RELATÓRIO DO SPRINT C

Turma 2DH _ Grupo 43
1190929 _ Patrícia Barbosa
1190947 _ Pedro Fraga
1190956 _ Pedro Garcia
1190963 _ Pedro Preto

Professora:
Brígida Teixeira, BCT

Unidade Curricular:
Algoritmia Avançada

Data: 02/01/2022

ÍNDICE

ÍNDICE DE FIGURAS

Rede à Parte com utilizadores que podem ser atingidos até N ligações:

```
/* O(n * log^2(n)) */
net_size_up_to_level(UserName, Level):- /* Este predicado ex
    no(UserId,UserName,_),
    nb_setval(rede,[]),
    net_size_up_to_level_aux(UserId, Level, _AllInNet),
    nb_getval(rede, R),
    sort(R, AuxR),
    length(AuxR, AuxRLength),
    write("Rede: "), write(AuxR), nl,
    write("Numero de Connections: "), write(AuxRLength),nl,
    getWorkingDirectory(Dir),write(Dir),nl,
    concat(Dir, 'netOfUserUpToLevelBC.pl', FileName),

    open(FileName, write, File),
    write_to_file(AuxR, File),
    close(File), !.
```

Figura 1 - net_size_up_to_level/2

Este predicado recebe o nome do utilizador pretendido e o nível que é pretendido obter a rede até, escreve na consola a rede até ao nível selecionado (os IDs dos utilizadores) e o número total de ligações encontradas. De seguida guarda as ligacoes em uma base de conhecimento com a rede até ao nível selecionado, escrevendo o conteúdo dessa nova base de conhecimento também na consola.

Usa o método “net_size_up_to_level_aux”, que encontrar todos os utilizadores até um nível partindo de um utilizado:

```
net_size_up_to_level_aux(UserId, Level, AllInNet):-
    findall(Path, dfscl(UserId, _Dest, Path, Level), AllPaths),  /* E
    (uma lista de listas de utilizadores, cada caminho é uma lista de
    flatten(AllPaths, Aux), /* Transforma a lista de listas de utili
    sort(Aux, AllInNet), /* O sort é utilizado únicamente para remove
    length(AllInNet, Size), /* Obtém o tamanho da lista de utilizador
    o número de utilizadores na rede até ao nível escolhido */
    nl,write('All Users in Net Up to Level: '), write(AllInNet),nl, /
    write('Size of Net up to selected Level: '), write(Size),nl. /* E
```

Figura 2 - net_size_up_to_level_aux/3

A escrita na base de conhecimento é feita da seguinte forma:

```
write_to_file([],_):-!.
write_to_file([[U1,U2|_R]|LConnections], File):-
    ligacao(U1, U2, A, B, C, D), !,
    write("ligacao("),write(U1),write(","),write(U2),write(","),write(A),write(","),write(B),write(","),write(C),write(","),write(D),write(")"),nl,
    write_term(File, ligacao(U1, U2, A, B, C, D),[fullstop(true),nl(true),quoted(true)]),
    write_to_file(LConnections, File).

write_to_file([[U1,U2|_R]|LConnections], File):-
    ligacao(U2, U1, A, B, C, D),nl,
    write_term(File, ligacao(U1, U2, A, B, C, D),[fullstop(true),nl(true),quoted(true)]),
    write_to_file(LConnections, File).
```

Figura 3 - write_to_ file/2

O ficheiro é aberto no diretório selecionado no ficheiro "directory.pl":

```
:-include('directory.pl').
```

Figura 4 - include diretorio.pl

De seguida as ligações que previamente foram escritas na consola são guardadas na base de conhecimento.

