Universidade do Porto

FEUP Faculdade de Engenharia

Planeamento de Itinerários Multimodais

Concepção e Análise de Algoritmos
MIEIC - 2014/2015

23 de Abril de 2015

Grupo D

Luís Figueiredo - up201304295 - up201304295@fe.up.pt
João Silva - up201305892 - up201305892@fe.up.pt
Pedro Castro - up201304205 - up201304205@fe.up.pt

Índice

Introdução

Atualmente o número de habitantes das designadas “smart-cities” tem vindo a aumentar significativamente e com isso o número de transportes particulares.

Para contrariar este crescimento de viaturas a circular, as cidades investiram em meios de transporte públicos que permitissem aos seus habitantes se deslocar sem problemas para praticamente todos os locais da cidade.

Como seria previsível, é várias vezes impossível alcançar todas as áreas através de um único meio de transporte, sendo por isso necessários transbordos para outros transportes.

As empresas responsáveis por esses meios de transporte públicos fornecem regularmente os horários e trajetos disponíveis assim como o custo associado a cada um deles ficando ao critério do utilizador definir o melhor percurso para atingir o seu objetivo de viagem.

A tarefa de escolher o melhor percurso nem sempre é fácil tendo em conta que o número de rotas disponíveis é cada vez maior.

Visando facilitar a tarefa desenvolvemos uma plataforma que permitisse ao utilizador definir o seu ponto de partida e o seu ponto de chegada e ser informado sobre o melhor trajeto a ser efetuado. Com “melhor trajeto” pretende-se referir o trajeto com menor custo, menor distância percorrida ou o mais rápido.

Neste relatório iremos explicar como funciona o programa e como foi o seu desenvolvimento.

Formalização do Problema

Para desenvolver a plataforma foi necessário criar uma estrutura de dados que contivesse todos os locais de paragem dos variados meios de transporte públicos abordados no projeto, designadamente o Metro e os autocarros. Em acrescento, a mesma estrutura contempla a distância, o tempo de viagem e o custo associado ao troço, considerando também que é possível deslocar-se a pé.

Os dados de entrada esperados do utilizador são:

- **S:** Local de partida.
- **D:** Local de chegada.
- **P:** Preferência de percurso (mais rápido, menor distância ou mais económico).
- **G(S, D, P)**

Os dados de saída são os seguintes:

- **V(S, D):** Conjunto dos vértices que partindo de S (partida) atingem D (destino) segundo a preferência do utilizador.
- **E(S, D):** Conjunto de arestas que unem os vértices que pertencem ao conjunto resultado final.

A informação de saída é apresentada ao utilizador de forma gráfica seguida por um conjunto de instruções que guiam o utilizador ao seu destino.

O programa está limitado à cidade do Porto e como foi referido anteriormente, ao Metro e autocarros. Além disso foi considerado que o percurso entre locais segue uma linha reta (por questões de simplificação) e o tempo é apenas uma estimativa tendo em conta a velocidade média do meio de transporte e a distância do percurso. Cada local é obrigatoriamente atingível.

Descrição da Solução

Os dados utilizados no programa são reais e foram obtidos do site da STCP e do OpenStreetMaps.

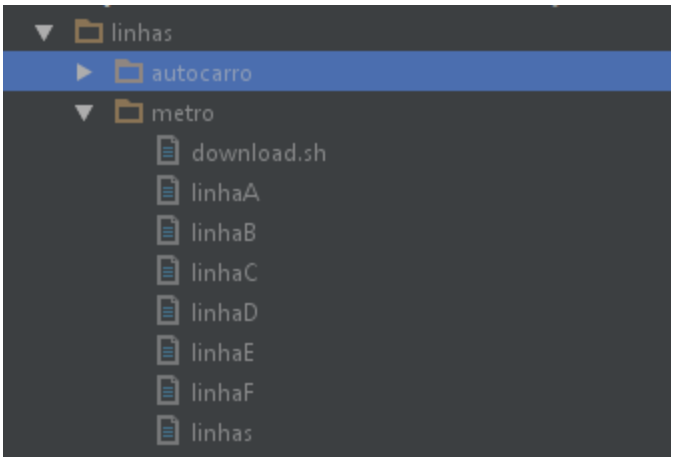

Fig. 1 - Ficheiros das linhas

Todos eles vinham numa formatação específica e foi por isso necessário criar um parser que extraísse os dados necessários à execução do código.

