Routing Algorithm for Ocean Shipping and Urban Deliveries

Projeto nº 2

- António Rodolfo de Almeida Seara Ferreira up202108834
- João Afonso Oliveira Teixeira Santos up202108805
- João Pedro Seabra Pedrosa Botelho de Sousa up202106996

Leitura dos dados fornecidos

A leitura dos dados fornecidos nos ficheiros csv realizou-se através de 3 funções pertencentes à classe Fileman que conseguem ler os dados através da função getline, armazenando estes em vetores.

Cada vez que o programa é iniciado, a função Fileman é chamada dando load a todos os dados sendo fácil e simples o seu acesso e utilização para diversos fins.

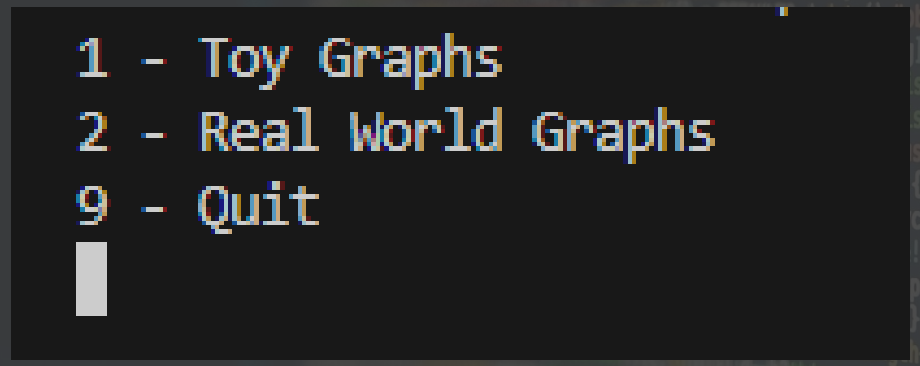
ENUMERAÇÃO DE CLASSES

- *Classe Graph (Criação, implementação do grafo e restantes funções necessárias)*
- *Classe Fileman (Carregamento de dados)*
- *Classe main (Interface)*

Classe Graph

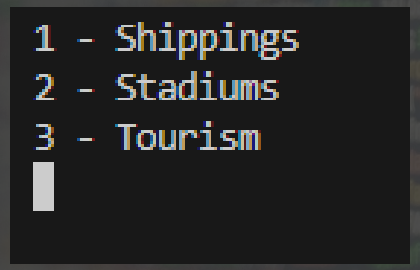
```
class Graph {
public:
    Graph();
    void addNode_RWG(int id, double longitude, double latitude);
    void addEdge_RWG(int origin, int destination, double distance);
    void addEdge(int origem, int destino, double distancia);
    double getMinCost() const;
    double getDistance(int source, int destination);
    vector<int> getBestPath() const;
    void tspBacktracking();
    void tspBacktrackingUtil(int currNode, int depth, double currCost);
    void tspBacktracking_RWG();
    void tspTriangularApproximation();
    void backtrack(vector<int>& path, vector<bool>& visited, int current, double cost);
    double calculateCost(const std::vector<int>& path);
    void resetVisited();
    double calculateDistance(int origin, int destination);
    void calculateGeographicDistances();
    double calculatePathCost(const vector<int>& path);
    double calculateHaversineDistance(double lat1, double lon1, double lat2, double lon2);
    double toRadians(double degrees);


private:
    unordered_map<int, Node_RWG> Nodes_RWG;
    unordered_map<int, Node> Nodes;
    double minCost;
    vector<int> currentPath;
    vector<int> bestPath;
    double currentCost;
    vector<Edge_RWG> edges;
    unordered_map<int, unordered_map<int, double>> distanceMatrix;
    int numNodes;
};
```

Menu principal



Menu principal com três opções que levam a sub-menus onde se vão realizar as várias operações descritas no enunciado do problema!

Toy Graphs




```
Minimum Cost: 86.7
Best Path: 0 1 11 10 12 13 3 2 4 6 9 7 8 5 0
Execution Time = 120ms
```

- Depois de escolher a opção “Toy Graphs” e “Shippings” no Menu Principal é-nos apresentada a solução do caminho optimal para o grafo em questão.

Exemplos de funcionalidades

- *Criação de uma interface simples "user-friendly"*

- *Leitura dos dados presentes nos ficheiros.csv*

4.1. Backtracking Algorithm [T2.1: 4 points]
In this approach, you are asked to develop a backtracking algorithmic approach to the TSP for a graph starting and ending the node tour on node labelled with the zero-identifier label. You are expected to use the small graphs to validate this approach and to observe that for the larger graphs this approach is not really feasible. To this extent, you are suggested to plot the execution time (or other performance metrics you find significant) to illustrate the feasibility or not of this approach for larger graphs.

4.2. Triangular Approximation Heuristic [T2.2: 4 points]
In this approach, you are asked to use the geographic node data, and implement the approximation algorithm for TSP that relies on the triangular inequality to ensure a 2-approximation algorithm for this problem again starting and ending the node tour on node with the zero-identifier label. You should use the results from this algorithm and compare them with the backtracking algorithm for the same small graphs.