

Aula 03 – Lab – JGraphT

Laboratório de Teoria dos Grafos

Prof^a: Patrícia D. L. Machado

UFCG – Unidade Acadêmica de Sistemas e Computação

Sumário

Criando um Grafo Simples	1
Criando Multigrafos e Pseudografos.....	3
Importando Grafos no formato CSV	4
Importando Grafos no formato GML.....	8

Este roteiro apresenta, algumas atividades básicas para criação e importação de grafos. Antes de construir o seu primeiro grafo, se estiver usando a IDE eclipse, crie um pacote (dentro do projeto do repositório que importou) para guardar seus programas. Assim poderá comparar sua produção com os exemplos já disponíveis no repositório ou mesmo consultá-los.

Se estiver usando o repl.it, crie uma cópia do repl em sua conta. Instruções sobre como fazer esta cópia encontra-se nas notas de aula para a Aula 03 (JGraphT).

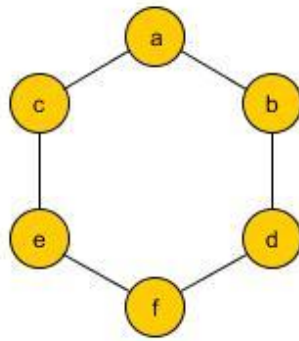
O repl configurado para este lab encontra-se neste endereço:

<https://repl.it/@pdlm/Aula03Lab>

Criando um Grafo Simples

TAREFA 01: O objetivo é criar um programa para construir o grafo $G = (V(G), E(G))$ abaixo ilustrado, onde $V(G) = \{a, b, c, d, e, f\}$ e $E(G) = \{ab, bd, df, ef, ce, ca\}$.

Se usando o repl.it, edite a classe **Tarefa01.java**.



Para tal, utilizaremos a classe [SimpleGraph](#), visto que se trata de um grafo simples. Os vértices serão instâncias da classe **String** e as arestas serão instâncias da classe **DefaultEdge**. A razão para esta escolha é que nosso grafo não possui outros atributos em vértices e arestas além de seu identificador. A classe [DefaultEdge](#) implementa uma aresta simples que identifica basicamente os dois vértices incidentes. Esta é a classe mais básica para arestas que não possuem atributos. Utilizamos desta classe basicamente o seu construtor. Métodos para inspeção de uma aresta estão disponíveis na classe de Grafo que estiver utilizando ou nas classes utilitárias que veremos mais à frente.

O código deverá importar as classes da JGraphT que serão utilizadas:

```
import org.jgrapht.graph.DefaultEdge;
import org.jgrapht.Graph;
import org.jgrapht.graph.SimpleGraph;
```

Em seguida, criamos uma instância da classe **SimpleGraph** para armazenar o grafo. Isto pode ser feito no método main da classe.

```
Graph<String,DefaultEdge> graph =
    new SimpleGraph<String,DefaultEdge> (DefaultEdge.class);
```

Aqui vemos que **SimpleGraph** é parametrizada pelos tipos dos vértices e arestas, neste caso **String** e **DefaultEdge** respectivamente. O construtor recebe como parâmetro a classe que supre (Supplier) de objetos do tipo de arestas.

Agora podemos adicionar um a um os vértices e arestas do grafo. Para tal, utilizados os métodos *addVertex* e *addEdge* da classe **SimpleGraph**.

```
graph.addVertex("a");
graph.addVertex("b");
...
graph.addEdge("a","b");
...
```

Em seguida, podemos imprimir no console o conjunto de vértices e o conjunto de arestas do grafo criado. Para tal, podemos usar os métodos *edgeSet* e *vertexSet* da interface **Graph**.

```
System.out.println(graph.edgeSet());  
System.out.println(graph.vertexSet());
```

Podemos também inspecionar o grafo, por exemplo, consultando o grau e as arestas incidentes em um vértice usando os métodos *degreeOf* e *edgesOf*. E utilizarmos outros métodos de inspeção disponíveis na interface **Graph**.

Um exemplo de um código semelhante ao proposto encontra-se na classe *Aula03MyFirstGraph* (pacote **classexamples** do nosso repositório).

Criando Multigrafos e Pseudografos

Vamos agora ver como criar multigrafos e pseudografos. Para tal, utilizamos as classes [Multigraph](#) e [Pseudograph](#) respectivamente, visto que a classe **SimpleGraph** não admite loops e arestas paralelas. Utilizamos também a classe **RelationshipEdge** que faz parte do pacote **util** do repositório de código. Esta classe admite que arestas possuam atributos, dentre eles, um *label*.

Os passos são semelhantes ao roteiro anterior. Nosso código deverá importar as classes da JGraphT e do nosso repositório que serão utilizadas.

```
import org.jgrapht.Graph;  
import org.jgrapht.graph.Pseudograph;  
import util.RelationshipEdge;
```

O método principal então cria uma instância do grafo.

```
Graph<String,RelationshipEdge> graph =  
    new Pseudograph<String,RelationshipEdge> (RelationshipEdge.class);
```

Vértices podem ser adicionados usando o método *addVertex* que usamos anteriormente, mas para arestas usaremos o método *addEdge* que recebe 3 parâmetros, onde os dois primeiros são os terminais e o terceiro é uma instância da classe *RelationshipEdge*. Se a aresta possui apenas um *label* como atributo, podemos usar o construtor que recebe este *label* como parâmetro para criar a instância de **RelationshipEdge**.

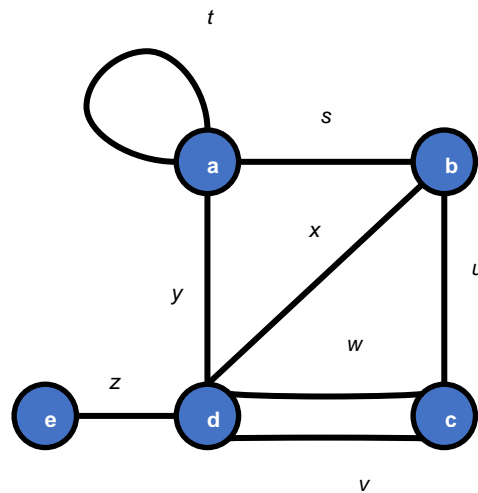
```

graph.addVertex("a");
graph.addVertex("b");
graph.addVertex("c");
graph.addEdge("a","b",new RelationshipEdge("ab1"));
graph.addEdge("a","b",new RelationshipEdge("ab2"));
graph.addEdge("a","c",new RelationshipEdge("ac"));
graph.addEdge("b","b",new RelationshipEdge("bb"));

```

TAREFA 02: Agora, construa um programa que crie o pseudografo que usamos como exemplo na Aula 02 (ilustrado abaixo). Veja um exemplo de um programa semelhante na classe Aula03MyPseudoGraph do pacote **classexamples**.

Se usando o repl.it, edite a classe **Tarefa02.java**.



Importando Grafos no formato CSV

Criar grafos adicionando vértices e arestas um a um é uma tarefa cansativa e inviável para grafos maiores. Assim, a melhor abordagem é criar grafos usando outras ferramentas tais como a yEd ou mesmo editores de texto e planilhas eletrônicas e exportá-los para serem carregados no seu programa JGraphT. Veremos agora como importar usando a JGraphT um grafo exportado em um arquivo CSV.

Para tal utilizaremos as classes **ImportUtil** e **VertexEdgeUtil** do pacote **util** de nosso repositório da JGraphT. Estas classes foram criadas a fim de simplificar o procedimento necessário a importação, abstraindo vários detalhes mais complexos considerados pela JGraphT. Estas classes devem ser importadas em seu código juntamente com outras classes da JGraphT:

```
import org.jgrapht.nio.csv.CSVFormat;  
import org.jgrapht.util.SupplierUtil;  
import util.ImportUtil;  
import util.VertexEdgeUtil;
```

Vamos considerar a importação de um arquivo no formato CSV que apresenta um grafo definido como uma lista de arestas. O conteúdo deste arquivo pode ser visto abaixo.

```
a,b  
b,c  
c,d  
e,f  
f,a
```

Como se trata de um grafo simples, vamos criar uma instância da classe **SimpleGraph**, como fizemos anteriormente, mas desta vez utilizaremos uma classe específica para os vértices, a Classe **util.DefaultVertex**, e a classe convencional da JGraphT para representar arestas, a classe **DefaultEdge**. Por que não definir os vértices com sendo do tipo **String** como fizemos anteriormente, já que os vértices não têm outros atributos? A resposta é que na importação, a JGraphT cria seus próprios identificadores para os vértices e assim os *labels* dos vértices tais como constam no arquivo de entrada seriam perdidos. Na classe **DefaultVertex**, os vértices podem ter atributos e assim estes *labels* serão armazenados e poderão ser referenciados.

Outra diferença na criação do grafo é que usaremos outro construtor da classe **SimpleGraph**. Este construtor recebe como parâmetros duas classes que são responsáveis por construir as instâncias de vértices e arestas (*Suppliers*). Este é um requisito das classes que realizam importação na JGraphT.

```
SimpleGraph<DefaultVertex, DefaultEdge> graph = new SimpleGraph <>  
    (VertexEdgeUtil.createDefaultVertexSupplier(), SupplierUtil.createDefaultEdgeSupplier(), false);
```

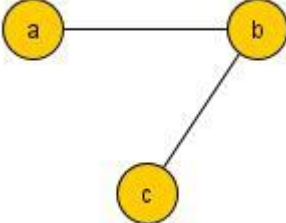
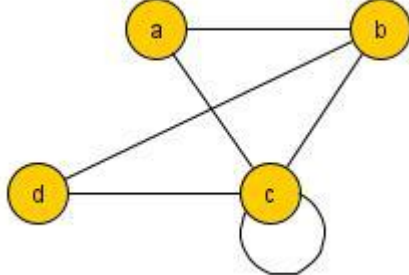
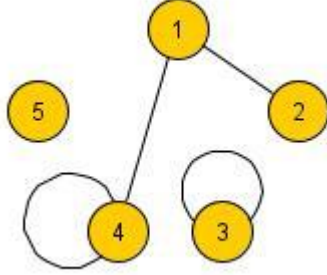
O *supplier* para vértices está implementado na classe **util.VertexEdgeUtil**, enquanto o *supplier* para arestas é o padrão para **DefaultEdge** definido na classe **SupplierUtil** da JGraphT. O terceiro parâmetro indica que este grafo não tem pesos nas arestas. Grafos com pesos é assunto de aulas posteriores.

Uma vez criado o grafo, agora invocamos o método *importGraphCSV* da classe **util.ImportUtil** que recebe como parâmetro o objeto grafo criado, o path onde o arquivo .csv pode ser encontrado, e a definição do formato deste arquivo, neste caso, *CSVFormat.EDGE_LIST*.

```
ImportUtil.importGraphCSV(graph, "./src/main/java/graphs/csv-example.txt",  
    CSVFormat.EDGE_LIST);
```

Com isto, o arquivo será lido e suas informações serão importadas na instância de grafo passada como parâmetro. A classe **Aula03ImportSimpleGraphEdgeCSV** do pacote **classexamples** apresenta um exemplo com este procedimento.

Existem três tipos básicos de formato de grafo que podem ser importados em um arquivo .csv como mostra a tabela abaixo.

EDGE_LIST	ADJACENCY_LIST	MATRIX
a,b b,c	a,b b,c,d c,a,c,d	0,1,0,1,0 1,0,0,0,0 0,0,1,0,0 1,0,0,1,0 0,0,0,0,0
		

Observe que no formato ADJACENCY_LIST, o primeiro elemento de cada linha é o vértice seguido por seus vizinhos. Como o grafo não é direcionado, não é necessário listar todos os vizinhos em ambos os sentidos. Por exemplo, o vértice **a** aparece na lista dos vizinhos de **c**, mas **c** não aparece na lista dos vizinhos de **a**. Mesmo assim a aresta **ac** será criada. Para este formato, podemos usar o mesmo método *importGraphCSV* que usamos para importar no formato EDGE_LIST; basta passar o formato correto.

