Experiment2.3

Student Name: Virat Samdarshi **UID: 22BCS12648**
Branch: B.E(CSE) **Section/Group: IOT_627-B**
Semester: Fifth **Date of Performance: 19/09/24**
Subject Name: AP LAB 1 **Subject Code: 22CSP-314**

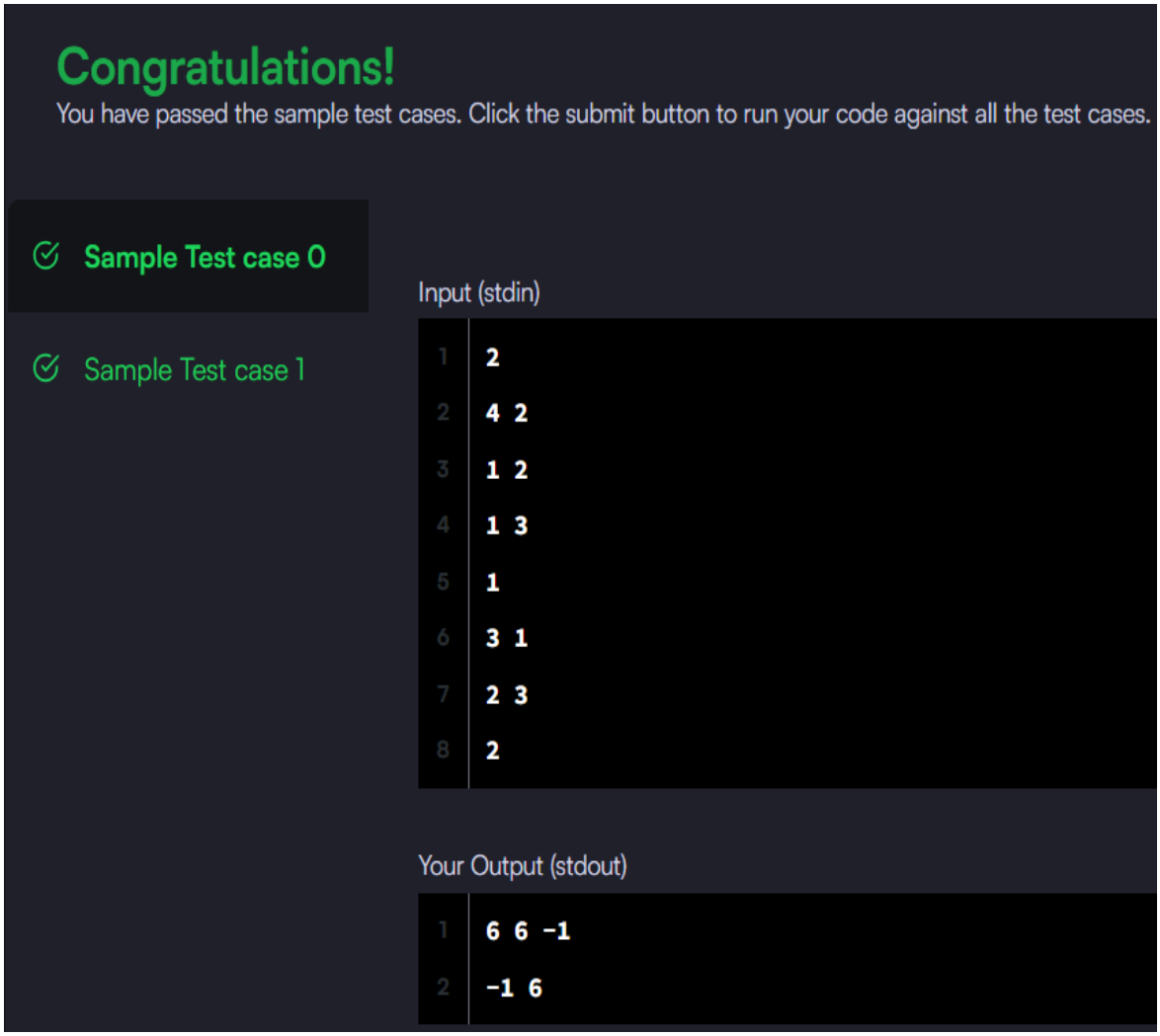
1. **Aim:** Consider an undirected graph where each edge weighs 6 units. Each of the nodes is labeled consecutively from 1 to n.
   You will be given a number of queries. For each query, you will be given a list of edges describing an undirected graph. After you create a representation of the graph, you must determine and report the shortest distance to each of the other nodes from a given starting position using the breadth-first search algorithm (BFS). Return an array of distances from the start node in node number order. If a node is unreachable, return -1 for that node.
2. **Objective:** The objective of the bfs function is to compute the shortest distances from a specified starting node to all other nodes in an undirected, unweighted graph represented by an adjacency list.
3. **Implementation/Code :-**

```
vector<int> bfs(int n, int m, vector<vector<int>> edges, int s) {
vector<vector<int>> adj(n + 1);
   for (const auto& edge : edges) {
      int u = edge[0];
      int v = edge[1];
      adj[u].push_back(v);
      adj[v].push_back(u);
   }
   vector<int> distances(n + 1, -1);
   distances[s] = 0;
   queue<int> q;
   q.push(s);
   while (!q.empty()) {
      int node = q.front();
      q.pop();
      for (int neighbor : adj[node]) {
         if (distances[neighbor] == -1) {
            distances[neighbor] = distances[node] + 6;
            q.push(neighbor);
         }
```

```
        }
    }
    vector<int> result;
    for (int i = 1; i <= n; ++i) {
        if (i != s) {
            result.push_back(distances[i]);
        }
    }
    return result;
}
```

```
vector<int> bfs(int n, int m, vector<vector<int>> edges, int s) {
vector<vector<int>> adj(n + 1);
    for (const auto& edge : edges) {
        int u = edge[0];
        int v = edge[1];
        adj[u].push_back(v);
        adj[v].push_back(u);}
    vector<int> distances(n + 1, -1);
    distances[s] = 0;
    queue<int> q;
    q.push(s);
    while (!q.empty()) {
        int node = q.front();
        q.pop();
        for (int neighbor : adj[node]) {
            if (distances[neighbor] == -1) {
                distances[neighbor] = distances[node] + 6;
                q.push(neighbor);} } }
    vector<int> result;
    for (int i = 1; i <= n; ++i) {
        if (i != s) {
            result.push_back(distances[i]);
        }
    }
    return result;}
```

