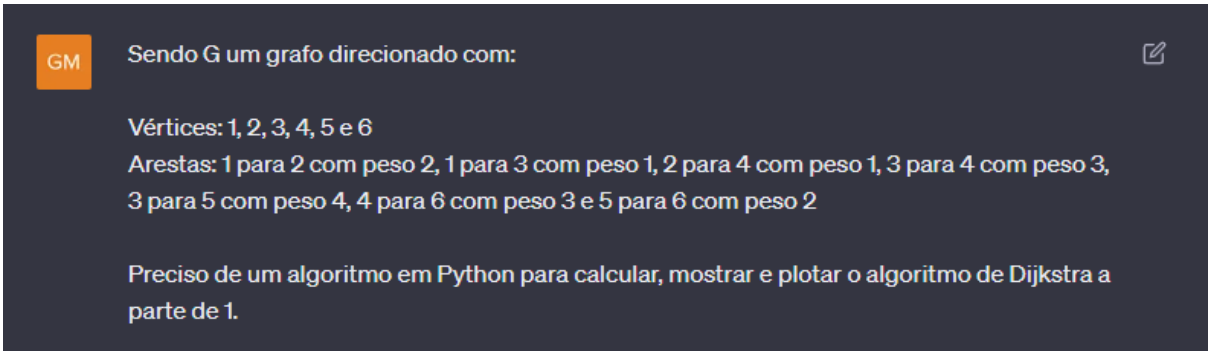
Aula 05 - Caminhos Mínimos e Algoritmos

Nome: Gustavo Silva Malvestiti

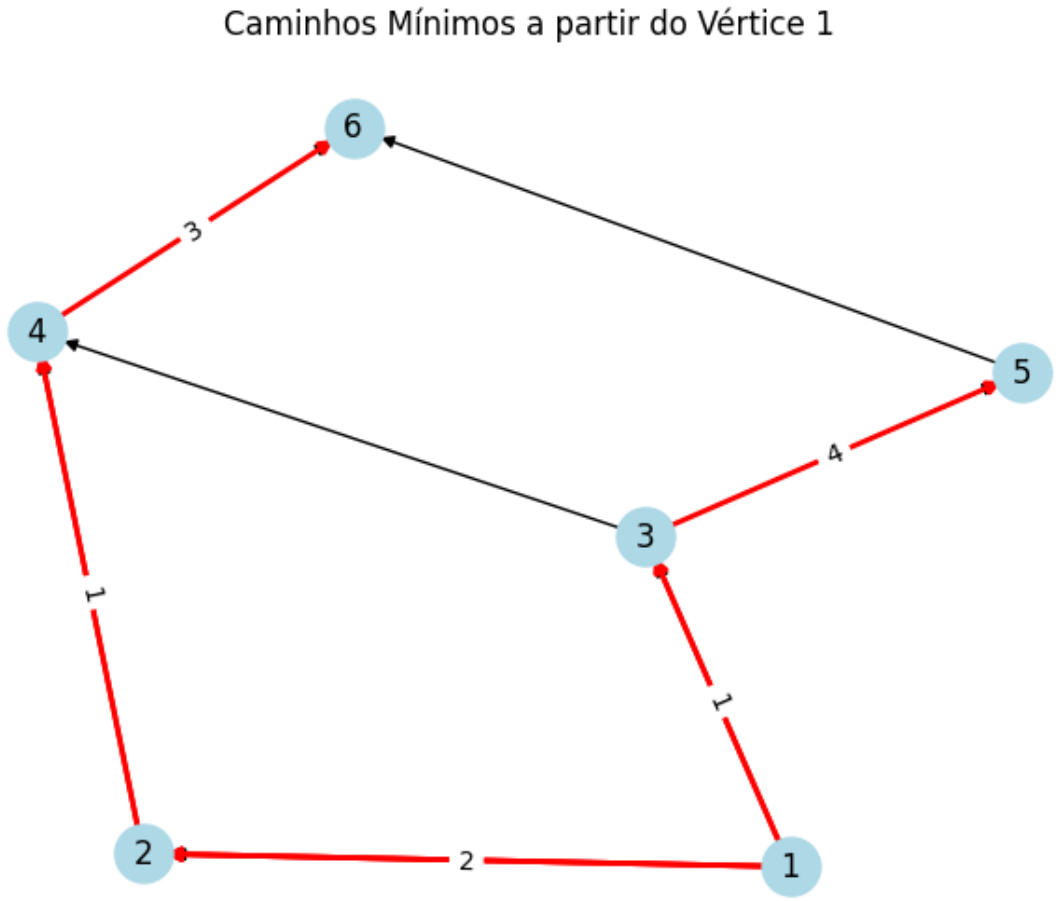
Exercício 01:



