USP-ICMC-BInfo
Introdução à Ciência de Computação II - SCC-0501
Prof. João Luís - versão 1 - 09/11/2011

Lista de Exercícios 5

1. Escrever o código de uma função C que faz uma busca sequencial por uma chave em um vetor de n elementos, com sentinela.

2. No exercício anterior, qual a vantagem do uso do sentinela em relação à busca sem sentinela?

3. Escrever o código de uma função C que faz busca sequencial indexada por um elemento em uma tabela com índice primário, e insere-o na tabela caso não seja encontrado (o índice precisa ser atualizado).

4. Escrever o código de uma função C que faz busca sequencial indexada por um elemento em uma tabela com índice primário, e remove-o da tabela caso seja encontrado (o índice precisa ser atualizado).

5. Escrever o código da versão recursiva da busca binária em vetor.

6. Defina uma função *hash* qualquer (suponha que você está lidando com números inteiros positivos lidos do usuário, até que -1 seja dado como entrada). Utilize *hashing* estático aberto (com listas encadeadas). Implemente o código da função de inserção. A cada inserção, imprima a posição da tabela em que o elemento foi inserido. Ao fim da execução, imprima todos elementos da tabela *hash* e libere toda a memória utilizada.

7. Considere as técnicas de pesquisa sequencial, pesquisa binária e a pesquisa baseada em *hashing*.
   (a) Descreva as vantagens e desvantagens de cada uma das técnicas acima, colocando em que situações você usaria cada uma delas.
   (b) Dê a ordem do pior caso e do caso esperado de tempo de execução para cada método.
   (c) Qual é a eficiência de utilização de memória (relação entre o espaço necessário para dados e o espaço total necessário) para cada método?

8. Quais as características de uma boa função *hash*?

9. Um dos métodos utilizado para se organizar dados é pelo uso de tabelas *hash*.
   (a) Em que situações a tabela *hash* deve ser utilizada?
   (b) Descreva dois mecanismos diferentes para resolver o problema de colisões de várias chaves em uma mesma posição da tabela. Quais são as vantagens e desvantagens de cada mecanismo?

10. Em uma tabela *hash* com cem entradas, as colisões são resolvidas usando listas encadeadas. Para reduzir o tempo de pesquisa, decidiu-se que cada lista seria organizada como uma árvore binária de busca. A função utilizada é $h(k) = k \ mod \ 100$. Infelizmente, as chaves inseridas seguem o padrão $k_i = 50i$, onde k_i corresponde a i-ésima chave inserida:

Veja a próxima página...

(a) Mostre a situação da tabela após a inserção de k_i, com i = 1, 2,..., 13 (faça um desenho).

(b) Depois que mil chaves são inseridas de acordo com o padrão acima, inicia-se a inserção de chaves escolhidas de forma aleatória (isto é, não seguem o padrão das chaves já inseridas). Assim responda:

(a) Qual é a ordem do pior caso (isto é, maior número de comparações) para inserir uma chave?

(b) Qual é o número esperado de comparações para inserir uma chave? (Assuma que cada uma das cem entradas da tabela é igualmente provável de ser endereçada pela função h.)

11. Substitua $XXXXXXXXXXXX$ pelas 12 primeiras letras do seu nome, desprezando branco e letras repetidas, nas duas partes desta questão. Para quem não tiver doze letras diferentes no nome, completar com as letras $PQRSTUWXYZ$, nesta ordem, sem repetir letras já usadas, até completar 12 letras. Por exemplo, eu deveria escolher $JOALUISPQRTW$. A segunda letra O de $JOAO$ não entra porque ela já apareceu antes, e assim por diante.

(a) Desenhe o conteúdo da tabela *hash* resultante da inserção de registros com as chaves $XXXXXXXXXXXX$, nesta ordem, em uma tabela inicialmente vazia de tamanho 7 (sete), usando listas encadeadas (*hashing* estático aberto). Use a função *hash* $h(k) = k \ mod \ 7$ para a k-ésima letra do alfabeto.

(b) Desenhe o conteúdo da tabela *hash* resultante da inserção de registros com as chaves $XXXXXXXXXXXX$, nesta ordem, em uma tabela inicialmente vazia de tamanho 13 (treze), usando *overflow* progressivo e sondagem linear (*hashing* estático fechado) para resolver as colisões. Use a função *hash* $h(k) = k \ mod \ 13$ para a k-ésima letra do alfabeto.

12. *Hashing* Estático Fechado:

(a) *Overflow* progressivo com sondagem linear. Desenhe o conteúdo da tabela *hash* resultante da inserção de registros com as chaves $QUESTAOFCIL$, nesta ordem, em uma tabela inicialmente vazia de tamanho 13 (treze) usando *overflow* progressivo com sondagem linear para a escolha de localizações alternativas. Use a função *hash* $h(k) = k \ mod \ 13$ para a k-ésima letra do alfabeto.

(b) *Hashing* Duplo. Desenhe o conteúdo da tabela *hash* resultante da inserção de registros com as chaves $QUESTAOFCIL$, nesta ordem, em uma tabela inicialmente vazia de tamanho 13 (treze) usando *hashing* duplo. Use a função *hash* $h_1 = k \ mod \ 13$ para calcular o endereço primário e $j = 1 + (k \ mod \ 11)$ para resolver colisões, ou seja, para a escolha de localizações alternativas. Logo $h_i(k) = (h_{i-1}(k) + j) \ mod \ 13$, para $2 \leq i \leq M$, onde M é o tamanho da tabela.

