

Warwick+

CS126 Design of Information Structures

Luis Armada 2202567

Term 2 Coursework Report

Department of Computer Science

University of Warwick

Contents

1	Data Structures	1
1.1	Choices and Justifications	1
1.2	ArrayList Implementation	1
1.3	LinkedList Implementation	1
1.4	HashMap Implementation	2
1.5	Queue Implementation	2
1.6	Graph Implementation	2
2	Ratings.java	3
2.1	Data Structure Justification	3
2.2	Implementation	3
2.3	Result	3
3	Credits.java	5
3.1	Data Structure Design	5
3.2	Implementation	5
3.3	Result	5
4	Movies.java	8
4.1	Data Structure Design	8
4.2	Implementation	8
4.3	Result	8

1 Data Structures

1.1 Choices and Justifications

This report will cover the development and implementation of new data structures for **Warwick+**, a Java application which stores and displays a large amount of film-data. The classes being worked on are Ratings, Credits, and Movies. Each of these store and utilise data differently, therefore implementations will differ, however this report aims to explain and justify the choices made for each class.

The classes Ratings, Credits, and Movies all store and use data differently, however similarities between the classes can be observed. For example, almost all 'getter' functions in each of the classes pass in a unique ID as a parameter. In this situation, a hashmap would be best suited to store the data as it has a time complexity of $O(1)$, which will have significant effects when it comes to working with massive sets of data.

Additionally, the use of ArrayLists will be beneficial as it enables the dynamic storing of data of the same type under one identifier. This will prove useful when it comes to adding films, cast members, ratings, etc, as the actual number of data objects to store is unknown.

More data structures used in this project are covered below, as well as how and why they were implemented.

1.2 ArrayList Implementation

The ArrayList.java class essentially stores three important variables which make up the ArrayList structure.

1. Object[] array: This is the actual array of objects being stored.
2. Size: An integer which is equal to the number of objects in the array.
3. Capacity: The length of the array - note that this is different to size as some indexes are empty (null).

The functions in the ArrayList class work by transferring the contents of the array to a new one with a different length if necessary. For example, if the user wants to add a new element to the array but $\text{Size} = \text{Capacity}$, then a new array is made with, say, $\text{Capacity} + 100$.

The remove function works by shifting elements after the removed item down the array to fill in the space.

There is also a function called "addX" which allows for insertion of an element into a specified index, and it works by shifting elements in front up by one.

1.3 LinkedList Implementation

A linked list is a data structure in which elements contain pointers which point to the next element in the list. The only pointer stored in a variable is the head, and from the head the other elements can be accessed.

The linked list plays a vital part in the implementation of the Hashmap. It is used to overcome collisions of hashes, which would occur if only an array was

used. Instead, each hash would point to a linked list so that the appropriate value can be returned.

1.4 Hashmap Implementation

A hashmap stores entries into linked lists (called buckets) based on a hash value. It returns the direct memory reference in integer form, allowing for $O(1)$ access. This time complexity is a major advantage especially for large data sets, which is why its implementation is important.

It will be used in the storage classes as it uses ID's, which are unique values - perfect for keys in a hashmap. The value partnered to the key can then refer to any data necessary in relation to the unique ID.

1.5 Queue Implementation

The implementation for a queue becomes simple if the ArrayList is used. As a queue uses a FIFO (First In First Out) structure, all it requires is using the 'addX' function in the ArrayList at index 0 to enqueue, and removing and returning elements at the last index to dequeue.

The queue will be used for breadth first searching in a graph, which is essential for finding relations between two cast members for example.

1.6 Graph Implementation

Simply put, a graph is a collection of graph nodes. A graph node is a subclass of the graph class, and it contains an ArrayList of graph nodes which store adjacent nodes. Functions in the graph node class allow for adding and removing from the list.

The graph class contains functions for adding new graph nodes and it stores them in a hashmap with the ID as the key and the graph node class as the value. Adding a new graph node requires to set adjacency, ensuring that the adjacency matrix is always correct.

The graph itself will be used to display relationships between cast members, for functions such as finding distance between them.

2 Ratings.java

2.1 Data Structure Justification

As the name of the class describes, the ratings class aims to store a large amount of user ratings for various movies. The ratings themselves consist of a rating, which is a *float* value from 0 to 5 inclusive, and a timestamp, which is a *Calendar* value denoting when the rating was posted.

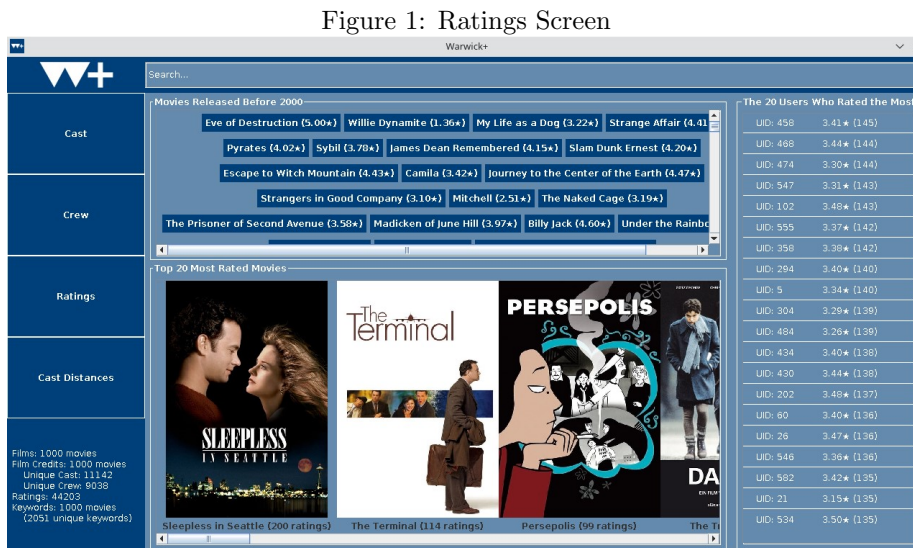
This allows for the use of a Hashmap which stores an ID for a key and a RatingData class as an object. This RatingData class will store all ratings for the given ID as well as provide other useful functions such as add, remove and find average of ratings. This will allow for easy access for ratings for users and movies as it only requires the ID. Additionally, hashmaps have $O(1)$ time complexity for getting values, which makes it more efficient for larger sets of data.

2.2 Implementation

This section will cover the functions and how the data structures mentioned are used to provide efficient functionality. The 'GetRatingsBetween' functions, which return ratings between a certain time period, are done using an ArrayList which contains ID's (which refer to ratings in the hashmaps) already stored in chronological order. This is achieved by placing the element in the ArrayList in order, and moving it when a rating for an ID is added. This means that all that is required in the function, is to convert the ArrayList to an array for n required elements.

Similarly, 'getTopRatings' functions work by using an ArrayList which inserts elements into order when a rating is added. Other functions such as 'getUserRating' or 'getAverageRating' just require getting from the hashmap and returning a value.

2.3 Result



3 Credits.java

3.1 Data Structure Design

The credits class stores information about the cast and crew members of a film. Like ratings, as films, cast and crew members have unique ID's, a hashmap will be used for relating an ID to a value. In this class, this value will be another class for each respective entity. The Film class will contain arrays of cast and crew members. The CastData and CrewData class will both contain ArrayLists consisting of movieID's they partook in. These classes will also have built in functionalities for use in the Credits functions.

This class will also use a graph, which is to create connections between cast members for use in the findDistance function.

Lastly, ArrayLists to store the IDs of entities (films, cast, crew). An ArrayList is used over an array because number of entities is unknown.

3.2 Implementation

Like Ratings, this section will cover the functions and how the data structures are used to complete them. The 'add' function, like before, adds to the hashmaps as well as ArrayLists containing the ID's. Additionally, cast members are added to a graph using the add function implemented in the graph class.

Other functions such as getting ID's only involves fetching from the hashmaps.

'FindCast/Crew' functions involve accessing the respective data classes and comparing the string parameter to the name to decide which members to output. Similarly, for 'findStarCast', cast members are looped through and a counter is used whilst iterating through their movies to determine if they are a star or not. The same is done with superstars, except it uses findDistance to compare stars.

3.3 Result

Figure 2: Cast Screen



Warwick+

Search...

Cast

Cast from top rated movies
Marty | Nicholas Lyndhurst | Emily Hirst | Suzanne Lloyd | Susie Lambert | Ken Crou | Angela Thorne | Peter Halliday | Peter Wright | Ian Mercer

Between Eleven and Midnight
Louis Jouvet | Madeleine Robinson | Robert Amoux | Leo Laparra | Monique Mélinand | Gisèle Casadesus | Yvette Etrévant | Jacqueline Pierreux

Crew

The Haunted World of El Superbeasto
Rosario Dawson | Danny Trejo | Shen Moon Zombie | Paul Giamatti | Cassandra Peterson | Dee Wallace | April Winchell | Laraine Newman | Jess Harnell | Harland Williams | Joe Alaskey | Ken Foss | Tom Papa | Rob Paulsen | Debra Wilson | Tom Klein | Clint Howard | Brian Posehn | Tom Kenny | Dee Bradley Baker | Geoffrey Lewis | John DiMaggio | Jeff Bennett | Carlos Alazraqui | Tira Satana | Bill Mosley | Rodger Bumpass | Sid Haig | Charlie Adler | Kevin Michael Richardson | Daniel Roebuck

Ratings

Gozu
Yûta Sone | Sho Aikawa | Kimika Yoshino | Shôhei Hino | Kellô Tomita | Harumi Sone | Renji Ishibashi | Kenichi Endô | Kanpei Hazama | Masaya Kato | Tamio Kawai | Susumu Kimura | Hiroyuki Nagato | Hitoshi Ozawa | Kazuyoshi Osawa | Sakichi Satô | Takatoshi Shôta | Tetsuro Tambo | Yoshiyuki Yamaguchi

Cast Distances

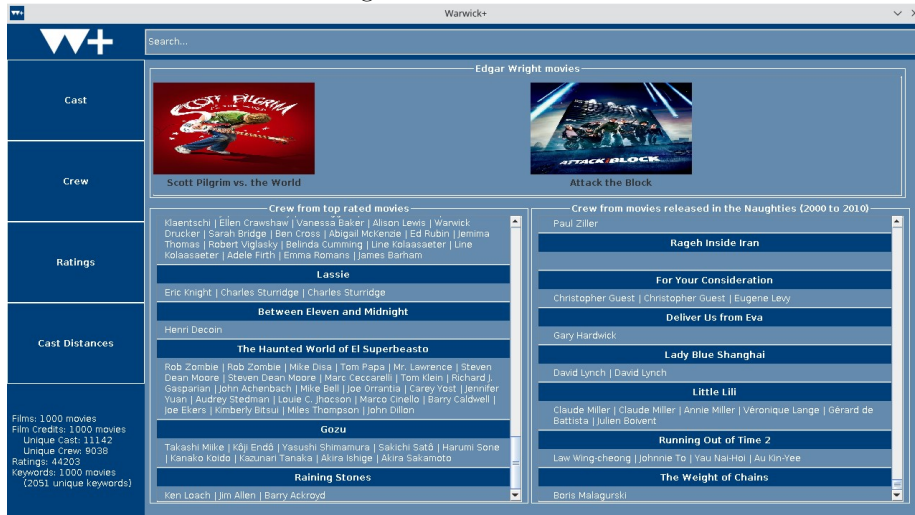
Raining Stones
Bruce Jones | Julie Brown | Gemma Phoenix | Ricky Tomlinson | Tom Hickey

Tom Hanks Movies

Extremely Loud & Incredibly Close | Saving Private Ryan | Sleepless in Seattle | Cast Away | Who Kill

Films: 1000 movies
Film Credits: 1000 movies
Unique Cast: 11142
Unique Crew: 9038
Ratings: 44203
Keywords: 1000 movies
(2051 unique keywords)

Figure 3: Crew Screen



Warwick+

Search...

Cast

Edgar Wright movies
Scott Pilgrim vs. the World | Attack the Block

Crew

Crew from top rated movies
Klaentschi | Ellen Crawshaw | Vanessa Baker | Alison Lewis | Warwick Drucker | Sarah Bridge | Ben Cross | Abigail McKenzie | Ed Rubin | Jemima Thomas | Robert Viglasky | Belinda Cumming | Line Kolassaeter | Line Kolassaeter | Adelle Firth | Emma Romans | James Barham

Ratings

Lassie
Eric Knight | Charles Sturridge | Charles Sturridge

Between Eleven and Midnight
Henri Decoin

Cast Distances

The Haunted World of El Superbeasto
Rob Zombie | Rob Zombie | Mike Dissa | Tom Papa | Mr. Lawrence | Steven Dean Moore | Steven Dean Moore | Marc Ceccarelli | Tom Klein | Richard J. Casapian | John Achenbach | Mike Bell | Joe Orlando | Carey Post | Jennifer Yuan | Audrey Stegman | Louie C. Jaccson | Marco Cinello | Barry Caldwell | Joe Ekers | Kimberly Bitzui | Miles Thompson | John Dillon

Gozu
Takashi Mike | Kôji Endô | Yasushi Shimamura | Sakichi Satô | Harumi Sone | Kanako Kôido | Kazunari Tanaka | Akira Ishige | Akira Sakamoto

Raining Stones
Ken Loach | Jim Allen | Barry Ackroyd

Crew from movies released in the Naughties (2000 to 2010)
Paul Ziller

Rageh Inside Iran
Christopher Guest | Christopher Guest | Eugene Levy

For Your Consideration
Gary Hardwick

Deliver Us from Eva
David Lynch | David Lynch

Lady Blue Shanghai
Claude Miller | Claude Miller | Annie Miller | Veronique Lange | Gérard de Battista | Julien Boivent

Little Lili
Law Wing-cheong | Johnnie To | Yau Nai-Hoi | Au Kin-Yee

Running Out of Time 2
Boris Malagurski

The Weight of Chains

Films: 1000 movies
Film Credits: 1000 movies
Unique Cast: 11142
Unique Crew: 9038
Ratings: 44203
Keywords: 1000 movies
(2051 unique keywords)

Figure 4: Cast Distances Screen



4 Movies.java

4.1 Data Structure Design

The last class, Movies, uses a similar approach where it uses a hashmap with a Movie and Collections class for values. These classes contain a bunch of metadata (variables) regarding the movie. It uses two hashmaps, one for movies and one for collections to store data given a unique film ID. Additionally, all film ID's are stored in an ArrayList as opposed to an array as again, number of films are unknown.

4.2 Implementation

The functions in this class are not as complicated as the previous classes, as most of them are just getter functions. All this requires is getting the relevant ID from the hashmap and returning the requested values.

The 'getAllIDsReleasedInRange' works similarly, as it iterates through the ID's stored in the ArrayList, to compare the release dates stored in the hashmap. 'FindFilms' works the same way, instead just comparing strings to determine if the titles contain a given search term.

Lastly, adding to a collection is simple. Given a filmID as a key in the CollectionsData hashmap, the collectionID can be used as a value so that the given film can always be correlated to it's respective collection.

4.3 Result

Figure 5: Movie Data Screen

