Universidade do Minho
Escola de Engenharia

Sistemas de Representação de Conhecimento e Raciocínio

Trabalho Individual

Luís Pedro Oliveira de Castro Vieira A89601

Figura 1: A89601

7 de junho de 2021

Conteúdo

1 Resumo

O presente relatório visa descrever as soluções implementadas e todo o raciocínio e decisões tomadas ao longo do trabalho proposto pela equipa docente da unidade curricular Sistema de Representação de Conhecimento e Raciocínio.

Este trabalho consiste na formulação, através de um dataset fornecido, de um circuito de recolha de lixo numa freguesia de Lisboa, mais especificamente a freguesia da Misericórdia, sobre o qual serão aplicados os algoritmos de pesquisa não informada e de pesquisa informada estudados e apresentados nas aulas práticas.

Primeiramente foi necessário analisar o dataset fornecido, de modo a estruturar e planear o trabalho que viria pela frente, isto é, como seria feita a construção do grafo que ditaria os circuitos a percorrer, o nível de abstração a usar para representar a informação pedida e corresponder aos requisitos do enunciado.

De seguida, foi necessário tratar do processamento do dataset utilizando para tal Python e, através de um script criado, guardar toda a informação definida no passo anterior, escrevendo em ficheiros que posteriormente serão aplicados no trabalho.

Por fim, foram construídos e testados os predicados dos algoritmos a serem implementados, bem como predicados auxiliares aos mesmos.

2 Introdução

O presente trabalho, proposto pela equipa docente da unidade curricular de Sistema de Representação de Conhecimento e Raciocínio, tem como objetivo a representação de um grafo que vise descrever circuitos de recolha de lixo, diferenciado ou não, na freguesia da Misericórdia em Lisboa.

Este grafo será alvo de algoritmos de pesquisa informada e não informada, de modo a responder a certos requisitos minimos presentes no enunciado, através do uso de PROLOG.

O sistema de conexão do grafo baseia-se na disposição geográfica das ruas da freguesia da Misericórdia e dos seus principais pontos de recolha de lixo nas mesmas, existindo então pontos de recolha, e respetivas informações associadas, e arcos, que representação as conexões entre pontos de recolha bem como as distâncias. Tudo isto é possível através da análise do dataset fornecido e com o complementar da informação através de uma pesquisa no Google Maps.

Para efeitos de consideração do tipo de trabalho, como requisitado no enunciado, devo mencionar que se trata da versão simplificada do projeto.

3 Preliminares

Na presente secção passará a ser explicado o planeamento e estruturação de todo o trabalho implementado, refletindo principalmente nas decisões tomadas para a construção do grafo a ser trabalhado.

3.1 Análise do Dataset

Num primeiro encontro com o dataset, sem previamente explorar a origem do mesmo, parecia algo confusa a informação que nele se encontrava, especialmente com denominações como *par*, *impar* e *ambos* nos pontos de recolha locais apresentados.

No entanto, e após algum debate com alguns docentes, rapidamente foi percetível que a informação disponibilizada para a realização do trabalho era apenas uma pequena fração de algo maior e portanto a informação na sua totalidade estava incompleta.

Isto levou à tomada de algumas decisões que certamente afetam todo o trabalho realizado. Neste sentido foi necessário definir o nível de abstração a utilizar durante a idealização do grafo de modo a que o trabalho no geral fizesse sentido e que fosse possível responder com eficácia e eficiência ao que é pedido no enunciado.

3.2 Decisões Tomadas

Assim, tendo em conta o que foi acima mencionado, houve algumas informações presentes no dataset que decidi descartar tais como:

- Menções de *par*, *ímpar* ou *ambos* e o ID presentes na coluna referente aos pontos de recolha;
- ObjectIDs;
- Tipo, quantidade e capacidade do contentor no ponto de recolha, trabalhando apenas com o resíduo presente no contentor e a quantidade total em litros;
- Em termos de coordenadas considerei, para uma dada rua, as que aparecerem na primeira entrada sobre essa rua.

O intuito de descartar estas informações é tornar o mais abstrato possível o conceito do trabalho, de modo a simplificar o raciocínio e implementação dos algoritmos de pesquisa.

Desta forma, os pontos de recolha que estabeleci serão factos com as seguintes informações:

```
ponto_recolha(Nome,Lat,Long,Tipo,QtdTotal)
```

Para cada ponto de recolha o nome atribuído é o nome que se encontra no dataset entre o identificador e a restante informação a seguir ao nome. Por exemplo, para a entrada **15805: R do Alecrim (Par (->)(26->30): R Ferragial - R Ataíde)**, o nome atribuído ao ponto de recolha será "R do Alecrim".

Terá também um valor para a latitude e longitude, sendo que os valores considerados, como mencionado previamente, são os da primeira entrada para cada ponto de recolha. Por exemplo, existem logo vinte e uma (21) entradas no início do dataset acerca de *'R do Alecrim'*, sendo que têm coordenadas diferentes, uma vez que também muda certa informação, a qual foi descartada, como os fatores *par*, *ímpar* ou *ambos*; no entanto, as coordenadas de latitude e longitude da primeira são as designadas para essa rua, sendo que este processo é igual para todas.

Esta decisão relativamente às coordenadas significa que o lixo a ser recolhido, estará concentrado todo no mesmo local. Em termos realistas isto seria inadequado e impensável, mas uma vez que se trata de um cenário de teste, o nível de abstração utilizado permite-nos proceder assim para simplificar o trabalho e o raciocínio.

No que diz respeito aos contentores, uma vez que estes se encontrarão todos no mesmo local, foi descartada a informação sobre a quantidade por contentor e o tipo de contentor existente num ponto de recolha do dataset, sendo apenas utilizada a quantidade total em litros no mesmo. Assim, para cada ponto de recolha, conseguimos associar todos os contentores por tipo de resíduo ('Lixos','Vidro','Embalagens','Orgânico' ou 'Papel e Cartão') e as quantidades totais em litro referidas em cada entrada do dataset.

Relativamente às ligações entre as ruas, existe informação no ponto de recolha referente às ruas adjacentes. Esta informação foi complementada com informação do Google Maps, onde pesquisei por cada rua e vi que mais ruas se ligavam à rua em questão. Isto permitiu, tendo em conta o nível de abstração elevado, aproximar um pouco mais daquilo que é a realidade na freguesia da Misericórdia. Para efeitos de simplificação o grafo construído será bidirecional, ou seja, se existe um arco de A para B, também existe de B para A, sem a necessidade de repetir essa informação.

Previamente tinha idealizado as ligações com a possibilidade de todas as ruas estarem ligadas entre si. No entanto isso seria algo muito custoso, levando a milhares

de arcos, o que traria uma ineficácia enorme ao trabalho a desenvolver, provavelmente nem sendo capaz de o executar. Outra alternativa seria: além das ligações mencionadas em cada ponto de recolha, admitir que as ruas acima e abaixo de uma dada entrada são consideradas ruas adjacentes.

No entanto, acho que o nível de abstração alcançado é o indicado e suficiente para poder trabalhar de forma eficaz e eficiente, podendo aplicar com alguma facilidade os algoritmos de pesquisa.

Estamos agora num ponto de situação em que podemos proceder à recolha da informação definida para uso e à construção dos algoritmos que devem ser implementados.

4 Descrição do Trabalho

Na secção que se segue serão apresentadas as componentes mais práticas do trabalho, isto é, a recolha de informação previamente estipulada e a construção da algoritmia necessária.

4.1 Recolha de Informação

A recolha da informação definida anteriormente será feita através de um script formulado na linguagem Python, a partir do qual serão criados os ficheiros sobre os pontos de recolha e sobre os arcos.

Como forma de diminuir potenciais erros de encoding e pelo facto do PROLOG não representar na sua capacidade máxima alguns carateres especiais, foi transformada a informação de algumas colunas presentes no dataset de forma a remover acentos, nomeadamente as colunas sobre o **PONTO_RECOLHA_LOCAL** e sobre o **CONTENTOR_RESÍDUO**. Para tal foram utilizados os seguinte métodos:

```
excel_file = 'dataset.xlsx'

data = pd.read_excel(excel_file, sheet_name='Folha1')

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

Listing 1: Normalização dos Valores nas Colunas

Passarei agora a explicar o processo de criação de ambos os ficheiros bem como a organização da informação dentro dos mesmos, ficheiros estes que serão a base de conhecimento do nosso trabalho.

4.1.1 Pontos de Recolha

Como mencionado na secção anterior, os pontos de recolha terão associados a si: as suas coordenadas de latitude e longitude, o nome da rua, o tipo de resíduo que nele se encontra e a capacidade total para esse tipo de resíduo.

Para armazenar toda esta informação faço uso de um dicionário em Python para o qual a chave é o nome da rua e o valor correspondente é um outro dicionário onde guardo para a chave *latitude* o seu valor, para a chave *longitude* o seu valor e para

a chave *Tipo_residuo* um dicionário cujas chaves são os diferentes tipos de resíduos presentes nessa rua e os seu valores são as quantidades totais em litros dos contentores nessa rua.

Depois de ter lido e guardado a informação presente no dataset numa variável, e de normalizar os seus valores, foi somente necessário iterar sobre cada entrada e guardar os campos acima mencionados, efetuando-o da seguinte maneira:

```
locais = {}

for line in data.values:
    lat = line[0]
    longi = line[1]
    tipo = line[5]
    qtd = line[9]

    if n := re.match(r'^\d+: ((\w+[ \-]?)+)',line[4]):
      nome = n.group(1)
      if nome[-1] == ' ':
        nome = nome[:-1]

    if nome in locais:
        content = locais.get(nome)
        if tipo in content['Tipo_residuo']:
            locais[nome]['Tipo_residuo'][tipo] += qtd
        else:
            locais[nome]['Tipo_residuo'][tipo] = qtd
    else:
        locais[nome] = {}
        locais[nome]['latitude'] = lat
        locais[nome]['longitude'] = longi
        locais[nome]['Tipo_residuo'] = {tipo : qtd}
```

Listing 2: Informação sobre os Pontos de Recolha

Uma vez que já tenho a informação que necessito para poder construir os pontos de recolha com os quais irei trabalhar, resta-me somente escreve-los num ficheiro *pontos_recolha.pl* com o formato previamente mencionado: ponto_recolha(Nome,Lat,Long,Tipo,Qtd).

```
full = open('pontos_recolha.pl','w+', encoding='utf-8')
full.write('%% ponto_recolha(Nome,Lat,Long,Tipo,QtdTotal)\n')
for local in locais:
    content = locais.get(local)
    for tipo in content['Tipo_residuo']:
        full.write(f"ponto_recolha('{local}',{content['latitude']},{
    content['longitude']},'{tipo}',{content['Tipo_residuo'].get(tipo)
    }).\n")
full.close()
```

Listing 3: Escrita do ficheiro pontos_recolha.pl

4.1.2 Arcos

Antes de proceder à explicação da forma como ficaram definidos os arcos, é importante relembrar o nível de abstração presente no trabalho.

As ligações que considerei para efetuar este trabalho foram aquelas que se encontravam presentes nas entradas do dataset, bem como algumas complementares que retirei depois de uma pesquisa cuidada e trabalhada na freguesia da Misericórdia em Lisboa, através do Google Maps. Assim, depois de analisar com cuidado a informação que tinha, e considerando que iria trabalhar com um grafo bidirecional para, novamente, tornar o mais abstrato possível o trabalho a realizar obtive o seguinte resultado:

```
ligacoes = {
  "R do Alecrim" : ['R Ferragial', 'R Ataide', 'Pc Duque da Terceira','Tv Guilherme Cossoul'],
  "R Corpo Santo" : ['Lg Corpo Santo', 'Tv Corpo Santo'],
  "Tv Corpo Santo" : ['R Bernardino da Costa', 'Cais do Sodre'],
  "R Bernardino da Costa" : ['Lg Corpo Santo', 'Pc Duque da Terceira'],
  "Lg Conde-Barao" : ['R da Boavista', 'Bqr do Duro', 'R Mastros', 'Tv do Cais do Tojo'],
  "Tv Marques de Sampaio" : ['R da Boavista'],
  "R da Boavista" : ['R Sao Paulo', 'Bqr dos Ferreiros', 'Pto Galega', 'R Instituto Insdustrial'],
  "Tv do Cais do Tojo" : ['R Cais do Tojo'],
  "R Cais do Tojo" : ['Av Dom Carlos I', 'Bqr do Duro'],
  "Bqr do Duro" : ['R Dom Luis I'],
  "Tv Santa Catarina" : ['R da Santa Catarina','Tv da Condessa do Rio','R O Seculo'],
  "R Instituto Industrial" : ['Av 24 de Julho', 'R Dom Luis I'],
  "R Santa Catarina" : ['Tv da Condessa do Rio','R Ferreiros a Santa Catarina'],
  "R Moeda" : ['R Sao Paulo', 'Pc Dom Luis I'],
  "Tv Carvalho" : ['R Ribeira Nova', 'Pc Sao Paulo', 'R Sao Paulo'],
  "R Dom Luis I" : ['Av Dom Carlos I', 'Bqr dos Ferreiros', 'Pc Dom Luis I'],
  "Pc Dom Luis I" : ['Av 24 de Julho', 'R Ribeira Nova'],
  "R Ribeira Nova" : ['R Remolares', 'R Instituto Dona Amelia'],
  "R Sao Paulo" : ['R Corpo Santo', 'R Flores', 'Pc Sao Paulo', 'Tv dos Remolares', 'Tv Corpo Santo', 'Bc da Moeda', 'Cc da Bica Grande'],
  "R Nova do Carvalho" : ['Pc Sao Paulo', 'Tv dos Remolares', 'R do Alecrim', 'Tv Corpo Santo'],
  "R Remolares" : ['Pc Duque da Terceira', 'Tv dos Remolares', 'Tv Ribeira Nova', 'Pc Ribeira Nova'],
  "Tv dos Remolares" : ['Av 24 de Julho'],
  "Cais do Sodre" : ['Av 24 de Julho', 'R Cintura', 'Pc Duque da
```

