Exame de Algoritmos e Complexidade
LCC LEI
10 de Fevereiro 2009

Parte I

Esta parte 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 exame.**

1. Desenhe os estados sucessivos da árvore AVL de números inteiros resultante da inserção por esta ordem dos números 50, 20, 10, 60, 40, 30, e 70. Identifique claramente todas as rotações efectuadas.

2. Analise o tempo de execução no melhor e no pior caso da seguinte função, que determina, para cada posição i do array A, se $A[i]$ é inferior a todos os elementos do array B até à posição $i-1$.

```
void minimos (int A[], int B[], int  C[]) {
  for (i = N-1; i >= 0; i--) {
    C[i] = 1;
    j = 0;
    while (C[i] && j < i) {
      if (B[j] <= A[i]) C[i] = 0;
      j++;
    }
  }
}
```

3. Analise o tempo de execução da seguinte função recursiva, utilizando para isso uma recorrência.

```
int binarymin (int a[], int first, int last) {
  if (first == last) return first;
  if (first < last) {
    int mid = (first + last) / 2;
    int m1 = binarymin (a, first, mid);
    int m2 = binarymin (a, mid+1, last);
    if (a[m1] <= a[m2]) return m1; else return m2;
   }
}
```

4. Escreva uma função `buildBST` que, dado um *array de inteiros ordenado por ordem crescente* `arr` com `n` elementos, constrói uma *árvore binária de procura* balanceada com todos os elementos do array.

```
typedef struct node {
  into info;
  struct node *esq, *dir;
} *Node;
Node buildBST(int arr[], int n);
```

5. Dados dois grafos R e S com o mesmo conjunto de vértices V, a composição dos grafos R e S, denota-se por $R \circ S$ e tem uma aresta de a para b sse existe um vértice c tal que a aresta $a \rightarrow c$ existe em S e a aresta $c \rightarrow b$ existe em R. Defina a função `void compoe (Grafo r, Grafo s, Grafo ros)` que coloca no terceiro argumento o resultado da composição dos 2 primeiros grafos. Assuma que os grafos estão representados em matrizes de adjacência.

```
#define N ...
typdef int Grafo [N][N];
```

6. Defina uma função `int maisLongo(Grafo g, int o)` que, dado um grafo orientado e acíclico (representado em matriz de adjacências) e um vértice, calcula o comprimento do caminho mais longo que tem como origem esse vértice

Parte II

Resolva 4 das seguintes 5 questões, à sua escolha.

1. Identifique um invariante que descreva adequadamente o comportamento de um dos ciclos da função da alínea 2 da primeira parte. Prove a sua inicialização e preservação.

2. Pretende-se uma estrutura de dados para representar conjuntos de inteiros (entre 0 e 1000), que suporte as seguintes operações: inserção, teste de pertença e listagem ordenada no écran. Defina um tipo de dados apropriado e as funções referidas, garantindo que o teste de pertença executa em tempo constante e a listagem ordenada em tempo linear (no número de elementos do conjunto).

3. Analise o tempo de execução no pior caso da seguinte função, sobre um grafo não-orientado e ligado, com N nós e M arcos. Considere que a matriz `visitado` contém inicialmente 0 em todas as suas posições. O grafo está representado por uma matriz de adjacência.

```
void funcao (Grafo g, int i) {
  for (j = 0; j < N; j++)
    if (g[i][j] && !visitado[i][j]) {
      visitado[i][j] = 1; visitado[j][i] = 1;
      funcao(g, j);
    }
  printf("%d ", i);
}
```

4. O problema de *subset sum* é um conhecido problema de decisão. Tem como entrada um conjunto C de **n** números inteiros (representados, por exemplo, por um array) e um número inteiro **s**, e dá como resposta se existe um sub-conjunto A de C (i.e. $A \subseteq C$) tal que a soma dos elementos de A é igual a **s**. Mostre que este problema é NP, descrevendo um algoritmo não determinístico polinomial que o resolva.

   Não se esqueça de definir exactamente qual o resultado da fase não determinística bem como de mostrar que a fase determinística é polinomial no tamanho do input do problema.

5. Defina uma função (algoritmo determinístico, possivelmente exponencial) para resolver o problema de *subset sum* descrito acima. Pronuncie-se sobre a complexidade dessa função no pior caso.