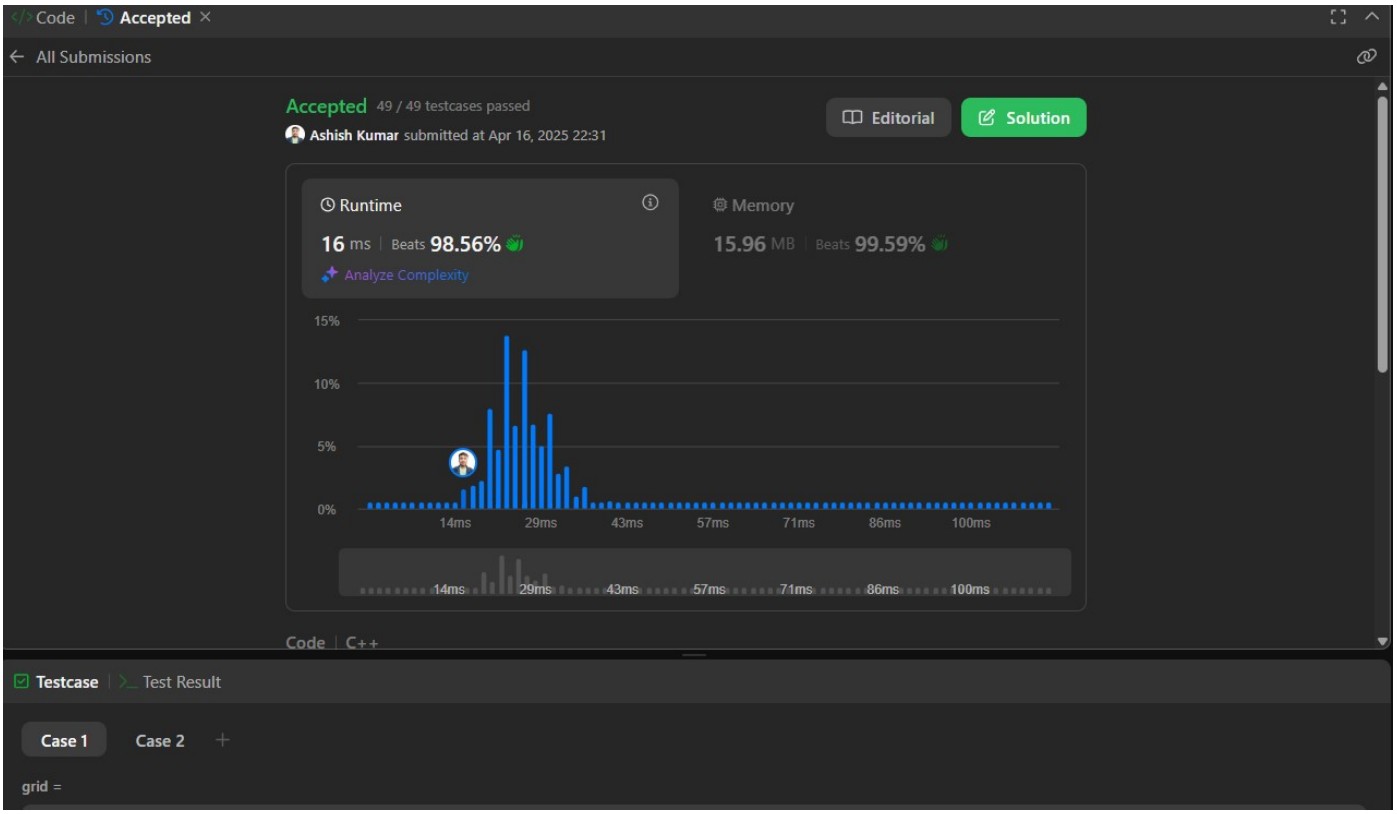
Experiment 9

Student Name: Ashish Kumar **UID:22bcs11958**
Branch: CSE **Section/Group:614(B)**
Semester: 6 **Date of Performance:11/04/25**
Subject Name: AP Lab **Subject Code: 22CSP-351**

Q1:-Number of Islands

```
class Solution {
public:
        int m;
        int n;
        void dfs(vector<vector<char>>& grid,int i, int j)
        {
            if(i<0|| i>=m || j<0 || j>=n || grid[i][j] !='1')
            {
                return;
            }
            if(grid[i][j]==-1)
            {
                return;
            }
            grid[i][j]='$';

            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) {
       m=grid.size();
       n=grid[0].size();
       int islands=0;
       for(int i=0;i<m;i++)
       {
        for(int j=0;j<n;j++)
        {
            if(grid[i][j]=='1')
            {
                dfs(grid,i,j);
                islands++;
            }
        }
       }
       return islands;
```

```
    }
};
```



Q2:-Word Ladder

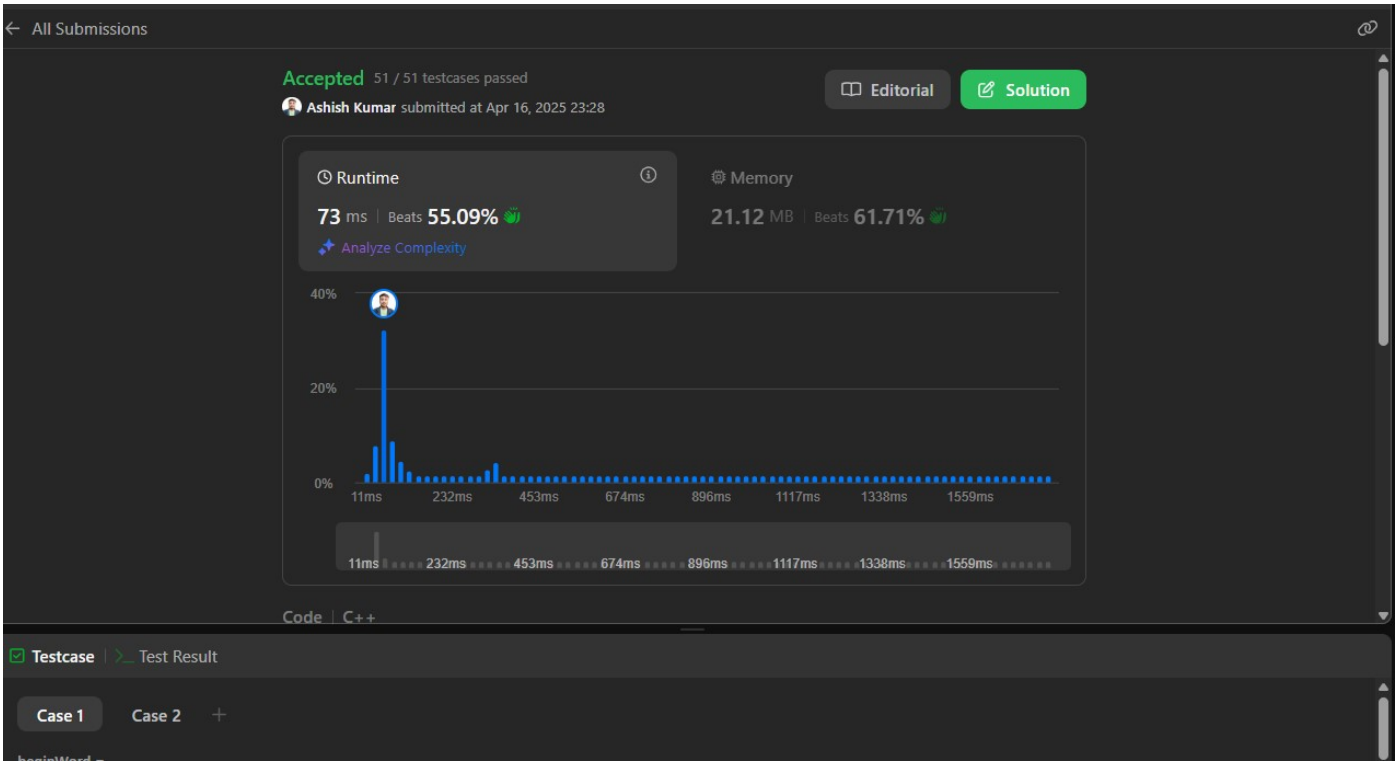
```
class Solution {
public:
    int ladderLength(string beginWord, string endWord, vector<string>& wordList) {
        unordered_set<string> wordSet(wordList.begin(), wordList.end());
        if (wordSet.find(endWord) == wordSet.end()) return 0;

        queue<pair<string, int>> q;
        q.push({beginWord, 1});
```

```
        while (!q.empty()) {
            auto [word, length] = 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 length + 1;
                    if (wordSet.find(temp) != wordSet.end()) {
                        q.push({temp, length + 1});
                        wordSet.erase(temp); // avoid revisiting
                    }
```

```
                }
            }
        }
```

```
        return 0; // no possible transformation
    }
};
```



