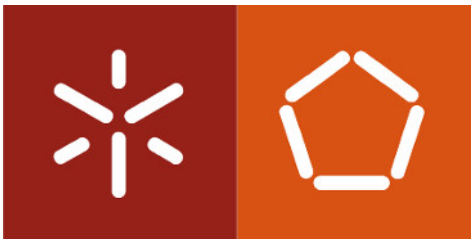

Sistemas de Representação de Conhecimento e Raciocínio

Trabalho realizado por:

Paulo Silva Sousa

A89465 Paulo Sousa

Projeto SRCR
2020/2021
Universidade do Minho

Índice

1 Introdução

Para este trabalho prático, foi nos imposto que tinhamos de desenvolver um sistema em que havia ligações entre pontos de recolha de resíduos pela cidade de Lisboa.
Assim, a partir do dataset, destacamos as informações relativas a cada ponto de recolha e a cada arco que liga dois pontos e, de seguida, implementei cada algoritmo de procura utilizando a ferramenta Prolog.
Por último, escolhi focar-me e desenvolver a versão simplicada.

2 Descrição do Trabalho

Para esta etapa, utilizei a linguagem Python de modo a fazer o parse dos dados que nos foram fornecidos, uma vez que considero uma linguagem acessível no que toca a trabalhar com dados.
Posto isto, criei o *pontos.pl* e o *arcos.pl*. O primeiro consiste na representação de cada ponto. tendo em conta a localização. latitude, longitude, as ligações adjacentes. o tipo e a capacidade. O segundo ficheiro apresenta as informações importantes para cada arco, como um primeiro ponto, um segundo ponto e a distância entre estes.

```
%%arco(Ponto1,Ponto2,Distancia)
arco('R do Alecrim','Pc Duque da Terceira',9066.173630069856).
arco('R Corpo Santo','Lg Corpo Santo',9065.009277711006).
arco('R Corpo Santo','Tv Corpo Santo',9067.01656012462).
arco('Tv Corpo Santo','R Bernardino da Costa',9063.568956897556).
arco('Tv Corpo Santo','Cais do Sodre',9064.603054546998).
```

Figura 1: Exemplo de arcos

```
%%ponto(Localizacao,Latitude,Longitude,[Ligacoes],Tipo,Capacidade)
ponto('R do Alecrim',-9.14330880914792,38.7080787857025,['R Ferragial','R Ataide','Pc Duque da Terceira'],'Lixos',90).
ponto('R Corpo Santo',-9.14255098678099,38.7073286838222,['Lg Corpo Santo','Tv Corpo Santo'],'Lixos',280).
ponto('Tv Corpo Santo',-9.14276596081499,38.7070836135523,['R Corpo Santo','R Bernardino da Costa','Cais do Sodre'],'Lixos',280).
ponto('R Bernardino da Costa',-9.14240835587344,38.7069966274245,['Lg Corpo Santo','Tv Corpo Santo','Pc Duque da Terceira'],'Lixos',420).
ponto('Lg Conde',-9.1512511235953,38.7087754470349,[],'Lixos',2400).
```

Figura 2: Exemplo de pontos

2.1 Tratamento de Dados

Considerei necessário haver um tratamento de dados depois de concluida a leitura do parser, de modo a não termos problemas futuros. Deste modo, temos:

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

Figura 3: Tratamento de Dados

3 Queries

3.1 Querie 2

```
% Querie 2
maisPontosRecolha(Tipo, L) :- dfsTipoTotal(R, Tipo), maximo(R,L).
```

Figura 4: Excerto da querie 2

```
?- maisPontosRecolha("Lixos",L).
L =  (['Tv Ribeira Nova', 'R Nova do Carvalho', 'Tv Corpo Santo', 'R Bernardino
da Costa', 'Pc Duque da Terceira', 'Av 24 de Julho'], 5) .
```

Figura 5: Output da querie 2

Para a querie 2, o objetivo é encontrar os circuitos que têm mais pontos de recolha. Assim, recorri à depth para conseguir os caminhos em que está aquele tipo de resíduos e o nº de arcos. Posteriormente, através da função maximo,vi qual é o maior nº de arcos.

