Relatório de ALGAV

Turma 2DL _ Grupo 67
1191430 - Fábio Fernandes
1190447 - Bruno Silva
1210308 - Alejandro Jorge
1190633 - Gonçalo Jordão
1190523 - Diogo Domingues

Data: 28/11/2021

1. Índice

2. Representação do conhecimento do domínio

Base de conhecimento utilizada (ficheiro ALGAV.pl):

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

ligacao(1,11,10,8).
ligacao(1,12,2,6).
ligacao(1,13,-3,-2).
ligacao(1,14,1,-5).
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).
ligacao(31,41,2,4).
ligacao(31,42,6,3).
ligacao(31,43,2,1).
ligacao(31,44,2,1).
ligacao(32,41,2,3).
ligacao(32,42,-1,0).
ligacao(32,43,0,1).
ligacao(32,44,1,2).
ligacao(33,41,4,-1).
ligacao(33,42,-1,3).
ligacao(33,43,7,2).
ligacao(33,44,5,-3).
ligacao(34,41,3,2).
ligacao(34,42,1,-1).
ligacao(34,43,2,4).
ligacao(34,44,1,-2).
ligacao(41,200,2,0).
ligacao(42,200,7,-2).
ligacao(43,200,-2,4).
ligacao(44,200,-1,-3).
ligacao(1,51,6,2).
ligacao(51,61,7,3).
ligacao(61,200,2,4).

Nota: Para o enquadramento em LAPR5, recorremos ao ficheiro FicheiroServerAtualizado.pl, de modo que a informação presente em Base de Dados seja carrega na base de conhecimento.

3. Determinação do tamanho da rede de um utilizador até um determinado nível

Para implementar este requisito a subdivisão em pequenas tarefas era essencial, para isso, decidi subdividir este problema nas seguintes tarefas:

1. Encontrar os vizinhos diretos do utilizador primário;
2. Encontrar os vizinhos diretos dos utilizadores que foram obtidos anteriormente;
3. Linearizar as listas de utilizadores;
4. Repetir o processo até corresponder ao nível pedido;

Tarefa nº1 – Encontrar os vizinhos diretos do utilizador:

```
findall(UserY,ligacao(UserX,UserY,_,_), ListaD)
```

Com este pequeno predicado consigo armazenar numa lista D os vizinhos diretos, ou seja, os vizinhos que tem ligação com o utilizador que lhe é atribuído. O User X, será o utilizador primário, o utilizador Origem que nos foi enviado, através do findall ele irá verificar para todos os utilizadores existentes na base de conhecimentos (User Y) se existe ligação direta com o User X. "_" Significa que não existe interesse nos dados que estão lá armazenados e serão ignorados, poderá estar lá o que se quiser.

Tarefa nº2 – Encontrar os vizinhos diretos dos utilizadores obtidos anteriormente:

```
tamanho_rede_utilizadores_nivelY([UserX|ListaB],[ListaD|ListaC]):- findall(UserY,ligacao(UserX,UserY,_,_), ListaD),
                                                                   tamanho_rede_utilizador_nivelY(ListaB,ListaC).
```

Como explicado anteriormente através de um predicado recursivo consigo verificar para todos os utilizadores fornecidos na primeira lista quais serão os seus vizinhos, isto é, a primeira lista será decomposta da seguinte forma: ([UserX|ListaB]), estou a retirar o primeiro utilizador(UserX) da lista enviada para ir procurar os seus vizinhos diretos, sobrando a cauda da lista que será iterada recursivamente.

Irei colocar esses vizinhos (vizinhos do UserX) na Lista D, a Lista D irá ser o header da nova lista para que seja adicionada recursivamente, de modo que a cada iteração de User X a lista dos vizinhos desse user seja adicionada.

```
tamanho_rede_utilizadores_nivelY([],[]):-!.
```

Como podemos observar por este “print” a condição de paragem do predicado é verificar quando a lista fornecida esteja complemente iterada, ou seja, vazia.

Tarefa nº3 - Linearizar as listas de utilizadores:

```
append(ListaA,ListaB),
sort(ListaB,ListaE),
```

Recorrendo ao append e ao sort a linearização das listas torna-se muito mais fácil, visto que, o append encarrega-se de juntar as diversas listas numa lista só e o sort faz com que os duplicados sejam reduzidos a uma única unidade.

Tarefa nº4 - Repetir o processo até corresponder ao nível pedido

```
NY is Nivel - 1,
```

Através desta pequena cláusula, a cada iteração final do predicado irei diminuir o nível que nos foi enviado, para que quando chegue a zero, significa que fui até ao nível da rede do utilizador pretendido.

4. Obtenção dos utilizadores que tenham em comum X tags

Problema

Descrição: Obter os utilizadores que tenham em comum Xtags sendo X parametrizável. Deve ter em atenção que duas tags sintaticamente diferentes podem ter o mesmo significado semântico (e.g. C# e CSharp).

Predicados para utilizadores objetivo

todas_combinacoes1/5

```
todas_combinacoes1(X,LTags,LcombXTags,ID,ListaUtilizadores):-
no(ID,_,LTags),todas_combinacoes(X,LTags,LcombXTags),utilizadorTagsComuns(LcombXTags,[],ListaUtilizadores,ID).
```

Explicação: Este predicado vai chamar o no com o ID do utilizador logado de modo a obter a sua Lista de Tags, de seguida chama o predicado todas_combinacoes/3 de modo a obter todas as combinações de tags possiveis e por fim chama o predicado utilizadorTagsComuns/4 de modo a obter todos os utilizadores com X tags em comum.

todas_combinacoes/3 e combinacao/3

```
todas_combinacoes(X,LTags,LcombXTags):-findall(L,combinacao(X,LTags,L),LcombXTags).
combinacao(0,_,[]):-!.
combinacao(X,[Tag|L],[Tag|T]):-X1 is X-1, combinacao(X1,L,T).
combinacao(X,[_|L],T):- combinacao(X,L,T).
```

Explicação: Estes predicados encontram todas as combinações de X tags dentro da Lista de Tags do utilizador logado.

utilizadorTagsComuns/4

```
utilizadorTagsComuns([],ListaUtilizadores,L6,_):-!,append([],ListaUtilizadores,L6).
utilizadorTagsComuns([X|L],ListaUtilizadores,L6,ID):-
(no(IdUtilizador,_,ListaTags),IdUtilizador\=ID,not(ligacao(ID,IdUtilizador,_,_)),not(ligacao(IdUtilizador,ID,
_,_)),verificaLista(X,ListaTags),not(member(IdUtilizador,ListaUtilizadores)),List=[IdUtilizador],append(List
aUtilizadores,List,L7),utilizadorTagsComuns([X|L],L7,L6,ID));(utilizadorTagsComuns(L,ListaUtilizadores,L6
,ID)).
```

Explicação: Este predicado vai receber a lista de listas de tags e vai criar uma nova lista de utilizadores tendo em conta as tags em comum entre o utilizador logado e os restantes utilizadores. Primeiro vai buscar a lista de tags dos vários utilizadores e o seu id, se o id for diferente do id do user logado continua (o próprio user não pode ser seu utilizador objetivo),de seguida é chamado o predicado verificaLista/2,e de seguida caso este user ainda não seja membro da lista de utilizadores é adicionado. Quando a lista de lista de Tags se encontrar vazia é chamada a condição de paragem que guarda a lista de utilizadores numa variável L6.

verificaLista/2

```
verificaLista([],_):-true.
verificaLista([X|L],L2):- member(X,L2),verificaLista(L,L2).
verificaLista(_,_):-false.
```

Explicação: Este predicado vai receber uma lista de tags da lista com todas as combinações e outra lista que é a lista das Tags de um utilizador. De seguida vai ver se as tags da lista de tags da lista com todas as combinações existem na lista de tags do utilizador. Caso estas existam todas quer dizer que o user pode ser adicionado à lista de users logo vai ser retornado true,caso contrário é retornado false.

Predicados para tags sinónimas

m1/0

```
m1():-forall(sinonimo(Palavra,Lista),percorrer(Palavra,Lista)).
```

Explicação: Este é o predicado base que vai percorrer todos os elementos da base de conhecimentos dos sinónimos e chama o predicado percorrer/2.

percorrer/2

```
percorrer(Palavra,Lista):forall(no(Id,Nome,ListaUser),percorrerUser(Lista,ListaUser,Id,Nome,Palavra,ListaUser)).
```

Explicação: Este é o predicado que vai percorrer os elementos da base de conhecimento dos nos e vai chamar o percorrerUser/6.

percorrerUser/6

```
percorrerUser(_,[],_,_,_,_):-!.
percorrerUser(Lista,[X|L],Id,Nome,Palavra,ListaUser):member(X,Lista),delete(ListaUser,X,L1),L2=[Palavra],append(L1,L2,L3),retract(no(Id,Nome,_)),asserta(no(Id,Nome,L3));percorrerUser(Lista,L,Id,Nome,Palavra,ListaUser).
```

Explicação: Este predicado vai percorrer a lista de tags do utilizador e vai ver se a tag pertence à lista de tags sinónimas,se isto acontecer será criada uma nova lista onde estão todos os elementos menos o elemento igual que vai ser substituido por um Id da base de conhecimento dos sinonimos.De seguida é feito um retract de modo a remover o no existente e um assert para colocar o no com a lista de tags atualizada.

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

Para implementar este requisito a subdivisão em pequenas tarefas era essencial, para isso, decidi subdividir este problema nas seguintes tarefas:

1. Encontrar os vizinhos diretos do utilizador primário que tenham tags em comum;
2. Encontrar as tags em comum;
3. Linearizar as listas de utilizadores;
4. Receber resultados;

Tarefa nº1 – Encontrar os vizinhos diretos do utilizador primário que tenham tags em comum.

```
findall(L,combination(X,LTags,L),LcombXTags),
```

Com este pequeno predicado consigo armazenar numa lista LcombXTags os vizinhos diretos, ou seja, os vizinhos que tem ligação com o utilizador que lhe é atribuído e que tenham tags em comum. O User X, será o utilizador primário, o utilizador Origem que nos foi enviado, através do findall ele irá verificar para todos os utilizadores existentes na base de conhecimentos (User Y) se existe ligação direta com o User X. "_"

Tarefa nº2 – Encontrar as tags em comum.

```
verificar_x(LcombXTags,LJogadores).
```

Aqui, vamos a verificar as tags em comum e que estão a um determinado nível.

Tarefa nº3 – Linearizar as listas de utilizadores.

```
sort(LJogadores1,LJogadores).
```

Recorrendo ao sort a linearização das listas torna-se muito mais fácil, visto que o sort faz com que os duplicados sejam reduzidos a uma única unidade.

Tarefa nº4 – Receber resultados.

```
?- todas_combinacoes(3,[naturaleza,pintura,musica,sw,porto],L).
L = [ana, jose, maria].
```

Uma vez, que já temos o programa tentaremos com o nível 3 e umas tags podem ser natureza, pintura, musica, sw e porto. Mais tarde, meteremos todo numa lista e como vemos de essa lista temos o resultado ana, jose e maria.

6. Determinação dos caminhos:

6.1 Caminho mais forte

Implementação do Algoritmo para o caminho mais forte bidirecional:

```
:- dynamic melhor_sol_maisForteLig/2.
:- dynamic contagem_solucoes/1.

plan_maisForteLig(Orig, Dest, LCaminho_maisForteLig) :-
    get_time(Ti),                          % retorna o tempo atual como TimeStamp (inicia contagem de tempo)
    (melhor_caminho_maisForteLig(Orig, Dest) ; true),          % chamada ao metodo melhor_caminho_strongerLig, sendo que caso este dê false, como temos "; true", será sempre feito o que vem a seguir
    retract(melhor_sol_maisForteLig(LCaminho_maisForteLig, Forca)),  % remocao da melhor solucao
    retract(contagem_solucoes(NSol)),          % remocao da contagem do numero de solucoes
    get_time(Tf),                          % retorna o tempo atual como TimeStamp (finaliza contagem de tempo)
    T is Tf-Ti,                            % determinar tempo decorrido
    write('Numero de solucoes obtidas: '), write(NSol), nl,
    write('Tempo de geracao da solucao obtida: '), write(T), write(' segundos'), nl,
    write('Forca de ligacao da solucao obtida: '), write(Forca), nl,
    write('Caminho mais forte obtido:'), write(LCaminho_maisForteLig), nl.
```

```
melhor_caminho_maisForteLig(Orig, Dest) :-
    asserta(melhor_sol_maisForteLig(_, -1000)),          % guardar informacao da melhor solucao
    asserta(contagem_solucoes(0)),                       % guardar contagem de solucoes
    dfsMaisForte(Orig, Dest, LCaminho, Forca),
    atualiza_melhor_maisFortelig(LCaminho, Forca),
    fail.                                                % obrigar a voltar atras e gerar um novo caminho (substitui desse modo o findall,
                                                         % porque garante que todas as solucoes serao geradas)


atualiza_melhor_maisFortelig(LCaminho, Forca) :-
    retract(contagem_solucoes(NS)),                      % remocao da contagem de solucoes
    NS1 is NS + 1,                                       % incremento em 1 do numero de solucoes
    asserta(contagem_solucoes(NS1)),                     % guardar número de solucoes atualizado
    melhor_sol_maisForteLig(_, F),
    Forca > F, retract(melhor_sol_maisForteLig(_, _)),   % caso o valor seja superior limpamos o caminho mais forte ate ao momento
    asserta(melhor_sol_maisForteLig(LCaminho, Forca)).   % bem como a sua forca e colocamos o novo (superior)
```

```
dfsMaisForte(Orig, Dest, Cam, Forca) :- dfsMaisForte2(Orig, Dest, [Orig], Cam, Forca).

dfsMaisForte2(Dest, Dest, LA, Cam, 0) :- !, reverse(LA, Cam).

dfsMaisForte2(Act, Dest, LA, Cam, Forca) :-
    no(NAct, Act, _),                          % se existir no com nome que se encontra em Act, prossegue
    ( ligacao(NAct, NX, F1, F2); ligacao(NX, NAct, F2, F1) ),      % se existir ligacao NAct->NX ou NX -> NAct, prossegue
    no(NX, X, _),                              % se existir no ao qual Act está conectado, prossegue
    \+ member(X, LA),                          % se X nao estiver presente em LA, prossegue
    dfsMaisForte2(X, Dest, [X|LA], Cam, FX),   % e volta a ser chamado o metodo dfsMaisForte2, colocando X como cabeça de LA
    Forca is (FX + F1 + F2).                   % Forca passara a ter o valor de FX que vem de tras, somado com as forcas de ligacao F1 e F2
```

Implementação do Algoritmo para o caminho mais forte unidirecional:

```
:- dynamic melhor_sol_maisForteUniLig/2.
:- dynamic contagem_solucoes_uni/1.

plan_maisForteUniLig(Orig, Dest, LCaminho_maisForteLig) :-
    get_time(Ti),                              % guardar informacao da melhor solucao
    (melhor_caminho_maisForteUniLig(Orig, Dest) ; true),   % chamada ao metodo melhor_caminho_strongerLig, sendo que caso este dê false, como temos "; true", será sempre feito o que vem a seguir
    retract(melhor_sol_maisForteUniLig(LCaminho_maisForteLig, Forca)),   % remocao da melhor solucao
    retract(contagem_solucoes_uni(NSol)),      % remocao da contagem do numero de solucoes
    get_time(Tf),                              % retorna o tempo atual como TimeStamp (finaliza contagem de tempo)
```

```
    T is Tf-Ti,                                % determinar tempo decorrido
    write('Numero de solucoes obtidas: '), write(NSol), nl,
    write('Tempo de geracao da solucao obtida: '), write(T), write(' segundos'), nl,
    write('Forca de ligacao da solucao obtida: '), write(Forca), nl,
    write('Caminho mais forte obtido:'), write(LCaminho_maisForteLig), nl.


melhor_caminho_maisForteUniLig(Orig, Dest) :-
    asserta(melhor_sol_maisForteUniLig(_, -1000)),          % guardar informacao da melhor solucao (inicalmente a força será -10000)
    asserta(contagem_solucoes_uni(0)),          % guardar contagem de solucoes (inicalmente terá o valor de 0)
    dfsMaisForteUni(Orig, Dest, LCaminho, Forca),
    atualiza_melhor_maisForteUniLig(LCaminho, Forca),
    fail.                                       % obrigar a voltar atras e gerar um novo caminho (substitui desse modo o findall,
                                                % porque garante que todas as solucoes serao geradas)


atualiza_melhor_maisForteUniLig(LCaminho, Forca) :-
    retract(contagem_solucoes_uni(NS)),         % remocao da contagem de solucoes
    NS1 is NS + 1,                              % incremento em 1 do numero de solucoes
    asserta(contagem_solucoes_uni(NS1)),        % guardar número de solucoes atualizado
    melhor_sol_maisForteUniLig(_, F),
    Forca > F, retract(melhor_sol_maisForteUniLig(_, _)),   % caso o valor seja superior limpamos o caminho mais forte ate ao momento
    asserta(melhor_sol_maisForteUniLig(LCaminho, Forca)).   % bem como a sua forca e colocamos o novo (superior)
```

```
dfsMaisForteUni(Orig, Dest, Cam, Forca) :- dfsMaisForteUni2(Orig, Dest, [Orig], Cam, Forca).

dfsMaisForteUni2(Dest, Dest, LA, Cam, 0) :- !, reverse(LA, Cam).

dfsMaisForteUni2(Act, Dest, LA, Cam, Forca) :-
    no(NAct, Act, _),                                   % se existir no com o nome que se encontra em Act, prossegue
    ( ligacao(NAct, NX, F1, _); ligacao(NX, NAct, _, F1) ),     % se existir ligacao NAct->NX ou NX -> NAct, prossegue
    no(NX, X, _),                                       % se existir no ao qual Act está conectado, prossegue
    \+ member(X, LA),                                   % se X nao estiver presente em LA, prossegue
    dfsMaisForteUni2(X, Dest, [X|LA], Cam, FX),         % e volta a ser chamado o metodo dfsMaisForte2, colocando X como cabeça de LA
    Forca is (FX + F1).                                 % Forca passara a ter o valor de FX que vem de tras, somado com a
```

6.2 Caminho mais curto

Problema:

Descrição: Determinar o caminho mais curto.

Predicados

Caminho mais Curto sem findall

```
:-dynamic melhor_sol_minlig/2.
dfs(Orig,Dest,Cam):-dfs2(Orig,Dest,[Orig],Cam).
dfs2(Dest,Dest,LA,Cam):-!,reverse(LA,Cam).
dfs2(Act,Dest,LA,Cam):-no(NAct,Act,_),(ligacao(NAct,NX,_,_);ligacao(NX,NAct,_,_)),
   no(NX,X,_),\+ member(X,LA),dfs2(X,Dest,[X|LA],Cam).

plan_minlig(Orig,Dest,LCaminho_minlig):-
                    get_time(Ti),
                    (melhor_caminho_minlig(Orig,Dest);true),
                    retract(melhor_sol_minlig(LCaminho_minlig,_)),
                    get_time(Tf),
                    T is Tf-Ti,
                    write('Tempo de geracao da solucao:'),write(T),nl.

melhor_caminho_minlig(Orig,Dest):-
                    asserta(melhor_sol_minlig(_,10000)),
                    dfs(Orig,Dest,LCaminho),
                    atualiza_melhor_minlig(LCaminho),
                    fail.

atualiza_melhor_minlig(LCaminho):-
                    melhor_sol_minlig(_,N),
                    length(LCaminho,C),
```

```
C<N,retract(melhor_sol_minlig(_,_)),
asserta(melhor_sol_minlig(LCaminho,C)).
```

Explicação: Primeiramente é usado o plan_minlig/3 que vai contar o tempo de execução do algoritmo e vai chamar o melhor_caminho_minlig/2, este por sua vez vai encontrar o caminho mais curto e para isso vai utilizar o dfs/3 para fazer a busca em profundidade do caminho atualizando o caminho se este for mais curto que o anteriormente encontrado.

Caminho mais curto com findall

```
caminho_mais_curto(X,Y,[],CaminhoMaisCurto):-
   get_time(Ti),
   allcon(X,Y,R),encontrar_caminho(R,CaminhoMaisCurto),
   get_time(Tf),
   T is Tf-Ti,
   write('Tempo de geracao da solucao:'),write(T),nl.

encontrar_caminho([X|L],CaminhoMaisCurtoFinal):-encontrar_caminho2(L,X,CaminhoMaisCurtoFinal).
encontrar_caminho2([],CaminhoMaisCurto,CaminhoMaisCurtoFinal):-
!,append([],CaminhoMaisCurto,CaminhoMaisCurtoFinal).
encontrar_caminho2([X|L],CaminhoMaisCurto,CaminhoMaisCurtoFinal):-
length(X,Result),length(CaminhoMaisCurto,Result2),Result<Result2,
encontrar_caminho2(L,X,CaminhoMaisCurtoFinal);encontrar_caminho2(L,CaminhoMaisCurto,Caminho
MaisCurtoFinal).
allcon(X,Y,R):-findall(R1,dfs(X,Y,R1),R).
```

Explicação: Primeiramente são encontrados todos os caminhos através do findall e de seguida é encontrado o caminho mais pequeno.

6.3 Caminho mais seguro

Problema

- **Descrição:** Determinar o **caminho mais seguro** (garante que **não há uma força de ligação inferior a x** considerando as forças nos dois sentidos da ligação) para determinado utilizador.

Predicados

Caminho Mais Seguro Só No Sentido Da Travessia

Unidirecional

```
mais_seguro_uni(Orig, Dest, CaminhoMaisSeguro, M) :-
        get_time(Ti),
        (melhor_caminho_mais_seguro_uni(Orig, Dest,M) ; true),
        retract(melhor_sol_mais_segura(CaminhoMaisSeguro, Forca)),
        get_time(Tf),
        T is Tf-Ti,
        write('Tempo de geracao da solucao obtida: '), write(T), write(' segundos'), nl,
        write('Forca de ligacao da solucao obtida: '), write(Forca), nl,
        write('Caminho mais seguro obtido: '), write(CaminhoMaisSeguro), nl,!.

melhor_caminho_mais_seguro_uni(Orig, Dest,M) :-
        asserta(melhor_sol_mais_segura(_, -1000)),
        dfsMaisSeguroUni(Orig, Dest, LCaminho, Forca,M),
        atualiza_melhor_mais_seguro(LCaminho, Forca),
        fail.

atualiza_melhor_mais_seguro(LCaminho, Forca) :-
        melhor_sol_mais_segura(_, F),
        Forca > F, retract(melhor_sol_mais_segura(_,_)),
        asserta(melhor_sol_mais_segura(LCaminho, Forca)).

dfsMaisSeguroUni(Orig, Dest, Cam, Forca, M) :- dfsMaisSeguroUni2(Orig, Dest, [Orig], Cam, Forca, M).

dfsMaisSeguroUni2(Dest, Dest, LA, Cam, 0, _) :- !, reverse(LA, Cam).

dfsMaisSeguroUni2(Act, Dest, LA, Cam, Forca, M) :-
        no(NAct, Act, _),
        ligacao(NAct, NX, F1, _),
        F1 >= M,
        no(NX, X, _),
        \+ member(X, LA),
        dfsMaisSeguroUni2(X, Dest, [X|LA], Cam, FX, M),
        Forca is (FX + F1).
```

- **Explicação:** Vão sendo encontrados os caminhos seguros, ou seja, caminhos em que não existe uma ligação inferior a `M` (`ligacao(NAct, NX, F1, _),F1 >= M`) e vai se calculando ao mesmo tempo a sua força total (`Forca is (FX + F1)`). Sempre que se encontra um caminho seguro (`dfsMaisSeguroUni(Orig, Dest, LCaminho, Forca,M)`) vai se verificar se esse caminho é mais seguro que o atual mais seguro e caso seja verdade é atualizado o caminho mais seguro para esse (`melhor_sol_mais_segura(_, F),Forca > F, retract(melhor_sol_mais_segura(_,_)), asserta(melhor_sol_mais_segura(LCaminho, Forca)).`). No final teremos assim o caminho mais seguro e sua respetiva força total.
- **Exemplo:**

```
?- mais_seguro_uni('ana','catia',L,0).
Tempo de geracao da solucao obtida: 0.0 segundos
Forca de ligacao da solucao obtida: 19
Caminho mais seguro obtido: [ana,antonio,eduardo,catia]
```

- **Estudo da complexidade do problema da determinação dos caminhos:**

Nº de camadas intermédias (sem nó 1 e 200)	Nº de nós por camada	Tempo para gerar a solução com menor nº de ligações
1	1	0.0s
2	2	0.0s
3	3	0.0s
4	4	0.0s

- ***Nota:*** Foi considerada a ligação mínima de 0.

Bidirecional

```
mais_seguro_bi(Orig, Dest, CaminhoMaisSeguro, M) :-
        get_time(Ti),
        (melhor_caminho_mais_seguro_bi(Orig, Dest,M) ; true),
        retract(melhor_sol_mais_segura(CaminhoMaisSeguro, Forca)),
        get_time(Tf),
        T is Tf-Ti,
        write('Tempo de geracao da solucao obtida: '), write(T), write(' segundos'), nl,
        write('Forca de ligacao da solucao obtida: '), write(Forca), nl,
        write('Caminho mais seguro obtido: '), write(CaminhoMaisSeguro), nl,!.

melhor_caminho_mais_seguro_bi(Orig, Dest,M) :-
        asserta(melhor_sol_mais_segura(_, -1000)),
        dfsMaisSeguroBi(Orig, Dest, LCaminho, Forca,M),
        atualiza_melhor_mais_seguro(LCaminho, Forca),
        fail.

dfsMaisSeguroBi(Orig, Dest, Cam, Forca, M) :- dfsMaisSeguroBi2(Orig, Dest, [Orig], Cam, Forca, M).

dfsMaisSeguroBi2(Dest, Dest, LA, Cam, 0, _) :- !, reverse(LA, Cam).

dfsMaisSeguroBi2(Act, Dest, LA, Cam, Forca, M) :-
        no(NAct, Act, _),
        (ligacao(NAct, NX, F1, _);ligacao(NX, NAct, F1, _)),
        F1 >= M,
        no(NX, X, _),
        \+ member(X, LA),
        dfsMaisSeguroBi2(X, Dest, [X|LA], Cam, FX, M),
        Forca is (FX + F1).
```

- **Explicação:** A diferença do Bidirecional para o Unidirecional anterior é que neste são consideradas as ligações em ambos os sentidos e para isso é preciso acrescentar uma disjunção (`(ligacao(NAct, NX, F1, _);ligacao(NX, NAct, F1, _))`) no predicado `dfsMaisSeguroBi2` .
- **Exemplo:**

```
?- mais_seguro_bi('ana','catia',L,0).
Tempo de geracao da solucao obtida: 2.4219748973846436 segundos
Forca de ligacao da solucao obtida: 81
Caminho mais seguro obtido: [ana,rodolfo,rita,sara,diogo,maria,isabel,daniel,jose,carlos,eduardo,antonio,luisa,anabela,cesar,andre,ernesto,catia]
```

- **Estudo da complexidade do problema da determinação dos caminhos:**

Nº de camadas intermédias (sem nó 1 e 200)	Nº de nós por camada	Tempo para gerar a solução com menor nº de ligações
1	1	0.0s
2	2	0.0s
3	3	0.002s
4	4	9.0s

- ***Nota:*** Foi considerada a ligação mínima de 0.

Caminho Mais seguro considerando a soma das duas forças em cada ramo da travessia e garantindo que em nenhuma delas temos uma força abaixo de um dado limiar

Unidirecional

```
mais_seguro_uni_2(Orig, Dest, CaminhoMaisSeguro, M) :-
        get_time(Ti),
        (melhor_caminho_mais_seguro_uni_2(Orig, Dest,M) ; true),
        retract(melhor_sol_mais_segura(CaminhoMaisSeguro, Forca)),
        get_time(Tf),
        T is Tf-Ti,
        write('Tempo de geracao da solucao obtida: '), write(T), write(' segundos'), nl,
        write('Forca de ligacao da solucao obtida: '), write(Forca), nl,
        write('Caminho mais seguro obtido: '), write(CaminhoMaisSeguro), nl,!.

melhor_caminho_mais_seguro_uni_2(Orig, Dest,M) :-
        asserta(melhor_sol_mais_segura(_, -1000)),
        dfsMaisSeguroUni_2(Orig, Dest, LCaminho, Forca,M),
        atualiza_melhor_mais_seguro(LCaminho, Forca),
        fail.

dfsMaisSeguroUni_2(Orig, Dest, Cam, Forca, M) :- dfsMaisSeguroUni2_2(Orig, Dest, [Orig], Cam, Forca, M).

dfsMaisSeguroUni2_2(Dest, Dest, LA, Cam, 0, _) :- !, reverse(LA, Cam).

dfsMaisSeguroUni2_2(Act, Dest, LA, Cam, Forca, M) :-
        no(NAct, Act, _),
        ligacao(NAct, NX, F1, F2),
        F1 >= M,
        F2 >= M,
        no(NX, X, _),
        \+ member(X, LA),
        dfsMaisSeguroUni2_2(X, Dest, [X|LA], Cam, FX, M),
        Forca is (FX + F1 + F2).
```

- **Explicação:** Vão sendo encontrados os caminhos seguros, ou seja, caminhos em que não existe uma ligação inferior a `M` em ambos os sentidos (`ligacao(NAct, NX, F1, F2),F1 >= M,F2 >= M`) e vai se calculando ao mesmo tempo a sua força total (`Forca is (FX + F1 + F2)`). Sempre que se encontra um caminho seguro (`dfsMaisSeguroUni_2(Orig, Dest, LCaminho, Forca,M)`) vai se verificar se esse caminho é mais seguro que o atual mais seguro e caso seja verdade é atualizado o caminho mais seguro para esse (`melhor_sol_mais_segura(_, F),Forca > F, retract(melhor_sol_mais_segura(_,_)), asserta(melhor_sol_mais_segura(LCaminho, Forca))` .). No final teremos assim o caminho mais seguro e sua respetiva força total.
- **Exemplo:**

```
?- mais_seguro_uni_2('ana','catia',L,0).
Tempo de geracao da solucao obtida: 0.0 segundos
Forca de ligacao da solucao obtida: 36
Caminho mais seguro obtido: [ana,antonio,eduardo,catia]
```

- **Estudo da complexidade do problema da determinação dos caminhos:**

Nº de camadas intermédias (sem nó 1 e 200)	Nº de nós por camada	Tempo para gerar a solução com menor nº de ligações
1	1	0.0s
2	2	0.0s
3	3	0.0s
4	4	0.0s

- ***Nota:*** Foi considerada a ligação mínima de 0.

Bidirecional

```
mais_seguro_bi_2(Orig, Dest, CaminhoMaisSeguro, M) :-
        get_time(Ti),
        (melhor_caminho_mais_seguro_bi_2(Orig, Dest,M) ; true),
        retract(melhor_sol_mais_segura(CaminhoMaisSeguro, Forca)),
        get_time(Tf),
        T is Tf-Ti,
        write('Tempo de geracao da solucao obtida: '), write(T), write(' segundos'), nl,
        write('Forca de ligacao da solucao obtida: '), write(Forca), nl,
        write('Caminho mais seguro obtido: '), write(CaminhoMaisSeguro), nl,!.

melhor_caminho_mais_seguro_bi_2(Orig, Dest,M) :-
        asserta(melhor_sol_mais_segura(_, -1000)),
        dfsMaisSeguroBi_2(Orig, Dest, LCaminho, Forca,M),
        atualiza_melhor_mais_seguro(LCaminho, Forca),
        fail.

dfsMaisSeguroBi_2(Orig, Dest, Cam, Forca, M) :- dfsMaisSeguroBi2_2(Orig, Dest, [Orig], Cam, Forca, M).

dfsMaisSeguroBi2_2(Dest, Dest, LA, Cam, 0, _) :- !, reverse(LA, Cam).

dfsMaisSeguroBi2_2(Act, Dest, LA, Cam, Forca, M) :-
        no(NAct, Act, _),
        (ligacao(NAct, NX, F1, F2);ligacao(NX,NAct, F1, F2)),
        F1 >= M,
        F2 >= M,
        no(NX, X, _),
        \+ member(X, LA),
        dfsMaisSeguroBi2_2(X, Dest, [X|LA], Cam, FX, M),
        Forca is (FX + F1 + F2).
```

- **Explicação**: A diferença do Bidirecional para o Unidirecional anterior é que neste são consideradas as ligações em ambos os sentidos e para isso é preciso acrescentar uma disjunção (`((ligacao(NAct, NX, F1, F2);ligacao(NX,NAct, F1, F2))`) no predicado `dfsMaisSeguroBi2_2` .
- **Exemplo:**

```
?- mais_seguro_bi_2('ana','catia',L,0).
Tempo de geracao da solucao obtida: 0.023988008499145508 segundos
Forca de ligacao da solucao obtida: 98
Caminho mais seguro obtido: [ana,rodolfo,rita,sara,cesar,maria,isaura,anabela,jose,daniel,luisa,antonio,eduardo,andre,ernesto,catia]
```

- **Estudo da complexidade do problema da determinação dos caminhos:**

Nº de camadas intermédias (sem nó 1 e 200)	Nº de nós por camada	Tempo para gerar a solução com menor nº de ligações
1	1	0.0s
2	2	0.0s
3	3	0.0s
4	4	0.08s

- ***Nota:*** Foi considerada a ligação mínima de 0.

Respostas às Perguntas

- 1. O nº de soluções dependerá só do nº de nós?

> **R:** Não, dependerá também do **número de ligações**. Por exemplo, se tivermos **4 nós com 1 ligação cada** teremos apenas **1 caminho** e se tivermos **1 nó com 5 ligações** teremos **5 caminhos**.

- 2. O nº de soluções dependerá só do nº de ligações?

> **R:** Não, dependerá também do **número de nós**. Por exemplo, se tivermos **1 nó com 2 ligações** teremos **2 caminhos** e se tivermos **2 nós com 2 ligações** teremos **4 caminhos**.

Ficheiro Base de Conhecimentos

- O Ficheiro da Base de Conhecimentos relativo ao **Determinar O Caminho Mais Seguro** é o Caminho_Mais_Seguro_Sprint_B_1190633

7. Estudo da complexidade do problema da determinação dos caminhos

Determinar o caminho mais forte entre dois utilizadores (Bidirecional)

Nº de camadas intermédias (sem nó 1 e 200)	Nº de nós por camada	Nº de Soluções	Tempo para gerar a solução com menor nº de ligações
1	1	1	0.0000 s
2	2	2	0.0000 s
3	3	891	0.0250 s
4	4	8184064	327.6016 s

Determinar o caminho mais forte entre dois utilizadores (Unidirecional)

Nº de camadas intermédias (sem nó 1 e 200)	Nº de nós por camada	Nº de Soluções	Tempo para gerar a solução com menor nº de ligações
1	1	1	0.0000 s
2	2	2	0.0000 s
3	3	891	0.0139 s
4	4	8184064	312.5943 s

Para o caso da implementação recorrendo ao “findall”, obviamente que ao longo que aumentamos o número de camadas intermédias e o número de nós por camada, tanto o número de soluções como o tempo de geração de solução irá aumentar, sendo que o tempo de geração de solução terá um aumento significativo em relação às implementações realizadas sem recorrer ao “findall”, uma vez que com este, obteremos todos os caminhos possíveis em primeiro lugar e só depois procuraremos pela solução mais forte. É importante ter em conta que, recorrendo ao “findall”, para a 4ª linha desta tabela obtemos o erro de “Stack Limit Excedeed”, uma vez que os resultados no “findall” são imensos.

Questão: Que complexidade acha que tem o problema que estudou (notação O)?

Resposta: O(n^n).

Valorização:

Questão: O nº de soluções dependerá só do nº de nós?

Resposta: Não, o número de soluções depende do nº de nós, mas não apenas, pois o número de soluções depende do **número de ligações por nó**. Conclui-se então que o nº de soluções depende do nº de nós, mas não apenas.

Questão: O nº de soluções dependerá só do nº de ligações?

Resposta: Não, o número de soluções depende do nº de ligações, mas não apenas, pois tal como já referido na resposta à pergunta anterior, o número de soluções depende do **número de ligações por nó**. Conclui-se então que o nº de soluções depende do nº de ligações, mas não apenas.

Resposta: Sim, tal como podemos ver pelo estudo realizado nas tabelas acima representadas.

Determinar o caminho mais curto entre dois utilizadores (sem findall)

Nº de camadas intermédias (sem nó 1 e 200)	Nº de nós por camada	Nº de soluções	Tempo para gerar a solução com menor nº de ligações
1	1	1	0.0069s
2	2	2	0.0060s
3	3	891	0.0130s
4	4	8184064	100.7441s

Determinar o caminho mais curto entre dois utilizadores (com findall)

Nº de camadas intermédias (sem nó 1 e 200)	Nº de nós por camada	Nº de soluções	Tempo para gerar a solução com menor nº de ligações
1	1	1	0.0000 s
2	2	2	0.0000 s
3	3	891	0.0090s
4	4	----------	----------

Questão: Que complexidade acha que tem o problema que estudou (notação O)?

