CS 345 (Fall 18): Computer Organization

Project #4

AVL and Red-Black Trees

due at 5pm, Wed 31 Oct 2018

1 Introduction

In this project, you'll be implementing two new BSTs: an AVL tree, and a Red-Black tree. In the AVL tree, you will implement all of the same operations as in Project 3 - except that I won't have a way to **force** you to do a rotate. Instead, you'll do it, on your own, when rebalancing is required. In the Red-Black tree, you will implement a slightly smaller set of operations: we won't attempt deletions, and we also will never attempt to call `set()` when the key already exists.

A lot of the basic mechanisms will be familiar from Project 3. We are still using Java Generics - although this time, we'll only support two different trees (si=String/Integer and is=Integer/String). As before, we'll have example .class files (two of them, this time) and you will write student .java files to match their results. You will still write most of the testcases, and you will be generating `.dot` files to help you debug your work.

What's new in this project is, first of all, that you will be implementing two different tree types, each of which includes self-balancing features; to support this, the new `TestDriver` has a **mandatory** first argument, which is either `AVL` or `RedBlack`. Second, your code for the Red-Black tree must include a second constructor, which takes a `234Node` as a parameter. This constructor will create a pre-generated tree, without having to do `set()` operations first.

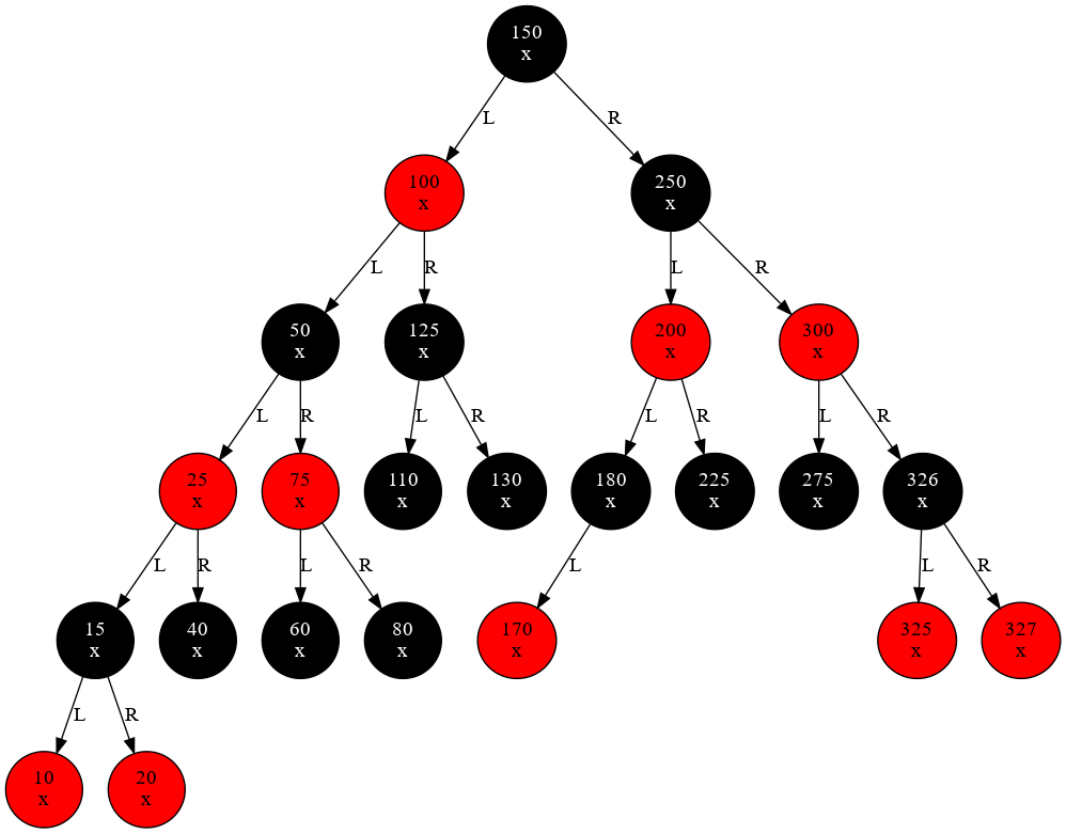
1.1 Test Code and Interface Updates

The `Proj04_TestDriver` class works more or less like `Proj03_TestDriver`, except that you **must** pass either `AVL` or `RedBlack` on the command line, to tell it what sort of tree to test. Moreover, the testcases no longer support the rotate commands, since the interface doesn't support it, either.

The new interface that your student code must implement, `Proj04_Dictionary`, is basically the same as `Proj03_BST`, except that the rotate methods have been removed. In addition, the `inOrder()` and `postOrder()` methods have had their third array updated; instead of using it to report counts, this is now an array of `String`, and you are to put the "auxiliary" information for the node there. For a Red-Black tree, the auxiliary information is the string `"b"` or `"r"`, indicating the color of the node; for an AVL tree, this is the height, reported like `"h=1"`. (Look at the example output from the traversal commands to see how this all gets printed out.)

Finally, we have added one new test class, `Proj04_TestBuildRBFrom234`. Like the `TestDriver`, this reads inputs from `stdin`; however, in this case, these

commands simply describe the nodes of a 2-3-4 tree. This code builds a 2-3-4 tree from the text file, and then uses the 2nd constructor of your Red-Black code to convert the 2-3-4 tree into a red black tree. (Experiment with the example code to see how my code handles nodes with 2 keys in them.) This code will print out both traversals (so that we can do automatic debug), and then also generate a .dot file. Here is the result of the .dot file generated from the testcase `test234_01`:



Finally, the testcase naming convention has gotten a little more specific. Any file named `test_*` will be interpreted by the grading script as a **general testcase** - that is, a series of commands. This will be tested against **both** your AVL code and your Red-Black code, and it works the same way as in Project 3.

Any file named `test234_*` simply contains a conversion test. Instead of having normal commands, it simply has a list of nodes. This, of course, is only tested against your Red-Black code.

2 The Files I Provide

I have provided you the following files:

- `Proj04_Dictionary.java`

  This is the java interface, which must be implemented by both of your classes.

- `Proj04_(AVL|RedBlack)_example.class, BSTNode_russ.class`

  These files are the example code I've provided. I have provided the two required data structures, plus a single Node class. (I have chosen to have a single Node class, that is shared by both data structures - when you implement your Node, you may do the same, or you may implement 2 of them.)

- `Proj04_TestDriver.java, Proj04_TestBuildRBFrom234.java`

  These are the two testcase classes.

- `Proj04_234Node.java`

  This is the node class that is used when we build the 2-3-4 tree in the 2nd Test class.

- `test_01, test234_01`

  An example testcase in each of the two forms.

3 `x=change(x)` is Optional

For this project, I've decided to make the `x=change(x)` style entirely optional. I still **recommend** it - and I used it heavily in my solution - but it is **not required** this time.

4 Tree Limitations and Algorithm Expectations

As we've discussed, there are some details that can vary, from implemenation to implementation. While there are lots of good options, we have to choose one of them, so that we can compare your output to mine. Therefore, we have some assumptions and limitations:

4.1 Common Requirements

Both trees must:

- Update the count efficiently (see the Project 3 spec). Likewise, the same requirement applies for updating **any** metadata - including, but not limited to, the height in the AVL tree.

- Generate meaningful and useful .dot files.

  At a **minimum**, the Red-Black .dot file must indicate the color of each node, either with a word or letter, or by actually coloring the node. Likewise, the AVL tree must include the height of each node - since this is a piece of information that's often critical for debugging.

4.2 AVL

Your AVL tree must:

- In delete case 3, pull up the predecessor (just like Project 3)
- In a rebalance, if two grandchildren tie for "worst" (which never happens when inserting nodes, but can happen when we delete), always assume that the "outer" grandchild is the one that needs to be fixed. In other words, always do the single rotation fix if you can get away with it.

4.3 Red-Black

Your Red-Black tree must:

- Use the top-down insertion method. So "split" a full widget (by recoloring its nodes) as soon as you see that it is full - whether or not you will eventually need the space.
- You are **not** required to support the `remove()` operation. The testcase won't ever call it. However, you still have to **define** the function (to make the interface happy) - I would simply create an empty method, and throw an exception inside it.
- You are **not** required to support any `set()` operation where the key already exists. Again, the TestDriver will filter the testcase, running only the operations which are legal for a red-black tree.
- When converting from a 2-3-4 tree to a Red-Black tree, use appropriate widgets. (Run the example code to see how to handle the 2-keys-in-a-node case.)

5 Base Code

Download all of the files from the project directory

`http://lecturer-russ.appspot.com/classes/cs345/fall18/projects/proj04/`

If you want to access any of the files from Lectura, you can also find a mirror of the class website (on any department computer) at:

`/home/russelll/cs345f18_website/`

6 A Note About Grading

Your code will be tested automatically. Therefore, your code must:

- Use exactly the filenames that we specify (remember that names are case sensitive).
- **Not** use any other files (unless allowed by the project spec) - since our grading script won't know to use them.
- Follow the spec precisely (don't change any names, or edit the files I give you, unless the spec says to do so).
- (In projects that require output) match the required output **exactly!** Any extra spaces, blank lines misspelled words, etc. will cause the testcase to fail.

To make it easy to check, I have provided the grading script. I **strongly recommend** that you download the grading script and all of the testcases, and use them to test your code from the beginning. You want to detect any problems early on!

6.1 Testcases

You can find a set of testcases for this project at
`http://lecturer-russ.appspot.com/classes/cs345/fall18/projects/proj04/`

See the descriptions above for the precise testcase format, and also for information about how to run the two test driver classes.

6.2 Other Testcases

For many projects, we will have "secret testcases," which are additional testcases that we do not publish until after the solutions have been posted. These may cover corner cases not covered by the basic testcase, or may simply provide additional testing. **You are encouraged to write testcaes of your own, in order to better test your code.** You are also encouraged to share your testcases on Piazza!

6.3 Automatic Testing

We have provided a testing script (in the same directory), named `grade_proj04`. Place this script, all of the testcase files, and your program files in the same directory. (I recommend that you do this on Lectura, or a similar department machine. It **might** also work on your Mac, but no promises!)

7 Turning in Your Solution

Turn in the following file(s):

```
Proj04_AVL_student.java
Proj04_RedBlack_student.java
```

In addition, you may turn in additional files; for instance, I expect that most students will probably turn in a class to represent the BST Node (or, if you prefer, two different such classes). However, this isn't strictly required; if you want to use private classes inside your two files, that is also acceptable.

You must turn in your code using D2L, using the Assignment folder for this project. Turn in only your program; do not turn in any testcases or other files.