Teoria da Computação - Relatório técnico

Universidade de Ribeirão Preto - Unaerp
Ana Clara Carnavalli Pereira - 837024

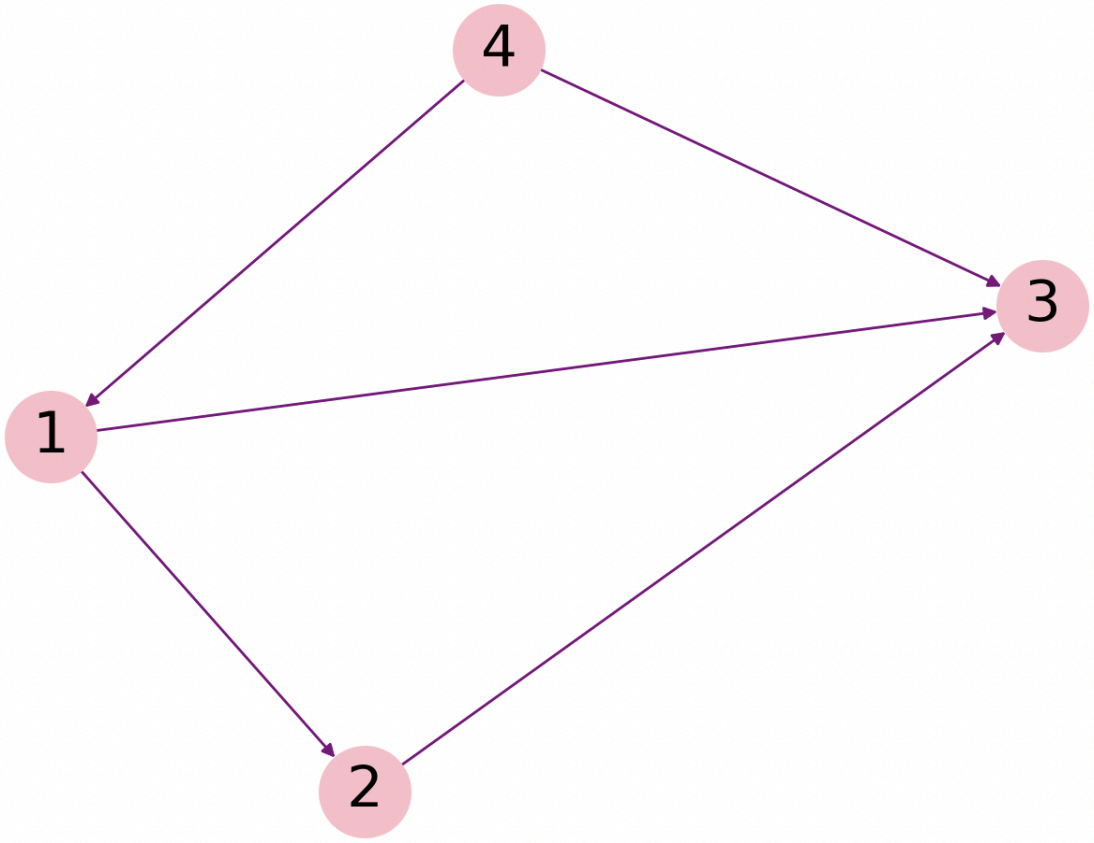
Utilizar o módulo NetworkX para manipulação de grafos.
Utilizei o VSCode para fazer a lista de exercícios.
Instalei a biblioteca matplotlib networkx para utilizar o networkx.
Exercício 1-

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

# Criar um grafo vazio
grafo_vazio = nx.Graph()

# Verificar se o grafo está vazio
if grafo_vazio.number_of_nodes() == 0 and grafo_vazio.number_of_edges() == 0:
print("O grafo está vazio!")
else:
print("O grafo não está vazio.")

# Criar um grafo direcionado vazio
grafo_direcionado = nx.DiGraph()

grafo_direcionado.add_nodes_from([1, 2, 3, 4])
grafo_direcionado.add_edges_from([(1, 2),(2, 3),(1, 3),(4, 1),(4, 3)])

# Desenhar o grafo direcionado
plt.figure()
nx.draw(grafo_direcionado, with_labels=True,
pos=nx.spring_layout(grafo_direcionado),
node_size=1200, node_color='pink', font_size=22, edge_color='purple')
plt.title('Grafo Direcionado')
plt.show()

if grafo_direcionado.number_of_nodes() == 0 and grafo_vazio.number_of_edges()
== 0:
print("O grafo está vazio!")
else:
print("O grafo não está vazio.")

