Description | Editorial | Solutions (1.4K) | Submissions

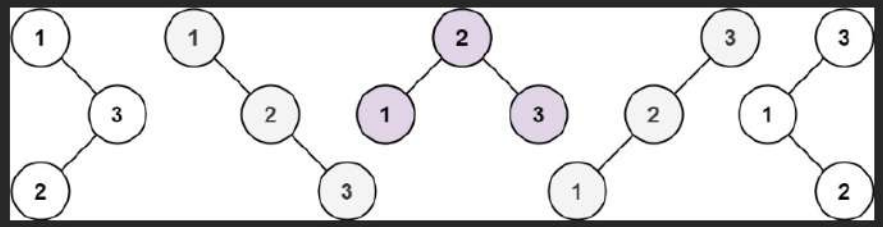
95. Unique Binary Search Trees II

Medium | 6.1K | 398

Companies

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`

Java | Auto

```java
/**
 */
class Solution {
    public List<TreeNode> generateTrees(int n) {
        List<TreeNode>[][] dp=new ArrayList[n+1][n+1];
        return memo(1,n,dp);
    }
    List<TreeNode> memo(int s,int e,List<TreeNode>[][] dp){
        if(s>e){
            List<TreeNode> a=new ArrayList<>();
            a.add(null);
            return a;
        }
        if(dp[s][e]!=null) return dp[s][e];
        dp[s][e]=new ArrayList<>();
        for(int i=s;i<=e;i++){
            List<TreeNode> left=memo(s,i-1,dp);
            List<TreeNode> right=memo(i+1,e,dp);
            for(TreeNode l:left)
                for(TreeNode r:right)
```

Testcase | Result

Accepted Runtime: 0 ms

Case 1 | Case 2

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]]
```

Expected

```
[[1,null,2,null,3],[1,null,3,2],[2,1,3],[3,1,null,null,2],[3,2,null,1]]
```

Console | Run | Submit

Description | Editorial | Solutions (2.5K) | Submissions

435. Non-overlapping Intervals

Medium 6.9K 180

Companies

Given an array of intervals `intervals` where `intervals[i] = [start_i, end_i]`, return *the minimum number of intervals you need to remove to make the rest of the intervals non-overlapping.*

Example 1:

```
Input: intervals = [[1,2],[2,3],[3,4],[1,3]]
Output: 1
Explanation: [1,3] can be removed and the rest of the intervals are non-overlapping.
```

Example 2:

```
Input: intervals = [[1,2],[1,2],[1,2]]
Output: 2
Explanation: You need to remove two [1,2] to make the rest of the intervals non-overlapping.
```

Example 3:

```
Input: intervals = [[1,2],[2,3]]
Output: 0
Explanation: You don't need to remove any of the intervals since they're already non-overlapping.
```

Constraints:

Java | Auto

```java
class Solution {
    public int eraseOverlapIntervals(int[][] intervals) {
        int n = intervals.length;
        Arrays.sort(intervals, (a, b) -> Integer.compare(a[1], b[1]));

        int prev = 0;
        int count = 1;

        for (int i = 1; i < n; i++) {
            if (intervals[i][0] >= intervals[prev][1]) {
                prev = i;
                count++;
            }
        }
        return n - count;
    }
}
```

Testcase | Result

Accepted Runtime: 1 ms

Case 1 | Case 2 | Case 3

Input

intervals =

[[1,2],[2,3],[3,4],[1,3]]

Output

1

Expected

1

Console | Run | Submit