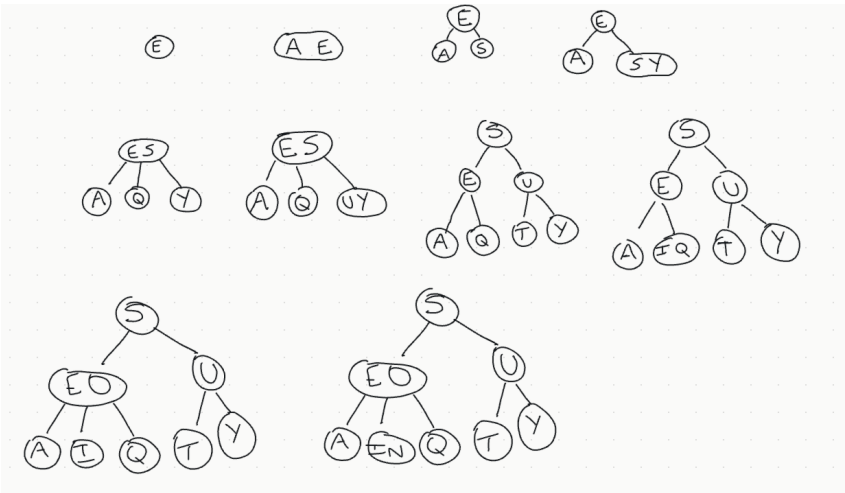

Figure 1: Sequence of 2-3 trees when inserting the keys in Exercise 1.

Sergio E. Garcia Tapia
Algorithms by Sedgewick and Wayne (4th edition) [SW11]
December 31st, 2024

3.3: Balanced Search Trees

Exercise 1. Draw the 2-3 tree that results when you insert the keys `E A S Y Q U T I O N` in that order into an initially empty tree.

Solution. See Figure 1

Exercise 2. Draw the 2-3 tree that results when you insert the keys `Y L P M X H C R A E S` in that order into an initially empty tree.

Solution. See Figure 2

Exercise 3. Find an insertion order for the keys `S E A R C H X M` that leads to a 2-3 tree of height 1.

Solution. See Figure 3.

First, consider the the sort of the keys is `A C E H M R S X`. Since there are 8 keys, we can obtain a tree of height 1 if we have four 3-nodes. In particular, `E` and `R` are at the root, and `A` and `X` are in the leftmost and rightmost leaves, respectively. We can begin by inserting `AEX`, which places `E` at the root as desired. We can then insert `R` and `H`. Both fall in the rightmost leaf, and the split of the 4-node that is created causes `R` to move to the root node, and the creation of the new leaf `H`. Now inserting `C` yields the 3-node with `A` and `C`, inserting `M` yields the node with `H` and `M`, and inserting `S` yields the node with `S` and `X`.

Exercise 4. Prove that the height of a 2-3 tree with n keys is between $\lfloor \log_3 n \rfloor \approx 0.63 \lg n$ (for a tree that is all 3-nodes) and $\lfloor \lg_n \rfloor$ (for a tree that is all 2-nodes).

Solution.

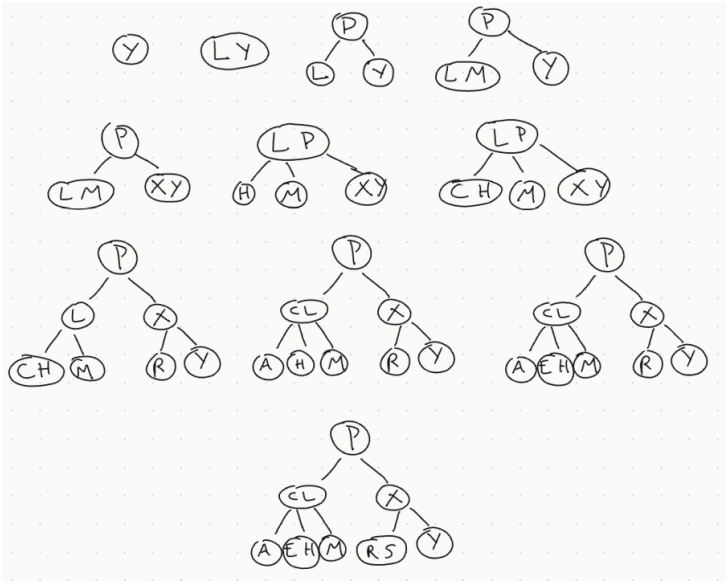

Figure 2: Sequence of 2-3 trees when inserting the keys in Exercise 2.