Código:

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

# Crie o grafo direcionado
G = nx.DiGraph()

# Adicione os vértices
G.add_nodes_from([1, 2, 3, 4, 5, 6])

# Adicione as arestas com pesos
edges = [(1, 2, 2), (1, 3, 1), (2, 4, 1), (3, 4, 3), (3, 5, 4), (4, 6, 3), (5, 6, 2)]
G.add_weighted_edges_from(edges)

# Calcula os caminhos mínimos a partir do vértice 1 usando o algoritmo de Dijkstra
shortest_paths = nx.single_source_dijkstra(G, source=1)

# Obtém o dicionário de caminhos mínimos
path_lengths, paths = shortest_paths

# Plotar o grafo original
pos = nx.spring_layout(G, seed=42)
nx.draw(G, pos, with_labels=True, node_size=500, node_color='lightblue', font_size=12)

# Plotar os caminhos mínimos em vermelho com os pesos das arestas
for target, path in paths.items():
```

```
    if target != 1:
        edges = [(path[i], path[i + 1]) for i in range(len(path) - 1)]
        edge_labels = {(u, v): G[u][v]['weight'] for u, v in edges}
                    nx.draw_networkx_edges(G,  pos,  edgelist=edges,
edge_color='red', width=2)
        nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels)

# Exibir o gráfico
plt.title("Caminhos Mínimos a partir do Vértice 1")
plt.axis('off')
plt.show()
```



Exercício 02:

GM

Sendo G um grafo direcionado com:

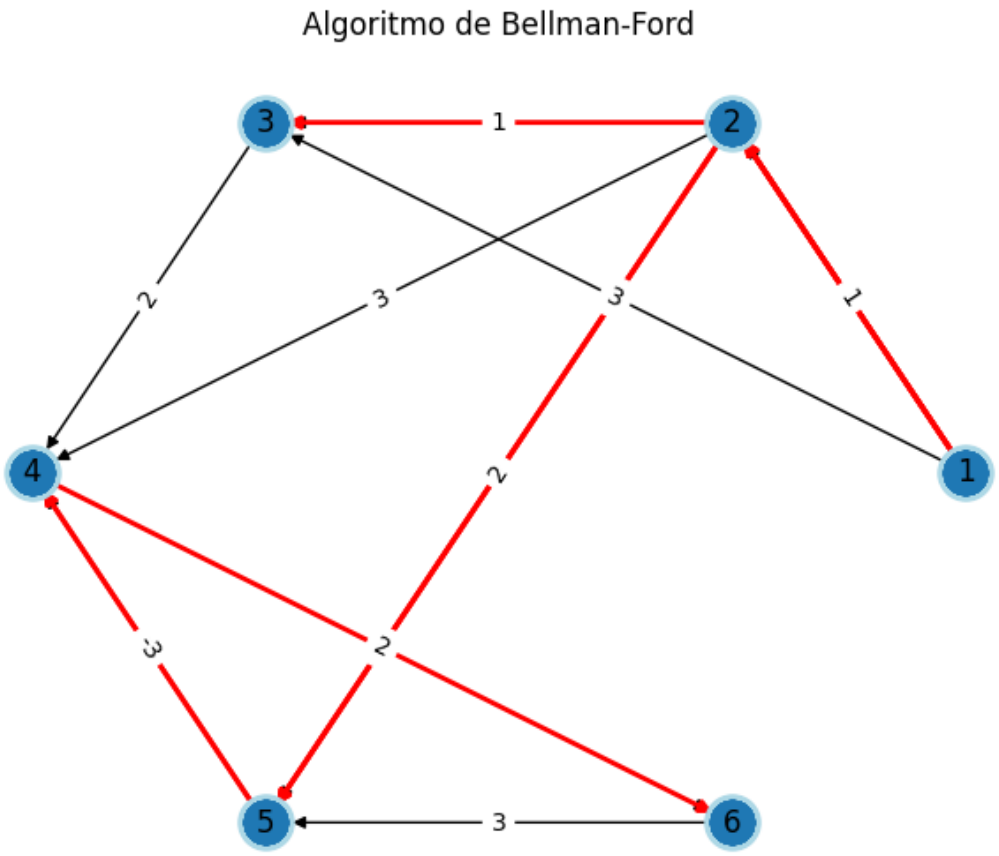
Vértices: 1, 2, 3, 4, 5 e 6
Arestas: 1 para 2 com peso 1, 1 para 3 com peso 3, 2 para 3 com peso 1, 2 para 4 com peso 3, 2 para 5 com peso 2, 3 para 4 com peso 2, 4 para 6 com peso 2, 5 para 4 com peso -3 e 6 para 5 com peso 3

Preciso de um algoritmo de Bellman-Ford parte de 1 em Python para calcular e plotar os caminhos mínimos em vermelho sobre o grafo original.

