

Documentação JavaGraphPajek

Emanuel Costa Pedro Henrique Ribeiro
Henrique Resende Carlos Eduardo Meints
Thiago Guimarães

Novembro 2023

1 Introdução

A biblioteca "JavaGraphPajek" é uma ferramenta em Java para criar, ler e exportar grafos, utilizando o formato Pajek para representação de grafos simples/intermediários.

2 Como Começar

2.1 Requisitos

Certifique-se de que você tenha o Java instalado na sua máquina.

2.2 Instalação

Faça o download da biblioteca "JavaGraphPajek". Adicione a biblioteca ao seu projeto Java.

3 Funcionalidades Principais

3.1 Criar um Grafo

Para criar um grafo direcionado, passe como parâmetro 'true', caso não queira direcionado, 'false'.

```
Grafo grafoDirecionado = new Grafo(true);
```

```
Grafo grafoNaoDirecionado = new Grafo(direcionado:false);
```

3.2 Criar Vértices

Para criar um vértice, basta passar como parâmetro um rótulo(String) e um peso(Double).

```
Vertice vertice1 = new Vertice(rotulo:"a", peso:10);  
Vertice vertice2 = new Vertice(rotulo:"b", peso:20);  
Vertice vertice3 = new Vertice(rotulo:"c", peso:30);
```

3.3 Adicionar Vértices

Para adicionar um vértice ao grafo já criado, basta usar a função 'adicionarVertice' passando como parâmetro o vértice que deseja adicionar.

```
grafoDirecionado.adicionarVertice(vertice1);  
grafoDirecionado.adicionarVertice(vertice2);  
grafoDirecionado.adicionarVertice(vertice3);
```

3.4 Criar Arestas

Para criar uma aresta, basta passar como parâmetro os dois vértices(Vertice), um peso(Double) e um rótulo(String).

```
Aresta aresta1 = new Aresta(vertice1, vertice2, peso:1, rotulo:"a");  
Aresta aresta2 = new Aresta(vertice2, vertice3, peso:2, rotulo:"b");  
Aresta aresta3 = new Aresta(vertice3, vertice1, peso:3, rotulo:"c");
```

3.5 Adicionar Arestas

Para adicionar uma aresta ao grafo já criado, basta usar a função 'adicionarAresta' passando como parâmetro a aresta(Aresta) que deseja adicionar.

```
grafoDirecionado.adicionarAresta(aresta1);  
grafoDirecionado.adicionarAresta(aresta2);  
grafoDirecionado.adicionarAresta(aresta3);
```

3.6 Remover aresta

Para remover uma aresta basta utilizar a função 'removerAresta' no grafo desejado passando como parâmetro a aresta(Aresta) que deseja remover, funcionando tanto em um grafo direcionado como um não direcionado.

```
grafoDirecionado.removerAresta(aresta3);  
grafoNaoDirecionado.removerAresta(arestaNaoDirecionada3);
```

3.7 Verificar existência de aresta entre dois vértices

Para verificar se uma aresta existe basta utilizar a função 'arestaExiste' no grafo desejado passando como parâmetro os dois vértices(Vertex) que deseja verificar, funcionando tanto em um grafo direcionado como um não direcionado.

```
grafoDirecionado.arestaExiste(vertice1, vertice2);  
grafoNaoDirecionado.arestaExiste(vertice1, vertice2);
```

3.8 Conferindo incidências em um grafo direcionado

Para conferir a incidência entre uma aresta e um vértice, basta utilizar a função 'incidenciaArestaVertice', passando como parâmetro a aresta(Aresta) e o vértice(Vertex) que deseja conferir.

```
23 boolean incidencia = grafoDirecionado.incidenciaArestaVertice(aresta2, vertice2);  
24 System.out.println("Aresta 2 incide no vértice 2? " + incidencia);  
25
```

