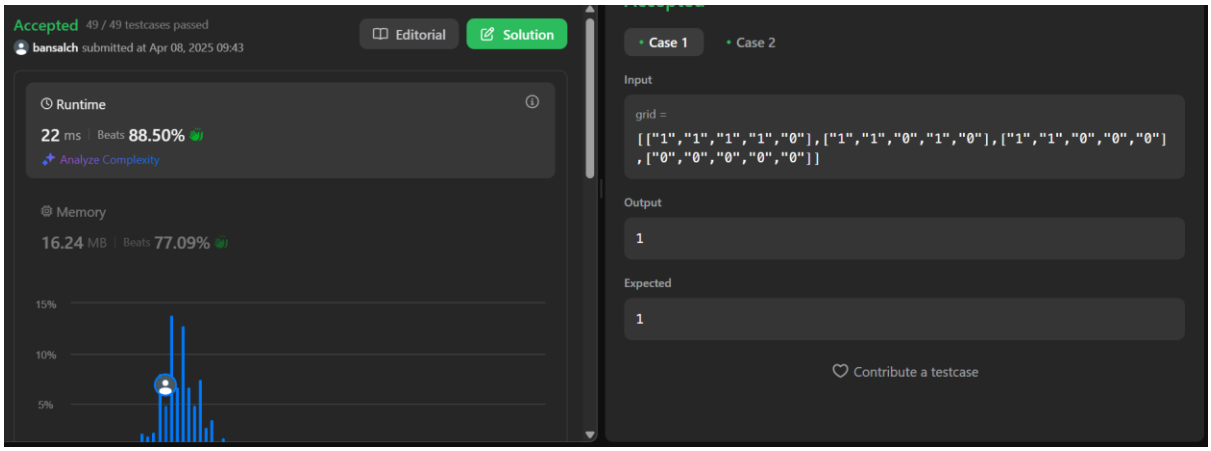
AP 9th Experiment

Q1. Number of Islands

Code:

```
public:
    void dfs(vector<vector<char>>& g, int i, int j) {
        if (i < 0 || j < 0 || i >= g.size() || j >= g[0].size() || g[i][j]
== '0') return;
        g[i][j] = '0';
        dfs(g, i + 1, j);
        dfs(g, i - 1, j);
        dfs(g, i, j + 1);
        dfs(g, i, j - 1);
    }
    int numIslands(vector<vector<char>>& g) {
        int count = 0;
        for (int i = 0; i < g.size(); i++) {
            for (int j = 0; j < g[0].size(); j++) {
                if (g[i][j] == '1') {
                    count++;
                    dfs(g, i, j);
                }
            }
        }
        return count;
    }
};
```

Output:

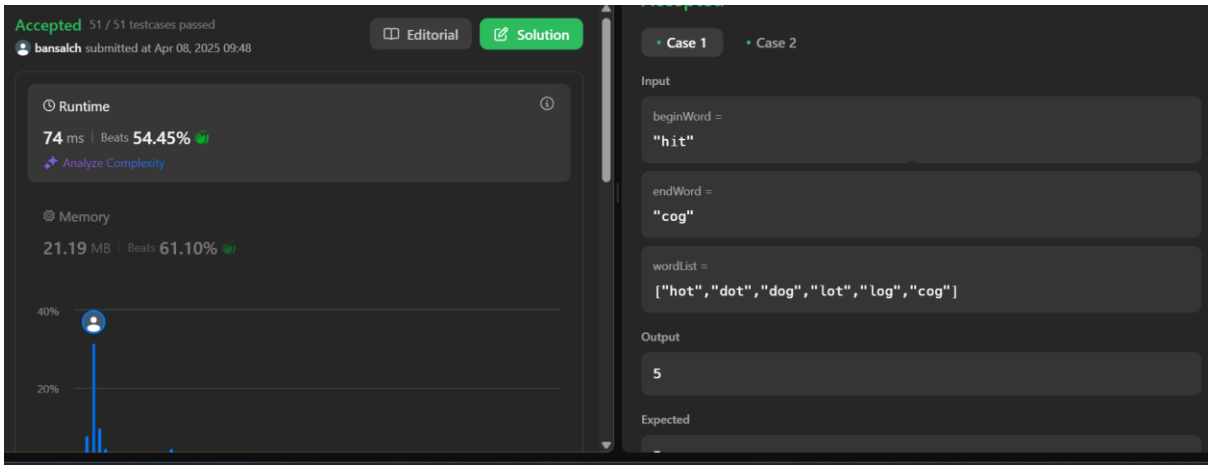


Q2. Word Ladder

Code:

```
public:
    int ladderLength(string beginWord, string endWord, vector<string>&
wordList) {
        unordered_set<string> words(wordList.begin(), wordList.end());
        if (!words.count(endWord)) return 0;
        queue<pair<string, int>> q;
        q.push({beginWord, 1});
        while (!q.empty()) {
            auto [word, len] = q.front(); q.pop();
            for (int i = 0; i < word.size(); i++) {
                string temp = word;
                for (char c = 'a'; c <= 'z'; c++) {
                    temp[i] = c;
                    if (temp == endWord) return len + 1;
                    if (words.count(temp)) {
                        q.push({temp, len + 1});
                        words.erase(temp);
                    }
                }
            }
        }
        return 0;
    }
```

Output:

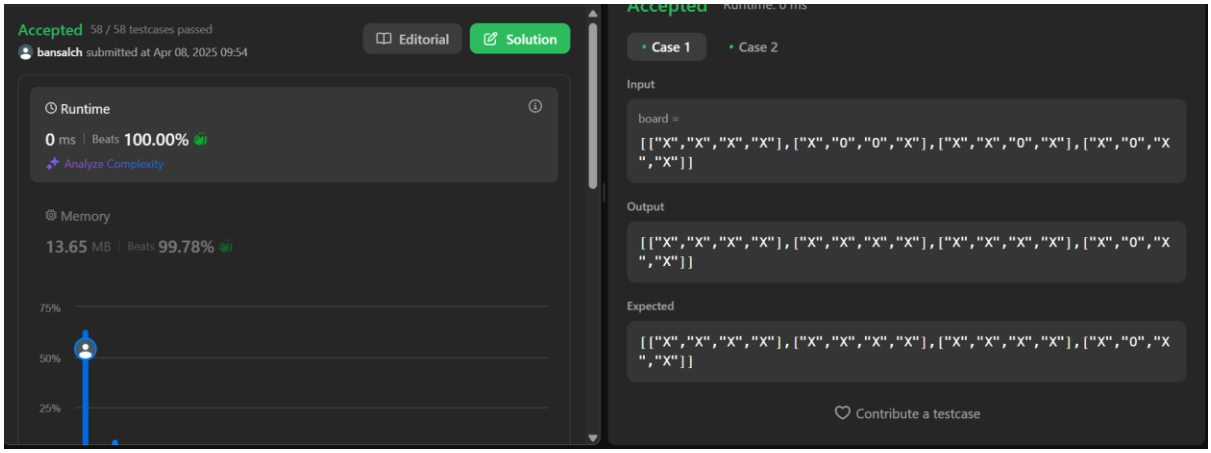


Q3. Surrounded Regions

Code:

```
    void dfs(vector<vector<char>>& board, int i, int j) {
        if (i < 0 || j < 0 || i >= board.size() || j >= board[0].size() ||
board[i][j] != 'O')
            return;
        board[i][j] = 'S';
        dfs(board, i + 1, j);
        dfs(board, i - 1, j);
        dfs(board, i, j + 1);
        dfs(board, i, j - 1); }
    void solve(vector<vector<char>>& board) {
        if (board.empty()) return;
        int m = board.size(), n = board[0].size();
        for (int i = 0; i < m; i++) {
            dfs(board, i, 0);
            dfs(board, i, n - 1);  }
        for (int j = 0; j < n; j++) {
            dfs(board, 0, j);
            dfs(board, 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';
                else if (board[i][j] == 'S') board[i][j] = 'O';
            } } } };
```

Output:

