Advanced Pragramming

ASSIGNMENT 09

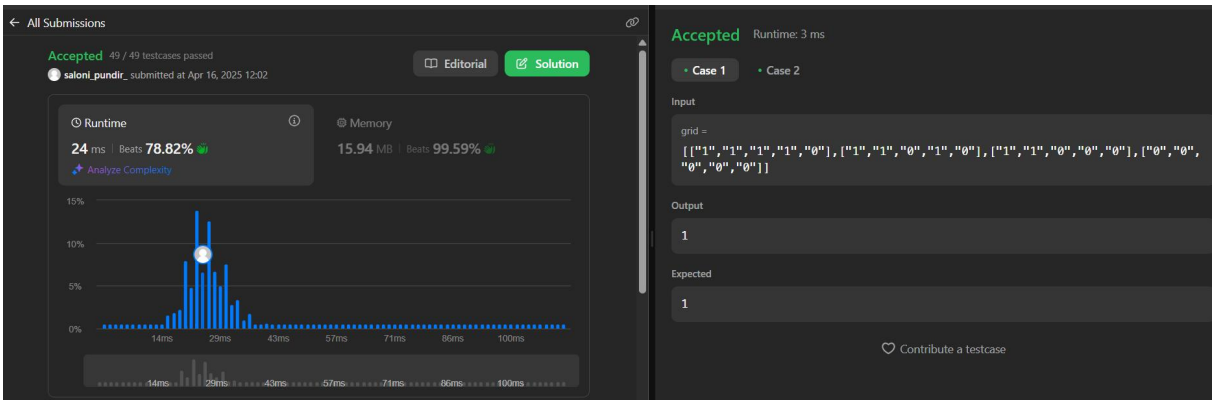
Q1. Number of Islands.

Code:

C++ Auto

```cpp
#include<vector>
using namespace std;
class Solution{
public:
void dfs(vector<vector<char>>&grid,int i,int j){
int m=grid.size(),n=grid[0].size();
if(i<0||j<0||i>=m||j>=n||grid[i][j]=='0')return;
grid[i][j]='0';
dfs(grid,i+1,j);
dfs(grid,i-1,j);
dfs(grid,i,j+1);
dfs(grid,i,j-1);
}
int numIslands(vector<vector<char>>&grid){
int m=grid.size(),n=grid[0].size(),count=0;
for(int i=0;i<m;++i){
for(int j=0;j<n;++j){
if(grid[i][j]=='1'){
++count;
dfs(grid,i,j);
}
}
}
return count;
}
};
```

Output:



Q2. Surrounded Regions.

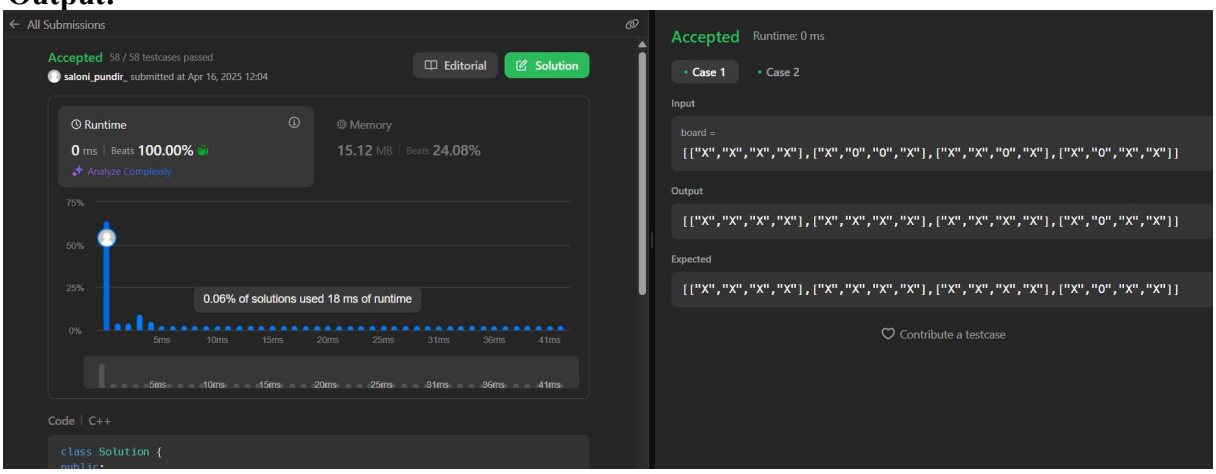
Code:

Code | Testcase | Test Result

C++ Auto

```cpp
class Solution {
public:
    void solve(vector<vector<char>>& board) {
        if (board.empty() || board[0].empty()) return;
        int m = board.size(), n = board[0].size();
        function<void(int, int)> dfs = [&](int i, int j) {
            if (i < 0 || i >= m || j < 0 || j >= n || board[i][j] != 'O') return;
            board[i][j] = 'E';
            dfs(i - 1, j);
            dfs(i + 1, j);
            dfs(i, j - 1);
            dfs(i, j + 1);
        };

        for (int i = 0; i < m; i++) {
            dfs(i, 0);
            dfs(i, n - 1);
        }
        for (int j = 0; j < n; j++) {
            dfs(0, j);
            dfs(m - 1, j);
        }
        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
                if (board[i][j] == 'O') board[i][j] = 'X';
                if (board[i][j] == 'E') board[i][j] = 'O';
            }
        }
    }
```

Output:



Q3. Friend Circles.

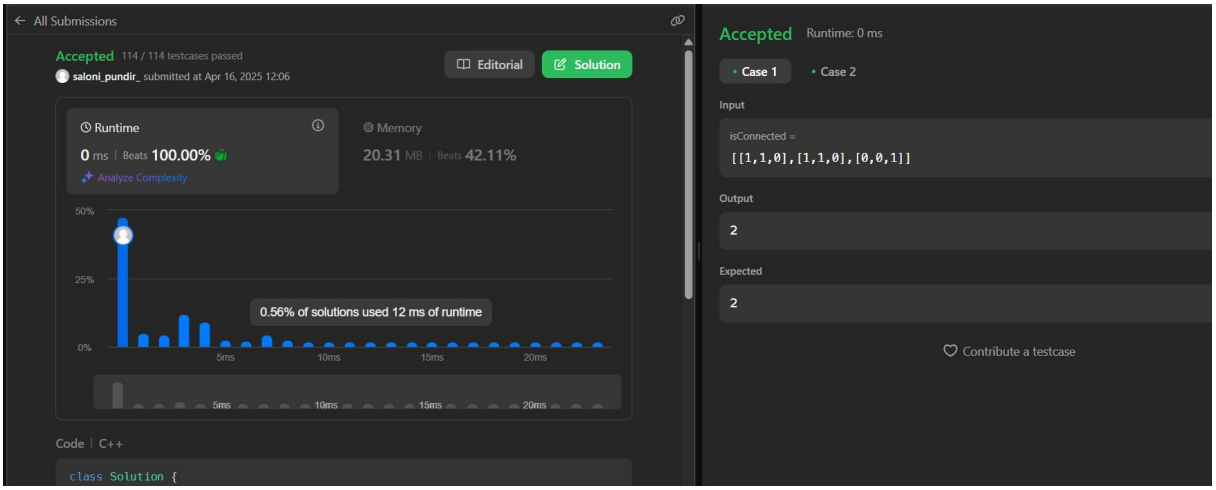
Code:

C++ Auto

```cpp
class Solution {
public:
    int findCircleNum(vector<vector<int>>& isConnected) {
        int n = isConnected.size();
        vector<bool> visited(n, false);
        int provinceCount = 0;

        for (int i = 0; i < n; i++) {
            if (!visited[i]) {
                vector<int> queue;
                queue.push_back(i);
                visited[i] = true;
                bfs1(isConnected, queue, visited);
                provinceCount++;
            }
        }

        return provinceCount;
    }

private:
    void bfs1(vector<vector<int>>& isConnected, vector<int>& queue, vector<bool>& visited) {
        if (queue.empty()) return;

        int city = queue.front();
        queue.erase(queue.begin());

        vector<int> nextQueue;
        for (int neighbor = 0; neighbor < isConnected.size(); neighbor++) {
```

Saved Ln 46, Col 3

Output:

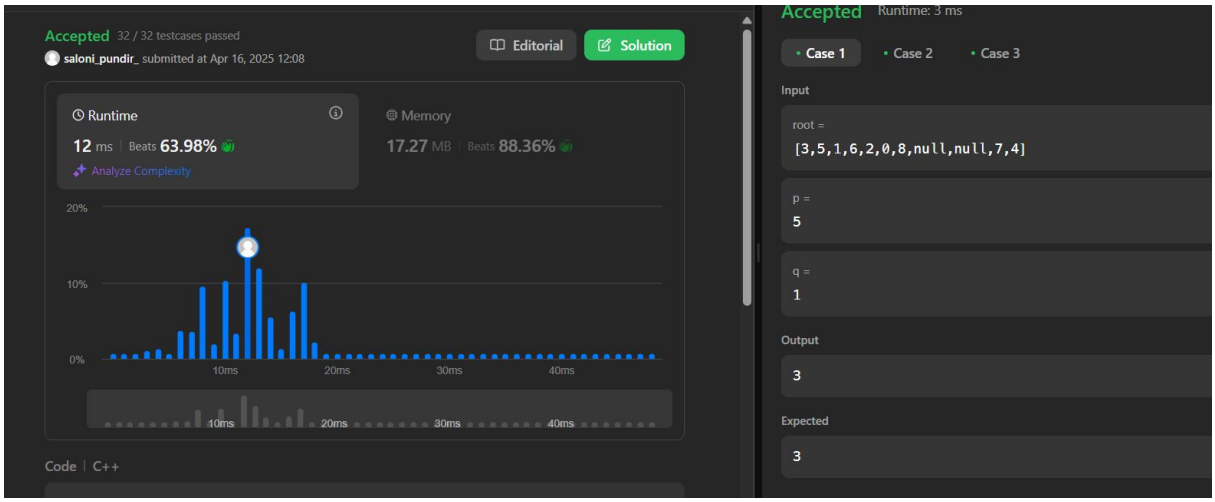


Q4. Lowest Common Ancestor of a Binary Tree.

Code:

```cpp
class Solution {
public:
    TreeNode* lowestCommonAncestor(TreeNode* root, TreeNode* p, TreeNode* q) {
        if (root == nullptr || root == p || root == q) {
            return root;
        }

        TreeNode* left = lowestCommonAncestor(root->left, p, q);
        TreeNode* right = lowestCommonAncestor(root->right, p, q);

        if (left != nullptr && right != nullptr) {
            return root;
        }

        return left != nullptr ? left : right;
    }
};
```

Output:



Assignment

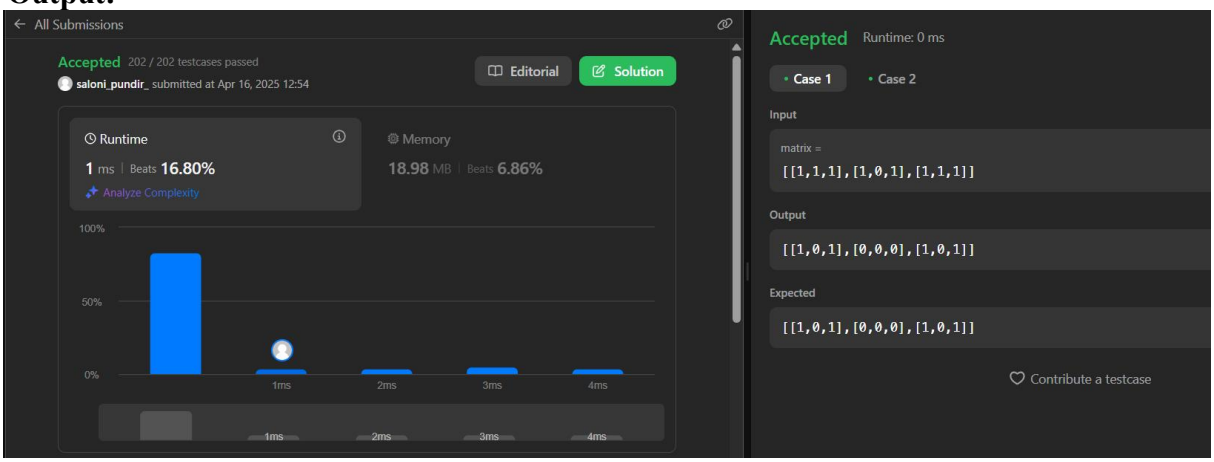
Q1.Set Matrix Zeroes.

Code:

C++ Auto

```cpp
class Solution {
public:
    void setZeroes(vector<vector<int>>& matrix) { int n = matrix.size();
        int m = matrix[0].size();
        vector<vector<int>>v(n,vector<int>(m,0));
        for(int i=0;i<n;i++){
            for(int j=0;j<m;j++){
                if(matrix[i][j]==0){
                    for(int k=0;k<n;k++){
                        v[k][j]=-1;
                    }
                    for(int k=0;k<m;k++){
                        v[i][k]=-1;
                    }
                }
            }
        }
        for(int i=0;i<n;i++){
            for(int j=0;j<m;j++){
                if(v[i][j]==-1)matrix[i][j]=0;
            }
        }
    }
};
```

Output:

