Universidade Federal de São Carlos
Centro de Ciências Exatas e Tecnologia
Departamento da Computação
Prof. Jander Moreira

Hash Extensível

Carla Florentino de Souza	230995
Priscila Furuya	231231
Rafael Ito Takashi	230000
Sabrina Stachfledt	231010

Introdução

Tabela Hash é um método de pesquisa onde a busca é feita baseada no valor da chave, sendo esta a única identificação da posição. Como a chave é conhecida, a posição na tabela pode ser acessada diretamente, diferente de outros métodos onde é necessário outros tipos de testes, conforme exigido na busca binária ou durante uma pesquisa em uma árvore.

O método hashing consiste basicamente no armazenamento de cada entrada em um endereço calculado pela aplicação de uma função (função hashing) à chave da entrada. O processo de pesquisa sobre uma tabela organizada dessa maneira é similar ao processo de inserção de uma entrada, e consiste na aplicação da função hashing (que calcula o endereço) ao argumento de pesquisa, obtendo como resultado o endereço da entrada procurado.

O esquema hashing descrito pelo professor em aula é chamado hashing estático, porque um número fixo M de entradas na tabela é alocado. Isso causa um sério inconveniente, pois pode haver mais registros do que as M's posições na tabela.

Técnicas têm sido desenvolvidas para levar em consideração o tamanho da tabela, ou do arquivo. Podemos distinguir duas classes de tais técnicas: diretório e sem diretório.

O esquema de diretório consiste no acesso ao registro através de um índice de chaves na estrutura (diretório). Alguns destes tipos de esquemas são: hash extensível, hash expansível e hash dinâmico. Um tipo de hash que não usa diretório é linear .

O enfoque do nosso trabalho está sobre o hash extensível.

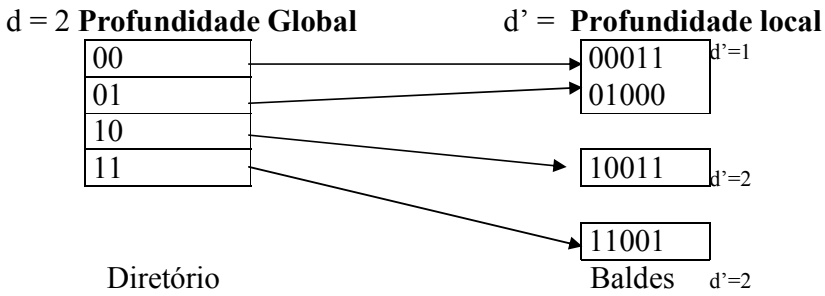
Hash Extensível

O hash extensível usa um diretório dinâmico de registros que armazena uma tabela, onde cada registro contém um ponteiro para balde (tabela que armazena os registros) e cada balde tem um número fixo de itens.

Aplicando-se a função nas chaves obteremos um número binário. Desse número devemos escolher uma quantidade de bits que fará a diferenciação entre os índices a serem armazenados no diretório. A esse número de bits escolhidos damos o nome de **profundidade global** (**d**) que especificará o número de linhas da tabela (diretório), que será 2^d.

Cada índice aponta para um determinado balde, no entanto, um mesmo balde pode ser apontado por vários índices. Cada balde, também tem sua **profundidade local (d')**, que é a especificação de quantos bits o balde é baseado.

Como exemplo, se tivermos uma função que gere padrões de 5 bits e definirmos a profundidade como 2, então será gerada o seguinte diretório:



Cada balde pode guardar um número especifico de entradas (chaves hashing). Numa inserção se ocorrer o transbordamento (número de registros do balde já estiver no máximo), teremos duas possibilidades de divisão, que pode acontecer entre os baldes ou no diretório. A primeira ocorre quando a profundidade local do balde, onde está sendo inserida a chave, é menor do que a profundidade global. Neste caso, criam-se mais baldes com uma profundidade um bit maior e dividem-se as chaves entre eles.

A segunda possibilidade ocorre quando a profundidade do balde é igual a global, ocorrendo então a duplicação do diretório, aumentando-se a profundidade global em um bit. O balde "transbordado" é então dividido e reorganizado assim como na primeira possibilidade.

Podemos ter não somente a duplicação do diretório como também sua divisão. Se formos excluir uma chave e a profundidade global for maior que a profundidade local para todos os baldes, então ocorrerá a divisão do diretório (diminuindo o número de baldes).

Vantagens

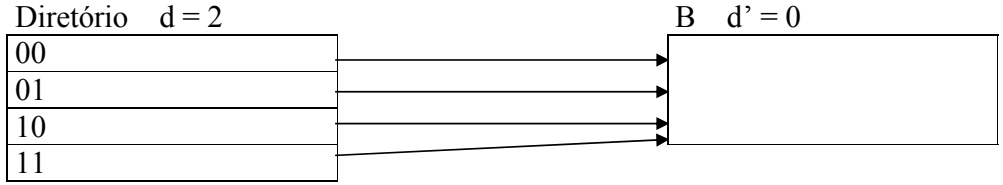
- ✓ Evitar reorganização do arquivo, caso o diretório transborde. Somente o diretório é afetado.
- ✓ O desempenho do arquivo não se degrada `a medida que o arquivo cresce.
- ✓ Torna mais fácil a expansão ou redução dinâmica do arquivo, armazenado uma estrutura de acesso além do arquivo.