# Criar grafo não-direcionado
grafo_nao_direcionado = nx.Graph()

grafo_nao_direcionado.add_nodes_from([1, 2, 3, 4])
grafo_nao_direcionado.add_edges_from([(1, 2),(2, 4),(1, 3),(2, 1),(3, 4)])
```

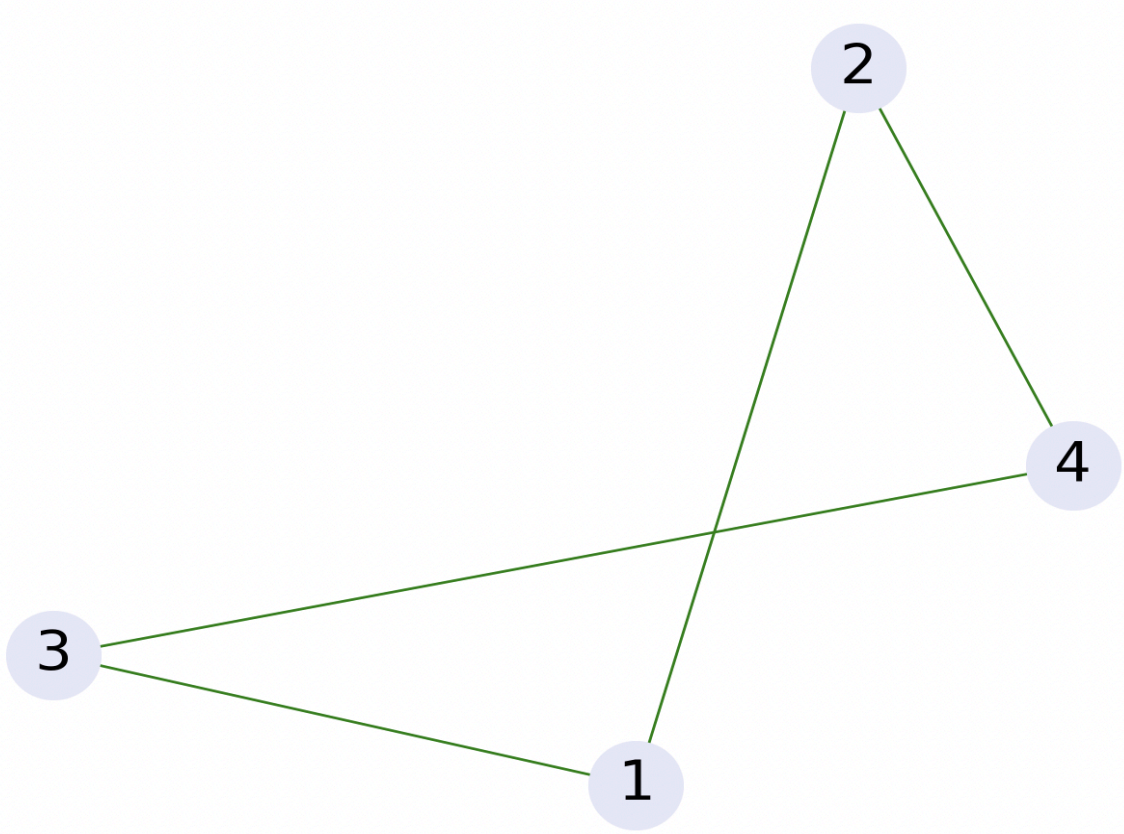
```
plt.figure()
nx.draw(grafo_nao_direcionado, with_labels=True,
pos=nx.spring_layout(grafo_nao_direcionado),
node_size=1200, node_color='lavender', font_size=22, edge_color='green')
plt.title('Grafo Não Direcionado')
plt.show()

