APPENDIX D: EDUCATIONAL MATERIAL

Conflicting Inserts in a Lock-Free BST Example

Research into lock-free binary search trees is still relatively new. In fact, the first "complete, non-blocking, linearizable BST implementation using only reads, writes, and single-word compare-and-swap (CAS) operations" was introduced in 2010 by Faith Ellen, Panagiota Fatourou, Eric Ruppert, and Franck van Breugel [1]. One of the major issues in implementing lock-free binary search trees is that deletes on a node with two children are very complicated, because the algorithm has to find a successor node, promote it, and then delete the successor node requiring it to touch and modify multiple nodes. Ellen, et. al. use an external tree binary search tree to avoid this issue. An external binary search tree uses only the leaf nodes for storing the values and uses the internal nodes simply for routing. For example, in Figure 1 below, the values in the binary search tree are 2, 3, 5, 6, 9. The other values help find values in the tree otherwise are not meaningful. Nevertheless, deletes are still relatively complex, so in this section we discuss the simpler modification operation, insertion.

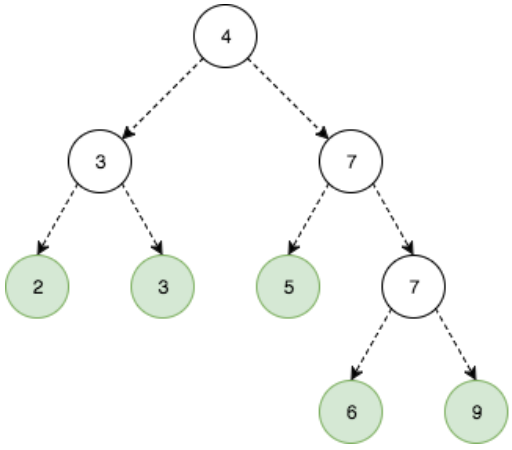

Figure 1: A leaf-based binary search tree.

During an insert operation in an external-lock free binary search tree, a thread will use a compare-and-swap (CAS) instruction to leave information about its operation, so that if another thread interrupts and attempts to perform a conflicting operation, the other thread can finish the original operation before doing its own operation.

We now present an example to show how conflicting inserts would occur in this tree in a system with two threads. Suppose the tree we have the tree shown in Figure 1. Further suppose that Thread 1 wants to add 10 to the tree and Thread 2 want to add 11 to the tree. Since both 10 and 11 "belong" in the same spot (to the right of 9), we need to ensure that both updates are preserved if these updates occur concurrently. Furthermore, we need to maintain 9 as a value in the tree, meaning that after the insert, 9 still needs to be in a leaf. Suppose Thread 1 is scheduled first. Thread 1 first searches for where to insert its value, 10. It traverses to 9 and remembers the parent of 9 (the 7). Thread 1 then creates three new nodes shown in figure 2 to insert into the tree such that 9 and 10 are both leaves and the tree maintains the BST property. Now suppose Thread 2 is scheduled. Thread 2 traverses to 9 and remember the parent of 9 (the 7). Thread 2 also creates three new nodes to insert into the tree as shown in figure 3. Now Thread 2 uses a single CAS operation to flag the parent of 9 and leave the information on how to complete its update. Now suppose Thread 1 is scheduled again. Thread 1 picks up where it left off and attempts to flag the parent of 9 and leave information on its insert instruction. However, the parent of 9 has already been flagged and this operation fails. Recall that when Thread 2 flagged the parent of 9, it also left the information on how its three nodes were structured. Thus, Thread 1 will now read that information and use another CAS operation to swing 7's right pointer to the new set of nodes. Then Thread 1 uses a CAS operation to unflag 7. The tree now looks are shown in figure

4. Now Thread 1 starts over from the beginning, performing a new search, creating a new set of nodes, using a CAS to flag the new parent and leave information, using a CAS operation to swing to the new pointer, and finally unflag the new parent. When Thread 2 is scheduled again, Thread 2 will attempt to use a CAS operation to swing the pointers and unflag its parent, but its operation will fail since the expected value provided by Thread 2 will not be the current value. Figure 5 shows the final tree. Providing pseudo-code for this algorithm is left as an exercise to the reader.

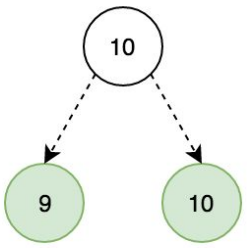

Figure 2

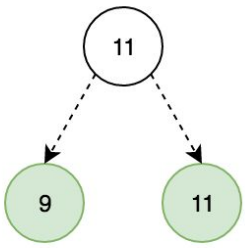

Figure 3

Figure 2 to the left shows Thread 1's insert information and **Figure 3** to the right shows Thread 2's insert information.

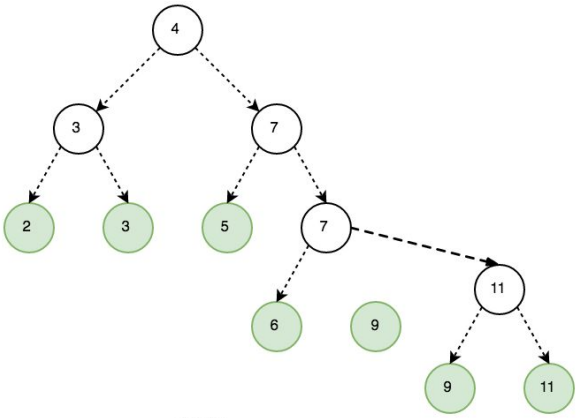

Figure 4: The tree after Thread 1 inserts Thread 2's update.

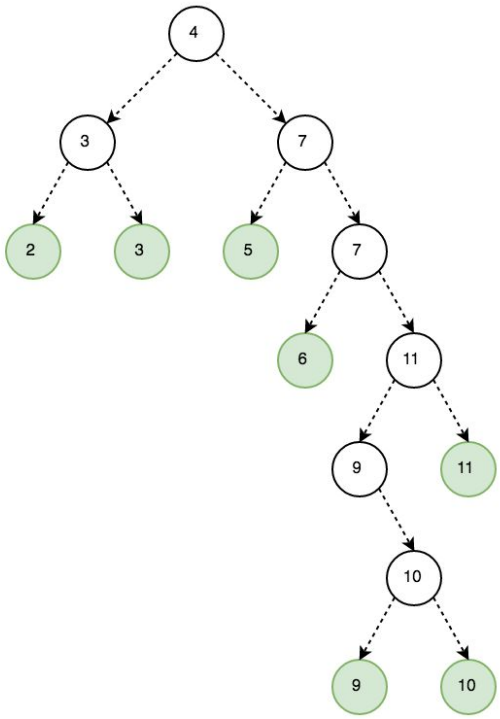

Figure 5: Final tree after both inserts.

References

[1] F. Ellen, P. Fataourou, E. Ruppert, and F. van Breugel. Non-Blocking Binary Search Trees. In Proceedings of the 29th ACM Symposium on Principles of Distributed Computing (PODC), pages 131–140, July 2010.