Universidade do Minho

Escola de Engenharia - Departamento de Informática

Relatório do Trabalho Prático 2

Mestrado Integrado em Engenharia Informática

Sistemas de Representação de Conhecimento e Raciocínio

A84610 – Gonçalo Almeida

Braga, Junho 2020

Introdução

Este trabalho prático tem como objetivo aprofundar o conhecimento da unidade curricular de Sistemas de Representação de Conhecimento e Raciocínio, através do desenvolvimento de métodos de resolução de problemas e de algoritmos de pesquisa.

Este trabalho serve-se da linguagem de programação em lógica PROLOG para representar o conhecimento e raciocínio relativo ao sistema de transportes do concelho de Oeiras.

Representação dos dados

De modo a formatar os dados fornecidos em Excel para que sejam válidos em PROLOG foi necessário desenvolver um parser, este sendo desenvolvido em Java devido à simplicidade da biblioteca jxl existente. Para cada sheet (representante da carreira) foram lidas as paragens e criadas as adjacências entre estas (duas paragens seguidas representam uma adjacência).
A representação das paragens contém 10 componentes, a gid, a latitude e longitude, o estado, o tipo de abrigo, a existência de publicidade, a operadora, o código da rua, o nome da rua e a freguesia.
Quanto às adjacências, estas contêm 4 componentes, o gid da paragem origem, o gid da paragem destino, a carreira e a distância euclidiana entre estas.
Foi também desenvolvida a representação das carreiras de cada paragem de modo a facilitar o desenvolvimento dos problemas propostos.

Durante o parse dos dados fornecidos ocorreram algumas incongruências que foram resolvidas da seguinte forma:

- Latitude ou longitude não existente: foi atribuída a soma destes parâmentos na paragem adjacente anterior com 100;

- Parâmetros textuais não existentes: foi atribuído o valor de 'desconhecido'.

```
% Extensao do predicado paragem: Gid, Lat, Lon, Est, TAb, Pub, Ope, CRua, NRua, Fre -> {V,F}

paragem(5, -106997.31, -95311.49, 'Bom', 'Sem Abrigo', 'No', 'Carris', 103, 'Rua Damiao de Gois', 'Alges, Linda-a-Velha e Cruz Quebrada-Dafundo').
paragem(6, -106992.24, -95299.38, 'Bom', 'Sem Abrigo', 'No', 'Vimeca', 103, 'Rua Damiao de Gois', 'Alges, Linda-a-Velha e Cruz Quebrada-Dafundo').
paragem(8, -106980.35, -95289.3, 'Bom', 'Sem Abrigo', 'No', 'Carris', 103, 'Rua Damiao de Gois', 'Alges, Linda-a-Velha e Cruz Quebrada-Dafundo').
paragem(9, -107003.0, -95216.21, 'Bom', 'Fechado dos Lados', 'Yes', 'Vimeca', 103, 'Rua Damiao de Gois', 'Alges, Linda-a-Velha e Cruz Quebrada-Dafundo').
paragem(10, -107129.12, -102327.55, 'Bom', 'Fechado dos Lados', 'Yes', 'LT', 545, 'Praca Comandante Henrique Moreira Rato', 'Oeiras e Sao Juliao da Barra, Paco de Arcos e Caxias').
```

Figura 1 - Representação de paragens

```
% Extensao do predicado adjacencia: GidOrigem, GidDestino, Carreira, Distancia -> {V,F}

adjacente(183, 791, 1, 87.63541293336934).
adjacente(791, 595, 1, 698.6929317661744).
adjacente(595, 182, 1, 421.9863463431075).
adjacente(182, 499, 1, 2003.3340491291042).
adjacente(499, 593, 1, 245.01549440800457).
```

Figura 2 - Representação de adjacências

```
% Extensao do predicado adjacencia: Gid, Carreiras -> {V,F}

carreiras(5, [776]).
carreiras(6, [6]).
carreiras(8, [776]).
carreiras(9, [6]).
carreiras(10, [112, 106, 122]).
carreiras(11, [201, 748, 751]).
```

Figura 3 - Representação das carreiras de cada paragem

Resolução dos problemas propostos

Para este trabalho prático foram propostos os seguintes problemas:

- Calcular um trajeto entre dois pontos;
- Selecionar apenas algumas das operadoras de transporte para um determinado percurso;
- Excluir um ou mais operadores de transporte para o percurso;
- Identificar quais as paragens com o maior número de carreiras num determinado percurso.
- Escolher o menor percurso (usando critério menor número de paragens);
- Escolher o percurso mais rápido (usando critério da distância);
- Escolher o percurso que passe apenas por abrigos com publicidade;
- Escolher o percurso que passe apenas por paragens abrigadas;
- Escolher um ou mais pontos intermédios por onde o percurso deverá passar.

Conforme estes problemas sugeridos, foram desenvolvidos os seguintes predicados:

```
trajeto(Origem, Destino, Caminho) :- profundidade(Origem, Destino, [Origem], Caminho),
                                     write('Trajeto: '),
                                     escrever(Caminho).

profundidade(Destino, Destino, Visitados, Caminho) :- inverso(Visitados, Caminho).

profundidade(Origem, Destino, Visitados, Caminho) :- adjacente_h(Origem, Prox),
                                                     \+ member(Prox, Visitados),
                                                     profundidade(Prox, Destino, [Prox|Visitados], Caminho).

adjacente_h(Paragem, Prox) :- adjacente(Paragem, Prox, _, _).
```

Figura 4 - Calcular um trajeto entre dois pontos

```
com_operadoras(Origem, Destino, Operadoras, Caminho) :- paragem(Destino, _, _, _, _, _, Ope, _, _, _),
                                                        pertence(Ope, Operadoras),
                                                        profundidade2(Origem, Destino, Operadoras, [Origem], Caminho),
                                                        write('Trajeto: '),
                                                        escrever(Caminho).

profundidade2(Destino, Destino, Operadoras, Visitados, Caminho) :- inverso(Visitados, Caminho).

profundidade2(Origem, Destino, Operadoras, Visitados, Caminho) :- adjacente_h2(Origem, Prox, Operadoras),
                                                                  \+ member(Prox, Visitados),
                                                                  profundidade2(Prox, Destino, Operadoras, [Prox|Visitados], Caminho).

adjacente_h2(Paragem, Prox, Operadoras) :- paragem(Paragem, _, _, _, _, _, Ope, _, _, _),
                                           pertence(Ope, Operadoras),
                                           adjacente(Paragem, Prox, _, _).
```

Figura 5 - Selecionar apenas algumas das operadoras de transporte para um determinado percurso

```
sem_operadoras(Origem, Destino, Operadoras, Caminho) :- paragem(Destino, _, _, _, _, _, Ope, _, _, _),
                                                        nao(pertence(Ope, Operadoras)),
                                                        profundidade3(Origem, Destino, Operadoras, [Origem], Caminho),
                                                        write('Trajeto: '),
                                                        escrever(Caminho).

profundidade3(Destino, Destino, Operadoras, Visitados, Caminho) :- inverso(Visitados, Caminho).

profundidade3(Origem, Destino, Operadoras, Visitados, Caminho) :- adjacente_h3(Origem, Prox, Operadoras),
                                                                  \+ member(Prox, Visitados),
                                                                  profundidade3(Prox, Destino, Operadoras, [Prox|Visitados], Caminho).

adjacente_h3(Paragem, Prox, Operadoras) :- paragem(Paragem, _, _, _, _, _, Ope, _, _, _),
                                           nao(pertence(Ope, Operadoras)),
                                           adjacente(Paragem, Prox, _, _).
```

Figura 6 - Excluir um ou mais operadores de transporte do percurso

```
mais_carreiras(Origem, Destino, Caminho) :- mais_carreiras(Origem, Destino, Caminho, Origem, 0).

mais_carreiras(Origem, Destino, Caminho, Gid, N) :- profundidade4(Origem, Destino, [Origem], Caminho, Gid, N),
                                                    write('Trajeto: '),
                                                    escrever(Caminho).

profundidade4(Destino, Destino, Visitados, Caminho, Gid, N) :- inverso(Visitados, Caminho),
                                                               write('Paragem: '),
                                                               write(Gid),
                                                               write(' -> '),
                                                               write(N),
                                                               write(' carreiras\n').

profundidade4(Origem, Destino, Visitados, Caminho, Gid, N) :- adjacente_h(Origem, Prox),
                                                              \+ member(Prox, Visitados),
                                                              carreiras(Origem, Lista),
                                                              length(Lista, N1),
                                                              N1 > N,
                                                              profundidade4(Prox, Destino, [Prox|Visitados], Caminho, Origem, N1).

profundidade4(Origem, Destino, Visitados, Caminho, Gid, N) :- adjacente_h(Origem, Prox),
                                                              \+ member(Prox, Visitados),
                                                              carreiras(Origem,Lista),
                                                              length(Lista,N1),
                                                              N1 =< N,
                                                              profundidade4(Prox, Destino, [Prox|Visitados], Caminho, Gid, N).
```

Figura 7 - Identificar quais as paragens com o maior número de carreiras num determinado percurso

```
menos_paragens(Origem, Destino, R) :- findall( (Caminho,N),
                                               ( profundidade(Origem, Destino, [Origem], Caminho),
                                                 length(Caminho, N) ),
                                               S ),
                                      menor(S, R),
                                      write('Trajeto: '),
                                      escrever(R).
```

Figura 8 - Escolher o menor percurso usando critério menor número de paragens

```
menor_distancia(Origem, Destino, Caminho/Distancia) :- estima(Origem, Destino, Estima),
                                                       aestrela([[Origem]/0/Estima], InvCaminho/Distancia/_, Destino),
                                                       inverso(InvCaminho, Caminho),
                                                       write('Trajeto: '),
                                                       escrever(Caminho),
                                                       write('\n'),
                                                       write('Distancia: '),
                                                       write(Distancia).

aestrela(Caminhos, Caminho, Destino) :- obtem_melhor(Caminhos, Caminho),
                                        Caminho = [Destino|_]/_/_.

aestrela(Caminhos, SolucaoCaminho, Destino) :- obtem_melhor(Caminhos, MelhorCaminho),
                                               seleciona(MelhorCaminho, Caminhos, OutrosCaminhos),
                                               expande_aestrela(MelhorCaminho, ExpCaminhos, Destino),
                                               append(OutrosCaminhos, ExpCaminhos, NovoCaminhos),
                                               aestrela(NovoCaminhos, SolucaoCaminho, Destino).

obtem_melhor([Caminho], Caminho) :- !.

obtem_melhor([Caminho1/Custo1/Est1,_/Custo2/Est2|Caminhos], MelhorCaminho) :- Custo1 + Est1 =< Custo2 + Est2, !,
                                                                              obtem_melhor([Caminho1/Custo1/Est1|Caminhos], MelhorCaminho).

obtem_melhor([_|Caminhos], MelhorCaminho) :- obtem_melhor(Caminhos, MelhorCaminho).

expande_aestrela(Caminho, ExpCaminhos, Destino) :- findall(NovoCaminho, adj(Caminho, NovoCaminho, Destino), ExpCaminhos).

adj([Paragem|Caminho]/Custo/_, [Prox,Paragem|Caminho]/NovoCusto/Est, Destino) :- adjacente(Paragem, Prox, _, PassoCusto),
                                                                                 \+ member(Prox, Caminho),
                                                                                 NovoCusto is Custo + PassoCusto,
                                                                                 estima(Prox, Destino, Est).
```

Figura 9 - Escolher o percurso mais rápido usando critério da distância

```
com_publicidade(Origem, Destino, Caminho) :- paragem(Destino, _, _, _, _, Pub, _, _, _, _),
                                             pertence(Pub, ['Yes']),
                                             profundidade5(Origem, Destino, [Origem], Caminho),
                                             write('Trajeto: '),
                                             escrever(Caminho).

profundidade5(Destino, Destino, Visitados, Caminho) :- inverso(Visitados, Caminho).

profundidade5(Origem, Destino, Visitados, Caminho) :- adjacente_h5(Origem, Prox),
                                                      \+ member(Prox, Visitados),
                                                      profundidade5(Prox, Destino, [Prox|Visitados], Caminho).

adjacente_h5(Paragem, Prox) :- paragem(Paragem, _, _, _, _, Pub, _, _, _, _),
                               pertence(Pub, ['Yes']),
                               adjacente(Paragem, Prox, _, _).
```

Figura 10 -Escolher o percurso que passe apenas por abrigos com publicidade

```
com_abrigo(Origem, Destino, Caminho) :- paragem(Destino, _, _, _, TAb, _, _, _, _, _),
                                        nao(pertence(TAb, ['Sem Abrigo'])),
                                        profundidade6(Origem, Destino, [Origem], Caminho),
                                        write('Trajeto: '),
                                        escrever(Caminho).

profundidade6(Destino, Destino, Visitados, Caminho) :- inverso(Visitados, Caminho).

profundidade6(Origem, Destino, Visitados, Caminho) :- adjacente_h6(Origem, Prox),
                                                      \+ member(Prox, Visitados),
                                                      profundidade6(Prox, Destino, [Prox|Visitados], Caminho).

adjacente_h6(Paragem, Prox) :- paragem(Paragem, _, _, _, TAb, _, _, _, _, _),
                               nao(pertence(TAb, ['Sem Abrigo'])),
                               adjacente(Paragem, Prox, _, _).
```

Figura 11 - Escolher o percurso que passe apenas por paragens abrigadas

```
pontos_intermedios(Origem, Destino, Pontos, Caminho) :- pertence(Origem, Pontos),
                                                        remove(Origem, Pontos, L),
                                                        profundidade7(Origem, Destino, L, [Origem], Caminho),
                                                        write('Trajeto: '),
                                                        escrever(Caminho).

pontos_intermedios(Origem, Destino, Pontos, Caminho) :- nao(pertence(Origem, Pontos)),
                                                        profundidade7(Origem, Destino, Pontos, [Origem], Caminho),
                                                        write('Trajeto: '),
                                                        escrever(Caminho).

profundidade7(Destino, Destino, [], Visitados, Caminho) :- inverso(Visitados, Caminho).

profundidade7(Origem, Destino, Pontos, Visitados, Caminho) :- adjacente_h(Origem, Prox),
                                                              \+ member(Prox, Visitados),
                                                              pertence(Prox, Pontos),
                                                              remove(Prox, Pontos, L),
                                                              profundidade7(Prox, Destino, L, [Prox|Visitados], Caminho).

profundidade7(Origem, Destino, Pontos, Visitados, Caminho) :- adjacente_h(Origem, Prox),
                                                              \+ member(Prox, Visitados),
                                                              nao(pertence(Prox, Pontos)),
                                                              profundidade7(Prox, Destino, Pontos, [Prox|Visitados], Caminho).
```

Figura 12 - Escolher um ou mais pontos intermédios por onde o percurso deverá passar

No desenvolvimento dos predicados acima apresentados, foi necessário desenvolver os seguintes predicados auxiliares:

```
nao( Questao ) :- Questao, !, fail.
nao( Questao ).
```

Figura 13 - Extensão do meta-predicado não

```
inverso(Xs, Ys) :- inverso(Xs, [], Ys).

inverso([], Xs, Xs).
inverso([X|Xs],Ys, Zs) :- inverso(Xs, [X|Ys], Zs).
```

Figura 14 - Inverter uma lista

```
seleciona(E, [E|Xs], Xs).
seleciona(E, [X|Xs], [X|Ys]) :- seleciona(E, Xs, Ys).
```

Figura 15 - Extensão do predicado seleciona

```
estima(Paragem, Destino, Estima) :- paragem(Paragem, Lat1, Lon1, _, _, _, _, _, _, _),
                                    paragem(Destino, Lat2, Lon2, _, _, _, _, _, _, _),
                                    X is Lat2 - Lat1,
                                    Y is Lon2 - Lon1,
                                    X2 is exp(X,2),
                                    Y2 is exp(Y,2),
                                    Z is X2 + Y2,
                                    Estima is sqrt(Z).
```

Figura 16 - Calcular a distância entre duas paragens

```
escrever([]).
escrever([X]) :- write(X).
escrever([X|L]) :- write(X), write(','), escrever(L).
```

Figura 17 - Escrever uma lista

```
pertence( X,[X|_] ).
pertence( X,[Y|L] ) :- X \= Y, pertence( X,L ).
```

Figura 18 - Verificar se um elemento pertence a uma lista

```
remove(X, [], []).
remove(X, [X|T], T).
remove(X, [H|T], [H|L]) :- remove(X, T, L).
```

Figura 19 - Remover um elemento de uma lista

```
menor(L, R) :- menor(L, [], 0, R).
menor([], Caminho, Tamanho, Caminho).
menor([(C,N)|T], [], 0, R) :- menor(T, C, N, R).
menor([(C,N)|T], Caminho, Tamanho, R) :- N < Tamanho, menor(T, C, N, R).
menor([(C,N)|T], Caminho, Tamanho, R) :- N >= Tamanho, menor(T, Caminho, Tamanho, R).
```

Figura 20 - Determinar o caminho com menos paragens dada uma lista de pares (caminho, tamanho)

```
help(R) :- write('\n'),
           write('Trajeto entre dois pontos: trajeto(Origem, Destino, R)\n\n'),
           write('Trajeto que passe apenas por paragens com determinadas operadoras: com_operadoras(Origem, Destino, Operadoras, R)\n\n'),
           write('Trajeto que passe apenas por paragens sem determinadas operadoras: sem_operadoras(Origem, Destino, Operadoras, R)\n\n'),
           write('Trajeto identificando a paragem com o maior numero de carreiras: mais_carreiras(Origem, Destino, R)\n\n'),
           write('Trajeto com menos paragens: menos_paragens(Origem, Destino, R)\n\n'),
           write('Trajeto com menor distancia: menor_distancia(Origem, Destino, R)\n\n'),
           write('Trajeto que passe apenas por paragens com publicidade: com_publicidade(Origem, Destino, R)\n\n'),
           write('Trajeto que passe apenas por paragens abrigadas: com_abrigo(Origem, Destino, R)\n\n'),
           write('Trajeto que passe por determinados pontos intermedios: pontos_intermedios(Origem, Destino, Pontos, R)\n\n').
```

Figura 21 - Mostrar como executar os predicados

Comparação de algoritmos

Neste trabalho prático foram usados os algoritmos de pesquisa em profundidade primeiro multi-estados para pesquisa não informada e o algoritmo A* para pesquisa informada.

Pesquisa em profundidade primeiro

- A estratégia é expandir sempre um dos nós mais profundos da árvore;
- Necessita de pouca memória, o que é bom para problemas com muitas soluções;

Pesquisa A*

- Evita expandir caminhos que são caros;
- Combina a pesquisa gulosa com a uniforme, minimizando a soma do caminho já efetuado com o mínimo previsto que falta até a solução.

Usa a função

f(n) = g(n) + h(n)

em que

g(n) é o custo total, até ao momento, para chegar ao estado n (custo do percurso);

h(n) é o custo estimado para chegar ao objetivo;

f(n) é o custo estimado da solução mais barata que passa pelo nó n.

A seguinte tabela apresenta informações relativamente à complexidade e eficiência das estratégias utilizadas.

	Completo	Tempo (Complexidade)	Espaço (Complexidade)
Primeiro em Profundidade (DFS)	Não	O (b^m)	O (bm)
A*	Sim	Número de nodos com g(n) =< C*	Número de nodos com g(n) =< C*

Figura 22 - Estratégias de pesquisa

Conclusões

O desenvolvimento deste trabalho ajudou a aprofundar o conhecimento sobre a linguagem de programação PROLOG e a sua utilização na construção de sistemas com recurso a algoritmos de pesquisa.

O desenvolvimento deste trabalho utilizando o algoritmo de pesquisa primeiro em largura para pesquisa não informada aumentaria a eficiência devido à possível ocorrência de loops neste mundo de sistemas de transportes.

Contudo, considero que a grande parte dos objetivos foram alcançados de acordo com as especificações do enunciado.