Project 6: BST

A binary search tree to store a collection of values.

Overview

In this assignment, you'll start with a codebase for a binary search tree (BST), which has an ordering property that any node's left subtree keys ≤ the node's key, and the right subtree's keys ≥ the node's key. That property enables fast searching for an item. You'll need to implement several methods for the binary search tree code given to you.

Learning Goals:

- Demonstrate an understanding of binary trees, sufficient to implement several non-trivial methods.
- Apply forethought to determine if iterative or recursive methods are more straightforward in various cases.
- Learn and implement a simple extension to binary search trees based on a textual description of the extension.
- Use debugging and test code.

Project Specification

Code Structure.

The starter code contains the following files.

src: This is the source folder where all code you are submitting must go. You can change anything you want in this folder (unless otherwise specified), you can add new files, etc.

support: This folder contains support code that we encourage you to use (and must be used to pass certain tests). You must not change or add anything in this folder.

test: Feel free to add your own tests here.

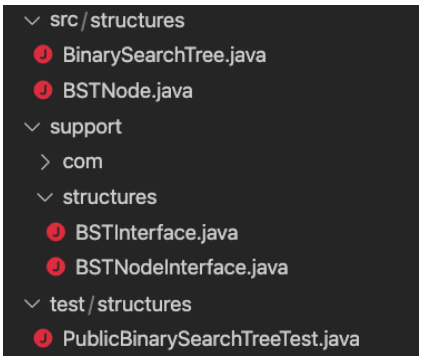

Fig 1: Code structure for the project.

You will first focus on finishing the implementation of the `BinarySearchTree` class, along with its supporting classes. Note that in the starter code:

- some methods return default or random values.
- the main methods and some tests cannot run as there are incomplete TODOs.

Tasks and TODOs

Complete `BinarySearchTree.java`

The project TODOs denote where you need to make changes in the source code.

Note: that you are allowed to use classes that implement the `Collection` class when needed. For example, you may use `java.util.Queue` in your implementations of the required iterators. You may also add private helper methods as required.

Review the code in `BinarySearchTree` and `BSTInterface`. We'll make the same assumptions for BSTs as ordered lists (no nulls, multiple copies of values allowed, etc.). We'll also mandate that BSTs must obey the BST rule in this problem. i.e. the ordering property that any node's left subtree keys ≤ the node's key, and the right subtree's keys ≥ the node's key.

Tasks

Your task is to implement the methods that are not yet implemented (that is, whose method bodies are marked with TODOs). In doing so, you may find that you need or want to change other methods. Note these restrictions:

1. Your changes must not break the semantics of the methods i.e. change the meaning of the java statement).
2. You may not change the signatures (header) of public methods, or add or remove such methods in the interface.

Carefully read the comments given in the `BSTInterface.java` class.
Some hints:

`reorderIterator and postorderIterator:` These methods are probably easiest to implement in terms of a Queue - see `inorderIterator` for an example. Recursively traverse the tree in the correct order, and insert each node into the queue as it is visited. Then, return the result of calling `iterator()` on the queue. The **`inorderTraverse`** method is given to you.

`equals:` This method is best expressed with a recursive helper method.

`sameValues:` Returns true if and only if the trees are value-equivalent. Note that two trees are value-equivalent if they contain exactly the same values. That means they contain the same number of elements. Duplicate values are allowed in a BST, so two trees are value-equivalent only if they both contain the same number of duplicate values.
Example: 1,2,2,3,4,5 is value-equivalent to 1,2,2,3,4,5, but not to 1,2,3,4,5.
Hint: Think about reusing the code from the equals method to implement this method.

You'll probably want to modify `BSTNode` to maintain a reference to the node's parent. You might add a get and set method for this reference, but you could also do this update when calling `setLeft` and `setRight`.

Here are some notes on some of the utility methods we've provided to you, that you should not (and in the case of **`getRoot`**, must not) change:

`getRoot:` This method returns the root node of the tree. Normally, you wouldn't expose such a detail in your implementation, but we require it in order to run some of the autograder tests. You may want to use it in your own testing, or with the following method.

`toDotFormat:` This method will output a representation of the tree rooted at the given node in the DOT language, as described by its left and right child references. There are many programs that can read this language and display the results. Here is a link you can use to visualize the output from the toDotFormat() method in Project 6: Graphviz Onlinehttps://dreampuf.github.io › GraphvizOnline.
Note that you will have to remove the final ";" from the output in order for the program to work. You should post your output in starting with "digraph G {" and end with "}".

Finally, note that you may want to allocate an array of generic (T) objects in your implementation of BinarySearchTree. If you use: `T[] array = (T[]) new Object[size()];` you'll get a `ClassCastException`. Why? Because T is no longer an unconstrained generic type. Its full signature is `T extends Comparable<T>`. To satisfy the JVM's run-time type constraints on arrays, you'll need an array capable of holding objects compatible with that type:
`T[] array = (T[]) new Comparable[size()];`

Testing

You'll note that for the problem, we have provided some tests. These are an absolutely minimal set of tests, and definitely do not cover all possible cases! Take the time to create some tests of your own. Try to think of all of the cases that could occur in each method you write, and write tests to check each of them.

Export and Submit

Step 1: Export your project
Within VSCode click on the "*View* > *Command Palette*…" menu option. Then type into the Command Palette: "*Archive Folder*" and hit enter. This will produce a Zip file of your project folder. You can then upload that zip file to the corresponding project assignment in Gradescope. You can add the Archive extension to VSCode if you don't have it.

Step 2: Submit the zip file to Gradescope
Log into Gradescope, select the assignment, and submit the zip file for grading.

The autograder will run successfully only if you submit a **correctly formatted zip file**. The autograder will also not run if your code **does not compile**, or if you i**mport libraries** that were not specifically allowed in the instructions.

Remember, you can re-submit the assignment to gradescope as many times as you want, until the deadline. If it turns out you missed something and your code doesn't pass 100% of the tests, you can keep working until it does.

Academic Honesty Policy Reminder

Copying partial or whole solutions, obtained from other students or elsewhere, is **academic dishonesty**. Do not share your code with your classmates, and do not use your classmates' code. Posting solutions to the project on public sites is not allowed. If you are confused about what constitutes academic dishonesty, you should re-read the course policies.