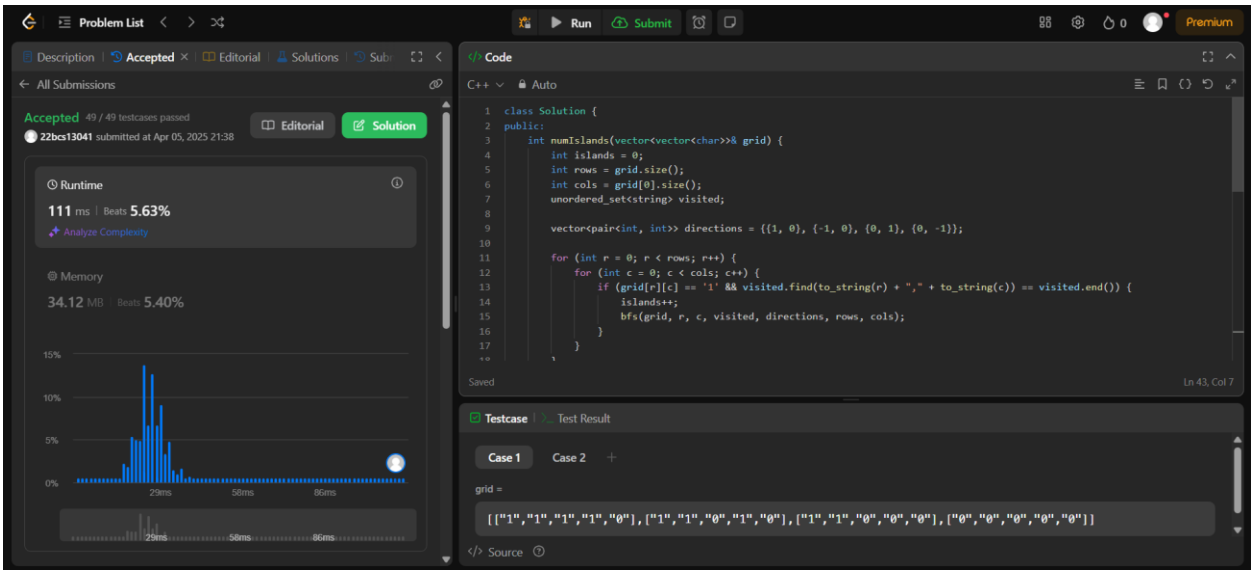
Name: Pranjal Singh **UID:** 22BCS13041
Sec: FL_IOT-601/ A **Sub:** AP Lab -II

Number of Islands

```
class Solution {
public:
   int numIslands(vector<vector<char>>& grid) {
      int islands = 0;
      int rows = grid.size();
      int cols = grid[0].size();
      unordered_set<string> visited;

      vector<pair<int, int>> directions = {{1, 0}, {-1, 0}, {0, 1}, {0, -1}};

      for (int r = 0; r < rows; r++) {
         for (int c = 0; c < cols; c++) {
            if (grid[r][c] == '1' && visited.find(to_string(r) + "," + to_string(c)) == visited.end()) {
               islands++;
               bfs(grid, r, c, visited, directions, rows, cols);
            }
         }
      }

      return islands;
   }

private:
   void bfs(vector<vector<char>>& grid, int r, int c, unordered_set<string>& visited,
vector<pair<int, int>>& directions, int rows, int cols) {
      queue<pair<int, int>> q;
      visited.insert(to_string(r) + "," + to_string(c));
      q.push({r, c});

      while (!q.empty()) {
         auto [row, col] = q.front();
         q.pop();
```

