Relatório

Algoritmos e Estruturas de Dados II

Aluno:
19432 - Carlos Santos
19433 - Rúben Silva

Professor:

Alberto Simões

Licenciatura em Engenharia de Sistemas Informaticos

Barcelos, junho, 2020

Resumo

Neste Trabalho Prático foi requisitado que a partir de um par de ficheiros de texto, representativos da relação entre atores que tenham trabalhado no mesmo filme, se construísse um grafo simples que represente corretamente a contracenação entre estes atores.
A partir deste grafo foram apresentados vários problemas para resolver.

Índice de figuras:

Índice

1. Introdução

1.1. 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.

1.2. 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.

1.3. 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.*

2. Implementação

2.1. Descrição do problema

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

2.2. 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 género do ator (respetivamente)
EdgeList **edges*;	->	Com quem o ator contracenou
Struct _collList **next_coll*.	->	Próxima collision

EdgeList:

int *target*;	->	Identificativo de outro ator
struct _edgeList **next_edge*.	->	Próxima edge

AuxRecord:

int *node*;	->	Identificativo do contracenador
int *connection*;	->	Ligação ao próximo ator da lista
int *weight*;	->	"Custo" até chegar a este node
struct _auxRecord **next_rec*.	->	Próximo record

- **main.c**

No ficheiro main.c é apresentado 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* -> Dado um identificador, obtém a chave correta para encontrar a sua posição na Hash.

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 a 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 *readTxtActors* -> 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;

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.

3. Análise 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.

```
- ID dos atores com o nome: Fred Astaire

ID: 1          Nome: Fred Astaire  (M)

Quantidade de atores com esse nome: 1

(1) VOLTAR AO MENU
(0) SAIR
 >
```

Obter contracenadores por ID:

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

```
 -> Contracenadores de um ator

 - ID do ator a pesquisar: 1
Ator 1: Fred Astaire
 Contracenou com:  818700   | Leonard Spigelgass          | ?
 Contracenou com:  70361    | Robert Benchley             | M
 Contracenou com:  752528   | Elaine Ryan                 | ?
 Contracenou com:  790654   | Arthur Sheekman             | ?
 Contracenou com:  656712   | Janis Paige                 | F
 Contracenou com:  852313   | Dwight Taylor               | ?
 Contracenou com:  1571     | Kim Novak                   | F
 Contracenou com:  804026   | Red Skelton                 | M
 Contracenou com:  314826   | Leonard Gershe              | ?
 Contracenou com:  2147     | Tab Hunter                  | M
 Contracenou com:  173679   | Betty Comden                | F
 Contracenou com:  1077     | Richard Crenna              | M
 Contracenou com:  491076   | S.K. Lauren                 | ?
 Contracenou com:  122675   | George Burns                | M
 Contracenou com:  791084   | Sidney Sheldon              | ?
```

Apresentar atores que contracenaram apenas com mulheres:

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

```
 -> Atores que apenas contracenaram com mulheres

| ID         | Nome
|            |
| 940000     | Sharon Wood
| 1822002    | Andr├®s Bagg
| 2450005    | Theo Gutierrez
| 6902009    | Denise Tantucci
| 10318010   | Kurumi Akizuki
| 406012     | Judith Hoersch
| 584013     | Jason A. Micallef
| 1486025    | Alejandro Brugu├®s
| 1384026    | Mia Marcus
| 1300029    | Emeka Ike
| 2820036    | W. Keith Scott
| 5354037    | Wasir Chou
| 7006041    | AnToNieTa
| 7562044    | Jacob Kyle Young
| 4047       | Jack Sojka
| 8084052    | Sue M Swank
| 8174053    | Jason Maydew
| 116054     | Richard Brundage
```

Ator que contracenou com mais pessoas:

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

```
 -> Ator que contracenou com mais pessoas

 ID: 636
 Nome: William Shakespeare
 Genero: ?

Contracenou com 691 atores.

(1) VOLTAR AO MENU
(0) SAIR
 >
```

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.

```
- Existe um caminho entre os dois atores:

 -> Fred Astaire (1) -> Lionel Barrymore (859) -> Lauren Bacall (2)

(1) VOLTAR AO MENU
(0) SAIR
 >
```

4. 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.

5. Bibliografia

https://www.sas.com/pt_br/insights/analytics/processamento-de-linguagem-natural.html

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