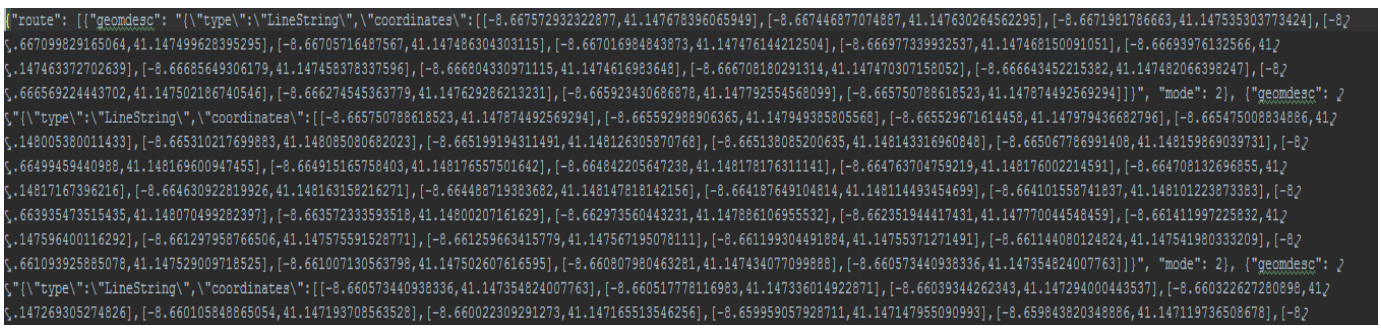

Fig. 2 - Formatação do ficheiro das linhas de autocarro

Para facilitar o processamento da informação foi efetuado um pré-processamento que removeu os locais que eram repetidos e as informações que eram desnecessárias.

Visando guardar as informações relativas aos locais de paragem dos meios de transporte abordados foram utilizados grafos pesados não restringido a grafos acíclicos.

- Cada nó do grafo representa um local que contém pelo menos uma paragem e uma linha de transporte disponível. Em cada nó é guardado o conjunto de arestas que tem esse mesmo nó como ponto de partida.

- A cada aresta está associado um meio de transporte e um destino, tendo essa aresta a direção do destino. Além desta informação é guardado o custo, a distância e o tempo de viagem associada à mesma.

- Grafo G = (Vértice V -> local, Aresta A -> ligação entre locais)

Para calcular o trajeto pretendido foram utilizados dois métodos.

- O método **A* (A Star)** sendo a complexidade temporal esperada é de O(|E| log|V|) em que “E” é o número de arestas (“edges”) e “V” de vértices (“vertex”). A complexidade espacial deste método é de ordem “O(1)”.
- O método de **Dijkstra** tem também uma complexidade temporal de O(|E| log|V|) em que “E” é o número de arestas (“edges”) e “V” de vértices (“vertex”). A complexidade espacial deste método de ordem “O(1)”.

O método A* baseia-se no método de Dijkstra. Contudo tem uma grande vantagem em relação a este uma vez que privilegia os vértices que estão mais próximos do vértice de destino baseando-se no princípio de que o melhor caminho será muito provavelmente aquele que estiver mais perto do destino.

Refrisando o que já foi referido anteriormente, a função objetivo é a de reduzir ao máximo o custo, o tempo ou a distância necessária à realização da viagem.

```
template <class T>
void Graph<T>::dijkstraShortestPath(Vertex<T> &source, Vertex<T> &dest, Preference preferencia){
    for (unsigned int i = 0; i < vertexSet.size(); i++) {
        vertexSet[i]->path = NULL;
        vertexSet[i]->cost = INT_INFINITY;
        vertexSet[i]->processing = false;
    }

    double costEdge;

    Vertex<T> *v = &source;
    v->cost = 0;

    vector<Vertex<T> *> pq;
    pq.push_back(v);

    make_heap(pq.begin(), pq.end());

    while (!pq.empty()) {

        v = pq.front();
        pop_heap(pq.begin(), pq.end());
        pq.pop_back();

        Vertex<T> *w;

        for (unsigned int i = 0; i < v->adj.size(); i++) {
            w = v->adj[i]->dest;

            if (preferencia == PRICE) {
                costEdge = v->adj[i]->price;
            } else if (preferencia == DISTANCE) {
                costEdge = v->adj[i]->distance;
            } else if (preferencia == TIME) {
                if (v->adj[i]->transport == METRO) {
                    costEdge = v->adj[i]->distance / 50;
                } else if (v->adj[i]->transport == BUS) {
                    costEdge = v->adj[i]->distance / 20;
                } else {
                    costEdge = v->adj[i]->distance / 5;
                }
            }

            if (v->cost + costEdge < w->cost) {
                w->cost = v->cost + costEdge;
                w->path = v;

                //se ja estiver na lista, apenas a actualiza
                if (!w->processing) {
                    w->processing = true;
                    pq.push_back(w);
                }

                make_heap(pq.begin(), pq.end(), vertex_greater_than<T>());
            }
        }
    }
}
```

Fig. 3 - Algoritmo de Dijkstra

```
template <class T>
void Graph<T>::aStarShortestPath(const Vertex<T> &source, const Vertex<T> &dest, Preference preferencia) {
    for(unsigned int i = 0; i < vertexSet.size(); i++) {
        vertexSet[i]->path = NULL;
        vertexSet[i]->cost = INT_INFINITY;
        vertexSet[i]->processing = false;
    }

    Transport lastTransport;

    double xGeoSource, yGeoSource, xGeoDest, yGeoDest, costEdge, distanceToDest;

    xGeoDest = (((dest.x - MARGIN) * DELTAH) / HSIZE) + XINICIAL;
    xGeoDest *= LONGITUDE_UNIT;
    yGeoDest = (((dest.y - MARGIN) * DELTAV) / VSIZE) + YINICIAL;
    yGeoDest *= LATITUDE_UNIT;

    Vertex<T>* v = &source;
    v->cost = 0;

    vector< Vertex<T>* > pq;
    pq.push_back(v);

    make_heap(pq.begin(), pq.end());

    while( !pq.empty() ) {

        v = pq.front();
        pop_heap(pq.begin(), pq.end());
        pq.pop_back();

        if(v->getInfo() == dest)
            break;

        xGeoSource = (((v->x - MARGIN) * DELTAH) / HSIZE) + XINICIAL;
        xGeoSource *= LONGITUDE_UNIT;
        yGeoSource = (((v->y - MARGIN) * DELTAV) / VSIZE) + YINICIAL;
        yGeoSource *= LATITUDE_UNIT;

        for(unsigned int i = 0; i < v->adj.size(); i++) {
            Vertex<T>* w = v->adj[i]->dest;

            distanceToDest = sqrt(pow(xGeoDest - xGeoSource, 2) + pow(yGeoDest - yGeoSource, 2));

            if(preferencia == PRICE) {
                costEdge = distanceToDest * v->adj[i]->price;
            } else if(preferencia == DISTANCE) {
                costEdge = distanceToDest * v->adj[i]->distance;
            } else if(preferencia == TIME) {
                if(v->adj[i]->transport == METRO) {
                    costEdge = distanceToDest * v->adj[i]->distance / 25;
                } else if(v->adj[i]->transport == BUS) {
                    costEdge = distanceToDest * v->adj[i]->distance / 15;
                } else {
                    costEdge = distanceToDest * v->adj[i]->distance / 5;
                }
            } else if(preferencia == SWAP) {
                if(v->adj[i]->transport != lastTransport)
                    costEdge = distanceToDest * 1;
                else
                    costEdge = 0;
            }

            if(v->cost + costEdge < w->cost ) {
                w->cost = v->cost + costEdge;
                w->path = v;

                //se ja estiver na lista, apenas a actualiza
                if(!w->processing)
                {
                    w->processing = true;
                    pq.push_back(w);
                }

                lastTransport = v->adj[i]->transport;

                make_heap (pq.begin(),pq.end(),vertex_greater_than<T>());
            }
        }
    }
}
```

Fig. 4 - Algoritmo A Star

Diagrama de Classes

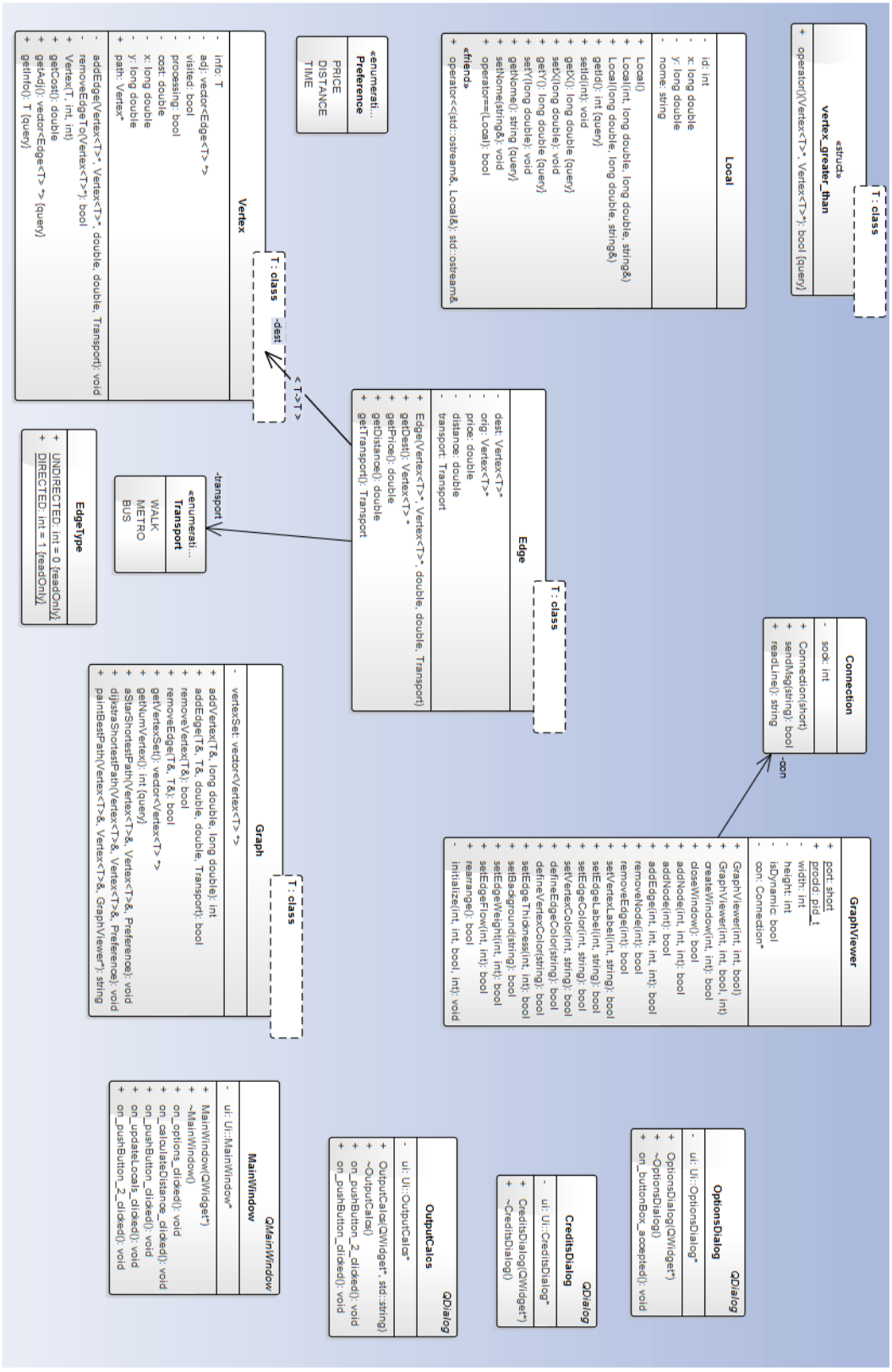

Fig 5. - Diagrama de Classes

Interface do Utilizador

Todo o projeto é baseado numa interface gráfica, tornando assim a sua utilização mais fácil e intuitiva.

Ao abrir o programa é aberta uma janela com várias opções.

Fig. 6 - Menu principal do programa

As funções de cada um dos botões são as seguintes:

- **Calcular caminho:** Calcula o caminho a percorrer tendo em conta as opções previamente definidas pelo utilizador.

Fig. 7 - Exemplo do resultado do melhor caminho para o utilizador se deslocar.

- **Mostrar mapa do Porto:** Apresenta ao utilizador o mapa total da cidade do Porto incluindo todas os locais e ligações entre eles.

Fig. 8 - Mapa do Porto

- **Opções:** Menu que inclui várias opções de personalização para que o utilizador escolha os locais de partida e chegada, preferência, resolução do ecrã e o algoritmo a ser utilizado no cálculo do caminho.

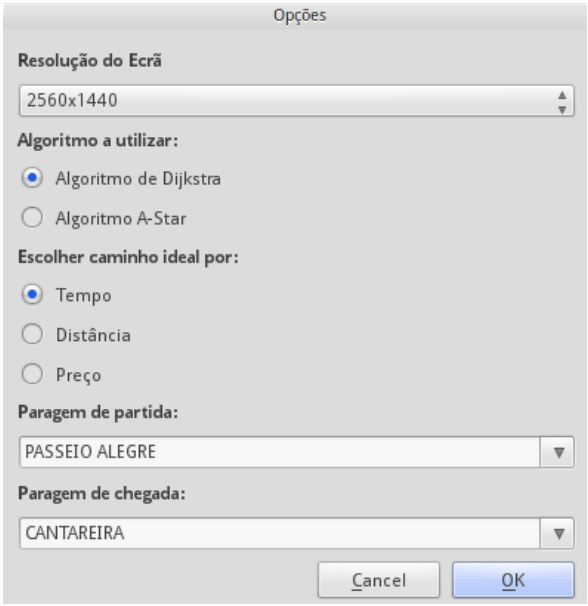

Fig. 9 - Menu de opções do programa

- **Créditos:** Apresenta os créditos dos seus criadores.

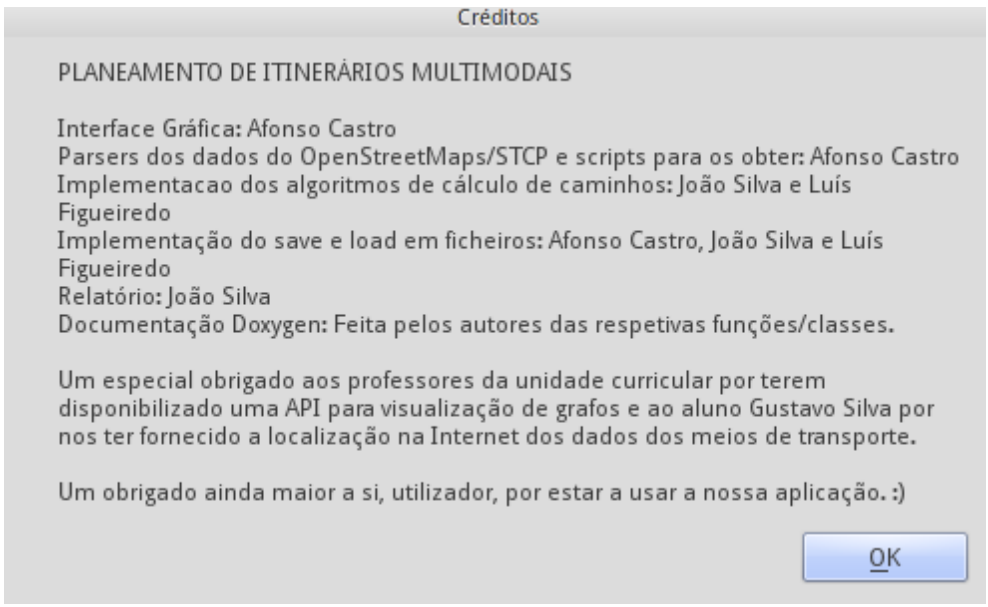

Fig. 10 - Créditos do programa

- **Sair:** Desliga o programa.

Dificuldades e Soluções

Durante o desenvolvimento do projeto foram encontradas várias dificuldades que exigiram soluções inteligentes.

A principal dificuldade e a qual iniciou o projeto foi a obtenção dos dados relativos às paragens de autocarro e Metro do Porto, sendo que a solução implementada foi a de criar dois scripts (um para cada meio de transporte) que acedessem à “API” da STCP e do OpenStreetMaps e que fizesse o download dos ficheiros que continham essa informação. Tal como seria de esperar os ficheiros vinham numa formatação diferente da pretendida o que levou à necessidade de criar “parsers” especialmente desenhados para a situação.

Na implementação do algoritmo A* houve problemas sobre como considerar arestas com pesos diferentes dependendo da preferência do utilizador. Para o solucionarmos criamos uma enumeração que contemplava as possíveis preferências do utilizador e dependendo da escolhida o peso da aresta escolhido é o correspondente à preferência.

Indicação do Esforço Por Elemento

Luís Figueiredo:

- Parsers dos ficheiros dos locais de paragem.
- Implementação dos algoritmos de cálculo de caminhos.
- Implementação dos métodos de guardar e carregar os ficheiros.

João Silva:

- Relatório.
- Parsers dos ficheiros dos locais de paragem.
- Implementação dos algoritmos de cálculo de caminhos.
- Implementação dos métodos de guardar os ficheiros.

Pedro Castro:

- Scripts para extrair os locais de paragem.
- Parsers dos ficheiros dos locais de paragem.
- Interface gráfica.
- Implementação dos métodos de carregar os ficheiros.
- Doxygen.