

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

Accepted

Prethika A submitted at Nov 19, 2024 14:21

Runtime: 0 ms | Beats: 100.00%

Memory: 42.43 MB | Beats: 98.57%

```

1 class Solution {
2     public void nextPermutation(int[] nums) {
3         int n = nums.length;
4         int i = n - 2;
5         while (i >= 0 && nums[i] >= nums[i + 1]) {
6             i--;
7         }
8         if (i >= 0) {
9             int j = n - 1;
10            while (nums[j] <= nums[i]) {
11                j--;
12            }
13            swap(nums, i, j);

```

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

Time Complexity : $O(n)$

2.Remove linked list elements

Accepted

Prethika A submitted at Nov 19, 2024 14:23

Runtime: 1 ms | Beats: 94.73%

Memory: 45.91 MB | Beats: 7.36%

```

1 class Solution {
2     public ListNode removeElements(ListNode head, int val) {
3         ListNode dummy = new ListNode(0);
4         dummy.next = head;
5         ListNode current = dummy;
6         while (current.next != null) {
7             if (current.next.val == val) {
8                 current.next = current.next.next;
9             } else {
10                current = current.next;
11            }
12        }
13        return dummy.next;

```

Accepted Runtime: 0 ms

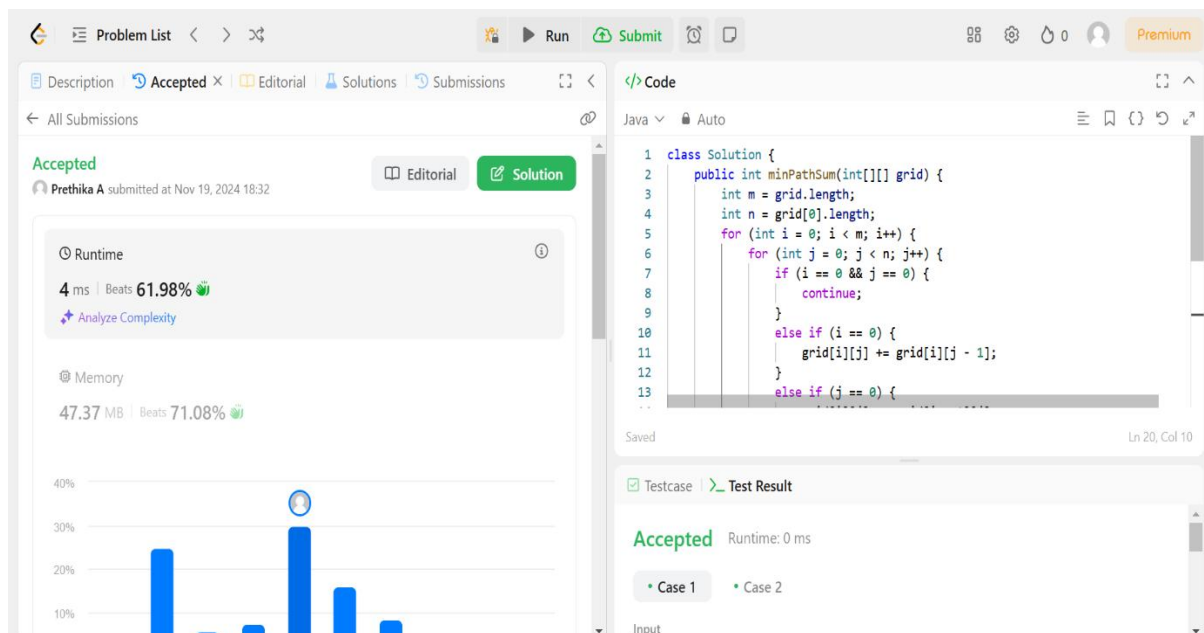
Case 1 Case 2 Case 3

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 : $O(m \times n)$

4. Validate binary search tree

```
class Solution {
    public boolean isValidBST(TreeNode root) {
        return validate(root, null, null);
    }

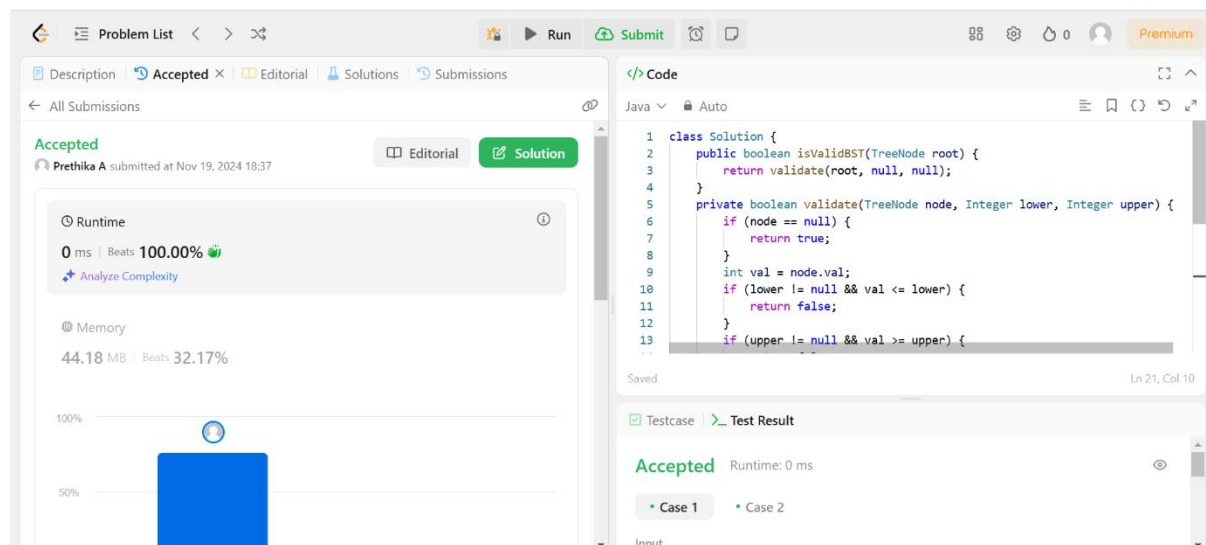
    private boolean validate(TreeNode node, Integer lower, Integer upper) {
        if (node == null) {
            return true;
        }
        int val = node.val;
        if (lower != null && val <= lower) {
            return false;
        }
        if (upper != null && val >= upper) {
           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

```

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

```

Output



The screenshot displays a code editor interface for a Java solution. The left sidebar shows the 'Accepted' status, the user 'Prethika A' submitted on Nov 19, 2024 at 18:37, and performance metrics: Runtime 0 ms (Beats 100.00%) and Memory 44.18 MB (Beats 32.17%). The main editor shows the following Java code:

```

1 class Solution {
2     public boolean isValidBST(TreeNode root) {
3         return validate(root, null, null);
4     }
5     private boolean validate(TreeNode node, Integer lower, Integer upper) {
6         if (node == null) {
7             return true;
8         }
9         int val = node.val;
10        if (lower != null && val <= lower) {
11            return false;
12        }
13        if (upper != null && val >= upper) {

```

Below the code editor, the 'Testcase' tab shows 'Accepted' with a runtime of 0 ms. The 'Case 1' tab is selected, showing the input field.

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

The screenshot displays the LeetCode submission interface for the problem "Longest Substring Without Repeating Characters". The submission is marked as "Accepted" and was submitted by "Prethika A" on Nov 19, 2024, at 18:28. The runtime is 6 ms, which beats 70.21% of other submissions, and the memory usage is 44.67 MB, which beats 46.01%. The code is written in Java and implements a sliding window algorithm using a HashSet to track 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);
        }
    }
}

```

The test result shows that the solution is "Accepted" with a runtime of 0 ms. The test cases are Case 1, Case 2, and Case 3.

Time Complexity : $O(n)$

