Craig Lombardo

CS 150 Lab 9

Professor Liew

4/19/15

1. Introduction

The purpose, or goal, of this lab was to explore how to create a 2-3 tree in java, to implement a generic 2-3 tree class and finally to compare these findings with the TreeMap in the java API. We also wanted to test the performance of insertion and find operations on these trees. For this lab the main problem I ran into included figuring out how to make a 2-3 tree that will recursively restructure itself. I assumed it would be easier to create this data structure with only one inner node class, as directed by you. The main goal of this lab is to juxtapose TreeMaps with 2-3 trees to see which is more efficient in its data storage and retrieval.

2. Approach:

For the purposes of this lab I have designed a few classes to conduct my different tree testing. There is a class called AbstractTree, which will serve as a list of necessary methods for different trees should they choose to extend the AbstractTree class. Every tree will extend this class for my purposes, which means each tree will have the ability to add to the tree, clear the tree, check if a key is contained in the tree, find the first element, the last element, the size of the tree and a method which will return all of it’s elements in ascending order, based on their keys. There are also classes called TwoThreeTree and TreeSetContainer which will store our data and will provide us with the proper testing structure. There is an ExperimentController class, which will actually test the TreeSetContainer and will provide me with the feedback needed to compare things. Unfortunately my code for the TwoThreeTree class fell flat a little and was not completely functional. Things worked for the primary cases; however, the recursive restructure did not work and thus made testing impossible. There is also a Wheel class that will generate random numbers, which will be added to the TreeSetContainer.

3. Methods

The setup of this program was broken down into different parts. A key aspect was the creation of methods that ran the experimental work for me. Each test was run in two stages, the first included a small test to ensure things worked as expected on a small scale, and then the second part was a more extensive test. The first test gave me a general sense of how long the program would take to run given the time taken for n elements. This small test also helped me get a general sense on what to expect for my more extensive tests. I decided to implement this functionality as so as I have done so in previous labs and it has worked well. I also had to employ this method as I could not really test my two three tree as it was not complete. I could however test the Java API TreeMap and check the correlation between the size for insertion and find. It is important to note that I did not put in exceptions that will determine whether or not a valid string was passed in for the first argument, nor a check for non-integer values passed in for the second/third argument, nor a Boolean check for the last. If anything else is input other than this, then desired results may not occur. For the actual testing portion of the lab I decided I would start at 1000000 elements and increment by 1000000 as I went along. Both test types, insert and find were done with these data sets which incremented a total of 20 times. Each test had a different seed value and was tested 3 times. In each test, each data point collected resulted from an average of 5 different tests at that sample size to ensure I get a better handle on the nature of the data structure.

4. Data and Analysis

This graph represents the comparison between the average time taken based on inserting n elements into the TreeMap. This graph shows the linear correlation between adding the n elements which means the complexity for adding a new element into a TreeMap is O(1) as each data point is the time required to add n data points so while the graph seems to signify O(n), since it is on n elements we get a O(1).

This graph represents the comparison between the average time taken based on finding n elements in the TreeMap. This graph shows the (mostly) linear correlation between finding an element in the TreeMap after adding n elements. It is tough to see the relationship between the other points as there are 2 outliers in the beginning of the testing; however, it is roughly linear with an increasing trend. It is tough to see as the scale gets messed up; however, every test had these “anomaly” type points so I decided to keep them. The complexity for finding an element in a TreeMap is O(n), which is off, as each data point is the time required to find an element based on n.

5. Conclusion

Through my experimental data, I have determined that TreeMaps are very fast when it comes to inserting and finding data. As far as two three trees, the most I can say about them considering I don’t have testing data to back up this reasoning is that they are difficult to construct. The base cases work; however, the more complex restructuring algorithm unfortunately did not work.

6. References

"3.3   Balanced Search Trees." *Balanced Search Trees*. N.p., n.d. Web. 19 Apr. 2015. <http://algs4.cs.princeton.edu/33balanced/>.

CS 150: Guidelines For Writing Lab and Project Reports: https://moodle.lafayette.edu/pluginfile.php/151579/mod\_resource/content/1/writeup-guidelines.pdf

CS150: Lab 9 -- 2-3 Tree Construction

http://www.cs.lafayette.edu/~liew/courses/cs150/lab/labs/lab09e/

Robert Sedgewick "TwoThreeTree.java - Twothreetree - An Implementation of 2-3 Tree Data Structure - Google Project Hosting." *TwoThreeTree.java - Twothreetree - An Implementation of 2-3 Tree Data Structure - Google Project Hosting*. N.p., n.d. Web. 19 Apr. 2015. <https://code.google.com/p/twothreetree/source/browse/trunk/src/sergey/melderis/twothreetree/TwoThreeTree.java?r=2>.

Weiss, Mark Allen. *Data Structures & Problem Solving Using Java*. Boston: Pearson/Addison Wesley, 2010. Print. Fourth Edition.