Trabalho 2

Flávio Marcílio de Oliveira

1. Modelagem Computacional do Problema

O problema consiste em encontrar um caminho entre uma cidade de origem e uma cidade de destino que permite transportar o maior peso possível, sabendo que o peso transportado é limitado pelo menor peso permitido em uma rodovia que pertence a esse caminho. Assim, a solução consiste em determinar qual caminho tem o peso limitante o maior possível. Como, pela especificação do trabalho, é possível ter uma ou várias rotas que atendam ao critério estabelecido, a solução será dada pelo peso máximo encontrado para este caminho ótimo.

Para determinar o peso permitido da rota ótima, o sistema rodoviário será modelado como um grafo direcionado e ponderado, onde as cidades serão representadas pelos vértices do grafo e as rodovias serão representadas pelas arestas, com a direção das arestas indicando que o percurso é possível somente nesta direção. O peso permitido em cada rodovia será representado pelos pesos das arestas.

Uma vez construído o grafo, o algoritmo de Busca em Largura (BFS) é utilizado como uma estratégia para percorrer no grafo, a partir de uma origem dada, calculando o peso que poderá chegar em cada vértice tendo como base o peso disponível no vértice pai e o peso da aresta. Ao final do algoritmo BFS adaptado, teremos o maior peso que poderá chegar em qualquer vértice do grafo considerando um caminho ótimo.

2. Descrição da Solução

a. Estruturas de Dados

struct aresta para identificar uma aresta com o vértice origem, o vértice destino e o peso respectivo;

vector<aresta> adj[CIDADES] para representar o grafo por uma lista de adjacência, em que cada posição representa um vértice com as arestas que saem dele;

arestaVisitada[CIDADES][CIDADES] matriz utilizada para identificar cada aresta visitada no algoritmo de Busca em Largura;

vector<int> pesoMaximo para calcular o peso máximo que é possível chegar nos vértices;

b. Algoritmos

bfs(s) - Algoritmo de Busca em Largura utilizado para percorrer todo o grafo a partir da origem ***s***, atualizando os valores dos pesos máximos em cada vértice. O algoritmo foi implementado utilizando uma fila (estratégia FIFO).

Ao iniciar o algoritmo, o vértice de origem é colocado na fila, o vetor ***pesoMaximo*** é inicializado como **0**, significando que neste momento o peso máximo disponível nos vértices

é **0** e a matriz ***arestaVisitada*** também é inicializada como ***false***, indicando que nenhuma aresta foi visitada.

O processo de percorrer nos vértices, assim como no algoritmo convencional, ocorre até que a fila esteja vazia. Assim, o primeiro elemento (***origem***) é retirado da fila e todas as arestas que saem da origem são percorridas. Se a aresta ainda não foi visitada, colocamos o vértice de destino desta aresta na fila e marcamos essa aresta como visitada. Em seguida, determinamos o peso que será enviado para o vértice de destino: se o vértice de origem da aresta analisada corresponde a origem (início do processo) o peso enviado será o peso da aresta, caso contrário, será o peso do vértice de origem (armazenado no vetor ***pesoMaximo***). Logo em seguida, determinamos o peso que chega no vértice de destino: se o peso enviado é maior que o peso da aresta e se o peso da aresta é maior que o peso disponível no vértice de destino, trocamos o peso máximo disponível no vértice de destino pelo peso da aresta; por outro lado, se o peso enviado é menor ou igual ao peso da aresta e o peso enviado é maior que o peso disponível no vértice de destino, trocamos o peso máximo disponível no vértice de destino pelo peso enviado. Se nenhuma destas duas condições forem atendidas significa que o peso no vértice de destino já é o maior que é possível chegar neste vértice. Portanto, ao final do algoritmo teremos o vetor ***pesoMaximo*** com os maiores pesos disponíveis em cada vértice, sabendo que o peso disponível em cada vértice só aumenta. Entretanto, quando o peso disponível em um determinado vértice aumenta, todas as arestas que saem desse vértice devem ser novamente percorridas para possibilitar encontrar um novo caminho que permita um peso maior. Sendo assim, no caso de ocorrer aumento de peso disponível em um vértice, a linha correspondente a este vértice, na matriz ***arestaVisitada***, é novamente preenchida com ***false***.

Ao final deste algoritmo, teremos o vetor ***pesoMaximo*** preenchido com com os maiores pesos que é possível chegar em todos os vértices do grafo partindo de um vértice específico, neste caso ***s***.

O pseudocódigo deste algoritmo está apresentado abaixo:

```
bfs(origem):
      insere origem na fila;
      inicializa o vetor pesoMaximo com 0;
      inicializa a matriz arestaVisitada com false;
      enquanto a fila não estiver vazia faça:
            retira o primeiro elemento da fila;
            para cada aresta saindo do elemento retirado faça:
                  se a aresta ainda não foi visitada faça:
                        coloca o nó destino da aresta na fila;
                        marca a aresta como visitada;
                        se a origem da aresta analisada é a origem
                        faça:
                              peso enviado = peso da aresta;
                        senão:
                              peso enviado = peso disponível no vértice
                        de origem da aresta analisada;

                        se o peso enviado é maior que o peso da aresta
                        e é maior que o peso disponível no vértice de
                        destino da aresta faça:
```

```
        pesoMaximo do vértice de destino = peso
da aresta;
senão se o peso enviado é menor ou igual ao
peso da aresta e é maior que o peso disponível
no vértice de destino da aresta faça:
        pesoMaximo do vértice de destino = peso
enviado;

se o peso disponível no vértice de destino
aumentou e as arestas que saem desse vértice
já foram visitadas faça:
        habilita a busca nestas arestas
novamente;
```

main - Programa principal

No programa principal o arquivo de entrada é lido e o grafo é construído.

Após a criação do grafo, o algoritmo de Busca em Largura (bfs) é rodado para cada consulta solicitada passando como parâmetro o vértice de origem. Para cada consulta, o peso máximo disponível no destino é escrito na saída.

3. Análise de Complexidade de Tempo Assintótica

A análise de complexidade será realizada considerando que o grafo já está construído, pois é um processo que será fortemente afetado pelo tempo de leitura do arquivo de entrada e o acesso a memória secundária supera o processamento utilizado na criação do grafo.

Assim, considerando um grafo com N vértices e M arestas, temos:

A complexidade do algoritmo Busca em largura é O(M + N) considerando que todas as arestas são percorridas apenas uma vez. Entretanto, quando o peso em um vértice que já foi visitado é alterado, este vértice será novamente acessado para que suas arestas adjacentes sejam novamente verificadas com o objetivo de encontrar um novo caminho possível para este novo peso. Porém, esse peso só é alterado, no pior caso, o número de arestas que chegam nele e essa alteração implica revisitar novamente as arestas dos vértices que são alcançadas pelas arestas que saem desse vértice. Assim, no pior caso, um vértice terá M/2 arestas chegando e M/2 arestas saindo implicando M/2 novas visitas nas arestas, resultando uma complexidade $O(M^2 + MN)$.

Todo o pré-processamento realizado - inicialização dos vetores e das matrizes - é no máximo $O(N^2)$.

Portanto, a complexidade global do programa será, para uma consulta, $O(M^2 + MN + N^2)$. Considerando Q consultas, a complexidade será $O(Q(M^2 + MN + N^2))$.

4. Conclusão

A implementação foi desenvolvida utilizando um grafo representado por lista de adjacência por usar a memória de maneira mais eficiente.

O algoritmo de Busca em Largura foi adaptado para calcular o peso máximo disponível em todos os vértices partindo de uma origem específica.

Alguns testes foram realizados e os resultados corretos foram obtidos mostrando a corretude da implementação em solucionar o problema proposto.

Apêndice

O programa foi desenvolvido, compilado e executado em ambiente Linux: WSL: UBUNTU-20.0

A. Compilação

O programa é compilado utilizando o comando ***g++ main.cpp***

B. Execução

A execução é realizada utilizando o comando ***./a.out < input.txt***

onde input.txt é o arquivo com os dados do problema