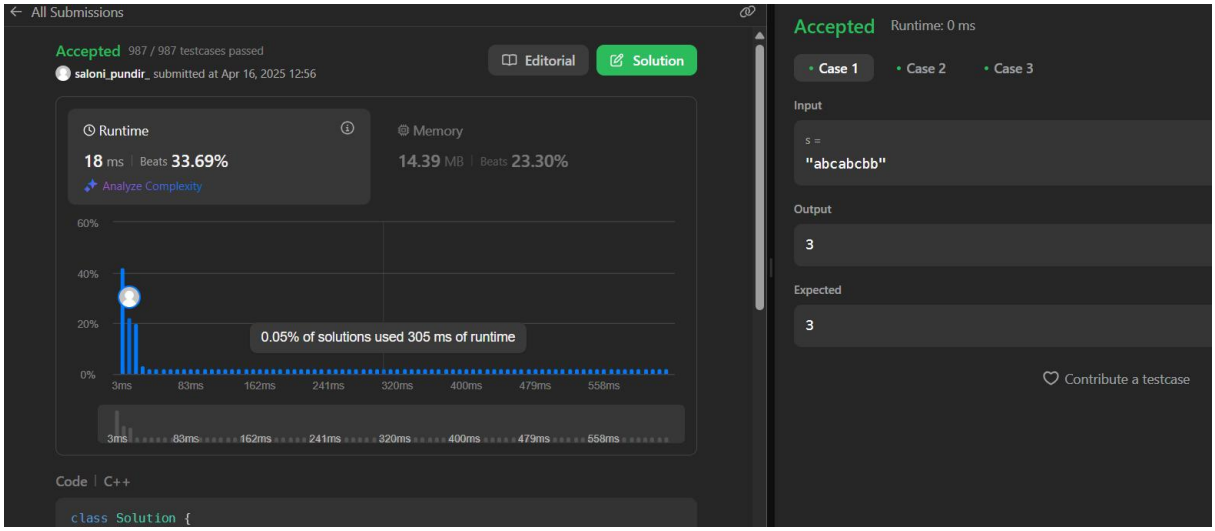


Q2. Longest Substring Without Repeating Characters.

Code:

```cpp
class Solution {
public:
    int lengthOfLongestSubstring(string s) {
        int n = s.length();
        int maxLength = 0;
        unordered_set<char> charSet;
        int left = 0;

        for (int right = 0; right < n; right++) {
            if (charSet.count(s[right]) == 0) {
                charSet.insert(s[right]);
                maxLength = max(maxLength, right - left + 1);
            } else {
                while (charSet.count(s[right])) {
                    charSet.erase(s[left]);
                    left++;
                }
                charSet.insert(s[right]);
            }
        }

        return maxLength;
    }
};
```

Output:



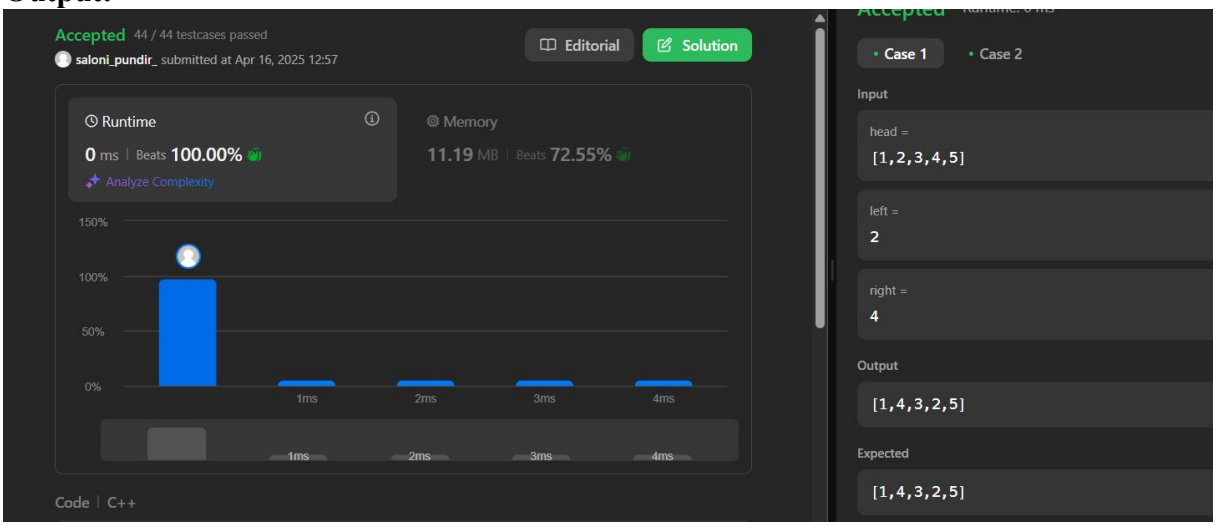
Q3. Reverse Linked List II.

Code:

C++ Auto

```cpp
class Solution {
public:
    ListNode* reverseBetween(ListNode* head, int left, int right) {
        if (!head || left == right) {
            return head;
        }

        ListNode* dummy = new ListNode(0);
        dummy->next = head;
        ListNode* prev = dummy;

        for (int i = 0; i < left - 1; i++) {
            prev = prev->next;
        }

        ListNode* cur = prev->next;

        for (int i = 0; i < right - left; i++) {
            ListNode* temp = cur->next;
            cur->next = temp->next;
            temp->next = prev->next;
            prev->next = temp;
        }

        return dummy->next;
    }
};
```

Output:

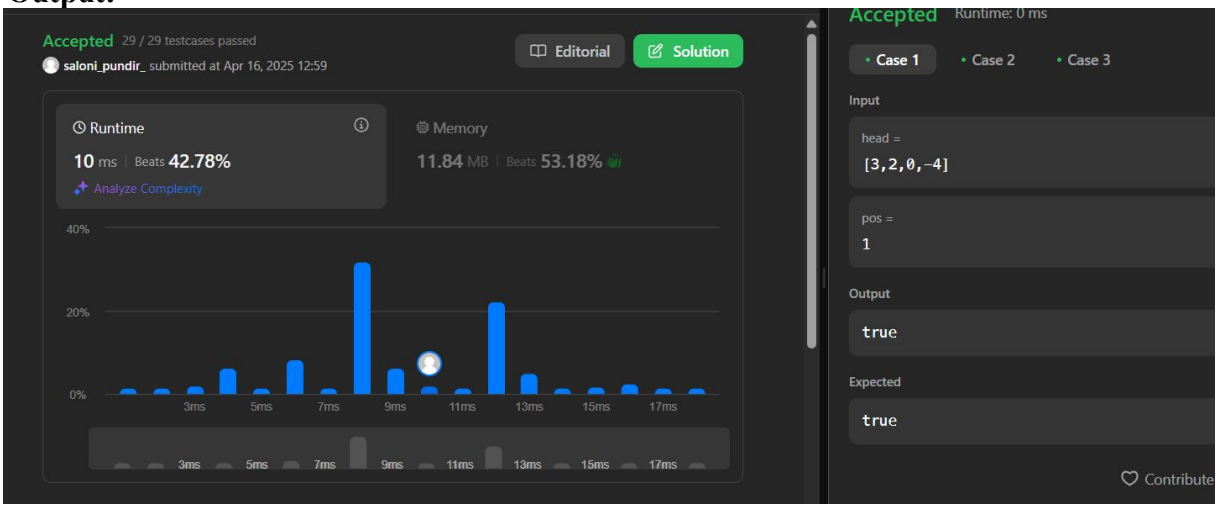


Q4. Detect a Cycle in a Linked List.

Code:

```
class Solution {
public:
    bool hasCycle(ListNode *head) {
ListNode* slow = head;
    ListNode* fast = head;
    while (fast != nullptr && fast->next != nullptr) {
      slow = slow->next;
      fast = fast->next->next;
      if (slow == fast)
        return true;
    }
    return false;
  }

};
```

Output:


Accepted 29 / 29 testcases passed
saloni_pundir_ submitted at Apr 16, 2025 12:59
Editorial
Solution
Runtime
10 ms | Beats 42.78%
Analyze Complexity
Memory
11.84 MB | Beats 53.18%
40%
20%
0%
3ms
5ms
7ms
9ms
11ms
13ms
15ms
17ms
Accepted Runtime: 0 ms
Case 1
Case 2
Case 3
Input
head =
[3,2,0,-4]
pos =
1
Output
true
Expected
true
Contribute

Q5. The Skyline

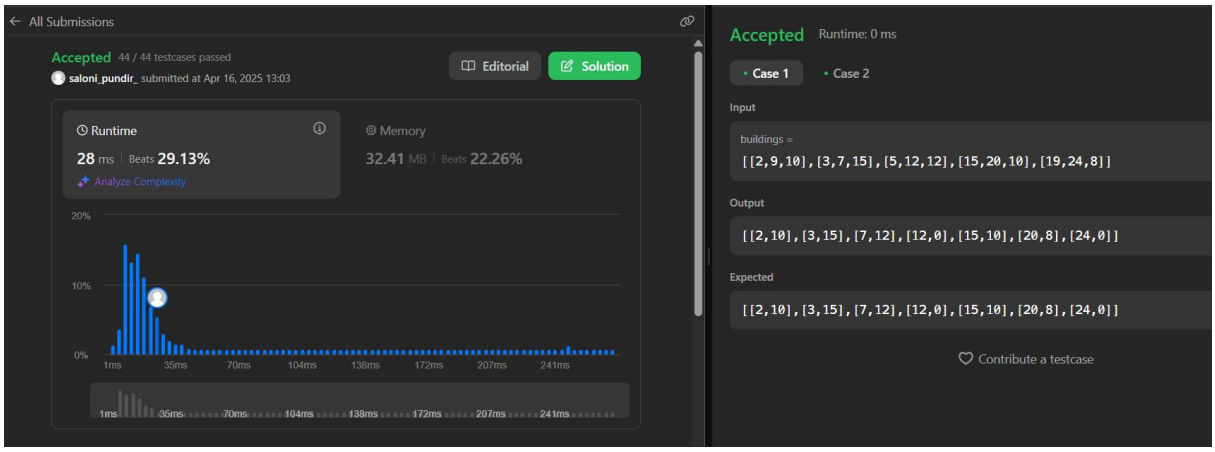
Problem.

Code:

C++ Auto

```cpp
class Solution {
public:
    vector<vector<int>> getSkyline(vector<vector<int>>& buildings) {
        vector<vector<int>> skyline;
        map<int, vector<pair<int, int>>> map;
        for (auto& building : buildings) {
            map[building[0]].push_back({building[2], 0});
            map[building[1]].push_back({building[2], 1});
        }
        multiset<int> q;
        for (auto& [pos, heights] : map) {
            for (auto& [height, type] : heights) {
                if (type == 0) q.insert(height);
                else q.erase(q.find(height));
            }
            int newHeight = q.empty() ? 0 : *q.rbegin();
            if (!skyline.empty() && skyline.back()[1] == newHeight) continue;
            else skyline.push_back(vector<int>({pos, newHeight}));
        }
        return skyline;
    }
};
```

Output:



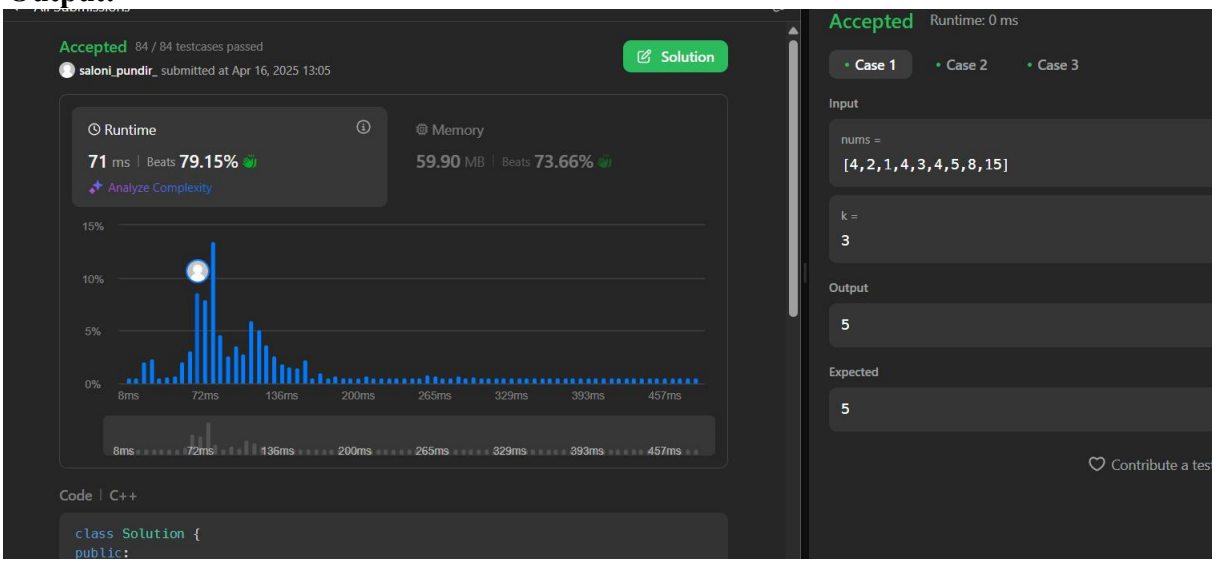
Q6.Longest Increasing Subsequence II

Code:

C++ Auto

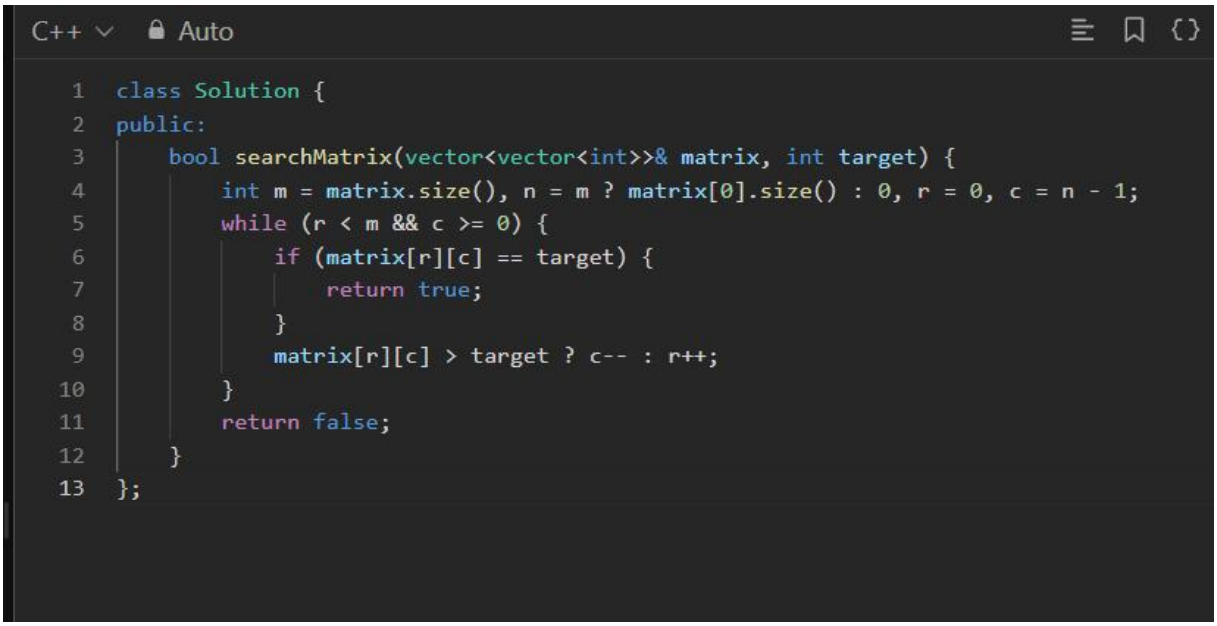
```cpp
class Solution {
public:
    vector<int>tree;
    void update(int node,int st,int end,int i,int val){
        if(st==end){
            tree[node]=max(tree[node],val);
            return;
        }
        int mid=(st+end)/2;
        if(i<=mid){
            update(node*2,st,mid,i,val);
        }else{
            update(node*2+1,mid+1,end,i,val);
        }
        tree[node]=max(tree[node*2],tree[node*2+1]);
    }
    int query(int node,int st,int end,int x,int y){
        if(x>end || y<st) return -1e9;
        if(st>=x && end<=y){
            return tree[node];
        }
        int mid=(st+end)/2;
        int left=query(2*node,st,mid,x,y);
        int right=query(2*node+1,mid+1,end,x,y);
        return max(left,right);
    }
    int lengthOfLIS(vector<int>& nums, int k) {
        int n=nums.size();
        if(n==1) return 1:
```

Output:



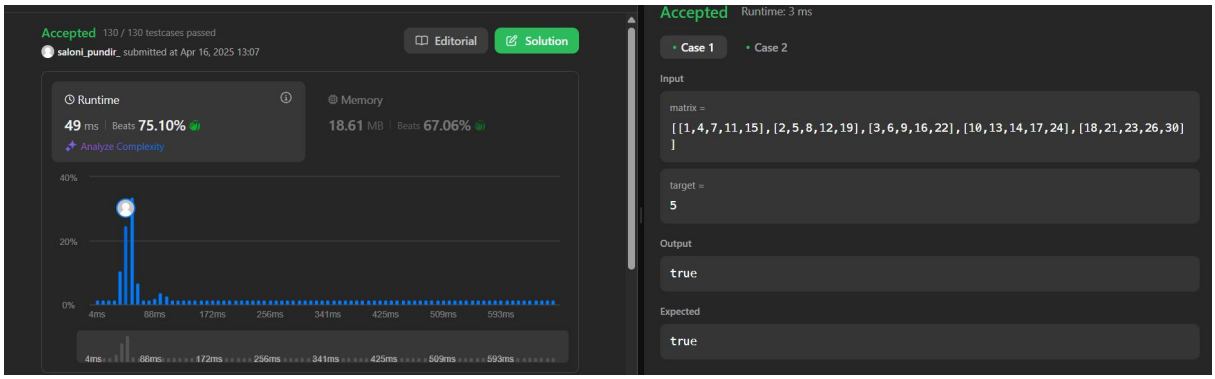
Q7. Search a 2D Matrix II.

Code:



```cpp
class Solution {
public:
    bool searchMatrix(vector<vector<int>>& matrix, int target) {
        int m = matrix.size(), n = m ? matrix[0].size() : 0, r = 0, c = n - 1;
        while (r < m && c >= 0) {
            if (matrix[r][c] == target) {
                return true;
            }
            matrix[r][c] > target ? c-- : r++;
        }
        return false;
    }
};
```

Output:

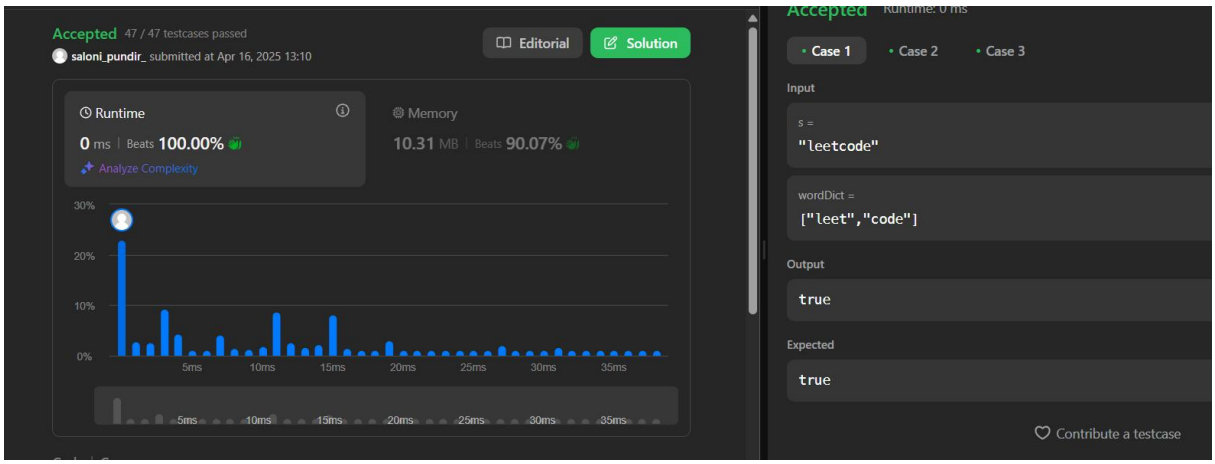


Q8. Word Break.

Code:

```cpp
class Solution {
public:
    bool wordBreak(string s, vector<string>& wordDict) {
        vector<bool> dp(s.size() + 1, false);
        dp[0] = true;

        for (int i = 1; i <= s.size(); i++) {
            for (const string& w : wordDict) {
                int start = i - w.length();
                if (start >= 0 && dp[start] && s.substr(start, w.length()) == w) {
                    dp[i] = true;
                    break;
                }
            }
        }
        return dp[s.size()];
    }
};
```

Output:



Q9.Longest Increasing Path in a Matrix.

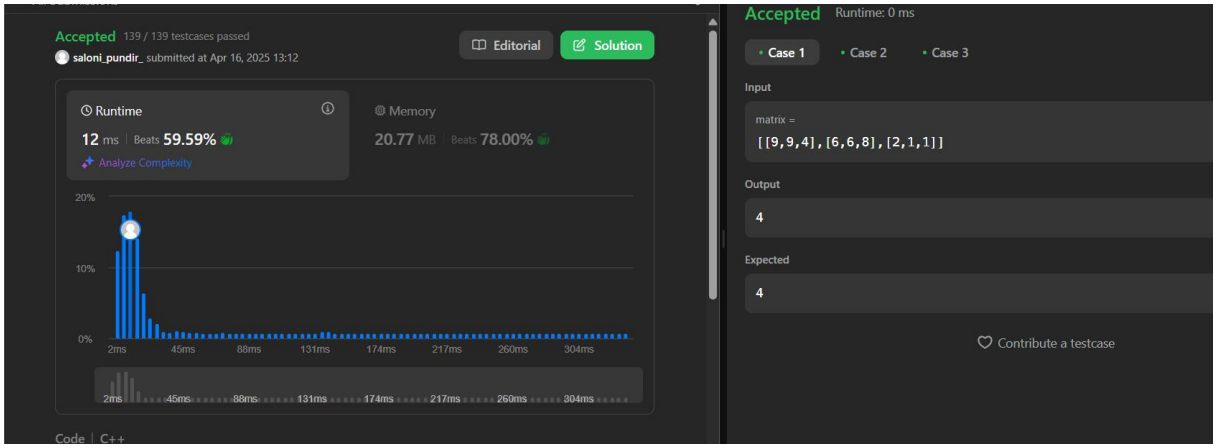
Code:

C++ Auto

```cpp
class Solution {
    vector<vector<int>> dp;
    int dr[4] = {-1,0,1,0};
    int dc[4] = {0,-1,0,1};
public:
    int dfs(int row, int col, vector<vector<int>>& mat) {
        int m = mat.size();
        int n = mat[0].size();

        if (dp[row][col] != 0) {
            return dp[row][col];
        }

        int maxLength = 1;

        for (int k=0; k<4; k++) {
            int nr = row + dr[k];
            int nc = col + dc[k];
            if (nr>=0 && nr<m && nc>=0 && nc<n && mat[nr][nc] > mat[row][col]) {
                maxLength = max(maxLength, 1 + dfs(nr, nc, mat));
            }
        }

        return dp[row][col] = maxLength;
    }

    int longestIncreasingPath(vector<vector<int>>& matrix) {
        int m = matrix.size();
        int n = matrix[0].size();
```

Saved Ln 42,

Output:



Q10.Trapping Rain Water.

Code:

C++ Auto

```cpp
class Solution {
public:
    int trap(vector<int>& height) {
        int left = 0;
        int right = height.size() - 1;
        int leftMax = height[left];
        int rightMax = height[right];
        int water = 0;

        while (left < right) {
            if (leftMax < rightMax) {
                left++;
                leftMax = max(leftMax, height[left]);
                water += leftMax - height[left];
            } else {
                right--;
                rightMax = max(rightMax, height[right]);
                water += rightMax - height[right];
            }
        }

        return water;
    }
};
```

Output: