Algoritmos e Complexidade
2º Ano – LEI

Teste – Duração: 2 horas

23 de Janeiro de 2008

Parte I

Esta parte do teste 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 teste.

1. Faça a análise assimptótica do tempo de execução da função `minpairs`.

```
typedef struct {int p; int s;} Pair;

void minpairs(Pair a[], char v[], int n) {
   int i,j;
   for (i=0; i<n, i++) v[i]=1;
   for (i=n-1; i>=0; i--)
      for (j=i-1; j>=0; j--)
         if (a[i].s <= a[j].s) v[j]=0;
}
```

2. Determine as condições de verificação necessárias para provar a correcção parcial do seguinte programa (anotado em comentários) usado para calcular o máximo divisor comum entre dois números positivos.

```
// (a = a0 > 0) /\ (b = b0 > 0)
while (a != b)
    // mdc(a0,b0) = mdc(a,b)
    if (a > b) a = a - b;
    else b = b - a;
// a = mdc(a0,b0)
```

3. Considere a seguinte função recursiva:

```
void exemplo(int a[], int n) {
   int x = n/2;
   if (n>0) {
      exemplo(a,x);
      processa(a,n);
      exemplo(a+x,n-x);
   }
}
```

   Sabendo que $T_{processa}(n) = \Theta(n)$, escreva uma recorrência que descreva o comportamento temporal da função `exemplo` e indique, justificando, a solução dessa recorrência.

4. Pretende-se implementar um dicionário de sinónimos usando uma *tabela de hash*. Esta tabela pode ser definida da seguinte forma, onde a cada palavra está associada uma lista ligada com os seus sinónimos.

```
typedef struct s {
   char *sin;
   struct s *next;
} Sin;

typedef struct p {
   char *pal;
   Sin *sins;
   struct p *next;
} Pal;

#define TAM ...
typedef Pal *Dic[TAM];

int hash(char *pal);
```

Implemente a seguinte função que, dada uma palavra, imprime todos os seus sinónimos:

```
void sinonimos(Dic d, char *pal);
```

5. Considere o seguinte tipo de dados que permite representar grafos orientados e não pesados usando listas de adjacências.

```
typedef struct arco {
  int dest;
  struct arco *seg;
} Arco;

#define TAM ... // número de vértices do grafo
typedef Arco *Grafo[TAM];
```

Implemente a seguinte função que determina se há caminho entre dois vértices:

```
int haCaminho(Grafo g, int o, int d);
```

6. Relembre o conceito de *min-heap*. Implemente uma função que converte uma *min-heap* representada num array para uma *min-heap* representada como uma árvore do tipo indicado.

```
#define TAM ...

typedef int Heap[TAM];

typedef struct nodo {
   int val;
   struct no *esq, *dir;
} Nodo, *Tree;
```

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 numa matriz de adjacências implemente uma função que calcule o seu diâmetro.

2. Justifique a seguinte afirmação:

   > *Na inserção de um elemento numa AVL, no caso em que o balanceamento se faz com uma rotação simples para a direita, o factor de balanço dos dois nodos envolvidos fica zero.*

3. Defina uma função (o mais eficiente possível) que receba a árvore resultado de uma travessia de um grafo (vector dos pais) e faça uma travessia pre-order dessa árvore (imprimindo os índices correspondente). Considere que o grafo tem `N` vértices e que a raíz da árvore tem pai `-1`.