Trabalho 2 de AED:

Navegação nos transportes públicos do Porto

Grupo 67

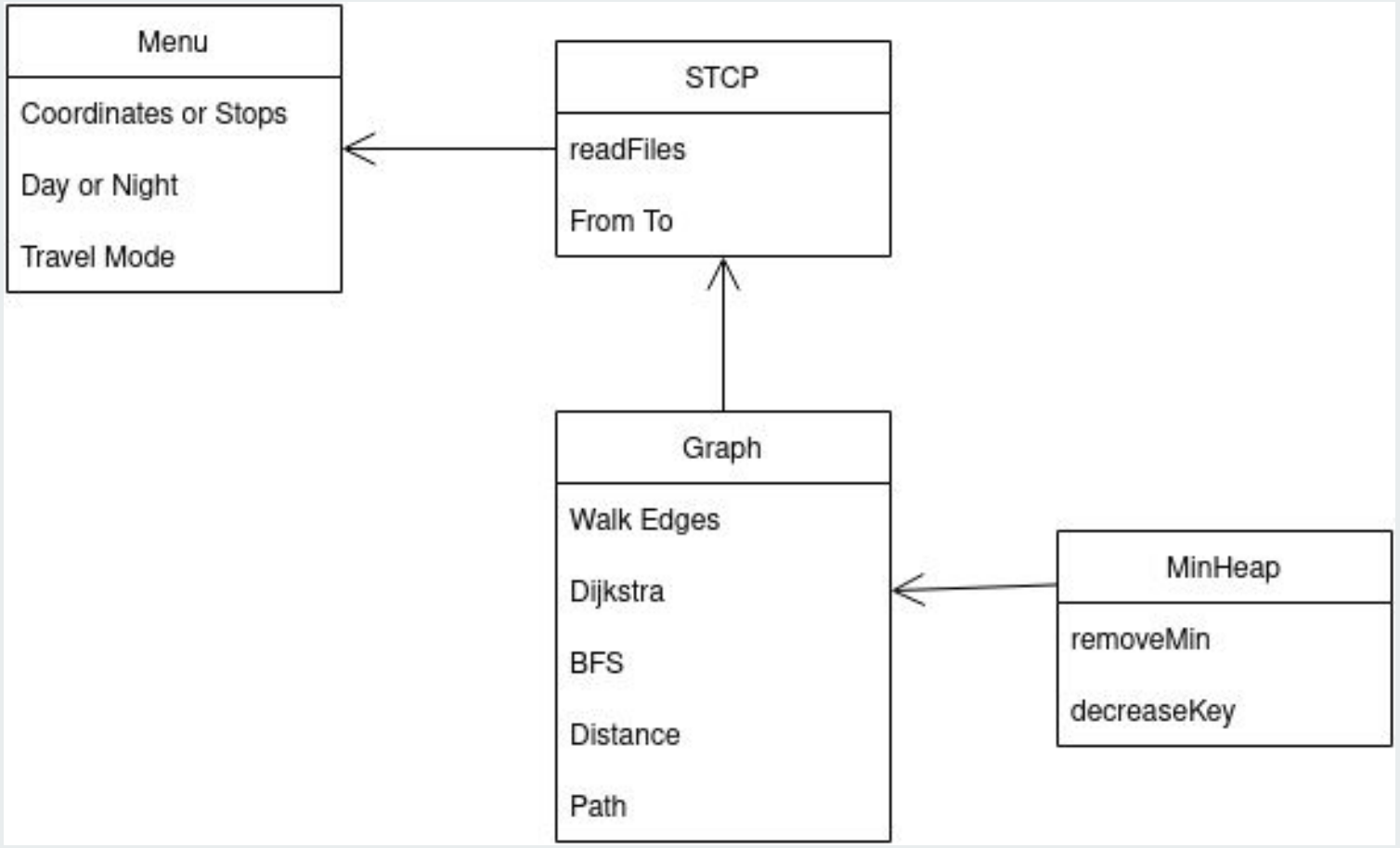
Guilherme Valler Moreira (up202007036)
José Luis Barbosa de Araújo (up202007921)
João de Oliveira Gigante Pinheiro (up202008133)

O problema proposto

Como certamente sabe, a Sociedade de Transportes Colectivos do Porto (STCP) é a empresa que gere a rede de autocarros do concelho do Porto (e também em algumas zonas limítrofes).

