Relatório CAL - 1ª Parte Trabalho Prático
11/4/2019

Tema 8 — SecurityVan: Entrega de Valores

Grupo D - Turma 4

Eduardo Ribeiro —> up201705421@fe.up.pt

Eduardo Macedo —> up201703658@fe.up.pt

Diogo Machado —> up201706832@fe.up.pt

Nota: a 2ª parte do relatório pode ser encontrada no final.

Descrição do Tema

O trabalho está relacionado com o conceito de veículos especializados em transporte e entrega de valores, que transportam grandes quantias de dinheiro e outros objetos valiosos, de um ponto para outro.

O trabalho consiste em implementar um sistema que permite a identificação de rotas ótimas para tais transportes, de modo a ser utilizado por uma empresa que se especializa em transporte de valores. Inicialmente, deve ser considerado que a empresa apenas dispõe de um veículo, a fazer todos os trajetos/entregas. Numa fase mais posterior, considerar-se-à que a empresa tem vários veículos e que os seus trabalhos/trajetos poderão ser especializados por tipo de cliente (bancos, museus, etc).

Estes transportes implicam a recolha prévia dos valores em certos pontos, para depois os entregar.

Deveremos, então, inicialmente, definir a sequência ordenada dos pontos de interesse pelos quais o(s) veículo(s) terão de passar, considerando que os trajetos deste(s) começarão e acabarão sempre na central. Há que ter em conta que os destinos têm de aparecer depois que as origens correspondentes, na sequência ordenada.

Um exemplo de um trajeto que um veículo pode fazer é:

central —> origem1 —> origem2 —> destino2 —> origem3—> destino3 —> destino1 —> central

Depois de se saber a sequência ordenada dos pontos de interesse, é necessário saber qual o caminho mais rápido de um ponto até outro, dois a dois, definindo assim a melhor rota para o veículo.

A entrega só poderá ser efetuada se existirem caminhos que liguem a central dos camiões, as origens, os destinos, e de volta à central, dois a dois. Ou seja, para a entrega se poder concretizar, é necessário que os pontos façam parte do mesmo componente fortemente conexo do grafo, pelo que é de extrema importância a avaliação da conectividade do mesmo.

Por último, será também necessário considerar que obras nas vias públicas poderão tornar certas zonas do mapa inacessíveis, podendo ser impossível chegar a certos clientes; devem ser identificados os pontos de recolha/entrega com acessibilidade reduzida.

Identificação e Formalização do Problema

Dados de Entrada

- R —> tabela com valores das distâncias/tempos mínimos entre cada par ordenado de vértices, e possivelmente o caminho que leva a essas distâncias/tempos (ver perspetiva de solução);

- Ti —> sequência de trajetos/entregas que será necessário fazer, sendo i o i-ésimo elemento. Cada entrega será caracterizada por:

    - type —> tipo de cliente (museu, banco, loja, …) (na 1ª parte do problema, isto não vai ser considerado, uma vez que há apenas um veículo para todo o tipo de transportes);

    - origem —> vértice em que será necessário ir recolher os bens;

    - destino —> vértice em que será necessário entregar os bens, depois de estarem dentro do veículo;

    - ID —> um ID único e específico, para cada trajeto/entrega.

- Ci —> veículo número i, que está disponível para fazer um percurso (na 1ª parte do problema, apenas vai ser considerado um único veículo). Cada um tem:

    - type —> tipo de entrega/trajeto (museu, banco, loja, …) em que o veículo é especializado (na 1ª parte, o único veículo irá fazer todos os trajetos, pelo que não terá especialidade).

- G = (V, E) —> grafo dirigido(*) pesado, representando o mapa da cidade. Este grafo irá ser constituído por vértices (V) e arestas (E).

Cada vértice (representam pontos de importância no mapa: possíveis clientes, interseções entre vias de trânsito, etc.) é caracterizado por:

- gpsCoords —> coordenadas GPS do ponto no mapa (opcional; ver perspetiva de solução);

- type —> o vértice representa um museu, banco, loja, ou nenhum dos anteriores;

- Adj $\subseteq$ E —> arestas que saem desse vértice.

Cada aresta (representam ruas, pontes e outras vias) é caracterizada por:

- w —> peso da aresta, que representa o tempo/distância do percurso entre os dois vértices que esta liga;

- ID —> identificador único de uma aresta;

- dest $\in$ V —> vértice de destino da aresta;

- S $\in$ V —> vértice inicial, que representa a central de onde os veículos saem.

(*) —> é considerado dirigido, pois algumas ruas são de sentido único.

Dados de Saída

- Gf = (Vf,Ef) grafo dirigido pesado, tendo Vf e Ef os mesmos atributos que V e E.
- Cf —> sequência ordenada de todos os veículos usados, cada um com:
  - type —> tipo de serviço/trajeto que esse veículo fez (igual ao desse veículo no início do algoritmo) (na 1ª parte do problema, não vai ser considerado, porque será um veículo a fazer todos os trajetos).
  - D —> vetor ordenado com os IDs das arestas pelo qual o veículo terá de passar. Pode haver repetidos.
  - Tf —> vetor com os IDs dos trajetos/entregas que esse veículo fez.

Restrições

Restrições nos dados de entrada (Pré condições):

- ∀ i ∈ [1; T.size()] :
  - type(T[i]) = “BANK” v “MUSEUM” v “SHOP” —> o tipo de transporte/cliente terá de ser: agência bancária, ou museu, ou loja.
  - origem(T[i]) ∈ V —> a origem terá de ser um dos vértices do grafo.
  - destino(T[i]) ∈ V —> o destino terá de ser um dos vértices do grafo.
  - type(origem(T[i])) = type(destino(T[i])) = type(T[i]) —> será considerado apenas trajetos de museu para museu, banco para banco, etc. O tipo do vértice de destino terá de ser igual ao tipo do trajeto, bem como o tipo da origem. Esta restrição também assegura que a origem e o destino serão sempre ou um banco ou um museu ou uma loja, e não vértices normais.
  - origem(T[i]) ≠ destino(T[i]) —> os vértices de destino e origem terão de ser diferentes.
- ∀ V, type(V) = “BANK” v “MUSEUM” v “SHOP” v “NONE” —> para além de termos vértices a representar os possíveis clientes, também temos vértices “normais”.
- ∀ C ∈ Ci, type(C) = “BANK” v “MUSEUM” v “SHOP”, exceto na 1ª parte do problema, em que apenas é considerado um veículo para tudo.
- type(S) = “NONE” —> a central dos veículos será um vértice do tipo “NONE”.

- ∀ E, w(E) > 0 —> o peso das arestas terá de ser positivo uma vez que representam distâncias/tempos.

- ∀ E, E deve ser utilizável pelo(s) veículo(s). As arestas que não cumprirem isto serão removidas por um pré-processamento, e não serão incluídas no grafo G inicial.

- Seja L o conjunto de vértices constituído por S (central), todos as origens, e todos os destinos. É necessário que todos os elementos de L estejam no mesmo componente fortemente conexo do grafo (CFC).

  Deste modo, a partir de qualquer elemento do conjunto, é possível aceder qualquer outro, no grafo dirigido. Dado que todos os percursos começam e acabam no mesmo ponto (central), então todos os vértices do percurso devem pertencer ao mesmo CFC. Cada percurso então pode ser considerado como um CFC, sendo que como terão pelo menos um vértice em comum (central), irão originar um grande CFC, composto pela central, todas as origens, e todos os destinos (conjunto L).

Restrições nos dados de saída (Pós condições):

- Cf.size() <= Ci.size() —> obviamente, não será possível utilizar mais veículos do que os disponíveis no início.

- ∀ C ∈ Cf:

  - ∀ i ∈ [1; Tf.size()], type(C) = type (Tf[i]) —> um veículo de um dado tipo só pode fazer trajetos/entregas desse tipo (não aplicável à 1ª parte do projeto).

  - Seja di o primeiro elemento de D. É preciso que di ∈ Adj(S), uma vez que todos os percursos começam a sair da central dos veículos.

  - Seja df o último elemento de D. É necessário que dest(df) = S, uma vez que todos os percursos devem acabar por retornar à central dos veículos.

  - ∀ i ∈ [1; Tf.size()], ∃ d1, d2 ∈ D tal que (dest(d1) = origem(Tf[i])) ^ (dest(d2) = destino(Tf[i])) ^ (d1 < d2) —> se um trajeto foi incluído no vetor de trajetos desse veículo, significa que este concluiu o dado trajeto, ou seja, passou pelo menos uma vez na origem, e depois no destino do trajeto em causa. Por causa disto, o vetor de arestas pelo qual o veículo terá de passar terá de possuir pelo menos duas arestas, cujos destinos sejam a origem e o destino do trajeto, respetivamente, sendo que esta última terá de vir depois da primeira, no vetor ordenado (isto não elimina a possibilidade de se passar no destino antes da origem).

$$\min \sum_{c \in Cf} \sum_{d \in D} w(d)$$

—> os caminhos percorridos pelos veículos serão os que permitirem fazer as entregas na menor distância/tempo possível.

Função objetivo

A solução ótima do problema reside em, como já sabemos, encontrar as rotas ótimas para a entrega dos valores por parte dos veículos. Ou seja, queremos que eles terminem as suas tarefas/trajetos na menor distância/tempo total possível. Assim, a solução ótima irá passar pela minimização da soma das distâncias/tempos por todos os veículos utilizados:

$$\min \sum_{c \in Cf} \sum_{d \in D} w(d)$$

Perspetiva de solução

Para resolver o problema proposto, formulamos um plano com cinco etapas, sendo algumas delas “pré-etapas” auxiliares:

0) Cálculo dos valores da tabela auxiliar, que será utilizada para acelerar o algoritmo em si (só necessário da primeira vez que o algoritmo é executado);

1) Remoção das arestas indesejáveis para a execução do algoritmo:
   - 1ª iteração) remoção das que representam vias inutilizáveis pelos veículos (estas serão indicadas no grafo com peso = 0);
   - 2ª iteração) no caso de haver mais que uma aresta com a mesma origem e destino, remoção das que não têm peso mínimo (caso improvável, mas possível);

2) Verificar se os pontos de interesse (central, origens e destinos) pertencem todos a um comum Componente Fortemente Conexo;

3) Ordenação dos pontos de interesse (origens/destinos) a ser incluídos no(s) trajeto(s):
   - Numa etapa inicial, o algoritmo será apenas para um veículo, sendo que a ordenação será entre todas as origens e destinos em questão;
   - Posteriormente, passarão a poder ser utilizados vários veículos, sendo que será feita uma divisão das entregas por diferentes trajetos, cada um associado a um dos veículos (é tido em conta, também, a especialização dos veículos por tipos de cliente).

4) Sabendo já a ordem do(s) trajeto(s), calcular as arestas a serem percorridas, de modo a garantir a rota ótima para cada veículo, com o tempo mínimo.

Os **passos 0) e 1)** apenas teriam de ser executados quando se é inserido um novo grafo. O passo 2) ocorreria numa fase de pré-processamento, sendo que o algoritmo em si seria composto pelos passos 3) e 4).

Para o **segundo passo**, utilizaremos o algoritmo apresentado nas aulas teóricas para determinação dos CFCs de um grafo:

1. Pesquisa em profundidade no grafo G origina uma floresta de expansão, numerando os vértices em pós-ordem;
2. Inverter todas as arestas de G;
3. Segunda pesquisa em profundidade no grafo invertido, começando sempre pelo vértice não visitado de numeração mais alta;
4. Cada árvore obtida é um CFC;

5. Percorrer cada CFC até encontrar um que inclua todos os vértices do percurso (se não for encontrado nenhum, então não existe trajeto possível, logo o algoritmo não pode ser executado).

O **terceiro passo** poderia ser resolvido de várias formas, mas para o fazer em tempo útil, e sem utilizar uma quantidade desproporcional de recursos, foram concebidas duas possíveis abordagens, que determinam o tipo de dados que estarão contidos na tabela auxiliar:

Abordagem 1 – O peso de cada aresta representaria o tempo de viagem entre os dois vértices ligados por esta. Para além disso, cada vértice teria as suas coordenadas GPS, de modo a possibilitar o cálculo da distância diagonal entre dois dados pontos do mapa. Na tabela auxiliar seriam armazenadas as distâncias diagonais entre todos os pares de vértices. A ordenação das origens e destinos de cada trajeto iria ser feita da seguinte maneira:

- Começa-se por ter apenas a Central como ponto de partida e de chegada (percurso Central —> Central). A comparação das distâncias entre os pontos é feita da seguinte maneira:
  - Se houver um caminho direto entre os pontos em questão (os pontos são adjacentes), utiliza-se o valor da aresta que os liga como dado de comparação; caso contrário, acede-se à tabela auxiliar, e utiliza-se a distância diagonal entre os dois vértices;

- Escolhe-se um par origem-destino arbitrário. Para cada posição possível da sequência (entre os pontos de partida e de chegada, que devem manter-se), calcula-se o acréscimo de distância total do percurso, caso a origem seja aí inserida:
  - Exemplo: para inserir a Origem 2 (O2 daqui em diante) entre a Central (C) e O1 – calcula-se

    (dist(C, O2) + dist(O2, O1) – dist(C, O1) **)**

  - Caso particular: para a inserção do primeiro par origem-destino, não é necessário este cálculo, pois há apenas uma posição possível.
- A origem é adicionada na posição para a qual o acréscimo de distância/tempo total é menor;
- Para o destino correspondente, é feito o mesmo processo, mas apenas para as posições a seguir à sua origem.
- Também seria possível fazer da forma inversa (inserir primeiro o destino, e inserir a origem da mesma maneira, numa posição anterior ao destino), sendo que uma hipótese seria calcular ambas as possibilidades, que seguem uma heurística diferente, e eleger a que melhor cumpre a função objetivo.

Abordagem 2 – Para a abordagem 2 o processo é semelhante ao da abordagem 1. No entanto, neste caso, não seria utilizado o conceito de coordenadas GPS para os pontos do mapa. Em vez disso, será armazenado na tabela o tempo de viagem mínimo entre cada par ordenado de dois vértices do grafo (obtido através das arestas, que representam tempos), bem como, para cada par ordenado de vértices, a última aresta a ser percorrida no caminho entre eles, de modo a saber que arestas é que fazem parte do caminho que origina esse tempo mínimo. Há duas maneiras de fazer esses cálculos:

- Se o grafo for esparso (**$|E| = O(V)$**), será preferível utilizar o algoritmo de Dijkstra para cada um dos vértices do grafo (**$O(|V| * |E| * \log(V))$**)
- Se o grafo for mais denso (**$|E| = O(V^2)$**), será mais adequado usar o algoritmo de Floyd-Warshall (**$O(V^3)$**)

No cálculo das distâncias entre os pontos, utiliza-se sempre os valores armazenados na tabela, dado que temos o valor calculado para qualquer par de vértices. Apesar de exigir mais na etapa de preparação, esta abordagem trabalharia sempre com valores certos, e não com aproximações, como é o caso das distâncias diagonais pelas coordenadas GPS da abordagem 1. Para além disso, as arestas a serem percorridas para cada veículo poderão ser simplesmente obtidas através da tabela, sendo que não será necessário repetir esses cálculos.

Na segunda etapa do terceiro passo, será necessário efetuar a divisão das entregas em percursos, possivelmente para mais que um veículo, de modo a melhor cumprir a função objetivo. Aqui, cabe-nos decidir o que é mais importante:

- minimizar o tempo total de viagem (e assim, os custos de operação para a empresa);
- efetuar todas as entregas no menor tempo possível.

Embora este segundo ponto seja importante, além de não ser tão interessante resolver o problema desta maneira (afinal, a solução ótima seria sempre enviar um carro a cada origem, e depois ao destino correspondente), a solução resultante não faria sentido no contexto do problema, já que desperdiçar recursos não faz, de todo, parte de uma estratégia de negócio ideal. Assim, apresenta-se o problema de minimizar a distância/ tempo total percorrida pelos carros.

Para cada entrega:

- verifica-se qual o tipo de entrega (banco, museu, loja, …);
- identifica-se os veículos correspondentes a esse tipo (são os veículos candidatos a fazer essa entrega);
- para cada veículo candidato, tenta-se inserir a origem na sua sequência de trajetos, pelo método apresentado na abordagem 1 ou 2; o acréscimo da distância total para esse veículo será **deltaOrigem**; insere-se também o destino dessa entrega na sequência, cujo acréscimo será **deltaDestino**;

- a entrega será atribuída ao veículo cuja soma **deltaOrigem + deltaDestino** assuma o valor mínimo; naturalmente, a origem e o destino serão inseridos nas posições que resultem neste acréscimo mínimo, segundo as abordagens 1 e 2.

Este método permite atribuir uma dada entrega a um veículo que já tenha um percurso definido, ou a um veículo que ainda não tem percurso, e irá sair da central (irá escolher o que compensar mais).

Esta solução garantirá um balanço positivo na grande maioria dos casos, poupando, simultaneamente, muito tempo de processamento para o cálculo de novos percursos (em comparação, por exemplo, a uma resolução do tipo *brute-force*).

No **quarto passo** é necessário, para a sequência do trajeto de cada veículo, calcular o caminho mais curto entre cada par de vértices consecutivos. Tal pode ser feito de várias maneiras:

- aplicação do algoritmo de Dijkstra, com começo no primeiro vértice do par, parando o algoritmo quando for processado o segundo vértice do par;
- aplicação do Dijkstra bidirecional entre os dois vértices do par (pode haver ganhos temporais, mas devido à necessidade de computar o grafo invertido seria ocupada mais memória, e o tempo de pré-processamento poderia invalidar a vantagem temporal);
- entre outros…

Independentemente do algoritmo aplicado, iremos ter as arestas que o veículo deverá percorrer para seguir a rota ótima. Este conjunto de arestas irá ser retornado como dado de saída do algoritmo.

NOTA: Caso seja adoptada a abordagem 2 no terceiro passo do algoritmo, este quarto passo não será necessário, uma vez que as arestas a percorrer já estão calculadas, e presentes na tabela auxiliar.

Casos de utilização

- Importação de um grafo a partir de um ficheiro de texto;
- Visualização do grafo e da informação que este representa (vias de trânsito e pontos de interesse numa cidade) no GraphViewer;
- Importação dos veículos disponíveis, bem como o tipo de transporte que cada um efetua, a partir de um ficheiro de texto;
- O utilizador tem a possibilidade de escolher os pontos de interesse para a execução do algoritmo (origens e destinos, verifica a validade dos pontos escolhidos);
- O utilizador tem a possibilidade de exigir, ou não, a especialização dos veículos para o seu trajeto (escolha entre vários veículos para tudo ou vários veículos especializados);
- Depois de calculadas as rotas ótimas para cada veículo, mostrar as arestas escolhidas e os vértices por onde passam no GraphViewer (para cada veículo, uma cor diferente, por exemplo).

Conclusão

Desde cedo percebemos que existem diversas maneiras de tentar resolver este tipo de problemas. Após uma cuidada análise do enunciado do tema e da matéria das aulas teóricas, bem como aconselhamento por parte do professor das aulas práticas e dos monitores, achamos que conseguimos estruturar um bom plano, que leva a uma boa solução ao problema da identificação das rotas ótimas.

A nosso ver, a parte mais exigente e difícil do trabalho foi a determinação de uma estratégia que levasse à ordenação das origens e destinos por que cada veículo tem de passar. Como não há nenhum algoritmo que faça isto diretamente, definimos nós próprios uma estratégia de ordenação dos pontos de interesse.

Também achamos desafiador conceber a parte do algoritmo que decide que trajetos é que devem ser feitos por que veículos.

De resto, não foram encontradas mais nenhumas dificuldades de grande calibre. Somos da opinião que com este trabalho, dominamos agora a matéria que incide na manipulação de grafos com algoritmos, pelo que nos encontramos bem preparados para a implementação desta ideia na 2ª parte do trabalho.

Diogo Machado - up201706832@fe.up.pt

Tarefas:
- descrição do tema;
- identificação e formalização do problema (dados de entrada e saída, restrições);
- perspetiva de solução (principalmente passos 3 e 4);

Esforço Dedicado: 1 / 3

Eduardo Ribeiro - up2017054212@fe.up.pt

Tarefas:
- descrição do tema;
- identificação e formalização do problema (dados de entrada e saída, restrições);
- perspetiva de solução (principalmente passos 0, 1, 2 e 4);
- casos de utilização;

Esforço Dedicado: 1 / 3

Eduardo Macedo - up201703658@fe.up.pt

Tarefas:
- descrição do tema;
- identificação e formalização do problema (restrições, função objetivo);
- perspetiva de solução (principalmente passos 0, 1 e 4);
- casos de utilização;

Esforço Dedicado: 1 / 3

Bibliografia Utilizada: slides das aulas teóricas.

Relatório CAL - 2ª Parte Trabalho Prático
11/4/2019

Diferenças entre o planeamento da 1ª parte e a respetiva implementação final

Como seria de esperar, à medida que foi implementada a nossa visão do projeto, percebemos que existiam certos aspetos relativos ao nosso planeamento que teriam de ser modificados, por diversas razões.

Primeiramente, foi necessária a adaptação da nossa visão e implementação, devido aos grafos que nos foram disponibilizados pelos monitores: estes contém arestas bidirecionais, e não unidirecionais, como era esperado.

Seguidamente, acrescentamos mais alguns algoritmos e funcionalidades ao nosso programa, que achamos pertinentes no contexto do problema.

Em suma, o trabalho disponibiliza as seguintes opções:

- Leitura e carregamento de um grafo a partir de um ficheiro de texto;
- Leitura e carregamento de listagens de veículos e entregas, bem como a identificação do vértice do grafo que representa a central (devem ser especificados pelos utilizadores);
- Cálculo de uma tabela contendo a distância e o último vértice no caminho entre todos os pares de vértices acessíveis desde a central (feito com o algoritmo de Dijkstra ou com Floyd-Warshall; a escolha é do utilizador);
- Apresentação no GraphViewer do grafo total da região;

- Apresentação no GraphViewer do grafo constituído pelos vértices acessíveis a partir da central dos veículos (através de uma pesquisa em profundidade começando na central);

- Aplicação de um algoritmo de deteção de pontos de articulação, e identificação no GraphViewer dos mesmos (contribuí para a avaliação da conectividade do grafo);

- Cálculo das rotas ótimas para os veículos, tendo em conta as entregas que devem ser feitas e o tipo de veículos e entregas que estão definidos;

- Apresentação no GraphViewer dos resultados, sendo dada a opção de display dos resultados para todos os veículos, ou apenas para um grupo definido pelo utilizador.

Relativamente às heurísticas definidas na 1ª parte para calcular as rotas ótimas para os veículos, estas foram mantidas na implementação.

Discussão sobre estruturas de dados utilizadas

Tabela

A principal estrutura de dados utilizada no nosso projeto, gerada num pré-processamento de uma parte do grafo total dado, é a tabela referida acima, que contém a distância e o último vértice (path) no caminho entre todos os pares de vértices. A informação da tabela é calculada numa fase inicial, quando o utilizador indicar o vértice onde a central dos veículos se situa. Optamos pela utilização desta tabela pois, embora a fase de pré-processamento seja um bocado mais demorada, as distâncias e os paths nunca mais precisam de ser calculados (desde que não se mude a central, o que consideramos um acontecimento pouco comum). Para o cálculo das rotas ótimas para as entregas futuras que venham a ser processadas, basta aceder, em tempo logarítmico, aos conteúdos da tabela.

A estrutura de dados utilizada para a tabela é:

```
map< pair<Vertex<Node>*, Vertex<Node>*>, pair<double, Vertex<Node>*> >
```

Ou seja, para cada par de vértices, haverá um par que indica a distância entre os dois, representada por um double, e o path (vértice anterior no caminho), representado pelo endereço do vértice.

Inicialmente, no nosso programa, a tabela era gerada para todos os vértices do grafo total. Embora isso possa trazer algumas vantagens, sendo um pré-processamento mais abrangente, chegamos a conclusão que era demasiado demorado e pouco eficiente, pelo que optamos então por gerar a tabela para apenas os vértices acessíveis a partir da central.

Outra estratégia que tentamos implementar foi a não repetição de entradas na tabela (ex: se o par 1 - 2 já está na tabela, não meter o par 2 - 1). Para determinar a distância de um par não presente na tabela, apenas era necessário pesquisar a distância para o par invertido (como os grafos são bidirecionais, a distância entre 1 - 2 é igual a 2 - 1).

Porém, para calcular o path, era necessário uma função mais complexa e recursiva, o que levava a perdas significativas de performance ao calcular o trajeto dos veículos. Por isso, apesar de fazer com que a tabela ocupe mais memória, optamos por guardar todos os pares na mesma.

Graph, Vertex e Edge

As classes fornecidas nas aulas que representam um grafo, um vértice e uma aresta também foram utilizadas neste trabalho. A maior parte do funcionamento das mesmas não mudou, mas foram acrescentados alguns métodos que achamos conveniente ter.

Node

Uma vez que queríamos guardar várias informações em cada vértice do grafo, definimos uma classe Node, que contém um ID específico do vértice, as coordenadas X e Y do mesmo, o tipo de vértice (uma enumeração definida por nós) e as coordenadas X e Y de display (para o GraphViewer). O grafo utilizado é, então, um Graph<Node>.

```
enum TYPE { BANK,
            FIN_ADVICE,
            ATM,
            TAX_ADVISOR,
            AUDIT,
            MONEY_MOV,
            OTHER,
            CENTRAL };

class Node {

private:
    /**
     * ID do node.
     */
    int id;

    /**
     * Coordenada X do node.
     */
    double xCoord;

    /**
     * Coordenada Y do node.
     */
    double yCoord;

    /**
     * Tipo do node (BANK, MUSEUM, SHOP, etc)
     */
    TYPE type;

    /**
     * Coordenada X a ser utilizada para fazer display do nó no GraphViewer.
     */
    int displayX = 0;

    /**
     * Coordenada Y a ser utilizada para fazer display do nó no GraphViewer.
     */
    int displayY = 0;
```

Delivery

Para representar as entregas que têm de ser feitas, definimos uma classe Delivery, com alguns atributos e métodos de acesso aos mesmos. É mantido um vetor de deliveries, de modo a saber quais as entregas que têm de ser feitas nesse momento.

```
class Delivery {

private:

    /**
     * ID da entrega.
     */
    int id;

    /**
     * Tipo da entrega.
     */
    TYPE type;

    /**
     * Vertice origem da entrega.
     */
    Vertex<Node>* origem;

    /**
     * Vertice destino da entrega.
     */
    Vertex<Node>* destino;
```

Vehicle

Para representar os veículos que estão disponíveis num determinado momento, foi criada a classe Vehicle, que contém o ID específico do veículo e o tipo do mesmo.

Contém também um vetor de vértices, que representa os vértices pelos quais o veículo terá de passar para cumprir o seu trajeto (ex: se a central for 4, e o veículo tiver de fazer uma entrega de 2 a 3, o vetor será {4, 2, 3, 4}). Para além disso, armazena um vetor com todas as entregas que este irá fazer no seu percurso.

Para além das típicas funções get e set, tem uma função que indica qual a melhor posição para inserir a origem e o destino de uma entrega no vetor de vértices dele, bem como o acréscimo de distância que este

Análise da complexidade temporal dos algoritmos utilizados

NOTAS E OBSERVAÇÕES

O programa, para além de fazer a leitura de ficheiros dados pelos monitores relativos ao grafo, lê também ficheiros em que estão especificados os veículos e as entregas; estes têm de ser definidos pelo utilizador antes de correr o programa ou antes de selecionar a opção de leitura dos mesmos.

Nos ficheiros enviados na submissão, apenas estão presentes dados relativos a Aveiro. Para testar em outros grafos, é necessário fazer novos ficheiros relativos à nova cidade/país.

Juntamente com os grafos disponibilizados, encontra-se uma pasta com um grafo de Teste, significativamente mais pequeno e simples, que foi utilizado à medida que o trabalho foi desenvolvido, para testar o correto funcionamento das funcionalidades do mesmo.

Os ficheiros de informação relativa à latitude e longitude não foram utilizados no nosso programa, uma vez que não achamos necessário.

Cotações

Diogo Machado (up201706832):

- Estruturação dos menus do programa, e interface com o utilizador;
- Adaptação e implementação das estratégias definidas para o cálculo de rotas ótimas dos veículos;
- Configuração do display dos resultados calculados no GraphViewer;
- Etc.

Cotação: 30%

Eduardo Ribeiro (up201705421):

- Estruturação dos menus do programa, e interface com o utilizador;
- Adaptação e implementação das estratégias definidas para o cálculo de rotas ótimas dos veículos;
- Implementação do algoritmo de cálculo dos pontos de articulação;
- Definição da estrutura e leitura dos ficheiros de texto utilizados para a importação de dados para a aplicação;
- Etc.

Cotação: 40%

Eduardo Macedo (up201703658):

- Definição da estrutura e leitura dos ficheiros de texto utilizados para a importação de dados para a aplicação;

- Configuração do display dos resultados calculados no GraphViewer;
- Deteção e teste das funcionalidades do programa;
- Etc.

Cotação: 30%