Trabalho Prático 1

Thaís Ferreira da Silva - 2021092571

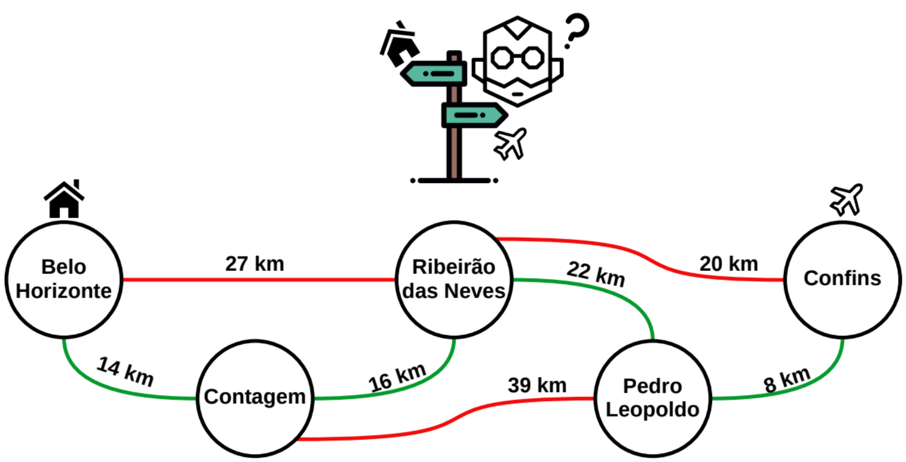
Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte - MG - Brasil

thaisfds@ufmg.br

1 Introdução

O problema proposto para esse trabalho prático foi a modelagem de um algoritmo que ajudasse Steven Jodds, um empresário que sofre de fobia a números ímpares, a planejar suas viagens de carro de maneira eficiente.

O algoritmo deve determinar o menor caminho entre duas cidades, considerando que Steven só viaja entre duas cidades adjacentes se a estrada que as conecta tiver comprimento par e o caminho traçado pelo algoritmo deve passar por um número par de estradas. Para solucionar esse problema, o algoritmo escolhido foi o Dijkstra, um dos algoritmos mais eficientes para encontrar o menor caminho em um grafo.



2 Modelagem

2.1 Estruturas

A estrutura utilizada para a implementação do código foi um Grafo Não Direcionado modificado para auxiliar na busca por caminhos que passam por um número par ou impar de estradas. Através dessa estrutura é possível armazenar as cidades como vértice, e as estradas como arestas.

- **Grafo:** Para que a estrutura fosse eficiente na busca por caminhos que passam por um número par de estradas, foi necessário a realização de algumas modificações. A logica por trás do desenvolvimento dessa nova estrutura pode ser facilmente comparada com a operação de multiplicação entre números positivos e negativos.
  - A multiplicação de dois números de mesmo sinal gera um resultado positivo
  - A multiplicação de dois números de sinais opostos gera um resultado negativo

  Sendo assim, também é possível encontrar uma relação semelhante entre a soma de dois números, levando em consideração a classificação como par ou impar.
  - A soma de dois números de mesma classificação sempre vai gerar um numero par
  - A soma de dois números de classificação oposta vai gerar um número impar

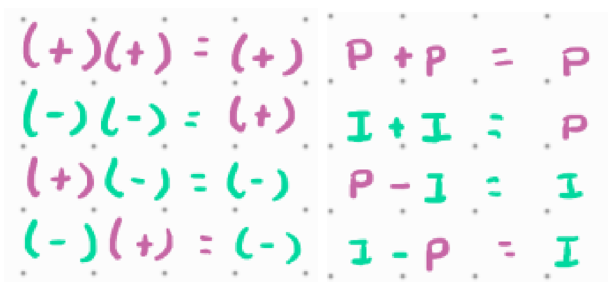


Figura 1: Relação entre as operações

  Tendo isso em mente, pode-se criar um gráfico duplicado capaz de facilitar a analise do número de estradas percorridas entre duas cidades, onde:
  - O grafo possui o dobro de vértices e arestas que o original;
  - O número de estradas percorridas será par quando a origem e o destino estiverem do mesmo lado do grafo. Origem par tem uma conexão com o destino par, ou a origem impar tem uma conexão com o destino impar;
  - O número de estradas percorridas será impar quando a origem e destino estiverem em lados opostos do grafo. Origem par tem uma conexão com o destino impar, ou a origem impar tem uma conexão com o destino par.

  Por fim, o grafo utilizado recebe apenas as arestas de distancia par e é inicializado armazenando o número de vértices original e o número de vértices duplicados, e ao adicionarmos uma aresta entre dois pontos com distancia D, criamos uma lista de adjacência capaz de armazenar todos os vértices do grafo e um par contendo os vizinhos desse vértice e a distancia até ele. Para adicionar uma aresta entre v e u é necessário criar:
  - uma aresta de v par para u impar, e
  - uma aresta de v impar para u par

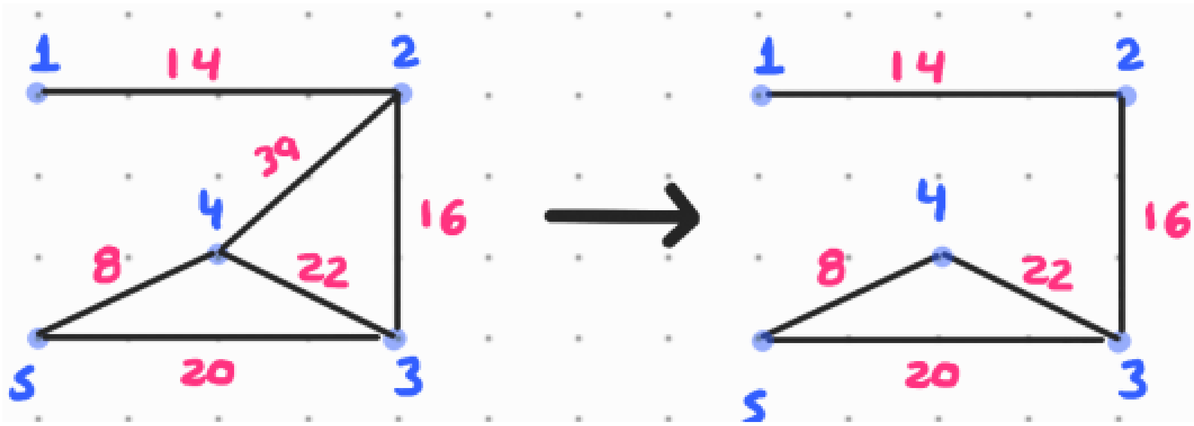


Figura 2: Remoção das arestas de peso impar

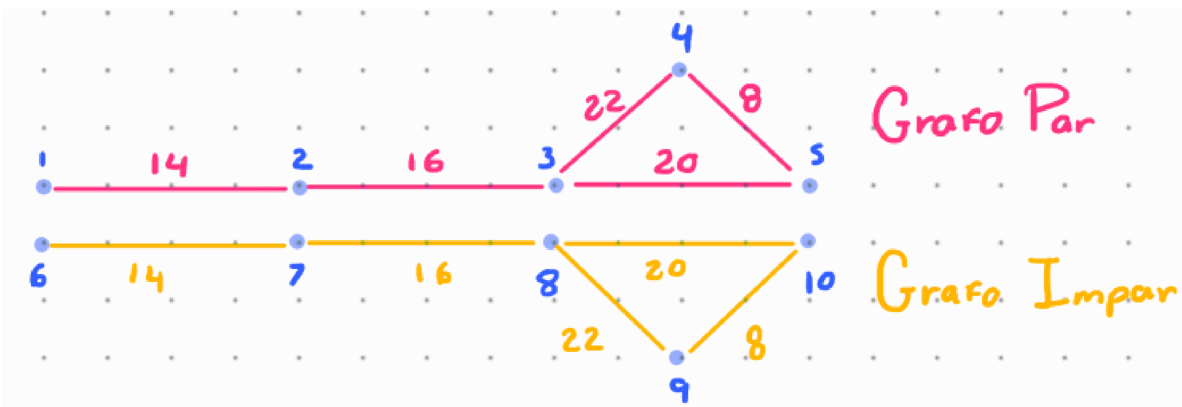
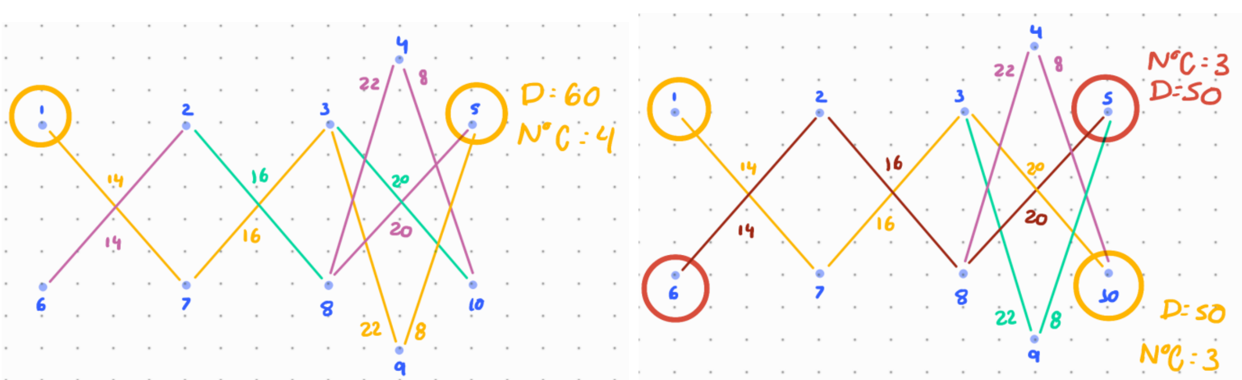


Figura 3: Duplicação do grafo



(a) Caminho mínimo par

(b) Caminho mínimo impar

Figura 4: Caminhos mínimos

2.2 Algoritmos

O algoritmo utilizado para resolução do problema é o Dijkstra visto em sala de aula, pois ele é capaz de encontrar os menores caminhos de um vértice para todos os outros vértices de um grafo.

- **Dijkstra:** O algoritmo é inicializado recebendo um grafo G já construído, e possui uma função capaz de calcular o menor caminho de um vértice de origem s para um vértice de destino d. Seguindo o pseudocódigo abaixo estudado ao longo das aulas, ele é capaz de calcular todos os menores caminho de s para todos os outros vértices de G, e depois retorna apenas a menor distancia entre s e d.

PSEUDOCODIGO DIJKSTRA

```
Inicia o método criando a fila de prioridade e o vetor de distancias
Adiciona a origem na fila e armazena a distancia para a origem como 0
Repete n vezes ou enquanto a fila não for vazia
   Acha v desmarcado com menor d[v]
   Marca v como visitado
   Para cada vizinho w de v
      Se w é desmarcado e d + c(v,w) < d[w]
         d[w] = d + c(v,w)
         Atualiza(w,d[w])
Retorna o caminho se houver, caso contrário retorna -1
```

Para execução do dijkstra foram utilizadas as estruturas da biblioteca padrão priorityqueue para construção da fila de prioridade, list para manipulações no grafo e vector para armazenar a distância da origem para cada vértice. Essa escolha é devido a praticidade e facilidade da utilização do std na hora de percorrer e analisar o grafo.

3 Análise de Complexidade

3.1 Tempo

Em relação a complexidade de tempo da estrutura grafo implementado podemos concluir que é O(1) devido a utilização da lista de adjacência. Já em relação ao algoritmo Dijkstra depende do número de arestas E e do número de vértices V do grafo. O loop externo, que itera enquanto a fila de prioridade não está vazia, tem uma complexidade de O(V log V), pois cada iteração requer a remoção do menor elemento da fila de prioridade, já o loop interno, que itera sobre os vizinhos de um vértice, tem uma complexidade de O(E) , o que resulta na complexidade final O(E + V log V).

3.2 Espaço

Sobre a complexidade de Espaço da estrutura grafo implementado é O(V+E), onde V é o número de vértices do grafo e E é o número de arestas. Isso ocorre porque é necessário armazenar a lista de adjacência para cada vértice, o que resulta em um espaço proporcional a V+E. Já a complexidade de espaço do algoritmo de Dijkstra é O(V), pois é necessário armazenar um vetor de distâncias e uma fila de prioridade. O vetor de distâncias tem um tamanho igual ao número de vértices do grafo (V) e a fila de prioridade tem no máximo V elementos. Portanto, a complexidade de espaço é proporcional a V.