Description | Editorial | Solutions (997) | Submissions

894. All Possible Full Binary Trees

Medium 4.4K 294

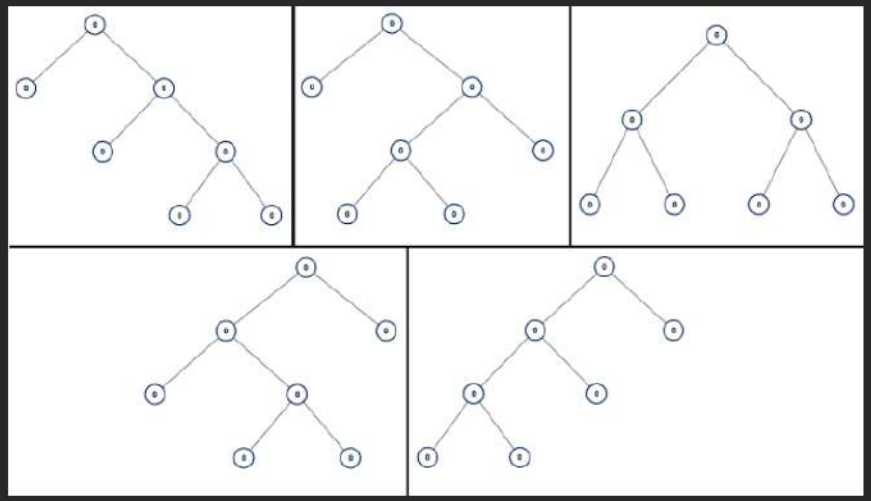
Companies

Given an integer `n`, return *a list of all possible* ***full binary trees*** *with* `n` *nodes*. Each node of each tree in the answer must have `Node.val == 0`.

Each element of the answer is the root node of one possible tree. You may return the final list of trees in **any order**.

A **full binary tree** is a binary tree where each node has exactly `0` or `2` children.

Example 1:



Input: n = 7
Output: [[0,0,0,null,null,0,0,null,null,0,0],

Java | Auto

```java
*/
class Solution {
    static Map<Integer, List<TreeNode>> saved = new HashMap<>();

    public List<TreeNode> allPossibleFBT(int n) {
        if (n%2==0)
            return new ArrayList<>();

        if (!saved.containsKey(n)) {
            List<TreeNode> list = new ArrayList<>();

            if (n==1)
                list.add(new TreeNode(0));
            else {
                for (int i=1; i<=n-1; i+=2) {
                    List<TreeNode> lTrees = allPossibleFBT(i);
                    List<TreeNode> rTrees = allPossibleFBT(n-i-1);

```

Testcase | Result

Accepted Runtime: 0 ms

Case 1 | Case 2

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,null,null,null,null,0,0],[0,0,0,0,0,null,null,0,0]]
```

Console | Run | Submit

Description | Editorial | Solutions (2.4K) | Submissions

890. Find and Replace Pattern

Medium | 3.7K | 167

Companies

Given a list of strings `words` and a string `pattern`, return *a list of* `words[i]` *that match* `pattern`. You may return the answer in **any order**.

A word matches the pattern if there exists a permutation of letters `p` so that after replacing every letter `x` in the pattern with `p(x)`, we get the desired word.

Recall that a permutation of letters is a bijection from letters to letters: every letter maps to another letter, and no two letters map to the same letter.

Example 1:

```
Input: words = ["abc","deq","mee","aqq","dkd","ccc"], pattern = "abb"
Output: ["mee","aqq"]
Explanation: "mee" matches the pattern because there is a permutation {a -> m, b -> e, ...}.
"ccc" does not match the pattern because {a -> c, b -> c, ...} is not a permutation, since a and b map to the same letter.
```

Example 2:

```
Input: words = ["a","b","c"], pattern = "a"
Output: ["a","b","c"]
```

Java | Auto

```java
class Solution {
    public List<String> findAndReplacePattern(String[] words, String pattern) {
        List<String> res = new ArrayList<>();
        for (String word : words) {
            if (check(word, pattern)) res.add(word);
        }
        return res;
    }
    boolean check(String a, String b) {
        for (int i = 0; i < a.length(); i++) {
            if (a.indexOf(a.charAt(i)) != b.indexOf(b.charAt(i))) return false;
        }
        return true;
    }
}
```

Testcase | Result

Accepted Runtime: 0 ms

Case 1 | Case 2

Input

words =
`["abc","deq","mee","aqq","dkd","ccc"]`

pattern =
`"abb"`

Output

`["mee","aqq"]`

Console | Run | Submit