4. Output :-



5. Time Complexity: O(n+m)

PROBLEM 2

1.Aim The member states of the UN are planning to send 2 people to the moon. They want them to be from different countries. You will be given a list of pairs of astronaut ID's. Each pair is made of astronauts from the same country. Determine how many pairs of astronauts from different countries they can choose from.

2.Objective: The objective of the journeyToMoon function is to calculate the number of valid pairs of astronauts that can be chosen from different countries, given a list of astronaut pairs that represent connections between them.

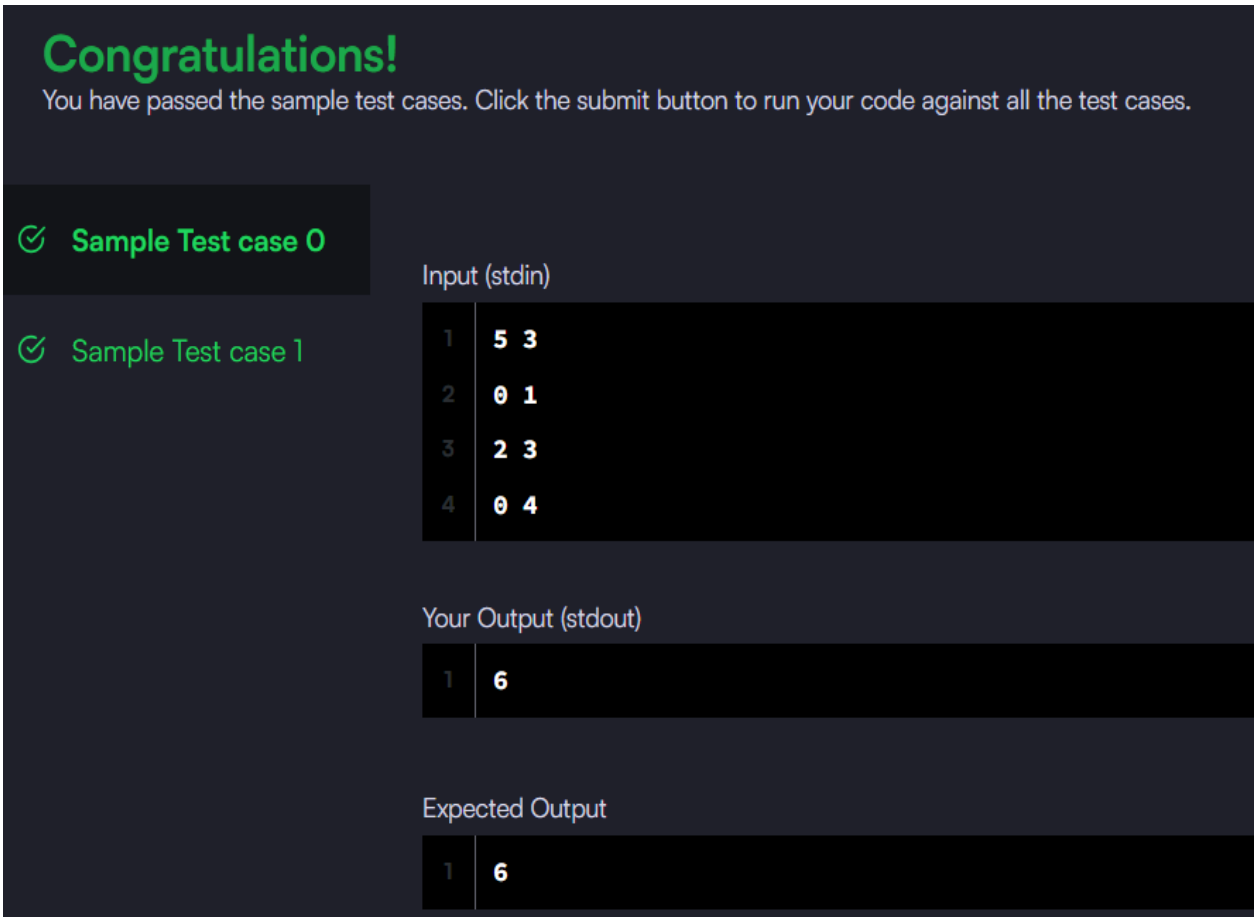
3.Implementation/Code :-

```
int journeyToMoon(int n, vector<vector<int>> astronaut) {
  vector<vector<int>> adj(n);
  for (const auto& pair : astronaut) {
     adj[pair[0]].push_back(pair[1]);
     adj[pair[1]].push_back(pair[0]);
  }
  vector<bool> visited(n, false);
  vector<int> countrySizes;
  for (int i = 0; i < n; i++) {
     if (!visited[i]) {
        int count = 0;
        queue<int> q;
        q.push(i);
        visited[i] = true;
        while (!q.empty()) {
           int astronaut = q.front();
           q.pop();
           count++;
           for (int neighbor : adj[astronaut]) {
              if (!visited[neighbor]) {
                 visited[neighbor] = true;
                 q.push(neighbor);
              }
           }
        }
        countrySizes.push_back(count);
     }
```

```
    }
    long long totalPairs = 0;
    long long totalAstronauts = n;
    long long combinations = (totalAstronauts * (totalAstronauts - 1)) / 2;
    for (int size : countrySizes) {
        totalPairs += (size * (size - 1)) / 2;
    }
    return combinations - totalPairs;

}
int journeyToMoon(int n, vector<vector<int>> astronaut) {
    vector<vector<int>> adj(n);
    for (const auto& pair : astronaut) {
        adj[pair[0]].push_back(pair[1]);
        adj[pair[1]].push_back(pair[0]);}
    vector<bool> visited(n, false);
    vector<int> countrySizes;
    for (int i = 0; i < n; i++) {
        if (!visited[i]) {
            int count = 0;
            queue<int> q;
            q.push(i);
            visited[i] = true;
            while (!q.empty()) {
                int astronaut = q.front();
                q.pop();
                count++;
  for (int neighbor : adj[astronaut]) {
  if (!visited[neighbor]) {
  visited[neighbor] = true;
  q.push(neighbor);}
countrySizes.push_back(count);}}
    long long totalPairs = 0;
    long long totalAstronauts = n;
    long long combinations = (totalAstronauts * (totalAstronauts - 1)) / 2;
    for (int size : countrySizes) {
        totalPairs += (size * (size - 1)) / 2;
    }
    return combinations - totalPairs;
```

4. Output :-



5. Time Complexity: O(n+m)