Algoritmos e Complexidade
2º Ano – LEI/LCC

11 de Janeiro de 2011 – Duração: 2 horas

Teste

Parte I

Esta parte do teste 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 `//`)

```
// (exists (i in [0..n-1]) v[i] == x)
k = 0;
// (exists (i in [k..n-1]) v[i] == x)
while (v[k] != x)
    // (exists (i in [k..n-1]) v[i] == x)
    k=k+1
// v[k] == x
```

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

   (b) Indique um variante do ciclo em causa, e justifique informalmente porque se trata efectivamente de um variante.

2. Suponha que está a utilizar uma tabela de Hash com resolução de colisões por Chaining, e em que o factor de carga atingiu os 50%.

   - Apresente uma definição de factor de carga e indique qual o valor máximo para este parâmetro no caso do método de resolução de colisões apresentado.
   - Qual o número de comparações no melhor caso de uma pesquisa falhada na tabela indicada? E no pior caso? Justifique a sua resposta e apresente o resultado da sua análise utilizando notação assimptótica.

3. Considere o seguinte algoritmo recursivo para cálculo do máximo numa árvore binária com inteiros positivos (não necessariamente ordenada):

```
typedef struct tree {
     int val;
     struct tree *left;
     struct tree *right;
} *Tree;

int maximo(Tree t) {
     int dir, esq, res;
     if (!t) return -1;
     esq = maximo(t->left);
     dir = maximo(t->right);
     res = t->val;
     if  (res < esq) res = esq;
```

```
    if (res < dir) res = dir;
    return res;
}
```

(a) Escreva uma equação de recorrência para o tempo de execução deste algoritmo, desenhe a respectiva árvore, e represente o seu tempo de execução utilizando notação assimptótica.

(b) Imagine que se trata de uma árvore binária de pesquisa. Apresente um algoritmo que resolva o mesmo problema de forma mais eficiente, e repita a análise que efectuou na alínea anterior.

4. 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];
```

Defina em C uma função que calcula o inverso de um grafo (um grafo que tem uma aresta `i->j` se e só se existe uma aresta `j->i` no grafo original).

Parte II

1. Faça a análise da complexidade da função que apresentou na alínea **??** da Parte I e justifique porque não é possível obter uma definição com uma complexidade assimptótica inferior a $\Theta(V + E)$ (em que V e E são respectivamente o número de vértices e de arestas do grafo).

2. Considere a seguinte definição de uma função que testa se uma dada função `teste` é válida para algum subconjunto de `v`:

```
int f (int v[], n) {
int i, r;

r = 0;
if (n==0) return 0;
if (teste (v,n)) r = n;
else for (i=0;((r == 0 ) && (i<n-1));i++) {
        swap (v,i,n-1);
        r = f(v,n-1));
        swap (v,i,n-1);
    }
return r;
}
```

Sabendo que a função `teste` tem uma complexidade linear no tamanho do vector de entrada, apresente uma relação de recorrência que traduza a complexidade da função `f` em função do tamanho `n` do vector recebido como argumento.

Apresente ainda uma solução dessa recorrência, justificando-a informalmente.

3. Um conhecido problema NP-completo é o problema da soma de sub-conjuntos e que consiste em dado um conjunto de inteiros (representado num vector), determinar se a soma de algum dos seus subconjuntos é igual a zero.

Mostre que se trata realmente de um problema da classe NP apresentando um algoritmo não determinístico que o resolva em tempo polinomial. Nomeadamente, indique o que a fase não determinística deve produzir e apresente uma definição em C da parte determinística do dito algoritmo.