Esta obra propõe a hipótese da descoberta progressiva, através de programas, das noções fundamentais que estão na base da maior parte dos trabalhos em inteligência artificial: recursividade, representação dos conhecimentos, métodos de percurso das árvores, estratégias de pesquisa, heurísticas, etc.

Para juntar o útil ao agradável, os programas tratam de problemas lúdicos, embora de tipo particularmente didáctico.

Escritos inicialmente em BASIC *standard*, foram adaptados para funcionarem quer no *Spectrum* quer em outros computadores.

Todos os programas desta obra foram verificados e testados pelo Gabinete Verbo de Informática.

BIBLIOTECA VERBO DE INFORMÁTICA

12

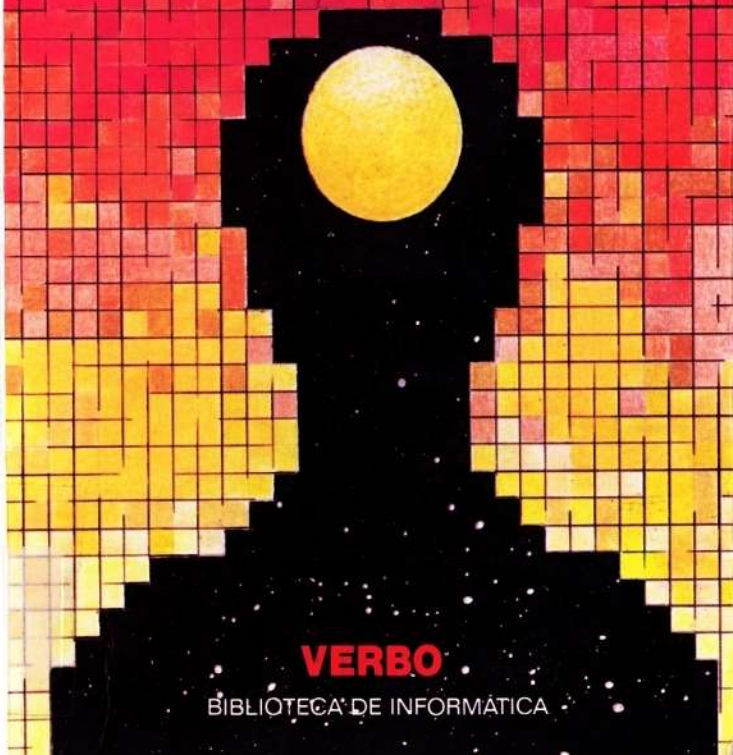
INTELIGÊNCIA ARTIFICIAL MONTEIL SCHOMBERG

Verbo

INTELIGÊNCIA ARTIFICIAL

COM PROGRAMAS PARA O SPECTRUM E OUTROS COMPUTADORES

M.G. MONTEIL R. SCHOMBERG



VERBO
BIBLIOTECA DE INFORMÁTICA

M. G. MONTEIL
R. SCHOMBERG

Inteligência Artificial

Com programas em Basic adaptados ao *Spectrum* e a outros computadores

Verbo

Título do original francês
Programmes d'Inteligence Artificielle en BASIC
Publicado por Editions EYROLLES em Paris

Tradução de
Eng.º Carlos Sebastião e Silva

Editorial Verbo SA — Lisboa/São Paulo
Composto por Fotocompográfica, Lda.
Impresso por Empresa Litográfica do Sul
em Fevereiro de 1987
N.º Ed.: 1743
Depósito Legal n.º 14 204/86

ÍNDICE

Prefácio

A inteligência artificial (IA) é um domínio fascinante, objecto de numerosas pesquisas, e no qual as diferentes classes de aplicação alicerçam grandes esperanças.

Este livro propõe-nos a descoberta progressiva, através de programas codificados em BASIC, das noções fundamentais que estão na base da maior parte dos trabalhos em IA: recursividade, representação dos conhecimentos, métodos de percurso das árvores, estratégias de pesquisa, heurísticas, etc...

Para juntar o útil ao agradável, os programas tratam de problemas de jogos de reflexão a um ou a dois jogadores. O critério de selecção destes programas não foi o seu lado espectacular mas sim a sua qualidade didáctica. Cada programa, codificado em BASIC Sinclair, é analisado em pormenor; embora compactos, toda a dificuldade dos programas reside nos algoritmos utilizados. A fim de permitir um domínio conveniente desses algoritmos, parte dos programas são apresentados acompanhados de uma versão que possui a capacidade de imprimir numa impressora *ZX* um traçado do programa, com algumas semelhanças com uma função TRACE, o qual permite seguir passo a passo a sua execução. A conversão para o traçado sair no *écran* é extremamente simples.

Os programas originais desta obra encontravam-se escritos em BASIC Microsoft, dimensionados para funcionarem num *Apple* 48K, mas foram convertidos pelo Gabinete Verbo de Informática para o BASIC Sinclair, utilizado pelo *ZX Spectrum, ZX Spectrum+* e similares (*Timex 2048,* etc.), visto ser esta gama de microcomputadores a mais difundida no nosso país. A conversão implicou forçosamente que fossem introduzidas algumas alterações aos programas originais, principalmente no que diz respeito ao tratamento de cadeias *(strings)* de caracteres e de variáveis de cadeia *(string variables)*. Contudo, a conversão destes programas para outros dialectos de BASIC

não oferece problemas e, em muitos casos, foram incluídas notas sobre a forma de efectuar essa conversão.

Foram igualmente criadas versões suplementares de alguns programas, versões que, explorando algumas capacidades próprias do *ZX Spectrum* (ao contrário das versões base), se destinam a melhorar a comodidade de utilização dos programas, a compreensão dos algoritmos que lhes estão na base e o grafismo do seu *output*. Estas versões encontram-se geralmente listadas no fim dos capítulos e não são, obviamente, de tão fácil conversão para outros dialectos de BASIC como as versões que lhes servem de base (praticamente idênticas às da obra original). A sua listagem é igualmente mais complexa.

Nesta obra, apesar de muito acessível, certas passagens e certos programas, especialmente no último capítulo, exigem alguns conhecimentos de programação e familiaridade com a terminologia e simbologia próprias. Nesses casos, sempre que possível, foram incluídas breves notas explicativas.

Resta lembrar que os programas devem ser digitados no computador com extremo cuidado, já que o seu bom funcionamento depende disso, em especial no caso do último programa do livro.

Capítulo 1

A inteligência artificial

1.1. A INTELIGÊNCIA ARTIFICIAL

A inteligência artificial, IA em notação, não deve ser confundida, nem assimilada, com a informática.

É verdade que o computador constitui uma ferramenta excelente para trabalhos à base de IA, mas é preciso ter em mente o facto de que o computador não possui qualquer espécie de inteligência.

O que é a inteligência?

Esta pergunta está desde há séculos na base dos trabalhos de inúmeros investigadores (psicólogos, filósofos, teólogos, neurologistas), mas, até hoje, a inteligência das máquinas não foi encarada senão no campo da ficção científica.

No seguimento da evolução das ciências e das técnicas, o homem inventou (graças à sua inteligência) o computador, o qual executa incansavelmente e na perfeição tarefas repetitivas. O computador é muito rápido e fiável na repetitividade, mas as tarefas devem estar perfeitamente definidas, analisadas e formalizadas de antemão (programação). O computador liberta o homem de uma quantidade de tarefas fastidiosas, quer sejam trabalhos de cálculo de todo o género (científico, financeiro, técnico) ou trabalhos de fabrico em série (por adjunção de uma periferia do tipo *process control*).

O avanço da humanidade não se alicerça num comportamento de autómato mas sim sobre a vasta área do conhecimento humano, a qual não pára de crescer exponencialmente. Estima-se que esse conhecimento progrida a um ritmo da ordem dos vários milhões de palavras por minuto, sobre um número incalculável de campos: medicina, física, química, astronomia, tecnologia, etc.

Só peritos extremamente avançados no seu domínio possuem e trabalham esse conhecimento. A tecnologia permite criar dispositivos de memorização que possibilitam o armazenamento de quantidades gigantescas de informação, quase ilimitadas, pela utilização de suportes de memória de massa amovíveis (discos e bandas magnéticas, discos ópticos, etc.)

A dificuldade reside na concepção de programas que manipulem esse conhecimento.

1.2. O CONHECIMENTO

O conhecimento acerca de qualquer coisa pode ser factual ou heurístico.

O **conhecimento factual** consiste numa simples enumeração ou catalogação de informações respeitantes ao objecto a representar.

Assim, hoje em dia, classicamente, existem ficheiros de existências, de clientes, de pessoal, etc.

O computador serve então de super-repertório capaz de localizar essas informações e de as manipular segundo um certo número de tratamentos perfeitamente definidos e formalizados de antemão pelo homem, o qual deve ter previsto todos os casos possíveis.

É a programação clássica, efectuada a partir de algoritmos[1], que permite precisar nos menores detalhes o que o computador deve fazer.

É possível escrever programas do tipo pergunta-resposta que contenham as respostas a todas as perguntas que possam ser postas. Não se pode, neste caso, verdadeiramente falar de inteligência.

[1] Algoritmo: descrição do esquema de realização de um acontecimento, com base num repertório finito de acções elementares. Consultar o início do capítulo 3. *(N. do T.)*

O **conhecimento heurístico** é muito mais difícil de armazenar num computador, pois é composto pela combinação de intuições, de regras de apreciação, de associações que, aplicada aos conhecimentos factuais, permite ao homem demonstrar a sua inteligência.

É, portanto, frequente que saibamos fazer coisas sem sabermos explicar muito bem como as fazemos; avança-se tacteando o caminho, mudando de direcção de vez em quando, não sabendo, a não ser de forma instantânea, se a direcção seguida é a certa.

Por exemplo, o médico ao pronunciar o seu diagnóstico, o perito em prospecção mineira ao procurar uma boa jazida, o jogador de xadrez ao tentar romper a defesa do seu adversário, não seguem um método inteiramente formalizado consistindo num encadeamento de deduções determinadas. Na verdade, o que eles praticam são tentativas baseadas nas suas experiências passadas.

Tratar este tipo de problemas por meio de um computador, quando o procedimento não está inteiramente definido, implica o uso daquilo que se convencionou chamar «inteligência artificial».

A análise de um tal procedimento por tentativas e erros conduz a uma representação por «árvore de opções» ou de «decisões», a qual se encontra bem adaptada e muito divulgada em IA, como o iremos tentar demonstrar em todos os exemplos desta obra.

1.3. COMPARAÇÃO CÉREBRO-COMPUTADOR

O poder de tratamento dos computadores encontra-se todo centralizado num ou em vários processadores (em número limitado) que acedem a um espaço de memória central (RAM) limitado. Só se conseguem grandes áreas de memória por meio de dispositivos relativamente lentos (memórias de massa ou au-

xiliares, como os discos magnéticos, por exemplo), em virtude da inércia do seu funcionamento parcialmente mecânico.

Tal como o cérebro, o computador trata informação proveniente das suas unidades periféricas, verdadeiros órgãos de sentidos. Contudo, a grande diferença reside no facto de o cérebro não concentrar as informações a tratar em alguns poucos centros processadores, mas distribuí-las por milhões de células lógicas que se activam simultaneamente. Deste modo, o paralelismo do cérebro é muito dificilmente imitável pelo funcionamento série do computador.

A título de informação, eis um artigo tirado do *Monde Informatique* de 14 de Janeiro de 1985:

> **«O cérebro humano: uma capacidade ainda não atingida**
> Considerado sob um ponto de vista matemático, o conteúdo da memória do cérebro humano médio equivale a 62,5 milhões de páginas A4 dactilografadas de 2000 caracteres cada, sendo cada carácter representado por 8 *bits* (1 *byte* ou octeto), o que corresponde a uma capacidade total de cerca de um milhão de *megabits*[1] (0,128 *gigabytes*).
>
> No extremo inferior da hierarquia dos suportes de memória, situa-se a folha de papel que, sob o formato de uma página A4 dactilografada, compreende 2000 caracteres de 8 *bits* cada, ou seja, uma capacidade de cerca[2] de 16 *kilobits* (k) (2 *kbytes* — KB). Uma RAM de 16 k (2 KB) permite assim memorizar o conteúdo de uma tal página. Evidentemente, uma RAM de 64 k (8 KB) pode conter 4 páginas e uma RAM de 256 k (32 KB), 16 páginas. Uma RAM de um *Mbit* (128 KB) será suficiente para memorizar cerca de 64 páginas. O cérebro humano, com o seu milhão de *Mbits* está, portanto, longe de ser ultrapassável por uma memória de semicondutores.
>
> As bandas (*tapes*) e discos vídeo, com cerca de 150 000 *Mbits* (18 750 *Mbits*), aproximam-se um pouco dessa capacidade. Segundo um documento da Polygram e da Siemens, estes dois suportes podem gravar cerca de 9,4 milhões de páginas dactilografadas. Seriam pois necessárias oito bandas ou discos para «substituir» o cérebro. Os discos magnéticos utilizados em informática (560 *Mbits* ou 70 *MBytes*) e as bandas magnéticas com a mesma utilização (720 *Mbits* ou 90 *MBytes*) memorizam o conteúdo de 35 000 ou de 45 000 páginas dactilografadas, respectivamente. As memórias de bolhas magnéticas atingem 4 Mbits (512 *KBytes*), ou seja, 250 páginas.
>
> Se a massa cinzenta possui uma capacidade de memorização no limite do imaginável, possui, por contrapartida, um funcionamento muito lento. A parte do cérebro que armazena as informações a curto prazo tem uma velocidade de leitura máxima de 50 *bits* por segundo, enquanto que a que regista os dados a longo prazo se limita a 1 *bit* por segundo. As memórias de semicondutores (5 *Mbits*/s ou 640 *KBytes*/s) são, a título de comparação, verdadeiros bólides!»

Tomemos um exemplo que mostra bem as diferentes organizações de tratamento para efectuar uma mesma função: o reconhecimento visual de uma pessoa, que é quase instantâneo. Segundo teorias recentes, o processo de reconhecimento consiste em projetar a informação pré-tratada representando a imagem a analisar numa gigantesca memória associativa multidimensio-

[1] Nos países anglo-saxónicos, a unidade básica de capacidade de memória de computador é o *byte*, que corresponde a 8 *bits*, enquanto, nos países francófonos se considera o *bit* como unidade básica. Em Portugal é seguida a notação anglo-saxónica mas, por fidelidade ao texto original, mantém-se a notação francesa, sendo os valores acompanhados, **entre parêntesis**, dos seus equivalentes na outra notação. *(N. do T.)*

[2] A unidade **K**, aplicável a qualquer das notações, equivale a 1024 (*bits* ou *bytes*), sendo frequentemente «arredondada» para o milhar. O «mega» (*bit* ou *byte*) será 1024^2 e o «giga» 1024^3. O *Spectrum* possui 48 K (*bytes*) de memória RAM, ou 384 K (*bits*). *(N. do T.)*

nal. O reconhecimento é então quase imediato, por um modelo interno constituído por todas as informações que caracterizam para nós a pessoa em causa.

O mesmo tratamento pedido a um computador ao qual está ligada uma câmara de TV consiste em efectuar um «varrimento» ponto por ponto da imagem e em representá-la por *bits* em memória. Seguidamente, o computador deve tentar, por meio de cálculos numéricos relativamente volumosos, estabelecer o grau de correlação da imagem analisada com cada uma das imagens pré-registadas, a fim de determinar a que mais se assemelha. Mesmo assim, seria necessário que a pessoa a reconhecer se vestisse sempre da mesma maneira!

Não faltam aos detractores da informática os exemplos que mostram que o computador está muito longe de suplantar o cérebro humano.

Depois de uma exposição tão pouco optimista, pode perguntar-se em que é que o computador pode simular um comportamento inteligente.

Se, por um lado, é evidente que o computador não funciona, nem funcionará nunca, como o cérebro, não é menos verdade que a máquina poderá reproduzir uma pequena parte do seu comportamento, graças a programas de concepção lógica muito particular.

Como funciona então o cérebro, segundo as últimas teorias? Segundo o Prof. Francis Crick, não existe no cérebro qualquer supervisor inteligente que decida do tratamento de tal ou tal informação. A memória associativa funciona por conexões ou ligações que se estabelecem entre as células nervosas. Quando do desenvolvimento do cérebro, mesmo antes do nascimento, essas conexões efectuam-se ao acaso. É claro que um grande número dessas conexões não é compatível com um bom funcionamento do sistema. Por isso, devem ser eliminadas e não voltar mais a aparecer.

É pelo sonho que, provavelmente, se efectua a «desaprendizagem» das ligações erradas. A inteligência, permitindo, por associação de ideias, a criação de novas ideias pode, de certa forma, ser imitada por programas informáticos.

Com efeito, pode conceber-se um programa que, para resolver um dado problema, elabore aleatoriamente uma árvore de pesquisa de soluções que avalie, a cada fase, se se está próximo de uma solução ou se, pelo contrário, a pesquisa se afasta dela.

De preferência a ser apenas aleatória, esta pesquisa pode ser muito mais eficaz se se apoiar em regras empíricas.

Estes métodos de resolução por árvore, muito difundidos, apresentam-se muito minuciosamente nos programas deste livro.

Com base nas técnicas de programação por árvores, podem igualmente conceber-se programas que, a partir de uma base de factos e de uma base de conhecimentos constituída por regras aplicáveis a esses factos, elaborem novos factos sobre os quais se aplicam de novo regras e assim por diante. Quando mais nenhuma regra for aplicável, constrói-se uma dedução inteligente a partir dos factos de partida. Um programa deste tipo denomina-se «sistema perito» ou *expert system* e permite obter diagnósticos pertinentes sobre assuntos no âmbito da sua competência.

Veremos posteriormente um exemplo simples de realização.

1.4. LINGUAGENS DE PROGRAMAÇÃO

Linguagens como o Fortran, o Cobol ou o BASIC prestam-se bem à programação de algoritmos e de procedimentos, assim como ao cálculo algébrico: utilizando sub-rotinas, estruturas de ciclos *(loops)*, saltos condicionais e incondicionais, permitem precisar nos mínimos pormenores as tarefas repetitivas que um computador deve executar.

Mas em IA manipulam-se mais símbolos do que grandezas algébricas. O que se torna necessário neste caso é poder representar a associação desses símbolos e o seu encadeamento.

Linguagens como o LISP são boas ferramentas para o tratamento de estruturas de dados encadeados, como listas, símbolos, árvores ou grafos. É possível, inclusivamente, fazer evoluir com uma tal linguagem o ambiente de programação, fazendo com que ele se modifique a si próprio à medida das suas aquisições de conhecimentos.

O LOGO é uma linguagem concebida dentro do mesmo espírito do LISP, permitindo o tratamento de estruturas de dados semelhantes às do LISP, mas com uma sintaxe simplificada. O aparecimento da tartaruga gráfica em 1969, no MIT, foi uma inovação do maior interesse pedagógico. A dita tartaruga é um dispositivo electromecânico destinado a ensinar às crianças a programação em LOGO.

Contudo, uma linguagem criada por volta de 1970 nos laboratórios de IA de Marselha vem trazer novas hipóteses: é o PROLOG, que permite a introdução de variáveis e dá a possibilidade de enunciar um problema sob a forma de um certo número de regras que utilizam essas variáveis e de deixar a máquina trabalhar por si própria para encontrar soluções. A procura de soluções efectua-se grosseiramente pela construção de uma árvore[1] a partir de cada variável. Logicamente, a máquina deveria pesquisar todas as possibilidades para encontrar soluções, o que pode conduzir rapidamente a um número de casos tal que o tempo-máquina necessário à resolução seja excessivo.

Convém, portanto, efectuar apenas a escolha de certos ramos da árvore. O PROLOG permite, aliás, tanto enunciar problemas como, em seguida, resolvê-los.

[1] A definição de «árvore» é dada no capítulo 2, em 2.2.1.

De notar que os Japoneses seleccionaram o PROLOG como linguagem de base para os seus computadores ditos de «quinta geração», os quais são concebidos de forma a optimizar o tempo de execução deste tipo de programa.

Este breve panorama mostra-nos que o BASIC não é a melhor linguagem para escrever programas IA. No entanto, é a única suficientemente difundida para permitir a um máximo de leitores a prática deste tipo de programas.

É preciso notar que, apesar de os programas apresentados neste livro parecerem curtos, o procedimento seguido é relativamente complexo; deste modo, iremos procurar pormenorizá-lo o mais possível ao longo do livro.

O computador em si não é inteligente. São os programas que «vestem» o *hardware* que podem imitar a inteligência humana. Estes programas devem ser escritos (mais precisamente, codificados) em linguagens adaptadas ao tratamento das estruturas de dados e conceitos de IA.

Em resumo, a finalidade desta introdução foi a de situar realisticamente o microcomputador programável em BASIC em relação à dimensão dos problemas colocados pela IA.

É preciso que afastemos de nós a imagem publicitária do computador sobrenatural, místico, espacial, que incita ao sonho.

Todos os programas apresentados e analisados neste livro têm uma finalidade essencialmente didáctica e não devemos pois esperar deles efeitos espectaculares.

As listagens apresentadas destinam-se ao *Spectrum* 48 K e *Spectrum* +, visto ser este micro o mais difundido entre nós. Podem, contudo, ser facilmente adaptadas a outros microcomputadores.

Capítulo 2

A utilização das árvores em IA

2.1. INTRODUÇÃO

Talvez seja útil precisar a razão pela qual consagramos este capítulo aos métodos de pesquisa em árvore.

A noção de árvore é essencial em inteligência artificial. A maioria dos problemas emprega esta estrutura de dados privilegiada e algumas linguagens de programação, como o LISP ou o LOGO, permitem manipulá-la directamente.

Dado um problema, põem-se as seguintes questões:

— que modo de representação adoptar (matriz, quadro, grafo, árvore),

— caso seja uma árvore, como percorrê-la — percurso anterior, interior, posterior, largura primeiro,

— que estratégia adoptar para optimizar a procura (heurística, *back-track,* função de avalização, função de decisão).

Os programas apresentados nesta obra representam um panorama cronológico evolutivo da inteligência artificial e ilustram as noções enumeradas acima.

Iremos tratar, logo de início, da noção de *recursividade* através de dois exemplos concretos: o cálculo de CNP[1] e o jogo das torres de Hanói. Em ambos os casos, o problema é resolvido formalizando-o por meio de uma representação arborescente adequada. A árvore assim construída é percorrida por um *percurso de tipo anterior.*

Para o programa das «Oito Rainhas», precisamos de tomar uma série de decisões. Formaliza-se esta etapa construindo

[1] CNP: combinações de n elementos p a p. *(N. do T.)*

uma árvore de decisões onde cada nó constitui uma decisão possível. Graças a uma função de avaliação de cada nó, eliminam-se as decisões erróneas. A árvore em questão é percorrida seguindo o método anterior. Com efeito, é por vezes necessário voltar a opções tomadas anteriormente e que, subsequentemente, se revelaram inadequadas. A este procedimento dá-se o nome de *back-track.*

O programa apresentado a seguir a este é o referente ao jogo de Nim (também conhecido como jogo de Marienbad), problema *a priori* de natureza combinatória e para o qual existe um algoritmo que permite uma resolução rápida.

Neste caso, a exploração combinatória é elegantemente evitada.

Para o programa «Puzzle-8» e do «Jogo do Galo», desenvolve-se uma linha de pensamento sobre o que se denomina em IA por «espaço de estados». Para estes dois jogos, cada estado é uma configuração do jogo num dado momento. O mencionado espaço de estados é representado por uma árvore cujos nós são os diferentes estados considerados.

Esta árvore é percorrida parcial ou completamente segundo os métodos de percurso adoptados. Alguns são puramente combinatórios, enquanto outros são, pelo contrário, heurísticos e melhoram sensivelmente o rendimento dos programas.

Estes métodos serão apresentados por complexidade crescente ao longo do livro.

Um outro programa, o «Caixeiro Viajante», será, do mesmo modo, apresentado e explicado. Trabalhar-se-á também aqui no contexto de um espaço de estados, sendo cada estado uma lista de cidades a percorrer.

A árvore construída será percorrida segundo métodos heurísticos sofisticados, por um lado para conservar apenas os dados utilizáveis eficazmente (poda ou redução da matriz de dados) e, por outro, para que a árvore possa ser eficazmente per-

corrida (com heurísticas de opção e técnicas como o *back--track*).

Concluiremos com um programa de outro tipo: um «mini--sistema perito». Este tipo de sistema separa dados e modo de utilização desses dados. Como iremos observar, os dados são enunciados sob a forma de regras do tipo *«SE antecedentes ENTÃO consequentes» (IF/THEN)* e armazenados em instruções DATA. São, portanto, facilmente modificáveis.

O sistema procura demonstrar uma das sete hipóteses (armazenadas nas instruções DATA) a partir de factos iniciais pedidos.

O programa utilizado para efectuar a resolução denomina-se *motor de inferências*. Utiliza-se uma estrutura de árvore *(árvore E/OU* ou *AND/OR)* para representar a interligação dos factos e, seguidamente, efectuar o raciocínio.

Em resumo, a estrutura de árvore é usada para:

— formalizar alguns problemas recursivos;

— representar um conjunto de decisões sucessivas a tomar (árvore de decisões);

— representar um espaço de estados;

— estabelecer o *plano de resolução* (árvore E/OU) de um objectivo.

Mas, antes de mais, apresentemos algumas definições que serão utilizadas com frequência ao longo do livro:

2.2. DEFINIÇÕES

2.2.1. Árvore

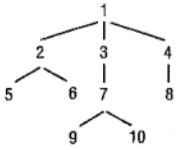
É um conjunto finito de elementos (também chamados *nós* ou *nódulos*) no qual se distingue:

— a *raiz,* na qual têm origem todos os outros nós;

— os outros nós, também chamados *filhos* ou descendência da raiz.

Falar-se-á ainda de *subárvore* para designar um subconjunto de nós saídos do mesmo nó.

Exemplo:



Se um nó possuir vários filhos, distinguiremos o filho mais à esquerda: chamar-lhe-emos portanto o *filho esquerdo* (um filho único será, evidentemente, chamado sempre filho esquerdo).

Se vários nós estiverem ao mesmo nível, saídos de um mesmo nó, chamaremos ao nó em questão *pai* desses nós.

Se um nó tiver vários irmãos, distinguiremos o *irmão direito*, nó imediatamente vizinho do nó em questão.

Um nó terminal da árvore (ou seja, um nó sem filhos) é chamado *folha.*

Notas:

1: Reparemos que a árvore é um *grafo* particular que hierarquiza os dados a partir de uma raiz única.

2: A representação em (*filho esquerdo, irmão direito*) é a habitualmente utilizada. Podem-se citar outras representações como, por exemplo, a em (*filho esquerdo, filho direito*).

2.2.2. Modo de representação

Retomemos a árvore precedente. Adopte-se a representação em (filho esquerdo, irmão direito). Daí resulta que (ver fig.):

1 é a raiz.

2, 3 e 4 são filhos.

2 é filho esquerdo de 1.

Analogamente, 2 tem como filho esquerdo 5.

O nó 3 é o irmão direito de 2.

Analogamente, 4 é o irmão direito de 3.

As folhas da árvore são os nós 5, 6, 9, 10 e 8.

É bastante frequente adoptar-se uma representação da árvore com quatro matrizes-coluna ou quadros a uma dimensão (unidimensionais). Exemplificando com a árvore considerada atrás, temos:

— o quadro que contém os nós que representam os OBJectos a serem examinados no momento (correntemente): OBJ

— o quadro que contém o Filho Esquerdo do nó corrente: FE

— o quadro que contém o Irmão Direito do nó corrente: ID

— o quadro que contém o PAi do nó corrente: PA

A árvore será representada então pela estrutura seguinte:

OBJ	FE	ID	PA
1	2	0	0
2	5	3	1
3	7	4	1
4	8	0	1
5	0	6	2
6	0	0	2
7	9	0	3
8	0	0	4
9	0	10	7
10	0	0	7

Analisemos um pouco estes quadros:

O nó 1 (raiz) — tem por filho esquerdo 2
— não tem qualquer irmão
— nem pai.

O nó 2 tem por filho esquerdo o nó 5, por irmão direito o nó 3 e por pai o nó 1.

etc.

Segundo o uso que dela se pretende fazer, uma árvore pode ser percorrida de maneiras diferentes, como iremos ver no parágrafo seguinte.

2.3. DIFERENTES PERCURSOS DE ÁRVORE

Antes de mais, percorrer uma árvore (ou «descer uma árvore») consiste em enumerar todos os nós da dita árvore e isso, em geral, apenas uma vez, segundo uma ordem bem precisa.

Existem quatro percursos de árvore principais:

— O mais vulgarmente utilizado denomina-se *percurso anterior* (ou de *profundidade primeiro*). É o percurso utilizado na maior parte dos programas desta obra;

— Muito utilizado é também o *percurso em largura primeiro*;

— Um outro percurso também muito utilizado é o *percurso interior*, principalmente nos programas de ordenação (ou *sorting*, também conhecido — erradamente — por «sorteamento»);

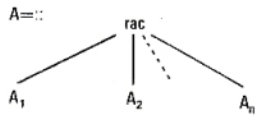
— O último tipo de percurso a analisar: o *percurso posterior*.

Ilustremos estes métodos com base no exemplo precedente:

2.3.1. O percurso anterior (ou de profundidade primeiro ou de pré-ordem)

A sua definição formal é *recursiva* (esta noção será estudada no capítulo seguinte).

Consideremos a árvore A e as subárvores A_1, A_2,..., A_n.

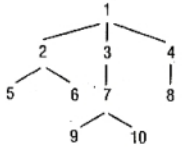


a) «raiz» representa a raiz da árvore
b) raiz

Percurso anterior (A)=raiz, Percurso anterior (A_1),..., Percurso anterior (A_n).

Desce-se, portanto, o mais profundamente possível na árvore partindo da raiz e explorando primeiro o filho mais à esquerda, depois o filho do filho, etc., até à exaustão. Passa-se, em seguida, aos outros ramos (ou descendência de outros filhos da raiz).

Vejamos isto no exemplo definido segundo os três quadros FE, ID, PA:



O percurso anterior desta árvore dá a seguinte ordem de exame:

1, 2, 5, 6, 3, 7, 9, 10, 4, 8.

Dos três quadros FE, ID, PA que definimos atrás, podemos dispensar o quadro PA (PAi) utilizanto uma pilha (denotada P no programa listrado a seguir) onde se memoriza o elemento corrente e o seu nível na árvore antes de ir examinar o seu filho esquerdo.

Quando da execução do programa, devemos simplesmente introduzir o número do nó, o seu filho esquerdo e o seu irmão direito.

Por exemplo: 1, 2, 0

Repare-se que o programa apenas prevê a introdução de dez nós. Para o modificar, basta alterar esse valor na linha 100, valor final da variável de controle de ciclo I e adaptar as dimensões dos quadros da linha 2.

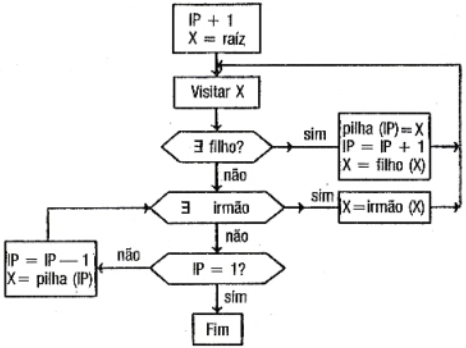
```
   1 REM ************************
         ARVORE
         PERCURSO ANTERIOR
*********************************

   2 DIM P(20): DIM F(40): DIM I
(40)
   5 GO SUB 100
  10 LET IP=1
  20 LET J=IR
  30 PRINT "IMPRIMIR ELEMENTO PE
RCORRIDO ";J
  40 IF F(J)=0 THEN GO TO 50
  45 LET P(IP)=J: LET IP=IP+1: L
ET J=F(J): GO TO 30
  50 IF I(J)=0 THEN GO TO 60
  55 LET J=I(J): GO TO 30
  60 IF IP<=1 THEN STOP
  65 LET IP=IP-1: LET J=P(IP): G
O TO 50
 100 FOR I=1 TO 10: PRINT "NO";
I;"-->";: INPUT J,F(J),I(J): IF
I=1 THEN LET IR=J
 105 PRINT J;",";F(J);",";I(J)
 110 NEXT I: PRINT : RETURN
```

As notações são as seguintes:

P : é a pilha, IP é o indicador de pilha (*stack pointer,* em inglês)
F : é o quadro referente ao filho esquerdo
I : é o quadro referente ao irmão direito
IR: é a raiz

```
NO 1-->1,2,0
NO 2-->2,5,3
NO 3-->3,7,4
NO 4-->4,8,0
NO 5-->5,0,6
NO 6-->6,0,0
NO 7-->7,9,0
NO 8-->8,0,0
NO 9-->9,0,10
NO 10-->10,0,0
```

```
IMPRIMIR ELEMENTO PERCORRIDO 1
IMPRIMIR ELEMENTO PERCORRIDO 2
IMPRIMIR ELEMENTO PERCORRIDO 5
IMPRIMIR ELEMENTO PERCORRIDO 6
IMPRIMIR ELEMENTO PERCORRIDO 3
IMPRIMIR ELEMENTO PERCORRIDO 7
IMPRIMIR ELEMENTO PERCORRIDO 9
IMPRIMIR ELEMENTO PERCORRIDO 10
IMPRIMIR ELEMENTO PERCORRIDO 4
IMPRIMIR ELEMENTO PERCORRIDO 8
```



NOTA: O símbolo ∃ representa um quantificador existencial e deve ler-se «Existe pelo menos um...».

2.3.2. O percurso largura primeiro

Este percurso consiste em explorar iterativamente a árvore, nível por nível, começando pela raiz.

No nosso exemplo de um pouco antes, o percurso deste tipo efectua-se na seguinte ordem:

Nível 0	1
Nível 1	2
	3
	4
Nível 2	5
	6
	7
	8
Nível 3	9
	10

Este algoritmo irá ser utilizado no «Puzzle-8», no capítulo 6.

Repare-se que, contrariamente ao anterior, este percurso é iterativo. As noções de recursão e de iteração serão examinadas no capítulo seguinte.

2.3.3. O percurso interior

Sendo mais difícil de apreender, este percurso não é, contudo, menos utilizado que os precedentes.

A sua definição formal (voltando a utilizar as mesmas convenções empregadas anteriormente) é, também ela, recursiva:

Percurso interior (A)=Percurso interior (A_1), raiz, Percurso interior (A_2),..., Percurso interior (A_n)

Começa-se por explorar as folhas, em seguida a raiz de onde elas saíram, depois a subárvore à direita, etc...

No nosso exemplo, isso dá: 5, 2, 6, 1, 9, 7, 10, 3, 8, 4.

A programação necessita do emprego dos três quadros, FE, ID, PA, embora, como foi anteriormente referido em nota, devamos empregar uma notação algo diferente se utilizarmos o BASIC do *Spectrum* ou similar. Assim virá:

F: quadro referente ao filho esquerdo.
I: quadro referente ao irmão direito.
P: quadro referente ao pai.

```
NO'1-->1,2,0,0
NO'2-->2,5,3,1
NO'3-->3,7,4,1
NO'4-->4,8,0,1
NO'5-->5,0,6,2
NO'6-->6,0,0,2
NO'7-->7,9,0,3
NO'8-->8,0,0,4
NO'9-->9,0,10,7
NO'10-->10,0,0,7

IMPRIMIR ELEMENTO PERCORRIDO 5
IMPRIMIR ELEMENTO PERCORRIDO 2
IMPRIMIR ELEMENTO PERCORRIDO 6
IMPRIMIR ELEMENTO PERCORRIDO 1
IMPRIMIR ELEMENTO PERCORRIDO 9
IMPRIMIR ELEMENTO PERCORRIDO 7
IMPRIMIR ELEMENTO PERCORRIDO 10
IMPRIMIR ELEMENTO PERCORRIDO 3
IMPRIMIR ELEMENTO PERCORRIDO 8
IMPRIMIR ELEMENTO PERCORRIDO 4
```

```
   1 REM ************************
         ARVORE
         PERCURSO INTERIOR
*********************************
   2 DIM S(20): DIM F(40): DIM I
(40): DIM P(40)
   3 REM ************************
         AQUI,A PILHA E'DENOTADA
           S (STACK) E O PAI P
*********************************
   5 GO SUB 100
  10 LET J=IR: LET IP=1
  15 IF F(J)<>0 THEN LET J=F(J):
 GO TO 15
  20 PRINT "IMPRIMIR ELEMENTO PE
RCORRIDO ";J: LET S(IP)=J
  25 LET IP=IP+1
  30 IF P(J)=0 THEN STOP
  35 LET J1=P(J): FOR K=1 TO IP-
1: IF J1=S(K) THEN GO TO 50
  40 NEXT K
```



```
  45 PRINT "IMPRIMIR ELEMENTO PE
RCORRIDO ";J1: LET S(IP)=J1: LET
 IP=IP+1
  50 IF I(J)<>0 THEN LET J=I(J):
 GO TO 15
  55 LET J=J1: GO TO 30
 100 FOR I=1 TO 10: PRINT "NO'";
I;"-->";: INPUT J,F(J),I(J),P(J)
: IF I=1 THEN LET IR=J
 105 PRINT J;",";F(J);",";I(J);"
,";P(J)
 110 NEXT I: PRINT : RETURN
```

2.3.4. **O percurso posterior** (ou pós-ordem)

A sua definição formal é a seguinte:

Percurso posterior (A)=Percurso posterior (A_1), Percurso posterior (A_2),..., Percurso posterior (A_n), raiz.

Tal como para o percurso interior, os três quadros FE, ID e PA são necessários para a programação do algoritmo, mantendo-se a notação referida atrás.

```
NO'1-->1,2,0,0
NO'2-->2,5,3,1
NO'3-->3,7,4,1
NO'4-->4,8,0,1
NO'5-->5,0,6,2
NO'6-->6,0,0,2
NO'7-->7,9,0,3
NO'8-->8,0,0,4
NO'9-->9,0,10,7
NO'10-->10,0,0,7

IMPRIMIR ELEMENTO PERCORRIDO 5
IMPRIMIR ELEMENTO PERCORRIDO 6
IMPRIMIR ELEMENTO PERCORRIDO 2
IMPRIMIR ELEMENTO PERCORRIDO 9
IMPRIMIR ELEMENTO PERCORRIDO 10
IMPRIMIR ELEMENTO PERCORRIDO 7
IMPRIMIR ELEMENTO PERCORRIDO 3
IMPRIMIR ELEMENTO PERCORRIDO 8
IMPRIMIR ELEMENTO PERCORRIDO 4
IMPRIMIR ELEMENTO PERCORRIDO 1
```

```
   1 REM ************************
         ARVORE
         PERCURSO POSTERIOR
*********************************

   2 DIM S(20): DIM F(40): DIM I
(40): DIM P(40)
   3 REM ************************
         AQUI,A PILHA E'DENOTADA
           S (STACK) E O PRI P
*********************************

   5 GO SUB 100
  10 LET J=IR
  15 IF F(J)<>0 THEN LET J=F(J):
 GO TO 15
  20 PRINT "IMPRIMIR ELEMENTO PE
RCORRIDO ";J
  25 IF I(J)<>0 THEN LET J=I(J):
 GO TO 15
  30 IF P(J)<>0 THEN LET J=P(J):
 GO TO 20
  35 STOP
 100 FOR I=1 TO 10: PRINT "NO";
I;"-->";: INPUT J,F(J),I(J),P(J)
: IF I=1 THEN LET IR=J
 105 PRINT J;",";F(J);",";I(J);"
,";P(J)
 110 NEXT I: PRINT : RETURN
```

2.4. NOTAS GENÉRICAS SOBRE OS PROGRAMAS

Nos três programas precedentes, convém modificar o valor máximo da variável de controle de ciclo, I, na linha 100, a fim de tratar as árvores com mais de 10 nós.

Nesse caso, é também necessário adaptar as dimensões dos quadros declarados no início dos programas (instruções DIM).

Em resumo, a finalidade deste capítulo foi, sobretudo, expor a noção de árvore e o seu tratamento. Os programas apresentados nesta obra utilizam largamente esta noção essencial em inteligência artificial.

Capítulo 3

Recursividade

3.1. ALGORITMOS RECURSIVOS OU ITERATIVOS

3.1.1. Generalidades

Um algoritmo pode sempre ser considerado como um conjunto finito de operações que permite avaliar uma determinada função; essa função pode, por exemplo, permitir calcular a melhor jogada a efectuar, a partir das informações presentes e passadas de uma partida em curso.

A informática não se interessa senão pelas funções para as quais existe pelo menos um algoritmo cuja execução pára após um tempo finito, qualquer que sejam os valores das variáveis à entrada dessa função[1].

Com efeito, só os cálculos de duração finita têm sentido.

Mesmo tendo em conta as actuais capacidades de cálculo dos maiores computadores, será necessário limitarmo-nos aos algoritmos que não ultrapassem um determinado tempo de execução.

Os informáticos estabeleceram uma diferença entre os algoritmos *iterativos* e os algoritmos *recursivos*. Tomemos o exemplo clássico da função C_n^p que permite calcular o número

[1] As variáveis que figuram numa função podem ser consideradas como «entrada» dessa função; ao serem-lhes atribuídos valores, o valor que a função então toma, o seu «resultado», será a «saída» da função. *(N. do T.)*

de combinações[2] de n elementos tomados p a p:

$$0 < p < n \quad C_n^p = C_{n-1}^p + C_{n-1}^{p-1} \qquad (1)$$

$$0 < n \quad C_n^0 = C_n^n = 1 \qquad (2)$$

Um algoritmo *iterativo* permite calcular sucessivamente as C_p^n partindo dos valores mais baixos de n e de p. Cada novo resultado é calculado a partir de dois resultados precedentes.

$C_0^0 = 1$			
$C_1^0 = 1$	$C_1^1 = 1$		
$C_2^0 = 1$	$C_2^1 = C_1^1 + C_1^0 = 2$	$C_2^2 = 1$	
$C_3^0 = 1$	$C_3^1 = C_2^1 + C_2^0 = 2 + 1 = 3$	$C_3^2 = C_2^2 + C_2^1 = 1 + 2 = 3$	$C_3^3 = 1$

Constrói-se assim um quadro em forma de triângulo, conhecido pelo nome de triângulo de Pascal ou triângulo de Tartaglia-Pascal, cuja primeira «coluna» e a «hipotenusa» são constituídas apenas por 1's, por aplicação da expressão (2). O resto do quadro preenche-se por aplicação de (1), que põe em acção o algoritmo iterativo.

Um algoritmo *iterativo* permite calcular, a partir de dados iniciais, resultados que irão servir a esse mesmo algoritmo como novos dados iniciais e assim por diante. O algoritmo ir-se-á, portanto, aplicar sucessivamente ao último resultado que ele próprio permitiu calcular, até à obtenção do resultado final.

[2] Dados n elementos quaisquer, chamam-se *combinações* desses n elementos tomados p a p todos os conjuntos que é possível obter com p elementos escolhidos entre os n dados, sem atender a qualquer ordem. *(N. do T.)*

O programa que utilize este tipo de algoritmo deverá, assim, controlar *explicitamente* o número de iterações (por contagem ou por teste de uma variável que indique que o resultado final foi atingido).

Um algoritmo *recursivo* funciona, de certo modo, ao contrário de um algoritmo iterativo. Para determinar C_3^1, por exemplo, este tipo de algoritmo começaria pelo cálculo da última iteração do algoritmo precedente:

$$C_3^1 = C_2^1 + C_2^0$$

Nesta expressão, ele iria calcular *implicitamente* C_2^1 e C_2^0 por aplicação de si próprio, do mesmo modo como havia calculado C_3^1. E assim por diante até à obtenção do resultado.

Contrariamente ao caso do algoritmo iterativo, o número de aplicações sucessivas das igualdades (1) e (2) não é explícita, pois que elas serão efectuadas automaticamente até serem encontradas as relações do tipo (2) que permitam o cálculo numérico efectivo. É este mecanismo implícito que torna os algoritmos recursivos mais difíceis de apreender, quando da leitura de uma listagem, por exemplo.

Sempre que tal seja possível, é vantajoso transformar um algoritmo recursivo num algoritmo iterativo, pois:

— o próprio conceito de recorrência é, para muitas pessoas, difícil de assimilar;

— a eficácia da execução de um algoritmo recursivo é difícil de obter tecnicamente numa linguagem como o BASIC, por exemplo.

3.1.2. Construção de programas recursivos

A execução de um programa correspondente a um algoritmo recursivo necessita da auto-imbricação[1] de, pelo menos, uma parte desse programa. Geralmente, essa auto-imbricação ou auto-sobreposição tem lugar a vários níveis, sendo a única regra a de que a execução da parte auto-imbricada do programa, a um nível i, termine antes de se iniciar a execução ao nível i−1. O número de níveis tem de ser finito.

A existência de uma parte auto-imbricada num programa conduz à gestão de uma pilha de memória (*memory stack*, em inglês) destinada a armazenar temporariamente as informações relativas à execução dessa parte a um determinado nível de auto-imbricação, durante todo o tempo da sua execução ao nível seguinte.

As informações que podem ser «empilhadas» são:

— a localização no nível n da sequência interrompida pela execução do nível n + 1 («endereço de retorno»);

— os valores dos argumentos ao nível n quando esses valores são reutilizados na continuação da sequência interrompida após o regresso do nível n + 1;

— os valores das variáveis «locais», próprias à parte em questão, definidas e calculadas ao nível n, quando esses valores são úteis para terminar a execução do nível n após o regresso do nível n + 1.

Uma sub-rotina ou subprograma assim construído diz-se «reentrante». É uma condição necessária à recursividade. A gestão da dita pilha, comum a todas as partes auto-imbricadas do programa, está a cargo do programador ou, pelo contrário, é realizada pelo compilador (ou pelo interpretador, conforme o

[1] *Nesting*, em inglês. Poderemos também traduzir «imbricação» por «encaixe». O vocábulo francês utilizado no original é *«imbrication»*. *(N. do T.)*

computador) da linguagem, segundo a linguagem de programação escolhida.

A gestão da pilha fica a cargo do programador no caso das linguagens que não permitem que uma sub-rotina contenha uma chamada a si própria, isto é, que interditem uma instrução de salto (um GOSUB, em BASIC) para o seu próprio início. É este o caso de linguagens como o BASIC, o FORTRAN ou o COBOL.

A gestão da pilha é automática quando se utilizam linguagens que permitem a construção de sub-rotinas contendo chamadas a si próprias. Uma sub-rotina destas é chamada recursiva, para lembrar a utilização do esquema de recorrência. O programador não tem de se preocupar com a definição e gestão da pilha necessária ao tratamento das auto-imbricações, se utilizar linguagens como o PASCAL, o LISP, o LOGO, o PL/1, o APL ou o ALGOL.

Convém, contudo, notar que, apesar de estas últimas linguagens mencionadas facilitarem grandemente a escrita (codificação) de programas recursivos, tal escrita continua a ser possível com linguagens como o BASIC, como iremos demonstrar, ficando então a cargo do programador a gestão da recursividade, ou seja, como o dissemos, a pilha das auto-imbricações.

Vamos analisar dois problemas recursivos bem conhecidos: o cálculo de CNP (combinações de N elementos P a P) e as torres de Hanói, a cada um dos quais será consagrado um parágrafo do capítulo.

3.2. PROGRAMA DE CÁLCULO RECURSIVO DE CNP (N,P).

Convirá, desde já, esclarecer algumas coisas no que respeita a notações. De facto, estas mudam frequentemente de país para país e mesmo de autor para autor, dificultando a vida a quem tenha de ler uma obra que empregue uma notação diferente da habitual.

As notações utilizadas para representar as combinações não fogem à regra, e é frequente encontrarmos ${}^{n}C_{p}$, C_{p}^{n}, C_{n}^{p}, $\binom{n}{p}$ ou $C_{(n,p)}$ para representar a mesma coisa[1], e isto não mencionando as variações de índices que tais notações comportam. Assim, iremos utilizar uma notação que se assemelha à última apresentada, mas que tem a vantagem de permitir a representação numa única linha. Teremos pois que:

$CNP(N,P)=C_{(n,p)}$

3.2.1. O algoritmo

O algoritmo recursivo da função CNP(N,P) pode ser esquematizado da seguinte forma:

PROCEDIMENTO CNP(N,P)

se P=0 ou P=N então CNP(N,P)=1 (1)
senão, CNP(N,P)=CNP(N−1,P)+CNP(N−1,P−1) (2)

FIM DE *PROCEDIMENTO*[2]

Notar-se-á em (2) a chamada do próprio procedimento (neste caso a própria função) CNP para os valores (N−1,P) e (N−1,P−1), donde a auto-imbricação que prossegue a um número de níveis implícito. Por outras palavras, no caso de P não tomar os valores indicados em (1), o cálculo de CNP far-se-á *recorrendo* a outros valores de CNP calculados anteriormente, como se explicita em (2). Retomemos agora o princípio de cálculo recursivo com o exemplo de CNP(4,2):

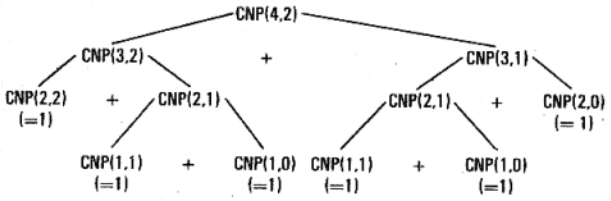
CNP(4,2)=CNP(3,2)+CNP(3,1)
CNP(3,2)=CNP(2,2)+CNP(2,1)
CNP(2,2)=1
CNP(2,1)=CNP(1,1)+CNP(1,0)

[1] Embora $\binom{n}{p}$ seja utilizado, em substituição de C_{p}^{n}, para coeficientes binomiais.

[2] Procedimento (*procedure*, em inglês): entende-se, neste caso, por um conjunto de regras que permitem calcular uma função determinada.

CNP(1,1)=1
CNP(1,0)=1
CNP(3,1)=CNP(2,1)+CNP(2,0)
CNP(2,1)=CNP(1,1)+CNP(1,0)
CNP(2,0)=1
CNP(1,1)=1
CNP(1,0)=1

donde a representação em forma de árvore:



Por aplicação da relação (2) do algoritmo, verifica-se que cada nó (incluindo a raiz) se desdobra até à obtenção de nós terminais, os quais, por aplicação da relação (1), dão 1.

A chamada do procedimento CNP equivalerá assim ao desenvolvimento automático de uma árvore segundo as igualdades precedentes, o que equivale, por sua vez, a adoptar um percurso de tipo anterior. A construção e codificação de um tal procedimento numa linguagem recursiva como o PASCAL, por exemplo, é imediata.

Vamos, então, escrever este programa em BASIC.

3.2.2. Estrutura do programa

A construção e o percurso da árvore efectuam-se graças à utilização de uma pilha que será representada por um quadro unidimensional. Os valores armazenados na pilha permitirão que, quando se atinge um nó terminal, o percurso seja retomado no ramo seguinte, voltando, para isso, a subir até ao nível conveniente.

Na pilha é armazenada uma informação de regresso: o valor 12 para regressar ao label 20, o valor 13 para regressar ao label 26 e o valor corrente da função CNP, resultado intermédio de cálculo[1].

São utilizadas as seguintes variáveis:

Variáveis globais:	N, P	entradas *(input)*
	CNP=C	valor da variável local C à saída do programa.
Variáveis locais:	P(100)	pilha de 100 elementos
	IP	indicador *(pointer)* da pilha
	C	resultado intermédio de cálculo

Todas estas variáveis são inteiras, donde a necessidade, se for utilizado um computador com um BASIC diferente do do *Spectrum*, de se acrescentar o carácter % logo a seguir ao nome da variável (ex.: IP%, P%(100), etc.). Não é este contudo o caso do *Spectrum* ou similares, que utilizam o Sinclair BASIC, e a inexistência de divisões no programa não obriga a precauções neste domínio.

O programa imprime o valor corrente que cada variável vai tomando, para facilitar a análise da sua execução.

3.2.3 Funcionamento do programa

Linha 10: O cálculo de CNP é imediato por aplicação da relação (1) nos dois casos particulares P=0 ou P=N. Está-se, então, num nó terminal (no extremo de um ramo).

Linha 11: No caso geral, efectua-se uma primeira imbricação ou recorrência para calcular CNP(N−1,P).
Há a preparação do regresso ao *label* 20 (neste caso

[1] A escolha dos valores 12 e 13 é justificada em 3.3.2. *(N. do T.)*

linha) graças ao valor 12 armazenado em pilha. O parâmetro N é o decrementado.

Linha 20: Regresso ao 1.º caso (regresso da 1.ª imbricação). O resultado do cálculo de CNP(N−1,P) está na variável C; é colocado em pilha para que seja restituído após a 2.ª imbricação, que irá calcular CNP(N−1,P−1). Para o regresso, o valor 13 é colocado na pilha, depois de ter incrementado o indicador ou *pointer* IP. O regresso efectua-se em 26.

Linha 26: Regresso ao 2.º caso (regresso da 2.ª imbricação). A variável C contém o resultado do cálculo de CNP(N−1,P−1), o qual é acrescentado ao resultado de CNP(N−1,P), que havia sido colocado na pilha. Os três parâmetros são reacertados:
P=P+1; N=N+1; IP=IP−2
(regresso ao nível superior da árvore).

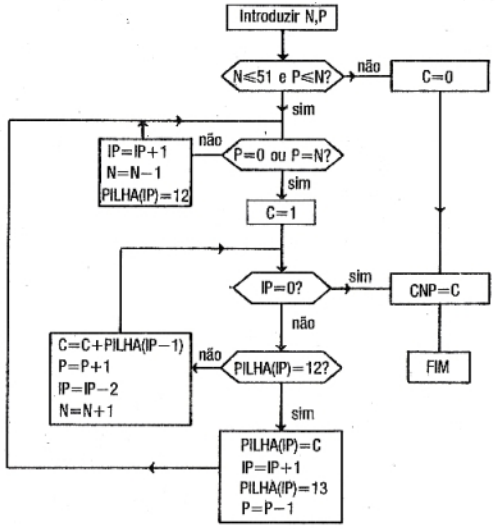
Linha 31: O cálculo é terminado a um nível dado. Se IP=0, então foi obtido o valor da função.
Caso contrário, é preciso que o programa volte à linha 20 ou 26, segundo o valor armazenado então na pilha.

3.2.4. Listagem com edição de variáveis locais

```
 1 REM ************************
        FUNCAO CNP(N,P)
*********************************

 2 DIM P(100)
 3 REM ************************
        O VALOR MAXIMO DE N
        NAO DEVE SUPERAR A
        METADE DA PILHA + 1

 4 REM ************************
        O VALOR DE P NAO
        DEVE SUPERAR N
       ************************

 5 INPUT "N=";N,"P=";P
```

```
   6 IF N>51 OR P>N THEN LET C=0
: GO TO 50
   7 LET IP=0
   8 INPUT "VER VARIAVEIS LOCAIS
 (S/N)? ";V$
   9 IF V$<>"S" AND V$<>"s" THEN
 PRINT "AGUARDE, POR FAVOR...":
PRINT
  10 IF P=0 OR P=N THEN GO TO 30
  11 REM ************************
            CASO GERAL
*********************************

  12 LET IP=IP+1: LET P(IP)=12:
LET N=N-1: IF V$<>"S" AND V$<>"s
" THEN GO TO 15
  13 PRINT : PRINT "IP=";IP;"  P
(IP)=";P(IP);"  N=";N: PRINT
  15 GO TO 10
  16 REM ************************
        REGRESSO DO I CASO
*********************************

  20 LET P(IP)=C: LET IP=IP+1: L
ET P(IP)=13: LET P=P-1: IF V$<>"
S" AND V$<>"s" THEN GO TO 24
  21 PRINT : PRINT "P(IP)=";C;"
 IP=";IP;"  P(IP)=";13;"  P=";P
  22 PRINT
  24 GO TO 10
  25 REM ************************
            II CASO
*********************************

  26 LET C=C+P(IP-1): LET P=P+1:
 LET N=N+1: LET IP=IP-2: IF V$<>
"S" AND V$<>"s" THEN GO TO 26
  27 PRINT : PRINT "C=";C;"  P="
;P;"  N=";N;"  IP=";IP
  28 GO TO 40
  30 LET C=1: GO TO 40
  31 REM ************************
            NIVEL 0 ?
*********************************

  40 IF IP=0 THEN GO TO 50
  41 IF P(IP)=12 THEN GO TO 20
  42 GO TO 25
  50 PRINT : PRINT "CNP=";C
  51 STOP
```


Fluxograma

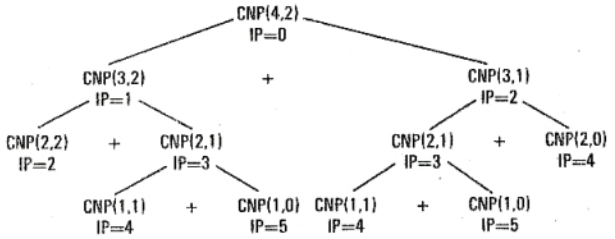


Nota: no programa, o que designamos aqui por PILHA tem a notação P().

3.2.5. Traçado do programa

O vocábulo inglês *trace* designa uma lista de resultados intermediários utilizada no decorrer do acerto de um programa e podemos traduzi-lo por «traçado» do programa. Este traçado inclui geralmente a impressão do conteúdo das variáveis do programa à medida que o processamento se desenrola, de modo a permitir, em conjunto com outras indicações, seguir a execução do dito programa. O programa listado anteriormente permite imprimir esse traçado, se o desejarmos.

Tomemos, por exemplo, os valores N=4 e P=2; neste caso, é construída a árvore seguinte:



Quando da impressão do traçado, são editados todos os valores das variáveis locais do programa:

N,P	: os dados a introduzir
IP	: o valor do indicador de pilha
P(IP)	: o valor da pilha do nível IP
C	: o valor corrente da função CNP

De notar, contudo, que o programa é de execução um tanto lenta, especialmente se os valores de N e P forem elevados, devido ao número rapidamente crescente de ramos a explorar. No caso de valores elevados, o tempo de execução pode bem superar 1 hora!

O programa permite optar pela não impressão dos vários valores das variáveis locais, para aumentar um pouco a velocidade de execução. Se, pelo contrário, desejarmos a sua impressão sem que o computador pare com o inquisitivo «scroll?» mal encha um *écran* (no caso do *Spectrum*), deveremos acrescentar a seguinte linha:

9 POKE 23692,0

Eis o cálculo de CNP(4,2), acompanhado do traçado do programa:


```
N=4  P=2
VER VARIAVEIS LOCAIS (S/N)? S

IP=1  P(IP)=12  N=3

IP=2  P(IP)=12  N=2

P(IP)=1  IP=3  P(IP)=13  P=1

IP=4  P(IP)=12  N=1

P(IP)=1  IP=5  P(IP)=13  P=0

C=2  P=1  N=2  IP=3
C=3  P=2  N=3  IP=1
P(IP)=3  IP=2  P(IP)=13  P=1

IP=3  P(IP)=12  N=2

IP=4  P(IP)=12  N=1

P(IP)=1  IP=5  P(IP)=13  P=0

C=2  P=1  N=2  IP=3
P(IP)=2  IP=4  P(IP)=13  P=0

C=3  P=1  N=3  IP=2
C=6  P=2  N=4  IP=0
CNP=6
```

Se optarmos pela supressão da impressão dos valores intermediários, a execução do programa fornecerá, por exemplo, o

seguinte conjunto de valores de teste (francês *jeux d'essai*):

```
N=52   P=34

CNP=0


N=34   P=35

CNP=0


N=4   P=2
VER VARIAVEIS LOCAIS (S/N)? N

CNP=6


N=60   P=49

CNP=0


N=10   P=8
VER VARIAVEIS LOCAIS (S/N)? N

CNP=45


N=5   P=2
VER VARIAVEIS LOCAIS (S/N)? N

CNP=10
```


Nota: devido à já referida lentidão do programa quando os números a processar são grandes e às limitações da capacidade de cálculo do *Spectrum* (e de outros computadores de 8 *bits*) nas mesmas condições, o valor de N é limitado a 50 (linha 6 do programa). É por este motivo que se obtém CNP(52,34)=0, assim como CNP(60,49)=0, quando, na verdade, estes números são, sensivelmente, $4{,}267\times10^{13}$ e $3{,}427\times10^{11}$.

Nos valores apresentados também aparece CNP(34,35)=0 pois, neste caso, N é menor do que P.

Lista-se, seguidamente, um programa alternativo para o cálculo de CNP(N,P), baseado no conhecido algoritmo que emprega a fórmula

$$\mathrm{CNP}(N,P) = \frac{N!}{P!\,(N-P)!}$$

É dado apenas a título de curiosidade, já que não ilustra nenhum dos métodos de IA descritos e apenas pode servir de comparação em termos de velocidade de execução (muitíssimo mais elevada). Mantêm-se, contudo, as restrições respeitantes à capacidade de cálculo.

```
   1 REM ************************
           COMBINACOES
**********************************

   5 GO TO 100
  10 LET I=1: FOR J=K TO 1 STEP
-1: LET I=I*J: NEXT J
  20 RETURN
 100 CLS : PRINT "***    CALCULO
DE CNP(N,P)    ***"
 105 PRINT '"O VALOR DE N TEM D
E SER INFERIOR          A 34 E P<=
N !"
 110 INPUT "INTRODUZA: N=";N,"P=
";P
 120 IF N>=34 OR P>N THEN GO TO
100
 130 LET K=N: GO SUB 10
```

```
 140 LET FN=I: LET K=P: GO SUB 1
0
 150 LET FP=I: LET K=N-P: GO SUB
 10
 160 LET FPN=I
 170 LET CNP=FN/(FP*FPN)
 180 CLS : PRINT "CNP(";N;",";P;
");"=";CNP
 190 STOP
```

3.3. AS TORRES DE HANÓI

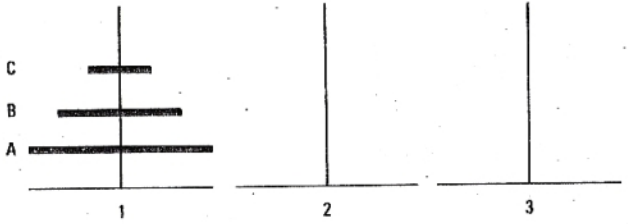
Este problema, muito conhecido em informática, constitui um bom exemplo de programação recursiva e é uma sequência adequada ao cálculo de CNP.

Chama-se torre de Hanói a um conjunto finito de anéis de diâmetros todos diferentes, empilhados por ordem decrescente de diâmetro. Segundo a tradição hindu, o jogo comporta normalmente três estacas que simbolizam torres. Inicialmente, todos os anéis se encontram enfiados na mesma estaca. O jogo consiste em transportá-los a todos para outra estaca, respeitando as seguintes regras:

— não transportar senão um anel de cada vez de uma estaca para outra;

— nunca colocar numa mesma estaca um anel por cima de outro de diâmetro inferior.

A situação inicial é representada a seguir:



Vamos abordar neste capítulo um algoritmo um pouco mais genérico. Limitar-nos-emos ainda a três estacas, mas poderemos partir de qualquer delas e chegar a não importa qual.

O problema apresentado é formalizado por uma representação arborescente adaptada. A árvore assim construída será explorada segundo o percurso anterior (ou de profundidade primeiro).

3.3.1. Resolução do problema

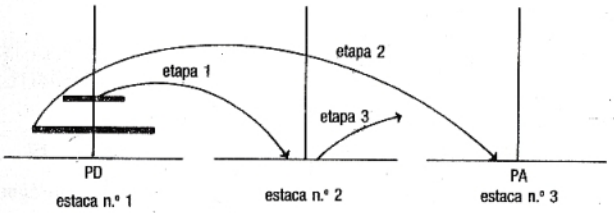
Consideremos, primeiramente, dois discos ou anéis (N=2).

Estabeleçamos as notações:

N	é o número de anéis
EP	é a estaca de partida
EC	é a estaca de chegada
L	é uma estaca intermediária

Consideremos agora a situação inicial seguinte:

A estaca de partida é a estaca n.º 1
A estaca de chegada é a estaca n.º 3



Repare-se que a soma dos números das estacas é igual a 6. Seja pois:

$$\Sigma \text{ n.}^{os} \text{ estacas} = 6$$

O número da estaca intermediária (aqui a estaca n.º 2) é facilmente calculável: n.º estaca intermediária=6−EP−EC.

A resolução do problema é evidente para N=2; ou seja, para dois anéis. Com efeito, basta executar, ordenadamente, os seguintes pontos:

1. passar o primeiro anel da estaca 1 para a estaca 2
2. passar o anel seguinte da estaca 1 para a estaca 3
3. por fim, passar o anel colocado na estaca 2 para a estaca 3.

Tomando as notações adoptadas mais adiante no programa, a sequência das operações é, portanto, a seguinte:

1 para 2

1 para 3

2 para 3

Sabendo que se pode deslocar apenas um anel de cada vez, a resolução do problema pode efectuar-se por recorrência e só é válida para N>1 (o caso N=1 não é tratado).

A heurística é a seguinte:

*N>1 Hanói (EP, EC, N)=Hanói (EP, L, N−1)+1 deslocamento de EP para EC+Hanói (L, EC, N−1)

*N=1 Hanói (EP, EC, N)=1 deslocamento de EP para EC.

Parecendo à primeira vista complexa, a notação empregue aqui clarifica-se com um exemplo. Assim, verifiquemos a heurística para N=2 e, por exemplo, EP=1 e EC=3:

Hanói (1,3,2)=Hanói (1,2,1)+1 deslocamento da estaca 1 para a estaca 3+Hanói (2,3,1)

sendo: Hanói (1,2,1)=1 deslocamento da estaca 1 para a estaca 2

Hanói (2,3,1)=1 deslocamento da estaca 2 para a estaca 3

Com efeito, se deslocarmos os N−1 primeiros anéis para a estaca intermédia (do meio), o último anel, sendo o de maior diâmetro, pode ser transportado para a estaca de chegada, pois ficará necessariamente debaixo de todos os outros anéis na situação final.

Não restará pois, seguidamente, senão deslocar os N−1 outros anéis para a estaca de chegada. É importante notarmos que, no caso de N ser superior a 2, como só se pode deslocar um anel de cada vez, se terá de proceder por etapas elementares, deslocando-se em cada uma delas um anel, segundo a fórmula de recorrência apresentada acima (heurística).

A definição prosposta é, portanto, uma solução recursiva:

Resolver o problema para N anéis consiste em chamar a mesma função para N−1 anéis: a função Hanói chama-se, pois, a ela própria em vários níveis.

3.3.2. Solução do problema

As variáveis utilizadas são todas variáveis inteiras (no caso de o computador utilizado não ser um *Spectrum* ou similar, devem ser seguidas do carácter «%»). Explicitemo-las:

N	:	número de anéis
EP	:	estaca de partida de um deslocamento elementar
EC	:	estaca de chegada de um deslocamento elementar
P	:	pilha utilizada para armazenar os acontecimentos. É um quadro (vector ou *array*) dimensionado a 100.
IP	:	indicador *(pointer)* de pilha
L	:	variável intermediária que, para um deslocamento elementar, toma alternativamente como valor o de EC ou o de EP.

Notemos também que, ao longo de todo o programa, as variáveis EP, EC e L são modificadas.

Far-se-á uso do facto de a soma dos números das três estacas ser igual a 6.

Assim, seja: EP+EC+L=6 (com efeito, 1+2+3=6)

Esta fórmula irá servir, no programa, para calcular o número da estaca corrente.

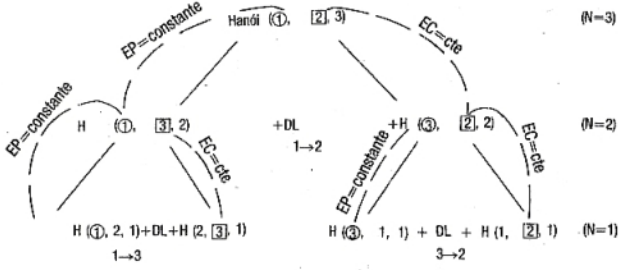
A solução admite uma representação na forma de árvore, examinada da mesma forma que para o cálculo de CNP, ou seja, segundo o percurso anterior. Para N=3, EC=1, EP=2, o procedimento

Hanói (EP,EC,N) onde N é o número de anéis, exprime-se do seguinte modo:

Hanói (1,2,3)=Hanói (1,3,2)+DL 1→2+Hanói (3,2,2)

onde DL 1→2 representa um deslocamento da estaca 1 para a estaca 2.

Procedendo ao desenvolvimento, obtém-se, finalmente, a arborescência:



Os arcos, ou ramos, representam dois contextos de evolução:

— um a EC=constante;
— outro a EP=constante.

Isto é aproveitado na solução que propomos.

Na pilha P é armazenado o contexto da evolução, a saber:

* Se EP=cte, é colocado na pilha o valor 12.
* Se EC=cte, é colocado na pilha o valor 13.

Os valores 12 e 13 são arbitrários, e apenas servem de indicadores: são escolhidos para representar deslocamentos segundo o esquema seguinte:

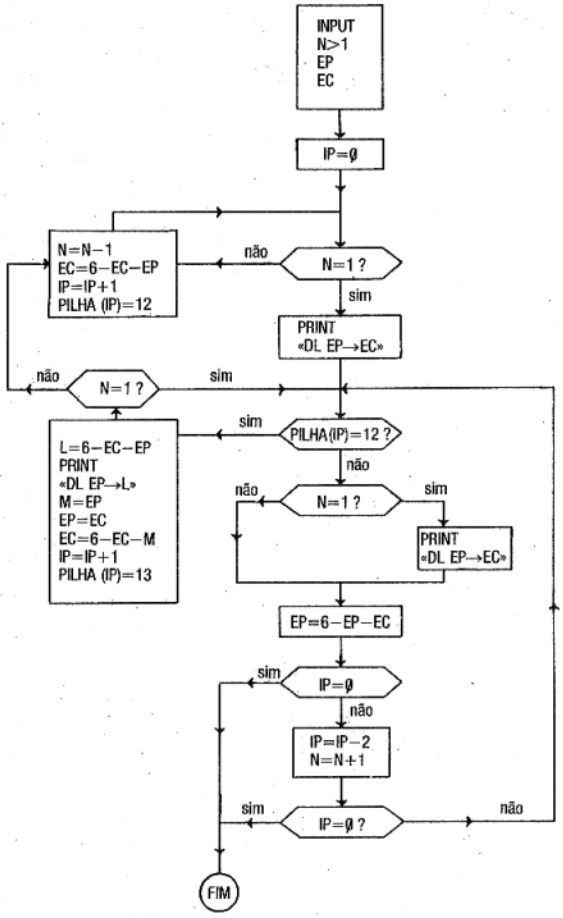
Isto permite saber, em seguida, se se acabou de desenvolver a função Hanói a um nível N dado (N é também o número de anéis).

Eis a listagem em BASIC Sinclair e, seguidamente, o fluxograma do programa:

```
  10 REM
**********************************
  PROGRAMA DAS TORRES DE HANOI
**********************************

  20 DIM P(100)
  30 INPUT "NUMERO DE ANEIS ? ";
N;"NUMERO DA ESTACA DE PARTIDA
? ";EP;"NUMERO DA ESTACA DE CH
EGADA? ";EC
  50 PRINT : PRINT TAB 4;"TRANSP
ORTE DE UM ANEL DE:": PRINT
  60 PRINT : LET IP=0: PRINT
  70 IF N=1 THEN GO TO 100
  80 LET N=N-1: LET EC=6-EC-EP:
LET IP=IP+1: LET P(IP)=12
  90 GO TO 70
 100 PRINT TAB 11;EP;" PARA ";EC
 110 IF P(IP)=12 THEN GO TO 190
 120 IF N=1 THEN GO TO 180
```

```
130 LET EP=6-EP-EC
140 IF IP=0 THEN GO TO 260
150 LET IP=IP-2: LET N=N+1
160 IF IP=0 THEN GO TO 260
170 GO TO 110
180 PRINT TAB 11;EP;" PARA ";EC
: GO TO 130
190 LET L=6-EC-EP
200 PRINT TAB 11;EP;" PARA ";L
210 LET M=EP
220 LET EP=EC
230 LET EC=6-EC-M: LET IP=IP+1:
LET P(IP)=13
240 IF N=1 THEN GO TO 110
250 GO TO 60
260 STOP
```


Fluxograma



Lista-se seguidamente uma variação do programa que permite a impressão das variáveis passo a passo (traçado), tal como são apresentadas no primeiro exemplo, no fim deste capítulo. Se podemos dispor de uma impressora, bastará substituir todas as palavras de comando *(keywords)* PRINT por LPRINT, em qualquer das versões do programa, para ficarmos com um registo permanente *(hard copy)* do *output,* com a desvantagem de um grande consumo de papel no caso de valores de N elevados (para N=10, o número de deslocamentos — e, portanto, a quantidade de papel impresso — já é considerável).

Repare-se ainda que a versão de traçado é obtida da primeira pela simples inclusão das linhas 85, 135, 155, 195 e 235, facilmente localizáveis na listagem.

```
  10 REM
*********************************
  PROGRAMA DAS TORRES DE HANOI
*********************************

  15 REM ************************
         COM IMPRESSAO DOS VA-
         LORES DAS VARIAVEIS
*********************************
  20 DIM P(100)
  30 INPUT "NUMERO DE ANEIS ? ";
N"/";"NUMERO DA ESTACA DE PARTIDA
? ";EP"/";"NUMERO DA ESTACA DE CH
EGADA? ";EC
  50 PRINT : PRINT TAB 4;"TRANSP
ORTE DE UM ANEL DE:": PRINT
  60 PRINT : LET IP=0: PRINT
  70 IF N=1 THEN GO TO 100
  80 LET N=N-1: LET EC=6-EC-EP:
LET IP=IP+1: LET P(IP)=12
  85 PRINT "N=";N;"   EC=";EC;"
 IP=";IP;"  P(IP)=";P(IP)
  90 GO TO 70
 100 PRINT : PRINT TAB 11;EP;" P
ARA ";EC
 110 IF P(IP)=12 THEN GO TO 190
 120 IF N=1 THEN GO TO 180
 130 LET EP=6-EP-EC
 135 PRINT "EP=";EP
 140 IF IP=0 THEN GO TO 260
 150 LET IP=IP-2: LET N=N+1
```



```
 155 PRINT "IP=";IP;"  N=";N
 160 IF IP=0 THEN GO TO 260
 170 GO TO 110
 180 PRINT : PRINT TAB 11;EP;" P
ARA ";EC: GO TO 130
 190 LET L=6-EC-EP
 195 PRINT "L=";L
 200 PRINT : PRINT TAB 11;EP;" P
ARA ";L
 210 LET M=EP
 220 LET EP=EC
 230 LET EC=6-EC-M: LET IP=IP+1:
 LET P(IP)=13
 235 PRINT "EP=";EP;"  EC=";EC;
"  IP=";IP;"  P(IP)=";P(IP)
 240 IF N=1 THEN GO TO 110
 250 GO TO 80
 260 STOP
```

Notações utilizadas no programa:

IP	: indicador *(pointer)* de pilha
N	: número de anéis
EC	: estaca de chegada
EP	: estaca de partida
P( )	: pilha, quadro dimensionado a 100

3.3.3 Traçado do algoritmo

Procede-se da seguinte forma:

Inicialmente, percorre-se o ciclo *(loop)* das linhas (no programa) 70 a 90:

N	=N−1
EC	=6−EC−EP
IP	=IP+1
P(IP)	=12

portanto até que N=1, ficando EP inalterada.

Se N ficar com o valor 1, a execução do programa encontra-se numa «folha» (um nó terminal) de árvore precedente e, no

caso do nosso exemplo com traçado, sobre a folha H(1,2,1). Desloca-se o primeiro anel da estaca 1 para a estaca 2 e imprime-se o deslocamento (linha 100).

Depois, como P(IP)=12 (linha 110), calcula-se o valor de L, neste caso 3 (linha 190 do programa), deslocando-se um disco de 1 para 3 e imprime-se o deslocamento.

A execução posiciona-se, seguidamente, sobre a parte direita da expressão (em baixo, à esquerda, no diagrama de árvore precedente), ou seja, no termo H(2,3,1).

Para isso, executam-se sequencialmente as seguintes instruções (linhas 210 a 230):

```
M      =EP
EP     =EC
EC     =6-EC-M
P(IP)  =13
```

Como, no nosso exemplo, EP=2 e EC=3, N=1 e P(3)=13, imprime-se o deslocamento de 2 para 3 (linha 180).

P(3)=13 e, portanto, EC=cte. Resta modificar EP graças à fórmula (linha 130):

```
EP=6-EP-EC
```

Testa-se seguidamente IP para verificar se o seu valor é zero (linha 140).

Se IP=0, a pilha não contém mais informações e passa-se à etiqueta ou *label* «FIM» (linha 260 STOP).

Caso contrário, sobe-se na árvore ao nível superior, o que equivale a decrementar IP de 2 (linha 150):

```
IP=IP-2
N=N+1
```

No nosso exemplo, está-se então no termo H(1,3,2) para o qual IP=1 e P(1)=12 e, portanto (linha 110), é impresso o deslocamento de 1 para 2 (linha 200) e executada a lista de instruções seguintes (linhas 190 a 230):

```
L      =6-EC-EP
M      =EP
EP     =EC
EC     =6-EC-M
IP     =IP+1
P(IP)  =13
```

Isto feito, a execução posiciona-se no termo H(3,2,2), para o qual IP=2 e P(IP)=13.

A este nível N=2, é novamente percorrido o ciclo de início (linha 80):

```
N      =N-1
EC     =6-EC-EP
IP     =IP+1
P(IP)  =12
```

o que permite a descida na árvore até ao termo H(3,1,1).

Nessa posição, imprime-se o deslocamento de 3 para 1 (linha 100, depois do salto incondicional para a linha 70).

IP=3, P(IP)=12 e N=1

Depois do teste positivo a P(IP)=12 (linha 110), imprime-se o deslocamento de 3 para 2 (linha 200).

Os valores de EC e de EP são alterados: EC=2, EP=1, IP=4 e P(IP)=13. A sequência de execução posiciona-se então sobre o termo H(1,2,1).

Para este termo, N=1 e P(IP)=13, donde resulta (linhas 240, 120 e 180) a impressão do deslocamento de 1 para 2.

Seguidamente, faz-se um teste para IP=0 (linha 140) e, então, são executadas as instruções seguintes (linha 150):

```
IP     =IP-2
N      =N+1
```

Sobe-se, portanto, ao termo H(3,2,2), para o qual

P(IP) =13
N =2
IP =2

Depois, fica EP=1 e IP=Ø (linhas 11Ø a 17Ø, executadas duas vezes). Seguidamente, como IP=Ø, o algoritmo pára (salto da linha 16Ø para 26Ø), com N=3.

Para concluir, damos alguns exemplos de execução:

De início, um primeiro exemplo, obtido com a segunda versão do programa, edita todas as variáveis passo a passo. Os exemplos seguintes, conseguidos com a primeira versão, limitam-se a editar os deslocamentos necessários à resolução do problema.

```
NUMERO DE ANEIS ? 3
NUMERO DA ESTACA DE PARTIDA? 1
NUMERO DA ESTACA DE CHEGADA? 2

     TRANSPORTE DE UM ANEL DE:


N=2    EC=3   IP=1   P(IP)=12
N=1    EC=2   IP=2   P(IP)=12
              1 PARA 2

L=3
              1 PARA 3
EP=2   EC=3   IP=3   P(IP)=13
              2 PARA 3

EP=1
IP=1   N=2
L=2
```



```
              1 PARA 2

EP=3   EC=2   IP=2   P(IP)=13
N=1    EC=1   IP=3   P(IP)=12
              3 PARA 1

L=2
              3 PARA 2
EP=1   EC=2   IP=4   P(IP)=13
              1 PARA 2

EP=3
IP=2   N=2
EP=1
IP=Ø   N=3


NUMERO DE ANEIS ? 2
NUMERO DA ESTACA DE PARTIDA? 1
NUMERO DA ESTACA DE CHEGADA? 3

    TRANSPORTE DE UM ANEL DE:


              1 PARA 2
              1 PARA 3
              2 PARA 3


NUMERO DE ANEIS ? 3
NUMERO DA ESTACA DE PARTIDA? 1
NUMERO DA ESTACA DE CHEGADA? 3

    TRANSPORTE DE UM ANEL DE:
```

```
1 PARA 3
1 PARA 2
3 PARA 2
1 PARA 3
2 PARA 1
2 PARA 3
1 PARA 3
```

```
NUMERO DE ANEIS ? 4
NUMERO DA ESTACA DE PARTIDA? 1
NUMERO DA ESTACA DE CHEGADA? 2

    TRANSPORTE DE UM ANEL DE:

1 PARA 3
1 PARA 2
3 PARA 2
1 PARA 3
2 PARA 1
2 PARA 3
1 PARA 3
1 PARA 2
3 PARA 2
3 PARA 1
2 PARA 1
3 PARA 2
1 PARA 3
1 PARA 2
3 PARA 2
```


Capítulo 4

O problema das Oito Rainhas

Em inteligência artifical, existem numerosíssimos problemas de pesquisa em árvore, sendo esta explorada exaustivamente ou parcialmente. Isto sucede em domínios bem diversos, como a demonstração automática de teoremas, jogos, sistemas peritos, etc...

Vamos ilustrar este capítulo com um caso simples: enumerar todas as possibilidades de colocar várias rainhas sobre um mesmo tabuleiro de xadrez de tal forma a que não fiquem em situação de ameaça mútua, ou seja, que não se encontrem sobre a mesma linha, coluna ou diagonal pois, se tal sucedesse, segundo as regras do xadrez, qualquer delas poderia tomar (ou comer) a ou as que se encontrassem numa dessas três condições.

