Programming Project I

An Analysis Tool for Railway Network Management

W E L C O M E

DA - April 2023 - Group G09_1

Our Group

Rúben Correia Pereira

up202006195

Tomás Ballester Carvalho Sousa Pereira

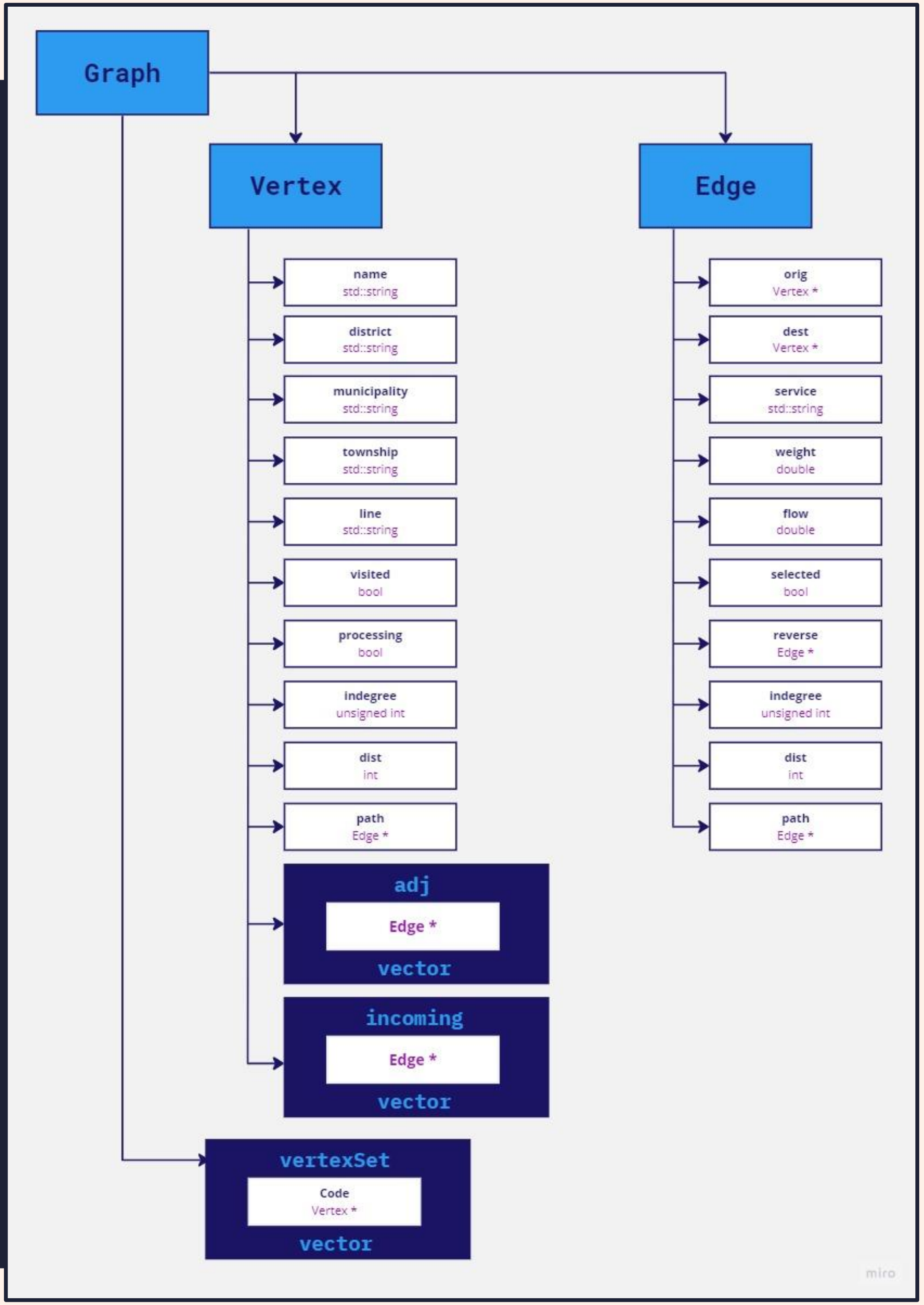
up202108812

Simão Queirós Rodrigues

up202005700

Class Diagram

We have 3 classes implemented into our project: Graph, Vertex and Edge.



The Menu

Parsing the data

Basic Service Metrics

Operation Cost Optimization

Reliability and Sensitivity to Line Failures

We've divided our project in these **four** parts, as well as our menu.

```
-----------------------------------------------------------------
      [1]     Parse
      [2]     Basic Service Metrics
      [3]     Max No. of Trains Between 2 Stations Using the
              Cheapest Path (Operation Cost Optimization)
      [4]     Reliability and Sensitivity to Line Failures

      [0]     Exit
-----------------------------------------------------------------
```

Parsing the data

The 1st menu option

You can populate/parse this program and its data structures with any file you want. Just input the path to the stations and network data files and it's ready to use. Now we have a Graph with Vertices and Edges that represent the stations and their connections.

```
Enter an option: 1

Enter the path of the desired stations data file: ../data/stations.csv

Enter the path of the desired network data file: ../data/network.csv

Parsed!
```

```
bool Graph::parser(string stations_p, string network_p){
    Graph g = Graph();
    fstream stations_f,network_f;
    stations_f.open( s: stations_p, mode: ios::in);

    if(!stations_f.is_open()){
        return false;
    }
    string name,district,municipality,township,line;
    getline( &: stations_f, &: name);
    while(getline( &: stations_f, &: name, delim: ',')){
        getline( &: stations_f, &: district, delim: ',');
        getline( &: stations_f, &: municipality, delim: ',');
        getline( &: stations_f, &: township, delim: ',');
        getline( &: stations_f, &: line);

        addVertex(name,district,municipality,township,line);
    }
    stations_f.close();
    network_f.open( s: network_p, mode: ios::in);

    if(!network_f.is_open()){
        return false;
    }
    string sta_A,sta_B,cap,ser;
    getline( &: network_f, &: sta_A);
    while(getline( &: network_f, &: sta_A, delim: ',')){
        getline( &: network_f, &: sta_B, delim: ',');
        getline( &: network_f, &: cap, delim: ',');
        getline( &: network_f, &: ser);

        addBidirectionalEdge(sta_A, sta_B, cap: stod( str: cap), service: ser);
    }
    network_f.close();
    return true;
}
```

Basic Service Metrics

The 2nd menu option | 2.1

Max. Number of Trains that can simultaneously Travel between Two Specific Stations

To find out the maximum number of trains that simultaneously travel between Porto Campanhã and Lisboa Oriente (represented by two **Vertices**),we find them in our Graph by their names and then perform the **Edmonds-Karp** algorithm. That way, we can find our desired number of trains.

```
Enter an option:1

Enter the name of the source station:Porto Campanhã

Enter the name of the target station:Lisboa Oriente

The max number of trains that can simultaneously travel between
Porto Campanhã and Lisboa Oriente are 10 trains.
Press any key to continue . . .
```

```
--------------------------------------------------------------------------------
    [1]     Max Number of Trains that can simultaneously travel
            between two specific stations
    [2]     The Pair(s) of Stations that Require the Most Amount of
            Trains (taking full advantage of the network's capacity)
    [3]     Current Top-k Municipalities/Districts that need more
            budget to sustain its services
    [4]     Max No. of Trains that arrive simultaneously at a station

    [0]     Back
--------------------------------------------------------------------------------
```

This is how the menu looks for the 2nd menu option

```
double Graph::edmondsKarp(string source, string target) {
    Vertex* s = findVertex( name: source);
    Vertex* t = findVertex( name: target);
    if (s == nullptr || t == nullptr || s == t)
        throw std::logic_error("Invalid source and/or target vertex");

    // Reset the flows
    for (auto v :Vertex * : vertexSet) {
        for (auto e :Edge * : v->getAdj()) {
            e->setFlow(0);
        }
    }
    double maxFlow = 0;
    // Loop to find augmentation paths
    while( findAugmentingPath(s, t) ) {
        double f = findMinResidualAlongPath(s, t);
        augmentFlowAlongPath(s, t, f);
        maxFlow += f;
    }
    return maxFlow;
}
```

Code implementation of the Edmonds-Karp algorithm.

Basic Service Metrics

The 2nd menu option | 2.2

The Pair(s) of Stations that Require the Most Amount of Trains (taking full advantage of the network's capacity)

To get the pair(s) of stations that require the most amount of trains while taking full advantage of the network’s capacity, we calculate the maximum max flow (calculated via the Edmonds-Karp algorithm) and keep track of those pairs of stations.

```
Enter an option:2

Loading...

Finishing...

Lisboa Oriente - Entroncamento
Lisboa Oriente - Santarém
Entroncamento - Santarém

This/these pair(s) require 22 trains when
taking full advantage of the network's capacity.

Press any key to continue . . .
```

```cpp
vector<pair<pair<string,string>,double>> Graph::getPairs_MostAmountOfTrains(){
    vector<pair<string,string>> pairs;
    vector<int> capacities;
    vector<pair<pair<string,string>,double>> pairs_f;
    double trains_max = vertexSet[0]->getAdj()[0]->getWeight();
    int num_stations = getNumVertex();
    cout << "\nLoading..." << endl;
    for(int i = 0; i < num_stations;i++){
        for(int j = i+1;j < num_stations;j++){
            double cap = edmondsKarp( vertexSet[i]->getName(), vertexSet[j]->getName());
            if(cap >= trains_max) {
                trains_max = cap;
                pairs.push_back(make_pair( vertexSet[i]->getName(), vertexSet[j]->getName()));
                capacities.push_back(cap);
            }
        }
    }
    cout << "\nFinishing..." << endl;
    for(int i = 0; i < pairs.size(); i++){
        if(capacities[i] == trains_max){
            pairs_f.push_back(make_pair( pairs[i], trains_max));
        }
    }
    return pairs_f;
}
```

Code implementation of the getPairs_MostAmountOfTrains() function.

Basic Service Metrics

The 2nd menu option | 2.3

Current Top-k Municipalities/Districts that need more budget to sustain its services

To get the current top 5 of municipalities that require more budget to sustain its services, for example, we can simply ask the program, as the image below shows. The **budgetMunicipalities()** finds out all the current municipalities that exists in the Graph and then calculates the sum of every max. flow between all stations inside that municipality.

```
Enter an option:3

Do you want to search Municipalities(M/m) or Districts(D/d)?m

How big will the top be? 5

TOP 5 MUNICIPALITIES

1. || LISBOA || 678
2. || POMBAL || 156
3. || AVEIRO || 134
4. || COIMBRA || 126
5. || SINTRA || 118
Press any key to continue . . .
```

(We're using the municipalities as an example, but there's also the district equivalent process)

```cpp
vector<pair<string,double>> Graph::budgetMunicipalities(int k){
    unordered_map<string,vector<Vertex*>> municipalities;
    vector<pair<string,double>> maxflowMunicipalities;
    for(auto v : vertexSet){
        municipalities[v->getMunicipality()].push_back(v);
    }
    for(auto m :municipalities){
        double d = 0;
        auto v = m.second;
        if(m.second.size()==1){
            maxflowMunicipalities.push_back(make_pair( m.first, d));
            continue;
        }
        for (int i=0;i!=v.size();i++){
            for(int j=i+1;j!=v.size();j++) {
                d += edmondsKarp( v[i]->getName(), v[j]->getName());
            }
        }
        maxflowMunicipalities.push_back(make_pair( m.first, d));
    }
    sort( maxflowMunicipalities.begin(), maxflowMunicipalities.end(), [](auto& lhs , auto& rhs ) {
        return lhs.second > rhs.second;
    });
    if (k < maxflowMunicipalities.size()) {
        maxflowMunicipalities.erase( maxflowMunicipalities.begin() + k, maxflowMunicipalities.end());
    }
    else{
        cout << "ERROR: Please input a k value smaller than " << maxflowMunicipalities.size() << "." << endl;
    }
    return maxflowMunicipalities;
}
```

Code implementation of the budgetMunicipalities() function.

Basic Service Metrics

The 2nd menu option | 2.4

Max. Number of Trains that arrive simultaneously at a station

To calculate the maximum number of trains that arrive simultaneously at a station, we create a super Vertex that connects to every source station (that isn't the target station), and then calculate the max. Flow between that super Vertex and the desired station, using the **Edmonds-Karp** algorithm.

```
Enter an option:4

Enter the name of the desired station:Porto Campanhã

The max number of trains that arrive at Porto Campanhã is 20 trains.
Press any key to continue . . .
```

```
double Graph::maxTrainsAtStation(string station){
    Vertex super_v = Vertex( name: "super_v");
    vertexSet.push_back(&super_v);
    double flow = 0;
    for(auto e :Vertex * : vertexSet){
        if(e->getAdj().size()==1 and e->getName()!=station){
            super_v.addEdge( dest: e, w: INT_MAX, service: "");
        }
    }
    flow = edmondsKarp( source: super_v.getName(), target: station);
    removeVertex( s: "super_v");
    return flow;
}
```

Code implementation of the maxTrainsAtStation() function.

Operation Cost Optimization

```
Enter an option: 3

Name of first station?Porto Campanhã

Name of second station?Lisboa Oriente

The minimum cost is: 80
Max num of trains with min cost: 4

Press any key to continue . . .
```

To find the maximum number of Trains between two stations using the cheapest path, we've implemented the **max_trains_min_cost** function that call the **maxFlow_minCost**.

```cpp
void Graph::max_trains_min_cost(string station1, string station2) {

    int max_trains = maxFlow_minCost(std::move(station1), std::move(station2));

    if(max_trains == 0){
        cout << "Found no connection between this stations\n";
    }
    else{
        cout << "Max num of trains with min cost: " << max_trains << endl;
        cout << endl;
    }
}
```

```cpp
int Graph::maxFlow_minCost(string source, string target) {
    Vertex *s = findVertex(std::move(source));
    Vertex *t = findVertex(std::move(target));
    if(s == nullptr || t == nullptr){
        cout << "\nInvalid station inserted" << endl;
        return 0;
    }
    for(auto v : vertexSet){
        auto e = v->getPath();
        if(e==NULL){
            continue;
        }
        e->setSelected(false);
    }

    for (int i = 0; i < getNumVertex(); i++) {
        for (int j = 0; j < vertexSet[i]->getAdj().size(); j++) {
            vertexSet[i]->getAdj()[j]->setFlow(0);
        }
    }
    std::vector<double> aux;
    int i = 0;
    vector<pair<string,int>> services ={{"ALFA PENDULAR", 0},{"STANDARD", 0}};
    while (bfs_for_max_flow(s, t)) {
        double path_flow = INF;
        for (auto v = t; v != s;) {
            auto e = v->getPath();
            if(i==0){
            if(e->getService()=="ALFA PENDULAR" && !e->isSelected()){
                services[0].second++;
                e->setSelected(true);
            }
            else if (e->getService()=="STANDARD" && !e->isSelected()){
                services[1].second++;
                e->setSelected(true);
            }}
            if (e->getDest() == v) {
                path_flow = std::min(path_flow, e->getWeight() - e->getFlow());
                v = e->getOrig();
            } else {
                path_flow = std::min(path_flow, e->getFlow());
                v = e->getDest();
            }
        }
        i++;
        for (auto v = t; v != s;) {
            auto e = v->getPath();
            double flow = e->getFlow();
            if (e->getDest() == v) {
                e->setFlow(flow + path_flow);
                v = e->getOrig();
```

Reliability and Sensitivity to Line Failures

Max. Number of Trains that can simultaneously travel between two specific stations

We ask the user to remove (a) station(s) (and its associated lines) or just (a) specific line(s). Then we use the Edmonds-Karp algorithm to calculate the number of trains between the desired stations.

```
Graph f = Graph();
f.parser( stations, network);
string station1, station2;
cout << "\nPlease input the name of the origin station: " << endl;
cin.ignore();
getline( cin, station1);
cout << "\nPlease input the name of the target station: " << endl;
getline( cin, station2);
bool options=true;
while(options){
    string cutstation,answer1,answer2,sta_A,sta_B;
    cout << "\nDo you want to remove a station or a line? Station(S/s) or Line(L/l): " << endl;
    getline( cin, answer1);
    if(answer1=="S" || answer1=="s") {
        cout << "\nPlease input the name of the station: " << endl;
        getline( cin, cutstation);
        f.removeVertex( cutstation);
    }
    else if(answer1=="L" || answer1=="l"){
        cout << "\nPlease input the name of the source station: " << endl;
        getline( cin, sta_A);
        cout << "\nPlease input the name of the target station: " << endl;
        getline( cin, sta_B);
        f.findVertex( sta_A)->removeEdge( sta_B);
    }
    else{
        cout << "\nInsert a valid character... " << endl;
    }
    cout << "\nDo you want to keep adding more failures to the network? Yes(Y/y) or No(N/n): " << endl;
    getline( cin, answer2);
    if(answer2=="Y" || answer2=="y"){
        continue;
    }
    else if (answer2=="N" || answer2=="n"){
        options=false;
    }
    else{
        cout << "\nInsert a valid character... " << endl;
    }
}
cout << flush;
auto trains = f.edmondsKarp( station1, station2);
if(trains==0){
    cout << "There's no connection between "<< station1 << " and " << station2 << endl;
}
else {
    cout << "The max number of trains that can simultaneously travel between" << endl;
    cout << station1 << " and " << station2 << " are " << trains << " trains." << endl;
}
system( "pause");
}break;
```

Top-k stations most affected by (a) failure(s)

Similar to the top-k budget of municipalities/districts, a list is returned of the most affected stations after a stations failure occurs. This is determined by the difference of the maximum trains that can arrive simultaneously at a station before and after the failure.

```
Do you want to remove a station or a line? Station(S/s) or Line(L/l):
s

Please input the name of the station:
Porto Campanhã

Do you want to keep adding more failures to the network? Yes (Press '1') or No (Press
n

How big will the top be? 3

TOP 3 MOST AFFECTED STATIONS

Loading...

Finishing...

1. || Trofa || 10
2. || Vila Nova de Gaia-Devesas || 8
3. || Espinho || 4
Press any key to continue . . .
```

Thank you!