

**Transporte de Mercadorias**

Concepção e Análise de Algoritmos

**Turma 6, Grupo E**

**Jorge Miguel Rodrigues Ferreira** (up201207133)

[up201207133@fe.up.pt](mailto:up201404022@fe.up.pt)

**Nuno Filipe Sousa e Silva** (up201404380)

[up201404380@fe.up.pt](mailto:up201404380@fe.up.pt)

**Pedro Daniel Viana Lima** (up201403381)

[up201403381@fe.up.pt](mailto:up201403381@fe.up.pt)

**Índice**

[1. Resumo](#_m1en2luufcq)

[2. Palavras-chave](#_dp66k3shzs9x)

[3. Descrição do contexto](#_2t0hlez2bhpz)

[3.1 Formalização](#_cv371qiutguh)

[3.1.1 Breve Explicação](#_t3unk73hx1yu)

[3.1.2 Mapa](#_8yccfb1sqtdu)

[3.1.2 Dados de entrada/saída](#_cjn4zn5eiucs)

[3.1.3 Limitações](#_pepktqg931pi)

[4. Descrição da solução](#_rj75npq6hiyz)

[4.1 Explicação do Algoritmo de Floyd-Warshall](#_n4ptrn76zjn8)

[4.1.1 Pseudocódigo](#_t3vh1fr1a2jp)

[4.1.2 Análise de Complexidade do algoritmo](#_hwa3we3rd53v)

[4.2 Algoritmo de Knuth-Morris-Pratt](#_fc4frsimrlgk)

[4.2.1 Pré Processamento do Padrão](#_mathwuvrnbsf)

[4.2.1.1 Pseudo-código](#_igvp9b1u5eqc)

[4.2.2 String Matching KMP](#_6c5ph5ni6p1r)

[4.2.2.1 Pseudo-código](#_txrlg1osoq8a)

[4.2.3 Análise de Complexidade do algoritmo](#_h7m3534i57mt)

[4.2.3.1 Teórica](#_okvw4bjelc1h)

[4.2.3.2 Empírica](#_dd6m3u1oot34)

[4.3 Algoritmo de Pesquisa Aproximada](#_psex8lm5dr64)

[4.4 Diagrama UML](#_qfqwfvbo1gd)

[5. Lista de casos de utilização](#_24mnc9wpu1rn)

[6. Reflexões sobre o trabalho](#_2xp5cixx6x18)

[7. Medição do Esforço](#_rd5hlgth47ff)

[Referências](#_fncdbiro3jep)

# **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
* Algoritmo de Knuth-Morris-Pratt
* Algoritmo de Pesquisa Aproximada

# **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.

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 possuí 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 la da extensão do mapa no GraphViewer que advém da leitura dos ficheiros.
* O algoritmo de Brute Force que utilizamos, sempre que encontra uma palavra no texto idêntica ao padrão, termina, assim,não é possível retornar o numero de vezes que a palavra surgiu no texto.

# **4. Descrição da solução**

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

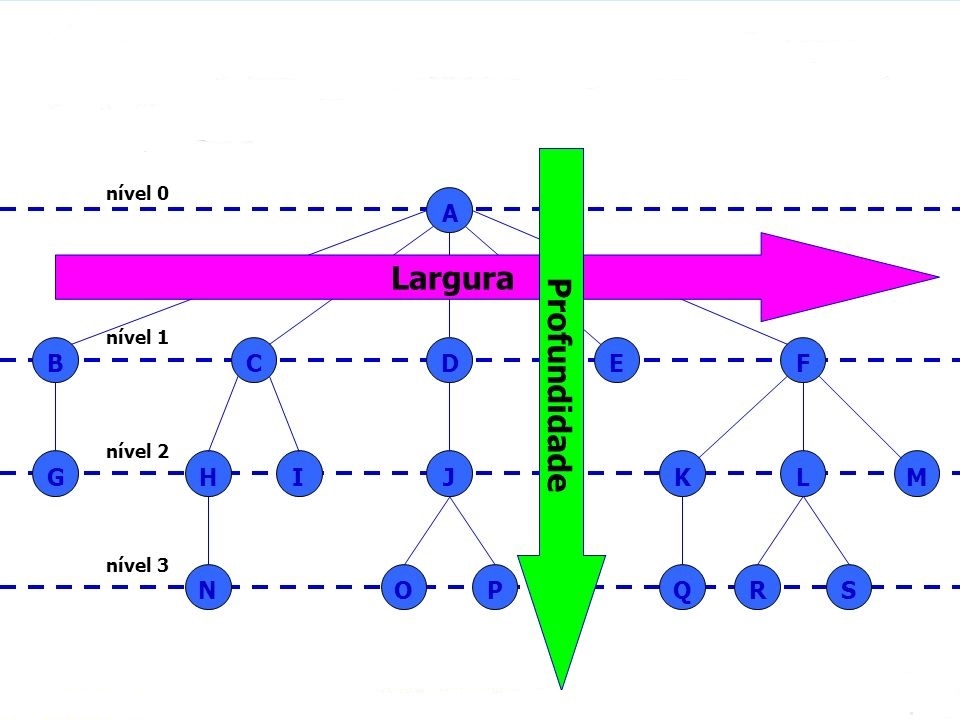
A base de dados encontra-se na classe cidade que contém um nome representativo da cidade, um *graph* que representa o mapa de uma zona da cidade e uma companhia responsável pelas entregas nessa zona da cidade. A companhia possui um vector com todos os items que estão a aguardar a distribuição e um vector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

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 sequencia de destinos utilizando para isso a função permutation da biblioteca algorithm. A complexidade do algoritmo de *Floyd-Warshall* é deO(**|**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-Warshal é melhor para uma situação especifica,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 deposito - 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 deposito.

Na segunda parte do trabalho, implementamos vários algoritmos de pesquisa, comparamos os seus tempos de execução e discutimos a sua pertinência para a resolução de problemas.



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

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

Além disso, 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);

Quanto à segunda parte do projeto, o objetivo era introduzir mais algumas funcionalidades à nossa aplicação, tais como:

* permitir ao utilizador procurar, entre os itens a serem entregues por um camião, um determinado cliente pelo seu nome.
* permitir identificar quantos itens existem endereçados para uma determinada rua, identificada pelo seu nome.

Sendo que a solução para cada uma dessas funcionalidades foi implementada com recurso à pesquisa exata e aproximada em strings.

Em relação à pesquisa exata, servimo-nos do algoritmo de Knuth-Morris-Pratt e um algoritmo exaustivo desenvolvido por nós para poder efetuar comparações quanto à eficiência em termos de tempo e a conclusão a que chegamos é que o algoritmo de Knuth-Morris-Pratt é bem superior. Enquanto que o Knuth-Morris-Pratt no pior caso requer ***O(T + P)***, onde ***T*** corresponde ao numero de caracteres do texto onde se pretende efetuar a pesquisa e ***P***, o número de caracteres do padrão a procurar, o algoritmo exaustivo pode requerer ***O(T \* P)***.

## **4.1 Explicação do Algoritmo de Floyd-Warshall**

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

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

**W[i][j]** = INT\_INFINITY, se não há ligação direta entre i e j;

**W[i][j]** = peso da aresta entre i 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:

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:

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

Assim, podemos afirmar com certeza que o algoritmo de Floyd-Warshall possui complexidade ***O()***, onde n é o numero de vértices do grafo.

## **4.2 Algoritmo de Knuth-Morris-Pratt**

O algoritmo consiste em procurar uma palavra (padrão) ***P*** dentro de um texto ***T***. Quando aparece uma diferença (mismatch), a palavra tem em si a informação necessária para determinar onde começar a próxima comparação, ou seja, um mismatch consiste em caracteres que já conhecemos, porque eles estão na palavra padrão (isso evita retroceder pelos caracteres já conhecidos e efetuar comparações totalmente desnecessárias).

A implementação deste algoritmo está dividido em duas etapas:

* Função prefixo
* String matching

## **4.2.1 Pré Processamento do Padrão**

Para determinar a função prefixo, compara-se o padrão com deslocações do mesmo.

* q = 1, ..., | P |
* P[i..j] - substring entre índices i e j
* índices a começar em 1
* é o comprimento do maior prefixo de P que é sufixo próprio do prefixo de P em comprimento q

### **4.2.1.1 Pseudo-código**

func pre\_kmp(P,)

m = length[P]

k = -1

for q = 1 to m

while(k > -1 and P[k +1] =/ P[q]

k =

if P[k + 1] = P[q]

k= k + 1

return

## **4.2.2 String Matching KMP**

Desloca-se o padrão para a direita de uma forma que permite continuar a comparação na mesma posição do texto! Esse deslocamento foi calculado previamente na etapa de pré-processamento.

### **4.2.2.1 Pseudo-código**

func kmp(T,P)

n = length[T]

m = length[P]

q = 0;

for i = 1 to n

while(q > 0 and P[q +1] =/ T[i]

q =

if P[q + 1] = T[i]

q= q + 1

if q = m

printf “Pattern occurs with shift” i - m

q =

## **4.2.3 Análise de Complexidade do algoritmo**

### **4.2.3.1 Teórica**

String Matcher

* Eficiência depende do número de iterações do ciclo “while” interno.
* Dado que 0 , cada vez que a instrução 7 é executada, o valor de q é decrementado de pelo menos 1, sem nunca chegar a ser negativo.
* Dado que o valor de q começa em 0 e só é incrementado no máximo n vezes (+1 de cada vez, na linha 9), o número máximo de vezes que pode ser decrementado (nas linhas 7 e 12) é também ***n***.
* Numero máximo de iterações do ciclo “while” interno é ***n***.
* Tempo de execução da rotina é ***O(n)***, i.e., ***O(|T|)***.

Prefix-Function

* Tempo de execução é O(n).

**Total:**

O(|T| + |P|)

\* informação retirada dos slides teóricos disponibilizados no moodle.

### **4.2.3.2 Empírica**

INPUTS

**Items-Clientes**

*Item1-* Alfredo Manuel da Silva Ferreira

*Item2-* José Miguel Rodrigues Ferreira

*Item3-* Alfredo Manuel da Silva Ferreira

*Item4-* José Miguel Rodrigues Ferreira

*Item5-* José Miguel Rodrigues Ferreira

*Item6-* Ana Carolina Coelho

*Item7-* Mariana Filipa Santos

*Item8-* Ana Carolina Coelho

*Item9-* Carlos Gomes Oliveira

*Item10-* Carlos Manuel da Costa Marques Oliveira

**Items-Localização**

*Item1-* 504941364

*Item2-* 276949541

*Item3-*  276949540

*Item4*- 1417925635

*Item5-* 276949539

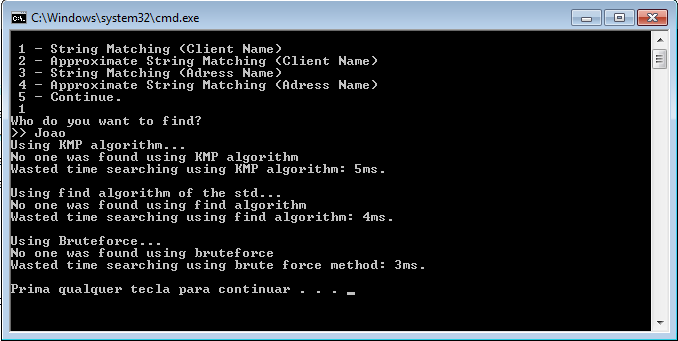
*Item6-* 276949538

*Item7-* 276949537

*Item8-* 276949536

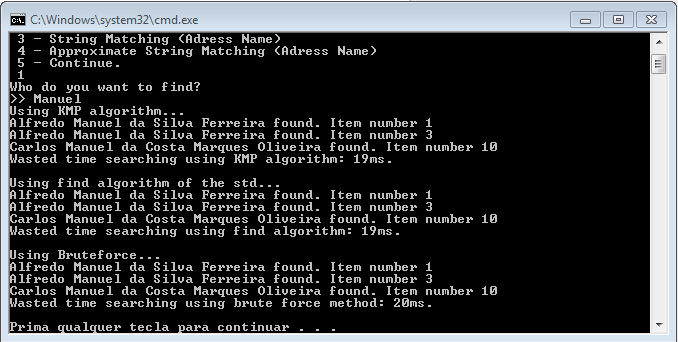
*Item9-* 276949535

*Item10-* 276949533

Testes:

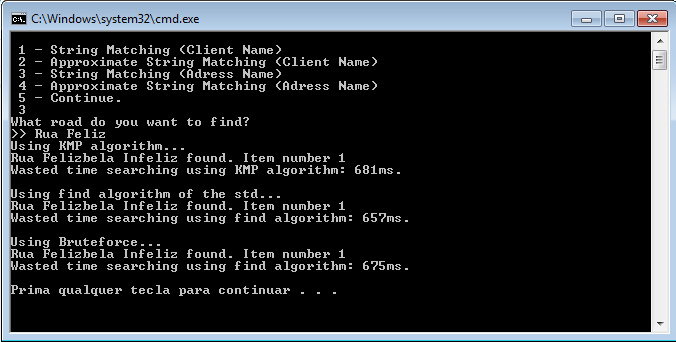
**Fig. 2 -** Pesquisa pelo nome João.

Resultado: Não foi encontrado qualquer cliente com esse nome. Analisando os tempos obtidos é possível observar que o método que demorou menos tempo foi o exaustivo.



**Fig. 3 -** Pesquisa pelo nome Manuel.

Resultado: Retornou todos os nomes da lista de clientes que possuem “Manuel”. Neste caso o algoritmo de Brute Force já apresenta um tempo superior aos restantes.



**Fig. 4 -** Pesquisa dos itens por nome da rua -“Rua Felizbela”. O 1º item faz parte da “Rua Felizbela Infeliz” logo está correto.

**Conclusão:** Pela análise dos tempos, tem-se a percepção de que o algoritmo de Brute Force é o mais eficiente, no entanto, o mesmo não é verdade já que a complexidade temporal é, tal como já referimos umas paginas acima, O(T\*P), ou seja para padrões e textos de grandes dimensões este vai ser bastante inferior aos restantes algoritmos (irá fazer comparações inúteis gastando mais tempo de CPU).

De notar também que o algoritmo KMP pode encontrar uma dada palavra diversas vezes num texto, o que não acontece no find e no bruteforce, logo, para textos grandes com palavras repetidas, o KMP é bastante mais eficiente do que qualquer outro algoritmo que possa ser aplicado para efeitos semelhantes.

## 

## **4.3 Algoritmo de Pesquisa Aproximada**

Usado para detetar strings similares, este algoritmo, de programação dinâmica, é útil para detecção de erros de escrita. O algoritmo verifica o quão similares duas strings são, contando o número mínimo de operações necessárias para transformar uma *string* na outra.

**4.3.1 Distância de edição entre duas strings**

**4.3.1.1 Pseudo Código**

func EditDistance(P,T)

// inicialização

for i = 0 to |P| do D[i,0] = i

for j = 0 to |T| do D[0,j] = j

// recorrência

for i = 1 to |P| do

for j = 1 to |T| do

if P[i] == T[j] then D[i,j] = D[i-1,j-1]

else D[i,j] = 1 + min(D[i-1,j-1],

D[i-1,j],

D[i,j-1])

// finalização

return D[|P|, |T|]

**4.3.2 Analise da Complexidade do Algoritmo**

**4.3.2.1 Teórica**

**Complexidade temporal:**

O(d\*min(p, t))

**d**- distancia máxima de edição

**p** e **t**- tamanho das strings a comparar

**4.3.2.2 Empírica**

**Items-Clientes**

*Item1- Alfredo Manuel da Silva Ferreira*

*Item2- José Miguel Rodrigues Ferreira*

*Item3- Alfredo Manuel da Silva Ferreira*

*Item4- José Miguel Rodrigues Ferreira*

*Item5- José Miguel Rodrigues Ferreira*

*Item6- Ana Carolina Coelho*

*Item7- Mariana Filipa Santos*

*Item8- Ana Carolina Coelho*

*Item9- Carlos Gomes Oliveira*

*Item10- Carlos Manuel da Costa Marques Oliveira*

**Items-Localização**

*deposito- 276949542*

*garagem- 276949541*

*Item1-* 504941364

*Item2-* 276949541

*Item3-*  276949540

*Item4*- 1417925635

*Item5-* 276949539

*Item6-* 276949538

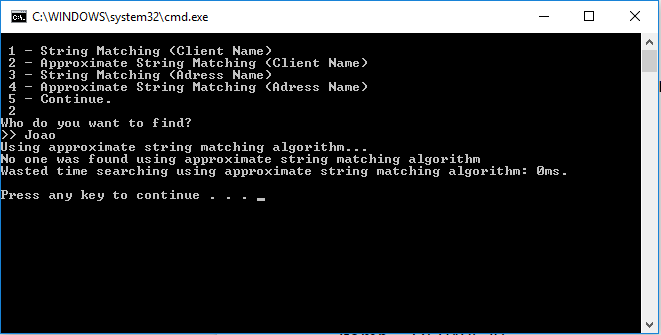
*Item7-* 276949537

*Item8-* 276949536

*Item9-* 276949535

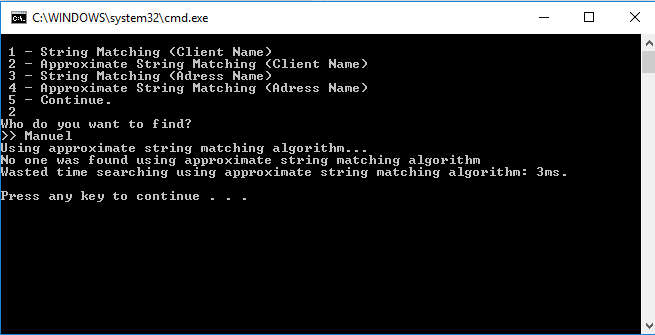
*Item10-* 276949533

Testes:



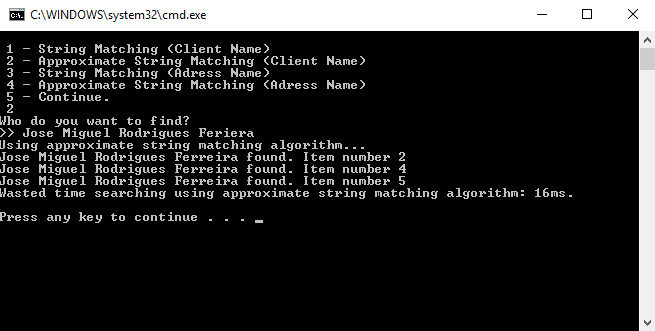
**Fig. 5-** Pesquisa pelo nome “Joao”.

Resultado: Como esperado não foi encontrado qualquer cliente com esse nome com tempos ótimos.

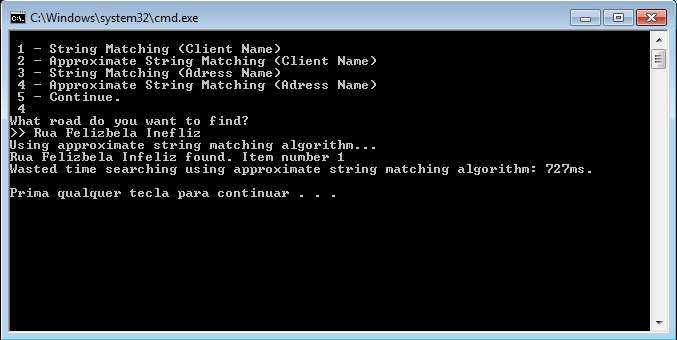


**Fig. 6-** Pesquisa pelo nome “Manuel”.

Resultado: Não foi encontrado qualquer cliente com esse nome porque apesar de existir clientes com o nome “Manuel” na sua constituição, o algoritmo detetou que a diferença era muito grande do nome completo. Devido a essa verificação, a pesquisa demorou mais tempo que a pesquisa de “Joao”.

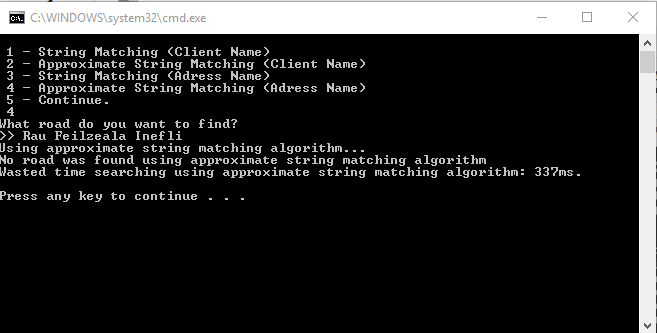


**Fig. 7-** Pesquisa pelo nome “Jose Miguel Rodrigues Feriera”.

Resultado: Como esperado foi encontrado 3 resultados, pois as diferenças de “Jose Miguel Rodrigues Ferreira” são mínimas.

**Fig. 8-** Pesquisa pela rua “Rua Feilzbela Inefli”.

Resultado: Como esperado foi encontrado 2 resultados, pois as diferenças para “Rua Felizbela Infeliz” são mínimas.



**Fig. 9-** Pesquisa pela rua “Rau Feilzeala Inefli”.

Resultado: Neste cado já diz que não encontrou a “Rua Felizbela Infeliz” porque as diferenças são demasiadas. (Nós definimos para 6).

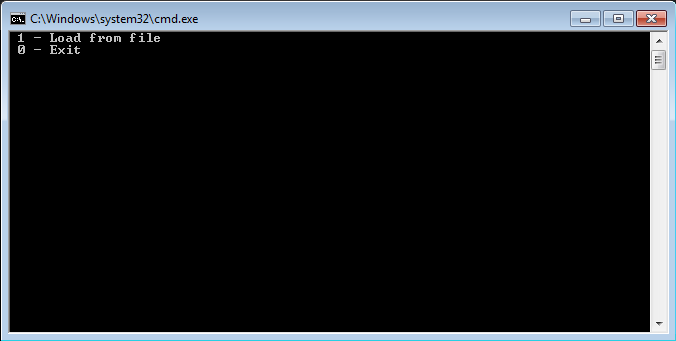
## **4.4 Diagrama UMLUML.png**

(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. Comecemos

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

****

**Fig. 2 -** Primeira janela de contato com o utilizador.

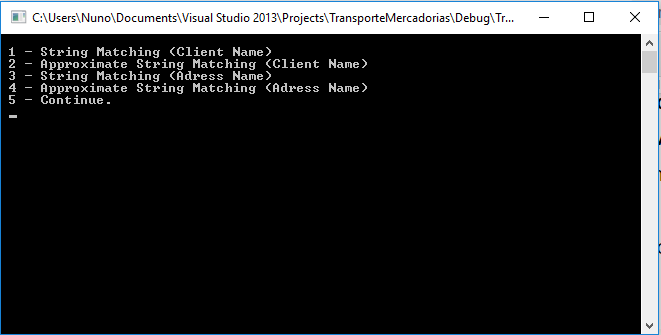
Tal como o nome indica “Load from File” dá acesso a um outro menu para efetuar o devido carregamento dos ficheiros em que o nome deles é inserido manualmente pelo utilizador na consola.

**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.

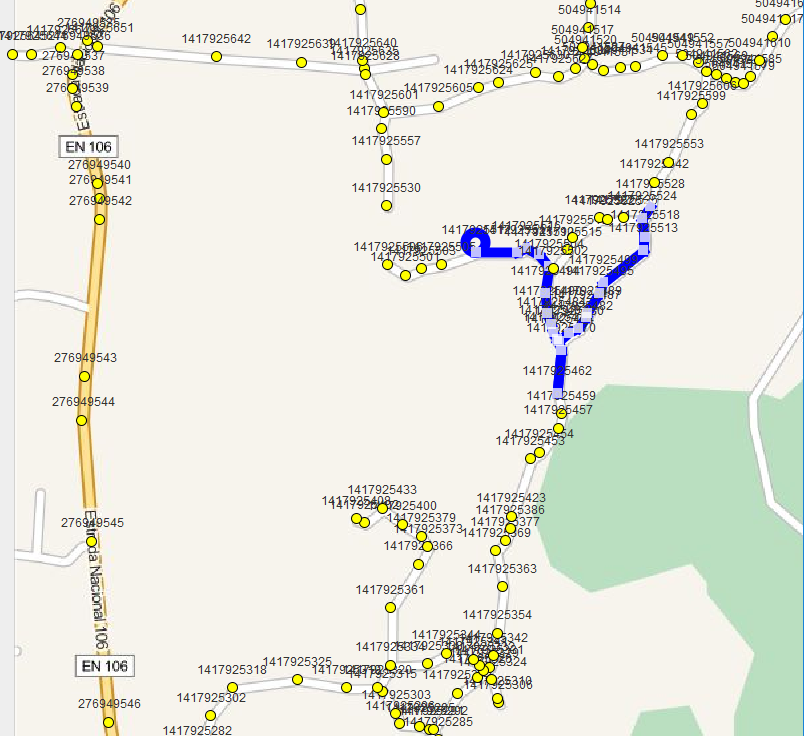
Após o carregamento dos ficheiros, é pedido ao utilizador que adicione o destino que deseja para cada um dos itens lidos, para a garagem e para o deposito. (**Se colocar destino do deposito igual ao destino da garagem o utilizador é obrigado a inserir de novo um destino para ambos ou apenas para um**).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

De seguida, considerando que já selecionou os pontos pretendidos, será apresentado ao utilizador cinco opções.

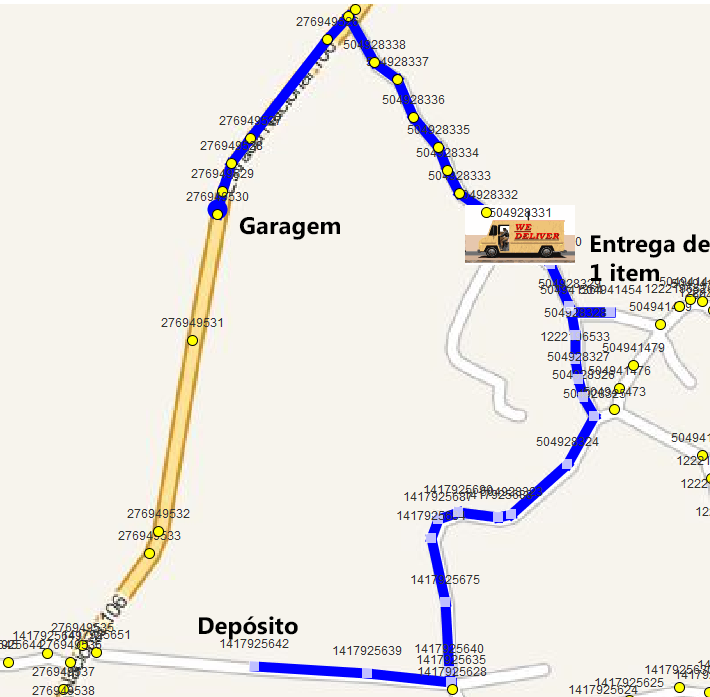
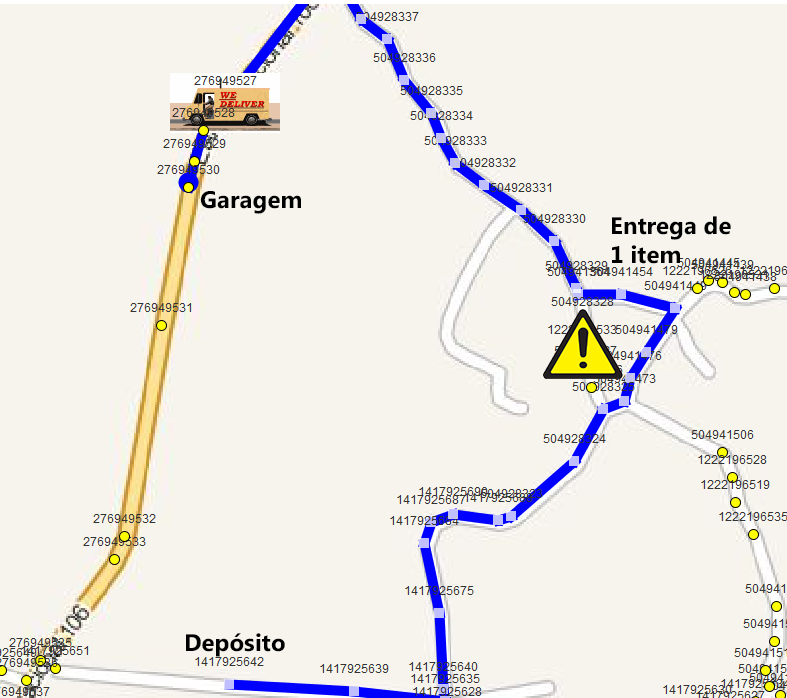


**Fig. 3 -** Opções de pesquisa

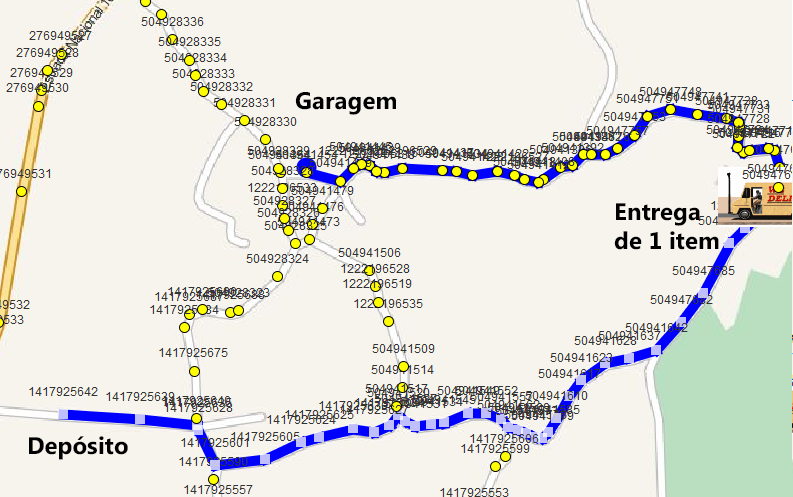
Caso selecione ‘5’, irá ser perguntado ao utilizador se pretende simular alguma estrada em obras. Esta simulação traduz-se na eliminação de arestas numa dada via impossibilitando então o camião de aí transitar.

**Fig. 4 -** 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.

**Fig. 5 -** Estrada cortada.

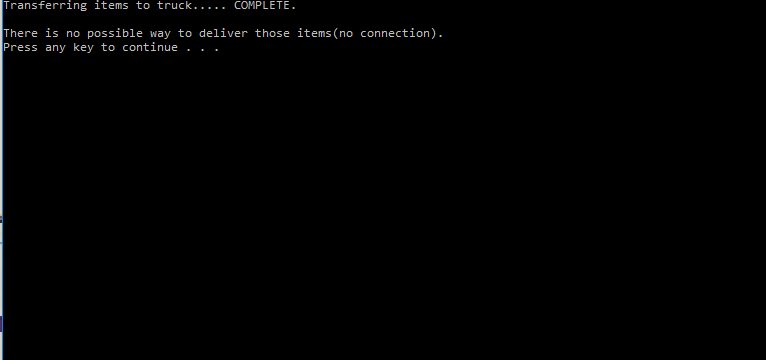
Nas vias por onde o camião passa podem existir estradas cortadas devido a obras. Nestes casos é calculado o percurso alternativo de menor caminho.



**Fig. 6 -** 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.



# 

**Fig. 7 -** Impossibilidade de entrega.

Caso, no menu da **fig. 3**, o utilizador tenha selecionado uma das outras opções, será lhe pedido que escreva o que quer procurar e será respondido com os algoritmos usados, assim como o tempo decorrido e o sucesso, ou insucesso, no resultado da procura.

Após o termino da procura, irá voltar ao menu da **fig. 3**

# **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, ⅓ 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.