Código:

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

# Crie o grafo
G = nx.DiGraph()

# Adicione os vértices ao grafo
G.add_nodes_from([1, 2, 3, 4, 5, 6])

# Adicione as arestas com pesos ao grafo
edges_with_weights = [(1, 2, 1), (1, 3, 3), (2, 3, 1), (2, 4, 3), (2,
5, 2),
                      (3, 4, 2), (4, 6, 2), (5, 4, -3), (6, 5, 3)]

G.add_weighted_edges_from(edges_with_weights)

# Execute o algoritmo de Bellman-Ford a partir do nó 1
shortest_paths    =    nx.single_source_bellman_ford_path(G,    source=1,
weight='weight')
shortest_distances    =    nx.single_source_bellman_ford_path_length(G,
source=1, weight='weight')

# Desenhe o grafo com os caminhos mínimos em vermelho
pos = nx.circular_layout(G)
nx.draw(G,          pos,          with_labels=True,          node_size=500,
node_color='lightblue')

# Desenhe as arestas do caminho mínimo em vermelho
for node in G.nodes():
    path = shortest_paths[node]
    for i in range(len(path) - 1):
```

```
        edge = (path[i], path[i+1])
            nx.draw(G, pos, edgelist=[edge], edge_color='red', width=2,
arrows=True)

# Adicione rótulos de peso nas arestas
edge_labels = {(u, v): d['weight'] for u, v, d in G.edges(data=True)}
nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels)

plt.title("Algoritmo de Bellman-Ford")
plt.show()
```



Exercício 03:

GM

Sendo G um grafo direcionado com:

Vértices: 1, 2, 3, 4, 5 e 6
Arestas: 1 para 2 com peso 1, 1 para 3 com peso 3, 2 para 3 com peso 1, 2 para 4 com peso 3, 2 para 5 com peso 2, 3 para 4 com peso 2, 4 para 6 com peso 2, 5 para 4 com peso -3 e 6 para 5 com peso 3

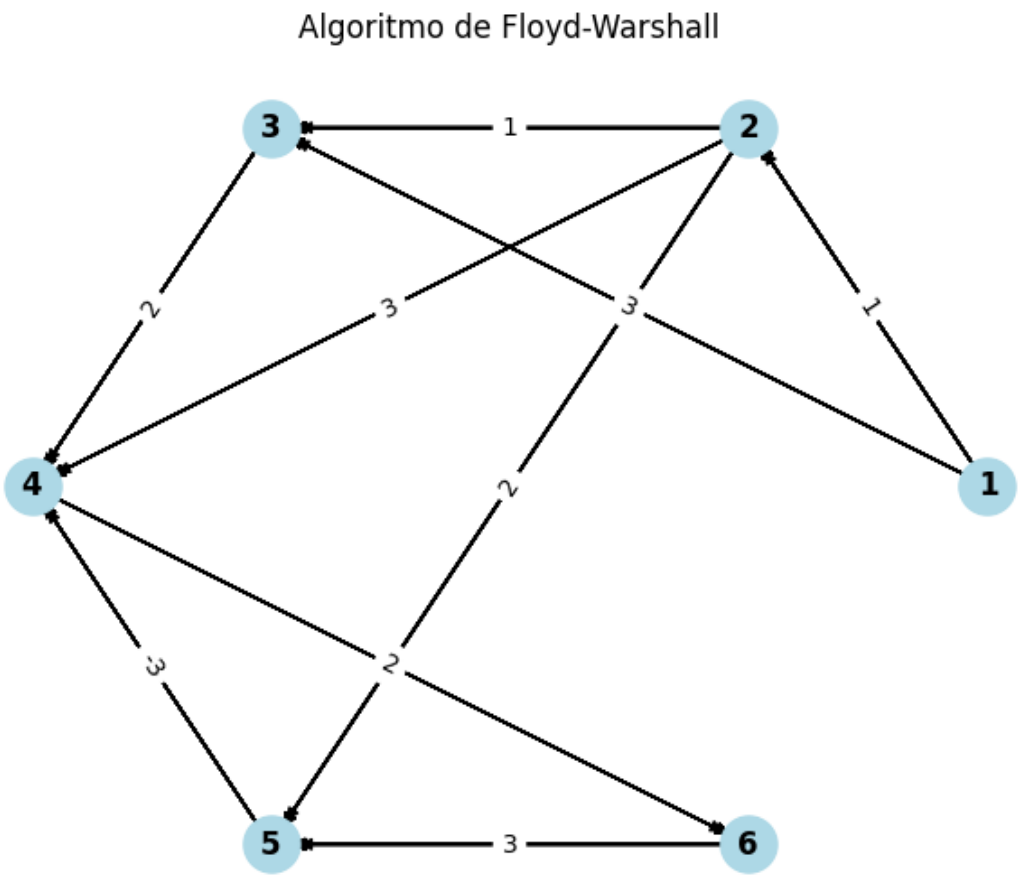
Preciso de um algoritmo de Floyd-Warshall em Python para calcular e plotar os caminhos mínimos em vermelho sobre o grafo original e mostrar a matriz de custo.

Código:

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

# Definir o grafo
G = nx.DiGraph()
G.add_edges_from([(1, 2, {'weight': 1}),
                  (1, 3, {'weight': 3}),
                  (2, 3, {'weight': 1}),
                  (2, 4, {'weight': 3}),
                  (2, 5, {'weight': 2}),
                  (3, 4, {'weight': 2}),
                  (4, 6, {'weight': 2}),
                  (5, 4, {'weight': -3}),
                  (6, 5, {'weight': 3})])

# Calcular as distâncias mínimas usando o algoritmo de Floyd-Warshall
dist_matrix = np.array(nx.floyd_warshall_numpy(G, weight='weight'))

# Plotar o grafo original
pos = nx.circular_layout(G)
nx.draw(G,          pos,          with_labels=True,          node_size=500,
node_color='lightblue', font_size=12, font_weight='bold')

# Plotar as arestas dos caminhos mínimos em vermelho
for i in range(len(G.nodes)):
    for j in range(len(G.nodes)):
        if i != j:
            if dist_matrix[i][j] < float('inf'):
                    path = nx.shortest_path(G, source=i+1, target=j+1,
weight='weight')
```

