Experiment 6

Student Name: Anshul Verma **UID: 22BCS11915**
Branch: CSE **Section/Group: 637-B**
Semester: 6th **Date of Performance:27/2/25**
Subject Name: Advanced Programming - 2 **Subject Code: 22CSH-351**

Ques 1: Aim:

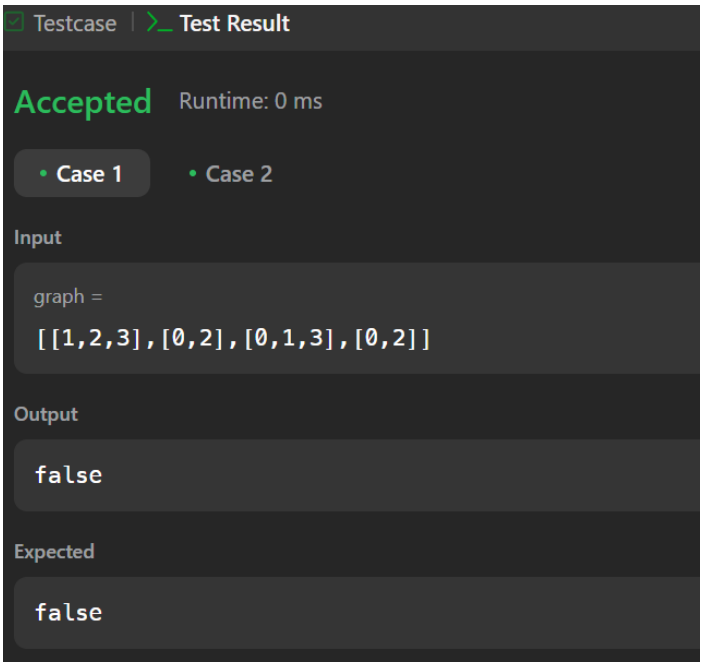
is graph bipartite?

Code:

```
#include <vector>
#include <queue>
using namespace std;

class Solution {
public:
    bool isBipartite(vector<vector<int>>& graph) {
        int n = graph.size();
        vector<int> color(n, -1); // -1 means uncolored, 0 and 1 are the two colors

        for (int i = 0; i < n; i++) {
            if (color[i] == -1) { // If the node is not colored, perform BFS
                queue<int> q;
                q.push(i);
                color[i] = 0; // Start coloring with 0

                while (!q.empty()) {
                    int node = q.front();
                    q.pop();

                    for (int neighbor : graph[node]) {
                        if (color[neighbor] == -1) {
                            color[neighbor] = 1 - color[node]; // Assign the opposite color
```

```
                    q.push(neighbor);
                } else if (color[neighbor] == color[node]) {
                    return false; // If two adjacent nodes have the same color, the graph is not
bipartite
                }
            }
        }
    }
    return true;
  }
};
```

Submission Screenshot:



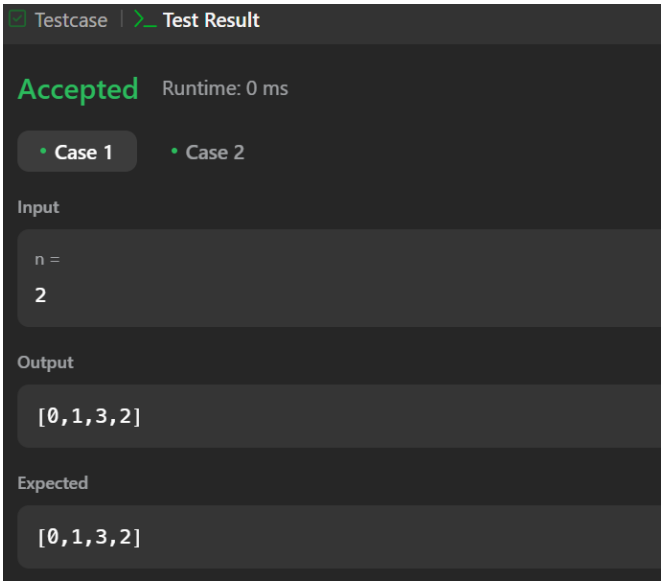
Ques 2:

Aim: Gray code

Code:

```
#include <vector>
using namespace std;

class Solution {
public:
    vector<int> grayCode(int n) {
        vector<int> result;
        int total = 1 << n; // 2^n
        for (int i = 0; i < total; i++) {
            result.push_back(i ^ (i >> 1)); // Generate Gray code
        }
        return result;
    }
};
```

Submission Screenshot:



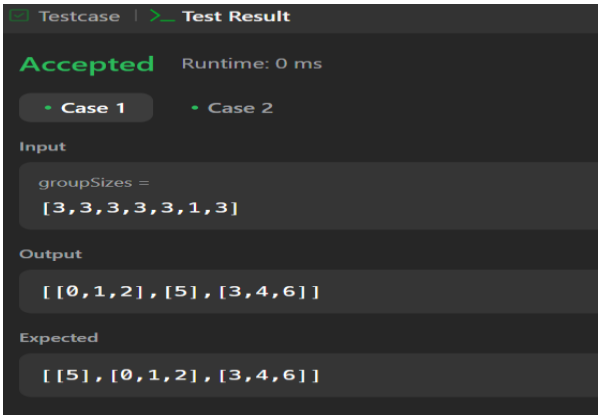
Ques 3:

Aim: Group the People Given the Group Size They Belong To

Code:

```
#include <vector>
#include <unordered_map>
using namespace std;

class Solution {
public:
    vector<vector<int>> groupThePeople(vector<int>& groupSizes) {
        unordered_map<int, vector<int>> groups;
        vector<vector<int>> result;

        for (int i = 0; i < groupSizes.size(); i++) {
            groups[groupSizes[i]].push_back(i);
            if (groups[groupSizes[i]].size() == groupSizes[i]) {
                result.push_back(groups[groupSizes[i]]);
                groups[groupSizes[i]].clear();
            }
        }
        return result;
    }
};
```

Submission Screenshot:



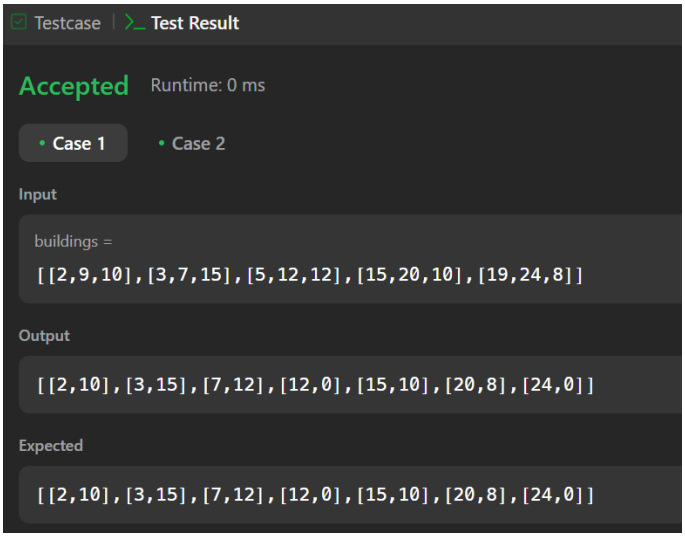
Ques 4:

Aim: The Skyline Problem

Code:

```
#include <vector>
#include <queue>
#include <set>
using namespace std;

class Solution {
public:
    vector<vector<int>> getSkyline(vector<vector<int>>& buildings) {
        vector<pair<int, int>> events;
        vector<vector<int>> result;

        for (auto& b : buildings) {
            events.emplace_back(b[0], -b[2]); // Start of building
            events.emplace_back(b[1], b[2]); // End of building
        }
        sort(events.begin(), events.end());
        multiset<int> heights = {0};
        int prevHeight = 0;

        for (auto& e : events) {
            if (e.second < 0) {
                heights.insert(-e.second);
            } else {
                heights.erase(heights.find(e.second));
            }
            int currHeight = *heights.rbegin();
            if (currHeight != prevHeight) {
                result.push_back({e.first, currHeight});
                prevHeight = currHeight;
            }
        }
        return result;
    }
};
```

Submission Screenshot:



Ques 5:

Aim: Find the Difference

Code:

```
#include <vector>
#include <queue>
#include <set>
using namespace std;

class Solution {
public:
    char findTheDifference(string s, string t) {
        char result = 0;
        for (char c : s) {
            result ^= c;
        }
        for (char c : t) {
            result ^= c;
        }
        return result;
```

```
    }
};
```

