Sean Penney
CS 562 Final Report

I was able to accomplish most of the testing of methods in the BinaryTree and RBTree classes that I wanted to, and was able to do performance testing for both classes. In particular, performance testing was done by timing methods in the BinaryTree/RBTree classes and comparing the results to using a python dict. Mostly missing from the final version is testing on set methods. I became hesitant to include these, because I would have had to write potentially buggy code on top of the dict class in order to do the comparisons between BinaryTree/RBTree and dict. There is some code I had to include on top of the basic dict type for the existing differential testing, however, it is not complicated so I don't believe it would introduce any bugs. Methods that were tested include insert(), get(), len(), max(), min(), remove_items(), is_empty(), clear(), foreach().

One found bug turned out to be a false positive, it appears that one of the guards is not being checked properly. Specifically, there is a guard to check that an item exists in the binary tree before removing the item. The code is as follows:

~<binarytree>.get(<letter>) > 0 -> ~<binarytree>.remove_items(<letter>)

The get() method on the binary tree should only return a number greater than zero if the key exists in the tree. Nevertheless, when running the tests, tstl occasionally finds a bug when calling the remove method. As it turns out, the key it was trying to remove was not in the binary tree, however. An example of this can be found in the 'remove_bug_false_positive.txt' file. This problem occurred with both the BinaryTree and RBTree implementations.

A bug was found for the min() method, and in this case there does not seem to be a bug with the test code or tstl. I was able to run the reduced test case in the python interpreter, and came up with the same result. To summarize, two ints (5 and 14) were inserted into the binary tree. The min() method for the BinaryTree class returned 14 as the min, however. The dict class being used as a referenced returned 5 as the min value, which resulted in an assertion being thrown. A similar problem occurs for the max() method. The same problem occurred for the RBTree, so they must have the same underlying implementation problems.

Another bug seems to be a false positive, also because of the guard not being checked properly. Similar to the tstl code for the remove_items() method, before calling the min() function a check a user-defined guard should be executed to make sure that the size of the binary tree is greater than 0. It is possible that the get() method is not functioning correctly, rather than the guard not being checked. The tstl code for the RBTree also had the same issue.

For performance testing, I first tested the insert method for the BinaryTree/RBTree. The average insert times were written to a file, and I looked at both the max and min values.

	BinaryTree	RBTree	Dict
Insert() max	.0049	.0030	.0040
Insert() min	$3.124 * 10^{-5}$	$2.94 * 10^{-5}$	$2.856 * 10^{-5}$

As we can see, the RBTree indeed performs slightly better than the BinaryTree. While the python dict clearly performs better than the BinaryTree class, it is not clear whether is it better than the RBTree.

Overall, the tester seems to be working alright, however, there are some concerns I have. The performance tests seem to be working well, since those results are roughly what was expected. The main concern would be why the guards are not being checked properly. If the get() method is incorrectly returning numbers greater than zero, that would be another bug with the bintrees library. Otherwise, for some reason tstl is not handling my guard properly.

The tester is correctly generating random integers and characters in the ranges specified, which are correctly used in the methods on the BinaryTree/RBTree classes. As can be seen from the output of reduced test cases, the methods specified in the tstl file are being called as they should be (insert, remove, max, min, etc). The dict is properly being compared as a reference, as can be seen from the output from when the max and min methods on the binary tree fail. The reference class consistently returns the correct value, and the assertion then fails when the BinaryTree/RBTree return incorrect values.

The main thing wrong with tstl would be the lack of documentation. The examples in the tstl repository do an alright job of explaining the syntax, but more formal documentation would be nice. Obviously this is different, there is the saying "An API is only as good as its documentation." As for more technical aspects, it would be nice if properties could be selectively turned on and off.

Another issue I ran into, that I think would be something to improve with tstl, is specifying the methods to be used by the reference objects. In particular, sometimes I would need the parameters to be passed in a slightly different. For example, I may want the same character and letter passed to each method but I would want the order to be different. I wasn't sure if I could specify how the parameters were passed or if they had to be passed the same way in the reference methods.

One other idea I thought of was having some sort of macro for doing the timing for certain actions. If I could somehow signify that I wanted certain actions to be timed, and the timing data to be logged that would be useful.

One good thing I liked about tstl is the ability to mix tstl syntax with python, and it was very streamlined. Obviously it was nice that as a programmer I didn't have to put much effort into generating the values to feed the methods I was testing. There is a lot of potential in tstl as it is expanded to support other languages.