

Mestrado Integrado em Engenharia Informática e Computação

Conceção e Análise de Algoritmos

Central de atendimento de urgências

Turma 4 / Grupo 9

Carlos Miguel da Silva de Freitas (201504749) - up201504749@fe.up.pt

Luís Noites Martins (201503344) - up201503344@fe.up.pt

Rui Emanuel Cabral de Almeida Quaresma (201503005) - up201503005@fe.up.pt

9 abril 2017

Índice

Introdução	3
Identificação do Problema	4
Descrição do Problema	5
Input	5
Output	5
Dados de entrada	6
Restrição	6
Objetivo	6
Formalização do Problema	7
Input	7
Output	7
Objetivo	7
Restrição	7
Solução implementada	8
Algoritmos	8
Funcionamento do programa	10
Análise da complexidade	13
Avaliação analítica da complexidade temporal	13
Avaliação analítica da complexidade espacial	13
Avaliação empírica da complexidade temporal	14
Lista de casos de utilização	15
Diagrama de classes	16
Principais dificuldades	17
Soluções encontradas	17
Contribuição dos membros do grupo	18
Conclusão	19

Introdução

Este trabalho foi realizado no âmbito da Unidade Curricular de Conceção e Análise de Algoritmos do 2º ano do MIEIC.

O objetivo é recorrer aos algoritmos de cálculo do caminho mais curto, abordados nas aulas, para encontrar uma solução para um tema que nos foi proposto.

Neste relatório é descrito o problema, feita a sua formalização, bem como apresentada a solução implementada. São ainda elencadas as principais dificuldades sentidas na elaboração do trabalho e as soluções encontradas para as ultrapassar.

No final, apresenta-se uma breve conclusão, que inclui uma apreciação sobre a aprendizagem decorrente deste trabalho.

Identificação do Problema

Uma central de atendimento de urgências é responsável por receber chamadas telefônicas realizadas para o número de urgência (e.g. 112) e desencadear de forma coordenada os processos e os meios adequados ao tratamento da situação de emergência que está na origem da chamada.

Para uma maior eficiência de resposta aos diversos tipos de emergência, é necessário que seja enviado ao local da ocorrência (normalmente coincidente com o local da chamada) quer o veículo que se encontre mais próximo, quer a(s) entidade(s) mais adequada(s), consoante o tipo e a gravidade da situação.

Dado este problema, procuramos desenvolver uma interface que permita mostrar ao utilizador, mediante a localização da ocorrência num mapa e a seleção do tipo de emergência, o itinerário mais curto desde o(s) veículo(s) de emergência até ao local da ocorrência.

O caminho mais curto consiste na distância mais reduzida desde o local de um veículo até ao local da emergência.

Descrição do Problema

Pretende-se identificar qual(ais) o(s) veículo(s), de entre os disponíveis, que se encontra(m) mais perto do local da ocorrência (dado pelo utilizador) e de acordo com o tipo de emergência (dado pelo utilizador), mostrando de seguida o itinerário mais curto do(s) veículo(s).

Input

Criação de um grafo, $G = (V, A)$, de cruzamentos e ruas em que:

- V (vértices): representam pontos de interceção de duas ruas, ou seja, cruzamentos;
- A (arestas): ligações entre dois cruzamentos (distâncias entre dois vértices);
- Nó/ Rua: representa o local da ocorrência;
- Tipo: representa o tipo de emergência;
- Nível: gravidade da emergência.

Output

Nós (cruzamentos) pelos quais o(s) veículo(s) passa(m) desde o local onde se encontra(m) até ao local da emergência, pertencentes ao caminho mais curto/otimizado.

Dados de entrada

- Cinco ficheiros que contêm os nós (cruzamentos) do mapa, com os ids, coordenadas x e y dos mesmos:
 - Um ficheiro que contém os nós onde não está localizado nenhum veículo ou hospital, tem os ids, o x e o y dos nós;
 - Um ficheiro que contém os nós onde estão localizadas as ambulâncias;
 - Um ficheiro que contém os nós onde estão localizados os carros dos bombeiros;
 - Um ficheiro que contém os nós onde estão localizados os carros da PSP;
 - Um ficheiro que contém os nós onde estão localizados os hospitais.
- Um ficheiro que contém as arestas, ou seja, as ligações entre os diferentes vértices, com os ids das mesmas e os ids dos nós de início e de fim de cada uma.
- Um ficheiro que contém as ruas, ou seja, conjuntos de arestas que podem ser unidireccionais ou bidireccionais, com os ids e o nome das mesmas, a indicação sobre se são ou não bidireccionais e ainda os ids dos nós que as compõem.

Restrição

Para que o resultado obtido seja o melhor, é necessário que o grafo, representativo do mapa, seja fortemente conexo entre pontos em que existem os veículos disponíveis (polícias, bombeiros, INEM e hospitais) e o ponto em que a ocorrência acontece.

Objetivo

Providenciar ao utilizador quais os veículos de emergência mais próximos, de acordo com o solicitado, e o respetivo percurso até ao local da chamada.

Formalização do Problema

Input

$G(V, A)$,

V : cruzamentos

A : ligações entre cruzamentos (distancia da ligacao)

C : ponto da chamada

T : Tipo de emergência

Output

Caminho de cada um dos veiculos de emergência = $\{V_i\}, i = 1 \dots n$

Id dos nós onde estão os veiculos que assistirão a emergência

Objetivo

Min (distancia):

$$distancia = \sum_{i=1}^n (A_{ij}), ij \in Caminho$$

Restrição

$\forall_i \in Caminho = \{V_i\}$

$\sim A_j \in Caminho : V_i = V_j \wedge i \neq j$

Solução implementada

Algoritmos

Optamos por implementar dois algoritmos, o de Dijkstra e o de Floyd-Warshall.

O algoritmo de Dijkstra, que encontra o caminho mais curto num grafo dirigido ou não dirigido, tem a restrição de só poder ser usado em grafos com pesos positivos, situação que se verifica no tema do trabalho.

Este algoritmo é um algoritmo ganancioso, ou seja, toma as decisões que parecem melhores no momento, o que lhe permite determinar o conjunto de melhores caminhos intermediários. O peso de cada aresta está associado à distância, calculada através das coordenadas (x e y) de cada cruzamento.

Assim, o algoritmo de Dijkstra, partindo de um vértice, itera o grafo e calcula qual a menor distância e respetivo caminho desse mesmo vértice a todos os outros. Como referido anteriormente, este algoritmo é ganancioso, e como tal, durante o cálculo para cada vértice escolhe, de entre os que lhe estão adjacentes, aquele que se encontra a uma menor distância. Para manter sempre os vértices ordenados pela menor distância, recorre-se a uma fila de prioridade.

```
1: function Dijkstra(Graph, source):
2:   for each vertex v in Graph:           // Initialization
3:     dist[v] := infinity                 // initial distance from source to vertex v is set to infinite
4:     previous[v] := undefined           // Previous node in optimal path from source
5:   dist[source] := 0                     // Distance from source to source
6:   Q := the set of all nodes in Graph    // all nodes in the graph are unoptimized - thus are in Q
7:   while Q is not empty:                 // main loop
8:     u := node in Q with smallest dist[ ]
9:     remove u from Q
10:    for each neighbor v of u:           // where v has not yet been removed from Q.
11:      alt := dist[u] + dist_between(u, v)
12:      if alt < dist[v]                  // Relax (u,v)
13:        dist[v] := alt
14:        previous[v] := u
15:  return previous[ ]
```


O algoritmo de Floyd-Warshall, que encontra o caminho mais curto num grafo dirigido, é mais aconselhado para grafos densos. Pode ser usado em grafos com pesos positivos, bem como com pesos negativos. O peso de cada aresta está associado à distância e é calculado através das coordenadas (x e y) de cada cruzamento.

Este algoritmo dispensa grande parte do seu cálculo em pré processamento, através do cálculo da distância entre todos os vértices. Estes cálculos servem para preencher duas matrizes (uma relativa à distância entre cada um dos nós, e outra relativa ao caminho para essa mesma distância), as quais permitirão que, com facilidade, se obtenham os valores necessários à execução do programa. Por exemplo, recorrendo à primeira matriz sabe-se qual o veículo que com mais facilidade chegará ao local da ocorrência, sendo que a segunda matriz mostra qual o caminho que esse mesmo veículo terá que percorrer para satisfazer esse pedido.

```
for i = 1 to N
  for j = 1 to N
    if there is an edge from i to j
      dist[0][i][j] = the length of the edge from i to j
    else
      dist[0][i][j] = INFINITY

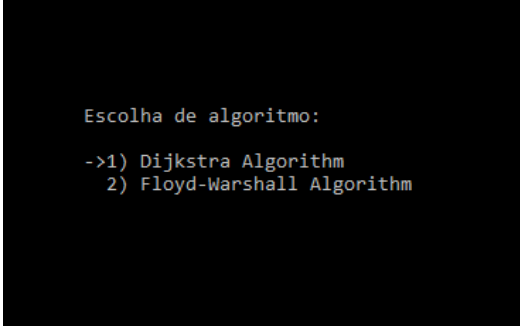
for k = 1 to N
  for i = 1 to N
    for j = 1 to N
      dist[k][i][j] = min(dist[k-1][i][j], dist[k-1][i][k] + dist[k-1][k][j])
```

No nosso trabalho, o algoritmo de Dijkstra é mais eficiente, dado que temos um mapa com poucos nós (pouco denso). Se usássemos o algoritmo de Floyd-Warshall perderíamos eficiência, dado que ele criaria uma matriz com o peso da ligação de cada dois nós, o que para um grafo pouco denso não seria o melhor.

Funcionamento do programa

Para implementar a solução para o problema em causa, foram utilizados os algoritmos acima referidos.

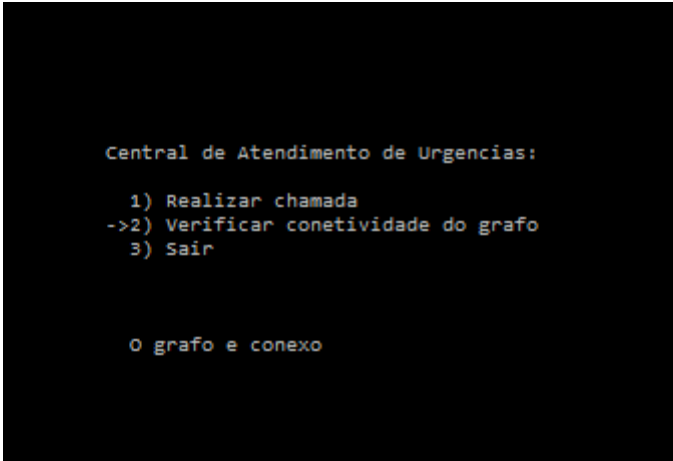
Assim, ao iniciar o programa o utilizador é questionado sobre qual o algoritmo que pretende que seja utilizado para a determinação do caminho mais curto.



```
Escolha de algoritmo:  
->1) Dijkstra Algorithm  
   2) Floyd-Warshall Algorithm
```

É possível verificar a conectividade do grafo criado com base nos ficheiros lidos. Esta verificação é feita com base na execução de uma pesquisa em profundidade em pós ordem, tanto para o grafo original, como para o grafo original invertido.

Desta forma, obtém-se de cada um dos grafos, os componentes fortemente conexos. Por fim, basta comparar se os componentes conexos de ambos os grafos são iguais, concluindo assim se o grafo é totalmente conexo ou não. Esta verificação permite saber se a partir de um qualquer vértice se consegue alcançar qualquer um dos vértices pertencentes ao grafo original. Esta condição é essencial para garantir que os veículos de emergência conseguem alcançar o local da ocorrência.



```
Central de Atendimento de Urgencias:  
  
1) Realizar chamada  
->2) Verificar conectividade do grafo  
3) Sair  
  
O grafo e conexo
```

Por outro lado, o utilizador poderá realizar uma chamada, na qual terá que escolher um dos tipos de ocorrência previstos (Acidente, Crime ou Incêndio) e o nível dessa mesma ocorrência (1,2 ou 3).

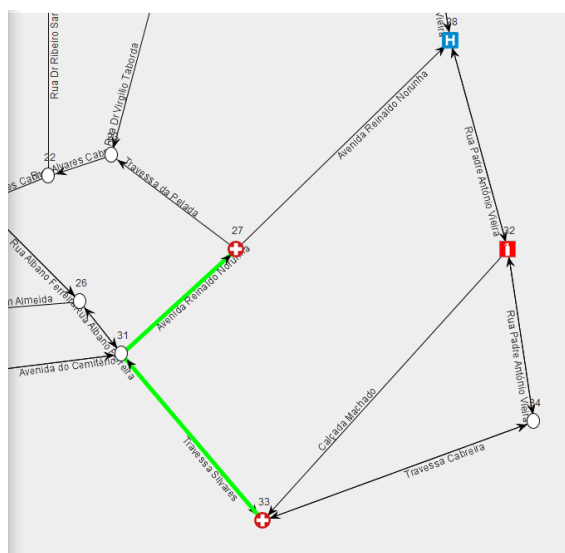
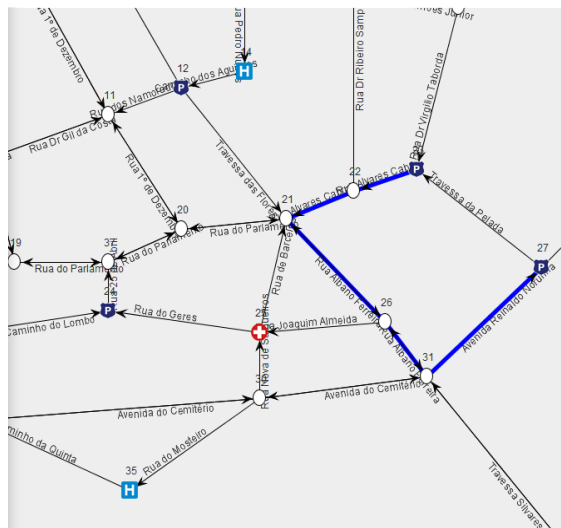
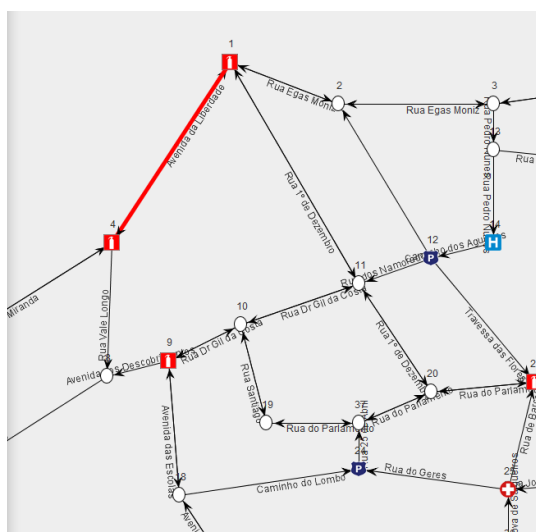
```
Tipo de Urgencia:
->1) Incendio
   2) Crime
   3) Acidente
```

```
Tipo de Urgencia:
->1) Nivel 1
   2) Nivel 2
   3) Nivel 3
```

Terá que indicar também a sua localização, através da respetiva rua. É posteriormente selecionado, de forma aleatória, um dos nós dessa rua.

```
Lista de ruas
-> Rua Dr Figueredo
   Caminho da Quinta
   Rua do Mosteiro
   Rua dos Namorados
   Rua Dr Ribeiro Sampaio
   Travessa das Flores
   Rua Santiago
   Rua Joaquim Almeida
   Caminho do Lombo
   Rua 25 de Abril
```

A partir deste ponto, desencadeia-se todo o processo de seleção do melhor percurso e veículo(s) a ser(em) chamado(s) para a emergência, sendo que os veículos requisitados mudarão a sua posição para o nó onde se deu a emergência (exceto a ambulância, que poderá terminar no hospital para onde levou o doente). Estes mesmos veículos ficam inativos durante 2 chamadas consecutivas, voltando a ficar disponíveis após esse período.



Análise da complexidade

Avaliação analítica da complexidade temporal

A complexidade temporal do algoritmo de Dijkstra é $O([vértices + arestas] + [vértices + arestas] * \log(vértices))$, sendo que o mesmo algoritmo será executado tantas vezes quantos os veículos desse(s) tipo(s) disponíveis. Por exemplo, considerando que estamos perante um Acidente de tipo 2, são chamadas 2 ambulâncias e 1 polícia.

Já no caso do algoritmo do Floyd-Warshall, o algoritmo é executado apenas uma vez no início do programa, preenchendo 2 matrizes, uma com as distâncias entre cada 2 pontos, e a outra com os vértices intermédios entre cada 2 pontos, para que seja possível obter todos os nós pertencentes ao caminho entre os 2 nós que se pretende. Deste modo, este algoritmo tem uma complexidade temporal $O([vértices]^3)$.

Avaliação analítica da complexidade espacial

Ambos os algoritmos utilizados têm complexidade espacial de $O(vertices^2)$, já que para cada vértice é guardada a informação relativa *as arestas adjacentes que terão um apontador para o vértice destino.

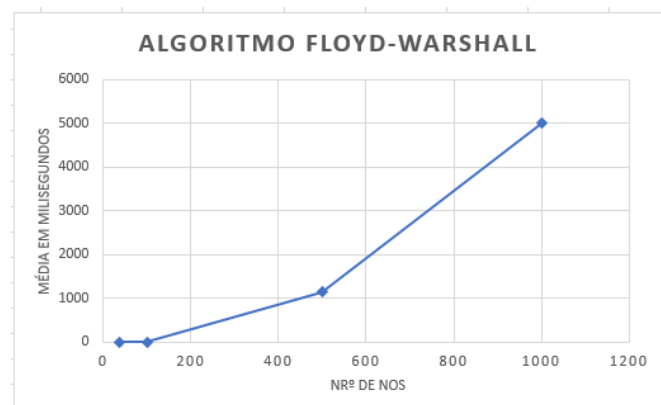
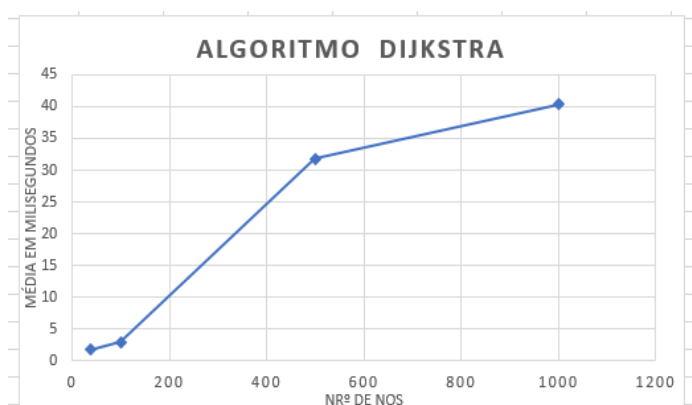
Avaliação empírica da complexidade temporal

Para analisar a complexidade temporal, medimos os tempos de execução do programa, usando diferentes dados de entrada (diferente número de vértices e arestas; diferente número de veículos e hospitais e localização dos mesmos).

Como forma de comparação, para cada um dos algoritmos foram realizados os mesmos testes, fazendo variar o tamanho do mapa. Posteriormente, foi calculado o valor médio para cada conjunto de testes de cada mapa.

Obtivemos desta forma os seguintes resultados:

Nr Nós	Tempo do método (ms)	
	Dijkstra	Floyd-Warshall
37	1.8	2.2
100	3	5.6
500	31.8	1142.8
1000	40.4	5006



Lista de casos de utilização

- Leitura dos dados de ficheiros representativos das componentes de um mapa;
- Identificação dos veículos a deslocar através da verificação do tipo de emergência selecionada e dos respetivos recursos necessários;
- Identificação do caminho mais curto desde os veículos selecionados até ao local da chamada, através da utilização do caminho obtido pela execução dos algoritmos;
- Possível retorno de um dos veículos ao hospital mais próximo através do caminho mais curto, verificação do hospital mais próximo e do caminho que a ambulância terá que percorrer para chegar até lá;
- Visualização do mapa usando o GraphViewer;
- Apresentação do caminho mais eficiente usando o GraphViewer, através do realce das arestas por onde os veículos passarão.

Diagrama de classes



Principais dificuldades

No início do projeto, deparamo-nos com alguns entraves à sua execução, enunciados de seguida:

- Como obter um mapa do site *openstreetmaps* e como convertê-lo para ficheiros de texto, e fazer a respetiva leitura.
- Perceber qual o melhor algoritmo a utilizar.

Soluções encontradas

Decidimos criar o nosso próprio mapa (criação de ficheiros de nós, arestas e ruas).

Com recurso a alguma pesquisa, nomeadamente os slides das aulas teóricas e soluções das aulas práticas, conseguimos compreender e implementar os algoritmos pretendidos.

Contribuição dos membros do grupo

Todos os membros contribuíram de forma equitativa para a realização do projeto. Assim, com exceção do relatório, que foi elaborado com o contributo dos três elementos do grupo, a distribuição do trabalho foi a seguinte:

Carlos Freitas:

- Desenvolvimento da interface do utilizador (menus e display do grafo);
- Cálculo de verificação de conectividade do grafo.

Luis Martins:

- Implementação dos algoritmos utilizados para a escolha dos veículos a serem utilizados em cada situação e cálculo do caminho dos mesmos.

Rui Quaresma:

- Criação e leitura dos ficheiros utilizados tanto para a realização de testes de tempo de execução dos algoritmos como para demonstração do programa.

Conclusão

Perante o problema apresentado no início deste relatório, pretendia-se obter, através de um algoritmo eficiente, a melhor combinação de veículos e percursos, considerando uma ocorrência num qualquer nó do grafo. Assim, com recurso a 2 algoritmos diferentes foram obtidos resultados satisfatórios.

Concluimos que o algoritmo mais eficiente para o nosso tema seria o de Dijkstra, já que o número de nós existentes com veículos de emergência e hospitais, é muito inferior ao número total de nós, o que faz com que uma grande parte do tempo de cálculo do algoritmo Floyd-Warshall constitua um desperdício e seja dispendioso demais para a resolução do problema.

O tema deste trabalho representa uma situação que, embora aqui seja analisada de forma simplificada, corresponde a uma situação real, que é muito complexa e muito importante para a sociedade. Assim, consideramos que foi um desafio interessante, que permitiu uma melhor compreensão dos algoritmos e, ao mesmo tempo, a perceção da sua aplicabilidade em situações reais.