CS 61B Spring 2026

BSTs, 2-3 Trees, LLRBs

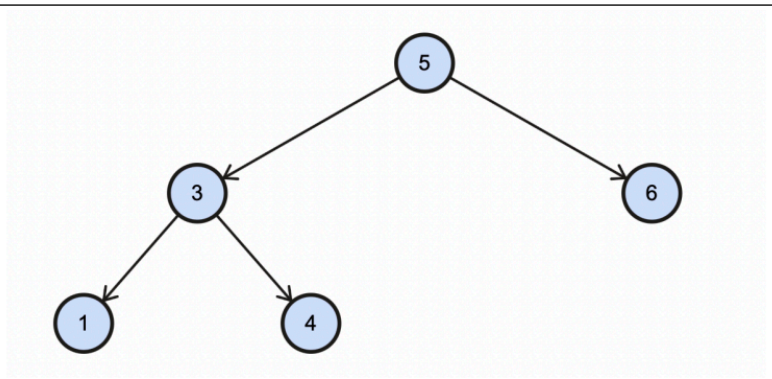
Discussion 08: March 9, 2026

1 Binary Search Trees

(a) We implement a binary search tree with the following methods:

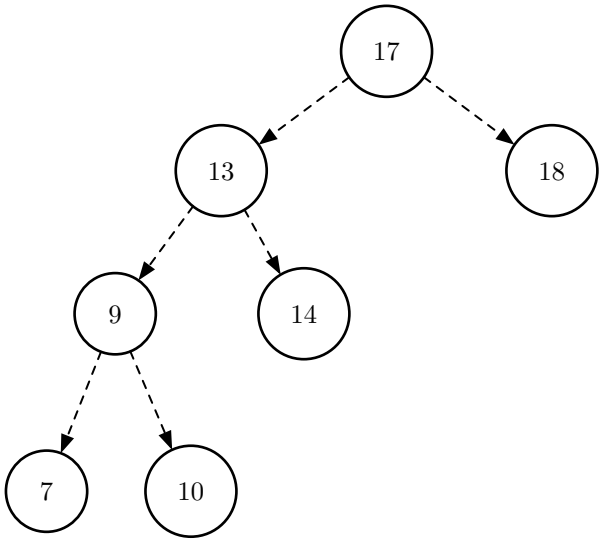
```
// Inserts an item into the binary search tree.
public void insert(T item) {
    // Implementation has been omitted
}

// Deletes an item from the binary search tree.
public void delete(T item) {
    // Implementation has been omitted
}
```

Draw the binary search tree that results from the following operations. Assume we start from an empty tree.




```
insert(5)
insert(7)
insert(10)
insert(6)
insert(3)
insert(1)
insert(4)
delete(10)
delete(7)
```

(b) Given the following binary search tree:



Suppose we delete the root node. Which node(s) can we replace 17 with as the new root node?

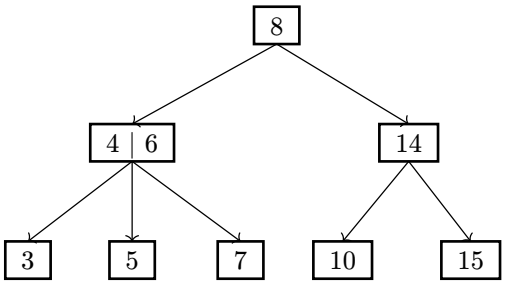
Upon deleting the root node, we can promote either the node containing 14 or 18 as our new root node. When choosing a new root node, as per BST properties, the new root must be greater than all nodes on the left subtree and less than all nodes on the right subtree (this is the working behind Hibbard Deletion from lecture).

(c) Suppose we create a BST by inserting the nodes $V_0, V_1, ..., V_n$, where V_i is strictly smaller than $V_{\{i+1\}}$, in order. That is, we first insert V_0, then V_1, and so on. What is the runtime to find an element in this BST in the worst case, where N is the number of nodes?

The runtime for a search on this BST is $\Theta(N)$. This is essentially a linked list, where in the worst case the item being searched for is at the very end of the list.

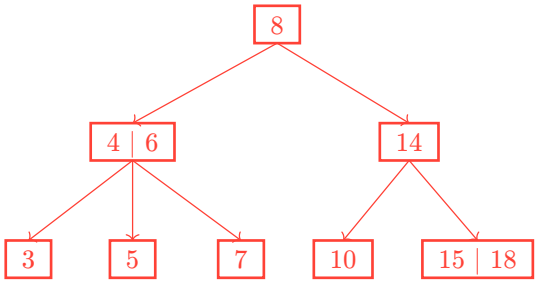
2 2-3 Trees and LLRBs

(a) Draw what the following 2-3 tree would look like after inserting 18, 38, 12, 13, and 20.

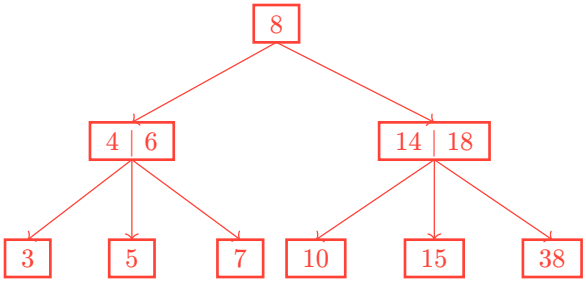


Solution:

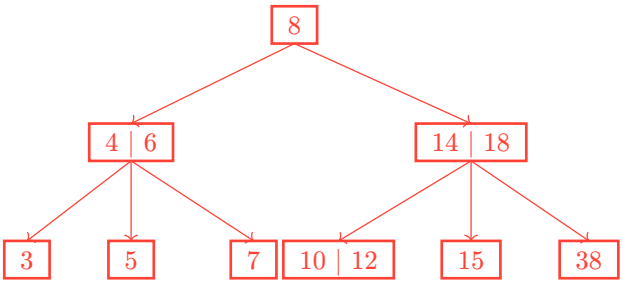
Adding 18:



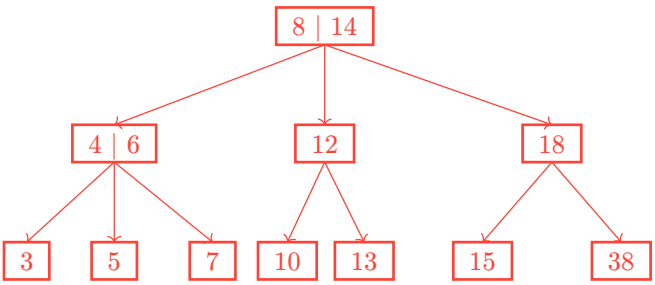
Adding 38:



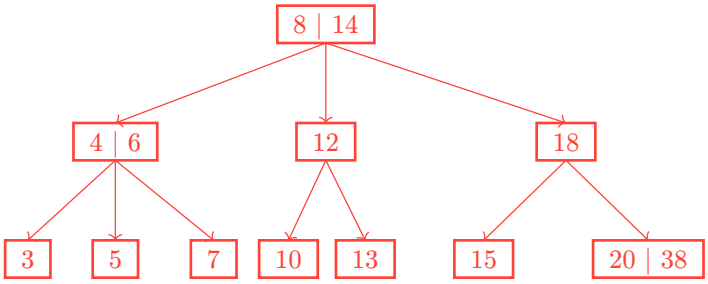
Adding 12:



Adding 13:

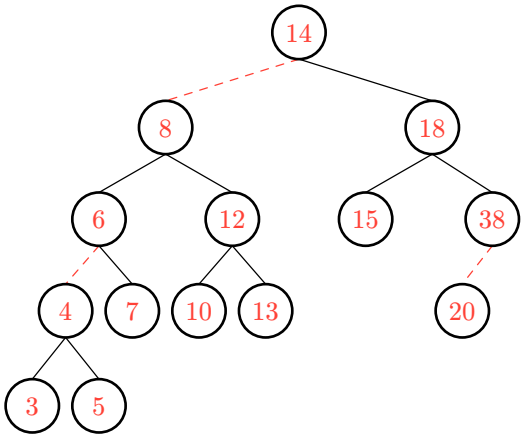


Adding 20:



(b) Now, convert the resulting 2-3 tree to a left-leaning red-black tree.

Solution:

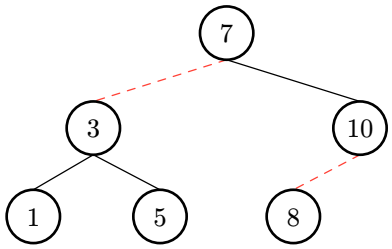


(c) Suppose we create a 2-3 tree by inserting the nodes $V_0, V_1, ..., V_n$, where V_i is strictly smaller than $V_{\{i+1\}}$, in order. That is, we first insert V_0, then V_1, and so on. What is the runtime to find an element in this 2-3 tree in the **worst case**, where N is the number of nodes?

$\Theta(\log N)$

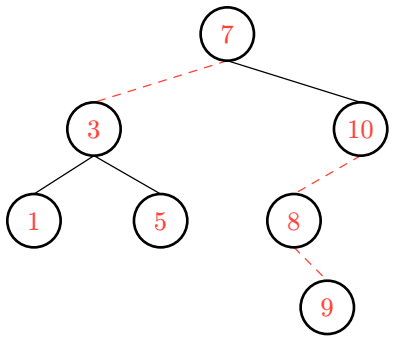
The purpose of a 2-3 tree is to ensure a "bushy" structure to improve the runtime of binary search. The worst-case height of a 2-3 tree is $\log_2(N+1) - 1$, which is a logarithmic function of N. Theta-bounding this, we can simplify to $\Theta(\log N)$.

(d) Now, insert 9 into the LLRB Tree below. Describe where you would insert this node, and what balancing operations (`rotateLeft`, `rotateRight`, `colorSwap`) you'd take to balance the tree after insertion. Assume that in the given LLRB, dotted links between nodes are red and solid links between nodes are black.

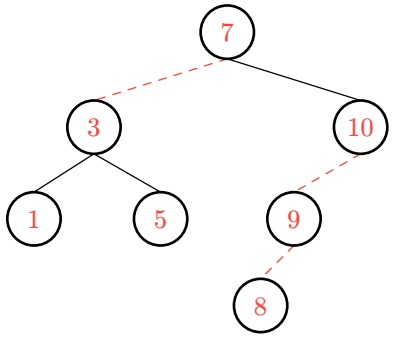


Solution:

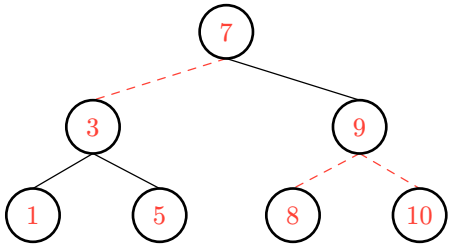
Remember that we always insert elements as leaf nodes with red links, so after initial insertion, the tree looks like:



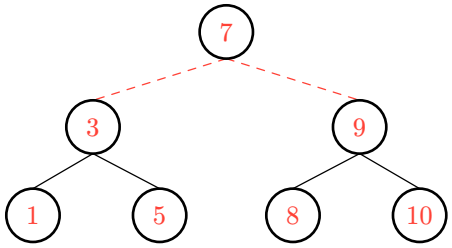
Now there is a right-leaning red link, so we will have to `rotateLeft(8)`:



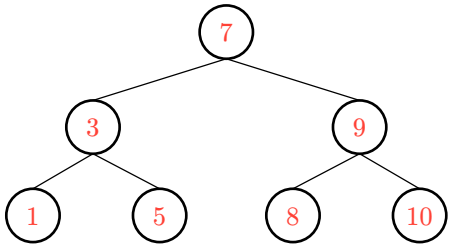
With the previous rotation, we now have two consecutive red left links, which makes our tree unbalanced. To fix, we `rotateRight(10)`:



Next, we have red links on both the left and the right, which can be fixed with a `colorFlip(9)`:



Finally, we still have red links on both the left and the right, which can be fixed with a `colorFlip(7)`:



(e) If a 2-3 tree has depth `H` (that is, the leaves are at distance `H` from the root), what is the maximum number of comparisons done in the corresponding red-black tree to find whether a certain key is present in the tree?

Solution:

$2H + 2$ comparisons.

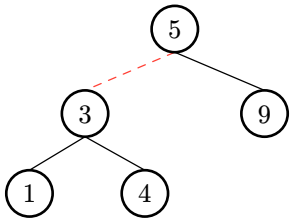
The maximum number of comparisons occur from a root to leaf path with the most nodes. Because the height of the tree is H, we know that there is a path down the leaf-leaning red-black tree that consists of at most H black links, for black links in the left-leaning red-black tree are the links that add to the height of the corresponding 2-3 tree. This means that there are $H + 1$ nodes on the path from the root to the leaf, since there is one less link than nodes.

In the worst case, in the 2-3 tree representation, this path can consist entirely of nodes with two items, meaning in the left-leaning red-black tree representation, each black link is followed by a red link. This doubles the amount of nodes on this path from the root to the leaf.

This example will represent our longest path, which is $2H + 2$ nodes long, meaning we make at most $2H + 2$ comparisons in the left-leaning red-black tree.

3 LLRB Insertions

Given the LLRB below, perform the following insertions and draw the final state of the LLRB. In addition, for each insertion, write the balancing operations needed in the correct order (**`rotateRight`**, **`rotateLeft`**, or **`colorFlip`**). If no balancing operations are needed, write "Nothing". Assume that the link between 5 and 3 is red and all other links are black at the start.



(a) 1. Insert 7

2. Insert 6

3. Insert 2

4. Insert 8

5. Insert 8.5

6. Final state

Solution: For a visualization of the process, see here

1. Insert 7:
   - Nothing

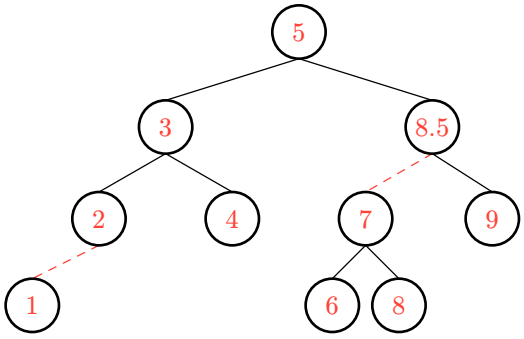
2. Insert 6:
   - rotateRight(9)
   - colorFlip(7)
   - colorFlip(5)

3. Insert 2:
   - rotateLeft(1)

4. Insert 8:
   - Nothing

5. Insert 8.5:
   - rotateLeft(8)
   - rotateRight(9)
   - colorFlip(8.5)
   - rotateLeft(7)

6. Final state



(b) Convert the final LLRB to its corresponding 2-3 Tree.

Solution: