Closest Binary Search Tree Value II

Given a non-empty binary search tree and a target value, find k values in the BST that are closest to the target.

Note:

- Given target value is a floating point.
- You may assume k is always valid, that is: $k \leq$ total nodes.
- You are guaranteed to have only one unique set of k values in the BST that are closest to the target.

Follow up:
Assume that the BST is balanced, could you solve it in less than $O(n)$ runtime (where n = total nodes)?

1. Consider implement these two helper functions:
   i. `getPredecessor(N)`, which returns the next smaller node to N.
   ii. `getSuccessor(N)`, which returns the next larger node to N.
2. Try to assume that each node has a parent pointer, it makes the problem much easier.
3. Without parent pointer we just need to keep track of the path from the root to the current node using a stack.
4. You would need two stacks to track the path in finding predecessor and successor node separately.

Solution 1

The idea is to compare the predecessors and successors of the closest node to the target, we can use two stacks to track the predecessors and successors, then like what we do in merge sort, we compare and pick the closest one to the target and put it to the result list.

As we know, inorder traversal gives us sorted predecessors, whereas reverse-inorder traversal gives us sorted successors.

We can use iterative inorder traversal rather than recursion, but to keep the code clean, here is the recursion version.

```
public List<Integer> closestKValues(TreeNode root, double target, int k) {
  List<Integer> res = new ArrayList<>();

  Stack<Integer> s1 = new Stack<>(); // predecessors
  Stack<Integer> s2 = new Stack<>(); // successors

  inorder(root, target, false, s1);
  inorder(root, target, true, s2);

  while (k-- > 0) {
    if (s1.isEmpty())
      res.add(s2.pop());
    else if (s2.isEmpty())
      res.add(s1.pop());
    else if (Math.abs(s1.peek() - target) < Math.abs(s2.peek() - target))
      res.add(s1.pop());
    else
      res.add(s2.pop());
  }

  return res;
}

// inorder traversal
void inorder(TreeNode root, double target, boolean reverse, Stack<Integer> stack)
{
  if (root == null) return;

  inorder(reverse ? root.right : root.left, target, reverse, stack);
  // early terminate, no need to traverse the whole tree
  if ((reverse && root.val <= target) || (!reverse && root.val > target)) return;
  // track the value of current node
  stack.push(root.val);
  inorder(reverse ? root.left : root.right, target, reverse, stack);
}
```

written by jeantimex original link here

Solution 2

```java
public class Solution {
    public List<Integer> closestKValues(TreeNode root, double target, int k) {
        List<Integer> ret = new LinkedList<>();
        Stack<TreeNode> succ = new Stack<>();
        Stack<TreeNode> pred = new Stack<>();
        initializePredecessorStack(root, target, pred);
        initializeSuccessorStack(root, target, succ);
        if(!succ.isEmpty() && !pred.isEmpty() && succ.peek().val == pred.peek().v
al) {
            getNextPredecessor(pred);
        }
        while(k-- > 0) {
            if(succ.isEmpty()) {
                ret.add(getNextPredecessor(pred));
            } else if(pred.isEmpty()) {
                ret.add(getNextSuccessor(succ));
            } else {
                double succ_diff = Math.abs((double)succ.peek().val - target);
                double pred_diff = Math.abs((double)pred.peek().val - target);
                if(succ_diff < pred_diff) {
                    ret.add(getNextSuccessor(succ));
                } else {
                    ret.add(getNextPredecessor(pred));
                }
            }
        }
        return ret;
    }

    private void initializeSuccessorStack(TreeNode root, double target, Stack<Tre
eNode> succ) {
        while(root != null) {
            if(root.val == target) {
                succ.push(root);
                break;
            } else if(root.val > target) {
                succ.push(root);
                root = root.left;
            } else {
                root = root.right;
            }
        }
    }

    private void initializePredecessorStack(TreeNode root, double target, Stack<T
reeNode> pred) {
        while(root != null){
            if(root.val == target){
                pred.push(root);
                break;
            } else if(root.val < target){
                pred.push(root);
                root = root.right;
            } else{
                root = root.left;
            }
        }
    }

    private int getNextSuccessor(Stack<TreeNode> succ) {
        TreeNode curr = succ.pop();
        int ret = curr.val;
        curr = curr.right;
        while(curr != null) {
            succ.push(curr);
            curr = curr.left;
        }
        return ret;
    }

    private int getNextPredecessor(Stack<TreeNode> pred) {
        TreeNode curr = pred.pop();
        int ret = curr.val;
        curr = curr.left;
        while(curr != null) {
            pred.push(curr);
            curr = curr.right;
        }
        return ret;
    }
}
```

written by qianzhige original link here

Solution 3

```
public class Solution {
    public List<Integer> closestKValues(TreeNode root, double target, int k) {
        LinkedList<Integer> list = new LinkedList<Integer>();
        closestKValuesHelper(list, root, target, k);
        return list;
    }

    /**
     * @return <code>true</code> if result is already found.
     */
    private boolean closestKValuesHelper(LinkedList<Integer> list, TreeNode root,
double target, int k) {
        if (root == null) {
            return false;
        }

        if (closestKValuesHelper(list, root.left, target, k)) {
            return true;
        }

        if (list.size() == k) {
            if (Math.abs(list.getFirst() - target) < Math.abs(root.val - target))
{
                return true;
            } else {
                list.removeFirst();
            }
        }

        list.addLast(root.val);
        return closestKValuesHelper(list, root.right, target, k);
    }
}
```

written by magma917 original link here