Exemplificação:

- Exemplo 1:
  User - ana
  Nível - 1

```
?- net_size_up_to_level(ana, 1).

All Users in Net Up to Level: [1,11,12,13,14,51]
Size of Net up to selected Level: 6
Rede: [[1,11],[1,12],[1,13],[1,14],[1,51]]
Numero de Connections: 5
C:/Users/topel/Documents/lapr5_g43/projeto base exemplo/nunopsilva-dddnetcore-c2122b82f1b9/ALGAV/Sprint C/
ligacao(1,11,10,8,4,-1)
ligacao(1,12,2,6,2,8)
ligacao(1,13,-3,-2,1,-1)
ligacao(1,14,1,-5,8,7)
ligacao(1,51,6,2,2,5)
true.
```

  Base de conhecimento criada:

```
ligacao(1,11,10,8,4,-1).
ligacao(1,12,2,6,2,8).
ligacao(1,13,-3,-2,1,-1).
ligacao(1,14,1,-5,8,7).
ligacao(1,51,6,2,2,5).
```

Figura 5 - base de conhecimento

- Exemplo 2:
  User - ana
  Nível - 2

```
?- net_size_up_to_level(ana, 2).

All Users in Net Up to Level: [1,11,12,13,14,21,22,23,24,51,61]
Size of Net up to selected Level: 11
Rede: [[1,11],[1,12],[1,13],[1,14],[1,51],[11,21],[11,22],[11,23],[11,24],[12,21],[12,22],[12,23],[12,24],[13,21],[
13,22],[13,23],[13,24],[14,21],[14,22],[14,23],[14,24],[51,61]]
Numero de Connections: 22
C:/Users/topel/Documents/lapr5_g43/projeto base exemplo/nunopsilva-dddnetcore-c2122b82f1b9/ALGAV/Sprint C/
ligacao(1,11,10,8,4,-1)
ligacao(1,12,2,6,2,8)
ligacao(1,13,-3,-2,1,-1)
ligacao(1,14,1,-5,8,7)
ligacao(1,51,6,2,2,5)
ligacao(11,21,5,7,3,2)
ligacao(11,22,2,-4,1,1)
ligacao(11,23,-2,8,7,6)
ligacao(11,24,6,0,2,-5)
ligacao(12,21,4,9,-4,-1)
ligacao(12,22,-3,-8,8,4)
ligacao(12,23,2,4,4,7)
ligacao(12,24,-2,4,2,7)
ligacao(13,21,3,2,3,2)
ligacao(13,22,0,-3,1,-6)
ligacao(13,23,5,9,4,-1)
ligacao(13,24,-2,4,-7,0)
ligacao(14,21,2,6,1,1)
ligacao(14,22,6,-3,9,-2)
ligacao(14,23,7,0,8,8)
ligacao(14,24,2,2,0,1)
ligacao(51,61,7,3,-5,-5)
true.
```

Figura 6 - net_size_up_to_level/2

Base de conhecimento criada:

```
ligacao(1,11,10,8,4,-1).
ligacao(1,12,2,6,2,8).
ligacao(1,13,-3,-2,1,-1).
ligacao(1,14,1,-5,8,7).
ligacao(1,51,6,2,2,5).
ligacao(11,21,5,7,3,2).
ligacao(11,22,2,-4,1,1).
ligacao(11,23,-2,8,7,6).
ligacao(11,24,6,0,2,-5).
ligacao(12,21,4,9,-4,-1).
ligacao(12,22,-3,-8,8,4).
ligacao(12,23,2,4,4,7).
ligacao(12,24,-2,4,2,7).
ligacao(13,21,3,2,3,2).
ligacao(13,22,0,-3,1,-6).
ligacao(13,23,5,9,4,-1).
ligacao(13,24,-2,4,-7,0).
ligacao(14,21,2,6,1,1).
ligacao(14,22,6,-3,9,-2).
ligacao(14,23,7,0,8,8).
ligacao(14,24,2,2,0,1).
ligacao(51,61,7,3,-5,-5).
```

Figura 7 - base de conhecimento

Adaptação do A*:

```
aStar(Orig,Dest,MaximoLigacoes,Cam,Custo):-
  aStar2(Dest,[(_,0,[Orig])],MaximoLigacoes,Cam, Custo).

aStar2(Dest,[(_,Custo,[Dest|T])|_],_MaximoLigacoes,Cam,Custo):-
  reverse([Dest|T],Cam).

aStar2(Dest,[(_,Ca,LA)|Outros],MaximoLigacoes,Cam,Custo):-
  LA=[Act|_],
  findall((CEX,CaX,[X|LA]),
    (Dest\==Act,edge(Act,X,CustoX),\+ member(X,LA),
    CaX is CustoX + Ca, estimativa(Act,X,EstX),
    CEX is CaX + EstX),Novos),
  append(Outros,Novos,Todos),
  %write("New="),write(Novos),nl,nl,
  sort(Todos,AllSorted),
  retrack_paths(AllSorted,MaximoLigacoes,AllSortedMaxLig),
  %write("AllSorted="),write(AllSortedMaxLig),nl,nl,
  aStar2(Dest,AllSortedMaxLig,MaximoLigacoes,Cam,Custo).
```

Figura 8 – aStar/5

Para a adaptação do "A*" foi adicionada a chamada do predicado "retract_paths" que retira da lista com todos os caminhos possíveis, todos os caminhos com mais de n ligações.

A base de conhecimento a que o "A*" vai aceder ("netOfUserUpToLevelBC.pl"), é gerada por outro predicado("net_size_up_to_level"), que cria uma base de conhecimento só com um máximo de n ligações.

```
edge(U1,U2,F):-
no(Id1,U1,_),
no(Id2,U2,_),
is_bidirectional(Id1,Id2,F).

is_bidirectional(X,Y,Z):- (ligacao(X,Y,F1,F2,_,_),Z is F1+F2;ligacao(Y,X,F1,F2,_,_),Z is F1+F2),!.
```

Figura 9 - edge/3

O predicado egde recebe dois nomes de certos "users" e acede aos seus respetivos ids, que por sua vez vão ser enviados para o predicado "is_bidirectional" devolvendo assim a soma das forças de ligação entre os dois nós.

```
retrack_paths([],_,[]).
retrack_paths([First|Next],MaximoLigacoes,Lnew):-
        length(First,Size), S is Size-1, S > MaximoLigacoes,!,
        retrack_paths(Next,MaximoLigacoes,Lnew).
retrack_paths([First|Next],MaximoLigacoes,[First|Lnew]):-
        retrack_paths(Next,MaximoLigacoes,Lnew).
```

Figura 10 - retract_paths/3

O predicado "retract_paths" recebe uma Lista com todos os caminhos possíveis e devolve outra lista apenas com os caminhos até n ligações (MaximoLigacoes).

Exemplificação:

- Exemplo:
  Origem - ana
  Destino - eduardo
  Máximo de Ligações - 2

```
?- aStar(ana, eduardo, 2, Caminho, Custo).
Caminho = [ana, carlos, eduardo],
Custo = 50.0 ;

?- aStar(ana, eduardo, 1, Caminho, Custo).
false.
```

Figura 11 - exemplo aStar

Neste caso, o melhor caminho encontrado foi ana -> carlos -> eduardo, com um custo final de 50, no caso do predicado ser chamado com apenas uma ligação, nenhum caminho é encontrado entre os utilizadores ana e eduardo, retornando então false.

```
?- aStar(ana, eduardo, 2, Caminho, Custo).
Caminho = [ana, carlos, eduardo],
Custo = 50.0 ;
Caminho = [ana, daniel, eduardo],
Custo = 51.625 ;
Caminho = [ana, beatriz, eduardo],
Custo = 54.875 ;
Caminho = [ana, antonio, eduardo],
Custo = 59.375 ;
false.
```

Figura 12 - output do exemplo da figura 11

Ao listar todos os resultados obtidos com esta chamada podemos verificar que o caminho obtido é o caminho com menor custo.

Adaptação do Best First:

```
bestfs1(Orig,Dest,MaximoLigacoes,Cam,Custo):-
bestfs12(Dest,[[Orig]],MaximoLigacoes,Cam,Custo),
    write('Caminho-'),write(Cam),nl,!.

bestfs12(Dest,[[Dest|T]|_],_,Cam,Custo):- reverse([Dest|T],Cam),
    calcula_custo(Cam,Custo).

bestfs12(Dest,[[Dest|_]|LLA2],MaximoLigacoes,Cam,Custo):- !, bestfs12(Dest,LLA2,MaximoLigacoes,Cam,Custo).

bestfs12(Dest,LLA,MaximoLigacoes,Cam,Custo):- member1(LA,LLA,LLA1), LA=[Act|_],
    ((Act==Dest,!,bestfs12(Dest,[LA|LLA1],MaximoLigacoes,Cam,Custo))
    ;
    (findall((CX,[X|LA]),(edge(Act,X,CX), \+member(X,LA)),Novos),
    Novos\==[],!,
    sort(0,@>=,Novos,NovosOrd),
    retira_custos(NovosOrd,NovosOrd1),
    append(NovosOrd1,LLA1,LLA2),
    write('============================================='),nl,
    write('LLA2='),write(LLA2),nl,
    retrack_paths(LLA2,MaximoLigacoes,LLA2V2),
    write('LLA2V2='),write(LLA2V2),nl,
    write('============================================='),nl,

    bestfs12(Dest,LLA2V2,MaximoLigacoes,Cam,Custo) )).
```

Figura 13 - bestfs1/5

Na adaptação do “best first” apenas adicionei um novo parâmetro “MaximoLigacoes” que define o número máximo de ligações, e adicionei a chamada do predicado “retract_paths” que retira da lista com todos os caminhos possíveis, todos os caminhos com mais de n ligações.

A base de conhecimento a que o “best first” vai aceder (“netOfUserUpToLevelBC.pl”), é gerada por outro predicado(“net_size_up_to_level”), que cria uma base de conhecimento só com um máximo de n ligações.

O predicado egde e is_bidirectional é novamente usado como mencionado na secção do A*, tal como o retrack_paths.

Exemplificação:

- Exemplo 1:
  Origem-ana
  Destino-preto
  Máximo de Ligações-4

```
?- bestfs1(ana, preto,4,Caminho,Custo).

Caminho = [ana, rodolfo, rita, preto],
Custo = 83.625.
```

Figura 14 - exemplo bestfs1/4

- Exemplo 2:
  Origem-ana
  Destino-preto
  Máximo de Ligações-6

```
?- bestfs1(ana, preto,6,Caminho,Custo).

Caminho = [ana, antonio, eduardo, catia, garcia, preto],
Custo = 139.375.
```

Figura 15 - exemplo bestfs1/4

Adaptação do Primeiro em Profundidade:

```
dfs(Orig,Dest,MaxLig,Cam):-dfs2(Orig,Dest,[Orig],MaxLig,Cam).

dfs2(Dest,Dest,LA,MaxLig,Cam):-!,length(LA,Size),S is Size-1,MaxLig>=S,reverse(LA,Cam).
dfs2(Act,Dest,LA,MaxLig,Cam):-no(NAct,Act,_),ligacao(NAct,NX,_,_,_,_),
    no(NX,X,_),\+ member(X,LA),dfs2(X,Dest,[X|LA],MaxLig,Cam).
```

Figura 16 - dfs/4

A adaptação do Primeiro em Profundidade (dfs) consistiu na adição do parâmetro de máximo de ligações, pelo que não é necessário utilizar qualquer outro método de suporte, a pesquisa em profundidade é simplesmente "cortada" quando chega ao nível máximo e prossegue para o próximo "ramo".

Exemplificação:

- Exemplo 1:
  Origem - ana
  Destino - eduardo
  Máximo de Ligações - 1, 2 e 3

```
?- dfs(ana,eduardo,1,Cam).
false.

?- dfs(ana,eduardo,2,Cam).
Cam = [ana, antonio, eduardo] .

?- dfs(ana,eduardo,3,Cam).
Cam = [ana, antonio, eduardo] .

?- dfs(ana,eduardo,3,Cam).
Cam = [ana, antonio, eduardo] ;
Cam = [ana, beatriz, eduardo] ;
Cam = [ana, carlos, eduardo] ;
Cam = [ana, daniel, eduardo] ;
false.
```

Figura 17 - exemplo dfs/4

No primeiro caso não é possível encontrar um caminho com apenas uma ligação entre a ana e o eduardo, visto que eles não são amigos diretos.
No segundo caso é encontrado o caminho ana -> antonio -> eduardo.
No terceiro caso o caminho encontrado é o mesmo que no segundo caso, visto que continua a ser o melhor caminho, isto pode ser verificado com a última chamada do predicado, em que podem ser verificados todos os caminhos encontrados, não existe nenhum caminho entre a ana e o eduardo com 3 ligações, portanto o melhor caminho terá duas ligações, ou seja, o mesmo resultado que no segundo caso.

- Exemplo 2:
  Origem - ana
  Destino - eduardo
  Máximo de Ligações - 1, 2 e 3
  Nota: Diferencia-se do primeiro exemplo, pois foi adicionada manualmente uma ligação entre a ana e o eduardo, por motivos de demonstração.

```
?- dfs(ana,eduardo,1,Cam).
Cam = [ana, eduardo].

?- dfs(ana,eduardo,2,Cam).
Cam = [ana, antonio, eduardo] .

?- dfs(ana,eduardo,3,Cam).
Cam = [ana, antonio, eduardo] .

?- dfs(ana,eduardo,3,Cam).
Cam = [ana, antonio, eduardo] ;
Cam = [ana, beatriz, eduardo] ;
Cam = [ana, carlos, eduardo] ;
Cam = [ana, daniel, eduardo] ;
Cam = [ana, eduardo].
```

Figura 18 - exemplo dfs/4

Com este exemplo pode-se verificar que agora é encontrado um caminho entre a ana e o eduardo, mas este será sempre a última opção encontrada nos outros dois casos, visto que, aumentando o número de ligações a percorrer, aumenta também a força de ligação total do caminho, portanto, geralmente, quantas mais ligações em um caminho, maior a sua força.

Comparação dos 3 métodos:

Para efetuar a comparação dos 3 métodos, decidi criar uma rede à parte, para controlar melhor o caso de estudo. Para além disso, criei um ficheiro "Teste.pl", onde consigo testar os 3 métodos ao mesmo tempo, utilizando o predicado run_test/3. Este ficheiro contém uma cópia do DFS e do Best_First, e inclui o ficheiro "A_Star.pl", visto que não estava a conseguir integrá-lo no ficheiro da mesma forma que os anteriores. Para os testes correrem com a mesma rede, é necessário alterar a base de conhecimentos incluída pelo "A_Star.pl", para a base de conhecimentos criar para testes, "BC_teste.pl".

O objetivo da comparação é testar a eficiência, rapidez e qualidade das soluções geradas pelos 3 algoritmos.

O caso de estudo será o caminho entre "nuno" e "fraga" com um máximo de 6 ligações.

De início, a base de conhecimentos é a seguinte:

```
no(1,nuno,[natureza,pintura,musica,sw,porto]).
no(2,maria,[natureza,cinema,teatro,carros,coimbra]).
no(3,isabel,[natureza,musica,porto,lisboa,cinema]).
no(4,jose,[natureza,pintura,sw,musica,carros,lisboa]).
no(5,luisa,[natureza,cinema,jogos,moda,porto]).
no(6,leonor,[natureza,pintura,musica,moda,porto]).
no(7,anabela,[natureza,cinema,musica,tecnologia,porto]).
no(8,andre,[natureza,carros,futebol,coimbra]).
no(9,catia,[natureza,musica,cinema,lisboa,moda]).
no(10,garcia,[natureza,teatro,tecnologia,futebol,porto,bolachas]).
no(11,patricia,[natureza,futebol,sw,jogos,porto,bolachas]).
no(12,fraga,[natureza,teatro,carros,porto,bolachas]).
no(13,tatiana,[natureza,moda,tecnologia,cinema]).
no(14,preto,[natureza,bolachas,moda,musica,sw,coimbra]).
no(15,antonio,[natureza,pintura,carros,futebol,lisboa]).
no(16,beatriz,[natureza,musica,carros,porto,moda]).
no(17,silva,[natureza,cinema]).
no(18,carlos,[natureza,musica,sw,futebol,coimbra]).
no(51,rodolfo,[natureza,musica,sw]).
no(61,rita,[moda,tecnologia,cinema]).

ligacao(1,12,5,7,3,2).
ligacao(1,2,-10,-8,-4,-1).
ligacao(2,6,-2,-6,-2,-8).
ligacao(6,16,-3,-2,-1,-1).
ligacao(1,14,-1,-5,-8,-7).
ligacao(16,12,-6,-2,-2,-5).
```

Figura 19: Base de conhecimentos inicial para estudo

Existem 2 caminhos possíveis: [nuno,fraga] e [nuno,maria,leonor,beatriz,fraga], sendo a ligação direta entre os dois nós a opção de menor custo. Quando corremos o teste, estes são os resultados.

```
?- run_test(nuno,fraga,6).
DFS
[nuno,fraga]
Tempo de geracao da solucao:0.0

BEST FIRST
[nuno,fraga]
Tempo de geracao da solucao:0.0

ASTAR
[nuno,fraga]
Tempo de geracao da solucao:0.00650787353515625
true ;
```

Figura 20: Resultado do primeiro teste

Podemos observar que o A* foi ligeiramente mais lento, mas que todos devolveram o mesmo caminho.

De seguida decidi trocar a ordem das ligações, passando a ligação direta entre os nós "nuno" e "fraga" a ser a última da lista. Pelo meio, adicionei também mais ligações, que não interferem com o caso de estudo, com o intuito de comparar melhor o tempo de geração de solução e simular um caso mais realista. Os resultados são ligeiramente diferentes:

```
ligacao(1,2,-10,-8,-4,-1).
ligacao(2,6,-2,-6,-2,-8).
ligacao(6,16,-3,-2,-1,-1).
ligacao(1,14,-1,-5,-8,-7).
ligacao(16,12,-6,-2,-2,-5).
ligacao(11,22,2,-4,1,1).
ligacao(11,23,-2,8,7,6).
ligacao(11,24,6,0,2,-5).
ligacao(12,21,4,9,-4,-1).
ligacao(12,22,-3,-8,8,4).
ligacao(12,23,2,4,4,7).
ligacao(12,24,-2,4,2,7).
ligacao(13,21,3,2,3,2).
ligacao(13,22,0,-3,1,-6).
ligacao(13,23,5,9,4,-1).
ligacao(13,24,-2,4,-7,0).
ligacao(14,21,2,6,1,1).
ligacao(14,22,6,-3,9,-2).
ligacao(14,23,7,0,8,8).
ligacao(14,24,2,2,0,1).
ligacao(51,61,7,3,-5,-5).
ligacao(1,12,5,7,3,2).
```

Figura 21: Base de conhecimentos com novas ligacoes

```
?- run_test(nuno,fraga,6).
DFS
[nuno,maria,leonor,beatriz,fraga]
Tempo de geracao da solucao:0.0049860477447509766

BEST FIRST
[nuno,fraga]
Tempo de geracao da solucao:0.004985809326171875

ASTAR
[nuno,fraga]
Tempo de geracao da solucao:0.009003877639770508
true ;
```

Figura 22: Resultado do segundo teste

Tanto o A* como o Best First continuam a devolver o melhor caminho, sendo que o A* demora o dobro do tempo. Já o DFS, devolveu o caminho mais comprido. Isto acontece porque o DFS devolve o primeiro caminho que encontra. Por isso podemos concluir que é um algoritmo rápido, mas devolve o primeiro caminho encontrado em profundidade, pelo que nunca podemos afirmar que o caminho encontrado é o melhor ou mais curto.

Por fim, removi a ligação direta entre os dois nós de estudo, adicionei mais ligações, de modo a existirem mais caminhos alternativos, e imprimi o custo, para melhor averiguar a diferença do A* para o Best First.

Para este caso, o percurso mais otimizado e de menor custo será [nuno,maria,fraga]. A base de conhecimentos e os resultados são os seguintes:

```
ligacao(1,2,-10,-8,-4,-1).
ligacao(2,6,-2,-6,-2,-8).
ligacao(6,16,-3,-2,-1,-1).
ligacao(16,12,-6,-2,-2,-5).
ligacao(1,14,2,5,5,7).
ligacao(14,21,8,9,-8,9).
ligacao(21,12,-5,3,-5,-9).
ligacao(11,22,2,-4,1,1).
ligacao(11,23,-2,8,7,6).
ligacao(11,24,6,0,2,-5).
ligacao(12,21,4,9,-4,-1).
ligacao(12,22,-3,-8,8,4).
ligacao(12,23,2,4,4,7).
ligacao(12,24,-2,4,2,7).
ligacao(13,21,3,2,3,2).
ligacao(13,22,0,-3,1,-6).
ligacao(13,23,5,9,4,-1).
ligacao(13,24,-2,4,-7,0).
ligacao(14,22,6,-3,9,-2).
ligacao(14,23,7,0,8,8).
ligacao(23,12,6,0,2,1).
ligacao(13,12,6,0,2,4).
ligacao(1,23,5,4,-4,2).
ligacao(14,24,2,2,0,1).
ligacao(51,61,7,3,-5,-5).
%ligacao(1,12,5,7,3,2).
ligacao(2,12,-5,-5,-5,-5).
```

Figura 23: Alteração final na base de conhecimentos para teste

```
?- run_test(nuno,fraga,6).
DFS
[nuno,maria,leonor,beatriz,fraga]
Tempo de geracao da solucao:0.0

BEST FIRST
[nuno,maria,leonor,beatriz,fraga]
Custo = 87.25
Tempo de geracao da solucao:0.08356809616088867

ASTAR
[nuno,maria,fraga]
Custo = 40.875
Tempo de geracao da solucao:0.021745920181274414
true
```

Figura 24: Resultado do último teste

Como podemos ver, desta vez apenas o A* devolveu o percurso mais otimizado, e de forma mais rápida. Isto acontece porque o best first escolhe o "nó mais promissor" sempre que tem de escolher um novo nó, o que nem sempre leva à resposta mais acertada, porque

neste caso, o nó mais promissor ao chegar à “maria”, é “leonor”, mas após verificar o custo total (que o best first não faz) do percurso, teria sido mais vantajoso escolher como próximo nó “fraga”.

Em conclusão:

NOME	OTIMIZACAO	TEMPO
Primeiro em profundidade	Devolve o primeiro caminho encontrado em profundidade. Não faz qualquer tipo de otimização.	Regra geral, o mais rápido dos três.
Best first	Faz otimizações a cada nó, mas não garante que o resultado final é o mais otimizado.	Regra geral um pouco mais lento que o primeiro em profundidade, e um puco mais rápido que o A star
A Star	Garante que o caminho devolvido é o mais otimizado	Geralmente o mais lento. Depende do tamanho da rede considerada.

Função Multicritério:

Se considerarmos uma conexão vamos ter uma parcela que varia entre 0 e 100 (força de ligação) e outra que, mesmo que limitada, variará entre um mínimo e um máximo (por exemplo, entre -200 e +200). A resultante variará entre -200 e 300. Notar que a força da ligação contribui com uma variação de 100 e a da relação com uma variação de 400.

