Universidade do Minho Departamento de Informática

Mestrado Integrado em Engenharia Informática

Sistemas de Representação de Conhecimento e Raciocínio

Trabalho Individual - Época Recurso

A82238 João Gomes

Índice

Introdução

Este projeto teve por objetivo o desenvolvimento de uma base de conhecimento que permitisse a representação de diversas cidades de Portugal, que tinham a si atribuídas diversas características, como posição, a sua capital de distrito, se era uma cidade capital de distrito ou não.

Após o desenvolvimento desta base de conhecimento, tive também de criar um sistema que fizesse uso desta base de conhecimento e conseguisse criar percursos entre as cidades, tendo em conta restrições específicas, fazendo uso dos algoritmos de pesquisar estudados nas aulas.

Descrição do Trabalho e Análise de Resultados

Base de conhecimento

Os dados referentes às cidades foram dados num ficheiro de formato excel. Para realizar o parse deste ficheiro, fiz um pequeno programa em java responsável por fazer o parse da informação das cidades e construir um ficheiro com a informação formatada, de forma a poder ser importado para o Prolog. Não nos foi fornecida informação referente às ligações entre as cidades, pelo que ficou ao nosso critério o processo da sua criação.

Decidi criar as ligações das cidades em função da distância entre elas, criando um arco em duas cidades caso a distância entre elas fosse menor a um valor arbitrário escolhido. Este valor foi testado e alterado de forma a criar um número relevante de arcos mas não muito elevado. Este processo foi também feito em java.

Para que a base de conhecimentos fosse corretamente caracterizada, foram criados os seguintes predicados:

- cidade: # id, nome, latitude, longitude, distrito, importância, se tinha acesso a praia, se tinha acesso a monumentos históricos -> {V,F}
- arco: #idOrigem, #idDestino, distância -> {V,F}

As características das cidades relativamente à existência ou não de praia e monumentos foi adicionada para permitir ter mais casos de usos e demonstrar melhor alguns algoritmos. É de referir que estas duas características foram atribuídas aleatoriamente às cidades, podendo não corresponder à realidade.

```
cidade(43,'Aguiar da Beira',40.817264,-7.544307,'Guarda','minor','No','No').
cidade(44,'Idanha-a-Nova',39.923157,-7.240819,'Castelo Branco','minor','No','Yes').
cidade(45,'Ribeira de Pena',41.52147,-7.802372,'Vila Real','minor','No','No').
```

Figura 1- Base de conhecimento relativo às cidades

```
arco(3,248,0.10601444265759458).
arco(5,53,0.14819800810064926).
arco(5,127,0.18849776821225286).
```

Figura 2- Base de conhecimento relativo aos arcos

Algoritmos de Pesquisa

Para a realização dos objetivos propostos no enunciado, foi necessário usar alguns dos algoritmos de pesquisa abordados e realizados nas aulas, adaptando-os apenas para cumprir as diferentes restrições necessárias. As diferentes estratégias de pesquisa utilizadas são constituídas tanto por algoritmos de pesquisa não-informada como por algoritmos de pesquisa informada. Usando estes dois tipos de algoritmos permite no fim fazer algumas comparações relativas aos tempos de execução de cada um.

- Caminho

```
caminho(A,B,P) :- caminho1(A,[B],P).
caminho1(A,[A|P1],[A|P1]).
caminho1(A,[Y],P):- arco(X,Y,_), caminho1(A,[X,Y],P).
caminho1(A,[Y|P1],P) :-
     arco(X,Y,_), \+ memberchk(X,[Y|P1]), caminho1(A,[X,Y|P1],P).
```

Figura 3 – Algoritmo Caminho

- Profundidade Primeiro

```
%   PROFUNDIDADE PRIMEIRO

resolve_pp(Nodo, Destino, [Nodo|Caminho]) :-
    profundidadeprimeiro(Nodo, Destino, Caminho,8).

profundidadeprimeiro(Nodo, Destino,[],MAX) :- MAX > 0, Nodo == Destino, !.


profundidadeprimeiro(Nodo, Destino,[ProxNodo|Caminho],MAX) :- MAX > 0, DEC is MAX - 1, !,
    arco(Nodo, ProxNodo,_),
    profundidadeprimeiro(ProxNodo, Destino,Caminho,DEC).
```

Figura 4 - Algoritmo Profundidade Primeiro

- Estrela

```
%   ESTRELA

resolve_estrela(Nodo, Destino, Caminho/Custo) :-
        tamanho([Nodo,Destino],Estima),
        estrela([[Nodo]/0/Estima], InvCaminho/Custo/_, Destino,40),
        inverso(InvCaminho, Caminho),!.

estrela(Caminhos, Caminho, Destino,LIMIT) :- LIMIT>0,
        obtem_melhor(Caminhos, Caminho),
        Caminho = [Nodo|_]/_/_,
        Nodo==Destino.

estrela(Caminhos, SolucaoCaminho, Destino, LIMIT) :-
        LIMIT>0, DEC is LIMIT - 1, !,
        obtem_melhor(Caminhos, MelhorCaminho),
        seleciona(MelhorCaminho, Caminhos, OutrosCaminhos),
        expande_estrela(Destino,MelhorCaminho, ExpCaminhos),
        append(OutrosCaminhos, ExpCaminhos, NovoCaminhos),
        estrela(NovoCaminhos, SolucaoCaminho, Destino,DEC).


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_estrela(Destino, Caminho, ExpCaminhos) :- !,
        findall(NovoCaminho, adjacente(Destino,Caminho,NovoCaminho), ExpCaminhos).

adjacente(Destino,[Nodo|Caminho]/Custo/_, [ProxNodo,Nodo|Caminho]/NovoCusto/Est) :-
        arco(Nodo, ProxNodo,_),\+ member(ProxNodo, Caminho),
        tamanho([Nodo,ProxNodo],PassoCusto),
        NovoCusto is Custo + PassoCusto,
        tamanho([ProxNodo,Destino],Est).
```

Figura 5-Algoritmo Estrela

- Gulosa

```
%   GULOSA
resolve_gulosa(Nodo, Destino,Caminho/Custo) :-
    tamanho([Nodo,Destino],Estima),
    gulosa([[Nodo]/0/Estima], Destino, InvCaminho/Custo/_),
    inverso(InvCaminho, Caminho).

gulosa(Caminhos, Destino, Caminho) :-
    obtem_melhor_g(Caminhos, Caminho),
    Caminho = [Nodo|_]/_/_,
    Nodo == Destino.

gulosa(Caminhos, Destino,  SolucaoCaminho) :-
    obtem_melhor_g(Caminhos, MelhorCaminho),
    seleciona(MelhorCaminho, Caminhos, OutrosCaminhos),
    expande_gulosa(MelhorCaminho, Destino, ExpCaminhos),
    append(OutrosCaminhos, ExpCaminhos, NovoCaminhos),
        gulosa(NovoCaminhos, Destino, SolucaoCaminho).


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

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

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

expande_gulosa(Caminho, Destino, ExpCaminhos) :-
    findall(NovoCaminho, adjacente(Destino, Caminho,NovoCaminho), ExpCaminhos).
```

Figura 6 - Algoritmo Gulosa

A pesquisa não-informada encontra-se representada pelo algoritmo **Caminho** e **Profundidade Primeiro**. A pesquisa informada é representada pelos algoritmos **Estrela** e **Gulosa**.

Para todas as funcionalidades que pediam percursos baseados em certas características das cidades, foram criadas versões destes algoritmos de pesquisa que acomodavam a essas características.

É de referir também que por vezes as funções entram em ciclos infinitos, uma vez que os arcos criados funcionam em ambos os sentidos, podendo por vezes um algoritmo ficar preso sempre no mesmo arco, por exemplo 1 -> 5 -> 1 -> 5 -> 1 ...

Para resolver este problema, resolvi pôr um limite no número de iterações de um algoritmo para garantir que ele não entra nestes ciclos infinitos. Mas esta solução traz a desvantagem de por vezes limitar as soluções encontradas pelos algoritmos podendo até não permitir aos algoritmos encontrar uma solução.

Tendo isto, resolvi apenas deixar o limite de iterações no algoritmo de procura Esfera e Profundidade Primeiro.

Calcular um trajeto entre dois pontos

No enunciado do trabalho, um dos pontos pedidos é o cálculo do trajeto entre dois pontos.

Todos os algoritmos de procura são capazes de encontrar um percurso possível entre duas cidades, caso esse percurso exista. A principal diferença está nos tempos de execução que variam consoante o algoritmo e a distância do percurso (a nível de nodos a percorrer).

```
| ?- caminho(1,185,X).
X = [1,53,5,185] ? yes
yes
| ?- resolve_pp(1,185,X).
X = [1,53,1,53,1,53,5,185] ? yes
yes
| ?- resolve_estrela(1,185,X).
X = [1,157,185]/0.27642227905493677 ? yes
yes
| ?- resolve_gulosa(1,185,X).
X = [1,53,185]/0.2783506956883002 ? yes
yes
```

Figura 7- Percursos

Selecionar percurso apenas com cidades com certas características

Para esta funcionalidade, usei partido dos algoritmos de procura já definidos, mas desta vez recebiam alguns argumentos extras indicando as características que queriam. Usando os algoritmos de procura para gerar caminhos, depois usamos uma outra função **verifica** para certificar que as cidades do percurso correspondem às características pedidas.

```
147 % --------- PERCURSO POR CIDADES COM DETERMINADAS CARACTERISTICAS ---------
148
149 caminhoCaracteristicas(A,B,P,D,I,O,M) :- caminho(A,B,P), verifica(P,D,I,O,M).
150
151 profundidadeCaracteristicas(A,B,P,D,I,O,M) :- resolve_pp(A,B,P), verifica(P,D,I,O,M).
152
153 estrelaCaracteristicas(A,B,P,D,I,O,M) :- resolve_estrela(A,B,P/_), verifica(P,D,I,O,M).
154
155 gulosaMarMonu(A,B,P,D,I,O,M) :- resolve_gulosa(A,B,P/_), verifica(P,D,I,O,M).
156
```

Figura 8 – Selecciona características para as cidades

Excluir cidades com determinadas características

Esta funcionalidade foi realizada de forma muito semelhante à anterior. A única diferença sendo a função que verifica o percurso dado pelos algoritmos, verificando agora que nenhuma das cidades dos algoritmos contém as características passadas como parâmetros.

```
157  % ----------- PERCURSO QUE EXCLUI CIDADES COM CERTAS CARACTERISTICAS
158
159  caminhoExclui(A,B,P,D,I,O,M) :- caminho(A,B,P), exclui(P,D,I,O,M).
160
161  profundidadeExclui(A,B,P,D,I,O,M) :- resolve_pp(A,B,P), exclui(P,D,I,O,M).
162
163  estrelaExclui(A,B,P,D,I,O,M) :- resolve_estrela(A,B,P/_), exclui(P,D,I,O,M).
164
165  gulosaExclui(A,B,P,D,I,O,M) :- resolve_gulosa(A,B,P/_), exclui(P,D,I,O,M).
166
```

Figura 9- Exclui características das cidades

Cidade num percurso com o maior número de ligações

Para esta funcionalidade, comecei por usar a função findall para calcular o número de arcos que cada cidade tinha no percurso. Depois fui comparando as cidades duas a duas, deixando na lista a cidade que tivesse maior ligações entre as duas. Este processo foi repetido até a lista apenas conter uma cidade, que será a cidade com maior número de ligações.

```
191  % ---------------DETERMINAR CIDADE COM MAIOR NUMERO DE LIGAÇÕES-----------
192
193  maisLigacoes([X],X).
194  maisLigacoes([X,Z|T],R) :- findall(Y,arco(X,Y,_),L1), findall(J,arco(Z,J,_),L2),
195                             length(L1,N1), length(L2,N2), N1 > N2, maisLigacoes([X|T],R).
196  maisLigacoes([X,Z|T],R) :- findall(Y,arco(X,Y,_),L1), findall(J,arco(Z,J,_),L2),
197                             length(L1,N1), length(L2,N2), N1 =< N2, maisLigacoes([Z|T],R).
198
```

Figura 10 – Maior número de ligações.

Escolher o menor percurso (usando critério menor número de cidades)

Para esta funcionalidade fez-se uso dos algoritmos de procura em junção com a função **findall** para encontrar todos os percursos possíveis e a função **menor** que calcula qual desses percursos tem menos cidades, ou seja, qual dessas listas tem menos elementos.

```
% ------------MENOR PERCURSO -> MENOR NUMERO DE CIDADES----------

caminho_MenorNCidades(A,B,P) :- findall(T,caminho(A,B,T),L), menor(L,P).

profundidade_MenorNCidades(A,B,P) :- findall(T,resolve_pp(A,B,T),L), menor(L,P).

estrela_MenorNCidades(A,B,P) :- findall(T,resolve_estrela(A,B,T/_),L), menor(L,P).

gulosa_MenorNCidades(A,B,P) :- findall(T,resolve_gulosa(A,B,T/_),L), menor(L,P).
```

Figura 11– Menor número de cidades num percurso

Escolher o percurso mais rápido (usando critério da distância)

Nesta funcionalidade existe novamente uma versão para cada um dos algoritmos de pesquisa.

```
% ---------------PERCURSO MAIS RAPIDO -> DISTANCIA-----------------

caminho_MaisRapido(A,B,P) :- findall(T,caminho(A,B,T),L), distancia(L,P).

profundidade_MaisRapido(A,B,P) :- findall(T,resolve_pp(A,B,T),L), distancia(L,P).

estrela_MaisRapido(A,B,P) :- findall(T,resolve_estrela(A,B,T/_),L), distancia(L,P).

gulosa_MaisRapido(A,B,P) :- findall(T,resolve_gulosa(A,B,T/_),L), distancia(L,P).
```

Figura 12– Percurso mais rápido

Esta função faz uso da função **findall** para encontrar todos os caminhos e depois usa a função **distância** para calcular o percurso mais rápido desses. A função **distância** faz uso da função **tamanho** que calcula a distância total de um percurso, permitindo depois que a função **distância** vá filtrando a lista de percursos, deixando sempre o percurso menor, até só sobrar um percurso na lista.

```
%calcula o tamnho/distancia de uma lista/percurso
tamanho([X],0).
tamanho([X,Y|T],P) :- cidade(X,_,LAT1,LONG1,_,_,_,_), cidade(Y,_,LAT2,LONG2,_,_,_,_),
                          dist(LAT1,LONG1,LAT2,LONG2,D), tamanho([Y|T], P1), P = D + P1.

%fica com a lista de menor distancia
distancia([X],X).
distancia([X,Y|T],P) :- tamanho(X,D1), tamanho(Y,D2), D1 <  D2, distancia([X|T],P).
distancia([X,Y|T],P) :- tamanho(X,D1), tamanho(Y,D2), D1 >= D2,  distancia([Y|T],P).

%função responsável por calcular a distância entre duas cidades
dist(Lat1,Long1,Lat2,Long2,D):- D is (sqrt((Lat1-Lat2)^2 + (Long1-Long2)^2)).
```

Figura 13- Função auxiliar distancia e tamanho

Escolher o percurso que passe apenas por cidades minor

Esta função é apenas um caso particular das funções que permitem selecionar um percurso que apenas passe por cidades que sejam **'minor'**. O algoritmo caminho foi modificado de modo a só aceitar cidades que correspondam a esta especificidade, enquanto que os outros três algoritmos foram usados em junção com uma função auxiliar **verificaMinor** que verifica se o percurso calculado apenas usa cidades minor.

```
% ---------PERCURSO POR CIDADES APENAS MINOR-------------

caminhoMinor(A,B,P) :- caminho1Minor(A,[B],P).
caminho1Minor(A,[A|P1],[A|P1]).
caminho1Minor(A,[Y|P1],P) :- arco(X,Y,_), \+ memberchk(X,[Y|P1]), cidade(X,_,_,_,_,'minor',_,_),
                             caminho1Minor(A,[X,Y|P1],P).

profundidadeMinor(A,B,P) :- resolve_pp(A,B,P), verificaMinor(P).

estrelaMinor(A,B,P) :- resolve_estrela(A,B,P/_), verificaMinor(P).

gulosaMinor(A,B,P) :- resolve_gulosa(A,B,P/_), verificaMinor(P).
```

Figura 14 – Percurso por cidades minor

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

Para esta funcionalidade, além do ponto origem e ponto destino do percurso, é também passado uma lista de 1 ou mais cidades por onde o percurso tem obrigatoriamente de passar.

Isto é feito gerando um percurso com os algoritmos já referidos e fazer a verificação da solução usando a função **passaIntermedios** que verifica se as cidades intermedias fazem todas parte do percurso.

```
% --------------- PERCURSO QUE PASSA POR CIDADES OBRIGATORIAMENTE

caminho_Obrigatorio(A,B,P,C) :- caminho(A,B,P), intermedio(C,P).

profundidade_Obrigatorio(A,B,P,C) :- resolve_pp(A,B,P), intermedio(C,P).

estrela_Obrigatorio(A,B,P,C) :- resolve_estrela(A,B,P/_), intermedio(C,P).

gulosa_Obrigatorio(A,B,P,C) :- resolve_gulosa(A,B,P/_), intermedio(C,P).
```

Figura 15 – Cidades intermédias

```
%#     Funções auxiliares
%- - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
inverso(Xs, Ys):-
    inverso(Xs, [], Ys).

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

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

adicionar([],X,[X]).
adicionar([H|T],X,[H|T]) :- pertence(X, [H|T]).
adicionar([H|T],X, [X|[H|T]]) :- nao(pertence(X,[H|T])).

% calcula o menor elemento de uma lista
menor([X],X).
menor([X,Y|T], M) :- length(X,N1), length(Y,N2), N1<N2, menor([X|T],M).
menor([X,Y|T], M) :- length(X,N1), length(Y,N2), N1 >= N2,menor([Y|T],M).

%calcula o tamnho/distancia de uma lista/percurso
tamanho([X],0).
tamanho([X,Y|T],P) :- cidade(X,_,LAT1,LONG1,_,_,_,_), cidade(Y,_,LAT2,LONG2,_,_,_,_),
                        dist(LAT1,LONG1,LAT2,LONG2,D), tamanho([Y|T], P1), P = D + P1.

%fica com a lista de menor distancia
distancia([X],X).
distancia([X,Y|T],P) :- tamanho(X,D1), tamanho(Y,D2), D1 <  D2, distancia([X|T],P).
distancia([X,Y|T],P) :- tamanho(X,D1), tamanho(Y,D2), D1 >= D2,  distancia([Y|T],P).

%função responsável por calcular a distância entre duas cidades
dist(Lat1,Long1,Lat2,Long2,D):- D is (sqrt((Lat1-Lat2)^2 + (Long1-Long2)^2)).

% funcao que verifica se o percurso só passa por cidades minor
verificaMinor([]).
verificaMinor([X|T]) :- cidade(X,_,_,_,_,'minor',_,_), verificaMinor(T).

% funçao que verifica se o percurso só passa por cidades com mar
verificaMar([]).
verificaMar([X|T]) :- cidade(X,_,_,_,_,_,'Yes',_), verificaMar(T).

% funçao que verifica se o percurso só passa por cidades com monumentos
verificaMonu([]).
verificaMonu([X|T]) :- cidade(X,_,_,_,_,_,_,'Yes'), verificaMonu(T).

% funcao que verifica se o percurso passa por cidades com caracteristicas especificas
verifica([],D,I,O,M).
verifica([X|T],D,I,O,M) :- cidade(X,_,_,_,_,D,I,O,M), verifica(T,D,I,O,M).

% funcao que verifica se o percurso passa por cidades com caracteristicas a excluir
exclui([],D,I,O,M).
exclui([X|T],D,I,O,M) :- \+ cidade(X,_,_,_,_,D,I,O,M), exclui(T,D,I,O,M).

% funcao que verifica se o percurso passa pelas cidades obrigatorias
intermedio([],C).
intermedio([X|T],C) :- memberchk(X,C), intermedio(T,C).
```

Figura 16 – Funções auxiliares

Análise de tempos de execução e alguns exemplos de resultados de algumas funcionalidades

De seguida apresentam-se resultados de performance dos diferentes algoritmos em alguns casos, indicando apenas se o algoritmo encontra uma solução em tempo aceitável ou não.

	Percurso entre duas cidades	**Percurso só por cidades minor**	**Cidades Intermédias**	***Findall* Menor Percurso**	**Findall Percurso mais rápido**
Caminho	Sim	Sim	Sim	Não	Não
ProfundidadeP	Sim	Sim	Sim	Sim	Sim
Estrela	Sim	Sim	Sim	Sim	Sim
Gulosa	Sim	Sim	Não	Não	Não

```
| ?- estrelaMinor(2,173,X).
X = [2,67,173] ? yes
yes
| ?- gulosaMinor(2,173,X).
X = [2,67,173] ? yes
yes
| ?- profundidadeMinor(2,173,X).
X = [2,67,2,67,2,67,104,173] ? yes
yes
| ?- caminhoMinor(2,173,X).
X = [2,67,104,55,128,40,50,137,30,173] ? yes
yes
| ?-
```

Figura 17 – Resultados de Minor

```
| ?- caminho_Obrigatorio(2,173,X,[67,104,55]).
X = [2,67,104,144,3,55,128,40,50,137,30,173] ? yes
yes
| ?- profundidade_Obrigatorio(2,173,X,[67,104,55]).
X = [2,32,2,67,104,55,50,173] ? yes
yes
| ?- estrela_Obrigatorio(2,173,X,[67,104,55]).
no
| ?- gulosa_Obrigatorio(2,173,X,[67,104,55]).
```

Figura 18 – Resultado de Cidade Intermedias

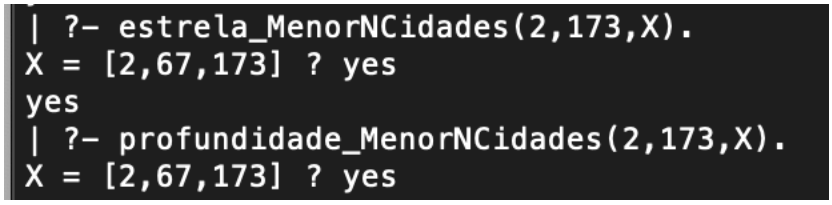

Figura 19 – Calcular menor percurso

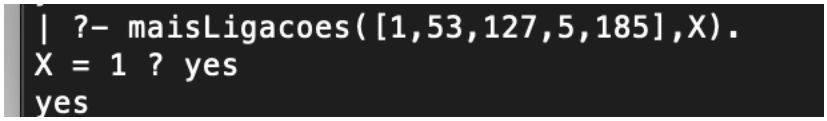

Figura 20 – Maior número de ligações

Conclusões

No final deste exercício, conclui que foi desenvolvido com sucesso um sistema de caraterização universo do sistema de cidades de Portugal.

A minha base de conhecimento podia ter sido melhor desenvolvida, criando arcos de forma aleatória para complementar os arcos que foram criados de forma determinística a partir da distância. É uma consideração talvez não permitir que os arcos tivessem 2 sentidos, para evitar a presença de ciclos infinitos em alguns algoritmos.

Também é de referir que uma das funções auxiliares, **verifica**, não está funcional por razões que eu não consigo compreender.

Na conclusão do projeto sou capaz de afirmar um aumento no conhecimento e compreensão da aplicação dos algoritmos de pesquisar à resolução de problemas. A mesma evolução no conhecimento se verificou relativamente à programação lógica estendida e à ferramenta Prolog.