13. Escreva uma função em C *search (tabela, key)* que procure numa tabela *hash*, um registro com a chave *key*. A função recebe um inteiro *key* e a tabela declarada abaixo:

```
struct record {
KEYTYPE k;
RECTYPE r;
int flag;
} tabela [TABLESIZE];
```

    $tabela[i].k$ e $tabela[i].r$ são a i-ésima chave e o i-ésimo registro respectivamente, $tabela[i].flag$ será *falso* se a i-ésima posição na tabela estiver vazia e *verdadeiro* se estiver ocupada. A função retorna um inteiro 0 até $(tablesize - 1)$ se um registro com a chave key está presente na tabela. Se o registro não existir a função retorna -1 (menos um). Assuma que já existe uma função *hash*, $h(key)$, e uma função *rehash* $rh(index)$ e que as duas produzam inteiros de 0 até $(tablesize - 1)$.

14. Escreva uma função $sinsert(tabela, key, rec)$ para procurar e inserir em uma tabela *hash* idêntica ao exercício anterior.

15. Desenvolva um mecanismo para detectar quando todas as posições possíveis para o *rehash* de uma chave já tenham sido procuradas. Incorpore essa função às funções já escritas em C *search* e *sinsert*, dos exercícios anteriores.

16. Considere um método de espalhamento duplo (*hashing* duplo) usando a função de espalhamento básica, $h_1(key)$, e a função de re-espalhamento, $rh(i) = tablesize\%(i + h_2(key), tablesize)$. Suponha que $h_2(key)$ seja primo em relação à $tablesize$, para toda chave key. Desenvolva um algoritmo de busca e um algoritmo para inserir um registro cuja chave sabe-se não existir na tabela, de modo que as chaves em sucessivos re-espalhamentos de uma única chave estejam em ordem ascendente. O algoritmo de inserção pode reordenar os registros inseridos anteriormente na tabela. Você pode estender esses algoritmos para um algoritmo de busca e inserção?

17. Demonstre a inserção das chaves 5, 28, 19, 20, 33, 12, 17, 10 em uma tabela *hash*, em que as colisões são resolvidas por encadeamento. A tabela tem tamanho 9, e a função *hash* $h(k) = k \ mod \ 9$.

18. Sugira como o armazenamento para elementos pode ser alocado e desalocado dentro da própria tabela *hash*, através da vinculação de todas as posições não utilizadas em uma lista livre. Suponha que uma posição possa armazenar um *flag* e um elemento mais um ponteiro ou dois ponteiros. Todas as operações de dicionário e de lista livre devem ser executadas no tempo esperado $\mathcal{O}(1)$. A lista livre precisa ser duplamente encadeada, ou seria suficiente uma lista livre simplesmente encadeada?

19. Suponha que se deseja pesquisar uma lista encadeada de comprimento n, onde cada elemento contém uma chave k juntamente com um valor *hash* $h(k)$. Cada chave é uma cadeia de caracteres longa. Como seria possível tirar proveito dos valores *hash* ao procurar na lista por um elemento com uma chave específica?

20. Considere uma tabela *hash* de tamanho $m = 1000$ e a função *hash* correspondente $h(k) = \lfloor m(k\ A\ mod\ 1) \rfloor$ para $A = (\sqrt{5}-1)/2$. Calcule as localizações para as quais as chaves 61, 62, 63, 64 e 65 estão mapeadas.

21. Considere a inserção das chaves 10, 22, 21, 4, 15, 28, 17, 88, 59 em uma tabela *hash* de comprimento $m = 11$ usando o endereçamento aberto com a função *hash* primário $h'(k) = k\ mod\ m$. Ilustre o resultado da inserção dessas chaves com o uso da sondagem linear, empregando a sondagem quadrática com $c_1 = 1$ e $c_2 = 3$, e com a utilização do *hash* duplo com $h_2(k) = 1 + (k\ mod\ (m-1))$.

22. Considere uma tabela *hash* de endereço aberto com *hashing* uniforme. Forneça limites superiores sobre o número esperado de sondagens em uma pesquisa mal-sucedida e sobre o número esperado de sondagens em uma pesquisa bem-sucedida quando o fator de carga é $3/4$ e quando ele é $7/8$.

References

[1] Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C., *Algoritmos - Teoria e Prática*. Ed. Campus, Rio de Janeiro, Segunda Edição, 2002.

[2] Oliveira, Maria Cristina Ferreira de, *Quinta Lista de Exercícios - Hashing*, SCE0181 - Introdução à Ciência da Computação II. ICMC-USP, 2008.

[3] Oliveira, Maria Cristina Ferreira de, *Sexta Lista de Exercícios - Busca*, SCE0181 - Introdução à Ciência da Computação II. ICMC-USP, 2008.

[4] Sedgewick, R., *Algorithms*, second edn, Addison Wesley, 1988.

[5] Tenenbaum, A. M., Langsam, Y., Augestein, M. J., *Estruturas de Dados Usando C*. Makron Books, 1995.

[6] Ziviani, N., *Projeto de Algoritmos - com implementações em Pascal e C*, 2a. Edição. Thomson, 2004.