U.PORTO

FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

InspectIT: Rotas de Inspeção

Tema 10

Conceção e Análise de Algoritmos

Grupo 1 – Turma 7

Mestrado Integrado em Engenharia Informática e Computação

Luísa Maria Mesquita Correia	up201704805
Flávio Lobo Vaz	up201509918
Amanda de Oliveira Silva	up201800698

Índice

NOTA: A primeira entrega corresponde aos pontos 1 a 3 inclusive, enquanto que a segunda entrega corresponde aos restantes pontos, a começar no 4.

1. Introdução

O nosso tema está relacionado com inspecções alimentares e económicas por parte de uma autoridade pública, a vários tipos de atividades económicas (e.g. ramo alimentar, têxtil). No que resultará num elevado número de agentes económicos (e.g. restaurantes, talhos, supermercados, fábricas, lojas, entre outros), com um ou mais tipos de atividade económica.

Este trabalho consiste em implementar um sistema que permita a identificação de rotas de inspeção ótimas para as várias brigadas da autoridade pública, de modo a que, a utilidade das inspeções seja maximizada, visto que o número de agentes económicos a serem inspecionados, por norma será muito maior que o número de brigadas.

Por questões de simplicidade, a autoridade atuará apenas na região do Porto, sendo que a cada brigada será atribuído uma área do Porto.

Iremos considerar também que as brigadas tanto podem ter uma especialidade como várias, à escolha do utilizador.

O processo de definição de rotas óptimas pretendidas será efectuado, de uma forma geral, da seguinte forma:

1. Uma função de seleção irá selecionar por ordem os agentes económicos a ser inspecionados tendo em conta, o número de queixas muito graves (e.g. problemas susceptíveis de terem um grande impacto na saúde das pessoas), número de inspeções insatisfatórias e o número total de queixas, priorizando pela ordem apresentada.

2. Estando o ponto 1 estabelecido, calcularemos o tempo/distância de viagem óptimos entre cada agente económico, dois a dois, o que permite obter as melhores rotas entre agentes económicos e o ponto de partida, que é a autoridade pública.

3. Tendo em conta o horário de funcionamento de cada agente económico, o horário máximo de inspeção diário de cada brigada (horário de trabalho diário), a especialidade ou ausência da mesma e o tempo de inspeção num agente económico, as brigadas serão distribuídas por vários agentes económicos, onde iremos obter as melhores rotas para cada brigada segundo objetivos pré-definidos neste relatório.

Serão implementadas duas soluções para o "cálculo" da rota óptima, o qual será o utilizador a decidir qual delas quer:

1. A rota prioriza inspeções urgentes a fazer (e.g. alimentação)
2. A rota prioriza visitar o maior número de estabelecimentos num dia.

O percurso efectuado por uma brigada entre os agentes económicos irá assegurar que existem caminhos entre cada agente económico atribuído numa possível rota ótima e chegando ao último agente económico, voltar para o ponto de partida, ou seja, a entidade pública.

É importante salientar, que como referido no enunciado do nosso trabalho, iremos considerar problemas (e.g. obras na via pública), que impeçam de aceder a alguma zona do mapa da área geográfica, e que por sua vez possam impedir a inspeção de algum agente económico. Identificados esses agentes económicos, não será efetuada a sua inspeção.

De seguida, apresentam-se os seguintes tipos de atividades económicas (1), os quais os agentes se integram em pelo menos um deles. Por exemplo, um restaurante integra-se na atividade de Géneros Alimentícios se a denúncia estiver relacionada com alimentos estragados. Ou então, esse mesmo restaurante pode integrar-se na atividade Geral e de Atividade Comercial se a denúncia estiver relacionada com os impostos. Nota: poderão ser adicionados mais à medida que o trabalho for implementado.

- Obras
- Atividade Comercial
- Ambiental e Intervenção na Via Pública
- Segurança e Salubridade de Edificações
- Géneros Alimentícios (e.g. talhos, restauração...)

Por fim, é importante também referir que o tempo de inspeção a cada estabelecimento será calculado segundo uma função que combina o tipo da fiscalização, a área do estabelecimento e outros fatores relevantes.

2. Formalização do Problema

Agentes económicos operantes estão suscetíveis de receber denúncias quanto ao seu mau funcionamento. Cabe assim à entidade pública de inspeções direcioná-las, com o objectivo de:

- denúncias urgentes serem priorizadas de forma a que se fiscalize o máximo de agentes económicos possíveis e tendo em conta denúncias e gravidade das mesmas
  ou
- maximizar o número de inspeções feitas num dia

2.1. Dados de Entrada

- **T** : Tabela onde guardaremos os tempos/distâncias mínimos de viagem entre cada par de vértices e arestas que levam a cada vértice para os tempos/distâncias calculados.
- **Bi** : Brigada de índice i disponível para fazer inspeções numa determinada área económica (ou qualquer área como será abordado no ...)
- **TAE**: estrutura onde guardaremos todos os tipos de atividade económica em que uma brigada pode ser especializada ou em que um agente económico opera.

- Cada brigada é caracterizada por:
  - **AEi** : conjunto de atividades económicas que são especialidade de uma brigada (poderá ser uma ou várias como será explicado nos casos a serem explorados) sendo i o i-ésimo elemento.
  - **HI** : Número de horas diário total de trabalho para efetuar inspeções e viagens inerentes às mesmas.
  - **ID** : um valor único e identificativo para cada brigada.
  - **HIB :** hora do dia em que brigada começa a sua atividade**.**

- **ACi** : agentes económicos de índice i que têm de ser inspecionados. Cada agente económico é caracterizado por:

- **AEi** : conjunto de atividades económicas de um agente económico, sendo i o i-ésimo elemento.
- **ID** : um valor único e identificativo para cada agente económico.
- **AM :** Área em metros quadrados (servirá para calcular **TI**)
- **HF** : horário de funcionamento.
- **TI** : Tempo de Inspeção (usaremos uma função para determinar este parâmetro).
- **QG** : Número de queixas muito graves.
- **TQ :** Número total de queixas.
- **IC** : Número de inspeções correctas.
- **IF** :Número de inspeções falhadas.
- **UI**: Data da última inspeção

- **G = (V, E)** : grafo dirigido pesado, representando o mapa da cidade. Este grafo irá ser constituído por vértices (V) e arestas (E). Consideramos ser dirigido pois podem existir pares de vértices com apenas um sentido (e.g. estradas de sentido único) e pesado pois as arestas entre pares de vértices terão um peso associado.

  - Cada **vértice** (representa pontos de importância no mapa: possíveis agentes económicos a inspecionar, interseções entre vias de trânsito, entre outros) é caracterizado por:
    - **GPS**: coordenadas GPS do ponto no mapa
    - **tipo** : identifica se o vértice representa um agente económico, a autoridade pública de inspeção ou outra entidade importante do mapa utilizado.
    - **Adj** $\subseteq$ E : Conjunto de arestas que saem desse vértice para um outro.
    - **VAP** $\in$ V : vértice inicial que representa a autoridade pública de onde as brigadas saem para fazer inspeções a agentes económicos e que no final de fazer todas as inspeções possíveis num dia têm de regressar a este.

  - Cada **aresta** (representa ruas, pontes e outras vias do mapa) é caracterizada por:
    - **PA** : peso da aresta, que representa o tempo/distância de viagem entre os dois vértices que a aresta liga.
    - **ID** : identificação única para cada aresta;
    - **Dest** $\in$ V : vértice de destino da aresta;

2.2. Dados de Saída

- **G = (Vs,Es)** : grafo dirigido e pesado, em que Vs e Es são caracterizados da mesma forma que V e E.
    - **Bs** : sequência ordenada de brigadas pela ordem de utilização, onde s é a s-ésima brigada, em que teremos para cada uma:
        - **EID** : vetor com todos os IDs dos vértices ordenados pela ordem de passagem dos mesmas pela brigada em questão, com possibilidade de repetição (e.g. podemos ter que passar mais que uma vez pela mesma rua)
        - **AEID** : vetor com os IDs dos agentes económicos (vértices) ordenados pela ordem com que foram inspecionados pela brigada num dia e aresta respectiva que tem como valor **Dest** o agente económico respectivo.
        - **TMP: t**empo de trabalho e viagens total utilizado nas inspeções referentes aos agentes económicos em **AEID** e rotas representadas pelos vértices eem **EID**.

2.3. Restrições

2.3.1 Restrições Referentes aos Dados de Entrada:

- Seja |Bi|, o número de brigadas existentes na autoridade pública, $\forall i \in [1,|Bi|]$, temos para cada brigada Bi :
    - $|AEi| = 1 \ V \ \ |AEi| = |TAE|$ (só uma especialidade num tipo de atividade económica ou brigada especializada em todas)
    - $0 < HI \leq 8$ (para HI = 0 horas, brigada está indisponível)

- As atividades económicas em que uma brigada pode ser especializada são:
    - Obras
    - Atividade Comercial
    - Ambiental e Intervenção na Via Pública
    - Segurança e Salubridade de Edificações
    - Géneros Alimentícios (e.g. talhos, restauração...)

- Seja M, o número de agentes económicos existentes para serem inspecionados pela autoridade pública, $\forall i \in [1, M]$ temos que para cada agente económico ACi:
    - $|TAE| \geq |AEi| \geq 1$

- TI ≥ 0,5 (tempo em horas que consideramos minimamente plausível para uma inspeção)
- QG ≥ 0
- IC ≥ 0
- IF ≥ 0

- Seja |V| o número de vértices do nosso grafo G = (V, E), ∀ i ,e ∈ [1, |V|], temos que para cada vértice vi ,ve ∈ V,
  - **tipo**: os tipos possíveis de vi serão: AE (agente económico), AU (autoridade pública), OZM (outra zona do mapa).
  - Existe um só vértice vi que corresponde à autoridade pública, a origem do grafo.
  - Todos os agentes económicos, para serem inspecionados têm de ter um só vértice vi correspondente no grafo.
  - Seja o vértice vi correspondente à origem (autoridade pública) e seja qualquer vértice ve correspondente a um qualquer agente económico ou outra zona do mapa, temos então que vi ≠ ve.

- Seja |E| o número de arestas do nosso grafo G = (V, E), ∀ i ,t ∈ [1, |E|], temos que para cada aresta ei, et ∈ E:
  - PA (ei) > 0 : o peso de cada aresta tem de ser positivo visto que representam tempo/distância de viagem entre pares de vértices.
  - et tem de ser uma aresta possível de ser utilizada por qualquer brigada.
  - Se et for uma aresta impossível de utilizar, a mesma será retirada do grafo, de forma a garantir que posteriormente identificamos algum agente económico com acessibilidade reduzida.

- Outras restrições:
  - **HF** tem de ser definido tentando que seja o mais realista possível.
  - Cada brigada só faz inspeções num tipo de atividade económica ou a todos os tipos de atividade económica.
  - Em relação à conectividade do nosso grafo: Suponhamos que seja k o vértice que representa a origem (autoridade pública) e C o conjunto de vértices que representam cada Agente Económico e com k incluído, temos que garantir que ∀c ∈ C, c existe na mesma componente fortemente conexa do nosso Grafo. Sendo

k o ponto de onde todas as rotas começam e acabam, toda as rotas de cada brigada pelos agentes económicos têm um ponto em comum (vértice referente à autoridade pública). Sendo assim, pela definição de componente fortemente conexa de um grafo, uma rota será por si só uma componente fortemente conexa, e tendo um ponto em comum com todas as outras rotas como referido, as mesmas têm que pertencer à mesma componente fortemente conexa do grafo. Ou seja, o conjunto de componentes fortementes conexas referente a cada rota de uma brigada, seja este RCFC, é também uma componente fortemente conexa que tem todos os vértices referentes aos agentes económicos e a autoridade pública. Conclusão: ∀c1, c2 ∈ C com c1 ≠ c2 onde lc1, c2 ∈ RCFC, temos que partindo do vértice c1 conseguimos aceder outro qualquer vértice c2 no grafo dirigido.

2.3.2. Restrições referentes aos Dados de Saída:

- |**Bi**| ≥ |**Bs**| : número de brigadas nos dados de saída, tem de ser menor ou igual ao número de brigadas iniciais nos dados de entrada.

- ∀ i ∈ [1, |AEi|] e t ∈ [1, |AEt|] onde AEi e AEt, são os conjuntos de atividades económicas que são especialidade de uma brigada e o tipo de atividade económica de um agente económico respectivamente, sendo i e t o i-ésimo e t-ésimo elemento. Temos que para cada brigada **Bs** com a sua rota de inspeção e todos os agentes económicos que foram inspecionados nesta rota, existe pelo menos um elemento AEi tal que AEi = AEt, onde AEt um elemento qualquer, que identifica uma das atividades económicas de um agente económico inspecionado pela brigada. (No caso em que uma brigada pode inspecionar qualquer agente económico, isso significa basicamente que é especializado em todos os tipos de atividade económica).

- Em relação ao vetor de vértices **EID** de cada brigada **Bs** :
  - Seja ci, cf, o primeiro elemento e último elemento do vetor **EID** respectivamente**,** ci = **VAP**, **Dest** o vértice de destino cf, temos que: **Dest** = **VAP,** já que todas as rotas começam e acabam obrigatoriamente na autoridade pública (**VAP**) .
  - Qualquer rota atribuída a uma qualquer brigada **Bs,** definida pelo vetor de vértices **EID** necessita de ter pelo menos duas arestas**.** Pois, num caso limite com duas arestas, o primeiro elemento de **EID,** ci ∈ **V (VAP) e o último cf= Dest = VAP**, que é um caso de uma rota onde apenas um agente económico é inspecionado voltando a brigada para a autoridade pública de seguida.

- Em relação às arestas correspondentes às rotas representadas pelos vértices em **EID** de cada brigada **Bs** :
  - Temos que ∀ **c** ∈ **EID** e a aresta ac ∈ Adj de c, o seu tempo de viagem em **PA** para um determinado agente **económico p**∈ **AEID,** a soma dos tempos de viagem com os respectivos tempos de inspeção do agente económico **p** obtidos em **AEID** para cada brigada Bs, não ultrapassam o valor **HI** da mesma**.**
  - Tendo em conta uma hora inicial do funcionamento da autoridade pública, temos de garantir que ao inspecionar um agente económico, na altura em que a brigada **Bs** chega a esse agente económico para o inspecionar, o começo da inspeção cumpre o horário de funcionamento do mesmo.

  - ∀ **p** ∈ **AEID,** agente económico, é visitado por apenas e só uma brigada que tenha a especialidade necessária para a atividade ou atividades económicas do mesmo (excepto quando propositadamente por motivos didáticos ignoramos as especialidades).

2.4. Função Objectivo

Tendo em conta que se trata de uma autoridade pública o objectivo da mesma é salvaguardar o bem estar dos consumidores e a qualidade dos vários tipos de atividade económica. Portanto faz todo o sentido que no nosso problema usemos todas as brigadas disponíveis para efetuar inspeções diariamente segundo objectivos que mais se adequem às suas especialidades. Portanto, temos dois objetivos para abordar o nosso problema:

1. Obter rotas óptimas (tempo/distância) de inspeção onde se pretende sempre visitar primeiro os agentes económicos que são classificados como mais urgentes. Tal será implementado através de uma função (explicada no ponto 3), utilizando o maior número de brigadas possível.

   Assim, as rotas óptimas serão aquelas onde se efectuam inspecções na menor distância/tempo possível desde a autoridade pública aos vários agentes económicos até se esgotar o **HI** diário de cada brigada **Bs**, em que os tempos de funcionamento **HF** de cada agente económico são respeitados. Também teremos em conta os **TI** dos mesmos. Ora, como certos agentes económicos terão prioridade sobre outros (por várias razões já mencionadas) e portanto aparecerão no vector **AEID** por ordem de prioridade.

   Concluindo, as funções objectivo para este cenário serão:

   Seja **Bs** conjunto de todas as brigadas utilizadas e |Bs| o número de brigadas utilizadas:

   $$\text{Max } f = |Bs|$$

   $$\text{Max } t\,(AE) \quad \forall\, AE \in \textbf{ACi}$$ **(função que define urgência de inspeções dos agentes económicos)**

   $$\text{Min } h = \sum_{b \in \textbf{Bs}} \sum_{e \in \textbf{EID}} PA(e)$$

2. O segundo cenário terá como objectivo apenas obter o maior número de inspeções possível num dia (sem haver prioridades entre agentes económicos), utilizando o maior número de brigadas possível.

   A função objectivo é:

   Seja **Bs** conjunto de todas as brigadas utilizadas:

$$\text{Max } f = |Bs|$$

$$\text{Min } h = \sum_{b \in \mathbf{Bs}} \sum_{e \in \mathbf{EID}} PA(e)$$

(Onde se tenta esgotar o tempo **HI** de cada brigada, onde se respeitam os tempos de funcionamento **HF** de cada agente económico e onde se contabilizam também os **TI** dos mesmos.)

É de salientar que em ambos os casos, a função f deve ser vista como prioritária a ser maximizada em relação à função h, visto que queremos sempre utilizar o maior número de brigadas pelos motivos já enumerados e no **caso 1 a função t** também deve ser vista como prioritária a ser maximizada (optar sempre pelos agentes económicos mais urgentes de ser inspecionados).

3. Perspectiva de Solução

A asserção a nós dirigida assemelha-se muito ao problema do caixeiro-viajante (**2**) e é um problema do tipo NP-hard (**3**). Neste problema, ao iniciar uma rota em determinado vértice e após passar pelos demais vértices orientados, deve-se voltar para a origem (com a rota traçada a fim de reduzir o tempo/distância necessário para a viagem e os possíveis custos com transporte e combustível). Contudo, para o nosso trabalho, o foco será obter tempos óptimos de viagem desde a origem, passando por vários agentes económicos tentando sempre que para cada brigada **Bs** se utilize totalmente o **HI**, respeitando o **HF** de cada agente económico ao longo da rota e que os **TI** dos mesmos também possam ser contabilizados e analisados, e usados como critério de escolha se assim quisermos, visto que por exemplo uma agente económico com um tempo necessário de inspeção muito elevado pode ocupar demasiado tempo a uma brigada e assim prejudicar o número total de agentes económicos inspecionados na rota mas também no total. Já que um dos objectivos pode ser maximizar o número total de agentes económicos inspecionados.

De forma geral, o objetivo será obter os ciclos (rotas) hamiltonianos (4) mais eficientes num dado grafo (o grafo hamiltoniano), ou seja, a rota começa e termina no mesmo vértice de origem (autoridade pública) passando pelos vários agentes económicos pelo menos uma vez, e tendo em conta as restrições do nosso problema. Isto vai também ao encontro do que foi referido em relação a componentes fortemente conexos.

No programa, **não consideramos a parte da rota em que cada brigada retorna à autoridade pública** (apesar de nos certificarmos que isso é possível), visto que a mesma não é contabilizada como tempo de trabalho das brigadas, sendo assim redundante para o que pretendemos.

Portanto, o nosso plano será constituído por várias etapas e possível pré-processamento de informação (ou não) para utilizações futuras do programa que iremos implementar.

O factor **tempo de viagem** entre cada par de vértices (e assim também entre cada agente económico e ou autoridade pública entre outras zonas do mapa), do grafo que representa o mapa das zonas consideradas, mas também o **tempo de inspeção** de cada agente económico agentes serão o mais preponderante no nosso problema.

Contudo poderemos equacionar posteriormente uma implementação e resolução do problema, que também utilize as coordenadas GPS para distribuir brigadas pelos agentes económicos, de forma a concentrar as inspeções nos agentes económicos que estão numa zona limitada constituída por um local (centro

de uma circunferência com longitude e latitude) e um **raio de ação** escolhido por nós.

Também será implementado uma forma **"híbrida"** de abordar os nossos objetivos, será possível utilizar ou não o que foi em cima descrito referente a **um raio de ação**, **contabilizar o tempo de inspeção ou não** antes de equacionar se um dado agente económico deve ou não fazer parte de uma determinada rota , introduzir o factor de urgência como restrição adicional e não como principal factor .

Numa primeira fase iremos considerar que a autoridade terá apenas uma brigada de inspeção que pode inspecionar qualquer tipo de atividade económica.

Numa segunda fase consideraremos que a autoridade pública terá várias brigadas em que cada uma tenha apenas uma especialidade ou alguma com todas as especialidades.

3.1. Pré-Processamento

Processar tabela com tempos de viagem mínimos entre cada par de vértices, o que permitirá obter os vetores **AEID** e **EID** referidos nos dados de saída e que nos permitem saber os agentes económicos inspecionados os vértices visitados na rota entre todos os agentes económicos, de forma ordenada.As arestas terão como peso (**PA)** o tempo de viagem entre cada vértice do grafo que será obtido utilizando as coordenadas de latitude e longitude de cada vértice para distâncias (obtendo assim distâncias reais) e uma velocidade média (em Km/h) escolhida por nós para todas as brigadas e que deve ser realista para um contexto rodoviário urbano (por exemplo velocidade média das brigadas com valores entre 10 a 50 Km/h).

Este pré-processamento apesar de requerer algum tempo e memória considerável, posteriormente permite acelerar a obtenção das rotas óptimas. Visto que é feito apenas uma vez , onde os valores da tabela são reutilizáveis.

Algoritmos a utilizar:

1. Se o grafo for esparso (5), tendo em conta o que foi dado nas aulas teóricas, achamos que o algoritmo a utilizar será **Dijkstra** (6) que no limite usando uma **Fibonacci Heap** (7), terá complexidade O(|E| + |V| * log(V)) e complexidade espacial O(|V|), para cada um dos vértices do grafo.
2. Se o grafo for denso (8), consideramos ser melhor utilizar o algoritmo Floyd-Warshall com complexidade O (V^3) e complexidade espacial
3. Utilizar algoritmo Bellman Ford, que tem complexidade temporal O(|V| |E|) complexidade espacial O(V), por uma questão de comparação, mas também porque pode ser interessante noutro contexto, em que temos algumas arestas com peso negativo (sem ciclos de peso negativo).
4. Remover se necessário arestas indisponíveis (com **PA** = 0), e que assim afetam a acessibilidade dos agentes económicos.
5. Verificação se todos os vértices referentes à autoridade pública e agentes económicos, pertencem todos a uma componente fortemente conexa utilizando o algoritmo **Kosaraju complexidade temporal** O(V+E) e com complexidade espacial O(V).
   Consiste resumidamente em::
   - Criar uma pilha **P** vazia e efectuar uma visita em profundidade (**DFS)** ao grafo tendo em conta:
     - Que ao chamar **DFS** recursivamente para vértices adjacentes de um dado vértice, coloco esse vértice em **P.**
   - Obtenho um **grafo transposto** referente ao grafo original, onde as arestas entre cada par de vértice estão no sentido oposto do nosso grafo em questão.
   - Para cada vértice v ∈ **P,** um por um até que **P** fique vazia, efetuo DFS partindo desse vértice, e obtenho assim a componente fortemente conexa referente a v.

Nota: Por uma questão de rapidez, mas também visto que nenhum dos nossos mapas se traduz num grafo denso, utilizaremos o algoritmo **Dijkstra(** ou o algoritmo **Bellman Ford** por uma questão de comparação) para obter os tempos de viagem mínimos entre cada **vértice relevante** ao nosso problema na escolha de uma rota, e todos os restantes vértices, sem ter de processar primeiro nenhuma tabela.Deixando assim de ser fundamental usar o algoritmo Floyd-Warshall ou até mesmo Dijkstra para calcular tempo de viagem mínimo entre cada par de vértice, a não ser que haja intenções futuras de usar **mapas que se traduzem em grafos densos** ou que seja plausível esperar o tempo necessário para processar a tabela e poder aceder à mesma no futuro de forma quase imediata.

3.2. Possível Resolução do Problema Para o 1º Objectivo

1. Uma função por nós implementada, dá uma pontuação a cada agente económico (quanto maior a pontuação, mais urgente é), usando uma fórmula e restrição :

   Fórmula: p = 0.5* QG + IF + 0.2* TQ + AM* 0.1
   Se ano da UI < 2020: p = p * 2

   Foto de função implementada:

```
double AgenteEconomico:: calcUrgenciaInspec() const{

    //Fórmula para classificação de urgência dos agentes económicos
    double urgencyPontuation = 0.0;
    urgencyPontuation += this->getDenuncias()->get_num_graves()*0.5;
    urgencyPontuation += this->getDenuncias()->get_num_total() * 0.2;
    urgencyPontuation += this->get_area()* 0.1;
    urgencyPontuation += this->getInspecoes()->get_num_reprovadas() * 0.2;



    if(this->getDataUi()->getAno() < 2020)
        urgencyPontuation *= 2;



    return urgencyPontuation;


}
```

2. Usando a tabela pré-processada com os tempos de viagem mínimos entre cada par de vértices, distribuímos a uma brigada os agentes económicos pela ordem de urgência obtida , sendo que **uma brigada** pode inspecionar qualquer tipo de atividade económica ou teremos várias rotas para todas as brigadas r**espeitando a especialidade das mesmas.**

3. A rota óptima será a que tem os tempos de viagem mínimos entre cada agente económico e autoridade pública, onde cada tempo de viagem para cada agente económico (desde a origem **VAP**) é somado com o **TI (**tempo de inspeção) do mesmo**.** Escolhendo sempre **a cada passo,** o agente económico mais urgente de ser inspecionado onde se **cumprem todas as restrições** referentes às brigadas (ou brigada) e os agentes económicos em questão. Restrições essas já referidas de forma detalhada anteriormente.

4. A **soma** com o tempo de viagem (até ao agente económico) mínimo e o tempo de inspeção (**TI)** do mesmo é **mínima ,** sendo inseridos à rota se o **HF** for respeitado, caso contrário passamos ao próximo pela ordem definida dada pela pontuação definida pela **função** de prioridade de urgência, mas que a cada novo passo seguinte na mesma, os que têm urgência prioritária em que **HF** anteriormente não era compatível, até que o **HI** da brigada seja esgotado , que já não exista nenhum agente económico a ser inspecionado.
5. O tempo de inspeção de cada Agente económico será sempre tido em conta e forma a que nenhuma brigada ultrapasse o seu **HI.**

6. O caso de várias brigadas com apenas uma especialidade, que a nosso ver, terá o maior interesse, será simplesmente aplicar o raciocínio em cima, tendo em conta as prioridades entre agentes económicos que operam no **mesmo tipo** de atividade económica, tentando sempre utilizar todas ou o maior número de brigadas possível e inspecionar os agentes económicos mais prioritários.

3.3. Possível Resolução do Problema Para o 2º Objectivo

Tendo em conta o que foi dito para o 1º objectivo, mas agora sem haver uma prioridade entre agentes económicos temos que:

1. Usando a tabela referida, a nossa rota óptima para **uma brigada especializada em todos os tipos de atividade económica**, será aquela que terá os tempos de viagem óptimos (contabilizando os respectivos **TI** e tendo em conta os **HF)**, entre os agentes económicos e autoridade pública mas que tenha o maior número de agentes económicos possível, respeitando o **HI** da brigada, procurando assim a cada passo, escolher o agente económico mais perto localmente (menor soma do tempo de viagem e **TI** ou apenas menor tempo de viagem)e que cumpre todas as restrições referidas.A rota óptima que pretendemos obter, tem ser selecionada de inúmeras possíveis mas que inspecionam menos agentes económicos.

2. Para o **caso de várias brigadas especializadas numa só atividade económica**, o raciocínio será o mesmo, mas agora distribuindo rotas óptimas para várias brigadas tentando sempre utilizar todas ou o maior número possível de brigadas.

Nota: É de salientar, que as rotas que tenham tempos de viagem mínimos entre cada agente económico , começando sempre na autoridade pública podem permitir ter um maior número de agentes económicos inspecionados, contudo as mesmas também dependem do **TI** de cada agente económico, podendo por exemplo um tempo de viagem muito curto entre a autoridade pública e o agente económico candidato a ser inspecionado, mas este ter um **TI** muito elevado, o que já permite ir ao encontro de obter um maior número de inspeções numa rota, se só considerarmos o **tempo de viagem** entre cada agente económico, visto que podemos ter **outro agente económico com tempo de viagem mais longo**, mas que ao ter um **TI** mais curto, é **mais rentável** de ser inspecionado para se obter o nosso objectivo . Deste factos, vem a ideia de implementar uma **abordagem híbrida** e mais flexível ao nosso problema.

3.4. Solução híbrida para possível solução de ambos os objectivos

3. Esta solução, que será explicada mais detalhadamente de seguida à medida que formos descrevendo os algoritmos utilizados e funcionalidades, consiste resumidamente em ter a flexibilidade de conjugar vários fatores de forma a escolher uma rota possível, que já foram referidos: Urgência de inspeção como principal fator, ou apenas um fator adicional a outras fatores tais como: a cada passo da definição de uma rota, contabilizar a soma de tempo de viagem até um agente económico com o **TI** do mesmo, ou apenas o tempo de viagem. E também conjugar o facto de limitar as inspeções a uma certa zona utilizando a estratégia com um raio e local como centro utilizando as coordenadas GPS como já se referiu e se explica melhor a seguir. Ou seja, uma estratégia que permite resolver os problemas em cima descritos, mas que depois permite conjugar vários fatores para obter uma solução "intermédia" ou que sirva situações específicas que podem acontecer no mundo real.

3.5. Resolução Para os Objectivos 1 e 2 (Usando Coordenadas GPS)

Todas as etapas para os objectivos referidos serão como descritas em cima, contudo com uma mudança essencial, estabelecendo um raio desde a autoridade pública ou outro local possível de selecionar, de tamanho à escolha e minimamente realista, os agentes econômicos a serem inspecionados terão também como

restrição estarem a uma distância em linha reta da autoridade pública menor ou igual ao raio escolhido.

Outros critérios usando as coordenadas GPS, podem ser definir um raio de ação numa zona que tenha grande aglomerado de agentes económicos com o mesmo tipo de atividade económica (e.g. zona industrial), ou numa zona onde se identifica um número anormal de queixas de vários agentes económicos,

Tempos de inspeção **(TI)** serão calculados usando uma função que usa a área como fator predominante e que será mostrada.

A verificação do **HF** de cada agente económico**,** simplesmente terá em conta o **HIB** de cada brigada, ou seja, **somando HIB** os : tempos de viagem e **TI** dos agentes económicos anteriormente processados e alocados à rota em construção e o tempo de viagem mínimo até ao agente económico em que estamos a verificar **HF,** a soma tem de ser inferior ou igual ao horário de fecho presente em **HF**(horário de funcionamento).

