COMPSCI 2XC3 — Lab 6

DUE: Tuesday, March 8th, 11:59pm

Submit the lab on Avenue. Note, Avenue must receive your lab by the due date. **Do not leave your submission to the last minute! Late submissions, even by seconds, will not be graded.** You have been warned.

This lab is worth (59/7)% of your final grade in the course. Read this document carefully and completely. **Whenever I ask you to discuss something, I implicitly mean to include that discussion in your lab report. Furthermore, when I ask you to run an experiment or evaluate the performance of a function/method include scatter plots where appropriate. Do not import external libraries, especially those containing data structures you are meant to implement!** Include all timing experiments in your `code.py` file.

Purpose

The goals of this lab are as follows:

1. Implement the insert functionality of Red-Black Trees (RBTs).
2. Evaluate the advantages and disadvantages of RBTs compared to BSTs.

Submission

You will submit your lab via Avenue. You will submit your lab as three separate files:

- code.py
- rbt.py
- report.pdf (or docx, etc.)

Only one member of your lab group will submit to Avenue – see the section below for more details on that. Your report .pdf will contain all information regarding your group members, i.e. name, students number, McMaster email, and enrolled lab section. This will be given on the title page of the report. For the remainder of this document, read carefully to gauge what other material is required in the report. Your report should be professional and free from egregious grammar, spelling, and formatting errors. Moreover, all graphs/figures in your report should be professional and clear. You may lose grades if this is not the case. Organize the report in a such a way to make things easy for the TA to grade. You are not doing yourself any favors by making your report difficult to mark!

Implementing Insert [70%]

In your 2C03 lectures you should have seen the insert functionality of Red-Black Trees (RBTs). I will also briefly go over the starter code in your lectures this week. I do not have time in lecture to review all of the insert functionality. However, for a more advanced data structure like RBTs even something like insert can get confusing – it is worth reviewing. There are many good resources regarding the functionality and pseudo-code implementation of Red-black insert. For example:

- Algorithms, 4th Edition by Robert Sedgewick and Kevin Wayne
- Introduction to Algorithms, 3rd Edition

both go over RBTs in detail. There are also many good online articles/tutorials regarding RBTS, such as:

`https://www.geeksforgeeks.org/c-program-red-black-tree-insertion/`

Or if you are like me and cannot get enough of YouTube the following two videos are also good. Careful, there is an error in the second one (see the comments).

- `https://www.youtube.com/watch?v=v6eDztNiJwo&t=924s&ab_channel=RobEdwards`
- `https://www.youtube.com/watch?v=5IBxA-bZZH8&t=242s&ab_channel=MichaelSambol`

In `lab6.py` you will find two partially completed classes: RBNode and RBTree. Together, once completed, they implement RBTs. It is your responsibility to complete three methods across these two classes: `rotate_left()`, `rotate_right()`, and `fix()`.

- Rotate left and right are traditional rotate methods for binary trees. Pay close attention to what is pointing to what. Note that the nodes also point back to their parents. Also pay attention to how `root` works in the RBTree class. Once a rotation is made this may need to be updated!
- Read the insert method already partially implemented. Right now it is quite similar to a traditional BST insert method. This is because none of the RBT properties are being enforce. As you know RBTs have 4-6 crucial properties (depending on what source your read). Here, the two most crucial to pay attention to are:
  - Red nodes cannot have red children.
  - All simple paths from the root to a leaf must contain the same number of black nodes.

  As it stands, these properties may be violated by an insert. You need to fix this. That is, you need to implement the fix() method to properly implement a RBT insert. Be careful, the devil is in the details here. I have given you a print method to help you test your tree's functionality. Submit your completed implementation in rbt.py.

Red-Black Tree Height [30%]

Consider the following scenario when you are testing your RBT implementation. You insert the numbers 1 to 10,000 in ascending order. You query the height and find the tree has height 24. You take your tree and again insert the numbers 1 to 10,000 in ascending order. You query the height, it is still 24. You once more insert the numbers 1 to 10,000 in ascending order and query the height. This time it has changed... but it now 16! Is there a problem with your insert method? Give a brief explanation as to what may be going on.

We know that RBTs are self balancing, and BSTs are not. So should we always use RBTs over BSTs? Run an experiment where you create RBTs and BSTs based of randomly generated lists of numbers of length 10,000. What is the average difference in the height between the two? Comment on whether you think there is a case where you would prefer a BST over an RBT. You do not need to empirically validate this, but what is your instinct on the performance of a BST insert to a RBT insert for trees of similar heights?

Consider near sorted lists based off a factor (like we have seen previously). For lists of length 10,000. Graph the height of a BST and RBT vs the near sorted factor of the list. What do you observe?