Para a resolução do problema, deve-se tomar uma sequência de decisões. Isto é formalizado construindo uma árvore de decisões, onde cada nó representa uma decisão possível. Graças a uma função de decisão de cada nó, eliminam-se as escolhas incorrectas. A árvore é percorrida segundo o percurso anterior. Por vezes, verifica-se que decisões tomadas anteriormente se revelam incorrectas; nesses casos, «volta-se atrás» aos pontos onde foram tomadas essas decisões para se tomarem novas decisões, mais coerentes. Este método designa-se, em inglês, por *back-track*.

4.1. APRESENTAÇÃO DO ALGORITMO

Consideremos o tabuleiro de xadrez que acabámos de mencionar como podendo ter qualquer dimensão, e não apenas a usual de 64 quadrados ou casas (8×8). Contudo, neste problema, refere-se geralmente um tabuleiro de 8×8 casas, normal,

portanto, sobre o qual devem ser colocadas 8 rainhas, sendo isto que dá o nome ao problema.

Deve-se, portanto, colocar *a priori* N rainhas sobre um tabuleiro de N×N casas de modo a que nenhum par de rainhas esteja em condição de toma mútua. Uma pequena análise permite resolver com simplicidade o problema: um par encontra-se na situação de toma mútua se ocupar a mesma linha, a mesma coluna ou a mesma paralela a uma das duas diagonais principais do tabuleiro.

1.ª condição: É necessário haver uma rainha por linha.

Tendo sido colocadas P rainhas sobre as P primeiras linhas, será seguidamente necessário colocar a rainha P+1 sobre uma certa coluna da linha seguinte.

Torna-se pois aparente a necessidade de tomar, sucessivamente, uma determinada sequência de decisões, dependendo a decisão P+1, obviamente, das primeiras P decisões já tomadas.

No nosso caso concreto, tomar uma decisão equivale a colocar uma rainha sobre a linha considerada (linha corrente) e sobre uma coluna judiciosamente escolhida, ou seja, sobre a casa ou quadrado situado na intersecção dessa linha com essa coluna.

De uma forma mais geral, isto é característico dos problemas de pesquisa em árvore nos jogos com um só jogador, como é o caso. A sequência de decisões é finita (com efeito, no tabuleiro N×N, há N rainhas a colocar, o que equivale a N decisões a tomar, ou seja, uma por linha).

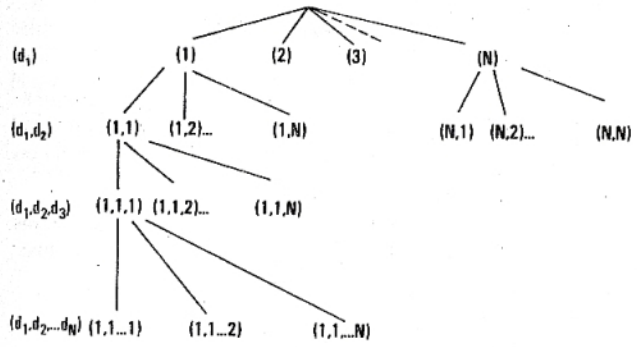
Além disso, cada decisão não pode tomar senão um número finito de valores (no nosso caso, há N valores possíveis para cada decisão, sendo N o número de colunas).

Representemos por D a sequência de decisões:

$D=(d_1, d_2, \ldots d_N)$

onde cada d_i pode tomar N valores diferentes.

A pesquisa ou procura da sequência D de decisões efectua-se graças à construção de uma árvore.



Cada decisão equivale a um nó da árvore.

Além disso, os valores possíveis para a decisão P+1 são função dos valores já tomados pelas P primeiras.

Cada decisão é obtida acrescentando uma variável, nova em relação aos níveis precedentes da árvore.

4.2. RESOLUÇÃO DO PROBLEMA

Quando não existe nenhuma regra que permita, *a priori,* uma orientação de percurso na árvore de decisões, o único método consiste em percorrer exaustivamente toda a árvore. Neste caso, a pesquisa é apenas parcial. O percurso utilizado é o percurso anterior, ou de pré-ordem, que já analisámos no capítulo 2 e ao qual nos podemos reportar para mais pormenores. É este o percurso escolhido pois, naturalmente, as decisões são tomadas e avaliadas à medida que cada nova rainha é colocada: a árvore é construída e seguidamente analisada a partir da raiz...

As variáveis utilizadas são as seguintes:

I e I1 são variáveis globais: I representa a dimensão do tabuleiro e I1 é o contador do número de soluções.
J e K são variáveis intermediárias de cálculo.
C representa a coluna corrente $1 \leq C \leq I$
L representa a linha corrente $1 \leq L \leq I$

Iremos também utilizar uma pilha, P( ), quadro unidimensional, no qual é empilhado o número de coluna escolhido na linha L (L servirá, portanto, também, de indicador de pilha ou *pointer*).

Traduzamos então agora as condições que já enunciámos para que um par de rainhas não se encontre em ameaça mútua:

1.ª condição:
1 só rainha por linha.

2.ª condição:
Tendo sido colocadas P rainhas nas P primeiras linhas, será, em seguida, necessário colocar a rainha P+1 sobre a linha P+1 e sobre uma coluna correctamente escolhida (na intersecção da linha P+1 com a coluna).

a saber:

Condição 1: $C \neq P(J) \quad 1 \leq J \leq L-1$

C : coluna corrente.
L−1 : nível da decisão P−1. É igualmente a linha P−1.

e,

Condição 2: $|(C-P(J))| \neq L-J$

onde os símbolos | indicam que se deve tomar o valor absoluto (módulo) do que se encontra entre eles, ou seja, sem sinal negativo ou positivo, qualquer que seja o valor.

Recordemos também que se armazenam na pilha os números de colunas escolhidas, relativas à linha L.

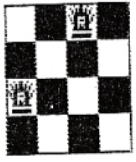
A primeira condição implica que, ao nível P e, portanto, na linha P, se deve colocar a P-ésima rainha sobre uma coluna ainda não escolhida nas linhas precedentes.

A segunda condição traduz o facto de se ter de colocar a rainha em causa (corrente) de tal forma que não se encontra alinhada com uma outra rainha sobre uma mesma linha paralela a uma diagonal principal do tabuleiro.

Exemplo:

para N=4

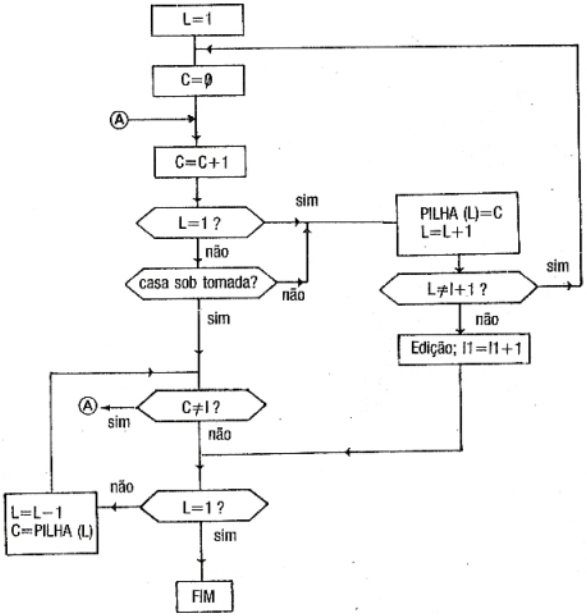
as duas rainhas estão em posição de ameaça mútua segundo a condição 1.



as duas rainhas estão em posição de ameaça mútua segundo a condição 2.

No nosso problema, uma função de decisão irá associar a cada nó gerado na árvore de decisão o valor 1 para um nó possível e $\emptyset$ para um nó a evitar. No caso de impossibilidade, será, por vezes, necessário voltar atrás, a decisões já tomadas, e optar por outra via. Esta técnica é chamada *back-track*.

Analisemos então o fluxograma do programa:



Notações:

I	**: dimensão do tabuleiro**
I1	**: número de soluções**
C	**: coluna corrente**
L	**: linha corrente e, simultaneamente, indicador de pilha**
P()	**: pilha (quadro unidimensional)**

O fluxograma assemelha-se ao apresentado para o percurso anterior (3.1.1.).

Podemos verificar que:

— o teste ∃ filho? do organigrama do percurso anterior corresponde, no caso presente, ao teste L≠I+1;

— o teste ∃ irmão? do organigrama do percurso anterior corresponde ao teste C≠I.

```
   1 REM
*********************************
  PROBLEMA DAS OITO RAINHAS
*********************************

   5 REM    CALCULO DAS POSICOES
DAS RAINHAS NUM TABULEIRO DE XA-
DREZ DE MODO A QUE NENHUM PAR DE
RAINHAS FIQUE EM POSICAO DE TOMA
MUTUA.
  10 DIM P(10): DIM T$(10)
  20 PRINT "*******************
************  PROBLEMA DAS  OIT
O RAINHAS   *******************
************"
  30 INPUT "- DIMENSAO DO TABULE
IRO ? ";I
  40 IF I>10 THEN PRINT AT 5,0;"
NUMERO GRANDE DEMAIS!!": GO TO 3
0
  50 PRINT AT 5,0;"- DIMENSAO DO
 TABULEIRO - ";I;"X";I: PRINT :
PRINT
  60 LET I1=0: LET L=1
  70 LET C=0
  80 LET C=C+1
  90 REM
*********************************
        CASA SOB TOMADA ?
*********************************

 100 LET K=L-1: IF L=1 THEN GO T
O 140
 110 FOR J=1 TO K: IF C=P(J) OR
ABS (C-P(J))=L-J THEN GO TO 200
 120 NEXT J
 130 REM
*********************************
  CASO DA CASA NAO ESTAR SOB TO-
MADA -->POR NA PILHA O NUMERO DA
COLUNA E PASSAR A LINHA SEGUINTE
     -->EDICAO POSSIVEL
*********************************

 140 LET P(L)=C: LET L=L+1: IF L
<>I+1 THEN GO TO 70
 150 LET I1=I1+1
 160 FOR L=1 TO I: FOR K=1 TO I:
```

```
 LET T$(K)="-": NEXT K: LET T$(P
(L))="X"
 170 FOR S=1 TO I: PRINT T$(S);:
 NEXT S: PRINT : NEXT L: PRINT
 160 GO TO 220
 190 REM
*********************************
        CASA SOB TOMADA
-->PASSAGEM AO IRMAO OU AO PAI
*********************************

 200 IF C<>I THEN GO TO 80
 210 REM
*********************************
       REGRESSO AO PAI
*********************************

 220 IF L=1 THEN GO TO 250
 230 LET L=L-1: LET C=P(L): GO T
O 200
 240 REM
*********************************
     IMPRESSAO DAS SOLUCOES
*********************************

 250 PRINT : PRINT "O PROBLEMA T
EM ";I1;" SOLUCOES": PRINT
 260 INPUT "OUTRO JOGO ? (S/N) "
;A$: IF A$="S" OR A$="s" THEN CL
S : GO TO 20
```

As variáveis utilizadas na listagem são as mesmas que foram definidas anteriormente.

4.3. TRAÇADO DO PROGRAMA

Analisemos, passo a passo, como funciona o algoritmo.

Tomemos, por exemplo, o caso em que I=4.

L=1 (Linha 60)
C=0 (Linha 70)

na linha 80 da listagem,

C=1

Seguidamente, a execução salta para a linha 140, pois L=1.

Então, PILHA(1)=1 (P(L)=C)
L=2 (L=L+1)

Considera-se provisoriamente colocada uma rainha na casa ou quadrado (1,1). A mudança de linha para L=2 significa que qualquer outra casa da linha 1 estaria sob ameaça da casa (1,1).

Regresso à linha 70.

C=0
C=1

Na linha 110, testa-se para ver se a casa corrente (L=2, C=1) está em situação de ameaça (em «perigo», digamos) das casas previamente escolhidas.

Ora C=PILHA(1)[1] e, portanto, a casa está sob ameaça (mesma coluna).

Depois, C≠4 (linha 200)

passa-se, portanto, para C=2 (linha 80).

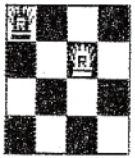
Voltando à linha 110, verifica-se se a casa (2,2) está em situação de ameaça da casa (1,1). É esse o caso, pois encontram-se na mesma diagonal. Depois,

C≠4 (linha 200)

donde fica

C=3 (linha 80)

Aqui, não há problema: PILHA(2)=3 (linha 140)



[1] C=P(J). *(N. do T.)*

L=3
L≠5

Passa-se pois à linha 3 do tabuleiro.

como L<>I+1 ou seja, L≠5, a execução salta para a linha 70:

C=0
C=1

Ora a casa (3,1) está em situação de ameaça da casa (1,1), o que é verificado no ciclo das linhas 110 e 120 e, portanto, salta-se para a linha 200 da listagem:

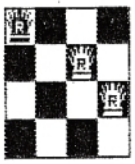
C≠4 donde,
C=2 (linha 80)

Repete-se o procedimento e verifica-se que a casa (3,2) está sob ameaça da casa (2,3), pois, de facto, encontram-se sobre uma mesma paralela a uma das diagonais do tabuleiro.

Assim, C=3. A casa (3,3) está sob ameaça da casa (2,3), pois encontram-se sobre uma mesma coluna.

Assim, C=4. Neste caso também a casa (3,4) está sob ameaça da casa (2,3), pois encontram-se ambas sobre uma mesma paralela a uma das diagonais do tabuleiro.

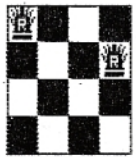
C=I=4 (significa que é a última coluna da linha)
e L=3



Efectua-se um retrocesso (ou *back-track*) no processamento, o «Regresso ao pai» das linhas 220 e 230 da listagem.

L=2 (linha 230)

C=PILHA(2)=3



Como C≠4 (linha 200), a sequência salta para linha 80, ficando C=4.

A casa 2,4 não se encontra sob ameaça da casa (1,1).

Portanto faz-se: PILHA(2)=4 (linha 140)
L=3

Passa-se então à linha seguinte do tabuleiro e faz-se C=1 (linha 80)

Ora verifica-se (linha 110) que a casa (3,1) está sob ameaça da casa (1,1).

Como C≠4 (linha 200), vem C=2 (linha 80).

Aqui não há problema:

PILHA(3)=2 (linha 140)
L=4

como L≠5, passa-se à linha de tabuleiro seguinte (4), primeira casa:

C=1 (linha 80)

Ora verifica-se que a casa (4,1) está sob ameaça da casa (1,1) e da casa (3,2) — ciclo das linhas 110 e 120 —, logo, há um «salto» para a linha 200 (C≠4) e desta para a linha 80, donde:

C=2

Verifica-se que a casa (4,2) está sob ameaça da casa (3,2) e da casa (2,4). Seguindo o mesmo percurso anterior, faz-se

C=3

Repete-se o processo e constata-se que a casa (4,3) está sob ameaça da casa (3,2). Assim, faz-se

C=4

Ora a casa (4,4), a última da última linha, está em situação de ameaça mútua com as casas (1,1) e (2,4), como é facilmente compreensível. Há, assim, um «salto» da linha 11Ø para a linha 2ØØ da listagem, como sucedeu nos casos anteriores. Contudo, como C=4, processa-se um novo *back-track*, um «regresso ao pai», entrando a execução nas linhas 22Ø e 23Ø do programa. Então, faz-se

L=3
C=PILHA(3)=2

Regressa-se, pois, à 2.ª casa da linha anterior (3.ª). Volta-se a testar C na linha 2ØØ) e, como C≠4, volta-se à linha 8Ø, donde

C=3

Isto significa que se trata da casa (3,3), a qual se encontra sob ameaça da casa (2,4). Assim, novo regresso à linha 8Ø e

C=4

Ora a casa (3,4) também se encontra sob ameaça da casa (2,4) e, como C=4, efectua-se novo *back-track*, ficando

L=2 (linha 23Ø)
C=PILHA(L)=4

Como ainda se tem C=4, a linha 2ØØ permite que se efectue novo *back-track*, ficando então

L=1
C=PILHA(L)=1

Como C≠4, há um novo «salto» para a linha 8Ø e faz-se

C=2

Trata-se, pois, da casa (1,2), ainda não verificada anteriormente.

Como L=1, a execução passa (linha 1ØØ) para a linha 14Ø, ficando

PILHA (1)=2
L=2

Novo «salto» para a linha 7Ø, etc...

Paramos aqui a análise do traçado do programa, pois, até ao ponto da impressão da primeira solução para I=4, ainda faltaria um bocado. Mencionemos apenas que as casas «livres de perigo» vão sendo sucessivamente memorizadas na pilha P() e, quando L=I+1 (neste caso L=5) ou seja, quando todas as casas estão verificadas, a linha 14Ø permite que a sequência de execução entre na rotina de impressão. Para a primeira das duas soluções deste caso, as casas livres são as seguintes:

(1,2)	onde	L=1	e	C=P(L)=2
(2,4)	onde	L=2	e	C=P(L)=4
(3,1)	onde	L=3	e	C=P(L)=1
(4,3)	onde	L=4	e	C=P(L)=3

Durante o ciclo de impressão, as casas «livres» são assinaladas por um «X», enquanto as outras por um hífen «-».

Fizemos uma minuciosa explicação do algoritmo de forma a mostrar como se utilizam, na prática, as técnicas de árvores e de percursos de árvore, a pilha e o *back-track,* essencial neste tipo de aplicação. Com efeito, é muito frequente ter de se «voltar atrás» e reconsiderar opções já anteriormente tomadas se se verifica que elas não permitem a aproximação do objectivo pretendido.

Contudo, quem dispuser de uma impressora e desejar uma análise completa do traçado do programa, poderá utilizar a versão do programa preparada para este efeito, listada no fim do capítulo. Convenhamos que o consumo de papel poderá ser considerável, especialmente para valores de I superiores a 4, o que ilustra bem a relativa complexidade da execução do algoritmo.

Listam-se, seguidamente, alguns exemplos de execução do programa. Os exemplos são repetidos para ilustrar a utilização de uma versão do programa, apresentada no fim do capítulo a par da versão com traçagem, que permite um melhor *output* gráfico, já que o programa original descura intencionalmente este aspecto, com vista a uma maior facilidade de compreensão do seu funcionamento. As figuras apresentadas atrás são semelhantes às da versão «melhorada».

Quem quiser utilizar o programa para o caso mais conhecido, ou seja, I=8, o tabuleiro do xadrez clássico, arme-se de paciência (e de uma boa quantidade de papel na impressora, se quiser uma *hard copy*), pois existem 92 soluções possíveis!


```
**********************************
   PROBLEMA DAS  OITO RAINHAS
**********************************

- DIMENSAO DO TABULEIRO ? 1

- DIMENSAO DO TABULEIRO - 1X1
X

O PROBLEMA TEM 1 SOLUCOES

OUTRO JOGO ? (S/N) S


**********************************
   PROBLEMA DAS  OITO RAINHAS
**********************************

- DIMENSAO DO TABULEIRO ? 2

- DIMENSAO DO TABULEIRO - 2X2

O PROBLEMA TEM 0 SOLUCOES

OUTRO JOGO ? (S/N) S


**********************************
   PROBLEMA DAS  OITO RAINHAS
**********************************

- DIMENSAO DO TABULEIRO ? 3

- DIMENSAO DO TABULEIRO - 3X3

O PROBLEMA TEM 0 SOLUCOES
```

```
OUTRO JOGO ? (S/N) S



**********************************
   PROBLEMA DAS  OITO RAINHAS
**********************************

- DIMENSAO DO TABULEIRO ? 4

- DIMENSAO DO TABULEIRO - 4X4

-X--
---X
X---
--X-

--X-
X---
---X
-X--

O PROBLEMA TEM 2 SOLUCOES

OUTRO JOGO ? (S/N) S



**********************************
   PROBLEMA DAS  OITO RAINHAS
**********************************

- DIMENSAO DO TABULEIRO ? 5

- DIMENSAO DO TABULEIRO - 5X5

X----
--X--
----X
-X---
---X-
```



```
X----
---X-
-X---
----X
--X--

-X---
---X-
X----
--X--
----X

-X---
----X
--X--
X----
---X-

--X--
X----
---X-
-X---
----X

--X--
----X
-X---
---X-
X----

---X-
X----
--X--
----X
-X---

---X-
-X---
----X
--X--
X----

----X
-X---
---X-
X----
--X--

----X
--X--
X----
---X-
-X---
```

```
O PROBLEMA TEM 10 SOLUCOES

OUTRO JOGO ? (S/N) N
```

Eis a repetição do exemplo precedente executada pela versão modificada do programa (listada no parágrafo 4.5):

```
**********************************
   PROBLEMA DAS  OITO RAINHAS
**********************************

- DIMENSAO DO TABULEIRO - 1X1

   CALCULO DAS POSICOES DAS RAI-
NHAS NUM TABULEIRO DE XADREZ DE
MODO A QUE NENHUM PAR DE RAINHAS
FIQUE EM POSICAO DE TOMA MUTUA.

      SAIDA POR IMPRESSORA

**********************************

                          SOLUCAO
                            1

** O PROBLEMA TEM 1 SOLUCAO **


**********************************
   PROBLEMA DAS  OITO RAINHAS
**********************************

- DIMENSAO DO TABULEIRO - 2X2
```

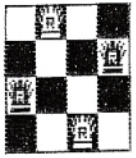


```
   CALCULO DAS POSICOES DAS RAI-
NHAS NUM TABULEIRO DE XADREZ DE
MODO A QUE NENHUM PAR DE RAINHAS
FIQUE EM POSICAO DE TOMA MUTUA.

      SAIDA POR IMPRESSORA

**********************************

** O PROBLEMA TEM 0 SOLUCOES **


**********************************
   PROBLEMA DAS  OITO RAINHAS
**********************************

- DIMENSAO DO TABULEIRO - 3X3

   CALCULO DAS POSICOES DAS RAI-
NHAS NUM TABULEIRO DE XADREZ DE
MODO A QUE NENHUM PAR DE RAINHAS
FIQUE EM POSICAO DE TOMA MUTUA.

      SAIDA POR IMPRESSORA

**********************************

** O PROBLEMA TEM 0 SOLUCOES **


**********************************
   PROBLEMA DAS  OITO RAINHAS
**********************************

- DIMENSAO DO TABULEIRO - 4X4
```

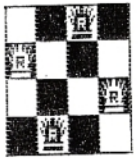
CALCULO DAS POSICOES DAS RAI-
NHAS NUM TABULEIRO DE XADREZ DE
MODO A QUE NENHUM PAR DE RAINHAS
FIQUE EM POSICAO DE TOMA MUTUA.

SAIDA POR IMPRESSORA

****



SOLUCAO
1



SOLUCAO
2

** O PROBLEMA TEM 2 SOLUCOES **

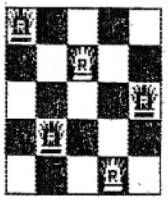
****
PROBLEMA DAS OITO RAINHAS
****

- DIMENSAO DO TABULEIRO - 5X5

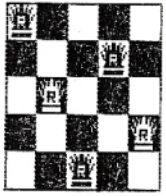
CALCULO DAS POSICOES DAS RAI-
NHAS NUM TABULEIRO DE XADREZ DE
MODO A QUE NENHUM PAR DE RAINHAS
FIQUE EM POSICAO DE TOMA MUTUA.

SAIDA POR IMPRESSORA

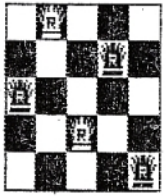
****



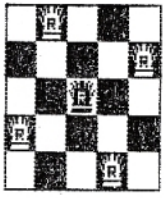
SOLUCAO
1



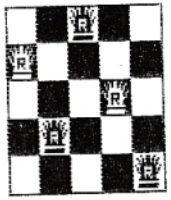
SOLUCAO
2



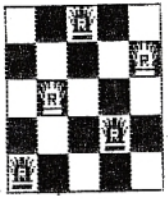
SOLUCAO
3



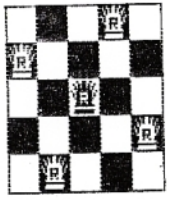
SOLUCAO
4



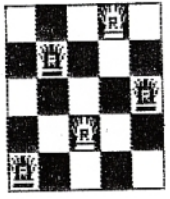
```
SOLUCAO
 5
```



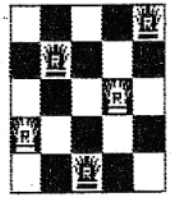
```
SOLUCAO
 6
```



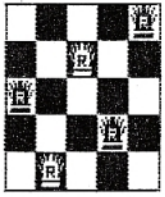
```
SOLUCAO
 7
```



```
SOLUCAO
 8
```

```
SOLUCAO
 9
```



```
SOLUCAO
 10
```

```
** O PROBLEMA TEM 10 SOLUCOES **
```

4.4. CONCLUSÃO

O exemplo que acabámos de elaborar fez-nos construir uma árvore de decisões. Isto é característico nos jogos a um jogador, como este, ou mesmo nos jogos a dois jogadores, como o *Kalah*[1], o xadrez, o jogo do galo, etc.

Precisemos no entanto que introduzimos aqui uma função que analisava a validade de cada nó possível (valor 1) ou não (valor ∅).

[1] *Kalah:* dois jogadores Min e Max (ver «Jogo do Galo», no capítulo 7) têm cada um, à sua frente, seis pilhas contendo um número igual de peças e uma pilha vazia denominada *Kalah*. A finalidade do jogo consiste em um jogador conseguir que o seu *Kalah* venha a conter mais de metade das peças do jogo. Se isso acontecer, a partida termina. O jogo também acaba se um jogador tiver todas as suas pilhas vazias quando for a sua vez de jogar. *(N. do T.)*

Nos jogos, empregam-se funções que permitem igualmente a análise de cada nó mas, de um modo geral, esta função de análise é mais sofisticada. No nosso caso, a função permite apenas determinar se uma subárvore (subconjunto de nós saídos do mesmo nó) é ou não interdita (se não permite ou se permite a aproximação ao objectivo pretendido), enquanto que, muito frequentemente, há necessidade de saber se uma subárvore é «melhor» do que uma outra, ou seja, se ela conduz a um valor da função de decisão melhor do que aquele já encontrado.

Voltaremos a este ponto um pouco mais adiante, mais concretamente quando analisarmos o jogo do *puzzle* 8 e o jogo do galo.

4.5 PROGRAMAS SUPLEMENTARES

Tendo em vista as particularidades de funcionamento do *ZX Spectrum* e a possibilidade de as podermos aproveitar (inclusive algumas *a priori* desfavoráveis) para uma melhor análise e compreensão dos conceitos apresentados, assim como para a obtenção de um *output* mais atraente, a par de uma maior comodidade de operação, decidimos incluir neste capítulo duas versões suplementares do programa original. Assim, a primeira destina-se a dar, via impressora, um traçado do programa à medida que este vai sendo executado (se não possuir impressora, bastará substituir os LPRINT por PRINT para obter o traçado no *écran*); a segunda tem como finalidade a apresentação dos resultados de uma forma graficamente mais atraente, além de permitir a opção de saída por *écran* ou impressora, e de ser, na generalidade, bastante mais *user-friendly* do que o original, intencionalmente simplificado na sua estrutura.

Se analisarmos as duas versões apresentadas seguidamente, podemos verificar que são facilmente obtidas do programa original acrescentando e/ou alterando a este algumas linhas (na segunda versão o caso não é tão simples, mas toda a estrutura do programa original ainda lá se encontra praticamente inalterada).


```
   1 REM
*********************************
    PROBLEMA DAS OITO RAINHAS
*********************************
   2 REM
        VERSAO PARA TRACADO
           VIA IMPRESSORA
*********************************

   5 REM    CALCULO DAS POSICOES
DAS RAINHAS NUM TABULEIRO DE XA-
DREZ DE MODO A QUE NENHUM PAR DE
RAINHAS FIQUE EM POSICAO DE TOMA
MUTUA.
  10 DIM P(10): DIM T$(10)
  20 PRINT "********************
************   PROBLEMA DAS OIT
O RAINHAS   ********************
************"
  30 INPUT "- DIMENSAO DO TABULE
IRO ? ";I
  40 IF I>10 THEN PRINT AT 5,0;"
NUMERO GRANDE DEMAIS!!": GO TO 3
0
  50 PRINT AT 5,0;"- DIMENSAO DO
 TABULEIRO - ";I;"X";I;"";
  55 LPRINT "I=";I
  60 LET I1=0: LET L=1
  65 LPRINT "";"LINHA 60    I1=";I
1;" L=";L;
  70 LET C=0
  75 LPRINT "";"LINHA 70    C=";C;
  80 LET C=C+1
  85 LPRINT "";"LINHA 80    C=";C
,"L=";L;
  90 REM
*********************************
        CASA SOB TOMADA ?
*********************************

 100 LET K=L-1: LPRINT "";"LINHA
 100  K=";K,"L=";L;: IF L=1 THEN
 GO TO 140
 105 LPRINT "";"LINHA 110";
 110 FOR J=1 TO K: IF C=P(J) OR
ABS (C-P(J))=L-J THEN GO TO 200
 120 NEXT J
 130 REM
*********************************
  CASO DA CASA NAO ESTAR SOB TO-
MADA -->POR NA PILHA O NUMERO DA
COLUNA E PASSAR A LINHA SEGUINTE
     -->EDICAO POSSIVEL
*********************************
```

```
 140 LPRINT "";"LINHA 140  L=";L;
: LET P(L)=C: LET L=L+1: LPRINT
;"  P(L)=";C;"  L=";L;: IF L<>I+
1 THEN GO TO 70
 150 LET I1=I1+1
 155 LPRINT "";"LINHA 150  I1=";
I1;
 160 LPRINT "";"LINHA 160  PREPA
RA IMPRESSAO **": FOR L=1 TO I:
FOR K=1 TO I: LET T$(K)="-": NEX
T K: LPRINT ";"P(L)=";P(L): LET
T$(P(L))="X"
 170 FOR S=1 TO I: PRINT T$(S);:
 NEXT S: PRINT : NEXT L: PRINT
 180 GO TO 220
 190 REM
**********************************
          CASA SOB TOMADA
-->PASSAGEM AO IRMAO OU AO PAI
**********************************

 200 LPRINT "";"LINHA 200  C=";C
;"  J=";J;: IF C<>I THEN GO TO 8
0
 210 REM
**********************************
         REGRESSO AO PAI
**********************************

 220 LPRINT "";"LINHA 220  L=";L
;: IF L=1 THEN GO TO 250
 230 LET L=L-1: LET C=P(L): LPRI
NT "";"LINHA 230  L=";L;"  C=PIL
HA(L)=";C;: GO TO 200
 240 REM
**********************************
     IMPRESSAO DAS SOLUCOES
**********************************

 250 PRINT : PRINT "O PROBLEMA T
EM ";I1;" SOLUCOES": PRINT
 255 LPRINT "";"LINHA 250  SOLUCO
ES: I1=";I1
 260 INPUT "OUTRO JOGO ? (S/N) "
;A$: IF A$="S" OR A$="s" THEN CL
S : GO TO 20
```



```
   1 REM
**********************************
    PROBLEMA DAS OITO RAINHAS
**********************************

   2 REM
        VERSAO GRAFICA

© Carlos Peres Sebastiao e Silva
  GABINETE VERBO DE INFORMATICA
  1986

**********************************

   5 LET H$="   CALCULO DAS POSI
COES DAS RAI-NHAS NUM TABULEIRO
DE XADREZ DE MODO A QUE NENHUM P
AR DE RAINHASFIQUE EM POSICAO DE
 TOMA MUTUA."
   6 LET I$="******************
*************"
  10 DIM P(10): DIM T$(10)
  15 GO SUB 9000
  20 PRINT "******************
***********   PROBLEMA DAS  OIT
O RAINHAS   ******************
*************"
  30 GO SUB 2000: INPUT "- DIMEN
SAO DO TABULEIRO ? ";I
  40 IF I>10 THEN PRINT AT 5,0;"
NUMERO GRANDE DEMAIS!!": GO TO 3
0
  50 PRINT AT 5,0;"- DIMENSAO DO
 TABULEIRO - ";I;"X";I;"";H$;"""
'TAB 6;X$;"""";I$: PAUSE 300: IF
 Z$="S" OR Z$="s" THEN COPY
  55 CLS : PRINT AT 10,6; FLASH
1;"AGUARDE SE FAZ FAVOR"
  60 LET I1=0: LET L=1
  70 LET C=0
  80 LET C=C+1
  90 REM
**********************************
      CASA SOB TOMADA ?
**********************************

 100 LET K=L-1: IF L=1 THEN GO T
O 140
 110 FOR J=1 TO K: IF C=P(J) OR
ABS (C-P(J))=L-J THEN GO TO 200
 120 NEXT J
 130 REM
**********************************
```

```
  CASO DA CASA NAO ESTAR SOB TO-
MADA -->POR NA PILHA O NUMERO DA
COLUNA E PASSAR A LINHA SEGUINTE
     -->EDICAO POSSIVEL
*********************************

 140 LET P(L)=C: LET L=L+1: IF L
<>I+1 THEN GO TO 70
 150 LET I1=I1+1
 155 CLS
 160 FOR L=1 TO I: FOR K=1 TO I:
 LET T$(K)="-": NEXT K: LET T$(P
(L))="X"
 170 GO SUB 1000: NEXT L: GO SUB
 1300: GO SUB 1200
 180 GO TO 220
 190 REM
*********************************
          CASA SOB TOMADA
-->PASSAGEM AO IRMAO OU AO PAI
*********************************

 200 IF C<>I THEN GO TO 80
 210 REM
*********************************
          REGRESSO AO PAI
*********************************

 220 IF L=1 THEN GO TO 250
 230 LET L=L-1: LET C=P(L): GO T
O 200
 240 REM
*********************************
     IMPRESSAO DAS SOLUCOES
*********************************

 250 PRINT AT 10,0;"** O PROBLEM
A TEM ";I1;" SOLUC";"OES **" AND
 (I1<>1);"AO **" AND (I1=1): PRI
NT
 255 IF Z$="S" OR Z$="s" THEN LP
RINT "** O PROBLEMA TEM ";I1;" S
OLUC";"OES **" AND (I1<>1);"AO *
*" AND (I1=1);,,,,,,,
 260 INPUT "OUTRO JOGO ? (S/N) "
;A$: IF A$="S" OR A$="s" THEN CL
S : GO TO 20
 270 STOP
1000 REM
*********************************
       ROTINA DE IMPRESSAO
*********************************

1005 LET LP=1: LET CASA=1: IF L/
```



```
2=INT (L/2) THEN LET LP=2: LET C
ASA=0
1010 FOR F=1 TO 2: FOR S=1 TO I
1020 IF LP=2 THEN GO TO 1060
1030 IF T$(S)="-" AND S/2=INT (S
/2) THEN PRINT "▓▓";: LET CASA=1
: GO TO 1100
1040 IF T$(S)="-" THEN PRINT "
";: LET CASA=0: GO TO 1100
1050 IF T$(S)="X" THEN GO SUB 15
00: GO TO 1100
1060 IF T$(S)="-" AND S/2=INT (S
/2) THEN PRINT "  ";: LET CASA=0
: GO TO 1100
1070 IF T$(S)="-" THEN PRINT "▓▓
";: LET CASA=1: GO TO 1100
1080 IF T$(S)="X" THEN GO SUB 15
00
1100 NEXT S: PRINT
1110 IF LP=1 THEN LET CASA=1
1120 IF LP=2 THEN LET CASA=0
1130 NEXT F
1140 RETURN
1200 PRINT AT 0,22;"SOLUCAO";AT
2,25;I1
1210 IF Z$="S" OR Z$="s" THEN CO
PY : CLS : GO TO 1230
1220 PRINT #0;"*PRIMA UMA TECLA
PARA CONTINUAR*": PAUSE 0: CLS
1230 RETURN
1300 REM
*********************************
          ESQUADRIA
*********************************

1310 PLOT 0,175: DRAW I*16,0: DR
AW 0,-I*15: DRAW -I*16,0: DRAW 0
,I*16
1320 RETURN
1500 REM
*********************************
      IMPRESSAO DA RAINHA
*********************************

1510 IF CASA=0 THEN GO TO 1550
1520 IF F=1 THEN PRINT CHR$ 144;
CHR$ 145;
1530 IF F=2 THEN PRINT CHR$ 146;
CHR$ 147;
1540 GO TO 1580
1550 IF F=1 THEN PRINT CHR$ 148;
CHR$ 149;
1560 IF F=2 THEN PRINT CHR$ 150;
CHR$ 151;
```

```
1580 RETURN
2000 REM
*********************************
     SAIDA POR IMPRESSORA
*********************************

2010 INPUT "SAIDA VIA IMPRESSORA
? (S/N) ";Z$
2020 IF Z$<>"S" AND Z$<>"s" AND
Z$<>"N" AND Z$<>"n" THEN GO TO 2
010
2030 LET X$="  SAIDA POR ECRAN
"
2040 IF Z$="S" OR Z$="s" THEN LE
T X$="SAIDA POR IMPRESSORA"
2050 RETURN
8999 REM
*********************************
              RAINHA
*********************************

9000 FOR a=USR "a" TO USR "a"+7:
 READ aa: POKE a,aa: NEXT a
9010 FOR b=USR "b" TO USR "b"+7:
 READ bb: POKE b,bb: NEXT b
9020 FOR c=USR "c" TO USR "c"+7:
 READ cc: POKE c,cc: NEXT c
9030 FOR d=USR "d" TO USR "d"+7:
 READ dd: POKE d,dd: NEXT d
9040 FOR e=USR "e" TO USR "e"+7:
 READ ee: POKE e,ee: NEXT e
9050 FOR f=USR "f" TO USR "f"+7:
 READ ff: POKE f,ff: NEXT f
9060 FOR g=USR "g" TO USR "g"+7:
 READ gg: POKE g,gg: NEXT g
9070 FOR h=USR "h" TO USR "h"+7:
 READ hh: POKE h,hh: NEXT h
9080 RETURN
9100 DATA 0,9,37,21,23,79,44,45
9110 DATA 0,144,164,168,232,242,
52,180
9120 DATA 28,13,13,31,0,31,31,0
9130 DATA 56,112,176,248,0,248,2
48,0
9140 DATA 255,246,218,234,232,17
6,211,210
9150 DATA 255,111,91,87,23,13,20
3,75
9160 DATA 227,242,242,224,255,22
4,224,255
9170 DATA 199,143,79,7,255,7,7,2
55
```


Capítulo 5

O jogo de Nim

5.1 OS MÉTODOS NA RESOLUÇÃO DE PROBLEMAS

Alguns problemas, convenientemente analisados e para os quais existem algoritmos convenientemente redigidos, levariam milhões de anos a serem executados nos computadores actuais, por mais rápidos que eles sejam.

Tomemos o caso do xadrez: à partida, se jogarmos com as brancas temos a hipótese de escolhermos 20 lances possíveis. O adversário pode também seguidamente, optar entre 20 lances possíveis. Assim, após dois lances de uma partida, há já 400 situações possíveis; para n lances efectuados, chega-se a cerca de 20^n situações possíveis.

Podemos, assim, definir dois tipos de problemas:

— Aqueles que exigem tempos de cálculo demasiadamente elevados, qualquer que seja o método utilizado. É o caso dos problemas de índole puramente combinatória, isto é, para os quais se examinam exaustivamente todas as situações possíveis, as quais são, seguidamente, representadas por uma árvore;

— Aqueles para os quais o método combinatório pode ser melhorado de forma a limitar a arborescência e a torná-la compatível com as capacidades de memória e velocidade de cálculo dos computadores actuais.

Foi efectuado um certo número de pesquisas nesse sentido (algoritmos de ordenação e decisão melhorados, transformada rápida de Fourier[1], etc...) visando encontrar um algoritmo que melhore o tempo de execução.

---

[1] A transformada rápida de Fourier (em inglês, *fast Fourier transform*) é um algoritmo que permite o cálculo da transformada de Fourier discreta. Deve-se a J. Cooley e a J. Tuckey e é muito utilizado em aplicações como a filtragem numérica, a espectroscopia de Fourier ou a resolução de equações diferenciais. *(N. do T.)*

O capítulo precedente utilizava um algoritmo puramente combinatório para enumerar todos os casos possíveis de N rainhas poderem ser colocadas num tabuleiro de xadrez com N × N casas sem que nenhuma delas esteja sob ameaça de qualquer outra.

Este capítulo trata igualmente de um problema *a priori* puramente combinatório. Toma-se à partida um número qualquer de filas de fósforos, contendo cada uma um número qualquer de fósforos. Cada jogador, à vez, retira de uma das filas pelo menos um fósforo. O que tirar o último fósforo perde.

Este jogo, conhecido por jogo de Nim ou jogo de Marienbad *a priori* necessita de um tratamento exaustivo, consistindo em enumerar todas as jogadas possíveis.

Existe, contudo, um algoritmo que permite resolver o problema por um método não combinatório.

Por outro lado, para certos tipos de problemas, se os métodos puramente combinatórios, embora optimizados, originarem tempos de cálculo proibitivos e se não existir qualquer algoritmo que os evite, utilizam-se *heurísticas*.

Podemos pois dizer que uma heurística é um método empírico para permitir encontrar rapidamente a solução de um problema, muito embora não haja a certeza de que essa solução seja sempre encontrada, nem mesmo de saber se o problema está resolvido ou se é insolúvel.

É este tipo de método que vai ser utilizado mais adiante, no programa do *puzzle* 8 e no problema do caixeiro viajante.

