

UNIVERSIDADE DO MINHO

Sistema de Transportes

MESTRADO INTEGRADO EM ENGENHARIA
INFORMÁTICA

SISTEMAS DE REPRESENTAÇÃO DE CONHECIMENTO E RACIOCÍNIO

2º SEMESTRE 2019/20

Autor:
Etienne Costa

Docente:
Paulo Novais

5 de Junho de 2020

Conteúdo

1	Resumo	2
2	Introdução	3
3	Preliminares	4
3.1	Programação em lógica e PROLOG	4
4	Base de conhecimento	5
4.1	Entidades	5
4.1.1	Adjacente e Paragens	5
5	Problemas Propostos	7
5.1	Calcular um trajecto entre dois pontos	7
5.2	Selecionar apenas algumas operadoras de transporte para um de- terminado percurso	10
5.3	Excluir um ou mais operadores de transporte para o percurso	12
5.4	Identificar quais as paragens com o maior número de carreiras num determinado percurso	14
5.5	Escolher o menor percurso usando critério menor número de paragens	15
5.6	Escolher o percurso mais rápido usando critério da distância	15
5.7	Escolher o percurso que passe apenas por abrigos com publicidade .	16
5.8	Escolher o percurso que passe apenas por paragens abrigadas	17
5.9	Escolher um ou mais pontos intermédios por onde o percurso de- verá passar	17
6	Conclusão	18
7	Referências Bibliográficas	19
8	Funções Auxiliares	20

1 Resumo

O presente trabalho tem como principal objetivo aprofundar os conhecimentos na linguagem de programação em lógica PROLOG.

Com base nisso foi desenvolvido o relatório explicando o desenvolvimento do trabalho prático no âmbito da unidade curricular de Sistemas de Representação de Conhecimento e Raciocínio.

Este trabalho desenvolvido consiste na implementação de um sistema, que permite importar os dados relativos às paragens de autocarro, e representá-los numa base de conhecimento, podendo ser possível ter um sistema de recomendação de transportes públicos.

2 Introdução

O relatório apresentado diz respeito ao trabalho proposto no âmbito da unidade curricular de Sistemas de Representação de Conhecimento e Raciocínio, utilizando a linguagem de programação em lógica PROLOG. O universo de discurso para qual estamos a desenvolver este sistema é o de transportes públicos, sendo assim a base de conhecimento consiste em adjacentes e paragens de modo a ser possível fazer a representação de rotas realizadas por determinadas carreiras .

3 Preliminares

Para o desenvolvimento deste projeto foi necessário alguns conhecimentos previamente adquiridos de programação em lógica, e a utilização da linguagem PROLOG. Este conhecimento foi absorvido durante as aulas de Sistemas de Representação de Conhecimento e Raciocínio, e também com alguma pesquisa. De seguida, é apresentado alguns conceitos fundamentais para a compreensão e realização deste trabalho.

3.1 Programação em lógica e PROLOG

Uma das principais ideias da programação em lógica é de que um algoritmo é constituído por dois elementos disjuntos: a lógica e o controle. O componente lógico corresponde à definição do que deve ser solucionado, enquanto que o componente de controle estabelece como a solução pode ser obtida. O programador precisa somente descrever o componente lógico de um algoritmo, deixando o controle da execução para ser exercido pelo sistema de programação em lógica utilizado. Em outras palavras, a tarefa do programador passa a ser simplesmente a especificação do problema que deve ser solucionado, razão pela qual as linguagens lógicas podem ser vistas simultaneamente como linguagens para especificação formal e linguagens para a programação de computadores. O paradigma fundamental da programação em lógica é o da programação declarativa, em oposição à programação procedimental típica das linguagens convencionais. Um programa em lógica é então a representação de determinado problema ou situação expressa através de um conjunto finito de um tipo especial de sentenças lógicas denominadas cláusulas. Pode-se então expressar conhecimento (programas e/ou dados) em Prolog por meio de cláusulas de dois tipos: fatos e regras. Um fato denota uma verdade incondicional, enquanto que as regras definem as condições que devem ser satisfeitas para que uma certa declaração seja considerada verdadeira. Como fatos e regras podem ser utilizados conjuntamente, nenhum componente dedutivo adicional precisa ser utilizado. Além disso, como regras recursivas e não-determinismo são permitidos, os programadores podem obter descrições muito claras, concisas e não-redundantes da informação que desejam representar. Como não há distinção entre argumentos de entrada e de saída, qualquer combinação de argumentos pode ser empregada. Os termos "programação em lógica" e "programação Prolog" tendem a ser empregados indistintamente. Deve-se, entretanto, destacar que a linguagem Prolog é apenas uma particular abordagem da programação em lógica.

