Universidade do Minho
Escola de Engenharia

Circuitos de Recolha de Resíduos Urbanos

Mestrado Integrado em Engenharia Informática

Sistemas de Representação de Conhecimento e Raciocínio

Autor:
A89618 - Henrique Gabriel dos Santos Neto

8 de junho de 2021

Resumo

Este trabalho tem como objetivo o desenvolvimento a resolução de um problema apresentado para avaliação da disciplina Sistemas de Representação de Conhecimento.

O trabalho consiste na modelação e desenvolvimento de uma aplicação capaz de gerir percursos de recolha de resíduos urbanos na freguesia de Misericórdia em Lisboa realizados por camiões do lixo. Adicionalmente, foi considerado que cada camião tem uma capacidade de máxima de 15 m^3.

Para tal, foram construídos algoritmos para pesquisas informadas e não informadas, a partir dos formatos estabelecidos de dados inerentes a *datasets* fornecidos.´

Estes *datasets* não permitem por eles próprios representar a informação a ser trabalhada sendo por este motivo necessário fazer um pré processamento a eles.

Índice

1 Formulação do problema **2**
1.1 Introdução . . . 2
1.2 Objetivo . . . 2
1.3 Preliminares . . . 3
1.3.1 Formato do *Dataset* . . . 3
1.3.2 Pré-Processamento do *Dataset* . . . 5
1.4 Problema de Pesquisa . . . 7
1.4.1 Estado Inicial e Final . . . 7
1.4.2 Operadores e Testes Objetivos . . . 7
1.4.3 Custos e parâmetros de eficiência . . . 8
1.5 Estrutura do Sistema . . . 8

2 Estratégias de procura **9**
2.1 Procura não informada . . . 9
2.1.1 Recolha de Resíduos . . . 9
2.1.2 Seleção de Pontos de Recolha . . . 10
2.1.3 Exclusão de Pontos de Recolha . . . 10
2.1.4 Detalhes sobre Branch And Bound . . . 11
2.2 Procura informada . . . 12

3 Resultados **13**

4 Comentários Finais e Conclusão **14**

A Pré-Processador **15**
A.1 analisador_lexico.py . . . 15
A.2 grafo.py . . . 15
A.3 preprocessador.py . . . 17

B Sistema em Prolog **21**
B.1 baseDeConhecimento.pl . . . 21
B.2 utilidades.pl . . . 22
B.3 main.pl . . . 24

Formulação do problema

1.1 Introdução

Este trabalho teve como objetivo a representação de circuitos de recolha de resíduos urbanos para o distrito de Lisboa e consequentemente a aplicação de algoritmos para o seu gestão recorrendo ao sistema de inferência do *prolog*.

Desta forma, o sistema de recolha será constituído por vários pontos de recolha e os respetivos caminhos que entre estes. Para o reconhecimento do sistema são usados *datasets* que especificam alguns paramentos deste mapa e que após um pré-processamento permitem resolver o problema imposto.

Para o calculo dos percursos foram adaptados vários algoritmos de pesquisa informada e não informada, que permite avaliar assim quais o circuitos desejados conforme as vantagens e utilidades de cada um respeitando támbém o limite máximo que cada camião ($15m^3$).

1.2 Objetivo

Como referido anteriormente, objetivo de projeto é o desenvolvimento um sistema que permita importar os dados relativos aos diferentes circuitos e que permita obter recomendações sobre estes. Um circuito é constituído por vários percursos que estão divididos nas seguintes considerações elementares:

- Inicial – Percurso entre a garagem e o primeiro ponto de recolha;
- Ponto de Recolha - Remoção de resíduos num local de paragem;
- Entre Pontos – Percurso entre dois pontos de Recolha;
- Transporte – Percurso entre o último ponto de Recolha e o local de deposição de resíduos;
- Final – Percurso entre o local de deposição e a garagem.

1.3 Preliminares

Para se implementar os algoritmos de procura, foi necessário processar o *dataset* fornecido para desta forma se poder trabalhar com a informação.

1.3.1 Formato do *Dataset*

O *dataset* fornecido provêem dos "Circuitos de Recolha de Resíduos Urbanos"[1] disponibilizados pela Câmara de Lisboa. Desta forma, a informação fornecida pelo instituto é fornecida como uma tabela com vários atributos registam informação sobre um especifico conjunto de contentores. Neste trabalho, foram utilizados oito atributos sendos estes:

Latitude/Longitude corresponde ás coordenadas globais do ponto de recolha, a partir deles são calculas as distâncias entre os vários nodos existentes.

OBJECTID identifica o objeto a que correspondente a informação. Permite certificar que não existem repetições no *dataset*.

PONTO_RECOLHA_FREGUESIA corresponde á freguesia onde se encontra o ponto de recolha referido.

PONTO_RECOLHA_LOCAL corresponde a uma série de informação variável sobre o ponto de recolha em questão. Desta forma, este atributo é por si constituído por um número correspondente ao identificação do ponto de recolha. De seguida é apresentada o nome da rua que por sua vez pode ter um parâmetro de informação extra entre parêntesis (como por exemplo a rua "R Cintura (Santos)"). Seguidamente o formato varia caso o ponto de recolha retratar a rua na sua totalidade ou apenas de um troço da mesma. Caso se trate da rua na sua totalidade, poderá simplesmente ser apresentado o número das casas correspondestes. Por outro lado, caso se trate de um segmento de uma rua, é apresentado entre parêntesis o lado em que se encontra os contentores (Par, Ímpar ou Ambos), seguido do número dos edifícios e as ruas que intervêm nos cruzamentos delimitantes.

[1] http://dados.cm-lisboa.pt/no/dataset/circuitos-de-recolha-de-residuos-urbanos

Exemplificando, para o valor "15805 : *R do Alecrim* (*Par* (→)(26 → 30) : *R Ferragial* – *R Ataíde*)", a sua respetiva entrada pertenceria ao ponto de recolha 15805 presente na *Rua do Alecrim*, no troço entre a *Rua Ferragial* e a *Rua Ataíde*. Também indica que se tratam dos prédios 26 a 30 e que os contentores encontram-se no lado par da estrada. No mapa o troço referido corresponde a:

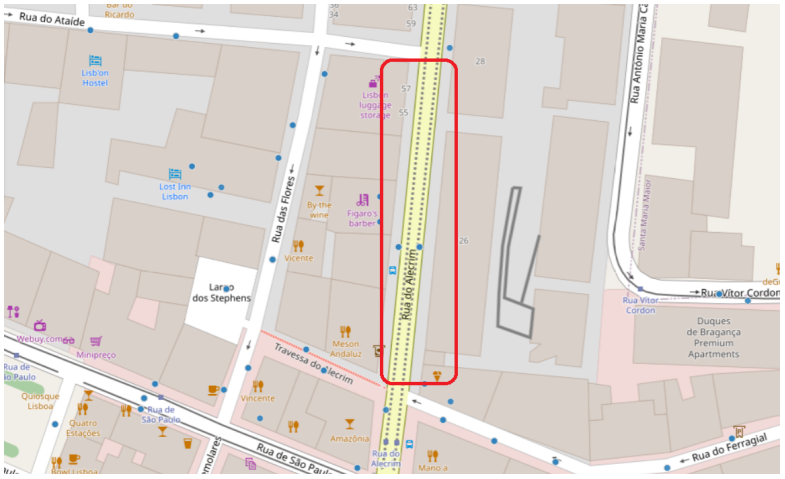

Figura 1.1: Exemplo de um ponto de recolha numa rua segmentada

CONTENTOR_RESÍDUO corresponde ao tipo de resíduos que os contentores se destinam. No *dataset* em estudo, os tipos podem ser Papel e Cartão, Organicos, Vidro, Embalagens e Lixos.