5.2 O JOGO DE NIM

Recordemos rapidamente o enunciado do jogo:

Dispõem-se (sobre uma mesa, p. ex.) várias filas de fósforos, sendo cada uma constituída por um número qualquer de fósforos. Cada jogdor tira, à vez, de uma das filas, um número qualquer de fósforos (pelo menos um).

No caso que vamos analisar, seremos nós o primeiro jogador, ou seja, aquele que inicia o jogo.

O computador jogará em segundo lugar.

Deveremos, primeiramente, introduzir o número de filas (FN, sendo FN≤1Ø) e, depois, o número de fósforos por fila (F(), sendo F()≤16).

Vejamos as variáveis utilizadas (ver listagem):

FN	:	número de filas
F(I)	:	número de fósforos por fila I F() é, portanto, um quadro unidimensional, ou vector.
JJ	:	jogador corrente. Se JJ=1, isso significa que é a vez do jogador 1 jogar. Caso contrário, JJ=2.
B(I,T) C T(J)	}	são quadros e variáveis auxiliares, para valores intermédios de cálculo.

Como já dissemos, existe um método que permite determinar se uma situação é vencedora ou não.

Analisemos então o método que permite decidir se uma situação é perdedora ou ganhadora (é possível demonstrá-lo matematicamente, mas não o faremos aqui).

Distinguem-se três casos principais:

① Não resta senão um fósforo em cada fila não vazia. Se o número de filas não vazias é par, o jogador que tem a vez ganha: basta-lhe tirar um fósforo de uma das filas não vazias.

②Uma única fila tem mais do que um fósforo. As restantes, ou já não têm fósforos ou possuem apenas um. Para ganhar, basta reportar-se ao caso vencedor da situação precedente, ou seja, o jogador que tem a vez deve jogar de modo a deixar um número **ímpar** de filas com um único fósforo.

③Nos outros casos, é necessário decompor em binário o número de fósforos de cada fila.

Tomemos, por exemplo, a situação seguinte (clássica):

Fila 1: 1 fósforo		1
Fila 2: 2 fósforos		10
Fila 3: 5 fósforos	em binário:	101
Fila 4: 7 fósforos		+111
		223 (em décimal)

O método diz que, se a soma de cada coluna é par, o jogador adversário perde qualquer que seja a sua defesa.

Caso contrário, existe (pelo menos) uma coluna cuja soma é ímpar, o que significa que o jogador a ter a vez está em situação vencedora. Para ganhar, bastar-lhe-á colocar o jogador contrário num contexto perdedor, ou seja, escolher uma fila e retirar-lhe um certo número de fósforos de forma a que todas as colunas da soma indicada atrás tenham um número par como soma das suas parcelas.

No nosso exemplo, o jogador que tem a vez está em situação vencedora. Para ganhar, ele pode retirar um fósforo à fila 1.

Lista-se, seguidamente, o programa.

Fazendo-o executar, as primeiras linhas impressas no *écran* recordam-nos quais são as variáveis a inicializar, a saber:

— o número de filas.
— o número de fósforos por fila.

```
  1 REM
*********************************
          JOGO DE NIM
*********************************

 10 DIM F(10): DIM B(10,5): DIM
T(5)
```



```
 20 REM
*********************************
       INTRODUZIR DADOS
*********************************

 30 INPUT "NUMERO DE FILAS (<=1
0)? ";FN
 40 IF FN>10 THEN GO TO 30
 50 FOR I=1 TO FN: PRINT "FILA
";I;" --> ";
 60 INPUT "No.DE FOSFOROS ? ";A
: IF A>16 THEN GO TO 60
 70 PRINT "FOSFORO(S) -- ";A
 80 LET F(I)=A: NEXT I: PRINT :
PRINT
 90 GO SUB 620: GO SUB 700: GO
SUB 110: GO TO 90
100 REM
*********************************
      EXAMINAR A SITUACAO
*********************************

110 LET C=0
115 REM
*********************************
            CASO 1
*********************************

120 FOR I=1 TO FN: IF F(I)>1 TH
EN GO TO 210
130 NEXT I
140 FOR I=1 TO FN: IF F(I)<>0 T
HEN LET C=C+1
150 NEXT I
160 IF C=INT (C/2)*2+1 THEN GO
TO 180
170 PRINT
180 PRINT : FOR I=1 TO FN: IF F
(I)=0 THEN NEXT I
190 PRINT "A MINHA JOGADA - TIR
AR 1 FOSFORO                DA
FILA ";I: LET F(I)=0
200 GO TO 540
205 REM
*********************************
            CASO 2
*********************************

210 FOR I=1 TO FN: IF F(I)>1 TH
EN LET C=C+1
220 NEXT I
230 IF C>=2 THEN GO TO 330
240 PRINT : LET C=0
```

```
 250 FOR I=1 TO FN: IF F(I)=1 TH
EN LET C=C+1
 260 NEXT I: IF C=INT (C/2)*2 TH
EN GO TO 300
 270 FOR I=1 TO FN: IF F(I)<2 TH
EN NEXT I
 280 PRINT "A MINHA JOGADA - TIR
AR ";F(I);" FOSFO-"/"
      ROS DA FILA ";I: LET F(I)=
0
 290 GO TO 540
 300 FOR I=1 TO FN: IF F(I)<2 TH
EN NEXT I
 310 PRINT "A MINHA JOGADA - TIR
AR ";F(I)-1;" FOSFO-"/"
         RO";(+"S" AND F(I)>2);"
DA FILA ";I: LET F(I)=1
 320 GO TO 540
 325 REM
**********************************
              CASO 3
**********************************

 330 FOR K=1 TO FN: GO SUB 550
 340 NEXT K
 350 FOR J=1 TO 5: LET T(J)=0: N
EXT J
 360 FOR J=1 TO 5: FOR K=1 TO FN
: LET T(J)=T(J)+B(K,J): NEXT K:
NEXT J
 370 FOR J=1 TO 5: IF T(J)=(INT
(T(J)/2))*2+1 THEN GO TO 420
 380 NEXT J
 390 FOR I=1 TO FN: IF F(I)=0 TH
EN NEXT I
 400 PRINT : PRINT "A MINHA JOGA
DA - TIRAR 1 FOSFORO
      DA FILA ";I: LET F(I)=F(I)-
1
 410 GO TO 540
 420 LET I=1
 430 LET L=0: LET F1=F(I)
 435 IF F(I)=0 THEN GO TO 510
 440 LET L=L+1: LET F(I)=F1-L: L
ET F(K)=F(I)
 450 FOR K=1 TO FN: GO SUB 550
 460 NEXT K
 470 FOR J=1 TO 5: LET T(J)=0: N
EXT J: FOR J=1 TO 5: FOR K=1 TO
FN: LET T(J)=T(J)+B(K,J): NEXT K
: NEXT J
 480 FOR J=1 TO 5: IF T(J)=INT (
T(J)/2)*2 THEN NEXT J: GO TO 530
```



```
 490 IF L<F1 THEN GO TO 440
 500 LET F(I)=F1
 510 IF I<FN THEN LET I=I+1: GO
TO 430
 520 GO TO 400
 530 PRINT : PRINT "A MINHA JOGA
DA - TIRAR ";L;" FOSFO-"/"
         RO";(+"S" AND L>1);"
DA FILA ";I
 540 RETURN
 545 REM
**********************************
  CONVERSAO DO No. DE FOSFOROS
          EM BINARIO
**********************************

 550 LET N=F(K)
 560 FOR J=1 TO 5: LET B(K,J)=0:
 NEXT J
 570 FOR J=1 TO 5: IF N=0 THEN R
ETURN
 580 LET B(K,J)=N-INT (N/2)*2
 590 LET N=INT (N/2)
 600 NEXT J
 610 RETURN
 615 REM
**********************************
      JOGADA DO JOGADOR 1
**********************************

 620 LET JJ=1: POKE 23692,0: PRI
NT : GO SUB 700
 630 PRINT "E'A SUA VEZ DE JOGAR
---": PRINT : PRINT : FOR I=1 TO
 FN: PRINT "FILA ";I;"--> ";F(I)
;" FOSFORO";(+"S" AND F(I)<>1)
 640 NEXT I
 650 INPUT "FILA --> ";I,"FOSFOR
OS--> ";J
 660 IF I>FN THEN PRINT #0;"No.D
E FILA INCOERENTE, REPITA...": P
AUSE 150: GO TO 650
 665 IF J>F(I) THEN PRINT #0;"No
.DE FOSFOROS INCOERENTE,REPITA":
 PAUSE 150: GO TO 650
 670 IF J=0 THEN GO TO 650
 675 PRINT '/"A SUA JOGADA - FIL
A -----> ";I'TAB 15;"FOSFOROS--->
 ";J
 680 LET F(I)=F(I)-J: PRINT : PR
INT : FOR I=1 TO FN: PRINT "FILA
 ";I;" --> ";F(I);" FOSFORO";(+"
S" AND F(I)<>1)
```

```
 690 NEXT I: LET JJ=2: RETURN
 695 REM
*********************************
          FILA VAZIA ?
*********************************

 700 FOR K=1 TO FN: IF F(K)=0 TH
EN NEXT K: PRINT : PRINT : GO TO
 720
 710 RETURN
 720 PRINT "JOGO TERMINADO": IF
JJ=1 THEN PRINT "VOCE GANHOU ...
 SNIF...": STOP
 730 PRINT "GANHEI EU!!!...": ST
OP
```

5.3 O PROGRAMA

As linhas 10 a 80 permitem dimensionar os quadros utilizados e introduzir os dados.

Como já dissemos, os limites de utilização são os seguintes:

FN, número de filas, limitado a 10.

O número de fósforos por fila, F(I), é limitado a 16.

Vejamos agora as sub-rotinas:

700–730

Chamada sistematicamente antes de cada jogador efectuar a sua jogada, esta sub-rotina verifica se existe ainda alguma fila não vazia. Caso contrário, a partida terminou e o último jogador a ter vez perdeu.

620–690

Esta sub-rotina segue a jogada a efectuar pelo jogador 1 (JJ=1)

A rotina determina, primeiramente, se a partida terminou ou não (por chamada da sub-rotina 700–730) e imprime a situação corrente. Pede, seguidamente, ao jogador 1 que:

— escolha uma fila
— indique o número de fósforos a retirar dela.

Se o número de fósforos for demasiado grande, é impressa uma mensagem de erro e o número é novamente pedido, após uma curta pausa (linha 660). O número da fila escolhida também é controlado.

550–610

Dada uma fila K, esta sub-rotina converte em binário o número N de fósforos existentes nessa fila. O quadro B (K,J) — com $1 \leq J \leq 5$ — contém o resultado da conversão.

110– 540

Esta sub-rotina executa a jogada a efectuar pelo computador. A estratégia deste é, evidentemente, baseada nos casos ①, ② e ③ que examinámos atrás. Assim, temos:

Caso ① linhas 120–200

Cada fila contém, no máximo, um único fósforo. Utiliza-se um contador C, posto a 0 à partida. Segundo o que já foi dito, calcula-se C. Se C for ímpar, a situação é perdedora; vai-se à linha 180 e retira-se um fósforo à primeira fila disponível. Caso contrário, salta-se igualmente para a linha 180, seguidamente para a linha 540 e RETURN, saindo-se da sub-rotina.

Caso ② linhas 210–320

O contador C é posto a zero à partida (linha 110).

Conta-se o número de filas que contêm um só fósforo e armazena-se em C.

Se C for ímpar, a jogada a efectuar consiste em retirar 2 fósforos à fila que os tenha.

Caso contrário, é necessário retirar um só fósforo à fila que tenha 2.

Caso ③ llinhas 330–410

Neste caso, é necessário converter em binário cada fila. É chamada a sub-rotina 550–610 e preenche-se o quadro B (K,J). Isto é feito nas linhas 330 e 340.

Deve-se, seguidamente, efectuar a soma de cada coluna e

guardar o resultado no quadro T(J). Isto é executado pelas linhas 350 e 360.

Depois, procede-se como foi dito anteriormente:

— testam-se os valores T(J), nas linhas 370 e 380;
— linhas 390–410, o computador está em situação perdedora: retira-se um fósforo à primeira fila não vazia;
— caso contrário, linhas 420–540, o computador está em situação ganhadora: é necessário escolher 1 fila e um certo número de fósforos de forma a colocar o jogador contrário em situação perdedora.

Na linha 90, o programa efectua as chamadas às várias sub-rotinas:
— o primeiro jogador é o utilizador (chamada da sub-rotina 620);
— o segundo jogador é o computador, o qual verifica se está em situação vencedora ou perdedora para, seguidamente, jogar em consequência.

Esta descrição completa do algoritmo deverá possibilitar a compreensão, passo a passo, de como funciona e de como foi realizado este programa. Contudo, quem o desejar, pode tomar como exemplo a versão com função de traçagem do programa do problema das 8 rainhas e adaptar o presente programa de modo a obter uma *hard-copy* do traçado (se possuir uma impressora).

Na listagem deste programa apenas foram incluídas quatro instruções próprias do Basic Sinclair do *Spectrum* e similares; trata-se do POKE da linha 620, destinado a permitir que a imagem no *écran* avance sem pedir o conhecido «scroll?», do conjunto entre parêntesis das linhas 310, 530, 630 e 680, destinado a distinguir o caso plural do caso singular relativo ao número de fósforos, e, finalmente, os PRINT#0 das linhas 660 e 665 para a emissão das mensagens de erro, acompanhados dos PAUSE. Suprimindo essas instruções, o programa ficará num BASIC compatível com o da maioria dos microcomputadores do mercado (para os que exigirem a distinção de variáveis inteiras, fazer F%(I), B%(K,J) e T%(J) nas variáveis respectivas, não sendo necessário nas outras).

Apresentam-se, seguidamente, alguns exemplos de execução do programa.

Exemplos

```
**********************************
          JOGO DE NIM
**********************************

NUMERO DE FILAS (<=10)? 3

FILA 1 --> FOSFORO(S)  -- 14
FILA 2 --> FOSFORO(S)  -- 2
FILA 3 --> FOSFORO(S)  -- 2


E'A SUA VEZ DE JOGAR---

FILA 1--> 14 FOSFOROS
FILA 2--> 2 FOSFOROS
FILA 3--> 2 FOSFOROS

FILA --> 1       FOSFOROS--> 3

FILA 1 --> 11 FOSFOROS
FILA 2 --> 2 FOSFOROS
FILA 3 --> 2 FOSFOROS

A MINHA JOGADA - TIRAR 11 FOSFO-
                 ROS DA FILA 1

E'A SUA VEZ DE JOGAR---

FILA 1--> 0 FOSFOROS
FILA 2--> 2 FOSFOROS
FILA 3--> 2 FOSFOROS

FILA --> 2       FOSFOROS--> 1

FILA 1 --> 0 FOSFOROS
FILA 2 --> 1 FOSFORO
FILA 3 --> 2 FOSFOROS
```

```
A MINHA JOGADA - TIRAR 2 FOSFO-
                 ROS DA FILA 3

E'A SUA VEZ DE JOGAR---

FILA 1--> 0 FOSFOROS
FILA 2--> 1 FOSFORO
FILA 3--> 0 FOSFOROS

FILA --> 2       FOSFOROS--> 1

FILA 1 --> 0 FOSFOROS
FILA 2 --> 0 FOSFOROS
FILA 3 --> 0 FOSFOROS

JOGO TERMINADO
GANHEI EU!!!...


**********************************
           JOGO DE NIM
**********************************

NUMERO DE FILAS (<=10)? 5

FILA 1 --> FOSFORO(S)  -- 1
FILA 2 --> FOSFORO(S)  -- 0
FILA 3 --> FOSFORO(S)  -- 1
FILA 4 --> FOSFORO(S)  -- 1
FILA 5 --> FOSFORO(S)  -- 1


E'A SUA VEZ DE JOGAR---

FILA 1--> 1 FOSFORO
FILA 2--> 0 FOSFOROS
FILA 3--> 1 FOSFORO
FILA 4--> 1 FOSFORO
FILA 5--> 1 FOSFORO

FILA --> 3       FOSFOROS--> 1

FILA 1 --> 1 FOSFORO
FILA 2 --> 0 FOSFOROS
FILA 3 --> 0 FOSFOROS
```



```
FILA 4 --> 1 FOSFORO
FILA 5 --> 1 FOSFORO

A MINHA JOGADA - TIRAR 1 FOSFORO
                 DA FILA 1

E'A SUA VEZ DE JOGAR---

FILA 1--> 0 FOSFOROS
FILA 2--> 0 FOSFOROS
FILA 3--> 0 FOSFOROS
FILA 4--> 1 FOSFORO
FILA 5--> 1 FOSFORO

FILA --> 4       FOSFOROS--> 1

FILA 1 --> 0 FOSFOROS
FILA 2 --> 0 FOSFOROS
FILA 3 --> 0 FOSFOROS
FILA 4 --> 0 FOSFOROS
FILA 5 --> 1 FOSFORO

A MINHA JOGADA - TIRAR 1 FOSFORO
                 DA FILA 5


JOGO TERMINADO
VOCE GANHOU ... SNIF...


**********************************
           JOGO DE NIM
**********************************

NUMERO DE FILAS (<=10)? 3

FILA 1 --> FOSFORO(S)  -- 16
FILA 2 --> FOSFORO(S)  -- 4
FILA 3 --> FOSFORO(S)  -- 12


E'A SUA VEZ DE JOGAR---

FILA 1--> 16 FOSFOROS
FILA 2--> 4 FOSFOROS
FILA 3--> 12 FOSFOROS
```

```
FILA --> 3       FOSFOROS--> 2


FILA 1 --> 16 FOSFOROS
FILA 2 --> 4 FOSFOROS
FILA 3 --> 10 FOSFOROS

A MINHA JOGADA - TIRAR 2 FOSFO-
                 ROS DA FILA 1

E'A SUA VEZ DE JOGAR---

FILA 1--> 14 FOSFOROS
FILA 2--> 4 FOSFOROS
FILA 3--> 10 FOSFOROS


FILA --> 2       FOSFOROS--> 3


FILA 1 --> 14 FOSFOROS
FILA 2 --> 1 FOSFORO
FILA 3 --> 10 FOSFOROS

A MINHA JOGADA - TIRAR 3 FOSFO-
                 ROS DA FILA 1

E'A SUA VEZ DE JOGAR---

FILA 1--> 11 FOSFOROS
FILA 2--> 1 FOSFORO
FILA 3--> 10 FOSFOROS


FILA --> 1       FOSFOROS--> 10


FILA 1 --> 1 FOSFORO
FILA 2 --> 1 FOSFORO
FILA 3 --> 10 FOSFOROS

A MINHA JOGADA - TIRAR 9 FOSFO-
                 ROS DA FILA 3

E'A SUA VEZ DE JOGAR---

FILA 1--> 1 FOSFORO
FILA 2--> 1 FOSFORO
FILA 3--> 1 FOSFORO


FILA --> 1       FOSFOROS--> 1
```



```
FILA 1 --> 0 FOSFOROS
FILA 2 --> 1 FOSFORO
FILA 3 --> 1 FOSFORO

A MINHA JOGADA - TIRAR 1 FOSFORO
                 DA FILA 2

E'A SUA VEZ DE JOGAR---

FILA 1--> 0 FOSFOROS
FILA 2--> 0 FOSFOROS
FILA 3--> 1 FOSFORO


FILA --> 3       FOSFOROS--> 1


FILA 1 --> 0 FOSFOROS
FILA 2 --> 0 FOSFOROS
FILA 3 --> 0 FOSFOROS


JOGO TERMINADO
GANHEI EU!!!...



**********************************
          JOGO DE NIM
**********************************

NUMERO DE FILAS (<=10)? 3

FILA 1 --> FOSFORO(S)  -- 1
FILA 2 --> FOSFORO(S)  -- 5
FILA 3 --> FOSFORO(S)  -- 7


E'A SUA VEZ DE JOGAR---

FILA 1--> 1 FOSFORO
FILA 2--> 5 FOSFOROS
FILA 3--> 7 FOSFOROS


FILA --> 3       FOSFOROS--> 3


FILA 1 --> 1 FOSFORO
```

```
FILA 2 --> 5 FOSFOROS
FILA 3 --> 4 FOSFOROS

A MINHA JOGADA - TIRAR 1 FOSFORO
                 DA FILA 1

E'A SUA VEZ DE JOGAR---

FILA 1--> 0 FOSFOROS
FILA 2--> 5 FOSFOROS
FILA 3--> 4 FOSFOROS

FILA --> 2       FOSFOROS--> 1

FILA 1 --> 0 FOSFOROS
FILA 2 --> 4 FOSFOROS
FILA 3 --> 4 FOSFOROS

A MINHA JOGADA - TIRAR 1 FOSFORO
                 DA FILA 2

E'A SUA VEZ DE JOGAR---

FILA 1--> 0 FOSFOROS
FILA 2--> 3 FOSFOROS
FILA 3--> 4 FOSFOROS

FILA --> 3       FOSFOROS--> 1

FILA 1 --> 0 FOSFOROS
FILA 2 --> 3 FOSFOROS
FILA 3 --> 3 FOSFOROS

A MINHA JOGADA - TIRAR 1 FOSFORO
                 DA FILA 2

E'A SUA VEZ DE JOGAR---

FILA 1--> 0 FOSFOROS
FILA 2--> 2 FOSFOROS
FILA 3--> 3 FOSFOROS

FILA --> 3       FOSFOROS--> 1

FILA 1 --> 0 FOSFOROS
FILA 2 --> 2 FOSFOROS
FILA 3 --> 2 FOSFOROS
```



```
A MINHA JOGADA - TIRAR 1 FOSFORO
                 DA FILA 2

E'A SUA VEZ DE JOGAR---

FILA 1--> 0 FOSFOROS
FILA 2--> 1 FOSFORO
FILA 3--> 2 FOSFOROS

FILA --> 3       FOSFOROS--> 2

FILA 1 --> 0 FOSFOROS
FILA 2 --> 1 FOSFORO
FILA 3 --> 0 FOSFOROS

A MINHA JOGADA - TIRAR 1 FOSFORO
                 DA FILA 2


JOGO TERMINADO
VOCE GANHOU ... SNIF...
```

Capítulo 6

O «puzzle» 8

Os capítulos anteriores apresentavam dois jogos: o jogo de Nim ou de Marienbad e o problema das rainhas que tinham de ser colocadas sobre um tabuleiro de xadrez de modo a que nenhuma delas fique em posição de ser tomada por (ou tomar) outra. Vimos que, para o jogo de Nim, existia um procedimento de decisão que permitia evitar o método combinatório. Em relação ao problema das rainhas, contentámo-nos em desenvolver exaustivamente a árvore de decisões a tomar e em evitar os nós impossíveis pelo processo de *back-track* aliado a uma função de decisão.

No caso do *puzzle* 8 e do jogo do galo, raciocina-se sobre o que, em inteligência artificial, se denomina por *espaço de estados*. Cada estado é uma configuração do jogo num dado momento. O dito espaço de estados é representado por uma árvore cujos nós são os diferentes estados considerados.

Neste capítulo, iremos passar em revista, num mesmo exemplo, métodos combinatórios e, seguidamente, heurísticos para resolver um problema bem preciso: partindo de um *dado estado S*, qual, de entre todos, o melhor caminho para chegar a um *dado estado G*?

6.1. A REPRESENTAÇÃO ADOPTADA

O nosso problema é o seguinte: partindo de uma situação dada S, deseja-se chegar à situação G o mais depressa possível. Para isso, é bem evidente que, a partir da situação S, pode ser gerado um bom número de situações, e assim por diante a partir dessas situações intermédias.

Desenvolve-se assim uma árvore de pesquisa cuja *raiz* é o nó

S e o objectivo ou *finalidade* o nó G [1]. A árvore em questão também se designa por «espaço de estados». As diversas situações geradas a partir de um nó corrente designar-se-ão por *sucessores* desse nó.

Os nós que ainda não tenham sido desenvolvidos ou explorados (a partir dos quais ainda não foram gerados sucessores) serão chamados nós da *lista aberta*.

Pelo contrário, caso eles já tenham sido desenvolvidos, pertencerão à lista fechada. De um modo muito genérico, é esta a terminologia normalmente utilizada para formalizar ou definir este tipo de problema.

Examinemos o nosso jogo.

Conhecido sob vários nomes, o *puzzle* 8 existe também numa versão de 15 piões (a mais próxima da versão original, dos finais do século XIX) e de inúmeras versões «gráficas», onde cada pião constitui um fragmento de um desenho a recompor.

No caso que vamos considerar, o de 8 piões, dispõe-se de um tabuleiro quadriculado de 3×3, com 8 das suas casas ocupadas por 8 algarismos de 1 a 8, estando a última casa vazia. Uma configuração possível do tabuleiro poderá ser, por exemplo, a seguinte:

1	2	4
5	3	7
6	.	8

Se fizermos as contas, poderemos verificar que existem

9!=362 880 estados possíveis.

Aquilo a que chamámos espaço de estados é, na verdade, finito, mas de dimensões impressionantes...

[1] As letras S e G são as iniciais dos substantivos ingleses *Start* (partida, saída) e *Goal* (objectivo, golo). *(N. do T.)*

Estabeleçamos um espaço de partida S e um estado de chegada (objectivo) G:

1	2	3
4	5	6
7	8	.

estado S

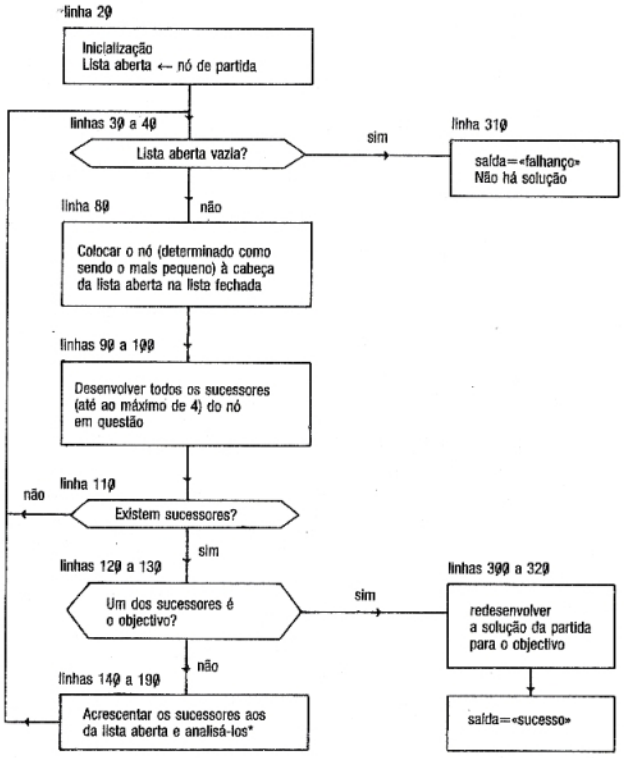
1	2	3
7	4	6
5	8	.

estado G

Resolvamos então o nosso problema: passar o mais rapidamente possível de S a G. Para isso, utilizaremos quatro métodos de exploração da árvore. Os dois primeiros, puramente combinatórios, não irão dar, geralmente, senão resultados pouco satisfatórios. Os dois últimos utilizam heurísticas que lhes melhoram sensivelmente as capacidades de execução. Mas analisemos primeiramente o algoritmo genérico que iremos utilizar.

6.2. O ALGORITMO

Em primeiro lugar, vejamos o fluxograma geral do programa, qualquer que seja o método de pesquisa utilizado:



(*) Isto na condição limite de o número de nós já desenvolvidos ser inferior a 1ØØ, dimensão reservada em memória no nosso programa (linha 1Ø).

Vejamos algumas das variáveis utilizadas e cujos valores são impressos em *écran*:

S$ contém o *puzzle* de partida (S) sob forma linearizada (cadeia de caracteres ou *string*).

G$ contém o *puzzle* de chegada (G), sob forma idêntica.

D$(N1) contém, sob a mesma forma, o *puzzle* (ou estado) constituído pelo nó N1, sendo D(N1) o valor desse nó, calculado diferentemente, dependendo da estratégia de pesquisa que for utilizada.

Exemplo:

Consideremos o puzzle de partida:

1 2 3
4 5 6
7 8 .

S$ contém, portanto, a cadeia de caracteres "12345678.".

À partida, tem-se apenas um nó e, assim N1=1
D(N1)=1

Como existe um único nó desenvolvido, tem-se D4=1

Esta variável contém, no nosso programa, a cada momento, a soma dos nós que figuram na lista aberta e fechada. A pesquisa é detida se D4 atingir o valor 100, o que nos parece um tempo de cálculo proibitivo para constatar que não se atingiu a solução do problema.

Recordemos as regras do jogo: tem-se o direito, em cada jogada, de deslocar unicamente a casa vazia, ou seja, de deslocar uma casa adjacente a ela para a sua posição, ficando a casa vazia no lugar precedentemente ocupado pela casa adjacente (uma troca de posições, portanto). É possível efectuar, em cada jogada, até quatro movimentos, ou lances, isto é, pode-se deslocar a casa vazia para cima *ou* para baixo *ou* para a esquerda *ou* para a direita.

No exemplo 1 2 3 há apenas a possibilidade de deslo-
4 5 6
7 8 .

car a casa vazia para cima ou para a esquerda.

A partir de uma situação ou estado dado, portanto de um nó dado, podem-se gerar, no máximo, quatro sucessores.

Na linha 500 da listagem tem início a sub-rotina que efectua a inicialização do programa: são introduzidos, em particular, S$ e G$ e depois aparece a forma sob a qual cada nó irá ser identificado na sequência do programa:

D$(1)="00 1C12345678."

Os três primeiros caracteres indicam o número do nó a partir do qual se faz o movimento.

A letra «C» indica aqui que se está no começo da árvore (estado S).

Os restantes nove caracteres dão a forma linearizada do *puzzle* ou estado.

Nos exemplos de execução do programa que apresentamos mais adiante, há a impressão de D$ (portanto do nó corrente) e do valor D( ) desse nó.

Com efeito, se analisarmos cuidadosamente os exemplos de execução apresentados, podemos verificar que o início da cadeia D$ contém a referência do nó a partir da qual se construiu efectivamente o nó D$.

Assim, tomemos o nó de partida:

D$(1)="00 1C12345678."

1 2 3
4 5 6
7 8 .

O nó 2 é representado pela cadeia

D$(2)=«001B12345.786»

1 2 3
4 5 .
7 8 6

O início da cadeia 001B indica que o nó 2 foi obtido a partir do nó 1 deslocando uma casa para baixo (neste caso, a casa 6).

A nossa função de avaliação calcula o valor de cada nó; assim, o valor do nó 2 é 5.

Não vamos efectuar aqui uma descrição detalhada e exaustiva do algoritmo e da sua programação; limitar-nos-emos a apresentar uma explicação genérica destinada a servir de guia no estudo do algoritmo.

Assim, tendo já analisado o fluxograma geral do programa, vejamos mais alguns elementos:

A lista aberta contém os nós ainda não desenvolvidos.

A lista fechada contém os nós já desenvolvidos (ou sejam, aqueles para os quais foram encontrados sucessores).

Se não houver mais nós a desenvolver e se o objectivo (estado G) ainda não tiver sido atingido, há uma situação de fracasso e a impressão de uma mensagem correspondente (linha 310), com paragem do programa.

Caso contrário, escolhe-se o nó que tiver o valor mais pequeno encontrado graças à função de avaliação: verifica-se se entre os sucessores o objectivo foi atingido e, caso afirmativo, o programa pára (sucesso); senão, acrescentam-se os sucessores aos da lista aberta e continua-se.

Adoptámos aqui quatro funções de avaliação diferentes, determinando cada uma um tipo de percurso diferente da árvore.

As duas primeiras funções utilizadas (cada uma delas executada pela sub-rotina com início na linha 1400 da listagem) levam aos seguintes tipos de percurso:

1) A pesquisa em largura primeiro.
   Neste caso, o valor da função f será dado por
   f=(D(N1)+1)−5000

2) A pesquisa em profundidade primeiro.
   f=(D(N1)−5000)−1

O primeiro capítulo deste livro contém a explicação detalhada de cada um deste tipos de percurso de pesquisa.

Qualquer destes tipos de pesquisa é puramente combinatório e, como iremos verificar nos exemplos de execução apresentados adiante, raramente permitem que a solução seja atingida rapidamente: a pesquisa é perfeitamente «cega».

Os outros dois tipos de métodos são métodos heurísticos que incorporam alguns elementos de «bom senso», os quais se destinam a dirigir a pesquisa para nós susceptíveis de serem etapas intermediárias no sentido do objectivo a atingir.

Quando utilizam boas heurísticas, estes métodos conduzem em geral muito mais rapidamente à solução, mas também acontece por vezes passar-se «ao lado» dela sem a atingir.

A primeira desta funções calcula o número de posições diferentes entre o *puzzle* ou estado corrente e o *puzzle* ou estado de chegada (objectivo).

A segunda função calcula, de certo modo, a distância mínima entre o *puzzle* corrente e o *puzzle* de chegada. A função verifica qual a posição (x,y) que cada casa ocupa no *puzzle* corrente e qual a sua posição no *puzzle* de chegada.

Assim, temos:

1 2 3	1 2 3
4 5 6	4 5 6
7 8 .	. 8 7
① *puzzle* corrente	② *puzzle* de chegada

— a casa 7 encontra-se na posição (3,1) ou seja, na 3.ª linha, 1.ª coluna, no estado ou *puzzle* ①.

— a casa 7 encontra-se na posição (3,3) no estado ②.

Para a casa em questão, a distância é 2. Procede-se de modo idêntico para cada casa, adicionando as diferenças em número de linha e de coluna e, seguidamente, adicionam-se as «distâncias» de cada uma das casas.

Esta última função tem, de certa forma, em conta o estado do *puzzle* a cada instante e permite seleccionar melhor as jogadas a efectuar. É esta a função que, de entre as quatro, dá os melhores resultados.

Mas, de preferência, vejamos os exemplos.

6.3. LISTAGEM E EXEMPLOS DE EXECUÇÃO

Antes de mais, eis a listagem do programa.

As linhas 10 a 1380, que formam a parte principal do programa, são idênticas para os quatro métodos diferentes que nos propomos analisar.

A função de avaliação, que determina o percurso da árvore, é programada a partir da linha 1400 em cada um dos quatro casos. Assim, começamos por listar a parte comum do programa, sem a rotina da linha 1400, a qual, variando de método para método, será apresentada antes dos exemplos, referentes a cada um dos métodos.

```
   5 REM
*********************************
          PUZZLE 8
*********************************

  10 DIM A$(30,13): DIM U$(30):
DIM D$(100,13): DIM D(100): DIM
U$(30,9): DIM K$(3,3): DIM X$(3,
3): DIM X(4): DIM Y(4)
  20 GO SUB 500
  30 IF E$="YES" THEN GO TO 300
  40 GO SUB 1000: IF D(N1)>=5999
 THEN GO TO 300
  45 POKE 23692,0
  50 PRINT : PRINT "NO ";  FLASH
```



```
 1;N1; FLASH 0;"  ";D$(N1);"  VA
LOR ";D(N1): PRINT
  60 FOR B=1 TO 3: FOR C=1 TO 3:
 PRINT D$(N1)(LEN D$(N1)-8 TO )(
3*(B-1)+C);" ";: NEXT C: PRINT
  70 NEXT B: PRINT
  80 LET D5=D5+1: LET D(N1)=5000
+D(N1)
  90 LET H$=D$(N1)(5 TO 13): LET
 J$=D$(N1)(4)
 100 GO SUB 700: GO SUB 1200
 110 IF G1<=0 THEN GO TO 30
 120 LET F$="NO": FOR A=1 TO G1:
 GO SUB 900
 130 IF F$="YES" THEN LET E$="YE
S"
 140 LET H$=A$(A)(5 TO 13): GO S
UB 700
 150 PRINT A$(A)
 160 GO SUB 1400
 170 IF D4>=99 THEN GO TO 300
 180 LET D4=D4+1: LET D$(D4)=A$(
A): LET D(D4)=P5: NEXT A
 190 GO TO 30
 300 IF F$="YES" THEN GO SUB 130
0: GO TO 320
 310 PRINT : PRINT "************
********************": PRINT : P
RINT "NAO HA SOLUCOES"
 320 PRINT : PRINT "LISTA ABERTA
 ---> ";D4-D5;" NOS": PRINT : P
RINT "LISTA FECHADA ---> ";D5;"
NOS"
 330 PRINT : PRINT "************
********************"
 340 STOP
 500 REM
*********************************
          INICIALIZACAO
*********************************

 510 DATA -1,0,0,1,1,0,0,-1
 520 FOR I=1 TO 4: READ X(I),Y(I
): NEXT I
 530 PRINT : PRINT "*INTRODUZIR
O PUZZLE DE PARTIDA*";TAB 11;"
(PUZZLE 5)"''''"DEVE-O INTRODUZIR
 COMO UMA UNICACADEIA DE CARACTE
RES,ISTO E',TO-DAS AS TRES ""LIN
HAS"" DO QUADRADONUMA UNICA LINH
A CONTINUA."
 540 INPUT "PUZZLE 5 --> ";5$: I
F LEN 5$<>9 THEN GO TO 540
 545 CLS
```

```
 550 PRINT : PRINT "*INTRODUZIR
O PUZZLE DE CHEGADA*"'';TAB 11;"
(PUZZLE G)"''"DEVE-O INTRODUZIR
 COMO UMA UNICACADEIA DE CARACTE
RES,ISTO E',TO-DAS AS TRES ""LIN
HAS"" DO QUADRADONUMA UNICA LINH
A CONTINUA."
 560 INPUT "PUZZLE G --> ";G$: I
F LEN G$<>9 THEN GO TO 560
 565 CLS
 570 LET Y$="ADBEC": LET B$="BEA
DC": LET J$="C": LET D(1)=0: LET
 D4=1: LET N1=1: GO SUB 600: LET
 D$(1)=Z$+"C"+S$
 580 LET D5=0: LET E$="NO": LET
F$="NO": RETURN
 600 REM
*********************************
  CONVERSAO DE N1 EM CADEIA DE
          CARACTERES
*********************************

 610 LET Z$=STR$ N1: LET Z=LEN Z
$: LET Z=3-Z
 620 IF Z=0 THEN RETURN
 630 FOR I=1 TO Z: LET Z$="0"+Z$
: NEXT I: RETURN
 700 REM
*********************************
    TRANSFORMAR H$ EM QUADRO
*********************************

 710 FOR I=1 TO 3: FOR J=1 TO 3
 720 LET K=3*(I-1)+J
 730 LET K$(I,J)=H$(K): LET X$(I
,J)=H$(K): NEXT J: NEXT I
 740 RETURN
 800 REM
*********************************
    TRANSFORMAR UM QUADRO EM
      CADEIA DE CARACTERES
*********************************

 810 LET I$="": FOR I=1 TO 3: FO
R J=1 TO 3
 820 LET I$=I$+X$(I,J): NEXT J:
NEXT I
 830 RETURN
 900 REM
*********************************
    VERIFICAR SE O OBJECTIVO
         FOI ALCANCADO
*********************************
```



```
 910 LET P$=A$(A)(LEN A$(A)-8+1
TO ): IF G$=P$ THEN LET F$="YES"
: LET D6=D4+1
 920 RETURN
1000 REM
*********************************
  ENCONTRAR O NO'A DESENVOLVER
*********************************

1010 LET N1=1: LET P1=5999: FOR
I=1 TO D4
1020 IF D(I)>=P1 THEN GO TO 1040
1030 LET P1=D(I): LET N1=I
1040 NEXT I: RETURN
1100 REM
*********************************
   VER SE A SOLUCAO JA'EXISTE
*********************************

1110 LET P$="NAO"
1120 FOR I=1 TO D4: IF D(I)<5000
 THEN GO TO 1150
1130 LET O$=D$(I)(5 TO 13)
1140 IF O$=I$ THEN LET P$="SIM":
 RETURN
1150 NEXT I
1160 IF P$="NAO" THEN RETURN
1200 REM
*********************************
   DESENVOLVER OS SUCESSORES
*********************************

1205 FOR I=1 TO 5: IF J$=B$(I) T
HEN GO TO 1220
1210 NEXT I
1220 LET Q$=Y$(I)
1225 FOR X=1 TO 3: FOR Y=1 TO 3
1230 IF K$(X,Y)="," THEN GO TO 1
240
1235 NEXT Y: NEXT X
1240 LET G1=0
1245 LET P1=1: LET P9=0
1250 IF P1>4 THEN GO TO 1295
1255 IF B$(P1)=Q$ THEN GO TO 129
0
1260 LET X2=X+X(P1): LET Y2=Y+Y(
P1): FOR I=1 TO 3: FOR J=1 TO 3:
 LET X$(I,J)=K$(I,J): NEXT J: NE
XT I
1265 IF X2<1 OR X2>3 THEN GO TO
1290
1270 IF Y2<1 OR Y2>3 THEN GO TO
1290
```

```
1275 LET X$(X,Y)=X$(X2,Y2): LET
X$(X2,Y2)=",": GO SUB 800: GO SU
B 1100
1280 IF P$="SIM" THEN GO TO 1290
1285 LET P9=P9+1: GO SUB 600: LE
T G1=G1+1: LET A$(P9)=Z$+B$(P1)+
I$
1290 LET P1=P1+1: GO TO 1250
1295 RETURN
1300 REM
*********************************
    REDESENVOLVER A SOLUCAO
*********************************

1310 LET P5=0
1320 LET T$=D$(D6)(4): IF T$="C"
 THEN GO TO 1340
1330 LET P5=P5+1: LET U$(P5)=T$:
 LET W$(P5)=D$(D6)(LEN D$(D6)-8
TO )
1340 LET V$=D$(D6)( TO 3): LET D
6=VAL V$: IF T$<>"C" THEN GO TO
1320
1350 PRINT : PRINT "*******  EIS
 A SOLUCAO : *******": PRINT
1360 FOR I=P5 TO 1 STEP -1: PRIN
T "   ";U$(I);" --> ";W$(I): NEXT
 I
1370 PRINT : PRINT "EM ";P5;" ET
APA";(+"S" AND P5<>1): PRINT : P
RINT
1375 PRINT "********************
************"
1380 RETURN
1400 REM         ATENCAO !!!
*********************************
 ===> ESTA LINHA DEVE SER SUBS-
      TITUIDA PELA LINHA OU LI-
      NHAS LISTADAS NOS PARAGRA-
      FOS SEGUINTES !!!
*********************************
```

6.3.1. O percurso dito de largura primeiro

Para que o programa utilize este percurso, devemos acrescentar à listagem anterior a linha seguinte, a qual programa o método combinatório respectivo:

1400 LET P5 = (D(N1) + 1) − 5000: RETURN

Ao fazermos o computador executar o programa, este pede-

-nos para introduzirmos o *puzzle* de partida e, depois, o *puzzle* de chegada. Apesar de o programa dispor de uma pequena rotina para controlar o número de caracteres introduzidos para cada um dos *puzzles*, devemos ter cuidado a introduzir cada *puzzle*, pois um carácter repetido ou não permitido (diferente dos algarismos 1 a 8 e do ponto) levará inevitavelmente ao mau funcionamento do programa.

