# AEDSIII - Algoritmos de Caminho Mínimo

**Caio D.Alves**[1,2]

[1]Instituto de Ciências Exatas e Aplicadas – Universidade Federal de Ouro Preto (UFOP)
CEP 35931-008 – João Monlevade – MG – Brasil

[2]Departamento de Computação e Sistemas (DECSI)
Universidade Federal de Ouro Preto (UFOP) – João Monlevade, MG – Brasil

{caio}@caio.damasceno@aluno.ufop.edu.br

## 1. Introdução

Os algoritmos de caminho mínimo são fundamentais em teoria dos grafos e têm uma vasta gama de aplicações práticas. Este relatório aborda a implementação dos algoritmos de Dijkstra, Bellman-Ford e Floyd-Warshall, que foram primeiramente estudados e compreendidos por meio do material didático utilizado em sala de aula, conforme apresentado em [de Ouro Preto 2024].

O exercício proposto envolveu a implementação desses algoritmos em Python, aplicando-os a um grafo específico que também foi utilizado durante as aulas. O grafo é representado em formato JSON, e o objetivo do trabalho é a aplicabilidade dos algoritmos na resolução do problema do caminho mínimo.

Neste relatório, além de descrever a implementação de cada algoritmo, faremos uma análise comparativa das suas características e complexidades, destacando as situações em que cada um deles se mostra mais eficiente. A ferramenta de modelos de linguagem, como o ChatGPT, foi utilizada para ajustar e otimizar os códigos desenvolvidos. Esses modelos ajudaram a refinar a implementação e garantir que os algoritmos fossem ajustados às necessidades específicas do projeto, proporcionando sugestões valiosas e corrigindo detalhes técnicos.

O código fonte completo para os algoritmos discutidos pode ser acessado no GitHub através do seguinte link: Acessar o código fonte no GitHub.

## 2. Algoritmo de Dijkstra

### 2.1. Descrição

O algoritmo de Dijkstra é uma técnica eficiente para encontrar o caminho mais curto de um nó de origem a todos os outros nós em um grafo com arestas de peso não negativo. Ele opera de forma gulosa, selecionando o nó com a menor distância acumulada e explorando seus vizinhos, até que todos os caminhos mais curtos tenham sido determinados.

Inicialmente, este algoritmo foi aprendido e compreendido em sala de aula através do material didático [de Ouro Preto 2024]. A implementação em Python foi inspirada em recursos adicionais, incluindo o artigo da DIO [DIO 2024], que fornece uma base prática para a aplicação deste algoritmo em diferentes cenários.

### 2.2. Complexidade

A implementação clássica do algoritmo de Dijkstra, utilizando uma lista não ordenada, possui uma complexidade de tempo de $O(V^2)$, onde $V$ é o número de vértices no grafo.

No entanto, ao utilizar uma fila de prioridade (como uma heap binária), a complexidade pode ser reduzida para $O((V + E) \log V)$, onde $E$ é o número de arestas, tornando o algoritmo mais eficiente para grafos grandes e densos.

### 2.3. Implementação em Python

```python
import heapq

class Dijkstra:
    def __init__(self, grafo):
        self.grafo = grafo

    def calcular_menor_caminho(self, inicio):
        distancias = {no: float('infinity') for no in self.grafo.arestas}
        distancias[inicio] = 0

        fila_prioridade = [(0, inicio)]

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

            if distancia_atual > distancias[no_atual]:
                continue

            for vizinho, peso in self.grafo.arestas.get(no_atual, []):
                distancia = distancia_atual + peso

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

        return distancias
```

**Listing 1. Implementação do Algoritmo de Dijkstra em Python**

Este código implementa o algoritmo de Dijkstra utilizando uma fila de prioridade para melhorar a eficiência da busca pelo caminho mais curto. Ele calcula as distâncias mínimas de um nó inicial a todos os outros nós do grafo, retornando um dicionário com essas distâncias.