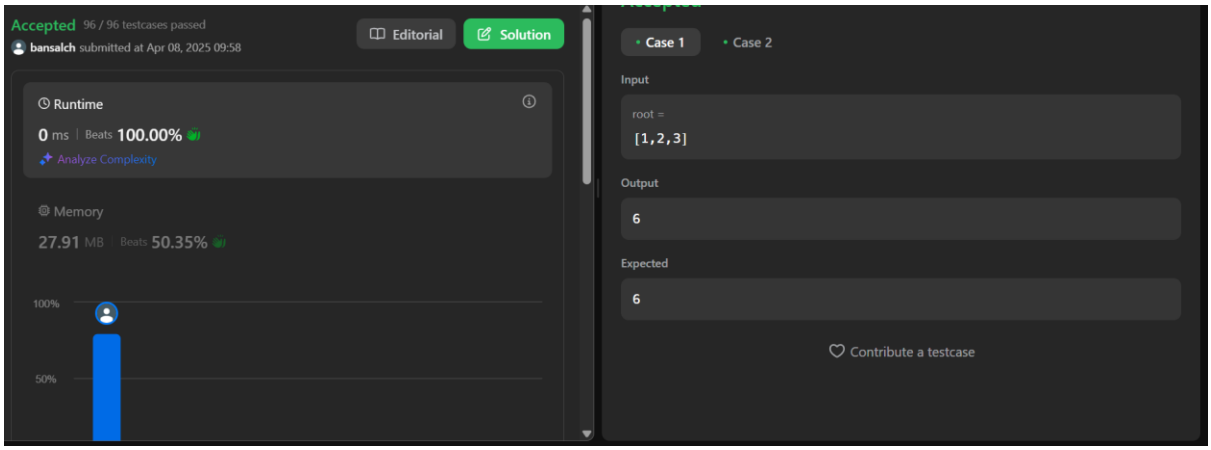


Q4. Binary Tree Maximum Path Sum

Code:

```
class Solution {
public:
    int maxSum = INT_MIN;
    int dfs(TreeNode* node) {
        if (!node) return 0;
        int left = max(0, dfs(node->left));
        int right = max(0, dfs(node->right));
        int curr = node->val + left + right;
        maxSum = max(maxSum, curr);
        return node->val + max(left, right);
    }
    int maxPathSum(TreeNode* root) {
        dfs(root);
        return maxSum;
    }
};
```

Output:

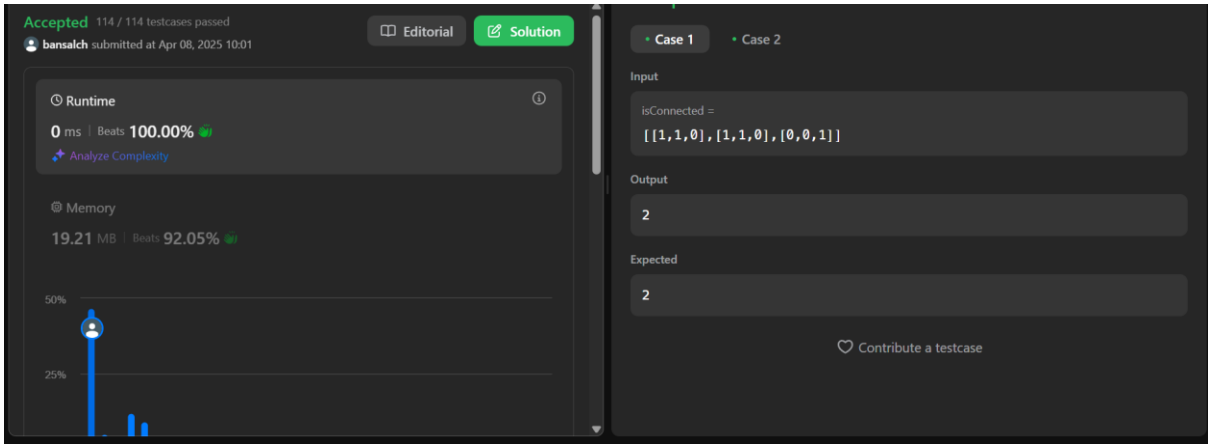


Q5. Friend Circles

Code:

```cpp
class Solution {
public:
    void dfs(vector<vector<int>>& g, vector<bool>& seen, int i) {
        seen[i] = true;
        for (int j = 0; j < g.size(); j++) {
            if (g[i][j] == 1 && !seen[j]) {
                dfs(g, seen, j);
            }
        }
    }
    int findCircleNum(vector<vector<int>>& isConnected) {
        int n = isConnected.size();
        vector<bool> seen(n, false);
        int count = 0;
        for (int i = 0; i < n; i++) {
            if (!seen[i]) {
                dfs(isConnected, seen, i);
                count++;
            }
        }
        return count;
    }
};
```

Output:

