DIJKSTRA PARA O CÁLCULO DE ROTAS

ESCOLA DE ENGENHARIA ELÉTRICA, MECÂNICA E DE COMPUTAÇÃO

ESTRUTURA DE DADOS 1 – BRUNA, ALDO

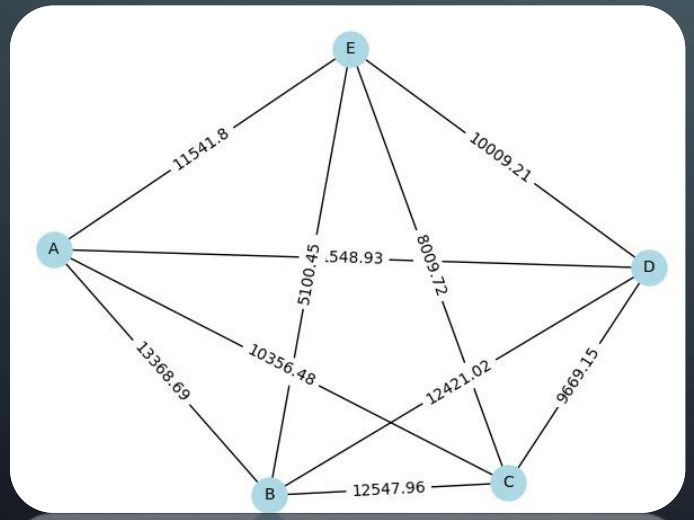
GHUSTAVO BARBOSA FERNANDES – 201703673

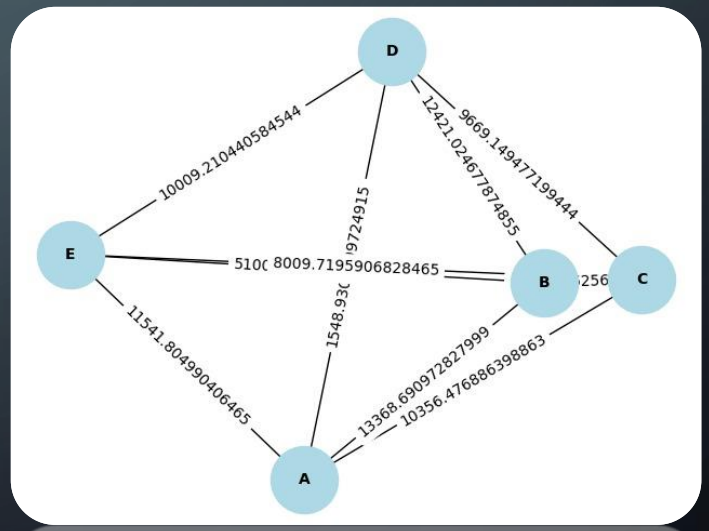
WALLYSON MIRANDA AGUIAR - 202201677

YURI CASSIANO MATSUOKA - 202302595

MOTIVAÇÃO:

CÁLCULO DO CAMINHO MAIS CURTO EM GRAFOS PONDERADOS. O PROBLEMA É COMUM EM ROTEAMENTO, LOGÍSTICA E REDES DE COMUNICAÇÃO. A SOLUÇÃO PROPOSTA UTILIZA O ALGORITMO DE DIJKSTRA, EXPLORANDO SUAS APLICAÇÕES EM SISTEMAS DE NAVEGAÇÃO GPS, OTIMIZAÇÃO DE ROTAS EM REDES DE COMPUTADORES E PLANEJAMENTO LOGÍSTICO.





DATASET E METODOLOGIA:

1. Dataset:

    1.1 Grafos Sintéticos

    1.2 Dados reais

2. Implementação Empregada:

    2.1 Clássica, com uso de PriorityQueue simples

3. Ambiente de Produção:

    3.1 Máquina com especificações padrão

    3.2 Linguagem de Programação: Python

ESTUDO:

- Validação de acurácia
- Analises Computacionais:
  - ✓ Tempo de execução
  - ✓ Gasto energético
  - ✓ Consumo de espaço
- Comparativo com o principal concorrente

IMPLEMENTAÇÃO:

- Bibliotecas:

```
1    import heapq
2    import networkx as nx
3    import matplotlib.pyplot as plt
4    from geopy.distance import geodesic
5    from geopy.geocoders import Nominatim
6    import itertools
```

- Funções Preliminares:

```
7
8    def addEdge(graph, u, v, weight):
9        if u not in graph:
10           graph[u] = {}
11       if v not in graph:
12           graph[v] = {}
13       if v not in graph[u]:
14           graph[u][v] = weight
15       if u not in graph[v]:
16           graph[v][u] = weight
17
18   def calcDistance(coord_a, coord_b):
19       coord_a = tuple(map(float, coord_a.split(',')))
20       coord_b = tuple(map(float, coord_b.split(',')))
21       return geodesic(coord_a, coord_b).km
22
```

IMPLEMENTAÇÃO:

- Dijkstra propriamente dito:

```python
def dijkstra(graph, start, end):
    priority_queue = []
    distances = {vertex: float('infinity') for vertex in graph}
    previous_vertices = {vertex: None for vertex in graph}
    distances[start] = 0
    heapq.heappush(priority_queue, (0, start))

    while priority_queue:
        current_distance, current_vertex = heapq.heappop(priority_queue)

        if current_vertex == end:
            break

        if current_distance > distances[current_vertex]:
            continue

        for neighbor, weight in graph[current_vertex].items():
            distance = current_distance + weight
            if distance < distances[neighbor]:
                distances[neighbor] = distance
                previous_vertices[neighbor] = current_vertex
                heapq.heappush(priority_queue, (distance, neighbor))

    path = []
    current_vertex = end
    while current_vertex is not None:
        path.append(current_vertex)
        current_vertex = previous_vertices[current_vertex]
    path.reverse()

    return distances[end], path
```

IMPLEMENTAÇÃO:

- Funções auxiliares à execução:

```
54
55  def fetchCoordinatesFromCEP(cep):
56      geolocator = Nominatim(user_agent="dijkstra_app")
57      try:
58          cep = cep.replace('-', '')
59          location = geolocator.geocode(cep)
60          if location:
61              return f"{location.latitude},{location.longitude}"
62          else:
63              print(f"Erro: Não foi possível encontrar o CEP '{cep}'.")
64              return None
65      except Exception as e:
66          print(f"Erro ao buscar o CEP '{cep}': {e}")
67          return None
68
```

IMPLEMENTAÇÃO:

- Execução geral:

```python
def plotGraph(graph, path):
    G = nx.Graph()

    for u in graph:
        for v, weight in graph[u].items():
            G.add_edge(u, v, weight=weight)

    pos = nx.spring_layout(G)

    nx.draw(G, pos, with_labels=True, node_size=2000, node_color='lightblue', font_size=10, font_weight='bold')

    weights = nx.get_edge_attributes(G, 'weight')
    edge_labels = {edge: f"{weights[edge]:.2f}" for edge in G.edges()}

    path_edges = list(zip(path, path[1:]))
    nx.draw_networkx_edges(G, pos, edgelist=path_edges, edge_color='red', width=3, alpha=0.7)
    nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels)

    plt.show()

print("Bem-vindo ao sistema de cálculo de distâncias!")
num_places = int(input("Quantos locais você deseja adicionar? "))

places = []
```

IMPLEMENTAÇÃO:

- Execução geral:

```
93
94      for i in range(num_places):
95          name = input(f"Digite o nome do local {i + 1}: ")
96          cep = input(f"Digite o CEP do local '{name}': ")
97          coordinates = fetchCoordinatesFromCEP(cep)
98          if coordinates:
99              places.append((name, coordinates))
100         else:
101             print(f"CEP '{cep}' inválido. Tente novamente.")
102
103     if len(places) < 2:
104         print("É necessário adicionar pelo menos dois locais para calcular distâncias.")
105         exit()
106
107     print("\nLocais adicionados:")
108     for place in places:
109         print(place)
110
111     print("\nCalculando distâncias entre os locais...")
112     combinations = itertools.combinations(places, 2)
113     data = []
114     for place1, place2 in combinations:
115         u, coord_u = place1
116         v, coord_v = place2
117         weight = calcDistance(coord_u, coord_v)
118         data.append((u, v, weight))
119         print(f"Distância de {u} a {v}: {weight:.2f} km")
120
```

IMPLEMENTAÇÃO:

- Execução geral:

```
121     graph = {}
122     for u, v, weight in data:
123         addEdge(graph, u, v, weight)
124 
125     start_vertex = input("\nDigite o nome do local de partida: ")
126     end_vertex = input("Digite o nome do local de destino: ")
127 
128     if start_vertex not in graph or end_vertex not in graph:
129         print(f"Erro: Os nós '{start_vertex}' ou '{end_vertex}' não estão no grafo.")
130     else:
131         distance, path = dijkstra(graph, start_vertex, end_vertex)
132 
133         print(f"\nA menor distância de {start_vertex} a {end_vertex} é {distance:.2f} km")
134         print(f"O caminho é: {' -> '.join(path)}")
135 
136         plotGraph(graph, path)
```

