Algoritmos e Complexidade Métodos de Programação II
2° Ano – LEI/LESI

Exame – Época Especial: 2:30 horas

23 de Setembro 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 factorial(int n) {
   int i, k;
   // n >= 0
   k = 1; i=0;
   // n > 0 && k = 1
   while (i < n) {// i <= n 0 && k = i!
      i++; k*=i;
   }
   // k = n!
   return k;
}
```

2. Analise a complexidade da seguinte função que usa a função da alínea anterior:

```
void factoriais (int a[], int n) {
   int i;
   for (i=0; (i<n); i++) a[i] = factorial (i);
}
```

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;
```

   Defina uma função que, para esta representação, calcule o grau de entrada de um dado vértice (i.e., o número de antecessores desse vértice).

4. Analise a complexidade da seguinte função que calcula quantos elementos de um vector de inteiros são menores do que um dado inteiro:

```
int menores (int a [], int n, int x) {
  int u, l, m;
  l = 0; u=n-1;
  while (l<u) {
     m = (l+u) / 2;
     if (a[m] < x) l = m+1;
     else u=m;
  }
  return l
}
```

5. Para o ciclo da função da alínea anterior, determine um variante que lhe permita mostrar que a função termina. Indique além disso as condições que devem ser verificadas para completar essa demonstração.

6. Relembre o algoritmo de inserção balanceada (AVL). Identifique pelo menos um dos casos em que o desbalanceamento não pode ser resolvido por uma rotação simples, dizendo, para esse caso qual a estratégia para voltar a balancear a árvore.

Parte II

1. Mostre, apresentando um exemplo (pre-condição, pós-condição e programa), que nem todos os programas parcialmente correctos estão correctos.

2. O problema de coloração de grafos é conhecido como sendo um problema difícil. O que pretendemos aqui é validar soluções para esse problema.

   Considere o tipo `Grafo` para armazenar as arestas de um grafo.

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

   Uma coloração de um grafo com `V` vértices pode ser guardada num vector com `V` inteiros, cada um deles identificando a cor do vértice respectivo. Defina então uma função `int corOK (Grafo g, int cor[V])` que determina se a coloração `cor` é apropriada, i.e., que não existem vértices adjacentes com a mesma cor. A função deve retornar `0` se tal **não** for o caso ou então deve retornar o número de cores diferentes usadas.

3. Analise a complexidade da função apresentada na alínea anterior, em função do número de vértices e arestas do grafo. Indique se necessário o melhor e o pior casos.

4. Com base no resultado da alínea anterior, justifique a afirmação *"o problema da coloração de grafos é um problema NP"*.