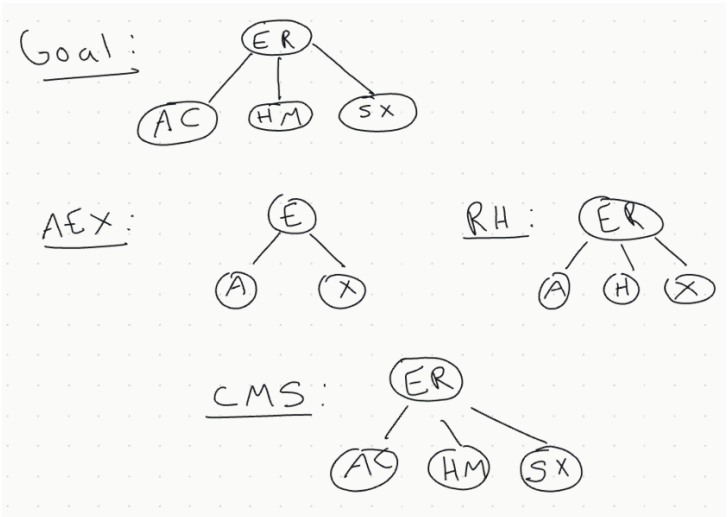

Figure 3: Tree of height 1 for keys in Exercise 3.

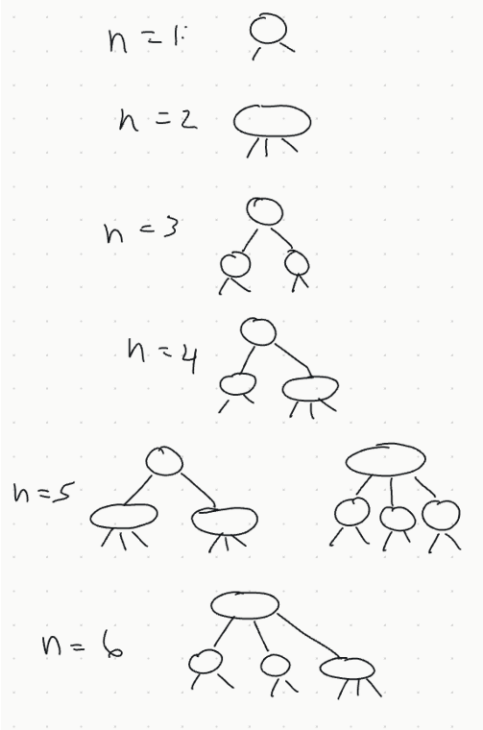

Figure 4: All of the structurally different 2-3 trees with n keys, for n from 1 through 6.

Proof. Suppose T is a 2-3 search tree. By construction, a 2-3 search tree is perfectly balanced, so T is perfectly balanced also. In the worst case, if T consists of only 2-nodes, then it will have 2^k nodes at all depths (except possibly the last one) because it is balanced. Similarly, in the best case, if T consists only of 3-nodes then it will have 3^k nodes at all depths (except possibly the last one). The former tree has a height of $\lfloor \lg_n \rfloor$, and the latter has a height of $\lfloor \log_3 n \rfloor$. We conclude that T's height lies in between the two. □

Exercise 5. Figure 4 shows all the *structurally different* 2-3 trees with n keys, for n from 1 up to 6 (ignore the order of the subtrees). Draw all the structurally different trees for $n = 7$, 8, 9, and 10.

Solution. See Figure 5.

Exercise 6. Find the probability that each of the 2-3 trees in Exercise 3.3.5 is the result of the insertion of n random distinct keys into an initially empty tree.

Solution. See Figure 6 and Figure 7, both of which I have annotated. Note that when a given tree can lead to more than one structurally different tree, I have annotated the arrows indicating the probability with which it can give rise to said structure. This "transitional probability" then plays a role in the computation of the probabilities of the resulting tree.

Exercise 8. Show all possible ways that one might represent a 4-node with three 2-nodes bound together with red links (not necessarily left-leaning).

Solution. See Figure 8.

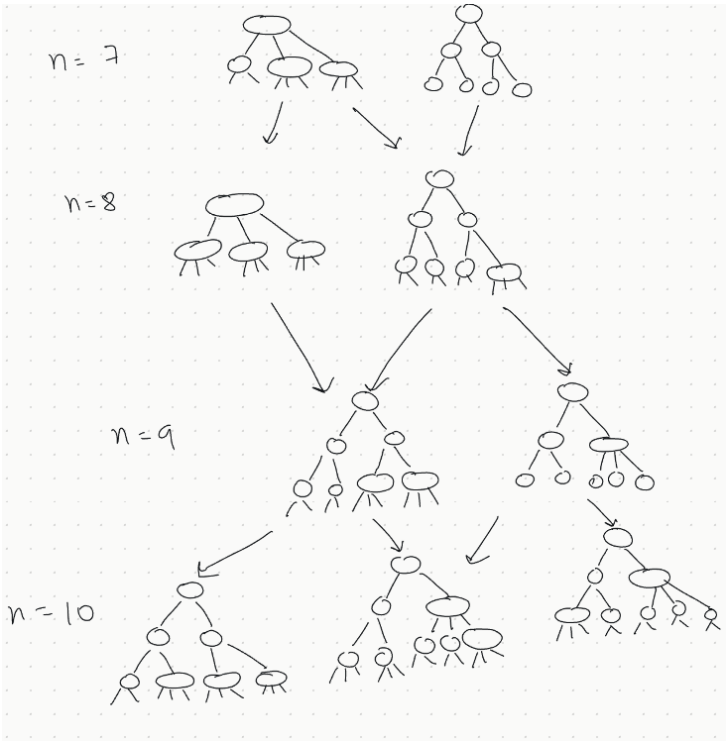

Figure 5: All of the structurally different 2-3 trees with n keys, for n from 7 through 10.

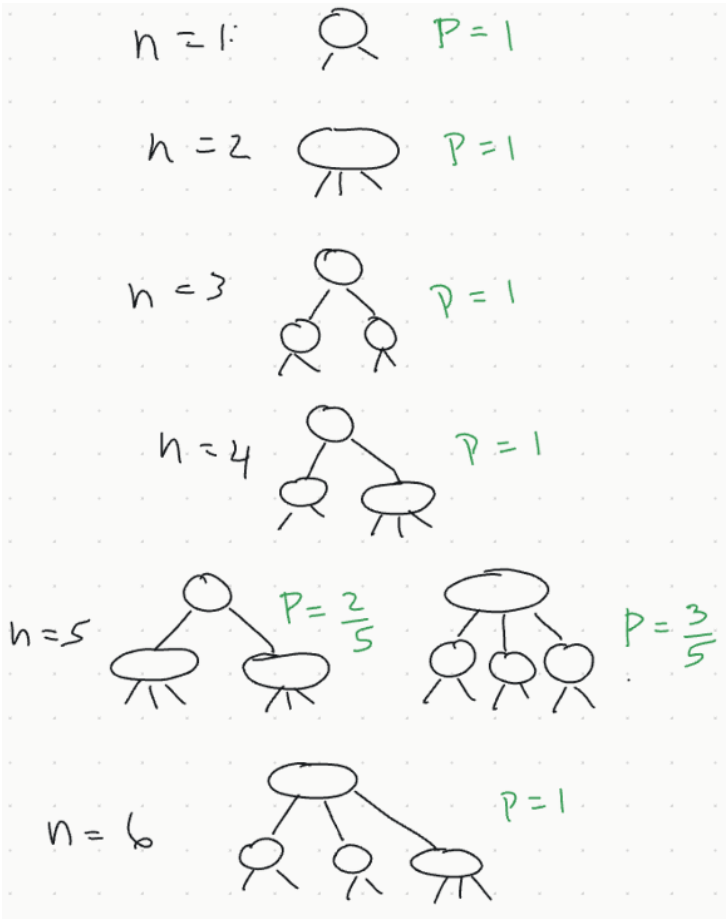

Figure 6: All of the structurally different 2-3 trees with n keys, for n from 1 through 6 and their probabilities.