RESULTADOS:

- ✓ Verificado acurácia do algoritmo e sua implementação inicialmente com dados inseridos em primeira instância, e após com dados obtidos com a solicitação de entrada do usuário e a utilização da API escolhida.
- ✓ Analisado o tempo de execução assim como seu gasto energético e de memória, posteriormente comparado com resultados de seu concorrente mais direto o algoritmo de Bellman-Ford, provando (nessas circunstâncias específicas) sua superioridade olhando a relação escalabilidade/tempo-de-execução.

CONCLUSÃO

- Neste trabalho de estudo e anállise qualitativa e quantitativa do algoritimo de Dijkstra é possível concluir que levando em consideração condições específicas de dataset, não há opção melhor para sua função de cálculo e escolha de melhor (menor) rota entre dois vértices de um grafo. Foi observado no entanto que a implementação pode e deve ser otimizada e já é possível ver o caminho a seguir. O algoritimo de Bellman-Ford citado é uma outra opção direta e também viável para essa solução mas, mesmo sendo mais geral e seu tratamento, é também mais gastoso. Dijkstra já é extremamente utilizado para suas devidas finalidades e ainda pode ser vinculado à outras.

BIOGRAFIA E LITERATURA

- Introduction to Algorithms (Cormen et al.)
- Artigo acadêmico: Applications of Dijkstra's Algorithm in Network Routing
- Understanding Dijkstra's Algorithm - https://www.youtube.com/watch?v=qp7wOb-KLn4
- W3 Schools - https://www.w3schools.com/dsa/dsa_algo_graphs_dijkstra.ph
- Dijkstra's algorithm weighted by travel time in OSM networks | Bryan R. Vallej | Towards Data Science https://towardsdatascience.com/dijkstras-algorithm-weighted-by-travel-time-in-osm-networks-792aa92e03af
- Directions-Generator | @mitchelltmarino | Github https://github.com/mitchelltmarino/
- Directions-Generator- Shortest Path Algorithms Explained (Dijkstra's & Bellman-Ford) | b001 | YouTube https://youtu.be/j0OUwduDOS0
- Let's Show #186 - Python Tutorial - Shortest Route | Dijkstra Algorithm #2 | Event Handler | YouTube https://youtu.be/fayie7eGBlc
- Implementing the Dijkstra Algorithm in Python: A Step-by-Step Tutorial | Bex Tuychiyev | DataCamp https://www.datacamp.com/tutorial/dijkstra-algorithm-in-python
- DSA Dijkstra's Algorithm | W3 Schools https://www.w3schools.com/dsa/dsa_algo_graphs_dijkstra.php
- O Algoritmo de Dijkstra em Python: Encontrando o caminho mais curto | Carlos Herrera | LinkedIn https://www.linkedin.com/pulse/o-algoritmo-de-dijkstra-encontrando-caminho-mais-curto-carlos-Herrera
- 7.20. Dijkstra's Algorithm | Panda | IME-USP https://panda.ime.usp.br/panda/static/pythonds_pt/07-Grafos/DijkstrasAlgorithm.html
- Implementing Dijkstra's Algorithm with a Priority Queue | Mary Elaine Califf | YouTube https://youtu.be/CerlT7tTZfY
- Dijkstra em C | @heltonricardo | GitHub https://github.com/heltonricardo/dijkstra-c/
- Dijkstra's Algorithm | Doug Mahugh | Doug's World https://www.dougmahugh.com/dijkstra/
- Implementing Dijkstra's Algorithm in Python | Alexey Klochay | Udacity https://www.udacity.com/blog/2021/10/implementing-dijkstras-algorithm-in-python.html
- [7.5] Dijkstra Shortest Path Algorithm in Python | ThinkX Academy | YouTube https://youtu.be/OrJ004Wid4o
- Dijkstra's Algorithm in Python : Finding The Shortest Path | Azka Redhia | Medium https://medium.com/@azkardm/dijkstras-algorithm-in-python-finding-the-shortest-path-bcb3bcd4a4ea
- Introduction to Priority Queues in Python | BuiltIn https://builtin.com/data-science/priority-queues-in-python

GRATOS PELA ATENÇÃO