EDUCAÇÃO e TECNOLOGIA

Revista do Instituto Politécnico da Guarda

"EDUCAÇÃO E TECNOLOGIA"
Revista do Instituto Politécnico da Guarda

DIRECTOR: João Bento Raimundo

REDACÇÃO: Rua Comandante Salvador do Nascimento
Telef. 21634 6300 GUARDA

PROPRIEDADE: Instituto Politécnico da Guarda

EXECUÇÃO GRÁFICA: Secção de Reprografia do IPG

Depósito Legal N.º 17.891/87

N.º 2 / Janeiro / 88

IMPORTA SABER SER

Não é sem um certo orgulho que publicamos o segundo número da revista do Instituto Politécnico da Guarda.

Pretendemos abrir um espaço de comunicação — fizemo-lo;

Pretendemos a participação de alunos, professores, comunidade — tivemo-la;

Pretendemos que houvesse lugar à informação, à reflexão, à troca de experiências — conseguimo-lo.

A verdade é que a adesão foi entusiástica e a aceitação foi expressa em inúmeras palavras de estímulo que nos incentivam a continuar na procura de maior qualidade.

O segundo número de "Educação e Tecnologia" vai coincidir não só com a abertura do novo ano escolar da Escola Superior de Educação, o segundo, mas também com o início das actividades lectivas de mais uma escola — a Escola Superior de Tecnologia da Guarda.

Numa sociedade confrontada com rápidos e constantes progressos no domínio científico tecnológico e com a consequente evolução, ou seja, uma sociedade em constante mutação, requere-se um homem novo capaz de protagonizar a mudança e, ao mesmo tempo, capaz de se manter fiel a si próprio nessa mesma sociedade.

Na formação dos técnicos e professores do I.P.G. tentamos que se desenvolva a capacidade de participar livre e plenamente em actividades comuns numa perspectiva de realização em comunidade. Tal só é possível com o alargamento do horizonte cultural e cada um interagindo, aprendendo com os outros e proporcionando aos outros condições de aprendizagem na condição de que, mais do que saber **ou** saber fazer, **o que importa é** saber ser.

"Educação e Tecnologia" pretende, afinal, afirmar-se como um dos muitos meios para o conseguir.

João Bento Raimundo

Presidente da C.I. do Instituto Politécnico da Guarda

ESTRUTURAS ARBORECENTES — UM EXEMPLO

Por: Álvaro Bento Leal: Prof. Coordenador do I.P.G.

RESUMO: **Como exemplo, da utilização das estruturas arborecentes, é apresentada e desenvolvida uma base de dados, onde o acesso se processa através de chaves de comprimento variável.**

Ao autor, apreciador do jogo de xadrez, pôs-se o problema do registo, no computador, das sequências de jogadas que formam aquilo a que se chama a teoria das aberturas e que se encontra dispersa por milhares de publicações da especialidade.

Consiste o problema no armazenamento e posterior acesso, de sequências de jogadas, alternadamente das brancas e pretas, iniciadas pelo 1.º lance e formadas por sucessivas respostas até determinado nível. O grau de profundidade atingido, da ordem dos 15 a 40 lances, depende, regra geral, da preferência que à linha de jogo é dada pelos jogadores mais fortes. Note-se que cada jogada tem várias respostas a serem consideradas, em termos gerais, o número de hipóteses pode ir até 6 ou 7, isto porque os lances obviamente fracos não necessitam de registo.

O primeiro aspecto a ter em conta é o do elevado volume de informação a armazenar, pelo que são críticos os aspectos da facilidade de acesso e da não redundância no registo da informação.

Um outro aspecto a assinalar e que não tem merecido muita atenção na literatura, reside no facto de o acesso pretender fazer-se através de chaves, supostas formadas pela concatenação dos lances que formam uma sequência, de comprimento variável.

Deixando para trás o problema do xadrez, pretende-se a estruturação de dados identificados por concatenações, de comprimento variável, de um conjunto de valores de um dado atributo, não existindo qualquer chave cuja sequência de valores seja idêntica a uma sub-sequência, iniciada no 1.º elemento, de uma outra chave.

A estrutura adequada que ocorre de imediato é a arborecente. O estudo das árvores encontra-se com frequência na literatura, ver (1) e (2). No que se segue o aspecto focado será a da implantação prática.

