Austin Kelly

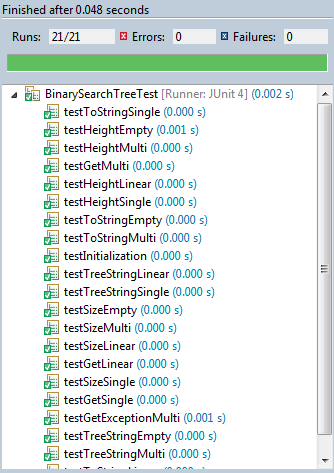
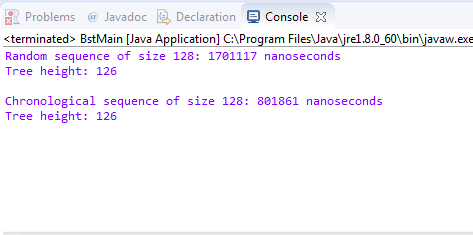
CS249

11/21/16

**Project 4**

**Binary Search Tree**

The first section of our project had us create a binary tree from the ground up based on an interface given. A binary tree is a tree structure in which every existing left child of every node is less than its parent, and every right child is greater. I approached this by creating a node class which takes in a generic key and element in the constructor. Within this, I had a print statement exclusive to which type of tree would be used. One with a color included for part 2, and one without. In the BST class, I created a put method that created a node, and if there was no root, assigned it to be one, while incrementing the overall size. If the given key is less than the current key, and there is space available, it assigns the created node to the left. I did the same thing with the right side for if it was greater. The delete method was a good amount trickier and required a few external things. I created an isLeft Boolean, parent node, and a getReplacement method to complete this. Within the delete, I decremented size, and iterated through a while loop as long as the key isn’t the key we are looking for. I updated the current variable in the same way I did in put, this time using compareTo instead of operands. I then used a series of if else statements to determine which spots are to be reassigned, and called my getReplacement to adjust the tree. I had to use helper functions that took in node parameters for my getHeight and getTreeString methods. The original methods just called them while accounting for empty trees. I had a lot of trouble constructing my string building methods, but was ultimately able to move through them recursively using an in order traversal, while adding to the resulting string with necessary commas and brackets. This is all explained in my comments of my code. I also ran into a problem that I couldn’t resolve in my main method, which would not allow me to search through a tree larger than a bytes worth of data, so I implemented my tests using 128 instead of 10,000 to show what I could on a smaller scale. Overall, I was able to pass every single unit test and have flawlessly running code besides the one issue in the main method.



Big O review:

getReplacement = O(n)

moveInOrder() = O(n Log (n))

buildString() = O(n Log(n))

put() = O(n^2)

get() = O(2n)

getSize() = O(1)

toString() = O(1)

findHeight() = O(N)

getHeight() = O(1)

getHeight() = O(n Log(n))

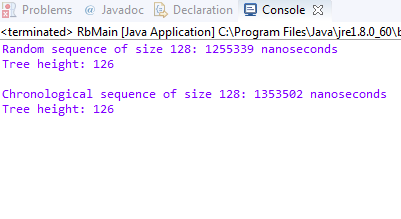
startTime() = O(1)

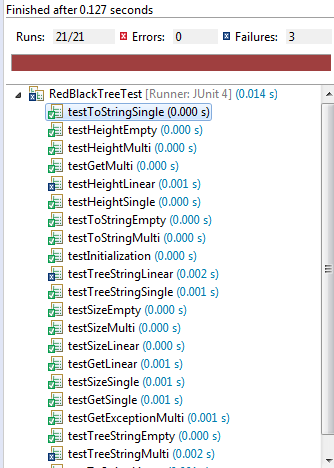
endTime() = O(1)

print() = O(1)

**Red Black Search Tree**

The second section tasked us with creating a Red Black tree. A red black tree is a type of binary search tree that is self balancing, using color assignments of red and black to constrain how unbalanced a regular tree could get regularly. I approached this by basing my code off of my original binary search tree, since a red black tree is still a binary search tree. I then changed the put and delete methods and used the color value I assigned in the node class to give each node its red or black property. In my in order traversal, I set the root to black, since that is a constant property of red black trees. In my put method, I defaulted to red. I tried a lot of different methods to get the colors to balance throughout the tree, including a helper method I called balance. In the end I wasn’t able to get a fully working upgrade from a binary search tree to a red black, but I did manage to pass all but 3 of the tests in the end. This portion of the project was very difficult for me, but I was able to get a solid amount of it working I think. This classes main method is near identical to the last one. With that, it suffers from the same problem that I couldn’t figure out, and only tests up to 128 instead of 10,000. For the random array, I added the desired number of elements to an arraylist and shuffled. I then called get in the resulting random sequence.





Big O review:

getReplacement = O(n)

moveInOrder() = O(n Log (n))

buildString() = O(n Log(n))

put() = O(n^2)

get() = O(2n)

getSize() = O(1)

toString() = O(1)

findHeight() = O(N)

getHeight() = O(1)

getHeight() = O(n Log(n))

startTime() = O(1)

endTime() = O(1)

print() = O(1)