ISSN 2177-9139

GRAFOS E SAGE

Monique Rubenich Nascimento - monique.rn@gmail.com
Universidade Federal de Santa Maria, Campus Sede, Av. Roraima nº 1000, Bairro Camobi, CEP: 97105-900, Santa Maria, RS, Brasil
Vinicius Schmidt Monego - viniciussm@rocketmail.com
Universidade Federal de Santa Maria, Campus Sede, Av. Roraima nº 1000, Bairro Camobi, CEP: 97105-900, Santa Maria, RS, Brasil
Alice de Jesus Kozakevicius - alicek@ufsm.br
Universidade Federal de Santa Maria, Campus Sede, Av. Roraima nº 1000, Bairro Camobi, CEP: 97105-900, Santa Maria, RS, Brasil

Resumo. *Neste mini-curso é apresentada uma ferramenta computacional gratuita e on-line, denominada SAGE, como sendo uma alternativa para o ensino-aprendizagem da teoria básica de GRAFOS. Este material pode ser explorado, tanto dentro quanto fora de sala de aula, uma vez que o aplicativo está on-line e não apresenta nenhuma dificuldade de instalação ou utilização. Além disso, o material apresentado explora todos os tópicos fundamentais da teoria de GRAFOS vistos em textos clássicos e que são utilizados como referência da disciplina de Matemática Discreta, dada em cursos de graduação em matemática e computação.*

Palavras chave. Grafos, Aplicativo online, SAGE

1 INTRODUÇÃO

A disciplina de Matemática Discreta, que faz parte de muitos currículos de graduação, incluindo Licenciatura e Bacharelado em Matemática, Ciência da Computação e Sistemas de Informação, traz em sua súmula uma introdução à Teoria de Grafos. Em cursos tecnológicos, esse assunto ainda é explorado de forma computacional. Já no contexto dos cursos de Matemática, a teoria de Grafos quase nunca é revisitada em outras disciplinas de graduação e seus aspectos computacionais praticamente não são abordados ou são omitidos.

Esta proposta de minicurso tem como objetivo principal apresentar uma alternativa computacional e divulgá-la, especialmente no contexto dos cursos de Licenciatura e Bacharelado em Matemática, como sendo uma ferramenta eficiente para o ensino e aprendizagem da teoria de Grafos. Além disso, como consequência, tem-se também como meta promover uma familiarização dos alunos com ferramentas computacionais que permitam explorar a teoria de Grafos de forma mais ampla, dando espaço a aspectos que envolvam aplicações e modelagem.

Existem muitas opções de pacotes computacionais para explorar além de Grafos, outros tópicos das mais variadas disciplinas da graduação. No entanto, neste minicurso opta-se pelo Sage. O Sage é um Sistema de Álgebra Computacional (do inglês Computer Algebra System-CAS) de código livre construído por muitos pacotes livres distintos unidos em uma interface Python. Sua missão é ser uma alternativa livre aos CAS proprietários tradicionais, como MATLAB ou Mathematica. Além disso, pelo fato de poder ser utilizado de forma on-line, dis-

pensando toda a parte de instalação, o Sage acaba sendo muito mais amigável e viável de ser utilizado nos mais diferentes ambientes.

Neste minicurso serão utilizados recursos do Sage que permitam a definição, visualização e manipulação de grafos. O material apresentado nesta proposta ainda será espandido em um tutorial a ser entregue aos participantes do minicurso durante o EREMATSUL 2014 e os tópicos abordados estão baseados nos conceitos e definições do livro Introdução à Análise Combinatória do autor J. Plínio O. Santos (1995), capítulo 8.

A seguir apresentam-se os passos principais para se trabalhar nesse ambiente computacional, uma vez que o aplicativo será utilizado "na nuvem", dispensando a necessidade de instalá-lo. Logo após, os principais conceitos e definições da teoria de Grafos serão explorados via Sage. As instruçõeas necessárias para a geração dos grafos e sua manipulação serão apresentadas nesta proposta de forma sucinta e tratadas de forma mais detalhada ao longo do minicurso proposto.

2 CADASTRO NO SAGEMATHCLOUD

1. A forma mais fácil de se usar o Sage é se cadastrando em https://cloud.sagemath.com
2. Cadastro do usuário:
   (a) Se for novo usuário: clicar em “Create an account”
   (b) Se já estiver cadastrado, apenas entrar na página informando email e senha.

3 UTILIZAÇÃO DO SAGE

3.1 Criar novo projeto

Clicar na opção “New Project” e dar nome ao projeto na linha da opção correspondente (será aberta uma janelinha para estas informações). Automaticamente o projeto criado aparecerá na sua lista de projetos existentes.

Abrir o projeto e clicar em “New” para criar um novo arquivo ou diretório. Clicar em “Sage Worksheet” para abrir uma planilha (espaço de trabalho).

É possível criar outro tipo de arquivo, como um arquivo LaTeXna opção “LaTeX Document” ou um arquivo de texto na opção “File”. Para criar outro tipo de arquivo dentro de um mesmo projeto, é só clicar em “New” e escolher a opção desejada (Sage Worksheet, IPython Notebook , File, Folder, From Internet, LaTeX Document , Terminal, Task List).

3.2 Trabalhar em projeto já existente

Para abrir o projeto é preciso clicar em “Projects” e clicar no projeto que deseja abrir. Aparecerão todos os arquivos e diretórios criados dentro do projeto. É só clicar na opção que desejar abrir.

Uma planilha no Sage é composta por várias células onde se escrevem códigos. Pode-se pensar nessas células como sendo linhas com uma ou mais instruções. Para avaliar cada célula individualmente, é necessária a combinação de teclas $< shift > + < enter >$ no final de cada

instrução. Isso fará com que o resultado da célula (instrução) apareça em uma janela. Este comando será considerado sempre que se desejar visualizar os resultados das operações em questão.

4 GRAFOS COM O SAGE

Depois de criar o projeto e a planilha, pode-se começar a construir e manipular grafos. Antes de sua definição ser apresentada, exploram-se alguns exemplos de grafos pré-definidos, para motivar os conceitos.
Para obter-se a lista disponível dos exemplos de grafos pré-definidos, digita-se:
graphs. $< tab >$
Assim, logo após teclar $< tab >$, um menu com os diferentes exemplos aparecerá. Basta selecionar com o mouse alguma das opções. A primeira delas é **Balaban10Cage**. Para chamar o método que define este grafo, deve-se digitar parêntesis aberto e fechado após o nome escolhido, como ilustrado a seguir:

```
graphs.Balaban10Cage()
```

Dentro dos parêntesis são colocados os argumentos do método. Neste caso, estamos chamando um método sem argumentos.

4.1 Visualização de Grafos

Para visualizar o grafo recém definido, utiliza-se o método plot(), da seguinte forma:

```
graphs.Balaban10Cage().plot()
```

Observação: Se no final desta célula não for dada a instrução $< shift > + < enter >$, o gráfico associado ao grafo do exemplo escolhido não será apresentado na janela.

Por conveniência, pode-se atribuir o grafo a uma variável:

```
a = graphs.Balaban10Cage()
```

sendo 'a' uma variável arbitrária que a partir desse momento recebe um objeto do tipo grafo.

Assim, pode-se visualizar o grafo utilizando a variável a associada ao método plot() :

`a.plot()`$< shift > + < enter >$

Para se obter uma representação 3D do grafo contido na variável a, utiliza-se:

`a.plot3d()`$< shift > + < enter >$

É possível aumentar ou diminuir o tamanho do grafo, considerando-se a barra de rolagem do mouse. Para que o grafo seja rotacionado, basta segurar o mouse sobre a figura e movimentá-lo.

4.2 Definição de um Grafo qualquer

Definição 1. *Um grafo $G = (V, A)$ é constituído por um conjunto finito não vazio $V \subset \mathbb{N}$ de vértices (ou nós) e um conjunto A de arestas (ou arcos), contendo as ligações entre estes vértices.*

Graficamente, cada vértice é representado no Sage por um círculo e cada aresta é representada por uma linha ligando estes círculos, como visto nos exemplos dados na seção anterior.

Cada aresta, denotada por $\{i, j\}$, é um par não-ordenado de vértices distintos. Se uma aresta corresponde ao par de vértices $\{i, j\}$ dizemos que i e j são as extremidades desta aresta.

Define-se computacionalmente um grafo $G = (V,A)$ a partir dos seus conjuntos de vétices V e arestas A, associando estes conjuntos a uma variável d:

`d = { 0: [3], 1: [2], 2:[3], 3:[4] }` sendo $\{0, 1, 2, 3, 4\}$ o conjunto V dos vértices. A notação 0: [3] significa que o vértice 0 tem uma aresta ligada em 3, 1: [2] significa que o vértice 1 tem uma aresta ligada no vértice 2, e assim por diante. No Python, a estrutura dada em d é conhecida como dicionário, que é uma lista associativa. A cada palavra-chave, que são os vértices individuais, associa-se uma lista de outros vértices aos quais este será ligado.

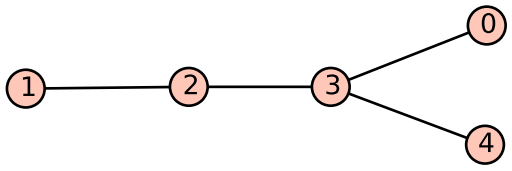
Agora para se construir o grafo a partir do dicionário definido em d, usa-se o comando:

`G = Graph(d)`

Exemplo 1

```
d = {0: [3], 1: [2], 2:[3], 3:[4]} # Definição do conjunto
                                   # de vértices e arestas
G = Graph(d) # Construção do grafo
G.plot() # Visualização do grafo
# <shift>+<enter> para aparecer o resultado na janela gráfica
```

Será construído o grafo 'G' apresentado abaixo:



Também pode-se definir o dicionário anterior da seguinte forma:

```
e = { 2 : [1], 3: [0,2,4] }
```

que produzirá o mesmo grafo definido no exemplo anterior, no entanto agora o dicionário e contém todas as ligações do vértice 3 em uma única lista.

Uma terceira forma de se criar o mesmo grafo é a partir de um grafo vazio e adicionar os vértices e arestas por partes. Esse processo é ilustrado na Figura **??**. A Figura **??**(a) apresenta a inserção da primeira aresta, incidente aos vértices 1 e 2. Assim, pode-se considerar o Grafo G, representado neste estágio, como sendo composto por 4 subgrafos, cada um desconexo dos demais. Conforme mais arestas são inseridas, o número de subgrafos desconexos vai diminuindo, até que apenas uma única componente conexa seja obtida no final do processo.

Para se criar um grafo vazio, usa-se:

`G=Graph()`

Abaixo reproduz-se o grafo definido acima no Exemplo 1, inserindo as arestas uma a uma.

```
G = Graph()
G.add_vertices([0,1,2,3,4])
```

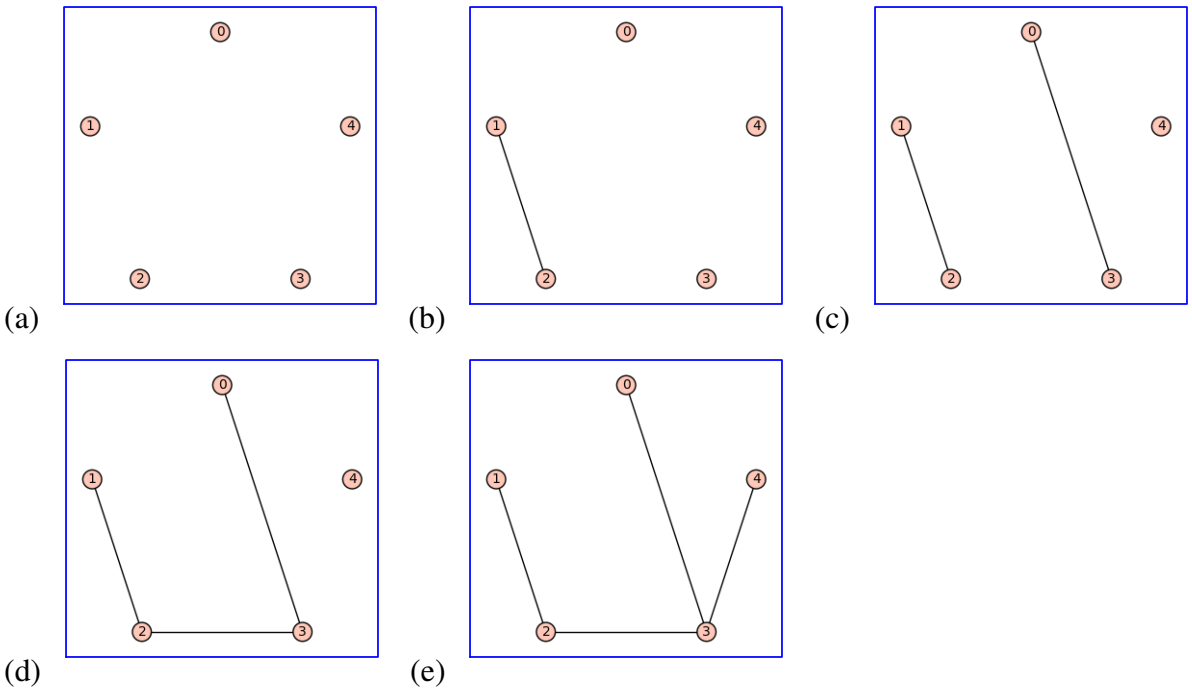


Figura 1: Grafo G: inserção gradual de arestas. (a) 5 componentes conexas, dadas apenas pelos vértices iniciais do grafo;(b) 4 componentes conexas;(c)3 componentes;(d) 2 componentes e (e) uma única componente conexa.

```
G.plot(layout='circular',graph_border=True)
G.add_edge((2,1))
G.plot(layout='circular',graph_border=True)
G.add_edge((3,0))
G.plot(layout='circular',graph_border=True)
G.add_edge((3,2))
G.plot(layout='circular',graph_border=True)
G.add_edge((3,4))
G.plot(layout='circular',graph_border=True)
```

Observação 1: De acordo com a Definição **??** da página **??**, o conjunto de vértices de G não poderia ser um conjunto vazio. No entanto, computacionalmente, o que se cria através da instrução G=Graph(), quando um grafo vazio é definido, é uma estrutura pronta para receber os conjuntos de vértices e arestas. Assim, podemos inserir novos vértices e arestas dentro desta estrutura já pronta.

Observação 2: O grafo final, apresentado na Figura **??**(e), é idêntico ao grafo obtido no Exemplo 1, isto é, tanto o conjunto de vértices quanto o conjunto de arestas são os mesmos. No entanto suas representações gráficas diferem em relação à posição na qual os vértices foram dispostos. Isso ocorre devido ao fato de um mesmo grafo poder ter várias representações planares.

Observação 3: No Sage, a cada vez que desenhamos um grafo, seus vértices são apresenta-

dos em um espaço livre e dispostos de acordo com a escolha feita automaticamente pelo método plot. Uma forma de fixar essas posições é através da opção layout='circular', que dispõe os vértices em forma circular dentro de uma região quadrada. Essa opção foi usada para uniformizar os 5 grafos da Figura **??**.

Definição 2. *Uma aresta $\{i, j\}$ é incidente nos vértices i e j. Analogamente, dizemos que um vértice é incidente nas arestas às quais está associado.*

No exemplo 1, o vértice 1 é incidente na aresta $\{1,2\}$. A aresta $\{2,3\}$ é incidente nos vértices 2 e 3. Quando um vértice não está associado a nenhum outro vértice é chamado de nó isolado, logo não exite aresta incidente a esse vértice.

Definição 3. *Duas arestas incidentes num mesmo vértice são chamadas adjacentes e dois vértices incidentes numa mesma aresta também são adjacentes.*

No exemplo 1, os vértices 2 e 3 são adjacentes e as arestas 1,2 e 2,3 são adjacentes.

Definição 4. *O grau de um vértice é o número de arestas incidentes neste vértice.*

Pode-se pensar no grau de um vértice como sendo a função

$$\begin{array}{rcl} gr : V & \to & \mathbb{N} \\ i & & gr(i) = n_i \end{array}$$

No grafo G do Exemplo 1, o grau do vértice 0 é 1 e o grau do vértice 3 é 3.

Teorema 1. *A soma dos vértices de um grafo é igual ao dobro do número de arestas.*

4.3 Matriz de Adjacência e Matriz de Incidência

Dado um grafo G, as ligações entre seus vértices (arestas) podem ser representadas por duas matrizes: a matriz de adjacência vértice $\times$ vértice e a matriz de incidência vértice $\times$ aresta.

Definição 5. *A matriz de adjacência tem linhas e colunas associadas aos vértices. O elemento na linha i e coluna j é o número de arestas que têm os vértices i e j como suas extremidades.*

Definição 6. *Na matriz de incidência as linhas estão associadas aos vértices e as colunas, às arestas. O elemento na linha i e coluna j é 1 se o vértice i é incidente na aresta j e 0, caso contrário.*

O comando para mostrar a matriz de adjacência é:

```
G.adjacency_matrix()
```

A matriz de adjacência do Exemplo 1 é:

$$Adj = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Tanto as linhas quanto as colunas da matriz *Adj* representam os vértices do grafo G. A primeira linha está associada ao primeiro vértice, neste exemplo indexado por 0. Esta linha indica todas as conexões existentes entre o vértice 0 e os demais vértices do grafo. Dessa forma, cada uma das demais linhas apresentará as ligações do vértice associado à linha com os demais. A matriz de adjacência é uma matriz simétrica, cujos elementos da diagonal são todos nulos, já que em um grafo não são permitidas ligações de um vértice com ele mesmo.

