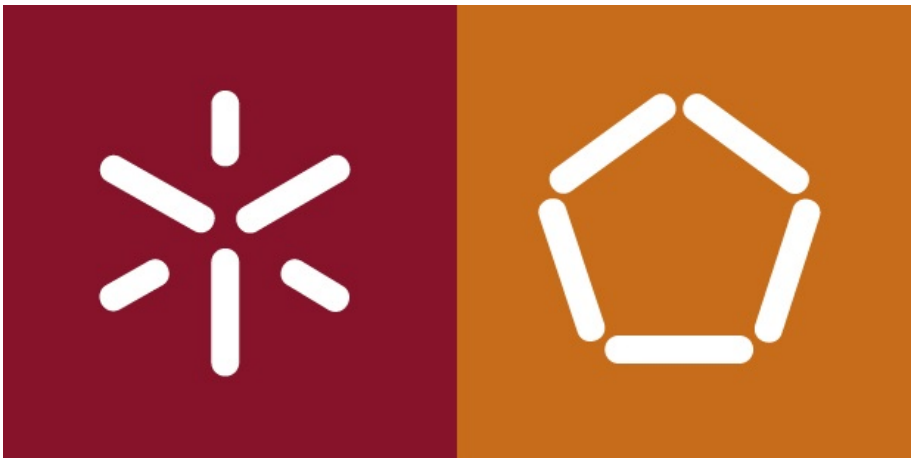
Universidade do Minho

Mestrado Integrado em Engenharia Informática



Sistemas de Representação de Conhecimento e Raciocínio

Métodos de Resolução de Problemas e de Procura

Trabalho realizado por: *Número*

Adriana Martins Gonçalves A75119

Braga, 7 de junho de 2021

1 Resumo

No âmbito da unidade curricular de **Sistemas de Representação de Conhecimento e Raciocínio**, foi proposto o desenvolvimento de um trabalho prático individual cuja tema é o estudo dos circuitos de recolha de resíduos urbanos do concelho de Lisboa.

Para a elaboração deste projeto será utilizada a linguagem de programação lecionada ao longo de toda a UC, neste caso o *PROLOG*, e ainda a utilização de *Java* para a definição do *parser* e assim ser possível fazer a leitura dos dados fornecidos.

No presente relatório serão explicadas todas as decisões e abordagens tomadas para a obtenção dos resultados pretendidos, bem como uma explicação das diferentes estratégias de procura (informada e não-informada).

Conteúdo

1 Resumo **1**

2 Lista de Figuras **3**

3 Introdução **4**

4 Base de Conhecimento **5**

5 Algoritmos de Pesquisa **7**

6 Implementação da Solução **8**

6.1 Query 1: Gerar os circuitos de recolha tanto indiferenciada como seletiva, caso existam, que cubram um determinado território 8

6.2 Query 2: Identificar quais os circuitos com mais pontos de recolha (por tipo de resíduo a recolher) 8

6.3 Query 3: Comparar circuitos de recolha tendo em conta os indicadores de produtividade 10

6.4 Query 4: Escolher o circuito mais rápido (usando o critério da distância) 11

6.5 Query 5: Escolher o circuito mais eficiente (recolha de resíduos 'Orgânicos') . . 12

7 Pesquisa Informada **13**

8 Predicados auxiliares **13**

9 Conclusão **14**

10 Referências **15**

2 Lista de Figuras

3.1 Código do Main.java 6
3.2 Tabela comparativa da utilização de algoritmos de pesquisa não-informada 7
3.3 Tabela explicativa relativa aos algoritmos de pesquisa informada 7
3.4 Código do predicado geraCircuito 8
3.5 Código do predicado RecolhaLixo 8
3.6 Código do predicado RecolhaPapelCartao 9
3.7 Código do predicado RecolhaEmbalagens 9
3.8 Código do predicado RecolhaOrganico 9
3.9 Código do predicado RecolhaVidro 9
3.10 Código do predicado maximoPontosRecolha 10
3.11 Código do predicado circuitoMaisRapido 11
3.12 Código do predicado circuitoOrganico 12
3.13 Código dos predicados auxiliares 12

3 Introdução

O presente trabalho tem como principal objetivo o estudo dos circuitos efetuados no concelho de Lisboa para a recolha dos seus resíduos urbanos, e por sua vez, a implementação de diversas estratégias de pesquisa, através da utilização da linguagem *PROLOG*.

O trabalho é composto por pontos de recolha, representando todas as informações relativas a cada ponto, e arcos, que representam as conexões possíveis entre cada ponto de recolha. O teste de toda esta informação é possível através do processamento do *dataset* fornecido. Para o desenvolvimento deste trabalho, decidi não estender o conhecimento, visto que a quantidade de dados fornecida pelos docentes já é suficiente. Decidi também não ter em conta o fator de carga máxima do veículo, tendo assim o camião capacidade de recolha ilimitada, não entrando este indicador para a produtividade de recolha.

Para a definição dos arcos adjacentes entre todos os pontos de recolha de resíduos foi definido um *parser*, e para responder a todas as *queries* recorri a algoritmos de pesquisa informada e não-informada, sendo que será dada uma breve explicação de ambas as estratégias bem como as vantagens que cada um apresenta para o trabalho em questão.

4 Base de Conhecimento

Como referido anteriormente, optei por não entender o conhecimento visto que a quantidade de dados fornecida pelos docentes já ser suficiente. Para a definição da base de conhecimento, foi necessário analisar o excel fornecido para que toda a informação fosse organizada de forma a ser possível ser utilizada pelo *PROLOG* posteriormente.

Desta forma, optei por criar um ***parser***, desenvolvido no ficheiro **Main.java**, para que fosse possível ler todos os dados e, posteriormente, escreve-los num outro ficheiro (**circuitos.pl**). De todo o conjunto de informação, optei por separar o ID da rua (*'ID_Rua'*) do restante nome do local (*Pt_Recolha_Local*), visto que, como exposto na imagem 1 referente ao *parser*, as adjacências serão definidas através do ID das ruas. A minha ideia era formar as adjacências quando um ID de uma rua é igual ao ID de outra, ou quando o ID de uma rua é anterior ao ID de outra, ou finalmente o ID de uma rua ser posterior ao ID de outra. Infelizmente, apenas consegui definir para os casos em que os ID's das ruas são iguais e assim ficaram definidas as ruas adjacentes.

- localrecolha(-9.14751690960422;38.7146089969724;39;Misericórdia;19365;R O Século 150; Lixos;CV0240;240;4;960)

que representa todas as características de um ponto de recolha.

- adjacente(localrecolha(-9.14751690960422,38.7146089969724,39,'Misericórdia',19365,'R O Século 150','Lixos','CV0240',240,4,960),localrecolha(-9.14751690960422,38.7146089969724, 40,'Misericórdia',19365,'R O Século 150','Lixos','CV0240',240,4,960)).

que representa dois pontos de recolha adjacentes, e consequentemente, há uma ligação entre eles.

O **'localrecolha'** tem como parâmetros uma latitude, uma longitude, um ID, a freguesia a que são referentes os ponto de recolha (PONTO_RECOLHA_FREGUESIA), o ID da rua (ID_Rua), o nome da rua desses pontos de recolha (PONTO_RECOLHA_LOCAL), o que cada contentor armazena em termos de resíduos (se lixos, papel e cartão, embalagens, vidro, etc) (CONTENTOR_RESÍDUO), o tipo de contentor (CONTENTOR_TIPO), a capacidade (CONTENTOR_CAPACIDADE), a quantidade (CONTENTOR_QT) e o total de litros de cada contentor de acordo com a quantidade e capacidade dos contentores de um determinado local (CONTENTOR_TOTAL_LITROS).

O ***parser*** foi escrito em java visto que esta linguagem possui diversas vantagens relativamente à manipulação de dados. Este está escrito da seguinte forma:

- Todos os locais de recolha foram ligados às linhas diretamente acima e abaixo através do ID das ruas, de forma a descobrir os pontos adjacentes. Estas ligações depois de conectadas serão apresentadas num outro ficheiro de escrita.