```
    Terceira', 'Lg Corpo Santo'],
  "Bc da Boavista" : ['R da Boavista'],
  "Tv Ribeira Nova" : ['R Nova do Carvalho', 'Av 24 Julho', 'R
    Ribeira Nova'],
  "Pc Sao Paulo" : ['Tv Ribeira Nova'],
  "Lg Corpo Santo" : ['R Arsenal', 'Cc do Ferragial'],
  "Bc Francisco Andre" : ['R da Boavista'],
  "Tv de Sao Paulo" : ['Pc Sao Paulo', 'R Ribeira Nova'],
  "Av 24 de Julho" : ['Pc Duque da Terceira', 'Pc Ribeira Nova'],
  "R Marcos Portugal" : ['R Imprensa Nacional','R Quintinha'],
  "Tv Andre Valente" : ['R O Seculo'],
  "R Emenda" : ['Tv Guilherme Cossoul'],
  "R Salgadeiras" : ['R Diario de Noticias'],
  "Tv Guilherme Cossoul" : ['R Ataide'],
  "R Sao Bento" : ['R Poco dos Negros', 'R Imprensa Nacional','R
    Correia Garcao'],
  "R Vitor Cordon" : ['Cc do Ferragial'],
  "Cc do Ferragial" : ['R Ferragial'],
  "R Teixeira": ['R Diario de Noticias'],
  "R Poco dos Negros" : ['Av Dom Carlos I', 'R dos Mastros','R Cruz
    dos Poiais de Sao Bento'],
  "R Correia Garcao" : ['Av Dom Carlos I'],
  "R Cruz do Poiais de Sao Bento" : ['R Sao Bento'],
  "R da Cintura do Porto de Lisboa" : ['Av 24 de Julho'],
  "R Marechal Saldanha" : ['R Santa Catarina'],
  "R Loreto" : ['R Emenda'],
  "R Antonio Maria Cardoso" : ['R Vitor Cordon'],
  "Tv Sequeiro" : ['R Marechal Saldanha', 'R Emenda'],
  "R Quintinha" : ['R Sao Bento']
}
```

Listing 4: Ligações entre Ruas

Todas as ligações por mim encontradas foram armazenadas num dicionário onde a chave é o nome de uma rua e o valor é uma lista com nomes de ruas às quais se pode chegar a partir da chave. A leitura destas ligações deve ser feita da seguinte maneira: se eu tenho uma chave A ao qual está associada uma lista com os valores B e C, então significa que eu consigo ir de A para B e de A para C.

No entanto, como considero o grafo bidirecional, significa também que consigo ir de B para A e de C para A. No fim, significa que tendo A como intermédio consigo ir de B para C sem precisar de uma ligação direta entre os dois, ou vice-versa.

Os arcos estão então definidos da seguinte forma: *arco(Ponto1,Ponto2,Dist)*, onde *Dist* representa a distância entre o Ponto1 e o Ponto2.

Assim para cada rua presente na informação obtida sobre os pontos de recolha, é calculada a distância a todas as outras ruas da seguinte forma:

```
def calc_distance(lat1,lon1,lat2,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))

  distance = R * c
  return distance*1000 # Retornar em metros


result = {}

for c1, o1 in locais.items():
  for c2, o2 in locais.items():
    if c1 == c2: # Skip distance to itself
      continue
    if c1 not in result:
      result[c1] = {}
    if c2 not in result:
      result[c2] = {}
    if c2 in result[c1]: # Skip calculation if result exists
      continue
    lat1, long1 = o1['latitude'] , o1['longitude']
    lat2, long2 = o2['latitude'] , o2['longitude']
    lat1, long1, lat2, long2 = float(lat1), float(long1), float(lat2
   ), float(long2)
    dist = calc_distance(radians(lat1), radians(long1), radians(lat2
   ), radians(long2))
    result[c1][c2] = dist
    result[c2][c1] = dist
```

Listing 5: Cálculo das Distâncias entre Pontos

Desta forma consigo obter a distância entre dois pontos sem repetir a informação que depois será escrita para um ficheiro *arcos.pl*.

Por fim, relativamente à construção dos arcos, falta somente combinar a informação relativa às adjacências entre pontos de recolha e a informação relativa à distância entre dois pontos.

```
verificados = set()

count = 0
for ponto in result:
```

```
    ligacoes_ponto1 = ligacoes.get(ponto)
    content = result.get(ponto)
    for ponto2 in content:
      if ((ponto,ponto2) not in verificados) or ((ponto2,ponto) not
   in verificados):
        if ligacoes_ponto1:
          if ponto2 in ligacoes_ponto1:
            arcos.add(f"arco('{ponto}','{ponto2}',{content.get(
   ponto2)}).\n")
            verificados.add((ponto,ponto2))
      else:
        pass



for l in arcos:
  arc.write(l)
arc.close()
```

Listing 6: Escrita Final dos Arcos

Temos agora toda a informação necessária para começar a construir os algoritmos necessários.

4.2 Formulação do Problema

O trabalho consiste num problema de pesquisa baseado num circuito de recolha de lixos real, referente à freguesia da Misericórdia em Lisboa, circuito esse que foi alvo de análise e sobre o qual foi efetuada uma recolha de informação pertinente para o projeto.

No entanto é necessário ter em conta mais algumas informações para podermos prosseguir à construção algorítmica.

Como o trabalho considerado é a versão simplificada, não existe a necessidade de guardar um estado inicial, final ou representativo de uma transação no circuito, pois não trabalhamos com uma capacidade limitada. Não obstante, precisamos de definir um ponto inicial de partida e um ponto de chegada.

No dataset existe uma rua sobre a qual não temos informação, a *Rua Ataíde*, mas que consta como ligação à *Rua do Alecrim*. Por isso, considerei que esse ponto seria o local de partida para o camião de recolha do lixo. Foram também escolhidos outros pontos de partida para efeitos de teste. Como ponto de chegada foram escolhidos vários pontos de recolha, também para efeitos de teste.

Para além disso, adicionando novamente mais uma camada ao nível de abstração, considero que a deposição do lixo que o camião recolheu ao longo do percurso é feita no ponto final indicado.

Temos agora o necessário para proceder à implementação e teste dos algoritmos de procura quer informada, quer não-informada.

4.3 Algoritmos Implementados

Na seguinte secção passarão a ser apresentados os algoritmos de pesquisa bem como o seu funcionamento, quando devem ser aplicados e a sua complexidade temporal e espacial.

4.3.1 Pesquisa Não-Informada

- **Profundidade - DFS** *Depth-First Search*

Dada a natureza de procura em profundidade do algoritmo DFS, o mesmo é usado para uma procura exaustiva de todas as possibilidades para, por exemplo, encontrar o caminho mais longo entre dois nodos.

Relativamente à complexidade espacial a pesquisa depth-first apresenta uma complexidade *O(bm)* e a nível da complexidade temporal apresenta um tempo de pesquisa $O(b^m)$, onde **b** representa o fator de ramificação e **m** o tamanho da árvore, ou seja, a sua profundidade máxima.

Foram implementados seis algoritmos diferentes: um em relação ao número de arcos, outro tendo em conta a distância entre os pontos, outro tendo em conta a capacidade e os outros três são semelhantes aos anteriores mas têm em consideração o tipo de lixo indicado por parte do utilizador.

```
% --------------------------
%   Procura Depth-First
% --------------------------

% ------------------ PESQUISA POR RESIDUO ------------------

% Depth-first que da solucoes de acordo com o numero de arcos
resolvedf_arcos_tipo([X|Caminho],T):-
                inicial(X),
                resolvedf_arcos_tipo(X,[X],Caminho,T).

resolvedf_arcos_tipo(A, _ ,[],T):- final(A),!.
resolvedf_arcos_tipo(A, Historico, [Prox|Caminho],T):-
                adjacenteTipo(A,Prox,_,T),
                nao(membro(Prox,Historico)),
                resolvedf_arcos_tipo(Prox, [Prox|Historico], Caminho
   ,T).

df_TodasSolucoes_arcos_tipo(L,T):- findall((C,N),(
   resolvedf_arcos_tipo(C,T),comprimento(C,N)),L).
```

```
19
20 df_Arcos_min_tipo(X,T):- findall((P,C),(resolvedf_arcos_tipo(P,T),
       comprimento(P,C)),L),pairwise_min(L,X).
21
22 df_Arcos_max_tipo(X,T):- findall((P,C),(resolvedf_arcos_tipo(P,T),
       comprimento(P,C)),L),pairwise_max(L,X).
23
24 % Depth-first que da solucoes de acordo com a distancia
25 resolvedf_dist_tipo(Nodo,[Nodo|Caminho],Dist,T):-
26                 inicial(Nodo),
27                 df_dist_tipo(Nodo,[Nodo],Caminho,Dist,T).
28
29 df_dist_tipo(Nodo, _, [], 0,T):-
30                 final(Nodo).
31
32 df_dist_tipo(Nodo,Historico,[NodoProx|Caminho],Dist,T):-
33                 adjacenteTipo(Nodo,NodoProx,Dist1,T),
34                 nao(membro(NodoProx, Historico)),
35                 df_dist_tipo(NodoProx,[NodoProx|Historico],Caminho,
       Dist2,T),
36                 Dist is Dist1 + Dist2.
37
38 df_DistTodasSolucoes_tipo(L,T):- findall((S,D),(resolvedf_dist_tipo(
       N,S,D,T)),L).
39
40 df_Dist_melhor_tipo(Nodo,Cam,Dist,T):- findall((Ca,D),
       resolvedf_dist_tipo(Nodo,Ca,D,T), L),pairwise_min(L,(Cam,Dist)),
       imprime(Cam).
41
42 % Depth-first que da solucoes de acordo com a capacidade
43 resolvedf_cap_tipo(Nodo,[Nodo|Caminho],Cap,T):-
44                 inicial(Nodo),
45                 df_cap_tipo(Nodo,[Nodo],Caminho,Cap,T).
46
47 df_cap_tipo(Nodo, _, [], 0,T):-
48                 final(Nodo).
49
50 df_cap_tipo(Nodo,Historico,[NodoProx|Caminho],Cap,T):-
51                 adjacenteTipo(Nodo,NodoProx,_,T),
52                 getCapacidadeTipo(NodoProx,Cap1,T),
53                 nao(membro(NodoProx, Historico)),
54                 df_cap_tipo(NodoProx,[NodoProx|Historico],Caminho,
       Cap2,T),
55                 Cap is Cap1 + Cap2.
56
57 df_CapTodasSolucoes_tipo(L,T):- findall((S,C),(resolvedf_cap_tipo(N,
       S,C,T)),L).
58
```

```
59 df_Cap_melhor_tipo(Nodo,Cam,Cap,T):- findall((Ca,C),
       resolvedf_cap_tipo(Nodo,Ca,C,T), L),pairwise_min(L,(Cam,Cap)),
       imprime(Cam).
60
61
62 % ------------------ PESQUISA NORMAL ------------------
63
64
65 % Depth-first que da solucoes de acordo com o numero de arcos
66 resolvedf_arcos([X|Caminho]):-
67             inicial(X),
68             resolvedf_arcos(X,[X],Caminho).
69
70 resolvedf_arcos(A, _ ,[]):- final(A),!.
71 resolvedf_arcos(A, Historico, [Prox|Caminho]):-
72             adjacente(A,Prox,_),
73             nao(membro(Prox,Historico)),
74             resolvedf_arcos(Prox, [Prox|Historico], Caminho).
75
76 df_TodasSolucoes_arcos(L):- findall((C,N),(resolvedf_arcos(C),
       comprimento(C,N)),L).
77
78 df_Arcos_min(X):- findall((P,C),(resolvedf_arcos(P),comprimento(P,C)
       ),L),pairwise_min(L,X).
79
80 df_Arcos_max(X):- findall((P,C),(resolvedf_arcos(P),comprimento(P,C)
       ),L),pairwise_max(L,X).
81
82 % Depth-first que da solucoes de acordo com a distancia
83 resolvedf_dist(Nodo,[Nodo|Caminho],Dist):-
84                 inicial(Nodo),
85                 df_dist(Nodo,[Nodo],Caminho,Dist).
86
87 df_dist(Nodo, _, [], 0):-
88                 final(Nodo).
89
90 df_dist(Nodo,Historico,[NodoProx|Caminho],Dist):-
91                 adjacente(Nodo,NodoProx,Dist1),
92                 nao(membro(NodoProx, Historico)),
93                 df_dist(NodoProx,[NodoProx|Historico],Caminho,Dist2)
       ,
94                 Dist is Dist1 + Dist2.
95
96 df_DistTodasSolucoes(L):- findall((S,D),(resolvedf_dist(N,S,D)),L).
97
98 df_Dist_melhor(Nodo,Cam,Dist):- findall((Ca,D), resolvedf_dist(Nodo,
       Ca,D), L),pairwise_min(L,(Cam,Dist)),imprime(Cam).
99
```

```
% Depth-first que da solucoes de acordo com a capacidade
resolvedf_cap(Nodo,[Nodo|Caminho],Cap):-
                inicial(Nodo),
                df_cap(Nodo,[Nodo],Caminho,Cap).

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

df_cap(Nodo,Historico,[NodoProx|Caminho],Cap):-
                adjacente(Nodo,NodoProx,_),
                getCapacidadeTotal(NodoProx,Cap1),
                nao(membro(NodoProx, Historico)),
                df_cap(NodoProx,[NodoProx|Historico],Caminho,Cap2),
                Cap is Cap1 + Cap2.

df_CapTodasSolucoes(L):- findall((S,C),(resolvedf_dist(N,S,C)),L).

df_Cap_melhor(Nodo,Cam,Cap):- findall((Ca,C), resolvedf_cap(Nodo,Ca,
   C), L),pairwise_max(L,(Cam,Cap)),imprime(Cam).
```

Listing 7: Implementação do Algoritmo Depth-First

Como estou a trabalhar a versão simplificada do trabalho, decidi usar também como critério de pesquisa a capacidade em cada ponto de recolha para poder procurar pelo circuito que recolha o máximo de lixo possível.

- **Largura - BFS** *Breadth-First*

A procura em largura, *breadth-first*, pode e deve ser utilizada quando pretendemos encontrar o caminho mais curto entre dois nodos. Em relação à sua complexidade espacial a pesquisa em largura apresenta uma complexidade $O(b^d)$ e a nível da complexidade espacial apresenta um tempo de pesquisa $O(b^d)$, onde **b** representa o fator de ramificação e **d** a profundidade da solução.

Foram implementados dois algoritmos breadth-first diferentes, tendo em conta o número de nodos que percorre e o mesmo mas relativamente a um tipo de resíduo concreto.

```
% --------------------------
%   Procura Breadth-First
% --------------------------

% ------------------ PESQUISA NORMAl ------------------

% Pesquisa breadth first de acordo com o numero de arcos
resolve_bfs(Nodo,Final) :- inicial(Nodo),solve_bfs([[Nodo]],F),
   reverseL(F,Final).

```

```
10 solve_bfs([[N|Path]|_],[N|Path]) :- final(N).
11
12 solve_bfs([Path|Paths],Solution) :-
13         extend_bfs(Path,NewPaths),
14         append(Paths,NewPaths,Paths1),
15         solve_bfs(Paths1,Solution).
16
17 extend_bfs([Node|Path], NewPaths) :-
18         findall([NewNode, Node|Path],
19         (adjacente(Node, NewNode,_),
20         nao(membro(NewNode,[Node|Path]))),
21         NewPaths),!.
22
23 extend_bfs(Path,_).
24
25 bfs_todas_solucoes(L):- findall((F,C),resolve_bfs(N,F),comprimento(F
       ,C),L).
26
27 bfs_min(C,A):- findall((F,X),(resolve_bfs(N,F),comprimento(F,X)),L),
       pairwise_min(L,(C,A)).
28
29 bfs_max(C,A):- findall((F,X),(resolve_bfs(N,F),comprimento(F,X)),L),
       pairwise_max(L,(C,A)).
30
31 % ------------------ PESQUISA POR RESIDUO ------------------
32
33 resolve_bfs_tipo(Nodo,Final,T,L) :- inicial(Nodo),solve_bfs_tipo([[
       Nodo]],F,T),reverseL(F,Final),comprimento(Final,L).
34
35 solve_bfs_tipo([[N|Path]|_],[N|Path],T) :- final(N).
36
37 solve_bfs_tipo([Path|Paths],Solution,T) :-
38         extend_bfs_tipo(Path,NewPaths,T),
39         append(Paths,NewPaths,Paths1),
40         solve_bfs_tipo(Paths1,Solution,T).
41
42 extend_bfs_tipo([Node|Path], NewPaths,T) :-
43         findall([NewNode, Node|Path],
44         (adjacenteTipo(Node, NewNode,_,T),
45         nao(membro(NewNode,[Node|Path]))),
46         NewPaths),!.
47
48 extend_bfs_tipo(Path,_,T).
```

Listing 8: Algoritmos Breadth-First

No entanto, tendo em conta a dimensão do grafo e a sua profundidade, este algoritmo acaba por se revelar ineficiente dado os gastos de memória necessários para

calcular todas as soluções possíveis para o grafo construído.

- **Busca Iterativa Limitada em Profundidade**

A busca iterativa limitada em profundidade é uma combinação dos dois algoritmos previamente apresentados. Passamos então a ter um algoritmo combinado que acaba por apresentar melhores resultados a diferentes níveis. Por exemplo, a sua complexidade espacial é *O(bM)* e a nível temporal apresenta um tempo de pesquisa $O(b^M)$,onde **b** é o fator de ramificação e **M** é a profundidade máxima que o algoritmo deverá percorrer.

Foram implementados quatro algoritmos diferentes: um cuja solução é de acordo com o número de arcos, outro cuja solução é em função da distância e os últimos dois são semelhantes mas têm em consideração o tipo de resíduo a recolher.

```
% --------------------------------
%   Procura Depth-First Limitada
% --------------------------------

% ------------------ PESQUISA NORMAl ------------------

% Depth-first com profundidade limitada que da solucoes de acordo
    com o numero de arcos
resolvedf_arcos_profundidade([X|Caminho],Prof):-
                inicial(X),
                resolvedf_arcos_profundidade(X,[X],Caminho,Prof).

resolvedf_arcos_profundidade(A, _ ,[],_):- final(A),!.
resolvedf_arcos_profundidade(A, Historico, [Prox|Caminho],ProfMax):-
                ProfMax > 0,
                adjacente(A,Prox,_),
                nao(membro(Prox,Historico)),
                NewProf is ProfMax - 1,
                resolvedf_arcos_profundidade(Prox, [Prox|Historico],
     Caminho,NewProf).

df_TodasSolucoes_arcos_profundidade(L,Prof):- findall((C,N),(
    resolvedf_arcos_profundidade(C,Prof),comprimento(C,N)),L).

df_Arcos_profundidade_min(X,Prof):- findall((P,C),(
    resolvedf_arcos_profundidade(P,Prof),comprimento(P,C)),L),
    pairwise_min(L,X).

df_Arcos_profundidade_max(X,Prof):- findall((P,C),(
    resolvedf_arcos_profundidade(P,Prof),comprimento(P,C)),L),
    pairwise_max(L,X).

```

```
% Depth-first com profundidade limitada que da solucoes de acordo com a distancia
resolvedf_dist_profundidade(Nodo,[Nodo|Caminho],Dist,Prof):-
                inicial(Nodo),
                df_dist_profundidade(Nodo,[Nodo],Caminho,Dist,Prof).

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

df_dist_profundidade(Nodo,Historico,[NodoProx|Caminho],Dist,ProfMax) :-
                ProfMax > 0,
                adjacente(Nodo,NodoProx,Dist1),
                nao(membro(NodoProx, Historico)),
                NewProf is ProfMax - 1,
                df_dist_profundidade(NodoProx,[NodoProx|Historico], Caminho,Dist2,NewProf),
                Dist is Dist1 + Dist2.

df_DistTodasSolucoes_profundidade(L,P):- findall((S,D),( resolvedf_dist_profundidade(N,S,D,P)),L).

df_Dist_melhor_profundidade(Nodo,Cam,Dist,P):- findall((Ca,D), resolvedf_dist_profundidade(Nodo,Ca,D,P), L),pairwise_min(L,(Cam, Dist)).

% ------------------ PESQUISA POR RESIDUO ------------------

% Depth-first com profundidade limitada que da solucoes de acordo com o numero de arcos
resolvedf_arcos_profundidade_tipo([X|Caminho],Prof,T):-
                inicial(X),
                resolvedf_arcos_profundidade_tipo(X,[X],Caminho,Prof,T).

resolvedf_arcos_profundidade_tipo(A, _ ,[],_,_):- final(A),!.
resolvedf_arcos_profundidade_tipo(A, Historico, [Prox|Caminho], ProfMax,T):-
                ProfMax > 0,
                adjacenteTipo(A,Prox,_,T),
                nao(membro(Prox,Historico)),
                NewProf is ProfMax - 1,
                resolvedf_arcos_profundidade_tipo(Prox, [Prox| Historico], Caminho,NewProf,T).

df_TodasSolucoes_arcos_profundidade_tipo(L,Prof,T):- findall((C,N),( resolvedf_arcos_profundidade_tipo(C,Prof,T),comprimento(C,N)),L).

```

```
63 df_Arcos_profundidade_melhor_tipo(X,Prof,T):- findall((P,C),(
       resolvedf_arcos_profundidade_tipo(P,Prof,T),comprimento(P,C)),L),
       pairwise_max(L,X).
64
65 % Depth-first com profundidade limitada que da solucoes de acordo
       com a distancia
66 resolvedf_dist_profundidade_tipo(Nodo,[Nodo|Caminho],Dist,Prof,T):-
67                 inicial(Nodo),
68                 df_dist_profundidade_tipo(Nodo,[Nodo],Caminho,Dist,
       Prof,T).
69
70 df_dist_profundidade_tipo(Nodo, _, [], 0,_,_):-
71                 final(Nodo).
72
73 df_dist_profundidade_tipo(Nodo,Historico,[NodoProx|Caminho],Dist,
       ProfMax,T):-
74                 ProfMax > 0,
75                 adjacenteTipo(Nodo,NodoProx,Dist1,T),
76                 nao(membro(NodoProx, Historico)),
77                 NewProf is ProfMax - 1,
78                 df_dist_profundidade_tipo(NodoProx,[NodoProx|
       Historico],Caminho,Dist2,NewProf,T),
79                 Dist is Dist1 + Dist2.
80
81 df_DistTodasSolucoes_profundidade_tipo(L,P,T):- findall((S,D),(
       resolvedf_dist_profundidade_tipo(N,S,D,P,T)),L).
82
83 df_Dist_melhor_profundidade_tipo(Nodo,Cam,Dist,P,T):- findall((Ca,D)
       , resolvedf_dist_profundidade_tipo(Nodo,Ca,D,P,T), L),
       pairwise_min(L,(Cam,Dist)),imprime(Cam).
```

Listing 9: Algoritmos de Profundidade Limitada

4.3.2 Pesquisa Informada

Os algortimos de pesquisa informada fazem uso de heurísticas como forma de auxílio à pesquisa. Duas dessas heurísticas são: gulosa e A*.

- **Gulosa**

O algoritmo que aplica a heurística da gulosa tenta resolver o problema fazendo a escolha ótima em cada iteração, com a esperança de encontrar um ótimo a nível global. Neste, as escolhas realizadas são definitivas, não tem em consideração consequências futuras, pode fazer cálculos repetitivos e nem sempre produz a solução ótima. No entanto, quanto mais informação lhe for transmitido maior as possibilidades de encontrar a melhor solução.

Foram implementadas duas versões deste algoritmo: uma normal, e uma que tem em conta o tipo de resíduo nos pontos.

```
% --------------------------
%       Pesquisa Gulosa
% --------------------------

% ------------------ PESQUISA NORMAl ------------------

resolve_gulosa(Nodo,Caminho/Custo):-
        estima(Nodo,Estima),
        agulosa([[Nodo]/0/Estima],InvCaminho/Custo/_),
        reverseL(InvCaminho,Caminho),!.

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

agulosa(Caminhos, SolucaoCaminho):-
        obtem_melhor_g(Caminhos,MelhorCaminho),
        escolhe(MelhorCaminho,Caminhos,OutrosCaminhos),
        expande_gulosa(MelhorCaminho,ExpCaminhos),
        append(OutrosCaminhos,ExpCaminhos,NovoCaminhos),
        agulosa(NovoCaminhos,SolucaoCaminho).

expande_gulosa(Caminho,ExpCaminhos):-
        findall(NovoCaminho,adjacente_gulosa(Caminho,NovoCaminho),
    ExpCaminhos).

adjacente_gulosa([Nodo|Caminho]/Custo/_,[ProxNodo,Nodo|Caminho]/
    NovoCusto/Est):-
        adjacente(Nodo,ProxNodo,PassoCusto),
        nao(membro(ProxNodo,Caminho)),
        NovoCusto is Custo + PassoCusto,
        estima(ProxNodo,Est).

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).

```

```
42
43 % ------------------ PESQUISA POR RESIDUO ------------------
44
45 resolve_gulosa_tipo(Nodo,Caminho/Custo,T):-
46         estimaTipo(Nodo,Estima,T),
47         agulosa_tipo([[Nodo]/0/Estima],InvCaminho/Custo/_,T),
48         reverseL(InvCaminho,Caminho),!.
49
50 agulosa_tipo(Caminhos,Caminho,T):-
51         obtem_melhor_g(Caminhos,Caminho),
52         Caminho = [Nodo|_]/_/_,
53         final(Nodo).
54
55 agulosa_tipo(Caminhos, SolucaoCaminho,T):-
56         obtem_melhor_g(Caminhos,MelhorCaminho),
57         escolhe(MelhorCaminho,Caminhos,OutrosCaminhos),
58         expande_gulosa_tipo(MelhorCaminho,ExpCaminhos,T),
59         append(OutrosCaminhos,ExpCaminhos,NovoCaminhos),
60         agulosa_tipo(NovoCaminhos,SolucaoCaminho,T).
61
62 expande_gulosa_tipo(Caminho,ExpCaminhos,T):-
63         findall(NovoCaminho,adjacente_gulosa_tipo(Caminho,
    NovoCaminho,T),ExpCaminhos).
64
65 adjacente_gulosa_tipo([Nodo|Caminho]/Custo/_,[ProxNodo,Nodo|Caminho
    ]/NovoCusto/Est,T):-
66         adjacente(Nodo,ProxNodo,PassoCusto),
67         nao(membro(ProxNodo,Caminho)),
68         NovoCusto is Custo + PassoCusto,
69         estimaTipo(ProxNodo,Est,T).
70
71 obtem_melhor_g([Caminho],Caminho):-!.
72
73 obtem_melhor_g([Caminho1/Custo1/Est1,_/Custo2/Est2|Caminhos],
    MelhorCaminho):-
74         Est1 =< Est2 , !,
75         obtem_melhor_g([Caminho1/Custo1/Est1|Caminhos],MelhorCaminho
    ).
76
77 obtem_melhor_g([_|Caminhos],MelhorCaminho):-
78         obtem_melhor_g(Caminhos,MelhorCaminho).
79
80
81 descobre_todas_gulosa_tipo(S,T):- findall((N,Caminho/Custo/Estima/T)
    ,resolve_gulosa_tipo(N,Caminho/Custo/Estima/T,T),S).
```

Listing 10: Algoritmos Gulosa

- **A-Estrela (A*)**

O algoritmo A-Estrela representa uma combinação das aproximações heurísticas do algoritmo de pesquisa em largura (BFS) e da formalidade do algoritmo de Dijkstra.

Foram implementadas duas versões deste algoritmo: uma normal e uma refente ao tipo de resíduo num dado ponto.

```
% -------------------------
%    Pesquisa A Estrela
% -------------------------

% ------------------ PESQUISA NORMAl ------------------

resolve_aestrela(Nodo,Caminho/Custo):-
        estima(Nodo,Estima),
        aestrela([[Nodo]/0/Estima],InvCaminho/Custo/_),
        reverseL(InvCaminho,Caminho),!.

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

aestrela(Caminhos,SolucaoCaminho):-
        obtem_melhor(Caminhos,MelhorCaminho),
        escolhe(MelhorCaminho,Caminhos,OutrosCaminhos),
        expande_aestrela(MelhorCaminho,ExpCaminhos),
        append(OutrosCaminhos,ExpCaminhos,NovoCaminhos),
        aestrela(NovoCaminhos,SolucaoCaminho).

expande_aestrela(Caminho,ExpCaminhos):-
        findall(NovoCaminho,adjacente_aestrela(Caminho,NovoCaminho),
    ExpCaminhos).

adjacente_aestrela([Nodo|Caminho]/Custo/_,[ProxNodo,Nodo|Caminho]/
    NovoCusto/Est):-
        adjacente(Nodo,ProxNodo,PassoCusto),
        nao(membro(ProxNodo,Caminho)),
        NovoCusto is Custo + PassoCusto,
        estima(ProxNodo,Est).

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).

% ------------------ PESQUISA POR RESIDUO ------------------

resolve_aestrela_tipo(Nodo,Caminho/Custo,T):-
        estimaTipo(Nodo,Estima,T),
        aestrela_tipo([[Nodo]/0/Estima],InvCaminho/Custo/_,T),
        reverseL(InvCaminho,Caminho),!.

aestrela_tipo(Caminhos,Caminho,T):-
        obtem_melhor(Caminhos,Caminho),
        Caminho = [Nodo|_]/_/_,
        final(Nodo).

aestrela_tipo(Caminhos,SolucaoCaminho,T):-
        obtem_melhor(Caminhos,MelhorCaminho),
        escolhe(MelhorCaminho,Caminhos,OutrosCaminhos),
        expande_aestrela_tipo(MelhorCaminho,ExpCaminhos,T),
        append(OutrosCaminhos,ExpCaminhos,NovoCaminhos),
        aestrela_tipo(NovoCaminhos,SolucaoCaminho,T).

adjacenteEstrela_tipo([Nodo|Caminho]/Custo/_,[ProxNodo,Nodo|Caminho
    ]/NovoCusto/Est,T):-
        adjacente(Nodo,ProxNodo,PassoCusto),
        nao(membro(ProxNodo,Caminho)),
        NovoCusto is Custo + PassoCusto,
        estimaTipo(ProxNodo,Est,T).

expande_aestrela_tipo(Caminho,ExpCaminhos,T):-
        findall(NovoCaminho,adjacenteEstrela_tipo(Caminho,
    NovoCaminho,T),ExpCaminhos).

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).
```

Listing 11: Algoritmo A-Estrela

- **Heurística Escolhida**

A heurística escolhida para este trabalho foi a distância real entre as coordenadas de latitude e longitude de dois pontos, uma vez que esse é o maior fator em

consideração ao longo da elaboração deste trabalho.

4.4 Predicados Auxiliares

Na presente secção apresentarei alguns predicados auxiliares utilizados ao longo do trabalho, especialmente para a elaboração dos algoritmos.

4.4.1 Predicados Gerais

```
% ---------------------------------------------------------------------
%                          Predicados Uteis
% ---------------------------------------------------------------------
% Extensao do meta-predicado nao
nao( Questao ) :-
    Questao, !, fail.
nao( _ ).

% 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).

% Tira o elemento D de uma lista
escolhe(E,[E|Xs],Xs).
escolhe(E,[X|Xs],[X|Ys]):- escolhe(E,Xs,Ys).

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

% Verifica se um dado elemento ja pertence a lista
membro(X, [X|_]).
membro(X, [_|Xs]):-
  membro(X, Xs).

% Predicado que devolve o par de menor valor no 2o elemento de uma
   lista
pairwise_min(L, (A,B)) :-
    select((A,B), L, R),
    \+ ( member((A1,B1), R), B1 < B ).

% Predicado que devolve o par de maior valor no 2o elemento de uma
   lista
```

```
pairwise_max(L, (A,B)) :-
    select((A,B), L, R),
    \+ ( member((A1,B1), R), B1 > B ).

% Permite limpar o terminal do PROLOG
cls :- write('\33\[2J').

% Predicado que permite medir o tempo de execucao de um outro predicado
measuretime(Predicado):- statistics(walltime, [TimeSinceStart | [TimeSinceLastCall]]),
                         Predicado,
                         statistics(walltime, [NewTimeSinceStart | [ExecutionTime]]),
                         write('Execution took '), write(ExecutionTime), write(' ms.'), nl.
```

Listing 12: Predicados Gerais

4.4.2 Predicados Sobre Arcos

```
% Obtem o arco entre dois pontos
% arco(Ponto1,Ponto2,Distancia)
getArco(Origem,Destino,Dist) :- arco(Origem,Destino,Dist).

% Predicado que verifica se dois pontos sao adjacentes
adjacente(A,B,C) :- getArco(A,B,C).
adjacente(A,B,C) :- getArco(B,A,C).

% Predicado que verifica se um ponto de recolha contem um tipo de residuos
isTipo(Nodo,Tipo):- ponto_recolha(Nodo,_,_,Tipo,_).

% Predicado que verifica se dois pontos de recolha sao adjacentes e de um dado tipo de residuos
adjacenteTipo(A,B,C,T) :- isTipo(A,T), isTipo(B,T),getArco(A,B,C).
adjacenteTipo(A,B,C,T) :- isTipo(A,T), isTipo(B,T),getArco(B,A,C).
```

Listing 13: Predicados Sobre Arcos

4.4.3 Predicados Sobre Pontos de Recolha

```
% ponto_recolha(Nome,Lat,Long,Tipo,QtdTotal)
getPonto(Nome,Lat,Long,Tipo,Qtd):- ponto_recolha(Nome,Lat,Long,Tipo,Qtd).

```

```
% Obt m a capacidade num dado ponto de um dado tipo
getCapacidadeTipo(Nodo,Capacidade,T):- ponto_recolha(Nodo,_,_,T,
    Capacidade).

% Obt m a capacidade total num dado ponto indiferente ao tipo
somaCapacidades([],0).
somaCapacidades([X|T],Capacidade):- somaCapacidades(T,Y), Capacidade
     is X + Y.

getCapacidadeTotal(Nodo,Capacidade) :- findall( Cap,ponto_recolha(
    Nodo,_,_,_,Cap),L), somaCapacidades(L,Capacidade).

distance(Origem, Dis):-
     ponto_recolha(Origem,Lat1,Lon1,_,_),
     final(Destino),
     ponto_recolha(Destino,Lat2,Lon2,_,_),
     P is 0.017453292519943295,
     A is (0.5 - cos((Lat2 - Lat1) * P) / 2 + cos(Lat1 * P) * cos(
    Lat2 * P) * (1 - cos((Lon2 - Lon1) * P)) / 2),
     Dis1 is (12742 * asin(sqrt(A))),
     Dis is Dis1 * 1000.


estimaTipo(Nodo,Estima,Tipo) :- ponto_recolha(Nodo,_,_,Tipo,_),
    distance(Nodo,Estima).

estima(Nodo,Estima) :- distance(Nodo,Estima).
```

Listing 14: Predicados Sobre Pontos de Recolha

4.5 Funcionalidades Adicionais

Na presente secção passarei a apresentar algumas funcionalidades adicionais simples que decidi adicionar.

```
% --------------------------
%   Funcionalidades Extra
% --------------------------
getTotalPontosRecolha(Qtd) :- findall(Nodo,ponto_recolha(Nodo,_,_,_,
    _),R),comprimento(R,Qtd).

getTotalPontosRecolhaTipo(Tipo,Qtd) :- findall(Nodo,ponto_recolha(
    Nodo,_,_,Tipo,_),R),comprimento(R,Qtd).

getTotalArcos(Qtd) :- findall(Nodo,arco(Nodo,_,_),R),comprimento(R,
    Qtd).
```

Listing 15: Funcionalidades Adicionais

5 Análise de Resultados

Estando construídos os algoritmos devemos agora fazer uma breve comparação entre os mesmos a nível de resultados. Para tal, vamos considerar um ponto inicial na *"R do Alecrim"* e um ponto final na *"R Sao Paulo"* e executar todos os algoritmos normais sobre estes dois pontos. Obtemos então os seguintes resultados:

Estratégia	Tempo(s)	Espaço	Profundidade/Custo	Melhor Solução?
DFS	1.7	O(bm)	24/340.606	Sim
BFS	-	O(bˆd)	-	Desconhecido
DFS - Limitada (20)	0.08	O(bM)	21/340.606	Sim
Gulosa	0.001	-	-/399.87	Possivelmente
A-Estrela	0.029	-	-/340.606	Sim

Como mencionado antes, a procura em largura é demasiado expendiosa em termos de memória dado o grafo com que nos encontramos a trabalhar, por isso fui incapaz de encontrar a melhor solução.

Relativamente aos restantes quatro algoritmos os resultados obtidos são bastante parecidos, no entanto o tempo de execução é algo a ter em conta. Apesar da Gulosa ser a mais rápida, não podemos ter a certeza de que a solução é a ótima, dada a natureza do algoritmo.

No que diz respeito à DFS-Limitada e à A-Estrela, os tempos de execução são muito próximos e apresentam a mesma solução ótima, o que também acontece com a DFS mas que acaba por demorar quase dois segundos a executar.

De seguida, e tendo em conta estes resultados, irei responder a certos requisitos mencionados no enunciado.

5.1 Questões a Responder

Nas seguintes questões irei usar como ponto inicial a *"R do Alecrim"* e como ponto de chegada *"R Sao Paulo"*. De notar que, dado o nível de abstração até agora, considerei que o camião de recolha parte do ponto inicial, ou seja, é a garagem onde o mesmo se encontra, e deposita o lixo no ponto de chegada.

5.1.1 Gerar os circuitos de recolha tanto indiferenciada como seletiva, caso existam, que cubram um determinado território

De forma a testar este requisito irei fazer uso do algoritmo DFS, quer indiferenciado, quer em relação a um tipo de resíduo específico, neste caso 'Papel e Cartao'.

```
[1]  ?- df_Arcos_max(X).
X =  (['R do Alecrim', 'Pc Duque da Terceira', 'R Bernardino da Costa', 'Lg Corpo San
to', 'R Corpo Santo', 'Tv Corpo Santo', 'Cais do Sodre', 'Av 24 de Julho'|...], 24) ;
```

Figura 2: Pesquisa DFS indiferenciada

```
[1]  ?- df_Arcos_max_tipo(X,'Papel e Cartao').
X =  (['R do Alecrim', 'R Nova do Carvalho', 'Tv Corpo Santo', 'R Bernardino da Costa
', 'Lg Corpo Santo', 'Cais do Sodre', 'Pc Duque da Terceira', 'R Remolares'|...], 13)
 ;
```

Figura 3: Pesquisa DFS por Resíduo 'Papel e Cartao'

Caso seja pretendido averiguar outro território a ser alvo de pesquisa, devemos trocar de ponto inicial e ponto de chegada.

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

Novamente, para responder a este requisito irei fazer uso do algoritmo DFS, com os tipos de resíduo 'Lixos' e 'Papel e Cartão'.

```
?- df_Arcos_max_tipo(X,'Lixos').
X =  (['R do Alecrim', 'Pc Duque da Terceira', 'R Bernardino da Costa', 'Lg Corpo San
to', 'R Corpo Santo', 'Tv Corpo Santo', 'Cais do Sodre', 'Av 24 de Julho'|...], 24) ;
```

Figura 4: Pesquisa DFS por Resíduo 'Lixos'

```
[1]  ?- df_Arcos_max_tipo(X,'Papel e Cartao').
X =  (['R do Alecrim', 'R Nova do Carvalho', 'Tv Corpo Santo', 'R Bernardino da Costa
', 'Lg Corpo Santo', 'Cais do Sodre', 'Pc Duque da Terceira', 'R Remolares'|...], 13)
 ;
```

Figura 5: Pesquisa DFS por Resíduo 'Papel e Cartao'

5.1.3 Comparar circuitos de recolha tendo em conta os indicadores de produtividade

Tendo em conta os indicadores de produtividade mencionados no enunciado, decidi utilizar o algoritmo DFS para comparar os circuitos de recolha.

```
[1]  ?- df_Cap_melhor(N,Cam,Cap).
R do Alecrim
Pc Duque da Terceira
Cais do Sodre
Lg Corpo Santo
R Bernardino da Costa
Tv Corpo Santo
R Nova do Carvalho
Tv dos Remolares
R Remolares
Tv Ribeira Nova
Pc Sao Paulo
Tv Carvalho
R Ribeira Nova
Pc Dom Luis I
Av 24 de Julho
R Instituto Industrial
R Dom Luis I
Bqr do Duro
R Cais do Tojo
Tv do Cais do Tojo
Lg Conde-Barao
R da Boavista
R Sao Paulo
Cam = ['R do Alecrim', 'Pc Duque da Terceira', 'Cais do Sodre', 'Lg Corpo Santo', 'R
Bernardino da Costa', 'Tv Corpo Santo', 'R Nova do Carvalho', 'Tv dos Remolares', 'R
Remolares'|...],
Cap = 158290 ;
```

Figura 6: Pesquisa DFS por Capacidade

Como termo de comparação alterei o ponto de chegada para "Tv Corpo Santo".

```
[1]  ?- df_Cap_melhor(N,Cam,Cap).
R do Alecrim
R Nova do Carvalho
Tv dos Remolares
R Remolares
Tv Ribeira Nova
Pc Sao Paulo
Tv Carvalho
R Ribeira Nova
Pc Dom Luis I
R Moeda
R Sao Paulo
R da Boavista
Lg Conde-Barao
Tv do Cais do Tojo
R Cais do Tojo
Bqr do Duro
R Dom Luis I
R Instituto Industrial
Av 24 de Julho
Cais do Sodre
Pc Duque da Terceira
R Bernardino da Costa
Lg Corpo Santo
R Corpo Santo
Tv Corpo Santo
Cam = ['R do Alecrim', 'R Nova do Carvalho', 'Tv dos Remolares', 'R Remolares', 'Tv R
ibeira Nova', 'Pc Sao Paulo', 'Tv Carvalho', 'R Ribeira Nova', 'Pc Dom Luis I'|...],
Cap = 166490 ,
```

Figura 7: Pesquisa DFS por Capacidade com Ponto de Chegada diferente

5.1.4 Escolher o circuito mais rápido (usando o critério da distância)

Para escolher o circuito mais rápido em termos de distância faremos uso do algoritmo A-Estrela, uma vez que a heurística escolhida para os mesmos foi exatamente essa.

```
[1]  ?- resolve_aestrela(N,R).
N = 'R do Alecrim',
R = ['R do Alecrim', 'R Nova do Carvalho', 'Tv dos Remolares', 'R Sao Paulo']/340.606
17301582374.
```

Figura 8: Pesquisa A-Estrela com Distância como Heurística

5.1.5 Escolher o circuito mais eficiente (usando um critério de eficiência à escolha)

Como critério de eficiência, tendo em conta que se trata da versão simplificada do trabalho, decidi escolher a quantidade de lixo recolhida num circuito. Para tal utilizarei o algoritmo DFS.

```
[1]  ?- df_Cap_melhor(N,Cam,Cap).
R do Alecrim
Pc Duque da Terceira
Cais do Sodre
Lg Corpo Santo
R Bernardino da Costa
Tv Corpo Santo
R Nova do Carvalho
Tv dos Remolares
R Remolares
Tv Ribeira Nova
Pc Sao Paulo
Tv Carvalho
R Ribeira Nova
Pc Dom Luis I
Av 24 de Julho
R Instituto Industrial
R Dom Luis I
Bqr do Duro
R Cais do Tojo
Tv do Cais do Tojo
Lg Conde-Barao
R da Boavista
R Sao Paulo
Cam = ['R do Alecrim', 'Pc Duque da Terceira', 'Cais do Sodre', 'Lg Corpo Santo', 'R
Bernardino da Costa', 'Tv Corpo Santo', 'R Nova do Carvalho', 'Tv dos Remolares', 'R
Remolares'|...],
Cap = 158290 ;
```

Figura 9: Pesquisa DFS por Capacidade

6 Conclusão

Dando o trabalho por concluído, e apesar de ser a versão simplificada do mesmo, sinto que consegui concluir o mesmo com sucesso tendo em conta os requisitos mencionados no enunciado e o trabalho que apresentei.

Sinto também que este trabalho me ajudou a aprofundar não só o conhecimento sobre a linguagem PROLOG, mas também a minha interpretação e análise de dados, os meus conhecimentos sobre os algoritmos explorados e a minha capacidade de tomar decisões e estruturar um trabalho sozinho de início ao fim.

Em termos de algoritmos, dada a dimensão do trabalho, o nível de abstração que coloquei sobre o mesmo permitiu-me concluir que para diferentes cenários existem diversas soluções, não havendo necessariamente uma resposta concreta a *qual o melhor algoritmo a ser usado no caso estudado*, uma vez que devemos ter em conta diversos fatores como: a capacidade máxima de um camião (algo que acabou por não ser explorado), a distância média percorrida num circuito e a quantidade recolhida durante o mesmo.

Assim, sinto-me confiante no trabalho que apresentei e acho que o mesmo corresponde quer às minhas expectativas quando fui primeiramente enfrentado com o enunciado, quer ao que nos era pedido durante toda a unidade curricular.

Referências

[1] SWI Prolog Manual : https://www.swi-prolog.org/pldoc/man?section=builtin