Resposta: O(n^n).

Valorização:

Questão: O nº de soluções dependerá só do nº de nós?

Resposta: Não, o número de soluções depende do nº de nós, mas não apenas, pois o número de soluções depende do **número de ligações por nó**. Conclui-se então que o nº de soluções depende do nº de nós, mas não apenas.

Questão: O nº de soluções dependerá só do nº de ligações?

Resposta: Não, o número de soluções depende do nº de ligações, mas não apenas, pois tal como já referido na resposta à pergunta anterior, o número de soluções depende do **número de ligações por nó**. Conclui-se então que o nº de soluções depende do nº de ligações, mas não apenas.

Resposta: Sim, tal como podemos ver pelo estudo realizado nas tabelas acima representadas.

Base de conhecimento utilizada para o estudo em "Nº de camadas intermédias (sem nó 1 e 200)" com o valor de 1 e em "Nº de nós por camada" com o valor de 1, bem como o seu resultado:

```
no(1,ana,[natureza,pintura,musica,sw,porto]).
no(11,antonio,[natureza,pintura,carros,futebol,lisboa]).
no(200,sara,[natureza,moda,musica,sw,coimbra]).

ligacao(1,11,10,8).
ligacao(1,12,2,6).
ligacao(11,200,10,6).
```

Caminho mais forte:

```
?- plan_maisForteLig(ana,sara,L).
Numero de solucoes obtidas: 1
Tempo de geracao da solucao obtida: 0.0 segundos
Forca de ligacao da solucao obtida: 24
Caminho mais forte obtido:[ana,antonio,sara]
L = [ana, antonio, sara].

?- plan_maisForteUniLig(ana,sara,L).
Numero de solucoes obtidas: 1
Tempo de geracao da solucao obtida: 0.0 segundos
Forca de ligacao da solucao obtida: 12
Caminho mais forte obtido:[ana,antonio,sara]
L = [ana, antonio, sara].
```

Caminho mais curto:

```
1 ?- plan_minlig(ana,sara,L).
Tempo de geracao da solucao:0.006939888000488281
L = [ana, antonio, sara].

2 ?- caminho_mais_curto(ana,sara,_,L).
Tempo de geracao da solucao:0.0
L = [ana, antonio, sara].
```

Base de conhecimento utilizada para o estudo em "Nº de camadas intermédias (sem nó 1 e 200)" com o valor de 2 e em "Nº de nós por camada" com o valor de 2, bem como o seu resultado:

```
no(1,ana,[natureza,pintura,musica,sw,porto]).
no(11,antonio,[natureza,pintura,carros,futebol,lisboa]).
no(12,beatriz,[natureza,musica,carros,porto,moda]).

no(21,eduardo,[natureza,cinema,teatro,carros,coimbra]).
no(22,isabel,[natureza,musica,porto,lisboa,cinema]).

no(200,sara,[natureza,moda,musica,sw,coimbra]).

ligacao(1,11,10,8).
ligacao(1,12,2,6).
ligacao(11,21,5,7).
ligacao(11,22,2,-4).
ligacao(21,200,3,4).
ligacao(22,200,3,5).
```

Caminho mais forte:

```
?- plan_maisForteUniLig(ana,sara,L).
Numero de solucoes obtidas: 2
Tempo de geracao da solucao obtida: 0.0 segundos
Forca de ligacao da solucao obtida: 17
Caminho mais forte obtido:[ana,antonio,eduardo,sara]
L = [ana, antonio, eduardo, sara].

?- plan_maisForteLig(ana,sara,L).
Numero de solucoes obtidas: 2
Tempo de geracao da solucao obtida: 0.0 segundos
Forca de ligacao da solucao obtida: 36
Caminho mais forte obtido:[ana,antonio,eduardo,sara]
L = [ana, antonio, eduardo, sara].
```

Caminho mais curto:

```
1 ?- plan_minlig(ana,sara,L).
Tempo de geracao da solucao:0.006000995635986328
L = [ana, antonio, eduardo, sara].

2 ?- caminho_mais_curto(ana,sara,_,L).
Tempo de geracao da solucao:0.0
L = [ana, antonio, eduardo, sara].
```

Base de conhecimento utilizada para o estudo em "Nº de camadas intermédias (sem nó 1 e 200)" com o valor de 3 e em "Nº de nós por camada" com o valor de 3, bem como o seu resultado:

```
no(1,ana,[natureza,pintura,musica,sw,porto]).

no(11,antonio,[natureza,pintura,carros,futebol,lisboa]).
no(12,beatriz,[natureza,musica,carros,porto,moda]).
no(13,carlos,[natureza,musica,sw,futebol,coimbra]).

no(21,eduardo,[natureza,cinema,teatro,carros,coimbra]).
no(22,isabel,[natureza,musica,porto,lisboa,cinema]).
no(23,jose,[natureza,pintura,sw,musica,carros,lisboa]).

no(31,maria,[natureza,pintura,musica,moda,porto]).
no(32,anabela,[natureza,cinema,musica,tecnologia,porto]).
no(33,andre,[natureza,carros,futebol,coimbra]).

no(200,sara,[natureza,moda,musica,sw,coimbra]).

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

ligacao(11,21,5,7).
ligacao(11,22,2,-4).
ligacao(11,23,-2,8).

ligacao(12,21,4,9).
ligacao(12,22,-3,-8).
ligacao(12,23,2,4).

ligacao(13,21,3,2).
ligacao(13,22,0,-3).
ligacao(13,23,5,9).
```

```
ligacao(21,31,2,1).
ligacao(21,32,-2,3).
ligacao(21,33,3,5).

ligacao(22,31,5,-4).
ligacao(22,32,-1,6).
ligacao(22,33,2,1).

ligacao(23,31,4,-3).
ligacao(23,32,3,5).
ligacao(23,33,4,1).

ligacao(31,200,3,2).
ligacao(32,200,4,3).
ligacao(33,200,-1,-4).
```

Caminho mais forte:

```
?- plan_maisForteLig(ana,sara,L).
Numero de solucoes obtidas: 891
Tempo de geracao da solucao obtida: 0.025002002716064453 segundos
Forca de ligacao da solucao obtida: 71
Caminho mais forte obtido:[ana,antonio,eduardo,beatriz,jose,anabela,isabel,andre,sara]
L = [ana, antonio, eduardo, beatriz, jose, anabela, isabel, andre, sara].

?- plan_maisForteUniLig(ana,sara,L).
Numero de solucoes obtidas: 891
Tempo de geracao da solucao obtida: 0.013993024826049805 segundos
Forca de ligacao da solucao obtida: 42
Caminho mais forte obtido:[ana,antonio,eduardo,beatriz,jose,anabela,isabel,maria,sara]
L = [ana, antonio, eduardo, beatriz, jose, anabela, isabel, maria, sara].
```

Caminho mais curto:

```
1 ?- plan_minlig(ana,sara,L).
Tempo de geracao da solucao:0.012994050979614258
L = [ana, antonio, eduardo, maria, sara].

2 ?- caminho_mais_curto(ana,sara,_,L).
Tempo de geracao da solucao:0.008997917175292969
L = [ana, antonio, eduardo, maria, sara].
```

Base de conhecimento utilizada para o estudo em “Nº de camadas intermédias (sem nó 1 e 200)” com o valor de 4 e em “Nº de nós por camada” com o valor de 4, bem como o seu resultado:

```
no(1,ana,[natureza,pintura,musica,sw,porto]).
no(11,antonio,[natureza,pintura,carros,futebol,lisboa]).
no(12,beatriz,[natureza,musica,carros,porto,moda]).
no(13,carlos,[natureza,musica,sw,futebol,coimbra]).
no(14,daniel,[natureza,cinema,jogos,sw,moda]).
no(21,eduardo,[natureza,cinema,teatro,carros,coimbra]).
no(22,isabel,[natureza,musica,porto,lisboa,cinema]).
no(23,jose,[natureza,pintura,sw,musica,carros,lisboa]).
no(24,luisa,[natureza,cinema,jogos,moda,porto]).
no(31,maria,[natureza,pintura,musica,moda,porto]).
no(32,anabela,[natureza,cinema,musica,tecnologia,porto]).
no(33,andre,[natureza,carros,futebol,coimbra]).
no(34,catia,[natureza,musica,cinema,lisboa,moda]).
no(41,cesar,[natureza,teatro,tecnologia,futebol,porto]).
no(42,diogo,[natureza,futebol,sw,jogos,porto]).
no(43,ernesto,[natureza,teatro,carros,porto]).
no(44,isaura,[natureza,moda,tecnologia,cinema]).
no(200,sara,[natureza,moda,musica,sw,coimbra]).

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

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).

ligacao(31,41,2,4).
ligacao(31,42,6,3).
ligacao(31,43,2,1).
ligacao(31,44,2,1).

ligacao(32,41,2,3).
ligacao(32,42,-1,0).
ligacao(32,43,0,1).
ligacao(32,44,1,2).

ligacao(33,41,4,-1).
ligacao(33,42,-1,3).
ligacao(33,43,7,2).
ligacao(33,44,5,-3).

ligacao(34,41,3,2).
ligacao(34,42,1,-1).
ligacao(34,43,2,4).
ligacao(34,44,1,-2).

ligacao(41,200,2,0).
ligacao(42,200,7,-2).
ligacao(43,200,-2,4).
ligacao(44,200,-1,-3).
```

Caminho mais forte:

```
?- plan_maisForteLig(ana,sara,L).
Numero de solucoes obtidas: 8184064
Tempo de geracao da solucao obtida: 327.61069989204407 segundos
Forca de ligacao da solucao obtida: 122
Caminho mais forte obtido:[ana,antonio,eduardo,beatriz,jose,carlos,luisa,daniel,isabel,catia,ernesto,andre,isaura,anabela,cesar,maria,diogo,sara]
L = [ana, antonio, eduardo, beatriz, jose, carlos, luisa, daniel, isabel|...].

?- plan_maisForteUniLig(ana,sara,L).
Numero de solucoes obtidas: 8184064
Tempo de geracao da solucao obtida: 312.5943069458008 segundos
Forca de ligacao da solucao obtida: 83
Caminho mais forte obtido:[ana,antonio,luisa,andre,ernesto,catia,cesar,anabela,jose,carlos,eduardo,daniel,isabel,maria,diogo,sara]
L = [ana, antonio, luisa, andre, ernesto, catia, cesar, anabela, jose|...].
```

Caminho mais curto:

```
1 ?- plan_minlig(ana,sara,L).
Tempo de geracao da solucao:100.7441349029541
L = [ana, antonio, eduardo, maria, cesar, sara].

2 ?- caminho_mais_curto(ana,sara,_,L).
ERROR: Stack limit (1.0Gb) exceeded
ERROR:   Stack sizes: local: 5Kb, global: 46Kb, trail: 1Kb
ERROR:   Stack depth: 17, last-call: 29%, Choice points: 25
ERROR:   In:
ERROR:     [17] system:'$add_findall_bag'([length:16])
ERROR:     [16] '$bags':findall_loop([length:16], <compound (:)/2>, _11886, [])
ERROR:     [15] system:setup_call_catcher_cleanup(<compound (:)/2>, <compound (:)/2>, _11922, <compound (:)/2>)
ERROR:     [10] user:caminho_mais_curto(ana, sara, [], _11966)
ERROR:     [9] '$toplevel':toplevel_call(<compound (:)/2>)
ERROR:
ERROR: Use the --stack_limit=size[KMG] command line option or
ERROR: ?- set_prolog_flag(stack_limit, 2_147_483_648). to double the limit.
^  Exception: (15) setup_call_catcher_cleanup('$bags':'$new_findall_bag', '$bags':findall_loop(_10466, user:dfs(ana
dall_bag') ?
```

8. Conclusões:

Após a implementação das funcionalidades pedidas no âmbito da cadeira de ALGAV, podemos retirar as seguintes conclusões:

- Os métodos de encontrar caminhos com findall são menos eficientes do que os métodos sem findall.
- Com o método findall se tivermos muitas soluções obtemos um erro de Stack Limit Exceeded.
- À medida em que aumentamos o nº de nós por camada e o nº de camadas intermédias os valores temporais aumentam também tornando os métodos de determinação de caminhos pouco eficientes para grandes quantidades de informação.