Sistemas de Representação de Conhecimento e Raciocínio
(3º ano MIEI)

Trabalho de Recurso

Relatório de Desenvolvimento

Pedro Gomes (a84220)

14 de Julho de 2020

Resumo

Este relatório inicia-se com uma breve contextualização, de seguida é feita uma descrição do problema e é apresentada a resolução com uma pequena explicação e também com alguns exemplos de testes realizados. Por fim é feita uma analise e as conclusões tiradas sobre o desenvolvimento deste trabalho pratico individual de recurso.

Conteúdo

Capítulo 1

Introdução

Este trabalho foi realizado no âmbito da UC de Sistemas de Representação de Conhecimento e Raciocínio, do 2º semestre do 3º ano do Mestrado Integrado Em Engenharia Informática.

O trabalho consiste em primeiro lugar em processar os dados fornecidos sobre as cidades, tanto na conversão de xlsx para CVS como na definição das ligações entre as cidades.De seguida são aplicados métodos de pesquisa sobre esses mesmos dados de modo a apresentar informação útil ao utilizador sobre as cidades. Os métodos de pesquisa usados são o método Depth-first, em profundidade, para a pesquisa não informada e para a pesquisa informada é usado o método A estrela.

Capítulo 2

Concepção/Desenho da Resolução

2.1 Descrição do problema

O problema consiste em primeiro lugar em tratar os dados fornecidos e em seguida conseguir fazer uma serie de queries relativamente a cidades e percursos possíveis entre as mesmas, por exemplo,identificar num determinado percurso qual a cidade com o maior número de ligações ou pesquisar um percurso possível entre duas cidades sem ligação directa.

2.2 Tratamento dos Dados

Em primeiro lugar foi feita a conversão dos dados de xlsx para csv com a ajuda de um script. Depois de termos os dados no formato pretendido foi desenvolvido um programa em Java que percorre o ficheiro cidades.csv e calcula a distancia entre as mesmas através da latitude e longitude que possuem. Se a distancia entre as cidades for menor a um valor Máximo, determinado por mim, é considerado que tem ligação directa uma com a outra. Este registo é posteriormente registado no ficheiro ligacoes.csv no formato id1,id2,distancia .De notar que por razoes de performance decidi por só um registo da ligação, exemplo se a cidade 1 e 5 estão ligadas só vai aparecer no ficheiro 1,5,d ou 5,1,d mas é assumido posteriormente que se existe uma ligação 1-5 também existe 5-1. Também é de notar que adicionei mas um parâmetro as cidades no ficheiro cidades.csv que é se tem património cultural ou não em que nesse campo só pode ter 'yes' ou 'no'.

Por fim para importar os dados do cidades.csv e do ligacoes.csv para o interpretador é usado o comando "import"após consultar o ficheiro trabalho_recurso.pl.

```
:- use_module(library(csv)).


import:-
    csv_read_file('cidades.csv', Data, [functor(cidade), skip_header(iD)]),
    csv_read_file('ligacoes.csv', DataL, [functor(ligacao), skip_header(iD)]),
    maplist(assert, DataL),
    maplist(assert, Data).
```

Figura 2.1: Import dos dois ficheiros de dados

2.3 Entidades do Sistema

Como anteriormente foi explicado existe 2 ficheiros de dados que são importados logo vai haver duas entidades no sistema sendo elas, cidade que vai conter a informação de cada cidade e também ligação que representa através dos id's das cidades quais é tem ligações directas umas com as outras alem de conter também a distancia caso haja ligação directa.

Visto isto as entidades tem o respectivo template que é apresentado de seguida:

- cidade(ID, NOME, LATITUDE, LONGITUDE, ADMIN, CAPITAL,PATRIMONIO)
- ligacao(ID1,ID2,DISTANCIA)

2.4 Predicados e Funcionalidades do Sistema

Nesta secção vou apresentar as funcionalidades implementadas assim como qualquer tipo de justificação caso seja apropriado.

2.4.1 Calcular um trajeto possível entre duas cidades

Pesquisa não informada - Depth First

Este algoritmo em profundidade consiste em ter uma função auxiliar recursiva que vai retornar um caminho valido que encontre, isto é, em que as cidades tem ligações com o nodo anterior e seguinte. Este algoritmo vai também servir de base para as diferentes funcionalidades do sistema que vão surgir posteriormente.

```
% Pesquisa Depth-First de um trajeto possível entre duas cidades
trajetoDp(C1, C2, Percurso/Distancia) :-
    trajetoDpRec([C1], C2, InvPercurso, 0, Distancia),
    inverso(InvPercurso, Percurso).

trajetoDpRec([H | T], H, [H | T], Distancia, Distancia).
trajetoDpRec([H | T], B, Percurso, CurrDistancia, Distancia) :-
    cidadesLigadasDist(H,C,D1),
    \+member(C, [H | T]),
    NewDistancia is CurrDistancia + D1,
    trajetoDpRec([C, H | T], B, Percurso, NewDistancia, Distancia).
```

Figura 2.2: Pesquisa Depth-First de um trajeto possível entre duas cidades

Pesquisa informada - A estrela

Este algoritmo vai devolver o melhor caminho possível entre as duas cidades. Este algoritmo usa o critério da distancia entre cidades e vai também ser a base para posteriores funcionalidades do sistema.

```
% Pesquisa A* do melhor trajeto entre duas cidades

trajeto_A(ID, Objetivo, Caminho/Comprimento):-
    distanciaEntreCidades(ID, Objetivo, Estima),
    aestrela([[ID]/0/Estima], InvCaminho/Comprimento/_, Objetivo),
    inverso(InvCaminho, Caminho).

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

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

expande_aestrela(Caminho, ExpCaminhos, Objetivo) :-
    findall(NovoCaminho, adjacente(Caminho,NovoCaminho, Objetivo), ExpCaminhos).

adjacente([Nodo|Caminho]/Custo/_, [ProxNodo,Nodo|Caminho]/NovoCusto/Est, Objetivo) :-
    cidadesLigadasDist(Nodo, ProxNodo, Resultado),
    \+member(ProxNodo, Caminho),
    NovoCusto is Custo + Resultado,
    distanciaEntreCidades(ProxNodo, Objetivo, Est).
```

Figura 2.3: Pesquisa A* do melhor trajecto entre duas cidades

2.4.2 Selecionar apenas cidades com certas responsabilidades administrativas ou património cultural para um determinado trajeto

Pesquisa não informada - Depth First

```
% Pesquisa em Profundidade restringindo o tipo de responsabilidades administrativas das cidade

trajetoDp_Resp(C1,C2,Responsabilidades,Percurso/Distancia):-
    trajetoDpRec_Resp([C1], C2, Responsabilidades, InvPercurso, 0, Distancia),
    inverso(InvPercurso, Percurso).

trajetoDpRec_Resp([H | T], H, _, [H | T], Distancia, Distancia).

trajetoDpRec_Resp([H | T], B, Resp, Percurso, CurrDistancia, Distancia) :-
    cidadesLigadasDist_Resp(H,C,Resp,Dist),
    \+member(C, [H | T]),
    NewDistancia is CurrDistancia + Dist,
    trajetoDpRec_Resp([C, H | T], B, Resp, Percurso, NewDistancia, Distancia).
```

Figura 2.4: Pesquisa em Profundidade restringindo o tipo de responsabilidades administrativas das cidade

Pesquisa informada - A estrela

```
% Pesquisa A* restringindo o tipo de responsabilidades administrativas das cidade

trajeto_A_Resp(ID, Objetivo,Resp,Caminho/Comprimento):-
    distanciaEntreCidades(ID, Objetivo, Estima),
    aestrela_Resp([[ID]/0/Estima], InvCaminho/Comprimento/_, Objetivo,Resp),
    inverso(InvCaminho, Caminho).

aestrela_Resp(Caminhos, Caminho, Objetivo,_):-
    obtem_melhor(Caminhos, Caminho),
    Caminho = [Objetivo|_]/_/_.

aestrela_Resp(Caminhos, SolucaoCaminho, Objetivo,Resp) :-
    obtem_melhor(Caminhos, MelhorCaminho),
    seleciona(MelhorCaminho, Caminhos, OutrosCaminhos),
    expande_aestrela_Resp(MelhorCaminho, ExpCaminhos, Objetivo,Resp),
    append(OutrosCaminhos, ExpCaminhos, NovoCaminhos),
    aestrela_Resp(NovoCaminhos, SolucaoCaminho, Objetivo,Resp).

expande_aestrela_Resp(Caminho, ExpCaminhos, Objetivo,Resp) :-
    findall(NovoCaminho, adjacente_Resp(Caminho,NovoCaminho, Objetivo,Resp), ExpCaminhos).


adjacente_Resp([Nodo|Caminho]/Custo/_, [ProxNodo,Nodo|Caminho]/NovoCusto/Est, Objetivo,Resp) :-
    cidadesLigadasDist_Resp(Nodo, ProxNodo,Resp,Resultado),
    \+member(ProxNodo, Caminho),
    NovoCusto is Custo + Resultado,
    distanciaEntreCidades(ProxNodo, Objetivo, Est).
```

Figura 2.5: Pesquisa A* restringindo o tipo de responsabilidades administrativas das cidade

Pesquisa não informada - Depth First

```
% Pesquisa em Profundidade restringindo o tipo de cultura das cidade

trajetoDp_Cult(C1,C2,Percurso/Distancia):-
    trajetoDpRec_Cult([C1], C2, InvPercurso, 0, Distancia),
    inverso(InvPercurso, Percurso).

trajetoDpRec_Cult([H | T], H, [H | T], Distancia, Distancia).

trajetoDpRec_Cult([H | T], B, Percurso, CurrDistancia, Distancia) :-
    cidadesLigadasDist_Cult(H,C,Dist),
    \+member(C, [H | T]),
    NewDistancia is CurrDistancia + Dist,
    trajetoDpRec_Cult([C, H | T], B, Percurso, NewDistancia, Distancia).
```

Figura 2.6: Pesquisa em Profundidade restringindo o tipo de cultura das cidade

Pesquisa informada - A estrela

```
% Pesquisa A* restringindo o tipo de cultura das cidade

trajeto_A_Cult(ID, Objetivo,Caminho/Comprimento):-
    distanciaEntreCidades(ID, Objetivo, Estima),
    aestrela_Cult([[ID]/0/Estima], InvCaminho/Comprimento/_, Objetivo),
    inverso(InvCaminho, Caminho).

aestrela_Cult(Caminhos, Caminho, Objetivo):-
    obtem_melhor(Caminhos, Caminho),
    Caminho = [Objetivo|_]/_/_.

aestrela_Cult(Caminhos, SolucaoCaminho, Objetivo) :-
    obtem_melhor(Caminhos, MelhorCaminho),
    seleciona(MelhorCaminho, Caminhos, OutrosCaminhos),
    expande_aestrela_Cult(MelhorCaminho, ExpCaminhos, Objetivo),
    append(OutrosCaminhos, ExpCaminhos, NovoCaminhos),
    aestrela_Cult(NovoCaminhos, SolucaoCaminho, Objetivo).

expande_aestrela_Cult(Caminho, ExpCaminhos, Objetivo) :-
    findall(NovoCaminho, adjacente_Cult(Caminho,NovoCaminho, Objetivo), ExpCaminhos).


adjacente_Cult([Nodo|Caminho]/Custo/_, [ProxNodo,Nodo|Caminho]/NovoCusto/Est, Objetivo) :-
    cidadesLigadasDist_Cult(Nodo, ProxNodo,Resultado),
    \+member(ProxNodo, Caminho),
    NovoCusto is Custo + Resultado,
    distanciaEntreCidades(ProxNodo, Objetivo, Est).
```

Figura 2.7: Pesquisa A* restringindo o tipo de cultura das cidade

2.4.3 Excluir uma ou mais características de cidades para um percurso

Pesquisa não informada - Depth First

```
% Pesquisa em Profundidade para Excluir o tipo de responsabilidades administrativas das cidade

trajetoDp_Exc(C1,C2,Responsabilidades,Percurso/Distancia):-
    trajetoDpRec_Exc([C1], C2, Responsabilidades, InvPercurso, 0, Distancia),
    inverso(InvPercurso, Percurso).

trajetoDpRec_Exc([H | T], H, _, [H | T], Distancia, Distancia).

trajetoDpRec_Exc([H | T], B, Resp, Percurso, CurrDistancia, Distancia) :-
    cidadesLigadasDist_Exc(H,C,Resp,Dist),
    \+member(C, [H | T]),
    NewDistancia is CurrDistancia + Dist,
    trajetoDpRec_Exc([C, H | T], B, Resp, Percurso, NewDistancia, Distancia).
```

Figura 2.8: Pesquisa em Profundidade para Excluir o tipo de responsabilidades administrativas das cidade

Pesquisa informada - A estrela

```
% Pesquisa A* para Excluir o tipo de responsabilidades administrativas das cidade

trajeto_A_Exc(ID, Objetivo,Resp,Caminho/Comprimento):-
    distanciaEntreCidades(ID, Objetivo, Estima),
    aestrela_Exc([[ID]/0/Estima], InvCaminho/Comprimento/_, Objetivo,Resp),
    inverso(InvCaminho, Caminho).

aestrela_Exc(Caminhos, Caminho, Objetivo,_):-
    obtem_melhor(Caminhos, Caminho),
    Caminho = [Objetivo|_]/_/_.

aestrela_Exc(Caminhos, SolucaoCaminho, Objetivo,Resp) :-
    obtem_melhor(Caminhos, MelhorCaminho),
    seleciona(MelhorCaminho, Caminhos, OutrosCaminhos),
    expande_aestrela_Exc(MelhorCaminho, ExpCaminhos, Objetivo,Resp),
    append(OutrosCaminhos, ExpCaminhos, NovoCaminhos),
    aestrela_Exc(NovoCaminhos, SolucaoCaminho, Objetivo,Resp).

expande_aestrela_Exc(Caminho, ExpCaminhos, Objetivo,Resp) :-
    findall(NovoCaminho, adjacente_Exc(Caminho,NovoCaminho, Objetivo,Resp), ExpCaminhos).


adjacente_Exc([Nodo|Caminho]/Custo/_, [ProxNodo,Nodo|Caminho]/NovoCusto/Est, Objetivo,Resp) :-
    cidadesLigadasDist_Exc(Nodo, ProxNodo,Resp,Resultado),
    \+member(ProxNodo, Caminho),
    NovoCusto is Custo + Resultado,
    distanciaEntreCidades(ProxNodo, Objetivo, Est).
```

Figura 2.9: Pesquisa A* para Excluir o tipo de responsabilidades administrativas das cidade

2.4.4 Identificar num determinado percurso qual a cidade com o maior número de ligações

Pesquisa não informada - Depth First

```
% Pesquisa identificar num determinado percurso qual a cidade com o maior número de ligações

maisLigacoes([ID |[]], ID/R):- numeroLigacoes(ID,R).

maisLigacoes([ID1,ID2|T],ID1/Res):-
    cidadesLigadas(ID1,ID2),
    numeroLigacoes(ID1,Res),
    maisLigacoes([ID2|T], _/R),
    R < Res.

maisLigacoes([ID1,ID2|T],Paragens/R):-
    cidadesLigadas(ID1,ID2),
    numeroLigacoes(ID1, Res),
    maisLigacoes([ID2|T], Paragens/R),
    Res < R.

maisLigacoes([ID1,ID2|T], [ID1 |Paragens]/Res):-
    cidadesLigadas(ID1,ID2),
    numeroLigacoes(ID1,Res),
    maisLigacoes([ID2|T], Paragens/Res).
```

Figura 2.10: Pesquisa identificar num determinado percurso qual a cidade com o maior número de ligações

2.4.5 Escolher o menor percurso (usando o critério do menor número de cidades percorridas

Pesquisa não informada - Depth First

```
% Predicado menor percurso, o critério do menor número de cidades percorridas, A*

trajeto_A_Less(ID, Objetivo, Caminho/Nparagens):-
    distanciaEntreCidades(ID, Objetivo, Estima),
    aestrela_Less([[ID]/0/Estima], InvCaminho/Nparagens/_, Objetivo),
    inverso(InvCaminho, Caminho).

aestrela_Less(Caminhos, Caminho, Objetivo):-
    obtem_melhor(Caminhos, Caminho),
    Caminho = [Objetivo|_]/_/_.

aestrela_Less(Caminhos, SolucaoCaminho, Objetivo) :-
    obtem_melhor(Caminhos, MelhorCaminho),
    seleciona(MelhorCaminho, Caminhos, OutrosCaminhos),
    expande_aestrela_Less(MelhorCaminho, ExpCaminhos, Objetivo),
    append(OutrosCaminhos, ExpCaminhos, NovoCaminhos),
    aestrela_Less(NovoCaminhos, SolucaoCaminho, Objetivo).

expande_aestrela_Less(Caminho, ExpCaminhos, Objetivo) :-
    findall(NovoCaminho, adjacente_Less(Caminho,NovoCaminho, Objetivo), ExpCaminhos).

adjacente_Less([Nodo|Caminho]/Custo/_, [ProxNodo,Nodo|Caminho]/NovoCusto/Est, Objetivo) :-
    cidadesLigadas(Nodo, ProxNodo),
    \+member(ProxNodo, Caminho),
    NovoCusto is Custo + 1,
    distanciaEntreCidades(ProxNodo, Objetivo, Est).
```

Figura 2.11: Predicado menor percurso, o critério do menor número de cidades percorridas, A*

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

Este algoritmo é mesmo que está apresentado na figura 2.3

2.4.7 Escolher um percurso que passe apenas por cidades "minor"

Pesquisa não informada - Depth First

```
% Predicado de um percurso que passe apenas por cidades "minor" Depth-First

trajetoDp_Minor(C1,C2,Percurso/Distancia):-
    trajetoDpRec_Minor([C1], C2, InvPercurso, 0, Distancia),
    inverso(InvPercurso, Percurso).

trajetoDpRec_Minor([H | T], H, [H | T], Distancia, Distancia).

trajetoDpRec_Minor([H | T], B, Percurso, CurrDistancia, Distancia) :-
    cidadesLigadasDist_Minor(H,C,Dist),
    \+member(C, [H | T]),
    NewDistancia is CurrDistancia + Dist,
    trajetoDpRec_Minor([C, H | T], B, Percurso, NewDistancia, Distancia).
```

Figura 2.12: Predicado de um percurso que passe apenas por cidades "minor" Depth-First

Pesquisa informada - A estrela

```
% Predicado de um percurso que passe apenas por cidades "minor" A*

trajeto_A_Minor(ID, Objetivo, Caminho/Comprimento):-
    distanciaEntreCidades(ID, Objetivo, Estima),
    aestrela_Minor([[ID]/0/Estima], InvCaminho/Comprimento/_, Objetivo),
    inverso(InvCaminho, Caminho).

aestrela_Minor(Caminhos, Caminho, Objetivo):-
    obtem_melhor(Caminhos, Caminho),
    Caminho = [Objetivo|_]/_/_.

aestrela_Minor(Caminhos, SolucaoCaminho, Objetivo) :-
    obtem_melhor(Caminhos, MelhorCaminho),
    seleciona(MelhorCaminho, Caminhos, OutrosCaminhos),
    expande_aestrela_Minor(MelhorCaminho, ExpCaminhos, Objetivo),
    append(OutrosCaminhos, ExpCaminhos, NovoCaminhos),
    aestrela_Minor(NovoCaminhos, SolucaoCaminho, Objetivo).

expande_aestrela_Minor(Caminho, ExpCaminhos, Objetivo) :-
    findall(NovoCaminho, adjacente_Minor(Caminho,NovoCaminho, Objetivo), ExpCaminhos).

adjacente_Minor([Nodo|Caminho]/Custo/_, [ProxNodo,Nodo|Caminho]/NovoCusto/Est, Objetivo) :-
    cidadesLigadasDist_Minor(Nodo, ProxNodo,Resultado),
    \+member(ProxNodo, Caminho),
    NovoCusto is Custo + Resultado,
    distanciaEntreCidades(ProxNodo, Objetivo, Est).
```

Figura 2.13: Predicado de um percurso que passe apenas por cidades "minor" A*

2.4.8 Escolher uma ou mais cidades intermédias por onde o percurso deverá obrigatoriamente passar

Pesquisa não informada - Depth First

```
% Predicado de uma ou mais cidades intermédias por onde o percurso deverá obrigatoriamente passar Depth-First

trajeto_Cidades_Intermedias(A, B, [], Caminho/Comprimento):-
    trajetoDpRec([A], B, InvCaminho, 0, Comprimento),
    inverso(InvCaminho, Caminho).

trajeto_Cidades_Intermedias(A,B,[H|T],Caminho/Comprimento):-
    trajetoDpRec([A], H, InvCaminho, 0, Comprimento1),
    inverso(InvCaminho, Caminho1),
    percurso_Cidades_Intermedias(H,B,T,CaminhoO/Comprimento2),
    Comprimento = Comprimento1 + Comprimento2,
    myappend(Caminho1,CaminhoO,Caminho).


myappend(L1,[_|T],R):- append(L1,T,R).
```

Figura 2.14: Predicado de uma ou mais cidades intermédias por onde o percurso deverá obrigatoriamente passar Depth-First

2.5 Comparação dos dois algoritmos de pesquisa usados

Embora o algoritmo em profundidade ser mais simples , não precisar de um algoritmo de comparação e ser geralmente mais rápido, este algoritmo normalmente produz soluções que estão muito longe da solução ideal para o problema.

No sentido contrario o algoritmo A estrela é mais complexo, tem algoritmo de comparação, mais lento mas no entanto este algoritmo produz a solução ideal para o problema seja em termos de distancia ou numero de cidades.

2.6 Testes ao sistema

Nesta secção vou por alguns exemplos de testes que achei pertinente para mostrar algumas funcionalidades do sistema.

2.6.1 Teste 1

```
?- trajetoDp(1,2,R).
R = [1, 5, 53, 56, 57, 84, 86, 95|...]/72.86183351660706 [write]
R = [1, 5, 53, 56, 57, 84, 86, 95, 97, 127, 148, 154, 157, 169, 170, 175, 181, 184, 186, 188, 189, 191,
239, 244, 279, 17, 47, 58, 62, 70, 76, 85, 100, 122, 132, 137, 144, 145, 146, 149, 152, 153, 155, 173, 1
76, 178, 187, 190, 206, 207, 227, 232, 246, 250, 275, 283, 285, 11, 48, 59, 63, 66, 96, 98, 119, 121, 13
1, 133, 166, 177, 201, 210, 211, 217, 230, 234, 238, 16, 19, 29, 37, 42, 43, 46, 90, 92, 147, 158, 171,
172, 199, 205, 213, 225, 242, 257, 260, 271, 280, 20, 27, 36, 109, 111, 128, 136, 160, 163, 164, 167, 18
0, 224, 235, 243, 255, 262, 6, 33, 49, 78, 91, 106, 117, 120, 125, 134, 138, 139, 140, 143, 159, 196, 20
2, 237, 259, 114, 129, 156, 197, 218, 223, 231, 236, 277, 278, 99, 221, 240, 258, 151, 185, 216, 226, 22
9, 9, 31, 64, 174, 195, 233, 256, 266, 281, 13, 93, 165, 142, 215, 65, 79, 80, 82, 88, 124, 141, 253, 26
1, 265, 21, 35, 39, 116, 150, 219, 252, 15, 44, 89, 241, 282, 87, 110, 274, 18, 102, 107, 108, 192, 212,
 269, 45, 55, 61, 68, 83, 94, 103, 105, 179, 182, 209, 247, 248, 272, 26, 54, 71, 104, 115, 123, 204, 2]
/72.86183351660706 ,

?- trajeto_A(1,2,R).
R = [1, 197, 47, 11, 159, 10, 235, 2]/2.546488353573151 
```

Figura 2.15: Pesquisa de um trajeto possível entre duas cidades em profundidade e A estrela

2.6.2 Teste 2

```
?- trajetoDp_Exc(1,2,admin,R).
R = [1, 53, 56, 57, 84, 86, 95, 97, 127, 148, 154, 157, 169, 170, 175, 181, 184, 186, 188, 189, 191, 239
, 244, 279, 47, 58, 62, 70, 76, 85, 100, 122, 132, 137, 144, 145, 146, 149, 152, 153, 155, 173, 176, 178
, 187, 190, 206, 207, 232, 246, 250, 275, 283, 285, 101, 114, 117, 120, 125, 134, 138, 139, 140, 143, 15
9, 164, 167, 180, 213, 225, 242, 257, 260, 271, 280, 20, 27, 36, 109, 119, 121, 131, 133, 166, 177, 201,
 210, 211, 217, 230, 234, 238, 37, 42, 43, 46, 90, 92, 98, 161, 162, 163, 224, 235, 243, 255, 262, 6, 33
, 49, 78, 91, 106, 141, 142, 165, 174, 195, 229, 113, 135, 222, 263, 8, 25, 34, 73, 75, 126, 245, 249, 2
51, 273, 276, 284, 38, 198, 254, 266, 281, 31, 64, 200, 233, 261, 265, 39, 60, 79, 80, 82, 88, 124, 35,
65, 215, 216, 226, 258, 277, 278, 99, 129, 156, 197, 218, 223, 231, 236, 259, 168, 183, 196, 202, 237, 4
8, 59, 63, 66, 96, 115, 123, 136, 160, 204, 209, 247, 248, 272, 26, 54, 55, 61, 68, 83, 94, 103, 105, 10
8, 110, 147, 158, 171, 172, 199, 205, 274, 29, 41, 45, 72, 102, 107, 51, 87, 194, 214, 203, 268, 269, 19
2, 212, 74, 104, 128, 179, 182, 40, 50, 67, 71, 2]/75.70109319728672 ,

?- trajeto_A_Exc(1,2,admin,R).
R = [1, 197, 47, 259, 237, 262, 167, 2]/2.5532968243319183 
```

Figura 2.16: Pesquisa de um trajeto possível entre duas cidades em profundidade e A estrela em que a permissões não pode ser admin

2.6.3 Teste 3

```
?- maisLigacoes([1,197,47,259,237,262,167,2],R).
R = [197|2]/33 .
```

Figura 2.17: identificar num determinado percurso qual a cidade com o maior número de ligações

Capítulo 3

Conclusão

Para concluir, este trabalho permitiu compreender melhor os dois tipos de pesquisa, informada e não informada, as suas vantagens e limitações alem de também permitir conhecer melhor a linguagem prolog.

Visto isto eu acho que o trabalho realizado é positivo dado ao problema proposto e tenho a certeza que me trouxe competências que vão ser valiosas num futuro próximo seja a nível pessoal ou profissional.