Concurrent Tree Algorithms

Jose Camacho, Dhruv Sandesara, Jae Lim, Matthew Hall, Megan Peregrino

Our task was to implement concurrent tree algorithms for binary search trees and red black trees. More precisely, we implemented

1) Fine grained lock-based binary search trees
2) Lock-free binary search trees
3) Fine grained lock-based red black trees
4) Lock-free red black trees

There's a separate directory for each one of these, and more precise READMEs can be found inside these directories.

How to run the code

Fine grained lock-based BST:
The main method is located in FineGrainBST.java, so just compile all the included java files and run FineGrainBST through the command line or an IDE with the desired parameters, described below.

The fine grain lock implementation of the binary search tree has three parameters to test the code. The first parameter is the number of threads created to access the tree, (this number must be positive and non-zero). The second parameter determines the number of insert operations that each thread will execute, (negative numbers will halt thread execution without any operations). The third parameter determines the specific test case, (0-2) to use with the previously specified parameters. Zero specifies a test case where a number of random elements are inserted into the tree and deletes occur during the insert process. One specifies the test case in which duplicate numbers are repetitively inserted. Two specifies a test case where a large range of numbers are inserted into the tree.

Lock-free BST:
The main method to test the code is located in Test.java. To run the code, just compile all the included files and run Test through the command line or an IDE. No parameters are needed in this one; to change different values, the main method in Test.java must be modifed.

The main function in Test.java will run three tests.
The first test runs insert on a given number of threads, and inserts a given number of keys per thread.
The second test runs find on random numbers within the given key range.
The third test runs delete similarly to the insertion test.
These tests were also used to compare runtimes with the Lock-Based BST. To compare runtimes, we ran three tests on 8 threads, each doing one insert operation, for 10, 100, 1000, or 10000 operations.
The results of our runtimes are attached in the folder, as well as the README.

Fine grained lock-based red black trees:
The main method to test the code is located in TestTrees.java. To run the code, compile all the included java files and run TestTrees through the command line or an IDE. No parameters are needed for this

one; to change different values such as number of threads, a designated section in main method in TestTrees.java must be modified. Note that some methods from JUnit5 were used, so Junit5 must be included in order for the code to compile and run properly. This is easy in most IDEs and can be easily Googled. To include it in the command line through the javac and java commands, run

```
javac -cp junit.jar *.java
java -cp junit.jar:. TestTrees
```

in the source directory with the java files and a junit.jar file that can be downloaded from the official JUnit website. The first line will compile our code, and the second line will run test cases.

For test cases, we initially try to add unique elements and assert that they are added successfully. Then we do a search and insert of other elements concurrently to see if both can be efficiently run together. Also search is done on the previous unique elements to check if the previously inserted elements are found.

There are variables in the main method of TestTrees.java that can be modified:
Modify run num to run the tests multiple times
Modify num treads to change the concurrent threads
Modify insert_nodes_per_threads to change how many elements the threads add
Modify the display_tree to see a visual representation of the rbtree which looks decent with a small number of nodes, ugly with a large number of nodes, and will immensely slow down the machine if a large amount of trees are being drawn.

The tests will print:

* Number of threads used.
* Nodes inserted by each thread.
* Time for all insert operations to end.

For an example, look into the more detailed README available in the FineGrainedLockRBTree directory.

Lock-free red black trees:

To run our code, compile all the included java files and run TestRedBlackTree, which contains the main method for testing our code. There are no extra parameters needed for this one.

Sometimes insert will deadlock and might need to restart, delete never has that problem.

Can change int num_threads in TestRedBlackTree.java to change number of threads used in insert, search, and delete operations.

Will print:

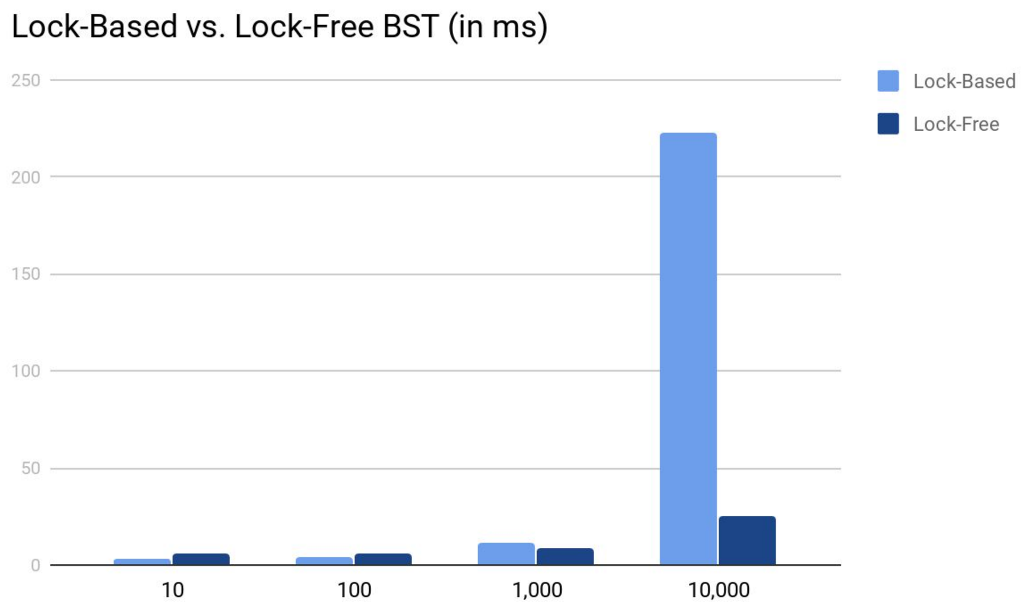
* Number of threads used.
* Nodes inserted by each thread.
* Time for all insert operations to end.

* LevelOrder representation of RedBlack Tree after insert
* InOrder representation of RedBlack Tree after insert
* PreOrder representation of RedBlack Tree after insert
* PostOrder representation of RedBlack Tree after insert
* ThreadID found which integer in tree
* ThreadID, corresponding Node deleted, and LevelOrder reperesentation after insert

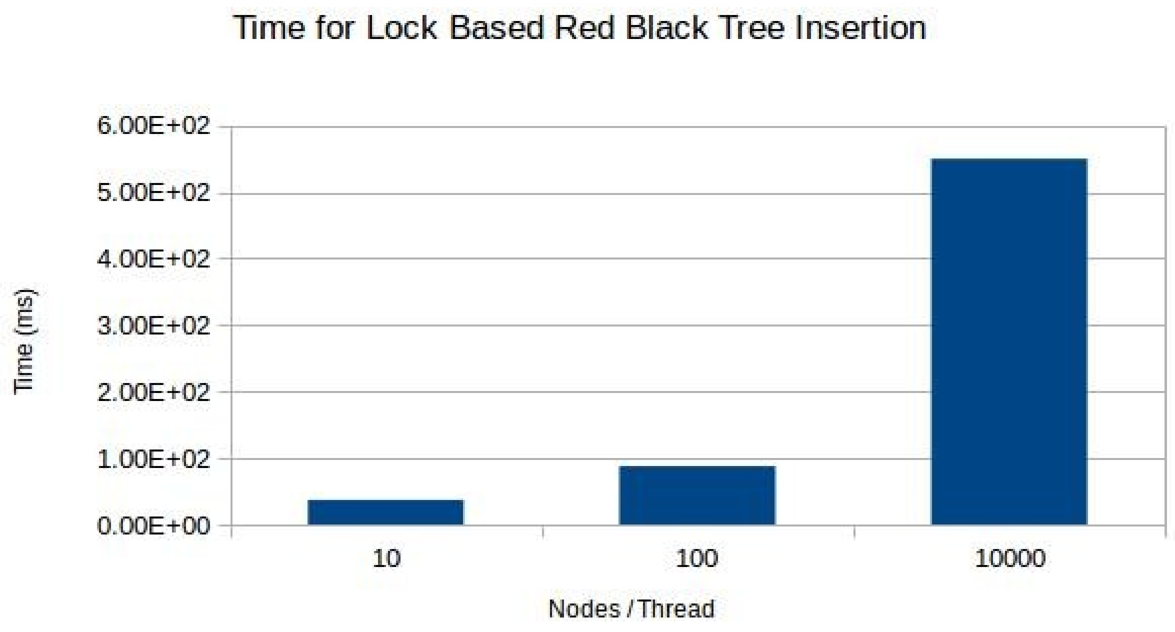
For an example, look into the more detailed README available in the RedBlackTree-LockFree directory.

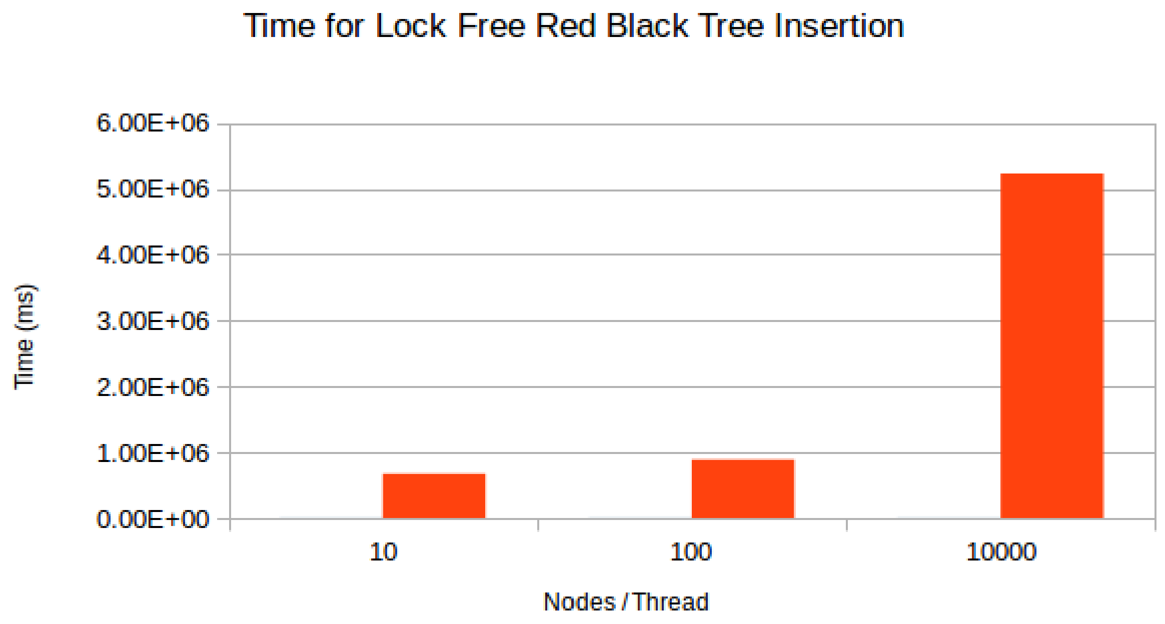
<u>Graphs</u>

Binary Search Tree Comparisons:



Red Black Tree Comparisons:


Time for Lock Based Red Black Tree Insertion
6.00E+02
5.00E+02
4.00E+02
3.00E+02
2.00E+02
1.00E+02
0.00E+00
Time (ms)
Nodes / Thread


Time for Lock Free Red Black Tree Insertion
6.00E+06
5.00E+06
4.00E+06
3.00E+06
2.00E+06
1.00E+06
0.00E+00
Time (ms)
Nodes / Thread