

Classificação e Pesquisa de Dados

Aula 13

Redimensionamento de Tabelas e Hashing Dinâmico;
Funções Dependentes de Distribuição

UFRGS

INF01124

Redimensionamento de Tabelas

◆ Quando redimensionar:

- O tamanho da tabela se torna insuficiente para acomodar as entradas
- A quantidade média de acessos para localizar as chaves se torna muito grande devido ao excesso de colisões
- Quando houver uma quantidade significativa de entradas marcadas como removidas

Como Redimensionar

◆ Por **realocação** ou **rehashing**

- Permitir a realocação das entradas segundo uma outra função de cálculo de endereço, de acordo com o novo tamanho da tabela (**maior** ou **menor**)
- Percorre-se a tabela original, inserindo na nova tabela todas as entradas ocupadas usando a nova função

◆ Por **realocação *in situ*** ou **realocação local**

- Proceder a realocação na mesma tabela agora redimensionada
- A tabela é aumentada ou diminuída na sua parte final

Realocação Local

◆ Contexto do Algoritmo

- Cada entrada da tabela possui um campo indicando: ocupada, livre ou realocada
- Somente entradas marcadas como ocupadas serão realocadas
- Entradas livres: nunca ocupadas ou excluídas
- Entradas realocadas: utilizadas para acomodar as chaves realocadas

Realocação Local (Cont.)

◆ Algoritmo

- Examinar todas as entradas da tabela original a partir da primeira
- Quando for encontrada uma entrada ocupada, ela é removida da tabela, seu novo endereço é calculado (usando a nova função), e a entrada é então reinserida
- Caso o novo endereço estava marcado como realocado, realiza-se uma busca linear a partir dele até encontrar-se a primeira entrada livre
- Caso o novo endereço esteja marcado como ocupado, o mesmo procedimento é aplicado recursivamente à chave ali instalada, podendo provocar uma cadeia de realocações.

Hashing Dinâmico

◆ Os métodos de organização de tabelas por cálculo de endereço vistos até agora, apesar de apresentarem um excelente desempenho médio, têm algumas limitações:

- Não são sensíveis ao conjunto de chaves efetivamente armazenadas
- A função de cálculo de endereço deve ser pré-definida e não pode ser modificada
- A adequação da função à evolução do perfil dos dados implica em uma completa reorganização da tabela
- As reorganizações são viáveis apenas em tabelas pequenas

Hashing Dinâmico

- ◆ O método *hashing* dinâmico tem como princípio a adequação dos mecanismos de cálculo de endereço à evolução do conjunto de dados

Tabela Estendível

◆ Objetivos

- Identificar as regiões da tabela onde está ocorrendo maior número de colisões
 - ◆ Proceder a expansão da tabela nessas regiões, sem mudar a função de aleatorização
- Identificar as regiões que sofreram sucessivas remoções de entradas na tabela
 - ◆ Proceder a fusão de blocos, também preservando a função de aleatorização

Tabela Estendível

◆ Implementação (i)

- A tabela de dados é dividida em blocos de tamanhos iguais
- Cada bloco é tratado como uma região que pode ser expandida:
 - ◆ Quando um bloco está cheio (ou próximo disso), ele é dividido em dois outros de mesmo tamanho, e as chaves são distribuídas entre os novos blocos criados
- O acesso à área de dados é feito através de um diretório

Tabela Estendível

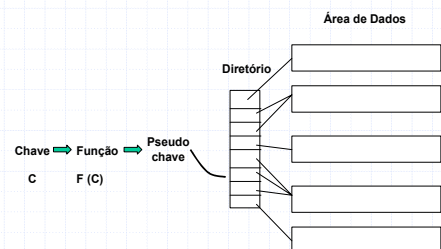


Tabela Estendível

◆ Implementação (ii)

- Cada entrada do diretório contém o endereço do bloco onde se encontra a entrada procurada, se ela estiver presente
- O acesso ao diretório se faz pela função *hash* aplicada sobre a chave de busca
- A função *hash* gera uma pseudo-chave

Tabela Estendível

- ◆ O diretório pode ser estendido (ou comprimido) de acordo com a evolução da tabela
- ◆ A função $f(c)$ gera pseudo chaves (por exemplo, de 32 bits) com distribuição uniforme. Portanto, há quantidades aproximadamente iguais de pseudo chaves com valores 0 e 1, em cada um de seus bits
- ◆ Cada bloco é identificado pela sequência dos primeiros "k" bits da pseudo chave
- ◆ O número "k" de bits utilizado determina o número de entradas do diretório: 2^k

Tabela Estendível

- ◆ Caso mais simples

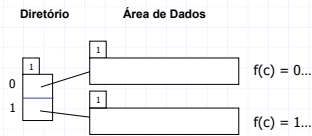


Tabela Estendível

- ◆ Após um desdobramento

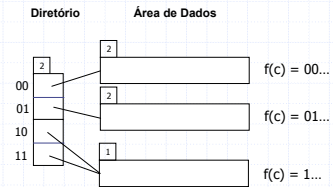


Tabela Estendível

- ◆ A estrutura interna de cada bloco pode ser organizada utilizando um esquema convencional de *hashing*
- ◆ Qualquer entrada da tabela pode ser acessada pela leitura de um único bloco
- ◆ Fica dispensado o uso de blocos de extensão e o armazenamento de entradas em bloco distinto daquele que lhe corresponde

Tabela Estendível

◆ Fusão de blocos

- Arbitra-se quando a taxa de ocupação de um bloco (p.e., < 30%) justifica uma fusão
- Condições necessárias para a fusão
 - Somente em blocos adjacentes
 - Os dois blocos são originários do desdobramento de um mesmo bloco
 - Os dois blocos possuem a mesma profundidade
- Uma fusão de blocos pode causar a redução do tamanho do diretório

Compressão de Chaves Alfanuméricas

◆ Objetivo

- Minimizar o problema da representação numérica excessivamente grande para ser armazenada em uma palavra, para fins de operação.

◆ Implementação

- Emprego do operador lógico "ou exclusivo" (xor): maneira prática e usual de se fazer compressão

Operador XOR

A	B	A xor B
0	0	0
0	1	1
1	0	1
1	1	0

Exemplo

- Exemplo: Comprimindo a chave BRASIL para 16 bits

Representação binária do código ASCII

(BR) 1100001011010010
(AS) 1100000111010011
(IL) 1100100111001100
xor 1100101011001101 = 51.917₍₁₀₎

- Considerando uma tabela de 521 entradas, teríamos:

$$(51.917 \bmod 521) + 1 = 338$$

Funções de Cálculo de Endereço

Método da divisão

$$F(x) = (x \bmod n)$$

- Resultando em endereços no intervalo [0, n-1]
- Tende a preservar a uniformidade no conjunto de chaves, mapeando valores de chave consecutivas em endereços consecutivos

Método da divisão

- O valor de "n" (divisor) deve ser escolhido com muito critério
- Na prática os melhores resultados são obtidos quando o divisor é um número primo
- A quantidade de colisões depende da taxa de ocupação da tabela
- Sugestão: acrescentar 20% ao total de chaves a serem instaladas e tomar o menor número primo maior ou igual ao resultado obtido

Funções dependentes da distribuição

- Até agora se considerou que as chaves estivessem uniformemente distribuídas em seu domínio
- Em tabelas dinâmicas, a questão da distribuição das chaves dificilmente pode ser levada em conta, pois isto exige um conhecimento prévio dos valores das chaves
- No caso de tabelas estáticas, o estudo da distribuição das chaves pode ser feito preliminarmente, permitindo a escolha de uma função de aleatorização mais adequada à distribuição

Funções dependentes da distribuição

Método de Análise de Frequências das Chaves

- Analisa a frequência com que os dígitos (ou caracteres) ocorrem em cada posição da chave.
- Exemplo: seja uma tabela com 500 chaves de 4 posições, que apresenta a seguinte tabulação de frequências

	Dígito									
	0	1	2	3	4	5	6	7	8	9
Frequências										
1ª	0	500	0	0	0	0	0	0	0	0
2ª	0	0	0	0	0	0	0	0	0	500
3ª	7	25	28	35	48	80	95	112	50	20
4ª	43	57	48	32	50	92	37	41	63	37

- Escolhendo apenas o 3º e o 4º dígitos das chaves, estaremos espalhando-as de maneira mais uniforme ao longo do espaço de endereçamento, o qual ficará restrito ao intervalo [0, 99]