

Conceção e Análise de Algoritmos

Recolha de Lixo Inteligente

2MIEIC02 – Grupo 4

(8 de abril de 2018)

Bruno Sousa	up201604145@fe.up.pt
Joaquim Santos	up201604602@fe.up.pt
Luís Silva	up201503730@fe.up.pt

Índice

Introdução	2
Descrição do Tema	2
Formalização do Problema	3
Classes	4
Ficheiros	5
Solução Implementada	6
Manual de utilização	12
Dificuldades	16
Distribuição do trabalho	17
Conclusão	18
Bibliografia	19

Introdução

Descrição do Tema – “Recolha de Lixo Inteligente”

O projeto realizado no âmbito da unidade curricular de Conceção e Análise de Algoritmos destina-se à criação de um sistema de recolha inteligente de lixo que se pretende implementar de forma a maximizar a recolha de lixo minimizando a distância percorrida pelos camiões.

Ao longo do mapa estão espalhados contentores de diversos tipos de lixo com diferentes níveis de acumulação de lixo. Existem também camiões de diversos tipos, podendo apenas recolher o tipo de lixo a que estão associados.

Formalização do Problema

Dados de entrada

T_i – Truck que o utilizador pretende enviar, que contém informação sobre o seu tipo e a sua capacidade.

$G_i = (V_i, E_i)$ – grafo dirigido pesado, composto por :

- V - Vértices (que representam pontos da cidade) com:
 - Info – (Landmark) contém informação sobre o ponto (se for Container, contém o seu tipo), principalmente o seu peso.
 - filling – quantidade de lixo acumulada no caminho até esse vértice
 - path – caminho até esse vértice
 - $Adj \subseteq E$ – conjunto de arestas que partem do vértice
- E - Arestas (representam ligações entre os pontos) com:
 - $dest \subseteq V_i$ – Vértice de destino
 - weight – distância entre os dois vértices que a aresta liga
 - name – nome da rua a que pertence a aresta

$S \in V_i$ - Vértice inicial (Garage)

$F \subseteq V_i$ – Vértice final (Treatment Station)

Dados de saída

$G_f = (V_f, E_f)$ – grafo dirigido pesado, tendo V_f e E_f os mesmos atributos que V_i e E_i .

G_{br} – Quantidade de lixo recolhida

$P = \{v \in V_i\}$ - sequência ordenada de vértices pelos quais o Truck deverá passar.

Restrições

S tem que ser uma Garage

F tem que ser uma Treatment Station

Tem que existir pelo menos um caminho possível entre S e F

Função objetivo

A solução ótima do problema passa por maximizar a quantidade de lixo recolhida por cada camião, tentando ao máximo minimizar a distância percorrida.

Classes

Company → *Singleton class* (instância da mesma limitada a um objeto) que contém a informação do programa em execução assim como os métodos para o iniciar.

Landmark → classe que contém id, coordenada x e coordenada y.

Node → classe representativa de um local (nó), um apontador para uma Landmark, podendo esta ser uma Landmark(default,placeholder), uma Garage, uma Treatment Station ou um Container(de um determinado tipo) e alguns atributos auxiliares para possibilitar/ otimizar o correto funcionamento dos algoritmos do grafo.

Edge → classe representativa de ligação entre nós.

Road → classe representativa de uma estrada (várias arestas), contendo um ID, o nome da rua e um booleano indicando se é ou não de dois sentidos.

Truck → classe representativa de um caminhão, contendo um indicador do tipo de lixo que transporta, capacidade e um indicador da garagem a que pertence.

Garage → classe representativa de uma garagem, contendo um vetor de Trucks que pertencem à garagem.

Container → classe representativa de um contentor, contendo um indicador do tipo de lixo que contém, um fator de resíduo (lixo acumulado por unidade de tempo), capacidade e quantidade de lixo atual.

Graph → classe que contém os vértices e arestas, implementando alguns algoritmos dados nas aulas e alguns auxiliares ao nosso algoritmo principal.

Treatment Station → classe representativa de uma estação de tratamento (local de destino do lixo).