if grafo_nao_direcionado.number_of_nodes() == 0 and
grafo_vazio.number_of_edges() == 0:
print("O grafo está vazio!")
else:
print("O grafo não está vazio.")
```

Terminal e Grafos
O código cria um grafo vazio (não mostra) e depois preenche ele e além disso cria um grafo direcionado e outro não direcionado.

Grafo direcionado:



Grafo não direcionado:



Terminal:

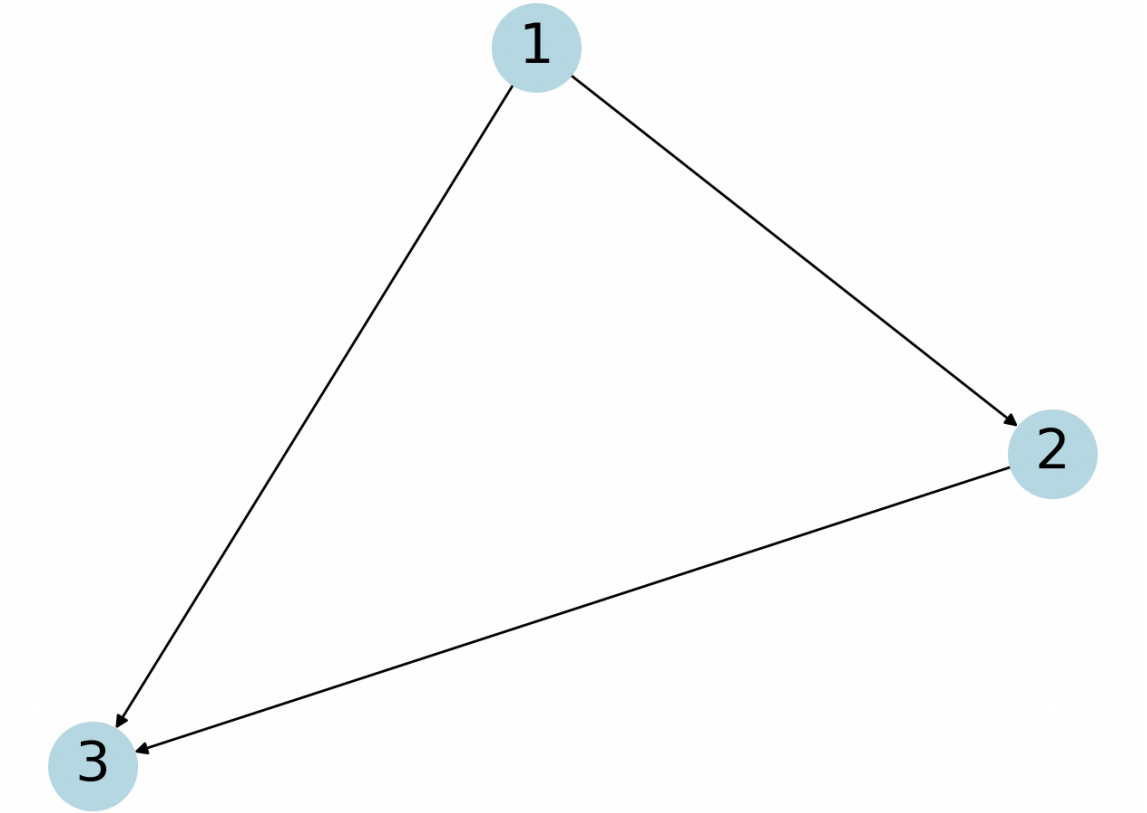
```
anaclara@MBPdeAnaClara lista-casa % /usr/local/bin/python3 "/Users/anaclara/Desktop/py plotz
e/lista-casa/grafo-vazio.py"
O grafo está vazio!
O grafo não está vazio.
O grafo não está vazio.
```

Exercícios 2 e 3:

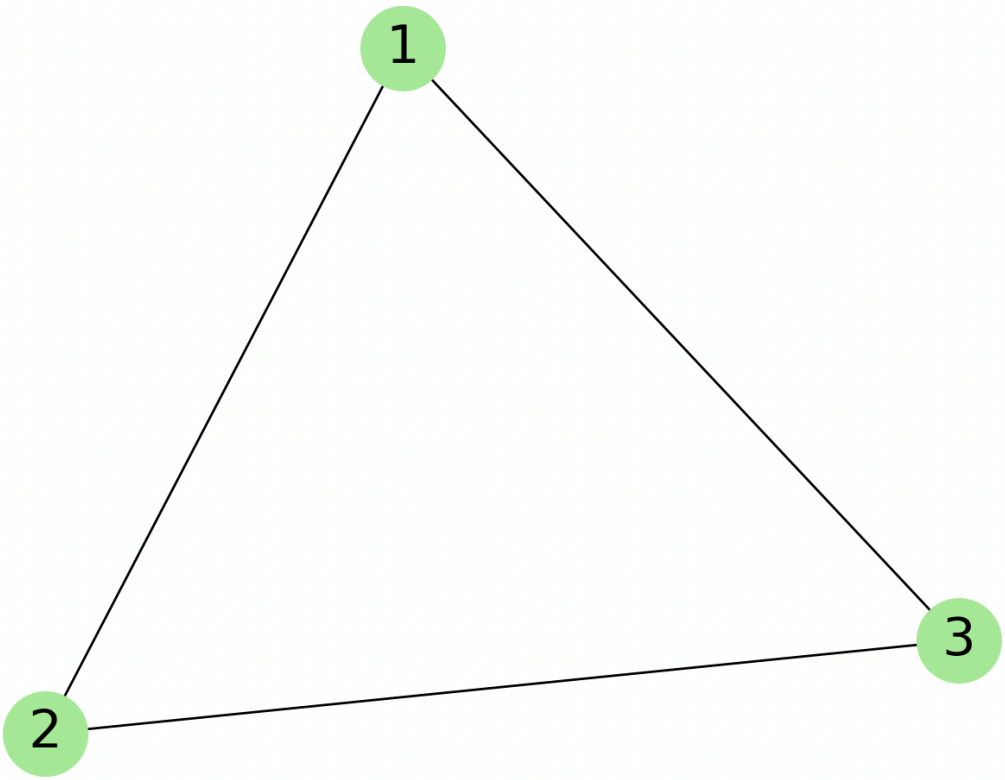
```python
#2- Adicionar nós e arestas: Escreva um código em Python que adicione três
nós (1, 2, 3) e três arestas ((1, 2), (1, 3), (2, 3)) a um grafo. Apresente
duas soluções: uma para grafo direcionado e outra para grafo não direcionado.
import networkx as nx
import matplotlib.pyplot as plt
# Grafo direcionado
grafo_direcionado = nx.DiGraph()
grafo_direcionado.add_nodes_from([1, 2, 3])
grafo_direcionado.add_edges_from([(1, 2), (1, 3), (2, 3)])

nx.draw(grafo_direcionado, with_labels=True,
pos=nx.spring_layout(grafo_direcionado),
node_size=1200, node_color='lightblue', font_size=22, edge_color='black')
```

```
plt.rcParams['axes.facecolor'] = 'lightpink'
plt.gca().set_facecolor('lightpink')

plt.axis('off')
plt.show()

# Grafo não direcionado
grafo_nao_direcionado = nx.Graph()
grafo_nao_direcionado.add_nodes_from([1, 2, 3])
grafo_nao_direcionado.add_edges_from([(1, 2), (1, 3), (2, 3)])

nx.draw(grafo_nao_direcionado, with_labels=True,
pos=nx.spring_layout(grafo_nao_direcionado),
node_size=1200, node_color='lightgreen', font_size=22, edge_color='black')

plt.rcParams['axes.facecolor'] = 'lightyellow'
plt.gca().set_facecolor('lightyellow')

plt.axis('off')
plt.show()

def verificar_presenca_no_grafo(grafo, node):
if node in grafo:
print(f"O nó de número {node} está presente no grafo.")
else:
print(f"O nó de número {node} não está presente no grafo.")

grafo = nx.Graph()
grafo.add_nodes_from([1, 2, 3])
grafo.add_edges_from([(1, 2), (1, 3), (2, 3)])

verificar_presenca_no_grafo(grafo, 1)

grafo2 = nx.Graph()
grafo2.add_nodes_from([4, 2, 3])
grafo2.add_edges_from([(4, 2), (4, 3), (4, 3)])

verificar_presenca_no_grafo(grafo2, 1)
```

Grafo direcionado



Grafo não direcionado



Terminal:

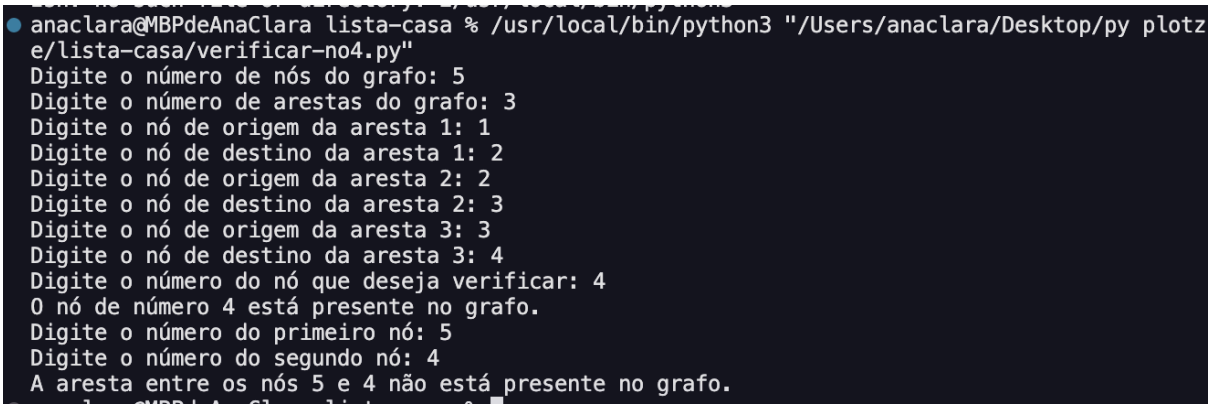
```
anaclara@MBPdeAnaClara lista-casa % /usr/local/bin/python3 "/Users/anaclara/Desktop/py plotz
e/lista-casa/nodes-edges2.py"
O nó de número 1 está presente no grafo.
O nó de número 1 não está presente no grafo.
```

Exercício 4 e 5:

```python
import networkx as nx

# Criando um grafo vazio
grafo = nx.Graph()

num_nos = int(input("Digite o número de nós do grafo: "))
num_arestas = int(input("Digite o número de arestas do grafo: "))

#Nós
for i in range(num_nos):
grafo.add_node(i+1)
#Arestas
for i in range(num_arestas):
no1 = int(input(f"Digite o nó de origem da aresta {i+1}: "))
no2 = int(input(f"Digite o nó de destino da aresta {i+1}: "))
grafo.add_edge(no1, no2)

#Exercício 4: Verificar a presença de um nó
node = int(input("Digite o número do nó que deseja verificar: "))
if grafo.has_node(node):
print(f"O nó de número {node} está presente no grafo.")
else:
print(f"O nó de número {node} não está presente no grafo.")

#Exercício 5: Verificar a presença de uma aresta
no1 = int(input("Digite o número do primeiro nó: "))
no2 = int(input("Digite o número do segundo nó: "))
if grafo.has_edge(no1, no2):
print(f"A aresta entre os nós {no1} e {no2} está presente no grafo.")
else:
print(f"A aresta entre os nós {no1} e {no2} não está presente no grafo.")
```

Terminal:

```
Digite o número de nós do grafo: 5
Digite o número de arestas do grafo: 3
Digite o nó de origem da aresta 1: 1
Digite o nó de destino da aresta 1: 2
Digite o nó de origem da aresta 2: 2
Digite o nó de destino da aresta 2: 3
Digite o nó de origem da aresta 3: 3
Digite o nó de destino da aresta 3: 4
Digite o número do nó que deseja verificar: 4
O nó de número 4 está presente no grafo.
Digite o número do primeiro nó: 5
Digite o número do segundo nó: 4
A aresta entre os nós 5 e 4 não está presente no grafo.
```

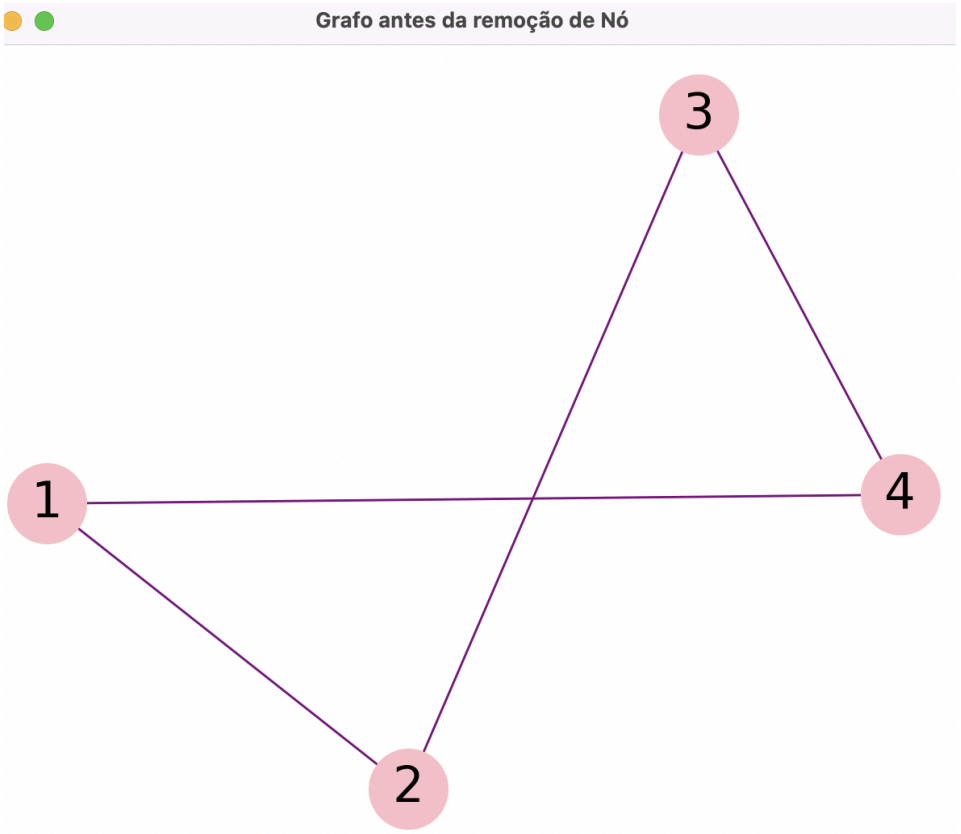


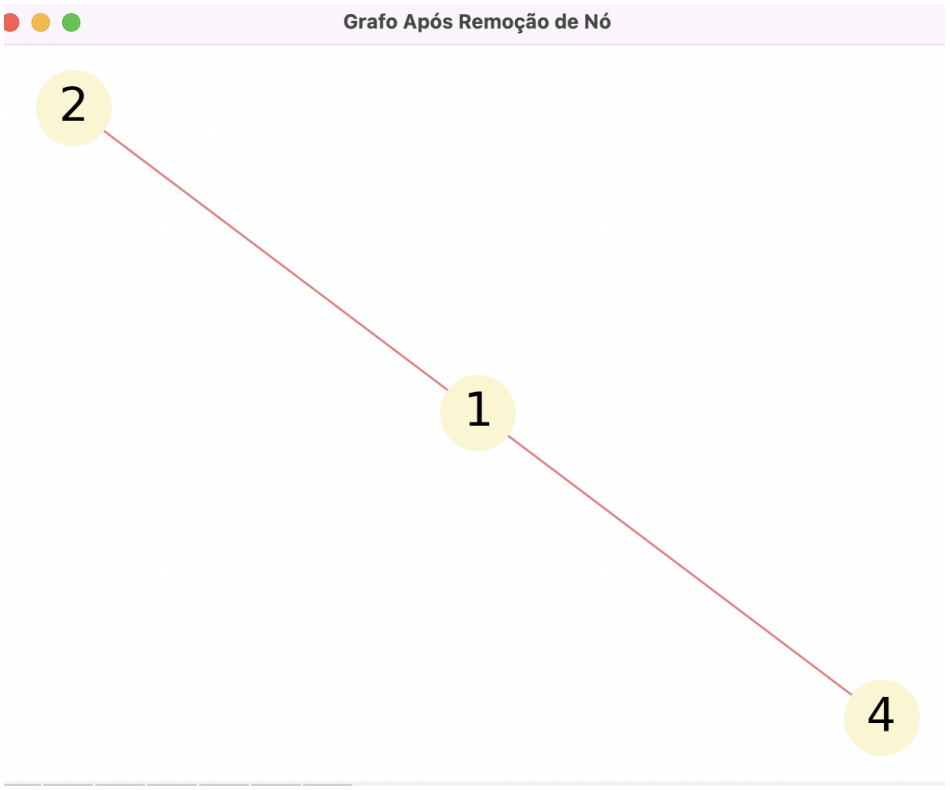
Exercício 6:

```python
# 6- Remover nós e arestas: Escreva um código para demonstrar como remover um
nó e uma aresta no grafo.
import networkx as nx
import matplotlib.pyplot as plt

