



Universidade Minho

Mestrado Integrado em Engenharia Informática
Sistemas de Representação de Conhecimento e Raciocínio

MÉTODOS DE RESOLUÇÃO DE PROBLEMAS E DE PROCURA

AUTOR
Nuno Silva - A78156

14 de Julho de 2020

Resumo

O presente relatório, possui como finalidade, a descrição das soluções implementadas, para os desafios impostos pela equipa docente da unidade curricular Sistemas de Representação de Conhecimento e Raciocínio.

O propósito deste trabalho, consiste na representação e modelação de um sistema de conexões entre cidades de Portugal, e por sua vez, a aplicação de algoritmos de pesquisa informada e não-informada. Assim sendo, foi necessário o processamento do *dataset* disponibilizado de forma de um grafo, as ligações entre cidades de todo o país.

Por fim, todo este desenvolvimento, resultará na construção de conexões, que por sua vez permitiram a travessia dessas mesmas.

Conteúdo

1	Introdução	1
2	Geração dos Grafos	2
2.1	Tratamento Dados	2
2.2	Adição de Características	2
2.3	Conexões Inter-cidades	4
3	Algoritmos de Pesquisa	5
4	Implementação da Solução	6
4.1	Pesquisa Não Informada	6
4.1.1	Query 1: Calcular um trajeto possível entre duas cidades	6
4.1.2	Query 2: Selecionar apenas cidades,com uma determinada característica,para um determinado trajeto	7
4.1.3	Query 3: Excluir uma ou mais características de cidades para um percurso	7
4.1.4	Query 4: Identificar num determinado percurso qual a cidade com o maior número de ligações	8
4.1.5	Query 5: Escolher o menor percurso (usando o critério do menor número de cidades percorridas)	9
4.1.6	Query 6: Escolher o percurso mais rápido (usando o critério da distância)	10
4.1.7	Query 7: Escolher um percurso que passe apenas por cidades “minor”	10
4.1.8	Query 8: Escolher uma ou mais cidades intermédias por onde o percurso deverá obrigatoriamente passar.	11
4.1.9	Query Extra: Cálculo de percursos que passam por pelo menos N monumentos	12
4.2	Pesquisa Informada	13
5	Conclusão	14
6	Apêndice	15

Introdução

Este trabalho teve como finalidade a representação de um sistema conexões inter-cidades de Portugal, e por sua vez, aplicação de algoritmos de pesquisa, através do uso de Prolog.

O sistema de conexão baseia-se na disposição geográfica das cidades, existindo assim cidades, representando as informações dessa mesma, e arcos, que por sua vez representam as conexões possíveis de uma cidade e o seu tamanho/distância. Tudo isto, é possível dado ao processamento do *dataset* disponibilizado.

Para a criação de ligações entre cidades, foram adaptados algoritmos, de pesquisa informada e não-informada, em que cada um deles apresenta as suas vantagens consoante o ambiente em questão.

Geração dos Grafos

Como solicitado no enunciado, era necessário o *parse* dos dados e por conseguinte a criação de ligações entre cidades. Assim sendo, foi escolhida a linguagem de programação **Python**, esta apresenta diversas vantagens, relativamente ao manuseamento de dados, e de fácil *syntax*. Assim sendo, foram criados dois ficheiros, o **idades.pl** e o **arcos.pl**, onde no primeiro se encontra as informações de cada uma das cidades, como por exemplo o seu *Id*, *Latitude*, *Longitude*, etc. Já para o ficheiro *arcos.pl*, neste estão presentes todas as conexões possíveis entre cidades e a distância entre elas.

2.1 Tratamento Dados

Após a leitura do *dataset* disponibilizado para um *dataframe*, possibilitado pela biblioteca **pandas**, é necessário o tratamento de dados, a uma primeira vista, verificámos que a cidade de Lisboa se encontra escrita em inglês, onde de imediato é trocado pelo nome da cidade em português.

De seguida, é evidente que diversas cidades apresentam acentos, podendo resultar em conflitos numa fase mais tardia, dessa forma todas *strings* existentes foram tornadas *ascii friendly*, através dos seguintes comandos:

```
#Replace all accents, to become ascii compatible
dataset['city'] = dataset['city'].str.normalize('NFKD').str.encode('ascii',
↪ errors='ignore').str.decode('utf-8')
dataset['admin'] = dataset['admin'].str.normalize('NFKD').str.encode('ascii',
↪ errors='ignore').str.decode('utf-8')
dataset['capital'] = dataset['capital'].str.normalize('NFKD').str.encode('ascii',
↪ errors='ignore').str.decode('utf-8')
```

2.2 Adição de Características

Numa análise sobre o enunciado, podemos constatar, que nos é solicitado a adição de características. Dessa forma, foram recolhidos diversos monumentos conhecidos a nível nacional, e agregados num dicionário. Onde a partir da cidade/chave podemos aceder a uma lista de monumentos presentes nessa cidade, como podemos verificar de seguida.

```
#Dictionary with the connections between districts and famous monuments
mon_dict = {
    'Braga' : ['Bom Jesus do Monte','Santuário da Nossa Senhora do Sameiro','Mosteiro
↪ Tibães','Ponte da Mizarela','Castelo Guimarães','Paco dos duques'],
    'Evora' : ['Cromleque dos Almendres','Templo de Diana','Convento dos Loios','Mosteiro
↪ dos Ossos'],
    'Faro' : ['Arco da Vila','Ruínas de Milreu','Palácio de Estoi','Rosa dos
↪ Ventos','Castelo de Silves'],
```

```

'Porto' : ['Cemiterio de Agramonte','Monumento aos Herois da Guerra
↳ Peninsular','Palacio da Bolsa','Torre dos Clerigos','Aquaduto de Santa
↳ Clara','Igreja de Sao Goncalo'],
'Lisboa' : ['Cristo Rei','Arco Triunfal da Rua Augusta','Torre de Belem','Padrao dos
↳ Descobrimentos','Panteao Nacional','Monumento aos Combatentes do
↳ Ultramar','Mosteiro dos Jeronimos','Palacio da Pena','Quinta da
↳ Regaleira','Palacio de Monserrate','Palacio Queluz','Convento de Mafra'],
'Guarda' : ['Se da Guarda','Fortaleza de Almeida','Castelo de Sabugal','Castelo de
↳ Trancoso','Castelo de Folgoso'],
'Viana do Castelo' : ['Santa Luzia','Santuário de Nossa Senhora da Peneda','Ponte de
↳ Ponte Lima','Espigueiros de Soajo','Ponte Medieval de Vilar de Mouros','Palacio
↳ da Brejoeira'],
'Coimbra' : ['Portugal dos Pequeninos','Biblioteca Joanina da Universidade de
↳ Coimbra','Mosteiro de Santa Clara-a-Velha'],
'Santarem' : ['Castelo de Almourol','Castelo de Ourem','Castelo Templario de
↳ Tomar','Convento de Cristo','Aquaduto dos Pegoes','Santuário de Fatima'],
'Viseu' : ['Catedral de Viseu','Mosteiro de Sao Joao de Tarouca','Castelo de
↳ Penedono','Ruínas do Castelo de Sernacelhe','Santuário nossa Senhora dos
↳ Remedios','Castelo de Lamego','Catedral de Lamego'],
'Beja' : ['Convento da Nossa Senhora da Conceicao','Castelo de Mertola','Castelo de
↳ Noudar','Castelo de Beja','Castelo de Serpa','Pulo do Lobo'],
'Portalegre' : ['Castelo de Portalegre','Torre de Atalaiao','Castelo de
↳ Alegrete','Forte da Graca'],
'Castelo Branco' : ['Santuário de Nossa Senhora de Almortao','Castelo de Castelo
↳ Branco','Castelo de Monsanto','Castelo de Penha','Ruínas Romanas de Centum
↳ Cellas','Castelo de Belmonte'],
'Setubal' : ['Santuário de Nossa Senhora do Cabo','Ruínas Romanas de
↳ Mirobriga','Igreja Convento de Jesus Setubal','Convento da Nossa Senhora da
↳ Arrabida'],
'Leiria' : ['Grutas de Santo Antonio','Grutas de Mira de Aire','Grutas de
↳ Alvados','Grutas da Moeda','Mosteiro da Batalha','Mosteiro de Alcobaca','Castelo
↳ Obidos'],
'Vila Real' : ['Igreja Matriz Alijo','Palacio Mateus','Igreja Nossa Senhora
↳ Guadalupe','Pelourinho de Vila Real','Parque Natural do Alvao','Ponte romana de
↳ Trajano'],
'Braganca' : ['Castelo de Braganca','Gravuras Rupestres de Mazouco','Ponte Medieval
↳ de Mirandela','Castelo de Miranda do Douro'],
'Aveiro' : ['Salinas de Aveiro','Convento de Arouca','Castelo de Santa Maria da
↳ Feira','Igreja Paroquial de valega','Palacio do Bucaco']

```

}

Para facilitar uma possível pesquisa relacionada com estes monumentos, foi adicionado um campo que poderá tomar o valor de 'Sim' ou 'Não', estes representam se a dada cidade, possui monumentos. Estas características, senão adicionadas ao registo da sua cidade, que posteriormente será exportada para o ficheiro *idades.pl*.

2.3 Conexões Inter-cidades

A partir do *dataset* disponibilizado, é nos impossível estabelecer uma ligação, por exemplo entre as cidades 'Braga' e 'Lisboa', isto pois, não nos é apresentado nada para além das coordenadas de cada uma. Assim, e tendo por base um mapa geográfico de Portugal Continental, foi possível definir conexões entre distritos, ou seja, um dado distrito apresentará uma lista de distritos, com o qual partilha a sua fronteira. Esta informação, esta presente num dicionário definido manualmente.

```
#Dictionary with the connections between districts
dist_dict = {
    'Braga' : ['Porto', 'Viana do Castelo', 'Vila Real'],
    'Viana do Castelo' : ['Braga'],
    'Porto' : ['Braga', 'Vila Real', 'Aveiro', 'Viseu'],
    'Vila Real' : ['Porto', 'Braga', 'Braganca', 'Viseu'],
    'Braganca' : ['Vila Real', 'Guarda', 'Viseu'],
    'Aveiro' : ['Viseu', 'Porto', 'Coimbra'],
    'Viseu' : ['Aveiro', 'Porto', 'Vila Real', 'Guarda', 'Braganca', 'Coimbra'],
    'Guarda' : ['Braganca', 'Castelo Branco', 'Viseu', 'Coimbra'],
    'Coimbra' : ['Aveiro', 'Viseu', 'Leiria', 'Castelo Branco', 'Guarda'],
    'Castelo Branco' : ['Guarda', 'Coimbra', 'Leiria', 'Santarem', 'Portalegre'],
    'Leiria' : ['Coimbra', 'Castelo Branco', 'Santarem', 'Lisboa'],
    'Santarem' : ['Leiria', 'Lisboa', 'Setubal', 'Portalegre', 'Evora', 'Castelo Branco'],
    'Portalegre' : ['Castelo Branco', 'Santarem', 'Evora'],
    'Lisboa' : ['Leiria', 'Santarem', 'Setubal'],
    'Setubal' : ['Evora', 'Beja', 'Santarem', 'Lisboa'],
    'Evora' : ['Setubal', 'Santarem', 'Portalegre', 'Beja'],
    'Beja' : ['Setubal', 'Evora', 'Faro'],
    'Faro' : ['Beja']
}
```

Dado que nos é apresentado também outras cidades de cada distrito, falta então apresentar as conexões para essas mesmas. Dessa forma, e com intuito de não criar um grafo demasiado denso e complexo, foi estabelecido que para as cidades menores, estas conectam-se apenas com uma cidade *admin* do seu distrito. E para conexões entre dois distritos, é feita através das suas capitais.

Pela figura anteriormente apresentada podemos visualizar que existe na mesma ligações entre as cidades *minor*, contudo o percurso terá de passar pela sua capital de distrito. Ou seja um percurso entre Guimarães e Esposende, terá de passar obrigatoriamente por Braga.



Figura 1: Representação das ligações entre cidades

Algoritmos de Pesquisa

Para a resolução das *queries* que posteriormente iremos apresentar, foi nos solicitado o uso de algoritmos de pesquisa. Estes poderão ser em duas secções, **Pesquisa Informada** e **Pesquisa Não Informada**. Cada um destes tipos apresenta as suas vantagens e desvantagens, relativamente à pesquisa não informada podemos destacar o algoritmo *Depth First*. Este requiere um tempo de procura maior para caminhos mais complexos, porém quando encontra uma solução depressa apresenta alternativas. Existe ainda um outro, denominado *Breadth First*, contudo este exige gastos de memória muito avultados, e o seu tempo de pesquisa é superior ao algoritmo apresentado anteriormente, assim sendo a possibilidade da sua implementação foi descartada.

Tendo em conta que um grafo poderá ser representado sob a forma de árvore, então é nos facilitado a estimação da complexidade e do tempo de pesquisa, como poderemos verificar de seguida. Denotar que b pretende representar o fator de ramificação, d a profundidade da solução e m o tamanho da árvore, ou seja a sua profundidade máxima.

Relativamente a algoritmos de pesquisa informada, estes fazem uso de heurísticas como forma de auxílio à pesquisa. A heurística seleccionada para o problema que nos debruça-mos será a distância euclidiana entre a origem e o destino, este calculo é feito utilizando as latitudes e longitudes de cada um dos pontos. Para este tipo de algoritmos, foi apenas implementado o algoritmo A^* .

	Depth First	Breadth First
Início Procura	Origem	Origem
Complexidade	$O(bm)$	$O(b^d)$
Óptima?	Não	Sim
Completa?	Não	Sim
Tempo pesquisa 1ª solução	$O(b^m)$	$O(b^d)$
Apresenta mais do que uma solução?	Sim	Não

Tabela 1: Tabela comparativa para algoritmos de pesquisa não informada

	A*
Conhecimento	Sim
Eficiente?	Muito
Óptima?	Sim
Completa?	Sim
Tempo pesquisa 1ª solução	Baixo

Implementação da Solução

4.1 Pesquisa Não Informada

De seguida serão apresentadas as implementações feitas para a pesquisa não informada, juntamente com uma breve explicação sobre predicados auxiliares, e sobre casos de teste, como poderemos ver a seguir.

4.1.1 Query 1: Calcular um trajeto possível entre duas cidades

O método de resolução desta questão baseia-se na testagem de diferentes possibilidades, em que é acrescentado um campo denominado *Visitados*, este tem como objectivo registar todas as cidades já percorridas, de modo a evitar possíveis *loops* infinitos.

Neste tipo de implementações, a lista referente ao percurso, apenas será preenchida quando a Origem equivale ao Destino.

A figura a cima representa todos os percursos sugeridos e a respectiva distância percorrida. No exemplo acima mostrado, pretendemos os caminhos entre a cidade de Lisboa, representado pelo *id* 1, e a cidade de Barcelos, com *id* 209. De forma a concluir esta pesquisa, foi definido o seguinte caso de paragem.

```
caminhoAux(Destino, Destino, [], 0, _).
```

Como podemos ver, a pesquisa termina quando a Origem equivale ao Destino, a lista referente ao percurso, encontra-se vazia, pois o seu conteúdo é agora o da lista Visitados e a distância entre os dois pontos será 0, como esperado.

```
| ?- getCaminho(1,209,L).
[1,17,9,21,15,16,18,111,3,209],6.092930230055972
[1,17,9,21,15,16,18,111,32,3,209],6.696161663774582
[1,17,9,21,15,16,18,32,3,209],6.010251214221826
[1,17,9,21,15,16,18,32,111,3,209],7.166858889844752
[1,17,9,21,15,16,18,227,10,32,3,209],6.839076716558706
[1,17,9,21,15,16,18,227,10,32,111,3,209],7.995684392181632
[1,17,9,21,15,16,18,10,32,3,209],6.398077555815531
_
```

Figura 2: Excerto output da query 1

4.1.2 Query 2: Selecionar apenas cidades,com uma determinada característica,para um determinado trajecto

A linha de pensamento referente a esta questão será a mesma que a da *query* anterior. Porém nesta, através dos *ids* dos arcos adjacentes, foi utilizado o predicado *getCar*, de forma a obter a característica, neste caso se possui monumentos ou não. A partir daí é feita uma verificação se a próxima cidade se encontra na lista dos visitados e a característica corresponde com requerido.

```
| ?- getCaracteristica(3,71,'Sim').
[3,32,71],0.688096360314278
```

Figura 3: Output da query 2

A figura anteriormente apresentada, mostra a procura de um percurso entre Braga e Vila do Conde, em que durante este percurso, todas as cidades têm de possuir monumentos.

4.1.3 Query 3: Excluir uma ou mais características de cidades para um percurso

Esta query apresenta-se muito idêntica, com a anterior, porém para este caso pretendemos evitar cidades que possuam certas características, neste caso a característica será um monumento indicado. Esta exclusão é possibilitada com o uso da negação do predicado auxiliar:

```
temElem([],_).
temElem(L,[H|T]):- member(H,L).
temElem(L,[H|T]):- temElem(L,T);memberchk(H,L).
```

Este mesmo, verifica se pelo menos um elemento de uma lista se encontra noutra.

No caso acima exposto, pretendemos os caminhos entre Lisboa e Braga, em que uma das cidades percorridas não pode conter o monumento "*Palacio da Bolsa, Templo de Diana e Portugal dos Pequeninos*", neste caso não poderá passar pelas cidades do Porto, Evora e Coimbra.

```
| ?- excluiCaracteristica(1,3,['Palacio da Bolsa','Templo de Diana','Portugal dos Pequeninos']).
[1,17,11,15,16,18,111,3],5.3167093292076935
[1,17,11,15,16,18,14,111,3],7.389209034247502
[1,17,11,15,16,14,111,3],6.475645411523772
[1,17,11,15,16,14,18,111,3],7.652076474865456
[1,17,21,15,16,18,111,3],5.25521200753559
[1,17,21,15,16,18,14,111,3],7.3277117125753986
[1,17,21,15,16,14,111,3],6.414148089851668
[1,17,21,15,16,14,18,111,3],7.590579153193353
[1,17,15,16,18,111,3],4.801826241293389
```

Figura 4: Excerto do output da query 3

4.1.4 Query 4: Identificar num determinado percurso qual a cidade com o maior número de ligações

A linha de pensamento para com esta query, é similar à primeira contudo, é necessário fazer uma contabilização de todos os arcos que uma dada cidade possui. Assim sendo, foram criados dois novos predicados auxiliares, que de seguida serão apresentados e explicitados.

```
cidadeVizinha(Origem,Proximo,Dist,Menor) :-
    getArco(Origem,Proximo,Dist),
    nrLig(Proximo,Menor).
```

O predicado anteriormente indicado, numa primeira fase recolhe o *id* do próximo destino de um arco, tendo em conta a sua origem, de seguida este *id*, é passado como argumento para um outro predicado com a finalidade de calcular o número de ligações.

```
nrLig(Id,(Id,L)) :-
    findall(X,getArco(Id,_,_),Z),
    comprimento(Z,L).
```

Neste predicado, a partir do *id*, recolhemos todos os arcos em que este é uma origem, através de um **findall**. Este mesmo, retorna uma lista com todos esses arcos, que por sua vez, é calculada o seu comprimento, representado assim o número de ligações.

```
| ?- maisLigacoes(182,94).
[(182,1),(3,17),(94,1)],[182,3,94]
yes
```

Figura 5: Output da query 4

Nesta teste, pretendemos demonstrar o percurso entre Vieira do Minho e Esposende, e por sua vez indicar o número de ligações de cada uma das cidades percorridas. Como podemos ver, a cidade de Braga apresenta 17 ligações, enquanto as outras duas apresentam apenas 1.

4.1.5 Query 5: Escolher o menor percurso (usando o critério do menor número de cidades percorridas)

Dado que nesta *query* apenas pretendemos os percursos com o menor número de cidades, podemos reaproveitar o *output* de um dos predicados utilizados na *query* 1. Tendo por isso os percursos já "recolhidos", é necessário um predicado para obter o tamanho de um percurso/lista, outro para obter o mínimo de todas as listas, e por fim outro destinado a excluir todos percursos com o número de cidades superiores ao mínimo calculado.

```
tamanhoListaLig([], []).
tamanhoListaLig([(H,T)],[(X,H),T]) :- comprimento(H,X).
tamanhoListaLig([(H,T)|Z],[(X,H),T)|Y]) :-
    → comprimento(H,X), tamanhoListaLig(Z,Y).
```

Este predicado, tal como já indicado, tem a finalidade de calcular o tamanho de cada percurso.

Com o calculo do tamanho de cada um dos percursos, é agora necessário obter o mínimo de todas as listas, geradas anteriormente pelo predicado **tamanhoListaLig**.

```
minimoListas([(A,_),_],A).
minimoListas([(A,_),_|L],B):- minimoListas(L,B), B <= A.
minimoListas([(A,_),_|L],A):- minimoListas(L,B), A < B.
```

Por fim, falta excluir as listas/percursos que não são pretendidos, ou seja as que apresentam um número de percursos maior.

```
filtroListas([],_,[]).
filtroListas([(X,_),_|Y],Min,R) :-
    Min \== X ,
    filtroListas(Y,Min,R).
filtroListas([(X,A),B|Y],Min,K) :-
    Min == X,
    filtroListas(Y,Min,R),
    append([(X,A),B],R,K).
```

De seguida, é apresentado o output, para o percurso entre Esposende e Armamar, em que podemos constatar que surgem dois caminhos com o mesmo número de cidades percorridas.

```
| ?- caminhoCurto(94,102).
(5,[94,3,111,18,102]),2.240641362502587
(5,[94,3,32,18,102]),2.1579623466684414
```

Figura 6: Output da query 5

4.1.6 Query 6: Escolher o percurso mais rápido (usando o critério da distância)

Para a resolução deste problema, a linha de pensamento foi similar a da *query* anterior. Contudo em vez de calcular o número de cidades percorridas apenas será necessário comparar as distâncias, e por sua vez aplicar um filtro de forma a obter apenas o/s percurso/s que apresentam uma menor distância percorrida.

Com essas necessidades foram criados os seguintes predicados.

```
mininoDist([(_,H)],H) :- !,true.
mininoDist([(_,H)|T], M) :-
    mininoDist(T, M), M =< H.
mininoDist([(_,H)|T], H) :-
    mininoDist(T, M), H < M.

filtroDist([],_,[]).
filtroDist([(_,D)|T],Min,R) :-
    Min \== D,
    filtroDist(T,Min,R).
filtroDist([(X,D)|T],Min,K) :-
    Min == D,
    filtroDist(T,Min,R),
    append([(X,D)],R,K).
```

Assim sendo, esta mesma *query* apresenta o seguinte resultado para um percurso entre Arcos de Valdevez e Lamego.

```
| ?- maisRapido(103,119).
[103,22,3,32,18,119],2.632865559955265
```

Figura 7: Output da query 6

4.1.7 Query 7: Escolher um percurso que passe apenas por cidades “minor”

Dada a forma como foi desenvolvido o *parser*, e por sua vez a conexões inter-cidades, a resolução desta *query* pela forma como é pretendida é nos impossibilitada, contudo, foi desenvolvida uma proposta de resolução, para a escolha de um percurso que apenas passe por cidades do tipo *admin*.

Assim sendo, para a resolução desta adaptação, foi seguida a mesma linha de pensamento como o da *query* 2.

```
| ?- getAdmin(22,32).
[22,3,111,18,16,227,10,32],4.674319734247984
[22,3,111,18,16,15,17,11,227,10,32],6.680583523393368
[22,3,111,18,16,15,21,9,13,5,17,11,227,10,32],9.19241390473434
[22,3,111,18,16,15,21,9,5,17,11,227,10,32],8.481047028476048
```

Figura 8: Excerto do output da query 7

A imagem acima demonstrada, representa um excerto do output da *query* desenvolvida para o percurso entre Portalegre e Porto.

4.1.8 Query 8: Escolher uma ou mais cidades intermédias por onde o percurso deverá obrigatoriamente passar.

Para a resolução desta *query*, foi idealizado, em reaproveitar o *output* da primeira *query*. Onde posteriormente é feita uma verificação, se a lista com a cidades intermédias é uma sub-lista do percurso. Caso assim o seja, essa mesma será imprimida. Isto através dos seguintes predicados auxiliares:

```
check([],_,[]).
check([(X,_)|T],L,R) :- check(T,L,R), \+temTodos(L,X).
check([(X,D)|T],L,R) :- check(T,L,K), temTodos(L,X),
    ↪ append([(X,D)],K,R),imprime(R).

%verifica se uma lista tem elementos de outra
temTodos([],_).
temTodos([H|T],L) :- member(H,L),temTodos(T,L).
```

O *output* anteriormente apresentado, contém duplicados, isto pois, o predicado de impressão, encontra-se num predicado auxiliar, criado para verificar quais os percursos, cumprem com o indicado.

```

| ?- caminhosEntre(26,40,[22,3,32]).
[26,22,3,32,40],1.7712563082010169
[26,22,3,32,40],1.7712563082010169
[26,22,3,111,14,18,10,32,40],6.023664047986104
[26,22,3,32,40],1.7712563082010169
[26,22,3,32,40],1.7712563082010169
[26,22,3,32,40],1.7712563082010169
[26,22,3,111,14,18,10,32,40],6.023664047986104
[26,22,3,32,40],1.7712563082010169
[26,22,3,111,14,18,227,10,32,40],6.464663208729277
[26,22,3,111,14,18,10,32,40],6.023664047986104
[26,22,3,32,40],1.7712563082010169
[26,22,3,32,40],1.7712563082010169
[26,22,3,32,40],1.7712563082010169
[26,22,3,111,14,18,10,32,40],6.023664047986104
[26,22,3,32,40],1.7712563082010169
[26,22,3,32,40],1.7712563082010169
[26,22,3,111,14,18,10,32,40],6.023664047986104
[26,22,3,32,40],1.7712563082010169
[26,22,3,111,14,18,227,10,32,40],6.464663208729277
[26,22,3,111,14,18,10,32,40],6.023664047986104
[26,22,3,32,40],1.7712563082010169
[26,22,3,111,14,18,32,40],5.635837706392398
[26,22,3,111,14,18,227,10,32,40],6.464663208729277
[26,22,3,111,14,18,10,32,40],6.023664047986104

```

Figura 9: Excerto do output da query 8

4.1.9 Query Extra: Cálculo de percursos que passam por pelo menos N monumentos

Com intuito de aprender e melhorar, os conhecimentos em Prolog, foi criada uma *query* extra. Esta tem o objetivo, de obter todos os percursos entre dois pontos, em que existe em todos os pontos de travessia pelo menos N monumentos. A linha de pensamento, para esta mesma segue a da *query 1*, contudo apenas foi adicionada uma condição para verificar se o número de monumentos cumpre o requisitos, e um outro predicado, com o propósito de obter a lista de monumentos da cidade.

```

\%Calcula o numero de Monumentos de uma cidade
getNumMon(Origem,Proximo,N,Dist) :-
    getArco(Origem,Proximo,Dist),
    cidade(Proximo,_,_,_,_,Monumento,_, 'admin'),
    comprimento(Monumento,N).

```

Em cima podemos ver o predicado auxiliar desenvolvido, como podemos ver foi utilizado o predicado *'getArco'* para obter o *id* da próxima cidade do percurso, para por sua vez usa-lo para obter a lista de monumentos. De seguida, apresentámos o percurso entre Braga e Lisboa, em que se pretende apenas os percursos com pelo menos quatro monumentos.

```
| ?- maisMonumentos(1,3,4).
([1,17,9,21,15,16,14,111,18,32,3],6),8.295228415636839
([1,17,9,21,15,16,14,111,18,10,32,3],6),8.683054757230545
([1,17,9,21,15,16,14,111,32,3],6),7.659754398108191
([1,17,9,21,15,16,14,111,3],6),7.056522964389581
([1,17,9,21,15,16,14,18,32,111,3],6),9.306882687520048
([1,17,9,21,15,16,14,18,32,3],6),8.150275011897122
([1,17,9,21,15,16,14,18,10,32,111,3],6),9.694709029113753
([1,17,9,21,15,16,14,18,10,32,3],6),8.538101353490827
([1,17,9,21,15,16,14,18,111,32,3],6),8.836185461449878
```

Figura 10: Excerto do output da query extra

4.2 Pesquisa Informada

Aqui iremos debruçar-nos sobre pesquisa informada, aqui utilizada a informação do problema de forma a evitar que a procura entre num *loop*. Em termos de optimização pode se apresentar mais vantajosa, relativamente à função em si. Quanto à função esta é uma heurística, neste caso, é utilizado um predicado com a finalidade do cálculo da distância euclidiana entre dois pontos, apresentando sempre o mínimo para estes percursos.

Tendo em conta a heurística implementada, o desenvolvimento das queries 1 e 6, foram feitas em conjunto para este tipo de pesquisa. Assim sendo, a solução implementada, teve por base os predicados referentes a A*, disponibilizados na ficha 12. Como predicados auxiliares, foi implementado um com o propósito de fazer um *reverse* a listas, e outro para seleccionar um determinado elemento.

```
\% Tira o elemento D de uma lista
escolhe(D, [D|Ds], Ds).
escolhe(D, [D|Ds], [D|Es]) :- escolhe(D,Ds,Es).
\% reverte uma lista
reverseL(Ds,Es) :- reverseL(Ds, [], Es).
reverseL([],Ds,Ds).
reverseL([D|Ds],Es,Fs) :- reverseL(Ds, [D|Es], Fs).
```

Como tal, como caso de teste foi seleccionado o percurso entre Braga e Porto, pois apesar de possuírem uma ligação directa entre si, apresentam na mesma outras alternativas, como por exemplo uma passagem por Vila Real.

```
| ?- usaAestrela(3,32,C).
C = [(3,32),(32,32)]/0.4438835202730558 ?
```

Figura 11: Excerto do output da query 1 e 6, com Pesquisa Informada

Conclusão

Com o término deste trabalho prático, posso concluir que graças a ele pude aprofundar conhecimentos obtidos ao longo deste semestre, nas aulas teóricas e práticas. Para além do conhecimento de novos algoritmos de pesquisa como por exemplo o A^* , ao qual se revela muito eficiente.

Numa nota de curiosidade pessoal, seria interessante, a replicação deste trabalho mas fazendo uso da linguagem R, dado que foi pouco o uso dela durante as aulas.

Apêndice

Script para a geração dos arcos:

```

1 import pandas as pd
2 import numpy as np
3
4 #Read data
5 dataset = pd.read_excel(r'Data/cidades.xlsx',encoding='utf-8')
6
7 #Replace all accents, to become ascii compatible
8 dataset['city'] = dataset['city'].str.normalize('NFKD').str.encode('ascii', errors='ignore')
9 dataset['admin'] = dataset['admin'].str.normalize('NFKD').str.encode('ascii', errors='ignore').str.decode('utf-8')
10 dataset['capital'] = dataset['capital'].str.normalize('NFKD').str.encode('ascii', errors='ignore').str.decode('utf-8')
11
12 #Dictionary with the connections between districts
13 dist_dict = {
14     'Braga' : ['Porto','Viana do Castelo','Vila Real'],
15     'Viana do Castelo' : ['Braga'],
16     'Porto' : ['Braga','Vila Real','Aveiro','Viseu'],
17     'Vila Real' : ['Porto','Braga','Braganca','Viseu'],
18     'Braganca' : ['Vila Real','Guarda','Viseu'],
19     'Aveiro' : ['Viseu','Porto','Coimbra'],
20     'Viseu' : ['Aveiro','Porto','Vila Real','Guarda','Braganca','Coimbra'],
21     'Guarda' : ['Braganca','Castelo Branco','Viseu','Coimbra'],
22     'Coimbra' : ['Aveiro','Viseu','Leiria','Castelo Branco','Guarda'],
23     'Castelo Branco' : ['Guarda','Coimbra','Leiria','Santarem','Portalegre'],
24     'Leiria' : ['Coimbra','Castelo Branco','Santarem','Lisboa'],
25     'Santarem' : ['Leiria','Lisboa','Setubal','Portalegre','Evora','Castelo Branco'],
26     'Portalegre' : ['Castelo Branco','Santarem','Evora'],
27     'Lisboa' : ['Leiria','Santarem','Setubal'],
28     'Setubal' : ['Evora','Beja','Santarem','Lisboa'],
29     'Evora' : ['Setubal','Santarem','Portalegre','Beja'],
30     'Beja' : ['Setubal','Evora','Faro'],
31     'Faro' : ['Beja']
32 }
33
34 #Dictionary with the connections between districts and famous monuments
35 mon_dict = {
36     'Braga' : ['Bom Jesus do Monte','Santuário da Nossa Senhora do Sameiro','Mosteiro Tibaes','Ponte da Mizarela','Castelo Guimaraes','Paco dos duques'],
37     'Evora' : ['Cromleque dos Almendres','Templo de Diana','Convento dos Loios','Mosteiro dos Ossos'],
38     'Faro' : ['Arco da Vila','Ruínas de Milreu','Palácio de Estoi','Rosa dos Ventos','Castelo de Silves'],
39     'Porto' : ['Cemitério de Agramonte','Monumento aos Heróis da Guerra Peninsular','Palácio da Bolsa','Torre dos Clerigos','Aqueduto de Santa Clara','Igreja de São Gonçalo'],
40     'Lisboa' : ['Cristo Rei','Arco Triunfal da Rua Augusta','Torre de Belem','Padrão dos Descobrimentos','Panteão Nacional','Monumento aos Combatentes do Ultramar','Mosteiro

```

```

dos Jeronimos','Palacio da Pena','Quinta da Regaleira','Palacio de Monserrate','
Palacio Queluz','Convento de Mafra'],
41 'Guarda' : ['Se da Guarda','Fortaleza de Almeida','Castelo de Sabugal','Castelo de
Trancoso','Castelo de Folgoso'],
42 'Viana do Castelo' : ['Santa Luzia','Santuário de Nossa Senhora da Peneda','Ponte de
Ponte Lima','Espigueiros de Soajo','Ponte Medieval de Vilar de Mouros','Palacio da
Brejoeira'],
43 'Coimbra' : ['Portugal dos Pequeninos','Biblioteca Joanina da Universidade de Coimbra'
,'Mosteiro de Santa Clara-a-Velha'],
44 'Santarem' : ['Castelo de Almourol','Castelo de Ourem','Castelo Templario de Tomar','
Convento de Cristo','Aquaduto dos Pegoes','Santuário de Fatima'],
45 'Viseu' : ['Catedral de Viseu','Mosteiro de Sao Joao de Tarouca','Castelo de Penedono'
,'Ruínas do Castelo de Sernacelhe','Santuário nossa Senhora dos Remedios','Castelo de
Lamego','Catedral de Lamego'],
46 'Beja' : ['Convento da Nossa Senhora da Conceicao','Castelo de Mertola','Castelo de
Noudar','Castelo de Beja','Castelo de Serpa','Pulo do Lobo'],
47 'Portalegre' : ['Castelo de Portalegre','Torre de Atalaiao','Castelo de Alegrete','
Forte da Graca'],
48 'Castelo Branco' : ['Santuário de Nossa Senhora de Almortao','Castelo de Castelo
Branco','Castelo de Monsanto','Castelo de Penha','Ruínas Romanas de Centum Cellas','
Castelo de Belmonte'],
49 'Setubal' : ['Santuário de Nossa Senhora do Cabo','Ruínas Romanas de Mirobriga','
Igreja Convento de Jesus Setubal','Convento da Nossa Senhora da Arrabida'],
50 'Leiria' : ['Grutas de Santo Antonio','Grutas de Mira de Aire','Grutas de Alvados','
Grutas da Moeda','Mosteiro da Batalha','Mosteiro de Alcobaca','Castelo Obidos'],
51 'Vila Real' : ['Igreja Matriz Alijo','Palacio Mateus','Igreja Nossa Senhora Guadalupe'
,'Pelourinho de Vila Real','Parque Natural do Alvao','Ponte romana de Trajano'],
52 'Braganca' : ['Castelo de Braganca','Gravuras Rupestres de Mazouco','Ponte Medieval de
Mirandela','Castelo de Miranda do Douro'],
53 'Aveiro' : ['Salinas de Aveiro','Convento de Arouca','Castelo de Santa Maria da Feira'
,'Igreja Paroquial de valega','Palacio do Bucaco']
54 }
55
56 #Write complete dataset
57 full = open('cidades.pl','w+',encoding='utf-8')
58 full.write('%%cidade(ID,Cidade,Lat,Long,Distrito,Ligac es,Monumentos,temMonumentos,
Capital)\n')
59 for line in dataset.values:
60     if (line[1] in mon_dict):
61         full.write("cidade(%d,'%s',%f,%f,'%s',%s,%s,'Sim','%s').\n" %(line[0],line[1],line
[2],line[3],line[4],dist_dict[line[4]],mon_dict[line[1]],line[5]))
62     else:
63         full.write("cidade(%d,'%s',%f,%f,'%s',%s,[],'Nao','%s').\n" %(line[0],line[1],line
[2],line[3],line[4],dist_dict[line[4]],line[5]))
64 full.close()
65
66
67 #Calculate distance
68 def calc_distance(latitude1,longitude1,latitude2,longitude2):
69     result = np.sqrt( (latitude1-latitude2)**2 + (longitude1-longitude2)**2 )
70     return result
71
72
73 #Write the base knowledge, with the information of the frontiers
74 arc = open('arcos.pl','w+',encoding='utf-8')

```

```

75 arcos = set()
76 arc.write('%arco(Cidade1,Cidade2,Distancia)\n')
77 for line in dataset.values: #iter by lines
78     #Calc distance between inner citys in district
79     aux = dataset.loc[dataset['admin'] == line[4]] #Subset frame by district
80     aux2 = aux.loc[aux['capital'] == 'minor']
81     aux = aux.loc[aux['capital'] != 'minor']
82     for i in range(0,len(aux)):
83         for j in range(0,len(aux2)):
84             arcos.add("arco({}, {}, {}) \n".format(aux.values[i][0], aux2.values[j][0],
85             calc_distance(aux.values[i][2], aux.values[i][3], aux2.values[j][2], aux2.values[j][3])))
86             arcos.add("arco({}, {}, {}) \n".format(aux2.values[j][0], aux.values[i][0],
87             calc_distance(aux.values[i][2], aux.values[i][3], aux2.values[j][2], aux2.values[j][3])))
88     #Calc distance between neighbour districts
89     for distc in dist_dict[line[4]]:
90         #Neighbour district
91         aux2 = dataset.loc[dataset['admin'] == distc]
92         aux2 = aux2.loc[aux2['capital'] != 'minor']
93         aux = aux.loc[aux['capital'] != 'minor']
94         for i in range(0,len(aux)):
95             for j in range(0,len(aux2)):
96                 arcos.add("arco({}, {}, {}) \n".format(aux.values[i][0], aux2.values[j][0],
97                 calc_distance(aux.values[i][2], aux.values[i][3], aux2.values[j][2], aux2.values[j][3])))
98                 arcos.add("arco({}, {}, {}) \n".format(aux2.values[j][0], aux.values[i][0],
99                 calc_distance(aux.values[i][2], aux.values[i][3], aux2.values[j][2], aux2.values[j][3])))
100 #write to file
101 for l in arcos:
102     arc.write(l)
103 arc.close()

```

Soluções criadas:

```

1 %Declaracoes iniciais
2
3 :- set_prolog_flag(discontiguous_warnings,off).
4 :- set_prolog_flag(single_var_warnings,off).
5
6 %Definicoes iniciais
7 :- op( 900,xfy,':' ).
8 :- use_module(library(lists)).
9
10 :- include('cidades.pl').
11 :- include('arcos.pl').
12
13 %----- Predicados -----
14
15 %Extensao do meta-predicado nao
16 nao( Questao ) :- Questao, !, fail.
17 nao( Questao ).
18
19 %Calcula o comprimento de uma lista
20 comprimento(S,N) :- length(S,N).
21
22 %Predicado que da um print no terminal
23 imprime([]).

```

```

24 imprime([X|T]) :- write(X), nl, imprime(T).
25
26 %verifica se um elemento pertence a uma lista
27 temElem([],_).
28 temElem(L,[H|T]):- member(H,L).
29 temElem(L,[H|T]):- temElem(L,T);memberchk(H,L).
30
31 %Obtem um arco a partir de um id
32 getArco(Origem, Destino, Dist) :- arco(Origem, Destino, Dist).
33
34 %verifica se uma lista esta vazia
35 estaVazia(L,V) :- comprimento(L,V), nao(V>0).
36
37 %----- N o Informada -----%
38
39 %----- Query 1 -----%
40
41 %arco(IDCidade1, IDCidade2, Distancia)
42 %cidade(ID, Cidade, Lat, Long, Distrito, Liga es, Monumentos, Capital)
43
44 %getCaminho(1, 209).
45
46
47 getCaminho(Origem, Destino) :-
48     findall((P, Dist), caminho(Origem, Destino, Dist, P), L),
49     imprime(L).
50
51 caminho(Origem, Destino, Dist, [Origem|Percurso]) :-
52     caminhoAux(Origem, Destino, Percurso, Dist, []).
53
54 caminhoAux(Destino, Destino, [], 0, _).
55 caminhoAux(Origem, Destino, [Proximo|Percurso], Dist, Visitados) :-
56     Origem \= Destino,
57     getArco(Origem, Proximo, Dist1),
58     \+member(Proximo, Visitados),
59     caminhoAux(Proximo, Destino, Percurso, Dist2, [Origem|Visitados]),
60     Dist is Dist1 + Dist2.
61
62
63 %----- Query 2 -----%
64
65 %getCaracteristica(3,71, 'Sim').
66
67 getCaracteristica(Origem, Destino, Car) :-
68     findall((P, Dist), caracteristica(Origem, Destino, P, Car, Dist), L),
69     imprime(L).
70
71 caracteristica(Origem, Destino, [Origem|Percurso], Car, Dist) :-
72     caracteristicaAux(Origem, Destino, Percurso, Car, Dist, []).
73
74 caracteristicaAux(Destino, Destino, [], Car, 0, _).
75 caracteristicaAux(Origem, Destino, [Proximo|Percurso], Car, Dist, Visitados) :-
76     Origem \= Destino,
77     getCar(Origem, Proximo, Car1, Dist1),
78     \+member(Proximo, Visitados),

```

```

79 Car1 == Car,
80 caracteristicaAux(Proximo, Destino, Percurso, Car, Dist2, [Origem|Visitados]),
81 Dist is Dist1 + Dist2.
82
83 %Obtencao das caracteristicas de uma cidade apartir do id
84 getCar(Origem, Proximo, Car, Dist) :-
85     getArco(Origem, Proximo, Dist),
86     cidade(Proximo, _, _, _, _, Car, _).
87
88
89 %----- Query 3 -----%
90
91 %excluiCaracteristica(1,3,['Palacio da Bolsa','Templo de Diana','Portugal dos Pequeninos
92     '])
93
94 excluiCaracteristica(Origem, Destino, Monumentos) :-
95     findall((P, Dist), excluiCaminho(Origem, Destino, P, Monumentos, Dist), L),
96     imprime(L).
97
98 excluiCaminho(Origem, Destino, [Origem|Percurso], Monumentos, Dist) :-
99     excluiCaminhoAux(Origem, Destino, Percurso, Monumentos, Dist, []).
100
101 excluiCaminhoAux(Destino, Destino, [], _, 0, _).
102 excluiCaminhoAux(Origem, Destino, [Proximo|Percurso], Monumentos, Dist, Visitados) :-
103     Origem \= Destino,
104     getNaoVai(Origem, Proximo, Monumento, Dist1),
105     \+member(Proximo, Visitados),
106     \+temElem(Monumentos, Monumento),
107     excluiCaminhoAux(Proximo, Destino, Percurso, Monumentos, Dist2, [Origem|Visitados]),
108     Dist is Dist1 + Dist2.
109
110 %Obtencao dos Monumentos de uma cidade apartir do id
111 getNaoVai(Origem, Proximo, Monumento, Dist) :-
112     getArco(Origem, Proximo, Dist),
113     cidade(Proximo, _, _, _, _, Monumento, _, _).
114
115 %----- Query 4 -----%
116
117 %maisLigacoes(182,94).
118
119 maisLigacoes(Origem, Destino) :-
120     findall((Maior, P), mais(Origem, Destino, P, _, Maior), L),
121     imprime(L).
122
123 mais(Origem, Destino, [Origem|Percurso], Dist, M) :-
124     nrLig(Origem, C),
125     maisAux(Origem, Destino, Percurso, Dist, Maior, []),
126     append([C], Maior, M).
127
128 maisAux(Destino, Destino, [], 0, [], _).
129 maisAux(Origem, Destino, [Proximo|Percurso], Dist, Maior, Visitados) :-
130     Origem \= Destino,
131     cidadeVizinha(Origem, Proximo, Dist1, Maior1),
132     \+member(Proximo, Visitados),
133     maisAux(Proximo, Destino, Percurso, Dist2, Maior2, [Origem|Visitados]),

```

```

133 Dist is Dist1 + Dist2,
134 append([Maior1],Maior2,Maior).
135
136 %Calculo do numero ligacoes
137 nrLig(Id,(Id,L)) :-
138     findall(X,getArco(Id,_,_),Z),
139     comprimento(Z,L).
140
141 %Obtencao de um destino e calculo de ligacoes desse destino
142 cidadeVizinha(Origem,Proximo,Dist,Maior) :-
143     getArco(Origem,Proximo,Dist),
144     nrLig(Proximo,Maior).
145
146 %----- Query 5 -----%
147
148 %caminhoCurto(94,102).
149
150 caminhoCurto(Origem,Destino) :-
151     findall((P,Dist),caminho(Origem,Destino,Dist,P),C),
152     tamanhoListaLig(C,D),
153     minimoListas(D,E),
154     filtroListas(D,E,U),
155     imprime(U).
156
157 %Calculo do menores de uma lista de listas
158 minimoListas([((A,_),_)],A).
159 minimoListas([((A,_),_)|L],B):- minimoListas(L,B), B <= A.
160 minimoListas([((A,_),_)|L],A):- minimoListas(L,B), A < B.
161
162 %Elimina as listas que nao tenham valor igual ao calculado
163 filtroListas([],_,[]).
164 filtroListas([((X,_),_)|Y],Min,R) :-
165     Min \== X,
166     filtroListas(Y,Min,R).
167 filtroListas([((X,A),B)|Y],Min,K) :-
168     Min == X,
169     filtroListas(Y,Min,R),
170     append([((X,A),B)],R,K).
171
172 %Calculo do numero de cidades de um percurso
173 tamanhoListaLig([],[]).
174 tamanhoListaLig([(H,T)],[((X,H),T)]) :- comprimento(H,X).
175 tamanhoListaLig([(H,T)|Z],[((X,H),T)|Y]) :- comprimento(H,X),tamanhoListaLig(Z,Y).
176
177
178
179 %----- Query 6 -----%
180
181 %maisRapido(103,119).
182
183 maisRapido(Origem,Destino) :-
184     findall((P,Dist),caminho(Origem,Destino,Dist,P),C),
185     mininoDist(C,Dist),
186     filtroDist(C,Dist,L),
187     imprime(L).

```

```
%Procura qual os percursos com distancia minima
mininoDist([(_,H)],H) :- !,true.
mininoDist([(_,H)|T], M) :-
    mininoDist(T, M), M <= H.
mininoDist([(_,H)|T], H):-
    mininoDist(T, M), H < M.

%Elemina os percursos que nao tenham a distancia minima
filtroDist([],_,[]).
filtroDist([(_,D)|T],Min,R) :-
    Min \== D,
    filtroDist(T,Min,R).
filtroDist([(X,D)|T],Min,K) :-
    Min == D,
    filtroDist(T,Min,R),
    append([(X,D)],R,K).

%----- Query 7 -----%

%getAdmin(22,32).

getAdmin(Origem,Destino) :-
    findall((P,Dist),municipio(Origem,Destino,P,Dist,'admin'),L),
    imprime(L).

municipio(Origem,Destino,[Origem|Percurso],Dist,'admin') :-
    municipioAux(Origem,Destino,Percurso,Dist,'admin',[]).

municipioAux(Destino, Destino,[],0,_,_).
municipioAux(Origem, Destino,[Proximo|Percurso],Dist,'admin',Visitados) :-
    Origem \= Destino,
    getArco(Origem,Proximo,Dist1),
    cidade(Proximo,_,_,_,_,_,_,_,_,'admin'),
    \+member(Proximo,Visitados),
    municipioAux(Proximo, Destino,Percurso,Dist2,'admin',[Origem|Visitados]),
    Dist is (Dist1 + Dist2).

%----- Query 8 -----%

%caminhosEntre(26,40,[22,3,32])

caminhosEntre(Origem, Destino, Intermedios) :-
    findall((P,Dist), caminho(Origem, Destino, Dist, P), L),
    check(L, Intermedios, R).

%Filtra os Percursos que cumprem com o itinerario
check([],_,[]).
check([(X,_)|T],L,R) :- check(T,L,R), \+temTodos(L,X).
check([(X,D)|T],L,R) :- check(T,L,K), temTodos(L,X), append([(X,D)],K,R),imprime(R).

%verifica se uma lista tem elementos de outra
temTodos([],_).
temTodos([H|T],L) :- member(H,L),temTodos(T,L).
```



```

243
244 %----- Query Extra -----%
245
246 %maisMonumentos(3,32,3).
247
248 maisMonumentos(Origem, Destino, NumeroMon) :-
249     findall((P,N), Dist), maisMon(Origem, Destino, P, Dist, NumeroMon, N), L),
250     imprime(L).
251
252 maisMon(Origem, Destino, [Origem|Percurso], Dist, NumeroMon, N) :-
253     maisMonAux(Origem, Destino, Percurso, Dist, NumeroMon, N, []).
254
255 maisMonAux(Destino, Destino, [], 0, _, _, _).
256 maisMonAux(Origem, Destino, [Proximo|Percurso], Dist, NumeroMon, N, Visitados) :-
257     Origem \= Destino,
258     getNumMon(Origem, Proximo, N, Dist1),
259     NumeroMon =< N,
260     \+member(Proximo, Visitados),
261     maisMonAux(Proximo, Destino, Percurso, Dist2, NumeroMon, N1, [Origem|Visitados]),
262     Dist is Dist1 + Dist2.
263
264 %Calcula o numero de Monumentos de uma cidade
265 getNumMon(Origem, Proximo, N, Dist) :-
266     getArco(Origem, Proximo, Dist),
267     cidade(Proximo, _, _, _, _, Monumento, _, 'admin'),
268     comprimento(Monumento, N).
269
270
271 %----- Informada -----%
272
273 %----- Query 1/6 -----%
274
275 usaAestrela(Origem, Destino, Percurso/Custo) :-
276     distance(Origem, Destino, X),
277     aestrela([[Origem, Destino]]/0/X, PercursoInv/Custo/_),
278     reverseL(PercursoInv, Percurso).
279
280 aestrela(Percursos, Percurso) :-
281     temMelhor(Percursos, Percurso),
282     Percurso = [(Cidade, Destino)|_]/_/_/,
283     Cidade == Destino.
284 aestrela(Percursos, Solucao) :-
285     temMelhor(Percursos, MelhorPercurso),
286     escolhe(MelhorPercurso, Percursos, OutrosC),
287     aestrelaExtendida(MelhorPercurso, PercursosExt),
288     append(OutrosC, PercursosExt, NovosC),
289     aestrela(NovosC, Solucao).
290
291 temMelhor([Percurso], Percurso) :- !.
292 temMelhor([Percurso1/Custo1/Estimativa1, _/Custo2/Estimativa2|Percursos], MelhorPercurso) :-
293     (Custo1 + Estimativa1) =< (Custo2 + Estimativa2), !,
294     temMelhor([Percurso1/Custo1/Estimativa1|Percursos], MelhorPercurso).
295 temMelhor([_|Percursos], MelhorPercurso) :-
296     temMelhor(Percursos, MelhorPercurso).
297

```

```
298 aEstrelaExtendida(Percurso,PercursosExt) :-
299     findall(NovosC, arcoAdj(Percurso,NovosC), PercursosExt).
300
301 arcoAdj([(Cidade,Destino)|Percurso]/Custo/_, [(Proximo,Destino),(Cidade,Destino)|Percurso
302     ]/NCusto/Estimativa) :-
303     getArco(Cidade,Proximo,ProximoCusto),
304     \+member(Proximo,Percurso),
305     distance(Proximo,Destino,Estimativa),
306     NCusto is Custo + ProximoCusto.
307
308 % Tira o elemento D de uma lista
309 escolhe(D, [D|Ds], Ds).
310 escolhe(D, [D|Ds], [D|Es]) :- escolhe(D,Ds,Es).
311
312 % reverte uma lista
313 reverseL(Ds,Es) :- reverseL(Ds, [], Es).
314 reverseL([],Ds,Ds).
315 reverseL([D|Ds],Es,Fs) :- reverseL(Ds, [D|Es], Fs).
316
317 getCoord(Id,Lat,Long) :- cidade(Id,_,Lat,Long,_,_,_,_).
318
319 distance(Origem,Destino,C) :-
320     getCoord(Origem,Lat0,Long0),
321     getCoord(Destino,LatD,LongD),
322     C is sqrt((LatD-Lat0)^2 + (LongD-Long0)^2).
```