4. Casos de Utilização Implementados e Funcionalidades

No menu principal, o utilizador pode escolher entre várias opções como se pode observar na seguinte figura.

```
================================================================================================================
                    InspectIT: Rotas de Inspecao
================================================================================================================
          AGENTES ECONOMICOS            |              BRIGADAS
========================================|=======================================================================
Visualizar informacao dos agentes  [1]  |  Visualizar informacao das brigadas                          [5]
Adicionar agente economico         [2]  |  Adicionar brigada                                           [6]
Inserir denuncia                   [3]  |  Remover brigada                                             [7]
Remover agente economico           [4]  |  Cálculo e Visualização de rotas das brigadas                [8]
Exit                               [0]  |  Conectividade                                               [9]
                                        |  Performance de Algoritmos                                   [10]
                                           Cálculo de uma rota simples                                  [11]
Escolha uma opcao (0-11):
```

interface inicial do programa

Funcionalidades implementadas:

- **Visualizar informação relativa aos agentes económicos (opção 1)**:

```
Lista de todos os agentes economicos:
ID          |ATIVIDADE ECONOMICA            |AREA       |HORARIO FUNCIONAMENTO |N° DENÚNCIAS GRAVES |N° TOTAL DENÚNCIAS|APROVADAS      |REPROVADAS
35          |Obras                          |100        |8-19                  |4                   |30                |6              | 2
34          |IntervencaoViaPublica          |50         |8-17                  |2                   |5                 |1              | 1
33          |GenerosAlimenticios            |70         |7-18                  |1                   |6                 |1              | 1
32          |SegurancaSalubridadeEdificacoes|80         |8-18                  |4                   |15                |7              | 2
31          |Comercial                      |90         |8-17                  |2                   |30                |2              | 1
30          |Ambiental                      |150        |9-18                  |4                   |15                |7              | 2
29          |IntervencaoViaPublica          |60         |6-18                  |2                   |5                 |1              | 3
28          |Obras                          |50         |6-20                  |5                   |9                 |3              | 5
27          |Todas                          |50         |6-20                  |5                   |9                 |3              | 5
26          |Obras                          |80         |9-19                  |5                   |6                 |3              | 3
25          |IntervencaoViaPublica          |50         |6-18                  |2                   |5                 |1              | 3
24          |Comercial                      |90         |9-17                  |2                   |30                |2              | 1
23          |Ambiental                      |150        |9-18                  |4                   |15                |7              | 2
```

- **Adicionar um novo agente económico (opção 2):**
    - Faz uma série de perguntas ao utilizador e guarda as novas informações.

```
Escreva a atividade economica do agente que deseja adicionar
OPCOES:
 - Todas
 - Obras
 - Comercial
 - Ambiental
 - IntervencaoViaPublica
 - SegurancaSalubridadeEdificacoes
 - GenerosAlimenticios
```

- **Inserir denúncia (opção 3):**

```
Escreva a o id do agente economico que deseja prestar uma queixa
```

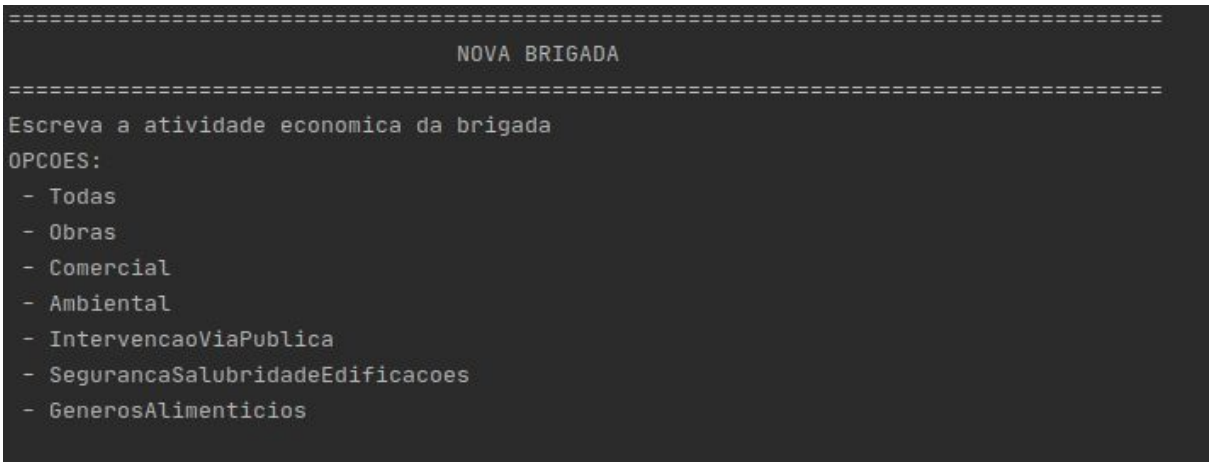
- **Remover um agente económico (opção 4):**

```
Escreva o id do agente que deseja remover [EXIT 0]
```

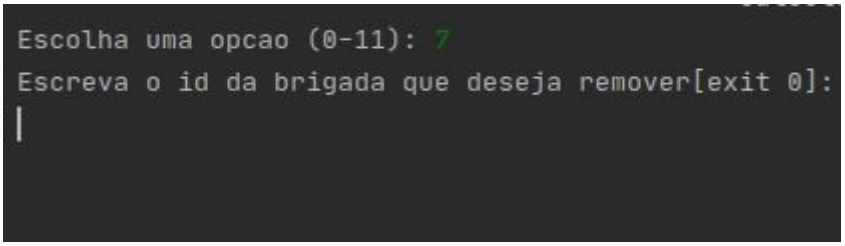
- **Visualizar informação relativa às brigadas da autoridade (opção 5):**

```
Lista de todos as brigadas:
ID            |ATIVIDADE ECONOMICA             |NUMERO HORAS TRABALHO|HORA INICIO
6             |GenerosAlimenticios             |8                    | 9
5             |SegurancaSalubridadeEdificacoes |8                    | 9
4             |IntervencaoViaPublica           |8                    | 9
3             |Obras                           |8                    | 9
1             |Comercial                       |8                    | 9
2             |Ambiental                       |8                    | 9

[PRESS ENTER TO CONTINUE]
```

- **Adicionar uma nova brigada (opção 6):**
    - Faz uma série de perguntas ao utilizador e guarda as novas informações.



- **Remover uma brigada (opção 7):**



- **Cálculo e Visualização de rotas das brigadas (opção 8):**
    - Faz uma série de perguntas ao utilizador e guarda as novas informações.
    - Nesta funcionalidade é onde temos a **abordagem híbrida** que permite resolver o problema 1 e 2 mas também obter soluções intermédias e com outros fatores conjugados ou não que podem ser interessantes.
    - Veremos de seguida uma série de imagens de uma possível abordagem ao cálculo de rotas das brigadas:

```
Escolha uma opcao (0-11): 8
Porto: 1
Espinho: 2
Penafiel: 3
Exit: 0

Escolha uma opcao (1-3):
```

```
Escolha uma opcao (1-3): 1
Introduza velocidade média das brigadas: (Km/h)
Se pretende reutilizar um mapa com o algoritmo FloydWarshall o valor inserido fica sem efeito!

Escolha uma opcao (0-120):
```

```
Se pretende reutilizar um mapa com o algoritmo FloydWarshall o valor inserido fica sem efeito!

Escolha uma opcao (0-120): 40
Introduza o valor para o nó que identifica a Autoridade Pública:

Escolha uma opcao (1-26098):
```

```
Escolha uma opcao (1-26098): 1
Considerar tempo de inspeção de cada Agente Económico no cálculo das rotas? (Sim: 1 ou Não: 2)

Escolha uma opcao (1-2):
```

```
Escolha uma opcao (1-2): 1
Algoritmos:
Dijkstra: 1
BellManFord: 2
Floyd-Warshall: 3

Escolha uma opcao (1-3):
```

```
Escolha uma opcao (1-3): 1
Pretende limitar as inspeções a uma zona,  escolhendo um local e raio envolvente? (Sim: 1 Não: 2)

Escolha uma opcao (1-2):
```

```
Escolha uma opcao (1-2): 1
Indique o IdNo da zona pretendida no mapa:

Escolha uma opcao (1-26098):
```

```
Escolha uma opcao (1-26098): 1
Indique o valor do raio pretendido em Km:

Escolha uma opcao (0-1000):
```

```
Escolha uma opcao (0-1000): 2
Pretende que seja também utilizado um critério de urgência de Inspeção  no cálculo das rotas? (Sim: 1 ou Não: 2)
Pretende que a urgência de inspeção seja o principal factor: 3

Escolha uma opcao (1-3):
```

```
Escolha uma opcao (1-3): 1
Pretende que especialidades das brigadas sejam respeitadas? (Sim: 1 Não: 2):

Escolha uma opcao (1-2):
```

- **Conetividade (opção 9):**
  - Aqui será onde usamos algoritmo Kosoraju para obter todas as componentes fortemente conexas do nosso grafo para três mapas: Porto ,Espinho e Penafiel.
  - Posteriormente se utiliza a maior componente fortemente conexa calcular as rotas pretendidas. Esta coincide com os mapas dados já com a palavra **strong** no nome dos ficheiros.
  - Aparecerá duas janelas com o mapa original e o mapa da mesma cidade mas com a maior componente fortemente conexa e já com a autoridade e agentes económicos localizados devidamente.
  - Teremos também que aparecerão no terminal todas as componentes fortemente conexas referentes ao mapa(e o seu grafo) inicialmente escolhido.
  -

```
Escolha uma opcao (0-3): 3
Introduza velocidade média das brigadas: (Tente ser realista tendo em conta uma zona urbana urbano)

Escolha uma opcao (0-120): 40
Será mostrado o mapa completo da cidade selecionada e de seguida serão calculadas todas as componentes fortemente conexas.
Será escolhida para representar o mapa da cidade pretendida a componente fortemente conexa com maior número de nós.
```

- **Performance de algoritmos (opção 10):**
  - Funcionalidade que permite testar a complexidade empírica dos principais algoritmos utilizados, inclusive aquele que permite calcular as rotas das brigadas usando a nossa abordagem híbrida ou simplesmente resolver os principais problemas do nosso trabalho.
  - Serão testados em 6 mapas, os mapas das cidades já referidas em versão “full”, ou seja, com um maior número de nós mas também os mapas Grid dados nas aulas.

- Permite fazer testes com um número elevado de iterações escolhidas por nós.

```
Escolha uma opcao (0-200): 40
No que pretende testar a Performance?
Dijkstra: 1
BellmanFord: 2
FloydWarshall: 3
Cálculo de rotas com uma ou mais brigadas: 4
Kosaraju (Conetividade): 5
```

```
Escolha uma opcao (1-5): 1
Número de iterações para os testes:

Escolha uma opcao (1-10000000): 10

MapGrid 4x4
Algoritmo Dijkstra
Valores de tempos em microssegundos
Numéro de iterações: 10
Tempo máximo: 7
Tempo mínimo: 1
Tempo médio: 3.4
Numéro de nós: 25
Número de arestas: 40
Total de tempo: 34
```

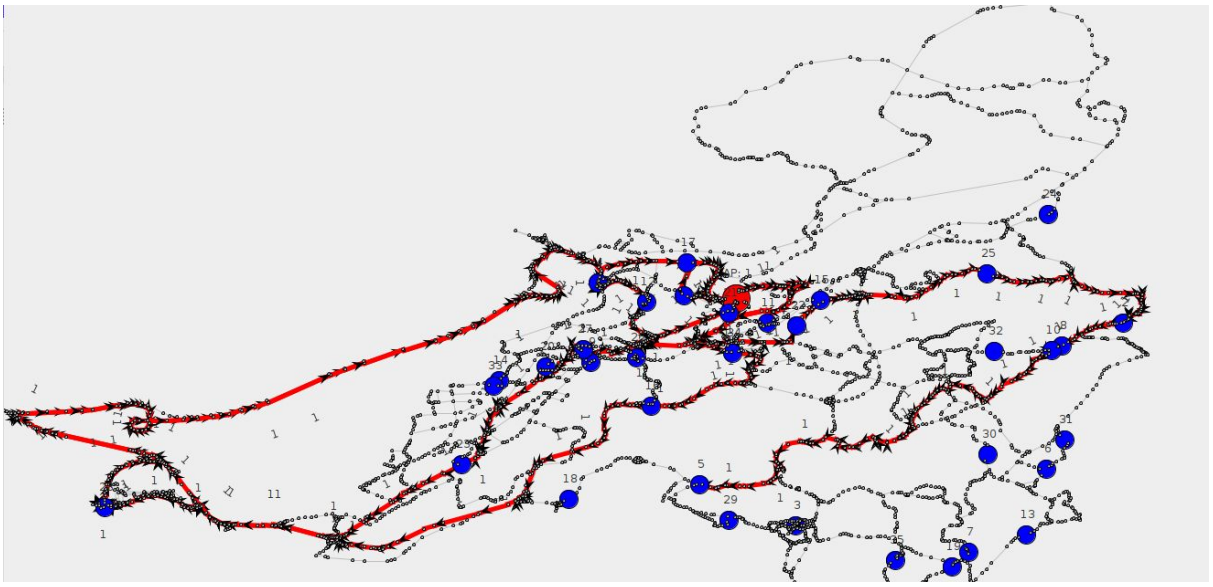
- **Cálculo de uma rota simples (opção 11):**
  - Resolução do problema 2 para apenas uma uma brigada, com ou sem restrição de especialidades.Serve como um exemplo simples e prático de testar.

```
Dijkstra: 1
BellManFord: 2
Floyd-Warshall: 3

Escolha uma opcao (1-3): 1
Introduza velocidade média das brigadas: (Km/h)

Escolha uma opcao (0-120): 40
Introduza o valor para o nó que identifica a Autoridade Pública:

Escolha uma opcao (1-3964): 1
Pretende que especialidades das brigadas sejam respeitadas? (Sim: 1

Escolha uma opcao (1-2): 2
Agentes Económicos Inspecionados: 1 34 16 28 4 27 25 5 |TmpViagInspe
|TmpViagInspecBrig: 8.93556 |Brigada: 1 | |Nós visitados: 1 2651 308
Não fechar janela do graphView se quer continuar no programa:

Deseja visualizar de forma automática as rotas: 1
Deseja controlar o aparecimento das rotas: 2

Escolha uma opcao (1-2):
```

Considerações sobre a parte gráfica dos resultados de cálculo de rotas e outras funcionalidades:

Tanto nesta funcionalidade, como em outras de cálculo de rotas, as brigadas , cada tempo de trabalho diário das mesmas, os agentes econômicos e as rotas calculadas aparecem em janelas (e no terminal) utilizando o GraphView de forma interativa automática ou com o nosso controlo. O número da brigada vai aparecendo ao longo da sua respectiva rota que tem setas de direção e é de cor vermelha.
Informações sobre cálculo de rotas, conetividade ou testes de performance aparecem sempre no terminal de forma detalhada.

Exemplo:



- **Sair (opção 0):**
    - Quando esta opção é selecionada, é chamado um “destructor” que coloca todas as informações nos respectivos ficheiros, guardando assim novas informações que possam ter sido adicionadas, e apaga todos os objectos criados.

    - Visualização do grafo e da informação que este representa no GraphViewer
    - Visualização de autoridades e brigadas existentes neste sistema (Cada autoridade tem várias brigadas)
    - Visualização das informações de cada brigada (e.g. inspeções efetuadas e a efetuar, ...)
    - Depois de calculadas as rotas óptimas para cada brigada, mostrar as arestas escolhidas e os vértices por onde passam no GraphViewer
    - A cada brigada será automaticamente atribuído uma rota e os pontos a inspecionar, sendo esta atribuição dependente das restrições e do que escolhemos. Ou seja, esta rota será calculada segundo a sua localização, a sua especialidade, etc.
    - Importação das informações dos ficheiros disponíveis para visualização ordenada do mesmo
    - O utilizador tem a possibilidade de escolher uma rota que priorize mais inspeções urgentes a fazer, ou uma rota que prioriza mais visitar o maior número de agentes económicos num dia entre outros fatores já explicados em cima.

- O utilizador terá a possibilidade de criar brigadas, definindo para estas a sua especialidade (uma ou mais)

4. Estruturas de Dados Utilizadas

4.1. Estruturas auxiliares

- Para auxiliar as classes principais, criaram-se estruturas, classes e funções:
  - **AtividadeEconomica**: estrutura do tipo enum que guarda as várias atividades económicos (Todas, Obras, Comercial, Ambiental, IntervencaoViaPublica, SegurancaSalubridadeEdificacoes, GenerosAlimenticios)
    (definida em AgenteEconomico.h)
  - **Data**: estrutura que permite definir uma data. Através do seu constructor é possível converter uma string da forma "aaaa/mm/dd" num objecto com um ano, mês e dia.
    (definida em Data.h)
  - **Error**: estrutura que permite lançar uma exceção, e por sua vez, uma mensagem de erro ao utilizador.
    (definida em Erro.h)
  - Todas as funções auxiliares foram definidas em "Utils.h" e a sua utilidade é explicada nos comentários do código.

4.2. Autoridade Pública

- A nossa autoridade pública é representada por dois conjuntos: os agentes económicos e as brigadas.

```
unordered_map<unsigned int, Brigada *> brigadas;
unordered_map<unsigned int, AgenteEconomico *> agentes;
```

atributos da classe Autoridade_Publica

- Para além das funções de get e set, implementámos outras funções:

- **AutoridadePublica():** constructor que chama as funções que processam ficheiros
- **destrutor():** "destructor" que escreve a informação dos mapas para os respetivos ficheiros e apaga todos os objetos
- **processarFicheiroAgentes():** lê e processa a informação do ficheiro agentes.txt
- **processarFicheiroBrigadas():** lê e processa a informação do ficheiro brigadas.txt
- **imprimirAgentesEconomicos():** imprime na consola a informação em relação aos agentes económicos de forma formatada
- **imprimirBrigadas():** imprime na consola a informação em relação às brigadas de forma formatada
- **adicionarAgenteEconomico():** faz várias perguntas ao utilizador, com essas respostas cria um novo objecto do tipo *AgenteEconomico* e guarda o novo objecto no mapa *agentes*.
- **removerAgente():** remove do mapa *agentes* um agente económico com um dado *id*
- **inserirDenuncia():** insere uma denúncia a um agente económico com um dado *id*, atualizando informações de objectos do tipo *AgenteEconomico* já existentes.
- **adicionarBrigada():** faz várias perguntas ao utilizador, com essas respostas cria um novo objecto do tipo *Brigada* e guarda o novo objecto no mapa *brigadas*.
- **removerBrigada():** remove do mapa *brigadas* um agente económico com um dado *id*.
- **stringToAE():** converte um objeto do tipo *AtividadeEconomica* para uma string, o que é útil para as funções de imprimir.
- Outras funções auxiliares mas não tão relevantes

4.3. Agente Económico

- O agente económico é representado pelos seguintes atributos

```
this->atividades_economicas = atividadesEconomicas;
this->area = area;
this->horario_funcionamento = horario_funcionamento;
this->denuncias = denuncias;
this->inspecoes = inspecoes;
this->tmpInspecao = gerarTmpInspecao();
this->dataUI = dataUI;
this->id = id;
this->urgencyPontuation = calcUrgenciaInspec();
```

- Para além das funções de get e set, implementámos outras funções:
    - **AgenteEconomico():** constructor
    - getAtividadeString(): apagar!!
    - **gerarTmpInspecao():** calcula o tempo de inspeção em horas usando a área do mesmo.
    - **imprimirFicheiro():** escreve para uma ostream (neste caso, será o ficheiro de destino agentes.txt) o valor dos atributos com a formatação necessária.
    - **calcUrgenciaInspec()** que calcula uma pontuação para cada agente económico e determina a sua urgência de ser inspecionado e que já temos uma imagem em cima.

4.4. Denúncias

- As denúncias são representadas pelos seguintes atributos:

```
unsigned int num_graves; //Número de queixas muito graves (QG)
unsigned int num_total; //Número total de queixas (TQ)
```

atributos da classe Denuncias

- Para além das funções de get e set, implementámos outras funções:
    - **Denuncias:** constructor
    - **adicionar_uma_grave()**: incrementa por 1, *num_graves*
    - **adicionar_uma_normal()**: incrementa por 1, *num_total*

4.5. Inspeções

- As inspeções são representadas pelos seguintes atributos:

```
unsigned int num_aprovadas; //Número de  inspeções aprovadas
unsigned int num_reprovadas; //Número de  inspeções reprovadas
```

atributos da classe Inspecoes

- Para além das funções de get, set e o constructor não foram implementadas mais funções.

4.6. Brigada

- A brigada é representada pelos seguintes atributos:

```
unsigned  int id; //id da brigada (ID)
AtividadeEconomica atividades_economicas; //Atividades económicas em que a brigada se especializa (AEi)
unsigned int horas_trabalho; //Número de horas diário total de trabalho (HI)
unsigned int hora_inicio; //Hora do dia em que a brigada começa a sua atividade
```

atributos da classe Brigada

- Para além das funções de get e set, implementámos outras funções:
    - **Brigada():** construtor.
    - getAtividadeString_: apagar!!!
    - **imprimirFicheiro():** escreve para uma ostream (neste caso, será o ficheiro de destino brigadas.txt) o valor dos atributos com a formatação necessária.
- **MutablePriorityQueue**: esta estrutura de dados é utilizada para o algoritmo Dijkstra.
- **Edge**: a classe Edge que retrata uma aresta, guarda o vértice de início, o vértice de fim e neste caso o tempo de viagem entre dois vértices.
- aresta (definida em Graph.h).
- **Local**: a classe local, guarda as coordenadas x, y, longitude e latitude. Onde x e y são utilizados para identificar de forma fácil cada vértice utilizando o graphView e a latitude e longitude exclusivamente medir distâncias reais nos mapas utilizados e assim obter tempos de viagem entre vértices como já referimos.
- **Vertex**: esta classe guarda um objeto da classe **Local,** o seu id, o tipo de ponto (tags utilizadas para identificar agentes económicos, autoridade pública ou outros locais), objecto da classe Edge entre outros ,atributos gets e sets e funções auxiliares que permitem utilizar os algoritmos utilizados por nós nos grafos função **getTempViagem(...)** e **getDistLatLong(...)** que permitem calcular distância entre vértices utilizando latitude e longitude e assim posteriormente permitindo obter tempos de viagem entre vértices (definida Graph.h).
- **Graph:** classe que guarda um vetor de vértices e um vetor de vetores de doubles, que guarda as distâncias entre cada par de vértices, quando pretendemos fazer o pré-processamento de tabela já referido. Temos também funções auxiliares para ajudar a ajustar a informação das janelas usando o GrapView. Tem também os algoritmos **Dijkstra, Bellman Ford e Floyd-Warshal ,Kosaraju** e funções auxiliares dos mesmos. Temos também atributos que ajudam a ajustar melhor o grafo utilizando o GraphView.
  Exemplos:

```
<T>
class Graph {

    vector<Vertex<T> *> vertexSet;    // vértices para guardar Locais
    vector<vector<double>> distMin; //Matriz para algoritmo FloyWarshall e não só

    vector<vector<Vertex<T> *>> pathMin; //Matriz para algoritmo FloydWarshall e não só
    double minXGraphView = std::numeric_limits<double>::max();
    double maxXGraphView = (std::numeric_limits<double>::min() - 1) * -1;
    double numberOfEdges = 0;
    double minYGraphView = std::numeric_limits<double>::max();
    double maxYGraphView = (std::numeric_limits<double>::min() - 1) * -1;


public:
```

```
  bool addEdge(const T &sourc, const T &dest, double w);
  int getNumVertex() const;
  vector<Vertex<T> *> getVertexSet();

  bool relax(Vertex<T> *vOrig, Vertex<T> *wDest, double edgeValue);
  void unweightedShortestPath(const T &origin);
  void dijkstraShortestPath(const T &origin);
  void bellmanFordShortestPath(const T &orig);
  vector<T> getPathTo(const T &dest) const;

  // Distancia mínima entre  todos os pares de vértices para guardar numa tabela de pré-processamento


  vector<T> getfloydWarshallPath(const T &orig, const T &dest) const;
  void floydWarshallShortestPath(unsigned int indexStartidNodes, double errorEPS);
  vector<T> bfs(const T &source) const;


  void dfsVisit(Vertex<T> *v, vector<T> &res) const;


  vector<T> dfs() const;
```


```
vector<T> getPathTo(const T &dest) const;

// Distancia mínima entre  todos os pares de vértices para guardar numa tabela de pré-processamento


vector<T> getfloydWarshallPath(const T &orig, const T &dest) const;
void floydWarshallShortestPath(unsigned int indexStartidNodes, double errorEPS);
vector<T> bfs(const T &source) const;


void dfsVisit(Vertex<T> *v, vector<T> &res) const;


vector<T> dfs() const;
void  increasingTimeOrder(Vertex<T> *vertex, stack<Vertex<T> *> &stackVertex);
Graph<int> getInverseEdgesGraph();
vector<vector<int >>  KosarajuMainAlgo();
void  saveStrongCC(Vertex<T> *v, vector<int> &saveSSC);

void eraseGraph();
```

5. Algoritmos Implementados considerações relevantes ao nosso Problema e solução encontrada.

Dijkstra:

Algoritmo ganancioso, utilizado para encontrar caminho mais curto(no nosso casos com tempo de viagem mínimo) entre vértices de um grafo pesado, com pesos negativos. Implementado utilizando uma **MutablePriorityQueue,** sendo esta uma fila de prioridade mutável com o mínimo à cabeça de forma a escolher o próximo vértice a ser processado**.** Comporta de uma forma semelhante à pesquisa em largura (**BFS**).

Não permite ter arestas de pesos negativo e ou ciclos de peso negativo.

É o algoritmo mais utilizado por nós, visto que todos os nossos mapas são esparsos e podemos não querer efetuar o pré-processamento da tabela referida por nós anteriormente.Permite calcular os tempos de viagem mínimos entre cada agente económico escolhido para uma rota, simplesmente obtendo a cada passo da

construção da rota, obter os tempos de viagem entre um agente econômico a ser considerado e todos os outros vértices(e agentes económicos restantes), passando para o seguinte e assim sucessivamente..Procedendo dessa forma é possível obter o pretendido para cada rota.

Complexidade temporal O ((| V | + | E |) * log(| V |) no nosso caso, mas que no limite pode ser melhorada para O (| V | * log(| V |) recorrendo a Fibonacci Heaps. A complexidade espacial é O (| V |).

```
DIJKSTRA(G, s):   // G=(V,E), s ∈ V
1.    for each v ∈ V do
2.       dist(v) ← ∞
3.       path(v) ← nil
4.    dist(s) ← 0
5.    Q ← ∅   // min-priority queue
6.    INSERT(Q, (s, 0)) // inserts s with key 0
7.    while Q ≠ ∅ do
8.       v ← EXTRACT-MIN(Q) // greedy
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  // old dist(w) was ∞
14.               INSERT(Q, (w, dist(w)))
15.            else
16.               DECREASE-KEY(Q, (w, dist(w)))
```

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

Bellman Ford:

Algoritmo que utiliza programação dinâmica, que permite calcular distâncias(no nosso casos com tempo de viagem mínimo) entre um vértice de um grafo e todos os outros.Que permite haver arestas com peso negativo, mas não ciclos de peso negativo.Utilizado no nosso problema, apenas para comparação.

Complexidade temporal O(|V||E|) e complexidade espacial O(V).

```
BELLMAN-FORD(G, s):   // G=(V,E), s ∈ V
1.   for each v ∈ V do
2.      dist(v) ← ∞
3.      path(v) ← nil
4.   dist(s) ← 0
5.   for i = 1 to |V|-1 do
6.      for each (v, w) ∈ E do
7.         if dist(w) > dist(v) + weight(v,w) then
8.            dist(w) ← dist(v)+ weight(v,w)
9.            path(w) ← v
10.  for each (v, w) ∈ E do
11.     if dist(v) + weight(v,w) < dist(w) then
12.        fail("there are cycles of negative weight")
```

Tempo de execução: O(|E| |V|)

Floyd-Warshall:

O algoritmo de Floyd-Warshall é um algoritmo interessante nos casos de grafos pesados. Permite calcular as distâncias (tempos de viagem) mínimos entre cada par de vértices. Como não temos nenhum grafo desse gênero, e o pré-processamento não é uma prioridade pelo facto de usarmos o algoritmo de Dijkstra, este serve apenas como ponto de comparação. Funciona para grafos orientados e não orientados, mas nunca para grafos com ciclos negativos.

É de salientar que na nossa implementação introduzimos um **fator de erro,** de forma a salvaguardar a propagação de erros que acontece no cálculos de tempos de viagem do algoritmo.

A complexidade temporal é O(|V| 3) e complexidade espacial O(|V|^2).

```
ROTINA fw(Inteiro[1..n,1..n] grafo)
    # Inicialização
    VAR Inteiro[1..n,1..n] dist := grafo
    VAR Inteiro[1..n,1..n] pred
    PARA i DE 1 A n
        PARA j DE 1 A n
            SE dist[i,j] < Infinito ENTÃO
                pred[i,j] := i
    # Laço principal do algoritmo
    PARA k DE 1 A n
        PARA i DE 1 A n
            PARA j DE 1 A n
                SE dist[i,j] > dist[i,k] + dist[k,j] ENTÃO
                    dist[i,j] = dist[i,k] + dist[k,j]
                    pred[i,j] = pred[k,j]
    RETORNE dist
```

Algoritmo Kosaraju:

Algoritmo utilizado para obter todas as componentes fortemente conexas de uma determinado grafo (mapa).

Como já explicado anteriormente, utiliza duas visitar em profundidade. uma pilha e uma inversão de arestas do grafo como que estamos a lidar.

Todos os outros algoritmos mas também o algoritmo que utilizaremos para obter as rotas das brigadas, dependem deste.

Pois todo o nosso problema, depende do facto de que é possível ir de um qualquer agente económico, autoridade pública ou vértice em geral para um outro qualquer do nosso grafo.

Tem complexidade temporal e espacial de **O(V+E)** **e** **O(V)** respectivamente.

```
algorithm tarjan is
 input: graph G = (V, E)
 output: set of strongly connected components (sets of vertices)

 index := 0
 S := empty array
 for each v in V do
   if (v.index is undefined) then
     strongconnect(v)
   end if
 end for

 function strongconnect(v)
   // Set the depth index for v to the smallest unused index
   v.index := index
   v.lowlink := index
   index := index + 1
   S.push(v)
   v.onStack := true

   // Consider successors of v
   for each (v, w) in E do
     if (w.index is undefined) then
       // Successor w has not yet been visited; recurse on it
       strongconnect(w)
       v.lowlink  := min(v.lowlink, w.lowlink)
     else if (w.onStack) then
       // Successor w is in stack S and hence in the current SCC
       // If w is not on stack, then (v, w) is a cross-edge in the DFS tree and must be ignored
       // Note: The next line may look odd - but is correct.
       // It says w.index not w.lowlink; that is deliberate and from the original paper
       v.lowlink  := min(v.lowlink, w.index)
     end if
   end for

   // If v is a root node, pop the stack and generate an SCC
   if (v.lowlink = v.index) then
     start a new strongly connected component
     repeat
       w := S.pop()
       w.onStack := false
       add w to current strongly connected component
     while (w != v)
     output the current strongly connected component
   end if
 end function
```

Construção das rotas de forma a resolver o nosso problema:

Como já referimos o nosso problema e objetivos a serem cumpridos, são muito parecidos com instâncias do Traveling Salesman Problem(TSP).

A sua solução sem qualquer tipo de **método heurístico** para obter **soluções aproximadas**, significa testar **todas as possibilidades de rotas possíveis,** o que nos dá uma complexidade de **O (n!)**.

No nosso caso mais simples, temos uma brigada sem especialidade, o que equivale a um caso homogéneo do TSP, mas com a limitação de trabalho diário da brigada e as restrições inerentes ao nosso problema já referidas e os objectivos específicos do problema.

No caso de várias brigadas, com a possibilidade de tempo de horário de trabalho diferentes (a capacidade da brigada), temos um caso

heterogéneo do **TSP** e que claro também cumpre as restrições e objetivos específicos do nosso problema.

Optámos por tentar criar uma estratégia nossa, a estratégia **híbrida** que por um lado utiliza a mesma estratégia que o algoritmo **Nearest-Neighbour** que basicamente escolhe o vértice mais próximo como sendo a ser visitado, ou seja os nossos agentes económicos.Entre cada par de agentes económicos vamos utilizando essa estratégia.

Esta estratégia por vezes, pode excluir possibilidades de escolha do próximo agente económico a inspecionar que poderiam a levar a melhores rotas consoante os objectivos pretendidos.

Ao mesmo tempo todas as restrições e fatores mais ou menos preponderantes (escolhidos por nós) têm de ser cumpridos, isto é controladopor uma série de funções que estarão dentro do algoritmo principal e que calcula rotas que iremos mostrar a seguir.

Um facto interessante de ser referido, é que a manipulação de alguns desses fatores por nós, permite alguma **aleatoriedade no algoritmo Nearest-Neighbour** o que por vezes permite dar melhores resultados.

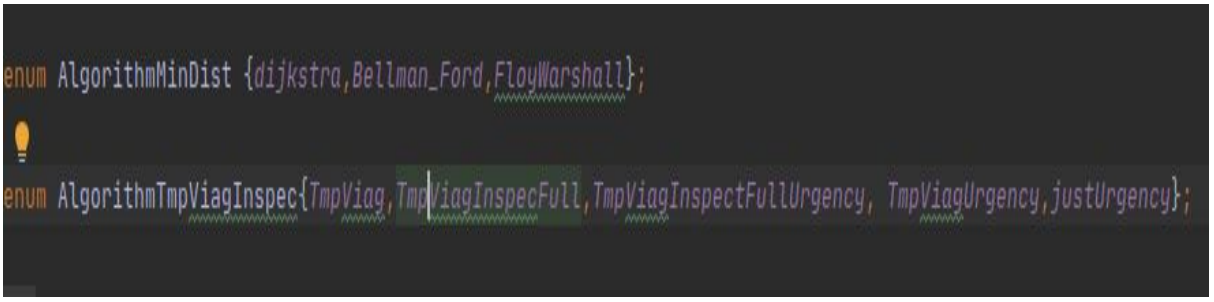
Pseudo-código de **Nearest-Neighbour:**

Algoritmo utilizado para cálculo de rotas:

getRotaBrigada(....):

Este algoritmo utiliza em parte um design pattern **Strategy,** pois modifica o seu comportamento consoante os parâmetros passados.

Os parâmetros referidos e que alteram o seu comportamento são:



AlgorithmMinDist: Permite utilizar os três algoritmos já referidos e implementos por nós.

Abordagem Híbrida ao nosso problema:

AlgorithmTmpViagInspec: Permite escolher as seguintes restrições que permitem resolver os nossos problemas enunciados e ao mesmo tempo permitir alguma flexibilidade e resultados interessantes:

-**TmpViag:** Significa que a escolha do próximo agente econômico a ser visitado é aquele que está a menos tempo de viagem, sem contar como o tempo de inspeção, desde que o horário de trabalho e especialidade (ou não se assim quiseremos) da brigada seja cumprido tal como todas as outras restrições.

-**TmpViagInspectFull:** o critério de escolha do próximo agente económico a cada passo é a soma do tempo de viagem e o tempo de inspeção do próximo agente económico, cumprindo as restrições já referidas.

-**JustUrgency**: é o critério que indica que o fator de escolha principal do próximo agente econômico a ser visitado na construção de uma rota, é a pontuação de urgência do mesmo.

-**TmpViagUrgency**: permite que os fatores de escolha na rota s seja o tempo de viagem entre cada agente económico **ou** o tempo de viagem entre eles.

```
, bool distOrFullCheck,bool GPSCheck,pair<int,double> idNoRadius)
```

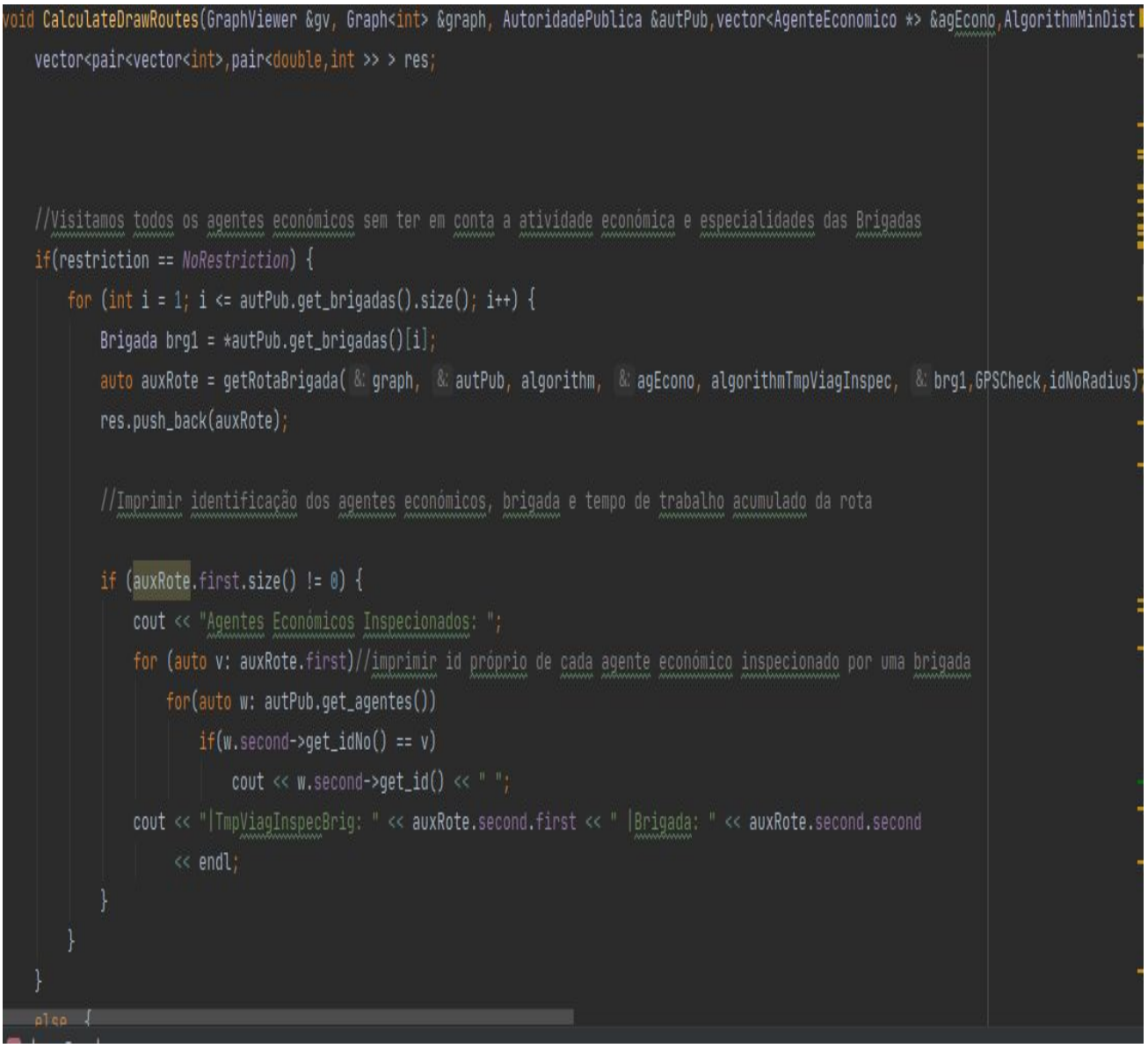
-distorFullCheck: serve apenas como variável auxiliar aos parâmetros TmpViagInspectFull e TmpViag, de forma a ter o comportamento pretendido.

-GPSCheck e idNoRadius: permitem como já referimos, que independente do escolhido por nós referido anteriormente, podemos limitar a escolha de inspeção de agentes económicos a uma zona limitada por um centro e raio escolhido por nós.

Funções auxiliares e que modificam o comportamento de getRotaBrigada(...):

```
if(v!= NULL && agEcono.size()!= 0) {
    // Escolhas das quatro formas de calcular as rotas, : Usando só TmpViagem ou TmpViagem + TmpInspec, TmpViagem + urgencia ou TmpViagem + Tm
    if(algorithmTmpViagInspec == TmpViag)
        v = searchNearestAgEconoTmpViagTmpInpecFull(v->getInfo(),graph,autPub,algorithm,agEcono,tmpViagInspecCounter,brg,false,GPSCheck,idNoRa



    else if( algorithmTmpViagInspec == TmpViagInspecFull ){
        v = searchNearestAgEconoTmpViagTmpInpecFull(v->getInfo(),graph,autPub,algorithm,agEcono,tmpViagInspecCounter,brg,true,GPSCheck,idNoRad

    }
    else if(algorithmTmpViagInspec == TmpViagInspectFullUrgency){
        v = searchNearestAgEconoTViagTInspecFullUrgency(v->getInfo(),graph,autPub,algorithm,agEcono,tmpViagInspecCounter,brg,true,GPSCheck,idN



    }
    else if(algorithmTmpViagInspec == TmpViagUrgency){
        v = searchNearestAgEconoTViagTInspecFullUrgency(v->getInfo(),graph,autPub,algorithm,agEcono,tmpViagInspecCounter,brg,false, GPSCheck,i

    }else {
        v = searchNextUrgency(v->getInfo(),graph,autPub,algorithm,agEcono,tmpViagInspecCounter,brg,GPSCheck,idNoRadius);

    }

    if(v!= NULL){
```

Função auxiliar que permite decidir se queremos especialidades das brigadas sejam cumpridas ou não e cálculo das várias rotas para todas as brigadas:



```
void CalculateDrawRoutes(GraphViewer &gv, Graph<int> &graph, AutoridadePublica &autPub,vector<AgenteEconomico *> &agEcono,AlgorithmMinDist
    vector<pair<vector<int>,pair<double,int >> > res;



    //Visitamos todos os agentes económicos sem ter em conta a atividade económica e especialidades das Brigadas
    if(restriction == NoRestriction) {
        for (int i = 1; i <= autPub.get_brigadas().size(); i++) {
            Brigada brg1 = *autPub.get_brigadas()[i];
            auto auxRote = getRotaBrigada( &graph,  &autPub, algorithm,  &agEcono, algorithmTmpViagInspec,  &brg1,GPSCheck,idNoRadius);
            res.push_back(auxRote);

            //Imprimir identificação dos agentes económicos, brigada e tempo de trabalho acumulado da rota

            if (auxRote.first.size() != 0) {
                cout << "Agentes Económicos Inspecionados: ";
                for (auto v: auxRote.first)//imprimir id próprio de cada agente económico inspecionado por uma brigada
                    for(auto w: autPub.get_agentes())
                        if(w.second->get_idNo() == v)
                            cout << w.second->get_id() << " ";
                cout << "|TmpViagInspecBrig: " << auxRote.second.first << " |Brigada: " << auxRote.second.second
                     << endl;
            }
        }
    }
    else {
```

Complexidade temporal de **getRotaBrigada(): O(|AE| * ((|V|+|E|) * log |V|)* |AE|)**

Complexidade espacial: **O(|V|):**

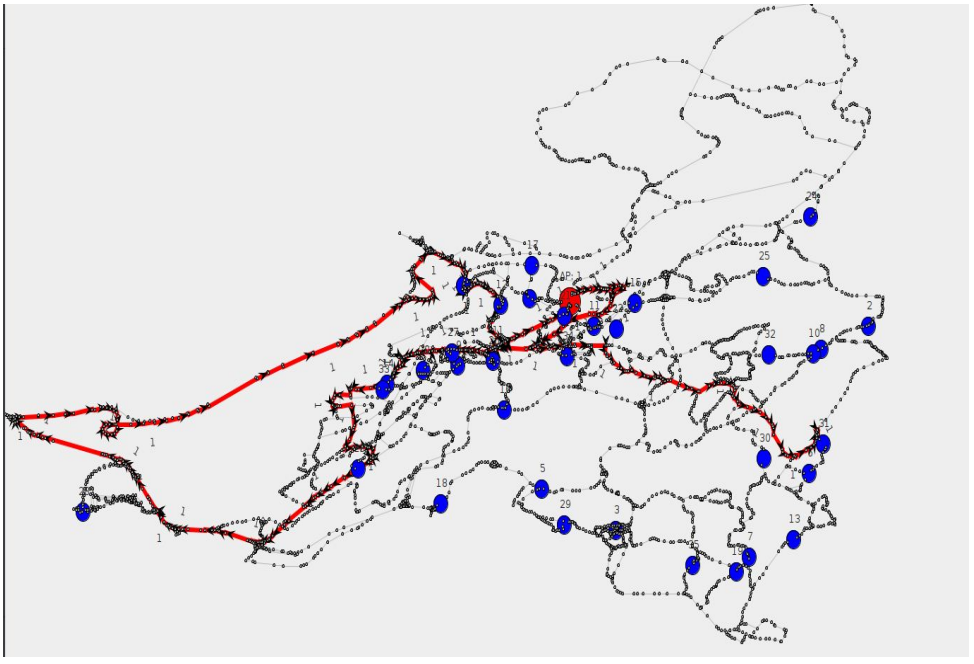
Complexidade temporal do cálculo de todas as rotas: **O((|AE| * ((|V|+|E|) * log |V|)* |AE|) * |Brg|)**

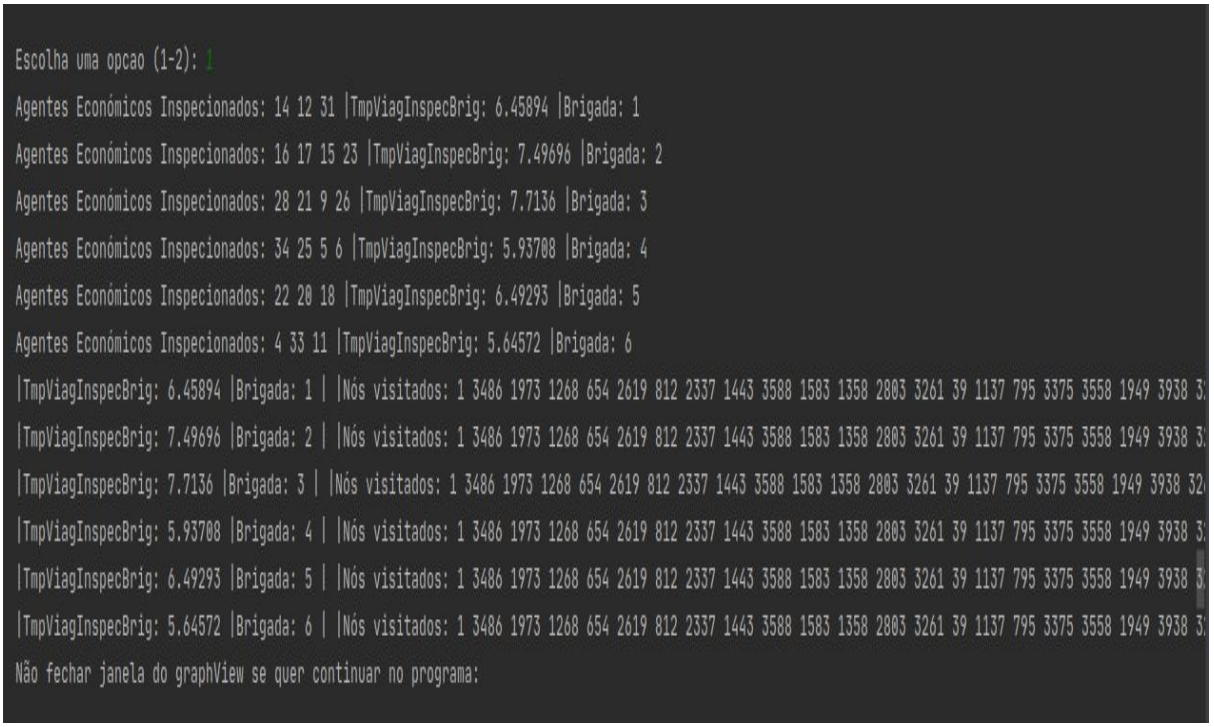
Onde |AE| , |Brg| , V e E são número total de vértices, arestas , agentes económicos e brigadas respectivamente.

Complexidades explicadas pelo facto de utilização do algoritmo Dijkstra entre cada par de agentes económicos que vamos escolhendo para a nossa rota, até que o tempo de trabalho diário da brigada se esgote ou não tenhamos mais agentes económicos para inspecionar (pior caso).

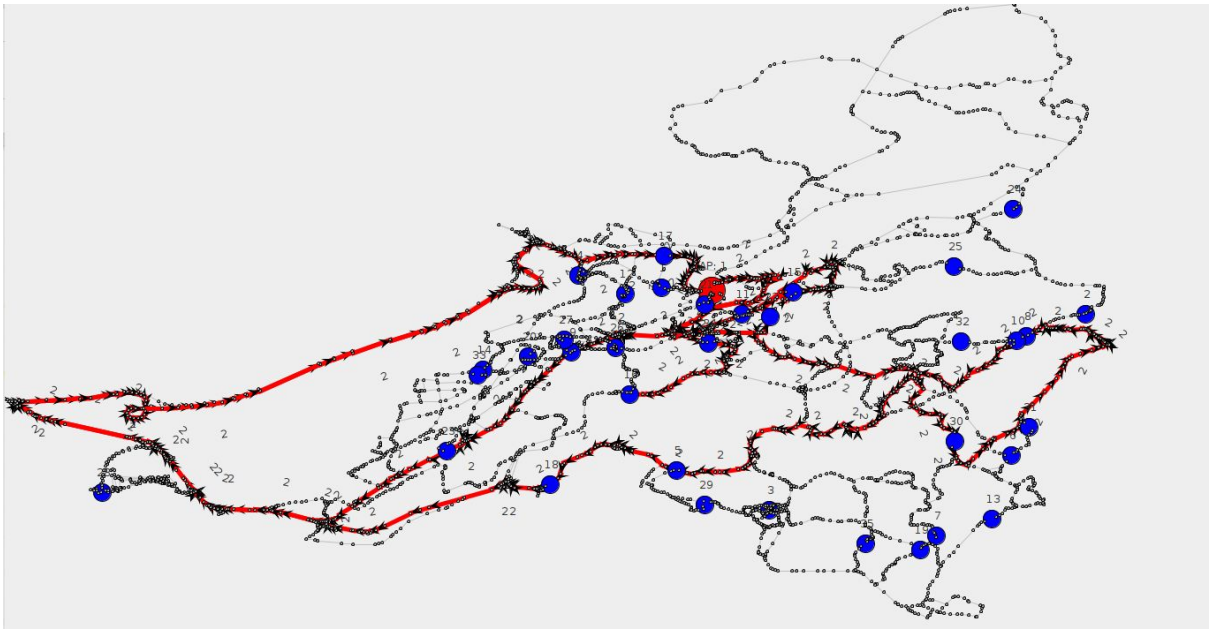
No caso de todas as rotas, basta verificar que a função **getRotaBrigada(...) é chamada |**Brg**| vezes** no caso de ser possível distribuir rotas para todas as brigadas.

Exemplos de resolução para o nosso primeiro e segundo objectivo utilizando todas as brigadas **onde a soma de tempo de viagem com tempo de inspeção** é fator de escolha de agentes económicos na rota, sem ter urgência ou limitação e zona por um raio escolhido como fatores de escolha, e especialidades são respeitadas. Velocidade média de **40 km/h das brigadas.Uso do Algoritmo de Dijkstra.**Apresentaremos apenas uma rota como exemplo de uma brigada.A velocidade média será igual para todos os casos**:**



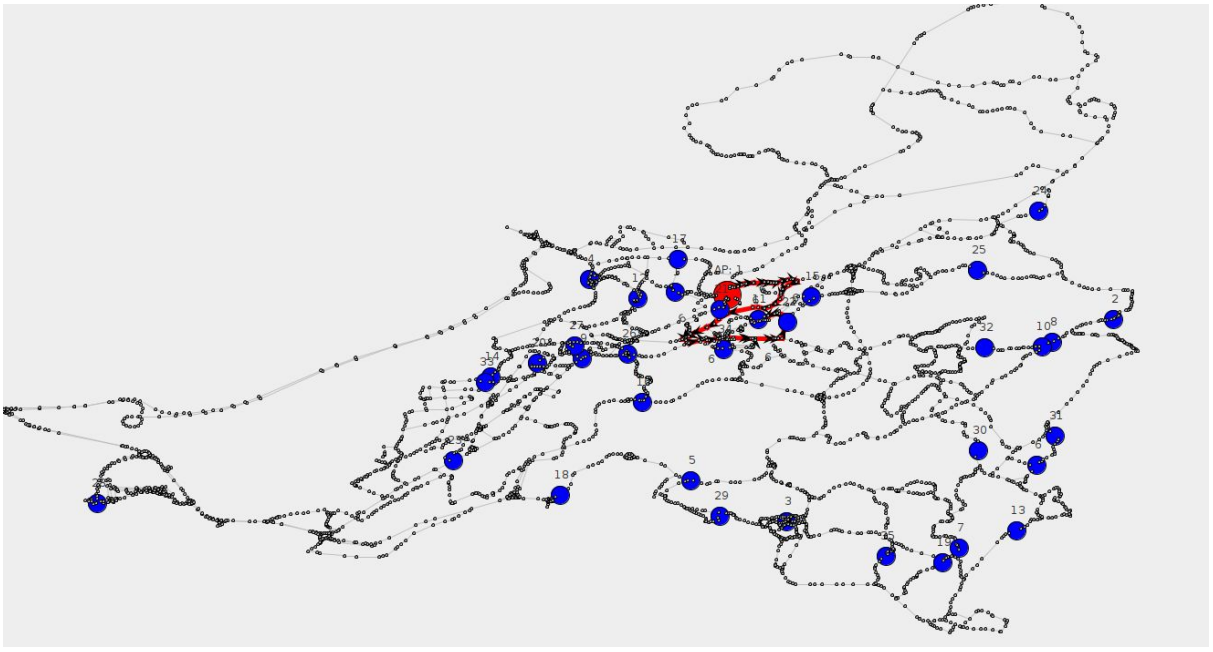


Resolução para o primeiro objectivo tendo a urgência com principal fator de escolha dos agentes económicos nas rotas:



```
Agentes Económicos Inspecionados: 3 12 31 13 24 14 |TmpViagInspecBrig: 7.37437 |Brigada: 1
Agentes Económicos Inspecionados: 23 15 30 17 16 |TmpViagInspecBrig: 5.45993 |Brigada: 2
Agentes Económicos Inspecionados: 2 28 8 35 21 9 |TmpViagInspecBrig: 6.76769 |Brigada: 3
Agentes Económicos Inspecionados: 6 7 29 25 5 34 |TmpViagInspecBrig: 5.76415 |Brigada: 4
Agentes Económicos Inspecionados: 22 20 19 32 18 |TmpViagInspecBrig: 5.7226 |Brigada: 5
Agentes Económicos Inspecionados: 10 4 33 11 |TmpViagInspecBrig: 6.31717 |Brigada: 6
|TmpViagInspecBrig: 7.37437 |Brigada: 1 | |Nós visitados: 1 3486 1973 1268 654 2619 812 2337 1443 3588 1583 1358 2803 3261 39 1137 795 3375 3558 1949 3938
|TmpViagInspecBrig: 5.45993 |Brigada: 2 | |Nós visitados: 1 3486 1973 1268 654 2619 812 2337 1443 3588 1583 1358 2803 3261 39 1137 795 3375 3558 1949 3938
|TmpViagInspecBrig: 6.76769 |Brigada: 3 | |Nós visitados: 1 3486 1973 1268 654 2619 812 2337 1443 3588 1583 1358 2803 3261 39 1137 795 3375 3558 1949 3938
|TmpViagInspecBrig: 5.76415 |Brigada: 4 | |Nós visitados: 1 3486 1973 1268 654 2619 812 2337 1443 3588 1583 1358 2803 3261 39 1137 795 3375 3558 1949 3938
|TmpViagInspecBrig: 5.7226 |Brigada: 5 | |Nós visitados: 1 3486 1973 1268 654 2619 812 2337 1443 3588 1583 1358 2803 3261 39 1137 795 3375 3558 1949 3938
|TmpViagInspecBrig: 6.31717 |Brigada: 6 | |Nós visitados: 1 3486 1973 1268 654 2619 812 2337 1443 3588 1583 1358 2803 3261 39 1137 795 3375 3558 1949 3938
Não fechar janela do graphView se quer continuar no programa:

Deseja visualizar de forma automática as rotas: 1
Deseja controlar o aparecimento das rotas: 2

Escolha uma opcao (1-2): 2
```

Resolução tendo em conta os mesmo fatores que a primeira resolução demonstrado anteriormente mas onde limitamos as inspeções a um raio de 1 km da autoridade pública e onde a urgência é um fator extra que pode influenciar na hora da escolha de agentes económicos:



:

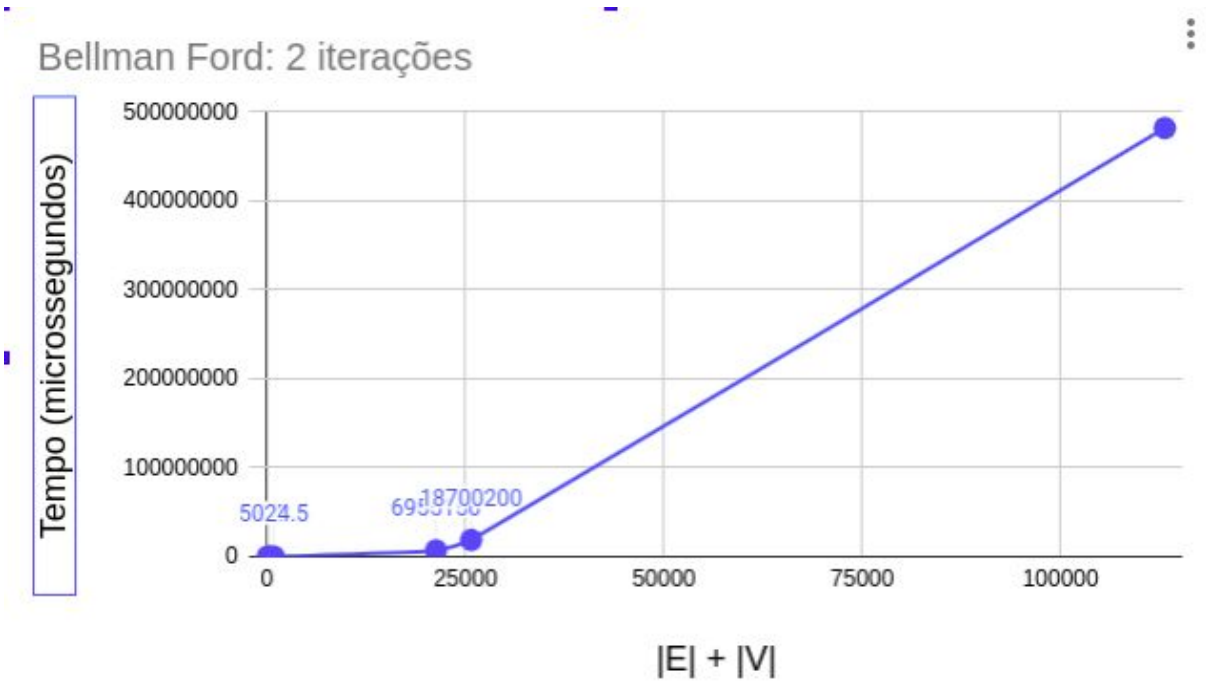
```
Escolha uma opcao (1-2): 1
Agentes Económicos Inspecionados: 12 3 13 |TmpViagInspecBrig: 6.27992 |Brigada: 1
Agentes Económicos Inspecionados: 15 23 17 |TmpViagInspecBrig: 6.37344 |Brigada: 2
Agentes Económicos Inspecionados: 2 28 8 35 |TmpViagInspecBrig: 7.73969 |Brigada: 3
Agentes Económicos Inspecionados: 6 7 29 25 |TmpViagInspecBrig: 7.82643 |Brigada: 4
Agentes Económicos Inspecionados: 22 20 19 |TmpViagInspecBrig: 6.4962 |Brigada: 5
Agentes Económicos Inspecionados: 11 |TmpViagInspecBrig: 2.05952 |Brigada: 6
|TmpViagInspecBrig: 6.27992 |Brigada: 1 | |Nós visitados: 1 3486 1973 1268 654 2619 812 2337 1443 3588 1583 1358 2803 3261 39 1137 795 3375 3558 1949 3938 3
|TmpViagInspecBrig: 6.37344 |Brigada: 2 | |Nós visitados: 1 3486 1973 1268 654 2619 812 2337 1443 3588 1583 1358 2803 3261 39 1137 795 3375 3558 1949 3938 3
|TmpViagInspecBrig: 7.73969 |Brigada: 3 | |Nós visitados: 1 3486 1973 1268 654 2619 812 2337 1443 3588 1583 1358 2803 3261 39 1137 795 3375 3558 1949 3938 3
|TmpViagInspecBrig: 7.82643 |Brigada: 4 | |Nós visitados: 1 3486 1973 1268 654 2619 812 2337 1443 3588 1583 1358 2803 3261 39 1137 795 3375 3558 1949 3938 3
|TmpViagInspecBrig: 6.4962 |Brigada: 5 | |Nós visitados: 1 3486 1973 1268 654 2619 812 2337 1443 3588 1583 1358 2803 3261 39 1137 795 3375 3558 1949 3938 32
|TmpViagInspecBrig: 2.05952 |Brigada: 6 | |Nós visitados: 1 3486 1973 1268 654 2619 812 2337 1443 3588 1583 1358 2803 3261 39 1137 795 3375 3558 1949 3938 3
Não fechar janela do graphView se quer continuar no programa:

Deseja visualizar de forma automática as rotas: 1
Deseja controlar o aparecimento das rotas: 2
```

