Block Two Project: Implementing Red-Black Trees

November 3, 2018
CS: DS&A
Proof School

Your mission is to code up insertion and deletion for red-black trees! You may work on this alone, or in groups of 2-3. Each group should hand in one file of code.

Part I: Cows

(In which we familiarize ourselves with RedBlackTrees, and write code that tests for red-blackiness)

1. Look at the file `red_black_tree_project.py`. Note that node objects now have a `color` attribute. Use only the strings `"B"` or `"R"` for `color`. Also note that empty children are defined by having `None` for their `key` attribute. (Finally, note that we removed the `value` attribute, which wasn't doing very much.)

2. Look at the `insert` routine; note the changes that have been made, such as creating empty children when necessary. Of course, `__adjust_insert` hasn't been written yet; we'll do that in Part II.

3. Now write the code for `is_red_black`, which should use the internal helper function `__is_balanced`. The former takes a `RedBlackTree` and returns `True` iff all of the red-black conditions are satisfied. (This includes the conditions that the root and empty nodes must be black.) The helper function `__is_balanced` checks that the black heights down all paths from the root are the same. (This function should probably be recursive, and you may find that it's surprisingly tricky!) You're free to create whatever other helper functions you want.

4. Test your `is_red_black` by using `insert` to create trees that are and are not red-black. You can go in and manually adjust their colors for testing. It's important to really get `is_red_black` correct before going on, so test thoroughly!

5. I've adjusted pretty_print to print colors, and dots for empty nodes. (Spaces that are actually blank are printed as `*` as before.) I've included the source code this time, and you can look at it if you want. This code is six years old, and seems rather ugly; I don't remember quite how it works, despite being its author. (My comments didn't stand the test of time.) However, it's still vacuously better than anything I've seen from my students! ◡̈)

6. *Warning:* Do not test for empty children with the expression `if not node.key`, because this will be true for a regular node with key `0`. Use `if node.key is None`.

Part II: Electric Emu

(In which we resolve red-red collisions by rotating our uncles)

1. Now write `__adjust_insert`, following the cases from class. One of the tricky things is that every case we looked at has a mirror-image case that has to be covered here too. My advice is to write the code first assuming that the new node is to the left of its parent. Then, using `getattr` and `setattr`, you can modify your code to handle the symmetric cases at the same time.

2. You might find it helpful to separately write a `rotate` function. Remember that there are left and right rotations. (Again, I suggest coding up one of them first, and then figuring out how to have your code handle both cases at the same time. Of course, you could just write separate functions `rotate_left` and `rotate_right`.)

3. Now test your code by inserting lots of things, and checking that `is_red_black` is always `True`.

Part III: Dactylosporangium

(In which we delete our worries, adjust our heights, and live as beautifully balanced trees)

1. This time I'm asking you to write both the `delete` and the `__adjust_delete` routines. The former is very similar to the routine you wrote for binary search trees, but there's some thought to be given to handling empty children.

2. To make life a little easier, I offer the following simplification: *When you need to exchange the node-to-be-deleted with its successor (in the case where the former has two regular children), it's fine to just switch their keys.* This will make `delete` a lot simpler. (Of course, this isn't good practice. In theory, when you call `delete(node)`, that node should be deleted, and the other nodes left alone. With our simplification, it's a different node object that is going to be deleted. But it does make things easier for us to code.)

3. Finally, write `__adjust_delete`, following the cases from class.

4. And now, test, test, test! You should first try to pass the test code I provided, and then make your own!

Good Luck!!