Os exemplos de execução apresentados seguidamente foram impressos por uma versão do programa destinada a permitir uma *hard-copy* da execução em impressora, versão essa listada no fim do presente capítulo. Não utilizando essa versão, se acharmos que a imagem no *écran* passa (*scrolls*) demasiado depressa para permitir uma análise conveniente do *output*, devemos omitir a linha 45 na listagem original.

```
*********************************
              PUZZLE 8
*********************************

PUZZLE DE PARTIDA (S)> 123.46758

1 2 3
. 4 6
7 5 8


PUZZLE DE CHEGADA (G)> 23.146758

2 3 .
1 4 6
7 5 8


*********************************


NO' 1   001C123.46758   VALOR 0

1 2 3
. 4 6
7 5 8

001B.23146758
001E1234.6758
001A123746.58
```

```
NO' 2  001B.23146758  VALOR 1

. 2 3
1 4 6
7 5 8

002E2.3146758

NO' 3  001E1234.5758  VALOR 1

1 2 3
4 . 6
7 5 8

003B1.3426758
003E12346.758
003A1234567.8

NO' 4  001A123746.58  VALOR 1

1 2 3
7 4 6
. 5 8

004E1237465.8

NO' 5  002E2.3146758  VALOR 2

2 . 3
1 4 6
7 5 8

005E23.146758
005A2431.6758

*******  EIS A SOLUCAO : *******

   B --> .23146758
   E --> 2.3146758
   E --> 23.146758

EM 3 ETAPAS

**********************************

LISTA ABERTA  ---> 6 NOS

LISTA FECHADA ---> 5 NOS

**********************************
```


Vejamos agora um caso um pouco mais complicado, que submeteremos sucessivamente aos quatro métodos. Como verificaremos, o percurso de largura primeiro não permite obter a solução rapidamente. Como dissemos, caso a solução não seja entretanto obtida, o programa pára quando tiverem sido desenvolvidos 100 nós.

```
**********************************
            PUZZLE 8
**********************************

PUZZLE DE PARTIDA (S)> 12345678.

1 2 3
4 5 6
7 8 .

PUZZLE DE CHEGADA (G)> 12374658.

1 2 3
7 4 6
5 8 .

**********************************


NO' 1  001C12345678.  VALOR 0

1 2 3
4 5 6
7 8 .

001B12345.786
001D1234567.8

NO' 2  001B12345.786  VALOR 1

1 2 3
4 5 .
7 8 6

002B12.453786
002D1234.5786

NO' 3  001D1234567.8  VALOR 1
```

```
1 2 3
4 5 6
7 . 8

003B1234.6758
003D123456.78

NO' 4  002B12.453786  VALOR 2

1 2 .
4 5 3
7 8 6

004D1.2453786

NO' 5  002D1234.5786  VALOR 2

1 2 3
4 . 5
7 8 6

005B1.3425786
005A1234857.6
005D123.45786

NO' 6  003B1234.6758  VALOR 2

1 2 3
4 . 6
7 5 8

006B1.3426758
006E12346.758
006D123.46758

NO' 7  003D123456.78  VALOR 2

1 2 3
4 5 6
. 7 8

007B123.56478

NO' 8  004D1.2453786  VALOR 3

1 . 2
4 5 3
7 8 6

008A1524.3786
008D.12453786

NO' 9  005B1.3425786  VALOR 3
```



```
1 . 3
4 5 5
7 8 6

009E13.425786
009D.13425786

NO' 10  005A1234857.6  VALOR 3

1 2 3
4 8 5
7 . 6

010E12348576.
010D123485.76

NO' 11  005D123.45786  VALOR 3

1 2 3
. 4 5
7 8 6

011B.23145786
011A123745.86

NO' 12  006B1.3426758  VALOR 3

1 . 3
4 2 6
7 5 8

012E13.426758
012D.13426758

NO' 13  006E12346.758  VALOR 3

1 2 3
4 6 .
7 5 8

013B12.463758
013A12346875.

NO' 14  006D123.46758  VALOR 3

1 2 3
. 4 6
7 5 8

014B.23146758
014A123746.58

NO' 15  007B123.56478  VALOR 3
```

```
1 2 3
. 5 6
4 7 8

015B.23156478
015E1235.6478

NO' 16  008A1524.3786  VALOR 4

1 5 2
4 . 3
7 8 6

016E15243.786
016A1524837.5
016D152.43786

NO' 17  008D.12453786  VALOR 4

. 1 2
4 5 3
7 8 6

017A412.53786

NO' 18  009E13.425786  VALOR 4

1 3 .
4 2 5
7 8 6

018A13542.786

NO' 19  009D.13425786  VALOR 4

. 1 3
4 2 5
7 8 6

019A413.25786

NO' 20  010E12348576.  VALOR 4

1 2 3
4 8 5
7 6 .

020B12348.765

NO' 21  010D123485.76  VALOR 4

1 2 3
4 8 5
. 7 6
```



```
021B123.85476

NO' 22  011B.23145786  VALOR 4

. 2 3
1 4 5
7 8 6

022E2.3145786

NO' 23  011A123745.86  VALOR 4

1 2 3
7 4 5
. 8 6

023E1237458.6

NO' 24  012E13.426758  VALOR 4

1 3 .
4 2 6
7 5 8

024A13642.758

NO' 25  012D.13426758  VALOR 4

. 1 3
4 2 6
7 5 8

025A413.26758

NO' 26  013B12.463758  VALOR 4

1 2 .
4 6 3
7 5 8

026D1.2463758

NO' 27  013A12346875.  VALOR 4

1 2 3
4 6 8
7 5 .

027D1234687.5

NO' 28  014B.23146758  VALOR 4

. 2 3
```

1 4 6
7 5 8

028E2.3146758

NO' 29 014A123746.58 VALOR 4

1 2 3
7 4 6
. 5 8

029E1237465.8

NO' 30 015B.23156478 VALOR 4

. 2 3
1 5 6
4 7 8

030E2.3156478

NO' 31 015E1235.6478 VALOR 4

1 2 3
5 . 6
4 7 8

031B1.3526478
031E12356.478
031A1235764.8

NO' 32 016E15243.786 VALOR 5

1 5 2
4 3 .
7 8 6

032B15.432786
032A15243678.

NO' 33 016A1524837.6 VALOR 5

1 5 2
4 8 3
7 . 6

033E15246376.
033D152463.76

NO' 34 016D152.43786 VALOR 5

1 5 2
. 4 3

7 8 6

034B.52143786
034A152743.86

NO' 35 017A412.53786 VALOR 5

4 1 2
. 5 3
7 8 6

035E4125.3786
035A412753.86

NO' 36 018A13542.786 VALOR 5

1 3 5
4 2 .
7 8 6

036A13542678.
036D1354.2786

NO' 37 019A413.25786 VALOR 5

4 1 3
. 2 5
7 8 6

037E4132.5786
037A413725.86

NO' 38 020B12348.765 VALOR 5

1 2 3
4 8 .
7 6 5

038B12.483765
038D1234.8765

NO' 39 021B123.85476 VALOR 5

1 2 3
. 8 5
4 7 6

039B.23185476
039E1238.5476

NO' 40 022E2.3145786 VALOR 5

2 . 3

```
1 4 5
7 8 6

040E23.145786
040A2431.5786

NO' 41  023E1237458.6  VALOR 5

1 2 3
7 4 5
8 . 6

041B1237.5845
041E12374586.

NO' 42  024A13642.758  VALOR 5

1 3 6
4 2 .
7 5 8

042A13642875.
042D1364.2758

NO' 43  025A413.26758  VALOR 5

4 1 3
. 2 6
7 5 8

043E4132.6758
043A413726.58

NO' 44  026D1.2463758  VALOR 5

1 . 2
4 6 3
7 5 8

044A1624.3758
044D.12463758

NO' 45  027D1234687.5  VALOR 5

1 2 3
4 6 8
7 . 5

045B1234.8765
045D123468.75

NO' 46  028E2.3146758  VALOR 5
```



```
2 . 3
1 4 6
7 5 8

046E23.146758
046A2431.6758

NO' 47  029E1237465.8  VALOR 5

1 2 3
7 4 6
5 . 8

047B1237.6548
047E12374658.

*******  EIS A SOLUCAO : *******

  D --> 1234567.8
  B --> 1234.6758
  D --> 123.46758
  A --> 123746.58
  F --> 1237465.8
  E --> 12374658.

EM 6 ETAPAS

**********************************

LISTA ABERTA  ---> 36 NOS

LISTA FECHADA ---> 47 NOS

**********************************
```

6.3.2. O percurso de profundidade primeiro

Vejamos primeiramente um caso simples e, depois, o caso do exemplo anterior. Tal como acontece com o método que utiliza o percurso de largura primeiro, este também não permite a obtenção rápida de uma solução.

Para obter o percurso de profundidade primeiro, devemos incluir na listagem a seguinte «versão» da linha 1400:

```
1400 LET P5 = (D(N1) - 5000) - 1: RETURN
```

Observemos agora o primeiro exemplo:

```
**********************************
              PUZZLE 8
**********************************
PUZZLE DE PARTIDA (S)> 123.46758

1 2 3
. 4 6
7 5 8

PUZZLE DE CHEGADA (G)> 23.146758

2 3 .
1 4 6
7 5 8

**********************************


NO' 1  001C123.46758  VALOR 0

1 2 3
. 4 6
7 5 8

001B.23146758
001E1234.6758
001A123746.58

NO' 2  001B.23146758  VALOR -1

. 2 3
1 4 6
7 5 8

002E2.3146758

NO' 5  002E2.3146758  VALOR -2

2 . 3
1 4 6
7 5 8

005E23.146758
005A2431.6758

*******  EIS A SOLUCAO : *******
```



```
B --> .23146758
E --> 2.3146758
E --> 23.146758

EM 3 ETAPAS

**********************************
LISTA ABERTA  ---> 4 NOS
LISTA FECHADA ---> 3 NOS
**********************************
```

Vejamos agora a forma como este percurso trata o caso (mais complicado) do 2.º exemplo do parágrafo anterior:

```
**********************************
              PUZZLE 8
**********************************
PUZZLE DE PARTIDA (S)> 12345678.

1 2 3
4 5 6
7 8 .

PUZZLE DE CHEGADA (G)> 12374658.

1 2 3
7 4 6
5 8 .

**********************************


NO' 1  001C12345678.  VALOR 0

1 2 3
4 5 6
7 8 .

001B12345.786
001D1234567.8

NO' 2  001B12345.786  VALOR -1
```

```
1 2 3
4 5 .
7 8 6

002B12.453786
002D1234.5786

NO' 4  002B12.453786  VALOR -2

1 2 .
4 5 3
7 8 6

004D1.2453786

NO' 6  004D1.2453786  VALOR -3

1 . 2
4 5 3
7 8 6

006A1524.3786
006D.12453786

NO' 7  006A1524.3786  VALOR -4

1 5 2
4 . 3
7 8 6

007E15243.786
007A1524837.6
007D152.43786

NO' 9  007E15243.786  VALOR -5

1 5 2
4 3 .
7 8 6

009B15.432786
009A15243678.

NO' 12  009B15.432786  VALOR -6

1 5 .
4 3 2
7 8 6

012D1.5432786

NO' 14  012D1.5432786  VALOR -7
```



```
1 . 5
4 3 2
7 8 6

014A1354.2786
014D.15432786

NO' 15  014A1354.2786  VALOR -8

1 3 5
4 . 2
7 8 6

015E13542.786
015A1354827.6
015D135.42786

NO' 17  015E13542.786  VALOR -9

1 3 5
4 2 .
7 8 6

017B13.425786
017A13542678.

NO' 20  017B13.425786  VALOR -10

1 3 .
4 2 5
7 8 6

020D1.3425786

NO' 22  020D1.3425786  VALOR -11

1 . 3
4 2 5
7 8 6

022A1234.5786
022D.13425786

NO' 23  022A1234.5786  VALOR -12

1 2 3
4 . 5
7 8 6

023E12345.786
023A1234857.6
023D123.45786

NO' 25  023E12345.786  VALOR -13
```

```
1 2 3
4 5 .
7 8 6

025B12.453786

NO' 28  025B12.453786  VALOR -14

1 2 .
4 5 3
7 8 6

028D1.2453786

NO' 29  028D1.2453786  VALOR -15

1 . 2
4 5 3
7 8 6

029A1524.3786
029D.12453786

NO' 30  029A1524.3786  VALOR -16

1 5 2
4 . 3
7 8 6

030E15243.786
030A1524837.6
030D152.43786

NO' 32  030E15243.786  VALOR -17

1 5 2
4 3 .
7 8 6

032B15.432786
032A15243678.

NO' 35  032B15.432786  VALOR -18

1 5 .
4 3 2
7 8 6

035D1.5432786

NO' 37  035D1.5432785  VALOR -19

1 . 5
```



```
4 3 2
7 8 6

037A1354.2786
037D.15432786

NO' 38  037A1354.2786  VALOR -20

1 3 5
4 . 2
7 8 6

038E13542.786
038A1354827.6
038D135.42786

NO' 40  038E13542.786  VALOR -21

1 3 5
4 2 .
7 8 6

040B13.425786
040A13542678.

NO' 43  040B13.425786  VALOR -22

1 3 .
4 2 5
7 8 6

043D1.3425786

NO' 45  043D1.3425786  VALOR -23

1 . 3
4 2 5
7 8 6

045A1234.5786
045D.13425786

NO' 46  045A1234.5786  VALOR -24

1 2 3
4 . 5
7 8 6

046E12345.786
046A1234857.6
046D123.45786

NO' 48  046E12345.786  VALOR -25
```

```
1 2 3
4 5 .
7 8 6

048B12.453786

NO' 51  048B12.453786  VALOR -26

1 2 .
4 5 3
7 8 6

051D1.2453786

NO' 52  051D1.2453786  VALOR -27

1 . 2
4 5 3
7 8 6

052A1524.3786
052D.12453786

NO' 53  052A1524.3786  VALOR -28

1 5 2
4 . 3
7 8 6

053E15243.786
053A1524837.6
053D152.43786

NO' 55  053E15243.786  VALOR -29

1 5 2
4 3 .
7 8 6

055B15.432786
055A15243678.

NO' 58  055B15.432786  VALOR -30

1 5 .
4 3 2
7 8 6

058D1.5432786

NO' 60  058D1.5432786  VALOR -31

1 . 5
```



```
4 3 2
7 8 6

060A1354.2786
060D.15432786

NO' 61  060A1354.2786  VALOR -32

1 3 5
4 . 2
7 8 6

061E13542.786
061A1354827.6
061D135.42786

NO' 63  061E13542.786  VALOR -33

1 3 5
4 2 .
7 8 6

063B13.425786
063A13542678.

NO' 66  063B13.425786  VALOR -34

1 3 .
4 2 5
7 8 6

066D1.3425786

NO' 68  066D1.3425786  VALOR -35

1 . 3
4 2 5
7 8 6

068A1234.5786
068D.13425786

NO' 69  068A1234.5786  VALOR -36

1 2 3
4 . 5
7 8 6

069E12345.786
069A1234857.6
069D123.45786

NO' 71  069E12345.786  VALOR -37
```

```
1 2 3
4 5 .
7 8 6

071B12.453786

NO' 74  071B12.453786  VALOR -38

1 2 .
4 5 3
7 8 6

074D1.2453786

NO' 75  074D1.24E3786  VALOR -39

1 . 2
4 5 3
7 8 6

075A1524.3786
075D.12453786

NO' 76  075A1524.3786  VALOR -40

1 5 2
4 . 3
7 8 6

076E15243.786
076A1524837.6
076D152.43786

NO' 78  076E15243.786  VALOR -41

1 5 2
4 3 .
7 8 6

078B15.432786
078A15243678.

NO' 81  078B15.432786  VALOR -42

1 5 .
4 3 2
7 8 6

081D1.5432786

NO' 83  081D1.5432786  VALOR -43

1 . 5
```



```
4 3 2
7 8 6

083A1354.2786
083D.15432786

NO' 84  083A1354.2786  VALOR -44

1 3 5
4 . 2
7 8 6

084E13542.786
084A1354827.6
084D135.42786

NO' 86  084E13542.786  VALOR -45

1 3 5
4 2 .
7 8 6

086B13.425786
086A13542678.

NO' 89  086B13.425786  VALOR -46

1 3 .
4 2 5
7 8 6

089D1.3425786

NO' 91  089D1.3425786  VALOR -47

1 . 3
4 2 5
7 8 6

091A1234.5786
091D.13425786

NO' 92  091A1234.5786  VALOR -48

1 2 3
4 . 5
7 8 6

092E12345.786
092A1234857.6
092D123.45786

NO' 94  092E12345.786  VALOR -49
```

```
1 2 3
4 5 .
7 8 6

094B12.453786

NO' 97  094B12.453786  VALOR -50

1 2 .
4 5 3
7 8 6

097D1.2453786

NO' 98  097D1.2453786  VALOR -51

1 . 2
4 5 3
7 8 6

098A1524.3786
098D.12453786

**********************************

NAO HA' SOLUCOES
LISTA ABERTA  ---> 47 NOS

LISTA FECHADA ---> 52 NOS

**********************************
```

6.3.3. Algoritmo «out-of-place»

Como o seu nome indica (*out of place* significa «fora de sítio» ou «mal colocado»), esta função de avaliação calcula o número de casas mal colocadas em relação ao estado final. O método empregue é um método heurístico, e não combinatório como os anteriores. Para que o nosso programa o utilize, devemos introduzir as linhas seguintes:

```
1400>FOR I=1 TO 3: FOR J=1 TO 3:
 LET K=3*(I-1)+J
1410 LET F(I,J)=CODE G$(K): NEXT
 J: NEXT I
```



```
1420 LET P5=0
1430 FOR I=1 TO 3: FOR J=1 TO 3:
 LET K=3*(I-1)+J
1440 LET Q=CODE ((A$(A)(LEN A$(A
)-8 TO ))(K))
1450 IF Q=46 THEN GO TO 1470
1460 IF Q<>F(I,J) THEN LET P5=P5
+1
1470 NEXT J: NEXT I: RETURN
```

Acrescentar a seguinte linha :

```
  14 DIM F(3,3)
```

Vejamos agora como este método trata o caso que o do percurso de profundidade primeiro não conseguiu resolver:

```
**********************************
          PUZZLE 8
**********************************

PUZZLE DE PARTIDA (S)> 12345678.

1 2 3
4 5 6
7 8 .

PUZZLE DE CHEGADA (G)> 12374658.

1 2 3
7 4 6
5 8 .

**********************************


NO' 1  001012345678.  VALOR 0

1 2 3
4 5 6
7 8 .

001B12345.786
001D1234567.8
```

```
NO' 2  001B12345.786  VALOR 4

1 2 3
4 5 .
7 8 6

002B12.453786
002D1234.5786

NO' 3  001D1234567.8  VALOR 4

1 2 3
4 5 6
7 . 8

003B1234.6758
003D123456.78

NO' 5  002D1234.5786  VALOR 4

1 2 3
4 . 5
7 8 6

005B1.3425786
005A1234857.6
005D123.45786

NO' 10  005D123.45786  VALOR 3

1 2 3
. 4 5
7 8 6

010B.23145786
010A123745.86

NO' 12  010A123745.86  VALOR 2

1 2 3
7 4 5
. 8 6

012E1237458.6

NO' 13  012E1237458.6  VALOR 3

1 2 3
7 4 5
8 . 6

013B1237.5846
013E12374586.
```



```
NO' 15  013E12374586.  VALOR 3

1 2 3
7 4 5
8 6 .

015B12374.865

NO' 16  015B12374.865  VALOR 3

1 2 3
7 4 .
8 6 5

016B12.743865
016D1237.4865

NO' 6  003B1234.6758  VALOR 4

1 2 3
4 . 6
7 5 8

006B1.3426758
006E12346.758
006D123.46758

NO' 21  006D123.46758  VALOR 3

1 2 3
. 4 6
7 5 8

021B.23146758
021A123746.58

NO' 23  021A123746.58  VALOR 2

1 2 3
7 4 6
. 5 8

023E1237465.8

NO' 24  023E1237465.8  VALOR 1

1 2 3
7 4 6
5 . 8

024B1237.6548
024E12374658.
```

```
*******  EIS A SOLUCAO : *******

  D --> 1234567.8
  B --> 1234.67758
  D --> 123.467758
  A --> 123746.58
  E --> 1237465.8
  E --> 12374658.

EM 6 ETAPAS

*********************************

LISTA ABERTA  ---> 13 NOS

LISTA FECHADA ---> 13 NOS

*********************************
```

Como vemos, o problema que não conseguíamos resolver pelo método 2 (percurso de profundidade primeiro) e que o método 1 (percurso de largura primeiro) resolveu em 6 etapas, laboriosamente percorrendo 83 nós[1], é resolvido aqui em 6 etapas mas percorrendo apenas 26 nós[2].

6.3.4. Algoritmo dito «de distância mínima»

Como o seu nome indica, esta função de avaliação implica, de certa forma, o cálculo da «distância» entre a posição de uma casa e a que essa casa deveria ter na configuração final (estado G). Foi dada uma breve explicação desse cálculo no final do parágrafo 6.2.

Para pôr em acção este algoritmo no nosso programa, temos de substituir as linhas da rotina de cálculo do algoritmo precedente pelas seguintes:

```
1400>FOR I=1 TO 3: FOR J=1 TO 3:
 LET K=3*(I-1)+J
1410 LET F(I,J)=CODE G$(K): NEXT
 J: NEXT I
1420 LET P5=0
1430 FOR I=1 TO 3: FOR J=1 TO 3:
 LET K=3*(I-1)+J
1440 LET Q=CODE ((A$(A)(LEN A$(A
)-8 TO ))(K))
1450 IF Q=46 THEN GO TO 1500
1460 FOR M=1 TO 3: FOR N=1 TO 3
1470 IF Q<>F(M,N) THEN NEXT N: N
EXT M
1480 IF M=4 OR N=4 THEN GO TO 15
00
1490 LET P5=P5+ABS (M-I)+ABS (N-
J)
1500 NEXT J: NEXT I: RETURN
```

Acrescentar, também neste caso, a linha:

```
  14 DIM F(3,3)
```

Vejamos agora como este algoritmo resolve o caso do exemplo anterior:

```
*********************************
            PUZZLE 6
*********************************

PUZZLE DE PARTIDA (S)> 12345678.

1 2 3
4 5 6
7 8 .

PUZZLE DE CHEGADA (G)> 12374658.

1 2 3
7 4 6
5 8 .

*********************************

NO' 1  001C12345678.  VALOR 0

1 2 3
4 5 6
7 8 .

001B12345.786
001D1234567.8
```

[1] 36 da lista aberta e 47 da lista fechada. *(N. do T.)*

[2] 13 da lista aberta e 13 da fechada. *(N. do T)*

```
NO' 2  001B12345.786  VALOR 5

1 2 3
4 5 .
7 8 6

002B12.453786
002D1234.5786

NO' 3  001D1234567.8  VALOR 5

1 2 3
4 5 6
7 . 8

003B1234.6758
003D123456.78

NO' 6  003B1234.6758  VALOR 4

1 2 3
4 . 6
7 5 8

006B1.3426758
006E12346.758
006D123.46758

NO' 10  006D123.46758  VALOR 3

1 2 3
. 4 6
7 5 8

010B.23146758
010A123746.58

NO' 12  010A123746.58  VALOR 2

1 2 3
7 4 6
. 5 8

012E1237465.8

NO' 13  012E1237465.8  VALOR 1

1 2 3
7 4 6
5 . 8

013B1237.6548
013E12374658.
```



```
*******  EIS A SOLUCAO :  *******

  D --> 1234567.8
  B --> 1234.6758
  D --> 123.46758
  A --> 123746.58
  E --> 1237465.8
  E --> 12374658.

EM 6 ETAPAS

*********************************

LISTA ABERTA  ---> 8 NOS
LISTA FECHADA ---> 7 NOS

*********************************
```

Como no caso do algoritmo precedente, o problema posto é resolvido em 6 etapas, mas com a principal diferença de aqui apenas serem gerados 15 nós na árvore, enquanto que o caso anterior totalizava 26 nós.

6.3.5. Conclusão

Em conclusão, verifica-se que os métodos puramente combinatórios tais como os de pesquisa pura em largura primeiro ou profundidade primeiro dão, em geral, rendimentos bastante fracos.

Para este tipo de problema, onde o número de «estados» possíveis (neste caso, as configurações possíveis do *puzzle*) é grande, é perfeitamente ilusório querermos contentar-nos com a utilização destes métodos.

Melhora-se notavelmente o rendimento com a utilização de heurísticas, o que foi feito nos dois últimos métodos apresentados; obtêm-se geralmente soluções com maior brevidade mas sucede por vezes não se chegar a solução alguma. De notar que

os métodos combinatórios chegam geralmente sempre a uma solução, caso esta exista, embora possam levar tempos de cálculo perfeitamente incríveis. A razão disto reside no facto de estes métodos pesquisarem todos os estados possíveis, desenvolvendo exaustivamente a árvore; só que esses estados possíveis totalizam, neste caso, 362 880...

Eis, por exemplo, um caso em que o programa dificilmente chega a encontrar rapidamente um caminho que lhe permita ir de uma configuração de partida até ao objectivo, mesmo empregando heurísticas: é o caso de o *puzzle* de partida e o *puzzle* de chegada serem idênticos.

Para poder lidar com casos particulares como este, o programa tem de ser dotado de alguns «requintes» suplementares. Uma forma simples de proceder consiste em acrescentar ao início do programa uma linha em BASIC que teste a igualdade de S$ e G$.

6.3.6. Notas e versão para «hard-copy»

Como já mencionámos, as versões originais dos programas deste livro foram escritas e codificadas em BASIC Microsoft. A sua adaptação para o BASIC Sinclair do *Spectrum* implicou forçosamente que fossem introduzidas alterações, já que o BASIC Sinclair não é das versões mais próximas do BASIC *standard*. Assim, quem quiser adaptar este programa a um micro que utilize um BASIC próximo do Microsoft terá de alterar várias instruções. Não deverá haver qualquer problema com quadros (*arrays*), cadeias (*strings*) ou mesmo variáveis, mas as instruções de segmentação de cadeias de caracteres (*string slicing*) serão quase certamente diferentes. Assim, listam-se seguidamente as linhas contendo essas instruções, recodificadas em BASIC Microsoft:

```
  60 FOR B=1 TO 3: FOR C=1 TO 3:
 PRINT MID$ ( RIGHT$ (D$(N1),9),
3*(B-1)+C,1);" ";: NEXT C: PRINT
```



```
  90 H$=MID$ (D$(N1),5,9): J$=MI
D$ (D$(N1),4,1)

 140 H$=MID$ (A$(A),5,9): GOSUB
700

 730 K$(I,J)=MID$ (H$,K,1): X$(I
,J)=MID$ (H$,K,1): NEXT J: NEXT
I

 910 P$=RIGHT$ (A$(A),9): IF G$=
P$ THEN F$="YES": D8=D4+1

1130 O$=MID$ (D$(I),5,9)

1205 FOR I=1 TO 5: IF J$=MID$ (B
$,I,1) THEN 1220

1220 Q$=MID$ (Y$,I,1)

1255 IF MID$ (B$,P1,1)=Q$ THEN 1
290

1285 P9=P9+1: GOSUB 600: G1=G1+1
: A$(P9)=Z$+MID$ (B$,P1,1)+I$

1320 T$=MID$ (D$(D8),4,1): IF T$
="C" THEN 1340

1330 P5=P5+1: U$(P5)=T$: W$(P5)=
RIGHT$ (D$(D8),9)

1340 V$=LEFT$ (D$(D8),3): D8=VAL
 (V$): IF T$<>"C" THEN 1320


NAS ROTINAS DOS ALGORITMOS "OUT
OF PLACE" E "DE DISTANCIA MINI-
MA", TEMOS AINDA:


1410 F(I,J)=ASC (MID$ (G$,K,1)):
 NEXT J: NEXT I

1440 Q=ASC (MID$ (RIGHT$ (A$(A),
9),K,1))
```

Deve igualmente eliminar-se a linha 45 e a totalidade da declaração AND contida entre parêntesis (ou substituí-las por outras equivalentes na versão de BASIC utilizada).

Voltando ao *Spectrum*, quem dispuser de uma impressora pode utilizar a versão do programa listada a seguir, a qual permite obter uma *hard-copy* do *output* para uma melhor análise, sendo idêntico o *output* para o *écran* ao da primeira versão.

```
   5 REM
**********************************
            PUZZLE 8
**********************************

   6 REM
**********************************
      VERSAO C/HARD COPY
**********************************

  10 DIM A$(30,13): DIM U$(30):
DIM D$(100,13): DIM D(100): DIM
W$(30,9): DIM K$(3,3): DIM X$(3,
3): DIM X(4): DIM Y(4)
  14 DIM F(3,3)
  15 PRINT : PRINT "****** DESEJ
A HARD-COPY ? ******      (VIA
IMPRESSORA)": INPUT "SIM --> 1
NAO --> 0    ";COPY: CLS : IF CO
PY<>0 AND COPY<>1 THEN GO TO 15
  20 GO SUB 500
  30 IF E$="YES" THEN GO TO 300
  40 GO SUB 1000: IF D(N1)>=5999
 THEN GO TO 300
  45 POKE 23692,0
  50 PRINT : PRINT "NO' "; FLASH
 1;N1; FLASH 0;"  ";D$(N1);"  VA
LOR ";D(N1): PRINT
  55 IF COPY=1 THEN LPRINT : LPR
INT "NO' ";N1;"  ";D$(N1);"  VAL
OR ";D(N1): LPRINT
  60 FOR B=1 TO 3: FOR C=1 TO 3:
 PRINT D$(N1)(LEN D$(N1)-8 TO )(
3*(B-1)+C);" ";: NEXT C: PRINT
  70 NEXT B: PRINT
  75 IF COPY=1 THEN FOR B=1 TO 3
: FOR C=1 TO 3: LPRINT D$(N1)(LE
N D$(N1)-8 TO )(3*(B-1)+C);" ";:
 NEXT C: LPRINT : NEXT B: LPRINT

  80 LET D5=D5+1: LET D(N1)=5000
+D(N1)
  90 LET H$=D$(N1)(5 TO 13): LET
 J$=D$(N1)(4)
 100 GO SUB 700: GO SUB 1200
 110 IF G1<=0 THEN GO TO 30
```



```
 120 LET F$="NO": FOR A=1 TO G1:
 GO SUB 900
 130 IF F$="YES" THEN LET E$="YE
S"
 140 LET H$=A$(A)(5 TO 13): GO S
UB 700
 150 PRINT A$(A)
 152 IF COPY=1 THEN LPRINT A$(A)
 160 GO SUB 1400
 170 IF D4>=99 THEN GO TO 300
 180 LET D4=D4+1: LET D$(D4)=A$(
A): LET D(D4)=P5: NEXT A
 190 GO TO 30
 300 IF F$="YES" THEN GO SUB 130
0: GO TO 320
 310 PRINT : PRINT "*************
*********************": PRINT : P
RINT "NAO HA' SOLUCOES"
 315 IF COPY=1 THEN LPRINT : LPR
INT "*****************************
*****": LPRINT : LPRINT "NAO HA'
SOLUCOES"
 320 PRINT : PRINT "LISTA ABERTA
  ---> ";D4-D5;" NOS": PRINT : P
RINT "LISTA FECHADA ---> ";D5;"
NOS"
 325 IF COPY=1 THEN LPRINT "LIST
A ABERTA  ---> ";D4-D5;" NOS": L
PRINT : LPRINT "LISTA FECHADA --
-> ";D5;" NOS"
 330 PRINT : PRINT "*************
*********************"
 335 IF COPY=1 THEN LPRINT : LPR
INT "*****************************
*****"////
 340 STOP
 400 REM
**********************************
     IMPRESSAO DE CABECALHOS
**********************************

 410 LPRINT ""*****************
***************          PUZZL
E 8          *****************
***************"
 420 LPRINT : LPRINT "PUZZLE DE
PARTIDA (S)> ";S$: LPRINT
 430 FOR M=0 TO 2: FOR N=1 TO 3
 440 LPRINT S$(3*M+N);" ";: NEXT
 N: LPRINT : NEXT M: LPRINT
 450 LPRINT : LPRINT "PUZZLE DE
CHEGADA (G)> ";G$: LPRINT
 470 FOR M=0 TO 2: FOR N=1 TO 3
```

```
 480 LPRINT G$(3*M+N);" ";: NEXT
 N: LPRINT : NEXT M: LPRINT
 490 LPRINT : LPRINT "**********
**********************": LPRINT
: LPRINT : RETURN
 500 REM
*********************************
            INICIALIZACAO
*********************************

 510 DATA -1,0,0,1,1,0,0,-1
 520 FOR I=1 TO 4: READ X(I),Y(I
): NEXT I
 530 PRINT : PRINT "*INTRODUZIR
O PUZZLE DE PARTIDA*"";TAB 11;"
(PUZZLE S)"""DEVE-O INTRODUZIR
 COMO UMA UNICACADEIA DE CARACTE
RES,ISTO E',TO-DAS AS TRES ""LIN
HAS"" DO QUADRADONUMA UNICA LINH
A CONTINUA."
 540 INPUT "PUZZLE S --> ";S$: I
F LEN S$<>9 THEN GO TO 540
 545 CLS
 550 PRINT : PRINT "*INTRODUZIR
O PUZZLE DE CHEGADA*"";TAB 11;"
(PUZZLE G)"""DEVE-O INTRODUZIR
 COMO UMA UNICACADEIA DE CARACTE
RES,ISTO E',TO-DAS AS TRES ""LIN
HAS"" DO QUADRADONUMA UNICA LINH
A CONTINUA."
 560 INPUT "PUZZLE G --> ";G$: I
F LEN G$<>9 THEN GO TO 560
 565 CLS : IF COPY=1 THEN GO SUB
 400
 570 LET Y$="ADBEC": LET B$="BEA
DC": LET J$="C": LET D(1)=0: LET
 D4=1: LET N1=1: GO SUB 600: LET
 D$(1)=Z$+"C"+S$
 580 LET D5=0: LET E$="NO": LET
F$="NO": RETURN
 600 REM
*********************************
  CONVERSAO DE N1 EM CADEIA DE
            CARACTERES
*********************************

 610 LET Z$=STR$ N1: LET Z=LEN Z
$: LET Z=3-Z
 620 IF Z=0 THEN RETURN
 630 FOR I=1 TO Z: LET Z$="0"+Z$
: NEXT I: RETURN
 700 REM
```



```
*********************************
    TRANSFORMAR H$ EM QUADRO
*********************************

 710 FOR I=1 TO 3: FOR J=1 TO 3
 720 LET K=3*(I-1)+J
 730 LET K$(I,J)=H$(K): LET X$(I
,J)=H$(K): NEXT J: NEXT I
 740 RETURN
 800 REM
*********************************
    TRANSFORMAR UM QUADRO EM
     CADEIA DE CARACTERES
*********************************

 810 LET I$="": FOR I=1 TO 3: FO
R J=1 TO 3
 820 LET I$=I$+X$(I,J): NEXT J:
NEXT I
 830 RETURN
 900 REM
*********************************
    VERIFICAR SE O OBJECTIVO
         FOI ALCANCADO
*********************************

 910 LET P$=A$(A)(LEN A$(A)-9+1
TO ): IF G$=P$ THEN LET F$="YES"
: LET D6=D4+1
 920 RETURN
1000 REM
*********************************
  ENCONTRAR O NO'A DESENVOLVER
*********************************

1010 LET N1=1: LET P1=5999: FOR
I=1 TO D4
1020 IF D(I)>=P1 THEN GO TO 1040
1030 LET P1=D(I): LET N1=I
1040 NEXT I: RETURN
1100 REM
*********************************
   VER SE A SOLUCAO JA'EXISTE
*********************************

1110 LET P$="NAO"
1120 FOR I=1 TO D4: IF D(I)<5000
 THEN GO TO 1150
1130 LET O$=D$(I)(5 TO 13)
1140 IF O$=I$ THEN LET P$="SIM":
 RETURN
1150 NEXT I
1160 IF P$="NAO" THEN RETURN
1200 REM
```

```
*********************************
  DESENVOLVER OS SUCESSORES
*********************************

1205 FOR I=1 TO 5: IF U$=B$(I) T
HEN GO TO 1220
1210 NEXT I
1220 LET O$=Y$(I)
1225 FOR X=1 TO 3: FOR Y=1 TO 3
1230 IF K$(X,Y)="," THEN GO TO 1
240
1235 NEXT Y: NEXT X
1240 LET G1=0
1245 LET P1=1: LET P9=0
1250 IF P1>4 THEN GO TO 1295
1255 IF B$(P1)=O$ THEN GO TO 129
0
1260 LET X2=X+X(P1): LET Y2=Y+Y(
P1): FOR I=1 TO 3: FOR J=1 TO 3:
 LET X$(I,J)=K$(I,J): NEXT J: NE
XT I
1265 IF X2<1 OR X2>3 THEN GO TO
1290
1270 IF Y2<1 OR Y2>3 THEN GO TO
1290
1275 LET X$(X,Y)=X$(X2,Y2): LET
X$(X2,Y2)=",": GO SUB 800: GO SU
B 1100
1280 IF P$="SIM" THEN GO TO 1290
1285 LET P9=P9+1: GO SUB 600: LE
T G1=G1+1: LET A$(P9)=Z$+B$(P1)+
I$
1290 LET P1=P1+1: GO TO 1250
1295 RETURN
1300 REM
*********************************
  REDESENVOLVER A SOLUCAO
*********************************

1310 LET P5=0
1320 LET T$=D$(D6)(4): IF T$="C"
 THEN GO TO 1340
1330 LET P5=P5+1: LET U$(P5)=T$:
 LET W$(P5)=D$(D6)(LEN D$(D6)-8
TO )
1340 LET V$=D$(D6)( TO 3): LET D
6=VAL V$: IF T$<>"C" THEN GO TO
1320
1350 PRINT : PRINT "*******  EIS
 A SOLUCAO : *******": PRINT
1360 FOR I=P5 TO 1 STEP -1: PRIN
T "   ";U$(I);" --> ";W$(I): NEXT
 I
```



```
1370 PRINT : PRINT "EM ";P5;" ET
APA";(+"S" AND P5<>1): PRINT : P
RINT
1375 PRINT "*********************
***********"
1376 IF COPY=1 THEN LPRINT : LPR
INT "*******  EIS A SOLUCAO : **
*****": LPRINT
1377 IF COPY=1 THEN FOR I=P5 TO
1 STEP -1: LPRINT "   ";U$(I);" -
--> ";W$(I): NEXT I
1378 IF COPY=1 THEN LPRINT : LPR
INT "EM ";P5;" ETAPA";(+"S" AND
P5<>1): LPRINT
1379 IF COPY=1 THEN LPRINT "****
*****************************"//
1380 RETURN
1400 REM     ATENCAO !!!
*********************************
--> INSERIR A PARTIR DESTA LINHA
    (INCLUSIVE!) A ROTINA (LISTA
    DA NO PARAGRAFO RESPECTIVO)
    REFERENTE AO PERCURSO OU AO
    ALGORITMO DESEJADO.
*********************************
```

Capítulo 7

O jogo do galo

Enquanto no capítulo precedente analisámos um jogo a um jogador, vamos neste debruçar-nos sobre os métodos classicamente utilizados para resolver o mesmo tipo de problema, mas no jogo a dois jogadores. Seleccionámos um jogo muito simples, certamente conhecido de todos: o jogo do galo, também chamado «três em linha» ou *tic-tac-toe*.

