

**Relatório**

**Algoritmos e Estruturas de Dados II**

**Aluno:**

**19432 - Carlos Santos**

**19433 - Rúben Silva**

**Professores: Alberto Simões;**

**Curso Técnico Superior Profissional <Engenharia de Sistemas Informaticos>**

Barcelos, Junho, 2020

Resumo

Faz um resumo

Índice de figuras:

Figura 1 - Teste 1 ............................................................................................................................ 12

Figura 2 - Teste 2 ............................................................................................................................ 12

Figura 3 - Teste 3 ............................................................................................................................ 13

Figura 4 - Teste 4 ............................................................................................................................ 13

Figura 5 - Teste 5 ............................................................................................................................ 14

Índice

[1. Introdução 5](#_Toc42310450)

[1.1. Contextualização 5](#_Toc42310451)

[1.2. Motivação e objetivos 5](#_Toc42310452)

[1.3. Estrutura do Documento 5](#_Toc42310453)

[2. Estado da arte 6](#_Toc42310454)

[3. Implementação 7](#_Toc42310455)

[3.1. Descrição do problema 7](#_Toc42310456)

[3.2. Solução 7](#_Toc42310457)

[4. Analise e Testes 12](#_Toc42310459)

[5. Conclusão 15](#_Toc42310460)

# Introdução

## Contextualização

A estruturação e organização de uma base de dados é essencial para que seja vários tipo de informações sejam guardadas de uma forma eficiente. Com isso, neste trabalho é pedido que seja desenvolvida uma estrutura de dados para guardar informações de actores e com quem estes contracenaram.

## Motivação e objetivos

Este trabalho tem como objetivo criar um programa capaz de analisar 2 ficheiros, 1 com dados dos atores e outro com as contracenações de cada ator e os seus respetivos identificativos(id) (ator x -> ator y). Recolhendo e analisando esta informação irá permitir que sejam apresentadas diferentes filtragens de dados e o algorítmo de dijkstra.

## Estrutura do Documento

O documento é composto por vários ficheiros, tanto de recolha de informação como de filtragem, e também pelo ficheiro principal *(main)*. Todos estes ficheiros encontram-se interligados através de um ficheiro *header.*

# Estado da arte

O processamento de Linguagem Natural é uma vertente da ciência da computação, inteligência artificial e da linguística que ajuda os computadores a entender e a manipular a linguagem humana. Este tipo de processamento procura preencher a lacuna entre a comunicação humana e o entendimento dos computadores.

As máquinas de hoje podem analisar mais dados baseados em linguagem do que os seres humanos de maneira consistente e imparcial. Considerando a quantidade enorme de dados não-estruturados que é gerada diariamente a automação será imprescindível para uma análise de texto e fala completa e eficiente.

Com o crescimento do uso de tecnologias com inteligência artificial, é cada vez mais importante que os dispositivos entendam os seus utilizadores, oferecendo experiências e respostas às suas necessidades. Os usos mais comuns do processamento de linguagem são nas plataformas de busca online, como por exemplo Google, visto que este precisa de entender o conteúdo que se encontra nas páginas que apresenta como resultado para uma melhor correspondência, este tipo de processamento é também usado nas assistentes virtuais e chatbots.

# Implementação

## Descrição do problema

Deve ser lido 2 ficheiros de texto que pode ter até milhões de linhas, e de seguida guardar a informação necessária numa hash para esta poder ser analisada e filtrada.

## Solução

# header.h

No ficheiro header.h é criada a estrutura de dados “Hash”(*CollList, EdgeList*), uma lista (*AuxRecord*) para o desonvolvimento do algoritmo de dijkstra e também são registadas todas as funções do programa.

**CollList:**

int *id*; -> Identificativo do ator

char \**name*, \**gender*; -> Nome e sexo do ator (respetivamente)

EdgeList \**edges*; -> Com quem o ator contracenou

Struct \_collList \**next\_coll*. -> Próxima collision

**EdgeList:**

int *target*; -> Identificativo do contracenador

struct \_edgeList \**next\_edge*. -> Próxima edge

**AuxRecord:**

int *node*; -> Identificativo do contracenador

int *connection*; -> Nao percebi

int *weight*; -> Peso do contracenador

struct \_auxRecord \**next\_rec*. -> Próximo record

* **main.c**

No ficheiro main.c apresenta um menu, a maioria das mensagens da consola e algumas recolhas de dados do utilizador (ex: “Id do ator a pesquisar”).

* **actorsInfo.c**

Neste ficheiro estão reunidas as seguintes funções:

int *hast\_entry* -> Encontrar a posição certa na Hash dado um identificador.

CollList \**hash\_update\_node* -> Encontra a posição na Hash para o node ser atualizado.

Inicialmente é criada a variável inteira *pos* que é igualada ao return da função *hash\_entry*.

Posteriormente, este variável irá permitir a manipulação da colisão certa.

CollList \**hash\_update\_node\_cl* -> Atualiza o node com novas informações.

Se existir a lista e se o id da lista for igual ao id necessário, o nome e o genero da lista são atualizados, senão e se a lista existir passa para o posição seguinte.

CollList \**hash\_search* -> Encontra a posição na Hash para um identificador poder ser pesquisado. Inicialmente é criada a variável inteira *pos* que é igualada ao return da função *hash\_entry*.

Posteriormente, este variável irá permitir a manipulação da colisão certa e será retornada a informação pretendida.

CollList \**hash\_search\_cl* -> Encontra o identificador na lista de incidencias.

Se existir a lista e se o id da lista for igual ao id necessário, retorna a lista. Se a lista existir passa para o posição seguinte caso não tenha sido retornado anteriormente.

CollList \*\**hash\_new* -> Reserva memória para uma nova Hash.

EdgeList \**insert\_incidence* -> Cria uma nova incidencia na Hash.

Inicialmente é reservado memória para a varíavel EdgeList \**new*. Esta variável irá receber o valor do *target* e a próxima posição que será igual à lista inicial. No fim, irá ser returnada.

CollList \**hash\_insert\_edge* -> Encontra a posição na lista de incidencia.

Se existir a lista e se o id da lista for igual ao id necessário, a edge da lista será igual ao valor returnado da função *insert\_incidence*. Senão, se a lista e o id for menor ao id necessário, a posição seguinte da edge da lista será igual ao valor returnado da função *hash\_insert\_edge\_cl.* Senão irá ser atribuidos valores de inicialização às variáveis. Por fim, a lista será returnada.

CollList \**hash\_indert\_edge\_cl* -> Cria uma nova edge na incidencia correta.

Inicialmente é criada a variável inteira *pos* que é igualada ao return da função *hash\_entry*.

Posteriormente, este variável irá permitir a manipulação da colisão certa.

* **Functions.c**

Neste ficheiro estão reunidas as seguintes funções:

void *readTxtActores* -> Lê os 2 ficheiros sobre os atores.

São inicializadas variáveis para receber os valores e os dados presentes nos ficheiros.

É criado 2 ciclos que irão ler e recolher os valores dos ficheiros .txt.

Os dados do ficheiro co-actors.txt serão processadas na função *hash\_insert\_edge* e os dados do ficheiro actors.txt serão processadas na função *hash\_update\_node* e no fim estas informações todas irão ser armazenadas na hash.

int *count\_names* -> Percorre a Hash para fornencer dados à função *count\_names\_cl*.

É criada uma variável que é responsável pela soma dos valores do returnados da função *count\_names\_cl* que é ciclada até ao fim da hash. No fim returnará esse valor para ser apresentado.

int *count\_names\_cl* -> Conta o número de atores dado um nome pelo utilizador.

É criada uma variável que é responsável pela contabilização de quantos vezes o nome do node da lista é igual ao dado pelo utilizador. Dentro do ciclo responsável pela rodagem da lista toda, também é apresentado quem é o ator que tem o mesmo nome.

No fim é returnado a contabilização.

void *only\_womans* -> Percorre a Hash para fornencer dados à função *only\_womans\_cl*.

void *only\_womans\_cl* -> Obtem os atores que só contracenaram com mulheres.

Inicialmente, é criada uma collision list auxiliar e um inteiro *verif* para ser usado como “true” or “false”.

É percorrida a lista de edges toda e a cada node é verificado a os dados do id recolhido e verificado se este indivíduo é Masculino, Feminino ou desconhecido, caso não seja feminino, *verif*=1 e break. Se *verif*=0 (quer dizer que todos são femininos), irá ser mostrado na consola o nome e o id do ator em questão.

void *mostACT* -> Percorre a Hash para fornencer dados à função *mostAct\_cl* e mostra o ator com mais contracenações.

Inicialmente é são criadas 2 Collision List, uma auxiliar = hash e outra para guardar os dados do ator com mais contracenações.

A Hash será completamente ciclada e sempre que o valor returnado da função “*mostACT\_cl*” for maior que a variável inteira *big*, irá ser alterado o valor de big sendo este igual ao returnado da função e os dados do ator serão guardados na collision list reservada para o caso.

int *mostACT\_cl* -> Obtem a quantidade de contracenções de cada ator e retorna.

Inicialmente é criada uma edge auxiliar e um contador. A edge auxiliar irá ser igual à edge do ator e irá ser ciclada até a lista acabar. Cada ciclo incrementa 1 no contador. No fim, o contador será returnado

AuxRecord \**new\_aux\_record* -> Cria um novo node na lista.

Inicialmente é reservado memória para a varíavel AuxRecord \**new*. Esta lista irá ser inicializada. No fim, irá ser returnada.

AuxRecord \**aux\_rec\_search\_id* -> Verifica se o identificador existe na lista.

A lista será toda ciclada e se o node da lista for igual o numero necessário, então break, senão irá a lista irá returnar.

AuxRecord \**aux\_rec\_sorted\_insert* -> Insere ordenadamente um node.

Inicialmente é reservado memória para a varíavel AuxRecord \**new* que será responsável por receber os novos valores,

Estes valores serão inseridos consoante na lista, consoante o valor da variável “*weight*”. Se este valor for o menor, então será criado um node novo na lista com os valores.

No fim, a lista será returnada.

void *aux\_rec\_list\_free* -> Limpa a lista.

EdgeList \**path\_find* -> Algoritmo de dijkstra (encontrar o caminho mais rápido entre relações).

Inicialmente são criadas 4 listas do tipo AuxRecord, 1 Collision List, 1 Edge List e 2 inteiros.

AuxRecord \**current* = NULL; -> Explica o porque de cada variável

AuxRecord \**destinationRecord*;

AuxRecord \**closed* = NULL;

AuxRecord \**open* = *new\_aux\_record(origin);*

EdgeList \**path* =NULL;

CollList \**actor*;

int *costSoFar*, *destination*.

Enquanto a lista *open* existir, a lista *current* será igual à *open*, a lista *open* passará para posição seginte e a posição seguinte da lista *current* será zerada. Se o node da lista *current* for igual ao *target*, break. A collision list *actor* irá receber os dados da função *hash\_search*.

Para cada edge da Collision list da variável *actor*:

*Destination* será igual ao *target* da edge do *ator*, *destinationRecord* será zerada. Se existir return da função *aux\_rec\_search\_id(closed,destination)* , a função continua e então, o *costSoFar* será igual ao peso do node da lista *current* +1. A lista *destinationRecord* irá receber as informações returnadas da função *aux\_rec\_search\_id(open, destination).*

Se esta exisitr, verifica se o *costSoFar* é maior ou igual ao peso do node da lista *destinationRecord*, se isto acontecer, então continua. Se não existir, *destinationRecord* irá receber o valor da função *new\_aux\_record(destination).*

O peso do node da lista *destinationRecord* será igual ao *costSoFar* e a *connection* do node da lista *destinationRecord* será igual ao valor do node da lista current.

Por fim, a lista *open* será igual ao valor de *aux\_rec\_sorted\_insert(open, destinationRecord).*

Depois do ciclo da *actor*->*edges*, a lista *closed* irá receber o valor de *aux\_rec\_sorted\_insert(closed, current).*

No final de tudo, se o valor do node da lista *current* for diferente do *target*, retorna NULL. A lista do *path* irá receber o valor de *insert\_incidence(path, current->node*) e enquanto o valor do node da lista *current* for diferente da origem, o *path* receberá valores *da função insert\_incidence(path, current->connection)* e a lista *current* receberá valores *aux\_rec\_search\_id(closed, current->connection).*

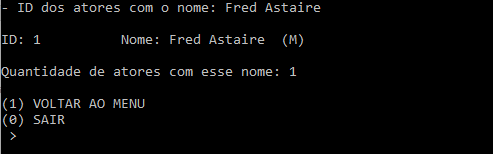
Antes de returnar o *path*, as listas *open* e *closed* são limpas.

# Analise e Testes

Usando como teste o maior ficheiro foram realizados testes. Como cobaia, foi utilizado a informação do ator com o identificador -> 0000001 (Fred Astaire). (Em alguns casos, os resultados serão parcialmente mostrados pois a lista é extensa com centenas ou milhares de linhas.)

**Obter Infos Ator por Nome:**

Esta opção mostra a quantidade de atores com o nome do ator dado pelo utilizador.



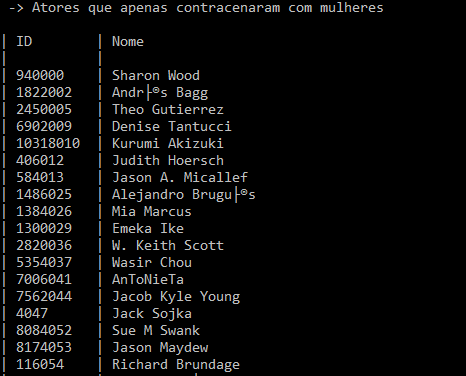
**Obter contracenadores por ID:**

Esta opção mostra com quem o ator ,dado pelo utilizador, contracenou.



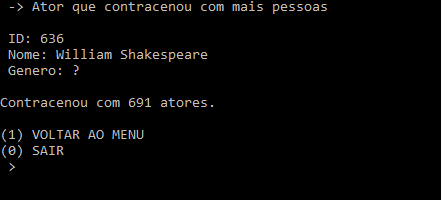
**Apresentar atores que contracenaram apenas com mulheres:**

Estão opção permite a filtragem dos atores que contracenaram apenas com mulheres.



**Ator que contracenou com mais pessoas:**

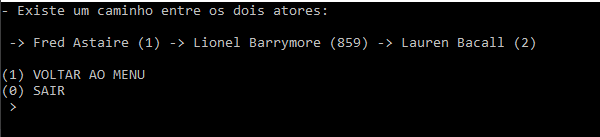
Esta opção permite filtrar o ator que contracenou com mais pessoas.



**Obter relação entre dois atores:**

Esta opção permite a visualização do caminho mais curto entre contracenações dado 2 identificadores pelo utilizador.

Neste caso específico, é a relação entre o id -> 0000001 e o id -> 0000002.



# Conclusão

O programa apresenta todos os requisitos e encontra-se visualmente agradável e de forma organizada.

Com este trabalho prático, um novo tipo de estruturas de dados, a hash, foi posto em prática através da leitura de ficheiros, cujos dados, sem limite definido, devem ser armazenados sem limite definido.

A eficiência do programa depende do tamanho do ficheiro, no entanto, foi melhorada ao máximo para que o tempo de leitura não seja demasiado grande.

Bibliografia

**https://www.sas.com/pt\_br/insights/analytics/processamento-de-linguagem-natural.html**

**https://take.net/blog/devs/nlp-processamento-linguagem-natural**