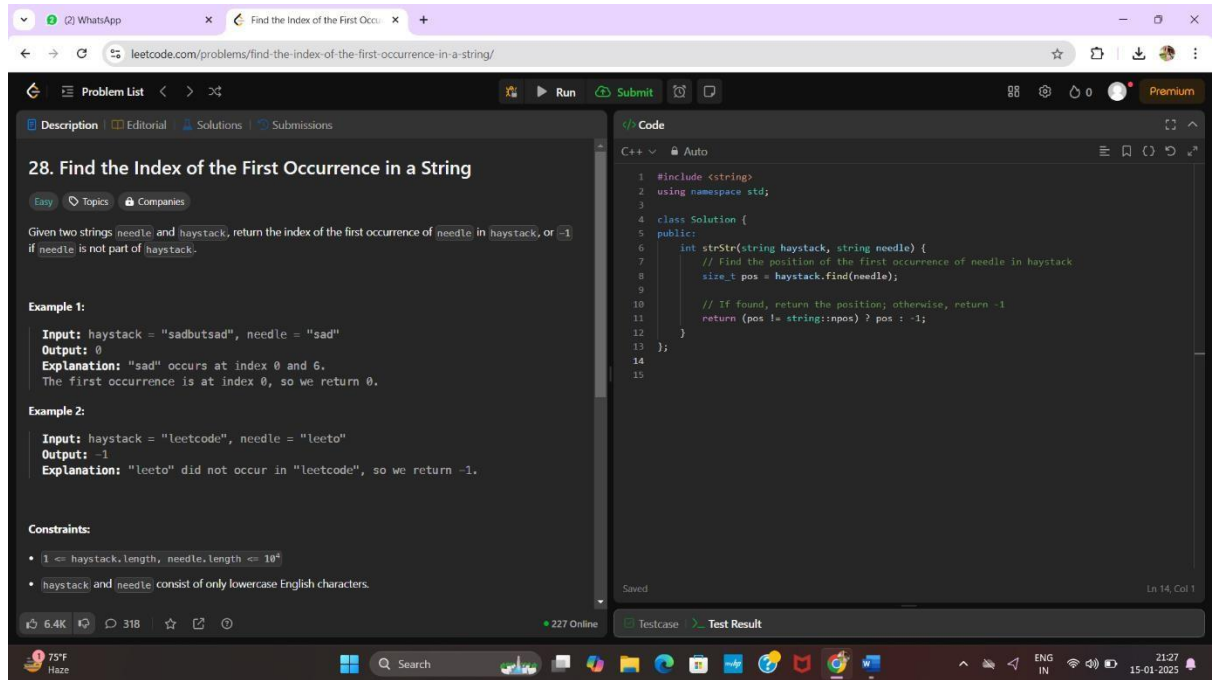


DAA HOLIDAY ASSIGNMENT

- 1) find-the-index-of-the-first-occurrence-in-a-string

<https://leetcode.com/problems/find-the-index-of-the-first-occurrence-in-a-string/>



28. Find the Index of the First Occurrence in a String

Given two strings `needle` and `haystack`, return the index of the first occurrence of `needle` in `haystack`, or `-1` if `needle` is not part of `haystack`.

Example 1:
Input: `haystack = "sadbutsad"`, `needle = "sad"`
Output: `0`
Explanation: "sad" occurs at index 0 and 6. The first occurrence is at index 0, so we return 0.

Example 2:
Input: `haystack = "leetcode"`, `needle = "leeto"`
Output: `-1`
Explanation: "leeto" did not occur in "leetcode", so we return -1.