Este trabalho tem como objetivo implementar um sistema capaz de providenciar ajuda para quem quer usar a rede dos STCP para se deslocar.

Diagrama de classes



Leitura do Dataset da STCP

- Para a leitura do dataset, criamos funções na classe STCP que têm como objetivo povoar a nossa aplicação com as paragens e linhas;

- Com essa informação, povoamos um grafo (nodes: paragens, edges: linhas);

- Desenvolvemos ReadEdges e ReadStops em conformidade com o ReadLines.

```
void STCP::readLines(const string& myFile) {
    int pos;
    string line, code, name;
    ifstream file( s: myFile);
    string delimiter = ",";

    if(file.is_open()){
        int count = 1;
        getline( &: file,  &: line);

        while(!file.eof()){
            getline( &: file,  &: line);

            pos = line.find( str: delimiter);
            code = line.substr( pos: 0,  n: pos);
            name = line.substr( pos: pos+1,  n: line.size()-pos);

            //If it is nighttime we are looking at
            if (code.find( c: 'M') != std::string::npos && this->time == "M") {
                lines.insert( x: make_pair( x: code,  y: name));
                readEdges(code);
            }

            //If it is daytime we are looking at
            else if (this->time.empty() && code.find( c: 'M') == std::string::npos) {
                lines.insert( x: make_pair( x: code,  y: name));
                readEdges(code);
            }
            count++;
        }
        file.close();
    }
}
```

Grafo usado para representar o dataset

- **Graph:**
  - n / nodes
  - Direção
- **Edge:**
  - Destino
  - Distância entre paragens
  - Linha com a qual faço travessia
- **Node:**
  - Paragens para as quais posso ir
  - Distância, usado no algoritmo de cálculo
  - Precedente
  - Linhas Precedentes
  - Visitado, usado pelo algoritmo
  - Nome da paragem
  - Código respetivo
  - Zona pertencente
  - Coordenadas

```
class Graph {

    struct Edge {
        int dest;   // Destination node
        double weight; // An integer weight
        string line;
    };

    struct Node {
        list<Edge> adj; // The list of outgoing edges (to adjacent nodes)
        double dist;
        int pred = 0;
        vector<string> predLines = {};
        bool visited;
        string name;
        string code;
        string zone;
        double latitude;
        double longitude;
    };

    int n;              // Graph size (vertices are numbered from 1 to n)
    bool hasDir;        // false: undirect; true: directed
    vector<Node> nodes; // The list of nodes being represented
```

Origem/Destino: Como pode o utilizador indicá-los

No que toca a escolher a origem e destino, implementamos um menu com duas abordagens possíveis:

- Através dos códigos das paragens (1);
- Através de coordenadas (latitude e longitude) e de uma distância máxima a caminhar indicada pelo utilizador, para a qual disponibilizamos as paragens dentro dessa mesma (2).

```
void Menu::procedureStops() {
    string departure;
    string arrival;

    cout << "Choose your departure stop:" << endl;
    cin >> departure;

    cout << "Choose your arrival stop:" << endl;
    cin >> arrival;

    string choiceString = travelMode();

    stcp.fromTo( a: departure,  b: arrival,  choice: choiceString);
}
```

(1)

```
void Menu::procedureCoordinates() {
    double depLat = 0, depLon = 0;
    double arrLat = 0, arrLon = 0;


    cout << "Choose your departure latitude: " << endl;
    cin >> depLat;
    cout << "Choose your departure longitude: " << endl;
    cin >> depLon;

    cout << "Choose your arrival latitude: " << endl;
    cin >> arrLat;
    cout << "Choose your arrival longitude: " << endl;
    cin >> arrLon;

    string start = stcp.auxDeparture(depLat, depLon);
    string end = stcp.auxArrival(arrLat, arrLon);

    string choiceString = travelMode();

    stcp.fromTo( a: start,  b: end,  choice: choiceString);
}
```

(2)

Como quero fazer a minha viagem

No menu, implementamos 4 opções de viagem:

- Menor distância;
- Menor trocas de linhas;
- Menor número de paragens;
- Menor número de zonas (mais barato visto que se paga por número de zonas que se atravessa).

```
case 1:
    choiceString = "shortest";
    break;

case 2:
    choiceString = "lessChanges";
    break;

case 3:
    choiceString = "lessStops";
    break;

case 4:
    choiceString = "lessZones";
    break;

default:
    choiceString = "shortest";
```

Dijkstra

Usado no cálculo de caminho com: menos mudanças de linha, mais curto e menor número de zonas.

```
MinHeap<int, double> q(n, notFound: -1);

for(int v=1; v<nodes.size(); ++v){
    nodes[v].dist=INT_MAX;
    q.insert( key: v, value: INT_MAX);
    nodes[v].visited=false;
}

nodes[0].visited=true;
nodes[s].dist=0;
q.decreaseKey( key: s, value: 0);
nodes[s].pred = s;
nodes[s].predLines = {};
for(auto e :Edge : nodes[s].adj){
    nodes[s].predLines.insert( position: nodes[s].predLines.begin(), x: e.line);
}
```

```
while(q.getSize()>0) {
    weight = q.getValue();
    int u = q.removeMin();

    nodes[u].visited=true;
    if(nodes[u].dist==INT_MAX) break;


    for(Edge& e: nodes[u].adj){
        int multiplier = 1;
        if(e.line=="walk" && type!="shortest") multiplier += 2; // walk takes more time


        // Less Zones
        if(nodes[e.dest].zone!=nodes[u].zone && type=="lessZones") multiplier = 1000; // zone change


        bool flag = false;
        // Posible ways from where I came to u
        for(const auto& l :const string & : nodes[u].predLines){
            if(e.line==l) flag = true; // can continue in the same line
        }

        // if changed line
        if(!flag && u!=s){
            multiplier += 15;
            if(type=="lessChanges") multiplier += 100;
        }


        if(!nodes[e.dest].visited && weight + e.weight * multiplier <= q.getValue( k: e.dest)){
            if(weight + e.weight * multiplier < q.getValue( k: e.dest)) nodes[e.dest].predLines = {};
            if(e.line!="walk")
                nodes[e.dest].predLines.insert( position: nodes[e.dest].predLines.begin(), x: e.line); // precedent lines

            nodes[e.dest].dist = nodes[u].dist + e.weight; // update dist (real km dist)
            q.decreaseKey( key: e.dest, value: weight + e.weight * multiplier); // update dist (weighted dist)
            nodes[e.dest].pred = u; // precedent stop
        }
```

BST

Usado no cálculo de caminho com menos paragens (grafo não pesado).

O algoritmo vai calcular quantas paragens tenho que atravessar até chegar a qualquer ponto do grafo.

Esse valor é guardado na informação do Node.

```
void Graph::bfs(int a, int b) {
    for (int v=1; v<=n; v++) nodes[v].visited = false;
    queue<int> q; // queue of unvisited nodes

    q.push( x: a);
    nodes[a].dist=0;
    nodes[a].visited=true;

    while (!q.empty()) { // while there are still unvisited nodes

        int u = q.front(); q.pop();

        for (const auto& e :const Edge & : nodes[u].adj) {
            int w = e.dest;

            if (!nodes[w].visited) {
                q.push( x: w);
                nodes[w].pred = u;
                nodes[w].visited = true;
                nodes[w].dist = nodes[u].dist + e.weight;
                //nodes[w].dist = nodes[u].dist + 1;
            }

            if(w==b) break;
        }
```

Possibilidade de caminhar

Mudança de Autocarro: Apenas numa mesma paragem? Andar a pé até paragem vizinha? Outros?

