Métodos de Programação II

ESI / MCC

Ano Lectivo de 2004/2005 (1a Chamada)

Questão 1 Considere a função `min` definida da seguinte forma

```
int min(int A[], int a, int b)
{
 int i, m;

 m = A[a];
 i = a+1;
 while (i <= b)
   if (m > A[i]) {
        m = A[i];
        i = i+1;
   }
   else i = i+1;

 return m;
}
```

1. Mostre que a função `min(A,a,b)` calcula correctamente o menor inteiro presente entre as posições `a` e `b` do array `A`. Isto é, prove que

   $$\{\ a \leq b\ \}\ \mathbf{F}\ \{\ \forall a \leq p \leq b.\ \ m \leq A[p]\ \}$$

   F representa todo o corpo da função `min` excepto a instrução `return m`.

2. Mostre que `min(A,a,b)` determina o elemento mínimo do array `A` entre as posições `a` e `b`, em tempo linear em ordem a `b-a`.

Questão 2 Considere a seguinte declaração de dados para implementar tabelas de hash com tratamento de colisões por encadeamento.

```
typedef Entry   *HashTable[HASHSIZE]

typedef struct entry {
  int           key;
  char          *value;
  struct entry  *next;
} Entry;
```

1. Explique o funcionamento de uma tabela deste tipo, referindo-se nomeadamente; à estratégia de resolução de colisões de chaves; dimensionamento da tabela (como deverá ser o valor de `HASHSIZE`); complexidade em termos de tempos de execução das operações básicas de inserção, remoção e consulta, para uma tabela bem dimensionada para o volume da informação a guardar.

2. Considere agora que para fazer a listagem da informação ordenada por chave, se pretende indexar a informação da tabela utilizando árvores binárias de procura. Para isso, definiram-se os seguintes tipos de dados.

```
typedef struct node {
  int          key;
  Entry        *info;;
  struct node  *left;
  struct node  *right;
} Node;

typedef Node   *Tree
```

   Defina uma função que recebe uma tabela de hash e gera uma árvore binária de procura que indexa a informação dada nessa tabela. A função terá o seguinte protótipo:

```
Tree geraArv (HashTable table);
```

3. Compare, em termos gerais, a complexidade da operação básica de consulta na tabela de hash, com uma hipotética operação de consulta que utilize a árvore binária de procura. Refira-se ao melhor e ao pior caso de execução.

Questão 3 As duas travessias de grafos estudadas nas aulas diferem apenas na estrutura de dados usada para armazenar os próximos vértices a serem visitados. Enquanto que na travessia *Depth-first* se usa uma *stack*, na *Breadth-first* usa-se uma *queue*. Considere agora uma outra alternativa de travessia em que a estrutura de dados utilizada para armazenar estes nodos é uma lista, ordenada por ordem crescente do peso da aresta que liga esse nodo aos já visitados.
Apresente uma codificação desta variante, assumindo que o grafo se encontra armazenado como listas de adjacência.

Questão 4 Considere que para um determinado problema se desenvolveram dois programas: P1 e P2:

P1 é um programa que determina uma solução para o problema. O seu tempo de execução (em função da dimensão N do seu input) é dado pela função T_1.

P2 é um programa que, dada uma solução proposta, verifica se se trata de uma solução válida. O seu tempo de execução (em função da dimensão N do seu input) é dado pela função T_2.

$$T_1(N) = \begin{cases} c_1 & \text{se} \quad N \leq 1 \\ 4T_1(N/3) & \text{se} \quad N > 1 \end{cases}$$

$$T_2(N) = \begin{cases} c_2 & \text{se} \quad N \leq 1 \\ 3T_2(N/3) & \text{se} \quad N > 1 \end{cases}$$

1. Para cada uma das recorrências T_1 e T_2, *adivinhe* uma solução e mostre informalmente a sua validade.

2. Em qual (ou quais) das classes de complexidade estudadas (P, NP e NP-completo) poderá incluir o problema em causa? Justifique a sua resposta.