Instituto Superior de Engenharia de Lisboa

Licenciatura em Engenharia Informática e de Computadores

ALGORITMOS E ESTRUTURAS DE DADOS

Relatório do Problema da 3ª Série de Problemas

Docente

Prof.ª Cátia Vaz

Alunos

36368 Mihail Cirja

43552 Samuel Costa

Junho 2018

Índice

Índice de Figuras

Introdução

Este relatório, referente ao problema do terceiro de três trabalhos práticos, foi elaborado no âmbito da unidade curricular de Algoritmos e Estruturas de Dados e acompanha a continuação do estudo de estruturas de dados e de algoritmos associados, com particular ênfase nos grafos e em algoritmos de pesquisa sobre este tipo de dados abstrato.

Este problema constituiu uma integração de vários conteúdos abordados nas aulas, designadamente:

1. Carregamento em memória de dados seguindo uma convenção definida;
2. Utilização de um contentor que encapsula os dados por forma a minimizar o tempo de pesquisa;
3. Pesquisa em profundidade sobre um grafo não orientado;
4. Pesquisa de custo uniforme, nomeadamente explorando as potencialidades da pesquisa dinâmica efetuada pelo algoritmo de Dijkstra;
5. Aplicação de algoritmos de pesquisa sobre um grafo com duas classes de peso;

Ao longo do documento apresentam-se a descrição da aplicação desenvolvida, com base nos seus requisitos; a indicação dos *Abstract Data Type* (ADT) utilizados; a explicação das operações suportadas pela aplicação; uma breve reflexão em que se discutem os resultados obtidos e a avaliação experimental, explicitando-se conclusões referentes ao trabalho desenvolvido; proposta de desenvolvimentos futuros ao presente estudo; e, por fim, a bibliografia consultada.

1 Descrição da Aplicação

Foi proposto o desenvolvimento de uma aplicação para efetuar três tipos de pesquisa sobre um grafo que representa a rede do metropolitano de Lisboa:

- Pesquisa de todos os caminhos entre as estações a e b;
- Pesquisa pelo caminho mais curto entre as estações a e b;
- Pesquisa pelo percurso entre as estações a e b que impliquem o menor número de trocas de linha.

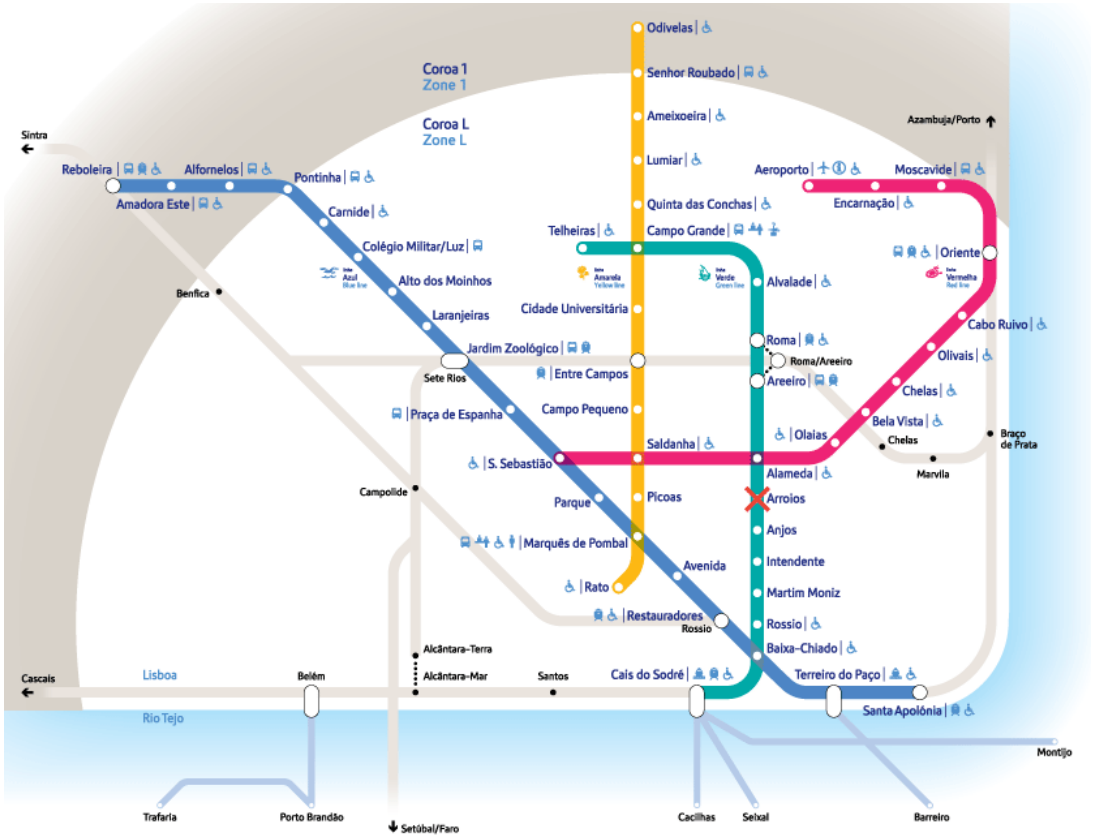

Figura 1 – diagrama da rede de metropolitano de Lisboa – fonte: https://www.metrolisboa.pt/viajar/mapas-e-diagramas/

Os dados usados na construção e preenchimento do grafo pertencem a dois ficheiros que foram fornecidos. Estes ficheiros foram uniformizados e apresentam conformidade com a seguinte convenção:

	Ficheiro de Linhas	Ficheiro de Estações
1ª linha	*N – número de linhas*	*N – número de estações*
Seguintes n linhas	*Linha[espaço]t* *…*	*Estação:linha* *…*
seguintes	*Linha1[espaço]Linha2[espaço]t*	*Est1:Est2-t*

Procurou-se estruturar o código por forma a diminuir a quantidade de código escrito, a fomentar a reutilização e a separação do código tendo em conta o contributo para o conjunto da aplicação. O diagrama apresentado na figura 1 dá conta da estrutura que emergiu do trabalho desenvolvido. Dá-se relevo ao facto de esta não se encontrar fechada, desde já porque depende do modo de apresentação dos dados e de interação com o utilizador, que neste caso é a consola, e da convenção ou do formato em que se aceitem os dados referentes às estações e às linhas de metro.

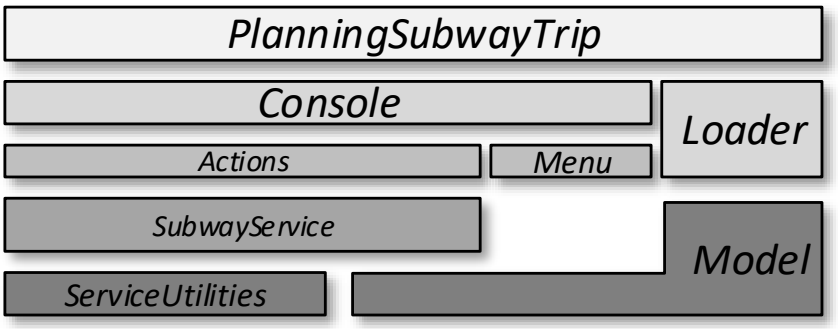

Figura 2 – diagrama lógico da aplicação Planning Subway Trip

Os dados em ficheiro são extraídos na classe Loader. A instância do grafo, que se encontra num contentor, é aí preenchida. Essa instância é então cedida à camada serviço. Esta opção de não restrição do acesso ao modelo por estas duas camadas têm em atenção a dimensão reduzida da aplicação. Em aplicações mais complexas poderia fazer sentido o estabelecimento de uma classe que encapsulasse as operações de acesso direto ao modelo.

Quando a aplicação é iniciada é instanciada a classe Loader e é chamado o método run de Console, que lista as opções do menu. A classe Actions contem métodos correspondentes às opções do menu e uma instância de SubwayService, sobre a qual efetua operações suportadas.

2 *Abstract Data Types*

Para dar resposta ao problema foi utilizado o ADT (*Abstract Data Type*) grafo, que foi implementado com recurso aos conceitos da programação orientada por objetos. Foi selecionado este ADT, pois permite a abstração no conceito de grafo não orientado da matemática, nomeadamente da teoria dos grafos. Seja um grafo G definido pelo conjunto V de todos os seus vértices e E de todas as suas arestas, é possível representar em memória os elementos destes conjuntos por recurso ao objecto Vertex, que representa um vértice do grafo e ao objecto Edge, que representa uma aresta do gráfico. Concretizou-se, no âmbito deste trabalho um vértice numa especialização deste, designada Station. Dada a tipologia do grafo estudado, designadamente por se tratar de um grafo esparso, em que$|E| \ll |V^2|$ [1], cada vértice contém uma lista de adjacências em vez de uma matriz de adjacências, o que fornece uma maneira compacta de representar os dados em memória.

A estrutura de dados utilizada para armazenar os dados do grafo foi uma tabela de dispersão dos pares <nome da estação,objeto estação> com encadeamento interno. Esta decisão é apoiada pela possibilidade de efetuar pesquisa em tempo constante.

Na verdade, poder-se-ia dizer que existem nesta aplicação dois grafos diferentes, um que é aquele representado por nós do tipo Station e arestas do tipo Edge, com referência para a próxima estação e o peso, relativo ao tempo de trânsito entre duas estações. Um segundo grafo seria composto por nós do tipo Line e arestas do tipo Edge, com a referência da linha que lhe está ligada e o peso referente ao tempo de comutação. Estes dois grafos foram compactados: os vértices Station do grafo têm referência para uma lista de adjacências das suas arestas e um array de Lines, vértices de um grafo que também contêm as arestas referentes às linhas que têm interseção naquela estação.

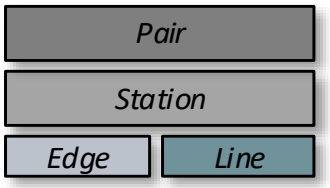

Figura 3 – diagrama lógico das entidades que representam o package model da aplicação

3 Operações

A aplicação desenvolvida suporta as seguintes operações:

- **allPaths a b**: permite obter a lista de todos os percursos entre duas estações de metro. Esta operação é realizada por intermedio de uma pesquisa em profundidade (Depth-First-Search).
- **fastestPath a b:** permite obter entre duas estações de metro um percurso que em média é mais rápido. Para obtenção do percurso mais rápido entre duas estações foi aplicado o algoritmo do Dijkstra.
- **pathWithLessChanges:** permite obter entre duas estações de metro um percurso que realize menos trocas de linha;

all Paths

Foi escrito o método `getAdjacent()` em Station que devolve uma lista das estações adjacentes do nó instanciado, como se mostra na figura 4. A figura 5 apresenta os detalhes de implementação do método `allPaths()`, que implementa uma pesquisa em profundidade primeiro. O algoritmo foi adaptado de [3].

```
public LinkedList<Station> getAdjacent(){
    LinkedList<Station> adj = new LinkedList<>();
    adj.add(this);
    Edge e = adjList;
    while(e!=null){
        adj.add(((Pair<String,Station>)e.connectsTo).v);
        e=e.next;
    }
    return adj;
}
```

Figura 4 – detalhe de implementação do método get adjacente na classe Station

```
public void allPathsFromTo( LinkedList<Station> visited) {
    LinkedList<Station> stations = visited.getLast().getAdjacent();
    // examine adjacent nodes
    for (Station station : stations) {
        if (visited.contains(station)) {
            continue;
        }
        if (station.getNome().equals(DESTINATION)) {
            visited.add(station);
            printPath(visited);
            visited.removeLast();
            break;
        }
    }
    for (Station station : stations) {
        if (visited.contains(station) || station.getNome().equals(DESTINATION)) {
            continue;
        }
        visited.addLast(station);
        allPathsFromTo(visited);
        visited.removeLast();
    }
}
```

Figura 5 – all paths.

Shortest Path

Para calcular o caminho menos pesado entre duas estações foi necessário adaptar o algoritmo de Dijkstra. Como tal, foi necessário escrever um método que dada uma lista de nós, permitisse obter o nó mais próximo e um outro método que permitisse calcular a distância mínima entre dois nós. Este algoritmo diferencia-se de outros algoritmos de custo uniforme pois é exaustivo e dinâmico, isto é, só depois de processar todos os nós é que está em condições de definir a menor distância entre nós e durante a sua execução o menor caminho entre dois nós pode ser mudado.

```
private static Station getLowestDistanceNode(List < Station > unsettledNodes) {
    Station lowestDistanceNode = null;
    int lowestDistance = Integer.MAX_VALUE;
    for (Station node: unsettledNodes) {
        int nodeDistance = node.distance;
        if (nodeDistance < lowestDistance) {
            lowestDistance = nodeDistance;
            lowestDistanceNode = node;
        }
    }
    return lowestDistanceNode;
}


private static void calculateMinimumDistance(Station evaluationNode,
                                             Integer edgeWeigh, Station sourceNode) {
    Integer sourceDistance = sourceNode.distance;
    if (sourceDistance + edgeWeigh < evaluationNode.distance) {
        evaluationNode.setDistance(sourceDistance + edgeWeigh);
        evaluationNode.predecessor = sourceNode;
        //DEBUG
        //System.out.println(evaluationNode);
    }
}
```

Figura 6 – detalhe de implementação dos métodos auxiliares ao algoritmo de Djikstra

No final da execução do algoritmo poderemos percorrer a lista desde nó de destino por via do seu predecessor até ao nó de origem. Lembre-se que como o algoritmo é dinâmico, a atribuição que é feita ao nó em avaliação do seu predecessor pode mudar durante a execução do algoritmo. A figura 7 mostra em detalhe a implementação de Dijkstra com recurso aos métodos auxiliares mostrados na figura 6. O algoritmo foi adaptado de [1] e [2].

```java
// Funtion that implements Dijkstra's single source shortest path
// algorithm for a graph represented using adjacency list
// representation
public void dijkstra ( Station src){
    src.setDistance(src.line[0].weight);

    List<Station> settledNodes   = new LinkedList<>();
    List<Station> unsettledNodes = new LinkedList<>();

        unsettledNodes.add(src);

        while (unsettledNodes.size() != 0) {
            Station currentNode = getLowestDistanceNode(unsettledNodes);
            unsettledNodes.remove(currentNode);
            Edge e = currentNode.adjList;
            while(e!=null){
                Station adjacentNode = ((Station)((Pair<String,Station>)e.connectsTo).getValue());
                Integer edgeWeight =e.weight + swapLinesWeight(currentNode,adjacentNode);
                if(!settledNodes.contains(adjacentNode)){
                    calculateMinimumDistance(adjacentNode, edgeWeight, currentNode);
                    unsettledNodes.add(adjacentNode);
                }
                e= e.next;
            }
            settledNodes.add(currentNode);
        }
        //print Path
        Station dst = ServiceUtilities.fetchStationFromNameOfStation(graph,DESTINATION);
        printShortestPath(dst);
    }
```

Figura 7 – Detalhe de implementação do algoritmo de Dijkstra

Foi ainda de relevo a introdução da noção de troca de linha, já que, ao calcular a distância entre dois nós, estes nós se podem encontrar em linhas diferentes, sendo necessário adicionar o tempo de troca de linha e o tempo de espera na plataforma da nova linha. A figura 8 mostra em detalhe como é calculado o peso adicional referente à troca de linha.

```java
// The concept of having to swap lines envolves predecessor of current, current and adjacent.
// Whenever current has more than one line and predecessor of current line is different from adjacent line
//there was a line swap. the time to be incremented is the avg waiting time of the new line and the swapping time.
private int swapLinesWeight(Station currentNode, Station adjacentNode) {
    //find the swappedLine
    Line[] linesFrst = currentNode.line;
    Line[] linesScnd = adjacentNode.line;
    for(int i=0;i<linesFrst.length;++i){
        for(int j=0;j<linesScnd.length;++j){
            Edge e= linesFrst[i].edge;
            while(e!=null){
                if(e.connectsTo.equals(linesScnd[j]))//return waiting time + swapping time
                    return e.getWeight()+linesScnd[j].weight;
                e=e.next;
            }
        }
    }
    return 0;
}
```

Figura 8 – método swappedLinesweight, que introduz a noção do peso associado a troca de linha

Path with less Changes

O algoritmo desenvolvido para a resolução deste exercício implementa uma transformação do algoritmo de Dijkstra, em que o critério de comparação para o cálculo da menor distância é o número de trocas. O algoritmo é incluído em detalhe na figura 9. A figura 10 apresenta os métodos auxiliares, que também são muito semelhantes àqueles utilizados no cálculo do caminho mais curto.

```
public void lessChanges(Station station) {
    station.nHops=0;

    Set<Station> settledNodes = new HashSet<>();
    Set<Station> unsettledNodes = new HashSet<>();

        unsettledNodes.add(station);

        while (unsettledNodes.size() != 0) {
            Station currentNode = getMinHopsNode(unsettledNodes);
            unsettledNodes.remove(currentNode);
            Edge e = currentNode.adjList;
            while(e!=null){
                Station adjacentNode = ((Station)((Pair<String,Station>)e.connectsTo).getValue());
                Integer hopNotHop = ServiceUtilities.isInTheSameLine(currentNode.predecessor,adjacentNode)?0:1;
                //System.out.println(adjacentNode);
                if (!settledNodes.contains(adjacentNode)) {
                    calculateMinimumHopDistance(adjacentNode, hopNotHop, currentNode);
                    unsettledNodes.add(adjacentNode);
                }
                e=e.next;
            }
            settledNodes.add(currentNode);
        }
    //print Path
    Station dst = ServiceUtilities.fetchStationFromNameOfStation(graph,DESTINATION);
    printShortestPath(dst);
}
```

Figura 9 – detalhe de implementação do método less changes

```
private static Station getMinHopsNode(Set < Station > unsettledNodes) {
    Station lowestDistanceNode = null;
    int lowestHopCount = Integer.MAX_VALUE;
    for (Station node: unsettledNodes) {
        int nodeHops = node.nHops;
        if (nodeHops < lowestHopCount) {
            lowestHopCount = nodeHops;
            lowestDistanceNode = node;
        }
    }
    return lowestDistanceNode;
}
private static void calculateMinimumHopDistance(Station evaluationNode,int hopNotHop,Station sourceNode){
    Integer sourceHops = sourceNode.nHops;
    if (sourceHops + hopNotHop < evaluationNode.nHops) {
        evaluationNode.nHops = sourceHops + hopNotHop;
        evaluationNode.predecessor = sourceNode;
    }
}
```

Figura 10- métodos auxiliares ao método less changes

4 Avaliação Experimental

A aplicação desenvolvida foi testada e pode ser consultada na pasta src que acompanha a entrega do presente relatório.

Para confirmar o correto funcionamento da aplicação foi realizada uma extensa bateria de testes, dos quais apresentamos os resultados mais relevantes. Assim, foram escolhidas duas estações com mais do que um caminho que as ligue. Também foi selecionada uma estação de origem que fica a um nó de distância de estações de troca e uma estação de destino numa linha diferente da de origem. Por último selecionaram-se duas estações que admitem caminhos com custo menor e menor número de estações, mas com maior número de trocas de linha.

Ao iniciar a execução da aplicação, é apresentado na consola um menu que exibe as opções de que o utilizador dispõe, como se mostra na figura 11.

```
------------------------------------------------------------
-------------------------Menu-------------------------------
0 - List All Paths
1 - Select Fastest Path
2 - Select Path with Less Changes
3 - Quit
------------------------------------------------------------
```

Figura 11 – Menu que apresenta ao utilizador as opções disponíveis

Escolheu-se a **opção "0 – List All Paths**" e foram introduzidas duas estações que admitem vários caminhos a ligá-las. Neste caso, escolheram-se Odivelas e Olaias.

```
Please select an option:0
Source?Odivelas
Destination?Olaias
```

Figura 12 – escolha da opção 0 – List All Paths e introdução de um par de estações que permitem testar a solução proposta

Foram listados os quatro caminhos que ligam Odivelas a Olaias. As figuras 13,15,17 e 19 apresentam capturas de ecrã das rotas mostradas ao utilizador. Sinaliza-se no diagrama de rede o caminho correspondente nas figuras 14,16,18 e 20.

```
[Station - nome:Odivelas, distance: 2147483647, nHops: 2147483647];
 [Station - nome:Senhor Roubado, distance: 2147483647, nHops: 2147483647];
 [Station - nome:Ameixoeira, distance: 2147483647, nHops: 2147483647];
 [Station - nome:Lumiar, distance: 2147483647, nHops: 2147483647];
 [Station - nome:Quinta das Conchas, distance: 2147483647, nHops: 2147483647];
 [Station - nome:Campo Grande, distance: 2147483647, nHops: 2147483647];
 [Station - nome:Alvalade, distance: 2147483647, nHops: 2147483647];
 [Station - nome:Roma, distance: 2147483647, nHops: 2147483647];
 [Station - nome:Areeiro, distance: 2147483647, nHops: 2147483647];
 [Station - nome:Alameda, distance: 2147483647, nHops: 2147483647];
 [Station - nome:Olaias, distance: 2147483647, nHops: 2147483647];
```

Figura 13 – captura de ecrã da rota 1 mostrado na consola ao utilizador

Figura 14 – sinalização da rota 1 no diagrama de redes

```
[Station - nome:Odivelas, distance: 2147483647, nHops: 2147483647];
 [Station - nome:Senhor Roubado, distance: 2147483647, nHops: 2147483647];
 [Station - nome:Ameixoeira, distance: 2147483647, nHops: 2147483647];
 [Station - nome:Lumiar, distance: 2147483647, nHops: 2147483647];
 [Station - nome:Quinta das Conchas, distance: 2147483647, nHops: 2147483647];
 [Station - nome:Campo Grande, distance: 2147483647, nHops: 2147483647];
 [Station - nome:Cidade Universitária, distance: 2147483647, nHops: 2147483647];
 [Station - nome:Entre Campos, distance: 2147483647, nHops: 2147483647];
 [Station - nome:Campo Pequeno, distance: 2147483647, nHops: 2147483647];
 [Station - nome:Saldanha, distance: 2147483647, nHops: 2147483647];
 [Station - nome:Picoas, distance: 2147483647, nHops: 2147483647];
 [Station - nome:Marquês de Pombal, distance: 2147483647, nHops: 2147483647];
 [Station - nome:Avenida, distance: 2147483647, nHops: 2147483647];
 [Station - nome:Restauradores, distance: 2147483647, nHops: 2147483647];
 [Station - nome:Baixa Chiado, distance: 2147483647, nHops: 2147483647];
 [Station - nome:Rossio, distance: 2147483647, nHops: 2147483647];
 [Station - nome:Martim Moniz, distance: 2147483647, nHops: 2147483647];
 [Station - nome:Intendente, distance: 2147483647, nHops: 2147483647];
 [Station - nome:Anjos, distance: 2147483647, nHops: 2147483647];
 [Station - nome:Arroios, distance: 2147483647, nHops: 2147483647];
 [Station - nome:Alameda, distance: 2147483647, nHops: 2147483647];
 [Station - nome:Olaias, distance: 2147483647, nHops: 2147483647];
```

Figura 15 – captura de ecrã da rota 2, conforme mostrada ao utilizador na consola

Figura 16 – sinalização da rota 2 no diagrama de rede

```
[Station - nome:Odivelas, distance: 2147483647, nHops: 2147483647];
 [Station - nome:Senhor Roubado, distance: 2147483647, nHops: 2147483647];
 [Station - nome:Ameixoeira, distance: 2147483647, nHops: 2147483647];
 [Station - nome:Lumiar, distance: 2147483647, nHops: 2147483647];
 [Station - nome:Quinta das Conchas, distance: 2147483647, nHops: 2147483647];
 [Station - nome:Campo Grande, distance: 2147483647, nHops: 2147483647];
 [Station - nome:Cidade Universitária, distance: 2147483647, nHops: 2147483647];
 [Station - nome:Entre Campos, distance: 2147483647, nHops: 2147483647];
 [Station - nome:Campo Pequeno, distance: 2147483647, nHops: 2147483647];
 [Station - nome:Saldanha, distance: 2147483647, nHops: 2147483647];
 [Station - nome:São Sebastião, distance: 2147483647, nHops: 2147483647];
 [Station - nome:Parque, distance: 2147483647, nHops: 2147483647];
 [Station - nome:Marquês de Pombal, distance: 2147483647, nHops: 2147483647];
 [Station - nome:Avenida, distance: 2147483647, nHops: 2147483647];
 [Station - nome:Restauradores, distance: 2147483647, nHops: 2147483647];
 [Station - nome:Baixa Chiado, distance: 2147483647, nHops: 2147483647];
 [Station - nome:Rossio, distance: 2147483647, nHops: 2147483647];
 [Station - nome:Martim Moniz, distance: 2147483647, nHops: 2147483647];
 [Station - nome:Intendente, distance: 2147483647, nHops: 2147483647];
 [Station - nome:Anjos, distance: 2147483647, nHops: 2147483647];
 [Station - nome:Arroios, distance: 2147483647, nHops: 2147483647];
 [Station - nome:Alameda, distance: 2147483647, nHops: 2147483647];
 [Station - nome:Olaias, distance: 2147483647, nHops: 2147483647];
```

Figura 17 – captura de ecrã da rota 3, conforme mostrada ao utilizador na consola

Figura 18 – sinalização da rota 3 no diagrama de rede

```
[Station - nome:Odivelas, distance: 2147483647, nHops: 2147483647];
 [Station - nome:Senhor Roubado, distance: 2147483647, nHops: 2147483647];
 [Station - nome:Ameixoeira, distance: 2147483647, nHops: 2147483647];
 [Station - nome:Lumiar, distance: 2147483647, nHops: 2147483647];
 [Station - nome:Quinta das Conchas, distance: 2147483647, nHops: 2147483647];
 [Station - nome:Campo Grande, distance: 2147483647, nHops: 2147483647];
 [Station - nome:Cidade Universitária, distance: 2147483647, nHops: 2147483647];
 [Station - nome:Entre Campos, distance: 2147483647, nHops: 2147483647];
 [Station - nome:Campo Pequeno, distance: 2147483647, nHops: 2147483647];
 [Station - nome:Saldanha, distance: 2147483647, nHops: 2147483647];
 [Station - nome:Alameda, distance: 2147483647, nHops: 2147483647];
 [Station - nome:Olaias, distance: 2147483647, nHops: 2147483647];
```

Figura 19 – captura de ecrã da rota 4 conforme mostrada ao utilizador em consola

Figura 20 – sinalização da rota 4 no diagrama de rede

Para testar a **opção um** foram escolhidas as estações de Parque e Saldanha, pois a estação Parque encontra-se a um nó de distância de duas estações de troca, portanto espera-se que tenha em conta no cálculo do caminho mais rápido os pesos associados à troca de linha. As figuras 21 a 23 apresentam capturas de ecrã relativas à escolha da opção, o caminho mostrado ao utilizador e a sinalização desse caminho no diagrama de rede do metro.

```
Please select an option:1
Source?Parque
Destination?Saldanha
```

Figura 21 – seleção da opção 1

```
[Station - nome:Parque, distance: 295, nHops: 2147483647];
[Station - nome:São Sebastião, distance: 355, nHops: 2147483647];
[Station - nome:Saldanha, distance: 415, nHops: 2147483647];
```

Figura 22 – captura de ecrã do caminho mais curto entre as estações de teste para a opção 1 do menu

Figura 23 – sinalização do caminho mais curto no diagrama de rede

Apurou-se que, seja E o numero de arestas e V o número de vértices, o algoritmo de Djikstra tal como implementado no âmbito deste trabalho garante tempo de execução na ordem de $\boldsymbol{O((E+V)\log V)}$.

Para **testar a opção 2** foram escolhidas as estações de Reboleira e Campo Grande, uma vez que há uma rota, que é na verdade mais curta (via linha vermelha e amarela) em relação à rota com menos trocas. A figura 24 apresenta uma captura de ecrã da seleção da opção 2 com as estações referidas.

```
Please select an option:2
Source?Reboleira
Destination?Campo Grande
```

Figura 24 – seleção da opção 2 com escolha de estações

A figura 25 apresenta uma captura de ecrã do caminho com menos trocas entre Reboleira e Campo Grande. O caminho obtido é aquele que se assinalou no diagrama da rede de metro e é apresentado na figura 26. O caminho com menos trocas apresenta uma troca, enquanto existe um caminho alternativo que envolve duas trocas, via troca em São Sebastião e Saldanha, apresentando portanto duas trocas.

```
[Station - nome:Reboleira, distance: 2147483647, nHops: 0];
[Station - nome:Amadora Este, distance: 2147483647, nHops: 0];
[Station - nome:Alfornelos, distance: 2147483647, nHops: 0];
[Station - nome:Pontinha, distance: 2147483647, nHops: 0];
[Station - nome:Carnide, distance: 2147483647, nHops: 0];
[Station - nome:Colégio Militar/Luz, distance: 2147483647, nHops: 0];
[Station - nome:Alto dos Moinhos, distance: 2147483647, nHops: 0];
[Station - nome:Laranjeiras, distance: 2147483647, nHops: 0];
[Station - nome:Jardim Zoológico, distance: 2147483647, nHops: 0];
[Station - nome:Praça de Espanha, distance: 2147483647, nHops: 0];
[Station - nome:São Sebastião, distance: 2147483647, nHops: 0];
[Station - nome:Parque, distance: 2147483647, nHops: 0];
[Station - nome:Marquês de Pombal, distance: 2147483647, nHops: 0];
[Station - nome:Avenida, distance: 2147483647, nHops: 0];
[Station - nome:Restauradores, distance: 2147483647, nHops: 0];
[Station - nome:Baixa Chiado, distance: 2147483647, nHops: 0];
[Station - nome:Rossio, distance: 2147483647, nHops: 1];
[Station - nome:Martim Moniz, distance: 2147483647, nHops: 1];
[Station - nome:Intendente, distance: 2147483647, nHops: 1];
[Station - nome:Anjos, distance: 2147483647, nHops: 1];
[Station - nome:Arroios, distance: 2147483647, nHops: 1];
[Station - nome:Alameda, distance: 2147483647, nHops: 1];
[Station - nome:Areeiro, distance: 2147483647, nHops: 1];
[Station - nome:Roma, distance: 2147483647, nHops: 1];
[Station - nome:Alvalade, distance: 2147483647, nHops: 1];
[Station - nome:Campo Grande, distance: 2147483647, nHops: 1];
```

Figura 25 – captura de ecrã do caminho com menos trocas conforme mostrado ao utilizador na consola

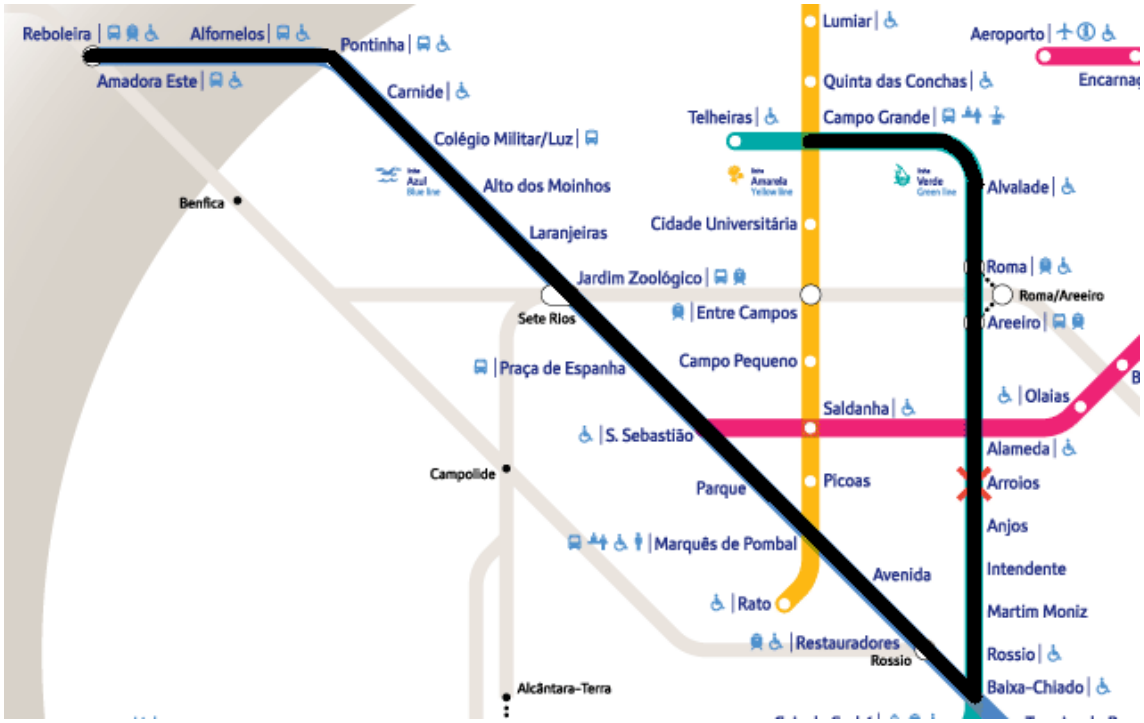

Figura 26 – sinalização no diagrama de rede do caminho com menos trocas obtido

Foi efetuado o estudo da quantidade da memória ocupada pelo grafo. Apesar de não serem constantes os tamanhos em memória das variáveis envolvidas, foi possível apurar de modo relativo a quantidade de memória ocupada por cada tipo de dados. O facto de as estruturas serem conterem auto-referência e não serem usados ponteiros torna difícil a estimativa da quantidade de memória necessária para armazenar o grafo estudado.

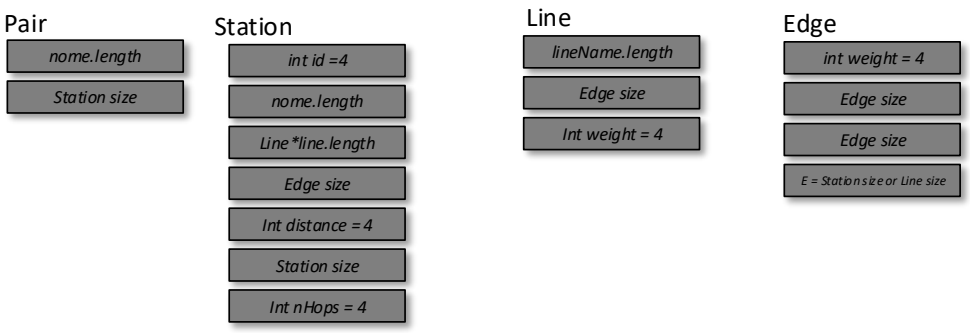

Figura 27 – diagrama dos blocos de memória em bytes necessários para armazenar os tipos de dados usados no contexto do trabalho

Considerou-se que este trabalho prático permitiu compreender e integrar conceitos relativamente novos que foram introduzidos nas aulas, como sejam o tópico dos grafos, a sua representação em memória e algoritmos de pesquisa associados a esta estruturas de dados, designadamente a pesquisa em profundidade e o algoritmo de Djikstra. Permitiu também abordar ordens de peso diferentes em grafos não orientados, e integrar essa informação num cálculo de forma transparente sobre o mesmo grafo. Para além disso, exigiu uma distribuição compacta da informação relativa à linha de metro. Permitiu, finalmente, a adaptação do mesmo esquema algorítmico, baseado no algoritmo de Dijkstra – o calculo de uma lista de menor custo – a dois problemas distintos: o caminho mais rápido e aquele com menor número de troca de linhas.

No global, apesar de introdutório à temática dos grafos, considera-se um exercício pleno de desafio e concluído com sucesso. Destacamos em particular, o cálculo do nó menor distância e separação na execução do algoritmo de Dijkstra entre dois conjuntos, o dos vértices tratados e não tratados, que requereu de entre todos os tópicos uma investigação mais aprofundada e tornou significativos e perenes os exercícios feitos nas aulas.

5 Desenvolvimentos futuros

Propõem-se os seguintes desenvolvimentos futuros à aplicação desenvolvida, por forma a dar continuidade ao estudo do planeamento de viagens em linhas de metro:

1. Tornar genérica a implementação do algoritmo do caminho mais curto entre dois vértices, passando-lhe como parâmetro diferentes comparadores de Station;
2. Dado um ficheiro com as frequências de passagem das carruagens nos diferentes horários do dia, planear uma viagem a partir da hora selecionada pelo utilizador, tendo em conta os tempos de espera, de trânsito e de troca de linha;
3. Carregar em memória a informação de redes de metro mais densas, como Nova Iorque, Paris ou Tóquio [4] usando GTFS [5].

6 Referências bibliográficas

[1] Cormen, T., Leiserson, C., Rivest, R. and Stein, C. (2009). Introduction to algorithms, 3rd ed. Cambridge,MA, U.S.A.: MIT Press, pp.589- 597,649,658-659.

[2] http://www.baeldung.com/java-dijkstra – consultado pela última vez em 13/06/2018.

[3] https://stackoverflow.com/questions/58306/graph-algorithm-to-find-all-connections-between-two-arbitrary-vertices - consultado pela última vez em 13/06/2018

[4] https://www.citylab.com/transportation/2016/02/most-complex-transit-subway-maps-world-tokyo-new-york-paris/470565/ - consultado pela última vez em 13/06/2018

[5] http://datamine.mta.info/ - consultado pela última vez em 13/06/2018