Project Report

Introduction

The project is done with python (version 3.8.3). I used the BinaryTree library from https://awesomeopensource.com/project/joowani/binarytree. The library can be installed via command line:

```
pip install binarytree
```

Specifically, I used the print functionality from this library so that I can visualize the BST. All algorithms are designed either by myself or followed the textbook/paper. The zip file contains three python file:

1) bst.py contains methods for binary search trees such as rotation, insert, and discard.
2) project.py contains A1, A2, A3, and all helper functions for these methods.
3) test.py contains all testing methods

The following tests are available:

1) T0: run the intended test (more on this in the next section)
2) T1: run A1 with random sized tree in range (3, 30), and visualize
3) T2: run A2 with random sized tree in range (3, 30), and visualize
4) T3: run A3 with random sized tree in range (3, 30), and visualize

These tests can be run with the following commands:

```
python test.py t0
python test.py t1
python test.py t2
python test.py t3
```

Testing

The T0 output is structured as:

1) Theorem 1 -> size 1000, 1100, 1200
2) Theorem 2 -> same as above
3) Theorem 3 -> same as above

For each size, we do five individual test. For each test, the testing method first generates an almost balanced tree T with random integer values. Then the source tree S is generated from T with the instruction: *take that initial T and mark each edge randomly as "do not rotate" with probability 0.01 then rotate each remaining edge with probability 0.5.*

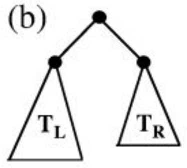
Next, the expected number of rotations is calculated, and the corresponding method (A1, A2, A3) are called. We record the actual number of rotation and print result for each theorem:

1) Theorem 1 and 2: pass if S is successfully transformed into T.
2) Theorem 3: pass if S is successfully transformed into T and the actual rotation is less than the expected rotation.

The rest of the testing methods, T1, T2, and T3 are intended to help the grader to visualize the tree S, T, and result tree so that they can see the algorithms output the correct resulting tree.

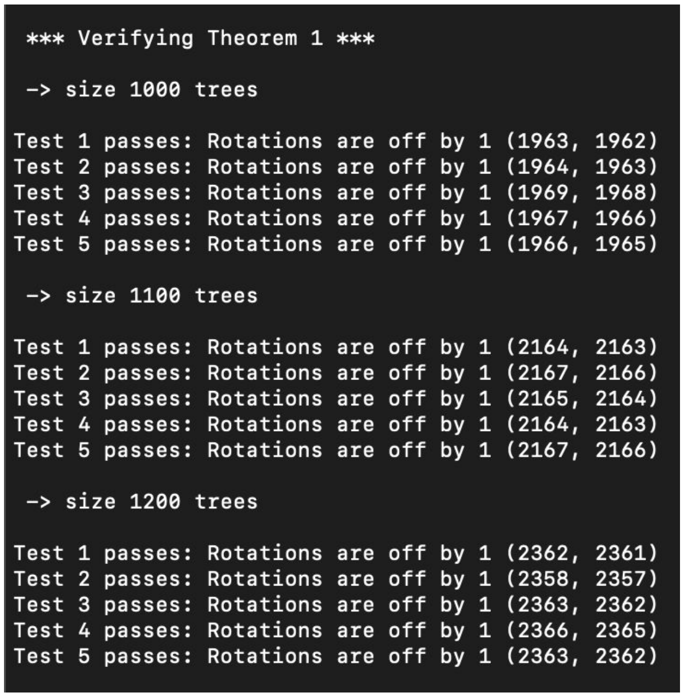
A1

My implementation of A1 strictly followed the paper. Except at step 6, before transforming the right forearm back, I check if the tree is case B - if so h is reduced by 1,



The reason behind this is because the height of the right subtree TR is indeed h-1, which is required to get the correct number of passes on the right subtree. This special case is not captured by the provided algorithm (possibly due to python indexing starting from 0 instead 1).

Here is the result of A1, as can be seen, the number of rotation is constantly off by 1:


```
 *** Verifying Theorem 1 ***

 -> size 1000 trees

Test 1 passes: Rotations are off by 1 (1963, 1962)
Test 2 passes: Rotations are off by 1 (1964, 1963)
Test 3 passes: Rotations are off by 1 (1969, 1968)
Test 4 passes: Rotations are off by 1 (1967, 1966)
Test 5 passes: Rotations are off by 1 (1966, 1965)

 -> size 1100 trees

Test 1 passes: Rotations are off by 1 (2164, 2163)
Test 2 passes: Rotations are off by 1 (2167, 2166)
Test 3 passes: Rotations are off by 1 (2165, 2164)
Test 4 passes: Rotations are off by 1 (2164, 2163)
Test 5 passes: Rotations are off by 1 (2167, 2166)

 -> size 1200 trees

Test 1 passes: Rotations are off by 1 (2362, 2361)
Test 2 passes: Rotations are off by 1 (2358, 2357)
Test 3 passes: Rotations are off by 1 (2363, 2362)
Test 4 passes: Rotations are off by 1 (2366, 2365)
Test 5 passes: Rotations are off by 1 (2363, 2362)
```

A2

For algorithm A2, I didn't go with the binary representation as suggested. Instead, I implemented another mechanism to keep track of the index of nodes.
Step 1 ~ 4 are exactly the same as A1. Before step 5, I mark the root node of all max identical subtrees then I made two list:

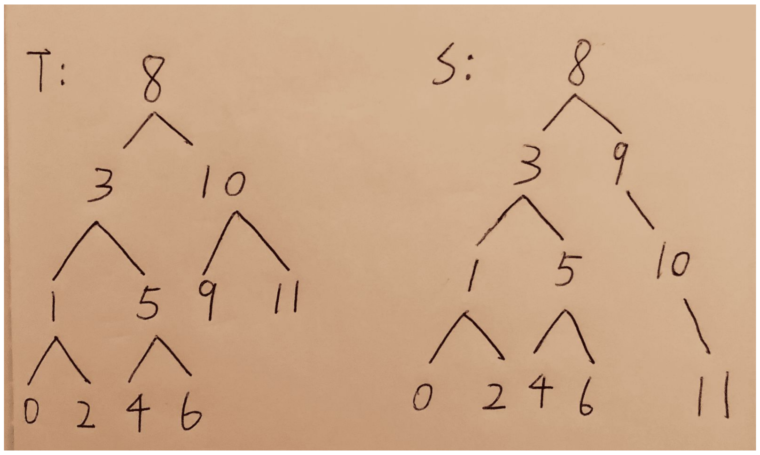
```
fls = []        # nodes not supposed to be on left forearm
frs = []        # nodes not supposed to be on right forearm
```

And then step 5 is done as instructed: any node that belongs to the identical tree is not rotated.
At step 6 the above lists are utilized. Specifically, I pretend I am still processing the original forearm (as in A1), this gives me the correct indexing of the nodes on forearms. When I encounter a node in this "fake" forearm that also belongs to the above lists, I know this node is already at its intended position so it is skipped.

Interestingly, the equation for expected rotation given by the author looks wrong to me:

$$2n - 2\lfloor \log_2 n \rfloor - c_S(\mathrm{root}_T) - 2\sum_{i=1}^{f} n_i + p$$

Specifically, ni is said to be the length of max identical subtree. However such ni can make the equation give a negative number. For example:

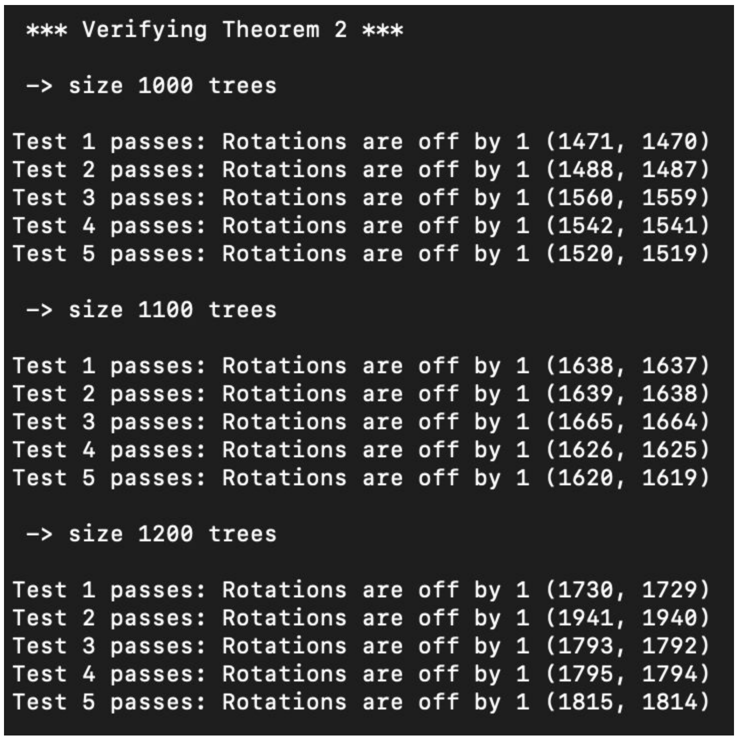


As can be seen, the max identical subtree in this example is the subtree starting at node(3), which has size = 7. With the above equation, we have:

Expected number of Rot = 2*11 - 2*floor(log2(11)) - 6 - 2***7** + 0 = -4

I discovered that if we ignore leaf nodes in max identical subtrees when counting their size, we get the correct expected number of rotations. In this case, ni = 3 gives 4 expected rotations.

With the adjusted ni, A2 also constantly takes 1 less rotation:



```
 *** Verifying Theorem 2 ***

 -> size 1000 trees

Test 1 passes: Rotations are off by 1 (1471, 1470)
Test 2 passes: Rotations are off by 1 (1488, 1487)
Test 3 passes: Rotations are off by 1 (1560, 1559)
Test 4 passes: Rotations are off by 1 (1542, 1541)
Test 5 passes: Rotations are off by 1 (1520, 1519)

 -> size 1100 trees

Test 1 passes: Rotations are off by 1 (1638, 1637)
Test 2 passes: Rotations are off by 1 (1639, 1638)
Test 3 passes: Rotations are off by 1 (1665, 1664)
Test 4 passes: Rotations are off by 1 (1626, 1625)
Test 5 passes: Rotations are off by 1 (1620, 1619)

 -> size 1200 trees

Test 1 passes: Rotations are off by 1 (1730, 1729)
Test 2 passes: Rotations are off by 1 (1941, 1940)
Test 3 passes: Rotations are off by 1 (1793, 1792)
Test 4 passes: Rotations are off by 1 (1795, 1794)
Test 5 passes: Rotations are off by 1 (1815, 1814)
```

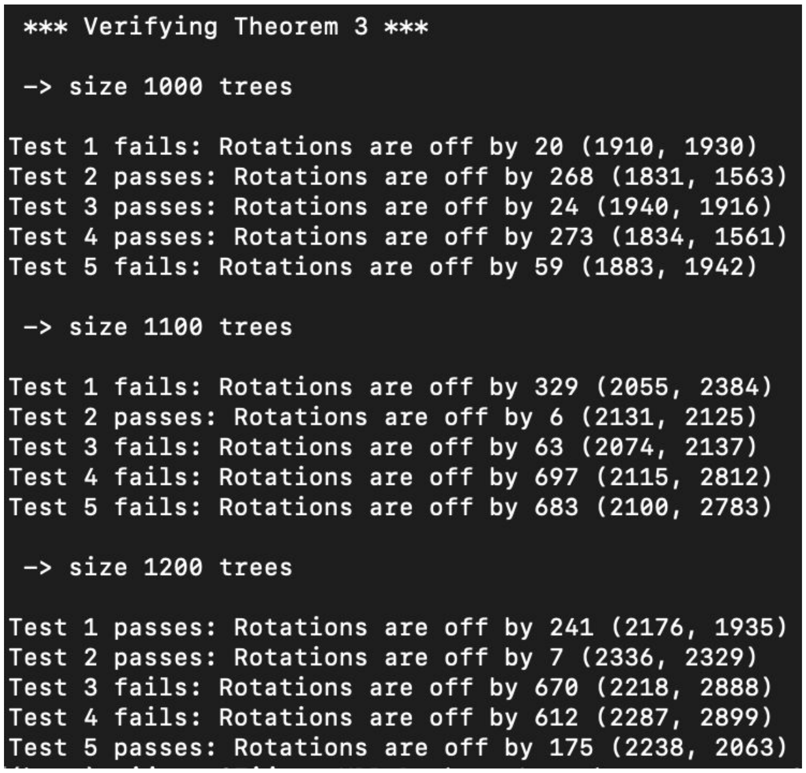
A3

My implementation of A3 mostly followed the author's instruction. First of all, I store the root node of all max equivalent trees. Then I loop through them and fix them with A1.

After this step, any max equivalent tree is transformed into a max identical tree and I simply call A2 on the result array.

Note: I made the same assumption about ni as mentioned in A2.

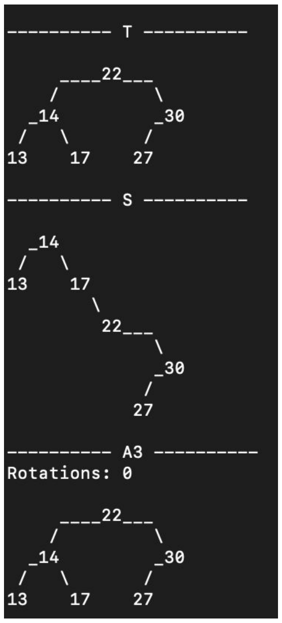
However the result is not as I expected. As can be seen in the next page.



```
*** Verifying Theorem 3 ***

 -> size 1000 trees

Test 1 fails: Rotations are off by 20 (1910, 1930)
Test 2 passes: Rotations are off by 268 (1831, 1563)
Test 3 passes: Rotations are off by 24 (1940, 1916)
Test 4 passes: Rotations are off by 273 (1834, 1561)
Test 5 fails: Rotations are off by 59 (1883, 1942)

 -> size 1100 trees

Test 1 fails: Rotations are off by 329 (2055, 2384)
Test 2 passes: Rotations are off by 6 (2131, 2125)
Test 3 fails: Rotations are off by 63 (2074, 2137)
Test 4 fails: Rotations are off by 697 (2115, 2812)
Test 5 fails: Rotations are off by 683 (2100, 2783)

 -> size 1200 trees

Test 1 passes: Rotations are off by 241 (2176, 1935)
Test 2 passes: Rotations are off by 7 (2336, 2329)
Test 3 fails: Rotations are off by 670 (2218, 2888)
Test 4 fails: Rotations are off by 612 (2287, 2899)
Test 5 passes: Rotations are off by 175 (2238, 2063)
```

Although my implementation of A3 always succeeds in transforming S to the shape of T, it fails on the number of rotations. In the above result screen, the tuple at the end of each test describes the (expected rotation, actual rotation). The theorem suggests that the expected number of rotations is an upper bound, however my A3 sometimes goes over this bound.

I think there are two possible causes of this behavior. First, it is possible that my counter is not working as intended. After careful walkthrough of every step of A3, I discovered that the rotations are done as intended, but the counter is clearly wrong in some situations, as shown in the right: T to S definitely is not 0 rotations.



```
---------- T ----------

     ____22___
    /         \
  _14         _30
 /   \       /
13    17    27

---------- S ----------

  _14
 /   \
13    17
        \
         22___
              \
              _30
             /
            27

---------- A3 ----------
Rotations: 0

     ____22___
    /         \
  _14         _30
 /   \       /
13    17    27
```

Another possible cause is the way I manipulate the equivalent subtree. Since my A1 takes BST object as input, I had to make a new tree and discard the equivalent subtree in S. After A1 is done, I insert these nodes back to S. For tree of larger size, some attributes of the nodes could be damaged during this process, such as the flag indicating whether current node is max equivalent subtree, causing A1 to do more rotations than intended.

Discussion

Based on testing result of size 1000 trees, we can sum up the expected rotation for each algorithm:

- A1: ~ 1900
- A2: ~ 1500
- A3: ~ 1900

Clearly, A2 requires the least amount of rotations. A3 requires roughly the same number of rotations as A1. I think the reason for this is because the way we generate random tree makes it very likely to have large equivalent subtrees, which essentially makes A3 call A1 one the most part of the tree.

Additionally, I think A3 will be more efficient if we change it as following:
For each pair of max equivalent subtrees Si, Ti, find the minimal subtree(s) in Si that makes Si and Ti not identical, and apply A1 on these subtree. Then apply A2 as usual.