Trabalho Prático

Gabriel Teixeira Carvalho

1. Introdução

Esta documentação lida com o problema de descobrir a capacidade máxima de carga que pode ser transportada entre dois pontos. O objetivo principal desta tarefa é criar um programa que, dado um mapa de cidades e a capacidade das estradas que conectam estas cidades, consiga descobrir qual é o máximo de carga que um caminhão consegue transportar entre 2 cidades. Para resolver o problema citado, foi utilizado um algoritmo de natureza gulosa em cima de uma representação da malha rodoviária como um grafo direcionado.

A seção 2 desta documentação trata sobre a modelagem computacional do problema, enquanto na seção 3 são apresentadas as estruturas de dados e algoritmos utilizados e o pseudocódigo do sistema. Por fim, na seção 4 são apresentadas análises de complexidade assintótica de tempo, seguida por um manual de compilação e execução do programa.

2. Modelagem computacional do problema

A modelagem computacional desse problema parte da conversão da malha rodoviária dada para uma lista de adjacências de um grafo e, a partir disso, da seleção gulosa das arestas de maior peso do grafo para serem consideradas. Essa seleção e ordenação é feita através de uma fila de prioridade que mantém as próximas arestas a serem checadas em ordem decrescente de capacidade. Então, as capacidades que podem ser enviadas do vértice inicial até todos os outros vértices são mantidas em um vetor e, ao encontrar uma aresta que é incidente ao vértice final, a capacidade máxima do vértice de origem até esse vértice de destino é retornada. No exemplo abaixo, o caminho encontrado entre os vértices 1 e 9 é 1-2-5-8-9, com capacidade máxima 1500.

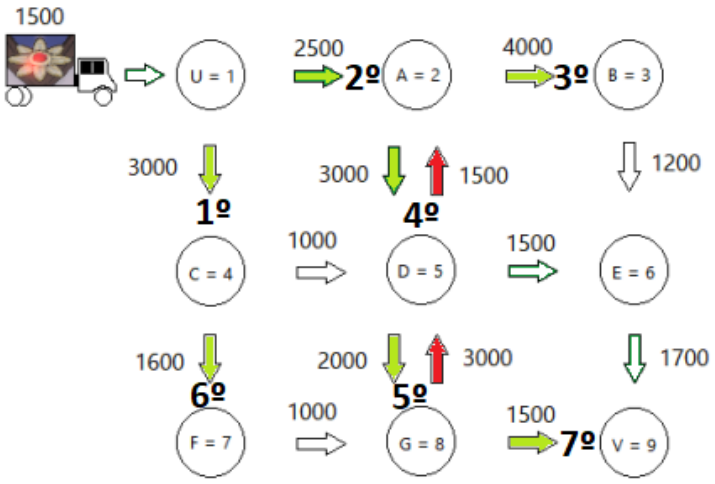

Figura 1: Exemplo de execução do algoritmo

3. Estruturas de dados e algoritmos utilizados

Neste trabalho prático utilizei das seguintes estruturas de dados e algoritmos:

- **Estruturas de dados:**

- Struct: Utilizada para armazenar as arestas que representam as rodovias entre as cidades, contendo os identificadores das cidades de origem e de destino e a capacidade máxima que pode ser transportada nessa estrada.
- Vector (STL): Utilizado bastante ao longo do código para representar o grafo da malha rodoviária, o vetor de capacidades entre as cidades, vértices já visitados, etc. Muito útil pelo seu acesso rápido aos elementos e fácil utilização.
- Priority Queue (STL): Utilizada para guardar e controlar as próximas arestas a serem analisadas pelo algoritmo, mantendo-as ordenadas de acordo com a capacidade da aresta com um custo computacional relativamente baixo.

- **Algoritmos:**

O algoritmo utilizado nesse programa para atribuição das capacidades máximas de carga entre as cidades é de natureza gulosa, pois, a cada iteração, faz a melhor escolha possível sem considerar as próximas iterações e chega em um resultado correto ao final.

O algoritmo implementado calcula a capacidade máxima de carga de uma cidade de origem para todas as outras cidades da malha. Para quaisquer dois vértices adjacentes, é trivial saber qual a capacidade de carga máxima entre eles: é o mínimo entre a capacidade que chega até o vértice de origem da aresta e o peso da aresta que une os dois vértices (para o vértice de origem da consulta, a capacidade que chega nele é definida como um valor muito grande para que o peso da aresta seja sempre a opção escolhida).

Após tomar um caminho, as arestas que partem do vértice de destino são adicionadas à fila de prioridade e ordenadas pelo peso pela estrutura de dados. Dessa forma, o algoritmo progride tomando cada aresta em ordem decrescente de peso até chegar no vértice de destino e, caso a aresta tenha um peso maior do que a capacidade atual que chega em um vértice, atualiza-se a capacidade seguindo a regra mencionada acima.

Cada aresta pode ser considerada apenas uma única vez e todas as arestas reversas são ignoradas, pois, qualquer caminho que volte a um vértice de origem poderá apenas manter a mesma capacidade original ou diminuí-la. Além disso, o algoritmo só executa enquanto não chegou ao vértice de destino e, como é garantido que há um caminho entre todos os vértices, o algoritmo termina.

É possível provar a corretude do algoritmo por contradição. Suponha, por contradição, que ao final do algoritmo o valor calculado para um caminho não seja máximo. Portanto, o caminho que chega primeiro ao vértice de destino possui arestas com valores menores que o valor máximo. Porém, como escolhemos sempre as maiores arestas possíveis, isso é uma contradição, pois, as arestas que diminuem o valor máximo seriam preteridas por outras com valor maior, formando um caminho com apenas arestas maiores ou iguais ao valor máximo. Logo, a premissa assumida é falsa e o caminho escolhido pelo algoritmo é máximo.

3.1. Pseudocódigo

```
Programa () {
  -   Recebe o número de cidades
  -   Recebe o número de rodovias
  -   Recebe o número de consultas
```

```
    -    Recebe arestas e cria lista de adjacências das
    cidades
    -    Recebe uma consulta, com vértice inicial e final
    -    Chama a função getCapacitiesFromNode que calcula
    a capacidade máxima do caminho
    -    Imprime a capacidade máxima do caminho
}
getCapacitiesFromNode(verticeInicial,
verticeFinal, listaDeAdjacencia,
numeroDeCidades) {
    -    Inicializa vetor de vértices descobertos todo
    como      falso
    -    Inicializa vetor com capacidades máximas do
    verticeInicial até todos os outros vértices do
    grafo. Inicia todos os elementos com capacidade 0
    -    Inicializa capacidadeMaxima = 0
    -    verticeInicial é descoberto e tem capacidade
    100000
    -    Inicializa fila de prioridade com as arestas do
    verticeInicial
    -    Enquanto (fila não está vazia) {
              -    Pega a aresta com a maior capacidade
              no topo da fila
              -    Se o vértice de destino da aresta é um
              vértice já visitado, pula a iteração
              -    Se a capacidade da aresta é maior que a
              capacidade atual do vértice de destino,
              atualiza a capacidade do vértice de destino
              com o mínimo entre a capacidade atual do
              vértice de origem e o peso da aresta
              -    Se o vértice de destino da aresta é o
              vértice final consultado, a capacidadeMaxima
              recebe a capacidade atual que chega no
              vértice de destino e é retornada
              -    Aresta é marcada como visitada
              -    Para cada aresta que sai do vértice de
              destino, adiciona-a na fila de prioridade
         }
    -    Retorna a capacidadeMaxima
}
```

4. Análise de complexidade de tempo

Dado que o número de rodovias seja E e o número de consultas seja Q, a complexidade de tempo dos algoritmos usados no código são:

getCapacitiesFromNode: Esta função coloca as arestas que saem do vértice inicial em uma fila de prioridade e, enquanto a fila de prioridade não está

vazia ou o vértice de destino não foi encontrado, retira uma aresta do topo da fila de prioridade (O(log(E))), checa condições e atualiza um vetor (O(1)) e coloca novas arestas na fila de prioridade, o que requer um rearranjo de sua estrutura (O(log(E))). Logo, essa função possui complexidade O(E*log(E)).

Programa: Receber as entradas (O(1)) e salvar a lista de adjacências (O(1) para cada aresta) possui complexidade O(E). Então, para cada consulta, os valores são recebidos (O(1)), a função getCapacitiesFromNode é chamada (O(Q*E*log(E))), e um valor é impresso (O(1)). Portanto, o programa possui complexidade O(E + Q*E*log(E)) = O(Q*E*log(E))**.**

Instruções para compilação e Execução

1 – Extraia o arquivo `.zip` na pasta desejada.

2 – Execute o comando `'g++ tp02.cpp -o tp02'` no terminal.

3 – Execute o programa `tp02` passando um arquivo de texto com a entrada para o programa pela linha de comando.