



Aula 02



Exercício 01:

 Chat, a partir de agora você servirá como uma ferramenta de criação de grafos. Esteja pronto para receber o problema e dê a resposta da seguinte forma:

1. Lista de adjacência
2. Matriz de adjacência
3. Nós, arestas conectadas e graus
4. Código em Python para representar graficamente essa solução, com bibliotecas de plotagem de grafos.

Informações adicionais:
O grafo é do tipo DIRECIONADO.
Quando for gerar as listas, utilize as iniciais no lugar de índices numéricos.
Não dê nenhuma resposta ainda, enquanto eu não inserir o problema.





 Entendido! Estou pronto para ajudar na criação de grafos de tipo direcionado. Por favor, forneça o problema que você gostaria de representar como um grafo direcionado, e então eu fornecerei as respostas nos formatos que você solicitou.

 **Problema:** 

Um grupo de amigos querem identificar todos aqueles que já fizeram empréstimos entre si através de um grafo.

Para isso, recorreram ao histórico de mensagens e notaram as seguinte transações: Alberto fez empréstimo para Beltrano, Ciclano fez empréstimo para Alberto, Beltrando fez empréstimo para Fulana, Fulana fez empréstimo para Deltrana, Beltrano fez empréstimo para Deltrana, Deletrana fez empréstimo para Ciclano e Gustavo fez empréstimo para Alberto.

Resolva e apresente em notação especial


 Vamos resolver esse problema passo a passo e representar o grafo direcionado resultante nas formas solicitadas.   

Passo 1: Identificar os nós (amigos) e as arestas (empréstimos):

Nós (Amigos): Alberto, Beltrano, Ciclano, Fulana, Deletrana, Gustavo


Arestas (Empréstimos):

- Alberto -> Beltrano
- Ciclano -> Alberto
- Beltrano -> Fulana
- Fulana -> Deletrana
- Beltrano -> Deletrana
- Deletrana -> Ciclano
- Gustavo -> Alberto

 Regenerate

Passo 2: Representação em Lista de Adjacência:


makefile

 Copy code

```
Alberto: [Beltrano]
Beltrano: [Fulana, Deletrana]
Ciclano: [Alberto]
Fulana: [Deletrana]
Deletrana: [Ciclano]
Gustavo: [Alberto]
```

Passo 3: Representação em Matriz de Adjacência (Ordenados alfabeticamente):


markdown

 Copy code

	Alberto	Beltrano	Ciclano	Deletrana	Fulana	Gustavo
Alberto	0	1	0	0	0	0
Beltrano	0	0	0	1	1	0
Ciclano	1	0	0	0	0	0
Deletrana	0	0	1	0	1	0
Fulana	0	0	0	0	0	0
Gustavo	1	0	0	0	0	0

Passo 4: Nós, Arestas Conectadas e Graus:


java

 Copy code

```
Nós: ['Alberto', 'Beltrano', 'Ciclano', 'Fulana', 'Deletrana', 'Gustavo']
Arestas Conectadas: 7
Graus:
- Alberto: Grau de Saída = 1, Grau de Entrada = 1
- Beltrano: Grau de Saída = 2, Grau de Entrada = 1
- Ciclano: Grau de Saída = 1, Grau de Entrada = 1
- Fulana: Grau de Saída = 1, Grau de Entrada = 1
- Deletrana: Grau de Saída = 1, Grau de Entrada = 2
- Gustavo: Grau de Saída = 1, Grau de Entrada = 0
```

Passo 5: Código em Python para representar graficamente o grafo:

python

 Copy code

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

G = nx.DiGraph()

# Adicionar nós
G.add_nodes_from(['Alberto', 'Beltrano', 'Ciclano', 'Fulana', 'Deletrana', 'Gustavo'])

# Adicionar arestas
edges = [('Alberto', 'Beltrano'), ('Ciclano', 'Alberto'), ('Beltrano', 'Fulana'),
         ('Fulana', 'Deletrana'), ('Beltrano', 'Deletrana'), ('Deletrana', 'Gustavo'),
         ('Gustavo', 'Alberto')]
G.add_edges_from(edges)

# Plotar o grafo
pos = nx.spring_layout(G, seed=42) # Layout para posicionar os nós
nx.draw(G, pos, with_labels=True, node_size=2000, node_color='lightblue', font_size=16)
plt.title("Grafo de Empréstimos entre Amigos")
plt.show()
```

Esse código utilizará a biblioteca NetworkX para criar e plotar o grafo direcionado com os nós e arestas correspondentes às transações de empréstimos entre os amigos. Certifique-se de instalar as bibliotecas necessárias, como NetworkX e Matplotlib, caso ainda não estejam instaladas em seu ambiente Python.

 Regenerar

Código:

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

# Criando o grafo
G = nx.DiGraph()

# Adicionando arestas
edges = [('Alberto', 'Beltrano'), ('Ciclano', 'Alberto'), ('Beltrano', 'Fulana'),
```

```

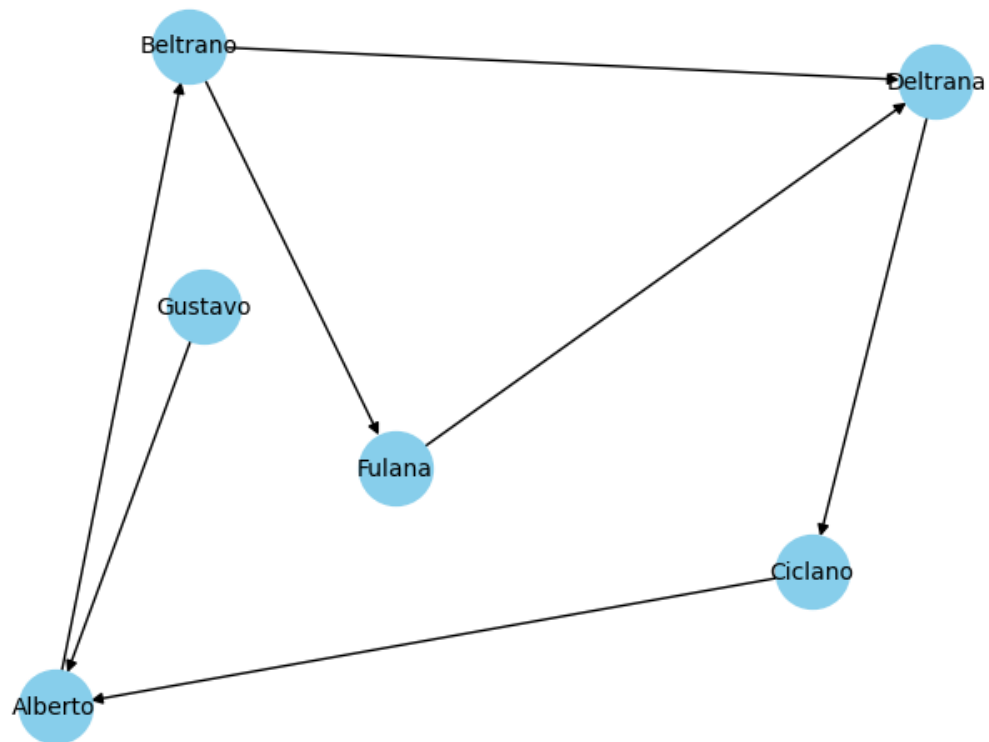
        ('Fulana', 'Deltrana'), ('Beltrano', 'Deltrana'), ('Deltrana',
'Ciclano'),
        ('Gustavo', 'Alberto')]
G.add_edges_from(edges)

# Plot do grafo
pos = nx.spring_layout(G, seed=42) # Define a posição dos nós
nx.draw(G, pos, with_labels=True, node_size=1000, node_color='skyblue',
font_size=10, font_color='black')
plt.title("Grafo Direcionado de Empréstimos Entre Amigos")
plt.show()

```

➤

Grafo Direcionado de Empréstimos Entre Amigos



Exercício 02:



O grafo A é um subgrafo de B se os vértices e arestas de A são vértices e arestas de B , ou seja, o subgrafo A é uma parte do grafo B .

O grafo é do tipo não direcionado.

Problema:

Os vértices de G são 1, 2, 3, 4, 5, 6 e as arestas são: 1 para 2, 1 para 5, 2 para 4, 2 para 3, 3 para 4, 3 para 6, 4 para 6, 4 para 5 e 5 para 6.

Os vértices de G_1 são 4, 5 e 6.

Resolva e mostre se G_1 é subgrafo de G ?



Vamos analisar os vértices e arestas de G_1 para determinar se é um subgrafo de G :



G_1 vértices: 4, 5, 6

G_1 arestas: Não temos informações diretas sobre as arestas de G_1 , então precisamos verificar se as arestas entre os vértices de G_1 estão presentes em G .

Arestas em G que envolvem vértices de G_1 :

- Aresta entre 4 e 5 (4 para 5) - presente em G
- Aresta entre 4 e 6 (4 para 6) - presente em G
- Aresta entre 5 e 6 (5 para 6) - presente em G

Todos os vértices de G_1 (4, 5 e 6) estão presentes em G e as arestas entre esses vértices também estão presentes em G . Portanto, G_1 é de fato um subgrafo de G .

Em termos de definição, um subgrafo de um grafo G é um grafo que possui um conjunto de vértices e um conjunto de arestas que são um subconjunto dos vértices e arestas de G .

Desde que todos os vértices de G_1 e suas arestas estejam presentes em G , podemos concluir que G_1 é um subgrafo de G .


Exercício 03:



Se o subgrafo G_1 de G for induzido pelos vértices 4, 5 e 6, significa que estaremos considerando apenas esses três vértices e as arestas que existem entre eles no grafo original G . Isso resultará em um subgrafo que possui somente os vértices 4, 5 e 6, e as arestas 4-5, 4-6 e 5-6.