A estrutura adoptada está representada esquematicamente na Fig. 1

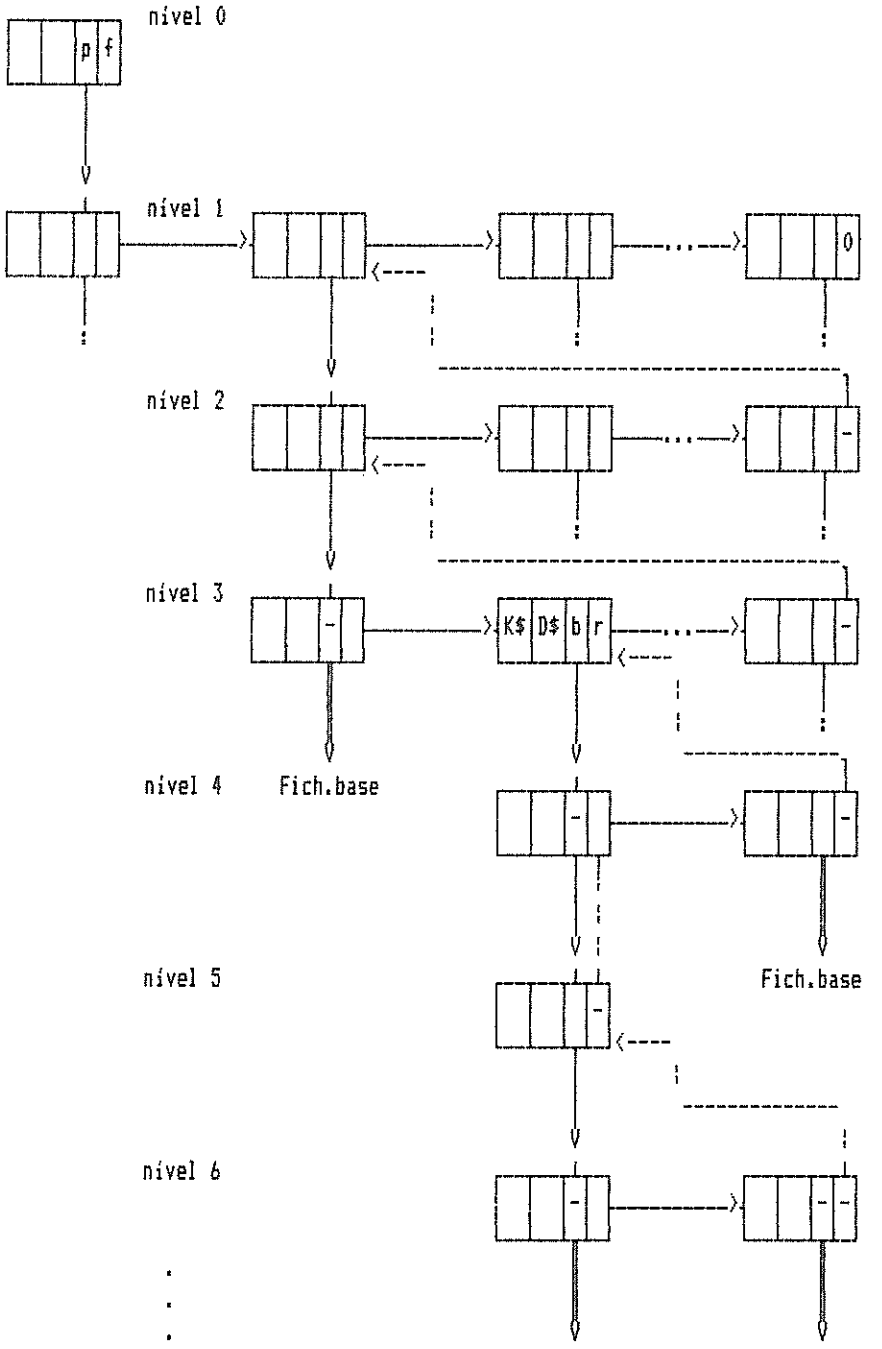
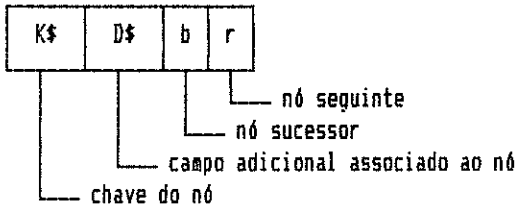


Fig. 1

Cada um dos nós da árvore consta do valor do atributo que forma a chave, de um campo com informação associada a esse valor e dois endereços.

O primeiro endereço referencia o nó sucessor na árvore, o segundo aponta para o nó que se designará por seguinte e que é caracterizado por ter antecessor comum e a chave com o valor imediatamente superior.



Alguns dos nós são excepção a este caso:

— O nó raíz da árvore, isto é, o que não tem antecessor, contem unicamente a referência do 1.º nó do 1.º nível.

— Os últimos nós das cadeias com nó antecessor comum referenciam o antecessor respectivo. A adopção deste critério simplifica o algoritmo de pesquisa na árvore. Para distinguir o valor do endereço, em relação ao caso normal, convenciona-se representá-lo por um número negativo em módulo igual ao endereço do nó que se pretende referenciar. No caso do 1.º nível, onde há somente uma cadeia a referência **r** do último nó faz-se igual a zero.

— Nos nós que não possuem sucessores, nós terminais, o endereço **b** é utilizado para apontar para um ficheiro de base a que a estrutura arborecente serve de índice. Para distinguir dos outros casos convenciona-se que o valor representado é o número do registo do ficheiro base, com sinal negativo.

