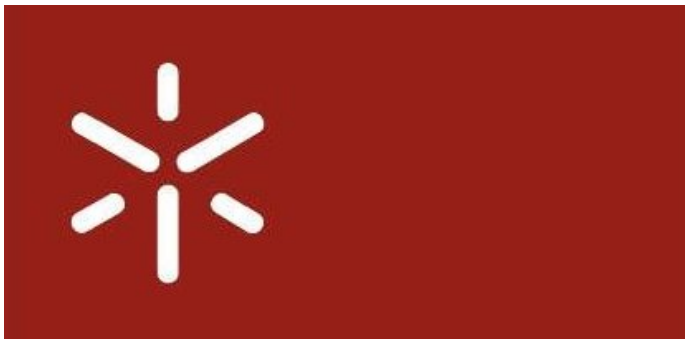

Métodos de Resolução de Problemas e de Procura

Sistemas de Representação de Conhecimento e Raciocínio

3º Ano de MIEInf

Universidade do Minho

Davide Matos A80970

15 de Julho de 2020

1 Introdução

Este relatório resulta da elaboração do exercício prático proposta para a época de recuso, com o tema de programação em lógica, proposto na unidade curricular de Sistemas de Representação de Conhecimento e Raciocínio. A elaboração deste exercício, permitiu que melhorasse as minhas capacidades e competências relativas à progração em lógica na linguagem PROLOG.

Neste exercício, foi proposto que fosse desenvolvido um Sistema de Representação de Conhecimento e Raciocínio capaz de caracterizar um universo de discurso na área de escolha de percusos em viagens por Portugal.

2 Geração da Base de Conhecimento

2.1 Perdicados

como solicitado no enunciado, era necessário fazer o *parsing* de um ficheiro com informações de localidades portuguesas e gerar um base de conhecimento com a qual seria possivel resolver os problemas propostos.

Como tal foi escrito um script usando a linguagem **Python** para gerar os perdicados em questão. Estes perdicados tratam-se de um perdicado **Cidade** que simplesmente lista a informação do ficheiro origem e um perdicado **arco** que representa um ligação entre duas localidades.

- **Cidade:** #idCid , Nome, Latitude, Longitude, Admin, Capital $\{V, F\}$
- **Arco:** #idCid1, #idCid2, Distancia $\{V, F\}$

O critério de geração de arcos entre cidades foi que cada capital de distrito *'admin'* tem um arco com todas as localidades que são seus *'minor'* e vice-versa. Para além destas conecções foram ainda criados arcos a conetar diretamente as capitais de distritos adjacentes.

Para além destes foram ainda criados 5 perdicados diferentes que definem um caracteristica para uma cidade.

- **hotel5Estrelas:** #idCid $\{V, F\}$
- **monumentos:** #idCid $\{V, F\}$
- **especialidadeGastronomica:** #idCid $\{V, F\}$
- **espacosVerdes:** #idCid $\{V, F\}$
- **atividadesNoturnas:** #idCid $\{V, F\}$

3 Algoritmos de Pesquisa

Como algoritmos de pesquisa foram apenas implementados algoritmos de pesquisa **não informada**.

4 Soluções Implementadas

4.1 query 1 : Escolher um qualquer percuso entre duas Localidades

método utilizado para a resolução desta query consistem em testar todas as possibilidades de percurso adicionando os nodos vistados a uma lista *Visitados* com o objectivo de evitar *loop infinitos*

```
percurso(Origem,Destino, [Origem|Caminho], Dist) :-
    percursoAux(Origem, Destino,Caminho, Dist,[]).

percursoAux(Destino,Destino, [], 0,_).
percursoAux(Origem,Destino ,[Prox|Caminho], Dist,Visitados) :-
    Origem \= Destino,
    adjacente(Origem, Prox, Dist1),
    nao(member(Prox,Visitados)),
    percursoAux(Prox, Destino, Caminho, Dist2,[Origem|Visitados]),
    Dist is Dist1 + Dist2.

adjacente(Origem, Prox, Dist) :-
    arco(Origem,Prox, Dist).

caminhos(Origem,Destino):-
    findall((S,Dist),percurso(Origem,Destino,S,Dist),L),
    escrever(L).
```

4.2 query 2 : Escolher um percuso entre duas Localidades que passe por cidades com uma determinada característica

Esta solução é baseada na solução desenvolvida para a query 1, no entanto é lhe adicinada um perdicado que verifica se a proxima localidade no perurso respeita a característica pertendida.

como consequência da forma como os perdicados das características foram criados, é necessario a criação de uma query para cada característica. Contudo é apresentado nesta secção do relatório apenas o exemplo para a característica *hotel5estrelas*:

```
percursocarHotel(Origem,Destino,[Origem|Percurso],Dist) :-
        percursocarHotelAux(Origem,Destino,Percurso,Dist,[]).

percursocarHotelAux(Dest, Dest, [], 0 , _ ).
percursocarHotelAux(Origem, Dest, [Prox|Caminho], Dist, Visitados) :-
    Origem \= Dest,
    adjacente(Origem, Prox, Dist1),
    nao(member(Prox,Visitados)),
    hotel5Estrelas(Prox),
    percursocarHotelAux(Prox, Dest, Caminho, Dist2, [Origem |
        Visitados]),
    Dist is Dist1 + Dist2.
```

4.3 query 3 : Escolher um percuso entre duas Localidades que passe por cidades sem um determinada característica.

esta solução é analoga à solução apresentada acima para a query 2.

```
percursoExccarAtiNoct(Origem,Destino,[Origem|Percurso],Dist) :-
        percursoExccarAtiNoctAux(Origem,Destino,Percurso,Dist,[]).

percursoExccarAtiNoctAux(Dest, Dest, [], 0 , _ ).
percursoExccarAtiNoctAux(Origem, Dest, [Prox|Caminho], Dist, Visitados
    ) :-
    Origem \= Dest,
    adjacente(Origem, Prox, Dist1),
    nao(member(Prox,Visitados)),
    nao(atividadesNoturnas(Prox)),
    percursoExccarAtiNoctAux(Prox, Dest, Caminho, Dist2, [Origem |
        Visitados]),
    Dist is Dist1 + Dist2.
```

A única diferença desta query para a query dois está no uso do perdicado **nao** quando se chama o perdicado correspondente à característica que se pertende evitar

4.4 query 4 : Identificar a Localidade com maior numero de ligaçoes dado um percurso.

A estratégia utilizada para esta query consiste em verificar a validade do percurso dado, seguindo de calcular o numero de conecções de cada Localidade do percurso e por fim calcular o minimo.

```
maiorNrConecoes([Origem|Caminho],L) :-
    maiorNrConecoesAux(Origem,Caminho,[],[]).

maiorNrConecoesAux(C,[],[],L):-
    nrConecoes(C,A),
    append([A],L,L1),
    max_in_list(L1,R),
    write(R).

maiorNrConecoesAux(Origem,[Prox|Caminho],Visitados,L):-
    adjacente(Origem,Prox,_),
    nao(member(Prox,Visitados)),
    nrConecoes(Origem,R),
    maiorNrConecoesAux(Prox,Caminho,[Origem|Visitados],L1),
    append([R],L1,L).

nrConecoes(Id,L) :-
        findall(X,adjacente(Id,_,_),Z),
        length(Z,L).

max_in_list([Max],Max).
max_in_list([H,K|T],M) :-
    H =< K,
    min_in_list([K|T],M).
max_in_list([H,K|T],M) :-
    H > K,
    min_in_list([H|T],M).
```

4.5 query 5 : Escolher o menor percurso (usando o critério do menor número de cidades percorridas

trata-se de uma adapação da query 1 na qual se adiciona um parametro para guardar o numero de arcos percoridos. Por fim escolhe-se o percurso no qual o numero de arcos percoridos é menor.

```
menorPercursoCidades(Origem,Destino):-
    findall((P,Dist,T),percurso2(Origem,Destino,Dist,P,T),L),
    min_in_list2(L,R),
    write(R).

percurso2(Origem,Destino,Dist,[Origem|Percurso],T) :-
        percursoAux2(Origem,Destino,Percurso,Dist,[],T).

percursoAux2(Destino,Destino,[],0,_,0).
percursoAux2(Origem,Destino,[Proximo|Percurso],Dist,Visitados,T) :-
        Origem \= Destino,
        arco(Origem,Proximo,Dist1),
        \+member(Proximo,Visitados),
        percursoAux2(Proximo,Destino,Percurso,Dist2,[Origem|Visitados
            ],T1),
    T is T1+1,
        Dist is Dist1 + Dist2.

min_in_list2([(P,D,Min)],(P,D,Min)).
min_in_list2([(A,D,H),(B,E,K)|T],(C,F,M)) :-
    H =< K,
    min_in_list2([(A,D,H)|T],(C,F,M)).

min_in_list2([(A,D,H),(B,E,K)|T],(C,F,M)) :-
    H > K,
    min_in_list2([(B,E,K)|T],(C,F,M)).
```

4.6 query 6 : Escolher o menor percurso (usando o critério de menor distância

É utilizada a query 1, apartir da qual se escolhe aquele percurso com menor distância.

```
maisPercursoDistancia(O,D) :- findall((P,Dist),caminhos(O,D,Dist,P),L)
    ,
                        min_in_list(L,R),
                    write(R).

min_in_list([(P,Min)],(P,Min)).
min_in_list([(A,H),(B,K)|T],(C,M)) :-
    H =< K,
    min_in_list([(A,H)|T],(C,M)).
min_in_list([(A,H),(B,K)|T],(C,M)) :-
    H > K,
    min_in_list([(B,K)|T],(C,M)).
```

4.7 query 7 : Escolher um percurso que passa apenas por cidade 'Admin'

Para esta query foi implementado um solução semelhante a query 1, no entanto é verificado se, no perdicado cidade, o proximo nodo é *admin*.

É importante também importante salientar que devido a forma como os arcos foram definidos, não existirá nenhum percurso que passe só por localidades **'minor'**. Por essa razão esta query foi implementada de forma um pouco diferente, ou seja, para as cidades **'admin'**

```
todosAdmin(Origem,Destino) :-
        findall((P,Dist),distrito(Origem,Destino,P,Dist,'admin'),L),
        escrever(L).

distrito(Origem,Destino,[Origem|Caminho],Dist,'admin') :-
        distritoAux(Origem,Destino,Caminho,Dist,'admin',[]).

distritoAux(Destino,Destino,[],0,_,_).
distritoAux(Origem,Destino,[Proximo|Caminho],Dist,'admin',Visitados)
    :-
        Origem \= Destino,
        adjacente(Origem,Proximo,Dist1),
        cidade(Proximo,_,_,_,_,'admin'),
        nao(member(Proximo,Visitados)),
        distritoAux(Proximo,Destino,Caminho,Dist2,'admin',[Origem|
            Visitados]),
        Dist is (Dist1 + Dist2).
```

4.8 query 8 : Escolher percurso com cidades intermediarias

Utiliza novamente a query 1 para escolhe apenas os percursos nos quais estão presentes todas as localidades intermediária passadas como argumento.

```
percursoPor(Origem,Destino,Intermedios) :-
        findall((P,Dist),percurso(Origem,Destino,P,Dist),L),
        testa(L,Intermedios,R),
    escrever(R).

testa([(X,D)|T],L,R) :-
testa(T,L,K),
temTodos(L,X), append([(X,D)],K,R).

temTodos([],_).
temTodos([H|T],L) :- member(H,L),temTodos(T,L).
```

5 Conclusão

Com a realização deste projeto aprofundei os meus conhecimentos na linguagem de programação em lógica **Prolog**. Apesar de não a versão final do trabalho não ter qualquer exemplo de pesquisa informada porque não conssegui implementar a tempo, Aprendi a diferenças entre pesquisa informada e não informada e as vantagens e desvantagens de cada uma

6 Anexos

Parse.py

```
import pandas as pd
import numpy as np
import math
import random


#ler dados
data = pd.read_excel(r'cidades.xlsx',encoding='utf-8')

data['city'] = data['city'].str.normalize('NFKD').str.encode('ascii',
    errors='ignore').str.decode('utf-8')
data['admin'] = data['admin'].str.normalize('NFKD').str.encode('ascii'
    , errors='ignore').str.decode('utf-8')
data['capital'] = data['capital'].str.normalize('NFKD').str.encode('
    ascii', errors='ignore').str.decode('utf-8')

lines = data.values


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']
}

#defenicao de algumas funcoes auxiliares

def cidadesDo_distrito(dist):
    ans = []
    for l in lines:
        if l[4] == dist:
            ans.append(l[0])
    return ans


def distancia(cidade1,cidade2):
    lat1=0
    long1=0
    lat2=0
    long2=0
    for l in lines:
        if l[0] == cidade1:
            lat1 = l[2]
            long1 = l[3]
        elif l[0] == cidade2:
            lat2 = l[2]
            long2 = l[3]
        else:
            continue
    ret = math.sqrt(((lat1-lat2)**2) + ((long1-long2)**2))
    return ret

def randomNCitys(N=100):
    cidIds = set()
    for _ in range(0,N):
        cId = random.randint(1,285)
        if cId in cidIds:
            continue
        else:
            cidIds.add(cId)
    return cidIds


def Idcidade(cidade):
    for l in lines:
        if cidade == l[1]:
            return l[0]
        else:
            continue

#criacao das cidades

cidfile = open('plsrc/cidades.pl','w+',encoding='utf-8')
cidfile.write(r'%cidade(ID,Nome,Latitide,Longitude,Admin,Capital)')
cidfile.write('\n')
```

```
for l in lines:
    s = "cidade({},'{}',{},{},'{}','{}').\n".format(l[0],l[1],l[2],l
        [3],l[4],l[5])
    cidfile.write(s)
cidfile.close()


#ciarcao das caracteristicas

carfile = open('plsrc/caracteristicas.pl','w+',encoding='utf-8')
carfile.write(r'%hotel5Estrelas(ID)')
carfile.write('\n')
carfile.write(r'%monumentos(ID)')
carfile.write('\n')
carfile.write(r'%especialidadeGastronomica(ID)')
carfile.write('\n')
carfile.write(r'%espacosVerdes(ID)')
carfile.write('\n')
carfile.write(r'%atividadesNoturnas(ID)')
carfile.write('\n')

carfile.write('\n\n\n')
carfile.write(r'%----------␣Cidades␣Com␣5␣Estrelas␣----------')
carfile.write('\n\n\n')

hoteis = set()

for k , v in dist_dict.items():
    s = 'hotel5Estrelas({}).\n'.format(Idcidade(k))
    hoteis.add(s)


cidComhotel=randomNCitys(100)
for i in cidComhotel:
    s = 'hotel5Estrelas({}).\n'.format(i)
    hoteis.add(s)

for s in hoteis:
    carfile.write(s)

carfile.write('\n\n\n')
carfile.write(r'%----------␣Cidades␣Com␣Monumentos␣----------')
carfile.write('\n\n\n')

monumentos = set()

for k , v in dist_dict.items():
    s = 'monumentos({}).\n'.format(Idcidade(k))
    monumentos.add(s)


cidComMonumento=randomNCitys(100)
for i in cidComMonumento:
```

```
    s = 'monumentos({}).\n'.format(i)
    monumentos.add(s)

for s in monumentos:
    carfile.write(s)

carfile.write('\n\n\n')
carfile.write(r'%----------␣Cidades␣Com␣Especialidades␣Gastronomicas␣
    ----------')
carfile.write('\n\n\n')

EspeGast = set()

for k , v in dist_dict.items():
    s = 'especialidadeGastronomica({}).\n'.format(Idcidade(k))
    EspeGast.add(s)

cidComEP=randomNCitys(100)
for i in cidComEP:
    s = 'especialidadeGastronomica({}).\n'.format(i)
    EspeGast.add(s)

for s in EspeGast:
    carfile.write(s)

carfile.write('\n\n\n')
carfile.write(r'%----------␣Cidades␣Com␣Espa os␣Verdes␣----------')
carfile.write('\n\n\n')

EspVerd = set()

for k , v in dist_dict.items():
    s = 'espacosVerdes({}).\n'.format(Idcidade(k))
    EspVerd.add(s)

cidComEV=randomNCitys(100)
for i in cidComEV:
    s = 'espacosVerdes({}).\n'.format(i)
    EspVerd.add(s)
for s in EspVerd:
    carfile.write(s)

carfile.write('\n\n\n')
carfile.write(r'%----------␣Cidades␣Com␣Atividades␣Noturnas␣
    ----------')
carfile.write('\n\n\n')

ActNot = set()

cidComAN=randomNCitys(100)
for i in cidComAN:
    s = 'atividadesNoturnas({}).\n'.format(i)
    ActNot.add(s)
```

```
for s in ActNot:
    carfile.write(s)

carfile.close()

#cria   o dos arcos

arcos = set()
acrfile = open('plsrc/grafo.pl','w+',encoding='utf-8')
acrfile.write(r'%arco(IDCidade1,IDCidade2,Distancia)')
acrfile.write('\n')
for k , v in dist_dict.items():
    cidadesk = cidadesDo_distrito(k)
    cap = Idcidade(k)
    for ck in cidadesk:
        if cap == ck:
            continue
        s = 'arco({},{},{}).\n'.format(ck,cap,distancia(ck,cap))
        a = 'arco({},{},{}).\n'.format(cap,ck,distancia(cap,ck))
        arcos.add(a)
        arcos.add(s)
    for ad in v:
        cap1 = Idcidade(ad)
        cap2 = Idcidade(k)
        s = 'arco({},{},{}).\n'.format(cap2,cap1,distancia(cap2,cap1))
        arcos.add(s)



for s in arcos:
    acrfile.write(s)


acrfile.close()
```

Main.pl

```
%-------------------------------------------------------------------------
% SIST. REPR. CONHECIMENTO E RACIOCINIO - MiEI
%
% Davide Matos - A80970

%-------------------------------------------------------------------------
% SICStus PROLOG: Declaracoes iniciais

:- set_prolog_flag( discontiguous_warnings,off ).
:- set_prolog_flag( single_var_warnings,off ).
:- include('cidades.pl').
:- include('grafo.pl').
:- include('caracteristicas.pl').


%-------------------------------------------------------------------------
% SICStus PROLOG: definicoes iniciais

:- op( 900,xfy,'::' ). % defini   o de um invariante



%-------------------------------------------------------------- Solucoes
% Termo, Predicado, Lista -> {V,F}
solucoes(T,Q,S) :- findall(T,Q,S).

%-------------------------------------------------------------
% Predicado nao
nao(Questao) :-
    Questao,!,fail.
nao(_).

%-------------------------------------------------------------------------
% Predicado comprimento
comprimento(S,N) :-
    length(S,N).

%-------------------------------------------------------------------------
% Predicado que da print
escrever([]).
escrever([X|T]) :-
    write(X),
    nl ,
    escrever(T).

%-------------------------------------------------------------------------
% Predicado que adiciona a uma lista
add(X, [], [X]).
add(X, [Y | T], [X,Y | T]) :- X @< Y, !.
add(X, [Y | T1], [Y | T2]) :- add(X, T1, T2).
```

```
%-------------------------------------------------------------------------%
%                          N O  INFORMADA                                 %
%-------------------------------------------------------------------------%

%-------------------------------DEPTH FIRST-------------------------------

%-------------------------------QUERIE 1----------------------------------
% escolher um qualquer percuso entre duas cidades

percurso(Origem,Destino, [Origem|Caminho], Dist) :-
                                                    % fun ao principal
    percursoAux(Origem, Destino,Caminho, Dist,[]).
                                                    % Lista de
        Visitados inicialmente Vazio

percursoAux(Destino,Destino, [], 0,_).
                                                    % caso de paragem
percursoAux(Origem,Destino  ,[Prox|Caminho], Dist,Visitados) :-
    Origem \= Destino,

        % se Origem diferente de Destino
    adjacente(Origem, Prox, Dist1),
                                                                    %
        procura o arco adjacente
    nao(member(Prox,Visitados)),
                                                                %
        verifica que o Proximo nao est  na lista de Visitados
    percursoAux(Prox, Destino, Caminho, Dist2,[Origem|Visitados]),
    Dist is Dist1 + Dist2.

        % somar as distancias

adjacente(Origem, Prox, Dist) :-
                                                                    %
    vai buscar o arco
    arco(Origem,Prox, Dist).

                % Procura o arco Origem,Destino


caminhos(Origem,Destino):-
    findall((S,Dist),percurso(Origem,Destino,S,Dist),L),
                                % imprime todos
    escrever(L).

% Exemplo caminhos(21,23).

%-------------------------------------QUERIE
    2-----------------------------------------
% Selecionar apenas cidades com uma determinhada caracteristica para
    um determinado trajeto.
```

```
%primeira query para a caracteristica hotel5estrelas.

percursocarHotel(Origem,Destino,[Origem|Percurso],Dist) :-
        percursocarHotelAux(Origem,Destino,Percurso,Dist,[]).

percursocarHotelAux(Dest, Dest, [], 0, _).
percursocarHotelAux(Origem, Dest, [Prox|Caminho], Dist, Visitados) :-
    Origem \= Dest,

        % se Origem diferente de Destino
    adjacente(Origem, Prox, Dist1),
                                                                %
        Procura arco adjacente
    nao(member(Prox,Visitados)),
                                                                    %
        Verifica que o proximo nao esta na lista de Visitados
    hotel5Estrelas(Prox),

        % Verificar a proxima cidade tem hotel 5 estrelas
    percursocarHotelAux(Prox, Dest, Caminho, Dist2, [Origem |
        Visitados]),
    Dist is Dist1 + Dist2.

        % somar as distancias



todosHotel(Origem,Destino):-
    findall((S,Dist,Car),percursocarHotel(Origem,Destino,S,Dist),L),
    escrever(L).        % print listas

% Exemplo todoshotel(2,32).

%query para a caracteristica monumentos.

percursocarMonumento(Origem,Destino,[Origem|Percurso],Dist) :-
        percursocarMonumentoAux(Origem,Destino,Percurso,Dist,[]).

percursocarMonumentoAux(Dest, Dest, [], 0, _).
percursocarMonumentoAux(Origem, Dest, [Prox|Caminho], Dist, Visitados)
    :-
    Origem \= Dest,

        % se Origem diferente de Destino
    adjacente(Origem, Prox, Dist1),
                                                                %
        Procura arco adjacente
    nao(member(Prox,Visitados)),
                                                                    %
        Verifica que o proximo nao esta na lista de Visitados
    monumentos(Prox),
```

```
        % Verificar a proxima cidade tem monumentos
    percursocarMonumentoAux(Prox, Dest, Caminho, Dist2, [Origem |
        Visitados]),
    Dist is Dist1 + Dist2.

        % somar as distancias



todosMonumento(Origem,Destino):-
    findall((S,Dist,Car),percursocarMonumento(Origem,Destino,S,Dist),L
        ),
    escrever(L).        % print listas

% Exemplo todosMonumento(2,32).

%query para a caracteristica especialidades gastronimocas.

percursocarEspGast(Origem,Destino,[Origem|Percurso],Dist) :-
        percursocarEspGastAux(Origem,Destino,Percurso,Dist,[]).

percursocarEspGastAux(Dest, Dest, [], 0, _).
percursocarEspGastAux(Origem, Dest, [Prox|Caminho], Dist, Visitados)
    :-
    Origem \= Dest,

        % se Origem diferente de Destino
    adjacente(Origem, Prox, Dist1),
                                                        %
        Procura arco adjacente
    nao(member(Prox,Visitados)),
                                                            %
        Verifica que o proximo nao esta na lista de Visitados
    especialidadeGastronomica(Prox),
                                                        %
        Verificar a proxima cidade tem especialidades gastronomicas
    percursocarEspGastAux(Prox, Dest, Caminho, Dist2, [Origem |
        Visitados]),
    Dist is Dist1 + Dist2.

        % somar as distancias



todosEspGast(Origem,Destino):-
    findall((S,Dist,Car),percursocarEspGast(Origem,Destino,S,Dist),L),
    escrever(L).        % print listas

% Exemplo todosEspGast(2,32).

%query para a caracteristica espa os verdes.
```

```
percursocarEspVerde(Origem,Destino,[Origem|Percurso],Dist) :-
        percursocarEspVerdeAux(Origem,Destino,Percurso,Dist,[]).

percursocarEspVerdeAux(Dest, Dest, [], 0, _).
percursocarEspVerdeAux(Origem, Dest, [Prox|Caminho], Dist, Visitados)
    :-
    Origem \= Dest,

       % se Origem diferente de Destino
    adjacente(Origem, Prox, Dist1),
                                                              %
       Procura arco adjacente
    nao(member(Prox,Visitados)),
                                                                  %
       Verifica que o proximo nao esta na lista de Visitados
    espacosVerdes(Prox),
                                                              %
       Verificar a proxima cidade tem espa os verdes
    percursocarEspVerdeAux(Prox, Dest, Caminho, Dist2, [Origem |
       Visitados]),
    Dist is Dist1 + Dist2.

       % somar as distancias



todosEspVerde(Origem,Destino):-
    findall((S,Dist,Car),percursocarEspVerde(Origem,Destino,S,Dist),L)
       ,
    escrever(L).       % print listas

% Exemplo todosEspVerde(2,32).

%query para a caracteristica actividades noturnas.

percursocarAtiNoct(Origem,Destino,[Origem|Percurso],Dist) :-
        percursocarAtiNoctAux(Origem,Destino,Percurso,Dist,[]).

percursocarAtiNoctAux(Dest, Dest, [], 0, _).
percursocarAtiNoctAux(Origem, Dest, [Prox|Caminho], Dist, Visitados)
    :-
    Origem \= Dest,

       % se Origem diferente de Destino
    adjacente(Origem, Prox, Dist1),
                                                              %
       Procura arco adjacente
    nao(member(Prox,Visitados)),
                                                                  %
       Verifica que o proximo nao esta na lista de Visitados
    atividadesNoturnas(Prox),
                                                              %
       Verificar a proxima cidade tem atividades noturnas
```

```
    percursocarAtiNoctAux(Prox, Dest, Caminho, Dist2, [Origem |
        Visitados]),
    Dist is Dist1 + Dist2.

        % somar as distancias



todosAtiNoct(Origem,Destino):-
    findall((S,Dist,Car),percursocarAtiNoct(Origem,Destino,S,Dist),L),
    escrever(L).        % print listas

% Exemplo todosAtiNoct(2,32).


%-------------------------------------QUERIE
    3-------------------------------------
% Escolher um trajeto que passa por cidades que excluem um determinada
     caracteristica.


percursoExccarAtiNoct(Origem,Destino,[Origem|Percurso],Dist) :-
        percursoExccarAtiNoctAux(Origem,Destino,Percurso,Dist,[]).

percursoExccarAtiNoctAux(Dest, Dest, [], 0, _).
percursoExccarAtiNoctAux(Origem, Dest, [Prox|Caminho], Dist, Visitados
    ) :-
    Origem \= Dest,

        % se Origem diferente de Destino
    adjacente(Origem, Prox, Dist1),
                                                                %
        Procura arco adjacente
    nao(member(Prox,Visitados)),
                                                                    %
        Verifica que o proximo nao esta na lista de Visitados
    nao(atividadesNoturnas(Prox)),
                                                                %
        Verificar a proxima cidade nao tem atividades noturnas
    percursoExccarAtiNoctAux(Prox, Dest, Caminho, Dist2, [Origem |
        Visitados]),
    Dist is Dist1 + Dist2.

        % somar as distancias



todosExcAtiNoct(Origem,Destino):-
    findall((S,Dist,Car),percursoExccarAtiNoct(Origem,Destino,S,Dist),
        L),
    escrever(L).        % print listas
```

```
%-------------------------------------QUERIE 4-------------------------------------
% identificar a cidade dum trajeto com o maior numerode ligaoes.

maiorNrConecoes([Origem|Caminho],L) :-
    maiorNrConecoesAux(Origem,Caminho,[],[]).


maiorNrConecoesAux(C,[],[],L):-
    nrConecoes(C,A),
    append([A],L,L1),
    max_in_list(L1,R),
    write(R).


maiorNrConecoesAux(Origem,[Prox|Caminho],Visitados,L):-
    adjacente(Origem,Prox,_),
    nao(member(Prox,Visitados)),
    nrConecoes(Origem,R),
    maiorNrConecoesAux(Prox,Caminho,[Origem|Visitados],L1),
    append([R],L1,L).



nrConecoes(Id,L) :-
        findall(X,adjacente(Id,_,_),Z),
        length(Z,L).



max_in_list([Max],Max).

max_in_list([H,K|T],M) :-
    H =< K,
    min_in_list([K|T],M).

max_in_list([H,K|T],M) :-
    H > K,
    min_in_list([H|T],M).


min_in_list([(P,Min)],(P,Min)).                    % We've found the
    minimum

min_in_list([(A,H),(B,K)|T],(C,M)) :-
    H =< K,                                  % H is less than or equal to K
    min_in_list([(A,H)|T],(C,M)).                % so use H

min_in_list([(A,H),(B,K)|T],(C,M)) :-
    H > K,                                   % H is greater than K
```

```
    min_in_list([(B,K)|T],(C,M)).                    % so use K

%-------------------------------------QUERIE 5-------------------------------------
% Escolher o menor percurso (crit rio de menor numero de cidades).


menorPercursoCidades(Origem,Destino):-
    findall((P,Dist,T),percurso2(Origem,Destino,Dist,P,T),L),
    min_in_list2(L,R),
    write(R).


percurso2(Origem,Destino,Dist,[Origem|Percurso],T) :-
        percursoAux2(Origem,Destino,Percurso,Dist,[],T).

percursoAux2(Destino,Destino,[],0,_,0).
percursoAux2(Origem,Destino,[Proximo|Percurso],Dist,Visitados,T) :-
        Origem \= Destino,
        arco(Origem,Proximo,Dist1),
        \+member(Proximo,Visitados),
        percursoAux2(Proximo,Destino,Percurso,Dist2,[Origem|Visitados
            ],T1),
    T is T1+1,
        Dist is Dist1 + Dist2.

min_in_list2([(P,D,Min)],(P,D,Min)).

min_in_list2([(A,D,H),(B,E,K)|T],(C,F,M)) :-
    H =< K,
    min_in_list2([(A,D,H)|T],(C,F,M)).

min_in_list2([(A,D,H),(B,E,K)|T],(C,F,M)) :-
    H > K,
    min_in_list2([(B,E,K)|T],(C,F,M)).

%exemplo menorPercursoCidades(3,10).

%-------------------------------------QUERIE 6-------------------------------------
% Escolher o menor percurso (crit rio da distancia).

maisPercursoDistancia(O,D) :- findall((P,Dist),caminhos(O,D,Dist,P),L)
    ,
                        min_in_list(L,R),
                    write(R).


min_in_list([(P,Min)],(P,Min)).

min_in_list([(A,H),(B,K)|T],(C,M)) :-
    H =< K,
    min_in_list([(A,H)|T],(C,M)).
```

```
min_in_list([(A,H),(B,K)|T],(C,M)) :-
    H > K,
    min_in_list([(B,K)|T],(C,M)).


%-----------------------------------------QUERIE
    7-----------------------------------------
% Percurso que passa so por cidades admin.

todosAdmin(Origem,Destino) :-
        findall((P,Dist),municipio(Origem,Destino,P,Dist,'admin'),L),
        escrever(L).

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

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



%------------------- Query 8 ----------------------------------------%
% Percurso com cidades intermediarias.



percursoPor(Origem,Destino,Intermedios) :-
        findall((P,Dist),percurso(Origem,Destino,P,Dist),L),
        testa(L,Intermedios,R),
    escrever(R).

%Filtra os Percursos que cumprem com o itenerario

testa([(X,D)|T],L,R) :-
testa(T,L,K),
temTodos(L,X), append([(X,D)],K,R).

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

%exemplo percursoPor(2,10,[32]).
```