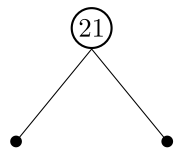
CS320 Homework 3

Dustin Randall

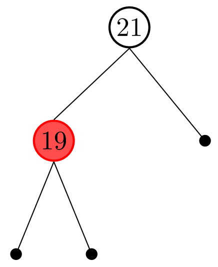
October 30, 2025

1 Construct RBT with the following keys: 21, 19, 17, 12, 15, 9

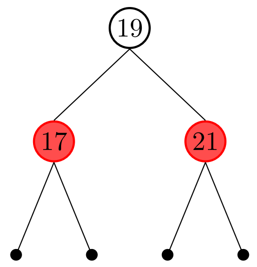
Insert 21



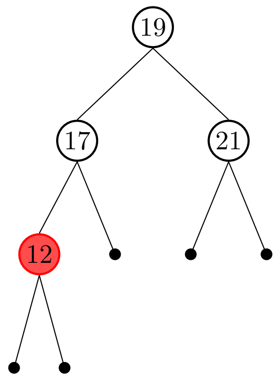
Insert 19



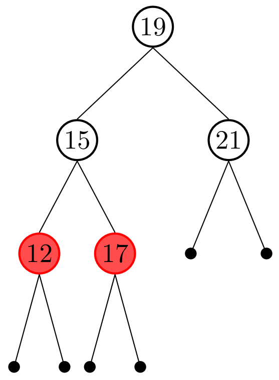
Insert 17



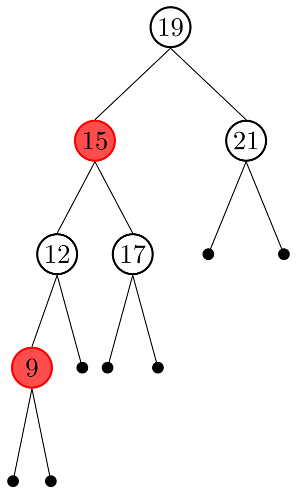
Insert 12



Insert 15

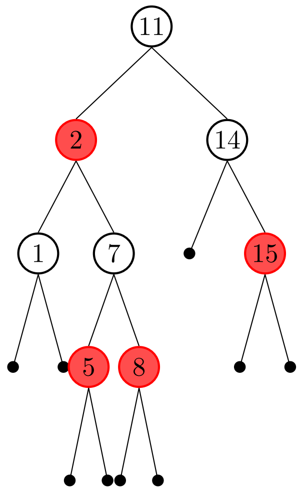


Insert 9

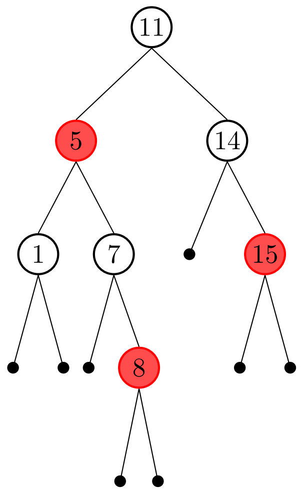


2 Delete the nodes of the RBT in the following order: 2, 5, 1, 14, 11, 15, 7, 8

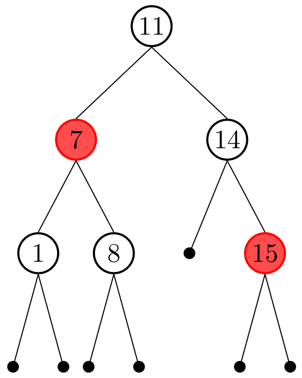
Initial Tree



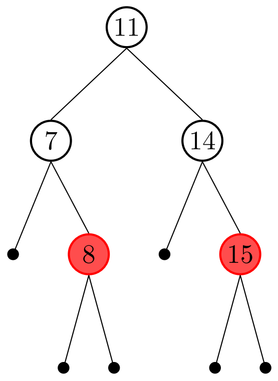
Delete 2



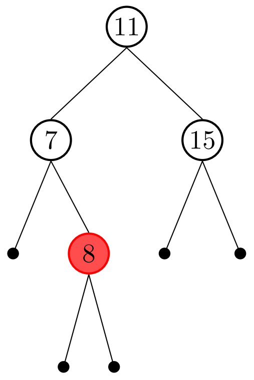
Delete 5



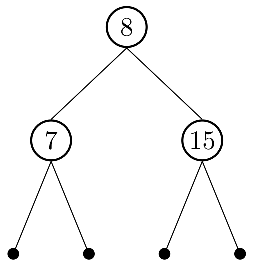
Delete 1



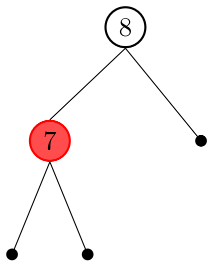
Delete 14



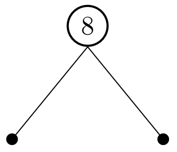
Delete 11



Delete 15



Delete 7



Delete 8

3 Trace the one-pass construction of a B-Tree with t=2 with the following nodes: S, G, W, H, O, U, M, A, C, X, P

Insert S

$\boxed{\text{S}}$

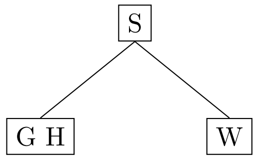
Insert G

$\boxed{\text{G S}}$

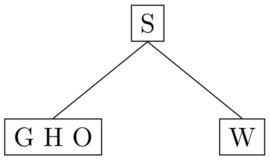
Insert W

$\boxed{\text{G S W}}$

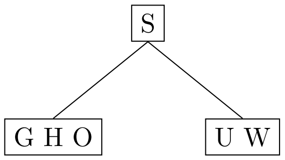
Insert H



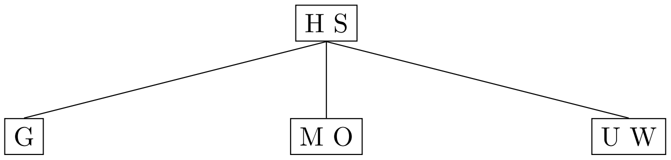
Insert O



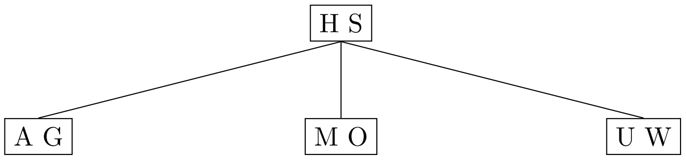
Insert U



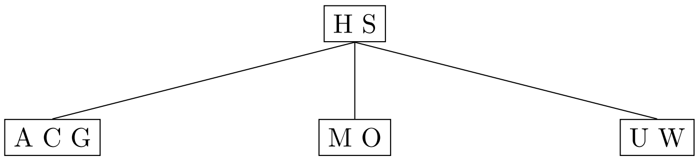
Insert M



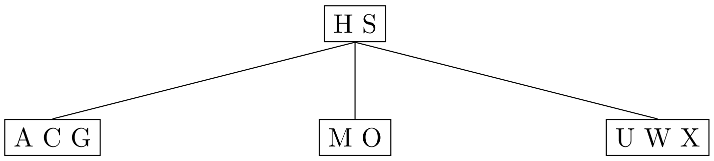
Insert A



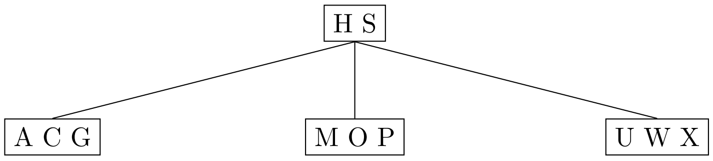
Insert C



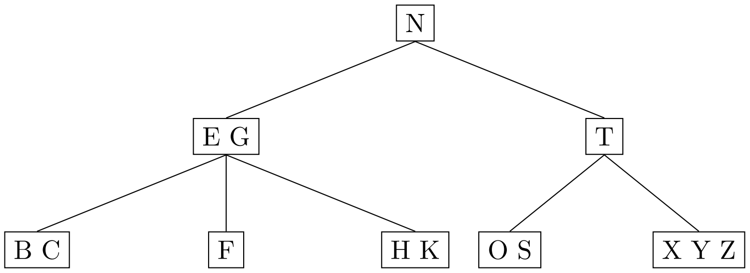
Insert X



Insert P

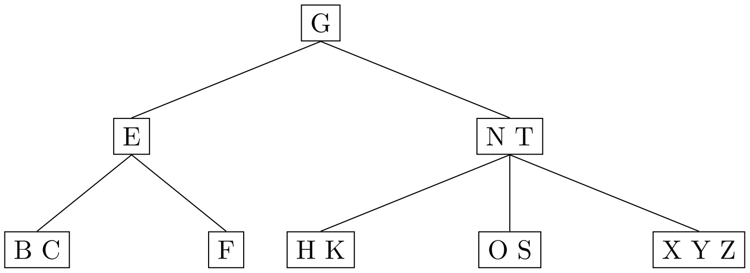


4 Delete the following nodes in order from the initial B-Tree: T, G, F, O

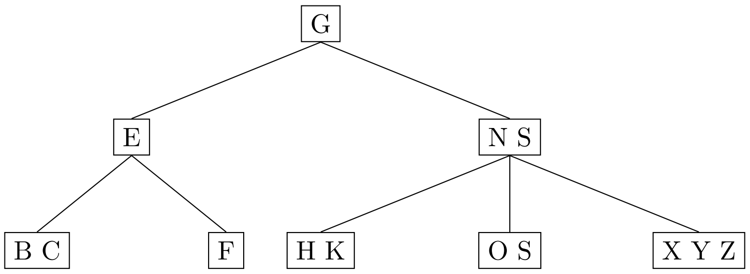


Delete T

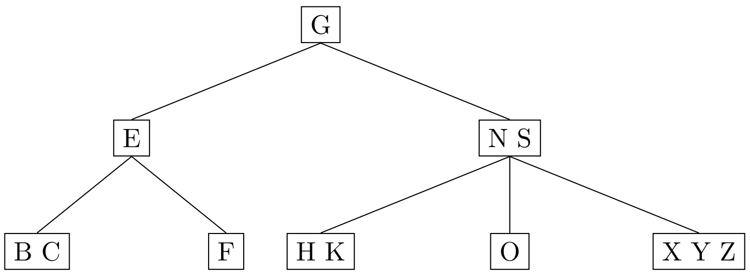
Borrow G from Left Child



Swap Predecessor S with T

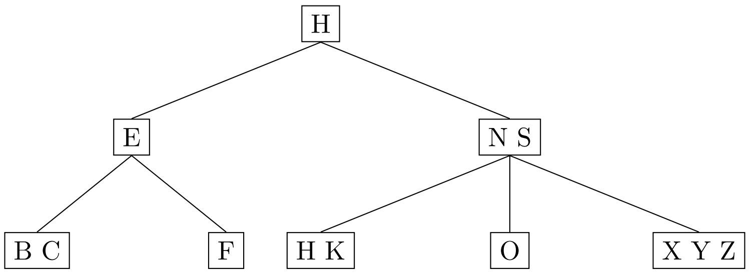


Delete S from Leaf

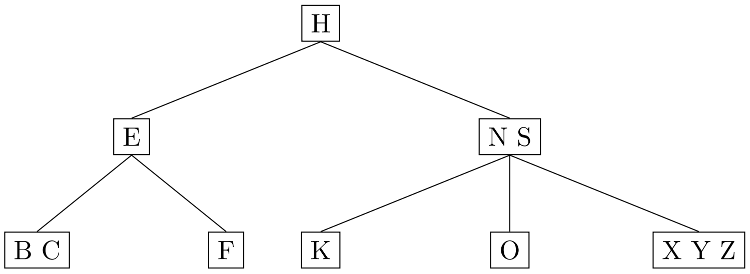


Delete G

Swap G with Successor H

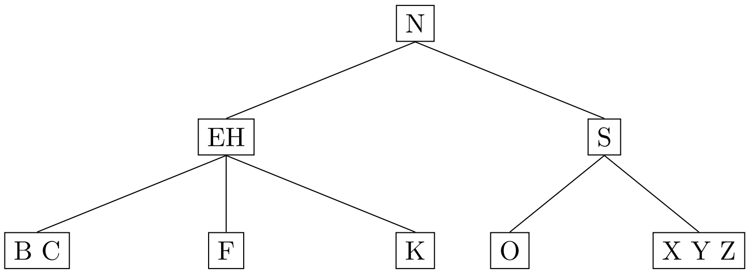


Delete H from Leaf

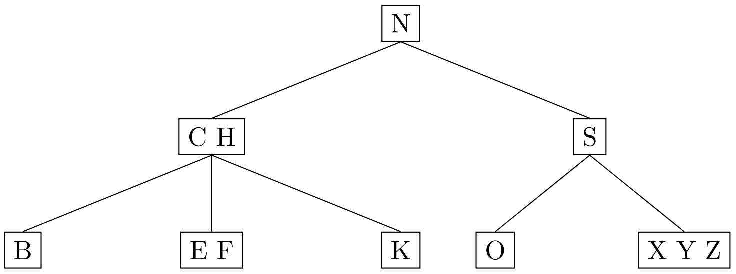


Delete F

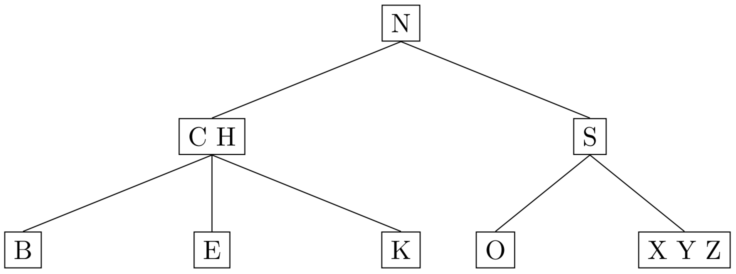
4.0.1 Borrow N from Right Sibling



4.0.2 Borrow C from Left Sibling

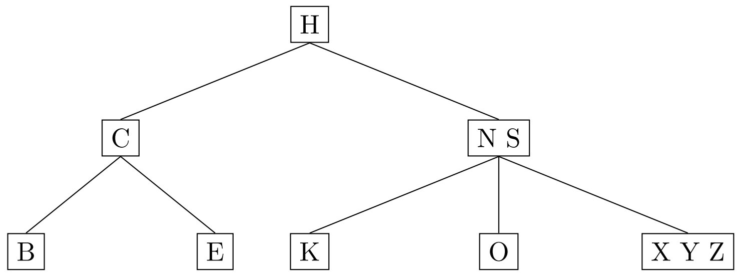


4.0.3 Remove F from Leaf

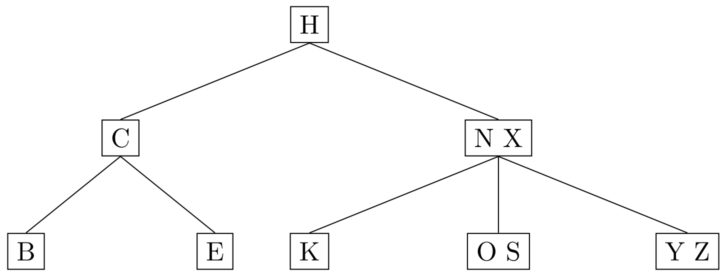


4.1 Delete O

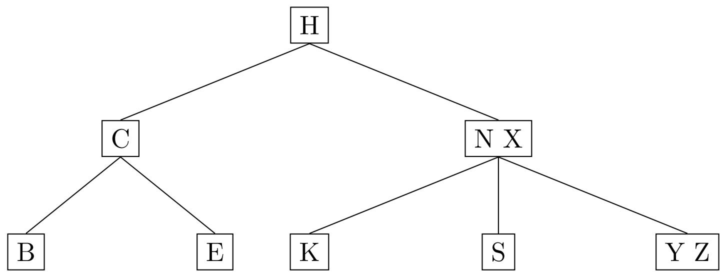
4.1.1 Borrow H from Left Sibling



4.1.2 Borrow X from Right Sibling



4.1.3 Delete O From Leaf



5 Explain BTreeMin and BTreeSuccessor

BTreeMin

Given a node, traverse down the left-most child until there are no more children. Once there, return the index of the smallest element in that node, along with a pointer to the node.

BTreeSuccessor

If I has a right child, the successor is the BTreeMin of the right child. Otherwise, while we're the right most child of our parent, get the next parent. Finally after finding a non-right-most parent, return the key after our node. If we run out of parents, return null.

Pseudocode

```
(Node, Index) BTreeMin(root) {
    while(root.Children.Count > 0) {
        root = root.Children[0]; // DISK_READ
    }
    // assuming keys are in order
    return (root, 0);
    // alternatively, return (root, min(root.Keys))
}

(successor_node, j) BTreeSuccessor(node, i) {
    if(node.Children.Count > i) {
        return BTreeMin(node.Children[i + 1]);
    }
    //in a leaf node, and successor is in this node
    if(node.IsLeaf && i + 1 < node.Keys.Count) {
        return (node, i + 1);
    } else {
        Node parent = node.Parent; // DISK_READ
        pIndex = parent.FindIndex(node);
        while(pIndex == parent.Keys.Count) {
            node = parent;
            parent = node.Parent; // DISK_READ
            if(parent == null) return null;
            pIndex = parent.FindIndex(node);
        }
        return (parent, pIndex);
    }
}
```