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>