CS249 Project 4
Binary Search Trees

Overview

This project requires the completion of the following tasks:

- Write a Red-Black Tree
- Apply search trees to example problems

Unit Testing

This project requires that for each part you must run the included unit tests to validate and verify the correctness and integrity of your programs, as well as to drive a test-driven development approach. Remember that test-driven development means that test and compilation errors are not necessarily a bad thing; you should use them as an indication of you what code you need to create or fix next.

Start by downloading the interface and unit tests files provided for this assignment and setting-up JUnit for your specific Java development setup. Below are some quick tips for getting started running JUnit tests in several common development setups.

After you turn in your assignment you will be graded using these tests, as well as additional unit tests, so make sure you are thorough in your implementation.

IntelliJ:

- Create a new project and add the interface and test files to the *src* directory
- Open up one of the test files from within IntelliJ and click on one of the lines underlined with an error
- Press ***alt+enter*** (***option+enter*** on mac) and select the option to import the JUnit library into your project
- You should now be able to run each of the tests (***right-click*** the file and choose *run*), or all of the tests (***right-click*** on the project and choose to run all tests) once you create the classes required for this project

Eclipse

- Create a new project and add the interface and test files to the project *src* directory (you might need to ***right-click*** and refresh the *src* icon in Eclipse)
- ***Right-click*** on the project in the Package Explorer panel, go to ***Build Path->Add Libraries***, and select JUnit
- You should now be able to run the unit tests by ***right-clicking*** on the project or test files under Project Explorer, or by going to the ***run->run as*** menu

Command Line

For a command line development environment, you will need to perform several steps:

- Download the required JUnit JAR files to your project directory
- Add the JAR files to your project class path when compiling and running
- Create your own driver class (a class with a main method) to run the tests and print out the results

Binary Search Trees

Binary search trees (or BSTs) are a method for indexing data by creating a tree in which you can follow one branch of the tree for larger values, or follow the other to find smaller values. Each node of the tree contains a key as well as right and left branches which themselves contain BSTs, allowing quick searching.

There are many methods by which to add nodes to a BST. In this section we will consider a simple method in which new elements added to the search tree by searching for their key, and adding a node where ever their search fails. In the next section we will consider the more sophisticated insertion method of a Red-Black Tree.

1. ***Implement a BinarySearchTree class satisfying the interface provided in IBinarySearchTree.java.***
    - When inserting, create a new node wherever a search would expect the inserted element to be found.
    - Ensure you can pass all BinarySearchTreeTest.java unit tests.
2. ***Profile your BinarySearchTree, comparing a randomized and an ordered addition of elements.***
    - Profile by adding the numbers 1 to 10,000 in a random order and then timing the search for each of those numbers.
    - Rerun the previous step while leaving the numbers in ascending order while adding.
    - In addition to the run time, provide the tree height for both of these trees.
    - Use System.nanoTime() to determine how long the lookups take.
    - Provide an explanation of why these results turn out as they do.

Red-Black Tree

A Red-Black Tree is an example of a self-balancing BST. By self-balancing it ensures that the tree can fulfill its O(log(n)) search time. This method uses a flag in each node to denote if the node is 'red' or 'black' and observes a set of rules about the coloration of the nodes. The tree holds as a rule, or invariant, that all nodes contain the same number of black nodes in their path to the root of the tree. In this section you will implement a Red-Black Tree. Refer to your class text for details on how to implement the insert method of the tree.

3. ***Implement a RedBlackTree class satisfying the interface provided in IBinarySearchTree.java.***
    - Make sure your insert function can handle all possible color combinations!
    - Remember, in order for strings to properly match you must compare your hashed objects using the .equals() method rather than using the == operator.

- The getTreeString() method for this version requires you to include the color of each node.
- Ensure you can pass all unit tests.
- The unit test do not test all combinations. Feel free to add additional tests until you are sure your solution is robust.

4. ***Profile your RedBlackTree***
    - Profile by adding the numbers 1 to 10,000 in a random order and then timing the search for each of those numbers.
    - Rerun the previous step while leaving the numbers in ascending order while adding.
    - In addition to the run time, provide the tree height for both of these trees.
    - Use System.nanoTime() to determine how long the lookups take.
    - Provide an explanation of why these results turn out as they do, and compare them to the binary search tree profiling results.

Write-up and Submission

- For each part of the project, describe your approach, solutions, describe any difficulties that you encountered, and detail the efficiency of all non-test methods in your code using Big-O notation. Include any requested output and screenshots for each part.
- Add all of your *.java* files and your write-up document to a .zip or .tar.gz archive and submit it on Bb Learn. Make sure all of your java files are in the root of the zip file, rather than inside additional folders.
- You **must** write and submit your own code. You must also clearly identify and online or text sources that you used as references, any collaborators with which you discussed the project (or lack thereof), and how the source was beneficial.

Grading

This project is worth 100 points.

- 10 Points – Design and code quality
    - Good object-oriented design, consistent comments, white space, indentation, etc.
- 60 Points – Implementation quality
    - Demonstrate that your solution is complete and accurate based on unit tests.
    - 10 points – BST searching
    - 10 points –BST (other features)
    - 10 points – BST profiling
    - 10 points – RedBlackTree insertion
    - 10 points – RedBlackTree (other features)
    - 10 points – RedBlackTree profiling
- 30 Points – Writeup and Submission