

Uma imagem com alimentação

Descrição gerada automaticamente

**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 3](#_Toc89516505)

[Parte II – Desenvolvimento 4](#_Toc89516506)

[II.1 Use Cases – Casos de Uso (UC) 4](#_Toc89516507)

[II.2 Explicação e Exemplos dos UC 4](#_Toc89516508)

[II.2.1 UC34 4](#_Toc89516509)

[II:2.2 UC35 7](#_Toc89516510)

[II.2.3 UC36 7](#_Toc89516511)

[II.2.4 UC37 7](#_Toc89516512)

[II.2.5 UC38 7](#_Toc89516513)

[II.2.6 UC39 7](#_Toc89516514)

[Parte III - Conclusões 7](#_Toc89516515)

[Referências 9](#_Toc89516516)

# 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*.

Uma imagem com texto

Descrição gerada automaticamente

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;

Uma imagem com mesa

Descrição gerada automaticamente

Figura 2-Exemplo 1.

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

Uma imagem com texto

Descrição gerada automaticamente

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” .

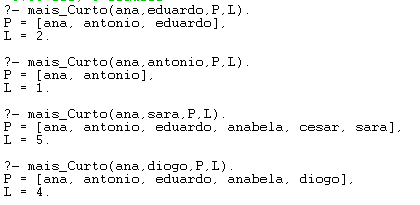
O predicado caminho/4 apenas chama o predicado atravessa/4 e dá reverse ao caminho para este vir na ordem correta.

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.

figura 4- Exemplo 1.
[[1]](#endnote-9384)

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



## 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.

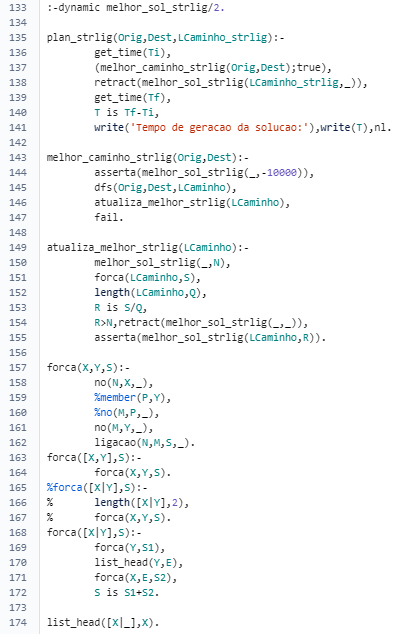
O segundo predicado forca/2 (forca([X|Y],S)) recebe uma lista de head X e tail Y. Primeiro chama-se a si próprio para a tail Y (percorre a tail recursivamente e faz a soma das forças) e guarda a soma das forças em S1. Em segundo, chama forca/3 para X e o primeiro membro de Y e guarda a forca em S2. Depois, 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.



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

Uma imagem com texto

Descrição gerada automaticamente

Uma imagem com texto

Descrição gerada automaticamente

## II.2.5 UC38

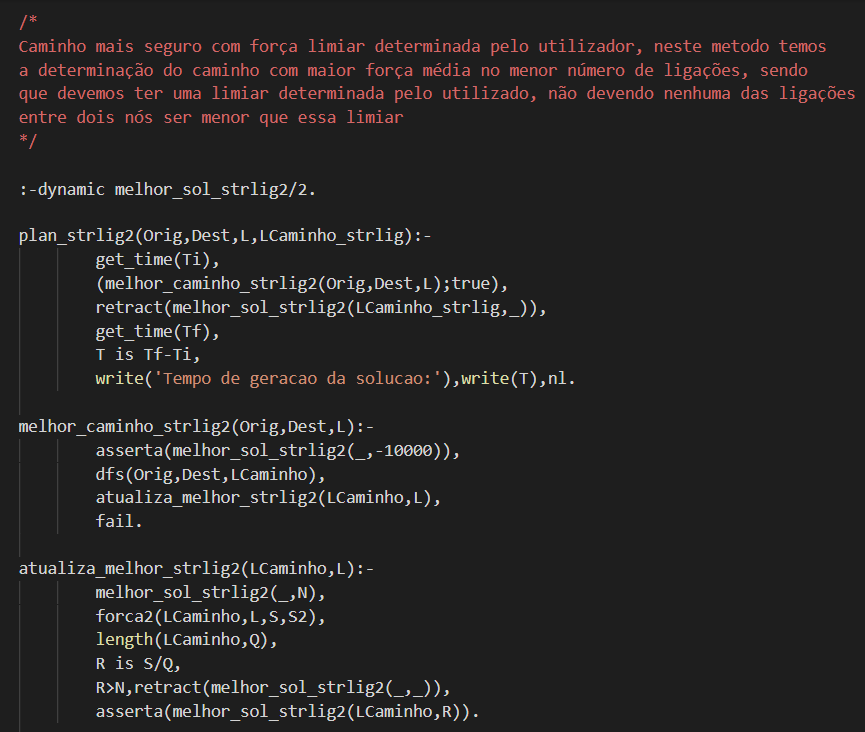
## II.2.6 UC39

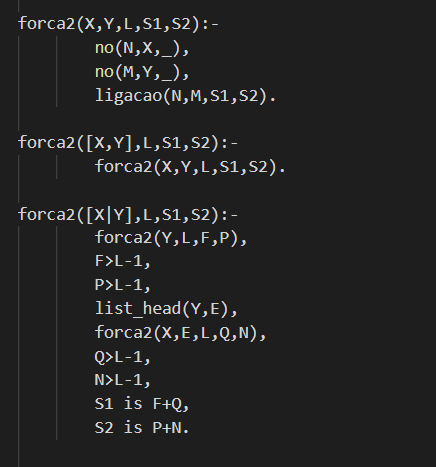
Neste caso de uso pretende-se determinar o caminho mais seguro entre 2 utilizadores. No caderno de encargos define-se como "mais seguro" o caminho que maximiza o somatório das forças de ligação, mas garantindo que o nenhuma das ligações tenha um valor abaixo de um dado limiar. No entanto, nós consideramos 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.

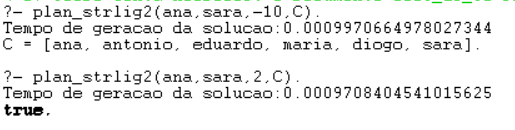
Para este UC foram desenvolvidos os predicados plan\_strlig2/4, melhor\_caminho\_strlig2/3, atualiza\_melhor\_strlig2/2, forca2/5, e 2 predicados forca2/4. Também reutiliza o predicado dfs/3, disponibilizado na TP de apoio ao projeto de ALGAV.

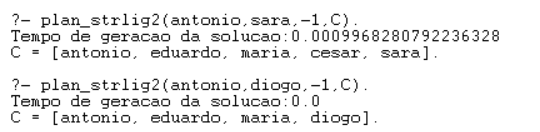
Na utilização do recurso começamos por invocar o predicado plan\_strlig2/4 que é definido por plan\_strlig(Orig,Dest,L,LCaminho\_strlig), sendo os parâmetros respetivamente o nó de origem, o nó de destino, o limiar mínimo da força entre os nós e o caminho resultante. Este predicado invoca o melhor\_caminho\_strlig2(Orig,Dest,L), 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.

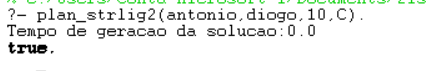
Para os predicados forca2/5 e forca2/4, temos chamadas recursivas em que são feitas comparações entre o valor do limiar e o valor das duas forças de ligação entre dois nós.











## 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]

1. [↑](#endnote-ref-9384)