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 Nº 17.891/87

Nº VI / Fevereiro de 1990

Scientia lucet omnibus

Com a presente edição, *"Educação e Tecnologia"* entrou no terceiro ano de existência e, simultaneamente, na década de noventa.

Publicação que tem acompanhado e reflectido o crescimento, progressivo, do Instituto Politécnico da Guarda, esta Revista é já hoje a certeza de um desafio ganho em termos editoriais, científicos, pedagógicos e culturais.

Integrada numa das várias vertentes da acção do Instituto Politécnico, *"Educação e Tecnologia"* tem-se afirmado como pólo aglutinador de múltiplas participações e colaborações, algumas oriundas de estabelecimentos de ensino superior inseridos no quadro da cooperação interuniversitária europeia.

Entendemos que este projecto é bem o símbolo da abertura às realidades hodiernas e "forum" de um diálogo multifacetado sob a trave mestra deste Instituto: *"Scientia lucet omnibus"*.

Aliás, as modificações resultantes de toda uma dinâmica ao nível económico e social, que se vêm registando no distrito, têm merecido uma particular atenção ao Instituto Politécnico da Guarda.

Como exemplo podemos referir a proposta, já apresentada oficialmente, de novos cursos — de que a região carece — para o próximo ano lectivo, cursos que se vêm juntar ao leque dos já existentes. Por outro lado, há todo um trabalho de organização e implementação de projectos subjacentes às duas Escolas Superiores que integram o I.P.G..

Factor de desenvolvimento regional, o Instituto Politécnico da Guarda tem nesta publicação um alicerce seguro de um vasto trabalho de informação, divulgação e reflexão.

João Bento Raimundo
Presidente da C. I. do
Instituto Politécnico da Guarda

O ACESSO SEQUENCIAL INDEXADO E O PROCESSAMENTO EM REDE

Álvaro Bento Leal*

RESUMO: É apresentado um conjunto de subprogramas em QuickBasic que gerem o acesso e organização duma estrutura em árvore - B adaptados ao acesso sequencial indexado em ambiente multi-tarefa.

No nº 3 desta revista, o autor apresentou uma forma de organização de dados adequada ao acesso sequencial indexado. A caracteristica salientada na altura era a simplicidade e a eficiência resultante da possibilidade de se recorrer extensivamente à memória real. Ver-se-á agora que a eliminação de alguns dos aspectos restritivos da estrutura, uma lista, não é possível duma forma satisfatória.

As principais limitações do referido modelo estão relacionadas com:

1) - A chave de pesquisa tem de ser única no índice.
2) - Num mesmo programa só é possível utilizar um ficheiro sequencial indexado.
3) - Os aspectos do acesso concorrencial em ambiente multi-tarefa não são contemplados.

As restrições 1) e 2) podem ser removidas sem problemas especiais no que respeita à concepção do modelo. A limitação 3) é crítica,não sendo possível eliminá-la duma forma prática. As razões são as seguintes:

Se duas tarefas concorrenciais realizam adições ao mesmo ficheiro e a reorganização é efectuada no final da tarefa, a última a ser concluída desconhece as adições da primeira.

* - Prof. Coordenador
Director do Departamento de Informática da ESTG

Pode argumentar-se que se em vez de a reorganização partir do ficheiro em memória virtual, isto é, na situação do início da tarefa, fizer a leitura da versão mais actualizada armazenada em disco e a partir daí executar a reorganização, parece não haver problema de maior. Tal não é exacto devido ao facto de no aspecto funcional surgirem situações de solução difícil. Assim uma chave nova, do ponto de vista da tarefa, pode entretanto ter sido introduzida pela tarefa concorrencial, ignorando-se mesmo, a menos que o tempo seja registado, qual a introdução que ocorreu em primeiro lugar. Também poderia acontecer surgirem eliminações de registos que a outra tarefa actualizou no pressuposto que a chave era válida, etc..

Resumindo, no modelo baseado numa lista simples é difícil reproduzir o fluxo de operações de alteração do ficheiro indexado com o rigor cronológico indispensável, assim como transmitir entre os utilizadores as informações que podem influenciar as decisões no momento do processamento.

Chama-se a atenção para o aspecto focado, na medida em que está integrado num problema mais amplo e que é o facto de, frequentemente, a passagem das aplicações mono-tarefa para ambiente multi-tarefa não exigir unicamente o "hardware" e "software" de rede adequado, mas obrigar a alterações, por vezes drásticas, do "software" de aplicação.

O processamento em rede é de utilização cada vez mais frequente, pelo que há todo o interesse em dispor de um método sequencial indexado adequado.

É óbvio que o problema está resolvido ao nível do "software" existente no mercado mas, do ponto de vista do programador , um sistema escrito numa linguagem fonte que lhe seja familiar, apresenta vantagens, dadas as possibilidades de adaptação e optimização que proporciona em cada aplicação concreta. Esta é a razão de se ter decidido a publicação do presente artigo.

A árvore - B

Do que foi dito atrás conclui-se que o acesso sequencial indexado deve basear-se numa estrutura de dados que se mantenha totalmente organizada em disco após qualquer operação de acesso. Das numerosas estruturas que podem garantir esta condição a árvore - B revela, na maioria dos casos, uma eficiência que a torna recomendável.

Duma forma sucinta recordam-se aqui as principais características deste tipo de estrutura:

— As chaves e endereços são agrupados em blocos com capacidade máxima fixa de M chaves. Cada bloco é armazenado fisicamente como um registo do ficheiro índice.

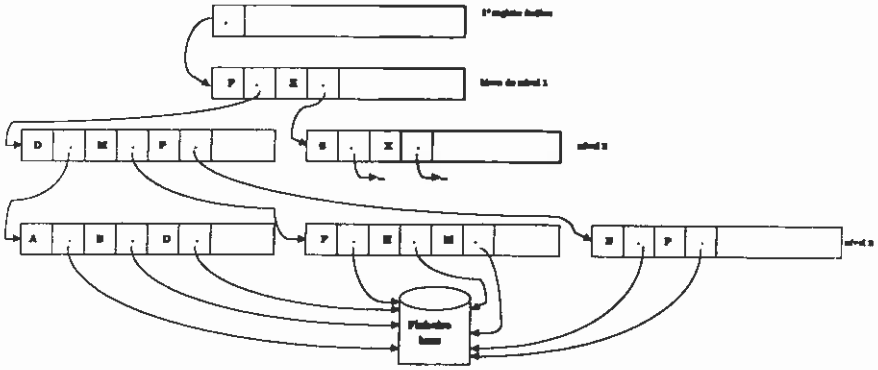
— O 1º bloco contém o endereço (número relativo do registo índice) de um bloco dito de nível 1.

— O bloco de nível 1 contém de 1 a M endereços de blocos de nível 2, associados com as correspondentes maiores chaves desses blocos. Os pares chave/endereço estão ordenados pelo valor da chave.

— Os blocos de nível 2 referenciam pelo mesmo método os blocos de nível 3 e assim sucessivamente. Nos blocos do último nível, em vez dos endereços dos blocos, são registados os números dos registos do ficheiro base em correspondência com as chaves.

— O número de pares chave/endereço é variável em cada bloco mas, exceptuando o bloco de nível 1, a árvore é reorganizada, quando necessário, de forma a que cada bloco contenha pelo menos M/2 elementos.

— O número de níveis existente é variável. O último nível, no caso presente, é detectado pelo facto de os endereços serem números negativos com o mesmo valor absoluto dos registos do ficheiro base que referenciam.



O algoritmo da pesquisa de um elemento na árvore consiste no acesso, do nível 1 ao nível máximo, seguindo os blocos endereçados pelas chaves iguais ou imediatamente superiores à chave dada. Só atingido o último nível é possível averiguar a existência da chave.

O bloco do último nível é reorganizado após a inclusão, se se trata duma adição, ou exclusão, no caso duma eliminação, da chave e endereço base.

No caso da inserção de uma chave maior que todas as existentes, é necessário viajar em sentido ascendente na árvore e alterar o valor da chave de referência do bloco em causa.

