11/19/2016

Project 4: Binary Search Trees

CS249

Instructor: Patrick Kelley

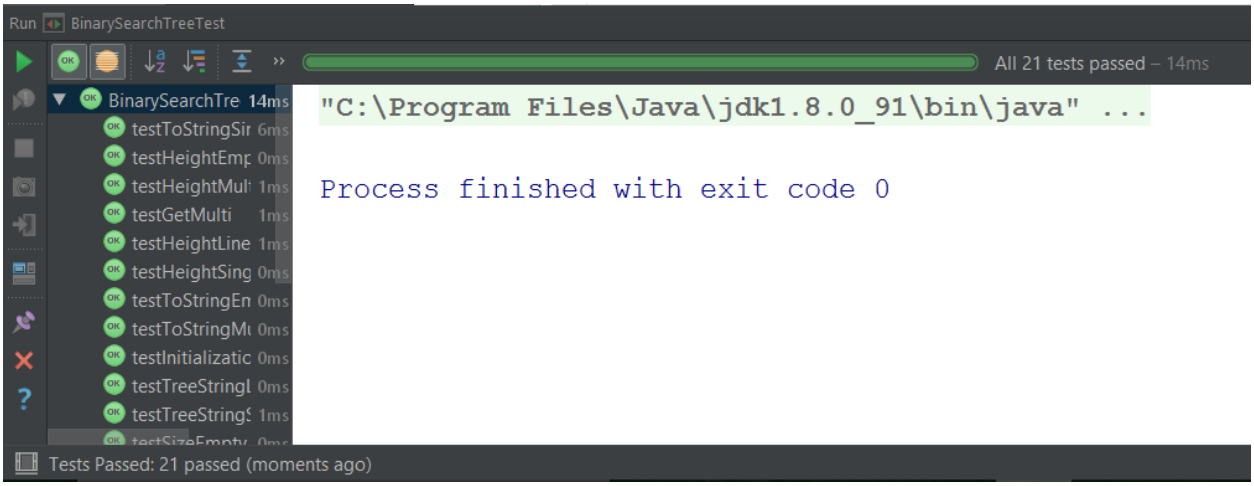
Stephen White

Overview: The overall purpose of this project was to write two separate programs that could model a simple binary search tree and a more complicated red black tree respectively, and to test the efficiency of each. This testing would require the use of java's System.nanoTime() procedure, and would prove that the efficiency of a red black tree is much better than a simple binary search tree.

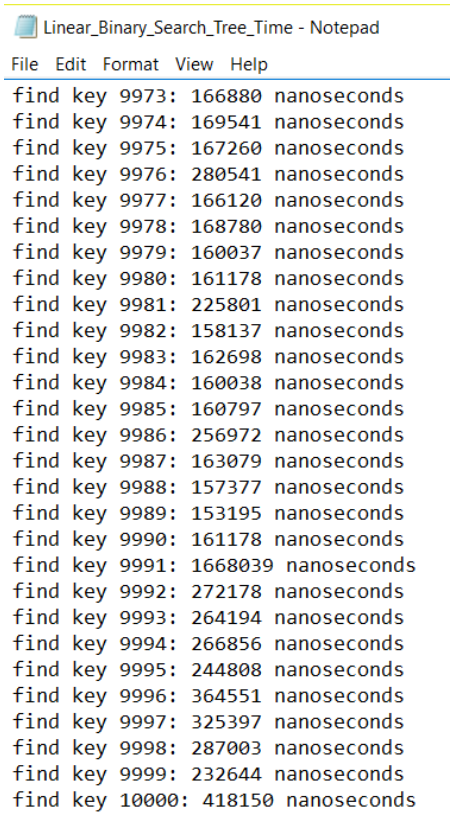
Process:

Binary Search Tree: For this project, I decided to do something that I have never really done for my projects… utilize version control. My thought process was that as I completed the project in chunks, I would keep track of how far I have come, and if at any point I wanted to attempt something drastic, I would be able to backtrack to a save point that was documented prior to my drastic changes. With that in mind I began my journey by reading over chapter 8 of the textbook and understanding exactly how binary search trees worked. From there coding it was a relatively simple process, and only a couple of hurdles kept me from completing that portion of the project very quickly. These hurdles came in the form of the recursive methods I utilized for getTreeString() and toString(), in which I needed to format a string in a very weird way. The complexity of put and get was O(Log(N)) as I had to traverse the tree using a while loop and comparisons to cut down the number of elements I had to look at by half each time. The getHeight() method's complexity was $O(2^n)$ because with every recursion, I had to check another multiple of two items. To test the efficiency of my simple binary search tree, I wrote a program (BST_Profiler.java) that would create two separate binary search trees (one linear tree from 1-10,000 & one random tree from 1-10,000) and time how long it took to get() each element in the tree. I was able to see that in a random tree, the search times were much smaller because there were less nodes to traverse while in a linear binary search tree, the times were incredibly large

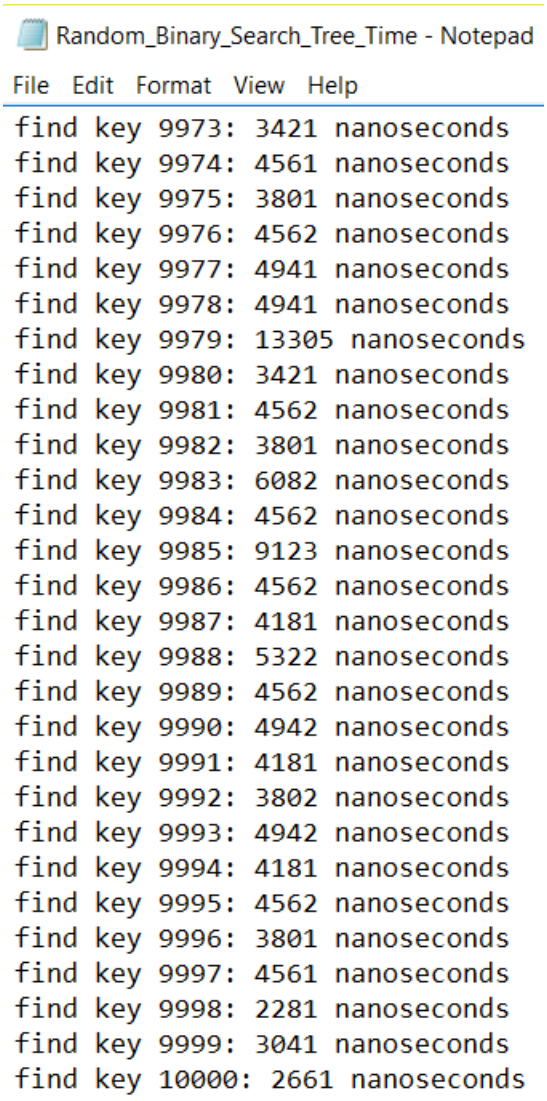
because to get to the 10,000th item, this would be the same thing as traveling 10,000 links in a singly linked list to find the final element. Show below are pictures of all unit tests passing, as well as some sample output from my profiler class.



(Above) All unit tests passing for a simple Binary Search Tree



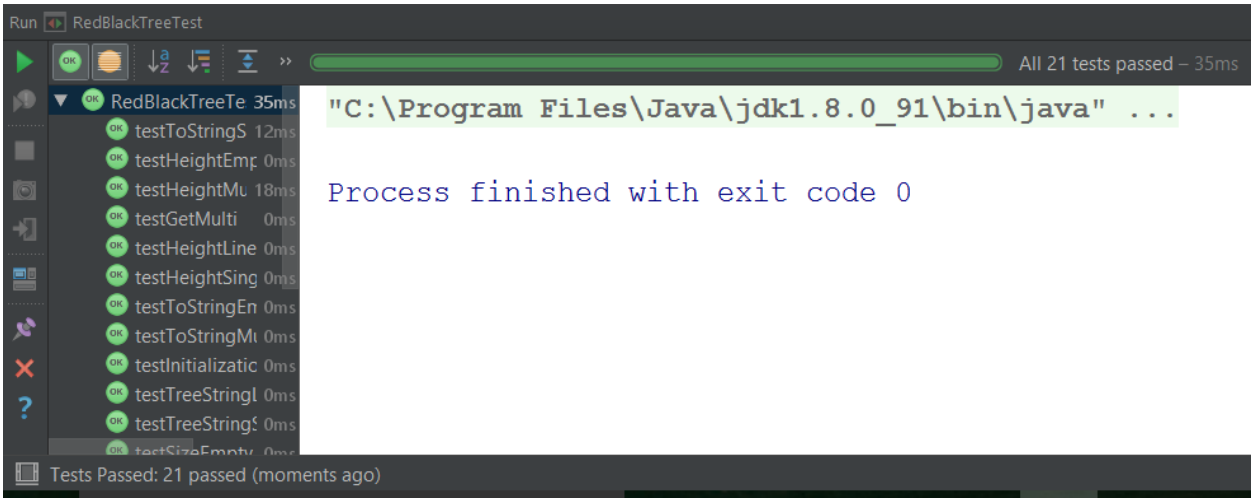
(Left) Times taken to find 10,000 elements in a linear binary search tree



(Left) Times taken to find 10,000 elements in a random binary search tree

Red Black Tree: This portion of the project was quite possibly the most complex thing I have ever programmed. If you would have asked me 9 days ago, my response would have been either the N-Queens problem, or probably Knapsack, but this is the problem to end all problems. Not only was this program conceptually difficult to understand to a large degree, but the coding process for it was incredibly painful. So painful, in fact, that it required me to spend eight straight hours working on it only for it to fail. For this portion of the project, I can attribute all of my success to my friend Connor Schwirian, whose dedication and willingness to assist not only

me, but countless others, deserves much more than a simple thank you. We went over all of the cases on a white board, discussed them for eight hours, and coded them up. I made sure to extensively comment my code, as to understand what exactly it was that I was doing in the future, and even still, I am confused by it. It was as if a was hit by a ton of bricks the moment I ran those unit tests only to see 17 null pointer exceptions appear, but through utilizing the debugger, I was able to correct these issues, and shrink my problem down to a mere 4 unit tests. After spending a second day on this project, Connor and I were eventually able to detect what the error was and correct it (oddly enough it was within my node class that I was stupid enough not to set a parent reference when I created a left and child reference to the current node). Excited, and having all of my unit tests passing, I went to work on transforming my profiler from above to fit the red black tree I had just created, only to find that I was receiving a NoSuchElementException that made me want to cry. For some odd reason, the program was deleting references to nodes, and I understand that the balancing of the nodes must be wrong in some way, but for the life of me, I could not tell you where I went wrong. I had worked very hard up to this point, and was at a loss for what to do, so I utilized a try, catch block to handle this exception and continue outputting my information to a text file. The complexity of my put method was still O(log(N) as well as get, and getHeight() was $O(2^n)$ again, for the same reasons mentioned above. As far as analyzing the times required to find each element in a self-balancing tree, I can state that they should be better than that of a simple binary search tree. My data definitely displays this as you should be able to see below. In regards to a linear red black tree versus a random red black tree, the times should be nearly identical, due to the fact that each tree should be balanced. Shown below are the times of each so that you may see for yourself.



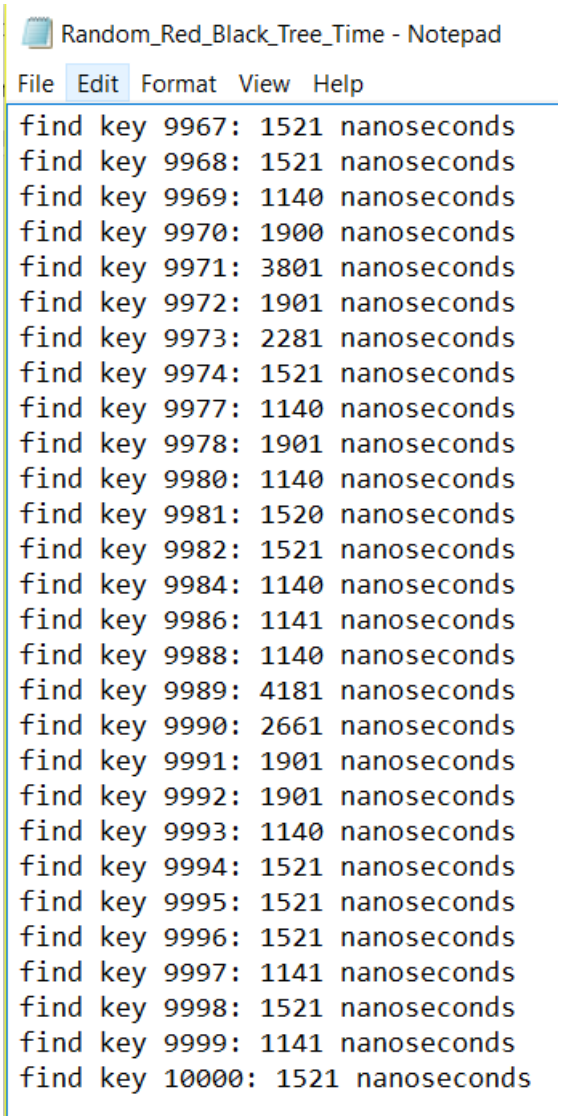
(Above) All unit tests passing for the Red Black Tree

Linear_Red_Black_Tree_Time - Notepad

File Edit Format View Help

```
find key 9973: 1140 nanoseconds
find key 9974: 760 nanoseconds
find key 9975: 761 nanoseconds
find key 9976: 760 nanoseconds
find key 9977: 1140 nanoseconds
find key 9978: 1141 nanoseconds
find key 9979: 760 nanoseconds
find key 9980: 760 nanoseconds
find key 9981: 1141 nanoseconds
find key 9982: 1141 nanoseconds
find key 9983: 760 nanoseconds
find key 9984: 1141 nanoseconds
find key 9985: 1141 nanoseconds
find key 9986: 1140 nanoseconds
find key 9987: 1140 nanoseconds
find key 9988: 760 nanoseconds
find key 9989: 1141 nanoseconds
find key 9990: 760 nanoseconds
find key 9991: 1140 nanoseconds
find key 9992: 761 nanoseconds
find key 9993: 1901 nanoseconds
find key 9994: 1140 nanoseconds
find key 9995: 760 nanoseconds
find key 9996: 761 nanoseconds
find key 9997: 1141 nanoseconds
find key 9998: 760 nanoseconds
find key 9999: 1141 nanoseconds
find key 10000: 1140 nanoseconds
```

(Left) Times taken to find 10,000 elements in a linear Red Black Tree



(Left) Times taken to find 10,000 elements in a random Red Black Tree

Conclusion: As far as projects go, this was quite possibly the hardest thing I had ever done. It tested my bravado, wit, and strength as a programmer. I was forced to work on this project for 8 straight hours until 4am and still did not receive the results that I thought I would. The most heart-wrenching thing about this, is that after so much work, I still fear that it will not pass all of your unit tests. All I can say is that I gave it my best shot, and all I can do is accept the work that I have, and know that this is my personal best. Nothing I have written is done poorly, and I am proud that I dealt with this project in such a professional way. If there is anything that I have

learned from this project is that you cannot win them all, but you can learn from your mistakes and keep marching forward, as I intend to do. As the semester comes to a close, I can sincerely reflect on the past several months and say that I am much stronger now for having endured, than I was when I first entered this class. I look forward to one more project, and intend to give it everything I have, as much effort as I have given the entire semester, in the hopes that I will finally pass this class. I may not like red black trees, but I have definitely seen their use, and can see how they work in a real world context. Here's to one more project!