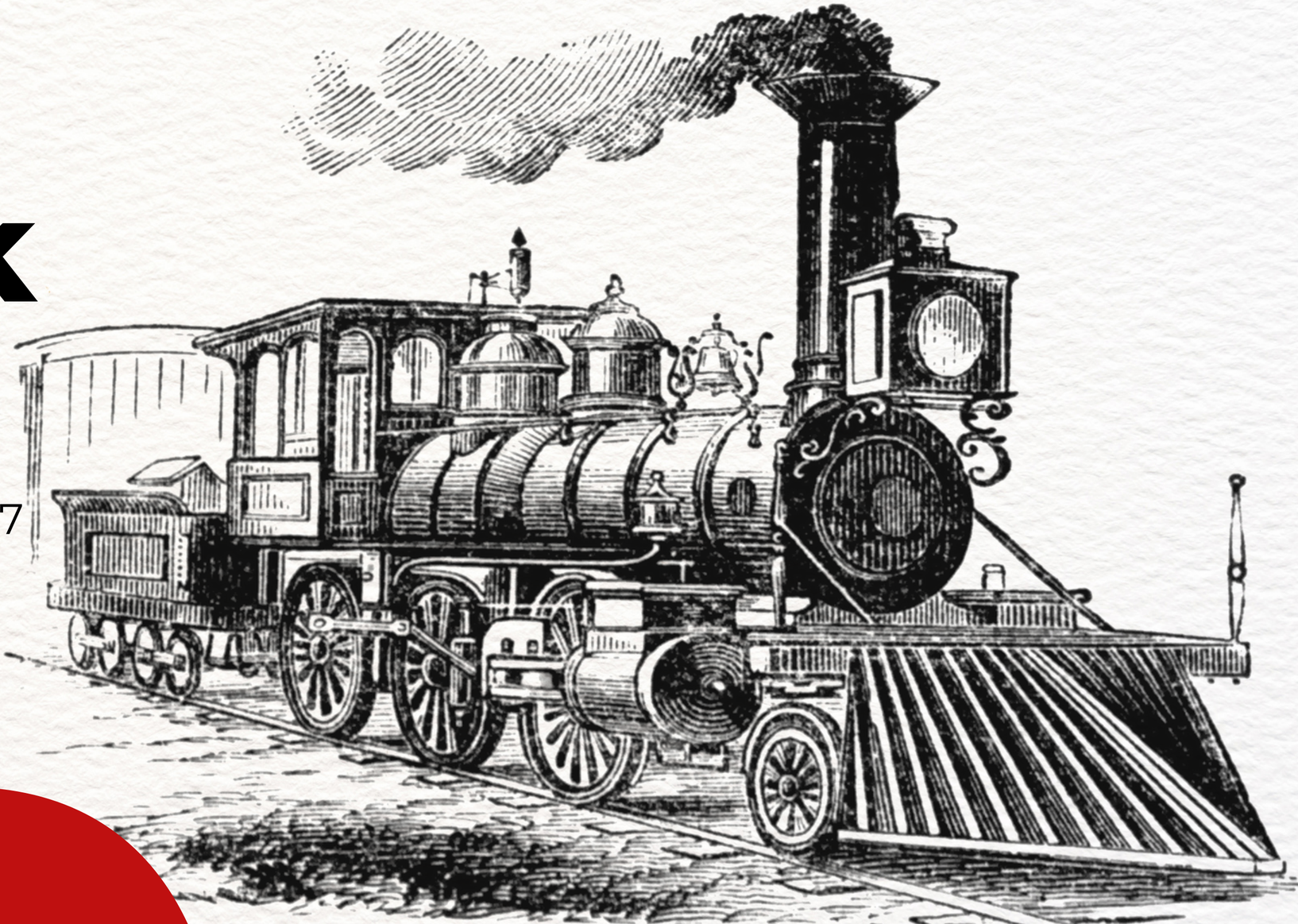
DA Project I

Railway Network



G03_1:

Felipe Ferreira - up202102359

Lucas Nakajima - up202001337

Ian Beltrão- up202102360

U.PORTO

FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Problem Description

Creating a graph based in a dataset of stations and connections between them to perform specific operations on it.

Classes organization

```
using namespace std;

class Station {
private:
    /// @brief The name of the station.
    string name;
    /// @brief The district where the station is located.
    string district;
    /// @brief The municipality where the station is located.
    string municipality;
    /// @brief The township where the station is located.
    string township;
    /// @brief The line to which the station belongs.
    string line;
```

Stations.h stores the information about the stations like name, municipality, district, township and line.

Classes organization

```
/**
 * @brief The Connection class represents a connection between two stat
 */
class Connection {
private:
    /// @brief The source station.
    Station source;
    /// @brief The destination station.
    Station destination;
    /// @brief The capacity of the connection.
    int capacity;
    /// @brief The type of service provided by the connection.
    string service;
    /// @brief The residual capacity of the connection.
    int residual; /// @brief The residual capacity of the connection.
```

Connections.h stores the information about the connections between stations like source, destination, capacity and service.

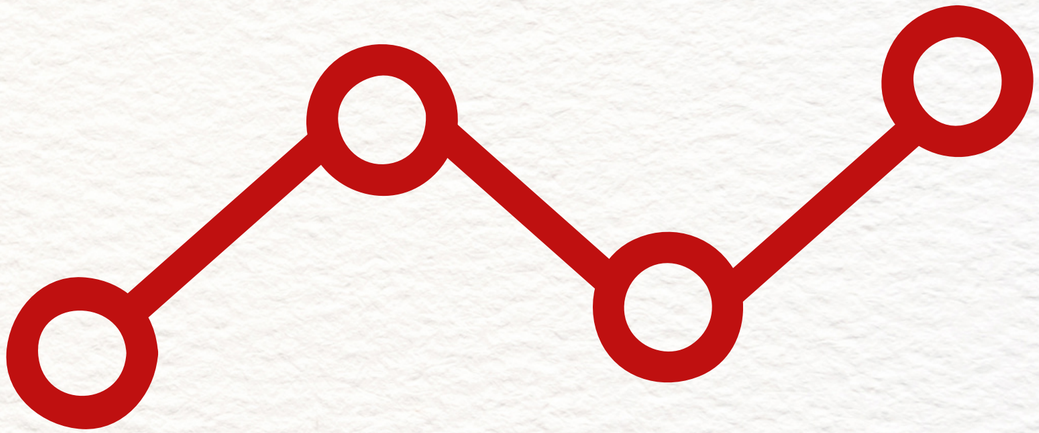
Dataset reading

Stations.csv and network.csv are both stored in the graph trough methods called loadStations( ) and loadConnections( ) respectively.

```
bool Graph::loadStations() {
    ifstream inputFile( s: "stations.csv");

    if (!inputFile.is_open()) {
        return false;
    }

    // Skip header line
    string line;
    getline( &: inputFile,  &: line);

    while (getline( &: inputFile,  &: line)) {
        istringstream iss( str: line);

        string name, district, municipality, township, lines;
        getline( &: iss,  &: name,  delim: ',');
        getline( &: iss,  &: district,  delim: ',');
        getline( &: iss,  &: municipality,  delim: ',');
        getline( &: iss,  &: township,  delim: ',');
        getline( &: iss,  &: lines,  delim: ',');

//        if(name == "Porto Campanhã"){
//            cout << name << endl;
//        }
        if(stations.find( x: name) != stations.end()){
//            cout << name << endl;
            continue;
        }
        Station station(name, district, municipality, township,  line: lines);
        stations.emplace( &: name,  &: station);
    }

    return true;
}
```

Graph

The graph implemented was a biderectional graph where the **nodes** are the **stations** and each **edge** has a capacity and a service associated to it.



Funcionalities

- Stations by municipalities and districts.
- Pairs of stations that require the most trains
- Operations between two specific stations

```
vector<string> Menu:: stationsFetch(){
    string source, target;
    while(true) {
        cout << "Type the source:";
        getline( &: cin, &: source);
        if (!validStation( a: g,  station: source)) {
            cout << "Invalid source! Make sure you typed correctly and try again!\n";
            sleep(1);
            cout << string( n: 2, c: '\n');
            continue;
        }
        cout << "Type the target:";
        getline( &: cin, &: target);
        if (!validStation( a: g,  station: target)) {
            cout << "Invalid target! Make sure you typed correctly and try again!\n";
            sleep(1);
            cout << string( n: 2, c: '\n');
            continue;
        }
        if (source==target){
            cout << "You typed the same station twice! Try again!\n";
            sleep(1);
            cout << string( n: 2, c: '\n');
            continue;
        }
        return {source, target};
    }
}
```

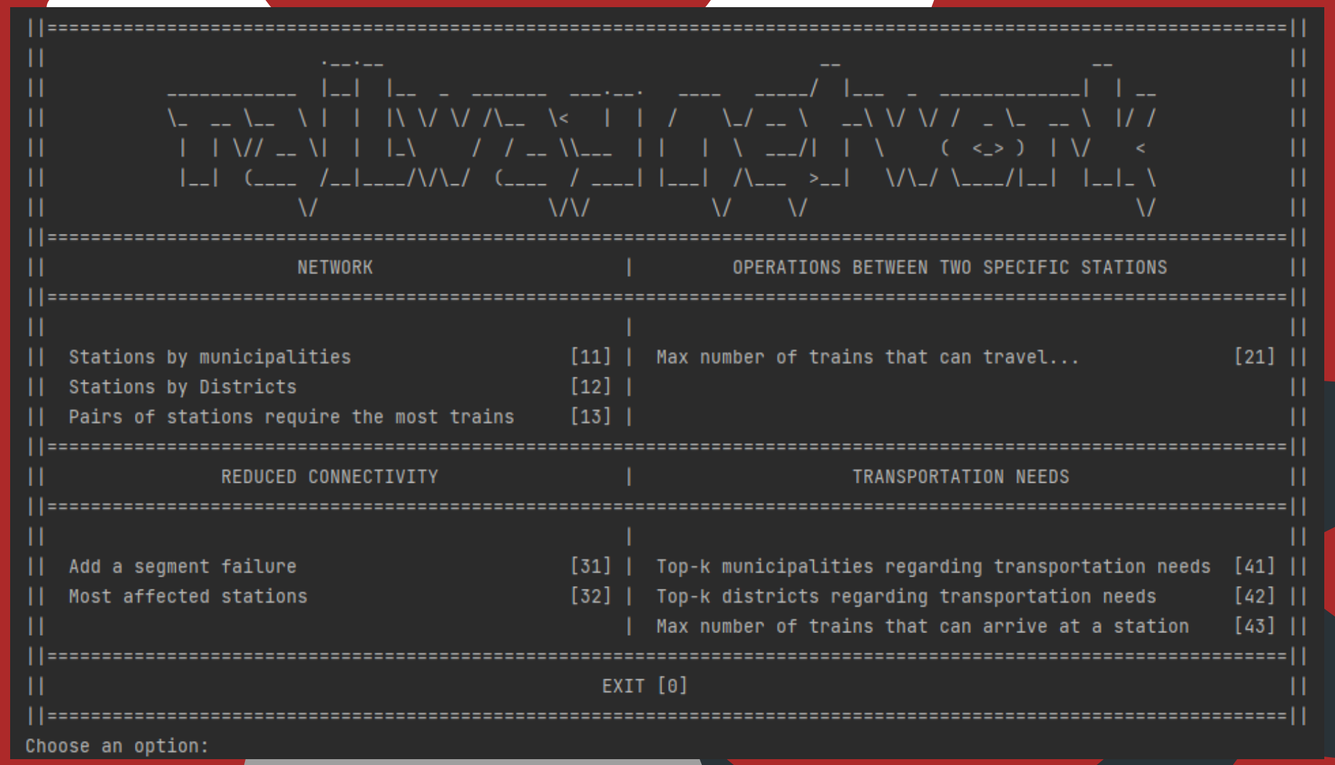
Funcionalities

- Reduced connectivity operations:
  - Add segment failure
  - Most affected stations
- Transportation needs:
  - top-k municipalities and districts
  - max number of trains that can arrive at a station

```
void Menu:: budgetDdistricts(){
    auto v : vector<string>  = g.topkbudgetDistrict();
    int k;
    while(true) {
        cout << "Type the k value:";
        cin >> k;
        if (cin.fail() || cin.peek() != '\n') {
            cin.clear();
            cin.ignore( n: INT_MAX,  delim: '\n');
            cout << "Invalid input! Try again!" << endl;
            sleep(1);
            continue;
        }
        else{
            cout << "Those are the top-" << k << " districts that need more budget:\n";
            for(int i=0; i<k and i<v.size(); i++){
                cout << v[i] << "\n";
            }
            sleep(1);
            break;
        }
    }
    cin.ignore( n: INT_MAX,  delim: '\n');
    pause();
}
```

User Interface

- A menu with the main functionalities
- Some submenus for source and target stations insertion.



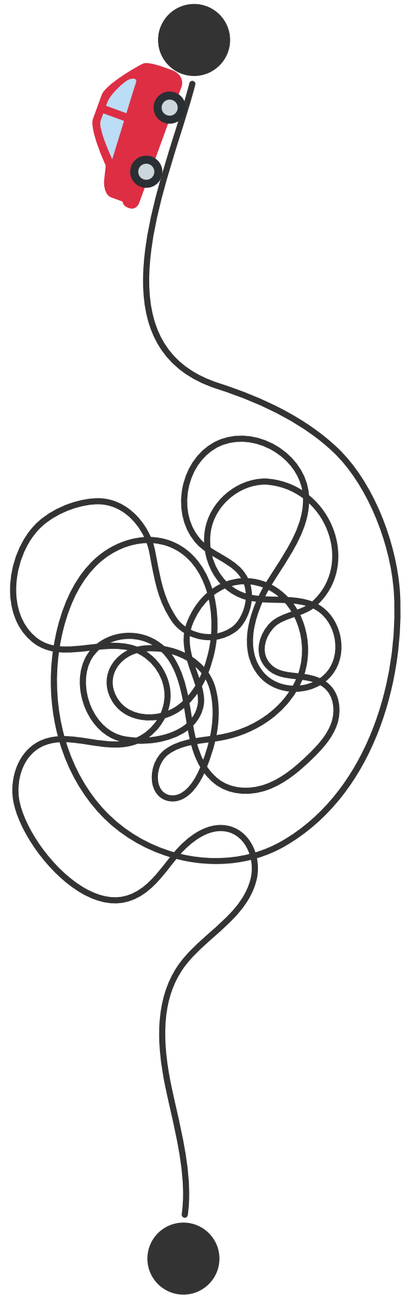
Algorithm Highlight

- Edmond Karp Algorithm

```
int Graph::calculateMaxFlow(string source, string sink) {
    updateResidualConnections();
    int maxFlow = 0;
    parent.clear();
    unordered_map<string, bool> visited;
    while(bfs(source, destination: sink)) {
        int pathFlow = INF;

        for (string v = sink; v != source; v = parent[v]) {
            string u = parent[v];
            for (const auto &connection : const Connection & : targets[u]) {
                if (connection.getDestination().getName() == v) {
                    pathFlow = min(pathFlow, connection.getCapacity());
                }
            }
        }
        for (string v = sink; v != source; v = parent[v]) {
            string u = parent[v];
            for (auto &connection : Connection & : targets[u]) {
                if (connection.getDestination().getName() == v) {
                    connection.setResidual( newResidual: connection.getResidual() - pathFlow);
                }
            }
            for (auto &reverseConnection : Connection & : targets[v]) {
                if (reverseConnection.getDestination().getName() == u) {
                    reverseConnection.setResidual( newResidual: reverseConnection.getResidual() + pathFlow);
                }
            }
        }

        maxFlow += pathFlow;
    }
    updateResidualConnections();
    return maxFlow;

}
```

Main difficulty

- Dealing with different kinds of flows



Thank You for your Atention!!