Pitágoras — Teoria da Computação

Trabalho Prático 2 - Biblioteca para Teoria de Grafos

Prof. Fernando Braz — 1º Semestre de 2015

Observações:

1. Comece a fazer este trabalho imediatamente. Você nunca terá tanto tempo para resolvê-la quanto agora!
2. **Demonstração valendo ponto na aula de:** 26/11/2015 ou 27/11/2015 (dependendo da sua turma)
3. Forma de entrega: arquivo compactado por e-mail (fernando.braz@pitagoras.com.br).
4. O trabalho prático é em grupo de até cinco pessoas. Obviamente que vocês podem discutir o trabalho com outros grupos, mas cópias literais serão anuladas, tanto do grupo copiador quanto do grupo copiado.

Implemente uma **biblioteca** para teoria de grafos, contendo estruturas de dados e algoritmos para representar e processar grafos. A biblioteca deve implementar cada uma das funcionalidades relacionadas abaixo, de forma fácil e prática para que possa ser utilizada por outros desenvolvedores.

- Representação de grafos direcionados ou não, com peso ou não
- Implementação dos algoritmos:
  - Cálculo da distância de uma rota
  - Busca em profundidade e busca em largura
  - Algoritmo de Dijkstra para encontrar o menor caminho de um vértice para os demais vértices
  - **(Pontos-Extra)** Algoritmos de Prim e Kruskal para obtenção de árvore/floresta geradora mínima

Além disso, a biblioteca deve possuir um programa principal que imprima como saída o resultado para cada um dos comandos relacionados em um arquivo de entrada (ambos descritos a seguir).

A entrada do programa principal é um arquivo texto contendo a descrição do grafo representado da seguinte forma:

Entrada de Exemplo

```
GRAFO
0 1 2 3; // Vértices do grafo
true ; // Se o grafo é direcionado ou não
true ; // Se o grafo tem pesos ou não

ARESTAS
0 1 10, // Existe uma aresta entre os vértices 0 e 1, e o peso dela é 10
1 2 11, // Existe uma aresta entre os vértices 1 e 2, e o peso dela é 11
0 2 20, // Existe uma aresta entre os vértices 0 e 2, e o peso dela é 20
2 3 12, // Existe uma aresta entre os vértices 2 e 3, e o peso dela é 12
1 3 13, // Existe uma aresta entre os vértices 1 e 3, e o peso dela é 13
2 0 14; // Existe uma aresta entre os vértices 2 e 0, e o peso dela é 14

COMANDOS
DISTANCIA 0 1 2; // Distância do caminho passando pelos vértices 0 1 2
DISTANCIA 0 1 2 0 2; // Distância do caminho passando pelos vértices 0 1 2 0 2
PROFUNDIDADE 0 2; // Busca em largura partindo do vértice 0 em busca do vértice 2
LARGURA 0 3; // Busca em largura partindo do vértice 0 em busca do vértice 3
MENOR CAMINHO 0 3; // Menor caminho entre os vértices 0 e 3
PRIM 2; // EXTRA. Árvore geradora mínima utilizando o algoritmo de Prim partindo do vértice 0
KRUSKAL; // EXTRA. Floresta geradora mínima utilizando o algoritmo de Kruskal
```

No arquivo de entrada, existem cabeçalhos entre as partes que descrevem o GRAFO, ARESTAS e COMANDOS. Observe que cada "parte" da descrição do grafo (vértices, se é direcionado ou não, se tem peso ou não) é finalizada com um ; (ponto-e-vírgula), e que as arestas do grafo são separadas por , (vírgula). Existem espaços entre cada caractere em cada linha. Além disso, as arestas são delimitadas por , (vírgula), sendo a última finalizada por um ; (ponto-e-vírgula). Finalmente, os comandos também são finalizados por ; (ponto-e-vírgula), e seus argumentos são separados por espaços.

Saída de Exemplo

```
DISTANCIA 0 1 2:
21 // Distância da rota 0 1 2

DISTANCIA 0 1 2 0 2:
55 // Distância da rota 0 1 2 0 2

PROFUNDIDADE 0 2:
0     // Origem
1 2   // 0 foi visitado, 1 e 2 são os vizinhos de 0
2 3 2 // 1 foi visitado, 2 e 3 são os vizinhos de 1 (começo). Destino (2) foi encontrado

LARGURA 0 3:
0     // Origem
1 2   // 0 foi visitado, 1 e 2 são os vizinhos de 0
2 2 3 // 1 foi visitado, 2 e 3 são os vizinhos de 1 (final)
2 3 3 // 2 foi visitado, 3 é vizinho de 2 (final)
3 3   // 2 já havia sido visitado. Destino (3) foi encontrado

MENOR CAMINHO 0 3:
0 1 3 // Menor caminho entre 0 e 3
23    // Distância do menor caminho entre 0 e 3

PRIM 2:
0 1 10,
2 3 12,
2 0 14;
36

KRUSKAL:
0 1 10,
1 2 11,
2 3 12;
33
```

No arquivo de saída, cada comando é repetido e finalizado por : (dois pontos), seguido do resultado do comando. O resultado de um comando é separado do próximo comando por uma quebra de linha. Para o comando de distância, basta imprimir a distância da rota. Para os comandos de busca em profundidade, basta imprimir a fronteira a cada passo (vide comentários no exemplo). Para o comando de menor caminho, basta imprimir a rota e a distância em linhas separadas. Finalmente, para os comandos de árvore/floresta geradora mínima, basta imprimir as arestas que pertencem a árvore/floresta no mesmo formato do arquivo de entrada e o peso total da árvore/floresta na linha seguinte às arestas.

O seu programa deve receber dois parâmetros pelo vetor `args[]` do método `main`, sendo o primeiro parâmetro **`-e`** que indica o arquivo de entrada (no exemplo, o arquivo se chama `entrada.txt`), e o segundo parâmetro **`-o`** que indica o arquivo de saída (no exemplo, o arquivo se chama `saida.txt`), conforme demonstrado abaixo (em Java, mas similar para outras linguagens):

```
java tp02 -e entrada.txt -s saida.txt
```

Critérios de Avaliação

- Acertos nos testes
- Modularização
- Legibilidade
- Qualidade
- Documentação

Pontos-Extra

Existem várias coisas que você pode fazer nesse trabalho prático para ganhar pontos-extra. A dificuldade do item está indicado ao lado, indo desde fácil ($\star$) até difícil ($\star\star\star\star\star$). A quantidade de pontos-extra obtidos é proporcional a dificuldade dos itens realizados e a qualidade da solução apresentada.

- ($\star\star\star$) Algoritmos de Prim e Kruskal mencionados previamente na descrição do trabalho prático
- ($\star$) Documente o trabalho utilizando um arquivo do tipo Markdown (README.md)
- ($\star$) Utilize alguma ferramenta de versionamento de código (SVN, Git, etc.) durante **TODA** a realização do trabalho prático
- ($\star$) Disponibilize o seu código em um repositório público (Bitbucket, Github, etc.)
- ($\star$) Utilize alguma ferramenta de automação (Maven, Ivy, Gradle, Grunt, etc.) para facilitar o processo de compilação e distribuição da sua biblioteca
- ($\star\star$) Faça testes unitários (de preferência utilizando a técnica TDD) para cada um dos seus algoritmos
- ($\star$) Utilize algum software para medir a cobertura do testes unitários no código fonte (Cobertura, Emma, Clover, etc.), atingindo pelo menos 90% de cobertura
- ($\star\star\star$) Implemente um serviço web em qualquer linguagem de programação que exponha sua biblioteca como uma API REST
- ($\star\star\star$) Implemente uma interface gráfica (desktop ou web) que permita visualizar o grafo descrito e executar qualquer um dos algoritmos
- ($\star\star\star\star$) Implemente uma interface gráfica (desktop ou web) que permita modelar um grafo qualquer
- ($\star\star\star\star\star$) Implemente uma interface gráfica (desktop ou web) que permita visualizar a execução passo-a-passo de cada algoritmo