Lista de Exercícios 07
Introdução à Ciência de Computação II (SCC0201/501)

Prof. Moacir P. Ponti Jr.

24 de novembro de 2010

1 Hashing

1. Suponha uma tabela *hash* de tamanho 10 com endereçamento aberto para armazenar chaves no intervalo [1, 999]. Insira as seguintes chaves nessa tabela: `371, 121, 173, 203, 11, 24`, nessa ordem, considerando diferentes métodos de resolução de colisões:
   a) Sondagem linear, funcao hash: $h(k) = k\%M + i$
   b) Sondagem quadrática, funcao hash: $h(k) = k\%M + i^2$
   c) Sondagem quadrática, funcao hash: $h(k) = k\%M + 2i + i^2$
   d) Hash duplo, funcao hash: $h_1(k) = k\%M$, função hash 2: $h_1(k) = 7 - (k\%7)$

2. Utilizando os resultados do exercício anterior, responda:
   a) qual o fator de carga final da tabela?
   b) quais as estratégias que aparentemente realizaram melhor espalhamento das chaves?
   c) como ficaria o mesmo cenário utilizando uma tabela *hash* de tamanho 7 com encadeamento?

3. O uso de tabelas hash com encadeamento facilita a operação de remoção, pois apenas é preciso remover o elemento da lista ligada. No caso de endereçamento aberto a remoção deve ser feita de maneira diferente.
   a) qual operação é realizada sobre a chave removida?
   b) em caso de existirem muitos elementos marcados como deletados na tabela, perde-se a eficiência nas buscas; como é possível tratar esse problema?

4. Quais as vantagens e desvantagens (com relação à criação da estrutura, inserção e remoção de elementos e busca) de cada uma das estruturas abaixo para o caso em que se deseja armazenar um dicionário, com chaves compostas por *strings* com tamanho de até 80 caracteres e suas definições compostas por *strings* de tamanho

até 2000 caracteres. Sabe-se que a quantidade de palavras pode variar entre 10.000 e 50.000.

a) Arranjo e busca binária

b) Lista encadeada e busca sequencial

c) Tabela *hash* com endereçamento aberto e *hashing* duplo

d) Tabela *hash* com encadeamento

2 Projeto de Algoritmos: parte 1 (referente à aula das semana 22-26/11)

5. Analise o algoritmo abaixo.

```
1. int fatorial(int n) {
2.     int i = 1, F = 1;
3.     while (i <= n) {
4.         F = F * i;
5.         i++;
6.     }
7.     return F;
8. }
```

Prove a corretude do algoritmo por invariantes de laço:

a) demonstrando que o algoritmo termina (ou seja, que o laço termina e que o programa eventualmente alcança a linha 7)

b) provando que após passar pelo laço k vezes, $F = k!$ e $i = k + 1$.

c) demonstrando que, após o laço terminar, $F = n!$.

6. Analise a função recursiva abaixo que retorna a soma dos números entre 1 e n.

```
1. int soman(int n) {
2.     if (n == 1) {
3.         return 1;
4.     } else {
5.         return n + soman(n-1);
6.
7. }
```

a) Com base na seguinte equação geral de recorrência tente encontrar qual seria a equação de recorrência para a função `soman`.

$$T(n) = aT\left(\frac{n}{b}\right) + f(n),$$

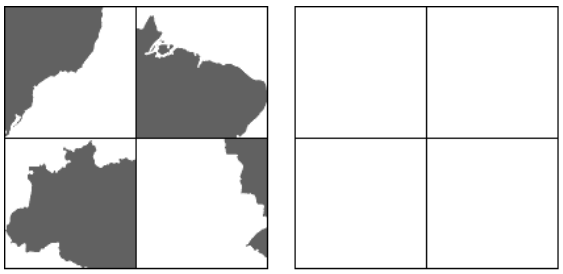
onde:

- a indica o número de sub-problemas gerados
- b o tamanho de cada um dos problemas
- $f(n)$ o custo de resolver cada sub-problema

b) Com base na equação de recorrência encontrada, prove por indução que $T(n)$ é $O(f(n))$ para $f(n) = n$, ou seja, encontre o caso base $T(1)$, assuma que $T(n)$ é $O(n)$ e prove para $T(n+1)$ e $O(n+1)$.