3.2 Querie 3

```
% Querie 3
maisProdutividade(Tipo, L) :- dfsQuantidadeTotal(R, Tipo), maximo(R,L).
```

Figura 6: Excerto da querie 3

```
?- maisProdutividade("Lixos",L).
L =  (['Tv Ribeira Nova', 'R Nova do Carvalho', 'Tv Corpo Santo', 'Cais do Sodre
', 'Pc Duque da Terceira', 'Av 24 de Julho'], 2670) .
```

Figura 7: Output da querie 3

Para a querie 3, o objetivo é comparar os circuitos de recolha a nível da produtividade. Assim, vi quais os caminhos que possuem mais resíduos e, com a função maximo, escolhi o que possui maior quantidade entre todos.

3.3 Querie 4

```
% Querie 4
circuitoMaisRapido(L) :- bfsCustoTotal(R), minimo(R,L).
```

Figura 8: Excerto da querie 4

```
?- circuitoMaisRapido(L).
L =  (['Tv Ribeira Nova', 'R Nova do Carvalho', 'Tv dos Remolares', 'Av 24 de Ju
lho'], 27173.85934403949) .
```

Figura 9: Output da querie 4

Para a querie 4, o objetivo é escolher o circuito mais rápido. Para isso, utilizei a breadth first e depois a função minimo para, no fim, ter o circuito que possuía menor número de arcos.

3.4 Querie 5

```
% Querie 5
circuitoMaisEficiente(L) :- dfsArcosTotal(R), minimo(R, L).
```

Figura 10: Excerto da querie 5

```
?- circuitoMaisEficiente(L).
L =  (['Tv Ribeira Nova', 'R Nova do Carvalho', 'Tv dos Remolares', 'Av 24 de Ju
lho'], 3) .
```

Figura 11: Output da querie 5

Para esta querie, o objetivo é escolher o circuito mais eficiente. Assim, utilizei a depth first e a função minimo de modo a ter o circuito que passasse por menos arcos.

4 Algoritmos de Pesquisa

4.1 DFS -Depth-First Search

O algoritmo de busca em profundidade é útil para percorrer grafos. Começa-se pelo nó raiz e depois vai para cada um dos ramos até chegar ao máximo e ter de retroceder.

```
?- dfsArcosTotal(L).
L = [(['Tv Ribeira Nova', 'R Nova do Carvalho', 'Tv dos Remolares', 'Av 24 de Ju
lho'], 3),  (['Tv Ribeira Nova', 'R Nova do Carvalho', 'R do Alecrim', 'Pc Duque
 da Terceira', 'Av 24 de Julho'], 4),  (['Tv Ribeira Nova', 'R Nova do Carvalho'
, 'Tv Corpo Santo', 'R Bernardino da Costa', 'Pc Duque da Terceira'|...], 5),  (
['Tv Ribeira Nova', 'R Nova do Carvalho', 'Tv Corpo Santo', 'Cais do Sodre'|...]
, 4),  (['Tv Ribeira Nova', 'R Nova do Carvalho', 'Tv Corpo Santo'|...], 5)].
```

Figura 12: Output da depth-first para número de arcos

```
?- dfsCustoTotal(L).
L = [(['Tv Ribeira Nova', 'R Nova do Carvalho', 'Tv dos Remolares', 'Av 24 de Ju
lho'], 27173.859344039498),  (['Tv Ribeira Nova', 'R Nova do Carvalho', 'R do Al
ecrim', 'Pc Duque da Terceira', 'Av 24 de Julho'], 36246.32410901808),  (['Tv Ri
beira Nova', 'R Nova do Carvalho', 'Tv Corpo Santo', 'R Bernardino da Costa', 'P
c Duque da Terceira'|...], 45306.6047025138),  (['Tv Ribeira Nova', 'R Nova do C
arvalho', 'Tv Corpo Santo', 'Cais do Sodre'|...], 36233.38812574846),  (['Tv Rib
eira Nova', 'R Nova do Carvalho', 'Tv Corpo Santo'|...], 45292.20332357356)].
```

Figura 13: Output da depth-first para custo

```
?- dfsTipoTotal(L,"Lixos").
L = [(['Tv Ribeira Nova', 'R Nova do Carvalho', 'R do Alecrim', 'Pc Duque da Ter
ceira', 'Av 24 de Julho'], 4),  (['Tv Ribeira Nova', 'R Nova do Carvalho', 'R do
 Alecrim', 'Pc Duque da Terceira', 'Av 24 de Julho'], 4),  (['Tv Ribeira Nova',
'R Nova do Carvalho', 'Tv Corpo Santo', 'R Bernardino da Costa', 'Pc Duque da Te
rceira'|...], 5),  (['Tv Ribeira Nova', 'R Nova do Carvalho', 'Tv Corpo Santo',
'R Bernardino da Costa'|...], 5),  (['Tv Ribeira Nova', 'R Nova do Carvalho', 'T
v Corpo Santo'|...], 5),  (['Tv Ribeira Nova', 'R Nova do Carvalho'|...], 4),  (
['Tv Ribeira Nova'|...], 5),  ([...|...], 5),  (..., ...)|...].
```

Figura 14: Output da depth-first para tipo de lixo

```
?- dfsQuantidadeTotal(L,"Lixos").
L = [(['Tv Ribeira Nova', 'R Nova do Carvalho', 'R do Alecrim', 'Pc Duque da Ter
ceira', 'Av 24 de Julho'], 780),  (['Tv Ribeira Nova', 'R Nova do Carvalho', 'R
do Alecrim', 'Pc Duque da Terceira', 'Av 24 de Julho'], 540),  (['Tv Ribeira Nov
a', 'R Nova do Carvalho', 'R do Alecrim', 'Pc Duque da Terceira', 'Av 24 de Julh
o'], 780),  (['Tv Ribeira Nova', 'R Nova do Carvalho', 'R do Alecrim', 'Pc Duque
 da Terceira'|...], 540),  (['Tv Ribeira Nova', 'R Nova do Carvalho', 'R do Alec
rim'|...], 2130),  (['Tv Ribeira Nova', 'R Nova do Carvalho'|...], 1890),  (['Tv
 Ribeira Nova'|...], 2130),  ([...|...], 1890),  (..., ...)|...].
```

Figura 15: Output da depth-first para quantidade de lixo transportado

4.2 BFS -Breadth-First Search

Este algoritmo de busca em largura também tem como objetivo realizar travessias em grafos. Para além disso, em oposição à depth first, é feita de uma forma muito diferente uma vez que começa no nodo raíz e vai percorrendo por níveis. No fim de cada nível, segue para o próximo.

```
?- bfsArcosTotal(L).
L = [(['Tv Ribeira Nova', 'R Nova do Carvalho', 'Tv dos Remolares', 'Av 24 de Ju
lho'], 3),  (['Tv Ribeira Nova', 'R Nova do Carvalho', 'R do Alecrim', 'Pc Duque
 da Terceira', 'Av 24 de Julho'], 4),  (['Tv Ribeira Nova', 'R Nova do Carvalho'
, 'Tv Corpo Santo', 'Cais do Sodre', 'Av 24 de Julho'], 4),  (['Tv Ribeira Nova'
, 'R Nova do Carvalho', 'Tv Corpo Santo', 'R Bernardino da Costa'|...], 5),  (['
Tv Ribeira Nova', 'R Nova do Carvalho', 'Tv Corpo Santo'|...], 5)].
```

Figura 16: Output da breadth-first para arcos

```
?- bfsCustoTotal(L).
L = [(['Tv Ribeira Nova', 'R Nova do Carvalho', 'Tv dos Remolares', 'Av 24 de Ju
lho'], 27173.85934403949),  (['Tv Ribeira Nova', 'R Nova do Carvalho', 'R do Ale
crim', 'Pc Duque da Terceira', 'Av 24 de Julho'], 36246.324109018075),  (['Tv Ri
beira Nova', 'R Nova do Carvalho', 'Tv Corpo Santo', 'Cais do Sodre', 'Av 24 de
Julho'], 36233.38812574846),  (['Tv Ribeira Nova', 'R Nova do Carvalho', 'Tv Cor
po Santo', 'R Bernardino da Costa'|...], 45306.604702513796),  (['Tv Ribeira Nov
a', 'R Nova do Carvalho', 'Tv Corpo Santo'|...], 45292.20332357355)].
```

Figura 17: Output da breadth-first para custo

4.3 Busca Iterativa Limitada em Profundidade

Já este algoritmo tem como base algumas partes do depth first, no entanto, por oposição, este possui um número máximo de iterações.

```
?- dfsArcosTotalLimit(3,L).
L = [(['Tv Ribeira Nova', 'R Nova do Carvalho', 'Tv dos Remolares', 'Av 24 de Ju
lho'], 3)].
```

Figura 18: Output da depth-first limitado para arcos

```
?- dfsCustoTotalLimit(30000,L).
L = [(['Tv Ribeira Nova', 'R Nova do Carvalho', 'Tv dos Remolares', 'Av 24 de Ju
lho'], 27173.859344039498)].
```

Figura 19: Output da depth-first limitado para custo

```
?- dfsTipoTotalLimit(L,"Lixos",10).
L = [['Tv Ribeira Nova', 'R Nova do Carvalho', 'R do Alecrim', 'Pc Duque da Terceira',
 'Av 24 de Julho'], ['Tv Ribeira Nova', 'R Nova do Carvalho', 'R do Alecrim', 'Pc Duqu
e da Terceira', 'Av 24 de Julho'], ['Tv Ribeira Nova', 'R Nova do Carvalho', 'Tv Corpo
 Santo', 'R Bernardino da Costa', 'Pc Duque da Terceira', 'Av 24 de Julho'], ['Tv Ribe
ira Nova', 'R Nova do Carvalho', 'Tv Corpo Santo', 'R Bernardino da Costa', 'Pc Duque
da Terceira'|...], ['Tv Ribeira Nova', 'R Nova do Carvalho', 'Tv Corpo Santo', 'Cais d
o Sodre'|...], ['Tv Ribeira Nova', 'R Nova do Carvalho', 'Tv Corpo Santo'|...], ['Tv R
ibeira Nova', 'R Nova do Carvalho'|...], ['Tv Ribeira Nova'|...], [...|...]|...].
```

Figura 20: Output da depth-first limitado para tipo

4.4 Gulosa

O algoritmo guloso, de modo a fazer a travessia, escolhe o próximo nodo a ir mediante o que achar uma melhor opção.

```
?- resolveGulosa(L).
L = ['Tv Ribeira Nova', 'R Nova do Carvalho', 'Tv dos Remolares', 'Av 24 de Julh
o']/27173.85934403949 .
```

Figura 21: Output da gulosa

4.5 A* (Aestrela)

O algoritmo AEstrela é um algoritmo que corre em busca de um caminho de um vértice inicial até um final. Concluindo, é uma fusão dos algoritmos Breadth first e Dijkstra.

```
?- resolveGulosa(L).
L = ['Tv Ribeira Nova', 'R Nova do Carvalho', 'Tv dos Remolares', 'Av 24 de Julh
o']/27173.85934403949 .
```

Figura 22: Output da gulosa

5 Resultados

Decidi criar esta tabela de resultados de modo a analisar da melhor forma vários aspetos como, por exemplo, tempo de execução (s), espaço, Profundidade/Custo e se encontrou a melhor solução ou não.

Estratégia	Tempo (s)	Espaço	Profundidade/Custo	Encontrou a melhor solução?
DFS	0.000s	O(bm)	4/27.173km	Sim
Limitada	0.000s	O(bˆm)	4/27.173km	Sim
BFS	0.000s	O(bM)	4/27.173km	Sim
Gulosa	0.001s	-	4/27.173km	Possivelmente
A* (A estrela)	0.002s	-	4/27.173km	Sim

Tabela 1: Resultados

```
?- time(dfsArcosTotal(L)).
% 204 inferences, 0.000 CPU in 0.000 seconds (91% CPU, 1606299 Lips)
```

Figura 23: Tempo Depth-first

```
?- time(bfsArcosTotal(L)).
% 495 inferences, 0.000 CPU in 0.000 seconds (93% CPU, 2142857 Lips)
```

Figura 24: Tempo Breadth-first

```
?- time(dfsArcosTotalLimit(3,L)).
% 101 inferences, 0.000 CPU in 0.000 seconds (89% CPU, 1603175 Lips)
```

Figura 25: Tempo Limitada em profundidade

```
?- time(resolveGulosa(L)).
% 262 inferences, 0.000 CPU in 0.001 seconds (36% CPU, 791541 Lips)
```

Figura 26: Tempo Gulosa

```
?- time(resolveAEstrela(L)).
% 2,512 inferences, 0.002 CPU in 0.002 seconds (97% CPU, 1444508 Lips)
```

Figura 27: Tempo A Estrela

6 Conclusão

Começando pelos resultados que consegui obter, considero que o algoritmo Depth First é muito eficiente, responde sempre ao que é requisitado e tem tempos de execução muito bons. Para além disso, faz procuras por quantidade de resíduos, arcos e por tipos de resíduos.
Em segundo lugar, penso que este projeto foi muito interessante para me dedicar à UC de Sistemas de Representação de Conhecimento e Raciocínio. Para além disso, exigiu muito raciocínio e concentração de modo a que conseguisse atingir todos os objetivos alcançados.
Por fim, foi muito útil também para trabalhar melhor com Prolog e creio que irá ser muito ímportante para o futuro.

A Parser

```
# res -> (tipo: rua : (latitude, longitude, [ligacoes]))

import pandas as pd
from math import sin, cos, sqrt, atan2
import re
import sys

sys.setrecursionlimit(10000)

dataset = pd.read_excel(r'dataset.xlsx')
res = dict()
res['Geral'] = dict()

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

recolhaRE = re.compile(r'[0-9]+: ([a-zA-Z0-9, ]+)(\(.+\))?')

for i in range(0, dataset['OBJECTID'].size - 1):
    tipo = dataset["CONTENTOR_RESÍDUO"][i]

    if tipo not in res:
        res[tipo] = dict()

    recolha = re.search(recolhaRE, dataset["PONTO_RECOLHA_LOCAL"][i]).groups()
    rua = recolha[0].strip()

    if rua not in res[tipo]:
        res[tipo][rua] = [None, None, list(), None]
        res[tipo][rua][0] = dataset["Latitude"][i]
        res[tipo][rua][1] = dataset["Longitude"][i]
```

```
            res[tipo][rua][3] = dataset["CONTENTOR_TOTAL_LITROS"][i]

        if rua not in res['Geral']:
            res['Geral'][rua] = [None, None, list(), None]
            res['Geral'][rua][0] = dataset["Latitude"][i]
            res['Geral'][rua][1] = dataset["Longitude"][i]
            res['Geral'][rua][3] = dataset["CONTENTOR_TOTAL_LITROS"][i]

        if recolha[1]:
            lixo = re.search(r':(.+)\)', recolha[1]).groups()
            ligacoes = re.split(r' - ', lixo[0].strip())

            for l in ligacoes:
                newl = re.match(r'[a-zA-Z0-9, ]+', l).group(0).strip()

                if newl not in res[tipo][rua][2]:
                    res[tipo][rua][2].append(newl)

                if newl not in res['Geral'][rua][2]:
                    res['Geral'][rua][2].append(newl)

# Pontos

fp = open("pontos.pl", "w")

fp.write('%%ponto(Localizacao,Latitude,Longitude,[Ligacoes],Tipo,Capacidade)\n')

for tipo, infor in res.items():
    if tipo != 'Geral':
        for rua, info in infor.items():
            ligacoes = '[]'

            if info[2]:
                ligacoes = '['
                for i in range(0,len(info[2])-1):
                    ligacoes += '\'' + info[2][i] + '\','
                ligacoes += '\'' + info[2][-1] + '\']'

            fp.write('ponto(' + '\'' + rua + '\','
                              + str(info[0]) + ','
                              + str(info[1]) + ','
                              + ligacoes + ','
                              + '\'' + tipo + '\','
                              + str(info[3]) + ').\n')

fp.close()

# Arcos

fa = open("arcos.pl", "w")
```

```
def calculaDistancia(lat1, lat2, lon1, lon2):
    R = 6373.0

    dlon = lon2 - lon1
    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

def inArcos(rua, l):
    for (rua1, rua2) in arcos:
        if (rua == rua1 and l == rua2) or (rua == rua2 and l == rua1):
            return True

    return False

fa.write('%%arco(Ponto1,Ponto2,Distancia)\n')

arcos = list()

for rua, info in res['Geral'].items():
    if info[2]:
        for l in info[2]:
            if l in res['Geral'] and not inArcos(rua, l):
                arcos.append((rua, l))
                distancia = calculaDistancia(
                    res['Geral'][rua][0], res['Geral'][rua][1],
                    res['Geral'][l][0], res['Geral'][l][1])
                fa.write('arco(\'' + rua + '\',' + '\'' +
                         l + '\',' + str(distancia) + ').\n')

fa.close()
```

B Código

```
:- style_check(-discontiguous).
:- style_check(-singleton).

:- set_prolog_flag(discontiguous_warnings,off).
:- set_prolog_flag(single_var_warnings,off).

:- op( 900,xfy,'::' ).
:- use_module(library(lists)).

:- include('pontos.pl').
:- include('arcos.pl').

%-------------------------------- - - - - - - - - - - - - - - - - -
% DADOS
%-------------------------------- - - - - - - - - - - - - - - - - -

inicio('Tv Ribeira Nova').
fim('Av 24 de Julho').

%-------------------------------- - - - - - - - - - - - - - - - - -
% QUERIES
%-------------------------------- - - - - - - - - - - - - - - - - -

% Querie 2
maisPontosRecolha(Tipo, L) :- dfsTipoTotal(R, Tipo), maximo(R,L).

% Querie 3
maisProdutividade(Tipo, L) :- dfsQuantidadeTotal(R, Tipo), maximo(R,L).

% Querie 4
circuitoMaisRapido(L) :- bfsCustoTotal(R), minimo(R,L).

% Querie 5
circuitoMaisEficiente(L) :- dfsArcosTotal(R), minimo(R, L).

%-------------------------------- - - - - - - - - - - - - - - - - -
% PROCURA NÃO INFORMADA
%-------------------------------- - - - - - - - - - - - - - - - - -

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

% Arcos

dfsArcosTotal(L):- findall((S,C),(dfsArcos(S,C)),L).

dfsArcos([Nodo|Caminho],Custo):-
    inicio(Nodo),
    dfsArcos2(Nodo,[Nodo],Caminho,Custo).
```

```
dfsArcos2(Nodo, _, [], 0):-
    fim(Nodo).

dfsArcos2(Nodo,Historico,[NodoProx|Caminho],Custo):-
    getArco(Nodo, NodoProx, _),
    nao(membro(NodoProx, Historico)),
    dfsArcos2(NodoProx,[NodoProx|Historico],Caminho,Custo2),
    Custo is Custo2 + 1.

% Custos

dfsCustoTotal(L):- findall((S,C),(dfsCusto(S,C)),L).

dfsCusto([Nodo|Caminho],Custo):-
    inicio(Nodo),
    dfsCusto2(Nodo,[Nodo],Caminho,Custo).

dfsCusto2(Nodo, _, [], 0):-
    fim(Nodo).

dfsCusto2(Nodo,Historico,[NodoProx|Caminho],Custo):-
    getArco(Nodo, NodoProx, CustoMovimento),
    nao(membro(NodoProx, Historico)),
    dfsCusto2(NodoProx,[NodoProx|Historico],Caminho,Custo2),
    Custo is CustoMovimento + Custo2.

% Tipo de Lixo

dfsTipoTotal(L,T):- findall((S,C), (dfsTipo(S, C, T)),L).

dfsTipo([Nodo|Caminho], Arcos, Tipo):-
    inicio(Nodo),
    dfsTipo2(Nodo,[Nodo],Caminho, Tipo, Arcos).

dfsTipo2(Nodo,_,[], Tipo, 0):-
    fim(Nodo).

dfsTipo2(Nodo, Historico, [NodoProx|Caminho], Tipo, Arcos):-
    getLixo(NodoProx, Tipo),
    getArco(Nodo, NodoProx,_),
    nao(membro(NodoProx, Historico)),
    dfsTipo2(NodoProx, [NodoProx|Historico], Caminho, Tipo, Arcos2),
    Arcos is Arcos2 + 1.

% Quantidade

dfsQuantidadeTotal(L, T):- findall((S,C),(dfsQuantidade(S,C,T)),L).

dfsQuantidade([Nodo|Caminho],Quantidade,Tipo):-
    inicio(Nodo),
```

```
    dfsQuantidade2(Nodo,[Nodo],Caminho,Quantidade,Tipo).

dfsQuantidade2(Nodo, _, [], 0,Tipo):-
    fim(Nodo).

dfsQuantidade2(Nodo,Historico,[NodoProx|Caminho],Quantidade, Tipo):-
    getLixo(NodoProx, Tipo),
    getArco(Nodo, NodoProx, CustoMovimento),
    getPonto(NodoProx, _, _, _, _, QuantidadeMovimento),
    nao(membro(NodoProx, Historico)),
    dfsQuantidade2(NodoProx,[NodoProx|Historico],Caminho,Quantidade2,Tipo),
    Quantidade is QuantidadeMovimento + Quantidade2.

%--------------------------------- - - - - - - - - - -  -  -  -  -   -
% BREADTH-FIRST SEARCH
%--------------------------------- - - - - - - - - - -  -  -  -  -   -

% Arcos

bfsArcosTotal(L):- findall((S,C),(bfsArcos(S,C)),L).

bfsArcos(Cam,Arcos):-
    inicio(Orig),
    fim(Dest),
    bfsArcos2(Dest,[[Orig]],Cam,Arcos).

bfsArcos2(Dest,[[Dest|T]|_],Cam,Arcos):-
    inversoArcos([Dest|T],Cam,Arcos2),
    Arcos is Arcos2 - 1.

bfsArcos2(Dest,[LA|Outros],Cam,Arcos):-
    LA=[Act|_],
    findall([X|LA],(Dest\==Act,getArco(Act,X,_),nao(membro(X,LA))),Novos),
    append(Outros,Novos,Todos),
    bfsArcos2(Dest,Todos,Cam,Arcos).

inversoArcos(Xs,Ys,Arcos) :- inversoArcos(Xs,[],Ys,Arcos).
inversoArcos([],Xs,Xs,0).
inversoArcos([X|Xs],Ys,Zs,Arcos) :-
    inversoArcos(Xs,[X|Ys],Zs,Arcos2),
    Arcos is Arcos2 + 1.

% Custos

bfsCustoTotal(L):- findall((S,C),(bfsCusto(S,C)),L).

bfsCusto(Cam,Custo):-
    inicio(Orig),
    fim(Dest),
    bfsCusto2(Dest,[[Orig/0]],Cam,Custo).
```

```
bfsCusto2(Dest,[[Dest/C|T]|_],Cam,Custo):-
    inversoCusto([Dest/C|T],Cam,Custo).

bfsCusto2(Dest,[LA|Outros],Cam,Custo):-
    LA=[Act/_|_],
    findall([X/CustoMovimento|LA],(Dest\==Act,getArco(Act,X,CustoMovimento),nao(membro(X/C
    append(Outros,Novos,Todos),
    bfsCusto2(Dest,Todos,Cam,Custo).

inversoCusto(Xs,Ys,Custo) :- inversoCusto(Xs,[],Ys,Custo).
inversoCusto([],Xs,Xs,0).
inversoCusto([X/C|Xs],Ys,Zs,Custo) :-
    inversoCusto(Xs,[X|Ys],Zs,Custo2),
    Custo is Custo2 + C.

%-------------------------------- - - - - - - - - - -  -  -  -  -   -
% BUSCA ITERATIVA LIMITADA EM PROFUNDIDADE
%-------------------------------- - - - - - - - - - -  -  -  -  -   -

% Arcos

dfsArcosTotalLimit(Maxdepth,L):- findall((S,C),(dfsArcosLimit(S,C,Maxdepth)),L).

dfsArcosLimit([Nodo|Caminho],Custo,Maxdepth):-
    inicio(Nodo),
    dfsArcos2Limit(Nodo,[Nodo],Caminho,Custo,Maxdepth).

dfsArcos2Limit(Nodo, _, [], 0,_):-
    fim(Nodo).

dfsArcos2Limit(Nodo,Historico,[NodoProx|Caminho],Custo,Maxdepth):-
    Maxdepth > 0,
    getArco(Nodo, NodoProx, _),
    nao(membro(NodoProx, Historico)),
    Max1 is Maxdepth - 1,
    dfsArcos2Limit(NodoProx,[NodoProx|Historico],Caminho,Custo2,Max1),
    Custo is Custo2 + 1.

% Custos

dfsCustoTotalLimit(Maxcusto, L):- findall((S,C),(dfsCustoLimit(S,C, Maxcusto)),L).

dfsCustoLimit([Nodo|Caminho],Custo, Maxcusto):-
    inicio(Nodo),
    dfsCusto2Limit(Nodo,[Nodo],Caminho,Custo, Maxcusto).

dfsCusto2Limit(Nodo, _, [], 0, _):-
    fim(Nodo).

dfsCusto2Limit(Nodo,Historico,[NodoProx|Caminho],Custo, Maxcusto):-
    getArco(Nodo, NodoProx, CustoMovimento),
```

```
    Max1 is Maxcusto - CustoMovimento,
    Max1 > 0,
    nao(membro(NodoProx, Historico)),
    dfsCusto2Limit(NodoProx,[NodoProx|Historico],Caminho,Custo2,Max1),
    Custo is CustoMovimento + Custo2.

% Tipos

dfsTipoTotalLimit(L,Max,T):- findall(S, (dfsTipoLimit(S, T, Max)),L).

dfsTipoLimit([Nodo|Caminho], Maxdepth, Tipo):-
    inicio(Nodo),
    dfsTipo2Limit(Nodo,[Nodo],Caminho, Tipo, Maxdepth).

dfsTipo2Limit(Nodo,_,[], Tipo, _):-
    fim(Nodo).

dfsTipo2Limit(Nodo, Historico, [NodoProx|Caminho], Tipo, Maxdepth):-
    Maxdepth > 0,
    getLixo(NodoProx, Tipo),
    getArco(Nodo, NodoProx,_),
    nao(membro(NodoProx, Historico)),
    Max1 is Maxdepth - 1,
    dfsTipo2Limit(NodoProx, [NodoProx|Historico], Caminho, Tipo, Max1).

%--------------------------------- - - - - - - - - - - -  -  -  -  -   -
% PROCURA INFORMADA
%--------------------------------- - - - - - - - - - - -  -  -  -  -   -

%--------------------------------- - - - - - - - - - - -  -  -  -  -   -
% A ESTRELA
%--------------------------------- - - - - - - - - - - -  -  -  -  -   -

resolveAEstrela(Caminho/Custo) :-
    inicio(Nodo),
    getPonto(Nodo, _, _, _, _, Capacidade),
    aestrela([[Nodo]/0/Capacidade], CaminhoInverso/Custo/_),
    inverso(CaminhoInverso, Caminho).

aestrela(Caminhos, Caminho) :-
        obtem_melhor(Caminhos, Caminho),
        Caminho = [Nodo|_]/_/_,
    fim(Nodo).

aestrela(Caminhos, SolucaoCaminho) :-
    obtem_melhor(Caminhos, MelhorCaminho),
    seleciona(MelhorCaminho, Caminhos, OutrosCaminhos),
    expandeAEstrela(MelhorCaminho, ExpCaminhos),
    append(OutrosCaminhos, ExpCaminhos, NovoCaminhos),
    aestrela(NovoCaminhos, SolucaoCaminho).
```

```
obtem_melhor([Caminho], Caminho) :- !.

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

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

expandeAEstrela(Caminho, ExpCaminhos) :-
    findall(NovoCaminho, adjacenteG(Caminho,NovoCaminho), ExpCaminhos).

%-------------------------------- - - - - - - - - - - - - - - - - -
% GULOSA
%-------------------------------- - - - - - - - - - - - - - - - - -

resolveGulosa(Caminho/Custo) :-
    inicio(Nodo),
    estimativa(Nodo, Est),
    agulosa([[Nodo]/0/Est], CaminhoInverso/Custo/_),
    inverso(CaminhoInverso, Caminho).

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

agulosa(Caminhos, SolucaoCaminho) :-
    obtem_melhor_g(Caminhos, MelhorCaminho),
    seleciona(MelhorCaminho, Caminhos, OutrosCaminhos),
    expandeGulosa(MelhorCaminho, ExpCaminhos),
    append(OutrosCaminhos, ExpCaminhos, NovoCaminhos),
    agulosa(NovoCaminhos, SolucaoCaminho).

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

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

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

expandeGulosa(Caminho, ExpCaminhos) :-
        findall(NovoCaminho, adjacenteG(Caminho,NovoCaminho), ExpCaminhos).

adjacenteG([Nodo|Caminho]/Custo/_, [ProxNodo,Nodo|Caminho]/NovoCusto/Est) :-
    getArco(Nodo, ProxNodo, PassoCusto),
    nao(member(ProxNodo, Caminho)),
        NovoCusto is Custo + PassoCusto,
    estimativa(ProxNodo, Est).
```

```
estimativa(Local,Est) :-
    ponto(Local,Lat1,Lon1,_,_,_),
    fim(Destino),
    ponto(Destino,Lat2,Lon2,_,_,_),
    R is 6373.0,
    Dlon is Lon2 - Lon1,
    Dlat is Lat2 - Lat1,
    A is sin(Dlat / 2)**2 + cos(Lat1) * cos(Lat2) * sin(Dlon / 2)**2,
    C is 2 * atan2(sqrt(A), sqrt(1 - A)),
    Est is C * R.

%--------------------------------- - - - - - - - - - -  -  -  -  -  -
% PREDICADOS AUXILIARES
%--------------------------------- - - - - - - - - - -  -  -  -  -  -

membro(X, [X|_]).
membro(X, [_|Xs]):-
        membro(X, Xs).

membros([X|Xs],Members) :- membro(X,Members),
                           membros(Xs,Members).

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

nao( Questao ) :-
    Questao, !, fail.
nao( Questao ).

escrever([]).
escrever([X|L]) :- write(X),nl,escrever(L).

minimo(L, (A,B)) :-
    seleciona((A,B), L, R),
    \+ (membro((A1,B1), R), B1 < B).

maximo(L, (A,B)) :-
    seleciona((A,B), L, R),
    \+ (membro((A1,B1), R), B1 > B).

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

getArco(Origem, Destino, Custo) :- arco(Origem, Destino, Custo).

getPonto(Rua, Latitude, Longitude, Adjacentes, Lixo, Capacidade) :- ponto(Rua, Latitude, L

getLixo(Rua, Tipo) :- ponto(Rua,_,_,_,Lixo,_).
```

```
pontosL :- listing(ponto).

arcosL :- listing(arco).
```