1 Assignment 3

Submit source code and running instructions and any textual responses to EAS[1] as "Programming Assignment 3". Do it in Java. Do not use java's search methods! Do not use Java's Collections or Subclasses, code your own if you need them. Place all textual responses in a **PDF** report submitted with the code.

This assignment has a significant design component. You will use UML class diagrams and text to explain your decisions inside the diagrams, as well as any additional decisions. Do the design first! Read the **whole** assignment before you start. Incorporate the diagrams into your report, supporting them with text as described in the assignment.

Don't use packages! Using packages is generally good, but is a pain when I want to test all of you easily and you make other minor changes. **Don't deviate from the file names indicated: TwoThree.java** That includes renaming the file, but leaving the class inside it the same. You may include other files, and those files may be in packages.

Posted: Sunday, July 29^{th}

Due: Sunday, August 12^{th}

Grade: 30%

1. Designing a flexible tree structure
   (a) Design a tree structure that you will implement and subclass to solve both the Two-Three Tree Part of the Assignment Problem and a theoretical Red-Black Tree problem.
   (b) Include a class diagram showing all the supporting classes of this tree structure.
   (c) Explain your decisions.
   (d) Include a class diagram showing how your Two-Three Tree implementation will extend your general tree structure
   (e) Explain your decisions textually
   (f) Include a class diagram showing how your Red-Black Tree implementation would extend either your general tree structure, or the structure of the Two-Three Tree.
   (g) Explain your decisions textually
   (h) Explain the advantages of using a Two-Three Tree over a Red-Black Tree, in general, referencing your diagrams and decisions as is appropriate.
2. Implement an Two-Three Tree based off of the design in question 1.
   (a) Using the command-line, your program should be called as
   java TwoThree <source> <step-to-traverse>
   (b) **<source>** will be the name of the local file to be loaded that contains operations
   (c) [**step-to-traverse**] will be a number, and after that many operations you will output a pre-order traversal of your tree. e.g. '1' would mean showing the traversal after a single operation.
   (d) Your program should output the following statistics after all operations are processed:
      - The total number of comparisons between nodes (not checks for null).
      - The total number of times a ThreeNode is created
      - The total number of add operations
      - The total number of find operations (when the tree is not empty)
      - the total number of remove operations (when the tree is not empty)
   (e) The operations in the source file will be one-per-line and will consist of the letters a,r or f, representing add, remove or find respectively, followed by digits, the value for the node.

[1] `https://fis.encs.concordia.ca/eas/`

(f) The sample source file that your should run your code against is
`https://users.encs.concordia.ca/~sthiel/coen352/Operations.txt`

(g) **note:** The tree starts out empty and is filled by the Operations.txt (or similar file). Your program should deal properly with trying to find or delete nods that aren't in the tree.

(h) **note:** Comparisons in a ThreeNode should go left-to-right sequentially.

(i) **note:** Comparisons should check for less than, greater than, and then equal for finding. For adding it should check for less than, then greater than or equal to.

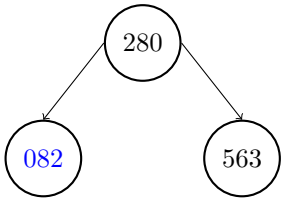
2-3 Search Tree Example

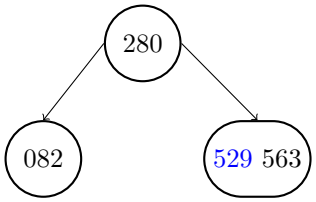
Source.txt:
1) a280
2) a563
3) a082
4) a529
5) a500
6) a900
7) a794
8) f563

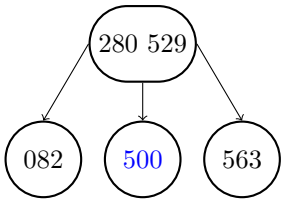
java TwoThree Source.txt 6

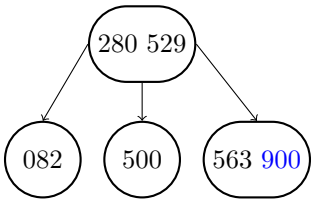
1)

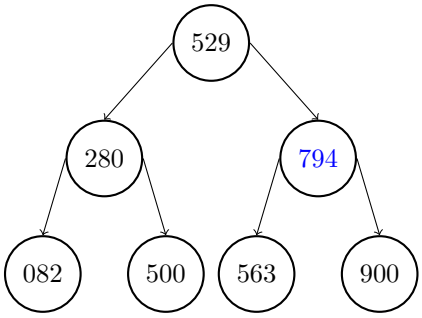

2)

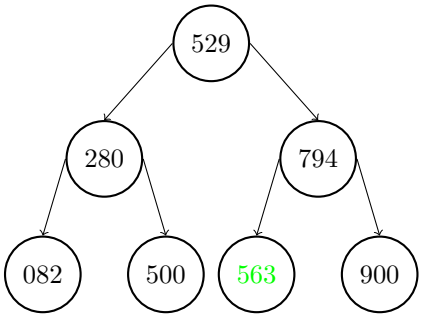

3)


4)


5)


6)


7)


8)


16 Comparisons Made
3 Three Nodes Created
7 add operations
1 find operations
0 remove operations
Pre-order traversal after step 6: 280 529 082 500 563 900