Node, Edge e Graph são baseadas nas desenvolvidas nas aulas práticas, com algumas modificações para otimizar/permitir o correto funcionamento do nosso algoritmo.

Ficheiros

Edges.txt – ficheiro contendo a informação relativa a arestas como o ID da rua a que pertence e os nós que liga.

Nodes.txt – ficheiro contendo o ID do nó, as coordenadas (latitude e longitude) em graus e radianos.

Edges_Info.txt – ficheiro contendo o ID de cada rua, o nome desta e informação sobre se tem dois sentidos ou não.

Os ficheiros que utilizámos foram obtidos correndo o OSM2TXT Parser sobre dados obtidos no site openstreetmap.org .

Solução Implementada

Estrutura de Dados

A. Representação de um Graph genérico

Para a estrutura de dados do Graph (definida no ficheiro “Graph.h”), foram usados dois templates, Vertex e Edge, que representam, respetivamente, de vértices e de arestas do grafo (definidos também no ficheiro “Graph.h”).

Um vértice genérico contém:

- info: uma variável com informação sobre o ponto(contendo também o seu peso/lixo contido). É necessário que o valor de V seja distinto para vértices distintos do grafo, para que se possa identificar, de forma única todos os vértices do grafo.
- adj: um vetor de apontadores para arestas com início neste vértice.
- dist: distância percorrida até esse ponto no caminho que estamos a procurar
- fullpath: conjunto de pontos pelos quais o camião passou (necessário em vez de apenas um apontador para o último ponto visitado pois o nosso algoritmo permite a passagem pelo mesmo ponto várias vezes, apenas o “limpando” (no caso dos Containers) uma vez).
- filling: quantidade de lixo recolhido até este ponto (necessário pois os camiões têm capacidade limitada).

Uma aresta genérica contém:

- dest: um apontador para o vértice de destino.
- weight: o peso da aresta.
- name: nome da rua a que pertence.

O Graph inclui um vetor de todos os vértices. Um vetor é a melhor opção pois é necessário inserir e remover vértices, bem como iterar sobre todos eles. O vetor permite inserir e remover em $O(1)$ e iterar sobre todos elementos em $O(n)$. Contém também outros atributos necessários para a possibilidade de implementar algoritmos comuns sobre grafos.

Toda a informação contida no nosso programa está incluída na classe Company, que contém, para além do nosso grafo, um vetor com as Garages, Treatment Stations e Trucks. Nesta classe estão contidas as funções para o correto funcionamento do nosso programa, bem como o nosso algoritmo principal.

Algoritmos implementados

Leitura de ficheiros:

A leitura de ficheiros é feita em três fases : em primeiro lugar é aberto o ficheiro “Nodes.txt”, em que a informação é lida de uma linha, sendo criado um objeto da class Vertex com a informação de cada linha e adicionando o ao Graph; de seguida abre-se o ficheiro “Edges_Info.txt”, em que a informação de cada linha é usada para criar um objeto da classe Road, que é depois inserido num map, para mais tarde poder ser usado na criação de Edges(contém informação complementar necessária a este processo). Este map pertence à STL e foi escolhido pois permite a inserção, procura e remoção em $O(1)$; o último ficheiro a ser lido foi o ficheiro “Edges.txt”, em que, de cada linha eram retirados o ID da rua, o ID do node de destino e o ID do node de destino. Estes ID’s dos nodes são usados para pesquisa nos nodes existentes, e o ID da rua é usado para pesquisa no map de Roads falado anteriormente, sendo usados os vértices e a Road para adicionar uma Edge ao Graph, calculando a distância entre os dois vértices.

Sendo assim, a complexidade temporal da leitura de ficheiros é : $O(|V|) + O(|E|) + O(|E| * |V|)$
 $= O(|E| * |V|)$.

Para além disso é criado, aleatoriamente, para cada vértice, um Landmark com a seguinte probabilidade:

- 0.4% - Garage
- 0.2% - Treatment Station
- 10% - INDISCRIMINATED Container
- 10% - PAPER Container
- 10% - PLASTIC Container
- 10% - GLASS Container
- 59,4% - Landmark(default)

Cada um destes possui um ID e uma latitude e longitude relativas. Os Containers incluem também uma capacidade e fator de resíduo gerados aleatoriamente.

Relativamente à conectividade do Graph,criamos Edges em ambos os sentidos para cada Road, o que garante,para além do facto de todos os vértices estarem associados a pelo menos uma Edge, que o Graph é conexo.

Itinerário a percorrer:

Este algoritmo pretende maximizar a quantidade de lixo recolhida, minimizando a distância percorrida. Por isso, decidimos utilizar o algoritmo A* com algumas modificações (sendo já este uma adaptação do algoritmo de Dijkstra).

Para isto, baseamo-nos no algoritmo de Dijkstra implementado nas aulas práticas:

```
DIJKSTRA(G, s) : //  $G=(V,E)$ ,  $s \in V$ 
1.  for each  $v \in V$  do
2.       $\text{dist}(v) \leftarrow \infty$ 
3.       $\text{path}(v) \leftarrow \text{nil}$ 
4.   $\text{dist}(s) \leftarrow 0$ 
5.   $Q \leftarrow \emptyset$  // min-priority queue
6.  INSERT( $Q$ , ( $s$ , 0)) // inserts  $s$  with key 0
7.  while  $Q \neq \emptyset$  do
8.       $v \leftarrow \text{EXTRACT-MIN}(Q)$  // greedy
9.      for each  $w \in \text{Adj}(v)$  do
10.         if  $\text{dist}(w) > \text{dist}(v) + \text{weight}(v,w)$  then
11.              $\text{dist}(w) \leftarrow \text{dist}(v) + \text{weight}(v,w)$ 
12.              $\text{path}(w) \leftarrow v$ 
13.             if  $w \notin Q$  then // old  $\text{dist}(w)$  was  $\infty$ 
14.                 INSERT( $Q$ , ( $w$ ,  $\text{dist}(w)$ ))
15.             else
16.                 DECREASE-KEY( $Q$ , ( $w$ ,  $\text{dist}(w)$ ))
```

Tempo de execução:
 $O((V|+|E|) * \log |V|)$

Sobre este algoritmo apenas modificamos a condição:

```
if  $\text{dist}(w) > \text{dist}(v) + \text{weight}(v,w)$  then
     $\text{dist}(w) \leftarrow \text{dist}(v) + \text{weight}(v,w)$ 
     $\text{path}(w) \leftarrow v$ 
```

Para o seguinte:

If $\text{tipo}(w) = \text{tipo}(\text{Truck})$ and $\text{full}(w)$

then $\text{Garbage} \leftarrow \text{capacity}(w)$

else $\text{Garbage} \leftarrow 0$

If $\text{Garbage} \neq 0$

If $\text{Garbage} + \text{filling}(v) > \text{capacity}(\text{Truck})$

then $\text{Garbage} \leftarrow 0$

If $w \in \text{path}(v)$

then $\text{Garbage} \leftarrow 0$

```
If dist(v) + weight(w) – (10* Garbage) < dist(w)
    then
        dist(w) <- dist(v) + weight(w) – (10* Garbage)
        path(w) <- path(v) + v
        filling(w) <- filling(v) + Garbage
```

A primeira condição serve para garantir que apenas são beneficiados caminhos que passam por contentores cheios do tipo de lixo do Truck.

A segunda condição serve para garantir que apenas são beneficiados contentores enquanto o Truck não está cheio.

A terceira condição serve para garantir que o mesmo contentor não beneficia o mesmo caminho duas vezes (visto que o nosso algoritmo permite a passagem pelo mesmo ponto várias vezes) (para isto precisamos de guardar todo o caminho e não apenas o vértice anterior).

A quarta condição serve para, caso este caminho seja melhor do que o melhor já encontrado até agora, este seja substituído pelo atual, atualizando a distância, o path e a quantidade de lixo até agora recolhida. De notar que o fator de lixo é dez vezes superior ao fator de distância entre os dois pontos, garantindo assim ser mais importante a maximização da recolha de lixo e não a minimização do caminho. Dez é um fator ao qual chegamos experimentalmente e que deverá ser adaptado ao Graph escolhido para melhores resultados do algoritmo.

Caso não haja nenhum contentor cheio do mesmo tipo de lixo do camião, este algoritmo comporta-se como o algoritmo de Dijkstra entre a garagem e a estação de tratamento mais próxima.

Este algoritmo preenche-nos todos os vértices do grafo com o caminho pesado mais próximo da garagem de partida, sendo necessário encontrar a estação de tratamento com menor distância relativa.

Escolher itinerário:

Para encontrar a estação de tratamento mais próxima é necessário percorrer todas as estações de tratamento (que se encontram num vetor em Company) e verificar qual tem a menor distância. Verificamos aqui se existe alguma estação de tratamento acessível (algo que acontece sempre para grafos conexos, que é o nosso caso).

Assim que escolhermos o novo itinerário, percorremo-lo, limpando os caixotes pretendidos até ao limite de capacidade do camião.

Análise de correção:

Dado que pretendemos ao mesmo tempo maximizar a quantidade de lixo recolhida e minimizar a distância percorrida, não existe um algoritmo correto para o nosso problema.

Mesmo assim encontramos-nos confiantes neste algoritmo pois tem como base o algoritmo de Dijkstra (cuja correção se encontra provada nos slides apresentados nas aulas teóricas) com apenas algumas alterações apresentadas no último tópico (que não são passíveis de avaliação de correção dado pretenderem obter mínimos/máximos). O único fator que pode ser alterado para uma melhor correção deste algoritmo é o fator de multiplicação pela quantidade de lixo em cada contentor que deverá ser adaptado ao grafo pretendido.

Para o caso de escolher o melhor caminho até uma estação de tratamento, escolhendo o caminho com a menor distancia dará sempre o melhor resultado (semelhante ao algoritmo de Dijkstra).

Para além disso, dado que geramos os nossos dados aleatoriamente é-nos impossível analisar a correção do algoritmo no nosso programa. Não obstante, realizamos alguns testes dos quais mostramos os resultados no próximo exemplo:

Para o mesmo grafo com as mesmas garagens, estações de tratamento, contentores e camião obtivemos os diferentes resultados em termos de lixo recolhido:

```
1: Type - INDISCRIMINATED   Capacity - 100000   Garage - 2579
Select truck to send (0 to cancel): 1
Garbage Collected: 34084
```

```
Available trucks:
1: Type - INDISCRIMINATED   Capacity - 100000   Garage - 2579
Select truck to send (0 to cancel): 1
Garbage Collected: 26685
```

```
Available trucks:
1: Type - INDISCRIMINATED   Capacity - 100000   Garage - 2579
Select truck to send (0 to cancel): 1
Garbage Collected: 32746
```

```
Available trucks:
1: Type - INDISCRIMINATED   Capacity - 100000   Garage - 2579
Select truck to send (0 to cancel): 1
Garbage Collected: 32589
```

```
Avaliable trucks:
1: Type - INDISCRIMINATED   Capacity - 100000   Garage - 2579
Select truck to send (0 to cancel): 1
Garbage Collected: 34084
```

Análise de complexidade:

Complexidade temporal

Para descobrir o melhor caminho, a única alteração em termos de tempo de execução em relação ao algoritmo de Dijkstra (cuja complexidade temporal é $O((|V| + |E|) * \log(V))$) é procurar no caminho percorrido anteriormente o ponto a analisar o que torna a complexidade $O((|V| + |E|) * \log(V) * |P|)$, em que $|P|$ é o caminho médio até cada vértice.

Posteriormente, ao fazer a verificação de qual a melhor estação de tratamento precisamos de percorrer todas elas, tornando assim a complexidade temporal final: $O((|V| + |E|) * \log(V) * |P| * |T|)$ em que $V = n.^{\circ}$ de vértices, $E = n.^{\circ}$ de Edges, P = tamanho médio do caminho até um determinado vértice e $T = n.^{\circ}$ de estações de tratamento.

Para o mesmo grafo com as mesmas garagens, estações de tratamento, contentores e camião obtivemos os diferentes resultados em termos de tempo de execução:

```
Garbage Collected: 41208
Numero de nos: 3839
Tempo para calcular:0.482 segundos
```

```
Garbage Collected: 41208
Numero de nos: 3839
Tempo para calcular:0.423 segundos
```

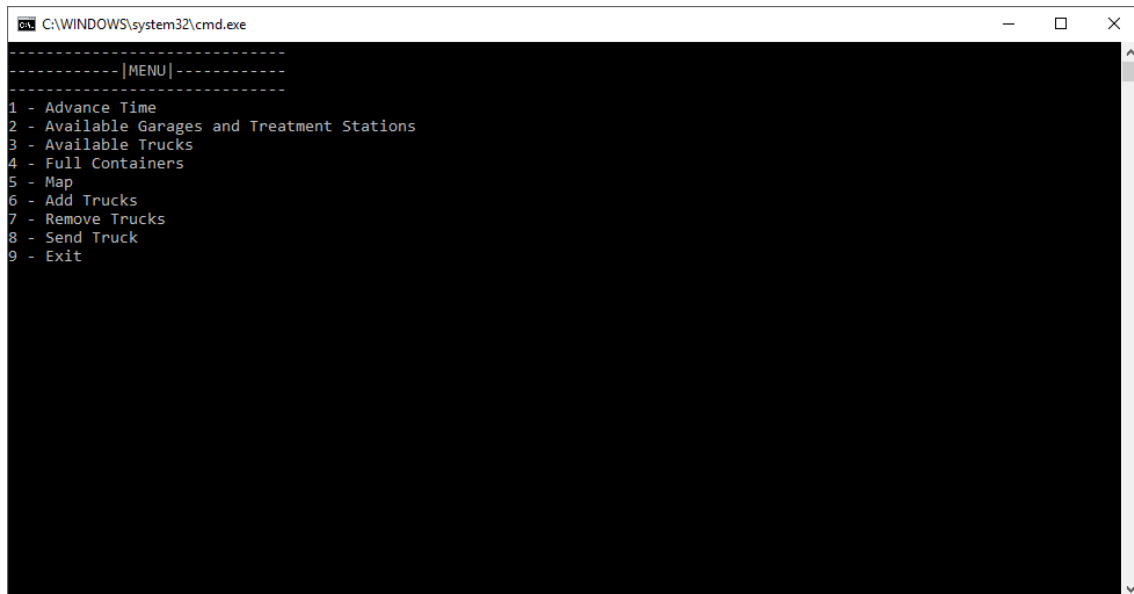
```
Garbage Collected: 41208
Numero de nos: 3839
Tempo para calcular:0.499 segundos
```

Complexidade espacial

Em termos de complexidade espacial, a única alteração do nosso algoritmo ao de Dijkstra é guardar todo o caminho até um determinado ponto, por isso, apenas acrescentamos $|P|$ à complexidade espacial do algoritmo de Dijkstra.

Manual de utilização

Menu Inicial:

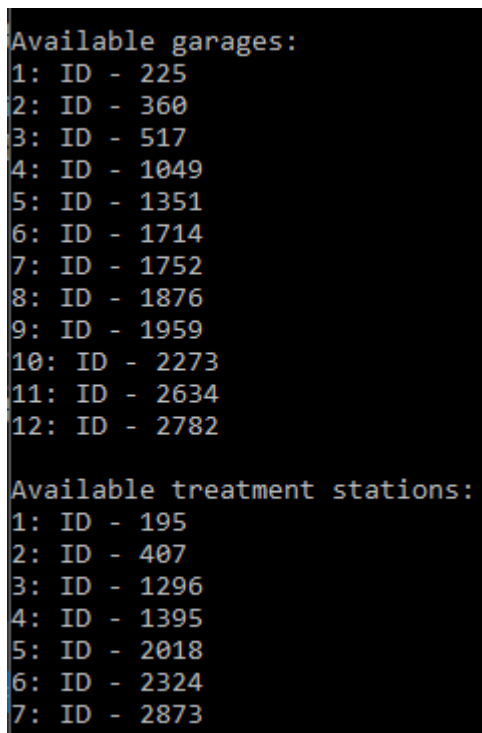


```
C:\WINDOWS\system32\cmd.exe

-----[MENU]-----
1 - Advance Time
2 - Available Garages and Treatment Stations
3 - Available Trucks
4 - Full Containers
5 - Map
6 - Add Trucks
7 - Remove Trucks
8 - Send Truck
9 - Exit
```

