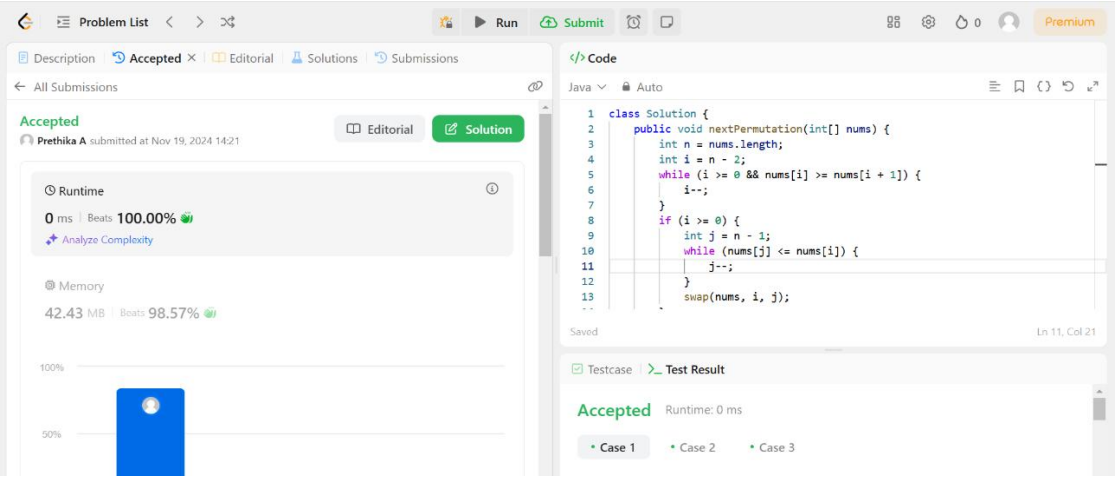
DSA Coding Practice – 7

1.Next permutation

```
class Solution {
    public void nextPermutation(int[] nums) {
        int n = nums.length;
        int i = n - 2;
        while (i >= 0 && nums[i] >= nums[i + 1]) {
            i--;
        }
        if (i >= 0) {
            int j = n - 1;
            while (nums[j] <= nums[i]) {
                j--;
            }
            swap(nums, i, j);
        }
        reverse(nums, i + 1, n - 1);
    }
    private void swap(int[] nums, int i, int j) {
        int temp = nums[i];
        nums[i] = nums[j];
        nums[j] = temp;
    }
    public void reverse(int[] nums, int start, int end) {
        while (start < end) {
            swap(nums, start, end);
```

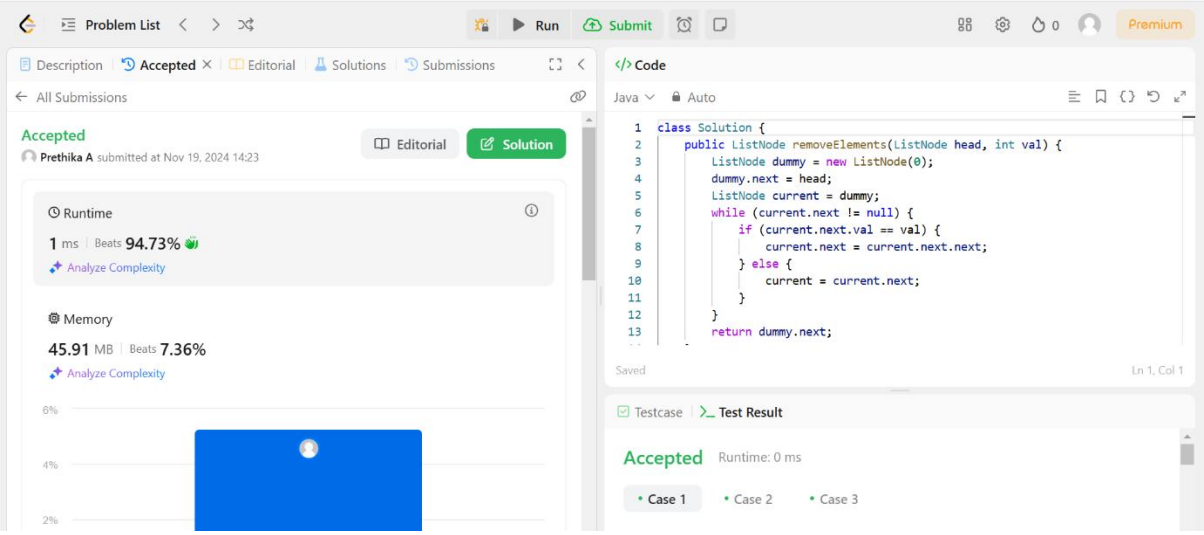
```
            start++;
            end--;
        }
    }
}
```

Output



Time Complexity : O(n)

2.Remove linked list elements

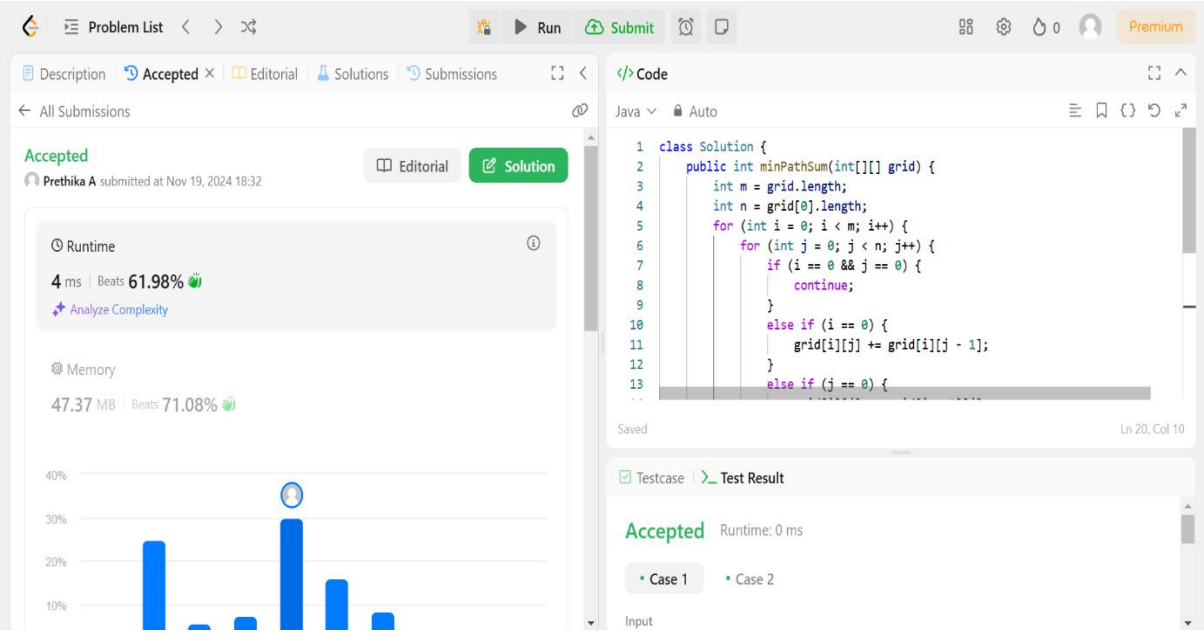


Time Complexity : O(n)

3. Minimum path sum

```
class Solution {
    public int minPathSum(int[][] grid) {
        int m = grid.length;
        int n = grid[0].length;
        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
                if (i == 0 && j == 0) {
                    continue;
                }
                else if (i == 0) {
                    grid[i][j] += grid[i][j - 1];
                }
                else if (j == 0) {
                    grid[i][j] += grid[i - 1][j];
                }
                else {
                    grid[i][j] += Math.min(grid[i - 1][j], grid[i][j - 1]);
                }
            }
        }
        return grid[m - 1][n - 1];
    }
}
```

Output



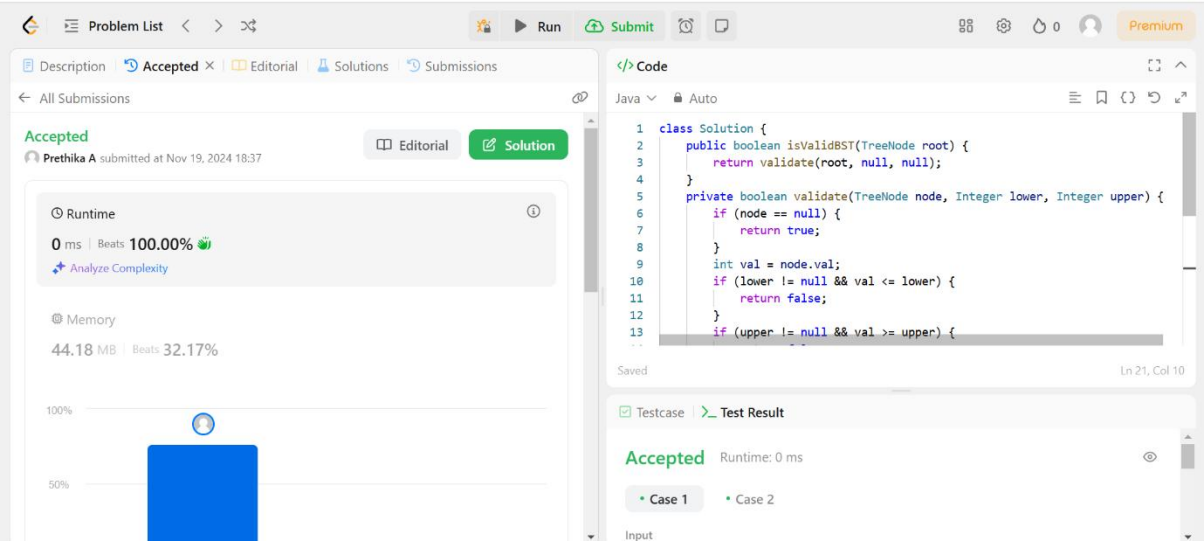
Time Complexity : O(m×n)

4. Validate binary search tree

```
class Solution {
    public boolean isValidBST(TreeNode root) {
        return validate(root, null, null);
    }
    private boolean validate(TreeNode node, Integer lower, Integer upper) {
        if (node == null) {
            return true;
        }
        int val = node.val;
        if (lower != null && val <= lower) {
            return false;
        }
        if (upper != null && val >= upper) {
```

```
            return false;
        }
        if (!validate(node.right, val, upper)) {
            return false;
        }
        if (!validate(node.left, lower, val)) {
            return false;
        }
        return true;
    }
}
```

Output



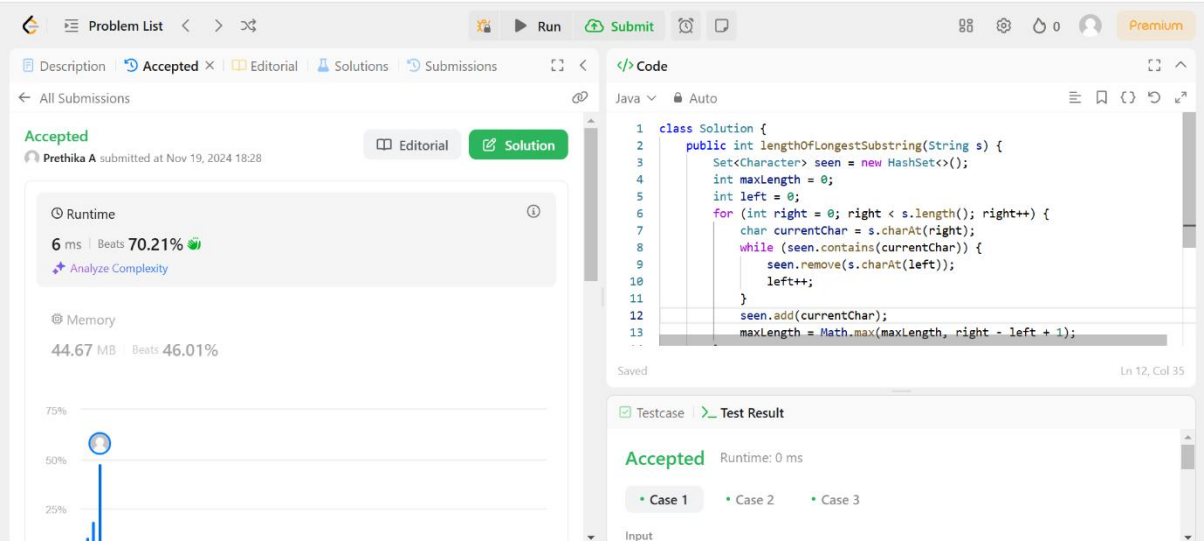
Time Complexity : O(n)

5.Longest substring without repeating characters

```
class Solution {
    public int lengthOfLongestSubstring(String s) {
        Set<Character> seen = new HashSet<>();
```

```
        int maxLength = 0;
        int left = 0;
        for (int right = 0; right < s.length(); right++) {
            char currentChar = s.charAt(right);
            while (seen.contains(currentChar)) {
                seen.remove(s.charAt(left));
                left++;
            }
            seen.add(currentChar);
            maxLength = Math.max(maxLength, right - left + 1);
        }
        return maxLength;
    }
}
```

Output



Time Complexity : O(n)