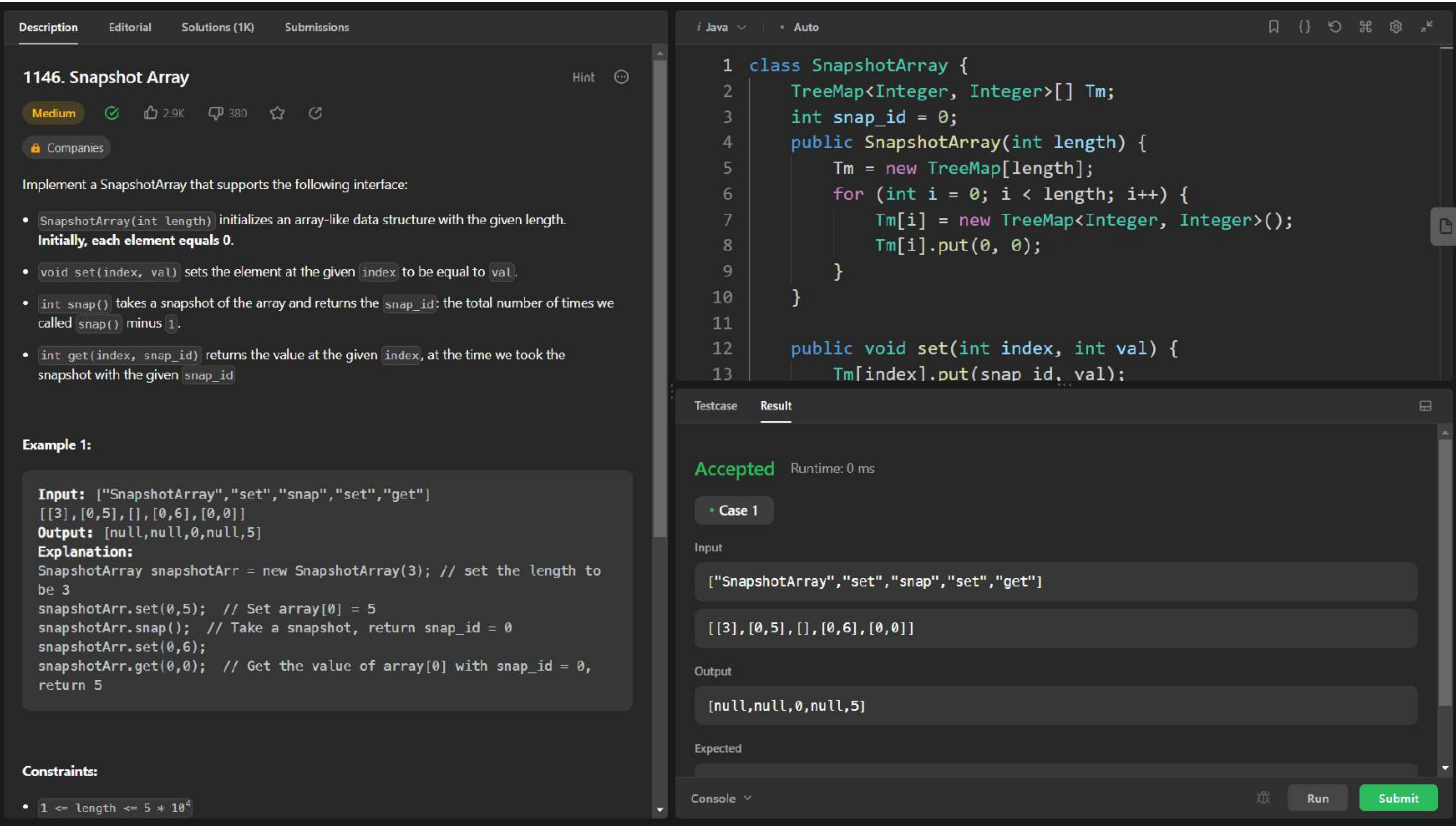


1146. Snapshot Array

Implement a SnapshotArray that supports the following interface:

- `SnapshotArray(int length)` initializes an array-like data structure with the given length. **Initially, each element equals 0**.
- `void set(index, val)` sets the element at the given `index` to be equal to `val`.
- `int snap()` takes a snapshot of the array and returns the `snap_id`: the total number of times we called `snap()` minus `1`.
- `int get(index, snap_id)` returns the value at the given `index`, at the time we took the snapshot with the given `snap_id`

Example 1:

```
Input: ["SnapshotArray","set","snap","set","get"]
[[3],[0,5],[],[0,6],[0,0]]
Output: [null,null,0,null,5]
Explanation:
SnapshotArray snapshotArr = new SnapshotArray(3); // set the length to
be 3
snapshotArr.set(0,5);  // Set array[0] = 5
snapshotArr.snap();  // Take a snapshot, return snap_id = 0
snapshotArr.set(0,6);
snapshotArr.get(0,0);  // Get the value of array[0] with snap_id = 0,
return 5
```

Constraints:

- `1 <= length <= 5 * 10^4`

```java
class SnapshotArray {
    TreeMap<Integer, Integer>[] Tm;
    int snap_id = 0;
    public SnapshotArray(int length) {
        Tm = new TreeMap[length];
        for (int i = 0; i < length; i++) {
            Tm[i] = new TreeMap<Integer, Integer>();
            Tm[i].put(0, 0);
        }
    }

    public void set(int index, int val) {
        Tm[index].put(snap_id, val);
```

Accepted Runtime: 0 ms

Case 1

Input

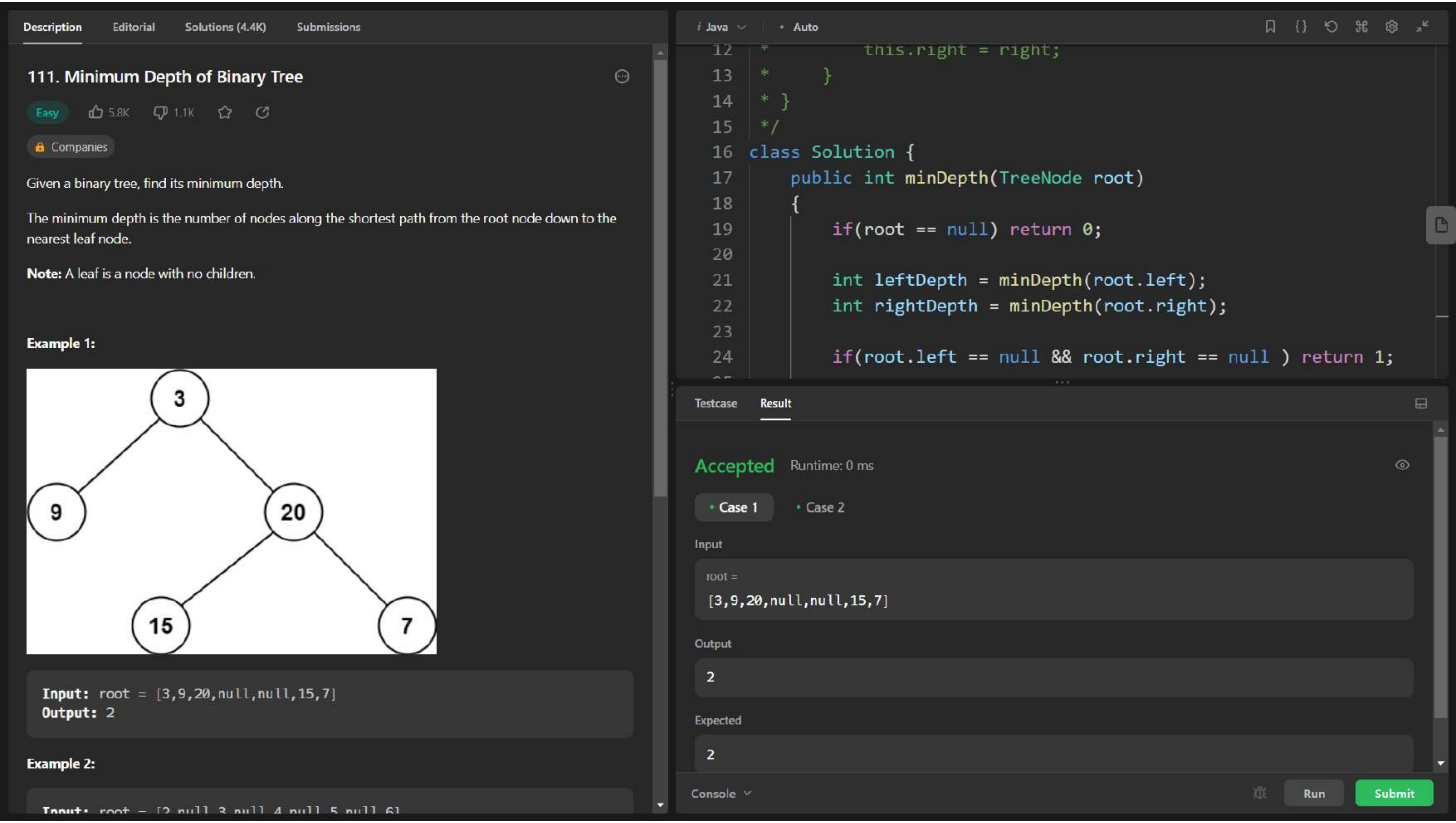
```
["SnapshotArray","set","snap","set","get"]
```

```
[[3],[0,5],[],[0,6],[0,0]]
```

Output

```
[null,null,0,null,5]
```

Expected



111. Minimum Depth of Binary Tree

Given a binary tree, find its minimum depth.

The minimum depth is the number of nodes along the shortest path from the root node down to the nearest leaf node.

Note: A leaf is a node with no children.

Example 1:

```
Input: root = [3,9,20,null,null,15,7]
Output: 2
```

Example 2:

```
Input: root = [2,null,3,null,4,null,5,null,6]
```

```
 *             this.right = right;
 *         }
 * }
 */
class Solution {
    public int minDepth(TreeNode root)
    {
        if(root == null) return 0;

        int leftDepth = minDepth(root.left);
        int rightDepth = minDepth(root.right);

        if(root.left == null && root.right == null ) return 1;
```

Accepted Runtime: 0 ms

• Case 1 • Case 2

Input

root =

```
[3,9,20,null,null,15,7]
```

Output

```
2
```

Expected

```
2
```