Faculdade de Engenharia da Universidade do Porto

Tema 2:

WasteApp

Recolha seletiva de lixo

Concepção e Análise de Algoritmos

2019-2020

24/04

Turma 5 - Grupo 3:

Abel Augusto Dias Tiago	201107963	ei11102@fe.up.pt
Miguel Carreira Neves	201608657	up201608657@fe.up.pt
Tomás Freitas Gonçalves	201806763	up201806763@fe.up.pt

Índice

1. Introdução ao Problema

1.1 Descrição do tema

De modo a tentar reduzir o desperdício e aumentar a reciclagem será criada a aplicação WasteApp. Esta permite que os utilizadores façam a gestão do seu próprio lixo e reciclagem, encontrem locais próximos para a recolha e obtenham mais informações acerca de onde os itens devem ser depositados.

A aplicação permite localizar os pontos de recolha seletiva, informando acerca do tipo de resíduo que pode ser depositado, assim como a capacidade total e disponível de cada um.

A app permite a cada utilizador saber quais os pontos de recolha mais próximos com capacidade para depósito de resíduos.

Através da WasteApp é possível gerar um itinerário de recolha otimizado, passando pelas residências de todos os utilizadores que desejam ter o lixo recolhido em casa (trazendo um custo associado).

Temos como objetivo minimizar o número de camiões, assim como o trajeto percorrido por cada um. Convém também realçar que cada contentor deve ser recolhido na sua totalidade (*sendo que se um camião não tiver espaço suficiente para um contentor completo deverá deixá-lo para o seguinte*). Consideramos que todos os tipos de resíduo serão associados a cada centro de reciclagem.

O mapa sobre o qual a WasteApp atua e analisa pode ser carregado por exemplo de OSM (Open Street Maps) ou de outras fontes compatíveis. Estas fontes devem conter a localização dos pontos de recolha, garagem dos camiões e centro de recolha.

2. Formalização do Problema

Após a análise do enunciado é possível separar os utilizadores da aplicação em três principais tipos:

Utilizador particular (**UP**): pretende saber onde se encontram os pontos de recolha, a sua capacidade atual e quais resíduos podem ser depositados nesses pontos. Pode também estabelecer a sua morada como ponto de recolha de resíduos domiciliário. Admite-se que na sua morada este terá um sistema de sensor no seu lixo tal como nos outros pontos de recolha que comunicam a ocupação atual do lixo.

Utilizador empreendedor (**UE**): pretende saber onde se encontram os pontos de recolha de resíduos domiciliários e qual o caminho mais eficaz de recolha de um particular resíduo até a sua central de reciclagem. Para tal tem de estabelecer o interesse na recolha de num determinado tipo de resíduo.

Utilizador municipal (**UM**): pretende saber as rotas mais eficazes para recolha dos resíduos entregues nos pontos de recolha desde a garagem até a central de tratamento e reciclagem.

ADMIN :

Tendo formalizado as necessidades de cada utilizador, chegamos à conclusão que apenas podem ser criados pontos de recolha domiciliários após um utilizador empreendedor estabelecer interesse na recolha de um tipo de resíduo. E que o utilizador empreendedor é uma derivação do utilizador particular, sendo que deve manter as mesmas funcionalidades de um utilizador particular. Portanto um utilizador empreendedor também consegue aceder a todas as funcionalidades que o particular acede na aplicação.

De modo a facilitar a implementação, decidimos resolver o problema em vários etapas:

1. Extração e tratamento de dados de entrada e criação de método de geração de dados automático

2. Criação de classes com a informação extraída / gerada

3. Criação de menus referentes a cada utilizador, com funcionalidades específicas para cada um

4. Criação de grafos dependentes da necessidade de cada utilizador

5. Implementação do algoritmo mais apropriado para cada utilizador

6. Tratamento e apresentação dos resultados a cada utilizador

2.1 Dados de Entrada

- $Gi = (Vi, Ei)$, grafo orientado pesado com base no mapa cujos vértices são todos os pontos de interesse para o problema e as arestas representam as ruas e o seu peso equipara-se à distância:

  $Vi \supset S, D, P, R$:

  idV - identificador único do vértice

  adji $\subseteq$ Ei - conjunto de arestas que partem do vértice

  Ei:

  w- peso da aresta(representa a distância entre os dois vértices das pontas)

  idE - identificador único da aresta

  dest $\in$ Vi - vértice de destino da aresta

- tipo - tipo de lixo a recolher/depositar (ex: indiferenciado, vidro, etc...).

- taxaVia - taxa de ocupação dos pontos de recolha a partir da qual a recolha já se torna interessante e viável para o UM;

- M - conjunto do tipo de resíduo, peso mínimo e peso máximo a ser transportado pelo UE, de forma a tornar rentável o seu transporte. Cada conjunto deve ter as seguintes informações:
  - t - tipo de resíduo;
  - pMin - peso mínimo viável;
  - pMax - peso máximo transportado pelo UE em Kg, por viagem executada.

- T - conjunto do tipo de resíduo e respetivo peso total a ser depositado pelo UP. Cada resíduo deve ter as seguintes informações:
  - t - tipo de resíduo;

        - pTot- peso total a transportar do respetivo resíduo;

- S - conjunto de pontos de recolha sendo S(i) o ponto de recolha com ID i.
    - Cada ponto de recolha deve ter as seguintes informações:
        - SID - ID do ponto de recolha;
        - XY - coordenada do ponto de recolha;
        - T - conjunto de tipos de resíduos que este ponto aborda sendo T(t) o tipo de resíduo;
        - O - conjunto das percentagens de ocupação sendo O(t) a percentagem de ocupação do tipo t de resíduos;
        - K - conjunto das capacidades máximas sendo K(t) a capacidade máxima em Kg do resíduo t.

- D - conjunto de pontos de recolha domiciliários sendo D(i) o ponto de recolha com ID i.
    - Cada ponto de recolha deve ter as seguintes informações:
        - DID - ID do ponto de recolha;
        - XY - coordenada do ponto de recolha;
        - T - conjunto de tipos de resíduos que este ponto aborda sendo T(t) o tipo de resíduo;
        - K - conjunto dos pesos de resíduos sendo K(t) o peso do tipo t de resíduos;

- P - conjunto de pontos de partida, seja municipal, empreendedora ou particular sendo P(i) o ponto partida com ID i.
    - Cada ponto de partida deve ser composto por:
        - PID - ID do ponto de partida;
        - XY - coordenadas do ponto de partida;

- R - conjunto dos centros de reciclagem sendo R(i) o centro de reciclagem com ID i.
    - Cada centro de reciclagem deve ter as seguintes informações:
        - RID - ID do ponto de recolha;
        - XY - coordenada do ponto de recolha;

- C - conjunto dos diferentes tipos de camiões sendo C(t) o camião que transporta o tipo t de resíduos.
    - Cada camião tem como características:
        - t - tipo de resíduos que recolhe;
        - cap - Capacidade atual em percentagem;
        - capMax - Capacidade máxima em Kg;

2.2 Dados de saída

- $Gs = (Vs,\ Es)$ - Grafo **Gs** dirigido pesado com pesos sempre positivos, composto por:
    - Vs - vértices que representam os pontos de recolha de resíduos, pontos de recolha domiciliários, pontos de partida da recolha e centros de reciclagem, sendo constituídas por:
        - VID - ID do vértice, igual ao ponto que o criou;
        - Adj $\subseteq$ Es - conjunto de arestas que partem do vértice;
        - XY - coordenada do ponto que a criou;

    - Es - arestas que representam as ligações entre os diferentes pontos, sendo constituídas por:
        - z - peso da aresta, determinado pela distância entre pontos;
        - ED - vértice de destino da aresta;

- U - sequência ordenada de arestas que representam o caminho ideal para o utilizador em particular.
    - U é dividido pelos diferentes utilizadores
        - U(UP) - conjunto ordenado de arestas com o caminho ideal dependente de:
            - T- conjunto de tipos de resíduos a serem depositados pelo EP
        - U(UE) - conjunto ordenado de arestas com o caminho ideal dependente de:
            - tipo - tipo de lixo a recolher/depositar;
            - pesoE - peso máximo transportado pelo UE em Kg;
            - M(t) - o peso mínimo do tipo t de resíduo
        - U(UM) - conjunto ordenado de arestas com o caminho ideal dependente de:
            - tipo - tipo de lixo a recolher/depositar;
            - C(t) o camião que transporta o tipo t de resíduos;
            - taxaVia - taxa de ocupação dos pontos de recolha a partir da qual a recolha já se torna interessante e viável para o UM;

Nota: o número de U(UM)s vai indicar o número de camiões necessários para percorrer todos os pontos válidos de recolha.

2.3 Restrições

2.3.1 Dados de Entrada:

Nenhum dado pode ter o seu conteúdo vazio.

O grafo Gi deve ter conectividade de forma a garantir que é possível chegar a todos os pontos de recolha e a pelo menos um centro de reciclagem.

Além disso $\forall e \in Ei$, 'e' tem de ter o seu w > 0 pois representa uma distância.

0 < taxaVia < 1 - Sendo *taxaVia* o parâmetro que representa a ocupação possível de resíduos nos contentores para recolha. Este deverá ser superior a 0, caso contrário estariam a ser recolhidos contentores vazios e inferior a 1 de forma a evitar que o contentor fique lotado.

Seja o conjunto $M \subset M(t,p)$ cada t representa um tipo de resíduo e p > 0, pois nunca será viável transportar 0 resíduos.

$\sum_{p \in M} M(pMin) < M(pMax)$,pois não se pode exceder o peso disponível do UE.

Seja o conjunto $T \subset T(t,pTot)$ cada t representa um tipo de resíduo e pTot > 0, pois nunca será viável depositar resíduos com peso nulo ou negativo.

Para $S \subset S(i)$:

Considerando o conjunto $O \subset O(t)$ cada t representa um tipo de resíduo e 0<= O(t) <=1 pois representa uma taxa.

Além disso, tendo em conta $K \subset K(t)$, $\forall kt \in K(t)$, kt > 0

Para $D \subset D(i)$:

Sendo $K \subset K(t)$, $\forall kt \in K(t)$, kt >= 0

Seja o conjunto $C \subset C(t)$ cada t representa um tipo de resíduo e para este mesmo C(t), 0<= cap <=1 e capMáx > 0 .

2.3.2 Dados de Saída:

No grafo Gs(Vs,Es), $\forall e \in Es$, 'e' tem de ter o seu z > 0 pois representa uma distância.

Na sequência U:

Seja e1 o primeiro elemento de U(UM). É preciso que e1 $\in$ Adj(P), pois o camião parte da central.

Seja ep o último elemento de U(UM) ou de U(UE). É preciso que ep $\in$ Adj(R), pois o camião termina sempre num centro de reciclagem.

2.4 Funções Objetivo

Dependendo dos diferentes utilizadores, as funções objetivo variam. Contudo uma função abrange os três tipos de utilizadores. Definimos primeiro o termo e_{ij} que representa uma aresta entre os vértice de índice *i* e o vértice de índice *j.* Definimos também **L** como o conjunto de vértices que ligam estas arestas. Relembramos que **S** foi definido como o conjunto de vértices que são pontos de recolha

$$e_{ij} \in E \wedge v_i, v_j \in V \rightarrow \forall\ v_i, v_j \in L$$

Podemos então definir a função objectiva do utilizador UP como a minimização do somatório de arestas contíguas que passem por todos os pontos de recolha (tendo como início e destino o vértice da sua casa).

