Cruzeiro do Sul Virtual
Educação a Distância

VISUALIZAÇÃO DE REDES E GRAFOS

Prof. Ismar Frango

Visualização De Grafos e Redes

Em uma era em que as pessoas estão cada vez mais conectadas de várias maneiras, assim como também diversos objetos (no contexto do que chamamos de Internet das Coisas – *IoT – Internet of Things*), um tipo de informação vem se tornando cada vez mais comum (e necessário) para representar os fenômenos relacionados a essa "hiperconectividade": são os grafos e as redes (entre as quais destacam-se as redes sociais, como Facebook, Instagram e Twitter)

Conheça um pouco mais sobre IoT em:
https://pt.wikipedia.org/wiki/Internet_das_coisas

A Teoria dos Grafos – importante área da Ciência da Computação – e o campo da Ciência que estuda esse tipo de estrutura de dados. Entretanto, essa é uma área de conhecimento da Matemática Discreta, muito anterior aos primeiros computadores. A palavra "grafo" é um neologismo derivado de *graph* em inglês – que também pode indicar "gráfico" (embora, para esse sentido, há a palavra ***chart***).

Conheça um pouco mais sobre Teoria dos Grafos e suas Aplicações:
https://www.ime.usp.br/~yw/publications/books/TeoriaDosGrafos.pdf e
http://www.rc.unesp.br/tmelo/diss-polyanna.pdf

Veremos que grafos são conjuntos de nós (ou vértices) ligados por arestas (ou arcos). Diferente das árvores, não há qualquer restrição de ligação entre os nós em um grafo. Há muitas aplicações de grafos hoje em dia: sistemas geográficos (como o OpenStreetMap ou o Google Maps) usam grafos para a representação de rotas: a interseção de duas (ou mais) estradas é considerada um **vértice** e a estrada que liga dois vértices é considerada uma **aresta**. Esses conceitos básicos são usados por aplicações como o Waze ou Uber que usam um algoritmo para calcular o menor caminho entre dois vértices (ou seja, uma rota). Já em redes sociais, como o Facebook, os usuários são considerados os **vértices** e as amizades são representadas como **arestas** entre eles. A própria organização dos sites na Internet, em geral, pode ser representada por um grafo: as páginas podem ser consideradas os **vértices**; se há um link de uma página para outra, isso pode ser representado como uma **aresta**.

Conheça um pouco mais sobre esse importante matemático: https://pt.wikipedia.org/wiki/Leonhard_Euler

Um pouco de História

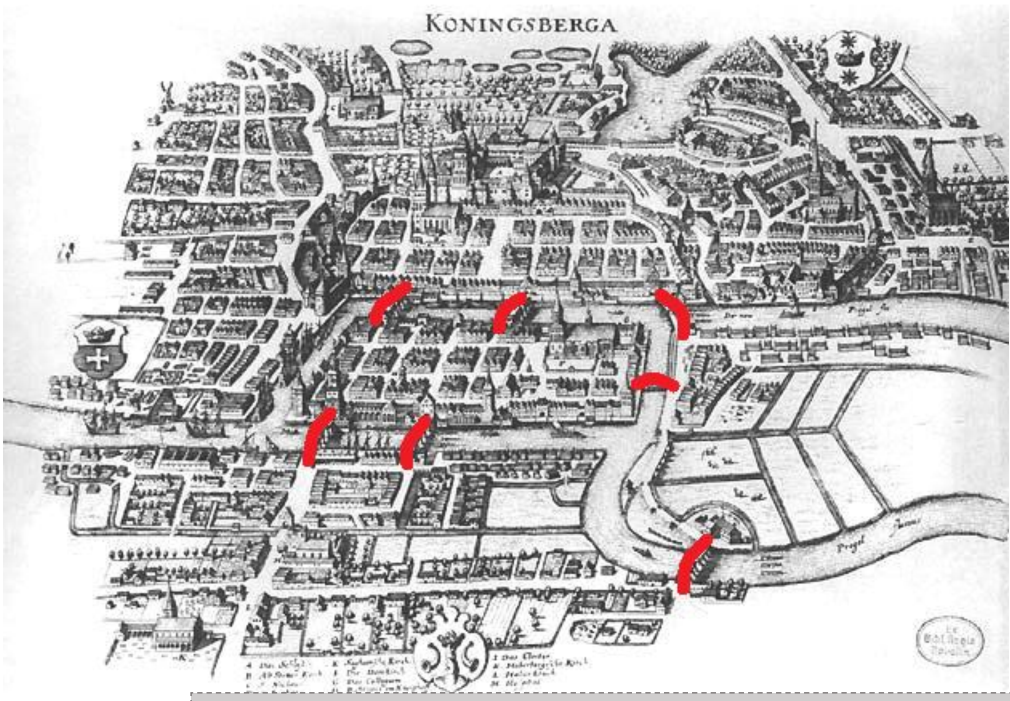
Ainda que o estudo das relações entre elementos possa ser mais antigo, o primeiro estudo formal na área é de 1736, quando Leonhard Euler, um matemático suíço, resolveu um problema conhecido como “As Pontes de Königsberg”.

O problema era assim formulado: a cidade, que é cortada pelo Rio Prególia, é composta por duas grandes ilhas que, juntas, formam um complexo que na época consistia sete pontes.

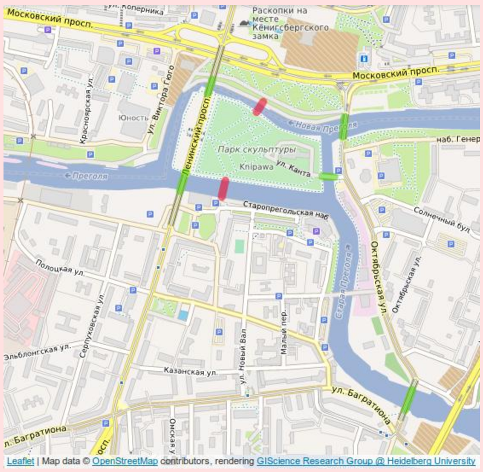
Havia uma lenda urbana de que seria possível atravessar todas as pontes sem repetir nenhuma. Euler provou matematicamente que não existia essa possibilidade.

A cidade de Königsberg, hoje chamada de Kaliningrado, chegou a ser capital e centro cultural e econômico da antiga Prússia. A partir a Segunda Guerra, se tornou um território soviético (hoje russo), ainda que esteja fora de seu território principal, fazendo fronteira com Polônia e Lituânia:

Esse mapa da época mostra a localização das pontes:

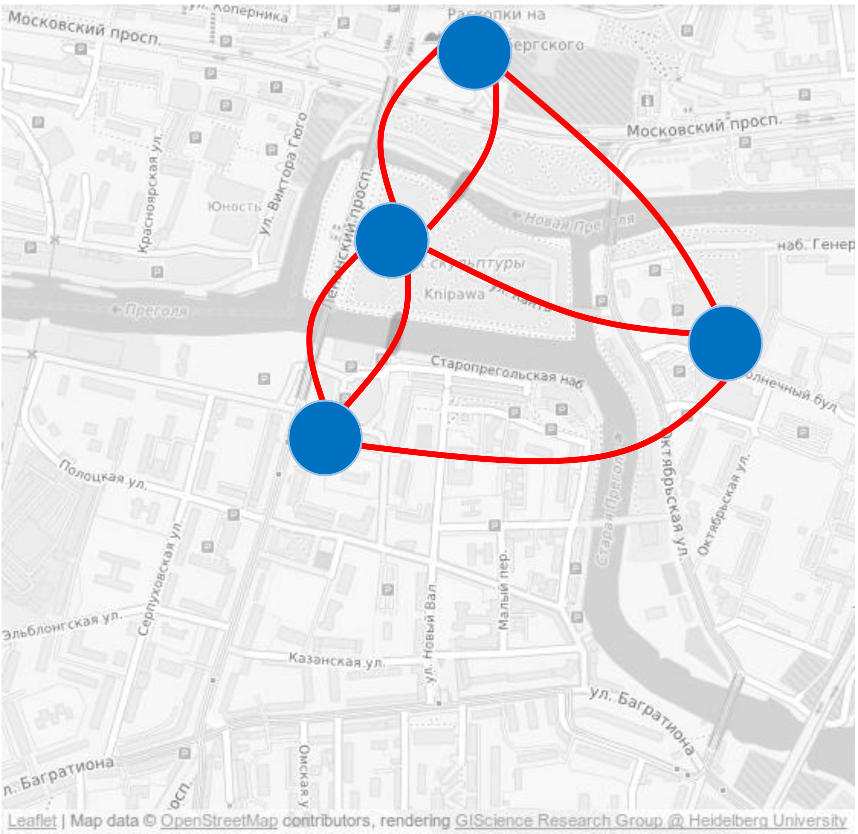


Esse é o mapa da região nos dias de hoje, em que restaram apenas cinco pontes (em verde). Duas foram destruídas em um bombardeio em 1944:



Fonte: https://pt.wikipedia.org/wiki/K%C3%B6nigsberg#/media/Ficheiro:Image-Koenigsberg,_Map_by_Merian-Erben_1652.jpg - Licença: CC-BY

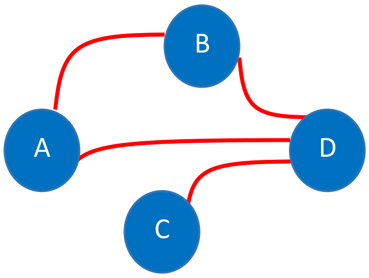
Como Euler resolveu esse problema? Na verdade, usando um raciocínio bastante simples: representou os **caminhos** em **linhas** (em **vermelho**) e as regiões do mapa em **círculos** (em **azul**). Vamos projetá-lo sobre o mapa de Kaliningrado hoje:



Euler fez então a primeira representação que temos de um grafo. E descobriu que só seria possível fazer o caminho inteiro passando uma única vez em cada ponte se houvesse exatamente zero ou dois pontos de onde saísse um número ímpar de caminhos – esse tipo de caminho é chamado hoje de *caminho Euleriano*.

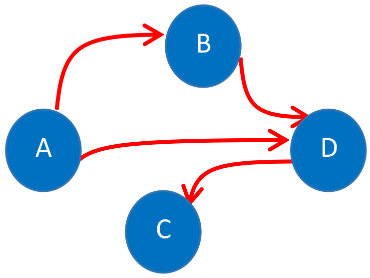
Definições

Como já mencionado, um grafo é um conjunto de nós e arestas que conectam esses nós. Os grafos podem ser **direcionados** (ou dirigidos – também conhecidos como dígrafos ou dígrafos) ou **não-direcionados** (ou não-dirigidos)



Este é um grafo **não-dirigido**. Significa que existe um caminho de A até B e vice-versa.

O mecanismo de amizades do Facebook usa um grafo não-dirigido: se A é amigo de B, B também é amigo de A.

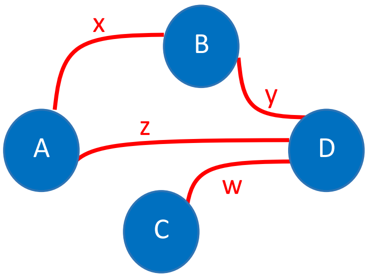


Este é um grafo **dirigido**. Significa que existe um caminho de A até B, mas não o contrário.

Seguir uma pessoa no Twitter usa a lógica dos grafos dirigidos: A segue B, mas B não segue A.

Em geral, os termos nós e arestas são usados para grafos não-dirigidos. Para grafos dirigidos, se usa mais os termos vértices e arcos. Entretanto, é comum encontrar referências (especialmente na Internet) com os termos usados em ambas as situações.

Outra definição importante é sobre grafos valorados:

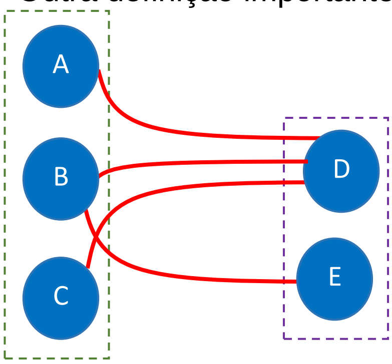


Um grafo **valorado** ou **ponderado** possui valores (pesos ou custos) em suas arestas. Pode ser dirigido ou não-dirigido.

Por exemplo, se A, B, C e D forem cidades, os valores x, y, w, z podem ser as distâncias diretas entre elas.

Seguindo a definição formal, uma **rede** é um **grafo dirigido valorado**. Porém, em redes sociais, essa definição é mais ampla, abarcando praticamente todo tipo de grafo.

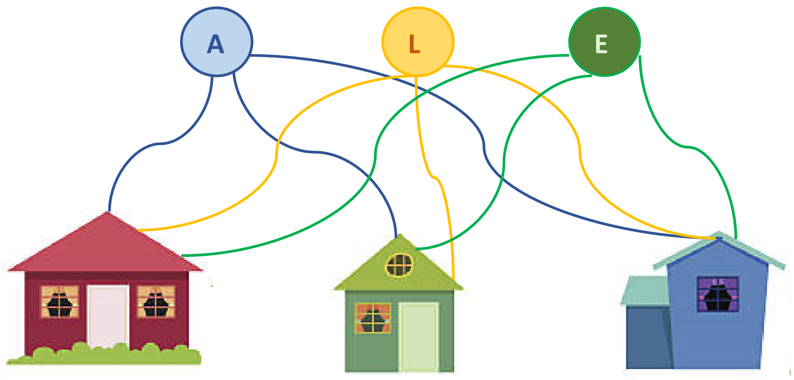
Outra definição importante é sobre grafos bipartidos:



Um grafo **bipartido** (ou **biclique**) consiste em um grafo em que os nós podem ser divididos em dois conjuntos. Os nós de um conjunto não têm arestas entre si.

Por exemplo, A, B e C podem ser prestadores de serviços de Internet; D e E podem representar os bairros que são atendidos pelos serviços.

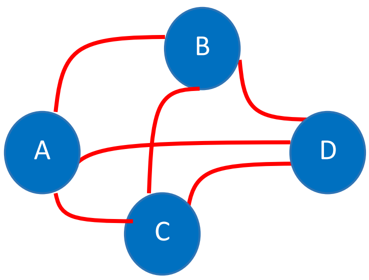
Um exemplo bastante popular de **grafo bipartido** está em um conhecido jogo matemático, em que três casas deveriam receber serviços de água (A), luz (L) e esgoto (E):



O desafio é encontrar uma disposição das arestas de maneira a que elas nunca se cruzem. Este problema não tem solução, pois se trata de um grafo não-planar.

Um grafo **planar** pode ser representado em duas dimensões (numa folha de papel, por exemplo) sem que as arestas se cruzem. Grafos **não-planares** não têm representação em duas dimensões sem que ao menos duas arestas se cruzem.

Uma outra definição importante é sobre o que é um grafo completo: se trata de um grafo em que todos os nós são conectados entre si.

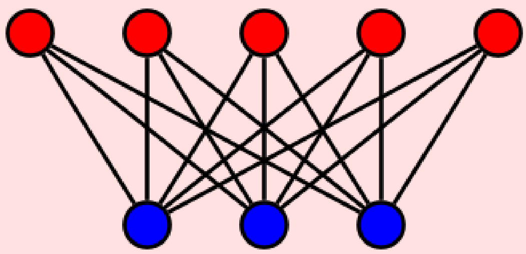


Este é um grafo **completo**, pois todos os nós estão diretamente conectados. Todo grafo completo com uma quantidade n de nós tem exatamente $\frac{n*(n-1)}{2}$ arestas.

No exemplo, com 4 nós, temos:

$\frac{4*3}{2} = 6$ arestas

Podemos ter também **grafos bipartidos completos**, como no exemplo abaixo:



Todo grafo **bipartido completo** com **n** nós de um lado e **m** nós de outro sempre terá m*n arestas.

No exemplo, o grafo ao lado tem 5 nós em um lado e 3 de outro, totalizando:
5*3 = 15 arestas

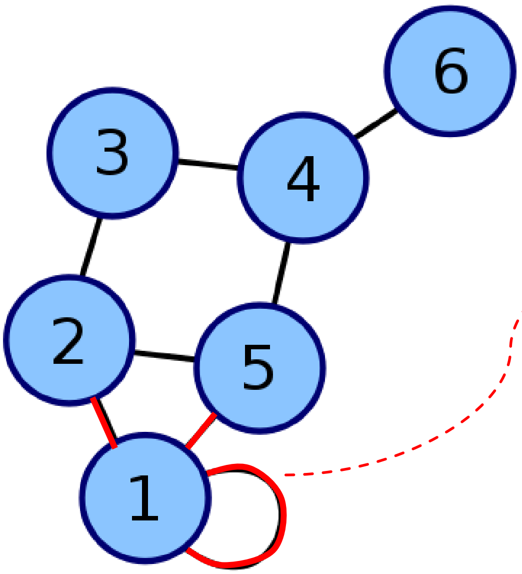
Representação de dados

Como identificar se a natureza de um dataset é adequada para ser representada como um grafo?

A resposta é bastante simples: todo conjunto de dados que puder ser representado por uma matriz de adjacência pode ser visualizado como um grafo.

Uma matriz de adjacência é em geral uma matriz quadrada (ou seja, com o mesmo número de linhas e de colunas). Cada nó do grafo é representado por uma linha e por uma coluna.

Vejamos o seguinte exemplo de um grafo não-dirigido e não-valorado:



$$A = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{bmatrix} \boxed{1 & 1 & 0 & 0 & 1 & 0} \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \end{array}$$

Cada valor 1 da matriz indica que há uma aresta entre os nós representados pela linha e pela coluna.
Por exemplo, aqui temos arestas:

- do nó 1 para ele mesmo
- do nó 1 para o 2
- do nó 1 para o 5

Para grafos não-dirigidos, a matriz de adjacência sempre será uma **matriz triangular**, ou seja, dividindo a matriz em duas partes a partir da diagonal principal, basta saber uma das metades dos valores da matriz, pois a outra metade será idêntica à primeira metade, espelhada.

$$A = \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

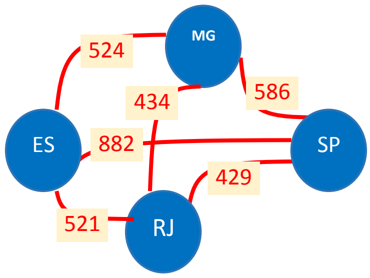
Uma forma alternativa de representar a matriz de adjacência, especialmente em grafos não-valorados, é por meio de uma **lista de adjacência**. Numa lista de adjacência, o primeiro item de cada linha mostra de onde partem os vértices e os demais itens indicam onde chegam (em grafos não-orientados, essa diferença não existe)
Neste exemplo, a **lista de adjacência** correspondente seria:

1 1 2 5
2 3 5
3 4
4 5 6

Quando o grafo é valorado, os valores são colocados na matriz de adjacência no lugar do valor “1”. Vejamos um exemplo prático, com dados reais contendo as distâncias rodoviárias entre as capitais dos quatro estados da Região Sudeste do Brasil, em km:

	ES	MG	RJ	SP
ES	0	524	521	882
MG	524	0	434	586
RJ	521	434	0	429
SP	882	586	429	0

Veja que basta representar esta parte da matriz de adjacência visto que se trata de um grafo não-dirigido (a distância para ir e voltar é a mesma)



Recomenda-se verificar bibliotecas alternativas, como **Bokeh**: https://docs.bokeh.org/en/latest/docs/user_guide/graph.html

É possível também gerar visualizações via Python com o software GraphViz:

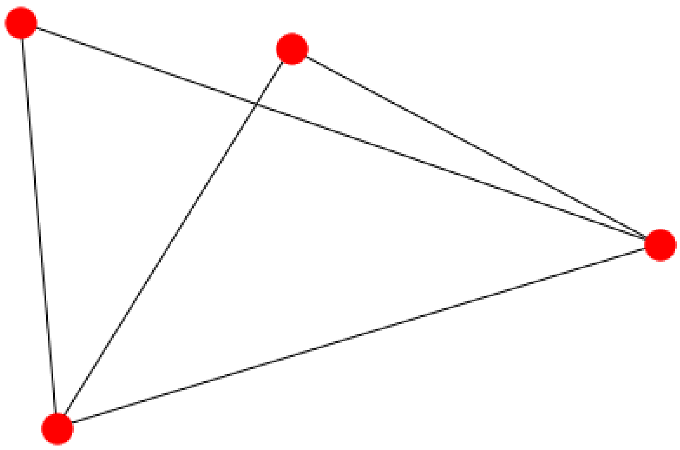
https://pypi.org/project/graphviz/

Implementação em Python

Um fato importante a mencionar é que, utilizando as bibliotecas básicas do Python (incluindo a Matplotlib e a Plotly), há métodos para a visualização de grafos básicos seguindo a representação node-link – os mesmos métodos podem ser adaptados para visualização de árvores . O exemplo a seguir utiliza a biblioteca Matplotlib (mais diretamente o pacote pyplot), assim como a biblioteca NetworkX.

NetworkX é uma biblioteca do Python para representação interna de grafos/redes. Conheça mais em: https://networkx.github.io/

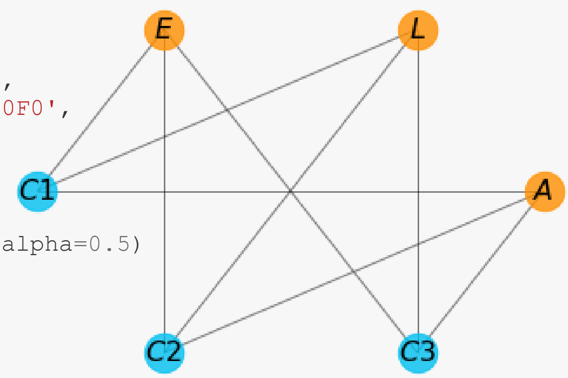
```python
import networkx as nx
import matplotlib.pyplot as plt
G=nx.Graph() #o método Graph cria um grafo não-orientado
G.add_nodes_from([1,2,3,4]) #definição dos nós
G.add_edges_from([(1,2),(1,3),(2,4),(3,4),(2,3)]) #definição das arestas
nx.draw_random(G) #desenha os nós em posições aleatórias
plt.show()
```



Implementando o exemplo do jogo água-luz-esgoto mencionado anteriormente, ainda usando apenas as bibliotecas **networkx** e **matplotlib** :

```python
import matplotlib.pyplot as plt
import networkx as nx

G=nx.complete_bipartite_graph(3,3) # grafo bipartido pré-definido
pos=nx.shell_layout(G) # layout pré-definido

# nós
nx.draw_networkx_nodes(G,pos,
                       nodelist=[0,1,2],
                       node_color='#FF9000',
                       node_size=500,
                       alpha=0.8)
nx.draw_networkx_nodes(G,pos,
                       nodelist=[3,4,5],
                       node_color='#00C0F0',
                       node_size=500,
                       alpha=0.8)

# arestas
nx.draw_networkx_edges(G,pos,width=1.0,alpha=0.5)

# colocando rótulos
labels={}
labels[0]=r'$A$'
labels[1]=r'$L$'
labels[2]=r'$E$'
labels[3]=r'$C1$'
labels[4]=r'$C2$'
labels[5]=r'$C3$'
nx.draw_networkx_labels(G,pos,labels,font_size=16)
plt.axis('off')
plt.show() # exibe o grafo
```

Veja mais sobre layouts: https://networkx.github.io/documentation/stable/_modules/networkx/drawing/layout.html



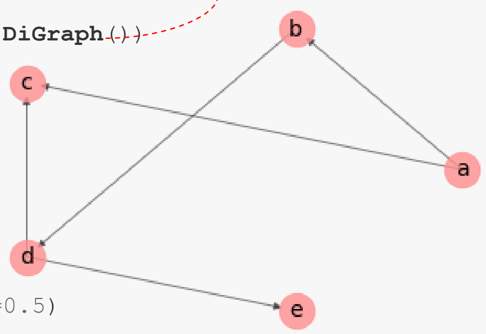
Este outro exemplo lê uma lista de adjacências e o representa como um grafo orientado (*digraph*):

Conteúdo do arquivo:

```
a b c
b d
d c e
```

Sem isso, seria criado um grafo não-orientado

```
import matplotlib.pyplot as plt
import networkx as nx

G=nx.read_adjlist("adj.txt", create_using=nx.DiGraph())
pos=nx.shell_layout(G) # layout pré-definido

# nós
nx.draw_networkx_nodes(G,pos,
                       node_color='#FF9090',
                       node_size=500,
                       alpha=0.8)

# arestas
nx.draw_networkx_edges(G,pos,width=1.0,alpha=0.5)

# colocando rótulos
labels=nx.draw_networkx_labels(G,pos,font_size=14)

plt.axis('off')
plt.show() # display
```

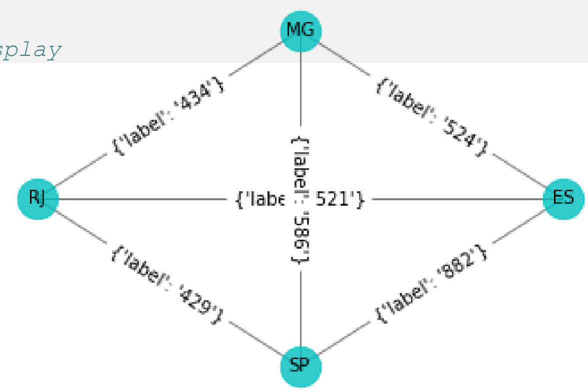


O seguinte exemplo implementa o exemplo do grafo valorado completo mostrado anteriormente (o das distâncias entre as capitais do Sudeste), a partir de um arquivo GML.

GML é um formato de arquivo de representação de grafos.
Conteúdo do arquivo:

```
graph
[
  node
  [
   id "ES"
   label "ES"
  ]
  node
  [
   id "MG"
   label "MG"
  ]
  node
  [
   id "RJ"
   label "RJ"
  ]
  node
  [
   id "SP"
   label "SP"
  ]
   edge
  [
   source "ES"
   target "MG"
   label "524"
  ]
  edge
  [
   source "ES"
   target "RJ"
   label "521"
  ]
  edge
  [
   source "ES"
   target "SP"
   label "882"
  ]
  edge
  [
   source "MG"
   target "RJ"
   label "434"
  ]
  edge
  [
   source "MG"
   target "SP"
   label "586"
  ]
  edge
  [
   source "RJ"
   target "SP"
   label "429"
  ]
]
```

```
import matplotlib.pyplot as plt
import networkx as nx

G=nx.read_gml("graph.gml")
pos=nx.shell_layout(G) # layouts pré-definidos

# nós
nx.draw_networkx_nodes(G,pos,
                       node_color='#00C0C0',
                       node_size=500,
                       alpha=0.8)

# arestas
nx.draw_networkx_edges(G,pos,width=1.0,alpha=0.5)

# colocando rótulos nos nós e arestas
labels=nx.draw_networkx_labels(G,pos,font_size=10)
edge_labels=nx.draw_networkx_edge_labels(G,pos,font_size=10)

plt.axis('off')
plt.show() # display
```

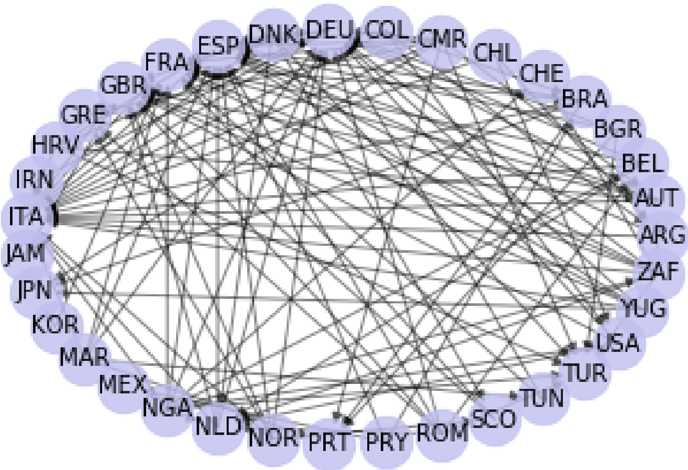


Este próximo exemplo carrega um arquivo Pajek e o processa em Python. O *dataset* utilizado no exemplo traz dados das seleções que jogaram na Copa do Mundo de 1998. É representado por um grafo dirigido em que os nós representam países; há arcos entre um país e outro se algum jogador de uma seleção de um país atua em outro país.