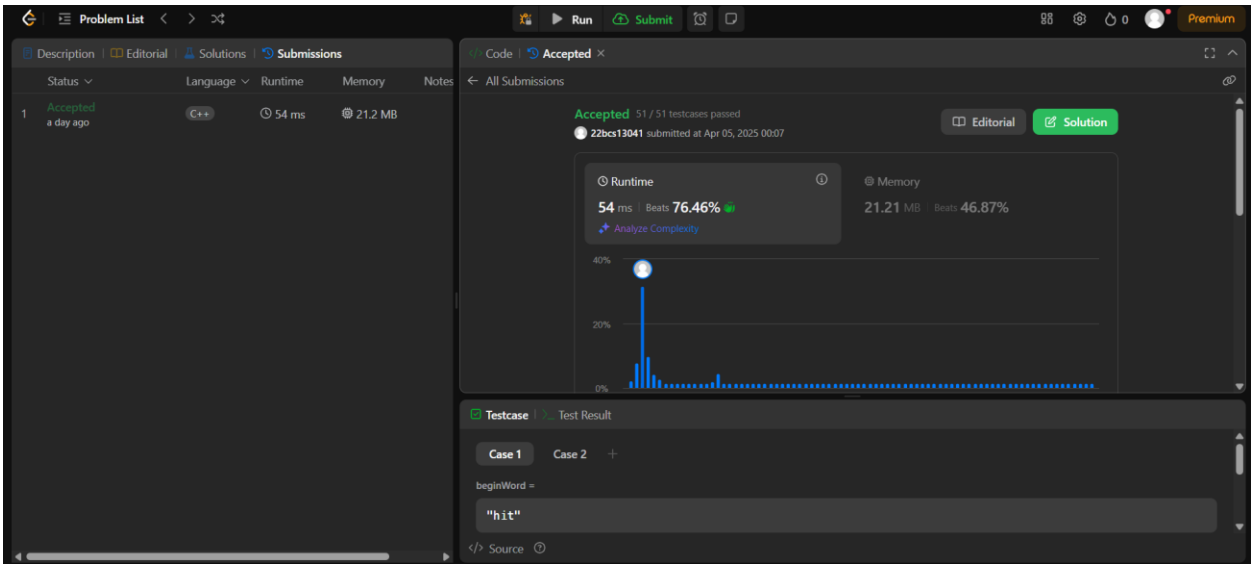
```
        for (auto [dr, dc] : directions) {
            int nr = row + dr;
            int nc = col + dc;
            if (nr >= 0 && nr < rows && nc >= 0 && nc < cols && grid[nr][nc] == '1' &&
visited.find(to_string(nr) + "," + to_string(nc)) == visited.end()) {
                q.push({nr, nc});
                visited.insert(to_string(nr) + "," + to_string(nc));
            }
        }
    }
  }
};
```



Word Ladder

```
class Solution {
public:
   int ladderLength(string beginWord, string endWord, vector<string>& wordList) {

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

      unordered_set<string>st(wordList.begin(),wordList.end());
      st.erase(beginWord);

      while(!q.empty()){
         string word=q.front().first;
```

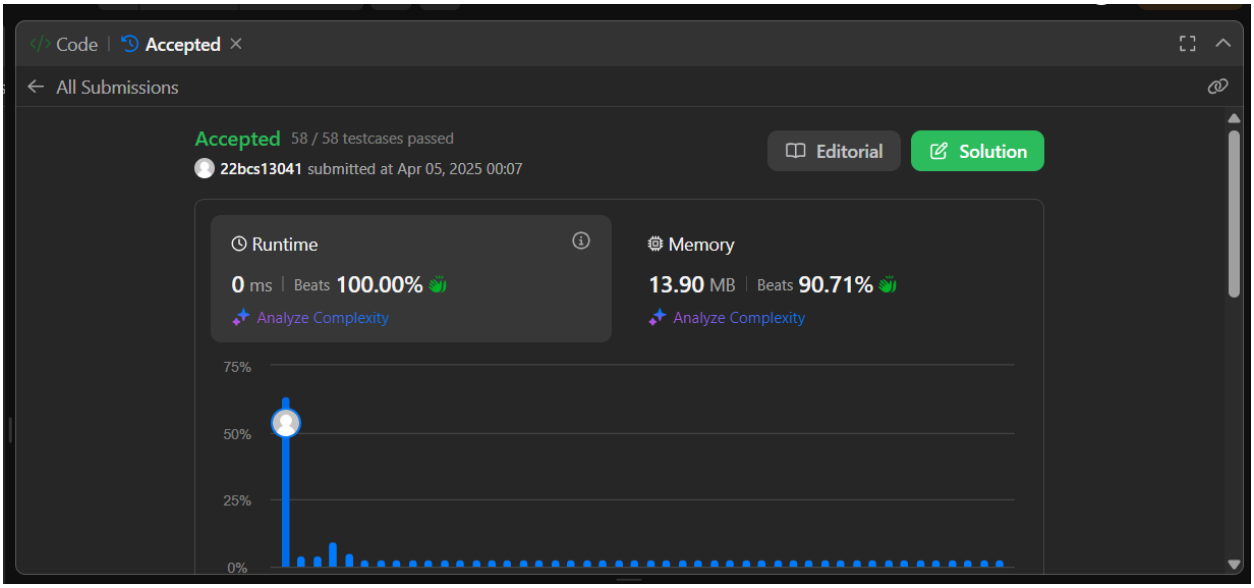
```
        int steps=q.front().second;
        q.pop();

        if(word==endWord) return steps;
        for(int i=0;i<word.size();i++){
           char original=word[i];
           for(int ch='a';ch<='z';ch++){
              word[i]=ch;
              if(st.find(word)!=st.end()){
                 st.erase(word);
                 q.push({word,steps+1});
              }
           }
           word[i]=original;
        }
     }return 0;
  }
};
```



Surrounded Regions

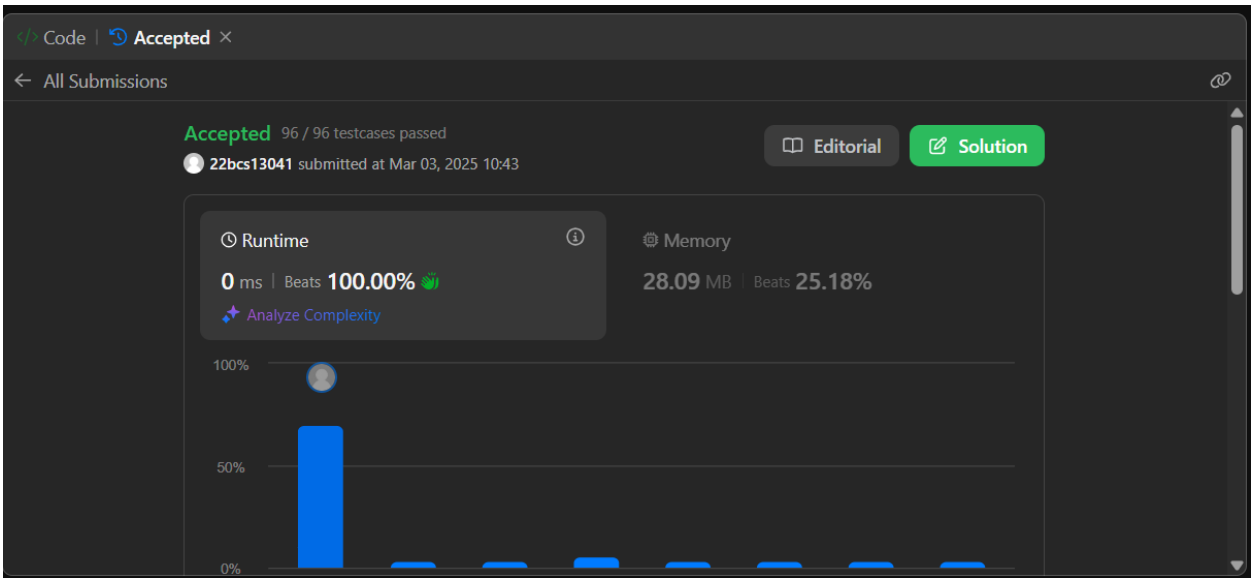
```
class Solution {
public:
   void DFS(vector<vector<char>>& board, int i, int j, int m, int n) {
      if(i<0 or j<0 or i>=m or j>=n or board[i][j] != 'O') return;
      board[i][j] = '#';
      DFS(board, i-1, j, m, n);
      DFS(board, i+1, j, m, n);
      DFS(board, i, j-1, m, n);
      DFS(board, i, j+1, m, n);
   }

   void solve(vector<vector<char>>& board) {
    int m = board.size();
     if(m == 0) return;
    int n = board[0].size();
    //Moving over firts and last column
    for(int i=0; i<m; i++) {
       if(board[i][0] == 'O')
          DFS(board, i, 0, m, n);
       if(board[i][n-1] == 'O')
          DFS(board, i, n-1, m, n);
    }
    //Moving over first and last row
    for(int j=0; j<n; j++) {
       if(board[0][j] == 'O')
          DFS(board, 0, j, m, n);
       if(board[m-1][j] == 'O')
          DFS(board, m-1, j, m, n);
    }

    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] == '#')
             board[i][j] = 'O';
```

```
        }
    }
};
```



Binary Tree Maximum Path Sum

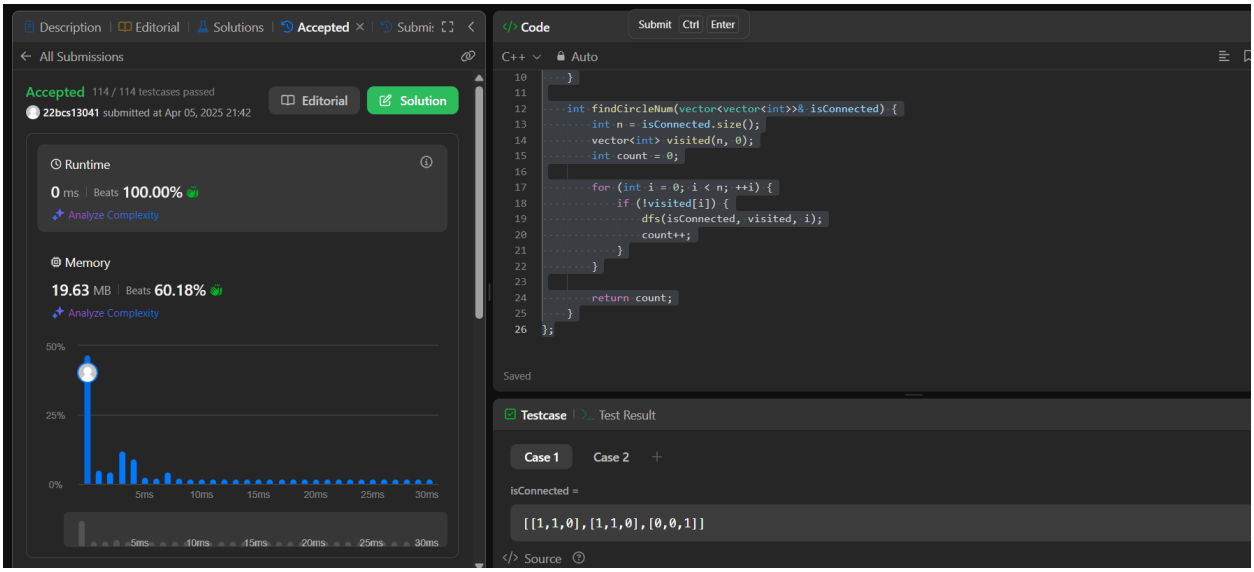
```
class Solution {
 public:
  int maxPathSum(TreeNode* root) {
    int ans = INT_MIN;
    maxPathSumDownFrom(root, ans);
    return ans;
  }
 private:
  int maxPathSumDownFrom(TreeNode* root, int& ans) {
    if (root == nullptr)
      return 0;

    const int l = max(0, maxPathSumDownFrom(root->left, ans));
    const int r = max(0, maxPathSumDownFrom(root->right, ans));
    ans = max(ans, root->val + l + r);
    return root->val + max(l, r);
  }
};
```



Friend Circles

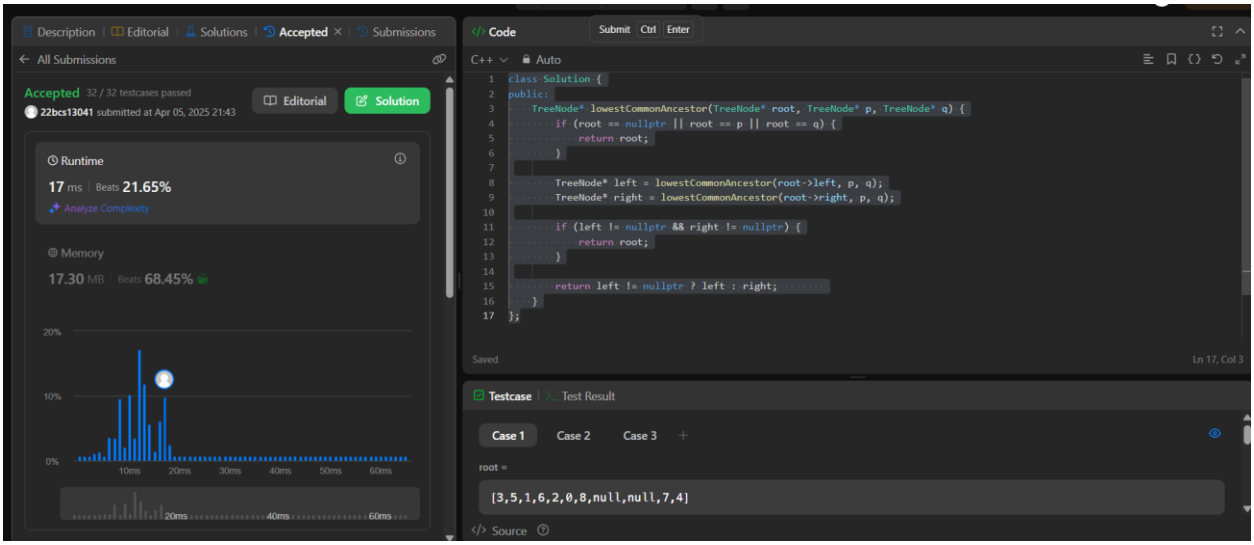
```
class Solution {
public:
    void dfs(vector<vector<int>>& isConnected, vector<int>& visited, int i) {
        visited[i] = 1;
        for (int j = 0; j < isConnected.size(); ++j) {
            if (isConnected[i][j] == 1 && !visited[j]) {
                dfs(isConnected, visited, j);
            }
        }
    }

    int findCircleNum(vector<vector<int>>& isConnected) {
        int n = isConnected.size();
        vector<int> visited(n, 0);
        int count = 0;

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

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



Lowest Common Ancestor of a Binary Tree

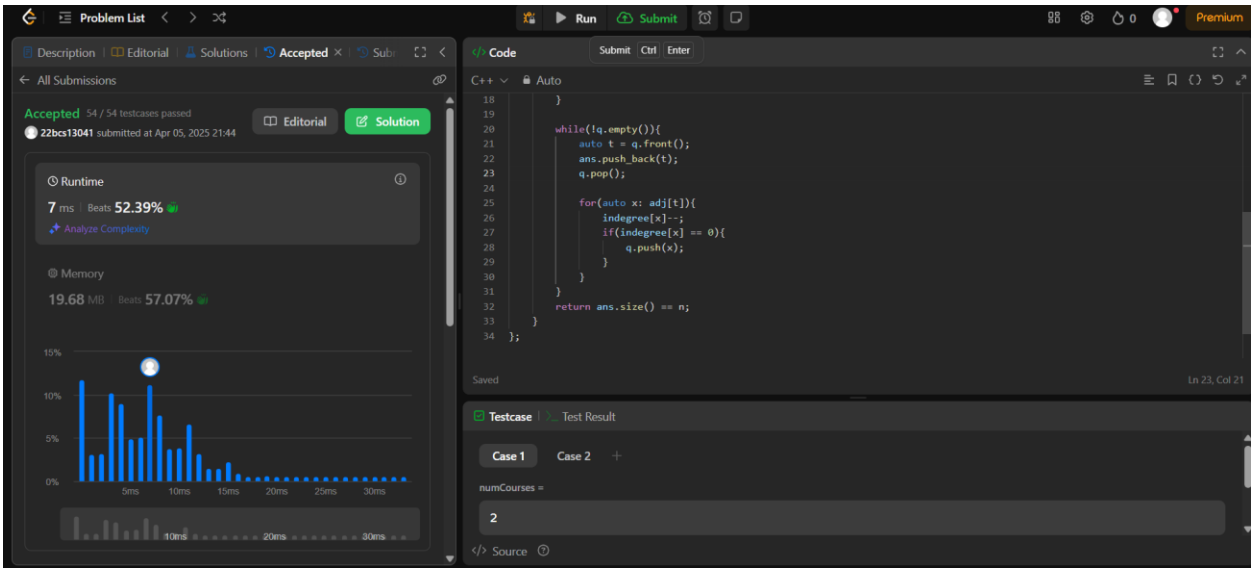
```
class Solution {
public:
    TreeNode* lowestCommonAncestor(TreeNode* root, TreeNode* p, TreeNode* q) {
        if (root == nullptr || root == p || root == q) {
            return root;
        }

        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;
    }
};
```



Course Schedule

```
class Solution {
public:
    bool canFinish(int n, vector<vector<int>>& prerequisites) {
        vector<int> adj[n];
        vector<int> indegree(n, 0);
        vector<int> ans;

        for(auto x: prerequisites){
            adj[x[0]].push_back(x[1]);
            indegree[x[1]]++;
        }

        queue<int> q;
        for(int i = 0; i < n; i++){
            if(indegree[i] == 0){
                q.push(i);
            }
        }

        while(!q.empty()){
            auto t = q.front();
            ans.push_back(t);
            q.pop();
```

```
            for(auto x: adj[t]){
                indegree[x]--;
                if(indegree[x] == 0){
                    q.push(x);
                }
            }
        }
        return ans.size() == n;
    }
};
```



Longest Increasing Path in a Matrix

```
class Solution {
public:
    int dfs(int row, int col, vector<vector<int>>& matrix, vector<vector<int>>& dp){
        int m = matrix.size(), n = matrix[0].size();
        int ans = 1;
        if(dp[row][col] != -1) return dp[row][col];
        for(int tempR=-1; tempR<=1; tempR++){
            for(int tempC=-1; tempC<=1; tempC++){
                if(abs(tempR) + abs(tempC) != 1) continue;
                int Nrow = row + tempR;
                int Ncol = col + tempC;
                if(Nrow>=0 && Nrow<m && Ncol>=0 && Ncol<n && matrix[Nrow][Ncol] >
matrix[row][col]){
```

```
                ans = max(ans, 1+dfs(Nrow, Ncol, matrix, dp));
            }
        }
    }
    return dp[row][col] = ans;
}
int longestIncreasingPath(vector<vector<int>>& matrix) {
    int m = matrix.size(), n = matrix[0].size();
    int result = 0;
    vector<vector<int>> dp(m, vector<int>(n, -1));
    for(int i=0; i<m; i++){
        for(int j=0; j<n; j++){
            result = max(result, dfs(i, j, matrix, dp));
        }
    }
    return result;
}
};
```