UNIVERSIDADE ESTADUAL DO MATO GROSSO DO SUL

CAUÊ MENDONÇA MAGELA DO Ó
VINICIUS SILVA BALBINO

PROBLEMA DO CAMINHO MÍNIMO

Dourados - MS
2023

CAUÊ MENDONÇA MAGELA DO Ó
VINICIUS SILVA BALBINO

PROBLEMA DO CAMINHO MÍNIMO

Trabalho apresentado à disciplina de Teoria dos Grafos do curso de Ciência da Computação, como requisito para a obtenção de nota, Universidade Estadual de Mato Grosso do Sul.

Orientador: Prof Me. Delair Osvaldo Martinelli Júnior

SUMÁRIO

1. INTRODUÇÃO

O problema dos caminhos mínimos é um dos problemas fundamentais na teoria dos grafos. Ele envolve a busca pelo caminho mais curto entre dois vértices em um grafo ponderado, onde o "caminho mais curto" é geralmente definido como o caminho com o menor peso total. Os caminhos mínimos têm inúmeras aplicações em várias áreas, desde sistemas de navegação até telecomunicações, logística e muito mais.

Formalmente, o problema dos caminhos mínimos pode ser descrito da seguinte maneira:

Dado um grafo ponderado $G = (V, E)$, onde V é o conjunto de vértices e E é o conjunto de arestas, e um vértice de origem s, encontre os caminhos mínimos de s para todos os outros vértices em V. Isso envolve o cálculo das distâncias mínimas (pesos mínimos) de s para todos os vértices em V. A distância mínima de s para qualquer vértice v é frequentemente denotada como $d(s, v)$. Além disso, podemos querer calcular não apenas as distâncias mínimas, mas também os próprios caminhos mínimos.

A modelagem matemática do problema do caminho mínimo envolve a representação formal dos elementos do problema, incluindo o grafo, seus vértices, arestas e pesos. Aqui está a modelagem matemática geral do problema dos caminhos mínimos:

Grafo: O grafo é representado por um par ordenado$G = (V, E)$, onde:

V: O conjunto de vértices (nós) do grafo.

E: O conjunto de arestas (conexões) entre os vértices.

Pesos das Arestas: Cada aresta (u, v) em E é associada a um peso ou custo não negativo, denotado por $w(u, v)$. Esses pesos podem representar distâncias, custos, tempos ou qualquer métrica relevante. Para grafos não ponderados, os pesos são frequentemente iguais a 1.

Vértice de Origem: Um vértice de origem s, que é um elemento de V, é selecionado como o ponto de partida para calcular os caminhos mínimos para todos os outros vértices no grafo.

Variáveis de Decisão: O problema dos caminhos mínimos envolve a determinação das seguintes variáveis de decisão:
$d(s, v)$: A distância mínima (peso mínimo) do vértice de origem s para um vértice v em V. Esta é a principal quantidade que queremos calcular.

$\pi(v)$: A predecessora de um vértice v no caminho mínimo de s para v. Isso permite reconstruir o caminho mínimo real.

Restrições: As principais restrições do problema dos caminhos mínimos incluem:
Para qualquer par de vértices (u, v) em E, a distância mínima de s a v deve ser menor ou igual à distância de s a u mais o peso da aresta entre u e v: $d(s, v) \leq d(s, u) + w(u, v)$.

O objetivo do problema é encontrar todas as distâncias mínimas d(s, v) para todos os vértices v em V a partir do vértice de origem s, bem como as predecessoras $\pi(v)$ para reconstruir os caminhos mínimos.

A resolução do problema dos caminhos mínimos visa otimizar a escolha dos caminhos com base nos pesos das arestas e nas métricas relevantes, tornando-o uma parte crucial de muitas aplicações do mundo real, como sistemas de navegação, redes de telecomunicações, logística, entre outros. Para alcançar esse objetivo, são usados algoritmos específicos, e neste artigo abordaremos o algoritmo de Dijkstra.

