Ivan Zelenkov

CSCI 2125

Dr. Summa

22 October 2021

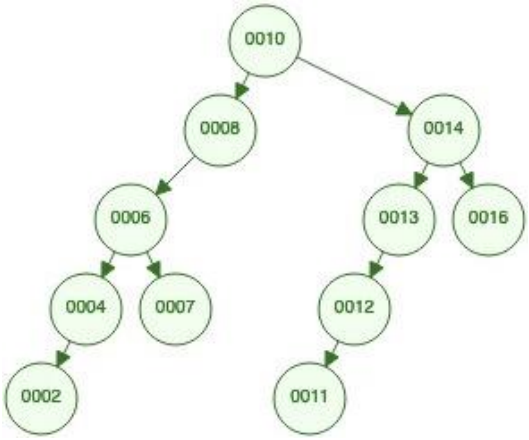
Homework 3

```
public void inclusivePrintValuesBetween(AnyType k1, AnyType k2){
    printInRange(root, k1, k2);
}

private void printInRange(BinaryNode<AnyType> node, AnyType k1, AnyType k2) {
    if (node == null)
        return;

    if (node.element.compareTo(k1) >= 0)
        printInRange(node.left, k1, k2);

    if (node.element.compareTo(k1) >= 0 && node.element.compareTo(k2) <= 0)
        System.out.print(node.element + " ");

    if (node.element.compareTo(k2) <= 0)
        printInRange(node.right, k1, k2);
}
```

Implementation idea:

In the implementation, I used two methods, inclusivePrintValuesBetween, which calls the printInRange method with arguments root, k1, k2. In the printRange method, I used the inorder traversal algorithm to display the items from k1 to k2 in order, where K is the number of items, which is O (K). If the node element is greater than or equal to k1, then we move along the tree to the left. If the node element is less than or equal to k2, then we move along the tree to the right. In general, as we know, the average BST depth is O (log N) on average, therefore we can say that the time complexity of the algorithm will be O (K + log N) on average.

Let's look at an example from the first testFromLeftRight() method of RedirectTester_junit4.java class. The values k1 = 2 and k2 = 14, that is, we want to get all the elements in the tree from 2 to 14. The result should be 2 4 6 7 8 10 11 12 13 14.



We start with the root = 10. Since 10 ≥ k1(2), we move to the left through nodes 8 → 6 → 4 → 2. As soon as we get to 2, the first if-statement is encountered, which is true, because 2 is a leaf. Therefore, we recursively back from 2 to 6. While backtracking occurs, items 2 4 6 will be printed to the screen via System.out.print. Further, as soon as we returned to 6, the fourth condition of the if-statement will be true, which means that we are moving to 7. Since 7 is a leaf, we return to the root 10 (7 → 6 → 8 → 10) and display the nodes that have not been withdrawn.

When it returns to 10, the fourth if-statement will be checked. It will be true because 10 ≤ k2(14). So, we move to the right to 14. As soon as we get to 14, then with a recursive call, everything starts from the beginning of the block of code, therefore the condition of the second if-statement will be true, and we move to the left through 14 → 13 → 12 → 11. Since 11 is a leaf, we go back and output all the nodes along the path to 14, namely 11, 12, 13, 14. After returning to 14, the fourth condition of the if-statement will be true, which means that we move right to 16. 16 is a leaf, therefore we return recursively from 16 to 14, but 16 ∉ [2, 14]. 16 will not be displayed because the condition of the third if-statement will be false (16 is not in range). After returning to 14, the printInRange method exits, thereby completing the inclusivePrintValuesBetween method. Actual result is as expected 2 4 6 7 8 10 11 12 13 14.