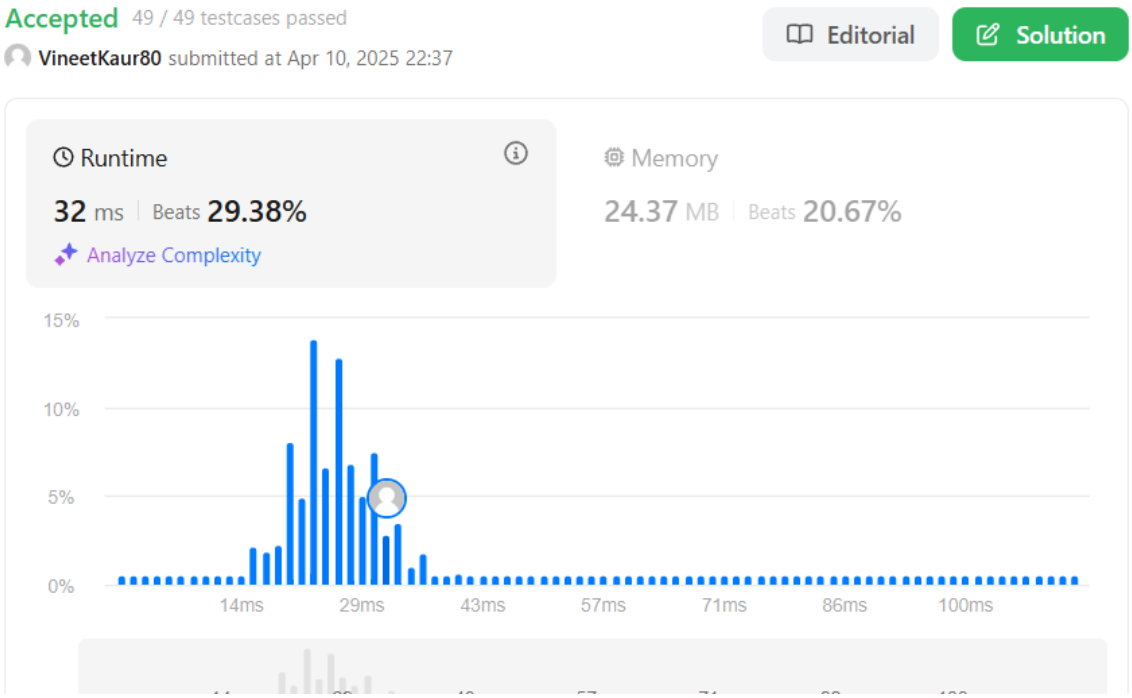
AP 9TH EXPRIMENT

1. **Number of Islands**

CODE

```
class Solution {
public:
    void bfs(vector<vector<char>>& grid, vector<vector<int>>& visited, int i, int j) {
        int n = grid.size();
        int m = grid[0].size();
        queue<pair<int, int>> q;
        visited[i][j] = 1;
        q.push({i, j});

        int dx[] = {1, -1, 0, 0};
        int dy[] = {0, 0, 1, -1};

        while (!q.empty()) {
            int x = q.front().first;
            int y = q.front().second;
            q.pop();

            for (int d = 0; d < 4; ++d) {
                int nx = x + dx[d];
                int ny = y + dy[d];

                if (nx >= 0 && nx < n && ny >= 0 && ny < m &&
                    grid[nx][ny] == '1' && !visited[nx][ny]) {
                    visited[nx][ny] = 1;
                    q.push({nx, ny});
                }
            }
        }
    }
```

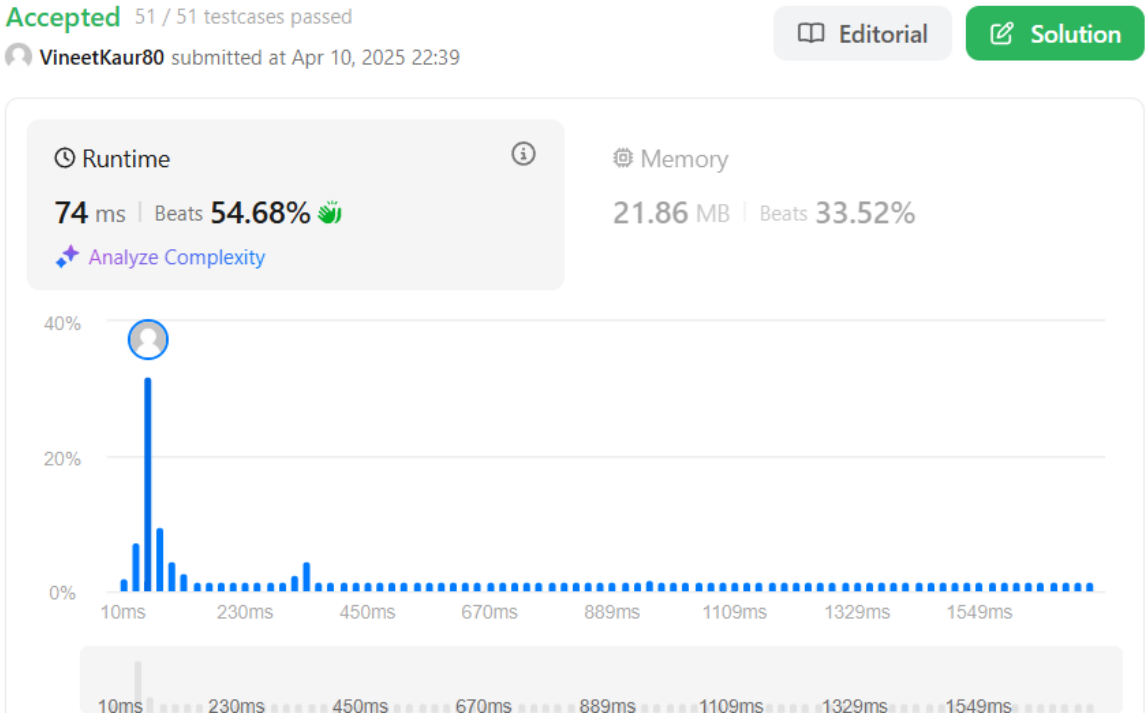
```
    int numIslands(vector<vector<char>>& grid) {
        int n = grid.size();
        int m = grid[0].size();
        vector<vector<int>> visited(n, vector<int>(m, 0));
        int count = 0;

        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < m; ++j) {
                if (grid[i][j] == '1' && !visited[i][j]) {
                    bfs(grid, visited, i, j);
                    count++;
                }
            }
        }
        return count;
    }
};
```



2. Word Ladder

```
class Solution {
public:
    int ladderLength(string s, string t, vector<string>& word) {

        unordered_map<string,int>m1;

        for(auto a:word){
            m1[a]++;
        }
        queue<pair<string,int>>q1;

        q1.push({s,1});
        while(!q1.empty()){

            string st=q1.front().first;
            int cost=q1.front().second;

            q1.pop();
            for(int i=0;i<st.size();i++){

                string str=st;
                for(char c='a';c<='z';c++){
                    str[i]=c;

                    if(str==t && m1.find(str)!=m1.end()){
                        //cout<<st<<" ";
                        return cost+1;
                    }

                    if(m1.find(str)!=m1.end()){
                        q1.push({str,cost+1});
```

```
                m1.erase(str);
            }
        }
    }
}

    return 0;

}
};
```