Tomemos um «tabuleiro» quadriculado de 3×3 casas. Um dos jogadores joga com «cruzes» (X). O outro com «bolas» (O). Para ganhar, é necessário alinhar 3 cruzes (ou 3 «bolas»), devendo cada jogador colocar um único símbolo de cada vez no tabuleiro.

O programa que propomos incorpora uma estratégia de jogo: utiliza um método clássico em inteligência artificial, conhecido por Min-Max.

7.1. O MÉTODO DO MIN-MAX

A finalidade consiste em encontrar jogadas a efectuar, que sejam positivas localmente. O método não pretende partir do início do jogo e encontrar a sequência de jogadas vencedoras que leva sistematicamente à vitória, mas decide, a cada dada etapa do jogo, qual a jogada mais interessante a tentar.

Procede-se da seguinte forma: desenvolve-se uma jogada (isto quer dizer que se constrói a árvore das possibilidades de jogo), efectua-se a seguir a escolha da melhor jogada a fazer, etc...

Para isso, é necessário dispor de uma função de avaliação estática, elaborar uma jogada em três níveis, depois avaliar, graças à função estática, cada uma das jogadas elaboradas a fim de avaliar o nó primitivo e de, portanto, poder decidir qual a melhor jogada a efectuar na etapa em questão de jogo.

Estabeleçamos algumas definições próprias do método. Distinguem-se os dois jogadores chamando a um o jogador Max e ao outro o jogador Min. Um nó diz-se do tipo Max se os seus sucessores constituem uma jogada efectuada pelo jogador Max. Este nó tem por valor o máximo dos valores dos nós sucessores, sendo a avaliação efectuada pela função estática.

Um nó dir-se-á de tipo Min se os seus sucessores constituírem uma jogada do jogador Min. Tem por valor o mínimo dos valores dos sucessores.

Como no capítulo precedente, o problema é identificado com um espaço de estados, sendo cada estado a situação do jogo num dado momento. Contam-se 9! (cerca de 300 000) estados possíveis! Felizmente, existem simetrias, ou estados simétricos.

A função f estática é construída da seguinte forma:

f = (Número de alinhamentos completos, ainda possíveis para MAX) − (Número de alinhamentos completos, ainda possíveis para MIN).

Um alinhamento pode ser uma linha, uma coluna ou ainda uma das duas diagonais do quadrado.

Exemplo:

	O	
	X	

f=6−4=2

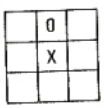
Com efeito, têm-se, para O, 4 alinhamentos possíveis:
— a 1.ª linha
— a 3.ª linha
— a 1.ª coluna
— a 3.ª coluna

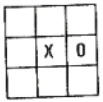
Para X, têm-se, neste caso, 6 alinhamentos possíveis:

— a 2.ª linha
— a 3.ª linha

— a 1.ª e a 3.ª coluna
— as duas diagonais

Falámos de simetrias eventuais para certas situações do jogo. Assim, por exemplo, as situações



e


são equivalentes, etc.

O algoritmo que é aqui programado incorpora uma sub-rotina encarregada de detectar as configurações equivalentes por simetria, o que tem por efeito restringir consideravelmente as dimensões da árvore desenvolvida.

Notemos, por fim, antes de comentarmos brevemente a listagem apresentada um pouco mais adiante, que a função de avaliação é de molde a permitir dar o mais possível uma ideia do avanço do jogo.

Por convenção, em teoria dos jogos, atribui-se a um nó Max um valor da função $f>\emptyset$ e a um nó Min um valor$\leq\emptyset$. Além disso, os valores de f próximos de $\emptyset$ não são favoráveis nem a Min nem a Max.

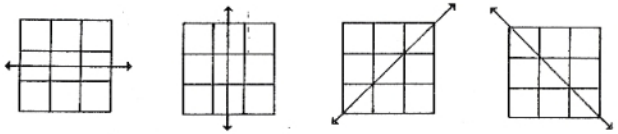
O jogador Max jogará o valor mais alto possível e Min terá todo o interesse em jogar o menos elevado.

7.2. PROGRAMAÇÃO DO JOGO

Passemos agora à programação propriamente dita do jogo. Programámo-lo da seguinte forma: o jogador Max é o computador e nós somos o jogador Min. A máquina terá portanto, para ganhar, de alinhar 3 X, enquanto que nós teremos de alinhar 3 O.

No início da execução, o programa pergunta-nos qual dos jogadores 1 e 2 joga em primeiro lugar. Devemos introduzir 1 para o jogador 1 (o computador) e 2 para o jogador 2.

Dissemos já que o programa tinha em conta as simetrias. Na verdade, ele só gera posições diferentes por simetria. Existem com efeito 4 eixos de simetria possíveis, a saber:



Para uma configuração do jogo dada, com um certo número de casas preenchidas quer com X, quer com O, o programa determina os eixos de simetria efectivos e toma-os em consideração aquando da geração das jogadas possíveis.

O programa funciona da seguinte forma:

— pede-nos, logo de início, para indicarmos o jogador que deve começar o jogo;

— analisa todas as jogadas possíveis, executáveis numa situação dada, tendo em conta simetrias eventuais. Deste modo, quando for a nossa vez de jogar, o programa apresenta-nos, preliminarmente, todas as jogadas possíveis na situação em questão e pede-nos para introduzirmos o número da jogada a efectuar que escolhermos;

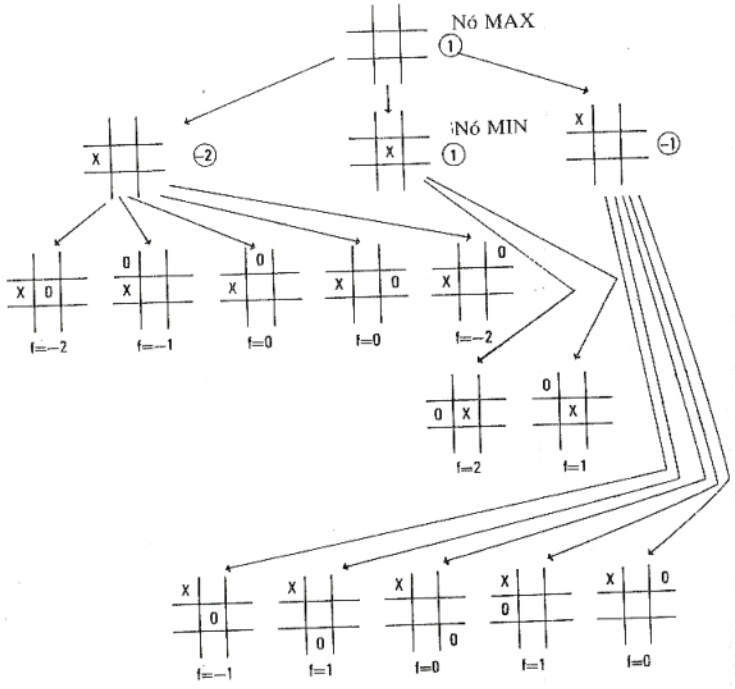
— os jogadores jogam, evidentemente, à vez.

Vejamos a estratégia de Max.

O nosso raciocínio baseia-se nesta figura logo no início do jogo: o nó de partida é um nó Max (pois os seus sucessores constituem uma jogada efectuada pelo jogador Max).

Os nós seguintes são nós Min, e assim por diante...

O método permite um raciocínio sobre dois níveis de nós sucessores:



Por desenvolvimento, obtém-se, pois, a figura anterior. Calcula-se a função f e dela se pode deduzir posteriormente o valor dos nós predecessores. O nó de partida vale, assim, 1. Max terá, portanto, interesse em fazer a seguinte jogada:

a qual corresponde ao valor f=1 da função de avaliação.

Se procurarmos compreender linha por linha a listagem apresentada, verificaremos que, quando 3(X) estão alinhados, a função f vale + ∞ (neste caso + 1000) e quando 3(O) estão alinhados a função vale − ∞ (− 1000). Em qualquer destes dois casos, em que a máquina ganhou (1.º caso) ou em que nós ganhámos (2.º caso), a partida está terminada. No caso de todas as casas ficarem preenchidas, é impressa no *écran* a mensagem «partida terminada», quer tenha sido o computador o vencedor, quer tenhamos sido nós ou não tenha havido vencedor.

7.3 LISTAGEM E COMENTÁRIOS

```
   5 REM
**********************************
          JOGO DO GALO
**********************************

   6 REM  INICIALIZACAO

  10 PRINT "**********************
**********          JOGO DO GAL
O                  **************
**********"""""    A FINALIDADE
 DO JOGO E:"""""---PARA A MAQUI
NA-->ALINHAR 3 X"""---PARA SI -
-------->ALINHAR 3 O": PRINT : PR
INT
  15 PRINT "====================
==========": PRINT
```

```
  20 PRINT "  CADA UM JOGA A VEZ
"/"- VOCE E/ O JOGADOR 2"/"- O
COMPUTADOR E/O JOGADOR 1"
  25 INPUT "QUAL O NUMERO DO JOG
ADOR"/"QUE COMECA (1 OU 2) ? ---
--)";FF
  30 IF FF=0 OR FF>2 THEN GO TO
25
  40 DIM S(4): DIM T(10): DIM F(
10): DIM M(3,3): DIM N(3,3): DIM
 U(3,3): DIM V(3,3)
  45 FOR I=1 TO 3: FOR J=1 TO 3:
 LET N(I,J)=0: NEXT J: NEXT I
  50 GO SUB 1200: GO TO 800
 490 REM
*********************************
     CALCULO DO MINIMO DE F
*********************************

 500 LET MI=1000: FOR V=1 TO VL-
1: IF F(V)<MI THEN LET MI=F(V)
 510 NEXT V
 520 RETURN
 540 REM
*********************************
     CALCULO DO MAXIMO DE T
*********************************

 550 LET MA=-1000: FOR V=1 TO Z:
 IF T(V)>MA THEN LET MA=T(V)
 560 NEXT V
 570 RETURN
 600 REM
*********************************
          JOGADA DE MAX
*********************************

 605 PRINT : PRINT : PRINT "* AG
ORA EU... A MINHA JOGADA E/:"/"/"
(DEIXE-ME PENSAR UM POUCO S,F,F)
": PRINT
 610 LET T$=C$
 615 FOR I=1 TO 3: FOR J=1 TO 3:
 LET N(I,J)=M(I,J): NEXT J: NEXT
 I
 620 LET L=LEN T$: FOR Z=0 TO (L
/9)-1: FOR I=1 TO 3: FOR J=1 TO
3
 625 LET M(I,J)=VAL (T$(Z*9+(I-1
)*3+J)): NEXT J: NEXT I
 630 GO SUB 900: LET IN=2: LET C
$="": GO SUB 1500: LET VL=1
 635 FOR P=1 TO 3: FOR Q=1 TO 3:
```



```
 IF M(P,Q)=0 AND V(P,Q)=0 THEN L
ET M(P,Q)=IN: GO SUB 1200: LET F
(VL)=F: LET VL=VL+1: GO SUB 1400
: LET M(P,Q)=0
 640 NEXT Q: NEXT P
 645 GO SUB 500: LET T(Z+1)=MI
 650 NEXT Z
 655 GO SUB 550: FOR L=1 TO Z: I
F T(L)<>MA THEN NEXT L
 660 LET Z=L-1: FOR I=1 TO 3: FO
R J=1 TO 3
 665 LET M(I,J)=VAL (T$(Z*9+(I-1
)*3+J))
 670 IF M(I,J)=1 THEN PRINT "X "
;
 675 IF M(I,J)=2 THEN PRINT "O "
;
 680 IF M(I,J)=0 THEN PRINT "- "
;
 685 NEXT J: PRINT : NEXT I: PRI
NT : RETURN
 700 REM
*********************************
          JOGADA DE MIN
*********************************

 705 PRINT : PRINT "E/A SUA VEZ
DE EFECTUAR UMA DAS JOGADAS SEGU
INTES: ": PRINT
 710 LET L=LEN C$: FOR Z=0 TO (L
/9)-1: PRINT TAB 9; FLASH 1;Z+1
 715 FOR I=1 TO 3: FOR J=1 TO 3:
 LET U(I,J)=VAL (C$(Z*9+(I-1)*3+
J))
 720 IF U(I,J)=1 THEN PRINT "X "
;
 725 IF U(I,J)=2 THEN PRINT "O "
;
 730 IF U(I,J)=0 THEN PRINT "- "
;
 735 NEXT J: PRINT : NEXT I: PRI
NT : NEXT Z
 740 PRINT "** QUAL A SUA JOGADA
 (<=";L/9;") ?": INPUT "JOGADA -
------>";I
 745 IF I=0 OR I>(L/9) THEN GO T
O 740
 747 PRINT : PRINT "ESCOLHEU A J
OGADA "; FLASH 1;I
 750 LET Z=I-1: FOR I=1 TO 3: FO
R J=1 TO 3: LET M(I,J)=VAL (C$(Z
*9+(I-1)*3+J)): NEXT J: NEXT I
```

```
 760 RETURN
 800 REM
*********************************
        ROTINA  PRINCIPAL
*********************************

 805 LET IA=0: LET IC=0: FOR I=1
 TO 3: FOR J=1 TO 3: IF M(I,J)=1
 THEN LET IA=IA+1
 810 IF M(I,J)=2 THEN LET IC=IC+
1
 815 NEXT J: NEXT I: LET IN=1: G
O SUB 900: LET C$="": IF FF=1 AN
D IC<IA THEN LET IN=2
 820 IF FF=2 AND IC<=IA THEN LET
 IN=2
 825 GO SUB 1500: FOR P=1 TO 3:
FOR Q=1 TO 3: IF M(P,Q)=0 AND V(
P,Q)=0 THEN LET M(P,Q)=IN: GO SU
B 1400: LET M(P,Q)=0
 830 NEXT Q: NEXT P
 835 IF IN=2 THEN GO SUB 700
 840 IF IN=1 THEN GO SUB 600
 845 FOR I=1 TO 3: FOR J=1 TO 3:
 IF M(I,J)=0 THEN GO TO 860
 850 NEXT J: NEXT I: PRINT : PRI
NT FLASH 1;"PARTIDA TERMINADA...
";

 855 STOP
 860 GO SUB 1200: IF F=-1000 THE
N PRINT : PRINT FLASH 1;"VOCE GA
NHOU !,,,": STOP
 865 LET II=1: GO SUB 1100: IF C
=3 THEN LET F=1000
 870 IF F=1000 THEN PRINT FLASH
1;"GANHEI EU !!!": STOP
 875 GO TO 800
 890 REM
*********************************
  CALCULO DOS EIXOS DE SIMETRIA
*********************************

 900 FOR I=1 TO 3: FOR J=1 TO 3:
 GO SUB 1300: IF M(I,J)<>M(II,JJ
) THEN LET S(1)=0: GO TO 910
 905 NEXT J: NEXT I: LET S(1)=1
 910 FOR I=1 TO 3: FOR J=1 TO 3:
 GO SUB 1320: IF M(I,J)<>M(II,JJ
) THEN LET S(2)=0: GO TO 920
 915 NEXT J: NEXT I: LET S(2)=1
 920 FOR I=1 TO 3: FOR J=1 TO 3:
 GO SUB 1335: IF M(I,J)<>M(II,JJ
```



```
) THEN LET S(3)=0: GO TO 930
 925 NEXT J: NEXT I: LET S(3)=1
 930 FOR I=1 TO 3: FOR J=1 TO 3:
 GO SUB 1340: IF M(I,J)<>M(II,JJ
) THEN LET S(4)=0: GO TO 940
 935 NEXT J: NEXT I: LET S(4)=1
 940 RETURN
1000 REM
*********************************
        CALCULO DE C
*********************************

1005 LET C=0
1010 FOR I=1 TO 3: FOR J=1 TO 3
1020 IF M(I,J)=II THEN GO TO 103
0
1025 NEXT J: LET C=C+1
1030 NEXT I
1035 FOR J=1 TO 3: FOR I=1 TO 3
1040 IF M(I,J)=II THEN GO TO 105
0
1045 NEXT I: LET C=C+1
1050 NEXT J
1055 FOR I=1 TO 3: IF M(I,I)=II
THEN GO TO 1060
1056 NEXT I: LET C=C+1
1060 FOR I=1 TO 3: IF M(4-I,I)=I
I THEN RETURN
1065 NEXT I: LET C=C+1: RETURN
1090 REM
*********************************
   SABER SE 3 SIMBOLOS IGUAIS
        ESTAO ALINHADOS
*********************************

1100 FOR I=1 TO 3: LET C=0: FOR
J=1 TO 3: IF M(I,J)<>II THEN GO
TO 1110
1105 LET C=C+1: NEXT J: IF C=3 T
HEN RETURN
1110 NEXT I
1115 FOR J=1 TO 3: LET C=0: FOR
I=1 TO 3: IF M(I,J)<>II THEN GO
TO 1120
1117 LET C=C+1: NEXT I: IF C=3 T
HEN RETURN
1120 NEXT J
1125 LET C=0: FOR I=1 TO 3: IF M
(I,I)<>II THEN GO TO 1130
1128 LET C=C+1: NEXT I: IF C=3 T
HEN RETURN
1130 LET C=0: FOR I=1 TO 3: IF M
(4-I,I)<>II THEN RETURN
```

```
1135 LET C=C+1: NEXT I: RETURN
1200 REM
*********************************
          CALCULO DE f
*********************************

1205 LET II=2: GO SUB 1000: LET
CA=C: LET II=1: GO SUB 1000: LET
 CI=C
1210 LET II=2: GO SUB 1100: IF C
=3 THEN LET F=-1000: RETURN
1215 LET F=CA-CI: RETURN
1300 REM
*********************************
    CALCULO DA CASA SIMETRICA
*********************************

1305 LET II=I: IF I=1 THEN LET I
I=3
1310 IF I=3 THEN LET II=1
1315 LET JJ=J: RETURN
1320 LET JJ=2: IF J=1 THEN LET J
J=3
1325 IF J=3 THEN LET JJ=1
1330 LET II=I: RETURN
1335 LET JJ=I: LET II=J: RETURN
1340 LET II=I: LET JJ=J: IF I+J=
4 THEN RETURN
1345 IF I+J<4 THEN GO TO 1360
1350 LET II=II-1: LET JJ=JJ-1: I
F II+JJ>=4 THEN GO TO 1350
1355 RETURN
1360 LET II=II+1: LET JJ=JJ+1: I
F II+JJ<=4 THEN GO TO 1360
1365 RETURN
1400 REM
*********************************
    LINEARIZACAO DE M(I,J)
*********************************

1405 FOR I=1 TO 3: FOR J=1 TO 3
1410 LET C$=C$+STR$ M(I,J): NEXT
 J: NEXT I
1420 RETURN
1500 REM
*********************************
       POSICAO INTERDITA ?
*********************************

1505 FOR I=1 TO 3: FOR J=1 TO 3:
 LET V(I,J)=0: NEXT J: NEXT I: F
OR K=1 TO 4: IF S(K)=0 THEN GO T
O 1550
```



```
1510 FOR I=1 TO 3: FOR J=1 TO 3:
 IF V(I,J)=1 THEN GO TO 1545
1515 IF K=1 THEN GO SUB 1300
1520 IF K=2 THEN GO SUB 1320
1525 IF K=3 THEN GO SUB 1335
1530 IF K=4 THEN GO SUB 1340
1535 IF I=II AND J=JJ THEN GO TO
 1545
1540 LET V(II,JJ)=1
1545 NEXT J: NEXT I
1550 NEXT K: RETURN
```

```
NOTA:SE DESEJARMOS QUE O PROGRA-
MA NAO PARE COM UM "Scroll?" DU-
RANTE A IMPRESSAO (EMBORA SEJA
POR VEZES UTIL),DEVEMOS ACRES-
CENTAR AS LINHAS SEGUINTES :

 667 POKE 23692,0

 717 POKE 23692,0
```

Analisemos brevemente a listagem:

- *Linhas 10 a 45:*

Inicialização, colocação a zero da matriz M (I,J) do *puzzle*. Introdução do número do jogador que começa o jogo.

- *Sub-rotina 500–520:*

Cálculo do mínimo de um vector F comportando VL–1 elementos.

- *Sub-rotina 550–570:*

Cálculo do máximo de um vector T comportando Z elementos.

- *Sub-rotina 1200–1215:*

Cálculo da função f, valor estático do *puzzle*. Note-se CA, que é o número de alinhamentos livres para o jogador Max, e CI, que é o número de alinhamentos livres para o jogador Min.

• *Sub-rotina 1000–1065:*

Cálculo de C, que é o número de alinhamentos livres para um jogador J. São examinadas:

— as linhas (1000–1030)
— as colunas (1035–1050)
— a 1.ª diagonal (1055–1056)
— a 2.ª diagonal (1060–1065)

• *Sub-rotina 1100–1135:*

O cálculo aqui efectuado destina-se a saber se três símbolos (ou piões) do mesmo jogador se encontram alinhados. Segue-se o mesmo procedimento adoptado na sub-rotina anterior, ou seja, são examinadas primeiro as linhas, depois as colunas e as diagonais.

• *Sub-rotina 900–940:*

Cálculo dos eixos de simetria do *puzzle* em qualquer configuração dada. O resultado é colocado no quadro S, podendo S(i) tomar o valor 0 ou o valor 1, variando i de 1 a 4.

• *Sub-rotina 1300–1365:*

Determinação da casa simétrica de uma casa dada, em relação a um eixo dado.

• *Sub-rotina 1400–1420:*

Colocação da matriz M(I,J) sob forma linearizada, representando esta matriz o estado do jogo.

• *Sub-rotina 1500–1550:*

Cálculo para determinar se uma posição é autorizada ou possível (caso ela ainda não exista) ou se é interdita (por já ter sido gerada sob a forma de uma posição equivalente).

• *Sub-rotina 700–760:*

Sub-rotina chamada quando é a vez de Min jogar. O programa gera e imprime no *écran* todas as soluções possíveis, ou seja, todas as jogadas possíveis para Min no momento, devendo este optar por uma.

• *Sub-rotina 600–685:*

Sub-rotina que efectua uma jogada de Max com todas as etapas de estratégia que descrevemos no parágrafo precedente. São gerados os sucessores de primeiro nível, depois os do segundo nível, que são avaliados, podendo seguidamente serem avaliados os do primeiro nível e escolher-se de entre eles o nó mais favorável.

• *Linhas 800–875:*

Parte principal do programa, encarregada de gerar o início, o fim da partida e de passar sucessivamente a vez aos dois jogadores. Esta parte, que poderíamos designar por «programa principal», efectua a chamada de todos os módulos ou sub-rotinas que descrevemos anteriormente.

Passemos agora a exemplos de execução.

7.4 EXEMPLOS

Vejamos um primeiro exemplo onde o jogador 2 (nós próprios, se fizermos executar o programa) joga em primeiro lugar:

```
**********************************
          JOGO DO GALO
**********************************

   A FINALIDADE DO JOGO E':

---PARA A MAQUINA-->ALINHAR 3 X
---PARA SI --------->ALINHAR 3 O

==================================
  CADA UM JOGA A VEZ

- VOCE E' O JOGADOR 2
- O COMPUTADOR E'O JOGADOR 1

QUAL O NUMERO DO JOGADOR
QUE COMECA (1 OU 2) ? ----->2
```

```
E'A SUA VEZ DE EFECTUAR UMA DAS
JOGADAS SEGUINTES:

          1

O - -
- - -
- - -

          2

- O -
- - -
- - -

          3

- - -
- O -
- - -

** QUAL A SUA JOGADA (<=3) ?

JOGADA ------->3

ESCOLHEU A JOGADA 3

* AGORA EU... A MINHA JOGADA E':

(DEIXE-ME PENSAR UM POUCO S,F,F)

X - -
- O -
- - -
```

```
E'A SUA VEZ DE EFECTUAR UMA DAS
JOGADAS SEGUINTES:

          1

X O -
- O -
- - -

          2

X - O
- O -
- - -

          3

X - -
- O O
- - -
```



```
          4

X - -
- O -
- - O

** QUAL A SUA JOGADA (<=4) ?

JOGADA ------->2

ESCOLHEU A JOGADA 2

* AGORA EU... A MINHA JOGADA E':

(DEIXE-ME PENSAR UM POUCO S,F,F)

X - O
- O -
X - -
```

```
E'A SUA VEZ DE EFECTUAR UMA DAS
JOGADAS SEGUINTES:

          1

X O O
- O -
X - -

          2

X - O
O O -
X - -

          3

X - O
- O O
X - -

          4

X - O
- O -
X O -

          5

X - O
- O -
X - O

** QUAL A SUA JOGADA (<=5) ?

JOGADA ------->3
```

```
ESCOLHEU A JOGADA 3

* AGORA EU... A MINHA JOGADA E': 
(DEIXE-ME PENSAR UM POUCO S,F,F)

X X O
- O O
X - -

E'A SUA VEZ DE EFECTUAR UMA DAS
JOGADAS SEGUINTES:

          1
X X O
O O O
X - -

          2
X X O
- O O
X O -

          3
X X O
- O O
X - O

** QUAL A SUA JOGADA (<=3) ?

JOGADA ------->1

ESCOLHEU A JOGADA 1

VOCE GANHOU !!!!
```

É agora a vez de a máquina iniciar o jogo. Reparemos que ela começa por colocar o seu «pião» (a cruz ou X) ao centro, como vimos no parágrafo 2.

```
**********************************
          JOGO DO GALO
**********************************

   A FINALIDADE DO JOGO E': 

---PARA A MAQUINA-->ALINHAR 3 X
---PARA SI -------->ALINHAR 3 O

==================================
  CADA UM JOGA A VEZ

- VOCE E' O JOGADOR 2
- O COMPUTADOR E'O JOGADOR 1

QUAL O NUMERO DO JOGADOR
QUE COMECA (1 OU 2) ? ----->1

* AGORA EU... A MINHA JOGADA E': 
(DEIXE-ME PENSAR UM POUCO S,F,F)

- - -
- X -
- - -

E'A SUA VEZ DE EFECTUAR UMA DAS
JOGADAS SEGUINTES:

          1
O - -
- X -
- - -

          2
- O -
- X -
- - -

** QUAL A SUA JOGADA (<=2) ?

JOGADA ------->1

ESCOLHEU A JOGADA 1

* AGORA EU... A MINHA JOGADA E': 
(DEIXE-ME PENSAR UM POUCO S,F,F)

O - X
- X -
- - -
```

```
E'A SUA VEZ DE EFECTUAR UMA DAS
JOGADAS SEGUINTES:

          1
O O X
- X -
- - -

          2
O - X
O X -
- - -

          3
O - X
- X O
- - -

          4
O - X
- X -
O - -

          5
O - X
- X -
- O -

          6
O - X
- X -
- - O

** QUAL A SUA JOGADA (<=6) ?

JOGADA ------->4

ESCOLHEU A JOGADA 4

* AGORA EU... A MINHA JOGADA E':

(DEIXE-ME PENSAR UM POUCO S,F,F)

O - X
X X -
O - -

E'A SUA VEZ DE EFECTUAR UMA DAS
JOGADAS SEGUINTES:
```



```
          1
O O X
X X -
O - -

          2
O - X
X X O
O - -

          3
O - X
X X -
O O -

          4
O - X
X X -
O - O

** QUAL A SUA JOGADA (<=4) ?

JOGADA ------->4

ESCOLHEU A JOGADA 4

* AGORA EU... A MINHA JOGADA E':

(DEIXE-ME PENSAR UM POUCO S,F,F)

O - X
X X -
O X O


E'A SUA VEZ DE EFECTUAR UMA DAS
JOGADAS SEGUINTES:

          1
O O X
X X -
O X O

** QUAL A SUA JOGADA (<=1) ?

JOGADA ------->1

ESCOLHEU A JOGADA 1

* AGORA EU... A MINHA JOGADA E':
```

```
(DEIXE-ME PENSAR UM POUCO 5,F,F)

O O X
X X X
O X O

PARTIDA TERMINADA...
```

Neste caso, o computador ganhou e todas as casas foram preenchidas. Aparece a mensagem «PARTIDA TERMINADA...» de que falámos anteriormente.

```
**********************************
          JOGO DO GALO
**********************************

   A FINALIDADE DO JOGO E':

---PARA A MAQUINA-->ALINHAR 3 X

---PARA SI --------->ALINHAR 3 O

==================================

  CADA UM JOGA A VEZ

- VOCE E' O JOGADOR 2
- O COMPUTADOR E'O JOGADOR 1

QUAL O NUMERO DO JOGADOR
QUE COMECA (1 OU 2) ? ----->1

* AGORA EU... A MINHA JOGADA E':

(DEIXE-ME PENSAR UM POUCO 5,F,F)

- - -
- X -
- - -

E'A SUA VEZ DE EFECTUAR UMA DAS
JOGADAS SEGUINTES:
```



```
          1
O - -
- X -
- - -

          2
- O -
- X -
- - -

** QUAL A SUA JOGADA (<=2) ?

JOGADA ------->2

ESCOLHEU A JOGADA 2

* AGORA EU... A MINHA JOGADA E':

(DEIXE-ME PENSAR UM POUCO 5,F,F)

X O -
- X -
- - -

E'A SUA VEZ DE EFECTUAR UMA DAS
JOGADAS SEGUINTES:

          1
X O O
- X -
- - -

          2
X O -
O X -
- - -

          3
X O -
- X O
- - -

          4
X O -
- X -
O - -

          5
X O -
- X -
- O -
```

```
          6
X O -
- X -
- - O

** QUAL A SUA JOGADA (<=6) ?
JOGADA -------->4
ESCOLHEU A JOGADA 4

* AGORA EU... A MINHA JOGADA E':
(DEIXE-ME PENSAR UM POUCO S.F.F)

X O -
- X -
O - X

GANHEI EU !!!
```

Terminamos aqui a exposição deste tipo de métodos correntemente utilizados na teoria dos jogos.

Para a aquisição de maiores conhecimentos sobre este domínio aconselhamos a consulta de obras especializadas.

É dada em anexo uma bibliografia detalhada. Podemos no entanto citar uma excelente obra sobre o assunto (embora redigida em inglês): trata-se de *Principles of Artificial Intelligence,* de Nilsson. A editora é a Tioga Publishing Company, Palo Alto, California (1980). Neste livro encontramos, entre outras coisas, as diferentes técnicas de pesquisas heurísticas (ou não) relativas aos problemas dos jogos, e à resolução de problemas mais na generalidade.

Nota do tradutor: A teoria dos jogos é uma técnica matemática, fazendo parte da teoria da decisão, destinada à análise de situações cujos resultados ou consequências dependem das acções de dois ou mais participantes com interesses antagónicos. As

aplicações desta teoria são, obviamente, inúmeras.

Esta teoria teve início em 1921, nas notas de Emil Borel, com os seus primeiros estudos sobre problemas de estratégia, e atingiu a «maturidade» em 1944, graças aos trabalhos de Johannes von Neumann e Oskar Morgenstern.

Capítulo 8

O caixeiro viajante

Vejamos agora um clássico em inteligência artificial ou pesquisa operacional : o «caixeiro viajante», que é um bom exemplo de pesquisa de métodos heurísticos destinados a resolver um problema combinatório à partida.

Há problemas para os quais o tempo necessário à execução aumenta de forma exponencial com o seu tamanho. O «caixeiro viajante» é desse tipo.

O problema põe-se nestes termos:

Consideremos N cidades a visitar segundo a metodologia do caixeiro viajante, conhecendo a distância entre cada uma delas. É preciso visitar cada cidade apenas uma única vez e a visita das N cidades deve ser feita o mais economicamente possível, ou seja, tomando o caminho total mais curto.

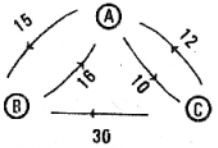
Trabalha-se também aqui no contexto de um espaço de estados, sendo cada estado uma lista de cidades a percorrer. À partida, a lista é constituída por uma só cidade e vai sendo depois aumentada à medida do percurso.

A árvore construída é explorada por métodos clássicos (em profundidade primeiro, melhorados por técnicas de *back-track*). Ela é podada à partida graças a uma heurística de decisão, de molde a conservar apenas os dados utilizáveis eficazmente.

8.1 RESOLUÇÃO DO PROBLEMA

Têm-se N cidades e conhecem-se todas as distâncias D(I,J), designando I a cidade I e J a cidade J, sendo D(I,J) a distância entre I e J.

Tomemos, por exemplo, 3 cidades A, B e C:



As cidades A,B, C são nós do grafo, e as distâncias entre elas são os valores dos arcos que ligam os nós.

Partindo de um grafo como o do exemplo acima, as soluções são já variadas. Veja-se que só se pode ir da cidade C para a cidade B e não o contrário. Podemos, por exemplo, pensar nos trajectos seguintes:

A→C→B	valor	10+30+16 = 56
B→A→C	valor	16+10+30 = 56
C→B→A	valor	30+16+10 = 56

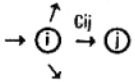
Neste caso, os trajectos são equivalentes, pois têm o mesmo valor, mas nem sempre isto acontece, como veremos mais adiante nos exemplos.

O problema consiste em encontrar um trajecto ou circuito que passe por todas as cidades uma única vez e que tenha um custo mínimo, ou seja, cujo valor seja mínimo.

O algoritmo que programámos para este livro é aproximadamente o elaborado por LITTLE em 1955.

Trabalha-se por estreitamento de limites. A ideia fundamental consiste em suprimir os arcos «caros» e só suprimir um arco na condição de se ter a certeza de existir uma alternativa menos «onerosa».

Representemos simbolicamente estes conceitos, com auxílio de uma representação gráfica:



Sendo dados o nó ⓘ e o nó ⓙ, para suprimir o arco i→j é necessário assegurarmo-nos primeiramente de que se pode chegar a ⓙ partindo de ⓘ e passando por outros arcos intermediários.

Utiliza-se uma matriz N×N, A(i,j) para armazenar os valores dos arcos entre as cidades.

Por convenção, consideram-se as distâncias como números inteiros. Assim, devemos apenas introduzir números inteiros ao inicializarmos a nossa matriz, embora o programa tenha a capacidade de eliminar, por truncagem, qualquer parte fraccionária eventualmente introduzida.

Além disso, se duas cidades não estiverem ligadas, o arco associado terá por valor 256. É pois importante ter em consideração estes limitações quando fizermos executar o programa.

A dimensão da matriz que armazena os dados introduzidos é limitada a 10. Limitamo-nos portanto a 10 cidades, principalmente devido a limitações relacionadas com o tempo de execução.

Quando introduzimos os dados devemos proceder como se segue:

Introduz-se a cidade de partida (o seu número), a cidade de chegada e a distância da cidade de partida à cidade de chegada.

Para indicar o fim da introdução de dados, termina-se por:

0,0,0

Para mais pormenores, ver os exemplos de execução.

8.2. O ALGORITMO

O programa tem a estrutura seguinte:

— um módulo de introdução e armazenamento dos números das cidades e das distâncias (linhas 500 a 580),

— um módulo de «poda» dos arcos «onerosos» da matriz: se um arco for considerado demasiadamente «oneroso», é-lhe atribuído o valor 256 (linhas 40 a 75),

— uma vez a matriz «podada» segundo uma heurística que esmiuçaremos mais adiante, efectuam-se escolhas entre os arcos restantes, gerando a par e passo o conjunto dos arcos, as suas ligações respectivas, tendo em conta as escolhas feitas. Por vezes, se a sequência das escolhas efectuadas não permitir chegar a uma conclusão, é preciso existir a capacidade de efectuar um retrocesso ou *back-track* e de refazer escolhas mais favoráveis.

Utilizam-se portanto várias sub-rotinas para, nomeadamente:

- gerar as escolhas possíveis e depois as incidências relativas às escolhas efectuadas;
- gerar as interdições;
- determinar o arco parasita;

— para finalizar, existe igualmente um módulo encarregado de editar uma solução.

Globalmente, o algoritmo de percurso é o seguinte:

Chama-se fila a uma linha ou a uma coluna e pode-se escolher deslocar-se à linha constante ou à coluna constante, segundo o contexto.

O algoritmo é dado em pseudocódigo.

Para i=1 a N *Executar*

E1: *se* existe uma fila interdita *então* ir a BT;
senão escolher a fila com menor número de escolhas possíveis
Fazer uma escolha
gerar as interdições relativas a essa escolha
se i=N−1 imprimir; ir a E2
senão ir a *Fim Para*;

BT: *se* i=1 *então FIM*;
senão i=i−1

E2: Eliminar as interdições relativas a esta escolha
Ir a E1

Fim Para

No programa em BASIC, as correspondências são as seguintes:

— o ciclo do «Para» tem início na linha 88Ø,
— E1 faz referência à linha 85,
— BT à linha 4ØØ,
— E2 à linha 415,
— FIM à linha 44Ø.

Se as *REMarks* ou comentários das linhas 9Ø, 115 e 135 causaram certa surpresa, devemo-nos lembrar de se ter referido a possibilidade de efectuar uma escolha segundo as linhas ou as colunas. A variável Y está relacionada com isto.

Com efeito, testa-se se Y tem o valor Ø, 1 ou 2, para um índice i dado (visto Y(i) ser uma variável indexada). Se Y(i) for Ø, isso significa que ainda não foi efectuada qualquer escolha para o respectivo valor de i.

Pelo contrário, se Y(i) for 1, foi efectuada uma escolha segundo as linhas, e segundo as colunas se Y(i) for 2.

Notemos também que se selecciona uma fila para nela se efectuar uma escolha se o número de escolhas possíveis é mínimo nessa fila. Isto é essencial, pois permite a deslocação num «espaço» o mais restrito possível e, portanto, limitar a «explosão»

combinatória (isto dentro do contexto de estreitamento ou restringimento de limites que preside à elaboração do algoritmo).

Analisemos agora a heurística que nos permite «podar» a matriz à partida.

Calcula-se, de certo modo, um limite ou restringimento D (trata-se aqui de um valor limitador da matriz, como vimos anteriormente, e não do conceito clássico de limite) e efectua-se o percurso da matriz restante (submatriz) segundo o algoritmo que acabámos de expor.

Para começar, calcula-se o valor de MI, que é a soma dos mínimos da matriz segundo as linhas, e o valor de MA, que representa a soma dos máximos, também segundo as linhas (linhas do programa 515 a 54Ø).

Depois, o valor limitador é dado por $D=\frac{MA-MI}{2*T*II}$ (linha 45)

com II inicializado em 6 e decrementado sempre que necessário e T=N.

Passemos agora a alguns exemplos de execução para melhor ilustrarmos o exposto:

8.3. EXECUÇÃO

```
**********************************
          CAIXEIRO VIAJANTE
**********************************

--- DIMENSAO DA MATRIZ --> 3

I,J,A(I,J)-->1,2,34
I,J,A(I,J)-->2,1,34
I,J,A(I,J)-->2,3,1
I,J,A(I,J)-->3,1,1
I,J,A(I,J)-->3,2,45
```

```
I,J,A(I,J)-->0,0,0

MATRIZ:

256  34   256
34   256  1
1    45   256

QUEIRA AGUARDAR UM POUCO...
.....
.....
IMPRESSAO DA SOLUCAO 1 :

CIDADES  1---2     DISTANCIA: 34
CIDADES  2---3     DISTANCIA: 1

CUSTO--->36

PARA DELTA=5 ---> 1 SOLUCAO
```

Como podemos verificar, os dados são reimpressos no *écran* uma vez terminada a sua introdução, permitindo a respectiva verificação.

Para o valor delta=5 (D=5), obtém-se uma solução. No caso que estamos a analisar, e em outros do género, começa-se utilizando as maiores restrições possíveis, de forma a limitar a explosão combinatória, e, no caso de não ser obtida qualquer solução, efectua-se uma segunda passagem, decrementando a variável II, o que permite «afrouxar» as restrições.