Vamos representar o subgrafo G_1 usando um dicionário em Python, onde as chaves representam os vértices e os valores são listas de vértices vizinhos (ou seja, aqueles que estão conectados por uma aresta). Aqui está como podemos construir o subgrafo G_1 com base nas informações fornecidas:

python

 Copy code

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

# Definindo as arestas do grafo original G
arestas_G = [(1, 2), (1, 5), (2, 3), (2, 4), (3, 4), (3, 6), (4, 5), (4, 6),

# Definindo o conjunto de vértices para o subgrafo G1
vertices_G1 = [4, 5, 6]

# Filtrando as arestas que pertencem ao subgrafo G1
arestas_G1 = [aresta for aresta in arestas_G if aresta[0] in vertices_G1 and

# Construindo o subgrafo G1
subgrafo_G1 = nx.Graph()
subgrafo_G1.add_nodes_from(vertices_G1)
subgrafo_G1.add_edges_from(arestas_G1)

# Plotando o subgrafo G1
pos = nx.spring_layout(subgrafo_G1) # Layout do plot
nx.draw(subgrafo_G1, pos, with_labels=True, node_color='lightblue', font_wi
plt.title("Subgrafo Induzido G1")
plt.show()
```



Temos o seguinte grafo não direcionado G:

Vértices = 1, 2, 3, 4, 5 e 6

Arestas = 1 para 2, 1 para 5, 2 para 3, 2 para 4, 3 para 4, 3 para 6, 4 para 5, 4 para 6 e 5 para 6

Um subgrafo induzido G1 de G pode ser induzido por vértices ou arestas.

Um subgrafo induzido G1 por vértices possui arestas que são as mesmas de G.

Um subgrafo induzido G1 por arestas possui vértices que são as mesmas de G.

Problema: Se o subgrafo G1 de G for induzido pelos vértices 4, 5 e 6, como será o subgrafo G1? Explique e demonstre com código python.

Código:

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

# Definindo as arestas do grafo original G
arestas_G = [(1, 2), (1, 5), (2, 3), (2, 4), (3, 4), (3, 6), (4, 5),
             (4, 6), (5, 6)]

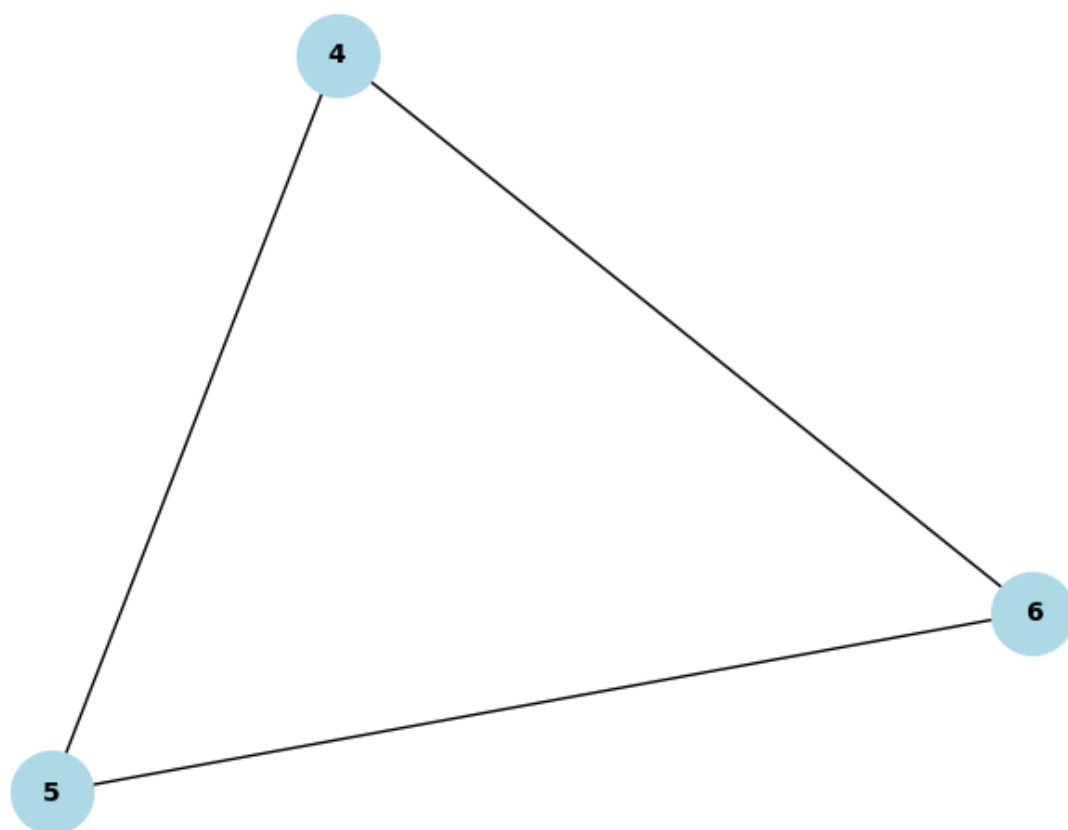
# Definindo o conjunto de vértices para o subgrafo G1
vertices_G1 = [4, 5, 6]

# Filtrando as arestas que pertencem ao subgrafo G1
arestas_G1 = [aresta for aresta in arestas_G if aresta[0] in
              vertices_G1 and aresta[1] in vertices_G1]

# Construindo o subgrafo G1
subgrafo_G1 = nx.Graph()
subgrafo_G1.add_nodes_from(vertices_G1)
subgrafo_G1.add_edges_from(arestas_G1)

# Plotando o subgrafo G1
pos = nx.spring_layout(subgrafo_G1) # Layout do plot
nx.draw(subgrafo_G1, pos, with_labels=True, node_color='lightblue',
        font_weight='bold', font_size=10, node_size=1000)
plt.title("Subgrafo Induzido G1")
plt.show()
```

Subgrafo Induzido G1



Exercício 04:

GM

Temos o seguinte grafo não direcionado G:



Vértices = 1, 2, 3, 4, 5 e 6

Arestas = 1 para 2, 1 para 5, 2 para 3, 2 para 4, 3 para 4, 3 para 6, 4 para 5, 4 para 6 e 5 para 6, respectivamente com nomes a, d, b, c, e, h, f, g e i

Um subgrafo induzido G1 de G pode ser induzido por vértices ou arestas.

Um subgrafo induzido G1 por arestas possui vértices que são os mesmos de G.

Problema: Quero que gere um código python para plotar apenas o subgrafo induzido G1 pelas arestas.

python

Copy code

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

# Criando o grafo G
G = nx.Graph()
edges = [("1", "2", {"label": "a"}), ("1", "5", {"label": "d"}), ("2", "3", {"label": "b"}), ("2", "4", {"label": "c"}), ("3", "4", {"label": "e"}), ("3", "6", {"label": "h"}), ("4", "5", {"label": "f"}), ("4", "6", {"label": "g"}), ("5", "6", {"label": "i"})]
G.add_edges_from(edges)

# Lista de arestas do subgrafo induzido G1
edges_g1 = [("2", "4"), ("3", "6"), ("4", "6")]

# Criando o subgrafo induzido G1
G1 = G.edge_subgraph(edges_g1)

# Plotando o grafo G1
pos = nx.spring_layout(G1) # Definindo o layout do plot
nx.draw(G1, pos, with_labels=True, node_size=1000, font_size=10, font_color='black')
nx.draw_networkx_edge_labels(G1, pos, edge_labels={(u, v): G[u][v]['label']} for (u, v) in edges_g1)
plt.title("Subgrafo Induzido G1 pelas Arestas c, g e h")
plt.show()
```

Código:

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

# Criando o grafo G
```

```

G = nx.Graph()
edges = [("1", "2", {"label": "a"}), ("1", "5", {"label": "d"}), ("2",
"3", {"label": "b"}),
        ("2", "4", {"label": "c"}), ("3", "4", {"label": "e"}), ("3",
"6", {"label": "h"}),
        ("4", "5", {"label": "f"}), ("4", "6", {"label": "g"}), ("5",
"6", {"label": "i"})]
G.add_edges_from(edges)

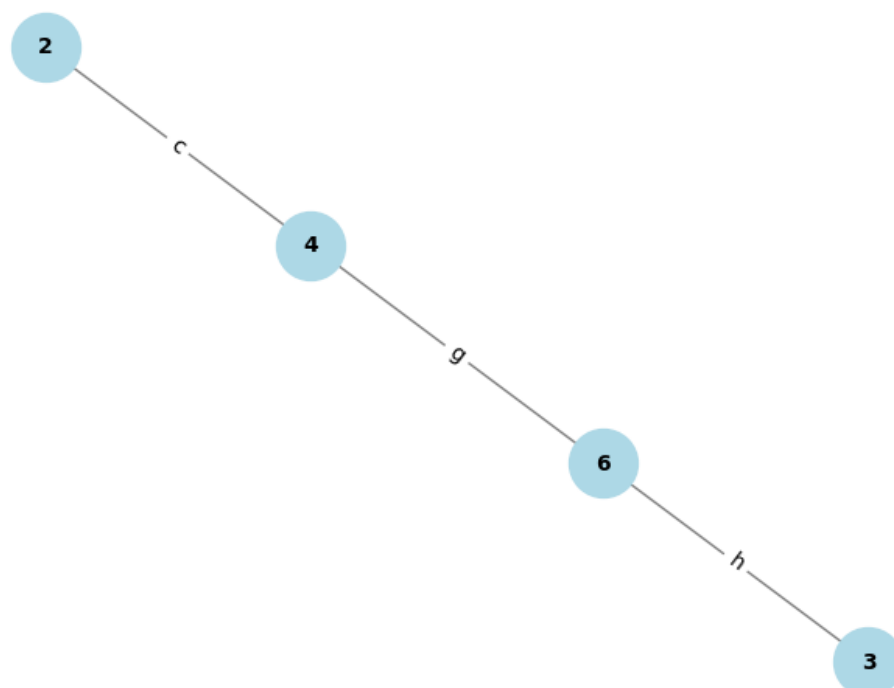
# Lista de arestas do subgrafo induzido G1
edges_g1 = [("2", "4"), ("3", "6"), ("4", "6")]

# Criando o subgrafo induzido G1
G1 = G.edge_subgraph(edges_g1)

# Plotando o grafo G1
pos = nx.spring_layout(G1) # Definindo o layout do plot
nx.draw(G1, pos, with_labels=True, node_size=1000, font_size=10,
font_color='black', font_weight='bold', node_color='lightblue',
edge_color='gray')
nx.draw_networkx_edge_labels(G1, pos, edge_labels={(u, v):
G[u][v]['label'] for u, v in G1.edges()}, font_color='black')
plt.title("Subgrafo Induzido G1 pelas Arestas c, g e h")
plt.show()

```

Subgrafo Induzido G1 pelas Arestas c, g e h



Exercício 05:

a) $V' = \{3, 5, 6, 8, 9\}$

Código:

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

# Definindo as arestas do grafo original G
arestas_G = [(1, 2), (1, 3), (2, 4), (3, 4), (3, 5), (4, 6), (4, 7),
             (6, 8), (6, 9), (7, 9), (7, 10), (8, 10)]

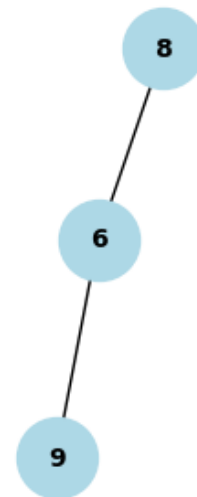
# Definindo o conjunto de vértices para o subgrafo G1
vertices_G1 = [3, 5, 6, 8, 9]

# Filtrando as arestas que pertencem ao subgrafo G1
arestas_G1 = [aresta for aresta in arestas_G if aresta[0] in
              vertices_G1 and aresta[1] in vertices_G1]

# Construindo o subgrafo G1
subgrafo_G1 = nx.Graph()
subgrafo_G1.add_nodes_from(vertices_G1)
subgrafo_G1.add_edges_from(arestas_G1)

# Plotando o subgrafo G1
pos = nx.spring_layout(subgrafo_G1) # Layout do plot
nx.draw(subgrafo_G1, pos, with_labels=True, node_color='lightblue',
        font_weight='bold', font_size=10, node_size=1000)
plt.title("Subgrafo Induzido G1")
plt.show()
```

Subgrafo Induzido G1



b) $E' = \{(1,3), (4,7), (7,9), (8,10)\}$

Código:

```
import networkx as nx
import matplotlib.pyplot as plt
arestas_G = [(1, 2), (1, 3), (2, 4), (3, 4), (3, 5), (4, 6), (4, 7),
(6, 8), (6, 9), (7, 9), (7, 10), (8, 10)]
# Criando o grafo G
G = nx.Graph()
edges = [("1", "2", {"label": "a"}), ("1", "3", {"label": "b"}), ("2",
"4", {"label": "c"}), ("3", "4", {"label": "d"}),
        ("3", "5", {"label": "e"}), ("4", "6", {"label": "f"}), ("4",
"7", {"label": "g"}), ("6", "8", {"label": "h"}),
        ("6", "9", {"label": "i"}), ("7", "9", {"label": "j"}), ("7",
"10", {"label": "k"}), ("8", "10", {"label": "l"})]
G.add_edges_from(edges)

# Lista de arestas do subgrafo induzido G1
edges_g1 = [("1", "3"), ("4", "7"), ("7", "9"), ("8", "10")]
```

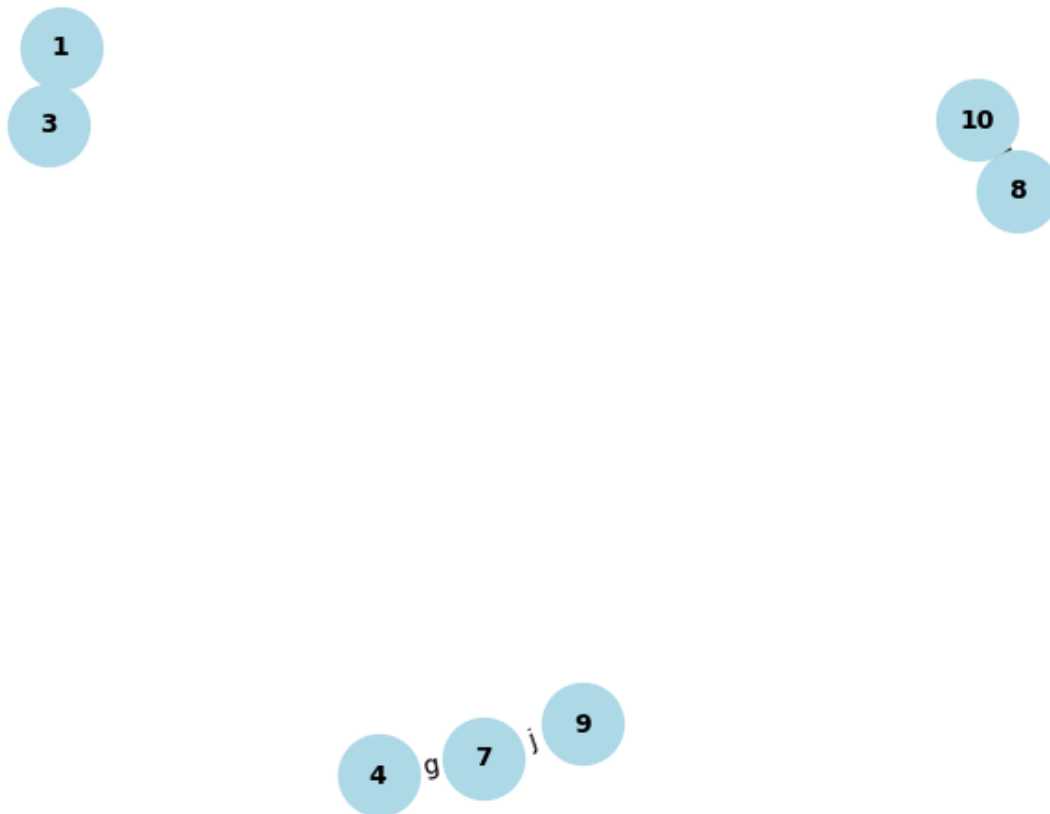
```

# Criando o subgrafo induzido G1
G1 = G.edge_subgraph(edges_g1)

# Plotando o grafo G1
pos = nx.spring_layout(G1) # Definindo o layout do plot
nx.draw(G1, pos, with_labels=True, node_size=1000, font_size=10,
font_color='black', font_weight='bold', node_color='lightblue',
edge_color='gray')
nx.draw_networkx_edge_labels(G1, pos, edge_labels={(u, v):
G[u][v]['label'] for u, v in G1.edges()} , font_color='black')
plt.title("Subgrafo Induzido G1 pelas Arestas c, g e h")
plt.show()

```

Subgrafo Induzido G1



c) $V' = \{1, 3, 4, 7, 8, 9, 10\}$

Código:

```

import networkx as nx
import matplotlib.pyplot as plt

# Definindo as arestas do grafo original G

```

```

arestas_G = [(1, 2), (1, 3), (2, 4), (3, 4), (3, 5), (4, 6), (4, 7),
(6, 8), (6, 9), (7, 9), (7, 10), (8, 10)]

# Definindo o conjunto de vértices para o subgrafo G1
vertices_G1 = [1, 3, 4, 7, 8, 9, 10]

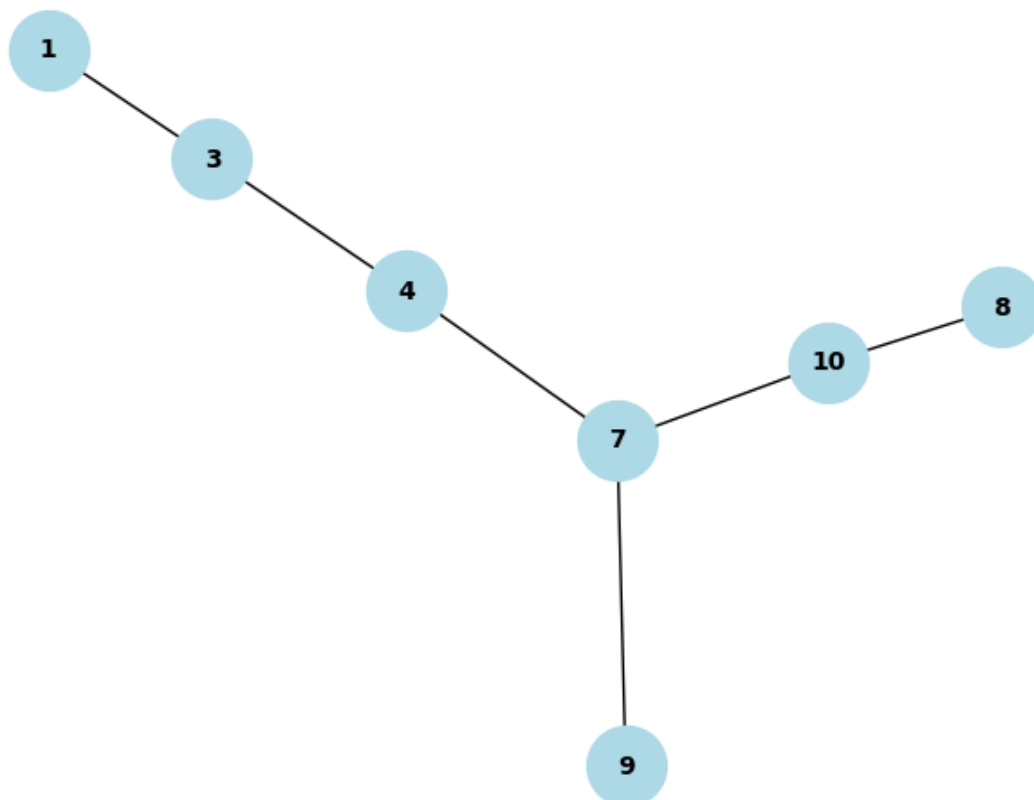
# Filtrando as arestas que pertencem ao subgrafo G1
arestas_G1 = [aresta for aresta in arestas_G if aresta[0] in
vertices_G1 and aresta[1] in vertices_G1]

