Aula 07 – JGraphT

Notas de Aula de Teoria dos Grafos

Profª.: Patrícia D. L. Machado

UFCG – Unidade Acadêmica de Sistemas e Computação

Sumário

Introdução

Nesta aula, exploraremos recursos da **JGraphT** e também de classes do pacote **util** de **GraphTheory-JGraphT** para criar, importar e manipular grafos direcionados e ponderados. Utilizaremos implementações da JGraphT para calcular menor caminho e ciclo hamiltoniano.

Importando Grafos Direcionados no Formato GML

O procedimento é semelhante ao que usamos para grafos não-direcionados.

Criamos uma instância da classe **DefaultDirectedGraph**, utilizando a classe **DefaultVertex** e a classe **RelationshipDirectedEdge** para representar vértices e arcos com *labels*. Note que a classe **RelationshipDirectedEdge** é diferente da classe **RelationshipEdge** a qual usamos para representar arestas em grafos não-direcionados. A classe **RelationshipDirectedEdge** implementa o conceito de arco. Esta classe também está disponível no pacote **util.**

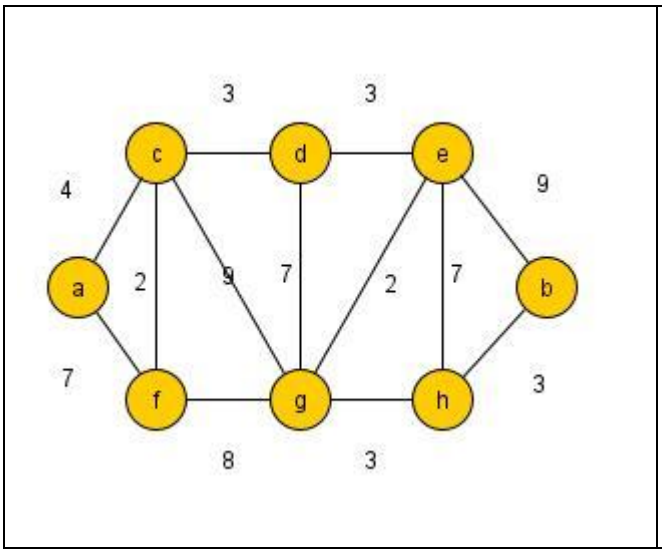
```
DefaultDirectedGraph<DefaultVertex,RelationshipDirectedEdge> g =
      new DefaultDirectedGraph<>(
            VertexEdgeUtil.createDefaultVertexSupplier(),
            VertexEdgeUtil.createRelationshipDirectedEdgeSupplier(),false);
```

Para realizar a importação, basta utilizar o método apropriado da classe **ImportUtil** do pacote **util**; para o grafo acima, o método *importDirectedGraphGML*, passando a instância do grafo a ser criado e o nome do arquivo.

```
ImportUtil.importDirectedGraphGML(g, graphpathname + "grid.gml");
```

Importando Grafos Simples Ponderados no Formato CSV (EDGE LIST)

Grafos ponderados podem ser representados por uma lista de arestas (EDGE_LIST) onde um terceiro elemento é adicionado: o peso da aresta. A tabela abaixo apresenta um grafo simples ponderado com a sua representação através de uma lista de arestas. Esta lista pode ser guardada em um arquivo no formato CSV para ser importada em um programa usando a **JGraphT**.




```
a,c,4
a,f,7
c,f,2
c,d,3
c,g,9
f,g,8
d,g,7
d,e,3
g,e,2
g,h,3
e,h,7
e,b,9
h,b,3
```

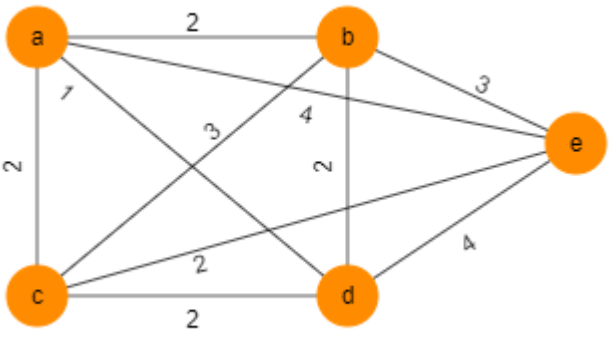
Para tal, criamos uma instância da classe **SimpleWeightedGraph** e importamos o arquivo usando o método *ImportWeightedGraphCSV* específico para WEIGHTED EDGE LIST da classe **ImportUtil** do pacote **util**.

```
SimpleWeightedGraph <DefaultVertex,DefaultWeightedEdge> graph =
      new SimpleWeightedGraph<DefaultVertex, DefaultWeightedEdge>(
            VertexEdgeUtil.createDefaultVertexSupplier(),
            SupplierUtil.createDefaultWeightedEdgeSupplier());

ImportUtil.importWeightedGraphCSV(
      graph, graphpathname + "weightededgelist.csv", false);
```

Importando Grafos Simples Ponderados no Formato CSV (MATRIX)

Grafos simples ponderados podem ser representados no formato de matriz de adjacência onde cada elemento da matriz representa um valor de peso caso os vértices sejam adjacentes. Para grafos não direcionados, a matriz deve ter apenas um dos seus quadrantes preenchidos, tal como no exemplo abaixo. Esta matriz pode ser guardada em um arquivo no formato CSV para ser importada em um programa usando a **JGraphT**.




```
,a,b,c,d,e
a,,2,2,1,4
b,,,3,2,3
c,,,,2,2
d,,,,,4
e,,,,,
```

Para tal, criamos novamente uma instância da classe **SimpleWeightedGraph** e importamos o arquivo usando o método *ImportWeightedGraphCSV* específico para o formato WEIGHTED MATRIX da classe **ImportUtil** do pacote **util**.

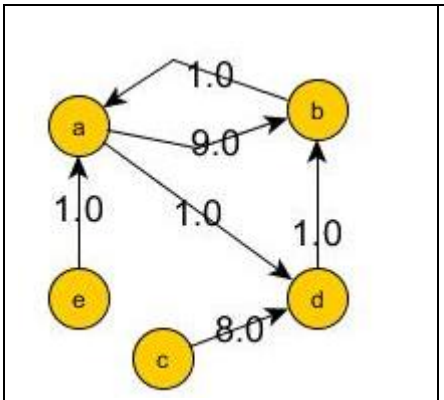
```
SimpleWeightedGraph <DefaultVertex,DefaultWeightedEdge> graph =
      new SimpleWeightedGraph<DefaultVertex, DefaultWeightedEdge>(
            VertexEdgeUtil.createDefaultVertexSupplier(),
            SupplierUtil.createDefaultWeightedEdgeSupplier());

ImportUtil.importWeightedGraphCSV(
      graph,
      graphpathname + "hamiltonian.csv",
      CSVFormat.MATRIX,false,true,true);
```

Importando Grafos Direcionados Estritos Ponderados no Formato CSV

O procedimento para importar grafos direcionados no formato CSV é semelhante ao seguido anteriormente para importar grafos não direcionados. A diferença é apenas com relação a classe

para representar o grafo que iremos criar. Abaixo temos um exemplo de um grafo direcionado e sua representação através de uma matriz de adjacência.




```
,a,b,c,d,e
a,,9.0,,1.0,
b,1.0,,,,
c,,,,8.0,
d,,1.0,,,
e,1.0,,,,
```

Para importar este grafo, criamos uma instância da classe **DefaultDirectedWeightedGraph** e usamos o método *ImportWeightedGraphCSV* específico para o formato WEIGHTED MATRIX da classe **ImportUtil** do pacote **util**.

```
Graph<DefaultVertex, DefaultWeightedEdge> graph =
      new DefaultDirectedWeightedGraph<DefaultVertex, DefaultWeightedEdge>(
            VertexEdgeUtil.createDefaultVertexSupplier(),
            SupplierUtil.createDefaultWeightedEdgeSupplier());

ImportUtil.importWeightedGraphCSV(
                        graph,
                        graphpathname + "csv-weighted-example.txt",
                        CSVFormat.MATRIX,
                        false,
                        true, // EDGE_WEIGHTS
                        true); // MATRIX_FORMAT_NODEID
```

Importando um Grafo a partir de Múltiplos arquivos CSV

No mundo real, grafos são criados a partir de bases de dados, onde vértices e arestas são objetos que possuem diferentes atributos. Neste caso, um dos formatos de entrada mais utilizados é o CSV, onde a primeira linha lista os nomes dos atributos separados por um delimitador, usualmente a vírgula, e as demais apresentam os dados, também separados por um delimitador. Neste caso, poderemos ter um ou mais arquivos para representar vértices e arestas a serem criados.

A classe **ImportUtil** do pacote **util** apresenta o método *importGraphMultipleCSV* que recebe os parâmetros descritos na tabela abaixo.

graph	Uma instância de `Graph<DefaultVertex,RelationshipWeightedEdge>` onde grafo será criado. O grafo pode ser ponderado ou não.
Vfilename	*FilePath* do sistema de arquivos indicando o arquivo CSV que descreve os vértices
idAtt	Nome do atributo do arquivo Vfilename que guarda o *id* dos vértices
labelAtt	Nome do atributo do arquivo Vfilename que guarda o *label* dos vértices
Efilename	*FilePath* do sistema de arquivos indicando o arquivo CSV que descreve as arestas
sourceAtt	Nome do atributo do arquivo Efilename que guarda o nome de um terminal da aresta.
targetAtt	Nome do outro atributo do arquivo Efilename que guarda o nome de um terminal da aresta.
weightAtt	Nome do atributo do arquivo Efilename que guarda o peso da aresta se existir.
simplegraph	O valor *true* indica que o grafo é simples
weighted	O valor *true* indica que o grafo é ponderado

Como exemplo, temos a chamada abaixo.

```
Graph<DefaultVertex, RelationshipWeightedEdge> unweightedgraph =
       new SimpleGraph <>
              (VertexEdgeUtil.createDefaultVertexSupplier(),
              VertexEdgeUtil.createRelationshipWeightedEdgeSupplier(),false);

ImportUtil.importGraphMultipleCSV(
       unweightedgraph,
       graphpathname + "estados.csv","Sigla","Nome",
       graphpathname + "fronteiras.csv", "Estado1", "Estado2", null, true, false);
```

Para compreendermos melhor os parâmetros utilizados na chamada do método importGraphMultipleCSV acima, vejamos a estrutura dos arquivos estados.csv e fronteira.csv na tabela abaixo. O primeiro representa os vértices e o segundo as arestas.

estados.csv	**fronteiras.csv**
Sigla,Nome,Litoral	ID,Estado1,Estado2,Distancia
MA,Maranhão,true	1,MA,PI,329
PI,Piauí,true	2,PI,CE,495
CE,Ceará,true	3,PI,PE,934
RN,R.Norte,true	4,PI,BA,994
PB,Paraíba,true	5,CE,PE,629
PE,Pernambuco,true	6,CE,PB,688
AL,Alagoas,true	7,CE,RN,537
SE,Sergipe,true	8,PE,PB,104
BA,Bahia,true	9,RN,PB,185
	10,PE,BA,839
	11,PE,AL,202
	12,BA,AL,475
	13,BA,SE,277
	14,AL,SE,201

No arquivo **estados.csv**, na primeira linha, observamos os nomes dos atributos. Dentre estes, no comando de importação acima, escolhemos *Sigla* para representar o *id* de cada vértice e *Nome* para representar o *label* de cada vértice (lembrando que o *label* é a informação que apresentamos para cada vértice nas visualizações e *id* é um campo para referência interna, em particular, é o valor utilizado no arquivo de arestas para indicar os seus terminais). Os demais atributos, se existirem, serão criados e farão parte da instância de cada vértice, podendo ser acessados através do método *getAtt* da classe **DefaultVertex** para cada vértice (por exemplo, *v.getAtt("Litoral")*, onde *v* é um vértice, instância da classe **DefaultVertex**).

No arquivo **fronteiras.csv**, na primeira linha, observamos os nomes dos atributos. Dentre eles, podemos observar no comando de importação acima que escolhemos *Estado1* e *Estado2* para representar os terminais da aresta através do seu valor de *id*. Da mesma forma que para os atributos dos vértices, os demais atributos da tabela, se existirem, são armazenados em cada objeto aresta e podem ser acessados através do método *getAtt* da classe **RelationshipWeightedEdge** (por exemplo, *e.getAtt("Distancia")*, onde *e* é uma aresta, instância da classe **RelationshipWeightedEdge**). Note que o atributo "*Distancia*" representa a distância entre as capitais dos estados relacionados e esta informação pode também ser utilizada como peso para as arestas do grafo. Assim, poderemos encontrar os menores caminhos reais.