Trabalho Prático 2 de Algoritmos 1

Nome: Artur Gaspar da Silva
Matrícula: 2020006388

Modelagem computacional do Problema:

Desejamos encontrar o maior peso que pode ser carregado pelo caminhão, para várias escolhas de ponto inicial e final. Para isso, nós modelamos o problema como um grafo direcionado, em que cada vértice representa uma cidade e cada aresta uma rodovia, além de cada rodovia possuir um limite de peso que elas suportam. Portanto, nosso caminhão não pode passar por uma rodovia cujo limite de peso é menor que a carga que ele carrega.

A implementação desse modelo a partir do input é muito simples e direto: guardamos o grafo fornecido na forma de lista de adjacência, e processamos cada um dos q consultas assim que os lemos, não sendo preciso memória extra para isso.

Dados u, v de uma consulta particular, a resposta do nosso problema foi modelada como sendo o menor inteiro $\boldsymbol{k}$ tal que é possível sair de u e chegar até v passando apenas por arestas de capacidade maior ou igual a k. Para ver que isso equivale ao que queremos, imagine que sabemos o caminho ideal do caminhão, e que x é a menor capacidade de rodovia nesse caminho:

- Se considerarmos apenas as arestas de capacidade maior que x, esse caminho não existirá mais no nosso novo grafo. Caso ainda haja um caminho, ele possui todas as arestas maiores que x e portanto é um caminho que o caminhão poderia ter usado para chegar ao destino, e o caminho original não era o melhor, uma contradição.
- Se considerarmos apenas arestas de capacidade maior ou igual a y, com $y < x$, também estaremos considerando as arestas de tamanho maior ou igual a x, e nosso caminho ótimo estará contido no novo grafo. Portanto, podemos usar ele para alcançar nosso destino.

Estruturas de Dados e Algoritmos utilizados na resolução:

Primeiramente, apresentamos o pseudocódigo do nosso programa (procedimentos auxiliares declarados abaixo do principal):

TP02(grafo direcionado das cidades e rodovias, consultas a serem executadas**)**:
    salvar o grafo como variável global
    Para toda consulta *u,v*:
        ans ← **bin_search(***u*, *v***)**
        imprimir ans

bin_search(orig, dest**)**:
    L ← 1
    R ← 100000
    enquanto L != R:
        M ← (L+R+1)/2
        resetar marcações de vértices visitados
        se **dfs(**orig, dest, M**)** retorna verdadeiro:
            L ← M
        senao:
            R ← M - 1

dfs(vértice atual, destino, mínimo):
se o vértice atual é o destino:
retorna verdadeiro
se o vértice atual já foi visitado antes:
retorna falso
marca o vértice atual como visitado
para cada aresta *e* que sai do vértice atual:
se a aresta *e* tem pelo menos mínimo de capacidade e **dfs(**vértice de chegada de *e*, destino, mínimo**)** retorna verdadeiro:
retorna verdadeiro
retorna falso

Para implementar esse algoritmo, utilizou-se estruturas da Standard Template Library (STL) do C++. Mais especificamente, utilizamos vetores e pares (*vector* e *pair*):

- Lemos a entrada utilizando *cin* e imprimimos a saída com *cout* (ambos da *iostream*);
- Guardamos o grafo numa lista de adjacência implementada como vetor de vetor de par de inteiros, tal que o *i*-ésimo vetor é uma lista de arestas que saem do vértice *i*, representada pelo par vértice de destino, peso da aresta.
- Deixamos um registro de quais vértices já foram visitados em cada vez que a DFS roda em um vetor de booleanos, em que a *i*-ésima posição indica se o vértice *i* já foi visitado.

Análise de complexidade:

Podemos dividir nosso programa nas seguintes etapas e subetapas:

- *Leitura da entrada:*

Para essa etapa, precisamos ler *m* trios de números que indicam as arestas do nosso grafo, e depois *q* pares de números que indicam as consultas. Isso significa que leremos **O(m+q)** inteiros. Como cada inteiro pode ser lido em tempo constante, a complexidade de tempo dessa etapa é de **O(m+q)** .

- *Depth-First-Search:*

Nessa etapa começamos em um vértice do grafo e buscamos chegar a algum outro usando a estratégia de busca em profundidade. No pior caso, teremos que consultar todas arestas do grafo, e note que pra cada vértice diferente do inicial que encontramos precisamos ter passado por uma aresta do grafo antes, e portanto a complexidade assintótica temporal de **O(m)**.

- *Busca Binária na resposta*:

Seja *w* a capacidade máxima dentre todas nossas rodovias. Como a cada passo dessa etapa retiramos pelo menos metade das possibilidades de valor-resposta para consulta em questão, e isso pode ser feito no máximo aproximadamente $log_2(w)$ vezes, teremos nessa etapa **O(log(w))** testes de nossa busca. Como cada teste chama uma DFS, que como vimos, tem complexidade de pior caso **O(m)**, teremos no total **O(m log(w))** como complexidade assintótica de tempo de pior caso nessa etapa.

- *Processamento das consultas:*

O processamento de cada consulta chama o procedimento de busca binária, que como já vimos tem complexidade assintótica temporal de pior caso de **O(m log(w))**, com *n*, *m*, *w* sendo a quantidade de cidades, de rodovias e a capacidade máxima dentre todas nossas rodovias. Como temos *q* dessas consultas, a complexidade assintótica de pior caso dessa etapa é de **O(q m log(w))**.

Ou seja, no total o algoritmo possui complexidade de tempo **O(m+q)+O(q m log(w)) = O(q m log(w))**, lembrando que *m* é o número de rodovias, *q* é a quantidade de consultas, e *w* é a capacidade máxima dentre as rodovias do nosso grafo.