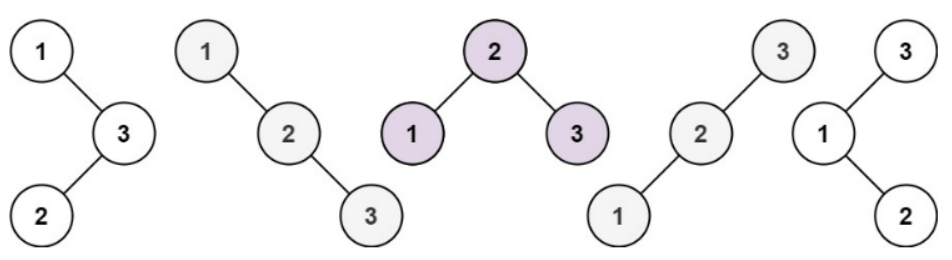
<https://leetcode.com/problems/unique-binary-search-trees-ii/>

Given an integer n, return *all the structurally unique* ***BST'****s (binary search trees), which has exactly* n *nodes of unique values from* 1 *to* n. Return the answer in **any order**.

**Example 1:**



Input: n = 3

Output: [[1,null,2,null,3],[1,null,3,2],[2,1,3],[3,1,null,null,2],[3,2,null,1]]

**Example 2:**

Input: n = 1

Output: [[1]]

**Constraints:**

* 1 <= n <= 8

**Attempt 1: 2023-01-20**

**Solution 1: Divide and Conquer (60 min)**

/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode() {}

\* TreeNode(int val) { this.val = val; }

\* TreeNode(int val, TreeNode left, TreeNode right) {

\* this.val = val;

\* this.left = left;

\* this.right = right;

\* }

\* }

\*/

class Solution {

public List<TreeNode> generateTrees(int n) {

return helper(1, n);

}

private List<TreeNode> helper(int lo, int hi) {

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

// Because when we reach to the leave node and you still call

// recursive helper() method, the boundary will become below:

// e.g i = 5 -> left = helper(5, 4), right = helper(6, 5)

// hence lo > hi always the terminate condition

if(lo > hi) {

// Empty tree

result.add(null);

return result;

}

for(int i = lo; i <= hi; i++) {

// Divide

List<TreeNode> left = helper(lo, i - 1);

List<TreeNode> right = helper(i + 1, hi);

// Conquer

for(int j = 0; j < left.size(); j++) {

for(int k = 0; k < right.size(); k++) {

TreeNode root = new TreeNode(i);

root.left = left.get(j);

root.right = right.get(k);

result.add(root);

}

}

}

return result;

}

}

**Refer to**

<https://leetcode.com/problems/unique-binary-search-trees-ii/solutions/31508/divide-and-conquer-f-i-g-i-1-g-n-i/>

This problem is a variant of the problem of [Unique Binary Search Trees](https://oj.leetcode.com/problems/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"](https://leetcode.com/problems/unique-binary-search-trees/discuss/31666/DP-Solution-in-6-lines-with-explanation.-F(i-n)-G(i-1)-*-G(n-i))

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;

}

**Refer to**

<https://leetcode.com/problems/unique-binary-search-trees-ii/solutions/1440128/python-dfs-with-memoization-clean-concise/>

**Idea**

* Let dfs(left, right) return all valid BSTs where values in the BST in range [left..right].
* Then dfs(1, n) is our result.
* To solve dfs(left, right), we just

1. Generate root value in range [left...right]
2. Get left subtrees by leftNodes = dfs(left, root-1)
3. Get right subtrees by rightNodes = dfs(root+1, right).
4. Add all combination between leftNodes and rightNodes to form root trees.

* Can we cache the result of dfs(left, right) to prevent it to re-compute multiple time.
* There is a simillar problem, which is **[894. All Possible Full Binary Trees](https://leetcode.com/problems/all-possible-full-binary-trees/)**, try to solve it yourself.

class Solution:

def generateTrees(self, n: int) -> List[Optional[TreeNode]]:

@lru\_cache(None)

def dfs(left, right):

if left > right: return [None]

if left == right: return [TreeNode(left)]

ans = []

for root in range(left, right+1):

leftNodes = dfs(left, root - 1)

rightNodes = dfs(root+1, right)

for leftNode in leftNodes:

for rightNode in rightNodes:

rootNode = TreeNode(root, leftNode, rightNode)

ans.append(rootNode)

return ans

return dfs(1, n)