# Construindo o subgrafo G1
subgrafo_G1 = nx.Graph()
subgrafo_G1.add_nodes_from(vertices_G1)
subgrafo_G1.add_edges_from(arestas_G1)

# Plotando o subgrafo G1
pos = nx.spring_layout(subgrafo_G1) # Layout do plot
nx.draw(subgrafo_G1, pos, with_labels=True, node_color='lightblue',
font_weight='bold', font_size=10, node_size=1000)
plt.title("Subgrafo Induzido G1")
plt.show()

```

Subgrafo Induzido G1



Exercício 06:

GM

Tenho um grafo não direcionado G, com:

Vértices = 1, 2, 3, 4, 5 e 6

Arestas = 1 para 2, 1 para 5, 2 para 3, 2 para 4, 3 para 4, 3 para 6, 4 para 5, 4 para 6 e 5 para 6

Plote o grafo em um código python.

GM



Agora quero caminhar pelo grafo. Os passeios podem ser de 4 tipos:

1. Passeio (walk) = sequência alternada de vértices e arestas adjacentes que podem se repetir no mesmo passeio.
2. Passeio fechado = é aquele que inicia e termina no mesmo vértice.
3. Caminho (path) = passeio em que os vértices não se repetem.
4. Trilha ou cadeia (trail) = passeio em que as arestas não se repetem.

Quero que gere funções para realizar esses caminhos sobre o grafo G e as plotar de forma que seja sempre diferente.

Código:

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

# Criando o grafo
G = nx.Graph()

# Adicionando vértices
vertices = [1, 2, 3, 4, 5, 6]
G.add_nodes_from(vertices)

# Adicionando arestas
arestas = [(1, 2), (1, 5), (2, 3), (2, 4), (3, 4), (3, 6), (4, 5), (4, 6), (5, 6)]
G.add_edges_from(arestas)

# Função para gerar um passeio aleatório
def generate_walk(graph, length):
    start_vertex = random.choice(list(graph.nodes()))
    walk = [start_vertex]
    for _ in range(length - 1):
```

```

        neighbors = list(graph.neighbors(walk[-1]))
        if neighbors:
            next_vertex = random.choice(neighbors)
            walk.append(next_vertex)
        else:
            break
    return walk

# Função para gerar um caminho aleatório
def generate_path(graph, length):
    start_vertex = random.choice(list(graph.nodes()))
    path = [start_vertex]
    visited_vertices = set(path)
    for _ in range(length - 1):
        neighbors = list(graph.neighbors(path[-1]))
        unvisited_neighbors = [n for n in neighbors if n not in
visited_vertices]
        if unvisited_neighbors:
            next_vertex = random.choice(unvisited_neighbors)
            path.append(next_vertex)
            visited_vertices.add(next_vertex)
        else:
            break
    return path

# Função para gerar uma trilha aleatória
def generate_trail(graph, length):
    edges = list(graph.edges())
    trail = []
    for _ in range(min(length, graph.number_of_edges())):
        edge = random.choice(edges)
        trail.append(edge)
        edges.remove(edge)
    return trail

# Função para plotar um grafo com um caminho específico
def plot_path(graph, path_or_trail, title):
    pos = nx.spring_layout(graph)
    nx.draw(graph, pos, with_labels=True, font_weight='bold',
node_size=1000, node_color='lightblue', font_color='black',
font_size=10, edge_color='gray')

