MÉTODOS DE HASHING EXTERNO

Isabel Helena Coelho Soares

DISSERTAÇÃO APRESENTADA
AO
INSTITUTO DE MATEMÁTICA E ESTATÍSTICA
DA
UNIVERSIDADE DE SÃO PAULO
PARA OBTENÇÃO DO GRAU DE MESTRE
EM
MATEMÁTICA APLICADA

Área de Concentração: **Ciência da Computação**
Orientador: **Prof. Dr. Valdemar W. Setzer**

-SÃO PAULO, Junho de 1988-

// AGRADECIMENTOS

O meu muito obrigada

Ao Professor Dr. Valdemar W. Setzer

que como orientador, mas principalmente como amigo foi o grande incentivador para a conclusão deste trabalho. Sua paciência, carinho e dedicação constantes excederam em muito o que seria de se esperar de uma orientação;

Ao Gilson Julio Guizardi

pela construção dos desenhos, e principalmente pela sua paciência e carinho durante a elaboração deste trabalho;

À Sarah Kohan

pela leitura e sugestões feitas no original, pelo apoio dado, mas principalmente pela grande incentivadora para o término deste trabalho;

Ao Luis Carlos Gaspar

pela ajuda na digitação, e pela grande amizade demonstrada nos momentos mais difíceis da elaboração deste trabalho;

Ao Antonio Fernando Carvalho Gomes

não só pelo grande incentivo mas também pelos recursos materiais oferecidos que facilitaram em muito o término deste trabalho;

Aos amigos

cujo apoio direto ou indireto foi decisivo nos momentos mais difíceis da elaboração deste trabalho;

ÍNDICE

pág.

CAP. I _ INTRODUÇÃO 1

1.1 _ Definições 3

CAP. II _ TRATAMENTO DE COLISÕES 5

CAP. III _ HASHING ESTÁTICO 10

3.1 _ Método dos separadores 11

CAP. IV _ HASHING DINÂMICO 16

4.1 _ Hashing linear indexado 19
4.1.1 _ Hashing dinâmico em árvore 20
4.1.2 _ Hashing virtual VH0 26
4.1.3 _ Hashing extensível 35

4.2 _ Hashing indexado não linear 44
4.2.1 _ Hashing virtual VH1 45

4.3 _ Hashing linear não indexado 53
4.3.1 _ Hashing virtual linear 54
4.3.2 _ Hashing linear com expansão parcial 60
4.3.3 _ Hashing linear com expansão em grupo 69
4.3.4 _ Hashing com armazenamento em espiral 74

CAP. V _ UM NOVO MÉTODO DE HASHING 80

5.1 _ Descrição geral do método 81

5.2 _ Processo de expansão e contração 83

5.3 _ Função de endereçamento 94

5.4 _ Análise da distribuição dos registros pelo arquivo 100

5.5 _ Modificações sugeridas para o Hashing linear decimal com distribuição não uniforme 108

REFERÊNCIAS 111

NOTAÇÕES

N : conjunto dos números naturais

N^* : conjunto dos números naturais positivos

Z : conjunto dos inteiros

Z_+ : conjunto dos inteiros positivos

$[a\,..\,b]$: $\{i \in Z | \; a \leq i \leq b\}$

R : conjunto dos números reais

$[a, b)$: $\{i \in R \; | \; a \leq i < b\}$

$\Rightarrow$: implica que

$\Longleftrightarrow$: se e somente se

$\lfloor i \rfloor$: maior inteiro menor ou igual ao real i

$\lceil i \rceil$: menor inteiro maior ou igual ao real i

$|$: tal que

$|J|$: cardinalidade do conjunto J

$f(n) = O(g(n))$: existem constantes $n_0 > 0$ e $c > 0$ tais que para todo $n > n_0$, $f(n) \leq cg(n)$

$\rightarrow$: aplicação de função ou convergência de sequência

Capítulo I

INTRODUÇÃO

"Hashing" é uma técnica de organizar arquivos onde o acesso aos seus registros é direta, isto é, o número de registros lidos para fazer um acesso a um determinado registro é no máximo uma constante. Esta técnica é simples e as operações de consulta, inserção, eliminação e procura de registros de um arquivo são rápidas. O *custo* de uma operação é dado pelo número de acessos ao disco necessário para executar a operação. O desempenho da operação é melhor quanto menor o seu custo, ou seja, quanto menos acessos ao disco forem feitos durante a sua execução.

O arquivo é dividido logicamente em páginas e os registros são distribuídos entre elas. Para fazer esta distribuição utilizam-se funções, chamadas de "funções de hashing". No entanto, pode existir página em que a quantidade de registros associados a ela é maior que sua capacidade. Quando isto ocorre dizemos que houve colisão. Ela é indesejável, pois em geral provoca o aumento do número de acessos, diminuindo o desempenho

das operações.

Hashing estático é a técnica de organizar arquivos, usando hashing, onde o tamanho do arquivo é estimado na sua criação. O espaço físico alocado para o arquivo é fixo. Quando ocorre a necessidade de mais área em disco o arquivo deve ser reorganizado, isto é, os seus registros devem ser redistribuídos em outra área maior. Para arquivos que crescem e diminuem dinamicamente esta técnica se torna muito dispendiosa, uma vez que superestimando o tamanho do arquivo há o pouco aproveitamento do disco; no entanto, se o seu tamanho for sub-estimado, pode ocorrer grande quantidade de colisões.

Hashing dinâmico é a técnica de hashing em que o tamanho do arquivo varia. Nesta técnica o espaço pode ser liberado ou obtido dinamicamente. O espaço alocado na criação do arquivo deve ser o suficiente para conter a coleção inicial de registros. Durante a existência do arquivo, o espaço é alocado ou liberado segundo um critério pré-determinado. Não iremos enfocar aqui o procedimento de alocação e liberação de espaço em disco. Vamos supor a existência de um sistema que executa tais funções, e quando necessitamos ou liberamos espaço, é feito um pedido a ele para executar a devida função.

Os métodos de Hashing diminuíram de importância com o aparecimento da B-árvore em 1972, pois esta permite o acesso ordenado pelo índice aliado a um pequeno número de acessos. No entanto, na atual década novos métodos de Hashing foram desenvolvidos superando o desempenho da B-árvore, chegando em alguns casos, como veremos neste trabalho, a garantir um único acesso aos registros. Além do desempenho satisfatório, eles são dinâmicos permitindo a expansão do arquivo de maneira simples e com poucos acessos ao disco, contrariamente aos métodos tradicionais de Hashing, que são estáticos.

No meio empresarial também verificamos o interesse e conhecimento do Hashing externo, pois encontramos banco de dados que utilizam métodos de Hashing como única possibilidade (tal como o ULTRA, PICK, BD-SOD, etc.), e outros como método complementar ou alternativo da B-árvore (por exemplo, ADABAS, IMS etc.).

O objetivo inicial deste trabalho foi fazer uma resenha dos métodos existentes, pois não há nada mais ou menos detalhado sobre esse assunto. Para isto, elaboramos uma subdivisão dos métodos, e para cada método uma apresentação sistemática por tópicos comuns. O estudo da bibliografia

levou-nos à pesquisa de um novo método, na qual tivemos sucesso. Na referência bibliográfica consta os trabalhos de Hashing externo que encontramos, mas nem todos eles estão citados no decorrer desta tese.

Na próxima seção desse capítulo introduzimos alguns conceitos necessários para o entedimento deste texto. No capítulo II apresentamos as técnicas clássicas para o tratamento das colisões. No capítulo seguinte relatamos os métodos de hashing estático; os métodos de hashing dinâmico encontram-se no capítulo IV. No último capítulo está descrito um novo método de hashing dinâmico.

1.1_ Definições

Esta seção, contém a definição de alguns conceitos utilizados nos próximos capítulos. Esses conceitos não estão definidos no local onde eles estão sendo usados, pois se isso ocorresse haveria desvio do principal assunto abordado, acarretando perdas no seu entendimento.

Def. 1.1_ sejam $A \subseteq R$, $F = \{J \subseteq R \mid J \subseteq A\}$, F é uma partição de A se as restrições i e ii estão satisfeitas:
i- $A = \cup_{J \in F} J$
ii- $I,J \in F \mid I \neq J \Rightarrow I \cap J = \{\ \}$.

Def. 1.2_ Família de conjuntos e conjunto de sub-conjuntos são sinônimos neste texto.

Def. 1.3_ Uma sequência é uma aplicação $x : N^* \to R$. O valor que a sequência x assume no número $n \in N$ é indicado por x_n, e é chamado de n-ésimo termo da sequência. A sequência x é denotada por $x_1, x_2, \ldots$.

Def. 1.4_ Uma subsequência da sequência $x_1, x_2, \ldots$ é uma restrição da aplicação $x : N^* \to R$ a um subconjunto infinito de N^*.

Def. 1.5_ A sequência $x_1, x_2, \ldots$ é limitada se existe $c \in R$ tal que $c > 0$ e $-c < x_m - x_n < c$ para quisquer $n, m \in N^*$.

Def. 1.6_ A sequência $x_1, x_2, \ldots$ é decrescente se $x_1 \leq x_2 \leq x_3 \leq \ldots \leq x_n \leq \ldots$. Ela é crescente se $x_1 \geq x_2 \geq \ldots \geq x_n \geq \ldots$.

Def. 1.7_ A sequência $x_1, x_2, \ldots$ é estritamente decrescente se $x_1 < x_2 < \ldots < x_n < \ldots$. Ela é estritamente crescente se $x_1 > x_2 > \ldots > x_n > \ldots$.

Def. 1.8_ Uma sequência $x_1, x_2, \ldots$ é convergente se existe $a \in R$ tal que $a = \lim x_n$.

Def. 1.9_ Espiral logarítmica é uma função definida da seguinte forma: seja $a \in R$ então $f(x) = a^x$ onde $x \in R$.

Def. 1.10_ Revolução completa de uma espiral logarítmica f que inicia-se no seu ponto $(y, f(y))$ para algum $y \in R$, é o conjunto dos pontos $\{(x, f(x)) \mid x \in R \text{ e } y \leq x \leq y+1\}$.

Def. 1.11_ Se f é uma função bijetora, f^{-1} denota a função inversa de f.

Def. 1.12_ Página p designa a página do arquivo que possui número p.

Def. 1.13_ Fator de ocupação de um arquivo é o número real $F = \frac{R}{QB}$, onde Q é a quantidade de páginas contidas nele, R o número de registros armazenados no arquivo e B é a capacidade de cada página. Note que $0 < F \leq 1$.

Lema 1.1 : *Toda sequência limitada estritamente crescente ou estritamente decrescente é convergente.*

Capítulo II

TRATAMENTO DE COLISÕES

Tanto no hashing dinâmico como no estático, os registros possuem um campo para a sua identificação, que chamaremos de *chave*. Cada registro possui uma única chave, e uma chave pode identificar um ou mais registros. Usamos a palavra "chave" pois assim é usado na literatura; mas na nomenclatura atual de Banco de Dados a palavra usada é "índice".

Chamaremos de *função de endereçamento* a função que associa os registros às páginas do arquivo. As funções de hashing são utilizadas pela função de endereçamento para distribuir os registros pelas páginas existentes no arquivo. A função de endereçamento mapeia o conjunto das chaves num conjunto de páginas do arquivo, ao qual denominaremos de *espaço de endereçamento* do arquivo. Definimos o *endereço* de um registro com chave k, como sendo a página F(k) onde F é a função de endereçamento. Esta página será chamada de *página original* do registro.

Diz-se que ocorreu *colisão* quando se tenta inserir um registro em

uma página cheia. Chamaremos de *registros de colisão* de uma página, os registros que são endereçados a ela mas não estão armazenados nela por que ocorreu colisão. A *região de colisão* de uma página é o local onde os seus registros de colisão são alocados.

Ao se fazer a procura de um registro de colisão, é realizada sua busca na região de colisão de sua página original. Por esta razão, quando uma operação é feita sobre ele há mais acessos ao disco, do que quando ela é realizada sobre um registro que não é de colisão. Por isto é que a análise dos métodos é feita sobre o número médio de acessos necessários para executar cada operação. Definimos *custo médio* de uma operação, como sendo a média (esperança) do número de acessos ao disco para executar a operação em um registro armazenado no arquivo. A procura de um registro que está armazenado no arquivo é chamada de *procura com sucesso.* Quando o registro não se encontra no arquivo a sua procura é chamada de *procura sem sucesso.*

Na literatura encontramos duas maneiras de tratar a colisão:

i- os registros de colisão são armazenados no próprio arquivo de dados;

ii- os registros de colisão são colocados em uma área separada, denominada arquivo de dados, que é criada para contê-los.

Quando existe uma área separada, ela também é dividida logicamente em páginas, as quais chamaremos de *páginas de colisão.* Cada página dessas armazena os registros de colisão de uma única página do espaço de endereçamento. As páginas de colisão que contêm os registros de colisão de uma mesma página formam uma lista ligada, e esta lista é encabeçada pela página original desses registros. Portanto, a região de colisão de uma página consiste nas páginas de colisão que pertencem à lista encabeçada por ela. Na figura abaixo estão representados dois arquivos. Um contém as páginas do espaço de endereçamento e o outro as páginas de colisão. Representa-se também nesta figura a região de colisão da página x.

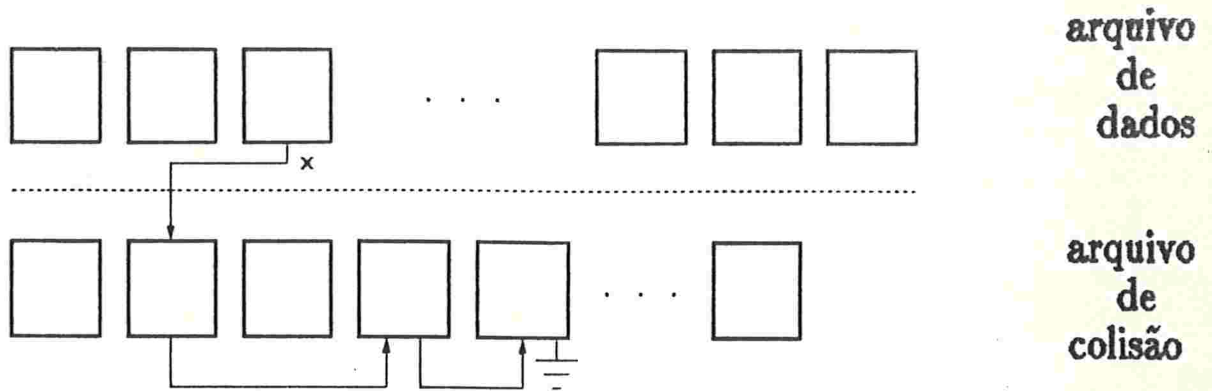

figura 2.1_ tratamento dos registros de colisão

Quando a colisão é tratada dessa forma, dizemos que o seu tratamento é feito pelo *encadeamento das páginas de colisão*. Mas se as páginas de colisão possuem capacidade igual a 1, dizemos que a colisão é tratada pelo *encadeamento de listas de colisão*.

Se os registros de colisão permanecem no arquivo de dados, o tratamento da colisão pode ser feito pelo *encadeamento dos registros de colisão* ou pelo *"open addressing"*.

Quando a colisão é tratada pelo encadeamento dos registros de colisão, cada página do arquivo encabeça uma lista ligada dos seus registros de colisão. Cada registro de colisão possui um campo extra, que aponta para outro registro de colisão de sua página original. Cada página do arquivo também possui um campo extra, que aponta para o primeiro registro de colisão da lista. Ao ocorrer colisão em uma página p, o seu registro de colisão é armazenado na página $p+1$ se esta contém espaço; caso contrário tenta-se colocá-lo na página $p+2$; se esta estiver cheia, tenta-se a página $p+3$, e assim por diante. Ao encontrar-se uma página que contém espaço para acolhê-lo, armazena-se nela, e esse registro é colocado na lista ligada dos registros de colisão da página p.

No "open addressing" não há ponteiros entre os registros de colisão e sua página original. A ligação entre eles é feita através de uma função, chamada de *função de colisão*. Cada página é associada a um conjunto ordenado de páginas pertencentes ao espaço de endereçamento do arquivo. Sejam p o número de uma página do espaço de endereçamento do arquivo, e $\{p_1, p_2, \dots p_m\}$ o conjunto ordenado das páginas associado a p pela função de colisão, onde p_i pertence ao espaço de endereçamento do arquivo. Os registros de colisão da página p são armazenados na página p_1 enquanto

esta página não estiver cheia. Quando a página p_1 ficar cheia, passa-se a colocar os registros de colisão na página p_2, depois na página p_3 e assim por diante. Ou seja, o registro de colisão é armazenado na página que contém espaço para acolhê-lo, e que é a página de menor ordem contida no conjunto imagem de sua página original, pela função de colisão. Note-se que, para procurar um registro de colisão da página p deve-se ler a página p_1; se ele não está armazenado em p_1 deve-se ler a página p_2, e assim por diante, até encontrá-lo ou concluir que o registro não está armazenado no arquivo. O registro não está armazenado no arquivo se ele não é encontrado nas páginas $p_1, \ldots, p_i$, onde $i \leq m$, e as páginas $p_1, \ldots, p_{i-1}$ estão cheias mas página p_i não está. Assim, a região de colisão de uma página p é formada pelas primeiras i páginas pertencentes ao conjunto associado a p pela função de colisão. Se esta função for pseudo-aleatória, então o esquema de "open addressing" é conhecido como "*random probing*". O esquema mais simples de "open addressing" é chamado de "*linear probing*". A função que o implementa é aquela que associa a cada página p do espaço de endereçamento, o conjunto $\{p+1, p+2, \ldots, n, 1, \ldots, p-1\}$, onde n é o número da páginas do arquivo. O tratamento da colisão pelo "linear probing" é bastante satisfatório quando o arquivo não está muito cheio.

A implementação do tratamento das colisões pelo encadeamento de páginas de colisão é bastante simples, mas a utilização do espaço em disco é melhor no "open addressing", pois no primeiro caso é alocada uma área adicional separada para conter os registros de colisão.

Uma das grandes dificuldades na técnica de hashing é encontrar "funções de hashing" que preservem a ordem das chaves. Como essas funções fazem a distribuição aleatória dos registros pelas páginas do arquivo, fazer o acesso a todos os registros de forma sequencial, segundo a ordem das chaves, tem baixo desempenho. Nos métodos existentes, o que normalmente se consegue é fazer acesso rápido aos registros de maneira sequencial, segundo a ordem do conjunto imagem das chaves pela função de hashing, ou seja, dados um registro r com chave k e a função de hashing H encontra-se rapidamente o registro r' do arquivo que possui com chave k', tal que H(k) e H(k') estão ordenados e não há registros no arquivo com chaves intermediárias. O problema é que k' não é conhecido, isto é, uma busca sequencial pelas chaves exige, para eficiência, o conhecimento dos valores das chaves, o que não acontece nos métodos de Hashing. Isto é encontrado somente em métodos sequencial-indexados, como a B-árvore

[13, 63]. A B-árvore é um método de organizar arquivos que mencionaremos neste trabalho, sem descrevê-lo. Assumiremos que o leitor possui o conhecimento desse método.

Capítulo III

HASHING ESTÁTICO

No hashing estático o tamanho do arquivo é fixo por que a função que endereça os registros no arquivo é inalterada. Na criação do arquivo, é alocado o espaço físico para conter os seus registros, e a função de endereçamento utiliza esse tamanho para calcular o endereço deles. Quando houver necessidade de mais espaço o arquivo é reorganizado, ou seja, primeiramente aloca-se uma outra área física, a qual é maior que a anterior, e depois deve-se inserir todos os registros nesta nova área utilizando outra função para endereçá-los. Isto é indesejável pois envolve muitos acessos ao disco, além do cálculo da função de endereçamento para todos os registros. Como uma boa estimativa do tamanho de um arquivo é uma tarefa difícil, podem ocorrer frequentes reorganizações. Isto levou os pesquisadores ao dessinteresse pela técnica de hashing estático. No entanto, Gonnet e Larson [23] apresentaram um método de hashing estático muito interessante. Nele a consulta a qualquer registro do arquivo é feita com um único acesso

ao disco. Este método é descrito na seção a seguir.

Observa-se que, a reorganização do arquivo é adiada se o tratamento dos registros de colisão é feito pelo encadeamento de páginas de colisão. Isto por que o espaço físico alocado para armazenar os registros é maior neste do que usando o "open addressing". Lembre-se que, no "open addressing" os registros de colisão são armazenados no próprio arquivo que contém os dados, e este não é expandido; enquanto que, no encadeamento de páginas de colisão eles são armazenados no arquivo de colisões que pode ser expandido à vontade, pois as suas páginas são encadeadas. Mas por outro lado, se for tolerada grande quantidade de colisões no arquivo, temos que o desempenho das suas operações torna-se muito baixo, devido ao grande número de leituras de páginas.

3.1_ MÉTODO DOS SEPARADORES [23]

3.1.1_ Descrição geral do método

Será denotada por n a quantidade de registros contidos num arquivo que possui m páginas de capacidade b cada uma.

Além da área em disco, usada para armazenar o arquivo, este método mantém em 'memória' interna uma tabela denominada de *tabela dos separadores*. Cada entrada dessa tabela é chamada de *separador* e há um separador para cada página do arquivo. O separador i está associado à página i, onde $1 \leq i \leq m$.

A cada registro do arquivo estão associadas duas sequências de m inteiros cada, a qual chamaremos de *sequência das assinaturas* e *sequência das páginas investigadas*. Elas são determinadas univocamente pela chave do registro. Por isso, as sequências das assinaturas e das páginas investigadas de um registro com chave k, são denotadas respectivamente por $S(k) = (s_1(k), s_2(k), \ldots, s_m(k))$ e $H(k) = (h_1(k), h_2(k), \ldots, h_m(k))$. A implementação de $H(k)$ e $S(k)$ será dada quando for descrita a função de endereçamento na seção 3.1.2.

A sequência $H(k)$ é usada para definir a ordem das páginas que são verificadas quando se faz a procura do registro de chave k. Portanto $\{h_1(k), \ldots, h_m(k)\}$ é uma permutação do conjunto $\{1, \ldots, m\}$.

A sequência $S(k)$ é uma sequência de inteiros não negativos. O componente $s_i(k)$ é chamado de *assinatura* do registro de chave k na página $h_i(k)$.

Para exemplificar, considere um arquivo com 4 páginas e um registro com chave k que possui $H(k) = (1, 4, 2, 3)$ e $S(k) = (9, 200, 3, 5)$. Então este registro possui assinatura 9 na página 1, assinatura 200 na página 4 e assim por diante. Temos também que, se for feita a procura deste registro no arquivo a página 1 é a primeira a ser verificada se o contém , em seguida será a página 4, depois a 2 e por último a 3.

A página que contém ou conterá o registro é definida como sendo a primeira página da sequência $H(k)$ que possui separador maior que a assinatura de registro naquela página. No exemplo dado, vamos supor que a tabela dos separadores é igual a $2, 7, 9, 10$. O endereço do registro é 2, pois na página 1 o registro possui assinatura 9, a qual é maior que o separador da página 1, o mesmo acontece na página 4.

O tratamento da colisão é feito pelo "open addressing", ou seja, os registros de colisão são armazenados no próprio espaço físico alocado para o arquivo e não há apontadores para eles. A colisão ocorre quando mais do que b registros são endereçados à mesma página. Ou seja, mais do que b registros possuem assinatura em uma página menor do que o separador dela. Uma maneira de resolver este problema, é decrementar o separador desta página até um valor para o qual garante-se que no máximo b registros continuarão endereçados a ela. Desta forma, não é alterado o endereço dos registros que estão armazenados nas outras páginas e, o novo separador irá isolar de maneira mais eficaz os registros que permanecerão na página. O novo valor para o separador da página cheia é igual a maior assinatura entre os registros endereçados a ela. Assim, os registros de colisão da página cheia são aqueles que possuem maior assinatura naquela página. Eles deverão ser reinseridos no arquivo após a modificação da tabela dos separadores. Notase que, poderá haver mais do que um registro de colisão em uma página que contém $b+1$ registros endereçados a ela, pois vários registros podem ter valor de assinatura máximo naquela página; e que após o tratamento da colisão a página cheia não terá mais registros de colisão, pois os seus endereços mudam após a modificação da tabela dos separadores. Assim,

uma página cheia pode tornar-se não cheia após o tratamento da colisão. Outra coisa que devemos notar é que o endereço de um registro pode mudar durante a existência do arquivo, mas a consulta a ele continua sendo feita com um único acesso.

3.1.2_ Função de endereçamento

O endereço de um registro com chave k é definido como sendo a primeira página da sequência $H(k)$ que possui separador maior que a assinatura do registro na tal página. Com isso, o valor inicial dado aos separadores influencia a distribuição dos registros pelas páginas do arquivo. Para que esta distribuição seja uniforme deve-se inicializar todos os separadores com o mesmo valor. No entanto, este valor não deve ser pequeno pois pode acarretar registros não endereçáveis num arquivo que contém espaço livre. Por exemplo, se os separadores forem inicializados com 0, todos os registros serão não endereçáveis, pois a assinatura de qualquer registro é um inteiro não negativo. Então, inicializa-se os separadores com o maior valor que uma assinatura pode assumir mais 1.

A forma de cálculo das sequências $H(k)$ e $S(k)$ também influem na distribuição dos registros pelas páginas do arquivo. Essa distribuição deve ser uniforme pois assim o número de colisões diminui. Larson e Kajla [25] definem a sequência das páginas investigadas por:

$$h_1(k) = k \bmod m + 1$$
$$h_i(k) = (h_{i-1}(k) + sl) \bmod m + 1 \qquad 1 < i \leq m$$

onde
$sl = (k \; div \; m) \bmod (m-2) + 1$

Para implementar a sequência das assinaturas, Larson e Kajla sugerem o uso de um gerador de números pseudo aleatórios. Em [25] a sequência $S(k)$ é definida por:

$$s_1(k) = (g(k) \; div \; c) \bmod 2^q$$
$$s_i(k) = (g(s_{i-1}(k)) \; div \; c) \bmod 2^q \qquad 1 < i \leq m$$

onde

g é o gerador de números pseudo-aleatórios
$c = 8191$
q = número de bits envolvidos na assinatura

Definindo as sequências dessa forma, temos que elas são facilmente calculadas, e cada uma das suas componentes dependem apenas da anterior. Dessa forma, o algoritmo de endereçamento não necessita de espaço extra para a sua implementação.

Algoritmo 3.1.2_ calcula o endereço de um registro com chave k em um arquivo organizado pelo método dos separadores.

entrada :
    k : chave do registro
saída :
    ass : assinatura do registro no seu endereço
    end : endereço do registro

```
{ para i de 1 a m faça {
    end ← hᵢ(k);
    ass ← sᵢ(k);
    se ass < separador de end então
      retorna (ass,end); }
  registro não endereçável; }
```

Note que um registro pode ser não endereçável em um arquivo. Isto acontece quando, a sua assinatura em todas as páginas do arquivo é maior que os separadores das páginas. Uma análise desse problema mostra que é difícil de acontecer isto, ou seja, a probabilidade de não endereçar um registro é muito pequena.

3.1.3_ Operações: consulta, eliminação e inserção

As operações de consulta e eliminação de um registro em um arquivo organizado pelo método dos separadores, são extremamente simples. Basta calcular o endereço do registro, ler a página e executar a operação se o registro estiver nela. Caso contrário, não existe o registro no arquivo.

Note que a operação de eliminação não envolve mudanças na tabela dos separadores.

A operação de inserção pode ser mais complicada. Para inserir um registro no arquivo deve-se calcular o seu endereço. Se não ocorrer colisão, ele é armazenado na sua página original e a operação está terminada. No entanto, se há colisão, o separador da página cheia é alterado e os seus registros de colisão são reinseridos no arquivo. Durante a reinserção podem acontecer novas colisões, que ao serem tratadas ocasionam outras colisões e assim por diante, sem que o processo termine. Este fenômeno é chamado de *cascata de registros.* Na prática, a cascata de comprimento infinito é evitada colocando-se um limite no número de páginas modificadas durante uma inserção. Se este limite for atingido então a inserção falha, ou seja, o registro não é armazenado no arquivo. O outro caso em que a inserção falha é quando não se consegue endereçar o registro. Isto acontece se, a assinatura do registro em todas as páginas do arquivo é maior que o separador da página.

Capítulo IV

HASHING DINÂMICO

No hashing dinâmico a área em disco alocada para o arquivo é variável. Após executar no arquivo uma série de inserções, pode haver a necessidade de mais área em disco, para evitar o acúmulo de colisões. Quando isso acontece, uma nova área é alocada no fim (sob o ponto de vista lógico) do arquivo. A função de endereçamento é alterada para associar registros à nova área alocada, ou seja , essa área passa a pertencer ao espaço de endereçamento do arquivo. Alguns registros, armazenados em poucas páginas do arquivo, mudam de endereço. Eles passam a ser endereçados para a nova área, sendo então transferidos para ela. As páginas onde esses registros estavam armazenados são chamadas de *páginas divididas*, pois os seus registros são distribuídos entre elas e a nova área alocada. É importante salientar que a quantidade de páginas divididas, em cada expansão do arquivo, deve ser pequena. Caso contrário, esse processo envolveria muitos acessos ao disco, sendo então muito caro. O processo de crescimento do

arquivo é chamado de *expansão do arquivo*, e *arquivo expandido* é aquele resultante do processo de expansão.

Ao eliminar muitos registros do arquivo, pode ocorrer excesso de área alocada. Quando isso acontece, uma área do arquivo é liberada. A função de endereçamento é modificada para a nova situação do arquivo, ou seja a área liberada é retirada do espaço de endereçamento do arquivo. Os registros armazenados nesta área são reinseridos no arquivo , considerando o novo espaço de endereçamento. Este processo é chamado de *contração do arquivo*, e *arquivo contraído* é o resultado da contração .

Existem dois critérios para fazer a expansão ou contração do arquivo, chamados de divisão controlada e divisão não controlada.

Na *divisão não controlada* o arquivo é expandido sempre que ocorre uma colisão. Este critério é usado, se a função de endereçamento distribui uniformemente os registros pelas páginas do espaço de endereçamento do arquivo. Assim, se ocorrer colisão em uma página podemos concluir que as outras contêm pouco espaço vazio, e é razoável crescer o arquivo neste momento, pois o seu fator de ocupação (def.1.13) deve estar próximo de 1. A contração do arquivo usando a divisão não controlada, ocorrerá se uma certa área do arquivo puder ser liberada, e os seus registros puderem ser reinseridos no novo espaço de endereçamento do arquivo, sem causar colisão.

Na *divisão controlada*, a expansão do arquivo ocorre quando certa porcentagem da área alocada a ele já foi preenchida. Este preenchimento pode ser medido pelo fator de ocupação do arquivo (def 1.13), ou pelo número de registros que foram inseridos no arquivo, após a última expansão. Assim, as inserções são executadas no arquivo até que o seu fator de ocupação ou número de inserções atinge um limite pré-determinado. Quando isso acontecer o arquivo é expandido. Nota-se que, fazendo o controle pelo fator de ocupação do arquivo ou pelo número de inserções obtem-se o mesmo resultado. Por exemplo, se o fator limite de ocupação do arquivo é 90% e, na sua expansão é alocada uma página por vez, então esse limite é sempre atingido após inserir-se $\lceil \frac{9.b}{10} \rceil$ registros, depois da última expansão. É esperado, em termos probabilísticos, que esse limite seja diretamente proporcional à quantidade de colisões que ocorrem no arquivo, ou seja, quanto maior o fator de ocupação do arquivo, maior o número de colisões e portanto, pior o desempenho das operações sobre os registros. A contração do arquivo na divisão controlada ocorre se certa quantidade

de páginas pode ser liberada, e o seu fator de ocupação, considerando o espaço de endereçamento sem essas páginas, for menor que o limite pré-determinado.

Há um maior interesse na expansão de um arquivo do que na sua contração. Isto acontece, porque a contração é feita somente para liberar área, visando ocupar melhor o espaço em disco. Enquanto que, a expansão evita o acúmulo de colisões no arquivo. Por isso, nesse trabalho dá-se maior enfase para a expansão de um arquivo.

O hashing dinâmico é chamado de *hashing dinâmico linear* se durante a expansão do arquivo aloca-se uma única página, ou seja o arquivo cresce linearmente sem causar grandes variações na quantidade de espaço alocado para ele. Assim, o fator de ocupação do arquivo mantém-se praticamente constante, se ele contém grande quantidade de páginas alocadas.

A forma como a função de endereçamento é implementada também divide os métodos de hashing dinâmicos em dois grupos. *Hashing dinâmico indexado* é aquele em que a função de endereçamento usa uma tabela que possui tamanho proporcional ao número de páginas contidas no arquivo. A função de hashing é utilizada para mapear o conjunto das chaves no conjunto de entradas dessa tabela. O endereço do registro é calculado usando o conteúdo da entrada que está associada à sua chave, pela função de hashing. Para obter esse conteúdo, a função de endereçamento faz acessos ao disco se a tabela estiver nele armazenada. De posse desse conteúdo, é necessário somente mais 1 acesso ao disco para se chegar até o registro. A figura abaixo mostra essas dependências.

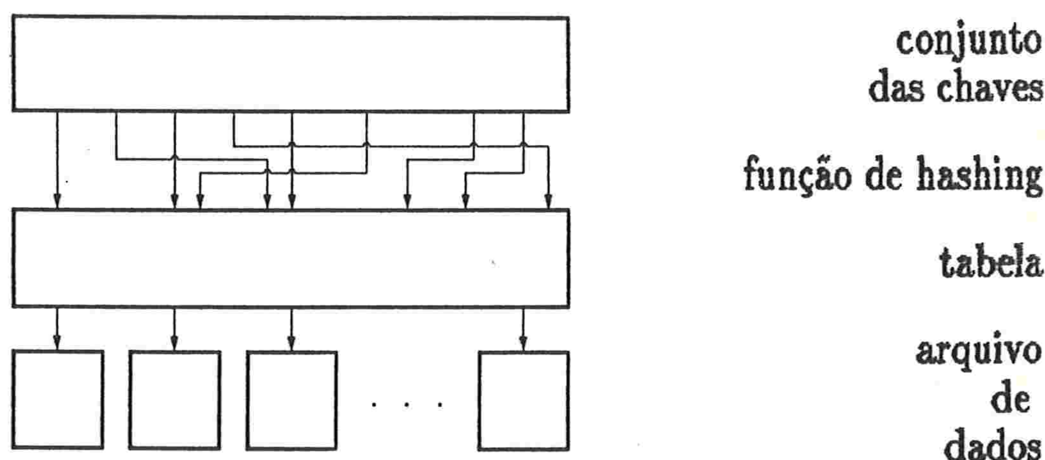

figura 4.1_ *modelo geral de hashing dinâmico indexado*

Quando a função de endereçamento não utiliza recurso indireto para calcular o endereço de um registro, diz-se que o hashing dinâmico é *não indexado.*

4.1_ HASHING LINEAR INDEXADO

No hashing linear indexado é alocada uma única página durante a expansão do arquivo, e a função de endereçamento utiliza para calcular o endereço, uma tabela que possui tamanho proporcional ao número de páginas nele contidas. Portanto, é inevitável o aumento do tamanho dessa tabela à medida que as expansões do arquivo ocorrem. Quando esta tabela não couber na memória interna, ela será armazenada em disco. Com isso, cada operação no arquivo irá requerer pelo menos dois acessos ao disco.

Tres métodos são descritos nesta seção. O Hashing dinâmico em árvore (HD) [18], o Hashing virtual VH0 [29,30] e o Hashing extensível (HE) [10].

O Hashing dinâmico em árvore lembra bastante a B-árvore. Como nesta, o HD também constroi uma árvore, onde os nós folha apontam para as páginas do arquivo. No entanto, o custo médio da procura com sucesso no HD é melhor do que na B-árvore. No HD, em média são necessários 3 acessos ao disco ([31]), enquanto que na B-árvore este número fica em torno de 4 ([2]). A utilização média do espaço alocado para o arquivo nos dois métodos, é por volta dos 70% ([31]).

No Hashing virtual VH0, a tabela usada pela função de endereçamento é de tamanho menor do que aquela usada pelo HD. No entanto, o VH0 exige que a função de endereçamento faça uma distribuição uniforme dos registros pelas páginas do arquivo. Se isto não ocorrer, essa tabela crescerá em ritmo exponencial, tornando-se maior do que aquela usada no HD. Este método faz em média 3 acessos ao disco ([31]) para executar uma procura com sucesso, quando o fator de utilização do espaço é em torno de 70% ([31]).

No Hashing extensível, a tabela usada pela função de endereçamento é bastante parecida com aquela do VH0. E o desempenho da procura com sucesso também alcança os mesmos números.

4.1.1_ HASHING DINÂMICO EM ÁRVORE (HD) [18]

4.1.1.1_ Descrição geral do método

Assim como na B-árvore, o método Hashing dinâmico em árvore constroi dois arquivos, chamados de arquivo de dados e arquivo de índices.

O arquivo de dados é dividido logicamente em páginas, e estas armazenam os registros. As páginas contidas no arquivo de dados constituem o seu espaço de endereçamento. O arquivo de índices contém um conjunto de árvores binárias, onde cada uma é chamada de árvore de índices. Os nós folha dessas árvores apontam para as páginas do arquivo de dados. Esses apontadores são usados pela função de endereçamento, para localizar o endereço do registro.

O arquivo de dados é criado com m páginas e o de índices com m árvores de índices, cada uma contendo apenas a raiz. Cada uma dessas raízes aponta para uma única página no arquivo de dados, e cada página é apontada por alguma raiz. Ou seja, há uma correspondência biunívoca entre as páginas do arquivo de dados e os nós folha das árvores de índices. A figura abaixo, ilustra a situação inicial de um arquivo com $m = 2$ páginas.

figura 4.1.1.1.1_ *criação de um arquivo, no HD, contendo 2 páginas*

Os registros são distribuídos uniformemente pelas páginas do arquivo de dados. Ao ocorrer colisão, o arquivo é expandido e na expansão uma nova página é alocada. Os registros pertencentes à página cheia são distribuídos entre ela e a nova página. Espera-se que metade deles permaneçam na página cheia e a outra metade seja endereçada para a nova página. Assim, nenhuma página terá registros de colisão armazenados no

arquivo. No arquivo de índices, o nó que apontava para a página cheia torna-se nó interno da árvore, isto é, deixa de ser nó folha, e ganha dois nós filhos. Um deles irá apontar para a página em que ocorreu colisão, e o outro para a nova página do arquivo. Nota-se que, apenas os nós folha da árvore de índices é que apontam para páginas do arquivo de dados.

Através da função de hashing localiza-se a árvore de índices e percorre-se o caminho nesta árvore, da raiz até o nó folha que contém o apontador para a página onde se encontra o registro.

4.1.1.2_ Processo de expansão e contração

O processo de expansão de um arquivo organizado pelo Hashing dinâmico em árvore é muito simples. O arquivo é expandido sempre que ocorre uma colisão. Na expansão do arquivo uma nova página é alocada, e passa a pertencer ao seu espaço de endereçamento. A função de endereçamento é naturalmente alterada para associar registros à nova página. A página dividida é a página cheia. Alguns registros dela são endereçados à nova página, sendo então transferidos para ela. O arquivo de índices é modificado. O nó folha que apontava para a página cheia é transformado em nó interno na árvore de índices, e ganha dois nós filhos. O nó filho esquerdo aponta para a página dividida, e o outro nó aponta para a nova página alocada. A figura abaixo, ilustra um arquivo criado com 2 páginas e que já passou por 4 expansões. A primeira colisão ocorreu na página 1. Na segunda expansão, a página 2 foi dividida. E as duas últimas colisões ocorreram na página 4.

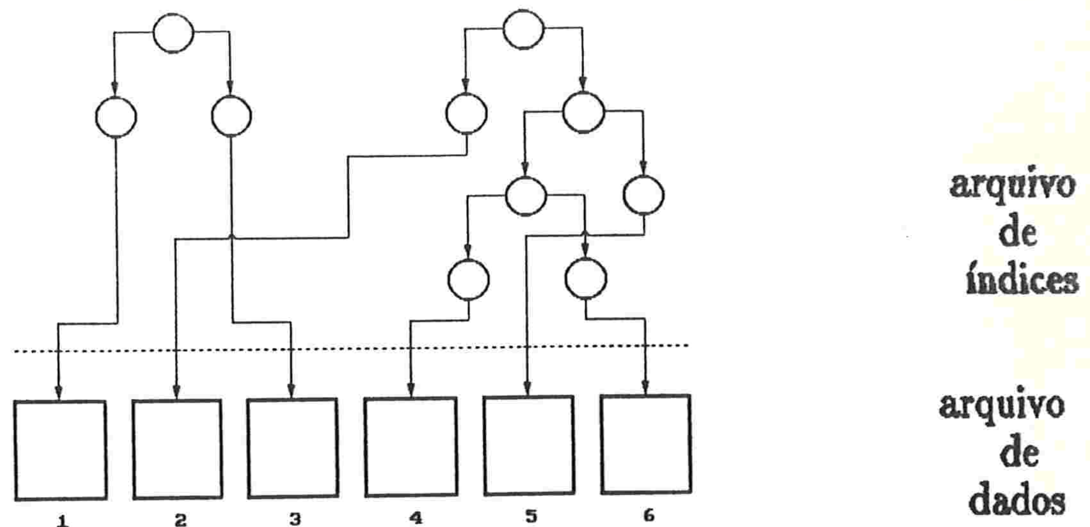

figura 4.1.1.2.1_ *expansão de um arquivo organizado pelo HD*

Algoritmo 4.1.1.2_ executa a expansão de um arquivo organizado pelo Hashing dinâmico em árvore

entrada :

P : número da página a ser dividida
Q : quantidade de páginas contidas no arquivo

comentário : a página $Q+1$ é alocada. Alguns registros da página P são transferidos para a página $Q+1$. Para saber quais dos registros contidos na página dividida são os transferidos, calcula-se o seu endereço (seção 4.1.1.3) considerando a nova página pertencente ao espaço de endereçamento do arquivo. Os registros com endereço na nova página, são os transferidos. Na seção 4.1.1.3, é dada a forma de localizar a árvore de índices, e o nó folha, dessa árvore que contém o apontador para a página dividida. O algoritmo abaixo supõe que o arquivo de índices está residente em memória interna.

```
{ alocar a página Q +1 no arquivo de dados;
  ler página P;
  para cada registro de P faça {
    calcular o endereço do registro considerando a página Q + 1;
    se este endereço mudou
      então escrever registro na página Q + 1
      senão escrever registro na página P; }
  localizar nó folha no arquivo de índices que aponta para P;
```

criar dois nós filhos para esse nó;
colocar no nó filho esquerdo o apontador para página P;
colocar no nó filho direito o apontador para página $Q + 1$;
escrever página P no arquivo de dados;
escrever página Q +1 no arquivo de dados;
incrementar Q; }

Vamos chamar de páginas irmãs aquelas que são apontadas por nós irmãos em uma árvore de índices. Por exemplo, na figura 4.1.1.2.1 as páginas 4 e 6 são irmãs.

Na contração do arquivo uma página é liberada. Isto ocorre quando, o número de registros armazenados nas páginas irmãs é menor ou igual à capacidade de uma página. Os registros são agrupados na página apontada pelo nó folha esquerdo, e a outra é liberada. Por exemplo, se a quantidade de registros armazenados nas páginas 4 e 6 for menor que a capacidade da página 4, agrupam-se todos os registros na página 4 e libera-se a página 6. No arquivo de índices, os nós folha que apontam para essas páginas são liberados, e o nó interno que apontava para eles passa a ser nó folha, apontando para a página que permanece no arquivo. A figura seguinte ilustra o arquivo da figura 4.1.1.2.1 após liberação da página 6.

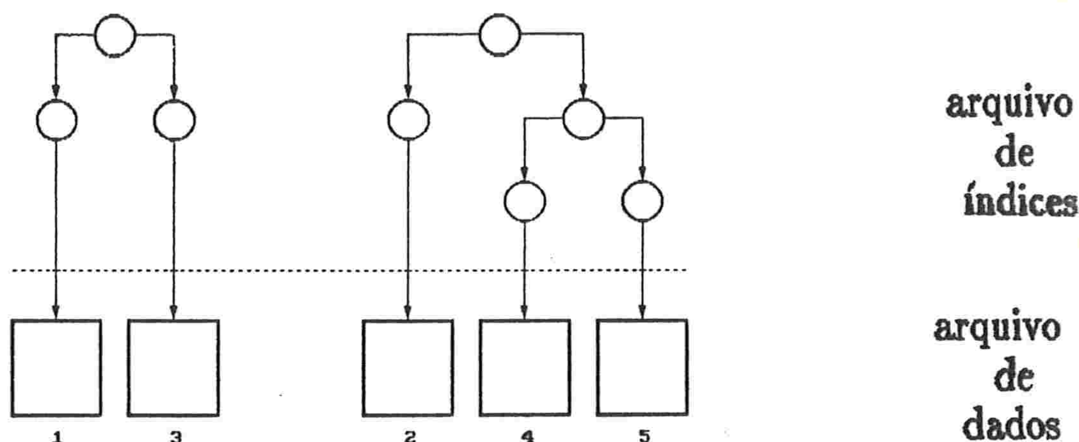

figura 4.1.1.2.2_ *contração de um arquivo organizado pelo HD*

4.1.1.3_ Função de endereçamento

Para endereçar um registro com chave *k*, o algoritmo de endereçamento do Hashing dinâmico em árvore usa duas funções de hashing, as

quais denotamos por H e C.

Com o auxílio da função de hashing H, este método escolhe entre as m árvores de índices, aquela que contém o nó folha que aponta para o endereço do registro. Esta função H pode ser implementada por uma função de hashing comum que mapeia o conjunto das chaves no conjunto dos inteiros positivos menores ou iguais a m, ou seja,

$$H : K \rightarrow \{1, \ldots, m\}$$

Através da função de hashing C, caminha-se na árvore H(k) até alcançar o nó folha que contém o endereço do registro. Esta função C pode ser implementada, como uma função pseudo-aleatória que mapeia o conjunto das chaves no espaço das sequências binárias infinitas, ou seja

$$C : K \rightarrow \{(c_1, c_2, \ldots) \text{ onde } c_i \in \{0, 1\} \text{ e } i \in N\}$$

A sequência binária define o caminho a ser percorrido na árvore de índices da seguinte forma:
a raiz da árvore de índices é o primeiro nó do caminho;
o filho esquerdo do (i−1)-ésimo nó do caminho pertence ao caminho se $c_i = 0$, caso contrário, o filho direito é escolhido para pertencer ao caminho.

Se um registro com chave k possui H(k)= 2 e C(k)= $(1, 0, 1, 0, \ldots)$ então o seu endereço no arquivo ilustrado na figura 4.1.1.2.1 é a página 6.

Algoritmo 4.1.1.3_ calcula o endereço de um registro em um arquivo organizado pelo Hashing dinâmico em árvore.

entrada :
  k : chave do registro
saída :
  endereço do registro

comentário : este algoritmo supõe que o arquivo de índices está residente em memória interna.

{ L ← H(k);
  enquanto L *não* é nó folha *faça*
    L ← *se* próxima componente de C(k)= 0
      então filho esquerdo de L

senão filho direito de L;
retorna (conteúdo do nó folha L); }

4.1.1.4_ Operações: inserção, eliminação e consulta

As operações de eliminação e consulta de um registro no Hashing dinâmico em árvore são extremamente simples. Basta calcular o endereço do registro e executar a operação. Se a operação que está sendo executada é a eliminação, deve-se verificar a possibilidade de contrair o arquivo. Ao liberar a página, pode ocorrer a possibilidade de outra contração. Isto ocorre, se a quantidade de registros contidos na página que permanece no arquivo mais os registros armazenados na sua nova página irmã, é menor que a capacidade de uma página. A contração no arquivo deve acontecer até que, não seja mais possível agrupar os registros das páginas irmãs, ou o nó raiz da árvore seja atingido.

Para inserir um registro no arquivo, calcula-se o seu endereço armazenando-o em sua página original, se esta não está cheia. Caso contrário, o arquivo é expandido. Durante essa expansão pode haver necessidade de outra expansão. Isto acontece se todos os registros endereçados à página cheia são enviados para a mesma página durante a expansão, ou seja a função de endereçamento não consegue quebrar o conjunto dos registros endereçados à página cheia em dois conjuntos, aqueles que permanecem na página cheia, e naqueles que irão para a nova página alocada. Este processo de expansão pode continuar infinitamente. Para evitá-lo, é fixado um limite máximo de expansões para o arquivo durante uma inserção. Se o limite for atingido, o registro não é armazenado no arquivo. Neste caso, dizemos que a inserção falhou. Em [18] é mostrado que a probabilidade disto ocorrer é muito pequena. Nota-se que se a função de endereçamento conseguir sempre quebrar o conjunto dos registros de uma página nas duas páginas resultantes da expansão, então o número de nós da árvore é $2^n - 1$, onde n é o número de páginas do arquivo.

4.1.1.5_ Alterações no Hashing dinâmico em árvore

As modificações sugeridas em [52] fazem aumentar, em média, o fator de ocupação do arquivo, e eliminam a probabilidade de uma inserção

falhar. Consegue-se isto adiando a divisão de uma página. Uma página é dividida, quando βb registros forem endereçados a ela, onde β é um real maior que 1 e b é a capacidade de cada página. Com isso, é liberada a existência de registros de colisão das páginas do arquivo. Esses registros são armazenados nas páginas de colisão, ou seja o tratamento da colisão é feito pelo encadeamento das páginas de colisão.

A Divisão Linear é um outro método que também está descrito em [52]. As árvores de índices construídas no HD são, neste método, substituídas por filas. Para cada árvore de índices corresponde uma fila, e cada fila ocupa menos espaço do que a árvore de índices correspondente a ela. Cada entrada da fila aponta para uma página do arquivo. Através de uma função de hashing os registros são distribuídos entre as filas existentes. Com outra função de hashing, os registros endereçados a uma fila são distribuídos entre as páginas apontadas pelas entradas dessa fila. Quando um número pré-determinado de registros forem endereçados a uma fila, o arquivo é expandido. Uma nova página é alocada no arquivo e a página, apontada pela primeira entrada dessa fila, é dividida. A primeira entrada é removida da fila e duas novas entradas são alocadas no fim. Uma dessas novas entradas aponta para a página que foi dividida e a outra para a nova página.

4.1.2_ HASHING VIRTUAL VH0 [30]

4.1.2.1_ Descrição Geral do Método

Quando um conjunto de informações é armazenado em disco usando o Hashing virtual VH0, são criados dois arquivos e uma tabela de apontadores. Os arquivos são o arquivo de dados e o arquivo de colisões. O tratamento da colisão é feito pelo encadeamento das páginas de colisão. A tabela dos apontadores contém apontadores para as páginas do arquivo de dados. A cada página do espaço de endereçamento corresponde uma entrada nessa tabela, contendo um apontador para a página. A função de endereçamento utiliza esses apontadores para calcular o endereço do registro.

O arquivo de dados é criado com m páginas e a tabela dos apontadores com m entradas. A i-ésima entrada aponta para a i-ésima página, onde $1 \leq i \leq m$. A figura abaixo ilustra um arquivo criado com $m = 5$ páginas. O arquivo de colisões não é mostrado nessa figura.

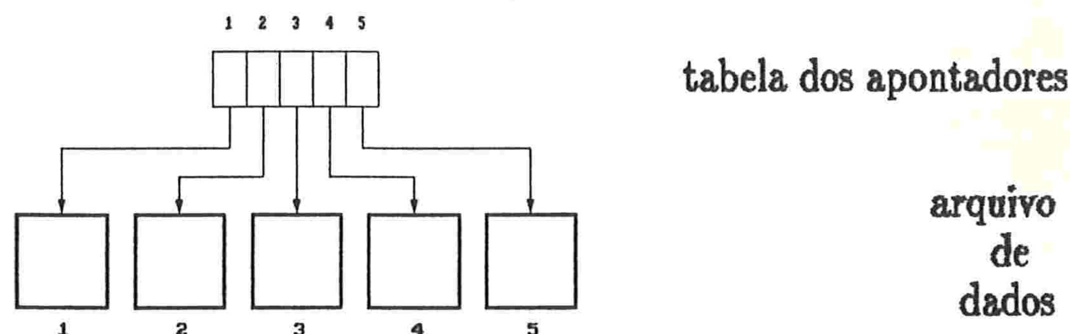

figura 4.1.2.1.1_ ***arquivo criado com*** **5** ***páginas pelo método VH0***

Os registros são uniformemente distribuídos pelas páginas do arquivo de dados. Ao ocorrer colisão, é feito o tratamento se o fator de ocupação do arquivo não atingiu um limite pré-determinado α. Os registros de colisão de uma página são aqueles a ela endereçados quando está cheia. Eles são armazenados na região de colisão da sua página original, que se encontra no arquivo de colisões. Este arquivo é criado durante o tratamento da primeira colisão.

O arquivo é expandido quando ocorrer uma colisão e o seu fator de ocupação alcançou α. Na expansão, uma nova página é alocada no fim do arquivo de dados. A página em que ocorreu colisão é dividida. Os registros a ela endereçados são distribuídos entre ela e a nova página alocada. A tabela dos apontadores é modificada para que uma de suas entradas aponte para a página mais recentemente alocada.

4.1.2.2_ Processo de expansão

O critério para expandir um arquivo no Hashing virtual VH0, é uma mistura da divisão controlada com a não controlada. O arquivo é expandido se ao ocorrer uma colisão o seu fator de ocupação alcançou o limite α. O algoritmo permite usar somente a divisão não controlada, isto é haverá expansão sempre que ocorrer uma colisão; nesse caso garante-se um só acesso ao disco para cada busca, a menos de mais um acesso aos apontadores, se estes estiverem também em disco.

A página alocada na expansão é incluída no espaço de endereçamento do arquivo. Para isso, uma entrada da tabela dos apontadores deve apontar para ela. Antes de definir o número dessa entrada, é apresentado o conteúdo das entradas da tabela, que é esperado pela função de endereçamento.

Quando a tabela dos apontadores possui m entradas, aquela de número i aponta para a página i, onde $1 \leq i \leq m$. Por exemplo, se $m = 5$ tem-se que a entrada 1 aponta para a página 1, a 2 para a página 2, a 3 para a 3, e assim por diante (ver figura 4.1.2.1.1). Se a tabela contém $2m$ entradas, aquela de índice i aponta para a página i, para $1 \leq i \leq m$. Nota-se que essas entradas são iguais às correspondentes entradas da tabela, quando o seu tamanho é m. A entrada de número $m + i$, onde $1 \leq i \leq m$, aponta para a página alocada durante a primeira divisão da página i. Tomando $m = 5$ e a tabela dos apontadores com 10 entradas, a página i é apontada pela entrada i, para $1 \leq i \leq 5$. A entrada 6 aponta para a página alocada durante a primeira divisão da página 1 se já tiver ocorrido, a 7 para a página alocada durante a primeira divisão da página 2 se já tiver ocorrido, e assim por diante. Quando a tabela possui $4m$ entradas, as primeiras $2m$ entradas contêm o mesmo conteúdo da correspondente entrada da tabela, quando o seu tamanho é $2m$, ou seja, a entrada i aponta para a página i, onde $1 \leq i \leq m$, aquela de índice $m + i$, para $1 \leq i \leq m$, aponta para a página alocada durante a primeira divisão da página i. A entrada $2m + i$, onde $1 \leq i \leq m$, aponta para a página alocada durante a segunda divisão da página i, e a entrada $3m + i$, onde $1 \leq i \leq m$, aponta para a página alocada durante a primeira divisão da página apontada pela entrada $m + i$. Mas, a página apontada pela entrada $m + i$ é aquela que foi alocada durante a primeira divisão da página i. Exemplificando, sejam $m = 5$ e a tabela dos apontadores com 20 entradas. A entrada i, se $1 \leq i \leq 10$, já foi descrita anteriormente. A de índice 11 aponta para a página alocada durante a segunda divisão da página 1, a 12 aponta para aquela que é alocada durante a segunda divisão da página 2, e assim segue até a entrada 15. Aquela de índice 16 contém o apontador para a página alocada durante a primeira divisão da página apontada pela entrada 6, como a entrada 6 aponta para a página alocada durante a primeira divisão da página 1, tem-se que a entrada 16 aponta para a página alocada durante a primeira divisão da página, que foi alocada durante a primeira divisão da página 1. A entrada 17 aponta para a página alocada na primeira divisão da página apontada pela entrada

7, e assim por diante. De maneira geral, se a tabela dos apontadores possui $2^j m$ entradas, onde $j \in N$, o conteúdo da entrada i, para $1 \leq i \leq 2^{j-1}m$, é igual ao da entrada i da tabela com $2^{j-1}m$ entradas. A entrada $2^{j-1}m + i$, onde $1 \leq i \leq m$, aponta para a página que é alocada durante a j-ésima divisão da página i. A de índice $2^{j-1}m + m + i$, para $1 \leq i \leq m$, aponta para a página alocada durante a $(j - 1)$-ésima divisão da página apontada pela entrada $m + i$. A entrada $2^{j-1}m + 2m + i$, quando $1 \leq i \leq m$, aponta para aquela alocada durante a $(j - 2)$-ésima divisão da página apontada pela entrada $2m + i$, e assim segue.

Sabendo qual o conteúdo esperado pela função de endereçamento para as entradas da tabela dos apontadores, tem-se que:
na primeira vez que uma página p é dividida, a página alocada é apontada pela entrada $2^n m + l$, onde a entrada l aponta para a página p, e n é o menor natural tal que $l \leq 2^n m$. Na segunda divisão da página p, aquela alocada é apontada pela entrada $2^{n+1}m + l$. A página alocada na terceira divisão da página p é apontada pela entrada $2^{n+2}m + l$, e assim por diante. Por exemplo, se $m = 5$, a página alocada na primeira divisão da página 3 é apontada pela entrada $2^0 m + 3 = 8$. Na segunda divisão da página 3, aquela alocada é apontada pela entrada $2^1 m + 3 = 13$. Se q é o número da página apontada pela entrada 8, a página alocada durante a primeira divisão da página q, é apontada pela entrada $2^1 m + 8 = 18$.

Os passos executados na expansão de um arquivo organizado pelo VH0, são dados a seguir. Suponha que a entrada l aponta para a página dividida, e que essa é a j-ésima divisão dessa página. A página alocada nessa expansão é apontada pela entrada $2^{n+j-1}m + l$, onde n é o menor natural tal que $l \leq 2^n m$. Se a tabela dos apontadores não possui essa entrada, duplica-se a quantidade de suas entradas. A entrada $2^{n+j-1}m + l$ aponta para a página alocada, e as outras novas entradas permanecem livres. Caso contrário, se a tabela possue essa entrada, basta fazê-la apontar para a nova página.

A figura abaixo ilustra o processo de expansão de um arquivo criado com 5 páginas. Na primeira expansão, a página 3 foi dividida. Como a tabela dos apontadores não possui a entrada de índice 8, dobrou-se o seu tamanho, ou seja, passou a ter 10 entradas. A entrada de índice 8 aponta para a página 6. Na segunda expansão do arquivo, a página 5 foi dividida. Como a entrada de índice 10 está vazia, passa a apontar para a página 7.

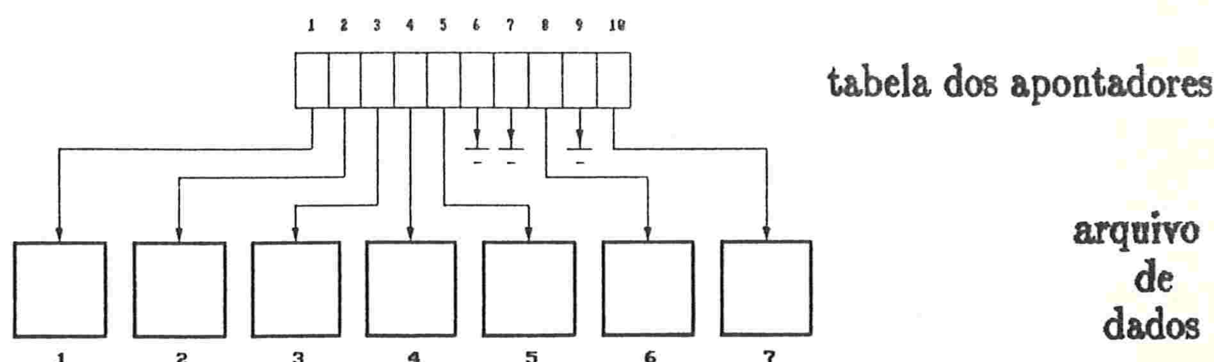

figura 4.1.2.2.1_ *expansão de um arquivo organizado pelo VH0*

Suponha que na próxima expansão do arquivo a página 3 é dividida novamente. Como esta é a segunda divisão da página 3, deve-se colocar na entrada 13, o ponteiro para a página 8. Mas, a tabela possui apenas 10 entradas, deve-se então duplicar o seu tamanho. Note que, após essa duplicação, a tabela dos apontadores conterá muito mais entradas do que o necessário. Isto por que muito mais registros foram endereçados à página 3 do que às outras páginas do arquivo de dados. Portanto, a função de endereçamento não está distribuindo uniformemente os registros pelas páginas. Logo, não haverá desperdício de espaço na tabela dos apontadores, se a função de endereçamento executar a distribuição uniforme dos registros, pelas páginas do arquivo de dados.

Para calcular o número da entrada da tabela dos apontadores, que aponta para a página alocada na expansão, é necessário saber quantas divisões sofreu a página que está sendo dividida. A quantidade de divisões de uma página, pode ser armazenada em um campo extra da página. No entanto, há uma maneira de se obter essa informação sem precisar armazená-la. Sejam $n \in N$, tal que a tabela dos apontadores contém $2^n m$ entradas, e l a entrada que aponta para a página dividida. Se $l > 2^{n-1}m$, a nova página é apontada pela entrada $2^n m + l$; como a tabela não possui essa entrada, duplica-se a quantidade de suas entradas. Quando $l \leq 2^{n-1}m$, verifica-se o conteúdo da entrada $2^{n-1}m + l$; se este é um ponteiro para alguma página do arquivo, a nova página deve ser apontada pela entrada $2^n m + l$, implicando também no crescimento da tabela. Mas se a entrada $2^{n-1}m + l$ está vazia, toma-se o conteúdo da entrada $2^{n-2}m + l$; se este não for vazio, a nova página é apontada pela entrada $2^{n-1}m + l$. Caso contrário, verifica-se o conteúdo da entrada $2^{n-3}m + l$, e assim segue a procura por uma entrada que não esteja vazia. Ao encontrá-la, aloca-se o ponteiro para

a nova página, na última entrada vazia entre aquelas que foram verificadas.

Algoritmo 4.1.2.2_ executa a expansão de um arquivo que está sendo organizado pelo hashing virtual VH0.

entrada :

P : número da página a ser dividida
Q : cardinalidade do espaço de endereçamento
j : inteiro j tal que, a tabela dos apontadores possui $2^j m$ entradas

comentários: supõe-se aqui a existência de um algoritmo que calcula o endereço de um registro, dada a sua chave. Este algoritmo será dado na seção 4.1.2.3, onde também é dada a forma de calcular o índice na tabela dos apontadores que aponta para uma página. O algoritmo abaixo supõe que a tabela dos apontadores se encontra residente em 'memória' interna.

