Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro

Algoritmos e Complexidade

Exame de Recurso — 2. Parte — 5 de Julho de 2010

1 – Considere um *"array"* de n elementos inteiros, todos distintos, e que se encontram armazenados por ordem crescente, a menos de, eventualmente, uma (ou mais) rotação(ões) apropriada(s).

Isto é, poderá não se encontrar armazenada a sequência ordenada de números inteiros, mas sim uma sua *permutação circular*. Por exemplo, os inteiros $\{1, 2, 3, 4, 5, 6\}$ poderão estar armazenados como $\{5, 6, 1, 2, 3, 4\}$.

2 **a)** Desenvolva uma função eficiente que permita *"rodar"* os elementos de um *"array"* desse tipo, de modo a que $v[i] < v[i+1]$, para $i = 0, 1, \ldots, n-2$.

Não deve copiar os elementos do *"array"* dado para um *"array"* auxiliar.

Construa as eventuais funções auxiliares de que possa necessitar.

0,5 **b)** Efectue a análise da complexidade do algoritmo desenvolvido, relativamente ao número de atribuições e de comparações associadas a elementos do vector, para o **Melhor Caso**. Identifique instâncias do vector que conduzam a esse caso.

1,5 **c)** Efectue a análise da complexidade do algoritmo desenvolvido, relativamente ao número de atribuições e de comparações associadas a elementos do vector, para o **Pior Caso**. Identifique instâncias do vector que conduzam a esse caso.

1,5 **d)** Efectue agora a análise da complexidade do algoritmo desenvolvido, relativamente ao número de atribuições e de comparações associadas a elementos do vector, para o **Caso Médio**.

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

Além dos ponteiros para as suas sub-árvores, cada nó contém um ponteiro adicional para o seu nó progenitor.

Considere também que os números inteiros **não** se encontram registados em qualquer ordem particular.

Desenvolva funções eficientes que permitam:

1,0 **a)** Dado um ponteiro para um nó de uma árvore, determinar o seu nível.

2,0 **b)** Dados ponteiros para dois nós distintos, a e b, de uma árvore, identificar o seu antepassado comum mais próximo.

Atenção: Para facilitar, considere que qualquer nó da árvore é *"antepassado"* de si próprio.

(v.s.f.f.)

4 **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. As distâncias (inteiros não negativos) associadas às arestas estão armazenadas numa segunda matriz, também $(n \times n)$.

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

Para um vértice $v_i \in V$, pretende-se obter a *Árvore dos Caminhos Mais Curtos* com raiz em v_i, usando o **Algoritmo de Dijkstra**.

Dado um grafo e um seu vértice v_i:

Desenvolva uma função que, usando o Algoritmo de Dijkstra, obtenha a informação que define a Árvore dos Caminhos Mais Curtos (i.e., o predecessor de cada vértice e a distância à raiz da árvore).

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 m:

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

— Assuma que está definido o tipo abstracto **PQueue-Dijkstra**, que implementa uma fila com prioridade orientada para os mínimos e permite armazenar o índice e o rótulo associado a cada vértice do grafo.

— Desenvolva outras eventuais funções auxiliares de que possa necessitar.