Rail Network Management

Projeto nº 1

- 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 manager que conseguiam ler os dados através da função getline, armazenando estes em vetores.

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

Grafo utilizado

```
class Graph {
public:
    Graph();
    void addStation(const Station& station); // node
    void addNetwork(const Network& network); // edge
    vector<Station> getAdjacentStations(const string& stationName) const;
    int getNetworkCapacity(const string& station_A, const string& station_B) const;
    void setResidualCapacity(const string& station_A, const string& station_B, int flow);
    int getResidualCapacity(const string& station_A, const string& station_B) const;
    bool bfs(const string& source, const string& destination, unordered_map<string, string>& parent);
    int maxFlow(const string& source, const string& destination); // 2.1
    void findMostTrainsRequired(); // 2.2
    int maxNumOfTrainsArrivingAt(const string& station); // 2.4
    void topTransportationNeedsDistrict(int k); // 2.3
    void topTransportationNeedsMunicipality(int k); // 2.3
    static Graph createSubGraph(const Graph& graph, const string& line, int num); // 4.1
    int maxFlowMinCost(const string& source, const string& destination); // 3.1
private:
    unordered_map<string, Station> stations;
    unordered_map<string, vector<Network>> stationNetworks;
    unordered_map<string, unordered_map<string, int>> initialCapacities;
};
```

ENUMERAÇÃO DE CLASSES

- *Classe Graph (Criação e implementação do grafo)*
- *Classe Network (Dados sobre as networks)*
- *Classe Station (Dados sobre as estações)*

Interface / menu

```
Select what you pretend to do:

1: Service Metrics
2: Calculate the maximum amount of trains that can simultaneously travel between two specific stations with minimum cost for the company
3: Reliability and sensitivity to line failures
4: Exit

>>
```

Menu principal com os vários sub-menus onde se vão realizar as várias operações!

Interface / menu

Exemplos de execução:

```
Select what you pretend to do:

1: Find maximum number of trains between 2 stations
2: Find station pairs with highest max flow
3: Top-k municipalities and districts, regarding transportation needs
4: Find the maximum number of trains that can simultaneously arrive at a given station

>> 1

Enter the source station name: Faro

Enter the destination station name: Pombal

Maximum number of trains that can simultaneously travel: 4
```

Interface / menu

Exemplos de execução:

```
Select what you pretend to do:

1: Find maximum number of trains between 2 stations
2: Find station pairs with highest max flow
3: Top-k municipalities and districts, regarding transportation needs
4: Find the maximum number of trains that can simultaneously arrive at a given station

>> 4

Enter the source station name: Funcheira

The maximum number of trains that can simultaneously arrive at the station Funcheira is 4!
```

Exemplos de funcionalidades

- *Viagem mais curta entre duas estações* $O(V + E)$
- *Número máximo de comboios entre duas estações* $O(|V| * E^2)$
- *Top k municípios e distritos, de acordo com as necessidades de transporte* $O(V^2 * E)$
- *Encontrar o número máximo de comboios que chega a uma estação* $O(V^2 * E^2)$
- *Encontrar os pares de estações com o flow máximo* $O(V^4 * E^2)$

DESTAQUE DE FUNCIONALIDADES

Edmond's karp

```cpp
int Graph::maxFlow(const string& source, const string& destination) { // edmon's-karp
    if (stations.count(source) == 0 || stations.count(destination) == 0) {
        cout << "\nError: Invalid source or destination station" << endl;
        return -1;
    }

    unordered_map<string, string> parent;
    int max_flow = 0;

    while (bfs(source, destination, parent)) {
        int aug_flow = numeric_limits<int>::max(); // infinite
        string currentNode = destination;

        // find the bottleneck capacity along the augmenting path
        while (currentNode != source) {
            string prevNode = parent[currentNode];
            aug_flow = min(aug_flow, getResidualCapacity(prevNode, currentNode));
            currentNode = prevNode;
        }

        // update the flow along the augmenting path
        currentNode = destination;
        while (currentNode != source) {
            string prevNode = parent[currentNode];
            setResidualCapacity(prevNode, currentNode, aug_flow);
            setResidualCapacity(currentNode, prevNode, -aug_flow);
            currentNode = prevNode;
        }

        // add the augmenting flow to the total flow
        max_flow += aug_flow;
    }

    return max_flow;
}
```

Pesquisa em largura

```cpp
bool Graph::bfs(const string &source, const string &destination, unordered_map<string, string> &parent) {
    unordered_map<string, bool> visited;
    unordered_map<string, int> distance;
    for(const auto& station: stations){
        visited[station.first] = false;
        distance[station.first] = numeric_limits<int>::max();
    }

    queue<string> q;
    q.push(source);
    visited[source] = true;
    distance[source] = 0;
    parent[source] = ""; // set an invalid parent

    while (!q.empty()) {
        string current = q.front();
        q.pop();
        vector<Station> adjacentStations = getAdjacentStations(current);

        for (const Station& adjacentStation : adjacentStations) {
            if (!visited[adjacentStation.getName()] && getResidualCapacity(current, adjacentStation.getName()
                parent[adjacentStation.getName()] = current;
                visited[adjacentStation.getName()] = true;
                distance[adjacentStation.getName()] = distance[current] + 1;
                q.push(adjacentStation.getName());

                if (adjacentStation.getName() == destination) {
                    return true;
                }
            }
        }
    }

    return false;
}
```