De salientar, que todas as restrições inerentes e já referidas são cumpridas (por exemplo nenhuma brigada trabalha mais horas do que aquelas que podem)

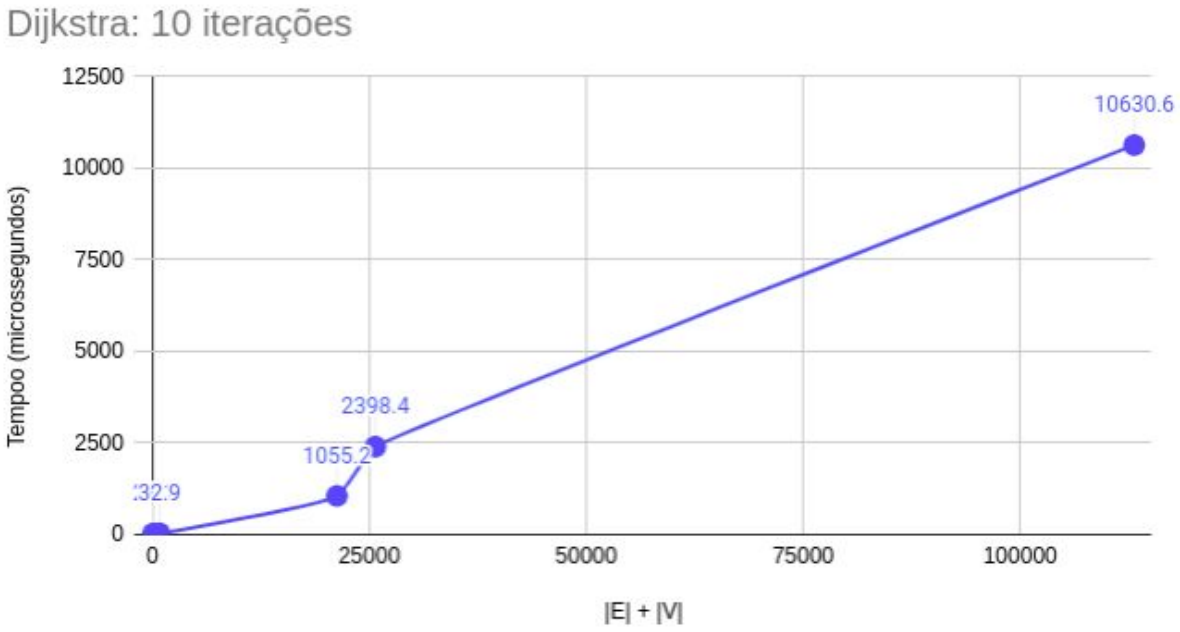
5. Complexidade dos Algoritmos Implementados

Iremos apresentar a complexidade empírica dos principais algoritmos tendo em conta os 6 mapas já referidos anteriormente visto que a temporal e espacial já foram dadas:

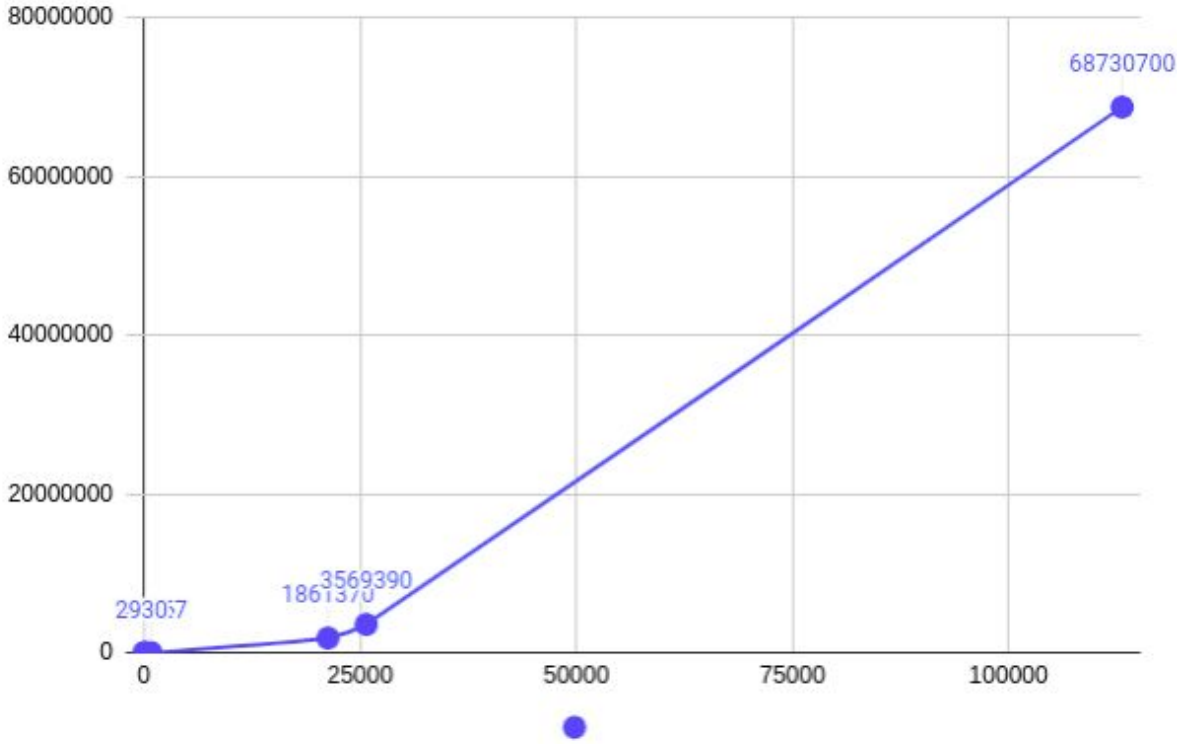
Complexidade Empírica Bellman Ford:



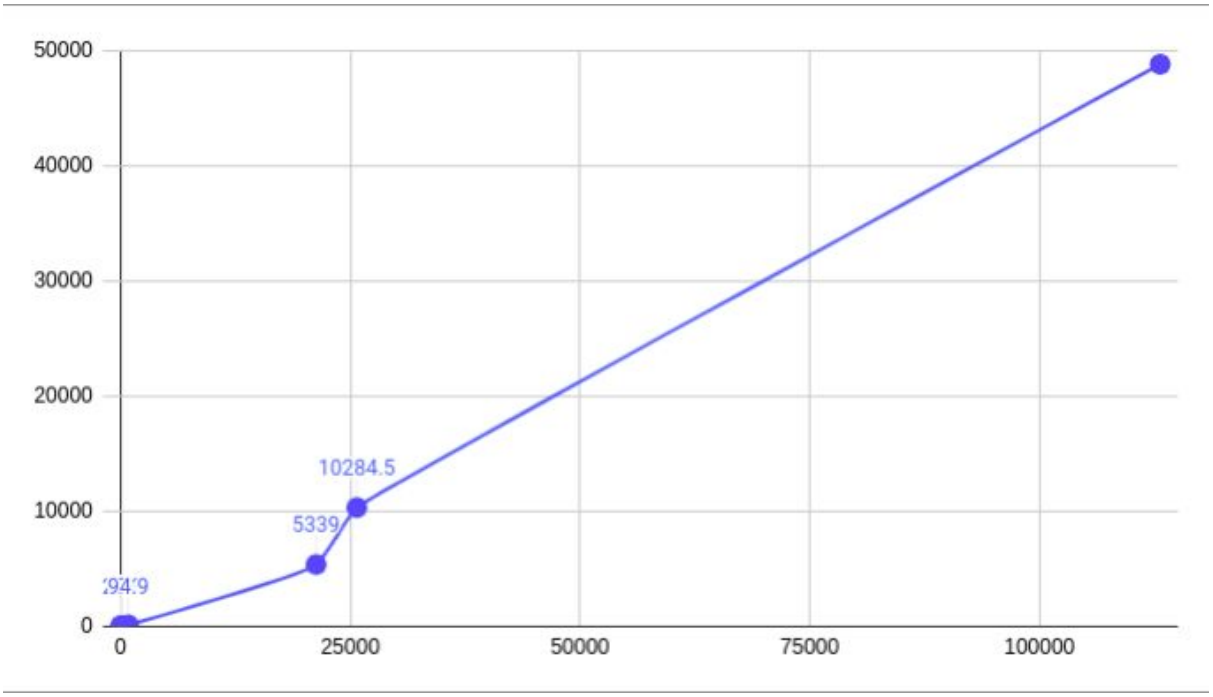
Complexidade Empírica Dijkstra:



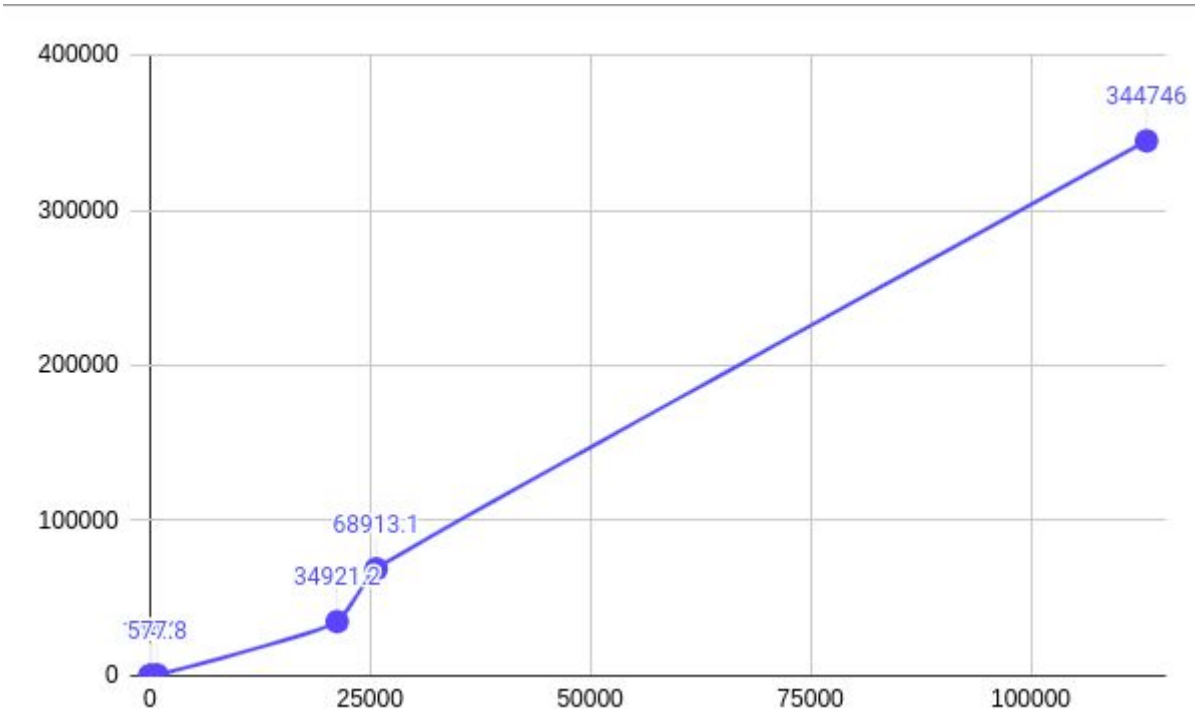
Complexidade Empírica Kosaraju:



Complexidade Empírica do algoritmo getRotaBrigada(...) e cálculo de rotas para uma brigada (aleatória) e todas as 7 brigadas da autoridade pública respectivamente:



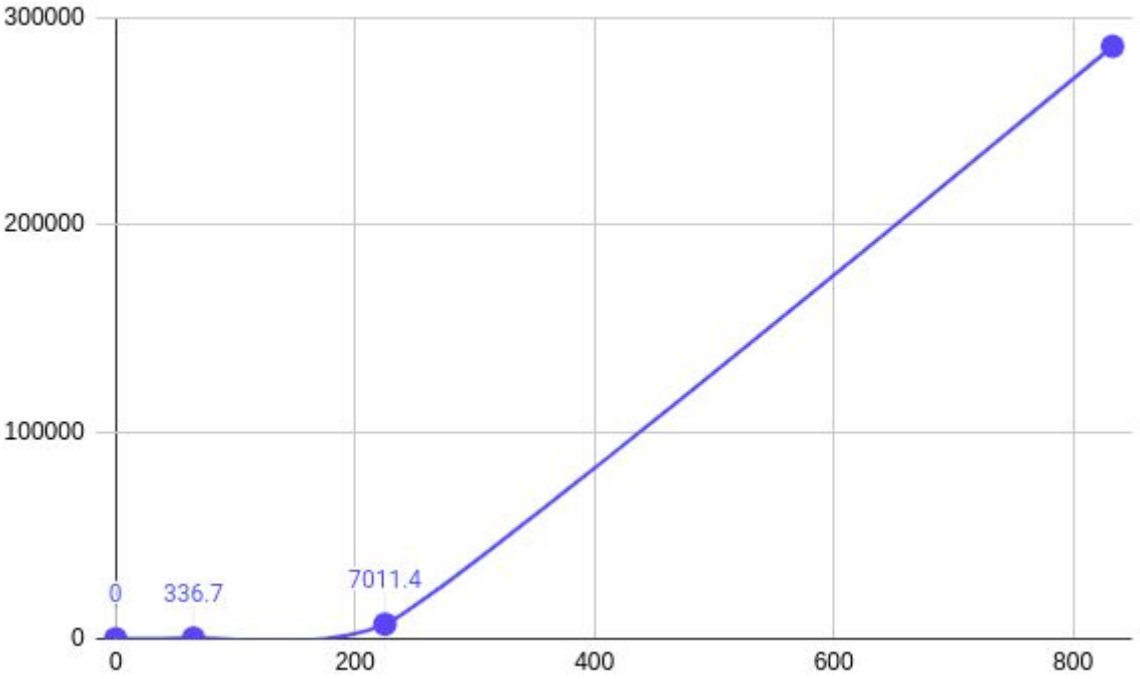
Legenda: 10 iterações onde a soma de tempo de viagem com tempo de inspeção é fator de escolha de agentes económicos, sem ter urgência ou limitação e zona por um raio escolhido como fatores de escolha e especialidades são respeitadas.



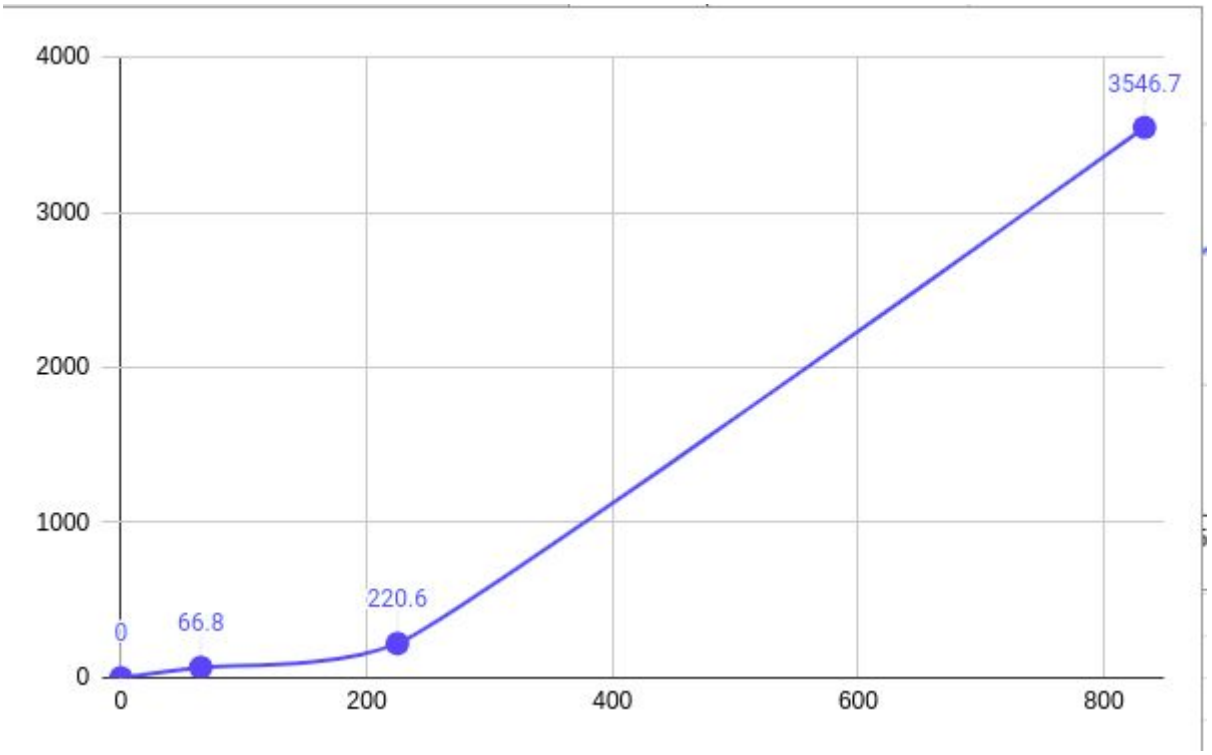
Legenda: 10 iterações onde a soma de tempo de viagem com tempo de inspeção é fator de escolha de agentes económicos, sem ter urgência ou limitação e zona por um raio escolhido como fatores de escolha e especialidades são respeitadas.

Complexidade Empírica Floyd-Warshall:

Primeiro processamento do algoritmo com 10 iterações 0.001 de margem de erro:



Complexidade empírica após algoritmo ter sido processado, com 10 iterações:



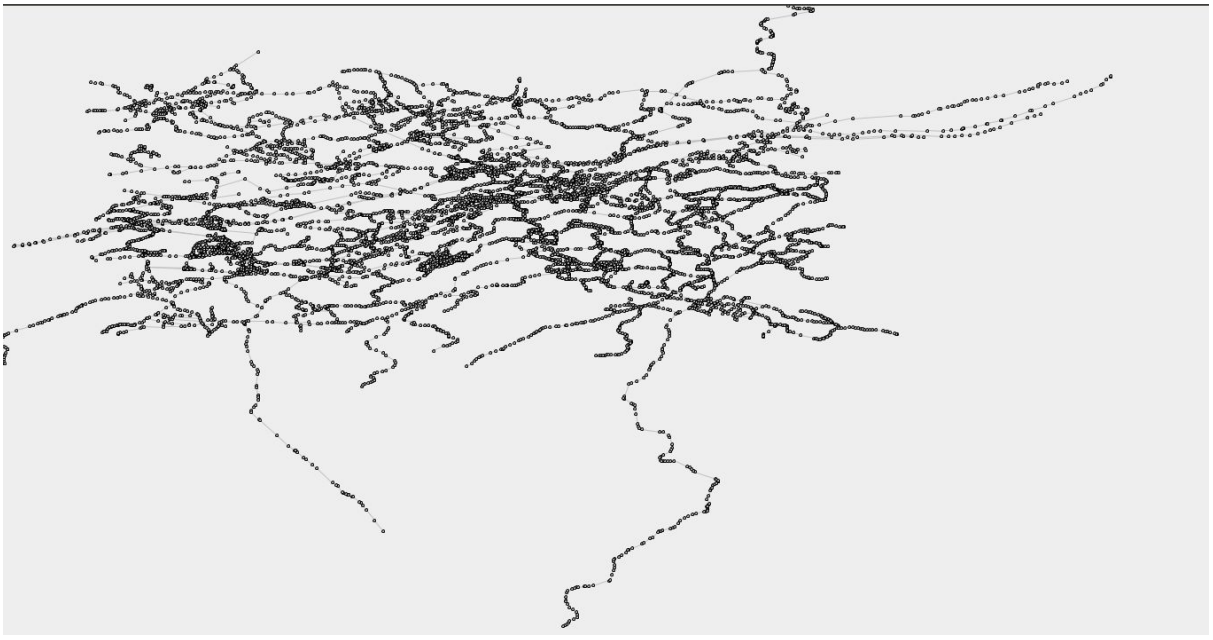
6. Conectividade dos Grafos Utilizados

Todos os grafos dos mapas utilizados, são constituídos por componentes fortemente conexas(neste caso utilizou-se a maior componente fortemente conexa calculada pelo algoritmo **Kosaraju**).

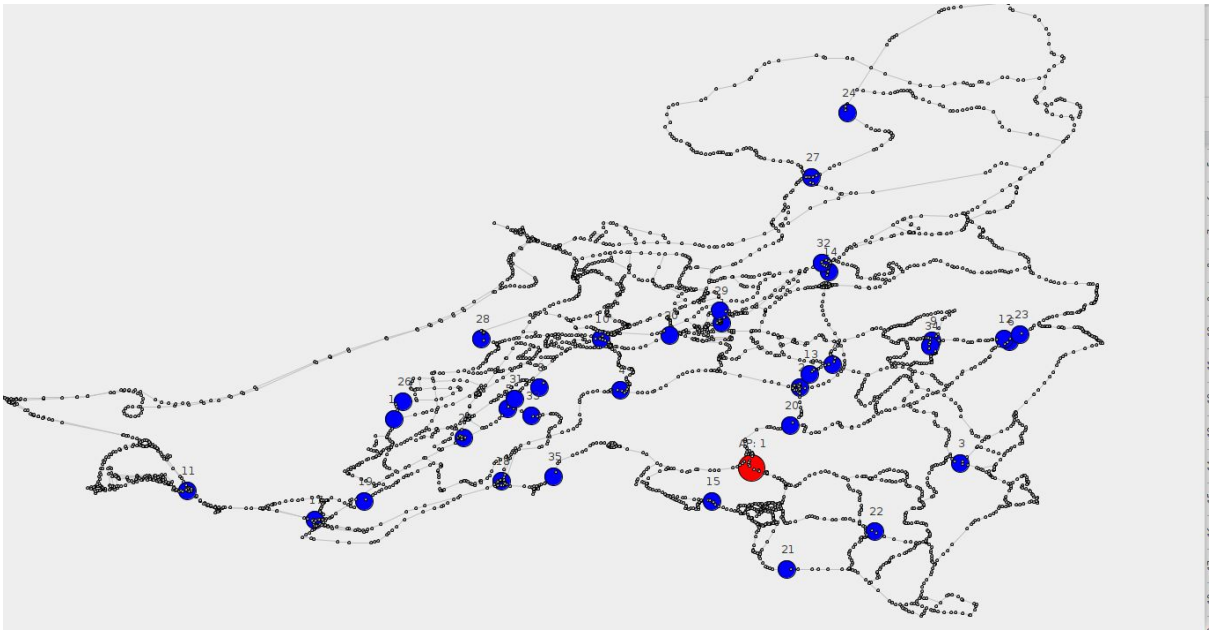
Ao utilizarmos o algoritmo Kosaraju, também garantimos que nenhum agente econômico correspondente a um vértice não acessível, é contabilizado para o cálculo de rotas. Desta forma resolvemos em parte o problema de pouca acessibilidade de algum agente econômico.Outra solução, seria considerar vértices de articulação.Mas achámos que tendo em conta o contexto real, e os mapas relativamente pequenos, não teríamos grande ganho ao considerar isso.

Em baixo teremos fotos do exemplo do mapa (versão full de Penafiel, e o resultado após obter o mapa final (a versão strong) utilizando o algoritmo **Kosaraju:**

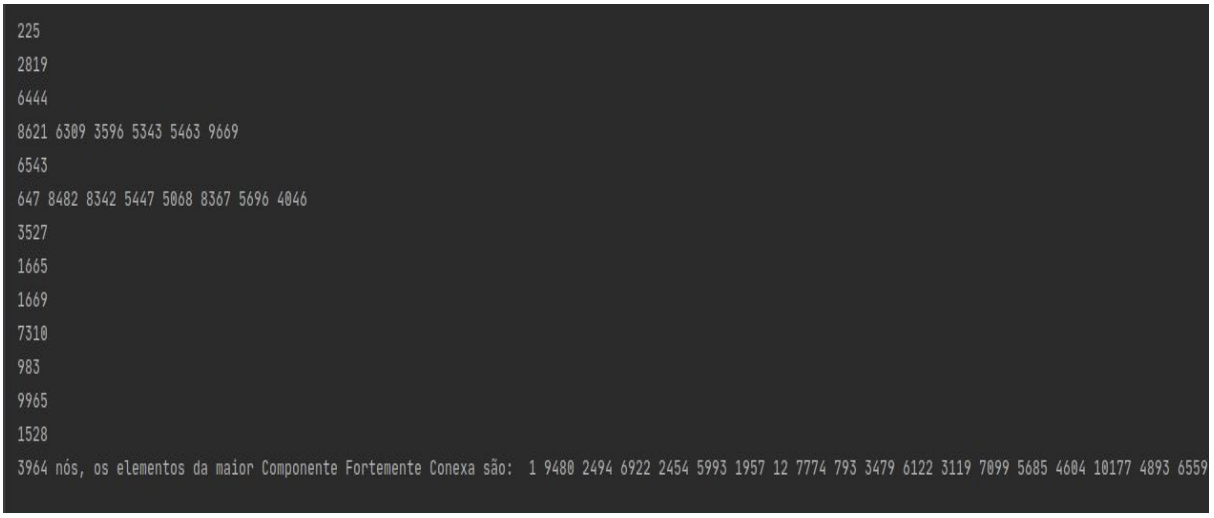
Versão full:



Versão strong (com a maior componente fortemente conexa)**:**



Algumas componentes fortemente conexas e a maior utilizada para obter o pretendido mostrado em cima:



7. Conclusão

7.1. Dificuldades Encontradas e Melhorias Futuras na primeira entrega

Na perspectiva de solução para o 2º objectivo, parece-nos que a tarefa de selecionar a rota óptima com o maior número de agentes económicos (inspecionados), terá de ser melhor trabalhada e assim ser implementada da forma mais eficiente possível e que permite selecionar melhor os agentes económicos a serem inspecionados, obtendo rota ótima de tempos de viagem mínimos , mas que a mesma tenha sempre o maior número de agentes possivel. Algo que não ficou a nosso ver, totalmente claro de ser alcançado para toda a possível diversidade de restrições que podemos ter no nosso problema.

7.2. Dificuldades Encontradas e Melhorias Futuras na segunda entrega

Penso que a existência de heurísticas e meta-heurísticas que no futuro aprenderemos podem melhorar imenso a escolha de rotas, como por exemplo opt-2.

Conclusão:

Achamos que o objectivo deste trabalho foi cumprido, visto que temos estratégias em que conseguimos num dia inspecionar 32 agentes económicos de um total de 35 existentes com apenas 6 brigadas. Outras estratégias que temos também podem ser interessantes para por exemplo permitir inspecionar certas zonas dadas como mais críticas por terem por exemplo agentes económicos como muitas queixas graves e ou conjunto na mesma zona os mesmo terem altas pontuações de urgência de inspeção.

Temos também de referir que corrigimos alguns problemas nos dados de saída e na forma como iamos verificar componentes fortemente conexas.

8.Tarefas executadas pelos membros do grupo

Na primeira entrega, todos contribuíram com ideias e pesquisas relevantes. Contudo, reconhece-se que o Flávio foi o elemento que mais contribuiu de uma forma relevante, uma vez que soube escrever com uma notação mais "matemática" e rigorosa. Não obstante, houve esforço dedicado por parte de todo o grupo com o intuito de fazer um bom trabalho. Sendo assim, enumeram-se de seguida as tarefas realizadas por cada um.

Luísa Maria Mesquita Correia

- 1ª Entrega:
  - Estruturação do relatório e revisão final (corrigindo eventuais erros)
  - Descrição do tema (agora incluída na introdução)
  - Identificação dos casos de utilização e funcionalidades
  - Pesquisa das várias atividades económicas existentes
  - Contribuição com ideias para a resolução do problema
- 2ª Entrega:
  - Efetuou 20% da entrega

Flávio Lobo Vaz

- 1ª Entrega:
  - Introdução (versão mais extensa)
  - Formalização do problema
  - Perspectiva de solução
  - Escrita com notação matemática e rigorosa
  - Dificuldades encontradas e melhoria futuras
  - Contribuição com ideias para a resolução do problema
- 2ª Entrega:
  - Efetuou 60% da 2ª entrega.

Amanda de Oliveira Silva

- 1ª Entrega:
  - Dados de entrada e de saída numa fase inicial
  - Perspectiva de solução numa fase inicial

        - Pesquisa de algoritmos para resolver o problema
        - Contribuição com ideias para a resolução do problema
    - 2ª Entrega
        - Efetuou 20% da 2ª entrega.

9. Bibliografia

(1) http://www.cm-porto.pt/direccoes-municipais/departamento-municipal-de-fiscalizacao
(2) https://mathworld.wolfram.com/TravelingSalesmanProblem.html
(3) https://mathworld.wolfram.com/NP-HardProblem.html
(4) https://mathworld.wolfram.com/HamiltonianCycle.html
(5) slides aulas teóricas, Algoritmos em Grafos: Caminho mais curto (Parte II).
(6) https://pt.wikipedia.org/wiki/Algoritmo_de_Dijkstra
(7) slides das aulas teóricas, Algoritmos em Grafos: Caminho mais curto (Parte I).
(8) slides aulas teóricas, Algoritmos em Grafos: Caminho mais curto (Parte II).
(9) slides das aulas teóricas, Algoritmos em grafos : Conectividade.
https://en.wikipedia.org/wiki/Nearest_neighbour_algorithm
https://www.geeksforgeeks.org/
Slides das aulas em geral.