Implementamos a função createWalkEdges() para que encontre todas as estações dentro de um raio de 0.2km da estação atual.

```
void Graph::createWalkEdges() {
    for(int i=0; i<nodes.size(); ++i){ // Stop x
        for(int j=i+1; j<nodes.size(); ++j){ // Stop y
            double distance12 = getDistance( lat1: nodes[i].latitude,  long1: nodes[i].longitude,  lat2: nodes[j].latitude,  long2: nodes[j].longitude); // Distance between x - y
            if(distance12 < 0.2){
                bool flag = false;
                for(const auto& e :const Edge & : nodes[i].adj) {
                    if(e.dest==j) {
                        flag = true;
                    }
                }
                if(flag) continue;

                addEdge( src: i,  dest: j,  weight: distance12,  line: "walk");
                addEdge( src: j,  dest: i,  weight: distance12,  line: "walk");
            }
```

Descrição do interface com o utilizador

No que toca à interface do utilizador, existem algumas etapas:

1. Escolha entre viajar de dia ou de noite;

```
1) Travel at day
2) Travel at night
Choice:
```

2. Escolha entre escrever os códigos das paragens ou as coordenadas requeridas (latitude e longitude).

```
1) Provide Stops
2) Provide Coordinates
Option: 1
Choose your departure stop:
BS8
Choose your arrival stop:
PRR3
```

```
2) Provide Coordinates
Option: 2
Choose your departure latitude:
41.14885
Choose your departure longitude:
-8.61043
Choose your arrival latitude:
41.23281
Choose your arrival longitude:
-8.62381
How far can the departure stop be from you?
0.05
Stops near you:
AAL2 (code) || AV. ALIADOS (name);
AAL4 (code) || AV.ALIADOS (name);
What stop code?
Option:
```

Descrição do interface com o utilizador (continuação)

Por fim escolhe-se qual o trajeto a ser tomado, de acordo com as 4 opções previamente referidas.

```
How do you want to travel?
1) Shortest distance
2) Less line changes
3) Less stops
4) Cheapest way (less zones)
Option:
```

Obtemos o caminho da seguinte forma:

```
Name                        |Code     |Zone     |Travel Dist   |
NATÁRIA                     NAT2      PRT1      0
BICA VELHA                   BVLH3     PRT1      0.290318       803  206  204
SILVA PORTO                  SVP2      PRT1      0.586915       803  206  204
VALE FORMOSO                 VFM3      PRT1      1.00211        803  206  204
VALE FORMOSO                 VFM1      PRT1      1.1631         803
CAMPO LINDO                  CPL1      PRT1      1.3359         803
LGO.CAMPO LINDO              LGCL1     PRT1      1.70509        803

UNIV. F. PESSOA              UFP1      PRT3      2.02835        803
IGREJA DE PARANHOS           IPRN1     PRT3      2.3433         803
ISEP/AGRA                    ISEP3     PRT3      2.66047        803
ESCOLA SUPERIOR SAÚDE       ESS1      PRT3      2.90853        803
PÓLO UNIVERSITÁRIO (METRO)  PUNV      PRT3      3.1804         803
FACULDADE DE ECONOMIA        FEP2      PRT3      3.47249        803
FACULDADE DE ENGENHARIA      FEUP1     PRT3      3.78095        803
3.78095 Km travelled
13 Stops
```

Destaque de funcionalidade (o que nos deixou mais orgulhoso)

- Conseguirmos desenvolver algumas funcionalidades adicionais, visto que completamos todas as essenciais;

- Na generalidade, acreditamos que o trabalho foi desenvolvido segundo os mesmos padrões de qualidade, mas algumas funcionalidades das quais nos empenhamos mais foram:

Criação das passagens pedonais entre paragens.

Diferença no cálculo de peso de uma aresta para obtenção de um melhor caminho.

Escolha de viagem entre dia e noite.

Principais dificuldades encontradas

Tivemos maiores dificuldades no desenvolvimento do algoritmo da menor troca de linhas e sentimos que o de menor zonas poderia ter sido aprimorado (com base nos resultados obtidos).

Tentamos igualmente desenvolver a MST, mas sem sucesso.

O trabalho foi dividido igualmente por todos os membros.

Obrigado pela atenção!