Algoritmos e Complexidade – LEI/LCC

13 de Setembro de 2011 – Duração: 2 horas

Exame da Época Especial

Parte I

Esta parte do exame representa 12 valores da cotação total. Cada alínea está cotada em 2 valores. **A não obtenção de uma classificação mínima de 8 valores nesta parte implica a reprovação no teste.**

1. Considere o seguinte programa (anotado em comentário)

   Apresente as condições de verificação necessárias à prova da correcção parcial deste programa.

```
    // n = n0 > 0
    x=1; y=0;
    // n = n0 > 0 /\ x=1 /\ y = 0
    while (n>1) {
    //  x = fib (n0 - n + 1) /\ y = fib (n0 - n) /\ n >= 1
        x = x+y; y = x-y; n = n-1;
    }
// x = fib (n0)
```

2. Considere a sequinte definição de uma função que calcula a altura de uma árvore.

```
typedef struct nodo {
    int v;
    struct nodo *esq, *dir;
   } Nodo, *BTree;

int altura (BTree a) {
   if (a == NULL) return 0;
   else return (1 + max (altura (a->esq), altura (a->dir)));
}
```

   Para cada um dos casos (extremos) de a árvore estar perfeitamente equilibrada ou perfeitamente desiquilibrada, apresente relações de recorrência que traduzam o tempo de execução desta função em função do tamanho da árvore de entrada (i.e., do número de nodos da árvore). Apresente os resultados da análise global em notação assimptótica.

3. Considere uma tabela de Hash implementada sobre um array de tamanho 7 para armazenar números inteiros. A função de hash utilizada é $h(x) = x\%7$ (em que % representa o resto da divisão inteira). O mecanismo de resolução de colisões utilizado é *open addressing* com *linear probing*.

   - Apresente a evolução desta estrutura de dados quando são inseridos os valores 1, 15, 14, 3, 9, 5 e 27, por esta ordem.
   - Descreva o processo de remoção de um elemento ensta estrutura de dados, exemplificando com a remoção do valor 1 depois das inserções acima.

4. Considere a seguinte representação de um grafo por listas de adjacência:

```
A -> B, F
B -> A, C, G
C -> B, G, D
D -> C, E
E -> D, G, F
F -> A, E, G
G -> B, C, E, F
```

Apresente uma execução de uma travessia depth-first neste grafo, começando pelo vértice **G** (não se esqueça de, para cada passo, descrever o conteúdo da stack auxiliar).

5. Considere as seguintes definições de um tipo para representar grafos em listas de adjacência:

```
#define MaxV ...
#define MaxE ...

typedef struct edge {
     int dest;
     int cost;
     struct edge *next;
} Edge, *Graph [MaxV];
```

(a) Defina uma função que calcule o número de antecessores de um dado vértice.

(b) Defina ainda uma função que determina qual o vértice do grafo que tem mais antecessores. Note que esta função deve executar em $\mathcal{O}(V+E)$ em que V e E são respectivamente o número de vértices e arestas do grafo. Apresente uma análise do tempo de execução para justificar a sua resposta.

Parte II

1. Aumente as anotações do programa apresentado na questão 1 da primeira parte de forma a poder provar a sua correcção total. Apresente ainda as novas condições de verificação que resultam dessa nova anotação do programa.

2. A função seguinte calcula o tamanho da maior sequência de bits 1 na representação de um número inteiro. Faça a análise assimptótica do seu tempo de execução (em função do número de bits do argumento). Explicite o comportamento no melhor e no pior caso, e utilize notação assimptótica para expressar as suas conclusões sobre o comportamento do algoritmo.

```
int longest(int n) {
    int c,l = 0;
    while(n!=0) {
       c = 0;
       while (n % 2 == 1) { c = c + 1; n = n / 2; }
       if (c>l) { l = c; };
       n = n/2;
    }
    return l;}
```

3. Em grafos não pesados a composição de grafos (com um mesmo conjunto de vértices) define-se como: existe uma aresta `i->j` em `apos(g,f)` sse para algum vértice `k` existem as arestas `i->k` em `f` e `k->j` em `g`.

   (a) Defina uma função que implemente a composição de grafos não pesados quando representados em matrizes de adjacência de inteiros (`typedef int GMat [V][V];`).

   (b) Uma possível generalização deste conceito para grafos pesados consiste em dizer que o peso da aresta `i->j` em `apos(g,f)` é o mínimo da soma das arestas `i->k` em `f` e `k->j` em `g` para todos os vértices `k`. Defina uma função `void apos (GMat g, GMat f, GMat res)` que implemente a composição de grafos pesados quando representados em matrizes de adjacência de inteiros (considere que um peso negativo corresponde à ausência de aresta e que o tipo `GMat` é definido por `typedef int GMat [V][V]`).

$$\frac{\{\, I \wedge c \,\}\; S\; \{\, I \,\} \quad I \wedge c \Rightarrow V \geq 0 \quad \{\, I \wedge c \wedge V = v_0 \,\}\; S\; \{\, V < v_0 \,\}}{\{\, I \,\}\; \texttt{while}\; c\; S\; \{\, I \wedge \neg c \,\}} \quad \text{(while)}$$