2. APLICAÇÕES

O problema de caminho mínimo é um dos problemas genéricos intensamente estudados e utilizados em diversas áreas como Engenharia de Transportes, Pesquisa Operacional, Ciência da Computação e Inteligência Artificial. Isso decorre do seu potencial de aplicação a inúmeros problemas práticos que ocorrem em transportes, logística, redes de computadores e de telecomunicações, entre outros.

Podem ser aplicados para encontrar automaticamente direções entre locais físicos, como instruções de direção em sites de mapeamento da web como o MapQuest ou o Google Maps. Para esta aplicação, algoritmos especializados rápidos estão disponíveis.

Outras possibilidades de aplicação incluem quaisquer problemas envolvendo redes ou grafos em que se tenha grandezas (distâncias, tempo, perdas, ganhos, despesas) que se acumulem linearmente ao longo do percurso da rede. Se alguém representa uma máquina abstrata não determinística como um gráfico em que os vértices descrevem estados e arestas descrevem possíveis transições, algoritmos de caminho mínimo podem ser usados para encontrar uma sequência ótima de opções para atingir um determinado estado objetivo ou para estabelecer limites mais baixos no tempo necessário para atingir um determinado estado. Por exemplo, se os vértices representam os estados de um quebra-cabeça como o Cubo de Rubik e cada borda direcionada corresponde a um único movimento ou turno, os algoritmos de caminho mínimo podem ser usados para encontrar uma solução que use o número mínimo possível de movimentos. Um problema relacionado é o Problema do Caixeiro-viajante, que consiste em determinar o caminho mais curto que passa exatamente uma vez por cada vértice e retorna ao vértice de partida. Este é um problema NP-Completo, para os quais não há uma solução eficiente.

Na teoria de grafos, o problema do caminho mínimo consiste na minimização do custo de travessia de um grafo entre dois nós (ou vértices); custo este dado pela soma dos pesos de cada aresta percorrida. Existem várias variantes para problemas de caminho mínimo, cada uma adequada a um conjunto de problemas diferente:

- **Problema de único destino**: consiste em determinar o menor caminho entre cada um dos nós do grafo e um nó de destino dado.
- **Problema de única origem**: determinar o menor caminho entre um nó dado e todos os demais nós do grafo.

- **Problema de origem-destino**: determinar o menor caminho entre nós dados.
- **Problemas de todos os pares**: determinar o menor caminho entre cada par de nós presentes no grafo.

Os algoritmos especializados em solucionar o problema do caminho mínimo são eventualmente chamados de algoritmos de busca de caminhos. Entre os algoritmos dessa classe, os mais conhecidos são:

- **Algoritmo de Dijkstra**: Resolve o problema com um vértice-fonte em grafos cujas arestas tenham peso maior ou igual a zero. Sem reduzir o desempenho, este algoritmo é capaz de determinar o caminho mínimo, partindo de um vértice de início v para todos os outros vértices do grafo.[1]
- **Algoritmo de Bellman-Ford**: Resolve o problema para grafos com um vértice-fonte e arestas que podem ter pesos negativos.
- **Algoritmo A***: um algoritmo heurístico que calcula o caminho mínimo com um vértice-fonte.

3. ALGORITMO DE DIJKSTRA E SEU FUNCIONAMENTO

O algoritmo de Dijkstra é amplamente utilizado para encontrar o caminho mínimo em grafos com arestas ponderadas não negativas. Ele começa a partir do vértice de origem e gradualmente explora os vértices vizinhos, sempre escolhendo o vértice com o caminho mínimo conhecido até o momento. O algoritmo mantém uma lista de distâncias mínimas conhecidas e é eficiente para grafos densos.

O algoritmo começa com o conjunto $S0$ contendo apenas o vértice de origem $(u0)$. A cada passo, ele encontra um vértice u que está mais próximo de u0 (com base em sua distância) e o adiciona ao conjunto Si. Isso é feito de acordo com a fórmula:

$d(u0,\ Si) = min\{d(u0,\ u)\ +\ w(uv)\}$ para $u\ \in\ Si$ e $v\ \in\ V\backslash S$

Onde:

$d(u0,\ Si)$ é a distância da origem $(u0)$ para o conjunto Si.

