UNIVERSIDADE FEDERAL DO MARANHÃO - UFMA

Estruturas de Dados II

Tabelas de dispersão (continuação)

Portela

Tabelas de dispersão (2)

- Funções de dispersão
- Tratamento de colisões
- Tabela de dimensão dinâmica

Tabelas de dispersão (2)

- Funções de Dispersão
  - Definição
  - Características
    - produzir um número baixo de colisões;
    - ser facilmente computável;
    - ser uniforme.

Tabelas de dispersão (2)

- Funções de Dispersão
  - Método da Divisão;
  - Método da Dobra;
  - Método da Multiplicação;
  - Método da Análise dos Dígitos.

Tabelas de dispersão (2)

- Método da Divisão

$$h(x) = h \bmod m$$

Tabelas de dispersão (2)

- Método da Dobra

  h(x) =

Método da dobra

- t = Tamanho da Dobra
  d = Total de 'dígitos' da chave
- int FuncDis(char * chave, int t, int d)
  {
  temp = chave // strcpy
  for(i = 0 ; i < (d/t)-1 ; ++i)
  {
  for(j = 0 ; j < t ; ++j)
  {
  temp[i*t+j+t] =
  soma(temp[i*t+j+t],temp[(i*t+j+t)-(2j+1)]);
  }
  }
  return(converte(temp[i*t ... i*t+t-1]));
  }
- char soma(char i, char j) // Soma ignorando vai-um
  converte() // Transforma 'string' em inteiro.

Método da dobra

- Chave: 279384
  t = 2
  d = 6

Método da Dobra:

- Dobra 27 em 93 e soma
  descartando 'vai-um'
- 7 + 9 = 6 (16 descartando 1) , 2+3 = 5
- Resultado Intermediário: 6584
- Dobra 65 em 84 e soma ...
- 5+8 = 3, 6+4 = 0
- Resultado Final: 30

- FuncDis("279384",2,6) = 30;

Tabelas de dispersão (2)

- Método da Multiplicação

  h(x) =

Método da Multiplicação

A chave pode ser multiplicada por ela mesma ou por uma constante, e ter o seu resultado armazenado em uma palavra de memória de *s* bits.

Tabelas de dispersão (2)

- Método da Análise dos Dígitos

  h(x) =

Método da Análise de Dígitos

- Acesso prévio a todas as chaves (população conhecida):
- Escolha, como endereço base, o valor do dígito
  na posição que resulta na menor quantidade de colisões.
- Acesso a uma amostra das chaves:
- Métodos de estimativa estatística.

Método da Análise de Dígitos

- Chaves: 443 462 492 681 712 972
- 1os Digitos: { 4 4 4 6 7 9 }
  -> 3 quatros, 1 seis, 1 sete e 1 nove.
  2os Digitos: { 4 6 9 8 1 7 }
  -> 1 um, 1 quatro, ... (adivinha o resto :-)
  3os Digitos: { 3 2 2 1 2 2 }
  -> 1 um, 4 dois,...
- Escolhendo o segundo dígito como endereço base não teríamos nenhuma colisão em uma tabela com 10 compartimentos [0..9]
- Já com o terceiro, teríamos 3.

Tabelas de dispersão (2)

- Tratamento de colisões
  - Por encadeamento exterior;
  - Por encadeamento interior;
  - Por endereçamento aberto.

Por encadeamento exterior

Chaves em colisão são armazenadas em uma lista encadeada, externa à tabela. Cada posição da tabela de dispersão armazena um ponteiro para o início desta lista.

Por encadeamento exterior

Exemplo:

Suponha que m = 10 e que estejamos usando o método da divisão, como função de dispersão.

Considerando a inserção das chaves {5 8 35 38 91 61 63 28}, na tabela dispersão, teríamos a seguinte situação:

	Tabela	Listas Ligadas
0	NULL	
1	--->	91 - 61
2	NULL	
3	--->	63
4	NULL	
5	--->	5 - 35
6	NULL	
7	NULL	
8	--->	8 - 38 - 28
9	NULL	

Por encadeamento interior

As 'listas encadeadas', com chaves em colisão, são armazenas 'dentro da própria tabela'.

Por encadeamento interior

Exemplo:

Utilizando as mesmas suposições do exercício anterior, e considerando a escolha da próxima posição disponível no vetor, depois do endereço base (com retorno ao início se necessário), para armazenar chaves em colisão.

NOK = No Key = Nenhuma Chave (-1)

EOL = End of List = Fim da Lista (-2)

	Tabela - chave (Prox.Elem.)
0	28 (EOL)
1	91 (2)
2	61 (EOL)
3	63 (EOL)
4	(NOK)
5	5 (6)
6	35 (EOL)
7	(NOK)
8	8 (9)
9	38 (0)

Tratamento de colisões por endereçamento aberto

Exige uma função de dispersão que forneça valores alternativos (até m) quando ocorre uma colisão. Com isto, não precisamos manter uma lista encadeada (economia de espaço).

Tratamento de colisões por endereçamento aberto

Exemplos

– Tentativa Linear;

– Tentativa Quadrática;

– Dispersão dupla.

Tentativa Linear

```
// Método da Tentativa Linear
    // k - Número da tentativa. (0,1,...m)
    int TentativaLinear(chave,m,k)
    {
        Base = FuncDis(chave,m)  // Uma função de dispersão
    qualquer.
        return(  (Base + k ) % m ) // Para cada tentativa devolve um
    endereço diferente.
    }
```

Tentativa Quadrática

```
// Método da Tentativa Quadrática
    // k - Número da tentativa. (0,1,...m)
    // Vantagem: Melhor Distribuição
    // Problema: Escolher c1 e c2 de forma que os valores sejam
    //                 diferentes para cada k.
    int TentativaQuadratica(chave,m,k)
    {
       Base = FuncDis(chave,m)  // Uma função de dispersão qualquer
       return(  (Base + c1*k + c2*k2 ) % m )
    }
```

Dispersão dupla

```
// Método da Dispersão Dupla
    // k - Número da tentativa. (0,1,...m)
    // Problema: Como garantir valores diferentes para cada k.
    int DispersaoDupla(chave,m,k)
    {
        Base1 = FuncDis1(chave,m)
        Base2 = FuncDis2(chave,m)
        return(  ( Base1 + k*Base2 ) % m ) // Para cada tentativa devolve um
    endereço diferente.
    }
```

Aplicações

Busca de elementos em base de dados:

- estruturas de dados em memória;
- bancos de dados;
- mecanismos de busca na Internet.

Verificação de integridade de dados grandes:

1. envio de dados com resultado da função *hash*;
2. receptor calcula função *hash* sobre dados recebidos e obtém novo resultado;
3. se resultados iguais, dados são iguais;
4. se diferentes, novo *download* é feito.

Exemplo: *download* imagem de disco do Linux.

Armazenamento de senhas com segurança:

Somente resultado função *hash* é armazenado no servidor;

Exemplos de *hash* (criptográficos):

*MD*5 e família *SHA* (*SHA* − 2, *SHA* − 256 e *SHA* − 512).

Aplicações: mecanismos de busca na Internet

spiders:

constroem listas de palavras dos Web sites;
começam busca em servidores mais utilizados e páginas populares;
começam busca em site popular, indexa suas palavras (e onde encontrou) e segue por seus links;

atenção especial: meta-tags, títulos, subtítulos;
Google: desconsidera artigos (“um”, “uma”, “o”, “a”);
Construção do índice: tabela hash;
usa peso: expressa número ocorrências de palavra, onde aparece (título, meta-tags, etc), se maiúscula, fonte, etc.

Combinação de indexação eficiente + armazenagem efetiva.