Nos moldes descritos verifica-se que a estriutura, assim definida, tem uma utilidade prática que ultrapassa o problema inicialmente posto. O caso mais usual de índices de chaves de comprimento fixo está incluído, basta supor a cadeia de caracteres que formam a chave, decomposta em sub-cadeias. Adoptando a divisão da chave em sub-cadeias é imediata a solução do problema da criação de índices formados por textos longos e comprimento variável.

A utilização em programação, da estrutura de dados descrita, pode realizar-se através de um conjunto de subprogramas que possibilitam a realização das operações de manutenção e pesquisa da informação suportada pela estrutura.

Cada subprograma executa uma determinada função sobre à base de dados. Para efeitos de utilização, são suficientes as seguintes funções: abertura, actualização, leitura sequencial, corte e fecho.

Antes de se passar à descrição destas funções, faz-se notar o seguinte:

Todo o armazenamento da estrutura é suposto ser realizado num

ficheiro de acesso directo por número relativo, correspondendo a cad nó um registo.

No caso de dados com eliminações e introduções frequentes, alt volatibilidade, o número de registos inutilizados devido às eliminaçõe pode ser importante, pelo que é adoptado um sistema que permite re-utilização desses registos.

Esse sistema consiste na formatação inicial do ficheiro com os regis tos sucessivos encadeados, isto é, cada registo contém o endereço d registo livre seguinte. Quando um registo da estrutura for eliminad será encadeado com o 1.º livre, passando a exercer esta função. Un registo introduzido na estrutura ocupará fisicamente a posição do 1. livre, passando esta função a ser preenchida pelo registo que era refe renciado por ele.

Na formatação, o 1.º registo passa a conter o nó raiz da árvore, send escrito com dois endereços, o 1.º é zero e o 2.º é 2, isto é, o 1.º regist livre no início. O 1.º endereço do nó raiz passa a apontar para o 1.º n do 1.º nível, logo que sejam introduzidos dados.

A lógica de construção dos subprogramas, que correspondem às fun ções indicadas atrás, é descrita em pseudo-código no Apêndice.

Os subprogramas compartilham informação comum, nomeadament a localização actual na estrutura. Esta localização é transmitida na cha mada de cada um dos subprogramas, excepto no que realiza a funçã de abertura, e é devolvida, após eventual alteração, na saída dess mesmo subprograma.

Entende-se por localização actual o percurso formado pelos nós do sucessivos níveis até ao nó que foi acedido mais recentemente.

Os dados compartilhados pelos subprogramas são:

"FILE" — indentificação do ficheiro que contem os registos com a informação dos nós.

LSTA — indicador que pode apresentar os valores:
LSTA = 0 — significa que se está localizado no nó raíz.
LSTA = 1 — indica que a localização actual é um nó terminal.
LSTA = − 1 — indica localização num nó que possue suces sores.

e — no caso de LSTA = 1, representa o número do registo base que é re ferenciado pelo nó terminal.

m — número inteiro que indica o nível do nó onde se está localizado

a (i) (i = 1,2,...m) — endereços dos nós que formam o percurso do 1.º nív até ao nó onde se está localizado.

TS (i) (i = 1,2,...m) — chaves dos nós do percurso da localização actual.

I$ (i) (i = 1,2,...m) — campos adicionais dos nós do percurso da localizaçã actual.

p — endereço do 1.º nó do 1.º nível.

f — endereço do 1.º registo livre.

Passa-se a descrever cada uma das funções a implantar assim com resumidamente, a lógica da sua construção.

— ABERTURA

Lê o registo número 1 do ficheiro onde se armazena a estrutura, registo esse que possue os valores de **p** e **f.**

Se o parâmetro **add** é diferente de zero, significa que as operações que irão ser realizadas na sessão ocasionarão, em princípio, um aumento do número de registos ocupados, pelo que se pretende que exista o número mínimo de **ext** de registos livres.

Após a leitura do registo 1, é lida a sequência de registos livres.

No caso de o número de registos livres ser menor que **ext,** e **add** ≠ **0,** escrevem-se **ext** registos encadeados com o último, anteriormente, livre.

A localização final é o nó raíz.

— LOCALIZAÇÃO

Esta função permite a viagem através dos nós da estrutura arborecente.

Na utilização é fornecido um percurso, isto é, uma sequência de **n** nós, identificados pelas chaves **V$** (i) (i = 1,2...n), onde **i** se refere ao nível do nó e, pretende-se o acesso ao percurso, armazenado na estrutura, formado por esses nós.

O valor de saída **C** indica o sucesso ou não da operação, assim: **C** = 0 significa que existe o percurso dado, **C** = 1 indica que o percurso não existe, neste caso é devolvido como percurso actual o que existir com o o maior número de nós comuns ao dado.

A lógica é a seguinte:

1) Identifica-se a parte comum do percurso dado e o da localização à entrada.

2) A partir do último nó encontrado em 1) desce-se, de acordo com o endereço **b,** para o nível seguinte.

