Instituto Superior de
Engenharia do Porto

Relatório ALGAV

Miguel Gonçalves 1190903

Ruben Rodrigues 1191018

Rui Pinto 1191042

Tomás Limbado 1191106

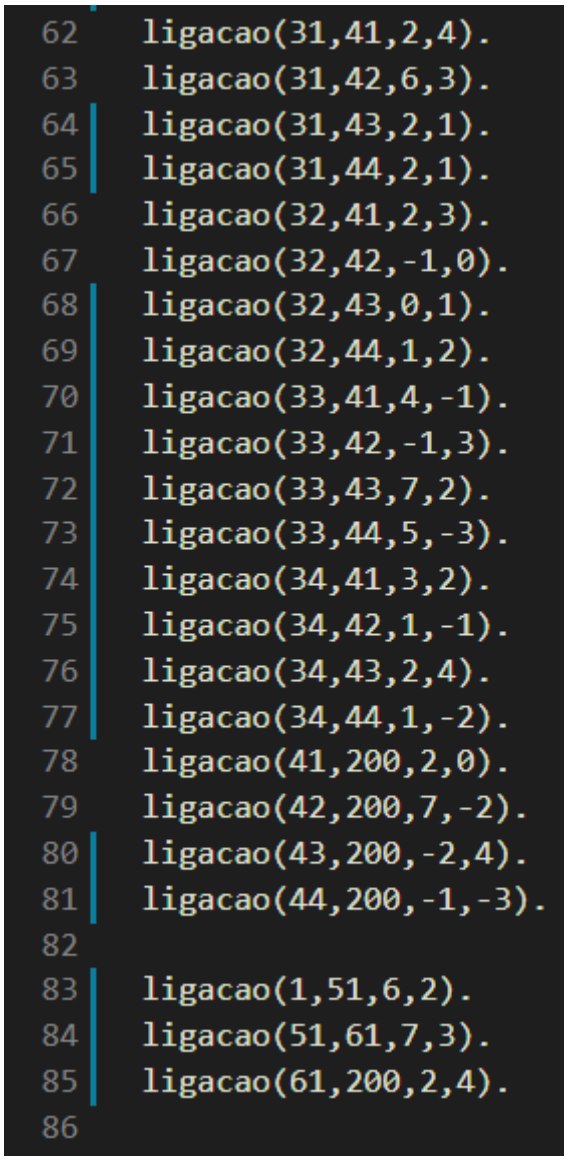
3DJ @ 2021-2022

- **Representação do conhecimento do domínio**

```
1   no(1,ana,[natureza,pintura,musica,sw,porto]).
2   no(11,antonio,[natureza,pintura,carros,futebol,lisboa]).
3   no(12,beatriz,[natureza,musica,carros,porto,moda]).
4   no(13,carlos,[natureza,musica,sw,futebol,coimbra]).
5   no(14,daniel,[natureza,cinema,jogos,sw,moda]).
6   no(21,eduardo,[natureza,cinema,teatro,carros,coimbra]).
7   no(22,isabel,[natureza,musica,porto,lisboa,cinema]).
8   no(23,jose,[natureza,pintura,sw,musica,carros,lisboa]).
9   no(24,luisa,[natureza,cinema,jogos,moda,porto]).
10  no(31,maria,[natureza,pintura,musica,moda,porto]).
11  no(32,anabela,[natureza,cinema,musica,tecnologia,porto]).
12  no(33,andre,[natureza,carros,futebol,coimbra]).
13  no(34,catia,[natureza,musica,cinema,lisboa,moda]).
14  no(41,cesar,[natureza,teatro,tecnologia,futebol,porto]).
15  no(42,diogo,[natureza,futebol,sw,jogos,porto]).
16  no(43,ernesto,[natureza,teatro,carros,porto]).
17  no(44,isaura,[natureza,moda,tecnologia,cinema]).
18  no(200,sara,[natureza,moda,musica,sw,coimbra]).
19
20  no(51,rodolfo,[natureza,musica,sw]).
21  no(61,rita,[moda,tecnologia,cinema]).
22
```

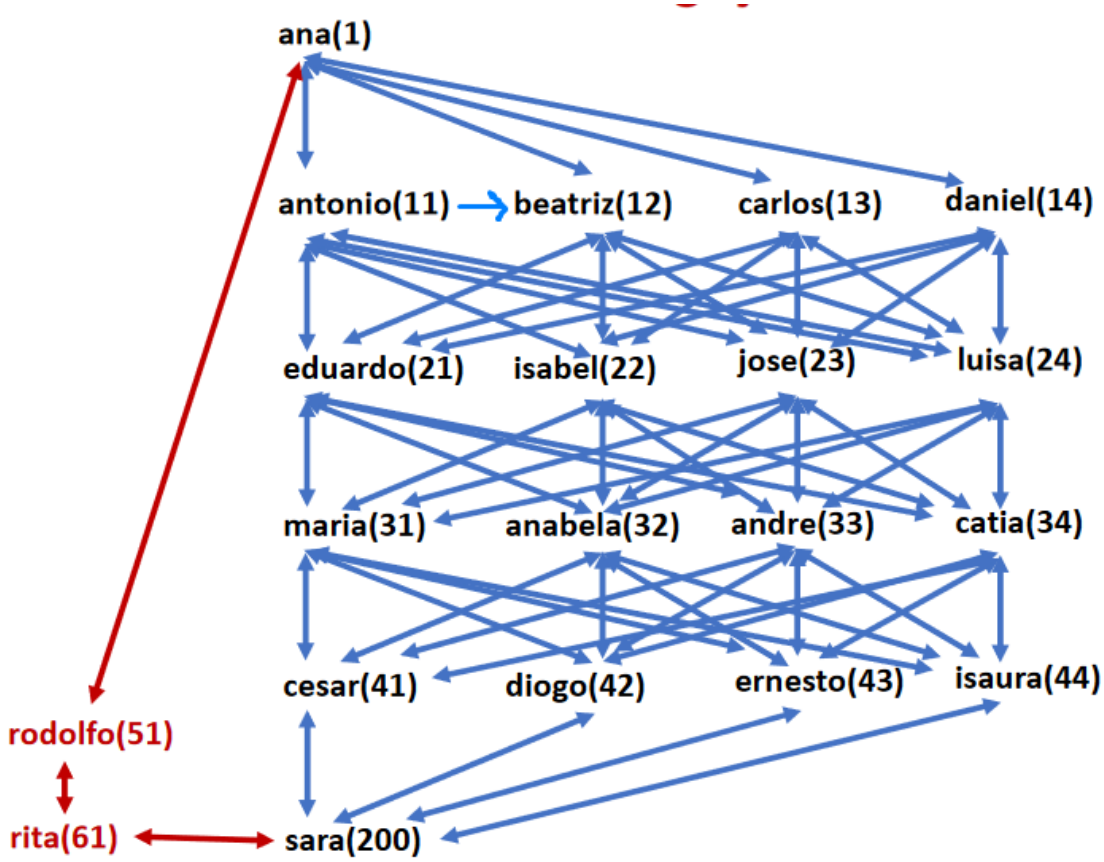
Nodes / Utilizadores (no/3 {id,nome,tags})

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

ligacao(11,12,0,-2).

ligacao(11,21,5,7).
ligacao(11,22,2,-4).
ligacao(11,23,-2,8).
ligacao(11,24,6,0).
ligacao(12,21,4,9).
ligacao(12,22,-3,-8).
ligacao(12,23,2,4).
ligacao(12,24,-2,4).
ligacao(13,21,3,2).
ligacao(13,22,0,-3).
ligacao(13,23,5,9).
ligacao(13,24,-2, 4).
ligacao(14,21,2,6).
ligacao(14,22,6,-3).
ligacao(14,23,7,0).
ligacao(14,24,2,2).
ligacao(21,31,2,1).
ligacao(21,32,-2,3).
ligacao(21,33,3,5).
ligacao(21,34,4,2).
ligacao(22,31,5,-4).
ligacao(22,32,-1,6).
ligacao(22,33,2,1).
ligacao(22,34,2,3).
ligacao(23,31,4,-3).
ligacao(23,32,3,5).
ligacao(23,33,4,1).
ligacao(23,34,-2,-3).
ligacao(24,31,1,-5).
ligacao(24,32,1,0).
ligacao(24,33,3,-1).
ligacao(24,34,-1,5).
```



