Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro

Algoritmos e Complexidade

Exame de Recurso — 2. Parte — 29 de Junho de 2009

1 – Seja dado um *"array"* com n ($n \geq 2$) elementos, em que cada elemento é uma letra minúscula.

Pretende-se manipular os elementos do *"array"* de maneira a separar as vogais das consoantes, usando de **modo eficiente** a estratégia do algoritmo **"Bubblesort"**.

Note que:

1. as vogais deverão ficar na primeira partição, e as consoantes na segunda;
2. o algoritmo deverá terminar, logo que seja verificado que as vogais e as consoantes já se encontram separadas.

a) Desenvolva a função que permite, dado um *"array"* de letras minúsculas, separar as vogais das consoantes.

b) Efectue a análise da complexidade do algoritmo da função anterior para o **Melhor Caso**, em termos das **comparações** e das **trocas** de caracteres efectuadas. Identifique **instâncias** do *"array"* que conduzam a esse caso.

c) Efectue a análise da complexidade do algoritmo da função anterior para o **Pior Caso**, em termos das **comparações** e das **trocas** de caracteres efectuadas. Identifique **instâncias** do *"array"* que conduzam a esse caso.

2 – Considere o tipo abstracto de dados **Árvore Binária de Inteiros**, em cujos nós é possível armazenar um número inteiro.

Considere também que os números inteiros se encontram registados **"em-ordem"** crescente.

Elabore uma função **recursiva e eficiente** que permita:

Listar, em ordem crescente, todos os elementos armazenados numa dada árvore e que pertençam ao intervalo $[a, b]$.

3 – Considere o tipo abstracto de dados **Grafo**, definido usando a **matriz de adjacências** que representa um dado grafo $G(V, E)$, com n vértices e m arestas.

Note que, assim, os n vértices de um grafo se encontram identificados pela sequência de números inteiros $0, 1, \ldots, (n-1)$.

Dado um vértice $v_i \in V$, pretende-se listar todos os vértices v_j alcançáveis a partir de v_i.

Essa listagem deverá ser iniciada pelo vértice v_i e corresponder a uma **travessia em profundidade** do grafo em que é indicado uma vez, para cada vértice alcançado v_j, o número de arestas que definem o primeiro caminho determinado de v_i para v_j, usando a travessia efectuada.

Assim, cada vértice alcançado só deverá ocorrer uma vez na listagem.

Por exemplo, para um dado grafo e para o caso dos vértices alcançáveis a partir do vértice 1, obter-se-ia:

```
No   1  -  Distancia   0
No   2  -  Distancia   1
No   4  -  Distancia   2
No   6  -  Distancia   3
No   3  -  Distancia   1
No   5  -  Distancia   2
```

Dado um grafo e um seu vértice v_i:

a) Desenvolva uma função que, usando uma estratégia **iterativa**, efectue a listagem desejada.

b) Desenvolva agora uma outra função que, usando uma estratégia **recursiva**, efectue a listagem desejada.

Atenção:

— Não se esqueça de que o grafo pode conter ciclos.

— Assuma que está definida uma função que devolve o elemento da linha i e da coluna j de uma matriz de adjacências m:

```
int getElemMatAdj( matriz m, int i, int j );
```

— Assuma que estão definidos os tipos abstractos **Pilha** e **Fila**; não é necessário implementá-los, caso os queira utilizar.

— Desenvolva outras eventuais funções auxiliares de que possa necessitar.