Vejamos agora um exemplo um pouco mais consequente:

```
*********************************
          CAIXEIRO VIAJANTE
*********************************

--- DIMENSAO DA MATRIZ --> 6

I,J,A(I,J)-->1,2,27
```



```
I,J,A(I,J)-->1,3,43
I,J,A(I,J)-->1,4,16
I,J,A(I,J)-->1,5,30
I,J,A(I,J)-->1,6,26
I,J,A(I,J)-->2,1,7
I,J,A(I,J)-->2,3,16
I,J,A(I,J)-->2,4,1
I,J,A(I,J)-->2,5,30
I,J,A(I,J)-->2,6,25
I,J,A(I,J)-->3,1,20
I,J,A(I,J)-->3,2,13
I,J,A(I,J)-->3,4,35
I,J,A(I,J)-->3,5,5
I,J,A(I,J)-->3,6,0
I,J,A(I,J)-->4,1,21
I,J,A(I,J)-->4,2,16
I,J,A(I,J)-->4,3,25
I,J,A(I,J)-->4,5,18
I,J,A(I,J)-->4,6,18
I,J,A(I,J)-->5,1,12
I,J,A(I,J)-->5,2,46
I,J,A(I,J)-->5,3,27
I,J,A(I,J)-->5,4,48
I,J,A(I,J)-->5,6,5
I,J,A(I,J)-->6,1,23
I,J,A(I,J)-->6,2,5
I,J,A(I,J)-->6,3,5
```

```
I,J,A(I,J)-->6,4,9

I,J,A(I,J)-->6,5,5

I,J,A(I,J)-->0,0,0

MATRIZ:

256  27   43   16   30   26
7    256  18   1    30   25
20   13   256  35   5    0
21   16   25   256  18   18
12   48   27   48   256  5
23   5    5    9    5    256

QUEIRA AGUARDAR UM POUCO...
..........
..........
..........
..........

PARA DELTA=6 ---> 0 SOLUCOES


QUEIRA AGUARDAR UM POUCO...
..........
..........
..........
..........
..........

IMPRESSAO DA SOLUCAO 1 :

CIDADES  3---2     DISTANCIA:  13
CIDADES  2---1     DISTANCIA:  7
CIDADES  1---4     DISTANCIA:  16
CIDADES  4---5     DISTANCIA:  13
CIDADES  5---6     DISTANCIA:  5

CUSTO--->64


PARA DELTA=6 ---> 1 SOLUCAO
```


Segue-se a listagem do programa. Quem dispuser de uma impressora *ZX* pode utilizar a versão para impressão de *hard-copy* dada no último parágrafo do capítulo 8.4.

```
    5 REM ************************
        .CAIXEIRO VIAJANTE
 ********************************
    6 REM
            INICIALIZACAO
 ********************************

   10 DIM A(10,10): DIM L(10): DI
M O(10): DIM B(10,10): DIM I(9):
 DIM J(9): DIM Y(9): DIM P(9): D
IM Q(9): DIM V(9): DIM T(9): DIM
 C(20)
   14 GO SUB 800
   15 INPUT "--- DIMENSAO DA MATR
IZ ? --> ";T: IF T>=10 THEN GO T
O 15
   20 FOR I=1 TO T: FOR J=1 TO T:
 LET B(I,J)=256: NEXT J: NEXT I:
 GO SUB 500: LET II=6
   25 FOR I=1 TO T: FOR J=1 TO T:
 LET A(I,J)=B(I,J): NEXT J: LET
O(I)=0: LET L(I)=0: NEXT I
   30 FOR I=1 TO 2*T: LET C(I)=0:
 NEXT I: FOR I=1 TO T-1: LET I(I
)=0: LET J(I)=0: LET Y(I)=0: NEX
T I: LET II=II-1
   35 IF II=0 THEN STOP
   40 PRINT : PRINT : PRINT "QUEI
RA AGUARDAR UM POUCO..."
   45 LET IM=0: LET D=(MA-MI)/(II
*2*T): LET D=(INT D)+1
   50 FOR K=1 TO T: FOR L=1 TO T:
 LET ML=256: LET MC=256
   55 FOR J=1 TO T: IF A(K,J)<=ML
 AND J<>L THEN LET ML=A(K,J)
   60 NEXT J
   65 FOR J=1 TO T: IF A(J,L)<=MC
 AND J<>K THEN LET MC=A(J,L)
   70 NEXT J: IF A(K,L)>(MC+D) AN
D A(K,L)>(ML+D) THEN LET A(K,L)=
256
   75 NEXT L: NEXT K
   80 FOR I=1 TO T
   85 IF Y(I)<>1 THEN GO TO 110
   90 REM
```

```
*********************************
          CASO OU TIPO=1
*********************************

  95 FOR L=J(I)+1 TO T: IF A(I(I
),L)=256 OR O(L)=1 THEN NEXT L
 100 IF L=T+1 THEN GO TO 400
 105 LET J(I)=L: GO TO 240
 110 IF Y(I)<>2 THEN GO TO 140
 115 REM
*********************************
          CASO OU TIPO=2
*********************************

 120 FOR K=I(I)+1 TO T: IF A(K,J
(I))=256 OR L(K)=1 THEN NEXT K
 125 IF K=T+1 THEN GO TO 400
 130 LET I(I)=K: GO TO 240
 135 REM
*********************************
          CASO OU TIPO=0
*********************************

 140 FOR K=1 TO T: IF L(K)<>0 TH
EN GO TO 155
 145 FOR L=1 TO T: IF A(K,L)=256
 OR O(L)=1 THEN NEXT L
 150 IF L=T+1 THEN GO TO 400
 155 NEXT K
 160 FOR L=1 TO T: IF O(L)<>0 TH
EN GO TO 175
 165 FOR K=1 TO T: IF A(K,L)=256
 OR L(K)=1 THEN NEXT K
 170 IF K=T+1 THEN GO TO 400
 175 NEXT L
 180 GO SUB 600
 185 LET M=T: FOR K=1 TO 2*T: IF
 C(K)>=M THEN GO TO 195
 190 LET M=C(K): LET K1=K
 195 NEXT K
 200 REM
*********************************
     GERAR AS INTERDICOES
*********************************

 205 IF K1>T THEN GO TO 225
 210 FOR L=1 TO T: IF A(K1,L)=25
6 OR O(L)=1 THEN NEXT L
 215 IF L=T+1 THEN GO TO 400
 220 LET I(I)=K1: LET J(I)=L: LE
T Y(I)=1: GO TO 240
 225 FOR K=1 TO T: IF A(K,K1-T)=
256 OR I(K)=1 THEN NEXT K
```



```
 230 IF K=T+1 THEN GO TO 400
 235 LET I(I)=K: LET J(I)=K1-T:
LET Y(I)=2
 240 LET I1=I
 241 REM
*********************************
   ENCONTRAR O ARCO PARASITA
*********************************

 245 FOR P=1 TO T-1: IF I(I1)=J(
P) THEN GO TO 300
 250 NEXT P: LET Q(I)=I(I1): LET
 I1=I: GO TO 305
 300 LET I1=P: GO TO 245
 305 FOR P=1 TO T-1: IF J(I1)=I(
P) THEN GO TO 315
 310 NEXT P: LET P(I)=J(I1): GO
TO 320
 315 LET I1=P: GO TO 305
 320 LET V(I)=A(P(I),Q(I)): LET
A(P(I),Q(I))=256
 325 LET L(I(I))=1: LET O(J(I))=
1
 330 IF I<T-1 THEN NEXT I
 335 GO SUB 700: GO TO 415
 400 IF I=1 THEN GO TO 425
 405 LET I(I)=0: LET J(I)=0: LET
 Y(I)=0: LET P(I)=0: LET Q(I)=0:
 LET V(I)=0
 410 LET I=I-1
 415 LET L(I(I))=0: LET O(J(I))=
0: LET A(P(I),Q(I))=V(I): LET CT
=0
 420 GO TO 85
 425 PRINT : PRINT "PARA DELTA="
;D;" ---> ";IM;" SOLUC";(+"OES"
AND IM<>1);(+"AO" AND IM=1): PRI
NT : IF IM=0 THEN GO TO 25
 440 STOP
 500 REM
*********************************
     INTRODUCAO DOS DADOS
*********************************

 502 LET MA=0: LET MI=0
 505 INPUT "INTRODUZA: I,J,A(I,J
)-->";I;",";J;",";L: IF I<=T AND
 J<=T AND I<>0 AND J<>0 THEN LET
 B(I,J)=INT L
 510 IF I<>0 THEN GO TO 500
 515 FOR I=1 TO T: LET MN=256: L
ET MX=0: FOR J=1 TO T: IF B(I,J)
<=MN THEN LET MN=B(I,J)
```

```
 520 IF B(I,J)>=MX AND B(I,J)<>2
56 THEN LET MX=B(I,J)
 525 NEXT J: LET MI=MI+MN: LET M
A=MA+MX: NEXT I
 530 FOR J=1 TO T: LET MN=256: L
ET MX=0: FOR I=1 TO T: IF B(I,J)
<=MN THEN LET MN=B(I,J)
 535 IF B(I,J)>=MX AND B(I,J)<>2
56 THEN LET MX=B(I,J)
 540 NEXT I: LET MI=MI+MN: LET M
A=MA+MX: NEXT J
 545 CLS : PRINT A$: PRINT : PRI
NT "MATRIZ:": PRINT : PRINT
 550 FOR I=1 TO T: FOR J=1 TO T:
 PRINT B(I,J);" ";
 555 IF B(I,J)<100 THEN PRINT "
";
 560 IF B(I,J)<10 THEN PRINT " "
;
 565 NEXT J: PRINT : NEXT I
 570 INPUT "CORRECTA (S/N) ? ";C
$: IF C$<>"S" AND C$<>"s" AND C$
<>"N" AND C$<>"n" THEN GO TO 570
 575 IF C$="N" OR C$="n" THEN GO
 TO 500
 580 RETURN
 600 REM
**********************************
      GERAR TODAS AS ESCOLHAS
              POSSIVEIS
**********************************

 605 FOR P=1 TO 2*T: LET C(P)=T:
 NEXT P
 610 LET M=0: FOR K=1 TO T:: FOR
 L=1 TO T: IF L(K)=1 THEN GO TO
630
 615 IF A(K,L)=256 OR O(L)=1 THE
N GO TO 625
 620 LET M=M+1: LET C(K)=M
 625 NEXT L
 630 LET M=0: NEXT K
 635 FOR L=1 TO T: FOR K=1 TO T
 640 IF O(L)=1 THEN GO TO 660
 645 IF A(K,L)=256 OR L(K)=1 THE
N GO TO 655
 650 LET M=M+1: LET C(L+T)=M
 655 NEXT K
 660 LET M=0: NEXT L: PRINT "...
......."
 665 RETURN
 700 REM
```



```
**********************************
       EDICAO DA SOLUCAO
**********************************

 705 LET CT=0: FOR P=1 TO T-1: L
ET CT=CT+A(I(P),J(P)): NEXT P
 710 LET CT=CT+V(T-1): LET IM=IM
+1
 715 PRINT : PRINT "IMPRESSAO DA
 SOLUCAO ";IM;" :": PRINT : PRIN
T
 720 LET K=1
 725 FOR P=1 TO T-1
 730 FOR L=1 TO T-1: IF I(P)<>J(
L) THEN NEXT L: LET T(K)=P
 735 NEXT P: FOR K=2 TO 5
 740 FOR P=1 TO T-1: IF I(P)<>J(
T(K-1)) THEN NEXT P
 745 LET T(K)=P: NEXT K
 750 FOR P=1 TO T-1: PRINT "CIDA
DES  ";I(T(P));"---";J(T(P));"
  DISTANCIA: ";A(I(T(P)),J(T(P))
): NEXT P: PRINT : PRINT "CUSTO-
-->";CT: PRINT : PRINT : RETURN
 800 REM
**********************************
    IMPRESSAO DAS INSTRUCOES
**********************************

 805 LET A$="******************
************      CAIXEIRO VI
AJANTE      ******************
**********"
 810 PRINT A$
 815 PRINT : PRINT " CONSIDERAM-
SE N CIDADES,AS DIS-TANCIAS,NUM
SENTIDO E NOUTRO,EN-TRE AS CIDAD
ES SAO COLOCADAS NU-MA MATRIZ A(
I,J) DE DIMENSAO TxTSENDO T=N, I
 E J SAO,RESPECTIVA-MENTE,AS CID
ADES DE PARTIDA E DECHEGADA,SEND
O A(I,J) A DISTANCIAENTRE ELAS,"
 820 PRINT " ASSIM, DEVEMOS INTR
ODUZIR PRI- MEIRO A DIMENSAO T D
A MATRIZ;DE-POIS, A VEZ, OS NUME
ROS DE DUAS CIDADES E O VALOR DA
 DISTANCIA  ENTRE ELAS,NO SENTID
O DA INTRO- DUCAO DOS DOIS NUMER
OS QUE AS   REPRESENTAM,"
 825 PRINT : PRINT " PARA FINALI
ZAR, INTRODUZIR OS  TRES VALORES
 A ZERO,"
 830 RETURN
```

As aplicações do «caixeiro viajante» são muito numerosas. É o problema clássico da electrificação de um conjunto de pontos distintos: como ligá-los o mais economicamente possível, ou seja, gastando a menor quantidade possível de fio. O mesmo se aplica a linhas telefónicas, a redes de telecomunicação informática, ao planeamento de carreiras de transportes públicos, etc.

Quem estiver interessado no aprofundamento destes conceitos terá toda a vantagem em consultar obras mais especialmente dedicadas aos problemas ligados aos grafos. Podemos citar, entre outras: *Graphes et Algorithmes* de Gondran e Minoux (Eyrolles 1979) e, é claro, as obras dedicadas à pesquisa operacional.

8.4. NOTAS E VERSÃO PARA «HARD-COPY»

A listagem apresentada pode ser facilmente convertida noutros dialectos de BASIC, pois não apresenta instruções específicas do BASIC Sinclair do *Spectrum*, à excepção, talvez, das condições entre parêntesis da linha 425 e do tratamento das variáveis inteiras, indexadas ou não. Poderão igualmente surgir discrepâncias na apresentação do *display*, devido a diferenças de tratamento das *strings* ou cadeias a serem impressas por uma declaração de INPUT.

Quanto ao primeiro ponto mencionado, a equivalência noutros BASICs pode ser encontrada nos manuais do computador utilizado, ou, simplesmente, podem omitir-se essas instruções, mantendo o plural da palavra «SOLUÇÕES». Assim, a linha 425 ficaria:

```
425 PRINT : PRINT "PARA DELTA=";D; " --> " ;IM;
    " SOLUÇÕES": PRINT : IF IM=0 THEN GO TO 25
```

Quanto ao caso das variáveis inteiras, as indexadas em particular (nomes de quadros ou vectores numéricos), bastará, na maior parte dos BASICs, acrescentar o caracter % à frente do nome da variável. No caso deste nosso programa, se o quisermos converter para BASIC Microsoft ou equivalente, podemos

limitar-nos a aplicar isso às variáveis indexadas. Desse modo, a linha 10 (por exemplo) viria (não esquecer que no BASIC Microsoft basta uma única declaração para DIMensionar uma série de *arrays*):

```
10 DIM A% (10,10), L%(10), O%(10), B%(10,10), I%(9),
   J%(9), Y%(9), P%(9), Q%(9), V%(9), T%(9), C%(20)
```

Assim, serão apenas essas as variáveis a alterar ao longo do programa.

Por fim, no que respeita às declarações de INPUT, o problema, a existir, não será de sintaxe mas sim de posições de impressão das cadeias incluídas na declaração (linha 505, por exemplo), pois alguns computadores não imprimem essas cadeias na linha de *input*. Deste modo, dever-se-á ver, para cada caso, qual o sistema utilizado e a sua influência na sequência da impressão em *écran* e, se necessário, efectuar as correcções devidas.

Tudo o que se disse não exclui que possam aparecer pequenas diferenças sintácticas em algumas declarações, facilmente resolúveis através do manual do computador utilizado.

8.4.1. Versão para «hard-copy»

Esta versão permite obter uma *hard-copy* idêntica à apresentada nos exemplos de execução. Destina-se, contudo, apenas ao Sinclair *Spectrum* e similares, equipados de uma impressora *ZX*. É claro que é possível adaptá-la a outros computadores e/ou a outras impressoras, dependendo o nível de modificação da forma como são tratados os comandos para a impressora.

Na listagem, os POKEs destinam-se a que o *Spectrum* não pare a execução (e, portanto, a impressão) com um *scroll*? sempre que o *écran* fique preenchido.

```
   5 REM ************************
         CAIXEIRO VIAJANTE
********************************
   6 REM

      VERSAO PARA HARD-COPY

   8 REM
********************************
           INICIALIZACAO
********************************

  10 DIM A(10,10): DIM L(10): DI
M O(10): DIM B(10,10): DIM I(9):
 DIM J(9): DIM Y(9): DIM P(9): D
IM Q(9): DIM V(9): DIM T(9): DIM
 C(20)
  14 GO SUB 800
  15 INPUT "--- DIMENSAO DA MATR
IZ ? --> ";T: IF T>=10 THEN GO T
O 15
  16 LPRINT A$: LPRINT : LPRINT
: LPRINT "--- DIMENSAO DA MATRIZ
 --> ";T: LPRINT
  20 FOR I=1 TO T: FOR J=1 TO T:
 LET B(I,J)=256: NEXT J: NEXT I:
 GO SUB 500: LET II=6
  25 FOR I=1 TO T: FOR J=1 TO T:
 LET A(I,J)=B(I,J): NEXT J: LET
O(I)=0: LET L(II)=0: NEXT I
  30 FOR I=1 TO 2*T: LET C(I)=0:
 NEXT I: FOR I=1 TO T-1: LET I(I
)=0: LET J(I)=0: LET Y(I)=0: NEX
T I: LET II=II-1
  35 IF II=0 THEN STOP
  38 POKE 23692,0
  40 PRINT : PRINT : PRINT "QUEI
RA AGUARDAR UM POUCO..."
  42 LPRINT : LPRINT : LPRINT "Q
UEIRA AGUARDAR UM POUCO..."
  45 LET IM=0: LET D=(MA-MI)/(II
*2*T): LET D=(INT D)+1
  50 FOR K=1 TO T: FOR L=1 TO T:
 LET ML=256: LET MC=256
  55 FOR J=1 TO T: IF A(K,J)<=ML
 AND J<>L THEN LET ML=A(K,J)
  60 NEXT J
  65 FOR J=1 TO T: IF A(J,L)<=MC
 AND J<>K THEN LET MC=A(J,L)
  70 NEXT J: IF A(K,L)>(MC+D) AN
D A(K,L)>(ML+D) THEN LET A(K,L)=
256
  75 NEXT L: NEXT K
```



```
  80 FOR I=1 TO T
  85 IF Y(I)<>1 THEN GO TO 110
  90 REM
********************************
         CASO OU TIPO=1
********************************

  95 FOR L=J(I)+1 TO T: IF A(I(I
),L)=256 OR O(L)=1 THEN NEXT L
 100 IF L=T+1 THEN GO TO 400
 105 LET J(I)=L: GO TO 240
 110 IF Y(I)<>2 THEN GO TO 140
 115 REM
********************************
         CASO OU TIPO=2
********************************

 120 FOR K=I(I)+1 TO T: IF A(K,J
(I))=256 OR L(K)=1 THEN NEXT K
 125 IF K=T+1 THEN GO TO 400
 130 LET I(I)=K: GO TO 240
 135 REM
********************************
         CASO OU TIPO=0
********************************

 140 FOR K=1 TO T: IF L(K)<>0 TH
EN GO TO 155
 145 FOR L=1 TO T: IF A(K,L)=256
 OR O(L)=1 THEN NEXT L
 150 IF L=T+1 THEN GO TO 400
 155 NEXT K
 160 FOR L=1 TO T: IF O(L)<>0 TH
EN GO TO 175
 165 FOR K=1 TO T: IF A(K,L)=256
 OR L(K)=1 THEN NEXT K
 170 IF K=T+1 THEN GO TO 400
 175 NEXT L
 180 GO SUB 600
 185 LET M=T: FOR K=1 TO 2*T: IF
 C(K)>=M THEN GO TO 195
 190 LET M=C(K): LET K1=K
 195 NEXT K
 200 REM
********************************
     GERAR AS INTERDICOES
********************************

 205 IF K1>T THEN GO TO 225
 210 FOR L=1 TO T: IF A(K1,L)=25
6 OR O(L)=1 THEN NEXT L
 215 IF L=T+1 THEN GO TO 400
 220 LET I(I)=K1: LET J(I)=L: LE
```

```
T Y(I)=1: GO TO 240
 225 FOR K=1 TO T: IF A(K,K1-T)=
256 OR I(K)=1 THEN NEXT K
 230 IF K=T+1 THEN GO TO 400
 235 LET I(I)=K: LET J(I)=K1-T:
LET Y(I)=2
 240 LET I1=I
 241 REM
**********************************
   ENCONTRAR O ARCO PARASITA
**********************************

 245 FOR P=1 TO T-1: IF I(I1)=J(
P) THEN GO TO 300
 250 NEXT P: LET Q(I)=I(I1): LET
 I1=I: GO TO 305
 300 LET I1=P: GO TO 245
 305 FOR P=1 TO T-1: IF J(I1)=I(
P) THEN GO TO 315
 310 NEXT P: LET P(I)=J(I1): GO
TO 320
 315 LET I1=P: GO TO 305
 320 LET V(I)=A(P(I),Q(I)): LET
A(P(I),Q(I))=256
 325 LET L(I(I))=1: LET O(J(I))=
1
 330 IF I<T-1 THEN NEXT I
 335 GO SUB 700: GO TO 415
 400 IF I=1 THEN GO TO 425
 405 LET I(I)=0: LET J(I)=0: LET
 Y(I)=0: LET P(I)=0: LET Q(I)=0:
 LET V(I)=0
 410 LET I=I-1
 415 LET L(I(I))=0: LET O(J(I))=
0: LET A(P(I),Q(I))=V(I): LET CT
=0
 420 GO TO 85
 425 PRINT : PRINT "PARA DELTA="
;D;" ---> ";IM;" SOLUC";(+"OES"
AND IM<>1);(+"AO" AND IM=1): PRI
NT
 430 LPRINT : LPRINT "PARA DELTA
=";D;" ---> ";IM;" SOLUC";(+"OES
" AND IM<>1);(+"AO" AND IM=1): L
PRINT : IF IM=0 THEN GO TO 25
 435 LPRINT : LPRINT : LPRINT
 440 STOP
 500 REM
**********************************
     INTRODUCAO DOS DADOS
**********************************

 502 LET MA=0: LET MI=0
```



```
 505 INPUT "INTRODUZA: I,J,A(I,J
)--->";I;",";J;",";L: IF I<=T AND
 J<=T AND I<>0 AND J<>0 THEN LET
 B(I,J)=INT L
 506 LPRINT : LPRINT "I,J,A(I,J)
--->";I;",";J;",";L
 510 IF I<>0 THEN GO TO 500
 515 FOR I=1 TO T: LET MN=256: L
ET MX=0: FOR J=1 TO T: IF B(I,J)
<=MN THEN LET MN=B(I,J)
 520 IF B(I,J)>=MX AND B(I,J)<>2
56 THEN LET MX=B(I,J)
 525 NEXT J: LET MI=MI+MN: LET M
A=MA+MX: NEXT I
 530 FOR J=1 TO T: LET MN=256: L
ET MX=0: FOR I=1 TO T: IF B(I,J)
<=MN THEN LET MN=B(I,J)
 535 IF B(I,J)>=MX AND B(I,J)<>2
56 THEN LET MX=B(I,J)
 540 NEXT I: LET MI=MI+MN: LET M
A=MA+MX: NEXT J
 545 CLS : PRINT A$: PRINT : PRI
NT "MATRIZ:": PRINT : PRINT
 550 FOR I=1 TO T: FOR J=1 TO T:
 PRINT B(I,J);" ";
 555 IF B(I,J)<100 THEN PRINT "
";
 560 IF B(I,J)<10 THEN PRINT " "
;
 565 NEXT J: PRINT : NEXT I
 570 INPUT "CORRECTA (S/N) ? ";C
$: IF C$<>"S" AND C$<>"s" AND C$
<>"N" AND C$<>"n" THEN GO TO 570
 575 IF C$="N" OR C$="n" THEN LP
RINT : LPRINT : LPRINT "CORRECCA
O:": GO TO 500
 576 LPRINT : LPRINT : LPRINT "M
ATRIZ:": LPRINT : LPRINT
 577 FOR I=1 TO T: FOR J=1 TO T:
 LPRINT B(I,J);" ";
 578 IF B(I,J)<100 THEN LPRINT "
";
 579 IF B(I,J)<10 THEN LPRINT "
";
 580 NEXT J: LPRINT : NEXT I
 590 RETURN
 600 REM
**********************************
     GERAR TODAS AS ESCOLHAS
          POSSIVEIS
**********************************

 605 FOR P=1 TO 2*T: LET C(P)=T:
```

```
 NEXT P
 610 LET M=0: FOR K=1 TO T:: FOR
 L=1 TO T: IF L(K)=1 THEN GO TO
630
 615 IF A(K,L)=256 OR O(L)=1 THE
N GO TO 625
 620 LET M=M+1: LET C(K)=M
 625 NEXT L
 630 LET M=0: NEXT K
 635 FOR L=1 TO T: FOR K=1 TO T
 640 IF O(L)=1 THEN GO TO 660
 645 IF A(K,L)=256 OR L(K)=1 THE
N GO TO 655
 650 LET M=M+1: LET C(L+T)=M
 655 NEXT K
 660 LET M=0: NEXT L: POKE 23692
,0: PRINT ".........."
 662 LPRINT ".........."
 665 RETURN
 700 REM
*********************************
        EDICAO DA SOLUCAO
*********************************

 705 LET CT=0: FOR P=1 TO T-1: L
ET CT=CT+A(I(P),J(P)): NEXT P
 710 LET CT=CT+V(T-1): LET IM=IM
+1
 711 POKE 23692,0
 715 PRINT : PRINT "IMPRESSAO DA
 SOLUCAO ";IM;" :": PRINT : PRIN
T
 717 LPRINT : LPRINT "IMPRESSAO
DA SOLUCAO ";IM;" :": LPRINT : L
PRINT
 720 LET K=1
 725 FOR P=1 TO T-1
 730 FOR L=1 TO T-1: IF I(P)<>J(
L) THEN NEXT L: LET T(K)=P
 735 NEXT P: FOR K=2 TO 5
 740 FOR P=1 TO T-1: IF I(P)<>J(
T(K-1)) THEN NEXT P
 745 LET T(K)=P: NEXT K
 746 POKE 23692,0
 750 FOR P=1 TO T-1: PRINT "CIDA
DES  ";I(T(P));"---";J(T(P));"
   DISTANCIA: ";A(I(T(P)),J(T(P))
): NEXT P: PRINT : PRINT "CUSTO-
-->";CT: PRINT : PRINT
 755 FOR P=1 TO T-1: LPRINT "CID
ADES  ";I(T(P));"---";J(T(P));"
   DISTANCIA: ";A(I(T(P)),J(T(P))
): NEXT P: LPRINT : LPRINT "CUS
```



```
TO--->";CT: LPRINT : LPRINT : RE
TURN
 760 FOR P=1 TO T-1: LPRINT "CID
ADES  ";I(T(P));"---";J(T(P));"
   DISTANCIA: ";A(I(T(P)),J(T(P))
): NEXT P: LPRINT : LPRINT "CUS
TO--->";CT: LPRINT : LPRINT : RE
TURN
 800 REM
*********************************
    IMPRESSAO DAS INSTRUCOES
*********************************

 805 LET A$="******************
*************          CAIXEIRO VI
AJANTE          ******************
*************"
 810 PRINT A$
 815 PRINT : PRINT " CONSIDERAM-
SE N CIDADES.AS DIS-TANCIAS,NUM
SENTIDO E NOUTRO,EN-TRE AS CIDAD
ES SAO COLOCADAS NU-MA MATRIZ A(
I,J) DE DIMENSAO TXTSENDO T=N. I
 E J SAO,RESPECTIVA-MENTE,AS CID
ADES DE PARTIDA E DECHEGADA,SEND
O A(I,J) A DISTANCIAENTRE ELAS."
 820 PRINT " ASSIM, DEVEMOS INTR
ODUZIR PRI- MEIRO A DIMENSAO T D
A MATRIZ;DE-POIS, A VEZ, OS NUME
ROS DE DUAS CIDADES E O VALOR DA
 DISTANCIA  ENTRE ELAS,NO SENTID
O DA INTRO- DUCAO DOS DOIS NUMER
OS QUE AS   REPRESENTAM."
 825 PRINT : PRINT " PARA FINALI
ZAR, INTRODUZIR OS  TRES VALORES
 A ZERO."
 830 RETURN
```

Capítulo 9

Os sistemas peritos

9.1 OS SISTEMAS PERITOS

Nos capítulos precedentes, apresentámos métodos heurísticos gerais com aplicação a problemas de jogos bem diversos. Esses métodos destinam-se a tratar eficazmente problemas combinatórios por natureza. Contudo, em aplicações complexas, obtêm-se frequentemente com esses métodos resultados bastante fracos.

Nos anos sessenta, os investigadores de inteligência artificial dedicaram-se a desenvolver métodos heurísticos eficazes e generalizáveis com aplicação a problemas diversos (xadrez, *puzzle* 8 e *puzzle* 15, tratamento automático de línguas, robótica, demonstração automática de teoremas, etc.).

Os resultados não foram brilhantes. A fases iniciais eufóricas de prometedoras sucederam-se fases mais ponderadas. Com efeito, os algoritmos utilizados em inteligência artificial assentam, como vimos, numa análise e numa enumeração mais ou menos exaustiva de escolhas ou decisões elementares e o rendimento da execução evolui de forma inversamente exponencial com a dimensão do problema.

Algoritmos como o Min-Max, a truncagem Alfa-Beta derivada do Min-Max e dos métodos precedentes, permitem «podar» ou reduzir de forma apreciável as arborescências.

Contudo, estes métodos são decepcionantes em casos concretos: não existe qualquer método geral que permita resolver correctamente problemas demasiadamente diversificados, a partir de um simples enunciado e de algumas regras de jogo, por exemplo.

Os investigadores de inteligência artificial defrontam-se desde há alguns anos com o problema de inserir e utilizar eficazmente em programas uma massa importante de conhecimentos.

Esses conhecimentos são de natureza diversa:

— factos que exprimem uma situação, relações, etc. Esses factos são, na maioria das vezes, independentes uns dos outros;

— conhecimentos sobre a maneira de tratar os factos (como utilizá-los, em que momento, por que ordem).

Está-se longe da organização hierarquizada ou relacional pura dos SGBD (Sistemas de Gestão das Bases de Dados), onde as ligações são «congeladas».

Além do mais, esses conhecimentos podem evoluir com o tempo; podem, por exemplo, tornar-se parcialmente incorrectos ou incompletos.

Quando da concepção de um programa que os utilize, é preciso ter isso em consideração.

Classicamente, os informáticos tratam procedimentarmente a informação, ou seja, adoptam estruturas de dados adaptadas a esses dados e elaboram algoritmos em função desses dados particulares. Assim, quando estes últimos variam, é necessário modificar o algoritmo.

Tomemos o caso inicial de uma contabilidade a informatizar. Escreve-se um programa para efectuar essa contabilidade. Contudo, com o decorrer do tempo, os dados evoluem; há mais coisas a ter em conta ou algumas regras de cálculo são modificadas. A adaptação necessita então de reprogramação de uma boa parte do sistema primitivo. A modificação, no nosso país, dos programas de contabilidade para adaptação ao IVA ainda estará certamente por bastante tempo na memória de muitos informáticos portugueses.

Um algoritmo tem, portanto, o inconveniente de estar «congelado»; a sua modificação é uma operação delicada e leva fre-

quentemente ao aparecimento de muitos erros.

Do mesmo modo, a pesquisa dos testes de optimização, das operações, etc. , é efectuada «à mão», o que implicita uma vez mais um risco de erro suplementar.

Pelo contrário, num sistema perito, os conhecimentos são enunciados de forma declarativa, isto é, exprimem factos (situações, relações) sem neles ser incluída a maneira como devem ser utilizados: dados e modo de utilização são separados.

Nos sistemas peritos, os conhecimentos são normalmente representados por regras de produção. *Mycin,* um sistema perito especializado no diagnóstico de afecções bacterianas, adopta uma representação de conhecimentos desse tipo.

Nesse sistema, a regra 85 é a seguinte:

Regra 85

Se	1)	o local da cultura é o sangue,
e *se*	2)	o organismo é Gram negativo,
e *se*	3)	o organismo tem a forma de bastonete,
e *se*	4)	o paciente é um hospedeiro potencial,
então		é provável (0,6 ou 60%) que o organismo seja o *pseudomonia aeruginosa.*

A vantagem evidente, sobretudo neste domínio onde o conhecimento é imperfeito, é a das regras serem facilmente modificáveis. A capacidade de ter simultaneamente em conta vários factores presta-se igualmente bem a um formalismo de tipo «regra de produção».

Precisemos que um conhecimento modular, como neste caso, permite facilmente o diálogo com o utilizador: o sistema tem a faculdade de explicar o seu comportamento.

Quando de um exame terapêutico, como no caso da utilização de *Mycin,* a obtenção de um diagnóstico tem de poder ser explicada, ou seja, o sistema deve poder precisar que passos seguiu para estabelecer o diagnóstico.

9.2. TERMINOLOGIA

Num sistema perito (ou *expert system*), os dados e o modo de utilização desses dados são, como vimos, separados.

Um sistema perito trata uma soma importante de conhecimentos relativos a uma domínio particular à semelhança de um perito humano, operacional num domínio pontual que domina muito particularmente, pois possui nele uma vasta experiência e dele tem conhecimentos suficientes.

Chama-se *base de conhecimentos* a esse conjunto de regras fornecidas pelo perito.

O *motor de inferências* é, pelo seu lado, a parte processual ou procedimentar do sistema: é ele que constrói o raciocínio a par e passo, revela o conjunto dos factos que são adquiridos a cada etapa do raciocínio, etc. O motor de inferências é portanto encarregado de efectuar os conhecimentos a utilizar num momento dado, de os encadear correctamente, etc.

Ao conjunto de factos adquiridos ao longo de todo o raciocínio desenvolvido dá-se o nome de *base de factos.*

Além disso, *interfaces*-utilizador, como módulos de acesso conversacional, completam estes sistemas. Permitindo colocar questões ao utilizador, eles são necessários quando da edição, da revelação da base de conhecimentos ou da consulta do sistema (em modo explicação do comportamento de um sistema perito por ele próprio, por exemplo).

Distinguem-se motores *sem variável,* também conhecidos por *motores essenciais* ou *motores zero,* e motores *com variáveis.* Para ambos os casos existem numerosos exemplos de sistemas peritos. Vejamos alguns:

1.º caso: (sem variável)

— DENDRAL dá a fórmula desenvolvida de um corpo orgânico a partir da sua fórmula bruta e do espectro de massa.

— MÉTA-DENDRAL infere automaticamente regras de produção utilizadas por DENDRAL.

— MYCIN-TEIRESIAS-GUIDON.

O primeiro é um sistema auxiliar de diagnóstico e de tratamento das afecções bacterianas do sangue.

TEIRESIAS é um módulo de aquisição de novos conhecimentos.

GUIDON (que se poderia traduzir por guia ou guião) é um sistema auxiliar de ensino baseado no programa Mycin (ensina aquilo em que *Mycin* trabalha).

— LITHO é um auxiliar à prospecção mineira.

— NUDGE permite a elaboração de empregos do tempo.

A lista é longa. Para mais detalhes, devemos consultar a bibliografia indicada no fim do livro.

2.º caso: (com variáveis)

Podemos citar os motores SNARK e TANGO. O primeiro é um motor de inferências sobre o qual foram desenvolvidos diversos sistemas (auxiliares no diagnóstico de avarias, jogo de bridge, simulação de um raciocínio de pesquisa arqueológica, demonstração automática, etc.). Existe uma versão deste motor em PL/1 e outra em Pascal.

TANGO é igualmente um motor com variáveis, mas escrito em LISP. É utilizado em aplicações do tipo «ensino assistido por computador».

Iremos debruçar-nos aqui sobre o motor essencial. Para mais detalhes sobre os motores de inferências com variáveis, podemos, entre outros, consultar o livro de M. Gondran *Introduction aux systèmes experts,* das Éditions Eyrolles.

Os motores essenciais adoptam, a maior parte das vezes, uma estratégia de encadeamento retrógrado (*back-chaining*), que definiremos adiante. Um motor deste tipo é utilizado nomeadamente no *Emycin,* motor de inferências da lógica das proposições, incorporado no *Mycin, Puff* e *Litho* (*Puff* é um sistema perito especializado no diagnóstico das doenças pulmonares).

Vejamos a lógica de funcionamento de um sistema perito por meio de um exemplo:

9.3 LÓGICA DO SISTEMA PERITO ESSENCIAL

Considere a seguinte base de conhecimentos:

R_1:	B e D e E	→F
R_2:	D e G	→A
R_3:	C e F	→A
R_4:	B	→X
R_5:	D	→E
R_6:	A e X	→H
R_7:	C	→D
R_8:	X e C	→A
R_9:	X e B	→D
base de factos:		B,C
objectivo:		H

9.3.1. O encadeamento directo

Em primeiro lugar, convém esclarecermos o que é «encadeamento», nesta acepção também conhecido pelo vocábulo inglês *chaining*. Assim, podemos defini-lo como sendo uma técnica que consiste em interligar informações a fim de as tratar sucessivamente num encadeamento automático, permitindo aplicar-lhes o mesmo tratamento.

Ora no encadeamento directo, ou *front chaining,* o raciocínio efectua-se utilizando as regras a partir dos factos conhecidos (à partida B e C) até que o objectivo H seja deduzido. Quando

uma regra é utilizada, são acrescentados novos factos à base de factos. Podem conceber-se diversas estratégias dirigidas à escolha da regra a aplicar primeiramente. Contudo, pode simplesmente optar-se por aplicar a primeira regra possível seguindo a ordem de enumeração (ver f.①). Senão, pode optar-se por aplicar a regra que comporte o maior número de condições (ver f.②). Uma outra estratégia poderia consistir em aplicar a regra contendo uma condição referente ao último facto acrescentado à base de factos (ver f.③).

9.3.2. O encadeamento retrógrado

Neste tipo de encadeamento, procura-se demonstrar se o facto H é verificado: é o objectivo procurado.

O método utilizado é recursivo: examinam-se todas as regras que contêm o objectivo (facto H) nas suas partes *consequências* (a parte direita no nosso exemplo do parágrafo 9.3).

Se o valor lógico do objectivo é conhecido (verdadeiro ou falso) terminou-se o encadeamento. Caso contrário, tenta-se demonstrar o objectivo a partir de regras de dedução: cada uma das *premissas* (elementos da parte esquerda no nosso exemplo) é, por sua vez, considerada como um subobjectivo, do qual é necessário conhecer o valor lógico (verdadeiro ou falso). A pesquisa é classicamente efectuada em profundidade primeiro (como no nosso exemplo) ou em largura primeiro, dependendo de o objectivo ter sido gerado como uma *pilha* ou como uma *fila*.

Quando um subobjectivo é demonstrado, é acrescentado à base de factos. No nosso exemplo, a base de factos inicial é constituída por dois factos B e C. O objectivo a demonstrar é H.

Determinar H equivale a demonstrar A e X. A estratégia adoptada é neste caso a de profundidade primeiro: procura-se primeiramente demonstrar A, o qual pode ser verificado pelas regras R_2, 3_3, R_8. A primeira regra aplicável conduz aos subobjectivos D e G. D é deduzido a partir de R_7, pois C encontra-se na base de factos inicial, e é, portanto, conhecido.

D é assim acrescentado à base de factos: {B,C,D}

G não pode ser deduzido de forma alguma (em certos sistemas, autoriza-se a este nível uma pergunta ao utilizador sobre a natureza lógica de G).

Neste caso, conhece-se *a priori* a base de factos à partida e G não é conhecido.

Há, portanto, *insucesso*; é necessário voltar atrás, efectuar um *back-track* até à regra deixada de lado em último lugar, ou seja, R_3, que é a segunda regra que permite determinar A.

Os subobjectivos a considerar são então C e F. F é dedutível por meio de R_1 a partir de B, D e E. B e D encontram-se na base de factos. Assim, apenas E tem de ser demonstrado. Isto é efectuado graças à regra R_5, pois D já é conhecido. Como B está incluído na base de factos, X está demonstrado (regra R_4).

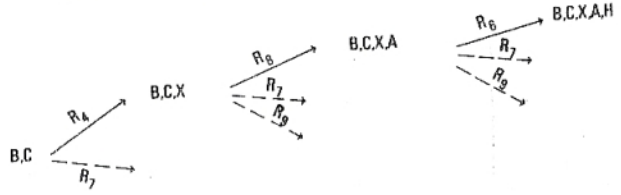
O processo então pára, pois o objectivo H está demonstrado (regra R_6).

Para um sistema sem variável, a organização do motor dito motor «zero» activado em encadeamento retrógrado é relativamente simples, do ponto de vista conceptual.

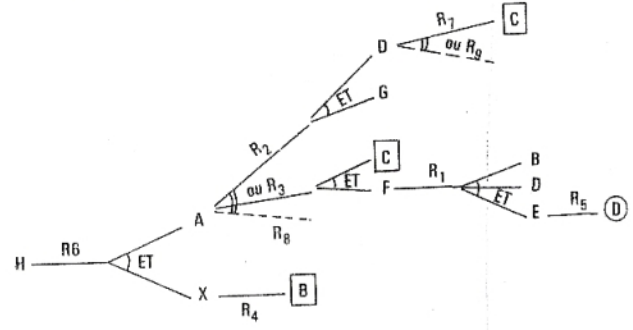
① *Funcionamento em encadeamento directo, de acordo com a primeira estratégia.*

Os arcos (setas) representam as regras aplicadas. Os tracejados representam as regras aplicáveis não seleccionadas pelo motor.

② *Funcionamento em encadeamento directo, de acordo com a segunda estratégia.*



③ *Funcionamento em encadeamento directo, de acordo com a terceira estratégia.*



④ *Funcionamento em encadeamento retrógrado, segundo uma pesquisa em profundidade primeiro.*

Por convenção, os factos representados dentro de um quadrado □ encontram-se na base de factos inicial.

Os que se encontram dentro de um círculo ○ são demonstrados no decurso da pesquisa, como é nomeadamente o caso do facto D.

Podemos esquematizar o funcionamento geral do motor da forma seguinte:

São utilizados dois procedimentos: a função VERIFICAR trata um nó OU da arborescência E/OU construída. A função VERIFICAR-E trata um nó E da arborescência E/OU.

Uma variável booleana OK toma o valor «verdadeiro» quando VERIFICAR ou VERIFICAR-E o são.

Para a função VERIFICAR, OK é verdadeira se se tem uma regra que permita demonstrar o objectivo.

Para VERIFICAR-E, OK só é verdadeira se todos os objectivos a estabelecer forem verificados como verdadeiros.

Algoritmicamente, podemos esquematizar isto da seguinte maneira:

procedimento OK:=VERIFICAR (objectivo)

OK:=falso
se objectivo está na base de factos
|então OK:=verdadeiro
|*senão*
| *para* todas as regras que tenham o objectivo em conclusão
| *e dado_que* OK é falso:
| |escolher uma regra R entre estas últimas
| |OK:=VERIFICAR-E (premissas de R)
| |
| |*Fim_dado_que*
| |
| *se OK falso e se* objectivo exigível (não está na base de factos)

então OK:=questão (objectivo)
acrescentar a resposta à base de factos

Fim_se

Fim_procedimento

procedimento OK:=VERIFICAR-E (conjunto de subobjectivos)

OK:=verdadeiro
para todos os objectivos do conjunto dos objectivos *e dado_que* OK é verdadeiro:

escolher um objectivo B entre os do conjunto
OK:=VERIFICAR (B)
Fim_dado_que

Fim_procedimento

9.4. UM EXEMPLO DE RESOLUÇÃO

Vejamos agora um microssistema perito, inspirado num número da revista *Byte* (Setembro de 1981). Existe uma versão em LISP deste microssistema na obra de Winston e Horn *LISP* da Addison-Wesley (1981), microssistema esse que podemos igualmente encontrar no livro de M. Gondran *Introduction aux systèmes experts*, Éditions Eyrolles (1983). O motor de inferências é activado em encadeamento retrógrado e a estratégia em profundidade primeiro é adoptada; utiliza-se pois uma pilha para armazenar elementos intermediários.

A sintaxe das regras é a seguinte (utilizando uma notação semelhante à notação BNF):

regara=::<nome de regra>, se, <antecedente 1>...<antecedente n>, ENTÃO, <consequente 1>, ...,<consequente m>

antecedente=::<proposição>

consequente=::<proposição>

No jogo de regras que propomos não há praticamente senão um único consequente.

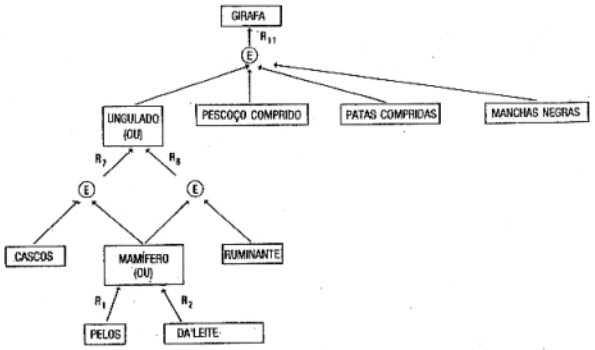
O programa funciona como o algoritmo precedente: para cada hipótese possível, o programa experimenta a lista das regras de forma a validar ou invalidar a hipótese em questão.

Se a hipótese pode ser deduzida, isso significa que ela é um termo consequente de uma regra. É preciso, pois, determinar se os termos antecedentes são verdadeiros ou falsos.

O funcionamento é recursivo: procura-se demonstrar uma lista de subobjectivos.

A representação do problema é arborescente. Para cada hipótese (considerada como objectivo corrente) é construída por programa, à medida da progressão da pesquisa, uma árvore de resolução do objectivo (ou plano de resolução do objectivo).

Deste modo, para o objectivo Girafa que iremos ver, temos a árvore seguinte:



Plano de resolução do objectivo Girafa

No programa, as regras encontram-se integradas nas declarações DATA.

A ordem pela qual elas são enunciadas na nossa listagem determina a ordem em que representámos os factos no diagrama acima. É também a ordem segundo a qual os factos irão, à vez, ser examinados (partindo do canto superior direito e progredindo para a esquerda).

Para mais detalhes, consultar os exemplos de resolução.

```
   5 REM
**********************************
            GIRAFA
**********************************

   6 REM
      MINI-SISTEMA PERITO

  VERSAO PARA O ZX SPECTRUM DE
  CARLOS PERES SEBASTIAO E SILVA
© GABINETE VERBO DE INFORMATICA
  1987
**********************************

  10 DIM A$(100,21): DIM E$(21):
 DIM F$(100,21): DIM H$(20,21):
DIM Q(51): DIM R$(250,21): DIM
R(50): DIM S$(200,21): DIM S(100
)
  15 LET A9=100: LET F9=100: LET
 H9=20: LET R9=250: LET S8=200:
LET S9=200: LET S1=0: LET S2=0
  20 GO SUB 9000: PRINT "BONS DI
AS..."
  25 GO SUB 500: IF R7>0 THEN GO
 TO 35
  30 PRINT "ERRO!!!": STOP
  35 GO SUB 550: IF H8>0 THEN GO
 TO 45
  40 PRINT "ERRO!!!": STOP
  45 PRINT "VOU UTILIZAR AS MINH
AS ";R7;" REGRAS": PRINT "PARA E
STABELECER UMA DAS ";H8;" HIPO-"
: PRINT "TESES (BEM, VOU TENTAR.
..) :": PRINT : PRINT
```



```
  46 PAUSE 200: POKE 23692,0
  50 FOR H=1 TO H8
  55 PRINT "     ";Z$;" ";H$(H):
NEXT H
  60 PRINT : PRINT "RESPONDA COM
:": PRINT "SIM (S), NAO (N) E P
ORQUE? (P)"
  65 LET A1=0: LET F1=0
  67 GO SUB 9200
  70 FOR H=1 TO H8: LET E$=H$(H)
: LET Y=1: GO TO 1200
  75 GO SUB 150: IF X$(1)<>"*" T
HEN GO TO 90
  80 NEXT H
  85 PRINT : PRINT T$: PRINT "
 NENHUMA HIPOTESE PODE SER
          CONFIRMADA": PRINT T$:
GO TO 95
  90 PRINT : PRINT : LET W$=H$(H
): GO SUB 9100: PRINT T$: PRINT
"   CONCLUO QUE :"/"    ";Z$;" ";
H$(H, TO CS);".": PRINT T$: PRIN
T
  95 PRINT : INPUT "RECOMECA-SE(
R) OU DESISTE-SE(D)?";C$: IF C$=
"R" OR C$="r" THEN PRINT : PRINT
 K$: GO TO 65
 100 IF C$<>"D" AND C$<>"d" THEN
 GO TO 95
 105 PRINT : PRINT "** TIVE MUIT
O GOSTO EM CONVERSAR   CONSIGO.,
.."/"    ATE' A PROXIMA!": STOP
 110 REM
**********************************
          EMPILHAMENTO
**********************************

 115 IF S1<59 THEN GO TO 125
 120 PRINT "A PILHA TRANSBORDOU!
!": STOP
 125 LET S1=S1+1: LET S$(S1)=X$:
 RETURN
 130 REM
**********************************
          EMPILHAMENTO
**********************************

 135 IF S2<58 THEN GO TO 145
 140 PRINT "A PILHA TRANSBORDOU!
!": STOP
 145 LET S2=S2+1: LET S(S2)=X: R
ETURN
 150 REM
```

```
*********************************
        DESEMPILHAMENTO
*********************************

 155 IF S1>0 THEN GO TO 165
 160 PRINT "PILHA VAZIA!": STOP
 165 LET X$=S$(S1): LET S1=S1-1:
 RETURN
 170 REM
*********************************
        DESEMPILHAMENTO
*********************************

 175 IF S2>0 THEN GO TO 185
 180 PRINT "PILHA VAZIA!": STOP
 185 LET X=S(S2): LET S2=S2-1: R
ETURN
 190 REM
*********************************
        CHAMADA DO FACTO
*********************************

 195 LET N$="*": IF F1=0 THEN RE
TURN
 200 FOR I=1 TO F1: IF G$=F$(I)
THEN GO TO 210
 205 NEXT I: RETURN
 210 LET N$=G$
 215 RETURN
 220 REM
*********************************
        ADICAO DO FACTO
*********************************

 225 LET M$="*"
 230 GO SUB 190: IF N$(1)<>"*" T
HEN RETURN
 235 IF F1<F9 THEN GO TO 245
 240 PRINT "FICHEIRO DE FACTOS C
HEIO!": STOP
 245 LET F1=F1+1: LET F$(F1)=G$:
 LET M$=G$: RETURN
 250 REM
*********************************
        APLICACAO DE REGRA
*********************************

 255 LET U$="*": LET R2=R(R1)+2
 260 LET G$=R$(R2): IF G$( TO 5)
="ENTAO" THEN GO TO 270
 265 LET R2=R2+1: GO TO 260
 270 LET R2=R2+1:: LET G$=R$(R2)
: IF G$( TO 4)="STOP" THEN RETUR
N
```



```
 275 IF R$(R2+1, TO 2)="SE" THEN
 RETURN
 280 GO SUB 220
 285 IF M$(1)="*" THEN GO TO 270
 286 LET W$=G$: GO SUB 9100
 290 PRINT : PRINT "DA REGRA ";R
$(R(R1), TO 3);" DEDUZ-SE :"/;Z$
;" ";G$( TO CS)
 295 LET U$="T": GO TO 270
 300 REM
*********************************
        REGRAS A TESTAR
*********************************

 305 LET Q(1)=0
 310 FOR V=1 TO R7: LET R4=R(V)+
2
 315 LET G$=R$(R4): IF G$( TO 5)
="ENTAO" THEN GO TO 325
 320 LET R4=R4+1: GO TO 315
 325 LET R4=R4+1: LET G$=R$(R4):
 IF G$( TO 4)="STOP" THEN GO TO
345
 330 IF R$(R4+1, TO 2)="SE" THEN
 GO TO 345
 335 IF G$<>E$ THEN GO TO 325
 340 LET Q(1)=Q(1)+1: LET Q(Q(1)
+1)=V
 345 NEXT V: RETURN
 350 REM
*********************************
        FACTO EXIGIVEL
*********************************

 355 LET B$="*": IF A1=0 THEN GO
 TO 370
 360 FOR B=1 TO A1: IF E$=A$(B)
THEN RETURN
 365 NEXT B
 370 IF A1<A9 THEN GO TO 380
 375 PRINT "FICHEIRO CHEIO!!": S
TOP
 380 LET A1=A1+1: LET A$(A1)=E$
 385 GO SUB 9230: LET W$=E$: GO
SUB 9100: PRINT : PRINT FLASH 1;
">"; FLASH 0;" E/VERDADE QUE "/Z
$;" ";E$( TO CS);" ? ": INPUT "S
IM(S),NAO(N),PORQUE?(P)-->";C$
 390 IF C$="S" OR C$="s" THEN GO
 TO 460
 395 IF C$="N" OR C$="n" THEN GO
 SUB 9200: RETURN
 400 IF C$<>"P" AND C$<>"p" THEN
 GO TO 385
```

```
 405 IF E$<>H$(H) THEN GO TO 415
 410 GO TO 385
 415 PRINT : PRINT "ESTOU A EXPE
RIMENTAR A REGRA ";R$(R(R1)): LE
T V=R(R1)+2: IF V=R2 THEN GO TO
430
 420 PRINT : PRINT "EU SEI QUE "
 425 PRINT Z$;" ";R$(V): LET V=V
+1: IF V<R2 THEN GO TO 425
 430 PRINT "SE:"
 435 PRINT Z$;" ";R$(V): LET V=V
+1: IF R$(V, TO 5)<>"ENTAO" THEN
 GO TO 435
 440 PRINT "ENTAO:": LET V=V+1
 445 PRINT Z$;" ";R$(V): LET V=V
+1: IF R$(V, TO 4)="STOP" THEN G
O TO 455
 450 IF R$(V+1, TO 2)<>"SE" THEN
 GO TO 445
 455 PRINT : GO TO 385
 460 LET G$=E$: GO SUB 220: LET
B$="T": RETURN
 500 REM
*********************************
     CARREGAMENTO DAS REGRAS
*********************************

 505 READ Z$: LET R7=0: LET R8=0
 510 IF R8<R9 THEN GO TO 520
 515 PRINT "ERRO!!!": STOP
 520 LET R8=R8+1: READ R$(R8)
 525 IF R$(R8, TO 2)<>"SE" THEN
GO TO 535
 530 LET R7=R7+1: LET R(R7)=R8-1
 535 IF R$(R8, TO 4)<>"STOP" THE
N GO TO 510
 540 RETURN
 545 REM
*********************************
     CARREGAMENTO DAS HIPOTESES
*********************************

 550 LET H8=0: IF H8<H9 THEN GO
TO 560
 555 PRINT "ERRO!!!": STOP
 560 LET H8=H8+1: READ H$(H8): I
F H$(H8, TO 4)<>"STOP" THEN GO T
O 560
 565 LET H8=H8-1
 570 RETURN
1190 REM
```



```
*********************************
      ROTINA DE AGULHAGEM
*********************************

1200 LET G$=E$: GO SUB 190: IF N
$(1)<>"*" THEN GO TO 1705
1205 GO SUB 300: LET Q8=Q(1): IF
 Q8>0 THEN GO TO 1310
1300 GO SUB 350: IF B$(1)="*" TH
EN GO TO 1710
1305 GO TO 1705
1310 LET Q1=1
1315 LET X$=E$: GO SUB 110: LET
X=Y: GO SUB 130
1320 FOR W=1 TO Q8: LET X=Q(W+1)
: GO SUB 130: NEXT W
1325 LET X=Q8: GO SUB 130: LET X
=Q1: GO SUB 130
1500 LET R1=Q(Q1+1): LET Y=2: GO
 TO 2840
1550 GO SUB 150: LET Y$=X$: GO S
UB 170: LET Q1=X: GO SUB 170: LE
T Q8=X
1600 FOR W=Q8 TO 1 STEP -1: GO S
UB 170: LET Q(W+1)=X: NEXT W
1650 GO SUB 170: LET Y=X: GO SUB
 150: LET E$=X$: IF Y$(1)="T" TH
EN GO TO 1705
1660 LET Q1=Q1+1: IF Q1<=Q8 THEN
 GO TO 1315
1700 GO TO 1710
1705 LET X$="T": GO TO 1715
1710 LET X$="*"
1715 GO SUB 110
1720 IF Y=1 THEN GO TO 75
1730 IF Y=2 THEN GO TO 1550
1740 IF Y=3 THEN GO TO 2950
1750 IF Y=4 THEN GO TO 3300
2840 LET X=R1: GO SUB 130: LET X
=Y: GO SUB 130: LET Y=3
2850 GO TO 3130
2950 GO SUB 150: GO SUB 170: LET
 Y=X: GO SUB 170: LET R1=X
3000 IF X$(1)="*" THEN GO TO 308
0
3010 GO SUB 250: IF U$(1)="*" TH
EN GO TO 3080
3060 LET X$="T": GO TO 3100
3080 LET X$="*"
3100 GO SUB 110
3105 IF Y=1 THEN GO TO 75
3110 IF Y=2 THEN GO TO 1550
3115 IF Y=3 THEN GO TO 2950
```

```
3120 IF Y=4 THEN GO TO 3300
3130 LET R2=R(R1)+2
3170 LET G$=R$(R2): IF G$( TO 5)
="ENTAO" THEN GO TO 3410
3200 LET X=Y: GO SUB 130: LET X=
R1: GO SUB 130: LET X=R2: GO SUB
 130
3210 LET Y=4: LET E$=G$: GO TO 1
200
3300 GO SUB 150: GO SUB 170: LET
 R2=X: GO SUB 170: LET R1=X: GO
SUB 170: LET Y=X
3310 IF X$(1)="*" THEN GO TO 343
0
3320 LET R2=R2+1: GO TO 3170
3410 LET X$="T"
3430 GO SUB 110
3440 IF Y=1 THEN GO TO 75
3450 IF Y=2 THEN GO TO 1550
3460 IF Y=3 THEN GO TO 2950
3470 IF Y=4 THEN GO TO 3300
5990 REM
*********************************
      INSTRUCOES DATA
*********************************

6000 DATA "ESTE ANIMAL"
6010 DATA "R1","SE","TEM PELOS",
"ENTAO","E'MAMIFERO"
6015 DATA "R2","SE","DA'LEITE","
ENTAO","E'MAMIFERO"
6020 DATA "R3","SE","TEM PENAS",
"ENTAO","E'AVE"
6025 DATA "R4","SE","VOA","E'OVI
PARO","ENTAO","E'AVE"
6030 DATA "R5","SE","COME CARNE"
,"ENTAO","E'CARNIVORO"
6035 DATA "R6","SE","TEM DENTES
AGUCADOS","TEM GARRAS","TEM OLHO
S A FRENTE","ENTAO","E'CARNIVORO
"
6040 DATA "R7","SE","E'MAMIFERO"
,"TEM CASCOS","ENTAO","E'UNGULAD
O"
6045 DATA "R8","SE","E'MAMIFERO"
,"RUMINA","ENTAO","E'UNGULADO"
6050 DATA "R9","SE","E'MAMIFERO"
,"E'CARNIVORO","TEM COR FULVA","
TEM MANCHAS NEGRAS","ENTAO","E'P
UMA"
6055 DATA "R10","SE","E'MAMIFERO
","E'CARNIVORO","TEM COR FULVA",
"TEM LISTAS NEGRAS","ENTAO","E'T
IGRE"
```



```
6058 DATA "R11","SE","E'UNGULADO
","TEM PESCOCO COMPRIDO","TEM PA
TAS COMPRIDAS","TEM MANCHAS NEGR
AS","ENTAO","E'GIRAFA"
6060 DATA "R12","SE","E'UNGULADO
","TEM LISTAS NEGRAS","ENTAO","E
'ZEBRA"
6065 DATA "R13","SE","E'AVE","NA
O VOA","TEM PESCOCO COMPRIDO","E
'PRETO E BRANCO","ENTAO","E'AVES
TRUZ"
6070 DATA "R14","SE","E'AVE","NA
O VOA","NADA","E'PRETO E BRANCO"
,"ENTAO","E'PINGUIM"
6075 DATA "R15","SE","E'AVE","VO
A BEM","ENTAO","E'ALBATROZ"
6080 DATA "STOP"
6085 DATA "E'ALBATROZ","E'PINGUI
M","E'AVESTRUZ","E'ZEBRA","E'GIR
AFA","E'TIGRE","E'PUMA","STOP"
8999 STOP
9000 REM
*********************************
     ROTINA DE CABECALHO
*********************************

9005 CLS
9010 LET K$="********************
*************          >>> GIRAFA
  <<<                 SISTEMA PE
RITO          ********************
*************"
9015 PRINT K$: PRINT : PRINT
9020 LET T$=K$( TO 32): LET K$="
*********** RECOMECO ***********
"
9025 RETURN
9100 REM
*********************************
  ROTINA DE ACERTO DE CADEIAS
*********************************

9110 FOR I=1 TO 20
9120 IF W$(I)=" " AND W$(I+1)="
" THEN GO TO 9140
9130 NEXT I
9140 LET CS=I-1
9150 RETURN
9200 REM
```

```
*********************************
*      ROTINA DE ESPERA         *
*********************************

9210 PRINT #0;"...AGUARDE UM POU
CO,POR FAVOR..."
9220 RETURN
9230 INPUT ""
9240 RETURN
```

As sub-rotinas incluídas nesta listagem realizam as operações que mencionámos anteriormente:

— gestão de pilha: «empilhamento» e «desempilhamento» dos factos;

— procura (evocação) de um facto para verificar se já foi determinada a sua natureza lógica;

— adição de um facto;

— aplicação de uma regra com gestão da base de factos (adição de novos factos à base);

— determinação das regras aplicáveis;

— pedido ao utilizador para que este indique a natureza de um facto (natureza verdadeira ou falsa).

Aqui, as únicas respostas aceitáveis são Sim (S) ou Não (N) e o sistema apenas faz perguntas acerca de factos terminais, embora se pudessem fazer perguntas ao utilizador sobre os factos intermediários, o que teria como efeito reduzir o número de «armazenamentos» a efectuar na pilha, e não desencadear a operação recursiva de dedução acerca do facto senão quando o dito utilizador respondesse «não sei».

Nos exemplos de execução seguintes, podemos verificar que a ordem pela qual as hipóteses são examinadas é a mesma dos factos incluídos nas declarações DATA: o sistema tenta sucessivamente demonstrar os objectivos: albatroz, pinguim, avestruz, zebra, girafa, tigre, puma.

O jogo de regras é o seguinte:

R_1 *se* <tem pelos> *então* <é mamífero>
R_2 *se* <dá leite> *então* <é mamífero>
R_3 *se* <tem penas> *então* <é ave>
R_4 *se* <voa>, <é ovíparo> *então* <é ave>
R_5 *se* <come carne> *então* <é carnívoro>
R_6 *se* <tem dentes pontiagudos>, <tem garras>, <tem olhos à frente> *então* <é carnívoro>
R_7 *se* <é mamífero>, <tem cascos> *então* <é ungulado>
R_8 *se* <é mamífero>, <rumina> *então* <é ungulado>
R_9 *se* <é mamífero>, <é carnívoro>, <tem cor fulva>, <tem manchas negras> *então* <é puma>
R_{10} *se* <é mamífero>, <é carnívoro>, <tem cor fulva>, <tem listas negras> *então* <é tigre>
R_{11} *se* <é ungulado>, <tem pescoço comprido>, <tem patas compridas>, <tem manchas negras>, *então* <é girafa>
R_{12} *se* <é ungulado>, <tem listas negras> *então* <é zebra>
R_{13} *se* <é ave>, <não voa>, < tem pescoço comprido>, <é preto e branco> *então* <é avestruz>
R_{14} *se* <é ave>, <não voa>, <nada>, <é preto e branco> *então* <é pinguim>
R_{15} *se* <é ave>, <voa bem> *então* <é albatroz>

É preciso, sobretudo, prestar atenção ao formato segundo o qual as regras são introduzidas nas declarações DATA. Um simples erro de ortografia ou um espaço em branco a mais ou a menos numa das cadeias de caracteres teria um efeito desastroso no comportamento do nosso programa. Vejamos agora alguns exemplos de execução:

```
*********************************
        >>> GIRAFA <<<
        SISTEMA PERITO
*********************************

BONS DIAS...
VOU UTILIZAR AS MINHAS 15 REGRAS
```

```
PARA ESTABELECER UMA DAS 7 HIPO-
TESES (BEM, VOU TENTAR...) :

     ESTE ANIMAL E'ALBATROZ

     ESTE ANIMAL E'PINGUIM

     ESTE ANIMAL E'AVESTRUZ

     ESTE ANIMAL E'ZEBRA

     ESTE ANIMAL E'GIRAFA

     ESTE ANIMAL E'TIGRE

     ESTE ANIMAL E'PUMA

RESPONDA COM :
SIM (S), NAO (N) E PORQUE? (P)

> E'VERDADE QUE
ESTE ANIMAL TEM PENAS ?

SIM(S),NAO(N),PORQUE?(P)-->S

DA REGRA R3  DEDUZ-SE :
ESTE ANIMAL E'AVE

> E'VERDADE QUE
ESTE ANIMAL VOA BEM ?

SIM(S),NAO(N),PORQUE?(P)-->S

DA REGRA R15 DEDUZ-SE :
ESTE ANIMAL E'ALBATROZ

**********************************
   CONCLUO QUE :
   ESTE ANIMAL E'ALBATROZ.
**********************************

RECOMECA-SE(R) OU DESISTE-SE(D)?
R

*********** RECOMECO ***********

> E'VERDADE QUE
ESTE ANIMAL TEM PENAS ?

SIM(S),NAO(N),PORQUE?(P)-->S
```



```
DA REGRA R3  DEDUZ-SE :
ESTE ANIMAL E'AVE

> E'VERDADE QUE
ESTE ANIMAL VOA BEM ?

SIM(S),NAO(N),PORQUE?(P)-->N

> E'VERDADE QUE
ESTE ANIMAL NAO VOA ?

SIM(S),NAO(N),PORQUE?(P)-->S

> E'VERDADE QUE
ESTE ANIMAL NADA ?

SIM(S),NAO(N),PORQUE?(P)-->S

> E'VERDADE QUE
ESTE ANIMAL E'PRETO E BRANCO ?

SIM(S),NAO(N),PORQUE?(P)-->S

DA REGRA R14 DEDUZ-SE :
ESTE ANIMAL E'PINGUIM

**********************************
   CONCLUO QUE :
   ESTE ANIMAL E'PINGUIM.
**********************************

RECOMECA-SE(R) OU DESISTE-SE(D)?
R

*********** RECOMECO ***********

> E'VERDADE QUE
ESTE ANIMAL TEM PENAS ?

SIM(S),NAO(N),PORQUE?(P)-->N

> E'VERDADE QUE
ESTE ANIMAL VOA ?

SIM(S),NAO(N),PORQUE?(P)-->N

> E'VERDADE QUE
ESTE ANIMAL TEM PELOS ?

SIM(S),NAO(N),PORQUE?(P)-->S

DA REGRA R1  DEDUZ-SE :
```

```
ESTE ANIMAL E'MAMIFERO

> E'VERDADE QUE
ESTE ANIMAL TEM CASCOS ?

SIM(S),NAO(N),PORQUE?(P)-->S

DA REGRA R7  DEDUZ-SE :
ESTE ANIMAL E'UNGULADO

> E'VERDADE QUE
ESTE ANIMAL TEM LISTAS NEGRAS ?

SIM(S),NAO(N),PORQUE?(P)-->P

ESTOU A EXPERIMENTAR A REGRA R12


EU SEI QUE
ESTE ANIMAL E'UNGULADO

SE:
ESTE ANIMAL TEM LISTAS NEGRAS

ENTAO:
ESTE ANIMAL E'ZEBRA


> E'VERDADE QUE
ESTE ANIMAL TEM LISTAS NEGRAS ?

SIM(S),NAO(N),PORQUE?(P)-->N

> E'VERDADE QUE
ESTE ANIMAL TEM PESCOCO COMPRIDO
 ?

SIM(S),NAO(N),PORQUE?(P)-->S

> E'VERDADE QUE
ESTE ANIMAL TEM PATAS COMPRIDAS
?

SIM(S),NAO(N),PORQUE?(P)-->S

> E'VERDADE QUE
ESTE ANIMAL TEM MANCHAS NEGRAS ?


SIM(S),NAO(N),PORQUE?(P)-->S

DA REGRA R11 DEDUZ-SE :
ESTE ANIMAL E'GIRAFA
```



```
**********************************
   CONCLUO QUE :
   ESTE ANIMAL E'GIRAFA.
**********************************


RECOMECA-SE(R) OU DESISTE-SE(D)?
R

*********** RECOMECO ***********

> E'VERDADE QUE
ESTE ANIMAL TEM PENAS ?

SIM(S),NAO(N),PORQUE?(P)-->N

> E'VERDADE QUE
ESTE ANIMAL VOA ?

SIM(S),NAO(N),PORQUE?(P)-->N

> E'VERDADE QUE
ESTE ANIMAL TEM PELOS ?

SIM(S),NAO(N),PORQUE?(P)-->N

> E'VERDADE QUE
ESTE ANIMAL DA'LEITE ?

SIM(S),NAO(N),PORQUE?(P)-->N

**********************************
.  NENHUMA HIPOTESE PODE SER
          CONFIRMADA
**********************************

RECOMECA-SE(R) OU DESISTE-SE(D)?
R

*********** RECOMECO ***********

> E'VERDADE QUE
ESTE ANIMAL TEM PENAS ?

SIM(S),NAO(N),PORQUE?(P)-->S

DA REGRA R3  DEDUZ-SE :
ESTE ANIMAL E'AVE

> E'VERDADE QUE
ESTE ANIMAL VOA BEM ?

SIM(S),NAO(N),PORQUE?(P)-->N
```

```
> E'VERDADE QUE
ESTE ANIMAL NAO VOA ?

SIM(S),NAO(N),PORQUE?(P)-->S

> E'VERDADE QUE
ESTE ANIMAL NADA ?

SIM(S),NAO(N),PORQUE?(P)-->N

> E'VERDADE QUE
ESTE ANIMAL TEM PESCOCO COMPRIDO
 ?

SIM(S),NAO(N),PORQUE?(P)-->S

> E'VERDADE QUE
ESTE ANIMAL E'PRETO E BRANCO ?

SIM(S),NAO(N),PORQUE?(P)-->P

ESTOU A EXPERIMENTAR A REGRA R13


EU SEI QUE
ESTE ANIMAL E'AVE

ESTE ANIMAL NAO VOA

ESTE ANIMAL TEM PESCOCO COMPRIDO

SE:
ESTE ANIMAL E'PRETO E BRANCO

ENTAO:
ESTE ANIMAL E'AVESTRUZ


> E'VERDADE QUE
ESTE ANIMAL E'PRETO E BRANCO ?

SIM(S),NAO(N),PORQUE?(P)-->S

DA REGRA R13 DEDUZ-SE :
ESTE ANIMAL E'AVESTRUZ


**********************************
   CONCLUO QUE :
   ESTE ANIMAL E'AVESTRUZ.
**********************************
```



```
RECOMECA-SE(R) OU DESISTE-SE(D) ?
R

*********** RECOMECO ***********

> E'VERDADE QUE
ESTE ANIMAL TEM PENAS ?

SIM(S),NAO(N),PORQUE?(P)-->N

> E'VERDADE QUE
ESTE ANIMAL VOA ?

SIM(S),NAO(N),PORQUE?(P)-->N

> E'VERDADE QUE
ESTE ANIMAL TEM PELOS ?

SIM(S),NAO(N),PORQUE?(P)-->S

DA REGRA R1  DEDUZ-SE :
ESTE ANIMAL E'MAMIFERO

> E'VERDADE QUE
ESTE ANIMAL TEM CASCOS ?

SIM(S),NAO(N),PORQUE?(P)-->N

> E'VERDADE QUE
ESTE ANIMAL RUMINA ?

SIM(S),NAO(N),PORQUE?(P)-->N

> E'VERDADE QUE
ESTE ANIMAL COME CARNE ?

SIM(S),NAO(N),PORQUE?(P)-->S

DA REGRA R5  DEDUZ-SE :
ESTE ANIMAL E'CARNIVORO

> E'VERDADE QUE
ESTE ANIMAL TEM COR FULVA ?

SIM(S),NAO(N),PORQUE?(P)-->P

ESTOU A EXPERIMENTAR A REGRA R10


EU SEI QUE
ESTE ANIMAL E'MAMIFERO

ESTE ANIMAL E'CARNIVORO
```

```
SE:
ESTE ANIMAL TEM COR FULVA

ESTE ANIMAL TEM LISTAS NEGRAS

ENTAO:
ESTE ANIMAL E'TIGRE


> E'VERDADE QUE
ESTE ANIMAL TEM COR FULVA ?

SIM(S),NAO(N),PORQUE?(P)-->S

> E'VERDADE QUE
ESTE ANIMAL TEM LISTAS NEGRAS ?

SIM(S),NAO(N),PORQUE?(P)-->N

> E'VERDADE QUE
ESTE ANIMAL TEM MANCHAS NEGRAS ?

SIM(S),NAO(N),PORQUE?(P)-->S

DA REGRA R9  DEDUZ-SE :
ESTE ANIMAL E'PUMA


**********************************
   CONCLUO QUE :
   ESTE ANIMAL E'PUMA.
**********************************

RECOMECA-SE(R) OU DESISTE-SE(D)?
D

** TIVE MUITO GOSTO EM CONVERSAR
   CONSIGO...
   ATE'A PROXIMA!
```

Embora a aplicação seja aqui rudimentar, o motor é, contudo, geral e pode adaptar-se sem problemas, por exemplo, a um outro jogo de regras muito conhecido, extraído de Laurière TSI, 1982, vol. 1.

Tal como para o jogo de regras anterior, convém prestar muita atenção ao introduzir as linhas DATA.

Assim, de modo a proceder à adaptação mencionada, devemos, na listagem do programa, substituir a linha

6000 DATA "ESTE ANIMAL"

por

6000 DATA "ESTE VEGETAL"

Também a linha 6085 deve ser substituída, neste caso por:

6085 DATA "E'COLIBACILO", "E'COGUMELO", "E'ALGA", "E'FETO", "E'MUSGO", "E'LILAS", "E'ANEMONA", "E'CAMPAINHA", "E'ABETO"

Modificar as linhas 6010 a 6075 de acordo com o seguinte conjunto de regras:

R_1 *se* flor *e* semente *então* fanerogâmica
R_2 *se* fanerogâmica *e* semente a descoberto *então* abeto
R_3 *se* fanerogâmica *e* 1-cotilédone *então* monocotiledónea
R_4 *se* fanerogâmica *e* 2-cotilédone *então* dicotiledónea
R_5 *se* monocotiledónea *e* rizoma *então* campainha
R_6 *se* dicotiledónea *então* anémona
R_7 *se* monocotiledónea *e* não-rizoma *então* lilás
R_8 *se* folha *e* flor *então* criptogâmica
R_9 *se* criptogâmica *e* não-raiz *então* musgo
R_{10} *se* criptogâmica *e* raiz *então* feto
R_{11} *se* não-folhas *e* planta *então* talófita
R_{12} *se* talófita *e* clorofila *então* alga
R_{13} *se* talófita *e* não-clorofila *então* cogumelo
R_{14} *se* não-folha *e* não-flor *e* não-planta *então* colibacilo

Devemos igualmente introduzi-las segundo o formato:

6010 DATA "R1", "SE", "E'FLOR", "E'SEMENTE", "ENTAO", "E'FANEROGAMICA"
6015 DATA "R2", "SE", "E'FANEROGAMICA", "E'SEMENTE A DESCOBERTO", "ENTÃO", "E'ABETO"

Etc...

Nas instruções DIM da linha 1Ø, substituir 21 por 23, e, na linha 911Ø, 20 por 22, de modo a acertar os quadros com o comprimento máximo das cadeias.

Este exemplo é igualmente tratado no livro *Pratiquez l'Intelligence Artificielle* de Aubert e Schomberg, das Éditions Eyrolles, 2.ª edição, 1985.

Em resumo, neste programa são assim utilizados todos os métodos e algoritmos sobre os quais nos debruçámos até aqui, a saber:

— a recursão ou recursividade (e a noção de pilha que lhe está associada);

— as representações de tipo arborescente ou em árvore e os seus percursos possíveis (percurso anterior, interior, posterior, largura primeiro);

— os métodos de pesquisa (combinatórios ou heurísticos, tais como funções de avaliação, *back-track*, etc.).

Além disso, comparando-o com os outros programas deste livro, podemos verificar que os conhecimentos são, neste caso, representados de forma externa (em delarações DATA) e são facilmente modificáveis sem que o núcleo procedimentar do sistema tenha de ser revisto ou modificado.

É, entre outras coisas, este aspecto de modularidade, que é tão sedutor, que explica em parte o sucesso dos sistemas peritos na medicina, na química, na robótica, na electrónica, no ensino assistido por computador, etc...

Precisemos ainda que a aplicação que acabámos de analisar aqui é muito rudimentar, e apenas se destina a mostrar «como a coisa funciona».

Para mais pormenores sobre os sistemas peritos, devemos reportar-nos à bibliografia apresentada no fim do livro.

9.5. NOTAS SOBRE O PROGRAMA

Como mencionámos já por várias vezes, os programas deste livro foram «traduzidos» para o BASIC do *Spectrum* a partir das versões originais codificadas em BASIC Microsoft, e sofreram, necessariamente, maior ou menor número de alterações.

As alterações introduzidas neste programa derivam principalmente do facto de o *Spectrum* tratar os quadros *(arrays)* de cadeias alfanuméricas *(strings)* de uma forma pouco *standard*. Com efeito, definido um quadro de cadeias (através de uma declaração DIM) como contendo x cadeias de comprimento y [DIM Q$(x,y), por exemplo], todas as cadeias colocadas no quadro (através de READ, por exemplo) ficarão sempre com comprimento y, sendo truncadas se o seu comprimento for superior ou, caso contrário, sendo-lhes acrescentados espaços em branco até o seu comprimento igualar y.

Assim, houve a necessidade de acrescentar uma rotina de acerto de cadeias (linhas 91ØØ a 915Ø) de modo a lidar com os espaços extra. Pela mesma razão, e em complemento à rotina, foram introduzidas instruções de segmentação de cadeias *(string slicing)* numa série de linhas, a saber:

Linhas 9Ø, 26Ø, 27Ø, 275, 29Ø, 315, 325, 33Ø, 385, 435, 445, 45Ø, 525, 535 e 56Ø.

Deste modo, se quisermos traduzir este programa para outra versão de BASIC que empregue um tratamento de quadros diferente do do *Spectrum,* deveremos eliminar essas instruções «TO» (no caso do BASIC Microsoft Standard) ou substituí-las por instruções equivalentes no dialecto de BASIC utilizado. As notas no final de alguns dos capítulos anteriores podem servir de esboço a essas alterações. A rotina de acerto deve, obviamente, ser eliminada ou alterada, assim como se devem modificar as declarações DIM da linha 1Ø, caso necessário. A linha 46, contendo declarações próprias ao BASIC do *Spectrum,* as declarações FLASH da linha 385 e a declaração de segmentação da cadeia K$ na linha 9Ø2Ø devem igualmente ser adapta-

das, se tal se impuser. As declarações DATA não devem exigir qualquer modificação.

Por fim, resta mencionar que o conjunto de declarações IF Y ... das linhas 1720-1750 e 3440-3470 pode ser substituído por uma declaração ON Y ... nos BASICs que a utilizem, não sendo, contudo, imprescindível tal substituição. Quaisquer outras discrepâncias deverão ser tratadas de acordo com o manual do computador utilizado.

Bibliografia

Para uma maior documentação, aconselhamos a consulta das seguintes obras:

A) Em língua francesa:

BONNET Alain. — *L'Intelligence Artificielle, Promesses et Réalités.* Inter-Éditions (1984).

CORDIER Marie-Odile. — *Les systèmes experts. La Recherche* (Janeiro 1984).

FIESCHI Marius. — *Systèmes experts en médecine.* Masson (1984).

GONDRAN Michel, MINOUX. — *Graphes et algorithmes.* Eyrolles (1979).

GONDRAN Michel. — *Introduction aux systèmes experts.* Eyrolles (1983).

KRUTCH John. — *Expériences d'Intelligence Artificielle en Basic.* Eyrolles (1984).

LAURIÈRE J. Louis; *Représentation et Utilisation des connaissances.* TSI, volumes 1 e 2 (1982).

QUEINNEC Christian. *LISP, langage d'un autre type.* Eyrolles.

SCHOMBERG Aubert. — *Pratiquez L'Intelligence Artificielle.* 1985, Eyrolles, 2.ª edição.

B) Em língua inglesa:

NILSSON Nils. — *Principles of Artificial Intelligence.* Tioga Company, Palo Alto, California (1980)

HAYES-ROTH, WATERMAN, LENAT. — *Building Expert Systemes.* Addison-Wesley (1983).

WINSTON Horn. — *LISP.* Addison-Wesley (1981).

Para o caso mais específico do *Spectrum,* podemos ainda consultar:

BRAIN, KEITH E STEVEN. — *Artificial Intelligence on the Spectrum Computer.* Sunshine Books (1984).

HARTENELL Tim. — *Exploring Artificial Intelligence on Your Spectrum + and Spectrum.* (1984).

Estes dois últimos livros, embora de menor envergadura que os outros citados, são todavia interessantes. Cremos existirem traduções em língua portuguesa.

Existe ainda, no que respeita a revistas da especialidade, um número de *Byte* inteiramente consagrado à inteligência artificial:

Byte, Setembro 1981.

e um outro consagrado aos sistemas peritos:

Byte, Janeiro 1985.

ERRATA

Página	*Linha*	*Onde está*	*Leia-se*
13	9	*Mbits*	*Mbytes*
15	6	arvore de	árvore ou grafo de
18	13 (do texto)	avalização	avaliação
24	gravura	rac	raiz
	legenda da gravura	eliminar a alínea b	
26	gravura	IP=1?	IP<=1?

27 Nota a que se refere a antepenúltima linha:
Devido ao facto de o BASIC do *Spectrum* não admitir nomes de quadros assim como de variáveis de cadeia, ou literais, com mais de um carácter, a notação difere um pouco da utilizada no texto para definir os quadros. Um microcomputador que empregue o BASIC Microsoft, por exemplo, pode utilizar notação idêntica à do texto.