Rio de Janeiro, 11 de Setembro de 2013.
PROVA 2 DE ANÁLISE DE ALGORITMOS
PROFESSOR: EDUARDO SANY LABER
DURAÇÃO: 1:50h

- Algumas questões pedem para explicações com palavras. Pode-se utilizar referências a algoritmos vistos em aula e a explicação deve ser suficientemente boa para permitir que alguém implemente, sem muitas dúvidas, a solução proposta.
- Cabe ao aluno deixar bastante clara a solução proposta e não ao professor gastar muita energia para entendê-la.

1. (2.4pt) Seja $G = (V, E, w)$ um grafo conexo direcionado com pesos nas aresta. Considere a versão do algoritmo de Dijkstra apresentada abaixo que encontra o peso do caminho de peso mínimo entre a origem s e um nó destino t. A estrutura PQ é uma fila de prioridade e a operação $ExtractMin(PQ)$ remove o elemento de PQ com menor valor de π.

   1. $\pi(s) \leftarrow 0$
   2. Insira s em PQ
   3. **Para** $v \in V - \{s\}$
   4. $\quad \pi(v) \leftarrow \infty$
   5. $\quad$ Insira v em PQ
   6. **Fim Para**
   7. **Enquanto** $t \notin S$
   8. $\quad v \leftarrow ExtractMin(PQ)$
   9. $\quad d(v) \leftarrow \pi(v)$; Insira v em S;
   10. $\quad$ **Para** $u \in Adj[v]$
   11. $\quad\quad$ **Se** $u \notin S$ e $\pi(v) + w_{vu} < \pi(u)$
   12. $\quad\quad\quad \pi(u) \leftarrow \pi(v) + w_{vu}$
   13. $\quad\quad$ **Fim Se**
   14. $\quad$ **Fim Para**
   15. **Fim Enquanto**

   a) Analise a complexidade do algoritmo quando a estrutura PQ é implementada como uma lista encadeada ordenada em ordem crescente dos valores de $\pi(\cdot)$. Note que sempre depois de uma operação a lista tem que voltar a ser ordenada.

   b) Modifique o código do algoritmo para que ele compute o número de arestas do caminho de peso mínimo entre s e o nó t.

2. (2.4pt) Seja $G = (V, E)$ um grafo direcionado acíclico.

   a) Quantas componentes fortemente conexas tem G? Por que?

   b) Ao executar uma DFS em G obtemos $Pos[v]$ para todo vértice $v \in V$. Seja O uma ordenação dos vértices em ordem decrescente destes valores de Pos. Podemos afirmar que O é uma ordenação topológica para G? Por que?

3. (2.4pt) Considere o pseudo código abaixo que executa em um grafo não direcionado $G = (V, E)$ que tem n vértices e m arestas.

```
1.  Para v ∈ V
2.      EXISTE(v) ← FALSE
3.      Para u ∈ Adj[v]
4.          Se existe uma caminho entre u e v no grafo G − uv
5.              EXISTE[v] ← TRUE
6.          Fim Se
7.      Fim Para
8.  Fim Para
```

a) Analise a sua complexidade em função de n e m. Assuma que a existência de um caminho na linha 4 é verificada através de uma BFS.

b) O que podemos afirmar para os vértices v tal que `EXISTE`[v]=TRUE ao término do algoritmo? E para os os vértices v tal que `EXISTE`[v]=FALSE ao término do algoritmo?

4. (2.4pt) Seja um grafo $G = (V, E)$ não direcionado com pesos nas arestas e seja **f** uma aresta de G.

a) Assuma que os pesos das arestas são **positivos**. Explique com palavras como seria um algoritmo polinomial para encontrar a árvore geradora com menor peso dentre as árvores geradoras para G que **não** contém a aresta **f**. Analise a complexidade deste algoritmo.

b) Para este item assuma que algumas arestas podem ter pesos negativos. Explique com palavras como seria um algoritmo polinomial para encontrar o subgrafo conexo gerador de G de peso mínimo.

5. (2.4pt) Considere um conjunto de n listas $L_1, \ldots, L_n$, cada uma delas contendo números inteiros. Desejamos obter uma única lista ordenada contendo os números de todas as n listas. A nossa estratégia consiste em ir realizando merges sucessivos entre pares de listas até sobrar uma única lista. Existem várias ordens possíveis para executar os merges, algumas mais eficientes do que outras. Por examplo, se nossas listas são L_1, L_2, L_3, temos três possibilidades: (i) fazer o merge de L_1 com L_2, obtendo uma lista L_4 e depois fazer o merge entre L_4 e L_3; (ii) fazer o merge de L_1 com L_3, obtendo uma lista L_4 e depois fazer o merge entre L_4 e L_2 e (iii) fazer o merge de L_2 com L_3, obtendo uma lista L_4 e depois fazer o merge entre L_4 e L_1. Se temos 4 listas L_1, L_2, L_3, L_4, uma das possibilidades seria fazer o merge de L_1 com L_2, obtendo uma lista L_5, depois fazer o merge entre L_3 e L_4, obtendo a lista L_6 e, finalmente, fazer o merge entre L_5 e L_6.

   Considere o seguinte procedimento guloso que é utilizado para definir a melhor ordem possível para executar os merges. A variável *custo* armazena uma estimativa do tempo computacional necessário para realizar a sequência de merges. Note que a estratégia abaixo não realiza merges, apenas computa a ordem em que eles devem ser realizados.

   Para $j = 1...n$

   $A[j] \leftarrow |L_j|$

   Insira i na estrutura C

   Fim Para

   $custo \leftarrow 0$; $i \leftarrow 0$

   Enquanto $|C| \geq 2$

   Seja k o índice do conjunto C tal que $A[k]$ é mínimo

   Remova k de C

   Seja j o índice do conjunto C tal que $A[j]$ é mínimo

   Remova j de C

   Output "Merge L_k e L_j"

   $custo \leftarrow custo + A[k] + A[j]$

   $i++$

   $A[n+i] \leftarrow A[k] + A[j]$

   Insira $(n+i)$ no conjunto C % corresponde a criar uma nova lista com índice $n+i$

   Fim Para

   a) Qual *custo* o algoritmo vai devolver quando $|L_i| = 2^{i-1}$ para $i = 1, \ldots, n$?

   b) Como devemos armazenar o conjunto C de modo a obter uma implementação eficiente para o algoritmo acima? Qual a complexidade obtida?