DSA Practice

Date: Nov 19

Problems:

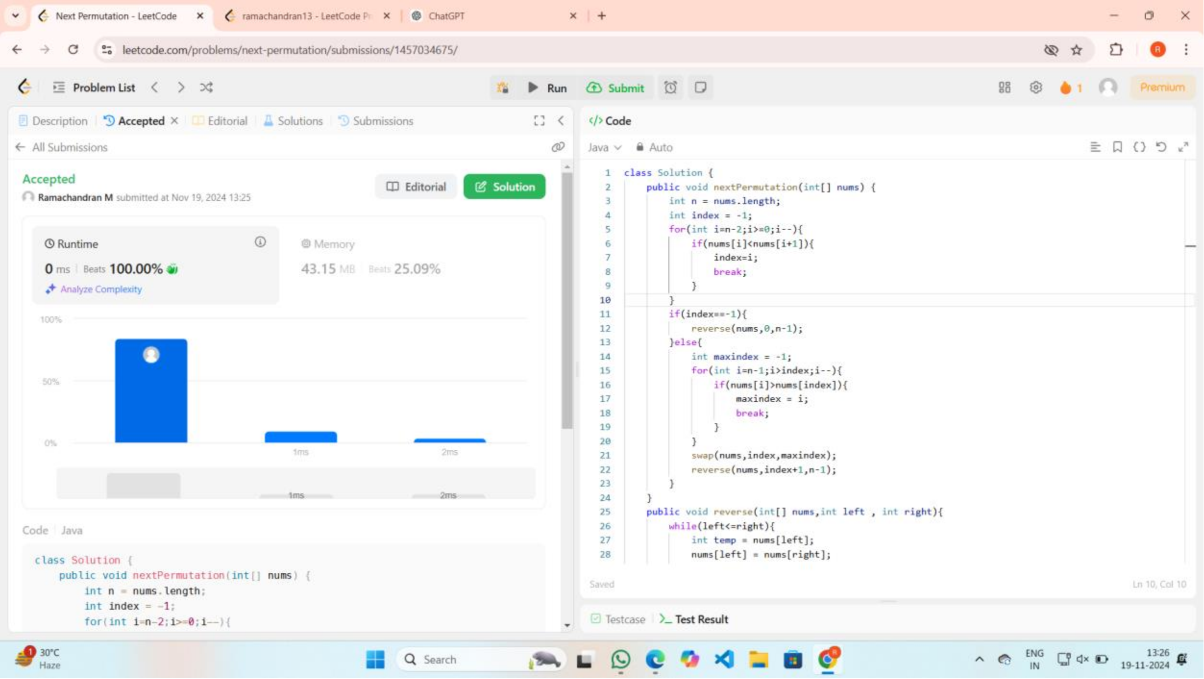
Problems of the day: leetcode problems :

1. next permutation
2. Spiral matrix
3. Longest substring without repeating characters
4. Remove linked list elements
5. Palindrome linked list

Problems of the day: leetcode problems :

1. Minimum path sum
2. Validate binary search tree
3. Word ladder
4. Word ladder -II
5. Course schedule
6. Design tic tac toe

Problem1:



Accepted

Ramachandran M submitted at Nov 19, 2024 13:25

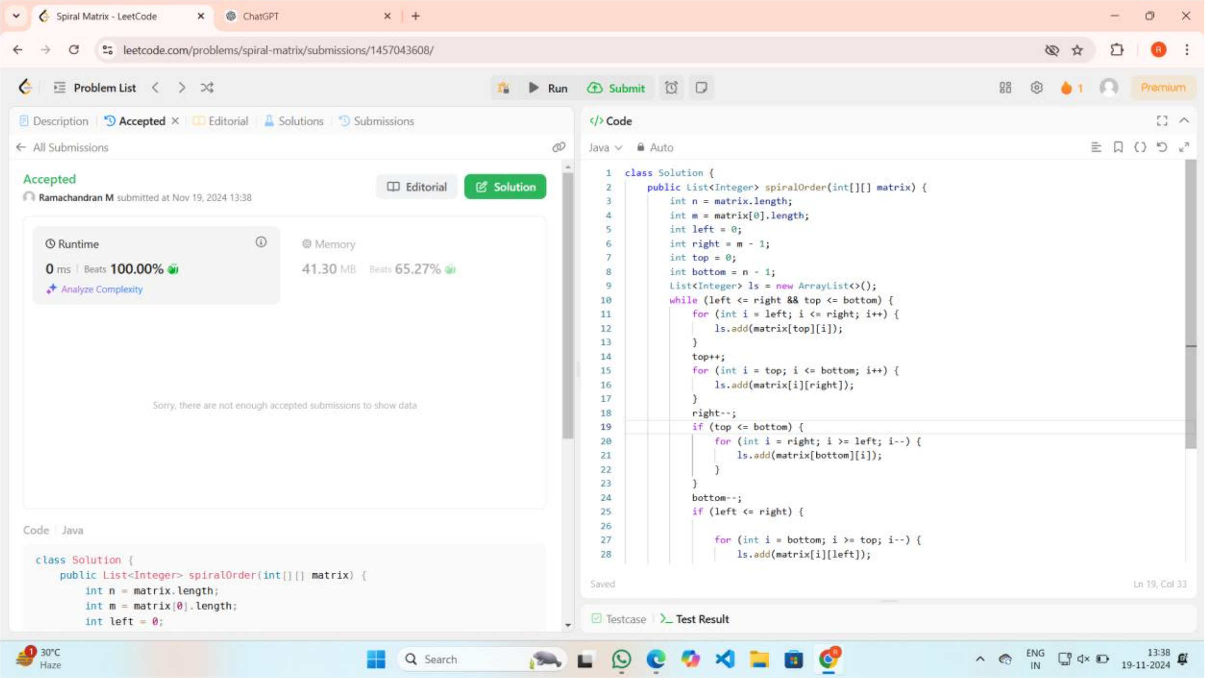
Runtime: 0 ms | Beats 100.00%

Memory: 43.15 MB | Beats 25.09%

```java
class Solution {
    public void nextPermutation(int[] nums) {
        int n = nums.length;
        int index = -1;
        for(int i=n-2;i>=0;i--){
            if(nums[i]<nums[i+1]){
                index=i;
                break;
            }
        }
        if(index==-1){
            reverse(nums,0,n-1);
        }else{
            int maxindex = -1;
            for(int i=n-1;i>index;i--){
                if(nums[i]>nums[index]){
                    maxindex = i;
                    break;
                }
            }
            swap(nums,index,maxindex);
            reverse(nums,index+1,n-1);
        }
    }
    public void reverse(int[] nums,int left , int right){
        while(left<=right){
            int temp = nums[left];
            nums[left] = nums[right];
```

```java
class Solution {
    public void nextPermutation(int[] nums) {
        int n = nums.length;
        int index = -1;
        for(int i=n-2;i>=0;i--){
```

Problem2:



Accepted

Ramachandran M submitted at Nov 19, 2024 13:38

Runtime: 0 ms | Beats 100.00%

Memory: 41.30 MB | Beats 65.27%

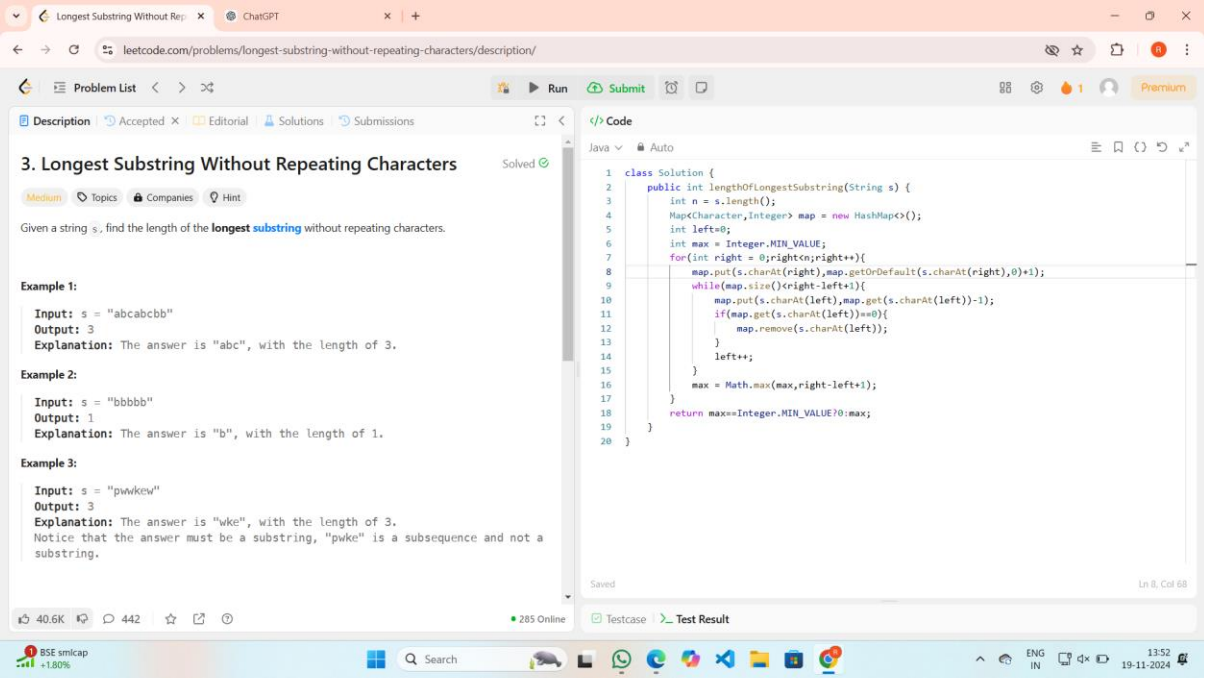
Sorry, there are not enough accepted submissions to show data

```java
class Solution {
    public List<Integer> spiralOrder(int[][] matrix) {
        int n = matrix.length;
        int m = matrix[0].length;
        int left = 0;
        int right = m - 1;
        int top = 0;
        int bottom = n - 1;
        List<Integer> ls = new ArrayList<>();
        while (left <= right && top <= bottom) {
            for (int i = left; i <= right; i++) {
                ls.add(matrix[top][i]);
            }
            top++;
            for (int i = top; i <= bottom; i++) {
                ls.add(matrix[i][right]);
            }
            right--;
            if (top <= bottom) {
                for (int i = right; i >= left; i--) {
                    ls.add(matrix[bottom][i]);
                }
            }
            bottom--;
            if (left <= right) {

                for (int i = bottom; i >= top; i--) {
                    ls.add(matrix[i][left]);
```

```java
class Solution {
    public List<Integer> spiralOrder(int[][] matrix) {
        int n = matrix.length;
        int m = matrix[0].length;
        int left = 0;
```

Problem3:



3. Longest Substring Without Repeating Characters

Given a string `s`, find the length of the **longest substring** without repeating characters.

Example 1:

```
Input: s = "abcabcbb"
Output: 3
Explanation: The answer is "abc", with the length of 3.
```

Example 2:

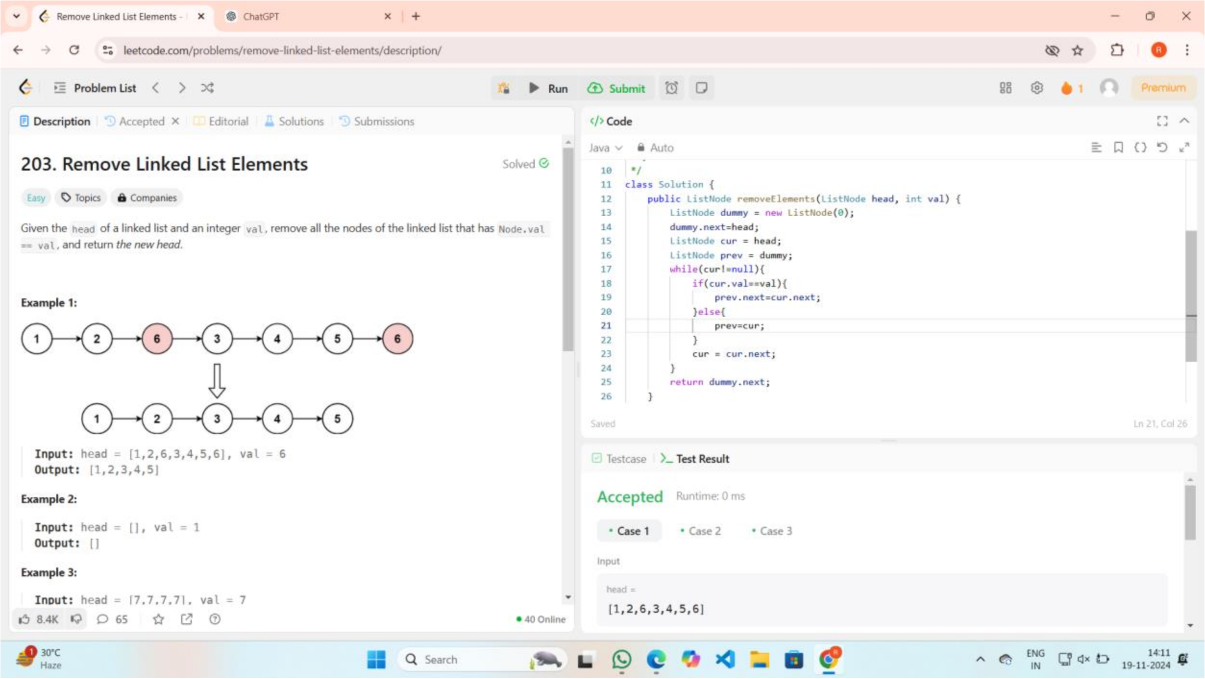
```
Input: s = "bbbbb"
Output: 1
Explanation: The answer is "b", with the length of 1.
```

Example 3:

```
Input: s = "pwwkew"
Output: 3
Explanation: The answer is "wke", with the length of 3.
Notice that the answer must be a substring, "pwke" is a subsequence and not a
substring.
```

```java
class Solution {
    public int lengthOfLongestSubstring(String s) {
        int n = s.length();
        Map<Character,Integer> map = new HashMap<>();
        int left=0;
        int max = Integer.MIN_VALUE;
        for(int right = 0;right<n;right++){
            map.put(s.charAt(right),map.getOrDefault(s.charAt(right),0)+1);
            while(map.size()<right-left+1){
                map.put(s.charAt(left),map.get(s.charAt(left))-1);
                if(map.get(s.charAt(left))==0){
                    map.remove(s.charAt(left));
                }
                left++;
            }
            max = Math.max(max,right-left+1);
        }
        return max==Integer.MIN_VALUE?0:max;
    }
}
```

Problem4:



203. Remove Linked List Elements

Given the `head` of a linked list and an integer `val`, remove all the nodes of the linked list that has `Node.val == val`, and return *the new head.*

Example 1:

```
Input: head = [1,2,6,3,4,5,6], val = 6
Output: [1,2,3,4,5]
```

Example 2:

```
Input: head = [], val = 1
Output: []
```

Example 3:

```
Input: head = [7,7,7,7], val = 7
```

```java
 */
class Solution {
    public ListNode removeElements(ListNode head, int val) {
        ListNode dummy = new ListNode(0);
        dummy.next=head;
        ListNode cur = head;
        ListNode prev = dummy;
        while(cur!=null){
            if(cur.val==val){
                prev.next=cur.next;
            }else{
                prev=cur;
            }
            cur = cur.next;
        }
        return dummy.next;
    }
```

Saved   Ln 21, Col 26

Testcase | Test Result

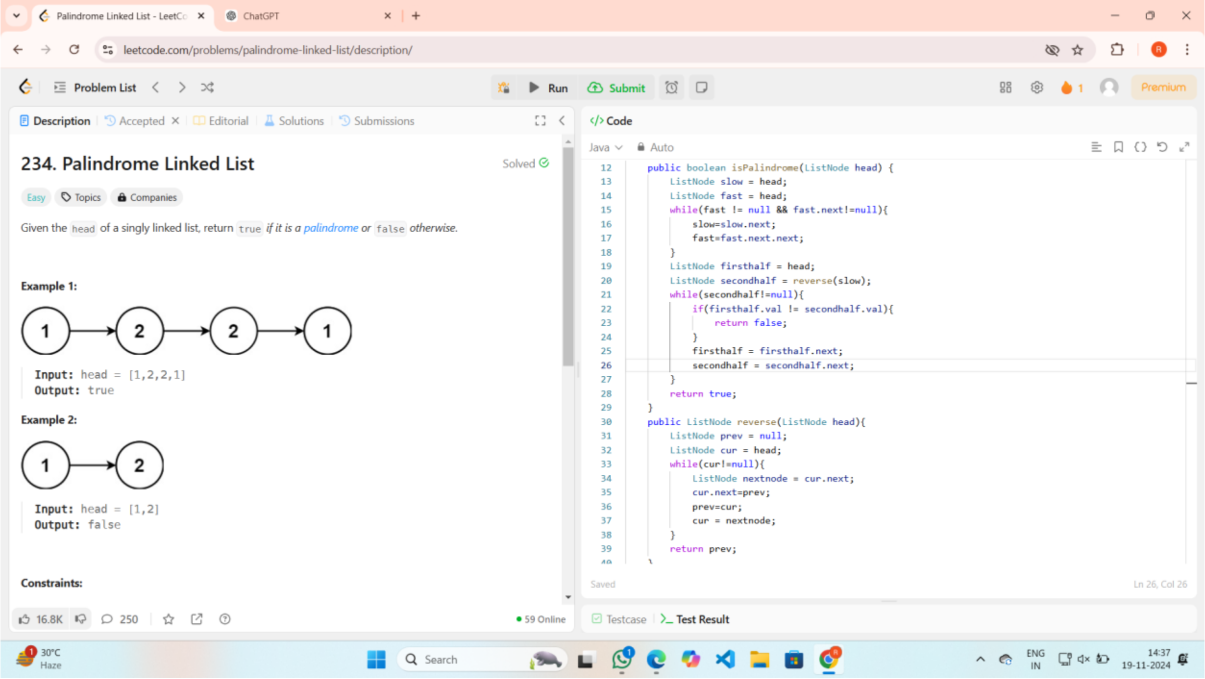
Accepted   Runtime: 0 ms

Case 1 · Case 2 · Case 3

Input

head =
[1,2,6,3,4,5,6]

Problem5:



234. Palindrome Linked List

Given the `head` of a singly linked list, return `true` *if it is a palindrome or* `false` *otherwise.*