```



```

        if len(path_or_trail) > 0 and isinstance(path_or_trail[0], tuple):
# Verifica se é uma trilha
            for edge in path_or_trail:
                nx.draw_networkx_edges(graph, pos, edgelist=[edge],
edge_color='red', width=2)
            else: # Caso contrário, é um caminho de vértices
                edges = [(path_or_trail[i], path_or_trail[i+1]) for i in
range(len(path_or_trail)-1)]
                nx.draw_networkx_edges(graph, pos, edgelist=edges,
edge_color='red', width=2)

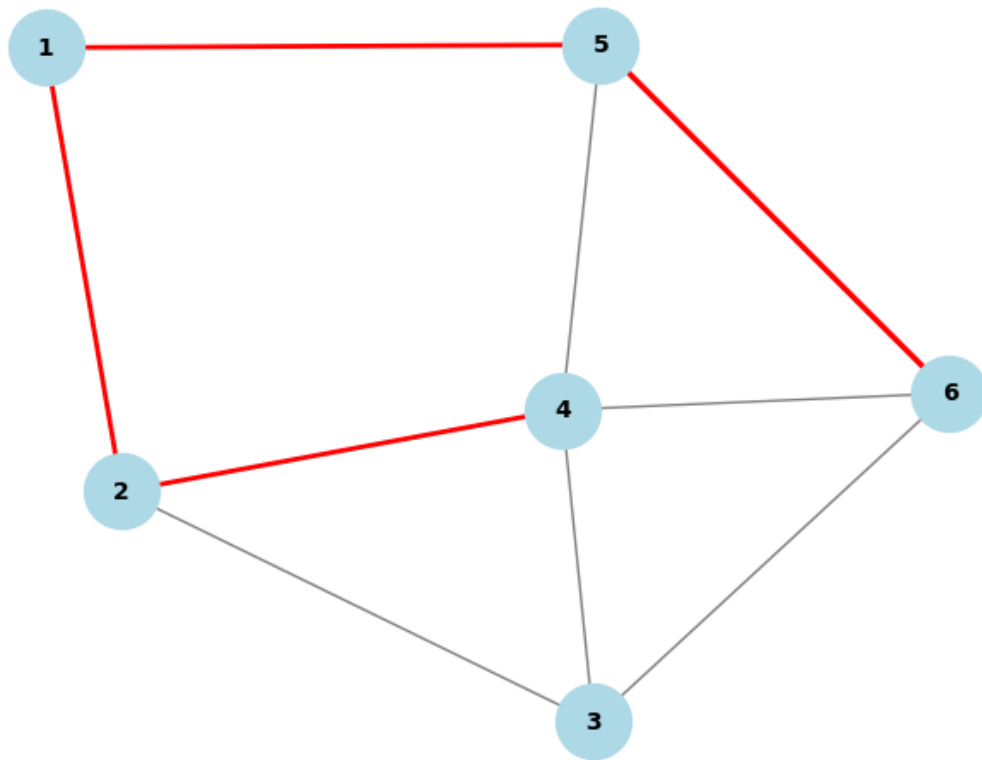
        plt.title(title)
        plt.show()

# Gerando e plotando diferentes tipos de caminhos
random_walk = generate_walk(G, 8)
random_path = generate_path(G, 5)
random_trail = generate_trail(G, 7)

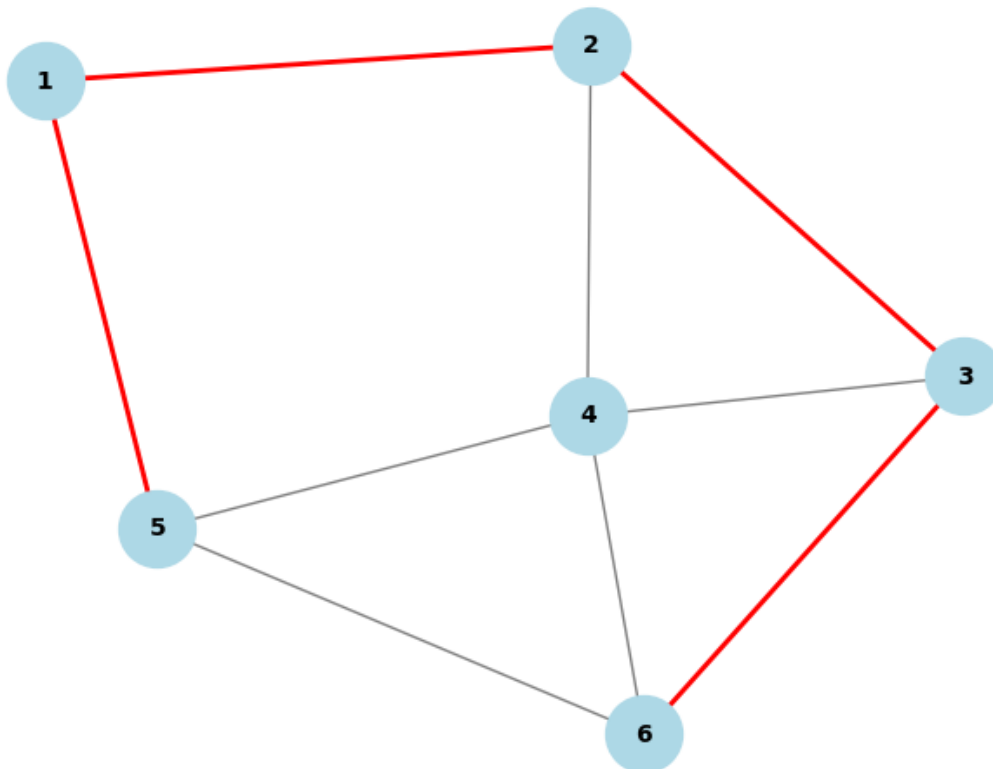
plot_path(G, random_walk, "Walk")
plot_path(G, random_path, "Path")
plot_path(G, random_trail, "Trail")

```

Walk



Path



Trail