TAREFA 03: Agora, crie um arquivo com a lista de adjacência apresentada na tabela e construa um programa para importar este arquivo. Observe que diferente do exemplo anterior o grafo é um pseudografo.

Se usando o repl.it, edite a classe **Tarefa03.java**.

Para importar no formato MATRIX, utilizaremos outro método *importGraphCVS* de **util.ImportUtil**, já que este formato possui parâmetros adicionais. No exemplo da tabela acima, o arquivo no formato matriz define um grafo com 5 vértices. Como os nomes dos vértices não foram fornecidos, eles serão criados como 1,2,3,4,5. Como esta é uma matriz de adjacências, note que o 1 indica a existência de uma aresta entre dois terminais e o 0 a ausência. Uma aresta loop é indicada pelo valor 1.

Os parâmetros adicionais para matriz são valores booleanos que indicam se:

EDGE WEIGHTS

Os valores na matriz representam pesos nas arestas

MATRIX FORMAT NODEID

Os *labels* dos vértices são fornecidos na matriz

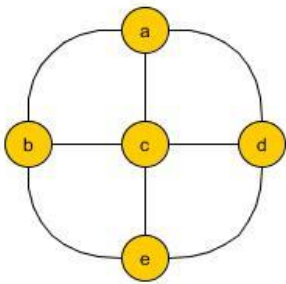
MATRIX FORMAT ZERO WHEN NO EDGE

É obrigatório indicar com 0 na matriz o fato de que dois vértices não são adjacentes. Caso não setado, basta deixar o campo em branco para indicar que dois vértices não são adjacentes.

Para o grafo acima, o método seria invocado com *true* para MATRIX FORMAT ZERO WHEN NO EDGE, *false* para MATRIX FORMAT NODEID e *false* para EDGE WEIGHTS. Lembrando que o formato de grafos com pesos (*weights*) em arestas será visto em aulas posteriores.

Agora, considere o seguinte arquivo, 5-3regular.csv, que se encontra no pacote **graphs** em nosso repositório:

,a,b,c,d,e	
a,,1,1,1,	
b,1,,1,,1	
c,,,,1,1	
d,,,,,1	
e,,,,,	

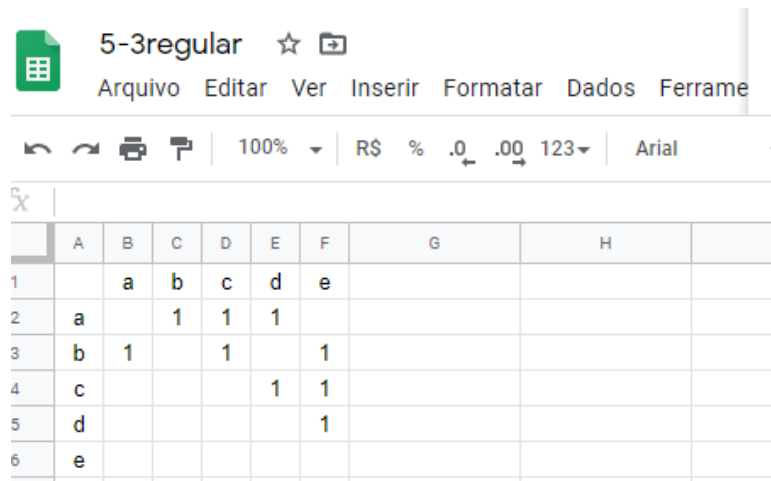


Para este grafo, o método seria invocado com *false* para MATRIX FORMAT ZERO WHEN NO EDGE (já que campos em branco no arquivo indicam que dois vértices não são adjacentes), *true* para MATRIX FORMAT NODEID (os *labels* dos vértices são fornecidos na primeira linha e na primeira coluna do arquivo) e *false* para EDGE WEIGHTS como podemos ver no comando abaixo.

```
ImportUtil.importGraphCSV( graph,  
    "/src/main/java/graphs/5-3regular.csv",  
    CSVFormat.MATRIX,  
    false,  
    false,  
    true); // MATRIX_FORMAT_NODEID
```

Um exemplo de programa que importa este grafo se encontra na classe `Aula03ImportSimpleGraphMatrixCSV` do pacote **classexamples**.

Para obtermos o arquivo de entrada, [5-3regular.csv](#), criamos uma planilha no Google Sheets (ilustrada abaixo), onde na primeira linha listamos os nomes dos vértices, pulando a primeira coluna. Depois adicionamos uma linha para cada vértice na matriz e fornecemos a informação adequada de vizinhança entre os vértices, deixando a célula em branco quando dois vértices não forem adjacentes. Note que, como estamos importando um grafo não-direcionado, é necessário preencher apenas um dos triângulos da planilha, visto que a adjacência entre os vértices **a** e **b** é igual a adjacência entre os vértices **b** e **a**. Neste exemplo, adicionamos o 1 tanto no elemento da matriz entre a e b como entre b e a. Mas apenas uma aresta será criada. Para obter o csv, é só realizar o download no formato csv com vírgula.

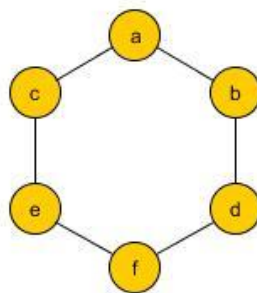


The screenshot shows a Google Sheet titled "5-3regular" with a menu bar (Arquivo, Editar, Ver, Inserir, Formatar, Dados, Ferramentas) and a toolbar. The spreadsheet has columns A through I and rows 1 through 6. The data is as follows:

	A	B	C	D	E	F	G	H	I
1		a	b	c	d	e			
2	a		1	1	1				
3	b	1		1		1			
4	c				1	1			
5	d					1			
6	e								

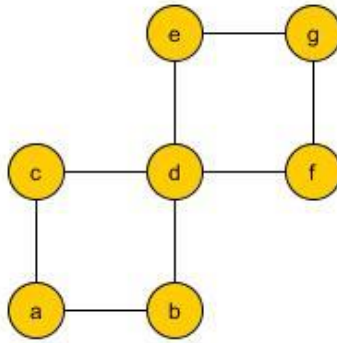
TAREFA 04: Como exercício, crie no Google Sheets um arquivo no formato CSV para representar o grafo abaixo.

Se usando o repl.it, edite a classe **Tarefa04.java**.



Importando Grafos no formato GML

Para importar um grafo no formato GML, o procedimento é similar. Vamos considerar o grafo abaixo que foi criado usando a ferramenta **yEd** e pode ser encontrado no pacote **graphs** com o nome `bp1.gml`.



Inicialmente, precisamos criar uma instância do grafo. Para vértices, utilizaremos a classe **util.DefaultVertex** e para arestas, utilizaremos a classe **DefaultEdge**.

```
Graph<DefaultVertex, DefaultEdge> graph = new SimpleGraph <>
    (VertexEdgeUtil.createDefaultVertexSupplier(),
     SupplierUtil.createDefaultEdgeSupplier(), false);
```

Para importar, utilizaremos o método *importDefaultGraphGML* da classe **util.ImportUtil**:

```
ImportUtil.importDefaultGraphGML(graph, "./src/main/java/graphs/bp1.gml");
```

Para visualizar o grafo que foi importado no console, podemos usar os métodos de inspeção da interface **Graph** como fizemos anteriormente ou usar o método *printGraph* da classe **util.PrintUtil**.

```
PrintUtil.printGraph(graph);
```

Este código pode ser encontrado no arquivo **Aula03ImportGraphGML** no pacote **classexamples**.

Para importar um pseudografo, seguimos um procedimento semelhante, mas criando uma instância de **Pseudograph** e usamos o método *importGraphGML*. A classe **Aula03ImportPseudoGraphGML** do pacote **classexamples** apresenta um exemplo.

TAREFA 05: Construa um programa que importe o grafo **cubo.gml**. O que teremos que fazer diferente da solução acima para que os *labels* das arestas sejam importados também? Dica: Use o método *importGraphGML* ao invés *importDefaultGraphGML*.

Se usando o repl.it, edite a classe **Tarefa05.java**.