Sistemas de Representação de Conhecimento e Raciocínio

MIEI - 3º ANO - 2º SEMESTRE
UNIVERSIDADE DO MINHO

TRABALHO PRÁTICO

Bernardo Viseu, A74618

Braga, 8 de Julho de 2020

Resumo

O presente documento diz respeito ao trabalho prático individual proposto na unidade curricular de Sistemas de Representação de Conhecimento e Raciocınio da Universidade do Minho.

Este trabalho tem o objectivo de motivar para a utilização da extensão à programação em lógica, usando a linguagem de programação em lógica PROLOG.

Este trabalho teve como tema a gestão de um conjunto de cidades e o cálculo do trajecto óptimo entre as mesmas.

Ao longo do presente relatório serão apresentadas as formas de conhecimento adoptadas e as caracterısticas e funcionalidades do sistema, assim como também serão descritas as estratégias utilizadas para a concretização das mesmas.

Conteúdo

1. Introdução

No âmbito da disciplina de Sistemas de Representação de Conhecimento e Raciocínio foi proposto a demonstração de funcionalidades subjacentes à programação em lógica estendida e à representação de conhecimento, num sistema com capacidade de calcular e filtrar trajectos entre várias cidades de Portugal, dado a sua localização e outras características.

Para isto, a cada Cidade está associado um Id, Nome, Latitude e Longitude, cidade primária a que está associada, e classificação.

Foi também necessário representar as ligações entre as cidades, que contém apenas o id de duas cidades que estejam ligadas, e a distancia entre elas, em Kms. O processo de gerar estas ligações é explicado na secção Parser.

Este trabalho tem como objectivo motivar para a utilização da extensão à programação em lógica, usando a linguagem de programação em lógica PROLOG.

2. Parser

Dado que a informação de Cidades fornecida está em formato *.xlsx*, foi necessário escrever um pequeno parser capaz de organizar tudo de maneira a ser compatível com o resto do código PROLOG.

Para isto, decidi fazer o parser na linguagem *Python*, devido a facilidade de escrita para um script pequeno como o que foi necessário.

Resumidamente, o que o parser faz é abrir o ficheiro *.xlsx*, pegar em todos os valores e organizá-los em *arrays* que vão depois ser utilizados para escrever num novo ficheiro de maneira a serem abertos com o programa PROLOG.

Foi utilizada uma pequena função auxiliar para substituir tirar os acentos das cidades como 'Évora', que estavam a causar um erro na compilação.

Neste parser também são geradas as ligações, para este projecto eu decidi considerar que existia uma ligação entre duas cidades caso estas se encontrem a menos de 15 kms. Para isto eu calculei a distancia entre todas as cidades e, num ficheiro *ligacoes.pl*, inseri as ligações encontradas.

```
arrayid = []
arraycity = []
arraylat = []
arraylng = []
arrayadmin = []
arraycapital = []
workbook = xlrd.open_workbook('cidades.xlsx')
for sheet in workbook.sheets():
    for column in range(sheet.nrows):
        arrayid.append(str(sheet.cell_value(column, 0)))
        arraycity.append(sheet.cell_value(column, 1))
        arraylat.append(sheet.cell_value(column, 2))
        arraylng.append(sheet.cell_value(column, 3))
        arrayadmin.append(sheet.cell_value(column, 4))
        arraycapital.append(sheet.cell_value(column, 5))
    arrayid.pop(0)
    arraycity.pop(0)
    arraylat.pop(0)
    arraylng.pop(0)
    arrayadmin.pop(0)
    arraycapital.pop(0)
```

Figura 2.1: Código Python para atribuir os valores aos arrays.

```
f = open('ligacoes.pl', 'w')
sys.stdout = f

arraylig = []
for x in range(281):
    for y in range(281):
        dist = math.sqrt((arraylat[x] - arraylat[y])**2 + (arraylng[x] - arraylng[y])**2) * 100
        if dist <= 15 and arraycity[x] != arraycity[y]:
            print(f'ligacao({str(arrayid[x]):s}, {str(arrayid[y]):s}, {str(dist):s}).')

sys.stdout = orig_stdout
f.close()
```

Figura 2.2: Código que calcula e gera as ligações.

3. Descrição do Trabalho e Análise de Resultados

3.1 Descrição do Sistema

Para o desenvolvimento deste projecto, foi considerado a seguinte representação:

- $Cidade : \#Id, Nome, Latitude, Longitude, CidadeAdmin, TipoAdmin$
- $Ligacao : \#Id1, Id2, Distancia$

```
:-dynamic cidade/6.
:-dynamic ligacao/3.
```

3.2 Predicados

3.2.1 Predicado Cidade/6

O predicado cidade tem como finalidade caracterizar uma cidade. Para isto é guardado na base de conhecimento através de um id, um nome, latitude, longitude, nome da cidade administradora e tipo de poder administrativo. Alguns exemplos de cidades:

```
cidade(1.0, 'Lisbon', 38.716667, -9.133333, 'Lisboa', 'primary').
cidade(2.0, 'Picotos', 41.192402, -8.619816, 'Porto', 'minor').
cidade(3.0, 'Braga', 41.550323, -8.420052, 'Braga', 'admin').
cidade(5.0, 'Setubal', 38.533333, -8.9, 'Setubal', 'admin').
cidade(6.0, 'Copeira', 40.176915, -8.424018, 'Coimbra', 'minor').
cidade(8.0, 'Portimao', 37.136636, -8.539796, 'Faro', 'minor').
cidade(9.0, 'Evora', 38.566667, -7.9, 'Evora', 'admin').
cidade(10.0, 'Aveiro', 40.644269, -8.645535, 'Aveiro', 'admin').
cidade(11.0, 'Leiria', 39.747724, -8.804995, 'Leiria', 'admin').
cidade(12.0, 'Faro', 37.019367, -7.932229, 'Faro', 'admin').
cidade(13.0, 'Beja', 38.015064, -7.863227, 'Beja', 'admin').
cidade(14.0, 'Braganca', 41.805817, -6.757189, 'Braganca', 'admin').
```

3.2.2 Predicado Ligação/3

O predicado *ligacao* tem como finalidade caracterizar uma ligação existente entre duas cidades. Para isto é guardado na base de conhecimento através de um id da primeira cidade, o id da segunda cidade e a distância entre as mesmas. Alguns exemplos de ligações:

```
ligacao(1.0, 57.0, 8.297300048811405).
ligacao(1.0, 86.0, 11.475376490991541).
ligacao(1.0, 157.0, 8.111042315263953).
ligacao(1.0, 193.0, 4.441890635754282).
ligacao(1.0, 221.0, 9.536565631819363).
ligacao(1.0, 278.0, 11.767027557543855).
ligacao(2.0, 32.0, 4.369602893856861).
ligacao(2.0, 67.0, 10.95769690263422).
ligacao(2.0, 115.0, 5.881778120602454).
ligacao(2.0, 123.0, 7.002640804724991).
ligacao(2.0, 136.0, 13.886794419159287).
ligacao(2.0, 152.0, 4.333707569737594).
ligacao(2.0, 213.0, 9.981300779457596).
ligacao(3.0, 145.0, 9.846682916089406).
```

4. Objectivos

Para a realização de este trabalho foi pedida a realização de objectivos e funcionalidades que o programa PROLOG consiga resolver:

4.1 Calcular o trajecto possível entre duas Cidades

Para implementar esta funcionalidade utilizei uma pesquisa entre as várias ligações de maneira a procurar um possível trajecto entre duas cidades, passando pelas várias ligações guardadas.

```
%--------------------------------Trajecto entre duas cidades:

trajecto(Inicio, Final, Path) :-
    trajecto(Inicio, Final, [], Path).

trajecto(Inicio, Inicio, _, [Inicio]).
trajecto(Inicio, Final, Visitados, [Inicio|Nodos]) :-
    \+ member(Inicio, Visitados),
    Inicio \== Final,
    ligacao(Inicio, Nodo, _),
    trajecto(Nodo, Final, [Inicio|Visitados], Nodos).
```

4.2 Seleccionar apenas Cidades com uma certa característica para um trajecto

Nesta segunda funcionalidade decidi implementar um filtro de maneira a filtrar cidades com um tipo de poder administrativo dado. Com isto, também aqui está realizada a funcionalidade de filtrar um trajecto com cidades apenas 'minor'.

```
%--------------------------------Filtro cidades com caracteristicas/ Filtro minor

filtra_cidades(Inicio, Final, Filter, Path) :-
    filtra_cidades(Inicio, Final, Filter, [], Path).

filtra_cidades(Inicio, Inicio,_ ,_ , [Inicio]).
filtra_cidades(Inicio, Final, Filter, Visitados, [Inicio|Nodos]) :-
    \+ member(Inicio, Visitados),
    Inicio \== Final,
    ligacao(Inicio, Nodo, _),
    cidade(Nodo,_,_,_,_,Filter),
    filtra_cidades(Nodo, Final, Filter, [Inicio|Visitados], Nodos).
```

4.3 Excluir uma característica de Cidades para um trajecto

Neste objectivo utilizei a negação para excluir cidades com certas características dadas.

```
%--------------------------------Exclui cidades

filtro_excluir(Inicio, Final,L, [Inicio|Path]) :-
    aux_excluir(Inicio, Final, L, Path).

aux_excluir(Final, Final, B, [Final|_]) :- findall((C),cidade(Dst,_,_,_,_,C),L),
                                           nao(conf_list(B,L)).

aux_excluir(Inicio, Final, B, [ProxNodo|Path]) :-
    ligacao(Inicio,ProxNodo,_),
    findall((C),cidade(Nodo,_,_,_,_,C),L),
    nao(conf_list(B,L)),
    aux_excluir(ProxNodo, Final, B, Path).
```

4.4 Percurso com menor numero de Cidades

Para este objectivo utilizei um sort para mostrar a menor lista do conjunto de trajectos entre duas cidades.

```
%--------------------------------Pesquisa com menor número de cidades percorridas

calc_length(P, [L,P]) :-
    length(P,L).

menos_cidades(A,B,S) :-
    findall(P, trajecto(A,B,P), Ps),
    maplist(calc_length, Ps, Ls),
    sort(Ls, [[_,S]|_]).
```

4.5 Percurso com menor distância

Aqui eu utilizei um calculo da distancia nos possíveis trajectos entre duas cidades, e depois escolhi o que tinha o valor menor.

```
%--------------------------------Menor distancia

path_min(A, B, Path, Length) :-
        travel(A, B, [A], Q, Length).

travel(A, B, P, [B|P], L) :-
        ligacao(A, B, L).
travel(A, B, Visited, Path, L) :-
        ligacao(A, C, D),
        C \== B,
        \+member(C, Visited),
        travel(C, B, [C|Visited], Path, L1),
        L is D+L1.

shortest(A, B, Path, Length) :-
    setof([P,L],path_min(A, B, P, L), Set),
    Set = [_|_],
    minimal(Set, [Path,Length]).

minimal([F|R], M) :-
    min(R,F,M).

% minimal path

min([],M,M).
min([[P,L]|R],[_,M],Min) :- L < M, !, min(R,[P,L],Min).
min([_|R],M,Min) :- min(R,M,Min).
```

5. Conclusão

De uma perspectiva geral, considero que a realização deste projecto foi relativamente sucedida, visto que existem ainda possíveis melhorias nalgumas funcionalidades.

A realização deste trabalho mostrou se ser muito esclarecedor quanto ao conteúdo e aprendizagem da cadeira, dado que para as várias funcionalidades e objectivos pedidos no enunciado foi necessário um bom entendimento de PROLOG.