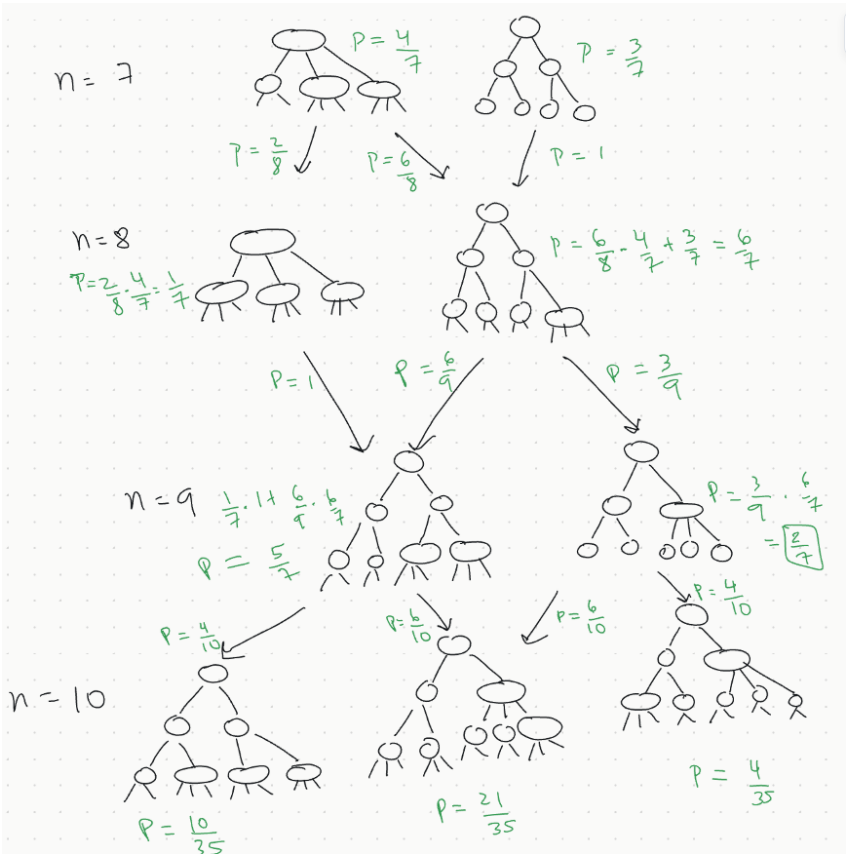

Figure 7: All of the structurally different 2-3 trees with n keys, for n from 7 through 10 and their probabilities.

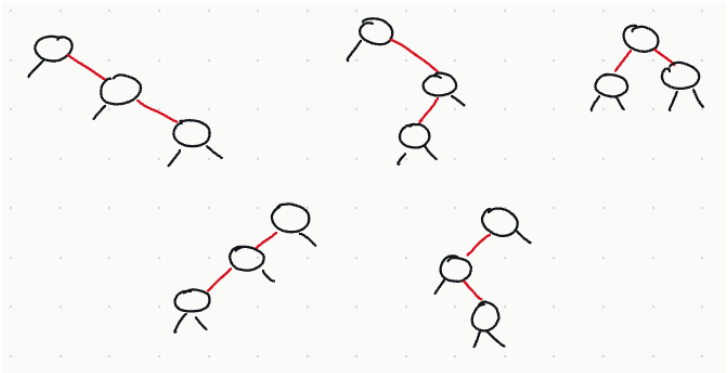

Figure 8: All ways to represent a 4-node with three 2-nodes bound by red links.

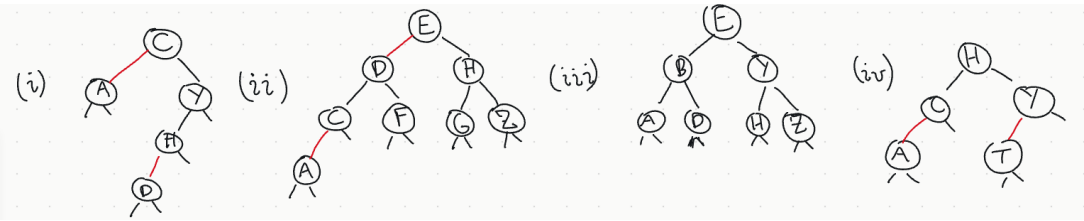

Figure 9: Options for Exercise 9.

Exercise 9. Which of the trees in Figure 9 are red-black BSTs?

Solution. Tree (i) is not a red-black BST because it does not have perfect black balance. Tree (ii) is not a red-black BST because it is not ordered. In particular, `F` is on the left subtree of `E` instead of right. The remaining trees are red-black BSTs.

Exercise 10. Draw the red-black BST that results when you insert items with the keys `E A S Y Q U T I O N` in that order into an initially empty tree.

Solution. See Figure 10.

Exercise 11. Draw the red-black BST that results when you insert the keys `Y L P M X H C R A E S` in that order into an initially empty tree.

Solution. By using the 1-1 corresponding of 2-3 trees and red-black BSTs, we can use the 2-3 tree in Exercise 3.3.2 to create the corresponding red-black BST, depicted in Figure 11.

Exercise 13. True or false: If you insert keys in increasing order into a red-black BST, the tree height is monotonically increasing.

Solution. True. Because the keys are inserted in increasing order, they always fall on the rightmost child of the rightmost leaf. However, such a tree would have a right-leaning red link, so it is rotated left to complete the insertion. When the link is rotated left, the height remains unaffected.

As the insertions continue, there will never be a tree with two consecutive right-leaning links, because the previous insertion would rotate the link left. We will have to perform color flips, but they do not change the height either. The color flips will create new right-leaning red links, which may need to be handled by rotation, but the tree height is not decreased as a result of this.

One effect that color flips have, however, is that because they always occur in a right subtree (due to the pattern of inserting keys in increasing order), they will create right-leaning red links. When these occur further up the tree, there is potential for them to decrease the tree height. However, this does not happen because when they occur the subtrees differ by height of 1 (with the right subtree being the larger), so the rotation to the left does not affect the overall tree height.

Exercise 14. Draw the red-black BST that results when you insert letters `A` through `K` in order into an initially empty tree, then describe what happens in general when trees are built by insertion of keys in ascending order.

Solution. See Figure 12 See the explanation in Exercise 13.

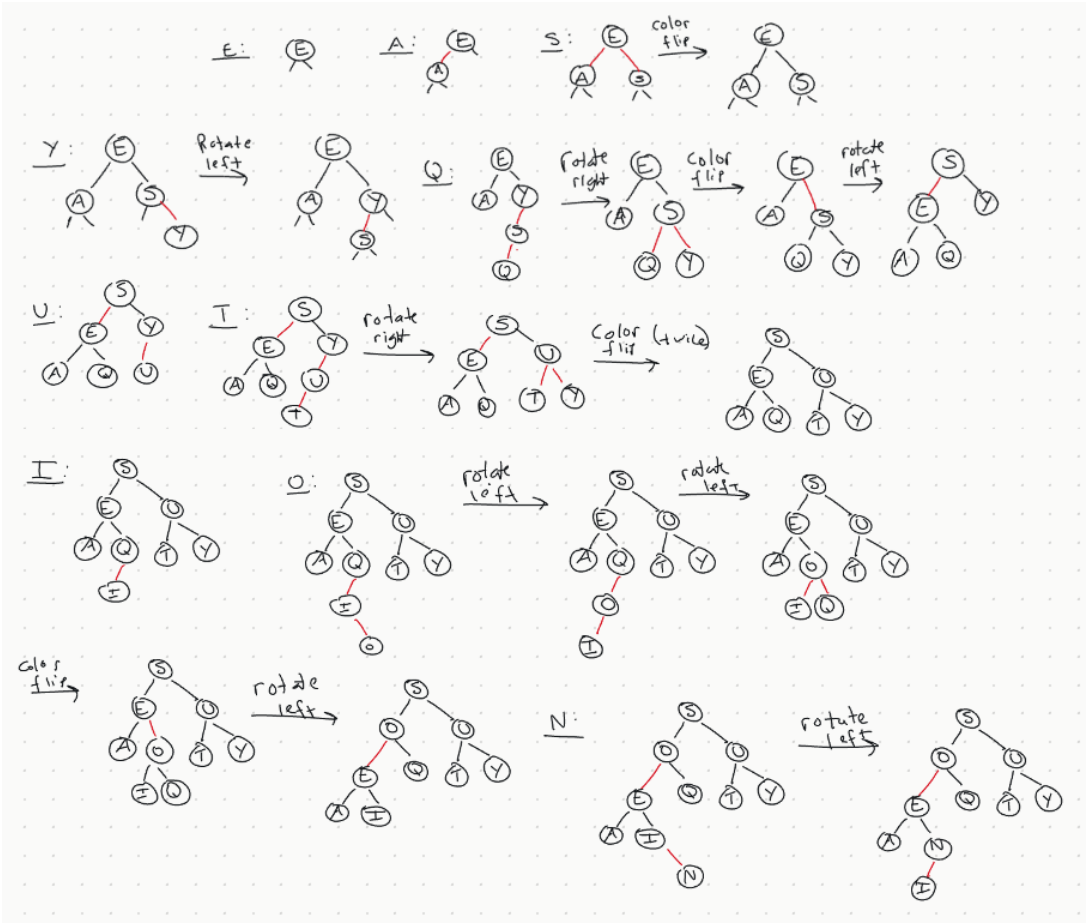

Figure 10: Sequence of red-black BSTs generated by the key sequence in Exercise 10.

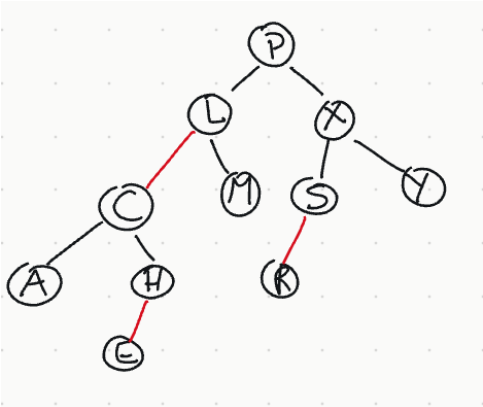

Figure 11: Red-black BST resulting from the key sequence in Exercise 11.

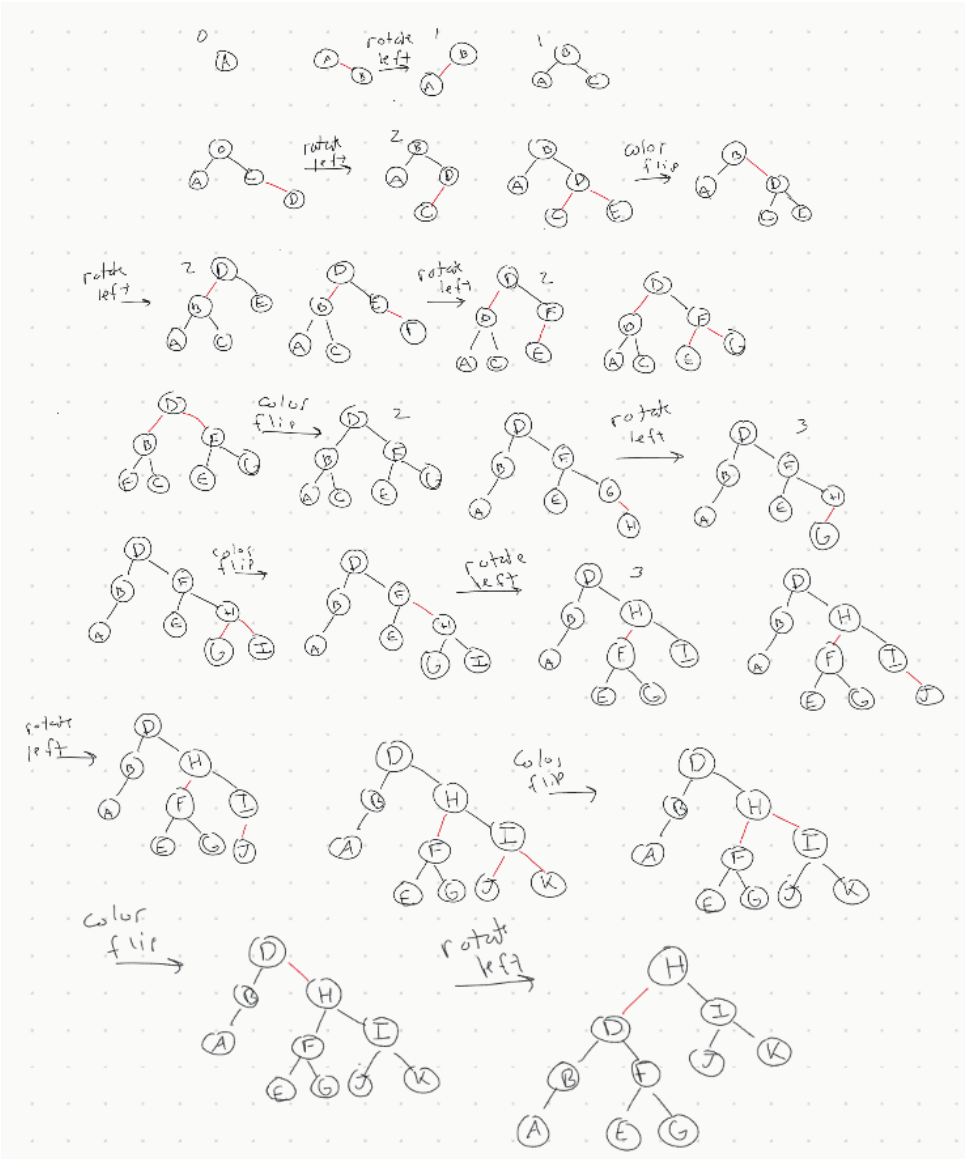

Figure 12: Inserting keys in decreasing order into a red-black BST.

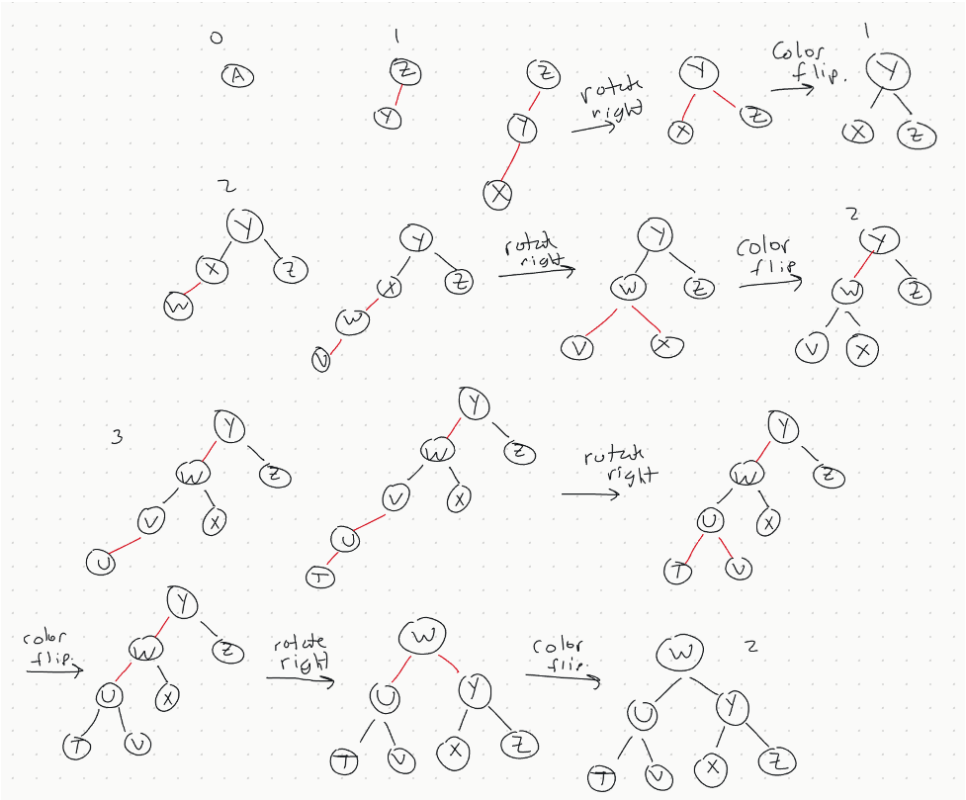

Figure 13: Inserting keys in decreasing order into a red-black BST.

Exercise 15. Answer the previous two questions for the case when the keys are inserted in *descending* order.

Solution. The tree height is not monotonically increasing. In Figure 13, we see that after `T` is inserted, the sequence of transformations takes us from a tree of height 3 to a tree of height 2. The reason is that since left-leaning red links are valid, they are allowed to remain. Since all insertions result in left-leaning red links, we will often encounter two consecutive left-leaning red links, which must be handle via a right rotation. At the bottom of a tree, this does not affect the height. However, the further rotations triggered can change the height, such as the one happening at the root in the figure. There, the rotation makes `W` the root, and the leaves of greatest depth in the tree are the leaves in `W`'s subtree. The rotation decreases their depth by 1, and hence the height of the overall tree.

Exercise 20. Compute the internal path length in a perfectly balanced BST of n nodes, when n is a power of 2 minus 1.

Solution. Let $n = 2^m - 1$. Then the BST has height $m - 1$. Such a tree is a perfect binary tree. At depth k there are 2^k nodes, so the internal path length is

$$\sum_{k=0}^{m-1} k \cdot 2^k$$

Exercise 23. *2-3 trees without balance restriction.* Develop an implementation of the basic symbol-table API that uses 2-3 trees that are not necessarily balanced as the un-

derlying data structure. Allow 3-nodes to lean either way. Hook the new node onto the bottom with a *black* link when inserting into a 3-node at the bottom. Run experiments to develop a hypothesis estimating the average length in a tree built from n random insertions.

Solution. See `com.segarciat.algs4.ch3.sec3.ex23.TwoThreeST`.

Exercise 25. *Top-down 2-3-4 trees.* Develop an implementation of the basic symbol-table API that uses balanced 2-3-4 trees as the underlying data structure, using the red-black representation and the insertion method described in the text, where 4-nodes are split by flipping colors on the way down the search path and balancing on the way up.

Solution. See `com.segarciat.algs4.ch3.sec3.ex25.TwoThreeFourST`

Exercise 26. *Single top-down pass.* Develop a modified version of your solution to **Exercise 3.2.25** that does *not* use recursion. Complete all the work splitting and balancing 4-nodes (and balancing 3-nodes) on the way down the tree, finishing with an insertion at the bottom.

Solution. See `com.segarciat.algs4.ch3.sec3.ex26.TwoThreeFourST`

Exercise 31. *Tree drawing.* Add a method `draw()` to `RedBlackBST` that draws red-black BST figures in the style of the text (see **Exercise 3.2.38**).

Solution. See `com.segarciat.algs4.ch3.sec3.ex31.RedBlackBST`

References

[SW11] Robert Sedgewick and Kevin Wayne. *Algorithms*. 4th ed. Addison-Wesley, 2011. ISBN: 9780321573513.