30	6	1.099
30	7	1.136
30	8	1.176
30	9	1.220
30	10	1.266
40	4	1.042
40	5	1.087
40	6	1.136
40	7	1.190
40	8	1.250
40	9	1.316
40	10	1.384
50	4	1.053
50	5	1.111
50	6	1.176
50	7	1.250
50	8	1.333
50	9	1.429
50	10	1.538

O segundo fator que afeta a eficiência da codificação de comprimento de fileira é o próprio mecanismo usado para implementar esta técnica de compressão de dados. A tabela 3.4 indica a razão de compressão para os três métodos comuns pelos quais a codificação de comprimento de fileira é implementada. A primeira coluna mostra a razão de compressão quando um caractere especial de compressão é seguido pelo caractere comprimido e pela contagem, formando uma seqüência de três caracteres para indicar a ocorrência da compressão de comprimento de fileira. A segunda coluna mostra a razão de compressão resultante do uso de um conjunto de códigos estendidos (ECS). Nesta situação, os caracteres SO e SI devem ser adicionados ao início (SO) e ao término (SI) da seqüência de três caracteres, descrita previamente, resultando

Tabela 3.4 - A eficiência dos métodos de codificação de comprimento de fileira

Número de caracteres repetidos	Razão de compressão		Comprimento de fileira MNP
	Comprimento de fileira	Comprimento de fileira ECS	
3	1.0000	0.6000	0.7500
4	1.3333	0.8000	1.0000
5	1.6667	1.0000	1.2500
6	2.0000	1.2000	1.5000
7	2.3333	1.4000	1.7500
8	2.6667	1.6000	2.0000
9	3.0000	1.8000	2.2500
10	3.3333	2.0000	2.5000
11	3.6667	2.2000	2.7500
12	4.0000	2.4000	3.0000
13	4.3333	2.6000	3.2500
14	4.6667	2.8000	3.5000
15	5.0000	3.0000	3.7500
16	5.3333	3.2000	4.0000
17	5.6667	3.4000	4.2500
18	6.0000	3.6000	4.5000
19	6.3333	3.8000	4.7500
20	6.6667	4.0000	5.0000

em cinco caracteres sendo exigidos para codificar uma cadeia de caracteres repetidos. A terceira coluna, que é chamada de Comprimento de Fileira MNP, mostra a razão de compressão quando o método de codificação de comprimento de fileira MNP Classe 5 é

usado. Sob esta técnica, as cadeias de três ou mais caracteres repetidos são codificadas como quatro caracteres.

Como mostrado na tabela 3.4, o método convencional de codificação de comprimento de fileira fornece a maior razão de compressão. Isto ocorre porque ele requer o menor número de caracteres para ser implementado. O MNP Classe 5 é o próximo método mais eficiente, uma vez que ele requer quatro caracteres. O uso de um conjunto de caracteres estendidos é o método menos eficiente para implementar a codificação de comprimento de fileira.

Exemplos de programação

Para ilustrar a programação necessária para implementar a codificação de comprimento de fileira e outras técnicas de compressão, desenvolvemos, neste livro, diversos exemplos de codificação na linguagem BASIC. Cada um destes pequenos segmentos de programa foram escritos na versão BÁSICA da linguagem de programação BASIC, que roda no IBM PC e compatíveis. Embora uma linguagem de programação diferente, como, por exemplo: assembler, Pascal ou C, fosse mais eficiente, usamos a BÁSICA por ser amplamente aceita como uma boa linguagem de programação e por ser possível usar a linguagem como uma ferramenta de aprendizado para a maioria dos leitores. Para uma perfeita utilização dos exemplos de programação apresentados neste livro, sugerimos que você use ou um compilador BASIC para acelerar a execução dos exemplos, ou reescreva cada segmento de programa, usando uma linguagem de programação mais adequada.

O IBM PC usa, em sua arquitetura interna, o código estendido de caracteres ASCII de oito bits. Este código estendido de caracteres atribui caracteres distintos para valores ASCII de 128 a 255. Já que todos os caracteres com valores ASCII de 0 a 255 são definidos e podem ocorrer ao se transmitir dados de um IBM PC para outro sistema de computação, você pode normalmente empregar os caracteres ASCII SO (shift out) e SI (shift in) no desenvolvimento do módulo de compressão que irá manipular dados ASCII, toda vez que haja a probabilidade de ocorrência de um caractere do conjunto de caracteres.

O caractere SO é usado para sair fora do conjunto de caracteres ASCII, resultando na possibilidade de o usuário redefinir cada caractere do conjunto de caracteres. De forma semelhante, o caractere SI é usado para retornar ao conjunto de caracteres ASCII definido.

Ao usar os caracteres SO e SI do ASCII, você obtém um conjunto de 128 ou de 256 caracteres novos, dependendo se você estiver utilizando um sistema que use um código ASCII ou estendidos de 8 bits e de 7 bits. Este novo conjunto de caracteres pode, então, ser usado para representar os caracteres de indicação de compressão.

A figura 3.15 ilustra a utilização dos caracteres ASCII SO e SI para obter um novo conjunto de caracteres, onde o valor ASCII 082 (ASCII R convencional) é usado para indicar a codificação de comprimento de fileira. Neste exemplo, supomos que uma cadeia de seis Xs é seguida por uma cadeia de sete Ys. Já que o valor 082 ASCII em um código ASCII, recentemente definido, será usado para indicar a compressão de comprimento de fileira, você deve primeiro pular (SO) para o novo código, enviar o caractere de indicação de compressão (R) e, então, voltar (SI) para o código ASCII normal, de maneira a transmitir o caractere que foi comprimido (X) e a quantidade de caracteres X comprimidos (6). Devido à exigência de sair fora do conjunto de caracteres para emitir o caractere de indicação de compressão e, então, voltar para o conjunto de caracteres ASCII normal, são necessários dois caracteres adicionais para representar uma cadeia codificada de comprimento de fileira. Além desta técnica necessitar de dois caracteres extras, o uso do código shift out 14 é usado para ligar o modo de largura dupla definido pela maioria das impressoras matriciais, enquanto o código shift in 15 é usado para ligar o modo de caractere comprimido definido pela maioria das impressoras, fazendo com que a ilustração gráfica desta técnica se torne, no melhor dos casos, enfadonha.

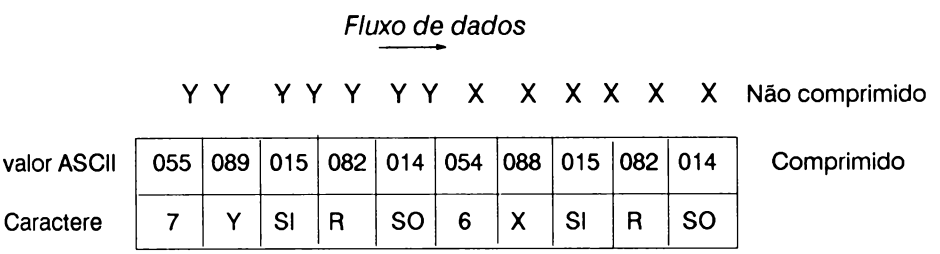


Figura 3.15 - Usando os caracteres SO e SI. O uso do SO e do SI proporciona um novo conjunto de caracteres que pode ser usado para indicar diferentes técnicas de compressão. Neste exemplo, o valor ASCII 082 é usado para indicar a codificação de comprimento de fileira.

Com base na informação anterior, foi determinado que, para os exemplos apresentados neste livro, o uso de um único caractere do conjunto de caracteres ASCII serviria, suficientemente, como um flag de compressão, além de realmente economizar dois caracteres na representação da compressão de dados com base no uso da codificação de comprimento de fileira.

Programa de compressão

A figura 3.16 contém a listagem de um programa em BASIC que ilustra a codificação exigida para executar a compressão de comprimento de fileira.

Para facilitar a consulta aos programas BASIC, usados para ilustrar as instruções necessárias para comprimir e descomprimir dados, uma simples convenção de nome foi usada em todo este livro. Cada nome de arquivo de programa finaliza com a letra C ou D. O C refere-se ao programa que comprime ou codifica os dados, enquanto o D ao programa que descomprime ou decodifica os dados. Assim, o programa chamado RUNLENC.BAS, na figura 3.16, ilustra as instruções exigidas para comprimir ou codificar dados, usando a compressão de comprimento de fileira. A extensão .BAS indica que o arquivo é um programa em linguagem BASIC. De forma semelhante, quaisquer arquivos de dados analisados terão um nome de arquivo que indique a técnica de compressão, que será aplicado ao arquivo, onde sua extensão será .DAT. Assim, uma referência ao arquivo RUNLEN.DAT indica que o arquivo de dados será comprimido por um programa de compressão de comprimento de fileira (run-length).

No programa RUNLENC.BAS, o valor ASCII 125 (fecha chave }) foi usado como o caractere de indicação de compressão, que foi, então, seguido pelo caractere ASCII sendo comprimido e pela contagem repetitiva em notação decimal. Assim, qualquer cadeia com mais de três caracteres que se repetem estaria sujeita à compressão.

Diversas declarações na listagem do programa da figura 3.16 levantam discussão entre os leitores não familiarizados com a versão BÁSICA da linguagem de programação BASIC para o IBM PC. A declaração LINE INPUT na linha 130, faz como que uma linha inteira de um arquivo seqüencial seja lida e atribuída à variável de cadeia X$. Na linha 140, é determinado o comprimento da cadeia que representa uma linha no arquivo de dados. O comprimento da cadeia é, então, usado no loop FOR NEXT delimitado pelas linhas 240 e 310 para processar a cadeia de caracteres que se repetem.

As funções MID$ nas linhas 250 e 260, extraem os caracteres lésimo e lésimo + 1 da cadeia e compara estes caracteres um com o outro. Quando forem iguais, o caractere repetido é salvo (linha 330) e a contagem de caracteres que se repetem é incrementada (linha 340). Quando a cadeia de repetição de caracteres for quebrada, a linha 260 se torna FALSE (falsa) e ocorre uma comparação da contagem de repetição (linhas 270 e 280). Quando a contagem ultrapassa três, os dados (linha 270) são comprimidos pelas instruções contidas nas linhas 360 a 400. Se a contagem se igualar a três, nenhuma vantagem será obtida com a compressão de comprimento de fileira e a rotina delimitada pelas linhas 420 e 450 simplesmente adiciona os caracteres de entrada ao buffer de saída. Quando os caracteres lésimo e lésimo + 1 não forem iguais, o caractere lésimo do buffer de entrada é simplesmente passado para o buffer de saída (linha 290). As linhas de 900 a 9999 não são realmente parte do processo de codificação de comprimento de fileira e estão somente incluídas para facilitar as operações de arquivo e a comparação dos buffers de entrada e de saída, para obter a medida da eficiência desta técnica quando aplicada a um arquivo de dados, que contenha uma variedade de cadeias de dados repetidos.

```
10 REM RUNLENC.BAS PROGRAM
20 DIM O$(132)
30 WIDTH 80:CLS
40 '**********MAIN ROUTINE**********************
50 '* THIS ROUTINE READS RECORDS FROM AN ASCII *
60 '* FILE INTO A STRING CALLED X$ WHICH IS    *
70 '* THEN PASSED TO SUBROUTINES FOR COMPRESSION
80 '*********************************************
90 PRINT "ENTER ASCII FILENAME. EG, RUNLEN.DAT"
100 INPUT F$: OPEN F$ FOR INPUT AS #2
105 OPEN "RUNLENC.DAT" FOR OUTPUT AS #3
110 PRINT "PATIENCE - INPUT PROCESSING"
120 IF EOF(2) THEN GOTO 9000
130 LINE INPUT #2, X$
140 N= LEN(X$)
150 GOSUB 180
160 GOSUB 900
170 GOTO 120
180 '*****RUN LENGTH ENCODING SUBROUTINE********
190 '* THIS ROUTINE PROCESSES RECORDS FROM X$  *
200 '* AND COMPRESSES OUT REPETITIVE CHARACTERS*
210 '* USING O$ AS THE OUTPUT BUFFER.          *
220 '*********************************************
230 K=1:J=1                          'RESET INDICES
240 FOR I= 1 TO N                    'STEP THRU RECORD
```

Figura 3.16 - Listagem do programa RUNLENC.BAS.

```
250 A$= MID$(X$,I,1)              'EXTRACT A CHAR
260 IF A$= MID$(X$,I+1,1) THEN 330 'SAME AS NEXT?
270 IF K>3 THEN 360               'COMPRESS
280 IF K=3 THEN 420               'DON'T COMPRESS
290 O$(J)=A$                      'STUFF IN OUTPUT BUFFER
300 J=J+1                         'BUMP BUFFER INDEX
310 NEXT I                        'GO BACK FOR MORE
320 RETURN                        'END OF STRING
330 B$=A$                         'SAVE REPEATED CHAR
340 K=K+1                         'BUMP COUNT
350 GOTO 310                      'KEEP LOOKING
355 '*****************************************************
360 'INSERT COMPRESSION NOTATION IN OUTPUT BUFFER
365 '*****************************************************
370 O$(J)=CHR$(125)              'SET FLAG FOR RUN-LENGTH
380 O$(J+1)=B$                    'INSERT REPEATED CHAR
390 O$(J+2)=CHR$(K)               'INSERT COUNT
400 J=J+3:K=1                     'RESET INDEX
410 GOTO 310
420 O$(J)=B$                      'STUFF 1ST REPEAT CHAR
430 O$(J+1)=B$                    'STUFF 2ND REPEAT CHAR
440 J=J+2:K=1                     'RESET INDEX
450 GOTO 310
900 '*****TALLY THE COMPRESSION COUNT & WRITE BUFFER******
910 '* DISPLAY BEFORE & AFTER RESULTS OF COMPRESSION     *
920 '* AND SHOW THE NET RESULTS OBTAINED BY EACH METHOD  *
930 '*****************************************************
931 N1=N1+N                       'TALLY INPUT CHAR COUNT
932 T=N-J+1                       'NET DIFFERENCE IN BUFFERS
936 T1=T1+T                       'SAVE COUNT FOR SUMMARY
940 FOR I= 1 TO J-1
950 PRINT #3, O$(I);
960 NEXT I
965 PRINT #3, ""
970 RETURN
9000 CLOSE: OPEN F$ FOR INPUT AS #2
9010 PRINT "FILE ";F$;" BEFORE COMPRESSION:"
9020 LINE INPUT #2,X$
9030 IF EOF(2) THEN 9060
9040 PRINT X$
9050 GOTO 9020
9060 PRINT X$:OPEN "RUNLENC.DAT" FOR INPUT AS #3
9070 PRINT "FILE ";F$;" AFTER COMPRESSION:"
9080 LINE INPUT #3,O$
9090 IF EOF(3) THEN 9998
9100 PRINT O$
9110 GOTO 9080
9998 PRINT O$:PRINT T1;" TOTAL CHARACTERS ELIMINATED FROM ";
9999 PRINT N1;"OR ";INT((T1/N1)*100);"%":CLOSE:END
```

Figura 3.16 - Continuação

A figura 3.17 ilustra uma execução de exemplo do programa *RUNLENC.BAS*, usando um arquivo ASCII chamado *RUNLENC.DAT* como entrada de dados do programa. Observe que *RUNLENC.BAS* foi propositadamente escrito de maneira a primeiro listar, o conteúdo do arquivo antes de sua compressão; isto está ilustrado nas linhas de 1 a 8 no topo da figura 3.17. A seguir, o programa lista o arquivo depois que os dados em seu conteúdo foram comprimidos, com base na aplicação da codificação de comprimento de fileira.

Deve-se observar que os valores decimais da cadeia que estavam abaixo do ASCII 32 foram propositadamente omitidos do arquivo de teste, já que estes provocariam retornos de linha, avanço de linha e outros caracteres não imprimíveis, todos estes indesejáveis e que tornariam incompreensível a ilustração desta técnica de compressão. Eles, contudo, seriam, bem apropriados em aplicações normais de compressão descompressão de cadeia.

```
ENTER ASCII FILENAME. EG, RUNLEN.DAT
? RUNLEN.DAT
PATIENCE - INPUT PROCESSING
FILE RUNLEN.DAT BEFORE COMPRESSION:
1 BEGIN**********************************
2 RRRRRRRRRRRRRRRRRRRRRRRRRRRRRR
3 EEEEEEEEEEEEEEEEEEEEEEEEEEEEEE
4 PPPPPPPPPPPPPPPPPPPPPPPPPPPPPP
5 EEEEEEEEEEEEEEEEEEEEEEEEEEEEEE
6 AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
7 TTTTTTTTTTTTTTTTTTTTTTTTTTTTTT
8 **********************************END
FILE RUNLEN.DAT AFTER COMPRESSION:
1 BEGIN}*#
2 }R!
3 }E"
4 }P#
5 }E$
6 }A%
7 }T&
8 }*%END
 261  TOTAL CHARACTERS ELIMINATED FROM  309 OR  84 %
Ok
```

Figura 3.17 - Execução de exemplo do RUNLENC.BAS.

Modificações a se considerar

O caractere ASCII 125 foi usado como um caractere de indicação de compressão, devido a sua representação ser o símbolo de "fecha", para a maioria das impressoras. Normalmente, se sua fonte de dados não incluir caracteres acima do ASCII 127, então um caractere do conjunto estendido de caracteres ASCII, tais como ASCII 129, ou outro acima do ASCII 127, deve ser usado para representar a ocorrência da codificação de comprimento de fileira. No caso do exemplo anterior, o ASCII 129 foi propositadamente excluído, porque ele aparece no monitor, como o caractere ii e, em algumas impressoras, como o caractere de representação de libra, enquanto outras impressoras simplesmente ignorarão os caracteres acima de ASCII 127. Para imprimir, corretamente, os caracteres acima do ASCII 127, usando um IBM PC, é necessário que você tenha uma impressora capaz de imprimir o conjunto estendido de caracteres ASCII. Além disso, um programa especial do DOS (sistema operacional em disco) chamado GRAFTABL, que está disponível a partir da versão 3.0 do sistema operacional DOS, deve ser carregado no computador antes de imprimir os dados. Devido a este fato, o caractere ASCII 125 foi usado como o caractere de indicação de compressão para efeito ilustrativo.

Se um caractere acima do ASCII 127 for usado para indicar a ocorrência da compressão e este caractere naturalmente aparece em seus dados, isto resultará em uma falsa indicação de compressão. Para evitar que um dispositivo de recepção interprete erroneamente o caractere como sendo de indicação de ocorrência de compressão de comprimento de fileira, o programa pode ser modificado para enviar dois destes caracteres sempre que um caractere de indicação de compressão aparecer, naturalmente, no fluxo de dados. Então, no dispositivo de recepção, o programa de descompressão examinaria primeiro todos os caracteres para ver se apareceu um caractere de indicação de compressão, contudo, quando encontrados não significaria imediatamente, que ocorreu a codificação de comprimento de fileira. O programa examinaria, então, o próximo caractere para verificar se aquele caractere é também um caractere de indicação de compressão. Se for, este servirá como um indicador que um caractere de indicação de compressão ocorreu naturalmente nos dados, resultando na remoção do segundo caractere de indicação de compressão pelo receptor. A técnica acima descrita, é mais chamada formalmente de método de inserção e deleção de caractere. Este método foi previamente descrito no capítulo 2 e os leitores devem consultá-lo para obter informações

adicionais a respeito do uso desta técnica, para evitar uma falsa indicação de ocorrência de compressão de dados.

Descompressão

Na figura 3.18, o leitor encontrará a listagem do programa RUNLEND.BAS, que é desenvolvido para descomprimir os dados previamente comprimidos pelo programa RUNLENC.BAS. Para a melhor compreensão possível, as variáveis e os módulos do programa foram mantidos iguais nos programas de compressão e descompressão apresentados neste livro, facilitando, assim, sua utilização e explicação.

Semelhante ao programa de compressão examinado anteriormente, este programa processa os dados linha a linha. A declaração LINE INPUT na linha 130, lê uma linha de dados do arquivo usado como entrada. A seguir, na linha 140, o comprimento da linha é determinado.

A sub-rotina delimitada pelas linhas 180 e 320 é, então, chamada. Nesta sub-rotina, a cadeia que representa uma linha do arquivo de entrada é examinada caractere por caractere, usando a função MID$ na linha 250, para extrair, da cadeia, um caractere por vez. Na linha 260, cada caractere extraído é comparado como o caractere de valor 125, que é o caractere de "fecha chave", para determinar se ocorreu um caractere de indicação de compressão. Em caso positivo, ocorre um salto para a linha 360, onde a contagem de repetição e o caractere repetido são extraídos da cadeia nas linhas 370 e 380. A seguir, um índice é obtido com base no valor numérico de K$, usando a função ASC na linha 390. Em seguida vem um loop FOR-NEXT delimitado pelas linhas 400 a 420, que coloca o caractere repetido no buffer de saída, tantas vezes quanto o número do caractere de contagem indicar. Então os índices J e I são incrementados e o programa salta de volta para a linha 250.

```
10 REM RUNLEND.BAS PROGRAM
20 DIM O$(132)
30 WIDTH 80:CLS
40 '**********MAIN ROUTINE**********************
50 '* THIS ROUTINE READS RECORDS FROM AN ASCII *
60 '* FILE INTO A STRING CALLED X$ WHICH IS    *
70 '* THEN PASSED TO DECOMPRESSION SUBROUTINE  *
80 '*********************************************
90 PRINT "ENTER ASCII FILENAME. EG, RUNLENC.DAT"
100 INPUT F$: OPEN F$ FOR INPUT AS #2
105 OPEN "RUNLEND.DAT" FOR OUTPUT AS #3
110 PRINT "PATIENCE - INPUT PROCESSING"
120 IF EOF(2) THEN GOTO 9000
130 LINE INPUT #2, X$
140 N= LEN(X$)
150 GOSUB 180
160 GOSUB 900
170 GOTO 120
180 '*****RUN LENGTH DECODING SUBROUTINE********
190 '* THIS ROUTINE PROCESSES RECORDS FROM X$  *
200 '* AND DECOMPRESSES RUN-ENCODED CHARACTERS *
210 '* USING O$ AS THE OUTPUT BUFFER.          *
220 '*********************************************
230 K=1:J=1                          'RESET INDICES
240 FOR I= 1 TO N                    'STEP THRU RECORD
250 A$= MID$(X$,I,1)                 'EXTRACT A CHAR
260 IF A$= CHR$(125) THEN 360        'COMPRESSION FLAG?
290 O$(J)=A$                         'STUFF IN OUTPUT BUFFER
300 J=J+1                            'BUMP BUFFER INDEX
310 NEXT I                           'GO BACK FOR MORE
320 RETURN                           'END OF STRING
355 '*****************************************************
360 'DECODE COMPRESSION NOTATION TO OUTPUT BUFFER
365 '*****************************************************
370 K$= MID$(X$,I+2,1)               'GET REPEAT COUNT
380 A$= MID$(X$,I+1,1)               'GET REPEAT CHAR
390 K= ASC(K$)                       'SET UP INDEX
400 FOR L= J TO J+K                  'SET OUTPUT LOOP
410 O$(L)= A$                        'STUFF REPEAT CHAR
420 NEXT L                           'KEEP GOING
430 J= L                             'BUMP OUTPUT INDEX
440 I= I+3                           'BUMP INPUT INDEX
450 GOTO 250                         'DONE
```

Figura 3.18 - Listagem do programa RUNLEND.BAS.

```
900 '*****TALLY THE DECOMPRESSION COUNT & WRITE BUFFER****
910 '* DISPLAY BEFORE & AFTER RESULTS OF DECOMPRESSION    *
920 '* AND SHOW THE NET RESULTS OBTAINED BY EACH METHOD   *
930 '*************************************************
931 N1=N1+N                          'TALLY INPUT CHAR COUNT
932 T=N-J+1                          'NET DIFFERENCE IN BUFFERS
936 T1=T1-T                          'SAVE COUNT FOR SUMMARY
940 FOR I= 1 TO J-1
950 PRINT #3, O$(I);
960 NEXT I
965 PRINT #3, ""
970 RETURN
9000 CLOSE: OPEN F$ FOR INPUT AS #2
9010 PRINT "FILE ";F$;" BEFORE DECOMPRESSION:"
9020 LINE INPUT #2,X$
9030 IF EOF(2) THEN 9060
9040 PRINT X$
9050 GOTO 9020
9060 PRINT X$:OPEN "BYTED.DAT" FOR INPUT AS #3
9070 PRINT "FILE ";F$;" AFTER DECOMPRESSION:"
9080 LINE INPUT #3,O$
9090 IF EOF(3) THEN 9998
9100 PRINT O$
9110 GOTO 9080
9998 PRINT O$:PRINT T1;" TOTAL CHARACTERS INSERTED"
9999 CLOSE:END
```

Figura 3.18 - (continuação)

Se não ocorrer um caractere de indicação de compressão nos dados, a linha 290 será executada. Esta linha faz com que o caractere, extraído da cadeia, seja colocado diretamente no buffer de saída. A seguir, o índice J é incrementado em 1 na linha 300 e o limite do loop original FOR-NEXT verifica se o fim do loop foi alcançado na linha 310.

As declarações da linha 900 ao fim do programa foram incluídas para atualizar a contagem de descompressão e mostrar os resultados antes e depois do programa. Assim, esta parte do programa foi incluída somente com propósitos ilustrativos.

A figura 3.19 contém uma execução de exemplo do programa RUNLEND.BAS. O leitor observará que o arquivo de dados RUNLENC.DAT foi usado como entrada de dados do programa. Este arquivo de dados foi criado pela execução do programa RUNLENC.BAS e as oito linhas superiores numeradas na Figura 3.19, correspondem às oito linhas inferiores numeradas na Figura 3.17. Já que o programa de descompressão restitui os dados comprimidos em sua forma original, as oito linhas numeradas na parte inferior da figura 3.18, são exatamente as mesmas oito linhas numeradas na parte superior da figura 3.17.

```
ENTER ASCII FILENAME. EG, RUNLENC.DAT
? RUNLENC.DAT
PATIENCE - INPUT PROCESSING
FILE RUNLENC.DAT BEFORE DECOMPRESSION:
1 BEGIN}*#
2 }R!
3 }E"
4 }P#
5 }E$
6 }A%
7 }T&
8 }*%END
FILE RUNLENC.DAT AFTER DECOMPRESSION:
1 BEGIN**************************************
2 RRRRRRRRRRRRRRRRRRRRRRRRRRRRRR
3 EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE
4 PPPPPPPPPPPPPPPPPPPPPPPPPPPPPPP
5 EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE
6 AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
7 TTTTTTTTTTTTTTTTTTTTTTTTTTTTTTT
8 ****************************************END
 276  TOTAL CHARACTERS INSERTED
Ok
```

Figura 3.19 - Execução de exemplo do programa RUNLEND.BAS.

3.4 COMPACTAÇÃO DE MEIO BYTE

Esta técnica de compressão de dados pode ser vista como uma derivação do processo de mapeamento de bit. Ela pode ser usada com sucesso sob diversas condições de estrutura de dados; contudo, diferente de uma versão da técnica de mapeamento de bit, esta nunca resultará em uma razão de compressão menor que o unitário.

A compactação de meio byte, como foi originalmente desenvolvida, se beneficia da estrutura de certos caracteres em um conjunto de caracteres. Esta técnica é efetiva quando uma parte do padrão de bit usado para representar aqueles caracteres, se torna repetitiva. Como um exemplo deste tipo de situação, considere o conjunto de caractere EBCDIC, onde as quatro primeiras posições de bit, usadas para representar numéricos, são todas ativadas como o binário 1, como apresentado na tabela 3.5

Se uma cadeia de dados não comprimidos contiver caracteres codificados em EBCDIC de 8 níveis, então a codificação de comprimento de fileira não permitirá a compressão de uma seqüência de dígitos que não se repete por um caractere. Contudo, já que os primeiros quatro bits se repetem, a compressão poderia ser realizada se você compactasse dois numéricos em um só caractere. De forma semelhante a diversas versões de codificação de comprimento de fileira, um caractere especial é necessário para indicar que ocorreu a compactação de meio byte. Novamente, da mesma forma as versões de codificação de comprimento de fileira, que usam um caractere especial de indicação de compressão, este caractere deve ser escolhido dentre os caracteres não designados do conjunto de caracteres.

Tabela 3.5 - Representação numérica EBCDIC. Quando um byte de 8 bits é usado para conter valores numéricos codificados pelo conjunto de caracteres EBCDIC, as quatro primeiras posições são sempre ativadas em 1.

ESTRUTURA DE BIT	CARACTERE NUMÉRICO
1111	00000
1111	00011
1111	00102
1111	00113
1111	01004
1111	01015
1111	01106
1111	01117
1111	10008
1111	10019

Como alternativa, você pode considerar o uso da técnica de inserir e apagar ou o uso de um conjunto estendido de caracteres, obtido pelo uso do caractere shift out (SO). Contudo, já que a maioria das seqüências de numéricos são pequenas se comparadas com as fileiras de caracteres iguais, resultantes de cabeçalhos e espaços entre colunas, o uso de métodos alternados para obter um caractere especial de indicação de compressão pode reduzir, de forma significativa, o potencial de compressão de dados por meio da compactação de meio byte.

Aplicações Financeiras

Quando os caracteres de dados não têm uma estrutura de bit repetitiva, a compactação de meio byte pode ainda ser empregada, com sucesso, sob certas condições predefinidas. Um exemplo seria predefinir a ocorrência em série do cifrão, de todos os 10 numéricos, dos caracteres de vírgula, asterisco e ponto decimal como sendo adequados para a compressão pelo método da compactação de meio byte. Na tabela 3.6, encontra-se listada a estrutura de bit dos caracteres de dados ASCII, freqüentemente, usados para representações financeiras. Se a ocorrência de uma cadeia, que consista de qualquer dígito numérico, bem como uma vírgula, um ponto decimal, um cifrão e um asterisco for predefinida como adequada à compactação de meio byte, então a ocorrência destas cadeias, tais

como: '$123.456,78', '123.456' ou '$****123.456,78', podem ser comprimidas.

Tabela 3.6 - Representação ASCII de caracteres financeiros. Nesta representação de dados, o bit de paridade foi ignorado. Se a paridade de bit existir, ela pode ser retirada juntamente com os três primeiros bits mostrados antes da compactação dos últimos quatro bits em meio bytes

ESTRUTURA DE BIT		CARACTERE
011	0000	0
011	0001	1
011	0010	2
011	0011	3
011	0100	4
011	0101	5
011	0110	6
011	0111	7
011	1000	8
011	1001	9
010	0100	$
010	1100	,
010	1110	.
010	1010	*

Ao examinar a estrutura de bit dos caracteres financeiros, listados na tabela 3.6, observe que quando os três bits de nível mais alto são 'retirados', para se tornar possível a codificação dos quatro bits remanescentes em meio byte, esta estrutura de bit terá entradas duplicadas. Exemplificando, tanto o cifrão ($), quanto o dígito quatro (4) têm a mesma representação de bit 0100. Para atenuar este problema, é necessário que você redefina a representação de meio byte resultante para cada caractere. A tabela 3.7 lista uma redefinição possível da estrutura de meio byte. Observe que quando você redefine os meio bytes, você pode incluir até seis caracteres não decimais como representações possíveis de meio byte, para serem compactados quando encontrados em uma cadeia compressível. Na tabela 3.7, adicionamos os caracteres de mais (+) e de menos (-) nas definições binárias de meio byte porque estes caracteres ocorrem com freqüência em cadeias de caracteres

financeiros, tais como: $1.506.203- e 487,32+. Embora começamos, originalmente, nossa representação de dados de meio byte usando o conjunto de caracteres ASCII, você pode seguir o mesmo procedimento para codificar caracteres financeiros EBCDIC em uma seqüência de até 16 meio bytes. Desta forma, você pode decidir usar os mesmos seis caracteres não decimais listados na Tabela 3.7, ou você poderia escolher qualquer mistura de até seis caracteres não decimais que você espera encontrar em cadeias que representam informações financeiras.

Tabela 3.7 - Redefinição da composição binária de meio byte

ESTRUTURA DE BIT	CARACTERE
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	$
1011	,
1100	.
1101	*
1110	+
1111	-

Formato de codificação e eficiência da técnica

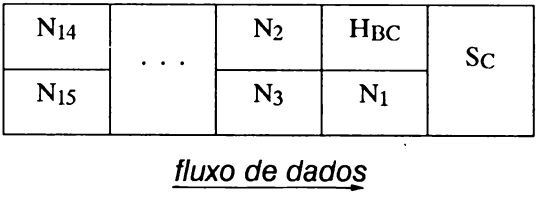
Para comprimir dados em meio bytes, pode-se levar em consideração diversos formatos de codificação. Cada formato fornece um certo nível de eficiência baseado na seqüência de caracteres encontrados na cadeia original de dados. Um formato típico encontra-se ilustrado na Figura 3.20. Usando este formato, pode-se comprimir até 15 caracteres numéricos em seqüência ou de dados predefinidos, que ocorrem seqüencialmente na cadeia. O limite de

15 caracteres é resultado do uso de um contador de meio byte com 4 bits para indicar o número de caracteres comprimidos.

Já que, por definição, o caractere especial de indicação de compressão indica que ocorreu a codificação de meio byte, você pode designar o valor zero no contador de meio byte para indicar a contagem de um. Assim, todos os quatro bits indicariam uma fileira de 16 caracteres comprimidos. Como alternativa, já que a codificação de meio byte somente é efetiva quando cinco ou mais caracteres compressíveis aparecem em seqüência, você pode designar o valor de zero na contagem de meio byte para indicar que cinco meio bytes estão codificados. Assim, quando o contador tiver o valor 15 (todos os quatro bits ativados em 1) indicaria que 20 meio bytes foram codificados.

Se, em vez de um contador de meio byte, um byte inteiro fosse usado para indicar a contagem de compactação de meio byte, até 2^8 (ou 255) numéricos podem ser compactados ou 256, se o contador começar em zero para indicar um caractere compactado. De forma semelhante, se o valor zero na contagem for usado para indicar uma cadeia de cinco meio bytes codificados, o contador

Contador de meio byte



Contador de byte inteiro

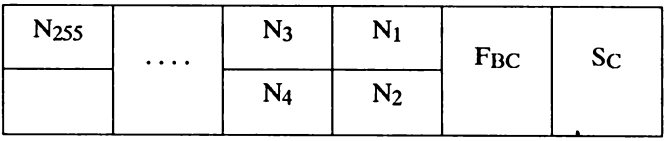


Figura 3.20 - O formato de codificação de meio byte.

S = Caractere especial indicando a codificação de meio byte.

H_{BC} = Contador de meio byte. Quatro bits são usados para indicar o número de numéricos que foram compactados.

Número ≤ 15, ou se um contador de 0 indicar quatro caracteres compactados então

Número ≤ 20.

F_{BC} = Contador de byte inteiro. Número ≤ 255.

N_1 to N_{255} = Até 255 numéricos compactados em 2 por caractere de 8 bits

de byte inteiro torna-se capaz de indicar até 260 caracteres compactados. Já que um meio byte extra é necessário para aumentar a capacidade do contador, devemos empregar o contador de byte inteiro somente quando se espera que o número médio de caracteres em seqüência exceda 15 (ou 20 quando o valor zero do contador indicar cinco caracteres compactados). De forma alternativa, você pode usar o formato de compressão de meio byte e de byte inteiro, e chavear entre os dois modos, dependendo do número de caracteres suscetíveis à compactação de meio byte, que são encontrados.

Para examinar a eficiência da compactação de meio byte, vamos primeiro explorar o padrão binário do fluxo de dados de exemplo e do fluxo resultante de dados comprimidos. Na figura 3.21, a seqüência numérica na parte superior da ilustração, consiste de sete caracteres de 8 bits, ou seja, 56 bits. Através do uso da técnica de compactação de meio byte, empregando um contador de meio byte (4 bits), o número de bits resultante na cadeia de dados comprimidos é reduzido para 40. Neste exemplo, o fluxo original de dados foi reduzido em 28 por cento (56 - 40)/56) para sete numéricos seqüenciais. Deve ser observado que seriam necessários também 40 bits para representar seis caracteres, encontrados seqüencialmente, suscetíveis à compactação de meio byte, caso a transmissão seja feita caractere por caractere. Assim, qualquer número par de caracteres encontrados seqüencialmente, adequados à compactação com um contador de meio byte, exige a transmissão de quatro bits nulos adicionais, no caso dos dados serem transferidos caractere por caractere.

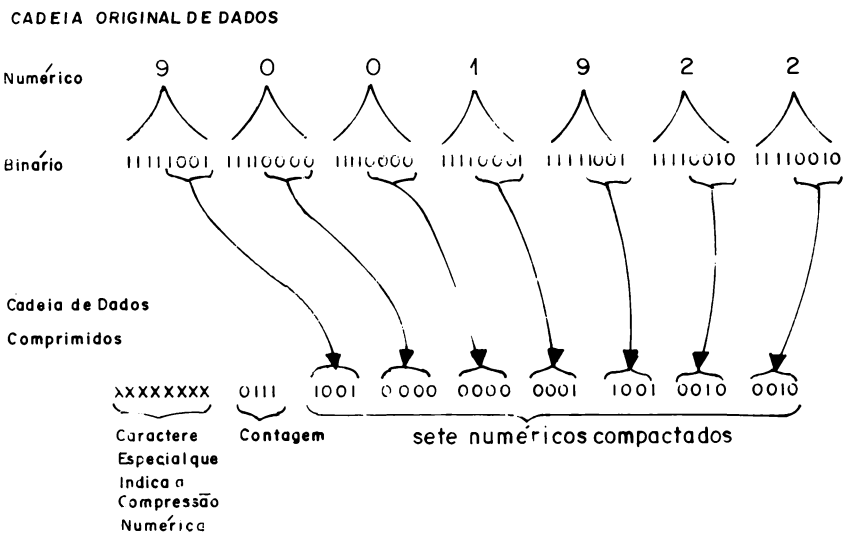


Figura 3.21 - Exemplo de codificação de meio byte. Para a transmissão de caracteres de 8 níveis, um múltiplo de 8 bits de dados comprimidos é transferido. Assim, um contador de meio byte com um número par de caracteres compactados exigirá quatro bits nulos à direita.

Na Tabela 3.8, o fluxo original de dados numéricos e o seu formato comprimido são comparados para o caso de usar um contador de quatro bits. Aqui, o número de numéricos contínuos variava de 1 a 15. Enquanto a quantidade de numéricos contínuos não for maior que quatro, o número de bits no fluxo original de dados, será menor ou igual ao número de bits do fluxo de dados comprimidos, sendo assim, a compactação de meio byte não deve ocorrer até que cinco ou mais numéricos ou caracteres predefinidos seqüenciais sejam encontrados no fluxo de dados.

O que foi dito acima, pode ser representado matematicamente como a seguir. Para uma seqüência de S caracteres compressíveis, S 4, o número de bits na cadeia não compressível é 8S. O número de bits na cadeia compressível, quando se usa um contador de meio byte, é

$$12 + 4 * \left[\frac{S}{2}\right]$$

resultando na razão de compressão de

$$\left(\frac{1}{8S} * \left\{12 + 4 * \left[\frac{S}{2}\right]\right\}\right)^{-1}$$

Tabela 3.8 - A eficiência da compressão de meio byte usando um contador de 4 bits

Número de caracteres seqüenciais compressíveis	Bits não comprimidos	Bits comprimidos	Porcentagem de redução de bits
1	81	6	N/A
2	16	24	N/A
3	24	24	N/A
4	32	32	0,00
5	40	32	20,00
6	48	40	16,66
7	56	40	28,00
8	64	48	25,00
9	72	48	33,33
10	80	56	30,00
11	88	56	36,36
12	96	64	33,33
13	104	64	38,46
14	112	72	35,71
15	120	72	40,00

Quando um contador de byte inteiro é usado, o número de bits na cadeia não comprimida permanece 8S. Contudo, o número de bits na cadeia comprimida agora se torna:

$$16 + 4 * \left[\frac{S}{2}\right]$$

resultando na razão de compressão de:

$$\left(\frac{1}{8S} * \left\{16 + 4 * \left[\frac{S}{2}\right]\right\}\right)^{-1}$$

Processo de codificação

O procedimento para comprimir caracteres numéricos, através da compactação de meio byte, está ilustrado em forma de fluxograma na Figura 3.22. Depois que o contador de caracteres numéricos iniciar em zero (1), um caractere é obtido da cadeia original de dados. Se o caractere for numérico (3), o contador é incrementado de um (4) e o próximo caractere da cadeia original de dados é examinado (2). Se a comparação de caractere (3) mostrar que o caractere não é numérico, o contador é comparado com quatro (5). Se o contador for menor ou igual a quatro, como previamente discutido, não se ganhará nada com a compressão e o contador será reiniciado em zero (1). Se o contador for maior que quatro (5), isto significa que nossa cadeia de numéricos seqüenciais terminou com um número suficiente de caracteres e que a compressão de meio byte será eficaz. Neste ponto, podemos ativar o formato de compressão (6). Embora o contador na Figura 3.22, não tenha um

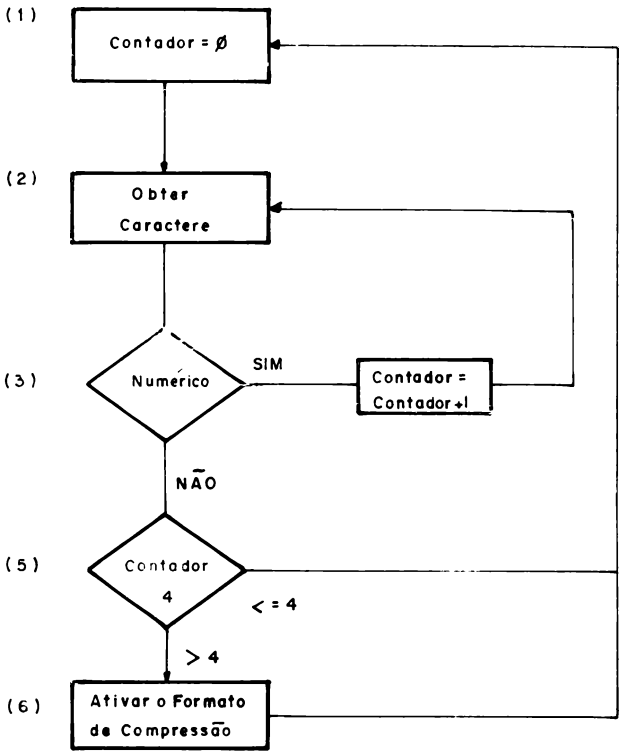


Figura 3.22 - Processo de compactação de meio byte para numéricos.

limite, se um contador de meio byte for empregado, o número máximo de caracteres que pode ser compactado é 15, a não ser que você modifique a designação de valores do contador de meio byte, de maneira que o valor zero, indique a compressão de cinco meio bytes, o valor 1 indique a compressão de seis meio bytes e etc. Assim, outra comparação de contador seria necessária entre os símbolos (5) e (6).

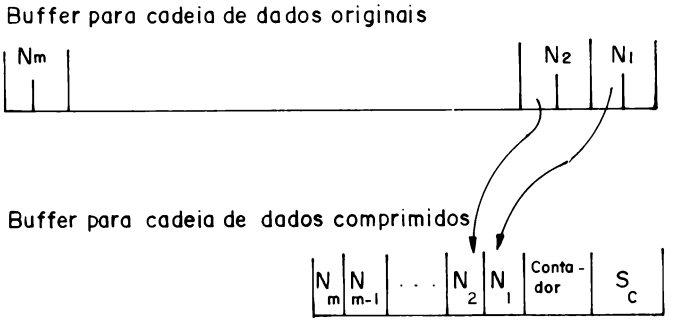
Se desejarmos comprimir as cadeias de caracteres predefinidos, encontradas seqüencialmente, incluindo cifrão, vírgula, ponto final, etc., deveríamos testar estes caracteres em vez de testar os numéricos.

Considerações do buffer

Quando um caractere inteiro ou caracteres múltiplos são usados como contador, as limitações da memória de buffer devem ser consideradas para se determinar o número máximo de caracteres seqüenciais que podem ser comprimidos; dois caracteres em um byte. Na Figura 3.23, estão ilustradas as considerações do buffer de compactação de meio-byte. À medida que o fluxo original de dados é examinado, os caracteres seqüenciais adequados à compactação de dois caracteres por byte, são colocados em um buffer, como ilustrado na parte superior da figura. Quando o contador for maior que quatro e o próximo caractere não for adequado à compactação, pode-se manipular os dados do primeiro buffer. A metade de cada caractere é, então, transferida para uma posição apropriada no buffer da cadeia de dados comprimidos, como ilustrado na parte inferior da Figura 3.23. Já que o caractere especial, usado para indicar a compressão de meio byte, e o caractere de contagem podem ser pré-colocados no buffer contíguo da cadeia de dados comprimidos, esta técnica de armazenamento temporário duplo é adequada se você desejar usar o dispositivo de acesso direto à memória (DMA) do computador ou do microprocessador usado na compressão. Através do uso do DMA, as transferências de dados podem ser realizadas independentemente do controle do programa e os blocos de dados são transferidos com base em word (bit paralelo) para e das partes da memória principal e dos dispositivos periféricos. Assim, uma vez que o buffer na parte inferior da Figura 3.23 estiver completo, pode ser estabelecida a transmissão, através do uso da transferência via DMA, enquanto isso, o computador limpa o buffer da cadeia original de dados e continua a processar o fluxo de dados de entrada. Como um exemplo de tamanho de buffer, considere o uso de um contador de 8 bits. Nesta situação, o buffer para o fluxo original de dados

teria de ser definido para conter até 256 caracteres, enquanto o buffer para o fluxo de dados comprimidos teria de conter até 130 caracteres, ou seja, 256 caracteres comprimidos compactados em dois por byte, além da contagem de caracteres e o caractere especial usado para indicar a compactação de meio byte.

A. Armazenamento em buffer duplo



B. Armazenamento em buffer único

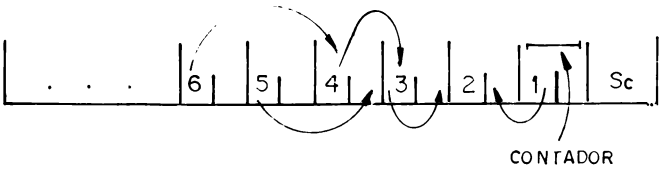


Figura 3.23 - Considerações do buffer para a codificação de meio byte. Embora o armazenamento em buffer único exija processamento adicional, ele elimina a necessidade de se manter um buffer separado para os dados comprimidos. Sc = Caractere especial de indicação de codificação de meio byte.

Embora a compactação de meio byte, ilustrada na parte superior da Figura 3.23, esteja utilizando o armazenamento em buffer duplo, na verdade, poderia ser usado o armazenamento em buffer único. Isto está ilustrado na parte inferior da figura. Nesta situação, os caracteres seqüenciais adequados à compactação são colocados, inicialmente, no buffer e uma vez que um caractere não compres-

sível é encontrado no fluxo original de dados e o contador excede quatro, os elementos de dados no buffer são manipulados conforme apresentado. Em contraste com o armazenamento em buffer duplo, esta técnica requer um processamento muito maior; contudo, ele elimina a necessidade de ter um buffer separado para os dados comprimidos. Para se determinar as necessidades totais de buffer, devemos examinar a inter-relação de todos os buffers de dados, como ilustrado na Figura 3.24. Neste exemplo, os dados a serem trabalhados são primeiro lidos e colocados no buffer de fluxo de dados, onde diversos tipos diferentes de processamento podem ser executados, dependendo da capacidade de processamento e da disponibilidade de área de memória do computador em uso. Este buffer de fluxo de dados pode ser pequeno ao ponto de conter somente um caractere, ou grande, podendo conter o bloco de dados usado na transmissão. O buffer pode ser examinado para detectar os caracteres compressíveis de diversos modos. Primeiro, pode-se fazer uma busca de qualquer caractere que seja adequado à compactação de meio byte; se nenhum caractere for encontrado, o buffer de fluxo de dados pode ser diretamente transferido ao buffer de fluxo de dados de saída. Outro método é examinar o buffer de fluxo de dados, caractere por caractere. Os caracteres não compressíveis podem, então, ser enviados ao buffer de fluxo de dados de saída, enquanto os caracteres compressíveis são transferidos para o buffer de dados originais. Se tiver menos de cinco caracteres compressíveis no buffer de dados originais, quando um caractere não compressível for encontrado no buffer de fluxo de dados, o conteúdo do buffer de dados originais é transferido para o buffer de fluxo de dados de saída. Se houver cinco ou mais caracteres no buffer de dados originais, quando um caractere não compressível é encontrado no buffer de fluxo de dados, a execução da compressão faz com que o conteúdo do buffer de dados originais seja transferido no formato comprimido ao buffer de dados comprimidos. Finalmente, o conteúdo do buffer de dados comprimidos é transferido para a posição apropriada no buffer de fluxo de dados de saída.

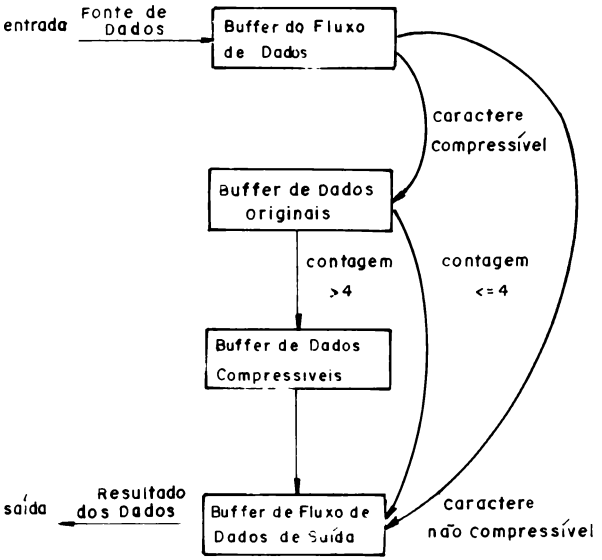


Figura 3.24 - As relações do buffer de dados. Para determinar as exigências totais de buffer, a inter-relação de todos os buffers de dados deve ser examinada.

Decodificação

A decodificação de dados comprimidos, de acordo com a técnica de compactação de meio byte, é um procedimento relativamente simples. A rotina de decodificação busca o caractere especial que é usado para indicar que ocorreu a compactação de meio byte. Uma vez que o caractere é encontrado, o próximo caractere ou o meio byte seguinte indicará a contagem do número de caracteres compactados. O caractere especial indicador de compressão pode ser usado para informar ao software de decodificação, se foi empregado um contador de byte inteiro ou de meio byte. Através do uso das técnicas de armazenamento em buffers, previamente discutidas, os caracteres compactados podem ser descompactados e o fluxo original de dados ser reconstruído.

Aplicação para a codificação

Já que cadeias de numéricos não repetidos não são compressíveis pela codificação de comprimento de fileira, o uso da compactação de meio byte pode ser muito vantajosa, quando os arquivos de dados contêm muitas seqüências numéricas. Se os caracteres predefinidos incluindo cifrão, vírgula, ponto decimal e asterisco são

acrescentados aos numéricos, a compactação de meio byte torna-se uma técnica muito apropriada para comprimir dados financeiros.

Exemplos de programação

Dois exemplos diferentes de codificação de dados por meio byte serão apresentados nesta seção. O primeiro conjunto de exemplos de programação utiliza somente os dígitos de 0 a 9 para a codificação de dados, seguindo a abordagem clássica de compactação de meio byte para dados numéricos. O segundo conjunto de exemplos de programação estende o número de caracteres que podem ser compactados dois por byte, incluindo caracteres como vírgula, ponto decimal, asterisco e cifrão, como discutido anteriormente nesta seção.

Codificação

Na figura 3.25, está listado o programa BYTEC.BAS em BASIC. Este programa contém as instruções exigidas para executar a codificação de meio byte simples, de cadeias que contêm cinco ou mais dígitos em seqüência. O caractere ASCII 126 foi usado neste exemplo de programação, para indicar a ocorrência da codificação de meio byte. Este caractere é impresso como til (~) na maioria das impressoras e, portanto, fornece uma boa indicação visual do uso de um caractere especial de indicação de compressão. Além disso, a ocorrência natural deste caractere em documentos, correspondência e programas escritos em linguagens diferentes de programação é insignificante. Isto faz com que o caractere til seja adequado ao método de inserção e deleção, usado para gerar um caractere especial de indicação de compressão.

Ao consultar a listagem da Figura 3,25, o arranjo 0$ é o buffer de saída no qual cada entrada de linha de um arquivo ASCII é colocada depois de ser primeiro analisada e comprimida, se for compressível de acordo com o esquema de codificação de meio byte. Cada linha do arquivo é lida na linha 130 e seu comprimento determinado na linha 140. A seguir, ocorre um salto para a sub-rotina, que começa na linha 180. Esta sub-rotina "caminha" através do registro obtido do arquivo em incrementos de duas posições de caracteres na linha 240. O registro é examinado em incrementos de duas posições de caracteres, já que as declarações nas linhas 250 e 260, comparam o caractere I e o caractere I + 1 na faixa entre 0 a 9, inclusive estes dígitos. Se o caractere lésimo ou lésimo + 1 estiver naquela faixa, ocorre um desvio para a linha 290.

```
10 REM BYTEC.BAS PROGRAM
20 DIM O$(132)
30 WIDTH 80:CLS
40 '**********MAIN ROUTINE**********************
50 '* THIS ROUTINE READS RECORDS FROM AN ASCII *
60 '* FILE INTO A STRING CALLED X$ WHICH IS    *
70 '* THEN PASSED TO SUBROUTINES FOR COMPRESSION
80 '*********************************************
90 PRINT "ENTER ASCII FILENAME. EG, BYTE.DAT"
100 INPUT F$: OPEN F$ FOR INPUT AS #2
105 OPEN "BYTEC.DAT" FOR OUTPUT AS #3
110 PRINT "PATIENCE - INPUT PROCESSING"
120 IF EOF(2) THEN GOTO 9000
130 LINE INPUT #2, X$
140 N= LEN(X$)
150 GOSUB 180
160 GOSUB 900
170 GOTO 120
180 '*****HALF-BYTE ENCODING SUBROUTINE*********
190 '* THIS ROUTINE PROCESSES RECORDS FROM X$  *
200 '* AND ENCODES NUMERIC STRINGS OF DATA INTO*
210 '* HALF-BYTE OR 4 BIT REPRESENTATION USING *
215 '* DOUBLE BUFFERING WITH O$ AS OUTPUT BUFF.*
220 '*********************************************
230 K=1:J=1                          'RESET INDICES
240 FOR I=1 TO N STEP 2              'STEP THRU RECORD
250 IF (MID$(X$,I,1)<"0") OR (MID$(X$,I,1)>"9") THEN 290
260 IF (MID$(X$,I+1,1)<"0") OR (MID$(X$,I+1,1)>"9") THEN 290
270 K=K+2                            'BOTH NUMERIC-BUMP COUNT
280 NEXT I                           'GO BACK FOR MORE
285 RETURN                           'END OF STRING
290 IF K > 4 THEN GOSUB 350          'ENOUGH TO ENCODE
300 IF K > 1 THEN GOSUB 440          'DON'T ENCODE
310 O$(J) = MID$(X$,I,1)             'OUTPUT 1ST CHAR.
320 O$(J+1) = MID$(X$,I+1,1)         'OUTPUT 2ND CHAR.
330 J=J+2:K=1                        'BUMP OUTPUT-RESET COUNT
340 GOTO 280                         'AND GO FOR MORE
345 '***** SUBROUTINE TO PERFORM HALF-BYTE ENCODING *****
350 O$(J)=CHR$(126)                  'FLAG FOR HALF-BYTE ENCODE
360 O$(J+1)=CHR$(K-1)                'INSERT LENGTH OF STRING
370 J=J+2                            'BUMP OUTPUT INDEX
380 FOR L=I-K+1  TO K STEP 2         'ENCODE 2 BYTES INTO 1
390 X= VAL(MID$(X$,L+1,1)):Y=VAL(MID$(X$,L,1))
400 O$(J)=CHR$(X+(Y*10))             'STUFF BYTE IN OUTPUT
410 J=J+1                            'BUMP OUTPUT INDEX
420 NEXT L                           'GO BACK FOR MORE
430 K=1:RETURN                       'RESET COUNT AND RETURN
435 '***** SUBROUTINE FOR STRING NOT WORTH ENCODING *****
440 FOR L=I-K+1 TO K                 'PICKUP SHORT STRING
450 O$(J)=MID$(X$,L,1)               'STUFF IN OUTPUT BUFFER
460 J=J+1                            'BUMP OUTPUT INDEX
470 NEXT L                           'GO BACK FOR MORE
480 K=1:RETURN                       'RESET COUNT AND RETURN
```

Figura 3.25 - Listagem do programa BYTEC.BAS.

```
900 '*****TALLY THE COMPRESSION COUNT & WRITE BUFFER******
910 '* DISPLAY BEFORE & AFTER RESULTS OF COMPRESSION     *
920 '* AND SHOW THE NET RESULTS OBTAINED BY EACH METHOD  *
930 '*************************************************
931 N1=N1+N                          'TALLY INPUT CHAR COUNT
932 T=N-J+1                          'NET DIFFERENCE IN BUFFERS
936 T1=T1+T                          'SAVE COUNT FOR SUMMARY
940 FOR I=1 TO J-1
950 PRINT #3, O$(I);
960 NEXT I
965 PRINT #3, ""
970 RETURN
1000 PRINT
1020 RETURN
9000 CLOSE: OPEN F$ FOR INPUT AS #2
9010 PRINT "FILE ";F$;" BEFORE COMPRESSION:"
9020 LINE INPUT #2,X$
9030 IF EOF(2) THEN 9060
9040 PRINT X$
9050 GOTO 9020
9060 PRINT X$:OPEN "BYTEC.DAT" FOR INPUT AS #3
9070 PRINT "FILE ";F$;" AFTER COMPRESSION:"
9080 LINE INPUT #3,O$
9090 IF EOF(3) THEN 9998
9100 PRINT O$
9110 GOTO 9080
9998 PRINT O$:PRINT T1;" TOTAL CHARACTERS ELIMINATED FROM ";
9999 PRINT N1;"OR ";INT((T1/N1)*100);"%":CLOSE:END
```

Figura 3.25 - continuação.

Para estender a codificação de meio byte aos caracteres $,. e * você poderia incluí-los nas comparações que ocorrem nas linhas 250 e 260. Isto seria chato e lento, devido ao tempo necessário para executar um grupo de funções MID colocadas juntas, através de muitos operadores OR. Uma solução mais rápida e mais elegante poderia ser obtida pela criação de um arranjo unidimensional, contendo os caracteres a serem codificados pela compressão de meio byte. Como exemplo, as seguintes declarações em BASIC iniciariam o arranjo HBYTE, de maneira que cada um dos 14 elementos conteria um dos caracteres que seriam adequados à compressão de meio byte:

```
DIM HBYTE (14)
FOR I = I TO 14
READ HBYTE (I)
NEXT I
DATA "$","*",".",",","0","1","2","3",
DATA "4","5","6","7","8","9"
```

Antes de examinar a segunda versão deste programa, apresentado nesta seção, um exercício prático e interessante para o leitor, seria a modificação da sub-rotina de codificação de meio byte para incluir a compressão de cadeias, contendo os caracteres $,. e *, bem como os dez numéricos.

Voltando à listagem, apresentada na Figura 3.25, se o caractere lésimo ou lésimo + 1 não for um dígito, o contador é incrementado de dois na linha 270 e a sub-rotina continua processando a linha de entrada obtida a partir do arquivo.

Quando o caractere lésimo ou lésimo + 1 na cadeia for um numérico, ocorrerá um desvio para a linha 290 do programa. Nesta linha, ocorre uma comparação para determinar se há suficientes caracteres em seqüência para se codificar. Quando K for maior que quatro, ocorre um desvio para a linha 350. Neste ponto do programa, a sub-rotina realmente executa a codificação de meio byte dos dados. Na linha 350, o caractere ASCII representado pelo valor 126, é colocado no elemento Jésimo do arranjo 0$. Este caractere é usado como um caractere de indicação de compressão e, como discutido anteriomente, será mostrado como um til (~). Na linha 360, o comprimento da cadeia é colocado no próximo elemento do arranjo 0$ e o índice de saída é, então, incrementado de 2 na linha 370. As linhas de 380 a 420 executam a codificação real de dois bytes de dados não compressíveis em sua representação de meio byte e juntam dois meio bytes em um único byte.

Antes de examinar a técnica empregada na linha 400, vamos primeiro examinar o método convencional de compactar, em BASIC, dois bytes numéricos em um único byte. Na linha 390, a função VAL é usada para obter a parte numérica dos caracteres L e L + 1 contidos na cadeia X$. Assim, X representa um caractere numérico, enquanto Y representa o segundo caractere numérico. Vamos supor que X fosse 6 e Y fosse 9. Sua composição de byte apareceria como a seguir:

0000 0110	X = 6
0000 1001	Y = 9

A compactação de dois numéricos em um byte pode ser realizada através da multiplicação de um caractere por 16 movendo, assim, quatro posições de bit à esquerda e, ainda, adicionando ou fazendo lógica AND com o segundo caractere. Supondo que Y seja multiplicado por 16, 9 * 16 é 144 e sua composição binária torna-se:

10010000	Y = 144

Assim, somando-se X e Y resulta no valor 150, cuja composição de byte é:

10010110	X + Y compactado = 150

O segundo método para realizar a colocação de dois numéricos em um byte foi usado na linha 400 da listagem do programa, na figura 3.25. Neste método, o valor numérico de Y foi primeiro multiplicado por 10 e, então, somado ao valor numérico de X. Então, o caractere representando o valor numérico da soma de X e Y multiplicado por 10 é colocado no arranjo 0$, como um único byte.

Voltando ao exemplo anterior, onde X era igual a 6 e Y a 9, ao multiplicar 9 por 10 e somar 6 resulta na compactação do caractere que tem o código ASCII 96, no elemento apropriado do arranjo 0$. Assim, se esta rotina de codificação de meio byte encontra a seqüência numérica 6 seguida por um 9 e há suficientes fileiras de numéricos para compactar dois caracteres juntos, eles seriam mostrados como um apóstrofe ('), já que o caractere é representado pelo ASCII 96. Nesta técnica, os códigos ASCII de 00 a 99 podem ser empregados para representar, diretamente, as 100 combinações possíveis de dois dígitos.

Para determinar os dados originais, você pode dividir por 10 o código ASCII recebido para obter um numérico e usar o restante do processo de divisão como o segundo numérico. Infelizmente, esta técnica não é aplicável se os caracteres adicionais, abordados anteriormente, são incluídos na cadeia de caracteres definida como sendo suscetível à codificação de meio byte.

Novamente, voltando à listagem do programa contido na Figura 3.25, observe que sempre que a contagem de caracteres adequados para a codificação de meio byte for menor que 5, ou um caractere não numérico for encontrado, ocorre um desvio para a

sub-rotina localizada na linha 440. Esta sub-rotina simplesmente tira o caractere de sua posição apropriada na cadeia X$ e coloca-o em sua posição apropriada no buffer de saída.

A última sub-rotina deste programa foi incluída para imprimir uma comparação de cada linha lida do arquivo, usada como entrada e a linha codificada na versão de meio byte. Além disso, a sub-rotina cria um arquivo que contém dados comprimidos, que serão usados como um arquivo de entrada para testar a rotina de descompressão, abordada a seguir. Começando na linha 900, esta sub-rotina também conta a entrada e saída dos caracteres e computa e imprimi a porcentagem de caracteres eliminados, como resultado da codificação de meio byte.

A figura 3.26 ilustra a execução do programa de codificação de meio byte BYTEC.BAS, mostrando as linhas originais dos dados contidos no arquivo de entrada, seguidas pelos dados comprimidos resultantes. O leitor deve observar que, para um melhor entendimento, os dados de entrada foram estruturados para assegurar que certos pares numéricos de caracteres fossem excluídos. Isto foi feito com o intuito de eliminar, como exemplo, dois meio bytes codificados, representando um caractere ASCII 31 ou menos, já que estes caracteres não podem ser impressos e não seriam apropriados para propósitos ilustrativos.

```
ENTER ASCII FILENAME. EG, BYTE.DAT
? BYTE.DAT
PATIENCE - INPUT PROCESSING
FILE BYTE.DAT BEFORE COMPRESSION:
1 '+43456789876543334567898765433345678987650
2 '-98357257894538629657398577526457497356872O
3 '$434445464748495051525354555657585960190
FILE BYTE.DAT AFTER COMPRESSION:
1 '+~&+-CYWA+!-CYWA+!-CY60
2 '-~(b#H9Y-&>'9'UM40911820
3 '$~$+,-./0123456789:;10
 54  TOTAL CHARACTERS ELIMINATED FROM  132 OR  40 %
Ok
```

Figura 3.26 - Execução de exemplo do programa BYTEC.BAS.

Descompressão

O programa BYTED.BAS, listado na figura 3.27, foi escrito para decodificar ou descomprimir dados previamente comprimidos pelo programa BYTEC.BAS.

Já que o programa BYTEC.BAS usou o caractere ASCII 126 como indicador de compressão de meio byte, o programa BYTEC.BAS foi

escrito para pesquisar a ocorrência deste caractere. Depois que uma linha de dados foi obtida de um arquivo na linha 130 do programa, o comprimento da linha é determinado na linha 140. Então, a sub-rotina da linha 180 é chamada para varrer a linha, caractere a caractere, procurando pela ocorrência de um ASCII 126.

```
10 REM BYTED.BAS PROGRAM
20 DIM O$(132)
30 WIDTH 80:CLS
40 '***********MAIN ROUTINE**********************
50 '* THIS ROUTINE READS RECORDS FROM AN ASCII *
60 '* FILE INTO A STRING CALLED X$ WHICH IS    *
70 '* THEN PASSED TO DECOMPRESSION SUBROUTINE  *
80 '*********************************************
90 PRINT "ENTER ASCII FILENAME. EG, BYTEC.DAT"
100 INPUT F$: OPEN F$ FOR INPUT AS #2
105 OPEN "BYTED.DAT" FOR OUTPUT AS #3
110 PRINT "PATIENCE - INPUT PROCESSING"
120 IF EOF(2) THEN GOTO 9000
130 LINE INPUT #2, X$
140 N= LEN(X$)
150 GOSUB 180
160 GOSUB 900
170 GOTO 120
180 '*****HALF BYTE  DECODING SUBROUTINE********
190 '* THIS ROUTINE PROCESSES RECORDS FROM X$  *
200 '* AND DECOMPRESSES BYTE-ENCODED CHARACTERS*
210 '* USING O$ AS THE OUTPUT BUFFER.          *
220 '*********************************************
230 K=1:J=1                         'RESET INDICES
240 FOR I= 1 TO N                   'STEP THRU RECORD
250 A$= MID$(X$,I,1)                'EXTRACT A CHAR
260 IF A$= CHR$(126) THEN 360       'COMPRESSION FLAG?
290 O$(J)=A$                        'STUFF IN OUTPUT BUFFER
300 J=J+1                           'BUMP BUFFER INDEX
310 NEXT I                          'GO BACK FOR MORE
320 RETURN                          'END OF STRING
355 '*****************************************************
360 'DECODE COMPRESSION NOTATION TO OUTPUT BUFFER
365 '*****************************************************
370 K$= MID$(X$,I+1,1)              'GET REPEAT COUNT
380 M= I+2                          'SETUP INPUT INDEX
390 K= ASC(K$)                      'SET UP LOOP INDEX
400 FOR L= J TO J+K-1 STEP 2        'SET OUTPUT LOOP
410 X= ASC(MID$(X$,M,1))            'GET ONE BYTE
420 Y= INT(X* .1)                   'SHIFT RIGHT
430 O$(L)= CHR$(Y+48)               'DECODE TENS POS
440 Z= INT(X-(Y* 10))               'SUBTRACT TENS POS
450 O$(L+1)= CHR$(Z+48)             'DECODE UNITS POS
455 M= M+1                          'BUMP INPUT INDEX
460 NEXT L                          'KEEP GOING
470 J= L+1                          'RESET OUTPUT INDEX
480 I= M                            'RESET INPUT INDEX
490 GOTO 250                        'DONE
```

Figura 3.27 - Listagem do programa BYTED.BAS.

O loop FOR-NEXT, delimitado pelas linhas 240 e 320, realiza esta tarefa, extraindo o caractere da cadeia, através do uso da função MID$ na linha 250 e, então, compara-se o caractere extraído com o ASCII 126 na linha 260.

Se o caractere extraído não for igual ao ASCII 126, ele é simplesmente colocado no buffer de saída na linha 290, o índice é incrementado em 1 na linha 300 e o processamento dos dados no loop continua. Se o caractere for igual ao ASCII 126, ocorre um desvio para a linha 360 e começa a decodificação dos dados comprimidos. Primeiro, a contagem de repetição, que é o próximo caractere na cadeia, é obtida na linha 370. Este caractere é convertido em valor numérico na linha 390, já que ele controlará o índice do loop para descomprimir os caracteres seguintes na cadeia, que foram previamente codificados em dois por byte. Esta decodificação é controlada pelo loop FOR-NEXT delimitado pelas linhas 400 a 460. Primeiro, o valor numérico do byte, que segue a contagem de repetição é obtida na primeira vez que a linha 410 for executada. Na linha 420, o valor obtido na linha anterior é multiplicado por 0,1, que, de fato, funciona como um deslocamento à direita. Ao pegar o número inteiro da multiplicação do valor numérico do byte por 0,1, obtemos um numérico entre 0 e 9. Este numérico representa o valor de Y, onde X e Y foram previamente codificados no programa BYTEC.BAS, ao multiplicar Y por 10 e ao adicionar o valor de X ao resultado. Já que estamos trabalhando com os caracteres com base em seus valores ASCII, 48 é adicionado ao valor de Y na linha 430, para obter o valor ASCII apropriado do dígito. Este valor, então é um caractere ASCII entre 0 e 9 que representa 10 posições dos dados codificados previamente. Na linha 440, o valor de Y multiplicado por 10 é subtraído do valor de X para se obter o valor numérico que representa a posição da unidade nos dados compactados. De forma semelhante à linha 430, a linha 450 adiciona 48 ao valor de Z para obter o código de caractere ASCII apropriado, que represente o dígito decodificado.

```
ENTER ASCII FILENAME. EG, BYTEC.DAT
? BYTEC.DAT
PATIENCE - INPUT PROCESSING
FILE BYTEC.DAT BEFORE DECOMPRESSION:
1 '+~&+-CYWA+!-CYWA+!-CY6◙
2 '-~(b#H9Y-&>'9'UM4◙91I82◙
3 '$~$+,-./0123456789:;1◙
FILE BYTEC.DAT AFTER DECOMPRESSION:
1 '+43456789876543334567898765433345678954◙
2 '-98357257894538629657398577526457497356500◙
3 '$434445464748495051525354555675859495◙
 54  TOTAL CHARACTERS INSERTED
Ok
```


Figura 3.28 - A execução de exemplo do programa BYTED.BAS.

A figura 3.28 ilustra a execução do programa BYTED.BAS, usando o arquivo BYTEC.DATA como entrada do programa. Já que o programa de compressão de meio byte, BYTEC.BAS, criou, previamente, este arquivo, não é nenhuma surpresa que as linha de 1 a 3, na parte superior da Figura 3.28, sejam iguais às linhas 1 a 3 da Figura 3.26, e que as linhas de 1 a 3, na parte inferior da Figura 3.28, sejam iguais às linhas 1 a 3 na parte superior da Figura 3.26.

A codificação estendida de meio byte

Um segundo exemplo de codificação de meio byte é resultante da inclusão de caracteres adicionais, além dos 10 numéricos, em meio bytes quando estes caracteres ocorrem seqüencialmente. Na figura 3.29, o leitor encontrará a listagem do programa PACKC.BAS que foi desenvolvido para comprimir uma cadeia contendo numéricos e/ou cifrão ($), vírgula (,), ponto decimal (.) e asterisco (*).

Programa de compressão

Semelhante ao programa BYTEC.BAS previamente descrito, uma linha de entrada de dados é obtida de um arquivo na linha 130, o comprimento da linha é determinado na linha 140 e um desvio para a subrotina de codificação de meio byte ocorre na linha 150 do programa.

A sub-rotina, delimitada pelas linhas 180 e 550, processa a linha de entrada e codifica as seqüências de numéricos e caracteres especiais previamente citados em meio bytes. O loop FOR-NEXT, delimitado pelas linhas 240 e 280, pesquisa as posições de caracteres na cadeia X$ que representa uma linha de dados de entrada. Na linha 242, o flag de arranjo C(I) é gerado, enquanto as linhas 243 e 244 extraem dois bytes da cadeia. O flag de arranjo C(I) é, então ativado com um valor entre 1 e 4, se o primeiro byte da cadeia (A$) for um dos caracteres especiais. Se A$ for um dígito entre 0 e 9, o flag de arranjo C(I) é, então, ativado com 5 na linha 256. De outra forma, o flag de arranjo C(I) permanece fixo em zero e ocorre um desvio para a linha 258, onde o segundo byte representado por B$, é processado. A seguir, as linhas de 258 a 266 processam o segundo byte, designando um valor entre 1 e 5 ao flag C(I + 1), dependendo se for encontrado um dos quatro caracteres especiais ou um numérico. Se C(I) ou C(I + 1) se igualarem a zero e forem encontrados quatro ou mais bytes contendo numéricos ou caracteres em seqüência, há o suficiente para se codificar e ocorre

```
10 REM PACKC.BAS PROGRAM
20 DIM O$(132),C(132)
30 WIDTH 80:CLS
40 '**********MAIN ROUTINE**********************
50 '* THIS ROUTINE READS RECORDS FROM AN ASCII *
60 '* FILE INTO A STRING CALLED X$ WHICH IS    *
70 '* THEN PASSED TO SUBROUTINES FOR COMPRESSION
80 '*********************************************
90 PRINT "ENTER ASCII FILENAME. EG, PACK.DAT"
100 INPUT F$: OPEN F$ FOR INPUT AS #2
105 OPEN "PACKC.DAT" FOR OUTPUT AS #3
110 PRINT "PATIENCE - INPUT PROCESSING"
120 IF EOF(2) THEN GOTO 9000
130 LINE INPUT #2, X$
140 N= LEN(X$)
150 GOSUB 180
160 GOSUB 900
170 GOTO 120
180 '*****HALF-BYTE ENCODING SUBROUTINE*********
190 '* THIS ROUTINE PROCESSES RECORDS FROM X$  *
200 '* AND ENCODES  MIXED  STRINGS OF DATA INTO*
210 '* HALF-BYTE OR 4 BIT REPRESENTATION USING *
215 '* DOUBLE BUFFERING WITH O$ AS OUTPUT BUFF.*
220 '*********************************************
230 K=1:J=1                          'RESET INDICES
240 FOR I=1 TO N STEP 2              'STEP THRU RECORD
242 C(I)=0:C(I+1)=0                  'RESET ENCODE FLAGS
243 A$= MID$(X$,I,1)                 'GET 1ST BYTE
244 B$= MID$(X$,I+1,1)               'GET 2ND BYTE
246 IF A$= "$" THEN C(I)= 1          'SET 1ST ENCODE FLAG
248 IF A$= "," THEN C(I)= 2
250 IF A$= "." THEN C(I)= 3
252 IF A$= "*" THEN C(I)= 4
254 IF A$< "0" OR A$> "9" THEN 258  'SKIP OTHERS
256 C(I)= 5
258 IF B$= "$" THEN C(I+1)= 1       'SET 2ND ENCODE FLAG
260 IF B$= "," THEN C(I+1)= 2
262 IF B$= "." THEN C(I+1)= 3
263 IF B$= "*" THEN C(I+1)= 4
264 IF B$< "0" OR B$> "9" THEN 268  'SKIP OTHERS
266 C(I+1)= 5
268 IF C(I)= 0 OR C(I+1)= 0 THEN 290  'NOT CANDIDATE
270 K=K+2                            'BOTH NUMERIC-BUMP COUNT
280 NEXT I                           'GO BACK FOR MORE
285 RETURN                           'END OF STRING
290 IF K > 4 THEN GOSUB 350          'ENOUGH TO ENCODE
300 IF K > 1 THEN GOSUB 500          'DON'T ENCODE
310 O$(J) = MID$(X$,I,1)             'OUTPUT 1ST CHAR.
320 O$(J+1) = MID$(X$,I+1,1)         'OUTPUT 2ND CHAR.
330 J=J+2:K=1                        'BUMP OUTPUT-RESET COUNT
340 GOTO 280                         'AND GO FOR MORE
350 O$(J)=CHR$(129)                  'FLAG FOR BYTE PACKING
352 MASK1= &HF0                      '11110000
354 MASK2= &HF                       '00001111
360 O$(J+1)=CHR$(K-1)                'INSERT LENGTH OF STRING
```

Figura 3.29 - Listagem do programa PACKC.BAS.

```
370 J=J+2                              'BUMP OUTPUT INDEX
371 FOR L=I-K+1 TO K STEP 2            'SETUP ENCODE LOOP
372 ON C(L) GOTO 376,378,380,382,384 'USE FLAG TO ENCODE
376 X=&HA0:GOTO 388                    '10100000
378 X=&HB0:GOTO 388                    '10110000
380 X=&HC0:GOTO 388                    '11000000
382 X=&HD0:GOTO 388                    '11010000
384 X=VAL(MID$(X$,L,1))                'GET NUM VALUE OF BYTE 1
386 X=X*16                             'SHIFT 4 BITS LEFT
388 X=X AND MASK1                      'MASK LOWER HALF-BYTE
390 ON C(L+1) GOTO 394,396,398,400,410 'USE ENCODE FLAG
394 Y=&HA:GOTO 420                     '00001010
396 Y=&HB:GOTO 420                     '00001011
398 Y=&HC:GOTO 420                     '00001100
400 Y=&HD:GOTO 420                     '00001101
410 Y=VAL(MID$(X$,L+1,1))              'GET NUM VALUE OF BYTE 2
420 Y=Y AND MASK2                      'MASK UPPER HALF-BYTE
440 Z= X OR Y                          'OR THE TWO TOGETHER
450 O$(J)= CHR$(Z)                     'OUTPUT BYTE TO BUFFER
460 J=J+1                              'BUMP OUTPUT INDEX
470 NEXT L                             'GO BACK FOR MORE
480 K=1:RETURN                         'RESET COUNT AND RETURN
500 '***** SUBROUTINE FOR STRING NOT WORTH ENCODING *****
510 FOR L=1-K+1 TO K                   'PICKUP SHORT STRING
520 O$(J)=MID$(X$,L,1)                 'STUFF IN OUTPUT BUFFER
530 J=J+1                              'BUMP OUTPUT INDEX
540 NEXT L                             'GO BACK FOR MORE
550 K=1:RETURN                         'RESET COUNT AND RETURN
900 '*****TALLY THE COMPRESSION COUNT & WRITE BUFFER******
910 '* DISPLAY BEFORE & AFTER RESULTS OF COMPRESSION     *
920 '* AND SHOW THE NET RESULTS OBTAINED BY EACH METHOD  *
930 '*************************************************
931 N1=N1+N                            'TALLY INPUT CHAR COUNT
932 T=N-J+1                            'NET DIFFERENCE IN BUFFERS
936 T1=T1+T                            'SAVE COUNT FOR SUMMARY
940 FOR I=1 TO J-1                     'OUTPUT FILE LOOP
950 PRINT #3, O$(I);                   'BUFFER CHAR STRING
960 NEXT I
965 PRINT #3, ""                       'NOW WRITE TO FILE
970 RETURN                             'DONE
9000 CLOSE: OPEN F$ FOR INPUT AS #2
9010 PRINT "FILE ";F$;" BEFORE COMPRESSION:"
9020 LINE INPUT #2,X$
9030 IF EOF(2) THEN GOTO 9060
9040 PRINT X$
9050 GOTO 9020
9060 PRINT X$: OPEN "PACKC.DAT" FOR INPUT AS #3
9070 PRINT "FILE ";F$;" AFTER COMPRESSION:"
9080 LINE INPUT #3,O$
9090 IF EOF(3) THEN 9998
9100 PRINT O$
9110 GOTO 9080
9998 PRINT O$:PRINT T1;" TOTAL CHARACTERS ELIMINATED FROM ";
9999 PRINT N1;"OR ";INT((T1/N1)*100);"%":CLOSE:END
```

Figura 3.29 - continuação.

um desvio para a sub-rotina que começa na linha 350. Se C(I) ou C(I + 1) se igualar a zero e se forem encontrados entre um e três bytes, ocorrerá um desvio para a sub-rotina que começa na linha 500. Esta sub-rotina simplesmente pega os caracteres encontrados na cadeia de entrada, e coloca-os em suas posições apropriadas no buffer de saída.

Quando dois byte são extraídos da cadeia de entrada e nenhum dos bytes anteriores foram caracteres numéricos ou especiais, C(I) e C(I + 1) são iguais a zero e a linha 268 faz com que ocorra um desvio para a 290. Já que K é igual a zero, as linhas de 310 a 330 são, então, executadas, resultando nos dois bytes, extraídos da cadeia de entrada, sendo colocados em posições apropriadas no buffer de saída.

As linhas de 350 a 480 contêm as instruções para gerar o caractere de indicação de compressão que, neste exemplo, é o ASCII 129 e, em seguida, compactar, em meio byte, os caracteres elegíveis para a compressão de meio byte. As linhas 352 e 354 habilitam dois flags de máscara, que permitirão que meio bytes superiores e inferiores sejam gerados pela função lógica AND do valor numérico do byte com o flag de máscara. A linha 360 insere o comprimento da cadeia no buffer de saída, enquanto a linha 372 examina o flag C e codifica o byte (linhas 376 a 382) com base no tipo de caractere especial no byte. Se o byte for numérico, a linha 384 será executada. Aqui, o valor numérico do byte é extraído. Na linha 386, é multiplicado por 16, que é equivalente a um deslocamento à esquerda de 4 bits, enquanto a linha 388 faz a função AND do valor do flag de caractere formado recentemente, ou do byte deslocado com a primeira máscara. De forma semelhante, as linhas de 390 a 420 executam a mesma operação no segundo byte, inicialmente examinando o segundo flag C. Finalmente, a linha 440 soma os dois meio bytes em um byte com o uso do operador OR e do caractere formado recentemente, que agora representa que dois caracteres são colocados no buffer de saída. Como nos outros programas previamente discutidos, as linhas 900 a 9999 acompanham a contagem da compressão e geram um arquivo chamado PACKC.DAT, que representa os dados comprimidos, contidos no arquivo PACK.DAT. Posteriormente, o programa de descompressão estendida de meio byte, chamado PACKD.BAS, usará o arquivo PACKC.DAT como entrada de dados para executar a descompressão estendida de meio byte.

A Figura 3.30 ilustra a execução do programa PACKC.BAS, usando um arquivo de dados de três linhas, cujo conteúdo está listado na parte superior da figura. Já que a compactação de alguns

meio bytes resultou na geração de um byte inteiro, cujo código ASCII estava abaixo de 31 e, conseqüentemente, não impressos, as duas primeiras linhas de dados comprimidos podem parecer estranhas, devido ao efeito que estes caracteres têm na impressora usada pelo autor.

```
ENTER ASCII FILENAME. EG, PACK.DAT
? PACK.DAT
PATIENCE - INPUT PROCESSING
FILE PACK.DAT BEFORE COMPRESSION:
1 '+$43,456,789.87$$654333456789876543334567898700
2 '-$9835$72.57$89.45$386,296,573.8577526457497356780O
3 '$434445464748495051525354555657585987OO
FILE PACK.DAT AFTER COMPRESSION:
1 '+,ñ;Ekx▌eC3Eg          eC3Eg    OO

2 '-O⌐Zr┼z        -Z8k)kW<wRdWIsVOO

3 '$$CDEFGHIPQRSTUVWXYOO
 61   TOTAL CHARACTERS ELIMINATED FROM  146 OR  41 %
Ok
```

Figura 3.30 - Execução de exemplo do programa PACKC.BAS.

Programa de descompressão

A figura 3.31 contém a listagem do programa PACKD.BAS, que foi desenvolvido para descomprimir dados comprimidos pelo programa PACKC.BAS.

Semelhante ao programa PACKC.BAS em termos de construção, o PACKD.BAS obtém uma linha de dados de um arquivo na linha 130, determina o comprimento da linha, na linha 140 e, então, desvia para a sub-rotina que começa na linha 180, para executar a decodificação necessária. O loop FOR-NEXT, delimitado pelas linhas 240 e 320, extrai um caractere por vez da cadeia de entrada, procurando o ASCII 129, que é o caractere de indicação de compressão, usado para indicar a ocorrência da compressão estendida de meio byte.

Quando o flag de compressão for encontrado na linha 260, ocorrerá um desvio para a linha 330 que é o início da rotina que descomprime os dados comprimidos. Depois de iniciar as máscaras nas linhas 330 e 335, o comprimento da cadeia é obtido na linha 340, enquanto o loop FOR-NEXT, delimitado pelas linhas 350 e 498, fragmenta cada byte em dois caracteres originais que foram previamente comprimidos. Inicialmente, a linha 370 pega um byte e faz a função AND com a primeira máscara e divide por 16, que é equiva-

lente ao deslocamento à direita de 4 posições de bits. Na linha 375, o caractere é testado para se verificar se é numérico. Se for assim, ocorre um desvio para a linha 430, onde 48 é adicionado ao caractere para obter o valor ASCII apropriado. Se o caractere não for numérico, as linhas de 380 a 410 farão o teste para determinar que caractere especial o caractere representa. Isto se dá pelo exame de seu valor de código e, então, com base em seu valor de código, o caractere é restaurado para seu valor original. A seguir, as linhas de 440 a 490 executam a mesma operação no segundo meio byte do caractere recebido.

```
10 REM PACKD.BAS PROGRAM
20 DIM O$(132)
30 WIDTH 80:CLS
40 '**********MAIN ROUTINE**********************
50 '* THIS ROUTINE READS RECORDS FROM AN ASCII *
60 '* FILE INTO A STRING CALLED X$ WHICH IS    *
70 '* THEN PASSED TO DECOMPRESSION SUBROUTINE  *
80 '***************************************************
90 PRINT "ENTER ASCII FILENAME. EG, PACKC.DAT"
100 INPUT F$: OPEN F$ FOR INPUT AS #2
105 OPEN "PACKD.DAT" FOR OUTPUT AS #3
110 PRINT "PATIENCE - INPUT PROCESSING"
120 IF EOF(2) THEN GOTO 9000
130 LINE INPUT #2, X$
140 N= LEN(X$)
150 GOSUB 180
160 GOSUB 900
170 GOTO 120
180 '*****HALF BYTE  DECODING SUBROUTINE********
190 '* THIS ROUTINE PROCESSES RECORDS FROM X$  *
200 '* AND DECOMPRESSES BYTE-ENCODED CHARACTERS*
210 '* USING O$ AS THE OUTPUT BUFFER.          *
220 '***********************************************
230 J=1                          'RESET INDEX
240 FOR I= 1 TO N                'STEP THRU RECORD
250 A$= MID$(X$,I,1)             'EXTRACT A CHAR
260 IF A$= CHR$(129) THEN 330    'COMPRESSION FLAG?
290 O$(J)=A$                     'STUFF IN OUTPUT BUFFER
300 J=J+1                        'BUMP BUFFER INDEX
310 NEXT I                       'GO BACK FOR MORE
320 RETURN                       'END OF STRING
322 '*********************************************************
324 'DECODE COMPRESSION NOTATION TO OUTPUT BUFFER       *
326 '*********************************************************
330 MASK1= &HF0                  '11110000
335 MASK2= &HF                   '00001111
340 K= ASC(MID$(X$,I+1,1))       'GET STRING LENGTH
345 M= I+(K/2)                   'SET END OF STRING
350 FOR L=I+2 TO M               'SETUP LOOP TO DECODE
362 Z= ASC(MID$(X$,L,1))         'GET BYTE
```

Figura 3.31 - Listagem do programa PACKD.BAS.

```
370 X= (Z AND MASK1)/16          'MASK LOWER HALF-BYTE
375 IF X< 10 THEN 430            'ITS NUMERIC
380 IF X= 10 THEN O$(J)= "$"     'SPECIAL
390 IF X= 11 THEN O$(J)= ","     'SPECIAL
400 IF X= 12 THEN O$(J)= "."     'SPECIAL
410 IF X= 13 THEN O$(J)= "*"     'SPECIAL
415 GOTO 440                     'SKIP IF SPECIAL
430 O$(J)= CHR$(X+48)            'OUTPUT 1ST NUMERIC
440 Y= Z AND MASK2               'MASK UPPER HALF-BYTE
445 IF Y< 10 THEN 490            'ITS NUMERIC
450 IF Y= 10 THEN O$(J+1)= "$"   'SPECIAL
460 IF Y= 11 THEN O$(J+1)= ","   'SPECIAL
470 IF Y= 12 THEN O$(J+1)= "."   'SPECIAL
480 IF Y= 13 THEN O$(J+1)= "*"   'SPECIAL
485 GOTO 495                     'SKIP IF SPECIAL
490 O$(J+1)= CHR$(Y+48)          'OUTPUT 2ND NUMERIC
495 J= J+2                       'BUMP OUTPUT BY TWO
498 NEXT L:I= M                  'CONTINUE, BUMP INPUT INDEX
499 GOTO 310                     'GO BACK FOR MORE
900 '*****TALLY THE DECOMPRESSION COUNT & WRITE BUFFER****
910 '* DISPLAY BEFORE & AFTER RESULTS OF DECOMPRESSION   *
920 '* AND SHOW THE NET RESULTS OBTAINED BY EACH METHOD  *
930 '*************************************************************
931 N1=N1+N                      'TALLY INPUT CHAR COUNT
932 T=N-J+1                      'NET DIFFERENCE IN BUFFERS
936 T1=T1-T                      'SAVE COUNT FOR SUMMARY
940 FOR I= 1 TO J-1
950 PRINT #3, O$(I);
960 NEXT I
965 PRINT #3, ""
970 RETURN
9000 CLOSE: OPEN F$ FOR INPUT AS #2
9010 PRINT "FILE ";F$;" BEFORE DECOMPRESSION:"
9020 LINE INPUT #2,X$
9030 IF EOF(2) THEN 9060
9040 PRINT X$
9050 GOTO 9020
9060 PRINT X$:OPEN "PACKD.DAT" FOR INPUT AS #3
9070 PRINT "FILE ";F$;" AFTER DECOMPRESSION:"
9080 LINE INPUT #3,O$
9090 IF EOF(3) THEN 9998
9100 PRINT O$
9110 GOTO 9080
9998 PRINT O$:PRINT T1;" TOTAL CHARACTERS INSERTED"
9999 CLOSE:END
```

Figura 3.31 - continuação.

Execução de programa

A figura 3.32 ilustra a execução do programa PACKD.BAS, usando o PACKC.DAT como arquivo de dados de entrada para ser descomprimido. O leitor observará que as três primeiras linhas na Figura 3.32, são idênticas às três últimas linhas da Figura 3.30, enquanto as três últimas linhas da Figura 3.32, que representam os dados descomprimidos, são idênticas às três linhas na parte superior da Figura 3.30. Novamente, isto não é surpresa, já que o programa de descompressão simplesmente reconstrói os dados comprimidos em sua forma original. O leitor deve também observar que os 61 caracteres indicados como eliminados pela compressão de meio byte na Figura 3.30, não levam em consideração os caracteres adicionais de compressão, necessários para indicar todas as ocorrências da codificação de meio byte. Se isto for feito, então, um total de 55 caracteres serão eliminados, o que combina com a contagem de inserção de 55 caracteres na Figura 3.32.

```
ENTER ASCII FILENAME. EG, PACKC.DAT
? PACKC.DAT
PATIENCE - INPUT PROCESSING
FILE PACKC.DAT BEFORE DECOMPRESSION:
1 '+,ñ;Ekx▌eC3Eg          eC3Eg    ØØ

2 '-O⌐Zr┼z        -Z8k)kW<wRdWIsVØØ

3 '$$CDEFGHIPQRSTUVWXYØØ
FILE PACKC.DAT AFTER DECOMPRESSION:
1 '+$43,456,789.87**6543334567898765433345678ØØ
2 '-$9835$72.57$89.45$386,296,573.857752645749735ØØ
3 '$434445464748495051525354555657585ØØ
 55  TOTAL CHARACTERS INSERTED
Ok
```

Figura 3.32 - Execução de exemplo do programa PACKD.BAS.

Variações da codificação e considerações de eficiência

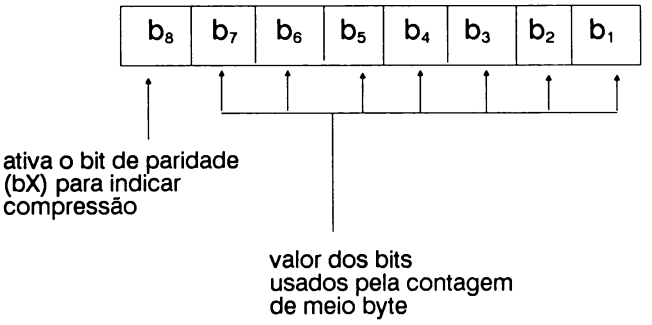
Os exemplos de codificação, previamente abordados nesta seção, ilustraram o uso de um caractere especial de indicação de compressão, ao supor que todos os caracteres estavam disponíveis para seleção. Vamos supor que seus algoritmos de decodificação e de codificação de meio byte devam trabalhar com dados ASCII de 7 bits ou ASCII de 8 bits não padronizado, como definido

pelo conjunto de caracteres usados pelos microcomputadores IBM PC e compatíveis. Nós gostaríamos de modificar nossa escolha do caractere de indicação de compressão e, se assim for, como poderíamos fazê-lo?

Vamos, primeiramente, supor que transmitiremos ou armazenaremos dados ASCII de 7 níveis. Já que os sistemas de computação operam com bytes de oito bits, se torna possível o uso da posição do oitavo bit como um indicador de compressão, semelhante ao método alternativo descrito pela técnica de codificação de mapeamento de bit. A parte superior da Figura 3.33 ilustra como poderíamos definir um caractere pelo uso do bit de paridade (bit 8) em um conjunto de caracteres ASCII de 7 bits, para representar tanto o caractere de indicação de compressão, como a contagem do número de meio bytes comprimidos. Neste exemplo, ocorreu a ativação da oitava posição de bit como um indicador de compressão de meio byte, enquanto que as posições de bit 1 a 7 contêm a contagem binária do número de meio bytes, que foram comprimidos. Aqui, o contador da posição de 7 bits pode representar uma seqüência de até 127 meio bytes comprimidos.

Já que usamos previamente esta técnica para indicar a ocorrência de mapeamento de bit, vamos supor que desejamos implementar o mapeamento de bits de caracteres nulos repetidos e codificação de meio byte de dados financeiros, usando um conjunto de caracteres ASCII de 7 níveis. Para realizar esta tarefa, usaríamos a ativação do bit 8 como um indicador 'geral' de compressão. Então, poderíamos usar o valor do bit 7 para indicar os dois tipos de compressão. Finalmente, as posições de bit 1 a 6 ou representariam o mapa de bit de 6 posições, ou a contagem de meio bytes comprimidos com base no valor do bit 7. A parte inferior da Figura 3.33 ilustra como um formato de caractere de 8 bits pode ser usado para definir dois tipos de compressão de dados, quando dados de 7 bits são encapsulados em 8 bits. É claro, ao usar o oitavo bit, você elimina a possibilidade de usar aquele bit para verificação de paridade. Contudo, já que a maioria dos software de comunicação incluem, agora, um mecanismo de verificação de bloco com propósitos de controle de erro, a perda da capacidade de verificação de paridade, para a maioria das pessoas, será considerada como transparente.

Combinando um indicador e um contador de compressão



Definindo técnicas múltiplas de compressão com um caractere

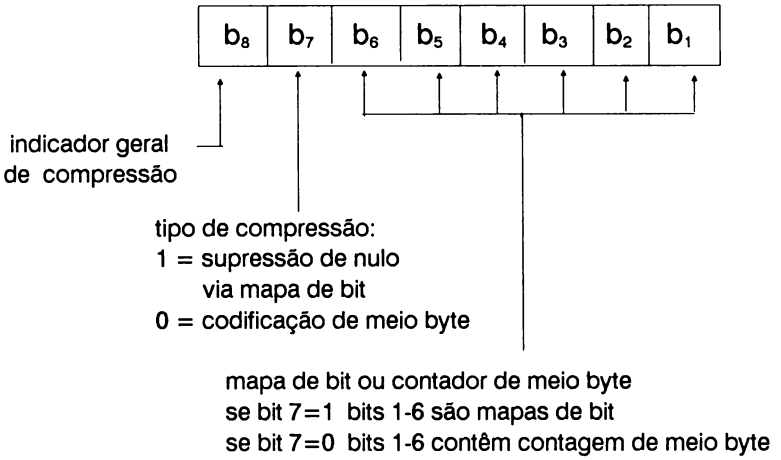


Figura 3.33 - Usando o ASCII de 7 níveis.

O formato ilustrado na parte inferior da Figura 3.33, é somente aplicável para definir os dois tipos de compressão. Você pode estender esta técnica ainda mais, usando as posições de bit 6 e 7 para indicar até quatro tipos de compressão. Desta forma, reduz o mapa de bit para cinco bits e a contagem máxima de meio bit para 31.

Se estamos usando um conjunto verdadeiro de caracteres de 8 bits, não vamos querer usar a técnica descrita anteriormente. Isto se dá porque o esquema anterior poderia, eventualmente, gerar todos os caracteres cujos valores ASCII excedem 127. Em vez disso, provavelmente usaríamos a técnica de inserção e deleção para obter dois caracteres que não aparecem freqüentemente nos dados, para usar como caracteres de indicação de compressão - talvez os caracteres cujos valores ASCII são 126 e 129!

3.5 CODIFICAÇÃO DIATÔMICA

Como o próprio nome sugere, a codificação diatômica é um processo de compressão de dados, através da qual um par de caracteres é substituído por um caractere especial. A estrutura binária do caractere especial representa o par codificado de caracteres e, assim, permite uma redução de dados de 50 por cento ou uma razão de compressão de 2:1.

Já que o número de caracteres especiais, que pode ser empregado para representar diferentes tipos de compressão, é limitado, o potencial teórico de se obter uma redução de dados de 50 por cento ao substituir todos os pares de caracteres por um caractere não pode ser obtido. Para aumentar o potencial de compressão, é necessário um entendimento prévio da composição de seus dados. Uma vez que você saiba a freqüência esperada de ocorrência de pares de caracteres, então os pares encontrados com mais freqüência podem ser escolhidos como candidatos à codificação diatômica. O número real de pares escolhidos dependerá do número de caracteres especiais disponíveis, para representar aqueles pares de caracteres que ocorrem freqüentemente.

Operação

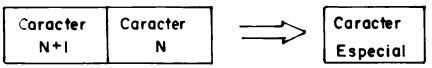
A representação por meio de diagrama de blocos do processo de codificação diatômica encontra-se na parte superior da Figura 3.34. Na parte inferior desta ilustração, encontra-se no fluxograma, indicando os principais processos necessários para codificar os dados diatomicamente. Observe que o fluxograma supõe que ocorra um fluxo contínuo de dados de entrada. Na realidade, os buffers de entrada e de saída teriam comprimento finito. Já que o buffer de saída sempre será menor ou igual ao tamanho do buffer de entrada, é possível designar um indicador que será incrementado através do buffer de entrada. Ao alcançar o fim daquele buffer, o conteúdo do buffer de saída será transmitido, enquanto o buffer

de entrada será repreenchido com dados adicionais não comprimidos.

Freqüência de ocorrência de pares

O principal problema da implementação da codificação diatômica é determinar que pares devem ser representados por caracteres especiais.

A. Processo de codificação diatômica



B. Fluxograma da codificação diatômica

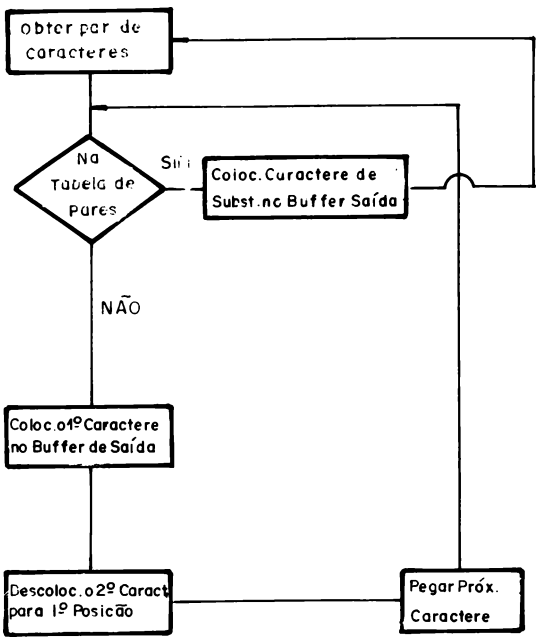


Figura 3.34 - O processo da codificação diatômica.

Para executar a codificação diatômica e obter uma razão de compressão significativa, é necessária a indicação de caracteres especiais para representar os pares de caracteres de ocorrência mais freqüente que, se espera, encontrar no fluxo original de dados.

Isto significa que você deve ter algum conhecimento prévio sobre o tipo de dados a serem manipulados, de maneira que você possa fundamentar a escolha de caracteres especiais de uma forma mais precisa (Snyderman e Hunt, 1970).

Para auxiliar os leitores na seleção dos pares de caracteres apropriados a serem substituídos por caracteres especiais, diversas tabelas de combinações de pares são apresentadas nesta seção. Na tabela 3.9, o leitor encontrará uma tabela contendo os primeiros 25 pares de caracteres encontrados com maior freqüência em um texto em Inglês com 12.198 caracteres (Aronson, 1977). Esta tabela, preparada por Jewell, indica a ordem, a combinação de pares, o número de ocorrências do par e a ocorrência por milhares de caracteres de dados (Jewell, 1976).

Tabela 3.9 - A combinação de pares de caracteres de Jewell

Ordem	Combinação	Ocorrências	Ocorrências por mil
1	E_	328	26.89
2	_T	292	23.94
3	TH	249	20.41
4	_A	244	20.00
5	S_	217	17.79
6	RE	200	16.40
7	IN	197	16.15
8	HE	183	15.00
9	ER	171	14.02
10	_I	156	12.79
11	_O	153	12.54
12	N_	152	12.46
13	ES	148	12.13
14	_V	141	11.56
15	ON	140	11.48
16	T_	137	11.23
17	TI	137	11.23
18	AN	133	10.90
19	D_	133	10.90
20	AT	119	9.76
21	TE	114	9.35
22	_C	113	9.26
23	_S	113	9.26
24	OR	112	9.18
25	R_	109	8.94

Observação: _ representa um caractere de espaço

Já que muitos usuários de transmissão de dados transferirão os arquivos de programa, junto com dados textuais, é apresentada uma análise da composição de pares de caracteres para os programas em BASIC, COBOL e FORTRAN. A análise destes programas foi obtida pela execução do programa DATANALYSIS, escrito pela 4-Degree Consulting, localizada em Macon, Georgia, USA. Este programa executa uma análise de susceptibilidade à compressão nos arquivos de dados, e a análise de pares de caracteres listada nas Tabelas 3.10 a 3.13, apresenta um dos diversos algoritmos de compressão analisados por aquele pacote de software. A listagem das declarações de software no programa DATANALYSIS, encontra-se no Apêndice A. Seu uso facilitará a escolha de um ou mais algoritmos de compressão com base na análise da susceptibilidade do seu tráfego de dados real ou previsto para os diversos algoritmos de compressão.

Tabela 3.10 - Análise da compressão de pares de caracteres, arquivo de dados em BASIC com 9.322 caracteres

Par/Cont	Par/Cont	Par/Cont	Par/Cont	Par/Cont	Par/Cont
_P 13	NT 13	RI 13	_T 12	_ 11	_I 8
O_ 7	HE 7	_B 6	_S 6	T_ 5	N_ 5
_A 5	F_ 5	_E 4	AB 4	BU 4	EX 4
_L 4	IN 4	IO 4	NI 4	_N 4	_U 4
TO 4	TS 4	_F 3	R_ 3	S_ 3	ND 3
E_ 3	_R 3	OF 3	UR 3	OU 3	_C 3
SE 3	ET 3	_O 3	_D 2	NP 2	NS 2
EL 2	AC 2	ON 2	IL 2	OT 2	IT 2
LI 2	RO 2	LL 2	AT 2	NG 2	UT 2
IM 1	AL 1	AN 1	_K 1	BO 1	IU 1
LA 1	LD 1	BR 1	M_ 1	LO 1	LU 1
MB 1	CK 1	NE 1	CO 1	CT 1	NO 1
DO 1	ED 1	_X 1	NU 1	EN 1	OL 1
EQ 1	ER 1	OS 1	ES 1	Y_ 1	PP 1
PS 1	PT 1	_Y 1	GH 1	_G 1	SO 1
SS 1	ST 1	TA 1	TE 1	TH 1	TI 1
HI 1	HT 1	TU 1	UG 1	UI 1	UL 1
UN 1	IA 1	VA 1	XX 1	YE 1	YI 1
ZE 1	** 0	** 0	** 0	** 0	** 0

Total de combinações encontradas: 288

Observação: _ representa um caractere de espaço cont - representa contagem

A Tabela 3.10 mostra os resultados da análise de compressão de pares de caracteres com base no exame de um programa em

BASIC com 9.322 caracteres. Em geral, a maioria dos programas em linguagem BASIC contém uma grande proporção de mensagens de entrada e prompts, bem como cabeçalhos de saída. Esta estrutura faz com que a consistência de pares de caracteres forme uma consistência modificada de pares de caracteres do texto em Inglês. Normalmente, o grau de desvio dos pares de dados textuais do Inglês normal é resultante da razão das declarações computacionais pelas declarações de entrada/saída no programa. Na tabela 3.10, observe que '_P' 'NT' e 'RI' são os pares encontrados com maior freqüência. Todos os três pares vêm das declarações PRINT no programa com o par '_P', sendo resultante do uso pelo programador de 1 espaço precedendo cada declaração PRINT. De forma semelhante, a declaração do formato 'IF X:Y THEN' pode ser indicada na linguagem BASIC pelos pares '_I', 'F_' e 'HE' encontrados freqüentemente.

Tabela 3.11 - Análise da compressão de pares de caracteres. Arquivo de dados em FORTRAN com 20.465 caracteres.

Par/Cont		Par/Cont		Par/Cont		Par/Cont		Par/Cont		Par/Cont	
_I	167	_F	116	TE	106	UT	105	OR	99	OU	99
RI	96	_W	87	MA	86	_C	86	IN	86	TP	81
IR	69	HA	66	_S	61	O_	60	ER	60	C_	57
IT	51	HE	48	EN	46	RA	44	_D	42	E_	42
AL	40	RE	39	SI	38	IX	38	ON	37	_T	36
HS	35	HD	34	TO	34	SU	33	_R	32	T_	31
IM	30	_V	30	TA	30	HB	30	HF	30	HN	30
HC	30	HO	29	HR	29	HG	29	HU	29	HV	29
TI	29	HH	29	HL	29	AR	29	HJ	28	HT	28
_N	28	HM	28	HW	28	HX	28	HY	28	HZ	28
IA	28	CT	28	HI	28	IP	28	HP	28	HQ	28
HJ	28	I_	27	IO	26	_G	26	EQ	25	NY	25
_O	25	UB	25	IY	25	LE	24	_E	24	_P	23
AN	23	__	23	N_	23	ND	22	CO	22	SE	22
_A	22	Y_	21	AT	21	R_	20	S_	20	IS	20
RO	20	IC	20	NC	20	PR	20	ED	19	TR	18
G_	18	UR	18	ES	18	OT	17	ET	16	NG	16
NA	16	LY	16	TH	15	AC	15	PE	14	D_	14
PU	14	UM	14	NU	14	CH	14	BL	13	IV	13
LI	13	_J	13	FI	12	PF	12	GO	12	_K	12
OW	12	ST	12	_M	11	IL	11	SS	11	LA	11
AI	11	EP	11	NS	11	DA	11	EA	10	EC	10
TS	10	TY	10	FO	10	UE	10	F_	10	UN	10

Total de combinações encontradas: 4.370

Observação: _ representa um caractere de espaço

Na Tabela 3.11, são apresentados os resultados de uma análise semelhante de um programa em FORTRAN com 20.465 caracteres, enquanto a Tabela 3.12 indica os pares encontrados, quando um programa em COBOL de 54.417 foi analisado. Na análise do programa FORTRAN, os pares mais comuns resultam das declarações usadas com maior freqüência como 'FORMAT', 'WRITE' e 'READ'. De forma semelhante, os pares mais comuns no programa em COBOL, geralmente resultam da declaração 'PICTURE IS'. Finalmente, a Tabela 3.13 mostra os resultados da análise da fusão dos programas em BASIC, FORTRAN e COBOL em uma entidade. Aqui, os 230 pares de caracteres representam um total de 16.266 combinações. Já que o arquivo continha um total de 84.204 caracteres de dados, a compressão diatômica dos 230 pares de caracteres encontrados com maior freqüência resultaria em uma redução de dados de 19,3 por cento (16.266/84.204). Observe que os 12 pares encontrados com maior freqüência representam uma redução potencial de dados de 4.388 caracteres ou, aproximadamente, de 25 por cento da redução teórica obtida ao se codificar, diatomicamente, os 230 pares encontrados com maior freqüência. A partir disto, é evidente que a codificação diatômica pode ser efetivamente usada em conjunto com outras técnicas de compressão, desde que se selecione somente uma parte dos pares, que se espera encontrar com maior freqüência, para a representação por caracteres especiais indicadores de compressão.

Tabela 3.12 - Análise da compressão de pares de caracteres, programa em COBOL contendo 54.417 caracteres.

Par/Cont		Par/Cont		Par/Cont		Par/Cont		Par/Cont		Par/Cont	
_P	542	_F	391	IC	342	AL	316	IN	309	RE	297
E_	286	_V	251	_X	243	_W	239	LE	235	R_	235
_T	231	IL	229	UE	229	TE	221	PG	212	_O	211
NT	204	M_	190	AR	186	RO	186	O_	182	UT	178
CN	164	_C	164	CT	156	LN	153	RI	152	_M	151
CH	149	_L	148	OV	144	CI	141	FO	139	TY	137
_V	136	OR	129	_S	129	_I	122	_A	118	TO	110
ER	109	G_	108	NG	106	RM	102	EF	101	SE	97
AD	91	__	91	VA	90	PA	88	ES	87	AC	84
NC	84	T_	82	F_	79	AN	79	TA	67	DV	66
FI	63	S_	61	WR	60	TI	59	ST	57	_E	55
Y_	54	PU	52	_R	50	BU	49	QU	48	OM	46
ON	44	OT	43	AT	43	OF	42	CO	40	RK	40
LS	36	CD	35	D_	35	NE	35	AS	34	OU	33
_Z	33	_G	33	ME	32	IV	30	RP	30	EN	30
ND	29	_U	28	BE	27	OP	27	L_	25	H_	24
EL	24	TP	23	SE	23	CA	22	_H	22	_D	22
TH	22	DD	21	TR	20	ET	20	VE	20	VI	20
EC	20	YI	20	_N	19	GS	19	IT	19	C_	19
GE	18	WA	18	UA	18	SH	18	_K	17	IM	17
RA	17	RS	17	MO	16	DE	16	GI	15	EX	15
ED	15	MP	15	CC	15	WO	15	EQ	15	UN	15
LI	14	IO	14	_Q	14	UR	14	DI	13	FL	13

Total de combinações encontradas: 12.509

Observação: _ representa um caractere de espaço

Tabela 3.13 - Análise da compressão de pares de caracteres, arquivo de 84.204 caracteres combinados

Par/Cont		Par/Cont		Par/Cont		Par/Cont		Par/Cont		Par/Cont	
__P	578	__F	510	IN	399	IC	362	AL	357	RE	336
E__	331	TE	328	__W	326	__I	297	UT	285	__T	279
RI	261	LE	259	R__	258	__V	255	__C	253	__X	252
O__	249	NY	242	IL	240	UE	239	__O	239	OR	231
AR	215	PG	212	RO	208	__S	196	M__	194	CT	185
__B	172	ER	170	CN	164	CH	163	__M	162	__L	159
LN	153	FO	149	TO	148	TY	147	__A	145	OV	144
CI	141	OU	135	G__	126	__	125	NG	124	T__	118
CE	107	ES	106	RM	105	TP	104	NC	104	AN	103
EF	102	AC	101	PA	98	TA	98	MA	94	F__	94
AD	91	VA	91	TI	89	__R	85	S__	84	__E	83
ON	83	EN	77	HA	77	C__	76	Y__	76	FI	75
IT	72	IR	71	ST	70	DV	66	PU	66	AT	66
__D	66	CO	63	OT	62	HE	62	RA	61	__G	60
WR	60	QU	56	ND	54	OF	4	BU	54	OM	53
L__	52	__N	51	D__	49	SE	48	IN	48	IO	44
IV	43	SI	41	NE	41	EQ	41	RK	40	N__	40
ET	39	IX	38	TH	38	TR	38	ME	38	HD	37
AS	37	LS	36	HR	36	UB	35	HS	35	CD	35
__U	35	ED	35	__Z	34	SU	33	HO	33	HP	32
HY	32	UR	32	IA	31	IS	31	HI	31	OP	31
HF	30	HB	30	HN	30	PR	30	HC	30	RP	30
__K	30	EC	30	LI	29	HV	29	HH	29	BE	29

Total de combinações encontradas: 16.266

Observação: _ representa um caractere de espaço

Implementação por hardware nas comunicações

O uso da técnica de compressão diatômica de dados, em combinação com vários outros algoritmos de compressão, foi implementado pela Infotron Systems em seu multiplexador estatístico TL780.

Em um multiplexador convencional de divisão de tempo, os dados de cada canal de entrada são designados para uma "janela" na linha de saída do multiplexador de alta velocidade, não importando se a largura de banda foi usada ou não. Já que para cada linha de entrada de dados é designada uma "janela" de tempo correspondente, a implementação da compressão em um enlace de alta velocidade, não aumentará a eficiência de nenhuma linha. Se a compressão for implementada no lado de baixa velocidade da linha, normalmente conhecido como o lado de canal ou nível, a eficiência de cada enlace comprimido de baixa velocidade será aumentada, uma vez que para cada ligação é reservada uma "janela" fixa no lado de alta velocidade. Isto encontra-se ilustrado na parte superior da Figura 3.35.

Em um multiplexador estatístico, a largura de banda para um determinado canal no enlace de alta velocidade é somente usada quando o canal está transmitindo dados ou sinais de controle. Conseqüentemente, a compressão de uma ou mais ligações de baixa velocidade permite que o multiplexador estatístico utilize menos largura de banda da linha de alta velocidade para o enlace de baixa velocidade sendo comprimido. A compressão do enlace de baixa velocidade no lado de canal, resultará, então, em uma taxa mais baixa de linha de alta velocidade, ou permitirá que mais canais de baixa velocidade sejam adicionados, uma vez que a compressão reduz o número total dos dados transmitidos pela linha de alta velocidade. Inversamente, se a compressão for executada ao nível da linha de alta velocidade, o número de caracteres transmitidos naquele enlace será reduzido. Isto tornará possível obter uma taxa mais baixa de dados operacionais de alta velocidade, ou permitirá que canais adicionais de baixa velocidade sejam ativados. Enquanto alguns fabricantes decidiram comprimir o enlace de alta velocidade, a Infotron usa um processo de codificação diatômica, combinado com técnicas adicionais de compressão de dados em seus adaptadores de canais de baixa velocidade, para executar a compressão ao nível de canal. Esta técnica permite ao usuário selecionar quais os canais, se houver algum, que devem ser comprimidos.

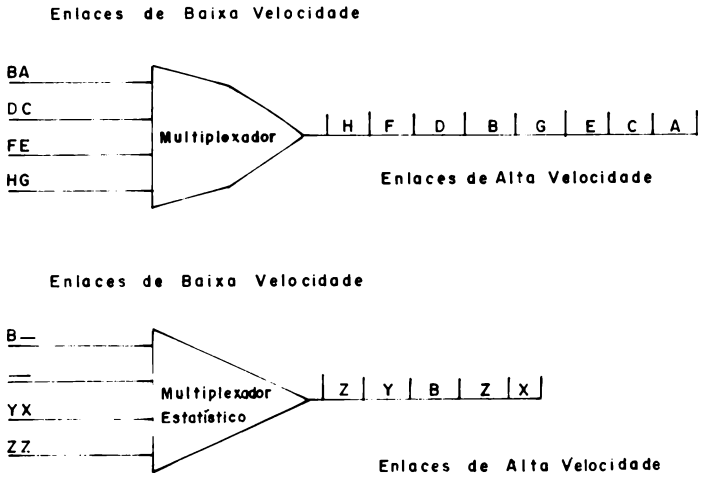


Figura 3.35 - Multiplexação e compressão. Se a compressão ocorrer em um ou mais enlaces de baixa velocidade, a razão efetiva de transferência de informações daqueles enlaces individuais aumentará, quando for empregado um multiplexador convencional de divisão de tempo. Quando os multiplexadores estatísticos são empregados, os dados podem ser comprimidos a nível de canal individual ou global a nível da linha de alta velocidade.

Na técnica da Infotron, a compressão do multiplexador estatístico ocorre através do uso de códigos de espaços múltiplos, códigos de caracteres repetidos, códigos de pares freqüentes de caracteres (codificação diatômica) e códigos decimais compactados (codificação de meio byte). Além disso, já que os dados devem ser enfileirados no nível de canal para comprimi-los, torna-se necessário transmitir sinais de controle através da trajetória de dados do enlace de alta velocidade, de maneira a preservar a relação de tempo entre os dados e os sinais de controle. A adição destes sinais reduz a eficiência global de compressão. Já que cada adaptador de canal no multiplexador necessita de uma área de buffer e um microprocessador para efetivar a compressão, a compressão de um grande número de canais de baixa velocidade fica mais cara, do ponto de vista do hardware, do que a compressão de dados a nível de linha de alta velocidade, onde somente uma área de buffer e um único microprocessador são necessários.

O adaptador de canal da Infotron, que executa a compressão, somente opera com dados assíncronos codificados em ASCII. Para se obter um número suficiente de códigos especiais de indicação de compressão, o bit de paridade no código ASCII normal de oito bits é retirado da transmissão. Isto resulta em 128 códigos de

caracteres que podem ser usados para representar e indicar as informações comprimidas. A retirada da paridade pelo microprocessador dentro do multiplexador não tem efeito sobre os erros, uma vez que o multiplexador emprega a transmissão de pacotes do tipo HDLC no nível de enlace de alta velocidade, incluindo a geração de verificação cíclica da redundância dos pacotes transmitidos.

Na Infotron Systems, os códigos são designados para representar grupos de 2 a 7 espaços consecutivos para vários esquemas de código de compressão de espaço múltiplo. Estes códigos são mais eficazes quando os dados transmitidos foram formatados em colunas separadas por grupos de espaços, ou são informações textuais que contêm alinhamentos de parágrafo e margem justificada, através do uso de espaços.

Para a representação dos caracteres repetidos, foram designados 16 códigos para representar grupos de 3 a 18 caracteres idênticos consecutivos. Este código é seguido pelo caractere a ser repetido, semelhante à codificação de comprimento de fileira, e resulta em um código de 2 bytes. Para representar pares freqüentes de caracteres, foram designados 48 códigos. Os pares de caracteres usados pela Infotron estão listados na Tabela 3.14. Com exceção de pares de ponto decimal, espaço e retorno de carro - avanço de linha, todos os outros pares incluem caracteres de letras maiúsculas e minúsculas.

Finalmente, os 16 códigos são designados para especificar quantos caracteres 4 a 19 estão no formato decimal compactado (meio-byte). Aqui, os caracteres são representados por códigos de 4 bits compactados em dois por byte de 8 bits. Além dos numéricos, o cifrão, o ponto final, a vírgula, o sinal de porcentagem, o sinal de diagonal e o espaço, se eles ocorrerem na cadeia, são retirados dos quatro bits principais e são incluídos no formato compactado.

Tabela 3.14 - Códigos de pares de caractere mais comuns comprimidos pela Infotron: tanto letra maiúscula, quanto minúscula.

S_	_T	IN	TE	AN
T_	_A	ED	ER	TI
E_	_N	AT	RF	ON
R_	_O	ES	TH	CRLF
D_	_I	SE	HE	
_				

Observação: _ representa um caractere de espaço. CRLF indica um retorno de carro seguido por um avanço de linha

Embora a eficácia da técnica de compressão empregada, obviamente dependa dos dados nos quais a técnica é aplicada, ao se usar técnicas múltiplas, aumenta a possibilidade de se usar uma técnica efetiva em uma parte do fluxo de dados. Durante os testes de compressão do adaptador de canal, a razão de compressão de até 1,8 foi observada pela Infotron, indicando que somente 55 por cento do fluxo de dados de entrada foi realmente transmitido.

Exemplos de programação

O programa PAIRC.BAS em BASIC, listado na Figura 3.36, foi desenvolvido para executar a compressão diatômica baseada na combinação de pares de caracteres de Jewell, previamente listada na Tabela 3.9.

Embora este exemplo de compressão diatômica fosse programado para usar a combinação de pares de caracteres de Jewell, pode-se modificá-lo facilmente para comprimir os dados com base no uso de outros pares de caracteres que podem refletir, de forma mais apropriada, os dados do leitor.

Semelhante a outras rotinas de compressão, previamente apresentadas neste capítulo, o programa de compressão diatômica foi desenvolvido, usando sub-rotinas ligadas entre si para fornecer módulos distintos de código, que possam ser facilmente analisados pelo leitor. Depois que o arquivo de dados é aberto na linha 100, é chamada a sub-rotina que começa na linha 400. Esta sub-rotina inicia os elementos do arranjo P$ com a combinação de pares de caracteres de Jewell, resultando em 25 pares de caracteres designados para o arranjo P$. O leitor pode mudar os pares de dados contidos nas linha 410 e 420 da sub-rotina; contudo, se o número de pares de dados for diferente de 25, os índices apropriados, no programa, devem ser mudados parar refletir o número real de pares. Além disso, a dimensão do arranjo P$ deve ser mudada para refletir o novo número de pares a ser usado na rotina de compressão diatômica. Assim, as linhas 400 e 425 necessitariam de modificação na sub-rotina previamente discutida, quando se entra com um novo conjunto de pares de caracteres nas linhas 410 e 420, cuja soma seja diferente de 25.

```
10 REM PAIRC.BAS PROGRAM
20 DIM O$(132)
30 WIDTH 80:CLS
40 '**********MAIN ROUTINE**********************
50 '* THIS ROUTINE READS RECORDS FROM AN ASCII *
60 '* FILE INTO A STRING CALLED X$ WHICH IS    *
70 '* THEN PASSED TO SUBROUTINES FOR COMPRESSION
80 '*********************************************
90 PRINT "ENTER ASCII FILENAME. EG, PAIR.DAT"
100 INPUT F$: OPEN F$ FOR INPUT AS #2
105 OPEN "PAIRC.DAT" FOR OUTPUT AS #3
110 PRINT "PATIENCE - INPUT PROCESSING"
115 GOSUB 400                    'PAUSE TO SET UP TABLE
120 IF EOF(2) THEN GOTO 9000
130 LINE INPUT #2, X$
140 N= LEN(X$)
150 GOSUB 180
160 GOSUB 900
170 GOTO 120
180 '*****DIATOMIC COMPRESSION SUBROUTINE*******
190 '* THIS ROUTINE PROCESSES RECORDS FROM X$  *
200 '* AND COMPRESSES OUT COMMON PAIRS         *
210 '* USING O$ AS THE OUTPUT BUFFER.          *
220 '*********************************************
230 I=1                          'RESET INDICES
240 FOR J= 1 TO N-1              'STEP THRU RECORD
250 A$= MID$(X$,J,2)             'EXTRACT A PAIR
260 FOR K = 1 TO 25              'SETUP PAIR TABLE LOOP
270 IF A$=P$(K) THEN GOSUB 350   'IS INPUT PAIR IN TABLE?
280 NEXT K                       'NO - TRY NEXT
290 IF M = 1 THEN 310            'IF MATCH FLAG SET?
300 O$(I) = MID$(A$,1,1)         'NO-STUFF 1ST CHAR IN BUFFER
310 I=I+1                        'BUMP INPUT STRING INDEX
320 M=0                          'RESET MATCH FLAG
330 NEXT J                       'GO BACK FOR MORE
340 RETURN                       'DONE
350 M=1                          'SET PAIR MATCH FLAG
355 '*********************************************************
360 'INSERT COMPRESSION NOTATION IN OUTPUT BUFFER
365 V = K + 224                  'INDEX OUT TO SUBSTITUTE CHAR
370 O$(I)=CHR$(V)                'INSERT PAIR SUBSTITUTION
380 J=J+1                        'FORCE INPUT SHIFT 2 OVER PAIR
390 K = 25                       'FORCE END OF PAIR SEARCH
395 RETURN                       'GO BACK FOR MORE
400 DIM P$(25)                   'JEWELL CHAR. COMBINATION PAIRS
410 DATA "E "," T",TH," A","S ",RE,IN,HE,ER," I"," O","N ",ES,
420 DATA " B",ON,"T ",TI,AN,"D ",AT,TE," C"," S",OR,"R "
425 FOR I = 1 TO 25              'SETUP PAIR TABLE
430 READ Z$                      'GET COMMON PAIR
440 P$(I) = Z$: NEXT I           'AND STUFF INTO PAIR TABLE
450 RETURN                       'DONE - TABLE COMPLETE
```

Figura 3.36 - A listagem do programa PAIRC.BAS.

```
900 '*****TALLY THE COMPRESSION COUNT & WRITE BUFFER******
910 '* DISPLAY BEFORE & AFTER RESULTS OF COMPRESSION     *
920 '* AND SHOW THE NET RESULTS OBTAINED BY EACH METHOD  *
930 '*************************************************
931 N1=N1+N                          'TALLY INPUT CHAR COUNT
932 T=N-I+1                          'NET DIFFERENCE IN BUFFERS
936 T1=T1+T                          'SAVE COUNT FOR SUMMARY
940 FOR I=1 TO J-1
950 PRINT #3, O$(I);
960 NEXT I
965 PRINT #3, ""
970 RETURN
1000 PRINT
1010 PRINT "**RUN-LENGTH ENCODING SAVED ";T;" CHARACTERS"
1020 RETURN
9000 CLOSE: OPEN F$ FOR INPUT AS #2
9010 PRINT "FILE ";F$;" BEFORE COMPRESSION:"
9020 LINE INPUT #2,X$
9030 IF EOF(2) THEN 9060
9040 PRINT X$
9050 GOTO 9020
9060 PRINT X$:OPEN "PAIRC.DAT" FOR INPUT AS #3
9070 PRINT "FILE ";F$;" AFTER COMPRESSION:"
9080 LINE INPUT #3,O$
9090 IF EOF(3) THEN 9998
9100 PRINT O$
9110 GOTO 9080
9998 PRINT O$:PRINT T1;" TOTAL CHARACTERS ELIMINATED FROM ";
9999 PRINT N1;"OR ";INT((T1/N1)*100);"%":CLOSE:END
```

Figura 3.36 - continuação

Depois que uma linha de dados é lida na linha 130, seu comprimento é determinado na linha 140. A sub-rotina chamada na linha 150, processa a linha de dados lida do arquivo, que começa na linha 230. Depois que os índices são reiniciados na linha 230, o loop FOR-NEXT delimitado pelas linhas 240 e 330 caminha pelo registro, extraindo pares de dados na linha 250. O loop FOR-NEXT interno, delimitado pelas linhas 260 e 280, compara o par extraído do registro na linha 250, com os pares contidos na tabela de pares, previamente montada pela sub-rotina na linha 400. O leitor deve observar que o limite exterior de 25 na linha 260, deve também ser mudado se o número de pares usados no programa for diferente daquele valor.

Se um par de caracteres extraído do registro combina com um par na tabela de pares, é chamada a sub-rotina na linha 350. A linha 350 usa a variável M para indicar que ocorreu uma combinação. Na linha 365, a variável V é definida como K + 224. Aqui, o valor de K é a posição na tabela de pares, onde o par extraído do registro

combinou. A razão por que foi somado 224 a este valor, deve-se à clareza da exibição dos resultados desta rotina de compressão. Isto é, os itálicos são impressos a partir do ASCII 225 em muitas impressoras, incluindo a impressora usada pelo autor. Assim, o par 'E espaço' é representado pelo itálico 'a', quando impresso, e assim por diante.

O caractere de substituição do par é inserido no elemento apropriado do arranjo 0$, como indicado na linha 370. Observe que J é incrementado de 1 na linha 380 para forçar um movimento na posição atual no registro de entrada. A seguir, a linha 390 define K com 25 para terminar a comparação de pares no loop FOR-NEXT, delimitado pelas linhas 260 e 280, das quais a rotina de compressão foi chamada e para a qual ela retorna pela execução da linha 395.

Já que a variável M foi ativada em 1 para indicar que ocorreu a combinação de pares, o término do loop FOR K faz com que a execução da linha 290 resulte em um desvio para a linha 310. Aqui, o índice usado para o arranjo 0$ é aumentado de um e o flag de combinação descansa em zero antes de terminar o loop.

A Figura 3.37 ilustra como a execução da rotina de compressão diatômica aparecerá em seu monitor, enquanto a Figura 3.38 ilustra a imagem da tela depois que ela foi 'descarregada' em uma impressora que imprime os valores ASCII acima de 255, como itálicos. Assim, alguns leitores podem querer usar a execução ilustrada na Figura 3.38, para comparar os caracteres de compressão em itálico, referentes aos dados originais com a combinação de pares de caractere de Jewell usados no programa. Já que um itálico minúsculo 'a' representa a primeira combinação de pares, enquanto um itálico 'b' representa a segunda combinação de pares e, assim por diante, deve ser mais fácil usar o segundo exemplo de execução do programa PAIRC.BAS, para os leitores que desejam seguir o fluxo lógico do programa, em detalhes.

Descompressão

A listagem do programa PAIRD.BAS está listada na Figura 3.39. Como indicado pela convenção de nomes usados neste livro, este programa executa a descompressão nos pares de caracteres previamente comprimidos.

```
ENTER ASCII FILENAME. EG, PAIR.DAT
? PAIR.DAT
PATIENCE - INPUT PROCESSING
FILE PAIR.DAT BEFORE COMPRESSION:
1 TO BE OR NOT TO BE THAT IS THE QUESTION
2 THE RAIN IN SPAIN FALLS MAINLY IN THE PLAIN
FILE PAIR.DAT AFTER COMPRESSION:
1ΓΟΠß· NO±TOΠßπJΩσπßQUØ≥≡
2Γ§ RAτΩ∞SPAτ FALLσMAτLYΩ∞πßPLAτ
 30  TOTAL CHARACTERS ELIMINATED FROM  91 OR  32 %
Ok


ENTER ASCII FILENAME. EG, PAIR.DAT
? PAIR.DAT
PATIENCE - INPUT PROCESSING
FILE PAIR.DAT BEFORE COMPRESSION:
1 TO BE OR NOT TO BE THAT IS THE QUESTION
2 THE RAIN IN SPAIN FALLS MAINLY IN THE PLAIN
FILE PAIR.DAT AFTER COMPRESSION:
1ΓΟΠß· NO±TOΠßπJΩσπßQUØ≥≡
2Γ§ RAτΩ∞SPAτ FALLσMAτLYΩ∞πßPLAτ
 29  TOTAL CHARACTERS ELIMINATED FROM  86 OR  33 %
Ok
```

Figura 3.37 - Execução de exemplo do programa PAIRC.BAS como mostrado em um monitor.

```
ENTER ASCII FILENAME. EG, PAIR.DAT
? PAIR.DAT
PATIENCE - INPUT PROCESSING
FILE PAIR.DAT BEFORE COMPRESSION:
1 TO BE OR NOT TO BE THAT IS THE QUESTION
2 THE RAIN IN SPAIN FALLS MAINLY IN THE PLAIN
FILE PAIR.DAT AFTER COMPRESSION:
1bOoay NOqTOoacujeca QUmrp
2bh RAgjISPAg FALLeMAgLYjlca PLAg
 30  TOTAL CHARACTERS ELIMINATED FROM  91 OR  32 %
```

Figura 3.38 - Execução de exemplo do programa PAIRC.BAS, quando impresso, usando uma impressora que mostra os caracteres ASCII acima de 224, como itálicos.

A partir do exame do programa listado na Figura 3.39, o leitor observará que a construção dos módulos de código para a descompressão, rigorosamente assemelha-se ao programa de compressão examinado previamente. Embora nosso objetivo de programação fosse fazer isto para facilitar uma comparação entre os programas, devido à relação entre a compressão e a descompressão, estas relações de codificação modular serão normalmente a regra e não a exceção.

Depois de abrir arquivos de entrada e saída, a sub-rotina, começando na linha 500, é chamada na linha 115 do programa.

```
10 REM PAIRD.BAS PROGRAM
20 DIM O$(132)
30 WIDTH 80:CLS
40 '**********MAIN ROUTINE**********************
50 '* THIS ROUTINE READS RECORDS FROM AN ASCII *
60 '* FILE INTO A STRING CALLED X$ WHICH IS    *
70 '* THEN PASSED TO DECOMPRESSION SUBROUTINE  *
80 '*********************************************
90 PRINT "ENTER ASCII FILENAME. EG, PAIRC.DAT"
100 INPUT F$: OPEN F$ FOR INPUT AS #2
105 OPEN "PAIRD.DAT" FOR OUTPUT AS #3
110 PRINT "PATIENCE - INPUT PROCESSING"
115 GOSUB 500
120 IF EOF(2) THEN GOTO 9000
130 LINE INPUT #2, X$
140 N= LEN(X$)
150 GOSUB 180
160 GOSUB 900
170 GOTO 120
180 '*****DIATOMIC   DECODING SUBROUTINE********
190 '* THIS ROUTINE PROCESSES RECORDS FROM X$  *
200 '* AND DECOMPRESSES PAIR-ENCODED CHARACTERS*
210 '* USING O$ AS THE OUTPUT BUFFER.          *
220 '*********************************************
230 K=1:J=1:V=0                    'RESET INDICES
240 FOR I= 1 TO N                  'STEP THRU RECORD
250 A$= MID$(X$,I,1)               'EXTRACT A CHAR
260 IF A$> CHR$(224) THEN 360      'COMPRESSED PAIR?
290 O$(J)=A$                       'STUFF IN OUTPUT BUFFER
300 J=J+1                          'BUMP BUFFER INDEX
310 NEXT I                         'GO BACK FOR MORE
320 RETURN                         'END OF STRING
355 '*****************************************************
360 'DECODE COMPRESSION NOTATION TO OUTPUT BUFFER
365 '*****************************************************
370 K= ASC(A$)                     'GET ORDINAL EQUIV.
380 K= K-224                       'SUBTRACT FOR INDEX
390 O$(J)= P$(K)                   'STUFF PAIR IN BUFFER
400 J= J+1                         'BUMP OUTPUT INDEX
405 V= V+1                         'SUM VARIABLE COUNT
410 GOTO 310                       'DONE
500 DIM P$(25)                     'JEWELL CHAR. COMBINATION PAIRS
510 DATA "E "," T",TH," A","S ",RE,IN,HE,ER," I"," O","N ",ES,
520 DATA " B",ON,"T ",TI,AN,"D ",AT,TE," C"," S",OR,"R "
530 FOR I = 1 TO 25                'SET UP PAIR TABLE
540 READ Z$                        'GET COMMON PAIR
550 P$(I) = Z$: NEXT I             'AND STUFF INTO PAIR TABLE
560 RETURN                         'DONE - TABLE COMPLETE
900 '*****TALLY THE DECOMPRESSION COUNT & WRITE BUFFER****
910 '* DISPLAY BEFORE & AFTER RESULTS OF DECOMPRESSION   *
920 '* AND SHOW THE NET RESULTS OBTAINED BY EACH METHOD  *
930 '*****************************************************
```

Figura 3.39 - A listagem do programa PAIRD.BAS.

```
931 N1=N1+N                              'TALLY INPUT CHAR COUNT
932 T=N-J+1+V                            'NET DIFFERENCE IN BUFFERS
936 T1=T1-T                              'SAVE COUNT FOR SUMMARY
940 FOR I= 1 TO J-1
950 PRINT #3, O$(I);
960 NEXT I
965 PRINT #3, ""
970 RETURN
9000 CLOSE: OPEN F$ FOR INPUT AS #2
9010 PRINT "FILE ";F$;" BEFORE DECOMPRESSION:"
9020 LINE INPUT #2,X$
9030 IF EOF(2) THEN 9060
9040 PRINT X$
9050 GOTO 9020
9060 PRINT X$:OPEN "PAIRD.DAT" FOR INPUT AS #3
9070 PRINT "FILE ";F$;" AFTER DECOMPRESSION:"
9080 LINE INPUT #3,O$
9090 IF EOF(3) THEN 9998
9100 PRINT O$
9110 GOTO 9080
9998 PRINT O$:PRINT ABS(T1);" TOTAL CHARACTERS INSERTED"
9999 CLOSE:END
```

Figura 3.39 - continuação

Esta sub-rotina simplesmente constroe a tabela P$ que conterá a combinação de pares de caractere de Jewell e que o programa procurará. Na linha 130, a familiar declaração LINE INPUT é usada para obter um registro do arquivo de entrada. A seguir, a linha 140 é empregada para determinar o comprimento do registro, enquanto a linha 150 chama a sub-rotina, começando na linha 180, que executa a descompressão real dos dados.

O loop FOR-NEXT, delimitado pelas linhas 240 e 310, pesquisa o registro previamente extraído do arquivo de entrada, caractere a caractere. Isto é realizado pelo uso da função MID$ na linha 250. Se o caractere extraído do registro for maior que o valor 224, supõe-se que ocorreu a compressão diatômica ou de pares. Neste exemplo de programa, esta suposição está fundamentada na escolha de todos os caracteres acima do ASCII 224, para representar um par de caracteres. Se o caractere extraído do registro for igual ou menor que o ASCII 224, este caractere não representa um par de caracteres previamente comprimidos. Assim, a linha 290 simplesmente coloca o caractere extraído em sua posição apropriada no buffer de saída.

Quando um caractere ASCII maior que 224 for encontrado, o desvio para a linha 360 no programa, resulta na descompressão

real de um par de caracteres previamente comprimidos. Na linha 370, o valor numérico do caractere que realmente representa um par de caracteres é obtido. A seguir, a linha 380 subtrai 224 do valor numérico do caractere, de maneira a obter o índice apropriado na tabela de pares (P$(25)). A linha 390 coloca no buffer de saída o par de caracteres, que foi previamente representado por um único caractere, enquanto as linhas 400 e 405 incrementam a posição de índice no buffer de saída e a variável V, que é somente empregada para computar a diferença de tamanho entre os buffers de entrada e de saída, não é necessária na descompressão.

A Figura 3.40 ilustra a execução do programa PAIRD.BAS, mostrando como apareceria em nosso monitor, usando o arquivo de dados PAIRC.DAT como entrada. O PAIRC.DAT foi criado pelo programa PAIRC.BAS. Assim, não é novidade que as duas linhas de dados comprimidos, na parte superior da Figura 3.40, combinem com as linhas 1 e 2 na parte inferior da Figura 3.38, enquanto as linhas 1 e 2 na parte inferior da Figura 3.40, combinem com as duas linhas na parte superior da Figura 3.38.

```
ENTER ASCII FILENAME. EG, PAIRC.DAT
? PAIRC.DAT
PATIENCE - INPUT PROCESSING
FILE PAIRC.DAT BEFORE DECOMPRESSION:
1ΓΟΠß· NO±TΟΠßπJΩσπßQUØ≥≡
2Γ§ RAτΩωSPAτ FALLσMAτLYΩωπßPLAτ
FILE PAIRC.DAT AFTER DECOMPRESSION:
1 TO BE OR NOT TO BE THAT IS THE QUESTION
2 THE RAIN IN SPAIN FALLS MAINLY IN THE PLAIN
 29  TOTAL CHARACTERS INSERTED
Ok
```

Figura 3.40 - Execução de exemplo do programa PAIRD.BAS.

Considerações na codificação

A execução da compressão diatômica está fundamentada na substituição de dois caracteres com um caractere especial de indicação de compressão. Assim, você deve usar um conjunto de caracteres onde os caracteres indefinidos estão disponíveis para substituir os pares de caracteres, ou você deve considerar a freqüência de ocorrência dos pares de caracteres versos a freqüência de ocorrência do caractere escolhido, usando a técnica de inserção e deleção. Com relação à segunda, vamos supor que você deseja aplicar a compressão diatômica e irá transmitir ou armazenar os

caracteres ASCII de 8 bits. Já que todos os caracteres naquele conjunto de caracteres são definidos, a única maneira de se efetivar a compressão diatômica é se a freqüência de ocorrência de um par de caracteres, for maior que duas vezes a freqüência de ocorrência do caractere escolhido para representar o par de caractere. Isto se deve ao fato de a técnica de inserção e deleção dobrar a freqüência de ocorrência do caractere escolhido, isto acontece porque cada ocorrência natural do caractere escolhido para representar o par é dobrada, já que a parte de inserção da técnica de inserção e deleção adiciona um caractere extra, sempre que o caractere escolhido for encontrado.

3.6 SUBSTITUIÇÃO DE PADRÕES

Esta técnica de compressão é basicamente uma forma sofisticada de codificação diatômica. Aqui, um código especial de caractere é substituído por um padrão de caractere predefinido. O emprego da técnica de compressão por substituição de padrões pode ser altamente vantajoso, quando você estiver transmitindo listagens de programa e outros tipos de arquivos de dados, contendo padrões de repetição conhecidos.

A vantagem oferecida pela substituição de padrões é melhor entendida ao examinar uma linguagem de nível mais alto, tal como o FORTRAN. Em qualquer programa em FORTRAN, há uma probabilidade bem alta de um ou mais tipos de declarações serem encontrados, contendo palavras chaves comuns, tais como: 'READ', 'WRITE' e 'FORMAT', entre outras. Em vez de se transmitir os caracteres destas palavras-chave, caractere por caractere, toda vez que aparecem, um dos caracteres não designados, do conjunto de caracteres empregado, pode ser substituído. Quando a substituição de padrões é aplicada ao texto de linguagem, as palavras ou frases-chave comuns podem, de forma semelhante, ser substituídas. Para a transmissão de um texto em Inglês, as palavras geralmente encontradas, tais como: 'and', 'the', 'that' e 'this' estariam entre as primeiras candidatas à substituição.

A tabela de padrões

Para empregar a substituição de padrões, é necessária uma tabela de padrões. Esta tabela contém um conjunto de argumentos em lista e um conjunto de valores de função. Cada valor de função é um caractere especial indicador de compressão que representa o valor comprimido de um determinado argumento (Aronson, 1977).

A Figura 3.41 mostra um exemplo do uso de uma tabela de padrões. Embora cada argumento em lista tenha comprimento semelhante, esta tabela pode ser expandida incluindo muitas entradas adicionais de vários comprimentos de caracteres. Cadeias de quatro, cinco, seis ou mais espaços em branco, poderiam, por exemplo, designar valores representados por caracteres especiais diferentes, bem como padrões de dados alfanuméricos.

NOW IS THE TIME FOR ALL GOOD MEN

Tabela de padrões

Argumentos de lista	Valores de função
THE	Sc_1
FOR	Sc_2
ALL	Sc_3

Fluxo de dados comprimidos

NOW IS Sc_1 TIME Sc_2 Sc_3 GOOD MEN

Figura 3.41 - Utilização da tabela de padrões. Em função da combinação do fluxo original de dados com os argumentos em lista, o valor de função apropriado é substituído. No exemplo acima, os caracteres especiais indicadores de compressão Sc_1, Sc_2 e Sc_3 são substitutos das palavras 'the', 'for' e 'all' à medida que são encontrados

Processo de codificação

Para obter o fluxo de dados comprimidos, os dados fontes devem estar divididos em argumentos distintos de busca, inicialmente igual ao argumento de menor tamanho na tabela de padrões. O argumento de busca é combinado com aqueles argumentos de listas de igual tamanho. Se a combinação for obtida, o valor de função, associado ao argumento de lista, substitui então aquela parte do fluxo original de dados, resultando na compressão de dados. Se nenhuma combinação for obtida, o tamanho do argumento de busca é aumentado para o tamanho do maior argumento em lista, ou série de argumentos em lista que for encontrada e o processo é repetido. Se depois de aumentar o tamanho do argumento de busca para o tamanho do maior argumento em lista não aparecer nenhuma combinação, o primeiro caractere da cadeia

original de dados é passado à cadeia de dados comprimidos e o processo é repetido, começando com o segundo caractere do fluxo original de dados.

Um segundo método de executar a substituição de padrões é resultante do uso de espaços em branco como delimitadores. O valor binário ou octal dos caracteres entre os espaços em branco, podem ser gerados e comparados com os valores binários ou octais na parte dos argumentos em lista da tabela de padrões. Este processo simplifica a busca de uma longa lista de argumentos e minimiza o tempo de processamento necessário para codificar os padrões.

Padrões nas linguagens de programação

Devido à utilização de palavras-chave ou palavras reservadas na maioria das linguagens de programação, a substituição de padrões é, freqüentemente, uma técnica de compressão muito eficaz para armazenar ou transmitir arquivos de programa. Já que o número de palavras-chave ou palavras reservadas em uma linguagem de programação pode chegar a diversas centenas, uma seqüência de 2 bytes pode ser empregada para representar cada substituição de padrões de palavra-chave. Aqui, o primeiro byte ou caractere seria usado para indicar que ocorreu a substituição de padrões, enquanto o caractere seguinte indicaria o padrão real que substituiu a palavra-chave ou a palavra reservada. Para ilustrar este conceito com maiores detalhes, vamos supor que a versão do BASIC com que estamos trabalhando é limitada a oito palavras-chave. A Tabela 3.15 lista estas palavras-chave e como poderiam ser construídos os valores de função equivalentes contidos na tabela de padrões.

Tabela 3.15 - Tabela de padrões da linguagem BASIC

Palavras-chave	Valores de função
END	$1
GOTO	$2
IF	$3
INPUT	$4
LET	$5
PRINT	$6
REM	$7
THEN	$8

Para um melhor entendimento, o cifrão ($) foi empregado como um caractere de indicação de compressão, na Tabela 3.15, embora obviamente qualquer caractere do conjunto de caracteres pudesse ser usado. De preferência, você deve selecionar um caractere que seja raramente, ou melhor ainda, nunca usado. Já que há sempre a possibilidade de o caractere ocorrer em um programa em BASIC, você pode substituir cada ocorrência de caractere padrão de indicação de compressão pela duplicação daquele caractere, quando ele for encontrado. Então a rotina de descompressão menosprezaria todo o caractere padrão de indicação de compressão duplicado. A compressão de um pequeno programa em BASIC está ilustrada na Figura 3.42, baseada no emprego da substituição de padrões, que é, na realidade, a substituição das palavras-chave do BASIC. Observe que a tabela de padrões contida na Tabela 3.15, foi usada no processo de compressão. Já que a maioria das linguagens BASIC exigem que as palavras-chave sejam delimitadas por espaços, presumimos que as palavras-chave da Tabela 3.15 continham espaços em branco antes e depois delas, permitindo que o valor funcional, em substituição à palavra-chave, seja uma subtituição mais eficiente. Usando o método de substituição, 25 espaços, além de 26 outros caracteres são eliminados do programa, enquanto dois caracteres são adicionados. Os caracteres adicionais devem-se à substituição da ocorrência natural do caractere $ no programa, pela seqüência especial $$ nas linhas 180 e 190.

Programa em BASIC	*Programa comprimido*
100 REM COMMISSION CALCULATION	100$7COMMISSION CALCULATION
110 PRINT "ENTER SALE PRICE"	110$6"ENTER SALE PRICE"
120 INPUT W	120$4W
130 PRINT "ENTER NUMBER SOLD"	130$6"ENTER NUMBER SOLD"
140 INPUT N	140$4N
150 LET C=W*N*.0875	150$5C=W*N*.0875
160 PRINT "COMMISSION=";C	160$6"COMMISSION=";C
170 PRINT "ANOTHER CALCULATION-Y/N"	170$6"ANOTHER CALCULATION-Y/N"
180 INPUT A$	180$4A$$
190 IF A$ <> "Y" THEN 210	190$3A$$<>"Y"$8210
200 GOTO 110	200$2110
210 END	210$1

Figura 3.42 - Comprimindo um programa em BASIC.

Embora a redução global de dados, que, neste exemplo, foi de aproximadamente 20 por cento, possa não parecer significativa, deve ser observado que o esforço real envolvido para comprimir dados, usando a substituição de padrões, é pequeno. Para aumentar a redução de dados resultante da compressão, é geralmente

necessária a aplicação de diversas técnicas de compressão nos seus dados. Neste exemplo, em particular, você pode, primeiro, preprocessar arquivos de programação, através da utilização da compressão de substituição de padrões. Em seguida, você poderia codificar, estatisticamente, os dados comprimidos resultantes. Já que o processo de codificação estatística resulta na substituição de caracteres que ocorrem freqüentemente por pequenas seqüências de bit, a codificação estatística dos dados onde palavras-chave foram previamente substituídas por pequenos padrões, é mais eficaz do que a codificação estatística dos dados originais. Como exemplo, uma seqüência de 5 bits pode ser necessária para representar a palavra-chave PRINT; contudo, uma pequena seqüência de bits seria necessária para representar a seqüência de caracteres $6 que foi substituta da palavra-chave. Consulte o capítulo 4 para obter maiores informações sobre a codificação estatística.

3.7 CODIFICAÇÃO RELATIVA

A codificação relativa é uma técnica de compressão que não é normalmente aplicável à transmissão de arquivos de dados convencionais. Este tipo de compressão é eficazmente empregado quando há seqüências de fileiras no fluxo original de dados, que variam ligeiramente uma da outra, ou quando as seqüências de fileiras podem ser quebradas em padrões relativos a cada uma delas. Um exemplo do primeiro caso são os dados de telemetria, e do segundo caso são os padrões de bit das máquinas de fac-símile digitais.

Compressão de telemetria

Na geração de dados de telemetria, um dispositivo sensor é usado para registrar medidas em intervalos predefinidos. Estas medidas são, então transmitidas para a central para processamento adicional. Um exemplo de sinais de telemetria são as diversas sondas espaciais que transmitem leituras de temperatura, análises do espectro de cores e outros dados, através do comando proveniente das estações em terra ou através de intervalos de tempo predefinidos. Normalmente, os sinais de telemetria contêm uma seqüência de campos numéricos, que consistem de subseqüências ou fileiras de numéricos que variam ligeiramente uma da outra, como ilustrado na parte superior da Figura 3.43.

Medidas originais de telemetria

46 46 46.1 46.1 46.1 46 46 46 46.1 46.1 46.1 46.2

Codificação relativa

46 0.1 0 0 —.1 0 0 .1 0 0 .1

Figura 3.43 - Processo de codificação relativa. Os sinais de telemetria geralmente consistem de uma seqüência de numéricos, que varia ligeiramente um do outro durante um determinado intervalo de tempo.

Antes da transmissão real de dados, a compressão ocorre para reduzir a quantia total de dados necessária para representar as medidas gravadas. Toda a medida, com exceção da primeira, é codificada com a diferença relativa entre ela e a medida anterior, enquanto o valor absoluto do aumento for menor que algum valor predeterminado. Isto é mostrado na parte inferior da Figura 3.43. Neste processo de codificação relativa, se o aumento for maior que este valor, um caractere especial é inserido para indicar que o valor específico na posição não está disponível, ou que o caractere especial poderia ser seguido pela medida que está fora da faixa limite. Isto limita as amplas flutuações, mas é uma desvantagem do uso desta técnica. Outra desvantagem é que se os valores de dados variam consistentemente, tanto para dentro quanto para fora da faixa limite da codificação relativa, e assim que a combinação do caractere especial e do valor real for transmitida, isto causará uma expansão em vez de uma compressão do fluxo de dados.

Técnicas adicionais podem ser empregadas para obter um grau maior de compressão, dependendo das medidas originais de telemetria e dos dados resultantes devido ao processo de codificação relativa. Na parte superior da Figura 3.43, as medidas originais de telemetria ilustradas são compostas de 38 caracteres, incluindo os numéricos e os pontos decimais. Como um resultado do processo de codificação relativa, o número de numéricos e caracteres de pontos decimais foi reduzido para 18. Pela incorporação de uma segunda técnica de compressão, o número de caracteres usados para representar o processo de codificação relativa pode ser reduzido ainda mais. Um método que poderia ser usado é o processo de compactação de meio byte, onde cada dígito numérico fica sem os seus primeiros 4 bits e são compactados dois por caractere. Se

usarmos uma representação de 4 bits para o ponto decimal e sinal de menos, a compactação de meio byte resultará na transmissão de nove bytes de 8 bits de dados. Assim, enquanto o processo de codificação relativa resultou em uma razão de compressão de 2,24 (38/17), recomprimir os resultados de codificação relativa empregando a técnica de compactação de meio byte, aproximadamente, dobra a razão de compressão para 4,223 (38/9).

Em vez do processo de compactação de meio byte, acima ilustrado, ser a técnica combinada ou para a segunda compressão, outras técnicas poderiam ser empregadas, com resultados que dependem da variação das medidas originais de telemetria. Se as medidas originais de telemetria indicaram um 46 estável, para o intervalo de tempo do exemplo, o processo de codificação relativa resultaria em uma longa cadeia de zeros depois do indicador de valor 46. Para esta situação, a codificação de comprimento de fileira seria mais eficaz como uma segunda técnica de compressão.

Fac-símile digital

Diversas técnicas de codificação relativas podem ser empregadas para comprimir os dados de fac-símile digital. Antes de discutir estas técnicas, uma revisão dos elementos da tecnologia de fac-símile é necessária.

Os sistemas de fac-símile usam o conceito básico de varredura - normalmente linha a linha - para criar um fluxo de informação referente aos pontos claros e escuros da pequena área, sendo varrida em um determinado momento. O fluxo resultante de informações é, então, transmitido e usado para orientar o dispositivo de reprodução de imagem em um receptor de fac-símile, onde as informações originais serão reproduzidas. Em geral, a operação de um dispositivo de fac-símile é bem semelhante à tecnologia empregada na televisão, onde no sistema doméstico de televisão dos EUA, são usadas 525 linhas para reproduzir imagens. Para os sistemas de fac-símile, a qualidade depende da precisão da varredura. Para se reproduzir uma página de material datilografado com sucesso, normalmente, são necessárias cerca de 100 linhas de varredura por polegada. Assim, uma folha de tamanho normal, 8,5 x 11 polegadas, varrida longitudinalmente, necessitaria de aproximadamente 850 linhas de varredura. Cada linha de varredura, por sua vez, é composta de aproximadamente 1.730 elementos de imagens (pixels, ou pels), resultando em, aproximadamente, 1 milhão de bits para uma folha de papel de 8,5 x 11 polegadas. Para

transmitir estes dados a 4.800 bps, sem compressão, seriam necessários 209 s ou, aproximadamente, 3,5 min.

Para os sistemas de fac-símile, o grau de compressão teoricamente obtido é normalmente muito alto para uma mensagem típica de fac-símile. Como exemplo, considere uma memorando datilografado contendo 500 caracteres. Na transmissão convencional de dados, cada caractere pode ser representado e transmitido por 8 bits. Assim, a mensagem inteira poderia ser transmitida, ignorando os caracteres de controle, por 500 * 8 ou 4.000 bits de informação. Se transmitida a 4.800 bps, o tempo total de transmissão seria menor que 1 s. Em compensação, a mesma mensagem enviada por um fac-símile convencional exigiria a transmissão de quase 1 milhão de bits e levaria cerca de 3,5 min sem a compressão de dados, uma diferença de aproximadamente 270 para 1, entre o código convencional de fac-símile e a transmissão de caracteres.

Técnicas de fac-símile

Uma das primeiras técnicas de compressão para fac-símile foi a codificação de comprimento de fileira. Aqui, a transmissão da varredura digital de linha é substituída pela transmissão da contagem de quantidade de cada uma das sucessivas fileiras de pixels em branco e preto que foram varridos.

Já que a grande maioria de documentos a serem varridos contém uma quantidade de pixels brancos muito maior do que pretos, transmitir a diferença entre as varreduras pode reduzir, de forma significativa, a quantidade de dados a serem transmitidos. Neste método de compressão, uma varredura completa é mantida na área de memória do dispositivo e comparada com a varredura subseqüente. A transmissão, somente das mudanças relativas à varredura anterior, resulta em um processo de compressão relativa. Uma vez que as diferenças entre a primeira e a segunda varreduras são transmitidas, a primeira varredura é removida da memória e substituída pela segunda varredura. A seguir, uma terceira varredura é comparada com a segunda, agora localizada na memória. O fluxograma mostrando os passos necessários para este tipo de processo de codificação relativa, está ilustrado na Figura 3.44.

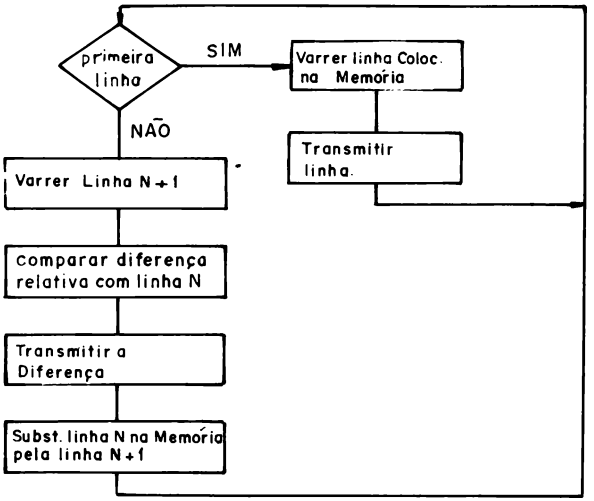


Figura 3.44 - Processo de codificação relativa do Fac-símile

Na Figura 3.45, é apresentada uma parte das mudanças relativas resultante da comparação de duas linhas de varredura. Diversos métodos podem ser usados para indicar as mudanças relativas entre as linhas de varredura Nésima e Nésima + 1. Um método consiste de indicar a posição da mudança, que é normalmente chamada de identificação posicional. Aqui, a posição de cada mudança relativa é indicada de acordo com o primeiro pixel da linha. Se houver muitas mudanças consecutivas, a transmissão de cada posição individual poderia exigir mais bits de dados, do que a transmissão da linha original antes da comparação com a linha anterior. Para tirar vantagem das sucessivas mudanças relativas entre as linhas de varreduras, o indicador de posição pode ser seguido por uma contagem de quantidade que contém o número de sucessivas mudanças relativas. Isto encontra-se ilustrado na Figura 3.46, onde a tabela na parte superior da figura, tabula a posição inicial da mudança relativa entre as varreduras de linha e o número de subseqüentes mudanças relativas. A seqüência de transmissão é indicada na parte inferior daquela figura. De

Linha de varredura N-ésima

. . . 0 0 0 0 0 0 1 1 1 0 0 0 1 1 0 . .

Linha de varredura (N + 1)-ésima

. . . 0 0 0 0 0 1 1 1 1 0 0 1 1 0 0 . .

Mudança relativa

. . . - - - - - X - - - - - X - X - - - - - . . .

Figura 3.45 - Mudança relativa. Para indicar as mudanças relativas entre as linhas de varredura, diversos métodos podem ser empregados, incluindo a identificação por posição e por deslocamento.

POSIÇÃO INICIAL DA MUDANÇA RELATIVA	NÚMERO DE MUDANÇAS RELATIVAS SUCESSIVAS
40	6
80	20
175	4
350	31
480	8
930	14
1250	16
1310	5
1340	4

Dados transmitidos

40	6	80	20	175	4	350	31	480	8	930	14	1250	16	1310	5	1340

Figura 3.46 - Transmitindo informações posicionais. Usando o processo de posicionamento relativo, a posição inicial de cada mudança relativa é seguida pelo número de mudanças relativas sucessivas.

acordo com os padrões para fac-símile digital da Consultive Committee for International Telephone e Telegraph (CCITT), órgão regulador da telefonia, há 1728 elementos de imagem ou pontos a serem lidos pelo scanner ao longo da largura de um documento de 21,5 cm de largura. Devido ao grande número de posições, a transmis-

são de informações posicionais pode rapidamente aumentar em duração, especialmente quando ocorrem várias mudanças relativas no final da linha de varredura. Um método usado para aliviar este aumento de 'fim de linha' é através do uso da notação de deslocamento. Assim como na notação posicional, são computadas inicialmente as mudanças relativas entre as linhas de varredura. Então, em vez de transmitir todas as posições iniciais das mudanças relativas e o número de mudanças sucessivas, como ilustrado na Figura 3.46, somente a primeira posição inicial é transmitida. Depois disso, é transmitido o deslocamento entre as mudanças relativas. Este método de deslocamento pode incluir a transmissão de informações de mudança relativa sucessiva e encontra-se ilustrado na Figura 3.47. Esta figura está fundamentada nos dados fornecidos na parte tabulada da Figura 3.46. Ao comparar os métodos ilustrados de deslocamento e posicional, vemos que o método posicional requer 41 caracteres numéricos, enquanto o método de deslocamento pode ser realizado com o uso de 35 destes caracteres. Se os numéricos forem compactados dois por byte, então a técnica de deslocamento resultará em 140 bits, sendo necessários para representar os 1728 pontos do exemplo, já o método posicional necessitará de 164 bits.

Embora a transmissão tanto das informações de deslocamento, quanto das informações posicionais seja aplicada em várias técnicas, usadas para comprimir as informações de fac-símile, um método mais eficiente é obtido ao se designar códigos predefinidos para diferentes fileiras de pixels em branco e preto. Os códigos são, então, substitutos de cada seqüência de fileiras de pixels encontrada em cada linha varredura. Esta técnica é fundamentada na versão modificada da codificação Huffman e está detalhada no capítulo 4.

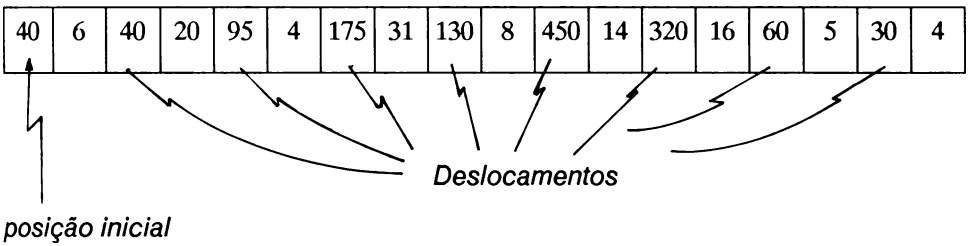


Figura 3.47 - Transmitindo informações de deslocamento. Outra técnica de codificação relativa resulta na transmissão de informações de deslocamento.

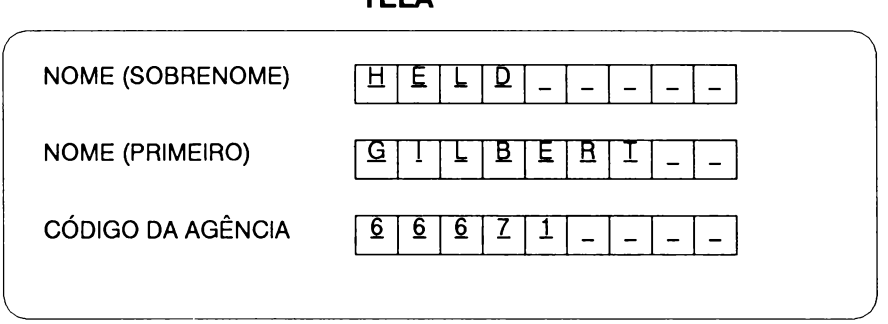
3.8 OPERAÇÃO NO MODO DE FORMULÁRIOS

A operação no modo de formulários é um método de compressão que pode ser empregado, quando os dados têm de ser comunicados de/para um display CRT em uma série predefinida de formulários. Quando se trabalha no modo de formulários, o display pode ser usado por um tipo de operação de preenchimento de espaços em branco. Neste modo de operação, dois tipos básicos de dados são mostrados - informações protegidas e informações variáveis. As informações fixas ou protegidas correspondem às informações pré-impressas de um campo de dados em um formulário impresso padrão, tal como: nome, endereço, número do seguro social e outros tipos semelhantes de informações. Estas informações, quando se está operando no modo de formulários, não são limpas quando a tela é apagada, não são transmitidas ao processador central e não são alteráveis por digitação acidental no teclado. Cada campo fixo é a metade de um par de campo, sendo que a outra metade corresponde ao campo variável. Assim, no modo de formulários, o campo fixo pode ser visto como a pergunta, enquanto as informações digitadas no campo variável correspondente podem ser consideradas como a resposta.

Um exemplo de digitação de dados no modo de formulários é apresentado na Figura 3.48. Aqui, os espaços em branco indicam as posições adicionais disponíveis para entrada de dados nos campos variáveis.

Ao se trabalhar com o modo de formulários, o operador indica o formulário que ele ou ela deseja completar e aquele formulário é transmitido do computador ao display do terminal, ou é gerado localmente pela memória do terminal ou do dispositivo periférico acoplado ao terminal. Os campos fixos são precedidos pelos caracteres 'FS' (iniciar campo fixo), enquanto os campos variáveis são precedidos por um caractere 'GS' (começar um campo variável) e de parâmetro. O caractere de parâmetro é usado para definir as operações admissíveis dentro de um campo variável, tais como: somente numérico, somente alfabético, alfanumérico, transmissão de inibição e etc. A seqüência exata dos caracteres GS e FS, bem como a configuração de bits do caractere de parâmetro para definir as operações admissíveis, dependem do programa do terminal. Quando se está no modo de formulários, certas operações de teclado são geralmente mudadas a partir das operações do modo

normal. Como um exemplo, a tecla TAB, na maioria dos terminais, permite que o operador movimente o cursor (marcador da posição de entrada de dados) para a primeira posição de caractere do próximo campo de variável que está em seqüência, permitindo um rápido salto sobre os campos de variáveis para os quais não se precisa entrar com nenhum dado (Peterson, Bitner e Howard, 1978).



Transmissão no modo de formulários

HELD $\overset{H}{T}$ GILBERT $\overset{H}{T}$ 66671

$\overset{H}{T}$ é o caractere de tabulação horizontal

Figura 3.48 - Entrada de dados no modo de formulários.

Transmissão

A transmissão de dados no modo de formulários é geralmente executada tela a tela. Quando o operador solta a tecla TRANSMIT, somente os dados previamente digitados nos campos variáveis são transmitidos, eliminando-se com todos os espaços em branco posteriores.

Aqui, a transmissão pode ocorrer on-line com o computador, ou pode ser feita para uma das unidades periféricas do terminal. Caso ocorra a segunda hipótese, um grande número de telas do terminal pode ser colocado em lote em um dispositivo periférico, tais como:

fita cassete ou disco flexível para então se transmitir para o computador de uma só vez. Usando esta combinação de codificação no modo de formulários e armazenamento off-line para transmissão de telas em lotes de informações, os recursos do sistema de computação, na forma de vias de acesso ao computador e exigências de linha, podem ser reduzidos ou usados de forma mais eficiente. Ao reduzir o tempo de transmissão necessário para enviar lotes de informações de tela, pode ser possível uma redução no número de vias de acesso ao computador, necessárias para operar terminais remotos.

Com relação ao uso mais eficiente da linha, vamos considerar uma situação onde 10 terminais operam no ambiente "poll and select" (consulta e seleção) conectados ao computador central via uma unidade de compartilhamento de modem e via um modem, como ilustrado na Figura 3.49. Na configuração ilustrada, todos os terminais, exceto aquele que está transmitindo ou recebendo dados, são bloqueados durante a transmissão. Normalmente, são transmitidos blocos de dados de até 1920 caracteres (80 x 24), que é o tamanho da tela. A transmissão de um bloco de 1920 caracteres de 8 bits, a 4.800 bps, necessitaria de 3,2 s para preencher totalmente uma tela. Se os 10 terminais estivessem conectados à unidade de compartilhamento de modem, com uma seqüência consulta "polling" circular e todos os operadores transmitissem ou recebecem uma tela completa de dados, seriam necessários 32 segundos, sem levar em consideração o baixo desempenho da transmissão, até que o primeiro operador de terminal pudesse novamente transmitir ou receber informações.

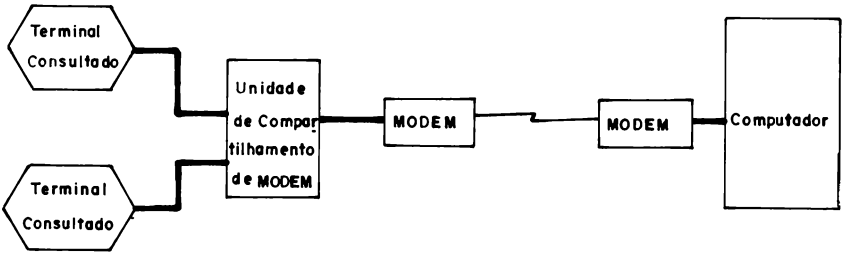


Figura 3.49 - A codificação no modo de formulários aumenta a disponibilidade da linha. A codificação no modo de formulários reduz o tempo de consulta ou seleção exigido por cada terminal, permitindo que mais terminais sejam conectados à linha compartilhada, ou que haja um aumento na produtividade dos terminais existentes, que compartilham a linha.

Assim, ao reduzir o número de caracteres transmitidos e recebidos através do emprego da codificação no modo de formulários, pode-se diminuir o tempo de resposta dos terminais existentes. ou permitir que terminais adicionais sejam agrupados, sem aumentar os tempos globais de resposta.

Voltando ao exemplo da Figura 3.48, o caractere 'HT' (tabulação horizontal) é normalmente usado como um separador de campo variável, resultando na mensagem transmitida, indicada na parte inferior daquela ilustração. Se o máximo de oito caracteres podem ser digitados em cada um dos três campos de variáveis, mostrados na Figura 3.48, serão transmitidos ao computador no máximo 26 caracteres (24 caracteres de dados e 2 caracteres de tabulação horizontal) para cada formulário completado. Este método de entrada de dados no modo de formulários deve ser comparado com o método convencional de compartilhamento de tempo, como na Figura 3.50. Aqui, as mensagens ENTRA NOME (SOBRENOME), NOME (PRIMEIRO), CODIGO DA AGENCIA servem como um indicador de campo variável, indicando ao operador do terminal que dados devem ser digitados. O caractere de retorno de carro (C/R) age como um caractere de terminação de linha; contudo, se os dados são digitados incorretamente, tais como caracteres alfabéticos em um campo totalmente numérico, os dados devem, primeiramente, ser enviados ao computador para o processamento onde determinará que um erro ocorreu. Nestes casos, o computador transmitiria uma mensagem de erro ao operador do terminal que, então, redigitaria, corretamente, a linha completa e retransmitiria os dados. Em contrapartida, ao usar um terminal inteligente e a operação no modo de formulários, a operação de entrada de dados pode ser preprocessada e estes erros corrigidos antes da transmissão.

Em comparação com o caso do operador soltando a tecla TRANSMIT no terminal e tendo o método de operação no modo de formulários, transmitindo e limpando os campos variáveis, de maneira que os novos dados possam ser digitados, o método convencional de compartilhamento de tempo requer que o programa oriente o operador a determinar se devem ser digitados mais dados. A seqüência MORE? e YES (C/R), na Figura 3.50, adiciona caracteres suplementares, além da mensagem repetida, usada como um indicador de campo variável. Ao comparar o exemplo de entrada de dados no modo de formulários, com o exemplo de entrada de dados no modo convencional de compartilhamento de tempo, 18 caracteres são necessários para o primeiro caso, enquanto 65 caracteres são necessários para o segundo caso, sem contar os caracteres de avanço de linha e de retorno de carro, que são

necessários para o modo convencional de compartilhamento de tempo.

ENTER NOME (SOBRENOME), NOME (PRIMEIRO), CODIGO DE AGENCIA

HELD, GILBERT, 6671 C/R

MORE?
YES C/R

ENTER NOME (SOBRENOME), NOME (PRIMEIRO), CODIGO DE AGENCIA

Figura 3.50 - Entrada de dados no compartilhamento de tempo convencional. No compartilhamento de tempo convencional, as mensagens do prompt solicitando dados, bem como as respostas do usuário são transmitidas.

Método modificado de modo de formulários

Podemos implementar um método modificado de compressão no modo de formulários, ao tratar uma tela do terminal como uma coleção de formulários. Para assim fazê-lo, podemos subdividir uma tela do terminal em n segmentos ou formulários, como ilustrado na Figura 3.51. Então, podemos manipular cada segmento como se fosse um formulário individual e assim reduzir a transmissão entre o dispositivo terminal e o computador.

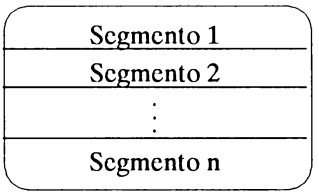


Figura 3.51 - Tratando a tela como uma coleção de formulários. A tela do terminal pode ser subdividida em n segmentos, com cada segmento sendo manipulado como se representasse um formulário. Em seguida, somente será transmitido o conteúdo do segmento, se ocorreram mudanças ao segmento.

A chave para a utilização do método modificado de operação no modo de formulários, descrito previamente, é a atribuição de um 'marcador de dados' para cada segmento de tela. Ao modificar o marcador de dados cada vez que um segmento de tela é alterado, permite-se que este marcador de dados sirva como um indicador se a tela deve ou não ser transmitida. Para ilustrar o potencial deste método, quanto à redução dos dados necessários para atualizar as telas dos terminais, vamos supor que uma tela de terminal com 80 x 25 caracteres como uma tela típica de IBM PC, é subdividida em segmentos de 40 caracteres por 5 linhas. Fazendo isto, a tela seria subdividida em um total de 10 segmentos que poderiam ser numerados como indicado na Figura 3.52.

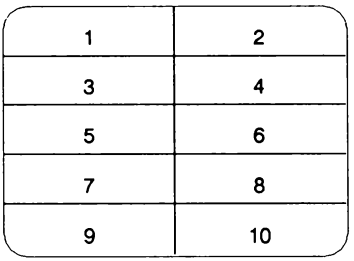


Figura 3.52 - Exemplo de segmentação de tela. Ao subdividir a tela do terminal em 10 segmentos de 40 caracteres por 5 linhas, as atualizações de tela podem ser implementadas através do incremento de 200 caracteres.

Quando um operador de terminal atualiza a tela, o terminal ou o microcomputador acompanha as mudanças feitas em cada segmento, mas não transmite quaisquer dados, até que uma tecla ENTER, RETURN, função de programa (PF) ou auxílio de programa (PA) seja pressionada. No momento em que o terminal ou o microcomputador transmite o marcador de dados de cada segmento, ele indicará se o segmento foi modificado ou não. Se o segmento foi modificado, o marcador de dados é, então, seguido por 200 caracteres contidos no segmento. Através da implementação desta técnica de compressão, torna-se possível transmitir as atualizações de tela inteira, através de um pequeno segmento com algumas centenas de caracteres, incluindo o marcador modificado de dados, que indica que o segmento foi modificado e que somente aquele segmento foi modificado.

Para ilustrar um método pelo qual uma tela de terminal deve ser segmentada, vamos supor que estamos usando um conjunto convencional de caracteres ASCII de 7 níveis. Poderíamos ativar o bit 8 para indicar, que ele é um caractere marcador de dados, que, de fato, nos permite usar até 128 caracteres únicos para representar informações diferentes referentes aos segmentos de tela.

A Figura 3.53 ilustra um formato possível para a construção de um caractere de marcador de dados. Aqui, a ativação da posição do bit 7 indicaria se o segmento que o marcador de dados representa, mudou ou não desde a última vez que uma das teclas ENTER, RETURN, PF ou PA foi pressionada. Os bits de 3 a 6 identificariam o segmento com que o marcador de dados é associado, enquanto o bit 1 seria usado para indicar se o segmento identificado pelo marcador de dados foi o último a ser mudado.

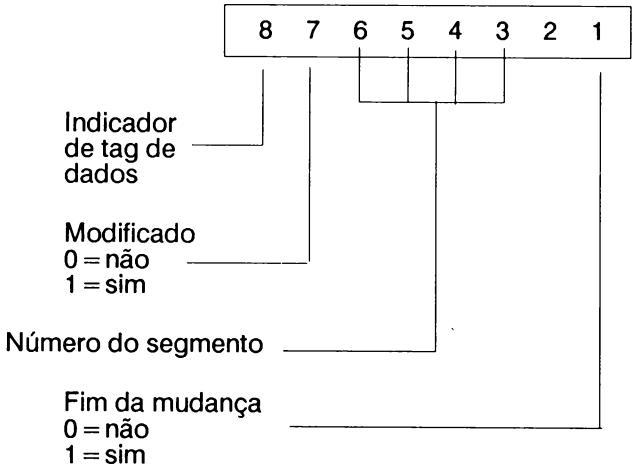


Figura 3.53 - Formato possível de marcador de dados.

Se o bit de fim de mudança estiver ativado com 0, os caracteres no segmento seguiriam o caractere de marcador de dados. Se o bit de fim de mudança estiver ativado com 1, isto indicaria que nenhum caractere adicional seguiria os caracteres que definem o segmento modificado, identificado pelo caractere de marcador de dados.

Embora a discussão anterior se preocupou com o uso de um conjunto de caracteres, do qual até 128 caracteres poderiam ser definidos para representar as informações do marcador de dados, como você poderia realizar um esquema de compressão semelhan-

te, usando EBCDIC, já que a maioria das operações de tela inteira nos terminais usa aquele conjunto de caracteres? Como discutido previamente no capítulo 2, há muitas representações de caracteres EBCDIC de 8 bits que são indefinidas. Assim, você poderia implementar o método modificado de compressão no modo de formulários previamente descrito, selecionando um caractere indefinido para representar o caractere marcador de dados. Então, você poderia definir um formato, no qual o caractere marcador de dados fosse seguido por outro caractere que indicaria se o segmento foi ou não modificado. Se o segmento foi modificado, você poderia então fazer com que os caracteres que representam o segmento sigam o caractere usado, para indicar um segmento modificado. Observe que ao usar este método, você pode construir seu formato, de maneira que você não transmitiria o número do segmento. Assim, se somente o segmento 5 mudou, você teria de transmitir quatro marcadores de dados para representar os segmentos de 1 a 4, sem que cada caractere marcador de dados, tenha em seguida um caractere indicando que o segmento foi mudado. Em comparação, ao usar o exemplo anterior, você poderia transmitir o marcador de dados com os bits de 3 a 6 ativados com o valor 5, para indicar que o segmento 5 mudou. Sem levar em consideração o método usado, o importante aspecto desta técnica é que para a maioria das aplicações interativas de terminal, somente uma pequena porção da tela é atualizada. Assim, ao subdividir a tela por segmentos, e somente transmitir aqueles segmentos que mudaram desde que uma tecla foi pressionada, podemos aumentar, significativamente, a eficiência do sistema de transmissão.

CAP. 04

CODIFICAÇÃO ESTATÍSTICA

Um elemento comum das oito técnicas de compressão de dados abordadas no Capítulo 3, é o fato de todas operarem com códigos de caracteres com um número fixo de bits. Em comparação com aqueles métodos de compressão, a codificação estatística se beneficia das probabilidades de ocorrência de caracteres únicos e grupos de caracteres, de maneira que códigos menores podem ser usados para representar caracteres freqüentemente usados ou grupos de caracteres, ao passo que códigos maiores são usados para representar caracteres encontrados com menos freqüência e grupos de caracteres.

O processo de codificação estatística pode ser usado para obter uma minimização do comprimento médio de código dos dados codificados, de uma forma semelhante àquela no qual o Morse selecionava as representações por ponto e traço para os caracteres, de maneira que um único ponto era usado para representar a letra E, que é o caractere mais freqüentemente encontrado na língua inglesa, enquanto grandes cadeias de pontos e traços foram usados para representar caracteres que aparecem com menos freqüência. A técnica de codificação Huffman, a codificação Shannon-Fano e o método de codificação em cadeia Lempel-Ziv, entre outros, fazem parte da classe de métodos de compressão estatística.

Neste capítulo, voltaremos nossa atenção na variedade de métodos de compressão de dados que estão fundamentados na codificação estatística de caracteres e cadeias. Começaremos a examinar o uso das tabelas de compressão estática e então a investigar a construção e operação de tabelas dinâmicas. Antes de

discorrermos sobre estas técnicas, detalhadamente, faremos uma revisão de alguns conceitos básicos de teoria da informação. Estes conceitos fornecerão um entendimento de como a redundância pode ser estatisticamente reduzida.

4.1 TEORIA DA INFORMAÇÃO

Para um sistema capaz de transmitir em n níveis discretos e em intervalos de λ s, o número de diferentes combinações de sinais em Ts é $n^{T/\lambda}$. Já que as informações são proporcionais ao tempo de transmissão, podemos pegar o logaritmo de $n^{T/\lambda}$, para obter as informações transmitidas em Ts, sendo proporcionais ao $(T/\lambda) \log n$.

O fator de proporcionalidade dependerá da base do logaritmo usado. A escolha mais comum está sendo a base 2. Isto resulta na unidade de informação H tornando

$$H = \frac{T}{\lambda} \log_2 n$$

A unidade de informação definida na forma anterior é conhecida como bit ou dígito binário. Para a transmissão de dados acima de um período de 20 s, usando dois níveis discretos (0 e 1) em intervalos de 1 s, o conteúdo das informações fica:

$$H = \frac{20}{1} \log_2 2 = 20 \; bits$$

A capacidade de um determinado sistema é definida como a quantidade máxima de informação por segundo que um sistema pode transmitir e pode ser expressa em bits por segundo (bps). Assim, a capacidade do exemplo anterior torna-se:

$$C = \frac{H}{T} = \frac{1}{\lambda} \log_2 n = \frac{1}{1} \log_2 2 = 1 \; bps$$

A freqüência relativa da ocorrência de qualquer combinação ou evento é definida como a probabilidade de ocorrência, indicando simbolicamente por P, onde, números de vezes que um evento ocorre

$$P = \frac{número\ de\ tempo\ e\ eventos\ ocorridos}{número\ total\ de\ possibilidades}$$

Se n possíveis eventos são especificados para serem n níveis de sinais possíveis, então $P = 1/n$ para eventos que podem igualmente ocorrer. A informação contida pelo aparecimento de qualquer evento em um intervalo de tempo (H1) torna-se:

$$H_1 = \log_2 n = -\log_2 P \text{ bits/intervalo}$$

Onde P representa $1/n$. Durante períodos t de tempo, compostos de períodos λ s de comprimento, devemos ter t vezes informações possíveis, ou

$$H = t\, H_1 = - t \log_2 P \text{ bits em períodos } t.$$

Uma vez que o número de períodos, t, se iguala ao tempo total, T, dividido pelo número de intervalos, λ, as informações disponíveis em T s tornam-se:

$$H = -\frac{T}{\lambda} \log_2 P = \frac{T}{\lambda} -\log_2 n \text{ bits em } T\ s.$$

Com a equação acima servindo de base, podemos considerar o caso onde diferentes eventos ou níveis de sinais não têm probabilidades iguais de ocorrência. Vamos supor que apenas dois níveis devam ser transmitidos, 0 ou 1, o primeiro com probabilidade P e o segundo com probabilidade Q, onde $P + Q = 1$. Assim:

$$P = \frac{\text{número de vezes que 0 ocore}}{\text{número total de possibilidades}}$$

$$Q = \frac{\text{número de vezes que 1 ocore}}{\text{número total de possibilidades}}$$

O conteúdo das informações de uma mensagem longa composta de 0s e 1s é, desta forma, dependente de $P*\log_2 P + Q*\log_2 Q$, que é a informação em bits por ocorrência de um 0 ou 1 vezes a freqüência relativa da ocorrência do valor de bit. Podemos deixar a freqüência de ocorrência de cada nível de sinal possível ser simbolizada por P_i, onde $P_1 + P_2 + ... + P_n = 1$. Assim, cada intervalo carrega $-\log_2 Pi$ bits de informação. Em t períodos de tempo, i aparecerá de $t*P_i$ vezes em média. Somando as informações dos bits, fornecidos na média por cada símbolo, aparecendo $t*P_i$ vezes durante t intervalos, obtemos:

$$H = -t * \sum_{i=1}^{n} P_i \log_2 P_i \quad bits\ em\ t\ períodos$$

Para o intervalo T, obtemos então:

$$H = -\frac{T}{\lambda} * \sum_{i=1}^{n} P \log_2 P_i \quad bits\ em\ T\ s$$

Para uma mensagem com n símbolos ou níveis possíveis com probabilidade de ocorrência P_i a P_n, a informação média por intervalo de símbolo λ é:

$$H_{avg} = -\sum_{i=1}^{n} P_i \log_2 P_i \quad bits/intervalo\ de\ símbolo$$

A equação acima ,representa a definição matemática de entropia, termo usado na teoria da informação para indicar o número médio de bits exigidos, para representar cada símbolo de um alfabeto fonte.

Com base no que foi visto anteriormente, fica possível computar a redundância contida na informação. Já que a unidade de informação é $\log_2 n$ para um sistema capaz de transmitir n níveis discretos, sua redundância se torna

$$R = \log_2 n - H_{avg}$$

Assim, quando há redundância zero:

$$H_{avg} = \log_2 n$$

Exemplos de entropia

Uma vez que a entropia representa o número médio de bits exigidos para representar cada símbolo de um alfabeto fonte, vamos examiná-la profundamente para verificar sua aplicabilidade na compressão de dados. Vamos, em primeiro lugar, começar com a experiência de jogar cara e coroa com uma moeda na qual a probabilidade de cara (H) é igual a de coroa (T). Assim, $PH = 0.5 = PT$. Quantos bits são necessários para codificar o resultado deste cara ou coroa?

A partir da definição anterior de entropia, $H_{avg} = -\sum_{i=1}^{n} P_i \log_2 P_i$. Aqui $\log_2 X$, para aqueles que já esqueceram, significa $2^? = x$ ou 2 elevado potência tal que gere *x*. Ao substituir a probabilidade encontrada de cara e a de coroa na equação para computar a entropia, obtemos:

$$H_{avg} = -\left[\frac{1}{2}\log_2\frac{1}{2} + \frac{1}{2}\log_2\frac{1}{2}\right]$$

Uma das propriedades dos logaritmos é que log (1/X) = - logX. Assim, $\log_2 \frac{1}{2}$ é equivalente a - $\log_2 2$. Uma vez que $\log_2 2$ é igual a -1, podemos voltar ao nosso cálculo de entropia para o jogo de cara e coroa com uma moeda, onde obtemos:

$$H_{avg} = -\left[\frac{1}{2} * (-1) + \frac{1}{2} * (-1)\right] = -[-1] = 1$$

A partir do que foi dito anteriormente, a computação da entropia nos informa que um bit seria necessário para codificar os resultados de um jogo de cara e coroa com uma moeda. Agora, vamos estender a experiência acima para duas moedas.

Os dois lados de uma moeda, cara (H) e coroa (T), correspondem aos membros X_1 e X_2 de um alfabeto X, contendo dois símbolos. Se nós jogarmos cara e coroa com duas moedas e codificarmos os resultados, de maneira que T = 0 e H = 1, as probabilidades do resultado da cara e coroa correspondem a um alfabeto de quatro símbolos como tabulado na Tabela 4.1. A entropia ou número médio de bits exigidos para representar cada resultado ou símbolo possível de nosso alfabeto de quatro símbolos torna-se:

$$H_{avg} = -\sum_{i=1}^{4} P_i \log_2 P_1 = -4 * 0.25 \log_2 0.25$$

Tabela 4.1 - Representação do alfabeto de quatro símbolos para o jogo de cara e coroa

Resultado	Símbolo do alfabeto	Probabilidade de resultado	Código representativo
TT	X_1	0.25	00
TH	X_2	0.25	01
HT	X_3	0.25	10
HH	X_4	0.25	11

Uma vez que $\log_2 0.25$ é -2, podemos reescrever a equação anterior, como a seguir:

$$H_{avg} = -4\,[\,0.25 * (-2)\,] = 2$$

Para os resultados da experiência com o jogo de cara e coroa listados na Tabela 4.1, dois símbolos binários foram necessários para codificar cada símbolo alfabético. Se, por alguma razão, no jogo de cara e coroa foi fixado de maneira que somente coroa (T) ocorra, o único símbolo necessário em nosso alfabeto será X_1. Sob esta condição, nunca teríamos de fazer qualquer jogada de cara e coroa para determinar o resultado, uma vez que este já é conhecido antecipadamente. A entropia deste alfabeto de um símbolo pode ser computado como se segue:

$$H_{avg} = -\sum_{i=1}^{4} P_i \log_2 P_i = -\sum_{i=1}^{1} \log_2 1 = 0$$

Neste caso, uma vez que o resultado já é conhecido antecipadamente, o símbolo não fornece informação; conseqüentemente sua entropia é zero.

Podemos, novamente, considerar a experiência do jogo de cara e coroa, mas desta vez faremos com que a probabilidade da ocorrência de coroas (T) seja elevada para 0.75, deixando a probabilidade de 0.25 para a ocorrência de caras. Sob estas circunstâncias, os resultados tabulados do jogo de cara e coroa, representando um alfabeto de quatro símbolos seria como listado na Tabela 4.2. Mesmo que o código representativo, número de resultados de cara e coroa e os símbolos do alfabeto tenham permanecido os mesmos, as probabilidades dos resultados se alteraram. Assim, a probabilidade de duas coroas é agora 0.75 vezes 0.75, ou 0.5625, e assim por diante. A entropia deste alfabeto de quatro símbolos é agora:

$$H_{avg} = -\sum_{i=1}^{4} P_i \log_2 P_i = 0.5625 \log_2 0.5625 + 0.1875 \log_2 0.1875$$

$$+ 0.1875 \log_2 0.1875 + 0.0625 \log_2 0.0625 = 1.62 \text{ bits por símbolo}$$

Tabela 4.2 - Representação fixa do alfabeto de quatro símbolos para o jogo de cara e coroa. A probabilidade de cara = 0.25 e de coroa = 0.75

Resultado cara e coroa	Símbolo do alfabeto	Probabilidade de resultado	Código representativo
TT	X_1	0.5625	00
TH	X_2	0.1875	01
HT	X_3	0.1875	10
HH	X_4	0.0625	11

Com base no que foi visto anteriormente, vamos computar a redundância na experiência fixada de cara e coroa. Uma vez que um evento de dois símbolos resulta em quatro níveis discretos.

$$R = \log_2 n - H_{avg} = \log_2 4 - 1.68 = 2 - 1.68 = 0.38$$

Em comparação com a primeira experiência de jogo de cara e coroa, o número médio de bits necessários para representar um símbolo do alfabeto de quatro símbolos foi reduzido por 0.38. Isto indica que usando outro tipo de esquema de codificação, para representar o alfabeto de quatro símbolos, poderia resultar em uma redução aproximada de 20 por cento, em relação aos dois bits por símbolo usado anteriormente, para representar o alfabeto de quatro símbolos. Para obter esta redução, devemos atribuir códigos pequenos para os símbolos do alfabeto de maior ocorrência e códigos maiores para os símbolos de menor ocorrência. Este método resultará em uma longa cadeia de símbolos de dados tendo, em média, menos bits por símbolo e, assim, constitui a base para a codificação Huffman (Dishon, 1977; Moilanen, 1978).

A codificação Huffman é uma das diversas técnicas onde os dados são codificados com base nas probabilidades de ocorrência. Embora o método original da codificação Huffman seja aplicado a caracteres, pode, também, ser aplicado a cadeias de caracteres, além de ser aplicado em dados previamente comprimidos, tais como pares de caracteres codificados diatomicamente. Devido ao fato de a codificação Huffman formar a base para um grande número de técnicas de compressão de dados, examinaremos este método como uma entidade separada na próxima seção deste capítulo.

4.2 CODIFICAÇÃO HUFFMAN

A codificação Huffman é uma técnica de compressão de dados estatística, cujo emprego reduzirá o tamanho médio de código usado para representar os símbolos de um alfabeto. O alfabeto pode ser o da língua inglesa, ou um tipo de alfabeto de dados codificados, como, por exemplo, conjuntos de caracteres ASCII ou EBCDIC.

Propriedade do prefixo de código

O código Huffman é ótimo, uma vez que este resulta no menor tamanho médio de código entre todas as técnicas de codificação estatística. Além disso, os códigos Huffman possuem a propriedade de prefixo, que estabelece que nenhum grupo pequeno de códigos seja duplicado como o início de um grupo maior. Isto significa que se um caractere for representado pela combinação de bits 100, então 10001 não pode ser o código para outra letra, já que ao examinar o fluxo de bits da esquerda para direita, o algoritmo de decodificação interpretaria os 5 bits, como o caractere de configuração de bit 100, seguido de um caractere de configuração de bit 01.

A propriedade de prefixo do código Huffman assegura que o código seja decifrável de forma única. Para entender a importância desta propriedade e como ela se relaciona com a construção de um código Huffman, considere o alfabeto de quatro símbolos X_1, X_2, X_3, X_4, codificado como a seguir:

$$X_1 = 0, \qquad X_2 = 01, \qquad X_3 = 11, \qquad X_4 = 00.$$

Se a mensagem recebida for 0001, esta seqüência de bits poderia representar $X_1X_1X_2$ ou X_4X_2. Sendo assim, o código não é de decodificação única.

Usando o diagrama de árvore de decisão, podemos determinar visualmente porque o alfabeto de quatro símbolos codificado anteriormente, não é de decodificação instantânea. Para construir uma árvore de decisão, vamos começar em um estado inicial e desenhar uma ramificação para um n.representado pelo símbolo X_1 e rotular o valor binário atribuído ao n da ramificação. A seguir, uma vez que se atribuiu o valor 01 para X_2, vamos desenhar uma ramificação a partir do *n* X_1 para um *n* rotulado de X_2 e atribuir o valor 1 ramificação entre os ns X_1 e X_2. De forma semelhante, podemos determinar a rota de uma ramificação a partir do n X1 para um *n* representando X_4 e atribuir o valor binário 0 para aquela ramificação.

Já que se atribuiu o valor 11 para X_3, podemos desenhar uma ramificação a partir do estado inicial para um n intermediário e outra ramificação a partir do n intermediário para um n rotulado X_3, atribuindo o valor binário 1 para cada ramificação. A Figura 4.1 ilustra a árvore de decisão que corresponde à atribuição de valores ao alfabeto de quatro símbolos.

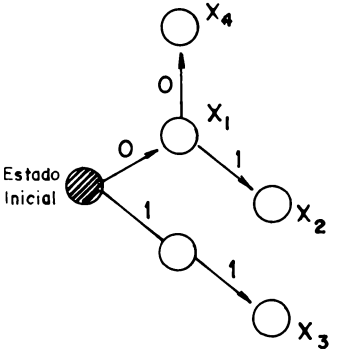


Figura 4.1 - A árvore de decisão para formar o alfabeto de quatro símbolos X_1, X_2, X_3, X_4; onde $X_1 = 0$, $X_2 = 01$, $X_3 = 11$ e $X_4 = 00$.

Ao examinar a árvore de decisão ilustrada na Figura 4.1, observe que a rota ao n X_3 é feita através de um n intermediário que não é necessário, uma vez que não há nenhuma rota do n intermediário para outro n, que não seja o n X_3. Observe, também, que o n X_1 é realmente um n intermediário e o caminho para X_1 não representa uma combinação única de bit. Agora, supondo que ao formar uma árvore de decisão, acrescentemos regras para que cada ramificação termine com um n como um estado terminal, ou funções como ponto de decisão que permita uma rota para um estado terminal. Seguindo estas regras, vamos redesenhar a árvore de decisão e atribuir valores binários para cada ramificação, de maneira a desenvolver códigos para um alfabeto de quatro símbolos. A Figura 4.2 ilustra a árvore de decisão revisada.

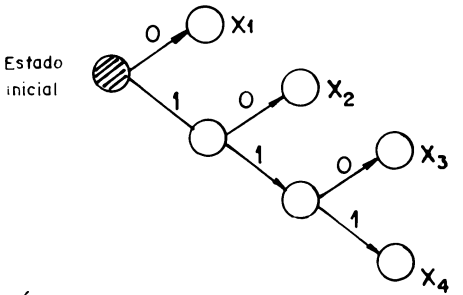


Figura 4.2 - Árvore de decisão revisada. Ao construir esta árvore, cada dígito binário produz uma ramificação, ou para um estado terminal ou para um ponto de decisão.

Observe que X_1 ainda tem o valor binário 0, mas foram atribuídos os valores 10, 110 e 111 para X_2, X_3 e X_4, respectivamente. Observe, também, que cada bit é somente examinado uma vez e cada caminho representa uma combinação de bits única.

As regras, que acabamos de desenvolver para a construção de uma árvore de decisão que resulte em um código que seja decodificado instantaneamente, formam as bases nas quais o código Huffman foi construído. O código Huffman pode ser desenvolvido através da utilização de uma estrutura de árvore, como ilustrado na Figura 4.3. Aqui, os símbolos são primeiramente listados em ordem decrescente de acordo com a freqüência de ocorrência. Os grupos com as freqüências menores (X_3 e X_4) são combinados em um *n* com a probabilidade conjunta de ocorrência de 0.25. A seguir, esse *n* é fundido com a menor probabilidade de ocorrência de símbolo ou par de símbolos mais próxima. Nesta ilustração, o par X_3 X_4 é fundido com X_2 para gerar um n cuja probabilidade desta junção seja 0.4375. Finalmente, o n representando as probabilidades de ocorrência de X_2,X_3 e X_4 é fundido com X_1, resultando em um *n* cuja probabilidade de ocorrência é unitária. Este n mestre representa a probabilidade de ocorrência de todos os quatro caracteres no conjunto de caracteres.

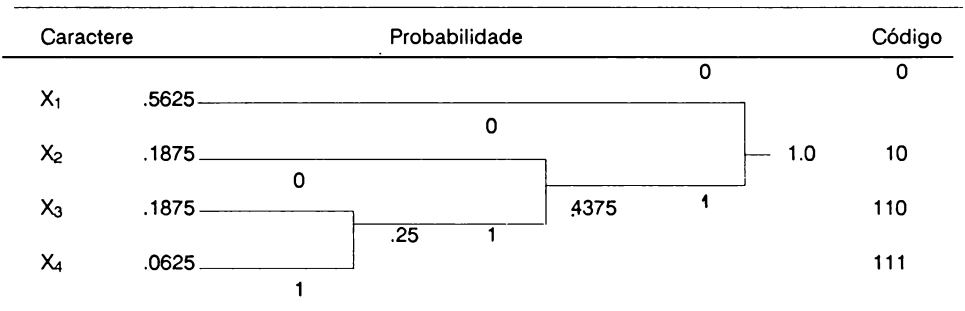


Figura 4.3 - O desenvolvimento do código Huffman, empregando uma estrutura de árvore. Os códigos Huffman podem ser desenvolvidos, empregando-se uma estrutura de árvore. O código Huffman, resultante deste método de construção, é gerado ao se traçar o n de probabilidade 1.0 para cada caractere fonte (símbolo), observando 1s e 0s encontrados.

Ao atribuir 0s e 1s binários para cada segmento originado de cada n, você pode gerar o código Huffman para cada caractere. O código é obtido ao rastrear o n de probabilidade 1.0 até cada símbolo de caractere, observando os 1s e 0s encontrados. Se você construir a árvore de decisão que representa os códigos atribuídos aos carac-

teres de X_1 a X_4, a árvore será exatamente a mesma, como a ilustrada na Figura 4.2. Já que cada dígito binário produz uma ramificação para um estado terminal ou para um ponto de decisão, este representa um código de decodificação única.

O número médio de bits por símbolo pode ser calculado, multiplicando os comprimentos do código Huffman por sua probabilidade de ocorrência. Sendo assim, o código usa:

$$1*0.5625 + 2*0.1875 + 3*0.1875 + 3*0.0625$$

ou 1.63 bits por símbolo. Observe que o resultado do código Huffman de 1.63 bits por símbolo se aproxima da entropia de 1.62 bits por símbolo (Dishon, 1977; Moilanen, 1978).

Como explicado anteriormente, a propriedade chave do código Huffman é o fato deste poder ser decodificado instantaneamente, À medida que os bits codificados no fluxo de dados comprimidos são encontrados. Um exemplo da propriedade de decodificação instanâtnea encontra-se ilustrado na Figura 4.4. Aqui, o fluxo de dados comprimidos pode ser decodificado imediatamente ao ser lido da esquerda para direita, sem esperar que ocorra o fim do bloco de dados.

A substituição de um número de bits o qual representa um caractere de dados ou um grupo de caracteres particular é um processo razoavelmente simples quando o número de substituições for limitado. À medida que o número de substituições aumenta, a complexidade do processo de substituição também aumenta.

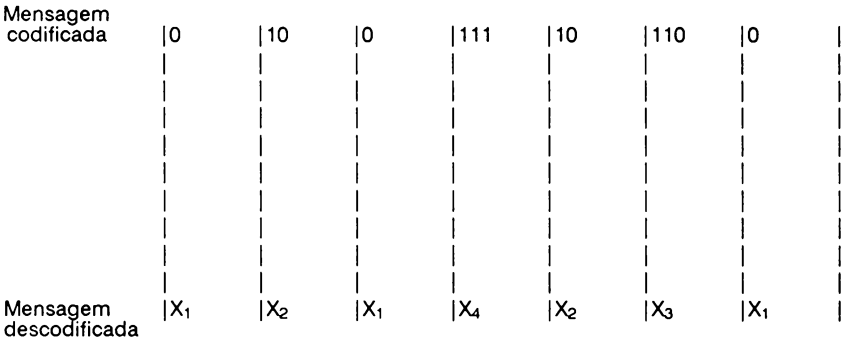


Figura 4.4 - Propriedade de decodificação instantânea. Uma das propriedades chaves da técnica Huffman é o fato de que os dados codificados podem ser decodificados instantaneamente.

O desenvolvimento de um código Huffman para o alfabeto Inglês está ilustrado nas Figuras 4.5 e 4.6. A estrutura em árvore usada para desenvolver o código, mostrado na Figura 4.5, é gerado como a seguir:

A. O conjunto de caracteres está disposto na coluna esquerda em ordem decrescente com a freqüência de ocorrência, onde a freqüência está colocada na coluna ao lado do caractere.

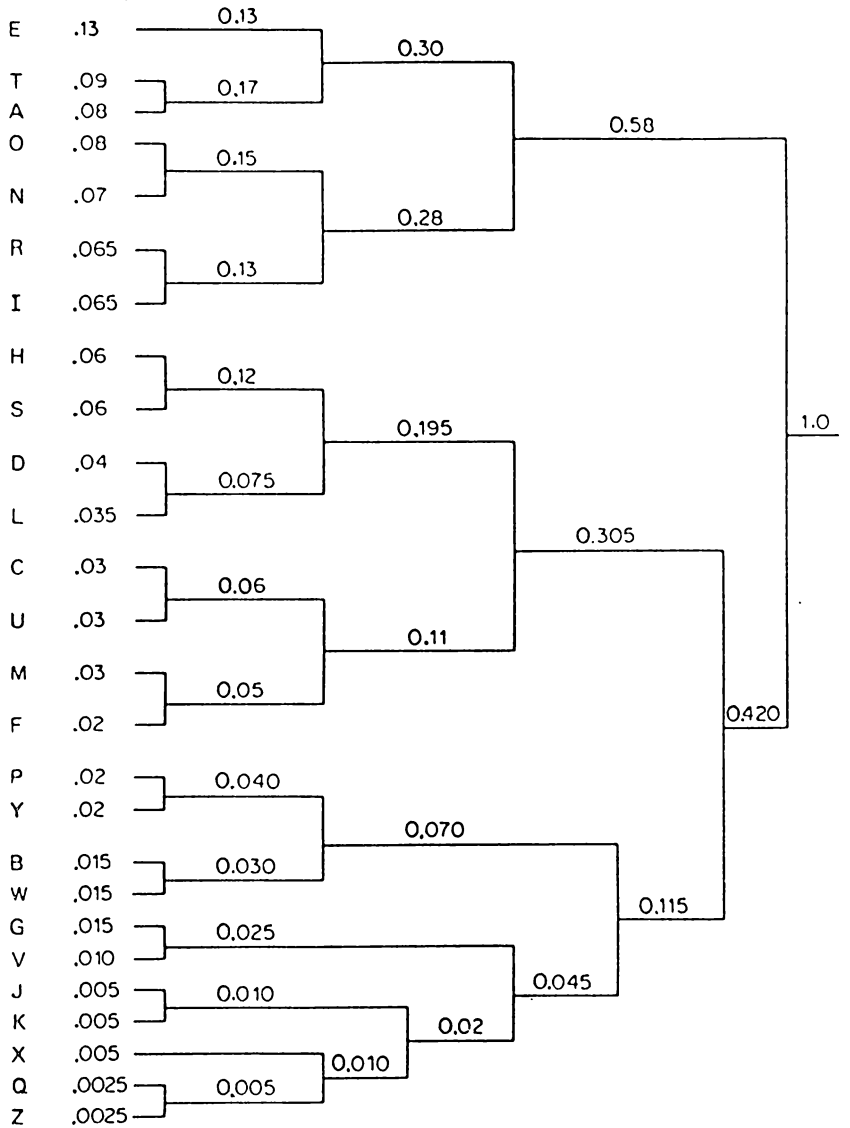


Figura 4.5 - Desenvolvendo uma estrutura em árvore para o alfabeto.

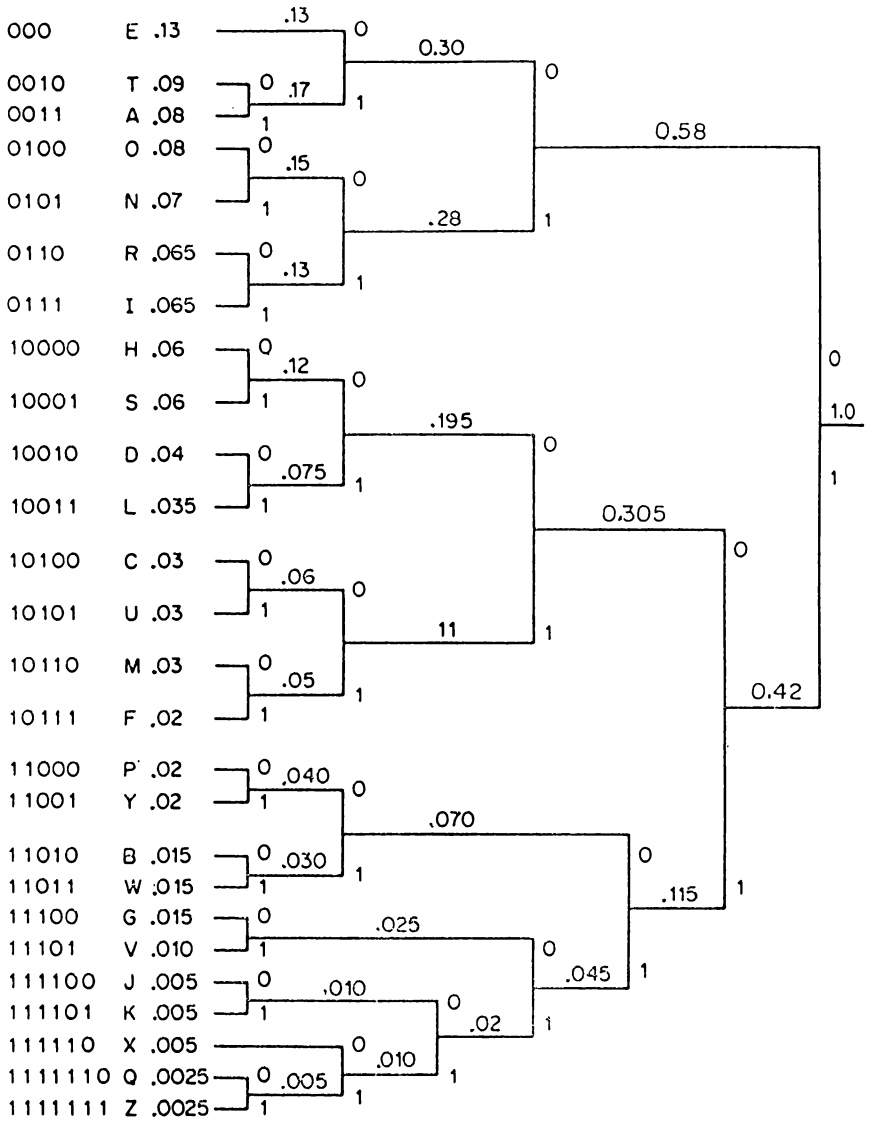


Figura 4.6 - Atribuindo o código Huffman.

B. Começando pela parte inferior da tabela, são desenhadas linhas horizontais para cada freqüência de caractere. As linhas das duas freqüências mais baixas de ocorrência são fundidas e suas respectivas freqüências são somadas para se obter a freqüência composta. Esta freqüência composta é colocada em uma linha nova que reflete a freqüência combinada dos caracteres previamente colocados em pares.

C. O processo de combinar duas linhas de freqüências mais baixas em uma única linha, contendo as freqüências combinadas, continua até que todas as linhas sejam fundidas.

Depois que a árvore foi desenvolvida, o código Huffman para cada caractere pode ser atribuído, colocando-se um bit 0 em um lado de cada ponto nodal, e um bit 1 no outro caminho derivado do ponto em direção ao símbolo esquerdo. A atribuição dos bits 0 e 1 é arbitrária. Contudo, a atribuição deve ser consistente. Assim, se o binário 0 for escolhido para representar uma rota superior, então, deverá ser usado para todas as rotas superiores. A seqüência de bit apropriada, atribuída a cada caractere de dado, é, então, determinada ao se traçar a rota do ponto nodal mestre, onde a probabilidade de todas as freqüências de ocorrência de caracteres é unitária ao n inicial para o caractere apropriado, observando os bits designados ao caminho. A atribuição de bits para os caminhos e os valores codificados de Huffman resultantes para o alfabeto Inglês estão ilustrados na Figura 4.6.

O número de bits necessários para codificar uma letra, usando a técnica Huffman pode ser determinado a partir da seguinte fórmula:

$$b = f(- \log_2 P)$$

onde: P = *probabilidade de ocorrência da letra*

$f(x)$ = *o número inteiro mais próximo que seja maior, ou igual x.*

Uma vez que a probabilidade de E é 0.13 e $- \log_2 0.13$ é 2.94, então o número inteiro maior que, ou igual a 2.94 é 3. Assim, 3 bits são necessários para codificar a letra E (Peterson, Bitner e Howard, 1978).

Considerações construção de código

Quando diversos caracteres têm a mesma freqüência de ocorrência, você pode ser capaz de desenvolver dois ou mais códigos, para representar o código Huffman para um conjunto de caracteres. Esta situação, que freqüentemente ocorre, nos fornece um problema interessante - decidir que código usar. Para ilustrar como este problema pode surgir, bem como fornecer a base para discutir sua solução, vamos considerar primeiramente um conjunto de caracteres contendo cinco caracteres, cujas freqüências de ocorrência estão indicadas na Tabela 4.3.

Tabela 4.3 - Amostra de um conjunto de caracteres composto de cinco caracteres

Caractere	Freqüência de ocorrência
X_1	0.4
X_2	0.2
X_3	0.2
X_4	0.1
X_5	0.1

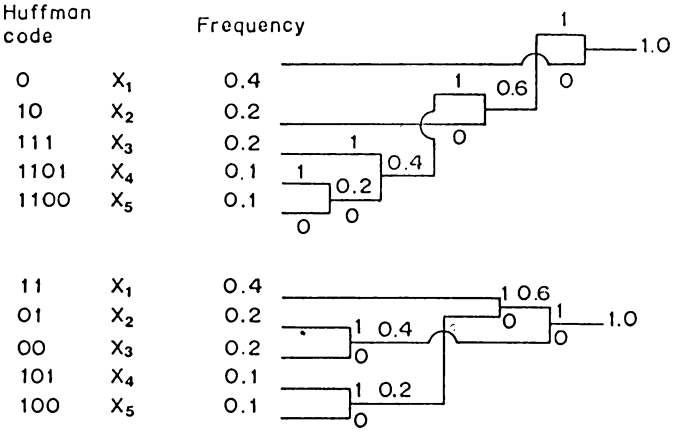


Figura 4.7 - Desenvolvendo diferentes árvores e códigos.

A Figura 4.7 ilustra o desenvolvimento de duas estruturas de árvore, cada uma resultando na construção de códigos Huffman diferentes para os caracteres listados na Tabela 4.3. Observe que na porção superior da Figura 4.7, X_4 e X_5 são combinados exatamente da mesma maneira que na segunda estrutura de árvore, ilustrada na porção inferior da Figura 4.7. Contudo, depois que X_4 e X_5 são combinados, a freqüência de ocorrência resultante dos dois caracteres é 0.2 e você tem diversas opções disponíveis ao desenvolver o código Huffman.

No porção superior da Figura 4.7, X_3 foi o próximo a ser combinado com a fusão de X_4 e X_5, já que X_3 tinha a probabilidade de ocorrência de 0.2, que igualou exatamente, a probabilidade combinada de ocorrência de X_4 e X_5. Na porção inferior da Figura 4.7, X_2 e X_3 foram fundidos, resultando na probabilidade combinada de ocorrência de 0.4. Uma vez que devemos fundir as freqüências mais

baixas de ocorrências com as mais altas para formar o código Huffman, combinamos os resultados da fusão de X_4 e X_5 com X_1 para gerar a freqüência combinada de ocorrência de 0.6. Isto foi seguido pela fusão dos resultados da combinação de X_2 e X_3 com os resultados obtidos da fusão de X_1 com a combinação de X_4 e X_5.

Ao examinar o código Huffman resultante na parte superior da Figura 4.7, como ele se compara ao código na parte inferior daquela ilustração? Um modo que podemos comparar códigos é ao computar seu tamanho médio de bit; então vamos fazê-lo. Para o código de Huffman na parte superior da Figura 4.7, seu tamanho médio em bits é:

$$L = 0.4*1 + 0.2*2 + 0.2*3 + 0.1*4 + 0.1*4 = 2.2 \text{ bits.}$$

Para o código Huffman na parte inferior da Figura 4.7, seu tamanho médio em bits é:

$$L = 0.4*2 + 0.2*2 + 0.2*2 + 0.1*3 + 0.1*3 = 2.2 \text{ bits.}$$

Observe que os dois códigos têm a mesma eficiência expressa em termos de número médio de bits necessários para codificar um caractere. Contudo, os dois códigos não têm o mesmo conjunto de tamanhos de bits para caracteres do conjunto de caracteres. Embora, durante um longo período de tempo, o uso de qualquer um dos dois códigos deve produzir o mesmo resultado em termos de eficiência expressa como a soma de bits codificados, na realidade, a escolha mais apropriada é selecionar o código cujo tamanho médio varie menos. Para ilustrar, vamos computar a variância para cada código Huffman.

A variância do código Huffman na parte superior da Figura 4.7 é:

$$\text{Var (1)} = 0.4(1-2.2)^2 + 0.2(2-2.2)_2 + 0.2(3-2.2)_2 + 0.1(4-2.2)_2 + 0.1(4-2.2)_2 = 1.36.$$

A variância do código Huffman desenvolvido na porção inferior da Figura 4.7 é:

$$\text{Var (2)} = 0.4(2-2.2)^2 + 0.2(2-2.2)^2 + 0.2(2-2.2.2)^2 + 0.1(3-2.2)^2 + 0.1(3-2.2)^2 = 1.6.$$

Como indicado, o segundo código Huffman tem menor variação e deve ser escolhido.

Voltando à Figura 4.7, se examinarmos a construção de cada árvore, observaremos que as freqüências combinadas de ocorrência de X_4 e X_5 na porção inferior da ilustração foram movidas diretamente na parte superior. Em compensação, a combinação das freqüências de ocorrência de X_4 e X_5 na porção superior da Figura 4.7, foram gradualmente fundidas com outros estados para alcançar o topo. Quando você move um estado combinado o mais alto possível, se reduz a variação do código Huffman resultante e pode ser considerado como a chave para a construção de um código Huffman de variação mínima.

Requisitos Informação

Para desenvolver um código Huffman cujo tamanho médio se aproxime de sua entropia, é necessário que a distribuição de freqüência dos caracteres ou símbolos seja codificada de maneira a ser conhecida antecipadamente. Uma vez que a distribuição de freqüência de um fluxo de dados é proporcional ao uso final do fluxo, este fator pode resultar em uma distribuição de freqüência pré-selecionada, usada para desenvolver um código Huffman,resultando em um código aquém do ideal, durante certas seqüências de transmissão de dados. Citando um exemplo, a distribuição de freqüência do texto em Inglês, como aquele resultante do arquivo de dados usado para composição tipográfica computadorizada, pode ser completamente diferente do arquivo de dados, contendo os resultados da compilação de um programa *FORTRAN*. Em primeiro lugar, a distribuição dos caracteres deve seguir a distribuição do Inglês normal, com o E sendo o caractere de ocorrência mais freqüente, e o Z sendo um dos caracteres de ocorrência menos freqüente. Para a compilação *FORTRAN*, caracteres especiais, tais como: parênteses, + para adição, - para subtração, * para multiplicação e / para divisão, têm um alto grau de ocorrência não encontrado normalmente no texto em Inglês.

Para compensar as diferenças nas distribuições de freqüência, diversos esquemas de codificação podem ser considerados. Primeiro, a análise de arquivos de dados misturados pode ser conduzida, empregando-se o programa de computador listado no Apêndice A. Isto lhe permitirá averiguar a relação apropriada entre as freqüências de ocorrência de caracteres em tipos diferentes de dados.

O segundo método para se considerar é a técnica auto-adaptável de codificação Huffman. Esta técnica pode primeiramente solicitar uma análise de freqüência de um grande bloco de dados que

seria, então, codificado com base naquela distribuição. Antes da transmissão dos dados codificados, uma tabela de símbolos e códigos Huffman, desenvolvida para cada símbolo, deve ser transmitida para possibilitar a decodificação perfeita dos dados codificados. Com um pouco de imaginação, você pode visualizar que fluxos de dados em freqüente mudança resultariam em numerosas tabelas e dados codificados sendo transmitidos. Estas tabelas podem ser consideradas como dispêndio, resultando na diminuição da freqüência de compressão medida, em que o número de distribuições de freqüência de fluxo de dados muda por unidade de tempo.

Outro problema encontrado em algumas técnicas auto-adaptáveis de codificação Huffman, é a determinação do tamanho do fluxo de dados para amostra e os intervalos entre amostras. Quanto maior for a amostra, maior se torna a exigência de processamento. Se os dados devem ser transmitidos, uma área de buffer é necessária para colocar a amostra, enquanto a análise de freqüência é conduzida. A respeito do intervalo entre amostras, se três jobs *FORTRAN* são seguidos por um serviço de texto em Inglês, todos do mesmo tamanho, T, a amostragem em T, T + 2 e T + 4 resultaria em um serviço de texto em Inglês, sendo excluído da amostra. Uma vez que um operador de terminal remoto por lote (remote batch terminal) submete jobs e extrai saídas do sistema, ele ou ela sabe antecipadamente o tipo de serviço que será transmitido para ou recebido do computador. Para este tipo de situação operacional, distribuições de freqüência predefinidas podem ser selecionadas pelo operador, e transportadas para a extremidade oposta da ligação de transmissão, através da transmissão de um código especial.

Para eliminar os problemas descritos anteriormente, que resultam da geração de tabelas de freqüência, você pode construir uma técnica de codificação Huffman verdadeiramente auto-adaptável. Esta técnica gera tabelas de freqüência nas duas extremidades da ligação de transmissão, à medida que ocorre a transmissão de dados e o ajuste adaptativo daquelas tabelas durante a transmissão. Consulte a Seção 4.5 que discute esta técnica detalhadamente.

O terceiro método de compensar as diferenças de distribuição de freqüência é através do uso de um código de texto simples, que é usado para indicar que o caractere seguinte deve ser reproduzido da mesma forma em que foi recebido. Isto permite que caracteres que ocorrem raramente nos dados fontes, sejam excluídos do processo de codificação e resulte no desenvolvimento de um tipo de código Huffman modificado. Aqui, você poderia agrupar todos os caractere de baixa freqüência de ocorrência em uma única

probabilidade de ocorrência e designar um código Huffman para representar a probabilidade somada. Isto seria o código de texto simples e indicaria que os próximos oito bits representam um caractere real de dados não codificados. Sem o uso de um código de texto simples, grandes cadeias de, por exemplo, 20 ou mais bits podem resultar na representação de baixa freqüência de ocorrência de caracteres. Se o código de texto simples tivesse quatro bits de comprimento, então um máximo de 12 bits seria solicitado para representar qualquer freqüência baixa de ocorrência de caractere. A preempção de um código de quatro bits, para significar que os próximos oito bits, formam a representação de texto simples de um caractere de oito bits significa que algumas freqüências relativamente altas de ocorrência de caractere que teria lido um código de 4 bits para sua representação Huffman, será representada por algum código maior. Assim, embora não haja códigos muito grandes, o número médio de bits por caractere aumentará quando o código de texto simples for empregado.

4.3 CóDIGOS HUFFMAN MODIFICADOS

A representação de caracteres e símbolos por um código Huffman apropriado é excelente em teoria, se você deseja ter um número médio de bits por símbolo aproximado de sua entropia. Na prática, contudo, inúmeras dificuldades podem surgir, quando a codificação Huffman é aplicada em certas aplicações, particularmente na área de transmissão por fac-símile.

Ao aplicar a codificação Huffman para transmissão por fac-símile, cada linha de fac-símile pode ser vista como a composição de uma série de fileiras em preto ou em branco, cada fileira consistindo de uma série de elementos de imagem semelhantes. Se o tipo da primeira fileira é conhecido, então o tipo de todas as sucessivas fileiras será conhecido, à medida em que fileiras em branco e preto devem se alternar. A probabilidade de ocorrência de cada fileira de um determinado comprimento de elementos de imagem pode ser calculada, e pequenas palavras de código podem ser usadas para representar fileiras que têm uma alta freqüência de ocorrência, enquanto palavras maiores de códigos podem ser usadas para representar fileiras que têm uma baixa probabilidade de ocorrência. De um modo semelhante, mudança de jobs de processamento de dados, as estatísticas para as probabilidades de comprimento de fileira associadas a varreduras de linhas mudam linha-a-linha e documento-a-documento. Assim, um código timo, ou quase timo, para uma linha ou documento específico pode ser ruim para outra

linha ou documento diferente. O segundo maior problema é o fato que a criação do código Huffman em tempo real, requer um grande poder de processamento, normalmente muito além da capacidade de máquinas fac símiles onde o custo do leitor, do transmissor/receptor, da lógica central e da fonte de alimentação resulta em um custo de máquina abaixo de mil dólares, necessário para permanecer competitivo. Para reduzir alguns requisitos de processamento de tempo real, uma abordagem de busca em tabela, pode ser empregada. Uma vez que os padrões CCITT requerem 1728 elementos de imagem por linha, o uso da técnica de busca em tabela necessitaria de armazenamento para 1728 localizações de comprimentos variáveis para máquina de fac símile, cada localização contendo uma palavra de código binário correspondente a um determinado comprimento de fileira. Os problemas de implementação associados à aplicação completa da técnica de codificação Huffman completa para aplicações em fac símile resultaram no desenvolvimento de um esquema de codificação Huffman modificado mais adequado à redução do custo do hardware para o competitivo mercado de fac símile.

No desenvolvimento de um esquema de codificação Huffman modificado, para aplicações em fac símile, uma alteração foi feita de maneira que, raramente o comprimento médio de símbolo se aproxima da entropia, permitindo, realmente, uma significante compressão, enquanto minimiza as exigências de hardware e de processamento. Aqui, a probabilidade de ocorrências de comprimentos diferentes de fileira de elementos de imagens (pels), foi calculada para todos os tamanhos de fileiras de branco e de preto, baseados nas estatísticas obtidas da análise de um grupo de 11 documentos recomendados pelo CCITT como sendo típicos. Para reduzir os requisitos de armazenamento pela busca em tabela, o conjunto de código Huffman foi truncado pela criação da representação na base 64, de cada comprimento de fileira e pela utilização de duas tabelas de códigos, para reduzir o tamanho global da tabela em comparação ao tamanho da tabela que seria necessária, se somente uma tabela fosse usada (McCullough, 1977).

Com base nas probabilidades de comprimento de fileira de 11 documentos típicos, tabelas de códigos foram desenvolvidas para comprimentos de fileira que variam de 1 a 63 elementos de imagem. Uma vez que a probabilidade de ocorrência de fileiras de branco difere da freqüência de ocorrência de fileiras de preto, uma tabela deve ser desenvolvida para ambas as fileiras. Esta tabela dual encontra-se listada na Tabela 4.4, para comprimentos de fileira que variam de 0 a 63 elementos de imagem. Os códigos nesta tabela, representam o dígito menos significativo (LSD) da palavra de código

e são freqüentemente referidos como código de terminação. Para permitir que a codificação de fileiras, com mais de 63 elementos de imagem, um segundo conjunto de tabelas de código deve ser empregado para manipular fileiras variando em tamanho de 64 elementos de imagem (pels), até o comprimento máximo de varredura de linha de 1728 elementos de imagem (pels). Estes códigos estão listados na Tabela 4.5. Eles representam o dígito mais significativo da palavra de código e são conhecidos como código mestre.

Tabela 4.4 - Códigos do dígito menos significativo para o processo Huffman modificado

Run Length brancos	Código da Palavra	Representação na base 64	Run Length Preto	Código da Palavra
0	00110101	0	0	0000110111
1	000111	1	1	010
2	0111	2	2	11
3	1000	3	3	10
4	1011	4	4	011
5	1100	5	5	0011
6	1110	6	6	0010
7	1111	7	7	00011
8	10011	8	8	000101
9	10100	9	9	000100
10	00111	a	10	0000100
11	01000	b	11	0000101
12	001000	c	12	0000111
13	000011	d	13	00000100
14	10100	e	14	00000111
15	110101	f	15	000011000
16	101010	g	16	0000010111
17	101011	h	17	0000011000
18	0100111	i	18	0000001000
19	0001100	j	19	00001100111
20	0001000	k	20	00001101000
21	0010111	l	21	00001101100
22	0000011	m	22	00000110111
23	0000100	n	23	00000101000
24	0101000	o	24	00000010111
25	0101011	p	25	00000011000
26	0010011	q	26	000011001010

Tabela 4.4 - Continuação

Run Length brancos	Código da Palavra	Representação na base 64	Run Length Preto	Código da Palavra
27	0100100	r	27	000011001011
28	0011000	s	28	000011001100
29	00000010	t	29	000011001101
30	00000011	u	30	000001101000
31	00011010	v	31	000001101001
32	00011011	w	32	000001101010
33	00010010	x	33	000001101011
34	00010011	y	34	000011010010
35	00010100	z	35	000011010011
36	00010101	A	36	000011010100
37	00010110	B	37	000011010101
38	00010111	C	38	000011010110
39	00101000	D	39	000011010111
40	00101001	E	40	000001101100
41	00101010	F	41	000001101101
42	00101011	G	42	000011011010
43	00101100	H	43	000011011011
44	00101101	I	44	000001010100
45	00000100	J	45	000001010101
46	00000101	K	46	000001010110
47	00001010	L	47	000001010111
48	00001011	M	48	000001100100
49	01010010	N	49	000001100101
50	01010011	O	50	000001010010
51	01010100	P	51	000001010011
52	01010101	Q	52	000000100100
53	00100100	R	53	000000110111
54	00100101	S	54	000000111000
55	01011000	T	55	000000100111
56	01011001	U	56	000000101000
57	01011010	V	57	000000101100
58	01011011	W	58	000001011001
59	01001010	X	59	000000101011
60	01001011	Y	60	000000101100
61	00110010	Z	61	000001011010
62	00110011	*	62	000001100110
63	00110100	#	63	000001100111

Tabela 4.5 - Códigos dos dígitos mais significativos para o processo Huffman modificado

Run Length brancos	Código da Palavra	Representação na base 64	Run Length Preto	Código da Palavra
64	11011	1	64	0000001111
128	10010	2	128	000011001000
192	010111	3	192	000011001001
256	0110111	4	256	000001011011
320	00110110	5	320	000000110011
384	00110111	6	384	000000110011
448	01100100	7	448	000000110101
512	01100101	8	512	0000001101100
576	01101000	9	576	0000001101101
640	01100111	a	640	0000001001010
704	011001100	b	704	0000001001011
768	011001101	c	768	0000001001100
832	011010010	d	832	0000001001101
836	011010011	e	836	0000001110010
960	011010100	f	960	0000001110011
1024	011010101	g	1024	0000001110100
1088	011010110	h	1088	0000001110101
1152	011010111	i	1152	0000001110110
1216	011011000	j	1216	0000001110111
1280	011011001	k	1280	0000001010010
1344	011011010	l	1344	0000001010011
1408	011011011	m	1408	0000001010100
1472	010011000	n	1472	0000001010101
1536	010011001	o	1536	0000001011010
1600	010011010	p	1600	0000001011011
1664	011000	q	1664	0000001100100
1728	010011011	r	1728	0000001100101
EOL	0000000000		EOL	00000000001

Quando uma fileira de 63 pels ou menos é encontrada, o tipo apropriado do conjunto de código LSD é acessado, para obter uma simples palavra de código na base 64. Para codificar uma fileira de 64 pels ou mais, duas palavras de código na base 64 devem ser usadas. Primeiro, a palavra de código de dígito mais significativo é obtida da tabela de códigos MSD (dígito mais significativo), de maneira que N*64, $1 \leq N \leq 27$, não ultrapasse o comprimento de fileira. Aqui, N não pode ultrapassar 27, visto que quando $N = 27$ o máximo de 1728 pels por linha. A seguir, a diferença entre o comprimento da fileira e *N*64* é obtida e o dígito menos significativo é acessado da tabela de código LSD apropriada. A Figura 4.8 mostra um exemplo das operações de busca em tabela, para uma amostra de seqüência de fileiras de branco e de preto de vários tamanhos de pels. Na porção superior desta ilustração, está tabulada a relação entre uma série de dados originais de vídeo e sua representação no código Huffman modificado. Com base no que foi visto, você pode considerar a aplicação da técnica de codificação Huffman modificada como um esquema de codificação unidimensional. Este esquema de codificação é aplicado na base de linha por linha para cada linha varrida e lida, e representa a correlação horizontal comprimida de elementos de imagem (pels) dentro da mesma linha.

Para empregar, com sucesso, o esquema de codificação Huffman modificado deve-se desenvolver e seguir algumas regras, para aliviar as diversas deficiências inerentes ao emprego de uma técnica de codificação estatística. Nestas técnicas, as palavras de código não contêm qualquer informação posicional inerente, à qual é necessária sincronização. Isto pode ser compensado fazendo disto uma regra onde a primeira fileira de cada linha deva ser uma fileira de brancos, mesmo que resulte em um comprimento de fileira zero. Depois disso, as fileiras devem alternar entre fileiras de branco e de preto. Para denotar o início e o fim de cada varredura de linha, um código único de delineação de linha, às vezes chamado de código de fim de linha (EOL - end-of-line), pode ser empregado. Quando todas as linhas estiverem codificadas, bits 0s de preenchimento podem ser empregados como bits de atenuação antes da transmissão do EOL para fins de sincronização. O resultado final da incorporação destas regras permite que o formato de linha seja definido, como mostrado na Figura 4.9. Através da incorporação da técnica de codificação Huffman modificada, o tempo de transmissão de um documento comercial típico foi reduzido para menos de 20s em uma taxa de transmissão de 9600 bps.

Dados de vídeo originais		Código Huffman modificado representação na base 64		Código Huffman modificado representação na base 2 MSD	LSD
5	pels preto	5	(preto)	NA	0011
17	pels branco	h	(branco)	NA	101011
32	pels preto	w	(preto)	NA	000001101010
32	pels branco	w	(branco)	NA	00011011
728	pels preto	b ϕ	(preto)	0000001001011	00000010111
1728	pels branco	r ϕ	(branco)	010011011	00110101
64	pels preto	1 ϕ	(preto)	0000001111	0000110111
55	pels branco	T	(branco)	NA	01011000 1
028	pels branco	g2	(branco)	011010101	0111

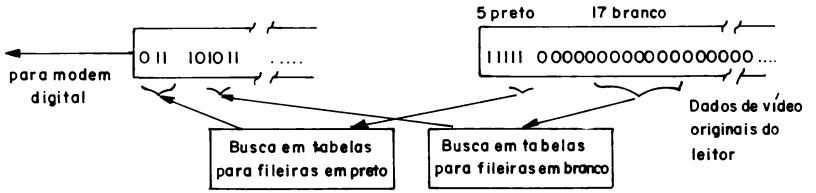
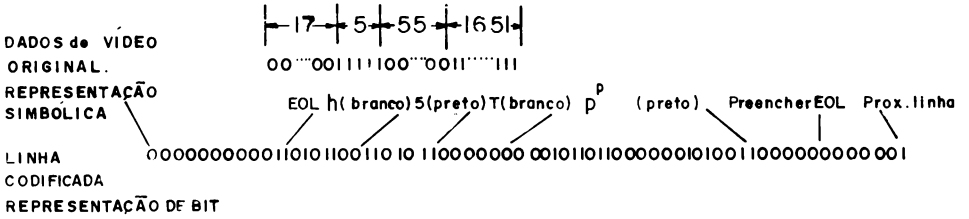


Figura 4.8 - A codificação usando o código Huffman modificado. O código Huffman modificado é construído por uma seqüência de referências tabulares para fileiras de branco e de preto.

O significado da redução se torna aparente, quando você considerar que a resolução de 1780 pels por linha e 96 linhas horizontais por polegada, resulta em um total de 1.410.048 pels por documento de 8.5 x 11 polegadas. Sem compressão, um tempo de transmissão de aproximadamente 2.5 minutos seria necessário para os dados, sem considerar a transmissão dos códigos de fim de linha.



Bits totais antes da compressão	1728
Bits totais depois da compressão	52 (não incluindo 'dados completos')
Razão de compressão para esta linha	33:1

Figura 4.9 - Regras definem o formato de linha. Para denotar o início e o fim de cada linha varrida, um código de fim de linha (EOL) é empregado.

4.4 CODIFICAÇÃO SHANNON-FANO

A codificação Shannon-Fano, semelhante à codificação Huffman, resulta em um código de comprimento variável que é codificável instantaneamente. Igual ao desenvolvimento de um código Huffman, antes de desenvolver o código Shannon-Fano para cada caractere em seu conjunto de caracteres, você deve determinar a probabilidade de ocorrência de cada caractere. Então, você arranjará seu conjunto de caracteres em ordem decrescente com base na probabilidade de ocorrência de cada caractere.

Uma vez que seu conjunto de caracteres esteja disposto em ordem decrescente pela probabilidade de ocorrência, o conjunto deve ser dividido em dois subconjuntos iguais ou quase iguais, com base na probabilidade de ocorrência dos caracteres em cada subconjunto. O primeiro dígito em um subconjunto que representa um grupo de freqüências de ocorrência é atribuído com o valor binário 0, enquanto um binário 1 é atribuído ao primeiro dígito no segundo subconjunto. Aqui, o segundo subconjunto representa todas as freqüências restantes de ocorrência. Este processo de formar subconjuntos é repetido até que o conjunto de caracteres esteja completamente subdividido. Então, um bit de sufixo é adicionado em todos os caracteres de um subconjunto de dois caracteres, como requisito para distinguir uma composição binária de caractere de um outro caractere no subconjunto.

Para obter um entendimento do procedimento da codificação Shannon-Fano, vamos supor que nosso conjunto de caracteres contenha sete caracteres cujas probabilidades de ocorrência sejam as indicadas na Tabela 4.6.

Tabela 4.6 - Probabilidade de ocorrência do conjunto de caracteres

Caractere	Freqüência de ocorrência
X_1	0.10
X_2	0.05
X_3	0.20
X_4	0.15
X_5	0.15
X_6	0.25
X_7	0.10

Ao arranjar os caracteres no conjunto de caracteres em ordem decrescente com base em suas probabilidades de ocorrência, podemos começar a formar os subconjuntos. No processo de construção do subconjunto, agruparemos os caracteres dentro de cada subconjunto, de maneira que a probabilidade de ocorrência dos caracteres em cada subconjunto seja igual ou quase igual. Então atribuiremos os binários 1 a um subconjunto e binários 0s ao outro subconjunto e continuaremos a repetir o processo, até que todos os subconjuntos possíveis estejam construídos. A Figura 4.10 ilustra este processo.

Caractere	Probabilidade	Código
X_6	0.25	1
X_3	0.20	1
X_4	0.15	0 1
X_5	0.15	0 1
X_1	0.10	0 0 1
X_7	0.10	0 0 0
X_2	0.05	0 0 0

Figura 4.10 - Processo inicial de codificação Shannon-Fano.

Observe que depois do processo inicial de codificação estar completo, os subconjuntos representados pelos pares de caracteres X_6,X_3 e X_4,X_5 não são únicos. Assim, um binário 1 e 0 deve ser acrescentado aos pares em cada subconjunto. Isto resulta no término do processo de codificação de comprimento variável, no qual cada caractere é representado por uma única combinação de bits instantaneamente decodificável. O código completo para cada caractere do nosso conjunto de caracteres encontra-se ilustrado na Figura 4.11.

Caractere	Probabilidade	Código
X_6	0.25	1 1
X_3	0.20	1 1
X_4	0.15	0 1 1
X_5	0.15	0 1 0
X_1	0.10	0 0 1
X_7	0.10	0 0 0 1
X_2	0.05	0 0 0 0

Figura 4.11 - Processo de codificação Shannon-Fano Completo.

Ambigüidade de código

Um dos problemas associados ao desenvolvimento de um código Shannon-Fano é o fato de que o procedimento para desenvolvê-lo pode ser ambíguo. Para ilustrar este problema, considere o processo de codificação Shannon-Fano ilustrado na Figura 4.12. Observe que a probabilidade de ocorrência dos caracteres no conjunto de 8 caracteres é tal, que não está claro se a primeira divisão para criar subconjuntos deve ocorrer entre X_1 e X_2 ou entre X_2 e X_3.

Caractere	Probabilidade	Desenvolvimento de Código	Código
X_1	0.35	1 1	11
X_2	0.30	1 0	10
X_3	0.15	0 1	01
X_4	0.08	0 0 1	001
X_5	0.05	0 0 0 1	0001
X_6	0.03	0 0 0 0 1	00001
X_7	0.02	0 0 0 0 0 1	000001
X_8	0.02	0 0 0 0 0 0	000000

Figura 4.12 Desenvolvimento de código ambíguo.

Ao fazer a primeira subdivisão entre X_2 e X_3, o código Shannon -Fano resultante, tem um comprimento médio de bits de 2.43, computado como a seguir:

$$L = 0.35*2 + 0.3*2 + 0.08*3 + 0.05*4 + 0.03*5 + 0.02*6 + 0.02*6 = 2.43.$$

Neste exemplo particular, fazendo a primeira subdivisão entre X_2 e X_3 resulta em um timo código, cujo tamanho médio de bit é o mesmo que aquele obtido pelo processo de desenvolvimento do código Huffman. Se uma subdivisão diferente ocorre, o comprimento médio de bits do código Shannon-Fano excederá aquele obtido pelo processo de desenvolvimento do código Huffman.

Comparação de eficiência

Para comparar a eficiência do processo de codificação Shannon-Fano com a técnica de codificação Huffman abordada anteriormente, vamos desenvolver o código Huffman para um conjunto de caracteres cuja probabilidade de ocorrência foi listada anteriormente na Tabela 4.6. A Figura 4.13 (parte superior) ilustra a construção de um código Huffman para o conjunto de caracteres, contendo sete caracteres listado na Tabela 4.6. A porção inferior daquela ilustração mostra a atribuição de binários 1s e 0s para cada membro de segmento e o código Huffman resultante para cada caractere, quando os dígitos binários em cada segmento são registrados, começando do unitário ou ponto vértice na árvore de codificação.

A Tabela 4.7 compara os códigos gerados pelo procedimento de codificação Shannon-Fano e pelo procedimento de codificação Huffman para o conjunto de caracteres, contendo sete caracteres, usado para cada exemplo de codificação. O comprimento médio de código gerado por cada procedimento de codificação pode ser computado, usando a fórmula:

$$L = \sum_{i=1}^{7} L_i P_i$$

Para o código Shannon-Fano, o tamanho médio de código é:

L = 2*0.25 + 2*0.20 + 3*0.15 + 3*0.15 + 3*0.10 + 4*0.10 + 4*0.05 = 2.7 bits.

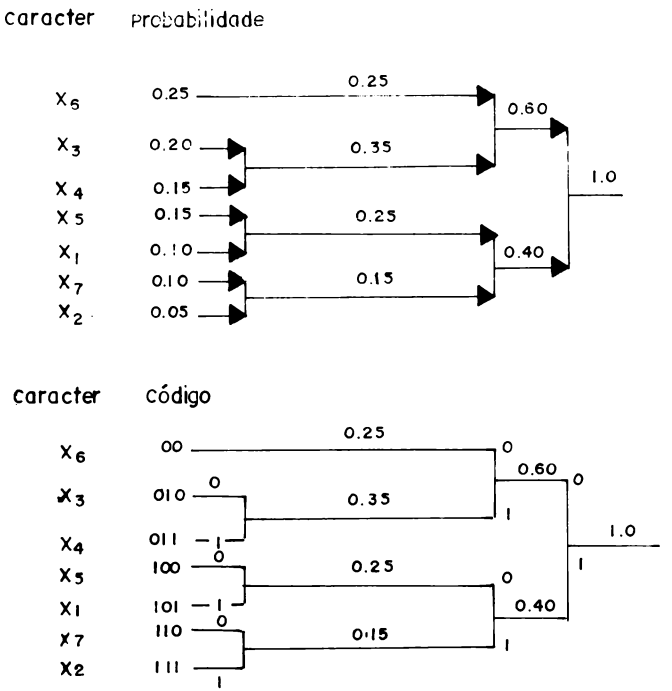


Figura 4.13 - Construção de código Huffman.

Tabela 4.7 - Comparação de código

Caractere	Probabilidade	Código Shannon-Fano	Código Huffman
X_6	0.25	11	00
X_3	0.20	10	010
X_4	0.15	011	011
X_5	0.15	010	100
X_1	0.10	001	101
X_7	0.10	0001	110
X_2	0.05	0000	111

Para o código Huffman desenvolvido na Figura 4.13, o comprimento médio de código é:

$$L = 2*0.25 + 3*0.75 = 2.75 \text{ bits.}$$

Embora o código Shannon-Fano pareça ser a mais eficiente, uma vez que seu comprimento médio de código é menor do que o do código Huffman, na realidade, não pode ser mais eficiente. Isto se deve ao fato do código Huffman resultar em um timo código quando a construção de código é eficiente, que propositadamente não foi o caso na construção ilustrada na Figura 4.13. Naquela ilustração, o autor, propositadamente, não moveu os grupos de freqüências de ocorrências para o mais alto possível de uma maneira vantajosa. Isto foi feito porque, de vez em quando, é fácil construir um código Huffman que não represente um código timo, sendo que a criação de um código Shannon-Fano pode ser usada como um método, para verificar se a construção do código Huffman teve como resultado um timo código ou não. Isto é realizado ao comparar o comprimento médio de bits do código Shannon-Fano com o comprimento médio de bits do código Huffman. Se o comprimento médio de bits do código Huffman ultrapassar o comprimento médio de bits do código Shannon-Fano, este fato deve ser usado como um indicador de que você deve tentar reconstruir seu código Huffman. Uma vez que o comprimento médio de bits Shannon-Fano era menor do que o comprimento médio de código Huffman, vamos aplicar o que acabamos de aprender e reconstruir o código Huffman. Esta reconstrução encontra-se ilustrada na Figura 4.14.

Observe que ao reconstruir o código Huffman, mais passos combinatórios ocorrem para agrupamentos de caracteres de menor freqüência de ocorrência, do que para caracteres de maior freqüên-

cia de ocorrência em comparação com a construção original do código ilustrado na Figura 4.13. Embora isto resulte em um bit de código adicional, sendo requisitado para representar os caracteres X_7 e X_2, ele também resulta em um bit a menos requisitado para representar X_3. Uma vez que a probabilidade de X_3 ultrapassa a probabilidade combinada de X_7 e X_2, o código reconstruído terá um menor comprimento médio de bits menor. Para verificar este fato, vamos computar o comprimento médio de bits para o código Huffman desenvolvido na Figura 4.14. Desta maneira, obtemos:

$$L = 2*0.25 + 2*0.20 + 3*0.15 + 3*0.15 + 3*0.10 + 4*0.10 + 4*0.05 = 2.7 \text{ bits.}$$

Observe que o comprimento médio de código foi reduzido em 0.05 bits e é, agora, exatamente igual ao comprimento médio de bits do código Shannon-Fano.

As ilustrações anteriores foram baseadas em um grupo de probabilidades de ocorrência, atribuídas a um conjunto de sete caracteres. Para ilustrar como as eficiências entre os dois códigos podem mudar, vamos assumir que as probabilidades de ocorrência dos caracteres no conjunto são, agora, representadas pelos dados listados na Tabela 4.8.

A porção superior da Figura 4.15 ilustra o processo de codificação Shannon-Fano, enquanto a porção inferior daquela ilustração mostra o processo de codificação Huffman. Observe que com base nas revisões da probabilidade de ocorrência dos caracteres no

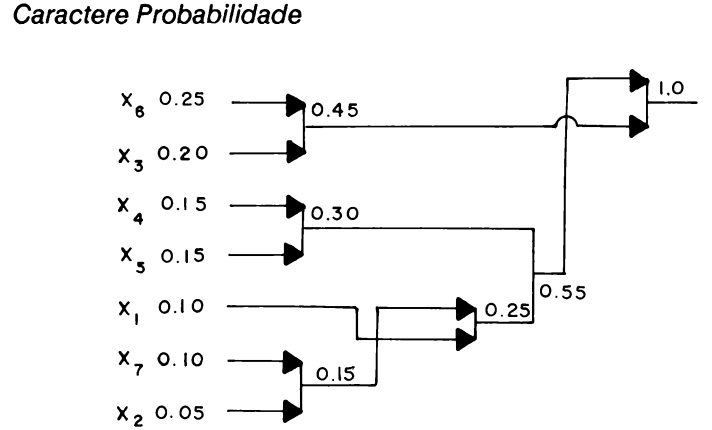


Figura 4.14 - Reconstrução do código Huffman.

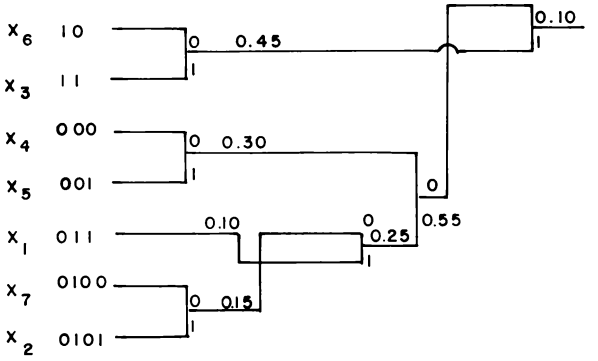


Figura 4.14 - continuação

Tabela 4.8 - Conjunto de caracteres revisado

Caractere	Probabilidade de ocorrência
X_1	0.0625
X_2	0.0625
X_3	0.1250
X_4	0.1250
X_5	0.0625
X_6	0.5000
X_7	0.0625

A. Codificação Shannon-Fano

X_6	0.50	1
X_3	0.125	0 1 1
X_4	0.125	0 1 0
X_5	0.0625	0 0 1 1
X_1	0.0625	0 0 1 0
X_7	0.0625	0 0 0 1
X_2	0.0625	0 0 0 0

B. Codificação Huffman

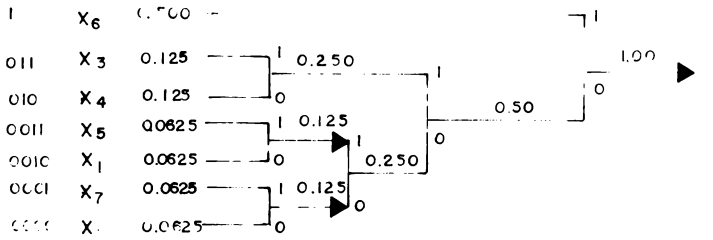


Figura 4.15 - Recodificando o novo conjunto de caracteres.

conjunto de caracteres, o comprimento médio de código para cada técnica de codificação é a mesma. Isto é, o comprimento médio de código para o processo de codificação Shannon-Fano é:

$$L = 1*0.5 + 3*0.125 + 3*0.125 + 4*(4*0.0625) = 2.25 \text{ bits}$$

que é exatamente o mesmo comprimento de código obtido do processo de codificação Huffman.

Agora, vamos supor que a probabilidade de ocorrência de cada caractere no conjunto, seja novamente alterada. Suponha que as novas probabilidades de ocorrência são como as indicadas na Tabela 4.9.

Tabela 4.9 - Novas probabilidades do conjunto de caracteres

Caractere	Probabilidade de ocorrência
X_1	0.10
X_2	0.10
X_3	0.10
X_4	0.10
X_5	0.40
X_6	0.10
X_7	0.10

A. Codificação Shannon-Fano

X_6	0.40	1	1			
X_1	0.10	1	0			
X_2	0.10	0	1	1		
X_3	0.10	0	1	0		
X_4	0.10	0	0	1		
X_5	0.10	0	0	0	1	
X_7	0.10	0	0	0	0	

B. Codificação Huffman

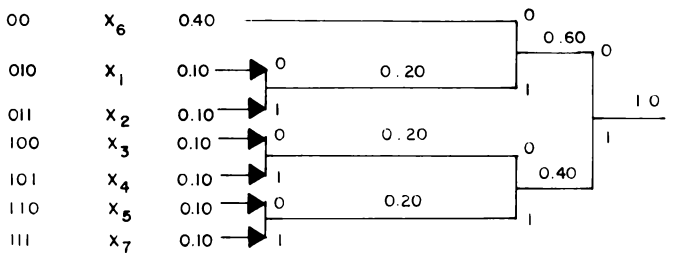


Figura 4.16 - Recodificando o conjunto de caracteres revisado.

A porção superior da Figura 4.16 ilustra o processo de codificação Shannon-Fano para o conjunto de caracteres revisado, enquanto a porção inferior mostra o processo de codificação Huffman.

Agora, vamos computar o comprimento médio de código para cada processo de codificação. Para o código Shannon-Fano, seu comprimento médio de código é:

$$L = 2*0.4 + 2*0.1 + 3*0.1 + 3*0.1 + 3*0.1 + 4*0.1 + 4*0.1 = 2.7 \text{ bits.}$$

Para o código Huffman, seu comprimento médio de código é:

$$L = 2*0.4 + 3*0.6 = 2.6 \text{ bits.}$$

Assim, neste exemplo, o código Huffman resulta em uma representação mais eficiente de bits para o conjunto de caracteres do que o método de codificação Shannon-Fano.

Em geral, quando as probabilidades de cada caractere no conjunto de caracteres se aproximam das probabilidades que são

potências negativas de 2, ambos os códigos terão seus comprimentos médios próximos da entropia. Isto é, se todas as probabilidades de caracteres no conjunto de caracteres fossem potências negativas de 2, o comprimento médio de código se igualaria à entropia e a eficiência de cada código seria 100 por cento. Se as probabilidades de ocorrência dos elementos em um conjunto têm uma grande variância, o código Shannon-Fano será mais eficiente, enquanto o código Huffman se torna mais eficiente conforme a variância das probabilidades diminui entre os elementos do conjunto.

4.5 CóDIGOS DE VíRGULA

Como discutido anteriormente neste capítulo, os métodos de codificação Shannon-Fano e Huffman têm diversas vantagens associadas ao seu uso. Em primeiro lugar, eles geram códigos cujo número médio de bits representa ou se aproxima do código timo. Em segundo lugar, os dois métodos de codificação resultam na ausência de palavras de código pequenas, que são prefixos de palavras de código maiores. Isto significa que cada código é instantaneamente decodificável. Contudo, o que podemos dizer sobre as etapas de processamento exigidas, para decodificar um código de decodificação instantânea e o efeito de um bit perdido sobre uma seqüência recebida de bits?

Com referência ao processo de decodificação, considere o código desenvolvido na Figura 4.16. As etapas de processamento necessárias para decodificar uma seqüência de bits recebida nas representações de caracteres apropriados, exigiram uma série significativa de comparações de bits, como ilustrado na Figura 4.17. Assim, podemos afirmar que os códigos Shannon-Fano e Huffman são códigos de processamento intensivo.

Com referência ao bit perdido ou à seqüência de bits perdidos, esta ocorrência resultaria na perda da sincronização entre o fluxo de bits a serem decodificados e o processo de decodificação. Esta situação obviamente resultaria em erros de decodificação. Devido a este fato, a codificação estatística deve sempre ser encapsulada dentro de um mecanismo de detecção e correção de erros.

Para aliviar a quantia substancial de processamento associada à codificação estatística, bem como o efeito da perda de um ou mais bits, você pode considerar o uso de um código de vírgula. Este nome de código se deve ao fato dele conter um símbolo de término, ou vírgula no fim de cada palavra de código.

A Figura 4.18 contém um código de vírgula desenvolvido para o conjunto de caracteres com sete símbolos, para os quais construí-

mos anteriormente o código Huffman na Figura 4.16. Ao construir o código de vírgula, usamos um bit '1' como uma vírgula ou símbolo de término. Quando aquele bit é encontrado, informa ao receptor que começará um novo código de caractere, fornecendo um método de sincronização dentro do código.

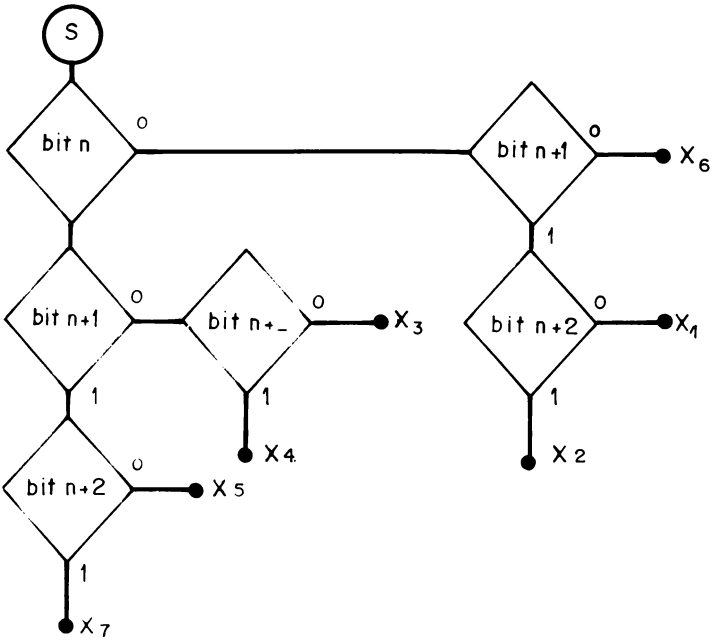


Figura 4.17 - O processo de decodificação para o código Huffman desenvolvido na Figura 4.16.

X_6	.4	1
X_1	.1	01
X_2	.1	001
X_3	.1	0001
X_4	.1	00001
X_5	.1	000001
X_7	.1	0000001

Figura 4.18 - O código de vírgula para o conjunto de caracteres com sete símbolos.

Ao examinar o código de vírgula listado na Figura 4.18, vamos computar seu comprimento médio de bits. Desta forma, obtemos:

$$L = 0.4*1 + 0.1*2 + 0.1*3 + 0.1*4 + 0.1*5 + 0.1*6 + 0.1*7 = 3.1 \text{ bits.}$$

Aqui, o número médio de bits por caractere, que era de 2.6 quando o código Huffman foi usado, aumentou para 3.1 bits para o código de vírgula. Isto ilustra a troca chave entre o código de vírgula e a codificação Huffman e Shannon-Fano - exigências de sincronização e processamento são limitantes de eficiência.

4.6 COMPRESSÃO AUTO-ADAPTÁVEL

Os exemplos de técnicas de compressão abordadas anteriormente no Capítulo 3 e neste capítulo, foram baseadas na suposição de conhecimento prévio dos dados a serem comprimidos. Usando este conhecimento prévio, permite-nos predefinir caracteres indicadores de compressão e as seqüências de caracteres que podem, então, ser substituídas por cadeias de dados contendo redundância predefinida. Além disso, podemos construir uma tabela de compressão fixa que capacitará a codificação estatística de dados, para que esta ocorra com base na freqüência esperada de ocorrência dos dados. A codificação de comprimento de fileira e a codificação diatômica são exemplos de seqüência de caracteres e de substituição de caracteres, onde algum conhecimento prévio ou expectativa de composição dos dados resultaram na definição de um único caractere, ou pequena seqüência de caracteres, para substituir longas seqüências de caracteres. A codificação Huffman e a Huffman modificada são exemplos de técnicas de compressão de dados que empregariam uma tabela fixa de compressão, cuja construção esteja fundamentada no conhecimento prévio ou no conhecimento assumido dos dados.

Tabela de compressão fixa

A Figura 4.19 ilustra o formato geral de uma tabela fixa de compressão. Na realidade, esta tabela pode ser duas tabelas separadas, com uma relação estabelecida entre os elementos de cada tabela, ou a tabela pode consistir de entradas em pares. Cada caractere no fluxo de dados original é comparado às entradas na parte de 'dados a comprimir' da tabela de compressão. Quando o caractere a ser codificado combina com uma entrada na parte de 'dados a comprimir' da tabela, o código que representa o caractere é extraído da tabela de compressão.

Figura 4.19 - Formato de tabela fixa de compressão.

Assim, o processo necessário para substituir cada caractere com seu código estatístico é reduzido a uma operação de busca em tabela.

Para ilustrar a utilização de uma tabela fixa de compressão, vamos supor que como resultado de uma análise do conjunto de caracteres com quatro caracteres (X_1,X_2,X_3 e X_4) determinamos que a probabilidade de ocorrência de cada caractere seja 0.5625, 0.1875, 0.1875 e 0.0625, respectivamente. O código Huffman previamente desenvolvido na Figura 4.3 para este conjunto de caracteres, resulta na atribuição de 0, 10, 110 e 111 aos caracteres X_1 a X_4. Assim, com base no conhecimento prévio dos dados, podemos desenvolver o código Huffman para o conjunto de caracteres que, então, nos permite gerar a tabela fixa de compressão para este conjunto de caracteres. Esta tabela está ilustrada na Figura 4.20.

Dados a comprimir	Código resultante
X_1	0
X_2	10
X_3	110
X_4	111

Figura 4.20 - Tabela fixa de compressão resultante.

A probabilidade de ocorrência dos caracteres no conjunto de caracteres, deve ser determinada antes de construir a tabela fixa de compressão.

O uso de uma tabela fixa de compressão requer que todos os caracteres na cadeia de dados original sejam comparados com as entradas de "dados a comprimir" na tabela. Quando a combinação ocorre, a entrada codificada troca, assim, o caractere na cadeia de dados original. Deste modo, a seqüência de caracteres

$$X_2X_4X_1X_2X_2$$

seria trocada por códigos Huffman para cada caractere, o qual resultaria na seqüência binária:

1011101010.

Eficiência

O que acontece à eficiência do código Huffman predefinido, quando a probabilidade de ocorrência dos caracteres no conjunto de caracteres difere do conhecimento prévio ou esperado de sua freqüência de ocorrência. Uma vez que códigos pequenos são empregados para representar caracteres de ocorrência freqüente, e códigos maiores representam caracteres de ocorrência menor, a variância da entropia do código Huffman predefinido aumenta à medida que os dados variam de sua freqüência de ocorrência anterior ou esperada. Uma técnica que pode ser usada para manter a eficiência do código resultante obtido ao comprimir os dados estatisticamente, é o uso de um esquema de compressão dinâmico ou adaptativo, que é o tópico principal desta seção.

Compressão auto-adaptável

Quando a compressão auto-adaptável é executada, os dados a serem comprimidos são analisados, de maneira a gerar mudanças apropriadas dentro da tabela variável de compressão.

Semelhante ao uso de uma tabela fixa de compressão, todos os caracteres no fluxo de dados original são comparados, primeiramente, com as entradas na parte de 'dados a comprimir' da tabela de compressão. Quando a combinação ocorre, a entrada correspondente na parte do 'código resultante' da tabela é extraída e representa o caractere codificado estatisticamente.

O lugar onde a compressão auto-adaptável difere da compressão fixa é no emprego de um campo de contagem na tabela de compressão. Este campo é atualizado continuamente e serve como um mecanismo para o reordenamento das entradas na tabela. A atualização do campo ocorre depois que um caractere no fluxo de dados original, é combinado com uma entrada na parte de 'dados a comprimir' da tabela de compressão e o 'código resultante' é extraído da tabela. Então, ocorre a comparação das entradas no campo de contagem. Com base nos resultados da comparação, o caractere e seu valor de contagem podem ser reposicionados na tabela de compressão. Esta técnica assegura que sempre que a composição de dados mudar, a tabela de compressão muda em série, resultando em uma tabela variável de compressão que fornece a compressão estatística mais eficiente possível. A Figura 4.21 ilustra como a tabela variável de compressão é reordenada com base na composição dos dados sendo transmitidos.

A Figura 4.21, parte A, ilustra a composição inicial da tabela variável de compressão. Embora esta tabela tenha sido inicialmente estabelecida com base na freqüência de ocorrência dos caracteres no conjunto de caracteres, uma vez que a tabela é auto-ajustável, não temos de nos preocupar com o tamanho de amostra usada para iniciar as entradas dentro da tabela.

A. Tabela inicial *Dados transmitidos*

Dados a comprimir	Contagem	Código resultante
X_1	0	0
X_2	0	10
X_3	0	110
X_4	0	111

B. X_2 encontrado *10*

Dados a comprimir	Contagem	Código resultante
X_2	1	0
X_1	0	10
X_3	0	110
X_4	0	111

C. X_4 encontrado *111*

Dados a comprimir	Contagem	Código resultante
X_2	1	0
X_4	1	10
X_1	0	110
X_3	0	111

D. X_2 encontrado *0*

Dados a comprimir	Contagem	Código resultante
X_2	2	0
X_4	1	10
X_1	0	110
X_3	0	111

Figura 4.21 - A tabela variável de compressão.

Na Figura 4.21, parte B, assumimos que o caractere X_2 foi encontrado. Uma vez que o código binário 10 é atribuído a X_2 (Figura 4.21, parte A), esta cadeia de bits é transmitida, a contagem de X_2 é incrementada de um e a tabela variável de compressão é reordenada. De forma semelhante, a seqüência de bit 10 é recebida pelo receptor, que é descomprimido no caractere X_2. O receptor incrementa, então, a contagem para X_2 em sua tabela de compressão e sua tabela é, também, reordenada.

Na Figura 4.21, parte C, assumimos que o caractere X_4 será o próximo caractere encontrado nos dados a serem comprimidos. Com base na tabela em uso (Figura 4.21, parte B), este caractere é codificado como a cadeia binária 111. A seguir, a contagem da freqüência de ocorrência de X_4 é incrementada e a tabela variável de compressão é reordenada.

A Figura 4.21, parte D, assume que o próximo caractere encontrado na cadeia de dados original é X_2. Uma vez que a tabela ilustrada na Figura 4.21, parte C, estava, então, em uso, X_2 é codificado como um único bit 0. Em seguida, a contagem de X_2 é incrementada de um; contudo, uma vez que X_2 estava no topo da tabela de compressão, a tabela não é reordenada.

Como ilustrado na Figura 4.21, a compressão auto-adaptável muda dinamicamente a ordem das entradas na tabela de compressão de acordo com as mudanças na freqüência de ocorrência dos caracteres no conjunto de caracteres. Assim, este método de implementação de técnica de compressão estatística deve sempre ser mais eficiente que a utilização de compressão fixa.

Exemplo Codificado

A Figura 4.22 contém a listagem do programa ADAPTC.BAS. Este programa em linguagem BASIC foi desenvolvido para ilustrar muitos dos conceitos de programação, envolvidos na compressão auto-adaptável. Para simplificar a ilustração somente quatro caracteres - E, T, I e O - são considerados para estarem no conjunto de caracteres adequado à compressão auto-adaptável. Todos os outros caracteres encontrados nas cadeias de dados, com o qual o programa operará, serão passados 'no estado em que se encontram' para o buffer de saída.

Na linha 115, o programa desvia para a sub-rotina, começando na linha 400, que inicia a tabela de caractere P$(I) com os caracteres E, T, I e O. De forma semelhante aos outros exemplos de codificação apresentados neste capítulo, a linha 130 obtém uma linha de até 132 caracteres de um arquivo de dados, enquanto a linha 140 obtém o comprimento da linha.

A sub-rotina, começando na linha 180, processa os registros lidos dos arquivos de dados. Para ilustrar a operação da compressão auto-adaptável, quando os caracteres E, T, I e O são encontrados, eles serão trocados pelos caracteres #, $, % e &. Para simplificar, a tabela Huffman resultante será exibida linha a linha, em vez de caractere a caractere, enquanto o código, de forma semelhante, muda linha a linha na tabela de compressão auto-adaptável.

```
10 REM ADAPTC.BAS PROGRAM
20 DIM O$(132)
30 WIDTH 80:CLS
40 '**********MAIN ROUTINE**********************
50 '* THIS ROUTINE READS RECORDS FROM AN ASCII *
60 '* FILE INTO A STRING CALLED X$ WHICH IS    *
70 '* THEN PASSED TO SUBROUTINES FOR COMPRESSION
80 '*********************************************
90 PRINT "ENTER ASCII FILENAME. EG, ADAPT.DAT"
100 INPUT F$: OPEN F$ FOR INPUT AS #2
105 OPEN "ADAPTC.DAT" FOR OUTPUT AS #3
110 PRINT "PATIENCE - INPUT PROCESSING"
112 PRINT "SUBSTITUTION BASED ON ENTRY IN TABLE: 1=# 2=$ 3=% 4=&"
115 GOSUB 400                    'PAUSE TO SET UP TABLE
120 IF EOF(2) THEN GOTO 9000
130 LINE INPUT #2, X$
140 N= LEN(X$)
150 GOSUB 180
160 GOSUB 900
170 GOTO 120
180 '*****ADAPTIVE COMPRESSION SUBROUTINE*******
190 '* THIS ROUTINE PROCESSES RECORDS FROM X$  *
200 '* AND COMPRESSES WITH HUFFMAN CODES       *
210 '* USING O$ AS THE OUTPUT BUFFER.          *
220 '*********************************************
230 GOSUB 2120                   'PRINT HUFFMAN TABLE USED
235 I=1                          'RESET INDICES
240 FOR J= 1 TO N                'STEP THRU RECORD
250 A$= MID$(X$,J,1)             'EXTRACT A CHARACTER
260 FOR K = 1 TO 4               'SETUP HUFFMAN LOOP
270 IF A$=P$(K) THEN GOSUB 350   'IS INPUT CHAR IN TABLE?
280 NEXT K                       'NO - TRY NEXT
290 IF M = 1 THEN 310            'IS MATCH FLAG SET?
300 O$(I) = MID$(A$,1,1)         'NO-STUFF CHAR IN BUFFER
310 I=I+1                        'BUMP INPUT STRING INDEX
320 M=0                          'RESET MATCH FLAG
330 NEXT J                       'GO BACK FOR MORE
340 RETURN                       'DONE
350 M=1                          'SET CHAR MATCH FLAG
355 '*********************************************************
360 'INSERT COMPRESSION NOTATION IN OUTPUT BUFFER
365 V = K + 34                   'INDEX OUT TO SUBSTITUTE CHAR
370 O$(I)=CHR$(V)                'INSERT SUBSTITUTION
380 P(K)= P(K) + 1               'BUMP COUNT OF OCCURANCE
390 K = 4                        'FORCE END OF SEARCH
395 RETURN                       'GO BACK FOR MORE
400 DIM P$(4)                    'COMMON HUFFMAN CANDIDATES
410 DATA E,T,I,O
420 FOR I = 1 TO 4               'SETUP CHARACTER TABLE
430 READ Z$                      'GET CHARACTER
440 P$(I) = Z$: NEXT I           'AND STUFF INTO TABLE
450 RETURN                       'DONE - TABLE COMPLETE
```

Figura 4.22 - Listagem do Programa ADAPTC.BAS.

```
900 '******TALLY THE COMPRESSION COUNT & WRITE BUFFER******
910 '* DISPLAY BEFORE & AFTER RESULTS OF COMPRESSION      *
920 '* AND SHOW THE NET RESULTS OBTAINED BY EACH METHOD   *
930 '*************************************************
931 N1=N1+N                          'TALLY INPUT CHAR COUNT
932 T=N-I+1                          'NET DIFFERENCE IN BUFFERS
936 T1=T1+T                          'SAVE COUNT FOR SUMMARY
940 FOR I=1 TO J-1
950 PRINT #3, O$(I);
960 NEXT I
965 PRINT #3, ""
966 GOSUB 2000                       'RESEQUENCE HUFFMAN TABLE
970 RETURN
2000 '*****RESEQUENCE & PRINT TABLE FOR ADAPTIVE COMPRESSION*****
2010 FOR J=1 TO 3                    'SETUP 1ST LOOP
2020 FOR K=J+1 TO 4                  'SETUP 2ND LOOP
2030 IF P(J) >= P(K) THEN 2100       'IS CURRENT ENTRY GREATER?
2040 TEMP= P(J)                      'NO-SAVE IN TEMP
2050 TEMP$= P$(J)                    'AND SAVE CHAR
2060 P(J)= P(K)                      'PICKUP GREATER COUNT
2070 P$(J)= P$(K)                    'AND ASSOC CHAR
2080 P(K)= TEMP                      'SWAP LESSER COUNT
2090 P$(K)= TEMP$                    'AND ASSOC CHAR
2100 NEXT K                          'FINISH 2ND LOOP
2110 NEXT J                          'FINISH 1ST LOOP
2115 RETURN                          'DONE-TABLE RESEQUENCED
2120 L= L + 1                        'REMEMBER LINE NO.
2130 PRINT "HUFFMAN TABLE USED FOR LINE";L;": ";
2140 FOR I=1 TO 4                    'SETUP PRINT TABLE LOOP
2150 PRINT P$(I);:PRINT P(I);        'PRINT CHAR AND COUNT
2160 NEXT I
2175 PRINT
2180 RETURN                          'DONE-TABLE PRINTED
9000 CLOSE: OPEN F$ FOR INPUT AS #2
9010 PRINT "FILE ";F$;" BEFORE SUBSTITUTION:"
9020 LINE INPUT #2,X$
9030 IF EOF(2) THEN 9060
9040 PRINT X$
9050 GOTO 9020
9060 PRINT X$:OPEN "ADAPTC.DAT" FOR INPUT AS #3
9070 PRINT "FILE ";F$;" AFTER SUBSTITUTION:"
9080 LINE INPUT #3,O$
9090 IF EOF(3) THEN 9998
9100 PRINT O$
9110 GOTO 9080
9998 PRINT O$
9999 CLOSE:END
```

Figura 4.22 - continuação.

Na linha 230, a sub-rotina começando na linha 2120 é invocada. Esta sub-rotina imprime os valores correntes da tabela de compressão. A seguir, as linhas 240 e 280 examinam o registro extraído do arquivo de dados linha por linha, comparando cada caractere do registro com todos os caracteres no conjunto compressível de quatro caracteres (E, T, I, O). Se a combinação ocorre, a sub-rotina começando na linha 350 é invocada. De outra forma, o programa simplesmente coloca o caractere extraído do registro de entrada dentro do buffer de saída.

A sub-rotina, começando na linha 350, estabelece o flag de comparação de caractere com 1 e, então, acrescenta 34 ao valor de K na linha 365. Esta ação define o valor ASCII de V para o caractere #, $, % ou &, que é usado neste exemplo para ilustrar a substituição de um código Huffman, para um caractere apropriado no conjunto de caracteres, com quatro caracteres que estamos usando. A seguir, a linha 370 insere o caractere substituído dentro do buffer saída e a contagem é, então, incrementada na linha 380.

Quando o flag de comparação é estabelecido, a linha 290 provoca um salto para a linha 310, onde o índice da cadeia de entrada é incrementado de um, que, a seguir, o flag de comparação é reestabelecido com zero na linha 320. Se o flag de comparação não foi definido, a linha 300 simplesmente extrai um caractere de sua posição apropriada no registro de entrada e coloca-o no buffer de saída.

Cada vez, antes de uma linha de entrada ser processada no programa, a chamada da sub-rotina contida na linha 230 será invocada. Esta sub-rotina simplesmente imprime o status atual da tabela de compressão 'Huffman' auto-adaptável, incluindo a ordem de caractere e a freqüência de ocorrência de cada caractere. Embora este programa tenha sido construído para facilitar a observação visual das mudanças na tabela de compressão auto-adaptável linha por linha, as tabelas estariam sujeitas à mudança a cada caractere individual, ao se desenvolver uma rotina de compressão auto-adaptável real.

O reordenamento real da tabela de compressão auto-adaptável ocorre nas linhas 2000 a 2115 do programa. Este módulo de sub-rotina ordena os caracteres na tabela de compressão auto-adaptável com base em sua freqüência de ocorrência.

A Figura 4.23 ilustra a execução experimental do programa ADAPTC.BAS, com os status da tabela de compressão exibidos para cada linha de dados no arquivo a ser processado. Além disso, o programa exibe o conteúdo do arquivo antes e depois da substituição de caracteres do conjunto de quatro caracteres definido

anteriormente. Como um exemplo da operação do programa, observe que antes da linha 1 ser processada, todas as entradas na tabela de compressão têm a contagem em zero e a ordem das entradas é E, T, I, O.

A primeira linha no arquivo de dados contém a cadeia BEGIN, seguida por muitos asteriscos. Uma vez que os caracteres E e I serão trocados pelos códigos 'Huffman' # e %, depois da linha 1 ser processada a contagem de E e I deve ser um, enquanto a tabela de compressão auto-adaptável deve ser reordenada para computar a nova freqüência de ocorrência.

```
ENTER ASCII FILENAME. EG, ADAPT.DAT
? ADAPT.DAT
PATIENCE - INPUT PROCESSING
SUBSTITUTION BASED ON ENTRY IN TABLE: 1=# 2=$ 3=% 4=&
HUFFMAN TABLE USED FOR LINE 1 : E 0 T 0 I 0 0 0
HUFFMAN TABLE USED FOR LINE 2 : E 1 I 1 T 0 0 0
HUFFMAN TABLE USED FOR LINE 3 : I 5 0 5 T 3 E 2
HUFFMAN TABLE USED FOR LINE 4 : 0 9 E 7 T 5 I 5
HUFFMAN TABLE USED FOR LINE 5 : 0 19 I 13 T 11 E 11
FILE ADAPT.DAT BEFORE SUBSTITUTION:
1 BEGIN**********************************
2 OVATION OVATION FOR THE MUSICIAN
3 ENCORE ENCORE FOR THE ACTOR
4 OOOOOOOOO IIIIIIII TTTTTT EEEE
5 **************************************END
FILE ADAPT.DAT AFTER SUBSTITUTION:
1 B#G%N**********************************
2 &VA%$&N &VA%$&N F&R %H# MUS$C$AN
3 &NC$R& &NC$R& F$R %H& AC%$R
4 ########## &&&&&&&& %%%%% $$$$
5 **************************************&ND
Ok
```

Figura 4.23 - Execução experimental do programa ADAPTC.BAS.

Ao examinar a tabela Huffman usada na linha 2 na Figura 4.23, o leitor observará que a contagem de E e I está estabelecida em 1, enquanto a ordem dos caracteres na tabela foi rearranjada para levar em consideração sua nova freqüência de ocorrência.

Ao examinar a linha 2 no arquivo de dados ADAPT.DAT, o leitor observará que OVATION contém quatro caracteres que podem ser substituídos pelo código 'Huffman' adaptativo. Uma vez que o caractere O não muda de lugar na tabela de compressão, o código 'Huffman' & é colocado em seu lugar. A seguir, o T do OVATION, que seria inicialmente trocado pelo código 'Huffman' $, é trocado

pelo código 'Huffman' %, já que as entradas da tabela auto-adaptável mudaram e, assim, provocam mudanças nas substituições de código 'Huffman'. A Tabela 4.10 resume as mudanças na tabela de compressão auto-adaptável, antes e depois da primeira linha de dados no arquivo de entrada ser processada. Como exercício, o leitor pode querer seguir as substituições de código do conjunto com quatro caracteres, para as linhas restantes no arquivo ADAPT.DAT que são processados pelo programa ADAPTC.BAS.

Tabela 4.10 Mudança na tabela de compressão auto-adaptável

Tabela inicial				
Seqüência de caractere	E	T	I	O
Substituição de código	#	$	%	&
Depois de uma linha processada				
Seqüência de caractere	E	I	T	O
Substituição de código	#	$	%	&

Considerações do campo de contagem

Até agora, assumimos que o campo de contagem da tabela variável de compressão era infinito. Obviamente, o campo de contagem tem um valor finito, baseado no tamanho do campo que deve ser considerado ao se desenvolver uma técnica de compressão auto-adaptável.

Há dois métodos de fácil implementação que você pode considerar, para eliminar a possibilidade da ocorrência de um estouro do campo de contagem. Estes métodos são fundamentados no uso da divisão inteira, para reduzir os valores previamente atribuídos ao campo de contagem, enquanto mantém a ordem relativa da freqüência de ocorrência dos caracteres transmitidos.

Um método para eliminar a ocorrência de estouro do campo de contagem é examinar os valores da entrada do campo superior depois de uma atualização de campo. Então, se os bits no campo estiverem todos definidos, você implementaria um algoritmo que executaria a divisão inteira por 2 em cada entrada de campo. Ao implementar esta técnica resultaria em restos depois da divisão, que seriam descartados. Como exemplo, 3/2 se igualaria a 1, com o restante sendo descartado.

O segundo método, que pode ser usado para evitar o estouro do campo de contagem, seria, simplesmente, a contagem dos caracteres transmitidos e a implementação da divisão inteira por 2, cada vez que a contagem se igualar ao valor de campo máximo. Embora este método resulte obviamente em operações de divisão inteira mais freqüentes, ele elimina a necessidade de comparar o valor do campo de contagem na posição mais alta, analisando a condição de todos os bits definidos cada vez que um caractere é recebido.

4.7 MÉTODOS MODERNOS DE COMPRESSÃO ESTATÍSTICA

Ao concluir este capítulo, voltaremos nossa atenção para diversos métodos populares de compressão de dados. Todos os métodos a serem examinados nesta seção estão relacionados com uma ou mais técnicas de compressão estatística abordadas neste capítulo. Além disso, todos os métodos incluem uma ou mais propriedades únicas que resultam em seu exame, fornecendo-nos informações adicionais a respeito da implementação e operação de compressão de dados.

MNP Classe 5

A compressão de dados MNP Classe 5 é um dos dois métodos de compressão de dados mantidos pela Microcom Networking Protocol (MNP). O protocolo MNP foi desenvolvido pela Microcom, Inc., para aumentar a capacidade de comunicações entre modems e é, atualmente, composto de 10 classes. Cada uma suporta operações da classe inferior, fornecendo compatibilidade decrescente entre produtos diferentes que suportam classes MNP diferentes.

As operações definidas pelo MNP incluem a compactação auto-adaptável de dados, conversão de dados assíncronos em um formato síncrono, esquemas diferentes de modulação, além de dois tipos de compressão de dados. A compressão de dados MNP Classe 5 foi o primeiro método de compressão incorporado no protocolo. Como explicado anteriormente no Capítulo 3, MNP Classe 5 incorpora duas manipulações para um fluxo de dados transmitidos. A primeira manipulação é uma versão de codificação

de comprimento de fileira na qual a seqüência de três ou mais caracteres de repetição, são comprimidos via inserção de uma contagem depois do terceiro caractere e a remoção de todos os caracteres repetidos após o terceiro. Sob a versão MNP Classe 5 de codificação de comprimento de fileira, se a seqüência de caracteres repetidos tem o comprimento de três caracteres, a seqüência é seguida por uma contagem de repetição 0. A máxima contagem de repetição suportada é 250.

Tanto a contagem do comprimento de fileira, como as três repetições de um caractere, como qualquer caractere no fluxo de dados, são comprimidos através do uso de uma técnica de codificação de freqüência auto-adaptável.

O método de codificação de freqüência auto-adaptável empregado pela compressão de dados MNP Classe 5 resulta na substituição por um símbolo de compressão para cada caractere de dados de oito bits. O símbolo usado muda com a freqüência de ocorrência do caractere de dados real, de maneira que símbolos menores são substituídos, onde os caracteres de dados que ocorrem mais freqüentemente, semelhante ao método descrito na Seção 4.6 que abordou a compressão auto-adaptável.

A formação de símbolos para representar caracteres de oito bits foi fundamentada no fato de que há 2^8 ou 256 caracteres diferentes, que podem ocorrer e todos os caracteres devem ser mapeados por 256 códigos diferentes. Uma vez que o ASCII convencional e o conjunto sem o oitavo bit EBCDIC representam a maioria dos caracteres usados na transmissão de dados, a Microcom considerou o fato de que há 128 caracteres, com menos de 8 bits, no desenvolvimento dos símbolos. Os símbolos de compressão foram desenvolvidas para reconhecer este fato, representando os caracteres que ocorrem mais freqüentemente com códigos que têm os zeros precedentes removidos. Para separar uns dos outros, os caracteres de dados recodificados, um símbolo de duas partes foi empregado. A primeira parte do símbolo, conhecido como cabeçalho, tem 3 bits de comprimento, capacitando, assim, um comprimento variável de até 8 bits para ser definido. Assim, o cabeçalho do símbolo pode ser visto como um código de vírgula reverso. A segunda parte do símbolo é referida como o corpo do símbolo e é variável no comprimento. O comprimento do corpo do símbolo é definido pela composição de bits do cabeçalho, que pode variar em valor do binário 000 a 111.

A Tabela 4.11 lista a relação entre o valor decimal de cada caractere e o símbolo substituído no início da compressão. Ao examinar as entradas na Tabela 4.11, há três casos especiais que

não seguem as relações descritas anteriormente. Primeiro, há dois símbolos com o cabeçalho 000, onde o comprimento real do corpo é 1. Finalmente, quando a porção do cabeçalho do símbolo indica um comprimento 7 e o corpo tem sete bits 1, o comprimento real do corpo é 8 bits. De outra forma, a porção do cabeçalho do símbolo indica o comprimento do corpo do símbolo, que tem comprimento variável.

Quando a compressão MNP Classe 5 é iniciada, a freqüência de ocorrência de cada um dos 256 caracteres possíveis é zero. Para fins de mapeamento caractere-para-símbolo, assume-se que o caractere cujo valor de bit representa o decimal 0 (binário 00000000) seja o caractere que ocorre mais freqüentemente e é representado pelo primeiro dos símbolos menores na Tabela 4.11. Os caracteres com valores decimais crescentes são representados por símbolos sucessivos, até assumirmos que o caractere cujo valor decimal é 255 (binário 11111111) seja o caractere que ocorre com maior freqüência.

À medida que um caractere é processado, este é substituído pelo símbolo no qual seu valor decimal esteja atualmente mapeado. Depois que esta substituição ocorre, a freqüência de ocorrência do caractere é incrementada de um. Se a freqüência de ocorrência do caractere que acabou de ser codificado em um símbolo, é maior depois da adição, do que a freqüência do próximo caractere mais freqüente, então os símbolos de compressão do caractere atual e do próximo caractere que ocorre com maior freqüência, serão trocados. A freqüência do caractere corrente é, então, comparada à freqüência do caractere que é agora o caractere que ocorre com maior freqüência. Se a freqüência do caractere corrente for maior, então os símbolos comprimidos são, mais uma vez, trocados. Este processo é repetido até que não sejam necessárias mais trocas, à época em que o mapeamento dos caracteres e dos símbolos de compressão esteja corretamente adaptado com base na freqüência de ocorrência dos caracteres processados.

Tabela 4.11 - Mapeamento inicial caractere-para-símbolo

Valor decimal de caracteres de dados	Composição do símbolo	
	Cabeçalho	Corpo
0	000	0
1	000	1
2	001	0
3	001	1
4	010	00
5	010	01
6	010	10
7	010	11
8	011	000
9	011	001
10	011	010
11	011	011
12	011	100
13	011	101
14	011	110
15	011	111
16	100	0000
17	100	0001
18	100	0010
19	100	0011
20	100	0100
21	100	0101
22	100	0110
23	100	0111
24	100	1000
25	100	1001
26	100	1010
27	100	1011
28	100	1100
29	100	1101
30	100	1110
31	100	1111
32	101	00000
33	101	00001
34	101	00010
(35-246 continua no mesmo padrão)		
247	111	1110111
248	111	1111000
249	111	1111001
250	111	1111010

Tabela 4.11 - (continuação)

Valor decimal de caracteres de dados	Composição do símbolo	
	Cabeçalho	Corpo
251	111	1111011
252	111	1111100
253	111	1111101
254	111	1111110
255	111	11111110

A freqüência de ocorrência para cada caractere é um campo de 8 bits que tem um valor máximo de 255 (binário 11111111). Para evitar que o campo estoure, toda vez que os mapeamentos dos símbolos de compressão de caractere são ordenados em ordem pela freqüência, onde a contagem de freqüência do caractere atual é comparada ao valor de contagem máxima 255. Se a contagem máxima foi alcançada, a freqüência de ocorrência de todos os caracteres é representada em escala decrescente ao dividir cada freqüência por 2, usando a divisão inteira.

Como discutido anteriormente no Capítulo 3, e também brevemente mencionado nesta seção, as seqüências de três ou mais caracteres de repetição são primeiramente colocadas em uma seqüência codificada de comprimento de fileira com 4 caracteres. Aqui os três primeiros caracteres são caracteres reais repetidos, enquanto o quarto caractere é a contagem de repetição. Embora os caracteres repetidos e a contagem sejam mapeados por símbolos, o caractere de contagem não é usado para aumentar a freqüência de ocorrência do caractere para o qual o símbolo é mapeado.

Flushing

Durante o processo de compressão, os caracteres de oito bits podem ser codificados em símbolos que variam de 4 a 11 bits de comprimento. Uma vez que os computadores operam com caracteres de 8 bits, pode ser necessário terminar uma seqüência de transmissão inserindo um número de bits de 'atenuação' para preencher a formação de oito bits. Para realizar isto sob a compressão MNP Classe 5, o transmissor insere um símbolo especial dentro do fluxo de dados, depois do último símbolo de dados do usuário. O símbolo especial é conhecido como um símbolo flush e tem a

composição de bit 11111111111. Depois do símbolo flush, bits '1' são suplementados ao adicioná-los na extremidade de ordem inferior do corpo do símbolo flush, como necessários para produzir um número inteiro de caracteres de oito bits.

Eficiência

Embora a eficiência da compressão de dados MNP Classe 5, como todos os métodos de compressão, seja dependente do fluxo de dados que opera, podemos observar diversas características com respeito a sua eficiência. Em primeiro lugar, ao descontar as cadeias repetitivas de caracteres, sua taxa de compressão máxima é 2. Isto se deve ao fato de o menor símbolo, um caractere de oito bits, poder ser mapeado em quatro bits. Em segundo lugar, somente 32 de 256 caracteres serão mapeados em símbolos que têm o comprimento de sete bits ou menos. Isto significa que este método de compressão resultará tanto na ausência de compressão, como na expansão para 224 das 256 representações de caracteres de oito bits. Apesar destas limitações, o uso da codificação de comprimento de fileira, e do fato que os 15 caracteres de ocorrência mais freqüente na maioria dos fluxos de dados cumulativamente excedem 50 por cento de todos os dados e podem ser codificados em seis bits ou menos, podem resultar na taxa de compressão entre 1.5 e 2.0. Assim, a compressão de dados MNP Classe 5 pode fornecer modems de 2400 bps com um rendimento efetivo entre 3600 e 4800 bps. De forma semelhante, espera-se que modems de 9600 bps que suportem a compressão de dados MNP Classe 5, forneçam um rendimento efetivo entre 14.400 e 19.200 bps, ao se comunicar com um modem compatível.

Compressão avançada de dados MNP Classe 7

A compressão avançada de dados MNP Classe 7 é o segundo método de compressão mantida pela Microcom Networking Protocol. Este método de compressão de dados usa o que é conhecido como o modelo Markov de primeira ordem, para predizer a probabilidade de ocorrência de caractere, com base no caractere prévio e executa a codificação Huffman auto-adaptável no fluxo de dados. Além disso, a MNP Classe 7 usa a codificação de comprimento de fileira para comprimir fluxos de caracteres duplicados. Contudo, o método de codificação de comprimento de fileira difere daquele usado pela compressão de dados MNP Classe 5.

A compressão avançada de dados MNP Classe 7 pode ser considerada como um processo de dois estágios. Primeiro, múltiplas cópias consecutivas do mesmo caractere ou padrão de 8 bits são comprimidas, usando a codificação de comprimento de fileira. Sob a MNP Classe 7, se o codificador enviou o mesmo caractere três vezes, ele enviará a contagem dos caracteres idênticos consecutivos restantes através de 4 bits. Uma vez que um algoritmo de codificação de comprimento de fileira é aplicado ao fluxo de dados este é, então, comprimido usando o modelo Markov de primeira ordem para a seleção de código.

O modelo Markov de primeira ordem usado para a seleção de código pode ser visto como uma matriz bidimensional de 256 x 256 elementos, que se encontra ilustrada na Figura 4.24. A linha superior da matriz contém 256 entradas que podem ser vistas como indicadoras para um local na coluna apropriada dentro de cada entrada de linha. Todos os indicadores dentro da linha superior da matriz correspondem a um dos 256 valores de caracteres de 8 bits, enquanto a coluna abaixo do indicador pode ser considerada como uma das 256 tabelas de codificação.

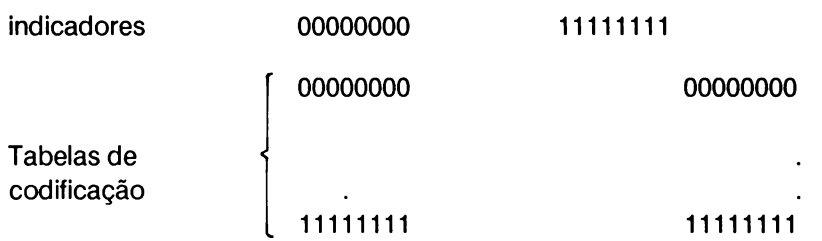


Figura 4.24 - Modelo Markov inicial.

Para codificar um caractere para transmissão, o caractere anterior é usado para selecionar a tabela de codificação apropriada ao usar a entrada do indicador para selecionar a coluna apropriada. A seguir, a coluna é procurada até o caractere corrente ser encontrado. No lugar de transmitir a composição de oito bits do caractere, será usado o código Huffman desenvolvido para aquele caractere. Além disso, a freqüência de ocorrência do caractere é atualizada e a tabela de codificação pode ser ajustada com base na freqüência do caractere transmitido, ao ser comparado com a freqüência de ocorrência de outros caracteres naquela tabela de codificação particular. Assim, todas as tabelas de codificação previamente mostradas na Figura 4.24 são realmente compostas de três campos

Um campo contém a composição de 8 bits de caractere, enquanto os outros campos contêm a freqüência de ocorrência do caractere e seu código Huffman. Semelhante à técnica de codificação Huffman auto-adaptável descrita anteriormente, o campo Huffman é estático e permanece fixo. Em comparação, cada composição de bits do caractere e campos de contagem serão ajustados pela localização na tabela de codificação, com base na freqüência de ocorrência.

O processo de codificação usado pela MNP Classe 7 reconhece o fato que há uma grande probabilidade de muitos caracteres serem seguidos por outros caracteres. Para ilustrar este conceito vamos levar em consideração a língua Inglesa. Aqui, a probabilidade de um U seguindo um Q é muito alta. Assim, a tabela de codificação do Q terá, provavelmente, o caractere U movendo-se rapidamente ao topo da tabela. Uma vez que a codificação Huffman é empregada, normalmente o U seguindo um Q será codificado como um bit 1. Se assumimos que somente as letras do alfabeto estão sendo codificados, uma porção do modelo Markov pode aparecer como indicado na Figura 4.25.

Observe que a letra T está na posição superior na tabela de codificação sob o indicador A, já que a letra T freqüentemente segue a letra A. De forma semelhante, a letra L está ilustrada no topo da tabela de codificação sob o indicador B, já que a letra L freqüentemente segue a letra B.

indicadores	A	B	C	D	E
posições de caracteres dentro das tabelas de codificação	T	L	H	O	D
	H	E	O	A	R
	C	U	R	E	S
	.	.	.	.	.
	.	.	.	.	.

Figura 4.25 - O modelo Markov de primeira ordem processando o alfabeto.

Eficiência

Ao contrário da compressão MNP Classe 5, na qual um mínimo de quatro bits é exigido para representar um caractere, a MNP Classe 7 pode representar um caractere com somente um bit. De fato, o caractere do topo em todas as 256 tabelas da codificação pode ser representado por um bit. Devido ao uso da codificação Huffman auto-adaptável, a compressão avançada de dados MNP Classe 7 pode ser esperada a fornecer, no mínimo, uma taxa de compressão entre 15 e 25 por cento acima daquela compressão obtida, usando MNP Classe 5.

CCITT V.42 bis

Antes de discorrermos sobre a compressão de dados CCITT V.42, é importante fazer diversas observações a respeito de como a eficiência da compressão estatística pode ser aumentada. Assim, seremos capazes de melhor entender por que a técnica de compressão baseada em cadeia Lempel-Ziv, sobre a qual a recomendação CCITT V.42 está baseada, pode-se tornar mais eficiente do que as técnicas de compressão de dados previamente descritas.

Eficiências da compressão de cadeia

Para entender por que a compressão de cadeia fornece maior eficiência do que os métodos de compressão baseados em caractere, vamos primeiramente examinar a entropia de um conjunto com dois caracteres à medida em que a probabilidade de cada caractere muda. Se X_1 e X_2 representam o nosso conjunto de caracteres, vamos assumir que X_1 ocorre 90 por cento das vezes e que X_2 ocorre 10 por cento das vezes. Então, a entropia é:

$$H = - [- 0.9 \log_2 0.9 + 0.1 \log_2 0.1]$$

$$H = 0.9*(0.15) + 0.1*(3.3) = 0.47 \text{ bits.}$$

Agora, vamos supor que X_1 ocorre 80 por cento das vezes, enquanto X_2 ocorre 20 por cento das vezes. A entropia, então, se torna:

$$H = -[- 0.8 \log_2 0.8 + 0.2 \log_2 0.8]$$

$$H = 0.8*(0.32) + 0.2*(2.31) = 0.72 \text{ bits.}$$

Podemos continuar variando a freqüência de ocorrência de cada caractere em nosso conjunto de caracteres. A Tabela 4.12 lista a entropia resultante para o conjunto com dois caracteres, à medida que X_1 diminui a freqüência de ocorrência de 90 por cento para 50 por cento, enquanto X_2 aumenta de 10 para 50 por cento.

Tabela 4.12 - Entropia versos freqüência de ocorrência

Freqüência de ocorrência		
X1	X2	Entropia em bits
90	10	0.47
80	20	0.72
70	30	0.88
60	40	0.97
50	50	1.00

Ao examinar a coluna de entropia da Tabela 4.12, observe que quanto maior a disparidade na probabilidade de ocorrência, menor o número médio teórico de bits necessários para representar cada caractere. Isto indica que o método de compressão que reconhece as cadeias comuns, o que a nível de caractere constitui uma grande porcentagem, oferece o potencial de representação por um número baixo de bits. Vamos verificar esta observação, examinando o efeito da codificação de grupo de caracteres em vez de caracteres únicos.

Vamos supor que a freqüência de ocorrência de X_1 seja 80 por cento e X seja 20 por cento. Como calculado anteriormente, a entropia para este conjunto de caracteres é 0.72 bits. Usando a codificação Huffman ou a Shannon-Fano, resultaria na atribuição de 0 para um caractere e 1 para o outro caractere. Aqui, o comprimento médio é 1 bit por caractere, que é 28 por cento maior do que a entropia. Se combinarmos os caracteres em pares, obteremos quatro conjuntos de caracteres possíveis cujas probabilidades resultantes estão indicadas na Tabela 4.13.

Tabela 4.13 - Probabilidades por par de caracteres

Par de caractere	Probabilidade de ocorrência.
X_1X_1	0.8*0.8 = 0.64
X_1X_2	0.8*0.2 = 0.16
X_2X_1	0.2*0.8 = 0.16
X_2X_2	0.2*0.2 = 0.4

A figura 4.26 ilustra a construção de um código Huffman para os pares de caracteres listados na Tabela 4.13, com base em suas probabilidades de ocorrência. Observe que seu comprimento médio de bits é:

$$L = 1*0.64 + 2*0.16 + 3*0.16 + 3*0.04 = 1.56 \text{ bits.}$$

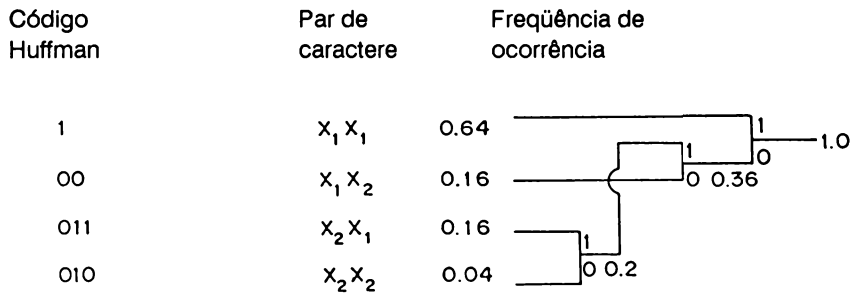


Figura 4.26 - A codificação Huffman dos pares de caracteres listados na Tabela 4.13.

Uma vez que todos os pares de caracteres representam dois caracteres, o comprimento médio de bits por par deve ser dividido por 2, para encontrar o comprimento médio de bits por caractere. Desta forma, resulta em uma média de 0.78 bits por caractere, quando os caracteres são agrupados em pares.

Agora, vamos prosseguir com mais uma etapa e codificar três caracteres por vez. A Tabela 4.14 lista as oito combinações distintas de três caracteres e sua probabilidade resultante de ocorrência.

Tabela 4.14 - Probabilidades com três caracteres

Grupo de caractere	Probabilidade de ocorrência
$X_1X_1X_1$	0.8*0.8*0.8 = 0.512
$X_1X_1X_2$	0.8*0.8*0.2 = 0.128
$X_1X_2X_1$	0.8*0.2*0.8 = 0.128
$X_1X_2X_2$	0.8*0.2*0.2 = 0.032
$X_2X_1X_1$	0.2*0.8*0.8 = 0.128
$X_2X_1X_2$	0.2*0.8*0.2 = 0.032
$X_2X_2X_1$	0.2*0.2*0.8 = 0.032
$X_2X_2X_2$	0.2*0.2*0.2 = 0.008

A Figura 4.27 ilustra a construção do código Huffman para grupos de três caracteres, baseados em sua probabilidade de ocorrência listada na Tabela 4.14. Agora, vamos examinar o comprimento médio de bits por grupo de três caracteres. O comprimento médio de bits é:

$$L = 1*0.512 + 3*0.128*3 + 5*0.032*3 + 5*0.008 = 2.184 \text{ bits.}$$

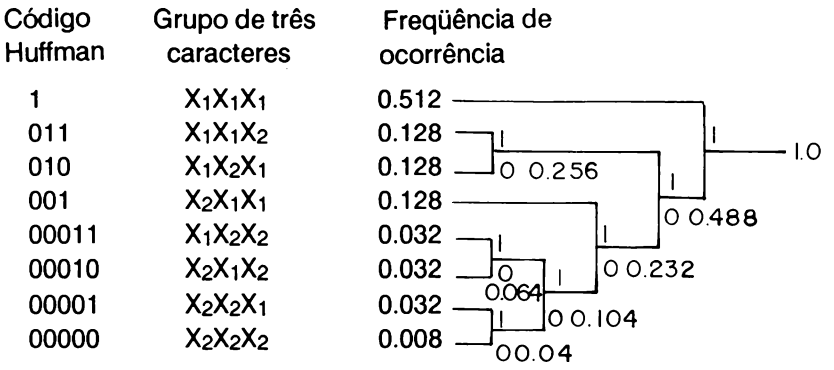


Figura 4.27 - Codificação Huffman de grupos de três caracteres listados na Tabela 4.14.

Já que o número médio de bits por grupo de três caracteres é 2.184, dividindo este número por 3 resulta no número médio de bits por caractere, que é 0.728. Dos cálculos anteriores, observamos que à medida que combinamos mais e mais caracteres, o comprimento médio de bits por caractere se aproxima da entropia. De fato, o comprimento médio se igualará à entropia se todas as probabilidades de ocorrência forem potências inversas de 2. Quando combinamos caracteres adicionais nos grupos de cadeias, suas probabilidades de ocorrência se aproximam das potências inversas de 2, que é a razão pela qual a técnica de compressão baseada na codificação de cadeias fica mais eficiente do que aquela baseada na codificação de caracteres. Esta foi uma das razões para a seleção da técnica Lempel-Ziv, para ser usada na compressão de dados recomendada pelo CCITT V.42 bis.

Algoritmo de codificação de cadeia Lempel-Ziv

Em 1975, Abraham Lempel e Jacob Ziv desenvolveram um algoritmo de codificação de cadeia que foi publicado no "IEEE Transactions on Information Theory" em maio de 1977. O algoritmo de codificação de cadeia, que ficou conhecido como técnica Lempel-Ziv, é composto de um conjunto de regras para se analisar as cadeias de símbolos de um alfabeto finito (A), como se fossem subcadeias, ou palavras, cujos comprimentos não ultrapassem um número inteiro predefinido L_s, e um esquema de codificação mapeia as subcadeias resultantes em palavras-código de comprimento fixo L_c. A relação entre o comprimento da palavra-código, se o comprimento da subcadeia for:

$$L_c = 1 + [\log (n - L_s)] + [\log L_s]$$

onde n é o comprimento de um buffer que armazena os últimos n símbolos. Na técnica de codificação Lempel-Ziv, uma cadeia de símbolos (S) é analisada como se fossem sucessivas palavras fonte $S = S_1S_2...S_n$, enquanto a palavra código C_i é atribuída a cada palavra Si. Para iniciar a sua codificação assume-se que a saída S da fonte seja precedida por uma cadeia Z, contendo n-Ls zeros que é armazenada no buffer como uma cadeia $B1 = Z_S (1,L_s)$. A seguir, os primeiros símbolos I_1 são deslocados para fora do buffer, enquanto os próximos símbolos L_1 entram no buffer para obter uma segunda cadeia B_2. Isto é seguido por um exame da cadeia para extensões reproduzíveis, para desenvolver uma palavra-código para representar a extensão. O leitor deve consultar o artigo de Lempel e Ziv (1977), de forma a obter informações detalhadas a respeito do algoritmo.

A recomendação V.42 bis

O trabalho de Lempel e Ziv foi posteriormente melhorado por funcionários da IBM, British Telecom, Bell Laboratories, Unisys e de outras empresas e serve de base para o método de compressão de dados CCITT V.42 bis, que é projetado para ser usado por modems. No V.42 bis uma cadeia que, tipicamente, varia de dois a quatro caracteres de comprimento, é trocada por uma palavra-código originada da construção de um dicionário que contém 512 ou mais cadeias de texto e palavras-chave associadas. O processo de compressão V.42 bis inclui a construção do dicionário e sua busca de entradas que combinem com os dados não compridos, resultando em palavras-código, sendo substituídas por cadeias de dados. No receptor, as palavras-código são analisadas e decodificadas com base na composição das cadeias do dicionário mantido no receptor.

O dicionário V.42 bis contém, inicialmente, o conjunto básico de caracteres e este se expande através da formação de novas cadeias, onde caracteres únicos são acrescentados às cadeias existentes. O dicionário é dinâmico, com novas cadeias sendo adicionadas e antigas cadeias sendo removidas, à medida que os dados são comprimidos.

Inicialmente, todas as cadeias com um caractere são predefinidas na tabela. Embora isto signifique que 256 cadeias com um caractere serão definidas. Com propósitos ilustrativos, vamos focalizar nossa atenção nas letras maiúsculas de A a Z. Então o dicionário no transmissor e no receptor, apareceria inicialmente como uma seqüência de nós-raiz como ilustrado na Figura 4.28.

A palavra-código para cada cadeia é inicialmente definida com o código ASCII para cada caractere. Isto é, A teria o número de cadeia 65, a B seria atribuído o número de cadeia 66 e assim em diante.

À medida em que os dados entram na codificador, estes são examinados, comparando-os com o conteúdo do dicionário. Se uma cadeia for encontrada e não estiver previamente armazenada no dicionário, esta é, então, acrescentada ao dicionário com um número de cadeia que funcionará como palavra-código da cadeia. Para ilustrar, vamos supor que o alfabeto composto das letras de A a Z deve ser transmitido repetitivamente. A primeira cadeia nova encontrada seria AB, onde A foi conhecido previamente quando o dicionário foi iniciado. Assim, esta nova cadeia seria armazenada no dicionário com a palavra-código 257, onde A é conhecido mas AB representa uma nova cadeia.

NUL	A	B	C		Z	DEL	caractere
	▪	▪	▪	.....	▪ ...		
	65	66	67		255	valor da cadeia	

Figura 4.28 - Estado inicial do dicionário.

Ao mesmo tempo que AB é colocado no dicionário do codificador, a palavra-código de B é transmitida ao receptor. Isto permite que o receptor construa um dicionário imagem de espelho que combina com o dicionário do codificador. Então, sempre que a cadeia AB for codificada, ela será transmitida como palavra-código 257, de maneira que o receptor irá reconhecê-la e decodificá-la corretamente. A próxima cadeia nova que seria identificada com o alfabeto sendo processada seria BC, sendo que o codificador encontraria o caractere C.

Agora, o dicionário é atualizado adicionando-se o C ao nó B. Ao mesmo tempo que o codificador transmite a palavra-código de C, que é 67, o receptor sabe que o terceiro caractere no fluxo de dados de entrada seria um C. Tanto o transmissor da codificação, quanto o receptor da decodificação incrementa o valor da palavra-chave de 1, indo assim para 258 e atribui este valor à cadeia BC no dicionário. Assim, a próxima vez que a cadeia BC for encontrada, esta seria codificada e transmitida como a palavra-código 258.

À medida em que os dados forem processados, o conjunto inicial de caracteres armazenado no dicionário se expandirá em uma estrutura baseada em árvore. Se considerarmos o conjunto inicial de caracteres como nós de rota, a expansão do dicionário resultará na geração de nós terminais. A Figura 4.29 ilustra os terminais formados a partir dos nós A e B, com base no exemplo anterior e ilustra também uma estrutura mais complexa de nó terminal originada do nó raiz T, que foi inserida com propósitos ilustrativos. A árvore na Figura 4.29, representa as cadeias A, AB, B, BC, ..., T, TH, THE e TO. Observe que cada nó terminal é dependente de um nó de nível superior (E depende de H) ou um nó raiz.

No processo de codificação, novas cadeias são sempre compostas de uma cadeia já definida no dicionário mais um caractere. Assim, a segunda seqüência do alfabeto, tida como dados originais, resultaria no acréscimo da cadeia ABC no dicionário, uma vez que a cadeia AB já era previamente conhecida. Quando ABC é acrescentada pela formação de uma nova ramificação do nó terminal B, a palavra-código da cadeia AB (257) é transmitida, seguida pela

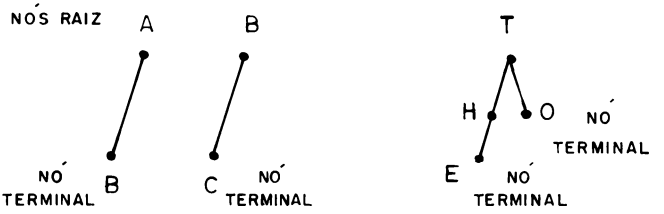


Figura 4.29 Evolução do dicionário.

palavra-código de C, que habilita o receptor a atualizar seu dicionário. Ao mesmo tempo, a próxima palavra-chave disponível á atribuída à cadeia ABC pelo transmissor e pelo receptor.

Como pôde ser observado na discussão anterior, a codificação de dados em cadeias poderia resultar em um número infinito de cadeias e palavras-código correspondentes. Para evitar que isto aconteça, a recomendação CCITT V.42 bis especifica uma variedade de comprimentos de cadeia, que é permitida e que pode ser negociada entre o transmissor e o receptor, bem como um valor predeterminado (default) de seis caracteres. O comprimento máximo permitido de cadeia pode ser variável de 6 a 250.

De forma semelhante, a recomendação V.42 bis especifica um valor predeterminado (default) de 512 palavras-código, que é o número mínimo de palavras-código ou de cadeias únicas que pode ocorrer de uma vez. Embora o valor máximo não seja especificado na recomendação atual, em cada expansão, você provavelmente vai querer aumentar o número de palavras-código em 256 e o comprimento de cadeia máximo em 3 caracteres, atingindo seu valor predeterminado (default) de 6.

Funções de dicionário

Na recomendação V.42 bis, cada dispositivo de compressão de dados mantém dois dicionários. Um é mantido pelo codificador para comprimir dados, enquanto o segundo dicionário é mantido pelo decodificador para descomprimir os dados recebidos. Além da comparação de cadeia e atualização de funções, a recomendação V.42 bis define um procedimento para eliminar cadeias raramente usadas. Esta função permite que o armazenamento no dicionário seja reusado e ocorra quando a última entrada disponível no dicionário tiver sido atribuída. Naquele momento, o contador (C1) que indica a palavra-código que seria atribuída à próxima entrada vazia do dicionário, é comparado com o número total de palavras-chave designado como N2. Se ultrapassar N2-1, então C1 é fixado com o número de índice da primeira entrada do dicionário, usada para armazenar uma cadeia designada como N5. Se o nó identificado pela palavra-código com o valor C1 estiver em uso e não for um nó terminal, o contador C1 é incrementado e seu valor é novamente comparado a N2-1. Se o nó identificado pela palavra-código com o valor C1 for um nó terminal, este é, então, separado de seu segmento, liberando uma entrada para reuso.

Embora os testes iniciais da compressão V.42 bis indiquem um nível de eficiência entre 10 e 20 por cento acima da MNP Classe 7, alcançar isto pode requerer uma investigação significativa dos parâmetros do modem, muito além do que muitos usuários podem normalmente considerar. Isto é devido ao comprimento máximo da cadeia e do número de palavras-código permitido, serem proporcionais à memória RAM contida nos modems V.42 bis. Embora todos os modems V.42 suportem as definições de parâmetros preestabelecidos, muitos modems se restringem, devido à memória RAM e podem não fornecer a eficiência de compressão adicional que você poderia esperar.

CAP. 05

CONSIDERAÇÕES DO SISTEMA E ANÁLISE DE DADOS

Para se obter uma rotina de compressão de dados ou uma série de algoritmos de compressão eficaz, é necessário um exame do sistema global de transmissão e uma análise e um entendimento da composição dos dados a serem transferidos. Os dois assuntos são abordados neste capítulo.

5.1 CONSIDERAÇÕES DO SISTEMA

Uma análise do sistema global de transmissão é um pré-requisito para maximizar a eficiência das comunicações. Esta análise não deve examinar somente o hardware e o software, mas também deve considerar o volume e o tipo de dado esperado no tráfego de comunicações. Um exemplo das considerações do sistema e seu efeito sobre as eficiências operacionais podem ser obtidos a partir de um exame do requisito de transmissão ilustrado na Figura 5.1.

Nesta figura, um terminal batch remoto é ligado através de uma interface à unidade de fita magnética e à linha de comunicações, funcionando como um sistema autônomo de transmissão fita-a-fita. Do ponto de vista global do sistema, muitos fatores afetam a eficiência operacional da transmissão. Para investigar estes fatores, podemos subdividir o sistema nos seus componentes lógicos - o subsistema de fita magnética, o terminal batch remoto e o elo de comunicação.

Com relação ao elo de comunicação, o protocolo empregado e a taxa de transferência de dados do modem regem a eficácia da utilização da linha.

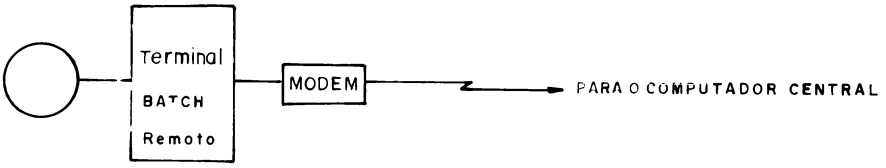


Figura 5.1 - Considerações do sistema de transmissão fita-a-fita. Para investigar a eficiência operacional da comunicação, as taxas de transferência de dados da unidade de fita magnética para o terminal, e do terminal para a linha de comunicação, devem ser determinadas antes de examinar a distribuição de freqüência dos dados e sua aplicabilidade a uma ou mais rotinas de compressão.

Se um modem de velocidade maior for empregado ou se um protocolo de transmissão mais eficiente for utilizado, pode-se esperar um aumento no número de caracteres transmitidos por unidade de tempo.

O próprio terminal batch remoto pode afetar a eficiência global da transmissão. Se examinarmos o terminal batch remoto como uma caixa preta, observaremos que este pode aceitar a entrada de dados vindos da unidade de fita magnética, a uma determinada taxa de transferência de dados. Em seguida, processa os dados e, então, transfere-os para a linha de comunicações em outra taxa de transferência. A taxa de transferência de dados na linha de comunicações dependerá do adaptador de canal que conecta o terminal ao modem, da taxa de transferência de dados do modem e do meio de transmissão empregado. Além disso, o processamento, executado pelo terminal, determinará se o dispositivo pode lançar os dados na saída, rápido o suficiente para usar as instalações de comunicações na taxa nominal de transferência de dados, ou em alguma taxa média abaixo do nível nominal. Para se gerar uma razão efetiva de transferência de informações (EITR) maior que a razão de transferência de dados da linha de comunicações, devem ser empregadas uma ou mais rotinas de compressão, de maneira que a razão global de transferência seja maior que o unitário. Se isto ocorrer, então a razão efetiva de transferência de informações ultrapassará a razão de transferência de dados. Os fatores chave para determinar a melhor rotina ou rotinas de compressão, a serem empregadas, estarão fundamentados na potência de processamento disponível para comprimir os dados e na análise de dados das informações a serem transmitidas.

Efeito da razão de compressão

Embora você possa querer utilizar uma rotina ou grupo de rotinas de compressão mais adequadas ao tráfego de dados esperado, em alguns casos, as limitações de processamento podem impedir o uso de uma técnica excelente. Sendo assim, o usuário deve determinar se a razão de compressão mais alta, oferecida por uma técnica em contraposição à outra, exigirá tempo de processamento adicional que poderia resultar em uma taxa global eficaz mais baixa de transferência de informações. Como exemplo, considere duas rotinas de compressão, uma resulta em uma razão de compressão de 1,52 , enquanto a outra resulta em uma razão de compressão de 1,61 . Se não examinássemos o efeito do processamento das duas rotinas, tenderíamos a selecionar a rotina com a maior razão de compressão. Vamos supor que uma investigação no software necessário para implementar a primeira rotina, mostre que este não tem efeito sobre o dispositivo que transfere dados à linha; conseqüentemente, o protocolo de transmissão gerenciará a taxa efetiva de dados na linha. Se a velocidade da linha for de 9600 bps e a razão de transferência de informações (ITR) da seqüência de controle de transmissão for 0,5 , a taxa de transferência de dados seria de 4800 bps, enquanto a EITR seria, aproximadamente, 1,52*0,5 ou 0,76 , resultando em uma taxa de transferência de informações em bits (TRIB) de 0,76*9600 ou 7296 bps.

Para a segunda rotina de compressão, vamos supor que sua natureza sofisticada exija uma potência maior de processamento computacional, afetando a taxa na qual os dados podem ser transferidos do terminal para a linha de comunicações. Vamos ir além e supor que a ITR é reduzida para 0,4 devido aos atrasos adicionais de processamento. Então, a razão de compressão de 1,61 desta rotina resulta em um TRIB de, aproximadamente, 0,4*1,61*9600 ou 6184 bps. Neste caso específico, o aumento da taxa de compressão resultou em um decréscimo na taxa global efetiva de transferência de dados efetiva.

Transferência de dados do dispositivo

A seguir, podemos considerar o efeito da taxa de transferência de dados das informações na unidade de fita magnética para o terminal batch remoto. Dois fatores regem a taxa de transferência de dados da unidade de fita magnética, já que a taxa na qual os dados podem ser transferidos entre a memória do terminal e a unidade de fita, é igual ao número de caracteres gravados por

polegada na fita, vezes a quantia de polegadas que passa, a cada segundo, pela cabeça de leitura da unidade. Além disso, o número de caracteres gravados por polegada de fita, depende do comprimento do registro e do fator de blocagem originalmente empregado para construir a fita. Do ponto de vista dos sistemas, se DTR1 for a taxa de transferência de dados da fita magnética ao terminal e DTR2 for a taxa de transferência de dados do terminal à linha de comunicações, então, para DTR1 < DTR2, não a nada a ganhar com a introdução de uma rotina ou várias rotinas de compressão, a não ser que seja implementada outra modificação no hardware ou no software para se tirar vantagem da compressão. Estas modificações poderiam estar presentas na geração da fita magnética, onde poderia se aumentar o fator de blocagem ou obter uma densidade de gravação maior, bem como obter unidades com cabeça de leitura/escrita mais rápidas que permitissem uma taxa de transferência de dados maior. Do ponto de vista das comunicações, você pode deixar que DTR1 < DTR2 continue a existir, mas ao incorporar a compressão para diminuir o DTR2, estaremos diminuindo, assim, os custos da linha e do modem, uma vez que reduzimos a taxa de transferência de dados da comunicação, mantendo a razão global efetiva de transferência de informações.

5.2 ANÁLISE DE DADOS

Depois que o hardware e o software foram analisados com relação ao sistema global, suponha que o emprego de uma ou mais técnicas de compressão indique que a eficiência operacional global das comunicações aumentará. Neste ponto, deve-se considerar a análise de dados.

A partir da análise de dados, você pode estruturar uma ou mais técnicas de compressão para atuar nos dados. O objetivo desta estruturação deve ser a obtenção do mais alto grau de compressão, proporcional à potência de processamento disponível, para comprimir o fluxo original de dados e a estrutura ou para voltar à sua forma original. Contudo, você deve observar que também tem de considerar o nível de habilidade do pessoal que possa ser designado para desenvolver o software de compressão de dados, o período de tempo que pode ser necessário para implementar algoritmos diferentes de compressão e a complexidade de programação dos dispositivos nos quais a compressão deve ser implementada.

Embora, há séculos, os tipógrafos tenham tirado vantagem do fato de algumas letras serem usadas mais freqüentemente do que

outras, foi Samuel Morse quem primeiro aplicou as propriedades da análise de freqüência nas comunicações. Quando Morse decidiu usar um sistema alfabético de sinais para o seu novo invento, o telégrafo, ele analisou a caixa de tipos do jornal da Filadélfia. Morse queria atribuir símbolos curtos de ponto-e-traço, usados pela transmissão de telégrafo às letras do alfabeto mais freqüentemente usadas, enquanto símbolos longos de ponto-e-traço seriam usados por caracteres menos freqüentemente usados. Ao contar os tipos de letra, ele descobriu 12.000 letras e, 9.000 letras t, 8000 letras a, o, m, i, s, 6.400 letras h e, etc. Com poucas exceções, Morse seguiu esta lista quando do desenvolvimento de seu código de teletipo, designando o símbolo mais curto, um ponto, à letra "e" que foi encontrada com mais freqüência. Outro símbolo curto, um traço, foi designado à letra "t".

Usando o Moderno Código Morse Internacional, que difere ligeiramente do código Morse original, a transmissão de uma mensagem em Inglês, composta de 100 letras, requer a transmissão de, aproximadamente, 940 unidades, onde a duração de um ponto equivale a uma unidade de ponto, um traço é igual a três unidades de ponto e o espaço entre pontos e traços de uma letra é igual a três unidades de ponto. Se, em vez de designar os símbolos da língua inglesa, os símbolos fossem designados aleatoriamente, a mesma mensagem necessitaria da transmissão de, aproximadamente, 1.160 unidades de ponto, um aumento de cerca de 23 por cento. Esta técnica de codificação representa a mais antiga aplicação da compressão estatística na transmissão de dados e permite quase um quarto de mensagens adicionais em uma linha telegráfica, durante o horário de pico, se comparado com atribuições de ponto-e-traço feitas no modo aleatório.

Considerações de Freqüência

Para as aplicações de compressão que serão empregadas em um tipo específico de dados, você deve determinar primeiro a freqüência dos caracteres individuais e os padrões de caracteres no fluxo original de dados a ser comprimido. Embora este pré-processamento de dados não seja possível em muitas situações, você pode estimar a freqüência prevista com base na categoria de informações esperadas no fluxo de dados. A tabela 5.1 contém uma tabela de freqüência desenvolvida a partir do exame de uma cadeia de 10.000 letras da língua inglesa normal pegas, aleatoriamente, de um livro. Se prevermos a transmissão ou o armazenamento do texto comum em língua inglesa, podemos usar a freqüência prevista de

ocorrência de cada letra, a partir da tabela para estabelecer, apropriadamente, um esquema de compressão estatística.

Um problema associado à análise de freqüência é que a natureza do armazenamento ou da transmissão de dados raramente permanece estática. Assim, a transmissão de um texto comum em língua inglesa em um determinado instante, pode ser precedido pela transmissão da compilação e da execução de um programa em FORTRAN. Estes tipos de dados são, via de regra, pesadamente constituídos de notação científica e de caracteres numéricos. Além disso, o texto em Inglês poderia ser seguido pela execução de um programa COBOL, geralmente orientado a caracteres alfanuméricos e a certos símbolos especiais como cifrão, vírgulas e pontos decimais. Para maximizar o desempenho da compressão, às vezes torna-se necessário reconhecer as mudanças nos padrões de freqüência de dados e empregar uma técnica diferente de compressão mais adequada à nova estrutura de dados. Em outros casos, um simples exame dos dados permitirá que o analista reconheça que a adição de uma simples técnica de compressão a outra técnica já sendo utilizada, acarretará em eficiências operacionais adicionais. De maneira contrária, se o processo de análise de dados identifica discrepâncias significativas, você vai provavelmente querer levar em consideração o desenvolvimento de uma técnica adaptativa de compressão de dados, semelhante àquela discutida no capítulo 4.

Tabela 5.1 - Tabela de Freqüência para a língua inglesa normal.

Letra	%	Ocorrência	Letra	%	Ocorrência
A	8.0	800	N	7.00	700
B	1.5	150	O	8.00	800
C	3.0	300	P	2.00	200
D	4.0	400	Q	0.25	25
E	13.0	1050	R	6.50	650
F	2.0	200	S	6.00	600
G	1.5	150	T	9.00	900
H	6.0	600	U	3.00	300
I	6.5	650	V	1.00	100
J	0.5	50	W	1.50	150
K	0.5	50	X	0.50	50
L	3.5	350	Y	2.00	200
M	3.0	300	Z	0.25	25

Já que a análise dos arquivos de dados pode ser um processo longo e enfadonho, um programa de computador chamado DATANALYSIS está listado no Apêndice A, para auxiliar os leitores a determinar a susceptibilidade de seus dados a um ou mais algoritmos de compressão. Através do uso do programa DATANALYSIS, você pode examinar a consistência dos arquivos típicos de dados e planejar a implementação de uma ou mais rotinas de compressão com uma expectativa predefinida de resultados. À medida que revisarmos a execução do programa e seu respectivo arquivo de dados de saída do sistema 'SYSOUT', seremos capazes de predeterminar o efeito de algoritmos diferentes de compressão.

O arquivo SYSOUT escolhido para a análise, reproduz um arquivo típico de dados de grandes sistemas de computação. Aqui, somente caracteres em letra maiúscula foram colocados no arquivo que representa um relatório de contabilidade, de arquivo de folha de pagamento ou de outro documento típico gerado por computador, ou, ainda um relatório de saída de dados do processador de texto.

Tabela 5.2

DATANALYSIS
RESUMO DO RELATÓRIO DE EXECUÇÃO
ARQUIVO ANALISADO: SYSOUT

SEGMENTO	*DESCRIÇÃO*
I	FREQÜÊNCIA DE OCORRÊNCIA PADRÃO DO SISTEMA
II	FREQÜÊNCIA DE OCORRÊNCIA CLASSIFICADA
III	ANÁLISE DA CADEIA DE CARACTERES REPETIDOS
IV	SUSCEPTIBILIDADE À CODIFICAÇÃO DE COMPRIMENTO DE FILEIRA
V	SUSCEPTIBILIDADE À CODIFICAÇÃO DE MEIO-BYTE
VI	ANÁLISE DE COMPRESSÃO DE CARACTERES EM PARES
VII	ANÁLISE DE ENTROPIA

Análise do DATANALYSIS

A Tabela 5.2 contém o cabeçalho ou folha de resumo do programa DATANALYSIS. Como indicado, o programa é dividido em sete segmentos, indo da tabulação da freqüência de ocorrência padrão do sistema, até a análise da entropia. A Tabela 5.3 lista o resultado do segmento I do programa. Este segmento lista os caracteres ASCII que ocorreram, a contagem de cada caractere encontrado no arquivo analisado, bem como sua freqüência de ocorrência em porcentagem. Deve ser observado, a partir da Tabela 5.3, que dos 99.132 caracteres analisados no arquivo SYSOUT, 73.509 eram espaços, o que representa 74,15 por cento do arquivo.

O segmento II do programa DATANALYSIS está listado na Tabela 5.4. Nesta tabela, os caracteres são imprimidos pela ordem da freqüência de ocorrência.

A Tabela 5.5 apresenta os resultados da análise da cadeia de caracteres repetidos, obtidos através da execução do Segment III do programa DATANALYSIS no arquivo SYSOUT do exemplo. Aqui, é tabulado o número de ocorrências de cadeias numéricas, incluindo: vírgulas, pontos decimais, cifrões e asteriscos, cadeias de espaços ou cadeias contendo qualquer caractere repetido. A partir desta tabela, fica aparente que os espaços são os caracteres que mais se repetem no arquivo analisado.

A Tabela 5.6 mostra os resultados do segmento IV do programa DATANALYSIS. Neste segmento de programa, o número real de caracteres, que podem ser salvos pelo emprego da codificação de comprimento de fileira, é computado e tabulado.

Tabela 5.3

SEGMENTO FREQÜÊNCIA DE OCORRÊNCIA PADRÃO DO SISTEMA DATANALYSIS
I TABELA DE CARACTERES ENCONTRADOS NO ARQUIVO SYSOUT

CARAC.	CONT.	%	CARAC.	CONT.	%	CARAC.	CONT.	%	CARAC.	CONT	%
SH	0.	0.	EP	13.	0.01	A	713.	0.72	a	0.	0.
EX	0.	0.	"	201.	0.20	B	252.	0.25	b	0.	0.
SX	0.	0.	#	2.	0.00	C	427.	0.43	c	0.	0.
ET	0.	0.	$	2.	0.00	D	290.	0.29	d	0.	0.
EQ	0.	0.	%	7.	0.01	E	928.	0.94	e	0.	0.
AK	0.	0.	&	105.	0.11	F	279.	0.28	f	0.	0.
BL	0.	0.	'	1.	0.00	G	192.	0.19	g	0.	0.
BS	0.	0.	(	542.	0.55	H	1095.	1.10	h	0.	0.
HT	0.	0.	)	542.	0.55	I	1223.	1.23	i	0.	0.
LF	0.	0.	*	1055.	1.06	J	97.	0.10	j	0.	0.
VT	0.	0.	+	74.	0.07	K	121.	0.12	k	0.	0.
FF	0.	0.	,	1327.	1.34	L	355.	0.36	l	0.	0.
CL	0.	0.	-	135.	0.14	M	332.	0.33	m	0.	0.
SO	0.	0.	.	135.	0.14	N	644.	0.65	m	0.	0.
SI	0.	0.	/	60.	0.06	O	694.	0.70	o	0.	0.
DE	0.	0.	0	1066.	1.08	P	432.	0.44	p	0.	0.
D1	0.	0.	1	905.	0.91	Q	98.	0.10	q	0.	0.
D2	0.	0.	2	1427.	1.44	R	813.	0.82	r	0.	0.
D3	0.	0.	3	405.	0.41	S	573.	0.58	s	0.	0.
D4	0.	0.	4	354.	0.36	T	1022.	1.03	t	0.	0.
NK	0.	0.	5	353.	0.36	U	511.	0.52	u	0.	0.
SY	0.	0.	6	248.	0.25	V	95.	0.10	v	0.	0.
EB	0.	0.	7	281.	0.28	W	197.	0.20	w	0.	0.
CN	0.	0.	8	312.	0.31	X	196.	0.20	x	0.	0.
EM	0.	0.	9	242.	0.24	Y	190.	0.19	y	0.	0.
SB	0.	0.	:	21.	0.02	Z	62.	0.06	z	0.	0.
EC	0.	0.	;	4.	0.00	[	1.	0.00	{	0.	0.
FS	0.	0.	<	1.	0.00	/	1.	0.00	|	0.	0.
GS	0.	0.	=	152.	0.15	]	1.	0.00	}	0.	0.
RS	0.	0.	>	1.	0.00	^	1.	0.00	~	0.	0.
US	0.	0.	QM	5.	0.01	_	53.	0.05	DL	0.	0.
SP	73509.	74.15	@	2.	0.00	@	0.	0.	**	99132.	0.
											100.00

** indica os caracteres totais no arquivo

Tabela 5.4

SEGMENTO FREQÜÊNCIA DE OCORRÊNCIA PADRÃO DO SISTEMA DATANALYSIS
I I TABELA DE CARACTERES ENCONTRADOS NO ARQUIVO SYSOUT

CARAC.	CONT.	%	CARAC.	CONT.	%	CARAC.	CONT.	%	CARAC	.CONT	%
SP	73509.	74.15	"	201.	0.20	SX	0.	0.	a	0.	0.
2	1427.	1.33	W	197.	0.20	ET	0.	0.	b	0.	0.
,	1327.	1.34	X	196.	0.20	EQ	0.	0.	c	0.	0.
I	1223.	1.23	G	192.	0.19	AK	0.	0.	d	0.	0.
H	1095.	1.10	T	190.	0.19	BL	0.	0.	e	0.	0.
0	1066.	1.08	=	152.	0.15	BS	0.	0.	f	0.	0.
@	1055.	1.06	-	135.	0.14	HQ	0.	0.	g	0.	0.
T	1022.	1.03		135.	0.14	LF	0.	0.	h	0.	0.
E	928.	0.94	K	121.	0.12	VT	0.	0.	i	0.	0.
1	905.	0.91	&	105.	0.11	FF	0.	0.	j	0.	0.
R	813.	0.82	C	98.	0.10	CR	0.	0.	k	0.	0.
A	713.	0.72	J	97.	0.10	SO	0.	0.	l	0.	0.
O	694.	0.70	v	95.	0.10	SI	0.	0.	m	0.	0.
N	644.	0.65	+	74.	0.07	DE	0.	0.	n	0.	0.
S	573.	0.58	Z	62.	0.06	D1	0.	0.	o	0.	0.
(	542.	0.55	/	60.	0.06	D2	0.	0.	p	0.	0.
)	542.	0.55	__	53.	0.05	D3	0.	0.	q	0.	0.
U	511.	0.52	:	21.	0.02	D4	0.	0.	r	0.	0.
P	432.	0.44	EP	13.	0.01	NK	0.	0.	s	0.	0.
C	427.	0.43	%	7.	0.01	SY	0.	0.	t	0.	0.
3	405.	0.41	QM	5.	0.01	EB	0.	0.	u	0.	0.
L	355.	0.36	;	4.	0.00	CN	0.	0.	v	0.	0.
4	354.	0.36	#	2.	0.00	EM	0.	0.	w	0.	0.
5	353.	0.36	$	2.	0.00	SB	0.	0.	x	0.	0.
M	332.	0.33	@	2.	0.00	EC	0.	0.	y	0.	0.
8	312.	0.31	>	1.	0.00	FS	0.	0.	z	0.	0.
D	990.	0.29	<	1.	0.00	GS	0.	0.	{	0.	0.
7	281.	0.28	[	1.	0.00	RS	0.	0.	|	0.	0.
F	279.	0.28	/	1.	0.00	US	0.	0.	}	0.	0.
B	252.	0.25	]	1.	0.00	SH	0.	0.	~	0.	0.
6	248.	0.25	^	1.	0.00	EX	0.	0.	DL	0.	0.
9	242.	0.24	'	1.	0.00	@	0.	0.	**	99132.	100.0

** indica os caracteres totais no arquivo

Tabela 5.5

SEGMENTO III DATANALYSIS
ANÁLISE DA CADEIA DE CARACTERES REPETIDOS NO ARQUIVO SYSOUT

COMPRIMENTO DA CADEIA	NÚMERO DE OCORRÊNCIAS			
	NUMÉRICOS	ESPAÇOS	OUTROS	TOTAL
3	12	93	36	141
4	0	6	7	13
5	1	22	755	778
6	0	160	1	161
7	0	472	5	475
9	0	13	0	13
12	0	2	0	2
14	0	498	0	498
15	0	90	0	90
16	0	12	0	12
22	0	0	3	3
23	0	0	5	5
24	0	0	4	4
25	0	0	2	2
26	0	0	1	1
27	0	2	1	3
28	0	2	3	5
29	0	2	0	2
30	0	7	0	7
31	0	6	0	6
32	0	5	2	7
33	0	3	0	3
34	0	5	1	6
35	0	1	1	2
36	0	2	0	2
37	0	1	0	1
38	0	6	2	8
39	0	2	0	2
40	0	1	0	1
41	0	1	0	1
43	0	1	0	1
44	0	3	0	3
45	0	2	0	2
46	0	3	0	3
47	0	4	0	4
48	0	4	0	4
49	0	1	0	1
50	0	2	0	2
51	0	3	0	3
52	0	82	0	82

Tabela 5.5 (continuação)

SEGMENTO III DATANALYSIS
ANÁLISE DA CADEIA DE CARACTERES REPETIDOS NO ARQUIVO SYSOUT

COMPRIMENTO DA CADEIA	NÚMERO DE OCORRÊNCIAS			
	NUMÉRICOS	ESPAÇOS	OUTROS	TOTAL
53	0	3	0	3
54	0	4	0	4
55	0	3	0	3
56	0	4	0	4
57	0	3	0	3
58	0	3	0	3
60	0	4	0	4
61	0	6	0	6
62	0	1	0	1
63	0	11	0	11
64	0	1	0	1
65	0	5	0	5
66	0	1	0	1
67	0	8	0	8
68	0	4	0	4
69	0	4	0	4
71	0	7	0	7
72	0	4	0	4
73	0	16	0	16
74	0	3	0	3
75	0	4	0	4
76	0	24	0	24
77	0	9	0	9
78	0	10	0	10
79	0	19	0	19
80	0	7	0	7
81	0	14	0	14
82	0	6	0	6
83	0	31	0	31
84	0	17	0	17
85	0	52	0	52
86	0	27	0	27
87	0	12	0	12
88	0	21	0	21
89	0	12	0	12
90	0	12	0	12
91	0	32	0	32
92	0	4	0	4
93	0	21	0	21

Tabela 5.5 (continuação)

SEGMENTO III DATANALYSIS
ANÁLISE DA CADEIA DE CARACTERES REPETIDOS NO ARQUIVO SYSOUT

COMPRIMENTO DA CADEIA	NÚMERO DE OCORRÊNCIAS			
	NUMÉRICOS	ESPAÇOS	OUTROS	TOTAL
94	0	8	0	8
95	0	10	0	10
96	0	15	0	15
97	0	6	0	6
98	0	10	0	10
99	0	23	0	23
127	0	82	0	82

RESUMO DA ANÁLISE:
REGISTROS PROCESSADOS - 751
TOTAL DE REPETIÇÕES - 76.750
TOTAL DE CARACTERES NO ARQUIVO - 99.132
PORCENTAGEM DE REPETIÇÕES - 77.42

Tabela 5.6

SEGMENTO IV DATANALYSIS

SUSCEPTIBILIDADE DA COMPRESSÃO POR CODIFICAÇÃO DE COMPRIMENTO DE FILEIRA PARA AS CADEIAS DE CARACTERES REPETIDOS NO ARQUIVO SYSOUT

COMPRIMENTO DA CADEIA	CARACT. REPETIDOS				CAR. SALVOS PELA COMPRESSÃO			
	NUMÉRICOS	ESPAÇOS	OUTROS	TOTAL	NUMÉRICOS	ESPAÇOS	OUTROS	TOTAL
3	36	279	108	423	0	0	0	0
4	0	24	28	52	0	6	7	13
5	5	110	3775	3890	2	44	1510	1556
6	0	960	6	966	0	480	3	483
7	0	3304	35	3339	0	1888	20	1908
9	0	117	0	117	0	78	0	78
12	0	24	0	24	0	18	0	18
14	0	6972	0	6972	0	5478	0	5478
15	0	1350	0	1350	0	1080	0	1080
16	0	192	0	192	0	156	0	156
22	0	0	66	66	0	0	57	57
23	0	0	115	115	0	0	100	100
24	0	0	96	96	0	0	84	84
25	0	0	50	50	0	0	44	44
26	0	0	26	26	0	0	23	23
27	0	54	27	81	0	48	24	72
28	0	56	84	140	0	50	75	125
29	0	58	0	58	0	52	0	52
30	0	210	0	210	0	189	0	189
31	0	186	0	186	0	168	0	168
32	0	160	64	224	0	145	58	203
33	0	99	0	99	0	90	0	90
34	0	170	34	204	0	155	31	186
35	0	35	35	70	0	32	32	64
36	0	72	0	72	0	66	0	66
37	0	37	0	37	0	34	0	34
38	0	228	76	304	0	210	70	280
39	0	78	0	78	0	72	0	72
40	0	40	0	40	0	37	0	37
41	0	41	0	41	0	38	0	38
43	0	43	0	43	0	40	0	40
44	0	132	0	132	0	123	0	123
45	0	90	0	90	0	84	0	84
46	0	138	0	138	0	129	0	129
47	0	188	0	188	0	176	0	176
48	0	192	0	192	0	180	0	180
49	0	49	0	49	0	46	0	46

Tabela 5.6 (continuação)

SEGMENTO IV — DATANALYSIS

SUSCEPTIBILIDADE DA COMPRESSÃO POR CODIFICAÇÃO DE COMPRIMENTO DE FILEIRA PARA AS CADEIAS DE CARACTERES REPETIDOS NO ARQUIVO SYSOUT

COMPRIMENTO DA CADEIA	CARACT. REPETIDOS				CAR. SALVOS PELA COMPRESSÃO			
	NUMÉRICOS	ESPAÇOS	OUTROS	TOTAL	NUMÉRICOS	ESPAÇOS	OUTROS	TOTAL
50	0	100	0	100	0	94	0	94
51	0	153	0	153	0	144	0	144
52	0	4264	0	4264	0	4018	0	4018
53	0	159	0	159	0	150	0	150
54	0	216	0	216	0	204	0	204
55	0	165	0	165	0	156	0	156
56	0	224	0	224	0	212	0	212
57	0	171	0	171	0	162	0	162
58	0	174	0	174	0	165	0	165
60	0	240	0	240	0	228	0	228
61	0	366	0	366	0	348	0	348
62	0	62	0	62	0	59	0	59
63	0	693	0	693	0	660	0	660
64	0	64	0	64	0	61	0	61
65	0	325	0	325	0	310	0	310
66	0	66	0	66	0	63	0	63
67	0	536	0	536	0	512	0	512
68	0	272	0	272	0	260	0	260
69	0	276	0	276	0	264	0	264
71	0	497	0	497	0	476	0	476
72	0	288	0	288	0	276	0	276
73	0	1168	0	1168	0	1120	0	1120
74	0	222	0	222	0	213	0	213
75	0	300	0	300	0	288	0	288
76	0	1824	0	1824	0	1752	0	1752
77	0	693	0	693	0	666	0	666
78	0	780	0	780	0	750	0	750
79	0	1501	0	1501	0	1444	0	1444
80	0	560	0	560	0	539	0	539
81	0	1134	0	1134	0	1092	0	1092
82	0	492	0	492	0	474	0	474
83	0	2573	0	2573	0	2480	0	2480
84	0	1428	0	1428	0	1377	0	1377
85	0	4420	0	4420	0	4264	0	4264
86	0	2322	0	2322	0	2241	0	2241
87	0	1044	0	1044	0	1008	0	1008
88	0	1848	0	1848	0	1785	0	1785

Tabela 5.6 (continuação)

SEGMENTO IV DATANALYSIS

SUSCEPTIBILIDADE DA COMPRESSÃO POR CODIFICAÇÃO DE COMPRIMENTO DE FILEIRA PARA AS CADEIAS DE CARACTERES REPETIDOS NO ARQUIVO SYSOUT

COMPRIMENTO DA CADEIA	CARACT. REPETIDOS				CAR. SALVOS PELA COMPRESSÃO			
	NUMÉRICOS	ESPAÇOS	OUTROS	TOTAL	NUMÉRICOS	ESPAÇOS	OUTROS	TOTAL
89	0	1068	0	1068	0	1032	0	1032
90	0	1080	0	1080	0	1044	0	1044
91	0	2912	0	2912	0	2816	0	2816
92	0	368	0	368	0	356	0	356
93	0	1953	0	1953	0	1890	0	1890
94	0	752	0	752	0	728	0	728
95	0	950	0	950	0	920	0	920
96	0	1440	0	1440	0	1395	0	1395
97	0	582	0	582	0	564	0	564
98	0	980	0	980	0	950	0	950
99	0	2277	0	2277	0	2208	0	2208
127	0	10414	0	10414	0	10168	0	10168
....	41	72084	4625	76750	2	65778	2138	67918

Tabela 5.6 (continuação)

INFORMAÇÕES DE RESUMO: TOTAL DE CARACTERES NO ARQUIVO DE DADOS: 99.132
PORCENTAGEM DOS CARACTERES REPETIDOS EM SEQÜÊNCIA:

NUMÉRICOS:	0
ESPAÇOS EM BRANCO:	72
NÃO NUMÉRICO / NÃO ESPAÇO EM BRANCO:	4
TOTAL DE REPETIDOS:	77

REDUÇÃO POTENCIAL DA COMPRESSÃO (%):

NUMÉRICOS:	0
ESPAÇOS EM BRANCO:	66
NÃO NUMÉRICO / NÃO ESPAÇO EM BRANCO:	2
TOTAL:	68

Observe que a redução potencial da compressão foi computada como sendo de 68 por cento, usando-se a técnica de codificação de comprimento de fileira.

A Tabela 5.7 mostra os resultados do segmento V do programa DATANALYSIS. Neste segmento de programa, a susceptibilidade do arquivo de dados à compactação de meio byte é computada e o número de caracteres que poderiam ser salvos, com base na ocorrência de comprimentos diferentes de cadeia, contendo numéricos e caracteres predefinidos é listado. Embora somente uma redução de arquivo de 0,15 por cento seja alcançada com a compactação de meio byte, ela é cumulativa e pode resultar em uma redução de arquivo de 68,15 por cento, quanto esta técnica é acrescentada à técnica de codificação de comprimento de fileira.

Tabela 5.7

SEGMENTO V DATANALYSIS

SUSCEPTIBILIDADE À COMPRESSÃO DA CODIFICAÇÃO DE MEIO BYTE PARA OS NUMÉRICOS SEQÜENCIAIS (NÃO REPETIDOS) NO ARQUIVO SYSOUT

COMPRIMENTO DE CADEIA	NÚMERO DE OCORRÊNCIAS	TOTAL DE CARACTERES	CARACTERES COMPRIMIDOS	CARACTERES SALVOS
4	655	2620	2620	0
5	43	215	215	0
6	17	102	85	17
8	34	272	204	68
11	11	121	88	33
14	6	84	54	30

RESUMO DA ANÁLISE:
TOTAL DE CARACTERES NO ARQUIVO - 99.132
TOTAL DE CARACTERES SUSCEPTÍVEIS À COMPRESSÃO - 3.414
TOTAL DE CARACTERES SALVOS PELA COMPRESSÃO - 148
PORCENTAGEM DE REDUÇÃO DA COMPRESSÃO DO ARQUIVO - 0,15

A Tabela 5.8 contém uma análise de pares ou diatômica do arquivo de dados, resultante do Segmento VI do programa DATANALYSIS. Embora 4.528 combinações de pares foram encontradas, este número baseou-se em um total de 138 pares diferentes de caracteres. Para economizar 4.528 caracteres, através da transmissão de um caractere especial no lugar de cada par encontrado, seria necessário o emprego de 138 caracteres especiais. Já que esta quantidade está além do número de caracteres não usados e disponíveis, na maioria dos conjuntos de caracteres, você deve selecionar um subconjunto para codificá-los. Se os 24 pares de caracteres encontrados com maior freqüência fossem codificados, isto resultaria

Tabela 5.8

SEGMENTO VI DATANALYSIS
ANÁLISE DE COMPRESSÃO DE PARES DE CARACTERES

PAR/CONT		PAR/CONT		PAR/CONT		PAR/CONT		PAR/CONT		PAR/CONT	
_I	156	TE	149	RI	110	UT	108	PU	96	_W	85
E_	80	TI	75	IN	75	_C	72	OR	70	_F	68
RE	68	N_	66	AT	66	MA	60	ER	59	SE	56
_D	55	AL	55	IR	53	FO	53	T_	52	O_	52
ON	50	L_	48	R_	46	D_	46	HA	46	S_	45
EN	45	SU	45	CO	43	AR	42	IS	40	_T	39
NT	39	LA	39	SI	38	_S	37	TH	37	AN	35
IT	34	_E	33	PA	33	IO	33	_R	33	HE	31
RA	30	RM	30	IA	30	TO	29	WE	29	ME	28
UB	28	LE	27	EQ	26	IM	26	H_	25	OU	25
AC	25	GE	24	CH	24	NE	24	_O	24	NU	24
LI	24	CT	23	OM	23	PR	23	ND	22	IL	22
TA	22	_P	22	DA	22	NA	21	_A	21	AI	21
NG	21	RO	20	MP	20	HS	20	AB	20	NO	19
CE	19	LY	19	RS	19	HC	19	YS	19	BE	18
RD	18	HG	18	IX	18	HI	18	HO	18	FI	18
IP	18	HU	18	ST	18	HQ	17	_B	17	DI	17
HW	17	HY	17	HK	17	HM	17	GO	17	_N	16
PI	16	CU	16	HD	16	G_	16	UE	16	UN	16
AP	16	EG	16	WO	16	CS	15	MI	15	ED	15
HN	15	EL	15	HH	14	PE	14	RR	14	TR	14
TS	14	HB	14	X_	14	HV	14	UR	14	_G	14
ES	14	OS	14	EP	14	HR	13	HF	13	HT	13

TOTAL DE COMBINAÇÕES ENCONTRADAS: 4.528

em uma economia de 1.839 caracteres, proporcionando uma redução adicional de dados de 1,85 por cento. Assim, quando combinado com a codificação de comprimento de fileira e de meio byte, pode-se esperar uma redução global de dados de 70 por cento. Finalmente, a Tabela 5.9 mostra os resultados do segmento VII do programa. Neste segmento de programa, a entropia do arquivo de dados é computada para indicar sua compressão estatística teórica. Muitas seqüências de transmissão operam com limites de 8*N bits, exigindo o preenchimento de bits fictícios no fim de uma palavra, para que se opere a transmissão de blocos de dados

codificados estatisticamente. Isto resultará na redução real de dados pela compressão estatística se aproximando, mas nunca alcançando a redução de dados calculada teoricamente. O grau de variação será uma função do tamanho do bloco de transmissão e da soma do número de bits por caractere, para cada caractere no bloco codificado. É interessante observar que a redução de dados teórica de 75,77 por cento é praticamente alcançada pelo uso de três técnicas de compressão de dados previamente abordadas e facilmente combinadas. Como mostrado, a execução deste programa serve como um valioso guia. Aqui, foram fornecidas informações indicando que a combinação de três técnicas de comprimento fixo, facilmente implementadas, alcançaram a redução de dados obtida a partir de uma técnica de comprimento variável que requer muita potência de processamento e cujo uso poderia resultar na expansão de dados sob certas condições.

Tabela 5.9

SEGMENTO VII DATANALYSIS
ANÁLISE DE ENTROPIA PARA O ARQUIVO SYSOUT

ENTROPIA: 1.94 BITS/CARACTERE
ARQUIVO CONTÉM:
99.132 CARACTERES
793.056 BITS
COMPRESSÃO ESTATÍSTICA TEÓRICA: 75,77%
ARQUIVO SOLICITARIA:
24.016 CARACTERES
192.130 BITS

CAP. 06

CONSIDERAÇÕES DE LIGAÇÃO DO SOFTWARE

Muitos fatores devem ser levados em conta ao se desenvolver softwares para executar a compressão. Estes fatores incluem o tipo de dispositivo sobre o qual o software operará, o método usado para ligar o software de compressão a outro software, a taxa de transferência dos dados comprimidos, tanto internamente de ou para unidades periféricas de armazenamento, quanto de e para um meio de transmissão e o número de instruções necessárias para se codificar o software apropriado. Neste capítulo, examinaremos as considerações para ligação de softwares pertinentes à aplicação de compressão de dados em sistemas de teleprocessamento on-line.

6.1 POSICIONAMENTO DA ROTINA DE COMPRESSÃO

O software de compressão será escrito, em geral, como uma rotina modular, cuja relação com a estrutura global do software do sistema dependerá da aplicação definitiva - para o armazenamento ou transmissão de dados comprimidos.

A estrutura do software para compressão de transmissão está ilustrada no formulário geral da Figura 6.1. Aqui, a rotina de compressão/descompressão estará no mesmo nível das outras rotinas de comunicações, tais como: reconhecimento baud automático de velocidade e conversão de código, com o tratador de transmissão, agindo como um controlador global da rotina, ou policial de tráfego, semelhante ao modo que o sistema operacional é o controlador de

programas aplicativos. Quando ocorre uma interrupção na linha, a rotina automática de detecção de velocidade automática pode ser invocada pelo tratador de transmissão, para determinar a velocidade de operação da transmissão de chegada. A seguir, supondo que a taxa de transferência fosse detectada e o posicionamento de armazenamento temporário fosse apropriado, a rotina de seqüência de controle da transmissão poderia ser invocada.

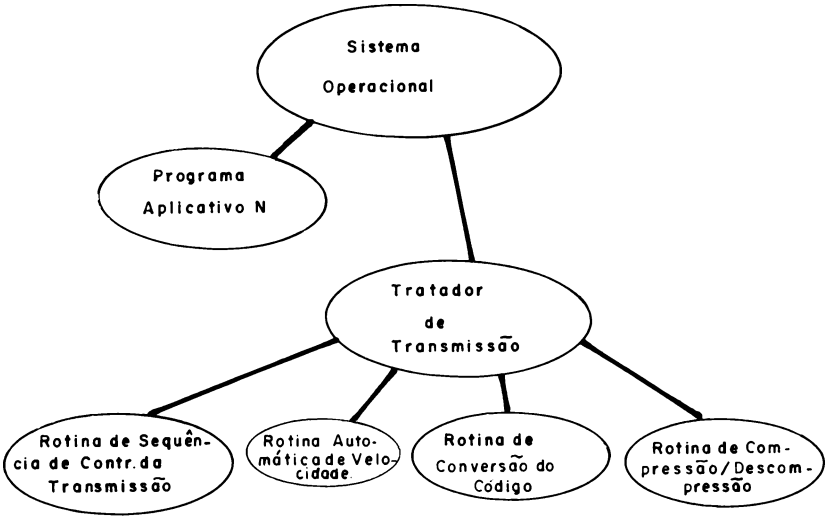


Figura 6.1 - Estrutura do software para compressão de transmissão.

Se a transmissão segue o protocolo BISYNC, os blocos de dados têm seu caractere de verificação de bloco recomputado no local de recepção e comparado com o caractere de verificação de bloco transmitido. Quando os caracteres de verificação transmitidos e os computados são equivalentes, uma aceitação positiva é enviada à estação de transmissão. Como parte da seqüência de controle da transmissão, os caracteres especiais de controle juntam-se ao bloco de dados, mas se for um propósito da transmissão, eles podem ser retirados, resultando em um bloco de dados comprimidos. Neste ponto, a rotina de descompressão pode ser invocada para trazer os dados de volta no formato original. Embora a descrição anterior faça com que o posicionamento da rotina de compressão pareça ser um problema simples, um estudo cuidadoso da rotina deve preceder seu posicionamento no software do sistema.

Considerações de Software

A rotina de compressão, como todas as rotinas de software, necessita tanto de memória do processador, quanto de tempo de execução do programa. A quantidade de memória e o tempo de execução do programa solicitados pela rotina, determinarão se a compressão pode ou não ser realizada, em uma máquina particular, juntamente com outras funções. Para a análise do requisito do buffer, vamos supor que estamos considerando a codificação de comprimento de fileira. Se o protocolo de transmissão for o BISYNC, devemos primeiro olhar o tamanho do bloco de dados existente tratado pelo protocolo. Se supusermos que nosso bloco tenha 240 caracteres, os dados comprimidos também ocorreram em blocos de 240 caracteres, a não ser que mudemos o tamanho do bloco de transmissão. Para determinar a área de buffer necessária para a descompressão, vamos examinar o pior caso para descompressão, como ilustrado na Figura 6.2. Aqui, o buffer do bloco de dados recebido é primeiro enchido com 240 caracteres do bloco de dados transmitidos.

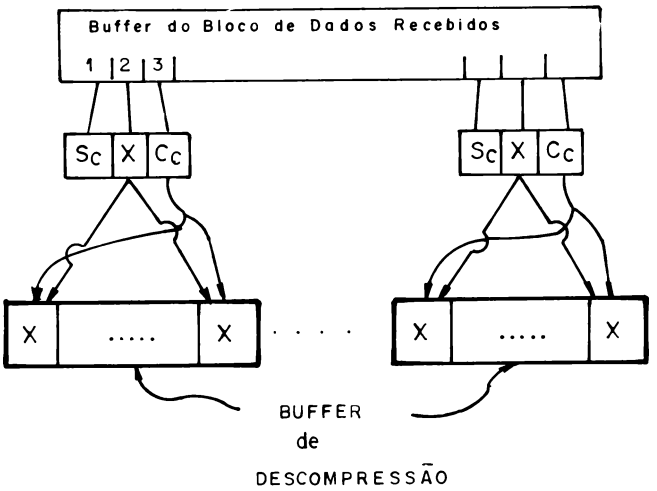


Figura 6.2 - Requisitos de memória. Quando os dados comprimidos são transmitidos, as áreas de buffer devem estar disponíveis para recuperar os dados na sua forma original. Sc = caractere especial de indicação de compressão, Cc = contagem de caracteres comprimidos.

Se a codificação de comprimento de fileira for completamente eficaz, o bloco de dados será preenchido com 80 seqüências de três caracteres, compostas de um caractere especial indicando a compressão, seguido do caractere repetido da cadeia que foi comprimida e a contagem de caractere que indica o número de caracteres repetidos. Se cada contagem de caracteres indica que a cadeia original era composta de 64 caracteres semelhantes, torna-se necessário um buffer de compressão de 5120 caracteres (64 x 80). Embora isto possa parecer um tamanho exagerado de buffer, já que é remota a probabilidade de se encontrar tal grau de compressão, você deve considerar a alocação deste espaço, a não ser que você execute a compressão em estágios cíclicos nos dados recebidos. De outra forma, o encurtamento do buffer de descompressão resultará em tempo adicional de transferência de dados, como veremos a seguir.

Se o buffer de descompressão for reduzido para o tamanho de 2.560 caracteres, enquanto todas as outras condições permanecerem as mesmas, para satisfazer as condições do pior caso, dois ciclos serão necessários para completar o retorno dos dados comprimidos ao formato original. No primeiro ciclo, metade do buffer do bloco de dados recebido é processado, o que resultará no preenchimento do buffer de descompressão com 2.560 caracteres. Neste ponto, uma transferência de I/O ou uma transferência por ciclo de DMA deve ser iniciada, para esvaziar o buffer e transferir seu conteúdo para uma unidade periférica de armazenamento. Enquanto a transferência é estabelecida e executada, o processo de descompressão não pode continuar; conseqüentemente, a redução no tamanho do buffer resulta em dois estados de espera (wait states), enquanto o buffer menor é esvaziado. Um modo de aliviar o processo de espera é reduzir, ainda mais, o tamanho do buffer de descompressão e, daí, empregar dois destes buffers. Desta maneira, a duplicação do buffer pode ocorrer e, assim, minimizar o processo de espera, como ilustrado na Figura 6.3.

Quando o buffer duplo é empregado, o buffer do bloco de dados recebido é processado em segmentos de comprimento variável. O fim do primeiro segmento ocorre quando o primeiro buffer de descompressão é preenchido, enquanto o fim do segundo segmento ocorre quando o segundo buffer de descompressão é preenchido.

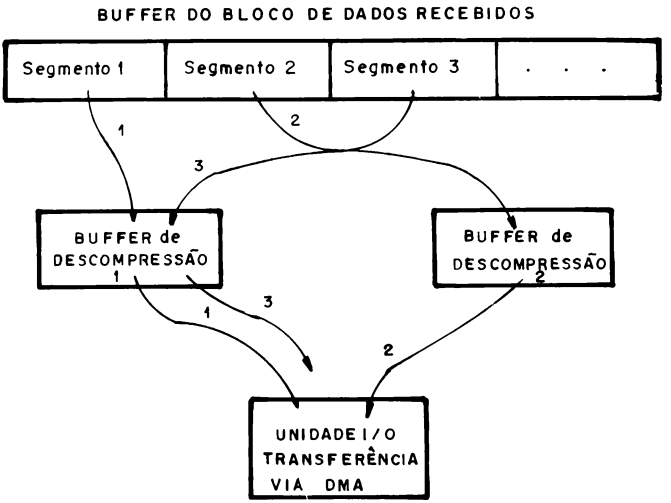


Figura 6.3 - Buffer duplo - isto pode ser empregado para minimizar os tempos de espera na transmissão.

Depois que o primeiro buffer de descompressão for preenchido e enquanto o segundo buffer estiver sendo preenchido, uma transferência por DMA pode ocorrer, de maneira a esvaziar o primeiro buffer. Quando o segundo buffer de descompressão é preenchido, outra transferência via DMA pode ocorrer e este buffer será esvaziado, enquanto o primeiro buffer estiver sendo preenchido novamente. Neste ponto, o fim do terceiro segmento do buffer do bloco de dados recebido ocorrerá quando o primeiro buffer de descompressão estiver preenchido novamente. Este preenchimento e esvaziamento simultâneo dos buffers duplos de descompressão continuará, até que o buffer do bloco de dados recebidos estiver completamente processado, em seguida um segundo bloco transmitido sobreporá o bloco previamente processado. O tamanho dos buffers de descompressão, bem como o buffer do bloco de dados recebidos dependerá do formato do bloco de transmissão, assim como da memória disponível do processador. Enquanto o buffer do bloco de dados recebidos deve igualar o tamanho do bloco transmitido, os buffers de descompressão, como explicado anteriormente, podem ser definidos com qualquer tamanho; contudo, quanto menor o tamanho, maior o processamento e o tempo de transferência de I/O necessários para descomprimir o bloco de dados recebidos. Assim, o programador deve, também, investigar a temporização do programa, antes de selecionar um tamanho definitivo de buffer de descompressão.

6.2 CONSIDERAÇÕES DE TEMPORIZAÇÃO

Para estimar a quantia de tempo do processador necessária à descompressão, diversos fatores requerem análise prévia. Primeiro, o programador deve estimar o número de instruções que a rotina de descompressão necessita e a quantidade de cada tipo de instrução, tais como: word, dupla word, referência de memória, instrução de deslocamento e número de posições de deslocamento e, etc. A partir do manual de referência de programação do fabricante do computador, a temporização de cada instrução pode ser obtida, ou expressa pelo número de ciclos de máquina necessários para executar o programa, ou indicada pelo período de tempo que é normalmente expresso em microssegundos ou 10^{-6}s. Se for expresso como uma função do número de ciclos de máquinas, o tempo por instrução pode ser obtido ao se multiplicar o tempo do ciclo de máquina pelo número de ciclos exigidos pela instrução. Em seguida, é obtida a soma do produto do número de instruções de cada tipo, vezes o tempo de execução por instrução. Isto se traduz no tempo de descompressão, estimado por bloco, menos o tempo de espera e o tempo de transferência de dados. O tempo de transferência de dados pode ser estimado ao se examinar a codificação necessária para programar e para iniciar uma transferência via DMA. Se o duplo buffer for empregado, o tempo real de transferência, que você deve considerar, pode ser zero ou um valor ínfimo. Como mostra a Figura 6.4, isto ocorre porque um buffer de descompressão é preenchido, enquanto o segundo buffer de descompressão está sendo esvaziado. Nesta figura, se um buffer de descompressão for esvaziado antes do segundo ser preenchido, não há tempo de espera e o único tempo extra adicional ao tempo normal de processamento de buffer, é o tempo de programação do DMA.

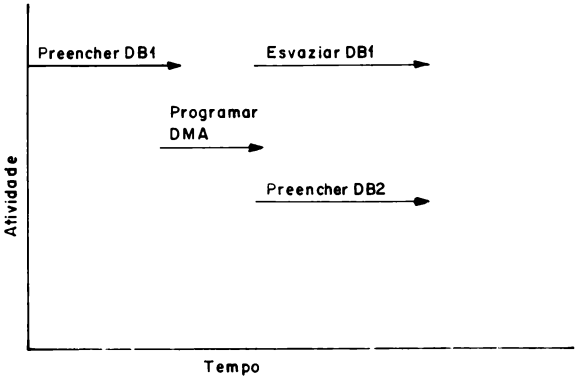


Figura 6.4 - Considerações de temporização usando o duplo buffer. Depois que o primeiro buffer é preenchido, os buffers subseqüentes são esvaziados e preenchidos com um tempo de espera mínimo, uma vez que somente a programação do DMA é necessária a cada operação, de maneira a ocorrer simultaneamente com o processamento.

Embora este tempo necessário à programação do DMA de set-up varie de computador para computador, o tempo total necessário é normalmente menor que 20μs. Para a maioria das máquinas, os endereços de início e de fim do buffers são carregados dentro de certos registros, juntamente com o código de endereçamento do periférico e, então, uma solicitação de transferência via DMA é iniciada. Já que é necessária à programação do DMA, é solicitada para esvaziar os buffers. Dois períodos de programação do DMA são necessários para esvaziar os dois buffers de descompressão.

Tempo Total

O tempo total de descompressão para os blocos de dados recebidos dependerá dos dados contidos nos blocos. Isto pode ser estimado para a condição de pior caso, onde o tipo mais eficiente de codificação de comprimento de fileira foi conduzido como explicado anteriormente. Como listado na Tabela 6.1, se são empregados buffers duplos de descompressão, 40 ciclos de segmento serão necessários se cada buffer de descompressão tiver 64 caracteres de tamanho. O tempo total de descompressão pode ser estimado, com base nas suposições prévias, ao obter o tempo para o ciclo do processo listado na Tabela 6.1, e multiplicar por 40. Se os dados estiverem sendo recebidos a 9600 bps, então 1200 cps (caracteres por segundo) estarão sendo recebidos. Naquela taxa de dados, ignorando os caracteres de controle e o tempo de

retransmissão, devido aos erros de linha, o buffer do bloco de dados recebidos de 240 caracteres será preenchido a cada 0,2 (240/1200) s. Isto significa que todo o processamento de descompressão, discutido previamente, bem como o tratamento da linha e outras funções de comunicações, devem ocorrer naquele tempo ou devem ser empregadas as filas de buffer, para permitir que o processamento sob condições de não pior caso seja encontrado e que, então permitirá que o processamento alcance os dados. Com contrapartidas entre o tamanho do buffer, a temporização e o emprego de buffer duplo, a maioria das rotinas de compressão, além das técnicas de codificação Huffman e Huffman modificada, pode ser facilmente adaptada para o processamento on-line.

Tabela 6.1 Ciclo de processamento

A. Processar o buffer do bloco de dados recebido.
Mover o ponteiro através do buffer, à medida que o buffer de descompressão 1 for preenchido.

B. Programar a transferência via DMA.
Buffer de descompressão 1 -> armazenamento.

C. Processar o buffer do bloco de dados recebido.
Mover o ponteiro através do buffer, à medida que o buffer de descompressão 2 for preenchido.

D. Programar a transferência via DMA.
Buffer de descompressão 2 - >armazenamento.

Como uma alternativa para se determinar o tempo, você pode computar o número de instruções de processamento disponíveis para a operação sobre os dados antes da compressão, resultando na degradação da performance. Como exemplo, considere um concentrador de dados servindo terminais de 1.200 bps e conectado ao computador central, via um elo de dados de alta velocidade, a 56.000 bps. Se houver 8 bits de dados por caractere, os terminais de 1.200 bps estarão se comunicando a 150 cps, enquanto o concentrador estará se comunicando a 7.000 cps.

Se os terminais conectados ao concentrador forem usados para compartilhamento de tempo, seu tamanho típico de mensagem é de 32 caracteres, que toma 32/150 ou 0,213 s para alcançar o concentrador a partir do terminal. Para enviar esta mensagem, do concentrador ao computador central, são necessários 32/7000 ou 0,004 57 s. A diferença de tempo entre o concentrador receber a mensagem e transmiti-la é de 0,208 76 s e pode ser considerada

uma "janela de operação". Esta é a quantia de tempo disponível para todo o processamento do concentrador, incluindo a compressão de dados antes de resultar em atraso da transmissão de mensagem.

Se assumirmos um ciclo de tempo de 2 μs e 4 ciclos por instrução, então é necessário um tempo de 8×10^{-6} para executar uma instrução. Dividindo o tempo da janela de operação, previamente computado, por este número, resulta em 26.095 instruções que podem ser executadas sobre os dados, antes de ocorrer um atraso na transmissão de mensagem. Se você conhecer o número atual das instruções programadas no software do concentrador, esse número pode ser subtraído de 26.095 para se determinar o número de instruções de compressão de dados, que podem ser escritos antes de causar um atraso na transmissão da mensagem.

Controle de Fluxo

Durante períodos de pesada atividade de transmissão, os buffers podem chegar ao seu limite e transferências de dados, subseqüentes capacidades farão com que os dados transbordem destes buffers, de fato perdendo-se. Para evitar este tipo de ocorrência, você pode incorporar vários procedimentos de controle de fluxo via software. Estes procedimentos são projetados para inibir e habilitar, seletivamente, as fontes de dados e, assim, evitar que os buffers de dados na memória transbordem.

O método mais comum de controle de fluxo é obtido pela transmissão de caracteres XON e XOFF para os terminais e para os computadores construídos para reconhecer estes caracteres de dados. A transmissão de um caractere XOFF pode ser usada para dizer à fonte de dados para inibir toda a atividade de transmissão futura, enquanto se processa o conteúdo do buffer de dados residente na memória. Uma vez que o buffer de dados é esvaziado ou alcançou uma determinada porcentagem de ocupação, pode-se transmitir um caractere XON para dizer à fonte de dados para continuar a transmissão. Devemos assegurar que estes dois caracteres não são comprimidos já que eles devem ser reconhecidos e trabalhados pelo hardware, que não está informado se os dados foram comprimidos.

Um segundo método de habilitar e inibir, seletivamente, a transmissão de dados pode ser obtido, ao pulsar para cima e para baixo o sinal liberado-para-envio (CTS) da interface RS-232 entre o dispositivo de execução de compressão e a fonte de dados. O nível alto e o nível baixo do sinal CTS funcionariam de um modo seme-

lhante à transmissão dos caracteres XON e XOFF discutidos previamente.

Uma seqüência XON, XOFF e o nível alto e baixo do sinal CTS são normalmente usados para controle de fluxo de dados assíncrono. Para o controle de fluxo de dados síncrono, você poderia considerar a alteração seletiva da taxa do gerador de clock. Como exemplo, usando este método você poderia transmitir os dados inicialmente a 19,2 kbps para um dispositivo de compressão onde a transmissão será de 9,6 kbps. À medida que os buffers de compressão se formam, pode-se diminuir a taxa de clock da entrada para evitar que os buffers transbordem. De modo oposto, você pode aumentar a taxa de clock se os dados forem suscetíveis à compressão e a ocupação de buffer for mínima.

Ligação de Rotinas

Quando se acrescenta a compressão de dados no processador auxiliar ou no controlador programável de comunicações, diversos métodos podem ser considerados para ligar o módulo de compressão ao software existente. O software de compressão pode ser adicionado a um módulo existente, ou pode ser codificado como um módulo separado. Para a última situação, diversos métodos podem ser usados para invocar sua operação. Se a rotina de compressão deve residir em um minicomputador, uma instrução desvie e guarde o endereço de retorno ou uma chamada de sub-rotina, são os dois métodos usados com mais freqüência para uma rotina invocar outra. Na conclusão da rotina, um retorno do endereço da instrução que a invocou mais um ocorre e a instrução seguinte à instrução de chamada ou de desvio é executada. Se a compressão tem de ser empregada com microcomputadores, diferenças de hardware e software podem impedir o uso de uma declaração de call ou de jump. Muitos microprocessadores são projetados de maneira que funções completas, tais como uma rotina de compressão de dados, podem ser colocadas dentro de chips que são conectados ao bus de I/O ou de DMA. Já que estes chips codificados são, então, considerados parte da memória do microprocessador, a ligação torna-se um problema de acesso à memória. Para estes microprocessadores que não têm instrução de (jump), desvie e guarde o endereço de retorno ou instrução de chamada de sub-rotina (call), o endereço do chip pode ser armazenado em um dos registros do processador. Então, um endereço indireto, através do registro, resultará no processador indo buscar o endereço do chip para iniciar a rotina de compressão.

CAP. 07

UTILIZANDO PRODUTOS DE HARDWARE E SOFTWARE PARA EXECUTAR A COMPRESSÃO

Muitas vezes você pode se beneficiar da compressão de dados, enquanto evita os esforços exigidos para analisar o tráfego de dados real e potencial, e desenvolver software para executar a compressão. Isto pode-se realizar alugando ou comprando dispositivos para executar a compressão de dados, que são especificamente projetados para serem usados em determinados ambientes de rede. Neste capítulo, examinaremos a utilização de diversos produtos de hardware e de software para obter um melhor entendimento do uso de dispositivos para executar a compressão. Os produtos abordados neste capítulo, foram escolhidos com propósitos ilustrativos e não devem ser analisados como um endosso a qualquer dispositivo.

A utilização de um dispositivo de hardware para executar a compressão ou o uso de um programa de software, elimina a necessidade de analisar o tráfego de transmissão de dados e desenvolver rotinas de software necessárias para comprimir dados. Contudo, os leitores estão avisados de que muitos dispositivos e programas de software para executar a compressão ,são projetados para serem mais eficientes ao operar com um tipo determinado de tráfego de dados, que pode não combinar com a consistência de dados que realmente transmite. Assim, estes dispositivos e programas não podem gerar um nível de taxa de compressão equivalente ao nível que poderia ser obtido, com o desenvolvimento de um sistema personalizado moldado para operar, com base na análise da consistência do tráfego de dados do usuário.

Na primeira seção deste capítulo, voltaremos nossa atenção para à operação e à utilização de diversos produtos de hardware disponíveis para executar a compressão comercialmente. Na segunda seção, voltaremos nossa atenção aos diversos tipos de pacotes de software existentes para executar a compressão. Esta seção inclui revisões operacionais de programas de software usados para aumentar a eficiência de monitores de teleprocessamento operando em mainframes, programas que podem ser usados para comprimir arquivos de banco de dados em mainframes e programas que podem ser usados para comprimir arquivos MS-DOS e PC-DOS nos computadores IBM PC e compatíveis.

7.1 PRODUTOS DE HARDWARE

Ao examinar os produtos de hardware para executar a compressão, distingüiremos estes produtos, de acordo com sua funcionalidade. Examinaremos, em primeiro lugar, os dispositivos que se restringem à compressão de fluxos de dados assíncronos. Em seguida, examinaremos a operação e a utilização de diversos produtos de hardware multifuncionais para executar a compressão.

Compressores de dados assíncronos

O número de terminais assíncronos ultrapassa, por uma fator de dez ou mais, o número de terminais síncronos em operação, e por esta razão muitos fornecedores desenvolveram produtos para uso em transmissão assíncrona. Estes produtos podem, normalmente, ser usados em transmissão por linhas alugadas, ou por rede telefônica chaveada. Seu uso primário, contudo, é para transmissão em rede telefônica chaveada pública. Quando usado neste ambiente de transmissão, a vantagem primária do dispositivo de compressão é sua habilidade de reduzir a duração da sessão de transmissão. Uma vez que o custo de uma chamada interurbana é aproximadamente proporcional a sua duração, ao diminuir a duração da sessão de transmissão, há uma redução no custo da chamada. Dois produtos especificamente projetados para comprimir dados assíncronos, que serão examinados com propósitos ilustrativos, são o CompressoRAD-1 da empresa RAD Computers e o turboMUX da empresa Chung Telecommunications.

CompressoRAD-1

O CompressoRAD-1 é uma unidade de compressão autônoma que também executa a conversão assíncrona para síncrona/ síncrona para assíncrona e fornece a capacidade de detecção e correção de erro para seu fluxo de dados. A Figura 7.1 ilustra como o CompressoRAD-1 poderia ser utilizado para transmitir dados via rede telefônica chaveada.

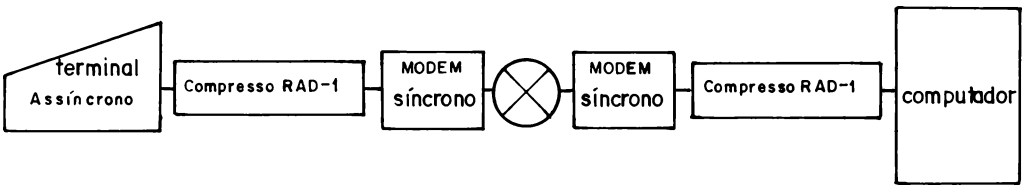


Figura 7.1 - Usando o CompressoRAD-1. O CompressoRAD-1 aceita dados assíncronos em 1200, 2400, 4800 e 9600 bps. Os dados são comprimidos pelo dispositivo, convertidos em um protocolo HDLC modificado e enviados ao modem acoplado, de acordo com a definição da taxa de relógio do modem.

O CompressoRAD-1 aceita dados assíncronos em 1200, 2400, 4800 ou 9600 bps. Os dados assíncronos podem ser compostos de sete ou oito bits por caractere, com um ou dois bits contendo parada, ímpar, par, marca, espaço ou sem paridade. Ao utilizar um algoritmo auto-adaptável automático, a taxa de compressão de dados de 2 : 1 a 4 : 1 é obtida, segundo a empresa RAD Computers, Inc.

Além de comprimir dados, o dispositivo converte o fluxo de dados assíncronos em um protocolo de controle de enlace de dados de alto nível (HDLC) modificado síncrono. Através da adição de um caractere de teste de redundância cíclica para cada bloco de dados transmitido, consegue-se que a capacidade de detecção e correção de erros de extremidade a extremidade, seja incorporada na transmissão de dados.

A taxa de dados enviados do CompressoRAD-1 é determinada pelo relógio do modem acoplado, com o dispositivo capaz de operar entre 600 e 4800 bps. Um buffer no dispositivo é usado para compensar as diferenças na compressibilidade dos dados de entrada. Sendo assim, quando os dados de entrada no CompressoRAD são comprimidos, e representam um fluxo de dados maior que

a taxa pela qual os dados são enviados ao modem síncrono acoplado, o buffer enche, servindo como uma área de armazenamento temporário. Uma vez que o buffer tem um tamanho finito, o CompressoRAD emprega dois métodos para impedir a entrada de dados adicionais, quando o buffer estiver preenchido até um nível predefinido. Conhecido como fluxo de controle, os dados são regulados no dispositivo pela transmissão de caracteres XOFF e XON e pela subida e descida dos sinais de controle liberados para envio RS-232. Quando o armazenamento no buffer o enche até um nível predefinido, o CompressoRAD transmitirá um caractere XOFF ou derrubará o sinal CTS na interface RS-232. A escolha do método, que você seleciona, baseia-se nas especificações operacionais do terminal ou da interface do computador acoplado ao CompressoRAD. Alguns terminais e interfaces de computador reconhecem XON e XOFF como caracteres de sinalização ao controle do fluxo, 73 habilitando o CompressoRAD a ser configurado de maneira a usar este método de controle do fluxo. Para terminais e interfaces de computador, que não reconhecem este método de controle de fluxo, o CompressoRAD pode ser configurado para capacitar e impedir a transmissão ao dispositivo via sinal de controle CTS. Assim, os dados são impedidos pelo dispositivo de serem transmitidos, uma vez que o CompressoRAD derrubou o CTS. Depois, ele levantará este sinal de controle, quando o buffer estiver esvaziado até um outro nível predefinido. A emissão de sinais de controle do fluxo, baseada na quantidade de dados no buffer do dispositivo, está ilustrada na Figura 7.2.

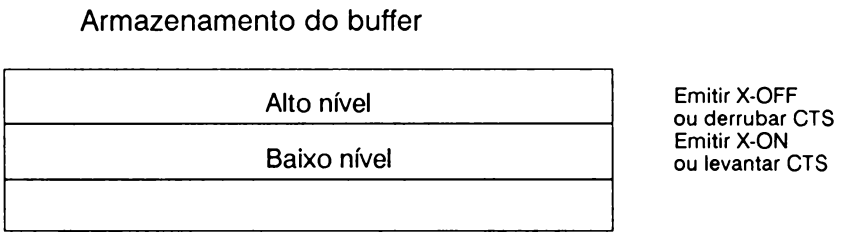


Figura 7.2 - Controle do buffer. Para evitar o estouro no armazenamento do buffer e evitar a perda do fluxo de dados , o controle ocorre através da subida e descida do sinal de controle clear-to-send (CTS) ou através da geração de caracteres XOFF e XON.

O uso dos caracteres XOFF e XON e a subida e descida do sinal de controle CTS são utilizados pela maioria dos produtos que executam a compressão em dados assíncronos. Quando dados síncronos são comprimidos, o controle do fluxo é normalmente realizado, reduzindo-se o sinal de taxa de relógio passado de um modem acoplado ao dispositivo de compressão para um terminal, uma interface de computador ou para um dispositivo semelhante, que opere sincronamente. Um método comum, usado pelos fabricantes de hardware, é dividir a taxa de relógio pela metade durante diversas vezes, até que o buffer esvazie até um nível predefinido. Quando isto ocorrer, a taxa de relógio é, então, dobrada diversas vezes até que a taxa de dados original seja restaurada.

Para entender a economia associada à utilização do CompressoRAD e dos produtos similares que executem a compressão, vamos assumir que o custo diário da comunicação na rede telefônica chaveada, entre o terminal remoto e a instalação do computador central seja de $5.00, custo esse correspondente a uma chamada interurbana de menos de 30 minutos. Ao supor que se faça uma chamada por dia nos 22 dias úteis do mês, o custo do uso da rede telefônica chaveada é $110.00 por mês ou $1,320.00 por ano. Agora, vamos supor que o uso de dois dispositivos de compressão de dados assíncronos reduza, pela metade, a duração da sessão de transmissão, resultando na possibilidade de redução de custo das comunicações em $660 por ano de operação. Ao que se pode comparar esta economia potencial de custo?

Ao usar o CompressoRAD, ilustrado na Figura 7.1, ou um produto similar que execute a compressão semelhante, você deve obter dois modems síncronos, além de dois dispositivos de compressão. Em comparação, você usaria dois modems assíncronos de baixo custo, ao transmitir dados sem o uso de dispositivos de compressão assíncronos. Sendo assim, do ponto de vista da economia:

$$(\Delta M) * 2 + 2 * C \leq 660 * EL$$

onde:

ΔM =	diferença de custo entre os modems assíncronos e os síncronos
C =	custo unitário de cada dispositivo para executar a compressão
EL =	vida ou uso útil em anos dos dispositivos para executar a compressão.

turboMUX

O turboMUX de empresa Chung Telecommunications é um compressor de dados assíncrono que tem dois modos de operação, dos quais um oferece uma operação em rede significativamente diferente do dispositivo CompressoRAD discutido anteriormente. O turboMUX suporta uma fonte de dados assíncronos de entrada a 2400 bps ou duas fontes de dados assíncronos, operando a 1200 bps. Neste último modo de operação, o dispositivo suporta duas sessões de transmissão de dados por chamada em uma rede telefônica pública chaveada, como apresentado na Figura 7.3.

Como apresentado na Figura 7.3, duas fontes de entrada de dados assíncronas são suportadas pelo turboMUX. Projetado para trabalhar com o modem Bell System 212A comumente usado, operando sincronamente, a unidade de compressão, em essência, permite que um modem de 1200 bps opere com o dobro de sua velocidade nominal. Como característica adicional, além do suporte à fonte dupla de dados, o turboMUX opera de forma semelhante ao CompressoRAD, contendo uma área de buffer que é controlada por meio do processo de controle de fluxo discutido anteriormente, e onde um mecanismo automático de detecção de erro e retransmissão é incorporado no fluxo de dados entre os dois turboMUX, fornecendo uma transmissão livre de erros de ponta a ponta.

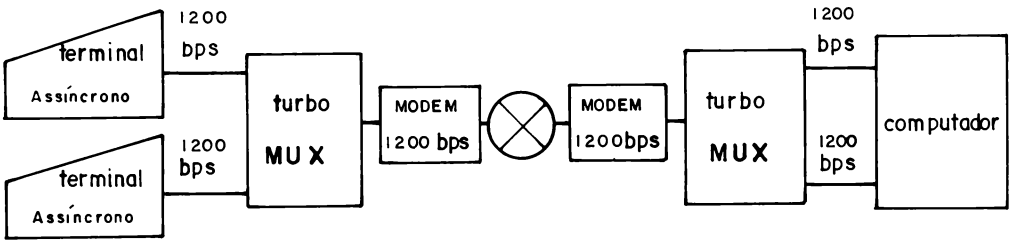


Figura 7.3 - Utilização típica da rede turboMUX. O turboMUX permite duas sessões de transmissão assíncronas de dados ocorrendo em uma sessão de rede telefônica chaveada pública.

Dispositivos multifuncionais de compressão

Uma das aplicações mais populares dos dispositivos de executar a compressão é reduzir a quantidade de dados a serem multiplexados, aumentando, assim, a capacidade de operação dos multiplexadores. Uma evolução natural no desenvolvimento dos

dispositivos de compressão foi incluir tanto a multiplexação estatística, como a compressão de dados em um único dispositivo de hardware. As duas linhas de produtos, que apresentam esta capacidade multifuncional, são: a linha de produtos Scotsman III da RacalVadic e a linha de produtos Streamer da Datagram Corporation que, ao ser adquirida pela empresa Memotec Data, recebeu o novo nome MC para a série de produtos MC.

Scotsman III

O Scotsman III da Racal-Vadic é um compressor de dados autônomo que pode ser configurado de diversos modos, para incluir a adição de um multiplexador opcional de quatro canais. A Figura 7.4 ilustra a utilização do Scotsman III como um multiplexador, comprimindo quatro fluxos de dados bissíncronos em uma única linha de transmissão.

A utilização do Scotsman III, ilustrado na Figura 7.4, deve ser comparado ao emprego de um multiplexador estatístico. Uma vez que a maioria dos multiplexadores estatísticos não execute compressão de dados, retardos significantes de tempo ocorreriam na tentativa de multiplexar quatro fontes de dados bissíncronas de 9600 bps em uma linha de transmissão de 9600 bps.

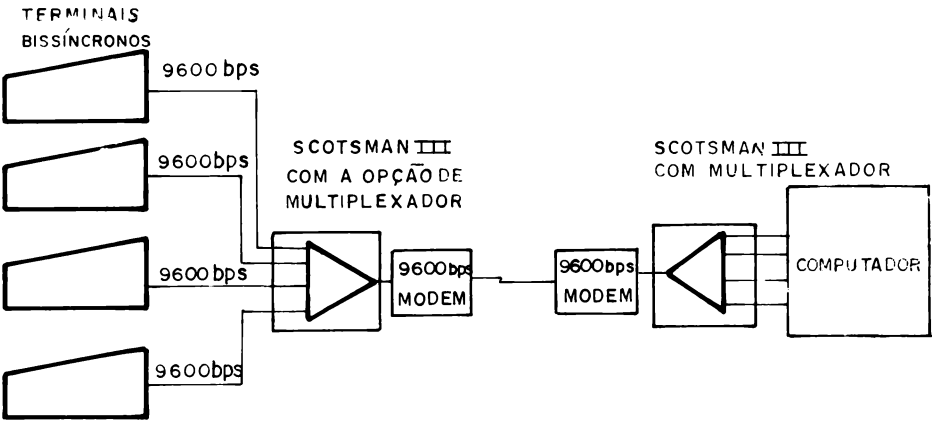


Figura 7.4 - Usando o Scotsman III como um multiplexador. Um multiplexador opcional de quatro canais pode ser acrescentado ao Scotsman III da Racal-Vadic, resultando na capacidade de transmitir quatro fontes de dados síncronas binários de alta velocidade em uma única linha de transmissão.

Em outras situações, os multiplexadores estatísticos podem ser úteis com dados bissíncronos, através do uso de canais de banda passante. Quando isto ocorre, somente dados assíncronos são realmente multiplexados estatisticamente, enquanto dados bissíncronos são multiplexados por divisão de tempo na janela de tempo predefinida na ligação multiplexada composta, como apresentado na Figura 7.5.

Quando dados bissíncronos são multiplexados via canal de banda passante, a taxa de dados de entrada bissíncronos reduz a parcela disponível, na largura de banda do canal composto, para multiplexar outras fontes de dados. Como exemplo, uma fonte de dados bissíncrona, operando a 4800 bps, reduziria a largura de banda do canal composto, que deveria multiplexar outras fontes de dados, de 4800 bps, uma vez que os dados bissíncronos são multiplexados via um canal de banda passante. Sendo assim, se o canal composto opera a 9600 bps, somente 4800 bps estariam disponíveis para a multiplexação estatística de outras fontes de dados.

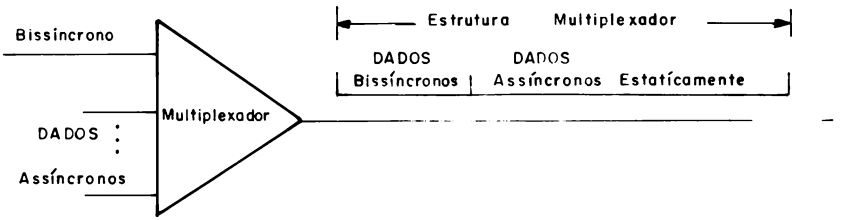


Figura 7.5 - Multiplexação em banda passante. Quando os dados são multiplexados usando um canal de banda passante, o tamanho do canal de banda passante reduz, proporcionalmente, a capacidade da estrutura do multiplexador para multiplexar outros dados.

Devido às limitações envolvidas ao processar dados bissíncronos com canais de banda passante, a maioria dos multiplexadores estatísticos, que usam esta técnica, são limitados ao suporte de apenas uma fonte de dados bissíncrona. Sendo assim, o Scotsman III pode ser efetivamente empregado em situações, onde diversas fontes de dados bissíncronas de alta velocidade transmitem para uma localização comum.

Streamer

A série de dispositivos Streamer da Datagram Corporation, que recebeu novos nomes: MC 504 e MC 508, quando esta empresa foi adquirida pela Memotec Data, são multiplexadores estatísticos que incorporam compressão de dados, resultando na eficiência estatística global de até 4 : 1, segundo a empresa. Em comparação com muitos produtos semelhantes, a série de dispositivos Streamer suporta uma ampla variedade de protocolos orientados-a-bit, que inclui controle de enlace de dados síncronos (SDLC), X.25, HDLC, controle de enlace de dados Univac (UDLC) e protocolos orientados-a-caractere que incluem transmissão bissíncrona e assíncrona.

Cada Streamer, agora comercializado como multiplexadores de compressão de dados MC 504 e MC 508, analisa os dados recebidos com base na interface de acesso individual e emprega um algoritmo auto-adaptável de compressão em cadeia, para comprimir as cadeias de ocorrência mais freqüente em cadeias menores. Uma vez que os dados transmitidos em cada direção podem variar na composição, cada multiplexador que executa a compressão mantém tabelas de compressão de chegada e de partida para cada interface de acesso, as quais são independentes umas das outras. Uma vez que o fluxo de dados da interface de acesso esteja comprimido, o fluxo de dados de todas as interfaces de acesso é, então, multiplexado.

Devido à habilidade da série de dispositivos Streamer suportar diversos protocolos de enlace de dados, este dispositivo de comunicação para executar a compressão pode ser usado em uma grande variedade de aplicações em rede. A Figura 7.6 ilustra uma aplicação potencial que mostra a versatilidade do Streamer. O Streamer executa, em primeiro lugar, funções típicas de um multiplexador estatístico; típicas para incluir: bits de partida e de parada da transmissão assíncrona e passagem de dados somente dos terminais ativos. Depois que o compressor do dispositivo usa uma tabela de compressão auto-adaptável para codificar os dados, estes dados comprimidos resultantes são multiplexados estatisticamente, resultando em eficiências globais de compresssão estatística de até [4 : 1]. Uma das características mais interessantes do Streamer é sua análise e construção de tabelas auto-adaptáveis para cada sentido do fluxo de dados interface por interface. Esta abordagem maximiza as eficiências potenciais obtidas ao comprimir dados à medida em que assegura diferentes características de dados fluindo em um sentido, como sendo prontamente reconhecidos.

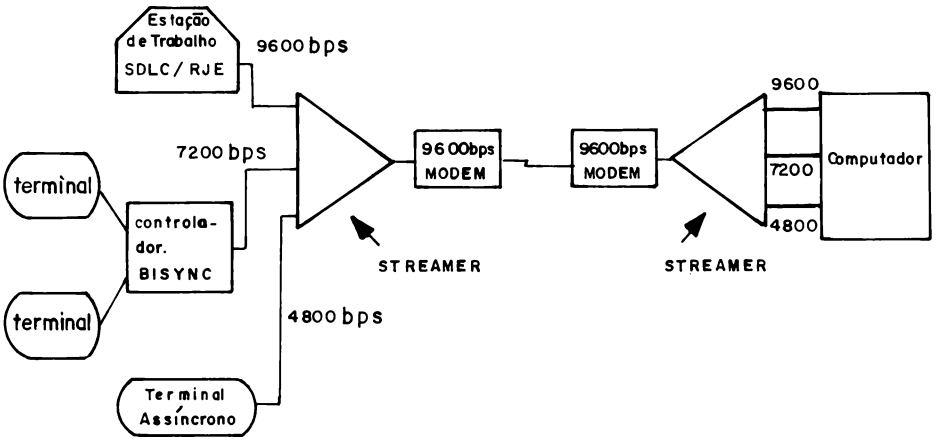


Figura 7.6 - Usando o Streamer. A série Streamer da Datagram de multiplexadores que executam a compressão, suporta um grande número de protocolos orientados-a-bit e orientados-a-caractere.

7.2 PRODUTOS DE SOFTWARE

Nesta seção, examinaremos os três tipos de produtos de software para compressão de dados. Em primeiro lugar, voltaremos nossa atenção ao software de compressão para monitor de teleprocessamento, projetado para reduzir o tempo de resposta do terminal, reduzindo a quantidade dos dados transmitidos, entre o aplicativo do mainframe e o dispositivo de terminal. Isto será seguido por uma visão detalhada de alguns produtos, para executar a compressão de banco de dados em mainframe e software projetado para comprimir arquivos em microcomputador.

Produtos de compressão para monitor de teleprocessamento

Várias empresas de software desenvolveram produtos de compressão de dados, para melhorar a eficiência de uma variedade de sistemas monitor de teleprocessamento IBM. Alguns produtos são projetados para trabalhar com sistemas monitor de teleprocessamento IBM, tais como CICS, IMS e TSO. Outras empresas desenvolveram produtos que operam ao nível do método de acesso virtual por telecomunicações (VTAM - Virtual Telecommunications Access Method). Ao operar no nível VTAM, o pacote de programas de

compressão fornece compressão para todos os aplicativos operando no computador central, o qual elimina a necessidade de obter um produto para executar a compressão separado para cada aplicação VTAM. Independente da localização, onde o programa de compressão opera, cada programa utiliza um núcleo de métodos de compressão de dados equivalente, que essencialmente iguala seu desempenho. As técnicas de compressão usadas pelo software avançado do monitor de teleprocessamento, incluem a eliminação de todos os caracteres de repetição consecutiva, economia de imagem de saída, e espelhamento na chegada.

Para ilustrar o efeito da operação de um programa de compressão no sistema monitor de teleprocessamento, considere o segmento de rede ilustrado na Figura 7.7. Nesta ilustração, um grande mainframe operando com CICS, IMS e TSO é acessado por 32 terminais localizados em um lugar remoto. Se cada operador de terminal apertasse uma tecla de Função de Programa (PF) para exibir uma nova imagem de tela, o número de bits que seria transmitido para cada terminal excluindo os controles do protocolo, seria 80*25*8 ou 16 000 por terminal. Assumindo o pior caso, onde os operadores dos 32 terminais apertassem, simultaneamente, a tecla PA, PF, ou enter, 16.000*32 ou 512.000 bits seriam exigidos para a transmissão. Se a linha conectando a localização remota ao local do computador central operasse a 19.2 kbps seria exigido 512.000 / 19.200 ou, aproximadamente, 27 s para transmitir todas as imagens de tela. Uma vez que este cálculo não leva em consideração, nem o tempo de processamento exigido pelo computador central para interpretar a solicitação chave e recuperar a tela solicitada, nem o efeito do dispêndio associado ao protocolo

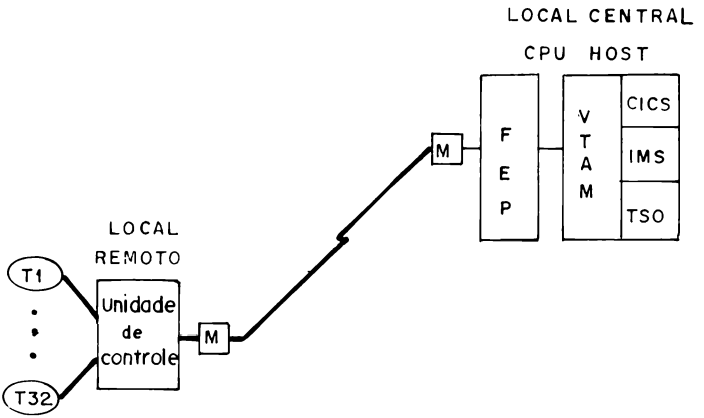


Figura 7.7 - Segmento de rede para ilustrar os benefícios do software de compressão para monitor de teleprocessamento.

de transmissão, o tempo real seria consideravelmente maior. Mesmo se não fosse assim, esperar 27 s para uma resposta é um nível inaceitável de tempo de resposta. Assim, qualquer técnica que possa reduzir o tempo de resposta impulsionará a produtividade do operador do terminal. De fato, embora 32 terminais, conectados à unidade de controle, estejam ilustrados na Figura 7.7, normalmente, conecta-se um número menor justamente para se evitar tempos de resposta excessivos. Para grandes empresas, com centenas de terminais localizados em diferentes escritórios longe do computador central, a redução no número de terminais conectados a cada unidade de controle, normalmente resultará no uso de unidades de controle adicionais, cada uma exigindo modems e uma linha de comunicação. Assim, qualquer método que reduza o tempo de resposta do terminal pode, também, permitir que mais terminais sejam conectados a uma unidade de controle, que pode resultar em uma redução no custo das comunicações.

Métodos de compressão

Em muitas aplicações, cadeias de caracteres que contenham uma letra ou caractere são repetidas, sucessivamente, diversas vezes. Por exemplo, a cadeia de caracteres '---...-Página n' é usada como um separador de página em muitos aplicativos. Esta cadeia pode ter uma fileira de 60 a 70 traços seguidos por um número de página. Em outros aplicativos, uma cadeia de sinais de igual '= = = = = = = =' pode ser usada para sublinhar um campo ou separá-lo de outros campos. Embora não seja visível, alguns aplicativos geralmente usam uma cadeia de espaços em branco, para apagar o conteúdo de um campo existente em uma tela.

Os produtos de compressão para monitor de teleprocessamento examinam nos fluxos de dados de partida, os caracteres de repetição consecutiva. Quando encontrados, os caracteres de repetição são trocados por uma seqüência de quatro bytes, que usa o código Repetição-no-endereço reconhecido pelos terminais já no primeiro byte da seqüência de quatro bytes. O código Repetição-no-endereço é seguido por um endereço de tela de dois bytes que identifica para onde irá o caractere repetido. Isto é, então, seguido pelo caractere real a ser repetido. A Figura 7.8 apresenta o formato usado para comprimir cadeias de repetição. Observe que cada caractere é codificado em hexadecimal. Assim, 3C, que é o código hexadeci-

mal para o código Repetição-no-endereço, representa um caractere.

O segundo método empregado pelo software para executar a compressão para monitor de teleprocessamento, é freqüentemente referido como economia de imagem de saída, ou supressão de imagens de dados redundantes. Para realizar esta tarefa, o programa de compressão intercepta as imagens de tela de saída e compara a imagem da tela atual, com a imagem transmitida anteriormente, suprimindo a transmissão de campos ou porções de campos comuns. As Figuras 7.9 e 7.10 ilustram a operação de economia de imagem. A Figura 7.9 mostra uma tela de terminal

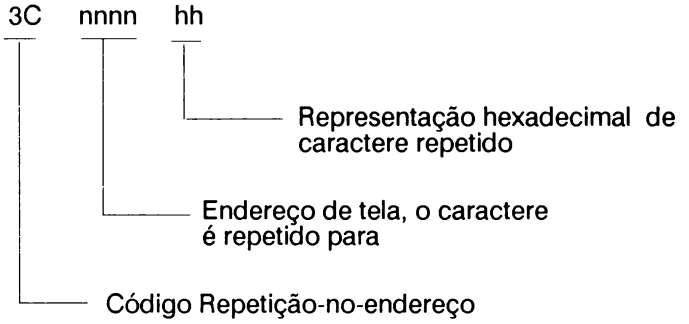


Figura 7.8 - Formato de compressão de caractere repetido.

típica, quando um operador está usando um programa de manutenção de mala direta de cliente. A Figura 7.10 mostra como a próxima tela pode aparecer. Observe que somente os campos de nome, data e quantia mudaram. Assim, transmitir somente aquelas mudanças, no lugar de uma tela completamente nova, pode reduzir, de forma significante, o número de caracteres necessários para atualizar a tela.

FRED UNGER HARDWARE CORP
==========
MANUTENÇÃO DE MALA DIRETA DE CLIENTE

Nome do Cliente
Sobrenome.....: Held
Nome.....: Beverly

Endereço do cliente
Rua.....: 4736 Oxford Road
Cidade, Estado....: Macon, GA
Cep.....: 31210

Última compra
Data.....: 6/18/89
Quantia.....: $3.27

Help: F1	Add: PF2	Update: PF3	Deletar: PF4
Fwd: F5	Back: PF6	Search: PF8	Return: PF12

Figura 7.9 - Tela inicial

O terceiro método, usado por muitos programas para executar a compressão para monitor de teleprocessamento, objetiva o aumento da eficiência de um programa aplicativo. Como previamente discutido no Capítulo 3, onde a compressão de modo de formulários foi descrita, quando um operador insere dados dentro no campo da tela, o rótulo de dados associado ao campo é modificado. O rótulo de dados modificado (MDT - modified data tag) também pode ser ligado por um aplicativo, quando os dados são transmitidos para o terminal. Se isto for feito, o campo será transmitido de volta pelo terminal, mesmo se o campo não for alterado pelo operador.

Para obter o espelhamento na chegada, o programa de compressão desliga todos os MDTs, à medida que a transmissão flui para os terminais, assim resulta que somente os campos modificados serão transmitidos pelo operador. Ao manter uma imagem da tela na memória, o programa de compressão sabe quais os campos que tinham seus MDTs ativados. Se estes campos não forem transmitidos de volta pelo terminal, eles serão inseridos no fluxo descendente pelo programa de compressão, antes daquele programa passar os dados ao programa aplicativo.

FRED UNGER HARDWARE CORP
==========
MANUTENÇÃO DE MALA DIRETA DE CLIENTE

Nome do Cliente
Sobrenome.....: Held
Nome.....: Gilbert

Endereço do cliente
Rua.....: 4736 Oxford Road
Cidade, Estado... : Macon, GA
Cep.....: 31210

Última compra
Data.....: 7/19/91
Quantia.....: $43.27

Help: F1 Add: PF2 Update: PF3 Deletar: PF4
Fwd: F5 Back: PF6 Search: PF8 Return: PF12

Figura 7.10 - Tela seguinte. Somente os campos de nome, data e quantia são transmitidos, uma vez que todos os outros campos não sofreram alteração, em relação á tela transmitida anteriormente

Dois programas de compressão para o monitor de teleprocessamento avançado são Datapacker/II comercializados pela empresa H&M Systems Software, da cidade de Maywood, NJ e VTAM-EXPRESS comercializado pela empresa SoftTouch Systems da cidade de Oklahoma, OK. O primeiro programa é um produto CICS, enquanto o segundo suporta aplicativos CICS, IMS e TSO.

Compressão de banco de dados

A empresa InfoTel Corporation de Tampa, FL, comercializa uma série de programas de software de compressão de dados para comprimir diferentes tipos de bancos de dados. Os programas comercializados sob o nome de INFOPAK incluem INFOPAK DB2, INFOPAK VSAM, INFOPAK IMS, INFOPAK IDMS e INFOPAK SEQ.

INFOPAK DB2 reduz a quantia de armazenamento do dispositivo de armazenamento de acesso direto (DASD) exigido por bancos de dados DB2. Segundo a empresa, as taxas de recuperação de espaço em disco, entre 50 e 75 por cento, podem ser tipicamente obtidas com um mínimo de sobrecarga de CPU. Além disso, já que as necessidades de armazenamento em disco são consideravelmente reduzidas, o número de operações de entrada/saída (I/O) requisitado pelos aplicativos DB2 é, também, reduzido.

INFOPAK DB2 usa uma técnica de codificação Huffman modificada em que o módulo de varredura do programa executa uma análise tridimensional de dados e, então, toma decisões de compressão, com base na natureza dos dados. O programa de varredura usa inteligência artificial para determinar, entre os diversos métodos diferentes de compressão existentes, que métodos usar para obter um ótimo desempenho. A maioria dos dados é comprimida, usando a técnica de codificação Huffman, mas outras técnicas de compressão, incluindo a substituição de caractere de repetição, a representação de bit dos campos em branco e de zeros e um algoritmo proprietário para certos tipos de dados, são também usadas.

O INFOPAK VSAM da InfoTel permite a compressão dos conjuntos de dados KSDS ou ESDS, sendo executados sob os sistemas operacionais MVS ou MVS/XA da IBM. Este programa é compatível com CICS e é completamente transparente ao programa aplicativo.

Para manter a transparência do programa aplicativo, o programa de compressão assegura que cada chave retém sua posição de registro exata. Para realizar isto, as chaves não comprimidas são guardadas em uma área de trabalho separada, durante a compressão. Depois da compressão, as chaves são inseridas dentro do registro comprimido nas posições exatas que elas ocuparam antes da compressão. Então, o novo registro comprimido é fornecido ao VSAM para armazenamento físico. Se os dados comprimidos não preencherem o registro depois da inserção das chaves, o registro é estendido pela adição de bytes de zeros binários, para obter o comprimento mínimo de registro. Isto serve para manter a transparência do programa aplicativo.

A Figura 7.11 ilustra a operação do INFOPAK VSAM no registro que tem chaves KSDS primária e alternativa, localizadas nas posições 12 e 40, respectivamente. Observe que quando o registro é comprimido, seu comprimento dependerá da susceptibilidade dos dados para compressão.

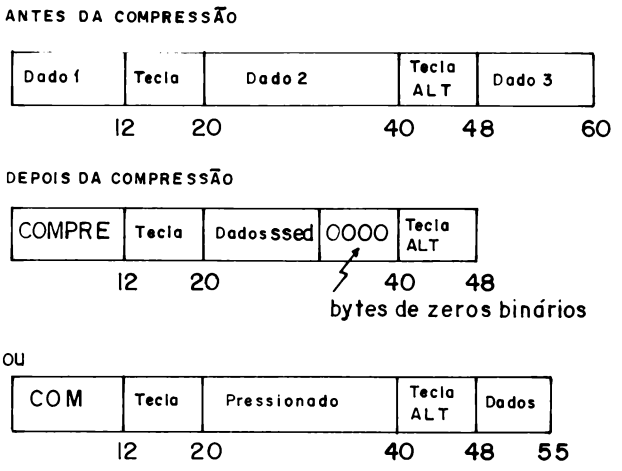


Figura 7.11- Operação do INFOPAK VSAM.

O INFOPAK IMS habilita os bancos de dados IMS para que sejam comprimidos, enquanto INFOPAK IDMS habilita a compressão de bancos de dados IDMS. Os programas de compressão resultam em bancos de dados comprimidos que são totalmente transparentes para os programas aplicativos. O INFOPAK SEQ da empresa InfoTel opera em arquivos que são processados seqüencialmente. Os arquivos seqüenciais comprimidos resultantes são identificados por um sufixo especial suplementar ao nome do conjunto de dados. Além de apresentar a característica do sufixo, o INFOPAK SEQ é transparente à linguagem de controle de serviço (JCL - job control language) e é completamente transparente a todos aplicativos.

Devido ao alto custo do DASD, a série de produtos INFOPAK ganhou ampla aceitação de diversas empresas privadas e agências governamentais. São usuários do INFOPAK: agências do Departamento de Defesa dos Estados Unidos, grandes universidades, organizações de assistência à saúde, companhias de seguro e diversas organizações que mantem uma variedade de bancos de dados grandes. Uma vez que a redução de apenas um sistema de disco IBM 3390 ou 3380 pode, em princípio, recuperar o custo do programa, o retorno de investimento pode ser instantâneo. De fato, muitas firmas podem economizar de forma significativa, ultrapassando o custo do software e, ainda, obtendo capacidade adicional de armazenamento, através do uso do espaço livre de armazenamento proveniente do uso do programa. Isto pode proporcionar

anos de capacidade de expansão às grandes organizações de processamento de dados, através do uso do equipamento existente.

Compressão de arquivo em computadores pessoais

O recente crescimento exponencial no uso dos computadores pessoais, provocou um aumento, tanto no número de programas sendo comercializados, como em sua complexidade. Como os produtos de software proliferaram, muitos operadores de bancos de software rapidamente preencheram os discos rígidos com programas e rapidamente ficaram sem espaço de armazenamento. Este problema levou ao desenvolvimento de uma série de programas de compressão de domínio público que começaram a alcançar popularidade em meados dos anos 80.

Entre os programas de compressão de domínio público mais recentes estão o SQ e USQ, escritos por Dick Greenlay. SQ (do Inglês squeeze - espremer) foi baseado no algoritmo de codificação Huffman e arquivos comprimidos para computadores pessoais, enquanto o USQ (do Inglês unsqueeze - não espremido) descomprimiu arquivos previamente comprimidos. SQ e USQ, bem como programas semelhantes, proporcionaram diversos benefícios aos usuários. Isto, por outro lado, permitiu que bancos de software armazenassem mais programas. Além disso, proporcionou aos fabricantes de software e aos usuários finais, a habilidade de distribuir mais software em um número menor de disquetes. Outro significante benefício foi a redução em tempo e custo com comunicação, uma vez que os arquivos comprimidos poderiam ser carregados, mais rapidamente, do que arquivos não comprimidos.

ARC - o arquivo utilitário que comprime dados

Fundamentando-se no desenvolvimento inicial de programas de compressão e descompressão de arquivo para computador pessoal, a empresa System Enhancement Associates da cidade de Wayne, NJ, nos Estados Unidos, desenvolveu um utilitário que comprime dados para computadores pessoais que operam sob o DOS e OS/2. Este programa, chamado ARC, executa uma operação de local de arquivamento, agrupando uma série de arquivos específicos junto a um arquivo, de modo que os arquivos individuais possam ser recuperados intactos. Além disso, o ARC comprime, automaticamente, os arquivos a serem arquivados, de maneira que o local de arquivamento resultante ocupe uma quantidade mínima de espaço.

Quando o ARC é utilizado para adicionar um arquivo em um local de arquivamento, ele analisa o arquivo para determinar qual dos três métodos de armazenamento resultará em maior economia. Estes três métodos são:

1. Sem compressão; o arquivo é armazenado como se encontra.
2. Compressão de caractere repetido; seqüências repetidas de mesmo valor de byte são encolhidos em uma seqüência de código de três bytes.
3. Compressão dinâmica Lempel-Ziv; o arquivo é armazenado como uma série de códigos de bit de tamanho variável que representam cadeias de caractere e que são criados 'na hora'.

Observe que, uma vez que um dos três métodos não envolve compressão, a entrada de local de arquivamento resultante nunca será maior que o arquivo original.

Usando o ARC

O ARC é chamado através de um comando com o seguinte formato:

ARC <x> <nomearc> [<gabarito>. . .]

onde <x> é uma letra de comando ARC (veja abaixo), tanto em letra maiúscula como minúscula.

<nomearc> é o nome do local de arquivamento a atuar, com ou sem uma extensão. Se não for fornecida uma extensão, aí o ARC é assumido. O nome do local de arquivamento pode incluir especificadores de caminho e de drive de disco.

<gabarito> é um ou mais gabaritos de nome de arquivo. Os caracteres 'wildcard' * e ? podem ser usados. Um gabarito de nome de arquivo pode incluir um especificador de caminho ou de drive de disco, embora não seja sempre significativo.

Se o ARC é chamado sem argumentos (digitando ARC e apertando a tecla Enter), então um breve resumo do comando é exibido.

Apresentamos, a seguir, um breve resumo dos comandos ARC disponíveis:

a = adicionar arquivos ao local de arquivamento
m = mover arquivos para local de arquivamento
u = atualizar arquivos no local de arquivamento
f = renovar arquivos no local de arquivamento
d = deletar arquivos do local de arquivamento
x,e = extrair arquivos do local de arquivamento
r = executar arquivos do local de arquivamento
p = copiar arquivos do local de arquivamento para saída padrão
l = listar arquivos no local de arquivamento
v = listagem verbosa de arquivos no local de arquivamento
t = testar a integridade do local de arquivamento
c = converter a entrada para novo método de armazenamento.

Encontra-se, a seguir, um breve resumo das opções ARC disponíveis, que podem alterar o modo de trabalhar de um comando:

m = mover arquivos para local de arquivamento
z = incluir subdiretórios no local de arquivamento
v = modo verboso
b = reter uma cópia backup do local de arquivamento
s = suprimir a compressão (somente armazenar)
w = suprimir as mensagens de perigo
n = suprimir observações e comentários
o = escrever por cima dos arquivos existentes ao extrair
5 = produzir somente arquivos compatíveis como o nível 5.
g = codificar ou decodificar a entrada do local de arquivamento.

Para ilustrar a operação dos comandos ARC, vamos examinar o uso dos comandos A (Adicionar), U (Atualizar), e F (renovar), que permitem que os arquivos sejam adicionados ao local de arquivamento:

Adicionar sempre adiciona o arquivo

Atualizar é diferente de Adicionar, no fato em que o arquivo é somente adicionado se não estiver no local de arquivamento, ou se for mais novo que a entrada correspondente no local de arquivamento.

Renovar é semelhante a Atualizar, exceto no fato de que novos arquivos não são adicionados ao local de arquivamento; somente os arquivos, que já estão no local de arquivamento, são atualizados.

Por exemplo, se você quisesse adicionar um arquivo chamado TEST.DAT ao local de arquivamento chamado MY.ARC, você usaria um comando do seguinte formato:

ARC a my test.dat

Se você quisesse adicionar todos os arquivos com uma extensão .C e todos os arquivos chamados STUFF ao local de arquivamento chamado JUNK.ARC, você poderia digitar:

ARC a junk.c stuff.**

Se você tivesse um local de arquivamento chamado TEST.ARC e quisesse adicionar nele todos os seus arquivos com uma extensão .TXT que foram criados ou alterados, uma vez que foram arquivados por último, então você digitaria:

ARC u text.txt*

Se você tiver um conjunto de arquivos em seu diretório atual, com cópias de segurança sendo armazenadas em um local de arquivamento chamado SAFE.ARC e, então, caso quisesse certificar-se de que todos os arquivos no local de arquivamento apresentam a última versão, você digitaria:

ARC f safe

Atualizar e Renovar - são semelhantes no fato de observarem a data e horário da última alteração do arquivo e somente adicioná-lo, se este arquivo foi alterado, já que foi arquivado por último. Diferem, contudo, pois Atualizar adicionará novos arquivos, enquanto Renovar não.

Em outras palavras, Atualizar procura os arquivos no disco e adiciona-os se são novos ou se foram alterados, enquanto Renovar examina o local de arquivamento e tenta atualizar os arquivos que já estão lá.

Para ilustrar a eficiência potencial do ARC, vamos examinar seu principal arquivo executável chamado ARC602.EXE, que, quando copiado ao computador pessoal do usuário, ele se desdobra. A Figura 7.12 ilustra a seqüência de operações executadas pelo autor, para copiar ARC602.EXE no disco rígido de seu computador pessoal, no diretório \ARC que ele criou. Liste a estrutura do diretório do disco rígido, desdobre ARC602.EXE e tire uma segunda listagem

do diretório para observar o efeito do desdobramento. Observe que, depois que ARC602.EXE foi copiado para o disco rígido do autor, a listagem do diretório mostra que este arquivo contém 85 527 bytes. Depois de entrar com o comando ARC602, o programa se desdobra ou se divide em quatro arquivos: ARC.DOC, ARC.EXE, ARC.TXT e ARCORDER.TXT. Observe que a segunda listagem do diretório indica que estes quatro arquivos requerem 136.102 bytes de armazenamento. Deste modo, quando se opera com o ARC, obtém-se uma taxa de compressão de 1.59.

```
C>md\arc

C>cd\arc

C>copy d:arc602.exe c:
        1 File(s) copied

C>dir

 Volume in drive C is GILSGARBAGE
 Directory of  C:\ARC

.            <DIR>       4-23-90   5:37p
..           <DIR>       4-23-90   5:37p
ARC602   EXE    85527    6-22-89   3:39p
        3 File(s)   1986560 bytes free

C>


C>arc602
SARC  Copyright 1986-89 by Wayne Chin and Vernon D. Buerg.  /H gives help
Archive self-extractor, Version 1.10, 03/06/89. ALL RIGHTS RESERVED.

Archive:  C:\ARC\ARC602.EXE
UnCrunching > ARC.DOC
UnCrunching > ARC.EXE
UnCrunching > ARC.TXT
UnCrunching > ARCORDER.TXT
C>dir

 Volume in drive C is GILSGARBAGE
 Directory of  C:\ARC

.            <DIR>       4-23-90   5:37p
..           <DIR>       4-23-90   5:37p
ARC602   EXE    85527    6-22-89   3:39p
ARC      DOC    64321    6-22-89   3:33p
ARC      EXE    65339    3-13-89  10:30a
ARC      TXT     2299    3-14-89   9:57a
ARCORDER TXT     4143    6-22-89   3:39p
      7 File(s)   1845248 bytes free

C>
```

Figura 7.12 - Instalando o ARC602.EXE.

Para avaliar a eficiência de compressão do ARC, os leitores devem observar que ARC.EXE é um arquivo executável, enquanto ARC.DOC, ARC.TXT e ARCORDER.TXT são arquivos ASCII.

Os leitores devem, também, observar que ARC é um programa de propriedade registrada da empresa System Enhancement Associates, Inc., que concedeu permissão ao autor para incluir o ARC e diversos programas associados no disco de conveniência, que você pode obter na John Wiley & Sons. Os compradores do disco de conveniência recebem uma Licença Limitada para usar ARC, copiá-lo e distribuí-lo, desde que as seguintes condições sejam cumpridas:

1. Nenhuma taxa pode ser cobrada pela cópia ou distribuição.
2. O ARC só pode ser distribuído em seu estado original, inalterado.
3. O ARC não pode ser distribuído, no todo ou em parte, como parte de qualquer produto comercial ou serviço, sem a expressa permissão escrita da empresa System Enhancement Associates, Inc.

As contribuições para o uso deste programa serão bem-vindas e devem ser enviadas para: System Enhancement Associates, Inc., 21 New Street, Wayne, NJ 07470, USA.

O disco de conveniência contém, além do ARC602.EXE, diversos programas adicionais, incluindo ARCE.COM, ARCE.DOC, ARCP.EXE, MARC.EXE, MARCP.EXE, MKSARC.EXE e PACKING.LST. O ARCP.EXE é um programa usado para retirar os arquivos do local de arquivamento de forma altamente otimizada e que pode ser mais rápida do que ao se usar o ARC.EXE. O ARCE.DOC contém a documentação do programa.

O MARC.EXE pode ser usado para incorporar locais de local de arquivamento previamente criados pelo ARC. O MKSARC.EXE é usado para criar um local de arquivamento dentro de um programa de autodesdobramento, usando o conceito semelhante àquele usado para se criar o ARC602.EXE. O PACKING.LST, como seu nome pode sugerir, é um arquivo ASCII que descreve os arquivos contidos no disco do programa ARC.

A Figura 7.13 ilustra a listagem do PACKING.LST. Os leitores devem observar que todos os arquivos listados na Figura 7.13, estão presentes no diretório \ARC do disco de conveniência.

Os leitores devem observar que todos estes programas são softwares mantidos pelo usuário, isto é, você pode copiá-los livremente e dar as cópias a quem você quiser, sem nenhum custo. Pede-se aos beneficiários destas cópias que enviem uma contribuição, caso decidam usar o programa.

O conceito de software apoiado pelo próprio usuário (geralmente chamado de 'shareware') é uma tentativa de fornecer software a baixo custo. O custo de oferecer um novo produto por meios convencionais é incerto e, conseqüentemente, desestimula muitos autores independentes e pequenas empresas a desenvolver e promover suas idéias. O software apoiado pelo próprio usuário é uma tentativa de desenvolver um novo canal de marketing, onde os produtos possam ser apresentados a baixo custo.

Se este sistema de software apoiado pelo próprio usuário funcionar, então todos se beneficiarão. O usuário se beneficiará ao receber produtos de qualidade a baixo custo e ao poder testar, minuciosamente, o software no computador, antes de comprá-lo. O autor se beneficia ao poder entrar no mercado do software comercial, sem precisar, primeiro, de grandes recursos de capital para investimento.

Mas isto só pode funcionar com o seu suporte. Não estamos apenas falando do ARC, aqui, mas de todos os softwares apoiados pelo próprio usuário. Se você receber, de um amigo ou um colega, um programa apoiado pelo próprio usuário e ainda o estiver usando, depois de algumas semanas, então é óbvio que é de alguma valia e uma contribuição deve ser enviada.

Encontra-se a seguir, uma lista do que está contido no disco do programa ARC 6.02:

Nome do arquivo	**Descrição**
ARC602.EXE	Esta é a versão 6.02 do ARC para o MS- DOS (mais a documentação) na forma de um 'local de arquivamento de autodesdobramento'. Ele se desdobra quando você o executa.
ARCE.COM	Este é um pequeno extrator de local de arquivamento de Vern Buerg e Wayne Chin, que se encontra no disco ARC, como um préstimo a nossos clientes.
ARCE.DOC	Esta é a documentação completa para o ARCE.

ARCP.EXE	Esta é uma versão de modalidade protegida do ARC para uso com o sistema operacional OS/2.
MARC.EXE	Este é uma utilidade de incorporação/ divisão de local de arquivamento. Está descrita na documentação do ARC.
MARCP.EXE	Esta é a versão de modalidade protegida do MARC para uso com o sistema operacional OS/2.
MKSARC.EXE	Este é o programa que cria locais de arquivamento de auto-extração. Está descrito na documentação ARC.
PACKING.LIST	Este é um arquivo ASCII que descreve os arquivos contidos no disco do ARC.
README	Este arquivo contém indicações de como instalar o ARC em seu sistema.
README.BAT	Este é o arquivo batch para imprimir o arquivo README.

Figura 7.13 - O conteúdo do arquivo PACKING.LST. Este arquivo lista e descreve os arquivos contidos no disco de programa ARC e que, também, se encontram no diretório \ARC no disquete de conveniência disponível na editora deste livro.

Sistema de compressão de arquivo PKWARE

Ao concluir este capítulo, examinaremos um segundo conjunto popular de programas de compressão de arquivo, desenvolvido para computadores pessoais compatíveis com MS-DOS e PC-DOS. Conhecidos, em conjunto, como PKWARE, que representa o nome da firma que comercializa este software, o sistema é composto de um arquivo principal que, quando executado, se desdobra em uma série de programas. Semelhante ao ARC, o PKWARE pode ser usado para comprimir e armazenar os arquivos juntos sob um mesmo nome de arquivo. Além disso, o PKWARE fornece aos usuários a capacidade de recriar arquivos previamente comprimidos, bem como observar as informações técnicas a respeito dos arquivos comprimidos, apagar os arquivos dentro de um arquivo comprimido, além de executar muitas funções adicionais.

Para ilustrar o uso do PKWARE, vamos examinar o conteúdo de seu disquete de distribuição e a execução de seu programa principal que 'explode' o programa em uma série de programas.

A Figura 7.14 ilustra o conteúdo do disquete de distribuição PKWARE depois de sua transferência ao disco rígido do autor. Observe que o arquivo executável PKZ110.EXE contém 140 116 bytes. O arquivo rotulado README contém as instruções de instalação e não é afetado pela execução de PKZ110.

```
C>dir

 Volume in drive C is GILSGARBAGE
 Directory of  C:\TEMP

.            <DIR>       3-21-90  11:08p
..           <DIR>       3-21-90  11:08p
PKZ110   EXE   140116    3-15-90   1:10a
README           1825    3-15-90   1:10a
        4 File(s)   1560576 bytes free

C>
```

Figura 7.14 - Conteúdo do disquete PKWARE.

Ao digitar o comando PKZ110, as funções do programa como a utilidade de auto-extração, expandindo, para o formato original, uma série de arquivos previamente comprimidos. A Figura 7.15 ilustra a auto-extração, executada por PKZ110, resultando na formação de 13 arquivos.

Ao obter uma listagem do diretório, você pode facilmente receber, em primeira mão, as informações a respeito da capacidade do programa. A Figura contém a listagem do diretório, depois que PKZ110 foi executado. Se você somar o tamanho de cada arquivo com exceção de PKZ110 e README, você obterá um total de 292.541 bytes de armazenamento. Assim, a taxa de compressão de PKZ110, obtida ao comprimir diversos arquivos de documento e executáveis, é 292.541 / 140.116, ou de, aproximadamente, 2.09.

O PKZIP é o programa principal de compressão, enquanto PKUNZIP é o programa principal de extração. Similar ao ARC, PKWARE fornece um programa que pode ser usado para criar arquivos de auto-extração. Este programa é rotulado ZIP2EXE.

```
C>pkz110

PKSFX (R)   FAST!   Self Extract Utility   Version 1.1.  03-15-90
Copr. 1989-1990 PKWARE Inc. All Rights Reserved. PKSFX/h for help
PKSFX Reg. U.S. Pat. and Tm. Off.

Searching EXE: C:/TEMP/PKZ110.EXE - Thanks for using PKWARE!
  Exploding: WHATSNEW.110
  Exploding: README.DOC
  Exploding: MANUAL.DOC
  Exploding: ADDENDUM.DOC
  Exploding: DEDICATE.DOC
  Exploding: LICENSE.DOC
  Exploding: ORDER.DOC
  Exploding: APPNOTE.TXT
  Exploding: OMBUDSMN.ASP
  Exploding: PKZIP.EXE
  Exploding: PKUNZIP.EXE
  Exploding: ZIP2EXE.EXE
  Exploding: PKZIPFIX.EXE

C>
```

Figura 7.15 - Processo de auto-extração do PKWARE.

```
 Directory of  C:\TEMP

.            <DIR>       5-15-90   8:08p
..           <DIR>       5-15-90   8:08p
PKZ110   EXE   140116   3-15-90   1:10a
README           1825   3-15-90   1:10a
WHATSNEW 110     2916   3-15-90   1:10a
README   DOC      534   3-15-90   1:10a
MANUAL   DOC   140355   7-21-89   1:01a
ADDENDUM DOC    21473   3-15-90   1:10a
DEDICATE DOC      720   3-15-90   1:10a
LICENSE  DOC     9366   3-15-90   1:10a
ORDER    DOC     4701   3-15-90   1:10a
APPNOTE  TXT    25811   3-15-90   1:10a
OMBUDSMN ASP      595   3-15-90   1:10a
PKZIP    EXE    32880   3-15-90   1:10a
PKUNZIP  EXE    22540   3-15-90   1:10a
ZIP2EXE  EXE    21426   3-15-90   1:10a
PKZIPFIX EXE     9224   3-15-90   1:10a
       17 File(s)  18569216 bytes free

C:\TEMP>
```

Figura 7.16 - Listagem do diretório depois do término do processo de auto-extração.

Outros arquivos contidos no PKWARE incluem README.DOC, que contém informações gerais, DEDICATE.DOC que é uma dedicação de formato e extensões do arquivo ao domínio público, ORDER.DOC que inclui informações de registro e um formulário de pedido, LICENSE.DOC que contém informações sobre licenças de distribuição, MANUAL.DOC que é o manual de referência do PKWARE, APPNOTE.TXT que contém material técnico detalhado, PKZIPFIX.EXE que é um programa que pode ser usado para recriar arquivos danificados de programa e OMBDSMAN.ASP que contém informações sobre a Associação dos Profissionais de Shareware (Association of Shareware Professionals).

Similar ao ARC, PKWARE é um programa shareware. PKWARE, Inc. como firme adepta do shareware, consentiu na distribuição de seu software em disquetes convenientes que você pode obter na editora deste livro. Se achar o software rápido, fácil e conveniente de se usar, uma inscrição de $25 seria bem-vinda. Se enviar $47 ou mais, receberá, quando disponível, um disquete com a documentação para a próxima versão. Favor mencionar a versão de software que você possui. Envie as inscrições para PKWARE, Inc., 7545 N. Port Washington Rd., Suite 205, Glendale, WI 53217-3422, USA.

Características do Programa

O PKWARE inclui diversas características que justificam atenção, incluindo a criptografia por senha, detecção automática e utilização dos computadores pessoais baseados no microprocessador 80386 da Intel e a capacidade de exibir um nível significante de informações sobre arquivos comprimidos.

A característica de criptografia por senha possibilita aos usuários embaralharem arquivos de dados sensíveis, enquanto a detecção automática e utilização de microprocessadores 80386, capacita o programa para usar instruções de 32 bits e modos de endereçamento estendidos para ter o desempenho melhorado, quando a CPU estiver disponível para o uso. Uma das melhores características do programa é obtida pelo uso de seu comando observar (v), que exibe as informações técnicas de diversos modos, com base na especificação de um sufixo opcional adicionado à opção v. A Tabela 7.1 indica as informações básicas a respeito de cada arquivo que será listado através do uso de um comando v.

Tabela 7.1 - Informações do arquivo PKWARE apresentadas ao usar o comando v

Comprimento	Comprimento original do arquivo
Método	Tipo de compressão de arquivo usada
Tamanho	Tamanho do arquivo comprimido
Taxa	Redução em porcentagem no tamanho do arquivo
Data	Data real do arquivo
Hora	Hora real do arquivo
CRC-32	O valor CRC-32 do arquivo
Atributo	O atributo do arquivo (s = Sistema, h = escondido, w = de gravação, r = arquivo somente de leitura e * = criptografado)
Nome	Nome do arquivo

Ao examinar as entradas na Tabela 7.1, o leitor deve observar que as funções CRC-32, como teste de integridade de dados que acompanha cada arquivo, quando comprimido, e que é testado pelo programa, quando se usa o recurso de explosão. O PKZIP e o PKUNZIP seguem um formato de comando de execução semelhante e são, essencialmente, muito fáceis de se usar. Por exemplo, para comprimir arquivos, usando PKZIP você entraria com um comando, observando-se a seguinte sintaxe de comando:

PKZIP ZIPFILE [options] [files]

onde: *PKZIP* é o nome do programa do arquivo executável que comprime arquivos de dados.

ZIPFILE representa o nome que você atribui ao arquivo comprimido resultante.

opções representam comandos que efetivam a operação de PKZIP, como, por exemplo, -a para adicionar arquivos.

*Arquivos r*epresentam os arquivos a serem comprimidos no ZIFFILE.

Embora não fosse possível fazer um exame do uso das opções PKWARE, o leitor pode consultar os disquetes convenientes que contêm os programas de compressão de arquivo ARC e PKWARE, bem como, outros programas abordados neste livro. Ao experimentar cada programa, você deve selecionar aquele que melhor se adeqüe aos tipos de arquivos que você usa no seu computador pessoal. Ao fazer isto, favor lembrar-se que todos estes programas são programas 'shareware' e que as empresas dependem de usuários satisfeitos, para que estes enviem a taxa apropriada, se pretenderem usar o software.

APÊNDICE A

DESCRIÇÃO E LISTAGENS DO PROGRAMA DATANALYSIS

A.1 PROGRAMA FORTRAN: DESCRIÇÃO OPERACIONAL

O programa original de análise de compressão de dados foi escrito no padrão FORTRAN IV para ser executado no sistema de computação Honeywell 66/80. Ele é composto de uma rotina principal que realiza todo o tratamento de arquivo. Doze sub-programas independentes são chamados pela rotina principal para executar funções específicas, tais como: análise de cadeia, classificação e formatação de relatório.

Nesta documentação, estão incluídas descrições narrativas de cada subprograma. O leitor deve consultar a listagem do programa ao ler a descrição operacional da rotina principal, subprogramas e a discussão a respeito da transferabilidade do programa e as atribuições às variáveis.

A.2 ROTINA PRINCIPAL

A rotina constrói primeiramente, dois arranjos simbólicos preenchidos com aqueles caracteres associados aos contadores de freqüência. O primeiro arranjo contém os 127 caracteres ASCII seguidos por um símbolo sumário e um espaço de trabalho temporário, usado enquanto se classifica o arranjo. O segundo arranjo contém todos os pares de caracteres ASCII geralmente encontra-

dos. O arranjo, embora estruturado em uma dimensão, pode ser visto como bidimensional diretamente proporcional ao arranjo de números inteiros em pares IP(28, 26). A declaração DATA foi escolhida para minimizar o I/O externo.

A rotina então faz três perguntas antes de realizar a análise real:

ENTER FILE TYPE - Até sete caracteres que identificam o arquivo a ser analisado. Isto é exibido em cada relatório para futura referência.

ENTER INPUT LINELENGTH - Até 132 caracteres em 'cadeias' de dados a serem lidas. Um arquivo tipo cartão seria composto de 80 vezes um arquivo de impressão de 132. Isto preserva o tempo de execução ao reduzir iterações desnecessárias. As cadeias maiores podem ser acomodadas ao se aumentar as dimensões.

TO SUPPRESS TRAILING SPACES ENTER 1 - Isto é útil para comprimentos variáveis de linha, tais como um programa em BASIC digitado em um terminal. Os espaços em branco, que vêm após o último caractere imprimível, são ignorados. Qualquer outra entrada fará com que o comprimento fixo de linha seja usado.

Um registro ou uma cadeia é, então, lido. Cada caractere é convertido em seu valor decimal ordinal e o valor é usado para indexação dentro do arranjo de contagens. Antes do próximo registro ser lido, três subprogramas são invocados e a cadeia alfanumérica é passada para cada subprograma para análise adicional. Estes subprogramas são: SUB-1 para solucionar cadeias puramente numéricas, SUB-7 para solucionar cadeias repetitivas e SUB-9 para solucionar combinações de pares.

Os subprogramas remanescentes são invocados ao encontrar o fim-de-arquivo para classificar e imprimir os oito relatórios.

Subprograma 1

Este subprograma calcula as cadeias numéricas seqüenciais, que são suscetíveis à codificação de meio-byte. Os seguintes caracteres especiais, normalmente encontrados em programas financeiros, são, também, incluídos como parte da cadeia:

$044_8 = \$ = 36_{10}$

$052_8 = * = 42_{10}$

$054_8 = , = 44_{10}$

$055_8 = - = 45_{10}$

$056_8 = . = 46_{10}$

$057_8 = / = 47_{10}$

O empacotamento numérico de meio-byte permitiria a compressão de:

$***1,234.56

12/12/80-2/11/81

026-36-1048

800-555-1212

45, 675, 109, 210.86

1941/1945/1952-1954

Subprograma 2

Este subprograma imprime o resumo do relatório de execução.

Subprograma 3

Este subprograma imprime a tabela da freqüência de ocorrência, primeiro colocando em seqüência ordinal, depois classificando pelo SUB-6, pela freqüência decrescente de ocorrência para facilidade de análise. O arranjo é colocado em uma tabela composta de 4 colunas e 32 linhas para caber em uma única página.

Subprograma 4

Este subprograma imprime o relatório de análise da cadeia de caracteres repetidos. Somente as cadeias de caracteres idênticos são incluídas. Os comprimentos nulos de cadeias são omitidos para preservar a impressão.

Subprograma 5

Este subprograma imprime uma análise adicional, do que foi exibido pelo SUB-4, onde ele gera economias reais alcançadas pelo emprego de técnicas de codificação para compressão de dados. Por exemplo, $4 \leq R \leq 256$, quando comprimido como:

CARACTERE ESPECIAL	CONTAGEM	CARACTERES COMPRIMIDO

Subprograma 6

Este subprograma executa uma classificação paralela interna da matriz de freqüência, com seu símbolo ASCII respectivo, em ordem decrescente de freqüência. Em outras palavras, para cada movimento do valor IR(N), há um correspondente do símbolo B(N).

Subprograma 7

Este subprograma calcula os caracteres repetidos seqüencialmente para os exibidos pelo SUB-4 e SUB-5. Três matrizes separadas são mantidas: IY(N) para cada ocorrência, IN(N) para numéricos $48_{10} \leq N \leq 57_{10}$ e IX(N) para espaços 32_{10}.

Subprograma 8

Este subprograma imprime a análise da cadeia numérica seqüencial para a susceptibilidade à codificação de meio-byte. Este subprograma usa a matriz IM(N) gerado pelo SUB-1. Ele usa o algoritmo para extrair os caracteres salvos pela codificação de meio-byte.

$$\left[\left[\frac{\textit{Tamanho da Cadeia} + 1}{2}\right] + 2\right] * \textit{Contagem}$$

Subprograma 9

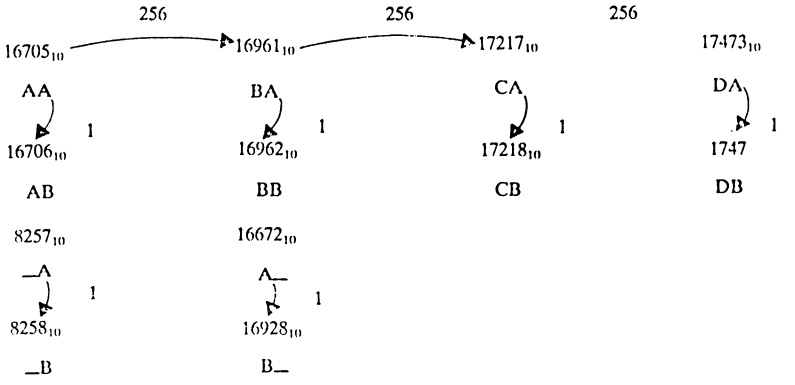
Este subprograma isola pares de caracteres e calcula sua freqüência de ocorrência. Ele começa com o primeiro par na cadeia e continua, incrementando de dois, até que a cadeia chegue ao fim. Nenhuma tentativa é feita para pular os limites dos pares.

Uma matriz bidimensional (28 x 26) é mantida para contar as seguintes combinações de pares, onde o sublinhado simboliza um caractere em branco:

_A	A_	AA	ZA
_B	B_	AB	ZB
_C	C_	AC	ZC
.			
.	.	.	.
_Z	Z_	AZ	ZZ

Uma contagem adicional é mantida para os pares de espaços ou nulos embutidos; por exemplo, espaços em branco, cercados por vazios, não são, deste modo, suscetíveis a outras formas de codificação. Para manter uma matriz simétrica, que é classificada pela SUB-11, a Q-cell é usada, já que este par nunca é encontrado na língua inglesa.

Os valores decimais iniciais são estabelecidos para representar pares e loops DO são executados da seguinte maneira:



O índice J é usado no programa para estabelecer valores iniciais. Devido a um 'bug' inexplicável no sistema usado para o desenvolvimento, nos valores de partida tinham de ser adicionados o valor 16. Os valores nos cartões 4750, 4860, 4990 e 5070 devem ser mudados, para os valores acima, para operar perfeitamente em um sistema diferente.

Subprograma 10

Este subprograma imprime o relatório de análise dos caracteres em pares depois que o SUB-11 classifica a matriz. Uma página é necessária para exibir as 132 freqüências mais altas de ocorrência. A matriz é colocada em 22 linhas com 6 colunas cada, para facilitar a referência.

O resumo final da contagem das combinações é exibido no rodapé da página, que pode ser usado para extrair a contagem daquelas combinações adicionais, além dos limites da tabela.

Subprograma 11

Este subprograma desenvolve uma classificação paralela interna da matriz de freqüência de pares IP (N1,N2) com a matriz de caracteres C(N) em ordem decrescente de freqüência. Para facilitar a técnica usada no SUB-6, uma conversão linear é executada para dar ao IP(N1, N2) a aparência de uma única dimensão correspondente à C(N).

Subprograma 12

Este subprograma computa a entropia do arquivo e gera o relatório final:

$$\text{Entropia} = \sum_{i=1}^{M} P_i \log_2 P_i$$

onde P_i corresponde à probabilidade de ocorrência de caracteres no arquivo de dados: por exemplo, se E ocorre 100 vezes em um arquivo de 1000 caracteres

$$P_i(E) = \frac{100}{1000} = 0.1$$

A.3 TRANSFERIBILIDADE DO PROGRAMA

Diversas instruções são sensíveis ao hardware e, por sua natureza, não são diretamente transferíveis entre hardwares orientados à word. O compilador FORTRAN deve ser examinado e, se necessário, três ou quatro instruções devem ser modificadas.

Por exemplo, o programa extrai o valor decimal ordinal de cada caractere que entra e usa-o como um indexador da matriz para incrementar as contagens de ocorrência. O compilador Honeywell exige a função de deslocamentos internos à direita - integer-right-shift (IRS) para alinhar os caracteres ASCII justificados à esquerda para a posição adequada de dupla word de 36 bits.

Por exemplo, o cartão 1380 alinha o dado da seguinte maneira (dado = A = 65):

	0	8 9	17 18	26 27	35
Antes:	65	0	0	0	

IRS (A(I),27)

	0	8 9	17 18	26 27	35
Depois	0	0	0	65	

Para operar no DECSYSTEM 10, as seguintes instruções específicas (cartões) devem ser alteradas como indicado:

CONVERTION TO DECSYSTEM 10

Mude as instruções específicas (cartões) como a seguir:

(1) Mude o cartão 1310 para: 2 FORMAT(132R1)
(2) Mude o cartão 1380 para: IP2 = A(I)
(3) Mude o cartão 4720 para: IV = (A(I)*256)
(4) Mude o cartão 4730 para: IV = IV + A(I + 1)
(5) Mude o cartão 4750 para: IF (IV - 8224)923, 920, 923
(6) Mude o cartão 4860 para: J = 8257
(7) Mude o cartão 4990 para: J = 16672
(8) Mude o cartão 5070 para: J = 16705
(9) Mude o cartão 5960 para: 1200 E = E + P(I)*ALOG(P(I))

Observação: As funções genéricas ALOG e ALOG 10 são discutidas superficialmente nas 10 publicações do DECSYSTEM. A função específica desejada é um log na base dois. Se o ALOG não fornecer isto de forma apropriada, substitua o cartão 5960, como a seguir:

$$\sum P_i \frac{\log_{10} P_i}{\log_{10} 2} \quad \textit{em vez de} \quad \sum P_i \log_2 P_i$$

A.4 ATRIBUIÇÕES ÀS VARIÁVEIS

Nome	Tipo	Propósito	Uso	I = Endógeno X = Exógeno
A(132)	DIM	Registro ASCII	Principal	X
B(129)	DIM	Símbolos gráficos	SUB 3,6	I
IA (132)	DIM	Linha de caracteres ASCII em notação octal	Principal; SUB1,2	I
IN(133)	DIM	Cadeias numéricas	SUB1,4	I
IX(133)	DIM	Cadeias em branco	SUB2,5	I
IY(133)	DIM	Todas as cadeias	SUB7,8	I
Z( )	DIM			
NI	INT	Comprimento de linha para busca	Principal; SUB1,2,4,5	X
X	Carac.	Nome do arquivo	SUB3,4,5	X
JI	INT	Contagem de registro	Principal; SUB4,5	I
I	INT	Índice	Principal; SUB1,2,3, 4,5,6	I
J	INT	Índice		
K	INT	Índice	SUB1,2,3 6	I
M	INT	Flag	SUB3	I
P(128)	Dupla Prec.	Porcentagem de ocorrência	SUB3,4,5	I
IR(128)	Dupla Prec.	Contagem de ocorrência	Principal; SUB3,4,5 6	I
IP1	INT	Equivalente octal do caractere	Principal	I

A.4 (continuação)

Nome	Tipo	Propósito	Uso	I = Endógeno X = Exógeno
L	INT	Índice	Principal	
IPI	INT	Índice de classificação	SUB6	I
IV	INT	Flag de supressão de espaços posteriores	Principal	X
N	INT	Comprimento de linha ajustado	Principal; SUB1,2,4 5	I
IM(133)	DIM	Números misturados com caracteres específicos	SUB8,1	I
NC	REAL	Contador de espaços posteriores	Principal	I
IC & ICI	INT	Contagem de caractere de compressão de meio byte	SUB8	I
ITN_X, ITX, ITY_X, ITT_X, ITS_X	INT	Células totais	SUB5	I
E	INT	Cálculo de entropia	SUB12	I
BI	INT	Bits depois da compressão	SUB12	I
B4	INT	Caracteres depois da compressão	SUB12	I
B2	INT	Caracteres antes da compressão	SUB12	I
B3	INT	Compressão teórica máxima	SUB12	I

A.5 PROGRAMA EM BASIC: DESCRIÇÃO OPERACIONAL

O programa DATANALYSIS original foi escrito em 1980, para operação em sistemas de computação de larga escala. Embora o uso do FORTRAN IV permitia que o programa fosse facilmente transportado para vários computadores de larga escala, sua transferabilidade excluía muitos minicomputadores e microcomputadores populares por diversas razões, inclusive a falta de um compilador FORTRAN econômico. Com base no grande número de pedidos recebidos dos leitores da primeira edição deste livro, começou em 1985, o trabalho de reescrever o programa em Microsoft BASIC, para se ajustar ao sempre crescente público de computadores compatíveis com o MS-DOS e o PC-DOS.

Para encurtar o tempo de execução do programa em BASIC, que era muitas vezes maior que o obtido pelo uso do programa em FORTRAN no sistema de computação de grande porte, foram necessárias extensivas revisões em diversas sub-rotinas. A classificação da combinação de par bidimensional, por exemplo, foi totalmente inaceitável do ponto de vista do tempo de execução de perspectiva de tempo e foi, consideravelmente, abreviada no BASIC para colunas seqüenciais somente em ordem decrescente de ocorrência. A versão interpretada do programa em BASIC, chamado DATANAL.BAS está listada no final deste apêndice. Infelizmente, mesmo reduzindo a classificação da combinação de par bidimensional, este ainda requer, aproximadamente, 15-25 min para completar a análise de um arquivo contendo, aproximadamente, 20.000 caracteres. Para reduzir, substancialmente, o tempo de execução exigido pela versão BASIC do DATANALYSIS será necessário que o leitor obtenha a versão compilada do programa. Os leitores que obtiverem o disco de conveniência oferecido com esta edição do livro, encontrarão o arquivo do programa chamado de DATANAL.EXE. Este é um arquivo de programa executável que foi gerado pelo uso do compilador Microsoft Quick BASIC e pode operar em qualquer sistema com 256KB compatível com o MS-DOS. Outros arquivos incluídos no disco, acima citado, são: BRUNIO.LIB, que é a biblioteca de módulo de execução do Quick BASIC; BRUNIO.EXE, que é o módulo de execução do Quick BASIC; DATANAL.OBJ que é a versão objeto do programa DATANALYSIS; DATANAL.LST, que é a listagem do arquivo fonte gerado pelo processo de compilação; e DATANAL.MAP, que é o arquivo que lista todos os símbolos externos no programa e suas extensões.

A versão BASIC do DATANALYSIS foi estruturada para permitir operações em sistemas com ou sem uma impressora paralela acoplada. Qualquer arquivo salvo em ASCII pode ser analisado por este programa.

Pelo bem da eficiência do tempo de execução, use o programa como se o arquivo estivesse, na maioria das vezes, em letra minúscula. O programa poderia ser modificado para acomodar minúscula e maiúscula; contudo, isto sacrificaria, de forma significativa, tanto a velocidade de execução, quanto a memória, já que dobraria o tamanho do IP e das matrizes C$ do programa. Para os leitores com PCs baseados em 80286 e com compiladores BASIC que podem acessar mais de 384K de memória, uma expansão de programa para acomodar maiúscula e minúscula pode valer bem a pena. Como o leitor poderá observar, olhando a listagem do programa no fim deste apêndice, um número extensivo de comentários foi incluído para facilitar a análise da lógica do programa, bem como permitir que o programa seja modificado pelo leitor. Quaisquer melhorias, comentários ou sugestões serão muito apreciados pelo autor.

Para executar a versão BASIC compilada do DATANALYSIS, é necessário que você carregue inicialmente o MS-DOS ou o PC-DOS e, então, insira o disco de conveniência no drive A. No prompt A>, digite o comando DATANAL. Os seguintes parágrafos descreverão a rotina principal e as sub-rotinas na versão BASIC do DATANALYSIS.

A.6 ROTINA PRINCIPAL

A rotina inicialmente constrói duas matrizes simbólicas preenchidas com aqueles caracteres associados aos contadores de freqüência. A primeira matriz contém os 127 caracteres ASCII seguidos por um símbolo de resumo e um espaço temporário de trabalho, usado ao classificar a matriz. A segunda matriz contém todos os pares de caracteres ASCII normalmente encontrados. A matriz, embora estruturada em uma dimensão, pode ser vista como uma matriz bidimensional diretamente proporcional à matriz de números inteiros em pares IP(28,26). A declaração DATA foi escolhida para minimizar o I/O externo.

A rotina faz, então, três perguntas antes de executar a análise real:

ENTER ASCII FILENAME - Qualquer [DRIVE:]FILENAME EXTENSION que identifique o arquivo a ser analisado. Ele é exibido em cada relatório para futura consulta. O arquivo deve estar no formato ASCII.

IS FILE MOSTLY IN LOWER CASE - Para a análise de par, a matriz é iniciada com letra minúscula 97_{10} a 122_{10} em vez de 65_{10} para 90_{10}.

DO YOU DO YOU WANT PRINTED OUTPUT - Permite que o programa seja executado em um sistema sem uma impressora paralela conectada.

Um registro de cadeia é, então, lido. Cada caractere é convertido em seu valor decimal ordinal e o valor é usado para indexação dentro da matriz de contagens. Antes do próximo registro ser lido, são invocados três subprogramas e a cadeia alfanumérica é passada para cada subprograma, para análise adicional. Estes subprogramas são: SUB-1 para solucionar cadeias puramente numéricas, SUB-7 para solucionar cadeias repetitivas e SUB-9 para solucionar combinações de pares.

Os subprogramas remanescentes são invocados ao encontrar o fim-de-arquivo para classificar e imprimir os oito relatórios.

Subprograma 1

Este subprograma calcula cadeias numéricas seqüenciais, que são suscetíveis à codificação de meio byte. Os seguintes caracteres especiais, normalmente encontrados em programas financeiros, são, também, incluídos como parte da cadeia:

$ = 36_{10}
* = 42_{10}
, = 44_{10}
- = 45_{10}
. = 46_{10}
/ = 47_{10}

O empacotamento numérico de meio-byte permitiria a compressão de:

$***1,234.56
12/12/80-2/11/81
026-36-1048
800-555-1212
45, 675, 109, 210.86
1941/1945/1952-1954

Subprograma 2

Este subprograma imprime o resumo do relatório de execução.

Subprograma 3

Este subprograma imprime a tabela de freqüência de ocorrência primeiro colocando em seqüência ordinal, depois classificando pelo SUB-6, pela freqüência decrescente de ocorrência para facilitar a análise. O arranjo é colocado em uma tabela composta de 4 colunas e 32 linhas para caber em uma única página.

Subprograma 4

Este subprograma imprime o relatório de análise da cadeia de caracteres repetidos. Somente as cadeias de caracteres idênticos são incluídas. Os comprimentos nulos de cadeia são omitidos para preservar a impressão.

Subprograma 5

Este subprograma imprime uma análise adicional do que foi exibido pelo SUB-4, no qual ele gera economias reais alcançadas pelo emprego de técnicas de codificação para compressão de dados. Por exemplo, $4 \leq R \leq 256$, quando comprimido como:

CARACTERE ESPECIAL	CONTAGEM	CARACTERE COMPRIMIDO

Subprograma 6

Este subprograma executa uma classificação paralela interna à matriz de freqüência, com seu símbolo ASCII respectivo, em ordem decrescente de freqüência. Em outras palavras, para cada movimento do valor IR(N), há um correspondente do símbolo B$(N).

Subprograma 7

Este subprograma calcula os caracteres repetidos seqüencialmente para os exibidos pelo SUB-4 e SUB-5. Três matrizes separadas são mantidas: IY(N) para qualquer ocorrência, IN(N) para numéricos $48_{10} \leq N \leq 57_{10}$ e IX(N) para espaços 32_{10}.

Subprograma 8

Este subprograma imprime a análise da cadeia numérica seqüencial para a susceptibilidade à codificação de meio byte. Ele usa a matriz IM(N) gerada pelo SUB-1.

Usa também o algoritmo para extrair os caracteres salvos pela codificação de meio byte.

$$\left[\left[\frac{\textit{Tamanho da cadeia} + 1}{2}\right] + 2\right] * \textit{Contagem}$$

Subprograma 9

Este subprograma isola os pares de caracteres e calcula sua freqüência de ocorrência. Ele começa com o primeiro par na cadeia e continua, incrementando de dois, até que a cadeia chegue ao fim. Uma tentativa é feita para deslocar os limites dos pares, quando um caractere não alfabético é encontrado.

Uma matriz bidimensional (28 * 26) é mantida para contar as seguintes combinações de pares, onde o sublinhado simboliza um caractere em branco:

_A	A_	AA	ZA
_B	B_	AB	ZB
_C	C_	AC	ZC
.	.		
.	.		
.	.	.	.
_Z	Z_	AZ	ZZ

Uma contagem adicional é mantida para os pares de espaços ou nulos embutidos; por exemplo, espaços em branco cercados por vazios, desse modo não são suscetíveis a outras formas de codificação. Para manter a matriz simétrica, que é classificada pela SUB-11, a Q-cell é usada, já que este par nunca é encontrado na língua inglesa.

Subprograma 10

Este subprograma imprime o relatório de análise dos caracteres em pares, depois que o SUB-11 classifica cada coluna da matriz em ordem decrescente de ocorrência. Uma página é necessária para exibir as 132 freqüências mais altas de ocorrência. A matriz é colocada em 22 linhas com 6 colunas cada, para facilitar a referência.

O resumo final da contagem das combinações é exibido no rodapé da página, que pode ser usado para extrair a contagem daquelas combinações adicionais, além dos limites da tabela.

Subprograma 11

Este subprograma desenvolve uma classificação paralela interna da matriz de freqüência de pares IP (N1,N2) com a matriz de caracteres C(N) em ordem decrescente de freqüência. Para facilitar a técnica usada no SUB-6, uma conversão linear é executada para dar IP(N1, N2) a aparência de uma única dimensão correspondente à C(N).

Subprograma 12

Este subprograma computa a entropia do arquivo e gera o relatório final:

$$\text{Entropia} = \sum_{i=1}^{M} P_i \log_2 P_i$$

onde P_i corresponde à probabilidade de ocorrência de caracteres no arquivo de dados: por exemplo, se E ocorre 100 vezes em um arquivo de 1000 caracteres

$$P_i(E) = \frac{100}{1000} = 0.1$$

A.7 ATRIBUIÇÕES ÀS VARIÁVEIS

Nome	Tipo	Propósito	Uso	I = Endógeno X = Exógeno
A(132)	DIM	Registro ASCII	Principal	X
B(129)	DIM	Símbolos gráficos	SUB 3,6	I
IA (132)	DIM	Linha de caracteres ASCII em notação octal	Principal; SUB1,2	I
IN(133)	DIM	Cadeias numéricas	SUB1,4	I
IX(133)	DIM	Cadeias em branco	SUB2,5	I
IY(133)	DIM	Todas as cadeias	SUB7,8	I
Z( )	DIM			
NI	INT	Comprimento de linha para busca	Principal; SUB1,2,4,5	I
F$	Carac.	Nome do arquivo	SUB3,4,5	X
JI	INT	Contagem de registro	Principal; SUB4,5	I
I	INT	Índice	Principal; SUB1,2,3, 4,5,6	I
J	INT	Índice		
K	INT	Índice	SUB1,2,3,6	I
M	INT	Sort Flag	SUB3	I
P(128)	DIM	Porcentagem de ocorrência	SUB3,4,5	I
IR(128)	DIM	Contagem de ocorrência	Principal; SUB3,4,5,6	I
IP2	INT	Equivalente decimal do caractere	Principal	I
IM(133)	DIM	Números misturados com caracteres específicos	SUB8,1	I
IC & ICI	INT	Contagem de caractere de compressão de meio byte	SUB8	I
ITN_X, ITX, ITY_X, ITT_X, ITS_X	INT	Células totais	SUB5	I

A.7 (continuação)

Nome	Tipo	Propósito	Uso	I = Endógeno X = Exógeno
E	INT	Cálculo de entropia	SUB12	I
BI	INT	Bits depois da compressão	SUB12	I
B4	INT	Caracteres depois da compressão	SUB12	I
B2	INT	Caracteres antes da compressão	SUB12	I
B3	INT	Compressão teórica máxima	SUB12	I
D$	CAR	opção LPT1: ou SCRN:	Principal	X
V$	CAR	flag de letra minúscula/ maiúscula	Principal	X

A.8 LISTAGEM DO PROGRAMA DATANALYSIS EM FORTRAN

*PRINCIPAL*ROT FREQ PRINC*HELD & MARSHALL*DATANALYSIS**1980*

*ESTE PROGRAMA FOI ESCRITO PARA ANALISAR ARQUIVOS QUANTO A SUSCEPTIBILIDADE A VÁRIAS TÉCNICAS DE COMPRESSÃO. ESTÁ ESCRITO NO PADRÃO FORTRAN IV PARA RODAR EM MODO BATCH OU EM MODO DE COMPARTILHAMENTO DE TEMPO. AS INSTRUÇÕES SENSÍVEIS AO HARDWARE FORAM COLOCADAS EM DESTAQUE COM COMENTÁRIOS. VER APÊNDICE A DA DOCUMENTAÇÃO PARA A TRANSFERABILIDADE DO PROGRAMA PARA RODAR H66/80 TSS: RUN # xxxx "09" (ONDE xxxx = NOMEARQUIVO PERM)

```
C *MAIN**MAIN FREQ ROUTINE**HELD & MARSHALL****DATANALYSIS**1980*        00000100
C *THIS PROGRAM WAS WRITTEN TO ANALYZE FILES FOR SUSCEPTABILITY *        00000110
C *TO VARIOUS COMPRESSION TECHNIQUES. IT IS WRITTEN IN INDUSTRY *        00000120
C *STANDARD FORTRAN IV FOR RUNNING IN TIME-SHARING OR BATCH MODE*        00000130
C *HARDWARE SENSITIVE INSTRUCTIONS  HAVE BEEN FLAGGED BY COMMENT*        00000140
C * SEE APPENDIX A OF DOCUMENTATION FOR PROGRAM TRANSFERABILITY *        00000150
C *TO RUN H66/80 TSS: RUN # XXXX "09" (WHERE XXXX=PERMFILE NAME)         00000160
      DIMENSION A(132),B(129),IA(132),IN(133),IM(133),IX(133),IY(132)    00000170
     &,Z(132)                                                           00000180
      DIMENSION C(729),IP(28,26)                                        00000190
      DOUBLE PRECISION IR(128),P(128)                                   00000200
      CHARACTER *1 A                                                    00000210
      CHARACTER *2 B,C                                                  00000220
      CHARACTER *7 X                                                    00000230
      INTEGER OUTPUT/6/,INPUT/5/,FILIN/9/                               00000240
C   STRUCTURE GRAPHIC SYMBOL ARRAY FOR  FREQ OF OCCURRENCE TABLE        00000250
      DATA(B(L),L=1,129)/                                               00000260
     &2HSH,2HEX,2HSX,2HET,2HEQ,2HAK,2HBL,2HBS,2HHT,2HLF,                00000270
     &2HVT,2HFF,2HCR,2HSO,2HSI,2HDE,2HD1,2HD2,2HD3,2HD4,                00000280
     &2HNK,2HSY,2HEB,2HCN,2HEM,2HSB,2HEC,2HFS,2HGS,2HRS,                00000290
     &2HUS,2HSP,2HEP,2H" ,2H# ,2H$ ,2H% ,2H' ,2H( ,                     00000300
     &2H) ,2H* ,2H+ ,2H, ,2H- ,2H. ,2H/ ,2H0 ,2H1 ,2H2 ,                00000310
     &2H3 ,2H4 ,2H5 ,2H6 ,2H7 ,2H8 ,2H9 ,2H: ,2H; ,2H< ,                00000320
     &2H= ,2H> ,2HQM,2H@ ,2HA ,2HB ,2HC ,2HD ,2HE ,2HF ,                00000330
     &2HG ,2HH ,2HI ,2HJ ,2HK ,2HL ,2HM ,2HN ,2HO ,2HP ,                00000340
     &2HQ ,2HR ,2HS ,2HT ,2HU ,2HV ,2HW ,2HX ,2HY ,2HZ ,                00000350
     &2H[ ,2H\ ,2H] ,2H^ ,2H+ ,2H@ ,2HA ,2HB ,2HC ,2HD ,               00000360
     &2HE ,2HF ,2HG ,2HH ,2HI ,2HJ ,2HK ,2HL ,2HM ,2HN ,               00000370
     &2HO ,2HP ,2HQ ,2HR ,2HS ,2HT ,2HU ,2HV ,2HW ,2HX ,               00000380
     &2HY ,2HZ ,2H[ ,2H  ,2H] ,2H  ,2HDL,2H**,2H??/                    00000390
C     STRUCTURE PAIRED SYMBOL ARRAY                                     00000400
      DATA(C(J),J=1,729)/                                               00000410
     &2H+A,2H+B,2H+C,2H+D,2H+E,2H+F,2H+G,2H+H,2H+I,2H+J,                00000420
     &2H+K,2H+L,2H+M,2H+N,2H+O,2H+P,2H+Q,2H+R,2H+S,2H+T,                00000430
     &2H+U,2H+V,2H+W,2H+X,2H+Y,2H+Z,2HA+,2HB+,2HC+,2HD+,                00000440
     &2HE+,2HF+,2HG+,2HH+,2HI+,2HJ+,2HK+,2HL+,2HM+,2HN+,                00000450
     &2HO+,2HP+,2H++,2HR+,2HS+,2HT+,2HU+,2HV+,2HW+,2HX+,                00000460
     &2HY+,2HZ+,2HAA,2HAB,2HAC,2HAD,2HAE,2HAF,2HAG,2HAH,                00000470
     &2HAI,2HAJ,2HAK,2HAL,2HAM,2HAN,2HAO,2HAP,2HAQ,2HAR,                00000480
     &2HAS,2HAT,2HAU,2HAV,2HAW,2HAX,2HAY,2HAZ,2HBA,2HBB,                00000490
     &2HBC,2HBD,2HBE,2HBF,2HBG,2HBH,2HBI,2HBJ,2HBK,2HBL,                00000500
     &2HBM,2HBN,2HBO,2HBP,2HBQ,2HBR,2HBS,2HBT,2HBU,2HBV,                00000510
     &2HBW,2HBX,2HBY,2HBZ,2HCA,2HCB,2HCC,2HCD,2HCE,2HCF,2HCG,           00000520
     &2HCH,2HCI,2HCJ,2HCK,2HCL,2HCM,2HCN,2HCO,2HCP,2HCQ,                00000530
     &2HCR,2HCS,2HCT,2HCU,2HCV,2HCW,2HCX,2HCY,2HCZ,2HDA,                00000540
     &2HDB,2HDC,2HDD,2HDE,2HDF,2HDG,2HDH,2HDI,2HDJ,2HDK,                00000550
     &2HDL,2HDM,2HDN,2HDO,2HDP,2HDQ,2HDR,2HDS,2HDT,2HDU,                00000560
     &2HDV,2HDW,2HDX,2HDY,2HDZ,2HEA,2HEB,2HEC,2HED,2HEE,                00000570
     &2HEF,2HEG,2HEH,2HEI,2HEJ,2HEK,2HEL,2HEM,2HEN,2HEO,                00000580
     &2HEP,2HEQ,2HER,2HES,2HET,2HEU,2HEV,2HEW,2HEX,2HEY,                00000590
     &2HEZ,2HFA,2HFB,2HFC,2HFD,2HFE,2HFF,2HFG,2HFH,2HFI,                00000600
     &2HFJ,2HFK,2HFL,2HFM,2HFN,2HFO,2HFP,2HFQ,2HFR,2HFS,                00000610
     &2HFT,2HFU,2HFV,2HFW,2HFX,2HFY,2HFZ,2HGA,2HGB,2HGC                 00000620
     &2HGD,2HGE,2HGF,2HGG,2HGH,2HGI,2HGJ,2HGK,2HGL,2HGM                 00000630
     &2HGN,2HGO,2HGP,2HGQ,2HGR,2HGS,2HGT,2HGU,2HGV,2HGW,                00000640
     &2HGX,2HGY,2HGZ,2HHA,2HHB,2HHC,2HHD,2HHE,2HHF,2HHG,                00000650
     &2HHH,2HHI,2HHJ,2HHK,2HHL,2HHM,2HHN,2HHO,2HHP,2HHQ,                00000660
```

```
     &2HHR,2HHS,2HHT,2HHU,2HHV,2HHW,2HHX,2HHY,2HHZ,2HIA,                00000670
     &2HIB,2HIC,2HID,2HIE,2HIF,2HIG,2HIH,2HII,2HIJ,2HIK,                00000680
     &2HIL,2HIM,2HIN,2HIO,2HIP,2HIQ,2HIR,2HIS,2HIT,2HIU,                00000690
     &2HIV,2HIW,2HIX,2HIY,2HIZ,2HJA,2HJB,2HJC,2HJD,2HJE,                00000700
     &2HJF,2HJG,2HJH,2HJI,2HJJ,2HJK,2HJL,2HJM,2HJN,2HJO,                00000710
     &2HJP,2HJQ,2HJR,2HJS,2HJT,2HJU,2HJV,2HJW,2HJX,2HJY,                00000720
     &2HJZ,2HKA,2HKB,2HKC,2HKD,2HKE,2HKF,2HKG,2HKH,2HKI,                00000730
     &2HKJ,2HKK,2HKL,2HKM,2HKN,2HKO,2HKP,2HKQ,2HKR,2HKS,                00000740
     &2HKT,2HKU,2HKV,2HKW,2HKX,2HKY,2HKZ,2HLA,2HLB,2HLC,                00000750
     &2HLD,2HLE,2HLF,2HLG,2HLH,2HLI,2HLJ,2HLK,2HLL,2HLM,                00000760
     &2HLN,2HLO,2HLP,2HLQ,2HLR,2HLS,2HLT,2HLU,2HLV,2HLW,                00000770
     &2HLX,2HLY,2HLZ,2HMA,2HMB,2HMC,2HMD,2HME,2HMF,2HMG,                00000780
     &2HMH,2HMI,2HMJ,2HMK,2HML,2HMM,2HMN,2HMO,2HMP,2HMQ,                00000790
     &2HMR,2HMS,2HMT,2HMU,2HMV,2HMW,2HMX,2HMY,2HMZ,2HNA,                00000800
     &2HNB,2HNC,2HND,2HNE,2HNF,2HNG,2HNH,2HNI,2HNJ,2HNK,                00000810
     &2HNL,2HNM,2HNN,2HNO,2HNP,2HNQ,2HNR,2HNS,2HNT,2HNU,                00000820
     &2HNV,2HNW,2HNX,2HNY,2HNZ,2HOA,2HOB,2HOC,2HOD,2HOE,                00000830
     &2HOF,2HOG,2HOH,2HOI,2HOJ,2HOK,2HOL,2HOM,2HON,2HOO,                00000840
     &2HOP,2HOQ,2HOR,2HOS,2HOT,2HOU,2HOV,2HOW,2HOX,2HOY,                00000850
     &2HOZ,2HPA,2HPB,2HPC,2HPD,2HPE,2HPF,2HPG,2HPH,2HPI,                00000860
     &2HPJ,2HPK,2HPL,2HPM,2HPN,2HPO,2HPP,2HPQ,2HPR,2HPS,                00000870
     &2HPT,2HPU,2HPV,2HPW,2HPX,2HPY,2HPZ,2HQA,2HQB,2HQC,                00000880
     &2HQD,2HQE,2HQF,2HQG,2HQH,2HQI,2HQJ,2HQK,2HQL,2HQM,                00000890
     &2HQN,2HQO,2HQP,2HQQ,2HQR,2HQS,2HQT,2HQU,2HQV,2HQW,                00000900
     &2HQX,2HQY,2HQZ,2HRA,2HRB,2HRC,2HRD,2HRE,2HRF,2HRG,                00000910
     &2HRH,2HRI,2HRJ,2HRK,2HRL,2HRM,2HRN,2HRO,2HRP,2HRQ,                00000920
     &2HRR,2HRS,2HRT,2HRU,2HRV,2HRW,2HRX,2HRY,2HRZ,2HSA,                00000930
     &2HSB,2HSC,2HSD,2HSE,2HSF,2HSG,2HSH,2HSI,2HSJ,2HSK,                00000940
     &2HSL,2HSM,2HSN,2HSO,2HSP,2HSQ,2HSR,2HSS,2HST,2HSU,                00000950
     &2HSV,2HSW,2HSX,2HSY,2HSZ,2HTA,2HTB,2HTC,2HTD,2HTE,                00000960
     &2HTF,2HTG,2HTH,2HTI,2HTJ,2HTK,2HTL,2HTM,2HTN,2HTO,                00000970
     &2HTP,2HTQ,2HTR,2HTS,2HTT,2HTU,2HTV,2HTW,2HTX,2HTY,                00000980
     &2HTZ,2HUA,2HUB,2HUC,2HUD,2HUE,2HUF,2HUG,2HUH,2HUI,                00000990
     &2HUJ,2HUK,2HUL,2HUM,2HUN,2HUO,2HUP,2HUQ,2HUR,2HUS,                00001000
     &2HUT,2HUU,2HUV,2HUW,2HUX,2HUY,2HUZ,2HVA,2HVB,2HVC,                00001010
     &2HVD,2HVE,2HVF,2HVG,2HVH,2HVI,2HVJ,2HVK,2HVL,2HVM,                00001020
     &2HVN,2HVO,2HVP,2HVQ,2HVR,2HVS,2HVT,2HVU,2HVV,2HVW,                00001030
     &2HVX,2HVY,2HVZ,2HWA,2HWB,2HWC,2HWD,2HWE,2HWF,2HWG,                00001040
     &2HWH,2HWI,2HWJ,2HWK,2HWL,2HWM,2HWN,2HWO,2HWP,2HWQ,                00001050
     &2HWR,2HWS,2HWT,2HWU,2HWV,2HWW,2HWX,2HWY,2HWZ,2HXA,                00001060
     &2HXB,2HXC,2HXD,2HXE,2HXF,2HXG,2HXH,2HXI,2HXJ,2HXK,                00001070
     &2HXL,2HXM,2HXN,2HXO,2HXP,2HXQ,2HXR,2HXS,2HXT,2HXU,                00001080
     &2HXV,2HXW,2HXX,2HXY,2HXZ,2HYA,2HYB,2HYC,2HYD,2HYE,                00001090
     &2HYF,2HYG,2HYH,2HYI,2HYJ,2HYK,2HYL,2HYM,2HYN,2HYO,                00001100
     &2HYP,2HYQ,2HYR,2HYS,2HYT,2HYU,2HYV,2HYW,2HYX,2HYY,                00001110
     &2HYZ,2HZA,2HZB,2HZC,2HZD,2HZE,2HZF,2HZG,2HZH,2HZI,                00001120
     &2HZJ,2HZK,2HZL,2HZM,2HZN,2HZO,2HZP,2HZQ,2HZR,2HZS,                00001130
     &2HZT,2HZU,2HZV,2HZW,2HZX,2HZY,2HZZ,2H??/                          00001140
C      HOW MANY CHAR. TO SEARCH IN INPUT RECORD                         00001150
       WRITE(OUTPUT,23)                                                 00001160
   23  FORMAT("ENTER FILE TYPE")                                        00001170
       READ(INPUT,24)X                                                  00001180
   24  FORMAT(A7)                                                       00001190
C      READ IN FILE TYPE FOR REPORT                                     00001200
       WRITE(OUTPUT,21)                                                 00001210
   21  FORMAT("ENTER INPUT LINELENGTH")                                 00001220
       READ(INPUT,22)N1                                                 00001230
   22  FORMAT(I3)                                                       00001240
       WRITE(OUTPUT,25)                                                 00001250
   25  FORMAT("TO SUPRESS TRAILING SPACES ENTER 1")                     00001260
       READ(INPUT,26)IV                                                 00001270
   26  FORMAT(I1)                                                       00001280
C     READ A RECORD FROM DESIGNATED FILE; CHECK FOR EOF                 00001290
    1  READ(FILIN,2,END=4)(A(I),I=1,127)                                00001300
    2  FORMAT(132A1)                                                    00001310
       J1=J1+1                                                          00001320
       NC=0                                                             00001330
C     INCREMENT COUNTERS IN THE IR ARRAY BY USING ORDINAL EQUIV OF      00001340
C     THE CHARACTER AS THE SUBSCRIPT TO THE ARRAY.                      00001350
       DO 3 I=1,N1                                                      00001360
```

```
C     NOTE-FOLLOWING 2 INST HARDWARE SENSITIVE-SEE APPENDIX A.          00001370
      IP2=IRS(A(I),27)                                                  00001380
      IA(I)=IP2                                                         00001390
      IR(IP2)=IR(IP2)+1                                                 00001400
      IF (IV.EQ.1.AND.IA(I).EQ.32) GOTO 7                               00001410
      NC=0                                                              00001420
      GO TO 3                                                           00001430
    7 NC=NC+1                                                           00001440
    3 IR(128)=IR(128)+1                                                 00001450
C     SUBTRACT TRAILING SPACE COUNT FROM SUMMARY ARRAY                  00001460
      IR(32)=IR(32)-NC                                                  00001470
      IR(128)=IR(128)-NC                                                00001480
C     REDUCE LINE LENGTH BY TRAILING SPACE COUNT IF OPTIONED            00001490
      N=N1-NC                                                           00001500
      CALL SUB1(IA,IM,N)                                                00001510
      CALL SUB7(IA,IX,IY,IN,N)                                          00001520
      CALL SUB9(A,IP,N)                                                 00001530
      GOTO 1                                                            00001540
    4 CONTINUE                                                          00001550
      N=N1                                                              00001560
C     PAUSE "POSITION PAPER NOW"                                        00001570
      CALL SUB2(X)                                                      00001580
      CALL SUB3(IR,P,B,X)                                               00001590
      CALL SUB5(IR,B)                                                   00001600
      CALL SUB3(IR,P,B,X)                                               00001610
      CALL SUB4(IN,IX,IY,IR,P,X,J1,N)                                   00001620
      CALL SUB5(IN,IX,IY,IR,P,X,J1,N)                                   00001630
      CALL SUB8(IM,IR,P,X,N)                                            00001640
      CALL SUB11(IP,C)                                                  00001650
      CALL SUB10(IP,IX,B,C)                                             00001660
      CALL SUB12(IR,P,X)                                                00001670
      STOP                                                              00001680
      END                                                               00001690
C       SUB-1**CALCULATE NUMERIC STRINGS**                              00001700
      SUBROUTINE SUB1(IA,IM,N)                                          00001710
      DIMENSION IA(132),IM(133)                                         00001720
      I=1                                                               00001730
   11 IF(IA(I).GE.48.AND.IA(I).LE.57)GO TO 20                           00001740
      IF(IA(I).GE.44.AND.IA(I).LE.47)GO TO 20                           00001750
      IF(IA(I).EQ.42)GO TO 20                                           00001760
      IF('A(I).EQ.36)GO TO 20                                           00001770
   15 I=I+1                                                             00001780
      IF(I-N)11,11,40                                                   00001790
   20 K=1                                                               00001800
      I=I+1                                                             00001810
      IT=IA(I)                                                          00001820
      M=0                                                               00001830
   30 IF(IA(I).GE.48.AND.IA(I).LE.57)GO TO 25                           00001840
      IF(IA(I).GE.44.AND.IA(I).LE.47)GO TO 25                           00001850
      IF(IA(I).EQ.42)GO TO 25                                           00001860
      IF(IA(I).EQ.36)GO TO 25                                           00001870
   39 IF (M.EQ.0) GO TO 15                                              00001880
      IM(K)=IM(K)+1                                                     00001890
      GO TO 15                                                          00001900
   25 I=I+1                                                             00001910
      K=K+1                                                             00001920
C     ELIMINATE FULLY REPITITIOUS STRINGS FROM CONSIDERATION            00001930
      IF(IA(I-1).EQ.IT) GO TO 16                                        00001940
      M=1                                                               00001950
   16 IF (I-N)30,30,39                                                  00001960
   40 RETURN                                                            00001970
      END                                                               00001980
C     SUB-2**PRINT EXECUTION REPORT SUMMARY**                           00001990
      SUBROUTINE SUB2(X)                                                00002000
      INTEGER OUTPUT/6/                                                 00002010
      CHARACTER *7 X                                                    00002020
      WRITE(OUTPUT,201)                                                 00002030
  201 FORMAT(1H1,30X,"DATANALYSIS")                                     00002040
      WRITE(OUTPUT,204)                                                 00002050
      WRITE(OUTPUT,202)                                                 00002060
  202 FORMAT(1H0,24X,"EXECUTION  REPORT  SUMMARY")                      00002070
      WRITE(OUTPUT,204)                                                 00002080
      WRITE(OUTPUT,203)X                                                00002090
```

```
  203 FORMAT(1HO,24X,"FILE ANALYZED:",4X,A7)                              00002100
      WRITE(OUTPUT,204)                                                   00002110
  204 FORMAT(1HO)                                                         00002120
      WRITE(OUTPUT,205)                                                   00002130
  205 FORMAT(1HO,15X,"SEGMENT",10X,"DESCRIPTION")                         00002140
      WRITE(OUTPUT,206)                                                   00002150
  206 FORMAT(1HO,18X,"I",13X,"SYSTEM STANDARD FREQUENCY OF OCCURRENCE     00002160
     $E")                                                                 00002170
      WRITE(OUTPUT,207)                                                   00002180
  207 FORMAT(1HO,18X,"II",12X,"SORTED FREQUENCY OF OCCURRENCE")           00002190
      WRITE(OUTPUT,208)                                                   00002200
  208 FORMAT(1HO,18X,"III",11X,"REPEATED CHARACTER STRING ANALYSIS")      00002210
      WRITE(OUTPUT,209)                                                   00002220
  209 FORMAT(1HO,18X,"IV",12X,"RUN LENGTH ENCODING SUSCEPTIBILITY")       00002230
      WRITE(OUTPUT,210)                                                   00002240
  210 FORMAT(1HO,18X,"V",13X,"HALF-BYTE  ENCODING SUSCEPTIBILITY")        00002250
      WRITE(OUTPUT,211)                                                   00002260
  211 FORMAT(1HO,18X,"VI",12X,"DIATOMIC COMPRESSION ANALYSIS")            00002270
      WRITE(OUTPUT,212)                                                   00002280
  212 FORMAT(1HO,18X,"VII",11X,"ENTROPY ANALYSIS")                        00002290
      WRITE(OUTPUT,204)                                                   00002300
      RETURN                                                              00002310
      END                                                                 00002320
C     SUB-3**PRINT FREQUENCY OF OCCURRANCE TABLE**                        00002330
      SUBROUTINE SUB3(IR,P,B,X)                                           00002340
      DOUBLE PRECISION IR(128),P(128)                                     00002350
      DIMENSION B(129)                                                    00002360
      INTEGER OUTPUT/6/                                                   00002370
      CHARACTER *2 B                                                      00002380
      CHARACTER *7 X                                                      00002390
C     NOW CALCULATE PERCENTAGES                                           00002400
      DO 8 K=1,128                                                        00002410
    8 P(K)=IR(K)/IR(128)                                                  00002420
C     PRINT FULL TABLE - 4 COLUMNS/32 ROWS                                00002430
      IF (M.EQ.1) GO TO 9                                                 00002440
      WRITE(OUTPUT,301)                                                   00002450
  301 FORMAT(1H1,"SEGMENT I",11X,"SYSTEM STANDARD FREQUENCY OF            00002460
     & OCCURRANCE",6X,"DATANALYSIS")                                      00002470
      GO TO 12                                                            00002480
    9 WRITE(OUTPUT,302)                                                   00002490
  302 FORMAT(1H1,"SEGMENT II",11X,"SORTED FREQUENCY OF OCCUR              00002500
     &RENCE",14X,"DATANALYSIS")                                           00002510
   12-WRITE(OUTPUT,303)X                                                  00002520
  303 FORMAT(18X,"TABLE OF CHARACTERS FOUND IN ",A7," FILE")              00002530
      WRITE(OUTPUT,305)                                                   00002540
  305 FORMAT(1HO,"CHAR    COUNT     %    CHAR    COUNT     %             00002550
     &    CHAR    COUNT     %    CHAR    COUNT     %")                   00002560
      DO 5 I=1,32                                                         00002570
    5 WRITE(OUTPUT,6)B(I),IR(I),P(I),B(I+32),IR(I+32),P(I+32),            00002580
     &B(I+64),IR(I+64),P(I+64),B(I+96),IR(I+96),P(I+96)                   00002590
    6 FORMAT(4(A4,0PF8.0,2PF8.2))                                         00002600
      M=1                                                                 00002610
      WRITE(OUTPUT,304)                                                   00002620
  304 FORMAT(1HO,"** DENOTES TOTAL CHARACTERS IN FILE")                   00002630
      RETURN                                                              00002640
      END                                                                 00002650
C     SUB-4**PRINT REPITITIOUS STRING ANALYSIS **                         00002660
      SUBROUTINE SUB4(IN,IX,IY,IR,P,X,J1,N)                               00002670
      DOUBLE PRECISION IR(128),P(128)                                     00002680
      DIMENSION IN(133),IX(133),IY(133)                                   00002690
      INTEGER OUTPUT/6/                                                   00002700
      CHARACTER *7 X                                                      00002710
      WRITE(OUTPUT,401)                                                   00002720
  401 FORMAT(1H1,"SEGMENT III",30X,"DATANALYSIS")                         00002730
      WRITE(OUTPUT,402)X                                                  00002740
  402 FORMAT(1HO,"REPEATED CHARACTER STRING ANALYSIS FOR ",A7," FILE      00002750
     &")                                                                  00002760
      WRITE(OUTPUT,403)                                                   00002770
  403 FORMAT(1HO,"STRING *****NUMBER OF OCCURRENCES******")               00002780
      WRITE(OUTPUT,404)                                                   00002790
```

```
  404 FORMAT("LENGTH  NUMERICS SPACES  OTHER  TOTAL")                    00002800
      DO 51 I=3,N                                                       00002810
      IN(N+1)=IN(N+1)+(IN(I)*I)                                         00002820
      IX(N+1)=IX(N+1)+(IX(I)*I)                                         00002830
      IY(N+1)=IY(N+1)+(IY(I)*I)                                         00002840
      IF(IY(I).EQ.0) GO TO 51                                           00002850
   50 WRITE(OUTPUT,61)I,IN(I),IX(I),(IY(I)-IN(I)-IX(I)),IY(I)           00002860
   61 FORMAT(I4,4I8)                                                    00002870
   51 CONTINUE                                                          00002880
      WRITE(OUTPUT,41)                                                  00002890
   41 FORMAT(1HO,"**ANALYSIS SUMMARY**")                                00002900
      WRITE(OUTPUT,42)J1                                                00002910
   42 FORMAT("RECORDS PROCESSED ",I6)                                   00002920
      WRITE(OUTPUT,43)IY(N+1)                                           00002930
   43 FORMAT("TOTAL REPETITIONS",I6)                                    00002940
      WRITE(OUTPUT,71)IR(128)                                           00002950
   71 FORMAT("TOTAL CHARACTERS IN FILE",0PF8.0)                         00002960
      P(128)=IY(N+1)/IR(128)                                            00002970
      WRITE(OUTPUT,81)P(128)                                            00002980
   81 FORMAT("PERCENT REPETITIOUS",2PF8.2)                              00002990
      RETURN                                                            00003000
      END                                                               00003010
C     SUB-S**PRINT RUN-LENGTH ENCODING ANALYSIS **                      00003020
      SUBROUTINE SUBS(IN,IX,IY,IR,P,X,J1,N)                             00003030
      DOUBLE PRECISION IR(128),P(128)                                   00003040
      DIMENSION IN(133),IX(133),IY(133)                                 00003050
      INTEGER OUTPUT/6/                                                 00003060
      CHARACTER *7 X                                                    00003070
      WRITE(OUTPUT,501)                                                 00003080
  501 FORMAT(1H1,"SEGMENT IV",45X,"DATANALYSIS")                        00003090
      WRITE(OUTPUT,502)                                                 00003100
  502 FORMAT(1HO,12X,"RUN LENGTH ENCODING COMPRESSION SUSCEPTIBILITY    00003110
     &")                                                                00003120
      WRITE(OUTPUT,503)X                                                00003130
  503 FORMAT(12X,"FOR REPEATED CHARACTER STRINGS IN ",A7," FILE")       00003140
      WRITE(OUTPUT,53)                                                  00003150
   53 FORMAT(1HO,"STRING  *****REPEATED CHARACTERS*****   CHAR.        00003160
     & SAVED BY COMPRESSION")                                           00003170
      WRITE(OUTPUT,54)                                                  00003180
   54 FORMAT("LENGTH  NUMERICS SPACES  OTHER  TOTAL   NUMERICS          00003190
     & SPACES OTHER TOTAL")                                             00003200
      WRITE(OUTPUT,55)                                                  00003210
   55 FORMAT(1HO)                                                       00003220
      DO 63 I=3,N                                                       00003230
      ITN1=ITN1+(IN(I)*I)                                               00003240
      ITX1=ITX1+(IX(I)*I)                                               00003250
      ITY1=ITY1+(IY(I)*I)                                               00003260
      ITN2=(I-3)*IN(I)                                                  00003270
      ITX2=(I-3)*IX(I)                                                  00003280
      ITY2=(I-3)*(IY(I)-IX(I)-IN(I))                                    00003290
      ITT2=(I-3)*(IY(I))                                                00003300
      ITN3=ITN3+ITN2                                                    00003310
      ITX3=ITX3+ITX2                                                    00003320
      ITY3=ITY3+ITY2                                                    00003330
      ITT3=ITT3+ITT2                                                    00003340
      IF(IY(I).EQ.0) GO TO 63                                           00003350
      IYT1=(IY(I)*I)-(IN(I)*I)-(IX(I)*I)                                00003360
   52 WRITE(OUTPUT,62)I,(IN(I)*I),(IX(I)*I),IYT1,(IY(I)*I),ITN2,ITX2    00003370
     &,ITY2,                                                            00003380
     &ITT2                                                              00003390
   62 FORMAT(I4,8I8)                                                    00003400
   63 CONTINUE                                                          00003410
      ITS1=IR(128)                                                      00003420
      ITS2=(ITN1/IR(128)*100)                                           00003430
      ITS3=(ITX1/IR(128)*100)                                           00003440
      ITS4=((ITY1-ITX1-ITN1)/IR(128)*100)                               00003450
      ITS5=(ITY1/IR(128)*100)                                           00003460
      ITS6=(ITN3/IR(128)*100)                                           00003470
      ITS7=(ITX3/IR(128)*100)                                           00003480
      ITS8=(ITY3/IR(128)*100;                                           00003490
      ITS9=(ITT3/IR(128)*100)                                           00003500
      I=99999                                                           00003510
      WRITE(OUTPUT,505)                                                 00003520
```

```
  505 FORMAT(1H0)                                                       00003530
      WRITE(OUTPUT,62)I,ITN1,ITX1,(ITY1-ITN1-ITX1),ITY1,ITN3,ITX3,IT    00003540
     $Y3,ITT3                                                           00003550
      WRITE(OUTPUT,506)ITS1                                             00003560
  506 FORMAT("SUMMARY INFORMATION: TOTAL CHAR. IN DATA FILE:",I6)       00003570
      WRITE(OUTPUT,507)                                                 00003580
  507 FORMAT(" PERCENTAGE SEQUENTIAL REPEATED CHAR:")                   00003590
      WRITE(OUTPUT,508)ITS2                                             00003600
  508 FORMAT("  NUMERIC:              ",I6)                             00003610
      WRITE(OUTPUT,509)ITS3                                             00003620
  509 FORMAT("  BLANK(SPACES):        ",I6)                             00003630
      WRITE(OUTPUT,510)ITS4                                             00003640
  510 FORMAT("  NONNUMERIC NONBLANK:",I6)                               00003650
      WRITE(OUTPUT,511)ITS5                                             00003660
  511 FORMAT("  TOTAL REPEATED:       ",I6)                             00003670
      WRITE(OUTPUT,512)                                                 00003680
  512 FORMAT(1H0,"POTENTIAL COMPRESSION REDUCTION (%):")                00003690
      WRITE(OUTPUT,513)ITS6                                             00003700
  513 FORMAT("  NUMERIC:              ",I6)                             00003710
      WRITE(OUTPUT,514)ITS7                                             00003720
  514 FORMAT("  BLANK(SPACES):        ",I6)                             00003730
      WRITE(OUTPUT,515)ITS8                                             00003740
  515 FORMAT("  NONNUMERIC NONBLANK:",I6)                               00003750
      WRITE(OUTPUT,516)ITS9                                             00003760
  516 FORMAT("  TOTAL:                ",I6)                             00003770
      RETURN                                                            00003780
      END                                                               00003790
C     SUB-6**SORT SYMBOL ARRAY IN DESCENDING SEQUENCE**                 00003800
      SUBROUTINE SUB6(IR,B)                                             00003810
      DOUBLE PRECISION IR(128)                                          00003820
      DIMENSION B(129)                                                  00003830
      CHARACTER *2 B                                                    00003840
C     SORT THE FREQUENCY AND SYMBOL ARRAYS IN DESCENDING ORDER;         00003850
C     WHERE IR(N) = VALUE AND B(N) = ASSOCIATED SYMBOL.                 00003860
      DO 10 I = 1,126                                                   00003870
      IP1 = I+1                                                         00003880
      DO 10 K=IP1,126                                                   00003890
      IF(IR(I).GE.IR(K))GO TO 10                                        00003900
      TEMP = IR(I)                                                      00003910
      IR(I) = IR(K)                                                     00003920
      IR(K) = TEMP                                                      00003930
      B(129) = B(I)                                                     00003940
      B(I) = B(K)                                                       00003950
      B(K) = B(129)                                                     00003960
   10 CONTINUE                                                          00003970
      RETURN                                                            00003980
      END                                                               00003990
C     SUB-7**CALCULATE SEQUENTIAL REPEATING CHARACTERS***               00004000
      SUBROUTINE SUB7(IA,IX,IY,IN,N)                                    00004010
      DIMENSION IA(132),IX(133),IY(133),IN(133)                         00004020
C     CALCULATE SEQUENTIAL REPEATING CHARACTERS IN STRING               00004030
      I=1                                                               00004040
      K=1                                                               00004050
   73 IP=IA(I)                                                          00004060
      IF(IP.EQ.IA(I+1)) GO TO 75                                        00004070
   74 K=1                                                               00004080
      I=I+1                                                             00004090
      IF(I-N) 73,73,79                                                  00004100
   75 K=K+1                                                             00004110
      I=I+1                                                             00004120
      IF(I-N) 68,68,69                                                  00004130
   68 IF(IP.EQ.IA(I+1)) GO TO 75                                        00004140
   69 IY(K)=IY(K)+1                                                     00004150
      IF(IP.GE.48.AND.IP.LE.57) GO TO 76                                00004160
      IF(IP.EQ.32) GO TO 77                                             00004170
   78 GOTO 74                                                           00004180
   76 IN(K)=IN(K)+1                                                     00004190
      GOTO 74                                                           00004200
   77 IX(K)=IX(K)+1                                                     00004210
      GOTO 74                                                           00004220
   79 RETURN                                                            00004230
      END                                                               00004240
```

```
C       SUB-8**PRINT SEQUENTIAL NUMERIC STRING ANALYSIS**                  00004250
        SUBROUTINE SUB8(IM,IR,P,X,N)                                       00004260
        DOUBLE PRECISION IR(128),P(128)                                    00004270
        DIMENSION IM(133)                                                  00004280
        INTEGER OUTPUT/6/                                                  00004290
        CHAPACTER *7 X                                                     00004300
        WRITE(OUTPUT,801)                                                  00004310
  801   FORMAT(1H1,"SEGMENT V",50X,"DATANALYSIS")                          00004320
        WRITE(OUTPUT,802)                                                  00004330
  802   FORMAT(1HO,12X,"HALF-BYTE ENCODING COMPRESSION SUSCEPTIBILITY"     00004340
       &)                                                                  00004350
        WRITE(OUTPUT,803)X                                                 00004360
  803   FORMAT(8X,"FOR SEQUENTIAL (NON-REPEATING) NUMERICS IN ",A7," F     00004370
       &ILE")                                                              00004380
        WRITE(OUTPUT,804)                                                  00004390
  804   FORMAT(1HO,"STRING      NUMBER OF     TOTAL       COMPRESSED  CHA  00004400
       &RACTERS")                                                          00004410
        WRITE(OUTPUT,805)                                                  00004420
  805   FORMAT("LENGTH      OCCURRANCES  CHARACTERS CHARACTERS     SAVED    00004430
       &")                                                                 00004440
        DO 80 I=4,N                                                        00004450
        IM(N+1)=(IM(N+1)+IM(I)*I)                                          00004460
        IF(IM(I).EQ.0) GO TO 80                                            00004470
        IC=(((I+1)/2)+2)*IM(I)                                             00004480
        IC1=IC1+IC                                                         00004490
        WRITE(OUTPUT,82)I,IM(I),(IM(I)*I),IC,((IM(I)*I)-IC)                00004500
   82   FORMAT(I4,4I13)                                                    00004510
   80   CONTINUE                                                           00004520
        WRITE(OUTPUT,806)                                                  00004530
  806   FORMAT(1HO,"**ANALYSIS SUMMARY**")                                 00004540
        WRITE(OUTPUT,83)IR(128)                                            00004550
   83   FORMAT("TOTAL CHARACTERS IN FILE",OPF8.0)                          00004560
        WRITE(OUTPUT,807)IM(N+1)                                           00004570
  807   FORMAT("TOTAL CHAR. SUSCEPTIBLE TO COMPRESSION",I6)                00004580
        WRITE(OUTPUT,808)(IM(N+1)-IC1)                                     00004590
  808   FORMAT("TOTAL CHARACTERS SAVED  BY COMPRESSION",I6)                00004600
        P(128)=(IM(N+1)-IC1)/IR(128)                                       00004610
        WRITE(OUTPUT,84)P(128)                                             00004620
   84   FORMAT("PERCENT FILE COMPRESSION REDUCTION",2PF8.2)                00004630
        RETURN                                                             00004640
        END                                                                00004650
C       SUB-9**ISOLATE CHARACTER PAIRS IN STRINGS (DIATOMIC)**             00004660
        SUBROUTINE SUB9(A,IP,N)                                            00004670
        DIMENSION A(132),IP(28,26)                                         00004680
        I=1                                                                00004690
C       NOTE-"J" VALUES INFLATED BY 16 DUE TO HONEYWELL BUG.               00004700
C       NOTE-FOLLOWING 2 INST. HARDWARE SENSITIVE-SEE APPENDIX A.          00004710
  901   IV=IRS(A(I),19)                                                    00004720
        IV=IV+(IRS(A(I+1),27))                                             00004730
C      CHECK FOR IMBEDDED BLANK PAIRS;USE Q+ COUNTER                       00004740
        IF(IV-8240)923,920,923                                             00004750
  920   IV1=IRS(A(I+2),27)                                                 00004760
        IF(IV1.EQ.32)GO TO 923                                             00004770
        IF(I-1)922,922,921                                                 00004780
  921   IV1=IRS(A(I-1),27)                                                 00004790
        IF(IV1.EQ.32)GO TO 923                                             00004800
  922   IP(2,17)=IP(2,17)+1                                                00004810
  923   CONTINUE                                                           00004820
        IC=1                                                               00004830
        IR=0                                                               00004840
C      NOW CHECK FOR BLANK/LETTER                                          00004850
        J=8273                                                             00004860
  902   IR=IR+1                                                            00004870
        IF(IV-J)903,905,903                                                00004880
  903   IF(IR-26)904,907,907                                               00004890
  904   J=J+1                                                              00004900
        GO TO 902                                                          00004910
  905   IP(IC,IR)=IP(IC,IR)+1                                              00004920
  906   I=I+2                                                              00004930
        IF(I-N)901,901,918                                                 00004940
  907   IC=2                                                               00004950
        IR=0                                                               00004960
```

```
C     NOW CHECK FOR LETTER/BLANK                                        00004970
C     USE J=24880 FOR LOWER CASE                                        00004980
      J=16688                                                           00004990
  908 IR=IR+1                                                           00005000
      IF(IV-J)909,905,909                                               00005010
  909 IF(IR-26)910,911,911                                              00005020
  910 J=J+256                                                           00005030
      GO TO 908                                                         00005040
C     NOW CHECK FOR LETTER/LETTER                                       00005050
C     USE J=24945 FOR LOWER CASE                                        00005060
  911 J=16721                                                           00005070
  912 IR=0                                                              00005080
      IC=IC+1                                                           00005090
  913 IR=IR+1                                                           00005100
      IF(IV-J)914,905,914                                               00005110
  914 IF(IR-26)915,916,916                                              00005120
  915 J=J+1                                                             00005130
      GO TO 913                                                         00005140
  916 IF(IC-28)917,906,906                                              00005150
  917 J=J+231                                                           00005160
      GO TO 912                                                         00005170
  918 RETURN                                                            00005180
      END                                                               00005190
C     SUB-10**PRINT PAIRED CHARACTER ANALYSIS REPORT**                  00005200
      SUBROUTINE SUB10(IP,IX,B,C)                                       00005210
      DIMENSION IP(28,26),IX(133),B(129),C(729)                         00005220
      INTEGER OUTPUT/6/                                                 00005230
      CHARACTER *2 B,C                                                  00005240
      WRITE(OUTPUT,101)                                                 00005250
  101 FORMAT(1H1,"SEGMENT VI",50X,"DATANALYSIS")                        00005260
      WRITE(OUTPUT,102)                                                 00005270
  102 FORMAT(1HO,15X,"PAIRED CHARACTER COMPRESSION ANALYSIS")           00005280
      WRITE(OUTPUT,103)                                                 00005290
  103 FORMAT(1HO," PAIR/COUNT  PAIR/COUNT  PAIR/COUNT  PAIR/COUNT       00005300
     &  PAIR/COUNT PAIR/COUNT ")                                        00005310
      I=1                                                               00005320
      N=0                                                               00005330
      DO 110 J=1,28                                                     00005340
      DO 110 K=1,26                                                     00005350
      IF(IP(J,K).EQ.0) GO TO 110                                        00005360
C      PRINT ONLY THE TOP 132 PAIR OCCURRENCES                          00005370
      IF(N-22)106,106,110                                               00005380
  106 IX(I)=IP(J,K)                                                     00005390
      IPS=IPS+IX(I)                                                     00005400
      M=((J-1)*26)+K                                                    00005410
      B(I)=C(M)                                                         00005420
      I=I+1                                                             00005430
      IF(I-6)110,110,109                                                00005440
  109 WRITE(OUTPUT,105)B(1),IX(1),B(2),IX(2),B(3),IX(3),B(4),IX(4),     00005450
     &B(5),IX(5),B(6),IX(6)                                             00005460
  105 FORMAT(6(A6,I6))                                                  00005470
      I=1                                                               00005480
      N=N+1                                                             00005490
  110 CONTINUE                                                          00005500
C      FINISH OFF PRINT LINE NOW IF PARTIALLY FILLED                    00005510
      IF(I-1)112,112,114                                                00005520
  114 DO 113 I=1,6                                                      00005530
      B(I)=B(128)                                                       00005540
      IX(I)=0                                                           00005550
  113 CONTINUE                                                          00005560
  111 WRITE(OUTPUT,105)B(1),IX(1),B(2),IX(2),B(3),IX(3),B(4),IX(4),     00005570
     &B(5),IX(5),B(6),IX(6)                                             00005580
  112 WRITE(OUTPUT,115)IPS                                              00005590
  115 FORMAT(1HO,"TOTAL COMBINATIONS FOUND:",I6)                        00005600
      RETURN                                                            00005610
      END                                                               00005620
C      SUB-11**SORT  PAIRED CHARACTER ARRAY ***                         00005630
      SUBROUTINE SUB11(IP,C)                                            00005640
      DIMENSION IP(28,26),C(729)                                        00005650
      CHARACTER *2 C                                                    00005660
C      SORT THE TWO-DIMENSION PAIRS ARRAY & ASSOCIATED SYMBOLS          00005670
```

```
C       WHERE IP(N1,N2) = VALUE AND C(N) = RELATED PAIR SYMBOL.          00005680
        DO 1101 J=1,28                                                  00005690
        DO 1101 K=1,26                                                  00005700
        KK=K                                                            00005710
        DO 1101 J1=J,28                                                 00005720
        DO 1101 K1=KK,26                                                00005730
        KK=1                                                            00005740
        IF(IP(J,K).GE.IP(J1,K1)) GO TO 1101                             00005750
        TEMP=IP(J,K)                                                    00005760
        IP(J,K)=IP(J1,K1)                                               00005770
        IP(J1,K1)=TEMP                                                  00005780
C       PERFORM LINEAR CONVERSION OF TWO DIMENSIONS INTO ONE.           00005790
        M1=((J-1)*26)+K                                                 00005800
        M2=((J1-1)*26)+K1                                               00005810
        C(729)=C(M1)                                                    00005820
        C(M1)=C(M2)                                                     00005830
        C(M2)=C(729)                                                    00005840
 1101   CONTINUE                                                        00005850
        RETURN                                                          00005860
        END                                                             00005870
C       SUB-12**COMPUTE ENTROPY & PRINT ANALYSIS REPORT**               00005880
        SUBROUTINE SUB12(IR,P,X)                                        00005890
        DOUBLE PRECISION IR(128),P(128),ZERO                            00005900
        INTEGER OUTPUT/6/                                               00005910
        CHARACTER *7 X                                                  00005920
        ZERO=0.D0                                                       00005930
        DO 1201 I=1,127                                                 00005940
        IF(P(I)-ZERO)1201,1201,1200                                     00005950
 1200   E=E+P(I)*DLOG2(P(I))                                            00005960
 1201   CONTINUE                                                        00005970
        E=-1*E                                                          00005980
        B1=E*IR(128)                                                    00005990
        B2=IR(128)*8                                                    00006000
        B3=(B2-B1)/B2*100                                               00006010
        B4=B1/8                                                         00006020
        WRITE(OUTPUT,1202)                                              00006030
 1202   FORMAT(1H1,"SEGMENT VII",50X,"DATANALYSIS")                     00006040
        WRITE(OUTPUT,1203)X                                             00006050
 1203   FORMAT(1HO,15X,"ENTROPY ANALYSIS FOR ",A7," FILE")              00006060
        WRITE(OUTPUT,1204)E                                             00006070
 1204   FORMAT(1HO,10X,"ENTROPY: ",OPF8.2," BITS/CHARACTER")            00006080
        WRITE(OUTPUT,1205)                                              00006090
 1205   FORMAT(1HO,10X,"FILE CONTAINS:")                                00006100
        WRITE(OUTPUT,1206)IR(128)                                       00006110
 1206   FORMAT(1HO,15X,OPF8.0," CHARACTERS")                            00006120
        WRITE(OUTPUT,1207)B2                                            00006130
 1207   FORMAT(1HO,15X,OPF8.0," BITS")                                  00006140
        WRITE(OUTPUT,1211)B3                                            00006150
 1211   FORMAT(1HO,10X,"THEORETICAL STATISTICAL COMPRESSION: ",OPF8.2,  00006160
     &"%")                                                              00006170
        WRITE(OUTPUT,1209)                                              00006180
 1209   FORMAT(1HO,10X,"FILE WOULD REQUIRE:")                           00006190
        WRITE(OUTPUT,1210)B4                                            00006200
 1210   FORMAT(1HO,15X,OPF8.0," CHARACTERS")                            00006210
        WRITE(OUTPUT,1212)B1                                            00006220
 1212   FORMAT(1HO,15X,OPF8.0," BITS")                                  00006230
        RETURN                                                          00006240
        END                                                             00006250
```

A.9 LISTAGEM DO PROGRAMA DATANALYSIS NA LINGUAGEM BASIC

```
*****************************************************************************************
COPYRIGHT GILBERT HELD & THOMAS MARSHALL - DATANALYSIS

ESTE PROGRAMA FOI ESCRITO PARA ANALISAR ARQUIVOS QUANTO A SUSCEPTIBILIDADE A
VÁRIAS TÉCNICAS DE COMPRESSÃO. CONSULTE A DOCUMENTAÇÃO.
*****************************************************************************************

10 '*************************************************************
20 '* COPYWRITE GILBERT HELD & THOMAS MARSHALL - DATANALYSIS    *
30 '* THIS PROGRAM WAS WRITTEN TO ANALYSE/FILES FOR SUSCEPTIBILITY *
40 '* TO VARIOUS COMPRESSION TECHNIQUES.  REFER TO DOCUMENTATION.  *
50 '*************************************************************
60 '----- INITIALIZATION -----------
70 DIM A(132),B$(129),IA(132),IN(133),IM(133),IX(133),IY(132)
80 DIM Z(132),C$(729),IP(26,28),IR(128),P(128)
90 WIDTH 80:CLS            'CLEAR SCREEN & SET WIDTH TO 80
100 ON ERROR GOTO 4050     'ERROR HANDLER
110  '- - SETUP PROCESSING OF INPUT FILE  - -
120  LOCATE ,,1
130  PRINT "ENTER ASCII FILENAME IN THE FORMAT [DRIVE:]FILESPEC";
140  INPUT F$: OPEN F$ FOR INPUT AS #2
150 PRINT "IS FILE MOSTLY LOWER CASE(Y/N)";
160 INPUT V$
170 IF V$ = "Y" OR V$ = "y" THEN Z= 97 ELSE Z= 65
180  PRINT "DO YOU WANT PRINTED OUTPUT (Y/N)";
190  INPUT D$
200  IF D$= "y" OR D$= "Y" THEN D$="LPT1:" ELSE D$="SCRN:"
210  OPEN D$ FOR OUTPUT AS #1
220 ' STRUCTURE GRAPHIC SYMBOL ARRAY
230  PRINT "DATANALYSIS - INITIALIZATION..."
240 DATA SH,SX,EX,ET,EQ,AK,BL,BS,HT,LF,VT,FF,CR,SO,SI,DL,D1,D2,D3,D4
250 DATA NK,SY,EB,CN,EM,SB,EC,FS,GS,RS,US,SP
260 FOR I=1 TO 32:READ A$:B$(I)=A$:NEXT I
270 FOR I= 33 TO 127
280 B$(I) = CHR$(I):NEXT I:B$(127)="DL":B$(128)="**"
290 '*********************************************************
300 '*  STRUCTURE THE PAIRED SYMBOL ARRAY FIRST                *
310 '*********************************************************
320 ' 1ST SPACE/LETTER COMBINATIONS
330 K=1
340 I=32
350 FOR J= Z TO Z+25
360 C$(K)= CHR$(I)+ CHR$(J)
370 K=K+1
380 NEXT J
390 ' NOW LETTERS/SPACE  COMBINATIONS
400 J=32
410 FOR I=Z TO Z+25
420 C$(K)= CHR$(I)+CHR$(J)
430 K=K+1
440 NEXT I
450 ' NOW LETTER/LETTER COMBINATIONS
460 FOR I=Z TO Z+25
470 FOR J=Z TO Z+25
480 C$(K)= CHR$(I)+CHR$(J)
490 K=K+1
500 NEXT J
510 NEXT I
520 PRINT "PATIENCE - INPUT PROCESSING";
530 IF EOF(2) THEN 760
```

```
540 LINE INPUT #2, X$    'READ FULL LINE WITH PUNCTUATION
550 PRINT ".";           'TELL THEM WE'RE STILL ALIVE
560 K= LEN(X$)           'SAVE LENGTH OF INPUT STRING
570 FOR I= 1 TO K        'AND SET UP END OF STRING
580 A$= MID$(X$,I,1)     'PULL OUT A CHARACTER TO|ANALYSE
590 '*************************************************************
600 '*INCREMENT COUNTERS BY THE ORDINAL EQUIV OF CHAR AS SUBSCRIPT*
610 '*************************************************************
620 IP2=ASC(A$)          'DERIVE DECIMAL REPRESENTATION
630 IR(IP2)= IR(IP2)+1   'ADD TO DISPLACEMENT COUNTER
640 IR(128)= IR(128)+1   'ADD TO THE TOTAL CHAR READ
650 IA(I)= IP2           'STRING THE RECORD OUT IN ONE DIMENSION
660 N1= N1+1             'ADD TO THE CHARACTER COUNT
670 NEXT I               'GET NEXT INPUT CHARACTER
680 GOSUB 880            'SUB 1 - CALCULATE NUMERICS
690 GOSUB 1180           'SUB 7 - CALCULATE REPEATS
700 GOSUB 1940           'SUB 9 - ISOLATE PAIRS
710 J1=J1+1              'ADD TO RECORD COUNT
720 N=N1                 'SAVE LAST RECORD LENGTH
730 N1=0                 'ZERO LENGTH FOR NEW RECORD
740 GOTO 530             'AND GO GET MORE INPUT
750 BEEP  '******** END OF MAINLINE PROGRAM ********************
760 GOSUB 4060           'SUB 2-PRINT THE EXECUTION REPORT
770 GOSUB 1420           'SUB 3-PRINT FREQ./OCCURRENCE ANALYSIS
780 GOSUB 3880           'SUB 6-SORT FREQ. ARRAY
790 GOSUB 1420           'SUB 3-REPRINT FREQ./OCCURRENCE ANALYSIS
800 GOSUB 1690           'SUB 4-PRINT STRING ANALYSIS
810 GOSUB 2770           'SUB 5-PRINT RUNLENGTH ANALYSIS
820 GOSUB 3300           'SUB 8-PRINT NUM. STRING ANALYSIS
830 GOSUB 2540           'SUB 11-SORT PAIR ARRAY
840 GOSUB 2250           'SUB 10-PRINT PAIR ARRAY
850 GOSUB 3620           'SUB 12-COMPUTE & PRINT ENTROPY
860 END                  'T-T-T-THATS ALL FOLKS
870 '*****************************************************
880 '*        SUBROUTINE 1 - CALCULATE NUMERIC STRINGS   *
890 '*****************************************************
900 I=1
910 IF IA(I)>= 48 AND IA(I)<= 57 THEN 980
920 IF IA(I)>= 44 AND IA(I)<= 47 THEN 980
930 IF IA(I) = 42 THEN 980
940 IF IA(I) = 36 THEN 980
950 I=I+1                    'NOT A NUMERIC OR SYMBOL
960 IF I> N1 THEN 1160       'RECORD STRING PROCESS COMPLETE
970 GOTO 910
980 K=1
990 I=I+1
1000 IT=IA(I)
1010 M=0
1020 IF IA(I)>= 48 AND IA(I)<= 57 THEN 1090  'NUMERIC?
1030 IF IA(I)>= 44 AND IA(I)<= 47 THEN 1090  'SYMBOL?
1040 IF IA(I) = 42 THEN 1090                 'ASTERISK?
1050 IF IA(I) = 36 THEN 1090                 'DOLLAR?
1060 IF M=0 THEN 950
1070 IM(K)= IM(K)+1          'ADD TO STRING SIZE COUNT
1080 GOTO 950
1090 I=I+1
1100 K=K+1
1110 ' ELIMINATE FULLY REPITITIOUS STRINGS FROM CONSIDERATION
1120 IF IA(I-1)= IT THEN 1140
1130 M=1
1140 IF I> N1 THEN 1060
1150 GOTO 1020
1160 RETURN
1170 '*****************************************************
1180 '*         SUBROUTINE 7 - CALCULATE SEQ. REPEATING CHAR. *
1190 '*****************************************************
1200 I=1
1210 K=1
1220 IC= IA(I)                    'PULL CHAR. FROM STRING
```

```
1230 IF IC= IA(I+1) THEN 1280      'SAME AS NEXT IN STRING?
1240 K=1
1250 I= I+1
1260 IF I> N1 THEN 1400            'END OF RECORD
1270 GOTO 1220
1280 K=K+1                         'BUMP REPITITION COUNT
1290 I= I+1
1300 IF I> N1 THEN 1320            'END OF STRING?
1310 IF IC = IA(I+1) THEN 1280     'LOOP ON REPITIONS
1320 IY(K) = IY(K)+1               'ADD TO "ANY" COUNT
1330 IF IC>= 48 AND IC<= 57 THEN 1360    ' ITS NUMERIC
1340 IF IC = 32 THEN 1380                ' ITS A SPACE
1350 GOTO 1240
1360 IN(K) = IN(K)+1               'ADD TO REPEAT COUNT
1370 GOTO 1240
1380 IX(K)= IX(K)+1                'ADD TO SPACE COUNT
1390 GOTO 1240
1400 M=0:RETURN                    'RESET TOGGLE & RETURN
1410 '*************************************************************
1420 '*       SUBROUTINE 3 - PRINT FREQ OF OCCURRENCE TABLE       *
1430 '*************************************************************
1435 IF D$= "SCRN:" THEN WIDTH #1,255     'ELIM. DOUBLE SPACE
1440 ' 1ST CALCULATE PERCENTAGES
1450 FOR K= 1 TO 128
1460 P(K)= IR(K)/IR(128)          'IR(128) IS TOTAL CHAR STORE
1470 NEXT K
1480 ' NOW PRINT FULL TABLE - 4 COLUMNS/32 ROWS
1490 GOSUB 3580                   'SET TOP OF PAGE
1500 IF M=1 THEN 1530             'THIS TIME IT IS SORTED
1510 PRINT #1, SPC(5)"SEGMENT I";SPC(11)"STANDARD FREQ. OF 'OCCURRENCE"
1520 GOTO 1540
1530 PRINT #1,SPC(5)"SEGMENT II";SPC(11)"SORTED  FREQ. OF 'OCCURRENCE"
1540 PRINT #1,SPC(18)"TABLE OF CHARACTERS FOUND IN ";F$;" FILE"
1550 PRINT #1,
1560 PRINT #1,"CHAR   COUNT    %   CHAR  COUNT    %   CHAR  COUNT    %";
1570 PRINT #1,"   CHAR  COUNT    %"
1580 PRINT #1,
1590 P1$= " \\ ######    .##    "
1600 FOR I = 1 TO 32
1610 PRINT #1,USING P1$;B$(I);IR(I);P(I);
1620 PRINT #1,USING P1$;B$(I+32);IR(I+32);P(I+32);
1630 PRINT #1,USING P1$;B$(I+64);IR(I+64);P(I+64);
1640 PRINT #1,USING P1$;B$(I+96);IR(I+96);P(I+96)
1650 NEXT I
1660 M=1
1670 RETURN
1680 '*************************************************************
1690 '*       SUBROUTINE 4 - PRINT REPITITIOUS STRING ANALYSIS    *
1700 '*************************************************************
1710 GOSUB 3580           'SET TOP OF PAGE
1720 PRINT #1,"SEGMENT III";SPC(11)"REPEATED CHARACTER STRING ANALYSIS"
1730 PRINT #1,
1740 PRINT #1,"STRING *****NUMBER OF OCCURRENCES***"
1750 PRINT #1,"LENGTH  NUMERICS SPACES  OTHER  TOTAL"
1760 PRINT #1,
1770 P2$="####    ####    ####    ####    ####"
1780 FOR I=3 TO N      'IGNORE STRINGS LESS THAN THREE
1790 IN(N+1)=IN(N+1)+(IN(I)*I)    'ADD TO NUMERIC TOTAL
1800 IX(N+1)=IX(N+1)+(IX(I)*I)    'ADD TO SPACE TOTAL
1810 IY(N+1)=IY(N+1)+(IY(I)*I)    'ADD TO SUM TOTAL
1820 IF IY(I)=0 THEN 1840         'SKIP IF ZERO COUNT
1830 PRINT #1,USING P2$;I;IN(I);IX(I);(IY(I)-IN(I)-IX(I));IY(I)
1840 NEXT I
1850 PRINT #1,
1860 PRINT #1,"**ANALYSIS SUMMARY**"
1870 PRINT #1," RECORDS PROCESSED ";J1
1880 PRINT #1," TOTAL REPITITIONS ";IY(N+1)
1890 PRINT #1," CHARACTERS IN FILE ";IR(128)
1900 P(128)=IY(N+1)/IR(128)*100
1910 PRINT #1," PERCENT REPITITIOUS ";INT(P(128))
1920 RETURN
```

```
1930 '*************************************************************
1940 '*        SUBROUTINE 9 - ISOLATE PAIRS IN STRINGS (DIATOMIC) *
1950 '*************************************************************
1960 I=1
1970 ' CHECK FOR IMBEDED BLANK PAIRS; USE Q_ COUNTER FOR TALLY
1980 IR=17:IC=2
1990 IF IA(I) <> 32 THEN 2020
2000 IF IA(I+1) <> 32 THEN 2020
2010 GOTO 2190                      'ITS A HIT
2020 ' NOW CHECK FOR BLANK/LETTER
2030 IR=IA(I+1)-(Z-1):IC=1          'SET TO ROW N, COL 1
2040 IF IA(I) <> 32 THEN 2070
2050 IF IA(I+1) < Z OR IA(I+1) > Z+25 THEN 2070   'IS IT WITHIN RANGE?
2060 GOTO 2190                      'ITS A HIT
2070 'NOW CHECK FOR LETTER/BLANK
2080 IR=IA(I)-(Z-1):IC=2            'SET TO ROW N,COL 2
2090 IF IA(I) < Z OR IA(I) > Z+25 THEN 2120       'IS IT WITHIN RANGE?
2100 IF IA(I+1) <> 32 THEN 2120
2110 GOTO 2190                      'ITS A HIT
2120 ' NOW CHECK FOR LETTER/LETTER
2130 IC= IA(I)-(Z-3)                'SET COLUMN TO 1ST PAIR CHAR
2140 IR= IA(I+1)-(Z-1)              'SET ROW TO 2ND PAIR CHAR
2150 IF IA(I) < Z OR IA(I) > Z+25 THEN 2180     '1ST WITHIN RANGE?
2160 IF IA(I+1) < Z OR IA(I+1) > Z+25 THEN 2180 '2ND WITHIN RANGE?
2170 GOTO 2190                      'ITS A HIT
2180 I=I+1:GOTO 2210                'NO HITS, STEP JUST ONE
2190 IP(IR,IC)=IP(IR,IC)+1          'INCREMENT PAIR COUNT IN ARRAY
2200 I=I+2                          'STEP TWO FOR NEXT PAIR TEST
2210 IF I>=N1 THEN 2230
2220 GOTO 1970                      'CONTINUE TO END OF RECORD
2230 RETURN
2240 '*****************************************************
2250 '*        SUBROUTINE 10 - PRINT PAIRED CHAR. ANALYSIS    *
2260 '*****************************************************
2270 GOSUB 3580                 'SET TOP OF PAGE
2280 PRINT #1,"SEGMENT VI";SPC(11)"PAIRED CHARACTER DIATOMIC ANALYSIS"
2290 PRINT #1,
2300 PRINT #1,"PAIR/COUNT  PAIR/COUNT  PAIR/COUNT  PAIR/COUNT  PAIR/COUNT  PAIR
COUNT"
2310 PRINT #1,
2320 I=1
2330 N=0
2340 P3$="\\ #####     \\ #####     \\ #####     \\ #####     \\ #####     \\ #####'
2350 FOR J=1 TO 26
2360 FOR K=1 TO 28
2370 IF IP(J,K)< 2 THEN 2510        'SKIP 0 AND 1 COUNTS
2380 ' PRINT ONLY THE TOP 132 PAIRS COUNT
2390 IF N < 23 THEN 2410
2400 J=26:K=28:GOTO 2510            'IM DONE
2410 IX(I)= IP(J,K)
2420 IPS=IPS+IX(I)
2430 M=((K-1)*26)+ J
2440 B$(I)=C$(M)
2450 IF B$(I)= "Q " OR B$(I)="q " THEN B$(I)= "  " 'IMBEDDED BLANK PAIR
2460 I=I+1
2470 IF I <=6 THEN 2510             'SETUP SIX PAIRS TO THE LINE
2480 PRINT #1,USING P3$;B$(1);IX(1);B$(2);IX(2);B$(3);IX(3);B$(4);IX(4);B$(5);
(5);B$(6);IX(6)
2490 I=1
2500 N=N+1
2510 NEXT K
2520 NEXT J
2530 RETURN
2540 '*************************************************************
2550 '*          SUBROUTINE 11 - SORT PAIRED CHAR. ARRAY          *
2560 '*************************************************************
2570 ' SORT THE TWO DIMENSION PAIRS ARRAY (IP) & ASSOC. SYMBOLS (C)
2580 PRINT:PRINT "(NOW SORTING PAIR ARRAY)";
2590 FOR K=1 TO 28              ' SET COLUMN LOOP
2600  FOR J=1 TO 25             ' SET ROW LOOP
2610   FOR JJ=J+1 TO 26         ' SET WITHIN-COLUMN LOOP
```

```
2620     IF IP(J,K) >= IP(JJ,K) THEN 2710
2630     TMP=IP(J,K)             'TEMP SAVE THE GREATER COUNT
2640     IP(J,K)=IP(JJ,K)        'SWAP HIGHER COUNT
2650     IP(JJ,K)=TMP            'WITH LOWER COUNT
2660 M1=((K-1)*26)+J               'CALC LOWER SYMBOL ADDR
2670 M2=((K-1)*26)+JJ              'CALC HIGHER SYMBOL ADDR
2680 C$(729)=C$(M1)                'TEMP SAVE THE SYMBOL
2690 C$(M1)=C$(M2)                 'SWAP 1
2700 C$(M2)=C$(729)                'SWAP 2
2710   NEXT JJ                     'COMPLETE 3RD LOOP
2720   NEXT J                      'COMPLETE 2ND LOOP
2730   PRINT ".";                  'LET EM KNOW WE'RE ALIVE
2740  NEXT K                       'COMPLETE 1ST LOOP
2750 RETURN
2760 '*****************************************************
2770 '*  SUBROUTINE 5 - RUN LENGTH ENCODING ANALYSIS      *
2780 '*****************************************************
2790 GOSUB 3580           'SET TOP OF PAGE
2800 PRINT #1,"SEGMENT IV - RUN LENGTH ENCODING ANALYSIS"
2810 PRINT #1,
2820 PRINT #1,"STRING  ****** REPEATED CHARACTERS *****   CHAR. SAVED"
2830 PRINT #1,"LENGTH  NUMERICS SPACES  OTHER  TOTAL   NUMERICS";
2840 PRINT #1," SPACES OTHER TOTAL"
2850 PRINT #1,
2860 P4$=" ####      ####    ####    ####  #####      ####    ####  ####  ####"
2870 FOR I=3 TO N
2880 ITN1=ITN1+(IN(I)*I)          'TALLY NUMERICS
2890 ITX1=ITX1+(IX(I)*I)          'TALLY SPACES
2900 ITY1=ITY1+(IY(I)*I)          'TALLY OTHER
2910 ITN2=(I-3)*IN(I)
2920 ITX2=(I-3)*IX(I)
2930 ITY2=(I-3)*(IY(I)-IX(I)-IN(I))
2940 ITT2=(I-3)*(IY(I))
2950 ITN3=ITN3+ITN2
2960 ITX3=ITX3+ITX2
2970 ITY3=ITY3+ITY2
2980 ITT3=ITT3+ITT2
2990 IF (IY(I)=0) THEN 3020     'SKIP 0 COUNTS
3000 IYT1=(IY(I)*I)-(IN(I)*I)-(IX(I)*I)
3010 PRINT#1,USING P4$;I;(IN(I)*I);(IX(I)*I);IYT1;(IY(I)*I);ITN2;ITX2;ITY2;ITT2
3020 NEXT I
3030 ITS1=IR(128)                 'PICKUP TOTAL CHAR INPUT
3040 ITS2=(ITN1/IR(128)*100)      'COMPUTE NUMERIC %
3050 ITS3=(ITX1/IR(128)*100)      'COMPUTE SPACE %
3060 ITS4=((ITY1-ITX1-ITN1)/IR(128)*100)  'COMPUTE OTHER %
3070 ITS5=(ITY1/IR(128)*100)
3080 ITS6=(ITN3/IR(128)*100)
3090 ITS7=(ITX3/IR(128)*100)
3100 ITS8=(ITY3/IR(128)*100)
3110 ITS9=(ITT3/IR(128)*100)
3120 PRINT #1,
3130 PRINT #1,USING P4$;I;ITN1;ITX1;(ITY1-ITN1-ITX1);ITY1;ITN3;ITX3;ITY3;ITT3
3140 PRINT #1,
3150 PRINT #1,"SUMMARY INFORMATION: TOTAL CHAR READ";ITS1
3160 PRINT #1,
3170 PRINT #1,"PERCENTAGE SEQUENTIAL REPEATED CHAR:"
3180 PRINT #1,"  NUMERIC:";INT(ITS2)
3190 PRINT #1,"  BLANKS(SPACES):";INT(ITS3)
3200 PRINT #1,"  NONNUMERIC/NONBLANK:";INT(ITS4)
3210 PRINT #1,"  TOTAL REPEATED:";INT(ITS5)
3220 PRINT #1,
3230 PRINT #1,"  POTENTIAL COMPRESSION REDUCTION(%):"
3240 PRINT #1,"    NUMERIC:";INT(ITS6)
3250 PRINT #1,"    BLANK(SPACES):";INT(ITS7)
3260 PRINT #1,"    NONNUMERIC/NONBLANK:";INT(ITS8)
3270 PRINT #1," TOTAL:";INT(ITS9);"%"
3280 RETURN
```

```
3290 '*************************************************************
3300 '*      SUBROUTINE 8 - NUMERIC STRING ANALYSIS                *
3310 '*************************************************************
3320 GOSUB 3580           'SET TOP OF PAGE
3330 PRINT #1,"SEGMENT V - HALF BYTE ENCODING ANALYSIS":PRINT
3340 PRINT #1,"FOR SEQUENTIAL (NON-REPEATING) NUMERIC DATA"
3350 PRINT #1,
3360 PRINT #1,"STRING      NUMBER OF     TOTAL       COMPRESSED  CHAR"
3370 PRINT #1,"LENGTH      OCCURANCES     CHAR.         CHAR.     SAVED"
3380 PRINT #1,
3390 P5$= "  ###          ####        ####         ####         ####      ###"
3400 FOR I=4 TO N
3410 IM(N+1)=IM(N+1)+(IM(I)*I)
3420 IF (IM(I)=0) THEN 3460      'SKIP 0 COUNTS
3425 IF I> 15 THEN IC= .5 ELSE IC= 0 'EXTRA HALF-BYTE OVER 15
3430 IC= IM(I)*CINT(IC+1.5+(I/2)+.4) 'COMPUTE COMPRESSION
3440 IC1=IC1+IC
3450 PRINT #1,USING P5$;I;IM(I);(IM(I)*I);IC;((IM(I)*I)-IC)
3460 NEXT I
3470 PRINT #1,
3480 PRINT #1,"*ANALYSIS SUMMARY*"
3490 PRINT #1,"TOTAL CHAR. IN FILE=";IR(128)
3500 PRINT #1,"TOTAL CHAR. SUSCEPTIBLE TO COMPRESSION:";IM(N+1)
3510 PRINT #1,"TOTAL CHAR. SAVED BY COMPRESSION:";INT(IM(N+1)-IC1)
3520 P(128)=(IM(N+1)-IC1)/IR(128)*100   'COMPUTE PERCENTAGE
3530 PRINT #1,"PERCENT FILE COMPRESSION REDUCTION:";INT(P(128))
3540 RETURN
3550 '*****************************************************
3560 '* SETS TOP OF FORM OR SCREEN DELAY BASED ON OPTION     *
3570 '*****************************************************
3580 IF D$= "LPT1:" THEN 3600
3590 PRINT:PRINT "STRIKE ENTER WHEN READY";:INPUT G$
3600 PRINT #1, CHR$(12)          'SET TOP OF FORM
3610 RETURN
3620 '************************************************
3630 '*  SUBROUTINE 12 - COMPUTE & PRINT ENTROPY       *
3640 '************************************************
3650 GOSUB 3580          'SET TOP OF PAGE
3660 FOR I= 1 TO 127
3670 IF P(I) = 0 THEN 3690
3680 E= E+P(I)* LOG(P(I))        'SEE DOCUMENTATION
3690 NEXT I
3700 E= -1*E
3710 B1= E*IR(128)
3720 B2= IR(128)*8
3730 B3= (B2-B1)/B2*100
3740 B4= B1/8
3760 PRINT #1, "SEGMENT VII - ENTROPY ANALYSIS"
3770 PRINT #1,
3780 PRINT #1, "ENTROPY:  ";E;" BITS/CHARACTER"
3790 PRINT #1, "FILE CONTAINS:"
3800 PRINT #1, INT(IR(128));" CHARACTERS"
3810 PRINT #1, INT(B2);" BITS"
3820 PRINT #1, "THEORETICAL STATISTICAL COMPRESSION: ";B3
3830 PRINT #1, "FILE WOULD REQUIRE:"
3840 PRINT #1, INT(B4);"  CHARACTERS"
3850 PRINT #1, INT(B1);" BITS"
3860 GOSUB 3580          'SET TOP OF PAGE
3870 RETURN
3880 '*************************************************************
3890 '*  SUBROUTINE 6 - SORT FREQ OCCURRENCE ARRAY DESCENDING *
3900 '*************************************************************
3910 PRINT:PRINT "(NOW SORTING FREQUENCY TABLE)";
3920 FOR I= 1 TO 125
3930 FOR K= I+1 TO 126
3940 IF (IR(I)) >= IR(K) THEN 4010       'GET GREATER COUNT
3950 TEMP= IR(I)                         'AND SWAP...
3960 IR(I)= IR(K)
3970 IR(K)= TEMP
3980 B$(129)= B$(I)
3990 B$(I)= B$(K)
```

```
4000 B$(K)= B$(129)
4010 NEXT K
4020 PRINT ".";                              'LET EM KNOW WE'RE ALIVE
4030 NEXT I
4040 RETURN
4050 PRINT "**ERROR";ERR;" ENCOUNTERED ON LINE";ERL;"***":BEEP:STOP
4060 '************************************************
4070 '*   SUBROUTINE 2 - PRINT EXECUTION REPORT SUMMARY *
4080 '************************************************
4090 FOR I= 1 TO 10:PRINT #1,:NEXT I     'SKIP 10 LINES
4100 PRINT #1,:PRINT #1,SPC(30)"DATANALYSIS"
4110 PRINT #1,:PRINT #1,SPC(24)"EXECUTION  REPORT  SUMMARY"
4120 PRINT #1,:PRINT #1,SPC(24)"FILE ANALYSED: ";F$
4130 PRINT #1,:PRINT #1,SPC(15)"SEGMENT";SPC(10)"DESCRIPTION"
4140 PRINT #1,:PRINT #1,SPC(18)"I";SPC(13)"SYSTEM STANDARD FREQUENCY OF OCCURRENCE
CF"
4150 PRINT #1,:PRINT #1,SPC(18)"II";SPC(12)"SORTED FREQUENCY OF OCCURRENCE"
4160 PRINT #1,:PRINT #1,SPC(18)"III";SPC(11)"REPEATED CHARACTER STRING ANALYSIS"

4170 PRINT #1,:PRINT #1,SPC(18)"IV";SPC(12)"RUN LENGTH ENCODING SUSCEPTIBILITY"
4180 PRINT #1,:PRINT #1,SPC(18)"V";SPC(13)"HALF-BYTE ENCODING SUSCEPTIBILITY"
4190 PRINT #1,:PRINT #1,SPC(18)"VI";SPC(12)"DIATOMIC COMPRESSION ANALYSIS"
4200 PRINT #1,:PRINT #1,SPC(18)"VII";SPC(11)"ENTROPY ANALYSIS"
4210 RETURN
4220 PRINT "**ERROR";ERR;" ENCOUNTERED ON LINE";ERL;"***":BEEP:STOP
```

APÊNDICE B

DESCRIÇÃO E LISTAGENS DO PROGRAMA SHRINK

Para fins ilustrativos, combinamos as rotinas de codificação de compressão: de meio byte, diatômica e de comprimento de fileira em dois programas, que chamamos coletivamente de SHRINK. O primeiro programa, cuja listagem está incluída na Figura B.1, é chamado MERGEC.BAS. Como o leitor pode supor, a partir do nome deste programa em BASIC, ele representa a fusão das três técnicas de compressão mencionadas anteriormente. O segundo programa em BASIC, apresentado neste apêndice, foi desenvolvido para descomprimir os dados comprimidos pelo MERGEC.DAT. Este programa é chamado MERGED.BAS e sua listagem é apresentada na Figura B.2.

Como todos os programas em BASIC apresentados neste livro, o leitor deve observar que a preocupação original foi o entendimento das técnicas de codificação e não a eficiência do programa. Assim, quase toda linha de cada programa em BASIC contém um comentário definindo a função da linha. Já que o BASIC interpretado verifica cada comentário durante a execução do programa, esta documentação excessiva torna sua execução , consideravelmente, lenta. Devido a isto, é altamente recomendado que o leitor que deseja usar estes programas, exclua todos os comentários. Além disso, a compilação de cada programa com o compilador BASIC aumentará, de forma significativa, a eficiência operacional de cada um deles.

B.1 MERGEC.BAS e MERGED.BAS

O programa MERGEC.BAS, listado na Figura B.1, tem algumas mudanças interessantes em relação aos programas anteriores, que foram usados para indicar uma técnica de compressão específica. Primeiro, a tabela Jewell foi expandida para incluir caracteres em minúsculo como indicado nas linhas de 1160 a 1190. Para reduzir o tempo de processamento do programa, os 100 primeiros caracteres do arquivo a ser comprimido, são examinados na sub-rotina de compressão diatômica. Se um par de letra minúscula for encontrado, os pares combinados de caracteres Jewell, inclusive os pares em minúscula, são combinados com o arquivo de dados. De outra forma, assume-se que o arquivo consiste de texto em maiúsculo e os dados são somente combinados com os pares em letra maiúscula, durante o processamento.

Devido à tabela expandida de pares de caracteres Jewell, a base dos caracteres usados para substituir pares de caracteres foi mudada de ASCII 224 para ASCII 199. Assim, os códigos ASCII de 199 a 249 são, agora, usados para representar a substituição de pares de caractere. Outra mudança, em relação aos dos exemplos anteriores, foi a mudança no flag de comprimento de fileira, de ASCII 125 para ASCII 128. Isto foi feito para assegurar que todos os caracteres de indicação de compressão estavam acima do ASCII 127.

Deve ser observado que os arquivos com caracteres gráficos acima do ASCII 127 podem fazer com que ocorram resultados inesperados. Isto se deve à possibilidade daqueles caracteres serem mal interpretados como se fossem caracteres de flag, que fornecerá uma falsa indicação de um tipo particular de compressão, quando o programa de descompressão atuar sobre os dados previamente comprimidos. Este problema pode ser aliviado ao se modificar o programa MERGEC.BAS, para inserir um segundo caractere de indicação de compressão, depois que um ocorre naturalmente nos dados de entrada. Então, o programa MERGED.BAS pode ser modificado para verificar o caractere seguinte aos caracteres de indicação de compressão. Se o segundo caractere for igual ao primeiro, o programa tirará, então, um caractere e ignorará o segundo, porque ele naturalmente representa dados de ocorrência e não significa que ocorreu compressão. Para os leitores que desejam expandir o escopo destes programas, as modificações descritas previamente, podem representar um desafio interessante, bem como resultar em um programa que possa comprimir programas marcados em BASIC.

```
10 '*****************************************************************
20 '* MERGEC.BAS PROGRAM COMBINES ALL COMPRESSION TECHNIQUES      *
30 '* INTO ONE PROGRAM.  THE COMPANION PROGRAM, MERGED.BAS IS     *
40 '* USED TO DECOMPRESS THIS PROGRAMS OUTPUT (COMPRESS.DAT)      *
50 '* INTO ITS ORIGINAL FORM.  IT ALLOWS UP TO 254 CHARACTER      *
60 '* INPUT STRINGS AND MIXED UPPER/LOWER CASE.  IF NO LOWER      *
70 '* CASE PAIRS ARE ENCOUNTERED IN THE 1ST 100 CHARACTERS THE   *
80 '* PROGRAM ADJUSTS ITS LOOKUPS TO UPPER CASE ONLY IN ORDER     *
90 '* TO SHORTEN RUN TIME.  AUTHOR: THOMAS R. MARSHALL  1986     *
100 '*****************************************************************
110 DIM O$(256), C(256)
120 WIDTH 80:CLS
130 '***********MAIN ROUTINE**********************
140 '* THIS ROUTINE READS RECORDS FROM AN ASCII *
150 '* FILE INTO A STRING CALLED X$ WHICH IS    *
160 '* THEN PASSED TO SUBROUTINES FOR COMPRESSION
170 '*********************************************
180 PRINT "ENTER ASCII FILENAME. EG, [DRIVE:]FILESPEC";
190 INPUT F$: OPEN F$ FOR INPUT AS #2
200 OPEN "COMPRESS.DAT" FOR OUTPUT AS #3
210 PRINT "PATIENCE - INPUT PROCESSING"
220 GOSUB 1150          'SETUP PAIR TABLE
230 IF EOF(2) THEN GOTO 1940
240 LINE INPUT #2, X$
250 N= LEN(X$)
260 GOSUB 340           'RUN LENGTH ENCODE
270 GOSUB 680:N=I-1     'SWAP I/O BUFFERS
280 GOSUB 820           'DIATOMIC ENCODE
290 GOSUB 700:N=I-1     'SWAP I/O BUFFERS
300 GOSUB 1250          'HALF BYTE ENCODE
310 GOSUB 740           'TALLY ONLY
320 GOSUB 770           'PRINT BUFFER
330 GOTO 230
340 '*****RUN LENGTH ENCODING SUBROUTINE*********
350 '* THIS ROUTINE PROCESSES RECORDS FROM X$  *
360 '* AND COMPRESSES OUT REPETITIVE CHARACTERS*
370 '* USING O$ AS THE OUTPUT BUFFER.          *
380 '*********************************************
390 K=1:J=1                     'RESET INDICES
400 FOR I= 1 TO N               'STEP THRU RECORD
410 A$= MID$(X$,I,1)            'EXTRACT A CHAR
420 IF A$= MID$(X$,I+1,1) THEN 490 'SAME AS NEXT?
430 IF K>3 THEN 530             'COMPRESS
440 IF K>1 THEN 610             'DON'T COMPRESS
450 O$(J)=A$                    'STUFF IN OUTPUT BUFFER
460 J=J+1                       'BUMP BUFFER INDEX
470 NEXT I                      'GO BACK FOR MORE
480 RETURN                      'END OF STRING
490 B$=A$                       'SAVE REPEATED CHAR
500 K=K+1                       'BUMP COUNT
510 GOTO 470                    'KEEP LOOKING
520 '*****************************************************
530 'INSERT COMPRESSION NOTATION IN OUTPUT BUFFER
540 '*****************************************************
550 O$(J)=CHR$(128)             'SET FLAG FOR RUN-LENGTH
560 O$(J+1)=B$                  'INSERT REPEATED CHAR
570 O$(J+2)=CHR$(K)             'INSERT COUNT
580 IF K=13 THEN O$(J+2)=CHR$(125) 'TRANSLATE CR
590 J=J+3:K=1                   'RESET INDEX
600 GOTO 470
610 O$(J)=B$                    'STUFF 1ST REPEAT CHAR
620 O$(J+1)=B$                  'STUFF 2ND REPEAT CHAR
630 J=J+2:K=1                   'RESET INDEX
640 GOTO 470                    'DONE
650 '*****TALLY THE COMPRESSION COUNT & WRITE BUFFER******
660 '* DISPLAY BEFORE & AFTER RESULTS OF COMPRESSION     *
670 '* AND SHOW THE NET RESULTS OBTAINED BY EACH METHOD  *
680 '*****************************************************
690 N1=N1+N                     'TALLY INPUT CHAR COUNT
```

Figura B.1

```
700 IF N=0 THEN X$=SPACE$(1)     'ALLOW FOR NEW PARA ONLY
710 FOR I= 1 TO J-1
720 MID$(X$,I,1)= O$(I)          'SWAP I/O FOR NEXT ROUTINE
730 NEXT I
740 T=N-J+1                      'NET DIFFERENCE IN BUFFERS
750 T1=T1+T                      'SAVE COUNT FOR SUMMARY
760 RETURN
770 FOR I=1 TO J-1
780 PRINT #3, O$(I);             'WRITE OUT BUFFER
790 NEXT I
800 PRINT #3, ""                 'FORCE END OF WRITE
810 RETURN
820 '*****DIATOMIC COMPRESSION SUBROUTINE*******
830 '* THIS ROUTINE PROCESSES RECORDS FROM X$  *
840 '* AND COMPRESSES OUT COMMON PAIRS         *
850 '* USING O$ AS THE OUTPUT BUFFER.          *
860 '*********************************************
870 I=1                          'RESET INDICES
880 FOR J= 1 TO N-1              'STEP THRU RECORD
890 A$= MID$(X$,J,2)             'EXTRACT A PAIR
900 IF N1>100 AND LCC=0 THEN LC=25 'NO LOWER CASE
910 FOR K = 1 TO LC              'SETUP PAIR TABLE LOOP
920 IF A$=P$(K) THEN GOSUB 1030  'IS INPUT PAIR IN TABLE?
930 NEXT K                       'NO - TRY NEXT
940 IF M = 1 THEN 960            'IF MATCH FLAG SET?
950 O$(I) = MID$(A$,1,1)         'NO-STUFF 1ST CHAR IN BUFFER
960 I=I+1                        'BUMP INPUT STRING INDEX
970 M=0                          'RESET MATCH FLAG
980 NEXT J                       'GO BACK FOR MORE
990 IF J=N+1 THEN J=I:GOTO 1020  'ONE TOO MANY
1000 O$(I)= MID$(X$,J,1)          'GET LAST CHAR
1010 J=I+1                        'RESET INDEX
1020 RETURN                       'DONE
1030 M=1                          'SET PAIR MATCH FLAG
1040 '************************************************************
1050 'INSERT COMPRESSION NOTATION IN OUTPUT BUFFER
1060 V = K + 199                  'INDEX OUT TO SUBSTITUTE CHAR
1070 O$(I)=CHR$(V)                'INSERT PAIR SUBSTITUTION
1080 J=J+1                        'FORCE INPUT SHIFT 2 OVER PAIR
1090 IF K>25 THEN LCC=1           'FOUND LOWER CASE PAIR
1100 K = LC                       'FORCE END OF PAIR SEARCH
1110 RETURN                       'GO BACK FOR MORE
1120 '*****************************************************
1130 '*  CONSTRUCT PAIR COMBINATION TABLE                 *
1140 '*****************************************************
1150 DIM P$(50)                   'JEWELL CHAR. COMBINATION PAIRS
1160 DATA "E "," T",TH," A","S ",RE,IN,HE,ER," I"," O","N ",ES,
1170 DATA " B",ON,"T ",TI,AN,"D ",AT,TE," C"," S",OR,"R "
1180 DATA "e "," t",th," a","s ",re,in,he,er," i"," o","n ",es,
1190 DATA " b",on,"t ",ti,an,"d ",at,te," c"," s",or,"r "
1200 FOR I = 1 TO 50              'SETUP PAIR TABLE
1210 READ Z$                      'GET COMMON PAIR
1220 P$(I) = Z$: NEXT I           'AND STUFF INTO PAIR TABLE
1230 LC=50:LCC=0                  'SETUP LOOP COUNT & LC FLAG
1240 RETURN                       'DONE - TABLE COMPLETE
1250 '*****HALF-BYTE ENCODING SUBROUTINE*********
1260 '* THIS ROUTINE PROCESSES RECORDS FROM X$  *
1270 '* AND ENCODES  MIXED  STRINGS OF DATA INTO*
1280 '* HALF-BYTE OR 4 BIT REPRESENTATION USING *
1290 '* DOUBLE BUFFERING WITH O$ AS OUTPUT BUFF.*
1300 '*********************************************
1310 K=1:J=1                      'RESET INDICES
1320 FOR I=1 TO N STEP 2          'STEP THRU RECORD
1330 IF I=N-1 THEN  1530          'EVEN STRING SIZE
1340 C(I)=0:C(I+1)=0              'RESET ENCODE FLAGS
1350 A$= MID$(X$,I,1)             'GET 1ST BYTE
1360 B$= MID$(X$,I+1,1)           'GET 2ND BYTE
1370 IF A$= "$" THEN C(I)= 1      'SET 1ST ENCODE FLAG
1380 IF A$= "," THEN C(I)= 2
```

Figura B.1 (continuação)

```
1390 IF A$= "." THEN C(I)= 3
1400 IF A$= "*" THEN C(I)= 4
1410 IF A$< "0" OR A$> "9" THEN 1430  'SKIP OTHERS
1420 C(I)= 5
1430 IF B$= "$" THEN C(I+1)= 1    'SET 2ND ENCODE FLAG
1440 IF B$= "," THEN C(I+1)= 2
1450 IF B$= "." THEN C(I+1)= 3
1460 IF B$= "*" THEN C(I+1)= 4
1470 IF B$< "0" OR B$> "9" THEN 1490  'SKIP OTHERS
1480 C(I+1)= 5
1490 IF C(I)= 0 OR C(I+1)= 0 THEN 1530  'NOT CANDIDATE
1500 K=K+2                            'BOTH NUMERIC-BUMP COUNT
1510 NEXT I                           'GO BACK FOR MORE
1520 RETURN                           'END OF STRING
1530 IF K > 4 THEN GOSUB 1620         'ENOUGH TO ENCODE
1540 IF K > 1 THEN GOSUB 1880         'DON'T ENCODE
1550 O$(J) = MID$(X$,I,1)             'OUTPUT 1ST CHAR.
1560 O$(J+1) = MID$(X$,I+1,1)         'OUTPUT 2ND CHAR.
1570 J=J+2:K=1                        'BUMP OUTPUT-RESET COUNT
1580 IF I=N THEN J=J-1                'ONE TOO MANY
1590 IF J=N+2 THEN J=N+1              'EVEN STRING, SUB 1
1600 GOTO 1510                        'AND GO FOR MORE
1610 '***** SUBROUTINE TO PERFORM HALF-BYTE ENCODING *****
1620 O$(J)=CHR$(129)                  'FLAG FOR BYTE PACKING
1630 MASK1= &HF0                      '11110000
1640 MASK2= &HF                       '00001111
1650 O$(J+1)=CHR$(K-1)                'INSERT LENGTH OF STRING
1660 J=J+2                            'BUMP OUTPUT INDEX
1670 FOR L=(I-K)+1 TO I-2 STEP 2 'SETUP ENCODE LOOP
1680 ON C(L) GOTO 1690,1700,1710,1720,1730 'USE FLAG TO ENCODE
1690 X=&HA0:GOTO 1750                 '10100000
1700 X=&HB0:GOTO 1750                 '10110000
1710 X=&HC0:GOTO 1750                 '11000000
1720 X=&HD0:GOTO 1750                 '11010000
1730 X=VAL(MID$(X$,L,1))              'GET NUM VALUE OF BYTE 1
1740 X=X*16                           'SHIFT 4 BITS LEFT
1750 X=X AND MASK1                    'MASK LOWER HALF-BYTE
1760 ON C(L+1) GOTO 1770,1780,1790,1800,1810 'USE ENCODE FLAG
1770 Y=&HA:GOTO 1820                  '00001010
1780 Y=&HB:GOTO 1820                  '00001011
1790 Y=&HC:GOTO 1820                  '00001100
1800 Y=&HD:GOTO 1820                  '00001101
1810 Y=VAL(MID$(X$,L+1,1))            'GET NUM VALUE OF BYTE 2
1820 Y=Y AND MASK2                    'MASK UPPER HALF-BYTE
1830 Z= X OR Y                        'OR THE TWO TOGETHER
1840 O$(J)= CHR$(Z)                   'OUTPUT BYTE TO BUFFER
1850 J=J+1                            'BUMP OUTPUT INDEX
1860 NEXT L                           'GO BACK FOR MORE
1870 K=1:RETURN                       'RESET COUNT AND RETURN
1880 '***** SUBROUTINE FOR STRING NOT WORTH ENCODING *****
1890 FOR L=(I-K)+1 TO I-2             'PICKUP SHORT STRING
1900 O$(J)=MID$(X$,L,1)               'STUFF IN OUTPUT BUFFER
1910 J=J+1                            'BUMP OUTPUT INDEX
1920 NEXT L                           'GO BACK FOR MORE
1930 K=1:RETURN                       'RESET COUNT AND RETURN
1940 CLOSE: OPEN F$ FOR INPUT AS #2
1950 PRINT "FILE ";F$;" BEFORE COMPRESSION:"
1960 LINE INPUT #2,X$
1970 IF EOF(2) THEN 2000
1980 PRINT X$
1990 GOTO 1960
2000 PRINT X$:OPEN "COMPRESS.DAT" FOR INPUT AS #3
2010 PRINT "FILE ";F$;" AFTER COMPRESSION:"
2020 LINE INPUT #3,O$
2030 IF EOF(3) THEN 2060
2040 PRINT O$
2050 GOTO 2020
2060 PRINT O$:PRINT T1;" TOTAL CHARACTERS ELIMINATED FROM ";
2070 PRINT N1;"OR ";INT((T1/N1)*100);"%":CLOSE:END
```

Figura B.1 (continuação)

```
10 '*****************************************************
20 '* MERGED.BAS PROGRAM WAS WRITTEN TO DECOMPRESS     *
30 '* FILES ENCODED BY ITS COMPANION PROG MERGEC.BAS.  *
40 '* THE INPUT FILE IS NORMALLY COMPRESS.DAT BUT CAN  *
50 '* BE CHANGED TO ACCOMMODATE ANY ENCODED FILE.      *
60 '* THE DECOMPRESSED OUTPUT FILE IS DECOMP.DAT AND   *
70 '* SHOULD BE SAVED UNDER ANOTHER NAME AFTER EACH    *
80 '* RUN OF MERGED.BAS. AUTHOR: THOMAS R. MARSHALL    *
90 '*****************************************************
100 DIM O$(256)
110 WIDTH 80:CLS
120 '***********MAIN ROUTINE**********************
130 '* THIS ROUTINE READS RECORDS FROM AN ASCII *
140 '* FILE INTO A STRING CALLED X$ WHICH IS    *
150 '* THEN PASSED TO DECOMPRESSION SUBROUTINE  *
160 '*****************************************************
170 PRINT "ENTER ASCII FILENAME. EG, COMPRESS.DAT";
180 INPUT F$: OPEN F$ FOR INPUT AS #2
190 OPEN "DECOMP.DAT" FOR OUTPUT AS #3
200 GOSUB 1030          'CONSTRUCT PAIR TABLE
210 PRINT "PATIENCE - INPUT PROCESSING"
220 IF EOF(2) THEN GOTO 1530
230 LINE INPUT #2, X$
240 N= LEN(X$)
250 GOSUB 330           'RUN LENGTH DECODE
260 GOSUB 620:N=I-1     'SWAP I/O BUFFERS
270 GOSUB 760           'DIATOMIC DECODE
280 GOSUB 640:N=I-1     'SWAP I/O BUFFERS
290 GOSUB 1120          'HALF BYTE DECODE
300 GOSUB 680           'PICKUP LAST COUNT
310 GOSUB 710           'PRINT BUFFERS
320 GOTO 220
330 '*****RUN LENGTH DECODING SUBROUTINE********
340 '* THIS ROUTINE PROCESSES RECORDS FROM X$  *
350 '* AND DECOMPRESSES RUN-ENCODED CHARACTERS *
360 '* USING O$ AS THE OUTPUT BUFFER.          *
370 '*********************************************
380 K=1:J=1                         'RESET INDICES
390 FOR I= 1 TO N                   'STEP THRU RECORD
400 A$= MID$(X$,I,1)                'EXTRACT A CHAR
410 IF A$= CHR$(128) THEN 470       'COMPRESSION FLAG?
420 O$(J)=A$                        'STUFF IN OUTPUT BUFFER
430 J=J+1                           'BUMP BUFFER INDEX
440 NEXT I                          'GO BACK FOR MORE
450 RETURN                          'END OF STRING
460 '*****************************************************
470 'DECODE COMPRESSION NOTATION TO OUTPUT BUFFER
480 '*****************************************************
490 K$= MID$(X$,I+2,1)              'GET REPEAT COUNT
500 A$= MID$(X$,I+1,1)              'GET REPEAT CHAR
510 K= ASC(K$)                      'SET UP INDEX
520 IF K=125 THEN K=13              'TRANSLATE CR
530 FOR L= J TO J+K-1               'SET OUTPUT LOOP
540 O$(L)= A$                       'STUFF REPEAT CHAR
550 NEXT L                          'KEEP GOING
560 J= L                            'BUMP OUTPUT INDEX
570 I= I+3                          'BUMP INPUT INDEX
580 GOTO 400                        'DONE
590 '*****TALLY THE DECOMPRESSION COUNT & WRITE BUFFER****
600 '* DISPLAY BEFORE & AFTER RESULTS OF DECOMPRESSION   *
610 '* AND SHOW THE NET RESULTS OBTAINED BY EACH METHOD  *
620 '*****************************************************
630 N1=N1+N                         'TALLY INPUT CHAR COUNT
640 X$=SPACE$(J-1)                  'EXPAND INPUT BUFFER
650 FOR I= 1 TO J-1
660 MID$(X$,I,1)= O$(I)             'SWAP I/O FOR NEXT ROUTINE
670 NEXT I
680 T=J-1-N                         'NET DIFFERENCE IN BUFFERS
690 T1=T1+T                         'SAVE COUNT FOR SUMMARY
700 RETURN
710 FOR I=1 TO J-1
720 PRINT #3, O$(I);                'WRITE OUT BUFFER
```

Figura B.2

```
730 NEXT I
740 PRINT #3, ""                    'END WRITE
750 RETURN
760 '*****DIATOMIC   DECODING SUBROUTINE********
770 '* THIS ROUTINE PROCESSES RECORDS FROM X$  *
780 '* AND DECOMPRESSES PAIR-ENCODED CHARACTERS*
790 '* USING O$ AS THE OUTPUT BUFFER.          *
800 '*********************************************
810 K=1:J=1:V=0                     'RESET INDICES
820 FOR I= 1 TO N                   'STEP THRU RECORD
830 A$= MID$(X$,I,1)                'EXTRACT A CHAR
840 IF A$> CHR$(199) THEN 900       'COMPRESSED PAIR?
850 O$(J)=A$                        'STUFF IN OUTPUT BUFFER
860 J=J+1                           'BUMP BUFFER INDEX
870 NEXT I                          'GO BACK FOR MORE
880 RETURN                          'END OF STRING
890 '*********************************************************
900 'DECODE COMPRESSION NOTATION TO OUTPUT BUFFER
910 '*********************************************************
920 K= ASC(A$)                      'GET ORDINAL EQUIV.
930 K= K-199                        'SUBTRACT FOR INDEX
940 T$= P$(K)                       'STUFF PAIR IN BUFFER
950 O$(J)=MID$(T$,1,1)
960 O$(J+1)=MID$(T$,2,1)
970 J= J+2                          'BUMP OUTPUT INDEX
980 V= V+1                          'SUM VARIABLE COUNT
990 GOTO 870                        'DONE
1000 '*********************************************
1010 "*  CONSTRUCT PAIR COMBINATION TABLE        *
1020 '*********************************************
1030 DIM P$(50)                      'JEWELL CHAR. COMBINATION PAIRS
1040 DATA "E "," T",TH," A","S ",RE,IN,HE,ER," I"," O","N ",ES,
1050 DATA " B",ON,"T ",TI,AN,"D ",AT,TE," C"," S",OR,"R "
1060 DATA "e "," t",th," a","s ",re,in,he,er," i"," o","n ",es,
1070 DATA " b",on,"t ",ti,an,"d ",at,te," c"," s",or,"r "
1080 FOR I = 1 TO 50                 'SET UP PAIR TABLE
1090 READ Z$                         'GET COMMON PAIR
1100 P$(I) = Z$: NEXT I              'AND STUFF INTO PAIR TABLE
1110 RETURN                         'DONE - TABLE COMPLETE
1120 '*****HALF BYTE  DECODING SUBROUTINE********
1130 '* THIS ROUTINE PROCESSES RECORDS FROM X$  *
1140 '* AND DECOMPRESSES BYTE-ENCODED CHARACTERS*
1150 '* USING O$ AS THE OUTPUT BUFFER.          *
1160 '*********************************************
1170 J=1                             'RESET INDEX
1180 FOR I= 1 TO N                   'STEP THRU RECORD
1190 A$= MID$(X$,I,1)                'EXTRACT A CHAR
1200 IF A$= CHR$(129) THEN 1280      'COMPRESSION FLAG?
1210 O$(J)=A$                        'STUFF IN OUTPUT BUFFER
1220 J=J+1                           'BUMP BUFFER INDEX
1230 NEXT I                          'GO BACK FOR MORE
1240 RETURN                          'END OF STRING
1250 '*********************************************************
1260 'DECODE COMPRESSION NOTATION TO OUTPUT BUFFER       *
1270 '*********************************************************
1280 MASK1= &HF0                     '11110000
1290 MASK2= &HF                      '00001111
1300 K= ASC(MID$(X$,I+1,1))          'GET STRING LENGTH
1310 M= I+2+(K/2)-1                  'SET END OF STRING
1320 FOR L=I+2 TO M                  'SETUP LOOP TO DECODE
```

Figura B.2 (continuação)

```
1330 Z= ASC(MID$(X$,L,1))          'GET BYTE
1340 X= (Z AND MASK1)/16           'MASK LOWER HALF-BYTE
1350 IF X< 10 THEN 1410            'ITS NUMERIC
1360 IF X= 10 THEN O$(J)= "$"      'SPECIAL
1370 IF X= 11 THEN O$(J)= ","      'SPECIAL
1380 IF X= 12 THEN O$(J)= "."      'SPECIAL
1390 IF X= 13 THEN O$(J)= "*"      'SPECIAL
1400 GOTO 1420                      'SKIP IF SPECIAL
1410 O$(J)= CHR$(X+48)             'OUTPUT 1ST NUMERIC
1420 Y= Z AND MASK2                'MASK UPPER HALF-BYTE
1430 IF Y< 10 THEN 1490             'ITS NUMERIC
1440 IF Y= 10 THEN O$(J+1)= "$"    'SPECIAL
1450 IF Y= 11 THEN O$(J+1)= ","    'SPECIAL
1460 IF Y= 12 THEN O$(J+1)= "."    'SPECIAL
1470 IF Y= 13 THEN O$(J+1)= "*"    'SPECIAL
1480 GOTO 1500                     'SKIP IF SPECIAL
1490 O$(J+1)= CHR$(Y+48)           'OUTPUT 2ND NUMERIC
1500 J= J+2                        'BUMP OUTPUT BY TWO
1510 NEXT L:I= M                   'CONTINUE, BUMP INPUT INDEX
1520 GOTO 1230                     'GO BACK FOR MORE
1530 CLOSE: OPEN F$ FOR INPUT AS #2
1540 PRINT K$
1550 PRINT "FILE ";F$;" BEFORE DECOMPRESSION:"
1560 LINE INPUT #2,X$
1570 IF EOF(2) THEN 1600
1580 PRINT X$
1590 GOTO 1560
1600 PRINT X$:OPEN "DECOMP.DAT" FOR INPUT AS #3
1610 PRINT "FILE ";F$;" AFTER DECOMPRESSION:"
1620 LINE INPUT #3,O$
1630 IF EOF(3) THEN 1660
1640 PRINT O$
1650 GOTO 1620
1660 PRINT O$:PRINT T1;" TOTAL CHARACTERS INSERTED"
1670 CLOSE:END
```

Figura B.2 (continuação)

B.2 OPERAÇÕES DE PROGRAMA

A não ser que seja alterado pelo leitor, o MERGEC.BAS criará um arquivo chamado COMPRESS.DAT, que representa uma forma comprimida do arquivo de entrada especificado, quando o programa é executado.

A Figura B.3 representa uma mensagem típica de correio eletrônico que você pode transmitir. Esta mensagem contém 776 caracteres. Já que muitos serviços de correio eletrônico cobram tanto pelo tempo de conexão, quanto pelo número de caracteres transmitidos, qualquer redução nos dados pode resultar em uma comensurável redução no custo de transmissão.

A Figura B.4 ilustra o arquivo comprimido resultante, depois que o MERGEC.BAS atuou sobre o arquivo de dados de amostra listado na Figura B.3. O leitor deve observar que alguns dos caracteres de flag de compressão são equivalentes aos caracteres de controle de impressão mais interessantes, que é a razão pela qual a listagem do arquivo comprimido assemelha-se, vagamente, ao arquivo original de dados. Como indicado, os 245 caracteres, representando 31 por cento dos dados originais, foram eliminados, devido à mistura de três rotinas de compressão usadas no programa.

```
J.J. ASTOR
ASTORIA, OREGON

MY GOOD SIR;

I AM RESPONDING TO YOUR CORRESPONDENCE LAST REGARDING THE DISPOSITION
OF PROCEEDS FROM SALE OF SAID PROPERTY.  THE GROSS PROCEEDS REALIZED
WERE $832,746,381.99.

FROM THIS, WE HAVE TAKEN THE LIBERTY OF DEDUCTING MINOR EXPENSES IN THE
FOLLOWING MANNER:

A. TRANSFER FEES          $136,941
B. REALITY FEES         $8,327,436
C. LEGAL FEES           $9,938,862
D. ADVERTISING            $422,977
E. TAXES                $1,646,311
F. SUNDRY               $7,139,774
G. MISCELLANEOUS      $462,114,283

THE NET PROCEEDS RESIDE IN A NON-INTEREST BEARING ACCOUNT UNTIL
INSTRUCTIONS TO THE CONTRARY ARE RECEIVED.  TO OUR VALUED CLIENT WE
REMAIN-

                                        VERY TRULY YOURS,

                                        DEWEY, CHEATHAM & HOWE
```

Figura B.3

Diversos itens importantes, com relação ao arquivo comprimido justificam discussão adicional, especialmente se você deseja transmitir estes dados através de um serviço de correio eletrônico. Primeiro, quando você comprimir os dados, certifique-se de remover quaisquer informações de destino do arquivo. Isto se dá porque o sistema de correio eletrônico não pode entender suas solicitações de rota, se eles estiverem comprimidos. Em segundo lugar, antes de transmitir os dados comprimidos, o leitor deve verificar se o serviço de correio eletrônico suporta o conjunto de caractere ASCII estendido, que é, às vezes, chamado ASCII de 8 bits. Se o serviço de correio eletrônico não suporta o ASCII estendido, você não será capaz de transmitir os dados comprimidos, já que os caracteres de indicação de compressão são todos acima do ASCII 127. Por último, vamos fazer uma pergunta ao leitor. A partir do exame da Figura B.4, você poderia determinar o que a mensagem original quis dizer?

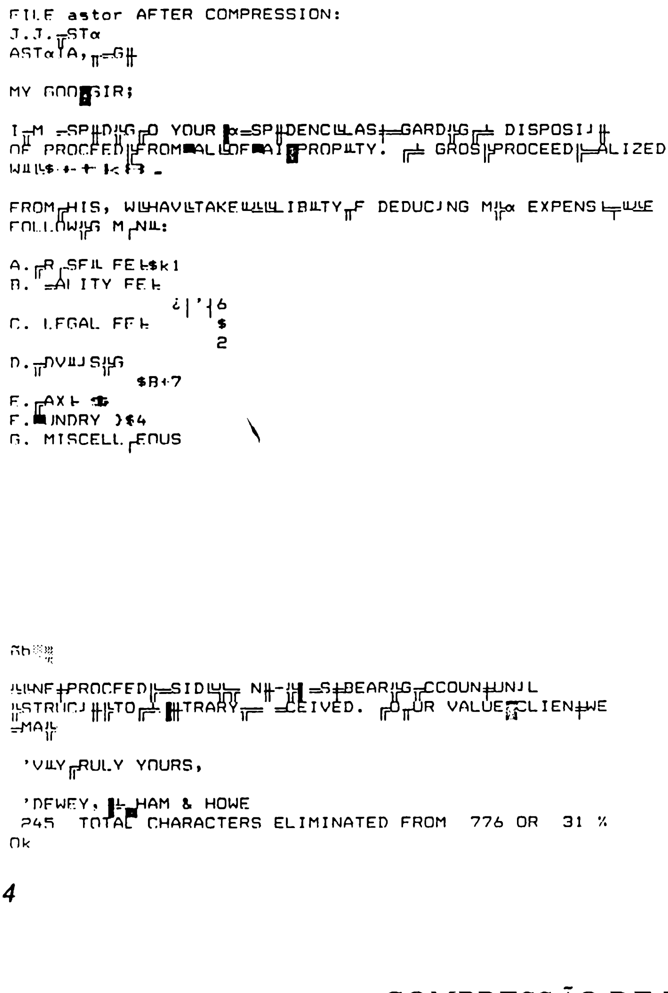


Figura B.4

A compressão não é um substituto da utilização de dispositivos de criptografia, mas ela pode tornar muito mais difícil para um leitor não autorizado, decifrar a mensagem.

Antes de deixar o leitor desvendar o SHRINK, vamos rever a compressão e descompressão completa de um pequeno arquivo. Isto está ilustrado na Figura B.5. No topo da Figura B.5, está listado o arquivo de quatro linhas para o qual demos o nome tst. A seguir, ao carregar e executar o programa MERGEC.BAS, somos impelidos a entrar com o arquivo ASCII que desejamos comprimir. Como indicado, o programa lista o arquivo original, bem como sua forma comprimida.

```
Lets take a look at the benefits of compression when we
have different types of data to include large numbers
such as $123,456,789.98 and repeating strings in a file,
such as ---------------.

ENTER ASCII FILENAME. EG, [DRIVE:]FILESPEC? tst
PATIENCE - INPUT PROCESSING
FILE tst BEFORE COMPRESSION:
Lets take a look at the benefits of compression when we
have different types of data to include large numbers
such as $123,456,789.98 and repeating strings in a file,
such as ---------------.
FILE tst AFTER COMPRESSION:
Letptakra lookσ≥Γrbenefitpof°omprssi± w0Øwe
havrdiffßen≥typ€∞f d÷aπoδncludrlargrnumbßs
suchσpí ##┤V┐          ┌BσnJ rpe÷Σg·tr Σgμ Σσ  file,

suchσp-▪

 53  TOTAL CHARACTERS ELIMINATED FROM  187 OR  28 %
Ok
```

Rodando MERGEC.BAS

```
ENTER ASCII FILENAME. EG, COMPRESS.DAT? compress.dat
PATIENCE - INPUT PROCESSING


FILE compress.dat BEFORE DECOMPRESSION:
Letptakra lookσ≥Γrbenefitpof°omprssi± w0Øwe
havrdiffßen≥typ€∞f d÷aπoδncludrlargrnumbßs
suchσpí ##┤V┐          ┌BσnJ rpe÷Σg·tr Σgμ Σσ  file,

suchσp-▪

FILE compress.dat AFTER DECOMPRESSION:
Lets take a look at the benefits of compression when we
have different types of data to include large numbers
such as $123,456,78920T8 and repeating strings in a file,
such as ---------------.
 54  TOTAL CHARACTERS INSERTED
Ok
```

Rodando MERGED.BAS

Figura B.5

A segunda parte da Figura B.5 ilustra a execução do MERGED.BAS, que converterá um arquivo comprimido de volta a sua forma original. Já que o MERGEC.BAS usou, automaticamente, o nome de arquivo compress.dat para armazenar os dados comprimidos, usamos aquele arquivo como entrada do MERGED.BAS. Como indicado, o arquivo é fielmente reconstruído para sua forma original; contudo, a partir de um exame cuidadoso, 53 caracteres foram eliminados e 54 foram inseridos, de acordo com o que foi relatado. Caso o leitor esteja confuso, deve-se observar que no caso da codificação de meio byte, a economia de um número ímpar de bytes foi arredondado para baixo, no momento em que a inserção de caracteres reais é contada, resultando, ocasionalmente, em discrepância de um caractere ao comparar as remoções de caracteres e as inserções de caracteres. Isto, contudo, não tem efeito sobre a decodificação de dados previamente comprimidos.

APÊNDICE C

DESCRIÇÕES E LISTAGENS DO PROGRAMA EM C

COMPRESSÃO DE DADOS EM C

Apresentamos as versões em linguagem C, de alguns programas em Basic descritos nos capítulos anteriores. Note que existe uma ligeira diferença tanto a nível de programação quanto a nível de resultado final. O principal objetivo é demonstrar aqui, as teorias de compactação envolvidas. Os programas PKFILE.C, SAVEPK.C, PK_SCR.C, PAIRC2.C, PAIRD2.C, bem como a versão em C dos programas escritos originalmente em Basic, são de autoria de Carlos Augusto P. Gomes e Antonio Carlos Barbosa, autores dos Livros "*Clipper com C - Sem Mistérios, Caixa de Ferramentas, Técnicas Gráficas Avançadas, Fundamentos de Programação em Linguagem C e Clipper 5.0, o que mudou...*".

RunLenC.c

```
#include <stdio.h>

char    o[133],
        arquivo[13],
        buffer[500];

int     t1=0,
        t=0,
        n=0,
        n1=0;


FILE    *input,
        *output;

main ()
{
    int  ch=0;

    printf("Entre com o arquivo (formato ASCII). Ex: RUNLEN.DAT :");
    scanf("%13s",arquivo);

    if ( (input  = fopen(arquivo,"rb")) == NULL )
    {
        printf("ERRO: Impossivel abrir arquivo %s para leitura\n",arquivo);
        exit(-1);
    }
    if ( (output = fopen("RUNLENC.DAT","wb")) == NULL )
    {
        printf("ERRO: Impossivel abrir arquivo RUNLENC.DAT para escrita\n");
        exit(-2);
    }

    while( fgets(buffer,500,input) != NULL )
    {
        n = strlen(buffer);
        tally( runlength() );
    }
    estatistic();
}

runlength()
{
```

```
    int         i=0,
                k=1,
                j=0;

    unsigned char   a,
                    b;

    for(i=0;i < n;i++)
    {
        a = buffer[i];

        if ( a == buffer[i+1] )
        {
            b=a;
            k++;
            continue;
        }
        if ( k > 3 )
        {
            o[j] = 125;
            o[j+1] = b;;
            o[j+2] = k;
            j+=3;
            k=1;
        }
        else
        {
            if ( k == 3 )
            {
                o[j]      = b;
                o[j+1]    = b;
                j+=2;
                k=1;
            }
            else
            {
                o[j] = a;
                j++;
            }
        }
    }
    return(j);
}

tally (j)
int   j;
{
```

```
    int         i=0;

    n1=n1+n;
    t=n-j;
    t1=t1+t;
    for(i=0;i<j;i++)
    {
        fputc(o[i],output);
    }
}

estatistic()
{
    fclose(input);
    fclose(output);

    if ( (input  = fopen(arquivo,"rb")) == NULL )
    {
        printf("ERRO: Impossivel abrir arquivo %s para leitura\n",arquivo);
        exit(-1);
    }

    printf("Arquivo %s antes da compressao:\n",arquivo);

    while( fgets(buffer,500,input) != NULL )
    {
        printf("%s",buffer);
    }

    fclose(input);

    if ( (input = fopen("runlenc.dat","rb")) == NULL )
    {
        printf("ERRO: Impossivel abrir arquivo RUNLENC.DAT para leitura\n");
        exit(-2);
    }

    printf("\n\nArquivo %s depois da compressao:\n",arquivo);

    while( fgets(buffer,500,input) != NULL )
    {
        printf("%s",buffer);
    }

    fclose(input);
```

```
    printf("\n%d Total de caracteres eliminados de %d ou %d%%\n\n",
                t1,n1,(int)((float)((float)t1/(float)n1)*100));
}
```

RunLenD.c

```
#include <stdio.h>

char    o[133],
        arquivo[13],
        buffer[500];

int     t1=0,
        t=0,
        n=0,
        n1=0;


FILE    *input,
        *output;

main()
{
    int     ch=0;

    printf("Entre com o arquivo. Ex: RUNLENC.DAT :");
    scanf("%13s",arquivo);

    if ( (input  = fopen(arquivo,"rb")) == NULL )
    {
        printf("ERRO: Impossivel abrir arquivo %s para leitura\n",arquivo);
        exit(-1);
    }
    if ( (output = fopen("RUNLEND.DAT","wb")) == NULL )
    {
        printf("ERRO: Impossivel abrir arquivo RUNLEND.DAT para escrita\n");
        exit(-2);
    }

    while( fgets(buffer,500,input) != NULL )
    {
        n = strlen(buffer);
        tally( runlength() );
    }
    estatistic();
```

```
}

runlength()
{
    int     i=0,
            l=0,
            k=0,
            j=0;

    unsigned char   a,
                    b;

    for(i=0;i < n;i++)
    {
        if ( buffer[i] == 125 )
        {
            a = buffer[i+1];
            k = buffer[i+2];

            for (l=0;l<k;l++)
            {
                o[j++] = a;
            }
            i+=2;
        }
        else
        {
            o[j++] = buffer[i];
        }
    }
    return(j);
}

tally(j)
int  j;
{
    int     i=0;

    n1=n1+n;
    t=n-j;
    t1=t1-t;
    for(i=0;i<j;i++)
    {
        fputc(o[i],output);
    }
}
```

```
estatistic()
{
    fclose(input);
    fclose(output);

    if ( (input  = fopen("runlenc.dat","rb")) == NULL )
    {
        printf("ERRO: Impossivel abrir arquivo %s para leitura\n",arquivo);
        exit(-1);
    }

    printf("Arquivo %s antes da compressao:\n",arquivo);

    while( fgets(buffer,500,input) != NULL )
    {
        printf("%s",buffer);
    }

    fclose(input);

    if ( (input = fopen("runlend.dat","rb")) == NULL )
    {
        printf("ERRO: Impossivel abrir arquivo RUNLEND.DAT para leitura\n");
        exit(-2);
    }

    printf("\n\nArquivo %s depois da compressao:\n",arquivo);

    while( fgets(buffer,500,input) != NULL )
    {
        printf("%s",buffer);
    }

    fclose(input);

    printf("\n%d Total de caracteres inseridos\n\n",t1);
}
```

ByteC.c

```
#include <stdio.h>

unsigned char       o[133],
                    arquivo[13],
                    buffer[500];
```

```
int     t1=0,
        t=0,
        n=0,
        j=0,
        k=0,
        i=0,
        n1=0;


FILE    *input,
        *output;

main()
{
    int  ch=0;

    printf("Entre com o arquivo (formato ASCII). Ex: BYTE.DAT :");
    scanf("%13s",arquivo);

    if ( (input  = fopen(arquivo,"rb")) == NULL )
    {
        printf("ERRO: Impossivel abrir arquivo %s para leitura\n",arquivo);
        exit(-1);
    }
    if ( (output = fopen("BYTEC.DAT","wb")) == NULL )
    {
        printf("ERRO: Impossivel abrir arquivo BYTEC.DAT para escrita\n");
        exit(-2);
    }

    while( fgets(buffer,500,input) != NULL )
    {
        n = strlen(buffer);
        bytec();
        tally();
    }
    estatistic();
}


bytec()
{
    k=0;
    j=0;
```

```
    for(i=0;i < n-1;i+=2)
    {
        if ( buffer[i]  < '0' || buffer[i]  > '9' )
        {
            halfbyte();
            continue;
        }
        if ( buffer[i+1] < '0' || buffer[i+1] > '9' )
        {
            halfbyte();
            continue;
        }
        k+=2;
    }
}

halfbyte()
{
    int     l=0;

    char    x=0,
            y=0;

    if ( k > 3 )
    {
        o[j] = 126;
        o[j+1] = k;
        j+=2;
        for (l=(i-k);l<=k;l+=2)
        {
            x = buffer[l+1];
            y = buffer[l];
            o[j] = x+(y*10) - 16;
            j++;
        }
    }
    else
    {
        if ( k > 0 )
        {
            for (l=(i-k);l<=k;l++)
            {
                o[j] = buffer[l];
                j++;
            }
        }
```

```
    }
    o[j]  = buffer[i];
    o[j+1]= buffer[i+1];
    j+=2;
    k=0;
}

tally()
{
    int       i=0;

    n1=n1+n;
    t=n-j;
    t1=t1+t;
    for(i=0;i<j;i++)
    {
        fputc(o[i],output);
    }
}

estatistic()
{
    fclose(input);
    fclose(output);

    if ( (input  = fopen(arquivo,"rb")) == NULL )
    {
        printf("ERRO: Impossivel abrir arquivo %s para leitura\n",arquivo);
        exit(-1);
    }

    printf("Arquivo %s antes da compressao:\n",arquivo);

    while( fgets(buffer,500,input) != NULL )
    {
        printf("%s",buffer);
    }

    fclose(input);

    if ( (input = fopen("bytec.dat","rb")) == NULL )
    {
        printf("ERRO: Impossivel abrir arquivo BYTEC.DAT para leitura\n");
        exit(-2);
    }
```

```
    printf("\n\nArquivo %s depois da compressao:\n",arquivo);

    while( fgets(buffer,500,input) != NULL )
    {
        printf("%s",buffer);
    }

    fclose(input);

    printf("\n%d Total de caracteres eliminados de %d ou %d%%\n\n",
                    t1,n1,(int)((float)((float)t1/(float)n1)*100));
}
```

ByteD.c

```
#include <stdio.h>

unsigned char       o[133],
                    arquivo[13],
                    buffer[500];

int     t1=0,
        t=0,
        n=0,
        j=0,
        k=0,
        i=0;


FILE    *input,
        *output;

main()
{
    int     ch=0;

    printf("Entre com o arquivo. Ex: BYTEC.DAT :");
    scanf("%13s",arquivo);

    if ( (input  = fopen(arquivo,"rb")) == NULL )
    {
        printf("ERRO: Impossivel abrir arquivo %s para leitura\n",arquivo);
        exit(-1);
    }
    if ( (output = fopen("BYTED.DAT","wb")) == NULL )
```

```
    {
        printf("ERRO: Impossivel abrir arquivo BYTED.DAT para escrita\n");
        exit(-2);
    }

    while( fgets(buffer,500,input) != NULL )
    {
        n = strlen(buffer);
        byted();
        tally();
    }
    estatistic();
}

byted()
{
    int     m=0,
            l=0,
            x=0,
            z=0,
            y=0;

    k=0;
    j=0;

    for(i=0;i < n;)
    {
        if ( buffer[i] == 126 )
        {
            k = buffer[i+1];
            m = i+2;
            for (l=j;l<j+k;l+=2)
            {
                x = buffer[m];
                y = x * .1;
                o[l] = y+48;
                z = x-(y*10);
                o[l+1] = z+48;
                m++;
            }
            j=l;
            i=m;
        }
        else
        {
            o[j] = buffer[i];
```

```
                j++;
                i++;
            }
        }
}

tally()
{
    int     i=0;

    t=(n-1)-j;
    t1=t1-t;
    for(i=0;i<j;i++)
    {
        fputc(o[i],output);
    }
}

estatistic()
{
    fclose(input);
    fclose(output);

    if ( (input  = fopen(arquivo,"rb")) == NULL )
    {
        printf("ERRO: Impossivel abrir arquivo %s para leitura\n",arquivo);
        exit(-1);
    }

    printf("Arquivo %s antes da compressao:\n",arquivo);

    while( fgets(buffer,500,input) != NULL )
    {
        printf("%s",buffer);
    }

    fclose(input);

    if ( (input = fopen("byted.dat","rb")) == NULL )
    {
        printf("ERRO: Impossivel abrir arquivo BYTED.DAT para leitura\n");
        exit(-2);
    }

    printf("\n\nArquivo %s depois da compressao:\n",arquivo);
```

```
    while( fgets(buffer,500,input) != NULL )
    {
        printf("%s",buffer);
    }

    fclose(input);

    printf("\n%d Total de caracteres inseridos\n\n",t1);
}
```

PackC.c

```
#include <stdio.h>

char    o[133],
        c[133],
        arquivo[13],
        buffer[500];

int     j=0,
        i=0,
        k=0,
        n1=0,
        n=0,
        t=0,
        t1=0;

char    mask1 = 0xf0,
        mask2 = 0xf;

FILE    *input,
        *output;

main()
{
    int     ch=0;

    printf("Entre com o arquivo (formato ASCII). Ex: PACK.DAT :");
    scanf("%13s",arquivo);

    if ( (input  = fopen(arquivo,"r")) == NULL )
    {
        printf("ERRO: Impossivel abrir arquivo %s para leitura\n",arquivo);
        exit(-1);
    }
```

```
    if ( (output = fopen("packc.dat","w")) == NULL )
    {
        printf("ERRO: Impossivel abrir arquivo PACKC.DAT para  escrita\n");
        exit(-2);
     }

    while( fgets(buffer,500,input) != NULL )
    {
        n = strlen(buffer);
        halfbyte();
        tally();
    }
    estatistic();
}

halfbyte()
{
    unsigned   char  a,
                     b;

    k=0;
    j=0;

    for(i=0;i < n;i+=2)
    {
        c[i]      = 0;
        c[i+1]    = 0;

        a = buffer[i];
        b = buffer[i+1];

        switch( a )
        {
            case '$' : c[i] = 1; break;
            case ',' : c[i] = 2; break;
            case '.' : c[i] = 3; break;
            case '*' : c[i] = 4; break;
        }
        if ( a >= '0' && a <= '9' )
        {
            c[i] = 5;
        }

        switch( b )
        {
            case '$' : c[i+1] = 1; break;
```

```
            case ',' : c[i+1] = 2; break;
            case '.' : c[i+1] = 3; break;
            case '*' : c[i+1] = 4; break;
        }
        if ( b >= '0' && b <= '9' )
        {
            c[i+1] = 5;
        }
        if ( c[i] == 0 || c[i+1] == 0 )
        {
            if ( k > 3 )
            {
                encode();
            }
            if ( k > 0 )
            {
                notencode();
            }
            o[j]     = buffer[i];
            o[j+1]   = buffer[i+1];
            j+=2;
            k=0;
            continue;
        }
        k+=2;
    }
}

encode()
{
    char    aux[500];

    int     x=0,
            y=0,
            z=0,
            ll=0,
            l=0;

    o[j]     = 129;
    o[j+1]   = k;
    j+=2;

    for (l=(i-k);l<=k;l+=2)
    {
        switch( c[l] )
        {
```

```
            case 1 : x = 0xa0;              break;
            case 2 : x = 0xb0;              break;
            case 3 : x = 0xc0;              break;
            case 4 : x = 0xd0;              break;
            case 5 : x = (buffer[l]<<4);    break;
        }
        x=x & mask1;

        switch( c[l+1] )
        {
            case 1 : y = 0xa;           break;
            case 2 : y = 0xb;           break;
            case 3 : y = 0xc;           break;
            case 4 : y = 0xd;           break;
            case 5 : y = buffer[l+1];   break;
        }
        y = y & mask2;
        z = x | y;
        o[j] = z;
        j++;
    }
    k=0;
}


notencode()
{
    int    l;

    for(l=1-k+1;l<=k;l++)
    {
        o[j]=buffer[l];
        j++;
    }
    k=0;
}

tally()
{
    int       i=0;

    n1=n1+(n-1);
    t=n-j+1;
    t1=t1+t;
    for(i=0;i<j-1;i++)
    {
```

```
            fputc(o[i],output);
        }
}

estatistic()
{
    fclose(input);
    fclose(output);

    if ( (input  = fopen(arquivo,"r")) == NULL )
    {
        printf("ERRO: Impossivel abrir arquivo %s para leitura\n",arquivo);
        exit(-1);
    }

    printf("Arquivo %s antes da compressao:\n",arquivo);

    while( fgets(buffer,500,input) != NULL )
    {
        printf("%s",buffer);
    }

    fclose(input);

    if ( (input = fopen("packc.dat","r")) == NULL )
    {
        printf("ERRO: Impossivel abrir arquivo PACKC.DAT para leitura\n");
        exit(-2);
    }

    printf("\n\nArquivo %s depois da compressao:\n",arquivo);

    while( fgets(buffer,500,input) != NULL )
    {
        printf("%s",buffer);
    }

    fclose(input);


    printf("%d Total de caracteres eliminados de %d ou %d%%\n\n",
                t1,n1,(int)((float)((float)t1/(float)n1)*100));
}
```

PackD.c

```
#include <stdio.h>

char    o[133],
        arquivo[13],
        buffer[500];

int     j=0,
        k=0,
        n1=0,
        m=0,
        n=0,
        t=0,
        t1=0;

char    mask1 = 0xf0,
        mask2 = 0xf;

FILE    *input,
        *output;

main()
{
    int  ch=0;

    printf("Entre com o arquivo. Ex: PACKC.DAT :");
    scanf("%13s",arquivo);

    if ( (input  = fopen(arquivo,"r")) == NULL )
    {
        printf("ERRO: Impossivel abrir arquivo %s para leitura\n",arquivo);
        exit(-1);
    }
    if ( (output = fopen("packd.dat","w")) == NULL )
    {
        printf("ERRO: Impossivel abrir arquivo PACKD.DAT para escrita\n");
        exit(-2);
    }

    while( fgets(buffer,500,input) != NULL )
    {
        n = strlen(buffer);
        halfbyte();
        tally();
    }
    estatistic();
```

```
}

halfbyte()
{
    unsigned char   a,
                    z;

    int         l=0,
                x=0,
                y=0,
                i=0;

    j=0;

    for(i=0;i <= strlen(buffer);i++)
    {
        a = buffer[i];

        if ( a == 129 )
        {
            k   = buffer[i+1];
            m   = i+(k/2);
            l   = i+2;

            for(;l<=m;l++)
            {
                z = buffer[l];
                x = (z & mask1) >> 4;

                if (x < 10)
                {
                    o[j] = x+48;
                }
                else
                {
                    switch ( x )
                    {
                        case 10 : o[j] = '$'; break;
                        case 11 : o[j] = ','; break;
                        case 12 : o[j] = '.'; break;
                        case 13 : o[j] = '*'; break;
                    }
                }
                y = z & mask2;

                if ( y < 10 )
```

```
                {
                    o[j+1] = y+48;
                }
                else
                {
                    switch( y )
                    {
                        case 10 : o[j+1] = '$'; break;
                        case 11 : o[j+1] = ','; break;
                        case 12 : o[j+1] = '.'; break;
                        case 13 : o[j+1] = '*'; break;
                    }
                }
                j+=2;
            }
            i=m;
        }
        else
        {
            o[j] = a;
            j++;
        }
    }
}

tally()
{
    int     i=0;

    n1=n1+n;
    t=n-j+1;
    t1=t1-t;
    for(i=0;i<j-1;i++)
    {
        fputc(o[i],output);
    }
}

estatistic()
{
    fclose(input);
    fclose(output);

    if ( (input  = fopen(arquivo,"rb")) == NULL )
    {
        printf("ERRO: Impossivel abrir arquivo %s para leitura\n",arquivo);
```

```
        exit(-1);
    }

    printf("Arquivo %s antes da descompressao:\n",arquivo);

    while( fgets(buffer,500,input) != NULL )
    {
        printf("%s",buffer);
    }

    fclose(input);

    if ( (input = fopen("packd.dat","rb")) == NULL )
    {
        printf("ERRO: Impossivel abrir arquivo PACKD.DAT para leitura\n");
        exit(-2);
    }

    printf("\n\nArquivo %s depois da descompressao:\n",arquivo);

    while( fgets(buffer,500,input) != NULL )
    {
        printf("%s",buffer);
    }

    fclose(input);

    printf("%d Total de caracteres inseridos\n\n",t1);
}
```

PairC.c

```
#include <stdio.h>

char    o[133],
        p[][3]= {   "E "," T","TH"," A","S ",
                    "RE","IN","HE","ER"," I",
                    " O","N ","ES"," B","ON",
                    "T ","TI","AN","D ","AT",
                    "TE"," C"," S","OR","R "
                },
        arquivo[13],
        buffer[500],
        *mids(char *,int,int);
```

```
int     t1=0,
        t=0,
        n=0,
        n1=0;


FILE    *input,
        *output;

main()
{
    int  ch=0;

    printf("Entre com o arquivo (formato ASCII). Ex: PAIR.DAT :");
    scanf("%13s",arquivo);

    if ( (input  = fopen(arquivo,"rb")) == NULL )
    {
        printf("ERRO: Impossivel abrir arquivo %s para leitura\n",arquivo);
        exit(-1);
    }
    if ( (output = fopen("pairc.dat","wb")) == NULL )
    {
        printf("ERRO: Impossivel abrir arquivo PAIRC.DAT para escrita\n");
        exit(-2);
    }

    while( fgets(buffer,500,input) != NULL )
    {
        n = strlen(buffer);
        tally( diatomic() );
    }
    estatistic();
}

diatomic()
{
    int     i=0,
            k=0,
            l=0,
            lk=0,
            flag=0;

    unsigned  char  a[3];

.   lk = sizeof(p) / sizeof(p[0]);
```

```
    for(i=0;i < n;i++)
    {
        strcpy(a,mids(buffer,i,2));

        flag=0;
        for (k=0;k<lk;k++)
        {
            if (strcmp(a,p[k])==0)
            {
                o[l++] = k+225;
                i++;
                flag=1;
                break;
            }
        }
        if ( !flag )
        {
            o[l++] = a[0];
        }
    }
    return(l);
}

char    *mids(buf,pos,qtde)
char    *buf;
int     pos,
        qtde;
{
    int     x=0,
            z=0;

    char    aux[500];

    for(x=pos;z<qtde && buf[x];z++,x++)
    {
        aux[z] = buf[x];
    }
    aux[z] = 0;
    return(aux);
}


tally(j)
int  j;
{
```

```
    int         i=0;

    n1=n1+(n-1);
    t=n-j;
    t1=t1+t;
    for(i=0;i<j;i++)
    {
        fputc(o[i],output);
    }
}

estatistic()
{
    fclose(input);
    fclose(output);

    if ( (input  = fopen(arquivo,"rb")) == NULL )
    {
        printf("ERRO: Impossivel abrir arquivo %s para leitura\n",arquivo);
        exit(-1);
    }

    printf("Arquivo %s antes da compressao:\n",arquivo);

    while( fgets(buffer,500,input) != NULL )
    {
        printf("%s",buffer);
    }

    fclose(input);

    if ( (input = fopen("pairc.dat","rb")) == NULL )
    {
        printf("ERRO: Impossivel abrir arquivo PAIRC.DAT para leitura\n");
        exit(-2);
    }

    printf("\n\nArquivo %s depois da compressao:\n",arquivo);

    while( fgets(buffer,500,input) != NULL )
    {
        printf("%s",buffer);
    }

    fclose(input);
```

```
    printf("\n%d Total de caracteres eliminados de %d ou %d%%\n\n",
                t1,n1,(int)((float)((float)t1/(float)n1)*100));
}
```

PairD.c

```
#include <stdio.h>

unsigned    char  o[133],
            p[][3]= {    "E "," T","TH"," A","S ",
                         "RE","IN","HE","ER"," I",
                         " O","N ","ES"," B","ON",
                         "T ","TI","AN","D ","AT",
                         "TE"," C"," S","OR","R "
                       },
            arquivo[13],
            buffer[500];

int         t1=0,
            t=0,
            n=0,
            n1=0;


FILE        *input,
            *output;

main()
{
    int         ch=0;

    printf("Entre com o arquivo. Ex: PAIRC.DAT :");
    scanf("%13s",arquivo);

    if ( (input  = fopen(arquivo,"rb")) == NULL )
    {
        printf("ERRO: Impossivel abrir arquivo %s para leitura\n",arquivo);
        exit(-1);
    }
    if ( (output = fopen("paird.dat","wb")) == NULL )
    {
        printf("ERRO: Impossivel abrir arquivo PAIRD.DAT para escrita\n");
        exit(-2);
    }
```

```
    while( fgets(buffer,500,input) != NULL )
    {
        n = strlen(buffer);
        tally( diatomic() );
    }
    estatistic();
}

diatomic()
{
    int     i=0,
            l=0;

    for(i=0;i < n;i++)
    {
        if ( buffer[i] > 224 )
        {
            o[l++] = p[buffer[i]-225][0];
            o[l++] = p[buffer[i]-225][1];
        }
        else
        {
            o[l++] = buffer[i];
        }
    }
    return(l);
}


tally(j)
int  j;
{
    int     i=0;

    n1=n1+(n-1);
    t=n-j;
    t1=t1-t;
    for(i=0;i<j;i++)
    {
        fputc(o[i],output);
    }
}

estatistic()
{
    fclose(input);
```

```
    fclose(output);

    if ( (input  = fopen(arquivo,"rb")) == NULL )
    {
        printf("ERRO: Impossivel abrir arquivo %s para leitura\n",arquivo);
        exit(-1);
    }

    printf("Arquivo %s antes da descompressao:\n",arquivo);

    while( fgets(buffer,500,input) != NULL )
    {
        printf("%s",buffer);
    }

    fclose(input);

    if ( (input = fopen("paird.dat","rb")) == NULL )
    {
        printf("ERRO: Impossivel abrir arquivo PAIRD.DAT para leitura\n");
        exit(-2);
    }

    printf("\n\nArquivo %s depois da descompressao:\n",arquivo);

    while( fgets(buffer,500,input) != NULL )
    {
        printf("%s",buffer);
    }

    fclose(input);

    printf("\n%d Total de caracteres inseridos\n\n",t1);
}
```

Analiza.c

```
#include <stdio.h>

char    arquivo[13],
        pares[500][3],
        buffer[500],
        *mids(char *,int,int);

int     paresidx=0,
```

```
        ocorre[500],
        n=0;

FILE    *input,
        *output;

main()
{
    int     ch=0;

    for(ch=0;ch<500;ch++)
    {
        ocorre[ch]=0;
    }
    printf("Entre com o arquivo (formato ASCII). Ex: ANALIZA.DAT :");
    scanf("%13s",arquivo);

    if ( (input  = fopen(arquivo,"rb")) == NULL )
    {
        printf("ERRO: Impossivel abrir arquivo %s para leitura\n",arquivo);
        exit(-1);
    }

    while( fgets(buffer,500,input) != NULL )
    {
        n = strlen(buffer);
        analize();
    }
    estatistic();
}

analize()
{
    int     i=0,
            k=0,
            z=0,
            flag=0;

    unsigned  char  a[3];

    for(i=0;i < n-2;i++)
    {
        strcpy(a,mids(buffer,i,2));

        flag=0;
```

```
        for (k=0;k < paresidx && k < 500;k++)
        {
            if (strcmp(a,pares[k])==0)
            {
                ocorre[k]++;
                flag=1;
                break;
            }
        }
        if ( !flag )
        {
            if ( paresidx < 499 )
            {
                strcpy(pares[paresidx],a);
                ocorre[paresidx++] = 1;
            }
        }
    }
}

char    *mids(buf,pos,qtde)
char    *buf;
int     pos,
        qtde;
{
    int     x=0,
            z=0;

    char    aux[500];

    for(x=pos;z<qtde && buf[x];z++,x++)
    {
        aux[z] = buf[x];
    }
    aux[z] = 0;
    return(aux);
}


estatistic()
{
    FILE    *device;

    int     k=0,
            x=0,
            z=0,
            flag=0;
```

```
char     a[3];

fclose(input);

fflush(stdin);
printf("\n\nDeseja enviar relatorio para impressora ? (S/N) ");
scanf("%c",&x);

if ( x=='S' || x=='s' )
{
    device = stdprn;
}
else
{
    device = stdout;
}

printf("\nOrdenando...\n\n");

flag=1;
while(flag)
{
    flag=0;

    for(k=0;k<paresidx-1;k++)
    {
        if ( ocorre[k] < ocorre[k+1] )
        {
            x = ocorre[k];
            ocorre[k] = ocorre[k+1];
            ocorre[k+1] = x;

            strcpy(a,pares[k]);
            strcpy(pares[k],pares[k+1]);
            strcpy(pares[k+1],a);

            flag=1;
        }
    }
}

if ( (output=fopen("analizad.dat","w")) == NULL )
{
    printf("\nImpossivel abrir arquivo ANALIZAD.DAT para escrita\n\n");
    exit(0);
}
```

```
fprintf(device," ----------------------------------------------------
                ----------------------------------\n");
fprintf(device,"\n<< Relatorio da Analise  -  Arquivo : %s >>\n",arquivo);
fprintf(device," ----------------------------------------------------
                ----------------------------------\n\n");

fprintf(output," ----------------------------------------------------
                ----------------------------------\n");
fprintf(output,"\n<< Relatorio da Analise  -  Arquivo : %s >>\n",arquivo);
fprintf(output," ----------------------------------------------------
                ----------------------------------\n\n");

flag=0;
z=0;
for (k=0;k < paresidx;k++)
{
    if  ( pares[k][0]!=13 && \
          pares[k][1]!=13 && \
          pares[k][0]!=10 && \
          pares[k][1]!=10 && \
          pares[k][0]!=9  && \
          pares[k][1]!=9  && \
          pares[k][0]!=8  && \
          pares[k][1]!=8  && \
          ocorre[k] > 1)
    {
          if ( k > 29 && !flag)
          {
               flag=1;
               fprintf(device,"\n\n ----------------------------------------
                                  ----------------------------------\n");
               fprintf(device,"Uma vez que para uma compactacao pela analise de
                               pares, utilizamos codificacao\n");
               fprintf(device,"baseada nos codigos 225 a 255, so podemos utilizar os
                               30 primeiros grupos.\n");
               fprintf(device," ----------------------------------------------
                                  ----------------------------------------\n\n");
               fprintf(output,"\n\n ----------------------------------------
                                  ----------------------------------\n");
               fprintf(output,"Uma vez que para uma compactacao pela analise de
                               pares, utilizamos codificacao\n");
               fprintf(output,"baseada nos codigos 225 a 255, so podemos utilizar os
                               30 primeiros grupos.\n");
               fprintf(output," ----------------------------------------------
                                  ----------------------------------------\n\n");
```

```
                z=0;
            }
            fprintf(device,"%c%c = %03d  ",(pares[k][0] != 32) ? pares[k][0] :
                                '_',(pares[k][1] != 32) ? pares[k][1] : '_',ocorre[k]);
            fprintf(output,"%c%c = %03d  ",(pares[k][0] != 32) ? pares[k][0] :
                                '_',(pares[k][1] != 32) ? pares[k][1] : '_',ocorre[k]);
            z++;

            if (z > 5)
            {
                fprintf(device,"\n");
                fprintf(output,"\n");
                z=0;
            }
        }
    }


    fprintf(device,"\nObservacao: _ significa espaco em branco (ASCII 32)\n");
    fprintf(output,"\nObservacao: _ significa espaco em branco (ASCII 32)\n");
    fclose(output);
}
```

PairC2.c

```
/*
 * PAIRC2 - Compactador de arquivos
 * Carlos Augusto Pereira Gomes & Antonio Carlos Barbosa (1990).
 *
 * Analiza melhores combinacoes para compactacao. Nao é eficiente em arquivos
 * muito pequenos, alem de nao compactar arquivos com caracteres de valor 225 a 255
 */

#include <stdio.h>
#include <string.h>

unsigned    char        o[32000],
                        p[30][3],
                        pares[500][3],
                        arquivo[13],
                        buffer[501],
                        anima[] = { '|','/','-','\\' },
                        anndx=0;
```

```
long  ocorre[500];

int       paresidx=0,
          n=0;

FILE      *input,
          *output;

main()
{
     int  ch=0,
          z=0;

     for(ch=0;ch<500;ch++)
     {
          ocorre[ch]=0;
     }

     printf("Entre com o arquivo (formato ASCII). Ex: PAIR2.DAT :");
     scanf("%13s",arquivo);

     if ( (input  = fopen(arquivo,"rb")) == NULL )
     {
          printf("ERRO: Impossivel abrir arquivo %s para leitura\n",arquivo);
          exit(-1);
     }
     if ( (output = fopen("pairc2.dat","wb")) == NULL )
     {
          printf("ERRO: Impossivel abrir arquivo PAIRC2.DAT para escrita\n");
          exit(-2);
     }

     printf("\nAnalizando...");

     while((n=fread((unsigned char*)buffer,sizeof(unsigned char),500,input))>0)
     {
          analize();
     }

     printf("\rPreparando...");

     prepare();

     rewind(input);

     printf("\rCompactando...");
```

```
    while((n=fread((unsigned char*)buffer,sizeof(unsigned char),500,input))>0)
    {
        memset((unsigned char *) o,0,32000);
        tally( diatomic() );
        memset((unsigned char *) buffer,0,500);
    }
    estatistic();
}

analize()
{
    long    i=0,
            z=0,
            flag=0;

    int     k=0;

    unsigned char a[3];

    mstanima();

    for(i=0;i < n;i++)
    {
        if ( i+2 < n )
        {
            a[0] = buffer[i];
            a[1] = buffer[i+1];
            a[2] = 0;

            flag=0;

            if ( a[0] > 224 || a[1] > 224 )
            {
                printf("\rArquivo possui caracteres ASCII com valores acima de
                        224.\n");
                exit(-1);
            }

            for (k=0;k < paresidx && k < 500;k++)
            {
                if (strcmp(a,pares[k])==0)
                {
                    ocorre[k]++;
                    i++;
                    flag=1;
```

```
                    break;
                }
            }
            if ( !flag )
            {
                if ( paresidx < 499 )
                {
                    strcpy(pares[paresidx],a);
                    ocorre[paresidx++] = 1;
                }
            }
        }
    }
}

prepare()
{
    long    x=0,
            z=0,
            flag=0;

    int     k=0,
            kl=0;

    char    a[3];

    long    pos=0L;

    flag=1;
    while(flag)
    {
        flag=0;

        mstanima();

        for(k=0;k<paresidx-1;k++)
        {
            if ( ocorre[k] < ocorre[k+1] )
            {
                x = ocorre[k];
                ocorre[k] = ocorre[k+1];
                ocorre[k+1] = x;

                strcpy(a,pares[k]);
                strcpy(pares[k],pares[k+1]);
                strcpy(pares[k+1],a);
```

```
                flag=1;
            }
        }
    }

    flag=0;
    z=0;

    fprintf(output,"[%c",32);
    for (k=0,kl=0;k < 29 && k < paresidx;k++)
    {
        if  (   pares[k][0]!=13 &&
                pares[k][1]!=13 &&
                pares[k][0]!=10 &&
                pares[k][1]!=10 &&
                ocorre[k] > 1)
        {
            fprintf(output,"%s",pares[k]);
            strcpy(p[kl],pares[k]);
            kl++;
        }
    }
    fprintf(output,"%c]",255);

    fgetpos(output,&pos);
    rewind(output);
    fprintf(output,"[%c",kl);
    fsetpos(output,&pos);

    paresidx = kl;
}

diatomic()
{
    int     k=0,
            i=0,
            l=0,
            flag=0;


    unsigned char a[3];

    mstanima();

    for(i=0;i < n;i++)
    {
```

```
        if ( i + 2  < n )
        {
            a[0] = buffer[i];
            a[1] = buffer[i+1];
            a[2] = 0;

            flag=0;
            for (k=0;k<paresidx;k++)
            {
                if (strcmp(a,p[k])==0)
                {
                    o[l++] = k+225;
                    i++;
                    flag=1;
                    break;
                }
            }
            if ( !flag )
            {
                o[l++] = a[0];
            }
            if ( l > 31500 )
            {
                tally( l );
                l=0;
            }
        }
        else
        {
            o[l++] = buffer[i];
            if ( l > 31500 )
            {
                tally( l );
                l=0;
            }
        }
    }
    return(l);
}

tally(j)
int  j;
{
    int i=0;

    for(i=0;i<j;i++)
```

```
    {
        fputc(o[i],output);
    }
}

estatistic()
{
    float   x,
            y;

    long    bytes=0L;

    fflush(output);
    fgetpos(input,&bytes);
    x = bytes;
    fgetpos(output,&bytes);
    y = bytes;

    fclose(input);
    fclose(output);

    printf("\n%6.0f Total de caracteres eliminados de %6.0f ou %3.2f%%\n\n",
            x-y,x,((x-y)*100)/x);
}

/*
 * Esta rotina simplesmente cria a sensacao de funcionamento do programa,
 * servindo para, em maquinas muito lentas operando com arquivos muito
 * extensos, informar que a operacao solicitada esta' em andamento. Note
 * porem, que ela reduz drasticamente a velocidade de compactacao ou
 * descompactacao.
 */

mstanima()
{
    printf("%c\b",anima[anndx]);
    anndx = (anndx > 2) ? 0 : anndx + 1;
}
```

PairD2.c

```
/*
 * PAIRD2 - Descompactador de arquivos
 * Carlos Augusto Pereira Gomes & Antonio Carlos Barbosa(1990).
 */
```

```
#include <stdio.h>
#include <string.h>

unsigned char   o[32000],
                p[30][3],
                arquivo[13],
                buffer[501],
                anima[] = { '|','/','-','\\' },
                anndx=0;

int     n=0,
        paresidx=0;

FILE    *input,
        *output;

main()
{
    int     ch=0,
            z=0;

    printf("Entre com o arquivo. Ex: PAIRC2.DAT :");
    scanf("%13s",arquivo);

    if ( (input  = fopen(arquivo,"rb")) == NULL )
    {
        printf("ERRO: Impossivel abrir arquivo %s para leitura\n",arquivo);
        exit(-1);
    }
    if ( (output = fopen("paird2.dat","wb")) == NULL )
    {
        printf("ERRO: Impossivel abrir arquivo PAIRD2.DAT para escrita\n");
        exit(-2);
    }

    printf("\nAnalizando...");

    analize();

    printf("\rDescompactando...");

    while((n=fread((unsigned char*) buffer,sizeof(unsignedchar),500,input))>0)
    {
        memset((unsigned char *) o,0,32000);
        tally( diatomic() );
```

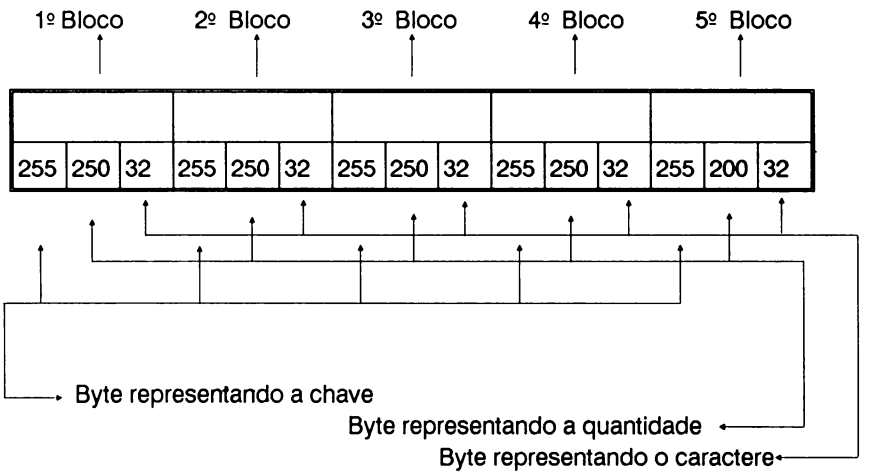
```
            memset((unsigned char *) buffer,0,500);
        }
        estatistic();
}

analize()
{
    int     ch=0,
            ch1=0,
            z=0;

    ch  = fgetc(input);
    ch1 = fgetc(input);

    if ( ch=='[' )
    {
        paresidx=ch1;
        for (z=0;z<ch1;z++)
        {
            p[z][0] = fgetc(input);
            p[z][1] = fgetc(input);
            p[z][2] = 0;
        }
        ch  = fgetc(input);
        ch1 = fgetc(input);

        if ( ch==255 && ch1 == ']' )
        {
            return;
        }
    }
    printf("\rArquivo nao foi compactado utilizando PAIRC2\n");
    exit(-1);
}

diatomic()
{
    int     i=0,
            l=0;

    mstanima();

    for(i=0;i < n;i++)
    {
        if ( buffer[i] > 224  &&  buffer[i]-225 < paresidx )
        {
```

```
                o[l++] = p[buffer[i]-225][0];
                o[l++] = p[buffer[i]-225][1];
            }
            else
            {
                o[l++] = buffer[i];
            }
            if ( l > 31500 )
            {
                tally(l);
                l=0;
            }
        }
        return(l);
}


tally(j)
int  j;
{
        int     i=0;

        for(i=0;i<j;i++)
        {
            fputc(o[i],output);
        }
}

estatistic()
{
        long    newbytes=0L,
                oldbytes=0L;

        fflush(output);
        fgetpos(input,&oldbytes);
        fgetpos(output,&newbytes);

        fclose(input);
        fclose(output);

        printf("\r%ld Total de caracteres inseridos\n\n",newbytes-oldbytes);
}

/*
 * Esta rotina simplesmente cria a sensacao de funcionamento do programa,
 * servindo para, em maquinas muito lentas operando com arquivos muito
```

```
 * extensos, informar que a operacao solicitada esta' emandamento. Note
 * porem, que ela reduz drasticamente a velocidade decompactacao ou
 * descompactacao.
 */

mstanima()
{
    printf("%c\b",anima[anndx]);
    anndx = (anndx > 2) ? 0 : anndx + 1;
}
```

PkFile.c

Este programa permite compactar/descompactar arquivos, utilizando a técnica de 3 bytes. Ele primeiramente analisa o arquivo, procurando por um caractere não utilizado(inicia com o valor 255). Encontrando um caractere que não foi usado em todo o arquivo, inicia o processo de compactação, procurando por caracteres repetidos. Quando encontrado, vai somando até localizar um caractere diferente; gravando em blocos de até 250 bytes, ou seja, se ele contar por exemplo, 1200 espaços-em-branco, irá gravar em 4 blocos de 250 e 1 bloco de 200, cada bloco terá o tamanho de 3 bytes, assim sendo, transformamos 1.200 bytes em apenas 15 bytes. O diagrama a seguir, exemplifica os blocos gravados deste nosso exemplo, considerando que o caractere que foi utilizado como chave, é o 255:




```
/*
 * PKFILE - Compactador/Descompactador de arquivos
 * Carlos Augusto Pereira Gomes & Antonio Carlos Barbosa(1990).
 *
 * Eficiente para compactacao de arquivos .DBF ou arquivos com muitas
 * incidencias de caracteres repetidos.
 */
```

```
#include <stdio.h>
#include <math.h>
#include <dos.h>
#include <string.h>
#include <stdlib.h>

char    anima[] = { '|','/','-','\\' },
        anndx=0,
        linha=0;

main(argc,argv)
int     argc;
char    *argv[];
{
    if ( argc < 2 )
    {
        printf("\nUso: pkfile arquivo.ext operacao\n\n");
        printf("onde: arquivo.ext e o arquivo a ser compactado/descompactado\n");
        printf("      operacao e' /C para compactar ou /D para descompactar.\n\n");
        exit(-1);
    }

    argv[2][1] = toupper(argv[2][1]);

    if ( argv[2][1] != 'C' && argv[2][1] != 'D' )
    {
        printf("Operacoes validas: /C - Compactar  ou  /D -Descompactar\n\n");
        exit(-2);
    }

    switch( argv[2][1] )
    {
        case  'C' :   compactar(argv[1]);
                      break;

        case  'D' :   descompactar(argv[1]);
                      break;
    }
}

compactar(arquivo)
char *arquivo;
{
    FILE    *input,
            *output;
```

```
int     ler=0,
        x=0,
        z=0,
        analize=0,
        qtde=0;

long  oldbytes=0;

float  total    bytes=0,
       bytes=0;

unsigned    char     ch=255;

if ( (input = fopen(arquivo,"rb")) == NULL )
{
    printf("\nImpossivel abrir arquivo %s para leitura\n",arquivo);
    exit(-1);
}

ler      = fgetc(input);
analize  = fgetc(input);
x        = fgetc(input);

if ( ler == '[' && x == ']' )
{
    printf("\nArquivo ja compactado anteriormente. Nada ha fazer.\n\n");
    exit(-1);
}

rewind(input);

/*
 * Procede com a analise dos caracteres utilizados no arquivo.
 * Inicialmente define a chave como 255 e procura ocorrencias deste mesmo
 * caractere pelo arquivo. Encontrando, decrementa a chave e reinicia a
 * pesquisa ate' encontrar um caractere nao utilizado no arquivo. Se nao
 * encontrar, nao compacta. Isto pode ser alterado, criando uma chave
 * mais complexa, tipo CHAVE+QTDE+CARACTER+CHAVE e alterando a rotina
 * de descompactacao para procurar por blocos deste tipo.
 */

printf("\nAnalizando... ");

while( (ler=fgetc(input)) != EOF )
{
```

```
        mstanima();
        if (ler == ch)
        {
            ch--;
            if (ch < 0 )
            {
                printf("\narquivo utiliza todos os caracteres da tabela ASCII.\n");
                printf("impossivel criar chave para compactacao.\n");
                fclose(input);
                exit(-1);
            }
            rewind(input);
        }
    }

    if ( (output=fopen("pkfl.tmp","wb")) == NULL )
    {
        printf("Impossivel abrir arquivo temporario\n");
        exit(-1);
    }

    fprintf(output,"[%c]",ch);

    /*
     * Salva compactando
     */

    printf("\rCompactando... ");

    rewind(input);
    analize = fgetc(input);
    qtde=1;

    while( (ler=fgetc(input)) != EOF )
    {
        if ( analize != ler )
        {
            if ( qtde > 3 )
            {
                mstanima();
                for(x=250;x > 0 && qtde > 0; x-- )
                {
                    while ( qtde >= x )
                    {
                        fprintf(output,"%c%c%c",ch,x,analize);
                        qtde-=x;
```

```
                    }
                }
            }
            else
            {
                for(;qtde>0;qtde--)
                {
                    fprintf(output,"%c",analize);
                }
            }
            analize = ler;
            qtde=1;
        }
        else
        {
            qtde++;
        }
    }
    mstanima();
    for(x=250;x > 0 && qtde > 0; x-- )
    {
        while ( qtde >= x )
        {
            fprintf(output,"%c%c%c",ch,x,analize);
            qtde-=x;
        }
    }
    fgetpos(input,&oldbytes);
    bytes = oldbytes;
    fgetpos(output,&oldbytes);
    total_bytes = oldbytes;

    printf("\r%6.0f bytes eliminados de %6.0f ou(%3.2f%%)\n",
            bytes-total_bytes,bytes,((bytes- total_bytes)*100)/bytes);
    fclose(input);
    fclose(output);
    unlink(arquivo);
    rename("pkfl.tmp",arquivo);
}


descompactar(arquivo)
char *arquivo;
{
    FILE    *input,
            *output;
```

```
int     ler=0,
        analize=0,
        qtde=0;

long    total_bytes=0,
        bytes=0;

unsigned    char    ch;

if ( (input = fopen(arquivo,"rb")) == NULL )
{
    printf("\nImpossivel abrir arquivo %s paraleitura\n",arquivo);
    exit(-1);
}

analize = fgetc(input);    /* [ descartado */

if ( analize != '[' )
{
    printf("Arquivo nao foi previamente compactado por PKFILE\n");
    exit(-1);
}

ch      = fgetc(input);   /* chave */

analize = fgetc(input);    /* ] descartado */

if ( analize != ']' )
{
    printf("Arquivo nao foi previamente compactado por PKFILE\n");
    exit(-1);
}

if ( (output=fopen("pkfl.tmp","wb")) == NULL )
{
    printf("Impossivel abrir arquivo temporario\n");
    exit(-1);
}

printf("\nProcessando... ");

while( (ler=fgetc(input)) != EOF )
{
    if ( ler != ch )
    {
```

```
            fprintf(output,"%c",ler);
        }
        else
        {
            qtde    = fgetc(input);
            analize = fgetc(input);
            mstanima();

            while ( qtde )
            {
                qtde--;
                fprintf(output,"%c",analize);
            }
        }
    }
    fgetpos(input,&bytes);
    fgetpos(output,&total_bytes);

    printf("\r%ld bytes inseridos\n",total_bytes-bytes);
    fclose(input);
    fclose(output);
    unlink(arquivo);
    rename("pkfl.tmp",arquivo);
}

/*
 * Esta rotina simplesmente cria a sensacao de funcionamento do programa,
 * servindo para, em maquinas muito lentas operando com arquivos muito extensos,
 * informar que a operacao solicitada esta' em andamento. Note porem, que ela
 * reduz drasticamente a velocidade de compactacao ou descompactacao.
 */

mstanima()
{
    printf("%c\b",anima[anndx]);
    anndx = (anndx > 2) ? 0 : anndx + 1;
}
```

SavePk.c

```
/*
 * SAVEPK.C - Exemplo de Salvamento de tela de forma compactada em C
 * Clipper 5.0, o que mudou...
 * Carlos Augusto P.Gomes & Antonio Carlos Barbosa (1991)
 *
```

```
* Compilar : cl savepk.c
*
* NOTA: Microsoft C, versões 5.00 ou 5.10
*/


#include <stdio.h>
#include <math.h>
#include <dos.h>
#include <string.h>
#include <stdlib.h>
#include <graph.h>

#define     VIDEO       0xB800

unsigned char *save_scr_pack( int,int,int,int,);
static int rest_scr_pack( char * );

static   int  pos_data;
char     fa   *p;
char          area[160];

main(argc,argv)
int      argc;
char     *argv[];
{
    unsigned     char  *slvtel;

    int          i=0,
                 linha1,coluna1,linha2,coluna2;

    if ( argc < 4 )
    {
        printf("\nUso: savepk linha1 coluna1 linha2 coluna2\n\n");
        exit(-1);
    }

    linha1   = atoi(argv[1]);
    coluna1 = atoi(argv[2]);
    linha2   = atoi(argv[3]);
    coluna2 = atoi(argv[4]);

    slvtel = save_scr_pack(linha1,coluna1,linha2,coluna2);

    _clearscreen(_GCLEARSCREEN);
    getch();
```

```
    rest_scr_pack(slvtel);
}

unsigned char *save_scr_pack(linha1,coluna1,linha2,coluna2)
int     linha1,
        coluna1,
        linha2,
        coluna2;
{
    register  i=0,
              j=0;

    int       total=0,
              flag_mudou=1,
              qtde=0,
              x=0,
              total_bytes=0;

    unsigned char   *aux,
                    *tela,
                    *trabalho;

    unsigned char   ch=255,
                    atr=254,
                    analize=0;

    trabalho = (char *) malloc (sizeof(char)*4000);

    if ( trabalho == NULL )
    {
        return(0);
    }

    p   = area;
    aux = trabalho;

    total = (coluna2-coluna1)+1;

    /*
     * Analiza os caracteres e atributos para descobrir qual nao esta' em uso
     */

    while( flag_mudou )
    {
        flag_mudou = 0;
```

```
    for(i=linha1;i<=linha2;i++)
    {
        pos_data = i * 160 + coluna1 * 2;
        movedata( VIDEO, pos_data, FP_SEG(p), FP_OFF(p),(total)*2 );

        for(j=0;j<(total*2);)
        {
            if ( area[j++] == atr )     /* analiza atributo */
            {
                atr--;
                flag_mudou=1;
            }

            if ( area[j++] == ch )      /* analiza caracter */
            {
                ch--;
                flag_mudou=1;
            }
        }
        if ( atr == ch )
        {
            atr--;
            flag_mudou=1;
        }
    }
}

/*
 * CH e' o caractere que sera' utilizado como flag de compactacao para caractere
 * ATR e' o caractere que sera' utilizado como flag de compactacao para
 * atributos
 */

*aux++ = ch;
*aux++ = atr;
*aux++ = linha1;
*aux++ = coluna1;
*aux++ = linha2;
*aux++ = coluna2;
total_bytes += 6;

/*
 * Salva compactando
 */
```

```
qtde=0;
for(i=linha1;i<=linha2;i++)
{
    pos_data = i * 160 + coluna1 * 2;
    movedata( VIDEO, pos_data, FP_SEG(p), FP_OFF(p),(total)*2 );

    if (i==linha1)
    {
        analize = area[0];
    }

    for(j=0;j<(total*2);j+=2)
    {
        if ( analize != area[j] )      /* compacta caracter */
        {
            if ( qtde > 3 )
            {
                for(x=250;x > 0 && qtde > 0; x-- )
                {
                    while ( qtde >= x )
                    {
                        *aux++ = ch;
                        *aux++ = x;
                        *aux++ = analize;
                        total_bytes+=3;
                        qtde-=x;
                    }
                }
            }
            else
            {
                for(;qtde>0;qtde--)
                {
                    *aux++ = analize;
                    total_bytes++;
                }
            }
            analize = area[j];
            qtde=1;
        }
        else
        {
            qtde++;
        }
    }
}
```

```
for(x=250;x > 0 && qtde > 0; x-- )
{
    while ( qtde >= x )
    {
        *aux++ = ch;
        *aux++ = x;
        *aux++ = analize;
        total_bytes+=3;
        qtde-=x;
    }
}

qtde=0;
for(i=linha1;i<=linha2;i++)
{
    pos_data = i * 160 + coluna1 * 2;
    movedata( VIDEO, pos_data, FP_SEG(p), FP_OFF(p),(total)*2 );

    if ( i==linha1 )
    {
        analize = area[1];
    }

    for(j=1;j<(total*2);j+=2)
    {
        if ( analize != area[j] )      /* compacta atributo */
        {
            if ( qtde > 3 )
            {
                for(x=250;x > 0 && qtde > 0; x-- )
                {
                    while ( qtde >= x )
                    {
                        *aux++ = atr;
                        *aux++ = x;
                        *aux++ = analize;
                        total_bytes+=3;
                        qtde-=x;
                    }
                }
            }
            else
            {
                for(;qtde>0;qtde--)
                {
                    *aux++ = analize;
```

```
                        total_bytes++;
                    }
                }
                analize = area[j];
                qtde=1;
            }
            else
            {
                qtde++;
            }
        }
    }
    for(x=250;x > 0 && qtde > 0; x-- )
    {
        while ( qtde >= x )
        {
            *aux++ = atr;
            *aux++ = x;
            *aux++ = analize;
            total_bytes+=3;
            qtde-=x;
        }
    }

    tela = (char *) malloc (sizeof(char)*total_bytes);

    if ( tela == NULL )
    {
        return(trabalho);
    }
    for(i=0;i<total_bytes;i++)
    {
        tela[i] = trabalho[i];
    }
    free(trabalho);
    return(tela);
}



static int rest_scr_pack(tela)
char *tela;
{
    register i=0,
             j=0;
```

```
int     total=0,
        qtde=0,
        linha1=0,
        linha2=0,
        coluna1=0,
        coluna2=0;

unsigned char *aux;

unsigned  char   ch=0,
                 atr=0,
                 analize=0;

p   = area;
aux = tela;

/*
 * CH e' o caractere que sera' utilizado como flag de descompactacao para
 * caractere.
 * ATR e' o caractere que sera' utilizado como flag de descompactacao para
 * atributos
 */

ch      = *aux++;
atr     = *aux++;
linha1  = *aux++;
coluna1 = *aux++;
linha2  = *aux++;
coluna2 = *aux++;

total = (coluna2-coluna1)+1;

/*
 * Restaura descompactando
 */

for(i=linha1;i<=linha2;)
{
    for(j=0;j<(total*2) && i <= linha2;)
    {
        if ( *aux != ch )
        {
            area[j++] = *aux;
            area[j++] = 7;
            aux++;
        }
        else
```

```
        {
            aux++;
            qtde = *aux++;
            analize = *aux++;

            while( qtde )
            {
                qtde--;
                area[j++]   = analize;
                area[j++] = 7;

                if ( j >= (total*2) )
                {
                    pos_data = i * 160 + coluna1* 2;
                    movedata( FP_SEG(p), FP_OFF(p), VIDEO,pos_data, (total)*2 );
                    i++;
                    j=0;
                }
            }
        }
    }
    if ( i < linha2 )
    {
            pos_data = i * 160 + coluna1 * 2;
        movedata( FP_SEG(p), FP_OFF(p), VIDEO, pos_data,(total)*2 );
        i++;
    }
}

qtde=0;

for(i=linha1;i<=linha2;)
{
    pos_data = i * 160 + coluna1 * 2;
    movedata( VIDEO, pos_data, FP_SEG(p), FP_OFF(p),(total)*2 );

    for(j=1;j<(total*2) && i <= linha2;)
    {
        if ( *aux != atr )
        {
            area[j] = *aux++;
            j+=2;
        }
        else
        {
            aux++;
```

```
                qtde = *aux++;
                analize = *aux++;

                while(qtde)
                {
                    qtde--;
                    area[j] = analize;
                    j+=2;
                    if ( j >= (total*2) )
                    {
                        j=1;
                        pos_data = i * 160 + coluna1 * 2;
                        movedata( FP_SEG(p), FP_OFF(p), VIDEO,pos_data, (total)*2 );
                        i++;
                        pos_data = i * 160 + coluna1 * 2;
                        movedata( VIDEO, pos_data, FP_SEG(p),FP_OFF(p), (total)*2 );
                    }
                }
            }
        }
        if ( i < linha2 )
        {
            pos_data = i * 160 + coluna1 * 2;
            movedata( FP_SEG(p), FP_OFF(p), VIDEO, pos_data,(total)*2 );
            i++;
        }
    }
}
```

Para usuários CLIPPER, duas funções que permitem o salvamento e recuperação de telas. Estas funções assemelham-se às SaveScreen() e RestScreen() disponíveis no próprio Clipper, sendo que a grande vantagem, é que além de salvarem os dados da tela de forma compactada, na restauração não é necessário informar as coordenadas originais.

Pk_Scr.c

```
/*
 * PK_SCR.C - Salva a tela de forma compactada
 * Clipper 5.0, o que mudou...
 * Carlos Augusto P.Gomes & Antonio Carlos Barbosa (1991)
 *
 * Compilar : cl -c -AL -Zl -Oalt -Gs -FPa  pk_scr.c
 *
```

```
 * Linkeditar:  link /NOE /SEG:340testepksc+pk_scr,,,clipper extend llibca
 *
 * NOTA: Microsoft C, versões 5.00 ou 5.10
 */

#include <stdio.h>
#include <dos.h>
#include <string.h>
#include "nandef.h"
#include "extend.h"

#define     VIDEO       0xB800

static   int  pos_data;
char     far  *p;
char          area[160];

CLIPPER pk_svscr()
{
     register  i=0,
               j=0;

     int       total=0,
               flag_mudou=1,
               qtde=0,
               x=0,
               total_bytes=0,
               linha1,
               linha2,
               coluna1,
               coluna2;

     unsigned char    *aux,
                      *tela,
                      *trabalho;

     unsigned    char     ch=255,
                          atr=254,
                          analize=0;

     if ( PCOUNT < 4 )
     {
          _retclen(0,0);
          return;
     }
```

```
trabalho = (char *) _exmgrab(sizeof(char)*4000);

if ( trabalho == NULL )
{
    _retclen(0,0);
    return;
}

linha1   = _parni(1);
coluna1 = _parni(2);
linha2   = _parni(3);
coluna2 = _parni(4);

p    = area;
aux = trabalho;

total = (coluna2-coluna1)+1;

/*
 * Analiza os caracteres e atributos para descobrir qual nao esta' em uso
 */

while( flag_mudou )
{
    flag_mudou = 0;

    for(i=linha1;i<=linha2;i++)
    {
        pos_data = i * 160 + coluna1 * 2;
        movedata( VIDEO, pos_data, FP_SEG(p), FP_OFF(p),(total)*2 );

        for(j=0;j<(total*2);)
        {
            if ( area[j++] == atr )     /* analiza atributo */
            {
                atr--;
                flag_mudou=1;
            }

            if ( area[j++] == ch )      /* analiza caracter */
            {
                ch--;
                flag_mudou=1;
            }
        }
        if ( atr == ch )
```

```
        {
            atr--;
            flag_mudou=1;
        }
    }
}

/*
 * CH e' o caractere que sera' utilizado como flag de compactacao para caracter
 * ATR e' o caractere que sera' utilizado como flag de compactacao para
 * atributos
 */

*aux++ = ch;
*aux++ = atr;
*aux++ = linha1;
*aux++ = coluna1;
*aux++ = linha2;
*aux++ = coluna2;
total_bytes += 6;

/*
 * Salva compactando
 */

qtde=0;
for(i=linha1;i<=linha2;i++)
{
        pos_data = i * 160 + coluna1 * 2;
    movedata( VIDEO, pos_data, FP_SEG(p), FP_OFF(p),(total)*2 );

    if (i==linha1)
    {
        analize = area[0];
    }

    for(j=0;j<(total*2);j+=2)
    {
            if ( analize != area[j] )      /* compacta caracter*/
            {
                if ( qtde > 3 )
                {
                    for(x=250;x > 0 && qtde > 0; x-- )
                    {
                        while ( qtde >= x )
                        {
```

```
                        *aux++ = ch;
                        *aux++ = x;
                        *aux++ = analize;
                        total_bytes+=3;
                        qtde-=x;
                    }
                }
            }
            else
            {
                for(;qtde>0;qtde--)
                {
                    *aux++ = analize;
                    total_bytes++;
                }
            }
            analize = area[j];
            qtde=1;
        }
        else
        {
            qtde++;
        }
    }
}
for(x=250;x > 0 && qtde > 0; x-- )
{
    while ( qtde >= x )
    {
        *aux++ = ch;
        *aux++ = x;
        *aux++ = analize;
        total_bytes+=3;
        qtde-=x;
    }
}

qtde=0;
for(i=linha1;i<=linha2;i++)
{
    pos_data = i * 160 + coluna1 * 2;
    movedata( VIDEO, pos_data, FP_SEG(p), FP_OFF(p),(total)*2 );

    if ( i==linha1 )
    {
        analize = area[1];
```

```
}

for(j=1;j<(total*2);j+=2)
{
        if ( analize != area[j] )      /* compacta atributo*/
        {
            if ( qtde > 3 )
            {
                for(x=250;x > 0 && qtde > 0; x-- )
                {
                    while ( qtde >= x )
                    {
                        *aux++ = atr;
                        *aux++ = x;
                        *aux++ = analize;
                        total_bytes+=3;
                        qtde-=x;
                    }
                }
            }
            else
            {
                for(;qtde>0;qtde--)
                {
                    *aux++ = analize;
                    total_bytes++;
                }
            }
            analize = area[j];
            qtde=1;
        }
        else
        {
            qtde++;
        }
    }
}
for(x=250;x > 0 && qtde > 0; x-- )
{
    while ( qtde >= x )
    {
        *aux++ = atr;
        *aux++ = x;
        *aux++ = analize;
        total_bytes+=3;
        qtde-=x;
```

```
        }
    }

    tela = (char *) _exmgrab (sizeof(char)*total_bytes);

    if ( tela == NULL )
    {
        _retclen(trabalho,4000);
        return;
    }
    for(i=0;i<total_bytes;i++)
    {
        tela[i] = trabalho[i];
    }
    _retclen(tela,total_bytes);
    _exmback(tela,sizeof(char)*total_bytes);
    _exmback(trabalho,sizeof(char)*4000);
    return;
}


CLIPPER pk_rtscr()
{
    register i=0,
             j=0;

    int      total=0,
             qtde=0,
             linha1=0,
             linha2=0,
             coluna1=0,
             coluna2=0;

    unsigned char   *aux,
                    *tela;

    unsigned char   ch=0,
                    atr=0,
                    analize=0;

    if ( PCOUNT == 0 )
    {
        _retl(FALSE);
        return;
    }
```

```
tela = _parc(1);

p    = area;
aux = tela;

/*
 * CH e' o caractere que sera' utilizado como flag de descompactacao para
 * caractere
 * ATR e' o caractere que sera' utilizado como flag de descompactacao para
 * atributos
 */

ch      = *aux++;
atr     = *aux++;
linha1  = *aux++;
coluna1 = *aux++;
linha2  = *aux++;
coluna2 = *aux++;

total = (coluna2-coluna1)+1;
                .
/*
 * Restaura descompactando
 */

for(i=linha1;i<=linha2;)
{
    for(j=0;j<(total*2) && i <= linha2;)
    {
        if ( *aux != ch )
        {
            area[j++] = *aux;
            area[j++] = 7;
            aux++;
        }
        else
        {
            aux++;
            qtde = *aux++;
            analize = *aux++;

            while( qtde )
            {
                qtde--;
                area[j++]   = analize;
                area[j++] = 7;
```

```
                    if ( j >= (total*2) )
                    {
                        pos_data = i * 160 + coluna1 * 2;
                        movedata( FP_SEG(p), FP_OFF(p), VIDEO,pos_data, (total)*2 );
                        i++;
                        j=0;
                    }
                }
            }
        }
        if ( i < linha2 )
        {
                pos_data = i * 160 + coluna1 * 2;
            movedata( FP_SEG(p), FP_OFF(p), VIDEO, pos_data,(total)*2 );
            i++;
        }
}

qtde=0;

for(i=linha1;i<=linha2;)
{
    pos_data = i * 160 + coluna1 * 2;
    movedata( VIDEO, pos_data, FP_SEG(p), FP_OFF(p),(total)*2 );

    for(j=1;j<(total*2) && i <= linha2;)
    {
        if ( *aux != atr )
        {
            area[j] = *aux++;
            j+=2;
        }
        else
        {
            aux++;
            qtde = *aux++;
            analize = *aux++;

            while(qtde)
            {
                qtde--;
                area[j] = analize;
                j+=2;
                if ( j >= (total*2) )
                {
```

```
                        j=1;
                        pos_data = i * 160 + coluna1 * 2;
                        movedata( FP_SEG(p), FP_OFF(p), VIDEO,pos_data, (total)*2 );
                        i++;
                        pos_data = i * 160 + coluna1 * 2;
                        movedata( VIDEO, pos_data, FP_SEG(p),FP_OFF(p), (total)*2 );
                    }
                }
            }
        }
        if ( i < linha2 )
        {
                pos_data = i * 160 + coluna1 * 2;
            movedata( FP_SEG(p), FP_OFF(p), VIDEO, pos_data,(total)*2 );
            i++;
        }
    }
    _retl(TRUE);
    return;
}
```

Para exemplificar o uso das funções pk_svscr(),pk_rtscr(), segue um exemplo em Clipper Summer'87:

```
*
* Arquivo :   TESTPKSC.prg
* Próposito :Exemplificar a utilização das funçõespk_svscr()
*            e pk_rtscr() escritas em Linguagem C
*
*

    Clear

    For i = 1 to 8
        cor = alltrim(str(i))
        bac = alltrim(str(i+8))
        set color to &cor/&bac
        ? replicate("*",78)
    Next

    ? "***  isto e um teste de compactacao de telas  ***"
    ? "***  isto e um teste de compactacao de telas  ***"
    ? "***  isto e um teste de compactacao de telas  ***"

    For i = 1 to 8
```

```
    cor = alltrim(str(i))
    bac = alltrim(str(i+8))
    set color to &cor/&bac
    ? replicate("*",78)
Next

inkey(0)

a = Pk_SvScr(0,0,24,79)

b = SaveScreen(0,0,24,79)

Clear

@ 1, 1 say   "Tamanho normal de uma tela salva pelo savescreen()"
@ 1,55 say  ltrim(str(len(b)))+" bytes"
@ 3, 1 say   "Tamanho total da tela salva pela PK_SVSCR()"
@ 3,55 say  ltrim(str(len(a)))+" bytes"
@ 5, 1 say   "Economia de "+ltrim(str(100-((len(a)/len(b))*100)))+;
             " ("+ltrim(str(len(b)-len(a)))+" bytes economizados)"
@ 9, 1 say   "Pressione qualquer tecla para restaurar..."

inkey(0)

Pk_RtScr(a)
```

ÍNDICE

A

Aceitação negativa (NAK) 15, 16
Acesso direto à memória (DMA) 90
    período de programação 243
ADAPT.DAT 197
Adaptador de canal da Infotron 121
ADAPTC.BAS 194
Algoritmo de codificação em cadeia Lempel-Ziv 211
Algoritmos de compressão 5
Análise de compressão de pares de caracteres116, 235, 236
Análise de dados 220, 223
Análise de freqüência 221, 222
Aplicações em armazenamento de dados 11
Aplicações em comunicações 8, 9
Arquivo SYSOUT 223, 224
Arquivo utilitário - ARC 264, 271

B

Banco de dados descontrolado 1
Bancos de dados
    produtos de software para compressão de dados 261, 264
    número de usuários e duração do uso 1
    'descontrolados' 1
BASIC 23, 70, 72, 94, 115, 117, 123, 133, 134, 311
    veja também o programa DATANALYSIS
BASICA 70, 72
Buffer duplo 241
BYTEC.BAS 94
BYTED.BAS 99

C

Campos de data 2, 3
Caractere de verificação de bloco (BCC) 17
Caracteres (SI) shift in 40, 70, 71
Caracteres (SO) shift out 40, 70, 71, 82
Caracteres indicadores de compressão 37, 41

Chip (ROM) de memória somente de leitura 10
Ciclo de processamento 244
COBOL 115, 117, 222
Codificação de comprimento de fileira 59, 81, 224, 233, 239
    aplicação 60
    processo básico 62
    processo de decodificação 64
    eficiência 67, 70
    versão da codificação de comprimento de fileira de Honeywell 65
    implementação da codificação de comprimento de fileira 65
    método MNP Classe 5 65-66
    modificações 76
    operação 59
    programação 70
    considerações especiais 62, 63
    usando o código estendido 40
    utilizações 63
Codificação diatômica 112, 131
    implementação por hardware nas comunicações 119
    descompressão 126, 130
    considerações na codificação 130
    operação 112
    freqüência de ocorrência de pares 112, 117
    exemplos de programação 123
Codificação estatística 151, 215
    métodos modernos 199, 215
Codificação Huffman 158, 168, 179, 180, 181, 182, 264
    técnica adaptável 167
    atribuição de bits 164
    considerações de construção de código 164, 167
    comparação com a codificação Shannon-Fano 181, 186
    processo de decodificação 187
    transmissão por facsímile 169
    diferenças de distribuição de freqüência 167, 168
    requisitos à informação 167
    propriedade de decodificação instantânea 161
    códigos do dígito menos significativo 171, 174

modificada 175, 188, 262
códigos dos dígitos mais significativos 174
propriedade de prefixo do código 158, 164
técnica auto-adaptável 168
estruturas em árvore 160, 162, 165
Codificação relativa 135, 141
compressão de dados por facsímile digital 137, 138
compressão das informações de deslocamento 141
técnicas de compressão por facsímile 138, 141
compressão de informações posicionais 141
compressão de dados de telemetria 135, 136
Codificação Shannon-Fano 176, 186, 208
ambiguidade de código 178, 179
comparação com a codificação Huffman 180, 186
processo completo 178
comparação de eficiência 179, 186
processo inicial 177
Código Baudot 31, 32, 59
Código de caractere ASCII 5, 37, 57, 58, 59, 64, 65, 70, 71, 72, 76, 94, 97, 99, 105, 106, 110, 112, 121, 126, 129, 196, 200, 224, 277, 312, 320, 321
Código de fim de linha (EOL) 174
Código decimal codificado em binário (BCD) 32
Código estendido de caracteres decimais codificados em binário para intercâmbio de informações ver EBCDIC
Código Morse 221
Códigos de dados 31, 38
Códigos de vírgula 186, 188
Comando FORMAT 11
Combinação de pares de caracteres 114, 126
Combinação de pares de caracteres de Jewell 114, 126
Combinações de três caracteres
Compactação de meio-byte 81, 93, 233
aplicação 93
considerações do 90
programa de compressão 102
buffer do fluxo de dados 90, 92
decodificação 93

programa de descompressão 99, 101, 106, 109
armazenamento em buffer duplo 91
considerações da eficiência 109
formato de codificação 84, 85
processo de codificação 89, 101
programa de codificação 94, 99
variações de codificação 109
codificação estendida 102
aplicações financeiras 82
execução do programa 109
exemplos de programação 94
armazenamento em buffer único 91
eficiência da técnica
COMPRESS.DAT 319
Compressão auto-adaptável 188 , 190
considerações do campo de contagem 198
eficiência 190
exemplo 193
tabela de compressão fixa 189 , 190
tabela de compressão variável 192
Compressão avançada de dados MNP Classe 7 204, 205
Compressão das informações de deslocamento 141
Compressão de dados
benefícios 4-5
diagrama de bloco 5
técnicas 40, 149
Compressão de dados CCITT V.42bis 207, 215
Compressão de dados de telemetria 135, 136
Compressão de dados MNP Classe 5 199, 201, 59, 65, 66
eficiência 204, 207
flushing 203
Compressão de dados por facsímile digital 137, 141
Compressão de transmissão - estrutura do software 19
Compressão física 3, 4
Compressão lógica 2, 3
Compressão orientada a caracteres 37, 44
CompressoRAD-1 248, 252

Compressores de dados assíncronos 248, 252
Comunicações de dados 5
Conceito de software apoiado pelo próprio usuário 270
Condições de erro 25
Conjunto de caracteres EBCDIC 32, 34, 58, 59, 64, 81, 149, 200
Considerações de temporização 224
Considerações do sistema 217, 220
Considerações para ligação do software 237, 246
Consultative Committee for International Telephone and Telegraph (CCITT) 140, 170
Controle de erro 15, 16, 20
Controle de fluxo 245

D

Datapacker/II 261
Diagrama de árvore de decisão 158, 160
Display CRT 142
Dispositivo de armazenamento de acesso direto (DASD)
    armazenamento 262
Dispositivos de compressão multifuncionais 252, 256
Disquetes 11, 12

E

Economia de imagens de saída 259
Efeito de compressão 21
Eficácia da transmissão 218
Eficiências da compressão de cadeia 207, 211
Elo de comunicações 217
Entropia 186, 207, 236
    computação da entropia 155
    definição 154
    exemplos 154, 158
Estouro de buffer 243
Estouro do campo de contagem 198
Estrutura do software para compressão de transmissão 238
Exatidão de bloco (ACK) 15, 16
Expansão de dados 6
Experiência com o jogo de cara e coroa 155, 156

F

Fator de proporcionalidade 152
Fatores de retardo 13, 14
Figura de mérito 7
Fluxo de dados codificados 7
Fluxo de dados comprimidos 6
Fluxo de dados decodificados 6
Fluxo original de dados 6
Formato de compressão de caracteres repetidos 258, 259
FORTRAN 115, 117, 167, 168, 222
    ver também programa DATANALYSIS
Freqüência de ocorrência 208, 224
Funções de dicionário 215

I

Indicador de mapa de bits 58
INFOPAK DB2 262
INFOPAK IDMS 263
INFOPAK IMS 263
INFOPAK SEQ 263
INFOPAK VSAM 262
International Standard Organization (ISO) 14, 15

L

Ligação de rotinas 246
Linguagem de controle de serviço (JCL) 263
Linha de comunicações 218

M

Mapeamento de Bit 49, 59
    processo de codificação 49, 50
    considerações de hardware 51, 53
    processo de mascaramento 54, 55
    porcentagem de nulos 54
    eficiência da supressão 54, 55
    processo de supressão 51, 53
    limitações da técnica 56, 58
    variações da técnica 56

Memória do processador 239
MERGEC.BAS 311
execução do programa 321, 322
listagem do programa 311, 312
operação do programa 319
MERGED.BAS 311
execução do programa 321, 322
listagem do programa 311, 319
operações do programa 319, 322
MERGED.DAT 311
Método de acesso virtual por telecomunicações (VTAM) 256, 257
Microcom Networking Protocol. Ver MNP
Modelo de canal de retorno 26, 27
Modelo full-duplex 284
Modelo half-duplex completo 17, 19
Modelo Markov 205
Modems 13, 14, 217, 218
Modems de rede chaveada 10
Moderno Código Morse Internacional 221
Multiplexação 253, 256, 120, 123
Multiplexação de banda passante 254
Multiplexador de compressão de dados MC 504 e 508
Ver Streamer
Multiplexador de divisão de tempo 120
Multiplexador estatístico 119
Multiplexador estatístico TL780 da Infotron 119, 123

O

Operação no modo de formulários 142, 149
formato do marcador de dados 148, 149
transmissão de dados 143, 147
método modificado 146, 149
tela do terminal como uma coleção de formulários 146, 148

P

PACKC.BAS 103, 105, 106
PACKC.DAT 95, 109
PACKD.BAS 105, 107

Padrões CCITT 140, 170
Padrões de freqüência 222
PAIRC.BAS 124, 127, 130
PAIRC.DAT 130
PAIRD.BAS 126, 130
Posicionamento da rotina de compressão 237, 242
Positional information compression 140, 141
Probabilidade de ocorrência 152, 153, 169, 184, 186, 190
Processo de codificação 6
Processo de descompressão 6
Produtos de hardware 248
Produtos de software 256, 276
Produtos de software de compressão para monitor de
    teleprocessamento 256, 261
Produtos de software, compressão de arquivo em computadores pessoais 264
Programa DATANALYSIS 115, 223, 224, 234, 277, 309
Programa de compressão 72, 75
Programa de descompressão 76, 80
Programa em BASIC
    rotina principal 287
    descrição operacional 286
    listagem do programa 303
    subprogramas 288, 291
    atribuições às variáveis 292
Programa em FORTRAN
    rotinal principal 277
    descrição operacional 277
    listagem do programa 294, 303
    transferabilidade do programa 283
    subprogramas 278, 282
    atribuições às variáveis 284
Programa SHRINK 311, 321
Protocolo BISYNC 14, 20, 238, 239
Protocolo de transmissão IBM 3780 BISYNC 44, 63
Protocolo HDLC (high-level data link control)
    protocolo14, 15, 40, 249

R

Razão de compressão7, 19, 55, 57, 69, 88, 219
Razão de transferência de dados 218
Razão efetiva de transferência de informações (EITR) 19, 21, 22, 28, 29, 218, 219
Rede multiponto 9
Redução de dados 6
Requisito de transmissão 217
Rótulo de dados modificados (MDT) 260
RUNLENC.BAS 72, 75, 80
RUNLENC.DAT 75, 80
RUNLEND.BAS 77, 80

S

Scotsman III 253, 254
Sinal (CTS) liberado-para-envio 245
Sinal de controle CTS 250
Sistema autônomo de fita-a-fita (SATTS) 65
Sistema de compressão de arquivo PKWARE 271, 274
Software de sistema de terminal remoto (GRTS) 65
Solicitação automática para repetição (ARQ) 15
SQ 264
Streamer 255, 256
Subsistema de fita magnética 217
Substituição de padrões 131, 135
    processo de codificação 132
    tabela de padrões 131
    padrões em linguagens de programação 133, 134
Supressão de imagens de dados redundantes 259
Supressão de nulos 44, 48, 63
    vantagens 48
    limitações 46, 47
    visão geral da técnica 44, 46
    variações da técnica 48, 49

T

Tabela de freqüência 221
Tamanho do buffer 240, 241

Taxa de transferência de dados 218
Taxa de transferência de informação em bits (TRIB) 20, 22, 219
Taxa de transferência de informações (ITR) 9, 18, 20, 21 25, 28
Taxa de transmissão de dados 20
Técnica de inserção e deleção 40, 41, 47
Técnicas de compressão por facsímile 137, 141
Tempo de execução do programa 239
Tempo de retardo 13, 14
Tempo de retardo de processamento 14
Tempo de retardo de propagação 13
Tempo de retardo do circuito 13
Tempo de retardo do modem para se obter RTS/CTS
(solicitação-para-envio/liberado-para-envio)
tempo de retardo do modem 17
Tempo de transmissão de dados 3
Tempo total de descompressão 243
Teoria da informação 152, 157
Terminal batch remoto 217, 219
Transferência de informações 13, 30
Transferência via (DMA) acesso direto à memória 240, 241, 242
Transmissão de facsímile 169
Transmissão fita-a-fita 217
Transmissões CICS 49
Transparência de dados 40
turboMUX 252

U

Unidade de fita magnética 219
USQ 264
Utilização de código estendido 40

V

VTAM-EXPRESS 261

Índice compilado por Geoffrey C. Jones

BIBLIOGRAFIA

Aronson, J. (1977). Data compression-a comparison of methods. National Bureau of Standards, PB-269 296, June.

Dishon, Y. (1977). Data compaction in computer systems. Computer Design, April, 85-90.

Jewell, G.C. (1976). Text compaction for information retrieval systems. IEEE SMC Newsletter, **5**, No.1.

Lempel, A. and Zvi, J. (1977). A universal algorithm for sequential data compression. IEEE Transactions on Information Theory, **IT-23**, No.5.

McCullough, T. (1977). Data compression in high-speed digital facsimile. Telecommunications, July, 40-43.

Moilanen, U. (1978). Information preserving codes compress binary pictorial data. Computer Design, November, 134-136.

Peterson, J.L., Bitner, J.R., and Howard, J.H. (1978). The selection of optimal tab settings. Communications of the ACM, **21**, No. 12, 1004-1007.

Rubin, F. (1976). Experiments in text file compression. Communications of the ACM, **19,** No. 11, 617-623.

Ruth, S., and Kreutzer, P. (1972). Data compression for large business file. Datamation, **18**, No. 9,62-66.

Snyderman, M., and Hunt, B. (1970). The myriad virtues of text compaction. Datamation, **1,** December, 36-40.

Ziv, J., and Lempel, A. (1977). A universal algorithm for sequential data compression. IEEE Transactions on Information Theory, **IT-23**, No. 3.

LEITURA COMPLEMENTAR

Andrews, C.A., Davies, J.M., and Schwarz, E. (1967). Adaptive data compression. Proceedings of the IEE, **55**, No. 3.

Ash, R. (1965). Information Theory, Interscience, New York.

Barton, I.J., Creasey, S.E., Lynch, M.F., and Snell, M.J. (1974). An information-theoretic approach to text searching in direct access systems. Communications of the ACM, **17**, No. 6, 345-350.

Bemer, R.W. (1960). Do it by the numbers-digital shorthand. Communications of the ACM, **3**, No. 810, 530-536.

Bentley, J.L., Skator, D.D., Tarjan, R.E., and Wei, V.K. (1986). A locally adaptive data compression scheme. Communications of the ACM, **29**, No. 4, 320-330.

Blasbalg, H., and Van Blerkom, R. (1972). Message compression. IRE Transactions on Space Eletronics and Telemetry, September, 228-238.

Bookstein, A., and Fouty, G. (1976). A mathematical model for estimating the effectiveness of bigram coding. Information Proceedings and Management, **12**, 111-116.

Bray, J.M., Nelson, V.P., deMaine, P.A.D., and Irwin, J.D. (1985). Data-compression techniques ease storage problems. Computer Desing, October, 102-105.

Clare, A.G., Cook, G.M., and Lynch, M.F. (1972). The identification of variable-length, equifrequency character strings in a natural language data base. Computer Journal, **15**.

Corbin, H. (1981). An introduction to data compression. BYTE, April, 218-250.

Cortesi, D. (1982). An effective text-compression algorithm. BYTE, January, 397-403.

Costlow, T. (1989). Compression doubles QIC capacity. Electronic Engineering Times, January, 4.

Costlow, T. (1989). What's new in data compression. Electronic Engineering Times, January, 53-57

Cullun, R.D. (1972). A method for the removal of redundancy in printed text. NTIS, AD751 407, September.

Davisson, L.D. (1966). Theory of adaptive data compression. In A. V. Balakrishinan (Ed.), Advances in Communications Systems, Academic Press, New York, 173-192.

Davisson, L.D. (1967). An approximate theory of prediction for data compression. IEEE Transactions on Information Theory, **IT-13**, No.2, 274-278.

Davisson, L.D. (1968). Data compression using straight line interpolation. IEEE Transactions on Information Theory, **IT-14**, No.3, 300-304

Davisson, L.D. (1968). The theoretical analysis of data compression systems. Proceedings of the IEEE, **56**, No. 2, 176-186.

Davisson, L.D., and Gray, R.M. (1976). Data Compression, Dowden, Hutchinson and Ross, Dowden.

De Main, P.A.D., Kloss, K., and Marron, B.A. (1967). The SOLID System III alphanumeric compression. Washington: US Government Printing Office, NBS Technical Note 413, August.

Doherty, R. (1989). System puts real-time squeeze on color videos. Electronic Engineering Times, February.

Ehrman, L. (1967). Analysis of some redundancy removal bandwidth compression techniques. Proceedings of the IEEE, **55**, No.3, 278-287.

Ellias, P. (1955). Predictive coding. IRE Transactions, **IT-1**, I6-44.

Fano, R.M. (1949). The transmission of information. Research Laboratory for Electronics, Massachusetts Institute of Technology, Technical Report, No. 65.

Forney, G.D., and Tao, W.Y. (1976). Data compression increases throughout. Data Communications, May/june.

Gilbert, E.N., and Moore, E.F. (1959). Variable-length binary encodings. Bell System Technical Journal, April, 933-967.

Gottlieb, B., Hagereth, S.E., Denot, P.G.H., and Rabinowitz, H.S. (1975). A classification of compression methods and their usefulness for a large data processing center. Proceedings of the National Computer Conference, 453-458.

Hann, B. (1974). A new technique for compression and storage of data. Communications of the ACM, **17**, No.8.

Harker, J. (1982). Byte oriented data compression techniques. Computer Design, October, 95-100.

Heaps, H.S. (1972). Storage analysis of a compression coding for document data bases. Information, **10**, No.1.

Held, G. (1979). Eliminating those blanks and zeros in data transmission. Data Communications, **8**, No.9, 75-77.

Honien, Liu. (1968). A file management system for a large corporate information system data bank. Fall Foint Computer Conference, Vol. 33, Part I, 145-156.

Hu, T.C., and Tucker, A.C. (1971). Optimal computer search trees and variable-length alphabetical codes. SIAM Journal of Applied Mathematics, **21**, No.4 514-532.

Huffman, D.A. (1952). A method for the construction of minimum redundancy codes. Proceedings of the IRE, **40**, 1098-1101.

Karp, R.M. (1961). Minimum-redundancy coding for the discrete noiseless channel. IEEE Transactions on Information Theory, **IT-7**, 27-38.

Kortman, C.M. (1965). Data compression and adaptive telemetry. IEEE Western Electronic Show and Convention (WESCON), Vol.9, Paper 14.4.

Kurmiss, J.M. (1974). An experiment in adaptive encoding. IBM Technical Report Troo. 2524, Poughkeepsie, NY.

Lesk, M.E. (1970). Compressed text storage. Computing Science Technical Report, No.3, Bell Telephone Laboratories.

Ling, H., and Palermo, F.P. (1975). Block-oriented information compression. IBM Journal of Research and Development, March.

Lynch, F.L., Petrie, H.J., and Snell, M.J. (1973). Analysis of the microstructure of titles in the INSPEC data-base. Infomation Storage and Retrieval, **9**, 331-337.

Lynch, M.F. (1973). Compression of bibliographic files using an adaption of run-length coding. Information Storage and Retrieval, **9**, 207-214.

Marron, B.A., and De Maine, P.A.D. (1967). Automatic data compression. Communications of the ACM **10**, No.3, 711-715.

Mayne, A., and James, E.B. (1975). Information compression by factorising common strings. Computer Journal, **18**, No.2, 157-160.

McCarthy, J.P. (1973). Automatic file compression. Proceedings of the International Computer Symposium, North-Holland, Amsterdam, 511-516.

Moilanen, U. (1978) . Information preserving codes compress binary pictorial data. Computer Design, November, 134-136.

Mulford, J.B., and Ridall, R.K. (1971). Data compression techniques for economic processing of large commercial files. ACM Symposium on Information Storage and Retrieval, 207-215.

Mommens, J.H., and Ravir, J. (1974). Coding for data compaction. IBM Research Report, RC 5150, T.J. Watson Research Center, Yorktown Heights, NY.

Nordling, K. (1982). A data compression modem. Telecommunicatons, September, 67-70.

Oliver, B.M. (1952). Efficient coding. Bell System Technical Journal, **21**, No.4, 724-750.

Ott, G. (1967). Compact encoding of stationary Markov sources. IEEE Transactions on Information Theory, **IT-13**, 82-86.

Peterson, J.L. (1979). Text compression. BYTE, December, 106-118.

Pountain, D. (1987). Run-length encoding. BYTE, June, 317-320.

Powell, D. (1989). The hidden benefits of data compression. Networking Management October, 46-54.

Ruth, S.R., and Villers, J.M. (1972). Data Compression and Data Compaction, Government Clearing House Study Number AD 723525.

Schieber, W.D., and Thomas, G. (1971). An algorithm for the compaction of alphanumeric data. Journal of Library Automation, **4**, 198-206.

Schuegraf, E.F., and Heaps, H.S. (1973). Selection of equifrequent word fragments for information retrieval. Infomation Storage and Retrieval, **9**, 697-711.

Schuegraf, E.F., and Heaps, H.S. (1974). A comparison of algorithms for data base compression by the use of fragments as language elements. Information Storage and Retrieval, **10**, 309-319.

Schwartz, E.S., and Kalleck, B. (1964). Generating a canonical prefix encoding. Communications of the ACM, **7**, 166-169.

Seither, M. (1989). Data compression doubles capacity of $\frac{1}{4}$ -inch tape drives. System Integration, May.

Shannon, C.E. (1948). A mathematical theory of communications. Bell System Technical Journal, **27**, 379-423 and 623-656.

Thiel, L.H., and Heaps, H.S. (1972). Program design for retrospective searches on large data bases. Information Storage and Retrieval, **8**, 1-20.

Tropper, R. (1982). Binary-coded text: a text-compression method. BYTE, April, 398-412.

Wagner, Robert A. (1973). Common phrases and minimum-space text storage. Communications of the ACM, **16**, No.3, 148-152.

Wells, M. (1972). File compression using variable length encodings. Computer Journal, April, 308-313.

Impressão e acabamento
(com filmes fornecidos):
EDITORA SANTUÁRIO
Fone (0125) 36-2140
APARECIDA - SP

Você está gastando mais tempo e dinheiro com o armazenamento e a transmissão de dados, do que gostaria? Cerca de 95 por cento de todas as transmissões de dados são compostas de repetições de espaços em branco, numéricos e caracteres alfabéticos, não só se movimentando pelas linhas de comunicações, mas também embutidos em um grande número de bancos de dados. Neste livro o autor mostra como aumentar a eficiência e como cortar os custos da transmissão e do armazenamento de dados, através da utilização de rotinas práticas de compressão de dados.

Mais de 15 técnicas de compressão de dados são descritas detalhadamente e é mostrado ao leitor como adaptar, automaticamente, a compressão aos dados a serem transmitidos. Diversos novos algoritmos de compressão estão presentes e a ênfase é dada à economia no armazenamento em disco para grandes mainframes, mini e microcomputadores.

Este livro é um guia prático, direto e sério para se implementar a compressão de dados. As técnicas apresentadas são muito valiosas, seja para grande ou pequena empresa, seja para ser usada em um mainframe ou em um microcomputador, seja você um usuário final ou um projetista de equipamento.

ÉRICA

LIVROS DE CONTEÚDO, COM QUALIDADE

RUA JARINÚ, 594 - TATUAPÉ - CEP 03306
CX.P. 15617 - TEL.: (011) 294-8686 - SP

ISBN: 085-7194-110-6

ÉRICA

COMPRESSÃO DE DADOS

Held