

Tecnólogo Ciência de Dados

Guilherme Giacomini Teixeira

Algoritmos para grafos em python:

Trabalho de Avaliação da Unidade 3 da Disciplina Estrutura de Dados

Balneário Camboriú - SC

2025

Balneário Camboriú - SC

2025

Guilherme Giacomini Teixeira

Algoritmos para grafos em python:

Trabalho de Avaliação da Unidade 3 da Disciplina Estrutura de Dados

­­­­

Trabalho de avaliação da unidade 3 da disciplina Estrutura de Dados apresentado como requisito parcial para a obtenção da média no curso Ciência de Dados.

Professora: Vanessa Matias Leite  
Tutor: João Henrique Correia dos Santos

SUMÁRIO

[1 INTRODUÇÃO 3](#_Toc129809424)

[2 DESENVOLVIMENTO 4](#_Toc129809425)

[3 RESULTADOS](#_Toc129809426) 6

[4 CONCLUSÃO](#_Toc129809427) 7

[5 REFERÊNCIAS](#_Toc129809428) 8

# INTRODUÇÃO

Este relatório apresenta a atividade solicitada na disciplina de **Estrutura de Dados**, realizada na Unidade 3, Aula 4, do curso de **Ciência de Dados**. O objetivo da atividade foi implementar um algoritmo de caminho mínimo para grafos usando a linguagem de programação Python.

O projeto consiste em desenvolver uma aplicação para uma empresa de logística, capaz de calcular a rota mais rápida entre dois pontos de entrega, considerando as diferentes distâncias entre eles. Para isso, é necessário implementar uma classe Grafo que contenha métodos para adicionar vértices e arestas com pesos. O grafo deve ser representado por listas de adjacência, e a classe precisa de um método para encontrar o caminho mais curto usando o algoritmo de Dijkstra.

Este relatório detalhará os passos seguidos para o desenvolvimento da aplicação, incluindo a configuração do ambiente no VS Code e a implementação do código para resolver o problema proposto.

# DESENVOLVIMENTO

A linguagem Python é uma escolha popular e versátil para a área de Ciência de Dados, em grande parte devido à sua sintaxe clara e legível. É uma linguagem de programação de alto nível, interpretada e multiparadigma, que oferece suporte a programação orientada a objetos, funcional e procedural. Sua tipagem dinâmica e forte, juntamente com a vasta quantidade de bibliotecas e frameworks disponíveis, a tornam ideal para diversas aplicações, incluindo desenvolvimento web, automação, análise de dados e aprendizado de máquina.

No contexto deste projeto, o Python foi a linguagem escolhida para implementar um algoritmo de caminho mínimo para grafos. A sua eficiência em lidar com estruturas de dados complexas, como a representação de grafos por meio de listas de adjacência, justifica plenamente sua utilização para resolver o problema de encontrar a rota mais rápida em uma rede logística.

Para a implementação do projeto, a ferramenta VS Code foi utilizada para desenvolver o programa. A atividade consistiu em criar uma aplicação que auxiliasse no mapeamento de rotas para uma nova empresa de logística, calculando a rota mais rápida entre dois pontos de entrega.

O projeto incluiu a criação de uma classe Grafo com métodos para adicionar vértices e arestas com pesos. O grafo foi representado usando listas de adjacência. O componente central da solução foi a implementação do algoritmo de Dijkstra para encontrar o caminho mais curto e o custo total entre um ponto de partida e um ponto de chegada fornecidos pelo usuário.

A seguir, está o código no VS Code com a implementação da solução em Python:

import heapq

# Classe para representar o Grafo

class Grafo:

def \_\_init\_\_(self):

# O grafo é um dicionário onde as chaves são os vértices e os valores são dicionários de vizinhos

# {vertice: {vizinho: peso}}

self.grafo = {}

def adicionar\_vertice(self, vertice):

# Adiciona um novo vértice ao grafo se ele ainda não existir

if vertice not in self.grafo:

self.grafo[vertice] = {}

def adicionar\_aresta(self, origem, destino, peso):

# Adiciona aresta com peso. Se os vértices não existirem, eles são criados.

self.adicionar\_vertice(origem)

self.adicionar\_vertice(destino)

# Aresta direcionada: origem -> destino

self.grafo[origem][destino] = peso

def dijkstra(self, inicio, fim):

# Inicializa as distâncias de todos os vértices como infinito e a do início como 0

distancias = {vertice: float('inf') for vertice in self.grafo}

distancias[inicio] = 0

# Dicionário para rastrear o caminho mais curto

caminhos = {vertice: [] for vertice in self.grafo}

caminhos[inicio] = [inicio]

# Fila de prioridade (min-heap) para os vértices a serem visitados.

# Armazena tuplas (distância, vértice)

fila\_prioridade = [(0, inicio)]

while fila\_prioridade:

distancia\_atual, vertice\_atual = heapq.heappop(fila\_prioridade)

# Se a distância atual for maior que a registrada, continue (já encontramos um caminho mais curto)

if distancia\_atual > distancias[vertice\_atual]:

continue

# Se chegamos ao destino, o caminho mais curto foi encontrado

if vertice\_atual == fim:

return caminhos[fim], distancias[fim]

# Explora os vizinhos do vértice atual

for vizinho, peso in self.grafo[vertice\_atual].items():

distancia = distancia\_atual + peso

# Se um caminho mais curto for encontrado, atualiza a distância e o caminho

if distancia < distancias[vizinho]:

distancias[vizinho] = distancia

caminhos[vizinho] = caminhos[vertice\_atual] + [vizinho]

heapq.heappush(fila\_prioridade, (distancia, vizinho))

# Se o destino não for alcançável a partir do início, retorna None

return None, float('inf')

# Função principal para executar o programa

def main():

# Instancia o grafo

grafo\_logistica = Grafo()

# Adiciona os vértices e arestas com os pesos, conforme o exemplo do problema.

# Vértices: A, B, C, D, E

# Arestas: A->B(4), A->C(2), B->C(5), B->D(10), C->E(3), D->E(4), E->A(7)

grafo\_logistica.adicionar\_aresta('A', 'B', 4)

grafo\_logistica.adicionar\_aresta('A', 'C', 2)

grafo\_logistica.adicionar\_aresta('B', 'C', 5)

grafo\_logistica.adicionar\_aresta('B', 'D', 10)

grafo\_logistica.adicionar\_aresta('C', 'E', 3)

grafo\_logistica.adicionar\_aresta('D', 'E', 4)

grafo\_logistica.adicionar\_aresta('E', 'A', 7)

# Recebe do usuário os pontos de partida e chegada

ponto\_partida = input("Digite o ponto de partida (ex: A): ").upper()

ponto\_chegada = input("Digite o ponto de chegada (ex: E): ").upper()

# Valida se os pontos de partida e chegada existem no grafo

if ponto\_partida not in grafo\_logistica.grafo or ponto\_chegada not in grafo\_logistica.grafo:

print("Um ou ambos os pontos não existem no grafo. Por favor, verifique e tente novamente.")

return

# Utiliza o algoritmo de Dijkstra para encontrar o caminho mais curto

caminho, custo = grafo\_logistica.dijkstra(ponto\_partida, ponto\_chegada)

# Imprime o resultado

if caminho:

print(f"O caminho mais curto é: {' -> '.join(caminho)} com custo total de {custo}.")

else:

print(f"Não há caminho do ponto {ponto\_partida} para o ponto {ponto\_chegada}.")

# Executa a função principal

if \_\_name\_\_ == "\_\_main\_\_":

main()

# RESULTADOS:

A atividade proposta, que envolveu a implementação de um algoritmo de caminho mínimo para grafos, permitiu a aplicação prática de conceitos teóricos de Estrutura de Dados. O objetivo foi desenvolver uma solução em Python capaz de calcular a rota mais rápida entre dois pontos de entrega para uma empresa de logística, utilizando o algoritmo de Dijkstra.

A implementação, conforme o código desenvolvido, demonstrou ser eficaz na representação de um grafo por meio de listas de adjacência. A classe Grafo e o algoritmo de Dijkstra foram capazes de receber um conjunto de dados estático, processar os vértices e arestas com seus respectivos pesos e, a partir dos pontos de partida e chegada informados pelo usuário, determinar o caminho mais curto e o seu custo total.

A solução validou o exemplo fornecido na atividade , onde, para um grafo com os vértices A, B, C, D, E e arestas com pesos específicos, a rota mais curta de "A" para "E" foi corretamente identificada como "A -> C -> E", com um custo total de 5. Em conclusão, o projeto cumpriu todos os requisitos, demonstrando a capacidade do aluno em aplicar os conceitos de grafos e o algoritmo de Dijkstra para resolver um problema de otimização de rotas de forma eficiente.

# CONCLUSÃO

A atividade prática proporcionou a oportunidade de aplicar os conceitos teóricos de grafos e o algoritmo de Dijkstra, aprendidos na disciplina de Estrutura de Dados. A implementação de um sistema de mapeamento de rotas para uma empresa de logística, feita no ambiente de desenvolvimento VS Code, demonstrou a importância de algoritmos de caminho mínimo para a otimização de processos e a tomada de decisões.

O projeto foi bem-sucedido, com a criação de um código funcional que atendeu a todos os requisitos propostos. A implementação da classe Grafo, a representação dos dados por listas de adjacência e a aplicação do algoritmo de Dijkstra para encontrar o caminho mais curto confirmaram a eficácia dessas estruturas de dados e algoritmos para a organização e manipulação de informações de forma eficiente.

Em conclusão, este trabalho reforçou o aprendizado sobre grafos e sua aplicação em problemas práticos, mostrando como estruturas de dados eficientes são cruciais para o desenvolvimento de softwares robustos e de alto desempenho na área de Ciência de Dados.

# REFERÊNCIAS

ALVES, W. P. Programação **Python: aprenda de forma rápida**. São Paulo: Expressa, 2021.

BORIN, V. P.**Estrutura de dados**. 1. ed. São Paulo: Contentus, 2020.

LAMBERT, K. A. **Fundamentos de Python**: estruturas de dados. São Paulo: Cengage Learning, 2022.

SZWARCFITER, J. L.; MARKENZON, L. **Estruturas de dados e seus algoritmos**. 3. ed. Rio de Janeiro: LTC, 2020.

TAKENAKA, R. M. **Introdução a grafos**. Estrutura de Dados, 2021. Disponível em: http://bit.ly/497p4MU. Acesso em: 6 fev. 2024.

LEITE, V. M. **Algoritmos para grafos em Python**. Aula 4. In: UNOPAR ANHANGUERA. Estrutura de Dados. [Material de curso]. Unidade 3, Grafos e suas Operações. [S. l.]: Anhanguera Unopar, 2025. Acesso restrito.