Algoritmos e Complexidade

Exame de Época Especial

16 de Setembro 2020

Questão 1 Recorde a estrutura de dados *min-heap*, e considere a seguinte função `hsort`, que ordena um *array* recorrendo a uma *min-heap* auxiliar. A função auxiliar `heapify` transforma um *array* numa *min-heap*.

```
void heapify (int v[], int N) {
  int i;
  for (i=(N-2)/2; i>=0; i--)
    bubbledown (v,N,i);
}

void hsort (int v[], int N) {
  heapify (v,N);
  while (--N > 0) {
    swap (v,0,N);
    bubbledown (v,N,0);
  }
}
```

1. Tendo em conta as propriedades das *min-heaps*, apresente um contrato (i.e., pré e pós-condições) para a função `heapify`.

2. Escreva agora um invariante para o ciclo da função `hsort`, apropriado para provar que implementa um algoritmo de ordenação.

Questão 2 Considere o seguinte problema:

> *Dadas duas sequências numéricas não ordenadas, sem repetições, X e Y, de comprimento máximo N, encontrar os M maiores elementos de X que não aparecem em Y.*

1. Considere que X e Y são implementadas como vectores (*arrays*) de números inteiros de tamanho N. Implemente a função `maiores1`, que resolve este problema da forma mais eficiente que lhe parecer possível.

   ```
   void maiores(int X[], int Y[], int N, int M, int resultado[]);
   ```

2. Analise o tempo de execução da função no*melhor e no pior caso*, bem como no *caso médio*.

Questão 3 Assuma a seguinte definição de tipos para árvores binárias, bem como a definição de uma função que testa se uma dada árvore binária é uma árvore AVL:

```
typedef struct node {
  int elem;
  struct node *esq;
  struct node *dir;
} Node, *ArvBin;

int avl (ArvBin t)
{
  if (!t) return 1;
  if abs(altura(t->esq) - altura(t->dir)) > 1 return 0;
  if (avl(t->esq) && avl(t->dir)) return 1;
}
```

Sabendo que a função `altura` executa em tempo linear no número de nós da árvore recebida como argumento, analise o comportamento da função AVL no melhor e no pior caso. Para isso identifique claramente esses casos, e escreva e resolva as recorrências respectivas.

Questão 4

1. Considere os seguintes tipos de dados para a representação de grafos orientados sem pesos, por listas de adjacências:

   ```
   struct edge {
     int dest;
     struct edge *next;
   };
   typedef struct edge *GraphL[MAX];
   ```

   Escreva uma função `int most_reachable (GraphL g, int N)` que determina qual o vértice do grafo g, com N vértices, que é alcançável a partir de um maior número de outros vértices do grafo. Se existir mais de um vértice nestas condições, deverá calcular um desses vértices.

2. Repita o exercício anterior, considerando agora que os grafos são representados por matrizes de adjacências:

   ```
   typedef struct GraphM[MAX][MAX];
   int most_reachable (GraphM g, int N)
   ```

3. Analise o tempo de execução, no melhor e no pior caso, de ambas as funções que escreveu.