Escolhendo '1': é avançado o tempo, ou seja, os contentores enchem-se.

Escolhendo '2': são mostradas todas as garagens e estações de tratamento.



```
Available garages:
1: ID - 225
2: ID - 360
3: ID - 517
4: ID - 1049
5: ID - 1351
6: ID - 1714
7: ID - 1752
8: ID - 1876
9: ID - 1959
10: ID - 2273
11: ID - 2634
12: ID - 2782

Available treatment stations:
1: ID - 195
2: ID - 407
3: ID - 1296
4: ID - 1395
5: ID - 2018
6: ID - 2324
7: ID - 2873
```

Escolhendo '3': mostra todos os camiões disponíveis.

Available trucks:

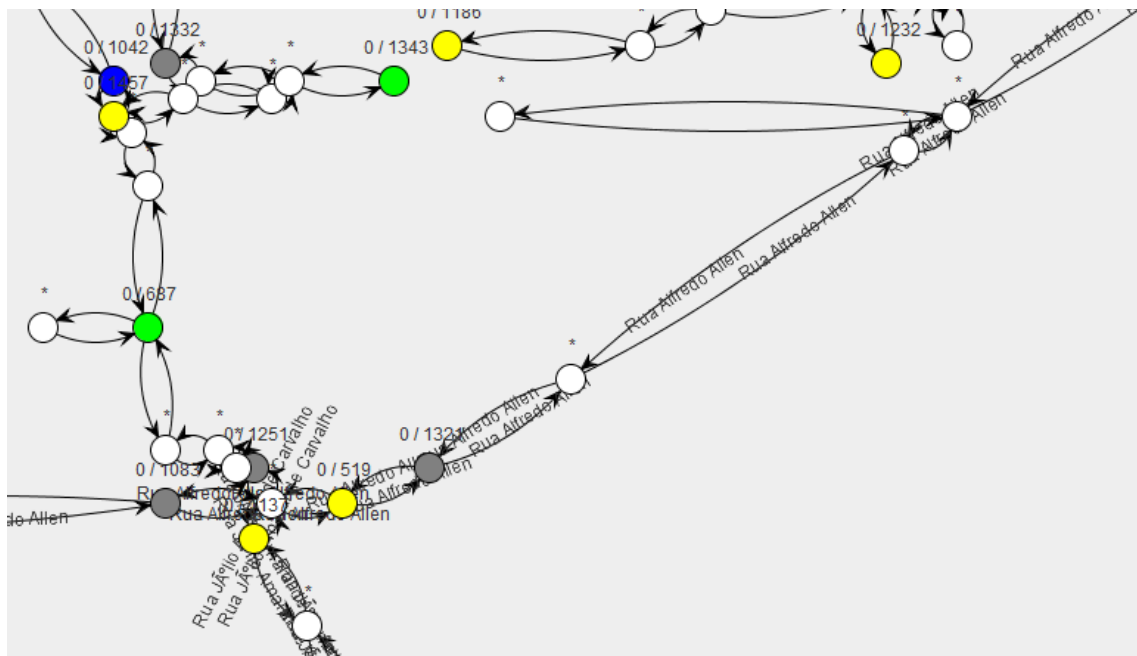
```
1: Type - INDISCRIMINATED Capacity - 20000 Garage - 1351
2: Type - PAPER Capacity - 1000 Garage - 2782
```

Escolhendo '4': mostra todos os contentores cheios.

