Métodos de Programação II

LESI / LMCC

15 de Junho 2004

Questão 1 [*Análise de correcção de algoritmos*]
Considere o problema de se encontrar o maior número de valores iguais num **array ordenado**, e um algoritmo que encontra a solução percorrendo o array apenas uma vez, armazenando uma estimativa do valor correcto numa variável `size`.

```
int maiorRepet(int A[], int n) {
   int i;
   int size;

   // Outras declaracoes

   if (n<0) return 0;
   else if (n==1) return 1;
   else {
     // Inicializacoes

     for (i=0; i< n; i++) {
        // Teste e actualizacao da variavel size
     }
   }
}
```

1. (a) Que propriedade têm os valores iguais, quando armazenados num array ordenado?
   (b) Que propriedade deverá satisfazer a variável `size` quando o algoritmo terminar?
   (c) Que região do array tem de ser considerada em cada iteração i do algoritmo, por forma a garantir a actualização correcta da variável `size`?
   (d) Que teste tem de ser efectuado para que essa actualização seja correcta?
2. Com base na resposta à alínea anterior, apresente um algoritmo que resolva o problema em questão, completando a função `maiorRepet`. Escreva ainda um invariante de ciclo para o algoritmo que escreveu, e utilize-o para demonstrar a sua correcção.

Nome:_____ **Número:**_____ **Curso:**_____

Métodos de Programação II

LESI / LMCC

15 de Junho 2004

Questão 2 [*Estruturas de Dados*] Uma *heap* é uma árvore binária que verifica a seguinte propriedade: o conteúdo de cada nó é menor ou igual que o conteúdo dos seus descendentes (não havendo, no entanto, qualquer relação de ordem entre os conteúdos das duas sub-árvores de um mesmo nó). A função de inserção pode ser escrita como se segue:

```
typedef struct node {
  int num;
  struct node *esq, *dir;
} *Heap;

void swap (Heap h)
{
  Heap aux = h->esq;
  h->esq = h->dir;
  h->dir = aux;
}

Heap insert (int x, Heap h)
{
  if (!h) {
    Heap new = malloc (sizeof(struct node));
    new -> num = x;
    new -> esq = new -> dir = NULL;
    return new;
  }
  if (x < h->num) {
    h->dir = insert(h->num, h->dir);
    swap(h);
    h->num = x;
  }
  else {
    h->dir = insert(x, h->dir);
    swap(h);
  }
  return h;
}
```

Nome:_____ **Número:**_____ **Curso:**_____

Métodos de Programação II

LESI / LMCC

15 de Junho 2004

1. Desenhe a *heap* que resulta de se inserir consecutivamente os números 10, 20, 30, 22, 21, 35, 40, 24 e 27.

2. Com base numa recorrência, efectue a análise assimptótica do tempo de execução da função `insert`.

3. Escreva uma função com protótipo

   ```
   int heap2sortedarray(Heap h, int A[], int j);
   ```

   que converte uma *heap* num vector ordenado, começando na posição `j` do vector, e devolve o número de elementos colocados no vector.

   Sugestão: Pode utilizar uma de duas estratégias alternativas para tratar os elementos armazenados nas duas *sub-heaps*:

   - converter recursivamente cada *sub-heap* numa sequência ordenada, e fundir os resultados; ou
   - fundir as duas *sub-heaps* numa só estrutura do mesmo tipo, e prosseguir recursivamente.

4. Com o auxílio de uma *heap* é possível ordenar uma sequência como se segue:

   ```
   void hsort (int A[], int n)
   {
     int i;
     Heap h = NULL;
     for (i=1; i<=n; i++)
       h = insert(A[i], h);
     heap2sortedarray(h, A, 1);
   }
   ```

   Analise o tempo de execução deste algoritmo.

Nome:_____ **Número:**_____ **Curso:**_____

Métodos de Programação II

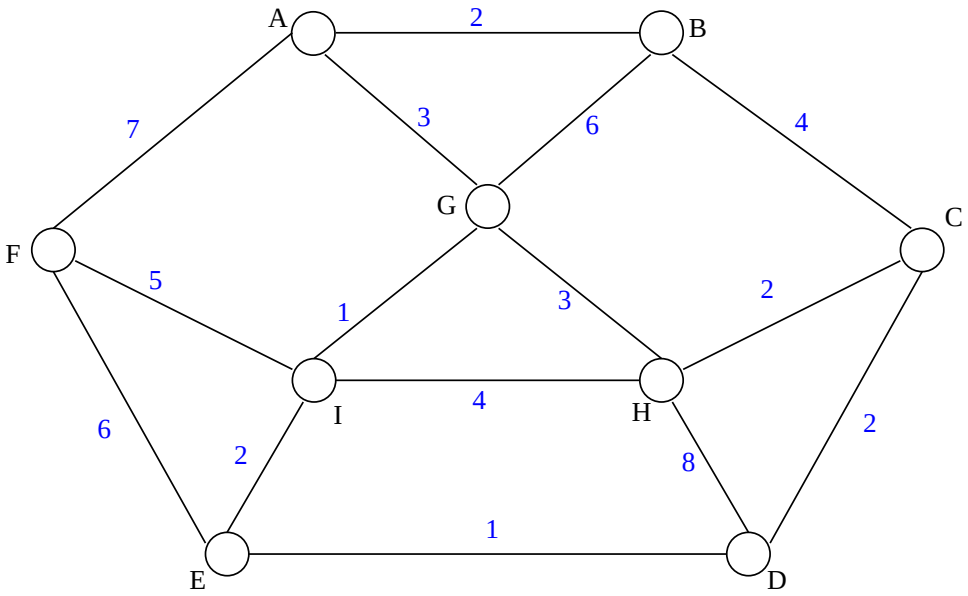
LESI / LMCC

15 de Junho 2004

Questão 3 [*Algoritmos de Grafos*] A versão simétrica do algoritmo de Dijkstra para o cálculo do caminho mais curto (entre um par de nós A e B, num grafo não-orientado) consiste na execução alternada de um passo do algoritmo a partir de cada nó, o que implica construir simultaneamente duas árvores (V_A, T_A) e (V_B, T_B), com raízes A e B. Esta versão permite optimizar a execução do algoritmo em certos casos.

1. Durante a execução do algoritmo simétrico, caso exista um caminho entre X e Y, atingir-se-á um estado em que um nó pertence a ambas as árvores V_X e V_Y. É então garantido que o caminho mais curto entre X e Y só pode passar por nós contidos em $V_X \cup V_Y$. Diga justificando se esta afirmação é verdadeira ou falsa.

2. Aplique o algoritmo simétrico para calcular o caminho de F para C no grafo seguinte. Diga justificando se este algoritmo optimizado apresenta vantagens em relação ao algoritmo tradicional, neste caso concreto.



Nome:_____ **Número:**_____ **Curso:**_____

Questão 4 [*Problemas NP-Completos*]
Considere o seguinte problema:

> **Problema**: Cobertura Exacta por Conjuntos de 3 Elementos
> **Input**: Um conjunto X, de tamanho múltiplo de 3, e uma colecção C, de sub-conjuntos de 3 elementos de X.
> **Pergunta**: Será que C contém uma cobertura exacta de X, isto é, existe uma sub-colecção C' de C, tal que cada elemento de X apareça uma e uma só vez em C'?
> **Exemplo**:
> X = {1,2,3,4,5,6,7,8,9},
> C={{1,2,3},{2,3,4},{4,5,6},{7,8,9}},
> C'={{1,2,3},{4,5,6},{7,8,9}}.

1. Explique o conceito de problema de decisão NP-completo, e diga que formas conhece de demonstrar que um problema é NP-completo, com base na noção de redução polinomial.

2. Sabendo que o problema apresentado acima é NP-completo, demonstre que o seguinte problema é também NP-completo:

> **Problema**: Cobertura Mínima
> **Input**: Um conjunto X e uma colecção C, de sub-conjuntos de X. Um inteiro K.
> **Pergunta**: Será que C contém uma cobertura de X de tamanho K, isto é, existe uma sub-colecção C' de C, com K ou menos sub-conjuntos de X, tal que cada elemento de X apareça uma e uma só vez em C'?

Nome:_____ **Número:**_____ **Curso:**_____