CS562: Final Report

Kazuki Kaneoka
932-277-488

What to test

I chose bintrees 2.0.4 as my System Under Test: https://pypi.python.org/pypi/bintrees/2.0.4. The library provided AVL Tree and Red Black Tree, and the implementations followed the algorithm from Juienne Walker: http://eternallyconfuzzled.com/jsw_home.aspx. Since the part of the library was written by Cython/C, it is required to install Cython before installing bintrees 2.0.4: http://cython.org.

What I've done

AVL Tree and Red Black Tree were two of the most common binary trees to use in the real world. So, it was siginificant to know that which binary tree was proper for which situation. Therefore, I've focused on performance test of the library rather than testing the llibrary itself.

I checked the performance test by following steps:

1. Create a list of integer.
2. Measure the time of inserting each element in list into AVL Tree and Red Black Tree.
3. Similarly, measure the time of remove each node from both trees.

I used several lists for performance test to see if there was any difference.

1. listDistinct: an integer list with distinct number. For example, if the size of list was 100, each integer in list was between -100 and 100, and there was no duplicate.
2. listDuplicate: an integer list with duplicate number. For example, if the size of list was 100, each integer in list was between -10 and 10, and there was duplicate integer.
3. listPositive: an integer list with positive number. For example, if the size of list was 100, each integer in list was between 1 and 200, and there was no duplicate.
4. listNegative: an integer list with positive number. For example, if the size of list was 100, each integer in list was between -200 and -1, and there was no duplicate.

The following table 1-4 showed a two-sample t test with equal variance for the insert function using each list.

Table 1: Mean of inserting time per second vs. size of list using listDistinct

	100	1000	10000	100000
AVL Tree	0.0021	0.0245	0.3161	3.7883

Red Black Tree	0.0011	0.0146	0.1944	2.5252
P value	2.20E-16	2.20E-16	2.20E-16	3.97E-06

Table 2: Mean of inserting time per second vs. size of list using listDuplicate

	100	1000	10000	100000
AVL Tree	0.0007	0.0116	0.1660	2.2155
Red Black Tree	0.0006	0.0098	0.1377	1.8937
P value	2.20E-16	2.20E-16	2.20E-16	5.78E-09

Table 3: Mean of inserting time per second vs. size of list using listPositive

	100	1000	10000	100000
AVL Tree	0.0022	0.0256	0.3408	4.2452
Red Black Tree	0.0012	0.0152	0.2157	2.8614
P value	2.20E-16	2.20E-16	2.20E-16	5.52E-07

Table 4: Mean of inserting time per second vs. size of list using listNegative

	100	1000	10000	100000
AVL Tree	0.0022	0.0253	0.3501	3.9689
Red Black Tree	0.0011	0.0168	0.2199	2.7290
P value	2.20E-16	2.20E-16	2.20E-16	6.59E-04

The following table 5-8 showed a two-sample t test with equal variance for the remove function using each list.

Table 5: Mean of removing time per second vs. size of list using listDistinct

	100	1000	10000	100000
AVL Tree	0.0015	0.0194	0.2013	2.4565
Red Black Tree	0.0020	0.0276	0.3692	4.6776
P value	2.20E-16	2.20E-16	2.20E-16	4.39E-07

Table 6: Mean of removing time per second vs. size of list using listDuplicate

	100	1000	10000	100000
AVL Tree	0.0003	0.0032	0.0391	0.4531
Red Black Tree	0.0003	0.0049	0.0665	0.8652
P value	2.20E-16	2.20E-16	2.20E-16	2.20E-16

Table 7: Mean of removing time per second vs. size of list using listPositive

	100	1000	10000	100000
AVL Tree	0.0015	0.0183	0.2220	2.6293
Red Black Tree	0.0021	0.0279	0.4051	5.1746
P value	2.20E-16	2.20E-16	2.20E-16	8.99E-15

Table 8: Mean of removing time per second vs. size of list using listNegative

	100	1000	10000	100000
AVL Tree	0.0015	0.0182	0.2215	2.4094
Red Black Tree	0.0020	0.0283	0.4116	4.9441
P value	2.20E-16	2.20E-16	2.20E-16	3.35E-07

What I've found

Theoretically, Red Black Tree is proper for inserting because it take less number of rotations than AVL Tree to maintain its balance. On the other hand, the height of AVL Tree is 1.44log(N) at most, but the height of Red Black Tree is 2log(N) in worst case (N is number of nodes in tree). It means that AVL Tree has advantage for removing because of the time of finding the removing node.

According to Table 1-4, there was strong evidence that average of the inserting time using Red Black Tree was faster than AVL Tree (two-sided P value < 0.005 from a two-sample t test with equal variance).

In contrast, from table 5-8, there was convincing evidence that average of the removing time using AVL Tree was better than Red Black Tree (two-sided P value < 0.005 from a two-sample t test with equal variance).

The statistic summery showed that the theoretical idea about Red Black Tree and AVL tree was correct for whatever duplicate or not and positive or negative.

How well my TSTL file worked for performance test

The python files for AVL Tree and Red Black Tree in bintrees 2.0.4 were avltree.py and rbtree.py in bintrees-2.0.4/bintrees, and they defined insert and remove functions. For performance test of those functions, my TSTL file worked well. However, bintrees 2.0.4 covered not only AVL Tree and Red Black Tree such as the functions using Cython. My TSTL file didn't cover them. So, it could be improved by covering them.

Comments about TSTL: pros and cons

TSTL was nice because it could hide detail from user. User just needed to know how to use SUT on TSTL. It would be also nice if TSTL supported command line arguments because we could use shell script for performance test. It is common for performance test to run program with different input size. It might be helpful not only performance test. For example, suppose there is some known bug in some

library. We can run program using the library with different input size to see whether the bug is occurred for all input size or only when the input size is bigger than some number. Another thing that I would like to have was concurrency test. If TSTL could simulate testing under synchronizing, we could test how SUT worked under distributed system.

Coverage summary

coverage.out, branches covered, and statements covered for listDistinct:

```
Stmts   Miss Branch BrPart  Cover  Missing
-----------------------------------------------------------------
/Library/Python/2.7/…/bintrees/abctree.py
396    393    182      1     2%    9-525, 530-820, 529->530
/Library/Python/2.7/…/bintrees/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
/Library/Python/2.7/…/bintrees/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
-----------------------------------------------------------------
TOTAL
2175   1916    902      9    12%

323 BRANCHES COVERED
259 STATEMENTS COVERED
```

abctree.py was abstract class such that avltree.py and rbtree.py were extended from it. It provided other trees and other functions. This was the reason of the low coverage percent for it.

avltree.py and rbtree.py missed some lines. However, they were meaningless such as comment out, importing other libraries. Also, they didn't cover some lines in the insert and remove functions. They didn't matter because they were if statements for inserting and removing same keys, which were not happened in listDistinct case.