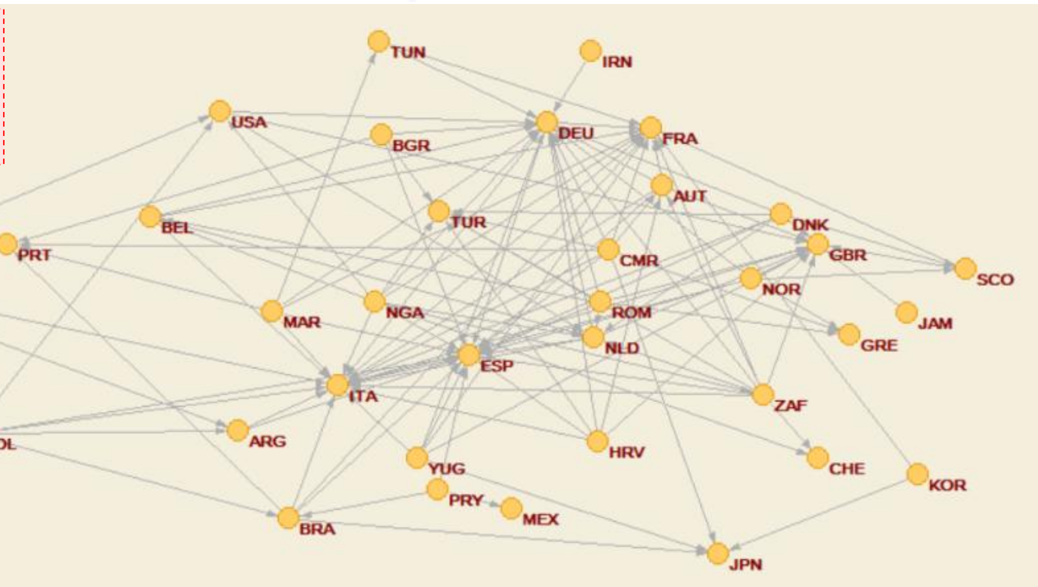
```python
import matplotlib.pyplot as plt
import networkx as nx

G=nx.read_pajek("football.net")
G1=nx.DiGraph(G)
pos=nx.circular_layout(G1) # layouts pré-definidos

# nós
nx.draw_networkx_nodes(G1,pos,
                       node_color='#C0C0F0',
                       node_size=500,
                       alpha=0.8)

# arestas
nx.draw_networkx_edges(G1,pos,width=1.0,alpha=0.5)

# colocando rótulos nos nós
labels=nx.draw_networkx_labels(G1,pos,font_size=10)

plt.axis('off')
plt.show() # display
```

Pajek é um dos primeiros softwares para análise de grandes redes. Para conhecê-lo, acesse: http://mrvar.fdv.uni-lj.si/pajek/

Baixe o *dataset* em: http://vlado.fmf.uni-lj.si/pub/networks/data/sport/football.net



O software Pajek possibilita a visualização dos grafos/redes. Esta visualização foi gerada no Pajek.

