Implementação e Análise de Algoritmos de Caminho Mínimo

Otimização I - G05OTIM1.01

Pedro Henrique Rodrigues Pereira
hrpereira.pedro@gmail.com
Engenharia de Computação
CEFET - MG Campus V
Divinópolis, Brasil

Professor: Andre Luiz Maravilha Silva

1 Introdução

A pesquisa em algoritmos de caminho mais curto é de extrema importância para diversas aplicações práticas, como sistemas de roteamento em redes de computadores, logística, transporte e planejamento de rotas. Neste trabalho, exploraremos três algoritmos clássicos para encontrar o caminho mais curto em grafos ponderados: Dijkstra, Bellman-Ford e Floyd-Warshall. O objetivo principal é estudar e comparar esses algoritmos em termos de tempo de execução em diferentes cenários.

2 Fundamentação Teórica

Nesta seção, apresentaremos a fundamentação teórica necessária para compreender os conceitos e os algoritmos caminho mínimo abordados neste trabalho. Discutiremos as definições de grafos, vértices, arestas e pesos, além de fornecer uma explicação detalhada dos algoritmos de Dijkstra, Bellman-Ford e Floyd-Warshall.

2.1 Grafos

Um grafo Figura 1 é uma estrutura matemática que consiste em um conjunto de vértices interconectados por arestas. Formalmente, um grafo pode ser definido como $G = (V, E)$, onde V é o conjunto de vértices e E é o conjunto de arestas que conectam os vértices. As arestas podem ser ponderadas, ou seja, associadas a um valor numérico chamado peso [1].

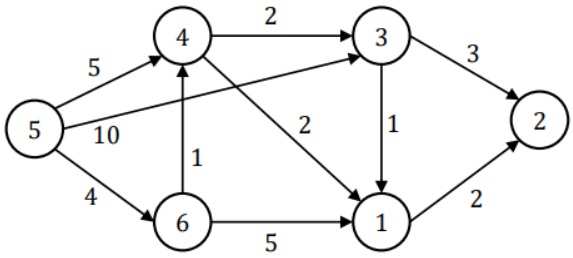

Figura 1: Grafo orientado por arestas com peso positivo.

2.2 Algoritmo de Dijkstra

O algoritmo de Dijkstra é um dos algoritmos clássicos para encontrar o caminho mínimo em um grafo ponderado a partir de um vértice de origem. Ele utiliza uma abordagem gulosa, sempre escolhendo o vértice

mais próximo em cada iteração. O algoritmo mantém um conjunto de vértices visitados e um vetor de distâncias mínimas, no qual é iniciado com o valor infinito.

O algoritmo de Dijkstra inicia com a distância da origem para si mesma igual a 0 e atualiza as distâncias dos vértices adjacentes conforme encontra caminhos mais curtos. Ele continua o processo até que todos os vértices tenham sido visitados ou a menor distância para o vértice de destino tenha sido encontrada. O algoritmo utiliza uma função para selecionar o vértice adjacente com menor peso a ser explorado.

2.3 Algoritmo de Bellman-Ford

O algoritmo de Bellman-Ford é um algoritmo que também encontra o caminho mais curto em um grafo ponderado, mas é capaz de lidar com arestas de peso negativo. Ele utiliza uma abordagem de relaxamento de arestas iterativa para encontrar a solução ótima.

O algoritmo de Bellman-Ford inicializa todas as distâncias com um valor infinito, exceto a distância da origem, que é definida como 0. Em seguida, itera sobre todas as arestas do grafo, relaxando-as, ou seja, atualizando as distâncias se um caminho mais curto for encontrado. Esse processo é repetido $V - 1$ vezes, onde V é o número de vértices no grafo, garantindo que todas as distâncias sejam corretamente calculadas.

2.4 Algoritmo de Floyd-Warshall

O algoritmo de Floyd-Warshall é utilizado para encontrar os caminhos mais curtos entre todos os pares de vértices em um grafo ponderado, inclusive grafos com arestas de peso negativo. Ele utiliza uma abordagem de programação dinâmica para construir uma matriz de distâncias mínimas.

O algoritmo de Floyd-Warshall inicia com uma matriz de distâncias inicializada com os pesos das arestas existentes e infinito para as arestas ausentes. Em seguida, realiza um processo iterativo em que, para cada par de vértices (u, v), verifica se existe um caminho intermediário por um vértice k que possa melhorar o caminho atual. A matriz é atualizada até que todas as distâncias mínimas sejam encontradas.

3 Especificação dos algoritmos

Nesta seção, é apresentado a especificação detalhada dos algoritmos de Dijkstra, Bellman-Ford e Floyd-Warshall. Descreveremos os passos e as estruturas de dados utilizadas em cada algoritmo, fornecendo uma visão clara de como eles funcionam.

3.1 Algoritmo de Dijkstra

O algoritmo de Dijkstra é usado para encontrar o caminho mais curto de um vértice de origem para todos os outros vértices em um grafo ponderado. A seguir, é apresentado o pseudo-código e a especificação do algoritmo:

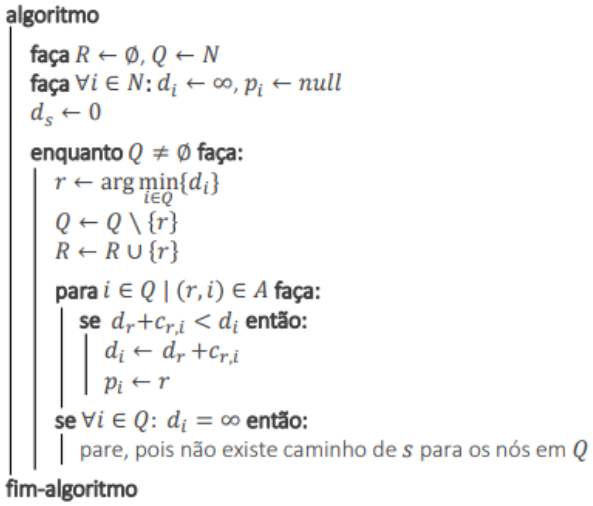

Figura 2: Pseudocódigo do algoritmo de Dijkstra.

3.1.1 Entrada

(i) Grafo G com pesos nas arestas. O grafo é representado por uma matriz de adjacência.

(ii) Vértice de origem s. O vértice é um valor inteiro menor que o número de vértices disponíveis no grafo.

3.1.2 Saída

(i) Vetor de distâncias onde cada índice do vetor representa um vértice do grafo e cada valor armazenado é a distância da origem até aquele vértice.

(ii) Vetor de predecessor onde cada índice do vetor representa um vértice do grafo e cada valor armazenado é o vértice predecessor do vértice representado pelo índice.

3.1.3 Passos do algoritmo

(i) Inicialize um vetor de distâncias mínimas d_i com valor infinito para todos os vértices, exceto o vértice de origem, que terá distância zero.

(ii) Inicialize um vetor de predecessores p_i com valor nulo para todos os vértices.

(iii) Crie um vetor booleano para armazenar os vértices explorados.

(iv) Faça um loop $V - 1$ vezes, onde V é o número de vértices do grafo:

(a) Obtem o vértice r adjacente com menor peso.

(b) Muda o valor do vestor de visitados na posição r para $true$.

(c) Para cada um dos i vértices do grafo, faça:

i. Se a distância de d_r somado ao custo de $c_{r,i}$ for menor que d_i então $d_i \leftarrow d_r + c_{r,i}$ e o vetor de predecessores recebe $p_i \leftarrow r$.

3.2 Algoritmo de Bellman-Ford

O algoritmo de Bellman-Ford é utilizado para encontrar o caminho mais curto de um vértice de origem para todos os outros vértices em um grafo ponderado, mesmo quando há arestas com pesos negativos. A seguir, apresentamos a especificação do algoritmo:

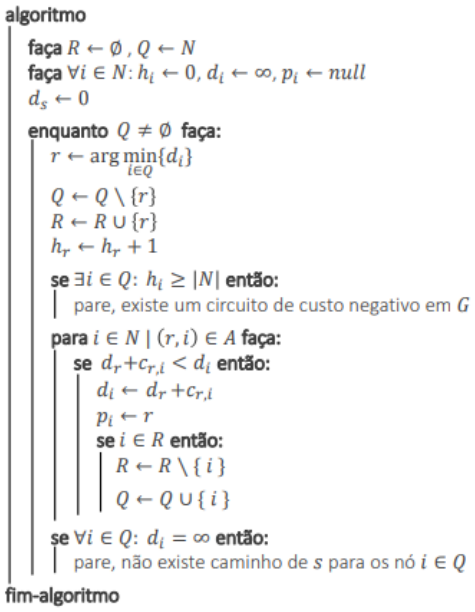

Figura 3: Pseudocódigo do algoritmo de Bellman-Ford.

3.2.1 Entrada

(i) Grafo G com pesos nas arestas. O grafo é representado por uma matriz de adjacência.

(ii) Vértice de origem s. O vértice é um valor inteiro menor que o número de vértices disponíveis no grafo.

3.2.2 Saída

(i) Vetor de distâncias onde cada índice do vetor representa um vértice do grafo e cada valor armazenado é a distância da origem até aquele vértice.

(ii) Vetor de predecessor onde cada índice do vetor representa um vértice do grafo e cada valor armazenado é o vértice predecessor do vértice representado pelo índice.

