TED University

CMPE223 Data Structures & Algorithms / 2021 Fall

Assignment 3 Report

Information

Name & Surname: **Sencer Yücel**

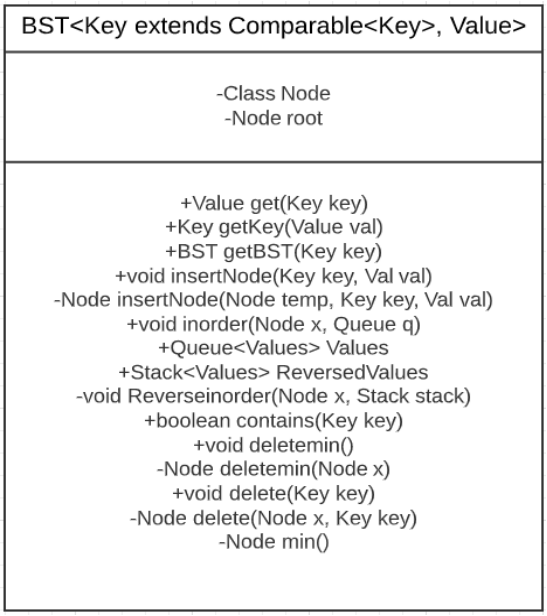
ID: **16933103152**

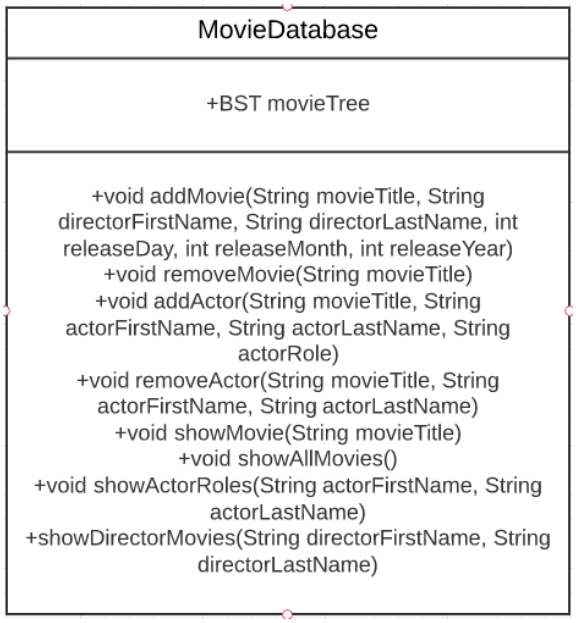
Section: **02**

Problem Statement and Code Design

I used a binary search tree to create a movie database for this assignment. I added movies to the database, searching for specific movie information, eliminating movies, and so on. I needed to generate a new binary search tree for each movie that contains the actors. I have introduced actor and delete actor methods to this binary search tree. If the movie technique is not already in the database, I must have added it. I should overwrite it if it's there. I must utilize the are showing all the films in the database and printing them in ascending order for the showAllMovie method. "—none— " should be used if there is no movie in the database. If the movie is in the database, the removeMovie method removes it; else, print "ERROR: There is no movie called (movieTitle)". I should have discovered the actors binary search tree class that I introduced to the node and add actors to it using the addActor method. It is not possible to add an actor who has already been defined in the movie. I must have performed the same thing with removeActor as I did with removeMovie, however, this timeI am only deleting the actor from a specific movie. I must have expressed all of the actors, roles, movie, day/month/year of a specific movie in set standards for the showMovie technique. Find all the movies and their roles in them for showActorRoles and express them. I needed to express the movies that they filmed with a given director for showDirectorMovies.

I have 5 classes for this task. 2 of them are basically Stack and Queue implementations which we have already learnt and worked on, that is why I am going to write down the attributes and methods which belongs to other 3 classes: **BST, MovieDatabase** and **main.** However, **main** has only few lines of codes which I copied from the pdf, so the remaining 2 classes are as follows:





Implementation and Functionality

I am starting with the BST class, then I will explain the methods of MovieDatabase class.

Value get(Key key): We are looking for our worth according to the BST principle. If our key is less than root, we recursively scan root.left. We're searching root.right recursively if our key is higher than ours. We'll return our key if it's the same as the root.key. We return null if there is no key in the tree.

Key getKey(Value val): The contains() function of String is used in this method. It returns our key if our value contains our key. We can do so because when we add actors, we're essentially adding the entire thing as value. It looks for all the nodes on the left side of the binary search tree first, and then all the nodes on the right side.

BST getBST(Key key): We're using the same technique we used in the get(Key key) method to search our binary search tree. The only difference is that we're returning our node's actor tree.

void insertNode(Key key, Value val): It's a function for calling the insertNode() method on a Node.

Node insertNode(Node temp, Key key, Value val): It compares our key values to the root node once more, and if they're higher, it invokes itself as root.left; if they're not, it invokes itself as root.right. If they are equal, the value is equalized with the new value, and our node is returned.

Queue<Values> Values(): It's a function for calling the inorder() method on a Node.

void inorder(Node x, Queue<Value> q): We use the traversal inorder technique to enqueue our values. We're calling inorder(x.left, Queue q) and enqueue recursively. Then we call recursively for the right nodes and enqueue them as well.

void ReverseInorder(): It works in the same way to our inorder method. Because we need to print them in descending order for some methods, we are pushing our value to the stack rather than enqueuing the queue.

boolean contains(Key key): It returns whether our get(key) method is null or not null, indicating whether it is inside or outside the tree.

void delete(Key key): In terms of the binary search tree rules, we're removing our value. First, we determine whether it has a right or left child. If not, return the node's opposite. If left and right are not null, we find the minimum using the min() method, then equalize the x's right link to deletemin(t.right), equalize x.left = t.left, and repeat these procedures.

Node min(Node x): We're looking for our left value recursively till it's null. After that, we'll return the node.

Now, let us look to our MovieDatabase class, which has all the methods that being asked in the pdf.

void addMovie: The key value for the binary search tree must be determined in this technique. Normally, we would choose our key based on the movie's release year, but I was bothered, so I added a small portion of their month to create a unique key value. Otherwise, I'll lose some information. Then we must determine whether or not the film is in the database. If not, we'll add the movie with all of the keys and values as a value. We print the relevant information if it's already defined.

void removeMovie: getKey() method is used in this method to obtain the value of our key. We'll put our movie here. The getKey() method returns the value of our key. "There is no movie called..." is printed if there is no key in the database. If there is, we will erase the movie using the delete technique.

void addActor: We'll use the getKey() method to get the value of our key in this method. We use our getBST method to find out our Node's actor binary search tree after we get our key value. When we get to our actors' tree, we'll start adding their characteristics. For our actor's tree, the actor's first name + last name is the key, and all values are values.

void removeActor: We're using getKey() to find our key value and reach our actor's tree, just like we did in the addActor method. We use the delete function of the binary search tree class to remove a specific actor from the actors' tree once we've reached our tree.

void showAllMovies: To get all of the movies in the database, we use the binary search tree Values() function in this method. If the queue is empty, "—none— " is printed. If this is not the case, we will dequeue the queue until it is empty, then print the required information in the specified sequence.

void showMovie: The getKey() method is used to obtain its key. After that, we'll go to the movie's values section and extract the essential information from there, according to the pdf. After that, we'll need to look at the movie's actor tree to figure out who's in it. We use the

getBST() function to find our actor tree, and then we use the Values() method to get the values. Following that, we will print the relevant information once more.

void showActorsRole: First, we'll get the reversed values from the movie binary search tree's ReversedValues() method. Then we pop the movie's value and take the movie title to acquire the key of our movie so we can use our getBST() function to get to the actors binary search tree. Then, for actors bst get our values, we use the reversedValue() function. After that, we check to see if any of our variables include the actor's first and second names combined. If it does, include the actor's name, the film's title, and the year it was released. Otherwise, don't print anything.

void showDirectorMovies: To acquire our reversed ordered data, we use ReversedValue methods in this section. We next pop our stack and print out the films and years till the stack is empty if its values meet the parameters.

Testing

In the testing part, I wrote some additional lines to the main method. I checked whether is there a director or actor named “İbrahim İleri” and “Kemal Kılıçdaroğlu”. After that, I controlled that if there is a movie named “Matrix 4” or not. Finally, I added some other movies than the pdf itself and used showAllMovies method. Nothing went wrong.

Final Assessments

For me, even the algorithms to create were not too easy, most challenging part (wasted my time) was to print the movies as wanted in pdf. showAllMovies method and showMovie(movieTitle) method wanted us to print movies in different ways, but the values for Node’s were same. I could not be able to solve it for a very long time, after I overcame it somehow. Other than this, even though the tree logic is not that hard to understand, to code it was not that easy as well. I worked so hard on my BST class to make everything correct. Well, in the end, I am proud what I have done.