A Função Multicritério recebe a força de ligação e a força de relação entre dois utilizadores e retorna um valor entre 0 e 100 em que é considerado ambas as forças com o mesmo peso no valor final.

```
funcao_multicriterio(FL,FR,Res):-
                        verifica_limites(FR,NFR),
                        calcula_peso_forca_ligacao(FL,VAR1),
                        calcula_peso_forca_relacao(NFR,VAR2), Res is (VAR1 + VAR2).
```

Figura 25 - funcao_multicriterio/3

Considerando que o nosso problema é de maximização, interessa que as estimativas sejam um majorante, definindo-se assim como valores limite para a força de relação -200 e 200. Sendo assim a força de relação tem de ser verificada à priori de se fazer qualquer cálculo, caso esta seja inferior a -200, o valor a ser considerado é -200. Caso esta seja superior a 200, o valor considerado é 200. Esta verificação é feita através do verifica_limites/2 que recebe uma força de relação como 1º parâmetro e guarda no segundo o novo valor da força de relação.

```
/* Verificar se a relação entre likes e dislikes é superior a 200 ou inferior a -200 e
atribuir os valores maximos caso aconteca */

verifica_limites(FR, NFR):- (FR < -200) -> NFR is -200; (FR > 200) -> NFR is 200; NFR is FR.
```

Figura 266 - verifica_limites/2

Tendo em conta que queremos que ambas as forças tenham o mesmo peso no valor final, e que o valor da força de ligação varia entre 0 e 100, calcular o peso desta força para o resultado final é apenas multiplicar o valor da própria força por 0.5.

```
/* Calcular o peso da força de ligação -> 50% do valor da força de ligação */

calcula_peso_forca_ligacao(FL, VAR1):- VAR1 is (0.5 * FL).
```

Figura 277 - calcula_peso_forca_ligacao/2

Continuando o raciocínio anterior, o peso da força de relação para o valor final também é de 50%, no entanto, uma vez que o valor desta força pode variar entre -200 e 200, é necessário passar este intervalo a positivo (somar 200 à força de relação, passando a variar entre 0 e 400) e dividir por 4 (para termos uma variação de 100 e não de 400). A este valor será então necessário apenas multiplicar por 0.5 e obtemos o peso da força de relação.

```
/* Calcular o peso da força de relação -> 50% de (FR+100)/4 */

calcula_peso_forca_relacao(FR, VAR2):- VAR2 is (((FR+200)/4)*0.5).
```

Figura 288 - calcula_peso_forca_relacao/2

Adaptação do A*, Best First e Primeiro em Profundidade para considerar a função multicritério:

Quer o algoritmo A* quer o Best First utilizavam a estimativa para calcular o custo do caminho de um nó a outro. Posteriormente ao desenvolvimento da função multicritério, anteriormente explicada neste relatório, o cálculo do custo do caminho passou a ser feito recorrendo a esta. Isto é, ao invés de ser calculado o somatório das forças de ligação e relação de um user para o outro, passou a usar-se uma função que tem em consideração os pesos de cada uma das forças no cálculo do valor final.

```
/*is_bidirectional(X,Y,Z):- (ligacao(X,Y,F1,F2,_,_),Z is F1+F2;ligacao(Y,X,F1,F2,_,_),Z is F1+F2),!.*/
is_bidirectional(X,Y,Z):- ligacao(X,Y,FL,FR,_,_), funcao_multicriterio(FL,FR,Res), Z is Res.
```

Figura 29 - adaptação do A e Best First com função multicritério*

Conclusões

Resumindo, no contexto do projeto e de forma a encontrar caminhos entre utilizadores baseados em diferentes critérios, o algoritmo A Star foi adaptado de forma a utilizar uma estimativa relevante para o projeto, para o custo total do caminho. Para aproximar esse valor o mais possível da realidade, a definição de uma função que calculasse o peso de cada força no custo final foi essencial.

Os algoritmos Best First , A Star e Primeiro em Profundidade foram também otimizados de forma a ser possível também filtrar o número máximo de ligações de cada caminho de forma a que, quando utilizado, retornasse apenas o primeiro caminho encontrado até N ligações.

De forma a facilitar a pesquisa dos caminhos de um determinado utilizador, uma rede à parte com utilizadores que podem ser atingidos até N ligações foi criada com o predicado net_size_up_to_level, filtrando as ligações entre o utilizador selecionado até ao nível também selecionado.