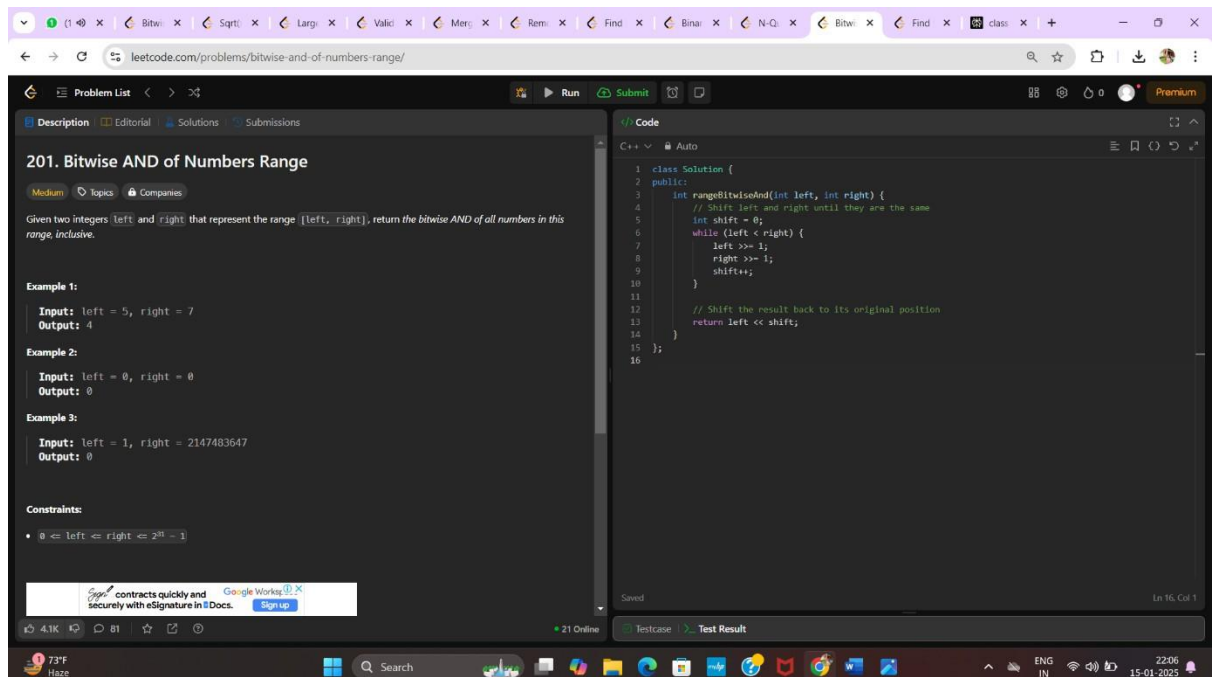
Constraints:

- $1 \leq \text{haystack.length}, \text{needle.length} \leq 10^4$
- `haystack` and `needle` consist of only lowercase English characters.

```
1 #include <string>
2 using namespace std;
3
4 class Solution {
5 public:
6     int strStr(string haystack, string needle) {
7         // Find the position of the first occurrence of needle in haystack
8         size_t pos = haystack.find(needle);
9
10        // If found, return the position; otherwise, return -1
11        return (pos != string::npos) ? pos : -1;
12    }
13 };
14
15
```

- 2) Bitwise AND of numberrange

<https://leetcode.com/problems/bitwise-and-of-numbers-range/description/>



201. Bitwise AND of Numbers Range

Given two integers `left` and `right` that represent the range `[left, right]`, return the *bitwise AND* of all numbers in this range, inclusive.

Example 1:
Input: `left = 5`, `right = 7`
Output: `4`

Example 2:
Input: `left = 0`, `right = 0`
Output: `0`

Example 3:
Input: `left = 1`, `right = 2147483647`
Output: `0`

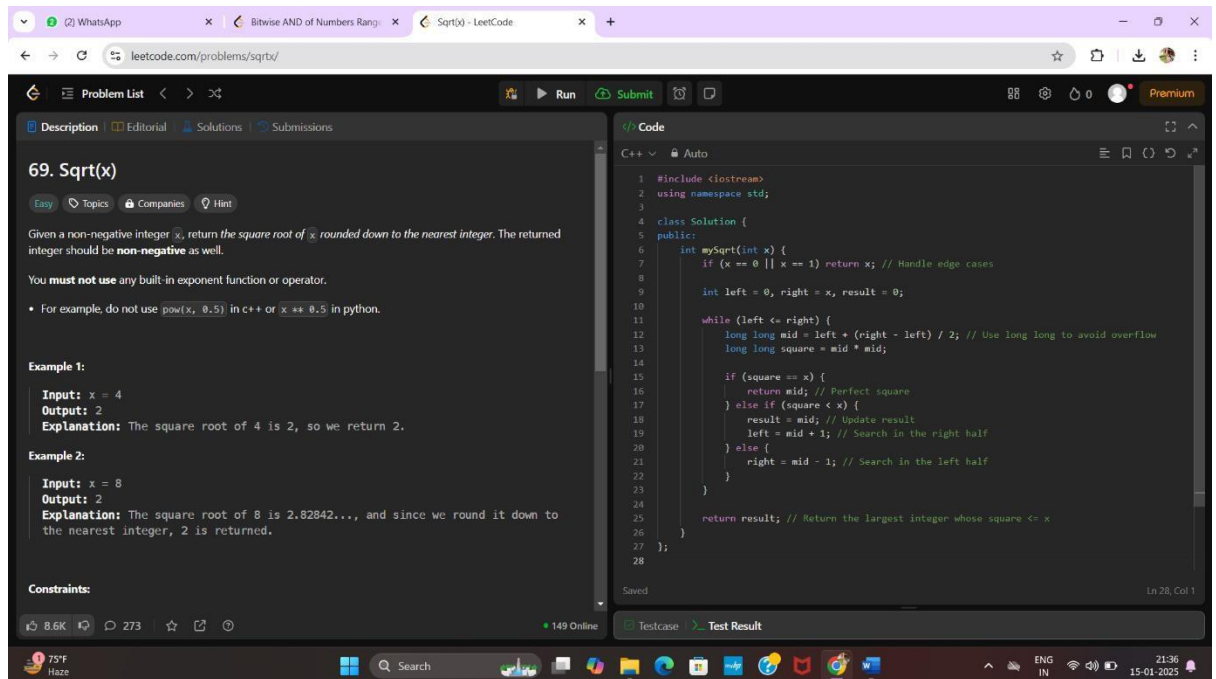
Constraints:

- $0 \leq \text{left} \leq \text{right} \leq 2^{31} - 1$

```
1 class Solution {
2 public:
3     int rangeBitwiseAnd(int left, int right) {
4         // Shift left and right until they are the same
5         int shift = 0;
6         while (left < right) {
7             left >>= 1;
8             right >>= 1;
9             shift++;
10        }
11
12        // Shift the result back to its original position
13        return left << shift;
14    }
15 };
16
```

3) SQRT(X)

<https://leetcode.com/problems/sqrtx>



The screenshot shows the LeetCode interface for problem 69, "Sqrt(x)". The problem description states: "Given a non-negative integer x , return the square root of x rounded down to the nearest integer. The returned integer should be non-negative as well. You must not use any built-in exponent function or operator. For example, do not use `pow(x, 0.5)` in c++ or `x ** 0.5` in python."

Example 1:
Input: $x = 4$
Output: 2
Explanation: The square root of 4 is 2, so we return 2.

Example 2:
Input: $x = 8$
Output: 2
Explanation: The square root of 8 is 2.8284..., and since we round it down to the nearest integer, 2 is returned.

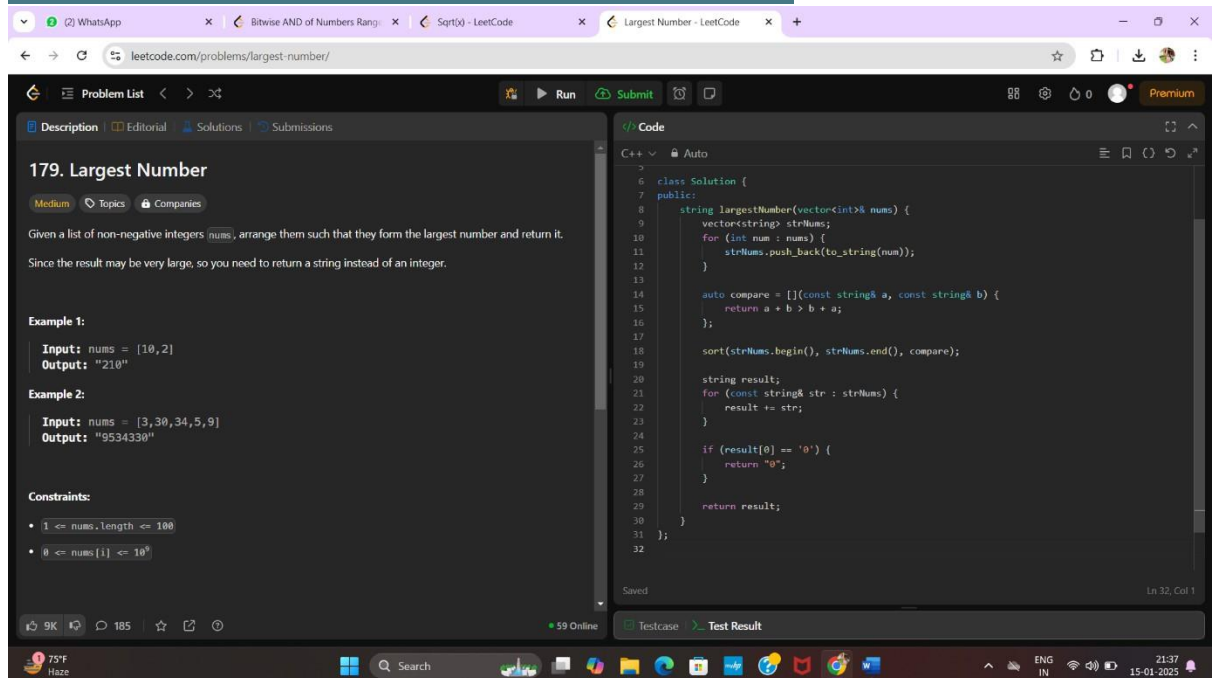
Constraints:
• $0 \leq x \leq 2^{31} - 1$

The C++ code on the right implements a binary search solution:

```
1 #include <iostream>
2 using namespace std;
3
4 class Solution {
5 public:
6     int mySqrt(int x) {
7         if (x == 0 || x == 1) return x; // Handle edge cases
8
9         int left = 0, right = x, result = 0;
10
11         while (left <= right) {
12             long long mid = left + (right - left) / 2; // Use long long to avoid overflow
13             long long square = mid * mid;
14
15             if (square == x) {
16                 return mid; // Perfect square
17             } else if (square < x) {
18                 result = mid; // Update result
19                 left = mid + 1; // Search in the right half
20             } else {
21                 right = mid - 1; // Search in the left half
22             }
23         }
24
25         return result; // Return the largest integer whose square <= x
26     }
27 };
28
```

4) Largest Number

<https://leetcode.com/problems/largest-number/description/>



The screenshot shows the LeetCode interface for problem 179, "Largest Number". The problem description states: "Given a list of non-negative integers `nums`, arrange them such that they form the largest number and return it. Since the result may be very large, so you need to return a string instead of an integer."

Example 1:
Input: `nums = [10, 2]`
Output: "210"

Example 2:
Input: `nums = [3, 30, 34, 5, 9]`
Output: "9534330"

Constraints:
• $1 \leq \text{nums.length} \leq 100$
• $0 \leq \text{nums}[i] \leq 10^9$

The C++ code on the right implements a solution using a custom comparator and sorting:

```
1
2
3 class Solution {
4 public:
5     string largestNumber(vector<int> & nums) {
6         vector<string> strNums;
7         for (int num : nums) {
8             strNums.push_back(to_string(num));
9         }
10
11         auto compare = [](const string& a, const string& b) {
12             return a + b > b + a;
13         };
14
15         sort(strNums.begin(), strNums.end(), compare);
16
17         string result;
18         for (const string& str : strNums) {
19             result += str;
20         }
21
22         if (result[0] == '0') {
23             return "0";
24         }
25
26         return result;
27     }
28 };
29
30
31
32
```

5) Valid Parenthesis

<https://leetcode.com/problems/valid-parentheses/>

20. Valid Parentheses

Given a string `s` containing just the characters `'('`, `'{'`, `'['`, `')'`, `'}'`, and `']'`, determine if the input string is valid.

An input string is valid if:

1. Open brackets must be closed by the same type of brackets.
2. Open brackets must be closed in the correct order.
3. Every close bracket has a corresponding open bracket of the same type.

Example 1:

Input: `s = "()"`
Output: `true`

Example 2:

Input: `s = "()[]{}"`
Output: `true`

Example 3:

Input: `s = "(]"`
Output: `false`

Example 4:

Input: `s = "([)]"`
Output: `false`

Example 5:

Input: `s = "{[]}"`
Output: `true`

```
1 #include <iostream>
2 #include <stack>
3 #include <string>
4 using namespace std;
5
6 class Solution {
7 public:
8     bool isValid(string s) {
9         stack<char> stack;
10
11         for (char c : s) {
12             // If the character is an opening bracket, push it into the stack
13             if (c == '(' || c == '[' || c == '{') {
14                 stack.push(c);
15             }
16             // If the character is a closing bracket, check if it matches the top of the stack
17             else if (c == ')' || c == ']' || c == '}') {
18                 if (stack.empty()) {
19                     return false; // No opening bracket to match with
20                 }
21                 char top = stack.top();
22                 stack.pop();
23
24                 // Check if the top of the stack matches the corresponding opening bracket
25                 if ((c == ')' && top != '(') || (c == ']' && top != '[') || (c == '}' && top != '{')) {
26                     return false;
27                 }
28             }
29         }
30
31         // If the stack is empty, all brackets were matched, otherwise return false
32         return stack.empty();
33     }
34 };
35
```

6) Merge Two Sorted Lists

<https://leetcode.com/problems/merge-two-sorted-lists/>

21. Merge Two Sorted Lists

You are given the heads of two sorted linked lists `list1` and `list2`.

Merge the two lists into one sorted list. The list should be made by splicing together the nodes of the first two lists.

Return the head of the merged linked list.

Example 1:

Input: `list1 = [1,2,4]`, `list2 = [1,3,4]`
Output: `[1,1,2,3,4,4]`

Example 2:

Input: `list1 = []`, `list2 = []`
Output: `[]`

Example 3:

Input: `list1 = [0]`, `list2 = [0]`
Output: `[0]`

```
1 #include <iostream>
2 #include <vector>
3 // Include the listNode header
4 using namespace std;
5
6 class Solution {
7 public:
8     listNode* mergeTwoLists(listNode* list1, listNode* list2) {
9         listNode* dummy = new listNode(0); // Create a dummy node to start the merged list
10         listNode* current = dummy; // Pointer to build the new list
11
12         // Traverse both lists
13         while (list1 != nullptr && list2 != nullptr) {
14             if (list1->val < list2->val) {
15                 current->next = list1; // Attach list1 node to merged list
16                 list1 = list1->next; // Move the list1 pointer forward
17             } else {
18                 current->next = list2; // Attach list2 node to merged list
19                 list2 = list2->next; // Move the list2 pointer forward
20             }
21             current = current->next; // Move the current pointer forward in the merged list
22         }
23
24         // If there are remaining nodes in list1 or list2, attach them
25         if (list1 != nullptr) {
26             current->next = list1;
27         } else if (list2 != nullptr) {
28             current->next = list2;
29         }
30
31         return dummy->next; // Return the merged list starting from the first node
32     }
33 };
34
35 // Helper function to create a linked list from a vector
36 listNode* createList(const vector<int>& nums) {
37     listNode* head = nullptr;
38     listNode* current = nullptr;
39     for (int num : nums) {
40         listNode* newNode = new listNode(num);
41         if (head == nullptr) {
42             head = newNode;
43         } else {
44             current->next = newNode;
45         }
46         current = newNode;
47     }
48     return head;
49 }
```

7) Remove Duplicates from sorted list

<https://leetcode.com/problems/remove-duplicates-from-sorted-list/>

83. Remove Duplicates from Sorted List

Given the *head* of a sorted linked list, delete all duplicates such that each element appears only once. Return the linked list *sorted* as well.

Example 1:

Input: *head* = [1,1,2]
Output: [1,2]

Example 2:

Input: *head* = [1,1,2,3,3]
Output: [1,2,3]

```
1 #include <iostream>
2 using namespace std;
3 // Assume ListNode structure is already defined and precompiled.
4
5 class Solution {
6 public:
7     ListNode* deleteDuplicates(ListNode* head) {
8         ListNode* current = head;
9         while (current != nullptr && current->next != nullptr) {
10             if (current->val == current->next->val) {
11                 // Skip the duplicate node
12                 current->next = current->next->next;
13             } else {
14                 // Move to the next node
15                 current = current->next;
16             }
17         }
18         return head;
19     }
20 };
21
22 // Helper function to print the linked list
23 void printList(ListNode* head) {
24     while (head != nullptr) {
25         cout << head->val << " ";
26         head = head->next;
27     }
28     cout << endl;
29 }
```

8) Find peak Element

<https://leetcode.com/problems/find-peak-element/>

162. Find Peak Element

A peak element is an element that is strictly greater than its neighbors.

Given a 0-indexed integer array *nums*, find a peak element, and return its index. If the array contains multiple peaks, return the index to *any* of the peaks.

You may imagine that *nums*[-1] = *nums*[*n*] = -∞. In other words, an element is always considered to be strictly greater than a neighbor that is outside the array.

You must write an algorithm that runs in *O*(log *n*) time.

Example 1:

Input: *nums* = [1,2,3,1]
Output: 2
Explanation: 3 is a peak element and your function should return the index number 2.

Example 2:

Input: *nums* = [1,2,1,3,5,6,4]
Output: 5
Explanation: Your function can return either index number 1 where the peak element is 2, or index number 5 where the peak element is 6.

Constraints:

- 1 ≤ *nums*.length ≤ 1000

```
1 #include <iostream>
2 #include <vector>
3 using namespace std;
4
5 class Solution {
6 public:
7     int findPeakElement(vector<int>& nums) {
8         int left = 0, right = nums.size() - 1;
9
10        while (left < right) {
11            int mid = left + (right - left) / 2;
12
13            if (nums[mid] > nums[mid + 1]) {
14                right = mid;
15            } else {
16                left = mid + 1;
17            }
18        }
19        return left;
20    }
21 };
22
23
24
```

9) Binary Tree Inorder Traversal

<https://leetcode.com/problems/binary-tree-inorder-traversal/>

The screenshot shows the LeetCode interface for problem 94. The description states: "Given the `root` of a binary tree, return the *inorder traversal* of its nodes' values." Example 1 shows an input tree with root 1, left child 3, and right child 2, resulting in an output of [1, 3, 2]. The code editor on the right contains a C++ solution using a recursive helper function to perform the inorder traversal.

```
1 #include <iostream>
2 #include <vector>
3
4 using namespace std;
5
6 class Solution {
7 public:
8     void inorderTraversalHelper(TreeNode* root, vector<int>& result) {
9         if (root == nullptr) {
10             return;
11         }
12         // Traverse left subtree
13         inorderTraversalHelper(root->left, result);
14         // Visit the root node
15         result.push_back(root->val);
16         // Traverse right subtree
17         inorderTraversalHelper(root->right, result);
18     }
19
20     vector<int> inorderTraversal(TreeNode* root) {
21         vector<int> result;
22         inorderTraversalHelper(root, result);
23         return result;
24     }
25 };
26
27 // Testcase
28 // Test Result
```

10) N-Queens

<https://leetcode.com/problems/n-queens/>

The screenshot shows the LeetCode interface for problem 51. The description states: "The *n-queens* puzzle is the problem of placing *n* queens on an *n* x *n* chessboard such that no two queens attack each other. Given an integer *n*, return all distinct solutions to the *n-queens* puzzle. You may return the answer in any order." Example 1 shows a 4x4 chessboard with two distinct solutions. The code editor on the right contains a C++ solution using a recursive backtracking approach to find all valid queen placements.

```
1 #include <vector>
2 #include <string>
3 using namespace std;
4
5 class Solution {
6 public:
7     vector<vector<string>>> solveNQueens(int n) {
8         vector<vector<string>>> solutions;
9         vector<string> board(n, string(n, '.')); // Initialize an empty n x n board
10         vector<int> leftRow(n, 0), upperDiag(2 * n - 1, 0), lowerDiag(2 * n - 1, 0);
11         backtrack(0, n, board, solutions, leftRow, upperDiag, lowerDiag);
12         return solutions;
13     }
14
15 private:
16     void backtrack(int col, int n, vector<string>& board, vector<vector<string>>& solutions,
17                     vector<int>& leftRow, vector<int>& upperDiag, vector<int>& lowerDiag) {
18         if (col == n) {
19             solutions.push_back(board);
20             return;
21         }
22         for (int row = 0; row < n; ++row) {
23             if (leftRow[row] == 0 && upperDiag[row + col] == 0 && lowerDiag[row - col + n - 1] == 0) {
24                 board[row][col] = 'Q';
25                 leftRow[row] = upperDiag[row + col] = lowerDiag[row - col + n - 1] = 1;
26                 backtrack(col + 1, n, board, solutions, leftRow, upperDiag, lowerDiag);
27                 // Undo the current placement
28                 board[row][col] = '.';
29                 leftRow[row] = upperDiag[row + col] = lowerDiag[row - col + n - 1] = 0;
30             }
31         }
32     }
33 };
34
35 // Testcase
36 // Test Result
```

V.RAHUL
2211CS020625