$$d:\ min(\sum w(e_{ij})) \wedge S \subseteq L$$

O utilizador UE para além de requerer o percurso mais curto, também pretende maximizar a quantidade de resíduo que se disponibiliza a recolher para obter mais lucro. Sendo assim, o caminho mais curto não é necessariamente o mais eficaz, sendo que devemos priorizar a quantidade recolhida antes de minimizar as distâncias entre esses pontos.

j : $máx(\sum K(t) \in K \in D)$ ← máximo lixo recolhido dos pontos de recolha domiciliários

O utilizador UM pretende minimizar o número de camiões usados assim como as distâncias totais percorridas por cada um. No entanto, o seu objectivo principal é conseguir recolher o máximo de contentores possíveis. As diferentes funções objetivo em ordem em relação à sua prioridade são:

1. c: Em Gs: $\forall\ v \in Vs$: VID == (SID $\in S$), $min(\sum O(t))$ ← para assegurar que foi recolhido a máxima quantidade de lixo, a soma da taxa de ocupação deve ser a menor possível

2. n : min (|U(UM)|) ← minimizar o número de camiões usados

3. f: $min(\sum w(e_{ij})) \wedge S \subseteq L$ ← minimizar a distância total percorrida

3. Perspetiva e Descrição da Solução

Para este problema é necessário passar por vértices específicos (contentores) antes de chegar ao seu destino final. Teremos de aplicar sucessivamente o algoritmo para cada ponto de interesse (sendo o ponto final desse algoritmo e portanto o ponto inicial do algoritmo na próxima iteração). Admitindo K como o conjunto dos vértices com os endereços dos contentores e da Estação de Tratamento, o seu tempo computacional é o original multiplicado por
K → $O((|V| + |E|) \cdot log|V| \cdot |K|)$

O problema da recolha de lixo é parecido com o Problema do Caixeiro Viajante abordado nas aulas teóricas com a exceção de que neste nosso problema o ponto de partida e o de chegada já estão definidos previamente. O problema surge da necessidade dos antigos Caixeiros Viajantes realizarem diversas entregas numa cidade, percorrendo o menor trajeto possível passando por todas as casas uma única vez. Minimizando o tempo necessário e custos de transporte.

Neste problema pretende-se determinar a menor rota possível passando por uma série de pontos predeterminados onde se encontram os contentores do lixo.

Decidimos utilizar o método Nearest Neighbour para abordar esta problemática do TSP:

(Cada vértice do grafo apresenta um atributo que indica se este foi visitado ou não)

1) É escolhido um ponto de partida;
2) Procura-se o vértice mais próximo não visitado;
3) Marca-se o atual como visitado e move-se para o seguinte;
4) Repete-se os passos 2) e 3) enquanto houver vértices por visitar;
5) Finalmente retorna-se ao ponto inicial.

3.1 Cálculo da Distância mínima entre dois vértices

Para implementar este cálculo, concluímos que o melhor algoritmo a usar seria o de Dijkstra que foi também falado nas aulas teóricas. Este permite encontrar um caminho mais curto num grafo dirigido com complexidade temporal O((|E| + |V|) * log V).

Este algoritmo é um algoritmo ganancioso, ou seja, toma decisões que parecem ótimas no momento, determinando o conjunto de melhores caminhos intermediários. Logo este algoritmo pode não apresentar a solução perfeita no entanto apresenta sempre uma solução ótima e de forma bastante rápida.

Uma das fragilidades deste algoritmo é não funcionar corretamente caso existam arestas com peso negativo no grafo, no entanto isto nunca acontece no nosso problema.

Como o peso de cada aresta representa uma distância entre dois vértices, este algoritmo irá ser utilizado para otimizar as distâncias percorridas pelos camiões ou UE nas suas rotas pelos pontos de recolha e com destino final no centro de recolha.

Pseudocódigo:

```
Dijkstra(G, s):
1.    for each v ∈ V do:
2.        dist(v) ← ∞
3.        path(v) ← nil
4.    dist(s) ←nil
5.    Q ← ∅                    //inicializa a fila de prioridade
6.    INSERT(Q, (s, 0))
7.    while Q ≠ ∅ do:
8.        v ← EXTRACT-MIN(Q)
9.        for each w ∈ Adj(v) do:
10.           if dist(w) > dist(v) + weight(v, w) then:
11.               dist(w) ← dist(v) + weight(v, w)
12.               path(w) ← v
13.               if w ∉ Q then:
14.                   INSERT(Q, (w, dist(w)))
15.               else:
16.                   DECREASE-KEY(Q, (w, dist(w))
```

3.2 Análise da conectividade do grafo

Terá de ser testada a conectividade dos diferentes vértices do grafo, de forma a verificar se é possível chegar a todos os pontos de recolha e a pelo menos um centro de reciclagem.

Para analisar da melhor forma a conectividade temos os seguinte algoritmo:

3.2.1 **Algoritmo de Pesquisa em Largura**

Iniciando num vértice origem, percorre-se todos os vértices a que se podem chegar a partir do vértice origem e só depois se passa para outro vértice origem e realiza-se o mesmo processo.

Este tem uma complexidade temporal de $O(|V| + |E|)$ e uma complexidade espacial $O(|V|)$.

Pseudocódigo:

```
BFS(G, s):
1.    for each v ∈ V do:
2.        visited(v) ← false
3.    Q ← ∅                          //inicializa a fila de prioridade
4.    push(Q, s)
5.    visited(s) ← true
6.    while Q ≠ ∅ do:
7.        v ←pop(Q)
8.        for each w ∈ Adj(v) do:
9.            if not visited(w) then:
10.               push(Q, w)
11.               visited(w) ← true
```

3.2.2 **Algoritmo de Pesquisa em Profundidade**

Neste algoritmo, as arestas a serem exploradas são as adjacentes do último vértice descoberto. Quando já não houver mais arestas adjacentes, será necessário recorrer ao *backtracking*.

Este tem uma complexidade temporal $O(|V| + |E|)$ e uma complexidade espacial $O(|V|)$.

Pseudocódigo:

```
DFS(G):
1.    for each v ∈ V do:
2.        visited(v) ← false
3.    for each v ∈ V do:
4.        if not visited(v) then:
5.            VisitDFS(G, v)

VisitDFS(G, v):
1.    visited(v) ← true
2.    for each w ∈ Adj(v) do:
3.        if not visited(v) then:
4.            VisitDFS(G, w)
```

3.3 Número de camiões necessários

Para determinar quantos camiões serão necessários é necessário avaliar quantos tipos de lixo há para recolher. Visto que cada camião apenas recolhe um tipo de lixo, é necessário, pelo menos, um veículo para cada tipo existente. De seguida analisa-se para cada tipo de resíduos o número de camiões necessários: dividindo-se a soma de todas as capacidades atuais dos pontos de recolha S (calculadas fazendo a percentagem de ocupação O(t) * capacidade máxima K(t)) pela capacidade máxima de um camião C(t), sendo o resultado arredondado por excesso de forma a garantir que existe capacidade suficiente.
Sendo o objetivo portanto minimizar a função objetivo n (na seção 2.4).

4. Identificação dos casos de utilização (Use cases)

1. Carregamento de um mapa real

Carregamento de um mapa representado por um grafo com arestas que representam ruas e vértices que representam pontos de interesse (pontos de recolha, garagem, etc) de forma a depois este mapa ser utilizado pelo programa. As tags são geradas aleatoriamente no caso em que os mapas já não as contém.

2. Adicionar ponto de recolha (ADMIN)

Adição de um ponto de recolha ao grafo, tendo especificado o seu tipo e a sua capacidade máxima, sendo feita uma análise ao grafo de forma a verificar a conectividade a este vértice para evitar pontos de recolha em zonas inacessíveis.
Nesta segunda entrega após refletir chegamos a conclusão que esta funcionalidade deveria ser da responsabilidade de algum administrador e não dum utilizador "normal" da aplicação.

3. Determinar rota dos camiões para recolha de resíduos (UM)

Cálculo do caminho mais curto entre a garagem e o centro de recolha, passando pelos pontos de recolha considerados interessantes para recolher (cuja percentagem de ocupação ultrapasse um valor definido previamente). Esta funcionalidade é das principais da aplicação, sendo usada para otimizar as viagens tendo em conta o número de camiões e as distâncias mas principalmente tentando maximizar a quantidade de lixo recolhido. Admitimos que o número de camiões previamente disponíveis é ilimitado e que têm todos a mesma capacidade limite. (É utilizada uma abordagem greedy que consiste em percorrer todos os lixos que necessitam ser recolhidos escolhendo sempre como próximo destino o lixo que necessita de ser recolhido mais próximo e após não conseguir carregar mais lixo, dirigir-se à central de recolha).

4. Executar recolhas ao domicílio (UE)

Esta funcionalidade será apenas utilizada pelo UE ao qual será pedido, *a priori,* a capacidade máxima que este consegue transportar, e demonstrando um caminho passando pelos pontos de recolha domiciliários apresentados de uma forma eficiente. O algoritmo em si utilizado é extremamente parecido ao utilizado no use case 3, com algumas ligeiras e quase insignificantes modificações. O seu destino final é o centro de Recolha e o ponto de partida a sua morada ou a sua localização

atual que este irá indicar. (No decorrer deste relatório é referido o lucro do UE numa recolha no entanto esta feature acabou por não ser utilizada).

5. Subscrever à recolha doméstica (UP)

Permite adicionar uma morada como um ponto de recolha doméstica, sendo necessário especificar o peso e o tipo de resíduo ou tipos de resíduos e as suas capacidades máximas. Esta funcionalidade tem um custo previamente especificado. É considerado que os pontos de recolha doméstica têm sensores também capazes de comunicar a sua taxa de ocupação.

6. Visualização dos pontos de recolha mais próximos (UP)

É apresentado o mapa atual e a localização dos pontos de recolha na vizinhança que ainda têm capacidade livre assim como os tipos de lixo que armazenam. É apenas utilizada pelos utilizadores particulares e não tem custo nenhum acrescido. O mapa apresentado mostra o ponto de recolha mais próximo para cada tipo de lixo. É apresentado um mapa com o grafo representando o ponto de localização atual do utilizador e o ponto de recolha com o tipo de lixo especificado (com cores diferentes dos restantes ambos estes ambos).

5. Estruturas de Dados

Na programação desta aplicação criamos um ficheiro Dados.h que contém as diferentes estruturas de dados:

```
class VerticeInfo
VerticeInfo(Coordenadas &coordenadas, int id)
```
-É utilizada como o tipo de info no grafo (a classe template T) e contem as coordenadas do ponto e o seu id

```
class PontoRecolhaDomiciliario : public virtual VerticeInfo
PontoRecolhaDomiciliario(Coordenadas coordenadas, vector<TipoLixo>
&tipoLixo, vector<double> capMax, int id)
```
-Representa todos os pontos domiciliários e extende a classe VerticeInfo de modo à poder ser inserida também no grafo como a info de cada vertice

```
class PontoRecolha : public virtual VerticeInfo
PontoRecolha(Coordenadas &coordenadas, vector<TipoLixo> &tipoLixo,
vector<double> capMax, int id)
```
-Equivalente aos PontosRecolhaDomiciliários apenas representam outro tipo de objetos