```
62  ligacao(31,41,2,4).
63  ligacao(31,42,6,3).
64  ligacao(31,43,2,1).
65  ligacao(31,44,2,1).
66  ligacao(32,41,2,3).
67  ligacao(32,42,-1,0).
68  ligacao(32,43,0,1).
69  ligacao(32,44,1,2).
70  ligacao(33,41,4,-1).
71  ligacao(33,42,-1,3).
72  ligacao(33,43,7,2).
73  ligacao(33,44,5,-3).
74  ligacao(34,41,3,2).
75  ligacao(34,42,1,-1).
76  ligacao(34,43,2,4).
77  ligacao(34,44,1,-2).
78  ligacao(41,200,2,0).
79  ligacao(42,200,7,-2).
80  ligacao(43,200,-2,4).
81  ligacao(44,200,-1,-3).
82
83  ligacao(1,51,6,2).
84  ligacao(51,61,7,3).
85  ligacao(61,200,2,4).
86
```

Ligações entre os nodes / utilizadores (ligação/4 {id1,id2,força1,força2})



Representação da rede social

- **Determinação do tamanho da rede de um utilizador até um determinado nível**

```
224
225 % tamanho da rede
226
227 calculatamanho(Origem,0,Tamanho):-!,
228     no(_,Origem,_),
229     Tamanho is 0.
230
231 calculatamanho(Origem,N,Tamanho):-
232     N1 is N-1,
233     amigos_proximos(Origem,L),
234     append([Origem],L,LX),
235     mais_amigos(LX,T1,N1),
236     Tamanho is T1 - 1.
237
238 amigos_proximos(Origem,L):-
239     findall(X,ligacao(Origem,X,_,_),L2),
240     findall(X,ligacao(X,Origem,_,_),L3),
241     append(L2,L3,L).
242
243 mais_amigos(L2,X,0):-
244     length(L2,X),!.
245
246 mais_amigos(L,Tamanho,N):-
247     amigos_dos_amigos(L,L2),
248     N1 is N-1,
249     mais_amigos(L2,Tamanho,N1),!.
250
251 amigos_dos_amigos([],[]):-!.
252
253 amigos_dos_amigos([H|T],LR):-
254     amigos_proximos(H,L),
255     amigos_dos_amigos(T,L2),union(L,L2,LR).
256
257
```

O predicado calculacaminho/3 recebe o id do utilizador e analisar a rede, o N máximo e a variável que vai indicar o tamanho da rede. Invoca o predicado amigos_proximos/2 de modo a obter os jogadores com ligação direta ao utilizador. De seguida inclui-se o utilizador origem na lista gerada e invoca-se o predicado mais_amigos/3 para pesquisar pelo resto da rede.

O predicado amigos_proximos/2, recebendo o id do utilizador a analisar e a lista de amigos próximos a retornar, pesquisa todas ligações que envolvam o utilizador origem e adiciona-os à lista a retornar.

O predicado mais_amigos/3, recebendo a lista de nível 1, a variável de tamanho final da rede e os níveis a analisar, invoca o predicado amigos_dos_amigos/2 para encontrar os amigos próximos dos amigos dos diferentes níveis, invocando-se a si próprio de seguida até os níveis acabarem.

Por fim, o predicado amigos_dos_amigos/2, que recebe a lista de amigos já definidos e a lista de retorno, invoca o predicado amigos_proximos/2 de modo a encontrar os amigos diretos dos utilizadores na lista recebida e une as listas com a lista de retorno.

Exemplo:

```
?- calculatamanho(1,1,R).
R = 5.

?- calculatamanho(1,2,R).
R = 10.

?- calculatamanho(1,3,R).
R = 15.

?- calculatamanho(1,4,R).
R = 19.

?- calculatamanho(1,5,R).
R = 19.
```

- **Obtenção dos utilizadores que tenham em comum X tags**

```
276
277 % xtags em comum
278
279 xtags_comum(X,LTags,LUtz):-
280     todas_combinacoes(X,LTags,LcombXTags),
281     todos_utilizadores(LcombXTags,LUtz).
282
283 todos_utilizadores([],[]).
284 todos_utilizadores([Tags|LcombXTags],[Utz|LUtz]):-utilizador_xcomum(Tags,Utz),
285         todos_utilizadores(LcombXTags,LUtz).
286
287 utilizador_xcomum(LTags,LUtz):-findall(Id,(no(Id,_,L),allMember(LTags,L)),LUtz).
288
289 todas_combinacoes(X,LTags,LcombXTags):-findall(L,combinacao(X,LTags,L),LcombXTags).
290
291 combinacao(0,_,[]):-!.
292 combinacao(X,[Tag|L],[Tag|T]):-X1 is X-1, combinacao(X1,L,T).
293 combinacao(X,[_|L],T):- combinacao(X,L,T).
294
295 allMember([],_):-!.
296 allMember([H|T],L):- member(H,L),allMember(T,L).
297
```

O predicado xtags_comum/3, que recebe o número de tags (X), a lista de tags a ser analisadas e a lista de retorno de users, dá inicio ao algoritmo. Invoca o predicado todas_combinacoes/3 de forma a obter todas as combinações de x tags dentro da lista recebida (predicado indicado na documentação das aulas Teórico-Práticas). Após isso é invocado o predicado todos_utilizadores/2 que retorna todos os jogadores que possuam uma combinação das tags.

O predicado todos_utilizadores/2, que recebe a lista das combinações e a lista de retorno de users, o método invoca o predicado utilizador_xcomum/2 de modo a encontrar os utilizadores que possuam as tags das combinação, e invoca-se a si próprio recursivamente até já ter analisado tudo.

Por fim, o predicado utilizador_xcomum/2, que recebe a combinação a analisar e a lista de retorno de users, pesquisa todos os utilizadores cujas tags se encontram na combinação e adiciona-os à lista de retorno.

Exemplo:

```
?- xtags_comum(2,[natureza,cinema,carros],L).
L = [[14, 21, 22, 24, 32, 34, 44], [11, 12, 21, 23, 33, 43], [21]].
```

- **Sugestão das conexões com outros utilizadores tendo por base as tags e conexões partilhadas até determinado nível**