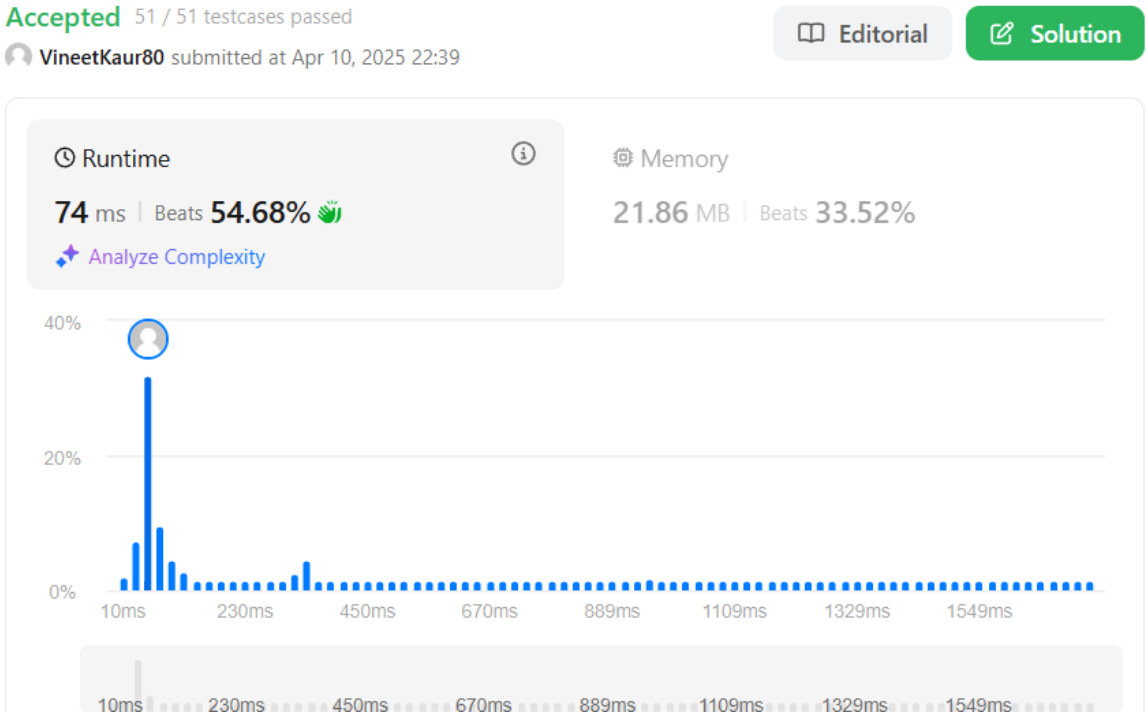


3. Surrounded Regions

```
class Solution {
private:
    void dfs(int i, int j, vector<vector<char>>& board) {
        int m = board.size();
        int n = board[0].size();
        board[i][j] = '#'; // Mark the cell as visited
```

```
        int x[] = {1, -1, 0, 0};
        int y[] = {0, 0, 1, -1};

        for (int k = 0; k < 4; k++) {
            int ni = i + x[k], nj = j + y[k];
            if (ni >= 0 && ni < m && nj >= 0 && nj < n && board[ni][nj] == 'O') {
                dfs(ni, nj, board);
            }
        }
    }

public:
    void solve(vector<vector<char>>& board) {
        int m = board.size();
        int n = board[0].size();

        // Step 1: Mark all boundary-connected 'O' cells with '#'
        for (int i = 0; i < n; i++) {
            if (board[0][i] == 'O') dfs(0, i, board);
            if (board[m - 1][i] == 'O') dfs(m - 1, i, board);
        }
        for (int i = 0; i < m; i++) {
            if (board[i][0] == 'O') dfs(i, 0, board);
            if (board[i][n - 1] == 'O') dfs(i, n - 1, board);
        }

        // Step 2: Replace remaining 'O' with 'X' and revert '#' back to 'O'
        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
                if (board[i][j] == '#') board[i][j] = 'O';
                else board[i][j] = 'X';
            }
```

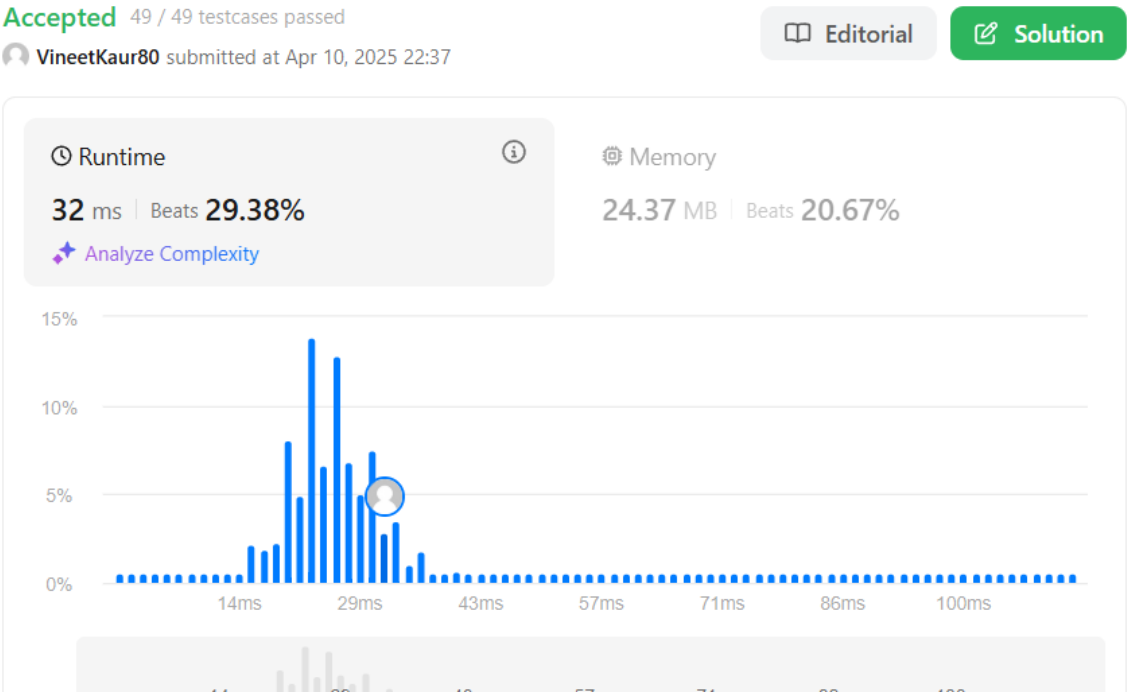
```
        }
    }
};
```



4. Binary Tree Maximum Path Sum

```
class Solution {
public:
    int height(TreeNode* root,int & maxi){
        if(root == nullptr)
            return 0;

        int lh =max(0,height(root->left, maxi));
        int rh = max(0,height(root->right, maxi));
        maxi = max(maxi, lh+rh+root->val);

        return max(rh,lh)+root->val;
    }
    int maxPathSum(TreeNode* root) {
        int maxi{INT_MIN};
```

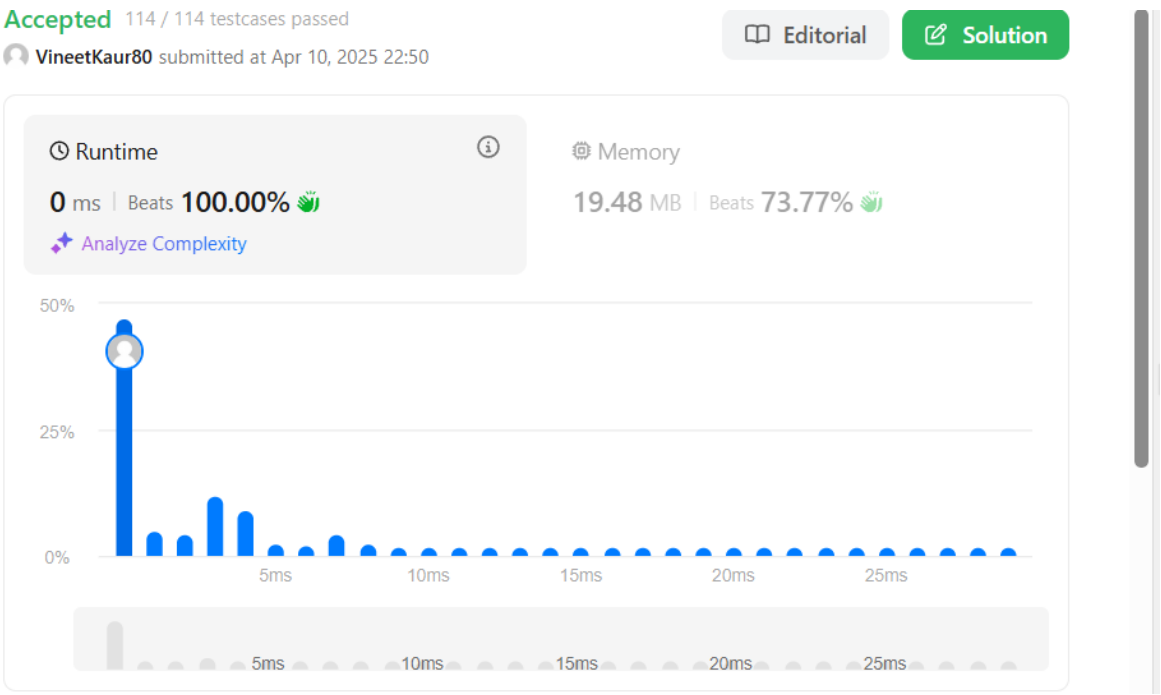
```
        height(root, maxi);
        return maxi;
    }
};
```



5. 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) {
```

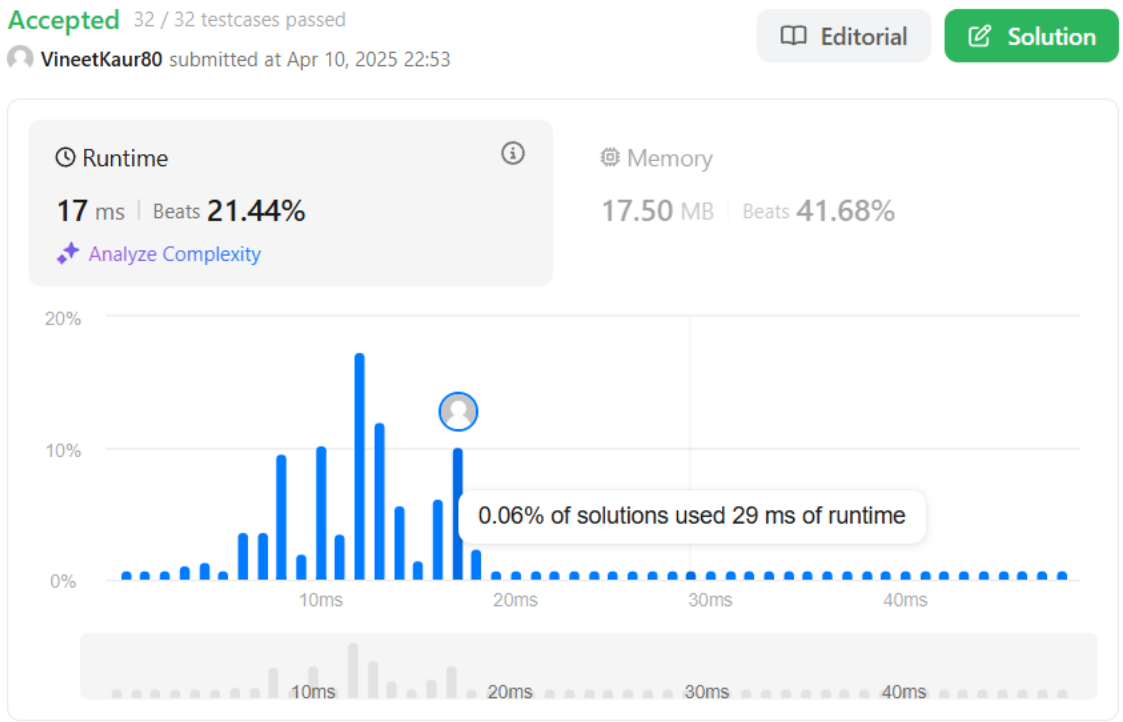
```
        int n = isConnected.size();
        vector<int> visited(n, 0);
        int count = 0;

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

        return count;
    }
};
```



6. Lowest Common Ancestor of a Binary Tree

```
class Solution {
public:
```

```
    TreeNode* lowestCommonAncestor(TreeNode* root, TreeNode* p, TreeNode* q) {
        if(!root)
            return NULL;

        if(root->val == p->val || root->val == q->val)
            return root;

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

        if(l && r)
            return root;

        return l?l:r;
    }
};
```



9. Course Schedule

```
class Solution {
```

```
public:
    bool canFinish(int numCourses, vector<vector<int>>& prerequisites) {
        vector<int> v(numCourses,0);
        map<int,vector<int>> m;
        for(int j=0;j<prerequisites.size();j++)
        {
            v[prerequisites[j][0]]++;
            m[prerequisites[j][1]].push_back(prerequisites[j][0]);
        }
        vector<int> s;
        vector<bool> vis(numCourses,false);
        for(int j=0;j<v.size();j++)
        {
            if(v[j] == 0)
            {
                s.push_back(j);
                vis[j]=true;
            }
        }
        while(s.size() != 0)
        {
            int p=s.back();
            s.pop_back();
            for(int j=0;j<m[p].size();j++)
            {
                if(!vis[m[p][j]])
                {
                    v[m[p][j]]--;
                    if(v[m[p][j]] == 0)
                    {
                        s.push_back(m[p][j]);
                        vis[m[p][j]]=true;
```

```
            }
        }
    }
}
for(int j=0;j<vis.size();j++)
{
    if(!vis[j])
    {
        return false;
    }
}
return true;
}
};
```