#Criação do grafo
G = nx.Graph()
G.add_nodes_from([1, 2, 3, 4])
G.add_edges_from([(1, 2), (2, 3), (3, 4), (4, 1)])

print("Grafo antes da remoção:")
plt.figure("Grafo antes da remoção de Nó")
nx.draw(G, with_labels=True, pos=nx.spring_layout(G),
node_size=1200, node_color='pink', font_size=22, edge_color='purple',
arrows=False)
plt.show()
```

```
#Nó
no_removido = 3
G.remove_node(no_removido)

#Exibir grafo após remoção
print("\nGrafo após a remoção do nó", no_removido, ":")
plt.figure("Grafo Após Remoção de Nó")
nx.draw(G, with_labels=True, pos=nx.spring_layout(G),
node_size=1200, node_color='lightgoldenrodyellow', font_size=22,
edge_color='lightcoral', arrows=False)
plt.show()

#Aresta
aresta_removida = (1, 2)
G.remove_edge(*aresta_removida)

#Exibir grafo após remoção
print("\nGrafo após a remoção da aresta", aresta_removida, ":")
plt.figure("Grafo Após Remoção de Aresta")
nx.draw(G, with_labels=True, pos=nx.spring_layout(G),
node_size=1200, node_color='lightblue', font_size=22,
edge_color='lightseagreen', arrows=False)
plt.show()
```

Demonstração com os grafos:





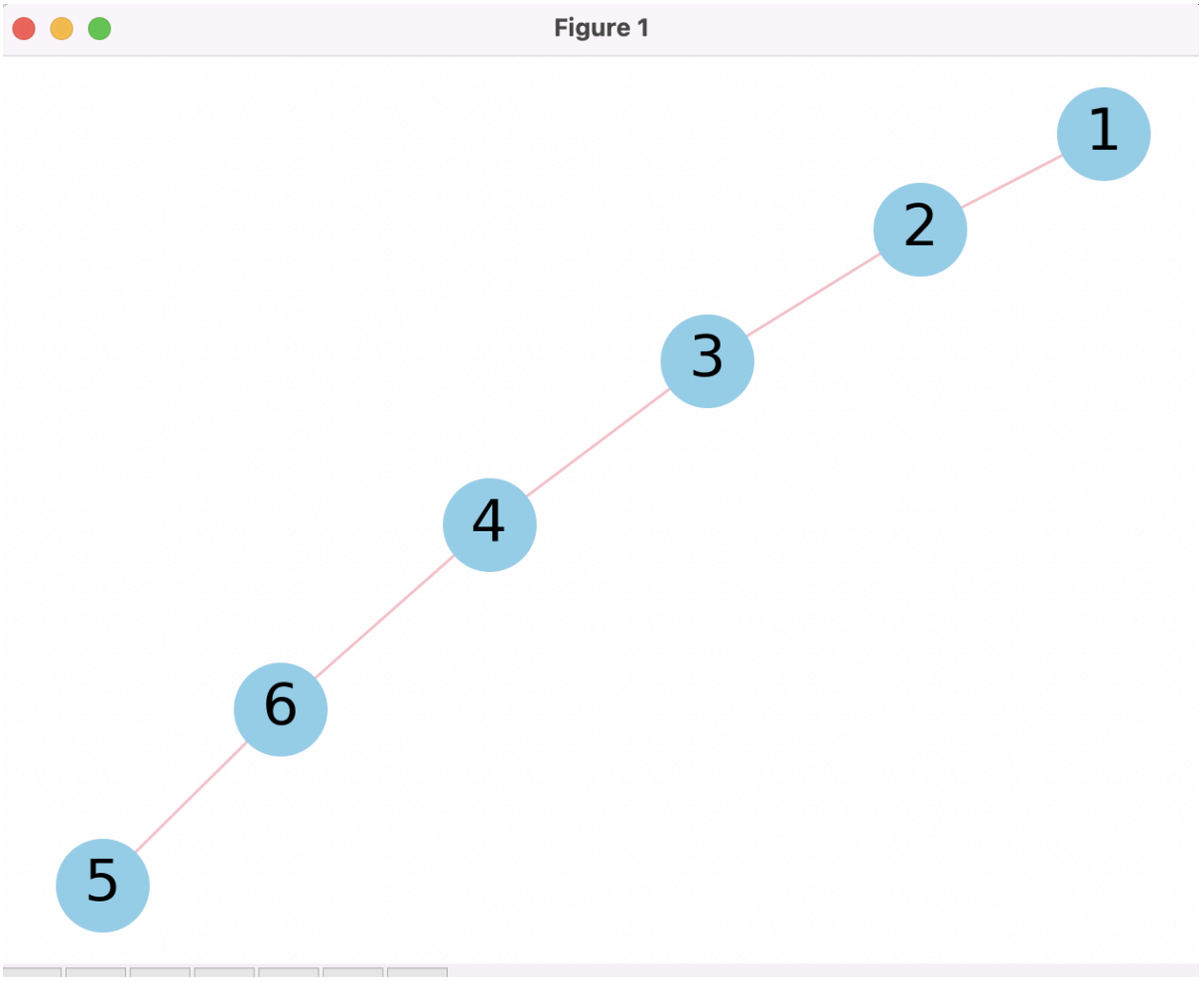
Exercícios: 7, 8 ,9, 10 e 12. Preferi implementar todos num código só

```
# 7- Calcular o número de nós e arestas: Escreva um código que calcule e
imprima o número de nós e o número de arestas de um grafo.

# 8- Calcular o grau de um nó: Escreva um código em Python que calcule e
imprima o grau de um nó do grafo.

# 9- Encontrar todos os vizinhos de um nó: Escreva um código que encontre e
imprima todos os vizinhos de um determinado nó do grafo.

# 10- Calcular o caminho mais curto entre dois nós: Escreva um código que
calcule e imprima o caminho mais curto entre dois nós de um grafo.

import networkx as nx
import matplotlib.pyplot as plt #visualizar o grafo

def calcular(grafo):
num_nos = grafo.number_of_nodes()
```

```
num_arestas = grafo.number_of_edges()
return num_nos, num_arestas

def calcular_grau(grafo, no):
grau = grafo.degree[no]
print(f"O grau do nó {no} é {grau}.")

def encontrar_vizinhos(grafo,no):
vizinhos = list(grafo.neighbors(no))
print(f"Os vizinhos do nó {no} são: ", vizinhos)

def caminho_mais_curto(grafo, no_origem, no_destino):
try:
caminho = nx.shortest_path(grafo, source=no_origem, target=no_destino)
print(f"O caminho mais curto entre os nós {no_origem} e {no_destino} é:",
caminho)
except nx.NetworkXNoPath:
print(f"Não há caminho entre os nós {no_origem} e {no_destino}.")

def verificar_ciclos(grafo):
ciclos = nx.cycle_basis(grafo)
if ciclos:
print("O grafo possui os seguintes ciclos:")
for ciclo in ciclos:
print(ciclo)
else:
print("O grafo não possui ciclos.")

def plotar_grafo(grafo):
pos = nx.spring_layout(grafo) # Posicionamento dos nós
nx.draw(grafo, pos, with_labels=True, node_size=1200, node_color='skyblue',
font_size=22, edge_color='pink', arrows=False)
plt.show()

grafo = nx.Graph()

#Entrada de dados
num_nos = int(input("Digite o número de nós do grafo: "))
num_arestas = int(input("Digite o número de arestas do grafo: "))

for i in range(1, num_nos + 1):
```

```
grafo.add_node(i)

#Entrada de dados2
print("Insira as arestas no formato 'nó1 nó2':")
for i in range(num_arestas):
aresta = input(f"Aresta {i+1}: ").split()
grafo.add_edge(int(aresta[0]), int(aresta[1]))

# Calcular
num_nos_calculado, num_arestas_calculado = calcular(grafo)

no_desejado = int(input("Digite o número do nó para calcular o grau: "))

no_origem = int(input("\nDigite o número do nó de origem: "))
no_destino = int(input("Digite o número do nó de destino: "))

print("\nNúmero de nós:", num_nos_calculado)
print("Número de arestas:", num_arestas_calculado)

#Calcular o grau do nó
calcular_grau(grafo, no_desejado)

#Vizinhos
encontrar_vizinhos(grafo, no_desejado)

# Caminho mais curto
caminho_mais_curto(grafo, no_origem, no_destino)

# Verificar ciclos
verificar_ciclos(grafo)

#Plotar grafo
plotar_grafo(grafo)
```

Grafo representado: (Optei representar para ficar melhor a compreensão.)



Terminal:

```
Digite o número de nós do grafo: 6
Digite o número de arestas do grafo: 5
Insira as arestas no formato 'nó1 nó2':
Aresta 1: 1 2
Aresta 2: 2 3
Aresta 3: 3 4
Aresta 4: 5 6
Aresta 5: 4 6
Digite o número do nó para calcular o grau: 4

Digite o número do nó de origem: 1
Digite o número do nó de destino: 4

Número de nós: 6
Número de arestas: 5
O grau do nó 4 é 2.
Os vizinhos do nó 4 são:  [3, 6]
```

```
O caminho mais curto entre os nós 1 e 4 é: [1, 2, 3, 4]
O grafo não possui ciclos.
```

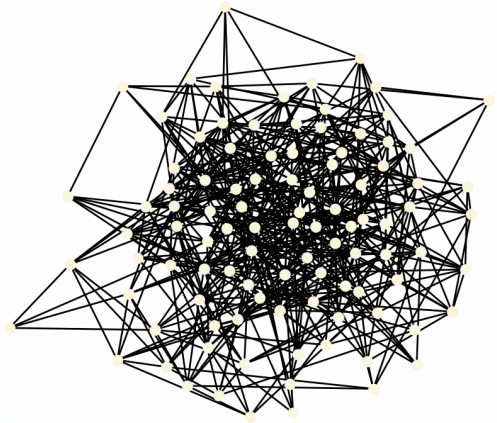
```
anaclara@MBPdeAnaClara lista-casa % /usr/local/bin/python3 "/Users/anaclara/Desktop/py plotz
e/lista-casa/calculos.py"
Digite o número de nós do grafo: 6
Digite o número de arestas do grafo: 5
Insira as arestas no formato 'nó1 nó2':
Aresta 1: 1 2
Aresta 2: 2 3
Aresta 3: 3 4
Aresta 4: 5 6
Aresta 5: 4 6
Digite o número do nó para calcular o grau: 4

