Samai Soares Pereira **TP01**

2020054811

1. Introdução

O problema proposto foi implementar um algoritmo de casamento estável e distancia em grafos para solucionar o problema de visitantes e bicicletas. O trabalho se baseava em uma situação hipotética no qual a prefeitura gostaria de alocar pessoas a bicicletas em uma região onde havia obstáculos. Para solucionar o trabalho, foram utilizados dois principais algoritmos a BFS para calcular a menor distância entre dois pontos em uma matriz ou grafo e o algoritmo de *Stable Matching* para alocar bicicletas a visitantes baseado no ranking de preferência do visitante e da distância das bicicletas.

2. Implementação

Por instrução do monitor, o programa foi implementado utilizando apenas um arquivo fonte: main.cpp no qual ele é constituído por duas principais funções

```
void CasamentoEstavel(int **Rank, int M, string *matriz,Tupla DimensaoMatriz)
```

Essa função tal como o nome já diz, ela é a responsável pela realização do *Stable Matching,* ela recebe uma matriz de ranks com as preferências de cada visitante, uma variável de dimensões das preferências dos visitantes, uma variável das, uma matriz do grafo do caminho a ser seguido e uma variável da dimensão da matriz grafo.

Essa função é responsável de, através de um loop, realizar o casamento estável entre os visitantes e as bicicletas, uma vez que a bicicleta esteja já alocada a um visitante, ela chama outra função para realizar o cálculo da distância para realizar o desempate.

A segunda função principal do código é a CalculaDistancia

```
int CalculaDistancia(string *matriz, Tupla T, char CiclistaNome, char bicicleta)
```

Essa função realiza um BFS para encontrar a distância mínima entre dois pontos em um grafo, ela recebe como parâmetro uma matriz de strings do grafo, uma tupla relacionada às dimensões da matriz mencionada anteriormente, o nome do Ciclista e da Bicicleta nos quais serão necessários para calcular a distância mínima entre eles.

Outra função essencial do programa foi a função ranking

```
void ranking(int **Rank, int M)
```

No qual ela ordena os índices das preferências de cada ciclista.

3. Complexidade

Para calcular a complexidade do algoritmo vamos analisar cada função separadamente.

Consideremos M como o número de ciclistas para nos basearmos para realizar o cálculo de complexidade.

Iniciaremos a análise através do arquivo main.cpp após casamentoestavel.cpp e finalmente distancia.cpp.

main.cpp:

A Função **preenchematriz** tem complexidade O(M), uma vez que o programa executa M vezes para cada ciclista ele copia a lista de preferências dele.

A Função **ranking** tem complexidade $O(M^3)$ pois, ele inicialmente ele preenche uma matriz auxiliar no qual essa operação tem complexidade $O(M^2)$, na parte da ordenação dos rankings nessa mesma função, ele realiza a operação em $O(M^3)$, uma vez que como na maioria das operações de ordenação, como bubble sort, insertion sort e selection sort, a complexidade desses algoritmos é de $O(M^2)$, para cada vetor de M elementos, uma vez que há M vetores a complexidade se torna $O(M)*O(M^2) = O(M^3)$.

Dessa forma, temos $O(M^2) + O(M^3) = O(M^3)$.

casamentoestavel.cpp

No arquivo casamentoestavel.cpp, temos apenas uma função chamada CasamentoEstavel.

Essa função se inicia com 2 loops de O(M) cada, responsáveis por preencher os nomes das bicicletas e dos ciclistas. Logo após realiza uma operação de casamento estável, realizando o algoritmo de *Stable Matching* em todos visitantes e bicicletas. Esse algoritmo em suma tem complexidade $O(M^2)$, uma vez que ele percorre necessita propor a todos os ciclistas, entretanto, durante essa procura, necessita realizar uma chamada para uma BFS, esse complexidade se estende para $O(M^2) * O(M+E)$ onde E é o caminho entre os vértices.

Dessa forma a complexidade assintótica é $O(M^3)$.

distancia.cpp

Nesse arquivo há uma função principal para realizar o cálculo das distancias, inicialmente há um loop de complexidade O(M+E) pois ele preenche toda a matriz com os vértices e com as arestas do grafo, logo a complexidade dele dependerá da quantidade de vértices e arestas.

Logo após esse loop, é realizado uma pesquisa por distância utilizando BFS, no qual tem uma complexidade também de O(M+E).

Dessa forma, observamos que a complexidade final do programa é de $O(M^3)$.

4. Pseudocódigo

Main:

PreencheMatriz(Matriz, M);

Ranking();

```
CasamentoEstaveç();

PreencheMatriz(Matriz,M)
{
While(i<M)
preenche string Matriz[i];
}

Ranking(Rank,M)
{
For(i<M)
For(J<M)
PreencheMatrizAuxiliar[i][j];
}
For(i<M)
        For(j<M){
                While(aux<M)
                {
                Procura maior;
                Indice = indice maior;
                }
                Rank[i][j]= indice;

        }

        CasamentoEstavel(Rank,M)
        {
        For(i<M)
                {
                Preenche matriz ciclista;
                Preenche matriz bicicleta;
```

```
        }

        For(i<M){
        For(j<M){
        if(ciclista[i] estiver casado)
        break;
        if(ciclista não está casado)
                                {
                                If(bicicleta não está casada){
                                        Aloque a bicicleta ao ciclista
                                        }
                                If(bicicleta não esta Casada)
                                {
                                CalculaDistancia(entre o ciclista já alocado e novo ciclista)
                                If(distancia ciclista novo<distancia ciclista alocado)
                                {
                                alocar o ciclista novo à bicicleta;
                                ciclista antigo = não casado;
                                i=-1;
                                break;
                                                }
                        }
                }
        }
}

CalculaDistancia(Matriz, dimensão matriz)
{
TrataMatriz
        {
                Preenche a matriz de Grafos com a Tupla Bicicleta-Ciclista desejado;
```

```
            Retorna posição de cada tupla na matriz;
        }
        While(fila!=vazia)
        {
        Algoritmo Realiza de BFS para achar menor distancia na fila
        If(filacopia[i][j]==bicicleta){
            Retorna posição;
            }
        }
        Retorna –1;
}
```

5. Instruções para Compilação

Para compilar o código necessita apenas executar o código na pasta do arquivo:

g++ main.cpp -o main

E para executar o programa, após compilado na linha de comendo escrever

main.exe < arquivoentrada.txt