## 3. Algoritmo de Bellman-Ford

### 3.1. Descrição

O algoritmo de Bellman-Ford é uma técnica utilizada para encontrar o caminho mais curto a partir de um nó de origem em um grafo, e é capaz de lidar com arestas de peso negativo. Além de calcular o caminho mais curto, o algoritmo também pode detectar

ciclos negativos no grafo, que são caminhos que permitem reduzir indefinidamente o custo total.

Este algoritmo foi inicialmente aprendido em sala de aula com a ajuda do material didático [de Ouro Preto 2024]. A implementação em Python foi inspirada por recursos adicionais, como o artigo do GeeksforGeeks [GeeksforGeeks 2024], que forneceu uma base prática para aplicar o algoritmo de forma eficaz em programação.

### 3.2. Complexidade

O algoritmo de Bellman-Ford possui uma complexidade de tempo de $O(V \cdot E)$, onde $V$ é o número de vértices e $E$ é o número de arestas no grafo. Embora seja mais lento do que o algoritmo de Dijkstra para grafos sem arestas de peso negativo, ele é mais versátil e capaz de detectar ciclos de peso negativo, o que pode ser crucial em algumas aplicações.

### 3.3. Implementação em Python

```python
class BellmanFord:
    def __init__(self, grafo):
        self.grafo = grafo

    def calcular_menor_caminho(self, inicio):
        distancias = {no: float('infinity') for no in self.grafo
            .arestas}
        distancias[inicio] = 0

        for _ in range(len(self.grafo.arestas) - 1):
            for origem in self.grafo.arestas:
                for destino, peso in self.grafo.arestas[origem]:
                    if distancias[origem] + peso < distancias[
                        destino]:
                        distancias[destino] = distancias[origem]
                            + peso

        # Verifica se h  ciclos de peso negativo
        for origem in self.grafo.arestas:
            for destino, peso in self.grafo.arestas[origem]:
                if distancias[origem] + peso < distancias[
                    destino]:
                    raise ValueError("Grafo cont m ciclo de
                        peso negativo")

        return distancias
```

**Listing 2. Implementação do Algoritmo de Bellman-Ford em Python**

O código acima implementa o algoritmo de Bellman-Ford para calcular os caminhos mínimos a partir de um nó inicial em um grafo. O algoritmo começa inicializando as distâncias de todos os nós para infinito, exceto o nó de origem, que é definido como 0. Em seguida, ele relaxa todas as arestas repetidamente, atualizando as distâncias caso um caminho mais curto seja encontrado.

Após o relaxamento das arestas, o algoritmo verifica a presença de ciclos de peso negativo no grafo, que podem indicar que a solução não é válida ou que o custo pode ser reduzido indefinidamente. Se algum ciclo negativo for detectado, uma exceção é lançada.

Essa abordagem garante que, ao final da execução, as distâncias mínimas corretas sejam encontradas e que a integridade do grafo seja verificada quanto à presença de ciclos de peso negativo.

## 4. Algoritmo de Floyd-Warshall

### 4.1. Descrição

O algoritmo de Floyd-Warshall é uma técnica utilizada para encontrar os caminhos mínimos entre todos os pares de nós em um grafo. Ele funciona iterativamente, atualizando as distâncias conhecidas entre todos os pares de nós, considerando possíveis nós intermediários que possam reduzir a distância total.

Este algoritmo foi inicialmente aprendido em sala de aula com o auxílio do material didático [de Ouro Preto 2024], e a implementação em Python foi inspirada por recursos adicionais, como o artigo da Programiz [Programiz 2024], que ajudou a estruturar a abordagem de programação para este problema.

### 4.2. Complexidade

O algoritmo de Floyd-Warshall possui uma complexidade de tempo de $O(V^3)$, onde $V$ é o número de vértices no grafo. Essa complexidade torna o algoritmo menos eficiente para grafos muito grandes, mas ele é extremamente útil em casos onde é necessário calcular os caminhos mínimos entre todos os pares de nós, especialmente em grafos densos ou de tamanho moderado.

### 4.3. Implementação em Python

```python
class FloydWarshall:
    def __init__(self, grafo):
        self.grafo = grafo

    def calcular_todos_caminhos_minimos(self):
        # Inicializando a matriz de dist ncias
        distancias = {}
        for origem in self.grafo.arestas:
            distancias[origem] = {}
            for destino in self.grafo.arestas:
                if origem == destino:
                    distancias[origem][destino] = 0
                else:
                    distancias[origem][destino] = float('infinity')
            for destino, peso in self.grafo.arestas[origem]:
                distancias[origem][destino] = peso

        # Algoritmo Floyd-Warshall
        for k in self.grafo.arestas:
            for i in self.grafo.arestas:
                for j in self.grafo.arestas:
                    if distancias[i][j] > distancias[i][k] + distancias[k][j]:
                        distancias[i][j] = distancias[i][k] + distancias[k][j]

        return distancias
```

**Listing 3. Implementação do Algoritmo de Floyd-Warshall em Python**

O código acima implementa o algoritmo de Floyd-Warshall para calcular os caminhos mínimos entre todos os pares de nós em um grafo. O algoritmo começa inicializando uma matriz de distâncias, onde cada entrada representa a menor distância conhecida entre dois nós. Em seguida, ele itera sobre todos os nós intermediários possíveis, atualizando as distâncias caso um caminho mais curto seja encontrado através de um nó intermediário.

Essa abordagem garante que, ao final das iterações, a matriz de distâncias contenha o menor caminho possível entre todos os pares de nós, considerando todos os caminhos intermediários possíveis no grafo.

## 5. Conclusão

Neste relatório, analisamos e implementamos três algoritmos fundamentais para o problema do caminho mínimo em grafos: Dijkstra, Bellman-Ford e Floyd-Warshall. Cada um desses algoritmos tem suas próprias características e aplicabilidades, o que pode influenciar sua escolha dependendo do tipo de grafo e das necessidades específicas do problema.

O algoritmo de Dijkstra é eficiente para grafos com arestas de peso não negativo e é particularmente útil para encontrar o caminho mais curto a partir de um único nó. Sua complexidade reduzida com o uso de filas de prioridade o torna adequado para muitos cenários práticos.

O algoritmo de Bellman-Ford, por outro lado, é mais versátil, pois pode lidar com arestas de peso negativo e detectar ciclos de peso negativo. No entanto, sua complexidade maior o torna menos eficiente para grafos grandes em comparação com o Dijkstra.

O algoritmo de Floyd-Warshall oferece uma solução para o problema de todos os pares de caminhos mínimos, sendo útil quando se precisa encontrar caminhos entre todos os pares de nós em um grafo. Embora tenha uma complexidade mais alta, sua abordagem abrangente é valiosa para grafos densos ou problemas específicos que requerem tal informação.

O exercício de implementar esses algoritmos em Python, utilizando um grafo representado em formato JSON, nos permitiu comparar diretamente suas performances e aplicações práticas. A experiência prática de programar e testar esses algoritmos aprofundou nossa compreensão de suas características e limitações, destacando a importância de escolher o algoritmo adequado conforme o contexto do problema.

## Referências

[de Ouro Preto 2024] de Ouro Preto, U. F. (2024). Problema do caminho mínimo. Acesso em: 15 ago. 2024.

[DIO 2024] DIO (2024). O algoritmo de dijkstra em python: Encontrando o caminho mais curto. Acesso em: 15 ago. 2024.

[GeeksforGeeks 2024] GeeksforGeeks (2024). Bellman-ford algorithm in python. Acesso em: 15 ago. 2024.

[Programiz 2024] Programiz (2024). Floyd-warshall algorithm. Acesso em: 15 ago. 2024.