CS2110 Homework 10 - Fall 2021
Actor Search - Trees

You will not need code from a previous assignment to complete this homework. You will use a tree data structure to solve a performance problem.

The Internet Movie Database (IMDB) is a company that manages and brokers information about video entertainment. One service provided by this company is the ability to search for actors and receive a list of that person's work. Unfortunately, there is a lot of this kind of data, as you might expect, and searches are consequently slow. You have been hired by IMDB to help them tune this service for better performance.

Once you have access to the source code, the problem is obvious. The IMDB data set of actors contains > 10M records and they are searching for actors using linear search. You make a mental note to make a meme for this ridiculous design decision once you are off the clock.

Your days in CS have prepared you for this problem, however; you know lists are optimized for keeping things in order, for adding and removing elements, and can be fast for maintaining a count. But lists are not so great for searching, which is the most frequent use of this particular data structure. You know what IMDB needs for this problem: **a Binary Search Tree (BST)**.

You conceive a custom BST data structure organizing all the actors by name, allowing the web services to perform binary search. You realize what you need to do first is construct a demonstration that will allow you to try linear search and binary search and demonstrate the performance advantage of the BST.

First, set up your project:

- Create a new Java project on Eclipse, and call it "Homework 10."
- Download and import the starting package for this assignment using the link below. This is a large zip file - almost 200 MB compressed. Expanded, it will be more than three times this size.

https://collab.its.virginia.edu/x/xWhYcS

Learning Goals

In this assignment, we will practice:

1. Use of a data structure
2. Experience the impact of an efficient algorithm

Getting Started

To save time and provide for a consistent implementation, some starter code has been provided for you. The starter package includes:

- `Actor.java` - The Actor Class
- `ActorQueryTool.java` - The class used for running queries against the data. It performs both a sequential search of IMDB AND a binary search using the binary search tree that you will complete (see below). It prints the time it takes to complete each search. You should try this out by searching for a variety of actors and actresses.
- `DataParser.java` - The class used for reading the IMDB data (in tab separated files) and loading the information into the list and tree data structures.
- `ActorTree.java` - This is the class you will complete.

It also includes a dataset of IMDB data. The dataset, named `name.basics.tsv` is **598MB** and contains **10,493,111** names.

You should copy the `.java` files from the `src` directory into your **src/** directory and the `name.basics.tsv` file into the main directory of your project (that is, outside of the src/ directory). Your Eclipse project should look similar to the following Figure 1.

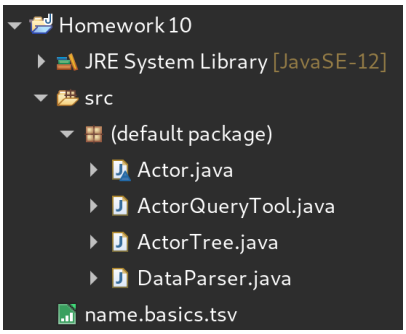

Figure 1. *Homework 10 project in Eclipse with all .java files and name.basics.tsv file.*

Implementing the Binary Search Tree

You must complete the implementation of the **ActorTree** and **Node** classes (in the ActorTree.java file) to provide a working Binary Search Tree.

ActorTree

This is the class that defines the Binary Search Tree.

- Complete the default constructor as needed.
- `public Actor find(String name)` - Finds an actor in the tree with the given name. Returns the matching `Actor` object or `null` if not found.
- `public int size()` - Returns the size of the tree, i.e., the number of nodes in the tree.

- `public int height()` - Returns the height of the tree.
- `public boolean isEmpty()` - Returns `true` if the tree is empty, `false` otherwise.
- `public boolean insert(Actor a)` - Inserts an actor into the tree. Returns `true` if the `Actor` object was inserted, `false` otherwise.
- `public String inOrder()` - Returns an in-order traversal of the tree. If the tree is empty, return a zero-length String (i.e., ""). If the tree is not empty, separate elements with a newline character ("\n").

Node

This is the inner class that defines the nodes of the Binary Search Tree. Remember that many of the functions in the `Node` class will be implemented recursively on the tree.

- Complete the default and overloaded constructors as needed.
- `public Actor find(String name)` - Finds an actor in the subtree rooted at this node with the given `name`. Returns the matching `Actor` object or `null` if not found.
- `public int size()` - Returns the size of the subtree rooted at this node, i.e., the number of nodes in the tree.
- `public int height()` - Returns the height of the subtree rooted at this node.
- `public boolean insert(Actor a)` - Inserts an actor into the subtree rooted at this node. Returns `true` if the `Actor` object was inserted, `false` otherwise.
- `public String inOrder()` - Returns an in-order traversal of the subtree rooted at this node. If the tree is empty, return a zero-length String (i.e., ""). If the tree is not empty, separate elements with a newline character ("\n").

Main Method (or JUnit) Testing

In this homework, we expect you to do your own testing before submitting to Gradescope.

Note: Your main method should have enough testing to provide sufficient evidence to determine that the behavior implemented matches the behavior described above. It is *highly recommended* to write the tests before implementation as it will help you to understand the exact behavior and what is the expected output of different inputs.

Our goal in this assignment is to encourage main method testing so that you know your code works well before submitting to Gradescope. Therefore, you will only be able to submit to Gradescope a total of 10 times for this assignment.

You can perform some tests on your code using the ActorQueryTool class. It provides a main() method that loads the IMDB data, creates a list data structure as well as the Binary Search Tree you implemented, then performs a search. The final static fields of the class can be modified

to determine how much data is loaded and which actor or actress to find. It prints out the time it takes to get the size of the data structures as well as finding the actor or actress. An *example* is shown below:

```
Data loaded, ready to query.
1000000 items in the list.
Calculating list size completed in 0ms
Found!
Actor [nconst=nm0933988, primaryName=Rainn Wilson, birthYear=1966, deathYear=\N,
primaryProfession=actor,producer,soundtrack,
knownForTitles=tt0465624,tt0467406,tt0386676,tt1512235]
Sequential search completed in 49ms
940847 items in the tree.
Calculating tree size completed in 43ms
Resulting BST has a height of 485
Found!
Actor [nconst=nm0933988, primaryName=Rainn Wilson, birthYear=1966, deathYear=\N,
primaryProfession=actor,producer,soundtrack,
knownForTitles=tt0465624,tt0467406,tt0386676,tt1512235]
Binary search completed in 0ms
```

Submission Information

Method and Class Naming: You must match method names, instance variable names, and data types exactly. You must use correctly formatted Java code. Declare fields in the class definition, and create a default constructor for each class that initializes every instance variable.

Coding Style: In real-world software development, it is paramount to create readable and easily maintainable code. That is typically achieved through the use of style and commenting guidelines. Since you will be updating this code over the next few weeks, we have provided a style guide and formatting guide that we strongly encourage you to follow:

- Coding Style Guide (includes installation instructions for Eclipse)
- Eclipse Style File

Submitting: Upload your Eclipse project (the `.java` file) to the "Homework 10 - Actor Search" assignment on Gradescope. You should submit `ActorTree.java`. **Do NOT submit your entire project. Do NOT submit the `name.basics.tsv` file!** This submission utilizes an autograder to check that your code follows these specifications. If it spots an issue, it will alert you, but you should **NOT** use the submission system as your testing. We encourage you to test your code during the implementation phase. Therefore, you may upload your code a **maximum of ten (10) times**.

Note: After the 10th submission, Gradescope will still allow submissions, but they will NOT be graded by the system.

Grading Rubric

The assignment is worth a total of 100 points:

15 points	- ActorTree and Node find methods
15 points	- ActorTree and Node size methods
15 points	- ActorTree and Node height methods
5 points	- ActorTree and Node isEmpty methods
20 points	- ActorTree and Node insert methods
15 points	- ActorTree and Node inOrder methods
15 points	- Code readability (organized, well-indented, readable by others)