
# Relatório de ALGAV

**Turma 3DF_ 032**
1181616 _ Tiago Oliveira
1180604 _ Vasco Silva
1190881 _ Matheus Figueira
1190835 _ Luís Teixeira

**Data: 05/12/2021**

# Índice

## Parte I – Introdução e estrutura do relatório

Este relatório visa expor e explicar os algoritmos criados para o módulo de análise da rede social (ARS), no âmbito da Unidade Curricular de Algoritmia Avançada, lecionada (Aulas Pratico-Laboratoriais) pelo professor Jorge Coelho (JMN). Este módulo será parte integrante do projeto integrador do 5º semestre da Licenciatura de Engenharia Informática do ISEP, no ano curricular 21/22. Os algoritmos estão escritos em PROLOG.

O relatório está dividido em três partes:

- Parte 1 – Introdução e estrutura do relatório;
- Parte 2 – Desenvolvimento: onde terá a exposição dos casos de uso, respetiva implementação em PROLOG e explicação com exemplos;
- Parte 3 - Conclusão.

No final, encontram-se as referências utilizadas para o desenvolvimento do trabalho.

## Parte II – Desenvolvimento

### II.1 Use Cases – Casos de Uso (UC)

Nesta secção iremos expor os casos de uso requisitados.

- UC34 – Determinar o tamanho da rede de um utilizador (até um determinado nível);
- UC35 – 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);
- UC36 – Sugerir conexões com outros utilizadores tendo por base as tags e conexões partilhadas (até determinado nível);
- UC37 – Determinar o caminho mais forte (maximiza o somatório das forças de ligação) para determinado utilizador;
- UC38 – Determinar o caminho mais curto (minimiza o número de ligações) para determinado utilizador;
- UC39 – 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.

### II.2 Explicação e Exemplos dos UC

Nesta secção iremos analisar, explicar e exemplificar cada um dos casos de uso.

#### II.2.1 UC34

Neste caso de uso o pretendido é determinar o tamanho da rede de um utilizador até um determinado nível, por exemplo, tenha-se um utilizador com uma rede com quatro níveis, sendo o nível pretendido *n=2,* o tamanho da rede será a soma de todos os amigos chegados, e, ainda, os amigos dos amigos chegados.

Para a implementação deste algoritmo, criamos seis predicados: ***vizinhança/3, vizinhanca2/3, atl/3, addToList/3, empty/1, tamanho_lista/3.***

O predicado ***vizinhança/3*** é definido como ***vizinhança(N, E, R)***, onde *N* é o nível pretendido, *E* é o utilizador *root,* e *R* é o resultado final. Este predicado inicia o programa, e utiliza os predicados ***vizinhança2/3***, ***flatten/2*** e ***tamanho_lista/2***.

O predicado ***vizinhança2/3*** é definido como ***vizinhança2(N, E, L),*** onde *N* é o nível atual, *E* é o utilizador atual, e *L* é uma lista de utilizadores, preenchida recursivamente. Este predicado é chamado recursivamente, analisando a rede toda, até que sejam encontrados todos os utilizadores, até ao nível *N*, da rede do utilizador. O predicado faz isso verificando se *N>0*; prosseguindo para a invocação do predicado ***findall/3***, para encontrar todas as ligações diretas a *E* (tais ligações serão guardadas em *L2*); de seguida, chama o predicado ***mapList/3*** para invocar o predicado ***vizinhança2/3*** para todos os elementos de *L2*, guardando em *L3*; feito este último passo, chama o predicado ***addToList/3*** para concatenar *L2* e *L3,* guardando o resultado em *L4*; remove os elementos repetidos e guarda na lista *L5* na invocação do predicado ***sort/2***; passando, depois, a remover os elementos vazios com o predicado ***exclude/3,*** guardando na lista final *L*.

```
vizinhanca2(N,E,L):-
    N>0,
    findall(Y,ligacao(E,Y,_,_),L2), %encontra ligacoes diretas de E, guarda em L2
    N2 is N-1,
    maplist(vizinhanca2(N2),L2,L3), %chama predicado para cada elemento de L2, guarda em L3
    addToList(L2,L3,L4), %concatena L2 e L3
    sort(L4,L5), %remove elementos repetidos
    exclude(empty,L5,L). %remove elementos vazios

vizinhanca2(0,_,[]).

atl(_, [], []).
atl(List, [H|T], R) :-
(  member(H, List)
-> R = Res
;  R = [H|Res]
),
atl(List, T, Res).
addToList(A, B, L) :-
atl(A, B, R),
append(A, R, L).

empty([]).
```

*Figura 1- Exposição dos predicados vizinhança2/3, atl/3, addToList/3*

Feita esta parte, o algoritmo retorna ao predicado ***vizinhança/3***, estando já *L* preenchido, o predicado chama ***flatten/2,*** para achatar a lista *L;* de seguida, é invocado o método ***tamanho_lista/3*** para determinar o tamanho de L e, assim, devolver o tamanho da rede do utilizador.

Passando a mostrar alguns exemplos, utilizando na base de conhecimento fornecida no moodle da disciplina:

- Exemplo 1: Diferentes níveis utilizados. E=1, N=1; N=2; N=3; N=4; N=5; N=6;

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

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

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

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

?- vizinhanca(5, 1, R).
R = 20 .

?- vizinhanca(6, 1, R).
R = 20 .
```

*Figura 2-Exemplo 1.*

- Exemplo 2: Outro utilizador *root* para os mesmos níveis acima representados E=11.

```
?- vizinhanca(1, 11, R).
R = 4 .

?- vizinhanca(2, 11, R).
R = 8 .

?- vizinhanca(3, 11, R).
R = 12 .

?- vizinhanca(4, 11, R).
R = 13 .

?- vizinhanca(5, 11, R).
R = 13 .
```

*Figura 3- Exemplo 2.*

**II:2.2 UC35**

**II.2.3 UC36**

Neste caso de uso pretende-se determinar o caminho mais curto entre 2 utilizadores. No caderno de encargos define-se como “mais curto” o caminho com menor número de ligações até determinado utilizador.

Para este UC foram desenvolvidos os predicados caminho/4, atravessa/5, atravessa/5, mais_Curto/4, mínimo/3 e min/3.

Começando pelos predicados que são chamados por último, o predicado min/3 recebe todos os caminhos possíveis entre dois utilizadores e verifica qual o menor recorrendo a recursividade. A cada novo caminho verifica se a “length” é menor que a atual, case seja, atualiza o caminho mais curto para o caminho lido, o predicado corre enquanto houver caminhos na lista recebida.

O predicado atravessa/5 recebe o utilizador Inicial e o final, a cada iteração verifica se o próximo amigo do utilizador é o final e caso não seja, muda o utilizador inicial para o amigo. Este método corre até encontrar o utilizador final mesmo que não seja o caminho mais curto. Quando encontra o utilizador final guarda esse caminho e “length” .

O predicado caminho/4 apenas chama o predicado atravessa/4 e dá reverse ao caminho para este vir na ordem correta.

O predicado mais_Curto /4 é o predicado que é chamado na linha de comandos, no qual se coloca o utilizador inicial (A), Utilizador Final (B) e a lista que se pretende receber o caminho mais curto(Path) e o comprimento da lista (Length). Este predicado evoca o método caminho até ter encontrado todos os caminhos possíveis de A para B, guardando cada Caminho e Comprimento juntos. De seguida verifica se a lista de caminhos e comprimentos está vazia, se estiver dá erro, ou seja, não existe caminho possível entre o utilizador A e B, caso não esteja evoca o método min para verificar qual o caminho mais curto.

Segue-se uma captura de todos os predicados implementados, para facilitar a compreensão.

```
:-consult('bc sprintB rede social').

caminho(A,B,Path,Len) :-
       atravessa(A,B,[A],Q,Len),
       reverse(Q,Path).

atravessa(A,B,P,[B|P],L) :-
        L is 1,
        no(NumA,A,_),
        no(NumB,B,_),
       ligacao(NumA,NumB,_,_).

atravessa(A,B,Visited,Path,L) :-
        no(NumA,A,_),
       ligacao(NumA,C,_,_),
       no(C,NomeC,_),
       NomeC \== B,
       \+member(NomeC,Visited),
       atravessa(NomeC,B,[NomeC|Visited],Path,L1),L is L1+1.

mais_Curto(A,B,Path,Length) :-
   setof([P,L],caminho(A,B,P,L),Set),
   Set = [_|_], % fail if empty
   minimo(Set,[Path,Length]).

minimo([F|R],M) :- min(R,F,M).

% minimo caminho
min([],M,M).
min([[P,L]|R],[_,M],Min) :- L < M, !, min(R,[P,L],Min).
min([_|R],M,Min) :- min(R,M,Min).
```

De seguida apresentam-se alguns exemplos do algoritmo em funcionamento.

```
?- mais_Curto(ana,eduardo,P,L).
P = [ana, antonio, eduardo],
L = 2.

?- mais_Curto(ana,antonio,P,L).
P = [ana, antonio],
L = 1.

?- mais_Curto(ana,sara,P,L).
P = [ana, antonio, eduardo, anabela, cesar, sara],
L = 5.

?- mais_Curto(ana,diogo,P,L).
P = [ana, antonio, eduardo, anabela, diogo],
L = 4.
```

### II.2.4 UC37

Neste caso de uso pretende-se determinar o caminho mais forte entre 2 utilizadores. No caderno de encargos define-se como "mais forte" o caminho que maximiza o somatório das forças de ligação. No

entanto, nós consideramos que esta abordagem iria gerar caminhos desnecessariamente longos, uma vez que mais ligações tem o potencial de gerar um somatório maior. De forma a alcançar um meio termo entre "força" e distância, decidimos considerar antes a média do somatório das forças. Se for necessário, a implementação pode ser facilmente adaptada para considerar a abordagem anterior.
Para este UC foram desenvolvidos os predicados plan_strlig/3, melhor_caminho_strlig/2, atualiza_melhor_strlig/1, forca/3, e 2 predicados forca/1. Também reutiliza o predicado dfs/3, disponibilizado na TP de apoio ao projeto de ALGAV.

Começando pelos predicados que são chamados por último, o predicado forca/3 (forca(X,Y,S)) recebe dois nós X e Y e guarda a força de ligação entre os 2 em S.
O primeiro predicado forca/2 (forca([X,Y],S)), recebe uma lista de 2 nós X e Y e chama o predicado forca/3 sobre os mesmos membros.
O segundo predicado forca/2 (forca([X|Y],S)) recebe uma lista de head X e tail Y. Primeiro chama-se a si próprio para a tail Y (percorre a tail recursivamente e faz a soma das forças) e guarda a soma das forças em S1. Em segundo, chama forca/3 para X e o primeiro membro de Y e guarda a forca em S2. Depois, soma S1 e S2, obtendo a soma das forças de ligação da lista inteira.
O predicado atualiza_melhor_strlig/1 (atualiza_melhor_strlig(LCaminho)) compara LCaminho com a solução atual (dynamic melhor_sol_strlig/2). Primeiro calcula a média da soma das forças de LCaminho, dividindo a soma pela length. Depois compara o resultado com a média da solução atual. Se for superior, atualiza a solução com LCaminho.

O predicado melhor_caminho_strlig/2 (melhor_caminho_strlig(Orig,Dest)) define a solução com valor -10000 (valor placeholder que vai ser substituído na primeira execução), chama dfs para Orig e Dest, guarda o caminho em LCaminho. Este predicado vai ser chamado várias vezes para cada caminho possível entre Orig e Dest. Depois chama atualiza_melhor_strlig/1 para o LCaminho atual.
O predicado plan_strlig/3(plan_strlig(Orig,Dest,LCaminho_strlig)) chama o predicado melhor_caminho_strlig/2 para cada solução de dfs. No final, a solução (dynamic_melhor_sol_strlig/2) deve representar o caminho com a maior média de forças de ligação.
Segue-se uma captura de todos os predicados implementados, para facilitar a compreensão.

```
:-dynamic melhor_sol_strlig/2.

plan_strlig(Orig,Dest,LCaminho_strlig):-
        get_time(Ti),
        (melhor_caminho_strlig(Orig,Dest);true),
        retract(melhor_sol_strlig(LCaminho_strlig,_)),
        get_time(Tf),
        T is Tf-Ti,
        write('Tempo de geracao da solucao:'),write(T),nl.

melhor_caminho_strlig(Orig,Dest):-
        asserta(melhor_sol_strlig(_,-10000)),
        dfs(Orig,Dest,LCaminho),
        atualiza_melhor_strlig(LCaminho),
        fail.

atualiza_melhor_strlig(LCaminho):-
        melhor_sol_strlig(_,N),
        forca(LCaminho,S),
        length(LCaminho,Q),
        R is S/Q,
        R>N,retract(melhor_sol_strlig(_,_)),
        asserta(melhor_sol_strlig(LCaminho,R)).

forca(X,Y,S):-
        no(N,X,_),
        %member(P,Y),
        %no(M,P,_),
        no(M,Y,_),
        ligacao(N,M,S,_).
forca([X,Y],S):-
        forca(X,Y,S).
%forca([X|Y],S):-
%       length([X|Y],2),
%       forca(X,Y,S).
forca([X|Y],S):-
        forca(Y,S1),
        list_head(Y,E),
        forca(X,E,S2),
        S is S1+S2.

list_head([X|_],X).
```

De seguida apresentam-se alguns exemplos do algoritmo em funcionamento.

```
?- plan_strlig(ana,sara,Cam).
Tempo de geracao da solucao:0.27104997634887695
Cam = [ana, antonio, eduardo, maria, diogo, sara] .

?- plan_strlig(ana,ernesto,Cam).
Tempo de geracao da solucao:0.20704889297485352
Cam = [ana, antonio, eduardo, andre, ernesto] .

?- plan_strlig(ana,anabela,Cam).
Tempo de geracao da solucao:0.10399699211120605
Cam = [ana, antonio, eduardo, anabela] .

?- plan_strlig(carlos,sara,Cam).
Tempo de geracao da solucao:0.2689931392669678
Cam = [carlos, jose, maria, diogo, sara] .

?- plan_strlig(beatriz,sara,Cam).
Tempo de geracao da solucao:0.2669999599456787
Cam = [beatriz, eduardo, maria, diogo, sara] .

?- plan_strlig(isabel,sara,Cam).
Tempo de geracao da solucao:0.1570439338684082
Cam = [isabel, maria, diogo, sara] .
```

### II.2.5 UC38

### II.2.6 UC39

Neste caso de uso pretende-se determinar o caminho mais seguro entre 2 utilizadores. No caderno de encargos define-se como "mais seguro" o caminho que maximiza o somatório das forças de ligação, mas garantindo que o nenhuma das ligações tenha um valor abaixo de um dado limiar. No entanto, nós consideramos que esta abordagem iria gerar caminhos desnecessariamente longos, uma vez que mais ligações tem o potencial de gerar um somatório maior. De forma a alcançar um meio termo entre "força" e distância, decidimos considerar antes a média do somatório das forças. Se for necessário, a implementação pode ser facilmente adaptada para considerar a abordagem anterior.

Para este UC foram desenvolvidos os predicados plan_strlig2/4, melhor_caminho_strlig2/3, atualiza_melhor_strlig2/2, forca2/5, e 2 predicados forca2/4. Também reutiliza o predicado dfs/3, disponibilizado na TP de apoio ao projeto de ALGAV.

Na utilização do recurso começamos por invocar o predicado plan_strlig2/4 que é definido por plan_strlig(Orig,Dest,L,LCaminho_strlig), sendo os parâmetros respetivamente o nó de origem, o nó de destino, o limiar mínimo da força entre os nós e o caminho resultante. Este predicado invoca o melhor_caminho_strlig2(Orig,Dest,L), que por sua vez define –10000 como placeholder para ser substituído na primeira execução, depois usa o dfs(Orig,Dest,LCaminho) e manda o resultado LCaminho para o atualiza_melhor_strlig2(LCaminho,L), que chama o predicado forca2(LCaminho,L,S,S2), que tem as variáveis S e S2 para as duas forças entre cada nó, depois são comparados os valores da média do caminho com o valor guardado em melhor_sol_strlig2/2.

Para os predicados forca2/5 e forca2/4, temos chamadas recursivas em que são feitas comparações entre o valor do limiar e o valor das duas forças de ligação entre dois nós.

```
/*
Caminho mais seguro com força limiar determinada pelo utilizador, neste metodo temos
a determinação do caminho com maior força média no menor número de ligações, sendo
que devemos ter uma limiar determinada pelo utilizado, não devendo nenhuma das ligações
entre dois nós ser menor que essa limiar
*/

:-dynamic melhor_sol_strlig2/2.

plan_strlig2(Orig,Dest,L,LCaminho_strlig):-
        get_time(Ti),
        (melhor_caminho_strlig2(Orig,Dest,L);true),
        retract(melhor_sol_strlig2(LCaminho_strlig,_)),
        get_time(Tf),
        T is Tf-Ti,
        write('Tempo de geracao da solucao:'),write(T),nl.

melhor_caminho_strlig2(Orig,Dest,L):-
        asserta(melhor_sol_strlig2(_,-10000)),
        dfs(Orig,Dest,LCaminho),
        atualiza_melhor_strlig2(LCaminho,L),
        fail.

atualiza_melhor_strlig2(LCaminho,L):-
        melhor_sol_strlig2(_,N),
        forca2(LCaminho,L,S,S2),
        length(LCaminho,Q),
        R is S/Q,
        R>N,retract(melhor_sol_strlig2(_,_)),
        asserta(melhor_sol_strlig2(LCaminho,R)).
```

```
forca2(X,Y,L,S1,S2):-
        no(N,X,_),
        no(M,Y,_),
        ligacao(N,M,S1,S2).

forca2([X,Y],L,S1,S2):-
        forca2(X,Y,L,S1,S2).

forca2([X|Y],L,S1,S2):-
        forca2(Y,L,F,P),
        F>L-1,
        P>L-1,
        list_head(Y,E),
        forca2(X,E,L,Q,N),
        Q>L-1,
        N>L-1,
        S1 is F+Q,
        S2 is P+N.
```

```
?- plan_strlig2(ana,sara,-10,C).
Tempo de geracao da solucao:0.0009970664978027344
C = [ana, antonio, eduardo, maria, diogo, sara].

?- plan_strlig2(ana,sara,2,C).
Tempo de geracao da solucao:0.0009708404541015625
true.


?- plan_strlig2(antonio,sara,-1,C).
Tempo de geracao da solucao:0.0009968280792236328
C = [antonio, eduardo, maria, cesar, sara].

?- plan_strlig2(antonio,diogo,-1,C).
Tempo de geracao da solucao:0.0
C = [antonio, eduardo, maria, diogo].

?- plan_strlig2(antonio,diogo,10,C).
Tempo de geracao da solucao:0.0
true.
```

**Parte III - Conclusões**

Para concluir, conseguimos implementar a maior parte dos algoritmos propostos para a unidade curricular de ALGAV, no entanto não conseguimos fazer a integração com os projetos relacionados as restantes unidades curriculares e por isso a demonstração dos algoritmos só é possível no SWI-Prolog, até o presente momento.

**Referências**


[1] C. Ramos, “Listas Prolog”.

[2] U. Curricular and A. Avançada, “Apoio ao Sprint B do Trabalho Prático de ALGAV Planeamento de Contatos em Redes Sociais”.

[3] C. Ramos, “Aspetos Complementares Prolog”.


[1], [2][3]

---

i