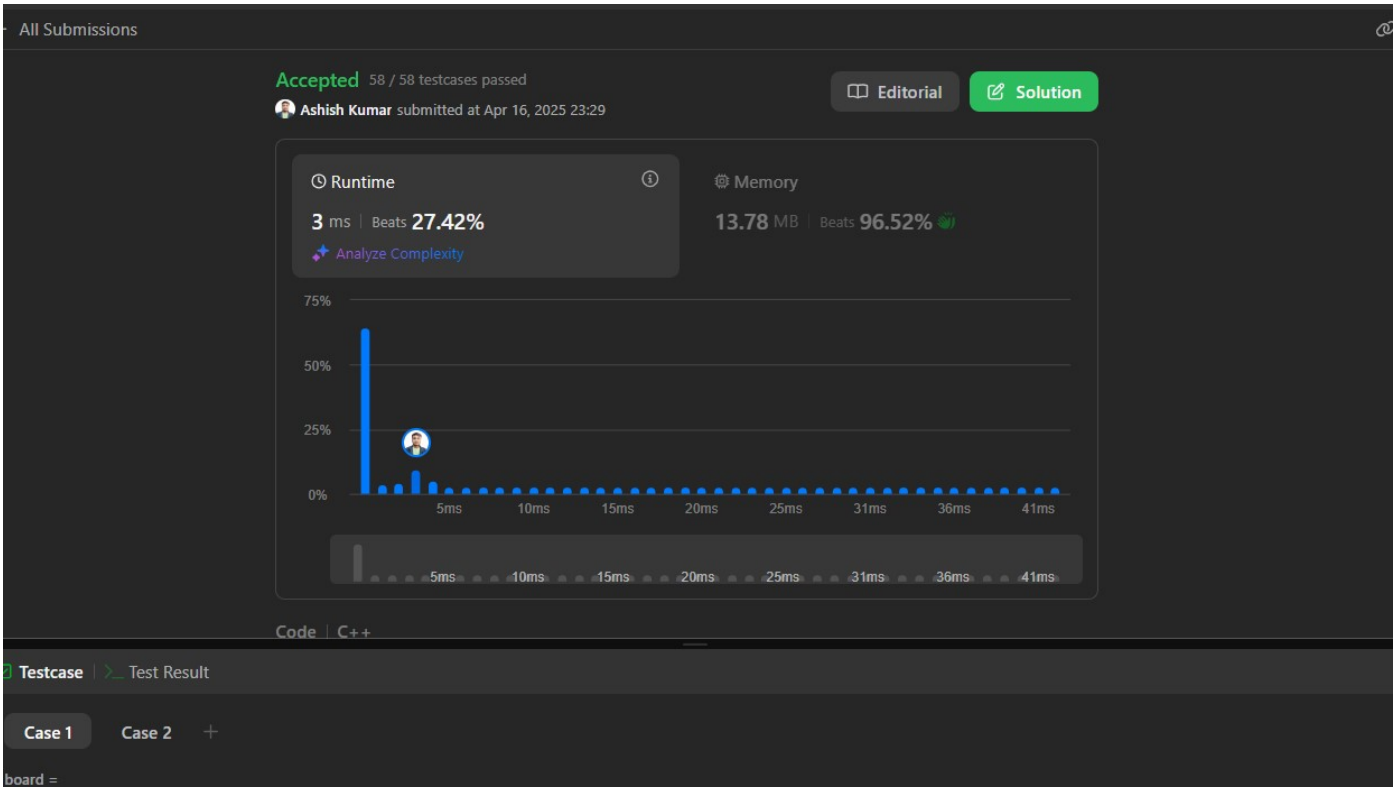
Q3:-Surrounded Regions

```
class Solution {
public:
    void dfs(vector<vector<char>>& board, int i, int j) {
        int m = board.size(), n = board[0].size();
        if (i < 0 || j < 0 || i >= m || j >= n || board[i][j] != 'O') return;

        board[i][j] = 'T'; // Mark as temporarily safe
```

```
        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();

        // 1. Mark all border-connected 'O's as 'T'
        for (int i = 0; i < m; ++i) {
            if (board[i][0] == 'O') dfs(board, i, 0);
            if (board[i][n-1] == 'O') dfs(board, i, n-1);
        }
        for (int j = 0; j < n; ++j) {
            if (board[0][j] == 'O') dfs(board, 0, j);
            if (board[m-1][j] == 'O') dfs(board, m-1, j);
        }

        // 2. Flip inner 'O' to 'X', and 'T' back to 'O'
        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] == 'T') board[i][j] = 'O';
            }
        }
    }
};
```



