unesp
UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
Campus de Bauru
Faculdade de Ciências

Disciplina: Estruturas de Dados I – **ED1**
Professora: Simone das Graças Domingues Prado
e-mail: simone.prado@unesp.br

Apostila 06 - Hashing

Objetivos:

⇒ Aprender o conceito de Hashing
⇒ Aprender as funções Hashing
⇒ Fazer tratamento de colisão nas tabelas

Conteúdo:

1. INTRODUÇÃO

Ao fazer a busca de um elemento X em um conjunto de elementos, devem-se comparar os elementos desse conjunto com o elemento X até encontrá-lo. Isso pode ser feito usando vários métodos: busca sequencial, busca binária, busca numa lista ordenada etc.

Imagine que a busca pelo elemento X não seja simplesmente usar comparação direta com elementos de uma tabela ou conjunto. Imagine outros problemas de recuperação de informação:
(1) distribuição de correspondências de funcionários em uma empresa: podem-se criar escaninhos rotulados com a(s) primeira(s) letra(s) do sobrenome e armazenar neles as correspondências;
(2) catalogação de documentos em gavetas rotuladas com os primeiros dígitos dos documentos;
(3) catalogação de fotografias, etc.

Esses exemplos de organização, que promovem a melhora na inserção e recuperação de uma informação, dão a ideia do método de busca a ser estudado aqui - Hashing.

Com o objetivo de melhorar a inserção e localização do elemento, não é dada ênfase na operação de remoção dos elementos e nem se preocupa com a ordenação dos elementos. A ideia é prover o sistema de um método

rápido e eficiente para inserção e recuperação de um elemento. Como trabalhar com um vetor (por exemplo: int x[]) é mais rápido (já que, a partir do índice, têm-se o elemento), a implementação buscará usar essa estrutura.

O Hashing (que pode ser traduzido por espalhamento, dispersão, escrutínio etc) é um método de busca onde se extrai o índice do sub-conjunto onde o elemento estaria. O índice, que é chamado por alguns autores de encaixe, é calculado a partir de funções, que são chamadas **funções de espalhamento**. A tabela de espalhamento é a implementação desse conceito numa estrutura de dados.

No exemplo de distribuição de cartas, o índice estaria relacionado com em que escaninho deveria estar a carta de um funcionário. A tabela seria o conjunto dos escaninhos. A função de espalhamento seria a verificação do sobrenome, por exemplo.

No Hashing procura-se evitar os espaços vazios, indexando a tabela através do cálculo de uma função de forma a criar classes que comportem os elementos. O ideal seria criar funções que, para cada elemento, criasse um espaço na tabela (uma classe). Isso tornaria o processo de busca muito eficiente. Só que nem sempre se consegue essa função de forma a garantir a injetividade, pois é possível a existência de um elemento que aplicada à função nele resulte no mesmo valor, ou seja, h(x) = h(y), para x diferente de y. Assim têm-se dois elementos pertencentes a uma mesma classe, o que é chamado de elementos sinônimos. Quando isso acontece temos um problema de **colisão**. Esse problema precisa ser tratado.

Quando se consegue uma função de espalhamento onde as classes têm quase a mesma quantidade de elementos (com, no máximo, uma diferença de um elemento de uma classe para outra) diz-se que a função é ótima e o espalhamento uniforme. Quando tem funções que para cada elemento existe uma classe, temos um espalhamento perfeito.

Pode-se perceber que o Hashing é uma técnica de redução de espaço de busca. Quanto mais uniforme for o espalhamento e quanto menor forem as classes, mais eficiente é o método de busca, já que a localização do elemento será imediata a partir da aplicação da função.

Nos livros do Drozdek(2002, p.451-452), Pereira(1996, p.214-220), Preiss(2000, p.174-178) e Szwarcfiter & Markenzon(1994, p.230-235) são mostradas várias funções, tais como: Divisão, Enlaçamento, Meio-Quadrado, Extração, Transformação de Raiz, Transformação de chaves alfanuméricas, Multiplicação, Dispersão de Fibonacci, Dobra e Análise de Dígitos. Cabe ressaltar que em Tenenbaum et al.(1995, p.653-659) e em Drozdek(2002, p.462-467) são discutidas as funções Hashing perfeitas.

Nos livros do Drozdek(2002, p.453-461), Preiss(2000, p.186-213), Szwarcfiter & Markenzon(1994, p.236-255) e Tenenbaum et al.(1995, p.598-603) são mostrados várias métodos para tratar o problema da colisão, tais como: Endereçamento aberto (tentativa linear, tentativa quadrática, dispersão dupla), Encadeamento (Interior e Exterior), Endereçamento em Balde e Dimensão Dinâmica.

Nessa apostila serão estudados algumas funções de espalhamento (Hashing) e alguns métodos para resolver colisão.

2. FUNÇÕES DE ESPALHAMENTO/HASHING

Espera-se, das funções Hashing, a geração de espalhamentos uniformes, funções facilmente computáveis e que produzam um número baixo de colisões. A uniformidade é difícil de ser testada já que o conjunto de elementos normalmente é desconhecido até ser usado. Criar funções com computação simples é mais fácil, basta restringir o estudo das funções Hashing. As colisões normalmente acontecem e é um ponto crítico nas tabelas Hashing e devem ser tratadas com muito cuidado.

Aqui serão vistas as funções: Divisão, Enlaçamento, Dobra, Meio-Quadrado, Extração, Transformação de Raiz e Transformação de chaves alfanuméricas.

2.1. Método da Divisão (ou Divisão inteira)

Essa função é muito usada e muito simples. Basta pegar o resto da divisão inteira entre o elemento X e o tamanho da tabela M, ou seja,

$$H(x) = \text{resto } (X / M)$$

Dessa forma, geram-se os índices entre 0 e M-1 na tabela, portanto M classes.

Exemplo:
(1) Para M = 4 e o conjunto de chaves: 48, 03, 80, 31, 20, obtêm os índices: 0, 3, 0, 3, 0
(2) Para M = 7 e o conjunto de chaves: 48, 03, 80, 31, 20, obtêm os índices: 6, 3, 3, 3, 6
(3) Para M = 23 e o conjunto de chaves: 44, 46,49,68,71,97, obtêm os índices: 21, 0, 3, 22, 2, 5

A principal vantagem de se utilizar essa função é a sua simplicidade. A principal desvantagem é existência de muitas colisões se M não for escolhido adequadamente.

Para a escolha de M existem alguns critérios que podem ser adotados: um deles é escolher M tal que seja um número primo, por exemplo.

2.2. Enlaçamento

Esse método pega a chave e a divide em várias partes homogêneas de algum tamanho fixo e depois essas partes são combinadas (ou enlaçadas) e transformadas no índice. O processo pode ser feito de duas formas: por enlaçamento deslocado ou enlaçamento limite.

No **enlaçamento deslocado**, uma parte é colocada embaixo da outra e elas são somadas. No **enlaçamento limite**, cada parte será colocada em uma ordem diferente (o primeiro é colocado normalmente, o segundo será o número invertido, o terceiro, na ordem normal e assim por diante) e depois somado. O resultado da soma pode passar por algum outro processo se for necessário.

Aqui, deve-se definir o tamanho das partes, o tamanho da tabela e que processamento fazer se o índice gerado não pertencer à tabela.

Exemplo: código 123-45-6789 para ser armazenado numa tabela de 150 posições (M=150)
Divide-se em três partes iguais: 123, 456 e 789.
No enlaçamento deslocado:

123
+ 456
+ 789
=1368

No enlaçamento limite:

123
+ 654
+ 789
=1566

Se a tabela tiver tamanho menor que não comporte esses índices, outro processamento deve ser feito fazendo a divisão das partes em tamanhos menores.

Por exemplo: o valor 1368 (obtido no enlaçamento deslocado) e divide-se em 2 partes iguais: 13 e 68 e refaz o enlaçamento deslocado:

13
+68
=81

Por exemplo: o valor 1566 (obtido no enlaçamento limite) e divide-se em 2 partes iguais: 15 e 66 e refaz o enlaçamento limite:

15
+66
=81

Agora os valores, independente do tipo de deslocamento, cabem na tabela.

2.3. Método da Dobra

Neste método, pega a chave, separa seus dígitos e vai somando seus dígitos sem considerar a parte decimal da soma. Esse processo lembra a dobra de um papel onde temos os dígitos escritos nele. As especificações são: o tamanho da tabela e por consequência quantos dígitos terá o índice.

Exemplo:
Seja a chave: 0123-45-6789, a tabela de 100 posições e, portanto, o índice com 2 dígitos (0 a 99).
Assim a dobradura deve ocorrer para cobrir dois números.
Então faz a primeira dobra: (01) e (23)(456789) e obtém: (01)+(*32*)=(0+3)(1+2)=(3)(3)=33 e papel fica (33)(45)(6789).
Segunda dobra: (33)+(*54*) = (3+5)(3+4) = (8)(7) = 87 e temos: (87)(67)(89).
Terceira dobra: (87)+(*76*) = (8+7)(7+6) = (~~1~~5)(~~1~~3) = (53) e temos: (53)(89),
Quarta dobra: (53)+(*98*) = (5+9)(3+8) = (~~1~~4)(~~1~~1) = (41) e temos: (41).

01	33	8 7	5 3
+32	+54	+ 7 6	+ 9 8
=33	=87	=15 13	=14 11
		= 5 3	= 4 1

Portanto para a chave 0123-45-6789, h(chave) = 41.

2.4. Função Meio-Quadrado

Neste método se pega a chave, calcula o quadrado da chave e pega uma parte do meio do número. Como especificações para essa função deve-se definir o número de dígitos a serem retirados do meio da chave ao quadrado além do tamanho da tabela.

Exemplos:
(1) Para M = 8 e chave = 25.
Primeiro, eleva a chave ao quadrado, o que dá 625.

Como os índices da tabela comportam um dígito, tem de pegar um dígito do meio, portanto h(25) = 2.
(2) Para M = 1000 e chave = 3121.
$(3121)^2$ = 9740641, h(3121) = 406 (válido no intervalo 0 e 999).

2.5. Extração

Aqui se pega parte da chave para calcular o endereço. Se essa parte é escolhida de forma eficiente, esse método é suficiente para o Hashing. Sabe-se que alguns códigos mantêm alguns dígitos iguais para certos grupos, por exemplo: no ISBN os primeiros dígitos representam o editor. Se estivermos catalogando dados de somente um editor, essas partes das chaves devem ser descartadas.

As especificações são: como escolher a chave e o tamanho da tabela.

Exemplos:
(1) Para M = 1700 e chave = 123-45-6789, pode pegar os dois primeiros dígitos e combinar com os dois últimos (**12**3-45-67**89**) e aí teremos: h(chave) = 1289.

(2) Para M = 100 e chave = 123-45-6789, pode pegar o primeiro dígito e combinar com o último (**1**23-45-678**9**) e aí teremos: h(chave) = 19.

2.6. Transformação da raiz

Pega-se a chave, transforma essa chave em outra base, divide-o pelo tamanho da tabela e pega o resto dessa divisão. As especificações são: qual a base usada e o tamanho da tabela.

Exemplos
(1) chave = 345, Base = 9 e N = 100. Portanto $345 = (423)_9$ e assim h(x) = 423 % 100 = 23
(2) chave = 264, Base = 8 e N = 100. Portanto $264 = (410)_8$ e assim h(x) = 410 % 100 = 10

2.7. Transformação de chaves alfanuméricas

Os métodos acima são usados diretamente quando as chaves são numéricas. Aqui transformaremos as chaves alfanuméricas para uma representação numérica primeiro e depois podemos aplicar os métodos acima.

Uma forma simples de transformar uma chave alfanumérica num valor numérico consiste em considerar cada caracter da chave como um valor inteiro (correspondente ao seu código ASCII) e realizar uma soma com todos eles.

```
int transfch(char *chave, int M)
{   int i, soma=0;
    for(i=0;i<strlen(chave);i++) soma += chave[i];
    return((soma % M));        // Método da Divisão
}
```

Exemplos:
(1) Para M = 8 e chave Simone, h(chave) = (83+105+109+111+110+101) % 8 = (619) % 8 = 3
(2) Para M = 8 e chave Barbara, h(chave) = (66+97+114+98+97+114+97) % 8 = (683) % 8 = 3
(3) Para M = 8 e chave Sueli, h(chave) = (83+117+101+108+105) % 8 = (514) % 8 = 2
(4) Para M = 8 e chabe Matheus, h(chave) = (77+97+116+104+101+117+115) % 8 = (727) % 8 = 7

Observe um problema sério: um conjunto de chaves alfanuméricas onde cada uma delas é apenas uma permutação dos mesmos caracteres básicos: ABC, ACB, BAC, BCA, CAB e CBA. Neste caso, se estas chaves forem espalhadas pela função acima numa tabela com M = 8 posições:

transfch ("ABC") = 6 transfch("ACB") = 6
transfch ("BAC") = 6 transfch("BCA") = 6
transfch ("CAB") = 6 transfch("CBA") = 6

Assim como está, todas as chaves serão alocadas num mesmo índice e portanto não haverá espalhamento quando se tem chave que usam as letras combinadas de forma diferentes.
Para resolver esse problema pode-se fazer um somatório com deslocamentos. Este algoritmo associa a cada caractere da chave uma quantidade de bits que deverá ser deslocada para a esquerda no seu código ASCII, antes de ele ser adicionado a soma total. Assim o índice dependerá também da posição que as letras ocupam na chave.

Por exemplo, para uma chave que possua N caracteres e uma tabela de tamanho 8, as quantidades a serem deslocadas a esquerda devem variar de 0 a 7, de forma cíclica, então tem o esboço abaixo:

Posição	1ª.	2ª.	3ª.	4ª.	5ª.	6ª.	7ª.	8ª.	9ª.	10ª. ...
Deslocamento	0	1	2	3	4	5	6	7	0	1 ...

Se a letra A aparece na primeira posição da chave, então o seu valor numérico será o seu próprio código ASCII, pois, para a primeira posição, o deslocamento é 0 e nada será alterado. Porém, se ela aparecer na segunda posição, seu código devera ser deslocado de 1 bit à esquerda; se ele aparecer na terceira, 2 bits; na quarta, 3 e assim sucessivamente. Assim cada caracter será deslocado para a esquerda de [(posição-1) % (M)] bits. Observe que M tem de ser múltiplo de 2.

Vejamos para M = 8, como ficam os deslocamentos para as letras A, B e C.

A = 65 com deslocamento 0 , A = 65 $(01000001)_2$
A = 65 com deslocamento 1 , A = -126 $(10000010)_2$
A = 65 com deslocamento 2 , A = 4 $(00000100)_2$
A = 65 com deslocamento 3 , A = 8 $(00001000)_2$
A = 65 com deslocamento 4 , A = 16 $(00010000)_2$
A = 65 com deslocamento 5 , A = 32 $(00100000)_2$
A = 65 com deslocamento 6 , A = 64 $(01000000)_2$
A = 65 com deslocamento 7 , A = -128 $(10000000)_2$
A = 65 com deslocamento 0 , A = 65 $(01000001)_2$

B = 66 com deslocamento 0 , B = 66 $(01000010)_2$
B = 66 com deslocamento 1 , B = -124 $(10000100)_2$
B = 66 com deslocamento 2 , B = 8 $(00001000)_2$
B = 66 com deslocamento 3 , B = 16 $(00010000)_2$
B = 66 com deslocamento 4 , B = 32 $(00100000)_2$
B = 66 com deslocamento 5 , B = 64 $(01000000)_2$
B = 66 com deslocamento 6 , B = -128 $(10000000)_2$
B = 66 com deslocamento 7 , B = 0 $(00000000)_2$
B = 66 com deslocamento 0 , B = 66 $(01000010)_2$

C = 67 com deslocamento 0 , C = 67 $(01000011)_2$
C = 67 com deslocamento 1 , C = -122 $(10000110)_2$
C = 67 com deslocamento 2 , C = 12 $(00001100)_2$
C = 67 com deslocamento 3 , C = 24 $(00011000)_2$
C = 67 com deslocamento 4 , C = 48 $(00110000)_2$
C = 67 com deslocamento 5 , C = 96 $(01100000)_2$
C = 67 com deslocamento 6 , C = -64 $(11000000)_2$
C = 67 com deslocamento 7 , C = -128 $(10000000)_2$
C = 67 com deslocamento 0 , C = 67 $(01000011)_2$

Em C, utilizam-se os operadores << (shift to left) e >> (shift to right) para se deslocar bits à esquerda e à direita, respectivamente. Perceba que para 8 bits o último bit é o do sinal e está em complemento de 2. Veja como fica a rotina:

```
int transfch(char *chave, int M)
{ int i, soma=0;
  char valor;
  for(i=0;i<strlen(chave);i++){
      valor = (chave[i]<<(i%(M)));
      soma += abs(valor);}
   return((soma % M));   // Método da Divisão
}
```

H("ABC") = (65+124+12) % 8 = (201) % 8 = 1
H("BAC") = (66+126+12) % 8 = (204) % 8 = 4
H("CAB") = (67+126+8) % 8 = (201) % 8 = 1
H("ACB") = (65+122+8) % 8 = (195) % 8 = 3
H("BCA") = (66+122+4) % 8 = (192) % 8 = 0
H("CBA") = (67+124+4) % 8 = (195) % 8 = 3

H("Simone") = (83+46+76+120+32+96) % 8 = (453) % 8 = 5
H("Barbara") = (66+62+56+16+16+64+64) % 8 = (344) % 8 = 0
H("Sueli") = (83+22+108+96+112) % 8 = (421) % 8 = 5
H("Matheus") = (77+62+48+64+80+96+64) % 8 = (491) % 8 = 3

Embora o somatório com deslocamentos apareça como uma forma de resolver o problema de chaves com permutação, pode-se empregá-lo sempre que for necessário para manipular chaves alfanuméricas; ainda que elas não sejam permutações do mesmo conjunto de caracteres.

3. TRATAMENTO DE COLISÃO

Quando acontecem colisões é porque existem mais de um elemento a ser colocado no mesmo índice, no mesmo encaixe, na mesma classe. É isso que os métodos abaixo vão tentar resolver, ou seja, qual o procedimento quando surgem elementos sinônimos.

Aqui são estudados os métodos: Encadeamento (Interior e Exterior) e Endereçamento Aberto (Tentativa linear, Tentativa quadrática e Dispersão Dupla).

3.1. Encadeamento

No encadeamento busca-se resolver o conflito criando estruturas dentro da tabela. A tabela não fica sendo simplesmente um vetor do tipo da chave. Aparecem outras estruturas como uma estrutura com um campo para a chave e outro campo para a referência (Encadeamento Interior) ou um vetor de listas encadeadas (Encadeamento Exterior)

3.1.1. Encadeamento Interior

No encadeamento interior busca-se resolver o problema compartilhando o mesmo espaço de memória que a tabela Hashing. Assim os elementos da tabela conterão dois valores. Um conterá a chave e o outro uma referência ao seu sinônimo.

A partir daí são visualizadas duas soluções. **Uma que separa parte da tabela Hashing para armazenar os sinônimos e outra que não faz essa separação**.

Veja essas duas soluções para uma tabela Hashing para as chaves: 48, 03, 80, 31, 20.
Usando divisão inteira por 4 (figura01(a)) tem os índices: 0, 3, 0, 3, 0.
Usando divisão por 7 (figura01(b)) tem os índices 6, 3, 3, 3, 6.

Figura01. Tabela Hashing - Encadeamento interior – exemplo 01

0	48	4
1		
2		
3	03	5
4	80	6
5	31	-1
6	20	-1

(a) N = 7, M = 4, b = 4 e s = 3

0		
1		
2	20	-1
3	03	5
4	31	-1
5	80	4
6	48	2

(b) N = M = 7

Na tabela da figura01(a) surge uma tabela Hashing com tamanho 7, só que houve separação da tabela deixando 4 posições (b = 4) para as chaves bases e 3 posições (s = 3) para as chaves sinônimas. Portanto, a função hashing – divisão inteira deve ter M=4.

O problema acontece quando uma das áreas é preenchida totalmente gerando um **falso estouro** da tabela. Por exemplo, se quisesse inserir a chave 12 - resultaria num índice 0. Só que não poderia fazer a inserção na tabela, já que não tem espaço na parte dos sinônimos. Só que ao tentar inserir os números 5 (h(5)=1) e 18 (h(18) = 2), a inserção poderia ser feita sem problema porque há espaço na parte das chaves bases.

Na tabela (b) da figura01 tem uma tabela Hashing com tamanho 7 e que não diferencia as duas partes da tabela. Assim qualquer valor na tabela pode ser uma chave base ou sinônima. A inserção dos sinônimos ocorre do final da tabela para cima. Assim não tem o problema de estouro falso da tabela.

Imagine que se queira inserir a chave 9. Pela função de Hashing essa chave deveria ocupar o índice 2, só que já está ocupado. Essa colisão é chamada de **colisão secundária**. A solução é inserir a chave numa posição livre da tabela e fazer o encadeamento a partir do índice original. Se a referência desse índice for -1 basta trocar -1 pelo índice onde o elemento está ocupando (veja figura02(a)). Se o índice não for -1 deve-se procurar o último elemento dessa cadeia e fazer a ligação. Por exemplo, ao inserir a chave 19 (indice = 5) na tabela (a) da figura02 teremos a tabela(b) da figura02.

Figura02. Tabela Hashing - Encadeamento interior – exemplo 02

0		
1	9	-1
2	20	1
3	03	5
4	31	-1
5	80	4
6	48	2

(a) M = 7, inserção da chave 9

0	19	-1
1	9	-1
2	20	1
3	03	5
4	31	0
5	80	4
6	48	2

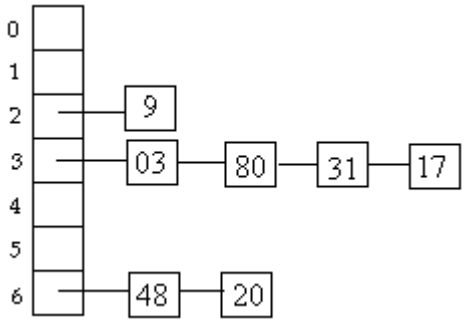
(b) M = 7, inserção do 19

3.1.2. Encadeamento Exterior

No encadeamento exterior usa-se uma estrutura fora da tabela para armazenar as chaves sinônimas. Normalmente se usa uma tabela de ponteiros para listas encadeadas. Veja a tabela da Figura02(b) representada como encadeamento exterior na Figura03 abaixo.

No encadeamento exterior a manipulação da tabela se reduz a aplicação da função Hashing e o uso de um vetor de listas encadeadas.

Figura03. Tabela Hashing - Encadeamento exterior



3.2. Endereçamento Aberto

No endereçamento aberto não se têm dois campos como elemento da tabela (como foi no caso do Encadeamento Interno) e nem um vetor de listas (no caso do Encadeamento Externo). Aqui cada posição da tabela possui somente a chave. Quando há colisão, determina-se por uma função a posição dentro da tabela que essa chave irá ocupar. Se esse próximo lugar também está ocupado, aplica-se a função novamente e encontra-se um novo lugar. O processo fica sendo executado até encontrar uma posição vazia para a chave. Perceba que é trabalhada a ideia de uma tabela circular.

Dessa forma são feitas tentativas, aplicando a função Hashing, que agora terá a forma de h(x,k), onde k pode variar de 0 a M-1, já que numa primeira tentativa aplicamos h(x,0), se não for uma posição vazia, aplica-se o h(x,1) e assim por diante. Dessa forma a função de Hashing é capaz de fornecer até M resultados para o índice.

A sequencia de tentativas { H(x,0), H(x,1), ..., H(x,M-1) } é da forma:

$$H(x,k) = (h(x) + c(k)) \% M$$

Vamos ver como calcular essa nova função.

3.2.1. Tentativa Linear

Na tentativa linear é feita **c(k) = k**. Então,

$$H(x,k) = (h(x) + k) \% M$$

Assim ao encontrar o índice onde a chave deveria ocupar, mas no índice já tem outra chave, procura-se de forma linear, ou seja, procura-se daquele ponto para baixo qual é a posição vazia que a nova chave poderia ocupar. A tentativa termina quando encontrar uma posição vazia ou quando voltar à posição inicial de busca.

Exemplo:
Seja o conjunto de chaves {44, 46, 49, 68, 71, 97, 26, 72, 27, 96} a serem inseridas numa tabela Hashing de tamanho 23 usando Divisão Inteira para h(x).
- Aplicando H(x,0) = (h(x)+0) % 23 teríamos os índices: { **21, 0, 3, 22, 2, 5**, 3, 3, 4, 4}
- Assim, até a inserção da chave 97 não ocorre colisão (veja Figura04(a)).
- Quando chega à chave 26, obtém-se H(26,0) = 3, que já está ocupado pela chave 49. Aí é procurada a próxima posição:

H(26,1) = (h(26)+1)%23 = 4 % 23 = 4,

que está livre. Então insere a chave 26 nessa posição.
- Ao inserir a chave 72: calcula-se a primeira possibilidade: H(72,0) = 3, que já está ocupado. Tenta-se outra vez,

H(72,1) = (h(72)+1)%23 = 4 % 23 = 4, que tem o elemento 26.
H(72,2) = (h(72)+2) % 23 = 5 % 23 = 5, que tem o elemento 97.
H(72,3) = (h(72)+3) % 23 = 6 % 23 = 6, que está vazia.

Aí o elemento 72 é inserido nessa posição.
- Quando for inserir a chave 27, que deveria ser colocado no índice 4, acaba sendo colocado no índice 7 que é o próximo vazio na tabela, pois:

H(27,1) = (h(27)+1)%23 = 4 % 23 = 4, que tem o elemento 26.
H(27,2) = (h(27)+2) % 23 = 5 % 23 = 5, que tem o elemento 97.
H(27,3) = (h(27)+3) % 23 = 6 % 23 = 6, que tem o elemento 72.
H(27,3) = (h(27)+4) % 23 = 7 % 23 = 7, que está vazia.

- O mesmo vai acontecer com a chave 96, que tem índice original igual a 4, mas só vai ser inserido no índice 8.

Figura04. Tabela Hashing - Endereçamento Aberto - Tentativa Linear

0	46
1	
2	71
3	49
4	
5	97
6	
7	
8	
...	...
21	44
22	68

(a)

0	46
1	
2	71
3	49
4	**26**
5	97
6	**72**
7	**27**
8	**96**
...	...
21	44
22	68

(b)

Percebe-se que nesse tipo de tentativa acaba por gerar agrupamentos (chamado agrupamento primário) que pode ocasionar comprometimento na eficiência da busca usando Hashing.

3.2.2. Tentativa Quadrática

Na Tentativa Linear usou-se **c(k) = k** e percebeu-se que podem acontecer agrupamentos primários. Para tentar evitar esse tipo de agrupamento usa-se a Tentativa Quadrática que tem em c(k) uma função quadrática do tipo:

$$c(k) = a_1*k^2 + a_2*k + a_3$$

Só que como para k = 0, c(k) tem de ser igual a zero (para obter a primeira tentativa), teremos de adotar $a_3 = 0$. Assim,

$$c(k) = a_1*k^2 + a_2*k$$

Portanto:

$$H(x,k) = (h(x) + a_1*k^2 + a_2*k) \% M$$

Exemplo01: Considerando $a_1 = 1$ e $a_2 = 0$ ($c(k) = k^2$) e o exemplo anterior com M = 23, h(x) sendo divisão inteira e o conjunto das chaves = {44,46,49,68,71,97,26,72,27,96}, veja como ficará o espalhamento, sabendo que até a inserção do 97 ocorre tudo certo (Figura05(a)).

Chave = 26 H(26,0) = 3 (posição ocupada)
H(26,1) = $(h(26) + 1^2)\%23 = (3 + 1) \% 23 = 4$ (posição vazia, então o 26 será inserido)

Chave = 72 H(72,0) = 3 (posição ocupada)
H(72,1) = $(h(72) + 1^2)\%23 = (3 + 1) \% 23 = 4$ (posição ocupada)
H(72,2) = $(h(72) + 2^2)\%23 = (3 + 4) \% 23 = 7$ (posição vazia, então o 72 será inserido)

Chave = 27 H(27,0) = 4 (posição ocupada)
H(27,1) = $(h(27) + 1^2)\%23 = (4 + 1) \% 23 = 5$ (posição ocupada)
H(27,2) = $(h(27) + 2^2)\%23 = (4 + 4) \% 23 = 8$ (posição vazia, então o 27 será inserido)

Chave = 96 H(96,0) = 4 (posição ocupada)
H(96,1) = $(h(96) + 1^2)\%23 = (4 + 1) \% 23 = 5$ (posição ocupada)
H(96,2) = $(h(96) + 2^2)\%23 = (4 + 4) \% 23 = 8$ (posição ocupada)
H(96,2) = $(h(96) + 3^2)\%23 = (4 + 9) \% 23 = 13$(posição vazia, então o 96 será inserido)

Veja o resultado da inserção na Figura05(b).

Figura05. Tabela Hashing - Endereçamento Aberto - Tentativa Quadrática

0	46
1	
2	71
3	49
4	
5	97
6	
7	
8	
9	
10	
11	
12	
13	
...	...
21	44
22	68

(a)

0	46
1	
2	71
3	49
4	**26**
5	97
6	
7	**72**
8	**27**
9	
10	
11	
12	
13	**96**
...	...
21	44
22	68

(b) $a_1 = 1$ e $a_2 = 0$

0	46
1	
2	71
3	49
4	**27**
5	97
6	**96**
7	
8	
9	
10	
11	**26**
12	**72**
13	
...	...
21	44
22	68

(c) $a_1 = 2$ e $a_2 = 0$

Exemplo02: Considerando $a_1 = 2$ e $a_2 = 0$ ($c(k) = 2*k^2$) e o exemplo anterior com M = 23 e o conjunto das chaves = {44,46,49,68,71,97,26,72,27}, veja como ficará o espalhamento. Foi isto que até a inserção do 97 estava tudo certo (Figura05(a)). Agora veja as inserções das outras chaves.

Chave = 26 H(26,0) = 3 (posição ocupada)
H(26,1) = (h(26) + $2*1^2$)%23 = (3+2) %23 = 5 (posição ocupada)
H(26,2) = (h(26) + $2*2^2$)%23 = (3+8) % 23 = 11 (posição vazia, então o 26 será inserido)

Chave = 72 H(72,0) = 3 (posição ocupada)
H(72,1) = (h(72) + $2*1^2$)%23 = (3+2) % 23 = 5 (posição ocupada)
H(72,2) = (h(72) + $2*2^2$)%23 = (3+8) % 23 = 11 (posição ocupada)
H(72,3) = (h(72) + $2*3^2$)%23 = (3+18) % 23 = 21 (posição ocupada)
H(72,4) = (h(72) + $2*4^2$)%23 = (3+32) % 23 = 12 (posição vazia, então o 72 será inserido)

Chave = 27 H(27,0) = 4 (posição vazia, então o 27 é inserido nessa posição)

Chave = 96 H(96,0) = 4 (posição ocupada)
H(96,1) = (h(96) + $2*1^2$)%23 = (4+2)%23 = 6 (posição vazia, então o 96 será inserido)

Veja o resultado da inserção na Figura05(c).

Perceba que a escolha de a_1 e a_2 é importante.

3.2.3. Tentativa por Dispersão Dupla

Na dispersão dupla a sequencia de tentativas usa a seguinte equação:

$$H(x,k) = (h(x) + k*g(x)) \% M$$

Usando o mesmo exemplo, M = 23 e o conjunto das chaves = {44,46,49,68,71,97,26,72,27}, e escolhendo h(x) = x % 23 e g(x) = 1 + (x % 13). Foi visto que até a inserção do 97 estava tudo certo (Figura06(a)). Veja a inserção das outras chaves.

Chave = 26 H(26,0) = 3 (posição ocupada)
H(26,1) = (h(26) + 1*g(26))%23 = (3+1*(1+(26%13)))%23 = (3+1*(1+0))%23 = (3+1*(1))%23 = (3+1)%23 = 4 (posição vazia)

Chave = 72 H(72,0) = 3 (posição ocupada)
H(72,1) = (h(72) + 1*g(72))%23 = (3+1*(1+7))%23 = (3+7)%23 = 10 (posição vazia)

Chave = 27 H(27,0) = 4 (posição ocupada)
H(27,1) = (h(27) + 1*g(27))%23 = (4+1*(1+1))%23 = (4+2)%23 = 6 (posição vazia)

Chave = 96 H(96,0) = 4 (posição ocupada)
H(96,1) = (h(96) + 1*g(96))%23 = (4+1*(1+5))%23 = (4+6)%23 = 10 (posição ocupada)
H(96,2) = (h(96) + 2*g(96))%23 = (4+2*(1+5))%23 = (4+12)%23 = 16 (posição vazia)

Veja o resultado da inserção na Figura06(b).

Figura06. Tabela Hashing - Endereçamento Aberto - Dispersão Dupla

0	46
1	
2	71
3	49
4	
5	97
6	
7	
8	
9	
10	
...	...
16	
...	...
21	44
22	68

(a)

0	46
1	
2	71
3	49
4	**26**
5	97
6	**27**
7	
8	
9	
10	**72**
...	...
16	**96**
...	...
21	44
22	68

(b)

4. EXERCÍCIOS

1. Suponha um conjunto de chaves {0, 7, 14, 21, 28, 35, 42, 49, 56, 63, 70}. Aplique a função Hashing Divisão Inteira para as seguintes tabelas:
(a) M = 5 (b) M = 7 (c) M = 11
Para cada tabela, verifique em quantas posições haverá colisões.

2. Suponha um conjunto de chaves {12, 44, 13, 88, 23, 94, 11, 39, 20, 16, 5}. Aplique a função Hashing Divisão Inteira para as seguintes tabelas:
(a) M = 5 (b) M = 7 (c) M = 11
Para cada tabela, verifique em quantas posições haverá colisões.

3. Usando a função Hashing Divisão Inteira e 7 posições, descubra quais os resultados da aplicação dessa função para as chaves: 19, 17, 20, 30, 97, 58, 15, 24, 36.

4. Uma determinada tabela não pode ter mais que 10 encaixes e as chaves estão entre 0 e 64. Codifique a função de Hashing (meio-quadrado) para realizar o espalhamento das chaves nessa tabela.

5. Usando as chaves: {12, 44, 13, 88, 23, 94, 11, 39, 20, 16, 5} e uma tabela de 11 posições, aplique as funções Hashing abaixo para calcular os seus índices na tabela.
(a) Meio-Quadrado (b) Extração (c) Transformação da Raiz

6. Refaça o exercício 05 para as chaves:
(a) 11, 39, 20, 5, 16, 44, 88, 12, 23, 13, 94
(b) 25, 19, 13, 39, 44, 89, 11, 23, 13, 95
(c) 11, 23, 20, 16, 39, 44, 94, 12, 88, 13, 5

7. Usando as chaves: {7264935-8, 9391153-4, 274686-5} e uma tabela de 14 posições, aplique as funções Hashing abaixo para calcular os seus índices na tabela.
(a) Enlaçamento deslocado (b) Enlaçamento limite (c) Dobra

8. Usando o método de **Enlaçamento Deslocado**, numa tabela que tem 100 posições, descubra qual o resultado da aplicação dessa função hashing para as chaves 35.844.766-10 e 8-236-542-0306

9. Use os métodos (1) **Enlaçamento Deslocado** e (2**) Dobra**, numa tabela que tem 110 posições, descubra quais os resultados da aplicação dessas funções para as chaves 7-896919-606926 e 978-85-7942-939-6

10. Use o método **Enlaçamento Limite**, numa tabela que tem 59 posições, descubra quais os resultados da aplicação dessa função para as chaves 7-896719-608226; 978-95-7772-989-6

11. Use o método da **Dobra**, numa tabela de 100 posições, e descubra os resultados da aplicação para as chaves: 7-894977-531088; 7-836363-5341384.

12. Use o método **Meio Quadrado**, numa tabela que tem 15 posições, descubra quais os resultados da aplicação da função para as chaves 39, 220.

13. Usando o método de **Extração**, numa tabela que tem 100 posições, descubra os resultados da aplicação dessa função hashing para as chaves 35.864.726-10 e 8-236-532-0006

14. Usando o método de **Transformação da raiz**, numa tabela de 100 posições, descubra os resultados da aplicação para as chaves: 588, 951, 365. Use a base 8 para o cálculo.

15. Usando as chaves do exercício 01: {0, 7, 14, 21, 28, 35, 42, 49, 56, 63, 70} com função Hashing divisão e tabela de tamanho 5, como fica a tabela que possui tratamento de colisão pelo método:
(a) Encadeamento interior
(b) Encadeamento exterior
(c) Endereçamento aberto, tentativa linear
(d) Endereçamento aberto, tentativa quadrática com $a_1 = 3$ e $a_2 = 2$
(e) Endereçamento aberto, dispersão dupla com g(x) = x mod 5

16. Usando a função Hashing Divisão Inteira, o tratamento de colisão por **Endereçamento Aberto por Tentativa Quadrática** (a1 = 3 e a2 = 1) e M = 13, mostre como fica a tabela hashing para a sequência de chaves: 19, 9, 17, 20, 30, 97, 58, 15, 24, 36.

17. Usando a função Hashing Divisão Inteira e fazendo o tratamento de colisão por **Encadeamento Interior com uma região** e M = 13, mostre como fica a tabela hashing para a sequência de chaves: 19, 9, 17, 20, 30, 97, 58, 15, 24, 36.

18. Usando a função hashing divisão (inteira) e fazendo o tratamento de colisão por **Encadeamento Interior com o uso de duas regiões** (uma de tamanho 10 e outra de tamanho 5), mostre como fica a tabela hashing para a seqüência de chaves: 19, 9, 17, 25, 90, 36 e 58.

19. Desenhe o conteúdo da tabela hash resultante da inserção de registros com as chaves Q U E S T A O F C I L, nesta ordem, em uma tabela inicialmente vazia de tamanho 13 (treze) usando endereçamento aberto – tentativa linear para a escolha de localizações alternativas. Use a função hashing h(k) = k%13 para a K-ésima letra do alfabeto.

20. Refaça o exercício anterior substituindo as chaves Q U E S T A O F C I L pelas 12 primeiras letras do seu nome, desprezando brancos e letras repetidas. Por exemplo, para Simone Domingues Prado teria: S I M O N E D G U S P R.

5. BIBLIOGRAFIA

ASCENCIO, Ana Fernanda Gomes. **Estruturas de dados: algoritmos, análise da complexidade e implementações em Java e C/C++** - São Paulo: Pearson Prentice Hall, 2010.

DROZDEK, Adam. **Estruturas de Dados e Algoritmos em C++**. Tradução de Luiz Sérgio de Castro Paiva; Revisão de Flávio Soares Corrêa da Silva. São Paulo: Pioneira Thomson Learning, 2002. 579p.

PEREIRA, Silvio do Lago. **Estruturas de Dados Fundamentais: Conceitos e Aplicações**. São Paulo: Érica, 1996. 246p.

PREISS, Bruno R. **Estruturas de Dados e Algoritmos: Padrões de projetos orientados a objeto com Java.** Tradução: Elizabeth Ferreira. Rio de Janeiro: Campus, 2000. 566p.

SCHILDT, Herbert. **C, Completo e Total**. 3a.ed. Tradução e Revisão Técnica: Roberto Carlos Mayer. São Paulo: Pearson Education do Brasil, 1997. 827p.

SZWARCFITER, James Luiz & MARKENZON, Lilian. **Estruturas de Dados e seus Algoritmos**. Rio de Janeiro: LTC-Livros Técnicos e Científicos Ed., 1994. 320p.

TENENBAUM, Aaron M; LANGSAM, Yedidyah; AUGENSTEIN, Moshe J**. Estruturas de Dados usando C**. Tradução: Teresa Cristina Félix de Souza; Revisão Técnica e Adaptação de Programas: Roberto Carlos Mayer. São Paulo: Makron Books, 1995. 884p.

ZIVIANE, Nívio. **Projeto de algoritmos: com implementações em Pascal e C** – 3ª ed. Ver. Ampl. São Paulo: Cengage Learning, 2011.