

# Exame 21 janeiro 2020

## Algoritmos e Complexidade — Universidade do Minho

### Questão 1 [5 valores]

(i) Considere um algoritmo de *pesquisa da última ocorrência de um valor  $k$  entre os índices  $a$  e  $b$  num array de inteiros*. O índice da última ocorrência é guardado na variável  $r$ , que ficará com o valor  $a-1$  caso não exista qualquer ocorrência. Escreva uma especificação (**pré- e pós-condição**) para esta função.

(ii) O programa seguinte implementa este algoritmo. Apresente um invariante e um variante de ciclo que permitam provar a sua correcção total face à especificação que escreveu em (i).

```
int search(int v[], int a, int b, int k) {  
    i = b;  
    while (i >= a && v[i] != k)  
        i := i - 1;  
    r := i;  
}
```

(iii) Analise o tempo de execução do algoritmo no melhor e no pior caso, explicando esses casos.

### Questão 2 [5 valores]

A seguinte função testa de forma recursiva se *todos os elementos de um array de caracteres de comprimento  $N$  são iguais*:

```
int allequal (int v[], int N) {  
    if (N < 2) r = 1;  
    else r = (v[0] == v[1] && allequal(v+1, N-1));  
}
```

```
    return r;
}
```

- (i) Analise o tempo de execução desta função no **melhor e no pior caso**, apresentando e resolvendo as **recorrências** apropriadas.
- (ii) Apresente uma expressão para o **caso médio** do tempo de execução do algoritmo.

### Questão 3 [4 valores]

Considere uma estrutura de dados *Árvore Binária AVL* de números inteiros, alterada da seguinte forma: *é guardado em cada nó o número de elementos da árvore que tem esse nó como raiz.*

```
typedef enum balancefactor { LH , EH , RH } BalanceFactor;
typedef struct treenode {
    BalanceFactor bf;        // indicador de equilíbrio AVL
    int entry;               // inteiro guardado no nó
    int n_elems;             // número de elementos desta árvore
    struct treeNode *left, *right;
} *Tree;
```

- (i) Escreva a função de rotação à esquerda para estas árvores, que deverá naturalmente ajustar adequadamente o valor do campo `n_elems` (não é necessário ajustar o valor do campo `bf`):

```
Tree rotateLeft(Tree t) { ... }
```

- (ii) Pretende-se utilizar estas árvores para a implementação de *conjuntos de números naturais*, suportando em particular uma operação de *rank*: dado um inteiro `x`, contar o número de elementos do conjunto de valor inferior ou igual a ele. Implemente esta função de forma tão eficiente quanto possível, e analise o seu tempo de execução no melhor e no pior caso.

```
int rank(Tree t, int x) { ... }
```

Nas questões que se seguem considere os seguintes tipos de dados para a representação de grafos sem pesos, por matrizes e por listas de adjacências:

```
typedef int GraphM[MAX][MAX];  
struct edge {  
    int dest;  
    struct edge *next;  
};  
typedef struct edge *GraphL[MAX];
```

#### Questão 4 [4 valores]

Escreva uma função `count_reachable` que conta o número de vértices que são alcançáveis a partir de um vértice `s` num grafo orientado com `n` vértices, representado por uma matriz de adjacências. Analise o tempo de execução do algoritmo que escreveu, no melhor e no pior caso.

```
int count_reachable (GraphM g, int s, int n) { ... }
```

#### Questão 5 [2 valores]

Um grafo orientado diz-se *fortemente ligado* se para qualquer par de vértices  $u, v$ , existem caminhos quer de  $u$  para  $v$  quer de  $v$  para  $u$  (i.e. cada vértice é alcançável a partir do outro). Escreva uma função que testa se um grafo orientado (representado por listas de adjacências) é ou não fortemente ligado. A sua função deverá ser tão eficiente quanto possível.

```
int stronglyConnected (GraphL g, int n) { ... }
```