Uma imagem com texto, ClipArt

Descrição gerada automaticamente

**ALGAV – Sprint B**

2021 / 2022

**1191454 - Bruno Pereira**

**1180658 - Rafael Faísca**

**1191059 - Sérgio Balio**

**1170385 - Rui Mendes**



**Representação do conhecimento do domínio**

Para o desenvolvimento do projeto foram utilizados os seguintes factos:

* **Nós:**

Para os factos nó foi definido o seguinte formato:

*nó(userId, userName, tagsList).*

Ex: *nó(1,ana,[natureza, pintura, musica, sw, porto]).*

* **Ligações:**

Para os factos ligação foi definido o seguinte formato:

*ligacao(userAId, userBId, connectionStrengthA-B, connectionStrengthB-A).*

Ex: *ligacao(1,11,10,8).*

**Tamanho da rede do utilizador**

Determinar o tamanho da rede de um utilizador até determinado nível.

O objetivo deste algoritmo é calcular o tamanho da rede de um utilizador e retornar uma lista com essa rede.

São adicionados os utilizadores que estão diretamente ligados ao utilizador origem, é verificado se existem utilizadores repetidos e a rede é atualizada removendo os repetidos, este processo é repetido para cada utilizador de cada nível até o nível pretendido.

Uma imagem com texto

Descrição gerada automaticamente**Sugerir conexões com tags em comum**

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

Uma imagem com texto

Descrição gerada automaticamenteO objetivo deste algoritmo é sugerir todas as conexões possíveis com utilizadores de determinado nível, onde todos os nós que fazem ligação até o utilizador destino partilham a mesma tag em comum com o utilizador origem.

Uma imagem com texto

Descrição gerada automaticamenteInicialmente é gerada a rede do utilizador até o nível pretendido, feito no exercício anterior do tamanho da rede de um utilizador, de seguida são removidos os utilizadores da rede que não partilhem nenhuma tag em comum com o utilizador origem.

Depois de obtermos a rede do utilizador filtrada, é usado o findall para obter todas as soluções possíveis. Para cada utilizador destino possível pesquisamos as tags em comum entre o utilizador origem e destino, de seguida é chamado o DFS que verifica se cada utilizador do caminho partilha a mesma tag da lista de tags em comum nas ligações entre o utilizador origem e destino.

Uma imagem com texto

Descrição gerada automaticamente

**Consultar o caminho mais curto**

Consultar o caminho mais curto, ou seja, com menor número de ligações, para chegar a um determinado utilizador.

O objetivo deste algoritmo é determinar o caminho com menor número de ligações para do utilizador origem chegar ao utilizador destino.

Explicação do algoritmo

 aqui definimos melhor\_sol\_minlig(caminho,numero\_ligacoes) na nossa base de conhecimentos para mais tarde a utilizarmos para guardar a solução mais curta ate ao momento.

Uma imagem com texto

Descrição gerada automaticamente

O predicado plan\_minlig é o predicado chamado pelo utilizador para se determinar o caminho mais curto. Na 3º linha chama o predicado melhor\_caminho\_minlig que ira colocar a melhor solução na base de conhecimentos(melhor\_sol\_minlig).Na 4º linha colocamos a melhor solução no LCaminho\_minlig para ser retornado ao utilizador e apagamos essa entrada da base de conhecimentos.

Uma imagem com texto

Descrição gerada automaticamente

O predicado melhor\_caminho\_minlig comeca por adicionar a base de conhecimentos a melhor solução, sendo esta sem caminho e com o valor 10000 no número de ligações com o objetivo de ser a pior solução para depois poder ser alterada (só e alterada quando a ligação encontrada e melhor do que a q já se encontra na base de conhecimentos). Depois utiliza o predicado dfs, disponibilizado nas tps, para fazer uma busca em profundidade, ficando o caminho em LCaminho que depois e enviado por parâmetro para o predicado atualiza\_melhor\_minlig que ira atualizar na base de conhecimentos a melhor solução (entre a que já se encontra lá e o LCaminho).

Uma imagem com texto

Descrição gerada automaticamente

Por fim, o predicado atualiza\_melhor\_minlig vai buscar a base de conhecimentos o número de ligações que a atual melhor solução tem, colocando-o em N e conta o número de ligações que o LCaminho, caminho recebido por parâmetro, colocando-o em C. Depois se C for menor que N então retira a melhor solução que estava na base de conhecimentos e insere a nova melhor solução (LCaminho).

**Consultar o caminho mais forte**

Consultar o caminho mais forte, ou seja, com maior soma das forças de ligação em ambos os sentidos, para chegar a um determinado utilizador.

O objetivo deste algoritmo é determinar o caminho com o maior somatório das forças de ligação para o utilizador origem chegar ao utilizador destino.

 Explicação do algoritmo

aqui definimos melhor\_sol\_forte (caminho, lista\_forças\_ligaçao, força\_ligaçoes) na nossa base de conhecimentos para mais tarde a utilizarmos para guardar a solução mais forte até ao momento.

Uma imagem com texto

Descrição gerada automaticamente

O predicado plan\_forte é o predicado chamado pelo utilizador para se determinar o caminho mais forte. Na 2º linha chama o predicado melhor\_caminho\_forte que ira colocar a melhor solução na base de conhecimentos (melhor\_sol\_forte). Na 3º linha colocamos a melhor solução no LCaminho\_minlig para ser retornado ao utilizador e apagamos essa entrada da base de conhecimentos.

Uma imagem com texto

Descrição gerada automaticamente

O predicado melhor\_caminho\_forte começa por adicionar a base de conhecimentos a melhor solução, sendo esta sem caminho e lista de forças e com o valor -10000 na força de ligações com o objetivo de ser a pior solução para depois poder ser alterada (só é alterada quando a ligação encontrada é melhor do que a que já se encontra na base de conhecimentos). Depois utiliza o predicado dfs\_forte, disponibilizado nas tps, para fazer uma busca em profundidade, ficando o caminho em LCaminho e a lista de forças de ligação em LF que depois são enviados por parâmetro para o predicado atualiza\_melhor\_forte que ira atualizar na base de conhecimentos a melhor solução (entre a que já se encontra la e o LCaminho).

Uma imagem com texto

Descrição gerada automaticamente

Por fim, o predicado atualiza\_melhor\_forte vai buscar a base de conhecimentos e a força de ligações que a atual melhor solução tem, colocando-a em N e soma as forças de ligações do LCaminho, recorrendo a LF, colocando-a em SF. Depois se SF for maior que N então retira a melhor solução que estava na base de conhecimentos e insere a nova melhor solução (LCaminho).

**Consultar o caminho mais seguro**

Consultar o caminho mais seguro, ou seja, garante que não há uma força de ligação inferior a X considerando as forças nos dois sentidos da ligação, para chegar a um determinado utilizador.

O objetivo deste algoritmo é determinar o caminho com maior somatório das forças de ligação para o utilizador origem chegar ao utilizador destino garantindo que as forças de ligação em ambos os sentidos são maiores ou iguais a X.

Explicação do algoritmo

Para este predicado utilizamos o algoritmo do caminho mais forte com umas pequenas alterações.

Uma imagem com texto

Descrição gerada automaticamente

O predicado plan\_secure é o predicado chamado pelo utilizador para se determinar o caminho mais seguro. Na 2º linha chama o predicado melhor\_caminho\_secure que ira colocar a melhor solução na base de conhecimentos (melhor\_sol\_forte). Na 3º linha colocamos a melhor solução no LCaminho\_forte para ser retornado ao utilizador e apagamos essa entrada da base de conhecimentos.

Uma imagem com texto

Descrição gerada automaticamente

O predicado melhor\_caminho\_secure é imuito semelhante ao melhor\_caminho\_forte, tendo apenas a adição da condição que nenhuma das forças pode ser inferior à força mínima (SEC). Esta condição encontra-se na 4º linha.

**Estudo da complexidade da determinação de caminhos**

Nesta entrega do projeto de ALGAV analisamos a complexidade O(n) da determinação do caminho mais forte, curto e seguro.

Figura – tabela para ligações unidirecionais

Uma imagem com mesa

Descrição gerada automaticamente

Figura - tabela para ligações bidirecionais

Com estes dados podemos concluir que com o aumento do número de nós o número de soluções e o tempo de execução também aumenta, sendo que para as ligações bidirecionais a diferença é mais espontânea.

Se considerarmos o número de nós por camada com a letra N e o número de camadas intermédias, com a expressão: C^N conseguimos chegar ao número de soluções por nível.

Podemos assim concluir que os nossos algoritmos têm uma complexidade de O(C^N).