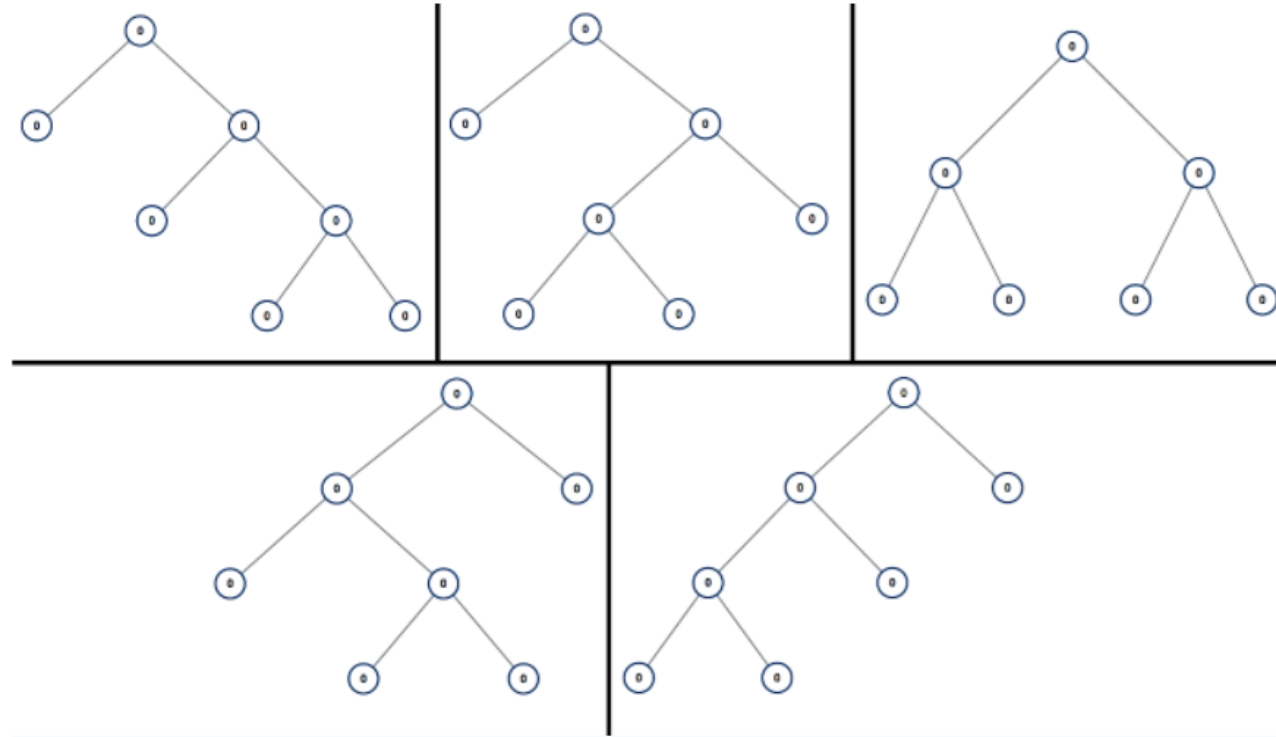




```
1 894. | Medium | All Possible Full BInary Trees | Recursion.
2
3 Given an integer n, return a list of all possible full binary trees with n nodes.
4 Each node of each tree in the answer must have Node.val == 0.
5 Each element of the answer is the root node of one possible tree.
6 You may return the final list of trees in any order.
7
8 A full binary tree is a binary tree where each node has exactly 0 or 2 children.
9
10 Constraints:
11 1 <= n <= 20
```

Example 1:



Input: $n = 7$

Output: `[[0,0,0,null,null,0,0,null,null,0,0],[0,0,0,null,null,0,0,0,0],[0,0,0,0,0,0,0,0],
[0,0,0,0,0,null,null,null,null,0,0],[0,0,0,0,0,null,null,0,0]]`

Example 2:

Input: $n = 3$

Output: `[[0,0,0]]`



```
1  vector<TreeNode*> allPossibleFBT(int n) {
2      vector<TreeNode*> result, leftTree, rightTree;
3
4      // base condition
5      if(n == 1) {
6          TreeNode* root = new TreeNode(0);
7          result.push_back(root);
8          return result;
9      }
10
11     // main logic
12     for(int i=1; i<n; i += 2) {
13         leftTree = allPossibleFBT(i);
14         rightTree = allPossibleFBT(n-i-1);
15
16         for(int j=0; j<leftTree.size(); j++) {
17             for(int k=0; k<rightTree.size(); k++) {
18                 TreeNode* root = new TreeNode(0);
19                 root->left = leftTree[j];
20                 root->right = rightTree[k];
21                 result.push_back(root);
22             }
23         }
24     }
25     return result;
26 }
```

We can further optimise the time, space and memory with dynamic programming by just writing 2 line in the same code.



```
1 unordered_map<int, vector<TreeNode*>> dp;
2 vector<TreeNode*> allPossibleFBT(int n) {
3     vector<TreeNode*> result, leftTree, rightTree;
4
5     // checking if present in the hashmap then don't calculate again.
6     if(dp.count(n) > 0) {
7         return dp[n];
8     }
9
10    // base condition
11    if(n == 1) {
12        TreeNode* root = new TreeNode(0);
13        result.push_back(root);
14        return result;
15    }
16
17    // main logic
18    for(int i=1; i<n; i += 2) {
19        leftTree = allPossibleFBT(i);
20        rightTree = allPossibleFBT(n-i-1);
21
22        for(int j=0; j<leftTree.size(); j++) {
23            for(int k=0; k<rightTree.size(); k++) {
24                TreeNode* root = new TreeNode(0);
25                root->left = leftTree[j];
26                root->right = rightTree[k];
27                result.push_back(root);
28            }
29        }
30    }
31
32    // storing in the dp.
33    dp.insert({n, result});
34    return result;
35 }
```

#100daysofDSA



/rvislive

Rakesh Vishwakarma