```
256
257  % sugestoes conexoes por base tags
258
259  determinarLista(Origem,N,L2):-
260      N1 is N-1,
261      amigos_proximos(Origem,LX),
262      append([],LX,LAMIGOS),
263      mais_amigos2(LX,L,N1),deleteList(L,LAMIGOS,L1),
264      deleteList(L1,[Origem],L3),
265      lista(Origem,L3,L2).
266
267  lista(_,[],[]):-!.
268  lista(Origem,[X|L],[X|L1]):-shared_tags(Origem,X),lista(Origem,L,L1),!.
269  lista(Origem,[X|L],L1):-lista(Origem,L,L1).
270
271  mais_amigos2(_,[],0):-!.
272
273  mais_amigos2(L,Tamanho,N):-
274      amigos_dos_amigos(L,L2),
275      N1 is N-1,
276      mais_amigos2(L2,Tamanho2,N1),
277      union(Tamanho2,L2,Tamanho),!.
278
279  deleteList([],_,[]):-!.
280  deleteList([X|L],Z,F):-member(X,Z),!,deleteList(L,Z,F).
281  deleteList([X|L],Z,[X|F]):-deleteList(L,Z,F).
282
283  shared_tags(NodeOrig,Node2):-
284      no(NodeOrig,_,LTOrig),
285      no(Node2,_,LT2),
286      shared_tags1(LTOrig,LT2).
287
288
289  shared_tags1(_,[]):-fail.
290  shared_tags1(LTOrig,[LH2|LT2]):-
291      (member(LH2,LTOrig),!);shared_tags1(LTOrig,LT2).
292
```

O método determinarLista/3 inicia a sugestão. O algoritmo recebe o id do utilizador origem, o nível até onde se quer efetuar a procura e a lista de retorno. Invoca o predicado amigos_proximos/2 (explicado anteriormente) e de seguida invoca o mais_amigos2/3, invocando de seguida o predicado deleteList/3 de modo a apagar os amigos sugeridos

repetidos. Por fim, invoca o predicado lista/3 que determina quais desses amigos têm tags em comum com ele.

O predicado mais_amigos2/3 funciona de forma semelhante ao mais_amigos/3, que foi explicado anteriormente, porém aqui une as listas geradas.

O predicado deleteList/3 recebe duas listas de ids e uma lista de retorno, sendo que a lista de retorno será preenchida com os membros da primeira lista de ids que não estão contidos na segunda lista de ids.

O predicado list/3 filtra a lista de sugestões com base nas tags dos utilizadores origem e sugeridos (através do predicado shared_tags/2).

- ## Determinação de caminhos
  - ### Métodos gerais (DFS)

```
101  % dfs
102
103  dfs(Orig,Dest,Cam):-dfs2(Orig,Dest,[Orig],Cam).
104
105  dfs2(Dest,Dest,LA,Cam):-!,reverse(LA,Cam).
106  dfs2(Act,Dest,LA,Cam):-
107      no(NAct,Act,_),(ligacao(NAct,NX,_,_);ligacao(NX,NAct,_,_)),
108      no(NX,X,_),\+ member(X,LA),dfs2(X,Dest,[X|LA],Cam).
109
```

Método de busca em profundidade (Depth First Search – DFS) de modo a obter todos os caminhos entre dois nodes de um grafo

- **Caminho mais forte**

```
% caminho mais forte

:-dynamic melhor_maisF/2.

plan_maisF(Orig,Dest,LCaminho_maisF):-
    get_time(Ti),
    (melhor_caminho_maisF(Orig,Dest);true),
    retract(melhor_maisF(LCaminho_maisF,_)),
    get_time(Tf),
    T is Tf-Ti,
    write('Tempo de geracao da solucao:'),write(T),nl.

melhor_caminho_maisF(Orig,Dest):-
    assert(melhor_maisF(_,0)),
    dfs(Orig,Dest,LCaminho),
    atualiza_melhor_maisF(LCaminho),
    fail.

atualiza_melhor_maisF(LCaminho):-
    melhor_maisF(_,N),
    calculo_caminho_maisF(LCaminho,C),
    C>N,retract(melhor_maisF(_,_)),
    assert(melhor_maisF(LCaminho,C)).

calculo_caminho_maisF([],0):-!.
calculo_caminho_maisF([_],0):-!.
calculo_caminho_maisF([X|[Y|Lc]],R):-
    % write('\nX:'),write(X),
    % write('\nY:'),write(Y),
    calculo_caminho_maisF([Y|Lc],R1),
    no(X1,X,_),
    no(Y1,Y,_),
    ligacao(X1,Y1,A,B),!,
    R is A + B + R1.
```

O predicado plan_maisF/3, que recebe o nome do jogador inicial, o nome do jogador destino e a lista onde vai ser introduzida a solução, é o predicado chamado de modo a iniciar a solução. Este predicado utiliza o predicado melhor_caminho_maisF/2 que calcula qual é o caminho mais forte entre os dois jogadores. De seguida seleciona o caminho mais forte e apresenta-o.

O predicado melhor_caminho_maisF/2, que recebe o nome do jogador inicial e o nome do jogador destino, inicia a variável dinâmica correspondente ao caminho mais forte e a respetiva força

(melhor_maisF/2) com uma força de 0. De seguida utiliza o dfs/3 para calcular todos os caminhos possíveis entre os dois jogadores introduzidos, enviando então essa lista para o predicado atualiza_melhor_maisF/1 de modo a determinar o melhor caminho. O *fail* está posto para obrigar a voltar atrás e gerar um novo caminho.

O predicado atualiza_melhor_maisF/1, que recebe um caminho, começa por ir ao melhor caminho atual buscar a sua força, de seguida convoca o predicado calculo_caminho_maisF/2 que calcula a força do caminho introduzido, de seguida compara a força calculada com o força do melhor caminho atual e se a calculada for maior o melhor caminho atual é substituído pelo novo caminho.

Por fim, o predicado calculo_caminho_maisF/2 recebe um caminho e calcula o somatório das forças de ligação no sentido da travessia, através da ligação entre dois utilizadores seguidos da lista.

Exemplo:

```
?- plan_maisF(ana,sara,L).
Tempo de geracao da solucao:148.0567011833191
L = [ana, antonio, eduardo, andre, ernesto, sara].
```

- **Caminho mais curto**

```
108
109  % caminho mais curto
110
111  :-dynamic melhor_sol_minlig/2.
112
113  plan_minlig(Orig,Dest,LCaminho_minlig):-
114      get_time(Ti),
115      (melhor_caminho_minlig(Orig,Dest);true),
116      retract(melhor_sol_minlig(LCaminho_minlig,_)),
117      get_time(Tf),
118      T is Tf-Ti,
119      write('Tempo de geracao da solucao:'),write(T),nl.
120
121  melhor_caminho_minlig(Orig,Dest):-
122      asserta(melhor_sol_minlig(_,10000)),
123      dfs(Orig,Dest,LCaminho),
124      atualiza_melhor_minlig(LCaminho),
125      fail.
126
127  atualiza_melhor_minlig(LCaminho):-
128      melhor_sol_minlig(_,N),
129      length(LCaminho,C),
130      C<N,retract(melhor_sol_minlig(_,_)),
131      asserta(melhor_sol_minlig(LCaminho,C)).
132
```

O predicado plan_minlig/3, que recebe o nome do jogador inicial, o nome do jogador destino e a lista onde vai ser introduzida a solução, é o predicado chamado de modo a iniciar a solução. Este predicado utiliza o predicado melhor_caminho_minlig /2 que calcula qual é o caminho mais curto entre os dois jogadores. De seguida seleciona o caminho mais curto e apresenta-o.

O predicado melhor_caminho_minlig /2, que recebe o nome do jogador inicial e o nome do jogador destino, inicia a variável dinâmica correspondente ao caminho mais curto e o respetivo tamanho (melhor_maisF/2) com um tamanho de 0. De seguida utiliza o dfs/3 para calcular todos os caminhos possíveis entre os dois jogadores introduzidos,

enviando então essa lista para o predicado atualiza_melhor_minlig/1 de modo a determinar o melhor caminho. O *fail* está posto para obrigar a voltar atrás e gerar um novo caminho.

O predicado atualiza_melhor_maisF/1, que recebe um caminho, começa por ir ao melhor caminho atual buscar o seu tamanho, de seguida convoca o predicado length/2 que calcula o tamanho do caminho introduzido, de seguida compara o tamanho do novo caminho com o tamanho do melhor caminho atual e se novo tamanho for menor o melhor caminho atual é substituído pelo novo caminho.

Exemplo:

```
?- plan_minlig(ana,sara,L).
Tempo de geracao da solucao:123.98591494560242
L = [ana, rodolfo, rita, sara].
```

○ Caminho mais seguro

```
% caminho mais seguro

:-dynamic melhor_maisSeguro/2.

plan_maisSeguro(Orig,Dest,Limite,LCaminho_maisF):-
    get_time(Ti),
    (melhor_caminho_maisSeguro(Orig,Dest,Limite);true),
    retract(melhor_maisSeguro(LCaminho_maisF,_)),
    get_time(Tf),
    T is Tf-Ti,
    write('Tempo de geracao da solucao:'),write(T),nl.

melhor_caminho_maisSeguro(Orig,Dest,Limite):-
    assert(melhor_maisSeguro(_,0)),
    dfsLim(Orig,Dest,LCaminho,Limite),
    atualiza_melhor_maisSeguro(LCaminho),
    fail.

atualiza_melhor_maisSeguro(LCaminho):-
    melhor_maisSeguro(_,N),
    calculo_caminho_maisF(LCaminho,C),
    C>N,retract(melhor_maisSeguro(_,_)),
    assert(melhor_maisSeguro(LCaminho,C)).


dfsLim(Orig,Dest,Cam,Limite):-
    dfsLim2(Orig,Dest,[Orig],Cam,Limite).

dfsLim2(Dest,Dest,LA,Cam,_):-!,reverse(LA,Cam).
dfsLim2(Act,Dest,LA,Cam,Limite):-
    no(NAct,Act,_),
    ( ligacao(NAct,NX,X1,Y1),(X1 + Y1 > Limite);ligacao(NX,NAct,X2,Y2),(X2 + Y2 > Limite) ),
    no(NX,X,_),
    \+ member(X,LA),
    dfsLim2(X,Dest,[X|LA],Cam,Limite).
```

Para o cálculo do caminho mais seguro abordámos de forma idêntica ao cálculo do caminho mais forte, alterando somente o dfs.

Os predicados plan_maisSeguro/4, melhor_caminho_maisSeguro/3, dfsLim/3 e dfsLim2/4 recebem agora o parâmetro Limite, cujas soma das forças da ligação tem que ser superior a esse limite.

Este novo DFS (dfsLim) apenas tem em consideração ligações cuja soma das forças é superior ao Limite introduzido.

Exemplo:

```
?- plan_maisF(ana,beatriz,L).
Tempo de geracao da solucao:224.847069978714
L = [ana, antonio, beatriz].

?- plan_maisSeguro(ana,beatriz,0,L).
Tempo de geracao da solucao:7.355835914611816
L = [ana, beatriz].
```

Neste exemplo verificamos que apesar do caminho mais forte ser Ana -> Antonio -> Beatriz, o caminho mais seguro é apenas Ana -> Beatriz, pois a ligação Antonio -> Beatriz tem uma força (-2) inferior ao limite imposto (0).

- **Estudo da complexidade do problema da determinação de caminhos**

Nº de camadas intermédias	Nº de nós por camada	Nº de soluções	Tempo para gerar todos os caminhos (unidirecional) (s)
1	1	1	~0
2	2	16	~0
3	3	81	~0
4	4	256	~0

Nº de camadas intermédias	Nº de nós por camada	Nº de soluções	Tempo para gerar o caminho mais curto (s)
1	1	1	~0
2	2	1	~0
3	3	1	~0
4	4	1	109

Nº de camadas intermédias	Nº de nós por camada	Nº de soluções	Tempo para gerar o caminho mais forte (s)
1	1	1	~0
2	2	1	~0
3	3	1	~0
4	4	1	134

Nº de camadas intermédias	Nº de nós por camada	Nº de soluções	Tempo para gerar o caminho mais seguro (s) (limite = 0)
1	1	1	~0
2	2	1	~0
3	3	1	~0
4	4	1	5

- ## Conclusões

A complexidade do algoritmo DFS é O(n) (sendo N a soma do número de nodes com o número de ligações). Sendo que cada algoritmo de seleção do melhor caminho necessita de uma pesquisa dentro do caminho para determinar a força de ligação ou o comprimento do caminho, a complexidade de cada algoritmo é O(n^2).

A complexidade revela-se relevante quando o grafo ultrapassa as 3 camadas intermédias com 3 nós por camadas.