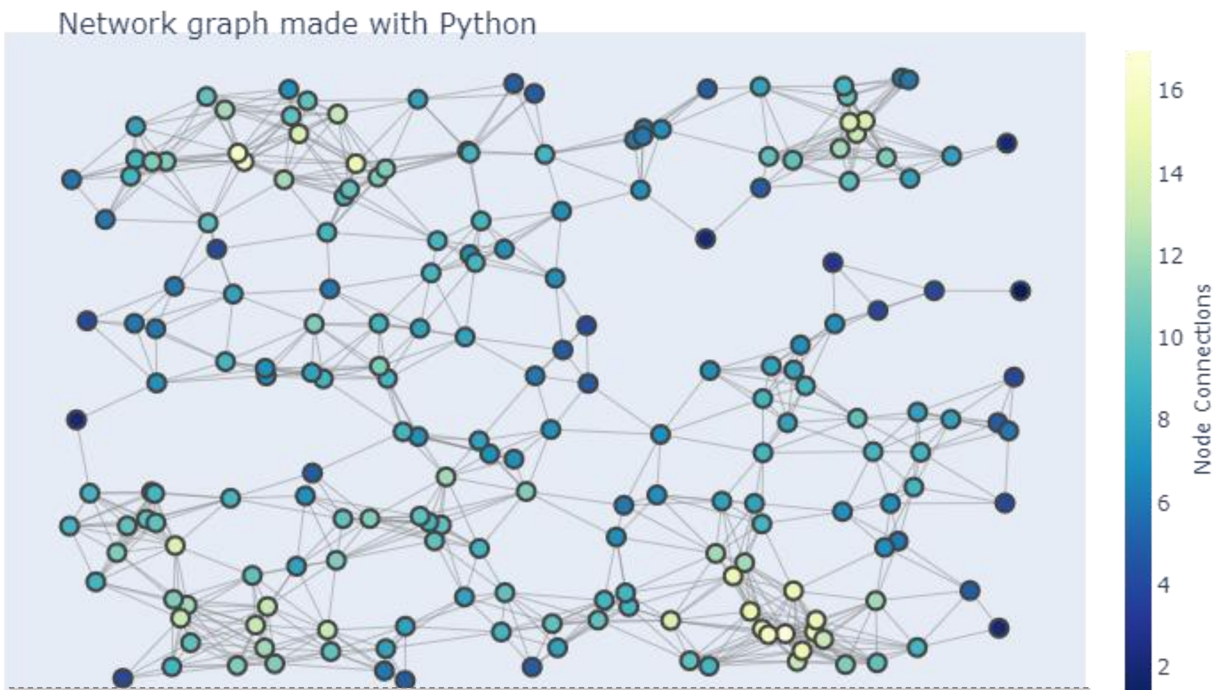


Este próximo exemplo já usa a biblioteca Plotly, junto com a NetworkX.

```
import plotly.graph_objects as go

import networkx as nx

G = nx.random_geometric_graph(200, 0.125)
edge_x = []
edge_y = []
for edge in G.edges():
    x0, y0 = G.nodes[edge[0]]['pos']
    x1, y1 = G.nodes[edge[1]]['pos']
    edge_x.append(x0)
    edge_x.append(x1)
    edge_x.append(None)
    edge_y.append(y0)
    edge_y.append(y1)
    edge_y.append(None)

edge_trace = go.Scatter(
    x=edge_x, y=edge_y,
    line=dict(width=0.5, color='#888'),
    hoverinfo='none',
    mode='lines')

node_x = []
node_y = []
for node in G.nodes():
    x, y = G.nodes[node]['pos']
    node_x.append(x)
    node_y.append(y)

node_trace = go.Scatter(
    x=node_x, y=node_y,
    mode='markers',
    hoverinfo='text',
    marker=dict(
        showscale=True,
        colorscale='YlGnBu',
        reversescale=True,
        color=[],
        size=10,
        colorbar=dict(
            thickness=15,
            title='Node Connections',
            xanchor='left',
            titleside='right'
        ),
        line_width=2))
node_adjacencies = []
node_text = []
for node, adjacencies in enumerate(G.adjacency()):
    node_adjacencies.append(len(adjacencies[1]))
    node_text.append('# of connections: '+str(len(adjacencies[1])))

node_trace.marker.color = node_adjacencies
node_trace.text = node_text
fig = go.Figure(data=[edge_trace, node_trace],
             layout=go.Layout(
                title='<br>Network graph made with Python',
                titlefont_size=16,
                showlegend=False,
                hovermode='closest',
                margin=dict(b=20,l=5,r=5,t=40),
                annotations=[ dict(
                    showarrow=False,
                    xref="paper", yref="paper",
                    x=0.005, y=-0.002 ) ],
                xaxis=dict(showgrid=False, zeroline=False, showticklabels=False),
                yaxis=dict(showgrid=False, zeroline=False, showticklabels=False))
                )
fig.show()
```

Essa linha de código gera um grafo geométrico randômico com 200 nós em posições aleatórias. É criada uma aresta entre dois nós se eles forem gerados a uma distância menor do que 0.125

O código é um pouco mais complexo, mas a intenção é exibir um grafo com mais nós e arestas. Para simular a visualização do grafo, foi utilizado um gráfico de dispersão (*scatter*).



Fonte: https://plotly.com/ipython-notebooks/network-graphs/

Diagrama de Cordas

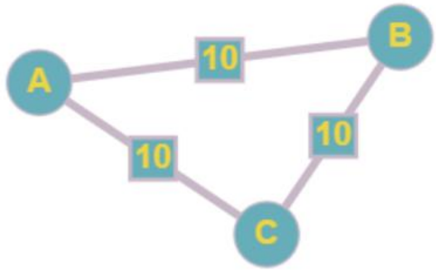
Todas as visualizações apresentadas até o momento são do tipo node-link, ou seja, representam grafos/redes como um conjunto de nós e arestas. Há vizualizações alternativas que podem ser mais adequadas para grafos valorados, como por exemplo, o diagrama de cordas.

Um diagrama de cordas representa um grafo dentro de um círculo. Cada elemento que seria um nó é representado como parte da circunferência. As arestas seriam representadas como “cordas” cuja largura seria proporcional ao valor/peso da aresta.

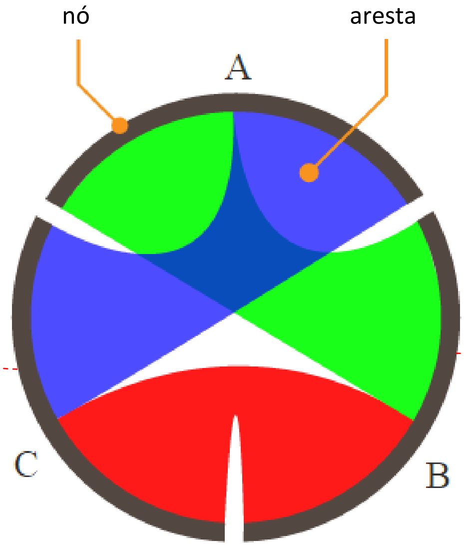
Veja o exemplo:

Considere a seguinte matriz de adjacência e sua representação em um grafo à direita (*node-link*):

	A	B	C
A		10	10
B	10		10
C	10	10	



A representação deste dataset como um diagrama de cordas seria:



Conheça mais sobre os Diagramas de Cordas em:
https://www.data-to-viz.com/graph/chord.html

Embora não haja suporte direto em Python, é possível construir gráficos de cordas usando **Plotly**, como no seguinte exemplo:
https://plotly.com/python/v3/filled-chord-diagram/
Uma biblioteca com melhor suporte a esses gráficos é a **Bokeh**.
https://holoviews.org/reference/elements/bokeh/Chord.html

Para saber mais, leia os capítulos iniciais dos e-books:

PERKOVIC, Ljubomir; VIEIRA, Daniel. Introdução à computação usando Python: um foco no desenvolvimento de aplicações. Rio de Janeiro: LTC, 2016.

McKinney, W. Python Para Análise de Dados: Tratamento de Dados com Pandas, NumPy e IPython. São Paulo: Novatec, 2018