Problema

- Rede de estações interligadas por linhas férreas bidirecionais
- Leitura dos ficheiros de forma prática e acessível
- Criação de algoritmos eficazes

Solução

→ Criação de classes

```
class Station {

private:
    std::string name;
    std::string district;
    std::string municipality;
    std::string township;
    std::string line;
```

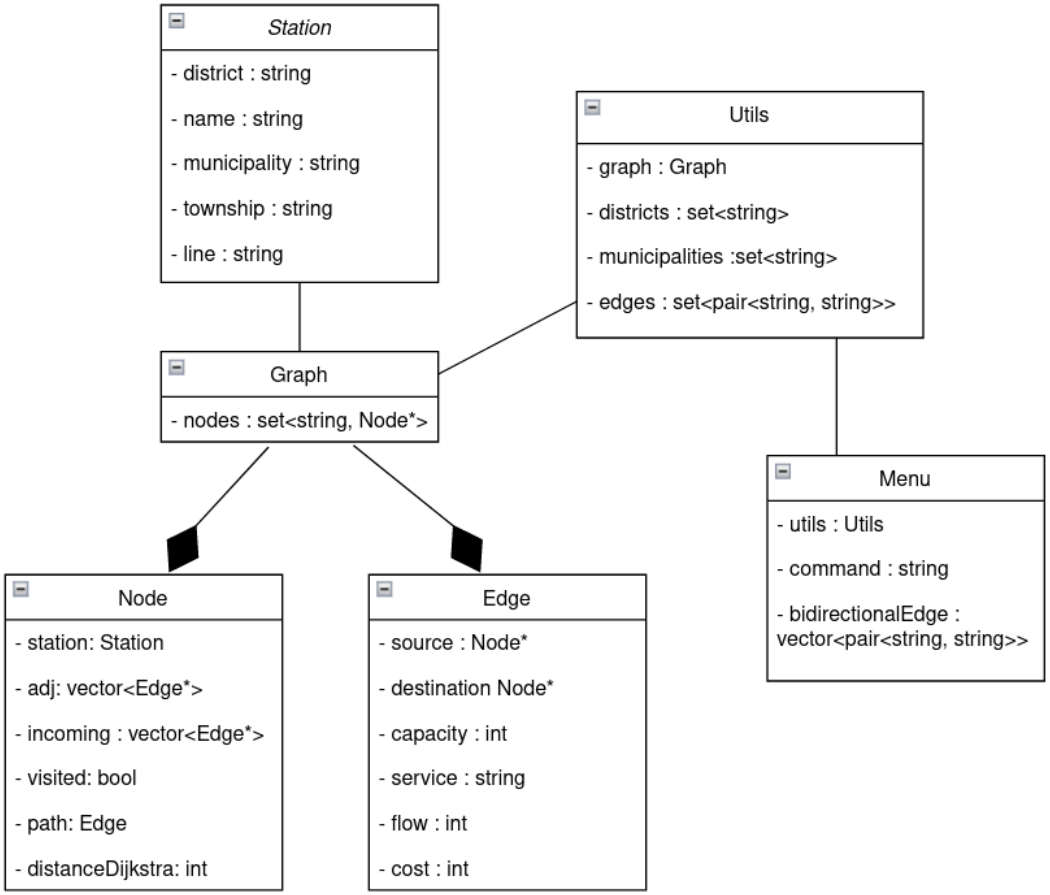
```
class Graph {

private:

    struct Node;

    struct Edge {
        Node* source;
        Node* destination;
        int capacity;
        std::string service;
        int flow;
        int cost = 2 * (service == "STANDARD") + 4 * (service == "ALFA PENDULAR");
    };

    struct Node {
        Station station;
        std::vector<Edge*> adj;
        std::vector<Edge*> incoming;
        bool visited = false;
        Edge* path = nullptr;
        int distanceDijkstra;
    };

    std::unordered_map<std::string, Node*> nodes;
```

```
class Utils {

private:
    Graph graph;
    std::set<std::string> districts;
    std::set<std::string> municipalities;
    std::set<std::pair<std::string, std::string>> edges;
```

```
class Menu {

private:
    Utils utils;
    std::string command;
    std::vector<std::pair<std::string, std::string>> bidirectionalEdges;
```

```
/**...*/
void mainMenu();

/**...*/
void stationsMenu();

/**...*/
void maxFlowMenu();

/**...*/
void reducedConMenu();

/**...*/
void changeEdges();
```

UML


Station
- district : string
- name : string
- municipality : string
- township : string
- line : string
Utils
- graph : Graph
- districts : set<string>
- municipalities :set<string>
- edges : set<pair<string, string>>
Graph
- nodes : set<string, Node*>
Menu
- utils : Utils
- command : string
- bidirectionalEdge : vector<pair<string, string>>
Node
- station: Station
- adj: vector<Edge*>
- incoming : vector<Edge*>
- visited: bool
- path: Edge
- distanceDijkstra: int
Edge
- source : Node*
- destination Node*
- capacity : int
- service : string
- flow : int
- cost : int

Funcionalidades

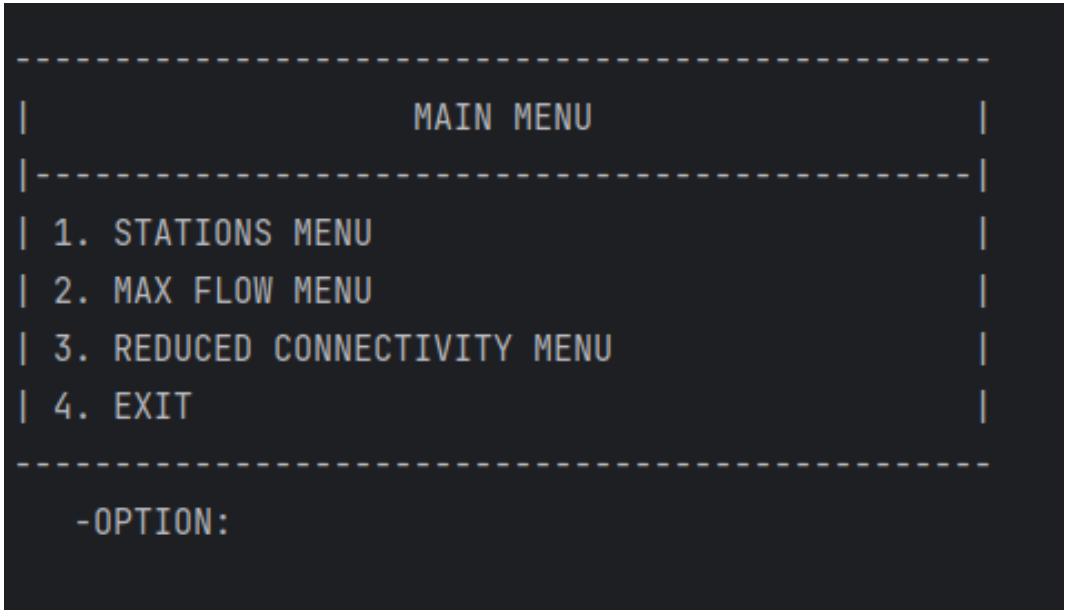
```cpp
int Graph::edmondsKarp(Node* source, Node* sink) {

    int maxFlow = 0;
    setAllFlows0();

    // Loop to find augmentation paths
    while (bfsFindAugmentingPath(source, sink)) {
        int flow = findMinResidualAlongPath(source, sink);
        augmentFlowAlongPath(source, sink, flow);
        maxFlow += flow;
    }

    return maxFlow;
}
```

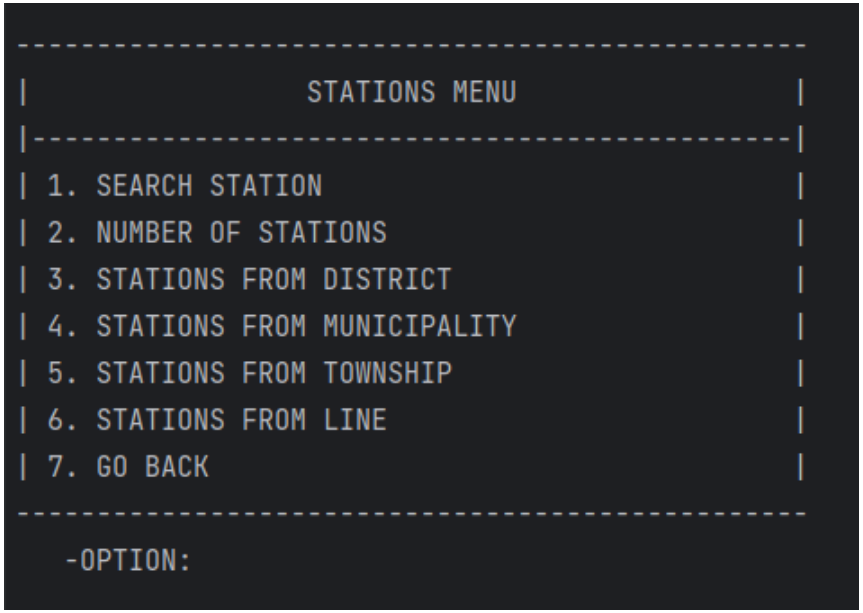
```cpp
bool Graph::bfsFindAugmentingPath(Node* source, Node* sink) {
    setAllNodesUnvisited();
    source->visited = true;

    queue<Node*> queue;
    queue.push(source);

    while (!queue.empty() && !sink->visited) {
        Node* node = queue.front();
        queue.pop();
        for (auto edge: node->adj) {
            testAndVisit(queue, edge, edge->destination, edge->capacity - edge->flow);
        }
        for (auto edge: node->incoming) {
            testAndVisit(queue, edge, edge->source, edge->flow);
        }
    }
    return sink->visited;
}
```

Funcionalidades (cont.)

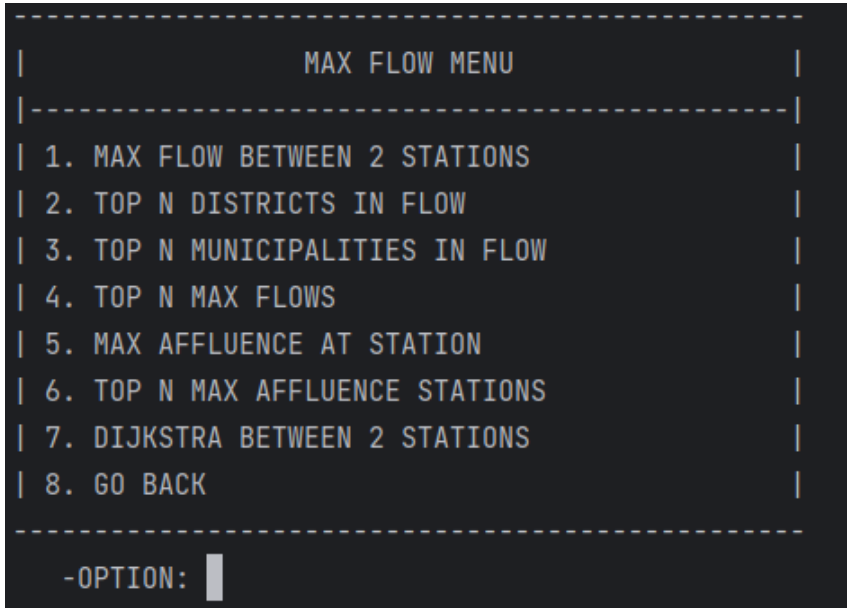
```
int Graph::dijkstra(Graph::Node* source, Graph::Node* target) {

    for (auto& node : nodes) {
        node.second->path = nullptr;
        node.second->distanceDijkstra = INT32_MAX;
    }

    priority_queue<Node*, vector<Node*>, function<bool(Node*, Node*)>> queue(compareDijkstra);
    queue.push(source);
    source->distanceDijkstra = 0;

    while (!queue.empty()) {

        Node* node = queue.top();
        queue.pop();

        for (auto edge: node->adj) {

            int newDistance = node->distanceDijkstra + edge->cost;
            if (newDistance < edge->destination->distanceDijkstra) {
                edge->destination->distanceDijkstra = newDistance;
                edge->destination->path = edge;
                queue.push(edge->destination);
            }
        }
    }
    return target->distanceDijkstra;
}
```

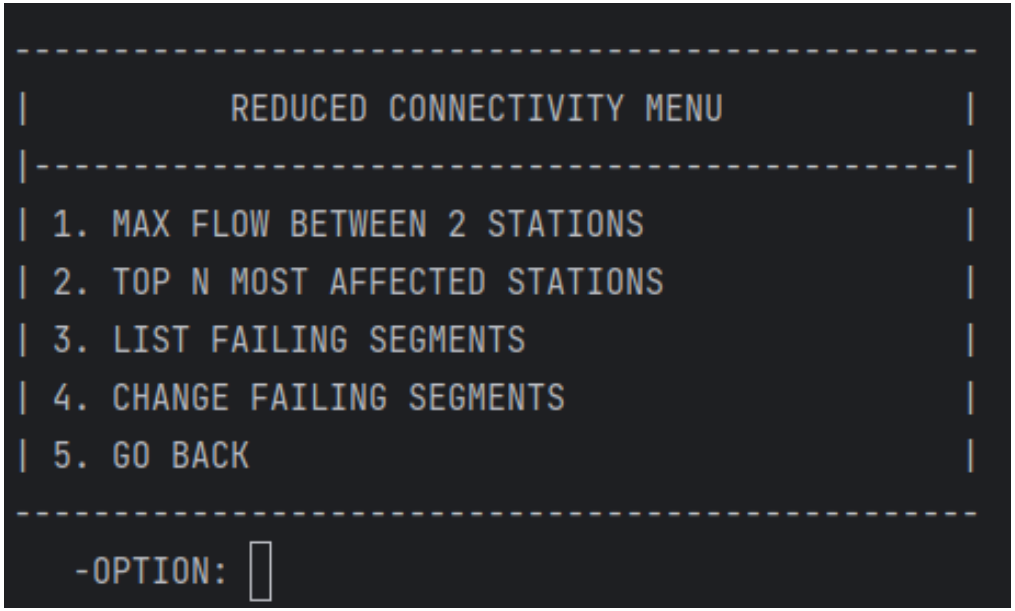
Menu


MAIN MENU
1. STATIONS MENU
2. MAX FLOW MENU
3. REDUCED CONNECTIVITY MENU
4. EXIT
-OPTION:


STATIONS MENU
1. SEARCH STATION
2. NUMBER OF STATIONS
3. STATIONS FROM DISTRICT
4. STATIONS FROM MUNICIPALITY
5. STATIONS FROM TOWNSHIP
6. STATIONS FROM LINE
7. GO BACK
-OPTION:

Menu (cont)


MAX FLOW MENU
1. MAX FLOW BETWEEN 2 STATIONS
2. TOP N DISTRICTS IN FLOW
3. TOP N MUNICIPALITIES IN FLOW
4. TOP N MAX FLOWS
5. MAX AFFLUENCE AT STATION
6. TOP N MAX AFFLUENCE STATIONS
7. DIJKSTRA BETWEEN 2 STATIONS
8. GO BACK
-OPTION:


REDUCED CONNECTIVITY MENU
1. MAX FLOW BETWEEN 2 STATIONS
2. TOP N MOST AFFECTED STATIONS
3. LIST FAILING SEGMENTS
4. CHANGE FAILING SEGMENTS
5. GO BACK
-OPTION: