Trabalho Prático 2

Raissa Miranda Maciel
16 de Junho de 2022
Matrícula: 2020006965

1. Introdução

Em vista do grande número de catástrofes que acometeram o estado de Minas Específicas nos últimos anos e os danos causados nas estradas, a ONG Luz Vermelha encontrou dificuldade ao planejar o caminho das expedições de ajuda. Esta documentação lida com o problema de encontrar o melhor caminho entre duas cidades que maximize a quantidade possível de suprimentos a ser carregada. O problema pode ser modelado com um grafo, com os vértices representando as cidades e as arestas representando as estradas entre elas. O peso das arestas é referente à carga máxima que essa estrada suporta. Dessa forma, a solução utiliza o algoritmo de Dijkstra modificado. Originalmente, esse algoritmo calcula o caminho de custo mínimo entre todos os vértices de um grafo com pesos não negativos a partir de um vértice raiz. A modificação consiste em calcular o calcular o fluxo máximo entre dois vértices, considerando o maior gargalo do caminho. Em outras palavras, está sendo calculado o melhor caminho entre duas cidades de acordo com a carga máxima que as estradas suportam. As seções estão divididas de forma a abordar a modelagem computacional do problema, as estruturas, os algoritmos usados e a análise de complexidade em relação ao tempo.

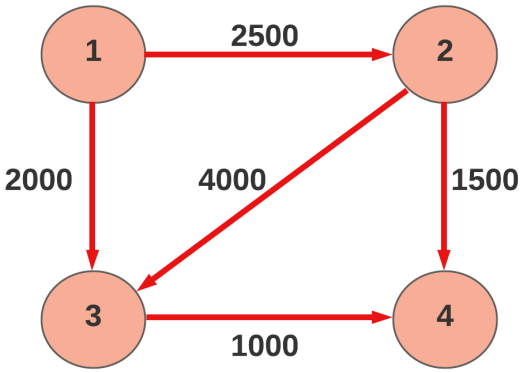
2. Modelagem Computacional do Problema

Essa seção contém a estrutura de dados e o algoritmo usado na solução do problema.

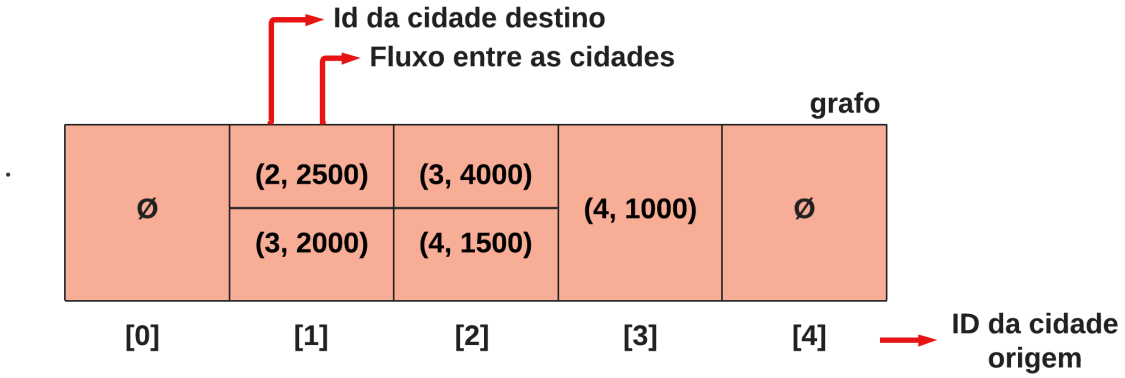
❖ Estrutura de Dados

A primeira parte desse problema consiste em montar o grafo que representa as cidades e estradas a partir do arquivo de entrada. A entrada consiste em uma linha com o número n de cidades, m de estradas e p consultas que serão realizadas, m linhas que informam uma estrada entre duas cidades e sua carga máxima e p linhas que informam as consultas desejadas. Para isso, a estrutura utilizada foi um vetor de vetor de pares que possui todas as estradas do problema. Os índices do primeiro vetor, chamado de grafo, representam o identificador da cidade de origem. Cada cidade possui um vetor de pares e cada par guarda o identificador da cidade que ela alcança e o fluxo entre elas. Um exemplo de entrada é mostrado a seguir com o seu respectivo grafo para melhor visualização.

1 2 2500
2 3 4000
2 4 1500
1 3 2000
3 4 1000



A modelagem para este exemplo ficaria da seguinte forma:

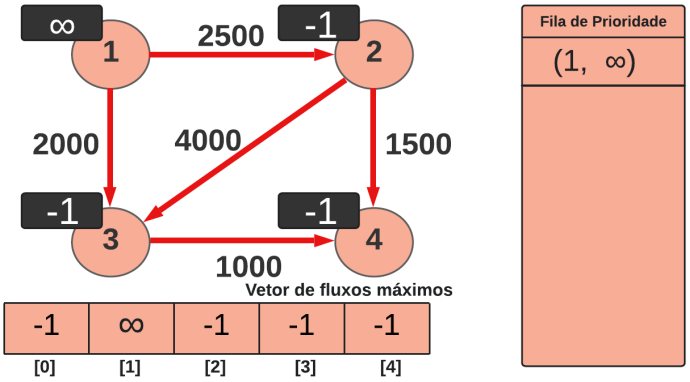


Uma vez tendo o grafo modelado nesse vetor, é possível calcular o melhor caminho entre duas cidades.

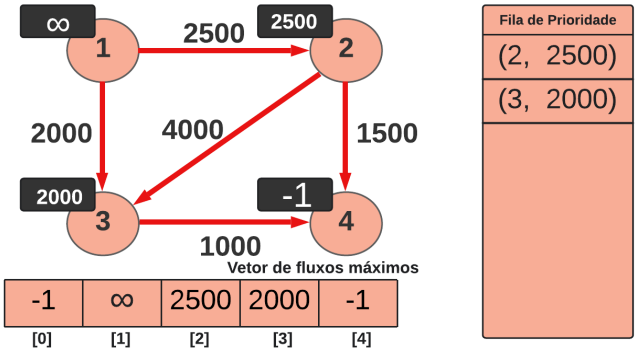
- **Algoritmo de Dijkstra modificado**

Esse algoritmo será utilizado para calcular o caminho que maximize a quantidade de suprimentos entre duas cidades. Para isso, será calculada a maior carga possível de passar por cada cidade (vértice). Será necessário um vetor que guarda a carga máxima de cada cidade no índice respectivo do identificador da cidade. Durante a execução do algoritmo, esses valores serão atualizados sempre que for descoberto um caminho que é possível levar uma carga maior do que o anteriormente descoberto. A ordem das cidades (origem) que terão suas estradas até outras cidades (destino) analisadas é construída por uma fila de prioridade em que os elementos (cidades) prioritários são aqueles que possuem maior fluxo. Sempre que uma cidade possui sua carga máxima atualizada, essa nova cidade é colocada na fila para ter suas estradas analisadas. Sempre que uma cidade já possui todas suas estradas analisadas, ela é retirada da fila. Dessa forma, todas as cidades serão analisadas e terão suas cargas máximas atualizadas até que todas possuam, garantidamente, a maior carga possível. Pelo fato de o algoritmo sempre escolher o vértice com maior fluxo é considerado guloso e, apesar de nem sempre levarem a resultados ótimos, sempre obtém o caminho de maior fluxo. É realizado a seguir o algoritmo para o exemplo simples criado anteriormente, mostrando o vetor com o fluxo máximo de cada cidade sendo atualizado, a fila de prioridade e o grafo. A consulta em questão será (1, 4).

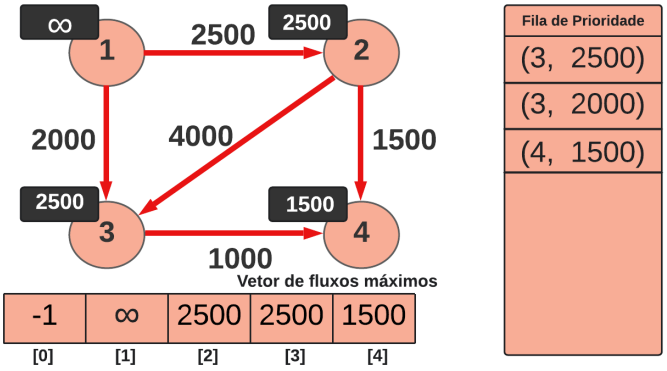
O algoritmo começa inicializando a cidade origem com o fluxo máximo infinito, já que nenhuma estrada chega até ela e limita sua carga. Todas as outras cidades são inicializadas com uma carga inválida. A fila de prioridade recebe a cidade de origem como primeiro elemento.



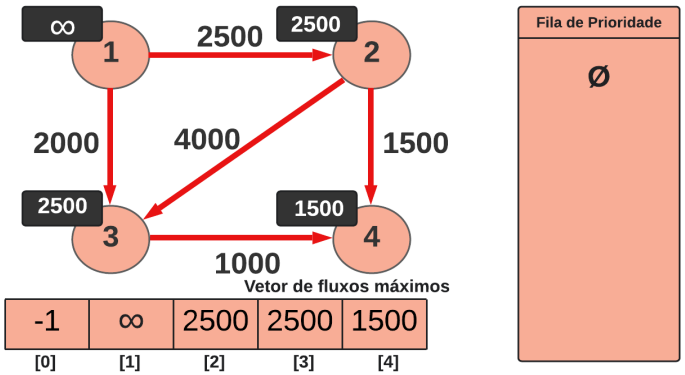
A cidade atual sendo visitada é a 1, então as cidades alcançadas por ela serão acessadas via sua estrada e atualizará o valor das cidades 2 e 3 com o maior valor entre a carga atual das cidades destino (presente no vetor de fluxo máximo) e o mínimo entre a carga da cidade de origem e a capacidade da estrada. Em seguida, as cidades atualizadas são colocadas na fila de prioridade. Como todas as cidades alcançadas pela 1 já foram visitadas, ela é tirada da fila de prioridade.



A cidade de origem agora será a primeira da fila de prioridade, a cidade 2. Assim, todas as suas estradas serão analisadas e os valores atualizados de acordo com as condições explicadas anteriormente. A fila de prioridade possui seu primeiro elemento excluído.



A cidade de origem agora será a primeira da fila de prioridade, a cidade 3. Assim, todas as suas estradas serão analisadas e não terá nenhum valor atualizado. A fila de prioridade possui seu primeiro elemento excluído. Como o próximo elemento da fila de prioridade também é a cidade 3 com um fluxo desatualizado, essa iteração também não atualizará nenhum valor. Por fim, sobrará apenas a cidade 4 na fila de prioridade e, como ela não possui nenhuma estrada que parte dela, não haverá nenhuma alteração. Dessa forma, as estruturas finais ficarão com os seguintes valores:



Portanto, como a cidade 4 é a cidade destino, basta acessar a posição 4 do vetor de fluxos máximos e o valor será a maior carga possível de passar por essa cidade após a análise de todos os caminhos feito pelo algoritmo.

O pseudocódigo desta implementação é mostrado a seguir:

Dijkstra (grafo, cidade_origem, cidade_destino){
Cria vetor de fluxos máximos inicializado com -1
Cria fila de prioridade
Adiciona cidade_origem na fila de prioridade
Atualiza índice da cidade origem para infinito
Enquanto a fila de prioridade não estiver vazia:
Pega primeiro elemento da fila
Visita e atualiza o fluxo de todas as cidades que ele alcança
Tira esse elemento da fila
Retorna o fluxo máximo no índice da cidade_destino
}

3. Análise de complexidade temporal

A análise de complexidade assintótica considera as variáveis n para o número de cidades (vértices), m para o número de estradas (arestas) e p para o número de consultas. Operações constantes como de declaração, atribuição, comparação, impressão e retorno são consideradas realizadas em tempo O(1).

A primeira parte do programa constitui a construção do grafo, ou seja, a leitura do arquivo de entrada. Para isso, há um laço (trecho linha 23) que percorre as m linhas do arquivo de entrada, referente às estradas e armazena em *grafo* as informações lidas. Essa leitura, portanto, é feita em tempo O(m). Em seguida, há a leitura das linhas referentes às p consultas (trecho linha 31) e o armazenamento das informações lidas no vetor *consultas*. Dessa forma, isso é feito com complexidade O(p). A leitura da entrada é, portanto, feita com complexidade assintótica de tempo O(m+p).

A segunda parte do programa constitui o algoritmo de Dijkstra modificado. O algoritmo é realizado para cada uma das p consultas com o uso de um laço (trecho linha 39).

O desempenho dependerá exclusivamente da forma como a fila de prioridades foi implementada. A fila de prioridade utilizada no programa simula um *heap* e, para a construção dele (trecho linha 47), o custo é O(n). O corpo do anel *while* (trecho linha 54) é executado no máximo n vezes e, o procedimento de retirar um novo elemento, considerando a reorganização dos seus elementos possui complexidade O(log n). O laço interno (trecho linha 63) é responsável por percorrer todos os vértices que um determinado vértice alcança (as estradas

que saem da cidade atual que está sendo analisada), ou seja, ele é executado no máximo n -1 vezes, se a cidade possui estrada para todas as outras, resultando em complexidade O(n). Quando são achadas cidades em que a carga máxima corrente dela é menor do que o caminho acabado de analisar, a carga máxima é atualizada e será necessário adicionar essa nova cidade com o valor atualizado na fila de prioridades, explicado anteriormente que pode ser feito a tempo O(log n). Logo, esse laço interno possui complexidade O(n*log n). Com essa análise, o tempo total para executar o algoritmo de Dijkstra modificado é O(n*n*log n) = O(n² * log n). Como essa execução é feita para p consultas, a complexidade total para todas elas é $O(p*n^2*\log n)$.
O programa como um todo, com a leitura da entrada e a execução do algoritmo para todas as consultas terá complexidade assintótica de tempo O(m+p) + $O(p*n^2*\log n)$ = $O(p*n^2*\log n)$.

4. Bibliografia

ZIVIANI, Nivio. **Projeto de Algoritmos: com implementação em PASCAL e C - 3° edição revista e ampliada.** Seção 7.9: Caminhos mais curtos. São Paulo: Cengage Learning, 2011.

PANCHAL, Aakash. **Widest Path Problem | Practical application of Dijkstra's Algorithm.** 2021. Disponível em https://www.geeksforgeeks.org/widest-path-problem-practical-application-of-dijkstras-algorithm . Acesso em: 15 de Jun de 2022.

OLIVEIRA, Marcos. **C++ - Utilitários std::pair e std::tuple.** 2021. Disponível em: https://terminalroot.com.br/2021/08/cpp-pair-e-tuple.html. Acesso em: 15 de Jun de 2022.

5. Instruções para execução:

Para realizar a instalação do programa, é necessário descompactar o arquivo 2020006965_RaissaMirandaMaciel.zip. Uma vez descompactado, basta acessar o diretório em que o programa foi armazenado:

> *cd <diretoriodestino>*

Agora, é necessário realizar a compilação e execução do programa informando um arquivo *.txt* de entrada:

> *g++ tp2.cpp -o tp02*
> *./tp02 < <arquivoentrada.txt>*

Será mostrado na linha de comando os fluxos máximos para cada consulta do arquivo de entrada.