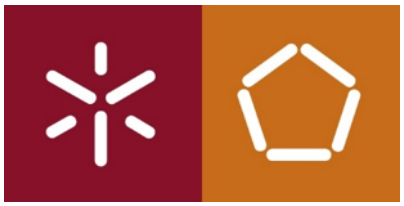

Universidade do Minho

Sistemas de Representação de Conhecimento e Raciocínio
(3º ano de curso, 2º semestre)

Componente Individual

Relatório de Desenvolvimento

Mestrado Integrado em Engenharia Informática

Realizado pelo aluno:
João da Cunha e Costa, a84775

5 de Junho de 2020

Resumo

O trabalho prático, de teor individual, realizado no âmbito da unidade curricular *Sistemas de Representação de Conhecimento e Raciocínio* incidiu no desenvolvimento de um sistema capaz de incorporar dados referentes ao sistema de transportes do concelho de Oeiras, representando-os numa base de conhecimento.

Neste documento será elucidado a forma como o problema foi abordado com vista a alcançar uma solução viável para o mesmo.

O principal foco deste trabalho prático passa por melhorar as competências ao nível da programação em lógica, resolução de problemas e no desenvolvimento de algoritmos de pesquisa, utilizando a linguagem de programação *PROLOG*.

Conteúdo

1 Introdução

O principal objetivo deste projeto é desenvolver um sistema capaz de tratar dados referentes às paragens de autocarro do concelho de Oeiras, e representá-los numa base de conhecimento. Para além disso, é também objetivo , a resolução de problemas e algoritmos de pesquisa.

Relativamente ao software adotado para o desenvolvimento deste projeto, utilizou-se o software **Pandas**, na linguagem **Python**, no que toca à parte do carregamento dos dados e o **SWI-Prolog**, no que toca à parte do processamento dos mesmos.

2 Descrição do Trabalho e Análise de Resultados

Este capítulo está fragmentado em diferentes secções, com vista numa explanação estruturada do projeto desenvolvido.

Primeiramente, é abordado a estratégia da leitura dos dados para formatos que o *Prolog* consiga processar.

De seguida, é exposta a **Base de Conhecimento** desenvolvida, apresentando as diversas abordagens à mesma.

Na terceira secção, é abordado os diferentes predicados dos algoritmos de pesquisa utilizados.

Finalmente, é apresentada a execução dos casos prátics.

2.1 Leitura de Dados

Uma vez que os dados referentes às paragens de autocarro de Oeiras estavam guardadas em formato **Excel**, foi necessário converter estes dados para ficheiros do tipo **.csv**.

Posteriormente, depois de alguma pesquisa, concluiu-se que a ferramenta **Pandas**, que utiliza a linguagem **Python**, seria uma ferramenta adequada para a leitura e tratamento dos ficheiros **.csv**.

Segue-se o programa desenvolvido para converter a informação de ficheiros **Excel** para ficheiros **.pl**:

```
In [2]: import pandas as pd

path = "D:/Projetos/Uni/3ano/2sem/SRCR/Dados/CSV/"

fl = pd.read_csv(path+"NomesCsvs.txt",header = None)

basefile = open(path+"BaseConhecimento.pl","w", encoding="utf-8")

for index,row in fl.iterrows():
    print("----------------------NOVA DF----------------------")
    print(row[0])
    df = pd.read_csv(path+str(row[0]))
    for index2,row2 in df.iterrows():
        if(index2==0):
            basefile.write('paragem(')
            basefile.write(str(row2[7])+","+str(index2+1)+","+str(row2[0])+","+str(row2[1])+","+str(row2[2])+",'"+str(row2[3])+"'
            basefile.write(").")
            basefile.write("\n")
        if((index2!=0)and((df.loc[index2,'gid'])!=(df.loc[index2-1,'gid']))):
            basefile.write('paragem(')
            basefile.write(str(row2[7])+","+str(index2+1)+","+str(row2[0])+","+str(row2[1])+","+str(row2[2])+",'"+str(row2[3])+"'
            basefile.write(").")
            basefile.write("\n")

for index,row in fl.iterrows():
    print("----------------------NOVA DF ADJ----------------------")
    print(row[0])
    df = pd.read_csv(path+str(row[0]))
    for index2,row2 in df.iterrows():
        if((index2!=0)and((df.loc[index2,'gid'])!=(df.loc[index2-1,'gid']))):
            basefile.write('arco((')
            basefile.write(str(df.loc[index2-1,'Carreira'])+","+str(index2)+","+str(df.loc[index2-1,'gid'])+",'"+str(df.loc[index
            basefile.write("("+str(row2[7])+","+str(index2+1)+","+str(row2[0])+",'"+str(row2[6])+"','"+str(row2[4])+"','"+str(row
            basefile.write(").")
            basefile.write("\n")
            basefile.write('arco(')
            basefile.write("("+str(row2[7])+","+str(index2+1)+","+str(row2[0])+",'"+str(row2[6])+"','"+str(row2[4])+"','"+str(row
            basefile.write("("+str(df.loc[index2-1,'Carreira'])+","+str(index2)+","+str(df.loc[index2-1,'gid'])+",'"+str(df.loc[i
            basefile.write(").")
            basefile.write("\n")


basefile.close()
```

Figura 1: Programa de leitura e tratamento dos dados

De referir que, com esta abordagem, é praticada a remoção de paragens que apareciam repetidas consecutivamente, o que se verificava em alguns ficheiros.

2.2 Base de Conhecimento

A principal entidade deste projeto, **paragem** de autocarro, é composta por diversos campos, nomeadamente **gid**, que representa o identificador da paragem; **latitude**, como o nome indica representa a latitude a que se encontra a paragem; **longitude**, que analogamente à latitude representa a longitude da paragem; **Estado de Conservação**, que representa se a paragem está em **Bom** ou **Mau** estado; **Tipo de Abrigo**, que representa o tipo de abrigo que a paragem tem; **Abrigo com Publicidade**, que representa se um abrigo tem ou não publicidade; **Operadora**, que representa a operadora a que pertence aquela **paragem**; **Carreira**, que representa a qual carreira pertence aquela paragem; **Código de Rua**, que representa o código da rua em que a paragem se encontra; **Nome de Rua**, que refere-se ao nome da rua em que a paragem se encontra; **Freguesia**, referente à freguesia em que a paragem se encontra.

Ao longo do desenvolvimento deste projeto, sempre que pertinente, o formato da **Base de Conhecimento** foi sofrendo alterações.

Primeiramente, definiu-se o predicado **paragem**. Para além dos campos referidos previamente, foi adicionado o campo **id**, para representar o identificador do paragem dentro de certa carreira.

```
% Carreira,id,gid,latitude,longitude,Estado de Conservacao,Tipo de Abrigo,Abrigo com Publicidade,Operadora,Codigo de Rua,Nome da Rua,Freguesia
:- dynamic paragem/12.
```

Figura 2: Estrutura predicado **paragem**

Posteriormente, devido a motivos relacionados com os algoritmos de pesquisa utilizados, motivos esses explicados mais à frente, foi adotada outro formato para o predicado **paragem**, em vez de termos um predicado , e um id para por paragem ,por cada carreira, utilizamos uma relação de adjacência.

Ou seja, uma **paragem** que seja seguida por outra, dizem-se **adjacentes** e formam um **arco**, como se segue:

```
% ((Carreira,id,gid),(Carreira,idAdjacente,gidAdjacente))
:- dynamic arco/2.
```

Figura 3: Estrutura predicado inicial de **arco**

Noutra fase, devido a problemas com a implementação de uns predicados, em vez de se guardar na **Base de Conhecimento** um predicado de arco por cada adjacência existente, passou-se a guardar o dobro. Ou seja se à paragem **X** se segue a paragem **Y**, em vez de se representar só: **arco((Carreira,IdX,GidX),(Carreira,IdY,GidY))**; Passou-se também a guardar o arco contrário: **arco((Carreira,IdY,GidY),(Carreira,IdX,GidX))**.

Por fim, de forma a facilitar a elaboração do caso prático(queries), foi adicionado ao predicado de **arco** três novas informações: **Operadora**, **Tipo de Abrigo** e **Abrigo com Publicidade**.

Sendo que o predicado final utilizado é o seguinte:

```
% ((Carreira,id,gid,Operadora,Tipo de Abrigo,Abrigo com Publicidade), (Carreira,idAdjacente,gidAdjacente,Operadora,Tipo de Abrigo,Abrigo com Publicidade))
:- dynamic arco/2.
```

Figura 4: Estrutura predicado final de **arco**

2.3 Estratégias de Pesquisa

2.3.1 Pesquisa Não Informada

Primeiramente, se quisermos ir da paragem **Start** para a paragem **Finish**, optou-se por utilizador um algoritmo de pesquisa em profundidade, não informada, em que se encontra o primeiro arco com **Gid** igual a **Start** e vai-se avançando pelos arcos adjacentes até se encontrar um arco com **Gid** igual a **Finish**.

```
%Profundidade , não informada, com ajdacências
zefAdj2(Start, Finish, Path,N):-
    arco((_,_,Finish,_,_,_),(_,_,_,_,_,_)),
    zefa2(Start, Finish, [Start], Path),
    length(Path, N).


zefa2(Gid, Gid, _, [Gid]).
zefa2(Start, Finish, Visited, [Start | Path]) :-
    arco((_,_,Start,_,_,_),(_,_,X,_,_,_)),
    \+(memberchk(X, Visited)),
    zefa2(X, Finish, [X | Visited], Path).
```

Figura 5: Pesquisa em profundidade não informada

2.3.2 Pesquisa Informada

Após algum tempo de testes ao algoritmo anterior, verificou-se que este ainda falhava em diversos casos e portanto foi pensada uma estratégia para o melhorar.

A abordagem encontrada baseava-se em 4 fases, na **1ª fase** é verificado se existe algum arco com a paragem **Final** pretendida, caso se verifice, na **2ª fase** verifica-se se a paragem **Final** pertence a uma carreira isolada, se pertencer só é bem sucedido se a paragem **Inicial** também estiver na mesma carreira. Se o anterior suceder,na **3ª fase** tenta-se procurar carreiras que a paragem **Atual** e a **Final** tenham em comum, caso haja faz-se logo caminho direto. Se não houver paragens em comum, é a **4ª fase**, em que se avança para o adjacente como o algoritmo anterior.

```
%Pesquisa informada, em que verifica as carreiras comuns
zefAdj(Start, Finish, Path,N):-
     arco((_,_,Finish,_,_,_),(_,_,_,_,_,_)),
     (isoladaG(Start,Finish),print("Carreira Isolada");
     zefa(Start, Finish, [Start], Path),
     length(Path, N)).

isoladaG(Start,Finish):-
     isolada(Finish,C),
     !,
     \+arco((C,_,Start,_,_,_),_).

isolada(Finish,C):-
     arco((C,_,Finish,_,_,_),_),
     findall(B,arco((C,_,B,_,_,_),_),L),
     sort(L,P),
     length(P,N),
     succ(N,T),
     aux(C,1,T).

aux(_,I,I).
aux(C,I,N):-
     arco((C,I,X,_,_,_),_),
     findall(S,arco((S,_,X,_,_,_),_),L),
     sort(L,P),
     length(P,1),
     succ(I,T),
     aux(C,T,N).

zefa(Gid, Gid,_, [Gid]).
zefa(Gid, Gid,_, _).
zefa(Start, Finish, Visited, [Start | Path]) :-
     (comuns(Start,Finish,Path),
     zefa(Finish, Finish, Visited, Path));
     (arco((_,_,Start,_,_,_),(_,_,X,_,_,_)),
     \+(memberchk(X, Visited)),
     zefa(X, Finish, [X | Visited], Path)).
```

Figura 6: Pesquisa informada

```
comuns(Ini,Fim,Cam):-
     comunsL(Ini,Fim,[C|_]),
     arco((C,A,Ini,_,_,_),_),
     arco((C,B,Fim,_,_,_),_),
     cam(Ini,Fim,C,A,B,[Ini],[_|Cam]).

sub(X,Y):-
     Y is X-1.

cam(Gid, Gid, _, _, _, _, [Gid]).
cam(Start, Finish, C, A, B, Visited, [Start | Path]) :-
     arco((C,I,Start,_,_,_),_),
     (B>A -> succ(I,T) ; sub(I,T)),
     arco((C,T,X,_,_,_),_),
     \+(memberchk(X, Visited)),
     cam(X, Finish, C, A, B, [X | Visited], Path).

comunsL(Ini,Fim,CarrCom):-
     findall(CI,arco((CI,_,Ini,_,_,_),_),CarrI),
     sort(CarrI,PI),
     findall(CF,arco((CF,_,Fim,_,_,_),_),CarrF),
     sort(CarrF,PF),
     intersection(PI,PF, CarrCom),
     \+length(CarrCom,0).
```

Figura 7: Pesquisa informada(Continuação)

2.3.3 Análise

Apesar do algoritmo de pesquisa informada não funcionar tão bem como inicialmente pensou-se que iria funcionar e ainda apresentar um ou outro caso em que falhe, penso que em geral é melhor que o de pesquisa não informada, visto que, este resolve mais caminhos e em geral caminhos mais curtos.

2.4 Caso Prático

Uma vez que o algoritmo de pesquisa informada é melhor que o de pesquisa não informada, este serviu de base às queries do caso prático e portanto não diferem muito , em geral, do algoritmo anterirormente explicado.

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

```
%Q2Selecionar apenas algumas das operadoras de transporte para um determinado percurso,versao com comuns
query2G(Start, Finish, Operadoras, Path,N):-
     arco((_,_,Finish,OPF,_,_),(_,_,_,_,_,_)),
     memberchk(OPF, Operadoras),
     (isoladaG(Start,Finish),print("Carreira Isolada");
     query2(Start, Finish, Operadoras,[Start], Path),
     length(Path, N)).

query2(Gid, Gid,_, _, [Gid]).
query2(Gid, Gid,_, _, _).
query2(Start, Finish, Operadoras, Visited, [Start | Path]) :-
     (comunsQ2(Start,Finish,Operadoras,Path),
     query2(Finish, Finish, Operadoras, Visited, Path));
     (arco((_,_,Start,OPF,_,_),(_,_,X,_,_,_)),
     memberchk(OPF, Operadoras),
     \+(memberchk(X, Visited)),
     query2(X, Finish, Operadoras, [X | Visited], Path)).

comunsQ2(Ini,Fim,Operadoras,Cam):-
     comunsLQ2(Ini,Fim,Operadoras,[C|_]),
     arco((C,A,Ini,OPF,_,_),_),
     memberchk(OPF, Operadoras),
     arco((C,B,Fim,OP,_,_),_),
     memberchk(OP, Operadoras),
     camQ2(Ini,Fim,C,A,B,Operadoras,[Ini],[_|Cam]).

camQ2(Gid, Gid, _, _, _, _, _, [Gid]).
camQ2(Start, Finish, C, A, B, Operadoras,Visited, [Start | Path]) :-
     arco((C,I,Start,OPF,_,_),_),
     memberchk(OPF, Operadoras),
     (B>A -> succ(I,T) ; sub(I,T)),
     arco((C,T,X,OP,_,_),_),
     memberchk(OP, Operadoras),
     \+(memberchk(X, Visited)),
     camQ2(X, Finish, C, A, B, Operadoras, [X | Visited], Path).

comunsLQ2(Ini,Fim,Operadoras,CarrCom):-
     findall(CI,(arco((CI,_,Ini,OPF,_,_),_),(memberchk(OPF, Operadoras))),CarrI),
     sort(CarrI,PI),
     findall(CF,(arco((CF,_,Fim,OP,_,_),_),(memberchk(OP, Operadoras))),CarrF),
     sort(CarrF,PF),
     intersection(PI,PF, CarrCom),
     \+length(CarrCom,0).
```

Figura 8: Querie2

2.4.2 Excluir um ou mais operadores de transporte para o percurso

```
%Q3Excluir um ou mais operadores de transporte para o percurso;
zeOperadoraExGlobal(Start, Finish, Visited, Operadoras, Path):-
    paragem(_,_,Finish,_,_,_,_,_,OPF,_,_,_),
    not(member(OPF, Operadoras)),
    zeOperadoraEx(Start, Finish, Visited, Operadoras, Path).

zeOperadoraEx(Gid, Gid, _, _,[Gid]).
zeOperadoraEx(Start, Finish, Visited, Operadoras, [(C,Start) | Path]) :-
    paragem(C,I,Start,_,_,_,_,_,OPS,_,_,_),
    not(member(OPS, Operadoras)),
    succ(I,T),
    paragem(C,T,X,_,_,_,_,_,OPN,_,_,_),
    not(member(OPN, Operadoras)),
    not(member(X, Visited)),
    zeOperadoraEx(X, Finish, [X | Visited], Operadoras, Path).

%Q3Excluir um ou mais operadores de transporte para o percurso; Versao com comuns.
query3G(Start, Finish, Operadoras, Path,N):-
    arco((_,_,Finish,OPF,_,_),(_,_,_,_,_,_,_,_)),
    \+(memberchk(OPF, Operadoras)),
    (isoladaG(Start,Finish),print("Carreira Isolada");
    query3(Start, Finish, Operadoras,[Start], Path),
    length(Path, N)).

query3(Gid, Gid,_, _, [Gid]).
query3(Gid, Gid,_, _, _).
query3(Start, Finish, Operadoras, Visited, [Start | Path]) :-
    (comunsQ3(Start,Finish,Operadoras,Path),
    query3(Finish, Finish, Operadoras, Visited, Path));
    (arco((_,_,Start,OPF,_,_),(_,_,X,_,_,_)),
    \+(memberchk(OPF, Operadoras)),
    \+(memberchk(X, Visited)),
    query3(X, Finish, Operadoras, [X | Visited], Path)).

comunsQ3(Ini,Fim,Operadoras,Cam):-
    comunsLQ3(Ini,Fim,Operadoras,[C|_]),
    arco((C,A,Ini,OPF,_,_),_),
    \+(memberchk(OPF, Operadoras)),
    arco((C,B,Fim,OP,_,_),_),
    \+(memberchk(OP, Operadoras)),
    camQ3(Ini,Fim,C,A,B,Operadoras,[Ini],[_|Cam]).
```

Figura 9: Querie3

```
camQ3(Gid, Gid, _, _, _, _, _, [Gid]).
camQ3(Start, Finish, C, A, B, Operadoras,Visited, [Start | Path]) :-
    arco((C,I,Start,OPF,_,_),_),
    \+(memberchk(OPF, Operadoras)),
    (B>A -> succ(I,T) ; sub(I,T)),
    arco((C,T,X,OP,_,_),_),
    \+(memberchk(OP, Operadoras)),
    \+(memberchk(X, Visited)),
    camQ3(X, Finish, C, A, B, Operadoras, [X | Visited], Path).

comunsLQ3(Ini,Fim,Operadoras,CarrCom):-
    findall(CI,(arco((CI,_,Ini,OPF,_,_),_),(\+(memberchk(OPF, Operadoras)))),CarrI),
    sort(CarrI,PI),
    findall(CF,(arco((CF,_,Fim,OP,_,_),_),(\+(memberchk(OP, Operadoras)))),CarrF),
    sort(CarrF,PF),
    intersection(PI,PF, CarrCom),
    \+length(CarrCom,0).
```

Figura 10: Querie3(Continuação)

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

```
%Q4Identificar quais as paragens com o maior número de carreiras num determinado percurso.

query4G(Path,C):-
    query4(Path,N),
    keysort(N,S),
    reverse(S, C).

query4([],[]).
query4([H|T], [P-H|F]):-
    auxQ4(H,P),
    query4(T,F).

maxi(H,P,A,T,X,F):-
    (P>=T,
    X is H,F is P);
    X is A,F is T.

auxQ4(H, N):-
    findall(CI,arco((CI,_,H,_,_,_),_),L),
    sort(L,G),
    length(G,N).
```

Figura 11: Querie4

2.4.4 Escolher o percurso que passe apenas por abrigos com publicidade

```
%Q8Escolher o percurso que passe apenas por abrigos com publicidade; Versao com comuns
query7G(Start, Finish, Publ, Path,N):-
    arco((_,_,Finish,_,_,Publ),(_,_,_,_,_,_)),
    (isoladaG(Start,Finish),print("Carreira Isolada");
    query7(Start, Finish, Publ, [Start], Path),
    length(Path, N)).

query7(Gid, Gid,_, _, [Gid]).
query7(Gid, Gid,_, _, _).
query7(Start, Finish, Publ, Visited, [Start | Path]) :-
    (comunsQ7(Start,Finish,Publ,Path),
    query7(Finish, Finish,Publ, Visited, Path));
    (arco((_,_,Start,_,_,Publ),(_,_,X,_,_,_)),
    \+(memberchk(X, Visited)),
    query7(X, Finish, Publ,[X | Visited], Path)).

comunsQ7(Ini,Fim,Publ,Cam):-
    comunsLQ7(Ini,Fim,Publ,[C|_]),
    arco((C,A,Ini,_,_,Publ),_),
    arco((C,B,Fim,_,_,Publ),_),
    camQ7(Ini,Fim,C,A,B,Publ,[Ini],[_|Cam]).

camQ7(Gid, Gid, _, _, _, _, _,[Gid]).
camQ7(Start, Finish, C, A, B, Publ, Visited, [Start | Path]) :-
    arco((C,I,Start,_,_,Publ),_),
    (B>A -> succ(I,T) ; sub(I,T)),
    arco((C,T,X,_,_,Publ),_),
    \+(memberchk(X, Visited)),
    camQ7(X, Finish, C, A, B, Publ, [X | Visited], Path).

comunsLQ7(Ini,Fim,Publ,CarrCom):-
    findall(CI,arco((CI,_,Ini,_,_,Publ),_),CarrI),
    sort(CarrI,PI),
    findall(CF,arco((CF,_,Fim,_,_,Publ),_),CarrF),
    sort(CarrF,PF),
    intersection(PI,PF, CarrCom),
    \+length(CarrCom,0).
```

Figura 12: Querie7

2.4.5 Escolher o percurso que passe apenas por paragens abrigadas

```
%Q8Escolher o percurso que passe apenas por paragens abrigadas; Versao comuns
query8G(Start, Finish, Path,N):-
    arco((_,_,Finish,_,AB,_),(_,_,_,_,_,_)),
    \+(memberchk(AB, ['Sem Abrigo'])),
    (isoladaG(Start,Finish),print("Carreira Isolada");
    query8(Start, Finish, ['Sem Abrigo'],[Start], Path),
    length(Path, N)).

query8(Gid, Gid,_, _, [Gid]).
query8(Gid, Gid,_, _, _).
query8(Start, Finish, Abri, Visited, [Start | Path]) :-
    (comunsQ8(Start,Finish,Abri,Path),
    query8(Finish, Finish, Abri, Visited, Path));
    (arco((_,_,Start,_,AB,_),(_,_,X,_,_,_)),
    \+(memberchk(AB, Abri)),
    \+(memberchk(X, Visited)),
    query8(X, Finish, Abri, [X | Visited], Path)).

comunsQ8(Ini,Fim,Abri,Cam):-
    comunsLQ8(Ini,Fim,Abri,[C|_]),
    arco((C,A,Ini,_,AB,_),_),
    \+(memberchk(AB, Abri)),
    arco((C,B,Fim,OP,_,_),_),
    \+(memberchk(OP, Abri)),
    camQ8(Ini,Fim,C,A,B,Abri,[Ini],[_|Cam]).

camQ8(Gid, Gid, _, _, _, _, _, [Gid]).
camQ8(Start, Finish, C, A, B, Abri,Visited, [Start | Path]) :-
    arco((C,I,Start,_,AB,_),_),
    \+(memberchk(AB, Abri)),
    (B>A -> succ(I,T) ; sub(I,T)),
    arco((C,T,X,OP,_,_),_),
    \+(memberchk(OP, Abri)),
    \+(memberchk(X, Visited)),
    camQ8(X, Finish, C, A, B, Abri, [X | Visited], Path).

comunsLQ8(Ini,Fim,Abri,CarrCom):-
    findall(CI,(arco((CI,_,Ini,_,AB,_),_),(\+(memberchk(AB, Abri)))),CarrI),
    sort(CarrI,PI),
    findall(CF,(arco((CF,_,Fim,OP,_,_),_),(\+(memberchk(OP, Abri)))),CarrF),
    sort(CarrF,PF),
    intersection(PI,PF, CarrCom),
    \+length(CarrCom,0).
```

Figura 13: Querie8

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

```
%Q9Escolher um ou mais pontos intermédios por onde o percurso deverá passar
query9G(Start,Finish,Inter,Path,N):-
     query9(Start,Finish,Inter,Path),
     length(Path,N).

query9(X,Finish,[],Path):-
     zefAdjQ9(X,Finish,Path).
query9(Start, Finish, [H|T], F):-
     zefAdjQ9(Start,H,P),
     query9(H,Finish,T,[_|Tail]),
     append(P, Tail, F).

zefAdjQ9(Start, Finish, Path):-
     arco((_,_,Finish,_,_,_),(_,_,_,_,_,_)),
     (isoladaG(Start,Finish),print("Carreira Isolada");
     zefa(Start, Finish, [Start], Path)).
```

Figura 14: Querie9

3 Conclusões e Melhorias

Começa-se por referir que, com muita pena, não foi possível implementar as queries 6 e 7, nomeadamente a do caminho mais curto e mais rápido, uma vez que os algoritmos inicialmente pensados e desenvolvidos não se revelaram suficientemente otimizados e orientados à resolução dessas tarefas isso, aliado à falta de tempo, não possibilitou a reformulação e adição de algoritmos de pesquisa melhores e orientados a esses casos e em geral a uma maior qualidade e eficiência do trabalho.

No entanto, o aluno revela-se relativamente satisfeito com a elaboração e conclusão dos algoritmos apresentados.