3) Visitam-se os nós seguindo os endereços **r** até encontrar um nó com chave maior ou igual à do correspondente nível no percurso dado. Se a chave encontrada for igual volta-se a 2), se não se tratar de um nó terminal e o nível atingido for menor que **n.** No caso de a chave ser maior ou o nó atingido ser o último da cadeia a pesquisa termina.

— LEITURA SEQUENCIAL

Fornece o percurso seguinte ao actual, ou o 1.º percurso no caso de **LSTA** = 0. Entende-se por percurso seguinte o que se obtém aplicando as seguintes regras:

1) Se **LSTA** ≠ 1 então, avança-se na árvore a partir de **a (m),** seguindo o percurso definido pelos endereços **b** (nós sucessores) até, atingir um nó terminal.

2) Se **LSTA** = 1, isto é, a localização actual é um nó terminal, segue-se o endereço **r,** isto é, acede-se ao nó seguinte e volta a aplicar-se a regra 1). Se o nó seguinte não existir, passa-se ao nó antecessor e aplica-se esta mesma regra 2).

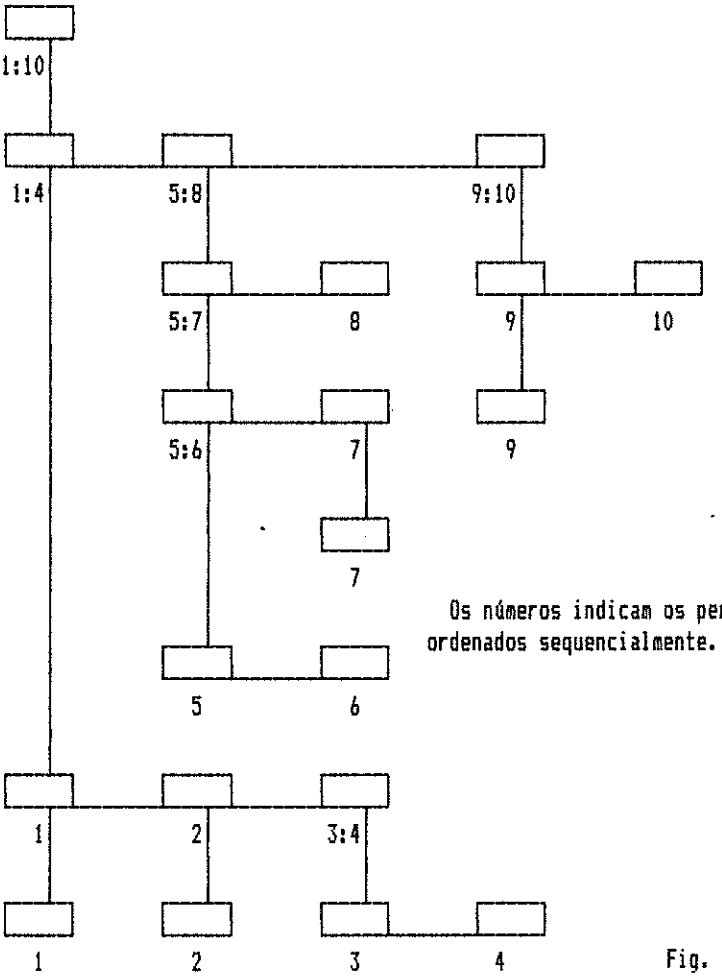


Fig. 2

O percurso encontrado passa a ser a localização actual.

C = 1 indica que a estrutura está vazia ou que a localização era formada pelos últimos nós das cadeias de todos os níveis.

— ACTUALIZAÇÃO

Nesta função engloba as operações de adição de novos nós à estrutura, a alteração dos campos adicionais e dos números de registo do ficheiro base, associados aos nós existentes.

Ao subprograma é fornecido um percurso definido por **n** e as chaves dos nós de cada nível, **V$** (i) (i = 1,2,...n), assim como o campo adicional de cada nó, **A$** (i) (i = 1,2,...n) e o número de registo base **u,** ao qual o percurso serve de chave.

Na entrada é dado o parâmetro **UPD,** como seguinte significado:

UPD = 0 — não é alterado o campo adicional dos nós já existentes, assim como o número do registo base se o nó terminal já existir.

UPD = 1 — são alterados os campos adicionais, mas não o número de registo base, dos nós existentes.

UPD = 2 — é alterado o número do registo base, se o nó terminal existir, mas não os campos adicionais.

UPD = 3 — são alterados os campos adicionais e o número do registo base.

O algoritmo segue os seguintes passos:

1) — Identifica-se o sub-percurso, do percurso actual, comum o percurso dado.

Os campos adicinais dos nós existentes são alterados de acordo com os valores de **A$** se **UPD** for 1 ou 3.

2) — Se todo o percurso estiver contido no percurso actual, termina-se após a alteração do número do registo base, se UPD for 2 ou 3.

3) —

3.1) Se o último nó comum não tiver sucessores actualiza-se a informação desse nó, de forma a endereçar como sucessor o 1.º nó não comum, que irá ocupar o 1.º registo livre.

3.2) No caso oposto a 3.1, procura-se na cadeia de nós endereçada pelo último nó comum a posição onde inserir o 1.º nó não comum e introduz-se esse nó na cadeia, ocupando para o efeito o 1.º registo livre. A lógica é a habitual na inserção em cadeias: o nó precedente passa a apontar o nó inserido e este fica a apontar o que era referenciado pelo precedente. Há que atender ao caso particular do nó ocupar a 1.ª posição.

4) Se o número de nós não comuns for superior a 1, serão introduzidos sucessivamente como sucessores do anterior.

Fisicamente ocuparão o registo **f,** sendo este valor simultaneamente actualizado.

A localização vai sendo actualizada à medida que as operações se desenvolvem.

O parâmetro de saída **C** indica as situações especiais encontradas:

C = 0 — caso normal em que se verificou adição de nós.

C = 1 — não houve adição de nós e o nó final não é terminal.

C = 2 — o mesmo que **C** = 1, mas o nó final é nó terminal.

C = – 1 — o mesmo que **C** = 1, mas **UPD** especifica alteração do registo base o que não é possível, dado o nó final não ser terminal.

C = – 2 — O percurso dado é um sub-percurso da localização actual.

C = – 100 — as operações da adição não foram concluídas devida à não existência de registos livres, há que realizar o fecho seguido da abertura, com **add** = 0 e, repetir a operação.

Notar que a função de actualização não permite alterar as chave dos nós existentes, pelo que é essencial esta função para que seja pos sível a eliminação de nós na árvore.

A eliminação de um nó acarreta a eliminação da sub-árvore que ten esse nó como raíz, como mostra o exemplo apresentado na Fig. 3

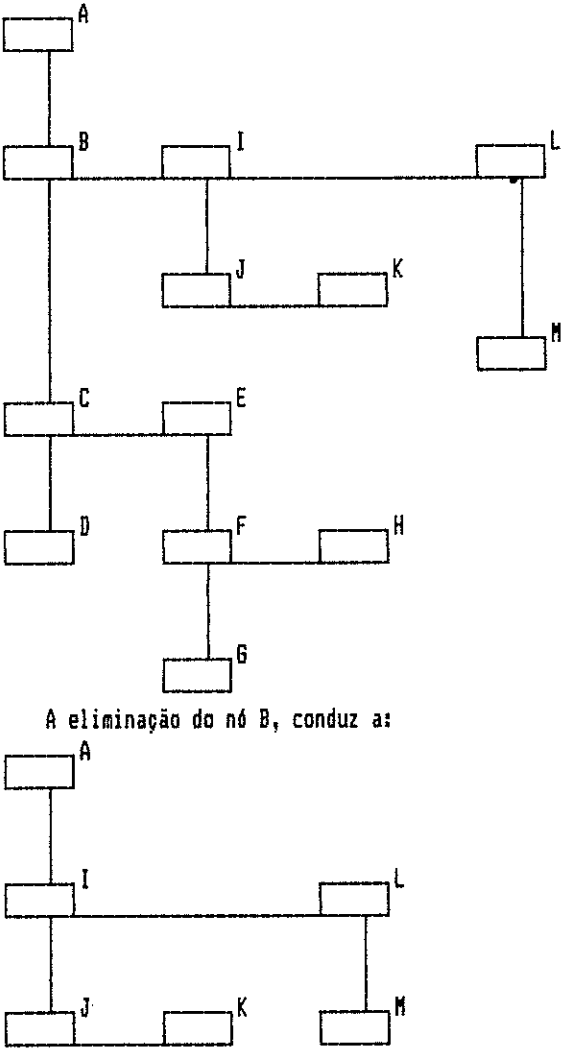


Fig. 3

Ao subprograma é fornecido o percurso definido por **n, a (i)** (i = 1,2,...m).

Pressupõe-se que o nó a eliminar é o que ocupa o registo **a (m).**

A lógica seguida é a seguinte:

1) Viaja-se na árvore até identificar o nó precedente a **a (m)**, ou o nó antecessor no caso de **a (m)** ser o 1.º da cadeia do nível **m.**

2) Actualiza-se a informação desse nó de forma a referenciar o seguinte, tendo em conta o caso particular de **a (m)** ser o último da cadeia.

Se **a (m)** for único na cadeia o nó encontrado em 1) passa a ser terminal pelo que o endereço **b** é feito igual a zero.

3) Acede-se sequencialmente aos nós, da sub-árvore que tem **a (m)** como raíz e, faz-se a ligação dos registos encontrados à cadeia dos registos livres.

A localização final é o percurso anterior truncado pelo nó **a (m).**

FECHO

Reescreve-se o primeiro registo com a informação de **p** e **f** actualizada.

Notar que esta operação é essencial para um correcto processamento futuro.

Como observação final dir-se-á que o sistema desenvolvido não é isento de crítica, especialmente no que respeita à eficiência no acesso.

A estrutura arborecente foi apresentada e desenvolvida como tratando-se de uma árvore-n, isto é, cada nó pode possuir até **n** sucessores.