Digite o número do nó de origem: 1
Digite o número do nó de destino: 4

Número de nós: 6
Número de arestas: 5
O grau do nó 4 é 2.
Os vizinhos do nó 4 são:  [3, 6]
O caminho mais curto entre os nós 1 e 4 é: [1, 2, 3, 4]
O grafo não possui ciclos.
```

Exercício 11-

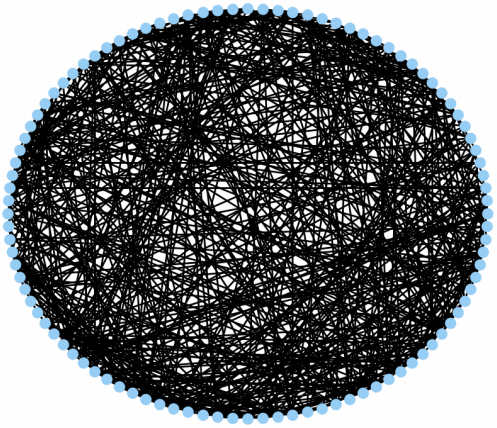
```
# 11- Grafo Aleatório: Escreva um código que crie um grafo aleatório com 100
nós e suas respectivas arestas utilizando a biblioteca NetworkX. Apresentar
diversas formas de visualização do grafo, explorando os tipos de layout.

import networkx as nx
import matplotlib.pyplot as plt

# Criar um grafo aleatório com 100 nós e probabilidade de aresta de 0.1
grafo_aleatorio = nx.gnp_random_graph(100, 0.1)

# Paleta de cores
paleta_cores = [
'#FF69B4',
'#FFF8DC',
'#87CEFA',
'#9370DB'
]
#Layouts escolhidos
layouts = [
("Spring Layout", nx.spring_layout),
("Circular Layout", nx.circular_layout),
("Random Layout", nx.random_layout),
("Shell Layout", nx.shell_layout),
("Spectral Layout", nx.spectral_layout),
```

```
("Kamada Kawai Layout", nx.kamada_kawai_layout)
]
# Plotar o grafo com as cores definidas
plt.figure(figsize=(15, 10))
for i, (layout_name, layout_function) in enumerate(layouts, 1):
plt.subplot(2, 3, i)
pos = layout_function(grafo_aleatorio)
nx.draw(grafo_aleatorio, pos, with_labels=False, node_size=30,
node_color=paleta_cores[i%len(paleta_cores)], edge_color='black')
plt.title(layout_name)
plt.tight_layout()
plt.show()
```

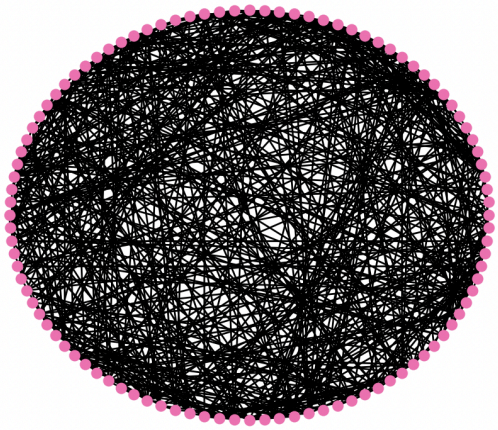
Spring Layout



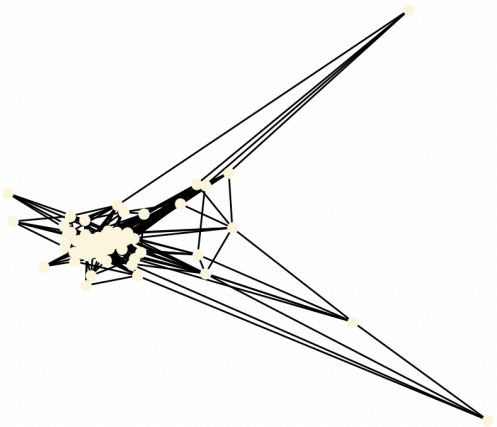
Circular Layout



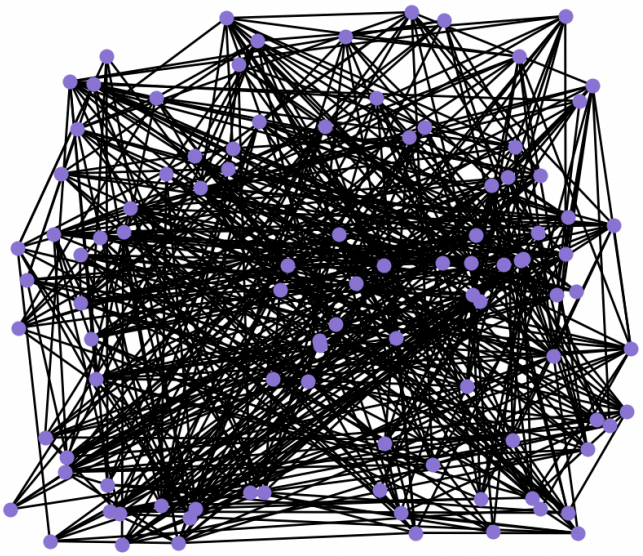
Shell Layout



Spectral Layout



Random Layout



Kamada Kawai Layout