Trabalho Prático 2
Algoritmos I

Euller Saez Lage Silva

2019054501

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

eullersaez@ufmg.br

1. Modelagem computacional

O trabalho proposto consistia em desenvolver um sistema computacional capaz de resolver o problema de escolher o menor número de caminhos para ligar todos os pontos turísticos de uma dada cidade, de modo que o custo de construção desses caminhos propostos fosse o menor possível, e ao mesmo tempo optando por aquela configuração cuja atratividade acumulada fosse a maior possível, ou seja, sempre que dois caminhos tivessem custos iguais e um deles devesse ser escolhido, deveria-se optar por aquele cuja soma dos valores turísticos de cada ponto sendo ligado fosse maior.

Desse modo, a modelagem mais simples e adequada para a resolução foi por meio de um grafo para representação da cidade, no qual os vértices representavam os pontos turísticos e as arestas os caminhos entre um ponto e outro. Como cada caminho possuía um custo, ou seja, um peso para aresta e cada ponto turístico um valor turístico, essas informações foram também armazenadas. Após estabelecidas as devidas ligações entre os pontos com os possíveis caminhos e seus pesos, bastou transformar esse grafo em uma árvore geradora mínima, ou Minimum Spanning Tree(MST), por meio do algoritmo de Prim realizando as modificações necessárias. A principal diz respeito a considerar a maior atratividade dentre caminhos como um fator de desempate na escolha da aresta quando duas ou mais arestas possuem o menor peso.

2. Estrutura de dados

As estruturas de dados neste trabalho prático consistiram basicamente na utilização de um grafo implementado por matriz de adjacência, conforme uma das sugestões vistas em aula, com pequenas alterações. Apesar de ocupar mais espaço, esse tipo de estrutura e implementação foi muito mais intuitivo e simples de se utilizar pelo fato de que as arestas possuíam pesos. Logo, a existência de uma aresta de um vértice i para j pôde ser representada diretamente pelo seu peso na posição [i][j] da matriz de adjacência. Como pesos negativos não eram permitidos, bastou considerar apenas aqueles elementos maiores ou iguais a 0 para identificar a existência da aresta.

A classe Grafo possui atributos inteiros *numero_vertices, atividade_agregada* e *custo_minimo,* vetores do tipo inteiro *valor_turistico*, responsável por armazenar o valor de

cada um dos pontos e *quantidade_trechos*, responsável por armazenar quantas arestas incidem/saem de cada ponto na MST. Possui também duas matrizes de adjacência do tipo inteiro sendo *matriz_adjacencia* a matriz que representa o grafo completo incicialmente e *matriz_adjacencia_MST* a árvore gerada após a execução do método *prim*.

O construtor da classe recebe um inteiro representando o número de vértices do grafo, zera inicialmente *atividade_agregada* e *custo_minimo*, aloca memória para todos seus vetores e matrizes atributos, zera todos os elementos de *quantidade_trechos* e atribui -1 para todos os elementos das matrizes *matriz_adjacencia* e *matriz_adjacencia_MST*.

O método *set_valor_turistico* recebe a posição de um vértice, seu valor turístico e atribui esse valor na posição do vértice no vetor *valor_turistico*. Analogamente, *adiciona_aresta* recebe as posições i e j dos vértices que serão ligados por uma aresta e o seu custo, que é atribuído na posição *matriz_adjacencia*[i][j] e *matriz_adjacencia*[j][i], pois o grafo não é direcionado.

imprime_saida imprime os atributos *custo_minimo, atividade_agregada* e os elementos do vetor *quantidade_trechos*, além das posições e custos de cada caminho (aresta) na *matriz_adjacencia_MST* .

calcula_atratividade_agregada calcula o valor de *atividade_agregada* ao multiplicar para cada vértice i *quantidade_trechos*[i] por *valor_turistico*[i] e somar todos esses resultados.

Finalmente, *prim* fica responsável por gerar a MST e teve sua implementação baseada na mesma disponível neste link[1], fazendo algumas alterações para que se considerasse a atratividade dos caminhos como fator de desempate quando dois ou mais caminhos possuírem peso mínimo durante a iteração.

```
prim():
  E = 0                          //onde E é o número de arestas
  selecionados[V]                //onde V é o número de vértices
  Para cada elemento i de selecionados
     selecionados[i] = falso
  fim-para
  selecionados[0] = verdadeiro
  linha, coluna                  //variáveis auxiliares para cada iteração
  Enquanto E < V-1
     min = infinito
     atratividade_temp = 0, atrat = 0
     linha = 0, coluna = 0
     Para cada vértice i
        Se i tiver sido selecionado em selecionados[i] então
           Para cada vértice j
```

[1] https://www.programiz.com/dsa/prim-algorithm

```
            atratividade_temp =  valor_turistico[i]+valor_turistico[j]
            Se j não tiver sido selecionado e  matriz_adjacencia[i][j] >= 0 então
                Se  min > matriz_adjacencia[i][j] ou atrat < atratividade_temp e min = matriz_adjacencia[i][j]  então
                    min = matriz_adjacencia[i][j]
                    atrat = atratividade_temp
                    linha = i
                    coluna = j
                fim-se
            fim-se
        fim-para
    fim-se
  fim-para
  quantidade_trechos[linha]++
  quantidade_trechos[coluna]++
  matriz_adjacencia_MST[linha][coluna] = matriz_adjacencia[linha][coluna]
  custo_minimo = custo_minimo + matriz_adjacencia[linha][coluna]
  selecionados[coluna] = true
  E++
fim-enquanto
calcula_atratividade_agregada()
```

---

3. Instruções de compilação e execução

O programa foi desenvolvido na linguagem C++ por meio do Visual Studio Code para Windows da Microsoft Corporation, utilizando o Windows Subsystem for Linux (WSL) e compilado pelo compilador G++ da GNU Compiler Collection.

3.1 Compilação

Para compilar o programa primeiro é necessário abrir o diretório 2019054501_EULLER/src no terminal do linux e digitar o comando “make”.

3.2 Execução

Para executar o programa, após compilado, ainda dentro do mesmo diretório 2019054501_EULLER/src no terminal do linux, basta digitar o comando “./tp02”.

3.3 Limpeza de arquivos temporários

Para limpar todos os arquivos temporários gerados durante a compilação, ainda dentro do mesmo diretório 2019054501_EULLER/src no terminal do linux, basta digitar o comando “make clean”.

4. Análise de complexidade

A complexidade de tempo do programa como um todo será o custo de todas as chamadas dos métodos implementados. Em especial, o construtor e *imprime_saida* sempre têm que percorrer todas as V^2 posições das matrizes de adjacência, onde V é o número de vértices do grafo, seja atribuindo inicialmente -1 aos elementos, seja imprimindo a posição dos vértices e pesos das arestas (custo do caminho na MST). Assim, ambos casos possuem custo quadrático $O(V^2)$.

Se desconsiderarmos o custo da única chamada de *calcula_atratividade_agregada* que é linear $O(V)$ pois percorre V vezes os vetores *valor_turistico* e *quantidade_trechos*, *prim* terá custo $O(E \log V)$, onde E é o número de arestas, pois os seus loops aninhados percorrerão na ordem de log V vértices (o número inicial vai se reduzindo à medida em que mais vértices vão entrando no cutset) E vezes. Logo, temos que

$$O(V) + O(E \log V) + O(V^2) = O(n^2)$$

e portanto a complexidade de tempo do programa é quadrática $O(n^2)$.

References

KLEINBERG, J., TARDOS, É. Algorithm Design. 1st. Edition. Pearson, March 16, 2005.