Algoritmos e Complexidade

LMCC

30 de Janeiro 2006

Questão 1 Considere o algoritmo de Prim para a construção de uma árvore geradora de custo mínimo num grafo não-orientado:

```
void MST((V, E)) {
  V' = {x}; T' = ∅;        /* x escolhido arbitrariamente */
  stuck = 0;
  while (V' ≠ V && !stuck) {
    for (y ∈ orla, y adjacente a x)
      if (w(x, y) < w(arco candidato de y))
        substituír arco candidato de y por (x, y);
    for (y ∉ V', y ∉ orla, y adjacente a x) {
      colocar y na orla;
      marcar (x, y) arco candidato;
    }
    if (não há arcos candidatos) stuck = 1;
    else { escolher arco candidato (u, v) de custo mínimo; x = v;
           V' = V' ∪ {x}; T' = T' ∪ {(u, v)};
           remover x da orla;
           desmarcar (u, v) como candidato;
    }
  }
}
```

1. O ciclo principal do algoritmo observa o seguinte invariante:

   No início de cada iteração do ciclo while, (V', T') *é uma sub-árvore de uma árvore geradora mínima de* (V, E).

   Prove que o invariante é verdadeiro no início da execução do ciclo (*inicialização*). Estude a *terminação* do algoritmo e diga se se pode concluír do invariante que o algoritmo constroi sempre uma árvore geradora mínima do grafo.

2. Efectue, explicando todos os passos, a análise do tempo de execução assimptótico do algoritmo, com base nas seguintes componentes

   - Tempo das operações de inicialização;
   - Tempo das operações executadas dentro dos ciclos *for* (análise agregada);
   - Número total de comparações executadas nas escolhas de arcos candidatos;
   - Restantes instruções do condicional `if` / `else`.

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

Algoritmos e Complexidade

LMCC

30 de Janeiro 2006

Questão 2 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;
}
```

1. Escreva uma equação de recorrência e analise o tempo de execução da função `avl`, utilizando notação assimptótica apropriada. Justifique o tempo de execução que considerou para a função `altura`.

2. Implemente agora a função `extractMin`, que extrai (i.e., devolve e remove da árvore) o elemento de menor valor de uma árvore binária de pesquisa, sem utilizar recursividade. Analise o seu tempo de execução assimptótico.

3. Escreva um invariante para o ciclo principal da função `extractMin`, e utilize-o para provar a sua correcção.

   ```
   Tree extractMin(ArvBin t, int *x);
   ```

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

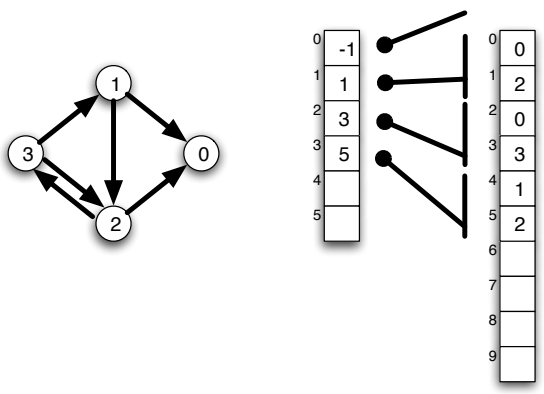
Algoritmos e Complexidade

LMCC

30 de Janeiro 2006

3

Questão 3 Considere a seguinte estrutura de dados para representar um grafo: existem dois vectores – num, cujo tamanho é o número máximo de arcos, guardam-se sequencialmente as adjacências dos vários vértices; num outro, cujo tamanho é o número máximo de vértices, guardam-se as posições das última adjacência associada a cada um dos vértices (ver figura com exemplo de um grafo e respectiva representação).



1. Defina a função `addAdj` que adiciona uma adjacência a um dado vértice (no caso de já existir, não altera nada).

2. Defina uma função que converta a representação apresentada numa matriz de adjacências (defina os tipos de dados que entender apropriados).

3. Represente o grafo desenhado em cima por uma matriz de adjacências. Relembre o algoritmo de Warshall para o cálculo do fecho transitivo de um grafo:

```
voidWarshallTC (int A[][], int R[][], int n){
/* Grafo representado por A, fecho transitivo calculado em R */
   R = copia_matriz(A);
   for(k=1; k<=n; k++)
     for(i=1; i<=n; i++)
       for(j=1; j<=n; j++)
         if(R[i][k] && R[k][j]) R[i][j]=1;
}
```

   Simule a sua execução sobre este grafo, identificando claramente a ordem pela qual os arcos são acrescentados.

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