1 Assignment 3

Submit source code and running instructions to EAS[1]. Do it in Java. Do not use java's search methods! Do not use Java's Collections or Subclasses, code your own. 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!** Huffman.java and ATree.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: Monday, July 24th

Due: Friday, August 11th

Grade: 30%

1. Designing a flexible tree structure
   - (a) Design a tree structure that you will implement and subclass to solve both the Huffman Coding Problem and the AVL 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 you Huffman Coding implementation will extend your general tree structure
   - (e) Explain your decisions textually
   - (f) Include a class diagram showing how your AVL Tree implementation will extend your general tree structure.
   - (g) Explain your decisions textually
   - (h) Include a class diagram showing how a Splay Tree implementation would extend your general tree structure, but don't implement the Splay Tree... just consider it.
   - (i) Explain your decisions textually
   - (j) Explain the advantages of using an AVL Tree aver a Splay Tree to solve question 3, referencing your diagrams and decisions as is appropriate.
2. Implement a Huffman Coding Tree based off of the design in question 1.
   - (a) Your tree should build itself using `https://users.encs.concordia.ca/~sthiel/coen352/Jabberwock.txt`
   - (b) Include the characters for spacing as well as all punctuation, but not line breaks.
   - (c) You should not make encoding for characters not found in the source text, and I will not use any such characters while testing your code.
   - (d) Weight should be based on the frequency of occurrence and the order in which a node is encountered. That is, if 'e' and ' ' have the same frequency, but 'e' occurs first, its weight would be considered lesser.
   - (e) When merging nodes, the new weight should be the sum of the previous weights but it should be considered as newly encountered, thus it would "weigh" less than something with the same traditional weight that is already on the priority queue.

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

(f) When merging nodes, lesser nodes (by weight, and then if there is a tie, by occurrence) should be on the left, the other node on the right.

(g) Using the command line, your program should be called as
java Huffman <source>

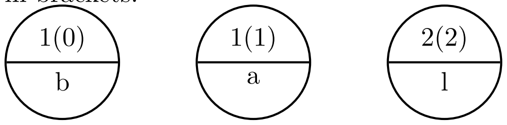
(h) **<source>** will be the name of the local file to be loaded to built your Huffman Tree

(i) Upon starting, a frequency table will be built. Your program should listen to stdin (you may use Scanner) and encode any line entered (terminated by a line break) and output the encoded version using 1s and 0s (vs. padded ASCII). No spaces should be shown between the encoding.

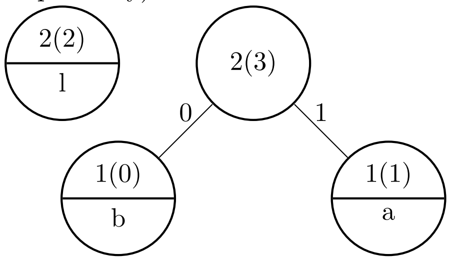
Huffman Example

Given the source text "ball", we will show the building of the tree:

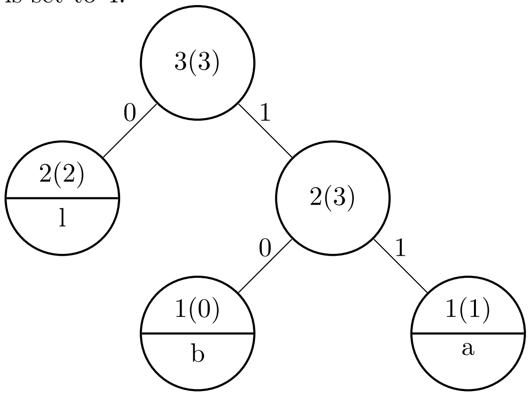
step 1: All values with their initially calculated frequencies, with their order of occurrence given in brackets.



step 2: Nodes b and a were merged as they are the two nodes with the least weight, with node b on the left because it occurred first. The new root node has weight 2, but since it is a new node, it's occurrence counter is set to 3 (since the first three values, b, a and l took up 0, 1 and 2 respectively).



step 3: Nodes l and ba were merged, with node l on the left because it occurred before the newly merged node. The new root node has weight 3, and since it is a new node, it's occurrence counter is set to 4.



3. Implement an AVL Tree based off of the design in question 1.

(a) Using the command-line, your program should be called as
java ATree <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 an optional number, and after that many operations you will output a post-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 node's parent changes (excluding new nodes getting their first parent, and excluding removed nodes "losing their parent").
- The total number of find operations (when the tree is not empty)
  - Where the item is found
  - Where the item is not found
- The total number of add operations
- the total number of remove operations (when the tree is not empty)
  - Where the item is found
  - Where the item is not found

(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.

(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) a traditional AVL tree balances itself when it has a difference of 2 in its subtrees. This implementation should only balance when it has a difference of 3. When an unbalanced node is in-line with the newly added node (all left or all right), then a single rotation is performed, otherwise a double rotation is performed (the first rotation at the first branch to the newly added node below the unbalanced node).

AVL Example

Given Operations1.txt:

```
a5
a2
a1
a4
a7
a8
a9
a10
```

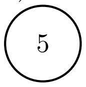
And command-line:

```
java ATree Operations1.txt 8
```

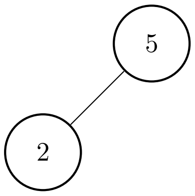
Output should be something like:

```
Traversal at 8: 1,4,2,7,10,9,8,5
15 compares
2 parents changed
0 successful finds, 0 failed finds
0 successful removes, 0 failed removes
8 values added
```

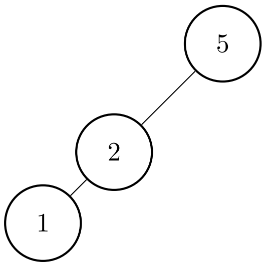
1)



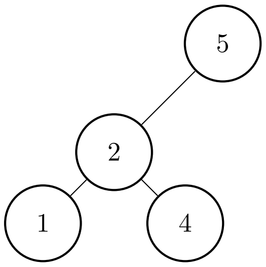
2)



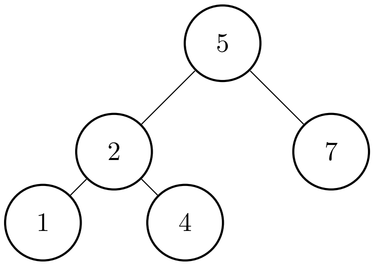
3)



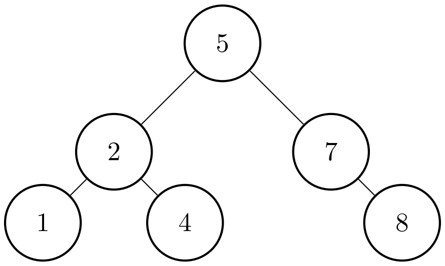
4)



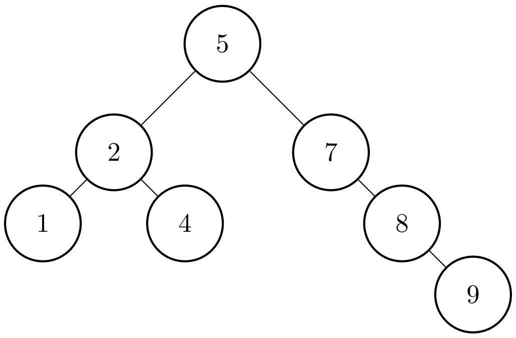
5)



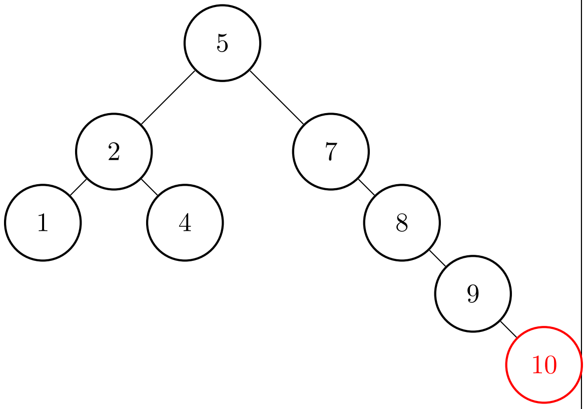
6)



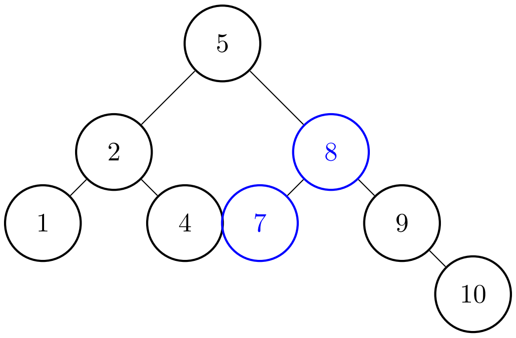
6)



7)unbalanced!



8) blue nodes had parents changed



4. Textual Responses, Keep in a separate file for easy reading by me! All responses should not exceed one page (10pt font, 1inch margin, for you weirdos)

   (a) Huffman

- Look up the file: `https://users.encs.concordia.ca/~sthiel/coen352/RandomStrings.txt`
- Encode the string associated with your student id and record the result here.
- Describe what percentage fewer bits were used than the fixed-length ASCII encoding would have taken.
- Indicate whether you feel that the encoded string matched the source text frequencies, and provide your reasoning.

(b) Advanced Trees

- Provide the output you recorded based on the Operations file provided in question 3
- Explain whether you feel that your that the AVL Tree was the correct choice, given that you have now implemented and tested it against the provided Representative data. Explain anything you have learned through implementing and testing that would affect any future designs.