FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Routing multimodal para transporte coletivo

Projeto de CAL 2018 -- Engenharia Informática e Computação

Turma 5

Grupo 2

Tema 8

Regente: Rosaldo José Fernandes Rossetti

- Carolina Vasconcelos Castro Azevedo --- up201506509 --- up201506509@fe.up.pt
- Eduardo Luís Pinheiro da Silva --- up201603135 --- up201603135@fe.up.pt
- Tomás Nuno Fernandes Novo --- up201604503 --- up201604503@fe.up.pt

MIEIC – Conceção e Análise de Algoritmos

Abril, 2018

Índice

Introdução e Descrição do Tema 3
Identificação do Problema 4
Formalização do Problema 5
Solução do Problema 7
Estruturas de dados 11
Casos de utilização 12
Dificuldades encontradas 12
Esforço e dedicação de cada elemento 13
Conclusão 14

Introdução e descrição do tema

No âmbito da unidade curricular Conceção e Análise de Algoritmos, foi-nos proposta como 1ª parte do trabalho prático a realização de um projeto cujo tema é ***Routing* multimodal para transporte coletivo.**

Introduzindo um pouco o problema a solucionar, o tema aborda a criação de um sistema de navegação capaz de gerar opções de itinerário entre uma origem e um destino tendo em conta a combinação dos diversos modos de transporte, em trechos específicos do itinerário.

Na nossa aplicação, o utilizador apenas preciso de introduzir a origem e o destino do itinerário para saber qual a melhor combinação de caminhos a percorrer quer nos meios de transporte quer a pé.

De maneira a simplificar e a influenciar a decisão do utilizador, o itinerário pode ser avaliado através de critérios como custo, tempo e distância, entre outros.

Para tal, decidimos construir uma estrutura de grafos contendo nós e arestas, sendo que cada nó representa uma interseção entre duas ou mais arestas e cada uma destas representa uma estrada, linha ferroviária ou vias especificamente destinadas aos transportes públicos (autocarro).

Identificação do Problema

Principiando por formalizar o problema em questão, foi-nos pedida a criação de um sistema de navegação gerador de itinerários entre uma origem e um destino dependendo do(s) critério(s) especificado(s) pelo utilizador.

Optámos por utilizar um grafo **G = (V, E)**, na qual os vértices (ou nós) **V** representam as conexões entre as arestas (**E**), que por sua vez correspondem a ruas, ligações de metro ou autocarros. Estas possuem a elas associadas um peso (*weigth*) que representa a distância real desde o nó de origem até ao nó final, um tempo (*time*) que representa o tempo médio que se leva a percorrer essa aresta (dependendo do meio de transporte) e um objeto da classe *Info* que contém informações adicionais (id, nome, …).

Usufruímos do *parser* fornecido para podermos utilizar dados de entrada reais, obtendo assim ficheiros de introdução de dados contendo Ids e coordenadas das linhas de transporte dos respetivos meios. Além deste, baseámo-nos também num *parser* modificado por um aluno do MIEIC, João Carvalho, de modo a conseguirmos informações adicionais, tal como o tipo de arestas apresentadas (rua, metro ou autocarro).

Durante a realização deste projeto objetivámos sempre uma simplicidade adjacente na interface aplicação-utilizador com o fim de proporcionar uma maior facilidade de escolha dos melhores trajetos tendo em conta os diversos critérios especificados. Para tal, a aplicação é simples de manipular, visto que os menus são acedidos apenas pela introdução de um número ou da inserção dos nomes das ruas pretendidas para análise. Como *output* do sistema de navegação, apresentamos uma resolução gráfica do problema, com os vértices de origem, destino e no caminho percorrido representados a uma cor diferente dos restantes vértices.

Formalização do Problema

Intencionando resolver da melhor forma o problema, formalizámo-lo para atingir os objetivos pretendidos.

Dados de entrada:

Como *input* é criado 1 ou 2 grafos dependendo da escolha do utilizador.
No grafo marcado como experimental são usados 4 ficheiros de texto com informação real obtida através do *parser* que serão lidos pela aplicação.
No grafo marcado como de teste, a aplicação constrói um grafo simplificado desenhado especialmente para testar as diferentes restrições e avaliar os resultados. Neste, os nós têm a sua identificação por cima deles.
Em ambos os grafos, as arestas correspondentes a caminhos realizados a pé são representadas a vermelho, as que correspondem às linhas do metro possuem cor verde e por fim as que representam as "linhas" de autocarros são coloridas a azul.

Dados de saída:

Como dados de saída temos um conjunto de nós que representam o melhor caminho entre a origem e o destino especificados pelo utilizador. Este caminho é talhado às preferências indicadas pelo utilizador, podendo haver vários cenários:

- Caminho mais curto
    - Com restrição de taxas monetárias.
    - Com transporte preferencial.
- Caminho mais rápido
    - Com transporte preferencial.
- Melhor caminho
    - Com restrição de taxas monetárias.
    - Com transporte preferencial.

De notar que as opções adicionais podem não ser selecionadas. Além disto, quando o utilizador selecionar o melhor caminho, haverá uma preferência pelos transportes públicos de modo a reduzir a distância percorrida a pé por ele.
Em relação ao caminho mais rápido, decidimos implementar apenas a opção de transporte preferencial, visto que a redução de taxas monetárias não se aplicaria num cenário real em que o objetivo era obter o caminho mais rápido de um ponto a outro.

Funções Objetivo:

As duas funções objetivo resumem-se à minimização do tempo total e à minimização do espaço percorrido, tendo em conta as opções escolhidas pelo utilizador e a distância percorrida a pé.

- $\sum_{n0}^{n} d\,(Vk)$
- $\sum_{n0}^{n} t\,(Vk)$

Restrições:

No grafo de teste, visto que tanto os nós como as arestas têm nomes simbólicos apenas, é aconselhável a utilizar a escolher o ponto de partida e destino através dos Ids dos pontos em si, visto que estes estão representados em cima dos nós correspondentes.

Em relação ao grafo construído com os dados do OpenStreetMap, como estes dados são de uma área limitada pode haver certos nós que estão isolados dos outros, sendo impossível chegar a estes.

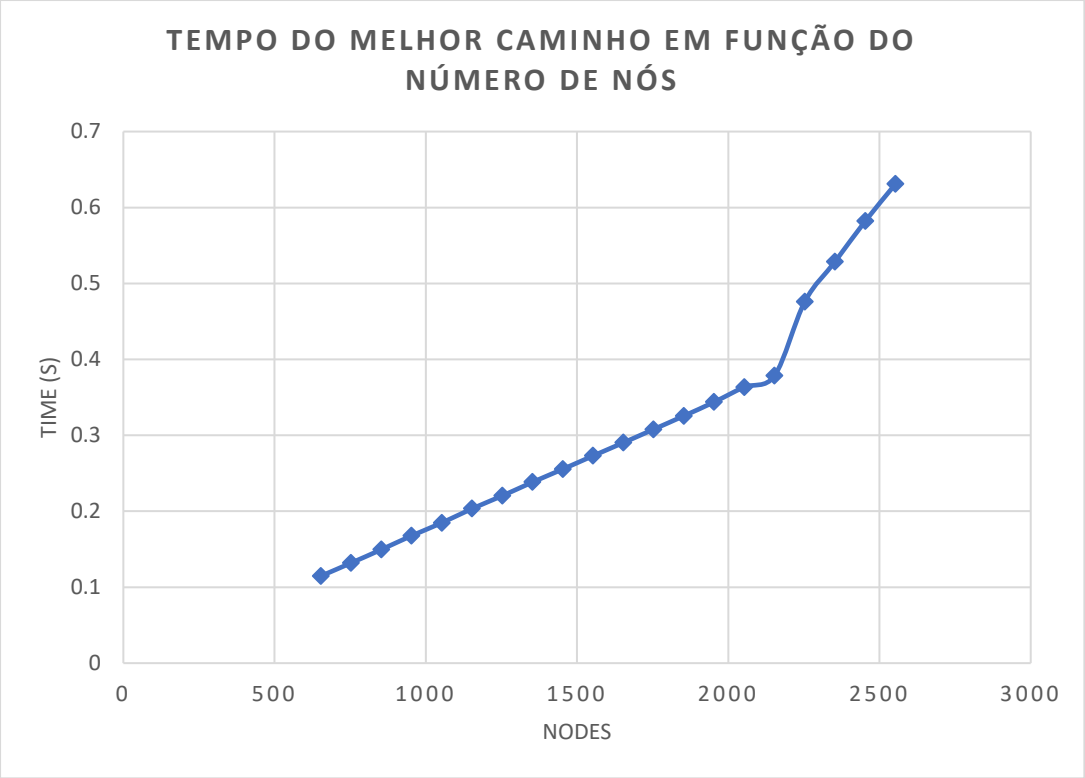
Solução do problema

Geração do grafo:

O primeiro obstáculo a ultrapassar foi a geração dos grafos, nomeadamente o grafo construído com base nos dados do OpenStreetMap. Aqui, é feita uma leitura dos ficheiros gerados pelo *parser*, sendo os vértices adicionados primeiro, seguidos das arestas. Estas são representadas na janela gráfica a cores diferentes, dependendo do tipo de transporte a que estão associadas (vermelho para a pé, azul para autocarros e verde para metro). Além disto, a cada aresta está associado uma distância, calculada matematicamente usando as coordenadas GPS de cada vértice através da fórmula de Haversine, e um tempo, calculado através da velocidade média do meio de transporte e da distância.
No entanto, o *parser* por si só raramente gera ligações entre as linhas pedonais e as do metro. Para resolver isto, durante a adição das arestas, os vértices com ligação direta a linhas de metro são marcados com uma variável booleana. Após este passo, o conjunto de vértices é percorrido, parando em cada um dos vértices marcados. Em cada destas paragens, o conjunto de vértices é percorrido outra vez, sendo criadas ligações entre os vértices marcados e todos os restantes que estejam dentro de uma distância predefinida (25 metros), de modo a simular a entrada para as estações de metro.

Algoritmos utilizados:

Após uma análise de diversos algoritmos, concluímos que, apesar de se tratar de um algoritmo ganancioso, o algoritmo mais proveitoso para o nosso projeto seria o algoritmo de Dijkstra, visto que o nosso problema se pode reduzir a um de caminho mais curto em grafos (dirigido ou não), tendo uma complexidade temporal O (|V|*|E|*log(V)).



1ª Iteração:

Numa primeira fase, é percorrido o conjunto de vértices de modo a colocar todos os parâmetros usados no algoritmo em si com os seus valores padrão. Esta operação é realizada, portanto, em O(|V|) e é praticamente idêntica quer no cálculo do caminho mais curto (com ou sem opções) quer no cálculo do menor tempo de viagem. Além disto, as cores e tamanhos dos nós na componente gráfica voltam a ser repostos para os valores originais.

Cálculo do melhor caminho com limite de custo e transporte preferencial:

A seguir apresentamos a estrutura do algoritmo implementado (em pseudo-código) após a 1ª iteração mencionada em cima:

```
1 While Q ≠ ∅ do
2     V ← min(Q)
3     Processing(V) ← False
4     For each E ∈ adj(V) do
```

```
5       Distance_to_add ← weight(E)
6       Walk_penalty ← 0
7       If is_busStation(Info(E)) = false ^ is_trainStation(Info(E)) = false
8               Walk_penalty ← weight(E) / 2
9               Distance_to_add ← distance_to_add + walk_penalty
10      Else
11              If isLimitReached(V)
12                      LimitReached(getDest(E)) ← True
13                      continue
14              Else
15                      If isCounting(V) = false
16                              Couting(V) ← True
17                              TimeCount(V) ← 0
18                              Cost(V) ← Cost(V) + 1.20
19      If FavoriteTransport ≠ “”
20              If is_busStation(Info(E)) ^ FavoriteTransport = “Bus”
21                      Distance_to_add = Distance_to_add / 4
22      Else
23              If is_trainStation(Info(E)) ^ FavoriteTransport = “Train”
24                      Distance_to_add = Distance_to_add / 4
25      If Cost(V) > limit
26              If limitReached(V) = false
27                      Cost(V) ← 0
28                      Counting(V) ← False
29                      limitReached(V) ← True
30                      limitReached(Dest(E)) ← True
31                      continue
32      If Dist(Dest(E)) > Dist(V) + Distance_to_add
33              If isCouting(V)
34                      If (TimeCount(V) + Time(E)) / 60 > 1
35                              Counting(V) ← False
36              Else
37                      TimeCount(Dest(E)) ← TimeCount(V) + Time(E)
```

```
38                  Cost(Dest(E)) ← Cost(V)
39                  Dist(Dest(E)) ← Dist(V) + Distance_to_add
40                  Path(Dest(E)) ← V
41                      If isProcessing(Dest(E)) = False
42                          Processing(Dest(E)) ← True
43                          Insert(Q, Dest(E))
44                      Else
45                          DecreaseKey(Q, Dest(E))
```

No ciclo principal, começa-se por extrair o vértice do *Priority Queue* com menor distância até à origem. É iniciado de seguida o 2º ciclo, em que são estudadas cada uma das suas arestas.

A primeira verificação a ser efetuada é se a aresta corresponde a uma linha de metro ou de autocarros. Se não for este o caso, é adicionada uma penalidade por andar a pé à distância do vértice à origem. Esta penalidade tem o valor de metade do comprimento da aresta correspondente. No entanto, se a aresta corresponder a um dos destes casos, é verificado se o utilizador já atingiu o limite monetário estipulado. Se não, é verificado ainda se uma contagem de tempo está em progresso ou não. Em caso negativo, esta é iniciada.

A próxima etapa consiste em verificar se existem transportes preferenciais, e se sim averiguar se o tipo da aresta atual corresponde a esse transporte. Em caso afirmativo, a distância do vértice à origem será um quarto da original.

De seguida, é verificado se o custo atual não ultrapassa o limite imposto pelo utilizador, sendo que a aresta é ignorada se este for o caso. Em caso contrário, se a distância do vértice de destino (à origem) desta aresta for maior que a distância do vértice atual somado com a distância customizada previamente, é decidido adicionar este vértice de destino ao *Priority Queue*, sendo "herdado" por este o custo atribuído ao vértice atual. Além disto, é também verificado se estivesse a ocorre uma contagem do tempo, se esta chegou ou ultrapassou uma hora, sendo a contagem parada se for esse o caso.

Estruturas de Dados

- GPSCoord (fornece as coordenadas GPS):
  - Latitude - indica a latitude de um ponto.
  - Longitude - indica a longitude de um ponto.

- Graph:
  - VertexSet - Set de vértices do grafo.
  - Gv - *Graphviewer* que vai auxiliar a representação gráfica do grafo.

- Info (contém informações acerca das arestas):
  - Name - Designa o nome.
  - ID - Representa o ID.
  - busStation/trainStation - Indicam se a aresta em questão é uma linha ferroviária ou uma via para autocarros.

- Edgetype (enumera os tipos de arestas)
  - Undirected - Especifica que as arestas podem ser percorridas em 2 sentidos.
  - Directed - Enuncia que as arestas são percorridas apenas num sentido.
- Vertex
  - Coord – Contém as coordenadas GPS do ponto.
  - Adj – Arestas adjacentes.
  - Path – Apontador para o próximo vértice no caminho desde a origem.
  - QueueIndex – Demonstra o índice do vértice na fila de prioridade.
  - Id – Revela o id do nó.
  - Processing – Indica que o vértice está a ser processado.
  - Counting – Indica que está a contar o tempo de viagem no algoritmo.
  - LimitReached – Revela se foi atingido o limite monetário imposto pelo utilizador.

- ConnectionToTrain – Indica que o vértice tem uma aresta adjacente que se trata de uma linha ferroviária.
- Time – Demonstra o tempo herdado.
- Cost – Indica preço a pagar durante a viagem até ao nó.
- Time_count – Relaciona-se com a contagem do tempo de viagem.

- Edge
  - Dest – Aponta o vértice de destino da aresta.
  - Weight – Indica o peso (comprimento) da aresta.
  - Time –Indica o tempo que se demora a percorrer a aresta.
  - Info – Contém informação adicional acerca da aresta.

Casos de Utilização

Com a nossa aplicação, é então possível realizar as seguintes ações:

- Seleção do melhor trajeto tendo em conta a distância.
- Escolha do melhor itinerário em termos de tempo.
- Decisão do melhor caminho considerando o custo da viagem, preferência por certos meios de transporte e distância total percorrida a pé.
- Análise de dados referentes à rede de transportes contidos em ficheiros.
- Visualização gráfica da rede de transportes e do melhor percurso com a ajuda do *GraphViewer.*

Dificuldades encontradas

Ao longo da realização deste projeto, deparámo-nos com certas dificuldades que retardaram o seu término, apesar de todas elas terem sido superadas com sucesso.

Aprofundando um pouco, os nossos maiores entraves residiram principalmente na extração e utilização de dados que obtivemos através do *Parser*.

Consideramos que o tempo gasto na aprendizagem da utilização do *Parser* e do *Graphviewer* foi demasiado para o valor que ambos possuem no trabalho.

Esforço dedicado por cada elemento

Interface: A interface foi construída por Eduardo Silva, visto que já tinha alguma experiência em criar este tipo de comunicação entre utilizador e programa através da consola.

Algoritmos: Todos os elementos do grupo tiveram um papel ativo nesta fase, sendo que numa primeira etapa foi Carolina Azevedo que organizou e elaborou uma estrutura básica para adaptarmos os algoritmos já conhecidos ao nosso projeto, sendo esta implementação feita de seguida maioritariamente por Eduardo Silva, além da correção de erros que surgiam ao longo do programa, ficando Tomás Novo responsável por testar e analisar os resultados obtidos e por melhorar estes mesmos.

Relatório: O relatório foi elaborado por Tomás Novo e Eduardo Silva

Conclusão

Com este projeto foi possível aumentar os nossos conhecimentos sobre a criação, funcionamento e manipulação dos grafos, bem como entender o funcionamento dos algoritmos de cálculo do caminho mais curto.

A qualidade da nossa programação foi também reforçada na medida em que criámos uma aplicação com uma interface bastante simples e pragmática, na qual o utilizador pode escolher o caminho mais curto em termos de tempo ou distância, podendo também impor um custo máximo da viagem.

Concluímos assim que este trabalho foi benéfico para todos os elementos e ambicionamos melhorar para a 2ª parte deste projeto.