# Algoritmos e Complexidade
# 2° Ano – LEI

Exame – Duração: 2 horas

13 de Fevereiro de 2008

## Parte I

Esta parte do exame representa 12 valores da cotação total. Cada uma das 6 alíneas está cotada em 2 valores.
**A obtenção de uma classificação abaixo de 8 valores nesta parte implica a reprovação no exame.**

1. Determine as condições de verificação necessárias para provar a correcção parcial da seguinte função anotada.

```
int bubble(int a[], int n) {
   int i, k;
   // n > 0
   i = n-1; k = 0;
   // n > 0 && i = n-1 && k = 0
   while (i > 0) {// i >= 0 && k < n
      if (a[i] < a[i-1]) {swap(a,i,i-1); k++}
      i--;
   }
   // k < n
   return k;
}
```

2. Analise a complexidade da seguinte função identificando o melhor e pior casos.

```
void bsort(int a[], int n) {
   while (bubble(a,n));
}
```

3. É possível armazenar um grafo orientado não pesado usando dois arrays: o primeiro armazena as arestas ordenadas por vértice de origem, indicando para cada uma delas qual o vértice destino; o segundo indica em que posição do primeiro array começam as arestas com origem em cada vértice; a última posição do array de vértices é usada para indicar qual é a primeira posição livre do array de arestas. Assumindo que o grafo tem exactamente `V` vértices e no máximo `A` arestas, este tipo de dados (`Grafo1`) pode ser definido da seguinte forma:

```
typedef struct {
   int arestas[A];
   int vertices[V+1];} Grafo1;
```

   Implemente uma função `void converte (Grafo1 *g, Grafo2 h)` que permita converter um grafo do tipo `Grafo1` num grafo implementado com listas de adjacência (`Grafo2`).

```
typedef struct arco {
   int destino;
   struct arco *next;} Arco;
typedef Arco *Grafo2 [V];
```

4. Pretende-se usar uma tabela de *hash* para armazenar o conjunto das matrículas dos carros com acesso aos parques reservados da universidade. Assumindo que o tratamento das colisões é feito usando *chaining*, esta tabela pode ser implementada da seguinte forma:

```
#define SIZE 1009
typedef struct no {
   char matricula[6];
   struct no *next;
} No;
typedef No *Tabela[SIZE];
```

   - Implemente uma função de *hash* razoável para este problema.

     `int hash(char matricula[6]);`

   - Implemente a função de inserção de uma matrícula na tabela, garantindo que não se armazenam matrículas repetidas.

     `int insert(Tabela t, char matricula[6]);`

5. Considere o seguinte tipo para implementar uma árvore AVL de inteiros.

```
typedef struct no {
   int info;
   int bal;
   struct no *esq, *dir;
} No;
typedef No *Arvore;
```

   Implemente a função `Arvore rr(Arvore arv)` que faz uma rotação simples para a direita numa determinada sub-árvore. Não se esqueça de actualizar o factor de balanço nos nós envolvidos na rotação.

6. Analise a complexidade da seguinte função que calcula os factores de balanço de uma árvore. Assuma que árvore está balanceada e que a função `altura` executa em tempo linear no tamanho da árvore de entrada.

```
void bals(Arvore a) {
   if (!a) return;
   a->bal = altura(a->dir) - altura(a->esq);
   bals(a->esq);
   bals(a->dir);
}
```

# Parte II

1. O diâmetro de um grafo pesado define-se como a distância (minima) entre os seus nodos mais afastados. Dado um grafo pesado representado com **listas de adjacências**, compare em termos de complexidade as seguintes alternativas para efectuar esse cálculo: (1) Uso repetido do algoritmo de Dijkstra ou (2) conversão para matrizes de adjacência e uso do algoritmo de Warshall.

2. Num grafo não orientado, um *click* é um subconjunto de vértices em que quaisquer dois elementos estão ligados por uma aresta.

   (a) Defina uma função que dado um grafo e um conjunto de vértices determina se esse conjunto é um *click*.

   (b) Um *click* diz-se maximal se qualquer conjunto que o contenha não for um *click*. Defina uma função que, dado um grafo e um vértice calcula um *click* maximal que contenha o vértice dado.

3. Demonstre que a função `bubble` apresentada na questão 1 da primeira parte, satisfaz as seguintes propriedades: (1) o número de trocas efectuadas é menor do que `n` e (2) coloca na primeira posição do vector o menor dos elementos do array.