Assignment 10: 2-3 Trees, R-B Trees
OCS25 Data Structures in Java Dr. Made
Vishak Srikanth
November 22 2021

2- 3 Trees

Note: I used a simplifying assumption that the letters are directly stored as keys and values in the 2-3 node so that when letters are repeated we do not get duplicated tree nodes with the same letter as key. Therefore, when we encounter the same repeated character, the existing node is just updated instead of creating a new node. This is consistent with the logic shown in Section 3.3 of Algorithms 4th Edition by Sedgewick et. al. I've submitted my Java implementation separately.

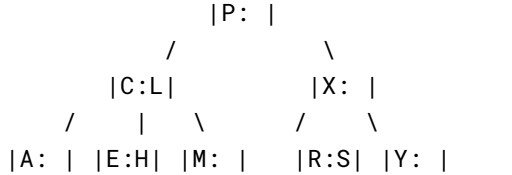
Problem 1: 2-3 trees that result when the keys YLPMXHCRAESTBCA in that order into an initially empty 2-3 tree.

In the figures below we indicate each node as |x:y| where x, y are the keys in the 2-3 node. If it is a 2-node we leave the right and side blank for example |x: |

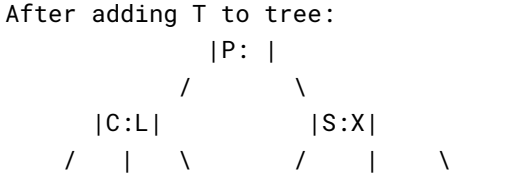
```
After adding Y to tree:
     |Y: |

After adding L to tree:
     |L:Y|

After adding P to tree:
    |P: |
   /    \
|L: |   |Y: |

After adding M to tree:
    |P: |
   /    \
|L:M|   |Y: |

After adding X to tree:
    |P: |
```

```
   /    \
|L:M| |X:Y|


After adding H to tree:
     |L:P|
   /   |   \
|H: ||M: ||X:Y|

After adding C to tree:
     |L:P|
   /   |   \
|C:H| |M: | |X:Y|

After adding R to tree:
        |P: |
      /      \
   |L: |      |X: |
  /    \     /   \
|C:H| |M: | |R: | |Y: |

After adding A to tree:
            |P: |
          /       \
      |C:L|        |X: |
   /    |    \      /   \
|A: | |H: | |M: | |R: | |Y: |

After adding E to tree:
             |P: |
           /        \
     |C:L|           |X: |
    /  |    \        /    \
|A: | |E:H| |M: |   |R: | |Y: |
```

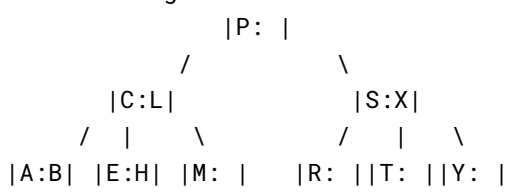
After adding S to tree:



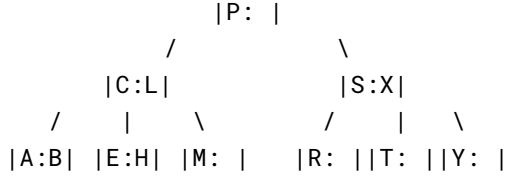
After adding T to tree:



After adding B to tree:



After adding C to tree:



```
After adding A to tree:
               |P: |
           /           \
       |C:L|           |S:X|
   /    |    \       /    |    \
|A:B| |E:H| |M: | |R: | |T: | |Y: |
```

Problem 2: 2-3 trees that result when the keys ALGORITHMSXYZ in that order into an initially empty tree. Red-Black Trees

```
After adding A to tree:
       |A: |

After adding L to tree:
       |A:L|

After adding G to tree:
          |G: |
          /    \
       |A: | |L: |

After adding O to tree:
          |G: |
         /    \
       |A: | |L:O|
```

```
After adding R to tree:
            |G:O|
          /   |   \
       |A: | |L: | |R: |

After adding I to tree:
            |G:O|
          /   |   \
       |A: | |I:L| |R: |

After adding T to tree:
            |G:O|
          /   |   \
       |A: ||I:L||R:T|

After adding H to tree:
           |I: |
           /     \
        |G: |      |O: |
       /    \      /    \
    |A: |  |H: | |L: | |R:T|

After adding M to tree:
                |I: |
              /     \
           |G: |       |O: |
          /    \      /    \
       |A: | |H: |  |L:M| |R:T|
```

```
After adding S to tree:
                 |I: |
              /        \
           |G: |        |O:S|
         /    \      /    |   \
       |A: | |H: |  |L:M| |R: | |T: |

After adding X to tree:
                   |I: |
               /           \
           |G: |           |O:S|
         /    \         /   |   \
       |A: | |H: |   |L:M| |R: | |T:X|

After adding Y to tree:
                    |I:S|
              /       |       \
           |G: |     |O: |     |X: |
         /    \    /     \    /    \
       |A: | |H: ||L:M| |R: ||T: | |Y: |

After adding Z to tree:
                      |I:S|
             /          |         \
          |G: |       |O: |      |X: |
          /    \     /    \      /    \
       |A: | |H: | |L:M| |R: | |T: | |Y:Z|
```

Left leaning red-black (LLRB) trees

Note: I used a simplifying assumption that the letters are directly stored as keys in the RB BST tree nodes so that when letters are repeated we do not get duplicated tree nodes with the same letter as key. Therefore, when we encounter the same repeated character, the existing node is just updated instead of creating a new node. This is consistent with the logic shown in Section 3.3 of Algorithms 4th Edition by Sedgewick et. al. I've submitted my Java implementation separately.

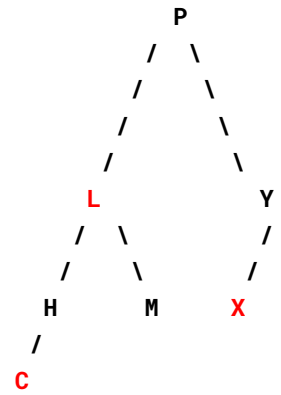
Problem 3: Red-black trees that result when you insert the keys Y L P M X H C R A E S T B C A in that order into an initially empty tree.

```
After adding Y to tree:
             Y


Inorder Traversal of Tree:
NodeType: BLACK, Key: Y, LeftNode: null, RightNode: null, Parent:null
********************************************************************************

********************************************************************************
After adding L to tree:
           Y
          /
         L


Inorder Traversal of Tree:
NodeType: RED, Key: L, LeftNode: null, RightNode: null, Parent:Y
NodeType: BLACK, Key: Y, LeftNode: L, RightNode: null, Parent:null
********************************************************************************
```

```
********************************************************************************
After adding P to tree:
           P
         /  \
        L    Y


Inorder Traversal of Tree:
NodeType: BLACK, Key: L, LeftNode: null, RightNode: null, Parent:P
NodeType: BLACK, Key: P, LeftNode: L, RightNode: Y, Parent:null
NodeType: BLACK, Key: Y, LeftNode: null, RightNode: null, Parent:P
********************************************************************************
********************************************************************************
After adding M to tree:
             P
            /  \
           /    \
          M      Y
         /
        L


Inorder Traversal of Tree:
NodeType: RED, Key: L, LeftNode: null, RightNode: null, Parent:M
NodeType: BLACK, Key: M, LeftNode: L, RightNode: null, Parent:P
NodeType: BLACK, Key: P, LeftNode: M, RightNode: Y, Parent:null
NodeType: BLACK, Key: Y, LeftNode: null, RightNode: null, Parent:P
********************************************************************************
```

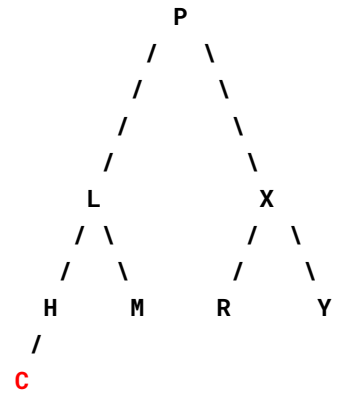
```
********************************************************************************
After adding X to tree:
             P
            /  \
           /    \
          M      Y
         /      /
        L      X


Inorder Traversal of Tree:
NodeType: RED, Key: L, LeftNode: null, RightNode: null, Parent:M
NodeType: BLACK, Key: M, LeftNode: L, RightNode: null, Parent:P
NodeType: BLACK, Key: P, LeftNode: M, RightNode: Y, Parent:null
NodeType: RED, Key: X, LeftNode: null, RightNode: null, Parent:Y
NodeType: BLACK, Key: Y, LeftNode: X, RightNode: null, Parent:P
********************************************************************************
********************************************************************************
After adding H to tree:
             P
            /  \
           /    \
          L      Y
         /  \   /
        H   M  X


Inorder Traversal of Tree:
NodeType: BLACK, Key: H, LeftNode: null, RightNode: null, Parent:L
NodeType: RED, Key: L, LeftNode: H, RightNode: M, Parent:P
NodeType: BLACK, Key: M, LeftNode: null, RightNode: null, Parent:L
```

```
NodeType: BLACK, Key: P, LeftNode: L, RightNode: Y, Parent:null
NodeType: RED, Key: X, LeftNode: null, RightNode: null, Parent:Y
NodeType: BLACK, Key: Y, LeftNode: X, RightNode: null, Parent:P
********************************************************************************
********************************************************************************
After adding C to tree:
```



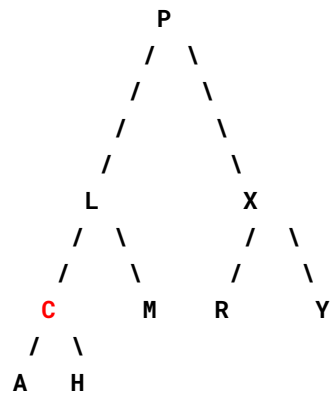

```
Inorder Traversal of Tree:
NodeType: RED, Key: C, LeftNode: null, RightNode: null, Parent:H
NodeType: BLACK, Key: H, LeftNode: C, RightNode: null, Parent:L
NodeType: RED, Key: L, LeftNode: H, RightNode: M, Parent:P
NodeType: BLACK, Key: M, LeftNode: null, RightNode: null, Parent:L
NodeType: BLACK, Key: P, LeftNode: L, RightNode: Y, Parent:null
NodeType: RED, Key: X, LeftNode: null, RightNode: null, Parent:Y
NodeType: BLACK, Key: Y, LeftNode: X, RightNode: null, Parent:P
********************************************************************************
```

```
********************************************************************************************
After adding R to tree:
```



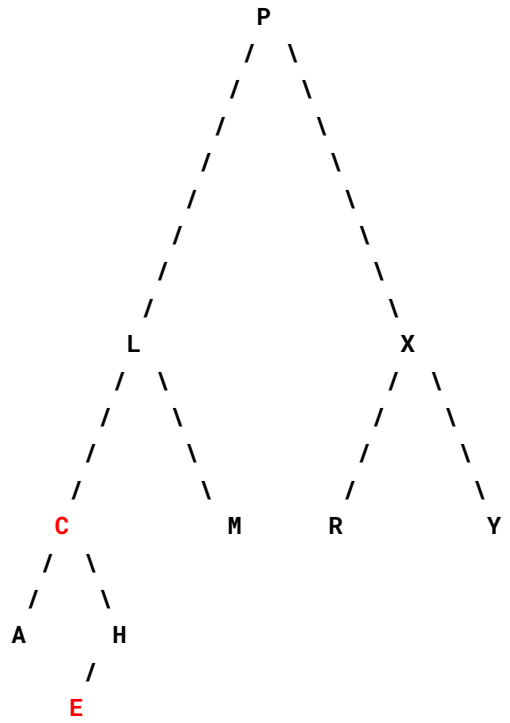

```
Inorder Traversal of Tree:
NodeType: RED, Key: C, LeftNode: null, RightNode: null, Parent:H
NodeType: BLACK, Key: H, LeftNode: C, RightNode: null, Parent:L
NodeType: BLACK, Key: L, LeftNode: H, RightNode: M, Parent:P
NodeType: BLACK, Key: M, LeftNode: null, RightNode: null, Parent:L
NodeType: BLACK, Key: P, LeftNode: L, RightNode: X, Parent:null
NodeType: BLACK, Key: R, LeftNode: null, RightNode: null, Parent:X
NodeType: BLACK, Key: X, LeftNode: R, RightNode: Y, Parent:P
NodeType: BLACK, Key: Y, LeftNode: null, RightNode: null, Parent:X
********************************************************************************************
```

```
********************************************************************************************
After adding A to tree:
```



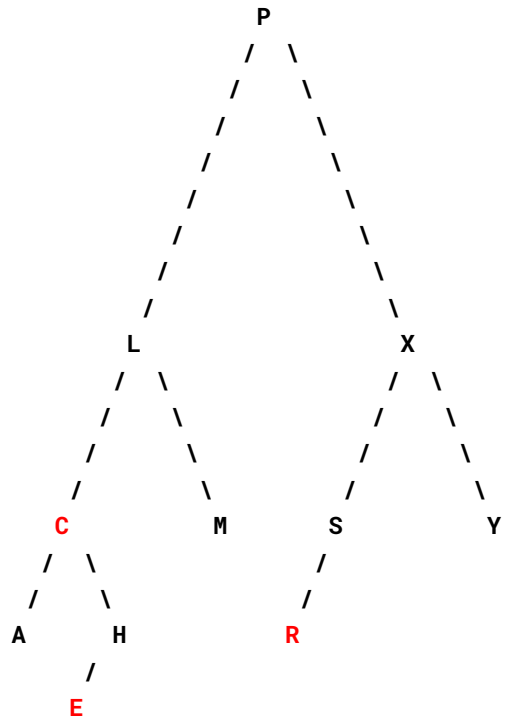

```
Inorder Traversal of Tree:
NodeType: BLACK, Key: A, LeftNode: null, RightNode: null, Parent:C
NodeType: RED, Key: C, LeftNode: A, RightNode: H, Parent:L
NodeType: BLACK, Key: H, LeftNode: null, RightNode: null, Parent:C
NodeType: BLACK, Key: L, LeftNode: C, RightNode: M, Parent:P
NodeType: BLACK, Key: M, LeftNode: null, RightNode: null, Parent:L
NodeType: BLACK, Key: P, LeftNode: L, RightNode: X, Parent:null
NodeType: BLACK, Key: R, LeftNode: null, RightNode: null, Parent:X
NodeType: BLACK, Key: X, LeftNode: R, RightNode: Y, Parent:P
NodeType: BLACK, Key: Y, LeftNode: null, RightNode: null, Parent:X
********************************************************************************************
```

```
********************************************************************************************
After adding E to tree:
```



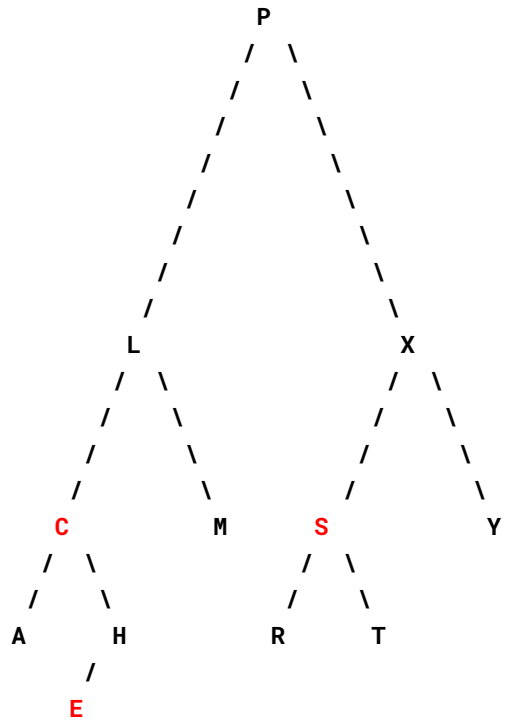

```
Inorder Traversal of Tree:
NodeType: BLACK, Key: A, LeftNode: null, RightNode: null, Parent:C
NodeType: RED, Key: C, LeftNode: A, RightNode: H, Parent:L
NodeType: RED, Key: E, LeftNode: null, RightNode: null, Parent:H
NodeType: BLACK, Key: H, LeftNode: E, RightNode: null, Parent:C
NodeType: BLACK, Key: L, LeftNode: C, RightNode: M, Parent:P
NodeType: BLACK, Key: M, LeftNode: null, RightNode: null, Parent:L
NodeType: BLACK, Key: P, LeftNode: L, RightNode: X, Parent:null
```

```
NodeType: BLACK, Key: R, LeftNode: null, RightNode: null, Parent:X
NodeType: BLACK, Key: X, LeftNode: R, RightNode: Y, Parent:P
NodeType: BLACK, Key: Y, LeftNode: null, RightNode: null, Parent:X
********************************************************************************
********************************************************************************
After adding S to tree:
```



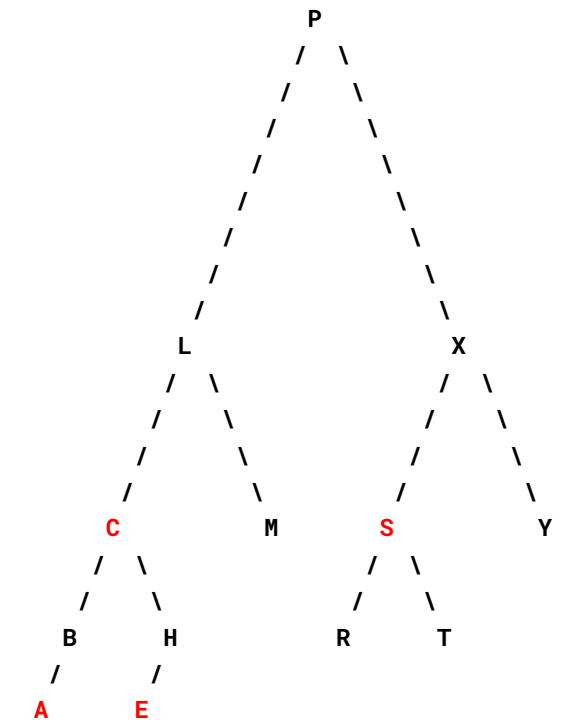

```
Inorder Traversal of Tree:
NodeType: BLACK, Key: A, LeftNode: null, RightNode: null, Parent:C
NodeType: RED, Key: C, LeftNode: A, RightNode: H, Parent:L
NodeType: RED, Key: E, LeftNode: null, RightNode: null, Parent:H
NodeType: BLACK, Key: H, LeftNode: E, RightNode: null, Parent:C
NodeType: BLACK, Key: L, LeftNode: C, RightNode: M, Parent:P
NodeType: BLACK, Key: M, LeftNode: null, RightNode: null, Parent:L
NodeType: BLACK, Key: P, LeftNode: L, RightNode: X, Parent:null
```

```
NodeType: RED, Key: R, LeftNode: null, RightNode: null, Parent:S
NodeType: BLACK, Key: S, LeftNode: R, RightNode: null, Parent:X
NodeType: BLACK, Key: X, LeftNode: S, RightNode: Y, Parent:P
NodeType: BLACK, Key: Y, LeftNode: null, RightNode: null, Parent:X
********************************************************************************
********************************************************************************
After adding T to tree:
```




```
Inorder Traversal of Tree:
NodeType: BLACK, Key: A, LeftNode: null, RightNode: null, Parent:C
NodeType: RED, Key: C, LeftNode: A, RightNode: H, Parent:L
NodeType: RED, Key: E, LeftNode: null, RightNode: null, Parent:H
NodeType: BLACK, Key: H, LeftNode: E, RightNode: null, Parent:C
NodeType: BLACK, Key: L, LeftNode: C, RightNode: M, Parent:P
NodeType: BLACK, Key: M, LeftNode: null, RightNode: null, Parent:L
```

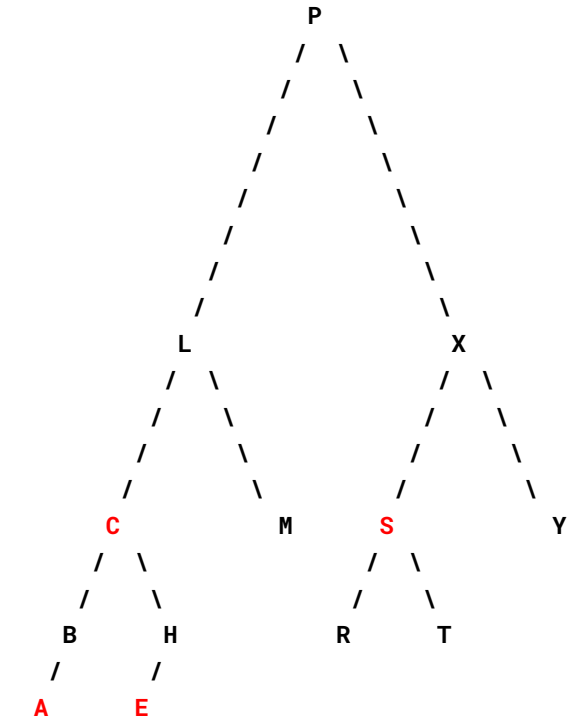
```
NodeType: BLACK, Key: P, LeftNode: L, RightNode: X, Parent:null
NodeType: BLACK, Key: R, LeftNode: null, RightNode: null, Parent:S
NodeType: RED, Key: S, LeftNode: R, RightNode: T, Parent:X
NodeType: BLACK, Key: T, LeftNode: null, RightNode: null, Parent:S
NodeType: BLACK, Key: X, LeftNode: S, RightNode: Y, Parent:P
NodeType: BLACK, Key: Y, LeftNode: null, RightNode: null, Parent:X
********************************************************************************
********************************************************************************
After adding B to tree:
```




```
Inorder Traversal of Tree:
NodeType: RED, Key: A, LeftNode: null, RightNode: null, Parent:B
NodeType: BLACK, Key: B, LeftNode: A, RightNode: null, Parent:C
```

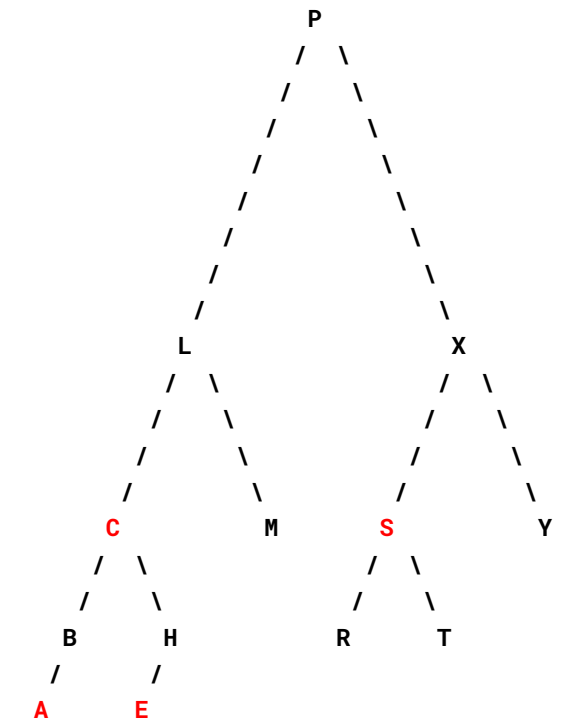
```
NodeType: RED, Key: C, LeftNode: B, RightNode: H, Parent:L
NodeType: RED, Key: E, LeftNode: null, RightNode: null, Parent:H
NodeType: BLACK, Key: H, LeftNode: E, RightNode: null, Parent:C
NodeType: BLACK, Key: L, LeftNode: C, RightNode: M, Parent:P
NodeType: BLACK, Key: M, LeftNode: null, RightNode: null, Parent:L
NodeType: BLACK, Key: P, LeftNode: L, RightNode: X, Parent:null
NodeType: BLACK, Key: R, LeftNode: null, RightNode: null, Parent:S
NodeType: RED, Key: S, LeftNode: R, RightNode: T, Parent:X
NodeType: BLACK, Key: T, LeftNode: null, RightNode: null, Parent:S
NodeType: BLACK, Key: X, LeftNode: S, RightNode: Y, Parent:P
NodeType: BLACK, Key: Y, LeftNode: null, RightNode: null, Parent:X
********************************************************************************
```

```
********************************************************************************************
After adding C to tree:
```



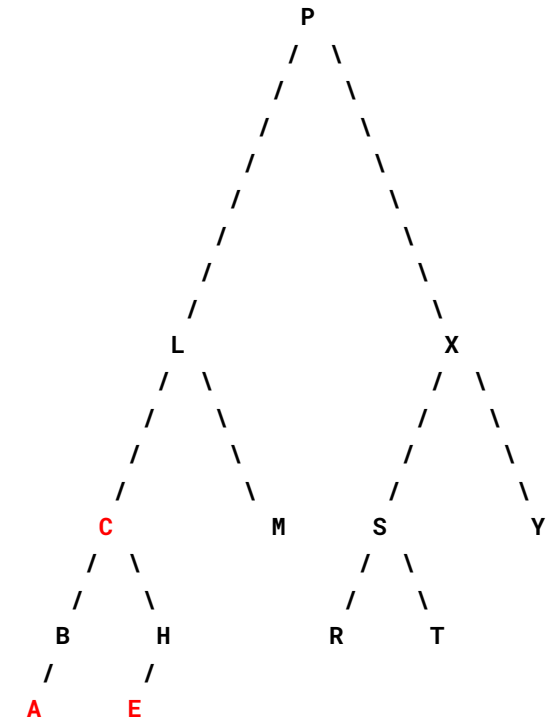

```
Inorder Traversal of Tree:
NodeType: RED, Key: A, LeftNode: null, RightNode: null, Parent:B
NodeType: BLACK, Key: B, LeftNode: A, RightNode: null, Parent:C
NodeType: RED, Key: C, LeftNode: B, RightNode: H, Parent:L
NodeType: RED, Key: E, LeftNode: null, RightNode: null, Parent:H
NodeType: BLACK, Key: H, LeftNode: E, RightNode: null, Parent:C
NodeType: BLACK, Key: L, LeftNode: C, RightNode: M, Parent:P
NodeType: BLACK, Key: M, LeftNode: null, RightNode: null, Parent:L
```

```
NodeType: BLACK, Key: P, LeftNode: L, RightNode: X, Parent:null
NodeType: BLACK, Key: R, LeftNode: null, RightNode: null, Parent:S
NodeType: RED, Key: S, LeftNode: R, RightNode: T, Parent:X
NodeType: BLACK, Key: T, LeftNode: null, RightNode: null, Parent:S
NodeType: BLACK, Key: X, LeftNode: S, RightNode: Y, Parent:P
NodeType: BLACK, Key: Y, LeftNode: null, RightNode: null, Parent:X
********************************************************************************
********************************************************************************
After adding A to tree:
```




```
Inorder Traversal of Tree:
NodeType: RED, Key: A, LeftNode: null, RightNode: null, Parent:B
NodeType: BLACK, Key: B, LeftNode: A, RightNode: null, Parent:C
NodeType: RED, Key: C, LeftNode: B, RightNode: H, Parent:L
```

```
NodeType: RED, Key: E, LeftNode: null, RightNode: null, Parent:H
NodeType: BLACK, Key: H, LeftNode: E, RightNode: null, Parent:C
NodeType: BLACK, Key: L, LeftNode: C, RightNode: M, Parent:P
NodeType: BLACK, Key: M, LeftNode: null, RightNode: null, Parent:L
NodeType: BLACK, Key: P, LeftNode: L, RightNode: X, Parent:null
NodeType: BLACK, Key: R, LeftNode: null, RightNode: null, Parent:S
NodeType: RED, Key: S, LeftNode: R, RightNode: T, Parent:X
NodeType: BLACK, Key: T, LeftNode: null, RightNode: null, Parent:S
NodeType: BLACK, Key: X, LeftNode: S, RightNode: Y, Parent:P
NodeType: BLACK, Key: Y, LeftNode: null, RightNode: null, Parent:X
********************************************************************************
********************************************************************************
Final tree is:
```

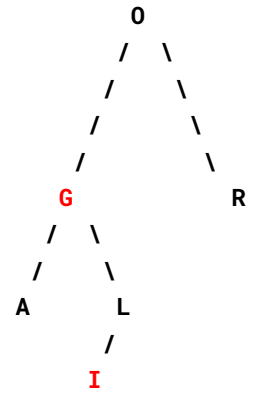


Problem 4: Red-black trees that result when you insert the keys A L G O R I T H M S X Y Z in that order into an initially empty tree.

```
********************************************************************************
After adding A to tree:
             A

Inorder Traversal of Tree:
NodeType: BLACK, Key: A, LeftNode: null, RightNode: null, Parent:null
********************************************************************************
********************************************************************************
After adding L to tree:
                L
               /
              A


Inorder Traversal of Tree:
NodeType: RED, Key: A, LeftNode: null, RightNode: null, Parent:L
NodeType: BLACK, Key: L, LeftNode: A, RightNode: null, Parent:null
********************************************************************************
********************************************************************************
After adding G to tree:
                G
               /  \
              A    L

Inorder Traversal of Tree:
NodeType: BLACK, Key: A, LeftNode: null, RightNode: null, Parent:G
NodeType: BLACK, Key: G, LeftNode: A, RightNode: L, Parent:null
NodeType: BLACK, Key: L, LeftNode: null, RightNode: null, Parent:G
********************************************************************************
```

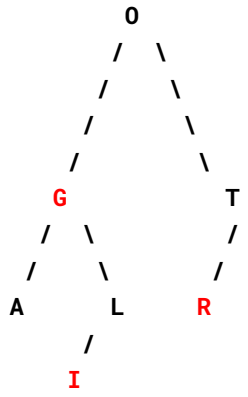
```
********************************************************************************
After adding O to tree:
                    G
                   /  \
                  /    \
                 A      O
                       /
                      L


Inorder Traversal of Tree:
NodeType: BLACK, Key: A, LeftNode: null, RightNode: null, Parent:G
NodeType: BLACK, Key: G, LeftNode: A, RightNode: O, Parent:null
NodeType: RED, Key: L, LeftNode: null, RightNode: null, Parent:O
NodeType: BLACK, Key: O, LeftNode: L, RightNode: null, Parent:G
********************************************************************************
********************************************************************************
After adding R to tree:
                          O
                         /  \
                        /    \
                       G      R
                      /  \
                     A    L


Inorder Traversal of Tree:
NodeType: BLACK, Key: A, LeftNode: null, RightNode: null, Parent:G
NodeType: RED, Key: G, LeftNode: A, RightNode: L, Parent:O
NodeType: BLACK, Key: L, LeftNode: null, RightNode: null, Parent:G
NodeType: BLACK, Key: O, LeftNode: G, RightNode: R, Parent:null
NodeType: BLACK, Key: R, LeftNode: null, RightNode: null, Parent:O
********************************************************************************
```

```
********************************************************************************
After adding I to tree:
```



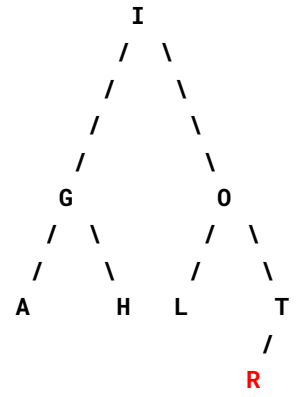

```
Inorder Traversal of Tree:
NodeType: BLACK, Key: A, LeftNode: null, RightNode: null, Parent:G
NodeType: RED, Key: G, LeftNode: A, RightNode: L, Parent:O
NodeType: RED, Key: I, LeftNode: null, RightNode: null, Parent:L
NodeType: BLACK, Key: L, LeftNode: I, RightNode: null, Parent:G
NodeType: BLACK, Key: O, LeftNode: G, RightNode: R, Parent:null
NodeType: BLACK, Key: R, LeftNode: null, RightNode: null, Parent:O
********************************************************************************
```

```
********************************************************************************
After adding T to tree:
```



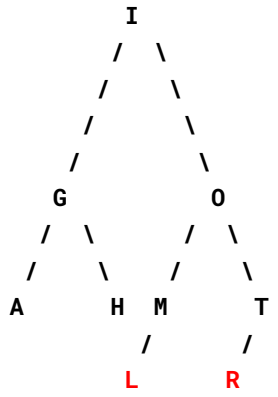

```
Inorder Traversal of Tree:
NodeType: BLACK, Key: A, LeftNode: null, RightNode: null, Parent:G
NodeType: RED, Key: G, LeftNode: A, RightNode: L, Parent:O
NodeType: RED, Key: I, LeftNode: null, RightNode: null, Parent:L
NodeType: BLACK, Key: L, LeftNode: I, RightNode: null, Parent:G
NodeType: BLACK, Key: O, LeftNode: G, RightNode: T, Parent:null
NodeType: RED, Key: R, LeftNode: null, RightNode: null, Parent:T
NodeType: BLACK, Key: T, LeftNode: R, RightNode: null, Parent:O
********************************************************************************
```

```
********************************************************************************************
After adding H to tree:
```



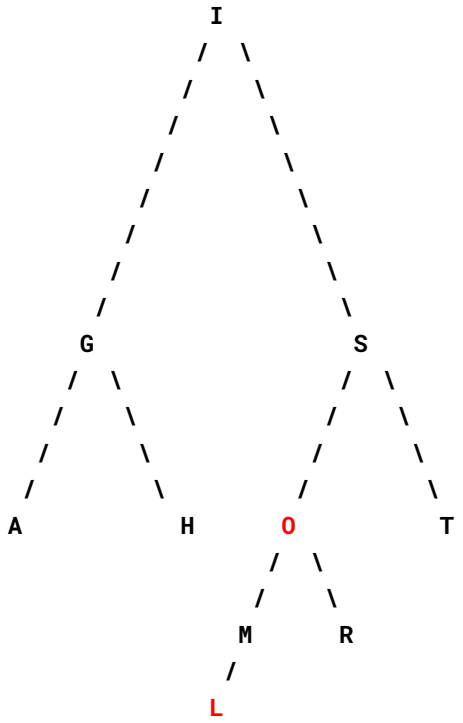

```
Inorder Traversal of Tree:
NodeType: BLACK, Key: A, LeftNode: null, RightNode: null, Parent:G
NodeType: BLACK, Key: G, LeftNode: A, RightNode: H, Parent:I
NodeType: BLACK, Key: H, LeftNode: null, RightNode: null, Parent:G
NodeType: BLACK, Key: I, LeftNode: G, RightNode: O, Parent:null
NodeType: BLACK, Key: L, LeftNode: null, RightNode: null, Parent:O
NodeType: BLACK, Key: O, LeftNode: L, RightNode: T, Parent:I
NodeType: RED, Key: R, LeftNode: null, RightNode: null, Parent:T
NodeType: BLACK, Key: T, LeftNode: R, RightNode: null, Parent:O
********************************************************************************************
```

```
********************************************************************************
After adding M to tree:
```



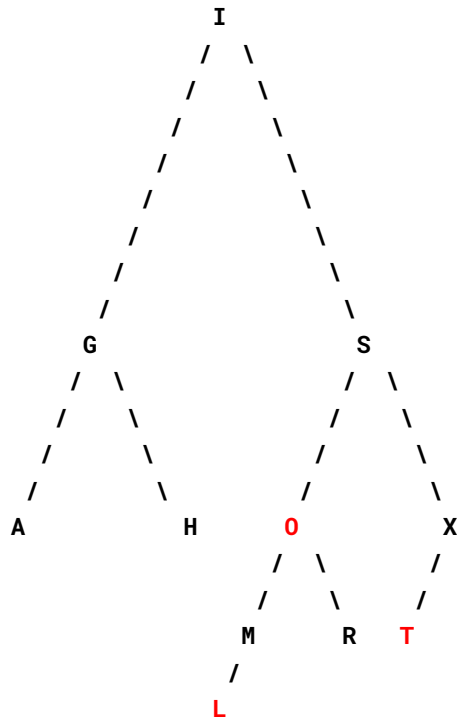

```
Inorder Traversal of Tree:
NodeType: BLACK, Key: A, LeftNode: null, RightNode: null, Parent:G
NodeType: BLACK, Key: G, LeftNode: A, RightNode: H, Parent:I
NodeType: BLACK, Key: H, LeftNode: null, RightNode: null, Parent:G
NodeType: BLACK, Key: I, LeftNode: G, RightNode: O, Parent:null
NodeType: RED, Key: L, LeftNode: null, RightNode: null, Parent:M
NodeType: BLACK, Key: M, LeftNode: L, RightNode: null, Parent:O
NodeType: BLACK, Key: O, LeftNode: M, RightNode: T, Parent:I
NodeType: RED, Key: R, LeftNode: null, RightNode: null, Parent:T
NodeType: BLACK, Key: T, LeftNode: R, RightNode: null, Parent:O
********************************************************************************
```

```
********************************************************************************
After adding S to tree:
```



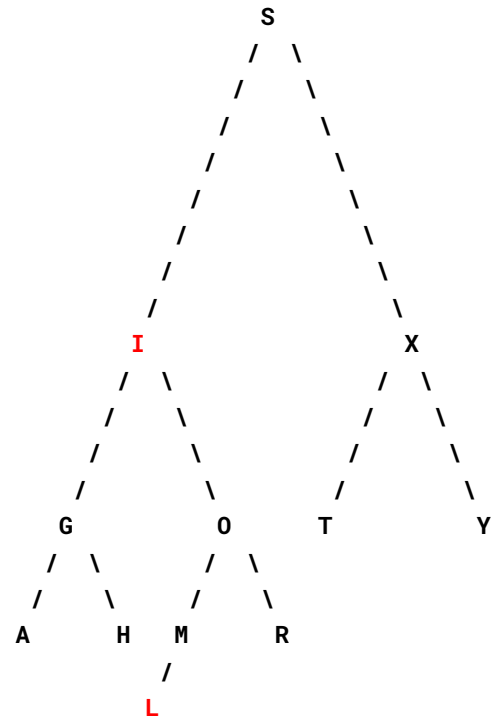

```
Inorder Traversal of Tree:
NodeType: BLACK, Key: A, LeftNode: null, RightNode: null, Parent:G
NodeType: BLACK, Key: G, LeftNode: A, RightNode: H, Parent:I
NodeType: BLACK, Key: H, LeftNode: null, RightNode: null, Parent:G
NodeType: BLACK, Key: I, LeftNode: G, RightNode: S, Parent:null
NodeType: RED, Key: L, LeftNode: null, RightNode: null, Parent:M
NodeType: BLACK, Key: M, LeftNode: L, RightNode: null, Parent:O
NodeType: RED, Key: O, LeftNode: M, RightNode: R, Parent:S
NodeType: BLACK, Key: R, LeftNode: null, RightNode: null, Parent:O
NodeType: BLACK, Key: S, LeftNode: O, RightNode: T, Parent:I
NodeType: BLACK, Key: T, LeftNode: null, RightNode: null, Parent:S
```

****
****
After adding X to tree:



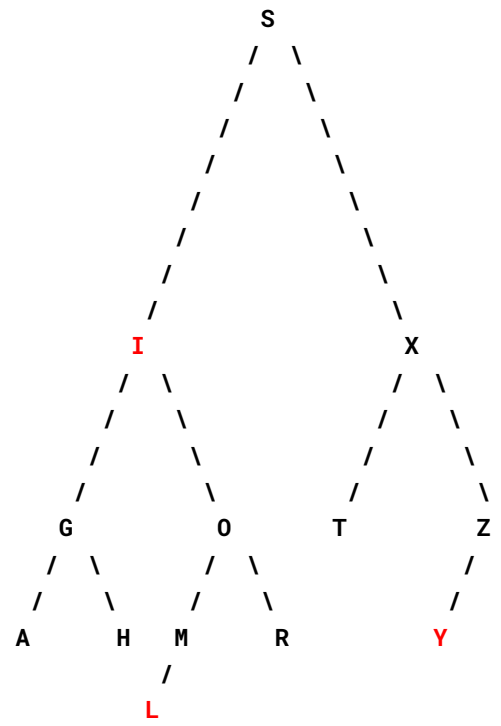
Inorder Traversal of Tree:
NodeType: BLACK, Key: A, LeftNode: null, RightNode: null, Parent:G
NodeType: BLACK, Key: G, LeftNode: A, RightNode: H, Parent:I
NodeType: BLACK, Key: H, LeftNode: null, RightNode: null, Parent:G
NodeType: BLACK, Key: I, LeftNode: G, RightNode: S, Parent:null
NodeType: RED, Key: L, LeftNode: null, RightNode: null, Parent:M
NodeType: BLACK, Key: M, LeftNode: L, RightNode: null, Parent:O
NodeType: RED, Key: O, LeftNode: M, RightNode: R, Parent:S
NodeType: BLACK, Key: R, LeftNode: null, RightNode: null, Parent:O

```
NodeType: BLACK, Key: S, LeftNode: O, RightNode: X, Parent:I
NodeType: RED, Key: T, LeftNode: null, RightNode: null, Parent:X
NodeType: BLACK, Key: X, LeftNode: T, RightNode: null, Parent:S
********************************************************************************
********************************************************************************
After adding Y to tree:
```



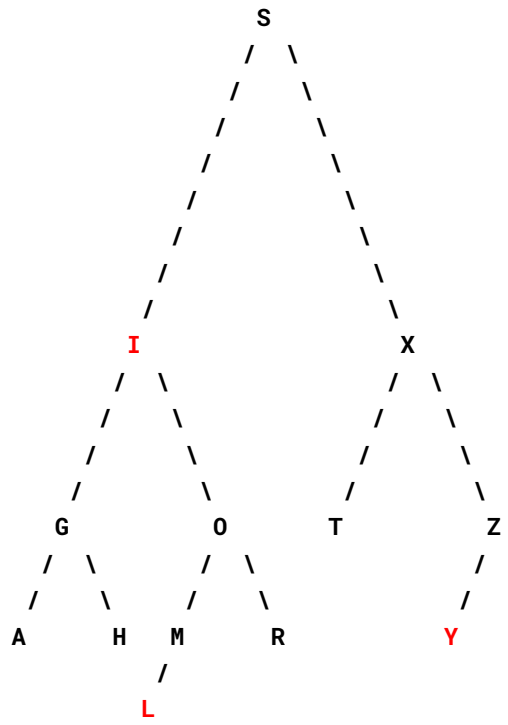

```
Inorder Traversal of Tree:
NodeType: BLACK, Key: A, LeftNode: null, RightNode: null, Parent:G
NodeType: BLACK, Key: G, LeftNode: A, RightNode: H, Parent:I
NodeType: BLACK, Key: H, LeftNode: null, RightNode: null, Parent:G
NodeType: RED, Key: I, LeftNode: G, RightNode: O, Parent:S
NodeType: RED, Key: L, LeftNode: null, RightNode: null, Parent:M
NodeType: BLACK, Key: M, LeftNode: L, RightNode: null, Parent:O
```

```
NodeType: BLACK, Key: O, LeftNode: M, RightNode: R, Parent:I
NodeType: BLACK, Key: R, LeftNode: null, RightNode: null, Parent:O
NodeType: BLACK, Key: S, LeftNode: I, RightNode: X, Parent:null
NodeType: BLACK, Key: T, LeftNode: null, RightNode: null, Parent:X
NodeType: BLACK, Key: X, LeftNode: T, RightNode: Y, Parent:S
NodeType: BLACK, Key: Y, LeftNode: null, RightNode: null, Parent:X
********************************************************************************
********************************************************************************
After adding Z to tree:
```




```
Inorder Traversal of Tree:
NodeType: BLACK, Key: A, LeftNode: null, RightNode: null, Parent:G
NodeType: BLACK, Key: G, LeftNode: A, RightNode: H, Parent:I
```

```
NodeType: BLACK, Key: H, LeftNode: null, RightNode: null, Parent:G
NodeType: RED, Key: I, LeftNode: G, RightNode: O, Parent:S
NodeType: RED, Key: L, LeftNode: null, RightNode: null, Parent:M
NodeType: BLACK, Key: M, LeftNode: L, RightNode: null, Parent:O
NodeType: BLACK, Key: O, LeftNode: M, RightNode: R, Parent:I
NodeType: BLACK, Key: R, LeftNode: null, RightNode: null, Parent:O
NodeType: BLACK, Key: S, LeftNode: I, RightNode: X, Parent:null
NodeType: BLACK, Key: T, LeftNode: null, RightNode: null, Parent:X
NodeType: BLACK, Key: X, LeftNode: T, RightNode: Z, Parent:S
NodeType: RED, Key: Y, LeftNode: null, RightNode: null, Parent:Z
NodeType: BLACK, Key: Z, LeftNode: Y, RightNode: null, Parent:X
********************************************************************************
********************************************************************************
Final tree is:
```



45. Programming Assignment

(a) A Red-Black tree was implemented using the code provided in section 3.3 and problems 3.39 – 3.41. Then keys 1 to 60 were inserted in increasing order into the empty R-B BST tree. Resulting tree output is shown in Figure 1 below.

While parts of the code for this program were derived from Section 3.3, Problems 3.39-3.41, and the code from textbook website (https://algs4.cs.princeton.edu/33balanced/RedBlackBST.java.html), there were several enhancements needed to complete the assignment:

1. There was no ability to visualize and print the resulting R-B trees. I added this to the base program and there is now the ability to visualize the resulting R-B tree from each step optionally.
2. A method to print each tree node in a human readable format to verify the tree nodes and their relationships and help debug the visualization was added (the inorder traversal output illustrates this below)
3. Also missing were various utility functions to print the tree with various types of traversals such as Preorder, Inorder, and Postorder. I added all of these methods. Inorder traversal in particular is very useful in this case as it will print the nodes in ascending order of keys.
4. Color coding of the output to show the red nodes in the console output properly was added as well as the ability to print the tree at each step and to show the node relationships.
5. Having the tree node as a private class within the RedBlack BST code was cumbersome. So I moved this to a separate top level class Node and added various utility methods for traversals and a few new member variables such as parent to improve the performance of navigating the trees.

It is interesting to note that the tree is still well balanced and the black tree height of all leaf nodes is the same (all black leaf nodes are at a depth of 4 from root) as the red nodes **16, 42, 52, 59** are considered the same as filled 3-nodes in the tree. Since we are adding the numbers in ascending order we see a tree rooted almost at the median value of the distribution of values (32) and is constructed in the order of inorder traversal!

****

After adding 60 to tree:

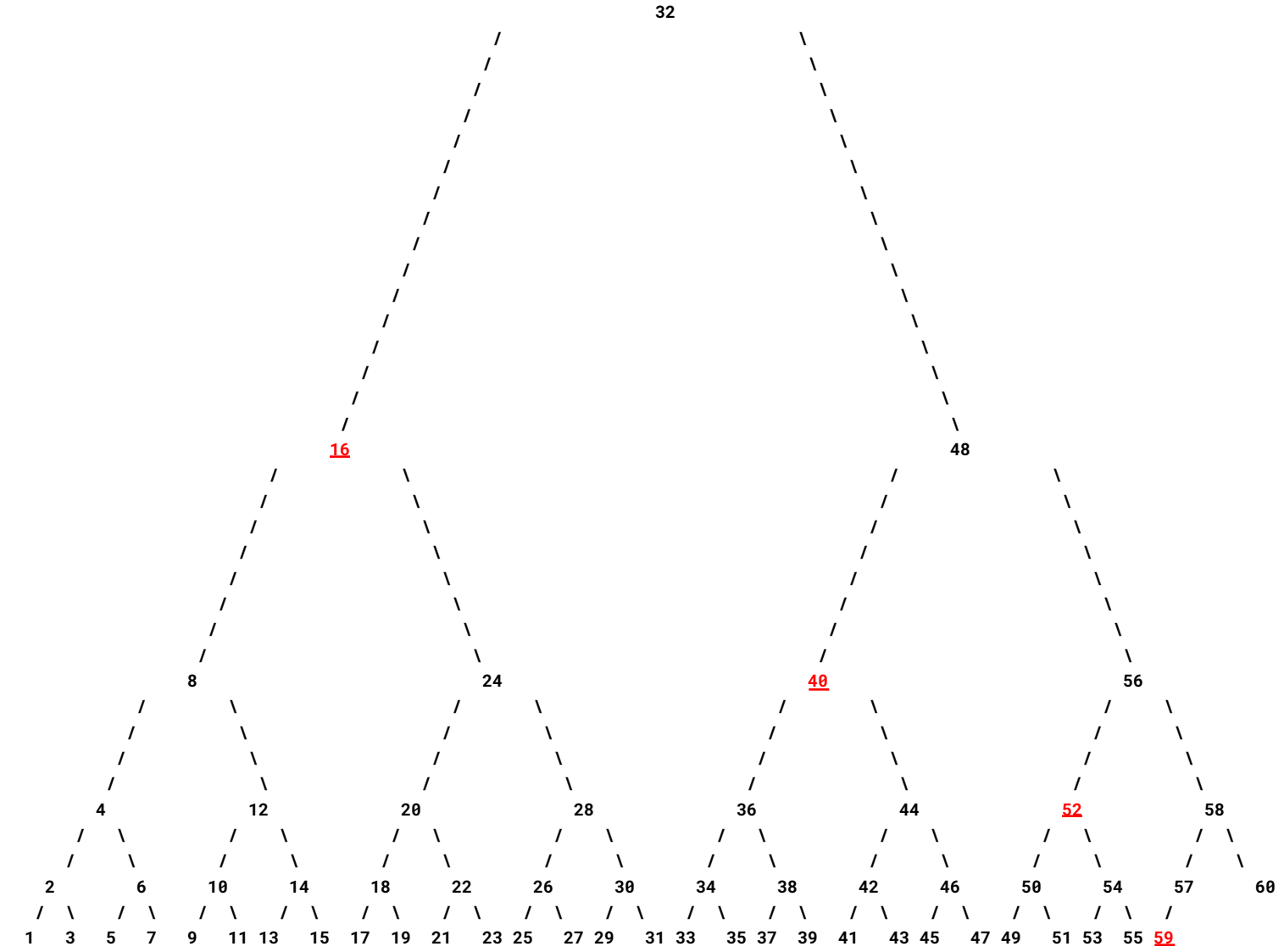


Figure 1: Resulting tree after inserting numbers 1-60 into an R-B BST constructed with numbers inserted in increasing order

```
Inorder Traversal of Tree:
NodeType: BLACK, Key: 1, LeftNode: null, RightNode: null, Parent:2
NodeType: BLACK, Key: 2, LeftNode: 1, RightNode: 3, Parent:4
NodeType: BLACK, Key: 3, LeftNode: null, RightNode: null, Parent:2
NodeType: BLACK, Key: 4, LeftNode: 2, RightNode: 6, Parent:8
NodeType: BLACK, Key: 5, LeftNode: null, RightNode: null, Parent:6
NodeType: BLACK, Key: 6, LeftNode: 5, RightNode: 7, Parent:4
NodeType: BLACK, Key: 7, LeftNode: null, RightNode: null, Parent:6
NodeType: BLACK, Key: 8, LeftNode: 4, RightNode: 12, Parent:16
NodeType: BLACK, Key: 9, LeftNode: null, RightNode: null, Parent:10
NodeType: BLACK, Key: 10, LeftNode: 9, RightNode: 11, Parent:12
NodeType: BLACK, Key: 11, LeftNode: null, RightNode: null, Parent:10
NodeType: BLACK, Key: 12, LeftNode: 10, RightNode: 14, Parent:8
NodeType: BLACK, Key: 13, LeftNode: null, RightNode: null, Parent:14
NodeType: BLACK, Key: 14, LeftNode: 13, RightNode: 15, Parent:12
NodeType: BLACK, Key: 15, LeftNode: null, RightNode: null, Parent:14
NodeType: RED, Key: 16, LeftNode: 8, RightNode: 24, Parent:32
NodeType: BLACK, Key: 17, LeftNode: null, RightNode: null, Parent:18
NodeType: BLACK, Key: 18, LeftNode: 17, RightNode: 19, Parent:20
NodeType: BLACK, Key: 19, LeftNode: null, RightNode: null, Parent:18
NodeType: BLACK, Key: 20, LeftNode: 18, RightNode: 22, Parent:24
NodeType: BLACK, Key: 21, LeftNode: null, RightNode: null, Parent:22
NodeType: BLACK, Key: 22, LeftNode: 21, RightNode: 23, Parent:20
NodeType: BLACK, Key: 23, LeftNode: null, RightNode: null, Parent:22
NodeType: BLACK, Key: 24, LeftNode: 20, RightNode: 28, Parent:16
NodeType: BLACK, Key: 25, LeftNode: null, RightNode: null, Parent:26
NodeType: BLACK, Key: 26, LeftNode: 25, RightNode: 27, Parent:28
NodeType: BLACK, Key: 27, LeftNode: null, RightNode: null, Parent:26
NodeType: BLACK, Key: 28, LeftNode: 26, RightNode: 30, Parent:24
NodeType: BLACK, Key: 29, LeftNode: null, RightNode: null, Parent:30
NodeType: BLACK, Key: 30, LeftNode: 29, RightNode: 31, Parent:28
NodeType: BLACK, Key: 31, LeftNode: null, RightNode: null, Parent:30
NodeType: BLACK, Key: 32, LeftNode: 16, RightNode: 48, Parent:null
NodeType: BLACK, Key: 33, LeftNode: null, RightNode: null, Parent:34
NodeType: BLACK, Key: 34, LeftNode: 33, RightNode: 35, Parent:36
NodeType: BLACK, Key: 35, LeftNode: null, RightNode: null, Parent:34
NodeType: BLACK, Key: 36, LeftNode: 34, RightNode: 38, Parent:40
NodeType: BLACK, Key: 37, LeftNode: null, RightNode: null, Parent:38
NodeType: BLACK, Key: 38, LeftNode: 37, RightNode: 39, Parent:36
NodeType: BLACK, Key: 39, LeftNode: null, RightNode: null, Parent:38
NodeType: RED, Key: 40, LeftNode: 36, RightNode: 44, Parent:48
NodeType: BLACK, Key: 41, LeftNode: null, RightNode: null, Parent:42
NodeType: BLACK, Key: 42, LeftNode: 41, RightNode: 43, Parent:44
```

```
NodeType: BLACK, Key: 43, LeftNode: null, RightNode: null, Parent:42
NodeType: BLACK, Key: 44, LeftNode: 42, RightNode: 46, Parent:40
NodeType: BLACK, Key: 45, LeftNode: null, RightNode: null, Parent:46
NodeType: BLACK, Key: 46, LeftNode: 45, RightNode: 47, Parent:44
NodeType: BLACK, Key: 47, LeftNode: null, RightNode: null, Parent:46
NodeType: BLACK, Key: 48, LeftNode: 40, RightNode: 56, Parent:32
NodeType: BLACK, Key: 49, LeftNode: null, RightNode: null, Parent:50
NodeType: BLACK, Key: 50, LeftNode: 49, RightNode: 51, Parent:52
NodeType: BLACK, Key: 51, LeftNode: null, RightNode: null, Parent:50
NodeType: RED, Key: 52, LeftNode: 50, RightNode: 54, Parent:56
NodeType: BLACK, Key: 53, LeftNode: null, RightNode: null, Parent:54
NodeType: BLACK, Key: 54, LeftNode: 53, RightNode: 55, Parent:52
NodeType: BLACK, Key: 55, LeftNode: null, RightNode: null, Parent:54
NodeType: BLACK, Key: 56, LeftNode: 52, RightNode: 58, Parent:48
NodeType: BLACK, Key: 57, LeftNode: null, RightNode: null, Parent:58
NodeType: BLACK, Key: 58, LeftNode: 57, RightNode: 60, Parent:56
NodeType: RED, Key: 59, LeftNode: null, RightNode: null, Parent:60
NodeType: BLACK, Key: 60, LeftNode: 59, RightNode: null, Parent:58
********************************************************************************
```

b) If we start deleting keys 1 to 20 in increasing order, we observe a similar pattern to how the tree was originally constructed: the root is the median of remaining range (21 to 60). Figure 2 below shows the resulting tree after deleting the first 20 numbers (i.e., after deleting 1-20).

```
********************************************************************************
After deleting 20 from tree:
```

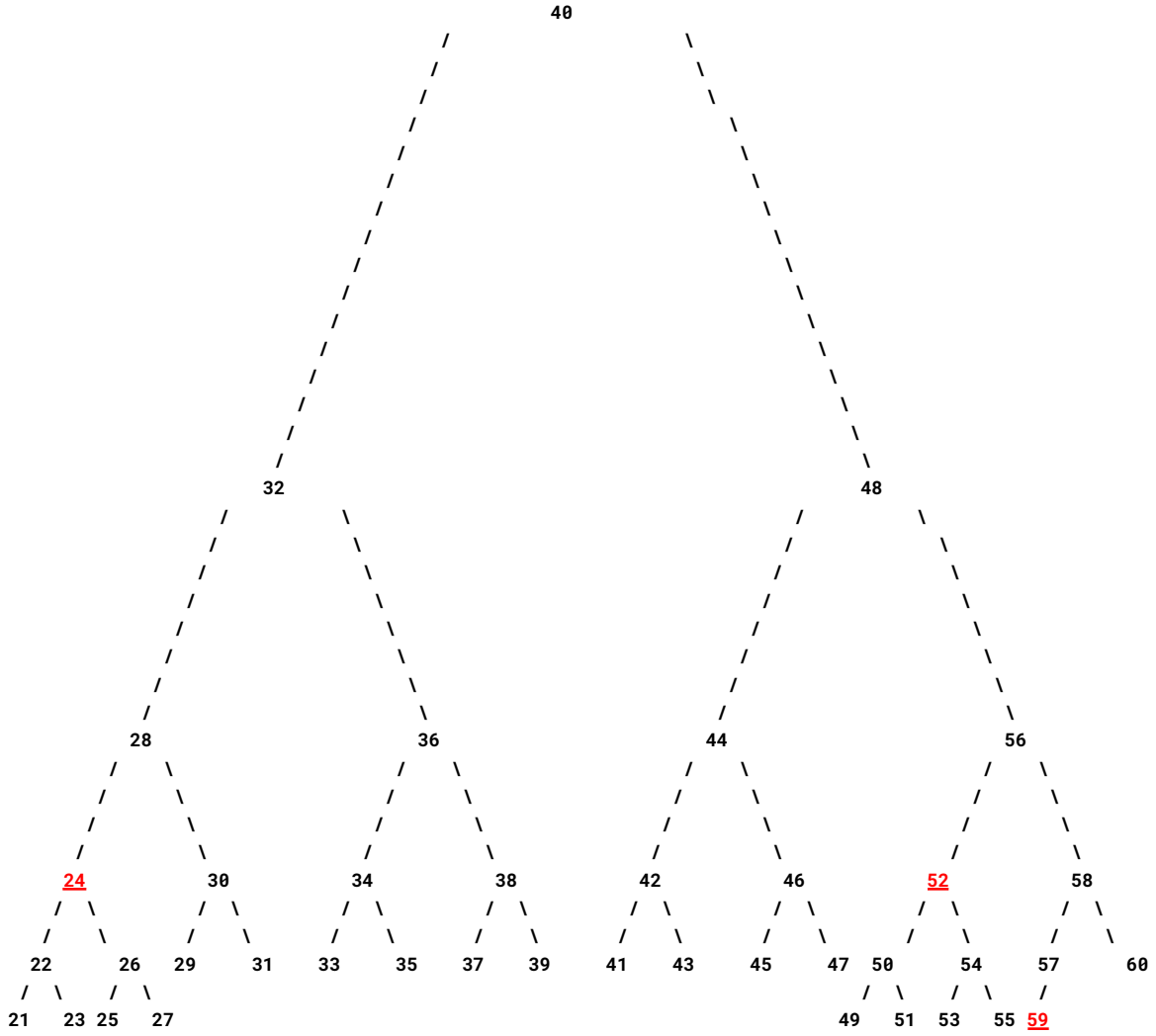


Figure 2: Resulting tree after deleting numbers 1-20 from a R-B BST constructed initially with the numbers 1-60 inserted in increasing order

```
Inorder Traversal of Tree:
NodeType: BLACK, Key: 21, LeftNode: null, RightNode: null, Parent:22
NodeType: BLACK, Key: 22, LeftNode: 21, RightNode: 23, Parent:24
NodeType: BLACK, Key: 23, LeftNode: null, RightNode: null, Parent:22
NodeType: RED, Key: 24, LeftNode: 22, RightNode: 26, Parent:28
NodeType: BLACK, Key: 25, LeftNode: null, RightNode: null, Parent:26
NodeType: BLACK, Key: 26, LeftNode: 25, RightNode: 27, Parent:24
NodeType: BLACK, Key: 27, LeftNode: null, RightNode: null, Parent:26
NodeType: BLACK, Key: 28, LeftNode: 24, RightNode: 30, Parent:32
NodeType: BLACK, Key: 29, LeftNode: null, RightNode: null, Parent:30
NodeType: BLACK, Key: 30, LeftNode: 29, RightNode: 31, Parent:28
NodeType: BLACK, Key: 31, LeftNode: null, RightNode: null, Parent:30
NodeType: BLACK, Key: 32, LeftNode: 28, RightNode: 36, Parent:40
NodeType: BLACK, Key: 33, LeftNode: null, RightNode: null, Parent:34
NodeType: BLACK, Key: 34, LeftNode: 33, RightNode: 35, Parent:36
NodeType: BLACK, Key: 35, LeftNode: null, RightNode: null, Parent:34
NodeType: BLACK, Key: 36, LeftNode: 34, RightNode: 38, Parent:32
NodeType: BLACK, Key: 37, LeftNode: null, RightNode: null, Parent:38
NodeType: BLACK, Key: 38, LeftNode: 37, RightNode: 39, Parent:36
NodeType: BLACK, Key: 39, LeftNode: null, RightNode: null, Parent:38
NodeType: BLACK, Key: 40, LeftNode: 32, RightNode: 48, Parent:null
NodeType: BLACK, Key: 41, LeftNode: null, RightNode: null, Parent:42
NodeType: BLACK, Key: 42, LeftNode: 41, RightNode: 43, Parent:44
NodeType: BLACK, Key: 43, LeftNode: null, RightNode: null, Parent:42
NodeType: BLACK, Key: 44, LeftNode: 42, RightNode: 46, Parent:48
NodeType: BLACK, Key: 45, LeftNode: null, RightNode: null, Parent:46
NodeType: BLACK, Key: 46, LeftNode: 45, RightNode: 47, Parent:44
NodeType: BLACK, Key: 47, LeftNode: null, RightNode: null, Parent:46
NodeType: BLACK, Key: 48, LeftNode: 44, RightNode: 56, Parent:40
NodeType: BLACK, Key: 49, LeftNode: null, RightNode: null, Parent:50
NodeType: BLACK, Key: 50, LeftNode: 49, RightNode: 51, Parent:52
NodeType: BLACK, Key: 51, LeftNode: null, RightNode: null, Parent:50
NodeType: RED, Key: 52, LeftNode: 50, RightNode: 54, Parent:56
NodeType: BLACK, Key: 53, LeftNode: null, RightNode: null, Parent:54
NodeType: BLACK, Key: 54, LeftNode: 53, RightNode: 55, Parent:52
NodeType: BLACK, Key: 55, LeftNode: null, RightNode: null, Parent:54
NodeType: BLACK, Key: 56, LeftNode: 52, RightNode: 58, Parent:48
NodeType: BLACK, Key: 57, LeftNode: null, RightNode: null, Parent:58
NodeType: BLACK, Key: 58, LeftNode: 57, RightNode: 60, Parent:56
NodeType: RED, Key: 59, LeftNode: null, RightNode: null, Parent:60
NodeType: BLACK, Key: 60, LeftNode: 59, RightNode: null, Parent:58
********************************************************************************
```