Desta forma, foi possível definir a base de conhecimento (definido no ficheiro **circuitos.pl**) e avançar no desenvolvimento do trabalho.

```java
package com.company;
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;

public class Main {
    Run | Debug
    public static void main(String[] args) {
        try {
            FileReader fr = new FileReader("pts_recolha.csv");

            try (BufferedReader br = new BufferedReader(fr)) {
                ArrayList<String> linhas = new ArrayList<>();
                String[] parte1;
                String[] parte2;
                String linha;
                FileWriter fw = new FileWriter("circuitos.pl");

                while ((linha = br.readLine()) != null) {
                    linhas.add(linha);
                }
                fr.close();
                fw.write(":- dynamic adjacente/2.\n");
                fw.write(":- dynamic localrecolha/11.\n");
                for (int i = 1; i < linhas.size() - 1; i++) {
                    parte1 = linhas.get(i).split(";");
                    parte2 = linhas.get(i + 1).split(";");
                    if (parte1[4].equals(parte2[4])) {
                        fw.write("adjacente(");
                        fw.write("localrecolha(" + parte1[0] + "," + parte1[1] + "," + parte1[2] + ",'" + parte1[3] + "'," + parte1[4] + ",
                        '" + parte1[5] + "','" + parte1[6] + "','" + parte1[7] + "'," + parte1[8] + "," + parte1[9] + "," + parte1[10] + "),");
                        fw.write("localrecolha(" + parte2[0] + "," + parte2[1] + "," + parte2[2] + ",'" + parte2[3] + "'," + parte2[4] + ",
                        '" + parte2[5] + "','" + parte2[6] + "','" + parte2[7] + "'," + parte2[8] + "," + parte2[9] + "," + parte2[10] + ")");
                        fw.write(").\n");
                        fw.write("adjacente(");
                        fw.write("localrecolha(" + parte2[0] + "," + parte2[1] + "," + parte2[2] + ",'" + parte2[3] + "'," + parte2[4] + ",
                        '" + parte2[5] + "','" + parte2[6] + "','" + parte2[7] + "'," + parte2[8] + "," + parte2[9] + "," + parte2[10] + "),");
                        fw.write("localrecolha(" + parte1[0] + "," + parte1[1] + "," + parte1[2] + ",'" + parte1[3] + "'," + parte1[4] + ",
                        '" + parte1[5] + "','" + parte1[6] + "','" + parte1[7] + "'," + parte1[8] + "," + parte1[9] + "," + parte1[10] + ")");
                        fw.write(").\n");
                    }
                }

                for (int i = 1; i < linhas.size() - 1; i++) {
                    parte1 = linhas.get(i).split(";");
                    fw.write("localrecolha(" + parte1[0] + "," + parte1[1] + "," + parte1[2] + ",'" + parte1[3] + "'," + parte1[4] + ",
                    '" + parte1[5] + "','" + parte1[6] + "','" + parte1[7] + "'," + parte1[8] + "," + parte1[9] + "," + parte1[10] + ").\n");
                }
                parte2 = linhas.get(linhas.size() - 1).split(";");
                fw.write("localrecolha(" + parte2[0] + "," + parte2[1] + "," + parte2[2] + ",'" + parte2[3] + "'," + parte2[4] + ",
                '" + parte2[5] + "','" + parte2[6] + "','" + parte2[7] + "'," + parte2[8] + "," + parte2[9] + "," + parte2[10] + ").\n");
                fw.close();
            }
        } catch (IOException e) {
            System.out.println("Erro a ler/escrever no ficheiro");
        }
    }

}
```

Figura 1: Código do ***Main.java***

5 Algoritmos de Pesquisa

Antes de passar à resolução e explicação das *queries*, começarei por explicar os tipos de algoritmos de pesquisa que nos foram solicitados abordar, neste caso a **Pesquisa Informada** e **Pesquisa Não-Informada**. Cada um destes algoritmos possui vantagens e desvantagens. Quanto à pesquisa não-informada podemos destacar o algoritmo ***Depth First*** que se caracteriza por ter um tempo de procura maior para caminhos mais complexos, porém quando encontra uma solução, depressa apresenta alternativas. Para além do ***Depth First***, temos ainda o ***Breadth First***, contudo este algoritmo exige gastos de memória muito avultados e o seu tempo de pesquisa é superior ao algoritmo apresentado anteriormente, fazendo com que a sua implementação seja descartada.

Tendo em consideração que um grafo pode ser representado sob a forma de árvore, torna mais fácil a estimativa da complexidade e do tempo de pesquisa, como poderemos verificar nas tabelas abaixo. É de realçar que **b** pretende representar o fator de ramificação, **d** a profundidade da solução e **m** o tamanho da árvore, ou seja, a sua profundidade máxima.

Quanto aos algoritmos de pesquisa informada, estes fazem uso de heurísticas como forma de auxílio à pesquisa. A heurística selecionada para este problema será a distância euclidiana entre a origem e o destino, sendo este calculo feito através das latitudes e longitudes de cada um dos pontos de recolha. Para este tipo de algoritmos, foi apenas implementado o algoritmo **A***.

	Depth First	**Breadth First**
Início Procura	Origem	Origem
Complexidade	$O(b^m)$	$O(b^d)$
Ótima?	Não	Sim
Completa?	Não	Sim
Tempo pesquisa 1ª solução	$O(b^m)$	$O(b^d)$
Apresenta mais do que uma solução?	Sim	Não

Figura 2: Tabela comparativa da utilização de algoritmos de pesquisa não-informada

	A*
Conhecimento	Sim
Ótima?	Sim
Completa?	Sim
Eficiente?	Muito
Tempo pesquisa 1ª solução	Baixo

Figura 3: Tabela explicativa relativa aos algoritmos de pesquisa informada

6 Implementação da Solução

Este capítulo contém as explicações de todas as decisões tomadas para o desenvolvimento das cinco *queries* através da **pesquisa não-informada** e, em alguns casos, da **pesquisa informada**.

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

A ***primeira query*** pede que se defina um predicado que crie os circuitos entre dois pontos de recolha, uma origem e um destino, e neste caso foi utilizado o algoritmo ***depth first***.

```
geraCircuitos(Origem,Destino,Circuito):-
profundidade(Origem,Destino,[Origem],Circuito).

profundidade(Destino, Destino, H, C):- inverso(H,C).

profundidade(Origem, Destino, His, C):-
            adjacente(localrecolha(_,_,Origem,_,Rua1,_,_,_,_,_,_),localrecolha(_,_,Proximo,_,Rua2,_,_,_,_,_,_)),
            \+ member(Proximo, His),
            profundidade(Proximo,Destino,[Proximo|His],C).
```

Figura 4: Código do predicado ***geraCircuito***

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

Como a ***segunda query*** é um pouco dúbia para mim e não percebo bem o que estão a pedir, optei por definir a rota que passe apenas por locais com um determinado tipo de resíduo. Neste caso temos cinco tipos de resíduos, lixos, papel e cartão, embalagens, orgânico e vidro, e cada predicado encontra-se nas imagens abaixo pela ordem referida.

- **Lixos**

```
recolhaLixo(Origem,Destino,Circuito):-
            localrecolha(_,_,Origem,_,_,_,Residuo1,_,_,_,_),
            Residuo1 = 'Lixos',
            localrecolha(_,_,Destino,_,_,_,Residuo2,_,_,_,_),
            Residuo2 = 'Lixos',
            profundidade_lixos(Origem,Destino,[Origem],Circuito).

profundidade_lixos(Destino, Destino, H, C):- inverso(H,C).

profundidade_lixos(Origem, Destino, His, C):-
            adjacente(localrecolha(_,_,Origem,_,_,_,R1,_,_,_,_),localrecolha(_,_,Prox,_,_,_,R2,_,_,_,_)),
            R2 = 'Lixos',
            \+ member(Prox, His),
            profundidade_lixos(Prox,Destino,[Prox|His],C)
```

Figura 5: Código do predicado ***recolhaLixo***

- **Papel e Cartão**

```
recolhaPapelCartao(Origem,Destino,Circuito):-
            localrecolha(_,_,Origem,_,_,_,Residuo1,_,_,_,_),
            Residuo1 = 'Papel e Cartão',
            localrecolha(_,_,Destino,_,_,_,Residuo2,_,_,_,_),
            Residuo2 = 'Papel e Cartão',
            profundidade_papelcartao(Origem,Destino,[Origem],Circuito).

profundidade_papelcartao(Destino, Destino, H, C):- inverso(H,C).

profundidade_papelcartao(Origem, Destino, His, C):-
            adjacente(localrecolha(_,_,Origem,_,_,_,R1,_,_,_,_),localrecolha(_,_,Prox,_,_,_,R2,_,_,_,_)),
            R2 = 'Papel e Cartão',
            \+ member(Prox, His),
            profundidade_papelcartao(Prox,Destino,[Prox|His],C)
```

Figura 6: Código do predicado ***recolhaPapelCartao***

- **Embalagens**

```
recolhaEmbalagens(Origem,Destino,Circuito):-
            localrecolha(_,_,Origem,_,_,_,Residuo1,_,_,_,_),
            Residuo1 = 'Embalagens',
            localrecolha(_,_,Destino,_,_,_,Residuo2,_,_,_,_),
            Residuo2 = 'Embalagens',
            profundidade_embalagens(Origem,Destino,[Origem],Circuito).

profundidade_embalagens(Destino, Destino, H, C):- inverso(H,C).

profundidade_embalagens(Origem, Destino, His, C):-
            adjacente(localrecolha(_,_,Origem,_,_,_,R1,_,_,_,_),localrecolha(_,_,Prox,_,_,_,R2,_,_,_,_)),
            R2 = 'Embalagens',
            \+ member(Prox, His),
            profundidade_embalagens(Prox,Destino,[Prox|His],C)
```

Figura 7: Código do predicado ***recolhaEmbalagens***

- **Orgânico**

```
recolhaOrganico(Origem,Destino,Circuito):-
            localrecolha(_,_,Origem,_,_,_,Residuo1,_,_,_,_),
            Residuo1 = 'Organicos',
            localrecolha(_,_,Destino,_,_,_,Residuo2,_,_,_,_),
            Residuo2 = 'Organicos',
            profundidade_organico(Origem,Destino,[Origem],Circuito).

profundidade_organico(Destino, Destino, H, C):- inverso(H,C).

profundidade_organico(Origem, Destino, His, C):-
            adjacente(localrecolha(_,_,Origem,_,_,_,R1,_,_,_,_),localrecolha(_,_,Prox,_,_,_,R2,_,_,_,_)),
            R2 = 'Organicos',
            \+ member(Prox, His),
            profundidade_organico(Prox,Destino,[Prox|His],C)
```

Figura 8: Código do predicado ***recolhaOrganico***

- **Vidro**

```
recolhaVidro(Origem,Destino,Circuito):-
            localrecolha(_,_,Origem,_,_,_,Residuo1,_,_,_,_),
            Residuo1 = 'Vidro',
            localrecolha(_,_,Destino,_,_,_,Residuo2,_,_,_,_),
            Residuo2 = 'Vidro',
            profundidade_vidro(Origem,Destino,[Origem],Circuito).

profundidade_vidro(Destino, Destino, H, C):- inverso(H,C).

profundidade_vidro(Origem, Destino, His, C):-
            adjacente(localrecolha(_,_,Origem,_,_,_,R1,_,_,_,_),localrecolha(_,_,Prox,_,_,_,R2,_,_,_,_)),
            R2 = 'Vidro',
            \+ member(Prox, His),
            profundidade_vidro(Prox,Destino,[Prox|His],C)
```

Figura 9: Código do predicado ***recolhaVidro***

6.3 Query 3: Comparar circuitos de recolha tendo em conta os indicadores de produtividade

A ***terceira query*** pede que se compare os circuitos de recolha tendo em conta os indicadores de produtividade. Visto ter optado pela versão reduzida do trabalho, tomei como critérios de eficiência o número máximo de pontos de recolha num circuito. Assim, o predicado foi criado através de pesquisa informada, e o objetivo é retornar o circuito entre uma origem e um destino que passe por mais pontos de recolha possíveis.

```
maximoPontosRecolha(Origem, Destino,Circuito/Custo) :-
            assert(goal(Destino)),
            maximos_pontos([[Origem]/0], Destino, InvCircuito/Custo),
            retract(goal(Destino)),
            inverso(InvCircuito, Circuito).

maximos_pontos(Circuitos, Destino, Circuito) :-
            obtem_maximos_pontos(Ciruitos, Circuito),
            Circuito = [Nodo|_]/_,goal(Nodo).

maximos_pontos(Circuitos, Destino, SolucaoCircuito) :-
            obtem_maximos_pontos(Circuitos, MelhorCircuito),
            seleciona(MelhorCircuito, Circuitos, OutrosCircuitos),
            expande_informado_recolha(MelhorCircuito, Destino, ExpCircuitos),
            append(OutrosCircuitos, ExpCircuitos, NovoCircuitos),
            maximos_pontos(NovoCircuitos, Destino, SolucaoCircuito).

obtem_maximos_pontos([Circuito], Circuito) :- !.

obtem_maximos_pontos([Circuito1/Custo1,_/Custo2|Circuitos], MelhorCircuito) :-
            Custo1 >= Custo2, !,
            obtem_maximos_pontos([Circuito1/Custo1|Circuitos], MelhorCircuito).

obtem_maximos_pontos([_|Circuitos], MelhorCircuito) :-
            obtem_maximos_pontos(Circuitos, MelhorCircuito).

expande_informado_recolha(Circuito, Destino, ExpCircuitos) :-
            findall(NovoCircuito, proximaRecolha(Circuito, Destino, NovoCircuito), ExpCircuitos).

proximaRecolha([Nodo|Circuito]/Custo, Destino, [ProxNodo,Nodo|Circuito]/NovoCusto) :-
            adjacente(localrecolha(_,_,Nodo,_,_,_,_,_,_,_,_,_),localrecolha(_,_,ProxNodo,_,_,_,_,_,_,_,_,_)),
            localrecolha(_,_,Destino,_,_,_,_,_,_,_,_,_),
            \+ member(ProxNodo, Circuito),
            NovoCusto is Custo + 1.
```

Figura 10: Código do predicado ***maximoPontosRecolha***

6.4 Query 4: Escolher o circuito mais rápido (usando o critério da distância)

A ***quarta query*** tem um nível de complexidade superior às restantes, visto ter optado por adotar uma estratégia de pesquisa informada. A resolução deste predicado foi baseado nos algoritmos discutidos nas aulas como o algoritmos de **A*** e **pesquisa gulosa**, sendo que o algoritmo tem em consideração a distância a que o próximo nodo fica do destino. Assim, o objetivo é retornar o circuito entre uma origem e um destino que percorra menos distância. Para isso, foi necessário definir um predicado auxiliar que dado as latitudes e as longitudes dos nodos retorna a essa distância.

```
CircuitoMaisRapido(Origem, Destino,Circuito/Custo) :-
    assert(goal(Destino)),
    localrecolha(LatO,LonO,Origem,_,_,_,_,_,_,_),
    localrecolha(LatD,LonD,Destino,_,_,_,_,_,_,_),
    dist(LatO,LonO,LatD,LonD, Dist),
    mais_rapido([[Origem]/0/Dist], Destino, InvCircuito/Custo/_),
    retract(goal(Destino)),
    inverso(InvCircuito, Circuito).

mais_rapido(Circuitos, Destino, Circuito) :-
    obtem_mais_rapido(Circuitos, Circuito),
    Circuito = [Nodo|_]/_/_,goal(Nodo).

mais_rapido(Circuitos, Destino, SolucaoCircuito) :-
    obtem_mais_rapido(Circuitos, MelhorCircuito),
    seleciona(MelhorCircuito, Circuitos, OutrosCircuitos),
    expande_mais_rapido(MelhorCircuito, Destino, ExpCircuitos),
    append(OutrosCircuitos, ExpCircuitos, NovoCircuitos),
    mais_rapido(NovoCircuitos, Destino, SolucaoCircuito).

obtem_mais_rapido([Circuito], Circuito) :- !.

obtem_mais_rapido([Circuito1/Custo1/Dist1,_/Custo2/Dist2|Circuitos], MelhorCircuito) :-
    Dist1 =< Dist2, !,
    obtem_mais_rapido([Circuito1/Custo1/Dist1|Circuitos], MelhorCircuito).

obtem_mais_rapido([_|Circuitos], MelhorCircuito) :-
            obtem_mais_rapido(Circuitos, MelhorCircuito).

expande_mais_rapido(Circuito, Destino, ExpCircuitos) :-
            findall(NovoCircuito, proximoDist(Circuito, Destino, NovoCircuito), ExpCircuitos).

proximoDist([Nodo|Circuito]/Custo/_, Destino, [ProxNodo,Nodo|Circuito]/NovoCusto/Dist) :-
            adjacente(localrecolha(LatO,LonO,Nodo,_,_,_,_,_,_,_),localrecolha(Lat,Lon,ProxNodo,_,_,_,_,_,_,_)),
            localrecolha(LatF,LonF,Destino,_,_,_,_,_,_,_),
            \+ member(ProxNodo, Circuito),
            dist(LatO,LonO,Lat,Lon,D),
            NovoCusto is Custo + D,
            dist(Lat,Lon,LatF,LonF,Dist).

dist(LatO,LonO,LatD,LonD,Dist):-
            Sqrt is (LatD-LatO)^2 + (LonD-LonO)^2,
            Dist is sqrt(Sqrt).
```

Figura 11: Código do predicado ***circuitoMaisRapido***

6.5 Query 5: Escolher o circuito mais eficiente (recolha de resíduos 'Orgânicos')

A ***quinta query*** pedia que determina-se o circuito mais eficiente entre dois pontos de recolha usando um critério à minha escolha. Neste caso, optei por definir o circuito mais eficiente entre dois locais que apenas recolhe-se resíduos 'Orgânicos'. Sendo assim, o predicado vai retornar apenas um circuito que passe por locais de recolha apenas com resíduos orgânicos.

```
circuitoOrganico(Origem,Destino,Circuito):-
            localrecolha(_,_,Origem,_,_,_,Residuo1,_,_,_,_),
            Residuo1 = 'Organicos',
            localrecolha(_,_,Destino,_,_,_,Residuo2,_,_,_,_),
            Residuo2 = 'Organicos',
            profundidade_organico(Origem,Destino,[Origem],Circuito).

profundidade_organico(Destino, Destino, H, C):- inverso(H,C).

profundidade_organico(Origem, Destino, His, C):-
            adjacente(localrecolha(_,_,Origem,_,_,_,R1,_,_,_,_),localrecolha(_,_,Prox,_,_,_,R2,_,_,_,_)),
            R2 = 'Organicos',
            \+ member(Prox, His),
            profundidade_organico(Prox,Destino,[Prox|His],C).
```

Figura 12: Código do predicado ***circuitoOrganico***

7 Pesquisa Informada

Devido a problemas na definição dos arcos adjacentes, não foi possível comprovar a diferença do tempo de procura entre as *queries* 1 e 4, sendo que a primeira foi definida sob o algortimo *depth first* e a segunda sob o algoritmos de pesquisa informada.

De qualquer forma, é fácil perceber que o sempre que um predicado utiliza algoritmos de pesquisa informada, o tempo de procura de circuitos é significativamente menor relativamente à utilização de pesquisa não-informada.

8 Predicados auxiliares

Ao longo do desenvolvimento desta fase do projeto, foi necessária a criação de funções auxiliares de forma a simplificar a abordagem a tomar e assim obtermos mais facilmente os resultados que nos foram propostos.

```
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).
```

Figura 13: Código dos predicados auxiliares

9 Conclusão

Durante a elaboração desta trabalho prático individual, foram vários os obstáculos com que me deparei para a realização do mesmo.

Para começar, na definição dos arcos adjacentes, apenas consegui definir as adjacências para as ruas com o mesmo ID, em vez de para os ID's superiores e inferiores de uma localização. Dado esse pormenor, não foi possível testar corretamente a resposta às *queries* pedidas, nem comprovar com dados concretos as diferenças entre os algoritmos de pesquisa informada e não-informada.

De qualquer forma, tentei, sempre que possível, contornar as dificuldades, sempre explicando o pensamento e o resultado esperado em todas as situações bem como colocando os *prints* de todas as partes do código desenvolvido ao longo do trabalho prático.

Concluindo, o resultado deste projeto poderia ter sido melhor caso tivesse obtido sucesso na definição dos pontos de recolha, porém, devido a todo o esforço e determinação no desempenho e conclusão do projeto, penso que consegui ultrapassar esses obstáculos.

10 Referências

Para o desenvolvimento deste trabalho utilizei como referência os conhecimentos adquiridos durante as aulas teóricas e praticas da UC, bem como todos os materiais disponibilizados na *blackboard* pelos docentes como slides e apontamentos.