CONTENTOR_TIPO indica o tipo de contentor a que se refere o objeto.

CONTENTOR_CAPACIDADE indica a capacidade do tipo de contentor referido.

CONTENTOR_QT indica o número de contentores especificados pelo objeto.

CONTENTOR_TOTAL_LITROS indica a capacidade total em litros (dcm^3) especificado pelo objeto.

1.3.2 Pré-Processamento do *Dataset*

Como referido anteriormente, o *dataset* embora possui muita informação vital, não é útil no formato em que está , e desta forma foi construido um pré-processador de dados usando a linguagem *Python*. Neste pré-processador, foi desenvolvido um analisador léxico e gramatical, a partir da biblioteca *ply* para reconhecer as entradas do *dataset* no formato *csv* (sendo o separador ';').

Problemas do *Dataset*

O pré-processador referido foi desenhado para resolver os seguintes problemas:

- A informação contida na coluna *PONTO_RECOLHA_LOCAL* encontra-se extremamente multiplexada, o que torna impossível o seu uso direto. Desta forma, foi necessário estabelecer diversas produções gramaticais de alta complexidade, para que desta forma seja possível interpretar a informação contida neste parâmetro.
- Existem vários objetos referentes ao mesmo ponto de recolha, visto que cada entrada na tabela só pode especificar um único tipo de resíduo, ou contentor.

  Desta forma, as entradas respetivas ao mesmo ponto de recolha foram reduzidas em uma só, passando cada uma a ter uma lista de pares que relacionam o tipo de lixo á sua capacidade máxima no respetico local.
- O *dataset* em si não define nenhuma aresta concreta entre os nodos. Este foi o maior problema a resolver, de maneira mais clara, a lista de entradas não possui nenhuma informação direta sobre ligações, a única informação presente sobre este assunto encontra-se nas entradas correspondentes a segmentos de uma dada rua, visto que estes são definidos pelas ruas que os delimitam. Para além disto, é impossível confirmar qualquer aresta entre os respetivos nodos.

  Desta forma, foi assumido um conjunto de regras para se poder formar um grafo a partir da informação dos nodos. Primeiro, cada segmento de uma rua estaria por si só ligado por um arco a todos os outros segmentos da mesma rua. De seguida, cada segmento estaria adjacente ás ruas que o limitavam, caso estas estivem registadas em algum ponto de recolha. Por fim, cada rua não segmentada teria uma conexão com os nodos das entradas adjacentes. Desta forma foi assegurado que o grafo era convexo, ou extremamente próximo de a um.
- Existem vários problemas associados aos *encondings* diferentes entre os vários

ficheiros geridos, o que provocava caracteres inválidos e frases corrompidas. Estes problemas foram resolvidos de forma trivial, a partir de ferramentas de conversão inerentes ao *Python* ou por substituição direta dos caracteres.

Estrutura do Pré-Processador

A aplicação de pré-processamento é constituida por três ficheiros (presentes no apêndice a este trabalho) que fazem recurso às bibliotecas *ply*, *math* e *re* para processar a informação.

O primeiro ficheiro designa-se por *grafo.py* e contém duas classes (*pontoRecolha* e *aresta*) responsáveis por facilitar a gestão da informação, permitindo assim alguma simplicidade nas operações de calculo de distancias e de redução de nodos.

O segundo ficheiro designa-se por *analisador_lexico.py* e contem as simples declarações da estrutura léxica a ser lida, como por exemplo, as sequências de caracteres que definem nomes de ruas, números reais, números das portas, etc.

Por fim , o terceiro ficheiro (*preprocessador.py*) contém a gramática que interpreta o *dataset*, juntamente com o conjunto de funções que permitem calcular as arestas a partir das três regras referidas anteriormente. Adicionalmente, também é responsável por escrever os repetidos ficheiros processados, que possuem os nomes *nodos.csv* e *arestas.csv*.

Resultado e Problemas resultades

Para o *dataset* fornecido o resultado do pré-processamento foi um grafo com 121 nodos e 338 arestas. Este resultado foi considerado ótimo, até que durante as procuras realizadas na fase seguinte, foi verificado que o *Prolog* não tem capacidade para gerir um grafo destas dimensões de maneira eficiente e viável. Para resolver este problema, foi implementada o modo de simplificação no pré-processador, que consiste em fazer uma redução adicional aos pontos de recolha pertencentes á mesma rua. Desta forma, cada rua teria no máximo um único ponto de recolha, cujas coordenadas corresponderiam ao ponto médio entre as coordenados dos vários pontos de recolha envolvidos. Este modo de simplificação permitiu reduzir bastante o grafo, passando este a ter 60 nodos e 114 arestas. Para efetuar o pré-processamento neste modo, é necessário apresentar ao programa a palavra '*simplicado*' , ou a flag '$-s$' na chamada para execução.

1.4 Problema de Pesquisa

Os problemas em consideração são determinísticos visto que o resultado será consistente para qualquer estado em consideração. Por outro lado, os circuitos podem ter várias relativamente aos pontos de inicio e de fim, sendo que cada camião por si pode provir de diferentes garagens e prover a diferentes pontos de depósito.

Desta forma, é possível concluir que estamos perante problemas de múltiplos estados, em que para para cada algoritmo de procura é necessário indicar quais os estados iniciais (e possivelmente finais) de cada procura, juntamente com o estado atual do sistema.

1.4.1 Estado Inicial e Final

Dependo do modo em que o circuito for calculado, o estado do sistema pode envolver apenas o estado atual, como também pode evolver a quantidade de resíduos presentes no camião e a distancia percorrida. No tabalho desenvolvido, caso o estado fosse representado por um tuplo, teria a forma:

$$(Nodo\ Atual,\ Quantidade\ Carregada)$$

O estado inicial e final do circuito total são os mesmos e possuem o camião vazio (quantidade nula) sendo o nodo atual a garagem porém em cada percurso constituinte, os estados iniciais e finais são distintos. As variações ocorrem da seguinte forma:

Percurso	Estado Inicial	Estado Final
Inicial	(Garagem, 0)	(Ponto Inicial, 0)
Entre Nodos	(Ponto Inicial, 0)	(Ponto Final, Quantidade Final)
Transporte	(Ponto Final, _)	(Deposito, 0)
Final	(Deposito, 0)	(Garagem, 0)

Tabela 1.1: Variação dos estados de cada percurso

1.4.2 Operadores e Testes Objetivos

Nas procuras também são tratados por outros dados alheios ao estado que permitem com este permitem fazer vários testes e garantir a validade do sistema. Estes dados são os seguintes.

Capacidade Máxima do camião Indica a capacidade do camião, sendo por esta forma um volume. Neste trabalho foi considerado que cada camião carrega no máximo $15m^3$ (15000 litros) de resíduo. Desta forma, sempre que existe uma recolha, o volume resultante no camião é comparado com este valor de maneira a que nunca seja superior.

Tipo de Resíduo Indica o tipo de resido que o camião pode estar especificado para, que garante que o camião não recolhe resíduos indesejados.

1.4.3 Custos e parâmetros de eficiência

Os custos associados a cada percurso estão diretamente relacionados com a distancia percorrida, sendo por isso este o fator principal contabilizado. Contudo, um aumento desta distancia pode ser justificado, se se tiver em conta a quantidade de resíduos recolhidos. Desta forma, foram estabelecidos dois parâmetros de eficiência para estudo.

Distancia percorrida - Este parâmetro baseia-se unicamente na distancia percorrida, sendo que desta forma quando menor a distancia maior a eficiência do circuito.

Distancia percorrida por Litro de Resíduo - Para além da distancia, este fator de eficiência varia de maneira inversa conforme o volume de resíduos recolhidos. Desta forma, quanto menor a distancia e quanto maior o quantidade de resíduos recolhidos, maior será a eficiência.

1.5 Estrutura do Sistema

O sistema encontra-se implementado em 3 ficheiros:

baseDeConhecimento.pl Implementa os predicados necessários para ler o *dataset* processado e popular a base de conhecimento.

utilidades.pl Implementa os predicados utilitários ao sistema como por exemplo as estimativas usadas na procura informada.

main.pl Implementa todos os predicados de procura fazendo recurso aos outros ficheiros.

Estratégias de procura

Foram estabelecidas 4 estratégias de procura não informada, e 2 de procura informada.

2.1 Procura não informada

Cada método de procura não informada estabelece três predicados para começar as procuras, cada um sendo uma expansão ao anterior. Um deles possui o prefixo "caminho"que representa a procura da maneira mais simplificada. Os outros dois, contem o prefixo "recolha", que implementação a procura tendo em conta a capacidade máxima do camião e o respetivo Tipo de resíduo em recolha se este for especificado.

Como referido anteriormente, neste trabalho foram implementados quatro tipos de procuras, sendo estas:

- Profundidade (DFS - Depth-First Search)
- Busca Iterativa Limitada em Profundidade
- Largura (BFS - Breadth-First Search)
- Ramificar e limitar (BNB - Branch And Bound Search)

Embora, as formas com que as decisões são diferentes em cada um dos algoritmos, cada um segue um padrão semelhante no processo de procura. Este padrão será descrito nas secções seguintes.

2.1.1 Recolha de Resíduos

Caso a procura corresponda ao percurso entre nós, ou seja, caso se trate de um predicado de "recolha"que tenha em consideração a capacidade do camião, são primeiramente recolhidos os resíduos do local através do predicado *encher*/4 (ou *encher*/5, caso seja especificado o tipo de resíduo). Este predicado acumula o valor do volume presente, até ao limite máximo especificado. De seguida é feito um teste para verificar se o camião está cheio. Se estiver a procura no ramo atual é terminada, senão os

algoritmos seguem para a estapa seguinte. É de notar, que no caso das recolhas *DFS* e Limitada, os próximos passos são realizados em outros predicados (que possuem o prefixo "continuar") para possibilitar a dedução de destinos possiveis, caso o utilizador pretenda.

```
(...):-
    nodo(Atual,_,_,_,Lixo),
    encher(Tipo, Lixo, QuantAtual, Max, NovaQuantidade),
    NovaQuantidade<Max,
    (...)

(...):-
    nodo(Atual,_,_,_,Lixo),
    encher(Tipo, Lixo, QuantAtual, Max, NovaQuantidade),
    NovaQuantidade=:=Max,
    (...)
```

Listing 2.1: Exemplo de recolha de um residuo especifico

2.1.2 Seleção de Pontos de Recolha

Este passo consiste na seleção do(s) próximo(s) ponto(s) de recolha a ser(em) processados. No caso das procuras em profundidade este consiste em selecionar um dos nodos adjacentes ao atual que ainda não foi visitado enquanto que nas outras procuras são selecionados todos os nodos adjacentes que ainda não foram visitados, sucessivamente são adicionados à fila da procura. No caso da procura *BFS*, os nodos são escritos pela ordem em que forem calculados enquanto que na procura *BNB*, os nodos são inseridos por ordem crescente da distancia percorrida, o que permite um convergência mais rápida para a solução óptima.

2.1.3 Exclusão de Pontos de Recolha

Esta fase pode estar unificada depois da selecção, como também pode estar distribuía ao longo do algoritmo. Adicionalmente é a fase mais única a cada processo de procura, sendo assim uma das etapas mais complexas do algoritmo.

Exclusão na Procura em Profundidade Neste caso, a exclusão ocorre depois da seleção do nó e desta forma, este é apenas escolhido se ainda não foi visitado.

```
(...)
adjacencia(Act,X,_),
\+ member(X,LA),
(...)
```

Exclusão na Procura em Profundidade Limitada Nesta procura, para além da exclusão feita na procura em profundidade normal, é feita uma exclusão adicional antes da seleção que verifica se a profundidade da árvore não excedeu o limite máximo especificado.

```
(...)
Profundidade < ProfundidadeMax,
(...)
adjacencia(Act,X,_),
\+ member(X,LA),
NP is Profundidade +1,
(...)
```

Exclusão na Procura em Largura e Branch And Bound Neste caso, a exclusão ocorre durante da seleção do nó sendo que cada um só é escolhido para processamento se ainda não foi visitado.

```
(...)
findall((DistX,NovaQuantidade,[X|LA]),
(
    Destino\==Atual,
    arco(Atual,X,DistanciaX),
    \+ member(X,LA),
    DistX is DistanciaX + Da),
Novos),
(...)
```

2.1.4 Detalhes sobre Branch And Bound

Este algoritmo pode ser visto como uma expansão ao algoritmo de Procura em Largura, e desta forma associado a cada caminho existe adicionalmente o seu custo e quantidade de resíduos recolhida, para que desta forma seja possível ordenar corretamente os pontos para a seu processamento.

2.2 Procura informada

Os algoritmos de pesquisa informada fazem recurso a heurísticas para a auxiliar a pesquisa e decisão de qual o ponto de recolha a escolher e seguir. Neste trabalho foram exploradas duas heurísticas, a da distancia entre cada ponto de recolha, que consiste na distancia euclidiana entre as coordenadas dos pontos e a da distancia por quantidade, que se baseia na distancia percorrida para cada litro de resíduo que será recolhido.

Desta forma, para cada algoritmo existem dois predicados correspondentes ao uso das diferentes heurísticas em estudo. Foram adaptados os dois algoritmos de pesquisa *A* (A estrela / Melhor Primeiro)* e o algoritmo da *Gulosa (Greedy)* estudados nas aulas.

Resultados

Para os obter os resultados foram usados *findall*, os circuitos foram gerados por partes para dimensionar a carga total num intervalo de tempo, embora existem alguns predicados que permitem calcular o circuito todo de uma vez. Nas procuras não informadas, foi simplesmente calculado ate que a solução converja.

Os resultados obtidos foram realizados de forma a que a garagem e o depósito estivessem o máximo afastadas uma da outra de maneira a exigir um grande esforço do sistema.

Estratégia	Tempo (Segundos)	Complexidade	Distancia (euc.) / Quantidade (l)	Encontrou a melhor solução?
DFS	42	$O(V^2)$	0.0466/15000	Sim
Limitada	38	$O(V^2)$	0.0466/15000	Sim
BFS	41	$O(V^2)$	0.0466/15000	Sim
BNB	32	$O(V^2)$	0.0466/15000	Sim
A*	< 1	$O(V^S)$	0.0466/15000	Sim
Gulosa	< 1	$O(V^S)$	0.0466/15000	Sim

Tendo em conta que $15m^3$ é um volume muito pequeno tendo em conta a capacidade média dos caixotes do lixo, os camiões ficavam cheios muito rapidamente o que fazia os percursos entre nodos curto. Isto a meu ver, pode ser a razões pelas quais todas as procuras conseguiram convergir na solução ótima eventualmente.

Os algoritmos que usavam mais memória são o em largura e o gulosa. Aliás, foi necessário aumentar o tamanho máximo da stack do *Prolog*, pois em percursos grandes, o tamanho desta poderia chegar aos 5 *Gigabytes*.

Comentários Finais e Conclusão

Em suma, todas as funcionalidades foram implementadas com sucesso, o que permitiu a exploração dos problemas de forma imparcial e consequentemente permitiu obter várias soluções otimas conforme o contexto.

A maior dificuldade com que fui deparado foi na adaptação dos algoritmos de pesquisa, que por si só também foi agravada pelos problemas de memória e eficiência que surgiam no sistema de inferência, dado às dimensões do grafo em estudo.

No caso de haver futuro desenvolvimento, gostaria de explorar mais processos de procura, e se possível, melhor as funcionalidades atuais e possivelmente implementar uma interface gráfica para as mesmas.

Pré-Processador

A.1 analisador_lexico.py

```
import ply.lex as lex
import re

literals = [';', ':', '(', ')', '-', ',', '/']
tokens = ['FNUM', 'HousesNDirections', 'NUM', 'STRING']

def t_FNUM(t):
    r'[+\-]?\d+[,.]\d+'
    t.value = re.sub(r',', '.', t.value)
    return t

t_HousesNDirections = r'(\(\d*->\d*\))+'
t_NUM = r'[+\-]?\d+'
t_ignore = ' \n\t'
t_STRING = r'[^\W\d](((\w-\w|\w)|[. ])*\w)?'

def t_error(t):
    print("Syntax Error: ", t)
    t.lexer.skip(1)

lexer = lex.lex(reflags=re.UNICODE)
```

A.2 grafo.py

```
import math

def strSplitBy(dict, regex):
    res = ""
    for i in dict:
        res = res + str(i) + "=" + str(dict[i]) + regex
    return res[:-1]

class pontoRecolha:
    collapseCount = 1
    id_ = latitude = longitude = rua = residuos = None

    def __init__(self, id_, latitude, longitude, rua, capacidade, residuo):
        self.id_ = id_
        self.latitude = latitude
        self.longitude = longitude
        self.rua = rua
```

```
        self.residuos = {residuo: capacidade}

    def append(self, capacidade, tipoResiduo):
        if tipoResiduo in self.residuos:
            self.residuos[tipoResiduo] += capacidade
        else:
            self.residuos[tipoResiduo] = capacidade

    def distanciaRealKm(self, outro):
        lat1 = math.radians(self.latitude)
        lon1 = math.radians(self.longitude)
        lat2 = math.radians(outro.latitude)
        lon2 = math.radians(outro.longitude)
        dlon = lon2 - lon1
        dlat = lat2 - lat1
        a = math.sin(dlat / 2) ** 2 + math.cos(lat1) * math.cos(lat2) * math.sin(dlon
            / 2) ** 2
        c = 2 * math.atan2(math.sqrt(a), math.sqrt(1 - a))
        return 6373.0 * c  # Raio da terra * c

    def distanciaEucladiana(self, outro):
        return math.sqrt(pow(self.latitude - outro.latitude, 2) + pow(self.longitude -
             outro.longitude, 2))

    def mergeWith(self, outro):
        lat = self.latitude * self.collapseCount
        long = self.longitude * self.collapseCount
        self.collapseCount += 1
        self.longitude = (lat + outro.latitude) / self.collapseCount
        self.longitude = (long + outro.longitude) / self.collapseCount
        self.appendResiduos(self.residuos)

    def __str__(self) -> str:
        return "PontoRecolha\n\tid:" + str(self.id_) + "\t rua:'" + str(self.rua) + "
            '\n\tlat/long:\t" + str(
            self.latitude) + "\t" + str(self.longitude) + "\n\tresiduos:\t" + str(self
                .residuos)

    def csvline(self):
        return str(self.id_) + ";" + str(self.rua) + ";" + str(self.latitude) + ";" +
            str(
            self.longitude) + ";" + strSplitBy(self.residuos, '|') + "\n"

    def appendResiduos(self, residuos):
        if residuos:
            for nome in residuos:
                capacidade = residuos[nome]
                self.append(capacidade, nome)

class aresta:
    inicio = fim = custo = None

    def __init__(self, inicio, fim, custo):
        self.inicio = inicio
```

```
        self.fim = fim
        self.custo = custo

    def __str__(self):
        return "I:" + str(self.inicio) + " D: " + str(self.fim) + " Custo: " + str(
            self.custo)

    def csvline(self):
        return str(self.inicio) + ";" + str(self.fim) + ";" + str(self.custo) + "\n"
```

A.3 preprocessador.py

```
import ply.yacc as yacc
from analisador_lexico import tokens
import grafo as graph
import sys

nodos = {}
nomesParaIds = {}
cruzamentos = {}
parsedRelations = []
listaArestas = []

# Gramatica:
def p_LINHA(p):
    """LINHA : OBJETO
             | LEGENDA"""
    p[0] = p[1]

def p_Legenda(p):
    """LEGENDA : LEGENDA ';'
               | LEGENDA STRING"""
    p[0] = p[1] + p[2]

def p_Legenda_empty(p):
    """LEGENDA : """
    p[0] = ''

def p_OBJETO(p):
    """OBJETO : FNUM ';' FNUM ';' NUM ';' STRING ';' PLocal ';' STRING ';' STRING ';'
        NUM ';' NUM ';' NUM """
    handleObjeto(float(p[1]), float(p[3]), p[9], p[11], int(p[19]))

def p_PLocal_segmentado(p):
    """PLocal : NUM ':' STRING Segmento"""
    extra, x = p[4]
    if extra:
        p[3] = p[3] + extra
    p[0] = (int(p[1]), p[3], x)

def p_Segmento(p):
```

```
    """Segmento :  '(' STRING HousesNDirections ':' Rua '-' Rua ')' """
    p[0] = (None, (p[5], p[7]))

def p_Segmento_Extra(p):
    """Segmento :  '(' STRING FimInfoRua '(' STRING HousesNDirections ':' Rua '-' Rua
        ')' """
    p[0] = (p[1] + p[2] + p[3], (p[8], p[10]))

def p_Segmento_Extra_fir(p):
    """Segmento :  '(' STRING FimInfoRua  """
    p[0] = (p[1] + p[2] + p[3], None)

def p_Segmento_empty(p):
    """Segmento : """
    p[0] = (None, None)

def p_Rua_empty(p):
    """Rua : STRING"""
    p[0] = p[1]

def p_Rua(p):
    """Rua : STRING '(' STRING FimInfoRua"""
    p[0] = p[1] + p[2] + p[3] + p[4]

def p_FimInfoRua_Nome(p):
    """FimInfoRua : ')' """
    p[0] = p[1]

def p_FimInfoRua_NomeENum(p):
    """FimInfoRua :  ',' NUM ')'"""
    p[0] = p[1] + p[2] + p[3]

def p_PLocal_virgulaString(p):
    """PLocal : NUM ':' STRING ',' STRING FatoresInuteis"""
    p[0] = (int(p[1]), p[3] + p[4] + p[5], None)

def p_PLocal_virgulaFatores(p):
    """PLocal : NUM ':' STRING ',' FatoresInuteis """
    p[0] = (int(p[1]), p[3], None)

def p_FatoresInuteis(p):
    """FatoresInuteis : FatoresInuteis Fator
                      | """

def p_Factor(p):
    """Fator : '-'
             | NUM
             | '/'
             | '('
             | ')'
             | STRING
             | ','
     """
    return p[1]
```

```
n = 1  # Line number

def p_error(p):
    print('Syntax error! Linha ', n, ' --> ', p)

parser = yacc.yacc()

# Fim da Gramática
# Processamento de Arestas :
def mkaresta(id1, id2):
    return graph.aresta(id1, id2, nodos[id1].distanciaEucladiana(nodos[id2]))

simplificar = False

def handleObjeto(latitude, longitude, PontoLocal, tipoResiduo, capacidadeTotal):
    id_, rua, segmento = PontoLocal
    if id_ in nodos:
        nodos[id_].append(capacidadeTotal, tipoResiduo)
    else:
        NovoObjeto = graph.pontoRecolha(id_, latitude, longitude, rua, capacidadeTotal
            , tipoResiduo)

        foiInserido = False
        if rua in nomesParaIds:
            if simplificar:
                id_ = nomesParaIds[rua][0]
                nodos[id_].mergeWith(NovoObjeto)
            else:
                nodos[id_] = NovoObjeto
                foiInserido = True
                for i in nomesParaIds[rua]:
                    listaArestas.append(mkaresta(i, id_))
                addIfAbstent(nomesParaIds[rua], id_)
        else:
            nomesParaIds[rua] = [id_]
            nodos[id_] = NovoObjeto
            foiInserido = True

        temVizinhos = False
        if segmento:
            x, y = segmento
            for count in range(2):
                if x is not None:
                    temVizinhos = True
                    if x not in cruzamentos:
                        cruzamentos[x] = [id_]
                    elif id_ not in cruzamentos[x]:
                        cruzamentos[x].append(id_)
                x = y
        if foiInserido:
            parsedRelations.append((id_, temVizinhos))

def addIfAbstent(lista, elemento):
```

```
    if lista is None:
        return [elemento]
    if elemento not in lista:
        lista.append(elemento)
    return lista

# MAIN

if 'simplicado' in sys.argv or '-s' in sys.argv:
    simplificar = True

# Ler e fazer Parse

with open('dataset.csv', encoding='UTF8') as file:
    linha = file.readline()
    print("Line ignored: ", linha)
    while linha:
        n += 1
        linha = file.readline()
        parser.parse(linha)

# Processar arestas de cruzamentos

for cruz in cruzamentos:
    if cruz in nomesParaIds:
        idx = nomesParaIds[cruz][0]
        for obj in cruzamentos[cruz]:
            if obj != idx:
                listaArestas.append(mkaresta(idx, obj))

# Processar arestas de nodos sem vizinhos diretos

count = len(parsedRelations)
curr = 0
old, conected = parsedRelations[count - 1]

while curr < count:
    obj, conected = parsedRelations[curr]
    if not conected:
        if obj is not old:
            listaArestas.append(mkaresta(obj, old))
        next, b = parsedRelations[(curr + 1) % count]
        if b and obj is not next:
            ares = mkaresta(obj, next)
    old = obj
    curr += 1

with open("nodos.csv", "w", encoding="UTF8") as file:
    for node in nodos:
        file.write(nodos[node].csvline())

with open("arestas.csv", "w", encoding="UTF8") as file:
    for ars in listaArestas:
        file.write(ars.csvline())
```

Sistema em Prolog

B.1 baseDeConhecimento.pl

```
use_module(library(csv)).

:- dynamic carregarPredefinicoes/0.
:- dynamic nodo/5.
:- dynamic arco/3.

carregarPredefinicoes():-
    carregaNodos("Parser/nodos.csv"),
    carregaArcos("Parser/arestas.csv").

carregaArcos(Nome_CSV):-
    csv_read_file(Nome_CSV, Data, [functor(arco), separator(0';),arity(3),match_arity(
        false)]),
    assertListaArcos(Data).

assertListaArcos([arco(Ponta1,Ponta2,Distancia)|T]):-
    not(existe(Ponta1,Ponta2)),
    assert(arco(Ponta1,Ponta2,Distancia)),
    assertListaArcos(T).

assertListaArcos([_|T]):- assertListaArcos(T).

assertListaArcos([]).

carregaNodos(Nome_CSV):-
    csv_read_file(Nome_CSV, Data, [functor(prenodo), separator(0';),arity(5),
        match_arity(false)]),
    assertListaNodos(Data).

assertListaNodos([prenodo(Id,Nome,Latitude,Longitude,Residuos)|T]):-
    split_string(Residuos, "|", '', ListaResiduos),
    paraPares(ListaResiduos,Res),
    assert(nodo(Id,Nome,Latitude,Longitude,Res)),
    assertListaNodos(T).


assertListaNodos([_|T]):-assertListaNodos(T).

assertListaNodos([]).

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

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

```
        obtem_melhor_g([Caminho1/Custo1/Est1|Caminhos], MelhorCaminho).

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

paraPares([String|T],[S|R]):-
    split_string(String, "=", '', [A,B|_]),
    atom_number(B, X),
    S = (A,X),
    paraPares(T,R).

paraPares([],[]).

existe(X,Y):-
    arco(X,Y,_).

existe(X,Y):-
    arco(Y,X,_).
```

B.2 utilidades.pl

```
% ------ Utilitades ------------------------------------------------------
:- dynamic nodo/5.
:- dynamic arco/3.

adjacencia(X,Y,Dist) :-
    arco(X,Y,Dist).

adjacencia(X,Y,Dist) :-
    arco(Y,X,Dist).

% baseado na distancia percorrida
estimativa(Nodo1,Nodo2,Estimativa):-
    nodo(Nodo1,_,X1,Y1,_),
    nodo(Nodo2,_,X2,Y2,_),
    Estimativa is sqrt((X1-X2)^2+(Y1-Y2)^2).

% baseado na distancia percorrida por volume de resíduos recolhidos
estimativaDistPorResid(Nodo1,Nodo2,Estimativa):-
    nodo(Nodo1,_,X1,Y1,_),
    nodo(Nodo2,_,X2,Y2,Residuos),
    sumResiduos(Residuos,Res),
    Estimativa is sqrt((X1-X2)^2+(Y1-Y2)^2)/Res.

sumResiduos([],0.0).

sumResiduos([(_,X)|T],Resultado):-
    sumResiduos(T,PreRes),
    Resultado is X + PreRes.
```

```
getResiduo(Tipo,[(Tipo,Res)|_],Res).

getResiduo(Tipo,[_|T],Res):- getResiduo(Tipo,T,Res).

encher([(_,Res)|T],Quant,Max,Fill):-
    Curr is Quant+Res,
    Max >= Curr,
    encher(T,Curr,Max,Fill).

encher([(_,Res)|_],Quant,Max,Max):-
    Curr is Quant+Res,
    Max < Curr.

encher([],Quant,_,Quant).

encher(Tipo,[(Tipo,Res)|T],Quant,Max,Fill):-
    Curr is Quant+Res,
    Max >= Curr,
    encher(T,Curr,Max,Fill).

encher(Tipo,[(Tipo,Res)|_],Quant,Max,Max):-
    Curr is Quant+Res,
    Max < Curr.

encher(Tipo,[_|T],Quant,Max,Fill):-
    encher(Tipo,T,Quant,Max,Fill).

encher(_,[],Quant,_,Quant).

calculaDistancia([P1, P2|T],Distancia):-
    adjacencia(P1, P2, N),
    calculaDistancia([P2|T], DistanciaAcumulada),
    Distancia is N + DistanciaAcumulada.

calculaDistancia([],0).

calculaQuantidadeTotal([Id|T],Quantidade):-
    nodo(Id, _, _,_,Lista),
    sumResiduos(Lista,Quant),
    calculaQuantidadeTotal(T, QuantAc),
    Quantidade is Quant + QuantAc.

calculaQuantidadeTotal([],0).

calculaQuantidadeTipo(Tipo,[Id|T],Quantidade):-
    nodo(Id, _, _,_,Lista),
    getResiduo(Tipo,Lista,Quant),
    calculaQuantidadeTipo(Tipo,T, QuantAc),
    Quantidade is Quant + QuantAc.

calculaQuantidadeTipo(Tipo,[_|T],Quantidade):- calculaQuantidadeTipo(Tipo,T,
    Quantidade).

calculaQuantidadeTipo(_,[],0).
```

```
merge([H|T],L,[H|Res]):- merge(T,L,Res).

merge([],L,L).

concat([A,B|T],Res) :- merge(A, B, Pr), concat([Pr|T],Res).

concat([X|[]],X).

concat([],[]).

pontoMaisPerto(A,X):-
    findall(F,arco(A,F,_),L),
    list_min(L,X).

list_min([L|Ls], Min) :-
    list_min(Ls, L, Min).

list_min([], Min, Min).
list_min([L|Ls], Min0, Min) :-
    Min1 is min(L, Min0),
    list_min(Ls, Min1, Min).

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

B.3 main.pl

```
:-include("baseDeConhecimento.pl"). % Contém os predicados para carregar a base de
    conhecimento
:-include("utilidades.pl"). % Contém os predicados para tramento de dados

:- dynamic nodo/5.
:- dynamic arco/3.
:- dynamic garagem/1.
:- dynamic deposito/1.

%Para Testes
garagem(21949).
deposito(21966).
% ------ Geraradores de circuitos ----------------------------------------------

circuitoDFS(Garagem,Circuito,[Garagem|Percurso],Distancia,QuantidadeRecolhida):-
    garagem(Garagem),pontoMaisPerto(Garagem,Inicio),
    recolhaDFS(Inicio,PontoFim,EntreNodos,QuantidadeRecolhida,15000),
    deposito(Descarga), caminhoDFS(PontoFim,Descarga,[_|Transporte]),
    caminhoDFS(Descarga,Garagem,[_|Final]),
    append(EntreNodos,Transporte,P1),
    append(P1,Final,Percurso),
    calculaDistancia(Circuito,Distancia).
```

```
circuitoDFS(Garagem,Tipo,Circuito,[Garagem|Percurso],Distancia,QuantidadeRecolhida):-
    garagem(Garagem),pontoMaisPerto(Garagem,Inicio),
    recolhaDFS(Inicio,PontoFim,EntreNodos,QuantidadeRecolhida,Tipo,15000),
    deposito(Descarga), caminhoDFS(PontoFim,Descarga,[_|Transporte]),
    caminhoDFS(Descarga,Garagem,[_|Final]),
    append(EntreNodos,Transporte,P1),
    append(P1,Final,Percurso),
    calculaDistancia(Circuito,Distancia).

circuitoDFS(Garagem,Descarga,PontoInicio,PontoFim,Circuito,Distancia,
    QuantidadeRecolhida):-
    caminhoDFS(Garagem,PontoInicio,PercusoInicial),
    recolhaDFS(PontoInicio,PontoFim,[_|Percurso],QuantidadeRecolhida,15000),
    caminhoDFS(PontoFim,Descarga,[_|PercursoDescarga]),
    caminhoDFS(Descarga,Garagem,[_|PercuroRetorno]),
    concat([PercusoInicial,Percurso,PercursoDescarga,PercuroRetorno],Circuito),
    calculaDistancia(Circuito,Distancia).


circuitoDFS(Garagem,Tipo,Descarga,PontoInicio,PontoFim,Circuito,Distancia,
    QuantidadeRecolhida):-
    caminhoDFS(Garagem,PontoInicio,PercusoInicial),
    recolhaDFS(PontoInicio,PontoFim,[_|Percurso],QuantidadeRecolhida,Tipo,15000),
    caminhoDFS(PontoFim,Descarga,[_|PercursoDescarga]),
    caminhoDFS(Descarga,Garagem,[_|PercuroRetorno]),
    concat([PercusoInicial,Percurso,PercursoDescarga,PercuroRetorno],Circuito),
    calculaDistancia(Circuito,Distancia).

circuitoBFS(Garagem,[Garagem|Percurso],Distancia,QuantidadeRecolhida):-
    garagem(Garagem),pontoMaisPerto(Garagem,Inicio),
    recolhaBFS(Inicio,PontoFim,EntreNodos,_,QuantidadeRecolhida,15000),
    deposito(Descarga), caminhoBFS(PontoFim,Descarga,[_|Transporte]),
    caminhoDFS(Descarga,Garagem,[_|Final]),
    append(EntreNodos,Transporte,P1),
    append(P1,Final,Percurso),
    calculaDistancia([Garagem|Percurso],Distancia).

circuitoBFS(Garagem,Descarga,PontoInicio,PontoFim,Circuito,Distancia,
    QuantidadeRecolhida):-
    caminhoBFS(Garagem,PontoInicio,PercusoInicial),
    recolhaBFS(PontoInicio,PontoFim,[_|Percurso],_,QuantidadeRecolhida,15000),
    caminhoBFS(PontoFim,Descarga,[X|PercursoDescarga]),
    caminhoBFS(Descarga,Garagem,[_|PercuroRetorno]),
    concat([PercusoInicial,Percurso,PercursoDescarga,PercuroRetorno],Circuito),
    calculaDistancia(Circuito,Distancia),
    calculaQuantidadeTotal([X|PercursoDescarga],QuantidadeRecolhida).

% ------ DFS ------------------------------------------------------------------

% Base

caminhoDFS(Orig,Dest,Cam):-
    depthfirst(Orig,Dest,[Orig],Cam).
```

```
    %condicao final: nodo actual = destinore

    depthfirst(Dest,Dest,LA,Cam):-
        reverse(LA,Cam).

    depthfirst(Act,Dest,LA,Cam):-
        adjacencia(Act,X,_),
        \+ member(X,LA),
        depthfirst(X,Dest,[X|LA],Cam).

% Recolhe todo o lixo até nao conseguir ir a mais nenhum nodo, ou estiver cheio

recolhaDFS(Orig,Dest,Cam,QuantRes,Max):-
    recolhaDepthfirst(Orig,Dest,[Orig],Cam,0,Max,QuantRes).
    %condicao final: nodo actual = destino

    recolhaDepthfirst(Act,Dest,LA,Cam,Quant,Max,QuantRes):-
        nodo(Act,_,_,_,Lixo),
        encher(Lixo,Quant,Max,NewQuant),
        continuarRecolhaDFS(Act,Dest,LA,Cam,NewQuant,Max,QuantRes).

    continuarRecolhaDFS(Act,Dest,LA,Cam,Quant,Max,QuantRes):-
        Quant < Max,
        adjacencia(Act,X,_),
        \+ member(X,LA),
        recolhaDepthfirst(X,Dest,[X|LA],Cam,Quant,Max,QuantRes).

    continuarRecolhaDFS(Dest,Dest,LA,Cam,Quant,_,Quant):- reverse(LA,Cam).

% Recolhe um tipo de lixo especifico até nao conseguir ir a mais nenhum nodo, ou
    estiver cheio

recolhaDFS(Orig,Dest,Cam,QuantRes,Tipo,Max):-
    recolhaDepthfirst(Orig,Dest,[Orig],Cam,0,Max,Tipo,QuantRes).
    %condicao final: nodo actual = destino

    recolhaDepthfirst(Act,Dest,LA,Cam,Quant,Max,Tipo,QuantRes):-
        nodo(Act,_,_,_,Lixo),
        encher(Tipo,Lixo,Quant,Max,NewQuant),
        continuarRecolhaDFS(Act,Dest,LA,Cam,NewQuant,Max,Tipo,QuantRes).

    continuarRecolhaDFS(Act,Dest,LA,Cam,Quant,Max,Tipo,QuantRes):-
        Quant < Max,
        adjacencia(Act,X,_),
        \+ member(X,LA),
        recolhaDepthfirst(X,Dest,[X|LA],Cam,Quant,Max,Tipo,QuantRes).

    continuarRecolhaDFS(Dest,Dest,LA,Cam,Quant,_,_,Quant):- reverse(LA,Cam).

%Profundidade Limitada

caminhoDFSLim(Orig,Dest,Cam,ProfundidadeMax):-
    depthfirstLim(Orig,Dest,[Orig],Cam,0,ProfundidadeMax).
    %condicao final: nodo actual = destino
```

```
    depthfirstLim(Dest,Dest,LA,Cam,_,_):-
        reverse(LA,Cam).

    depthfirstLim(Act,Dest,LA,Cam,Profundidade,ProfundidadeMax):-
        Profundidade < ProfundidadeMax,
        adjacencia(Act,X,_),
        \+ member(X,LA),
        NP is Profundidade + 1,
        depthfirstLim(X,Dest,[X|LA],Cam,NP,ProfundidadeMax).

% Recolhe todo o lixo até nao conseguir ir a mais nenhum nodo, ou estiver cheio

recolhaDFSLim(Orig,Dest,Cam,QuantRes,Max,ProfundidadeMax):-
    recolhaDepthfirstLim(Orig,Dest,[Orig],Cam,0,Max,QuantRes,0,ProfundidadeMax).
    %condicao final: nodo actual = destino

    recolhaDepthfirstLim(Act,Dest,LA,Cam,Quant,Max,QuantRes,Profundidade,
        ProfundidadeMax):-
        Profundidade < ProfundidadeMax,
        nodo(Act,_,_,_,Lixo),
        encher(Lixo,Quant,Max,NewQuant),
        continuarRecolhaDFSLim(Act,Dest,LA,Cam,NewQuant,Max,QuantRes,Profundidade,
            ProfundidadeMax).

    continuarRecolhaDFSLim(Act,Dest,LA,Cam,Quant,Max,QuantRes,Profundidade,
        ProfundidadeMax):-
        Quant < Max,
        adjacencia(Act,X,_),
        \+ member(X,LA),
        NP is Profundidade +1,
        recolhaDepthfirstLim(X,Dest,[X|LA],Cam,Quant,Max,QuantRes,NP,ProfundidadeMax).

    continuarRecolhaDFSLim(Dest,Dest,LA,Cam,Quant,_,Quant,_,_):- reverse(LA,Cam).

% Recolhe um tipo de lixo especifico até nao conseguir ir a mais nenhum nodo, ou
    estiver cheio

recolhaDFSLim(Orig,Dest,Cam,QuantRes,Tipo,Max,ProfundidadeMax):-
    recolhaDepthfirstLim(Orig,Dest,[Orig],Cam,0,Max,Tipo,QuantRes,0,ProfundidadeMax).
    %condicao final: nodo actual = destino

    recolhaDepthfirstLim(Act,Dest,LA,Cam,Quant,Max,Tipo,QuantRes,Profundidade,
        ProfundidadeMax):-
        Profundidade < ProfundidadeMax,
        nodo(Act,_,_,_,Lixo),
        encher(Tipo,Lixo,Quant,Max,NewQuant),
        continuarRecolhaDFSLim(Act,Dest,LA,Cam,NewQuant,Max,Tipo,QuantRes,Profundidade
            ,ProfundidadeMax).

    continuarRecolhaDFSLim(Act,Dest,LA,Cam,Quant,Max,Tipo,QuantRes,Profundidade,
        ProfundidadeMax):-
        Quant < Max,
        adjacencia(Act,X,_),
```

```
        \+ member(X,LA),
        NP is Profundidade +1,
        recolhaDepthfirstLim(X,Dest,[X|LA],Cam,Quant,Max,Tipo,QuantRes,NP,
            ProfundidadeMax).

    continuarRecolhaDFSLim(Dest,Dest,LA,Cam,Quant,_,_,Quant):- reverse(LA,Cam).



% ------ BFS ------------------------------------------------------------------
% Base

caminhoBFS(Orig,Dest,Cam):-breadthfirst(Dest,[[Orig]],Cam).

    breadthfirst(Dest,[[Dest|T]|_],Cam):-
        reverse([Dest|T],Cam).

    breadthfirst(Dest,[LA|Outros],Cam):-
        LA=[Act|_],
        findall([X|LA],
            (
                Dest\==Act,
                adjacencia(Act,X,_),
                \+ member(X,LA)
            ),
            Novos
        ),
        append(Outros,Novos,Todos),
        %chamada recursiva
        breadthfirst(Dest,Todos,Cam).


caminhoBFS(Orig,Dest,Cam,Distancia) :-
    caminhoBFS(Orig,Dest,Cam),
    calculaDistancia(Cam,Distancia).

recolhaBFS(Orig,Dest,Cam,Distancia,QuantRes,Max):-
    caminhoBFS(Orig,Dest,Cam),
    calculaDistancia(Cam,Distancia),
    calculaQuantidadeTotal(Cam,QuantRes),
    QuantRes =< Max.

recolhaBFS(Orig,Dest,Cam,Distancia,QuantRes,Tipo,Max) :-
    caminhoBFS(Orig,Dest,Cam),
    calculaDistancia(Cam,Distancia),
    calculaQuantidadeTipo(Tipo,Cam,QuantRes),
    QuantRes =< Max.

caminhoBNB(Inicio,Destino,Caminho,Distancia):-
    branchAndBound(Destino,[(0,[Inicio])],Caminho,Distancia).

    branchAndBound(Destino,[(Distancia,[Destino|T])|_],Caminho,Distancia):-
    reverse([Destino|T],Caminho).
```

```
    branchAndBound(Destino,[(Da,LA)|Outros],Caminho,Distancia):-
    LA=[Atual|_],
    findall((DistX,[X|LA]),
    (Destino\==Atual,arco(Atual,X,DistanciaX),\+ member(X,LA),
    DistX is DistanciaX + Da),Novos),
    append(Outros,Novos,Todos),
    sort(Todos,TodosOrd),
    branchAndBound(Destino,TodosOrd,Caminho,Distancia).

recolhaBNB(Inicio,Destino,Caminho,Distancia,QuantRes,Max):-
    branchAndBound(Destino,[(0,0,[Inicio])],Caminho,Distancia,QuantRes,Max).

    branchAndBound(Destino,[(Da,QuantAtual,LA)|Outros],Caminho,Distancia,QuantRes,Max)
        :-
    LA=[Atual|_],
    nodo(Atual,_,_,_,Lixo),
    encher(Lixo,QuantAtual,Max,NovaQuantidade),
    NovaQuantidade<Max,
    findall((DistX,NovaQuantidade,[X|LA]),
    (
        Destino\==Atual,
        arco(Atual,X,DistanciaX),
        \+ member(X,LA),
        DistX is DistanciaX + Da),
    Novos),
    append(Outros,Novos,Todos),
    sort(Todos,TodosOrd),
    branchAndBound(Destino,TodosOrd,Caminho,Distancia,QuantRes,Max).


branchAndBound(Destino,[(Distancia,QuantAtual,[Destino|T])|_],Caminho,Distancia,Max,
    Max):-
    nodo(Destino,_,_,_,Lixo),
    encher(Lixo,QuantAtual,Max,NovaQuantidade),
    NovaQuantidade=:=Max,
    reverse([Destino|T],Caminho).

branchAndBound(Destino,[(Distancia,QuantRes,[Destino|T])|_],Caminho,Distancia,QuantRes
    ,Max):-
    QuantRes =< Max,
    reverse([Destino|T],Caminho).


recolhaBNB(Inicio,Destino,Caminho,Distancia,QuantRes,Tipo,Max):-
    branchAndBound(Destino,[(0,0,[Inicio])],Caminho,Distancia,QuantRes,Tipo,Max).

    branchAndBound(Destino,[(Da,QuantAtual,LA)|Outros],Caminho,Distancia,QuantRes,Tipo
        ,Max):-
    LA=[Atual|_],
    nodo(Atual,_,_,_,Lixo),
    encher(Tipo,Lixo,QuantAtual,Max,NovaQuantidade),
    NovaQuantidade<Max,
    findall((DistX,NovaQuantidade,[X|LA]),
```

```
    (
        Destino\==Atual,
        arco(Atual,X,DistanciaX),
        \+ member(X,LA),
        DistX is DistanciaX + Da),
    Novos),
    append(Outros,Novos,Todos),
    sort(Todos,TodosOrd),
    branchAndBound(Destino,TodosOrd,Caminho,Distancia,QuantRes,Tipo,Max).

branchAndBound(Destino,[(Distancia,QuantAtual,[Destino|T])|_],Caminho,Distancia,Max,
    Tipo,Max):-
    nodo(Destino,_,_,_,Lixo),
    encher(Tipo,Lixo,QuantAtual,Max,NovaQuantidade),
    NovaQuantidade=:=Max,
    reverse([Destino|T],Caminho).

branchAndBound(Destino,[(Distancia,QuantRes,[Destino|T])|_],Caminho,Distancia,QuantRes
    ,_,Max):-
    QuantRes =< Max,
    reverse([Destino|T],Caminho).

% ---- A* ----------------------------------------------------------------

%Base
aEstrela(Orig,Dest,Cam,Distancia):-
    aEstrela2(Dest,[(_,0,[Orig])],Cam,Distancia).

    aEstrela2(Dest,[(_,Distancia,[Dest|T])|_],Cam,Distancia):-
        reverse([Dest|T],Cam).

    aEstrela2(Dest,[(_,Ca,LA)|Outros],Cam,Distancia):-
        LA=[Act|_],
        findall(
            (CEX,CaX,[X|LA]),
            (
                Dest\==Act,adjacencia(Act,X,DistanciaX),\+ member(X,LA),
                CaX is DistanciaX + Ca, estimativa(X,Dest,EstimativasX),
                CEX is CaX +EstimativasX
            ),
            Novos),
        append(Outros,Novos,Todos),
        sort(Todos,TodosOrd),
        aEstrela2(Dest,TodosOrd,Cam,Distancia).

aEstrelaDistanciaPorLixo(Orig,Dest,Cam,Distancia):-
    aEstrelaLPD2(Dest,[(_,0,[Orig])],Cam,Distancia).

    aEstrelaLPD2(Dest,[(_,Distancia,[Dest|T])|_],Cam,Distancia):-
        reverse([Dest|T],Cam).

    aEstrelaLPD2(Dest,[(_,Ca,LA)|Outros],Cam,Distancia):-
        LA=[Act|_],
        findall(
```

```
            (CEX,CaX,[X|LA]),
            (
                Dest\==Act,adjacencia(Act,X,DistanciaX),\+ member(X,LA),
                CaX is DistanciaX + Ca, estimativaResidPorDist(X,Dest,EstimativasX),
                CEX is CaX +EstimativasX
            ),
            Novos),
        append(Outros,Novos,Todos),
        sort(Todos,TodosOrd),
        aEstrelaLPD2(Dest,TodosOrd,Cam,Distancia).


% ------ Gulosa ------------------------------------------------------------

gulosa(Partida, Destino,Caminho,Distancia) :-
        estimativa(Partida,Destino ,Estimativa),
        gulosa2([[Partida]/0/Estimativa],Destino,InvCaminho/Distancia/_),
    reverse(InvCaminho, Caminho).


gulosa2(Cam, Destino ,Caminho) :-
        melhorCaminhoGul(Cam, Caminho),
    Caminho = [Nodo|_]/_/_,
    Nodo == Destino.

gulosa2(Cam, Destino ,SolucaoCaminho) :-
        melhorCaminhoGul(Cam,MelhorCaminho),
        seleciona(MelhorCaminho, Cam, OutrosCam),
    findall(NovoCam, adjacenteGul(MelhorCaminho,NovoCam,Destino), ExpCam),
        append(OutrosCam, ExpCam, NovoCam),
    gulosa2(NovoCam,Destino ,SolucaoCaminho).



adjacenteGul([Nodo|Caminho]/Distancia/_, [ProxNodo,Nodo|Caminho]/NovoDistancia/Est,
    Destino) :-
    adjacencia(Nodo, ProxNodo, PassoDistancia),\+ member(ProxNodo, Caminho),
    NovoDistancia is Distancia + PassoDistancia,
    estimativa(ProxNodo,Destino ,Est).



gulosaDistanciaPorLixo(Partida, Destino,Caminho,Distancia) :-
        estimativaResidPorDist(Partida,Destino ,Estimativa),
        gulosa2([[Partida]/0/Estimativa],Destino,InvCaminho/Distancia/_),
    reverse(InvCaminho, Caminho).

gulosaLPD2(Cam, Destino ,Caminho) :-
        melhorCaminhoGul(Cam, Caminho),
    Caminho = [Nodo|_]/_/_,
    Nodo == Destino.

gulosaLPD2(Cam, Destino ,SolucaoCaminho) :-
        melhorCaminhoGul(Cam,MelhorCaminho),
```

```
        seleciona(MelhorCaminho, Cam, OutrosCam),
    findall(NovoCam, adjacenteLPDGul(MelhorCaminho,NovoCam,Destino), ExpCam),
        append(OutrosCam, ExpCam, NovoCam),
    gulosaLPD2(NovoCam,Destino ,SolucaoCaminho).

adjacenteLPDGul([Nodo|Caminho]/Distancia/_, [ProxNodo,Nodo|Caminho]/NovoDistancia/Est,
    Destino) :-
    adjacencia(Nodo, ProxNodo, PassoDistancia),\+ member(ProxNodo, Caminho),
    NovoDistancia is Distancia + PassoDistancia,
    estimativaDistPorResid(ProxNodo,Destino ,Est).


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

melhorCaminhoGul([Caminho1/Distancia1/Est1,_/_/Est2|Cam], MelhorCaminho) :-
    Est1 =< Est2, !, melhorCaminhoGul([Caminho1/Distancia1/Est1|Cam], MelhorCaminho).

melhorCaminhoGul([_|Cam], MelhorCaminho) :- melhorCaminhoGul(Cam, MelhorCaminho).

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


estatisticas(Questao):-
    statistics(runtime,[Start|_]),
    Questao,
    statistics(runtime,[Stop|_]),
    Runtime is Stop-Start,
    write("Tempo: "),write(Runtime).
```