7. Considere os algoritmos baseados em tentativa e erro. Quando eles são utilizados? Qual é o principal problema que enfrentam os algoritmos que realizam uma busca exaustiva pela solução?

8. Considere um algoritmo que tem o objetivo de montar um quebra-cabeças. O quebra-cabeças possui 4 peças e está embaralhado numa mesa com (2x2) espaços, conforme demonstrado abaixo. O algoritmo irá tentar montá-lo de forma correta posicionando peça por peça em um outro espaço de 2x2 peças, que inicialmente está vazio.



O algoritmo se comporta:

- pegando um peça na mesa embaralhada e inserindo na mesa de montagem, verifica se a peça se encaixa com as peças vizinhas eventualmente já montadas.
  - se não encaixou remove a peça atual e tenta uma nova peça — se nenhuma peça disponível servir, remove a peça anterior e repete o procedimento para uma nova peça.
  - se encaixou repete o procedimento para uma nova peça.

Responda:

a) o paradigma do algoritmo é de tentativa e erro?

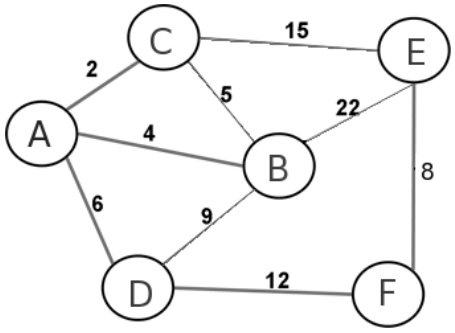
b) o algoritmo tenta obter a solução de forma exaustiva (força bruta)?

c) o algoritmo eventualmente termina? Se sim, é possível estimar qual seria o pior caso?

9. Considere os algoritmos baseados em tentativa e erro. Quando eles são utilizados? Qual é o principal problema que enfrentam os algoritmos que realizam uma busca exaustiva pela solução?

3 Projeto de Algoritmos: parte 2 (referente à aula da semana 29/11-03/12)

10. Suponha que, no gráfico abaixo, as letras representam cidades e os caminhos entre as cidades estão marcados com a distância entre as cidades conectadas. Imagine um problema de encontrar o menor caminho entre duas cidades.



O paradigma da programação dinâmica resolve sub-problemas menores e os armazena para posterior reuso, enquanto o paradigma de algoritmos gulosos escolhe sempre a solução que parece mais promissora num dado instante.

Como se comportaria e qual seria o provavel resultado se os seguintes algoritmos tivessem que obter o menor caminho entre a cidade B e a cidade F?

a) Programação dinâmica (caminhe pelas estradas armazenando possíveis soluções e descarte sub-soluções quando encontrar melhores opções).

b) Algoritmo guloso utilizando a heurística do vizinho mais próximo (ou seja, caminhe sempre pela próxima estrada cuja distância seja mínima, em direção à cidade destino).

11. Um problema muito famoso é o do escalonamento de grade horária, que consiste em, dadas as seguintes variáveis:

- janelas de tempo
- salas
- disciplinas (das quais participam alunos e professores)

montar um horário viável, ou seja, no qual as salas sejam ocupadas pelas disciplinas em diferentes janelas de tempo de forma que tanto alunos quanto professores participar de todas as disciplinas que estejam inscritos. Discuta esse problema do

ponto de vista dos seguintes paradigmas de projeto de algoritmos, descrevendo as vantagens e desvantagens de cada abordagem.

- Tentativa e erro (backtracking)
- Programação dinâmica
- Heurísticas
- Algoritmos aproximados