A L G O R I T M O S D E H A S H I N G

P A R A P R O B L E M A S E S P E C Í F I C O S

Jano Moreira de Souza

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS (M.Sc.)

Aprovada por:

RIO DE JANEIRO

ESTADO DO RIO DE JANEIRO - BRASIL

FEVEREIRO DE 1978

DE SOUZA, JANO MOREIRA

ALGORITMOS DE HASHING PARA PROBLEMAS ESPECÍFICOS (RIO DE JANEIRO) 1978.

V, 138p. 29,7cm (COPPE-UFRJ, M.Sc., Engenharia de Sistemas e Computação, 1978)

Tese - Univ. Fed. Rio de Janeiro - Fac. Engenharia

1. Banco de Dados I. COPPE/UFRJ II. Algoritmos de Hashing para Problemas Específicos.

Ã MARIA GRACINDA e

AOS MEUS AMIGOS

AGRADECIMENTOS

Ao professor Estevam Gilberto de Simone pela orientação e incentivo; ao professor João Lizardo de Araújo pelo apoio, orientação e pelo crédito na escolha do tema; aos professores Antonio Alberto F. de Oliveira, Dina Feigenbaum Cleiman e Nelson Maculan Filho pelo incentivo; aos alunos do Programa, turma de 1976 que através do questionamento dos métodos existentes, motivaram o presente trabalho.

RESUMO

É feita inicialmente uma exposição extensiva dos métodos existentes para transformação de chaves em endereços e para resolução das colisões, apresentando os algoritmos principais. Os capítulos II e III foram redigidos visando sua utilização como texto de cursos de pós-graduação sobre teoria e métodos de "hashing".

Em seguida, partindo do pressuposto que melhor eficiência seria alcançada considerando-se as características desejáveis de algoritmos para manuseio de arquivos sob um ponto de vista prático de projeto, efetuou-se o levantamento das principais propriedades que definem o uso de arquivos e verificou-se até onde os algoritmos conhecidos satisfazem cada uma dessas características e suas combinações principais.

Com tal diagnóstico foram desenvolvidos 14 novos algoritmos que atendem de forma eficiente aos problemas especificados, excetuando os requisitos de ordenação e alocação dinâmica de memória que não foram estudados. Alguns dos novos algoritmos permitiram inclusive, soluções mais eficientes para problemas considerados resolvidos no diagnóstico.

Todos os algoritmos são apresentados de forma padronizada e para os novos foram efetuadas intensas simulações para determinação de seus comportamentos.

ABSTRACT

This thesis presents initially a survey of the existing methods for key-to-address transformations and for the treatment of collisions, showing the more important algorithms. Chapters II and III were written havin in mind their possible utilization as a text for graduate courses in hashing methods and theory.

We next analyze the algorithms under consideration, from the point of view of desirable properties for the actual design of file handling systems, for specific applications.

After that analysis, 14 new algorithms are presented, which efficiently solve the proposed problems, except possibly for the sorting and dynamic allocation requisites, which were not considered.

In some cases, for problems already solved by algorithms found in the literature we show that some of the new algorithms are more efficient than the best ones previously known.

All the algorithms are presented in a standard form, for ease of comparison. All of them were extensively simulated, and the results of the simulation shown.

ÍNDICE

I. INTRODUÇÃO

A partir do estudo e ensino dos algoritmos de Busca em Arquivo e de alguns problemas práticos enfretados por colegas na estruturação de tabelas de símbolos e mnemônicos em compiladores e montadores ora em desenvolvimento no Programa de Engenharia de Sistemas e Computação, nos veio a consciência de que, para uma série de problemas práticos os algoritmos existentes eram ineficientes, ou mesmo inaplicáveis.

O trabalho surgiu de um problema real de duas pesquisas sendo desenvolvidas no programa, um montador e simulador MIX "time-sharing" e um compilador de uma linguagem tipo PL; onde o melhor método conhecido para a organização das tabelas de símbolos e mnemônicos não levava em conta uma informação importante a respeito das chaves: sua probabilidade de referência. Da utilização desta informação surgiu o primeiro algoritmo, que se mostrou mais eficiente do que o antigo, tanto com dados com distribuição teórica de probabilidades como com dados reais mostrados em muitos programas escritos em "assembler" do computador MITRA-15.

Da resolução do primeiro problema, pudemos concluir que os algoritmos conhecidos eram de tal forma gerais que desprezavam informações muitas vezes disponíveis, ou estimáveis nas aplicações específicas.

Esse enfoque "geral" dos algoritmos pode ser atribuido, pelo menos em parte, ao fato de que a maioria da pesquisa na área se originou nas equipes dos fabricantes de

equipamento, que por filosofia de comercialização, tentam generalizar ao máximo os seus produtos, mesmo às custas de grande ineficiência na maioria das aplicações.

Considerando o alto custo das máquinas no Brasil, e o fato dessas serem importadas, custando divisas preciosas ao país, pareceu-nos de extrema importancia que se fizesse um estudo mais detalhado dos problemas existentes, observando quais das suas características são determinantes no tipo de algoritmo a empregar, e, combinando-se estas características, que tipos de problemas se delineavam. Desta forma descobriu-se, pelo caminho inverso, que certas aplicações não tinham solução eficiente conhecida e usavam métodos totalmente diferentes daqueles que gostaríamos de utilizar por razões de eficiência, simplesmente porque os métodos tinham restrições a esta ou àquela característica.

Algumas características que podem determinar o método de busca a ser utilizado, são as seguintes:

- Baixo tempo médio para buscas com sucesso.
- Baixo tempo médio para buscas sem sucesso.
- Baixo tempo médio por inserção.
- Arquivo estático em termos de inserção.
- Arquivo estático em termos de remoção.
- Arquivo dinâmico em termos de inserção.
- Arquivo dinâmico em termos de remoção.
- Localização das chaves em memória interna.
- Localização das chaves em memória externa ou virtual.

- Probabilidades diferentes e conhecidas "a priori".
- Probabilidades diferentes mas variáveis com o tempo.
- Tempo de resposta limitado (pior caso garantido e conhecido).
- Arquivo ordenado.
- Alocação dinâmica de espaço.
- Simplicidade de programação.
- Grande aproveitamento de espaço.

Combinando essas características, obteríamos uma grande quantidade de problemas, que exigiriam, dentro do nosso enfoque uma grande quantidade de algoritmos específicos. Selecionamos alguns problemas que consideramos os mais importantes, procuramos determinar quais já dispunham de solução eficiente e quais não; dentre esses selecionamos aqueles que se identificavam com problemas reais e buscamos soluções. Neste trabalho são apresentadas soluções para diversos desses problemas e que, afortunadamente também resolvem alguns problemas já considerados com solução satisfatória, de uma forma melhor do que estas.

Todas soluções são calcadas sobre um método de transformação de chaves ("hashing"), por ser o método que apresenta melhores possibilidades teóricas, já que outros métodos geralmente utilizam estruturas de árvore cujo limite teórico inferior é da ordem $\log m$, onde m é o número de elementos na tabela.

Anexo a esta introdução, encontra-se uma lista de todos os algoritmos referenciados nesta tese.

ANEXO À INTRODUÇÃO

RELAÇÃO DOS ALGORITMOS

1. Listas coalescentes.
2. Listas independentes.
3. Listas independentes com encadeamento em área separada.
4. End. aberto visita linear.
5. Remoção com visita linear.
6. Método de Brent.
7. Hashing limitado.
8. Rearranjo com custo médio mínimo.
9. Rearranjo com carreira completa, decisão pelo comprimento da carreira.
10. Rearranjo com carreira completa, decisão pelo custo de carreira.
11. Hashing limitado, rearranjo constante, melhor troca e decisão pelo comprimento da carreira.
12. Hashing limitado, rearranjo ocasional, melhor troca e decisão pelo comprimento da carreira.
13. Hashing limitado, rearranjo ocasional, primeira troca.
14. Hashing limitado, rearranjo constante, melhor troca e decisão pelo custo.
15. Hashing limitado, rearranjo ocasional, melhor troca e decisão pelo custo.
16. Hashing com limite dinâmico.
17. Hashing com limite dinâmico, rearranjo constante , me-

lhor troca e decisão pelo comprimento da carreira.

18. Hashing com limite dinâmico, rearranjo ocasional, melhor troca e decisão pelo comprimento da carreira.
19. Hashing com limite dinâmico, rearranjo ocasional e primeira troca.

II. CONCEITOS BÁSICOS

II.1. DEFINIÇÃO DO MÉTODO DE HASHING

Seja o conjunto $K = \{k_1, k_2, \ldots, k_m\}$ de chaves que únicamente identificam uma informação a ser armazenada ou recuperada, e $E = \{e_1, e_2, \ldots, e_n\}$ o conjunto de endereços de posições de memória (interna ou externa) onde se pode guardar uma ou mais informações (por simplicidade, uma informação). Normalmente $e_i = i$, $i = 0,1,\ldots, n-1$.

O método funcionaria de uma forma ideal se encontrássemos uma função h(k) tal que:

$$h(k_i) \neq h(k_j) \leftrightarrow k_i \neq k_j \quad \text{com } h(K_i) \varepsilon |0, n-1|, \forall i$$

Desta forma teríamos uma "função de indexação" e bastaria colocar k_i na posição $h(k_i)$, $\forall i$ quando do armazenamento, e ir busca-lo no mesmo lugar na recuperação.

Infelizmente, é muito difícil descobrir a função h, pois existem m^m possíveis funções de K em E, mas somente $m!$ fornecem localizações distintas para cada K_i.

Uma saída que seria ainda bastante boa seria desperdiçar um pouco de espaço fazendo $n > m$, pois assim teríamos n^m possíveis funções e $(n!)/(n-m)!$ funções adequadas.

Mesmo com esse recurso é extremamente trabalhoso encontram uma de tais funções, se não quisermos desperdiçar espaço demais. Como exemplo, KNUTH[2], se tivéssemos 31 cha-

ves (m=31) e 41 endereços (n=41), teríamos cerca de 10^{50} possíveis funções para apenas 10^{43} que fornecem indexação; ou seja, somente 1 em cada 10 milhões será aproveitável. Apesar de existirem algoritmos que obtém tal função, eles podem fornecer um valor de n muito grande e são muito trabalhosos, só tendo aplicação em alguns casos, e para tabelas muito pequenas, vide SPRUGNOLI[5] e GRENIEWSKI[6]. Tornaremos a este item no capítulo III.

Uma observação importante neste ponto, é que o conjunto de chaves K nem sempre é todo conhecido "a priori" (arquivo estático), sendo que estas podem ir chegando aos poucos para serem armazenadas e podem, eventualmente serem até retiradas do arquivo.

Diante da dificuldade em se obter uma função de indexação, pensou-se em encontrar uma função h(k) onde para algumas poucas chaves

$$h(K_i) = h(K_j) \quad \text{se} \quad i \neq j,$$

ou seja, que mapeie K em E de uma forma razoávelmente uniforme. Chamaremos este evento de "colisão" e diremos que as chaves K_i e K_j são "sinonimos".

De uma forma geral o método consiste de duas partes: a primeira, escolher uma função h que nos obrigue a poucas colisões, e a segunda resolver essas colisões, pois se duas ou mais chaves são mapeados para um endereço onde só cabe uma, deve-se escolher, de alguma forma, outro lugar para as restantes.

II.2. ORGANIZAÇÃO DO ARQUIVO

O elemento básico do arquivo é o registro, que é um agregado lógico de informações. Podemos identificar três elementos não obrigatóriamente distintos, em um registro que são: chave, informação de controle e informação.

1) CHAVE - é um campo numérico, alfabético ou alfanumérico que unicamente identifica um registro. Em um arquivo podemos guardar a chave junto com a informação, figura (II.2-1) ou podemos grupar todas as chaves e, a cada uma associar um apontador para onde realmente está a informação, figura (II.2-2). Em casos onde a chave é de tamanho variável, pode ser interessante em cada posição endereçável colocar somente dois campos, o primeiro com o comprimento da chave, e o segundo com o apontador para o seu começo em uma área livre onde se colocou todas as chaves contíguamente, figura (II.2-3).

No caso de chaves longas ou quando a probabilidade de insucesso for grande, podemos utilizar as seguintes técnicas para melhorar a eficiência:

a) Na primeira, se usa uma função $h_2(k)$, que pode ser um subproduto de $h_1(k)$, para calcular um código chamado de "assinatura" por HARRISON[14] e sugerido primeiramente por MORRIS[13]. Esse código é armazenado junto com a chave ou, como no caso anterior, junto com os campos comprimento e apontador, e só se compara a chave que se quer buscar com a

chave armazenada se as assinaturas coincidirem.

b) A segunda é usada em listas encadeadas independentes, vide seção (III.2), e aplica uma série de funções $h_i(k)$, $i = 1,2,...,p$ $0 \leq h_i(k) < t$, onde t é o número de bits da assinatura da lista, a cada chave da lista. Os bits da assinatura, colocada na cabeça da lista, correspondentes aos h_i são ligados, figura (II.2-5).

Na busca de uma chave Kx, só se percorrerá a lista se a assinatura tiver bits ligados em todas as posições $h_i(Kx)$, $i = 1,2,...,t$, condição necessária mas não suficiente para que Kx pertença à lista, HARRISON[14] afirma que a maior quantidade de informação será guardada quando o número de bits ligados e o número de bist desligados forem em média iguais na assinatura.

c) BLOOM[15] sugere algumas alternativas parecidas e, em uma delas é feita uma assinatura do arquivo inteiro que fica em memória interna e, somente se acessará o arquivo realmente, se a assinatura preencher as condições necessárias. Isso é particularmente útil em aplicações onde a probabilidade de busca sem sucesso for alta.

2) INFORMAÇÃO - A informação ocupa um ou mais campos, numéricos, alfanuméricos ou alfabéticos, e inclue a chave. Em alguns casos a informação é somente a chave. Os campos podem ser de tamanho fixo ou variável,e, o número de campos pode ser também variável. Campos de tamanho variável podem ser alocados das seguintes formas: a) Superdimensiona-

do o registro. b) Guardando na área endereçada seu tamanho e endereço, como fizemos para as chaves. c) Usando áreas de tamanho fixo encadeadas.

Quando o número de campos é variável teremos um contador de campos e um descritor para cada campo e poderemos ter: a) Um registro superdimensionado de tamanho fixo. b) Cada campo é um nó de uma lista encadeada.

Campos de tamanho variável são muito comuns em informações alfanuméricas, principalmente quando se utilizam técnicas de compactação com códigos de tamanho variável com a frequência, e (ou) supressão de brancos ou zeros, PIETRACCI[16].

Número variável de campos em um registro é encontrado quando não se deseja superdimensionar o registro, e há diversas ocorrências de um certo tipo de campo de um registro. Exemplo: uma pessoa pode ter diverso números de telefones, ou uma máquina pode ter um número variável de peças.

3) INFORMAÇÃO DE CONTROLE - São todos os campos que não contém informação suprida pelo usuário. É recomendável que este também não tenha acesso a ela. Exmeplos de informação de controle são os apontadores, os campos de tamanho dos campos, assinatura, etc...

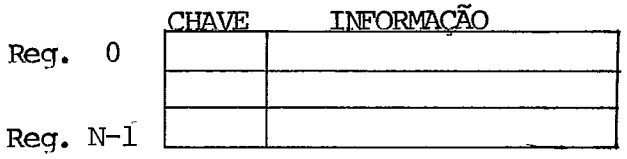


Figura II.2-1

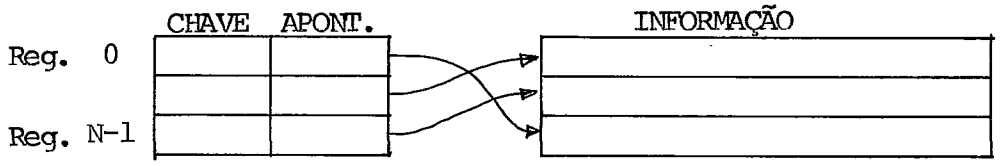


Figura II.2-2

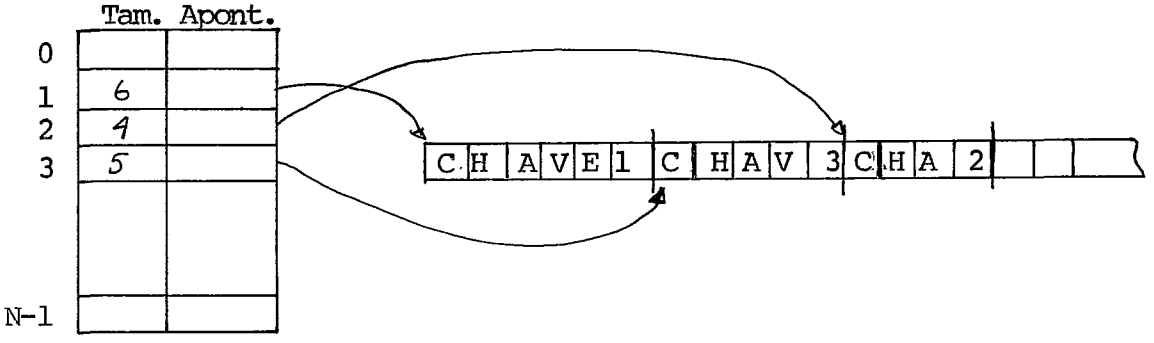


Figura II.2-3

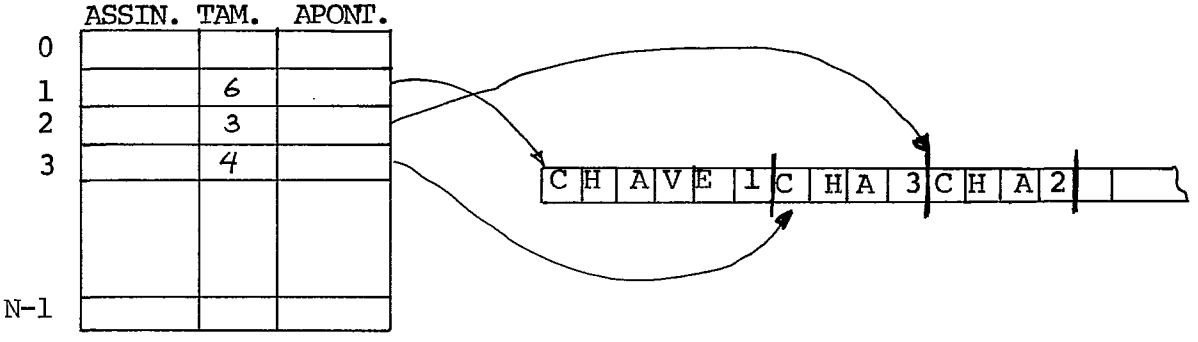


Figura II.2-4

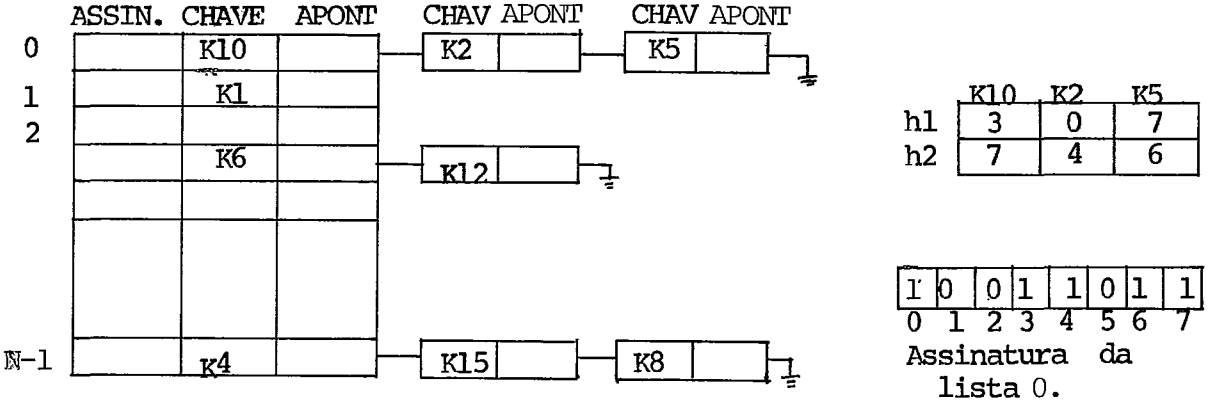


Figura II.2-5

II.3. NOTAÇÃO UTILIZADA

Diversas observações serão feitas neste ponto referentes à notação utilizada nos capítulos posteriores deste trabalho. Não pretendem ser exaustivos mas apenas facilitar ao leitor a compreensão do texto.

1. Notação dos algoritmos

Estão escritos em linguagem semelhante ao ALGOL, porém com as seguintes alterações:

a) as variáveis não são declaradas, exceto quanto se desejar sua condição local;

b) as palavras reservadas estão grafadas em minúscula sublinhadas e os identificadores em maiúsculas;

c) os comentários são do tipo "scape comment" e seu delimitador é "%" ou do tipo "comment" do ALGOL;

d) muitas vezes não são explicitadas certas funções ou subrotinas cujo funcionamento não é essencial à compreensão do algoritmo;

e) da mesma forma, rótulos em comandos de desvio para trechos não essenciais são explicitados mas não declarados e os trechos não constam do algoritmo;

f) procurou-se nomear os identificadores de modo a clarificar o sentido da variável ou trecho do programa a que se referem;

g) os subscritos são delimitados por '()', o sinal de atribuição é ' : = ' ;

h) em caso de qualquer sentença da linguagem

cujo sentido não esteja aqui explicitado, prevalece sua definição em ALGOL.

2. Notação para as colisões em tabela

Fator fundamental para a compreensão do funcionamento dos algoritmos foi a criação de uma notação clara para o percurso de busca por lugar vago ou deslocamento de chaves na tabela. Optamos pelo esquema abaixo com o seguinte significado:

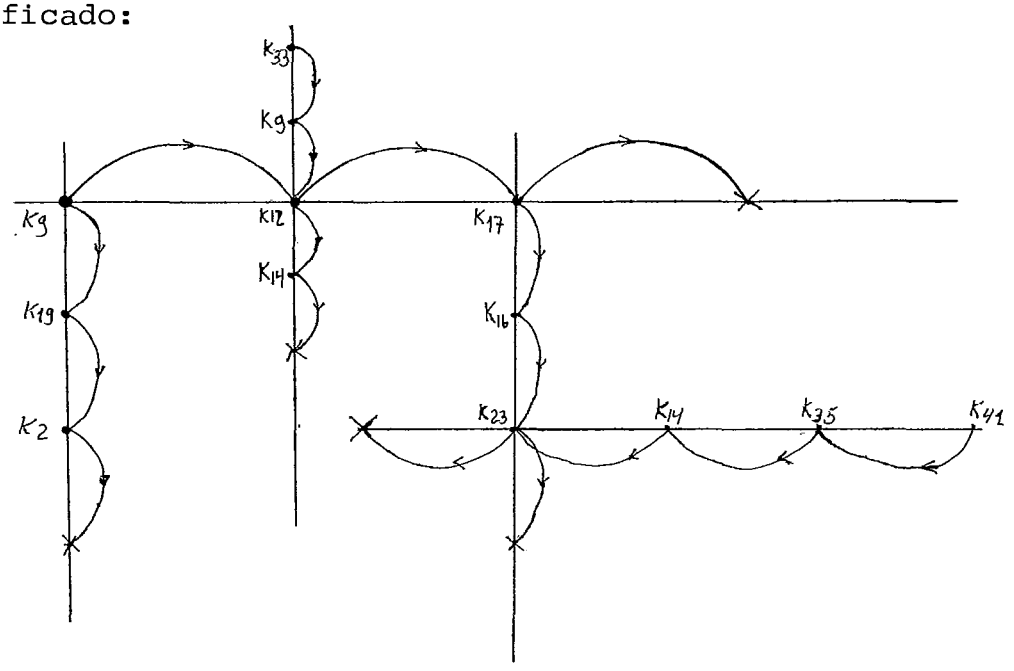


Figura II.3.1

a) as linhas horizontais e verticais representam os endereços na tabela; como as tabelas são, em geral, consideradas circulares uma linha pode representar uma ou mais "voltas" na tabela.

b) na verdade as linhas horizontais e verticais são trechos de um mesmo referencial. Por exemplo, supondo se uma tabela com 9 endereços podemos ter:

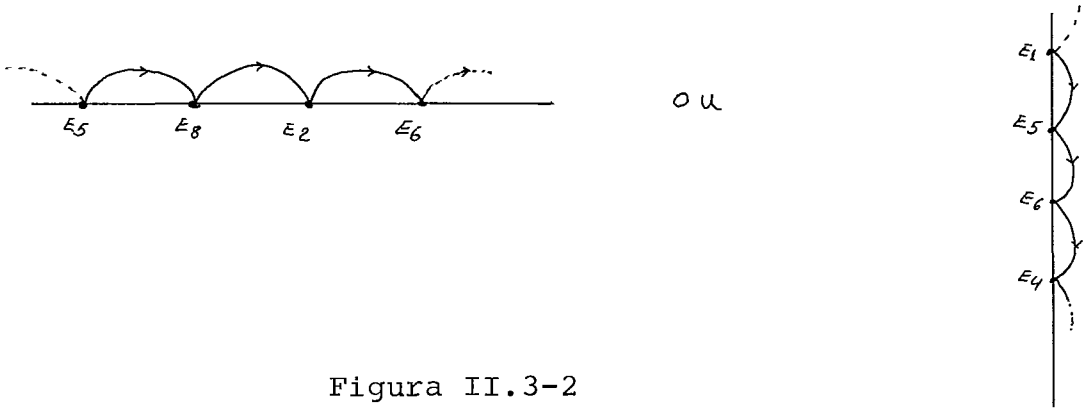


Figura II.3-2

c) os arcos representam saltos nesse referencial, de mesmo tamanho se na mesma linha, mas de tamanhos possívelmente diversos em linhas diversas:

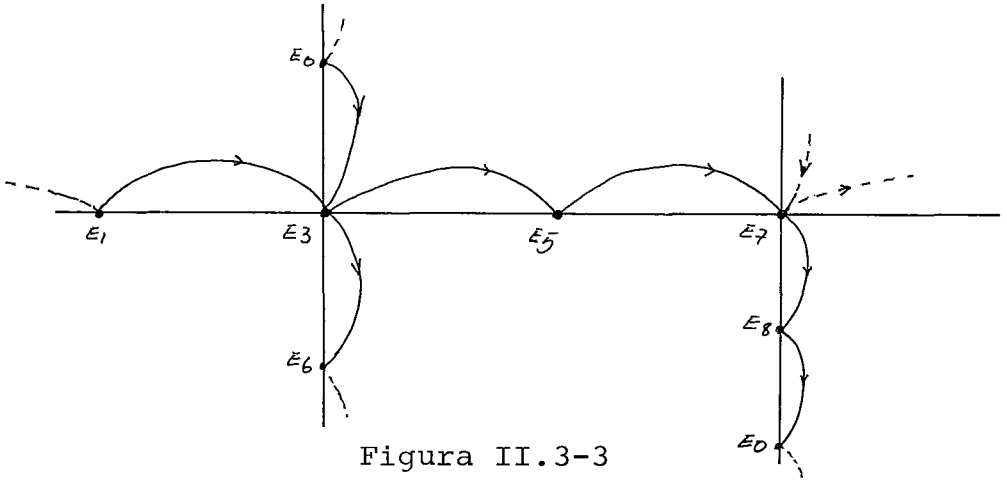


Figura II.3-3

d) endereços não podem se repetir na mesma linhas diferentes.

e) os endereços são, normalmente, omitidos em favor da indicação de quais chaves os ocupam:

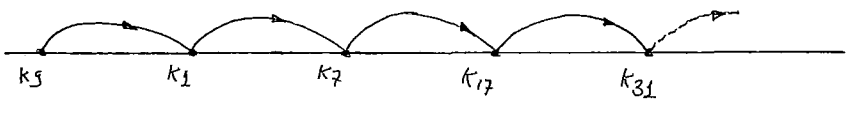


Figura II.3-4

f) lugares vagos são indicados por x:

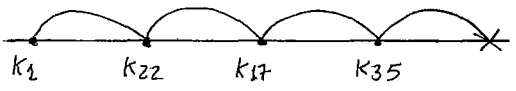


Figura II.3-5

g) chaves em começo de carreiras de saltos horizonatis e verticais simultaneos estão em seu "home-address", ex: $h_1(K_5) = E_1$

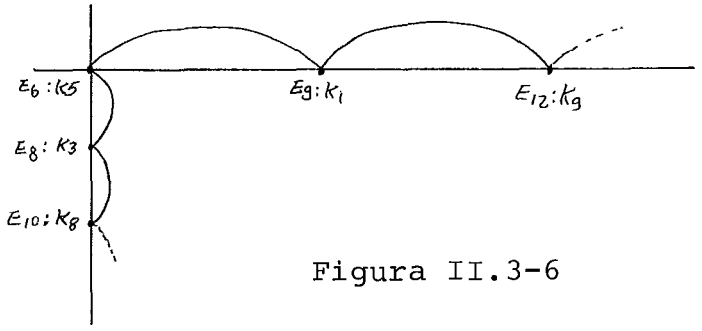


Figura II.3-6

3. Simbologia

A menos da referência explicita no local, no decorrer do texto fixamos:

K_i ou k_i - valor numérico ou alfanumérico da chave índice i.

E_i ou e_i - valor do endereço de índice i.

$h(k_i)$ - valor da função de "hashing" para a chave k_i

n - comprimento da tabela (número total de endereços).

m - número de elementos do conjunto de chaves à inserir ou número de chaves presentes na tabela.

λ - "null link", apontador vazio.

α - fator de ocupação da tabela, igual ao número de chaves presentes dividivo pelo comprimento da tabela.

C_m - número médio de comparações para busca com sucesso das m chaves na tabela.

C'_m - número médio de comparações para busca sem sucesso em tabela contendo m chaves.

$h_j(K_i)$ - valor da j-ésima função de hashing para a chave K_i

mod - resto da divisão inteira

s - número de saltos de uma chave após colisão

$\lfloor x \rfloor$ - maior inteiro contido em x real

$\lceil x \rceil$ - menor inteiro que contem x real

H_k - k-ésimo número harmonico de primeira ordem

p_j - probabilidade de referência à chave K_j

c_j - comprimento da carreira da chave K_j, igual ao número de saltos entre sua posição real e seu "home-address" + 1.

III. HISTÓRICO E REVISÃO DA LITERATURA

III.1. MÉTODOS PARA TRANSFORMAÇÕES DE CHAVES EM ENDEREÇOS

Segundo o que dissemos no capítulo II, e que é consenso entre os autores conhecidos, uma boa função de hashing deve ocasionar o mínimo de colisões e, para arquivos em memória interna deve ser também rápida de calcular. Teóricamente é impossível gerar números uniformemente distribuidos a partir de dados viciados, e as chaves geralmente tem certos padrões; mas, na prática é possível obter-se endereços razoavelmente bem distribuidos, com funções relativamente simples; e algumas vezes é possível tirar proveito de alguma não aleatoriedade dos dados e obter-se uma distribuição para os endereços "mais uniforme" do que a teórica para números aleatórios, KNUTH[2].

É importante notar que não existe função "ótima" no sentido de que apresente distribuições uniformes para quaisquer dados. A escolha da função depende dos dados que vamos armazenar (chaves), da máquina que se pretende utilizar, da linguagem em que se vai programar e da organização do arquivo que for escolhida (memória interna x disco, listas encadeadas x endereçamento aberto).

BUCHHOLZ[11] conjecturou e LUM[8] comprovou experimentalmente que a distribuição de endereços "mais uniforme" não ocorre quando se usa uma função que gere endereços aleató

riamente, independente dos vicios dos dados originais, e sim quando se aproveitam exatamente essas propriedades.

A seguir descreveremos os principais métodos para uso geral, e depois iremos comentá-los brevemente.

III.1.1. DIVISÃO - KNUTH[2], LUM[8]. é simplesmente $h(k_i) \leftarrow k_i \bmod n$ onde n é inteiro e positivo, de preferência um número primo. Segundo LUM[8], é suficiente que n não tenha divisores menores de 20 para que se obtenha bons resultados.

III.1.2. MEIO DO QUADRADO , KNUTH[2] e LUM[8]. Neste método, a chave Ki é multiplicada por si mesma e são tomados $\lfloor \log_2 n \rfloor$ bits do centro do produto que ocupa duas palavras de tamanho w , o que só permite endereçar diretamente, tabelas onde n é uma potência de 2. A idéia aqui é que os bits mais centrais do produto dependem praticamente da chave toda, figura (II.1-1).

$$h(Ki) = \left[(Ki \times Ki) \times 2^{(w - \lfloor \log_2 n \rfloor / 2)}\right] / 2^{(2w - \lfloor \log_2 n \rfloor)}$$

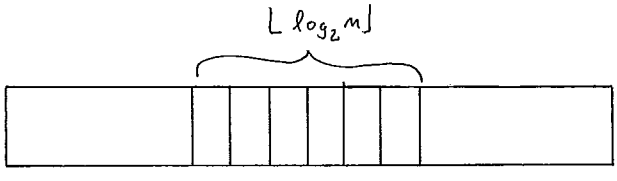


Figura II.1-1

III.1.3. MULTIPLICAÇÃO , KNUTH[2], neste método escolhemos uma constante inteira A , primo com o tamanho da palavra

do computador w , e tamaremos os $\lfloor \log_2 n \rfloor$ bits mais centrais da metade direita do produto A x Ki , figura (II.1-2).

$$h(Ki) = [(A.Ki).2^w]_{/2}^{(2w-\lfloor \log_2 n \rfloor)}$$

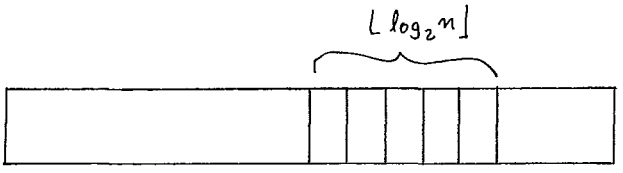
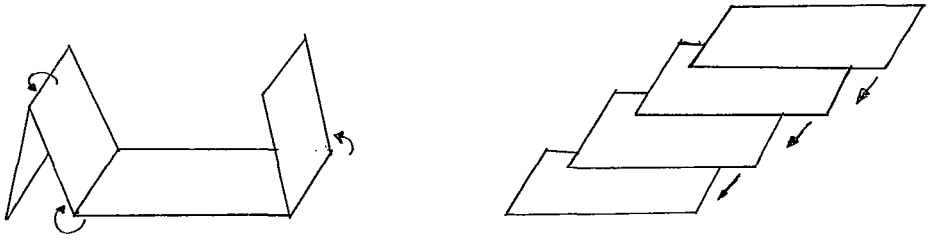


Figura II.1-2

III.1.4. DOBRAMENTO , LUM[8], aqui a chave Ki é dividida em partes de tamanho $\lfloor \log_2 n \rfloor$, exceto a última, e estas partes são superpostas com soma ou com "ou exclusivo", sendo que este último apresenta a vantagem de não causar tranbôrdo. A superposição pode ser feita de duas formas, dobrando-se a chave como um formulário contínuo, figura (II.1-3a) ou como folhas de papel soltas, figura (II.1-3b).



(a) (b)

Figura II.1-2

III.1.5. COMENTÁRIOS

a) Os 3 últimos métodos nos dão endereços no

intervalo $[0,2^{\lfloor \log_2 n \rfloor}-1]$, logo, para serem usados diretamente é necessário que o tamanho da tabela seja um potência de 2, do contrário teremos que tomar mod n.

b) No primeiro método, o tamalho da tabela não poderá ser uma potencia de 2 e deverá ser um número primo KNUTH[2] ou não ter divisores menores do que 20, LUM[8].

c) O método de DOBRAMENTO é especialmente utilizado quando as chaves são maiores do que uma palavra do computador, não podendo então serem aplicados os métodos de 1 a 3. Para melhores resultados pode-se aplicar um dos três primeiros métodos após o dobramento.

d) De experimentos com arquivos reais de diversos tipos e tamanhos, LUM[8] retirou as seguintes conclusões:embora o método do meio-do-quadrado forneça melhores resultados em média, para alguns arquivos ele se comporta muito mal enquanto o método da DIVISÃO deu em média resultados quase tão bons, sendo que de uma maneira razoável em todos eles.

e) Dependendo da velocidade de execução das instruções, pode não ser interessante usar o método da divisão, escolhendo-se um dos outros.

f) CLAPSON[12] estudou detalhadamente chaves numéricas em EBCDIC e obteve um conjunto de divisores com propriedades especiais, que aproveitam propriedades deste código. Como o menor destes números, 15329 era muito grande para endereçar trilhas de disco, ele adotou um processo de dupla divisão, em que a primeira divisão, por um desses divisores seria para uniformizar a distribuição.

g) Em alguns casos, onde os dados são conheci-

dos "a priori", e a tabela será buscada muitas vezes o que justifica um trabalho extra na escolha da função de hash, quatro métodos podem ser tentados:

1) Buscar uma função de indexação usando um dos algoritmos de SPRUGNOLI[5]. Só é aplicável para arquivos muito pequenos (no seu exemplo usou-se uma tabela de tamanho 12), e a funções terá divisões, o que pode não ser interessante em algumas máquinas onde essa operação é muito lenta.

2) Fazer uma análise da distribuição dos dígitos ou bits por posição, escolhendo-se as $\lfloor \log_2 n \rfloor$ posições mais bem distribuidas para, justapostas formarem o endereço, ou serem entrada no método de divisão se n não for potência de 2, LUM[8].

3) Escolher uma série de funções, aplicá-las aos dados e escolher a melhor. Isto pode ser trabalhoso, mas para tabelas pequenas muito pesquisadas (exemplo: tabela de palavras chaves de um montador "assembler" ou compilador), apresentou bons resultados.

4) Para reduzir o trabalho do método anterior, foi por nós aplicado à mão uma modificação do algoritmo de Dijkstra para encontrar o caminho de custo mínimo entre dois pontos de um grafo, com bons resultados. O algoritmo seleciona rapidamente a melhor função dentre as escolhidas "a priori", se as chaves (os arcos), forem ponderados com sua probabilidade de referência e inseridas em ordem decrescente de probabilidade. (O método utiliza uma tabela para cada função (que

podem ser guardadas em memória auxiliar) e vai totalizando o custo ($\sum p_i c_i$, onde c_i é o número de comparações que se necessitará para buscar Ki, essa informação é disponível no momento da inserção, e p_i é a probabilidade de referência à Ki) , para cada tabela. A cada passo o algoritmo escolhe a tabela de menor custo e segue nela inserido até que seu custo ultrapasse o de outra, mudando então para esta.

O método não foi por nós programado ainda,por estar fora do escopo deste trabalho, mas parece altamente promissor, mesmo para tabelas razoavelmente grandes.

III.2. TRATAMENTO DE COLISÕES POR ENCADEAMENTO

O encadeamento foi o primeiro método descoberto e é largamente utilizado com diversas organizações diferentes.

Se duas ou mais chaves forem mapeadas para o mesmo endereço, fazemos uma lista encadeada com cabeça nesse endereço e colocamos as chaves na lista.

As listas podem ter diversos aspectos, sendo que os nós podem ser alocados a partir de lugares vazios na própria tabela ou em área separada, as listas podem ser mantidas independentes (todas as chaves de uma lista são sinôminos), ou podem fundir-se em alguns pontos; as cabeças de lista podem estar na memória principal e o resto em memória auxiliar ou tudo em memória principal ou tudo em memória auxiliar.

O método de encadeamento é geralmente muito rápido, pois as listas em geral são pequenas, e se tivermos n chaves e n listas o tamanho médio destas deverá ser m/n.

III.2.1. ENCADEAMENTO NA PRÓPRIA TABELA

II.2.1.1. ALGORITMO 1 - LISTAS COALESCENTES - (WILLIAMS[17])

```
begin    %** A tabela tem n+1 posições,[0,n]e CHAVE0=VAGO
             sempre **
         %** "R" auxilia na procura de lugar vago. Inicialmen
             te R = n+1. **
         %** "KX" é a chave que se deseja buscar ou inserir.**
  I:=h(KX) + 1;
  while  CHAVE(I) ≠ VAGO do
   begin
    if KX= CHAVE(I) then go to ENCONTRADO;
    if A(I)=VAGO then          %** Final da lista **
                  begin
                    while CHAVE(R)≠VAGO do R:=R-1; %**Busco lugar**
                    if  R=0   then go to TABELACHEIA;
                    A(I):=R;  I:=R;
                  end;
                 else   I:=A(I);     %** Sigo a lista **
   end;
    %** Inserção.**
    CHAVE(I):=KX; A(I):=VAGO;
end.
```

ORGANIZAÇÃO DA TABELA PARA O ALGORITMO 1.

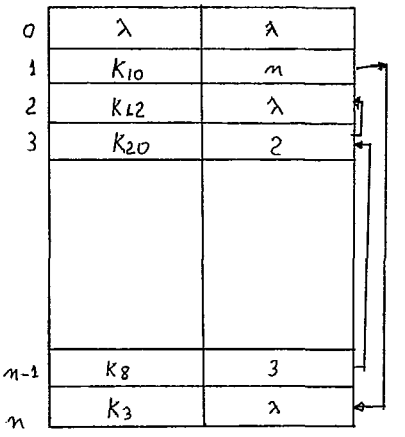


Figura III.2.1.2-1

OBSERVAÇÕES SOBRE O ALGORITMO 1.

a) As listas vão se fundindo e após algum tempo deverá haver listas que contenham elementos com "home-address" diferentes; chamamos "home-address" ao endereço h(Ki).

b)Com o método de procurar lugar vazio utilizado, proposto por WILLIAMS[17], haverá um acumulo de chaves no fim do arquivo, pois se aloca espaço a partir de lá; mas a busca por lugar vazio é mais rápida e, para enchermos a tabela inteira procuraremos no máximo n posições. Se procurarmos lugar vazio a partir da posição onde houve a colisão teremos a vantagem de grupar fisicamente os registros da mesma lista o que é significante em memória externa ou memória virtual, mas teremos no pior caso de procurar em n(n-1)/2 posições no enchimento da tabela toda.

c) A análise de KNUTH[2] nos dá como estimativa da performance o seguinte:

$$Cm \simeq 1 + \frac{1\alpha}{4} + \frac{1}{8\alpha}(e^{2\alpha}-1-2\alpha)$$ número esperado de comparaçoes para uma busca com sucesso.

e, o número esperado de comparações para uma busca sem sucesso será:

$$Cm' \simeq 1 + \frac{1}{4}(e^{2\alpha}-1-2\alpha)$$

onde α é a taxa, ou fator de ocupação definida como: $\alpha = m/n$.

d) Poderemos manter as listas independentes, se deslocarmos a chave que não estiver no seu "home-address" quando ali quisermos inserir uma chave. Em termos de eficiência na busca não se ganha muito, mas ganha-se algumas propriedades como facilidade de remoção , e a que é evidenciada no próximo algoritmo.

III.2.1.2. ALGORITMO 2 - LISTAS INDEPENDENTES (B. LAMPSON in KNUTH[2])

```
begin     %** Como as listas são independentes,se utilizarmos para a função
          %** de hashing o método da DIVISÃO, poderemos armazenar na tabela
          %** não a chave, e sim o quociente da divisão que é menor do
          %** esta, pois com o quociente e com o resto poderemos reconstitu
          %** ir a chave. Será utilizado um campo "TAG" de um bit para indi
          %** car (TAG=1) as cabeças das listas. As listas são circulares.
          %** CHAVE(0)=VAGO,
          %**    Q(KX)=⌊KX/n⌋          %** Quociente **
          %**    H(KX)=KX-Q(KX)*n      %** Resto     **
  I:=j:=h(KX)+1;
  if  CHAVE(I)=VAGO then begin          %** Inserção direta **
                           TAG(I):=1;   CHAVE(I):=Q; A(I):=I;
                         end
                    else                %** Já há alguém no lugar **
                      if TAG(I)=0
                       then begin       %** E não é cabeça de lista **
                              while A(I)≠J do I:=A(I);%**segue a lista
                              while CHAVE(R)≠VAGO do R:=R-1;
                              if R=0 then go to TABELACHEIA;
                              CHAVE(R):=CHAVE(I); A(R):=A(J); A(I):=R;
                              CHAVE(J):=Q; A(J):=J; TAG(J):=1;
                            end
                       else begin   %** É cabeça de lista. **
                              do begin
                                   if Q=CHAVE(I) then go to ENCONTRADO
                                 end
                              until A(I)=J;
                              while CHAVE(R)≠VAGO do R:R-1;
                              if  R=0  then go to TABELACHEIA;
                              A(I):=R; TAG(R):=0; CHAVE(R):=Q; A(R):=J;
                            end;
end.
```

ORGANIZAÇÃO DA TABELA PARA O ALGORITMO 2

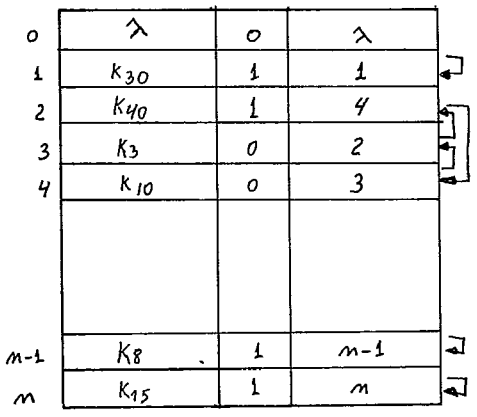


Figura II.2.1.2-2

OBSERVAÇÕES SOBRE O ALGORITMO 2

a) As listas são mantidas independentes, pois as chaves ou estão no seu "home-address" e são cabeças de lista ou estão em outro lugar mas pertencem a lista que começa em seu "home-address".

b) Valem as mesmas observações do algoritmo anterior, quanto à procura de lugar vazio.

c) Pela análise do KNUTH[2], temos as seguintes estimativas para a performance:

Número esperado de comparações para uma busca com sucesso:

$$Cm = 1 + \frac{m-1}{2n} \simeq 1 + \frac{1\alpha}{3}$$

Número esperado de comparações para uma busca sem sucesso:

$$Cm' = (1 - \frac{1}{n})^m + \frac{m}{n} \simeq e^{-\alpha} + \alpha$$

III.2.2. ENCADEAMENTO EM ÁREA SEPARADA

Aqui nós alocamos espaço extra para colocar as colisões, em uma área separada da tabela. Com essa organização o fator de ocupação pode ser maior do que 1, embora nessa faixa a busca já não é tão eficiente pois as listas começam a ter um comprimento médio maior do que 1.

III.2.2.1. ALGORITMO 3 - ENCADEAMENTO EM ÁREA SEPARADA

```
begin     %** AVAIL é a função que administra a lista de
          %** espaço disponível.
  I:=H(KX);
  if  CHAVE(I) = VAGO  then  CHAVE(I):=KX  %** Inserção direta **
                       else  begin
                               while  A(I)≠VAGO do I:=A(I);
                               J ← AVAIL;
                               CHAVE(J):=KX; A(I):=J;
                             end;
end.
```

ORGANIZAÇÃO DA TABELA PARA O ALGORITMO 3.

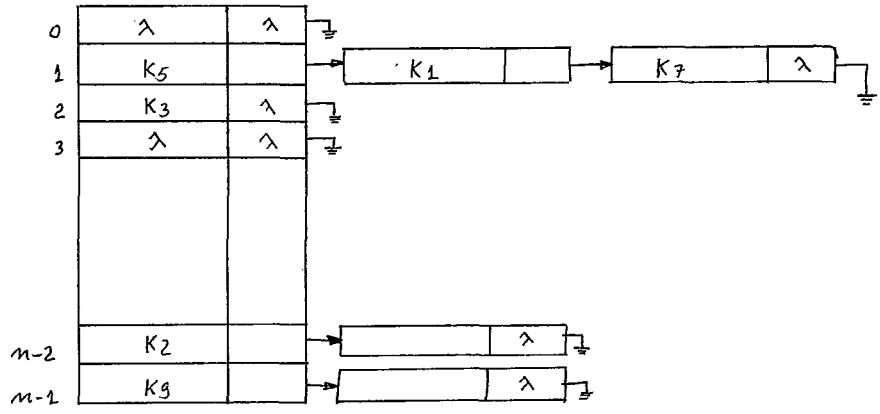


Figura III.2.2.1-1

OBSERVAÇÕES SOBRE O ALGORITMO 3

a) As listas são independentes, mas também podem ser usadas listas coalescentes em área separada.

b) Outra organização muito usada, talvez mais do que a descrita é ter na cabeça das listas somente os apontadores, o que é útil se os nós estão em memória externa, pois podemos ter uma tabela maior com o mesmo espaço, e listas muito mais curtas. Chama-se essa organização "tabela de espalhamento", MORRIS[13], figura (III.2.2.1-2).

TABELA DE ESPALHAMENTO

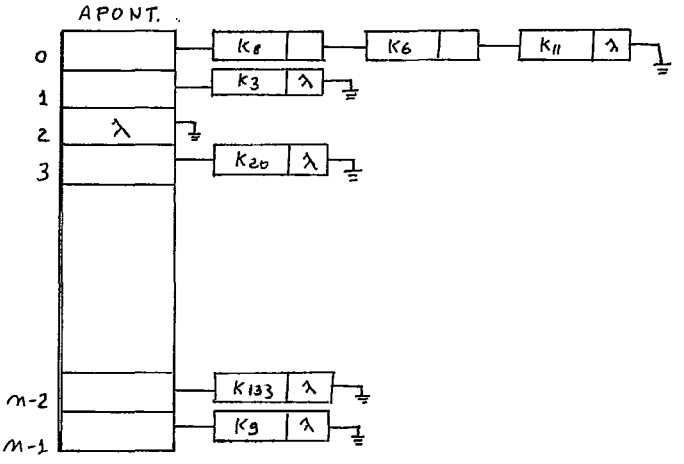


Figura III.2.2.1-2

III.3. TRATAMENTO DE COLISÕES POR ENDEREÇAMENTO ABERTO

Neste método, não são utilizados apontadores, o que economiza espaço, simplifica e acelera os algoritmos de busca. Os resultados em termos de número médio de comparações são piores do que os de encadeamento, mas com o espaço ganho aos apontadores geralmente se pode aumentar a tabela e trabalhar com um fator de ocupação mais baixo, com um algoritmo mais rápido, o que é melhor em muitos casos. Para busca em memória interna, o endereçamento aberto economiza memória, e para busca em memória externa é usado por grupar os sinônimos, fisicamente próximos (em alguns métodos), minimizando o tempo de movimentação da cabeça de leitura/escrita.

A característica importante desse método, é que a informação sobre o encadeamento é armazenada implicitamente na posição das chaves na tabela e na regra de "escolha

dos sucessores" usada para resolver as colisões.

O processo de uma forma geral, é o seguinte, que pode ser descrito com a ajuda do diagrama da figura (III. 3-1):

Para inserção: geramos uma sequência de endereços hj(Ki), j = 1,2,...,n , e inserimos no primeiro deles que estiver vago.

Para busca: geramos a mesma sequência de endereços até que, encontremos Ki, caso em que a busca é com sucesso, ou encontremos um lugar vago, e sabemos que Ki não se encontra na tabela e a busca é sem sucesso.

Figura III.3-1

O nome "endereçamento aberto" foi dado por PETERSON[19] em seu clássico artigo de 1957, que parece ter sido um dos seus descobridores, em paralelo com ERSHOV[20].

Com apenas duas exceções recentes, CLAPSON[12] em um artigo de 1977 e BRENT[18] em outro de 1973 todos os trabalhos publicados sobre endereçamento aberto versavam sobre a escolha das funções $h_1, h_2, \ldots, h_n$, de modo que fossem rápidas de calcular, dessem poucos agrupamentos e percorressem o máximo da tabela. BRENT[18], como veremos mais adiante, foi o pri-

meiro a explorar o fato de que em alguns métodos, se inserirmos K1, K2 e K3 nessa ordem ou em outra como K2,K1 e K3 o resultado obtido em termos de custo para as buscas futuras, podem ser diferentes. CLAPSON[12], que também será citado mais adiante, aproveitou o fato de que, se impusermos um limite L para o número de "saltos" (Nº de funções h geradas) feitos na inserção, teremos automáticamente limitado o número de comparações, ou de "saltos" possíveis na busca, evitando piores casos desastrosos que limitavam a gama de aplicações do endereçamento aberto.

Nos algoritmos que se seguem como em todos de endereçamento aberto, a tabela é feita circular, tomando-se para endereço o valor $h_j(Ki)$ mod n, onde n é o tamanho da tabela e começando o endereçamento de zero e indo até n-1. Estenderemos agora o conceito de "home-address", como sendo $h_1(Ki)$.

III.3.1. VISITA LINEAR - foi o primeiro método utilizado e o único até cerca de 1968, quando MORRIS[13] lançou a idéia de visita aleatória.

Esse método percorre a tabela circularmente, com saltos de tamanho fixo e independente da chave ou do lugar onde ocorreu a colisão, sendo geralmente o salto de tamanho 1. As funções tomam a seguinte forma:

$$\begin{cases} h_1(Ki) \text{ é uma das funções descritas na seção (III.1)} \\ h_j(Ki) = [h_1(Ki) + (j-1)] \bmod n,\ j > 1 \end{cases}$$

III.3.1.1. ALGORITMO 4 - BUSCA E INSERÇÃO COM VISITA LINEAR

```
begin
  integer procedure SEQ(I); integer i;
    begin
      SEQ:=if i < (n-1) then I+1 else 0;
    end;
  I:=h(KX);
  %** Laço de busca **
  while CHAVE(I)≠VAGO do if CHAVE(I)=KX then go to ENCONTRADO;
                                         else I:=seq(I);
  %** Inserção **
  if M=N-1 then go to TABELACHEIA
           else begin
                  M:=M+1;
                  CHAVE(I):=KX;
                end;
end.
```

OBSERVAÇÕES SOBRE O ALGORITMO 4

a) Duas chaves que colidiram tomarão o mesmo caminho ("carreira"), logo, se a função hash h_1 causar algum agrupamento, o que na prática sempre ocorre, este irá se repetir em alguns outros lugares da tabela, que chamaremos de "agrupamentos secundários", e se o incremento for pequeno, tenderá a aumentar o próprio agrupamento em torno do seu "home-address", que chamaremos de "agrupamento primário".

b) Os agrupamentos tendem a crescer rápidamente pelo seguinte fenomeno: se a posição i estiver ocupada e as posições i-1 e i+1 estiverem vazias, a probabilidade da posição i+1 vir a ser ocupada na próxima inserção, será o dobro daquela de uma posição vazia isolada, pois chaves que forem mapeadas para i e para i+1 serão armazenadas em i+1 , formando um agrupamento de tamanho 2. Na próxima inserção a probabilidade da posição i+2 vir a ser ocupada será o triplo daquela de uma posição isolada, e assim por diante.

c) Quando um agrupamento cresce e se funde a outro agrupamento, a posição seguinte a esse último tem sua probabilidade aumentada drásticamente, o que incentiva ainda mais o crescimento do novo agrupamento.

d) Segundo análise do KNUTH[2] temos as seguintes estimativas da performance;

Número médio esperado de comparações para uma busca com sucesso:

$$Cm \simeq \frac{1-\alpha/2}{1-\alpha} \qquad \text{(III.3.1.-1)}$$

Número médio esperado de comparações para uma busca sem sucesso:

$$Cm' \simeq \frac{1}{2}\left[1 + \left(\frac{1}{1-\alpha}\right)^2\right] \qquad \text{(III.3.1.1-2)}$$

e) A deleção pura e simples de uma entrada da tabela não é permitida, pois perderíamos o acesso às chaves que colidiram com esta ou com alguma outra em endereços "anteriores" e foram armazenadas em posições circularmente "posteriores". A primeira idéia que ocorre é colocar uma marca na posição em que a chave foi retirada, indicando que o lugar está vago para inserção, mas que não se pode parar uma busca nesse ponto. Essa solução só funciona se as deleções forem muito raras, pois do contrário após algum tempo só haveria dois tipos de entradas na tabela: as ocupadas e as removidas; e uma busca sem sucesso seria desastrosa pois percorreríamos o arquivo inteiro. Teríamos de incluir mais um teste no laço de busca para não entrar em "looping" e, após um longo ciclo de deleções/inserções, número médio de comparações para uma busca com sucesso deixa de obedecer à fórmula (III.3.1.1-1) e se aproxima dos valores dados por (III.3.1.1-2) enquanto que $Cm' \to n$, KNUTH[2].

Uma alternativa à esse processo de marcar as entradas que ficaram vazias, é mover as chaves que poderiam ter o seu acesso prejudicado pela remoção, para mais perto

do seu "home-address". O algoritmo 5 faz a remoção por esse processo, e a tabela resultante tem os mesmos valores Cm e Cm' que a original teria com uma chave a menos, não havendo portanto degradação da performance como no caso da marcação. O algoritmo 5 só tem o inconveniente de ser muito lento para tabelas que não estejam muito vazias.

III.3.1.2. ALGORITMO 5 - REMOÇÃO DA ENTRADA CHAVE(I)

```
begin
  CHAVE(I):=VAGO;   J:=I;   I:=seq(I);
  while  CHAVE(I)≠VAGO  do
    begin
      R:=h(CHAVE(I));
      if  R < =J < =I or I < R < =J or J < =I < =R then CHAVE(J):=
                                                        CHAVE(I);
      J:=I;     I:=seq(I);
    end;
end.
```

III.3.2. VISITA ALEATÓRIA - publicada por MORRIS[13] em 1968, o método usa um gerador de números pseudo-aleatórios para gerar os $h_j(Ki)$, $j > 1$. A sequência gerada é sempre a mesma e é somada mod n à $h_1(Ki)$. O método elimina os agrupamen-

tos primários, mas os agrupamentos secundários ainda permanecem, porque as carreiras começadas em posições contíguas correrão contíguas também.

Seja r_i, $i = 1,2,\ldots,n$ $\quad 0 \leq r_i \leq n-1$ a sequência de número aleatórios inteiros obtidos pelo gerador.

$$\begin{cases} h_1(Ki) \text{ é uma das funções descritas na seção (III.1)} \\ h_j(Ki) = (h_1(Ki) + r_{j-1}) \bmod n, \; j > 1 \end{cases}$$

III.3.3. VISITA QUADRÁTICA - proposto por MAURER[21] também em 1968, esse método usa como sequência:

$$\begin{cases} h_1(Ki) \text{ é uma das funções descritas na seção (III.1)} \\ h_j(Ki) = [h_1(Ki)+a(j-1)+b(j-1)^2] \bmod n, \; j > 1 \end{cases}$$

onde a e b são constantes. As características em relação à agrupamentos são as mesmas do método anterior e o cálculo do polinômio pode ser feito só por adições, o que é mais rápido do que gerar pseudos-aleatórios. Esse método só percorre metade da tabela, o que não é problema se não tivermos fator de ocupação excessivamente alto. RADKE[24] propos uma modificação no método, que faz com que ele percorra toda a tabela.

III.3.4. - VISITA COM INCREMENTO BINÁRIO - CLAPSON[12] propos em 1977 esse método como um compromisso entre a visita linear e os métodos que utilizam incrementos grandes (os dois

anteriores e os tres que vem a seguir). O seu interesse é para busca externa, onde é desejável que os sinônimos fiquem próximos fisicamente. A sequência visitada é a seguinte:

$$\begin{cases} h_1(Ki) \text{ é uma das funções descritas na seção (III.1)} \\ h_j(Ki) = [h_1(Ki)+2^{j-1}-1] \bmod n, \; j > 1 \end{cases}$$

A sequência obtida é: $h_1(Ki)+1$, $h_1(Ki)+3$, $h_1(Ki)+7$, ...
Afirma ele que duas carreiras somente se interceptarão uma vez, o que não é verdade, pois do diagrama (III.3.4-1) derivamos que se certas condições forem preenchidas as duas carreiras se encontrarão em x e novamente em y.

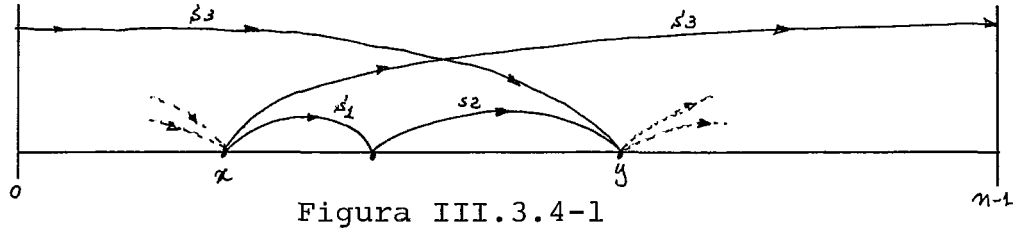


Figura III.3.4-1

$$x + s1 + s2 = (x + s3) \bmod n$$

para $x + s3 < 2m$ temos:

$$m = s3 - s1 - s2$$

Fazendo s1=7, s2=15, e s3=63, temos m=41 que é inclusive primo. Valores de m que satisfazem e não sejam primos são ainda mais fáceis de serem obtidos. As duas sequências cortar-se-ão em x e y com $m < 19$.

Mas se a tabela for grande e o número de saltos em cada carreira for baixo, não haverá mais de um cruzamento para duas carreiras que se originam próximas, resolvendo o problema dos agrupamentos primários, mas com agrupamen-

tos secundários ainda. Tem também as vantagens da sequencia começar com valores baixos, mas sem agrupamentos primários e ser extremamente rápida de gerar, bastando um "shift" para a esquerda e um "or" com 1.

Estamos providenciando a publicação da correção da afirmativa de CLAPSON[12].

III.3.5. DUPLA DIVISÃO - BALBINE[23] em 1968 apresentou a idéia que realmente eliminou os agrupamentos secundários, denominada DUPLO HASHING, que é a base dos três algoritmos que seguem.

Nestes métodos o incremento a ser somado a $h_{j-1}(Ki)$ para se obter $h_j(Ki)$ é função de Ki e não mais função de $h_1(Ki)$ (como nas visitas aleatória e quadrática), ou constante (como na visita linear). Obtém-se então uma segunda função independente (no sentido probabilístico do termo) de $h_1(Ki)$, que nos fornece o incremento.

OBSERVAÇÕES SOBRE HASH-DUPLO

a) KNUTH[2] nos dá as seguintes estimativas da performance do hash-duplo, que se comporta exatamente como o conceito de "hash uniforme" proposto por PETERSON[19].

Número médio de comparações esperadas para uma busca com sucesso:

$$Cm = \frac{n+1}{m} (H_{n+1} - H_{n+1-m}) \simeq -\alpha^{-1} \ln(1-\alpha)$$

Número médio esperado de comparações para uma busca sem sucesso:

$$Cm' = \frac{n+1}{n+1-m} \simeq (1-\alpha)^{-1}$$ que, para a tabela cheia nos dá (n+1)/2 comparações em média, que é o mesmo que para uma busca sequencial. Hn é o n-ésimo número harmônico de primeira ordem dado pela fórmula abaixo:

$$H_n = \sum_{i=1}^{n} 1/i$$

b) No método (III.3.5) podemos substituir a operação mod(n-2) por mod $2^{\log_2 n} - 1$ através de um "and" com uma máscara da forma $2^{\log_2 n} - 1$ o que nos dá uma segunda função tão boa quanto a primeira em termos de distribuição só que abrangendo uma faixa um pouco menor. Essa modificação foi utilizada por BELL[26] e KARMAN[26] no método (III.3.7) para garantir que $\lfloor Ki/n \rfloor < n$.

O método original consiste em:

$$\begin{cases} h_1(Ki) = Ki \bmod n \\ h_j(Ki) = [h_{j-1}(Ki) + (Ki \bmod (n-2) + 1] \bmod n, \quad j > 1 \end{cases}$$

III.3.6. QUOCIENTE QUADRÁTICO - BELL[25] em 1970 publicou esse método, que tem como idéia central a mesma de BALBI-

NE[23] mas que parece ter sido descoberto independentemente desse. Aqui não é o incremento que depende de Ki, mas sim a constante b do termo quadrático, vide (III.3.3). A nova constante passa a ser b(Ki), uma função de Ki independente de h_1(Ki) (no sentido probabilístico). Se h_1(Ki) for o método da divisão, b(Ki) pode ser obtido como sub-produto da primeira, sendo o quociente da divisão $\lfloor Ki/n \rfloor$.

A modificação proposta por RADKE[24] também pode ser aplicada de modo a percorrer toda a tabela. O método é o seguinte:

$$\begin{cases} h_1(Ki) = Ki \text{ nod } n \\ h_j(Ki) = \left[h_1(Ki) + a(j-1) + b(Ki)(j-1)^2\right] \bmod n, \quad j > 1 \end{cases}$$

onde $b(Ki) = \lfloor ki/n \rfloor \bmod n$

III.3.7. QUOCIENTE LINEAR - BELL[26] e KAMAN[26] depois de implementar o método anterior, propuseram o seguinte, que soma as vantagens dos dois métodos anteriores: função linear, mas com o incremento sendo $\lfloor Ki/n \rfloor$ que é fácil e rápido de obter. Em experimentos com a tabela de símbolos do compilador COBOL do PDP10, mostrou-se cerca de 7% mais rápido por busca, embora o número médio de comparações por busca fosse um pouco maior, cerca de 3%. Com esse método toda a tabela é percorrida. São as seguintes as funções:

$$\begin{cases} h_1(Ki) = Ki \bmod n \\ h_j(Ki) = [h_{j-1}(Ki) + \lfloor Ki/n \rfloor] \bmod n, \; j > 1 \text{ e } \lfloor Ki/n \rfloor > 0 \\ h_j(Ki) = [h_{j-1}(Ki) + 1] \bmod n, \; j > 1 \text{ e } \lfloor Ki/n \rfloor = 0 \end{cases}$$

III.3.8. MÉTODO DE BRENT - Aqui a novidade não está na forma como se resolve o problema das colisões (para o que pode ser utilizado qualquer método de hash-duplo; para demonstrar o seu método BRENT[18] utilizou o método (III.3.5) em 1973), mas sim em certos "rearranjos que são feitos na tabela de forma a otimizar (o ótimo aqui é local), o custo da tabela a cada inserção. O método é vantajoso sempre que o número médio de buscas à tabela for maior do que x vezes o número de inserções feitas nesta (que é o próprio número de chaves na tabela), onde x depende da máquina, linguagem e fator de ocupação utilizados. O pressuposto de que vale a pena gastar um pouco mais de tempo na inserção para ganhar na busca é válido para a maioria das aplicações e em seu exemplo BRENT encontrou $x < 3$ para $0,2 <= \alpha <= 0,99$, ou seja, se cada chave for buscada em média três ou mais vezes.

A idéia básica do método é que o número médio de comparações para uma busca depende da ordem de entrada das chaves na tabela para os métodos de hash duplo.

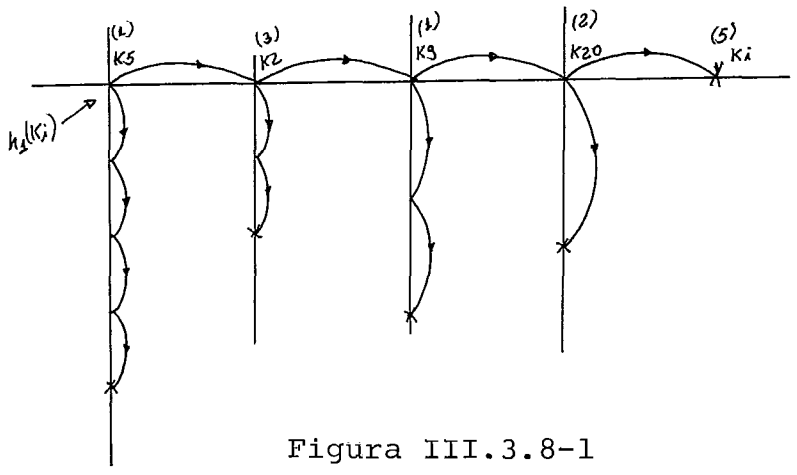


Figura III.3.8-1

O método será explicado com a ajuda das figuras (III.3.8-1) e (III.3.8-2) e logo após será apresentado o algoritmo. Podemos calcular o valor esperado do número médio de comparações para encontrar uma chave qualquer em uma tabela que contêm m chaves pela fórmula abaixo, e que chamaremos "custo da tabela".

(III.3.8-1) $Cm = \sum_{j=1}^{m} p_j c_j$, onde p_j é probabilidade de referência à chave j e c_j é 1 mais o comprimento da "carreira" de j, ou seja, o número de comparações para encontrar a chave Kj. BRENT supos que pi=1/m, i=1,2,...,m.

Aplicando à porção da tabela mostrada na figura (III.3.8-1), onde se acabou de inserir a chave Ki temos:

$$C_5 = \frac{1}{5} \sum_{j=1}^{5} c_j = \frac{1}{5}(1+3+1+2+5) = 2,4$$

Como o comprimento da carreira de Ki é t, verificaremos se é possível deslocar alguma das chaves desta carreira, com exceção da penúltima (K_{20}), colocando em seu lugar Ki, e resultando em diminuição no custo da tabela. Se

mais de uma puder ser deslocada com melhoria, escolheremos a que apresenta custo mínimo, em caso de empate a mais a esquerda. Os valores de Cm se forem deslocados K_5 ou K_2 ou K_9, são respectivamente 2,4:2,2 e 2,4. Logo inseriremos Ki não no final de sua carreira e sim na posição antes ocupada por K_2, sendo este remetido ao final de sua carreira, resultando a tabela da figura (III.3.8-2), que tem custo 2,2. O resultado final é o mesmo que seria obtido se a chave Ki houvesse sido inserida antes da chave K_2. Essa propriedade é válida para o hash-duplo, mas para a visita linear KNUTH[2] provou que qualquer que seja a ordem de inserção a tabela terá sempre o mesmo custo final.

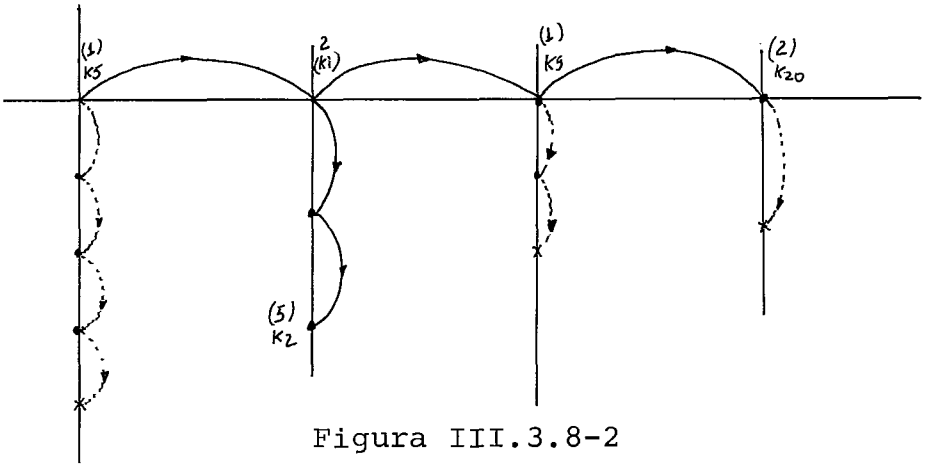


Figura III.3.8-2

OBSERVAÇÕES QUANTO AO MÉTODO DE BRENT

a) Não é necessário conhecermos toda a carreira das chaves que se quer deslocar, nem seu comprimento, mas apenas o "acréscimo" de comprimento que o seu deslocamento for provocar, ou seja quanto falta até o fim da carreira.

b) Também não é necessário sempre examinar todas as chaves

que colidem na carreira de Ki, pois se esta deveria ser armazenada em $h_s(Ki)$ - após s-1 saltos e encontramos uma chave em $H_q(Ki)$, com $q < s$ e acréscimo x tal que $x + q < s$ o que já é um ganho, a chave em h_{q+1} para ser melhor deveria ter acréscimo de $y < x-1$; a seguinte acréscimo de $z < x-2$ e assim por diante. Portanto só é necessário testar as chaves até a posição $h_p(Ki)$ onde $p = x + q - 2$. No exemplo anterior, quando em $h_2(Ki)$ encontramos $x = 2$, poderíamos parar, não havendo mais possibilidades de encontrar melhores condições adiante.

ALGORITMO 6 - BUSCA E INSERÇÃO PELO MÉTODO DE BRENT

```
procedure ALG6 ( KX ); value KX;
begin      %** Busca KX na tabela,e se não for encontrada,insere, **
           %** reorganizando a tabela se for possível.           **
  hash( KX, R, Q );  %** obtenho o "home-address","R" e incremento "Q" **
  %** Procurar **
  S:=0; HS:=R;  TROCA:=false;
  while K(HS)≠VAGO do
   begin
     if K(HS)=KX then go to ENCONTRADO;
     if HS=R then go to TABELACHEIA;
     S:=S+1;
     HS:= (HS+Q) mod n;
   end;
  if  S > 1              %** tento melhorar a tabela. **
   then begin    LIMI:=S-2;
     for  i:=0  step  1  while  I <=LIMI do
      begin
        ENDI:=(R+I*Q)mod N;   %** END. da chave que se quer desl.
        hash( K(ENDI), RI, QI ); %** seu "HOME-ADDRESS" e increm.
        HS2:= (ENDI+QI) mod N;   %** seu próx.salto. **
        S2:=1;  LIM:=LIMI-I+2;
        while  S2 < LIM  and  K(HS2)≠VAGO  do
         begin
           S2:=S2+1;
           HS2:=(HS2+QI) mod N;
         end;
        if S2 < LM       %** Vale a pena deslocar K(ENDI) p/HS2
          then begin     %** e inserir KX em ENDI. Anoto a troca.
                 COLOCAVELHO:= HS2;
                 COLOCANOVO:= ENDI;
                 LIMI:=S2 + I - 2;
                 TROCA:= true;
               end;
      end do loop I;
     end da tentativa de troca;
  if TROCA then begin K(COLOCAVELHO) :=K(COLOCANOVO) ; K(COLOCANOVO) :=KX;
                end
           else  K(HS) :=KX;       %** Inserção normal **
end.
```

OBSERVAÇÕES SOBRE O ALGORITMO 6

a) O algoritmo de busca é identico aos de hash-duplo, não havendo nenhum trabalho adicional.

b) Na inserção, se houver uma colisão ou nenhuma, não é tentado o rearranjo pois seria inútil.

c) As variáveis LIMI e LIM limitam a procura nas carreiras horizontal e vertical respectivamente.

d) Pode acontecer que uma chave que já possua uma carreira comprida, tenha esta aumentada ainda mais de modo a encurtar a carreira da chave que está sendo inserida, mesmo que esta tenha uma carreira menor do que aquela, desde que resulte um ganho para a tabela como um todo. Mais adiante veremos um algoritmo (algoritmo 7) que evita tal fenomeno, mas obtém tabelas um pouco piores na maioria dos casos.

e) BRENT[18] obteve a seguinte estimativa para o número médio de comparações esperado em uma busca com sucesso:

$$Cm = 1 + \alpha/2 + \alpha^3/4 + \alpha^4/15 - \alpha^5/18 + 2\alpha^6/15 + \\ + 9\alpha^7/80 - 293\alpha^8/5670 - 319\alpha^9/5600 + \ldots$$

Para $\alpha = 1$ temos $Cm \simeq 2.4941$ que é significamente melhor que $\sqrt{\pi n/8}$ que é o valor para visita linear ou

$$Cm \simeq H_{n+1} - 1 = \sum_{i=1}^{n-1} 1/i - 1, \text{ para hash-duplo.}$$

f) Para uma busca sem sucesso o valor é o mesmo do hash duplo que é o seguinte:

$Cm' = \frac{n+1}{n+1-m} \simeq (1-\alpha)^{-1}$ que para tabela cheia toma a forma $\frac{m+1}{2}$, o mesmo que em uma busca sequencial.

III.3.9. HASH LIMITADO - CLAPSON[12] em 1977, publicou um método a que deu o nome de "hash com busca limitada", que utiliza uma idéia muito simples que é a seguinte: se limitarmos a entrada ao arquivo das chaves que ultrapassarem um certo limite de comparações, o número máximo de comparações na busca à qualquer chave do arquivo estará automaticamente limitada. Essa idéia é muito simples e imediata, e acreditamos que já deva ter ocorrido à muitas pessoas, mas tem um inconveniente muito grande que é o desperdício de espaço no arquivo, Pois ele é considerado "cheio" com uma ocupação razoavelmente baixa. CLAPSON[12] utilizou uma série de recursos para aumentar a ocupação máxima atingida, mas mesmo assim os valores obtidos não foram muito altos. Ele utilizou o método de INCREMENTO BINÁRIO como descrito em (III.3.4), mas a unidade de endereçamento não foi mais a "entrada" e sim uma "caixa" (ver capítulo V), que poderia conter uma ou mais entradas. Intercaladas com as caixas endereçáveis foram colocadas caixas não endereçáveis.Na inserção de uma chave nova, caso não houvesse lugar na caixa "home-address", seguiríamos a carreira com INCREMENTO BINÁRIO e após um certo número de "saltos" (em torno de 4), visitaríamos um outro número limitado de caixas não endereçáveis. Se após essa carreira não fosse encontrado nenhum lugar vago, a tabela era considerada cheia e bloqueada a futuras inserções. Como exemplo: com 1/16 das caixas não endereçáveis, limite de

4+2 e caixa de tamanho 1, a ocupação máxima garantida (média-2σ) foi 38% com média de 50%. Com caixas de tamanho 10, os valores sobem a 80% e 87%. Para busca interna,não há sentido em utilizar caixas maiores do que 1, e os valores seriam realmente 38% e 50%.

A outra característica interessante no método é que a remoção pode ser feita sem problemas, simplesmente "apagando" a entrada, desde que se faça a busca sem sucesso até o limite.

O método tem importância para nós, pois contribuiu para alguns dos algoritmos desenvolvidos no capítulo IV.

III.4. ANÁLISE COMPARATIVA DOS MÉTODOS

A comparação dos métodos para transformação de chaves em endereços foi feita na seção (III.1.5), quando tratamos o assunto e aqui analisaremos somente os métodos para resolução de colisões apresentados nas seções (III.2) e (III.3).

Os gráficos das figuras (III.4-1) e (III.4-2) foram retirados de KNUTH[2] e representam as fórmulas para Cm e Cm' mostradas anteriormente junto à cada método com $n \to \infty$.

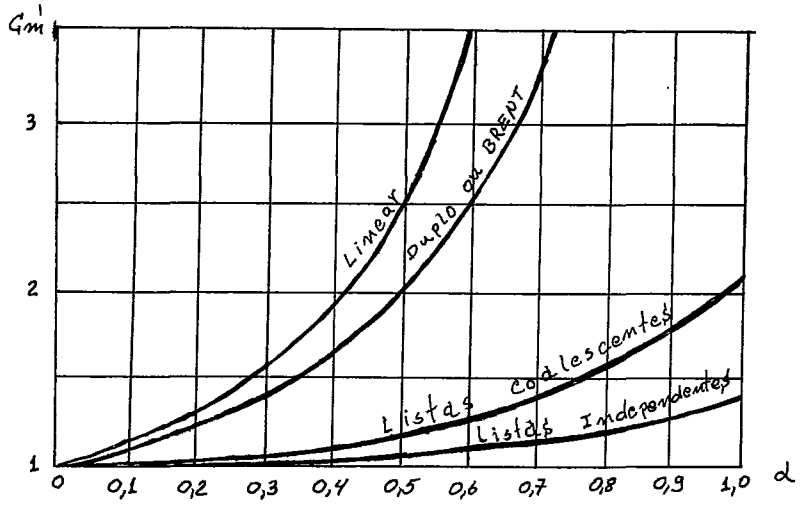


Figura III.4-1

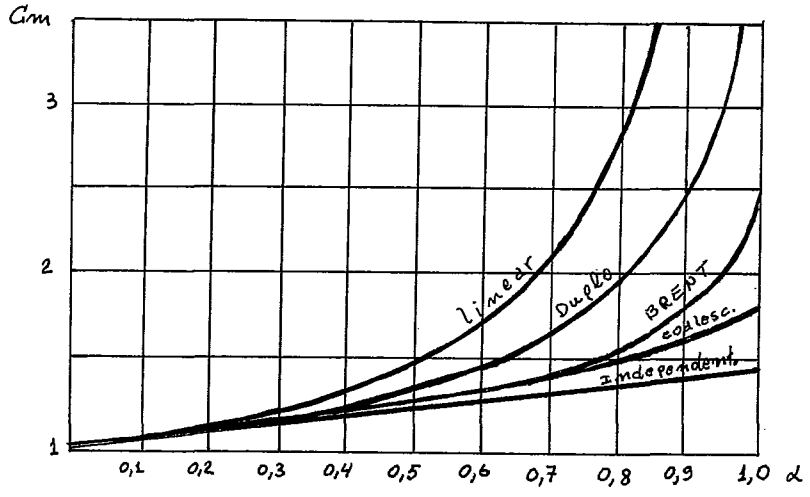


Figura III.4-2

Na tabela (III.4-1) vemos o número médio de comparações por chave para encher a tabela até determinada ocupação para os métodos de hash-duplo; e desvio padrão de 100 simulações para cada ocupação. Os valores serão de especial interêsse para comparação com os métodos do capítulo IV.

	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
μ	1,079	1,143	1,222	1,318	1,443	1,602	1,846	2,271	3,300	--
σ	0,032	0,029	0,033	0,036	0,046	0,060	0,083	0,177	0,330	--

Tabela III.4-1

Mas os números médios de comparações para inserir ou buscar são apenas algumas das características dos métodos, que dependendo da aplicação podem não ser os mais significantes na seleção deste ou daquele algoritmo.

Nas tabelas (III.4-2), (III.4-3) e (III.4-3) procuramos cruzar os métodos apresentados, com as possíveis características da aplicação que influam ou determinem a escolha do método a empregar.

OBSERVAÇÕES SOBRE AS TABELAS (III.4-2),(III.4-3) e (III.4-4)

a) Quando a característica requerida pela aplicação é quantitativa, como por exemplo: BAIXO TEMPO MÉDIO PARA BUSCAS COM SUCESSO, colocamos entre parênteses um peso (de zero a 10)que, grosseiramente indica a posição do método em relação aos demais. Naturalmente os valores podem variar dependendo da máquina, linguagem e detalhes de programação utilizados.

b) Os valores para os métodos de endereçamento aberto com hash limitado são para caixas de tamanho 1 como nos outros métodos.

As características do método nessas condições são péssimas, só tendo sido colocado na tabela para comparação com os métodos que tiveram-no como ponto de partida.

c) Vemos, que quanto as características determinantes da aplicação são quantitativas, fica difícil escolher pela tabela o método a empregar. A dificuldade maior é decidir entre encadeamento e endereçamento aberto, pois uma vez feita essa decisão, dentro de cada categoria, fica relativamenta mais fácil.

CARACT. DA APLICAÇÃO \ MÉTODO	Listas coalescen. na tabela	Listas independ. na tabela	Listas independ. area sep.	End.aberto visita linear	Hash duplo	Método de BRENT	End.aberto hash limitado
BAIXO TEMPO MÉDIO PARA UMA BUSCA COM SUCESSO	SIM (9)	SIM (10)	SIM (10)	SIM* (7) $\alpha<=0,7$	SIM (8) $\alpha\leq 0,9$	SIM (10) $\alpha\leq 0,9$	SIM (8) $\alpha\leq 0,4$
BAIXO TEMPO MÉDIO PARA UMA BUSCA SEM SUCESSO	SIM (9)	SIM (10)	SIM (10)	SIM (5) $\alpha\leq 0,6$	SIM (6) $\alpha\leq 0,7$	SIM (5) $\alpha\leq 0,7$	SIM (7) $\alpha\leq 0,4$
BAIXO TEMPO MÉDIO PARA UMA INSERÇÃO	SIM (10)	SIM (8)	SIM (10)	SIM (7) $\alpha\leq 0,7$	SIM (7) $\alpha\leq 0,7$	SIM (6) $\alpha\leq 0,7$	SIM (10) $\alpha\leq 0,4$
ARQUIVO ESTÁTICO EM TERMOS DE INSERÇÃO	SIM	SIM	SIM	SIM	SIM	SIM	SIM
ARQUIVO DINÂMICO EM TERMOS DE INSERÇÃO	SIM	SIM	SIM	SIM	SIM	SIM	SIM
ARQUIVO ESTÁTICO EM TERMOS DE REMOÇÃO	SIM	SIM	SIM	SIM	SIM	SIM	SIM

Tabela III.4-2

MÉTODO / CARACT. DA APLICAÇÃO	Listas coalescen. na tabela	Listas Independ. na tabela	Listas independ. área sep.	End.aberto visita linear	Hash duplo	Método de BRENT	End.aberto hash limitado
ARQUIVO DINÂMICO EM TERMOS DE REMOÇÃO	NÃO	SIM	SIM	SIM (7) $\alpha \leq 0,7$	NÃO	NÃO	SIM
LOCALIZAÇÃO DAS CHAVES EM MEMÓRIA INTERNA	SIM	SIM	SIM	SIM	SIM	SIM	SIM
LOCALIZAÇÃO DAS CHAVES EM MEMÓRIA EXTERNA	SIM	SIM	SIM	SIM	NÃO	NÃO	SIM
PROBABILIDADES DIFERENTES E CONHECIDAS a priori	Inserir em ordem decrescente de prob.	Inserir em ordem decrescente de prob.	Inserir em ordem decrescente de prob.	Inserir em ordem decrescente de prob.	Inserir em ordem decrescente de prob.	NÃO Aproveita	Inserir em ordem decrescente de prob.
PROBABILIDADES DIFERENTES MAS VARIÁVEIS COM O TEMPO	Não aproveita	Listas auto-organizáveis	Listas auto-organiz.	Não aproveita	Não aproveita	Não aproveita	Não aproveita
TEMPO DE RESPOSTA LIMITADO (PIOR CASO GARANTIDO E RAZOÁVEL)	NÃO	NÃO	NÃO	NÃO	NÃO	NÃO	SIM

Tabela III.4-3

MÉTODO / CARACT DA APLICAÇÃO	Listas coalesc. na tabela	Listas independ. na tabela	Listas Independ. área sep.	End.aberto visita linear	Hash duplo	Método de BRENT	End.aberto hash limitado
ARQUIVO ORDENADO	NÃO	NÃO	NÃO	NÃO	NÃO	NÃO	NÃO
ALOCAÇÃO DINÂMICA DE ESPAÇO	NÃO	NÃO	SIM mas à custa de eficiência $\alpha > 1$	NÃO	NÃO	NÃO	NÃO
SIMPLICIDADE DE PROGRAMAÇÃO	(8)	(7)	(7)	(10)	(10)	(5)	(9)
GRANDE APROVEITAMENTO DO ESPAÇO	SIM SE O APONTADOR FOR PEQUENO EM RELAÇÃO À CHAVE.			NÃO	NÃO	SIM	NÃO

Tabela III.4-4

III.5. DIAGNÓSTICO DE PROBLEMAS SEM ALGORITMOS EFICIENTES

A maior utilidade das tabelas (III.4-2),(III.4-3) e (III.4-4) é nos mostrar quais as características das aplicações que não são atendidas eficientemente ou nem mesmo são atendidas pelos algoritmos conhecidos. Se tomarmos algumas características como: remoção eficiente, grande aproveitamento de espaço e pior caso limitado, veremos que para cada uma delas há um ou mais algoritmos que atendem eficientemente, mas não há nenhum que atenda às tres de uma vez; e no entanto elas aparecem juntas em uma série de aplicações importantes (em tempo real por exemplo). Como resultado, o problema é resolvido por um método menos eficiente como árvores balanceadas por exemplo, e utilizando máquinas mais velozes do que se necessitaria se houvesse um método de hashing eficiente.

A seguir listaremos algumas características ou combinações de características para as quais gostaríamos de encontrar solução. Muitos dos problemas foram resolvidos e os resultados estão na seção IV. Dois problemas permeneceram sem solução: Ordenação e Alocação dinâmica de espaço. Para o primeiro acreditamos que não haja realmente solução sem um segundo arquivo de índice e para o segundo é mais provável que se possa obter solução melhor do que encadeamento em área separada com $\alpha > 1$, ou re-inserir todas as chaves em uma tabela maior.

CASO 1

a) Tabela estática (para inserção e remoção)

b) Memória Interna.

c) Aproveitamento de memória.

d) Baixo tempo médio para busca com sucesso.

e) Probabilidades de referência às chaves diferentes e conhecidas.

Essas características são encontradas em tabelas de menemonicos de montadores e de palavras chaves de compiladores. A escolha natural é o método de BRENT que atende às 4 primeiras características mas que simplesmente ignora uma informação importante que é o item e.

CASO 2

a) Baixo tempo médio para uma busca com sucesso.

b) Remoção eficiente.

c) Aproveitamento de memória.

d) Memória interna.

Essas características são encontradas em aplicações que funcionam em tempo real e o item b conflita com c se utilizarmos listas independentes.

CASO 3

a) Aproveitamento de memória.

b) Pior caso garantido.

c) Baixo tempo médio para buscas sem sucesso.

Em sistemas de tempo compartilhado e em sistemas de controle em tempo real, vemos frequentemente as características acima. O item a nos impede de utilizar o algoritmo de hash limitado, que, aliás foi desenvolvido para uma aplicação em tempo real, mas com grande desperdício de memória.

CASO 4

a) Máximo aproveitamento de memória.

b) Menor limite possível.

CASO 5

a) Arquivo ordenado

Resolver este problema tornaria praticamente obsoletos todos os outros algoritmos conhecidos que não utilizam hashing.

CASO 6

a) Alocação dinâmica de memória.

As duas únicas soluções conhecidas são:

1) Re-inserir todas as chaves em uma tabela maior.
2) Usar listas encadeadas em área separada, porém quando $\alpha >> 1$, o tempo de busca se torna alto.

IV. NOVOS ALGORITMOS PARA RESOLVER PROBLEMAS ESPECÍFICOS

IV.1. APRESENTAÇÃO - Nesse capítulo mostraremos as soluções encontradas para os casos 1, 2, 3 e 4, ficando sem solução eficiente somente os casos 5 (arquivo ordenado) e 6 (alocação dinâmica de espaço). Alguns algoritmos, além de resolver problemas apresentados na seção (III.5), ainda são ótimas alternativas para problemas onde já havia algoritmos eficientes. Eles se baseiam como todos os outros algoritmos de hashing em pequenos "ovos de colombo", como aliás foi a própria descoberta do hashing.

IV.2. CASO 1 - A idéia surgiu ao constatar-se que a melhor solução para dois problemas do desenvolvimento de um montador e de um compilador no programa de Sistemas e Computação, era o método de BRENT,que simplemente não aproveita as probabilidades de referência às chaves, na montagem da tabela. A idéia de inserir as chaves em ordem decrescente de probabilidade aqui não funciona pois o método muda as chaves de posição durante o processo de inserção.

O fato de que as probabilidades de referência as diversas chaves, no caso mnemonicos de instruções "assembler" e palavras chaves de uma linguagem tipo PL, eram diferentes, é evidente a toda pessoa que já programou, mas inicialmente surgiram as seguintes questões:

1) De que dependeriam essas probabilidade? Seriam uma consequência da própria estrutura da linguagem, do estilo do programador ou do tipo de aplicação que se está programando?

Em linguagem de baixo nível, acredita-se que o fator estilo de programação seja mais importante do que em linguagem de nível mais alto, pois o repertório de instruções diminui nessas, e elas aproximam-se mais da própria definição do problema a ser resolvido o que nos poderia levar a imaginar uma linguagem de altíssimo nível onde o programa seria a seguinte instrução: "Resolva o problema X", que não daria ao programador opção, sendo então a frequência de uso das instruções dependentes somente da estrutura da linguagem e da aplicação específica.

Por outro lado, em linguagem de baixo nível, o problema que está se resolvendo parace ter importância menor e GUILHERME CHAGAS RODRIGUES em um trabalho não publicado realizado no NCE/UFRJ sobre o "assembler" do terminal inteligente, observou que a frequência com que ocorrem certos agrupamentos de instruções, depende exclusivamente da linguagem em si e do estilo de programação, independendo da aplicação praticamente.Como esses resultados são para frequências de agrupamentos não indicam se a frequência de cada instrução realmente varia significativamente de programador para programador, mas indicam que não variam muito com as aplicações.

Fizemos uma amostragem sobre 8 programas escritos em "assembler" do computador MITRA-15, sendo 7 de um programador e um de outro, totalizando mais de 3.000 comandos. Poderíamos ter feito uma amostragem mais extensa e completa, mas esta já se mostrou suficientes para o que se propõe este trabalho.

Nas tabelas (IV.1-1) e (IV.1-2) mostramos os mnemonicos utilizados pelos dois programadores e suas respectivas frequências de utilização. No gráfico (IV.1-1) são apresentadas as frequências relativas para a amostra do programador 1, junto com as probabilidades dadas pela distribuição de ZIPF para o mesmo número de chaves (64).

ASSEMBLER DO MITRA-15 - PROGRAMADOR 1

Mnem.	Freq.Abs.	Freq.Rel.	Mnem.	Freq.Abs.	Freq.Rel.
LDA	396	0,1505	SRLS	12	0,0045
STA	234	0,0889	DCX	11	0,0041
RES	200	0,0760	MVS	11	0,0041
CMP	169	0,0642	SUB	11	0,0041
LDX	149	0,0566	CDS	7	0,0026
DATA	132	0,0501	END	7	0,0026
CLS	115	0,0437	SBL	7	0,0026
BRU	99	0,0376	WD	7	0,0026
BCF	94	0,0357	LBL	6	0,0022
FIN	80	0,0304	BAN	5	0,0019
CSV	78	0,0296	BAZ	5	0,0019
LES	61	0,0231	DIT	5	0,0019
XAX	52	0,0197	CNA	5	0,0019
RTS	49	0,0186	IOR	5	0,0019
ADD	45	0,0171	STE	5	0,0019
LPS	44	0,0167	SRLD	4	0,0015
ICX	43	0,0163	BRX	3	0,0011
LDE	41	0,0155	DIV	3	0,0011
ADM	38	0,0144	LDR	3	0,0011
BCT	35	0,0133	SLLD	3	0,0011
LBR	34	0,0129	EQU	2	0,0007
SBR	33	0,0125	XAA	2	0,0007
DST	32	0,0121	ACE	1	0,0004
TEXT	31	0,0117	BE	1	0,0004
LDS	29	0,0110	BND	1	0,0004
SLLS	29	0,0110	BNZ	1	0,0004
DLD	28	0,0106	BOF	1	0,0004
MUL	28	0,0106	DO	1	0,0004
STX	27	0,0102	LNE	1	0,0004
XAE	25	0,0095	RD	1	0,0004
BGE	23	0,0087	SLCD	1	0,0004
BL	18	0,0068	SRCD	1	0,0004

Tabela IV.1-1

ASSEMBLER DO MITRA-15 - PROGRAMADOR 2

Mnem.	Freq.Abs.	Freq.Rel.	Mnem.	Freq.Abs.	Freq.Rel.
LDA	61	0,1517	LBL	4	0,0099
STA	39	0,0970	BL	3	0,0074
DATA	32	0,0796	MVS	3	0,0074
RES	32	0,0796	XAE	3	0,0074
CMP	17	0,0422	STX	2	0,0049
FIN	17	0,0422	BNE	2	0,0049
LDX	15	0,0373	BCF	2	0,0049
CSV	15	0,0373	STE	2	0,0049
LEA	13	0,0323	SUB	2	0,0049
ADM	13	0,0323	ICX	2	0,0049
MUL	12	0,0298	BAN	1	0,0024
LDE	11	0,0273	SPV1	1	0,0024
RTS	10	0,0248	SLLS	1	0,0024
CLS	10	0,0248	SPVO	1	0,0024
ADD	8	0,0199	CDS	1	0,0024
BE	7	0,0174	SPV2	1	0,0024
LDS	7	0,0174	DO	1	0,0024
BRU	6	0,0149	END	1	0,0024
BCT	5	0,0124	DIV	1	0,0024
LBR	5	0,0124	SPV3	1	0,0024
TEXT	4	0,0099	DIT	1	0,0024
XAX	4	0,0099	EQU	1	0,0024
WD	4	0,0099			

Tabela IV.1-2

FREQUÊNCIAS RELATIVAS DA UTILIZAÇÃO DE MNEMONICOS

PROGRAMADOR 1 E DISTRIBUIÇÃO DE ZIPF

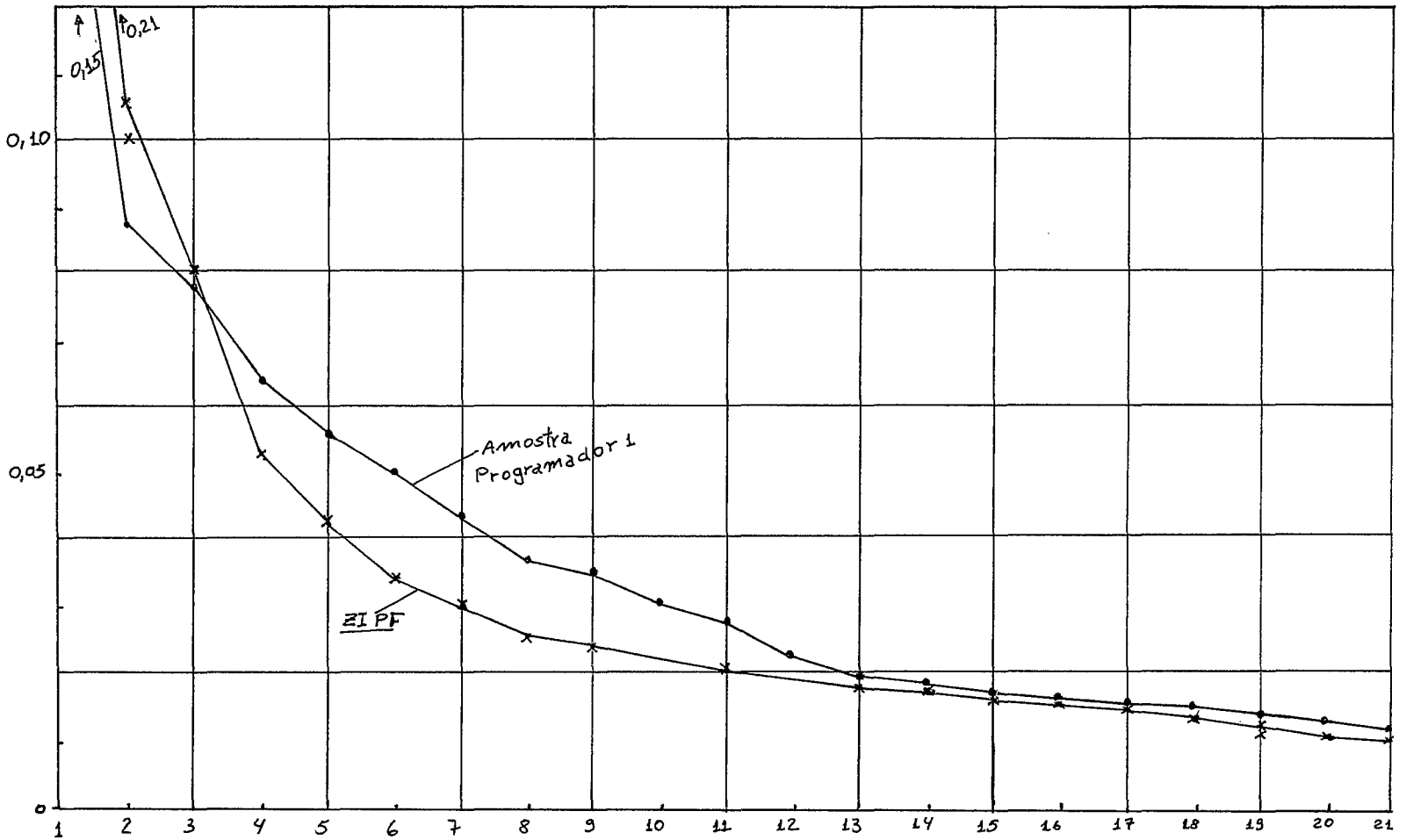


Gráfico IV.1-1

Observamos que realmente houve modificações nas posições relativas de instruções de uma amostra para outra, mas não diferiram muito das variações de um programa para outro do mesmo programador. Mas a conclusão importante que se tira dos gráficos é de que realmente há padrões bem definidos para a utilização de instruções que dependem principalmente das características da linguagem. Resultados semelhantes obteve HEHNER[29] que chegou a propor um novo tipo de computador que trabalhasse com campos de código de instrução, campos de endereço e variáveis com tamanho variável de bits, utilizando para representá-los um código de HUFFMAN, que nos daria para as instruções mais frequentes um tamanho menor em número de bits do que o das instruções menos usadas.

Uma vez concluido que há realmente uma distribuição de probabilidades, nos resta determinar se é possível obtê-la.

Quando se projeta um compilador ou montador para uma linguagem nova, não se pode "a priori" saber quais serão as probabilidades, e poderemos arbitrar uma distribuição uniforme ou outra qualquer, e depois que este já estiver implementado, poderemos ou amostrar uma grande quantidade de programas fonte ou monitorar a tabela de palavras chaves do compilador, e após um certo tempo gerar novamente a tabela utilizando um algoritmo que leve em conta essas probabilidades. Como não havia algoritmo que se aplicasse, desenvolvemos o seguinte:

Vimos na seção (III.3.8) como o método de BRENT, a cada inserção tenta minimizar o custo da tabela, mas considerando as chaves equiprováveis. A idéia aqui é tentar minimi-

zar o custo verdadeiro dado pela equação (III.3.8-1), que repetiremos abaixo por conveniência.

$$Cm = \sum_{i=1}^{m} p_i c_i \qquad \text{IV.1.1-1}$$

onde p_i é a probabilidade de referência à chave i e c_i é o número de comparações necessárias para encontrar a mesma.

Observamos que poucas modificações necessitam ser feitas no algoritmo de BRENT para se conseguir isso, e a alma do processo está em utilizar para decidir se deslocamos ou não uma chave para inserir outra em seu lugar e qual deslocamento é melhor, não o comprimento da carreira, e sim esse comprimento ponderado pela probabilidade da chave.

No exemplo do diagrama (IV.1.1-1), atribuimos probabilidades às chaves do diagrama (III.3.8-1). Essas são monstradas entre parênteses embaixo das chaves.

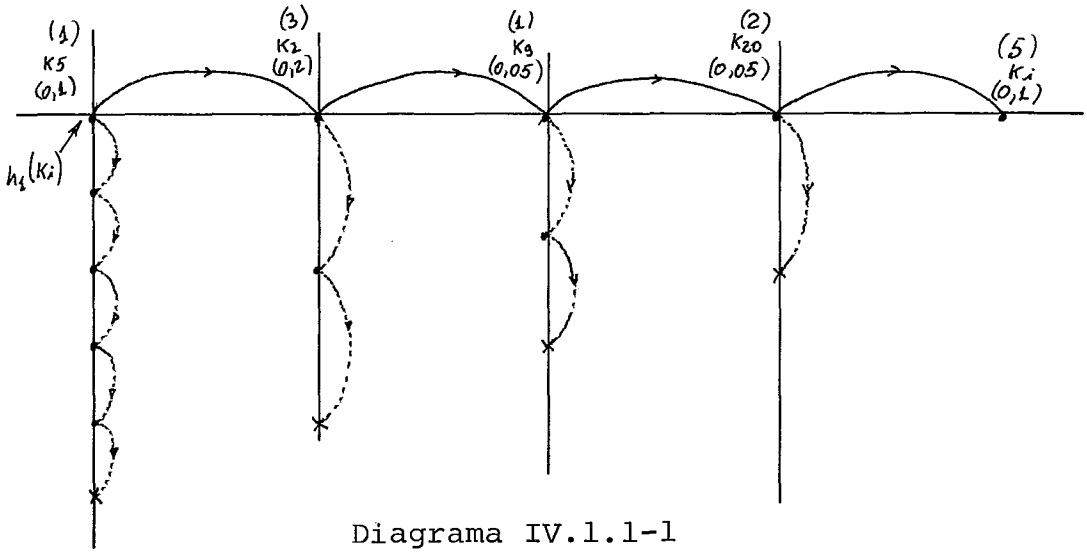


Diagrama IV.1.1-1

Se inserîssemos K_i como indicado no diagrama (IV.1.1.-1) terîamos pela equação (IV.1.1-1) um custo para a

porção da tabela igual a 1,35.

Se deslocássemos as chaves K_5, K_2, K_9 ou K_{20} para colocar K_i em seu lugar, teríamos os seguintes valores para o custo da tabela respectivamente: 1,35 , 1,45 , 1,25 e 1,30. O nosso algoritmo desloca a chave K_9, o que dá o menor custo possível para o custo da porção da tabela. Para comparação: o algoritmo de BRENT deslocaria a chave K_2, como vimos na seção (III.3.8) e nos daria 1,45 como custo.

Observamos que uma melhora significativa pode ocorrer se as chaves tiverem probabilidades bem diferentes, o que ocorre com os dados citados no começo dessa seção.

IV.2.1. - ALGORITMO 8 - BUSCA E INSERÇÃO COM MÍNIMO CUSTO MÉDIO

```
procedure ALG8 ( KX,PX ); value KX,PX;
begin comment Insere KX na tabela, reorganizando a msma se for inte-
          ressante. PX só necessita ser armazenado junto à KX em PROBI,
          enquanto houver inserções, sendo inútil na fase de busca.
  hash ( KX, R, Q );        %** "home-address" R e incremento Q.
  S:=0;  HS:=R;   TROCA:=false;
  while K(HS)≠VAGO do
    begin
      if K(HS)=KX then go to ENCONTRADO;
      if HS=R then go to TABELACHEIA;
      S:=S+1;
      HS:=(HS+Q) mod N;
    end;
  if S > 0                  %** Tentarei melhorar a tabela **
    then begin
           CLIM:=(S+L)*PX;
           LIMI:=S-1;
           for I:=0 step 1 while 1 ≤ LIMI do
           begin
             ENDI:=(R+I*Q) mod N;   %** End. da chave que se quer desl
             PI:=PROBI(ENDI)        %** Sua probabilidade.
             hash( K(ENDI),RI,QI);  %** Seu "home-address" e increm.
             HS2:=(ENDI+QI) mod N;  %** Seu primeiro salto.
             S2:=1;  CLIM:=CLIM-PX;
             while S2*PI < CLIM and K(HS2)≠VAGO do
               begin
                 S2:=S2+1;
                 HS2:=(HS2+QI) mod N;
               end;
             if S2*PI < CLIM
               then begin          %** Vale a pena trocar. **
                      COLOCAVELHO:=HS2;  COLOCANOVO:=ENDI;
                      CIM:=S2*PI;  %** Aperto o limite **
                      TROCA:=true;
                    end;
           end do loop I;
         end da tentativa de troca;
```

```
    if  TROCA  then  begin
                       K(COLOCAVELHO) :=K(COLOCANOVO) ;
                       PROBI(COLOCAVELHO) :=PROBI(COLOCANOVO) ;
                       K(COLOCANOVO) :=KX;
                       PROBI(COLOCANOVO) :=PX;
                     end
               else  begin
                       K(HS) :=KX;
                       PROBI(HS) :=PX;
                     end;
end.
```

OBSERVAÇÕES SOBRE O ALGORITMO 8

a) Agora é necessário verificar todas as chaves da carreira de K_i, não se podendo dispensar a penúltima nem parar quando se chega a um certo custo, pois a chave seguinte pode ter probabilidade muito baixa e dar uma melhor troca. Isso piora um pouco a eficiência do algoritmo na inserção em relação ao de BRENT, mas de uma forma insignificante para $\alpha < 0,9$ e, para $\alpha > 0,9$ a diferença no custo da tabela final gerada pelos dois algoritmos cresce tanto que o trabalho extra é largamente recompensado.

A limitação nas carreiras vertical continua igual só que agora poderada pelas probabilidades.

b) As probabilidades necessitam ser armazenadas na tabela junto às chaves, somente enquanto houver inserções sendo feitas, não sendo utilizadas na fase de busca.

c) Para evitar trocas com ganhos muito pequenos, ou devidos à

imprecisões aritméticas, podemos testar se o ganho é maior do que uma certa constante, e só assim fazer a troca.

d) Não é necessário que se use distribuição de probabilidades somente, pois o algoritmo funciona com pesos quaisquer que se dê às chaves (mesmo negativos), e que não somem 1.

e) Não deduzimos expressões teóricas para a performance do algoritmo, mas fizemos simulações intensivas com dois tipos de dados: aqueles mostrados no início da seção sobre o "assembler" do MITRA-15 e um outro conjunto que obedece à lei de ZIPF[49].

IV.2.2. TESTE DO ALGORITMO 8

Para obter valores sobre o desempenho do algoritmo, utilizamos dois tipos de dados:

a) Dados amostrados para o "assembler" do MITRA-15, que se justificam por si só por serem dados reais.

b) Dados aleatórios que obedecem à "lei de ZIPF" (distribuição hiperbólica) e que justificaremos a seguir.

G.K. ZIPF[49] observou que a n-ésima palavra mais comum em textos em lingua inglesa ocorre com frequência inversamente proporcional à n. Observou o mesmo fenômeno nas tabelas dos censos norte-americanos com a frequência com que ocorriam os nomes das cidades se elas fossem colocadas em ordem de crescente de população.

Diversos autores utilizaram para teste e projeto de algoritmos dados que obedecem à esta distribuição, KNUTH[2],

KNUTH[1] utilizaram para prever a performance de algoritmos processando textos, EASTON[4] a utilizou no estudo de "working-sets" pois ela se aproxima da distribuição de requisições de páginas, SCHUEGRAF[28] usou a distribuição em compactação de arquivos invertidos, onde naturalmente os dados a serem compactados são chaves secundárias ou primárias. Uma série de autores observou que para muitos arquivos comerciais vale a "lei dos 80-20", em que 80% das transações são com 20% do arquivo e também dentro desses 20% vale a lei, sendo 64% das transações com apenas 4% do arquivo e assim por diante. Esta distribuição é muito parecida com a lei de ZIPF e KNUTH[2] mostra que ambas podem ter a mesma equação, variando continuamente um parâmetro vamos de uma à outra.

Como justificativa final, constatamos que os dados do item a se aproximavam bastante desta distribuição, ver gráfico (IV.1-1).

A "lei de ZIPF" (distribuição hiperbólica) é a seguinte:

$$p_1 = c/1, \quad p_2 = c/2, \ldots, p_m = c/m \text{ , onde } c=1/H_m$$

e

$$H_m = 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{m} = \sum_{i=1}^{m} 1/i$$

A utilização da distribuição de ZIPF nos dá um meio de comparar os algoritmos com dados mais gerais e menos sujeitos à vicios, podendo ser realizados vários experimentos.

Foram gerados 100 conjuntos de chaves aleatórias uniformemente distribuidas no intervalo [1,131072] para ca-

da ocupação, e associadas à uma permutação aleatória da distribuição de ZIPF, e inseridas pelos dois algoritmos em tabelas de tamanho 1009.

Para os dados reais os tamanhos de tabela variaram de 67 a 653, com um número fixo de 64 chaves. O método de resolução de colisões utilizado foi o de divisão dupla.

Os resultados das simulações são apresentados nas tabelas (IV.2-1) a (IV.2-3) e gráficos (IV.2-2) e (IV.2-4).

CUSTO MÉDIO ESPERADO PARA UMA BUSCA COM SUCESSO

		10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
BRENT	Cm R	1,117	1,233	1,130	1,052	1,166	1,287	1,451	1,466	1,576	α = 0,95 1,840
	Cm Z	1,060	1,100	1,172	1,216	1,283	1,376	1,472	1,596	1,803	2,545
	σ	0,067	0,068	0,089	0,093	0,096	0,124	0,142	0,179	0,209	0,770
ALG8	Cm R	1,054	1,035	1,070	1,011	1,053	1,135	1,169	1,205	1,270	α = 0,95 1,255
	Cm Z	1,017	1,035	1,052	1,074	1,096	1,120	1,151	1,195	1,260	1,483
	σ	0,009	0,008	0,009	0,012	0,013	0,013	0,011	0,016	0,016	0,041

Tabela IV.2-1

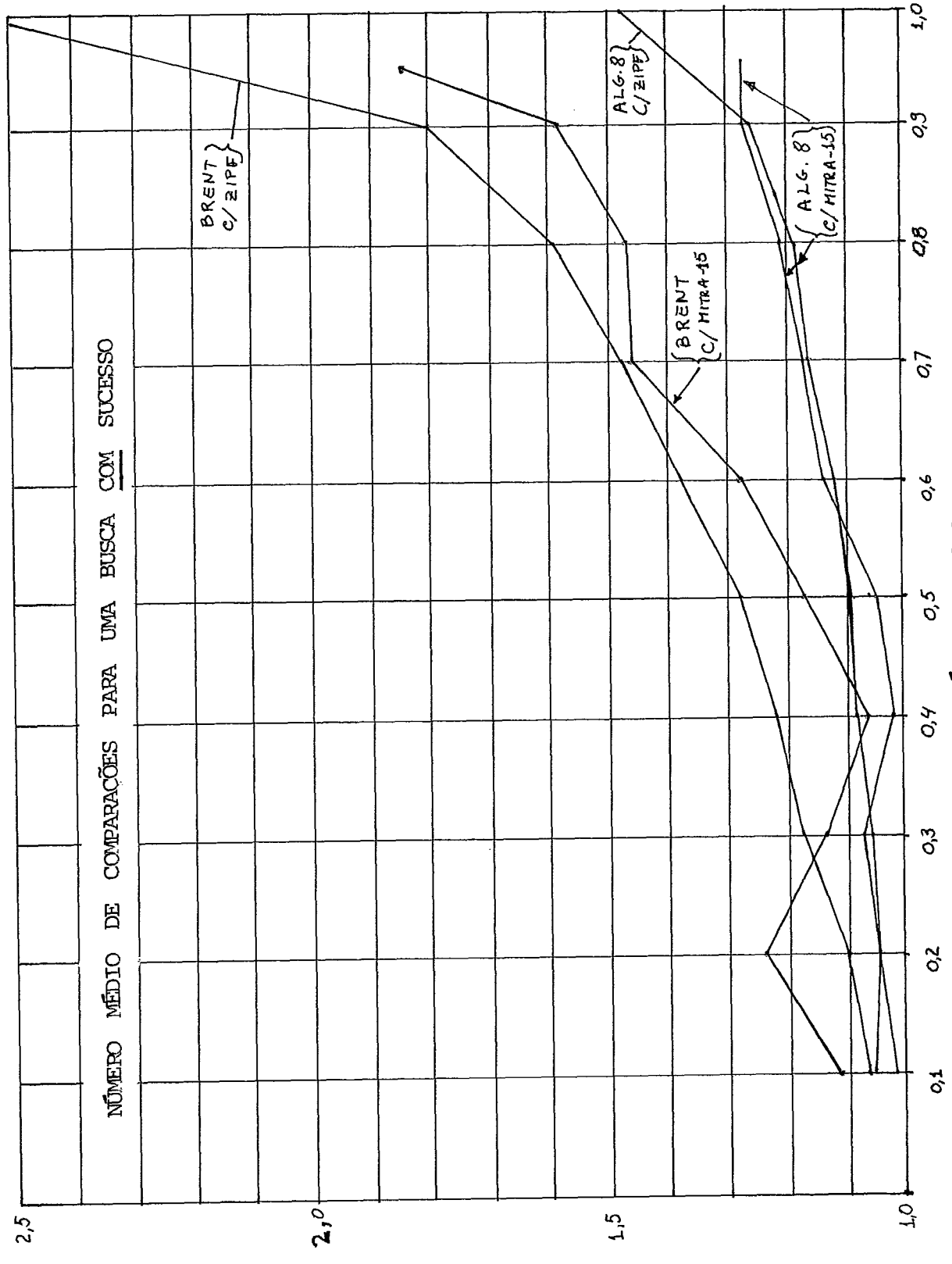


Gráfico IV.2-2

A discrepância entre os resultados com os dados reais e com os de ZIPF na última coluna da tabela é devido ao fato que para os dados reais a tabela não estava totalmente cheia (α = 0,955) e as tabelas tinham tamanho bem diferente,67 e 1009, pois o pior caso é bem diferente para os dois, e nesta faixa já ocorrem muitas carreiras de comprimento desta ordem.

Considerando que o menor valor para o custo seria 1, vemos que o custo extra para resolver as colisões, que é só o que pode ser diminuido, com o novo algoritmo é 3,1 vezes menor e a performance total é 41% melhor com α = 1.

NÚMERO MÉDIO DE COMPARAÇÕES POR CHAVE PARA INSERIR ATÉ CADA OCUPAÇÃO

		10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
BRENT	Cm R	1,09	1,13	1,13	1,11	1,36	1,77	1,84	2,23	3,25	α = 0,95 5,72
	Cm Z	1,053	1,129	1,221	1,344	1,508	1,734	2,078	2,639	3,838	19,40
	σ	0,025	0,027	0,031	0,042	0,047	0,061	0,078	0,110	0,170	3,169
ALG 8	Cm R	1,19	1,23	1,23	1,25	1,70	2,16	2,27	3,03	4,17	α = 0,95 7,92
	Cm Z	1,102	1,237	1,386	1,595	1,840	2,177	2,667	3,449	5,026	23,26
	σ	0,049	0,043	0,050	0,058	0,065	0,094	0,106	0,166	0,236	3,167

Tabela IV.2-2

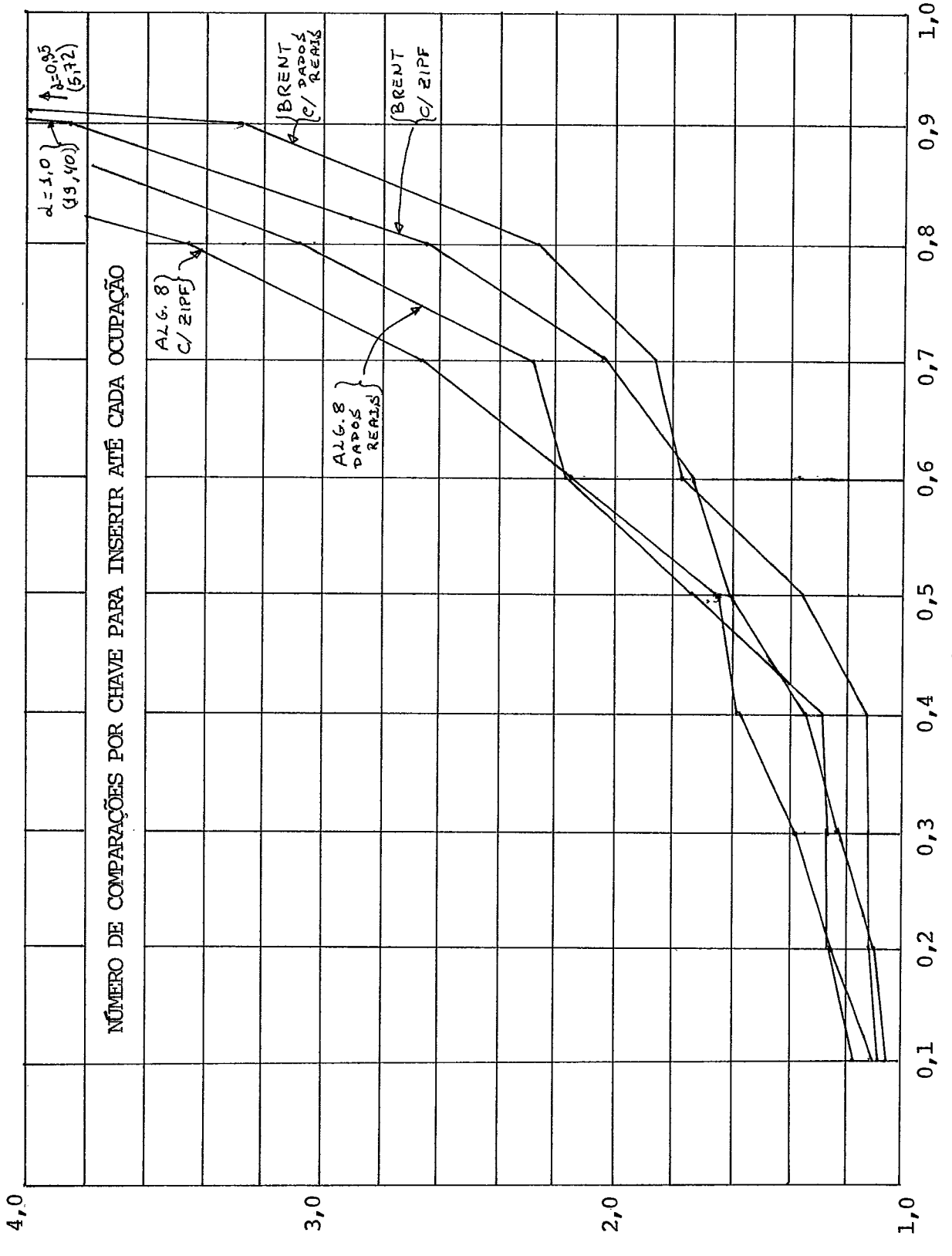


Gráfico IV.2-3

O aumento do número de comparações para inserir no alg. 3 não foi muito grande, exceto para $\alpha = 1$, mas que é plenamento justificado pela melhoria em Cm (para a maioria das aplicações). Ele se deve a dois fatores:

a) Todas as chaves na carreira de K_i são verificadas.

b) As carreiras tem maior média de comprimento, pois as chaves de baixa probabilidade são deixadas com carreiras mais compridas.

TEMPO MÉDIO DE PROCESSADOR POR CHAVE PARA INSERIR ATÉ CADA OCUPAÇÃO

As medidas de tempo são de pouca valia para decidir uma implementação pois dependem muito da máquina utilizada, da linguagem e do programador. Procurou-se, entretanto programá-los da melhor maneira, pelo mesmo programador, mesma máquina, inserções alternadas em um método e outro de modo a igualar ao máximo as condições ambientes, mas devido ao grande aparato de monitoração implementado (utilizando vários arranjos tri-dimensionais, etc...), introduziu-se grande "overhead", que pode distorçer um pouco os resultados, não se fez grande esforço em reduzir-se este, pois não consideramos as medidas de tempo o fator mais importante a ser monitorado. Nota-se certas descontinuidades nas medidas, que foram atribuidas ao sistema de memória virtual do B6700. A unidade de tempo é 2,4 microsegundos.

		10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
BRENT	R	965	221	230	225	252	318	294	273	382	$\alpha = 0,95$ 1120
	Z	307	301	272	282	291	331	344	365	407	891
	σ	27	17	14	12	7	21	19	13	15	98
ALG 8	R	262	266	234	236	286	389	309	375	470	$\alpha = 0,95$ 810
	Z	321	332	307	334	362	430	472	536	645	1780
	σ	26	21	18	13	11	29	29	22	30	187

Tabela IV.2-3

CONCLUSÕES SOBRE O ALGORITMO 8

Os resultados foram considerados satisfatórios, resolvendo adequadamente o problema proposto, e sendo recomendável inclusive para tabelas não estáticas, dede que se possa ponderar as chaves.

Exemplo: Em uma tabela de símbolos onde as palavras reservadas fiquem junto com identificadores definidos pelo usuário, poderíamos atribuir um peso alto à algumas palavras chaves e um peso baixo constante para todos os identificadores do usuário,de forma que ele competiriam entre si e com as palavras chaves menos frequentes, não deslocando do "home-address" as palavras chaves mais importantes. Para isso precisaríamos de apenas um bit a mais por entrada, com mais bits poderíamos refinar o método. Outra idéia seria uti

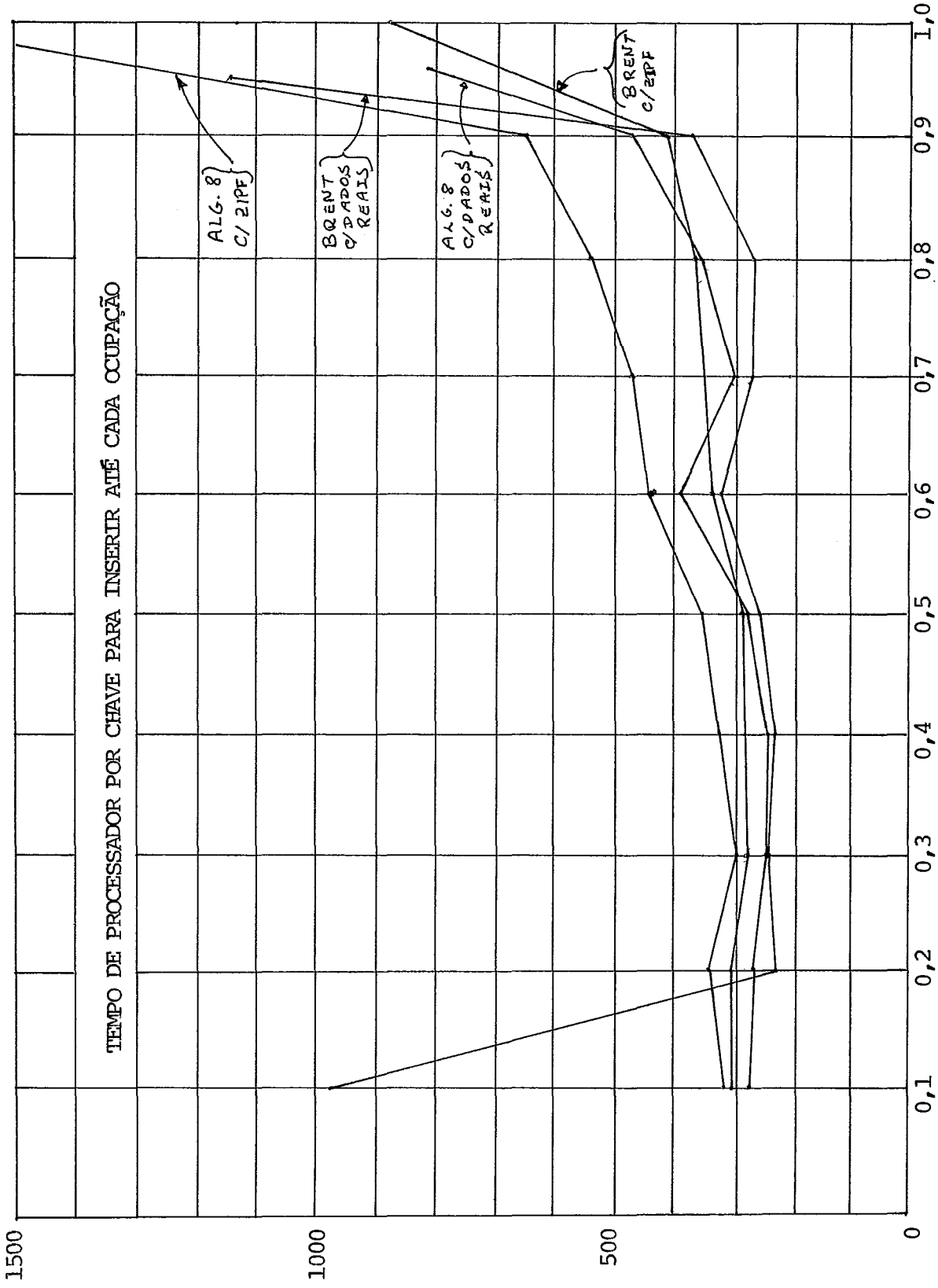


Gráfico IV.2-4

lizar o próprio tipo de identificador como pivô, podendo-se supor que arranjos sejam mais referenciados do que variáveis simples, etc.... Uma idéia sugerida por Carlos Flores Cunha mas ainda não comprovada experimentalmente, seria de correlacionar a frequência de utilização de um determinado identificador com o inverso do seu tamanho.

IV.3. CASO 2

IV.3.1. APRESENTAÇÃO - Durante o terceiro período de 1976 ao lecionar a cadeira de Busca em Arquivos na COPPE fui questionado pelos alunos quanto ao problema da impossibilidade de remoção na grande maioria dos métodos e após pesquisar a bibliografia disponível a resposta continuou a mesma: listas encadeadas independentes ou visita linear com o algoritmo 5 para remoção, que é bastante ineficiente para ocupações não muito baixas. Para hash duplo e o algoritmo de BRENT só restava a solução de colocar um sinal indicando que a entrada estava vaga para inserção, com os inconvenientes já citados. O artigo de CLAPSON[12] publicado em março de 1977 apresenta a idéia de hash limitado, descrito na seção (III.3.9). A idéia básica deste método é muito simples, mas tem um inconveniente que é o baixo aproveitamento de memória. CLAPSON[12] acrescentou à idéia original dois aperfeiçoamentos que permitiram elevar a ocupação máxima garantida até certos valores que ele considerou aceitáveis, e implementou o algoritmo à nível de sistema operacional em um sistema de processamento

distribuido da IBM, que está sendo comercializado. Embora o autor não cite qual a máquina, tudo leva a crer que seja o sistema /34.

Como decorrência lógica do nosso problema surgiu a idéia de fazer o hashing limitado, mas rearranjando as chaves como no método de BRENT, o que nos daria as seguintes vantagens: remoção trivial, número de comparações limitado para buscas com sucesso e sem sucesso, número de comparações para buscas com sucesso muito baixo e boa ocupação de memória.

IV.3.2. ALGORITMOS COM CARREIRA COMPLETA

O primeiro problema para aplicação do algoritmo de BRENT com hash limitado é o fato de que não é analisado o passado das chaves, olhando-se somente para as carreiras das posições onde estão as chaves em diante. Fizemos uma pequena modificação de modo que o comprimento da carreira de cada chave considerado para decisão fosse contado a partir do "home-address" da chave. Com essa modificação, algumas trocas que resultariam em diminuição do custo da tabela, não são feitas para não aumentar uma carreira já longa.

Com essa modificação chegamos ao algoritmo 9 e fazendo a mesma modificação no algoritmo 8 apresentado na seção anterior, se obteve o algoritmo 10 .

Fizemos os mesmos testes aplicados ao algoritmo 8 para determinar se a modificação acima, indispensável ao uso com hash limitado, representaria uma queda sensível na qualidade das tabelas em termos de buscas, e também para observar se era verdadeira a conjectura de que a variância dos comprimentos das carreiras diminuiria com a modificação, o que seria ótimo para utilização junto com o hashing limitado.

Na seção seguinte apresentaremos de uma forma mais detalhada o método e apresentaremos o algoritmo, na outra mostraremos os resultados das simulações para o algoritmo 9 . Nas seções (IV.3.2-3) e (IV.3.2-4) apresentaremos o mesmo para o algoritmo 10 .

IV.3.2.1. DECISÃO PELO COMPRIMENTO DA CARREIRA

A diferença em relação ao algoritmo 6 (BRENT) pode ser visualizada na figura (IV.3.2.1-1) em comparação com a figura (III.3.8-1). As carreiras "verticais" sempre começam no "home-address" da chave, que pode estar acima do eixo "horizontal", que é onde as chaves estão armazenadas.

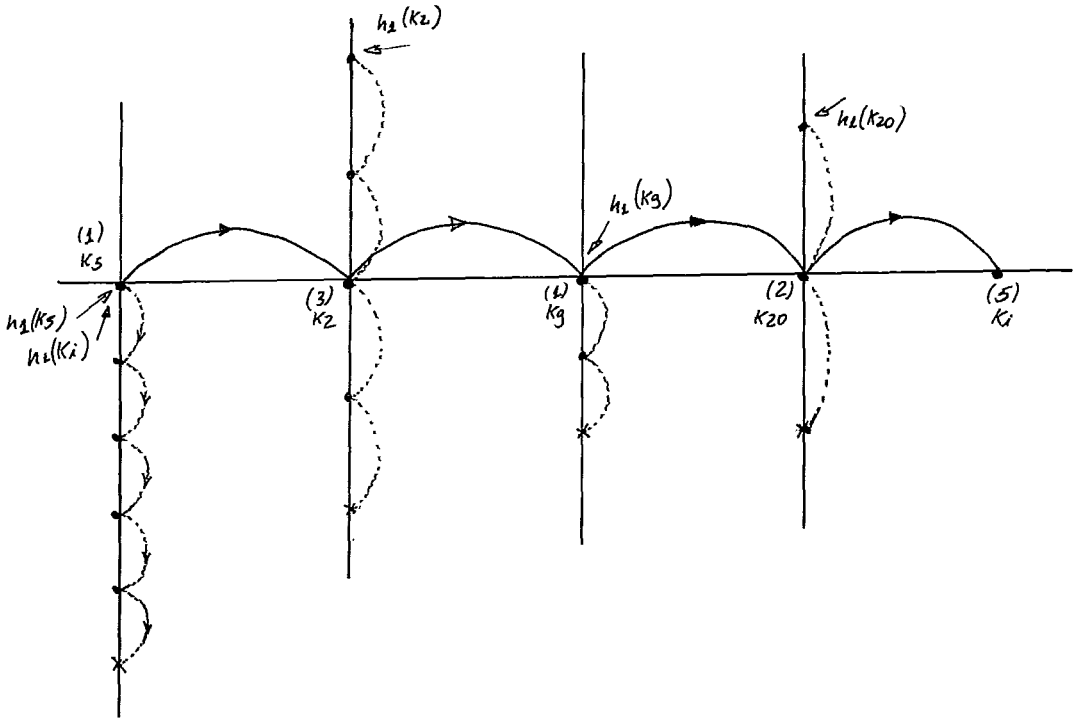


Figura IV.3.2.1-1

ALGORITMO 9 - BUSCA E INSERÇÃO COM REARRANJO E CARREIRA COMPLETA - DECISÃO PELO COMPRIMENTO DA CARREIRA

```
procedure  ALG9 ( KX ); value KX;
begin   comment  Insere KX na tabela, reorganizando a mesma se for possí-
        vel, e, se não acarretar o aumento de uma carreira já comprida;
  hash ( KX, R, Q );     %** "home-address" R e incremento Q. **
  S:=0;  HS:=R;  TROCA:=false;
  while  K(HS)≠VAGO  do
   begin
     if K(HS)=KX  then  go  to  ENCONTRADO;
     if HS=R  then  go  to  TABELACHEIA;
     S:=S+1;
     HS:=(HS+Q) mod N;
   end
  if  S > 1                %** Vou tentar melhorar a tabela. **
   then begin       LIMI:= S - 2;
     for  I:=0  step 1  while  I ≤ LIMI do
      begin
        ENDI:=(R+I*Q) mod N;   %** End. da chave que se quer des
        hash( K(ENDI), RI, QI);%** Seu "home-address" e increm.
        HS2:=(RI+QI) mod N;    %** Seu primeiro salto.
        S2:=1;  LIM:=LIMI-I+2;
        while  S2 < LIM  and  K(HS2)≠VAGO  do
         begin
           S2:=S2 + 1;
           HS2:=(HS2 + QI) mod N;
         end;
        if  S2 < LIM
         then begin    %** Vale a pena deslocar K(ENDI). **
                COLOCAVELHO:=HS2;   COLOCANOVO:=ENDI;
                LIMI:=S2+I-2;   %** Aperto o limite. **
                TROCA:=true;
              end;
      end  do loop I;
   end da tentativa de troca;
```

```
    if   TROCA  then  begin
                        K(COLOCAVELHO):=K(COLOCANOVO);
                        K(COLOCANOVO):=KX;
                      end
                else  K(HS):=KX;      %**Inserção normal.**
end da procedure ALG9;
```

TESTES DO ALGORITMO 9

NÚMERO MÉDIO ESPERADO DE COMPARAÇÕES PARA UMA BUSCA COM SUCESSO

BRENT	Cm	1,047	1,100	1,153	1,213	1,284	1,362	1,462	1,593	1,797	2,433
	σ	0,022	0,017	0,018	0,018	0,019	0,018	0,023	0,024	0,031	0,052
ALG 9	Cm	1,047	1,100	1,155	1,217	1,291	1,374	1,478	1,614	1,824	2,463
	σ	0,022	0,017	0,019	0,019	0,021	0,020	0,024	0,024	0,031	0,055

Tabela IV.3.2.1-1

VARIÂNCIA DOS COMPRIMENTOS DAS CARREIRAS

		10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
BRENTALG 9	$\bar{s}$	0,046	0,096	0,151	0,214	0,299	0,415	0,582	0,857	1,462	8,303
	σ	0,020	0,019	0,020	0,026	0,028	0,037	0,053	0,075	0,140	1,934
	$\bar{s}$	0,046	0,096	0,150	0,213	0,297	0,408	0,564	0,821	1,382	8,09
	σ	0,020	0,018	0,021	0,026	0,029	0,033	0,045	0,057	0,107	2,016

Tabela IV.3.2.1-2

NÚMERO MÉDIO DE COMPARAÇÕES POR CHAVE PARA INSERIR ATÉ CADA α

		10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
BRENTALG 9	$\bar{s}$	1,053	1,129	1,221	1,344	1,508	1,734	2,078	2,639	3,838	19,40
	σ	0,025	0,027	0,031	0,042	0,047	0,061	0,078	0,110	0,170	3,169
	$\bar{s}$	1,053	1,129	1,225	1,351	1,523	1,765	2,130	2,742	4,038	20,08
	σ	0,025	0,027	0,033	0,044	0,050	0,068	0,078	0,111	0,190	3,027

Tabela IV.3.2.1-3

OBSERVAÇÕES SOBRE O ALGORITMO 9

a) Dois resultados eram pretendidos com o algoritmo 9:

1) Que o desempenho não fosse muito inferior ao do algoritmo 6 (Brent).

2) Que diminuisse a variância dos comprimentos das carreiras, dando uma distribuição mais uniforme a estas e diminuindo a probabilidade de se encontrar a carreira de K_i maior do que o limite e encontrar na carreira de K_i todas as chaves já com suas carreiras completas, que é quando não se pode realmente inserir a chave K_i na tabela com hash limitado.

b) Na tabela (IV.3.2.1-1) comparamos o número médio de comparações para uma busca com sucesso do algoritmo 9 com o algoritmo 6 (Brent). Observamos que a perda no desempenho para a busca devido à carreira completa foi insignificante.

c) Na tabela (IV.3.2.1-3) comparamos o desempenho para a inserção e notamos que a piora também é desprezível, sendo de 0,5% com $\alpha = 0,9$.

d) Na tabela (IV.3.2.1-2) comparamos a variância dos comprimentos das carreiras e vemos que ela diminuiu, mas não tanto quanto era esperado.

e) É interessante comparar os valores da tabela (IV.3.2.1-1) e da tabela (IV.3.2.1-3) com aquelas obtidas para hash duplo, gráfico (III.4-2) e tabela (II.4-1), constatando que houve uma melhora substancial na busca com um aumento pequeno no nú-

mero de referências à tabela para a inserção. É claro que a cada referência (na inserção) nos métodos que rearranjam está associado um esforço computacional maior.

IV.3.2.2. ALGORITMO COM REARRANJO CARREIRA COMPLETA E DECISÃO PELO CUSTO

A mesma modificação (carreira completa) que se fez no algoritmo 6 para se obter o algoritmo 9 foi feita no algoritmo 7 (rearranjo usando probabilidades), obtendo-se o algoritmo 10. Este é apresentado a seguir.

ALGORITMO 10 - BUSCA E INSERÇÃO COM REARRANJO EM CARREIRA COMPLETA E DECISÃO PELO CUSTO

```
procedure  ALG10 ( KX ); value  KX;
begin    comment  insere KX e PX na tabela, reorganizando-a se for inte-
                  ressante e não aumentar o custo de uma carreira já com
                  custo alto;
  hash( KX, R, Q);                %** "Home-address" R e incremento Q **
  S:=0;     HS:=R;        TROCA:=false;
  while  K(HS)≠VAGO do
   begin
     if K(HS)=KX then go to ENCONTRADO;
     if S=N-1 then go to TABELACHEIA;
     S:= S + 1;
     HS := (HS+Q) mod N;
   end;
  if S > 0
   then begin
          CLIM:= (S+1)*PX;   LIMI:=S-1;
          for  i:=0  step 1 while  I ≤ LIMI do
           begin
             ENDI:= (R+I*Q) mod N;
             PI:=PROBI(ENDI)
             hash( K(ENDI), RI, QI );
             HS2:= (RI + QI) mod N;
             S2:=1;  CLIM:=CLIM-PX;
             while  S2*PI < CLIM and  K(HS2)≠VAGO do
              begin
                S2:=S2 + 1;
                HS2:= (HS2 + QI) mod N;
              end;
             if   S2*PI < CLIM
               then begin          %** Vale a pena deslocar K(ENDI). **
                      COLOCAVELHO:=HS2;
                      COLOCANOVO:=ENDI;
                      CLIM:=S2*PI;          %** Aperto o limite. **
                      TROCA:=true;
                    end;
           end do loop I;
        end da tentativa de rearranjo;
```

```
if TROCA then begin
                  K(COLOCAVELHO):=K(COLOCANOVO);
                  PROBI(COLOCAVELHO):=PROBI(CLOCANOVO)
                  K(COLOCANOVO):=KX;
                  PROBI(COLOCANOVO):=PX;
              end
         else begin
                  K(HS):=KX;
                  PROBI(HS):=PX;
              end;
end da procedure ALG10;
```

TESTES DO ALGORITMO 10

NÚMERO MÉDIO ESPERADO DE COMPARAÇÕES PARA UMA BUSCA COM SUCESSO

PROBABILIDADE UNIFORME

		10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
BRENT	μ	1,047	1,100	1,153	1,213	1,284	1,362	1,462	1,593	1,797	2,433
	σ	0,022	0,017	0,018	0,018	0,019	0,018	0,023	0,024	0,031	0,052
ALG 10	μ	1,047	1,100	1,155	1,217	1,291	1,374	1,478	1,614	1,824	2,463
	σ	0,022	0,017	0,019	0,019	0,021	0,020	0,024	0,024	0,031	0,055

Tabela IV.3.2.2-1

CUSTO MÉDIO ESPERADO PARA UMA BUSCA COM SUCESSO

		10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
ALG 8	μ	1,017	1,035	1,052	1,074	1,096	1,120	1,151	1,195	1,260	1,483
	σ	0,009	0,008	0,009	0,012	0,013	0,013	0,011	0,016	0,016	0,041
ALG 10	μ	1,017	1,035	0,052	1,074	1,098	1,122	1,155	1,200	1,267	1,493
	σ	0,009	0,008	0,009	0,012	0,013	0,013	0,011	0,017	0,017	0,042

Tabela IV.3.2.2-2

NÚMERO MÉDIO DE COMPARAÇÕES POR CHAVE PARA INSERIR ATÉ CADA OCUPAÇÃO

ALG 8	μ	1,102	1,237	1,386	1,695	1,840	2,177	2,667	3,449	5,026	23,26
	σ	0,049	0,043	0,050	0,058	0,065	0,094	0,106	0,166	0,236	3,167
ALG 10	μ	1,103	1,242	1,395	1,614	1,875	2,231	2,758	3,588	5,274	24,46
	σ	0,050	0,045	0,052	0,064	0,071	0,096	0,120	0,176	0,246	3,120

Tabela IV.3.2-3

OBSERVAÇÕES SOBRE O ALGORITMO 10

a) Nas tabelas (IV.3.2.2-1), (IV.3.2.2-2) e (IV.3.2.2-3) foram apresentados os resultados das simulações do algoritmo junto com os do algoritmo 8. Convém ressaltar que aqui, como também na seção anterior não se tentou obter um algoritmo que tivesse melhor desempenho, mas sim, que tivesse a propriedade da carreira completa (que será necessária para os próximos algoritmos), e que não fosse muito pior do que o algoritmo 8.

b) Vemos das mesmas tabelas que o custo subiu muito pouco, podendo ser considerado práticamente tão eficiente quanto o algoritmo 8.

IV.3.3. ALGORITMOS DE HASH LIMITADO COM REARRANJO

Uma vez que os algoritmos desenvolvidos na seção (IV.3.2), que atendem a nossa exigência de considerar a carreira completa para decidir a troca deram resultados razoáveis, resolvemos tomá-los como base para uma nova família de algoritmos que utilizam a idéia de hash limitado juntamente com a idéia de rearranjar a tabela.

SEM PROBABILIDADES (derivados do alg.9)

Alg.11 - Hashing limitado, rearranjando sempre que possível e procurando a melhor troca.

Alg.12 - Hashing limitado, rearranjando só quando houver problema na inserção (não for possível inserir dentro do limite) e procurando a melhor troca.

Alg.13 - Hashing limitado, rearranjando só quando houver problema porém se contantanto com a primeira troca.

COM PROBABILIDADES (derivados do alg.10)

Alg.14 - Idêntico ao alg.11 só que considerando as probabilidades.

Alg.15 - Idêntico ao algoritmo 12, mas com probabilidades.

Os algoritmos 11, 12 e 13 foram programados e testados, sendo mostrados a seguir, juntamente com seus testes. O algoritmo de hash limitado simples, devido a CLAPSON[12], utilizando hash-duplo ao invés de incremente binário o que certamente lhe melhora o desempenho, mas sem as caixas não endereçáveis, foi também programado e testado com o nome de algoritmo 7.

ALGUMAS OBSERVAÇÕES:

a) Não há sentido em se programar um algoritmo semelhante ao 13, mas utilizando as probabilidades na decisão, se ele toma a primeira solução encontrada.

b) Os algoritmo 14 e 15 devem se superpor em termos de resultados aos algoritmos 11 e 12 . Como a diferença no custo para inserir não deverá ser muito grande entre eles, e a diferença de custo para a busca deverá se afastar muito rapidamente quando o limite crescer, acreditamos que é razoável escolher-se o algoritmo 14 em qualquer situação. Diante disso, preferimos deixar de tratá-los de modo a poder analisar os demais em maior profundidade.

c) Em algumas situações haverá um conflito entre a melhoria do custo da tabela (que pode implicar em uma longa carreira para uma chave pouco provável), e a limitação máxima ao comprimento das carreiras. Esse conflito não invalida a aplicabilidade dos algoritmos desde que o limite não seja muito pequeno, pois ainda restará uma margem de manobra para que se consiga boas trocas.

ALGORITMO 11 - HASHING LIMITADO, REARRANJO CONSTANTE, MELHOR TROCA E DECISÃO PELO COMPRIMENTO DA CARREIRA

```
procedure  ALG11( KX, SLIM, CHEIO ); value KX, SLIM;
begin   comment  Busca KX na tabela, e se não for encontrado insere
         se for possível e rearranja se for necessário ou lucrativo.
         Após a primeira inserção frustrada : CHEIO:=true e a tabela
         é bloqueada à futuras inserções.
  if  CHEIO then go to FIM;
  hash(KX, R, Q);          %** "home-address" e incremento. **
  HS:=R;   S:=0;   HST:=ST:=-1;
  while S ≤ SLIM do
   begin
     if K(HS)=KX  then go to ENCONTRADO;
     if K(HS)=VAGO and HST=-1 then begin  %** Prim.vago. **
                                    HST:=HS;  ST:=S;
                                   end;
     S:=S + 1;    HS:=(HS + Q) mod N;
   end;
  if  HST≠-1then begin HS:=HST; S:=ST; end;  %** Prim.vago. **
  if S > 1
   then begin          %** Tento rearranjar a tabela. **
         if S > SLIM  then begin LIMI:=SLIM;LIM:=SLIM+1; end
                      else LIMI:S-2;
         for I:=0     step 1  while I ≤ LIMI do
          begin
           ENDI:= (R+I*Q) mod N;    %** End. da chave a deslocar. *
           hash( K(ENDI),RI,QI );   %** Seu "home-address" e increm
           HS2:=(RI + QI) mod N;    %** Seu prim.salto. **
           S2:=1;
           if TROCA or S ≤ SLIM then LIM:=LIMI-I+2;  %** aperto
           while S2 < LIM and K(HS2)≠ VAGO do
            begin
              S2:=S2 + 1;  HS2:= (HS2 + QI) mod N;
            end;
           if S2 < LIM
            then begin  COLOCAVELHO:=HS2; COLOCANOVO:=ENDI;
                        if S2+I-2 < LIMI then LIMI:=S2+I-2; %** aperto
                        TROCA:=true;
                 end;
```

```
                end do loop  I;
            end   da  tentativa de troca;
        if   TROCA  then  begin
                            K(COLOCAVELHO):=K(COLOCANOVO);
                            K(COLOCANOVO):=KX;
                          end
                     else if S > SLIM then CHEIO:=true  %** Inserção frustrada
                                        else K(HS):=KX;   %** Inserção normal **
        FIM:
end da procedure ALG11;
```

ALGORITMO 12 - HASHING LIMITADO, REARRANJO OCASIONAL, MELHOR TROCA E DECISÃO PELO COMPRIMENTO DA CARREIRA

```
procedure  ALG12( KX, SLIM, CHEIO );   value KX, SLIM;
begin   comment  Busca  KX na tabela, e se não for encontrado insere
          se for possível e rearranja só se for necessário. Após  a
          primeira inserção frustrada a tabela é bloqueada à futuras
          inserções;
 if  CHEIO  then  go to FIM;
 hash(KX, R, Q);
 HS:= R;   S:=0;   HST:=ST:= -1;
 while  S≤SLIM  do
  begin
    if  K(HS)=KX  then  go to  ENCONTRADO;
    if  K(HS)=VAGO  and HST= -1 then begin HST:=HS; ST:=S; end;
    S:= S + 1;     HS:= (HS + Q) mod N;
  end;
 if  HST≠ -1  then  begin  HS:=HST;  S:=ST;  end;  %** Prim. vago **
 if  S>SLIM
   then begin          %** Tento rearrumar a tabela p/ poder inserir. *
     LIMI:= SLIM;   LIM:= SLIM + 1;
     for I:=0  step 1  while  I≤LIMI  do
      begin
        ENDI:= (R+I*Q) mod N;
        hash( K(ENDI), RI, QI);
        HS2:= (RI + QI) mod N;  S:= 1;
        if  TROCA then LIM:= LIMI-I+2;   %** Aperto o limite. **
        while  S2<LIM  and  K(HS2)≠VAGO  do
         begin
           S2:= S2 + 1;  HS2:= (HS2 + QI) mod N;
         end;
        if  S2<LIM
          then begin
                 COLOCAVELHO:= HS2;  COLOCANOVO:= ENDI;
                 if S2+I-2<LIMI then  LIMI:= S2+I-2; %** Aperto
                 TROCA:= true;
               end;
      end  do  for  I;
```

```
        if TROCA then begin
                        K(COLOCAVELHO) := K(COLOCANOVO);
                        K(COLOCANOVO) := KX;
                      end
                 else CHEIO:= true;
      end da tentativa de rearrumar a tabela
    else K(HS) := KX;  %** Inserção normal **
  FIM:
end da procedure ALG12;
```

ALGORITMO 13 - HASHING LIMITADO, REARRANJO OCASIONAL, PRIMEIRA TROCA

```
procedure ALG13( KX, SLIM, CHEIO);      value KX, SLIM;
begin   comment Busca KX na tabela, e se não for encontrado insere se
          for possível e rearranja só se for necessário, não buscando
          a melhor troca. Após a primeira inserção frustrada, a tabela
          é bloqueada à futuras inserções;
  if CHEIO then go to FIM;
  hash( KX, R, Q );
  HS:= R;   S:=0;   HST:=ST:= -1;
  while S≤SLIM do
   begin
     if K(HS)=KX then go to ENCONTRADO;
     if K(HS)=VAGO and HST= -1 then begin HST:= HS; ST:=S; end;
     S:= S + 1 ;   HS:= ( HS + Q ) mod N;
   end;
  if HST≠ -1 then begin HS:=HST; S:= ST; end; %** Prim. vago.**
  if S>SLIM
    then begin
           LIMI:= SLIM;    LIM:= SLIM + 1;
           for I:=0 step 1 while I≤LIMI do
            begin
              ENDI:= ( R + I * Q ) mod N;
              hash( K(ENDI), RI, QI ) mod N;
              HS2:= (RI+QI) mod N;   S2:= 1;
              while S2<LIM and K(HS2)≠VAGO do
                begin S2:=S2+1; HS2:= (HS2+QI) mod N; end;
              if S2<LIM
                then begin
                       COLOCAVELHO:= HS2;  COLOCANOVO:= ENDI;
                       LIMI:= -1;  %** Se contenta e sai do loop **
                       TROCA:= true;
                     end;
            end do loop I;
           if TROCA then begin
                           K(COLOCAVELHO) := K(COLOCANOVO);
                           K(COLOCANOVO) := KX;
                         end
                    else CHEIO:= true;
         end da tentativa de rarranjar;
    else K(HS):=KX;
  FIM:
end da procedure ALG13;
```

ALGORITMO 7

NÚMERO MÉDIO ESPERADO DE COMPARAÇÕES PARA UMA BUSCA COM SUCESSO

LIMITE		10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
0	μ	–	–	–	–	–	–	–	–	–	–
	σ	–	–	–	–	–	–	–	–	–	–
1	μ	1,045*	1,090*	–	–	–	–	–	–	–	–
	σ	0,0232	0,0121	–	–	–	–	–	–	–	–
2	μ	1,049 *	1,1073*	1,1723*	–	–	–	–	–	–	–
	σ	0,0228	0,0188	0,0181	–	–	–	–	–	–	–
3	μ	1,0500	1,1117*	1,1788*	1,2450*	–	–	–	–	–	–
	σ	0,0233	0,0201	0,0208	0,0291	–	–	–	–	–	–
4	μ	1,0500	1,1127	1,1828*	1,2655*	1,3516*	–	–	–	–	–
	σ	0,0233	0,0202	0,0220	0,0268	0,0084	–	–	–	–	–
5	μ	1,0500	1,1127	1,1839*	1,2701	1,3696*	1,4628*	–	–	–	–
	σ	0,0233	0,0202	0,0226	0,0271	0,0332	–	–	–	–	–
6	μ	1,0500	1,1127	1,1842	1,2708*	1,3750*	1,5020	–	–	–	–
	σ	0,0233	0,0202	0,0226	0,0263	0,0306	0,0330	–	–	–	–
7	μ	1,0500	1,1127	1,1847	1,2747*	1,3768*	1,5056*	1,6495*	–	–	–
	σ	0,0233	0,0202	0,0231	0,0272	0,0297	0,3060	0,0001	–	–	–
8	μ	1,0500	1,1127	1,1847	1,2718*	1,3781*	1,5093*	1,6685*	–	–	–
	σ	0,0233	0,0202	0,0231	0,0271	0,0300	0,0356	0,0382	–	–	–
9	μ	1,0500	1,1127	1,1847	1,2717	1,3801*	1,5141*	1,6859*	–	–	–
	σ	0,0233	0,0202	0,0231	0,0269	0,0303	0,0360	0,0364	–	–	–
10	μ	1,050	1,1127	1,1847	1,2717	1,3809*	1,5150*	1,6947*	–	–	–
	σ	0,0233	0,0202	0,0231	0,0269	0,0306	0,0356	0,0499	–	–	–
15	μ	1,0500	1,1127	1,1847	1,2717	1,3828	1,5171*	1,7056*	1,9721*	–	–
	σ	0,0233	0,0202	0,0231	0,0269	0,0344	0,0357	0,0463	0,0559	–	–
50	μ	1,0500	1,1127	1,1847	1,2717	1,3812	1,5175	1,7082	2,0045	2,5291*	–
	σ	0,0233	0,0202	0,0231	0,0269	0,0304	0,0354	0,0455	0,0650	0,0936	–

Obs: 1) Um * sobre o número indica que nem todos os experim. atingiram a ocupação.

2) Os campos com um traço indicam que nenhum dos 100 experimentos atingiu a ocupação.

Tabela IV.3.3-1

ALGORITMO 13

NÚMERO MÉDIO ESPERADO DE COMPARAÇÕES PARA UMA BUSCA COM SUCESSO

LIMITE		10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
0	μ	-	-	-	-	-	-	-	-	-	-
	σ	-	-	-	-	-	-	-	-	-	-
1	μ	1,04 *	1,09 *	1,15*	1,20*	-	-	-	-	-	-
	σ	0,02	0,017	0,01	0,02	-	-	-	-	-	-
2	μ	1,04	1,10	1,17	1,24*	1,31*	1,39*	-	-	-	-
	σ	0,02	0,01	0,01	0,02	0,02	0,02	-	-	-	-
3	μ	1,05	1,11	1,18	1,25	1,34	1,44*	1,55*	1,68*	-	-
	σ	0,02	0,01	0,02	0,02	0,02	0,02	0,03	-	-	-
4	μ	1,05	1,11	1,18	1,26	1,35	1,46	1,59*	1,75*	-	-
	σ	0,02	0,02	0,02	0,02	0,02	0,02	0,03	0,03	-	-
5	μ	1,05	1,11	1,18	1,26	1,37	1,48	1,62	1,79*	2,01*	-
	σ	0,02	0,02	0,02	0,02	0,02	0,03	0,03	0,03	0,04	-
6	μ	1,05	1,11	1,18	1,27	1,37	1,49	1,64	1,83	2,07*	-
	σ	0,02	0,02	0,02	0,02	0,02	0,03	0,03	0,03	0,05	-
7	μ	1,05	1,11	1,18	1,27	1,37	1,50	1,66	1,87	2,12*	-
	σ	0,02	0,02	0,02	0,02	0,02	0,03	0,03	0,04	0,05	-
8	μ	1,05	1,11	1,18	1,27	1,37	1,51	1,67	1,89	2,16	-
	σ	0,02	0,02	0,02	0,02	0,02	0,03	0,04	0,04	0,05	-
9	μ	1,05	1,11	1,18	1,27	1,38	1,51	1,68	1,91	2,20	-
	σ	0,02	0,02	0,02	0,02	0,03	0,03	0,04	0,04	0,06	-
10	μ	1,05	1,11	1,18	1,27	1,38	1,51	1,69	1,93	2,23	-
	σ	0,02	0,02	0,02	0,02	0,03	0,03	0,04	0,05	0,06	-
15	μ	1,05	1,11	1,18	1,27	1,38	1,51	1,70	1,97	2,30	-
	σ	0,02	0,02	0,02	0,03	0,03	0,04	0,04	0,05	0,07	-
50	μ	1,05	1,11	1,18	1,27	1,38	1,51	1,70	2,00	2,52	4,04*
	σ	0,02	0,02	0,02	0,02	0,03	0,03	0,04	0,06	0,09	0,16

Obs: 1) Um * sobre o número indica que nem todos os experimentos atingiram a ocupação.
2) Os campos com um traço indicam que nenhum do 100 experimentos atingiu a ocupação.

Tabela IV.3.3-2

ALGORITMO 12

NÚMERO MÉDIO DE COMPARAÇÕES PARA UMA BUSCA COM SUCESSO

LIMITE		10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
0	μ	=	=	-	-	-	-	-	-	-	-
	σ	-	-	-	-	-	-	-	-	-	-
1	μ	1,0477*	1,0999*	1,1542*	1,2079*	-	-	-	-	-	-
	σ	0,0224	0,0177	0,0150	0,0282	-	-	-	-	-	-
2	μ	1,0496	1,1093	1,1723	1,2409*	1,3175*	1,3992*	-	-	-	-
	σ	0,0229	0,0183	0,0221	0,0215	0,0216	010223	-	-	-	-
3	μ	1,0500	1,1120	1,1808	1,2564	1,3441	1,4403	1,5535	-	-	-
	σ	0,0233	0,0197	0,0219	0,0246	0,0253	0,0247	0,0314	-	-	-
4	μ	1,0500	1,1127	1,1830	1,2653	1,3588	1,4673	1,5881*	1,7459 *	-	-
	σ	0,0233	0,0202	0,0225	0,0263	0,0267	0,0279	0,0324	0,0331	-	-
5	μ	1,0500	1,1127	1,1847	1,2649	1,3698	1,4861	1,6202	1,7858*	1,9976*	-
	σ	0,0233	0,0202	0,0227	0,0268	0,0304	0,0358	0,0351	0,0341	0,0490	
6	μ	1,050	1,1127	1,1845	1,2701	1,3758	1,4985	1,6441	1,8263	2,0504*	-
	σ	0,0233	0,0202	0,0228	0,0267	0,0295	0,0319	0,0367	0,0384	0,0532	-
7	μ	1,050	1,1127	1,1847	1,2713	1,3785	1,5070	1,6619	1,8593	2,0966*	-
	σ	0,0233	0,0202	0,0231	0,0271	0,0292	0,0329	0,0382	0,0428	0,0490	-
8	μ	1,050	1,1127	1,18475	1,2715	1,3792	1,5115	1,6754	1,8847	2,1366	-
	σ	0,0233	0,0202	0,0231	0,0271	0,0295	0,0341	0,0399	0,0450	0,0551	-
9	μ	1,050	1,1127	1,1847	1,2717	1,3804	1,5139	1,6845	1,9062	2,1732	-
	σ	0,0233	0,0202	0,0231	0,0269	0,0301	0,0347	0,0432	0,0477	0,0582	-
10	μ	1,050	1,1127	1,1847	1,2717	1,3809	1,5149	1,6914	1,9235	2,2052	-
	σ	0,0233	0,0202	0,0231	0,0269	0,0303	0,0350	0,0438	0,0504	0,0623	-
15	μ	1,050	1,1127	1,1847	1,2717	1,3812	1,5171	1,7052	1,9760	2,3392	-
	σ	0,0233	0,0202	0,0231	0,0269	0,0304	0,0353	0,0450	0,0558	0,0695	-
50	μ	1,0500	1,1127	1,1847	1,2717	1,3812	1,5175	1,7082	2,0045	2,5293	3,8579*
	σ	0,0233	0,0202	0,0231	0,0269	0,0304	0,0354	0,0455	0,0650	0,0930	0,1641

Tabela IV.3.3-3

ALGORITMO 11

NÚMERO MÉDIO DE COMPARAÇÕES PARA UMA BUSCA COM SUCESSO

LIMITE		10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
0	μ	-	-	-	-	-	-	-	-	-	-
	σ	-	-	-	-	-	-	-	-	-	-
1	μ	1,04*	1,09*	1,15*	1,20*	-	-	-	-	-	-
	σ	0,02	0,01	0,01	0,02	-	-	-	-	-	-
2	μ	1,04	1,10	1,15	1,21*	1,29*	1,37*	-	-	-	-
	σ	0,02	0,01	0,01	0,02	0,02	0,02	-	-	-	-
3	μ	1,04	1,10	1,15	1,21	1,29*	1,38*	1,49*	1,62*	-	-
	σ	0,02	0,01	0,01	0,02	0,02	0,02	0,02	0,01	-	-
4	μ	1,04	1,10	1,15	1,21	1,29	1,38	1,49	1,63	-	-
	σ	0,02	0,01	0,01	0,02	0,02	0,02	0,02	0,02	-	-
5	μ	1,04	1,10	1,15	1,21	1,29	1,38	1,49	1,63	1,86*	-
	σ	0,02	0,01	0,01	0,02	0,02	0,02	0,02	0,02	0,02	-
6	μ	1,04	1,10	1,15	1,21	1,29	1,38	1,49	1,63	1,86*	-
	σ	0,02	0,01	0,01	0,02	0,02	0,02	0,02	0,02	0,03	-
7	μ	1,04	1,10	1,15	1,21	1,29	1,38	1,49	1,63	1,86*	-
	σ	0,02	0,01	0,01	0,02	0,02	0,02	0,02	0,02	0,03	-
8	μ	1,04	1,10	1,15	1,21	1,29	1,38	1,49	1,63	1,86*	-
	σ	0,02	0,01	0,01	0,02	0,02	0,02	0,02	0,02	0,03	-
9	μ	1,04	1,10	1,15	1,21	1,29	1,38	1,49	1,63	1,86	-
	σ	0,02	0,01	0,01	0,02	0,02	0,02	0,02	0,02	0,03	-
10	μ	1,04	1,10	1,15	1,21	1,29	1,38	1,49	1,63	1,86	-
	σ	0,02	0,01	0,01	0,02	0,02	0,02	0,02	0,02	0,03	-
15	μ	1,04	1,10	1,15	1,21	1,29	1,38	1,49	1,63	1,86	-
	σ	0,02	0,01	0,01	0,02	0,02	0,02	0,02	0,02	0,03	-
50	μ	1,04	1,10	1,15	1,21	1,29	1,38	1,49	1,63	1,86	2,54*
	σ	0,02	0,01	0,01	0,02	0,02	0,02	0,02	0,02	0,03	0,06

Tabela IV.3.3-4

OCUPAÇÃO MÁXIMA ATINGIDA

LIMITE		7	13	12	11
0	μ	0,04	0,04	0,04	0,04
	σ	0,02	0,02	0,02	0,02
1	μ	0,13	0,28	0,28	0,28
	σ	0,04	0,05	0,05	0,05
2	μ	0,22	0,53	0,53	0,53
	σ	0,05	0,06	0,06	0,06
3	μ	0,31	0,70	0,70	0,70
	σ	0,07	0,05	0,05	0,05
4	μ	0,39	0,81	0,81	0,81
	σ	0,07	0,03	0,03	0,04
5	μ	0,45	0,86	0,86	0,87
	σ	0,08	0,03	0,03	0,02
6	μ	0,51	0,90	0,90	0,90
	σ	0,08	0,02	0,02	0,02
7	μ	0,55	0,93	0,93	0,93
	σ	0,08	0,02	0,02	0,01
8	μ	0,59	0,95	0,95	0,95
	σ	0,07	0,01	0,01	0,01
9	μ	0,62	0,96	0,96	0,96
	σ	0,07	0,01	0,01	0,01
10	μ	0,65	0,97	0,97	0,97
	σ	0,06	0,01	0,01	0,01
15	μ	0,76	0,99	0,99	0,99
	σ	0,05	0,00	0,00	0,00
50	μ	0,93	0,999	0,999	0,999
	σ	0,02	0,000	0,000	0,000

Tabela IV.3.3-5

ALGORITMO 7

NÚMERO DE COMPARAÇÕES POR CHAVE PARA INSERIR ATÉ CADA OCUPAÇÃO

LIMITE		10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
0	μ	–	–	–	–	–	–	–	–	–	–
	σ	–	–	–	–	–	–	–	–	–	–
1	μ	2,00*	2,00*	–	–	–	–	–	–	–	–
	σ	0,02	0,02	–	–	–	–	–	–	–	–
2	μ	3,00*	3,00*	3,00*	–	–	–	–	–	–	–
	σ	0,02	0,02	0,02	–	–	–	–	–	–	–
3	μ	4,00	4,00*	4,00*	4,00*	–	–	–	–	–	–
	σ	0,02	0,02	0,02	0,02	–	–	–	–	–	–
4	μ	5,00	5,00	5,00*	5,00*	5,00*	–	–	–	–	–
	σ	0,02	0,02	0,02	0,02	0,00*	–	–	–	–	–
5	μ	6,00	6,00*	6,00*	6,00*	6,00*	6,00*	–	–	–	–
	σ	0,02	0,02	0,02	0,02	0,03	–	–	–	–	–
6	μ	7,00	7,00*	7,00*	7,00*	7,00*	7,00*	–	–	–	–
	σ	0,02	0,02	0,02	0,02	0,03	0,03	–	–	–	–
7	μ	8,00	8,00	8,00	8,00	8,00*	8,00*	8,00*	–	–	–
	σ	0,02	0,02	0,02	0,02	0,02	0,03	0,00	–	–	–
8	μ	9,00	9,00	9,00	9,00	9,00*	9,00*	9,00*	–	–	–
	σ	0,02	0,02	0,02	0,02	0,03	0,03	0,03	–	–	–
9	μ	10,00	10,00	10,00	10,00	10,00*	10,00*	10,00*	–	–	–
	σ	0,02	0,02	0,02	0,02	0,03	0,03	0,04	–	–	–
10	μ	11,00	11,00	11,00	11,00	11,00*	11,00*	11,00*	–	–	–
	σ	0,02	0,02	0,02	0,02	0,03	0,03	0,04	–	–	–
15	μ	16,00	16,00	16,00	16,00	16,00*	16,00*	16,00*	16,00*	–	–
	σ	0,02	0,02	0,02	0,02	0,02	0,03	0,04	0,05	–	–
50	μ	51,00	51,00	51,00	51,00	51,00	51,00	51,00	51,00	51,00*	–
	σ	0,02	0,02	0,02	0,02	0,03	0,03	0,04	0,06	0,09	–

Tabela IV.3.3-6

ALGORITMO 13

NÚMERO DE COMPARAÇÕES POR CHAVE PARA INSERIR ATÉ CADA OCUPAÇÃO

LIMITE		10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
0	μ	–	–	–	–	–	–	–	–	–	–
	σ	–	–	–	–	–	–	–	–	–	–
1	μ	2,00*	2,01*	2,01*	2,02*	–	–	–	–	–	–
	σ	0,02	0,02	0,02	0,03	–	–	–	–	–	–
2	μ	3,00	3,00	3,00	3,01*	3,03*	3,04*	–	–	–	–
	σ	0,02	0,02	0,02	0,03	0,03	0,05	–	–	–	–
3	μ	4,00	4,00	4,00	4,00	4,02	4,03*	4,06*	4,19*	–	–
	σ	0,02	0,02	0,02	0,02	0,03	0,04	0,06	–	–	–
4	μ	5,00	5,00	5,00	5,00	5,00	5,00	5,03*	5,10*	–	–
	σ	0,02	0,02	0,02	0,02	0,03	0,04	0,06	0,08	–	–
5	μ	6,00	6,00	6,00	6,00	6,00	6,00	6,00	6,06*	6,15*	–
	σ	0,02	0,02	0,02	0,02	0,03	0,04	0,05	0,08	0,15	–
6	μ	7,00	7,00	7,00	7,00	7,00	7,00	7,00	7,03	7,08*	–
	σ	0,02	0,02	0,02	0,02	0,03	0,04	0,05	0,07	0,13	–
7	μ	8,00	8,00	8,00	8,00	8,00	8,00	8,00	8,00	8,05*	–
	σ	0,02	0,02	0,02	0,02	0,03	0,04	0,05	0,07	0,12	–
8	μ	9,00	9,00	9,00	9,00	9,00	9,00	9,00	9,00	9,02	–
	σ	0,02	0,02	0,02	0,02	0,03	0,04	0,05	0,07	0,12	–
9	μ	10,00	10,00	10,00	10,00	10,00	10,00	10,00	10,00	10,00	–
	σ	0,02	0,02	0,02	0,02	0,03	0,04	0,05	0,07	0,12	–
10	μ	11,00	11,00	11,00	11,00	11,00	11,00	11,00	11,00	11,00	–
	σ	0,02	0,02	0,02	0,02	0,03	0,04	0,05	0,07	0,12	–
15	μ	16,00	16,00	16,00	16,00	16,00	16,00	16,00	16,00	16,00	–
	σ	0,02	0,02	0,02	0,02	0,03	0,04	0,05	0,07	0,12	–
50	μ	51,00	51,00	51,00	51,00	51,00	51,00	51,00	51,00	51,00	–
	σ	0,02	0,02	0,02	0,02	0,03	0,04	0,05	0,07	0,12	–

Tabela IV.3.3-7

ALGORITMO 12

NÚMERO DE COMPARAÇÕES POR CHAVE PARA INSERIR ATÉ CADA OCUPAÇÃO

LIMITE		10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
0	μ	–	–	–	–	–	–	–	–	–	–
	σ	–	–	–	–	–	–	–	–	–	–
1	μ	2,00	2,01*	2,01*	2,02*	–	–	–	–	–	–
	σ	0,02	0,02	0,02	0,03	–	–	–	–	–	–
2	μ	3,00	3,00	3,00	3,01*	3,03*	3,04*	–	–	–	–
	σ	0,02	0,02	0,02	0,03	0,03	0,05	–	–	–	–
3	μ	4,00	4,00	4,00	4,00	4,01	4,04*	4,08*	–	–	–
	σ	0,02	0,02	0,02	0,02	0,03	0,04	0,06	–	–	–
4	μ	5,00	5,00	5,00	5,00	5,00	5,01*	5,05*	5,16*	–	–
	σ	0,02	0,02	0,02	0,02	0,03	0,04	0,06	0,09	–	–
5	μ	6,00	6,00	6,00	6,00	6,00	6,00	6,02	6,12*	6,29*	–
	σ	0,02	0,02	0,02	0,02	0,03	0,03	0,05	0,09	0,18	–
6	μ	7,00	7,00	7,00	7,00	7,00	7,00	7,00	7,08	7,26*	–
	σ	0,02	0,02	0,02	0,02	0,03	0,03	0,05	0,09	0,15	–
7	μ	8,00	8,00	8,00	8,00	8,00	8,00	8,00	8,05	8,23*	–
	σ	0,02	0,02	0,02	0,02	0,03	0,03	0,05	0,08	0,15	–
8	μ	9,00	9,00	9,00	9,00	9,00	9,00	9,00	9,03	9,17*	–
	σ	0,02	0,02	0,02	0,02	0,03	0,03	0,05	0,08	0,15	–
9	μ	10,00	10,00	10,00	10,00	10,00	10,00	10,00	10,02	10,14	–
	σ	0,02	0,02	0,02	0,02	0,03	0,03	0,05	0,07	0,14	–
10	μ	11,00	11,00	11,00	11,00	11,00	11,00	11,00	11,01	11,10	–
	σ	0,02	0,02	0,02	0,02	0,03	0,03	0,05	0,07	0,14	–
15	μ	16,00	16,00	16,00	16,00	16,00	16,00	16,00	16,00	16,01	–
	σ	0,02	0,02	0,02	0,02	0,03	0,03	0,05	0,06	0,13	–
50	μ	51,00	51,00	51,00	51,00	51,00	51,00	51,00	51,00	51,00	–
	σ	0,02	0,02	0,02	0,02	0,03	0,03	0,05	0,06	0,09	–

Tabela IV.3.3-8

ALGORITMO 11

NÚMERO DE COMPARAÇÕES POR CHAVE PARA INSERIR ATÉ CADA OCUPAÇÃO

LIMITE		10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
0	μ	–	–	–	–	–	–	–	–	–	–
	σ	–	–	–	–	–	–	–	–	–	–
1	μ	2,00*	2,01*	2,01*	2,02*	–	–	–	–	–	–
	σ	0,02	0,02	0,02	0,03	–	–	–	–	–	–
2	μ	3,00	3,01	3,03	3,06*	3,10*	3,15*	–	–	–	–
	σ	0,02	0,02	0,03	0,04	0,04	0,05	–	–	–	–
3	μ	4,00	4,01	4,03	4,06	4,12*	4,21*	4,32*	4,42*	–	–
	σ	0,02	0,02	0,02	0,02	0,04	0,06	0,06	0,08	–	–
4	μ	5,00	5,01	5,03	5,07	5,12	5,22	5,35*	5,56*	–	–
	σ	0,02	0,02	0,03	0,04	0,04	0,06	0,07	0,08	–	–
5	μ	6,00	6,01	6,03	6,07	6,13	6,22	6,37	6,61	6,96*	–
	σ	0,02	0,02	0,03	0,04	0,04	0,06	0,07	0,09	0,12	–
6	μ	7,00	7,01	7,03	7,07	7,13	7,23	7,38	7,64	8,05*	–
	σ	0,02	0,02	0,03	0,04	0,04	0,06	0,07	0,09	0,15	–
7	μ	8,00	8,01	8,03	8,07	8,13	8,23	8,39	8,66	9,13*	–
	σ	0,02	0,02	0,03	0,04	0,04	0,06	0,07	0,10	0,15	–
8	μ	9,00	9,01	9,03	9,07	9,13	9,23	9,39	9,67	10,17*	–
	σ	0,02	0,02	0,03	0,04	0,04	0,06	0,07	0,10	0,16	–
9	μ	10,00	10,01	10,03	10,07	10,13	10,23	10,39	10,68	11,22	–
	σ	0,02	0,02	0,03	0,04	0,04	0,06	0,07	0,10	0,16	–
10	μ	11,00	11,01	11,03	11,07	11,13	11,23	11,39	11,69	12,25	–
	σ	0,02	0,02	0,03	0,04	0,04	0,06	0,07	0,10	0,17	–
15	μ	16,00	16,01	16,03	16,07	16,13	16,23	16,40	16,70	17,34	–
	σ	0,02	0,02	0,03	0,04	0,04	0,06	0,07	0,10	0,18	–
50	μ	51,00	51,01	51,03	51,07	51,13	51,23	51,40	51,71	52,75	–
	σ	0,02	0,02	0,03	0,04	0,04	0,06	0,07	0,11	0,19	–

Tabela IV.3.3-9

OCUPAÇÃO MÁXIMA ATINGIDA PELOS ALGOTRITMOS 7, 11, 12 E 13

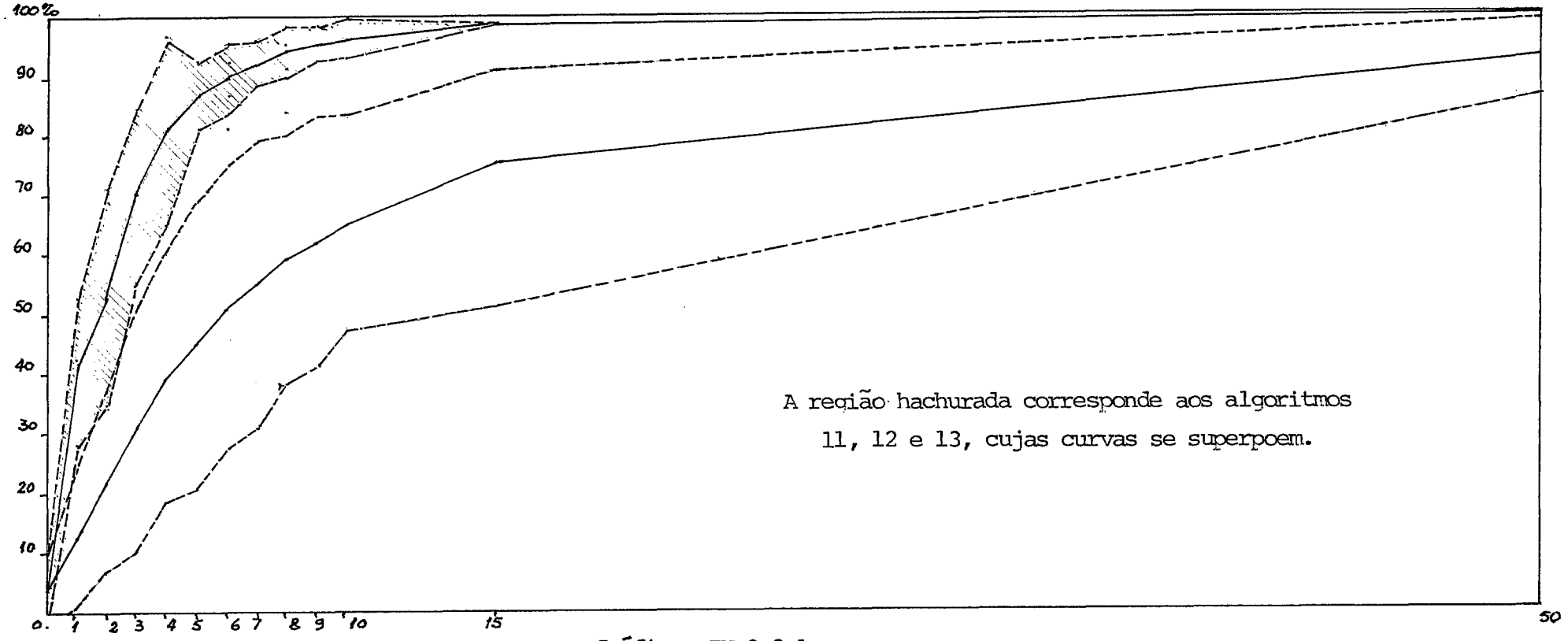


Gráfico IV.3.3-1

OBSERVAÇÕES SOBRE OS ALGORITMOS 7, 11, 12 E 13

a) Vemos das tabelas (IV.3.3-1) a (IV.3.3-5) que o algoritmo 7 (CLAPSON[12]) tem custo de busca muito maior do que os novos algoritmos e alcança um aproveitamento de espaço insatisfatório.

b) Em termos de ocupação máxima de memória, os algoritmos 11, 12 e 13 são praticamente equivalentes para limites altos, mas para limites menores do que 7 o algoritmo 11 obtém ocupações máximas melhores em média e com menor desvio padrão. O algoritmo 12 obtém valores ligeiramente inferiores ao algoritmo 13 para limites baixos.

c) Quanto ao custo para busca com sucesso, o algoritmo 11 é o melhor, seguido do 12 e do 13, e em termos de custo para inserção a ordem é a inversa. É importante notar que quanto maior for o limite, maior será a diferença entre o algoritmo 11 e os algoritmos 12 e 13 para as grandezas acima, pois esses cada vez necessitarão rearranjar a tabela menos vezes. Outro ponto importante é que quanto maior o fator de ocupação maior ficam as diferenças entre os tres algoritmos, tanto para a inserção quanto para a busca, o que indica que nenhum dos tres algoritmos é sempre melhor do que os outros, havendo uma escala que dependerá da relação entre o número de inserções e o número de buscas. Quanto menor for essa relação tanto melhor será o algoritmo 11 em comparação com os outros.

d) Para qualquer dos algoritmos a remoção tem o custo igual a uma busca com sucesso, pois simplesmente encontramos a chave e apagamos a entrada.

e) O número médio esperado de comparações para uma busca sem sucesso para todos os quatro algoritmos é sempre igual ao limite adotado.

f) CLAPSON[12] notou que à medida em que se fazia inserções e remoções com o algoritmo de hashing limitado (como descrito na seção (III.3.9)), o desempenho caia gradualmente e o valor máximo de ocupação conseguido também, sendo necessário reinserir todas as chaves. Acreditamos que isso seja devido aos "shadow-buckets" utilizados, que vão ficando cheios nas áreas mais carregadas. Embora não tenhamos feito testes de inserção/deleção, não vemos porque este problema possa ocorrer nos novos algoritmos.

g) Concluimos que os algoritmos l1, l2 e l3 atendem com eficiência aos requisitos de:

1) Baixo tempo médio esperado para buscas com sucesso.
2) Remoção eficiente
3) Aproveitamento de memória
4) Memória interna

e ainda podem ser boas opções para aplicações que não tenham todas estas características. Como veremos mais adiante no capítulo V, desde que seja utilizada uma blocagem conveniente e restringida a faixa de variação do incremento à valores menores, eles poderão ser utilizados para memória externa com

eficiência.

h) Os algoritmos 11, 12 e 13, buscam uma chave na tabela, e se esta não for encontrada fazem sua inserção na mesma. Então toda inserção deve ser precedida de uma busca sem sucesso que gasta um número de comparações igual ao limite. Em muitas aplicações não há deleção durante a criação do arquivo, quando por exemplo re-inserimos todas as chaves em um arquivo maior, o que nos permite parar a busca quando encontrarmos um lugar vago. Essa modificação nos dá uma economia considerável no número de comparações para inserir, principalmente na fase em que a tabela não está ainda muito cheia, pois existirá economia sempre que o Cm' (do hashing-duplo) for menor do que o limite.

Para fazermos isso basta mudar nos algoritmos 11, 12 e 13 a fase de busca pelos comandos que daremos a seguir:

```
HS:=R;     S:=0
while  S ≤ SLIM  and  K(HS)≠VAGO do
 ┌begin
 │  if  K(HS)=KX  then go to ENCONTRADO;
 │  S:=S + 1;
 │  HS:= (HS + Q) mod N;
 └end;
if S > 1 .............        ou if S ≥ SLIM ......
       -                              -
       -                              -
       -                              -
```

Com essas modificações obteremos valores para o custo de inserção bem inferiores àqueles da tabela (IV.3.3-9) e que podem ser obtidos a partir desta subtraindo-se o limite(que é o Cm' do hash limitado) e somando $\sum_{m=1}^{\alpha N} (\frac{N+1}{N+1-m})/(\alpha N)$ que é o Cm' do hash-duplo.

Poderemos obter um ganho a mais, se pudermos garantir de antemão que a chave a ser inserida não se encontra na tabela, quando então poderemos retirar o comando "if K(HS)=KX then go to ENCONTRADO;" do laço de busca.

IV.4. CASO 3

IV.4.1. INTRODUÇÃO

Vemos, que uma alta ocupação de memória pode ser conseguida com os algoritmos 11, 12 e 13 se permitirmos um limite razoável, (maior do que 7). Observamos também que o Cm obtido com os algoritmos 12 e 13 piora sensivelmente se o limite subir muito, pois eles práticamente nunca rearrumarão a tabela, e com o algoritmo 11, o Cm práticamente independe do limite quando este não é muito pequeno. E por último, observamos que quanto maior o limite, pior será a busca sem sucesso.

Tentando conciliar esses objetivos, conflitantes nos algoritmos 11, 12 e 13, acrescentamos uma pequena mo-

dificação em cada um deles que agora nos permite:

1) Alta ocupação de memória.
2) Busca sem sucesso baixa e limitada.
3) Busca com sucesso melhor nos algoritmos 12 e 13 mesmo com limites razoávelmente altos.

sem perder nenhuma das vantagens dos algoritmos 11, 12 e 13. Os novos algoritmos são apresentados a seguir.

IV.4.2. ALGORITMOS DE HASHING LIMITADO, COM REARRANJO E LIMITE DINÂMICO

A idéia agora é continuar tendo um limite máximo de comparações para uma busca, mas usar também um "limite atual", que será mantido tão baixo quanto possível, não podendo ultrapassar nunca o limite máximo.

Começamos a inserção com um limite baixo, e tentamos respeitá-lo; quando não for mais possível, incrementamos o limite e tentamos novamente a inserção, até alcançar o limite máximo. Isso nos permite trabalhar sempre com o menor limite possível, resolvendo os problemas descritos anteriormente.

Para complementar a idéia acima nos casos em que haja remoção, utilizamos um vetor de contadores, de 1 ao limite máximo, e a cada inserção incrementamos o contador correspondente ao comprimento da carreira da chave recém inserida, e a cada remoção decrementamos o mesmo contador; se ele se tornar igual a zero e for o contador de maior índice que

estava diferente de zero, poderemos decrementar o limite atual até o valor correspondente ao índice do maior contador diferente de zero.

Aplicando esta idéia aos algoritmos 7, 11, 12 e 13 obtivemos respectivamente os algoritmos 16, 17, 18 e 19.

IV.4.3. ALGORITMO 16 - HASHING LIMITADO COM LIMITE DINÂMICO

```
procedure   ALG16( KX,SLIM,CHEIO );  value KX, SLIM;
begin
  if not CHEIO
   then do begin
           ALG7 (KX, LA, CHEIO);
           if  CHEIO  then if LA=SLIM then SAI:=true
                                      else begin
                                             CHEIO:=false;
                                             LA:=LA + 1;
                                           end
                      else begin
                             CONT(LA):=CONT(LA) + 1;
                             SAI:=true;
                           end;
           end
        until SAI;
end da procedure ALG16;
```

Os algoritmos 17, 18 e 19 são idênticos ao acima, trocando apenas a chamada de ALG7 por ALG11, ALG12 e ALG13 e, dentro destas, atualizar os contadores sempre que

mudar alguém de lugar. No 2º parâmetro entrará o limite atual, e retornará em caso de inserção o comprimento da carreira.

IV.4.4. TESTES DOS ALGORITMOS 16, 17 18 E 19

As simulações foram realizadas nas mesmas condições das anteriores só que agora existe um limite máximo igual a 50, e um limite dinâmico que começa em 0. O limite é de "saltos", sendo que com limite zero todas as chaves deverão estar no seu "home-address". Na prática não haverá interesse em se começar com limite menor do que dois, que é quando já se pode tentar alguma rearrumação na tabela.

Os resultados são mostrados nas tabelas (IV.4-1) e (IV.4-4) e são comentados após estas.

TEMPO MÉDIO POR CHAVE PARA INSERIR ATÉ CADA OCUPAÇÃO

		10%	20%	30%	40%	50%	60%	70%	80%	90%	100%.
ALG16	μ	205	199	202	212	220	225	232	237	269*	-
	σ	11,7	4,6	5,3	7,1	5,3	5,1	5,0	7,5	11,8	-
ALG17	μ	212	206	213	225	239	252	269	287	338	1005*
	σ	13,5	5,6	5,3	7,5	6,2	6,0	6,7	7,8	11,3	329
ALG18	μ	205	201	207	216	226	235	246	257	296	936*
	σ	12,8	5,3	6,9	8,7	4,9	5,8	5,6	7,8	9,9	317
ALG19	μ	206	202	207	218	226	235	245	254	285	892*
	σ	12,4	4,8	6,9	7,3	5,6	5,4	6,1	7,7	9,8	334

Tabela IV.4-1

NÚMERO DE COMPARAÇÕES POR CHAVE PARA INSERIR ATÉ CADA OCUPAÇÃO

		10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
ALG16	μ	1,16	1,82	2,93	4,11	5,62	7,56	10,5	15,7	27,1	-
	σ	0,03	0,02	0,03	0,03	0,04	0,06	0,08	0,17	0,32	-
ALG17	μ	1,03	1,10	1,43	1,89	2,24	2,61	4,28	4,39	6,38	16,9*
	σ	0,03	0,03	0,03	0,04	0,05	0,06	0,08	0,10	0,17	11,2
ALG18	μ	1,03	1,09	1,41	1,86	2,16	2,46	3,30	4,23	5,92	15,5*
	σ	0,02	0,03	0,03	0,03	0,04	0,05	0,06	0,10	0,16	10,9
ALG19	μ	1,03	1,10	1,42	1,86	2,19	2,64	3,34	4,14	5,76	14,7*
	σ	0,02	0,03	0,03	0,03	0,04	0,05	0,06	0,09	0,16	11,5

Tabela IV.4-2

NÚMERO MÉDIO ESPERADO DE COMPARAÇÕES PARA UMA BUSCA COM SUCESSO

		10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
ALG16	μ	1,05	1,11	1,18	1,27	1,38	1,51	1,70	2,00	2,52	-
	σ	0,02	0,02	0,02	0,02	0,03	0,03	0,04	0,06	0,09	-
ALG17	μ	1,04	1,10	1,15	1,21	1,29	1,38	1,49	1,64	1,87	2,57*
	σ	0,02	0,01	0,01	0,02	0,02	0,02	0,02	0,02	0,03	0,06
ALG18	μ	1,04	1,10	1,16	1,23	1,31	1,41	1,53	1,69	1,94	2,64*
	σ	0,02	0,01	0,02	0,02	0,02	0,02	0,03	0,03	0,03	0,07
ALG19	μ	1,04	1,10	1,16	1,23	1,31	1,41	1,53	1,69	1,95	2,68*
	σ	0,02	0,01	0,02	0,02	0,02	0,02	0,03	0,03	0,04	0,07

Tabela IV.4-3

LIMITE MÍNIMO MÉDIO COM O QUAL SE CONSEGUE

ATINGIR CADA OCUPAÇÃO

		10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
ALG16	μ	1,31	2,26	3,45	4,71	6,43	8,55	12,2	18,6	34,4	50,0*
	σ	0,52	0,67	0,93	1,13	1,55	2,11	2,83	5,15	8,10	0,00
ALG17	μ	1,02	1,12	1,64	1,99	2,24	2,75	3,30	4,27	6,30	33,8*
	σ	0,14	0,32	0,48	0,26	0,47	0,43	0,54	0,54	0,83	9,58
ALG18	μ	1,02	1,12	1,64	1,99	2,26	2,83	3,43	4,40	6,23	33,8*
	σ	0,14	0,32	0,48	0,26	0,46	0,40	0,59	0,55	0,82	9,3
ALG19	μ	1,02	1,12	1,64	1,99	2,26	2,83	3,45	4,35	6,32	34,7*
	σ	0,14	0,32	0,48	0,26	0,46	0,40	0,60	0,57	0,97	9,30

Tabela IV.4-4

IV.4.5. OBSERVAÇÕES SOBRE OS ALGORITMOS 16, 17, 18 E 19

a) Vemos das tabelas (IV.4-1) a (IV.4-4) que com a introdução do limite dinâmico, os algoritmos 18 e 19 se aproximam bastante do algoritmo 17 em termos de custos e vantagens. Poderemos manter o custo de inserção dos alg. 18 e 19 mais baixo, com ganhos menores na busca, se inicializarmos o limite atual já com um valor razoávelmente alto (4 ou 5) o que levaria ao rearranjo somente nos casos que ultrapassassem esse limite inicial. Essa idéia pode ser implementada também fazendo um limite inferior um pouco menor do que o limite atual e subindo ou descento junto com ele.

b) O custo de uma busca com sucesso melhorou sensivelmente, pois quando a ocupação é baixa o limite atual também é baixo, sendo C'_m aproximadamente igual ao do hash duplo, e quando a ocupação sobe o hash limitado apresenta valores médios muito menores do que os encontrados em hash duplo e com a vantagem de serem garantidamente limitados.

c) Infelizmente pela forma como foram feitas as simulações,de 10% em 10%, a região entre 90% e 100% está muito pouco definida, mas muita informação intermediária pode ser obtida da tabela (IV.3.3-5). As simulações não foram repetidas utilizando-se fatores de ocupação menos espaçados por serem demoradas e onerosas devido a estrutura de monitoração dos algoritmos que utilizam arranjos tridimensionais muito grandes e manipulados com dificuldade pelo B6700.

d) Com a introdução do limite dinâmico pode-se permitir um limite máximo razoávelmente folgado que nos leva a alcançar uma ocupação de memória alta (com limite máximo de 15, podemos garantir uma ocupação mínima para o algoritmo 17 de 97,3% com 3 desvios padrão de confiança), em momentos de pico e trabalhar com limites mais baixos quando a ocupação da tabela for menor ou quando forem removidos os casos "patológicos" que tenham por ventura aumentado desmedidamente o limite atual.

e) Com estes algoritmos acreditamos que os requisitos de baixo custo de inserção, baixo custo para buscas com e sem sucesso, grande aproveitamento de memória, pior caso limitado e trivial podem ser compromissados entre si e, dependendo da aplicação, obter valores próximos do ótimo de cada um sem comprometer demasiadamente os demais.

Como um critério de ponderação entre as características desejadas poderemos inclusive determinar o ponto ótimo de trabalho do sistema, em que obteremos o menor custo total, a partir dos dados experimentais obtidos.

IV.5. CASO 4

IV.5.1. APRESENTAÇÃO

Observando a tabela (IV.4-4) vemos que embora para valores altos de α o limite necessário seja baixo, se desejassemos trabalhar com limite 3 ou 4 não conseguiríamos "garantir" mais do que 40% ou 55% de ocupação de memória respec-

tivamente.

Em alguns casos de tabelas muito pesquisadas desejamos ou necessitamos de limites baixos juntamente com grande aproveitamento de memória, como por exemplo tabelas utilizadas por sistemas de gerencia de bancos de dados, onde as funções básicas já estão sendo colocadas a nível de microprogramas, matrizes de transição esparsas de sistemas "table-driven" que podem ser armazenadas em uma tabela "hashing",etc.

Pensando nestes requisitos criamos o algoritmo descrito sumariamente a seguir:

Suponhamos que o algoritmo 17 de hashing limitado tentasse inserir a chave K_i na tabela da figura IV.5.1-1 com limite 4.

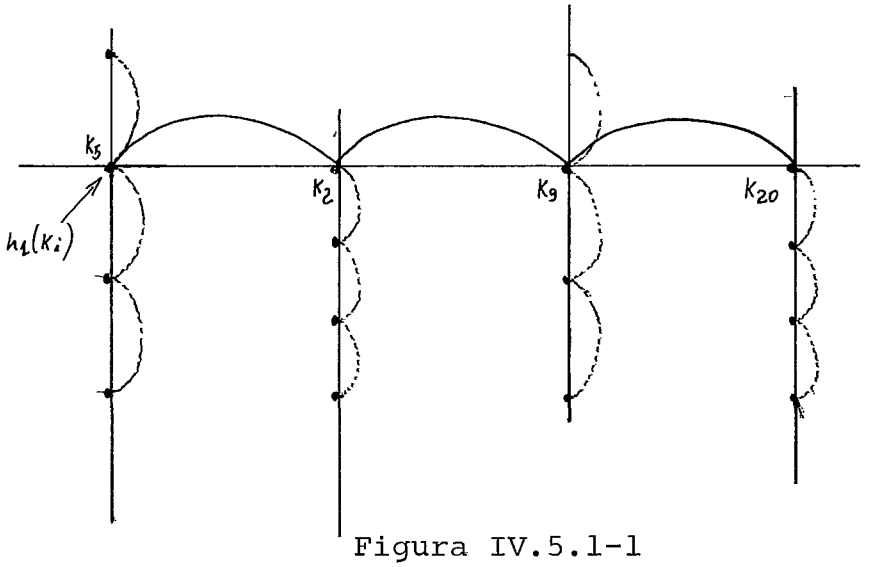


Figura IV.5.1-1

Vemos que todas as carreiras estão completas e K_i não poderia ser inserido. A probabilidade de ocorrencia desta situação é limitada superiormente pelo probabilidade de se retirar 16 bolas pretas de um saco com α% de bolas pretas

ço para o lugar que ficou vago e inseriríamos K_i no lugar vago da primeira carreira horizontal.

Exemplificando, suponhamos que o único lugar vago fosse o marcado com X na figura IV.5.1-3, após a inserção na tabela, teríamos o descrito na figura IV.4.1-3. A chave K_{11} seria deslocada para o lugar vago, K_9 iria para o seu lugar K_i entraria no lugar deixado por K_9. O limite foi mantido e o algoritmo de busca é o mesmo dos métodos anteriores, de extrema simplicidade.

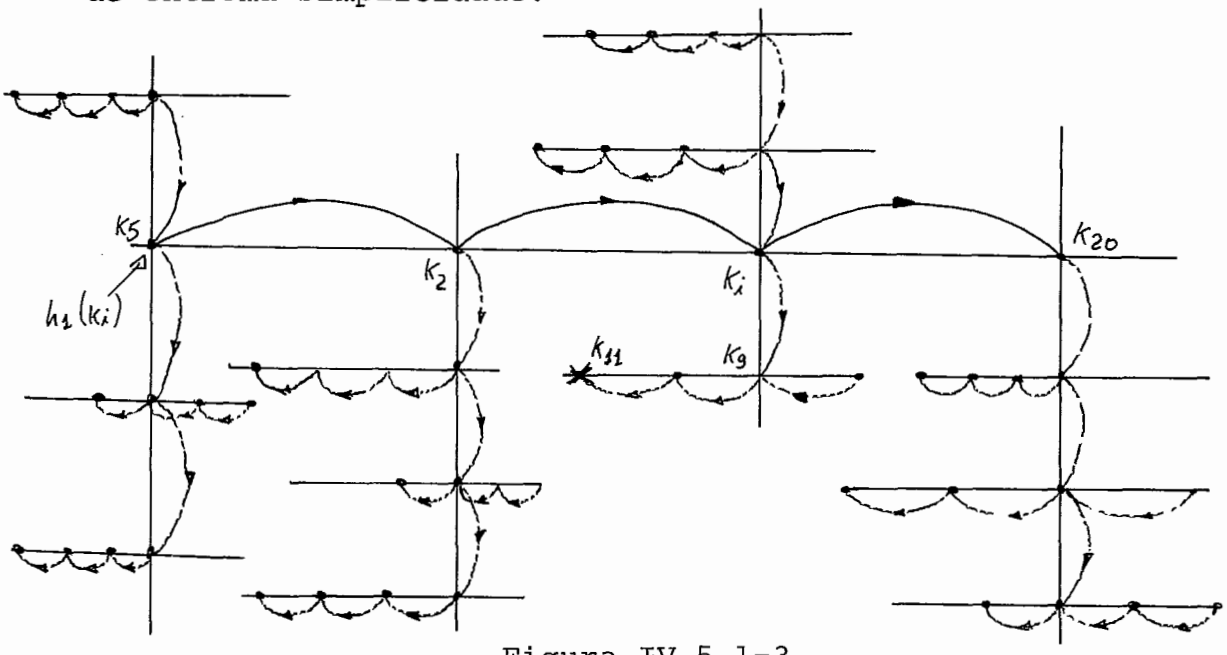


Figura IV.5.1-3

Se permitíssemos busca de lugar vago em profundidade 3 - com limite de 4 saltos - a probabilidade de não conseguirmos inserir K_i seria limitada superiormente pela probabilidade de retirarmos 160 bolas pretas de um saco com $\alpha\%$ de bolas pretas sem reposição e inferiormente com reposição para limite $<<N$. Na realidade o fenômeno é um pouco mais complicado pois retirar uma bola repetida no segundo nível implica em casos diversos do que retirar no terceiro nível. A par

tir do momento que duas carreiras se cruzem haverá uma redução do número de casas visitadas em todas as carreiras "descendentes" destas.

Decidimos não efetuar a programação, testes e desenvolvimento deste algoritmo por entendermos que este trabalho não poderia ser completamente exaustivo no assunto. Entretanto, a idéia de alongarmos o rearranjo da tabela oferece excelentes perspectivas de desenvolvimento, onde gostaríamos de ressaltar a possibilidade de escolha do caminho a percorrer na árvore de buscas se em profundidade ou horizontal, e ainda a transformação do problema em uma escolha de caminho ótimo em grafo.

Fica a referência como uma proposta de pesquisas futuras.

IV.6. PROBLEMAS PROPOSTOS

Gostaríamos a título de esclarecimento e como proposta para outros estudos de referenciar os dois casos adicionais cuja análise e solução deixamos além do esboço desta tese:

a) O problema de ordenação que não aparenta ter solução simples e, provavelmente, não possui solução que utilize "hashing"

b) A alocação dinâmica de memória que talvez seja um requisito solucionável com mais facilidade do que a manutenção do arquivo ordenado,mas que por si só justifica-se como tema de tese independente.

V. DETALHES PARA IMPLEMENTAÇÃO EM MEMÓRIA EXTERNA

A forma como foram propostos os algoritmos destina-se especialmente à sua utilização em tabelas onde as chaves e os apontadores para a informação ou as chaves e a informação sejam residentes em memórias interna não virtual. Algumas das características das aplicações em que isto ocorre são listadas abaixo:

a) A informação não é muito maior do que a chave, ocupando a última, parcela considerável do espaço

b) O tempo para cálculo da função é significativo em relação ao tempo de acesso à tabela

c) Tabelas não muito grandes

d) Mais provável ocorrencia de tabelas estáticas

e) Número de buscas por entrada geralmente grande

f) Pouca "localidade" nas sequências de buscas, sendo geralmente K_i totalmente independente de K_{i+1}.

Dependendo da existência em maior ou menor grau dessas características na aplicação, as variáveis que se procuram minimizar são diferentes.

Quando as chaves da tabela estão armazenadas em memória externa ou em memória virtual de grandes sistemas há duas variáveis preferencialmente minimizáveis:

a) O número de acessos físicos à memória externa

b) O tempo gasto para se transferir dados da memória externa para a interna.

Para diminuir o número de acessos físicos costuma-se grupar as entradas em "caixas" ou "buckets" de tamanho b, que são lidos de uma só vez para a memória interna e então se procura por um processo qualquer (busca binária, sequencial, etc) a chave dentro da caixa. Aqui as entradas não são endereçadas diretamente mas sim as caixas.

Quando uma caixa fica completa resolve-se o problema de uma das seguintes formas:

a) Endereçamento aberto com visita linear

b) endereçamento na própria tabela com listas coalescentes

c) Endereçamento em área separada com listas independentes

Sabe-se de dados experimentais (KNUTH[2], LUM[8], SEVERANCE[27]) e de análises teóricas que dentro de uma faixa razoável quanto maior for o tamanho da caixa em geral menor será o número médio esperado de acessos à disco para buscas com sucesso (para buscas sem sucesso ocorre o inverso). Mas se levarmos em conta o custo de transferência dos dados, o custo da área de "buffers", o tamanho das trilhas do disco e o custo da busca em memória; obtém-se uma faixa de valores aceitáveis para o tamanho das caixas, onde geralmente $1 \leq b \leq 100$.

Para sistemas pequenos com UCP lenta, taxas de transmissão de dados do disco para a memória lenta também,pouca área para "buffers"; caixas pequenas ficam mais interessantes.

Convém ainda estudar com atenção as tabelas de KNUTH[2] e LUM[8] e principalmente SEVERANCE [27] quando se for escolher o tamanho da caixa.

Uma observação interessante sobre o tamanho das caixas é que não parece ser muito melhor usar caixas maiores com baixo fator de bloco do que usar caixas menores com maior fator de bloco, de modo que em ambos os métodos se tenha o mesmo tamanho de bloco, se for usado endereçamento aberto e rearranjo no segundo caso. Naturalmente no segundo método haverá formação de agrupamentos que poderão levar a cruzar as fronteiras de um bloco com alguma frequência, mas não é evidente sua inferioridade. Por outro lado podemos tentar grupar físicamente a carreira de uma certa chave, sem usar visita linear. O método de incremento binário parece ser interessante, mas acreditamos que o hashing duplo com rearranjo é compensador desde que sua segunda função tenha uma limitação de faixa suficientemente pequena de modo a dar poucos cruzamentos de página. A idéia surgiu com BELL[26] e KAMAN[26] no método quociente linear. Eles diminuiram a faixa para cerca de 1/32 de n observando um acréscimo de somente 4% em C_m. Tentamos restringir a segunda função a faixa 3,7,15 e 31 e notamos que a medida que vamos aumentando a faixa o decréscimo marginal vai diminuindo havendo um ponto a partir do

qual praticamente não é lucrativo aumentar a faixa. Não acreditamos que este ponto dependa do tamanho da tabela sendo função exclusiva das probabilidades de ocorrerem agrupamentos de tamanho 1,2,3... que para funções perfeitamente uniformes dependerão somente de α, desde que este não esteje no entorno de 100%.

Segue-se um exemplo comparativo entre o endereçamento aberto com visita linear com caixas de tamanho 70 e o hash limitado com limite 10, e algumas tabelas retiradas de SEVERANCE[27] que são utilizadas para a comparação.

REGISTROS DE "OVERFLOW POR REGISTRO ARMAZENADO (%)

TAM. BUCKET	FATOR DE CARGA (α)							
	0.7	0.9	1.0	1.1	1.5	2.0	3.0	5.0
1	28.08	34.06	36.79	39.35	48.21	56.77	68.33	80.13
2	17.03	23.79	27.07	30.24	41.63	52.75	67.00	80.01
3	11.99	18.87	22.40	25.91	38.80	51.36	66.75	80.00
4	9.05	15.86	19.54	23.25	37.22	50.74	66.69	80.00
5	7.11	13.78	17.55	21.42	36.22	50.43	66.67	80.00
10	2.88	8.59	12.51	16.85	34.25	50.04	66.67	80.00
20	0.81	4.99	8.88	13.66	33.50	50.00	66.67	80.00
50	0.05	2.04	5.63	11.03	33.34	50.00	66.67	80.00
100	0.00	0.83	3.99	9.92	33.33	50.00	66.67	80.00

Tabela V-1

REGISTROS DESPERDIÇADOS POR REGISTRO ARMAZENADO (%)
(CONTANDO ÁREA SEPARADA PARA "OVERFLOW")

TAM. BUCKET	FATOR DE CARGA (α)							
	0.7	0.9	1.0	1.1	1.5	2.0	3.0	5.0
1	70.94	45.17	36.79	30.26	14.88	6.77	1.66	0.13
2	59.89	34.90	27.07	21.15	8.30	2.75	0.33	0.01
3	54.84	29.98	22.40	16.82	5.40	1.36	0.08	0.00
4	51.90	26.97	19.54	14.16	3.88	0.74	0.02	0.00
5	49.97	24.89	17.55	12.33	2.89	0.43	0.01	0.00
10	45.73	19.70	12.51	7.76	0.91	0.04	0.00	0.00
20	43.66	16.10	8.88	4.57	0.16	0.00	0.00	0.00
50	42.91	13.10	5.63	1.94	0.00	0.00	0.00	0.00
100	42.86	11.94	3.99	0.83	0.00	0.00	0.00	0.00

Tabela V-2

NÚMERO MÉDIO DE ACESSOS A "BUCKET" POR REGISTRO RECUPERADO
(END.ABERTO)

TAM.	FATOR DE CARGA (α)								
BUCKET	0.20	0.50	0.70	0.80	0.90	0.92	0.94	0.95	0.96
1	1.13	1.50	2.17	3.00	5.50	6.75	8.83	10.50	13.00
2	1.02	1.18	1.49	1.90	3.77	4.81	5.64	5.64	6.89
3	1.01	1.09	1.29	1.55	2.38	2.79	3.49	4.04	4.85
4	1.00	1.05	1.19	1.39	2.00	2.31	2.83	3.24	3.87
5	1.00	1.03	1.14	1.29	1.78	2.02	2.44	2.77	3.27
10	1.00	1.00	1.04	1.11	1.34	1.47	1.67	1.84	2.09
20	1.00	1.00	1.01	1.04	1.14	1.20	1.30	1.39	1.51
50	1.00	1.00	1.00	1.01	1.04	1.06	1.10	1.13	1.18
100	1.00	1.00	1.00	1.00	1.01	1.02	1.04	1.05	1.08

Tabela V.3

TAM.	FATOR DE CARGA (α)									
BUCKET	0.2	0.5	0.7	0.9	1.0	1.1	1.5	2.0	3.0	5.0
1	1.10	1.25	1.35	1.45	1.50	1.55	1.75	2.00	2.50	3.50
2	1.02	1.13	1.24	1.36	1.43	1.50	1.82	2.25	3.17	5.10
3	1.01	1.08	1.18	1.32	1.40	1.49	1.90	2.50	3.83	6.70
4	1.00	1.05	1.14	1.29	1.38	1.48	1.98	2.75	4.50	8.30
5	1.00	1.04	1.12	1.27	1.37	1.48	2.07	3.00	5.17	9.90
10	1.00	1.01	1.06	1.21	1.33	1.50	2.49	4.25	8.50	17.90
20	1.00	1.00	1.02	1.15	1.31	1.55	3.33	6.75	15.17	33.90
50	1.00	1.00	1.00	1.08	1.29	1.71	5.83	14.25	35.17	81.90
100	1.00	1.00	1.00	1.04	1.28	1.97	10.00	26.75	68.50	161.90

Tabela V.4

EXEMPLO COMPARATIVO

COM VISITA LINEAR E CAIXAS DE TAMANHO 70

Com $\begin{cases} b = 70 \\ \alpha = 0,7 \end{cases}$ terei 0.025% dos registros fora do "home-address" e número de acessos à disco médio ≃ 1.00

Para busca em memória:

binária $\begin{cases} Cm \simeq 7 \\ Cm' \simeq 7 \end{cases}$ inserção difícil

sequencial $\begin{cases} Cm \simeq 35 \\ Cm' \simeq 35 \end{cases}$ inserção fácil

Resultado:

Pior caso (0,7n)/b acessos à disco

COM O ALGORITMO 11 E 2ª. FUNÇÃO LIMITADA

b = 1

α = 0,7

fator de bloco = 70

limite = 7 com segundo hashing(incremento) no intervalo [1,7]

0,0% de overflow do bloco garantido

Busca em memória

End. aberto - Cm = 1,57

Cm' = 5,47 (limite dinâmico)

Inserção fácil

deleção mais simples.

Resultado:

Pior caso 1 acesso à disco

CONCLUSÕES: mesmo utilizando caixa de tamanho 1, o método tem desempenho comparável aos métodos que utilizam caixas maiores do que 1. Temos a vantagem de uma maior elegância na organização do arquivo e economizamos tempo de processador para processar as caixas, o que pode ser significativo em micro e mini-computadores onde a velocidade do processador pode ser uma limitação à aplicação. Mas, não há nenhum inpedimento à utilização dos nossos algoritmos com caixas maiores do que 1,quando, acredita-se, o número de acessos à disco deva diminuir ainda mais.

Nesse ponto gostaríamos de chamar a atenção para um fato que já vem sendo longamente difundido em diversos artigos, HAMMER[9], REGE[10], PANIGRAHI[22], FETH[32] e TORRERO[33] entre outros, que é o surgimento de memórias de massa de acesso realmente aleatório, com tempos de acesso menor do que disco magnético, menor custo, menor em tamanho, e sem os problemas de ordem mecânica deste. Com a utilização dessas memórias, que se preve possam desbancar os discos magnéticos em menos de 15 anos, toda a parafernália de: caixas, áreas de "overflow" separadas, "shadows buckets", tamanho de registro físico sub-múltiplo de tamanho de trilha, etc serão banidos da programação em benefício de uma maior clareza na estrutura de arquivos, utilizando endereçamento aberto com caixas de tamanho 1, ou seja, sem caixas.

VI. CONCLUSÕES FINAIS

Embora junto à cada seção já tenhamos analisado os resultados obtidos, cabe aqui uma avaliação geral:

Das 16 principais características detetadas nas aplicações práticas, listadas na seção III.4, somente duas não foram estudadas e permaneceram sem solução eficiente, são elas: a manutenção do arquivo ordenado e a alocação dinamica de memória.

Foram mostrados os principais algoritmos existentes, programando-se inclusive 5 algoritmos, e se desenvolveu 14 novos, que atendem no conjunto a todos os requisitos encontrados nas aplicações práticas, listados na seção III.4, com exceção dos dois citados no parágrafo anterior.

Não terá escapado ao leitor atento, que os 14 algoritmos aqui desenvolvidos podem ser reunidos em um só algoritmo, onde, através da atribuição de valores à parâmetros específicos, pode-se obter qualquer um dos 14 e também os de hashing-duplo, visita linear e o de Brent. Este fato só ficou claro para nós depois de ter desenvolvido e testado todos os algoritmos, mas preferimos apresentar o trabalho desta forma, do particular para o geral, por ficar mais claro o desenvolvimento natural de um para outro algoritmo.

São propostos diversos métodos para obtenção de boas funções de hashing para conjuntos específicos de dados.

Ficam como pontos para posterior desenvolvimento, os métodos citados no parágrafo anterior, os algoritmos que utilizam profundidade, e a utilização de caixas maiores nos novos algoritmos.

BIBLIOGRAFIA

1. KNUTH, D.E. - "The art of computer programming, Vol.III: Sorting and Searching", Addison-Wesley, Reading, Mass., 1973.

2. KNUTH, D.E. - "The arto of computer programming, Vol.I: Fundamental algorithms", Addison-Wesley, Reading, Mass., 1968.

3. CLARK, D.W. e GREEN, C.C. - "An empirical study of list structure in LISP" - Comm. ACM 20(2), Fev.1977, pp.78-87.

4. EASTON, M.C. e BENNET, B.T. - "Transient-free working-set satistics", Comm. ACM 20(2) , Fev.1977, pp.93-99.

5. SPRUGNOLI, RENZO - "Perfect hashing functions: a single probe retrieving method for static sets", Comm. ACM 20(11) Nov.1977, pp.841-850.

6. GRNIEWSKI,M e TURSKI, W. - "The external language KLIPA for the URAL-2 digital computer", Comm.ACM6(6), Jun.1963, pp.322-324.

7. ZIPF, Z.K. - "Humam behavior and the principle of least effort: an introduction to human ecology", Reading, Mass. Addison-Wesley, 1949.

8. LUM,V.Y.;YUEN,P.S.T. e DODD,M. - "Key to address transform techniques: a fundamental performance study on large existing formatted files", Comm.ACM 14(4),Abr.1977,pp.228-239.

9. HAMMER, CARL - "A forecast of the future of computation", Information and management, 1(1), Nov.1977, pp.3-9.

10. REGE,S.L. - "Cost, performance and size trade off for different levels in a memory hierarchy", Computer 9(4) Abr.1976, pp.43-50.

11. CRUZ,PEDRO NOGUEIRA - "Desenvolvimento e implementação de um sistema de arquivo em disco para acesso aleatório", Tese M.Sc., COPPE/UFRJ, 1975.

12. CLAPSON, PHILIP - "Improving the access time for random access files", Comm. ACM 20(3), Mar.1977, pp.127-135.

13. MORRIS, ROBERT - "Scatter storage techniques", Comm ACM 11(1), Jan.1968, pp.38-44.

14. HARRISON, MALCOLM C. - "Implementation of the substring test by hashing" , Comm.ACM 14(12), Dez.1971,pp.777-779.

15. BLOOM, BURTON H. - "Space-time trade-offs in hash coding with allowable erros", Comm.ACM 13(7), Jul.1970, pp.422-426.

16. PIETRACCI, LUCIANO - "Compressão de dados", Tese M.Sc., COPPE/UFRJ, 1973.

17. WILLIAMS, F.A. - in KNUTH[2], pp.514.

18. BRENT,RICHARD P. - "Reduzing the retrieval time of scatter storage techniques", Comm.ACM 16(2), Fev.1973, pp.105-109.

19. PETERSON, W.W. - "Addressing for random-access storage", IBM Journal R & D, 1(12), Abr.1957, pp.130-146.

20. ERSHOV, A.P. - in KNUTH[2], pp.541.

21. MAURER, W.D. - "An improved hash code for scatter storage", Comm. ACM 11(1), Jan.1968, pp.35-38.

22. PANIGRAHI, G. - "Charge-coupled memoried for comp. systems", Computer 9(4), Abr.1976, pp.33-41.

23. BALBINE, GAY de. - Tese Ph.D. Calif. Inst. of Technology, 1968, pp.149-150, in KNUTH[2].

24. RADKE, C.E. - "The use of quadratic residue research", Comm. ACM 13(2), Fev.1970, pp.103-105.

25. BELL, JAMER R. - "The quadratic quocient method: a hash code eliminating secondary clustering", Comm. ACM 13(2), Fev.1970, pp.107-109.

26. BELL, JAMES R. e KAMAN, CHARLES H. - "The linear quotient hash code", Comm. ACM 13(11), Nov.1970, pp.675-677.

27. SEVERANCE, DENNIS e DUHNE, RICARDO - "A practitioner's guide to addressing algorithms", Comm.ACM 19(6), Jun. 1976, pp.314-326.

28. SCHUEGRAF, E.J. - "Compression of large inverted files with hyperbolic ter distribution", Information Processing & Management 12(6) 1976, pp.377-384.

29. HEHNER, ERIC C.R. - "Computer design to minimize memory requirements", Computer 9(8), Ago.1976, pp.65-70.

30. HEISING, W.P. - "Note on random addressing techniques", IBM Systems Journal, Jun.1963, pp.112-116.

31. BLAKE, IAN F. e KONHEIM, ALAN - "Big buckets are (are not) better.", J.ACM 24(4), Out.1977, pp.591-606.

32. FETH, George C. - "Memories:smaller, faster and cheaper", SPECTRUM 13(3), Set.1976, pp.37-44.

33. TORRERO, EDWARD A. - "Bubbles rise from the lab.", SPECTRUM 13(9), Set.1976, pp.28-31.

34. GUIMARÃES, CÉLIO - Notas de aula do curso "Busca em Arquivos", COPPE/UFRJ, 1975.

35. SOUZA, JANO MOREIRA DE - Notas de aula do curso "Busca em arquivo", COPPE/UFRJ, 1976 e 1977.