Full containers:

1:	ID - 2	Type - GLASS	Capacity - 762	Current Load - 762
2:	ID - 6	Type - GLASS	Capacity - 744	Current Load - 744
3:	ID - 7	Type - PLASTIC	Capacity - 712	Current Load - 712
4:	ID - 8	Type - INDISCRIMINATED	Capacity - 863	Current Load - 863
5:	ID - 9	Type - INDISCRIMINATED	Capacity - 540	Current Load - 540
6:	ID - 15	Type - PAPER	Capacity - 917	Current Load - 917
7:	ID - 25	Type - GLASS	Capacity - 613	Current Load - 613
8:	ID - 30	Type - GLASS	Capacity - 522	Current Load - 522
9:	ID - 60	Type - GLASS	Capacity - 904	Current Load - 904
10:	ID - 62	Type - PLASTIC	Capacity - 1154	Current Load - 1154
11:	ID - 68	Type - GLASS	Capacity - 667	Current Load - 667
12:	ID - 82	Type - GLASS	Capacity - 630	Current Load - 630
13:	ID - 85	Type - INDISCRIMINATED	Capacity - 769	Current Load - 769
14:	ID - 88	Type - GLASS	Capacity - 552	Current Load - 552
15:	ID - 91	Type - GLASS	Capacity - 692	Current Load - 692
16:	ID - 96	Type - PLASTIC	Capacity - 962	Current Load - 962
17:	ID - 98	Type - GLASS	Capacity - 725	Current Load - 725
18:	ID - 105	Type - INDISCRIMINATED	Capacity - 1365	Current Load - 1365

Escolhendo '5': mostra o mapa no GraphViewer com os nomes das ruas, a amarelo caixotes de plástico, a azul de papel, a verde de vidro, a cinzento de indiferenciado, a magenta as garagens e a preto as estações de tratamento. Os contentores também mostram a sua ocupação e capacidade.



Escolhendo '6': adiciona um caminhão com dados fornecidos pelo utilizador.

```
Choose truck type:
1 - INDISCRIMINATED
2 - PLASTIC
3 - PAPER
4 - GLASS
2

Insert truck capacity: 10000

Available garages:
1: ID - 578
2: ID - 920
3: ID - 1315
4: ID - 1904
5: ID - 2062
6: ID - 2075
7: ID - 2520
8: ID - 2609
9: ID - 2876
10: ID - 2892
11: ID - 3035
12: ID - 3525
13: ID - 3628
8
```

Escolhendo '7': remove um caminhão com o id fornecido pelo utilizador.

```
Avaliable trucks:

1: Type - PLASTIC   Capacity - 10000   Garage - 2609
```

Escolhendo '8': envia um caminhão escolhido pelo utilizador mostrando a quantidade de lixo recolhida e mostrando a vermelho, no mapa, o caminho percorrido pelo caminhão.

```
Avaliable trucks:

1: Type - PLASTIC   Capacity - 100000   Garage - 3035

Select truck to send (0 to cancel): 1

Garbage Collected: 36350
```


Dificuldades

Durante a realização deste projeto não se apresentaram grandes dificuldades em termos de algoritmos.

As nossas maiores dificuldades encontraram-se na integração de dados do openstreetmap.org, utilização do GraphViewer e adaptação da classe Graph elaborada nas aulas práticas.

Distribuição do Trabalho

Todos os membros do grupo de esforçaram igualmente na realização do trabalho.

Conclusão

Tendo como objetivos a compreensão e familiarização com novas estruturas de dados como grafos e algoritmos de pesquisa nos mesmos, podemos concluir que os objetivos foram atingidos, quer a nível individual, quer a nível de grupo, uma vez que todos os elementos têm, após a realização deste trabalho, uma melhor compreensão dos temas lecionados na unidade curricular.

Bibliografia

- Slides das aulas teóricas
- GraphViewer API – <http://jung.sourceforge.net/>
- OSM2TXT Parser
- Open Street Maps – <http://openstreetmap.org/>