



FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Transporte de Mercadorias

Concepção e Análise de Algoritmos

Turma 6, Grupo E

Jorge Miguel Rodrigues Ferreira (up201207133)

up201207133@fe.up.pt

Nuno Filipe Sousa e Silva (up201404380)

up201404380@fe.up.pt

Pedro Daniel Viana Lima (up201403381)

up201403381@fe.up.pt

Índice

- [1. Resumo](#)
- [2. Palavras-chave](#)
- [3. Descrição do contexto](#)
 - [3.1 Formalização](#)
 - [3.1.1 Breve Explicação](#)
 - [3.1.2 Mapa](#)
 - [3.1.2 Dados de entrada/saída](#)
 - [3.1.3 Limitações](#)
- [4. Descrição da solução](#)
 - [4.1 Explicação do Algoritmo de Floyd-Warshall](#)
 - [4.1.1 Pseudocódigo](#)
 - [4.1.2 Análise de Complexidade do algoritmo](#)
- [5. Lista de casos de utilização](#)
- [6. Reflexões sobre o trabalho](#)
- [7. Medição do Esforço](#)
- [Referências](#)

1. Resumo

Este relatório foi feito no âmbito da unidade curricular “Concepção e Análise de Algoritmos” e centra-se no estudo do desenvolvimento de um sistema de transporte de mercadorias recorrendo para isso a teoria de grafos. Neste sistema espera-se que seja possível obter a melhor opção de percurso para a entrega das mercadorias.

2. Palavras-chave

- ☐ Grafo
- ☐ Densidade
- ☐ Breadth-first search
- ☐ Algoritmo de Floyd-Warshall
- ☐ Algoritmo de Dijkstra

3. Descrição do contexto

Cada vez mais em transito, se verifica a presença de carrinhas de entregas, isto advém, maioritariamente, do grande aumento progressivo de compras online que tem ocorrido nos últimos anos. Isso exige um melhor planeamento do percurso de entregas por parte das empresas de transporte de mercadorias, de modo a entregar o maior número de encomendas no menor percurso possível, minimizando assim o custo das entregas.

3.1 Formalização

Com este projeto, pretendia-se implementar um sistema que permita à empresa gerir as suas entregas.

Assim, para desenvolver a plataforma foi necessário criar uma estrutura de dados que contivesse todos os destinos possíveis, designada por grafo, em que cada destino é referenciado por um identificador. Além disso, foi necessário criar uma classe City que contém essa mesma estrutura de dados, uma companhia de distribuições, e um nome.

3.1.1 Breve Explicação

Uma empresa de distribuição de mercadorias possui uma frota de veículos que podem variar na capacidade e no tipo, sendo que esses camiões recolhem, diariamente, itens de um depósito (uma zona de partida comum) e levam-nos até aos seus respectivos destinatários. Terminadas as entregas, os camiões são recolhidos à garagem, num sítio distinto do depósito.

Os itens possuem vários atributos, contudo, os mais importantes são:

- ❖ volume
- ❖ destinatário

3.1.2 Mapa

Quanto ao mapa utilizado no projeto, consideramos aquilo que o professor referiu nas aulas e optamos por utilizar mapas reais juntamente com coordenadas geográficas para as localizações do depósito, garagem e destinos.

Para isso, recorremos ao OSM2TXT Parser disponível no moodle para converter o ficheiro exportado no OpenStreetMaps para ficheiro de texto. O mapa utilizado por nós corresponde a uma pequena parte de Penafiel, que achamos que seria de todo indicado para o projeto já que apresenta um grande número de cruzamentos e ao mesmo tempo é uma zona simples.

Os ficheiros gerados são três (ligeiramente modificados em relação aos originais), em que um primeiro ficheiro contém o identificador do nó/vértice, a latitude e longitude ambas em graus, o segundo contém o identificador da aresta, o nome e a informação de que se é orientada nos dois sentidos. E por último, um ficheiro que contém um par de nós em que a aresta de ligação entre eles é identificada pelo seu id. Dito isto, foi necessário introduzir um identificador, que não é nada mais nada menos que um inteiro, em cada aresta e vértice no código do grafo desenvolvido nas aulas práticas.

Como nos ficheiros convertidos uma aresta possui vários vértices, tornou-se necessário subdividir, nos ficheiros, em várias outras arestas que ligam os vértices que existiam na aresta inicial. Para isso foi necessário alterarmos os ids de cada aresta, para que, cada aresta que liga dois vértices ter um identificador único.

3.1.2 Dados de entrada/saída

Os dados de entrada esperados do utilizador são:

- **D**: Localização do depósito.
- **E**: Local de estacionamento dos camiões.
- **ID**: Destino de cada um dos itens.
- **G(D,E,ID)**

Os dados de saída são os seguintes:

- **V(S, D)**: Conjunto de vértices do caminho ótimo entre o **D** e **E** passando por todos os destinos das entregas(se for possível).

A informação de saída é apresentada ao utilizador de forma gráfica através do GraphViewer, onde o caminho mais curto apresenta-se colorido.

3.1.3 Limitações

O programa desenvolvido apresenta algumas limitações, podendo enumerar as seguintes:

- Nos ficheiros resultantes do Parser, mais propriamente, no ficheiro dois e três, havia a possibilidade de uma aresta fazer a ligação entre mais do que dois nós. Então, de forma a arranjar solução, colocamos um índice diferente para cada aresta, além disso, o ficheiro dois passa a ter igual numero de linhas que o terceiro ficheiro ficando o carregamento de ficheiros um pouco mais pesado.
- O mapa utilizado corresponde apenas à zona de Penafiel.
- Existem vértices que ficam para lá da extensão do mapa no GraphViewer que advém da leitura dos ficheiros.

4. Descrição da solução

Criou-se uma aplicação com estruturas de dados que representam entidades como: cidade, companhia, item e caminhão.

A base de dados encontra-se na classe cidade que contém um nome representativo da cidade, um *graph* que representa o mapa de uma zona da cidade e uma companhia responsável pelas entregas nessa zona da cidade. A companhia possui um vector com todos os itens que estão a aguardar a distribuição e um vector com todos os camiões que a empresa possui.

A aplicação implementa o algoritmo de *Floyd-Warshall* que calcula o caminho mais curto entre dois vértices, ou seja, é ideal para a situação da transportadora, visto que a carrinha parte do depósito e irá passar por vários pontos de entrega distintos, o que significa que temos de calcular o caminho mais curto para cada sequência de destinos utilizando para isso a função *permutation* da biblioteca *algorithm*. A complexidade do algoritmo de *Floyd-Warshall* é de $O(|Vértices|^3)$ como iremos verificar adiante.

Outra hipótese a este algoritmo, seria utilizar o algoritmo de Dijkstra, no entanto, o nosso objetivo era calcular caminho mais curto entre todos os pares de vértices e dado que o Dijkstra é idealmente um algoritmo para calcular o caminho entre um vértice e todos os outros, teríamos que proceder a uma execução repetida do mesmo, além disso, segundo os slides teóricos o de Floyd-Warshall é melhor para uma situação específica, ou seja para grafos densos que no nosso caso confirma-se.

Algo que também é verificado é a conectividade entre o depósito - todos os itens para entrega e entre depósito - garagem logo após o utilizador inserir os destinos pela consola, através de uma pesquisa em largura (*Breadth-first search*) a partir do vértice correspondente ao depósito.

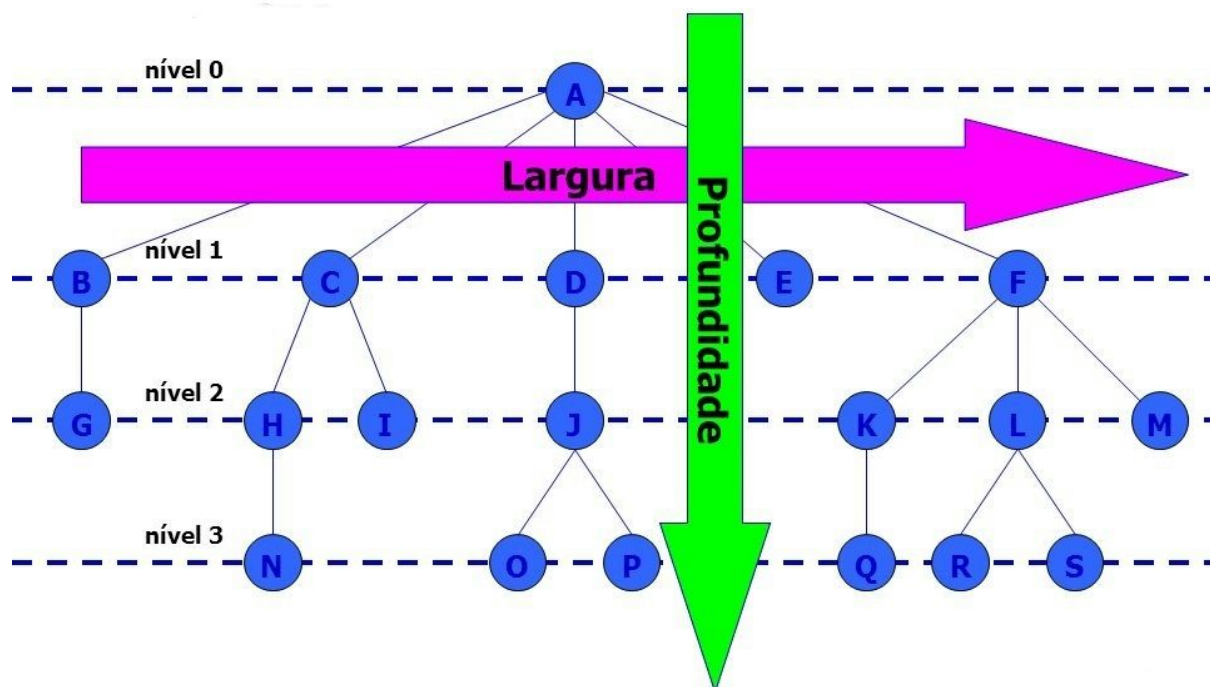


Fig 1 - Funcionamento do Breadth-first search a partir de A.

Caso não haja caminho possível a partir do depósito é mostrada uma mensagem de erro e o programa termina.

Por fim, aplicamos o **Algoritmo de detecção de pontos de articulação**, sugestão da professora, para o utilizador ter a percepção de todos os pontos de articulação presentes no mapa.

Pseudocódigo

//Procura Pontos de Articulação usando pesquisas em profundidade

// Contador global e inicializado a 1

```
void findArt(Vertex v) {
    v.visited = true;
    v.low = v.num = counter++;
    for each w adjacent to v
        if( !w.visited ) {           //ramo da árvore
            w.parent = v;
            findArt(w);
            v.low = min(v.low, w.low);
            if(w.low >= v.num)
                //mensagem a indicar que é um ponto de articulação
        }
    else
        if( v.parent != w )         //aresta de retorno
            v.low = min(v.low, w.num);
}
```

4.1 Explicação do Algoritmo de Floyd-Warshall

O algoritmo preenche uma matriz bidimensional, $W[i][j]$, onde $W[i][j]$ representa o tamanho do percurso mais curto entre os vértices i e j . Além disso, assume-se que esta matriz está inicialmente preenchida da seguinte forma:

$W[i][j] = 0$, se $i = j$;

$W[i][j] = \text{INT_INFINITY}$, se não há ligação direta entre i e j ;

$W[i][j] = \text{peso da aresta entre } i \text{ e } j$, se houver ligação direta entre i e j .

Começamos por fixar um vértice k do grafo para cada par (i,j) de vértices, então, verificamos se o menor caminho já conhecido entre (i,j) supera a

soma do tamanho de i para k com o de k para j. Caso supere, o tamanho do menor caminho passa a ser essa soma.

4.1.1 Pseudocódigo

```
func FloydWarshall(W[][])
for k = 1 to n
  for i = 1 to n
    for j = 1 to n
      W[i][j] = min( W[i][j], caminho [i] [k] +
caminho[k][j])
```

4.1.2 Análise de Complexidade do algoritmo

Cada ciclo for (linhas 2 a 4) pode ser convertido num somatório com o mesmo valor inicial e final. Considerando a comparação (decisão do valor mínimo entre dois números) como operação elementar e as atribuições e acessos a matrizes como tempo constante, temos que a complexidade do algoritmo de Floyd-Warshall é:

$$\sum_{k=1}^n \sum_{i=1}^n \sum_{j=1}^n 1$$

Pode-se converter o somatório da variável j em n, pois estamos a somar n vezes o valor 1, e isolar este termo que não depende de nenhuma forma da variável j, resultando no seguinte somatório:

$$n \sum_{k=1}^n \sum_{i=1}^n 1$$

Repetindo o mesmo processo de dedução usado em 1, temos como resultado:

$$n \sum_{k=1}^n n$$

$$n^2 \sum_{k=1}^n 1$$

$$n^3$$

Assim, podemos afirmar com certeza que o algoritmo de Floyd-Warshall possui complexidade $\mathbf{O}(n^3)$, onde n é o numero de vértices do grafo.

(ficheiro .eap fornecido)



5. Lista de casos de utilização

O nosso projeto possui uma interface gráfica em que a interação com o utilizador é feita de duas maneiras distintas: primeiramente através da linha de comandos e mais tarde utilizando o GraphViewer. Para facilitar a demonstração foi utilizado apenas a entrega de um item. Começemos

- Inicialmente aparece duas opções ao utilizador como é ilustrado na seguinte imagem.

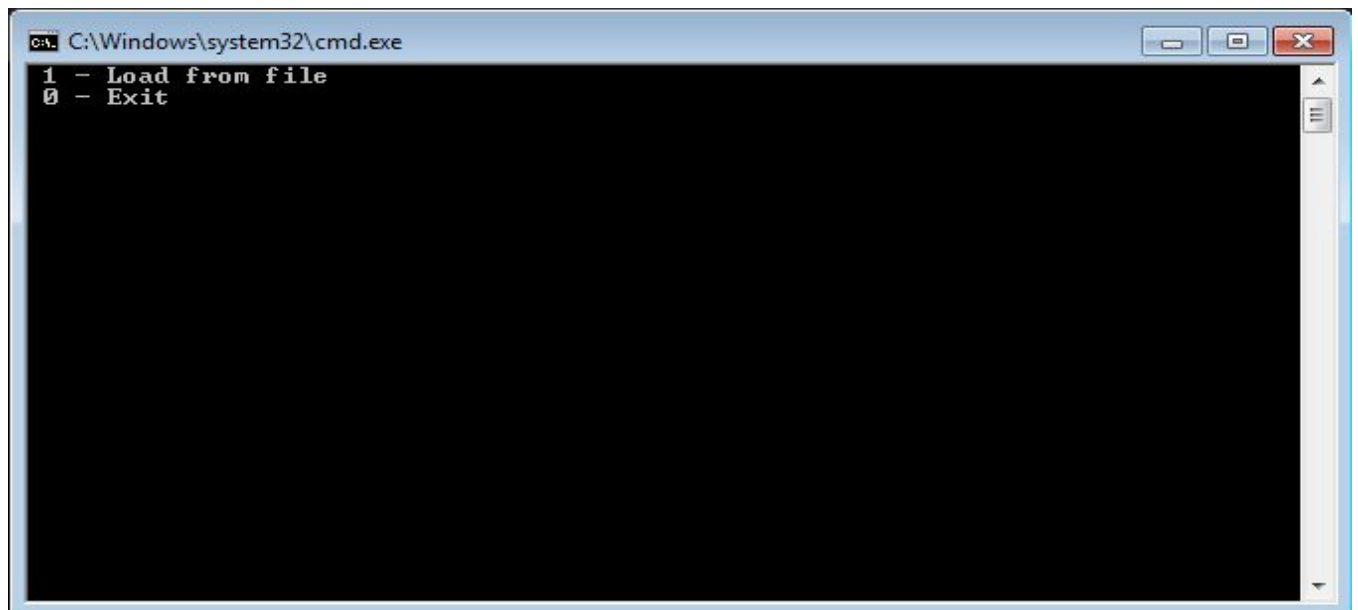


Fig 2 - Primeira janela de contato com o utilizador.

Tal como o nome indica “Load from File” dá acesso a um outro menu para efetuar o devido carregamento dos ficheiros em que o nome deles é inserido manualmente pelo utilizador na consola.

Nota: O carregamento dos ficheiros do mapa é um pouco demorado (~8 segundos), isso deve-se ao facto de termos utilizado um mapa um pouco extenso que possui um elevado numero de vértices aliado também à mudança que fizemos no segundo ficheiro.

São necessários cinco ficheiros. Um de itens, outro de camiões e, por último, três ficheiros do mapa. Estes últimos ficheiros devem ter o mesmo nome diferindo

apenas na sua terminação. Por exemplo, o ficheiro dos vértices poderá ser representado por map1.txt, o ficheiro das arestas por map2.txt e o ficheiro que indica que vértices ligam as arestas por map3.txt.

Após o carregamento dos ficheiros, é pedido ao utilizador que adicione o destino que deseja para cada um dos itens lidos, para a garagem e para o depósito. **(Se colocar destino do depósito igual ao destino da garagem o utilizador é obrigado a inserir de novo um destino para ambos ou apenas para um).** É feita uma amostra prévia na consola de todos os vértices que estão disponíveis.

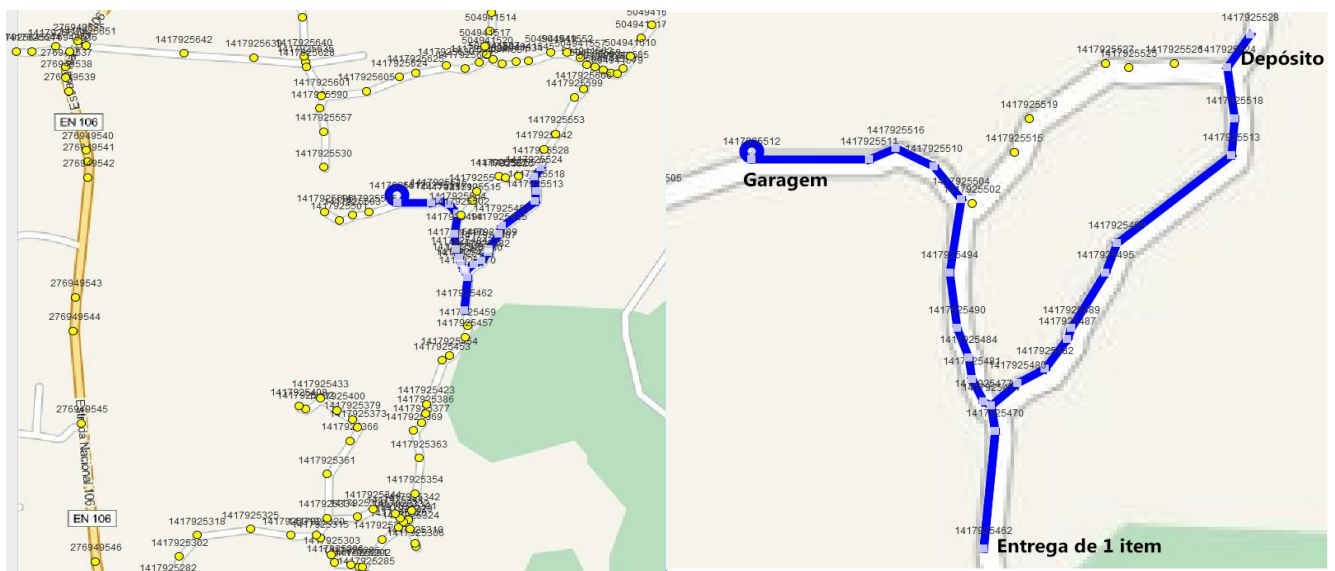


Fig 3 - Entrega de um item pelo caminho mais curto.

Na entrega de um item, é escolhido sempre o caminho mais curto. O caminho da direita representado na imagem de cima é efetivamente mais curto em relação ao da esquerda. O camião sai do depósito e entrega o item. No fim da entrega, regressa à garagem.

É também perguntado ao utilizador se pretende simular alguma estrada em obras. Esta simulação traduz-se na eliminação de arestas numa dada via impossibilitando então o camião de aí transitar.



Fig 4 - Estrada cortada.

Nas vias por onde o caminhão passa podem existir estradas cortadas devido a obras. Nestes casos é calculado o percurso alternativo de menor caminho.

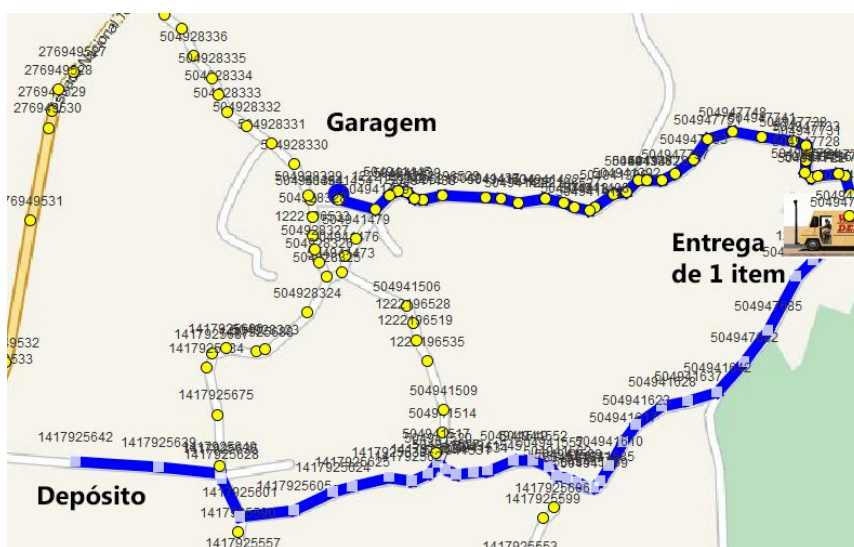
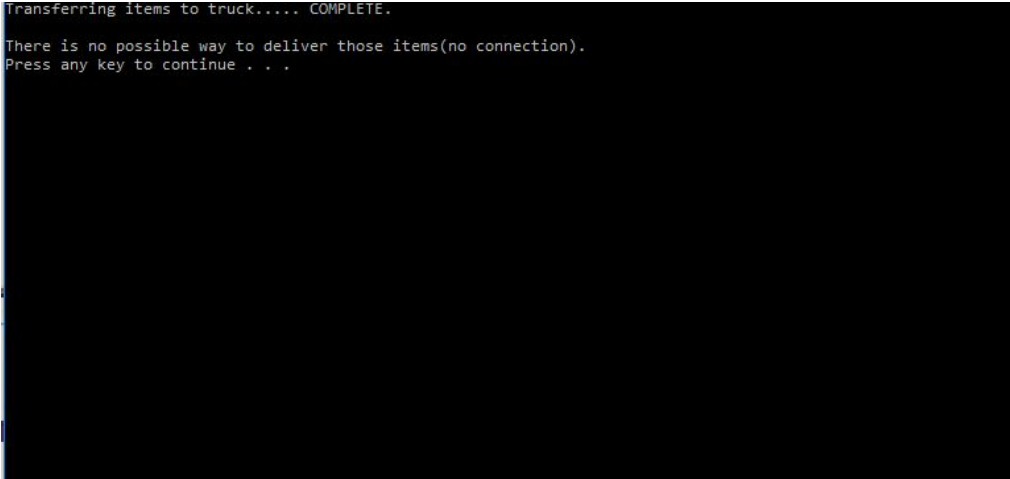


Fig 5 - Entrega de um item numa estrada de sentido único.

Existem também casos em que só existem estradas de um sentido e, estando a estrada em obras, torna-se impossível fazer a entrega.

No caso acima apresentado, a estrada percorrida do depósito à garagem é de apenas um sentido. Caso a estrada seja cortada, e não seja possível fazer a entrega, uma mensagem é apresentada na consola.

A screenshot of a terminal window with a black background and white text. The text displayed is: "Transferring items to truck..... COMPLETE." followed by "There is no possible way to deliver those items(no connection)." and "Press any key to continue . . .".

```
Transferring items to truck..... COMPLETE.  
There is no possible way to deliver those items(no connection).  
Press any key to continue . . .
```

Fig 6 - Impossibilidade de entrega.

6. Reflexões sobre o trabalho

À medida que íamos desenvolvendo o projeto deparamo-nos com algumas adversidades, uma pequena parte estavam relacionadas com a leitura dos ficheiros resultantes da conversão para formato .txt, visto que, para o terceiro ficheiro, uma mesma aresta(*id*) fazia a ligação entre vários vértices, o que resultava em erros na leitura para o grafo.

Nos mapas, muitas estradas não possuem conexão devido a erros nos ficheiros produzidos pelo OSM2TXT Parser. No terceiro ficheiro gerado pelo OSM2TXT Parser existiam estas linhas:

```
18;504928331;2429913962;
```

```
19;2429913962;504928332;
```

Estes ID's de vértices aparentemente levam a que haja conexão nessa mesma estrada, mas procurando no ficheiro um, verifica-se que o vértice 2429913962 não existe. Este vértice não é então adicionado ao vetor de vértices. Nunca vai ser portanto, possível adicionar uma aresta para este vértice ou a partir de este, sendo que a estrada real com boa conexão, fica sem conexão no graph.

Outra das dificuldades encontradas estava intrinsecamente relacionada com a interpretação de certas partes do enunciado do trabalho pois este não era muito claro.

7. Medição do Esforço

Todos contribuímos de igual forma para o desenvolvimento do projeto e do relatório, considerando assim uma avaliação justa, $\frac{1}{3}$ para cada um dos elementos do grupo.

Referências

[1] Daudt, César; Miranda, Caio – “Análise da Complexidade do algoritmo de Floyd-Warshall”, Universidade Federal de Rio Grande de Sul. 13 de Outubro de 2010.