Desvantagens

- ✓ Estruturas como hash extensível são vantajosas para sistemas de banco de dados, porém para programas menores que acrescentam e removem dados com freqüentemente essas estruturas tornam-se menos eficientes, já que a duplicação e remoção são processos demorados.

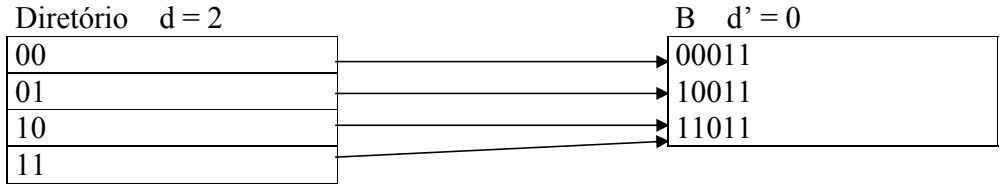
Exemplificando

No início temos o diretório com profundidade d = 2. Gerando uma tabela de tamanho $2^{d} = 2^{2} = 4$, onde todos os índices do diretório apontam para o mesmo balde de profundidade d'=0.



Vamos inserir o número **00011**, localizamos pelo diretório para qual balde a chave hash *00* aponta (2 primeiros dígitos porque a profundidade do balde é 2).

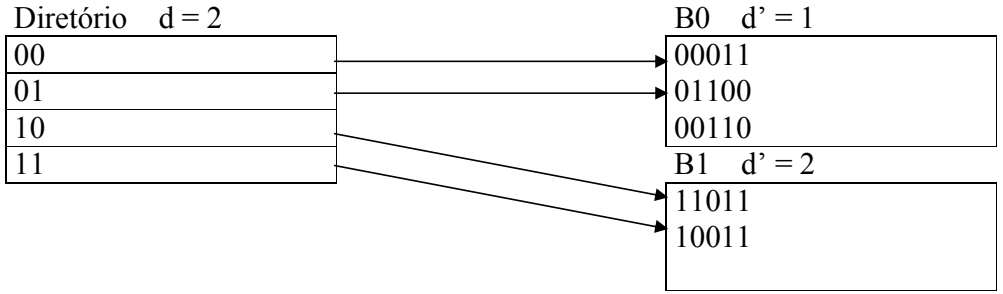
Pode-se inserir registros até que o balde esteja cheio como no caso de inserirmos **10011** e **11011** no mesmo balde.



Agora, queremos inserir **01100**, entretanto o balde está cheio, criaremos então dois outros balde e cada um deles terão profundidade d'=1 e serão chamados de B0 e B1. Os dois primeiros índices do diretório apontado para B0 e os dois últimos para B1 (perceba que o primeiro dígito do índice é igual ao nome do Balde).

Cada registro do balde B é colocado no seu balde equivalente, inclusive o registro **01100**.

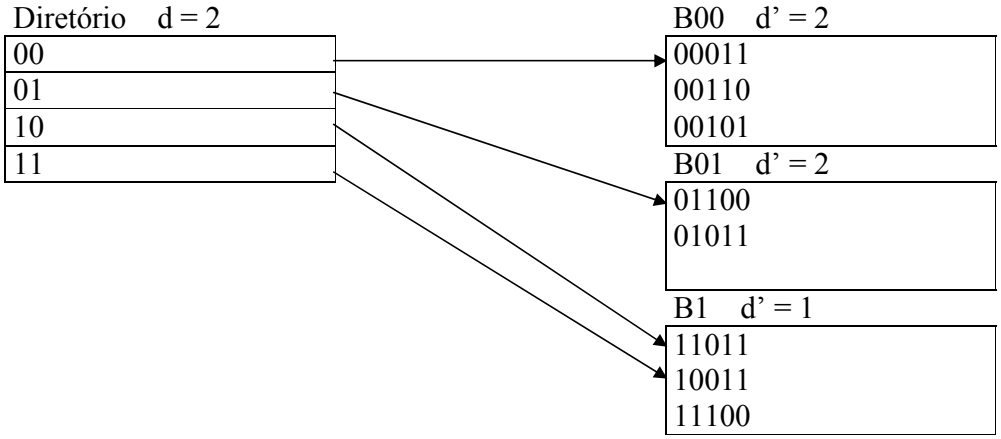
Para inserir o registro **00110**, vamos no diretório, escolhemos o balde para qual o índice *00* aponta e inserimos o registro.



Quando fomos inserir o registro **00100** o balde B0 estará cheio, tendo que ocorrer então mais uma divisão. Criaremos o balde B00 e B01 com profundidade d' = 2, onde o índice do diretório *00* aponta para o B00 e o índice do diretório aponta para B01. Cada registro é colocado no seu respectivo balde.

Agora vamos inserir um registro **01011**, verificamos para onde o índice *01* do diretório aponta e inserimos o registro no balde B01.

Por fim, inseriremos **11100** no balde B1 que é para onde o índice *11* aponta.



Considerações finais

Percebemos que a principal diferença entre hash e hash extensível está no fato do tamanho da tabela desta ser variável.

No entanto, ambos não estão livres das colisões causadas pela escolha de uma função que não seja perfeita.

Bibliografia

Drozdek, Adam , "Estrutura de Dados e Algoritmos em C++", 1ª edição, editora Thomson, 2002.

Elmasri e Navathe, "Sistema de Banco de Dados – Aplicações e Fundamentos", 3ª edição, editora LTC.

Garcia – Molina, "Database System Implementation", editora Prentice-Hall.