Example 1:

```
Input: head = [1,2,2,1]
Output: true
```

Example 2:

```
Input: head = [1,2]
Output: false
```

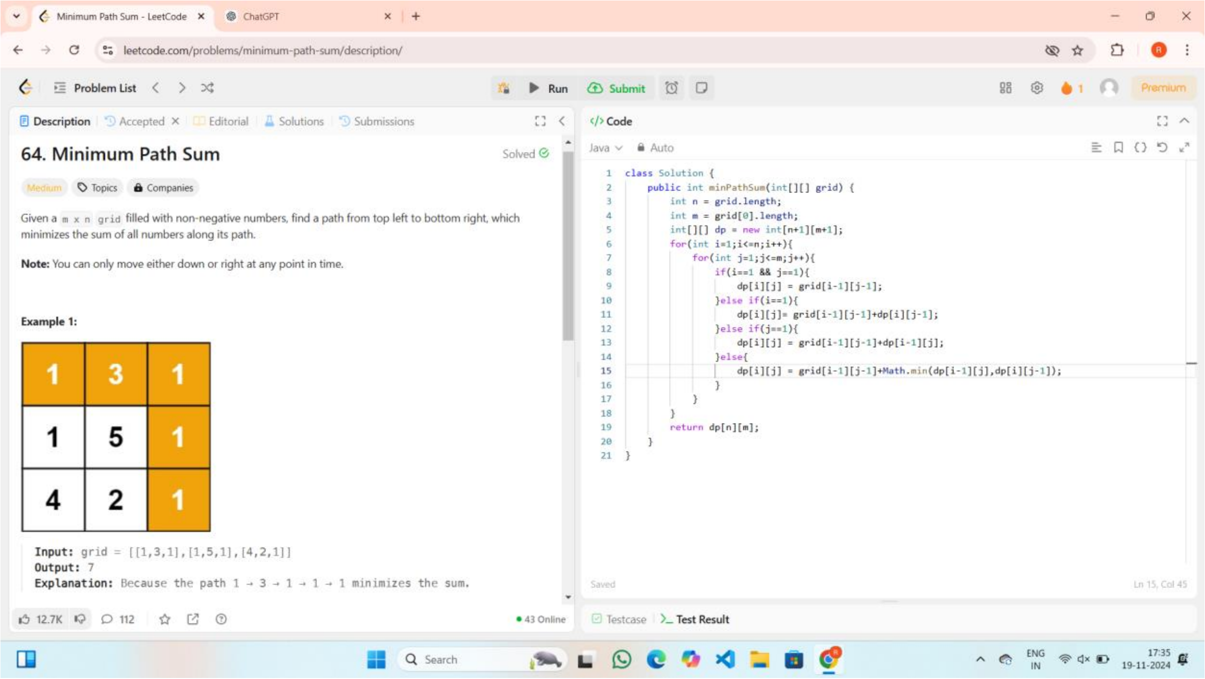
Constraints:

```java
    public boolean isPalindrome(ListNode head) {
        ListNode slow = head;
        ListNode fast = head;
        while(fast != null && fast.next!=null){
            slow=slow.next;
            fast=fast.next.next;
        }
        ListNode firsthalf = head;
        ListNode secondhalf = reverse(slow);
        while(secondhalf!=null){
            if(firsthalf.val != secondhalf.val){
                return false;
            }
            firsthalf = firsthalf.next;
            secondhalf = secondhalf.next;
        }
        return true;
    }
    public ListNode reverse(ListNode head){
        ListNode prev = null;
        ListNode cur = head;
        while(cur!=null){
            ListNode nextnode = cur.next;
            cur.next=prev;
            prev=cur;
            cur = nextnode;
        }
        return prev;
```

Saved   Ln 26, Col 26

Testcase | Test Result

Problem6:



64. Minimum Path Sum

Given a `m x n` `grid` filled with non-negative numbers, find a path from top left to bottom right, which minimizes the sum of all numbers along its path.

Note: You can only move either down or right at any point in time.

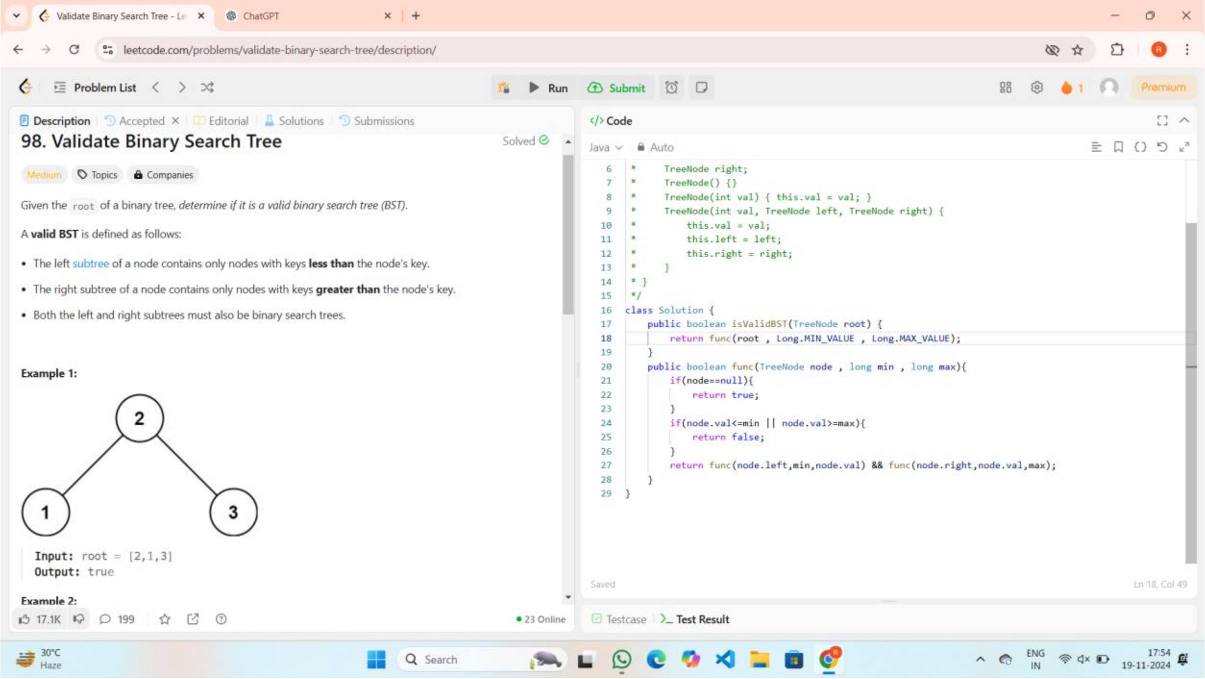
Example 1:

1	3	1
1	5	1
4	2	1

```
Input: grid = [[1,3,1],[1,5,1],[4,2,1]]
Output: 7
Explanation: Because the path 1 → 3 → 1 → 1 → 1 minimizes the sum.
```

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

Problem7:



98. Validate Binary Search Tree

Given the `root` of a binary tree, *determine if it is a valid binary search tree (BST).*

A **valid BST** is defined as follows:

- The left subtree of a node contains only nodes with keys **less than** the node's key.
- The right subtree of a node contains only nodes with keys **greater than** the node's key.
- Both the left and right subtrees must also be binary search trees.

