Unique Binary Search Trees II

Given *n*, generate all structurally unique **BST's** (binary search trees) that store values 1...*n*.

For example,
Given *n* = 3, your program should return all 5 unique BST's shown below.

```
   1         3     3      2      1
    \       /     /      / \      \
     3     2     1      1   3      2
    /     /       \                 \
   2     1         2                 3
```

confused what `"{1,#,2,3}"` means? > read more on how binary tree is serialized on OJ.

OJ's Binary Tree Serialization:
The serialization of a binary tree follows a level order traversal, where '#' signifies a path terminator where no node exists below.

Here's an example:

```
   1
  / \
 2   3
    /
   4
    \
     5
```

The above binary tree is serialized as `"{1,2,3,#,#,4,#,#,5}"`.

Solution 1

I start by noting that 1..n is the in-order traversal for any BST with nodes 1 to n. So if I pick i-th node as my root, the left subtree will contain elements 1 to (i-1), and the right subtree will contain elements (i+1) to n. I use recursive calls to get back all possible trees for left and right subtrees and combine them in all possible ways with the root.

```
public class Solution {
    public List<TreeNode> generateTrees(int n) {

        return genTrees(1,n);
    }

    public List<TreeNode> genTrees (int start, int end)
    {

        List<TreeNode> list = new ArrayList<TreeNode>();

        if(start>end)
        {
            list.add(null);
            return list;
        }

        if(start == end){
            list.add(new TreeNode(start));
            return list;
        }

        List<TreeNode> left,right;
        for(int i=start;i<=end;i++)
        {

            left = genTrees(start, i-1);
            right = genTrees(i+1,end);

            for(TreeNode lnode: left)
            {
                for(TreeNode rnode: right)
                {
                    TreeNode root = new TreeNode(i);
                    root.left = lnode;
                    root.right = rnode;
                    list.add(root);
                }
            }

        }

        return list;
    }
}
```

written by Jayanta original link here

Solution 2

Here is my java solution with DP:

```
public class Solution {
    public static List<TreeNode> generateTrees(int n) {
        List<TreeNode>[] result = new List[n+1];
        result[0] = new ArrayList<TreeNode>();
        result[0].add(null);

        for(int len = 1; len <= n; len++){
            result[len] = new ArrayList<TreeNode>();
            for(int j=0; j<len; j++){
                for(TreeNode nodeL : result[j]){
                    for(TreeNode nodeR : result[len-j-1]){
                        TreeNode node = new TreeNode(j+1);
                        node.left = nodeL;
                        node.right = clone(nodeR, j+1);
                        result[len].add(node);
                    }
                }
            }
        }
        return result[n];
    }

    private static TreeNode clone(TreeNode n, int offset){
        if(n == null)
            return null;
        TreeNode node = new TreeNode(n.val + offset);
        node.left = clone(n.left, offset);
        node.right = clone(n.right, offset);
        return node;
    }
}
```

result[i] stores the result until length **i**. For the result for length i+1, select the root node j from 0 to i, combine the result from left side and right side. Note for the right side we have to clone the nodes as the value will be offsetted by **j**.

written by jianwu original link here

Solution 3

This problem is a variant of the problem of Unique Binary Search Trees.

I provided a solution along with explanation for the above problem, in the question "DP solution in 6 lines with explanation"

It is intuitive to solve this problem by following the same algorithm. Here is the code in a divide-and-conquer style.

```
public List<TreeNode> generateTrees(int n) {
    return generateSubtrees(1, n);
}

private List<TreeNode> generateSubtrees(int s, int e) {
    List<TreeNode> res = new LinkedList<TreeNode>();
    if (s > e) {
        res.add(null); // empty tree
        return res;
    }

    for (int i = s; i <= e; ++i) {
        List<TreeNode> leftSubtrees = generateSubtrees(s, i - 1);
        List<TreeNode> rightSubtrees = generateSubtrees(i + 1, e);

        for (TreeNode left : leftSubtrees) {
            for (TreeNode right : rightSubtrees) {
                TreeNode root = new TreeNode(i);
                root.left = left;
                root.right = right;
                res.add(root);
            }
        }
    }
    return res;
}
```

written by liaison original link here

From Leetcoder.