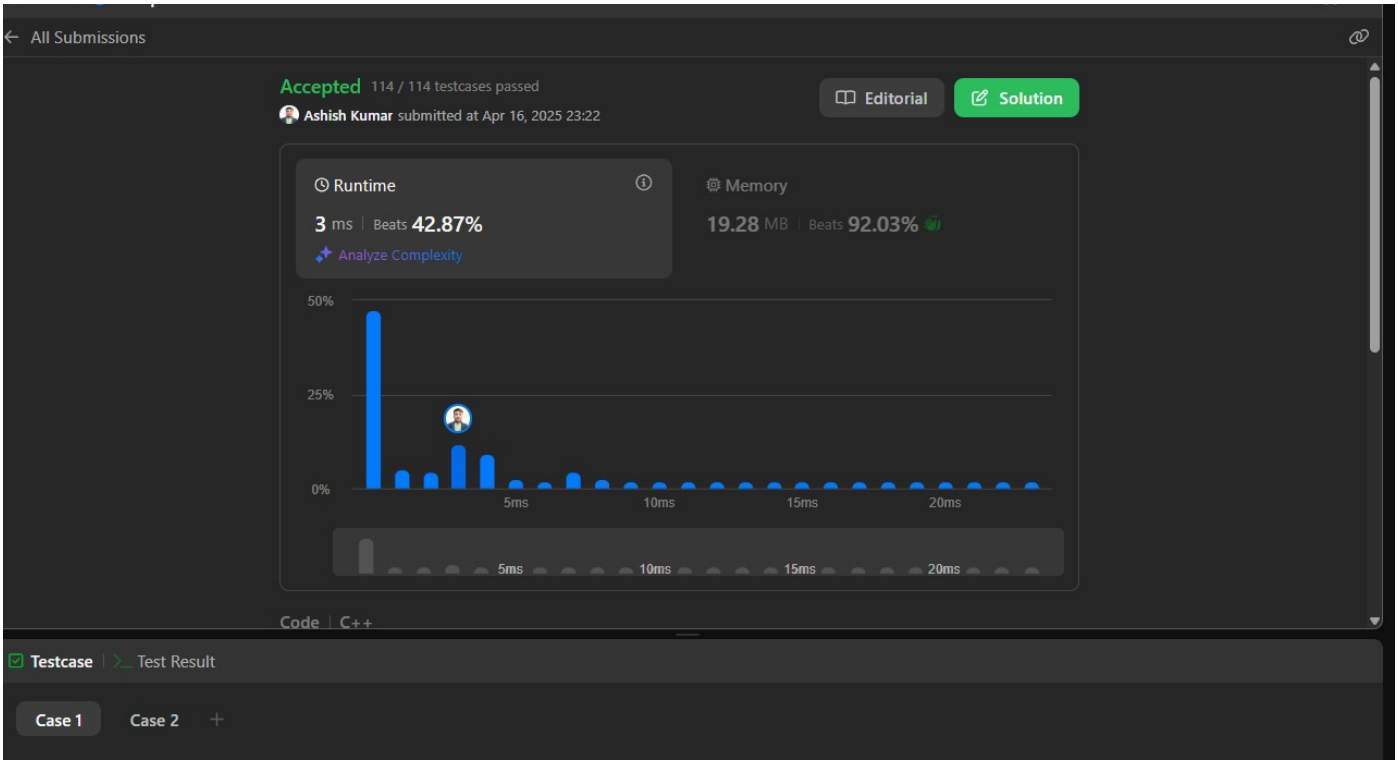
Q4:-Binary Tree Maximum Path Sum

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
 * };
 */
class Solution {
public:
  int maxSumPath(TreeNode*root,int &maxSum)
  {
    if(root==NULL)
    {
        return 0;
    }
    int leftMax =max(0,maxSumPath(root->left,maxSum));
    int rightMax =max(0,maxSumPath(root->right,maxSum));
    maxSum=max(maxSum,root->val+leftMax+rightMax);
    return root->val+max(leftMax,rightMax);
  }
    int maxPathSum(TreeNode* root) {
        int maxSum=INT_MIN;
        maxSumPath(root,maxSum);
        return maxSum;
    }
};
```

Q5:-Friend Circles

```
class Solution {
public:
   int n;
   void dfs(vector<vector<int>>&isConnected,int u,vector<bool>&visited)
   {
    visited[u]=true;
    for(int v=0;v<n;v++)
    {
        if(!visited[v] && isConnected[u][v]==1)
        {
            dfs(isConnected,v,visited);
        }
    }
   }
    int findCircleNum(vector<vector<int>>& isConnected) {
        n=isConnected.size();
        vector<bool>visited(n,false);

            int count=0;
            for(int i=0;i<n;i++)
            {
                if(!visited[i])
                {
                    count++;
                    dfs(isConnected,i,visited);
                }
```

```
            }
        return count;
    }
};
```



Q6:-Lowest Common Ancestor of a Binary Tree

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    TreeNode* lowestCommonAncestor(TreeNode* root, TreeNode* p, TreeNode* q) {
       if(root==NULL || root==p || root==q)
       {
        return root;
       }
       TreeNode*left=lowestCommonAncestor(root->left,p,q);
       TreeNode*right=lowestCommonAncestor(root->right,p,q);

       if(left==NULL)
```

```
        {
         return right;
        }
        else if(right==NULL)
        {
         return left;
        }
        else
        {
         return root;
        }
    }
};
```