

## Hashing

### Lista de exercícios 11

- 1) Use a função Hash(key, MaxAd) descrita no livro para responder às seguintes questões:
  - a) Qual o valor de Hash("Browns", 101)?
  - b) Encontre duas chaves diferentes de mais de quatro caracteres que sejam sinônimas (segundo essa função *hash*).
  - c) Assuma-se, no texto, que a função Hash() não precisa gerar um inteiro maior que 19937. Isto representa um problema se tivermos um arquivo com endereços maiores que 19937. Qual é esse problema? Sugira possíveis formas de contorná-lo.
  
- 2) Há um resultado matemático surpreendente chamado "paradoxo do aniversário" que afirma que, se há mais de 23 pessoas em uma sala, há mais de 50% de chance de que duas pessoas façam aniversário no mesmo dia. Explique porque este paradoxo é um exemplo do maior problema do hash.
  
- 3) Em nossos cálculos de comprimento médio de busca, consideramos apenas as buscas bem sucedidas. Se usássemos o hash em um arquivo em que algumas vezes o item não fosse encontrado, seria interessante manter estatísticas sobre o comprimento médio das buscas mal sucedidas. Se uma alta porcentagem das buscas for mal sucedida, como você imagina que isto afetará o desempenho geral se o overflow for tratado:
  - a. Por overflow progressivo;
  - b. Por overflow progressivo encadeado;
  - c. Usando uma área de overflow em separado.
  
- 4) Crie um arquivo hash com registros para 30 cidades do estado de São Paulo cujos nomes comecem com as letras a, b, c, s. A chave de cada registro será o nome da cidade e não são necessários outros campos para este exercício. Comece colocando os nomes destas cidades em ordem alfabética.
  - a) Examine a lista ordenada. Que padrões você nota que podem afetar sua escolha de uma função de hash?
  - b) Implemente uma função hash() que utiliza alguma combinação dos códigos ASCII das letras do nome, mas de forma que você possa alterar o número de caracteres que são utilizados na combinação. Execute o hash() várias vezes, cada vez utilizando um número diferente de caracteres e produzindo as seguintes estatísticas para cada execução:
    - O número de colisões;
    - O número de endereços com 0,1,2,3,...10, ou mais de 10 cidades associadas.Discuta os resultados de seu experimento em termos de efeitos da escolha de diferentes quantidades de caracteres e como eles se relacionam com o resultado que você poderia esperar de uma distribuição aleatória.  
(Implemente e teste um ou mais dos métodos de hash descritos no texto, ou use um método inventado por você).

5) Escreva uma função em C chamada `search(Tabela, Chave)` que busque uma chave em uma tabela hash. A função aceita uma chave inteira e uma tabela declarada por

```
struct record {  
    KeyType k;  
    RecType r;  
    int flag;  
} array [TableSize];
```

`tabela[i].k` e `tabela[i].r` são o *i*-ésimos chave e registro respectivamente. `tabela[i].flag` é igual a FALSE se a *i*-ésima posição da tabela estiver vazia e TRUE se estiver ocupada. A rotina retorna um inteiro entre 0 e `TableSize-1` se um registro com a chave `Chave` estiver presente na tabela. Se este registro não existir a função retorna -1. Assuma a existência de uma rotina de hash, `h(Chave)`, e uma rotina de *rehash* `rh(Índice)` que também retorna valores entre 0 e `TableSize-1`.

6) Escreva uma função em C `sinsert(table, key, rec)` para buscar e inserir chaves numa tabela hash como a do exercício anterior.

7) Desenvolva um mecanismo para detectar quando todas as posições possíveis para re-espalhamento foram acessadas. Incorpore este método nas rotinas `search` e `sinsert` dos exercícios anteriores.

8) Usando algum conjunto de chaves (por exemplo, nomes de cidades do estado de SP), faça o seguinte:

- a) Escreva e teste um programa que carregue as chaves em 3 tabelas hash distintas, usando cestos de tamanho 1, 2 e 5, respectivamente, e uma densidade de ocupação igual a 80%. Use overflow progressivo para tratar colisões. Inclua no seu programa código para gerar estatísticas, como o tamanho médio de busca, o tamanho máximo de busca, e a porcentagem de registros de overflow.
- b) Escreva um programa para gerenciar inserções e remoções na tabela, para o caso em que o tamanho do cesto é 5.

9) Considere a seguinte sequência de chaves: MALUF, QUÉRCIA, SERRA, CARDOSO, LULA, ALVES, GOMES, JEREISSATI, FREIRE, MAGALHÃES, FERREIRA, ANDRADE, CAMARGO. Construa a *trie* adequada para armazenar esse conjunto de chaves.

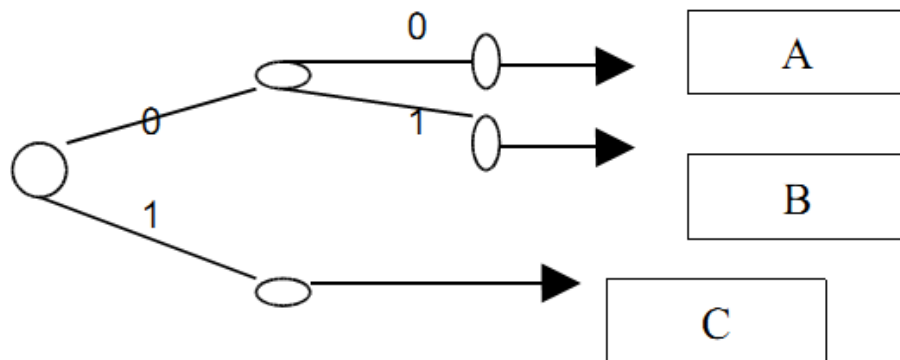
10) Explique como funciona o *hashing* extensível.

11) Qual a diferença entre o espalhamento extensível e o espalhamento convencional? Porque o segundo não é adequado para representar índices armazenados em disco?

13 ) Qual a vantagem de aplicar uma função de espalhamento sobre a chave para definir o seu endereço (cesto), ao invés de amostrar diretamente o valor da chave, como feito nas tries?

14 ) Qual a vantagem de usar a representação em diretório no *hashing* extensível, ao invés de usar a representação por árvore da *trie*?

15 ) Considere a seguinte trie de ordem (raio) 2, com ponteiros para *buckets* com capacidade para abrigar 100 chaves (ou registros):



- Desenhe a trie estendida e o diretório de endereços *hash* correspondente.
- Considerando que os *buckets* A, B e C contêm, respectivamente, 100, 50 e 03 registros, dê a configuração do diretório, e a condição de cada *bucket* após a inserção de uma nova chave cujo valor da função *hash* é 00.
- Ainda na configuração inicial, considere agora que todas as chaves de B são eliminadas. O que acontece com o diretório?