

Conceção e Análise de Algoritmos

*2017/2018*

***“TripPlanner”*** *Planeamento de viagens*

Trabalho realizado por:

* **Catarina Almeida** [up201606334@fe.up.pt](mailto:up201606334@fe.up.pt)
* **João Bernardo Sousa** [up201606649@fe.up.pt](mailto:up201606649@fe.up.pt)
* **Juliana Marques** [up201605568@fe.up.pt](mailto:up201605568@fe.up.pt)

MIEIC 2ºano turma 5 Grupo 1

**Índice**

[**1.** Introdução 2](#_Toc511243151)

[**1.1** Descrição sucinta do tema – *“Planeamento de viagens”* 2](#_Toc511243152)

[**2.** Identificação do problema e solução implementada 3](#_Toc511243153)

[**3.** Formalização do problema 5](#_Toc511243154)

[**3.1** Dados de entrada 5](#_Toc511243155)

[**3.2** Dados de saída 5](#_Toc511243156)

[**3.3** Funções objetivo 6](#_Toc511243157)

[**4.** Estrutura do programa 7](#_Toc511243158)

[**5.** Algoritmos implementados 8](#_Toc511243159)

[**5.1** Djikstra 8](#_Toc511243160)

[**5.1.1** Caminho mais curto 8](#_Toc511243161)

[**5.1.2** Caminho mais rápido 9](#_Toc511243162)

[**5.1.3** Caminho mais barato 9](#_Toc511243163)

[**5.1.4** Caminhos com transportes favoritos 10](#_Toc511243164)

[**5.2** Gerar linhas de autocarro e metro 10](#_Toc511243165)

[**6.** Casos de utilização 12](#_Toc511243166)

[**8.** Principais dificuldades encontradas no desenvolvimento do trabalho 13](#_Toc511243167)

[**9.** Esforço dedicado por cada elemento do grupo 14](#_Toc511243168)

[**9.1** Desenvolvimento do código 14](#_Toc511243169)

[**9.2** Análise do código e teste 14](#_Toc511243170)

[**9.3** Desenvolvimento do relatório 14](#_Toc511243171)

[**9.4** Desenvolvimento da interface e tratamento de erros de input 14](#_Toc511243172)

[**10.** Conclusão 15](#_Toc511243173)

[**11.** Bibliografia 16](#_Toc511243174)

# **1.** Introdução

## **1.1** Descrição sucinta do tema – *“Planeamento de viagens”*

O projeto realizado no âmbito da unidade curricular de Conceção e Análise de Algoritmos destina-se ao processamento de um mapa real. O nosso tema, tema 11- TripPlanner: itinerários para transportes públicos, destina-se principalmente a facilitar e assim incentivar o utilizador a usar os transportes públicos (autocarro e metro), de acordo com os seus interesses, da forma mais eficiente.

Logo, no contexto das preferências do utilizador, o “caminho ideal” assume diferentes significados. Podendo ser o caminho mais curto, o caminho mais rápido ou ainda o caminho mais barato. Para além disto ainda se terá em conta os transportes preferidos do utilizador. Assim sendo poderá ser sugerido um caminho só com autocarros, só com o metro ou então uma combinação dos dois, podendo por vezes ser necessário percorrer algumas zonas a pé.

# **2.** Identificação do problema e solução implementada

Como grupo de trabalho, consideramos que este problema poderia ser decomposto em 3 partes:

Numa **primeira fase**, tratamos de escolher uma área real de teste usando o parser fornecido, procedendo depois para a extração da informação gerada pelo parser em 3 ficheiros de texto.

Numa **segunda fase**, começamos a construir e a preencher a nossa estrutura de dados.

Começamos, então por criar nós e arestas que contivessem a informação extraída dos ficheiros de texto, dando origem assim ao nosso grafo de base.

Consideramos o grafo base, como contendo todos os nós e arestas da área escolhida, sendo assim podemos dizer que corresponde a todos os caminhos acessíveis, podendo alcançar cada nó do grafo, local de interesse, a pé.

Depois, começamos então a gerar de forma aleatória quais seriam as paragens de autocarro e metro, sendo que estas se sobrepõem aos nós iniciais do grafo base.

No final da construção da nossa estrutura, obtivemos um grafo constituído pela sobreposição de 3 grafos:

* **grafo base**, contém todos os nós, locais de interesse, e arestas, caminhos possíveis;
* **grafo de autocarro**, sobrepõem-se ao grafo base, contém menos nós e arestas que o grafo base, corresponde aos caminhos que podem de feitos através de autocarros, os nós seriam as paragens de autocarro e as arestas, as ruas que ligam as paragens;
* **grafo de metro**, tal como o de autocarros, estes também se sobrepõem ao inicial, correspondendo aos caminhos que podem ser percorridos de metro, os seus nós correspondem ás paragens de metro, e as arestas ás ruas que ligam as paragens.

Numa **terceira fase**, desenvolvemos todas as funcionalidades pedidas através dos algoritmos necessários, como o Dijkstra e a Haversine.

Implementamos então funções que nos devolvem o caminho “ótimo” dependendo dos parâmetros a minimizar:

* caminho ótimo com minimização do custo (*custo = Σ preço por distância*);
* caminho ótimo com minimização da distância (*distância calculada através da latitude e longitude de cada nó extraída dos ficheiros de texto*);
* caminho ótimo com minimização do tempo ( *, sendo v = velocidade média, a distância e o o intervalo de tempo*) A velocidade será estipulada á priori, sendo que as velocidades consideradas foram:
* a pé 5 km/h;
* de autocarro 18 km/h;
* de metro 28 km/h;
* caminho ótimo utilizando apenas autocarros (*menor distância*);
* caminho ótimo utilizando apenas metro (*menor distância*);

Numa **quarta fase**, desenvolvemos um menu, como interface com o utilizador.

# **3.** Formalização do problema

## **3.1** Dados de entrada

Os dados de entrada serão um grafo que representa uma rede de transporte. Usaremos cenários da vida real para testar as nossas implementações dos algoritmos. Os dados são obtidos a partir do openStreetMaps, que gerou 3 ficheiros de texto. Esses ficheiros de texto são posteriormente analisados, extraindo-se destes os dados necessários para o preenchimento do nosso grafo.

## **3.2** Dados de saída

Os dados de saída, serão um conjunto de nós que consideramos pertencer ao caminho “ótimo” que serão gerados de acordo com as preferências do utilizador.

Usando uma combinação do metro, autocarro e a pé:

* Caminho mais curto;
* Caminho mais rápido;
* Caminho mais barato;

Usando um transporte preferido (metro ou autocarro):

* Caminho mais curto;

## **3.3** Funções objetivo

* Minimizar a distância percorrer:
* Minimizar o tempo de deslocação:
* Minimizar o custo de deslocação:

Símbolos:

* **C-** Corresponde ao conjunto de caminhos possíveis;
* **distancia (i, j) -**corresponde á função que calcula a distância da aresta ();
* **tempo(i, j) -**corresponde á função que calcula o tempo em segundos para percorrer a distância da aresta ();
* **custo(i, j) -**corresponde á função que calcula o custo em euros para percorrer a aresta ();
* **aresta ()-** corresponde a aresta que liga o nó i ao nó j;
* **nFinal-** nó final;
* **nInicial-** nó inicial;

# **4.** Estrutura do programa

Para conseguirmos desenvolver com sucesso o nosso projeto, criamos algumas classes, com as seguintes responsabilidades:

* **Graph**-> é a nossa classe principal.
  + Trata da organização da estrutura do próprio grafo.
  + Inclui todas as funções relacionadas com o algoritmo *Dijkstra.*
  + Inclui todas as funções de geração de resultado.
  + Inclui todas as funções de otimização de resultado.
* **Node** -> Classe que inclui todas as informações relativas a cada nó.
  + Número identificador único.
  + Localização no espaço (Latitude e Longitude).
  + Arestas adjacentes
  + Outras informações utilizadas durante o processamento dos algoritmos de geração de trajeto.
* **Aresta** -> Classe que inclui todas as informações relativas a cada aresta.
  + Número identificador único;
  + Nome da rua
  + Tipo de veículo
  + Outras informações utilizadas durante o processamento dos algoritmos de geração de trajeto.
* **IDs** -> Classe auxiliar que incluí atributos estáticos de IDs de arestas e nós para que na geração de linhas de autocarro e de metro não existam conflitos.
* **AuxialiarMethods** -> Classe auxiliar
  + Inclui uma função de cálculo de distâncias entre dois pares de latitudes e longitudes através da fórmula de Haversine.
  + Inclui uma função de conversão de angulo de graus para radianos.
  + Inclui uma função de verificação da existência de ficheiros.

# **5.** Algoritmos implementados

## **5.1** Djikstra

**Análise analítica:**

**Análise empírica:**

### **5.1.1** Caminho mais curto

Para o cálculo do trajecto mais curto foi utilizado o *álgoritmo Dijkstra*, sendo que o peso de todas as arestas é previamente atualizado e toma o valor da distância em quilómetros entre nós. O cálculo desta distância é efectuado utilizando a *Fórmula de Haversine*, implementada na função *getDistance().*

### **5.1.2** Caminho mais rápido

Para o cálculo do trajecto mais rápido foi utilizado o álgoritmo Dijkstra, sendo que o peso de todas as arestas é previamente atualizado e toma o valor do tempo que se despende a percorrer essa aresta. Esse tempo é calculado pela divisão da distância em quilómetros entre nós pela velocidade média – em quilómetros por hora - do meio de transporte correspondente a cada aresta. O cálculo desta distância é efectuado utilizando a Fórmulade Haversine, implementada na funçao getDistance() e as velocidades médias correspodem às seguintes constantes:

*const int WALKVELOCITY = 5;*

*const int BUSVELOCITY = 18;*

*const int METROVELOCITY = 28;*

### **5.1.3** Caminho mais barato

Para o cálculo do trajecto mais barato foi utilizado o álgoritmo Dijkstra, sendo que o peso de todas as arestas é previamente atualizado e toma o valor da multiplicação da distância em quilómetros entre nós pelo preço por quilómetro do meio de transporte correspondente.

Uma fez que o que se pretende é calcular um trajeto que utilize transportes, para as arestas cujo transporte associado é andar a pé, o peso passa a ser a distância a multiplicar por um fator arbitrário. O cálculo desta distância é efectuado utilizando a Fórmulade Haversine, implementada na funçao getDistance() e os preços por quilómetro em centimos correspodem às seguintes constantes:

* *const int* BUSPRICE *= 10;*
* *const int* METROPRICE *= 40;*

### **5.1.4** Caminhos com transportes favoritos

Para o cálculo do trajecto tendo preferência por um meio de transporte foi utilizado o álgoritmo Dijkstra, sendo que o peso de todas as arestas é previamente atualizado e toma o valor da distancia a multiplicar por um fator arbitrario. O cálculo desta distância é efectuado utilizando a Fórmulade Haversine, implementada na funçao getDistance() e os fatores são os seguintes:

* fator para o meio de transporte preferido = 1;
* fator para o meio de transporte alternativo = 10;
* fator para caminhos a pé = 20;

## **5.2** Gerar linhas de autocarro e metro

**Análise analítica:**

* **P->** número de paragens/estações

Uma vez que os dados de linhas de autocarro e metro na cidade do Porto não são disponibilizados de forma publica, fomos obrigados a criar funções que geram de forma aleatória essa informação.

Para gerar linhas de autocarro utilizamos a função generateBusLines(). Esta função tem como parametro o número de paragens que cada linha deverá ter. Primeiramente, a função sorteia um *nó (N1)* pertencente ao grafo base. Seguidamente, cria um *nó* similar ao primeiro sendo que este é para utilização exclusiva de autocarros (NB1). Cria as ligações entre o nó original e nó de autocarros (N1 ↔ NB1). O próximo passo é sortear um nó adjacente ao primeiro (N2), criar um novo nó semelhante a este, mas só para autocarros (NB2), criar as ligações entre ambos (N2 ↔ NB2) e por fim criar as ligações de autocarros (NB1 ↔ NB2).

Executar em loop estes passos até obter a linha com o número de paragens pretendido.

Para gerar linhas de metro é utilizada a funçao generateMetroLines() e a abordagem é ligeiramente diferente. Esta funlão tem como parametro o número de paragens que cada linha deverá ter. Primeiramente a função sorteia um *nó* (N1) pertencente ao grafo base. Seguidamente, cria um nó similar ao primeiro sendo que este é para utilização exclusiva de metros (NM1). Cria as ligações entre o nó original e nó de autocarros (N1 ↔ NM1). O próximo passo é sortear um *nó* (N2) que esteja dentro de uma área de raio arbitrário cujo centro seja o nó original. Criar um nó semelhante a N2 sendo que este é só para metros (NM2). Criar as ligações entre ambos (N2 ↔ NM2) e por fim criar as ligações de metros (NM1 ↔ NM2).

Executar em loop estes passos até obter a linha com o número de paragens pretendido.

# **6.** Casos de utilização

O programa desenvolvido permite ao utilizador consultar qual o melhor trajeto entre dois pontos de acordo com a sua preferência, tendo em conta a existência de linha de metro e autocarro.

O nosso programa apresenta então as seguintes funcionalidades:

* Ler os nós e arestas do grafo a partir de três ficheiros .txt gerador pelo parser OSM2TXT através de ficheiros do OpenStreetMap.
* Criar um grafo com a informação extraída.
* Gerar de forma aleatória linhas de autocarro e de metro sobre o grafo original e acrescenta-las ao mesmo.
* Escolher o ponto inicial e final do trajeto pretendido através do ID dos respectivos nós.
* Escolher a opção de otimização do trajeto pretendida:
  + Caminho mais curto;
  + Caminho mais rápido;
  + Caminho mais barato utilizando transportes públicos;
  + Melhor caminho utilizando preferivelmente autocarros;
  + Melhor caminho utilizando preferivelmente metro;

## **8.** Principais dificuldades encontradas no desenvolvimento do trabalho

Durante a realização do projeto não surgiram muitas dificuldades a nível de programação, mas sim a nível de identificação e planeamento do mesmo.

Um dos nossos obstáculos foi a compreensão e utilização do *GraphViewer* e *Parser* que nos causou alguns problemas de compilação não esperados nos nossos IDE´s, atrasando desta forma o desenvolvimento do projeto.

Outro dos obstáculos que encontramos ao longo do desenvolvimento do projeto, foi a falta de tempo face aos elevados objetivos do trabalho e á elevada sobrecarga do curso, desenvolvimento de projetos de outras cadeiras ao mesmo tempo.

No final, todos os obstáculos foram ultrapassados, com alguma pesquisa, discussão e esforço coletivo.

# **9.** Esforço dedicado por cada elemento do grupo

O trabalho foi realizado em conjunto, utilizando a ferramenta GitKraken para sincronizarmos e organizarmos o nosso código. Reunimos-mos várias vezes na Feup, para discussão e desenvolvimento conjunto do projeto.

Consideramos por isso que todos os membros do grupo de esforçaram igualmente na estratificação do problema e compreensão do enunciado, tendo contribuído de igual forma, mas em áreas diferentes, para ao desenvolvimento do projeto.

## **9.1** Desenvolvimento do código

Realizada em conjunto, sendo que foi desenvolvido maioritariamente pelo João Bernardo.

## **9.2** Análise do código e teste

A análise e teste de código foi feita maioritariamente pela Catarina Almeida, com intervenção dos restantes elementos do grupo.

## **9.3** Desenvolvimento do relatório

O desenvolvimento do relatório foi realizado maioritariamente pela Juliana Marques, com participação e revisão dos restantes elementos do grupo.

## **9.4** Desenvolvimento da interface e tratamento de erros de input

O desenvolvimento da interface e do tratamento de inputs foi realizado maioritariamente pela Juliana Marques com contribuição dos restantes elementos do grupo.

# **10.** Conclusão

Com este projeto, conseguimos por em prática o conhecimento estudado ao longo do semestre, para além disso aprendemos de uma forma mais prática as vantagens e utilidade da utilização das novas estruturas de dados, como grafos, assim como diversos algoritmos úteis e aplicáveis para os diversos problemas do dia a dia. Para além disso desenvolvemos as nossas capacidades de trabalho de grupo, organização, discussão e partilha de ideias.

Concluímos, portanto, que os objetivos pretendidos com este projeto de grupo foram atingidos, quer a nível individual quer a nível coletivo, uma vez que cada elemento domina os temas lecionados na unidade curricular.

# **11.** Bibliografia

1. “Open Street Maps”, http://openstreetmaps.org