Pode acontecer, quando de uma inserção, encontrar-se o bloco do último nível totalmente preenchido. Neste caso há que recorrer a um bloco livre e distribuir os elementos em partes iguais pelos dois blocos, proceder à inserção e, no nível anterior, actualizar a chave de referência do bloco em causa e introduzir a chave e endereço do novo bloco criado. Este procedimento é

aplicado a todos os níveis precedentes enquanto a condição de blocos cheios for encontrada. No subprograma que realiza esta função, e que é apresentado no final, é utilizada a técnica de recursividade no procedimento da viagem em sentido inverso. O caso particular do bloco de nível 1 completo ocasiona a criação de um novo bloco de nível 1, aumentando o nível de todos os blocos existentes de uma unidade.

Mais complexo é o algoritmo de eliminação quando surge a situação de o bloco onde ela se processa ficar com menos de metade da capacidade preenchida.

Neste caso há que distribuir as chaves e endereços do bloco onde a eliminação se processou, pelos blocos vizinhos, isto é, com localizações à esquerda e à direita do bloco em causa no nível imediatamente anterior, passando o bloco em causa à situação de livre. No subprograma da listagem apresentada adiante, utilizaram-se os seguintes critérios:

Seja *me* o número de elementos no bloco vizinho à esquerda, *md* o mesmo para o bloco vizinho à direita e *ma* o número de elementos do bloco em causa menos 1.

1. - Se $me = 0$ e $md = 0$, está-se no bloco de nível 1, a chave é removida. Se o bloco ficar vazio, faz-se a referência do bloco de nível 1 igual a 0.
2. - Se $ma \geq M/2$, o elemento é eliminado no bloco. Se a chave eliminada ocupava a maior posição no bloco, viaja-se na árvore, no sentido ascendente, alterando a chave de referência do bloco em causa.
3. - Se $ma < M/2$, faz-se $nn = ma + mc + md$
   3.1. - Se $md = 0$
      3.1.1. - Se $nn < M$, junta-se a informação do bloco à esquerda com a do bloco em causa, sem a chave dada, neste bloco, elimina-se o bloco à esquerda e, recorrendo à recursividade, a referência deste bloco no nível anterior.
         3.1.1.1. - No caso particular de o bloco anterior ficar reduzido a um elemento, esse bloco é obrigatoriamente o de nível 1. Então passa a ser de nível 1 o bloco em causa e o anterior passa a livre, o nível de todos os blocos é reduzido de uma unidade.
      3.1.2. - Se $nn > M$, os elementos do bloco em causa, após a remoção da chave dada, e os do bloco à esquerda são distribuídos em partes iguais pelos dois blocos e actualizam-se as chaves de referência no nível anterior.
   3.2. - Se $me = 0$. O procedimento é análogo a 3.1., mas em

vez de se usar o bloco à esquerda, usa-se o bloco à direita.

3.3. - Se *me* > 0 e *md* > 0

3.3.1. - Se *nn* < 2M, os elementos do bloco à esquerda, do bloco à direita e do bloco em causa, são distribuídos igualmente pelos dois blocos vizinhos, passando a bloco livre o bloco em causa. As alterações que a operação ocasiona são efectuadas nos níveis anteriores.

3.3.2. - Se *nn* ≥ 2M, os elementos do bloco e blocos vizinhos são redistribuídos em partes iguais pelos 3 blocos e procede-se à rectificação das chaves nos níveis anteriores.

O algoritmo adoptado garante que, exceptuando o bloco de nível 1, todos os restantes possuem de M/2 a M elementos, pelo que para um nível máximo $m > 1$ é garantido o armazenamento de:

$$2 \times M/2 \times M/2 \ldots M/2 = 2\,(M/2)^{m-1} \text{ chaves.}$$

Note-se que as operações de adição, no caso de a condição de blocos cheios e, as de eliminação, nos casos em que os blocos estão pouco preenchidos, desencadeiam um grande número de operações em cascata, com a inevitável degradação da eficiência. O que há a fazer é procurar que aquelas condições sejam pouco frequentes, o que se consegue utilizando um valor M grande.

Um valor de M que provoca o preenchimento de mais de um "buffer" na leitura de um registo de índice ocasiona uma excessiva actividade da unidade de leitura. O ideal é fixar M de forma que o bloco se aproxime do comprimento do valor do "buffer" definido quando do arranque do sistema operativo.

Por exemplo: Se o buffer = 512 e o comprimento da chave = = 10, será M = 512 \ (10 + 4) = 36, se o nível máximo for 4, o ficheiro garante no mínimo a capacidade de $2 \times 18^3 = 11664$ chaves, crescendo a capacidade por um factor de 18 quando o nível aumenta de 1.

Para evitar desperdício do espaço em disco com os blocos que são removidos, adoptou-se manter os blocos livres encadeados, armazenando em cada bloco o endereço do bloco livre que lhe segue na cadeia e o endereço do 1º livre no 1º registo do ficheiro índice. Quando há necessidade de um bloco livre, recorre-se ao 1º da cadeia, quando um bloco é libertado, encadeia-se com o 1º livre, passando a realizar essa função. Se for necessário um bloco livre e a cadeia estiver vazia, acrescenta-se o ficheiro com um número fixo de novos registos livres.

Chaves em Duplicado

O armazenamento de múltiplas ocorrências da mesma chave tem uma solução trivial que consiste em considerar como chave do índice o valor da chave concatenado à direita com o número do registo base a que diz respeito. No entanto este procedimento conduz a uma ocupação de espaço que pode ser evitada.

No caso presente as chaves duplicadas podem existir, tendo havido unicamente o cuidado de no último nível os elementos com chaves iguais serem mantidos ordenados pelo número do registo base respectivo. No entanto algumas providências tiveram de ser tomadas no que respeita ao algoritmo de pesquisa.

Assim no caso de a chave ocupar a última posição do bloco, na pesquisa por ordem crescente, ou a primeira no caso da pesquisa por ordem decrescente, há que viajar até ao bloco vizinho à direita, ou à esquerda, para aí prosseguir a pesquisa.

Se o grau de duplicação for elevado, pode haver necessidade de um grande número de viagens ascendentes seguidas de descidas na árvore, pelo que a eficiência pode ressentir-se desse facto.

Concorrência de Tarefas

A principal consequência, do ponto de vista algorítmico, que a concorrência de acesso de utilizadores distintos ao mesmo ficheiro índice acarreta, é o facto de a informação sobre o posicionamento anterior na árvore não poder ser usada, uma vez que a estrutura pode ter sofrido alteração entre dois acessos sucessivos.

Pela mesma razão o controlo exclusivo (LOCK e UNLOCK) não pode ser solicitado ao nível do registo do ficheiro índice, mas sim ao nível de todo o ficheiro.

No caso presente, dado a leitura sequencial não poder basear-se no posicionamento prévio, optou-se por, em cada operação de acesso, armazenar em memória a chave e número de registo base acedido, iniciando a leitura sequencial pela pesquisa dos valores guardados, e somente depois passar à chave que segue na ordem.

O texto dos subprogramas em QuickBasic que se apresenta a seguir pode adaptar-se à utilização em ambiente de tarefa exclusiva, evita a utilização do comando SHARE do MS/DOS. Para tal é suficiente retirar o parâmetro SHARE no OPEN do subprograma ISOpen e todas as instruções LOCK e UNLOCK em todos os subprogramas.

MÉTODO DE ACESSO SEQUENCIAL INDEXADO

Conjunto de subprogramas que realizam as funções de manutenção e consulta de um, ou mais, ficheiros índice estruturados como árvores-B.

Cada fich. índice controla a chave (cadeia de caracteres) e a posição dos registos de um fich. base, mas nada impede que o mesmo fich. base possua vários índices.

Para cada índice o comprimento das chaves único.

São permitidas chaves em duplicado desde que as respectivas posições no fich. base sejam distintas.

Os subprogramas estão preparados para ser usados simultaneamente em ambiente multi-tarefa (em rede). Uma versão para uso exclusivo (evita a utilização do comando SHARE) pode obter-se eliminando o parâmetro SHARE da instrução OPEN da subrotina ISopen e as instruções LOCK e UNLOCK em todos os subprogramas.

Cada elemento da árvore ocupa um registo no fich. índice com informação relativa a um máximo de M elementos da mesma árvore, excepto no último nível onde a informação diz respeito às posições no ficheiro base. Cada elemento é formado por uma chave e um endereço do tipo LONG INTEGER, isto é, ocupa o comprimento da chave mais 4 bytes. Por razões de eficiência o valor de M não deve ser pequeno (16 no mínimo). Um bom critério é adoptar para comprimento dos registos índice o valor do buffer definido no DOS.

O nível máximo MAXL da árvore-B depende do número de chaves com que se lida, uma estimativa pode obter-se sabendo que para um dado valor de MAXL é garantido que no mínimo a árvore suporta M*(M/2)^(MAXL-2) chaves.

FUNÇÕES :

- Abertura do fich. índice

```
CALL ISopen(t%, nf%, nam$, cbuf%, ck%)
```

t% (entrada): >= 0 e < MAXF (núm. máx. de fich. ind.) referencia o fich. índice
nf% (entrada): Núm. do fich. QUICKBASIC (buffer) a ser usado pelo índice t%. Se = 0, o valor a ser usado é o devolvido pela instrução FREEFILE
nam$ (entrada): Nome DOS do ficheiro índice t%
cbuf% (entrada): Comprimento dos registos do ficheiro índice t%. É obrigatório (cbuf% \ (ck%+4)) > 3. São recomendados os valores cbuf%=256 ou 512
ck% (entrada): Comprimento das chaves

É obrigatória a chamada de ISopen antes da utilização de qualquer função relativa ao ficheiro t%. Normalmente ISopen com um dado número nf% só é chamada uma vez no programa, se houver necessidade de uma segunda chamada deve executar-se previamente CLOSE nf%

```
.....
```

- Adição de novas chaves ao fich. índice

```
ISadd%(t%, k$, a&, p%)
```

t% (entrada): Referência do fich. índice
k$ (entrada): Chave que se pretende adicionar
a& (entrada): Posição do registo do fich. base a que se refere a chave k$
p% (entrada): Se = 0, a chave só é introduzida se não existir previamente uma chave igual no índice. Se <> 0, a in-

trodução é efectuada se as chaves iguais existente estiverem associadas a registos base diferentes
ISadd% (saída): = 0, se a adição foi efectuada. = 1 no caso contrário

.....

- Eliminação de chaves no fich. índice

ISdel%(t%, k$, a&)

t% (entrada): Referência do fich. índice
k$ (entrada): Chave que se pretende eliminar
a& (entrada): Posição no registo base associada com k$, obrigatória quer haja ou não chaves duplicadas
ISdel% (saída): = 0, se a eliminação foi efectuada, isto é, a chave com a respectiva posição no fich. base foi encontrada. = 1, no caso contrário

.....

- Pesquisa por chave

ISfind%(t%, k$, a&)

t% (entrada): Referência do fich. índice
k$ (entrada): Chave que se pretende encontrar
a& (saída): Posição do registo no fich. base. Se existirem chaves k$ no índice, a& é devolvido com o valor relativo ao registo base com menor número e com chave k$. No caso oposto a& contem a posição do registo base de menor núm. com chave imediatamente superior a k$. a& = 0, se todas as chaves existentes forem menores que k$
ISfind% (saída): = 0, se no índice existir a chave k$. = 1, no caso oposto

ISfind% coloca a situação no índice de forma a que a pesquisa sequencial que se seguir devolve o registo que segue na ordem ao encontrado (ou o anterior no caso da leitura sequencial por ordem inversa)

.....

- Pesquisa sequencial por ordem crescente

ISnext%(t%, k$, a&)

t% (entrada): Referência do fich. índice
k$ (saída): Chave imediatamente seguinte no índice, à que foi obtida na pesquisa por chave, ou pesquisa sequencial, imediatamente anterior
a& (saída): Posição base correspondente à chave k$
ISnext% (saída): = 0, se foi encontrada uma chave. = 1 se não existir no índice chave seguinte relativamente à situação imediatamente anterior (fim do fich. índice)

ISnext% devolve a menor chave a seguir a uma chamada com resultado ISnext%=1, ou a seguir a ISopen, ou a seguir a ISfind% com resultado a&=0, ou ainda a seguir a ISprev% com resultado 1. No caso de chaves iguais a ordem é a dos núm. de registo do fich. base

.....

- Pesquisa sequencial por ordem decrescente

ISprev%(t%, k$, a&)