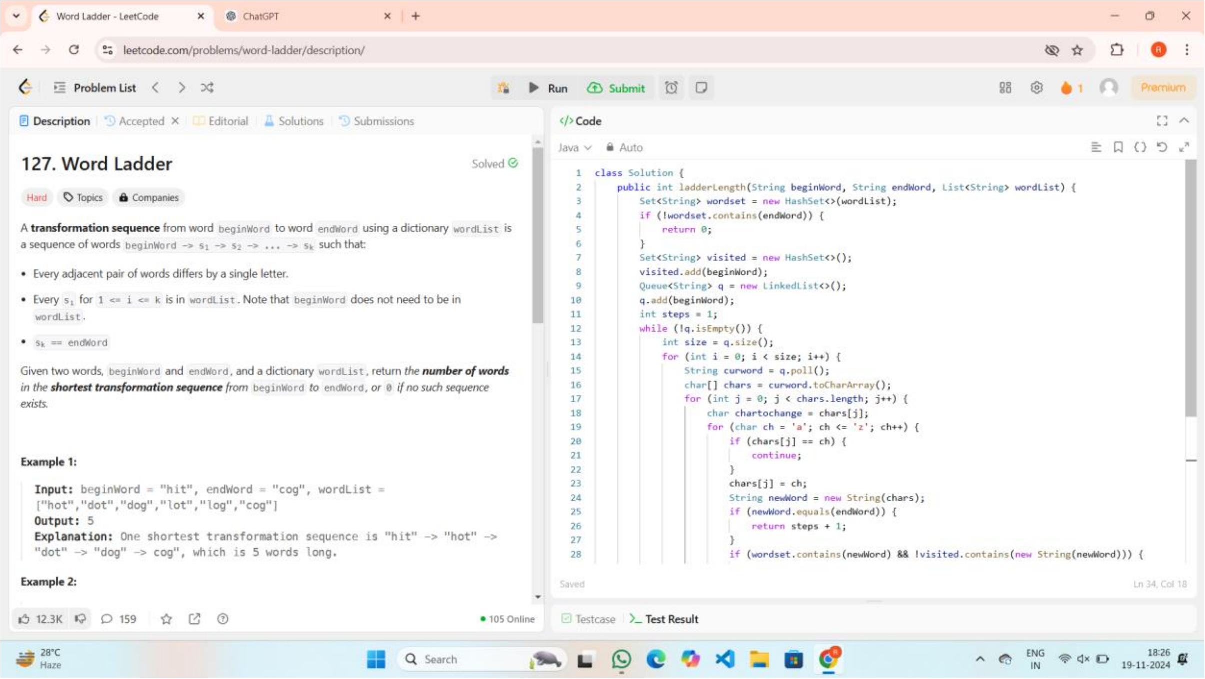
Example 1:

```
Input: root = [2,1,3]
Output: true
```

Example 2:

```java
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
class Solution {
    public boolean isValidBST(TreeNode root) {
        return func(root , Long.MIN_VALUE , Long.MAX_VALUE);
    }
    public boolean func(TreeNode node , long min , long max){
        if(node==null){
            return true;
        }
        if(node.val<=min || node.val>=max){
            return false;
        }
        return func(node.left,min,node.val) && func(node.right,node.val,max);
    }
}
```

Problem8:



127. Word Ladder

Solved ✓

Hard | Topics | Companies

A **transformation sequence** from word `beginWord` to word `endWord` using a dictionary `wordList` is a sequence of words `beginWord -> s1 -> s2 -> ... -> sk` such that:

- Every adjacent pair of words differs by a single letter.
- Every `si` for `1 <= i <= k` is in `wordList`. Note that `beginWord` does not need to be in `wordList`.
- `sk == endWord`

Given two words, `beginWord` and `endWord`, and a dictionary `wordList`, return *the **number of words** in the **shortest transformation sequence** from* `beginWord` *to* `endWord`*, or* `0` *if no such sequence exists.*

Example 1:

```
Input: beginWord = "hit", endWord = "cog", wordList =
["hot","dot","dog","lot","log","cog"]
Output: 5
Explanation: One shortest transformation sequence is "hit" -> "hot" ->
"dot" -> "dog" -> cog", which is 5 words long.
```

Example 2:

```java
class Solution {
    public int ladderLength(String beginWord, String endWord, List<String> wordList) {
        Set<String> wordset = new HashSet<>(wordList);
        if (!wordset.contains(endWord)) {
            return 0;
        }
        Set<String> visited = new HashSet<>();
        visited.add(beginWord);
        Queue<String> q = new LinkedList<>();
        q.add(beginWord);
        int steps = 1;
        while (!q.isEmpty()) {
            int size = q.size();
            for (int i = 0; i < size; i++) {
                String curword = q.poll();
                char[] chars = curword.toCharArray();
                for (int j = 0; j < chars.length; j++) {
                    char chartochange = chars[j];
                    for (char ch = 'a'; ch <= 'z'; ch++) {
                        if (chars[j] == ch) {
                            continue;
                        }
                        chars[j] = ch;
                        String newWord = new String(chars);
                        if (newWord.equals(endWord)) {
                            return steps + 1;
                        }
                        if (wordset.contains(newWord) && !visited.contains(new String(newWord))) {
```

Problem9: