Relatório do Trabalho Individual

João Diogo Mota (a80791)

Braga, junho de 2020

Universidade do Minho
Mestrado Integrado em Engenharia Informática
Sistemas de Representação de Conhecimento e Raciocínio
(2º Semestre/2019-2020)

Resumo

Este projeto terá como objetivo avaliar a componente de métodos de resolução de problemas e pesquisa lecionada na unidade curricular de Sistemas de Representação de Conhecimento e Raciocínio. Para esse fim, foram fornecidos pela equipa docente ficheiros relativos ao sistema de transportes do concelho de Oeiras, com o intuito de resolver diversos problemas relacionados com esta temática.

Conteúdo

Lista de Figuras

1 Introdução

O principal objetivo da realização deste projeto consistiu no aperfeiçoamento da utilização da linguagem de programação em lógica "Prolog", e no aperfeiçoamento de estratégias de pesquisa leccionadas durante as aulas de Sistemas de Representação de Conhecimento e Raciocinio.

Para a realização do projeto, foi fornecido um *dataset* relativo ao sistema de transportes do concelho de Oeiras. Este *dataset* engloba características das próprias paragens do concelho, tal como o seu estado de preservação, operadora, entre outras, mas também dados que nos permitem determinar quais as paragens adjacentes para as várias carreiras existentes.

Desta forma, com a elaboração do projeto proposto é esperada a obtenção de uma maior experiência e facilidade de utilização da linguagem "Prolog", tal como um maior à vontade e conhecimento de estratégias de procura e as suas vantagens e desvantagens.

2 Preliminares

Dado que o problema apresentado é um problema essencialmente relacionado com pesquisa, é importante destacar alguns dos diversos tipos de pesquisa que existem. É possível subdividir as estratégias de pesquisa em dois subgrupos: Pesquisa Não-Informada (ou Cega); Pesquisa Informada (ou heurística).

- Pesquisa Não-Informada (Cega) : Este tipo de pesquisa utiliza apenas informações disponíveis na definição do problema. Algumas das formas de pesquisa não informada são: Primeiro em largura (*Breadth First*, BFS); Primeiro em profundidade (*Depth First*, DFS); Custo Uniforme; Pesquisa iterativa; Pesquisa Bidirecional.

- Pesquisa Informada (Heurística) : É um tipo de pesquisa em que, ao longo desta, se vai dando "dicas"ao algoritmo sobre a adequação de diferentes estados. Entre as várias formas de pesquisa não informada, destaca-se: Pesquisa Gulosa; Algoritmo A*.

Uma das estratégias de Pesquisa Não-Informada utilizada para a resolução do problema foi a pesquisa "Primeiro em largura". A ideia deste tipo de pesquisa é expandir todos os nós de menor profundidade primeiro, permitindo a análise destes antes de avançar mais um nível de profundidade. Uma das vantagens deste método é a pesquisa ser muito sistemática, embora tenha como desvantagem o seu demorado tempo e a ocupação de espaço.

```
function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  frontier ← a FIFO queue with node as the only element
  explored ← an empty set
  loop do
      if EMPTY?(frontier) then return failure
      node ← POP(frontier)  /* chooses the shallowest node in frontier */
      add node.STATE to explored
      for each action in problem.ACTIONS(node.STATE) do
          child ← CHILD-NODE(problem, node, action)
          if child.STATE is not in explored or frontier then
              if problem.GOAL-TEST(child.STATE) then return SOLUTION(child)
              frontier ← INSERT(child, frontier)
```

Figura 1: Algoritmo BFS

Quanto à Pesquisa Informada, uma das estratégias referidas foi o Algoritmo A*. O algoritmo A* é uma forma de pesquisa que escolhe sempre o melhor caminho consoante o custo para alcançar o destino. Por este motivo prevê-se que o seu uso trará maiores vantagens em termos de eficiência que a pesquisa "Primeiro em Largura"enunciada anteriormente [1].

3 Breve Descrição do Enunciado

Para a resolução do trabalho proposto pela equipa docente, será necessária a análise e processamento de dois ficheiros *.xlsx*, referentes ao sistema de transportes do concelho de Oeiras [2].

Um dos ficheiros, "paragens.xlsx", contém dados relativos às paragens de autocarro do concelho. O conhecimento relativo às paragens é definido por:

- GID - Identificador de uma paragem;
- Latitude;
- Longitude;
- Estado de Conservação;
- Tipo de Abrigo (Fechado dos lados, Aberto dos lados, Sem abrigo);
- Abrigo com Publicidade (Yes, No) - Indicando se a paragem contém publicidade;
- Operadora;
- Carreira - Nº das carreiras que têm passagem por essa paragem;
- Código da Rua;
- Nome da Rua;
- Freguesia.

O segundo ficheiro fornecido, "adjacencias.xlsx", contém uma folha de cálculo para cada carreira, indicando as paragens que pertencem a essa carreira, dando também a informação necessária das paragens adjacentes por cada carreira. Cada linha deste ficheiro contém a seguinte informação:

- Carreira correspondente;
- Paragem de origem;
- Paragem de destino.

Tendo em consideração os dados dos dois ficheiros ".xlsx"fornecidos, o pretendido numa primeira fase será a tradução dos dados para a base de conhecimento, para que possam ser utilizados pelo Prolog. Após esta tradução, será possível a resolução dos requisitos expostos no enunciado do trabalho prático, sendo eles os seguintes:

- Calcular um trajeto entre dois pontos;
- Selecionar apenas algumas das operadoras de transporte para um determinado percurso;
- Excluir um ou mais operadores de transporte para o percurso;
- Identificar quais as paragens com o maior número de carreiras num determinado percurso;
- Escolher o menor percurso (usando critério menor número de paragens);
- Escolher o percurso mais rápido (usando critério da distância);
- Escolher o percurso que passe apenas por abrigos com publicidade;
- Escolher o percurso que passe apenas por paragens abrigadas;
- Escolher um ou mais pontos intermédios por onde o percurso deverá passar.

4 Leitura dos Ficheiros fornecidos

4.1 Leitura e tradução dos ficheiros .xlsx fornecidos

Tal como referido, o primeiro passo passou por conseguir traduzir os dados fornecidos nos ficheiros ".xlsx"para conhecimento que pudesse ser utilizado pela linguagem *Prolog*.

Para este efeito, foram utilizadas as linguagens de programação*Python* e *Java*, sendo a primeira utilizada para a conversão dos dados de ".xlsx" para ".csv" e a segunda para a escrita dos dados do ficheiro ".csv" em conhecimento elegível pelo *Prolog*. Esta escolha deveu-se à facilidade pessoal da utilização da linguagem *Java*, e à facilidade conhecida do *Python* de ler ficheiros ".xlsx".

Para a leitura dos ficheiros fornecidos, foi utilizada a ferramenta de análise e manipulação de dados *Pandas*. Esta escolha permitiu que o código se tornasse relativamente pequeno e simples, como se pode ver na figura 2.

```python
import xlrd
import csv
import pandas

def csv_from_excel():
    wb = xlrd.open_workbook('adjacencias.xlsx')
    sheets = wb.sheet_names()

    for sheet in range(0,len(sheets)):
        ficheiro = "Adjacencias/adjacencia" + sheets[sheet] +".csv"

        read = pandas.read_excel('adjacencias.xlsx',sheet_name = sheets[sheet])
        read.to_csv(ficheiro,index=None,header=True)


csv_from_excel()

```

Figura 2: Código Python

Importante notar que, devido a algumas imprecisões no *dataset* fornecido, estas foram corrigidas manualmente, de forma a ser possível a sua correta leitura.

4.2 Leitura dos ficheiros ".csv" e tradução para a base de conhecimento

Para a conversão dos ficheiros ".csv" para conhecimento elegível pelo *Prolog*, foram criadas 3 classes: Adjacencia; Paragem; Read. As duas primeiras contêm os dados a serem extraídos dos diversos ficheiros ".csv" de adjacências, e do ficheiro "paragens.csv", respectivamente. A classe Read, sendo a classe principal, irá fazer a leitura dos ficheiros ".csv" e a sua escrita para o ficheiro de resolução.

A leitura do ficheiro "paragens.csv" é feito linha a linha, criando para cada linha uma instância da classe Paragem, sendo posteriormente escrita no ficheiro ".pl" no formato

$$paragem(gid, latitude, longitude, estadodeconservao, tipodeabrigo, publicidade$$

$$, operadora, listadecarreiras, cdigoderua, nomederua, freguesia).$$

```
370  paragem(102,-101969.18,-99801.53,bom,"sem abrigo",no,lt,[101],217,"rua da constituição","barcarena").
371  paragem(101,-101994.64,-99805.01,bom,"sem abrigo",no,lt,[101],217,"rua da constituição","barcarena").
372  paragem(106,-101762.99,-99819.05,bom,"sem abrigo",no,lt,[101],261,"rua da juventude","barcarena").
```

Figura 3: Exemplo da representação de Paragens na base de conhecimento

Quanto aos diversos ficheiros de adjacências, foi feito um ciclo que permitirá ler e, posteriormente, escrever no ficheiro ".pl" os dados dos vários ficheiros, um a um. Já que uma paragem é adjacente a outra se se encontrarem em linhas consecutivas nos ficheiros, no ciclo criado que percorre as várias linhas, foi feita a leitura de duas linhas de cada vez, sendo assim possível criar uma instância da classe adjacência, onde a origem da adjacência é a paragem da primeira linha do ficheiro lida no ciclo e o destino a segunda. Após a leitura de cada ficheiro, é efectuada a escrita no ficheiro ".pl" no seguinte formato:

$$adjacencia(Carreira, Origem, Destino).$$

```
1495  adjacencia(15,192,190).
1496  adjacencia(15,190,49).
1497  adjacencia(15,49,232).
```

Figura 4: Exemplo da representação de adjacências na base de conhecimento

5 Respostas às Questões Colocadas

5.1 Calcular um trajeto entre dois pontos

Para a resolução deste problema, foi necessária a utilização de um algoritmo que permita percorrer os vários nodos de adjacências existentes para que seja possível encontrar um caminho entre duas paragens especificadas à entrada.

A estratégia utilizada para percorrer as paragens, que podem ser vistas como um grafo, foi uma estratégia de pesquisa em largura *Breadth-First*.

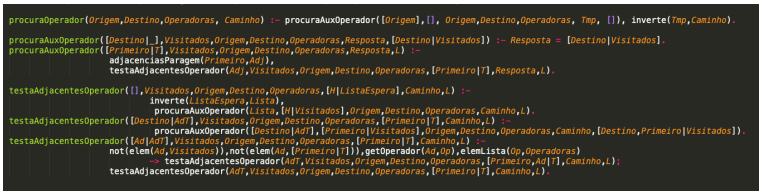
O primeiro passo é calcular as adjacências da paragem de origem, invocando o predicado "adjacenciasParagem(Paragem,Adjacentes)". Obtidas todas as paragens adjacentes, foi verificada uma a uma, percorrendo a lista de adjacentes, se a paragem em questão é a paragem destino. Caso não o seja, é verificado se esta se encontra na lista dos nodos previamente visitados e na lista de espera de nodos a visitar. Deste modo, se as condições se verificarem, é chamado este predicado novamente para o resto das adjacências e adicionando o nodo atual à lista de espera, para ser posteriormente expandido. Terminada a análise de todos os nodos adjacentes, foi realizado o mesmo processo sucessivamente para a lista de espera, até que seja encontrado o destino pretendido num dos nodos adjacentes a um nodo que está a ser visitado.

```
procura(Origem,Destino, Caminho) :- procuraAux([Origem],[], Origem,Destino,Tmp,[]),inverte(Tmp,Caminho).

procuraAux([Destino|_],Visitados,Origem,Destino,Resposta,[Destino|Visitados]) :- Resposta = [Destino|Visitados].
procuraAux([Primeiro|T],Visitados,Origem,Destino,Resposta,L) :-
                    adjacenciasParagem(Primeiro,Adj),
                    testaAdjacentes(Adj,Visitados,Origem,Destino,[Primeiro|T],Resposta,L).


testaAdjacentes([],Visitados,Origem,Destino,[H|ListaEspera],Caminho,L) :-
                            inverte(ListaEspera,Lista),
                            procuraAux(Lista,[H|Visitados],Origem,Destino,Caminho,L).
testaAdjacentes([Destino|AdT],Visitados,Origem,Destino,[Primeiro|T],Caminho,L) :-
                            procuraAux([Destino|AdT],[Primeiro|Visitados],Origem,Destino,Caminho,[Destino,Primeiro|Visitados]).
testaAdjacentes([Ad|AdT],Visitados,Origem,Destino,[Primeiro|T],Caminho,L) :-
                    not(elem(Ad,Visitados)),not(elem(Ad,[Primeiro|T])) -> testaAdjacentes(AdT,Visitados,Origem,Destino,[Primeiro,Ad|T],Caminho,L);
                    testaAdjacentes(AdT,Visitados,Origem,Destino,[Primeiro|T],Caminho,L).
```

Figura 5: Predicado que calcula um trajeto entre dois pontos

5.2 Selecionar apenas algumas das operadoras de transporte para um determinado percurso

Para a resolução desta questão, foi utilizado o mesmo método de procura mencionado na questão anterior. A única alteração esteve na inclusão de mais uma condição de verificação quando se percorrem todos os adjacentes de uma dada paragem. Para este caso, obter-se-á a operadora da paragem adjacente atual (recorrendo ao predicado que se encontra na figura 7, verificando-se de seguida se esta operadora se encontra na lista de operadoras fornecida de entrada.



```
procuraOperador(Origem,Destino,Operadoras, Caminho) :- procuraAuxOperador([Origem],[], Origem,Destino,Operadoras, Tmp, []), inverte(Tmp,Caminho).

procuraAuxOperador([Destino|_],Visitados,Origem,Destino,Operadoras,Resposta,[Destino|Visitados]) :- Resposta = [Destino|Visitados].
procuraAuxOperador([Primeiro|T],Visitados,Origem,Destino,Operadoras,Resposta,L) :-
                    adjacenciasParagem(Primeiro,Adj),
                    testaAdjacentesOperador(Adj,Visitados,Origem,Destino,Operadoras,[Primeiro|T],Resposta,L).

testaAdjacentesOperador([],Visitados,Origem,Destino,Operadoras,[H|ListaEspera],Caminho,L) :-
                        inverte(ListaEspera,Lista),
                        procuraAuxOperador(Lista,[H|Visitados],Origem,Destino,Operadoras,Caminho,L).
testaAdjacentesOperador([Destino|AdT],Visitados,Origem,Destino,Operadoras,[Primeiro|T],Caminho,L) :-
                        procuraAuxOperador([Destino|AdT],[Primeiro|Visitados],Origem,Destino,Operadoras,Caminho,[Destino,Primeiro|Visitados]).
testaAdjacentesOperador([Ad|AdT],Visitados,Origem,Destino,Operadoras,[Primeiro|T],Caminho,L) :-
                    not(elem(Ad,Visitados)),not(elem(Ad,[Primeiro|T])),getOperador(Ad,Op),elemLista(Op,Operadoras)
                        -> testaAdjacentesOperador(AdT,Visitados,Origem,Destino,Operadoras,[Primeiro,Ad|T],Caminho,L);
                    testaAdjacentesOperador(AdT,Visitados,Origem,Destino,Operadoras,[Primeiro|T],Caminho,L).
```

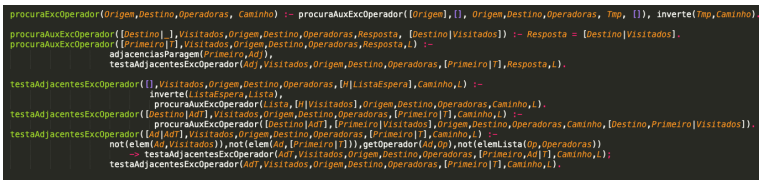
Figura 6: Predicado que calcula um trajeto entre dois pontos, passando apenas por operadoras referenciados de entrada

```
getOperador(Paragem,Op) :-
                solucoes(Operador,(paragem(Paragem,_,_,_,_,_,Operador,_,_,_,_)),Op).
```

Figura 7: Predicado que obtém a operadora para uma determinada paragem

5.3 Excluir uma ou mais operadoras de transporte para o percurso

O algoritmo de resolução desta questão foi idêntico ao da questão anterior. A única alteração realizada foi que, ao contrário da questão anterior, em que era verificado se a operadora de uma paragem se encontrava na lista de operadoras fornecida, foi obtida a operadora da carreira (utilizado o predicado da figura 7) e acrescentada uma condição que verifica se esta não se encontra na lista de operadoras a excluir do percurso.



```
procuraExcOperador(Origem,Destino,Operadoras, Caminho) :- procuraAuxExcOperador([Origem],[], Origem,Destino,Operadoras, Tmp, []), inverte(Tmp,Caminho).

procuraAuxExcOperador([Destino|_],Visitados,Origem,Destino,Operadoras,Resposta, [Destino|Visitados]) :- Resposta = [Destino|Visitados].
procuraAuxExcOperador([Primeiro|T],Visitados,Origem,Destino,Operadoras,Resposta,L) :-
                    adjacenciasParagem(Primeiro,Adj),
                    testaAdjacentesExcOperador(Adj,Visitados,Origem,Destino,Operadoras,[Primeiro|T],Resposta,L).

testaAdjacentesExcOperador([],Visitados,Origem,Destino,Operadoras,[H|ListaEspera],Caminho,L) :-
                        inverte(ListaEspera,Lista),
                        procuraAuxExcOperador(Lista,[H|Visitados],Origem,Destino,Operadoras,Caminho,L).
testaAdjacentesExcOperador([Destino|AdT],Visitados,Origem,Destino,Operadoras,[Primeiro|T],Caminho,L) :-
                        procuraAuxExcOperador([Destino|AdT],[Primeiro|Visitados],Origem,Destino,Operadoras,Caminho,[Destino,Primeiro|Visitados]).
testaAdjacentesExcOperador([Ad|AdT],Visitados,Origem,Destino,Operadoras,[Primeiro|T],Caminho,L) :-
                    not(elem(Ad,Visitados)),not(elem(Ad,[Primeiro|T])),getOperador(Ad,Op),not(elemLista(Op,Operadoras))
                        -> testaAdjacentesExcOperador(AdT,Visitados,Origem,Destino,Operadoras,[Primeiro,Ad|T],Caminho,L);
                    testaAdjacentesExcOperador(AdT,Visitados,Origem,Destino,Operadoras,[Primeiro|T],Caminho,L).
```

Figura 8: Predicado que calcula um trajeto entre dois pontos, excluindo certas operadoras fornecidas de entrada

5.4 Identificar quais as paragens com o maior número de carreiras num determinado percurso

De modo a resolver a questão, foi necessário a criação de um predicado que indicasse o número de carreiras que passam por uma determinada paragem (getNCarreiras(Paragem, NCarreiras)). Para isso, recorreu-se ao predicado auxiliar "size", que determina o número de elementos de uma lista.

De seguida, foi construído o predicado principal que, primeiramente, obtém todas as paragens de um percurso, utilizando o predicado "procura". Posteriormente, calcula o número de carreiras para cada uma das paragens, utilizando o predicado referenciado anteriormente e, por fim, ordena de forma decrescente, utilizando o predicado "SortDecres(Lista,ListaOrdenada)", de forma a que as paragens com mais carreiras apareçam primeiro na lista resultado.

Figura 9: Predicado que identifica as paragens com maior número de carreiras

5.5 Escolher o menor percurso (usando critério menor número de paragens)

Para a resolução desta questão, foi feita uma adaptação do algoritmo de procura "Primeiro em Largura". Para cada paragem ao longo do algoritmo, são calculadas as paragens adjacentes. Para cada uma das paragens adjacentes, é calculado o caminho usando o predicado "procura", e é calculado o número de nodos pelos quais este passa. Obtidos os números de nodos para cada percurso, determina-se qual deles tem o menor valor seguindo o algoritmo por essa paragem.

Figura 10: Escolher o menor percurso (usando critério menor número de paragens)

5.6 Escolher o percurso mais rápido (usando critério da distância)

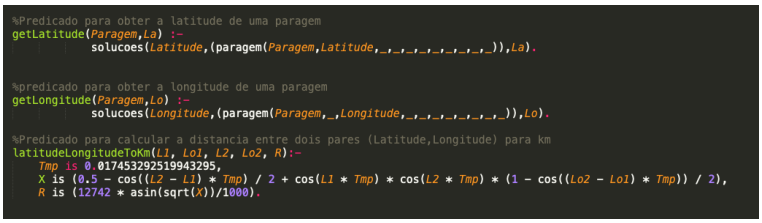
A resolução desta questão exigiu um método de pesquisa informada. O método escolhido foi o Algoritmo A*.

Tendo em conta que o critério a ser utilizado é o critério da distância, foi desenvolvido um predicado(latitudeLongitudeToKm(Latitude,Longitude, Latitude2,Longitude2,Kms)) que devolve a distância em quilómetros, consoante um par de pares latitude, longitude fornecidos.

O algoritmo A* desenvolvido tomou por base um algoritmo previamente desenvolvido nas aulas de Sistemas de Representação de Conhecimento e Raciocínio.

```
procuraMenor(Origem, Destino, Caminho/Custo) :-
    getLatitude(Origem,La1), getLongitude(Origem,Lo1), getLatitude(Destino,La2), getLongitude(Destino,Lo2),
    latitudeLongitudeToKm(La1,Lo1,La2,Lo2,Estima),
    aestrela([[Origem]/0/EstimaDistancia], Destino ,InvCaminho/Custo/_),inverte(InvCaminho, Caminho).


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

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


seleciona(E, [E|Xs], Xs).

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


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

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

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


expande_aestrela(Caminho, Destino,ExpCaminhos) :-
    findall(NovoCaminho, adjacente(Caminho,Destino,NovoCaminho), ExpCaminhos).


adjacente([Origem|Caminho]/Custo/_,Destino, [ProxNodo,Origem|Caminho]/NovoCusto/Est) :-
    eAdjacente(Origem, ProxNodo),\+ member(ProxNodo, Caminho),
    getLatitude(Origem,La1),getLongitude(Origem,Lo1),getLatitude(ProxNodo,La2),getLongitude(ProxNodo,Lo2),
    latitudeLongitudeToKm(La1,Lo1,La2,Lo2,PassoCusto),
    NovoCusto is Custo + PassoCusto,
    getLatitude(Destino,LaD),getLongitude(Destino,LoD),
    latitudeLongitudeToKm(La2,Lo2,LaD,LoD,Est).
```

Figura 11: Predicado para escolher o menor percurso (usando critério menor número de paragens)



```
%Predicado para obter a latitude de uma paragem
getLatitude(Paragem,La) :-
            solucoes(Latitude,(paragem(Paragem,Latitude,_,_,_,_,_,_,_,_,_)),La).


%predicado para obter a longitude de uma paragem
getLongitude(Paragem,Lo) :-
            solucoes(Longitude,(paragem(Paragem,_,Longitude,_,_,_,_,_,_,_,_)),Lo).

%Predicado para calcular a distancia entre dois pares (Latitude,Longitude) para km
latitudeLongitudeToKm(L1, Lo1, L2, Lo2, R):-
    Tmp is 0.017453292519943295,
    X is (0.5 - cos((L2 - L1) * Tmp) / 2 + cos(L1 * Tmp) * cos(L2 * Tmp) * (1 - cos((Lo2 - Lo1) * Tmp)) / 2),
    R is (12742 * asin(sqrt(X))/1000).
```

Figura 12:Predicados para obter a distância entre dois pares Latitude,Longitude

5.7 Escolher o percurso que passe apenas por abrigos com publicidade

De forma a resolver esta questão, foi necessário um predicado auxiliar para indicar se uma paragem tem publicidade ou não (getPublicidade(Paragem)).

Utilizando este predicado auxiliar e o predicado "procura" previamente construído, foi acrescentada a este uma nova condição aquando da verificação das paragens adjacentes a uma paragem, que permite verificar se a paragem em questão tem ou não publicidade.

Figura 13: Predicado para escolher o percurso que passe apenas por abrigos com publicidade.

5.8 Escolher o percurso que passe apenas por paragens abrigadas

De forma a resolver esta questão, foi necessário um predicado auxiliar para indicar se uma paragem é abrigada ou não (getAbrigada(Paragem)). Assim, este predicado verifica se o campo "Tipo de Abrigo"contém os dados: "fechado dos lados"ou "aberto dos lados".

Utilizando este predicado auxiliar e o predicado "procura"previamente construído, foi acrescentada a este uma nova condição aquando da verificação das paragens adjacentes a uma paragem, que permite verificar se a paragem em questão é ou não abrigada.

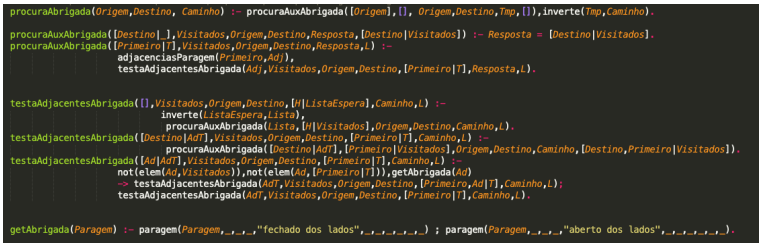


Figura 14:Predicado para escolher o percurso que passe apenas por paragens abrigadas

5.9 Escolher um ou mais pontos intermédios por onde o percurso deverá passar

Para a resolução do problema indicado, foi invocada o predicado "Procura" previamente referido. Este predicado foi sendo utilizado recursivamente para cada par de paragens entre a origem, paragens intermédias pretendidas e destino. Foi ainda utilizado o predicado pré-definido "append"de forma a concatenar as várias listas resultantes.

De notar que, para evitar repetições, foi implementado um predicado "removePrimeiro(Lista, ListaResultado)"para remover o primeiro elemento de uma lista.

```
procuraIntermedios(Origem,Destino,Intermedios,R) :- procuraIntermediosAux(Origem,Destino,Intermedios,[],Tmp),R=[Origem|Tmp].

procuraIntermediosAux(Origem,Destino,[],L,L).
procuraIntermediosAux(Origem,Destino,[Intermedio],L,R)
                :-procura(Origem,Intermedio,L1), removePrimeiro(L1,X),procura(Intermedio,Destino,L2),
                 removePrimeiro(L2,Y),append(X,Y,Temp),append(L,Temp,Nova), procuraIntermediosAux(Origem,Destino,[],Nova,R).
procuraIntermediosAux(Origem,Destino,[H|T],L,R) :-
                procura(Origem,H,L1),removePrimeiro(L1,Nova), append(L,Nova,Temp),procuraIntermediosAux(H,Destino,T,Temp,R).

removePrimeiro([X],[]).
removePrimeiro([H|T],T).
```

Figura 15: Predicado que permite escolher um ou mais pontos intermédios por onde o percurso deverá passar

6 Predicados Extra

6.1 Calcular o tempo de viagem entre duas paragens tendo em conta um tempo de espera médio

O predicado "getTempoViagem" presente na figura 16, calcula o tempo de um percurso entre duas paragens fornecidas, tendo em consideração que cada viagem entre duas paragens demora 5 minutos e fornecendo de igual forma o tempo de espera médio em cada paragem.

```
getTempoViagem(Origem,Destino,Espera,Tempo) :-
                contaProcura(Origem,Destino,R), Tempo is (((R-1)*(Espera+5))).
```

Figura 16: Predicado que calcula o tempo de viagem entre duas paragens tendo em conta um tempo de espera médio

6.2 Indicar qual a paragem dentro de uma freguesia que tem um percurso mais rápido para um determinado Destino

Este predicado pretende indicar a paragem ideal a tomar tendo em conta a freguesia e o destino. Como tal, é calculado o percurso de todas as paragens dessa freguesia, bem como o número de paragens por qual cada uma passa. Por fim, é escolhido o percurso com menor número de paragens.

De notar que este predicado não se encontra funcional, devido a conflitos com aspas e apóstrofes.

```
melhorParagem(Freguesia,Destino,Paragem) :- getParagensFreguesia(Freguesia,Par), write(Par), melhorParagemAux(Par,Destino,[],Paragem).

melhorParagemAux([],Destino,MelhorPar, Paragem) :- escolheMenor(MelhorPar,Tmp), getFirst(Tmp,Paragem).
melhorParagemAux([H|N],Destino,MelhorPar, Paragem) :- contaProcura(H,Destino,NParagens),melhorParagemAux(T,Destino,[(H,NParagens)|MelhorPar], Paragem).

getParagensFreguesia(Freguesia,Paragens) :- solucoes(Par,(paragem(Par,_,_,_,_,_,_,_,_,Freguesia)),Paragens).
```

Figura 17: Predicado que indica qual a paragem dentro de uma freguesia que tem um percurso mais rápido para um determinado destino

6.3 Devolver todas as paragens com abrigo de uma determinada rua

O predicado demonstrado na figura 18 tem como intuito devolver todas as paragens abrigadas de uma determinada rua. Para tal, é indicado de entrada o código desta, e devolvidas todas as paragens cujo campo "Tipo de Abrigo"é: "fechado dos lados"ou "aberto dos lados".

```
getParagensAbrigoRua(CodRua,Paragens) :-
                solucoes(Par,(paragem(Par,_,_,_,"fechado dos lados",_,_,_,_,_)),Tmp),
                solucoes(Par,(paragem(Par,_,_,_,"aberto dos lados",_,_,_,_,_)),Tmp2),append(Tmp,Tmp2,Paragens).
```

Figura 18: Predicado que indica qual a paragem dentro de uma freguesia que tem um percurso mais rápido para um determinado destino

7 Testes e Análise de Resultados

- Execução do predicado "Procura" para descobrir um caminho entre as paragens 183 e 185.

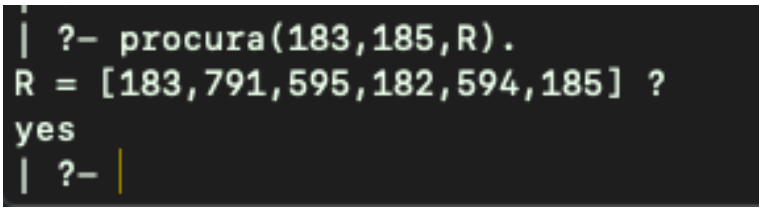


Figura 19: Execução do predicado "Procura"

- O predicado "procuraOperador" pode ser verificado calculando o caminho entre as paragens 183 e 185 apenas com passagem por paragens da operadora "vimeca"; e com passagem por paragens das operadoras "vimeca"e "scotturb". Tendo em conta as figuras 19 e 20, é possível verificar que as paragens resultantes da invocação do predicado "procura(183,185,R)." são da operadora "vimeca", excepto a paragem 182, que é da operadora "scotturb".

  Quando é invocado o predicado "procuraOperador" entre 183 e 185 apenas com a operadora "vimeca", terá de ser procurado um percurso alternativo à paragem 182 por esta ser da operadora "scotturb".

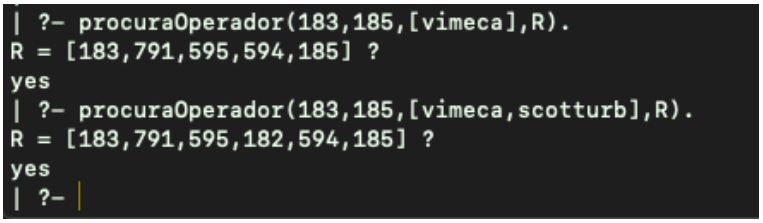


Figura 20: Execução do predicado "ProcuraOperador"

- Ao invés de se selecionar as paragens com certas operadores pelas quais um caminho pode passar, nesta questão a intenção era escolher as paragens com operadoras pelas quais não se pretende passar. Para tal, utilizou-se o exemplo do caminho entre 183-185 sem que esta possa passar por "scotturb", já que, por padrão, passaria por uma paragem com operadora "scotturb". Tal como é possível verificar pela figura 8 , foi procurado um caminho alternativo à paragem 182 por esta ser da operadora não pretendida, resultando no mesmo caminho da figura 20.

  paragem

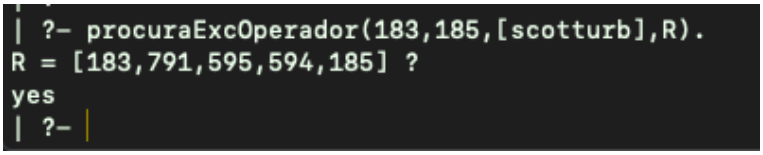


Figura 21: Execução do predicado "procuraExc"

- Para o teste do predicado "numeroCarreiras" , foi utilizado novamente o caminho 183-185. Como resultado, foi obtida uma lista de pares constituída pelos GID das paragens e números de carreiras que passam por esta, ordenada por ordem decrescente do número de carreiras.

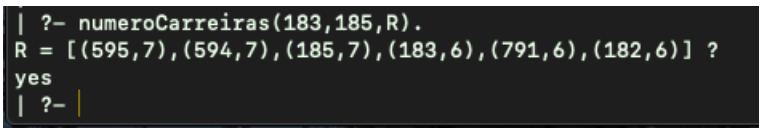


Figura 22: Execucão do predicado "numeroCarreiras"

- Para o teste do predicado "procuraMenorParagens", foi utilizado novamente o percurso 183-185 para teste. Como podemos ver pela figura 23, tal como com o algoritmo "A*" apresentado de seguida, também este consegue obter um melhor caminho que o algoritmo de pesquisa em largura puro.