4 Base de conhecimento

Um programa em Prolog é um conjunto de axiomas e de regras de inferência definindo relações entre objectos que descrevem um dado problema. A este conjunto chama-se normalmente base de conhecimento.

A base de conhecimento do sistema desenvolvido é essencial à representação do conhecimento e raciocínio, tendo em conta o sistema foram desenvolvidas as seguintes entidades:

- adjacente: Carreira, GidO, GidD, DistOD, Estado, Abrigo, Publicidade, Operadora, Código, Rua, Freguesia.
- paragem: Gid, Carreira, Latitude, Longitude, Estado, Abrigo, Publicidade, Operadora, Código, Rua, Freguesia.

Toda informação representada na base de conhecimento foi extraída de um dataset pré-processado com recurso a um parser desenvolvido em Java. A representação do conteúdo do dataset facilitou a construção da base de conhecimento visto que o mesmo era um conjunto de paragens representadas como uma lista de adjacência sendo que a transição entre paragens representava um arco e com base nisso foi possível desenvolver um grafo. De forma a sintetizar a informação optei por calcular as distâncias entre vértices adjacentes e converter as mesmas para KM. Relativamente as paragens foram adicionadas a base de conhecimento de modo a conservar informação relevante para calcular heurísticas através da distância euclidiana.

4.1 Entidades

Nesta secção são apresentadas e caracterizadas as Entidades acima propostas.

4.1.1 Adjacente e Paragens

A entidade Adjacente e Paragem estão fortemente relacionadas visto que acabam por conter quase a mesma informação, a grande diferença encontra-se em dois átomos que estão presentes no facto adjacente, átomos esses que representam a adjacência e a distância ao mesmo, podendo fazer a representação de um grafo. De seguida são identificados e caracterizados os seguintes átomos:

- Carreira: Itinerário de transportes públicos.
- Gid, GidO, GidD: Identificador para um ponto no mapa.
- DistOD: Distância euclidiana entre dois pontos no mapa.
- Estado: Estado de conservação de um determinado percurso.
- Abrigo: Informação referente a paragens.
- Latitude: é a distância ao Equador medida ao longo do meridiano de Greenwich.
- Longitude: é a distância ao meridiano de Greenwich medida ao longo do Equador.
- Operadora: Empresa transportes públicos colectivos.
- Código.
- Rua.

- Freguesia.
- Publicidade.

5 Problemas Propostos

5.1 Calcular um trajecto entre dois pontos

Para o cálculo de um trajecto entre dois pontos foram usados 3 algoritmos diferentes, sendo os três de pesquisa não informada:

```
%  
  
trajecto(O,D,T):- trajectoAux(O,[D],T),  
                  printf(T).  
  
trajectoAux(O,[O|T1],[O|T1]).  
trajectoAux(O,[D|T1],T):- adjacente(Carreira,X,D,_,_,_,_,Operadora,_,_,_),  
                             \+ memberchk(X,[D|T1]),  
                             trajectoAux(O,[X,(Carreira,Operadora,D)|T1],T).  
  
%
```

Aplicando este predicado as paragens 791 e 499 obtém-se o seguinte resultado, cuja a interpretação é a seguinte:

Um passageiro que se encontra na paragem 791 pode chegar a paragem 499 apanhando o autocarro número 1 da operadora Vimeca tendo o ponto 595 como primeira paragem, de seguida pelo mesmo autocarro segue para a paragem 182 tendo a necessidade de mudar de autocarro e operadora para chegar ao destino pretendido.

```
| ?- trajecto(791,499,Trajecto).  
-----  
PARAGEM: 791  
-----  
PARAGEM: 1,Vimeca,595  
-----  
PARAGEM: 1,Vimeca,182  
-----  
PARAGEM: 1,SCoTTURB,499  
-----  
Trajecto = [791,(1,'Vimeca',595),(1,'Vimeca',182),(1,'SCoTTURB',499)] ?  
yes  
| ?- █
```

Figura 1: Trajecto entre dois pontos.


```

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

resolveDF(Origem, Destino, [Origem | Solucao]): -
    assert(goal(Destino)),
        resolvedf(Origem, [Origem], Solucao).
    printf([Origem | Solucao]).

resolvedf(Node, __, []): -
    goal(Node),
    !,
    clean.

resolvedf(Node, Historico, [ProxNodo | Solucao]): -
    adjacente(Node, ProxNodo),
    \+(member(ProxNodo, Historico)),
    resolvedf(ProxNodo, [ProxNodo | Historico], Solucao).

%-----

```

```

| ?- resolveDF(183,613,Trajecto).
Trajecto = [183,791,595,182,499,593,181,180,594,185,89,107,250,261,597,953,609,242,255,604,628,39,50,599,40,985,608,249,254,622,51,44,251,38,620,45,614,46,42,600,602,601,48,49,612,613] ?
yes
| ?-

```

Figura 2: Trajecto entre dois pontos Depth First Search .

```

%-----Breadth First Search-----

resolveBF(Origin, Destiny, Visited) :-
    resolvebf([Origin],[],RevVisited, Destiny),
    removeNotConnected(RevVisited, Visited).

resolvebf([Destiny|_], History, [Destiny|History], Destiny).
resolvebf([Node|RestQ], History, RevVisited, Destiny) :-
    findall(NextNode, (adjacente(Node,NextNode,_), \+ member(NextNode, History)),
    append(RestQ, Successors, Queue),
    resolvebf(Queue, [Node|History], RevVisited, Destiny).

%-----

```

```

| ?- resolveBF(183,613,Trajecto).
Trajecto = [183,171,799,599,860,601,48,49,612,613] ?
yes
| ?-

```

Figura 3: Trajecto entre dois pontos Breadth First Search .

É de realçar que a implementação de diferentes algoritmos leva-nos por vezes a obter um resultado melhor. Um pequeno exemplo é o que acontece na implementação de duas travessias diferentes entre os pontos 183 e 613 obtendo um percurso com menos paragens com a travessia em largura.

```

| ?- resolveBF(183,613,Trajecto,Km).
-----
PARAGEM: 183
-----
PARAGEM: 171
-----
PARAGEM: 799
-----
PARAGEM: 599
-----
PARAGEM: 860
-----
PARAGEM: 601
-----
PARAGEM: 48
-----
PARAGEM: 49
-----
PARAGEM: 612
-----
PARAGEM: 613
-----
Trajecto = [183,171,799,599,860,601,48,49,612,613],
Km = 9.6644 ?
yes
| ?-

```

Figura 4: Trajecto entre dois pontos Breadth First Search com KM .

5.2 Selecionar apenas algumas operadoras de transporte para um determinado percurso

Para a implementação deste predicado bastou acrescentar uma pequena condição que verifica se o arco formado entre duas paragens é percorrido através de uma operadora que faça parte da lista. Foram implementados 3 predicados diferentes , sendo os três de pesquisa não informada:

```
%
-----

percursoCAO(O,D, Operadoras ,T): -
percursoCAOAux(O,[D] , Operadoras ,T) ,
printf(T).

percursoCAOAux(O,[O|T1] ,_,[O|T1] ) .
percursoCAOAux(O,[D|T1] , Operadoras ,T): -
adjacente( Carreira ,X,D,_,_,_,_, Operadora ,_,_,_) ,
memberchk( Operadora , Operadoras ) ,
\+ memberchk(X,[D|T1] ) ,
percursoCAOAux(O,[X,( Carreira , Operadora ,D)|T1] , Operadoras ,T) .
%
-----
```

```
| 7- percursoCAO(128,161,['Vimeca','SCoTTURB'],R).
PARAGEM: 128
-----
PARAGEM: 12,Vimeca,745
-----
PARAGEM: 2,Vimeca,736
-----
PARAGEM: 2,Vimeca,147
-----
PARAGEM: 2,Vimeca,156
-----
PARAGEM: 2,Vimeca,734
-----
PARAGEM: 2,Vimeca,161
-----
R = [128,(12,'Vimeca',745),(2,'Vimeca',736),(2,'Vimeca',147),(2,'Vimeca',156),(2,'Vimeca',734),(2,'Vimeca',161)] ?
yes
| 7- █
```

Figura 5: Trajecto com algumas operadoras.

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

```
resolveDFCAO(Origem, Destino, Operadoras, [Origem | Solucao]):-  
    assert(goal(Destino)),  
        resolvedfcao(Origem, [Origem], Operadoras, Solucao),  
    printf([Origem | Solucao]).
```

```
resolvedfcao(Node, __, __, []):-  
    goal(Node),  
    !,  
    clean.
```

```
resolvedfcao(Node, Historico, Operadoras, [ProxNodo | Solucao]):-  
    adjacente(__, Node, ProxNodo, __, __, __, Operadora, __, __, __),  
    member(Operadora, Operadoras),  
    \+(member(ProxNodo, Historico)),  
    resolvedfcao(ProxNodo, [ProxNodo | Historico], Operadoras, Solucao).
```

%-----

%-----Breadth First Search-----

```
resolveBFCAO(Origin, Destiny, Operadoras, Visited, Km):-  
    resolvebfcao([Origin], [], RevVisited, Destiny, Operadoras),  
    removeNotConnected(RevVisited, Visited),  
    printf(Visited),  
    custoTotal(Visited, Km).
```

```
resolvebfcao([Destiny | __], History, [Destiny | History], Destiny, Operadoras).  
resolvebfcao([Node | RestQ], History, RevVisited, Destiny, Operadoras):-  
    findall(NextNode, (adjacente(__, Node, NextNode, __, __, __, Operadora, __, __, __),  
        member(Operadora, Operadoras),  
        \+ member(NextNode, History), \+ member(NextNode, RestQ)), Successors),  
    append(RestQ, Successors, Queue),  
    resolvebfcao(Queue, [Node | History], RevVisited, Destiny, Operadoras).
```

%-----

5.3 Excluir um ou mais operadores de transporte para o percurso

Para a implementação deste predicado bastou acrescentar uma pequena condição que verifica se o arco formado entre duas paragens não é percorrido através de uma operadora que faça parte da lista de Operadoras. Foram implementados 3 predicados diferentes , sendo os três de pesquisa não informada:

```
%
-----

percursoSAO(O,D, Operadoras ,T): -
percursoSAOAux(O,[D] , Operadoras ,T).

percursoSAOAux(O,[O|T1] ,_,[O|T1] ).
percursoSAOAux(O,[D|T1] , Operadoras ,T): -
adjacente( Carreira ,X,D,_,_,_,_, Operadora ,_,_,_) ,
\+ memberchk( Operadora , Operadoras ) ,
\+ memberchk(X,[D|T1] ) ,
percursoSAOAux(O,[X,( Carreira , Operadora ,D)|T1] , Operadoras ,T) .
%
-----
```

```
| 7- percursoSAO(128,161,['SCoTTURB'],R).
PARAGEM: 128
-----
PARAGEM: 12,Vimeca,745
-----
PARAGEM: 2,Vimeca,736
-----
PARAGEM: 2,Vimeca,147
-----
PARAGEM: 2,Vimeca,156
-----
PARAGEM: 2,Vimeca,734
-----
PARAGEM: 2,Vimeca,161
-----
R = [128,(12,'Vimeca',745),(2,'Vimeca',736),(2,'Vimeca',147),(2,'Vimeca',156),(2,'Vimeca',734),(2,'Vimeca',161)] ? yes
| 7-
| 7- █
```

Figura 6: Trajecto sem algumas operadoras.

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

```
resolveDFS AO(Origem, Destino, Operadoras, [Origem | Solucao]):-  
  assert(goal(Destino)),  
  resolvedfsao(Origem, [Origem], Operadoras, Solucao).
```

```
resolvedfsao(Node, __, __, []):-  
  goal(Node),  
  !,  
  clean.
```

```
resolvedfsao(Node, Historico, Operadoras, [ProxNodo | Solucao]):-  
  adjacente(__, Node, ProxNodo, __, __, __, Operadora, __, __, __),  
  \+(member(Operadora, Operadoras)),  
  \+(member(ProxNodo, Historico)),  
  resolvedfsao(ProxNodo, [ProxNodo | Historico], Operadoras, Solucao).
```

%-----

%-----Breadth First Search-----

```
resolveBFS AO(Origin, Destiny, Operadoras, Visited, Km):-  
  resolvebfsao([Origin], [], RevVisited, Destiny, Operadoras),  
  removeNotConnected(RevVisited, Visited),  
  printf(Visited),  
  custoTotal(Visited, Km).
```

```
resolvebfsao([Destiny | __], History, [Destiny | History], Destiny, Operadoras).  
resolvebfsao([Node | RestQ], History, RevVisited, Destiny, Operadoras):-  
  findall(NextNode, (adjacente(__, Node, NextNode, __, __, __, Operadora, __, __, __),  
  \+ member(Operadora, Operadoras), \+ member(NextNode, History),  
  \+ member(NextNode, RestQ)), Successors),  
  append(RestQ, Successors, Queue),  
  resolvebfsao(Queue, [Node | History], RevVisited, Destiny, Operadoras).
```

%-----

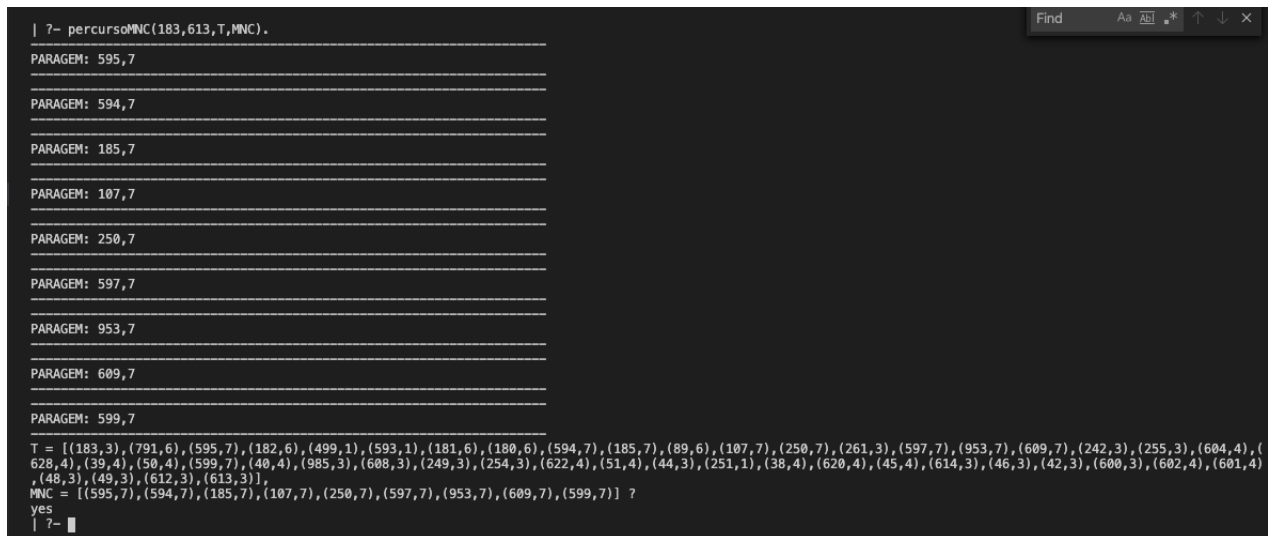
5.4 Identificar quais as paragens com o maior número de carreiras num determinado percurso

Para este predicado optei por calcular um determinado trajecto entre dois pontos e na construção deste trajecto fui calculando o número de carreiras que chegam a cada ponto deste trajecto, em seguida é feito o cálculo das paragens que possuem o maior número de carreiras.

```
%
percursoMNC(O,D,T,Final):-
percursumnc(O,[D],T,Final),
printf(Final).

percursumnc(O,[O|T1],[ (O,Size)|T1],Final):-
findall(Cs,adjacente(Cs,_,_,_,_,_,_,_,_,_),Car),
length(Car,Size).

percursumnc(O,[D|T1],T,Final):-
findall(C,adjacente(C,_,D,_,_,_,_,_,_,_),Carreiras),
length(Carreiras,Size),
adjacente(_,X,D,_,_,_,_,_,_,_),
\+(memberchk(X,[D|T1])),
percursumnc(O,[X,(D,Size)|T1],T,Final),
maiorNC(T,Maior),
foreachNC(T,Maior,Final).
%
```



```
| 7- percursoMNC(183,613,T,MNC).
PARAGEM: 595,7
PARAGEM: 594,7
PARAGEM: 185,7
PARAGEM: 107,7
PARAGEM: 250,7
PARAGEM: 597,7
PARAGEM: 953,7
PARAGEM: 609,7
PARAGEM: 599,7
T = [(183,3),(791,6),(595,7),(182,6),(499,1),(593,1),(181,6),(180,6),(594,7),(185,7),(89,6),(107,7),(250,7),(261,3),(597,7),(953,7),(609,7),(242,3),(255,3),(604,4),(628,4),(39,4),(50,4),(599,7),(40,4),(985,3),(608,3),(249,3),(254,3),(622,4),(51,4),(44,3),(251,1),(38,4),(620,4),(45,4),(614,3),(46,3),(42,3),(600,3),(602,4),(601,4),(48,3),(49,3),(612,3),(613,3)],
MNC = [(595,7),(594,7),(185,7),(107,7),(250,7),(597,7),(953,7),(609,7),(599,7)] ?
yes
| 7-
```

Figura 7: Maior número de carreiras em um trajecto.

5.5 Escolher o menor percurso usando critério menor número de paragens

Este algoritmo por sua vez não é dos mais eficientes devido ao facto de calcular todos os trajectos entre dois pontos e escolher a lista com menor comprimento visto que isto corresponde ao menor número de paragens.

```
percursoMNP(O,D,T):- findall((S,NrParagens),(resolveBF(O,D,S,_),length(S,NrParagens)),
minimo(L,T).
```

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

5.6 Escolher o percurso mais rápido usando critério da distância

Para este algoritmo decidi implementar o A* de modo a tirar partido de uma estimativa ao destino que por sua vez é calculada através da distância euclidiana.

```
resolve_aestrela(Origem, Destino, Caminho/Custo) :-
assert(goal(Destino)),
estima(Origem, Estima),
aestrela([[Origem]/0/Estima], InvCaminho/Custo/_),
inverso(InvCaminho, Caminho).
```

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

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

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

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

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

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

```
adjacenteEstrela([Nodo|Caminho]/Custo/_ , [ProxNodo, Nodo|Caminho]/NovoCusto/Est) :-
```



```

adjacente (Nodo, ProxNodo, PassoCusto),
\+ member(ProxNodo, Caminho),
NovoCusto is Custo + PassoCusto,
estima (ProxNodo, Est).

```

```

estima (Origem, Estima):-
goal (Destino),
paragem (Origem,_,Px,Py,_,_,_,_,_,_,_),
paragem (Destino,_,Qx,Qy,_,_,_,_,_,_,_),
distance (Px/Py,Qx/Qy,Estima).

```

```

PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
| 7- resolve_aestrela(791,499,Caminho/Custo).
Caminho = [791,595,182,499],
Custo = 3.1239999999999997 ?
yes
| 7- █

```

Figura 8: A* .

5.7 Escolher o percurso que passe apenas por abrigos com publicidade

```

%-----
percursoACP(O,D,T):- percursoacp(O,[D],T).
percursoacp(O,[O|T1],[O|T1]).
percursoacp(O,[D|T1],T):-
adjacente (Carreira,X,D,_,_,_,Publicidade,Operadora,_,_,_),
memberchk(Publicidade,['Yes']),
\+ memberchk(X,[D|T1]),
percursoacp(O,[X,(Carreira,Operadora,D,Publicidade)|T1],T).
%-----

```

```

| 7- percursoACP(268,827,T).
T = [268,(122,'LT',830,'Yes'),(112,'LT',832,'Yes'),(112,'LT',827,'Yes')] ?
yes
| 7- █

```

Figura 9: Abrigos com publicidade.

```
%-----Depth First Search-----
```

```
resolveDFACP(Origem, Destino, [Origem | Solucao]):-
assert(goal(Destino)),
resolvedfacp(Origem, [Origem], Solucao).

resolvedfacp(Node, _, []):-
goal(Node),
!,
clean.

resolvedfacp(Node, Historico, [ProxNodo | Solucao]):-
adjacente(_, Node, ProxNodo, _, _, _, 'Yes', _, _, _, _),
\+(member(ProxNodo, Historico)),
resolvedfacp(ProxNodo, [ProxNodo | Historico], Solucao).
```

```
%-----
```

5.8 Escolher o percurso que passe apenas por paragens abrigadas

```
percursoPA(O,D,T):- percursopa(O,[D],T).
percursopa(O,[O|T1],[O|T1]).
percursopa(O,[D|T1],T):- adjacente(Carreira,X,D,_,_,Abrigo,_,Operadora,_,_,_),
memberchk(Abrigo,['Aberto dos Lados','Fechado dos Lados']),
\+ memberchk(X,[D|T1]),
percursopa(O,[X,(Carreira,Operadora,D,Abrigo)|T1],T).
```

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

```
percursoCPI(O,D,Intermedios,T):-
findall(Or,adjacente(_,Or,_,_,_,_,_,_,_,_),Partidas),
findall(Des,adjacente(_,_,Des,_,_,_,_,_,_,_),Destinos),
sort(Partidas,R1),
sort(Destinos,R2),
mapMemberChk(Intermedios,R1),
mapMemberChk(Intermedios,R2),
percursocpi(O,[D],Intermedios,T),
printf(T).
```

```
percursocpi(O,[O|T1],[],[O|T1]).
percursocpi(O,[D|T1],Intermedios,T):-
adjacente(Carreira,X,D,_,_,_,_,Operadora,_,_,_),
memberchk(X,Intermedios),
apagaT(X,Intermedios,I),
\+ memberchk(X,[D|T1]),
percursocpi(O,[X,(Carreira,Operadora,D)|T1],I,T).
```

```
percursocpi(O,[D|T1],Intermedios,T):-
adjacente(Carreira,X,D,_,_,_,_,Operadora,_,_,_),
\+ memberchk(X,Intermedios),
```

$\backslash + \text{memberchk}(X, [D|T1]) ,$
 $\text{percursocpi}(O, [X, (\text{Carreira} , \text{Operadora} , D) | T1] , \text{Intermedios} , T) .$

6 Conclusão

A realização deste trabalho prático permitiu consolidar o conhecimento adquirido ao longo das aulas, no que concerne a utilização de algoritmos de pesquisa informada e não informada. Embora tenha feito tudo que foi proposto, houve uma dificuldade muito grande de lidar com a quantidade de informação que havia na base de conhecimento, visto que o prolog tem muitas limitações no que concerne a memória e a tentativa optimização de algoritmos que a partida já são óptimos não foi algo possível de fazer. Fora isto foi possível integrar um sistema capaz de responder os melhores trajectos entre zonas no concelho de oeiras e conseguir aproximar os resultados obtidos a um contexto real.

Sendo que uma das principais dificuldades encontradas no desenvolvimento deste sistema passou pela forma como era feita a evolução de conhecimentos face aos diferentes tipos de conhecimento, sendo assim no que concerne a melhorias , passa por num futuro próximo inserir outros factos e regras de modo a aproximar o trabalho de um contexto mais real.

7 Referências Bibliográficas

Referências

- [1] Ivan Bratko. *"PROLOG: Programming for Artificial Intelligence"*.

8 Funções Auxiliares

```
%----- Extras
% Lista todas as operadoras do sistema de transporte.

operadoras(L):- findall(Operadora, paragem(_,_,_,_,_,_,_, Operadora,_,_,_), R),
                sort(R,L).

%----- Dist ncia Euclidiana

distance(P1/P2,Q1/Q2,D):- X is exp((Q1-P1),2),
                          Y is exp((Q2-P2),2),
                          K is sqrt(X+Y)*0.001,
                          truncate(K,4,D).

%----- Truncate

truncate(X,N,Result):- X >= 0,
                      Result is floor(10^N*X)/10^N,
                      !.

%----- Estimativa

estima(Origem,Estima):- goal(Destino),
                      paragem(Origem,_,Px,Py,_,_,_,_,_,_,_),
                      paragem(Destino,_,Qx,Qy,_,_,_,_,_,_,_),
                      distance(Px/Py,Qx/Qy,Estima).

%----- MapLength

mapLength([],[]).
mapLength([H|Tail],[L1|R]):- length(H,L1),
                             mapLength(Tail,R).

%----- Clean -----

clean:- findall(G, goal(G),R1),
        foreachG(R1).

%----- Foreach Goal -----
```

```

forEachG ( []).
forEachG ([H|T]):- retract(goal(H)),
                  forEachG(T).

```

```

%viii. Calcula o maior de um conjunto de valores.
maximum([],R):- write('Empty List '),
                !,
                fail.
maximum([X],X).
maximum([Head|Tail],R):- maximum(Tail,Rest),
                        maior(Head,Rest,R).

```

```

maior(X,Y,X):- X>Y,
              !.
maior(X,Y,Y).

```

```

maiorNC([(_,R)],R).
maiorNC([Head|Tail],R):- maiorNC(Tail,Rest),
                        p2(Head,K),
                        maior(K,Rest,R).

p2((A,B),B).

```

```

%
```

```

foreEachNC([],_,[]).
foreEachNC([(A,B)|Tail],B,[(A,B)|R]):- foreEachNC(Tail,B,R).
foreEachNC([(A,B)|Tail],K,R):- foreEachNC(Tail,K,R).

```

```

%——Apaga todas as ocorrências de um dado elemento numa lista

```

```

apagaT(X,[],[]).
apagaT(X,[X|Tail],R):- apagaT(X,Tail,R).
apagaT(X,[Head|Tail],[Head|R]):- apagaT(X,Tail,R).

```

```

%
```

```

%—— Verifica se todos os elementos de uma lista pertencem a outra

```

```

mapMemberChk([],_).
mapMemberChk([H|T],L):- memberchk(H,L),
                        mapMemberChk(T,L).

```

```
%
%----- Adjacente

adjacente(Nodo, ProxNodo, Custo):-
adjacente(_, Nodo, ProxNodo, Custo, _, _, _, _, _, _, _).

adjacente(Nodo, ProxNodo, Custo):-
    adjacente(_, ProxNodo, Nodo, Custo, _, _, _, _, _, _, _).
```

```
%----- Equals
```

```
equals([], []).
equals([H|T], [H|T]).
```

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

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

```
%----- Selecciona
```

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

```
%----- Custo
```

```
custoTotal([X], 0).
custoTotal([X,Y|Tail], Result):- adjacente(X,Y,K),
                                custoTotal([Y|Tail], Temp),
                                Result is K + Temp.
```

```
% ----- Fun o auxiliar
auxiliar([], Acc, Acc).
auxiliar([Elem], Acc, Result) :- auxiliar([], [Elem|Acc], Result).
auxiliar([Node1,Node2|Rest], Acc, Result) :-
\+ adjacente(Node1,Node2,_),
auxiliar([Node1|Rest], Acc, Result).
auxiliar([Node1,Node2|Rest], Acc, Result) :-
```

```
adjacente(Node1,Node2,_,_),  
auxiliar([Node2|Rest],[Node1|Acc],Result).  
  
removeNotConnected(List,Result):- auxiliar(List,[],Result).
```