t% (entrada): Referência do fich. índice
k$ (saída): Chave imediatamente anterior no índice, à que foi obtida na pesquisa por chave, ou pesquisa sequencial por ordem crescente ou decrescente, imediatamente

```
                        anterior
          .a&       (saida): Posição base correspondente à chave k$
           ISprev% (saida): = 0, se foi encontrada uma chave. = 1 se não existir
                            no índice chave anterior relativamente à situação
                            imediatamente anterior (início do fich. índice)

           ISprev% devolve a maior chave a seguir a uma chamada com resultado
            ISnext%=1, ou a seguir a ISopen, ou a seguir a ISfind% com resultado
            a&=0, ou ainda a seguir a ISprev% com resultado 1
                    ..............

           Os subprogramas ISrmv, ISexp, ISins e ISpos% são auxiliares de execu-
            ção dos restantes
                    ..............

           Instruções a incluir no programa principal:

DECLARE SUB ISrmv (t%, c%)
DECLARE FUNCTION ISdel% (t%, k$, a&)
DECLARE FUNCTION ISprev% (t%, y$, a&)
DECLARE SUB ISexp (t%, r&)
DECLARE SUB ISins (t%, c%, k$, u&)
DECLARE FUNCTION ISadd% (t%, k$, a&, p%)
DECLARE FUNCTION ISnext% (t%, y$, a&)
DECLARE FUNCTION ISfind% (t%, k$, a&)
DECLARE FUNCTION ISpos% (t%, k$, u&, r$, m$, e&, a&, d&)
DECLARE SUB ISopen (t%, nf%, nam$, cbuf%, ck%)

CONST MAXF = 4                  ' Núm. máx. de fich. índice usados
CONST MAXL = 6                  ' Núm. máx. de níveis das árvores-B
CONST NADI = 5                  ' Núm. de registos com os fich. índice são acresci-
                                '  dos automaticamente, quando necessário (comum a
                                '  todos os fich. índice)

DIM nfi%(MAXF - 1), ck%(MAXF - 1), ct%(MAXF - 1), nti%(MAXF - 1), vbf$(MAXF - 1)
DIM kr$(MAXL), ae&(MAXL), aa&(MAXL), ad&(MAXL), ip%(MAXL), ri$(MAXL), ks$(MAXL), ps&(MAXL)

ON ERROR GOTO suberr

  A subrotina suberr tem como finalidade actuar quando da interrupção de erro
   (cod. 63 e 70) que ocorre pelo facto de o acesso ao fich. estar impedido pelo
   control exclusivo solicitado por outra tarefa. No caso de multi-tarefa esta
   subrotina é obrigatória.
  A forma aqui apresentada serve unicamente como exemplo:

suberr: LOCATE 24, 1: COLOR 0, 7: PRINT "AGUARDE UM MOMENTO"; : COLOR 7, 0
        Start! = TIMER
        DO WHILE TIMER - Start! < 3: LOOP
        LOCATE 24, 1: PRINT SPC(18);
        RESUME


FUNCTION ISadd% (t%, k$, a&, p%) STATIC
  SHARED nfi%(), ck%(), ct%(), nti%(), nv%, vbf$(), ri$(), rr$
  SHARED kr$(), ae&(), aa&(), ad&(), ip%()
  SHARED ks$(), ps&()
  LOCK nfi%(t%)
  GET nfi%(t%), 1: ri$(0) = vbf$(t%)
  aa&(0) = CVL(MID$(ri$(0), 5, 4))
  nv% = 0
  IF aa&(0) = 0 THEN
    aa&(0) = CVL(MID$(ri$(0), 1, 4))
    GET nfi%(t%), aa&(0): ri$(1) = vbf$(t%)
    MID$(ri$(0), 5, 4) = MID$(ri$(0), 1, 4)
    MID$(ri$(0), 1, 4) = MID$(ri$(1), 1, 4)
    LSET vbf$(t%) = ri$(0): PUT nfi%(t%), 1
```

```
    MID$(ri$(1), 1, ck%(t%)) = k$: MI      (1), ck%(t%) + 1, 4)   MKL$(-a&)
    LSET vbf$(t%) = ri$(1): PUT nfi%(t%), aa&(0)
    UNLOCK nfi%(t%)
    ISadd% = 0
    EXIT FUNCTION
  END IF
  kk$ = k$: ar& = -a&
  DO WHILE aa&(nv%) > 0
    nv% = nv% + 1
    GET nfi%(t%), aa&(nv% - 1): ri$(nv%) = vbf$(t%)
    IF CVL(MID$(ri$(nv%), ck%(t%) + 1, 4)) < 0 THEN zz& = -ar& ELSE zz& = 0
    i% = ISpos%(t%, k$, zz&, ri$(nv%), m$, ae&(nv%), aa&(nv%), ad&(nv%))
    IF i% < 0 THEN
      i% = -i%
      IF aa&(nv%) < 0 THEN
        ch% = nv%
        DO WHILE ad&(ch%) = 0 AND ch% > 0
          ch% = ch% - 1
        LOOP
        IF ch% = 0 THEN
          ip%(nv%) = i%
          FOR ch% = 1 TO nv%
            j% = ct%(t%) * (ip%(ch%) - 1) + 1
            MID$(ri$(ch%), j%, ck%(t%)) = k$
            IF aa&(ch%) < 0 THEN
              kk$ = m$: m$ = k$
              ar& = aa&(ch%): aa&(ch%) = -a&
              MID$(ri$(ch%), j% + ck%(t%), 4) = MKL$(-a&)
            END IF
            LSET vbf$(t%) = ri$(ch%): PUT nfi%(t%), aa&(ch% - 1)
          NEXT ch%
        ELSE
          nv% = ch%
          ae&(nv%) = aa&(nv%)
          aa&(nv%) = ad&(nv%)
          i% = ip%(nv%) + 1
          m$ = MID$(ri$(nv%), (i% - 1) * ct%(t%) + 1, ck%(t%))
          xy$ = MKL$(0)
          IF i% < nti%(t%) THEN xy$ = MID$(ri$(nv%), i% * ct%(t%) + ck%(t%) + 1, 4)
          ad&(nv%) = CVL(xy$)
        END IF
      END IF
    END IF
    ip%(nv%) = i%: kr$(nv%) = m$
  LOOP
  IF kk$ = kr$(nv%) THEN
    IF ar& = aa&(nv%) OR p% = 0 THEN
      UNLOCK nfi%(t%)
      ISadd% = 1
      EXIT FUNCTION
    END IF
  END IF
  ch% = nv%
  CALL ISins(t%, ch%, kk$, ar&)
  UNLOCK nfi%(t%)
  ISadd% = 0
END FUNCTION

FUNCTION ISdel% (t%, k$, a&) STATIC
  SHARED nfi%(), ck%(), ct%(), nti%(), nv%, vbf$(), ri$(), rr$
  SHARED kr$(), ae&(), aa&(), ad&(), ip%()
  LOCK nfi%(t%)
  GET nfi%(t%), 1: ri$(0) = vbf$(t%)
  aa&(0) = CVL(MID$(ri$(0), 5, 4))
  nv% = 0
  IF aa&(0) = 0 THEN
```

```
      UNLOCK nfi%(t%)
      a& = 0
      ISdel% = 1
      EXIT FUNCTION
    END IF
    DO WHILE aa&(nv%) > 0
      nv% = nv% + 1
      GET nfi%(t%), aa&(nv% - 1): ri$(nv%) = vbf$(t%)
      IF CVL(MID$(ri$(nv%), ck%(t%) + 1, 4)) < 0 THEN a& = aa&(nv%): EXIT FOR
      i% = ISpos%(t%, k$, rr&, ri$(nv%), m$, ae&(nv%), aa&(nv%), ad&(nv%))
      ip%(nv%) = i%
      IF i% < 0 THEN
        IF m$ <> k$ THEN
          UNLOCK nfi%(t%)
          ISdel% = 1
          EXIT FUNCTION
        END IF
        ch% = nv%
        DO WHILE ad&(ch%) = 0 AND ch% > 0
          ch% = ch% - 1
        LOOP
        IF ch% = 0 THEN
          UNLOCK nfi%(t%)
          ISdel% = 1
          EXIT FUNCTION
        END IF
        nv% = ch%
        ae&(nv%) = aa&(nv%)
        aa&(nv%) = ad&(nv%)
        kr$(nv%) = MID$(ri$(nv%), ip%(nv%) + ct%(t%) + 1, ck%(t%))
        ip%(nv%) = ip%(nv%) + 1
        xy$ = MKL$(0)
        IF ip%(nv%) < nti%(t%) THEN xy$ = MID$(ri$(nv%), ip%(nv%) + ct%(t%) + ck%(t%) + 1, 4)
        ad&(nv%) = CVL(xy$)
      END IF
      kr$(nv%) = m$
    LOOP
    IF m$ = k$ AND a& = aa&(nv%) THEN
      c% = nv%
      CALL ISrmv(t%, c%)
      ISdel% = 0
    ELSE
      ISdel% = 1
    END IF
    UNLOCK nfi%(t%)
  END FUNCTION

  SUB ISexp (t%, r&) STATIC
    SHARED nfi%(), ck%(), ct%(), nti%(), nv%, vbf$(), ri$(), rr$
    rr$ = STRING$(ct%(t%) + nti%(t%), CHR$(0))
    FOR i& = r& + 1 TO r& + NADI - 1
      MID$(rr$, 1, 4) = MKL$(i& + 1)
      LSET vbf$(t%) = rr$: PUT nfi%(t%), i&
    NEXT i&
    MID$(rr$, 1, 4) = MKL$(0)
    LSET vbf$(t%) = rr$: PUT nfi%(t%), r& + NADI
  END SUB

  FUNCTION ISfind% (t%, k$, a&) STATIC
    SHARED nfi%(), ck%(), ct%(), nti%(), nv%, vbf$(), ri$(), rr$
    SHARED kr$(), ae&(), aa&(), ad&(), ip%()
    SHARED ks$(), ps&()
    LOCK nfi%(t%)
    GET nfi%(t%), 1: ri$(0) = vbf$(t%)
    aa&(0) = CVL(MID$(ri$(0), 5, 4))
    nv% = 0
```

```
  IF aa&(0) = 0 THEN
    UNLOCK nfi%(t%)
    a& = 0
    ISfind% = 1
    EXIT FUNCTION
  END IF
  DO WHILE aa&(nv%) > 0
    nv% = nv% + 1
    GET nfi%(t%), aa&(nv% - 1): ri$(nv%) = vbf$(t%)
    i% = ISpos%(l%, k$, 0, ri$(nv%), m$, ae&(nv%), aa&(nv%), ad&(nv%))
    IF i% < 0 THEN
      UNLOCK nfi%(t%)
      ISfind% = 1: a& = 0
      ks$(t%) = k$: ps&(t%) = 0
      EXIT FUNCTION
    END IF
  LOOP
  UNLOCK nfi%(t%)
  a& = -aa&(nv%)
  ks$(t%) = k$: ps&(t%) = a&
  IF m$ = k$ THEN ISfind% = 0 ELSE ISfind% = 1
END FUNCTION

SUB ISins (t%, c%, k$, u&)
  SHARED nfi%(), ck%(), ct%(), nti%(), nv%, vbf$(), ri$(), rr$
  SHARED kr$(), ae&(), aa&(), ad&(), ip%()
  j% = ip%(c%) * ct%(t%) + 1
  FOR i% = ip%(c%) + 1 TO nti%(t%)
    IF CVL(MID$(ri$(c%), j% + ck%(t%), 4)) = 0 THEN
      jj% = (ip%(c%) - 1) * ct%(t%) + 1: jc% = (i% - ip%(c%)) * ct%(t%)
      rr$ = MID$(ri$(c%), jj%, jc%)
      MID$(ri$(c%), jj% + ct%(t%), jc%) = rr$
      MID$(ri$(c%), jj%, ck%(t%)) = k$
      MID$(ri$(c%), jj% + ck%(t%), 4) = MKL$(u&)
      LSET vbf$(t%) = ri$(c%): PUT nfi%(t%), aa&(c% - 1)
      EXIT SUB
    END IF
    j% = j% + ct%(t%)
  NEXT i%
  im% = (nti%(t%) + 1) \ 2
  m$ = MID$(ri$(c%), (im% - 1) * ct%(t%) + 1, ck%(t%))
  am& = aa&(c% - 1)
  w& = CVL(MID$(ri$(0), 1, 4))
  GET nfi%(t%), w&: rr$ = vbf$(t%)
  wy& = CVL(MID$(rr$, 1, 4))
  IF wy& = 0 THEN
    CALL ISexp(t%, w&): wy& = w& + 1
  END IF
  MID$(ri$(0), 1, 4) = MKL$(wy&)
  LSET vbf$(t%) = ri$(0): PUT nfi%(t%), 1
  IF ip%(c%) <= im% THEN
    rr$ = STRING$(ct%(t%) * nti%(t%), CHR$(0))
    MID$(rr$, 1, (ip%(c%) - 1) * ct%(t%)) = MID$(ri$(c%), 1, (ip%(c%) - 1) * ct%(t%))
    MID$(rr$, (ip%(c%) - 1) * ct%(t%) + 1, ck%(t%)) = k$
    MID$(rr$, (ip%(c%) - 1) * ct%(t%) + ck%(t%) + 1, 4) = MKL$(u&)
    MID$(rr$, ip%(c%) * ct%(t%) + 1, (im% - ip%(c%) + 1) * ct%(t%)) = MID$(ri$(c%), (ip%(c%)
- 1) * ct%(t%) + 1, (im% - ip%(c%) + 1) * ct%(t%))
    MID$(ri$(c%), 1, (nti%(t%) - im%) * ct%(t%)) = MID$(ri$(c%), im% * ct%(t%) + 1, (nti%(t%)
 - im%) * ct%(t%))
    ku$ = MID$(ri$(c%), (nti%(t%) - im% - 1) * ct%(t%) + 1, ck%(t%))
    MID$(ri$(c%), (nti%(t%) - im%) * ct%(t%) + 1, im% * ct%(t%)) = STRING$(im% * ct%(t%), CHR
$(0))
  ELSE
    ip%(c%) = ip%(c%) - im%
    rr$ = STRING$(ct%(t%) * nti%(t%), CHR$(0))
    MID$(rr$, 1, im% * ct%(t%)) = MID$(ri$(c%), 1, im% * ct%(t%))
```

```
    MID$(ri$(c%), 1, (ip%(c%) - 1) * ct%(t%)) = MID$(ri$(c%), im% * ct%(t%) + 1, (ip%(c%) - 1
) * ct%(t%))
    MID$(ri$(c%), ip%(c%) * ct%(t%) + 1, (nti%(t%) - im% - ip%(c%) + 1) * ct%(t%)) = MID$(ri$
(c%), (ip%(c%) + im% - 1) * ct%(t%) + 1, (nti%(t%) - im% - ip%(c%) + 1) * ct%(t%))
    MID$(ri$(c%), (ip%(c%) - 1) * ct%(t%) + 1, ck%(t%)) = k$
    MID$(ri$(c%), (ip%(c%) - 1) * ct%(t%) + ck%(t%) + 1, 4) = MKL$(u&)
    ku$ = MID$(ri$(c%), (nti%(t%) - im%) * ct%(t%) + 1, ck%(t%))
    MID$(ri$(c%), (nti%(t%) - im% + 1) * ct%(t%) + 1, im% * ct%(t%)) = STRING$(im% * ct%(t%),
CHR$(0))
  END IF
    LSET vbf$(t%) = rr$: PUT nfi%(t%), w&
    LSET vbf$(t%) = ri$(c%): PUT nfi%(t%), aa&(c% - 1)
    c% = c% - 1
    IF c% = 0 THEN
      ww& = CVL(MID$(ri$(0), 1, 4))
      GET nfi%(t%), ww&: rr$ = vbf$(t%)
      wy& = CVL(MID$(rr$, 1, 4))
      IF wy& = 0 THEN
        CALL ISexp(t%, ww&): wy& = ww& + 1
      END IF
      MID$(ri$(0), 1, 4) = MKL$(wy&)
      MID$(ri$(0), 5, 4) = MKL$(ww&)
      LSET vbf$(t%) = ri$(0): PUT nfi%(t%), 1
      rr$ = STRING$(ct%(t%) * nti%(t%), CHR$(0))
      MID$(rr$, 1, ck%(t%) + ck%(t%) + 8) = m$ + MKL$(w&) + ku$ + MKL$(am&)
      LSET vbf$(t%) = rr$: PUT nfi%(t%), ww&
      EXIT SUB
    END IF
    CALL ISins(t%, c%, m$, w&)
END SUB

FUNCTION ISnext% (t%, y$, a&) STATIC
  SHARED nfi%(), ck%(), ct%(), nti%(), nv%, vbf$(), ri$(), rr$
  SHARED kr$(), ae&(), aa&(), ad&(), ip%()
  SHARED ks$(), ps&()
  IF ks$(t%) = STRING$(ck%(t%), CHR$(255)) THEN ks$(t%) = STRING$(ck%(t%), CHR$(0))
  LOCK nfi%(t%)
  GET nfi%(t%), 1: ri$(0) = vbf$(t%)
  aa&(0) = CVL(MID$(ri$(0), 5, 4))
  nv% = 0
  IF aa&(0) = 0 THEN
    UNLOCK nfi%(t%)
    a& = 0: y$ = "": ks$(t%) = STRING$(ck%(t%), CHR$(0))
    ISnext% = 1
    EXIT FUNCTION
  END IF
  DO WHILE aa&(nv%) > 0
    nv% = nv% + 1
    GET nfi%(t%), aa&(nv% - 1): ri$(nv%) = vbf$(t%)
    IF CVL(MID$(ri$(nv%), ck%(t%) + 1, 4)) < 0 THEN a& = ps&(t%) ELSE a& = 0
    i% = ISpos%(t%, ks$(t%), a&, ri$(nv%), kr$(nv%), ae&(nv%), aa&(nv%), ad&(nv%))
    ip%(nv%) = i%
    IF i% < 0 THEN
      DO WHILE ad&(nv%) = 0 AND nv% > 0
        nv% = nv% - 1
      LOOP
      IF nv% = 0 THEN
        UNLOCK nfi%(t%)
        y$ = "": ks$(t%) = STRING$(ck%(t%), CHR$(0))
        ISnext% = 1: a& = 0
        EXIT FUNCTION
      END IF
      aa&(nv%) = ad&(nv%)
      kr$(nv%) = MID$(ri$(nv%), ip%(nv%) * ct%(t%) + 1, ck%(t%))
      ip%(nv%) = ip%(nv%) + 1
      xv$ = MKL$(0)
```

```
      IF ip%(nv%) < nti%(t%) THEN y$ = MID$(r... t... ), ip%(nv%) * ct%(t%) + ck%(t%) + 1, 4)
      ad&(nv%) = CVL(y$)
    END IF
    IF aa&(nv%) < 0 THEN
      IF kr$(nv%) > ks$(t%) OR (kr$(nv%) = ks$(t%) AND -aa&(nv%) > ps&(t%)) THEN
        UNLOCK nfi%(t%)
        y$ = kr$(nv%): a& = -aa&(nv%)
        ks$(t%) = y$: ps&(t%) = a&
        ISnext% = 0
        EXIT FUNCTION
      END IF
      IF ad&(nv%) <> 0 THEN
        UNLOCK nfi%(t%)
        a& = -ad&(nv%)
        y$ = MID$(ri$(nv%), i% * ct%(t%) + 1, ck%(t%))
        ks$(t%) = y$: ps&(t%) = a&
        ISnext% = 0
        EXIT FUNCTION
      END IF
      DO WHILE ad&(nv%) = 0
        nv% = nv% - 1
        IF nv% = 0 THEN
          UNLOCK nfi%(t%)
          ss% = 0: y$ = "": ks$(t%) = STRING$(ck%(t%), CHR$(0))
          ISnext% = 1: a& = 0
          EXIT FUNCTION
        END IF
      LOOP
      aa&(nv%) = ad&(nv%)
      kr$(nv%) = MID$(ri$(nv%), ip%(nv%) * ct%(t%), ck%(t%))
      ip%(nv%) = ip%(nv%) + 1
      xy$ = MKL$(0)
      IF ip%(nv%) < nti%(t%) THEN y$ = MID$(ri$(nv%), ip%(nv%) * ct%(t%) + ck%(t%) + 1, 4)
      ad&(nv%) = CVL(y$)
    END IF
  LOOP
END FUNCTION

SUB ISopen (t%, nf%, nam$, cbuf%, nk%) STATIC
  SHARED nfi%(), ck%(), ct%(), nti%(), nv%, vbf$(), ri$(), rr$
  SHARED kr$(), ae&(), aa&(), ad&(), ip%()
  SHARED ks$(), ps&()
  IF nf% = 0 THEN nf% = FREEFILE
  ck%(t%) = nk%: ct%(t%) = ck%(t%) + 4
  nti%(t%) = cbuf% \ ct%(t%)
  nfi%(t%) = nf%
  nv% = 0: ks$(t%) = STRING$(ck%(t%), CHR$(0)): ps&(t%) = 0
  OPEN nam$ FOR RANDOM ACCESS READ WRITE SHARED AS nfi%(t%) LEN = cbuf%
  FIELD nfi%(t%), cbuf% AS vbf$(t%)
  IF LOF(nfi%(t%)) = 0 THEN
    rr$ = STRING$(ct%(t%) * nti%(t%), CHR$(0))
    LOCK nfi%(t%)
    FOR i% = 1 TO NADI - 1
      MID$(rr$, 1, 4) = MKL$(i% + 1)
      LSET vbf$(t%) = rr$: PUT nfi%(t%), i%
    NEXT i%
    MID$(rr$, 1, 4) = MKI$(0)
    LSET vbf$(t%) = rr$: PUT nti%(t%), NADI
    UNLOCK nfi%(t%)
  END IF
END SUB

FUNCTION IOpos% (t%, k$, a&, r$, m$, e&, s&, d&) STATIC
  SHARED nfi%(), ck%(), ct%(), nti%(), nv%, vbf$(), ri$(), rr$
  SHARED kr$(), ae&(), aa&(), ad&(), ip%()
  i% = 1
```

```
  FOR i% = 1 TO nti%(t%)
    uk$ = MID$(r$, j%, ck%(t%))
    ua& = CVL(MID$(r$, j% + ck%(t%), 4))
    IF uk$ > k$ OR (uk$ = k$ AND (u& = 0 OR ua& > 0 OR -ua& >= u&)) THEN
      ISpos% = i%: m$ = MID$(r$, j%, ck%(t%)): a& = ua&
      IF i% = 1 THEN e& = 0 ELSE e& = CVL(MID$(r$, j% - 4, 4))
      IF i% = nti%(t%) THEN d& = 0 ELSE d& = CVL(MID$(r$, j% + ct%(t%) + ck%(t%), 4))
      EXIT FUNCTION
    END IF
    IF ua& = 0 THEN
      ISpos% = -i% + 1
      a& = CVL(MID$(r$, j% - 4, 4))
      IF i% < -2 THEN e& = CVL(MID$(r$, j% - ct%(t%) - 4, 4)) ELSE e& = 0
      d& = 0
      m$ = MID$(r$, j% - ct%(t%), ck%(t%))
      EXIT FUNCTION
    END IF
    j% = j% + ct%(t%)
  NEXT i%
  ISpos% = -nti%(t%)
  m$ = MID$(r$, j% - ct%(t%), ck%(t%)): d& = 0
  e& = CVL(MID$(r$, j% - ct%(t%) - 4, 4)): a& = CVL(MID$(r$, j% - 4, 4))
END FUNCTION

FUNCTION ISprev% (t%, y$, a&) STATIC
  SHARED nfi%(), ck%(), ct%(), nti%(), nv%, vbf$(), ri$(), rr$
  SHARED kr$(), ae&(), aa&(), ad&(), ip%()
  SHARED ks$(), ps&()
  IF ks$(t%) = STRING$(ck%(t%), CHR$(0)) THEN ks$(t%) = STRING$(ck%(t%), CHR$(255))
  LOCK nfi%(t%)
  GET nfi%(t%), 1: ri$(0) = vbf$(t%)
  aa&(0) = CVL(MID$(ri$(0), 5, 4))
  nv% = 0
  IF aa&(0) = 0 THEN
    a& = 0: y$ = "": ps&(t%) = 0: ks$(t%) = STRING$(ck%(t%), CHR$(255))
    ISprev% = 1
    UNLOCK nfi%(t%)
    EXIT FUNCTION
  END IF
  it% = 0
  DO WHILE aa&(nv%) > 0
    nv% = nv% + 1
    GET nfi%(t%), aa&(nv% - 1): ri$(nv%) = vbf$(t%)
    xy$ = MID$(ri$(nv%), ck%(t%) + 1, 4): zz& = CVL(xy$)
    IF zz& < 0 THEN zz& = ps&(t%) ELSE zz& = 0
    i% = ISpos%(t%, ks$(t%), zz&, ri$(nv%), kr$(nv%), ae&(nv%), aa&(nv%), ad&(nv%))
    IF it% = 1 AND aa&(nv%) > 0 AND i% < nti%(t%) THEN
      DO WHILE ad&(nv%) > 0
        i% = i% + 1: q% = i% * ct%(t%) + ck%(t%)
        xy$ = MID$(ri$(nv%), q%, 4)
        ae&(nv%) = aa&(nv%): aa&(nv%) = ad&(nv%)
        ad&(nv%) = CVL(xy$)
      LOOP
    END IF
    ip%(nv%) = i%
    IF i% < 0 THEN
      ip%(nv%) = -i%
      IF aa&(nv%) < 0 THEN
        IF kr$(nv%) <> ks$(t%) OR it% = 1 THEN
          UNLOCK nfi%(t%)
          ISprev% = 0: a& = -aa&(nv%): y$ = kr$(nv%)
          ps&(t%) = a&: ks$(t%) = y$
          EXIT FUNCTION
        END IF
        ch% = nv%
        DO WHILE ad&(ch%) = 0 AND ch% > 0
```

```
        ch% = ch% - 1
      LOOP
      IF ch% = 0 THEN
        UNLOCK nfi%(t%)
        ISprev% = 0: a& = -aa&(nv%): y$ = kr$(nv%)
        ps&(t%) = a&: ks$(t%) = y$
        EXIT FUNCTION
      END IF
      nv% = ch%
      ae&(nv%) = aa&(nv%)
      aa&(nv%) = ad&(nv%)
      kr$(nv%) = MID$(ri$(nv%), ip%(nv%) * ct%(t%) + 1, 4)
      ip%(nv%) = ip%(nv%) + 1
      xy$ = MKL$(0)
      IF ip%(nv%) < nti%(t%) THEN xy$ = MID$(ri$(nv%), ip%(nv%) * ct%(t%) + ck%(t%) + 1, 4)
      ad&(nv%) = CVL(xy$)
    END IF
  ELSE
    IF aa&(nv%) < 0 THEN
      IF kr$(nv%) < ks$(t%) OR (kr$(nv%) = ks$(t%) AND -aa&(nv%) < ps&(t%)) THEN
        UNLOCK nfi%(t%)
        ISprev% = 0: a& = -aa&(nv%): y$ = kr$(nv%)
        ps&(t%) = a&: ks$(t%) = y$
        EXIT FUNCTION
      END IF
      IF ae&(nv%) < 0 THEN
        UNLOCK nfi%(t%)
        ISprev% = 0
        a& = -ae&(nv%): y$ = MID$(ri$(nv%), (i% - 2) * ct%(t%) + 1, ck%(t%))
        ps&(t%) = a&: ks$(t%) = y$
        EXIT FUNCTION
      END IF
      ch% = nv%
      DO WHILE ae&(ch%) = 0 AND ch% > 0
        ch% = ch% - 1
      LOOP
      IF ch% = 0 THEN
        UNLOCK nfi%(t%)
        ISprev% = 1
        a& = 0: y$ = ""
        ps&(t%) = 0: ks$(t%) = STRING$(ck%(t%), CHR$(255))
        EXIT FUNCTION
      END IF
      nv% = ch%
      ad&(nv%) = aa&(nv%)
      aa&(nv%) = ae&(nv%)
      ip%(nv%) = ip%(nv%) - 1
      kr$(nv%) = MID$(ri$(nv%), (ip%(nv%) - 1) * ct%(t%) + 1, ck%(t%))
      xy$ = MKL$(0)
      IF ip%(nv%) > 1 THEN xy$ = MID$(ri$(nv%), (ip%(nv%) - 1) * ct%(t%) - 3, 4)
      ae& = CVL(xy$)
      it% = 1
    END IF
  END IF
LOOP
END FUNCTION

SUB ISrmv (t%, c%)
  SHARED nfi%(), ck%(), ct%(), nti%(), nv%, vbf$(), ri$(), rr$
  SHARED kr$(), ae&(), aa&(), ad&(), ip%()
  ma% = nti%(t%)
  FOR i% = nti%(t%) TO 1 STEP -1
    IF CVL(MID$(ri$(c%), i% * ct%(t%) - 3, 4)) = 0 THEN ma% = ma% - 1 ELSE EXIT FOR
  NEXT i%
  MID$(ri$(c%), (ip%(c%) - 1) * ct%(t%) + 1, (ma% - ip%(c%)) * ct%(t%)) = MID$(ri$(c%), ip%(c
%) * ct%(t%) + 1, (ma% - ip%(c%)) * ct%(t%))
```

```
  ma% = ma% - 1
  IF ma% = 1 AND c% = 1 AND CVL(MID$(ri$(c%), ck%(t%) + 1, 4)) > 0 THEN
    LSET vbf$(t%) = MID$(ri$(0), 1, 4) + STRING$(nti%(t%) * ct%(t%) - 4, CHR$(0))
    PUT nfi%(t%), aa&(0)
    MID$(ri$(0), 1, 8) = MKL$(aa&(0)) + MID$(ri$(1), ck%(t%) + 1, 4)
    LSET vbf$(t%) = ri$(0): PUT nfi%(t%), 1
    EXIT SUB
  END IF
  IF ma% > 0 THEN
    MID$(ri$(c%), ma% * ct%(t%) + 1, ct%(t%)) = STRING$(ct%(t%), CHR$(0))
    LSET vbf$(t%) = ri$(c%): PUT nfi%(t%), aa&(c% - 1)
  ELSE
    MID$(ri$(c%), 1, ct%(t%)) = MID$(ri$(0), 1, 4) + STRING$(ck%(t%), CHR$(0))
    LSET vbf$(t%) = ri$(c%): PUT nfi%(t%), aa&(c% - 1)
    MID$(ri$(0), 1, 4) = MKL$(aa&(c% - 1))
    MID$(ri$(0), 5, 4) = MKL$(0)
    LSET vbf$(t%) = ri$(0): PUT nfi%(t%), 1
    aa&(0) = 0
    EXIT SUB
  END IF
  IF ad&(c%) = 0 THEN
    ip%(c%) = ip%(c%) - 1
    kr$(c%) = MID$(ri$(c%), (ip%(c%) - 1) * ct%(t%) + 1, ck%(t%))
    ci% = c% - 1
    DO WHILE ci% > 0
      MID$(ri$(ci%), (ip%(ci%) - 1) * ct%(t%) + 1, ck%(t%)) = kr$(c%)
      LSET vbf$(t%) = ri$(ci%): PUT nfi%(t%), aa&(ci% - 1)
      IF ad&(ci%) = 0 THEN
        kr$(ci%) = kr$(c%): ci% = ci% - 1
      ELSE
        ci% = 0
      END IF
    LOOP
  ELSE
    kr$(c%) = MID$(ri$(c%), (ip%(c%) - 1) * ct%(t%) + 1, ck%(t%))
  END IF
  IF c% = 1 THEN EXIT SUB
  n2% = nti%(t%) \ 2
  IF ma% >= n2% AND (ae&(c% - 1) <> 0 OR ad&(c% - 1) <> 0) THEN EXIT SUB
  me% = 0: md% = 0
  IF ae&(c% - 1) <> 0 THEN
    me% = nti%(t%)
    GET nfi%(t%), ae&(c% - 1): rp$ = vbf$(t%)
    FOR i% = nti%(t%) TO 1 STEP -1
      IF CVL(MID$(rp$, i% * ct%(t%) - 3, 4)) = 0 THEN me% = me% - 1 ELSE EXIT FOR
    NEXT i%
  END IF
  IF ad&(c% - 1) <> 0 THEN
    md% = nti%(t%)
    GET nfi%(t%), ad&(c% - 1): rq$ = vbf$(t%)
    FOR i% = nti%(t%) TO 1 STEP -1
      IF CVL(MID$(rq$, i% * ct%(t%) - 3, 4)) = 0 THEN md% = md% - 1 ELSE EXIT FOR
    NEXT i%
  END IF
  nn% = ma% + me% + md%
  IF ad&(c% - 1) = 0 THEN
    IF nn% < nti%(t%) THEN
      ci% = c% - 1
      MID$(rp$, me% * ct%(t%) + 1, ma% * ct%(t%)) = MID$(ri$(c%), 1, ma% * ct%(t%))
      LSET vbf$(t%) = rp$: PUT nfi%(t%), ae&(ci%)
      bb& = aa&(ci%)
      MID$(ri$(ci%), (ip%(ci%) - 2) * ct%(t%) + 1, ck%(t%)) = kr$(c%)
      LSET vbf$(t%) = ri$(ci%): PUT nfi%(t%), aa&(ci% - 1)
      CALL ISrmv(t%, ci%)
      rp$ = MID$(ri$(0), 1, 4) + STRING$(nti%(t%) * ct%(t%) - 4, CHR$(0))
      LSET vbf$(t%) = rp$: PUT nfi%(t%), bb&
```

```
      MID$(ri$(0), 1, 4) = MKL$(bb&)
      LSET vbf$(t%) = ri$(0): PUT nfi%(t%), 1
      EXIT SUB
    END IF
    md% = (nn% \ 2) - ma%
    rq$ = ri$(c%)
    MID$(ri$(c%), 1, md% * ct%(t%)) = MID$(rp$, (me% - md%) * ct%(t%) + 1, md% * ct%(t%))
    MID$(ri$(c%), md% * ct%(t%) + 1, ma% * ct%(t%)) = MID$(rq$, 1, ma% * ct%(t%))
    LSET vbf$(t%) = ri$(c%): PUT nfi%(t%), aa&(c% - 1)
    MID$(rp$, (me% - md%) * ct%(t%) + 1, md% * ct%(t%)) = STRING$(md% * ct%(t%), CHR$(0))
    LSET vbf$(t%) = rp$: PUT nfi%(t%), ae&(c% - 1)
    MID$(ri$(c% - 1), (ip%(c% - 1) - 2) * ct%(t%) + 1, ck%(t%)) = MID$(rp$, (me% - md% - 1) *
ct%(t%) + 1, ck%(t%))
    LSET vbf$(t%) = ri$(c% - 1): PUT nfi%(t%), aa&(c% - 2)
    EXIT SUB
  END IF
  IF ae&(c% - 1) = 0 THEN
    IF nn% < nti%(t%) THEN
      ci% = c% - 1
      MID$(ri$(c%), ma% * ct%(t%) + 1, md% * ct%(t%)) = MID$(rq$, 1, md% * ct%(t%))
      LSET vbf$(t%) = ri$(c%): PUT nfi%(t%), ad&(c% - 1)
      bb& = aa&(ci%)
      CALL ISrmv(t%, ci%)
      rq$ = MID$(ri$(0), 1, 4) + STRING$(nti%(t%) * ct%(t%) - 4, CHR$(0))
      LSET vbf$(t%) = rq$: PUT nfi%(t%), bb&
      MID$(r1$(0), 1, 4) = MKL$(bb&)
      LSET vbf$(t%) = ri$(0): PUT nfi%(t%), 1
      EXIT SUB
    END IF
    me% = (nn% \ 2) - ma%
    MID$(ri$(c%), ma% * ct%(t%) + 1, me% * ct%(t%)) = MID$(rq$, 1, me% * ct%(t%))
    LSET vbf$(t%) = ri$(c%): PUT nfi%(t%), aa&(c% - 1)
    MID$(rq$, 1, (md% - me%) * ct%(t%)) = MID$(rq$, me% * ct%(t%) + 1, (md% - me%) * ct%(t%))
    MID$(rq$, (md% - me%) * ct%(t%) + 1, me% * ct%(t%)) = STRING$(me% * ct%(t%), CHR$(0))
    LSET vbf$(t%) = rq$: PUT nfi%(t%), ad&(c% - 1)
    kr$(c%) = MID$(ri$(c%), (ma% + me% - 1) * ct%(t%) + 1, ck%(t%))
    MID$(ri$(c% - 1), (ip%(c% - 1) - 1) * ct%(t%) + 1, ck%(t%)) = kr$(c%)
    LSET vbf$(t%) = ri$(c% - 1): PUT nfi%(t%), aa&(c% - 2)
    EXIT SUB
  END IF
  ru$ = STRING$(nti%(t%) * ct%(t%), CHR$(0))
  IF nn% < nti%(t%) + nti%(t%) THEN
    mm% = nn% \ 2
    bb& = aa&(c% - 1)
    IF mm% > me% THEN
      IF (mm% - me%) <= ma% THEN
        MID$(rp$, me% * ct%(t%) + 1, (mm% - me%) * ct%(t%)) = MID$(ri$(c%), 1, (mm% - me%) *
ct%(t%))
        MID$(ru$, 1, (ma% - mm% + me%) * ct%(t%)) = MID$(ri$(c%), (mm% - me%) * ct%(t%) + 1,
(ma% - mm% + me%) * ct%(t%))
        MID$(ru$, (ma% - mm% + me%) * ct%(t%) + 1, md% * ct%(t%)) = MID$(rq$, 1, md% * ct%(t%
))
      ELSE
        MID$(rp$, me% * ct%(t%) + 1, ma% * ct%(t%)) = MID$(ri$(c%), 1, ma% * ct%(t%))
        MID$(rp$, (me% + ma%) * ct%(t%) + 1, (mm% - me% - ma%) * ct%(t%)) = MID$(rq$, 1, (mm%
 - me% - ma%) * ct%(t%))
        MID$(ru$, 1, (nn% - mm%) * ct%(t%)) = MID$(rq$, (mm% - me% - ma%) * ct%(t%) + 1, (nn%
 - mm%) * ct%(t%))
      END IF
    ELSE
      MID$(ru$, 1, (me% - mm%) * ct%(t%)) = MID$(rp$, mm% * ct%(t%) + 1, (me% - mm%) * ct%(t%
))
      MID$(rp$, mm% * ct%(t%) + 1, (me% - mm%) * ct%(t%)) = STRING$((me% - mm%) * ct%(t%), CH
R$(0))
      MID$(ru$, (me% - mm%) * ct%(t%) + 1, ma% * ct%(t%)) = MID$(ri$(c%), 1, ma% * ct%(t%))
      MID$(ru$, (me% - mm% + ma%) * ct%(t%) + 1, md% * ct%(t%)) = MID$(rq$, 1, md% * ct%(t%))
```

```
    END IF
    LSET vbf$(t%) = rp$: PUT nfi%(t%), ae&(c% - 1)
    LSET vbf$(t%) = ru$: PUT nfi%(t%), ad&(c% - 1)
    ci% = c% - 1
    MID$(ri$(ci%), (ip%(ci%) - 2) * ct%(t%) + 1, ck%(t%)) = MID$(rp$, (mm% - 1) * ct%(t%) + 1
  ck%(t%))
    LSET vbf$(t%) = ri$(ci%): PUT nfi%(t%), aa&(ci% - 1)
    CALL ISrmv(t%, ci%)
    rq$ = MID$(ri$(0), 1, 4) + STRING$(nti%(t%) * ct%(t%) - 4, CHR$(0))
    LSET vbf$(t%) = rq$: PUT nfi%(t%), bb&
    MID$(ri$(0), 1, 4) = MKL$(bb&)
    LSET vbf$(t%) = ri$(0): PUT nfi%(t%), 1
    EXIT SUB
  END IF
  mm% = nn% \ 3
  IF me% < mm% THEN
    MID$(rp$, me% * ct%(t%) + 1, (mm% - me%) * ct%(t%)) = MID$(ri$(c%), 1, (mm% - me%) * ct%(
t%))
    LSET vbf$(t%) = rp$: PUT nfi%(t%), ae&(c% - 1)
    MID$(ri$(c% - 1), (ip%(c% - 1) - 2) * ct%(t%) + 1, ck%(t%)) = MID$(rp$, (mm% - 1) * ct%(t
%) + 1, ck%(t%))
    LSET vbf$(t%) = ri$(c% - 1): PUT nfi%(t%), aa&(c% - 2)
    MID$(ru$, 1, (ma% - mm% + me%) * ct%(t%)) = MID$(ri$(c%), (mm% - me%) * ct%(t%) + 1, (ma%
 - mm% + me%) * ct%(t%))
    ma% = ma% - mm% + me%
  ELSE
    MID$(ru$, 1, (me% - mm%) * ct%(t%)) = MID$(rp$, mm% * ct%(t%) + 1, (me% - mm%) * ct%(t%))
    MID$(ru$, (me% - mm%) * ct%(t%) + 1, ma% * ct%(t%)) = MID$(ri$(c%), 1, ma% * ct%(t%))
    ma% = ma% + me% - mm%
    MID$(rp$, mm% * ct%(t%) + 1, (me% - mm%) * ct%(t%)) = STRING$((me% - mm%) * ct%(t%), CHR$
(0))
    LSET vbf$(t%) = rp$: PUT nfi%(t%), ae&(c% - 1)
    MID$(ri$(c% - 1), (ip%(c% - 1) - 2) * ct%(t%) + 1, ck%(t%)) = MID$(rp$, (mm% - 1) * ct%(t
%) + 1, ck%(t%))
    LSET vbf$(t%) = ri$(c% - 1): PUT nfi%(t%), aa&(c% - 2)
  END IF
  rp$ = STRING$(nti%(t%) * ct%(t%), CHR$(0))
  IF ma% > mm% THEN
    MID$(rp$, 1, (ma% - mm%) * ct%(t%)) = MID$(ru$, mm% * ct%(t%) + 1, (ma% - mm%) * ct%(t%))
    MID$(rp$, (ma% - mm%) * ct%(t%) + 1, md% * ct%(t%)) = MID$(rq$, 1, md% * ct%(t%))
    MID$(ru$, mm% * ct%(t%) + 1, (ma% - mm%) * ct%(t%)) = STRING$((ma% - mm%) * ct%(t%), CHR$
(0))
  ELSE
    MID$(ru$, ma% * ct%(t%) + 1, (mm% - ma%) * ct%(t%)) = MID$(rq$, 1, (mm% - ma%) * ct%(t%))
    MID$(rp$, 1, (md% - mm% + ma%) * ct%(t%)) = MID$(rq$, (mm% - ma%) * ct%(t%) + 1, (md% - m
m% + ma%) * ct%(t%))
  END IF
  LSET vbf$(t%) = rp$: PUT nfi%(t%), ad&(c% - 1)
  LSET vbf$(t%) = ru$: PUT nfi%(t%), aa&(c% - 1)
  MID$(ri$(c% - 1), (ip%(c% - 1) - 1) * ct%(t%) + 1, ck%(t%)) = MID$(ru$, (mm% - 1) * ct%(t%)
 + 1, ck%(t%))
  LSET vbf$(t%) = ri$(c% - 1): PUT nfi%(t%), aa&(c% - 2)
  EXIT SUB
END SUB
```