ALEXANDRE DIAMANTE

FUNDAMENTOS DE ESTRUTURAS DE DADOS

3 – GRAFOS E SUAS OPERAÇÕES

Algoritmos para grafos em Python

Trabalho apresentado para Atividade Prática na disciplina ESTRUTURA DE DADOS, do curso de CIÊNCIA DE DADOS, turno EAD da Instituição de Ensino Ampli Anhanguera.

Umuarama-PR

2024

```
import heapq

class Grafo:
    def __init__(self):
        self.vertices = {}

    def adicionar_vertice(self, vertice):
        if vertice not in self.vertices:
            self.vertices[vertice] = []

    def adicionar_aresta(self, origem, destino, peso):
        if origem in self.vertices and destino in self.vertices:
            self.vertices[origem].append((destino, peso))

    def dijkstra(self, inicio, fim):
        distancias = {vertice: float('inf') for vertice in self.vertices}
        distancias[inicio] = 0

        caminho = {vertice: None for vertice in self.vertices}
        fila_prioridade = [(0, inicio)]

        while fila_prioridade:
            distancia_atual, vertice_atual = heapq.heappop(fila_prioridade)

            if distancia_atual > distancias[vertice_atual]:
                continue  # Ignore entradas obsoletas

            if vertice_atual == fim:
                break

            for vizinho, peso in self.vertices[vertice_atual]:
                distancia = distancia_atual + peso

                if distancia < distancias[vizinho]:
                    distancias[vizinho] = distancia
                    caminho[vizinho] = vertice_atual
                    heapq.heappush(fila_prioridade, (distancia, vizinho))

        return self.reconstruir_caminho(caminho, inicio, fim), distancias[fim]

    def reconstruir_caminho(self, caminho, inicio, fim):
        caminho_final = []
        atual = fim

        while atual is not None:  # Verifica se o atual é None
            caminho_final.append(atual)
            atual = caminho[atual]

        caminho_final.reverse()
```

```
if caminho_final[0] == inicio:
            return caminho_final
        return []  # Retorna lista vazia se não encontrar o caminho

def main():
    grafo = Grafo()

    for vertice in ['A', 'B', 'C', 'D', 'E']:
        grafo.adicionar_vertice(vertice)

    grafo.adicionar_aresta('A', 'B', 4)
    grafo.adicionar_aresta('A', 'C', 2)
    grafo.adicionar_aresta('B', 'C', 5)
    grafo.adicionar_aresta('B', 'D', 10)
    grafo.adicionar_aresta('C', 'E', 3)
    grafo.adicionar_aresta('D', 'E', 4)
    grafo.adicionar_aresta('E', 'A', 7)

    ponto_partida = input("Digite o ponto de partida: ").strip().upper()
    ponto_chegada = input("Digite o ponto de chegada: ").strip().upper()

    if ponto_partida not in grafo.vertices or ponto_chegada not in grafo.vertices:
        print("Pontos inválidos. Tente novamente.")
        return

    caminho, custo_total = grafo.dijkstra(ponto_partida, ponto_chegada)

    if caminho:
        print(f"O caminho mais curto de {ponto_partida} para {ponto_chegada} é: {' ->
'.join(caminho)} com um custo total de {custo_total}")
    else:
        print(f"Não há caminho disponível de {ponto_partida} para {ponto_chegada}.")

if __name__ == "__main__":
    main()


# COMO TESTAR ->>
# Digite o ponto de partida:
# A
# Digite o ponto de chegada:
# E
# O caminho mais curto de A para E é: A -> C -> E com um custo total de 5
```

Publicado no Git:
https://github.com/alexandrediamante/cienciadedados.git