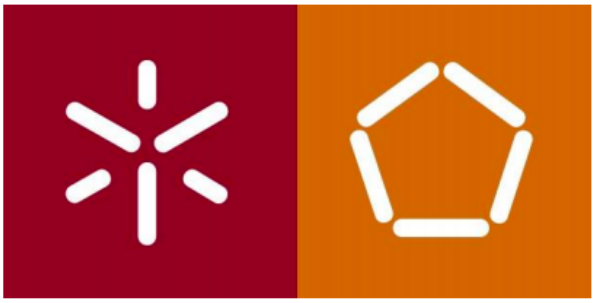

Universidade do Minho

Mestrado Integrado em Engenharia Informática

15/9/2020

Sistemas de Representação de Conhecimento e Raciocínio

Época Especial

António Gonçalves A85516

Resumo

Neste relatório irei descrever a resolução do trabalho individual pedido para a época especial, onde se deve ler e trabalhar informação relativa a estações de um metropolitano. Para isto, irei explicar as estratégias que utilizei, quer na leitura e escrita da informação, quer no seu tratamento.

Contents

1 Introdução

Para este trabalho foi-nos proposto que realizasse-mos uma base de conhecimento relativa a várias estações de um metropolitano.

Estas informações seriam lidas de um ficheiro Excel, tendo de ser trabalhadas para terem a formatação correta, para que possam ser inseridas na nossa base de conhecimento.

Uma vez inseridas, o utilizador poderá realizar várias *queries* que lhe permitem observar a informação presente, bem como realizar cálculos, escolher caminhos, entre outras. De seguida irei explicar como resolvi as várias fases necessárias à realização deste trabalho.

2 Preliminares

Tal como já foi referido anteriormente, a realização deste trabalho pode ser dividida em 2 partes principais: a leitura e inserção dos dados nas base de conhecimento, sendo nesta parte tambem criadas as ligações entre as várias estações, e o tratamento dos dados, ou seja, a resolução das *queries* pedidas, bem como qualquer outra funcionalidade que seja útil desenvolver.

Todo o trabalho foi realizado tendo em conta que não seria possível alterar a informação da base de conhecimento, isto é, caso houvesse alguma alteração a ser feita, esta seria realizada no próprio ficheiro Excel. Também não faz sentido representar conhecimento Imperfeito, uma vez que se trata de informação sobre paragens metropolitanas, sendo praticamente impossível ou mesmo não prático uma paragem estar a ser utilizada para a realização de viagens sem se saber alguma informação sobre a mesma.

3 Realização do Trabalho

De seguida irei explicar a minha realização do trabalho, dividindo a mesma em 2 partes: A leitura do ficheiro Excel e respetiva escrita dos predicados necessário, e o tratamento da base de conhecimento, bem como a resposta às várias alíneas que foram colocadas.

3.1 Leitura do Excel e Predicados

Para a leitura dos dados utilizei a linguagem de programação *Java*, bem como a *Apache* POI. Assim pude percorrer a folha inteira, linha a linha, e retirar a informação referente a cada campo dos predicados presente nas células.

Assim, presentes inicialmente no ficheiro, cada paragem têm informação referente ao seu ID, o GIS_ID, as várias linhas a que a paragem pertence e as coordenadas X e Y onde a paragem se encontra. Para além disto, adicionei mais duas váriaveis binárias, que representam a presença de Bilheteira e de Bar na paragem. Estas informações foram adicionadas para permitir a realização das questões 2 e 3, como irei explicar mais à frente.

Assim, ficamos com:

paragem: *#Id, GID_ID, [Linhas], X, Y, Bar, Bilheteira* $\rightarrow \{V, F\}$

Ao longo desta leitura, as informações vão sendo guardadas em vários *Maps*.

1. ***paragens***:guarda todas as paragens que pertencem a cada linha. Está informação serve mais para facilitar a leitura e verificar as soluções que mais tarde vão ser obtidas;

2. ***linhas***:guarda a relação entre o identificador da cor da linha e a *String* que a representa. Mais uma vez, esta informação não será relevante, mas ajuda a comprovar que a informação foi bem lida do *Excel*;

3. ***coords***:guarda as coordenadas lidas para cada paragem. Estes valores serão depois utilizados para calcular as distancias de cada aresta.

Finalmente, usei uma *List* de *Arestas*, para guardar o início, o fim e a distância que cada aresta representa.

Assim, para além do predicado *Paragem*, criei também:

linha/2: $\#IdLinha, IdsParagens \rightarrow \{V, F\}$;

aresta/3: $\#IdOrigem, IdDestino, Distncia \rightarrow \{V, F\}$;

linhas/2: $Cor, \#IdCor \rightarrow \{V, F\}$

A minha estratégia para a criação das arestas foi a seguinte: todas as paragens estarão ligadas à paragem anterior e à seguinte dentro da mesma linha, sendo isto diferente para a primeira e última, como é óbvio. Assim, paragens que estejam presentes em mais do que uma linha irão fazer a ligação entre as mesmas. Por exemplo:

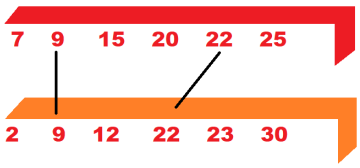

Figure 1: Exemplo de duas Linhas inter-ligadas

Neste caso, a paragem 9 teria arestas para 7, 15, 2 e 12, tendo também as arestas que representam o sentido contrário. Assim, será possível, por exemplo, realizar uma viagem de 7 para 12, usando o caminho 7->9->12.

O calculo da distância na criação das arestas foi baseado num método retirado de um *site*, sendo este referenciado no fim do relatório.

Tendo assim toda a informação devidamente guardada em memória, resta somente escreve-la no ficheiro *infopross.pl*, sendo este consultado para adicionar a informação necessária ao funcionamento do trabalho.

```
paragem( 1, 57,[0, 1], -77.0219166779745, 38.8936730087034,0.0,1.0).
paragem( 2, 58,[0], -77.0175074963564, 38.8764696487714,0.0,1.0).
paragem( 3, 59,[2, 1], -77.0440445887926, 38.8534241622265,0.0,1.0).
```

Figure 2: Exemplo de paragens no ficheiro *infopross.pl*

```
linha(5,[7, 9, 10, 13, 26, 28, 29, 40, 41, 42, 43, 44, 45, 46, 56, 59, 67, 71, 72, 73, 74, 75, 76, 77, 78, 86, 87]).
linha(1,[1, 3, 4, 5, 10, 11, 12, 18, 19, 20, 21, 30, 36, 39, 46, 47, 48, 60, 61, 64, 66, 88, 90]).
linha(0,[1, 2, 10, 11, 12, 14, 15, 16, 17, 22, 30, 36, 39, 46, 47, 48, 62, 64, 65, 66, 88]).
linha(3,[6, 8, 9, 23, 24, 25, 27, 37, 38, 49, 50, 51, 52, 53, 54, 55, 57, 63, 68, 69, 70, 79, 80, 81, 82, 83, 84, 85, 88, 89, 91]).
linha(2,[3, 4, 5, 6, 8, 9, 18, 19, 23, 24, 25, 37, 38, 49, 50, 55, 57, 58, 60, 61, 63, 79, 85, 88, 89, 90, 91]).
linha(4,[6, 8, 9, 23, 24, 25, 31, 32, 33, 34, 35, 37, 38, 49, 50, 55, 57, 63, 79, 80, 81, 82, 83, 84, 85, 88, 89, 91]).
```

Figure 3: Linhas escritas no ficheiro *infopross.pl*

```
aresta(29, 28, 0.009129963414110072).
aresta(29, 40, 0.3584524251569299).
aresta(40, 29, 0.3584524251569299).
```

Figure 4: Exemplo de arestas no ficheiro *infopross.pl*

3.2 Parte em *Prolog*

Após consultar o ficheiro *infopross.pl*, temos agora acesso a toda a informação que estava contida no *excel*, o que nos permite realizar as questões.

3.2.1 Questão 1

Calcular um trajeto entre duas estações;

Resolvi esta alínea recorrendo a dois algoritmos diferentes: um que percorre os ramos dos grafo em profundidade (*Depth-First*) e outro em largura (*Breadth-First*). Estes dois algoritmos são utilizados várias vezes ao longo do trabalho.

Resumidamente, o algoritmo *Depth-First*, irá primeiro, como o nome indica, procurar nos nodos mais fundos do grafo, indo depois procurar nos restantes. Este algoritmo é bastante rápido caso o caminho a encontrar seja sem qualquer mudança de Linha, caso haja, fica bastante lente, chegando até a não conseguir encontrar caminho num tempo aceitável. O algoritmo *Breadth-First* tem em conta os nos seguintes àqueles por onde está a passar ou onde já passou. Assim, este consegue tempos melhores em caminhos menos simples, ou seja, onde existe necessidade de troca de Linhas, mas um pouco mais lento caso contrário, uma vez que tem de observar na mesma as restantes arestas que podem ser Candidatos para o caminho.

3.2.2 Questão 2

Selecionar apenas estações com uma determinada característica, para um determinado percurso;

Como já referi anteriormente, para realizar esta alínea e a seguinte, adicionei às informações das Paragens no *Excel* duas variáveis binárias que representam a presença de Bar e de Bilheteira na paragem.

Assim, a única diferença entre esta e a alínea anterior é que antes de decidir se vai continuar para a próxima aresta, deve verificar se a paragem em questão tem a característica pedida.

Para destinguir as caracteristicas pedidas usei um valor de 1 a 3, sendo que o 1 representa paragens com Bilheteira, 2 paragens com Bar e o 3 paragens com Bilheteira e Bar.

3.2.3 Questão 3

Excluir uma ou mais características de estações para o percurso;

Esta alínea é em tudo igual à anterior, mas em vez de verificar se a próxima paragem possui a característica para puder prosseguir, faz o contrário. Caso as variáveis tenham o valor pretendido igual a zero, estas poderão fazer parte do caminho.

Uma vez que estes valores foram inseridos manualmente, não existem caminhos muito grandes para testar as duas ultimas alíneas, contudo, chega existir um caminho em que uma paragem não satisfaça as condições para o programa detetar que não é possível. Por exemplo, se quisermos ir da estação 11 para a 13, passando somente em paragens sem bar, caso exista o caminho 11->12->13 e a paragem 12 tenha bar, esta caminho não será aceite.

3.2.4 Questão 5

Escolher o menor percurso (usando o critério do menor número de estações intermédias);

Esta alínea é respondida imediatamente pelo algoritmo *Breadth-First*, uma vez que o caminho que este obtem é o caminho com o menor número de arestas percorrido.

3.2.5 Questão 6

Escolher o percurso mais rápido (usando o critério da distância);

Para esta alínea utilizei o algoritmo *A** que utilizamos nas aulas. Este algoritmo é bastante parecido com o *Breadth-First*, contudo, aquando da escolha do próximo nodo, é feita uma estimativa da distancia do candidato ao destino. O candidato que tiver esta estimativa somada com a distancia da aresta do nodo atual a si menor, será o escolhido para continuar o algoritmo.

Para a estimativa decidir utilizar a mesma função utilizada para calcular as distâncias entre as paragens na primeira parte do trabalho, sendo o resultado multiplicado por 2, uma vez que é necessário assegurar que a distância estimada é maior do que a verdadeira.

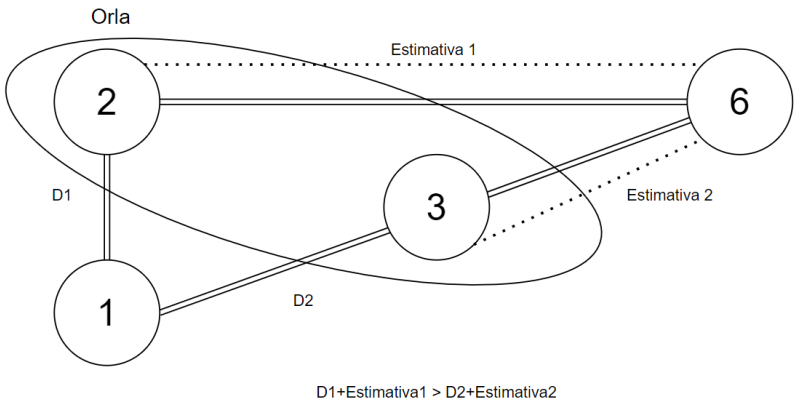

Figure 5: Escolha da paragem a seguir, neste caso, seria a 3.

3.2.6 Questão 7

Escolher o percurso que passe apenas por uma determinada linha;

Para esta alínea a ideia é exatamente a mesma da primeira, contudo, temos de confirmar que a paragem que vamos adicionar ao caminho pertence à linha que pedimos. Contudo, a minha implementação do algoritmo *Breadth-First* deste exercício não consegue parar caso o destino seja de outra linha, sendo este problema algo que não consegui resolver.

3.2.7 Questão 8

Escolher uma ou mais linhas por onde o percurso deverá passar, dadas duas estações;

Mais uma vez, a ideia desta alínea é igual à da primeira, sendo necessário, após obter um caminho, verificar se todas as Linhas pedidas estão presentes nas paragens do caminho. Caso não isto não aconteça, deverá ser encontrado outro caminho.

3.2.8 Exercício adicional

A primeira vez que li o exercício 4, interpretei-o de uma forma diferente, e uma vez que resolvi outra problema que não era pedido, decidi deixar aqui uma explicação.

Determinar as N paragens com mais saídas de um percurso

Para realizar este exercício, é calculado o caminho como seria feito na primeira alínea, sendo depois contada as saídas de cada paragem. Com esse valor é criado uma lista de pares, em que cada entrada é o número de saídas e o Id da paragem. De seguida ordena-se esta lista, e por fim, retiram-se o número de elementos pedidos.

4 Resultados

Ao longo do relatório já fui falando de qual algoritmo seria melhor para que tipo de exercício, mas nesta secção irei abordar isto mais aprofundadamente.

O algoritmo *Depth-First* será o mais rápido e mais apropriado para exercícios onde se quer um caminho que não mude várias vezes de linha, ou que, se mudar, seja poucas vezes. Alíneas como a 1 (caso seja um caminho simples) e a 7 são aquelas que são mais rápidas do que qualquer outro algoritmo.

Contudo, o algoritmo *Breadth-First* consegue melhores resultados na maioria das restantes alíneas, uma vez que este consegue chegar ao destino percorrendo menos nodos.

Mesmo assim, nenhum dos dois algoritmos anteriores consegue escolher um caminho com base nas distâncias percorridas. Para isso temos o algoritmo A*. Em adição ao *Breath-First*, este utiliza uma estimativa para escolher o próximo nodo que, muito provavelmente, esteja mais próximo do destino. Contudo, uma vez que a estimativa que eu utilizo para o algoritmo é a distância em linha reta multiplicada por 2, pode existir algum caso em que isto obrigue o algoritmo a não fazer a escolha certa.

Assim, servindo de resumo e de comparação final, tendo:

b: Factor máximo de ramificação de um nodo, neste caso será 6(linhas diferentes). Contudo este valor nunca é atingido;

d: Profundidade da melhor solução, ou seja, o número mínimo de nodos percorridos para uma solução;

m: Profundidade máxima do espaço de estados;
Temos que:

Algoritmo	**Completo**	**Complexidade no Tempo**	**Complexidade no espaço**
Depth-First	Não	O(b^d)	O(b^d)
Breadth-First	Sim	O(b^m)	O(b*m)
*A**	Sim	O(b^d)	O(b^d)

5 Conclusão

Em geral, penso ter atingido o objetivo do trabalho, apesar de não ter concluído a 100% aquilo que me tinha sido pedido.

Consegui alterar os algoritmos dados nas aulas para responder aos problemas propostos e penso que a forma de gerar as arestas que criei é bastante semelhante à realidade.

Para além de realizar a questão 4, que não consegui resolver, poderia ainda melhorar o algoritmo *Breadth-First* da questão 7 que não termina automaticamente quando o final não pertence à linha pretendida. Poderia também otimizar um pouco mais o código, uma vez que existe alguma repetição de código desnecessário, por exemplo, nas alíneas 2 e 3.

6 Referências

1. https://www.movable-type.co.uk/scripts/latlong.html
2. Carlsson, Mats, et al., “SICStus Prolog User’s Manual - Release 4.5.1", Abril, 2019
3. https://www.swi-prolog.org/