Uma imagem com texto

Descrição gerada automaticamente

Instituto Superior de Engenharia do Porto

Licenciatura em Engenharia Informática

3DI

Relatório de Algoritmia Avançada

**Grupo 051**

Por Beatriz Seixas (1190424),

Jéssica Alves (1190682)

Pedro Santos (1190967),

Tiago Costa (1191460)

Porto, dezembro de 2021

**Índice geral**

1. Introdução 7

2. Representação do Conhecimento do Domínio 8

2.1. Base de Conhecimento 8

3. Determinação do Tamanho da Rede de um utilizador até um determinado Nível 10

3.1. Funcionalidade desenvolvida por Beatriz Seixas (1190424) 10

3.2. 10

4. Obtenção dos Utilizadores que tenham em Comum X Tags 12

4.1. Funcionalidade desenvolvida por Tiago Costa (1191460) e Beatriz Seixas (1190424) 12

5. Sugestão das Conexões com outros Utilizadores tend por base as Tags e Conexões Partilhadas Até um determinado Nível 14

5.1. Funcionalidade Desenvolvida por Jéssica Alves (1190682) e Pedro Santos (1190967) 14

6. Determinação do Caminho Mais Forte 16

6.1. Funcionalidade desenvolvida por Tiago Costa (1191460) 16

7. Determinação do Caminho Mais Curto 18

7.1. Funcionalidade Desenvolvida por Jéssica Alves (1190682) 18

8. Determinação do Caminho Mais Seguro 20

8.1. Funcionalidade Desenvolvida por Pedro Santos (1190967) 20

9. Estudo da Complexidade do Problema da Determinação de Caminhos 22

9.1. Caminho Mais Curto Bidirecional (sem Findall) 22

9.2. Caminho Mais Curto Bidirecional (com findall) 22

9.3. Caminho Mais Seguro (Limite Menos Restrito) 22

9.4. Caminho Mais Seguro (Forças de Ligação Positivas) 23

9.5. Caminho Mais Forte Bidirecional 24

10. Conclusões 25

10.1. Titulo 25

**Índice de Figuras**

Figura 1 - Base de Conhecimento Completa (Connection) 8

Figura 2 - Facto dinâmico referente ao caminho mais seguro 8

Figura 3 - Base de Conhecimento Completa (Node) 9

Figura 4 - Representação Gráfica da Base de Conhecimento Completa 9

Figura 5-Figura 1 – Predicado network\_getNetworkByLevel/4 para determinação do Tamanho da Rede 11

Figura 6-Predicado dfs2/3 para determinação do Tamanho da Rede 11

Figura 7-exemplo 11

Figura 8 - Common Tags 13

Figura 9 - Common\_tags\_users\_combination\_aux 13

Figura 10 - Common\_tags\_combination 13

Figura 11 - Common\_tags\_change\_to\_synonyms 13

Figura 12 - Common\_tags example 13

Figura 13 - Sugerir Utilizadores 15

Figura 14 - Sugestão de Utilizadores Resultado 15

Figura 15 - Sugestão (Base de Conhecimento Parcial) 15

Figura 16 - Backtracking e fail (Caminho mais forte) 17

Figura 17 - Dfs modificado para o caminho mais forte 17

Figura 18 – Resultado do caminho mais forte na base de conhecimento completa 17

Figura 19 - Caminho Mais Curto Com Findall 19

Figura 20 - Caminho Mais Curto Resultado 19

Figura 21 - Caminho Mais Curto Resultado com Findall 19

Figura 22 - Backtracking e Fail (Caminho Mais Seguro) 21

Figura 23 - DFS modificado para o caminho mais seguro 21

Figura 24 - Resultado do caminho mais seguro na base de conhecimento completa 21

Figura 25 - Exemplo de caminho não encontrado devido à falta de forças de ligação inferiores ao limite 21

Figura 26 - Conexões entre "ana" e "sara" com força menor que 4 21

# Introdução

Lorem ipsum

# Representação do Conhecimento do Domínio

## Base de Conhecimento

No Desenvolvimento deste projeto, foram utilizadas duas bases de conhecimento locais, e uma base de conhecimento integrada no projeto integrador através do protocolo *HTTP*. Na Figura 1 e Figura 3, podemos verificar a base de conhecimento completa, fornecida pelos docentes de ALGAV. Esta apresenta dois factos: *node*/3 e *connection*/4. O *node*/3 é referente a um utilizador da aplicação, denominado por jogador e apresenta, respetivamente, o seu id, nome e lista de *tags*. Já *connection*/4, é referente às ligações de amizade entre jogadores, conectando assim a nossa rede, como podemos verificar na Figura 1. Para dos factos mencionados previamente, também são utilizados alguns factos dinâmicos, como por exemplo: *dynamic safest\_currentRoute*/2 que serve de auxílio para o caso de uso referente ao caminho mais seguro, onde é guardado o caminho mais seguro atual, durante a execução do programa, par determinar qual seria esse caminho (Figura 2). Desta base de conhecimento completa resultará uma rede representada graficamente pela Figura 4. Seguidamente, a base de conhecimento parcial apenas tem alguns nós para ser mais prático o teste e apresentação das funcionalidades. Por último, a versão integrada no projeto através de *HTTP*, constrói a sua base de conhecimento e vai a atualizando ao longo dos *requests* que recebe.



Figura - Base de Conhecimento Completa (Connection)



Figura - Facto dinâmico referente ao caminho mais seguro

Uma imagem com texto, placa

Descrição gerada automaticamente

Figura - Base de Conhecimento Completa (Node)

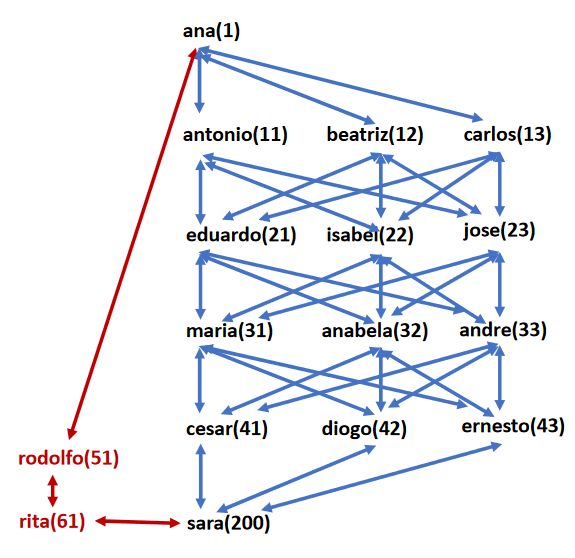


Figura - Representação Gráfica da Base de Conhecimento Completa

# Determinação do Tamanho da Rede de um utilizador até um determinado Nível

## Funcionalidade desenvolvida por Beatriz Seixas (1190424)

## 

Nesta funcionalidade, o predicado chamado irá ser o *network\_getNetworkByLevel*/4 (Figura 5). Os parâmetros para este predicado serão, em primeiro lugar, Orig(nó de origem), em segundo lugar, Level( nível pretendido), em terceiro o L(Lista) e por fim o Total( tamanho da rede).Primeiramente com o retractall(userVisited(\_)) removemos as regras e factos da base de conhecimento relativos ao userVisited/1. Depois utilizamos duas vezes o findall, a primeira vez recorre ao dfs/2 de modo a obtermos os vários caminhos possíveis que vão ser guardados no userVisited/1. O segundo findall tem como objetivo copiar os nós para a lista(L).

Seguidamente usamos novamente o retractall(userVisited(\_)) de modo a limpar os dados, e por ultimo obtemos o tamanho da lista.

Na segunda parte definimos o dfs2/3(figura6) que tem como parâmetros o nó origem(Act), o nível (Level ) e a lista auxiliar (LA), que guarda os nós.

Primeiro é feita uma validação para que o nível seja maior que zero. E seguidamente determinamos as conexões que podem ser obtidas. Verificamos depois que a conexão já não esta no userVisited e também não esta na lista auxiliar.

Depois, se não for membro vamos decrementar o nível e recorrendo ao asserta(userVisited(X)), guardamos o valor na base de conhecimento.

Por ultimo chamamos o predicado com o elemento já adicionado, tanto a lista como nível atual.

Para a base de conhecimento completa (Figura 7), com os parâmetros: “1” como Orig, “2” como Level, “L” de lista e “T” de total, este método retorna uma lista resultado assim como o tamanho da rede.

Uma imagem com texto

Descrição gerada automaticamente

Figura -Figura 1 – Predicado network\_getNetworkByLevel/4 para determinação do Tamanho da Rede

Uma imagem com texto

Descrição gerada automaticamente

Figura -Predicado dfs2/3 para determinação do Tamanho da Rede

Uma imagem com texto

Descrição gerada automaticamente

Figura -exemplo

# Obtenção dos Utilizadores que tenham em Comum X Tags

## Funcionalidade desenvolvida por Tiago Costa (1191460) e Beatriz Seixas (1190424)

Nesta funcionalidade, o predicado chamado irá ser o *common\_tags*/2 (Figura 8). Os parâmetros para este predicado serão, em primeiro lugar, x (número de tags em comum) e em segundo lugar, List\_Result, uma lista de pares de listas, as quais contêm, de um lado, as combinações de x tags (ou sinónimos) possíveis, e, do outro lado, os utilizadores aos quais essas tags correspondem. É de notar que logo no início deste predicado, todas as tags são alteradas pelo seu sinónimo. Após, com base num findall, este predicado chama o predicado *common\_tags\_combinations*/3, que irá calcular todas as possíveis permutações de X tags. Logo após, será chamado o método *common\_tags\_users\_combination/3*, que, para além de servir como um iterador recursivo para as combinações, também remove as combinações de tags possuídas por menos de 2 jogadores. Este, por sua vez, chama o predicado *common\_tags\_users\_combination\_aux*/4 (Figura 9), que tem como objetivo encontrar utilizadores com X número de tags iguais às tags existentes na totalidade através de uma simples interseção. É também de notar que as tags dos utilizadores foram trocadas pelos seus sinónimos, dessa forma, ficando em coerência com as tags existentes já anteriormente trocadas. O método que permite que isto aconteça tem de nome *common\_tags\_change\_to\_synonyms (Figura 10).* Este é um simples método, que ao percorrer as tags dadas pelo parâmetro All\_Tags, verifica se essa tag tem um sinónimo associado, sendo que se tiver, executa uma union entre o sinónimo dessa tag e uma lista inicialmente vazia, e se não tiver um sinónimo associado, executa uma uma union entre a própria tag e a já falada lista.

Para a base de conhecimento completa (Figura 11) com sinónimos como tecnologia-jogos e teatro-musica, com os parâmetros: “2” como x e “L” como lista de retorno, este método retorna uma lista extensiva de combinações possíveis de tags com os jogadores que as contêm em 0.0s.

Uma imagem com texto

Descrição gerada automaticamente

Figura 8 - Common Tags

Uma imagem com texto

Descrição gerada automaticamente

Figura 9 - Common\_tags\_users\_combination\_aux

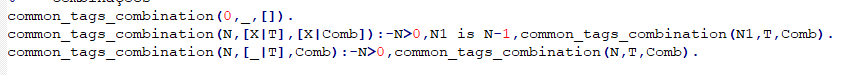


Figura 10 - Common\_tags\_combination

Uma imagem com texto, interior

Descrição gerada automaticamente

Figura 11 - Common\_tags\_change\_to\_synonyms

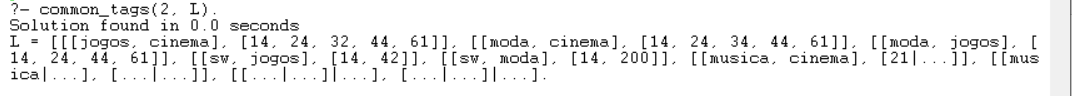


Figura 12 - Common\_tags example

# Sugestão das Conexões com outros Utilizadores tend por base as Tags e Conexões Partilhadas Até um determinado Nível

## Funcionalidade Desenvolvida por Jéssica Alves (1190682) e Pedro Santos (1190967)

O predicado de sugerir utilizadores (*suggest\_players*/3) tem como parâmetros: o id do jogar atual, o nível pretendido e a lista de utilizadores sugeridos, respetivamente. Este obtém as sugestões através dos seguintes passos: em primeiro lugar, ele irá buscar a rede utilizando o método desenvolvido noutra funcionalidade; de seguida, irá chamar um predicado que remove os amigos do jogador atual da lista retornada no método anterior; após a remoção, este irá chamar um predicado que irá percorrer recursivamente a lista retornada e identificar os jogadores que tem *tags* em comum com o utilizador e filtrar a lista conforme; por último, irá chamar um predicado que irá verificar para cada utilizador com *tags* em comum da sua rede, quais apresenta um caminho que contém uma *tag* em comum. Por exemplo, se uma das *tags* em comum for “música”, este vai tentar encontrar um caminho na rede em que todos os nós intermédios tenham essa *tag*, no caso de haver várias *tags* em comum, este irá pesquisar separadamente para cada *tag*, não aceitando caminhos com mistura de *tags*, a pedido do cliente (como podemos verificar na Figura 10).

Na Figura 11, temos o resultado da pesquisa feita na base de conhecimento completa e fornecida pelo cliente, com o “1” (ana) como jogador atual, “2” como nível e “L” como lista de retorno com as sugestões. Nesta rede todos os utilizadores do nível enviado têm a *tag* “natureza”, resultando a lista da figura e encontrando caminhos para os utilizadores lá representados. Para melhor teste desta funcionalidade (Figura 12), os autores recomendam a utilização da base de conhecimento parcial onde, para o segundo nível, ele não aceita um dos utilizadores que não tem *tags* em comum e, para outro utilizador, procura um caminho alternativo apesar deste ter caminhos sem *tags* em comum.

Uma imagem com texto

Descrição gerada automaticamente

Figura - Sugerir Utilizadores



Figura - Sugestão de Utilizadores Resultado



Figura - Sugestão (Base de Conhecimento Parcial)

# Determinação do Caminho Mais Forte

## Funcionalidade desenvolvida por Tiago Costa (1191460)

Nesta funcionalidade, o predicado chamado irá ser o *strongest\_route*/3. Os parâmetros para este predicado serão, em primeiro lugar, o nome do jogador atual (a origem do caminho); em segundo lugar, o nome do jogador objetivo (o destino do caminho retornado); por último, o terceiro parâmetro será referente ao caminho retornado e preenchido pelo predicado, sendo este caminho o caminho com maior número de força de ligação total e biderecional. Este predicado chama o predicado *strongest\_findRoute*/2, que tem como objetivo encontrar (através da chamada a outros predicados) o caminho mais forte e atualizá-lo no facto dinâmico *strongest\_currentRoute*/2, bem como após determinar um caminho, através do *fail*, fazer *backtracking* (Figura 8) e ir buscar caminhos alternativos que ainda não tinha determinado, comparando os com o melhor caminho atual no facto dinâmico através do predicado *strongest\_updateRoute*/2. Este método irá chamar um *dfs* modificado especificamente para o cálculo do caminho mais forte. Este predicado irá ter outro predicado auxiliar recursivo para determinar o caminho mais seguro, representado na Figura 9.

Para a base de conhecimento completa (Figura 10), com os parâmetros: “ana” como jogador atual, “eduardo” como jogador objetivo e “L” como lista de retorno, este método retorna um caminho com 118 de força de ligação total e que irá percorrer o caminho retornado na Lista “L”, com um tempo de execução de 52.59s. Com este caminho podemos verificar que este método não seria bastante prático no mundo real pois para uma ligação direta como “ana” e “eduardo” (representação gráfica na Figura 4), ele irá retornar um caminho bastante longo, que num contexto real, não seria prático, mas foi este o requisito do cliente para este projeto. Em suma, o método retorna no quatro parâmetro o caminho com maior força de ligação total que tem forças de ligação bidirecionais.

Uma imagem com texto, interior

Descrição gerada automaticamente

Figura 16 - Backtracking e fail (Caminho mais forte)

Uma imagem com texto

Descrição gerada automaticamente

Figura 17 - Dfs modificado para o caminho mais forte

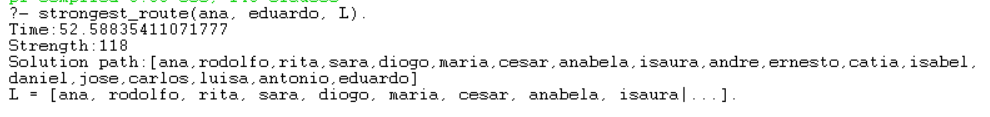


Figura 18 – Resultado do caminho mais forte na base de conhecimento completa

# Determinação do Caminho Mais Curto

## Funcionalidade Desenvolvida por Jéssica Alves (1190682)

A funcionalidade de determinar o caminho mais curto, apresenta duas opções de escolha: a primeira, que utiliza o predicado *findall*/3, sendo este mais ineficiente, estando este algoritmo presente no predicado *shortest\_allDfs*/3 (Figura 8); e a segunda, recomendada pelos autores, que é chamada no predicado *shortest\_route*/3 que não utiliza o findall, sendo bastante mais eficiente do que o previamente descrito. Ambas as soluções recebem como parâmetros o id do jogador atual (origem), o id do jogador objetivo (destino) e a lista de retorno do caminho calculado, respetivamente. Para calcular o caminho, os predicados utilizam um predicado shortest\_*dfs*/3 que determina o caminho com base no menor número de ligações, percorrendo cada nó e obtendo um caminho até a origem, com uma condição de paragem quando o nó atual for o nó referente ao jogador objetivo. Depois irá, através do *fail* e de *backtracking*, comparar os caminhos obtidos e verificar qual deles tem menor número de ligações, ou seja, que lista tem menor tamanho e guardar no facto dinâmico *shortest\_currentRoute*/2, sendo esta verificação e atualização feita no predicado *shortest\_updateRoute*/2. A diferença de eficiência entre as duas soluções é estudada no capítulo da complexidade e aí pode-se verificar a diferença entre os dois predicados.

Utilizando a base de conhecimento completa, com os parâmetros: “1” (ana) como jogador atual, “200” (sara) como objetivo e “L” como lista de retorno do caminho, obtém-se como seria de esperar, o caminho [ana (1), rodolfo (51), rita (61), sara (200)], como podemos verificar na Figura 9. Em contrapartida, na Figura 10, podemos verificar que utilizando o predicado que usa *findall*/3, para a mesma rede, a *stack* ultrapassa o limite.

Uma imagem com texto

Descrição gerada automaticamente

Figura - Caminho Mais Curto Com Findall



Figura - Caminho Mais Curto Resultado

Uma imagem com texto

Descrição gerada automaticamente

Figura - Caminho Mais Curto Resultado com Findall

# Determinação do Caminho Mais Seguro

## Funcionalidade Desenvolvida por Pedro Santos (1190967)

Nesta funcionalidade, o predicado chamado irá ser o *safest\_route*/4. Os parâmetros para este predicado serão, em primeiro lugar, o id do jogador atual (a origem do caminho); em segundo lugar, o id do jogador objetivo (o destino do caminho retornado); depois é requerido o limite mínimo para a força de ligação em ambas as direções (bidirecional), descartando caminho com valor inferior a este; por último, o quarto parâmetro será referente ao caminho retornado e preenchido pelo predicado, sendo este caminho o caminho com maior número de força de ligação total e biderecional que cumpre os requisitos de força mínimos. Este predicado chama o predicado *safest\_findRoute*/3, que tem como objetivo encontrar (através da chamada a outros predicados) o caminho mais seguro e atualizá-lo no facto dinâmico *safest\_currentRoute*/2, bem como após determinar um caminho, através do *fail*, fazer *backtracking* (Figura 11) e ir buscar caminhos alternativos que ainda não tinha determinado, comparando os com o melhor caminho atual no facto dinâmico através do predicado *safest\_updateRoute*/2. Este método irá chamar um *dfs* modificado especificamente para o caminho mais seguro, tendo em conta o limite de força de conexão. Este predicado irá ter outro predicado auxiliar recursivo para determinar o caminho mais seguro, representado na Figura 12. Como podemos verificar, este tem em conta as conexões bidirecionais requisitadas pelo enunciado e verifica, para cada uma delas verifica se a força de ligação é superior ao limite.

Para a base de conhecimento completa (Figura 13), com os parâmetros: “1” (ana) como jogador atual, “11” (antonio) como jogador objetivo, “2” como limite minimio de forca de ligação e “L” como lista de retorno, este método retorna um caminho com 67 de força de ligação total e que irá percorrer o caminho retornado na Lista “L”, com um tempo de execução de 0.0s. Com este caminho podemos verificar que este método não seria bastante prático no mundo real pois para uma ligação direta como “ana” e “antonio” (representação gráfica na Figura 4), ele irá retornar um caminho bastante longo, que num contexto real, não seria prático, mas foi este o requisito do cliente para este projeto. Outro exemplo, será a execução do caminho entre “1” (ana) e “200” (sara) com limite mínimo de 4, pois este não encontra caminho devido ao facto de existir uma ligação para chegar ao nó de destino que tem uma força menor que o limite fornecido (Figura 15), não retornando caminho e informando o utilizador disso (Figura 14). Em suma, o método retorna no quatro parâmetro o caminho com maior força de ligação total que tem forças de ligação bidirecionais acima do limite fornecido no terceiro parâmetro.

Uma imagem com texto

Descrição gerada automaticamente

Figura - Backtracking e Fail (Caminho Mais Seguro)

Uma imagem com texto

Descrição gerada automaticamente

Figura - DFS modificado para o caminho mais seguro



Figura - Resultado do caminho mais seguro na base de conhecimento completa



Figura - Exemplo de caminho não encontrado devido à falta de forças de ligação inferiores ao limite

Uma imagem com texto, laranja

Descrição gerada automaticamente

Figura - Conexões entre "ana" e "sara" com força menor que 4

# Estudo da Complexidade do Problema da Determinação de Caminhos

## Caminho Mais Curto Bidirecional (sem Findall)

|  |  |  |  |
| --- | --- | --- | --- |
| Nº de Camadas Intermédias (sem nós de origem e destino) | Nº de Nós por camada | Nº de soluções | Tempo para gerar a solução com menor nº de ligações |
| 1 | 1 | 1 | 0.0s |
| 2 | 2 | 8 | 0.0s |
| 3 | 3 | 891 | 0.06989s |
| 4 | 4 | 8184064 | 134.873291s |
| 5 | 5 | - | Mais de 7h |

Resultados acima de cinco camadas intermédias ultrapassam as 7h de execução, tendo o grupo terminado este estudo neste número.

## Caminho Mais Curto Bidirecional (com findall)

|  |  |  |  |
| --- | --- | --- | --- |
| Nº de Camadas Intermédias (sem nós de origem e destino) | Nº de Nós por camada | Nº de soluções | Tempo para gerar a solução com menor nº de ligações |
| 1 | 1 | 1 | 0.0s |
| 2 | 2 | 8 | 0.0s |
| 3 | 3 | 891 | 0.011994s |
| 4 | - | - | *Stack Limit* |

Resultados acima de quatro camadas intermédias chega ao *stack limit*, tendo o grupo terminado este estudo neste número.

## Caminho Mais Seguro (Limite Menos Restrito)

|  |  |  |  |
| --- | --- | --- | --- |
| Nº de Camadas Intermédias (sem nós de origem e destino) | Nº de Nós por camada | Nº de soluções | Tempo para gerar a solução com menor nº de ligações |
| 1 | 1 | 1 | 0.0s |
| 2 | 2 | 8 | 0.0s |
| 3 | 3 | 891 | 0.007994s |
| 4 | 4 | 8184064 | 147.586855s |
| 5 | 5 | - | Mais de 10h |

Para este estudo foi utilizado um limite mínimo de -10, ou seja, para testar a componente de encontrar os caminhos todos. Resultados acima de cinco camadas intermédias ultrapassam as 10h de execução, tendo o grupo terminado este estudo neste número.

## Caminho Mais Seguro (Forças de Ligação Positivas)

|  |  |  |  |
| --- | --- | --- | --- |
| Nº de Camadas Intermédias (sem nós de origem e destino) | Nº de Nós por camada | Nº de soluções | Tempo para gerar a solução com menor nº de ligações |
| 1 | 1 | 0 | 0.0s |
| 2 | 2 | 0 | 0.0s |
| 3 | 3 | 4 | 0.0s |
| 4 | 4 | 734 | 0.061983s |
| 5 | 5 | - | Mais de 10h |

Para este estudo foi utilizado um limite mínimo de 0, ou seja, apenas os caminhos com forças positivas seriam considerados, assim testando a componente do limite. Os resultados com nenhuma solução existem devido a existirem caminhos, mas com forças inferiores ao limite, assim podemos verificar o algoritmo em ação. Resultados acima de cinco camadas intermédias ultrapassam as 10h de execução pois as conexões adicionadas tinham forças positivas, tendo o grupo terminado este estudo neste número.

## Caminho Mais Forte Bidirecional

|  |  |  |  |
| --- | --- | --- | --- |
| Nº de Camadas Intermédias (sem nós de origem e destino) | Nº de Nós por camada | Nº de soluções | Tempo para gerar a solução com menor nº de ligações |
| 1 | 1 | 1 | 0.0s |
| 2 | 2 | 8 | 0.0s |
| 3 | 3 | 891 | 0.01s |
| 4 | 4 | 8184064 | 101.18s |
| 5 | 5 | - | Mais de 10h |

# Conclusões

## Titulo

Lorem ipsum