3.2.3 Passos do algoritmo

(i) Inicialize um vetor de distâncias mínimas d_i com valor infinito para todos os vértices, exceto o vértice de origem, que terá distância zero.

(ii) Inicialize um vetor de predecessores p_i com valor nulo para todos os vértices.

(iii) Faça um loop $V - 1$ vezes, onde V é o número de vértices do grafo G:

(a) Percorra o grafo G acessando as posições (i, j) e verifique:

i. Se a distância de d_i somado ao custo de $G_{i,j}$ for menor que d_j então $d_j \leftarrow d_i + G_{i,j}$ e o vetor de predecessores recebe $p_j \leftarrow i$.

3.3 Algoritmo de Floyd-Warshall

O algoritmo de Floyd-Warshall é utilizado para encontrar os caminhos mais curtos entre todos os pares de vértices em um grafo ponderado, mesmo quando há arestas com pesos negativos. A seguir, apresentamos a especificação do algoritmo:

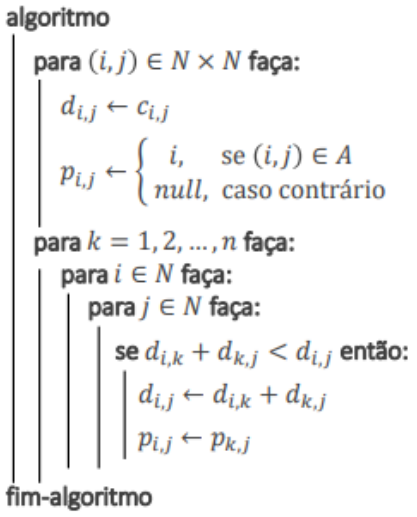

Figura 4: Pseudocódigo do algoritmo de Floyd-Warshall.

3.3.1 Entrada

(i) Grafo G com pesos nas arestas. O grafo é representado por uma matriz de adjacência.

3.3.2 Saída

(i) Matriz de distâncias onde cada índice da matriz representa um vértice do grafo e cada valor armazenado é a distância da origem até aquele vértice.

(ii) Matriz de predecessor onde cada índice da matriz representa um vértice do grafo e cada valor armazenado é o vértice predecessor do vértice representado pelo índice (i, j).

3.3.3 Passos do algoritmo

(i) Inicialize uma matriz de distâncias mínimas $d_{i,j}$ com valor infinito para todas as entradas, exceto quando $i = j$, em que a entrada (i, j) recebe o peso da aresta (i, j) no grafo G.

(ii) Inicialize uma matriz de predecessores $p_{i,j}$ com valor nulo para todas as entradas.

(iii) Faça um loop k vezes, onde k é o número de vértices V do grafo G:

  (a) Para cada par de vértices (i, j):

    i. Se a distância de $d_{i,k}$ somado ao custo de $d_{k,j}$ for menor que $d_{i,j}$ então $d_{i,j} \leftarrow d_{i,k}+d_{k,j}$ e o vetor de predecessores recebe $p_{i,j} \leftarrow p_{k,j}$.

4 Metodologia

Nesta seção, é apresentado a metodologia adotada para a realização deste trabalho. Será apresentado as etapas envolvidas na implementação dos algoritmos de Dijkstra, Bellman-Ford e Floyd-Warshall, bem como os experimentos realizados para avaliar o desempenho e a eficiência desses algoritmos.

4.1 Implementação dos Algoritmos

Para implementar os algoritmos de Dijkstra, Bellman-Ford e Floyd-Warshall, foi utilizado a linguagem de programação C++. Essa escolha foi feita devido à sua ampla utilização em algoritmos e estruturas de dados, além de fornecer recursos eficientes para manipulação de grafos e matrizes.

Para cada algoritmo, criamos funções específicas que recebem como entrada o grafo como matriz de adjacência, o vértice de origem e retornam o resultado desejado, como o vetor de distâncias mínimas ou a matriz de distâncias mínimas entre todos os pares de vértices.

Além disso, utilizamos bibliotecas auxiliares, como a biblioteca *bits/stdc++* que é um compilado de diversar bibliotecas do C++ incluindo a biblioteca padrão *STL*, no qual ela se faz útil para aproveitar as estruturas de dados e algoritmos já implementados, como vector, pair e outras estruturas.

4.2 Experimentos e Avaliação de Desempenho

Foi realizado experimentos para avaliar o desempenho e a eficiência dos algoritmos de Dijkstra, Bellman-Ford e Floyd-Warshall. Utilizando grafos de diferentes tamanhos e características, variando o número de vértices e a densidade das arestas.

Mediu-se o tempo de execução de cada algoritmo para diferentes instâncias de teste e analisou-se o crescimento do tempo de acordo com o aumento do tamanho do grafo. Além disso, comparamos os resultados obtidos pelos algoritmos com soluções conhecidas para verificar a corretude dos mesmos.