{ alocar página Q+1 no arquivo de dados;
  ler os registros da página P;
  ler região de colisão da página P;
  calcular a entrada de índice l que aponta para P;
  se $l < 2^{j-1}m$ *e* entrada de índice $2^{j-1}m + l$ está vazia
    então {
      $u \leftarrow j - 1$;
      enquanto entrada de índice $2^u m + l$ está vazia *faça*
        decrementar u;
      entrada de índice $2^{u+1}m + l$ deve apontar para página Q+1; }
    senão {
      duplicar o tamanho da tabela dos apontadores;
      entrada de índice $2^j m + l$ deve apontar para página Q+1;
      incrementar j; }
  para cada registro r endereçado à P *faça* {
    calcular o endereço do registro r considerando o espaço de
      endereçamento com Q+1 páginas;
    se endereço de r é P
      então
        escrever registro r na página P ou na sua região de colisão
      senão
        escrever registro r na página Q+1 ou na sua região de colisão; }

escrever no arquivo de dados a página P;
escrever no arquivo de colisão a região de colisão de P;
escrever no arquivo de dados a página Q+1;
escrever no arquivo de colisão a região de colisão de Q+1;
incrementar Q; }

4.1.2.3_ Função de endereçamento

Através da função de hashing, o algoritmo de endereçamento localiza a entrada na tabela dos apontadores, que contém o endereço do registro.

A função de endereçamento utiliza um conjunto de funções de hashing, denotadas por $f_0, f_1, \ldots$. A função f_i, onde $i \in N$, mapeia o conjunto das chaves no conjunto $\{1, \ldots, 2^i m\}$; e para $i \geq 1$ elas obedecem a seguinte restrição:

$$f_i(k) = f_{i-1}(k) \quad \text{ou} \quad f_i(k) = f_{i-1}(k) + 2^{i-1}m \tag{0}$$

No início, a tabela dos apontadores contém m entradas e usa-se a função de hashing f_0. A entrada $f_0(k)$ aponta para a página onde o registro de chave k está armazenado. Quando a tabela dos apontadores dobra de tamanho, a função f_0 não cobre mais toda a tabela, pois ela associa as chaves a índices menores ou iguais a m. O algoritmo de endereçamento passa a utilizar as funções f_0 e f_1, para distribuir os registros pelas páginas do arquivo. A função f_1 endereça registros às páginas que sofreram uma divisão, e aquelas que foram alocadas nessas divisões, isto é, a página apontada pela entrada $f_1(k)$ armazena o registro de chave k, se ela foi dividida uma vez, ou foi alocada na primeira divisão da página apontada pela entrada $f_0(k) = f_1(k) - m$. Caso contrário, o endereço do registro encontra-se na entrada $f_0(k)$. A função $f_1(k) = f_0(k)$ ou $f_1(k) = f_0(k) + m$, pois a função f_1 é usada durante a expansão do arquivo, para distinguir os registros que permanecem na página dividida daqueles que são transferidos para a nova página. Lembre-se que, na expansão do arquivo, a página alocada é apontada pela entrada $f_0(k) + m$, onde k é a chave de um registro armazenado na página dividida. Para exemplificar, considere o arquivo ilustrado na figura 4.1.2.2.1. A função f_1 endereça registros às páginas $3, 5, 6$ e 7, pois

as páginas 3 e 5 sofreram uma divisão, e as páginas 6 e 7 foram alocadas nessas divisões. Para as páginas restantes, utiliza-se a função f_0. Após outro crescimento da tabela dos apontadores, toma-se mais uma função de hashing, a função f_2. O endereço de um registro de chave k, é apontado pela entrada $f_2(k)$, se esta não está vazia. Caso contrário, o endereço do registro é a página apontada pela entrada $f_1(k)$, se esta entrada aponta para alguma página do arquivo. Se ela também está vazia, o registro é armazenado na página apontada pela entrada $f_0(k)$. Lembre-se que, a entrada $f_2(k)$ não está vazia, se:

i- a página apontada pela entrada $f_1(k)$ foi dividida duas vezes, neste caso $f_0(k) = f_1(k)$;

ii- a página apontada pela entrada $f_1(k)$ passou por uma divisão e $f_1(k) = f_0(k) + m$;

iii- a página apontada pela entrada $f_2(k)$ foi alocada durante a divisão da página $f_1(k)$ e $f_2(k) > 2m$.

De uma maneira geral, define-se o endereço de um registro que possui chave k como sendo o conteúdo da entrada $f_l(k)$, onde ou a entrada de índice $f_{l+1}(k)$ não existe na tabela dos apontadores ou está vazia.

Uma família de funções que obedecem a restrição (0), é dada a seguir:

$$f_i(k) = (k \bmod (2^i m)) + 1 \qquad i = 0, 1, 2, \ldots$$

A figura abaixo ilustra o algoritmo de endereçamento do Hashing Virtual VH0 para essa família. A parte A da figura representa o arquivo antes da expansão, e a B após a divisão da página 3. Ele foi criado com $m = 5$ páginas, e a inserção do registro de chave 147, provocou a sua expansão. Os números contidos nas páginas, são as chaves dos registros nela armazenados. Cada página possui capacidade igual a 3. Na figura está indicada, em cada entrada da tabela dos apontadores, a função de hashing que o algoritmo de endereçamento utiliza, para associar registros à página apontada por essa entrada.

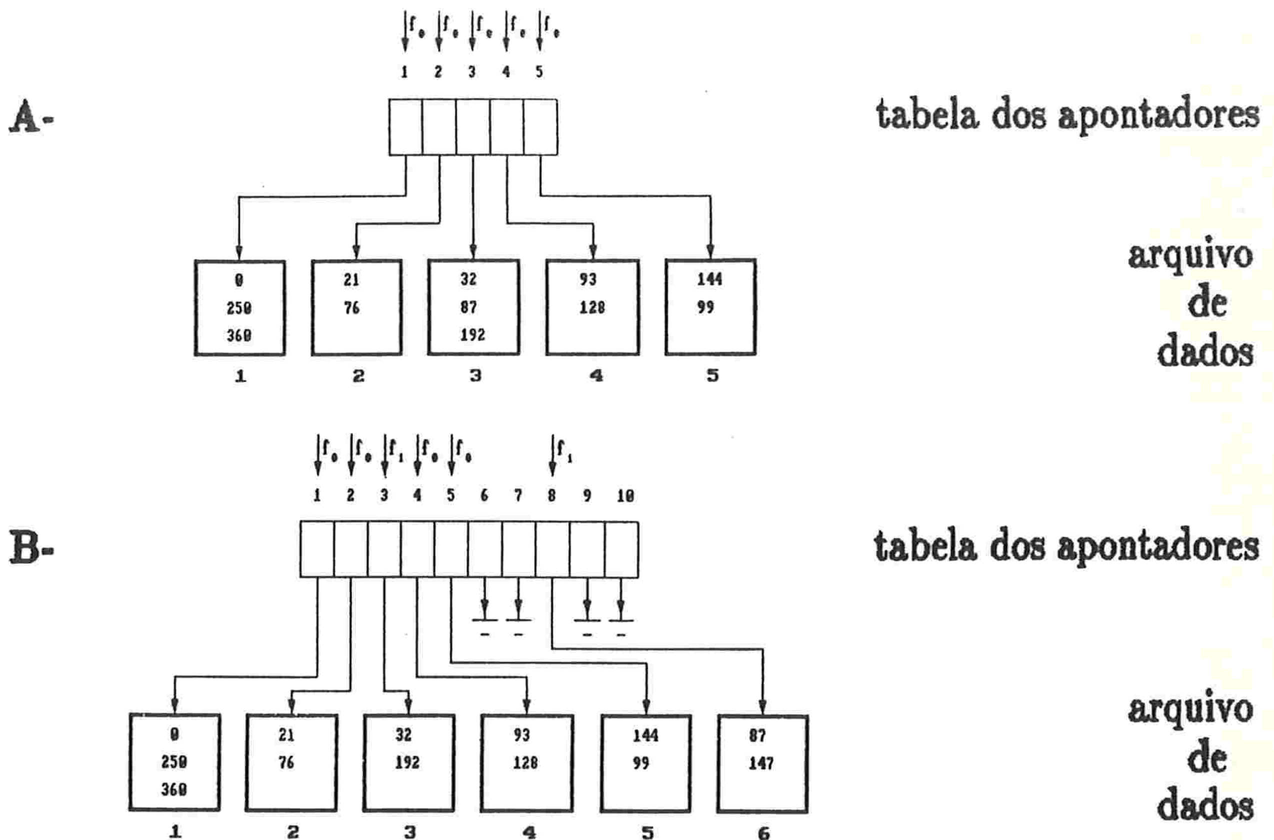

figura **4.1.2.3.1_** ***esquema geral do algoritmo de endereçamento do VH0***

Algoritmo 4.1.2.3_ calcula o endereço de um registro em um arquivo organizado pelo Hashing Virtual VH0.

entrada :

k : chave do registro

j : inteiro tal que, a tabela dos apontadores possui $2^j m$ entradas

saída :

endereço do registro

número da entrada na tabela, que aponta para o endereço do registro

comentários : o algoritmo abaixo assume que a tabela dos apontadores está residente em 'memória' interna.

{ $l \leftarrow f_j(k)$;
  enquanto entrada de índice l está vazia *faça* {
    decrementar j;
    $l \leftarrow f_j(k)$; }

retorna (conteúdo da entrada de índice l, l); }

4.1.2.4_ Operações : inserção e consulta

No arquivo organizado pelo Hashing virtual VH0, Litwin define como os registros são inseridos e consultados, mas não menciona a forma como são eliminados.

As operações de inserção e consulta são extremamente simples, basta calcular o endereço do registro, e executar a operação. Se esta for inserção e ocorrer colisão, verifica-se a possibilidade de expandir o arquivo.

4.1.3_ HASHING EXTENSÍVEL (HE) [10]

4.1.3.1_ Descrição geral do método

Quando um conjunto de informações é armazenado no disco, usando o Hashing extensível, são criados um arquivo, chamado de arquivo de dados, e também uma tabela chamada de diretório.

O arquivo de dados é dividido logicamente em páginas e os registros são endereçados a elas. Portanto, essas páginas constituem o espaço de endereçamento do arquivo. O diretório contém apontadores para as páginas do arquivo de dados. Cada página é apontada por pelo menos uma entrada do diretório. A função de endereçamento utiliza esse diretório para calcular o endereço do registro.

O arquivo de dados é criado com 2^n páginas e o diretório com 2^n entradas para algum $n \in N$. A i-ésima entrada aponta para a i-ésima página, onde $1 \leq i \leq 2^n$. A figura abaixo ilustra um arquivo criado com 4 páginas.

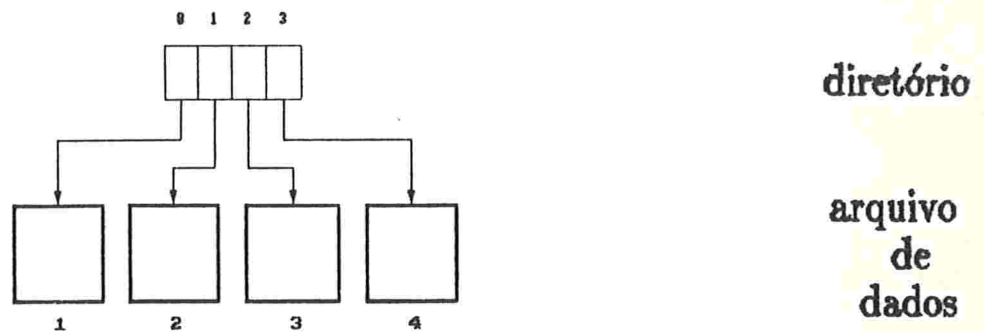

figura 4.1.3.1.1_ *arquivo criado pelo HE contendo 4 páginas*

Os registros são uniformemente distribuídos pelas páginas do arquivo de dados. Ao ocorrer colisão, o arquivo é expandido. Uma nova página é alocada no fim do arquivo de dados, e aquela em que ocorreu colisão, é dividida. Os seus registros são distribuídos entre ela e a nova página. O diretório é alterado, e uma das suas entradas apontará para a página mais recentemente alocada.

De uma maneira geral, a função de endereçamento funciona da seguinte forma: a função de hashing mapeia o conjunto das chaves num intervalo de inteiros, denotado por I . Este intervalo é particionado em m sub-intervalos, onde m é a quantidade de páginas do arquivo de dados. Cada sub-intervalo desses foi obtido por sucessivas divisões ao meio. Por exemplo, se $I = [0 \;..\; 7]$ a figura abaixo ilustra uma partição de I obtida dessa forma.

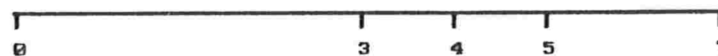

figura 4.1.3.1.2_ *partição do intervalo* $I = [0 \;..\; 7]$

Cada página do arquivo de dados é associada a um sub-intervalo. O diretório implementa a correspondência entre os sub-intervalos e as páginas. A figura abaixo ilustra tal situação.

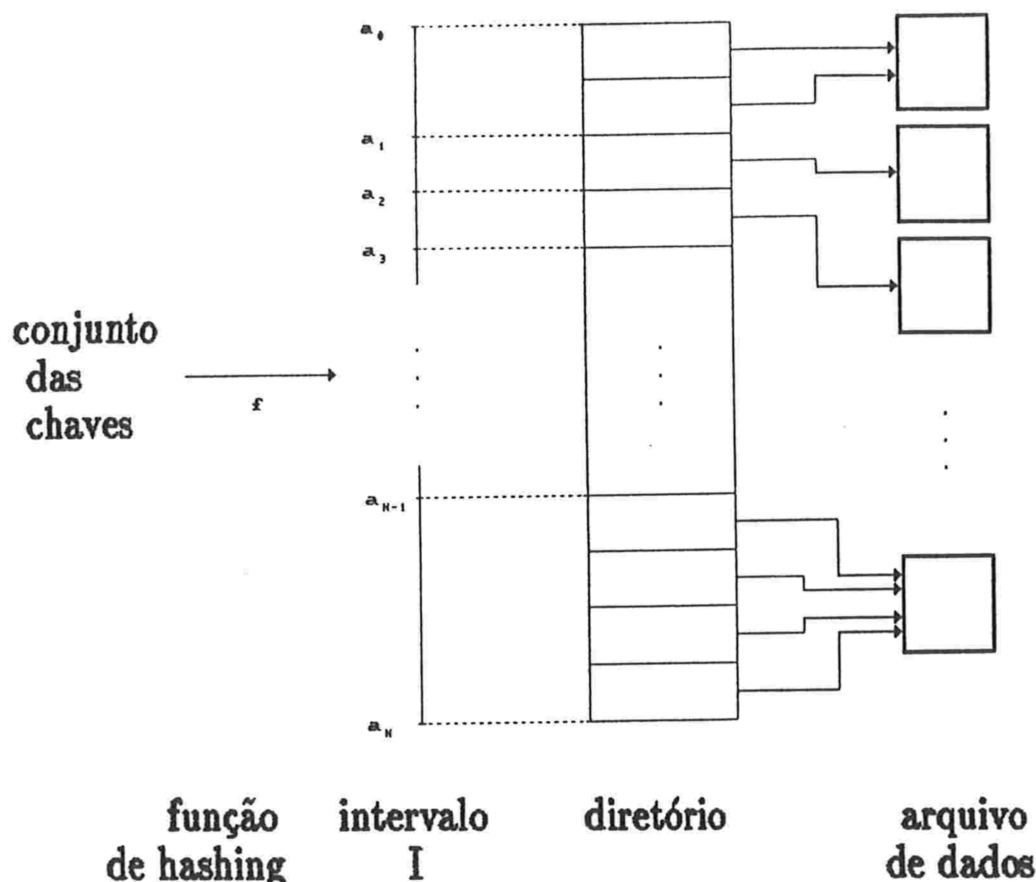

figura 4.1.3.1.3_ *modelo geral da função de endereçamento*

Ao ocorrer a colisão, o sub-intervalo correspondente à página cheia é dividido ao meio. Uma das suas metades é associada à página cheia e a outra à nova página.

4.1.3.2_ Processo de expansão e contração

O critério usado para expandir o arquivo organizado pelo Hashing extensível, é a divisão não controlada. Ao ocorrer uma colisão a página cheia é dividida.

Podemos distinguir duas configurações do diretório no momento da colisão:

i- existe uma única entrada que aponta para a página cheia;

ii- há mais de uma entrada apontando para a página cheia.

Vamos supor que há uma única entrada no diretório que aponta para a página cheia, e que seu índice é l. Neste caso, duplica-se a quantidade

de entradas do diretório. No novo diretório, a entrada de índice $2l$ irá apontar para a página que está sendo dividida, e a de índice $2l+1$ irá apontar para a nova página alocada. As outras páginas do arquivo de dados terão, no novo diretório, o dobro de entradas apontando para elas. Se a entrada de índice i apontava para uma dessas páginas antes da colisão, no novo diretório a página é apontada pelas entradas de índice $2i$ e $2i+1$. A figura dada a seguir ilustra a expansão no caso i para a figura 4.1.3.1.1. O arquivo foi criado com $m = 4$ páginas, e a página 2 foi dividida . Nota-se que, apenas as páginas 2 e 5 são apontadas por uma única entrada após a expansão.

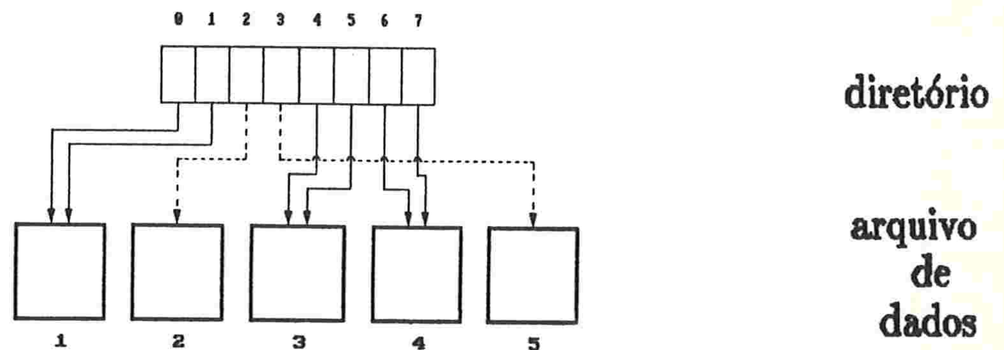

figura 4.1.3.2.1_ *expansão de um arquivo organizado pelo HE no caso i*

Vamos supor que a página 5 do arquivo ilustrado acima é dividida. A figura abaixo ilustra o diretório e o arquivo de dados após a expansão.

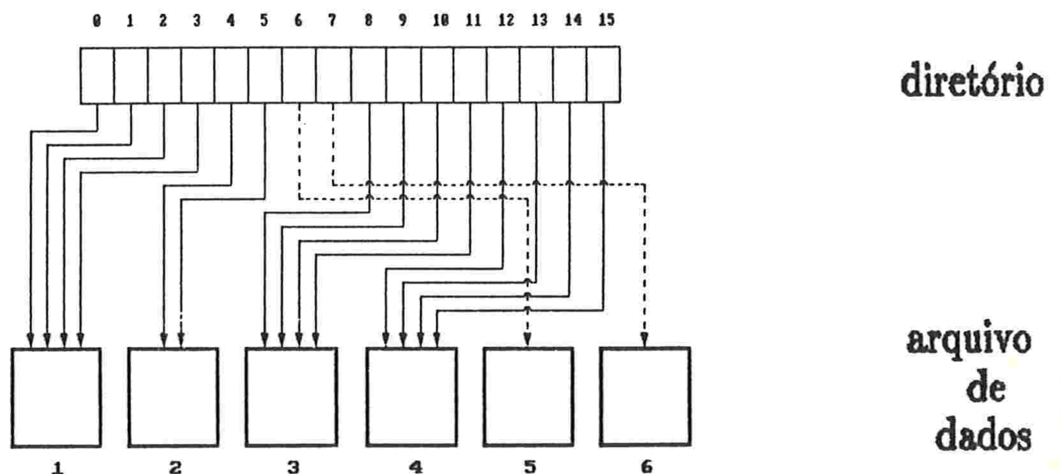

figura 4.1.3.2.2_ *expansão de um arquivo organizado pelo HE*

Observa-se que o número de apontadores para qualquer página é uma potência de dois.

Quando a página dividida é apontada por mais de uma entrada a expansão do arquivo é bastante simples. Metade dessas entradas continuam apontando para ela, e a outra metade aponta para a nova página. Entre as entradas que apontavam para a página dividida, aquelas que possuem índices maiores apontam para a nova página. A figura abaixo ilustra a expansão do arquivo após a divisão da página 4. O arquivo considerado é aquele ilustrado na figura 4.1.3.2.2.

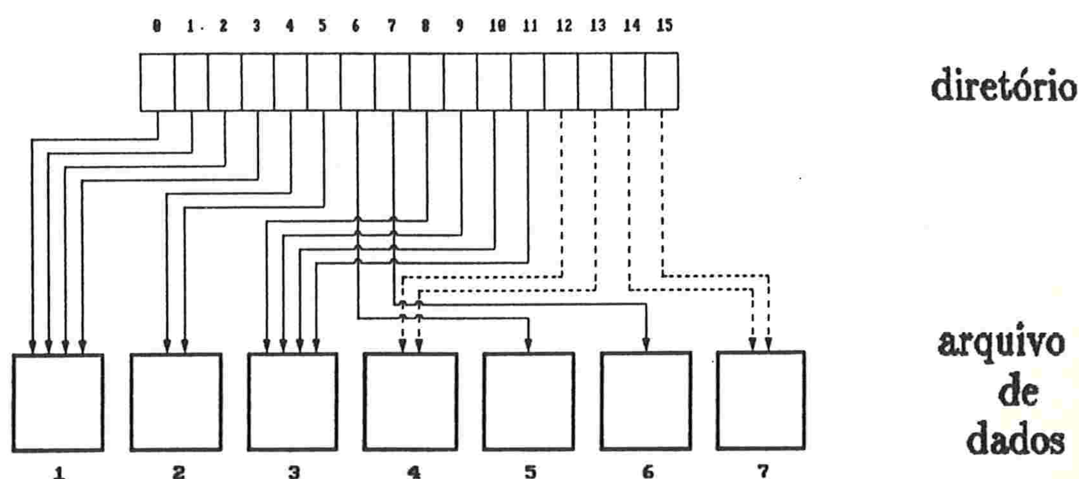

figura 4.1.3.2.3_ *expansão de um arquivo organizado pelo HE no caso ii*

Para saber qual a configuração do diretório no momento da expansão, é preciso varrê-lo localmente. Estamos no primeiro caso quando as entradas anterior e posterior àquela que aponta para a página dividida não apontam para essa página. Se o diretório encontra-se armazenado em disco, a consulta a essas entradas pode custar mais 1 acesso ao disco. Para evitar isto, é colocado um "header" em cada página do arquivo de dados, contendo o número de entradas do diretório que apontam para ela.

Algoritmo 4.1.3.2_ executa a expansão de um arquivo que organizado pelo Hashing extensível.

entrada :

P : número da página a ser dividida

l : entrada do diretório que aponta para a página dividida

Q : quantidade de páginas alocadas no arquivo de dados

comentários : o algoritmo abaixo supõe que o diretório está residente em 'memória' interna. O algoritmo de endereçamento encontra-se na seção

4.1.3.3.

```
{ alocar página Q+1 no arquivo de dados;
  ler página P;
  se a página P é apontada por apenas uma entrada
    então {
      dobrar o tamanho do diretório;
      entrada de índice 2l aponta para página P;
      entrada de índice 2l + 1 aponta para página Q+1;
      refazer os outros apontadores do diretório; }
    senão
      segunda metade dos apontadores que apontam para a página P
        passam a apontar para a página Q+1;
  para cada registro r endereçado à P faça {
    calcular endereço do registro r;
    se o endereço do registro r é P
      então escrever registro na página P
      senão escrever registro na página Q+1; }
  escrever página P no arquivo de dados;
  escrever página Q+1 no arquivo de dados;
  incrementar Q; }
```

A contração do arquivo é feita de forma inversa à expansão. Se duas páginas, que são apontadas por entradas vizinhas, possuem poucos registros, deve-se agrupar os seus registros e liberar uma das páginas. Todas as entradas apontarão para a página que permanece no arquivo.

4.1.3.3_ Função de endereçamento

Definimos nível do diretório como sendo o inteiro d tal que o diretório possui 2^d entradas. O diretório do arquivo ilustrado na figura 4.1.3.2.3 possui nível 4.

Através de uma função de hashing, que denotamos por f, os registros são distribuídos entre as entradas do diretório. A função f distribui o conjunto das chaves no intervalo dos naturais menores que 2^n, onde $n-1$ é o maior nível que o diretório pode assumir, isto é, o valor $f(k)$ é um natural com n dígitos binários. Seja d o nível em que o diretório se encontra, os

d 'bits' mais à esquerda do valor $f(k)$, ou seja os d dígitos binários mais significativos de $f(k)$, resultam em um número menor que 2^d, para qualquer chave k. Portanto, o número formado pelos d 'bits' mais à esquerda do valor $f(k)$, é um número de uma entrada existente no diretório. Assim, o endereço do registro é definido como sendo a página apontada pela entrada do diretório, que possui número igual ao valor dos d dígitos binários mais à esquerda de $f(k)$, ou seja a página apontada pela entrada de número $f(k)$ div 2^{n-d}. Considere o arquivo ilustrado na figura 4.1.3.2.1. Ele possui nível do diretório igual 3. Portanto, para endereçar os registros às páginas desse arquivo, são necessários os 3 'bits' mais à esquerda de $f(k)$. Assim, todos os registros de chave k, tal que $f(k) < 2^{n-3}$ são endereçados à página 1, pois os 3 dígitos binários mais à esquerda de $f(k)$ resultam no número 0, e a entrada 0 aponta para a página 1. Se $2^{n-3} \leq f(k) < 2^{n-2}$ os registros são endereçados também para a página 1, pois os 3 'bits' mais à esquerda de $f(k)$ formam o número 1, e a entrada 1 aponta para a página 1. Quando $2^{n-2} \leq f(k) < 2^{n-2} + 2^{n-3}$, os registros são endereçados para a página 2, pois os 3 'bits' formam o número 2. Para $2^{n-2} + 2^{n-3} \leq f(k) < 2^{n-1}$, o endereço do registro é 5, pois os 3 'bits' resultam no número 3, e assim segue. Na tabela abaixo é ilustrada a função de endereçamento para o arquivo da figura 4.1.3.2.3. Na primeira entrada da tabela estão os números binários formados por 4 dígitos, e na segunda entrada é dada a página do arquivo de dados, que é apontada pela entrada do diretório de índice indicado na primeira coluna da tabela (ver figura 4.1.3.2.3).

número de 4 dígitos binários	página do arquivo de dados
0000	1
0001	1
0010	1
0011	1
0100	2
0101	2
0110	5
0111	6
1000	3
1001	3
1010	3
1011	3

1100	4
1101	4
1110	7
1111	7

Algoritmo 4.1.3.3_ calcula o endereço de um registro em um arquivo organizado pelo Hashing extensível.

entrada :

k : chave do registro
d : nível do diretório

saída :

endereço do registro

comentários : o algoritmo dado abaixo assume que o diretório está residente em 'memória' interna.

{ L← d dígitos binários mais significativos de $f(k)$;
retorna (conteúdo da entrada de índice L do diretório); }

4.1.3.4_ Operações : consulta, inserção e eliminação

Após sucessívas eliminações de registros de um arquivo organizado pelo Hashing extensível, pode ocorrer a necessidade de contraí-lo. Para isto, precisa-se saber a quantidade de registros armazenados nas páginas envolvidas na contração. No entanto, o custo da eliminação é alto, se for necessário ler as páginas para decidir se o arquivo deve ou não ser contraído. Para evitar a leitura das páginas, pode-se armazenar em cada entrada do diretório, a quantidade de registros contidos na página apontada por essa entrada.

Ao armazenar um registro no arquivo, calcula-se o seu endereço. Se ocorrer colisão, o arquivo é expandido. Mas se todos os registros endereçados à página dividida forem enviados para a mesma página, ocorrerá outra expansão do arquivo, e este processo pode continuar infinitamente. Uma forma de interrompê-lo é limitar o número de expansões contíguas. Quando o limite é atingido a inserção não é executada. Uma outra maneira, é mudar o critério para expandir o arquivo. Usa-se a divisão controlada,

o tratamento da colisão pode ser feito pelo encadeamento das páginas de colisão ou pelo "open addressing".

4.1.3.5_ Modificações sugeridas para o Hashing extensível

Em [57] Tamminen sugere mudanças no tratamento das colisões e no critério para a expansão do arquivo. O arquivo é expandido quando ocorrer colisão e o seu fator de ocupação, neste momento, atingir o limite pré-determinado. Quando ocorrer a colisão, e o fator de ocupação do arquivo ainda está baixo, faz-se o tratamento pelo encadeamento das páginas de colisão.

Em [42] está uma generalização do Hashing extensível. Nele a chave do registro é composta por um vetor d-dimensional $k = (k_1, k_2, \ldots, k_d)$. O diretório é extendido para um 'array' d-dimensional, onde cada entrada aponta para uma página do arquivo. Somente a função de endereçamento e a forma de crescimento do diretório é que diferem este método do HE. A função de hashing, neste método, distribui os registros pelas entradas existentes no 'array'. Se durante a expansão do arquivo não há entrada no 'array' para apontar para a nova página, dobra-se o seu tamanho.

O Hashing exponencial com índice limitado (HEL) [34] também está baseado no HE. No HEL, o crescimento do arquivo e diretório se dá em 2 fases:
Na primeira, o processo é exatamente o mesmo do HE, até que o diretório atinja um tamanho pré-determinado. Naturalmente, esse tamanho deve ser dimensionado com vistas a manter o diretório em 'memória' interna.
Na segunda fase aparecem as diferenças com o HE. O diretório não cresce mais, e ao ocorrer colisão a página cheia e a nova página passam a formar um nó multipaginado. Este nó é apontado pela entrada do diretório que apontava para a página cheia. Quando uma dessas duas páginas estiver cheia, duplica-se a quantidade de páginas do nó. Na verdade, o que ocorre nesta segunda fase é o crescimento por duplicação de nós multipaginados e não de uma única página. Na segunda fase, a função de endereçamento distribui os registros pelas entradas do diretório, isto é pelos nós multipaginados, e aqueles endereçados ao mesmo nó, são distribuídos uniformemente pelas páginas que o formam. A distribuição dos registros pelas entradas do diretório continua sendo feita da mesma maneira, ou seja toma-se os d

bits mais à esquerda de $f(k)$, onde f é a função de hashing, e k a chave do registro, e d o nível do diretório. Para distribuir os registros pelas páginas do nó, toma-se os próximos l bits de $f(k)$, onde k é a chave do registro, f é a função de hashing, e o nó possui 2^l páginas. Cada entrada do diretório tem um campo a mais informando o tamanho do nó que ele está apontando. Com essa informação, é possível saber quantos bits a mais de $f(k)$ são necessários para localizar a página do nó, que contém o registro.

4.2_ HASHING INDEXADO NÃO LINEAR

Hashing indexado não linear é a técnica de organizar arquivos onde, a função de endereçamento utiliza uma tabela que possui tamanho proporcional ao número de páginas contidas no arquivo, e muitas páginas são alocadas na sua expansão. No momento em que essa tabela torna-se muito grande, ela é armazenada em disco. No entanto, se cada uma das suas entradas possui 1 bit de comprimento, isto faz com que sua transferência da 'memória' para o disco seja adiada, pois pode permanecer na 'memória' interna uma tabela com muito mais entradas. O Hashing virtual VH1 [24] é um exemplo em que isto ocorre. Como a tabela correspondente aos métodos VH0, HD e HE contém cada entrada com comprimento maior que 1 'bit', e as tabelas de todos os métodos possuem tamanho proporcional ao tamanho do arquivo, tem-se que, arquivos maiores têm sua tabela em 'memória' interna quando organizados pelo método VH1 do que quando organizados pelos outros métodos. Com isso, o número médio de acessos necessários para procurar um registro é menor no método VH1 do que no VH0, no HD ou no HE. Com relação à ocupação do espaço alocado para o arquivo, o VH1 ocupa, em média, mais área do que os outros métodos. No VH1 há grande oscilação no fator de ocupação do arquivo. Isto acontece porque o arquivo cresce duplicando o espaço alocado. Com isso, o fator de ocupação do arquivo logo após a expansão, é bem menor do que aquele anterior a ela.

4.2.1_ HASHING VIRTUAL VH1 [24]

4.2.1.1_ Descrição geral do método

O Hashing virtual VH1 é muito semelhante ao Hashing virtual VH0. Como no VH0, o VH1 também cria o arquivo de dados e o de colisões para armazenar um conjunto de informações. A tabela dos apontadores criada no VH0 é aqui substituída por uma tabela de bits. Tabela de bits é aquela em que cada entrada possui um bit de comprimento. A função de endereçamento utiliza-a para calcular o endereço de um registro.

O arquivo de dados é criado com m páginas. Elas constituem o espaço inicial de endereçamento do arquivo. Os registros são distribuídos uniformemente entre elas.

O critério usado no VH1 para expandir um arquivo, é o mesmo do VH0. Ele é expandido quando ocorre colisão e o seu fator de ocupação atinge o limite pré-determinado α. No VH0, de tempos em tempos duplica-se o tamanho da tabela dos apontadores, no VH1 isso ocorre com a tabela de bits e o arquivo de dados. Se durante a expansão do arquivo, a página a receber registros da página dividida não está alocada no arquivo, dobra-se a quantidade de páginas, e duplica-se o tamanho da tabela de bits. Na expansão, a página em que ocorreu a colisão é dividida. Uma página do arquivo de dados passa a pertencer ao espaço de endereçamento do arquivo. Os registros endereçados à página cheia são distribuídos entre ela e a nova integrante do espaço de endereçamento do arquivo. A tabela de bits é criada na primeira vez que ocorre a expansão do arquivo. Ela indica as páginas do arquivo de dados que pertencem ao espaço de endereçamento do arquivo. Ou seja, ela informa à função de endereçamento quais são as páginas do arquivo de dados que podem receber registros. As m páginas iniciais do arquivo de dados sempre petencerão ao espaço de endereçamento por isso, não há necessidade de associar entradas da tabela de bits a elas. A entrada de índice i está posicionada (contém 1), se a página $m+i$, do arquivo de dados, pertence ao espaço de endereçamento do arquivo.

O arquivo de colisões contém os registros de colisão das páginas pertencentes ao espaço de endereçamento do arquivo. Esses registros são aqueles endereçados a uma página qualquer do arquivo de dados, quando ela

está cheia. O tratamento da colisão é feito pelo encadeamento das páginas de colisões, e no primeiro tratamento cria-se este arquivo.

Assim como no VH0, este método utiliza um conjunto de funções de hashing para endereçar um registro. A função de endereçamento do VH1 é extremamente semelhante a do VH0.

4.2.1.2_ Processo de expansão

O arquivo é expandido quando ocorrer colisão e o seu fator de ocupação atingir o limite α, no momento da colisão.

Seja p o número da página dividida, e $1 \leq p \leq m$. Na sua primeira expansão, os registros endereçados à p são distribuídos entre as páginas p e $m+p$, e essa página é incorporada no espaço de endereçamento do arquivo. Na segunda divisão de p, a página $2m+p$ passa a pertencer ao espaço de endereçamento do arquivo, e alguns registros de p são endereçados a ela. Na terceira divisão de p, isso ocorre com a página $4m+p$, e assim segue. Por exemplo, se $m = 5$ e $p = 3$, a página $m+p = 8$ é incorporada no espaço de endereçamento do arquivo, na primeira divisão da página 3. Na sua segunda divisão, isto ocorre com a página $2m+p = 13$, depois com a página $4m+p = 23$. Se $m+1 \leq p \leq 2m$, na primeira divisão de p, a distribuição dos seus registros é feita entre as páginas p e $2m+p$. Na sua segunda divisão, isto ocorre entre as páginas p e $4m+p$, e assim segue. De forma geral, se $2^{i-1}+1 \leq p \leq 2^i$, na j-ésima divisão da página p, a página $2^{i+j-1}m+p$ é incorporada no espaço de endereçamento do arquivo, e alguns registros de p são endereçados a ela.

Os passos executados na expansão do arquivo são dados a seguir. Seja p o número da página dividida, onde $2^n+1 \leq p \leq 2^{n+1}$ para algum $n \in N$, e suponha que essa é a j-ésima divisão da página p. A distribuição dos registros de p é feita entre as páginas p e $2^{n+j}+p$. Pode-se verificar uma das duas condições no arquivo de dados:

i- a página $2^{n+j}m+p$ não está alocada no arquivo de dados;

ii- a página $2^{n+j}m+p$ está alocada no arquivo de dados.

Quando ocorre o primeiro caso, duplica-se a quantidade de páginas alocadas no arquivo de dados, e aloca-se igual número da entradas na tabela de bits. A página $2^{n+j}m+p$ é incorporada ao espaço de endereçamento do arquivo, ou seja, a função de endereçamento associa registros a ela. Para

isso, a entrada $2^{n+j}m + p - m$ é posicionada. As outras novas páginas apesar de estarem alocadas no arquivo de dados, não são imediatamente incorporadas no espaço de endereçamento, por isso a entrada na tabela de bits correspondente a cada uma dessas páginas não é posicionada, isto é possui conteúdo 0. Para ilustrar o caso i, considere um arquivo criado com $m = 5$ páginas. Na primeira expansão desse arquivo suponha que a página 3 é dividida. Como a página 8 deve ser incorporada no espaço de endereçamento, e ela não está alocada no arquivo de dados, duplica-se a quantidade de páginas, ou seja alocam-se as páginas 6, 7, 8, 9 e 10. Cria-se a tabela de bits com 5 entradas, e posiciona-se apenas a entrada 3, para indicar que entre as novas páginas, apenas a 8 pertence ao espaço de endereçamento. Na figura abaixo, representa-se este arquivo após a primeira expansão.

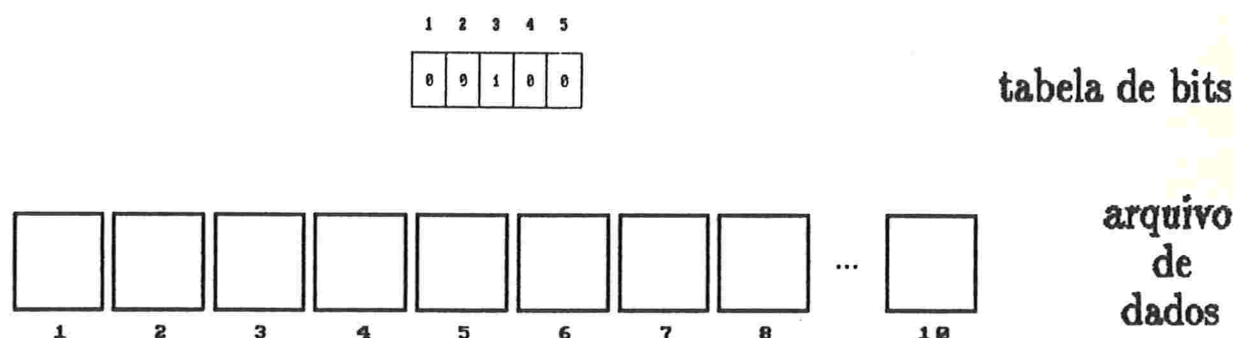

figura 4.2.1.2.1_ *arquivo organizado pelo VH1 após a primeira expansão*

Se ocorrer o caso ii, a expansão do arquivo é simples. Como a página $2^{n+j}m+p$ já está alocada no arquivo de dados, basta incorporá-la ao espaço de endereçamento do arquivo, e executar a distribuição dos registros da página p. Para que a página $2^{n+j}m + p$ passe a pertencer ao espaço de endereçamento do arquivo, é necessário posicionar a entrada da tabela de bits correspondente a ela, ou seja posicionar a entrada $2^{n+j}m+p-m$. Se a página 5 do arquivo considerado na figura 4.2.1.2.1 é dividida, ocorre o caso ii. A página 10 é incorporada no espaço de endereçamento do arquivo, ou seja a entrada 5 da tabela de bits é posicionada. A figura abaixo ilustra o arquivo após a divisão da página 5. Note que a tabela de bits possui apenas as entradas 3 e 5 posicionadas, indicando que apenas as páginas 1, 2, 3, 4, 5, 8 e 10 do arquivo de dados, pertencem ao espaço de endereçamento do arquivo.

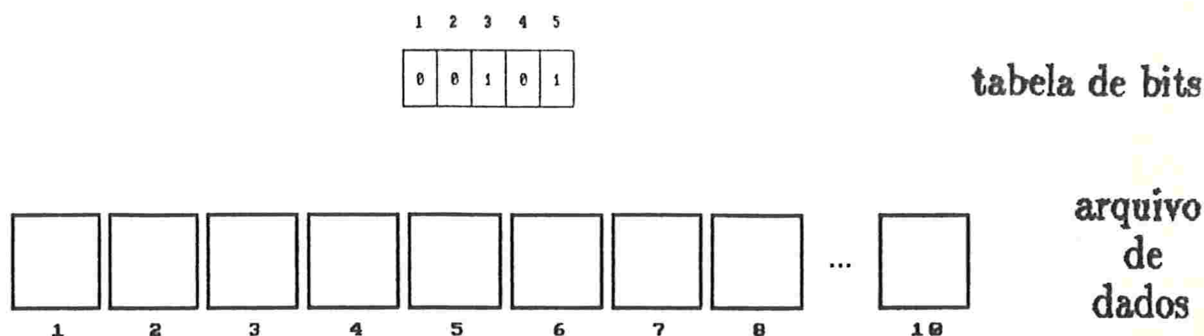

figura 4.2.1.2.2_ *arquivo organizado pelo VH1 após duas expansões*

Suponha que, na próxima expansão do arquivo ilustrado na figura 4.2.1.2.2, a página 3 é dividida novamente. A página $2^1m + 3 = 13$ é incorporada no espaço de endereçamento do arquivo. Como o arquivo de dados não possui esta página, estamos no caso i. Portanto deve-se alocar as páginas $11, 12, \ldots, 20$, e na tabela de bits adiciona-se 10 novas entradas, ou seja ela passa a possuir tamanho 15. A entrada 8 é posicionda, e nas outras novas entradas coloca-se o valor 0. A figura abaixo ilustra o arquivo após essa divisão. Note que, apesar do arquivo de dados possuir 20 páginas, a função de endereçamento associa registros apenas às páginas 1, 2, 3, 4, 5, 8, 10 e 13, pois somente elas pertencem ao espaço de endereçamento do arquivo.

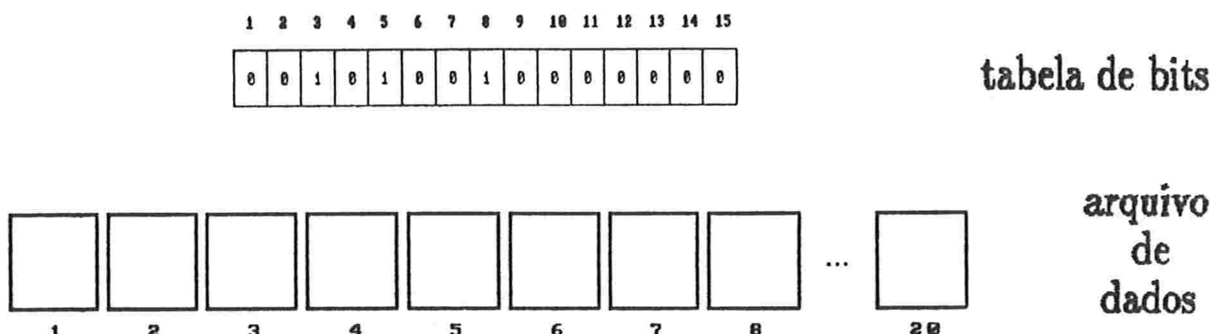

figura 4.2.1.2.3_ *expansão de um arquivo organizado pelo VH1 no caso ii*

Na figura acima notamos desperdício de espaço alocado no arquivo de dados. Isto ocorre por que são endereçados mais registros à página 3 do que às outras páginas. Para evitar a sub-utilização do espaço alocado, exige-se que a função de endereçamento distribua uniformemente os registros pelas páginas do espaço de endereçamento do arquivo.

Para calcular o número da página que é incorporada no espaço de endereçamento do arquivo durante a divisão, é necessário saber por quantas

divisões já passou a página que está sendo dividida. O armazenamento dessa informação nas páginas do arquivo de dados, é evitado da seguinte forma: Seja $n \in N$ tal que o arquivo de dados possui 2^n páginas alocadas, e a página p está sendo dividida durante a expansão. Se $p > 2^{n-1}m$ ou a página $2^{n-1}m + p$ já pertence ao espaço de endereçamento do arquivo, tem-se que a página a ser incorporada nessa expansão possui número $2^n m + p$. Caso contrário, ou seja $p \leq 2^{n-1}$ e a página $2^{n-1}m + p$ não pertence ao espaço de endereçamento do arquivo, verifica-se se a página $2^{n-2}m + p$ está incorporada nele. Se sim, a página $2^{n-1}m + p$ passa a fazer parte do espaço de endereçamento do arquivo. Se não, verifica-se a página $2^{n-3}m + p$, e assim segue a procura de uma página que faça parte do espaço de endereçamento do arquivo. Ao encontrá-la, toma-se a última página, entre aquelas que foram verificadas, para fazer parte do arquivo nessa expansão. Lembre-se que uma página p pertence ao espaço de endereçamento do arquivo, se a entrada $p - m$ da tabela de bits está posicionada.

Algoritmo 4.2.1.2_ executa a expansão de um arquivo organizado pelo Hashing virtual VH1

entrada :

P : número da página a ser dividida

j : inteiro tal que, o arquivo de dados possui $2^j m$ páginas alocadas

comentários : O algoritmo abaixo assume que a tabela de bits encontra-se em 'memória' interna. O algoritmo que calcula o endereço de um registro é dado na seção 4.2.1.3 .

{ ler a página P;
  ler a região de colisão de P;
  se esta é a primeira expansão do arquivo (j= 0)
    então {
      criar a tabela de bits com m entradas;
      posicionar a entrada de índice P;
      alocar m páginas no fim do arquivo de dados;
      incrementar j; }
    senão
      se P $\leq 2^{j-1}m$ *e* a entrada $2^{j-1}m + P - m$
        não está posicionada

```
        então {
          u ← j − 1;
          enquanto entrada de índice 2^u m + P − m não está
            posicionada faça
              decrementar u;
          posicionar a entrada 2^(u+1) m + P − m; }
        senão {
          alocar 2^j m novas entradas na tabela de bits;
          posicionar a entrada 2^j m + P − m;
          alocar 2^j m novas páginas no arquivo de dados;
          incrementar j; }
    para cada registro r endereçado à página P faça {
      calcular o endereço de r considerando a nova tabela de bits;
      se endereço do registro r é P
        então
          escrever registro na página P ou na sua região de colisão
        senão
          escrever registro na nova página ou na sua região de colisão; }
    escrever a página P no arquivo de dados;
    escrever a região de colisão da página P no arquivo de colisões;
    escrever a nova página no arquivo de dados;
    escrever a região de colisão da nova página no arquivo de colisões; }
```

4.2.1.3_ Função de endereçamento

Assim como no Hashing virtual VH0, o Hashing virtual VH1 usa um conjunto de funções de hashing a qual denotaremos por $f_0, f_1, f_2, \ldots$. A função f_i, $i \geq 0$, mapeia o conjunto das chaves no conjunto $\{1, 2, \ldots, 2^i m\}$; e para $i \geq 1$ e k uma chave qualquer, tem-se que:

$$f_i(k) = f_{i-1}(k)$$

ou

$$f_i(k) = f_{i-1}(k) + 2^{i-1} m$$

Enquanto o arquivo possui m páginas, o algoritmo de endereçamento utiliza a função de hashing f_0. A página $f_0(k)$ armazena o registro

de chave k. Quando ocorrer a expansão do arquivo são alocadas m novas páginas. A função f_0 não cobre mais todo o arquivo, pois ela associa as chaves às páginas de número menores ou iguais a m. O algoritmo de endereçamento passa a utilizar as funções f_0 e f_1, para distribuir os registros pelas páginas pertencentes ao seu espaço de endereçamento. A função f_1 endereça registros às páginas que sofreram uma divisão, ou aquelas que foram alocadas nessas divisões, isto é a página $f_1(k)$ contém o registro de chave k, se ela sofreu uma divisão, ou foi alocada durante a primeira divisão da página $f_0(k)$. Caso contrário, o endereço do registro é a página $f_0(k)$. Tem-se que $f_1(k) = f_0(k)$ ou $f_1(k) = f_0(k) + m$, pois f_1 é usada durante a expansão do arquivo, para distinguir os registros que permanecem na página dividida daqueles que são transferidos para a nova página. Lembre-se que, na primeira divisão de uma página p, onde $p \leq m$, a nova página possui número $m + p = m + f_0(k)$, onde k é a chave de um registro armazenado em p. Para exemplificar a função de endereçamento, considere o arquivo da figura 4.2.1.2.2 . A função f_1 endereça registros às páginas 3, 5, 8 e 10, pois as páginas 3 e 5 foram divididas uma vez, e as páginas 8 e 10 foram incorporadas ao espaço de endereçamento do arquivo nessas divisões. Para as páginas 1, 2 e 4 utiliza-se a função f_0. Após outro crescimento do arquivo de dados, toma-se mais uma função da hashing, a função f_2. A página $f_2(k)$ é o endereço do registro, se ela pertence ao espaço de endereçamento do arquivo, ou seja se a entrada $f_2(k) - m$ da tabela de bits está posicionada. Caso contrário, o endereço do registro é a página $f_1(k)$, se esta página pertence ao espaço de endereçamento do arquivo. Se não, a página $f_0(k)$ contém o registro. Lembre-se que, a página $f_2(k)$ pertence ao espaço de endereçamento do arquivo se:

- i- a página $f_1(k)$ foi dividida duas vezes, neste caso tem-se que $f_0(k) = f_1(k)$;
- ii- a página $f_1(k)$ foi dividida uma vez e $f_0(k) = f_1(k) + m$;
- iii- a página $f_2(k)$ foi incorporada ao espaço endereçamento do arquivo na divisão da página $f_1(k)$ e $f_2(k) > 2m$.

De uma maneira geral, define-se o endereço do registro de chave k como sendo a página $f_l(k)$ do arquivo de dados, onde l é o maior inteiro menor ou igual a j tal que a página $f_l(k)$ pertence ao espaço de endereçamento do arquivo, e o arquivo de dados possui $2^j m$ páginas. Lembre-se que, uma página p do arquivo de dados pertence ao seu espaço de endereçamento, se a entrada de índice $p - m$ da tabela de bits está posicionada

ou $p \leq m$.

A figura abaixo ilustra o algoritmo de endereçamento do Hashing Virtual VH1. A parte **A** da figura representa o arquivo antes da expansão, e a **B** após a divisão da página 3. Ele foi criado com $m = 5$ páginas, e a inserção do registro de chave 147 provocou a sua expansão. Os números contidos nas páginas, são as chaves dos registros nela armazenados. Cada página possui capacidade igual a 3. As funções de hashing utilizadas são definidas por $f_i(k) = (k \bmod (2^i m)) + 1$. Na figura está indicada, para cada página do espaço de endereçamento, a função de hashing que o algoritmo de endereçamento utiliza para associar registros a ela.

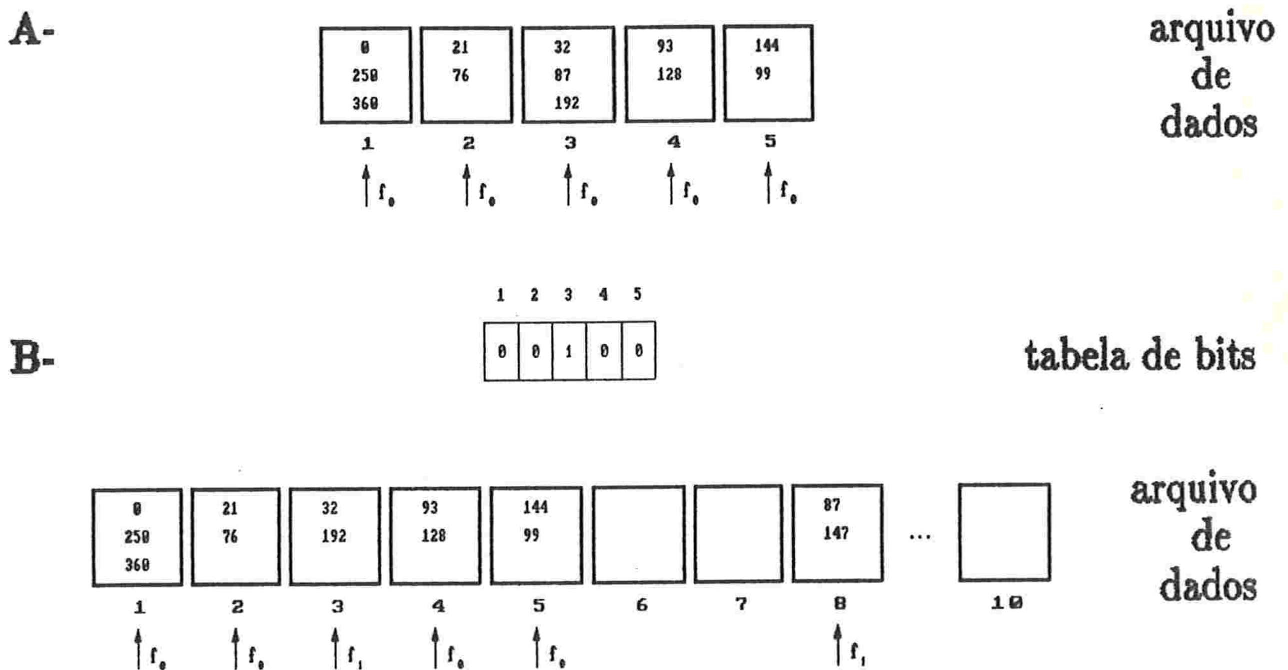

figura 4.2.1.3.1_ *esquema geral do algoritmo de endereçamento do VH1*

Algoritmo 4.2.1.3_ calcula o endereço de um registro em um arquivo organizado pelo Hashing virtual VH1.

entrada :

k : chave do registro

j : inteiro tal que, o arquivo de dados possui $2^j m$ páginas alocadas

saída :

endereço do registro

comentários: O algoritmo de endereçamento assume que a tabela de bits encontra-se armazenada em 'memória' interna.

{ $l \leftarrow f_j(k)$;
  $u \leftarrow j$;
  enquanto $l > m$ *e* entrada $l - m$ da tabela de bits *não* está
    posicionada *faça* {
      decrementar u;
      $l \leftarrow f_u(k)$; }
  retorna (l); }

4.3_ HASHING LINEAR NÃO INDEXADO

Hashing linear não indexado é a técnica de Hashing de organizar arquivos, onde uma única página é alocada durante a sua expansão, e a função de endereçamento não precisa de uma tabela, com tamanho proporcional ao tamanho do arquivo, para calcular o endereço do registro. Para qualquer arquivo, a quantidade de espaço necessário para calcular o endereço do registro é praticamente constante. Com isso obtem-se melhor desempenho na execução das operações, pois para calcular o endereço do registro não é necessário ir ao disco buscar informações mesmo quando o arquivo é grande. O preço pago pela eliminação dessa estrutura indireta é fixar uma ordem nas páginas a serem divididas. Assim, as páginas do arquivo podem possuir registros de colisão, pois na expansão, a página a ser dividida não é necessariamente aquela que está mais cheia, é aquela definida pela ordem fixada pelo método. Portanto, todos os métodos de hashing linear não indexado fazem o tratamento da colisão armazenando os registros de colisão em algum lugar, e definem a ordem das páginas divididas. Esses dois fatores influem no desempenho do método. A ordem das páginas divididas afeta a distribuição dos registros pelo arquivo, pois espera-se que uma página dividida contenha menos registros endereçados a ela do que aquelas que ainda não foram divididas. Com isso, as colisões ficam concentradas nas páginas que não foram divididas, alterando assim, em média, o desempenho do método.

Nesta seção estão descritos quatro métodos de hashing linear não indexado. O Hashing virtual linear (HL) [31], Hashing linear com expansão parcial (HLP) [19], o Hashing linear com expansão em grupo (HLG) [46] e o Hashing com armazenamento em espiral (HAE) [38].

O Hashing virtual linear foi o primeiro método proposto de hashing linear não indexado. Nele já se verifica um ganho em desempenho comparando com os métodos descritos nas seções anteriores. Em média, a procura com sucesso faz 1.27 acessos ao disco quando o fator de ocupação do arquivo está em torno de 85% [19].

O Hashing linear com expansão parcial é uma generalização do HL. O HLP faz a distribuição dos registros pelo arquivo de forma mais uniforme do que aquela feita pelo HL, obtendo com isso, desempenho mais uniforme. Em média, uma procura com sucesso faz 1.12 acessos ao disco quando o fator de ocupação do arquivo está em torno de 85% [19].

O Hashing linear com expansão em grupo (HLG) é também uma generalização do HL, porém mais simples do que o HLP. O HLP e o HLG possuem desempenho semelhante.

O Hashing com armazenamento em espiral é completamente diferente de todos os outros métodos. A principal diferença reside na forma que a função de endereçamento distribui os registros pelo arquivo. Em média, uma procura com sucesso faz 1.24 acessos ao disco quando o fator de ocupação do arquivo é limitado a 85% [24].

4.3.1_ HASHING VIRTUAL LINEAR (HL) [31]

4.3.1.1_ Descrição geral do método

Dois arquivos são criados quando um conjunto de informações é armazenado no disco, usando o Hashing virtual linear (HL). Eles são chamados de arquivo de dados e arquivo de colisões.

O arquivo de dados é criado com m páginas. Os registros são distribuídos uniformemente por essas páginas. Assim como no VH1, a função de endereçamento deste método usa um conjunto de funções de hashing, denotadas por $f_0, f_1, \ldots$. Para qualquer estágio que o arquivo se encontre, a função de endereçamento usa no máximo duas dessas funções. A medida que o arquivo vai crescendo as funções vão sendo trocadas dinamicamente.

Enquanto o arquivo de dados possui m páginas, somente a função f_0 é usada. O arquivo de colisões contém os registros de colisão das páginas do arquivo de dados. Este arquivo é criado durante o tratamento da primeira colisão, a qual é tratada pelo encadeamento das páginas de colisão.

O critério usado para expandir o arquivo é a divisão não controlada, ou seja, o arquivo é expandido ao ocorrer uma colisão. Na primeira expansão, aloca-se a página $m+1$ no arquivo de dados. Os registros endereçados à página 1 são distribuídos uniformemente entre ela e a página $m+1$. Uma outra função de hashing f_1 é usada para endereçar registros às páginas 1 e $m+1$. A função de endereçamento continua usando a função f_0 para endereçar registros às outras páginas do arquivo de dados. Note que a página dividida não é necessariamente a página onde ocorreu colisão. Por isso, é necessário fazer o tratamento do registro de colisão após executar a expansão do arquivo. A figura abaixo ilustra a primeira expansão de um arquivo. Ele foi criado com $m = 5$ páginas. Cada página possui capacidade igual a 3. A função f_i, $i = 0, 1$, é definida por $f_i(k) = 1 + (k \bmod (2^i m))$, onde k é a chave do registro. Os números contidos nas páginas são as chaves dos registros nela armazenados. O registro de chave $k = 17$ será inserido no arquivo. A figura **A** representa o arquivo antes da inserção, e a figura **B** após a inserção. Em cada página do arquivo de dados está indicada a função de hashing que endereça registros a ela.

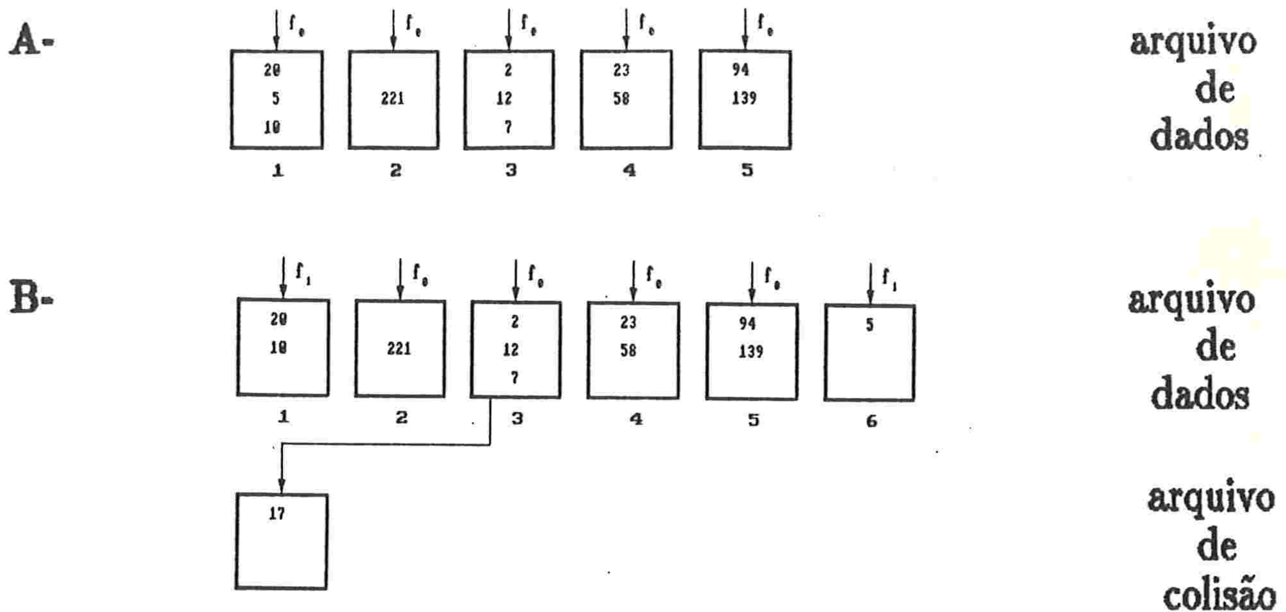

figura 4.3.1.1.1_ *primeira expansão de um arquivo organizado pelo HL*

Na segunda expansão do arquivo a página 2 é dividida e a página $m+2$ é alocada. A função f_1 passa a endereçar registros às páginas 2 e $m+2$. Assim, essa função endereça registros às páginas $1, 2, m+1$ e $m+2$, e a função f_0 distribui os registros entre as páginas restantes. E assim, o processo da expansão do arquivo continua até que a página m seja dividida. Após a divisão desta página, somente a função f_1 distribui os registros pelas $2m$ páginas do arquivo de dados. Na próxima expansão do arquivo, torna-se a dividir a página 1. A função de hashing f_2 é usada para distribuir, entre as páginas 1 e $2m+1$, os registros armazenados na página 1. E o crescimento do arquivo segue dessa forma. Note que é necessário no máximo duas funções em cada instante.

4.3.1.2_ Processo de expansão e contração

A expansão de um arquivo organizado pelo Hashing virtual linear, acontece quando ocorre uma colisão. Uma descrição geral do processo de expansão é dada a seguir.

Considere um arquivo com $2^j m$ páginas, onde $j \in N$. Quando o arquivo é criado, tem-se que $j = 0$. Nas $2^j m$ colisões seguintes, as páginas

$1, 2, \ldots, 2^j m$ são divididas nesta ordem, ou seja, a ordem das páginas divididas é linear. Os registros endereçados, pela função f_j, à página dividida, isto é os registros de chave k tal que $f_j(k)$ é igual o número da página dividida, são distribuídos entre ela e a página alocada durante a sua divisão. Essa distribuição é feita pela função de hashing f_{j+1}.

A página a ser dividida durante a expansão do arquivo é indicada por um apontador, que é denotado por AP. Na criação do arquivo, ele é inicializado com 1. O algoritmo que executa a expansão faz a sua atualização. Observe que AP é um parâmetro do arquivo, ou seja cada arquivo possui o seu apontador AP. Assim, a cada arquivo deve estar associado um campo adicional contendo esse valor.

Algoritmo 4.3.1.2_ executa a expansão de um arquivo organizado pelo Hashing virtual linear.

entrada :

AP : apontador para a página a ser dividida
Q : quantidade de páginas alocadas no arquivo de dados

comentários : o algoritmo que calcula o endereço de um registro é dado na seção 4.3.1.3.

{ alocar página Q+1 no arquivo de dados;
 ler página apontada por AP e a sua região de colisão;
 para cada registro lido *faça* {
  calcular endereço do registro considerando a nova página;
  se endereço do registro *não* mudou
   então
    escrever registro na página dividida ou na sua região de colisão
   senão
    escrever registro na página Q+1 ou na sua região de colisão; }
 escrever página apontada por AP no arquivo de dados;
 escrever a região de colisão da página dividida no arquivo de colisões;
 escrever página Q+1 no arquivo de dados;
 escrever a região de colisão da página Q+1 no arquivo de colisões;
 incrementar Q e AP;
 se AP $= \frac{Q}{2} + 1$
  então AP $\leftarrow$ 1; }

A contração do arquivo acontece quando, eliminando um registro do arquivo, pode-se armazenar na sua página original todos os seus registros de colisão, ou seja, a página original do registro que está sendo eliminado possuia registros de colisão, e após a sua retirada, não os tem mais. Na contração, a última página do arquivo de dados é liberada. Os registros armazenados nesta página são reinseridos no arquivo, e o apontador AP é decrementado.

4.3.1.3_ Função de endereçamento

Um arquivo organizado pelo Hashing virtual linear, possui *nível* d se d é o menor inteiro tal que, o número de páginas contidas no arquivo é menor ou igual $2^d m$. Por exemplo, quando o arquivo é criado ele possui nível 0.

A função de endereçamento do HL é semelhante à do VH1. Ela utiliza um conjunto de funções de hashing, denotadas por $f_0, f_1, \ldots$. A função f_j mapeia o conjunto das chaves no conjunto $\{1, \ldots, 2^j m\}$. Essas funções definem o endereço do registro. Quando o nível do arquivo é j, a função de endereçamento utiliza a função de hashing f_j para definir o endereço dos registros endereçados às páginas que não foram divididas neste nível. Aquelas que já se dividiram são endereçadas pela função f_{j+1}, ou seja, o endereço de um registro de chave k é $f_j(k)$ se $f_j(k) \geq AP$, pois esta página não foi dividida neste nível; caso contrário, é $f_{j+1}(k)$. A função f_{j+1} é usada para separar os registros endereçados à página dividida em dois conjuntos: aqueles que permanecem nela, e aqueles endereçados à nova página. Assim, a função f_{j+1} está relacionada com f_j da seguinte forma:

$$f_{j+1}(k) = f_j(k)$$

ou

$$f_{j+1}(k) = f_j(k) + 2^j m$$

para qualquer chave k.

A família de funções $f_j(k) = 1 + (k \bmod (2^j m))$, onde $j \geq 0$, é um exemplo de conjunto de funções que obedecem a restrição acima.

Algoritmo 4.3.1.3_ calcula o endereço de um registro em um arquivo organizado pelo Hashing virtual linear.

entrada :
k : chave do registro
d : nível do arquivo
AP : apontador para página a ser dividida
saída :
endereço do registro

{ *se* $f_d(k) \geq AP$
então retorna ($f_d(k)$)
senão retorna ($f_{d+1}(k)$); }

4.3.1.4_ Alterações no Hashing virtual linear

O próprio Litwin em [31] sugere modificar o critério para fazer a expansão de um arquivo. Se este for a divisão controlada, consegue-se melhor utilização do espaço alocado sem alterar o desempenho do método.

Mullin propõem em [40] armazenar os registros de colisão no arquivo de dados. A expansão do arquivo ocorrerá quando o seu fator de ocupação atingir um limite pré-determinado. Espera-se que, a última página dividida no arquivo seja aquela que contém a menor quantidade de registros armazenados. Assim, ao ocorrer colisão em uma página tenta-se armazenar o seu registro de colisão na última página dividida. Se ela estiver cheia, tenta-se a penúltima, e assim por diante. O arquivo contém espaço livre para armazenar o registro pois, o seu fator de ocupação está sendo controlado e é menor que 1. Cada página contém apontadores para as páginas do arquivo que armazenam os seus registros de colisão. Assim, a página é composta por dois conjuntos de informações. Uma parte da página armazena os registros, e a outra contém os apontadores para os seus registros de colisão.

O Hashing recursivo linear [49] é uma variação na forma de tratar a colisão do HL. Os registros de colisão, das páginas contidas no arquivo, são armazenados em um outro arquivo organizado pelo HL. Se ocorrer colisão neste segundo arquivo, cria-se um terceiro, também organizado pelo HL, para armazenar os registros de colisão das páginas contidas no segundo arquivo. Assim, os registros de colisão das páginas contidas no i-ésimo arquivo, estão armazenados no (i+1)-ésimo arquivo. Cada arquivo desses

é organizado pelo HL, ou seja, todos eles possuem a mesma função de endereçamento, e crescem e diminuem da mesma forma.

4.3.2_ HASHING LINEAR COM EXPANSÃO PARCIAL [19]

4.3.2.1_ Descrição geral do método

O Hashing linear com expansão parcial (HLP) é uma generalização do Hashing virtual linear (HL). O HL é um método que executa a expansão do arquivo de uma maneira simples e graciosa. No entanto, quando um arquivo é organizado pelo HL, verifica-se a não uniformidade da distribuição dos registros pelo arquivo. Isto porque as páginas divididas concentram-se no início do arquivo, e é esperado que após a divisão de uma página ela contenha menos registros do que qualquer outra página ainda não dividida. Assim, há aglomeração de registros contidos nas páginas do meio do arquivo. O gráfico abaixo ilustra grosseiramente a distribuição da armazenagem dos registros em um arquivo organizado pelo HL. As páginas $1, 2, \ldots, p$ já foram divididas. O eixo das abcissas representa as páginas do arquivo, e o da coordenadas a quantidade de registros armazenados na página.

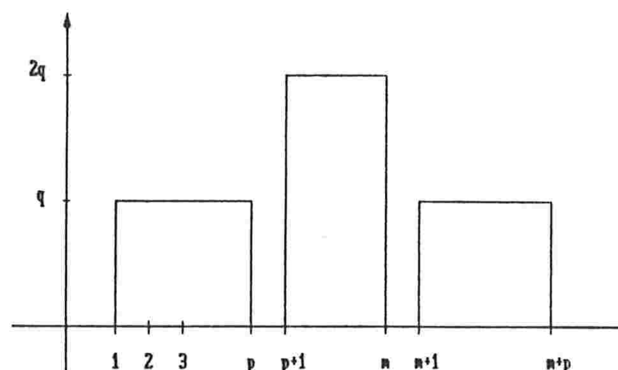

figura 4.3.2.1.1_ *distribuição dos registros num arquivo organizado pelo HL*

No HLP, a aglomeração dos registros é quebrada em blocos menores, ou seja, o conjunto das páginas não divididas fica composto por pequenos blocos de páginas consecutivas. Para fazer isto, as páginas do

arquivo são reunidas em grupos. Inicialmente, todos os grupos possuem a mesma quantidade de páginas, e as páginas de um grupo não são consecutivas. Na expansão, uma página é alocada no fim do arquivo e anexada a um grupo. Os registros endereçados às páginas contidas nesse grupo são uniformemente distribuídos entre elas e a nova página. Quando isso ocorre, diz-se que o grupo foi dividido. Após dividir todos os grupos do arquivo, diz-se que houve uma *expansão parcial*. Espera-se que, as páginas contidas nos grupos que foram divididos, contenham menos registros do que aquelas pertencentes aos grupos não divididos. Como as páginas de um grupo não são consecutivas, aquelas com grande probabilidade de estarem cheias estão espalhadas pelo arquivo.

Quando a quantidade de páginas contidas no arquivo dobrar diz-se que ocorreu uma *expansão total*. A expansão total ocorre após uma série de expansões parciais, ou seja, a quantidade de páginas do arquivo dobra após a quantidade de páginas de cada grupo dobrar. Se a expansão total ocorrer após 1 expansão parcial, o HLP é o HL. O critério, no HLP, para expandir o arquivo é a divisão controlada.

O tratamento das colisões é feito pelo encadeamento das páginas de colisão. O arquivo de colisão é criado durante o tratamento da primeira colisão, e os registros armazenados nele são aqueles endereçados à página cheia.

4.3.2.2_ Processo de expansão e contração

Para simplificar a exposição do processo de expansão de um arquivo organizado pelo Hashing linear com expansão parcial, vamos assumir que a expansão total ocorre após duas expansões parciais.

Inicialmente, o arquivo possui m páginas, onde $m = 2s$ e $s \in N$. Estas páginas estão logicamente reunidas em s grupos de 2 páginas cada. O grupo j, $1 \leq j \leq s$, é formado pelas páginas j e $j + s$. A figura abaixo ilustra um arquivo criado com $m = 6$ páginas, reunidas em 3 grupos de 2 páginas cada ($s = 3$). O grupo 1 é formado pelas páginas 1 e 4, o 2 pelas páginas 2 e 5 e as páginas 3 e 6 formam o último grupo. O grupo 1 está indicado na figura.

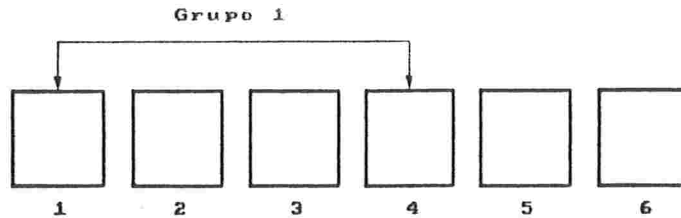

figura 4.3.2.2.1_ arquivo organizado pelo HLP e criado com 6 páginas

Os registros são uniformemente distribuídos entre as m páginas do arquivo. Quando o fator de ocupação do arquivo atingir um limite pré-determinado, o arquivo é expandido. O grupo 1 é dividido, e a página $m + 1 = 2s + 1$ é alocada e anexada a ele, ou seja, o grupo 1 passa a ser formado pelas páginas $1, 1 + s$ e $1 + 2s$. A figura abaixo ilustra o arquivo da figura anterior após a sua primeira expansão. O grupo 1 cresce de 2 para 3 páginas e está indicado na figura.

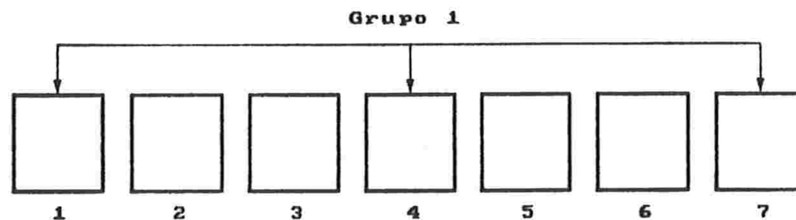

figura 4.3.2.2.2_ primeira expansão de um arquivo organizado pelo HLP

Nas próximas $s - 1$ expansões do arquivo os grupos $2, 3, \ldots, s$ são divididos, nessa ordem. Em cada expansão, um único grupo é dividido e passa a conter 3 páginas. A primeira expansão parcial acaba, após a divisão do último grupo, o grupo s. Inicia-se então, a segunda expansão parcial. Nesta expansão parcial, o arquivo está com $m + s = 3s$ páginas. Elas são logicamente reunidas em s grupos de 3 páginas cada. O grupo j, $1 \leq j \leq s$, é formado pelas páginas $j, j+s$ e $j+2s$. A figura abaixo ilustra o arquivo da figura 4.3.2.2.2, no início da segunda expansão parcial. O grupo 1 continua sendo indicado na figura.

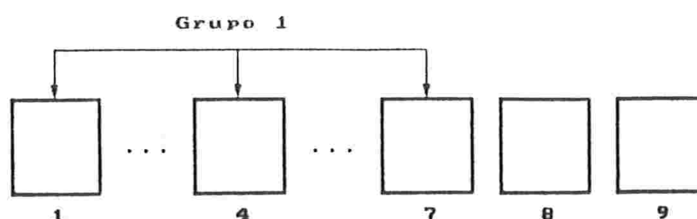

figura 4.3.2.2.3_ *início da segunda expansão parcial de um arquivo organizado pelo HLP*

O crescimento do arquivo, na segunda expansão parcial, é igual ao da primeira expansão parcial, diferem somente pela quantidade de páginas por grupo. Na segunda expansão parcial, os grupos crescem de 3 para 4 páginas. Em cada expansão do arquivo um único grupo é dividido. A ordem dos grupos divididos é $1, 2, \ldots, s$. Após a divisão de todos os grupos, a segunda expansão parcial acaba e o arquivo está com $4s = 2m$ páginas. Duplicou-se a quantidade de páginas contidas no arquivo, ou seja, ocorreu uma expansão total. Isto era esperado, pois estamos supondo que a expansão total ocorre após duas expansões parciais. A figura abaixo ilustra o arquivo da figura 4.3.2.2.3, após a segunda expansão parcial. O grupo 1 está indicado na figura.

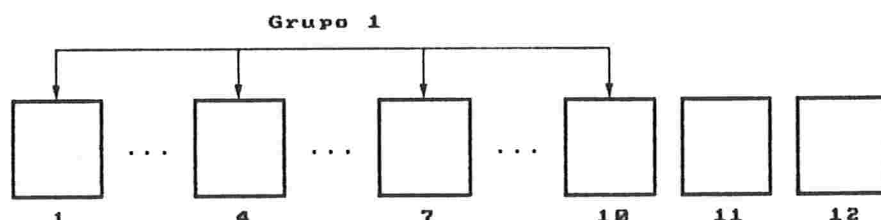

figura 4.3.2.2.4_ *fim da segunda expansão parcial de um arquivo organizado pelo HLP*

Ao ocorrer a expansão total, retorna-se a situação inicial. As páginas do arquivo são reunidas em $2s$ grupos de 2 páginas cada. Esses grupos crescem de 2 para 3 páginas durante a primeira expansão parcial, e passam a ser formados por 4 páginas, na segunda expansão parcial, e assim segue o crescimento do arquivo.

A generalização desse processo para g expansões parciais a cada expansão total é imediata. O arquivo é criado com $m = gs$ páginas, para algum $s \in N$. Essas páginas são reunidas em s grupos de g páginas cada. Na primeira expansão parcial, os grupos crescem de g para $g + 1$ páginas. Na segunda, de $g + 1$ para $g + 2$ páginas, e assim segue o crescimento dos

grupos até que cada um deles contenha $2g$ páginas. Quando isso ocorrer, a quantidade de páginas do arquivo dobrou, ou seja, ocorreu uma expansão total. Cada vez que ocorre a expansão total, torna-se a reunir as páginas contidas no arquivo em grupos de g páginas, retornando para a primeira expansão parcial.

O grupo a ser dividido, durante a expansão do arquivo, é indicado por um ponteiro denotado por AG. Na criação do arquivo, ele é inicializado com 1. A atualização desse ponteiro é feita pelo algoritmo que executa a expansão do arquivo.

Algoritmo 4.3.2.2_ executa a expansão de um arquivo organizado pelo Hashing linear com expansão parcial.

entrada :

AG : aponta para o grupo a ser dividido
g : número de expansões parciais por expansão total
s : número de grupos
Q : quantidade de páginas alocadas no arquivo
l : expansão parcial em que se encontra o arquivo

```
{ alocar a página Q+1 no arquivo;
  para cada página P contida no grupo AG faça
    ler página P e a sua região de colisão;
  para cada registro r lido faça {
    calcular o endereço de r considerando a página Q+1;
    se a página original de r está cheia
      então
        escrever r na região de colisão de sua página original
      senão
        escrever r em sua página original; }
  para cada página P contida no grupo AG faça
    { /* a página Q+1 foi anexada a esse grupo */
      escrever página P no arquivo;
      escrever a região de colisão da página P no arquivo de colisões; }
  incrementar Q e AG;
  se AG > s então
    { /* terminou a l-ésima expansão parcial */
      incrementar l;
```

```
se l > g então
  { /* ocorreu uma expansão total */
    inicializar l e AG;
    s← 2s; } } }
```

Na contração do arquivo, a última página é liberada. Como AG aponta para o grupo a ser dividido, esta página pertence ao grupo AG−1. Os registros endereçados à última página são distribuídos entre as páginas do grupo AG −1 que permanecem no arquivo. Após a reinserção, a última página é liberada e o ponteiro AG é decrementado.

4.3.2.3_ Função de endereçamento

Um arquivo organizado pelo Hashing linear com expansão parcial possui nível d, se ocorreram exatamente d expansões totais nesse arquivo.

A função de endereçamento distribui uniformemente os registros pelos grupos existentes no arquivo, e os registros endereçados a um grupo são também distribuídos uniformemente entre as páginas que o formam. Por exemplo, considere o arquivo ilustrado na figura 4.3.2.2.2. A função de endereçamento distribue os registros entre os grupos 1, 2 e 3. Os registros endereçados ao grupo 1 são distribuídos entre as páginas 1 e 4, enquanto ele não for dividido. Após a sua divisão, eles são distribuídos entre as páginas 1, 4 e 7. Observando ainda esse arquivo, pode-se notar que a página 4 pertence ao grupo 1 enquanto o nível do arquivo é 0. Mas, quando ele passar para o nível 1, a página 4 pertencerá ao grupo 4. Toda vez que o nível do arquivo mudar tem-se alterações desse gênero. Para saber o grupo que contém a página onde o registro está armazenado, a função de endereçamento calcula os endereços do registro em todos os níveis anteriores do arquivo.

Quando o nível do arquivo é 0 e o arquivo não foi expandido ainda, usa-se uma função de hashing qualquer, para distribuir os registros entre as m páginas. Essa função é denotada por H, e mapeia o conjunto das chaves no conjunto $\{1, \ldots, m\}$. A medida que o arquivo vai crescendo, os grupos vão sendo divididos. Ao dividir um grupo, os registros endereçados às páginas que o formam são uniformemente distribuídos entre elas e a nova página alocada. Para fazer essa distribuicão, a função de endereçamento usa

um conjunto de funções de hashing independentes. Elas são denotadas por $f_0, f_1, \ldots$, e mapeiam o conjunto das chaves no conjunto $\{0, 1, \ldots, 2g-1\}$, onde g é igual ao número de expansões parciais por expansão total. Quando a i-ésima expansão parcial acaba, e o arquivo possui nível d, o endereço de um registro com chave k é definido por:

$$e = l + f_t(k)s$$

onde
l = número do grupo que contém o endereço do registro
s = quantidade de grupos no arquivo
t = menor inteiro maior ou igual a d, tal que $f_t(k) \leq g + i - 1$

O inteiro t é tomado dessa forma para obter-se um endereço válido. O endereço é valido se a página, à qual o registro é endereçado, pertence ao arquivo. No fim da i-ésima expansão parcial, cada grupo possui $g + i$ páginas. Portanto, o endereço é válido se estiver garantido que $f_t(k) \leq g + i - 1$. Note que na última expansão parcial tem-se que $t = d$.

Se, na última expansão parcial do nível d, um registro possui endereço h então, no nível $d + 1$ o seu endereço pertence ao grupo $h \bmod s$, onde s é a quantidade de grupos no nível $d + 1$.

Algoritmo 4.3.2.3_ calcula o endereço de um registro armazenado em um arquivo organizado pelo HLP.

entrada :

k : chave do registro
d : nível do arquivo
l : expansão parcial em que o arquivo está no nível d
g : número de expansões parciais por expansão total
s_0 : número de grupos no arquivo quando é nível igual a 0

saída :

endereço do registro

{ $h \leftarrow H(k)$;
$s \leftarrow s_0$;
para j *de* 0 *até* $d - 1$ *faça*
{ /* calcula endereço do registro no início do nível $j + 1$ */
se $f_j(k) \geq g$ *então*

```
        h ← h mod s + s f_j(k);
      s ← 2s; /* número de grupos no nível j + 1 */ }
   /* calcula endereço do registro na l-ésima expansão parcial
      do nível d */
   t ← d;
   u ← f_d(k);
   enquanto u ≥ quantidade de páginas contidas no grupo h mod s
     faça {
        t ← t + 1;
        u ← f_t(k); }
   /* neste ponto está garantido que f_t(k) ≤ g + l − 1 */
   se u ≥ g   /* registro foi transferido para uma das páginas
                 alocadas no grupo h mod s no nível d */
     então retorna ( h mod s + us )
     senão retorna ( h ); }
```

4.3.2.4_ Outros métodos baseados no HLP

Dois novos métodos são propostos por Larson em [21]. Nos dois, é alterada a forma de tratar a colisão. O primeiro método foi sugerido após observar que as páginas pertencentes aos últimos grupos, aqueles de alto índice, possuem grandes cadeias de páginas de colisão. Isto acontece porque, a ordem dos grupos divididos é linear. Para diminuir o tamanho dessas cadeias é associada, a cada página do arquivo, um conjunto de cadeias de colisão, ou seja, a cadeia única de cada página do HLP original, é quebrada em r cadeias. Através de uma função de hashing é escolhida a cadeia que contém a página que armazena o registro de colisão. Essa função distribui uniformemente os registros de colisão de uma página entre as suas r cadeias. No segundo método não é criado um arquivo para conter apenas os registros de colisão, as páginas de colisão são alocadas no próprio arquivo que contém os registros de dados. Para algum $l \in N$, toda l-ésima página do arquivo é separada para conter os registros de colisão, ou seja, aquelas que possuem número $l, 2l, \ldots, tl$ são reservadas para serem páginas de colisão, onde tl é menor ou igual a quantidade de páginas do arquivo. Elas não fazem parte do espaço de endereçamento do arquivo, isto é a função de endereçamento não associa registros a elas, e os registros armazenados nelas são apenas os

de colisão. Quando todas as páginas de colisão estiverem cheias, o arquivo é expandido. Espera-se que seja liberado espaço nessas páginas após a divisão de um grupo. A cada $(l-1)$ expansões do arquivo uma nova página de colisão é alocada.

Um outro método, que também foi proposto por Larson em [27], são variações feitas no HLP original. As modificações foram feitas visando melhorar a distribuição de armazenagem no arquivo. No HLP original a ordem dos grupos divididos é linear. Com isso, espera-se que as páginas pertencentes aos grupos iniciais (aqueles que já foram divididos) contenham poucos registros. A distribuição esperada dos registros pelo arquivo, quando ele passou por p expansões, está ilustrada na figura abaixo.

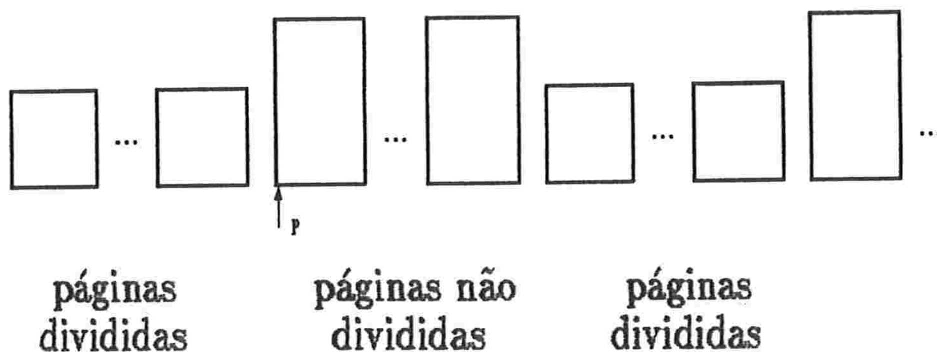

figura 4.3.2.4.1_ *distribuição de armazenagem num arquivo organizado pelo HLP*

Para evitar muitos grupos consecutivos com alto fator de ocupação Larson propõe mudar a ordem dos grupos divididos. Suponha que o arquivo possui s grupos. Tome $r \in Z_+$, tal que $r < s$. A ordem para dividir os grupos é definida como sendo $s, s-r, s-2r, \ldots, s-1, s-1-r, s-1-2r, \ldots, s-2, \ldots$. Assim, espera-se que em qualquer conjunto de r páginas consecutivas haja pelo menos uma página que não está cheia. Com essa observação é natural mudar o tratamento das colisões para ser feito pelo "linear probing", pois se ocorrer colisão em uma página espera-se encontrar espaço para armazenar o seu registro de colisão entre as r próximas páginas.

4.3.3_ HASHING LINEAR COM EXPANSÃO EM GRUPO [46]

4.3.3.1_ Descrição geral do método

Assim como no Hashing linear com expansão parcial (HLP), o Hashing linear com expansão em grupo (HLG) agrupa as páginas do arquivo em grupos e, na sua expansão um único grupo é dividido. No entanto, o processo de expansão do arquivo neste método é mais simples do que no HLP.

O arquivo é criado com m páginas, onde $m = gs_0$ e $g, s_0 \in N$. Essas páginas são agrupadas em s_0 grupos de g páginas cada. O inteiro g é fixo e definido na criação do arquivo. O grupo j, $1 \leq j \leq s_0$, é formado pelas páginas $j, j + s_0, j + 2s_0, \ldots, j + (g-1)s_0$. A figura abaixo ilustra um arquivo criado com $m = 6$ páginas agrupadas em 3 grupos de 2 páginas cada ($g = 2$). O grupo 1 é formado pelas páginas 1 e 4. O grupo 2, pelas páginas 2 e 5 e as páginas 3 e 6 formam o grupo 3. O grupo 1 está indicado na figura.

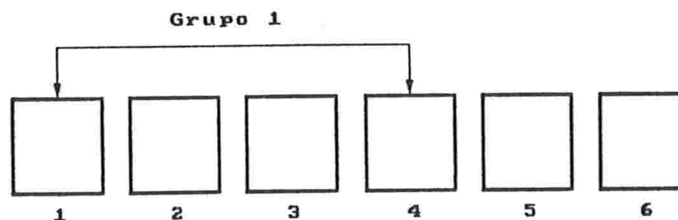

figura 4.3.3.1.1_ *arquivo criado com* 6 *páginas no HLG*

Os registros são distribuídos uniformemente pelas m páginas do arquivo, ou seja, o espaço de endereçamento do arquivo inicial é constituído pelas m páginas. Quando o fator de ocupação do arquivo atingir um limite pré-determinado o arquivo é expandido. A página $m+1 = s_0g+1$ é alocada. O grupo 1 é dividido, isto é, os registros endereçados às páginas desse grupo são distribuídos uniformemente entre elas e a nova página. Essa página é anexada ao grupo 1, ou seja, o grupo 1 passa a ser formado pelas páginas $1, 1 + s_0, \ldots, 1 + gs_0$. A figura abaixo ilustra o arquivo da figura 4.3.3.1.1 após a sua primeira expansão. A página 7 foi alocada para o arquivo e incorporada ao grupo 1, que é indicado na figura.

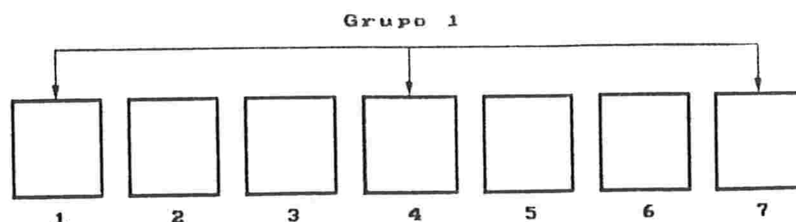

figura 4.3.3.1.2_ primeira expansão de um arquivo organizado pelo HLG

Na próxima expansão do arquivo o grupo 2 é dividido. Depois divide-se o grupo 3, o 4 e assim por diante. Ao dividir um grupo, uma nova página é alocada e anexada a ele. Ele cresce de g para $g+1$ páginas. Após dividir todos os grupos, volta-se ao estágio inicial. Toma-se grupos de g páginas, e faz sua divisão na ordem linear.

O tratamento da colisão, neste método, é feito pelo encadeamento das páginas de colisão. No primeiro tratamento cria-se um arquivo para conter os registros de colisão, que são aqueles endereçados às páginas cheias.

4.3.3.2_ Processo de expansão e contração

O critério usado para expandir um arquivo organizado pelo Hashing linear com expansão em grupo (HLG) é a divisão controlada.

Inicialmente, o arquivo possui s_0 grupos de g páginas cada. Na expansão do arquivo um único grupo é dividido. A ordem para dividir os grupos é linear, ou seja, primeiro divide-se o grupo 1, depois o 2 e assim por diante até dividir o grupo s_0. A página alocada durante a expansão é anexada ao grupo dividido. Após a divisão do grupo s_0, todos os grupos do arquivo foram divididos. No HLP, parte-se para segunda expansão parcial o que não acontece aqui. Neste método, volta-se para a situação inicial, ou seja, as páginas do arquivo são reunidas em grupos de g páginas. O arquivo neste momento possui $m + s_0 = (g+1)s_0$ páginas. Como $m + s_0$ não é necessariamente divisível por g, são alocadas r páginas extras para o arquivo, onde $0 \leq r < g$. Dessa forma, retorna-se para a situação inicial reunindo as páginas do arquivo em s_1 grupos de g páginas onde:

$$s_1 = \lceil \frac{(g+1)s_0}{g} \rceil$$

Essas novas páginas são chamadas de ***páginas ajustadoras***. Elas não são imediatamente incorporadas no espaço de endereçamento do arquivo, ou seja,

a função de endereçamento não associa registros para elas. Isto ocorrerá, quando os grupos aos quais elas pertencem forem divididos. Esse inicial desperdício de espaço é irrelevante, uma vez que a quantidade de páginas ajustadoras é limitada pela constante g, e esta é desprezível comparada com o tamanho do arquivo.

O grupo a ser dividido, durante a expansão, é indicado por um ponteiro denotado por AG. Na criação do arquivo, ele é inicializado com 1, aponta para o grupo 1. A atualização desse ponteiro é feita pelo algoritmo que expande o arquivo.

Note que se $g = 1$ tem-se o HL.

Algoritmo 4.3.3.2_ executa a expansão de um arquivo organizado pelo HLG.

entrada :

AG : ponteiro para o grupo a ser dividido
Q : quantidade de páginas do arquivo
s : quantidade de grupos no arquivo
g : número de páginas por grupo

{ alocar a página Q+1;
para cada página P do grupo AG *faça*
ler a página P e a sua região de colisão;
para cada registro r lido *faça* {
calcular o endereço do registro r considerando a página Q+1;
se a página original de r está cheia
então
escrever r na região de colisão de sua página original
senão
escrever r na sua página original; }
incrementar AG e Q;
se $AG = s + 1$ *então*
{ /* retorna para estágio inicial */
$s \leftarrow \lceil \frac{s(g+1)}{g} \rceil$; /* altera quantidade de grupos */
inicializar AG;
alocar as páginas ajustadoras;
atualizar o valor Q; } }

Na contração do arquivo, a última página alocada é liberada. Como AG aponta para o grupo a ser dividido, esta página pertence ao grupo AG−1. Os registros endereçados a esta página são distribuídos entre as outras páginas do grupo AG−1. Após a reinserção desses registros a última página é liberada e o ponteiro AG é decrementado. Quando AG= 0 libera-se as páginas ajustadoras, e AG aponta para o último grupo do arquivo.

4.3.3.3_ Função de endereçamento

Um arquivo organizado pelo Hashing linear em grupo (HLG) possui profundidade d se a quantidade de grupos foi alterada d vezes. Quando o grupo possui s_i grupos a sua profundidade é i.

A função de endereçamento distribui os registros pelos grupos existentes no arquivo. Em cada grupo, os registros endereçados a ele são distribuídos entre as páginas que o formam. Por exemplo, quando o arquivo possui $d = 1$ os registros são distribuídos pelos s_1 grupos. Os registros endereçados ao grupo j são distribuídos entre as páginas $j, j+s_1, \ldots, j+s_1 g$, onde esta última página pertence ao grupo j somente depois da sua divisão.

Note que, a página $s_0 + 1$ pertence ao grupo 1 enquanto a profundidade do arquivo é 0. Quando esta mudar para 1, isto é, o arquivo for composto por s_1 grupos, esta página fará parte do grupo $s_0 + 1$. Sempre que a profundidade do arquivo mudar tem-se alterações desse genero. Para saber o grupo que contém o endereço do registro na profundidade atual do arquivo, a função de endereçamento calcula o endereço do registro nas profundidades anteriores passadas pelo arquivo. Se e é o endereço de um registro na profundidade d, na profundidade $d + 1$ a página e pertence ao grupo e mod s_{d+1}. Através de uma função de hashing, a qual denotamos por H, os registros são distribuídos entre as m páginas alocadas na criação do arquivo. Assim, H mapeia o conjunto das chaves no conjunto $\{1, 2, \ldots, s_0 g = m\}$. Quando um grupo é dividido os registros endereçados a ele são distribuídos entre as páginas que o formam. Essa distribuição é feita através de uma função de hashing, que mapeia o conjunto das chaves no conjunto $\{0, \ldots .g\}$. Para que essa distribuição não seja viciada, usa-se funções de hashing distintas para profundidades distintas, ou seja, toma-se um conjunto de funções de hashing, denotadas por $f_0, f_1, \ldots$, onde a função f_i é usada na profundidade i.

A definição formal do endereço de um registro com chave k é dada a seguir. Seja g_d o número do grupo que contém o endereço do registro, quando o arquivo possui profundidade d. O endereço do registro quando a profundidade do arquivo é d, é definido indutivamente da seguinte forma:
se o grupo g_d não foi dividido na profundidade d

$$e_{(-d)} = \begin{cases} H(k) & \text{se } d = 0 \\ e_{d-1} & \text{se } d > 0 \end{cases}$$

se o grupo g_d foi dividido na profundidade d

$$e_d = e_{(-d)} \bmod s_d + f_d(k) s_d$$

Note que, $e_{(-d)} \bmod s_d = g_d$ e $e_{(-d)} = e_{d-1}$.

Algoritmo **4.3.3.3_** calcula o endereço de um registro num arquivo organizado pelo HLG.

entrada :
  k : chave do registro
  g : quantidade de páginas por grupo
  d : profundidade do arquivo
  s_0 : quantidade de grupos quando $d = 0$

saída :
  endereço do registro

{ $e \leftarrow H(k)$;
  $s \leftarrow s_0$;
  para j *de* 0 *a* $d - 1$ *faça*
    { /* calcula endereço do registro no início da profundidade $j + 1$ */
      $e \leftarrow e \bmod s + f_j(k)s$;
      $s \leftarrow \lceil \frac{s(g+1)}{g} \rceil$; } /* número de grupos na profundidade $j + 1$ */
  se grupo que contém e já foi dividido
    então retorna ($e \bmod s + f_d(k)s$)
    senão retorna (e); }
}

4.3.4_ HASHING COM ARMAZENAMENTO EM ESPIRAL [38]

4.3.4.1_ Descrição geral do método

Todos os métodos descritos anteriormente fazem a distribuição uniforme dos registros pelas páginas do espaço de endereçamento do arquivo. No Hashing com armazenamento em espiral (HAE), a função de endereçamento não distribui uniformemente os registros pelo arquivo. As páginas que estão no início do arquivo possuem maior probabilidade de armazenar registros do que aquelas que estão no fim, ou seja, espera-se que a página i do arquivo contenha mais registros do que a página $i+1$. Na expansão do arquivo, d novas páginas são alocadas no fim, e a primeira página é liberada. Denomina-se o inteiro d por *fator de crescimento* do arquivo. Os registros endereçados à primeira página são distribuídos de maneira não uniforme entre as d novas páginas. Na realidade, o fator de crescimento do arquivo pode ser real, mas para facilitar a exposição vamos assumir que é inteiro. Note que a generalização para real é imediata.

4.3.4.2_ Processo de expansão e contração

O critério para expandir um arquivo organizado pelo Hashing com armazenamento em espiral é a divisão controlada. O arquivo é criado com $d-1$ páginas, isto é, na sua criação são alocadas as páginas $1, 2, \ldots, d-1$. Na primeira expansão do arquivo, a página 1 é liberada, as páginas $d, d+1, \ldots, 2d-1$ são alocadas, e o arquivo passa a ser formado pelas páginas $2, 3, \ldots, 2d-1$. Os registros endereçados à página 1 são distribuídos não uniformemente entre as d novas páginas. Na segunda expansão do arquivo a página 2 é liberada e as páginas $2d, \ldots, 3d-1$ são alocadas, ou seja, as páginas $3, \ldots, 3d-1$ formam o arquivo. Os registros endereçados à página 2 são não uniformemente distribuídos pelas novas páginas; e assim segue o crescimento do arquivo.

Note que, se $d > 2$ este método não pertence a classe hashing linear, isto é, muitas páginas são alocadas quando ocorre a expansão do arquivo.

A página a ser liberada durante a expansão é indicada por um ponteiro denotado por AP. Na criação do arquivo ele é inicializado com 1,

ou seja, aponta para a página 1. A atualização desse ponteiro é feita pelo algoritmo que executa a expansão do arquivo.

Algoritmo 4.3.4.2_ executa a expansão de um arquivo organizado pelo HAE.

entrada :

d : fator de crescimento do arquivo

AP : aponta para a página a ser liberada

comentário : o tratamento das colisões será dado na seção 4.3.4.3.

```
{ alocar d novas páginas no fim do arquivo;
  ler página AP e a sua região de colisão;
  liberar página AP;
  para cada registro r lido faça {
     calcular endereço do registro r considerando as novas páginas;
     se página original de r está cheia
        então tratar a colisão
        senão escrever r em sua página original; }
  escrever as d novas páginas no arquivo;
  incrementar AP; }
```

Note que, se a primeira página do arquivo possui número p então ele é formado pelas páginas $p, p+1, \ldots, dp-1$.

A contração do arquivo também é bastante simples. As d últimas páginas do arquivo são liberadas, e os seus registros são armazenados em uma nova página alocada no início do arquivo. O ponteiro AP é decrementado.

4.3.4.3_ Função de endereçamento

Considere a espiral exponencial $g(x) = d^x$, onde $x \in R$. Tome um real s, tal que $\lfloor d^s \rfloor$ é o número da primeira página do arquivo. Considere a revolução completa na espiral (def. 1.10) que inicia-se no ponto $(s, g(s))$. Esta revolução é quebrada em setores que são mapeados para as páginas do arquivo. A cada registro é associado uma direção. Os registros são uniformemente distribuídos entre todas as direções. Como cada direção encontra uma única vez qualquer revolução completa da espiral, pode-se

definir o endereço do registro como sendo a página associada ao setor que encontra a sua direção. Na figura abaixo está ilustrada a função de endereçamento. Nela ilustramos uma revolução completa e as direções j_1, j_2 e j_3. Essas direções estão associadas respectivamente aos registros de chave k_1, k_2 e k_3. A direção j_1 cruza a revolução dada no setor s_1. Este setor está associado à página p. Portanto, o endereço do registro é p.

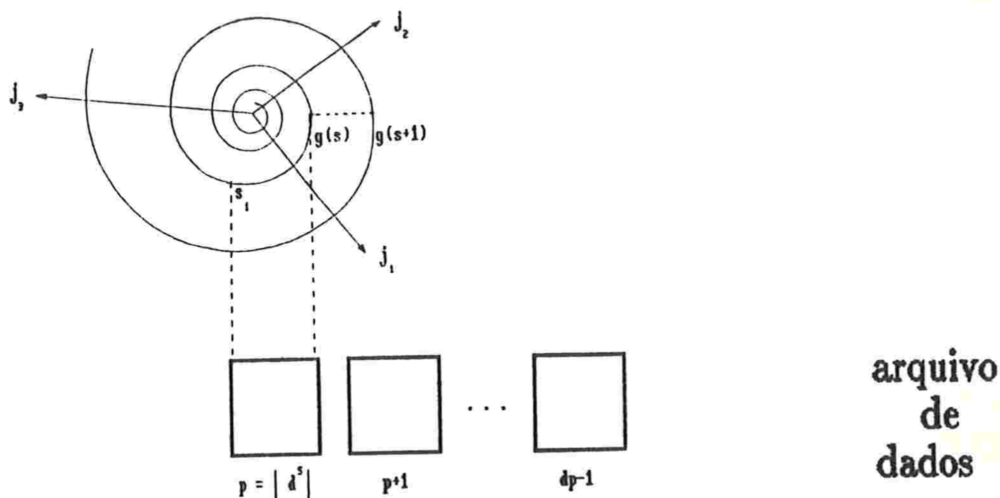

figura 4.3.4.3.1_ *modelo geral da função de endereçamento do HAE*

A direção de registro é dada por uma função de hashing qualquer, denotada por h, que mapeia o conjunto das chaves no intervalo real $[0, 1)$. A direção $h(k)$ do registro de chave k cruza a revolução da espiral que está sendo considerada, no ponto de abcissa $H = \lceil s - h(k) \rceil + h(k)$ e coordenada $g(H)$.

O setor da revolução considerada, que está associado a uma página p, é definido como sendo o conjunto dos pontos (x, d^x), tal que $\lfloor d^x \rfloor = p$. Portanto, o endereço de um registro de chave k é a página $\lfloor d^H \rfloor$ onde $H = \lceil s - h(k) \rceil + h(k)$.

Observe que o arquivo possui $d^{s+1} - d^s$ páginas, quando a primeira página possui número $\lfloor d^s \rfloor$.

Quando o arquivo ilustrado na figura 4.3.4.3.1 for expandido, tem-se que a página p é liberada e uma nova área, d vezes maior, é alocada. A figura abaixo ilustra a correspondência entre o arquivo expandido e a espiral. Note que a revolução a ser considerada mudou, ela agora é limitada pelos pontos $g(a) = c_1$ e $g(a + 1) = c_2$ onde $a \in R$ e $p + 1 = \lfloor d^a \rfloor$. Note também, que o endereço do registro que possui direção j_1 mudou.

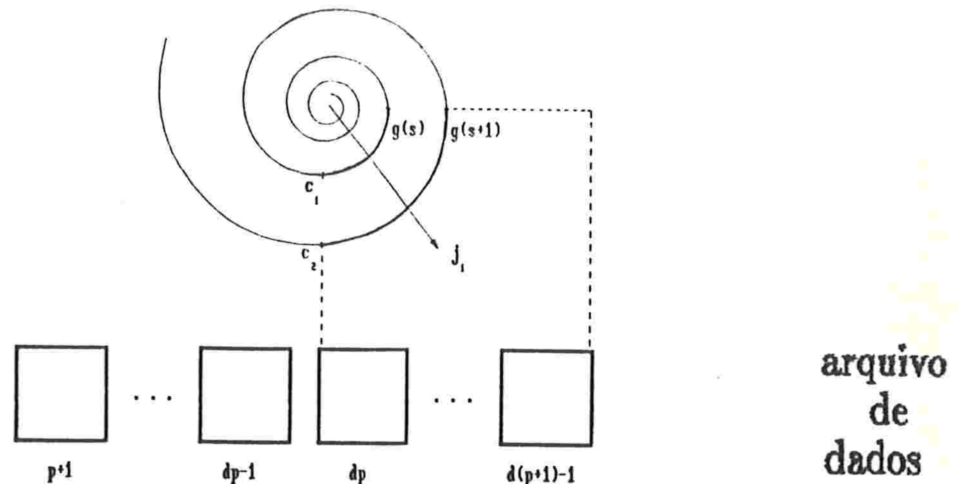

figura 4.3.4.3.2_ *associação da espiral com um arquivo após a expansão*

Pela definição de endereço, tem-se que os registros endereçados a uma página q são aqueles que possuem chave k tal que

$$q \leq d^H < q+1 \qquad \text{onde} \quad H = \lfloor s - h(k) \rfloor + h(k)$$

Portanto,

$$\log_d q \leq H < \log_d (q+1) \qquad \text{onde} \quad H = \underbrace{\lfloor s - h(k) \rfloor}_{\text{parte inteira}} + \underbrace{h(k)}_{\text{parte decimal}}$$

Como os números da páginas do arquivo são inteiros, tem-se que

$$f(\log_d q) \leq h(k) < f(\log_d (q+1))$$

onde a função f é aquela que devolve a parte decimal de um número real. Por exemplo, $f(3.1435) = 0.1435$.

Concluindo, os registros endereçados à página q são aqueles que possuem chave k tal que $f(\log_d q) \leq h(k) < f(\log_d (q+1))$.

Para exemplificar, considere um arquivo com 5 páginas e fator de crescimento igual a 2 ($d = 2$). Como $2^{s+1} - 2^s = 5 \Rightarrow s = \log_2 5$ e a primeira página alocada possui número $\lfloor 2^s \rfloor = \lfloor 2^{\log_2 5} \rfloor = 5$. As páginas $5, 6, 7, 8$ e 9 estão alocadas no arquivo. A tabela abaixo contém para cada página do arquivo o seu correspondente intervalo de direções (valores de $h(k)$), e a sua

esperada porcentagem de registros, que é dada pela diferença dos limites do intervalo. Note que, $\log_2 5 = 2,3219$, $\log_2 6 = 2,5850$, e assim por diante.

páginas	intervalo de $h(k)$	porcentagem de registros
5	$[0.3219, 0.5850)$	26,31%
6	$[0.5850, 0.8074)$	22,24%
7	$[0.8074, 1.0000)$	19,26%
8	$[0.0000, 0.1699)$	16,99%
9	$[0.1699, 0.3219)$	15,20%

Quando este arquivo for expandido, a página 5 é liberada e as páginas 10 e 11 são alocadas. Note que é razoável escolher a página 5 para ser liberada, pois é a que está associada ao maior intervalo. Ou seja, é ela que possui a maior probabilidade de armazenar registros. O intervalo associado a ela é quebrado em dois intervalos na expansão. O intervalo $[0.3219, 0.4594)$ é associado à página 10 e o intervalo $[0.4594, 0.5850)$ à página 11, provenientes de $\log_2 10 = 3,3219$, $\log_2 11 = 3,4594$ e $\log_2 12 = 3,5850$. Isto é, a probabilidade de armazenar registros da página 5 é dividida em duas partes desiguais. A parcela maior correspoderá à página 10 e a outra à página 11. E essas páginas são aquelas que terão a menor probabilidade de armazenar registros do arquivo.

Algoritmo 4.3.4.3_ calcula o endereço do registro num arquivo organizado pelo HAE.

entrada :

d : fator de crescimento

k : chave do registro

s : ponto de início da revolução completa

saída :

endereço do registro

$\{ \; H = \lceil s - h(k) \rceil + h(k);$
retorna $(\lfloor d^H \rfloor); \}$

O tratamento da colisão no HAE é feita pelo "random probing". Se a página original do registro está cheia, tenta-se armazená-lo em outra página do arquivo. Se a segunda também estiver cheia, tenta-se uma terceira e assim por diante até encontrar uma página que contenha espaço

para acolhê-lo. Essas páginas são definidas pela função de endereçamento, trocando-se apenas a direção do registro. Ou seja, toma-se um conjunto de funções de hashing, as quais denotamos por $h_0, h_1, \ldots$, que mapeiam o conjunto das chaves no intervalo $[0, 1)$. Cada função define uma direção do registro. Assim, para cada registro é definida uma sequência infinita de direções $j_0, j_1, \ldots$. Se a página $\lfloor d^{H_0} \rfloor$, onde $H_0 = \lceil s - j_0 \rceil + j_0$, está cheia, tenta-se a página $\lfloor d^{H_1} \rfloor$, onde $H_1 = \lceil s - j_1 \rceil + j_1$, e assim por diante.

Capítulo V

UM NOVO MÉTODO DE HASHING

Neste capítulo iremos apresentar um novo método de hashing dinâmico linear não indexado, que denominaremos de *Hashing linear decimal com distribuição não uniforme* (HLD). O nome "decimal" provém do fato de se poder também fazer uma versão "octal", apresentada na seção 5.5.

Assim como o Hashing com armazenamento em espiral (HAE), o Hashing linear decimal com distribuição não uniforme faz a distribuição não uniforme dos registros pelas páginas do arquivo, e a página a ser dividida é aquela que tem a maior probabilidade de conter mais registros. Com isso, esperamos diminuir a quantidade de registros de colisão no arquivo, pois na próxima expansão divide-se a página que deveria conter a maior quantidade de registros de colisão, e assim espera-se acabar com os seus registros de colisão. Como somente os registros de colisão necessitam mais de um acesso ao disco para serem localizados, espera-se que o custo médio da procura com sucesso, no HLD, esteja bem próximo de 1. O HAE é linear

somente se o fator de crescimento do arquivo for igual a 2. No método que vamos apresentar, o arquivo cresce por alocação de uma única página. Isto é bom, pois a variação do fator de ocupação do arquivo antes e depois da expansão é pequena, principalmente quando o arquivo é grande. Com isso, o fator esperado de utilização do espaço está bastante próximo do real. Outra vantagem desse método em relação ao HAE é que neste são alterados os dois extremos do arquivo durante a sua expansão. A primeira página do arquivo é liberada e novas páginas são alocadas no fim. Com isso, é necessário que o sistema que executa as funções de liberação e alocação de páginas, execute essas funções nos dois extremos do arquivo, e isso é uma tarefa bastante difícil. Em [38] Martin apresenta uma maneira de não liberar as páginas do início do arquivo, tornando o HAE mais complicado. Não iremos descrever aqui a forma como isto é feito, o leitor interessado deve procurar a referência dada. No HLD somente um extremo do arquivo é alterado.

A seguir iremos descrever o Hashing linear decimal com distribuição não uniforme. Na seção 5.1 é apresentada uma descrição geral desse método. Na seção 5.2 é dada a forma como o arquivo é expandido. Na seção 5.3 descrevemos a função de endereçamento do arquivo e na seção 5.4 provamos que o Hashing linear decimal com distribuição não uniforme executa a distribuição não uniforme dos registros pelo arquivo. Na seção 5.5 são dadas sugestões para modificar o Hashing linear decimal com distribuição não uniforme.

HASHING LINEAR DECIMAL COM DISTRIBUIÇÃO NÃO UNIFORME

5.1_ Descrição geral do método

No Hashing linear decimal com distribuição não uniforme o arquivo é criado com uma página. Os registros são armazenados nesta página

até que o fator de ocupação do arquivo atinga um limite predeterminado α. Quando isso acontecer, uma nova página, de número 2, é alocada e a página 1 é dividida. Os registros endereçados à página 1 são distribuídos não uniformemente entre as páginas 1 e 2. O algoritmo de endereçamento associa à página 1 maior probabilidade de receber registros. Assim, quando o fator de ocupação do arquivo atingir novamente o limite α, espera-se que a página 1 contenha mais registros do que a página 2. Por essa razão probabilística escolhemos a página 1 para ser dividida novamente nesta expansão do arquivo. Na expansão, uma nova página, de número 3, é alocada. Os registros endereçados à página 1 são distribuídos não uniformemente entre as páginas 1 e 3. O algoritmo de endereçamento associa maior probabilidade de receber registros à página 1 do que à página 3, e ambas menores do que a probabilidade da página 2, ou seja, a página 2 é aquela que possui a maior probabilidade de receber registros. Além disso, devido à divisão, as páginas 1 e 3 devem conter menos registros do que a página 2. Assim sendo, quando o fator de ocupação limite for atingido novamente a página 2 é escolhida para sofrer a divisão, pois probabilisticamente é ela que deve ter mais registros. Nessa expansão, a página 4 é alocada. Os registros endereçados à página 2 são distribuídos não uniformemente entre ela e a página 4. A probabilidade de endereçar registros à página 2 é alterada, ficando maior do que a probabilidade da página 4 e menor do que a da 3 e da 1. O processo de expansão do arquivo prossegue dessa forma. Na expansão do arquivo, a página a ser dividida é aquela que possui a maior probabilidade de receber registros. A distribuição dos registros endereçados à página dividida é feita de maneira não uniforme entre esta e a nova página. Após a divisão, essas páginas são aquelas que possuem a menor probabilidade de receber registros.

A árvore abaixo ilustra um arquivo após 5 divisões. O arquivo possui as páginas com seus números representados nos nós folhas. Os índices dos nós internos da árvore representam o número da página que foi dividida. As páginas, com probabilidade decrescente de armazenar registros são: $2, 4, 1, 5, 3$ e 6. Portanto, durante a próxima expansão do arquivo a página 2 é aquela que será dividida.

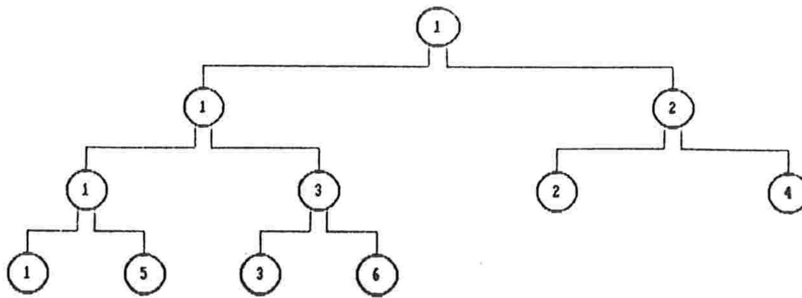

figura 5.1.1_ *páginas de um arquivo após* 5 *divisões*

O tratamento de colisão é feito pelo encadeamento de páginas de colisão. Um arquivo, chamado de arquivo de colisões, é criado para conter os registros de colisão das páginas pertencentes ao arquivo de dados. O arquivo de colisões é criado durante o tratamento da primeira colisão. Os registros de colisão de uma página são aqueles a ela endereçados quando ela está cheia.

5.2_ Processo de expansão e contração

Vamos denotar a quantidade de páginas contidas no arquivo por Q.

Diremos que um arquivo possui *nível* D se D é o menor inteiro tal que $2^D \geq Q$.

A expansão de um arquivo organizado pelo Hashing linear decimal com distribuição não uniforme ocorre da seguinte forma: quando o fator de ocupação do arquivo atinge um limite pré-determinado α uma página é dividida e uma nova é alocada no fim do arquivo. Essa nova página recebe alguns registros da página dividida. A função de endereçamento faz com que as probabilidades dessas páginas receberem novos registros sejam as menores do arquivo.

Antes de descrever o algoritmo que executa a expansão do arquivo, precisamos definir a ordem das páginas do arquivo a serem divididas. Ou seja, precisamos saber qual é a página do arquivo que possui a maior probabilidade de estar cheia. Isso é fornecido por uma função que é construída na seção 5.2.1. Essa função devolve o número da página a ser dividida. Na seção 5.4 prova-se que, essa função devolve o número da página que produz a distribuição de probabilidade desejada no arquivo.

5.2.1_ Definição do número da página dividida

O número da página a ser dividida é dado por uma função que é apresentada nesta seção. Para definir esta função, precisamos de alguns conceitos que são dados a seguir. A definição dessa função é dada no lema 5.2.1.3.

Seja $n \in N$. Denotaremos por I_n o conjunto dos naturais menores que 10^n, ou seja $I_n = [0 \;..\; 10^n - 1]$.

Dado I_n, vamos construir indutivamente, um conjunto de subconjuntos de I_n denotado por $P(I_n)$, da seguinte forma:

i- $P(I_0) = \{J_1\}$ onde $J_1 = [0 \;..\; 0]$

ii- Dado $P(I_n) = \{J_1, J_2, \ldots, J_{2^n}\}$ *onde* $J_i = [a_i \;..\; b_i]$ *para* $1 \leq i \leq 2^n$

construímos $P(I_{n+1}) = \{L_1, L_2, \ldots, L_{2^{n+1}}\}$ onde

$L_{2i-1} = [10a_i \;..\; 5a_i + 5b_i + 5 + \lfloor 2^{n-1} \rfloor]$

$L_{2i} = [5a_i + 5b_i + 6 + \lfloor 2^{n-1} \rfloor \;..\; 10b_i + 9]$

Antes de provar que $P(I_n)$ é uma partição de I_n (def 1.10), para qualquer $n \in N$, vamos construir alguns conjuntos $P(I_n)$ para uma maior familiarização com eles, facilitando assim as descrições a seguir. Na tabela abaixo estão representados os conjuntos I_n e $P(I_n)$ para alguns valores de n.

n	I_n	$P(I_n)$
0	[0 .. 0]	{[0 .. 0]}
1	[0 .. 9]	{[0 .. 5], [6 .. 9]}
2	[0 .. 99]	{[0 .. 31], [32 .. 59], [60 .. 81], [82 .. 99]}
3	[0 .. 999]	{[0 .. 162], [163 .. 319], [320 .. 462], [463 .. 599], [600 .. 712], [713 .. 819], [820 .. 912], [913 .. 999]}

tabela 5.2.1.1_ *descrição de* $P(I_n)$ *para* $n = 0, 1, 2, 3$

Lema 5.2.1.1: *Para qualquer* $n \in N$ *temos que* $P(I_n)$ *é uma partição de* I_n.

Prova:

Vamos fazer indução sobre n.

Se $n = 0$ temos que $P(I_0)$ é uma partição de I_0 por construção.

Por hipótese de indução, temos que $P(I_n)$ é uma partição de I_n, onde $P(I_n) = \{J_1, \ldots, J_{2^n}\}$ e $J_i = [a_i \mathrel{..} b_i]$ para $1 \le i \le 2^n$.

Por construção temos que $P(I_{n+1}) = \{L_1, \ldots, L_{2^{n+1}}\}$ onde

$L_{2i-1} = [10a_i \mathrel{..} 5a_i + 5b_i + 5 + \lfloor 2^{n-1} \rfloor]$

$L_{2i} = [5a_i + 5b_i + 6 + \lfloor 2^{n-1} \rfloor \mathrel{..} 10b_i + 9]$. Portanto,

$x \in I_{n+1} \iff \lfloor \frac{x}{10} \rfloor \in I_n \iff$ existe um único $i \in N \mid J_i \in P(I_n)$ e $\lfloor \frac{x}{10} \rfloor \in J_i \iff$ existem únicos $i, a_i, b_i \in N \mid J_i = [a_i \mathrel{..} b_i]$ e $a_i \le \lfloor \frac{x}{10} \rfloor \le b_i$ $\iff$ existe único $i \in N \mid 10a_i \le x \le 10b_i + 9 \iff$ ou $x \in L_{2i-1}$ ou $x \in L_{2i}$.

Portanto, $I_{n+1} = \cup_{i=1}^{2^{n+1}} L_i$.

Seja $x \in L_i \cap L_j$ onde $i < j \Rightarrow \lfloor \frac{x}{10} \rfloor \in J_{\lceil \frac{i}{2} \rceil}$ e $\lfloor \frac{x}{10} \rfloor \in J_{\lceil \frac{j}{2} \rceil}$.

Como $P(I_n)$ é uma partição de $I_n \Rightarrow \lceil \frac{i}{2} \rceil = \lceil \frac{j}{2} \rceil \Rightarrow$ $j = i + 1$ e $j = 2l$ para algum $l \in N$. Mas por construção temos que $L_{2l-1} \cap L_{2l} = \{\ \}$.

Portanto, não há $x \in I_{n+1} \mid x \in L_i \cap L_j$ para $i < j$.

Logo $P(I_n)$ é uma partição de I_n para qualquer $n \in N$.

□

Lema 5.2.1.2: *Seja $n \in N$ e $P(I_n) = \{J_1, J_2, \ldots, J_{2^n}\}$ a partição de I_n. Para qualquer $c_i \in J_i$ temos $c_1 < c_2 < c_3 < \ldots < c_{2^n}$.*

Prova:

Vamos fazer indução sobre n.

Se $n = 1$ a tese é válida por construção (ver tabela 5.2.1.1).

Pela hipótese de indução, temos que para qualquer $c_i \in J_i = [a_i \mathrel{..} b_i]$ e $J_i \in P(I_n)$ então $c_1 < c_2 < \ldots < c_{2^n}$.

Considere $P(I_{n+1}) = \{L_1, \ldots, L_{2^{n+1}}\}$ e $d_i \in L_i$ para $1 \le i \le 2_{n+1}$

Por construção de $P(I_{n+1})$ temos que

$$d_{2i-1} < d_{2i} \text{ para } 1 \le i \le 2^n$$

Portanto, basta provar que $d_{2i} < d_{2i+1}$ para $1 \le i \le 2^n$.

Por construção $d_{2i} \le 10b_i + 9$, $d_{2i+1} \ge 10a_{i+1}$ para $1 \le i < 2^n$. Por hipótese de indução $b_i < a_{i+1}$ para $1 \le i < 2^n \Rightarrow b_i + 1 \le a_{i+1}$. Logo, para todo $i \in [1 \mathrel{..} 2^n]$

$$d_{2i} \le 10b_i + 9 < 10b_i + 10 = 10(b_i + 1) \le 10a_i \le d_{2i+1}$$

Logo, $d_1 < d_2 < \ldots < d_{2^{n+1}}$.

□

Pelo lema anterior podemos concluir que a construção de $P(I_n)$ é feita de forma que os seus elementos possuem uma ordenação bem definida.

Lema 5.2.1.3: *Considere a família de funções definidas recursivamente da seguinte forma:*

$$f_0 : P(I_0) \to \{1\}$$

$$f_0(J_1) = 1$$

Tomando-se $P(I_{n-1}) = \{L_1, \ldots, L_{2^{n-1}}\}$ *definimos* f_n *por:*

$$f_n : P(I_n) \to [1 \ .. \ 2^n]$$

$$f_n(J_i) = \begin{cases} f_{n-1}(L_{\frac{i+1}{2}}) & \text{se } i \text{ é ímpar} \\ 2^{n-1} + \frac{i}{2} & \text{se } i \text{ é par} \end{cases}$$

Para qualquer $n \in N$ *a função* f_n *é bijetora.*

Prova:

Vamos fazer indução sobre n.

Se $n = 0$, por construção temos que f_0 é bijetora.

Por hipótese de indução, temos que f_n é uma função bijetora.

Sejam

$P_1(I_{n+1}) = \{J_i \mid J_i \in P(I_{n+1}) \text{ e } i \text{ é ímpar}\}$

$P_2(I_{n+1}) = \{J_i \mid J_i \in P(I_{n+1}) \text{ e } i \text{ é par}\}$

$P(I_n) = \{L_1, \ldots, L_{2^n}\}$

Temos que

$P_1(I_{n+1}) \cup P_2(I_{n+1}) = P(I_{n+1})$

Como $|P(I_{n+1})| = 2^{n+1} = |[1 \ .. \ 2^{n+1}]|$ basta provar que f_{n+1} é injetora para concluir que ela é bijetora.

Sejam $J_i, J_j \in P_1(I_{n+1})$

$$\text{se } J_i \neq J_j \underbrace{\Rightarrow}_{P(I_{n+1}) \text{ é partição}} i \neq j \Rightarrow \frac{i+1}{2} \neq \frac{j+1}{2} \underbrace{\Rightarrow}_{P(I_n) \text{ é partição}}$$

$$\Rightarrow L_{\frac{i+1}{2}} \neq L_{\frac{j+1}{2}}$$

$$\text{como } f_{n+1}(J_i) = f_n(L_{\frac{i+1}{2}}) \text{ e } f_{n+1}(J_j) = f_n(L_{\frac{j+1}{2}}) \underbrace{\Rightarrow}_{f_n \text{ é bijetora}}$$

$f_{n+1}(J_i) \neq f_{n+1}(J_j)$

Portanto, se $J_i, J_j \in P_1(I_{n+1}) \mid J_i \neq J_j \Rightarrow f_{n+1}(J_i) \neq f_{n+1}(J_j)$.

Sejam $J_i, J_j \in P_2(I_{n+1})$

se $J_i \neq J_j \Rightarrow i \neq j \Rightarrow \frac{i}{2} \neq \frac{j}{2} \Rightarrow 2^n + \frac{i}{2} \neq 2^n + \frac{j}{2} \Rightarrow f_{n+1}(J_i) \neq f_{n+1}(J_j)$.

Portanto, se $J_i, J_j \in P_2(I_{n_1}) \mid J_i \neq J_j \Rightarrow f_{n+1}(J_i) \neq f_{n+1}(J_j)$.

Sejam $J_i \in P_1(I_{n+1})$ *e* $J_j \in P_2(I_{n+1})$ temos que

$f_{n+1}(J_i) \leq 2^n$ *e* $f_{n+1}(J_j) = 2^n + \frac{j}{2} > 2^n \Rightarrow f_{n+1}(J_i) \neq f_{n+1}(J_j)$.

Portanto, para quaisquer $J_i, J_j \in P(I_{n+1})$ temos que

$f_{n+1}(J_i) \neq f_{n+1}(J_j) \Rightarrow f_{n+1}$ é injetora $\Rightarrow f_{n+1}$ é bijetora.

Logo, para qualquer $n \in N$, f_n é bijetora.

□

Na tabela abaixo, apresentamos algumas funções f_n pertencentes à família de funções construída no lema anterior. Os elementos da partição $P(I_n)$ estão representados simbolicamente, e sua descrição encontra-se na tabela 5.2.1.1.

$P(I_0)$	J_1							
$f_0(J_i)$	1							
$P(I_1)$	J_1	J_2						
$f_1(J_i)$	1	2						
$P(I_2)$	J_1	J_2	J_3	J_4				
$f_2(J_i)$	1	3	2	4				
$P(I_3)$	J_1	J_2	J_3	J_4	J_5	J_6	J_7	J_8
$f_3(J_i)$	1	5	3	6	2	7	4	8

tabela 5.2.1.2_ *descrição das funções* $f_n, n = 0, 1, 2, 3$

Uma outra maneira de representar essas funções é através de uma árvore abstrata. O nível D dessa árvore representa a função f_D. O i-ésimo nó de um nível, partindo do nó mais à esquerda, possui o rótulo $f_D(J_i)$. Na figura abaixo, as funções f_0, f_1, f_2 e f_3 estão representadas na forma de árvore.

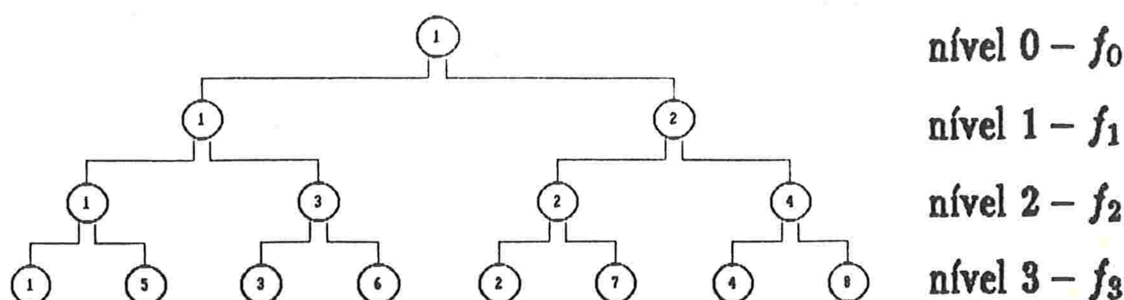

figura 5.2.1.1_ *representação em árvore das funções* $f_n, n = 0, 1, 2, 3$

Agora já estamos em condições de especificar a ordem das páginas a serem divididas durante o processo de expansão. Suponha que o arquivo possui nível atual D, isto é, as páginas sendo divididas são as de nível $D-1$, dando origem às páginas do nível D.

Se $D = 1$, não há dúvidas na escolha da página a ser dividida, pois no nível 0 o arquivo possui somente a página 1. A página, alocada durante a divisão da página 1, possui número 2.

Se $D > 1$ a i-ésima página a ser dividida neste nível, onde $1 \leq i \leq 2^{D-1}$, é aquela que possui número $f_{D-1}(J_i)$. A página alocada durante a divisão possui número $f_D(L_{2i}) = 2^{D-1} + i$, onde $J_i \in P(I_{D-1})$ *e* $L_{2i} \in P(I_D)$.

Exemplificando a ordenação das páginas divididas, considere as figuras 5.1.1 e 5.2.1.1

- a página 2 é a segunda página a ser dividida no nível atual 2, a terceira no nível atual 3 e a quinta no nível atual 4.

- a página 3 é a segunda página a ser dividida no nível atual 3, e a terceira no nível atual 4.

- a página 7 é a sexta página a ser dividida no nível atual 4.

Como a função f_{D-1} é bijetora, temos que todas as páginas que pertencem ao arquivo quando o seu nível é $D - 1$, são divididas uma única vez enquanto o arquivo está no nível D.

Algoritmo 5.2.1.1_ calcula o número da página a ser dividida durante uma expansão do arquivo

entrada :

D : nível atual do arquivo

P : P-ésima página a ser dividida no nível atual D

saída :

número da página a ser dividida $(f_{D-1}(J_P))$

comentário : o algoritmo assume que $P \leq 2^{D-1}$.

{ *se* $P = 1$
  então retorna (1)
  senão
    se P é par
      então retorna ($2^{D-2} + \frac{P}{2}$)
      senão
        executar o algoritmo 5.2.1.1 com as entradas $D - 1$ e $\frac{P+1}{2}$ }

Lema 5.2.1.4: *Corretude do algoritmo* 5.2.1.1.
Prova:

Será feita indução sobre o nível do arquivo (entrada D).

Se $D = 1$ e $P = 1$, o algoritmo 5.2.1.1 retorna o valor $1 = f_0(J_1)$. Portanto neste caso ele está correto.

Pela hipótese de indução, o algoritmo 5.2.1.1 retorna o valor $f_{D-1}(J_P)$, se as entradas são D e P, e $P \leq 2^{D-1}$.

Consideremos as entradas $D+1$ e P do algoritmo 5.2.1.1, tal que $P \leq 2^D$.

Se $P = 1$, o algoritmo retorna o valor $1 = f_D(J_1)$.

Se P é par, ele retorna $2^{D-1} + \frac{P}{2} = f_D(J_P)$.

Se P é ímpar, o algoritmo é chamado novamente com as entradas D e $\frac{P+1}{2}$. Como P é ímpar e $P \leq 2^D \Rightarrow \frac{P+1}{2} \leq 2^{D-1}$.

Por hipótese de indução, temos que o algoritmo retorna o valor $f_{D-1}(J_{\frac{P+1}{2}})$. Mas, por definição $f_{D-1}(J_{\frac{P+1}{2}}) = f_D(J_P)$.

Portanto o algoritmo 5.2.1.1 está correto.

□

Vamos estudar a seguir a complexidade do algoritmo 5.2.1.1. Como o algoritmo 5.2.1.1 requer uma quantidade fixa de espaço, a sua análise é feita sobre o tempo necessário para executá-lo.

Sejam D e P as entradas do algoritmo. Analisando a variação de P, temos que $P \geq 1$ sempre. Como $P \leq 2^{D-1} \Rightarrow D \geq 1$. Portanto, o algoritmo 5.2.1.1 poderá ser chamado no máximo D vezes. Mas, se $P = 2^{D-1} + 1$

temos que ele é chamado $D-1$ vezes. Portanto, no pior caso a complexidade do tempo é $O(\log_2 Q) = O(D)$.

Uma versão não recursiva do algoritmo 5.2.1.1 é dada a seguir. Segundo o trabalho de Vivian em [59], temos que a recursão do algoritmo 5.2.1.1 é uma "recursão iterativa"("tail recursion"), pois ao retornar da chamada do algoritmo é encerrada a sua execução. A prova de que o algoritmo abaixo é a versão não recursiva do algoritmo 5.2.1.1 encontra-se em [59].

Algoritmo 5.2.1.2_ calcula o número da página a ser dividida

entrada :
- D : nível atual do arquivo
- P : P-ésima página a ser dividida no nível atual D

saída :
- número da página a ser dividida $(f_{D-1}(J_P))$

```
{ L:
  se P = 1
    então retorna (1)
    senão
      se P é par
        então retorna ( 2^(D-1) + P/2 )
        senão {
          D ← D − 1;
          P ← (P+1)/2;
          vá para L; } }
```

5.2.2_ Algoritmo de expansão do arquivo

O algoritmo que executa a expansão de um arquivo organizado pelo Hashing linear decimal com distribuição não uniforme, usa o algoritmo 5.2.1.1 para calcular o número da página a ser dividida na expansão do arquivo. Lembre-se que o tratamento da colisão é feito pelo encadeamento das páginas de colisão.

Algoritmo 5.2.2_ executa a expansão de um arquivo organizado pelo HLD.

entrada :

D : nível do arquivo

Q : quantidade de páginas contidas no arquivo

comentários : dados D e Q, temos que $Q-2^{D-1}$ páginas foram divididas no nível atual D. Portanto, a página a ser dividida nesta expansão do arquivo possui número $f_{D-1}(J_{Q-2^{D-1}+1})$ se $Q < 2^D$. Caso contrário o seu número é $1 = f_{D-1}(J_{(Q-2^{D-1}+1) \bmod 2^{D-1}})$.
R e P são variáveis auxiliares usadas neste algoritmo.
O algoritmo que calcula o endereço do registro é dado na seção 5.3.
Note que $Q+1 = f_D(J_{2(Q-2^{D-1}+1)})$.

```
{ alocar a página Q + 1 no arquivo;
  R← Q - 2^(D-1) mod 2^(D-1);
  P ← f_(D-1)(J_(R+1)); /* P= número da página a ser dividida */
  ler página P e a sua região de colisão;
  para cada registro r lido faça {
    calcular o endereço de r considerando a página Q + 1;
    se o endereço de r mudou
      então
        escrever r na página Q + 1 ou na sua região de colisão
      senão
        escrever r na página P ou na região de colisão de P; }
  escrever página P no arquivo de dados;
  escrever região de colisão de P no arquivo de colisão;
  escrever página Q + 1 no arquivo de dados;
  escrever região de colisão de Q + 1 no arquivo de colisão;
  incrementar Q;
  se Q > 2^D
    então incrementar D; }
```

5.2.3_ Estrutura de dados para o algoritmo 5.2.2

Apesar do algoritmo 5.2.1.1 ser rápido, existe uma forma de saber, em tempo constante, qual é a próxima página a ser dividida.

Para obter esse tempo constante é construída uma lista de apontadores das páginas do arquivo. Cada página p do arquivo aponta para a

próxima página a ser dividida depois da sua divisão, e esse apontador é armazenado em um campo fixo da página. Na figura abaixo está representado um arquivo com 6 páginas e a lista dos apontadores. Em cada página, o quadrado do canto superior direito contém o apontador.

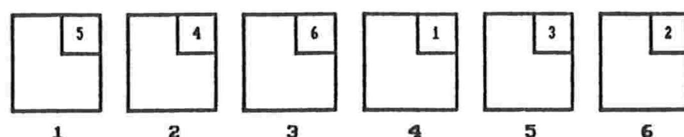

figura 5.2.3.1_ *arquivo contendo a lista dos apontadores*

Além dessa lista de apontadores é necessário um ponteiro, o qual denotamos por S. O ponteiro S aponta para a página a ser dividida na próxima expansão do arquivo. Após a divisão da página apontada por S, esse ponteiro precisa ser atualizado. Ou seja, ele deve apontar para a próxima página do arquivo a ser dividida, a qual é apontada pela página que sofreu a divisão. Note que para obter essa informação não é necessário fazer acesso ao disco, pois a página que sofreu a divisão encontra-se em 'mémória' interna. Dessa forma, elimina-se o cálculo do algoritmo 5.2.1.1 na expansão do arquivo. Considerando o arquivo ilustrado na figura anterior, temos que S aponta para a página 2. Na próxima expansão desse arquivo a página 2 é dividida, a página 7 é alocada, e o ponteiro S e a lista de apontadores são modificados. S apontará para a página 4, na listas dos apontadores a página 2 passará a apontar para a página 7 e esta para a página 4. Note que, a modificação feita na lista dos apontadores não envolve acessos extras ao discos, pois as páginas 2 e 7 são construídas durante a expansão. Essas alterações são mostradas na figura abaixo.

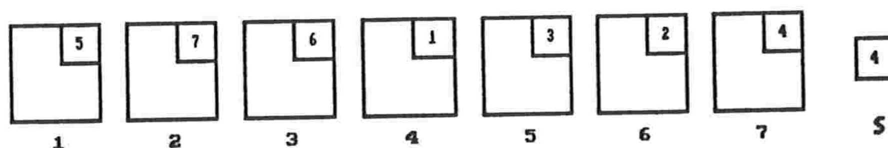

figura 5.2.3.2_ *arquivo contendo a lista dos apontadores e o ponteiro S*

Na figura abaixo é ilustrada uma outra forma de representar o arquivo da figura anterior, mostrando-se também todos os passos de subdivisão.

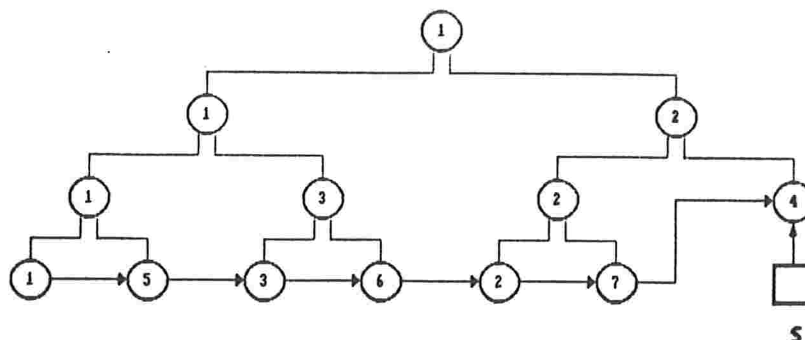

figura 5.2.3.3_ *representação em árvore de um arquivo contendo a lista dos apontadores e as subdivisões efetuadas*

5.2.4_ Contração do arquivo

Um arquivo pode sofrer uma série de eliminações e com isso o seu fator de ocupação diminuir drasticamente. Para não sub-utilizar o espaço em disco, a última página do arquivo é liberada. O critério usado para contrair o arquivo é a divisão controlada, ou seja, se eliminando a última página do arquivo o novo fator de ocupação é menor que o limite α, o arquivo é contraído.

Algoritmo 5.2.4_ executa a contração de um arquivo organizado pelo HLD

entrada :

D : nível do arquivo

Q : quantidade de páginas contidas no arquivo

comentário : a página de número Q a ser liberada pode conter registros. Estes devem retornar para a última página dividida, a página $f_{D-1}(J_{Q-2^{D-1}})$.

{ ler página Q e a sua região de colisão;
para cada registro r lido *faça*
escrever r na página $f_{D-1}(J_{Q-2^{D-1}})$ ou na sua região de colisão;
decrementar Q;
se $Q = 2^{D-1}$ *então*
decrementar D; }

Note que, não é necessário calcular a função f_D para contrair o arquivo se, for alterada a lista dos apontadores definida na seção 5.2.3, para conter ponteiros bidirecionais.

5.3_ Função de endereçamento

5.3.1_ Definição da função de endereçamento

Continuaremos denotando o nível do arquivo por D, e a sua quantidade de páginas por Q. Vamos denotar o conjunto das chaves dos registros por K. O conjunto $[0 .. 10^D - 1]$ é representado por I_D, $P(I_D)$ é a partição de I_D definida na seção 5.2.1, e f_D pertence à família de funções definidas no lema 5.2.1.3.

De uma maneira geral, um registro com chave k é endereçado da seguinte forma : através da função de hashing, a chave k é associada a um único inteiro l pertencente a I_D . A função de hashing é utilizada para distribuir uniformemente os registros pelo intervalo I_D. Como $P(I_D)$ é uma partição de I_D, existe um único $J_i \in P(I_D)$ tal que $l \in J_i$. O endereço do registro de chave k é a página $f_D(J_i)$ se esta página está alocada no arquivo; caso contrário o endereço é $f_{D-1}(L_{\lceil \frac{i}{2} \rceil})$ onde $L_{\lceil \frac{i}{2} \rceil} \in P(I_{D-1})$. Vamos caracterizar a página $f_D(J_i)$ que não pertence ao arquivo:

a página $f_D(J_i)$ não está alocada ao arquivo $\iff f_D(J_i) > Q$

$\underbrace{\iff}_{}$ i é par e $f_D(J_i) = 2^{D-1} + \frac{i}{2} > Q \iff$

definição de f_D

i é par, $i > 2(Q - 2^{D-1})$ e a página $f_{D-1}(L_{\frac{i}{2}})$ não foi dividida no nível D (definição da ordem das páginas divididas).

Após apresentar a idéia geral do algoritmo de endereçamento, vamos agora formalizá-lo.

O algoritmo de endereçamento utiliza :

i- Uma função T que converte a chave de um registro em um número natural. Esta função é necessária somente se a chave do registro não é um número natural.

$$T : K \to N$$

ii- Um gerador de números pseudo-aleatórios $RAND$, que possui um número natural como semente

$$RAND : N \to N$$

iii- A família de funções de hashing, G_n para $n \in N^*$, definidas abaixo. Cada função G_n mapeia o conjunto das chaves no sub-conjunto dos naturais I_n.

$$G_1 : K \rightarrow I_1$$
$$G_1(k) = RAND(T(K)) \bmod 10$$

para $n > 1$ temos que:

$$G_n : K \rightarrow I_n$$

$$G_n(k) = 10G_{n-1} + a_n \qquad \text{onde} \quad a_n = RAND^i(T(k)) \bmod 10$$

Tendo essas funções, o endereço de um registro com chave k é definido da seguinte forma:
Seja $J_i \in P(I_D)$ tal que $G_D(k) \in J_i$

$$e_D = \begin{cases} 1 & D = 0 \\ f_D(J_i) & D > 0 \; e \; i \leq 2(Q - 2^{D-1}) \\ f_{D-1}(J_{\lceil \frac{i}{2} \rceil}) & \text{caso contrário} \end{cases}$$

Note que a página $f_{D-1}(J_{\lceil \frac{i}{2} \rceil})$ já foi dividida no nível D, se $i \leq 2(Q - 2^{D-1})$.

Para exemplificar a função de endereçamento, considere o arquivo ilustrado na figura 5.2.3.3, e as tabelas 5.2.1.1 e 5.2.1.2. Este arquivo possui nível atual 3. Um registro de chave k possui endereço 1, se $0 \leq G_3(k) \leq 162$, pois neste caso temos que, $G_3(k) \in J_1$, $1 \leq 2(Q-4) = 6$ e $f_3(J_1) = 1$. Se $G_3(k) = 750$, temos que, $G_3(k) \in J_6$, como $6 \leq 2(Q-4) = 6$ e $f_3(J_6) = 7$, então o registro está armazenado na página 7. Mas se $820 \leq G_3(k) \leq 999$, o endereço do registro é a página 4, pois $G_3(k) \in J_7 \cup J_8$, $7 > 2(Q-4) = 6$, $\lceil \frac{7}{2} \rceil = \frac{8}{2} = 4$ e $f_2(J_4) = 4$.

Vamos verificar que a função de endereçamento está bem definida, ou seja, o endereço e_D associa qualquer registro a uma única página do arquivo.

O lema a seguir, relaciona os valores $G_n(k)$ e $G_{n-1}(k)$ com os conjuntos de $P(I_n)$ e $P(I_{n-1})$. Através dele, podemos analisar o comportamento dos registros do arquivo durante uma expansão. Com isso, identifica-se o momento em que um registro contido no arquivo pode mudar de endereço, e assim provar que a função de endereçamento está bem definida.

Lema 5.3.1.1: *Sejam* $n \in N^*$, $k \in K$, $J_i \in P(I_{n-1})$, $L_{2i-1} L_{2i} \in P(I_n)$.
$G_{n-1}(k) \in J_i \iff$ *ou* $G_n(k) \in L_{2i-1}$ *ou* $G_n(k) \in L_{2i}$
Prova:

Sejam $n \in N^*$, $k \in K$, $J_i \in P(I_{n-1})$ e L_{2i-1} , $L_{2i} \in P(I_n)$

Por construção temos que:

$G_n(k) = 10G_{n-1}(k) + a_n$ onde $a_n = RAND^n(T(k))$ mod 10

Sejam a_i,$b_i \in N$, tal que $J_i = [a_i .. b_i]$ então

$G_{n-1}(k) \in J_i \iff a_i \leq G_{n-1}(k) \leq b_i \iff$

$10a_i \leq 10G_{n-1}(k) + a_n \leq 10b_i + 9 \iff$

$10a_i \leq G_n(k) \leq 10b_i + 9 \iff$ ou $G_n(k) \in L_{2i-1}$ ou $G_n(k) \in L_{2i}$

□

Pelo lema 5.3.1.1 somente os registros de chave k tal que $G_{n-1}(k) \in J_i$ possuem $G_n(k) \in L_{2i-1}$ ou $G_n(k) \in L_{2i}$.

Portanto, somente os registros de chave k que possuem endereço $f_{n-1}(J_i)$, antes da divisão da página $f_{n-1}(J_i)$ no nível n, possuem endereço $f_n(L_{2i-1})$ ou $f_n(L_{2i})$ após a divisão dessa página. Logo, apenas os registros pertencentes à página dividida podem ter o seu endereço alterado durante uma expansão.

Pela definição da função de endereçamento, temos que o endereço de um registro pode mudar somente durante uma expansão. Com isso, podemos concluir que o endereço de um registro pode ser alterado somente quando a sua página original for dividida. Portanto, para provar que a função de endereçamento está bem definida basta provar que isso ocorre quando $Q = 2^D$.

Seja r um registro com chave k.

Como $P(I_D)$ é uma partição de I_D, e G_D é uma função $\Rightarrow$ existe um único $J_i \in P(I_D)$, tal que $G_D(k) \in J_i$.

Mas f_D é uma função $\Rightarrow$ existe uma única página $p = f_D(J_i)$ no arquivo, tal que endereço de r é igual a p.

Portanto, o endereço de r está bem definido quando o arquivo possui 2^D páginas.

O algoritmo de endereçamento deve calcular o valor $f_n(J_i)$ para $n \in N^*$ e $i \leq 2^n$. No entanto, sabemos que o algoritmo 5.2.1.1 retorna o valor $f_n(J_i)$ quando recebe os valores $n+1$ e i como entradas. Logo, para endereçar um registro com chave k, precisamos saber calcular o índice i tal que $G_D(k) \in J_i$ e $J_i \in P(I_D)$. A seguir iremos apresentar o algoritmo que calcula tal índice e depois o algoritmo de endereçamento .

5.3.2_ Cálculo do elemento de $P(I_D)$ que contém $G_D(k)$

Vamos apresentar aqui o algoritmo que calcula o conjunto de $P(I_D)$ que contém o número $G_D(k)$.

Pela definição da partição $P(I_D)$, temos que o cálculo do elemento de $P(I_D)$, que contém $G_D(k)$, é imediato quando se sabe o conjunto de $P(I_{D-1})$, que contém $G_{D-1}(k)$. Mas, o cálculo do conjunto de $P(I_{D-1})$ que contém $G_{D-1}(k)$, é imediato quando se sabe o elemento de $P(I_{D-2})$ que contém $G_{D-2}(k)$, e assim por diante. Quando $D = 1$ temos que $G_1(k) \in J_1 \iff 0 \leq G_1(k) \leq 5$. Portanto, para saber o índice do conjunto de $P(I_D)$ que contém $G_D(k)$ o algoritmo abaixo calcula, para $0 \leq i \leq D$, o valor $G_i(k)$ e os extremos do conjunto de $P(I_i)$ que contém $G_i(k)$.

Algoritmo 5.3.2_ calcula o índice i tal que $G_D(k) \in J_i$ e $J_i \in P(I_D)$.

entrada :

K : chave do registro
D : nível do arquivo

saída :

índice i tal que $G_D(k) \in J_i$ e $J_i \in P(I_D)$

comentários : I, F, G, P, nível, n são váriáveis usadas na descrição do algoritmo. A variável I contém o extremo inferior do intervalo que possui o valor $G_i(k)$, na profundidade i, e a F o extremo superior. A variável G mantém o valor $G_i(k)$, e a P o índice retornado na profundidade i

```
{ I ← 0;
  F ← 0;
  P ← 1;
  nível ← 1;
  n ← T(k);
  G ← 0;
  enquanto nível ≤ D faça {
    n ← RAND(n);
    G ← 10G + n mod 10;
    se G ≤ 5I + 5F + 5 + ⌊2^(nível −2)⌋
      então {
        F ← 5I + 5F + 5 + ⌊2^(nível−2)⌋;
```

```
        I ← 10I;
        P ← 2P − 1; }
      senão {
        I ← 5I + 5F + 6 + ⌊2^(nível−2)⌋;
        F ← 10F + 9;
        P ← 2P; }
    incrementar nível; }
  retorna ( P ); }
```

Lema 5.3.2.1: *Corretude do algoritmo* 5.3.2
Prova:

Para provar que o algoritmo 5.3.2 está correto, basta provar que após repetir l vezes, o seu *comando enquanto*, as suas variáveis contêm os seguintes valores:

$$\begin{aligned} \text{nível} &= l + 1 \\ G &= G_l(k) \\ n &= RAND^l(T(k)) \end{aligned} \tag{1}$$

e o conteúdo da variável P é tal que

$$J_P = [I \,..\, F],\ G_l(k) \in J_P \ \text{ e } \ J_P \in P(I_l)$$

Para provar que as afirmações acima estão corretas, vamos fazer indução sobre l.

Se $l = 0$ é imediato verificar que as afirmações feitas em (1) são verdadeiras.

Suponha que o *comando enquanto* foi repetido l vezes, $l > 0$. Por hipótese de indução, temos que após a $(l-1)$- ésima repetição do *comando enquanto* vale (1), ou seja
$\text{nível} = l$
$G = G_{l-1}(k)$
$n = RAND^{l-1}(T(k))$
e o conteúdo da variável P é tal que

$J_P = [I \,..\, F],\ G_{l-1}(k) \in J_P \ \text{ e } \ J_P \in P(I_{l-1})$

Portanto, com mais uma repetição do *comando enquanto* temos que:

nível $= l + 1$

$$n = RAND(n) = RAND(RAND^{l-1}(T(k)) = RAND^{l}(T(k))$$

$$G = 10G + n \bmod 10 = 10G_{l-1}(k) + RAND^{l}(T(k)) \bmod 10 = G_{l}(k)$$

Pelo lema 5.3.1.1 temos que

$$G_{l-1}(k) \in J_P \Rightarrow \text{ou } G_l(k) \in L_{2P-1} \text{ ou } G_l(k) \in J_{2P}$$

onde $L_{2P-1} = [10I \ .. \ 5I + 5F + 5 + \lfloor 2^{l-2} \rfloor]$
e $L_{2P} = [5I + 5F + 6 + \lfloor 2^{l-2} \rfloor \ .. \ 10F + 9]$

Portanto, o conteúdo da variável P é aquele que desejamos.

□

Vamos analisar a complexidade do algoritmo acima.

Com relação ao tempo necessário para executar o algoritmo 5.3.2 observa-se que, o *comando enquanto* existente nele é sempre repetido D vezes, onde D é o nível do arquivo. Portanto, no pior caso, no médio ou no melhor caso a complexidade de tempo do algoritmo 5.3.2 é $O(D) = O(\log_2 Q)$.

Considerando o espaço necessário para executar o algoritmo 5.3.2 temos que, o valor da função G_n cresce na potência 10, um dígito decimal é adicionado ao valor $G_n(k)$ a cada repetição do *comando enquanto*. O mesmo pode acontecer com o cálculo dos extremos do conjunto (variáveis I e F) que contém o valor G. Portanto, no pior caso a complexidade do espaço necessário para executar este algoritmo é $O(D) = O(\log_2 Q)$.

Denominaremos este método de Hashing linear decimal por que a função de hashing é calculada sobre a base decimal e na expansão do arquivo ele cresce por alocando 1 página nova.

5.3.3_ Algoritmo de endereçamento

O algoritmo de endereçamento é dado a seguir. Ele é imediato a partir do algoritmo 5.3.2 e do algoritmo 5.2.1.1.

Algoritmo 5.3.3_ calcula o endereço de um registro em um arquivo organizado pelo HLD.

entrada :

k : chave do registro
D : nível do arquivo
Q : quantidade de páginas contidas no arquivo

saída :

endereço do registro

{ $P \leftarrow$ chama algoritmo 5.3.2;
se $D = 0$ *então retorna* (1)
senão
se $P \leq 2(Q - 2^{D-1})$ *então retorna* ($f_D(J_P)$)
senão retorna ($f_{D-1}(J_{\lceil \frac{P}{2} \rceil})$); }

A complexidade, tanto do tempo como do espaço do algoritmo 5.3.3 é $O(D) = O(\log_2 Q)$, pois os algoritmos 5.3.2 e 5.2.1.1 possuem esta complexidade.

5.4_ Análise da distribuição dos registros pelo arquivo

Ao organizar um arquivo, contendo um número finito de páginas, pelo Hashing linear decimal com distribuição não uniforme, a distribuição dos registros pelas suas páginas não é uniforme. Isto é mostrado nessa seção. Prova-se também que, à medida que o arquivo vai crescendo a não uniformidade vai desaparecendo, tornando-se uniforme quando o arquivo possui infinitas páginas.

Continuaremos denotando o nível do arquivo por D e a quantidade de páginas contidas nele por Q. O conjunto $[0 \; .. \; 10^n - 1]$ é representado por I_n, $P(I_n)$ representa a partição de I_n definida na seção 5.2.1, e f_n é a função definida no lema 5.2.1.3.

Vamos denotar por Pr a probabilidade de endereçar um registro a uma página do arquivo. Pela definição da função de endereçamento, temos que a probabilidade de endereçar um registro à página p é:

$$(2) \quad Pr = \begin{cases} \frac{|f_D^{-1}(p)|}{10^D} & \text{se } p > 2^{D-1} \text{ ou } p \text{ foi dividida no nível } D \\ \frac{|f_{D-1}^{-1}(p)|}{10^{D-1}} & \text{caso contrário} \end{cases}$$

A probabilidade Pr está vinculada ao tamanho do conjunto, contido em $P(I_D)$ ou $P(I_{D-1})$, que está associado à página p pela função f_D ou f_{D-1}.

A seguir, vamos provar que a página escolhida para ser dividida durante o processo de expansão do arquivo, é aquela que esperamos estar mais cheia, ou seja, é aquela que possui a maior probabilidade de receber registros. Para chegar nesse resultado, primeiramente iremos, no lema 5.4.1, caracterizar a cardinalidade de cada elemento do conjunto $P(I_D)$. No lema 5.4.2 prova-se que os elementos de $P(I_D)$ possuem cardinalidade distintas que formam uma sequência decrescente. Nos lemas 5.4.3 e 5.4.4 é caracterizada a cardinalidade dos conjuntos $J_1, J_{2^D} \in P(I_D)$. Note que caracterizando a cardinalidade dos conjuntos pertencentes a $P(I_D)$, estamos caracterizando a probabilidade das páginas armazenarem um registros. No lema 5.4.5 são relacionadas as probabilidades de endereçar registros para as páginas que já foram divididas no nível corrente com aquelas que ainda não foram. Depois é mostrada a forma como é feita a distribuição dos registros pelas páginas do arquivo.

Lema 5.4.1: *Sejam $n \in N^*$, os sub-conjuntos I_n e I_{n+1}, e $P(I_n)$, $P(I_{n+1})$ as partições de I_n e I_{n+1} respectivamente. Considere $x_i = |J_i|$ para $J_i \in P(I_n)$ e $1 \leq i \leq 2^n$.*
Então para $1 \leq i \leq 2^n$

$$y_{2i-1} = 5x_i + \lfloor 2^{n-1} \rfloor + 1 \qquad onde \; y_{2i-1} = |L_{2i-1}| \; e \; L_{2i-1} \in P(I_{n+1})$$

$$y_{2i} = 5x_i - \lfloor 2^{n-1} \rfloor - 1 \qquad onde \; y_{2i} = |L_{2i}| \; e \; L_{2i} \in P(I_{n+1})$$

Prova:

Dados $J_i = [a_i \;..\; b_i]$ e $x_i = |J_i| = b_i - a_i + 1$

Por construção, temos que

$$L_{2i-1} = [10a_i \;..\; 5a_i + 5b_i + 5 + \lfloor 2^{n-1} \rfloor]$$
$$L_{2i} = [5a_i + 5b_i + 6 + \lfloor 2^{n-1} \rfloor \;..\; 10b_i + 9]$$

se $n = 0 \Rightarrow$

$$x_1 = 1 \;\; e \;\; y_1 = |L_1| = 6 = 5x_1 + 0 + 1 \;\; e \;\; y_2 = |L_2| = 4 = 5x_1 - 0 - 1$$

se $n > 0 \Rightarrow$

$$\begin{aligned} y_{2i-1} = |L_{2i-1}| &= 5a_i + 5b_i + 5 + 2^{n-1} - 10a_i + 1 \\ &= 5(b_i - a_i + 1) + 2^{n-1} + 1 \\ &= 5x_i + 2^{n-1} + 1 \end{aligned}$$

$$y_{2i} = |L_{2i}| = 10b_i + 9 - (5a_i + 5b_i + 6 + 2^{n-1}) + 1$$
$$= 5(b_i - a_i + 1) - 2^{n-1} - 1$$
$$= 5x_i - 2^{n-1} - 1$$

□

Lema 5.4.2: *Sejam $n \in N, I_n = [0 \;..\; 10^n - 1], P(I_n) = \{J_1, J_2, \ldots, J_{2^n}\}$ a partição de I_n e $x_i = |J_i|$ para $1 \leq i \leq 2^n$.*
Então $x_1, x_2, \ldots, x_{2^n}$ é uma sequência estritamente decrescente, ou seja, $x_1 > x_2 > \ldots > x_{2^n}$.
Prova:

Fazendo indução sobre n, onde $n \in N$, vamos provar que $x_1 > x_2 > \ldots > x_{2^n}$ e que $x_i - x_{i+1} \geq 2^{n-1}$, para $1 \leq i < 2^n$.

se $n = 1 \Rightarrow$
$x_1 = 6$ *e* $x_2 = 4 \Rightarrow x_1 > x_2$ e que $x_1 - x_2 = 2 \geq 2^{n-1}$.

Por hipótese de indução, temos que
$x_1 > x_2 > \ldots > x_{2^n}$ e $x_i - x_{i+1} \geq 2^{n-1}$ para $1 \leq i < 2^n$.

Seja $y_1, \ldots, y_{2^{n+1}}$ a sequência onde $y_i = |L_i|$ *e* $L_i \in P(I_{n+1})$.

Pelo lema 5.4.1 temos que para qualquer $1 \leq i \leq 2^n$
$y_{2i-1} = 5x_i + 2^{n-1} + 1$
$y_{2i} = 5x_i - 2^{n-1} - 1$

Portanto, $y_{2i-1} > y_{2i}$ *e* $y_{2i-1} - y_{2i} \geq 2^n$ para $1 \leq i \leq 2^n$.

Mas, para qualquer $i \leq 2^n - 1$

$$y_{2i} - y_{2i+1} = 5(x_i - x_{i+1}) - 2^n - 2$$
$$\geq 5.2^{n-1} - 2^n - 2$$
$$= 3.2^{n-1} - 2$$
$$\geq 2^n$$
$$> 0$$

Portanto, para qualquer $i \leq 2^n - 1$ temos $y_{2i-1} > y_{2i} > y_{2i+1} \Rightarrow$
$\Rightarrow y_1 > y_2 > \ldots > y_{2^{n+1}}$.

Logo, para qualquer $n \in N$, a sequência $x_1, \ldots, x_{2^n}$ é estritamente decrescente.

□

Seja $x_i = |J_i|$ onde $J_i \in P(I_D)$, para $1 \leq i \leq 2^D$. Como $J_i = f_D^{-1}(p)$ para algum $p \in [0 \ .. \ 2^D]$ e, por (2) temos que a probabilidade da página $p = f_D(J_i)$ receber registros é $Pr = \frac{x_i}{10^D}$ quando o arquivo possui 2^D páginas. Pelo lema 5.4.2, temos que a distribuição dos registros pelas páginas do arquivo é não uniforme quando ele contém 2^D páginas.

Lema 5.4.3: *Sejam* $n \in N, P(I_n) = \{J_1, \ldots, J_{2^n}\}$ *a partição de* I_n *e* $x_1 = |J_1|$.
Então, $x_1 = 6(5^{n-1}) + \sum_{i=0}^{n-2} 5^i 2^{n-2-i} + \sum_{i=0}^{n-2} 5^i$
Prova:

Vamos fazer indução sobre n, onde $n \in N^*$.
Se $n = 1$ então $x_1 = 6 = 6(5^{n-1}) + \sum_{i=0}^{n-2} 5^i 2^{n-2-i} + \sum_{i=0}^{n-2} 5^i$.
Pela hipótese de indução, temos que se $J_1 \in P(I_n)$ e $x_1 = |J_1|$ então $x_1 = 6(5^{n-1}) + \sum_{i=0}^{n-2} 5^i 2^{n-2-i} + \sum_{i=0}^{n-2} 5^i$.
Sejam $L_1 \in P(I_{n+1})$ *e* $y_1 = |L_1|$.
Pelo lema 5.4.1 temos que

$$\begin{aligned} y_1 &= 5x_1 + 2^{n-1} + 1 \\ &= 5(6(5^{n-1}) + \sum_{i=0}^{n-2} 5^i 2^{n-2-i} + \sum_{i=0}^{n-2} 5^i) + 2^{n-1} + 1 \\ &= 6(5^n) + \sum_{i=0}^{n-2} 5^{i+1} 2^{n-2-i} + \sum_{i=0}^{n-2} 5^{i+1} + 2^{n-1} + 1 \\ &= 6(5^n) + \sum_{i=0}^{n-1} 5^i 2^{n-1-i} + \sum_{i=0}^{n-1} 5^i \end{aligned}$$

Portanto, para qualquer $n \in N$,
$x_1 = 6(5^{n-1}) + \sum_{i=0}^{n-2} 5^i 2^{n-2-i} + \sum_{i=0}^{n-2} 5^i$, onde $x_1 = |J_1|$ e $J_1 \in P(I_n)$.
□

Lema 5.4.4: *Sejam* $n \in N^*, P(I_n) = \{J_1, \ldots, J_{2^n}\}$ *a partição de* I_n *e* $x_{2^n} = |J_{2^n}|$.
Então, $x_{2^n} = 4(5^{n-1}) - \sum_{i=0}^{n-2} 5^i 2^{n-2-i} - \sum_{i=0}^{n-2} 5^i$.
Prova:

Vamos fazer indução sobre $n, n \in N^*$.

Se $n = 1 \Rightarrow x_2 = 4 = 4(5^{n-1}) - \sum_{i=0}^{n-2} 5^i 2^{n-2-i} - \sum_{i=0}^{n-2} 5^i$.

Pela hipótese de indução, temos que se $n \geq 1, J_{2^n} \in P(I_n)$ e $x_{2^n} = |J_{2^n}|$ então $x_{2^n} = 4(5^{n-1}) - \sum_{i=0}^{n-2} 5^i 2^{n-2-i} - \sum_{i=0}^{n-2} 5^i$.

Sejam $L_{2^{n+1}} \in P(I_{n+1})$ *e* $y_{2^{n+1}} = |L_{2^{n+1}}|$.

Pelo lema 5.4.1, temos que

$$
\begin{aligned}
y_{2^{n+1}} &= 5x_{2^n} - 2^{n-1} - 1 \\
&= 4(5^n) - \sum_{i=0}^{n-2} 5^{i+1} 2^{n-2-i} - \sum_{i=0}^{n-2} 5^{i+1} - 2^{n-1} - 1 \\
&= 4(5^n) - \sum_{i=1}^{n-1} 5^i 2^{n-1-i} - \sum_{i=1}^{n-1} 5^i - 2^{n-1} - 1 \\
&= 4(5^n) - \sum_{i=0}^{n-1} 5^i 2^{n-1-i} - \sum_{i=0}^{n-1} 5^i
\end{aligned}
$$

Portanto, para qualquer $n \in N^*$, *e* $J_{2^n} \in P(I_n)$ *e* $y_{2^n} = |J_{2^n}| \Rightarrow y_{2^n} = 4(5^{n-1}) - \sum_{i=0}^{n-2} 5^i 2^{n-2-i} - \sum_{i=0}^{n-2} 5^i$.

□

Com os lemas 5.4.3 e 5.4.4 está caracterizada a probabilidade das páginas 1 e 2^D de receberem registros.

Lema 5.4.5: *Sejam* $n \in N, P(I_n) = \{J_1, \ldots, J_{2^n}\}$ *a partição de* I_n, $P(I_{n+1}) = \{L_1, \ldots, L_{2^{n+1}}\}$ *a partição de* $I_{n+1}, x_i = |J_i|$ *e* $y_l = |L_l|$. *Então para quaisquer* $i, l \in N$, *onde* $1 \leq i \leq 2^n$ *e* $1 \leq l \leq 2^{n+1}$ *tem-se que* $10x_i > y_l$

Prova:

Pela lema 5.4.2 temos que

$x_1 > x_2 > \ldots > x_{2^n}$

$y_1 > y_2 > \ldots > y_{2^{n+1}}$

portanto, basta provar que $10x_{2^n} > y_1$.

Pelo lema 5.4.4, temos que

$$10x_{2^n} = 8(5^n) - \sum_{i=1}^{n-1} 5^i 2^{n-i} - 2\sum_{i=1}^{n-1} 5^i$$

e pelo lema 5.4.3, temos que

$$y_1 = 6(5^n) + \sum_{i=0}^{n-1} 5^i 2^{n-1-i} + \sum_{i=0}^{n-1} 5^i$$

Portanto,

$$\begin{aligned} 10x_{2^n} - y_1 &= 2(5^n) - \sum_{i=1}^{n-1}(2^{n-i} + 2^{n-1-i} + 3)5^i - 2^{n-1} - 1 \\ &= 5^n - \sum_{i=1}^{n-2}(2^{n-i} + 2^{n-1-i} + 3)5^i - 2^{n-1} - 1 - 5^{n-1} \end{aligned}$$

Por indução verifica-se que, para qualquer $u \in N$ tal que $u \geq 2 \Rightarrow 2^u + 2^{u-1} + 3 < 2(5^{\frac{u}{2}})$

Portanto,

$$\begin{aligned} 10x_{2^n} - y_1 &> 5^n - \sum_{i=1}^{n-2} 2(5^{\frac{n-i}{2}})5^i - 2^{n-1} - 1 - 5^{n-1} \\ &> 5^n - \sum_{i=1}^{n-2} 2(5^{\frac{n-i}{2}})5^i - 5^{\frac{n-1}{2}} - 1 - 5^{n-1} \\ &= 5^n - (3(5^{n-1}) + 2(5^{\frac{2n-3}{2}}) + \ldots + 2(5^{\frac{n+1}{2}}) + 5^{\frac{n-1}{2}} + 1) \\ &> 5^n - (3(5^{n-1}) + 5^{\frac{1}{2}}\, 5^{\frac{2n-3}{2}} + \ldots + 5^{\frac{1}{2}}\, 5^{\frac{n+1}{2}} + 5^{\frac{n-1}{2}} + 1) \\ &= 5^n - (4(5^{n-1}) + 3(5^{n-2}) + 3(5^{n-3}) + \ldots + 5^{\frac{n-1}{2}} + 1) \\ &> 0 \end{aligned}$$

Logo, para qualquer $i, 1 \leq i \leq 2^n$ e para qualquer $l, 1 \leq l \leq 2^{n+1}$

$$10x_i > y_l$$

□

Com o lema 5.4.5 conclue-se que, as páginas divididas em um certo nível possuem probabilidade de receber registros menor do que aquelas que ainda não foram divididas naquele nível.

Teorema 5.4.1: *Considere um arquivo com Q páginas e nível D. A página a ser dividida na próxima expansão é aquela que possui a maior probabilidade de receber registros.*

Prova:

Se o arquivo possui Q páginas no nível D, a página a ser dividida na próxima expansão do arquivo possui número
$p = f_{D-1}(J_{(Q-2^{D-1}+1) \bmod 2^{D-1}})$ onde $J_i \in P(I_{D-1})$ para $1 \le i \le 2^{D-1}$.

Vamos denotar $x_i = |J_i|$ para $1 \le i \le 2^{D-1}$ e $y_l = |L_l|$ onde $L_l \in P(I_D)$ e $1 \le l \le 2^D$.

As páginas que já foram divididas no nível D e aquelas que foram alocadas durante essas divisões possuem probabilidade

$$\frac{y_1}{10^D}, \ldots, \frac{y_{2(Q-2^{D-1})}}{10^D}$$

de armazenar registros.

As páginas que não foram divididas no nível D possuem probabilidade

$$\frac{x_{((Q-2^{D-1}+1) \bmod 2^{D-1})}}{10^{D-1}}, \ldots, \frac{x_{2^{D-1}}}{10^{D-1}}$$

de armazenar registros.

Pelo lema 5.4.2, a página $p = f_{D-1}(J_{(Q-2^{D-1}+1) \bmod 2^{D-1}})$ é a que possui a maior probabilidade de armazenar registros entre as páginas que não foram divididas no nível D. E, pelo lema 5.4.5 temos que qualquer página não dividida no nível D, possui maior probabilidade de armazenar registros do que aquelas que já foram divididas nesse nível ou as que foram alocadas durante essas divisões. Portanto, a página p é a que possui a maior probabilidade de estar cheia.

□

Teorema 5.4.2: *O Hashing linear decimal com distribuição não uniforme faz a distribuição não uniforme dos registros pelas páginas do arquivo, quando ele possui um número finito de páginas.*

Prova:

Sejam D o nível do arquivo, Q a sua quantidade de páginas, $P(I_D)$ e $P(I_{D-1})$ as partições de I_D *e* I_{D-1} respectivamente, e $x_i = |J_i|$ para $J_i \in P(I_{D-1})$ e $y_l = |L_l|$ para $L_l \in P(I_D)$.

Temos que as páginas que foram divididas no nível D possuem número $f_{D-1}(J_1), \ldots, f_{D-1}(J_{Q-2^{D-1}})$.

Nas divisões das páginas acima foram alocadas as de número $2^{D-1}+1,\ 2^{D-1}+2,\ \ldots,\ Q$.

Por (2), temos que a probabilidade de endereçar registros a essas páginas é

$$\frac{y_1}{10^D}, \ldots, \frac{y_{2(Q-2^{D-1})}}{10^D}$$

As páginas que não foram divididas no nível D possuem número $f_{D-1}(J_{Q-2^{D-1}+1}), \ldots, f_{D-1}(J_{2^{D-1}})$.

Elas possuem as probabilidades de receber registros

$$\frac{x_{Q-2^{D-1}+1}}{10^{D-1}}, \ldots, \frac{x_{2^{D-1}}}{10^{D-1}}$$

Pelos lemas 5.4.5 e 5.4.2, as probabilidades das páginas contidas no arquivo são distintas. Portanto, temos que a distribuição dos registros pelas páginas do arquivo não é uniforme quando ele possui finitas páginas. □

Vamos verificar que, à medida que o arquivo cresce a tendência é ocorrer a uniformidade da distribuição dos registros pelas páginas do arquivo, ou seja, quando o arquivo possui infinitas páginas a distribuição dos seus registros é uniforme.

Teorema 5.4.3: *O Hashing linear decimal com distribuição não uniforme faz a distribuição uniforme dos registros pelo arquivo quando ele possui infinitas páginas.*

Prova:

Seja p_q^n = a probabilidade da página q de armazenar registros quando o arquivo possui 2^n páginas.

Pelo teorema 5.4.2,

$$p_1^1 > p_1^2 > p_1^3 > \ldots > p_1^n > \ldots$$

Como $p_1^n \geq 0$ para qualquer $n \in N$ temos que a sequência $p_1^1, p_1^2, \ldots$ é estritamente descrescente e limitada $\Rightarrow$ a sequência $p_1^1, p_1^2, \ldots$ é convergente. Portanto, existe $p_1 \in R$ tal que

$$p_1^1, p_1^2, \ldots \rightarrow p_1$$

Da mesma forma, prova-se que para quaisquer $i, l \in N$ tal que $i \leq 2^l$ existe $p_i \in R$ tal que

$$p_i^l, p_i^{l+1}, \ldots \rightarrow p_i$$

A distribuição dos registros pelas páginas do arquivo é uniforme $\iff p_1 = p_2 = p_3 = \ldots$

Pelo lema 5.4.5,

$$(3) \qquad p^1_{f_1(J_1)}, p^1_{f_1(J_2)}, p^2_{f_2(J_1)}, p^2_{f_2(J_2)}, \ldots, p^n_{f_n(J_i)}, \ldots$$

é uma sequência estritamente decrescente.

Como $p^i_{f_i(J_n)} \geq 0$, para qualquer $n, i \in N$, e $n \leq 2^i \Rightarrow$ a sequência (3) é convergente $\Rightarrow$ existe $p \in R$ tal que

$$p^1_{f_1(J_1)}, p^1_{f_1(J_2)}, p^2_{f_2(J_1)}, p^2_{f_2(J_2)}, \ldots, p^n_{f_n(J_i)}, \ldots \rightarrow p$$

Para quaisquer $i, l \in N$ onde $i \leq 2^l$ temos que $p_i^l, p_i^{l+1}, \ldots$ é uma sub-sequência da sequência (3) $\Rightarrow p_i = p$ para qualquer $i \in N$.

□

Com o teorema 5.4.3 podemos concluir que todas as páginas do arquivo, quando ele possui infinitas páginas, tendem a conter αb registros, onde α é o fator de ocupação limite do arquivo e b é a capacidade de cada página do arquivo.

5.5_ Modificações sugeridas para o Hashing linear decimal com distribuição não uniforme

Vimos que a função que calcula o endereço de um registro no Hashing linear decimal com distribuição não uniforme (HLD), necessita de espaço não constante para ser executado. Com isso, precisamos de rotinas especiais para manipular os cálculos, pois à medida que o arquivo cresce os números envolvidos no cálculo de um endereço vão também crescendo. Apesar da simplicidade da implementação dessas rotinas, propomos mudar a função de endereçamento, de forma a tornar esses cálculos mais simples.

No HLD os cálculos de endereço são feitos sobre a base decimal. Se trocarmos para a base octal (8), os cálculos dos grandes números envolvidos na execução do algoritmo de endereçamento resume-se a 'shifts ', ganhando-se com isso, maior rapidez no cálculo do endereço. O método resultante dessa modificação chamar-se-á Hashing linear octal com distribuição não uniforme (HLO).

Para fazer a troca dos dígitos decimais pelos dígitos octais, faz-se uma formalização sobre o conjunto $A_n = [0 \;..\; 2^{3n} - 1]$ igual àquela feita sobre o conjunto $I_n = [0 \;..\; 10^n - 1]$. Ou seja, define-se uma partição $F(A_n)$ de A_n igual àquela feita sobre I_n.

A partição $F(A_n)$, onde $n \geq 2$, é construída indutivamente da seguinte forma:

i- $F(A_2) = \{B_1, B_2, B_3, B_4\}$ onde
$B_1 = [0 \;..\; 18]$ $B_2 = [19 \;..\; 35]$ $B_3 = [36 \;..\; 50]$ $B_4 = [51 \;..\; 63]$

ii- Dado $F(A_n) = \{B_1, \ldots, B_{2^n}\}$ onde $B_i = [a_i \;..\; b_i]$ para $1 \leq i \leq 2^n$, construímos $F(A_{n+1}) = \{C_1, \ldots, C_{2^{n+1}}\}$ onde
$C_{2i-1} = [8a_i \;..\; 4a_i + 4b_i + 2^{n-1} + 3]$
$C_{2i} = [4a_i + 4b_i + 4 + 2^{n-1} \;..\; 8b_i + 7]$

Verifica-se por indução, que se $n \in N$ onde $n \geq 2$ e $B_i, B_{i+1} \in F(A_n)$ onde $1 \leq i < 2^n$ então $|B_i| - |B_{i+1}| = 2^{n-1}$. Com isso, temos que a cardinalidade dos conjuntos de $F(A_n)$ diminui em uma taxa constante, ou seja, as probabilidades de receber registros das páginas do arquivo formam uma sequência estritamente decrescente com diferença constante. As provas de que $F(A_n)$ é uma partição e que os elementos de $F(A_n)$ possuem uma ordem bem definida, são omitidas aqui, pois são equivalentes às dos lemas 5.2.1.1 e 5.2.1.2.

A família de funções $f_n, n \in N$, definidas no lema 5.2.1.3, são substituídas na função de endereçamento do HLO, pelas funções definidas abaixo:

$$h_i : F(A_n) \to [1 \;..\; 2^i]$$
$$h_i(B_j) = f_i(J_j)$$

onde $J_j \in P(I_i)$, $1 \leq j \leq 2^i$ e $i \in N$, $i \geq 4$.

A família de funções $G_n, n \in N$, definidas na função de endereçamento do HLD, são substituídas também. No HLO usa-se as funções definidas abaixo:

$g_2 : K \rightarrow A_2$

$g_2(k) = 8(RAND(T(k)) \bmod 8) + (RAND^2(T(K)) \bmod 8)$

$g_n : K \rightarrow A_n$

$g_n(k) = 8g_{n-1} + (RAND^n(T(k)) \bmod 8)$

As funções $RAND$ e T são aquelas definidas na seção 5.3.1.

O endereço do registro com chave k num arquivo que possui nível D é definido, no HLO, da seguinte forma:
seja $B_i \in F(I_D)$, tal que $g_D(k) \in B_i$

$$e_D = \begin{cases} h_D(B_i) & i \leq 2(Q - 2^{D-1}) \\ h_{D-1}(B_{\lceil \frac{i}{2} \rceil}) & \text{caso contrário} \end{cases}$$

REFERÊNCIAS BIBLIOGRÁFICAS

Abreviaturas:
TODS - Transactions on Database Systems
CACM - Communication of the ACM
VLDB - International Conference on Very Large Database

/1 / Bechtold U., Küspert K.,**On the use of Extendible Hashing without Hashing**, *Information Processing Letters*, 19 (1984), 21–26.

/2 / Bell D.A., Deen S.M.,**Hash Trees versus B-TREES**, *The Computer Journal*, 27, 3 (1984), 218 – 223.

/3 / Beel D.A., Deen S.M.,**Key space compression and Hashing in Preci**, *The computer Journal*, 25, 4 (1982), 486 – 492.

/4 / Brunk H.D.,*An introduction to Mathematical Statistics*, Blaisdell Publishing Company, New York, 1965.

/5 / Burkhard W.A.,**Partial-Match Hash Coding: benefits of redundancy**, *TODS*, 4, 2 (1979), 228 – 239.

/6 / Deen S.M.,**An implementation of Impure Surrogates**, Proc. 8th VLDB, México-city, 1982, 245 – 256.

/7 / Carter J.L., Wegman,**Universal classes of hash functions**, *J. Comput. Systems Sci.*, 18 (1979), 143 – 154.

/8 / Coffman E.G., Eve J.,**File structures using hashing functions**, *CACM*, 13, 7 (1970), 427 – 436.

/9 / Ellis C.S.,**Concurrency in Linear Hashing**, *TODS*, 12, 2 (1987).

/10 / Fagin,**Extendible Hashing - a fast access method for dynamic files**, *TODS*, 4, 3 (1979), 315 – 344.

/11 / Flajolet P.,On the performance evaluation of Extendible Hashing and Trie Searching, *Acta Informatica*, 20 (1983), 345 – 369.

/12 / Knott G.D.,Expandable open addressing hash table storage and retrieval, Proc. ACM SIGFIDET WORKSHOP on Data Desc., access and control, ACM, 1971, 187 – 206.

/13 / Knuth D.E.,*The Art of Computer Programming*, Vol. 3, Sorting and Searching, Addison-Wesley, Reading, Mass., 1973.

/14 / Knuth D.E.,*The Art of Computer Programming*, Vol. 2, Seminumerical Algorithms, Addison-Wesley, Reading, Mass., 1981.

/15 / Lam C. Y., Madnik S.E.,Properties of Storage Hierarchy Systems with multiple page sizes and redundant data, *TODS*, 4, 3 (1979), 345 – 367.

/16 / Larson P.Å.,Frequency loading and linear probing, *Bit*, 1, 9 (1979), 223 – 228.

/17 / Larson P.Å.,Analysis of Repetead Hashing, *Bit*, 20, (1980), 25 – 32.

/18 / Larson P.Å.,Dynamic Hashing, *Bit*, 18, 2 (1978), 184 – 201.

/19 / Larson P.Å.,Linear Hashing With Partial Expansions, Proc. 6th VLDB, Montreal, 1980, 224 – 232.

/20 / Larson P.Å.,Performance analysis of Linear Hashing With Partial Expansions, *TODS*, 7, 4 (1982), 566 – 587.

/21 / Larson P.Å.,A Single-file Version of Linear Hashing With Partial Expansions, Proc. 8th VLDB, Mexico-city, 1982, 300 – 309.

/22 / Larson P.Å.,Expected worst-case performance of hash files, *The Computer Journal*, 25, 3 (1982), 347 – 351.

/23 / Larson P., Gonnet G.H.,**External hashing with limited internal storage**, Proc. ACM Symposium on Principles of Database Sys., Los Angeles, 1982, 256 – 261.

/24 / Larson P.Å.,**Dynamische Hashverfahren**, *Informatik- Spektrum*, 6 (1983), 7 – 19.

/25 / Larson P.Å., Kaja A.,**File organization : implementation of a method guaranteeing retrieval in one access**, *CACM*, 27, 7 (1984), 670 – 677.

/26 / Larson P.Å., Ramakrishna M.V.,**External Perfect Hashing**, Proc. ACM- SIGMOD Conf., New York, 1985, 190 – 200.

/27 / Larson P.Å.,**Linear Hashing with Overflow-Handling by Linear Probing**, *TODS*, 10, 1 (1985), 75 – 89.

/28 / Lindgren B.W.,*Basic ideas of Statistics*, Macmiillan Publishing Co., New York, 1975.

/29 / Litwin W.,**Hachage Virtual: Une Nouvelle Technique D'Addressage de Memoires**, These de Doctorat d'Etat, Univ. de Paris, 1979.

/30 / Litwin W.,**Virtual Hashing: a dynamically changing hashing**, Proc. 4th VLDB, Berlin, 1978, 517 – 523.

/31 / Litwin W.,**Linear Hashing: a new tool for files and table addressing**, Proc. 6th VLDB, Montreal, 1980, 212 – 223.

/32 / Litwin W.,**Linear Virtual Hashing: a new tool for files and tables implementation**, Res. Rep. MAP-I-021, I.R.I.A., 1979, 24.

/33 / Litwin W.,**Trie Hashing**, Proc. ACM-SIGMOD Conf., 1981, 19 – 29.

/34 / Lomet D.B.,**Bounded Index Exponencial Hashing**, *TODS*, 8, 1 (1983), 136 – 165.

/35 / Lomet D.B.,**A high performance, universal, Key Associative Access Method**, Proc. ACM SIGMOD Conf., San Jose, 1983, 120 – 122.

/36 / Nivergelt J., Reingolg E.M.,**Binary Search of Trees of Bounded Balance**, SIAM J. Comptng, 2 (1973), 33 – 43.

/37 / Markowsky G., Carter J.L., Wegman M.,**Analysis of a Universal Class of Hash Functions**, *Lecture Notes in Computer Science*, 64 (1978), 345 – 354.

/38 / Martin G.N.N.,**Spiral Storage: incrementally argmentable hash addressed storage**, Theory of Computation Report, 27 (1979).

/39 / Mendelson H.,**Analysis of Extendible Hashing**, *IEEE Trans. on Soft. Eng.*, 8, 6 (1982), 611 – 619.

/40 / Mullin J.K.,**Tightly controlled Linear Hashing without separate overflow storage**, *Bit*, 21 (1981), 390 – 400.

/41 / Otoo E.J.,**A mapping function for the directory of a Multidimensional Extendible Hashing**, Proc. 10th VLDB, Singapore, 1984, 493 – 506.

/42 / Otoo E.J.,**Balanced Multidimensional Extendible Hash Tree**, Proc. ACM- SIGMOD Conf., 1986, 100 – 113.

/43 / Ouksel M.,Scheuermann P.,**Storage mapping for Multidimensional Linear Dynamic Hashing**, Proc. ACM SIGACT-SIGMOD Symp. on Principles of Database Sys., Portland, Oregon, 1985, 20 – 27.

/44 / Ouksel M.,The interpolation-based Grid File, ACM SIGMOD Symp. on Principles of Dat. Sys., Portland, Oregon, 1985, 20 – 27.

/45 / Quittner P.,Efficient combination of Index Tables and Hashing, *Software -Practice and Experience*, 13 (1983), 471 – 478.

/46 / Ramamohanarao K., Lloyd J.W.,Dynamic Hashing Schemes, *The Computer Journal*, 25, 4 (1982), 478 – 485.

/47 / Ramamohanarao K.,Lloyd J.W.,Partial-Match retrieval for Dynamic Files, *Bit*, 22, 2 (1982), 150 – 168.

/48 / Ramamohanarao K., Lloyd J.W., Thom,Partial-Match retrieval using Hashing and Descriptors, *TODS*, 8, 4 (1983), 552 – 576.

/49 / Ramamohanarao K., Sacks D.R.,Recursive Linear Hashing, *TODS*, 9, 3 (1984), 369 – 391.

/50 / Ramamohanarao K., Sacks D.R.,Partial-match retrieval using Recursive Linear Hashing, *Bit*, 25, 3 (1985), 477 – 484.

/51 / Sarwate D.V.,A note on universal classes of hash functions, *Information Processing Letters*, 10, 1 (1980), 41 – 45.

/52 / Scholl M.,New file organization based on Dynamic Hashing, *TODS*, 6, 1 (1981), 194 – 211.

/53 / Silberschatz A.,Principles of Database Systems, Proc. 5th ACM SIGAGT-SIGMOD Symposium, Cambridge, 1985, 24 – 26.

/54 / Sussenguth E.H.Jr.,Use of the structures for processing files, *CACM*, 26, 1 (1983), 17 – 20.

/55 / Tamminen M.,Order preserving Extendible Hashing and Bucket Tries, *Bit*, 21, 4 (1981), 419 – 435.

/56 / Tamminen M.,The Extendible cell method for Closest Point Problems, *Bit*, 22, 1 (1982), 27 – 41.

/57 / Tamminen M.,Extendible Hashing with overflow, *Information Processing Letters*, 15, 5 (1982), 227 – 232.

/58 / Torn A.A.,Hashing with Overflow Indexing, *Bit*, 24, 3 (1984), 317 – 332.

/59 / Vivian M.E.,Sobre eliminação de recursão em programas, Tese de mestrado, USP, 1979.

/60 / Yang W.P., Du M.W.,A Dynamic Perfect Hash Function defined by an Extended Hash Indicator Table, Proc. 10th VLDB, Singapore, 1984, 245 – 254.

/61 / Yao A. C.,A note on the analysis of Extendible Hashing, *Inf. Proc. Letters*, 11, 3 (1980), 224 – 232.

/62 / Yao A.C.,O Random 2 – 3 Trees, *Acta informatica*, 9 (1978), 159 – 170.

/63 / Bayer R., Mccreight,Organization and maintenance of Large Ordered Indexes, *Acta informatica*, 1 (1972), 173 – 189.