```
                    edges = [(path[k], path[k+1]) for k in range(len(path)
- 1)]
                        nx.draw_networkx_edges(G, pos, edgelist=G.edges(),
arrowsize=10)

# Configurar os rótulos das arestas com os pesos
labels = nx.get_edge_attributes(G, 'weight')
nx.draw_networkx_edge_labels(G, pos, edge_labels=labels)

# Mostrar a matriz de custo
print("Matriz de custo:")
print(dist_matrix)

# Exibir o gráfico
plt.title("Algoritmo de Floyd-Warshall")
plt.axis('off')
plt.show()
```

```
Matriz de custo:
[[ 0.  1.  2.  0.  3.  2.]
 [inf  0.  1. -1.  2.  1.]
 [inf inf  0.  2.  7.  4.]
 [inf inf inf  0.  5.  2.]
 [inf inf inf -3.  0. -1.]
 [inf inf inf  0.  3.  0.]]
```



Exercício 04:

GM

Sendo G um grafo direcionado com:

Vértices: s, 1, 2, 3, 4 e t
Arestas: s para 1 com peso 1, 1 para 2 com peso 2, 1 para 3 com peso 1, 2 para 4 com peso 5, 3 para 4 com peso 3, 3 para t com peso 4, 4 para t com peso 2, s para t com peso 10

Preciso de um algoritmo A* em Python para calcular e plotar caminho mínimo em vermelho sobre o grafo original com título de s para t.

Código:

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

def a_star(graph, start, end):
    open_list = [(0, start)]
    closed_set = set()
    g_score = {vertex: float('inf') for vertex in graph.nodes}
    g_score[start] = 0
    f_score = {vertex: float('inf') for vertex in graph.nodes}
    f_score[start] = heuristic(start, end)

    while open_list:
        _, current = heapq.heappop(open_list)

        if current == end:
            path = reconstruct_path(came_from, current)
            return path

        closed_set.add(current)

        for neighbor in graph.neighbors(current):
            if neighbor in closed_set:
                continue

            edge_weight = graph[current][neighbor]['weight']
            tentative_g_score = g_score[current] + edge_weight

            if tentative_g_score < g_score[neighbor]:
                came_from[neighbor] = current
                g_score[neighbor] = tentative_g_score
```

```
                                f_score[neighbor] = g_score[neighbor] +
heuristic(neighbor, end)
                            heapq.heappush(open_list, (f_score[neighbor],
neighbor))

    return None

def heuristic(node, end):
      # Neste exemplo, usaremos uma heurística simples - a distância
euclidiana
     return 0  # Você pode implementar uma heurística mais sofisticada
aqui

def reconstruct_path(came_from, current):
    path = [current]
    while current in came_from:
        current = came_from[current]
        path.append(current)
    path.reverse()
    return path

# Criando o grafo
G = nx.DiGraph()

G.add_edge('s', 1, weight=1)
G.add_edge(1, 2, weight=2)
G.add_edge(1, 3, weight=1)
G.add_edge(2, 4, weight=5)
G.add_edge(3, 4, weight=3)
G.add_edge(3, 't', weight=4)
G.add_edge(4, 't', weight=2)
G.add_edge('s', 't', weight=10)

# Encontrando o caminho mínimo
came_from = {}
path = a_star(G, 's', 't')

if path:
    # Plotando o grafo
    pos = nx.circular_layout(G)
    nx.draw(G, pos, with_labels=True, node_color='lightblue')

    # Destacando o caminho mínimo em vermelho
```

```
    path_edges = [(path[i], path[i + 1]) for i in range(len(path) - 1)]
    edge_colors = ['red' for edge in G.edges]
        nx.draw(G,  pos,  edgelist=path_edges,  edge_color=edge_colors,
with_labels=True)

    # Definindo o título
    plt.title('Caminho mínimo de s para t')

    # Exibindo o gráfico
    plt.show()
else:
    print("Caminho não encontrado")
```