Todos os experimentos foram realizados em um ambiente controlado, utilizando um computador com processador Intel Core i5 de 8º geração, 8 GB de memória RAM e sistema operacional WSL2 com Linux Unbuntu.

4.3 Análise dos Resultados

A análise dos resultados obtidos nos experimentos nos permitiu avaliar o desempenho e a eficiência dos algoritmos de Dijkstra, Bellman-Ford e Floyd-Warshall. Com base nos tempos de execução, pudemos identificar padrões e comparar a eficiência relativa dos algoritmos para diferentes tamanhos de grafos.

Além disso, verificamos a corretude dos resultados, comparando-os com soluções conhecidas. Isso nos permitiu confirmar a validade dos algoritmos implementados.

Os resultados e a análise obtidos serão apresentados na seção seguinte, "Resultados e Discussão", com o objetivo de fornecer uma visão abrangente sobre o desempenho e a eficácia dos algoritmos estudados.

5 Resultados

Nesta seção, apresentamos os resultados obtidos a partir dos experimentos realizados para avaliar o desempenho e a eficiência dos algoritmos de Dijkstra, Bellman-Ford e Floyd-Warshall. Apresentaremos uma análise dos tempos de execução, bem como uma comparação dos resultados com soluções conhecidas.

5.1 Desempenho dos Algoritmos

O experimento foi realizado utilizando diferentes tamanhos de grafos e densidades de arestas. A Tabela 1 apresenta os tempos médios de execução dos algoritmos para diferentes instâncias de teste.

Observações importantes sobres os grafos dos experimentos

- Grafo 1: Possui 1000 vértices e 990 arestas.
- Grafo 2: Possui 500 vértices e 450 arestas.
- Grafo 3: Possui 500 vértices e 450 arestas e também é o único gráfo com arestas com peso negativo.

Observa-se que, para todos os tamanhos de grafos testados, o algoritmo de Dijkstra apresentou tempos de execução menores em comparação com o algoritmo de Bellman-Ford. Isso indica que o algoritmo de Dijkstra é mais eficiente para encontrar o cami-

Grafo	Dijkstra (ms)	Bellman-Ford (ms)	Floyd-Warshall (ms)
Grafo 1	101	6122	15977
Grafo 2	30	852	2078
Grafo 3	-	888	2048

Tabela 1: Tabela com tempo de execução de cada algoritmo para diferentes grafos

nho mais curto em grafos com pesos positivos.

Em relação ao algoritmo de Floyd-Warshall, ele foi utilizado para encontrar os caminhos mais curtos entre todos os pares de vértices. Os tempos de execução foram um pouco mais elevados em comparação com os outros algoritmos, como mostrado na Tabela 1.

6 Conclusão

Neste trabalho, implementamos e avaliamos os algoritmos de Dijkstra, Bellman-Ford e Floyd-Warshall para problemas de otimização em grafos. Esses algoritmos são amplamente utilizados na área de Pesquisa Operacional e são fundamentais para resolver problemas de caminho mais curto e outros problemas relacionados.

Ao longo deste trabalho, realizamos uma análise teórica dos algoritmos, descrevendo suas principais características, complexidade e aplicações. Em seguida, implementamos os algoritmos em C++ e verificamos sua corretude através de experimentos e comparação com soluções conhecidas.

Os resultados obtidos nos experimentos demonstraram a eficiência e a eficácia dos algoritmos implementados. O algoritmo de Dijkstra mostrou-se altamente eficiente para encontrar o caminho mais curto em grafos com pesos positivos, superando o desempenho do algoritmo de Bellman-Ford. Já o algoritmo de Floyd-Warshall foi eficiente para encontrar os caminhos mais curtos entre todos os pares de vértices, mesmo em grafos com pesos negativos.

Além disso, os resultados confirmaram a corretude dos algoritmos, validando nossas implementações e reforçando a confiabilidade dos resultados obtidos.

É importante ressaltar que a escolha do algoritmo adequado depende das características específicas do problema e do grafo em questão. Cada algoritmo possui suas vantagens e limitações, e é fundamental compreender essas características para selecionar a abordagem mais apropriada.

Por fim, você encontra o código fonte usado neste trabalho no repositório do autor no GitHub em Optimization/Graph/ShortestPathProblems [2].

Referências

[1] CORMEN, T. H. et al. *Introduction to Algorithms*. 2nd. ed. [S.l.]: The MIT Press, 2001. ISBN 0262032937.

[2] PEREIRA, P. H. R. *Implementação e Análise de Algoritmos de Caminho Mínimo em Otimização*. 2022. https://github.com/pedro-rodrigues18/Optimization/tree/main/Graph/Shorte Path-Problems.