Hanoi University of Science & Technology
School of Information & Communication Technology

Final project report

Data structure and Algorithms advanced lab

Project: Movie recommendation

Instructor: Dr. Dao Thanh Chung

Team members:

Hoàng Mạnh Hà	20184250
Lê Anh Tuấn	20184322
Bùi Thanh Tùng	20184324
Phạm Thành Đạt	20184242

Table of Contents

1. Problem formulation:

In the year of 2020, Covid-19 has virtually prevented all human's outdoor activities from happening. One of various ways to keep oneself out of boredom is by watching movies. Among different components, the recommendation system plays an important role in many successful platforms such as Netflix, HBO and Amazon.

2. Solution:

We decided to make a replica of Netflix with minimal functionalities. An application is built from scratch that stores the data of movies like actors, directors,... in a database and based on that, provides user functions to find their wanted movies or actors by name, movies and a movies recommendation list for them to choose to watch later.

2.1. Data structures used:

2.1.1. Main data structures:

- struct MovieEntry;
    + Represents a movie in the database.
- struct DirectorEntry;
    + Represents a director in the database.
- struct CastEntry;
    + Represents a cast in the database.
- struct GenreEntry;
    + Represents a genre in the database.
- struct Database;
    + Represents the whole database
- struct SimilarityEntry
    + Stores a similar movie with its score
- struct SimilarityList;
    + Stores a list similar movies

2.1.2. Data structures in details:

```
typedef struct _MovieEntry
{
  int internal_id;

  const char* title;
  int show_id;
  enum MovieType type;
  int release_year;
  int duration;

  const char* description;

  int director_count;
  struct DirectorEntry** directors;

  int cast_count;
  struct CastEntry** casts;

  int genre_count;
  struct GenreEntry** genres;
} MovieEntry
```

- The first few fields are self-explanatory.
- ***directors*** is an array of ***DirectorEntry****
- ***casts*** is an array of ***CastEntry****
- ***genres*** is an array of ***GenreEntry****

```
typedef struct _DirectorEntry
{
  int internal_id;

  const char* name;
  // Value -> MovieEntry*
  Dllist movies;
} DirectorEntry;
```

- ***movies*** contains all movies which this director directed.

```
typedef struct _CastEntry
{
  int internal_id;

  const char* name;
  // Value -> MovieEntry*
  Dllist movies;
} CastEntry;
```

- ***movies*** contains all movies in which this actor/actress acts.

```
typedef struct _GenreEntry
{
  int internal_id;

  const char* name;
  // Value -> MovieEntry*
  Dllist movies;
} GenreEntry;
```

- ***movies*** contains all movies which belong to this genre.

```
typedef struct _Database
{
  int movies_capacity;
  int movies_count;
  MovieEntry* movies;

  int directors_capacity;
  int directors_count;
  DirectorEntry* directors;

  int casts_capacity;
```

```
    int casts_count;
    CastEntry* casts;

    int genres_capacity;
    int genres_count;
    GenreEntry* genres;

    // Key -> name, value -> MovieEntry*
    JRB movies_name_lookup;
    // Key -> name, value -> DirectorEntry*
    JRB directors_name_lookup;
    // Key -> name, value -> CastEntry*
    JRB casts_name_lookup;
    // Key -> name, value -> GenreEntry*
    JRB genres_name_lookup;

    SimilarityList* movies_similarity;
} Database;
```

- A database consists of a list of directors/movies/casts/genres. It also contains 4 JRB trees which provide a mapping from names to an Entry.

2.2. Application of required topics:

- JRB (Red Black Tree): provides fast movies/directors/casts lookup by name.
- JRB is also used as a HashSet (ensure uniqueness of elements).
- Dllist (Double Linked List): used as a resizable array.
- Sorting of large data: sort recommended movies by score

3. Functions:

3.1. Overview:

This project aims at the users that want to watch movies on an online movie application or website for watching movies like Netflix...

3.2. User functions

- Search for movies by name
- Search for casts by name
- Add movies to watched list
- Find recommendations from list of watched movies

3.3. Functions implementation

3.3.1. Database loading

Since the file is in .csv format, we have to use a library written in C++ to read data. All the data processing and storing after that is done in C.

3.3.2. Search for entities by name

The JRB library provides an efficient way to find any entity by name using ***jrb_find_str()***. Unfortunately, an exact match is required. Search by keywords is implemented by *traversing* a list of entities and applying *strstr()* to match substrings.

3.3.3. Manage watched list

A watched list is stored as a Dllist inside ***App*** structure. This list contains internal ids of watched movies. You can add, remove, show the content of the watched list.

3.3.4. Search for movies featuring actor/actress

Using the id of the actor/actress, we can search for the movies which they feature in.

3.3.5. Find recommendations from list of watched movies

For each watched movie, we extract top 10 similar movies (similarity score is calculated based on common directors/genres/casts). Then all similar movies are mixed in the final round, resulting in a recommended list.

The process of score calculating is done in a bruteforce fashion. Each pair of movies is assigned a score indicating how similar they are. 3-way quicksort is then used to retrieve top 10 movies with highest scores. See ***generate_similarity_list*** in ***database.c*** for more information.