CS562: Progress Report

Kazuki Kaneoka
932-277-488

1. What I've Done

Red Black Tree and AVL Tree are two of the most common binary trees in the real world. So, my project is to investigate which binary tree provides better performance in which situation. For the init test, I compared the performance of the insert and remove functions of the two binary trees to get general idea of performances of both trees.

2. Difference Between Red Black Tree and AVL Tree

Red Black Tree and AVL Tree are both self-balanced binary trees. We say tree is balanced if the height difference between left and right children is at most 1 for each node. So, self-balanced tree maintains this property by rotating left or right children after insert or delete node every time. Theoretically, Red Black Tree takes less number of rotations than AVL Tree to keep its balance. Furthermore, the height of AVL Tree is approximately 1.44log(N) at most, and the height of Red Black Tree can be 2log(N) in worst cast where N is number of nodes. To sum up, we can estimate the following results for this performance test.

For the insert function:

- If the keys of the inserting nodes are distinct, Red Black Tree is faster than AVL Tree because Red Black Tree needs less number of rotations.
- If there are many duplicate keys in the inserting nodes, AVL Tree is better than Red Black Tree because the height of AVL Tree is shorter. It implies that to find the location of the inserting node takes less time for AVL Tree.

For the remove function:

- AVL Tree is better than Red Black Tree because of the same reason with inserting duplicates keys.

3. How I Did Performance Test

I created a list using three different functions, listDistinct(low, high, size), listNormal(low, high, size), and listInorder(size), with different sizes, 100, 1000, 10000, and 100000.

- I used listDistinct(low, high, size) to create a list with no duplicate.
- I used listNormal(low, high, size) to create a list with duplicates.
- I used listInorder(size) to create an inorder list.

Then, I inserted each element in the list into Red Black Tree and AVL Tree, removed them from both trees, and measured time.

4. Results Of Performance Test

- timeInRBT: the inserting time of Red Black Tree.

- timeInAVL: the inserting time of AVL Tree.
- timeReRBT: the removing time of Red Black Tree.
- timeReAVL: the removing time of AVL Tree.

As I mentioned in section 2, Difference Between Red Black Tree and AVL Tree, the results were almost what I expected, but not every time. The results showed that sometimes timeInRBT was faster than timeInAVL, and also, timeReRBT was better than timeReAVL.

5. Estimate The Progress By End Of Term

I would focus on do performance test for the results, which is not my expectation. Theoretically, we know that Red Black Tree is faster than AVL Tree for insertion, and AVL Tree is faster than Red Black Tree for removing. However, as I mentioned in section 4, Results Of Performance Test , it is not true every time. So, I would investigate when that situation is happened.

6. Quality Of My SUT, bintrees-2.0.4

So far, bintrees-2.0.4 is well written and reliable. However, its performance is not as fast as other libraries, which also provide Red Black Tree and AVL Tree. Instead of it, bintrees-2.0.4 provides Cython version of both trees, which are written by C and this versions of trees are much faster than Python version.

7. The Code Coverage

According to the results, the init tstl provided the following results:

```
11.2338277059 PERCENT COVERED
323 BRANCHES COVERED
258 STATEMENTS COVERED
```

Furthermore, the coverage.out showed the following lines were missing:

```
abctree.py
396   393   182    1    2%   9-525, 530-820, 529->530

avltree.py
150    24    44    3   86%   30-44, 51, 55, 62, 70, 74, 87, 93-117, 122, 135-
136, 183,    186, 196, 134->135, 185->186, 195->196

rbtree.py
131    20    46    5   86%   35-46, 53, 59, 63, 71, 78, 88, 94-117, 122, 174,
177, 239,    176->177, 206->187, 218->221, 227->235, 238->239
```

According to the coverage.out, I tested 86% lines of avltree.py and rbtee.py. However, the total coverage showed 11.23% because redtree.py and avltree.py were extended from abctree.py.