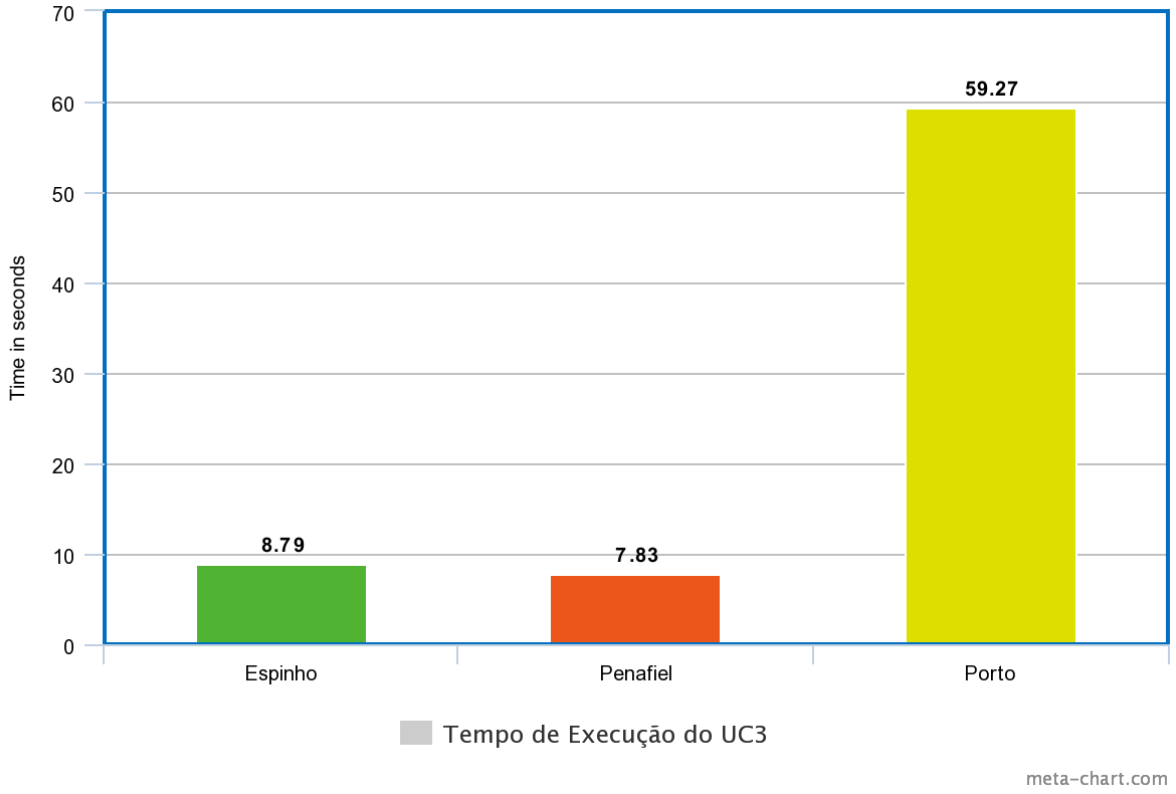
```
class PontoPartida : public virtual VerticeInfo
PontoPartida(Coordenadas coordenadas, int id) :
VerticeInfo(coordenadas, id)
```
-Equivalente ao VerticeInfo apenas serve para diferenciar a nivel de código o ponto de partida dos restantes

```
class CentroReciclagem : public virtual VerticeInfo
CentroReciclagem(Coordenadas coordenadas, int id) :
VerticeInfo(coordenadas, id)
```
-Semelhante desempenho ao PontoPartida

```
class Camiao
Camiao(TipoLixo tipoLixo) : tipoLixo(tipoLixo)
```
-Tem uma capacidade atual, capacidade máxima e um tipo de lixo específico que está destinado a carregar de momento

6. Tempo de Execução

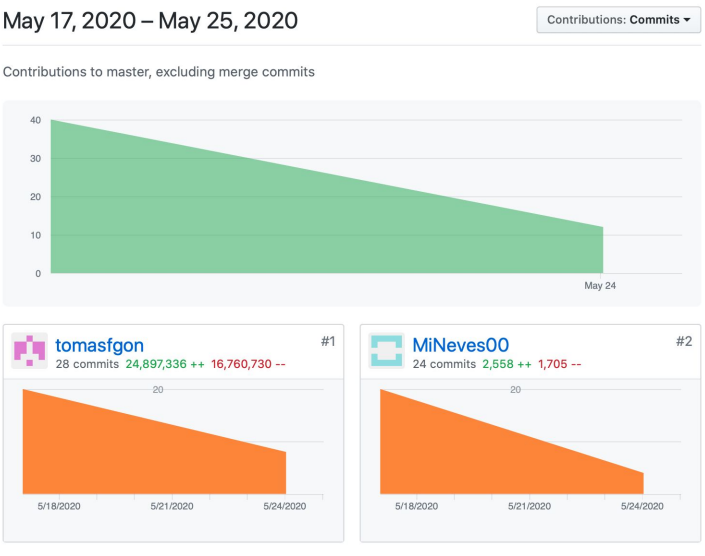
Testámos o tempo de execução para o UC3 variando o grafo inserido (cidades diferentes) portanto sendo uma análise empírica, há que relembrar que estes tempos de execução também contam com o tempo de impressão das strings contendo o id de todos os vértices na rota, daí os tempos também serem maiores visto que a impressão de strings atrasa bastante. A análise está representada no gráfico seguinte sendo os tempos apresentados uma média dos tempos que resultaram:



7. Conclusão

Ao longo de todo o trabalho foi discutida uma eventual solução para o problema que nos foi apresentado.
Ao longo da segunda parte do trabalho, tentamos implementar tudo o que tinha sido planeado no relatório da primeira entrega. Utilizámos o github nos ajudar com a implementação progressiva do código.
Pensamos ter coberto a grande maioria das funcionalidades pretendidas, contudo, certos aspetos do programa não ficaram implementadas da forma que desejávamos inicialmente:
a leitura de ficheiros apenas nos permite ler e analisar grafos fortemente conexos. Na criação da rota ideal (menor trajeto possível, passando por todos os contentores com uma quantidade superior 50% de lixo, utilizando o menor número de camiões possível), apesar da mesma ser bem definida, a rota não é imediatamente alterada após alguns contentores terem sido esvaziados (apenas numa nova chamada à funcionalidade esta rota é atualizada). Isto leva a que todos os camiões sigam uma mesma rota, passando por contentores esvaziados previamente. Além disso a aplicação apenas está feita para grafos fortemente conexos

A implementação do código foi efetuada na totalidade por apenas dois membros do grupo (Tomás Gonçalves e Miguel Neves) o que acabou por dificultar bastante o desenvolvimento do projeto... A ferramenta insights do github permite observar que o trabalho foi repartido corretamente pelos dois.



Auto-avaliação 2ª Entrega

Miguel Carreira Neves - 50%

Tomás Freitas Gonçalves - 50

8. Bibliografia

-Material fornecido pelos docentes

-Travelling salesman problem:
https://en.wikipedia.org/wiki/Travelling_salesman_problem

-Problema do caixeiro viajante:
http://www.mat.ufrgs.br/~portosil/caixeiro.html

-Algoritmo de Dijkstra, Wikipedia:
https://pt.wikipedia.org/wiki/Algoritmo_de_Dijkstra

-Algoritmo de Dijkstra, USP:
https://www.ime.usp.br/~pf/algoritmos_para_grafos/aulas/dijkstra.html

-Busca em largura, Wikipedia:
https://pt.wikipedia.org/wiki/Busca_em_largura

-Busca em profundidade, Wikipedia:
https://pt.wikipedia.org/wiki/Busca_em_profundidade