Universidade do Minho
Escola de Engenharia

Sistemas de Representação Conhecimento e Raciocínio

Carlos Miguel Luzia Carvalho A89605

7 de junho de 2021

Índice

1 Resumo

Este relatório tem como objetivo documentar o trabalho individual proposto na Unidade Curricular de Sistemas Representação de Conhecimento e Raciocínio usando a linguagem PROLOG. Numa fase inicial é apresentada uma introdução do projeto. Posteriormente, é explicado o raciocínio que usei para o tratamento do dataset e obtenção do conhecimento e também o raciocínio para os predicados de travessia. Por fim, é feita uma apreciação crítica do trabalho.

2 Introdução

Este relatório tem como objetivo documentar o trabalho individual proposto na Unidade Curricular de Sistemas Representação de Conhecimento e Raciocínio usando a linguagem PROLOG. O objetivo pretendido é desenvolver um sistema de representação de conhecimento e raciocínio capaz de caracterizar um universo de discurso na área dos transportes de lixos, sendo este caracterizado como um grafo.

O sistema desenvolvido deverá ser capaz definir vários caminhos, partindo de um Ponto Inicial (Garagem), ligando este a vários outros pontos efetuando a recolha do lixo, esta travessia é finalizada num ponto final/Destino, escolhido por mim.

É de notar que optei pela versão simplificada do projeto, ignorando assim a capacidade do Camião.

3 Preliminares

O sistema que foi desenvolvido ao longo deste trabalho pretende caracterizar o universo de discurso na área da recolha de lixo da freguesia da Misericórdia.

Para que este projeto pudesse ser iniciado, foi necessária a obtenção de conhecimento base na área. Para tal, este foi adquirido através das aulas frequentadas e resolução das fichas práticas dadas nas mesmas, permitindo um maior contacto com a linguagem utilizada. À parte disto a base de conhecimento usada na resolução do projeto foi fornecida pelos docentes num dataset.xlsx.

Neste projeto pretende-se o desenvolvimento de um sistema que permita importar os dados relativos aos diferentes circuitos e representá-los numa base de conhecimento adequada. Posteriormente, é suposto o desenvolvimento de um sistema de recomendação de circuitos de recolha para o caso de estudo.

Posto isto, o caso prático deverá permitir, tendo em conta a localização de um ponto de partida (uma garagem):

- Gerar circuitos de recolha tanta seletivo como indiferenciada.
- Identificar quais os circuitos com mais pontos de recolha.
- Encontrar as melhores soluções de percurso.
- Comparar circuitos de recolha tendo em conta os indicadores de produtividade

É de notar que ao longo do trabalho procurei responder a estes requisitos porém inseri algumas dessas respostas à medida que demonstro os algoritmos de procura.

4 Descrição do Trabalho e Análise de Requisitos

4.1 Base de conhecimento e Decisões Tomadas

Dado o dataset fornecido pelos docentes, tive que tomar algumas decisões críticas para conseguir avançar na resolução do problema, assim sendo comecei por remover manualmente algumas colunas que considerei menos importantes, sendo estas por exemplo a freguesia, o tipo de contentor, entre outras. Ficando assim apenas com Latitude, Longitude, Morada, Tipo de lixo e quantidade/litro por caixote, depois desta remoção de colunas, numa prespetiva de diminuir os nodos e os arcos que iria ter de percorrer, considerei o percurso entre ruas Bidirecional e agrupei manualmente todas as moradas iguais com o mesmo tipo de lixo (ex. todas as Ruas Alecrim com o mesmo tipo de lixo), agrupando estes pontos, tive também que lhes agrupar os destinos de cada um destes e ainda as capacidades de todos os caixotes, com o mesmo tipo de lixo, da mesma rua.

Com todas estas alterações e agregações consegui reduzir o tamanho do dataset mas mantendo a informação nele contida, ficando assim com 136 pontos de recolha, organizados da seguinte forma.

pontos_recolha(latitude, longitude, local, tipo, quantidade, [listaDeDestinos]).

```
1    pontos_recolha(-9.14330880914792, 38.7080787857025, 'R do
     Alecrim', 'Lixos', 5940, ['R Ferragial','R Ataide','Pc Duque da
     Terceira']).
2    pontos_recolha(-9.14330880914792, 38.7080787857025, 'R do
     Alecrim', 'Papel e Cartao', 2370, ['R Ferragial','R Ataide','Pc
     Duque da Terceira']).
3    pontos_recolha(-9.14255098678099, 38.7073286838222, 'R Corpo
     Santo', 'Lixos', 2480, ['Lg Corpo Santo','Tv Corpo Santo']).
4
```

Para definir os pontos desta forma e conseguir criar arcos a partir destes pontos foi necessário o uso de um parser. Neste parser criei o ficheiro pontos.pl e arcos_final.pl onde estão definidos os pontos_recolha e os arcos, sendo os arcos definidos da seguinte forma:

arco(RUA1, RUA2, DISTANCIA).

```
1    arco('R do Alecrim', 'R Ferragial', 2.686029460947423).
2    arco('R do Alecrim', 'Pc Duque da Terceira', 9.31966255819804).
3    arco('R Corpo Santo', 'Lg Corpo Santo', 4.256597837516517).
```

```
4     arco('R Corpo Santo', 'Tv Corpo Santo', 2.031563467876351).
5     arco('Tv Corpo Santo', 'R Bernardino da Costa',
      2.340385361805613).
6
```

4.2 Parser

No parser comecei por fazer a leitura sem acentos do dataset fornecido que alterei como referido acima, e passei essa informação no formato referido acima para o ficheiro pontos.pl, simultaneamente a quando faço essa escrita no ficheiro divido todos os pontos de recolha em 5 dicionários python, referentes ao tipo de lixo a que cada ponto de recolha tem, e ainda os coloco a todos num outro dicionário Universal que os vai possuir a todos. Depois do ficheiro pontos.pl completo procedo a adicionar-lhe dois pontos de recolha em Bqr dos Ferreiros e AV Dom Carlos I, decidi adicionar estes 2 pontos de recolha uma vez que são destinos de vários outros pontos existentes, porém eles próprios não tem lixo algum, são apenas pontos de passagem, porém como considerei os caminhos bidirecionais faz sentido considerá-los como ponto de recolha. De seguida, começo por criar o ficheiro arcos.pl onde vou avaliar para cada ponto se os destinos que ele possui são também pontos de recolha, caso afirmativo, calculo a distância geográfica entre o ponto e os seus destinos e adiciono-o ao fileArcos no formato referido acima. Tendo já o fileArcos completo e sendo as minhas ligações entre pontos bidirecionais reparei que no ficheiro arcos.pl existiam distâncias duplicadas as quais equivaliam a distancias entre um ponto e o seu destino e o destino e o ponto, assim achei lógico retirar estes pontos duplicados. Para isso, criei um novo ficheiro arcos_final.pl onde à medida que ia percorrendo o ficheiro arcos.pl e encontro dois arcos com uma distância igual não escrevo no ficheiro arcos_final.pl a segunda aparição do arco que contivesse uma distância já existente. Com este parser fiquei com dois ficheiros pontos.pl e arcos_final.pl contendo o conhecimento do meu trabalho, estando este pronto a ser interpretado pelo Prolog.

5 Formulação do Problema e Decisões de Procura

Este projeto consiste num problema de pesquisa baseado num trajeto de recolha de Lixo em Lisboa no bairro da Misericórdia, tendo já a informação dividida e organizada explicada nos pontos acima, é, no entanto, necessário tomar mais algumas decisões. Apesar de existirem diversas formas de escolher um Ponto Inicial e Final desde adicionar um ponto garagem ao dataset,escolher ruas sem pontos de recolha , entre outros, eu decidi atribuir como ponto inicial e ponto final, pontos de recolha existentes, alterando estes algumas vezes para questões de testes.

Além deste tipo de decisões vi-me obrigado a optar pela simplificação do trabalho, ignorando assim a capacidade do Camião, não sendo assim necessário o armazenamento de informação respetiva ao estado inicial, final, representativo de uma transação no circuito.

6 Predicados Auxiliares e Factos

Nesta secção vou demonstrar alguns Predicados auxiliares e ou adicionais que necessitei de utilizar para conseguir criar os Predicados de Essencias/Travessia.

```
%verifica se    membro da lista
membro(X, [X|_]).
membro(X, [_|Xs]):-
  membro(X, Xs).


%Extensao do meta-predicado nao
nao( Questao ) :- Questao, !, fail.
nao( Questao ).

%Calcula o comprimento de uma lista
comprimento(S,N) :- length(S,N).

%Predicado que da um print no terminal
imprime([]).
imprime([X|T]) :- write(X), nl, imprime(T).

%verifica se um elemento pertence a uma  lista
temElem([],_).
temElem(L,[H|T]):- membro(H,L).
temElem(L,[H|T]):- temElem(L,T);memberchk(H,L).

%Obtem um arco apartir de um id
adjacente(Origem,Destino,Dist) :- arco(Origem,Destino,Dist).
adjacente(Origem,Destino,Dist) :- arco(Destino,Origem,Dist).

%verifica se uma lista esta vazia
estaVazia(L,V) :- comprimento(L,V),nao(V>0).

%encontra o menor elemento
minimo(L, (A,B)) :-
    seleciona((A,B), L, R),
    \+ ( membro((A1,B1), R), B1 < B ).

%encontra o maior elemento
maximo(L, (A,B)) :-
    seleciona((A,B), L, R),
    \+ ( membro((A1,B1), R), B1 > B ).

%reverte uma lista
reverseL(Ds,Es) :- reverseL(Ds, [], Es).
reverseL([],Ds,Ds).
reverseL([D|Ds],Es,Fs) :- reverseL(Ds, [D|Es], Fs).
```

```
44
45 %verifica qual o tipo de lixo de um ponto recolha
46 getLixo(Local, Tipo) :- pontos_recolha(_,_,Local,Tipo,_, _).
47
48 %Estima o custo at  ao nodo final
49 estimativa(Nodo,Est):-
50     distance(Nodo, Est).
51
52 %Calcula a distancia at  ao nodo final
53 distance(Origem,Dis):-
54     pontos_recolha(Lat1,Lon1,Origem,_,_,_),
55     final(Destino),
56     pontos_recolha(Lat2,Lon2,Destino,_,_,_),
57     P is 0.017453292519943295,
58     A is (0.5 - cos((Lat2 - Lat1) * P) / 2 + cos(Lat1 * P) * cos(
    Lat2 * P) * (1 - cos((Lon2 - Lon1) * P)) / 2),
59     Dis is (12742 * asin(sqrt(A))).
60
61
62 seleciona(E, [E|Xs], Xs).
63 seleciona(E, [X|Xs], [X|Ys]) :- seleciona(E, Xs, Ys).
```

A parte isto decidi considerar como meu ponto inicial e Ponto final um ponto de recolha do dataset, assim defini como facto um inicial e um final.

```
1 inicial('R do Alecrim').
2 final('Av 24 de Julho').
```

7 Predicados Essenciais

7.1 Procura não Informada

7.1.1 Depth First

A Depth First é um algoritmo de travessia de grafos muitas vezes utilizado em Prolog pelo facto de utilizar pouca memória, esta é utilizada para problemas com muitas soluções, embora porém não possa ser usada para árvores com profundidade infinita, para evitar que esta procura falhe é muitas vezes utilizado o algoritmo Depth First com Profundidade limitada.

Como estratégia este algoritmo procura expandir sempre um dos nós mais profundos da árvore.

Nesta secção vou apresentar as 3 procuras em profundidade, para o numero de arcos para o custo e por tipo de resíduo.

- Depth First menor número de Arcos

  Predicado que efetua uma pesquisa Depth First (resolveDP) que devolve a lista dos arcos que precorre.

```
1  resolveDP([Nodo|Caminho]):-
2      inicial(Nodo),
3      primeiroprofundidade(Nodo,[Nodo],Caminho).
4
5  primeiroprofundidade(Nodo,_, []):-
6      final(Nodo).
7
8  primeiroprofundidade(Nodo, Historico, [NodoProx|Caminho]):-
9      adjacente(Nodo, NodoProx,_),
10     nao(membro(NodoProx, Historico)),
11     primeiroprofundidade(NodoProx, [NodoProx|Historico], Caminho
       ).
12
13
14 ppTodasSolucoes(L):- findall((S,C), (resolveDP(S), length(S,C)),
       L).
15
```

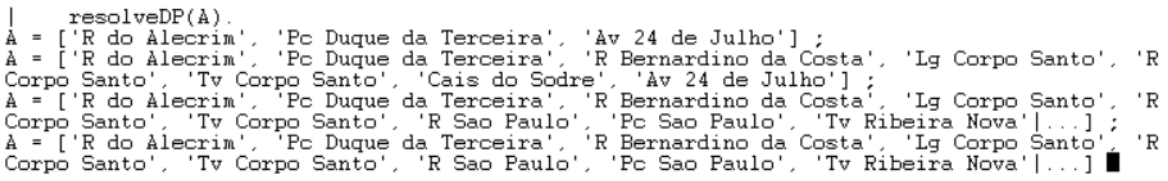

Figure 1: resolveDP

- Depth First com Custos

  Predicado que efetua uma pesquisa Depth First, devolvendo a lista de Arcos e respetivo custo.

```
1
2  resolveDPC([Nodo|Caminho], Custo):-
3      inicial(Nodo),
4      depthFirstC(Nodo,[Nodo], Caminho, Custo).
5
6  depthFirstC(Nodo, _, [], 0):-
7      final(Nodo).
8
9  depthFirstC(Nodo, Historico, [NodoProx|Caminho], Custo):-
10     adjacente(Nodo, NodoProx, CustoArco),
11     nao(membro(NodoProx, Historico)),
12     depthFirstC(NodoProx, [NodoProx|Historico], Caminho,
       CustoAux),
13     Custo is CustoArco + CustoAux.
14
```

```
?- resolveDPC(A,C).
A = ['R do Alecrim', 'Pc Duque da Terceira', 'Av 24 de Julho'],
C = 16.28674656442929 ;
A = ['R do Alecrim', 'Pc Duque da Terceira', 'R Bernardino da Costa', 'Lg Corpo Santo', 'R
Corpo Santo', 'Tv Corpo Santo', 'Cais do Sodre', 'Av 24 de Julho'],
C = 39.608336766851494 ;
A = ['R do Alecrim', 'Pc Duque da Terceira', 'R Bernardino da Costa', 'Lg Corpo Santo', 'R
Corpo Santo', 'Tv Corpo Santo', 'R Sao Paulo', 'Pc Sao Paulo', 'Tv Ribeira Nova'|...],
C = 53.5135425504076
```

Figure 2: resolveDPC

- Depth First Produtividade

  Como fator de produtividade considerei a quantidade de lixo transportada, assim vi-me obrigado a criar um novo algoritmo de procura Depth First que tem em conta a quantidade de lixo transportada pelo camião. De seguida apresento o Predicado que faz uma travessia Depth First ao mapa, dando uma lista de vários caminhos e a quantidade de lixo em litros que é transportada nesse caminho.

```
resolveDPProdutividade([Nodo|Caminho], Custo):-
    inicial(Nodo),
    depthFirstP(Nodo,[Nodo], Caminho, Custo).

depthFirstP(Nodo, _, [], 0):-
    final(Nodo).

depthFirstP(Nodo, Historico, [NodoProx|Caminho], Custo):-
    adjacente(Nodo, NodoProx,_),
    calculaP(NodoProx, CustoArco),
    nao(membro(NodoProx, Historico)),
    depthFirstP(NodoProx, [NodoProx|Historico], Caminho,
   CustoAux),
    Custo is CustoArco + CustoAux.

totalLixo([],0).
totalLixo([X|T],Tx):- totalLixo(T,Ty), Tx is X + Ty.

calculaP(Nodo,Total) :- findall(C,pontos_recolha(_,_,Nodo,_,C,_)
   ,R),totalLixo(R,Total).

```

```
?- resolveDPProdutividade(A,B).
A = ['R do Alecrim', 'Pc Duque da Terceira', 'Av 24 de Julho'],
B = 4690 ;
A = ['R do Alecrim', 'Pc Duque da Terceira', 'R Bernardino da Costa', 'Lg Corpo Santo', 'R
Corpo Santo', 'Tv Corpo Santo', 'Cais do Sodre', 'Av 24 de Julho'],
B = 23350 ;
```

Figure 3: resolveDPProdutividade

- Depth First Procura por Tipos

  Predicado que efetua uma pesquisa Depth First para caixotes com o mesmo tipo de lixo, devolvendo a lista de Arcos.

```
resolveDPTipos([Nodo|Caminho], Tipo):-
    inicial(Nodo),
    depthFirstT(Nodo,[Nodo],Caminho, Tipo).

depthFirstT(Nodo,_,[], Tipo):-
    final(Nodo).

depthFirstT(Nodo, Historico, [NodoProx|Caminho], Tipo):-
    getLixo(NodoProx, Tipo),
    adjacente(Nodo, NodoProx,_),
    nao(membro(NodoProx, Historico)),
    depthFirstT(NodoProx, [NodoProx|Historico], Caminho, Tipo).

resolveDPTiposAll(L,T):- findall((S,C), (resolveDPTipos(S, T),
   length(S,C)),L).


```

```
?- resolveDPTipos(A,'Lixos').
A = ['R do Alecrim', 'R Nova do Carvalho', 'Tv Corpo Santo', 'R Corpo
Santo', 'R Sao Paulo', 'R da Boavista', 'Lg Conde-Barao', 'Tv do Cais
do Tojo', 'R Cais do Tojo'|...]
```

Figure 4: resolveDPtipos

7.1.2 Breadth First

Em contraste com a busca em profundidade, a busca em largura escolhe primeiro visitar aqueles nós mais próximos do nó inicial. O algoritmo não é tão simples, pois é necessário manter um conjunto de nós candidatos alternativos e não apenas um único, como na busca em profundidade, além disso, só o conjunto por si só é insuficiente se o caminho da solução também for necessário, assim, ao invés de manter um conjunto de nós candidatos, é necessário manter um conjunto de caminhos candidatos. O que torna esta procura pouco eficiente sendo que utiliza muita memória.

- Breadth First

  Predicado que faz uma travessia Breadth First pelo grafo devolvendo as listas de nodos por onde passa.

```
1  resolveBFSall(L) :- findall((S,C), (resolveBFS(S,C)), L).
2
3  resolveBFS(No,Solucao) :-
4      inicial(No),
5      breadthFirst([[No]],Sol),
6      reverseL(Sol,Solucao).
7
8
9  breadthFirst([[No|Caminho]|_],[No|Caminho]) :-
10     final(No).
11
12 breadthFirst([Caminho|Caminhos],Solucao) :-
13     estender(Caminho,NovosCaminhos),
14     append(Caminhos,NovosCaminhos,Caminhos1),
15     breadthFirst(Caminhos1,Solucao).
16
17 estender([No|Caminho],NovosCaminhos) :-
18     findall([NovoNo,No|Caminho],
19         (adjacente(No,NovoNo,_), \+ membro(NovoNo,[No|Caminho]))
       ,
20         NovosCaminhos).
21
22
```

```
|    resolveBFS(A,B).
A = 'R do Alecrim',
B = ['R do Alecrim', 'Pc Duque da Terceira', 'Av 24 de Julho']
Unknown action: , (h for help)
Action? ;
A = 'R do Alecrim',
B = ['R do Alecrim', 'Pc Duque da Terceira', 'Cais do Sodre', 'Av 24 d
e Julho'] ;
A = 'R do Alecrim',
B = ['R do Alecrim', 'R Nova do Carvalho', 'Tv dos Remolares', 'Av 24
de Julho'] ;
A = 'R do Alecrim',
B = ['R do Alecrim', 'Pc Duque da Terceira', 'R Remolares', 'Tv dos Re
molares', 'Av 24 de Julho'] ;
A = 'R do Alecrim',
B = ['R do Alecrim', 'R Nova do Carvalho', 'Tv Corpo Santo', 'Cais do
Sodre', 'Av 24 de Julho'] ;
```

Figure 5: resolveBFS

- Breadth First Custos

  Predicado que percorre o grafo em largura e devolve uma lista com os nodos percorridos e o custo dessa travessia.

```
melhorBFS(Nodo,Cam,Custo):- findall((Ca,Cus), (resolveBFSC(Ca,
    Cus)), L),minimo(L,(Cam,Custo)).

resolveBFSCustosAll(L):- findall((S,C), (resolveBFSC(S,C)),L).

resolveBFSC(Solucao, Custo) :-
    inicial(No),
    resolveBFSC2([[No]],Sol),
    reverseL(Sol,Solucao),
    custoTotal(Solucao, Custo).

resolveBFSC2([[No|Caminho]|_],[No|Caminho]) :-
    final(No).

resolveBFSC2([Caminho|Caminhos],Solucao) :-
    estender(Caminho,NovosCaminhos),
    append(Caminhos,NovosCaminhos,Caminhos1),
    resolveBFSC2(Caminhos1,Solucao).

solveBreadthFirstAll(L) :- findall((S,C,Length), (resolveBFS(S,C
    ), comprimento(C,Length)), L).

custoTotal([],0).
custoTotal([No],0).
custoTotal([No1,No2|Caminho],Custo) :-
    adjacente(No1,No2,CustoArco),
    custoTotal([No2|Caminho],CustoResto),
    Custo is CustoArco + CustoResto.
```

```
?- resolveBFSC(A,C).
A = ['R do Alecrim', 'Pc Duque da Terceira', 'Av 24 de Julho'],
C = 16.28674656442929
```

Figure 6: resolveBFSC

7.1.3 Depth First Limitada

De forma a tentar combater os problemas da Depth First normal, surge a Depth First limitada, utiliza um custo limite decrementando em cada iteração até chegar a 0, esta procura trás problemas porque não se tem previamente um limite razoável a impor, caindo muitas vezes na tentativa erro.

- Depth First Limitada

  Predicado que faz a travessia ao grafo em profundidade até atingir um limite.

```
resolveDPlimitadaAll(L,Size):- findall((S,C), (resolveDPlimitada
   (S,Size), length(S,C)), L).

resolveDPlimitada(Solucao,L) :-
    inicial(No),
    depthFirstLimited([],No,Sol,L),
    reverseL(Sol,Solucao).

depthFirstLimited(Caminho,No,[No|Caminho],_) :-
    final(No),!.

depthFirstLimited(Caminho,No,S,L) :-
    L > 0,
    adjacente(No,No1,_),
    \+ membro(No1,Caminho),
    L1 is L - 1,
    depthFirstLimited([No|Caminho],No1,S,L1).

```

```
|    resolveDPlimitada(L,5).
L = ['R do Alecrim', 'Pc Duque da Terceira', 'Av 24 de Julho'] ;
L = ['R do Alecrim', 'Pc Duque da Terceira', 'R Bernardino da Costa',
'Lg Corpo Santo', 'Cais do Sodre', 'Av 24 de Julho'] ;
L = ['R do Alecrim', 'Pc Duque da Terceira', 'R Bernardino da Costa',
'Tv Corpo Santo', 'Cais do Sodre', 'Av 24 de Julho'] ;
L = ['R do Alecrim', 'Pc Duque da Terceira', 'R Remolares', 'Tv dos Re
molares', 'Av 24 de Julho'] 
```

Figure 7: resolveDPlimitada

- Depth First Limitada Custos

  Predicado que faz a travessia ao grafo em profundidade até atingir um limite, devolvendo uma lista de nodos que percorre e o custo associado a essa travessia.

```
resolveDPlimitadaCostAll(L,Size) :- findall((S,C,Cost), (
   resolveDPlimitadaC(S,Size,Cost), comprimento(S,C)), L).

resolveDPlimitadaC(Solucao,L,Custo) :-
    inicial(No),
    depthFirstLimitedC([],No,Sol,L,Custo),
    reverseL(Sol,Solucao).



depthFirstLimitedC(Caminho,No,[No|Caminho],L,0) :-
    final(No),!.

depthFirstLimitedC(Caminho,No,S,L, Custo) :-
    L > 0,
    adjacente(No,No1,CustoArco),
    \+ membro(No1,Caminho),
    L1 is L - 1,
    depthFirstLimitedC([No|Caminho],No1,S,L1, CustoAux),
    Custo is CustoArco + CustoAux.

```

```
?- resolveDPlimitadaC(L,5,C).
L = ['R do Alecrim', 'Pc Duque da Terceira', 'Av 24 de Julho'],
C = 16.28674656442929 ;
L = ['R do Alecrim', 'Pc Duque da Terceira', 'R Bernardino da Costa',
'Lg Corpo Santo', 'Cais do Sodre', 'Av 24 de Julho'],
C = 36.76145287296344 ;
```

Figure 8: resolveDPlimitadaC

- Depth First Limitada Tipos

  Predicado que faz a travessia ao grafo em profundidade até atingir um limite, devolvendo uma lista de nodos que percorre sendo todos estes de um determinado tipo de lixo.

```
        resolveDPlimitadaTipo(Solucao,L, Tipo) :-
    inicial(No),
    depthFirstLimitedT([],No,Sol,L,Tipo),
    reverseL(Sol,Solucao).

depthFirstLimitedT(Caminho,No,[No|Caminho],L, Tipo) :-
    final(No),!.

depthFirstLimitedT(Caminho,No,S,L,Tipo) :-
    L > 0,
    getLixo(No1, Tipo),
    adjacente(No,No1,_),
    \+ membro(No1,Caminho),
    L1 is L - 1,
    depthFirstLimitedT([No|Caminho],No1,S,L1,Tipo).\\
```

```
|    resolveDPlimitadaTipo(L,5,'Lixos').
L = ['R do Alecrim', 'R Nova do Carvalho', 'Tv Corpo Santo', 'R Bernar
dino da Costa', 'Pc Duque da Terceira', 'Av 24 de Julho'] ;
L = ['R do Alecrim', 'R Nova do Carvalho', 'Tv Corpo Santo', 'R Sao Pa
ulo', 'Tv dos Remolares', 'Av 24 de Julho'] ;
```

Figure 9: resolveDPlimitadaTipo

7.2 Procura Informada

7.2.1 Procura Gulosa

´

Esta pesquisa, tem como estratégia estimar os nodos mais próximos do Destino final e tomar esse trajeto na travessia, porém não encontra sempre a melhor solução.

- Gulosa

  Predicado que faz a travessia por procura gulosa.

```
1
2  resolveGulosa(Nodo, Caminho/Custo) :-
3      estimativa(Nodo, Estimativa),
4      agulosa([[Nodo]/0/Estimativa], CaminhoInverso/Custo/_),
5      reverseL(CaminhoInverso, Caminho).
6
7  agulosa(Caminhos, Caminho) :-
8      obtem_melhor_g(Caminhos, Caminho),
9      Caminho = [Nodo|_]/_/_,final(Nodo).
10
11 agulosa(Caminhos, SolucaoCaminho) :-
12     obtem_melhor_g(Caminhos, MelhorCaminho),
13     seleciona(MelhorCaminho, Caminhos, OutrosCaminhos),
14     expandeGulosa(MelhorCaminho, ExpCaminhos),
15     append(OutrosCaminhos, ExpCaminhos, NovoCaminhos),
16     agulosa(NovoCaminhos, SolucaoCaminho).
17
18 obtem_melhor_g([Caminho], Caminho) :- !.
19
20 obtem_melhor_g([Caminho1/Custo1/Est1,_/Custo2/Est2|Caminhos],
       MelhorCaminho) :-
21   Est1 =< Est2, !,
22   obtem_melhor_g([Caminho1/Custo1/Est1|Caminhos], MelhorCaminho)
       .
23
24 obtem_melhor_g([_|Caminhos], MelhorCaminho) :-
25   obtem_melhor_g(Caminhos, MelhorCaminho).
26
27 expandeGulosa(Caminho, ExpCaminhos) :-
28   findall(NovoCaminho, adjacenteG(Caminho,NovoCaminho),
       ExpCaminhos).
29
30 adjacenteG([Nodo|Caminho]/Custo/_, [ProxNodo,Nodo|Caminho]/
       NovoCusto/Est) :-
31   adjacente(Nodo, ProxNodo, PassoCusto),
32     nao(membro(ProxNodo, Caminho)),
33   NovoCusto is Custo + PassoCusto,
```

```
34    estimativa(ProxNodo, Est).
35
36
37
```

```
|    resolveGulosa(A,L).
A = 'R do Alecrim',
L = ['R do Alecrim', 'R Nova do Carvalho', 'Tv dos Remolares', 'Av 24
de Julho']/13.977910300996466 ;
```

Figure 10: resolveGulosa

- Gulosa c/Tipo

  Predicado que faz a travessia por procura gulosa, atendendo ao tipo de Lixo que se encontra em cada nodo.

```
resolveGulosaT(Nodo,Caminho/Custo,Tipo):-
    estimativa(Nodo,Estimativa),
    agulosaT([[Nodo]/0/Estimativa],CaminhoInverso/Custo/_,Tipo),
    reverse(CaminhoInverso,Caminho).

agulosaT(Caminhos,Caminho,Tipo):-
    obtem_melhor_g(Caminhos,Caminho),
    Caminho = [Nodo|_]/_/_,final(Nodo).

agulosaT(Caminhos,SolucaoCaminho,Tipo):-
    obtem_melhor_g(Caminhos, MelhorCaminho),
    seleciona(MelhorCaminho, Caminhos, OutrosCaminhos),
    expandeGulosaT(MelhorCaminho,ExpCaminhos,Tipo),
    append(OutrosCaminhos,ExpCaminhos,NovoCaminhos),
    agulosaT(NovoCaminhos,SolucaoCaminho,Tipo).

expandeGulosaT(Caminho,ExpCaminhos,Tipo):-
    findall(NovoCaminho,adjacenteGT(Caminho,NovoCaminho,Tipo),
   ExpCaminhos).

adjacenteGT([Nodo|Caminho]/Custo/_,[ProxNodo,Nodo|Caminho]/
   NovoCusto/Est,Tipo):-
    getLixo(NodoProx,Tipo),
    adjacente(Nodo,ProxNodo,PassoCusto),
    nao(membro(ProxNodo,Caminho)),
    NovoCusto is Custo+PassoCusto,
    estimativa(ProxNodo,Est).


```

```
|    resolveGulosaT(A,L,'Lixos').
A = 'R do Alecrim',
L = ['R do Alecrim', 'R Nova do Carvalho', 'Tv dos Remolares', 'Av 24
de Julho']/13.977910300996466 ;
```

Figure 11: resolveGulosaT

7.2.2 Procura A*

Esta procura evita expandir caminhos que são caros ,combina de certa forma a pesquisa gulosa com a uniforme, minimizando a soma do caminho já efetuado com o mínimo previsto que falta até a solução, em termos de complexidade é mais eficiente que a procura gulosa.

Predicado que faz uma travessia por A*.

```
1
2  resolveAEstrela(Caminho/Custo) :-
3      inicial(Nodo),
4      pontos_recolha(_,_,Nodo,_,Cap,_),
5      aestrela([[Nodo]/0/pontos_recolha], CaminhoInverso/Custo/_),
6      reverseL(CaminhoInverso, Caminho).
7
8  aestrela(Caminhos, Caminho) :-
9      obtem_melhor(Caminhos, Caminho),
10     Caminho = [Nodo|_]/_/_,
11     final(Nodo).
12
13 aestrela(Caminhos, SolucaoCaminho) :-
14     obtem_melhor(Caminhos, MelhorCaminho),
15     seleciona(MelhorCaminho, Caminhos, OutrosCaminhos),
16     expandeAEstrela(MelhorCaminho, ExpCaminhos),
17     append(OutrosCaminhos, ExpCaminhos, NovoCaminhos),
18     aestrela(NovoCaminhos, SolucaoCaminho).
19
20 obtem_melhor([Caminho], Caminho) :- !.
21
22 obtem_melhor([Caminho1/Custo1/Est1,_/Custo2/Est2|Caminhos],
       MelhorCaminho) :-
23
24     Custo1 + Est1 =< Custo2 + Est2, !,
25     obtem_melhor([Caminho1/Custo1/Est1|Caminhos], MelhorCaminho).
26
27 obtem_melhor([_|Caminhos], MelhorCaminho) :-
28     obtem_melhor(Caminhos, MelhorCaminho).
29
30 expandeAEstrela(Caminho, ExpCaminhos) :-
31 findall(NovoCaminho, adjacenteG(Caminho,NovoCaminho), ExpCaminhos).
```

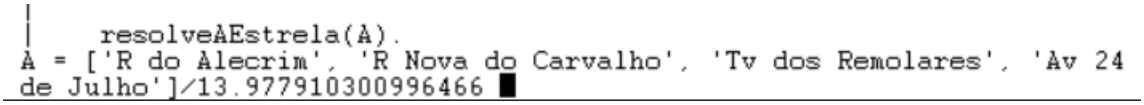

Figure 12: resolveAEstrela

7.3 Outros Requisitos

Considerando que um dos requisitos do projeto era conseguir gerar os circuitos de recolha tanto indiferenciada como seletiva, tendo este requisito cumpridos com os predicados de travessia demostrados a cima, vou então mostrar a minha resposta aos restantes requisitos.

- Identificar quais os circuitos com mais pontos de recolha (por tipo de resíduo a recolher);

  Predicado que devolve o menor número de arcos necessário para chegar da Origem ao Destino.

```
1 menorNArcosDP(Nodo,Cam,NA):- findall((Ca,N), (resolveDP(Ca),
     length(Ca,N)),L),minimo(L,(Cam,NA)).
2
3
```

  Predicado que devolve a travessia com maior número de arcos para chegar da Origem ao Destino.

```
1 maiorNARCOSDP(Nodo,Cam,NA):- findall((Ca,N), (resolveDP(Ca),
     length(Ca,N)),L),maximo(L,(Cam,NA)).
2
```

```
?- menorNArcosDP(A,B,L).
B = ['R do Alecrim', 'Pc Duque da Terceira', 'Av 24 de Julho'],
L = 3 .

?- maiorNARCOSDP(A,B,L).
B = ['R do Alecrim', 'Pc Duque da Terceira', 'R Bernardino da Costa',
'Lg Corpo Santo', 'R Corpo Santo', 'Tv Corpo Santo', 'R Nova do Carval
ho', 'Tv dos Remolares', 'R Remolares'|...],
L = 25
```

Figure 13: menor e maior nº de Arcos com DP

  Predicados que testam qual o caminho para um tipo de lixo especifico com mais nodos e com menos nodos.

```
1 menorNArcosDPT(Tipo,Cam,NA):- findall((Ca,N), (resolveDPTipos(Ca
     ,Tipo), length(Ca,N)),L),minimo(L,(Cam,NA)).
2 maiorNARCOSDPT(Tipo,Cam,NA):- findall((Ca,N), (resolveDPTipos(Ca
     ,Tipo), length(Ca,N)),L),maximo(L,(Cam,NA)).\\
```

  Predicados que encontram o caminho com menos e com mais nodos na travessia, para um dado limite.

```
|
|    maiorNARCOSDPT('Lixos',A,B).
A = ['R do Alecrim', 'Pc Duque da Terceira', 'R Bernardino da Costa', 'Lg Corpo Santo', 'R
Corpo Santo', 'Tv Corpo Santo', 'R Nova do Carvalho', 'Tv dos Remolares', 'R Remolares'|...
],
B = 25 ,

?- menorNArcosDPT('Lixos',A,B).
A = ['R do Alecrim', 'Pc Duque da Terceira', 'Av 24 de Julho'],
B = 3
```

Figure 14: menor e maior nº de Arcos com DP por Tipo de lixo

```
1 menorNArcosDPlimitada(Limite,Cam,NA):- findall((Ca,N), (
      resolveDPlimitada(Ca,Limite), length(Ca,N)),L),minimo(L,(Cam
      ,NA)).
2 maiorNArcosDPlimitada(Limite,Cam,NA):- findall((Ca,N), (
      resolveDPlimitada(Ca,Limite), length(Ca,N)),L),maximo(L,(Cam
      ,NA)).
3
```

```
|
|    menorNArcosDPlimitada(5,C,N).
C = ['R do Alecrim', 'Pc Duque da Terceira', 'Av 24 de Julho'],
N = 3 ,

?- maiorNArcosDPlimitada(5,C,N).
C = ['R do Alecrim', 'Pc Duque da Terceira', 'R Bernardino da Costa',
'Lg Corpo Santo', 'Cais do Sodre', 'Av 24 de Julho'],
N = 6
```

Figure 15: menor e maior nº de Arcos com DP limitada

- Comparar circuitos de recolha tendo em conta os indicadores de produtividade;

  Considerei como indicador de produtividade a quantidade de lixo que recolhe na travessia, assim criei o Predicado mais Lixo que encontra qual o caminho em que o camião transporta mais lixo e que quantidade/litro transporta.

```
1 maisLixo(Nodo,Cam,Custo):- findall((Ca,Cus), (
     resolveDPProdutividade(Ca,Cus)), L),maximo(L,(Cam,Custo)).
2
```

```
?- maisLixo(A,B,C).
B = ['R do Alecrim', 'Pc Duque da Terceira', 'R Bernardino da Costa', 'Lg Corpo Santo', 'Ca
is do Sodre', 'Tv Corpo Santo', 'R Nova do Carvalho', 'Tv dos Remolares', 'R Remolares'|...
],
C = 165190
```

Figure 16: Caminho com mais Lixo

- Escolher o circuito mais rápido (usando o critério da distância;

  Considerei também que o caminho mais eficiente é o caminho mais rápido logo este ponto alberga o 4^{o} e 5^{o} requisito.

  Predicado que encontra o caminho com menor custo.

```
1 melhor(Nodo,Cam,Custo):- findall((Ca,Cus), (resolveDPC(Ca,Cus)),
     L),minimo(L,(Cam,Custo)).
```

  Predicado que encontra o caminho com maior custo.

```
1 pior(Nodo,Cam,Custo):- findall((Ca,Cus), (resolveDPC(Ca,Cus)), L
    ),maximo(L,(Cam,Custo)).
```

```
?- melhor(A,B,C).
B = ['R do Alecrim', 'R Nova do Carvalho', 'Tv dos Remolares', 'Av 24
de Julho'],
C = 13.977910300996466 .

?- pior(A,B,C).
B = ['R do Alecrim', 'Pc Duque da Terceira', 'R Bernardino da Costa',
'Lg Corpo Santo', 'Cais do Sodre', 'Tv Corpo Santo', 'R Nova do Carval
ho', 'Tv dos Remolares', 'R Remolares'|...],
C = 270.4683447300729
```

Figure 17: Caminho mais curto e mais longo

  Predicados que encontram o caminho com menor e maior custo na travessia, para um dado limite.

```
melhorDPlimitada(Limite,Cam,Custo):- findall((Ca,Cus), (
    resolveDPlimitadaC(Ca,Limite,Cus)),L),minimo(L,(Cam,Custo)).
piorDPlimitada(Limite,Cam,Custo):- findall((Ca,Cus), (
    resolveDPlimitadaC(Ca,Limite,Cus)),L),maximo(L,(Cam,Custo)).


```

```
|
|    melhorDPlimitada(5,C,A).
C = ['R do Alecrim', 'R Nova do Carvalho', 'Tv dos Remolares', 'Av 24
de Julho'],
A = 13.977910300996466 ,

?- piorDPlimitada(5,C,A).
C = ['R do Alecrim', 'Pc Duque da Terceira', 'R Remolares', 'R Ribeira
 Nova', 'Pc Dom Luis I', 'Av 24 de Julho'],
A = 53.00171159968518 ,

?-
```

Figure 18: Caminho mais curto e mais longo usando DP limitada

8 Resultados Obtidos

Para avaliar os tempos de cada algoritmo de procura decidi procurar correr os algoritmos de forma a obter a melhor solução que cada um desses algoritmos consegue encontrar assim defini apenas os predicados melhor custo para as pesquisas por procura não Informada, uma vez que as Informada já vão directamente a procura do melhor caminho estimado possível.

Estratégia	Tempo (segundos)	Espaço	Profundidade/Custo	Encontrou a melhor solução?
DFS	9.43	O(bm)	13.97791	SIM
DPS limitada	0.191	O(bl)	13.97791	SIM quando Limite>Profundidade Solução
BFS	<->	O(bˆd)	<->	NÃO (mas na teoria devia encontrar)
Gulosa	0.001	<->	13.97791	SIM
A*	0.001	<->	13.97791	SIM

Como podemos verificar pela tabela a cima a travessia Breadth First é a menos eficiente de todas, uma vez que nem consegui correr o algoritmo de melhor caminho, devido ao facto de utilizar muita memória.

```
|    time(melhor(A,B,C)).
% 86,186,693 inferences, 9.344 CPU in 9.343 seconds (100% CPU, 9223994 Lips)
B = ['R do Alecrim', 'R Nova do Carvalho', 'Tv dos Remolares', 'Av 24 de Julho'],
C = 13.977910300996466 .

?- time(melhorDPlimitada(15,B,C)).
% 3,622,377 inferences, 0.203 CPU in 0.191 seconds (106% CPU, 17833241 Lips)
B = ['R do Alecrim', 'R Nova do Carvalho', 'Tv dos Remolares', 'Av 24 de Julho'],
C = 13.977910300996466 .

?- time(melhorBFS(A,B,C)).
```

```
|    time(resolveGulosa(A,B)).
% 393 inferences, 0.000 CPU in 0.001 seconds (0% CPU, Infinite Lips)
A = 'R do Alecrim',
B = ['R do Alecrim', 'R Nova do Carvalho', 'Tv dos Remolares', 'Av 24 de Julho']/13.9779103
00996466 .

?-
|    time(resolveAEstrela(A)).
% 3,933 inferences, 0.000 CPU in 0.001 seconds (0% CPU, Infinite Lips)
A = ['R do Alecrim', 'R Nova do Carvalho', 'Tv dos Remolares', 'Av 24 de Julho']/13.9779103
00996466
```

8.1 Análise de Resultados

No que toca a Procura não Informada em termos de eficácia, a Depth First revelou-se mais eficaz que a Breadth First uma vez que consegue identificar os melhores caminhos com menores custos em menos tempo, porém penso que a travessia em Largura apesar de ser claramente mais custosa, (tão custosa que nem consegui encontrar todos os caminhos com o seu uso), poderia ser um pouco otimizada de forma a tentar equilibrar um pouco estes resultados. Em contra partida, a Depth First Limitada é melhor em termos de tempo que a Depth First normal, aquando o limite é superior à profundidade da solução, uma vez que reduz a quantidade total de caminhos que ele tem de descobrir/percorrer, para o limite, isso explica o porquê de existir uma diferença tão significativa entres estes dois tempos.

Relativamente à procura Informada, o método da procura Gulosa deu o resultado para o melhor custo correto, porém existe a possibilidade de esta poder falhar,já a A* deu também o melhor resultado e como já referido anteriormente oferece mais garantias, do que a procura gulosa, sendo por isso talvez a melhor alternativa da procura Informada.

Em termos de resultados, praticamente todas as procuras deram valores em concordância com o esperado, uma vez que existe já um conhecimento sobre o funcionamento destes algoritmos.

Relativamente às Queries pedidas pelos docentes, penso que consegui realizalas todas, umas vez que tenho predicados que são capazes de gerar os circuitos de recolha tanto indiferenciada como seletiva (todos os algoritmos de travessia apresentados a cima), identificar quais os circuitos com mais pontos de recolha (Depth First menorNArcos e maiorNArcos), comparar circuitos de recolha tendo em conta os indicadores de produtividade (predicado maisLixo), escolher o circuito mais rápido (usando o critério da distância).

9 Conclusão

Este trabalho foi bastante útil para sedimentar os conhecimentos apreendidos ao longo da UC, mais especificamente os das últimas aulas. Relativamente ao trabalho penso que a parte que me foi mais difícil foram as decisões que tive de tomar na fase inicial do projeto relativamente à necessidade de reestruturação do dataset fornecido pelos docentes, e consequentemente fazer o parse dessa informação. Ter de avaliar que informação ia considerar relevante ou não, o que aliado á falta de tempo, tornou este projeto um pouco mais complicado. Relativamente aos predicados utilizados para a resolução das Queries pedidas, não houve grande dificuldade uma vez que praticamente todos os algoritmos foram lecionados nas aulas.

Fiquei um pouco desapontado por não ter tido tempo de ter em conta a capacidade do camião, pondo este tópico como uma das principais melhorias a fazer ao meu projeto, porém existem mais melhorias que poderia ter procurado fazer como, por exemplo, a procura de algoritmos mais eficientes de pesquisa e a inserção/procura de novos algoritmos não fornecidos nas aulas práticas, entre outros.

Por fim, vejo-me feliz com o resultado final, apesar de haver ainda muito espaço para melhorar penso que consegui cumprir todos os objetivos propostos pelos docentes, espero porém ter feito as melhores decisões relativamente ao parsing do dataset fornecido e ter avaliado da melhor forma as queries propostas.

10 Referências

SWI-Prolog: https://www.swi-prolog.org/pldoc/doc$_f or?object = manual$

11 Apêndice

11.1 Parser

```
from math import sin, cos, sqrt, atan2
import numpy as np
import re
import pandas as pd



def calcDist(lat1, long1, lat2, long2):
    R = 6373.0
    dlon = long2 - long1
    dlat = lat2 - lat1
    a = (sin(dlat/2))**2 + cos(lat1) * cos(lat2) * (sin(dlon/2))**2
    c = 2 * atan2(sqrt(a), sqrt(1-a))

    return R*c

dataset = pd.read_excel(r
"c:\Users\LENOVO\\Desktop\Carlos\Universidade\\SRCR\Individual\\dataset_otimo.xlsx")

dataset['PONTO_RECOLHA_LOCAL'] = dataset['PONTO_RECOLHA_LOCAL'].str.normalize('NFKD'
str.encode('ascii', errors='ignore').str.decode('utf-8')
dataset['CONTENTOR_RESÍDUO'] = dataset['CONTENTOR_RESÍDUO'].str.normalize('NFKD').
str.encode('ascii', errors='ignore').str.decode('utf-8')


dadosDatasetUniversal = {}
dadosDataset = {}
dadosDataset['Lixos'] = {}
dadosDataset['Vidro'] = {}
dadosDataset['Papel e Cartao'] = {}
dadosDataset['Embalagens'] = {}
dadosDataset['Organicos'] = {}


filePontos = open("pontos.pl", "w", encoding="UTF-8")
filePontos.write("%%pontos_recolha(latitude, longitude,
```

```
"local, tipo, quantidade, [lista]).\n\n")


for linha in dataset.values:

    ponto = linha[2]
    tipo = linha[3]
    res = re.search(r'([\w, -\/]+)(\[(.*)\])?', ponto)
    rua = re.split(r',', res[1])[0]
    if rua[-1] == " ":
        rua = rua[:-1]

    if rua not in dadosDataset[tipo]:
         dadosDataset[tipo][rua] = {
            'lat':linha[0],
            'long':linha[1],
            'dest':[],
            'quant':0
            }
    dadosDataset[tipo][rua]['quant'] += linha[4]

    if rua not in dadosDatasetUniversal:
        dadosDatasetUniversal[rua] = {
                'lat':linha[0],
                'long':linha[1],
                'dest':[],
                'quant':0
        }
    dadosDatasetUniversal[rua]['quant'] += linha[4]

    caminhos = ""
    if res[3] is not None:
        destinos = re.split(r' ?, ?', res[3])
        for destino in destinos:
            caminhos += "'" + destino + "',"
            dadosDataset[tipo][rua]['dest'].append(destino)
            if destino not in dadosDatasetUniversal[rua]['dest']:
                dadosDatasetUniversal[rua]['dest'].append(destino)
        caminhos = caminhos[:-1]
```

```
    filePontos.write("pontos_recolha("  + str(linha[0])
                                + ", "  + str(linha[1])
                                + ", '" + rua + "'"
                                + ", '" + tipo + "'"
                                + ", "  + str(linha[4])
                                + ", [" + str(caminhos)
                                + "]).\n")

dadosDatasetUniversal['Bqr dos Ferreiros'] = {'lat':-9.149144,
'long':38.708209, 'dest':[], 'quant':0}
dadosDatasetUniversal['Av Dom Carlos I']   = {'lat':-9.153078,
'long':38.709049, 'dest':[], 'quant':0}

fileArcos = open("arcos.pl", "w+", encoding="UTF-8")
fileArcos.write("%%arco(RUA1, RUA2, DIST).\n\n")

for (rua,valor) in dadosDatasetUniversal.items():
    for destino in valor['dest']:
        if destino in dadosDatasetUniversal:
            quantidade = calcDist(valor['lat'],valor['long'],
            dadosDatasetUniversal[destino]['lat'],
            dadosDatasetUniversal[destino]['long'])
            fileArcos.write("arco(" + "'" + rua + "'" +
                            ", " + "'" + destino + "'" +
                            ", " + str(quantidade) + ").\n")
filePontos.close()
fileArcos.close()


fileArcos_aux = open("arcos.pl", "r", encoding="UTF-8")
fileFinalArcos = open("arcos_final.pl", "w", encoding="UTF-8")
fileFinalArcos.write("%%arco(RUA1, RUA2, DIST).\n\n")

i =1
listquant =[]
for line in fileArcos_aux:

    if i>2 and i <912:
        par = re.search(r'.* ([\d.]+)\).', line)
```

```
            quant = par[1]
            if quant not in listquant:
                listquant.append(quant)
                fileFinalArcos.write(line)
        i+=1

fileArcos_aux.close()
fileFinalArcos.close()
```

11.2 travessias.pl

```
%--------------------------------- - - - - - - - - - - - - - - - - -
% SIST. REPR. CONHECIMENTO E RACIOCINIO - MiEI/3

%--------------------------------- - - - - - - - - - - - - - - - - -
% Programacao em logica

%---------------------------------  base de conhecimento ---------


:-include('pontos.pl').
%%pontos_recolha(latitude, longitude, local, tipo, quantidade, [
    lista]).

:-include('arcos_final.pl').
%%arco(RUA1, RUA2, DIST).


inicial('R do Alecrim').
%final('Av 24 de Julho').
final('Pc Duque da Terceira').
%final('R da Boavista').
%final('R Ferragial').

% -------------------Auxiliares -------------

%verifica se    membro da lista
membro(X, [X|_]).
membro(X, [_|Xs]):-
  membro(X, Xs).


%Extensao do meta-predicado nao
```

```
32 nao( Questao ) :- Questao, !, fail.
33 nao( Questao ).
34
35 %Calcula o comprimento de uma lista
36 comprimento(S,N) :- length(S,N).
37
38 %Predicado que da um print no terminal
39 imprime([]).
40 imprime([X|T]) :- write(X), nl, imprime(T).
41
42 %verifica se um elemento pertence a uma  lista
43 temElem([],_).
44 temElem(L,[H|T]):- membro(H,L).
45 temElem(L,[H|T]):- temElem(L,T);memberchk(H,L).
46
47 %Obtem um arco apartir de um id
48 adjacente(Origem,Destino,Dist) :- arco(Origem,Destino,Dist).
49 adjacente(Origem,Destino,Dist) :- arco(Destino,Origem,Dist).
50
51 %verifica se uma lista esta vazia
52 estaVazia(L,V) :- comprimento(L,V),nao(V>0).
53
54 %encontra o menor elemento
55 minimo(L, (A,B)) :-
56     seleciona((A,B), L, R),
57     \+ ( membro((A1,B1), R), B1 < B ).
58
59 %encontra o maior elemento
60 maximo(L, (A,B)) :-
61     seleciona((A,B), L, R),
62     \+ ( membro((A1,B1), R), B1 > B ).
63
64 %reverte uma lista
65 reverseL(Ds,Es) :- reverseL(Ds, [], Es).
66 reverseL([],Ds,Ds).
67 reverseL([D|Ds],Es,Fs) :- reverseL(Ds, [D|Es], Fs).
68
69 %verifica qual o tipo de lixo de um ponto recolha
70 getLixo(Local, Tipo) :- pontos_recolha(_,_,Local,Tipo,_, _).
71
72 %Estima o custo at  ao nodo final
73 estimativa(Nodo,Est):-
74     distance(Nodo, Est).
75
76 %Calcula a distancia at  ao nodo final
77 distance(Origem,Dis):-
78     pontos_recolha(Lat1,Lon1,Origem,_,_,_),
79     final(Destino),
```

```
80      pontos_recolha(Lat2,Lon2,Destino,_,_,_),
81      P is 0.017453292519943295,
82      A is (0.5 - cos((Lat2 - Lat1) * P) / 2 + cos(Lat1 * P) * cos(
        Lat2 * P) * (1 - cos((Lon2 - Lon1) * P)) / 2),
83      Dis is (12742 * asin(sqrt(A))).
84
85
86  seleciona(E, [E|Xs], Xs).
87  seleciona(E, [X|Xs], [X|Ys]) :- seleciona(E, Xs, Ys).
88
89  %--------------------------------------
90
91
92
93
94
95  %------------------------------Procura  n  o  informada
        ------------------
96
97  %% --------------------------Depth first  n   arcos
        ---------------------------
98
99  resolveDP([Nodo|Caminho]):-
100     inicial(Nodo),
101     primeiroprofundidade(Nodo,[Nodo],Caminho).
102
103 primeiroprofundidade(Nodo,_, []):-
104     final(Nodo).
105
106 primeiroprofundidade(Nodo, Historico, [NodoProx|Caminho]):-
107     adjacente(Nodo, NodoProx,_),
108     nao(membro(NodoProx, Historico)),
109     primeiroprofundidade(NodoProx, [NodoProx|Historico], Caminho).
110
111
112 ppTodasSolucoes(L):- findall((S,C), (resolveDP(S), length(S,C)),L).
113 % ppTodasSolucoes(L):- findall((S,C), (resolveDP(S), length(S,C)),L)
114 %                     ,length(L,Tam)
115 %                     ,write(Tam).
116
117
118
119 menorNArcosDP(Nodo,Cam,NA):- findall((Ca,N), (resolveDP(Ca), length(
        Ca,N)),L),minimo(L,(Cam,NA)).
120 maiorNARCOSDP(Nodo,Cam,NA):- findall((Ca,N), (resolveDP(Ca), length(
        Ca,N)),L),maximo(L,(Cam,NA)).
121
122 %%----------------------Depth First custo/distancia
```

```
    ------------------------

resolveDPC([Nodo|Caminho], Custo):-
    inicial(Nodo),
    depthFirstC(Nodo,[Nodo], Caminho, Custo).

depthFirstC(Nodo, _, [], 0):-
    final(Nodo).

depthFirstC(Nodo, Historico, [NodoProx|Caminho], Custo):-
    adjacente(Nodo, NodoProx, CustoArco),
    nao(membro(NodoProx, Historico)),
    depthFirstC(NodoProx, [NodoProx|Historico], Caminho, CustoAux),
    Custo is CustoArco + CustoAux.


melhor(Nodo,Cam,Custo):- findall((Ca,Cus), (resolveDPC(Ca,Cus)), L),
    minimo(L,(Cam,Custo)).
pior(Nodo,Cam,Custo):- findall((Ca,Cus), (resolveDPC(Ca,Cus)), L),
    maximo(L,(Cam,Custo)).


%----------------------------Depth First Produtividade
    -------------------------

resolveDPProdutividade([Nodo|Caminho], Custo):-
    inicial(Nodo),
    depthFirstP(Nodo,[Nodo], Caminho, Custo).

depthFirstP(Nodo, _, [], 0):-
    final(Nodo).

depthFirstP(Nodo, Historico, [NodoProx|Caminho], Custo):-
    adjacente(Nodo, NodoProx,_),
    calculaP(NodoProx, CustoArco),
    nao(membro(NodoProx, Historico)),
    depthFirstP(NodoProx, [NodoProx|Historico], Caminho, CustoAux),
    Custo is CustoArco + CustoAux.

totalLixo([],0).
totalLixo([X|T],Tx):- totalLixo(T,Ty), Tx is X + Ty.

calculaP(Nodo,Total) :- findall(C,pontos_recolha(_,_,Nodo,_,C,_),R),
    totalLixo(R,Total).

maisLixo(Nodo,Cam,Custo):- findall((Ca,Cus), (resolveDPProdutividade
    (Ca,Cus)), L),maximo(L,(Cam,Custo)).
```

```
165
166
167 %----------------------Depth First Tipos
    ----------------------------
168
169 menorNArcosDPT(Tipo,Cam,NA):- findall((Ca,N), (resolveDPTipos(Ca,
    Tipo), length(Ca,N)),L),minimo(L,(Cam,NA)).
170 maiorNARCOSDPT(Tipo,Cam,NA):- findall((Ca,N), (resolveDPTipos(Ca,
    Tipo), length(Ca,N)),L),maximo(L,(Cam,NA)).
171
172
173
174 resolveDPTipos([Nodo|Caminho], Tipo):-
175     inicial(Nodo),
176     depthFirstT(Nodo,[Nodo],Caminho, Tipo).
177
178 depthFirstT(Nodo,_,[], Tipo):-
179     final(Nodo).
180
181 depthFirstT(Nodo, Historico, [NodoProx|Caminho], Tipo):-
182     getLixo(NodoProx, Tipo),
183     adjacente(Nodo, NodoProx,_),
184     nao(membro(NodoProx, Historico)),
185     depthFirstT(NodoProx, [NodoProx|Historico], Caminho, Tipo).
186
187 resolveDPTiposAll(L,T):- findall((S,C), (resolveDPTipos(S, T),
    length(S,C)),L).
188
189
190
191
192
193 %%----------------------Depth first limitada
    -------------------------
194
195
196 menorNArcosDPlimitada(Limite,Cam,NA):- findall((Ca,N), (
    resolveDPlimitada(Ca,Limite), length(Ca,N)),L),minimo(L,(Cam,NA))
    .
197 maiorNArcosDPlimitada(Limite,Cam,NA):- findall((Ca,N), (
    resolveDPlimitada(Ca,Limite), length(Ca,N)),L),maximo(L,(Cam,NA))
    .
198
199 resolveDPlimitadaAll(L,Size):- findall((S,C), (resolveDPlimitada(S,
    Size), length(S,C)), L).
200
201 resolveDPlimitada(Solucao,L) :-
202     inicial(No),
```

```
203     depthFirstLimited([],No,Sol,L),
204     reverseL(Sol,Solucao).
205
206 depthFirstLimited(Caminho,No,[No|Caminho],_) :-
207     final(No),!.
208
209 depthFirstLimited(Caminho,No,S,L) :-
210     L > 0,
211     adjacente(No,No1,_),
212     \+ membro(No1,Caminho),
213     L1 is L - 1,
214     depthFirstLimited([No|Caminho],No1,S,L1).
215
216
217
218 %%--------------------Depth first limitada custo ------------------
219
220
221 melhorDPlimitada(Limite,Cam,Custo):- findall((Ca,Cus), (
       resolveDPlimitadaC(Ca,Limite,Cus)),L),minimo(L,(Cam,Custo)).
222 piorDPlimitada(Limite,Cam,Custo):- findall((Ca,Cus), (
       resolveDPlimitadaC(Ca,Limite,Cus)),L),maximo(L,(Cam,Custo)).
223
224 resolveDPlimitadaCostAll(L,Size) :- findall((S,C,Cost), (
       resolveDPlimitadaC(S,Size,Cost), comprimento(S,C)), L).
225
226 resolveDPlimitadaC(Solucao,L,Custo) :-
227     inicial(No),
228     depthFirstLimitedC([],No,Sol,L,Custo),
229     reverseL(Sol,Solucao).
230
231
232
233 depthFirstLimitedC(Caminho,No,[No|Caminho],L,0) :-
234     final(No),!.
235
236 depthFirstLimitedC(Caminho,No,S,L, Custo) :-
237     L > 0,
238     adjacente(No,No1,CustoArco),
239     \+ membro(No1,Caminho),
240     L1 is L - 1,
241     depthFirstLimitedC([No|Caminho],No1,S,L1, CustoAux),
242     Custo is CustoArco + CustoAux.
243
244
245
246 %-----------------------Depth First Limitada Tipos
       ----------------------------
```

```
247
248
249
250 resolveDPlimitadaTipo(Solucao,L, Tipo) :-
251     inicial(No),
252     depthFirstLimitedT([],No,Sol,L,Tipo),
253     reverseL(Sol,Solucao).
254
255 depthFirstLimitedT(Caminho,No,[No|Caminho],L, Tipo) :-
256     final(No),!.
257
258 depthFirstLimitedT(Caminho,No,S,L,Tipo) :-
259     L > 0,
260     getLixo(No1, Tipo),
261     adjacente(No,No1,_),
262     \+ membro(No1,Caminho),
263     L1 is L - 1,
264     depthFirstLimitedT([No|Caminho],No1,S,L1,Tipo).
265
266
267
268
269 %%----------------Breadth First
    ---------------------------------------
270 resolveBFSall(L) :- findall((S,C), (resolveBFS(S,C)), L).
271
272 %menorNodosBFS(Cam,NA):- findall((A,S,N), (resolveBFS(A,S), length(S
    ,N)),L) ,minimo(L,(Cam,NA)).
273 %menorNArcosDP(Nodo,Cam,NA):- findall((Ca,N), (resolveDP(Ca), length
    (Ca,N)),L),minimo(L,(Cam,NA)).
274
275 resolveBFS(No,Solucao) :-
276     inicial(No),
277     breadthFirst([[No]],Sol),
278     reverseL(Sol,Solucao).
279
280
281 breadthFirst([[No|Caminho]|_],[No|Caminho]) :-
282     final(No).
283
284 breadthFirst([Caminho|Caminhos],Solucao) :-
285     estender(Caminho,NovosCaminhos),
286     append(Caminhos,NovosCaminhos,Caminhos1),
287     breadthFirst(Caminhos1,Solucao).
288
289 estender([No|Caminho],NovosCaminhos) :-
290     findall([NovoNo,No|Caminho],
291         (adjacente(No,NovoNo,_), \+ membro(NovoNo,[No|Caminho])),
```

```
292         NovosCaminhos).
293
294
295
296
297
298 %%-----------------Breadth First Custos
    ------------------------------------
299
300 melhorBFS(Nodo,Cam,Custo):- findall((Ca,Cus), (resolveBFSC(Ca,Cus)),
    L),minimo(L,(Cam,Custo)).
301
302
303 resolveBFSCustosAll(L):- findall((S,C), (resolveBFSC(S,C)),L).
304
305 resolveBFSC(Solucao, Custo) :-
306     inicial(No),
307     resolveBFSC2([[No]],Sol),
308     reverseL(Sol,Solucao),
309     custoTotal(Solucao, Custo).
310
311 resolveBFSC2([[No|Caminho]|_],[No|Caminho]) :-
312     final(No).
313
314 resolveBFSC2([Caminho|Caminhos],Solucao) :-
315     estender(Caminho,NovosCaminhos),
316     append(Caminhos,NovosCaminhos,Caminhos1),
317     resolveBFSC2(Caminhos1,Solucao).
318
319
320 custoTotal([],0).
321 custoTotal([No],0).
322 custoTotal([No1,No2|Caminho],Custo) :-
323     adjacente(No1,No2,CustoArco),
324     custoTotal([No2|Caminho],CustoResto),
325     Custo is CustoArco + CustoResto.
326
327
328
329 %%------------------- Procura informada
    ---------------------------------
330
331 %%------------------Procura gulosa -------------------------------
332
333
334 resolveGulosa(Nodo, Caminho/Custo) :-
335     estimativa(Nodo, Estimativa),
336     agulosa([[Nodo]/0/Estimativa], CaminhoInverso/Custo/_),
```

```
337     reverseL(CaminhoInverso, Caminho).
338
339 agulosa(Caminhos, Caminho) :-
340     obtem_melhor_g(Caminhos, Caminho),
341     Caminho = [Nodo|_]/_/_,final(Nodo).
342
343 agulosa(Caminhos, SolucaoCaminho) :-
344     obtem_melhor_g(Caminhos, MelhorCaminho),
345     seleciona(MelhorCaminho, Caminhos, OutrosCaminhos),
346     expandeGulosa(MelhorCaminho, ExpCaminhos),
347     append(OutrosCaminhos, ExpCaminhos, NovoCaminhos),
348     agulosa(NovoCaminhos, SolucaoCaminho).
349
350 obtem_melhor_g([Caminho], Caminho) :- !.
351
352 obtem_melhor_g([Caminho1/Custo1/Est1,_/Custo2/Est2|Caminhos],
      MelhorCaminho) :-
353   Est1 =< Est2, !,
354   obtem_melhor_g([Caminho1/Custo1/Est1|Caminhos], MelhorCaminho).
355
356 obtem_melhor_g([_|Caminhos], MelhorCaminho) :-
357   obtem_melhor_g(Caminhos, MelhorCaminho).
358
359 expandeGulosa(Caminho, ExpCaminhos) :-
360   findall(NovoCaminho, adjacenteG(Caminho,NovoCaminho), ExpCaminhos)
      .
361
362 adjacenteG([Nodo|Caminho]/Custo/_, [ProxNodo,Nodo|Caminho]/NovoCusto
      /Est) :-
363   adjacente(Nodo, ProxNodo, PassoCusto),
364     nao(membro(ProxNodo, Caminho)),
365   NovoCusto is Custo + PassoCusto,
366   estimativa(ProxNodo, Est).
367
368
369
370 %%-------------------------Gulosa Tipos
      ----------------------------------
371
372 resolveGulosaT(Nodo,Caminho/Custo,Tipo):-
373     estimativa(Nodo,Estimativa),
374     agulosaT([[Nodo]/0/Estimativa],CaminhoInverso/Custo/_,Tipo),
375     reverse(CaminhoInverso,Caminho).
376
377 agulosaT(Caminhos,Caminho,Tipo):-
378     obtem_melhor_g(Caminhos,Caminho),
379     Caminho = [Nodo|_]/_/_,final(Nodo).
380
```

```
381 agulosaT(Caminhos,SolucaoCaminho,Tipo):-
382     obtem_melhor_g(Caminhos, MelhorCaminho),
383     seleciona(MelhorCaminho, Caminhos, OutrosCaminhos),
384     expandeGulosaT(MelhorCaminho,ExpCaminhos,Tipo),
385     append(OutrosCaminhos,ExpCaminhos,NovoCaminhos),
386     agulosaT(NovoCaminhos,SolucaoCaminho,Tipo).
387
388 expandeGulosaT(Caminho,ExpCaminhos,Tipo):-
389     findall(NovoCaminho,adjacenteGT(Caminho,NovoCaminho,Tipo),
    ExpCaminhos).
390
391 adjacenteGT([Nodo|Caminho]/Custo/_,[ProxNodo,Nodo|Caminho]/NovoCusto
    /Est,Tipo):-
392     getLixo(NodoProx,Tipo),
393     adjacente(Nodo,ProxNodo,PassoCusto),
394     nao(membro(ProxNodo,Caminho)),
395     NovoCusto is Custo+PassoCusto,
396     estimativa(ProxNodo,Est).
397
398
399
400
401
402 %----------------------------A*
    ------------------------------------------
403
404 resolveAEstrela(Caminho/Custo) :-
405     inicial(Nodo),
406     pontos_recolha(_,_,Nodo,_,Cap,_),
407     aestrela([[Nodo]/0/pontos_recolha], CaminhoInverso/Custo/_),
408     reverseL(CaminhoInverso, Caminho).
409
410 aestrela(Caminhos, Caminho) :-
411     obtem_melhor(Caminhos, Caminho),
412     Caminho = [Nodo|_]/_/_,
413     final(Nodo).
414
415 aestrela(Caminhos, SolucaoCaminho) :-
416     obtem_melhor(Caminhos, MelhorCaminho),
417     seleciona(MelhorCaminho, Caminhos, OutrosCaminhos),
418     expandeAEstrela(MelhorCaminho, ExpCaminhos),
419     append(OutrosCaminhos, ExpCaminhos, NovoCaminhos),
420     aestrela(NovoCaminhos, SolucaoCaminho).
421
422 obtem_melhor([Caminho], Caminho) :- !.
423
424 obtem_melhor([Caminho1/Custo1/Est1,_/Custo2/Est2|Caminhos],
    MelhorCaminho) :-
```

```
425
426     Custo1 + Est1 =< Custo2 + Est2, !,
427     obtem_melhor([Caminho1/Custo1/Est1|Caminhos], MelhorCaminho).
428
429 obtem_melhor([_|Caminhos], MelhorCaminho) :-
430     obtem_melhor(Caminhos, MelhorCaminho).
431
432 expandeAEstrela(Caminho, ExpCaminhos) :-
433 findall(NovoCaminho, adjacenteG(Caminho,NovoCaminho), ExpCaminhos).
```