Submission Screenshot:

Ques 6:

Aim: Predict the Winner

Code:

```
 #include <vector>
#include <queue>
#include <set>
using namespace std;

class Solution {
```

```
public:
    bool
        predictTheWinner
        (vector<int>&
        nums) {
        int n = nums.size();
        vector<vector<int>>
        dp(n,
        vector<int>(n,
        0));

        for (int i = 0; i < n;
        i++) {
        dp[i][i] = nums[i];
        }

        for (int len = 1; len
        < n; len++) {
        for (int i = 0; i < n
        - len; i++) {
            int j = i + len;
            dp[i][j] =
        max(nums[i] -
        dp[i + 1][j],
        nums[j] - dp[i][j -
        1]);
        }
        }

        return dp[0][n - 1]
        >= 0;
    }
};
```

Submission Screenshot:

Ques 7:

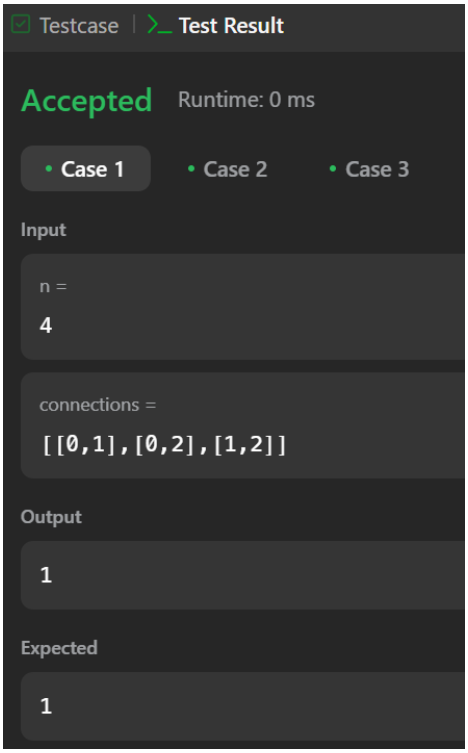
Aim: Number of Operations to Make Network Connected

Code:

```
#include <vector>
#include <queue>
#include <set>
using namespace std;

class Solution {
public:
    int makeConnected(int n, vector<vector<int>>& connections) {
        if (connections.size() < n - 1) return -1; // Not enough cables to connect all computers

        vector<vector<int>> adj(n);
```

```
        for (auto& conn : connections) {
            adj[conn[0]].push_back(conn[1]);
            adj[conn[1]].push_back(conn[0]);
        }

        vector<bool> visited(n, false);
        int components = 0;

        function<void(int)> dfs = [&](int node) {
            visited[node] = true;
            for (int neighbor : adj[node]) {
                if (!visited[neighbor]) {
                    dfs(neighbor);
                }
            }
        };

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

        return components - 1;
    }
};
```

Submission Screenshot:



Ques 8:

Aim: Critical Connections in a Network

Code:

```
#include <vector>
#include <algorithm>
using namespace std;

class Solution {
public:
    vector<vector<int>> criticalConnections(int n, vector<vector<int>>& connections) {
        vector<vector<int>> adj(n), result;
        vector<int> discovery(n, -1), lowest(n, -1);
        int time = 0;
```

```
        for (auto& conn : connections) {
            adj[conn[0]].push_back(conn[1]);
            adj[conn[1]].push_back(conn[0]);
        }

        function<void(int, int)> dfs = [&](int node, int parent) {
            discovery[node] = lowest[node] = time++;
            for (int neighbor : adj[node]) {
                if (neighbor == parent) continue;
                if (discovery[neighbor] == -1) {
                    dfs(neighbor, node);
                    lowest[node] = min(lowest[node], lowest[neighbor]);
                    if (lowest[neighbor] > discovery[node]) {
                        result.push_back({node, neighbor});
                    }
                } else {
                    lowest[node] = min(lowest[node], discovery[neighbor]);
                }
            }
        };

        for (int i = 0; i < n; i++) {
            if (discovery[i] == -1) {
                dfs(i, -1);
            }
        }

        return result;
    }
};
```

Submission Screenshot: