**HASHING**

Hashing é um método usado na recuperação ou na segurança de dados. Em sistemas gerenciadores de bancos de dados é necessário utilizar um método de busca rápido para localizar dados identificados por um valor chave. Nos compiladores ou interpretadores, as tabelas que contém os identificadores definidos nos programas devem permitir a localização imediata dos mesmos. Mensagens, arquivos e documentos enviados pela Internet devem ser identificados de modo único.

A idéia do método hashing é associar ao valor chave que está sendo pesquisado um código, que é um número inteiro, se possível, de modo único (sobrejeção). A função que calcula esse código é denominada função hash. A utilização de uma técnica de espalhamento (hashing) faz com que o procedimento de busca seja feito em tempo constante O(1), porque exige somente o cálculo de h(v) para qualquer valor da chave.

Um conjunto de dados pode ser armazenado em uma tabela de dispersão (hash table), que é uma estrutura de armazenamento de dados em que o código hash identifica a localização dos dados na estrutura.

Por exemplo, em um cadastro contendo número de matrícula e nome, além de outros dados, teríamos o campo número de matrícula como chave, porque identifica univocamente cada registro.

Se o número de matrícula tem o formato “RAxxxxxxxx”, com 8 dígitos numéricos após a cadeia “RA”, a parte do código de matrícula formada pelos 8 dígitos pode ser usada como chave de busca. O universo de chaves é o conjunto de códigos de 8 dígitos e contém 100.000.000 valores possíveis.

typedef struct{

char nMat[10];

char nome[81];

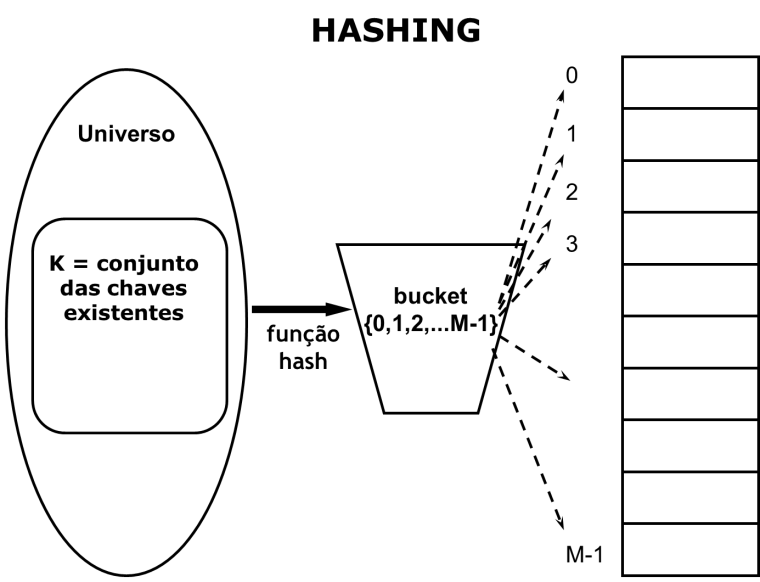
char email[41];

}Aluno;

Usando os 8 dígitos como código hash associado ao número de matrícula de um aluno, o conjunto de dados poderia ser armazenado em uma estrutura de acesso direto indexada com números de 0 até 99.999.999 para ter índices de 0 a 99.999.999. Com isso, teríamos uma reserva de espaço na memória de pelo menos 132x100.000.000 = 13.200.000.000 bytes, que corresponde a mais de 10 GB. A função hash neste caso, é h(v) = número formado pelos 8 últimos algarismos do número de matrícula.

Podemos utilizar apenas uma parte do código de matrícula como chave de busca, por exemplo, os 4 últimos algarismos. A estrutura de dados, neste caso, poderia ter espaço para 10000 registros (índices de 0 a 9999). Com isso, a reserva de espaço seria de pelo menos 1.320.000 bytes, cerca de 1,3 MB.

Mas diversos números de matrícula podem ter os mesmos 4 últimos dígitos (RA00004092 e RA00104092, por exemplo). A isso, chamamos de colisão e neste caso, é necessário fazer o tratamento das colisões, isto é, resolver em que local armazenamos os registros que são identificados pelo mesmo índice na tabela e como recuperar esses dados.



**TRATAMENTO DE COLISÕES**

Denominamos bucket a cada valor do contra-domínio da função hash e todas as chaves que são associadas a um mesmo valor pertencem a um mesmo bucket.

O tratamento de colisões é um critério de decisão que estabelece qual o novo valor que será associado à chave sinônima. Há alguns métodos clássicos de tratamento de colisões:

1) Por encadeamento – a cada bucket corresponde uma lista ligada contendo os registros de chaves sinônimas.

2) Por endereçamento aberto com novo localizador: teste linear, quadrático ou de um novo hashing.

3) Por endereçamento aberto, com encadeamento e com tabela de descritores.

**FUNÇÃO HASH**

A função hash escolhida determina o algoritmo hashing. Há alguns métodos para a criação de funções hash, exemplificados a seguir. A função hash depende do tipo de dados da chave usada para a busca.

**(I) A chave é um valor numérico**

**Exemplo 1** - A chave é um número inteiro e m um número primo.

h(v) = v MOD m, m = quantidade de buckets (número de posições na tabela hash)

m = 701, h(v) = v MOD 701

v = 35775  h(35775) = 35775 MOD 701 = 24

**Exemplo 2** - A chave não é um número inteiro e pertence ao intervalo [s,t], m é um número primo.

h(v) = ((v-s)/(t-s))m

m = 701, v  [1,10], h(v) = ((v-1)/9)x701

v = 3.523  h(3.523) = (2.523/9)x701 = 196.5136667 = 196

**Exemplo 3** – m não é um número primo

h(v) = [Av] MOD m, A = 0.618033

m = 700, h(v) = 0.618033v MOD 700

v = 35775  h(35775) = 0.618033 x 35775 MOD 700 = 22110.131 MOD 700 = 22110 MOD 700 = 410

**(II) A chave é uma cadeia de caracteres.**

Inicialmente, convertemos a cadeia de caracteres em um número inteiro, utilizando o código ASCII de cada caractere.

**Exemplo 1**

m = 31, v = NOW  v = 78,79,87



h(NOW) = (78 + 79 + 87) MOD 31 = 238 MOD 31 = 21

**Exemplo 2**

m = 31, v = NOW  v = 78,79,87



h(NOW) = (1x78 + 2x79 + 3x87) MOD 31 = 497 MOD 31 = 1

**Exemplo 3** – método da dobra

m = 31, v = CAIXA  v = 67,65,73,88,65

67 + 65 = 132 32,61,65 32 + 61 = 93 93,65 93 + 65 = 168

73 + 88 = 161

h(CAIXA) = 68 MOD 31 = 6

**Exemplo 4**

M = 127, v = CAIXA  v = 67,65,73,88,65



h(CAIXA) = (67 x 1284 + 65 x 1283 + 73 x 1282 + 88 x 128 + 65) MOD 127 = 104

**III) Funções hash usadas em tratamento de colisões em hashing por endereçamento aberto com novo localizador.**

**Exemplo 1**

m = 11, i = teste, v = 14



h(14,0) = 14 MOD 11 = 3 h(14,1) = 15 MOD 11 = 4

**Exemplo 2**

m = 11, i = teste, v = 14



h(14,0) = 14 MOD 11 = 3 h(14,1) = 18 MOD 11 = 7 h(14,2) = 28 MOD 11 = 6

**Exemplo 3**

m = 11, p = 13, i = teste, v = 14



h(14,0) = (14 MOD 13) + 0x(1+14 MOD 11) = 1

h(14,1) = (14 MOD 13) + 1x(1 + 14 MOD 11)= 5

h(14,2) = (14 MOD 13) + 2x(1 + 14 MOD 11) = 9

**EXERCÍCIOS**

Considere as estruturas de dados definidas a seguir para a resolução das questões 1 e 2.

# define Max \_\_\_\_\_

typedef struct{

unsigned char NPUC[10];

unsigned char nome[80];

unsigned char email[30];

unsigned char turma[4];

}Aluno;

Aluno tabela[Max];

1. Supondo que o número de matrícula de um aluno seja composto por 8 dígitos numéricos, qual o espaço de memória (em bytes) necessário para o armazenamento do array tabela, para todos os possíveis números de matrícula?
2. 122 bytes
3. 1 Kbyte
4. 1 Mbyte
5. 1 Gbyte
6. 1 Tbyte

*1 Kbyte = 1024 bytes, 1 Mbyte = 1024 Kbyte, 1 Gbyte = 1024 Mbyte, 1 Mbyte = 1024 Gbyte, 1 Tbyte = 1024 Mbyte*

1. Se o cadastro de alunos não tem mais do que 1000 alunos, qual o espaço de memória (em bytes) necessário para o armazenamento do array tabela?

Resposta: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Para cada função hash definida a seguir, determine os localizadores calculados para as chaves da lista a seguir:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Chave | 26 | 44 | 46 | 49 | 68 | 71 | 97 | 118 | 182 |
| h1 |  |  |  |  |  |  |  |  |  |
| h2 |  |  |  |  |  |  |  |  |  |
| h3 |  |  |  |  |  |  |  |  |  |

int h1(TipoChave chave){ int h2(TipoChave chave){

int k; int k;

k = (chave%23) + 1; k = ((2\*chave+3)%23)) + 1;

return k; return k;

} }

int h3(TipoChave chave){

int q, aux, k;

q = chave\*chave;

aux = q/100;

k = ((q – aux \* 100) % 23) + 1;

return k;

}

1. Considere a estrutura de armazenamento de dados de uma tabela hash apresentada a seguir e preencha com os dados referentes à inserção das chaves 17, 26, 41, 44, 46, 55, 68, 71, 97, 118, 121, 182, 242, nesta ordem. Considere a função hash h(v) = (v % 11) + 1 e o tratamento das colisões por encadeamento com tabela de descritores.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | chave | next | ant |  |  | prim | ult |
| 1 |  |  |  |  | 1 |  |  |
| 2 |  |  |  |  | 2 |  |  |
| 3 |  |  |  |  | 3 |  |  |
| 4 |  |  |  |  | 4 |  |  |
| 5 |  |  |  |  | 5 |  |  |
| 6 |  |  |  |  | 6 |  |  |
| 7 |  |  |  |  | 7 |  |  |
| 8 |  |  |  |  | 8 |  |  |
| 9 |  |  |  |  | 9 |  |  |
| 10 |  |  |  |  | 10 |  |  |
| 11 |  |  |  |  | 11 |  |  |
| 12 |  |  |  |  |  |  |  |
| 13 |  |  |  |  |  |  |  |
| 14 |  |  |  |  |  |  |  |
| 15 |  |  |  |  |  |  |  |
| 16 |  |  |  |  |  |  |  |
| 17 |  |  |  |  |  |  |  |
| 18 |  |  |  |  |  |  |  |
| 19 |  |  |  |  |  |  |  |
| 20 |  |  |  |  |  |  |  |