UFMG
UNIVERSIDADE FEDERAL DE MINAS GERAIS

Universidade Federal de Minas Gerais

Turma: Ciência da Computação **Prof.:** Vinícius

Nome: Rubia Alice Moreira de Souza
Matrícula: 2022043507

Aritmofobia

Belo Horizonte
2023

1. Introdução

O problema a ser resolvido consiste em encontrar o menor caminho em um grafo não direcionado sendo que na construção desse menor caminho não poderá ser considerada nenhuma aresta com peso ímpar e não poderá haver uma quantidade ímpar de arestas percorridas.

Todo o código e histórico de commits durante o desenvolvimento do trabalho podem ser acessados no repositório: https://github.com/Rubia-Souza/UFMG-ALG-Dijkstra. Este repositório só ficará público após a data de entrega do trabalho.

2. Modelagem

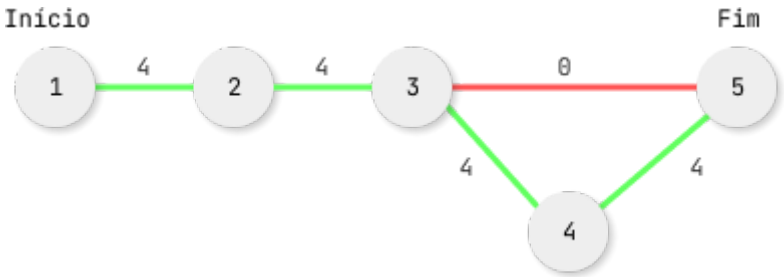
A estrutura utilizada foi um grafo não direcionado com pesos em suas arestas. Para implementar a representação dele, foi feita uma lista de adjacência por meio da estrutura padrão Vector. Assim, o grafo era, basicamente, uma lista de vértices, sendo que cada vértice possui uma lista de arestas, que carregam consigo o vizinho no qual ele está associado e o peso dela.

Para encontrar o menor caminho no grafo, foi utilizado o algoritmo de Dijkstra, uma vez que os pesos das arestas nunca são negativos. O algoritmo foi ajustado para atender as condições impostas pelo enunciado.

2.1. Solução 1

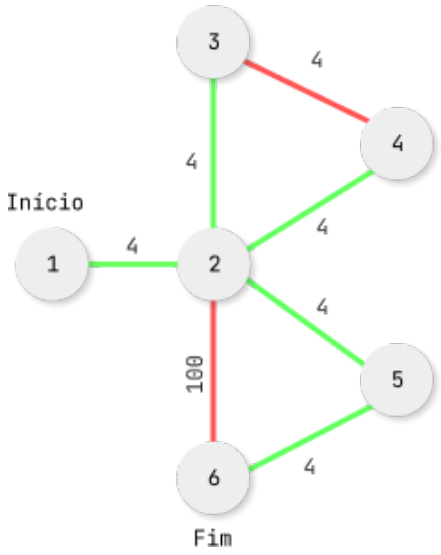
A fim de atender a primeira condição, que consiste em impedir que qualquer aresta ímpar seja considerada na construção do menor caminho, foi adicionada uma verificação quando o algoritmo de Dijkstra está avaliando os vizinhos do menor vértice encontrado até então. Neste ponto, se a aresta possui um valor ímpar, o loop pula para a próxima iteração sem atualizar as estimativas do vértice.

Já para implementar a segunda condição, foram necessárias duas tentativas. Na primeira, também foi adicionada uma condição quando se está avaliando os vizinhos do menor vértice. Nela, era verificado se a contagem de passos obtida até agora era um valor par e se o próximo vértice era o último. A contagem de passos era um contador atualizado sempre que um elemento era removido do heap mínimo. Essa condição era capaz de lidar com casos simples, como este:



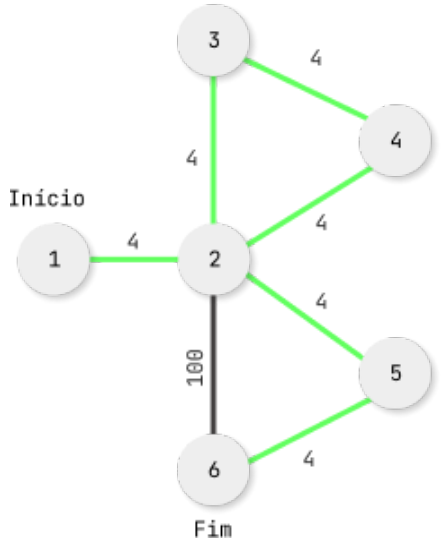
Nesse caso, o algoritmo estaria com o contador de passos em 2 ao chegar no vértice 3. Quando verificasse que o 5 seria o ponto final, mesmo tendo o menor caminho, o algoritmo iria pular para a próxima iteração, pois até neste momento ele deu uma quantidade par de passos e o próximo faria com que tivesse passado por uma quantidade ímpar de arestas.

No entanto, para casos como o seguinte, o algoritmo não detecta o caminho válido, mesmo que ele exista:



Neste caso, o algoritmo definiria que o menor caminho para o 2 é 4. Depois, ao analisar os vizinhos do 2, os vértices 3, 4 e 5 teriam estimativa 8 e o 6 teria 104. Por fim, avançaria para o vértice 5 a partir do 2, mas não atualizaria o 6, uma vez que ele chegaria ao vértice final com uma quantidade de passos ímpar. Assim, o menor caminho, supostamente, seria 104.

No entanto, o menor caminho na verdade é 1 → 2 → 3 → 4 → 2 → 5 → 6, totalizando 24:



Esse problema ocorre, pois o algoritmo de Dijkstra não reconsidera um vértice após ele ter sido removido do heap. Assim, o 2, que aparece no ciclo 1 → 2 → 3 → 4 → 2 → 5 → 6, não é utilizado novamente para atualizar o menor caminho para o 6.

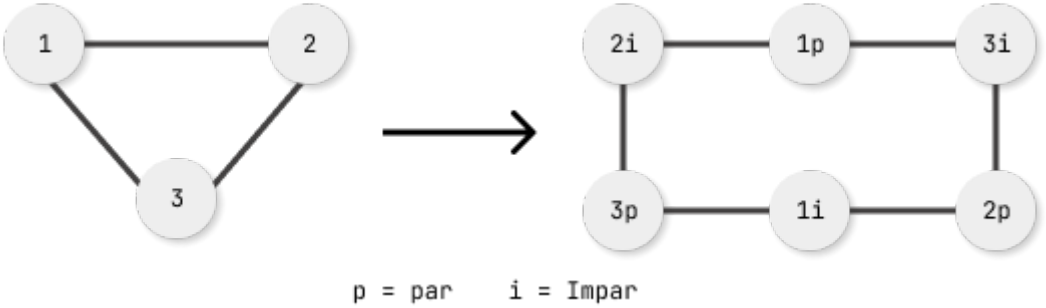
2.2. Solução 2

Ao invés de utilizar o contador de passos e a condição associada a ele, foi feita uma manipulação na estrutura do grafo de modo que cada vértice foi duplicado.

Para cada vértice “v”, foram criados outros dois tipos, um vértice “v par” e um vértice “v ímpar". Com isso, agora o grafo possui um vetor de “2n” posições, sendo “n” a quantidade de vértices. De 0 a “n-1” são alocados todos os vértices pares. Já de “n” até “2n-1”, todos os vértices ímpares. Assim, para acessar um vértice ímpar a partir de um par, basta somar “n” ao seu index.

Além disso, para cada aresta que conecta um vértice “v” a um vértice “u”, no novo grafo há uma conexão de “v par” para “u ímpar" e de “v ímpar" para “u par”.

Seguindo essa construção para um grafo como o abaixo, teremos o seguinte resultado:



Ao reconstruir o grafo dessa forma, você garante que ou haverá um caminho com quantidade par de arestas de um vértice par para outro par ou não haverá nenhum caminho.

Assim, ao executar o algoritmo de Dijkstra no grafo derivado partindo do ponto inicial par até o ponto final par, só haverá caminhos com quantidade par de arestas. Por outro lado, para todo caminho que começa em um vértice par e termina em um vértice ímpar (ou o contrário), haverá uma quantidade ímpar de arestas percorridas.

3. Complexidade

A complexidade do algoritmo de Dijkstra permanece inalterada, pois foi adicionada apenas uma verificação condicional que possui custo constante: O((m+n)*log(n)).

Já para a construção do grafo derivado, é necessário alocar cada uma das “2n” posições: O(2n) = O(n).

Além disso, há o custo de leitura e criação das arestas entre os vértices, que é um valor de “m”: O(m).

Portanto, no fim temos a complexidade:

$$O(n) + O(m) + O((m+n)*\log(n)) = O((m+n)*\log(n))$$

4. Referências

Geeks for Geeks. Shortest Path with even number of Edges from Source to Destination. Disponível em: <https://www.geeksforgeeks.org/shortest-path-with-even-number-of-edges-from-source-to-destination/>. Acesso em: 26/04/2023.

Stack Exchange. Find the shortest path that goes through an even number of red edges. Disponível em: <https://cs.stackexchange.com/questions/125993/find-the-shortest-path-that-goes-through-an-even-number-of-red-edges>. Acesso em: 26/04/2023.

Stack Overflow. Find a minimum distance with a number of even green edges. Disponível em: <https://stackoverflow.com/questions/61989234/find-a-minimum-distance-with-a-number-of-even-green-edges>. Acesso em: 26/04/2023.

Stack Overflow. Shortest path with even number of edges. Disponível em: <https://stackoverflow.com/questions/32722448/shortest-path-with-even-number-of-edges>. Acesso em: 26/04/2023.