A mesma árvore pode interpretar-se como árvore binária: em cada nó, o sucessor passa a ser o sucessor à esquerda e o nó seguinte passa a ser o sucessor à direita.

É conhecido da teoria das árvores binárias que a eficiência pode degradar-se no caso de a árvore não ser equilibrada, isto é, quando o nível máximo não é o menor possível.

Na construção da árvore, no caso presente, este aspecto não foi tido em conta, pelo que, em certos casos práticos, o sistema pode não dar o resultado desejável.

Este problema será função dos objectivos que se pretende com a aplicação. Por exemplo, voltando ao problema do xadrez, referido inicialmente:

O número médio de níveis na árvore será da ordem dos 25, em cada cadeia de nós sucessores haverá em média 4 nós. A localização de um percurso acarreta um número de leituras de registos, em média, da ordem de 25 x 4 / 2 = 50 (notar que na cadeia de sucessores a média de pesquisas é metade do número de nós). Parece exagerado, no entanto atenda-se a que, em geral, as localizações são efectuadas sequencialmente, pelo que somente a 1.ª ocupará um tempo apreciável.

BIBLIOGRAFIA

(1) *Knuth* - "The Art of Computer Programming" - Vol. 1; Ed. Addison - Wesley.

(2) *Tremblay, Bunt* — "An Introduction to Computer Science. An Algorithmic Approach"; Ed. McGraw — Hill.

A B E R T U R A

```
Dados: add ,    { se ≠ 0 significa que se pretende aumentar a extensão
                  física do ficheiro índice }
       ext ,    { se add ≠ 0 indica o número mínimo de registos livres }
       'FILE' ;

Início: Abrir 'FILE': relativo ;
        Ler 'FILE' reg.num. 1 : p, f ;
        m:=0 , LSTA:=0 ,
        Se add=0 então: fim ; senão: ;
        s:=f ,                    { averigua número de registos livres }
        Para i=1,2,... enquanto s>0 fazer:
          | q:=s ,
          | Ler 'FILE' reg.num. q : i ;
          L ;
        Se i>ext então: fim ;
                senão: i:=q+1 ,
                       Escrever 'FILE' reg.num. q : i ;
                       Fechar 'FILE' ,
                       Abrir 'FILE': consecutivo , encosto ;  { append }
                       s:=q ,
                       Para j=1,2,...ext-1 fazer:
                         | s:=s+1 ,
                         | Escrever 'FILE': s ;
                         L ;
                       Escrever 'FILE': 0 ;
                       Fechar 'FILE' ,
                       Abrir 'FILE': relativo ;
                       fim ;

Resultado: p , f , LSTA , m , 'FILE'
```

L O C A L I Z A Ç Ã O

```
Dados: n ,     { número de nós do percurso que se pretende localizar }
       V$(i) (i=1,2,...n) , { chaves dos nós }
       A$(i) (i=1,2,...n) , { campo adicional dos nós }
       m , T$(j) , I$(j) , a(j) (j=1,2,...m) ,
       p , 'FILE' ;
```

```
Início: C:=0 , LSTA:=1 , e:=0 ,
       Se m>0 então: j:=0 ,
                     Para i=1,2,...m enquanto T$(i)=V$(i) e i<n fazer:
                      └ j:=i ;
                     m:=j ;
              senão: ;
       Se m=0 então: s:=p ;
              senão: Ler 'FILE' reg.num. a(m): b, r, W$, D$ ;
                     s:=b ;
       Se m=n então: Se s>0 então: LSTA:=-1 ; senão: e:=-s ;
                     fim ;
              senão: Se s<0 então: C:=1 , e:=-s , fim ; senão: ; ;
       Para j=m+1,m+2,...n fazer:
        │ t:=s ,
        │ Ler 'FILE' reg.num. s: b, r, W$, D$ ;
        │ s:=r ,
        │ Para k=1,2,... enquanto W$<V$(j) e r>0 fazer:
        │  │ t:=s ,
        │  │ Ler 'FILE' reg.num. s: b, r, W$, D$ ;
        │  └ s:=r ;
        │ Se W$=V$(j) então: m:=j , a(m):=t , T$(m):=W$ , I$(m):=D$ ,
        │                    Se b<0 então: Se m<n então: C:=1 ; senão: ;
        │                                        e:=-b , fim ;
        │                                  senão: Se m=n então: LSTA:=-1 , fim ;
        │                                                senão: s:=b ;
        │                                  ;
        │                    senão: C:=1 , LSTA:=-1 , fim ;
        └ ;

Resultado: m , T$(i) , I$(i) , a(i) (i=1,2,...m) , LSTA
           e ,   { endereço base, no caso em que LSTA=1 }
           C ;   { indicador da situação encontrada }
```

L E I T U R A S E Q U E N C I A L

```
Dados: m , T$(j) , I$(j) , a(j) (j=1,2,...m) ,
       p , LSTA , 'FILE' ;

Início: C:=0 ,
       Se m=0 então: Se p=0 então: C:=1 , e:=0 , LSTA:=0 , fim ;
                              senão: s:=p , m:=1 ;
              senão: Ler 'FILE' reg.num. a(m) : b, r, W$, D$ ;
```

```
                    Se LSTA=-1
                      então: s:=b , m:=m+1 ;
                      senão: Se r>0 então: s:=r ;
                                   senão: s:=-r ,
                                          Para i=1,2,... enquanto r<0 faz
                                            | m:=m-1 ,
                                            | Ler 'FILE' reg.num. s :
                                            |     b, r, W$, D$ ;
                                            L  s:=-r ;
                                          s:=r ;
          Se s=0 então: C:=1 , e:=0 , LSTA:=0 , fim ;
                 senão: Para i=1,2,... enquanto s>0 fazer:
                          | Ler 'FILE' reg.num. s : b, r, W$, D$ ;
                          | T$(m):=W$ , I$(m):=D$ , a(m):=s ,
                          | m:=m+1 ,
                          L s:=b ;
                        e:=-s , m:=m-1 , LSTA:=i ,
                        fim ;

Resultado: m , T$(j) , I$(j) , a(j) (j=1,2,...m) ,
           LSTA , e ,
           C ;   { indicador da situação encontrada }

A C T U A L I Z A Ç Ã O

Dados: n ,    { número de nós na chave a introduzir ou alterar }
       V$(i) (i=1,2,...n)   { chaves dos nós }
       A$(i) (i=1,2,...n)   { campo adicional dos nós }
       u      { endereço base a introduzir no nó terminal }
       UPD ,  { indicador }
       m , T$(j) , I$(j) , a(j) (j=1,2,...m) , e ,
       p , f , 'FILE' ;
Inicio: C:=0 ,
        Se m>0
          então: i:=0 ,       { viagem pelos nós comuns ao percurso definido
                                pelos dados e o definido pela localização }
                 Para k=1,2,...m enquanto i<n e T$(k)=V$(k) fazer:
                   | i:=k ,
                   | Se I$(k)==A$(k) e (UPD=1 ou UPD=3)
                   |   então: Ler 'FILE' reg.num. a(k) : b, r, W$, D$ ;
                   |            Escrever 'FILE' reg.num. a(k): b, r, W$, A$(k) ;
                   |            I$(k):=A$(k) ;
                   |   senão: ;
                   L;
```

```
                Se i=n      { todo o percurso está incluido na localização }
                  então: Se UPD<2
                           então: C:=1 , fim ;
                           senão: Ler 'FILE' reg.num. a(i): b, r, W$, D$ ;
                                  Se b<0 então: Escrever 'FILE' reg.num. a(i):
                                                -u, r, W$, D$ ;
                                                C:=1 , fim ;
                                         senão: C:=-1 , fim ;
                                  ;
                  senão: m:=i ;
          senão: ;

{ Averigua-se a existência prévia de nós do percurso dado e actualiza-se a in-
  formação do último nó já existente }

      Se m=0 então: Se p>0 então: s:=p ; senão: p:=f , s:=0 ;
            senão: Ler 'FILE' reg.num. a(m): b, r, W$, D$ ;
                   Se b>0 então: s:=b ;
                          senão: C:=2 , s:=0 , Escrever 'FILE' reg.num. a(m):
                                 f, r, W$, D$ ; ;
                   ;

      Se s>0
        então: E:=0 , t:=0 ,
               Para i=1,2,... enquanto E=0 fazer:
                 | Ler 'FILE' reg.num. s: b, r, W$, D$ ;
                 | Se V$(m+1)<W$
                 |   então: E:=1 ;
                 |   senão: Se V$(m+1)>W$
                 |             então: ba:=b , t:=s , WA$:=W$ , DA$:=D$ ,
                 |                    Se r≤0 então: E:=3 ; senão: ;
                 |                    ;
                 |             senão: m:=m+1 , T$(m):=W$ , a(m):=s ,
                 |                    Se UPD=1 ou UPD=3
                 |                      então: Escrever 'FILE' reg.num. s:
                 |                             b, r, W$, A$(m) ;
                 |                             I$(m):=A$(m) ;
                 |                      senão: I$(m):=D$ ;
                 |                     Se m=n
                 |                       então: Se UPD<2
                 |                                then: Se b>0 então: C:=1 ;
                 |                                                    senão: C:=2 ,
                 |                                                           e:=-b ;
                 |                                      senão: Se b>0 então: C:=-1 ;
                 |                                                    senão: C:=1 ;
```

```
                                        Escrever 'FILE' reg.num. s
                                         -u, r, W$, I$(m) ;
                                         e:=-u ;
                                   fim ;
                              senão: Se b<0 então: E:=2 , C:=-2 ,
                                                   e:=-b ;
                                     senão: t:=s , s:=b ;
                                   ;
                             ;
                  ;
            Se ((E=1 e t=0) ou E=2) e m>0
              então: Ler 'FILE' reg.num. a(m): b, r, W$, D$ ;
                     Escrever 'FILE' reg.num. a(m): f, r, W$, d$ ;
                     ;
              senão: ;
            Se E==2 e t>0
              então: Escrever 'FILE' reg.num. t: ba, f, WA$, DA$ ; ;
              senão: ;
            Ler 'FILE' reg.num. f: fs ;
            Se E==2
              então: Se E=1 então: r:=s ; senão: ;
                     m:=m+1 , T$(m):=V$(m) , I$(m):=A$(m) , a(m):=f ,
                     Se m=n então: Se UPD>1 então: b:=-u ; senão: b:=0 ;
                                   e:=-b ;
                           senão: b:=fs ;
                     Escrever 'FILE' reg.num. f: b, r, V$(m), A$(m) ;
                     f:=fs ;
              senão: ;
      senão: ;
{ Introdução dos nós que não existiam previamente na estrutura }
    Se m=0 então: r:=0 ; senão: r:=-a(m) ;
    Para j=m+1,m+2,...n fazer:
      | Ler 'FILE' reg.num. f: fs ;
      | Se j=n então: Se UPD>1 então: b:=-u ; senão: b:=0 ;
      |       senão: b:=fs ;
      | Escrever 'FILE' reg.num. f: b, r, V$(j), A$(j) ;
      | T$(j):=V$(j) , I$(j):=A$(j) , a(j):=f ,
      | r:=-f ,
      | Se fs=0 então: C:=-100 , m:=j , fim ; senão: ;
      L f:=fs ;
    m:=n , LSTA:=1 , fim ;

Resultado: m , T$(i) , I$(i) , a(i) (i=1,2,...m) , e ,
           p , f , LSTA , 'FILE'
           C   { indicador da situação encontrada }
```

C O R T E

```
Dados: m , a(i) (i=1,2,...m) ,
       p , f , LSTA , 'FILE' , UPD ;


Início: Se LSTA=0 então: fim ; senão: ;
        Se m=0 então: Se p=0 então: LSTA:=0 , fim ; senão: s:=p , p:=0 ;
 { Procura-se o nó precedente do nó a ser eliminado a(m) }
              senão: t:=a(m) ,
                     Ler 'FILE' reg.num. t: b, r, W$, D$ ;
                     rr:=r ,   { guarda-se o endereço do nó seguinte a a(m) }
                     Para i=1,2,... enquanto r>0 fazer:
                      | Ler 'FILE' reg.num. r: b, r, W$, D$ ;
                      L ;
                     q:=-r ,                { q é o antecessor de a(m) }
                     Se q=0 então: k:=p ;
                           senão: Ler 'FILE' reg.num. q: b, r, W$, D$ ;
                                  k:=b ;
                     j:=0 ,
                     Para i=1,2,... enquanto k≠t fazer:
                      | kk:=k ,
                      | Ler 'FILE' reg.num. k: b, r, W$, D$ ;
                      | j:=i ,
                      L k:=r ;
                     Se j=0  { se j=0 , a(m) é o único descendente de q }
                       então: Se k=p então: p:=rr , LSTA:=0 ;
                                    senão: Se rr<0 então: Se UPD<2
                                                            então: i:=0 ;
                                                            senão: i:=-u ,
                                                                   e:=i ;
                                                          ;
                                                   senão: i:=rr ;
                                           Escrever 'FILE' reg.num. q:
                                            i, r, W$, D$ ;
                                           LSTA:=1 ;
                                  ;
                       senão: Escrever 'FILE' reg,num. kk: b, rr, W$, D$ ;
                              LSTA:=-1 ;
                     Escrever 'FILE' reg.num. t: f ;
                     f:=t ,
                     Se s<0 então: fim ; senão: ;
                     ;
      mm:=m+1 ,
{ os nós da sub-árvore de raiz a(m) são ligados à cadeia dos nós livres }
```

```
        Para i=1,2,... enquanto m≠mm fazer:
          | Ler 'FILE' reg.num. s: b, r, W$, D$ ;
          | Se b>0 então: mm:=mm+1 , s:=b ;
          |       senão: Escrever 'FILE' reg.num. s: f ;
          |               f:=s ,
          |               Se r>0
          |                 então: s:=r ;
          |                 senão: s:=-r ,
          |                        Se s=0 então: fim ; senão: ;
          |                        Para j=1,2,... enquanto r<0 e m≠mm fazer:
          |                          | Ler 'FILE' reg.num. s: b, r, W$, D$ ;
          |                          | Escrever 'FILE' reg.num. s: f ;
          |                          | f:=s , mm:=mm-1 ,
          |                          L s:=-r ;
          |                        s:=r ;
          L ;
        m:=m-1 , fim ;

Resultado: m , p , f , LSTA , e , 'FILE' ;
```

```
F E C H O

Dados: p , f , 'FILE' ;

Início: Escrever 'FILE' reg.num. 1: p, f ;
        Fechar 'FILE';
        fim ;

Resultado: 'FILE' ;
```