

Amadeu Prazeres Pereira **up201605646**@fe.up.pt

João Augusto dos Santos Lima **up201605314**@fe.up.pt

Nuno Tiago Tavares Lopes **up201605337**@fe.up.pt

**Conceção e Análise de Algoritmos**

*Sistema de evacuação*

2MIEIC05 – Grupo C

8 de abril de 2018

**Índice**

[Introdução 2](#_Toc510820498)

[Descrição do Tema – “Sistema de evacuação” 2](#_Toc510820499)

[Identificação e Formalização do Problema 2](#_Toc510820500)

[Solução Implementada 4](#_Toc510820501)

[Diagrama de Classes 6](#_Toc510820502)

[Casos de Utilização 7](#_Toc510820503)

[Classes 7](#_Toc510820504)

[Ficheiros 7](#_Toc510820505)

[Algoritmos 8](#_Toc510820506)

[ Haversine 8](#_Toc510820508)

[ Djikstra 8](#_Toc510820507)

[Programa 8](#_Toc510820509)

[Dificuldades 9](#_Toc510820510)

[Distribuição do Trabalho 10](#_Toc510820511)

[Conclusão 11](#_Toc510820512)

# Introdução

## Descrição do Tema – “Sistema de evacuação”

O projeto realizado no âmbito da unidade curricular de Conceção e Análise de Algoritmos destina-se ao processamento de um mapa real, neste caso uma área por nós escolhida *à priori,* e respetivo cálculo do caminho ideal de um local escolhido pelo utilizador, origem, para um local de destino, indicado também pelo utilizador.

Neste caso o “caminho ideal” é o caminho mais curto desde a origem até ao destino escolhido pelo utilizador. Este caminho é calculado tendo em conta o estado de cada estrada, ou seja, se esta se encontra transitável, ou não, e também quanto à lotação da mesma, ou seja, o número de carros que por ali passam.

Assim, identificamos como objetivos as seguintes funcionalidades do projeto:

* Possibilidade de o utilizador alterar o estado de uma estrada;
* Possibilidade de o utilizador calcular o percurso entre uma origem e um destino;
* Possibilidade de o utilizador ver o percurso definido para cada carro;
* Possibilidade de o utilizador remover um carro;
* Possibilidade de o utilizador ver o mapa com o estado de cada estrada e a sua lotação.

## Identificação e Formalização do Problema

Enquanto grupo de trabalho procurámos desde cedo ler e compreender o enunciado proposto e formalizar o problema, estruturando-o e repartindo problemas até que se tornassem elementares e de simples resolução.

Assim, identificamos e modulámos o nosso projeto em alguns tópicos:

* A escolha de uma área real de teste;
* A extração de informação dos ficheiros de texto;
* A criação da estrutura de dados a partir dos mesmos ficheiros:
* Criação de nós que contivessem a informação pretendida;
* Criação de arestas com os valores essenciais aos cálculos futuros;
* Criação conjunta do grafo, englobando nós e arestas;
* A determinação do caminho ideal no grafo usando o algoritmo adequado;
* A criação de um menu que serviria de interface com o utilizador.

# Solução Implementada

O primeiro passo foi, então, escolher a área a testar. No início tentamos utilizar o parser disponibilizado, mas chegamos à conclusão que este não seria útil pois a área que seria possível obter não faria muito sentido para o nosso tema. Como tal, decidimos criar nós os nosso próprios ficheiros, com isto já foi possível obter um mapa com o país inteiro.

Efetivamente, tendo os ficheiros prontos e uma ideia bem definida da área a utilizar, facilmente criámos as funções necessárias à leitura de ficheiros de texto, criando também os nós e arestas do nosso grafo.

Cada nó possui atributos necessários ao funcionamento dos algoritmos à frente explicados. Em cada nó está também armazenada a lista de arestas a que o vértice (nó) está ligado, as coordenadas (latitude e longitude), o seu ID único e o nome da cidade correspondente.

Cada carro é um objeto que contém informações sobre a sua rota (ID do nó da origem e ID do nó do destino), contém o seu ID único, e informações relativas ao percurso calculado pelos algoritmos utilizados.

As estradas, que representam um conjunto de arestas, contêm não só um ID único e o nome da mesma, mas também a informação sobre a existência de dois sentidos ou apenas um, o seu estado, ou seja, se está transitável ou não e a quantidade de carros (para ser fácil de testar definimos 10 carros como limite de cada estrada). Por fim, cada estrada tem o nó de destino.

Para o cálculo do caminho mais curto, uma vez que se trata de um grafo pesado com apenas números inteiros positivos, foi usado o algoritmo de ***Dijkstra*** sempre que necessário.

O mapa por nós gerado foi o seguinte:

*//foto do mapa*

Para melhor compreensão visual do estado das estradas decidimos atribuir a estas diferentes cores. Sendo assim:

* Cor Verde: Estrada transitável com poucos carros;
* Cor Laranja: Estrada transitável com alguns carros;
* Cor Vermelha: Estrada não transitável devido ao congestionamento dos carros;
* Cor Preta: Estrada não transitável por ter sido cortada;
* Cor Amarela: Percurso calculado para o utilizador.

**Existência de zonas inacessíveis e congestionamento nas estradas**

As zonas inacessíveis correspondem a estradas (edges), definidas entre dois vértices, que estão bloqueadas, ou seja, a escolha do percurso ideal não poderá utilizar essas conexões.

O congestionamento de uma estrada é calculado tendo em conta o número de carros que por lá vão passar. Para ser mais fácil de testar utilizamos 10 carros como o limite.

Posto isto, foi necessário alterar a forma de funcionamento “normal” dos algoritmos implementados ao longo das aulas práticas, visto que a outra opção seria eliminar do grafo as conexões bloqueadas e adicioná-las novamente quando o utilizar assim o quisesse.

# Diagrama de Classes

*//foto do diagrama de classes*

# Casos de Utilização

## Classes

**Interface** -> classe que contém as funções que realizam a interação com o utilizador.

**RoadNetwork** -> classe principal que alberga o grafo gerado e utilizado por todo o programa e que contem as principais funções de adição/alteração/remoção de elementos do grafo.

**Carros** -> classe que contem a informação sobre um carro, ou seja, o seu ID, o ID da origem, o ID do destino e se tem percurso ou não.

**Graph** -> classe representativa do grafo, contem todas as funções de alteração do grafo bem como todos os algoritmos utilizados para cálculo dos percursos.

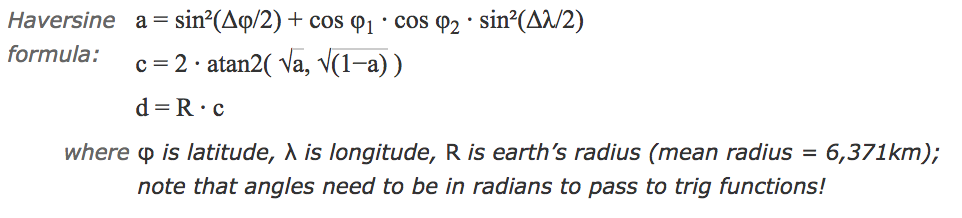
## Ficheiros

|  |  |  |
| --- | --- | --- |
| **Ficheiro** | **Informação do grafo (estrutura)** | **Equivalência Real** |
| edges.txt | Arestas(id, nome, transitável) | Estradas |
| nodes.txt | Nós (id, nome, latitude, longitude) | Cidades ou pontos de interseção de estradas |
| conections.txt | Conexões (idAresta, idNó1, idNó2) | Conexões entre estradas |
| cars.txt | Carros(id, idNóOrigem, idNóDestino) | Carros |

## Algoritmos

### Haversine – calculo do peso das arestas

Para o cálculo do peso das arestas utilizamos um algoritmo com o nome “Haversine” que permite calcular a distância entre dois pontos utilizando as suas coordenadas geográfica (latitude e longitude).



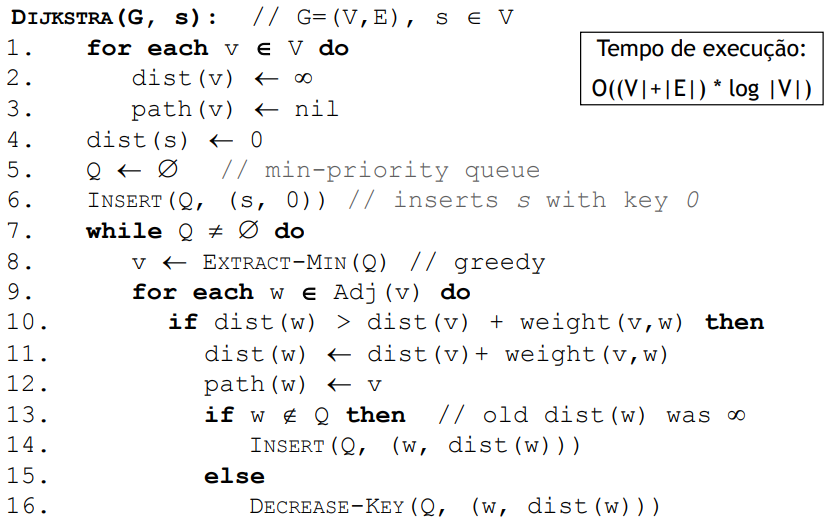
A distância obtida não corresponde à realidade pois o peso é calculado em linha reta entre os dois nós, enquanto que na realidade as estradas têm curvas e contracurvas.

### Djikstra

Foi usado o algoritmo de Djikstra para procurar o percurso mais curto em grafos pesados de valores positivos.

Em relação à complexidade espacial e temporal deste algoritmo, é possível afirmar:

* Espacialmente: O(N^2) onde N representa o número de nós;
* Temporalmente: O((|N|+|A|)\*log |N|) onde N representa o numero de nós e A representa o numero de arestas.

Pseudocódigo:

Este pseudocódigo foi adaptado para a utilização neste programa, de modo a funcionar segundo as nossas especificações.

É de seguida apresentado um gráfico com tempo de execução do algoritmo segundo um numero de nos e arestas:

## Programa

# Dificuldades

# Distribuição do Trabalho

**Amadeu Prazeres Pereira**

**João Augusto dos Santos Lima**

**Nuno Tiago Tavares Lopes**

# Conclusão