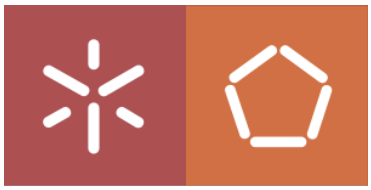

Universidade do Minho
Escola de Engenharia

Universidade do Minho
Mestrado Integrado em Engenharia Informática

SRCR

Sistemas de Representação de Conhecimento e Raciocínio
TP 2

Junho 2020

Tânia Rocha
(A85176)

1 Resumo

Este trabalho foi realizado no âmbito da Unicade Curricular de Sistemas de Representação de Conheciemnto e Raciocínio e consiste em **Métodos de Resolução de Problemas e de Procura** para a resolução da procura de caminhos ou percursos no contexto das paragens de autocarros do concelho de Oeiras, dados fornecidos pelos docentes.

O trabalho consiste na utilização de variados algoritmos de procura e a sua avaliação quanto à sua performance e caracteristicas.

Foi fornecido como base dois ficheiros de dados:

- Listas de adjacências das paragens de Autocarros. Este ficheiro foi interpretado como que em cada *sheet* a listagem das paragens significasse que estas estavam ligadas pela mesma ordem.
- Dataset: Paragens de Autocarro do concelho de Oeiras. Este ficheiro contém as paragens com o seu indice único e têm correspondente a lista de carreiras às quais pertencem.

Uma paragem é definida por 11 elementos. Estes são:

- ID - identificação de uma paragem
- Latitude
- Longitude
- Estado de Conservação
- Tipo de Abrigo - Paragem pode ser abrigada ou não;
- Abrigo Com Publicidade - Se a paragem tem publicidade ou não;
- Operadora - Cada paragem tem uma operadora que efetua o transporte;
- Carreira - Cada parahem pode pertencer a uma ou mais carreiras;
- Codigo de rua
- Nome de Rua
- Freguesia

Conteúdo

Lista de Figuras

2 Introdução

O trabalho prático descrito neste relatório consiste na utilização de algoritmos de Procura e resolução de problemas com principal objetivo obter diferentes caminhos conforme a formula definida.

A linguagem utilizada para implementar todo o exercício foi **prolog**, e para isso foi utilizado o programa **SICStus** como ferramenta.

Ao longo deste relatório vai ser descrito todo o processo de implementação, todas as conclusões e dificuldades.

3 Preliminares

Primeiramente, foi necessário transferir a informação fornecida pelos docentes para o ambiente onde seria implementado todo o exercício. Foi para isso, desenvolvido dois pequenos métodos em linguagem de programação *python*, mas mais facilitar o parcing dos ficheiros.

Este programa foi construído a partir do terminal *Anaconda* que permite utilizar o *Jupyter Lab*. A partir daí é possível usar *python* (pandas). O ficheiro em si que será enviado é apenas uma visualização não exacta.

Os dois Programas, criados de maneira distinta, tem como objectivo criar a Base de Conhecimento constituída por todas as paragens com um índice único, neste caso, o **GIC**, com as suas listas correspondentes de carreiras às quais pertencem, e criar um grafo com todas as ligações entre paragens, ou seja, todas as adjacências. A estas ligações foi dado o nome de **aresta(X,Y)**, com sentido de paragem X para paragem Y.

O primeiro método pode ser analisado de seguida:

```
[10]:
with open('ListaAdjacencias2.csv') as file:
        reader = csv.reader(file)
        prev = None
        value = None
        for row in reader:
            print("paragem("+row[0]+","+row[1]+","+row[2]+",'"+row[3]+"','"+row[4]+"','"+row[5]+"','"+row[6]+"',["+row[7]+"],"+row[8]+",'"+row[9]+"','"+row[10]+"').")
```

Figura 1: Método em Python para gerar a base de conhecimento

Como podemos observar, este método transcreve para string cada linha que lê do ficheiro cvs e imprime a mesma. Encontra-se de seguida uma amostra do output:

```
paragem(79,-107011.55,-95214.57,'Bom','Fechado dos Lados','Yes','Vimeca',[01],103,'Rua Damiao de Gois','Alges, Linda-a-Velha e Cruz Quebrada-Dafundo').
paragem(593,-103777.02,-94637.67,'Bom','Sem Abrigo','No','Vimeca',[01],300,'Avenida dos Cavaleiros','Carnaxide e Queijas').
paragem(499,-103758.44,-94393.36,'Bom','Fechado dos Lados','Yes','Vimeca',[01],300,'Avenida dos Cavaleiros','Carnaxide e Queijas').
paragem(494,-106803.2,-96265.84,'Bom','Sem Abrigo','No','Vimeca',[01],389,'Rua Sao Joao de Deus','Alges, Linda-a-Velha e Cruz Quebrada-Dafundo').
paragem(480,-106757.3,-96240.22,'Bom','Sem Abrigo','No','Vimeca',[01],389,'Rua Sao Joao de Deus','Alges, Linda-a-Velha e Cruz Quebrada-Dafundo').
paragem(957,-106911.18264993647,-96261.15727273725,'Bom','Sem Abrigo','No','Vimeca',[01],399,'Escadinhas da Fonte da Maruja','Alges, Linda-a-Velha e Cruz Quebrada-Dafundo').
paragem(366,-106021.37,-96684.5,'Bom','Fechado dos Lados','Yes','Vimeca',[01],411,'Avenida Dom Pedro V','Alges, Linda-a-Velha e Cruz Quebrada-Dafundo').
paragem(365,-106016.12,-96673.87,'Bom','Fechado dos Lados','Yes','Vimeca',[01],411,'Avenida Dom Pedro V','Alges, Linda-a-Velha e Cruz Quebrada-Dafundo').
paragem(357,-105236.99,-96664.4,'Bom','Fechado dos Lados','Yes','Vimeca',[01],1279,'Avenida Tomas Ribeiro','Alges, Linda-a-Velha e Cruz Quebrada-Dafundo').
```

Figura 2: Amostra do output

No caso do segundo método, a sua implementação foi realizada de maneira a gerar um grafo semelhante ao estudado nas aulas práticas. O método encontra-se de seguida:

A sua implementação foi pensada da mesma forma que o programa anterior, ou seja, impressão de uma string com toda a informação lida, linha a linha do ficheiro com as listas de adjacência. Podemos também observar que foi tido em conta o caso de duas linhas estarem seguidas mas não serem de facto uma ligação entre as paragens. Estes casos são as separações das *sheets* em que não foi considerado que a última paragem de uma *sheet* é de facto adjacente à primeira da *sheet* seguinte.

```python
zz = "grafo(["
final = "["
with open("ListaAdjacencias.csv") as file:
        reader = csv.reader(file)
        prev = None
        value = None
        for row in reader:
            if prev is not None and prev == row[7]:
                final = final +"aresta("+value+","+row[0]+"),"
            zz = zz + row[0]+","
            prev = row[7]
            value = row[0]
zz = zz+"]"
final = "g("+zz+","+final+"]))."

print(final)
```

Figura 3: Método em Python para gerar grafo

```
g2(grafo([183,791,595,182,499,593,
          ,[aresta(183,791)
          ,aresta(791,595)
          ,aresta(595,182)
          ,aresta(182,499)
          ,aresta(499,593)
          ,aresta(593,181)
          ,aresta(181,180)
          ,aresta(180,594)
          ,aresta(594,185)
          ,aresta(185,89)
          ,aresta(89,107)
          ,aresta(107,250)
          ,aresta(250,261)
          ,aresta(261,597)
          ,aresta(597,953)
```

Figura 4: Amostra de output do segundo método. Nota: a imagem continuava para a direita e para baixo.

3.1 Métodos de Resolução de Problemas e de Procura

3.1.1 Definição

A resolução de problemas e procura consiste no uso de métodos, de uma forma ordenada, para encontrar soluções de problemas específicos. Este mesmo termo é usado em muitas disciplinas e áreas do conhecimento , às vezes com diferentes perspectivas e geralmente com terminologias diferentes.

Neste contexto é considerado a resolução de problemas como um problema de pesquisa.

4 Descrição do Trabalho e Análise de Resultados

4.1 Contextualização

O projecto foi desenvolvido com três ficheiros distintos após a obtenção de dados antes descrita. Este ficheiros são:

- **grafo.pl** - Ficheiro que tem o grafo armazenado.
- **BaseC.pl** - Ficheiro que armazena todas as paragens em si, com todas as suas características. Temos ainda neste mesmo tópico a definição da paragem:

```
:- dynamic(paragem/11).
```

Figura 5: Definição

- **trab.pl** - Ficheiro com todas as alíneas e respectivos algoritmos definidos.

Tal como pedido no exercício, foram desenvolvidos vários algoritmos em prolog de modo a resolver as diferentes alíneas constituintes.

Ao longo do trabalho foram utilizados vários algoritmos distintos, nomeadamente:

- **Depth-First** - Na teoria dos grafos, procura em profundidade é um algoritmo usado para realizar uma procura ou travessia numa árvore, estrutura de árvore ou grafo. Este algoritmo começa num nodo, neste caso arbítrio, e explora tanto quanto possível cada um dos seus ramos, antes de retroceder(backtracking). Neste exercício, este algoritmo realiza uma procura não-informada.
- **Algoritmo guloso ou *Greedy*** - é uma técnica de algoritmos para resolver problemas de optimização, realizando sempre a escolha que parece ser a melhor no momento; fazendo uma escolha óptima local, na esperança de que esta escolha leve até a solução ótima global. Neste contexto este algoritmo toma como variável para tomar decisões a distância de um nodo ao destino desejado.
- **Algoritmo A* (estrela)** - é um algoritmo para procura de Caminho. Ele procura o caminho em um grafo de um vértice inicial até um vértice final e é a combinação de aproximações heurísticas semelhante ao algoritmo Breadth First (Procura em Largura) e da formalidade do Algoritmo de Dijkstra. Neste contexto, este algoritmo tem como heurística a escolha da menor soma entre o nodo cuja distancia do ponto onde estamos e a distancia ao destino.

4.2 Alínea 1 - Calcular um trajeto entre dois pontos

Nesta alínea podemos dizer que foram utilizados todos os algoritmos, ainda que tanto o algoritmo guloso como A* tenham sido usados como resolução das alíneas de procura por distancia, visto que estes tem em conta o peso, ou seja, a distancia.

Foi desenvolvido primeiro um algoritmo simples, implementado numa das aulas, que apenas procura um caminho entre dois nodos através da visita a todos os nodos. Este método é muito ineficiente, daí a não ser utilizado futuramente.

```
caminho(G,A,B,P) :- caminho1(G,A,[B],P).

caminho1(_,A,[A|P1],[A|P1]).
caminho1(G,A,[Y|P1],P) :-
   adjacente(X,Y,G), \+ memberchk(X,[Y|P1]), caminho1(G,A,[X,Y|P1],P).
```

Figura 6: Procura um caminho

4.2.1 Depth First

Partimos então para o desenvolvimento de um algoritmo **Depth-First** de procura de caminhos entre dois nodos. Este algoritmo é muito mais eficiente do que o método anterior, e procura eficientemente um caminho entre dois nodos, no entanto há determinados conjuntos de procura que expandem a procura a um nível muito elevado e o programa não encontra o caminho.

```
dFirst(G,X,Y,R) :- dFirst(G,X,Y,[X],R).

dFirst(G,X,Y,V,[aresta(X,Y)]) :- adjacente(X,Y,G).

dFirst(G,X,Y,V,[aresta(X,Z)|R]) :-
 adjacente(X,Z,G),
 \+ memberchk(aresta(X,Z),V),
 \+ member(Z,V),
 dFirst(G,Z,Y,[Z|V],R),
 Z \= Y.
```

Figura 7: Procura um caminho por Depth-First

Apesar de existirem extensões a este algoritmo para tornar a sua procura mais optimizada, não foram implementados neste exercício.

Como este algoritmo encontra uma grande parte de caminhos existentes, será utilizado para resolver as alíneas seguintes.

4.2.2 Greedy

A partir da interpretação da informação leccionada na UC, o algoritmo **Greedy** ou Guloso, foi implementado tendo em conta a heurística da distância entre os nodos adjacentes ao actual com o nodo final, escolhendo o que tem menor distância, visto que isso implica que ao estar mais próximo do destino vai implicar um menor conjunto de paragens para o caminho final.

Para isto foi desenvolvido um método que a partir de uma aresta devolve uma lista de todas as arestas adjacentes a esta mesma.

```
%sucessores a uma aresta
sucessor(grafo(_,Es),Z,R):- findall(aresta(Z,I),member(aresta(Z,I),Es),R).
```

Figura 8: Método que devolve arestas adjacentes

Tendo isto, o que este algoritmo faz é calcular estas adjacências e depois ordenar os mesmos numa lista de distância crescentes ao destino. Daí escolhe o primeiro elemento dessa lista, ou seja, o elemento mais próximo do destino. Este método é denominado de **sortSuccessors**. O algoritmo implementado apresenta-se de seguida:

```
greedy(G,X,Y,Soln) :-
        sucessor(G,X,PP),
        sortSuccessors(Y,PP,PriorityList),
        greedy1(G,X,Y,PriorityList,          % PRIORITY
        [],                                  % NODES VISITED
        Soln).                               % SOLUTION

greedy1(G,X,Y,[], Visited, []).              % NO SOLUTION FOUND

greedy1(G,X,Y,[aresta(X,Y)|OtherSolns], Visited, Soln):-
                                    append(Visited,[aresta(X,Y)],Soln).

greedy1(G,X,Y,[aresta(Z,W)|OtherSolns], Visited, Soln) :-
    sucessor2(G,W,Visited,Successors),
    expande_greedy(G,Y,Successors,[],Cams), %retirar nodos que não tem adjacencia(n Pretendidos)
    sortSuccessors(Y,Cams, PriorityQueue), !,
    greedy1(G,W,Y,PriorityQueue, [aresta(Z,W)|Visited], Soln).
```

Figura 9: Algoritmo greedy

Após testes efectuados com este algoritmo, foi reparado que este algoritmo ao escolher as arestas deste modo, pode escolher vértices que sejam próximos do destino mas que de facto não tenham depois mais adjacências, isto é, não chegam de facto ao destino. Foi então para isso desenvolvido um método, ***expande-greedy*** que a todas as listas de sucessores calculados retira desta mesma todos os vértices que não tenham adjacências a não ser que estes sejam de facto os vértices pretendidos.

No entanto, esta "expansão"não é suficiente, por isso existem de facto caminhos que este algoritmo não consegue encontrar. Ainda assim, será utilizado nas próximas alíneas.

4.2.3 A* (Estrela)

Tal como o algoritmo anterior, o algoritmo **estrela** devolve as suas soluções baseadas em heurísticas. A Heurística utilizada neste caso é um dos leccionados, em que a partir de um nodo, é calculado todos os seus adjacentes, da mesma forma que no algoritmo greedy, e ordenado numa lista de prioridade da qual será escolhido o primeiro. Esta *PriorityList* é calculada a partir do método **sortSuccDist**, que ordena as arestas decrescentemente da soma entre a distancia a cada um desses vértices e a distância destes ao nodo destino. O algoritmo pode ser observado de seguida:

```
estrela(G,X,Y,Soln) :-
        sucessor(G,X,PP),
        sortSuccDist(Y,PP,PriorityList),
        star1(G,X,Y,PriorityList,          % PRIORITY
        [],                                % NODES VISITED
        Soln).                             % SOLUTION

star1(G,X,Y,[], Visited, []).              % NO SOLUTION FOUND

star1(G,X,Y,[aresta(X,Y)|OtherSolns], Visited, Soln):-
                    append(Visited,[aresta(X,Y)],Soln).


star1(G,X,Y,[aresta(Z,W)|OtherSolns], Visited, Soln) :-
    sucessor2(G,W,Visited,Successors),
    expande_greedy(G,Y,Successors,[],Cams),
    sortSuccDist(Y,Cams, PriorityQueue), !,
    star1(G,W,Y,PriorityQueue, [aresta(Z,W)|Visited], Soln).
```

Figura 10: Algoritmo Estrela

Tal como o algoritmo greedy, de maneira a tentar combater caminhos sem fim foi utilizado o mesmo método ***expande-greedy***, que elimina da lista de prioridade todos os vértices sem adjacências, no entanto tal como anteriormente, esta "expansão"não é suficiente para todos os caminhos. Este algoritmo também vai ser utilizado nas alíneas seguintes.

4.3 Alínea 2 - Seleccionar apenas algumas operadoras de transporte para um determinado percurso

Para esta alínea é fornecido uma partida, um destino e uma lista de operadoras que se quer que seja composto o percurso. Para isto são utilizados os 3 algoritmos apresentados com a condição acrescida de que apenas aceita uma determinada aresta seja percorrida por uma determinada operadora. O método seguinte devolve a operadora de uma determinada paragem:

```
auxAl(aresta(X,Y),R):- paragem(X,_,_,_,_,_,OP,_,_,_,_),R = OP.
```

Figura 11: Devolve operadora a partir de uma aresta

4.3.1 Depth First

Esta condição é facilmente aplicada no algoritmo **Depth First**. É apenas necessário adiciona-la com condição no caso de paragem e de cada vez que se pretende aceitar uma aresta, ou seja, apenas é escolhido aquela aresta se esta for percorrida por uma das transportadoras desejadas:

```
dFOperadoras(G,X,Y,ListaOperadoras,R) :- dFOperadoras(G,X,Y,ListaOperadoras,[X],R).

dFOperadoras(G,X,Y,ListaOperadoras,V,[aresta(X,Y)]) :- adjacente(X,Y,G),
                                                       auxAl(aresta(X,Y),R),
                                                       memberchk(R,ListaOperadoras).

dFOperadoras(G,X,Y,ListaOperadoras,V,[aresta(X,Z)|R]) :-
 adjacente(X,Z,G),
 auxAl(aresta(X,Z),RR),
 memberchk(RR,ListaOperadoras),
 \+ memberchk(aresta(X,Z),V),
 \+ member(Z,V),
 dFOperadoras(G,Z,Y,ListaOperadoras,[Z|V],R),
 Z \= Y.
```

Figura 12: Depth first por determinadas transportadoras

4.3.2 Greedy

No caso do **Greedy**, esta condição é aplicada à lista de arestas adjacentes para a lista de prioridade, ou seja, remove todas as arestas que não sejam de de paragens com transporte efectuado pelas operadoras desejadas. Este processo é efectuado pelo método ***remove*** chamado numa nova implementação do algoritmo greedy apresentado de seguida:

```
grTrans(G,X,Y,ListaOperadoras,Soln) :-
            sucessor(G,X,PP),
            sortSuccessors(Y,PP,PriorityList),   % PRIORITY
            retira(PriorityList,[],ListaOperadoras,Res), % retira da lista arestas que não sao feito com as dadas operadoras
            greedy2(G,X,Y,ListaOperadoras, Res,
            [],                                  % NODES VISITED
            Soln).                               % SOLUTION

greedy2(G,X,Y,ListaOperadoras,[], Visited, []).                % NO SOLUTION FOUND

greedy2(G,X,Y,ListaOperadoras,[aresta(X,Y)|OtherSolns], Visited, Soln):-
                                    append(Visited,[aresta(X,Y)],Soln).


greedy2(G,X,Y,ListaOperadoras,[aresta(Z,W)|OtherSolns], Visited, Soln) :-
    sucessor2(G,W,Visited,Successors),
    expande_greedy(G,Y,Successors,[],Cams), %retirar nodos que não tem adjacencia se estes não forem os pretendidos
    sortSuccessors(Y,Cams, PriorityQueue),
    retira(PriorityQueue,[],ListaOperadoras,Res), !,
    greedy2(G,W,Y,ListaOperadoras,Res, [aresta(Z,W)|Visited], Soln).
```

Figura 13: Greedy por determinadas transportadoras

Este método encontra-se explícito nos anexos.

4.3.3 A* (Estrela)

Tal como no greedy, este algoritmo é aplicado da mesma maneira, isto é, acrescentando-lhe a condução das operadoras nas listas de Prioridade. Este novo algoritmo está presente nos anexos.

4.4 Alínea 3 - Excluir um ou mais operadores para um determinado percurso

Devido à semalhança entre esta alínea e a anterior, o método de resolução foi o mesmo, no entanto a condição de aceitar arestas ou não depende se estas são ou não de uma operadora ou não. Neste caso, todas as aresta de uma ou mais transportadoras são retiradas das listas de prioridades.

Todos os métodos utilizados encontram-se nos anexos.

4.5 Alínea 4 - Identificar quais as paragens com o maior número de carreiras num determindado percurso

Para esta alínea tiveram de ter implemnetadas vários métodos distintos para depois serem aplicados em funções finais de resolução.

Primeiramente,foi desenvolvido um método que dado uma lista e um número, retorna os **n** primeiros elementos dessa mesma lista ou , no caso da lista ser menor que **n** devolve a lista completa. Este método foi feito para depois aplicar à função final que retorna todo o *path* a fim de este retornar apenas os primeiros n elementos.

```
grTrans(G,X,Y,ListaOperadoras,Soln) :-
        sucessor(G,X,PP),
        sortSuccessors(Y,PP,PriorityList),   % PRIORITY
        retira(PriorityList,[],ListaOperadoras,Res), % retira da lista arestas que não sao feito com as dadas operadoras
        greedy2(G,X,Y,ListaOperadoras, Res,
        [],                                  % NODES VISITED
        Soln).                               % SOLUTION

greedy2(G,X,Y,ListaOperadoras,[], Visited, []).                % NO SOLUTION FOUND

greedy2(G,X,Y,ListaOperadoras,[aresta(X,Y)|OtherSolns], Visited, Soln):-
                                    append(Visited,[aresta(X,Y)],Soln).


greedy2(G,X,Y,ListaOperadoras,[aresta(Z,W)|OtherSolns], Visited, Soln) :-
    sucessor2(G,W,Visited,Successors),
    expande_greedy(G,Y,Successors,[],Cams), %retirar nodos que não tem adjacencia se estes não forem os pretendidos
    sortSuccessors(Y,Cams, PriorityQueue),
    retira(PriorityQueue,[],ListaOperadoras,Res), !,
    greedy2(G,W,Y,ListaOperadoras,Res, [aresta(Z,W)|Visited], Soln).
```

Figura 14: Greedy por determinadas transportadoras

Para retornar os primeiros n elementos na ordem desejada, isto é, as paragens com maior número de carreiras, é necessário ordenar estas mesmas paragens. Para isso, através do método **pp** que retorna uma paragem dada uma aresta, é ordenada a path obtida entre dois nodos.

```
%devolve paragem de uma aresta
pp(aresta(X,_),R):- solucoes(paragem(X,B,A,C,V,L,O,U,G,E,W),paragem(X,B,A,C,V,L,O,U,G,E,W),R).

%devolve paragens de arestas
getParagens(P,R):-getParagens2(P,[],R).
getParagens2([],R,R).
getParagens2([X|XS],Li,R):-append(Li,TTT,ZZ),pp(X,TTT),
                    getParagens2(XS,ZZ,R).
```

Figura 15: Método que de uma lista de arestas retorna paragens respectivas

Após estas pequenas construções, foram desenvolvidas os métodos finais para os três algoritmos, ou seja, para cada *path* obtido dados dois vértices, é calculada a lista ordenada das paragens com maior número de carreiras e logo depois é devolvido os primeiros n elementos.

- **Depth First**

```
mNCP(G,X,Y,N,R):- dFirst(G,X,Y,P),getParagens(P,L),
                quick_sort2(L,RR),nPrim(N,RR,R).
```

Figura 16: Depth First alínea 4

- **Greedy**

```
mNCP2(G,X,Y,N,R):- greedy(G,X,Y,P),getParagens(P,L),
                quick_sort2(L,RR),nPrim(N,RR,R).
```

Figura 17: Greedy alínea 4

- **A* (Estrela)**

```
mNCP3(G,X,Y,N,R):- estrela(G,X,Y,P),getParagens(P,L),
                   quick_sort2(L,RR),nPrim(N,RR,R).
```

Figura 18: Estrela alínea 4

4.6 Alínea 5 - Escolher o menor percurso

Tal como introduzido no inicio, esta alínea foi "respondida"através do metodo de desenvolvimento do algoritmo **greedy**, visto que este calcula os caminho por menor distância ao destino, o que faz com que sejam calculadas menos paragens.

No entanto, tal como analisado anteriormente este algoritmo tem a desvantagem de nem sempre encontrar o caminho. Podemos considerar que qualquer path devolvida pelo algoritmo greedy é já o menor curso possível usando o critério de menor número de paragens.

Para além disto, esta alínea for desenvolvida de maneira a ser resolvível pelos distintos algoritmos. Esta resolução basea-se na procura de todos os caminhos possíveis entre dois pontos e daí devolver o caminho com menos paragens, isto é, com menos aresta. Para auxilio da obtenção do melhor percurso, para cada caminho encontrado é devolvido um par **(path , Numero de arestas)**.

Um dos métodos auxiliares desenvolvidos tem como objectivo escolher da lista obtida de pares, que tem o menor segundo elemento, o percurso respectivo, ou seja o menor path possível. O método auxiliar está presente nos anexos.

4.6.1 Depth First

Implementando os métodos descritos anteriormente optemos a seguinte função para encontrar o menor caminho entre dois nodos com o algoritmo **Depth-First**:

```
menorPercursoDF(G,X,Y,R) :- findall((P,Q),(dFirst(G,X,Y,P),
                                length(P,Q), Q>0),Tamanhos),
                            devolveP(Tamanhos,B,300000,R).
```

Figura 19: Depth First alínea 4

No entanto, ente algoritmo não é eficiente o suficiente para fazer a procura devido à dimensão do grafo, por isso não consegue devolver qualquer caminho quando é pedido todos os paths possíveis.

4.6.2 Greedy e A*(Estrela)

Para estes algoritmos é feita a implementação desta alínea da mesma maneira que em Depth First. (ver em anexo)

Para ambos estes algoritmos, a procura de todos os paths é de facto efetuada o que indica que estes algoritmos são mais eficientes que o anterior. No entanto, devido à desvantagem de não encontrarem alguns paths como explicado anteriormente é concluído que a solução destes métodos podem não ser os correctos.

```
menorPercursoGreedy(G,X,Y,R) :- findall((P,Q),(greedy(G,X,Y,P),
                                length(P,Q), Q>0),Tamanhos),
                                devolveP(Tamanhos,B,300000,R).
menorPercursoEstrela(G,X,Y,R) :- findall((P,Q),(estrela(G,X,Y,P),
                                length(P,Q), Q>0),Tamanhos),
                                devolveP(Tamanhos,B,300000,R).
```

Figura 20: Greedy e A* respectivamente, alínea 5

4.7 Alínea 6 - Escolher o pecurso mais rápido

Seguindo a linha de raciocínio da alínea anterior, nesta alínea pode ser dada como resposta qualquer *path* obtido a partir do algoritmo **A* (Estrela)**, visto que este tem como heurística a soma da distancia final e a distancia entre os nodos adjacentes. Assim, este algoritmo devolverá sempre o path de menor distância.

Uma outra forma de resolver é através do mesmo raciocínio que a alínea anterior, ou seja, efectuando a pesquisa de todos os paths possíveis e devolver o de menor distância associada. Para isso foi construído método **calcDist** que calcula a distância percorrida num path.

Tal como visto anteriormente, o algoritmo **Depth First** não é eficiente o suficiente para efetuar esta alínea, no entanto tanto o **greedy** como o **A* (Estrela)** conseguem.

```
resolveDistDF(G,X,Y,P):- findall((P,Res),(dFirst(G,X,Y,P),
                          length(P,O),O>0,
                          calcDist(P,0,Res)),Tamanhos),
                          devolveP(Tamanhos,B,100000000000,P).

resolveDistG(G,X,Y,P):- findall((P,Res),(greedy(G,X,Y,P),
                          length(P,O),O>0,
                          calcDist(P,0,Res)),Tamanhos),
                          devolveP(Tamanhos,B,100000000000,P).

resolveDistE(G,X,Y,P):- findall((P,Q),(estrela(G,X,Y,P),
                          length(P,LL),LL>0,
                          calcDist(P,0,Q)),Tamanhos),
                          devolveP(Tamanhos,B,100000000000,P).
```

Figura 21: Depth First, Greedy e A* respetivamente, alínea 6

4.8 Alínea 7 - Escolher o percurso que passe apenas por abrigos com publicidade

Nesta alínea, o raciocinio foi o mesmo que nas alíneas de escolher caminhos com determinadas Operadoras. Desto modo foram simplesmente adicionados aos algoritmos a condição de uma paragem ter que ter publicidade. No caso da **Depth First** apenas adicionar o caso se cada aresta é composta por paragens com publicidade e no caso dos algoritmos **greedy** e **A* (Estrela)** retirar da lista prioritária todas as arestas que não tem publicadade. O método Utlizado é muito semelhante ao método **retira** utilizado anteriormente:

```
%metodo que diz que passa por abrigo com publicidade(yes) ou nao (no)
temPub(paragem(A,B,C,V,G,'Yes',D,O,S,Y,P)).

%auxiliar retira adjacencias sem publicidade

retiraComP([A],V,Res):-pp(A,P),head(P,PP),temPub(PP),append(V,[A],Res).
retiraComP([A],V,V):-pp(A,P),head(P,PP),nao(temPub(PP)).
retiraComP([A|S],V,Res):-pp(A,P),head(P,PP),nao(temPub(PP)), retiraComP(S,V,Res).
retiraComP([A|S],V,Res):-pp(A,P),head(P,PP),temPub(PP), append(V,[A],LL),
                         retiraComP(S,LL,Res).
```

Figura 22: Retira arestas sem publicidade, alínea 7

4.9 Alínea 8 - Escolher o percurso que passe apenas por paragens abrigadas

Esta alínea foi desenvolvida semelhantemente à alínea anterior, a unica diferença é que os elementos a ser testado são a paragem ter abrigo ou não. Neste casp considerou-se todas as paragens que não incluíssem "Sem Abrigo"como abrigadas:

```
%metodo que diz se é abrigado ou nao
temAbrigo(paragem(A,B,C,V,J,F,D,O,S,Y,P)):- nao(J = 'Sem Abrigo').

%auxiliar retira adjacencias sem abrigo

retiraSAb([A],V,Res):-pp(A,P),head(P,PP),temAbrigo(PP),
                      append(V,[A],Res).
retiraSAb([A],V,V):-pp(A,P),head(P,PP),nao(temAbrigo(PP)).
retiraSAb([A|S],V,Res):-pp(A,P),head(P,PP),nao(temAbrigo(PP)), retiraSAb(S,V,Res).
retiraSAb([A|S],V,Res):-pp(A,P),head(P,PP),temAbrigo(PP), append(V,[A],LL),
                        retiraSAb(S,LL,Res).
```

Figura 23: Retira arestas com abrigo, alínea 8

4.10 Alínea 9 - Escolher o menor percurso

Por fim, o objetivo desta alínea foi atinjido através da concatenação dos paths de ordem desejada, por exemplo, o caminho começado em 34 que passe por 45 e 67, por esta ordem, seria uma concatenação do path entre 34 e 45 e do path entre 45 e 67.

Este método pode ser aplicado de igual forma nos três algoritmos estudados. Observe-se de seguida através de **Depth First**, os outros são de aplicação semelhante:

```
dFIntermedios(G,X,L,R) :- dFIntermedios2(G,X,L,[],R).

dFIntermedios2(G,X,[L],V,R) :- dFirst(G,X,L,P),concat(V,P,R).
dFIntermedios2(G,X,[L|LS],V,R):-
    dFirst(G,X,L,P),
    concat(V,P,Z),
    dFIntermedios2(G,L,LS,Z,R).
```

Figura 24: Concatenação de paths Depth First, alínea 9

4.11 Visão geral

Após a resolução destas alíneas, podemos observar a utlidade e diversidade de aplicação destes algoritmos num mesmo contexto, o que realça a sua enorme aplicação em areas distintas. Podemos também observar que estes algoritmos têm diferentes performances e tempos de execução. Para uma melhor visualização:

Algoritmo	Completo	Otimal	Tempo complexidade	Espaço complexidade
Primeiro em Profundidade (DFS)	No	Não	$O(b^m)$	O(bm)
Greedy	No	Não	Worst case: $O(b^m)$ Best case: O(bd)	
A*	Yes	Yes (if heuristic is admissible)	Number of nodes with $g(n)+h(n) \leq C^*$	

Figura 25: Algoritmos

5 Conclusão

A elaboração deste projeto funcionou como uma forma de consolidação da matéria lecionada ao longo das aulas desta Unidade Curricular, mais concretamente sobre o tema de Métodos de resolução de problemas e procura. Assim, seria esperada a obtenção dos melhor caminhos entre dois ou mais nodos através da utilização de varios algoritmos.

Posto isto, neste exercício foi necessário tratar de dados fornecidos pelo docente efetuando o *parsing* dos mesmos de forma a serem utilizados como um grafo e como uma base de conhecimento de paragens de autocarros, nomeadamente do concelho de Oeiras.

Pode ser concluido que este é um tema de muita utilidade e aplicação em areas distintas que permite a resolução de problemas de procura complexos.

6 Anexos

6.1 alínea 1

```
%Devolve lista de arestas que não são "becos" sem saida

expande_greedy(G,FIM,[],V,V).
expande_greedy(G,FIM,[aresta(X,FIM)],V,ExpCaminhos):- append(V,[aresta(X,FIM)],ExpCaminhos).
expande_greedy(G,FIM,[aresta(X,Y)],V,V):- sucessor(G,Y,P),length(P,L),L=<0.
expande_greedy(G,FIM,[aresta(X,Y)],V,ExpCaminhos):- sucessor(G,Y,P),length(P,L),L>0,
                                                    append(V,[aresta(X,Y)],ExpCaminhos).
expande_greedy(G,FIM,[aresta(X,Y)|Caminhos],V,ExpCaminhos):- sucessor(G,Y,P),length(P,L),L=<0, nao(Y=FIM),
                                                    expande_greedy(G,FIM,Caminhos,V,ExpCaminhos).
expande_greedy(G,FIM,[aresta(X,Y)|Caminhos],V,ExpCaminhos):- sucessor(G,Y,P),length(P,L),L>0,
                                                    append(V,[aresta(X,Y)],Next),
                                                    expande_greedy(G,FIM,Caminhos,Next,ExpCaminhos).
expande_greedy(G,FIM,[aresta(X,FIM)|Caminhos],V,ExpCaminhos):-append(V,[aresta(X,FIM)],Next),
                                                    expande_greedy(G,FIM,Caminhos,Next,ExpCaminhos).
```

Figura 26: método expande-greedy

```
%ordena lista de paragens pela sua distancia ao Fim
sortSuccessors(Y,List,Sorted):-q_sort2(Y,List,[],Sorted).
q_sort2(Y,[],Acc,Acc).
q_sort2(Y,[H|T],Acc,Sorted):-
    pivoting2(Y,H,T,L1,L2),
    q_sort2(Y,L1,Acc,Sorted1),q_sort2(Y,L2,[H|Sorted1],Sorted).


pivoting2(Y,H,[],[],[]).
pivoting2(Y,H,[X|T],[X|L],G):-pp(aresta(Y,_),YY), devEl(YY,YYY),
                              pp(H,P), devEl(P,PP),
                              pp(X,P2), devEl(P2,P22),
                              distEntreParagens(PP,YYY,XX),
                              distEntreParagens(P22,YYY,HH),
                              XX=<HH,
                              pivoting2(Y,H,T,L,G).

pivoting2(Y,H,[X|T],L,[X|G]):-pp(aresta(Y,_),YY), devEl(YY,YYY),
                              pp(H,P), devEl(P,PP),
                              pp(X,P2), devEl(P2,P22),
                              distEntreParagens(PP,YYY,XX),
                              distEntreParagens(P22,YYY,HH),
                              XX>HH,
                              pivoting2(Y,H,T,L,G).
```

Figura 27: sortSuccessors (greedy)

```
%ordena lista de paragens pela sua distancia ao Fim + A sua distancia ao nodo seguinte
sortSuccDist(Y,List,Sorted):-q_sort3(Y,List,[],Sorted).
q_sort3(Y,[],Acc,Acc).
q_sort3(Y,[H|T],Acc,Sorted):-
    pivoting3(Y,H,T,L1,L2),
    q_sort3(Y,L1,Acc,Sorted1),q_sort3(Y,L2,[H|Sorted1],Sorted).

pivoting3(Y,H,[],[],[]).
pivoting3(Y,H,[X|T],[X|L],G):-pp(aresta(Y,_),YY), devEl(YY,YYY),
                              pp(H,P), devEl(P,PP),
                              pp(X,P2), devEl(P2,P22),
                              distEntreParagens(PP,YYY,XX),
                              distEntreParagens(P22,YYY,HH),
                              distEntreParagens(PP,P22,Dist),
                              XX+Dist=<HH+Dist,
                              pivoting3(Y,H,T,L,G).

pivoting3(Y,H,[X|T],L,[X|G]):-pp(aresta(Y,_),YY), devEl(YY,YYY),
                              pp(H,P), devEl(P,PP),
                              pp(X,P2), devEl(P2,P22),
                              distEntreParagens(PP,YYY,XX),
                              distEntreParagens(P22,YYY,HH),
                              distEntreParagens(PP,P22,Dist),
                              XX+Dist>HH+Dist,
                              pivoting3(Y,H,T,L,G).
```

Figura 28: sortSuccDist (A*)

6.2 alínea 2

```
%função auxiliar que devolve arestas de paragens desejadas

retira([A],V,L,Res):-auxAl(A,P), member(P,L),
                        append(V,[A],Res).
retira([A],V,L,V):-auxAl(A,P), nao(member(P,L)).
retira([A|S],V,L,Res):-auxAl(A,P), member(P,L), append(V,[A],LL), retira(S,LL,L,Res).
retira([A|S],V,L,Res):-auxAl(A,P),
                       nao(member(P,L)),
                       retira(S,V,L,Res).
```

Figura 29: retira - devolve arestas com transportadoras desejadas

```
%através de A*

estOpe(G,X,Y,ListaOperadoras,Soln) :-
            sucessor(G,X,PP),
            sortSuccDist(Y,PP,PriorityList),   % PRIORITY
            retira(PriorityList,[],ListaOperadoras,Res), %retira arestas de nao Operadoras
            star2(G,X,Y,ListaOperadoras,Res,
            [],                                % NODES VISITED
            Soln).                             % SOLUTION

star2(G,X,Y,ListaOperadoras,[], Visited, []).    % NO SOLUTION FOUND

star2(G,X,Y,ListaOperadoras,[aresta(X,Y)|OtherSolns], Visited, Soln):-
                                    append(Visited,[aresta(X,Y)],Soln).


star2(G,X,Y,ListaOperadoras,[aresta(Z,W)|OtherSolns], Visited, Soln) :-
    sucessor2(G,W,Visited,Successors),
    expande_greedy(G,Y,Successors,[],Cams),
    sortSuccDist(Y,Cams, PriorityQueue),
    retira(PriorityQueue,[],ListaOperadoras,Res), !,
    star2(G,W,Y,ListaOperadoras,Res, [aresta(Z,W)|Visited], Soln).
```

Figura 30: Estrela alínea 2

6.3 Alínea 3

```
%através da dFirst
dFNop(G,X,Y,ListaOperadoras,R) :- dFNop(G,X,Y,ListaOperadoras,[X],R).

dFNop(G,X,Y,ListaOperadoras,V,[aresta(X,Y)]) :-
                    adjacente(X,Y,G),
                    auxAl(aresta(X,Y),R),
                     \+ memberchk(R,ListaOperadoras).

dFNop(G,X,Y,ListaOperadoras,V,[aresta(X,Z)|R]) :-
 adjacente(X,Z,G),
 auxAl(aresta(X,Z),RR),
 \+ memberchk(RR,ListaOperadoras),
 \+ member(Z,V),
 dFNop(G,Z,Y,ListaOperadoras,[Z|V],R),
 Z \= Y.
```

Figura 31: DF alínea 3

6.4 Alínea 4

```
%através do alg Greedy

grTraN(G,X,Y,ListaOperadoras,Soln) :-
            sucessor(G,X,PP),
            sortSuccessors(Y,PP,PriorityList),   % PRIORITY
            retira2(PriorityList,[],ListaOperadoras,Res), % retira da lista arestas que não sao feito com as dadas operadoras
            greedy3(G,X,Y,ListaOperadoras, Res,
            [],                                  % NODES VISITED
            Soln).                               % SOLUTION

greedy3(G,X,Y,ListaOperadoras,[], Visited, []).                % NO SOLUTION FOUND

greedy3(G,X,Y,ListaOperadoras,[aresta(X,Y)|OtherSolns], Visited, Soln):-
                                    append(Visited,[aresta(X,Y)],Soln).

greedy3(G,X,Y,ListaOperadoras,[aresta(Z,W)|OtherSolns], Visited, Soln) :-
    sucessor2(G,W,Visited,Successors),
    expande_greedy(G,Y,Successors,[],Cams), %retirar nodos que não tem adjacencia se estes não forem os pretendidos
    sortSuccessors(Y,Cams, PriorityQueue),
    retira2(PriorityQueue,[],ListaOperadoras,Res), !,
    greedy3(G,W,Y,ListaOperadoras,Res, [aresta(Z,W)|Visited], Soln).
```

Figura 32: greedy alínea 3

```
%atraves do alg A*

estOpN(G,X,Y,ListaOperadoras,Soln) :-
            sucessor(G,X,PP),
            sortSuccDist(Y,PP,PriorityList),    % PRIORITY
            retira2(PriorityList,[],ListaOperadoras,Res), %retira arestas de nao Operadoras
            star3(G,X,Y,ListaOperadoras,Res,
            [],                                  % NODES VISITED
            Soln).                               % SOLUTION

star3(G,X,Y,ListaOperadoras,[], Visited, []).                  % NO SOLUTION FOUND

star3(G,X,Y,ListaOperadoras,[aresta(X,Y)|OtherSolns], Visited, Soln):-
                                    append(Visited,[aresta(X,Y)],Soln).


star3(G,X,Y,ListaOperadoras,[aresta(Z,W)|OtherSolns], Visited, Soln) :-
    sucessor2(G,W,Visited,Successors),
    expande_greedy(G,Y,Successors,[],Cams),
    sortSuccDist(Y,Cams, PriorityQueue),
    retira2(PriorityQueue,[],ListaOperadoras,Res), !,
    star3(G,W,Y,ListaOperadoras,Res, [aresta(Z,W)|Visited], Soln).
```

Figura 33: A* alínea 3

```
%funçaõ auxiliar que retira arestas das operadoras nao desejadas
retira2([A],V,L,Res):-auxAl(A,P), nao(member(P,L)),
                        append(V,[A],Res).
retira2([A],V,L,V):-auxAl(A,P), member(P,L).
retira2([A|S],V,L,Res):-auxAl(A,P), member(P,L), retira2(S,V,L,Res).
retira2([A|S],V,L,Res):-auxAl(A,P),
                        nao(member(P,L)), append(V,[A],LL),
                        retira2(S,LL,L,Res).
```

Figura 34: auxiliar alínea 3

```
%devolve os N primeiros elementos de uma lista
nPrim(N, _, Xs) :- N =< 0, !, N =:= 0, Xs = [].
nPrim(_, [], []).
nPrim(N, [X|Xs], [X|Ys]) :- M is N-1, nPrim(M, Xs, Ys).

%>Numero de carreiras de uma paragem
numCarrP(paragem(_,_,_,_,_,_,_,C,_,_,_),R):- length(C,R).
```

Figura 35: aux alínea 4

6.5 Alínea 5

```
%ordena lista de paragens pela sua quantidade de carreiras

quick_sort2(List,Sorted):-q_sort(List,[],Sorted).
q_sort([],Acc,Acc).
q_sort([H|T],Acc,Sorted):-
    pivoting(H,T,L1,L2),
    q_sort(L1,Acc,Sorted1),q_sort(L2,[H|Sorted1],Sorted).
pivoting(H,[],[],[]).
pivoting(H,[X|T],[X|L],G):-numCarrP(H,HH),numCarrP(X,XX),XX=<HH,pivoting(H,T,L,G).
pivoting(H,[X|T],L,[X|G]):-numCarrP(H,HH),numCarrP(X,XX),XX>HH,pivoting(H,T,L,G).
```

Figura 36: aux alínea 4

```
%devolve path com menos numero de paragens
menorPercursoDF(G,X,Y,R) :- findall((P,Q),(dFirst(G,X,Y,P),
                              length(P,Q), Q>0),Tamanhos),
                             devolveP(Tamanhos,B,300000,R).

menorPercursoGreedy(G,X,Y,R) :- findall((P,Q),(greedy(G,X,Y,P),
                              length(P,Q), Q>0),Tamanhos),
                             devolveP(Tamanhos,B,300000,R).
menorPercursoEstrela(G,X,Y,R) :- findall((P,Q),(estrela(G,X,Y,P),
                             length(P,Q), Q>0),Tamanhos),
                              devolveP(Tamanhos,B,300000,R).

devolveP([(C,D)],Q,MENOR,C):-D=<MENOR.
devolveP([(C,D)],Q,MENOR,Q):-D>MENOR.
devolveP([(C,D)|X],Q,MENOR,P):-D=<MENOR, devolveP(X,C,D,P).
devolveP([(C,D)|X],Q,MENOR,P):-D>MENOR, devolveP(X,Q,MENOR,P).
```

Figura 37: aux alínea 5

```
%devolve todos os caminhos possiveis entre dois pontos (diferentes algoritmos):

allPathsDF(G,X,Y,P):-setof(Caminhos,dFirst(G,X,Y,Caminhos),P).

allPGreedy(G,X,Y,P):-setof(Caminhos,greedy(G,X,Y,Caminhos),P).

allPEstrela(G,X,Y,P):-setof(Caminhos,estrela(G,X,Y,Caminhos),P).
```

Figura 38: aux alínea 5

6.6 Alínea 6

```
%%Calcula distancia total de um path
calcDist([],Di,Di).
calcDist([A],Di,Di).
calcDist([A|X],Di,Dist):- pp(A,A1), head(X,X2), pp(X2,X22), head(A1,A2), head(X22,X3),
                          distEntreParagens(A2,X3,Dist2),
                          ZZ is Dist2 + Di,
                          calcDist(X,ZZ,Dist).
```

Figura 39: A* alínea 6

6.7 Alínea 7

6.8 Alínea 8

```
%com a depth first

dFP(G,X,Y,R) :- dFP2(G,X,Y,[X],R).

dFP2(G,X,Y,V,[aresta(X,Y)]) :- adjacente(X,Y,G), pp(aresta(X,Y),P), head(P,H), temPub(H).

dFP2(G,X,Y,V,[aresta(X,Z)|R]) :-
 adjacente(X,Z,G),
 pp(aresta(X,Z),P),
 head(P,H),
 temPub(H),
 \+ memberchk(aresta(X,Z),V),
 \+ member(Z,V),
 dFP2(G,Z,Y,[Z|V],R),
 Z \= Y.
```

Figura 40: Depth First alínea 7

```
%Com alg Greedy

grePub(G,X,Y,Soln) :-
            sucessor(G,X,PP),
            sortSuccessors(Y,PP,PriorityList),   % PRIORITY
            retiraComP(PriorityList,[],Res),     % retira da paragens sem publicidade
            greedy4(G,X,Y, Res,
            [],                                  % NODES VISITED
            Soln).                               % SOLUTION

greedy4(G,X,Y,[], Visited, []).                  % NO SOLUTION FOUND

greedy4(G,X,Y,[aresta(X,Y)|OtherSolns], Visited, Soln):-
                                    append(Visited,[aresta(X,Y)],Soln).


greedy4(G,X,Y,[aresta(Z,W)|OtherSolns], Visited, Soln) :-
    sucessor2(G,W,Visited,Successors),
    expande_greedy(G,Y,Successors,[],Cams), %retirar nodos que não tem adjacencia se estes não forem os pretendidos
    sortSuccessors(Y,Cams, PriorityQueue),
    retiraComP(PriorityQueue,[],Res), !,
    greedy4(G,W,Y,Res, [aresta(Z,W)|Visited], Soln).
```

Figura 41: Greedy alínea 7

```
%Com alg estrela

estrelaPub(G,X,Y,Soln) :-
            sucessor(G,X,PP),
            sortSuccDist(Y,PP,PriorityList),
            retiraComP(PriorityList,[],Dev),        % Retira sem publicidade
            star4(G,X,Y,Dev,             % PRIORITY
            [],                                     % NODES VISITED
            Soln).                                  % SOLUTION

star4(G,X,Y,[], Visited, []).                  % NO SOLUTION FOUND

star4(G,X,Y,[aresta(X,Y)|OtherSolns], Visited, Soln):-append(Visited,[aresta(X,Y)],Soln).


star4(G,X,Y,[aresta(Z,W)|OtherSolns], Visited, Soln) :-
    sucessor2(G,W,Visited,Successors),
    expande_greedy(G,Y,Successors,[],Cams),
    sortSuccDist(Y,Cams, PriorityQueue),
    retiraComP(PriorityQueue,[],Dev), !,
    star4(G,W,Y,Dev, [aresta(Z,W)|Visited], Soln).
```

Figura 42: A* First alínea 7

6.9 Alínea 9

```
dFAbrigo(G,X,Y,R) :- dFab(G,X,Y,[X],R).

dFab(G,X,Y,V,[aresta(X,Y)]) :- adjacente(X,Y,G), pp(aresta(X,Y),P), head(P,H), temAbrigo(H).

dFab(G,X,Y,V,[aresta(X,Z)|R]) :-
 adjacente(X,Z,G),
 pp(aresta(X,Z),P),
 head(P,H),
 temAbrigo(H),
 \+ memberchk(aresta(X,Z),V),
 \+ member(Z,V),
 dFab(G,Z,Y,[Z|V],R),
 Z \= Y. %maybe unecessary
```

Figura 43: Depth First alínea 8

```
%Com alg Greedy

greAbrigo(G,X,Y,Soln) :-
            sucessor(G,X,PP),
            sortSuccessors(Y,PP,PriorityList),   % PRIORITY
            retiraSAb(PriorityList,[],Res),     % retira da paragens sem abrigo
            greedy5(G,X,Y, Res,
            [],                                  % NODES VISITED
            Soln).                               % SOLUTION

greedy5(G,X,Y,[], Visited, []).                  % NO SOLUTION FOUND

greedy5(G,X,Y,[aresta(X,Y)|OtherSolns], Visited, Soln):-
                                        append(Visited,[aresta(X,Y)],Soln).


greedy5(G,X,Y,[aresta(Z,W)|OtherSolns], Visited, Soln) :-
    sucessor2(G,W,Visited,Successors),
    expande_greedy(G,Y,Successors,[],Cams), %retirar nodos que não tem adjacencia se estes não forem os pretendidos
    sortSuccessors(Y,Cams, PriorityQueue),
    retiraSAb(PriorityQueue,[],Res), !,
    greedy5(G,W,Y,Res, [aresta(Z,W)|Visited], Soln).
```

Figura 44: Greedy alínea 8

```
%com alg Estrela

estrelaAbrigo(G,X,Y,Soln) :-
            sucessor(G,X,PP),
            sortSuccDist(Y,PP,PriorityList),
            retiraSAb(PriorityList,[],Dev),        % Retira sem abrigo
            star5(G,X,Y,Dev,                      % PRIORITY
            [],                                   % NODES VISITED
            Soln).                                % SOLUTION

star5(G,X,Y,[], Visited, []).                    % NO SOLUTION FOUND

star5(G,X,Y,[aresta(X,Y)|OtherSolns], Visited, Soln):-append(Visited,[aresta(X,Y)],Soln).


star5(G,X,Y,[aresta(Z,W)|OtherSolns], Visited, Soln) :-
    sucessor2(G,W,Visited,Successors),
    expande_greedy(G,Y,Successors,[],Cams),
    sortSuccDist(Y,Cams, PriorityQueue),
    retiraSAb(PriorityQueue,[],Dev), !,
    star5(G,W,Y,Dev, [aresta(Z,W)|Visited], Soln).
```

Figura 45: A* First alínea 8

```
%com alg greedy

greedyIntermedios(G,X,L,R) :- dFIntermedios3(G,X,L,[],R).

dFIntermedios3(G,X,[L],V,R) :- greedy(G,X,L,P),concat(V,P,R).
dFIntermedios3(G,X,[L|LS],V,R):-
    greedy(G,X,L,P),
    concat(V,P,Z),
    dFIntermedios3(G,L,LS,Z,R).

%com alg estrela

estrelaIntermedios(G,X,L,R) :- dFIntermedios4(G,X,L,[],R).

dFIntermedios4(G,X,[L],V,R) :- estrela(G,X,L,P),concat(V,P,R).
dFIntermedios4(G,X,[L|LS],V,R):-
    estrela(G,X,L,P),
    concat(V,P,Z),
    dFIntermedios4(G,L,LS,Z,R).
```

Figura 46: Greedy e A* alínea 9