PROBLEMS 2 OUTPUT PORTS GITLENS TERMINAL DEBUG CONSOLE

Aresta 2 incide no vértice 2? true

3.9 Conferindo se grafo é completo

Para conferir se o grafo é completo, basta utilizar a função 'verificaGrafoCompleto' no grafo desejado, funcionando tanto em um grafo direcionado como um não direcionado, retornando 'true' se for completo e 'false' se não.

```
grafoDirecionado.verificaGrafoCompleto()
```

```
grafoNaoDirecionado.verificaGrafoCompleto()
```

3.10 Conferindo se grafo é vazio

Para conferir se o grafo é vazio, basta utilizar a função 'verificaGrafoVazio' no grafo desejado, funcionando tanto em um grafo direcionado como um não direcionado, retornando 'true' se for vazio e 'false' se não.

```
grafoDirecionado.verificaGrafoVazio()
```

```
grafoNaoDirecionado.verificaGrafoVazio()
```

3.11 Matriz de adjacência

Para obter a matriz de adjacência, basta utilizar a função 'getMatrizAdjacencia' no grafo desejado, funcionando tanto em um grafo direcionado como um não direcionado, retorna uma matriz de inteiros.

```
grafoDirecionado.getMatrizAdjacencia();
```

```
grafoNaoDirecionado.getMatrizAdjacencia();
```

3.12 Ler um grafo a partir de um arquivo pajek

Para ler um grafo de um arquivo pajek, basta utilizar a classe 'GrafoPAJEK', criar uma instância dela e utilizar a função 'lerArquivoPAJEK' nessa instância, passando como parâmetro o caminho do arquivo(String) e se o grafo é direcionado ou não(Boolean). A função retorna o grafo gerado a partir do arquivo.

```
String caminhoArquivo = "grafoteste.net";  
Grafo grafo = GrafoPAJEK.lerArquivoPAJEK(caminhoArquivo, direcionado:true);
```

3.13 Gerar um arquivo pajek a partir de um grafo

Para gerar um arquivo pajek a partir de um grafo, basta utilizar a classe 'GrafoPAJEK', criar uma instância dela e utilizar a função 'gerarArquivoPAJEK' nessa instância, passando como parâmetro o caminho do arquivo(String) e o grafo que deseja exportar(Grafo).

```
GrafoPAJEK.gerarArquivoPAJEK(nomeArquivo, grafoTeste);
```

3.14 Aplicar Dijkstra em um grafo iniciando por um vértice específico

Para aplicar Dijkstra em um grafo, basta criar um grafo, criar um Map(Vértice, Double) para receber a o retorno da função grafo.dijkstra(Vértice origem), passando o vértice de origem, esse map terá os vértices que compõe o menor caminho entre esse vértice específico e todos os outros com o peso de cada um respectivamente.

```
Map<Vertice, Double> resultadoDijkstra = grafo.dijkstra(a);
```

3.15 Aplicar Dijkstra em um grafo para todos os vértices

Para aplicar Dijkstra em um grafo, basta criar um grafo, criar um Map(Vertice, Map(Vertice, Double)) para receber a o retorno da função grafo.dijkstraPorVertice(), esse Map terá os vértices, e para cada terá um Map com menor caminho entre o respectivo vértice e todos os outros com o peso de cada um respectivamente.

```
Map<Vertice, Map<Vertice, Double>> resultadoDijkstraPorVertice = grafo.dijkstraPorVertice();
```

3.16 Aplicar BellmanFord em um grafo iniciando por um vértice específico

Para aplicar BellmanFord em um grafo, basta criar um grafo, criar um Map(Vertice, Double) para receber a o retorno da função grafo.bellmanFord(Vertice origem), passando o vértice de origem, esse map terá as vértices que compõe o menor caminho entre esse vértice específico e todos os outros com o peso de cada um respectivamente.

```
Map<Vertice, Double> resultadoBellmanFordPorVertice = grafo.bellmanFord(origem);
```

3.17 Aplicar Floyd Warshall em um grafo

Para aplicar Floyd Warshall em um grafo, basta criar um grafo, criar um Map(Vertice, Map(Vertice, Double)) para receber a o retorno da função grafo.floydWarshall(), esse Map terá os vértices, e para cada terá um Map com menor caminho entre o respectivo vértice e todos os outros com o peso de cada um respectivamente.

```
Map<Vertice, Map<Vertice, Double>> distancias = grafo.floydWarshall();
```

3.18 A Estrela

Para aplicar A Estrela basta criar um grafo, criar uma List(Vertice) para receber a o retorno da função grafo.aEstrela(Vertice origem, Vertice destino), essa lista terá todos os vértices desde o vértice origem até o vértice de destino.

```
List<Vertice> distancias = meuGrafo.aEstrela(v1,v2);
```

4 Testes

4.1 Teste tempo de execução com 100 vértices

Obtivemos as seguintes médias de tempo de execução para um grafo de 100 vértices:

Dijkstra: 14 milissegundos
Bellman Ford: 7.3 milissegundos
Floyd Warshall: 79 milissegundos
Realizamos os testes em 3 computadores:

```
Dijkstra:
Tempo de execução: 5 milissegundos
-----
Bellman-Ford:
Tempo de execução: 5 milissegundos
-----
Floyd-Warshall:
Tempo de execução: 82 milissegundos
-----
```

```
Dijkstra:
Tempo de execução: 9 milissegundos
-----
Bellman-Ford:
Tempo de execução: 12 milissegundos
-----
Floyd-Warshall:
Tempo de execução: 104 milissegundos
-----
```

```
Dijkstra:
Tempo de execução: 34 milissegundos
-----
Bellman-Ford:
Tempo de execução: 5 milissegundos
-----
Floyd-Warshall:
Tempo de execução: 51 milissegundos
-----
```

4.2 Teste tempo de execução com 1.000 vértices

Obtivemos as seguintes médias de tempo de execução para um grafo de 1.000 vértices:

Dijkstra: 23 milissegundos
Bellman Ford: 109.6 milissegundos

Floyd Warshall: 37.2 segundos
Realizamos os testes em 3 computadores:

```
Dijkstra:
Tempo de execução: 12 milissegundos
-----
Bellman-Ford:
Há um ciclo de peso negativo, impossibilitando o algoritmo
Tempo de execução: 85 milissegundos
-----
Floyd-Warshall:
Há um ciclo de peso negativo, impossibilitando o algoritmo
Tempo de execução: 28035 milissegundos
-----
```

```
Dijkstra:
Tempo de execução: 20 milissegundos
-----
Bellman-Ford:
Há um ciclo de peso negativo, impossibilitando o algoritmo
Tempo de execução: 172 milissegundos
-----
Floyd-Warshall:
Há um ciclo de peso negativo, impossibilitando o algoritmo
Tempo de execução: 58247 milissegundos
-----
```

```
Dijkstra:
Tempo de execução: 37 milissegundos
-----
Bellman-Ford:
Há um ciclo de peso negativo, impossibilitando o algoritmo
Tempo de execução: 72 milissegundos
-----
Floyd-Warshall:
Há um ciclo de peso negativo, impossibilitando o algoritmo
Tempo de execução: 25383 milissegundos
-----
```

4.3 Teste tempo de execução com 10.000 vértices

Obtivemos as seguintes médias de tempo de execução para um grafo de 10.000 vértices:

Dijkstra: 54.6 milissegundos
Bellman Ford: 9.5 segundos
Realizamos os testes em 3 computadores:

```
Dijkstra:
Tempo de execução: 46 milissegundos
-----
Bellman-Ford:
Há um ciclo de peso negativo, impossibilitando o algoritmo
Tempo de execução: 8370 milissegundos
-----
```

```
Dijkstra:
Tempo de execução: 54 milissegundos
-----
Bellman-Ford:
Há um ciclo de peso negativo, impossibilitando o algoritmo
Tempo de execução: 13244 milissegundos
-----
```

```
Dijkstra:
Tempo de execução: 64 milissegundos
-----
Bellman-Ford:
Há um ciclo de peso negativo, impossibilitando o algoritmo
Tempo de execução: 7109 milissegundos
-----
```

Referências

Link para a especificação do formato Pajek: <https://pt.slideshare.net/AndrMarques18/tutorial-pajek>