A matriz de incidência é obtida pelo comando:

```
G.incidence_matrix()
```

A matriz de incidência do Exemplo 1 é:

$$Inc = \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 1 & 0 & 1 & -1 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Cada linha da matriz *Inc* representa um vértice e cada coluna representa uma aresta. A primeira linha está associada ao vértice 0, indicando por 1 na coluna j se o vértice 0 é incidente à aresta j. Naturalmente, as demais linhas são construídas seguindo o mesmo critério. Esta matriz não é necessariamente simétrica.

Observação 4: No Sage a matriz de incidência *Inc* de um grafo já traz em sua definição padrão o conceito de orientação. Para tal, a matriz apresenta o valor -1 associado ao vértice considerado como origem da aresta. Para indicar a extremidade final da aresta, o valor 1 é considerado.

4.4 Definição de G via Matrizes

Um grafo pode ser definido no Sage também através de qualquer uma de suas matrizes (incidência ou adjacência). Dentro das atividades do minicurso, serão abordados exemplos nos quais as matrizes são informadas inicialmente e essas informações são então associadas aos Grafos.

4.5 "Caminhando" nos Grafos

Nesta seção são dadas as definições de passeio, caminho, circuito e ciclo, que correspondem a conceitos importantes da teoria com relevância para as aplicações.

Definição 7. *Um passeio entre os vértices i e j de um grafo é uma sequência alternante de vértices e arestas começando em um dos vértices i, j e terminando no outro, tal que cada aresta é incidente aos vértices que a cercam na sequência.*

Definição 8. *Um caminho é um passeio que não contém nós repetidos.*

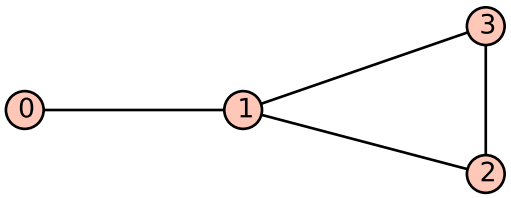
Definição 9. *Um circuito é um um passeio fechado, isto é, entre dois nós idênticos.*

Definição 10. *Um ciclo é um caminho fechado, isto é, um passeio que contém exatamente dois vértices iguais, o primeiro e o último.*

Os elementos apresentados nestas definições são reconhecidos via exemplos produzidos com comandos do Sage.

Para adicionar um caminho, utilizamos o método add_path

```
G = Graph() # Cria um grafo vazio
G.add_vertices([0,1,2,3]) # Adiciona os vértices de 0 a 3,
                          # sem conexões
G.add_path([0,1,3,2,1]) # Adiciona um caminho {0,1,3,2,1}
G.plot()
```



Todos os caminhos disponíveis de um vértice inicial a um vértice final podem ser vistos usando-se o método all_paths, passando-se o vértice de saída e o vértice de chegada como parâmetros. Por exemplo, para o grafo logo acima, o comando

```
G.all_paths(0,3)
```

dá como saída o seguinte resultado:

$[[0,1,2,3],[0,1,3]]$

Isso significa que para sair do vértice 0 e chegar no vértice 3, pode-se percorrer tanto o caminho {0,1,2,3} quanto o caminho {0,1,3}, e estes são os únicos caminhos possíveis para este grafo. Para adicionar um ciclo, utiliza-se o método add_cycle.

```
G = Graph() # Cria um grafo vazio
G.add_vertices([0,1,2,3,4,5]) # Adiciona os vértices de 0 a 5,
                              # sem conexões
G.add_cycle([0,1,2]) # Adiciona um ciclo {0,1,2}
G.add_cycle([3,4,5]) # Adiciona um ciclo {3,4,5}
G.plot()
```

imgs_plot/grafo.png

Com os vértices de 0 a 5 pode-se criar um único ciclo da seguinte maneira:

```
G.add_cycle([0,1,2,3,4,5]) # Adiciona um ciclo {0,1,2,3,4,5}
```

4.6 Digrafo

Definição 11. *Um digrafo $G = (V,A)$ é constituído por um conjunto finito de vértices V e um conjunto A de arestas direcionadas, de tal forma que exista uma correspondência biunívoca (1-1) entre os elementos de A e um subconjunto do produto cartesiano $\mathbb{N} \times \mathbb{N}$ que não contenha os pares (i,i). Cada aresta direcionada corresponde a um par ordenado de vértices.*

Neste caso o conceito de incidência é modificado e a aresta correspondente ao par $\{i, j\}$ é incidente do vértice i e incidente para o vértice j, e a direção da aresta é indicada graficamente como uma seta no vértice j.

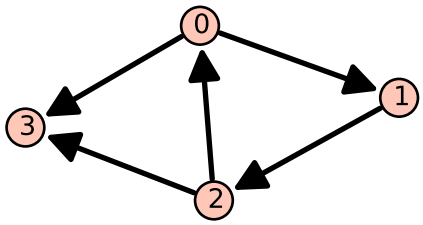
O grau de entrada de um vértice é o número de arestas que entram no vértice, ou seja, são incidentes para o vértice e o grau de saída é definido de maneira análoga.

Em um digrafo são dados novos conceitos dependentes da orientação: passeio orientado e caminho orientado de um vértice i para um vértice j. A diferença agora é que há uma orientação para o passeio ou caminho, e com isso cada aresta na sequência é incidente do vértice que o precede para o vértice que o sucede na sequência.

Para construir um digrafo no Sage usamos o construtor de classe `DiGraph()`

Exemplo :

```
h = {0: [1,3], 1: [2], 2: [0,3]} # vértices e arestas
                                 # direcionadas
R = DiGraph(h) # Construção do grafo
R.plot() # Vizualização do grafo
```



O Sage possui exemplos de digrafos pré-definidos. A lista dos exemplos pré-definidos disponíveis é obtida digitando-se:

digraphs.$< tab >$

Assim, analogamente ao que foi feito para a obtenção de exemplos de grafos, logo após teclar $< tab >$, um menu com os diferentes exemplos aparecerá. Basta selecionar com o mouse alguma das opções. A segunda da lista fornecida é Circuit. Para chamar este método que gerará o exemplo, deve-se colocar como argumento o número de vértices (no exemplo, 3) que o digrafo deve ter, como ilustrado a seguir:

```
G=digraphs.Circuit(3)
G.plot()
```

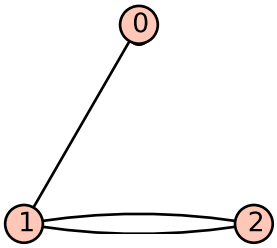
4.7 Multigrafo

Outro elemento básico e relevante na teoria de grafos que será explorado no minicurso, são os multigrafos, que são grafos nos quais são permitidos dois ou mais arcos associados a um mesmo par de vértices. Quando os dois vértices do par não-ordenado, que definem a aresta, não forem distintos, esta aresta será dita um laço, como mostra o Exemplo a seguir. Para se criar um multigrafo no Sage, é necessário passar uma opção que permita que o grafo tenha mais de uma aresta associada a um mesmo par de vértices:

```
a = Graph(multiedges=True)
```

Exemplo :

```
d = {0: [0,1], 1: [2,2]}
J = Graph(multiedges=True)
J = Graph(d)
J.plot(layout='circular')
```



4.7.1 Pontes de Königsberg

Problema resolvido por Euler em 1736 sobre a possibilidade de fazer um passeio pela cidade de Königsberg, começando e terminando no mesmo lugar, cruzando cada ponte exatamente uma vez. O problema foi transformado em um multigrafo, mostrado abaixo, onde cada margem e ilha representa um vértice e cada ponte uma aresta.

```
d = {0: [1,2,3], 1: [2,2], 2: [3,3]}
J = Graph(multiedges=True)
J = Graph(d)
J.plot()
```

A solução para o problema seria achar um circuito que percorra cada arco exatamente uma vez. Os (multi)grafos para os quais isto é possível são chamados de eulerianos.

Definição 12. *O (multi)grafo conexo $G = (V,A)$ é euleriano se, e somente se, os graus de todos os nós de G são pares.*

Portanto o problema não apresenta solução já que o grau de seus nós não são todos pares, 0, 1 e 3 têm grau 3 e 2 tem grau 5.

4.8 Planaridade

Definição 13. *Um grafo é planar se é possível desenhá-lo no plano de modo que as linhas correspondentes aos arcos não se cruzem.*

O desenho é uma realização gráfica planar do grafo, ou, simplismente, realização planar. No exemplo 5, no desenho à esquerda o grafo parece não satisfazer a condição, porém podemos desenhá-lo de forma a ver que se trata de um grafo planar, como é mostrado no desenho à direita.

Para sabermos se o grafo é planar usamos o método:

$G.is_planar()$

Exemplo 5:

```
d = {0: [1,2,3], 1: [0,2,3], 2: [0,1,3], 3: [0,1,2]}
G = Graph(d)
G.plot(title="Representacao nao planar")
G.plot(layout='planar', title="Representacao planar")
```

Usando o método $G.is_planar()$ para o grafo do Exemplo 5 teremos como resposta "True", ou seja, o grafo é planar.

Veremos agora outro exemplo, de um grafo bipartido completo.

Exemplo 6:

```
d = {0: [3,4,5], 1: [3,4,5], 2: [3,4,5]}
H = Graph(multiedges=True)
H = Graph(d)
H.plot()
```

Definição 14. *Grafos bipartidos são aqueles nos quais o conjunto de vértices V pode ser particionado em dois subconjuntos de tal maneira que vértices pertencentes a um mesmo subconjunto não são adjacentes.*

Definição 15. *O grafo bipartido completo contém todos as arestas possíveis, ou seja, se V_1 e V_2 são subconjuntos de vértices do conjunto V, então cada vértice de V_1 é adjacente a todos os vértices de V_2. A notação para este tipo de grafo é $K_{n,l}$, onde n é a cardinalidade de V_1 e l é a cardinalidade de V_2*

Usando o método $G.is_planar()$ para o grafo do Exemplo 6 teremos como resposta "False". O grafo não é planar então não podemos representá-lo graficamente em um plano sem que suas aresta se cruzem.

4.9 Isomorfismo

Definição 16. *Dois grafos $G=(V_1,A_1)$ e $H=(V_2,A_2)$ são isomorfos se existe uma relação 1-1 entre os vértices V_1 e V_1 que preserva adjacência, ou seja, se i e j em V_1 são adjacentes em G, então os nós correspondentes em V_2 pela relação também são adjacentes em H e vice versa.*

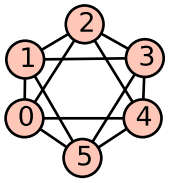
Grafos isomorfos são grafos que não são iguais, mas apresentam as mesmas propriedades estruturais (número de vértices e arestas, grau dos vértices, conectividade e planaridade).

No Sage, para sabermos se dois grafos G e H são isomorfos usamos o comando:

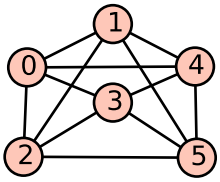
$G.is_isomorphic(H)$

Exemplo 7: Grafos isomorfos.

```
d = {0: [1,2,4,5], 1: [2,3,5], 2: [3,4], 3: [4,5], 4:[5]}
G = Graph(d)
G.plot()
```




```
d = {6: [7,8,9,10], 7: [8,10,11], 8: [9,11], 9: [10,11], 10:[11]}
H = Graph(d)
H.plot()
```



Usando o método $G.is_isomorphic(H)$ para os grafos do Exemplo 7 teremos como resposta "True", ou seja, os grafos são isomorfos.

4.10 Operações sobre Grafos

Durante o minicurso, serão exploradas operações sobre os grafos, os comandos correspondentes e detalhes específicos sobre a sintaxe desses comandos em Sage.

5 CONCLUSÃO

Esta proposta de minicurso, introduz alguns recursos computacionais disponíveis no Sage para apresentar conceitos iniciais da Teoria de Grafos via manipulação de exemplos de uma forma mais dinâmica. Desta forma pretende-se abrir um leque maior de possibilidades de se explorar essa teoria através de recursos computacionais, que facilitem e promovam uma maior disseminação da teoria de Grafos nos cursos de Licenciatura em Matemática, como sendo um tópico rico em desafios lógicos e grande potencial para serem aplicados a problemas realistas.

REFERÊNCIAS

SANTOS, J. P. O.; MELLO, M. P.; MURARI, I. T. C. **Introdução à Análise Combinatória**. Campinas: Editora da Unicamp, 1995.