$d(u0,\ u)$ é a distância da origem para um vértice u em Si.

$w(uv)$ é o peso da aresta que conecta u a um vértice v fora de Si.

Para cada vértice adicionado a Si, o algoritmo calcula o caminho mais curto de $u0$ para esse vértice e, em última análise, encontra os caminhos mais curtos para todos os vértices.

Imaginemos que temos um mapa de cidades interligadas por estradas, e cada estrada tem um comprimento (chamado de peso). O objetivo é encontrar o caminho mais curto de uma cidade de origem para todas as outras cidades.

O algoritmo de Dijkstra nos ajuda a fazer isso, passo a passo:

Passo 1: Inicialização

Começamos na cidade de origem. No início, a distância da cidade de origem para ela mesma é zero, pois já estamos lá. As distâncias para todas as outras cidades são definidas como infinito, porque não sabemos quanto tempo levará para alcançá-las.

Passo 2: Exploração

Agora, temos que explorar as cidades vizinhas. Começamos com a cidade de origem e vamos para a cidade vizinha mais próxima. Isso significa que escolhemos a cidade mais próxima da cidade de origem, que ainda não tenha sido visitada.

Para cada cidade vizinha, calculamos a distância total percorrida até lá. Se encontrarmos um caminho mais curto do que aquele que já conhecíamos, atualizamos a distância.

Repetimos esse processo até visitar todas as cidades vizinhas da cidade de origem.

Passo 3: Escolha da Próxima Cidade

Após explorar todas as cidades vizinhas da cidade de origem, escolhemos a cidade mais próxima que ainda não foi visitada. Essa cidade se torna a nossa nova cidade de origem.

Passo 4: Repita

Repetimos os passos 2 e 3 até visitar todas as cidades. À medida que exploramos mais cidades, atualizamos as distâncias e encontramos os caminhos mais curtos.

Passo 5: Resultados

Após visitar todas as cidades, teremos encontrado os caminhos mais curtos de nossa cidade de origem para todas as outras cidades. Essas distâncias são registradas e podem ser usadas para navegar pelo mapa de forma eficiente.

O algoritmo de Dijkstra é como se estivéssemos "explorando" o mapa, cidade por cidade, sempre escolhendo o caminho mais curto disponível. Isso nos ajuda a encontrar os caminhos mais curtos em um grafo ponderado, como um mapa de cidades com estradas de diferentes comprimentos.É uma ferramenta poderosa para resolver problemas de otimização em grafos e tem aplicações em várias áreas, desde redes de transporte até sistemas de GPS

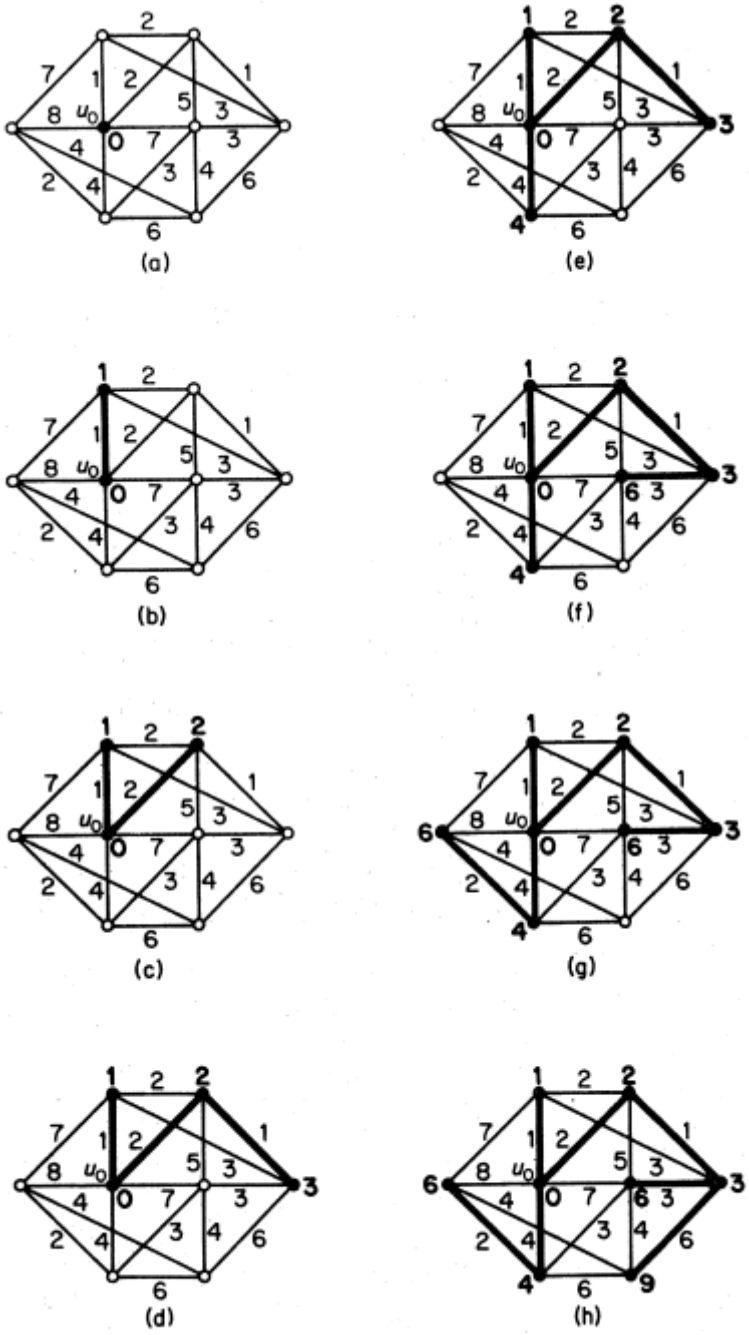

Figura 1: Exemplo 1,Problema do Caminho Mínimo.

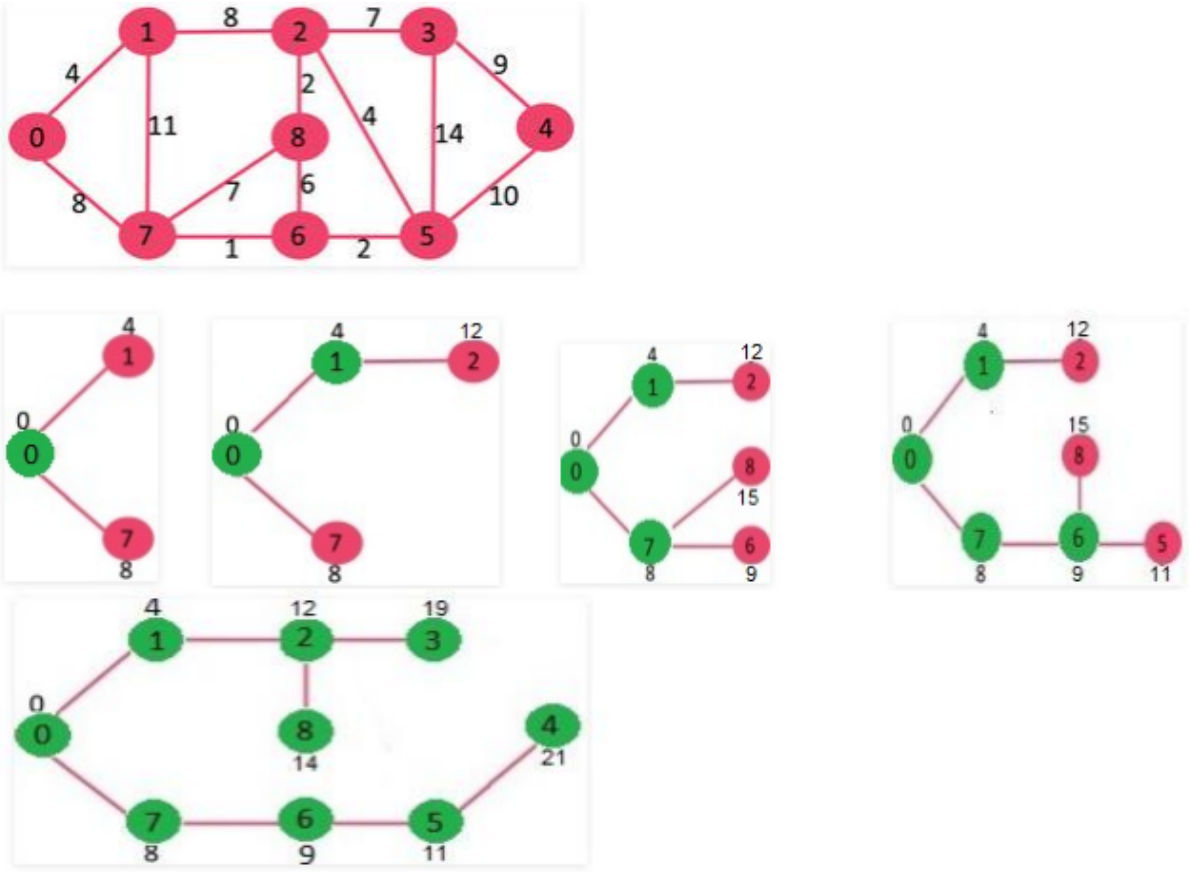

Figura 2: Exemplo 2,Problema do Caminho Mínimo.

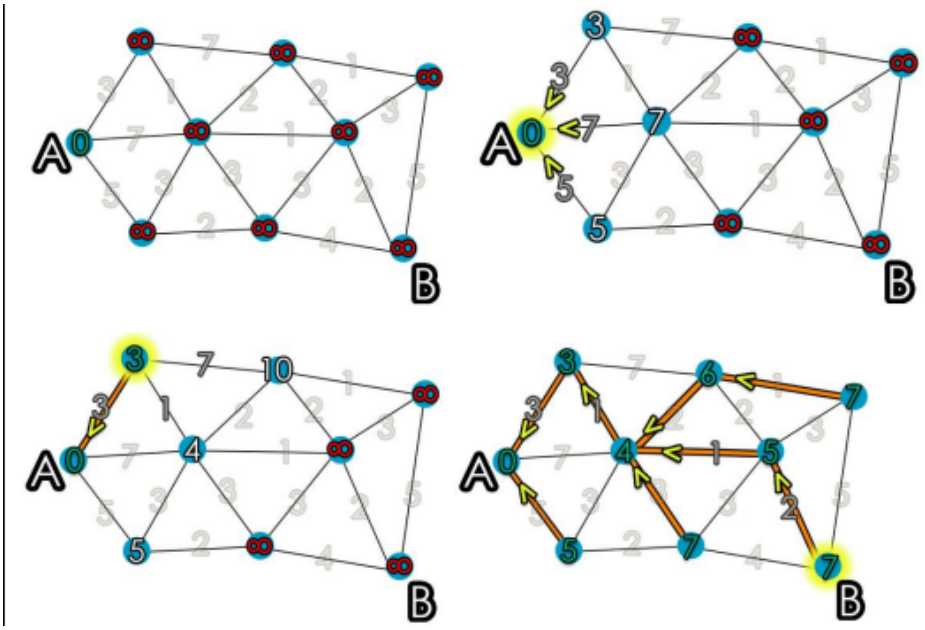

Figura 3: Exemplo 3,Problema do Caminho Mínimo.

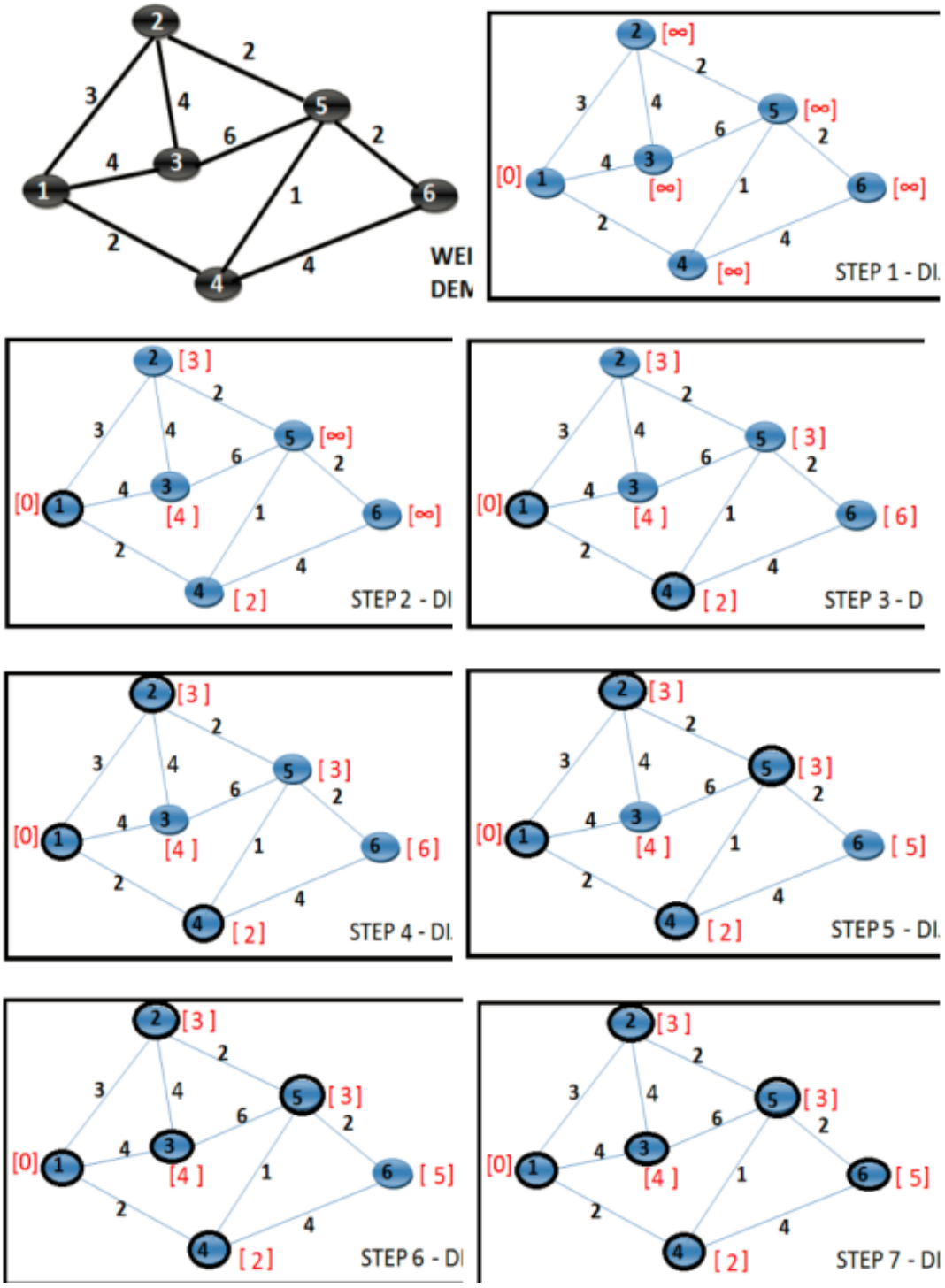

Figura 4: Exemplo 4,Problema do Caminho Mínimo.

4. IMPLEMENTAÇÃO

O código C++ apresentado neste artigo traz a implementação do algoritmo de Dijkstra aplicado a um grafo ponderado. Em um grafo ponderado, cada aresta possui um valor associado, que pode representar distâncias, custos ou qualquer outra grandeza significativa. O principal propósito deste algoritmo é determinar as distâncias mínimas a partir de um vértice de origem em direção a todos os outros vértices do grafo. Essa capacidade de encontrar caminhos mais curtos em um contexto de rede ou mapa é valiosa em diversas aplicações, como sistemas de navegação, planejamento logístico e otimização de recursos.

4.1. ESTRUTURA DO CÓDIGO

4.1.1. CLASSE GRAFO

A classe Grafo tem a finalidade de representar um grafo ponderado e oferece métodos para adicionar arestas e executar o algoritmo de Dijkstra. Os principais elementos desta classe incluem:

int V: Um atributo que armazena o número de vértices presentes no grafo.

vector<vector<pair<int, int>>> adj: Uma lista de adjacência utilizada para armazenar as arestas do grafo. Cada elemento nessa estrutura é um par composto pelo vértice de destino e o peso associado à aresta. Essa representação permite registrar as conexões entre os vértices e os respectivos pesos das arestas, essenciais para o cálculo das distâncias mínimas pelo algoritmo de Dijkstra.

4.1.2. MÉTODO adicionarAresta

A função adicionarAresta desempenha a função de permitir a inclusão de arestas no grafo. Ela requer três parâmetros: o vértice de partida (u), o vértice de chegada (v) e o peso associado à aresta. É relevante notar que, caso o grafo seja não direcionado, esta função adiciona a mesma aresta em ambos os sentidos, ou seja, estabelece a

conexão tanto de u para v quanto de v para u. Isso assegura que as relações sejam consideradas bilateralmente, possibilitando a representação de arestas em ambas as direções, se necessário.

4.1.3. MÉTODO menorCaminho

O método menorCaminho é a parte central da implementação do algoritmo de Dijkstra. Sua função principal é calcular as distâncias mínimas a partir de um vértice de origem. Para realizar essa tarefa, o método emprega uma estratégia que combina o uso de um conjunto (set) para manter os vértices organizados com base em suas distâncias e um vetor para armazenar as distâncias mínimas.

O processo se inicia com a inicialização das distâncias de todos os vértices, onde a distância é definida como infinito, com exceção do vértice de origem, o qual tem sua distância configurada como zero, afinal, a distância da origem para ela mesma é naturalmente nula

Em seguida, o vértice de origem é inserido em uma fila, sendo marcado com uma distância zero, indicando seu ponto de partida para a busca das distâncias mínimas.

O coração do algoritmo de Dijkstra reside no laço principal, que continua a operação enquanto a fila não estiver vazia. A cada iteração desse laço, o algoritmo executa as seguintes etapas:

Remove o vértice que possui a menor distância da fila, significando que esse vértice será o próximo a ser explorado.

Para cada vizinho do vértice atual, o algoritmo verifica se há um caminho mais curto passando pelo vértice atual. Caso seja encontrado um caminho mais curto, a distância até esse vizinho é atualizada.

A fila é reorganizada para considerar essa nova distância, garantindo que os vértices sejam visitados na ordem correta.

Após a conclusão do algoritmo, as distâncias mínimas a partir do vértice de origem para todos os outros vértices são calculadas e podem ser utilizadas conforme necessário.

Esse processo de atualização de distâncias e reorganização da fila é crucial para a eficiência e precisão do algoritmo de Dijkstra, permitindo encontrar os caminhos mais curtos em um grafo ponderado.

4.1.4. FUNÇÃO main

A função main desempenha o papel central no programa. Nela, é instanciado um objeto da classe Grafo, seguido pela adição de arestas ponderadas, pela definição do vértice de origem e, por fim, a invocação do método grafo para determinar as distâncias mínimas. Como exemplo, o código foi configurado para criar um grafo com 6 vértices e aplicar o algoritmo a partir do vértice 0.

4.2. USO DO CÓDIGO

Para usar o código, siga estas etapas:

1. Configure um ambiente de desenvolvimento C++.
2. Copie o código fornecido para um arquivo .cpp.
3. Compile o código usando um compilador C++ (por exemplo, g++ seu_programa.cpp -o seu_programa).
4. Execute o programa compilado (./seu_programa).

O programa calculará as distâncias mínimas a partir do vértice de origem especificado no fim do código e imprimirá as distâncias na saída padrão. Personalize o número de vértices, as arestas e o vértice de origem conforme necessário para seu problema específico.

Caso queira fazer alguma alteração do código, a variável *Vertices,* é a quantidades de vértices que o grafo irá possuir, logo abaixo existem os métodos *adicionarAresta* que faz a ligação de vértices onde existem o vértice 1 e vértice 2 e o respectivo peso da aresta. E logo após o método que faz o cálculo de menor caminho passando qual irá ser a origem.

4.3. EXPLICAÇÃO DO FUNCIONAMENTO EM BAIXO NÍVEL

Inicialização:

Inicialize um conjunto (set) que conterá os vértices a serem explorados. Este conjunto será chamado de fila.
Inicialize um vetor de distâncias *dist* com distâncias iniciais definidas como infinito para todos os vértices, exceto o vértice de origem, que tem distância zero.Insira o vértice de origem na fila com distância zero.

Iteração:

Enquanto a fila não estiver vazia:
Retire o vértice u da fila que possui a menor distância *dist[u]*.
Para cada vértice vizinho v de u:

Calcule a distância *dist[v]* se o caminho passando por u for mais curto do que o valor atual de *dist[v]*.
Se *dist[v]* for atualizado, remova a versão anterior de v da fila e insira a nova versão com a distância atualizada.

Resultado:

Após a iteração, o vetor dist conterá as distâncias mínimas de origem para todos os outros vértices no grafo.

5. CONCLUSÃO

O problema dos caminhos mínimos é fundamental na teoria dos grafos, com aplicações em diversas áreas, desde sistemas de navegação até logística e redes de telecomunicações. Este artigo explorou a modelagem matemática do problema, suas aplicações e se concentrou no algoritmo de Dijkstra, amplamente utilizado para encontrar caminhos mínimos em grafos com arestas ponderadas não negativas.

O algoritmo de Dijkstra é uma ferramenta poderosa para resolver problemas de otimização em grafos, e sua implementação foi apresentada em detalhes, desde a estrutura do código até o funcionamento em baixo nível. É importante destacar que o algoritmo de Dijkstra é adequado para grafos com arestas não negativas, o que o torna eficiente em muitos cenários do mundo real.

Em resumo, o problema dos caminhos mínimos é um desafio relevante e amplamente aplicável, e o algoritmo de Dijkstra é uma abordagem valiosa para sua resolução. À medida que a tecnologia e as necessidades de otimização continuam a evoluir, o entendimento e a aplicação desses conceitos desempenham um papel essencial na solução de problemas práticos em áreas diversas, tornando-se uma contribuição significativa para o avanço da ciência e da tecnologia.

6. REFERÊNCIAS

BONDY, J. A.; MURTY, U. S. R. Graph Theory with Applications. Editora: ELSEVIER, 1982.

FEOFILOFF, PAULO. Algoritmos para grafos: Caminhos de comprimento mínimo. Disponível em: https://www.ime.usp.br/~pf/algoritmos_para_grafos/aulas/shortestpaths.html. Acesso em: 30/10/2023.

A. DE OLIVEIRA, VALERIANO; RANGEL, SOCORRO; ARAUJO, S. A., PAULO. Teoria dos Grafos: Caminho mínimo - Algoritmo de Dijskstra. Disponível em: https://www.ibilce.unesp.br/Home/Departamentos/MatematicaAplicada/docentes/socorro/aula_dijkstra_2016_socorro.pdf. Acesso em: 30/10/2023.

Algoritmo de Dijkstra para cálculo do Caminho de Custo Mínimo. Disponível em: https://www.inf.ufsc.br/grafos/temas/custo-minimo/dijkstra.html. Acesso em: 30/10/2023.

WILHELM, V. E. Problema do Caminho Mínimo. Disponível em: https://docs.ufpr.br/~volmir/PO_II_10_caminho_minimo.pdf. Acesso em: 30/10/2023.