**Universidade de Aveiro**

**DETI**

**Métodos Probabilísticos de Engenharia Informática**

**Turma P5**

João Figueiredo nmec: 98506

Rafael Amorim nmec: 98197

27/01/2022

****

Índice

[**Introdução** 3](#_Toc94365970)

[**Objetivo** 3](#_Toc94365971)

[**Ficheiros Fornecidos** 3](#_Toc94365972)

[Users.txt, ficheiro com a seguinte informação, esta separada por um ‘;’, em que cada coluna representa o seguinte: 3](#_Toc94365973)

[Friends.txt, ficheiro com a seguinte informação, esta separada por *tabs*, em que cada coluna representa o seguinte: 3](#_Toc94365974)

[**Ficheiro dados.m** 4](#_Toc94365975)

[Segmentos de código (Nomes) 4](#_Toc94365976)

[Segmentos de código (Shingles) 4](#_Toc94365977)

[Segmentos de código (Amigos) 5](#_Toc94365978)

[Segmentos de código (Similaridade entre utilizadores em base de amigos) 5](#_Toc94365979)

[Segmentos de código (Similaridade entre utilizadores em base dos seus interesses) 6](#_Toc94365980)

[**Ficheiro projeto.m** 7](#_Toc94365981)

[1 -> amigos (id, Namigos) 8](#_Toc94365982)

[2 -> semelhante (id, SimilarUsers); 8](#_Toc94365983)

[3 -> SearchName(nomes, MinHashSig, bf); 9](#_Toc94365984)

[4-> SimilarInterests(id, SimilarUsersInterests, nomes, Namigos) 10](#_Toc94365985)

[Conclusão 10](#_Toc94365986)

# 

# **Introdução**

Neste trabalho, abordamos os conceitos lecionados e praticados ao longo guião prático nº 4, este relacionado com funções de dispersão, Filtros de Bloom, MinHash e Similaridade de *Jaccard*.

# **Objetivo**

No fundo foi-nos proposto desenvolver em Matlab, algumas funcionalidades de uma rede social, mais especificamente no que diz respeito à Amizade e aos interesses que têm em comum para com um certo utilizador sendo estes identificados por um ID. Antes de começar a implementar, estruturámos o nosso trabalho em duas partes, uma delas com intuito de ler os ficheiros de entrada (“dados.m”) fornecidos pelo guião e com o objetivo de obter toda a informação que nos foi necessária. A outra, com intuito de ler as estruturas de dados guardadas pela primeira parte, esta última realiza a interação com o utilizador (“projeto.m”).

# **Ficheiros Fornecidos**

## Users.txt, ficheiro com a seguinte informação, esta separada por um ‘;’, em que cada coluna representa o seguinte:

1ª 🡪ID da respetiva pessoa;

2ª 🡪Nome da respetiva pessoa;

3ª 🡪Apelido da respetiva pessoa;

4ª e próximas 🡪Interesses pessoais da respetiva pessoa.

Exemplo:

1;Matilde;Torres Moura;Música;Leitura;História;Andebol;Ténis

…

1000;Bianca;Anjos Teixeira;Jogos;Gastronomia;Política;História;Escrita;Pintura;Ténis

Verificou-se que o número total de utilizadores é 1000.

## Friends.txt, ficheiro com a seguinte informação, esta separada por *tabs*, em que cada coluna representa o seguinte:

1ª 🡪ID do utilizador;

2ª 🡪ID do Amigo;

3ª 🡪dia de último contacto com o amigo;

4ª 🡪 mês de último contacto com o amigo;

5ª🡪 ano de último contacto com o amigo;

Exemplo:

000095 000856 05 01 2004

…

000908 000309 18 03 2017

# **Ficheiro dados.m**

Este ficheiro tem como objetivo processar toda a informação que futuramente seja necessária e guardá-la num ficheiro ‘.*mat’,* que será usada pela aplicação principal. Portanto, este ficheiro só necessita de ser corrido uma única vez.

## Segmentos de código (Nomes)

De modo a facilitar a compreensão do código criámos uma variável chamada ‘***nomes***’ com 1000 linhas e 1 coluna, onde guardámos os nomes completos de cada utilizador.

Uma imagem com texto

Descrição gerada automaticamente

## Segmentos de código (Shingles)

Uma das opções pedidas serve para procurar strings, ou seja, ver se a string inserida corresponde ou não aos vários nomes existentes, avaliando a distância de Jaccard.

Decidimos tomar o tamanho de cada shingle como 3, pois foi nos aconselhado um valor entre 2 e 5 e já, previamente, tínhamos obtidos bons resultados com o mesmo tamanho em exercícios do guião nº 4.

Para a construção da matriz de assinaturas decidimos usar um número de funções de dispersão igual a 150, pois quanto maior for o número de funções de dispersão mais corretos serão os resultados obtidos.

**Uma imagem com texto

Descrição gerada automaticamente**Passamos todos os nomes para minúsculas para obter os mesmos resultados para strings iguais em minúsculas ou maiúsculas.

## **Uma imagem com texto Descrição gerada automaticamente**Segmentos de código (Amigos)

Este código, retira a informação de todos amigos de cada pessoa e guarda-os numa tabela.

*Namigos*, representa uma tabela, na qual, cada linha representa o id de uma pessoa e cada coluna dessa mesma linha é preenchida com os ids dos vários amigos, que a pessoa possuí.

## Segmentos de código (Similaridade entre utilizadores em base de amigos)

Esta porção de código, é muito semelhante ao código usado na secção 4.3 do guião nº4, com adaptações a este trabalho. Foi desenvolvido um método MinHash para avaliar as similaridades e distâncias de Jaccard entre todos os utilizadores, 2 a 2, em relação às suas listas de amigos. Utilizámos 100 funções de dispersão, pois com um nº grande garantimos uma melhor exatidão e utilizámos a função *DJB31MA,* pois já previamente, na resolução do guião, obtemos bons resultados com a mesma.No fim, todas as distâncias de Jaccard entre todos os utilizadores, inferiores ao *threshold,* são guardadas numa tabela.

Uma imagem com texto

Descrição gerada automaticamenteUma imagem com texto

Descrição gerada automaticamente

## Segmentos de código (Similaridade entre utilizadores em base dos seus interesses)

Esta porção de código, é muito semelhante ao código usado na secção 4.3 do guião nº4, com adaptações a este trabalho. Foi desenvolvido um método MinHash para avaliar as similaridades e distâncias de Jaccard entre todos os utilizadores, 2 a 2, em relação aos seus interesses. Utilizámos 100 funções de dispersão, pois com um nº grande garantimos uma melhor exatidão e utilizámos a função *DJB31MA,* pois já previamente, na resolução do guião, obtemos bons resultados com a mesma.No fim, todas as distâncias de Jaccard entre todos os utilizadores, inferiores ao *threshold,* são guardadas numa tabela. Por fim, todas as estruturas de dados são guardadas, fazendo com que este script não precise de ser corrido mais nenhuma vez.

**Uma imagem com texto

Descrição gerada automaticamenteUma imagem com mesa

Descrição gerada automaticamente**

# **Ficheiro projeto.m**

Como já foi referido, este ficheiro serve de interação para com o utilizador.

Logo de início temos uma advertência ao utilizador para não se esquecer de correr primeiro o ficheiro ‘dados.m’ pois sem a estrutura lá guardada não é possível dar procedimento à aplicação.

Uma imagem com texto

Descrição gerada automaticamente

Depois de seguir este passo fundamental, temos um pedido de inserção do ID correspondente ao próprio utilizador, com as suas restrições em termos de limites.

Quando validado o ID, entramos no loop do menu até ser inserida a opção 5, senão, entramos nas outras opções que nos levam a certas funções específicas.

Uma imagem com texto

Descrição gerada automaticamente

Resultado:

Uma imagem com texto, interior, captura de ecrã

Descrição gerada automaticamente

Tratando agora das funções por ordem:

1 -> amigos (id, Namigos)

Com a matriz de doubles criada nos dados.m conseguimos obter o **número de amigos** somando todas as colunas da respetiva linha referente ao ID do utilizador que são diferentes de zero.

A partir daí imprimimos os IDs, nomes e apelidos associados aos amigos, através do ‘***dic***’ que nos foi fornecido no enunciado.

Resultado:

Uma imagem com mesa

Descrição gerada automaticamente

2 -> semelhante (id, SimilarUsers);

Começamos por percorrer os interesses do utilizador e guardamos num novo cell array, depois calculamos o mais semelhante (carregado dos dados.m), identificamo-lo e apresentamos os interesses dele.

Depois de conhecer os dois utilizadores e os seus interesses, apresentamos os interesses não partilhados como sugestões.

Uma imagem com texto

Descrição gerada automaticamente

3 -> SearchName(nomes, MinHashSig, bf);

Inicialmente pede-se para escrever uma String, e seguidamente são apresentados, no máximo, 5 nomes, por ordem crescente da distância de ‘**Jaccard**’ e deve apenas mostrar os nomes com distância Jaccard menor ou igual a 0.9, portanto se não aparecer nenhum nome surge uma mensagem a indicar que não foi encontrado nenhum semelhante (<= 0.90).

Esta função começa por obter os shingles da string introduzida, depois, a partir desses mesmos shingles, é criada uma matriz de assinaturas da string, de forma semelhante à usada para obter os utilizadores semelhantes. Para cada shingle da string é aplicado k-funções de dispersão e guardado os seus resultados num vetor hash. Depois comparamos com os resultados obtidos para os outros *shingles* e guardamos o menor valor correspondente a cada função de dispersão. Depois de obter o vetor minHash, vai ser calculado e guardado as distâncias de Jaccard entre a string e todos os nomes.

Antes de isto tudo, todos os nomes são inseridos num *Bloom Filter,* e posteriormente é verificada a pertença da string introduzida. O *bloom filter* criado, tem um tamanho de 5000(superior à quantidade de elementos inseridos, neste caso, 1000). Se a string pertencer ao *bloom filter*, aparece uma mensagem na consola a advertir para isso, ou seja, avisa que a string introduzida, pode ser um nome existente.

Uma imagem com texto

Descrição gerada automaticamente

4-> SimilarInterests(id, SimilarUsersInterests, nomes, Namigos)

Apresenta-se a lista de amigos do utilizador, depois pede-se que escolha um deles, para posteriormente, com o auxílio da tabela já criada com as distâncias de Jaccard entre todos, apresentar os 3 utilizadores mais semelhantes em termos de interesses.

Uma imagem com texto

Descrição gerada automaticamente

# Conclusão

Por fim, podemos concluir este trabalho, realizado com êxito. Todos os parâmetros pedidos, foram implementados e funcionam corretamente. Para adicionar, o conhecimento sobre funções de dispersão, filtros de bloom e similaridade e distância de Jaccard foi aprofundado.