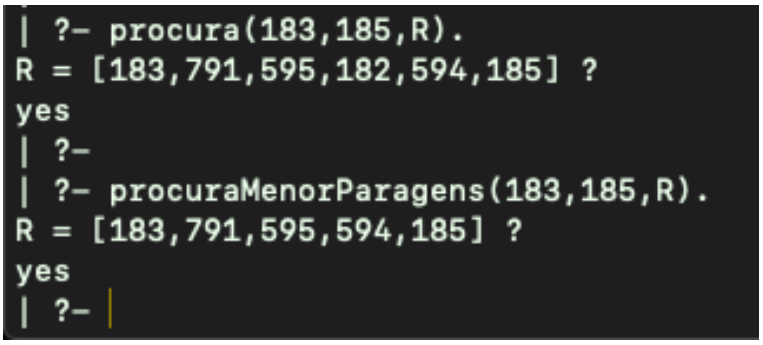


Figura 23: Execução do predicado "procuraMenorParagens"

- Como este predicado faz uso da pesquisa informada A*, é de esperar que obtenha o menor caminho entre duas paragens. Para tal, foi utilizado o caminho 183-185, de forma a se poder fazer uma comparação com o resultado obtido na figura 19. Como se pode ver na figura 24, com o algoritmo A*, o caminho 183-185 teve paragens em mais 3 carreiras além das inicial e final, enquanto que na pesquisa não-informada primeiro em largura, teve paragem em mais 4 além das terminais.

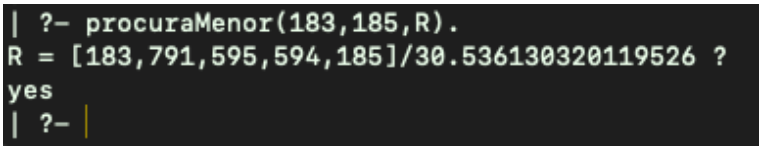


Figura 24: Execução do predicado "procuraMenor"

- Para o teste do predicado "procuraPublicidade", foram utilizados dois caminhos diferentes: 183-182 e 183-185. Como se pode verificar pela figura 25, de 183 para 182 foi encontrado um caminho passando apenas por paragens com publicidade, enquanto entre 183 e 185 tal não se verificou.

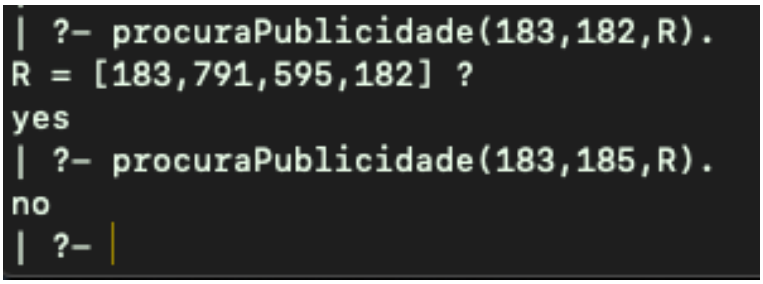


Figura 25: Execução do predicado "procuraPublicidade"

- Tal como na execução do predicado "procuraPublicidade", para a execução do "procuraAbrigada" foram utilizados dois caminhos diferentes: 183-185 e 745-161. Como se vê na figura 26, é possível realizar o caminho entre 183 e 185 passando apenas por paragens abrigadas, enquanto que para o caminho 745-161, o mesmo não é possível.

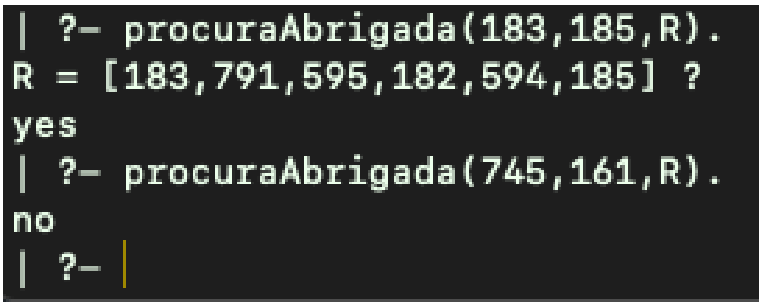


Figura 26: Execução do predicado "procuraAbrigada"

- De forma a comprovar o funcionamento do predicado "procuraIntermedios", foi comparado o caminho entre 183 e 185 utilizando o predicado "procura"e, posteriormente, o predicado "procuraIntermedios". Como tal, iremos obter caminhos diferentes para cada um dos predicados, sendo isto possível comprovar pela figura 27.

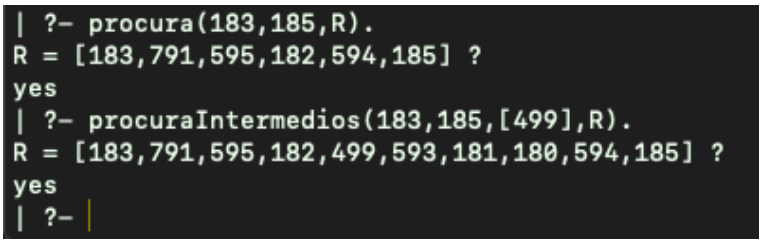


Figura 27: Execução do predicado "procuraIntermedios"

- Para demonstrar o predicado "getTempoViagem", foi utilizado o percurso 183-185 e um tempo de espera médio de três minutos.

Figura 28: Execução do predicado "getTempoViagem"

- De forma a comprovar o funcionamento do predicado "getParagensAbrigoRua", foi utilizado o exemplo da rua número 157.

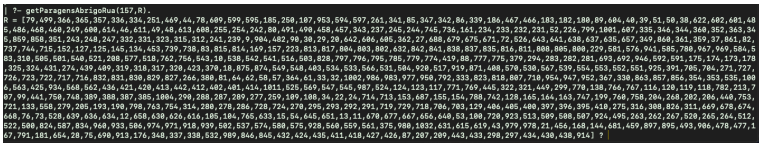


Figura 29: Execução do predicado "getParagensAbrigoRua"

7.1 Comparação de resultados

Na figura 30 encontra-se uma comparação de resultados entre pesquisa Não-Informada com o algoritmo "Primeiro em largura"e pesquisa Informada com o algoritmo "A*". Regra geral, e tal como era de prever, na maioria dos casos o algoritmo "A*" consegue obter resultados mais favoráveis do que o algoritmo "Primeiro em largura", dado que se trata de um algoritmo de pesquisa ótima.

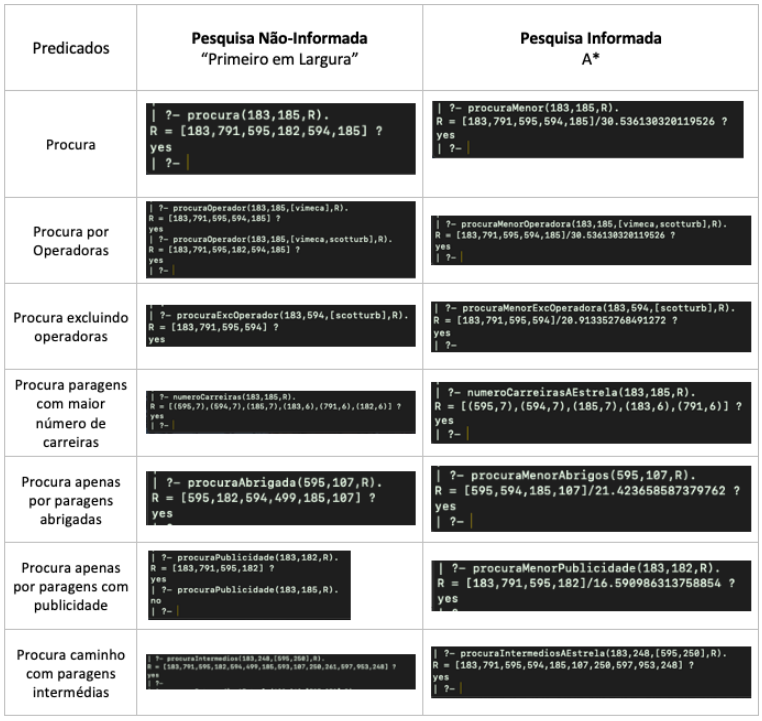


Figura 30: Comparação de performance dos algoritmos "Primeiro em Largura"e "A*"

8 Conclusão

O desenvolvimento do projecto permitiu o aperfeiçoamento e ganho de novas competências relativamente à linguagem "Prolog", e a mecanismos de pesquisa na mesma, bem como a necessidade e importância da sua implementação em situações reais.

Ao longo da realização do projeto, deparei-me com questões tais como quais as melhores abordagens a seguir para a concretização de certas funcionalidades requeridas para o projeto. Para além disso, a maior dificuldade esteve na correta implementação de alguns algoritmos, tais como o algoritmo "A*", cuja realização se deveu ao algoritmo base desenvolvido juntamente com o docente das aulas de Sistemas de Representação de Conhecimento e Raciocínio, durante as sessões das aulas práticas.

Desta forma, foram estudadas diversas estratégias de pesquisa, bem como as suas vantagens e desvantagens, permitindo a elucidação sobre que estratégias utilizar para cada caso.

Assim, verificou-se, através dos testes realizados, o esperado para cada tipo de algoritmo aplicado a cada situação.

Numa retrospetiva, de uma forma global, verifica-se o sucesso dos objetivos propostos para a realização deste projeto.

9 Referências

Referências

[1] MÉtodos de resoluÇÃo de problemas e de procura. *MÉTODOS DE RESOLUÇÃO DE PROBLEMAS E DE PROCURA*.

[2] Paragens de autocarro.

10 Anexos

10.1 Parser dos dados

10.1.1 .xlsx → .csv

```
import xlrd
import csv
import pandas

def csv_from_excel():
    wb = xlrd.open_workbook('adjacencias.xlsx')
    sheets = wb.sheet_names()

    for sheet in range(0,len(sheets)):
       ficheiro = "Adjacencias/adjacencia" + sheets[sheet] +".csv"

       read = pandas.read_excel('adjacencias.xlsx',sheet_name = sheets[sheet])
       read.to_csv(ficheiro,index=None,header=True)


csv_from_excel()
```

10.1.2 .csv → Base de Conhecimento

```
public class Adjacencia {

    private int carreira;
    private int origem;
    private int destino;

    public Adjacencia(int carreira, int origem, int destino) {
        this.carreira = carreira;
        this.origem = origem;
        this.destino = destino;
    }

    public int getCarreira() {
        return carreira;
    }

    public int getOrigem() {
        return origem;
    }
```

```
    public int getDestino() {
        return destino;
    }

    public void setCarreira(int carreira) {
        this.carreira = carreira;
    }

    public void setOrigem(int origem) {
        this.origem = origem;
    }

    public void setDestino(int destino) {
        this.destino = destino;
    }

    @Override
    public String toString() {
        return "adjacencia(" +
                carreira +
                "," + origem +
                "," + destino +
                ").";
    }
}
```

```
    import java.util.List;

public class Paragem {
    private int gid;
    private double latitude;
    private double longitude;
    private String estado;
    private String tipo;
    private String publicidade;
    private String operador;
    private List<Integer> carreiras;
    private int codigo;
    private String rua;
    private String freguesia;

    public Paragem(int gid, double latitude, double longitude,
    String estado, String tipo, String publicidade, String operador,
    List<Integer> carreiras, int codigo, String rua, String freguesia) {
        this.gid = gid;
        this.latitude = latitude;
        this.longitude = longitude;
        this.estado = estado;
        this.tipo = tipo;
        this.publicidade = publicidade;
        this.operador = operador;
        this.carreiras = carreiras;
        this.codigo = codigo;
        this.rua = rua;
        this.freguesia = freguesia;
    }

    public void setGid(int gid) {
        this.gid = gid;
    }

    public void setLatitude(double latitude) {
        this.latitude = latitude;
    }

    public void setLongitude(double longitude) {
        this.longitude = longitude;
    }

    public void setEstado(String estado) {
        this.estado = estado;
    }
```

```
public void setPublicidade(String publicidade) {
    this.publicidade = publicidade;
}

public void setTipo(String tipo) {
    this.tipo = tipo;
}

public void setOperador(String operador) {
    this.operador = operador;
}

public void setCarreiras(List<Integer> carreiras) {
    this.carreiras = carreiras;
}

public void setCodigo(int codigo) {
    this.codigo = codigo;
}

public void setRua(String rua) {
    this.rua = rua;
}

public void setFreguesia(String freguesia) {
    this.freguesia = freguesia;
}

public int getGid() {
    return gid;
}

public double getLatitude() {
    return latitude;
}

public double getLongitude() {
    return longitude;
}

public String getEstado() {
    return estado;
}

public String getTipo() {
```

```
        return tipo;
    }

    public String getPublicidade() {
        return publicidade;
    }

    public String getOperador() {
        return operador;
    }

    public List<Integer> getCarreiras() {
        return carreiras;
    }

    public int getCodigo() {
        return codigo;
    }

    public String getRua() {
        return rua;
    }

    public String getFreguesia() {
        return freguesia;
    }

    @Override
    public String toString() {
        return "paragem(" +
                 gid +
                "," + latitude +
                "," + longitude +
                "," + estado  +
                "," + '"' + tipo  + '"'+
                "," + publicidade +
                "," + operador  +
                "," + carreiras +
                "," + codigo +
                "," + '"' + rua + '"' +
                "," + '"'+ freguesia + '"' +
                ").";
    }
}
```

```
    import java.io.*;
import java.nio.charset.Charset;
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
//import org.apache.poi.openxml4j.exceptions.InvalidFormatException;

public class read {


    public static void main(String[] args) throws IOException {

        String teste = "TesTe";
         teste = teste.toLowerCase();
         System.out.println(teste);


        List<Paragem> paragens =
        readParagens("/Users/joaomota/Desktop/Trabalho_Individual/paragens.csv");

        writeFicheiro(paragens);



        File dir =
        new File("/Users/joaomota/Desktop/Trabalho_Individual/Adjacencias");
        File[] directoryListing = dir.listFiles();
        int i=0;
        if (directoryListing != null) {
            for (File child : directoryListing) {
                //System.out.println("NOMEEEEEEEEEEEEEEEEEE" + child.getName());
                List<Adjacencia> adjacencias = readAdjacencias(child.getName());
                writeFicheiroAdjacencias(adjacencias);
            }
        }


    }

    public static List<Paragem> readParagens(String file) {
```

```
    List<Paragem> paragens = new ArrayList<Paragem>();
    Path pathToFile = Paths.get(file);
    try (BufferedReader br =
    Files.newBufferedReader(pathToFile, Charset.forName("UTF-8"))) {
        String line = br.readLine();
        line = br.readLine();
        while (line != null) {
            String[] attributes = line.split(";");
            Paragem paragem = createParagem(attributes);
            paragens.add(paragem);
            line = br.readLine();
        }
    } catch (IOException ioe) {
        ioe.printStackTrace();
    }
    return paragens;
}

private static Paragem createParagem(String[] metadata) {
    int gid = Integer.parseInt(metadata[0]);
    Double latitude = Double.parseDouble(metadata[1]);
    Double longitude = Double.parseDouble(metadata[2]);
    String estado = metadata[3].toLowerCase();
    String tipo = metadata[4].toLowerCase();
    String publicidade = metadata[5].toLowerCase();
    String operador = metadata[6].toLowerCase();
    String[] c = metadata[7].split(",");
    List<Integer> carreiras = new ArrayList<Integer>();
    for (String carreira : c)
        carreiras.add(Integer.parseInt(carreira));
    int codigo = Integer.parseInt(metadata[8]);
    String rua = metadata[9].toLowerCase();
    String freguesia = metadata[10].toLowerCase();

    return new Paragem(gid, latitude, longitude, estado, tipo, publicidade,
    operador, carreiras, codigo, rua, freguesia);
}

public static void writeFicheiro(List<Paragem> paragens) {
    //Abrir o ficheiro
    try {
        File myObj =
        new File("/Users/joaomota/Desktop/Trabalho_Individual
        /projetoIndividual.prolog.BB.pl");
        if (myObj.createNewFile()) {
            System.out.println("File created: " + myObj.getName());
```

```
        } else {
            System.out.println("File already exists.");
        }
    } catch (IOException e) {
        System.out.println("An error occurred.");
        e.printStackTrace();
    }

    // Escrever para o ficheiro
    try {
        FileWriter myWriter =
        new FileWriter("/Users/joaomota/Desktop/Trabalho_Individual
        /projetoIndividual.prolog.BB.pl", true);
        for (Paragem p : paragens) {

            myWriter.write(p.toString() + '\n');
            //myWriter.flush();
        }
        myWriter.close();
    } catch (IOException e) {
        System.out.println("An error occurred.");
        e.printStackTrace();
    }


}


public static List<Adjacencia> readAdjacencias(String file){
    List<Adjacencia> adjacencias = new ArrayList<Adjacencia>();
    Path pathToFile = Paths.get("Adjacencias/"+file);
    //File in = new File(pathToFile);
    try (BufferedReader br
    = Files.newBufferedReader(pathToFile, Charset.forName("UTF-8"))) {
        String line = br.readLine();

        line = br.readLine();
        int res=0;
       /* String[] attributes;
        while(line!=null) {
            attributes = line.split(";");
            res++;
        }*/
        int i=0;
        while (line != null) {
           // System.out.println("Teste passagem");
```

```
            String[] attributes = line.split(",");
            line = br.readLine();
          //  System.out.println(line);
            if(line==null) break;
            String[] attributes2 = line.split(",");
            Adjacencia adjacencia = createAdjacencia(attributes,attributes2);
            if(adjacencia!=null)adjacencias.add(adjacencia);
            //line = br.readLine();
            //if(line==null) break;
           // System.out.println(line);
            //attributes = line.split(",");
            i++;
        }
    } catch (IOException ioe) {
        ioe.printStackTrace();
    }
    return adjacencias;
}


public static Adjacencia createAdjacencia(String[] metadata,String[] metadata2){
    int carreira = Integer.parseInt(metadata[7]);
    int origem = Integer.parseInt(metadata[0]);
    int destino = Integer.parseInt(metadata2[0]);
   // System.out.println("Adjacencia : " +carreira + origem + " | "+destino);
    if(origem!=destino)
    return new Adjacencia(carreira,origem,destino);
    else return null;
}


public static void writeFicheiroAdjacencias(List<Adjacencia> adjacencias){
    //Abrir o ficheiro
    try {
        File myObj =
        new File("/Users/joaomota/Desktop/Trabalho_Individual
        /projetoIndividual.prolog.BB.pl");
        if (myObj.createNewFile()) {
            System.out.println("File created: " + myObj.getName());
        } else {
            System.out.println("File already exists.");
        }
    } catch (IOException e) {
        System.out.println("An error occurred.");
        e.printStackTrace();
    }
```

```
// Escrever para o ficheiro
try {
    FileWriter myWriter =
    new FileWriter("/Users/joaomota/Desktop/Trabalho_Individual
    /projetoIndividual.prolog.BB.pl", true);
    for (Adjacencia a : adjacencias) {

        myWriter.write(a.toString() + '\n');
        //myWriter.flush();
    }
    myWriter.close();
} catch (IOException e) {
    System.out.println("An error occurred.");
    e.printStackTrace();
}}}
```

10.2 Código Prolog

10.2.1 Predicados Auxiliares

```
%Predicado solucoes:
solucoes(T,P,L)
:- findall(T,P,L).

%Predicado para obter todas as adjacencias de uma paragem
adjacenciasParagem(P,R)
:- solucoes((Destino),adjacencia(_,P,Destino),T),sort(T,R).

%Predicado para verificar se uma paragem é adjacente a outra
eAdjacente(Origem,Destino) :- adjacencia(_,Origem,Destino).

%Predicado para verificar se um elemento existe numa lista
elem(X,[]) :- !,fail.
elem(X,[L]) :- X==L.
elem(X,[H|T]) :- X == H; elem(X,T).

%Predicado para verificar se uma lista se encontra presente noutra
elemLista([X],[]) :- !,fail.
elemLista([],X).
elemLista([X|XS],[Y|YS]) :- X==Y -> elemLista(XS,YS); elemLista([X|XS],YS).

%Predicado que elimina todas as ocorrencias de um elemento de uma lista
delT(X,[],[]).
delT(X,[X],[]).
delT(X,[H|T],R) :- X == H -> delT(X,T,R); delT(X,T,P), R = [H|P].


%Predicado de negação
not( Questao ) :-
    Questao, !, fail.
not( Questao ).


%Predicado que inverte uma lista
init([X],[]).
init([H|T],R) :- init(T,P), R = [H|P].

last([X],X).
last([H|T],R) :- last(T,R).


inverte([],[]).
```

```
inverte([X],[X]).
inverte(L,R) :-
last(L,P), init(L,T),inverte(T,W), R=[P|W].


%Predicado que devolve o tamanho de uma lista
size([],0).
size([H|T],R) :- size(T,P), R is P+1.


%Predicado que devolve o primeiro elemento de um paragem
getFirst((X,Y),X).
```