BSc Computer Science Year 2

Software Engineering Management and Development

Report for Movie and TV Streaming Service, “Streamflix”

Project team members: Noor Mamdouh, Anakha Jayakumar, Joshua Wynter, Afsheen Siddiqui, Gabriela Campoverde Tacuri, Adeyinka Adesanya

Roles:
Scrum master: Noor Mamdouh
Developers: Joshua Wynter, Afsheen Siddiqui
Secretary: Gabriela Campoverde Tacuri
Testers: Adeyinka Adesanya, Anakha Jayakumar

Date of submission: 21/04/2024

Introduction

Brief description of the project
We were tasked with creating a project based off of an idea assigned at random, a movie and TV streaming service, in which we had to select an appropriate data structure and algorithm, with justification on why we selected it and appropriate analysis of the time complexity, to utilise in our project, all whilst ensuring we commit to Scrum methodology and proper planning.

Report Layout:
Our report is layed out with a brief description of our project, and what we intend to create, brainstormed from our initial planning meeting, a further in depth plan of the functionality we intend to create, and the data structure that we have chosen, alongside our chosen algorithm. Next, we have listed the steps we have decided to take for the initial sprint, from our initial meeting.
We have justified the reason for our chosen data structure and have calculated the time complexity for the algorithm with appropriate justification under the design section.
We have explained the testing approach we used and included a table of test cases in the testing section.

Brief project idea:

We intend to create a movie streaming service responsible for providing the latest movies and TV shows at the viewer's convenience and comfort. Basic premise is a website that prompts the user to login or register, checks whether they've purchased the premium plan, and then enables the user to choose a show to watch, and upon watching saves the show to the already watched or watch later list of movies and tv shows.

Detailed breakdown of plan:

First the system will prompt login and registration, then checks whether they're already a member of the streaming service premium plan. If not, they will be prompted to purchase a subscription after registration. After purchasing or logging in with premium access, the user can then choose what shows they'd like to watch from an array of options, upon selecting an option, they will be directed to

the link that allows them to view their film or tv show and have the options to favourite. After clicking on a movie or tv show the show or movie is then added to a “watch later” 3d array/vector with the duration watched, or an “already watched” list if they’ve completed the entire duration of the film or show.

Intended project steps:

-Create welcome menu GUI which prompts user to either login or register
-Start filling in CSV file with list of registered users and their purchased plan
-Create an array of movies and shows in terms of genres.
-Create CSV file could include data such as title, genre, release year, duration and actors
-Create a class for users, plan types and video options, with related classes for tv shows and movies. And load csv file into them appropriately
-Class members will be genre, release year, duration and actors

For registration:
-User has the option of 2 plans, either limited options (TV shows and movies) or unlimited access to all shows and movies
-Use user’s registered with premium access to check against new registered accounts.

Design

Selected data structure: Hash table

Justification:

We opted for hash tables as our primary data structure for three key reasons. Firstly, their quick access speed ensures efficient retrieval and insertion of data, important for managing large datasets like we would have with a list of movies and TV shows upon future development of our project. Secondly, hash tables' hashing techniques allow for quick and precise searching, removing the hassle of sifting through any unnecessary information. Moreover, their smart memory usage ensures optimal space utilisation, important for projects where memory efficiency is very important. We chose hash tables as they offer a perfect blend of speed, efficiency, and memory management, making hash tables the ideal data structure for our project's needs i.e. quickly retrieving movies and TV shows for the users convenience.

Time complexity: In our Streamflix project, we considered utilising hash tables due to their efficient time complexities for fundamental operations such as search, insertion, and deletion. Typically, hash tables offer an average-case time complexity of O(1) for these operations, meaning that they can execute in constant time. This efficiency stems from the hash table's ability to access elements directly through their keys, using a hash function that maps keys to specific indices in the table. However, it's important to note that in the worst-case scenario, these operations can degrade to O(n), where 'n' is the number of elements in the table. This occurs when the hash function causes many keys to collide at the same index, leading all entries to be concentrated in a single chain or bucket, thus requiring linear search within the chain. To mitigate this, an effective hash function and techniques to manage collisions are crucial. Moreover, maintaining a low load factor through appropriate sizing and resizing of the hash table can also help sustain the desired O(1) performance for most operations, making hash

tables highly suitable for scenarios in our project where quick data retrieval and management are necessary. (Cormen, Leiserson, Rivest, Stein, 1989)

Selected algorithm: Bubble sort

Justification: (Some ideas: easy to implement, less storage space required and a stable algorithm, which means that it preserves the relative order of equal elements found in the array)

We decided to use bubble sort as our algorithm for a couple of reasons. Bubble sort is a linear sorting algorithm that is used to sort items adjacently by swapping the items if they are not in the correct order. This algorithm is easy to implement and understand allowing us to gain a greater understanding making it easy to incorporate in our code. Additionally, Bubble Sort operates in-place, requiring no additional memory beyond what is used for the original vector. This is beneficial for our project's resource constraints. As Streamflix expands and our database grows, we plan to evaluate the efficiency of our sorting method and consider adopting more efficient algorithms that can handle larger volumes of data effectively.

Time complexity: we acknowledged that Bubble Sort's time complexity is O(n^2), where 'n' is the number of items being sorted. This quadratic time complexity means that the algorithm performs well with small datasets, which is the case for our initial movie collection. Although Bubble Sort is less efficient for large datasets compared to faster algorithms like Quick Sort or Merge Sort, which have average time complexities of O(n log n), the size of our data during the project's inception does not justify the complexity of implementing more advanced algorithms. (Karumanchi 2011); (Sedgewick, Wayne, 2010)

Pseudo code for the above mentioned algorithm,

```
procedure bubbleSort(A : list of sortable items)
   n := length(A)
   repeat
      swapped := false
      for i from 1 to n-1 do
         if A[i] > A[i+1] then
            swap A[i] and A[i+1]
            swapped := true
         end if
      end for
      n := n - 1
   until not swapped
end procedure. (Fullyunderstood.com, 2019)
```

Testing Approach :

Waterfall Approach: traditional approach, testing is conducted sequentially after each phase of development.

Agile Approach: Agile methodologies emphasise iterative development and testing. Testing is integrated throughout the development process, with frequent releases.

Exploratory Testing: in this approach simultaneous learning of test design, and test execution is involved.

Test cases

User Registration:

1. Test registering a new user with valid input (username, email, password).
2. Test registering a user with an invalid email format to ensure that email validation works.
3. Test registering a new user with a username or email that already exists in the system.

User Login:

1. Test logging in with valid username and password.
2. Test logging in with an invalid username.
3. Test logging in with an incorrect password.

User Existence Test Case:

1. Test checking for the existence of a user with a valid username and password.
2. Test checking for the existence of a user with an invalid username or password.

Select Plan Test Cases:

1. Test selecting the basic plan.
2. Test selecting the premium plan.
3. Test selecting an invalid plan option.
4. Test selecting a plan by entering '1' or '2'.
5. Test entering an invalid choice and ensure the system prompts the user to enter a valid choice

Display Movies Test Cases:

1. Test to display movies from a valid CSV file.

Display TV Shows Test Cases:

1. Test to display tv shows from a valid CSV file.

Error Handling Test Cases:

1. Test handling errors when opening CSV files.
2. Test handling errors when reading data from CSV files.

Edge Cases:

1. Test the application's behaviour when the user tries to register/login without providing any input.

ID	Test Name	Description	Expected results	Status
Test case1	Registration	Users need to fill username, email,password	Successful registration	Passed

Test case 2	Login	Username and password as used while registering	Login successful!Welcome Username!	Passed
Test case 3	Tvshow-test	Testing Tvshow class,testing for getSeasonCount,Testing for getEpisodeCount	Expected to return the number of seasons of the tv show and the total number of the tv show as provided in the test data vector.	passed
Test case 4	Movies test	To test the movie class and its functionality,to test the getters which get the method of the movie class.testing to print the display method the string stream captures the printing to the consoles and instead of an output.	Ensuring the return of the correct values Ensures it stores the movie details and print them in the expected format	passed
Test case 5	Plan test	To test that the system correctly allows users to select between a basic plan and a premium plan, and that the corresponding plan details are displayed accurately.	successfully prompts the user to select a plan and displays the corresponding plan details based on the user's choice.If the user selects the Basic Plan , the system displays the details of the Basic Plan accurately. If the user selects the Premium Plan, the system displays the details of the Premium Plan accurately.	Passed

Time complexity

- The time complexity of the display TVShowsFromCSV function is **O(n),** where n is the number of TV shows in the CSV file. This is because the function reads each TV show from the file and displays its details one by one, resulting in a linear time complexity.
- The time complexity of the `readMoviesFromCSV` function is **O(n*m)**, where n is the number of lines in the CSV file and m is the average number of columns in each line. This is because the function reads each line from the file and then splits it into columns, resulting in a linear time complexity with respect to the number of lines and columns.
- The time complexity of accessing a specific movie data in the CSV file is **O(1)** because we can directly access the record using its unique identifier (Movie ID).

- The time complexity of this data structure would be **O(1)** for accessing specific tv show data in a CSV file, as each record can be accessed directly by its unique IDs.

- The time complexity of reading TV shows from a CSV file is **O(n),** where n is the number of lines in the file. This is because as reading each line of the file sequentially and processing it to extract the respective information from the line of the code

- The time complexity of the readShowsFromCSV function is **O(n),** where n is the number of TV shows in the CSV file. This is because the function reads each line of the file and creates a TVShow object for each line.

- The time complexity of the Plan classes is **O(1)** for the constructor, totalPrice(), and isAvailable() methods. This is because these methods perform a constant number of operations regardless of the size of the input. This means that the run time will always be the same regardless of the input size.

- The time complexity of the code provided is **O(1)** for all the getter functions. This is because each getter function simply returns a member variable of the Movie class, which can be accessed in constant time.

- The time complexity of the displayInfo() method is **O(1)** because it has a constant number of operations to display the information of the movie.

- The time complexity of the DataValidation is **O(1)** because the number of iterations in the do-while loop is fixed and does not depend on the input size.

- The time complexity of this Makefile is **O(n),** where n is the number of source files being compiled. As each source file needs to be separately compiled , and the Makefile specifies rules for compiling each source file individually and separately.

- The time complexity of viewMoviesBasedOnCategory function is **O(1)** because it performs a fixed number of operations regardless of the size of the input.in this function there are no loop or recursive calls depending on size of the inputs.

Conclusion

Including
– Summary of work done.
– Limitations and critical reflection (including what caused the limitations)
– How we would change our approach on similar tasks in the future

We worked together to create the code to perform our given project idea, a movie streaming service, with our collectively chosen data structure and algorithm. We incorporated both hash tables and our bubble sort algorithm.
This involved creating a CSV file for users, movies, TV shows and subscription plans along with a header file for each one. We have ensured that users are able to register and sign in to their account. Upon doing this, they need to select a subscription plan that best suits them and will only then be able to proceed.
Once they select their plan, they are able to choose between what TV show and/or movie they want to watch. This will be differentiated with a “watch later” 3d array/vector that users will be able to add

into if they would like to watch anything later. In addition to this, if there is a show or movie that has been watched by the user completely, the show/movie is added to the “already watched” list.
Some of the limitations we encountered was the code not being able to read the CSV file even though we spent days trying to find out what was wrong. Another problem was with bubble sort, we couldn’t use hash tables. In order to solve this problem, we had to convert into vectors which made us lose some time.

In the future, to avoid making the same or any similar mistakes, we would make sure to use a function that converts hash tables into vectors or just use vectors outright to avoid any unnecessary confusion.

Citations:

Work Cited

“Hash table | Fast database access to hash values.” *Ionos*, 30 January 2023, https://www.ionos.co.uk/digitalguide/server/security/hash-table/. Accessed 15 April 2024.

Kundan *et al.* (2024) *Bubble sort algorithm, code, advantages*, *Learn Coding Anywhere Anytime -PW Skills Blog*. Available at: https://pwskills.com/blog/bubble-sort-algorithm-code-advantages/ (Accessed: 18 April 2024).

"Introduction to Algorithms" by Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein

"Algorithms" by Robert Sedgewick and Kevin Wayne

"Data Structures and Algorithms Made Easy" by Narasimha Karumanchi

Nikam, Indrajeet. “Bubble Sort.” *Fully Understood*, 9 September 2019, https://fullyunderstood.com/pseudocodes/bubble-sort/. Accessed 21 April 2024.