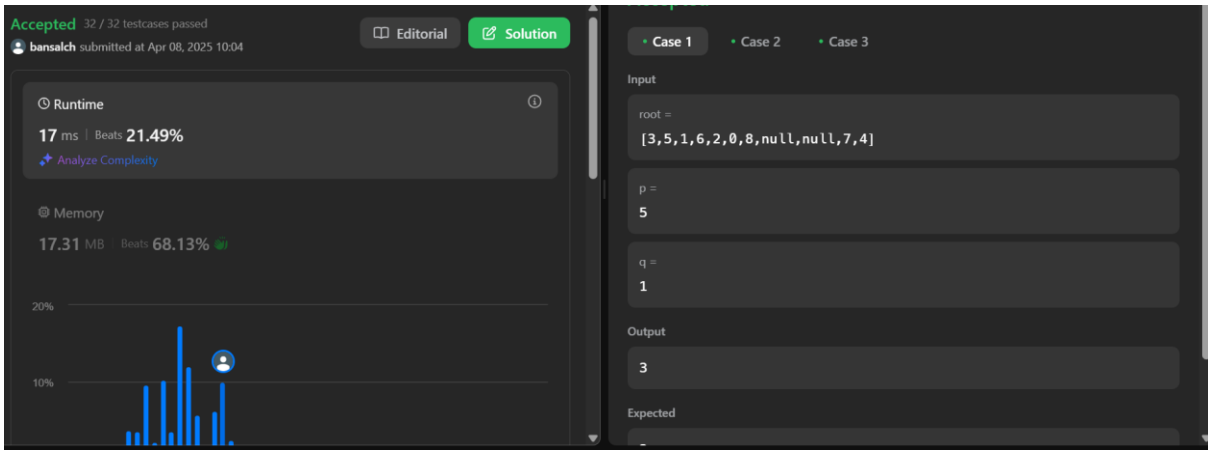


Q6. Lowest Common Ancestor of a Binary Tree

Code:

```cpp
class Solution {
public:
    TreeNode* lowestCommonAncestor(TreeNode* root, TreeNode* p, TreeNode* q)
{
        if (!root || root == p || root == q) return root;
        TreeNode* left = lowestCommonAncestor(root->left, p, q);
        TreeNode* right = lowestCommonAncestor(root->right, p, q);
        if (left && right) return root;
        return left ? left : right;
    }
};
```

Output:

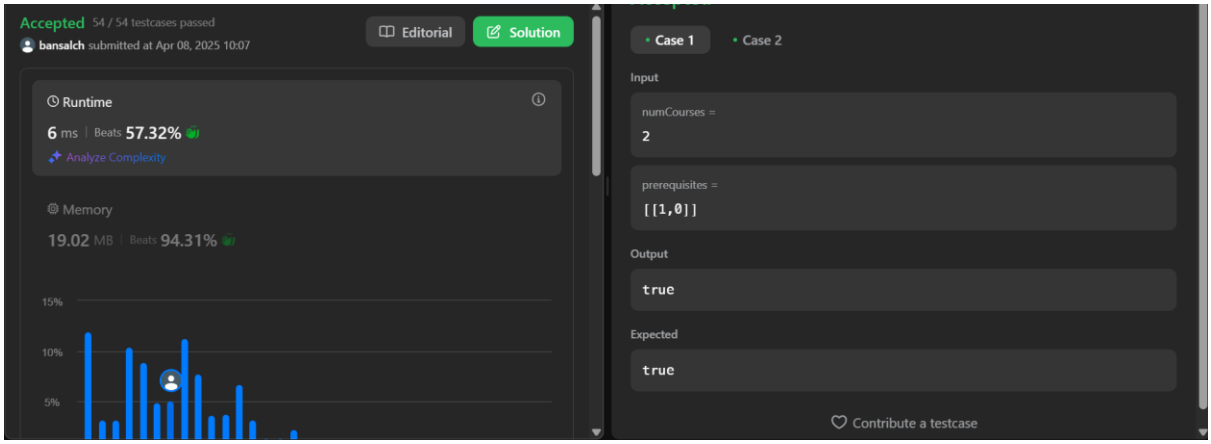


Q7. Course Schedule

Code:

```cpp
bool canFinish(int numCourses, vector<vector<int>>& prerequisites) {
    vector<vector<int>> graph(numCourses);
    vector<int> inDegree(numCourses, 0);
    for (auto& pre : prerequisites) {
        graph[pre[1]].push_back(pre[0]);
        inDegree[pre[0]]++;
    }
    queue<int> q;
    for (int i = 0; i < numCourses; i++) {
        if (inDegree[i] == 0) q.push(i);
    }
    int count = 0;
    while (!q.empty()) {
        int course = q.front(); q.pop();
        count++;
        for (int next : graph[course]) {
            inDegree[next]--;
            if (inDegree[next] == 0) q.push(next);
        }
    }
    return count == numCourses;
}
};
```

Output:



Q8. Longest Increasing Path in a Matrix

Code:

```
    int go(vector<vector<int>>& mat, int i, int j, vector<vector<int>>& dp) {
        if (dp[i][j] != 0) return dp[i][j];
        int m = mat.size();
        int n = mat[0].size();
        int best = 1;
        int dir[4][2] = {{0,1}, {1,0}, {0,-1}, {-1,0}};
        for (int d = 0; d < 4; d++) {
            int x = i + dir[d][0];
            int y = j + dir[d][1];
            if (x >= 0 && y >= 0 && x < m && y < n && mat[x][y] > mat[i][j])
{
                best = max(best, 1 + go(mat, x, y, dp));
            } }
        dp[i][j] = best;
        return best;  }
    int longestIncreasingPath(vector<vector<int>>& mat) {
        if (mat.empty()) return 0;
        int m = mat.size();
        int n = mat[0].size();
        vector<vector<int>> dp(m, vector<int>(n, 0));
        int ans = 0;
        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
```

Output: