Sistemas de Representação de Conhecimento e Raciocínio

Métodos de Resolução de Problemas e de Procura

Trabalho realizado por:

Luís Filipe Cruz Sobral

A89474 Luís Sobral

Projeto Individual SRCR
2020/2021
Universidade do Minho

Resumo

O presente relatorio tem como finalidade a apresentação e descrição das soluções implementadas para os desafios propostos na disciplina de Sistemas de Representação de Conhecimento e Raciocínio.
Este trabalho tem como propósito a representação e modelação de um sistema de recolha de resíduos urbanos nos concelho de Lisboa, tal como a aplicação de algoritmos de pesquisa informada e não-informada. Para isto, foi necessário processar um dataSet que forma uma grafo com todos os pontos de recolha.
Por fim, teremos como resultado final a construção dos vários circuitos possiveis e a identificação dos mais rápidos e eficiêntes.

Conteúdo

1 Introducao

Este trabalho teve como finalidade a representação de um sistema de circuitos compostos por vários pontos de recolha de resíduos urbanos na cidade de Lisboa e, por sua vez, usar o prolog para aplicar algoritmos de pesquisa.
O sistema baseia-se na disposição geográfica de pontos de recolha e arcos, que representam as conexões entre pontos de recolha e a distancia entre os mesmo. Estas informações são obtidas através do processamento do dataSet disponibilizado.
Para representar os circuitos foram desenvolvidos algoritmos de pesquisa informada e não-informada, tendo cada um as suas vantagens conforme o ambiente em questâo.

2 Geração dos grafos

Conforme o solicitado no enunciado, era essencial o parse do dados e as criação das liagações entre os pontos de recolha. Posto isto, foi escolhida a linguagem **Python**, pois apresenta vária vantagens, uma das quais o facil manuseamento dos dados. Assim, foram criados dois ficheiros **pontos_Recolha.pl** e **arco.pl**, o primeiro contém informações relativas a cada ponto de recolha tais como: *Latitude, Longitude, Nome da rua, Tipo de resíduos, Capacidade*, o segundo contém informações relativas aos arcos: *Rua origem, Rua destino, Distância.*

2.1 Tratamento de dados

Depois de analizar o dataSet verificamos que havia informação com acentos, algo que poderia resultar em problemas numa fase mais tardia, pelo que usamos os seguintes comandos para que todos as string fossem tornadas em ASCII Friendly:

```
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')
```

2.2 Pontos de recolha

Após analisarmos o problema identicamos os campos que cada ponto de recolha iria necessitar e obtemos o ficheiro **pontos_Recolha.pl** com informação sobre todos os pontos de recolha:

```
%%pontorecolha(ID,Lat,Long,Rua,[RuasAdjecentes],tipo,capacidade)
ponto_recolha(-9.143309,38.708079,'R do Alecrim',0,5940).
ponto_recolha(-9.143309,38.708079,'R do Alecrim',2,2370).
ponto_recolha(-9.143309,38.708079,'R do Alecrim',3,90).
ponto_recolha(-9.142551,38.707329,'R Corpo Santo',0,2480).
ponto_recolha(-9.142766,38.707084,'Tv Corpo Santo',0,2320).
ponto_recolha(-9.142766,38.707084,'Tv Corpo Santo',2,240).
ponto_recolha(-9.142408,38.706997,'R Bernardino da Costa',0,6320).
ponto_recolha(-9.142408,38.706997,'R Bernardino da Costa',2,240).
ponto_recolha(-9.151251,38.708775,'Lg Conde-Barao',0,11240).
ponto_recolha(-9.151251,38.708775,'Lg Conde-Barao',2,140).
ponto_recolha(-9.151251,38.708775,'Lg Conde-Barao',3,140).
ponto_recolha(-9.149106,38.709073,'Tv Marques de Sampaio',0,240).
ponto_recolha(-9.147650,38.708564,'R da Boavista',0,12600).
ponto_recolha(-9.147650,38.708564,'R da Boavista',1,90).
ponto_recolha(-9.147650,38.708564,'R da Boavista',2,90).
ponto_recolha(-9.147650,38.708564,'R da Boavista',3,90).
ponto_recolha(-9.147650,38.708564,'R da Boavista',4,140).
ponto_recolha(-9.152060,38.708282,'Tv do Cais do Tojo',0,140).
ponto_recolha(-9.152472,38.708134,'R Cais do Tojo',0,3150).
ponto_recolha(-9.151545,38.708404,'Bqr do Duro',0,2530).
ponto_recolha(-9.147514,38.710339,'Tv Santa Catarina',0,480).
ponto_recolha(-9.147514,38.710339,'Tv Santa Catarina',4,240).
ponto_recolha(-9.148566,38.709943,'R Ferreiros a Santa Catarina',0,1440).
ponto_recolha(-9.150464,38.706801,'R Instituto Industrial',0,7960).
ponto_recolha(-9.150464,38.706801,'R Instituto Industrial',1,720).
ponto_recolha(-9.150464,38.706801,'R Instituto Industrial',3,480).
ponto_recolha(-9.150464,38.706801,'R Instituto Industrial',4,560).
ponto_recolha(-9.148518,38.709808,'R Santa Catarina',0,1920).
ponto_recolha(-9.147044,38.708019,'R Moeda',0,5720).
ponto_recolha(-9.146144,38.707784,'Tv Carvalho',0,7590).
ponto_recolha(-9.146144,38.707784,'Tv Carvalho',2,6370).
ponto_recolha(-9.146144,38.707784,'Tv Carvalho',3,6000).
ponto_recolha(-9.146144,38.707784,'Tv Carvalho',4,3180).
ponto_recolha(-9.151994,38.707625,'R Dom Luis I',0,18290).
ponto_recolha(-9.151994,38.707625,'R Dom Luis I',2,480).
ponto_recolha(-9.151994,38.707625,'R Dom Luis I',3,330).
ponto_recolha(-9.147146,38.707100,'Pc Dom Luis I',0,1480).
ponto_recolha(-9.147146,38.707100,'Pc Dom Luis I',2,280).
ponto_recolha(-9.148354,38.710395,'Tv Condessa do Rio',0,1440).
ponto_recolha(-9.148354,38.710395,'Tv Condessa do Rio',3,240).
ponto_recolha(-9.144930,38.707281,'R Ribeira Nova',0,1700).
ponto_recolha(-9.144930,38.707281,'R Ribeira Nova',2,1730).
ponto_recolha(-9.143508,38.707528,'R Sao Paulo',0,23460).
ponto_recolha(-9.143508,38.707528,'R Sao Paulo',2,760).
ponto_recolha(-9.143508,38.707528,'R Sao Paulo',3,280).
ponto_recolha(-9.144288,38.707422,'R Nova do Carvalho',0,10920).
ponto_recolha(-9.144288,38.707422,'R Nova do Carvalho',2,860).
ponto_recolha(-9.144028,38.706893,'R Remolares',0,5580).
ponto_recolha(-9.144028,38.706893,'R Remolares',2,280).
ponto_recolha(-9.144028,38.706893,'R Remolares',4,3000).
ponto_recolha(-9.144416,38.706736,'Tv dos Remolares',0,2700).
ponto_recolha(-9.144047,38.705829,'Cais do Sodre',0,3240).
ponto_recolha(-9.144047,38.705829,'Cais do Sodre',2,140).
ponto_recolha(-9.146220,38.708270,'Bc da Moeda',0,180).
ponto_recolha(-9.149564,38.709006,'Pto Galega',0,240).
```

2.3 Arcos

Como podemos verificar, são criados 2 dicionários: o dict *adjacentes* guarda para cada rua um set com as ruas vizinhas, o dict *ruas* guarda para cada rua a sua localização. A partir destes dicionários e usando a função **calc_distance** é possivél criar o ficheiro **arco.pl**, usando o seguinte codigo:

```
def calc_distance(lat1,lon1,lat2,lon2):
    # Raio aproximado da terra
    R = 6371.0
    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))
    distance = R * c
    return distance
```

```
%%arco(Rua1,Rua2,Distancia)
arco('R do Alecrim','R Ferragial',2.685187).
arco('R do Alecrim','Pc Duque da Terceira',9.316738).
arco('Pc Duque da Terceira','Av 24 de Julho',6.964898).
arco('R Corpo Santo','Lg Corpo Santo',4.255262).
arco('R Corpo Santo','Tv Corpo Santo',2.030926).
arco('Tv Corpo Santo','R Bernardino da Costa',2.339651).
arco('Tv Corpo Santo','Cais do Sodre',11.205668).
arco('R Bernardino da Costa','Pc Duque da Terceira',6.280060).
arco('R Bernardino da Costa','Lg Corpo Santo',2.603633).
arco('Cais do Sodre','Pc Duque da Terceira',6.522797).
arco('Cais do Sodre','Lg Corpo Santo',14.645866).
arco('Cais do Sodre','Av 24 de Julho',3.903619).
arco('R da Boavista','R Sao Paulo',27.139436).
arco('R da Boavista','R Instituto Industrial',20.936870).
arco('Lg Conde-Barao','R da Boavista',22.981014).
arco('Lg Conde-Barao','Bqr do Duro',2.950182).
arco('Lg Conde-Barao','Tv do Cais do Tojo',5.978742).
arco('Bqr do Duro','R Dom Luis I',5.567015).
arco('Tv do Cais do Tojo','R Cais do Tojo',2.775527).
arco('Tv Marques de Sampaio','R da Boavista',9.786876).
arco('R Sao Paulo','Pc Sao Paulo',10.324381).
arco('R Sao Paulo','Tv dos Remolares',7.548610).
arco('R Sao Paulo','Tv Corpo Santo',5.452555).
arco('R Sao Paulo','Bc da Moeda',17.866313).
arco('R Sao Paulo','R Corpo Santo',6.217261).
arco('R Instituto Industrial','R Dom Luis I',10.976745).
arco('R Instituto Industrial','Av 24 de Julho',38.874514).
arco('R Cais do Tojo','Bqr do Duro',6.134996).
arco('R Dom Luis I','Pc Dom Luis I',31.049788).
arco('R Moeda','R Sao Paulo',22.730047).
arco('R Moeda','Pc Dom Luis I',5.672176).
arco('Pc Dom Luis I','R Ribeira Nova',14.162964).
arco('Pc Dom Luis I','Av 24 de Julho',18.205486).
arco('R Ribeira Nova','R Remolares',6.220552).
arco('Tv Carvalho','Tv Carvalho',0.000000).
arco('Tv Carvalho','Pc Sao Paulo',6.765970).
arco('Tv Carvalho','R Ribeira Nova',8.326286).
arco('Tv Carvalho','R Sao Paulo',16.868747).
arco('Pc Sao Paulo','Tv Ribeira Nova',4.516904).
arco('R Remolares','Pc Duque da Terceira',5.079338).
arco('R Remolares','Tv dos Remolares',2.655034).
arco('R Remolares','Tv Ribeira Nova',4.689738).
arco('Tv dos Remolares','Av 24 de Julho',2.273068).
arco('R Nova do Carvalho','R do Alecrim',7.423652).
arco('R Nova do Carvalho','Pc Sao Paulo',5.881879).
arco('R Nova do Carvalho','Tv dos Remolares',4.276803).
arco('R Nova do Carvalho','Tv Corpo Santo',9.916656).
arco('Tv Ribeira Nova','R Nova do Carvalho',2.166418).
arco('Pto Galega','R da Boavista',12.496597).
arco('Bc da Boavista','R da Boavista',19.012113).
arco('Pc Ribeira Nova','R Ribeira Nova',2.438684).
arco('Pc Ribeira Nova','Av 24 de Julho',4.199677).
arco('Bc Francisco Andre','R da Boavista',14.392665).
arco('Tv de Sao Paulo','Pc Sao Paulo',2.554051).
arco('Tv de Sao Paulo','R Ribeira Nova',3.245153).
```

3 Algoritmos de pesquisa

Para a resolução das queries que serão apresentados posteriormente, foi nos solicitados o uso de algoritmos de pesquisa, os quais podem ser de 2 tipos: Pesquisa Informada e Pesquisa Não Informada. Cada um destes algoritmos apresenta vantagens e desvantagens, relativamente à pesquisa não informada podemos destacar o Depth First. Para caminhos complexos este requere um tempo de procura maior, mas quando encontra uma solução depressa torna-se mais eficiênte. Em alternativa, existe o breath First, este exige elevados gastos de memória e tem um tempo de pesquisa superior ao anterior.
Relativamente a algoritmos de pesquisa não informada, estes fazem uso de heurísticas para auxiliar a pesquisa. Para o problema proposto a heurística selecionada será a distância entre 2 pontos de recolha, calculada através das suas coordenadas.

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening
Complete?	Yes	Yes	No	No	Yes
Time	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(b^m)$	$O(b^l)$	$O(b^d)$
Space	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(bm)$	$O(bl)$	$O(bd)$
Optimal?	Yes	Yes	No	No	Yes

4 Resultados

Inicialmente definimos o estado inicial e estado final.

```
%% Estado Inicial e Estado Final
inicial('R da Boavista').
final('Tv Corpo Santo').
```

4.1 Pesquisa Não Informada

De seguida serão apresentadas as implementações feitas para pesquisa não informada, juntamente com a sua explicação e casos de teste.

4.1.1 Querie 1: Gerar os circuitos de recolha tanto indiferenciada como seletiva, caso existam, que cubram um determinado território

Para esta questão, testamos todas a possibilidades em que é testado uma campo Visitados, que regista todos os ponto já visitados evitando assim loops infinitos.
A lista referente ao percurso apenas é preenchida quando atingimos o nodo final, pelo que foi definido o seguinte caso de paragem:

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

A figura a baixo apresenta todos os circuitos sugeridos e respetiva distância entre 'R da Boavista' e 'Tv Corpo Santo'.

```
?- ppCustoTodasSolucoes.
[R da Boavista,R Sao Paulo,Pc Sao Paulo,Tv Ribeira Nova,R Nova do Carvalho,Tv Corpo Santo],54.063795
[R da Boavista,R Sao Paulo,Tv Corpo Santo],32.591991
[R da Boavista,R Sao Paulo,R Corpo Santo,Tv Corpo Santo],35.387623
[R da Boavista,R Instituto Industrial,R Dom Luis I,Pc Dom Luis I,R Ribeira Nova,R Remolares,Tv Ribei
ra Nova,R Nova do Carvalho,Tv Corpo Santo],100.119731
true.
```

A cada tipo de resíduos corresponde um int como indicado em baixo.

```
def tipotoint(tipo):
    i = 0
    if tipo == 'Lixos':
        i = 0
    elif tipo == 'Organicos':
        i = 1
    elif tipo == 'Papel e Cartao':
        i = 2
    elif tipo == 'Embalagens':
        i = 3
    elif tipo == 'Vidro':
        i = 4
    return i
```

Para cada resíduo foram gerados os seguintes circuitos:

```
?- ppCustoTiposTodasSolucoes(0).
[R da Boavista,R Sao Paulo,Pc Sao Paulo,Tv Ribeira Nova,R Nova do Carvalho,Tv Corpo Santo],54.063795
[R da Boavista,R Sao Paulo,Tv Corpo Santo],32.591991
[R da Boavista,R Sao Paulo,R Corpo Santo,Tv Corpo Santo],35.387623
[R da Boavista,R Instituto Industrial,R Dom Luis I,Pc Dom Luis I,R Ribeira Nova,R Remolares,Tv Ribei
ra Nova,R Nova do Carvalho,Tv Corpo Santo],100.119731
true.

?- ppCustoTiposTodasSolucoes(1).
true.

?- ppCustoTiposTodasSolucoes(2).
[R da Boavista,R Sao Paulo,Tv Corpo Santo],32.591991
true.

?- ppCustoTiposTodasSolucoes(3).
[R da Boavista,R Sao Paulo,Tv Corpo Santo],32.591991
true.

?- ppCustoTiposTodasSolucoes(4).
true.
```

4.1.2 Querie 2: Identificar quais os circuitos com mais pontos de recolha (por tipo de resíduo a recolher)

Para esta questão, apenas tivemos de adaptar o predicado *ppCustoTiposTodasSolucoes* de modo a calcular também o número de pontos de recolha.

```
?- ppCustoTiposTodasSolucoes2(0).
[R da Boavista,R Sao Paulo,Pc Sao Paulo,Tv Ribeira Nova,R Nova do Carvalho,Tv Corpo Santo],6
[R da Boavista,R Sao Paulo,Tv Corpo Santo],3
[R da Boavista,R Sao Paulo,R Corpo Santo,Tv Corpo Santo],4
[R da Boavista,R Instituto Industrial,R Dom Luis I,Pc Dom Luis I,R Ribeira Nova,R Remolares,Tv Ribei
ra Nova,R Nova do Carvalho,Tv Corpo Santo],9
true.

?- ppCustoTiposTodasSolucoes2(1).
true.

?- ppCustoTiposTodasSolucoes2(2).
[R da Boavista,R Sao Paulo,Tv Corpo Santo],3
true.

?- ppCustoTiposTodasSolucoes2(3).
[R da Boavista,R Sao Paulo,Tv Corpo Santo],3
true.

?- ppCustoTiposTodasSolucoes2(4).
true.
```

4.1.3 Querie 3: Comparar circuitos de recolha tendo em conta os indicadores de produtividade

Para esta questão usamos como indicador de produtividade a distancia média percorrida entre pontos de recolha. Para isso calculamos a distancia e o número de arcos do circuito e, por fim, usamos a seguinte expressão para calcular a produtividade:

```
Prod is Custo/Arcos.
```

A figura a baixo apresenta todos os circuitos sugeridos e respetiva distância, nº arcos e produtividade entre 'R da Boavista' e 'Tv Corpo Santo'.

```
?- ppTodasSolucoes.
[R da Boavista,R Sao Paulo,Pc Sao Paulo,Tv Ribeira Nova,R Nova do Carvalho,Tv Corpo Santo],5,54.0637
95,10.812759
[R da Boavista,R Sao Paulo,Tv Corpo Santo],2,32.591991,16.2959955
[R da Boavista,R Sao Paulo,R Corpo Santo,Tv Corpo Santo],3,35.387623,11.795874333333332
[R da Boavista,R Instituto Industrial,R Dom Luis I,Pc Dom Luis I,R Ribeira Nova,R Remolares,Tv Ribei
ra Nova,R Nova do Carvalho,Tv Corpo Santo],8,100.119731,12.514966375
true.
```

4.1.4 Querie 4: Escolher o circuito mais rápido (usando o critério da distância)

Para esta questão, usamos o predicado minimo para calcular na lista de circuitos gerada pelo predicado ppCustoTodasSolucoes.

```
%% Caminho mais rápido
melhorCaminhoDF(R):- ppCustoTodasSolucoes(L),minimo(L,R).

minimo([(P,X)],(P,X)).
minimo([(P,X)|L],(Py,Y)):- minimo(L,(Py,Y)), X>Y.
minimo([(Px,X)|L],(Px,X)):- minimo(L,(Py,Y)), Y>=X.
```

Na figura a baixo apresenta-se o circuito mais rápido, tal como a sua distância.

```
?- melhorCaminhoDF(R).
R =  (['R da Boavista', 'R Sao Paulo', 'Tv Corpo Santo'], 32.591991) .
```

4.1.5 Querie 5: Escolher o circuito mais eficiente (usando um critério de eficiência à escolha)

Para esta questão, usamos o predicado minimo para calcular na lista de circuitos gerada pelo predicado ppTodasSolucoes.

```
%% Caminho mais eficiente
caminhoEficienteDF(R):- findall((S,P),(resolve_ppProd(S,A,C,P)),L),minimo(L,R).
```

```
?- melhorCaminhoDF(R).
R =  (['R da Boavista', 'R Sao Paulo', 'Tv Corpo Santo'], 32.591991) .
```

4.2 Pesquisa informada

A heurística escolhida para este tipo de pesquisa tem como função calcular a distância euclideana entre 2 pontos de recolha, procurando sempre pela distância mínima. Para esta questão usamos os seguintes predicados auxiliares:

```
inverso(Xs,Ys):-
    inverso(Xs,[],Ys).

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

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

Sendo que obtivemos os seguintes resultados:

```
?- resolve_gulosa(R,0).
R = ['R da Boavista', 'R Instituto Industrial', 'Av 24 de Julho', 'Cais do Sodre', 'Tv Corpo Santo']
/74.920671 .

?- resolve_aestrela(R).
R = ['R da Boavista', 'R Sao Paulo', 'Tv Corpo Santo']/32.591991 .
```

5 Analise de Resultados

Estratégia	**Tempo (segundos)**	**Espaço**	**Profundidade \Custo**	**Encontrou melhor solução?**
Depth-First	0.00	O(b_m)	.	Não
Breath-First	0.00	O(b^d)	.	Sim
Gulosa	0.01	Pior Caso: O(b^m) Melhor Caso: O(b_d)	74.920671	Sim
A*	0.01	Nº de Nodos com g(n) + h(n) <= C^*	32.591991	Sim

6 Conclusão

Com este trabalho pude aprofundar os conhecimentos obtidos durante o presente semestre, nas aulas teóricas e práticas. O algoritmos de busca em profundidade revelou-se bastante eficaz, conseguindo responder ao que era solicitado e oferencendo sempre soluções válidas. Este algoritmo foi capaz de abrangir todos os casos solicitados em tempo de execução aceitável. Relativamente a algoritmos de pesquisa informada o algorito A* revelou-se o mais eficiente e o que melhor se enquadra no problema proposto. De um modo geral, alcançei os objetivos propostos, apesar de algumas dificuldades, e reforçei as minhas habilidades na linguagem Prolog.

7 Apêndice

7.1 Main.pl

```
%Declaracoes iniciaias

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

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

:- include('pontos_Recolha.pl').
:- include('arco.pl').

%% Estado Inicial e Estado Final
inicial('R da Boavista').
final('Tv Corpo Santo').

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

%% Com dist ncia
ppCustoTodasSolucoes :- findall((S,C),(resolve_pp(S,C)),L),escreve
    (L).

resolve_pp([Nodo|Caminho],Custo) :-
  inicial(Nodo),
    profundidadeprimeiro(Nodo, [Nodo], Caminho, Custo).

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

profundidadeprimeiro(Nodo,Historico,[ProxNodo|Caminho],Custo) :-
    arco(Nodo,ProxNodo,Dist),
    nao(membro(ProxNodo,Historico)),
    profundidadeprimeiro(ProxNodo,[ProxNodo|Historico],Caminho,
       Custo2),
    Custo is Dist + Custo2.

%% Caminho mais r pido
melhorCaminhoDF(R):- findall((S,C),(resolve_pp(S,C)),L),minimo(L,R
    ).


```

```
38 %% N mero de arcos
   ------------------------------------------------------------
39 ppArcosTodasSolucoes :- findall((S,C),(resolve_ppArcos(S,C)),L),
   escreve(L).
40
41 resolve_ppArcos([Nodo|Caminho],Arcos) :-
42   inicial(Nodo),
43    profundidadeprimeiroArcos(Nodo, [Nodo], Caminho, Arcos).
44
45 profundidadeprimeiroArcos(Nodo,_,[],0) :- final(Nodo).
46
47 profundidadeprimeiroArcos(Nodo,Historico,[ProxNodo|Caminho],Arcos)
    :-
48    arco(Nodo,ProxNodo,Dist),
49    nao(membro(ProxNodo,Historico)),
50    profundidadeprimeiroArcos(ProxNodo,[ProxNodo|Historico],
      Caminho,Arcos2),
51    Arcos is 1 + Arcos2.
52
53  %% Caminho mais curto
54 caminhoCurtoDF(R):- findall((S,C),(resolve_ppArcos(S,C)),L),minimo
   (L,R).
55
56 %% Produtividade
   ------------------------------------------------------------
57
58 ppTodasSolucoes :- findall((S,A,C,P),(resolve_ppProd(S,A,C,P)),L),
   escreve(L).
59
60 resolve_ppProd([Nodo|Caminho],Arcos,Custo,Prod) :-
61   inicial(Nodo),
62    profundidadeprimeiroProd(Nodo, [Nodo], Caminho, Arcos, Custo),
63    Prod is Custo/Arcos.
64
65 profundidadeprimeiroProd(Nodo,_,[],0,0) :- final(Nodo).
66
67 profundidadeprimeiroProd(Nodo,Historico,[ProxNodo|Caminho],Arcos,
   Custo) :-
68    arco(Nodo,ProxNodo,Dist),
69    nao(membro(ProxNodo,Historico)),
70    profundidadeprimeiroProd(ProxNodo,[ProxNodo|Historico],Caminho
      ,Arcos2, Custo2),
71    Arcos is 1 + Arcos2,
72    Custo is Dist + Custo2.
73
74 %% Caminho mais eficiente
75 caminhoEficienteDF(R):- findall((S,P),(resolve_ppProd(S,A,C,P)),L)
   ,minimo(L,R).
76
77 %% Tipos de Res duos
   ------------------------------------------------------------
78
79 ppCustoTiposTodasSolucoes(T) :- findall((S,C),(resolve_ppTipo(S,C,
   T)),L),escreve(L).
80
81 resolve_ppTipo([Nodo|Caminho],Custo,Tipo) :-
82   inicial(Nodo),
83    profundidadeprimeiroTipo(Nodo, [Nodo], Caminho, Custo,Tipo).
```

```
84
85  profundidadeprimeiroTipo(Nodo,_,[],0,_) :- final(Nodo).
86
87  profundidadeprimeiroTipo(Nodo,Historico,[ProxNodo|Caminho],Custo,
        Tipo) :-
88    ponto_recolha(_,_,Nodo,Tipo,_),
89      arco(Nodo,ProxNodo,Dist),
90      nao(membro(ProxNodo,Historico)),
91      profundidadeprimeiroTipo(ProxNodo,[ProxNodo|Historico],Caminho
            ,Custo2,Tipo),
92      Custo is Dist + Custo2.
93
94  %% Tipos de Res duos
        ------------------------------------------------------
95
96
97  ppCustoTiposTodasSolucoes2(T) :- findall((S,C),(resolve_ppTipo2(S,
        C,T)),L),escreve(L).
98
99  resolve_ppTipo2([Nodo|Caminho],Arcos,Tipo) :-
100   inicial(Nodo),
101     profundidadeprimeiroTipo2(Nodo, [Nodo], Caminho, Arcos,Tipo).
102
103 profundidadeprimeiroTipo2(Nodo,_,[],1,_) :- final(Nodo).
104
105 profundidadeprimeiroTipo2(Nodo,Historico,[ProxNodo|Caminho],Arcos,
        Tipo) :-
106   ponto_recolha(_,_,Nodo,Tipo,_),
107     arco(Nodo,ProxNodo,Dist),
108     nao(membro(ProxNodo,Historico)),
109     profundidadeprimeiroTipo2(ProxNodo,[ProxNodo|Historico],
            Caminho,Arcos2,Tipo),
110     Arcos is 1 + Arcos2.
111
112
113
114 %% ---------- Breath First
        ------------------------------------------------
115
116 %% Com dist ncia
117 pdCustoTodasSolucoes(L) :- findall((S),(resolve_pd(S)),L).
118
119 resolve_pd([Nodo|Caminho]) :-
120   inicial(Nodo),
121     breathFirst(Nodo, [Nodo], Caminho).
122
123 breathFirst(Nodo,_,[]) :- final(Nodo).
124
125 breathFirst(Nodo,Historico,Caminho) :-
126   findall(X,
127             (arco(X,Nodo,_),not(member(X,Historico))),
128             [T|Extend]),
129     append(Historico, [T|Extend], Historico2),
130     append(Caminho, [T|Extend], [Next|Caminho2]),
131     breathFirst(Next,Historico2,Caminho2).
132
133 %% Busca limitada em profundidade
        ------------------------------------------------------------
```

```
134 pplArcosTodasSolucoes(L,Max) :- findall((S,C),(resolve_pplArcos(S,
    C,Max)),L).
135
136 resolve_pplArcos([Nodo|Caminho],Arcos,Max) :-
137   inicial(Nodo),
138     profundidadeprimeiroLimArcos(Nodo, [Nodo], Caminho, Arcos,Max)
        .
139
140 profundidadeprimeiroLimArcos(Nodo,_,[],0,_) :- final(Nodo).
141 profundidadeprimeiroLimArcos(Nodo,Historico,[ProxNodo|Caminho],
    Arcos,Max) :-
142     arco(Nodo,ProxNodo,Dist),
143     nao(membro(ProxNodo,Historico)),
144     profundidadeprimeiroLimArcos(ProxNodo,[ProxNodo|Historico],
        Caminho,Arcos2,Max),
145     Arcos is 1 + Arcos2,
146     Arcos < Max.
147
148 %% Busca limitada em profundidade eficiencia
    ------------------------------------------------
149 pplCustosTodasSolucoes(L,Max) :- findall((S,E),(resolve_pplCusto(S
    ,C,Max,E)),L).
150
151 resolve_pplCusto([Nodo|Caminho],Arcos,Max,Ef) :-
152   inicial(Nodo),
153     profundidadeprimeiroLimCusto(Nodo, [Nodo], Caminho, Arcos,Max,
        Dist),
154     Ef is Dist/Arcos.
155
156 profundidadeprimeiroLimCusto(Nodo,_,[],0,_,_) :- final(Nodo).
157 profundidadeprimeiroLimCusto(Nodo,Historico,[ProxNodo|Caminho],
    Arcos,Max,Dis) :-
158     arco(Nodo,ProxNodo,Dist),
159     nao(membro(ProxNodo,Historico)),
160     profundidadeprimeiroLimCusto(ProxNodo,[ProxNodo|Historico],
        Caminho,Arcos2,Max,Dis2),
161     Arcos is 1 + Arcos2,
162     Dis is Dist + Dis2,
163     Arcos < Max.
164
165
166 %--------------------------------
167 % s o l u   o
168
169 adjacente3([Nodo|Caminho]/Custo/_,[ProxNodo,Nodo|Caminho]/
    NovoCusto/Est) :-
170   (arco(Nodo,ProxNodo,PassoCusto) ; arco(ProxNodo,Nodo,PassoCusto)
      ),
171   nao(membro(ProxNodo,Caminho)),
172   NovoCusto is Custo + PassoCusto,
173   ponto_recolha(_,_,ProxNodo,_,Est).
174
175 adjacente2([Nodo|Caminho]/Custo/_,[ProxNodo,Nodo|Caminho]/
    NovoCusto/Est,Tipo) :-
176   (arco(Nodo,ProxNodo,PassoCusto) ; arco(ProxNodo,Nodo,PassoCusto)
      ),
177   nao(membro(ProxNodo,Caminho)),
178   NovoCusto is Custo + PassoCusto,
```

```
179   ponto_recolha(_,_,ProxNodo,Tipo,Est).
180
181 %--------------------------------
182 % solu   o
183
184 resolve_gulosa(Caminho/Custo,Tipo) :-
185   inicial(Nodo),
186     ponto_recolha(_,_,Nodo,Tipo,Estima),
187     agulosa([[Nodo]/0/Estima], InvCaminho/Custo/_,Tipo),
188     inverso(InvCaminho,Caminho).
189
190 agulosa(Caminhos,Caminho,Tipo) :-
191     obtem_melhor_g(Caminhos,Caminho),
192     Caminho = [Nodo|_]/_/_,
193     final(Nodo).
194
195 agulosa(Caminhos,SolucaoCaminho,Tipo) :-
196     obtem_melhor_g(Caminhos, MelhorCaminho),
197     seleciona(MelhorCaminho,Caminhos,OutrosCaminhos),
198     expande_gulosa(MelhorCaminho,ExpCaminhos,Tipo),
199     append(OutrosCaminhos, ExpCaminhos, NovosCaminhos),
200     agulosa(NovosCaminhos,SolucaoCaminho,Tipo).
201
202 expande_gulosa(Caminho,ExpCaminhos,Tipo):-
203     findall(NovoCaminho,adjacente2(Caminho,NovoCaminho,Tipo),
          ExpCaminhos).
204
205 obtem_melhor_g([Caminho],Caminho) :- !.
206
207 obtem_melhor_g([Caminho1/Custo1/Est1,_/Custo2/Est2|Caminhos],
      MelhorCaminho):-
208     Est1 =< Est2, !,
209     obtem_melhor_g([Caminho1/Custo1/Est1|Caminhos],MelhorCaminho).
210
211 obtem_melhor_g([_|Caminhos],MelhorCaminho) :-
212     obtem_melhor_g(Caminhos, MelhorCaminho).
213
214 %-----------------------------------------
215 % Pesquisa A*
216
217 resolve_aestrela(Caminho/Custo) :-
218   inicial(Nodo),
219     ponto_recolha(_,_,Nodo,_,Estima),
220     aestrela([[Nodo]/0/Estima], InvCaminho/Custo/_),
221     inverso(InvCaminho,Caminho).
222
223 aestrela(Caminhos,Caminho) :-
224     obtem_melhor(Caminhos,Caminho),
225     Caminho = [Nodo|_]/_/_,
226     final(Nodo).
227
228 aestrela(Caminhos,SolucaoCaminho) :-
229     obtem_melhor(Caminhos, MelhorCaminho),
230     seleciona(MelhorCaminho,Caminhos,OutrosCaminhos),
231     expande_aestrela(MelhorCaminho,ExpCaminhos),
232     append(OutrosCaminhos, ExpCaminhos, NovosCaminhos),
233     aestrela(NovosCaminhos,SolucaoCaminho).
234
```

```
expande_aestrela(Caminho,ExpCaminhos):-
    findall(NovoCaminho,adjacente3(Caminho,NovoCaminho),
        ExpCaminhos).

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

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

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

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


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

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

minimo([(P,X)],(P,X)).
minimo([(P,X)|L],(Py,Y)):- minimo(L,(Py,Y)), X>Y.
minimo([(Px,X)|L],(Px,X)):- minimo(L,(Py,Y)), Y>=X.

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

inverso(Xs,Ys):-
    inverso(Xs,[],Ys).

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

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

7.2 Parser.py

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

#Read data
dataset = pd.read_excel(r'dataset.xlsx')

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


full = open('pontos_recolha.pl','w+',encoding='utf-8')
full.write('%%pontorecolha(ID,Lat,Long,Rua,[RuasAdjecentes],tipo)\
    n')
contentRE = re.compile(
  r'\d{5}: ([A-Za-z0-9 \-]*)[ ,](\(.*\): (.+) - (.+)\))?')

def tipotoint(tipo):
  i = 0
  if tipo == 'Lixos':
   i = 0
  elif tipo == 'Organicos':
   i = 1
  elif tipo == 'Papel e Cartao':
   i = 2
  elif tipo == 'Embalagens':
   i = 3
  elif tipo == 'Vidro':
   i = 4
  return i

adjacentes = collections.defaultdict(set)
ruas = dict()
tipos = dict()
qt = dict()

for line in dataset.values:
  ponto = re.search(contentRE, line[4])
  if ponto:
    rua = ponto.group(1)
    if not(rua in ruas):
      ruas[rua] = [line[0],line[1]]
      tipos[rua] = [0,0,0,0,0]
      qt[rua] = [0,0,0,0,0]

    t = tipotoint(line[5])

    tipos[rua][t] = 1
    qt[rua][t] += line[9]
    if ponto.group(2):
      adj1 = ponto.group(3)
      adj2 = ponto.group(4)
      if not(rua in adjacentes[adj1]):
        adjacentes[rua].add(adj1)
      if not(rua in adjacentes[adj2]):
        adjacentes[rua].add(adj2)

  #full.write("ponto_recolha(%d,%f,%f,'%s',%s,'%s',%d).\n" %(line
      [2],line[0],line[1],rua,adjacente,line[5],line[9]))
#full.close()


```

```
64 for item in ruas.keys():
65   if tipos[item][0] == 1:
66     full.write("ponto_recolha(%f,%f,'%s',%d,%d).\n" %(ruas[item
       ][0],ruas[item][1],item,0,qt[item] [0]))
67   if tipos[item][1] == 1:
68     full.write("ponto_recolha(%f,%f,'%s',%d,%d).\n" %(ruas[item
       ][0],ruas[item][1],item,1,qt[item][1]))
69   if tipos[item][2] == 1:
70     full.write("ponto_recolha(%f,%f,'%s',%d,%d).\n" %(ruas[item
       ][0],ruas[item][1],item,2,qt[item] [2]))
71   if tipos[item][3] == 1:
72     full.write("ponto_recolha(%f,%f,'%s',%d,%d).\n" %(ruas[item
       ][0],ruas[item][1],item,3,qt[item] [3]))
73   if tipos[item][4] == 1:
74     full.write("ponto_recolha(%f,%f,'%s',%d,%d).\n" %(ruas[item
       ][0],ruas[item][1],item,4,qt[item] [4]))
75
76 #Calculate distance
77 def calc2_distance(latitude1,longitude1,latitude2,longitude2):
78     result = np.sqrt(  (latitude1-latitude2)**2 + (longitude1-
       longitude2)**2 )
79     return result
80
81 def calc_distance(lat1,lon1,lat2,lon2):
82     # Raio aproximado da terra
83     R = 6371.0
84
85     dlon = lon2 - lon1
86     dlat = lat2 - lat1
87
88     a = sin(dlat / 2)**2 + cos(lat1) * cos(lat2) * sin(dlon / 2)
       **2
89     c = 2 * atan2(sqrt(a), sqrt(1 - a))
90
91     distance = R * c
92
93     return distance
94
95 full2 = open('arco.pl','w+',encoding='utf-8')
96 full2.write('%%arco(Rua1,Rua2,Distancia)\n')
97
98 for item in adjacentes.items():
99   for rua in item[1]:
100     if (item[0] in ruas) and (rua in ruas):
101       full2.write("arco('%s','%s',%f).\n" %(item[0],rua,
          calc_distance(ruas[item[0]][0],ruas[item[0]][1],ruas[rua
          ][0],ruas[rua][1])))
102 full2.close()
```