Course Name- AP-I Course Code- 21CSP-314

Experiment-2.1

Aim- To implement the concept of Graphs.

Objectives- The objective of this program is to understand the concept of Graph and how to implement it in programs.

Problem1: https://www.hackerrank.com/challenges/bfsshortreach/problem?isFullScreen=true

Problem2: https://www.hackerrank.com/challenges/the-quickest-way-up/problem?isFullScreen=true

Description-

A graph can be defined as a group of vertices and edges that are used to connect these vertices. A graph can be seen as a cyclic tree, where the vertices (Nodes) maintain any complex relationship among them instead of having a parent-child relationship.

A graph can be directed or undirected. However, in an undirected graph, edges are not associated with the directions with them.

In a directed graph, edges form an ordered pair. Edges represent a specific path from some vertex A to another vertex B. Node A is called the initial node while node B is called the terminal node.

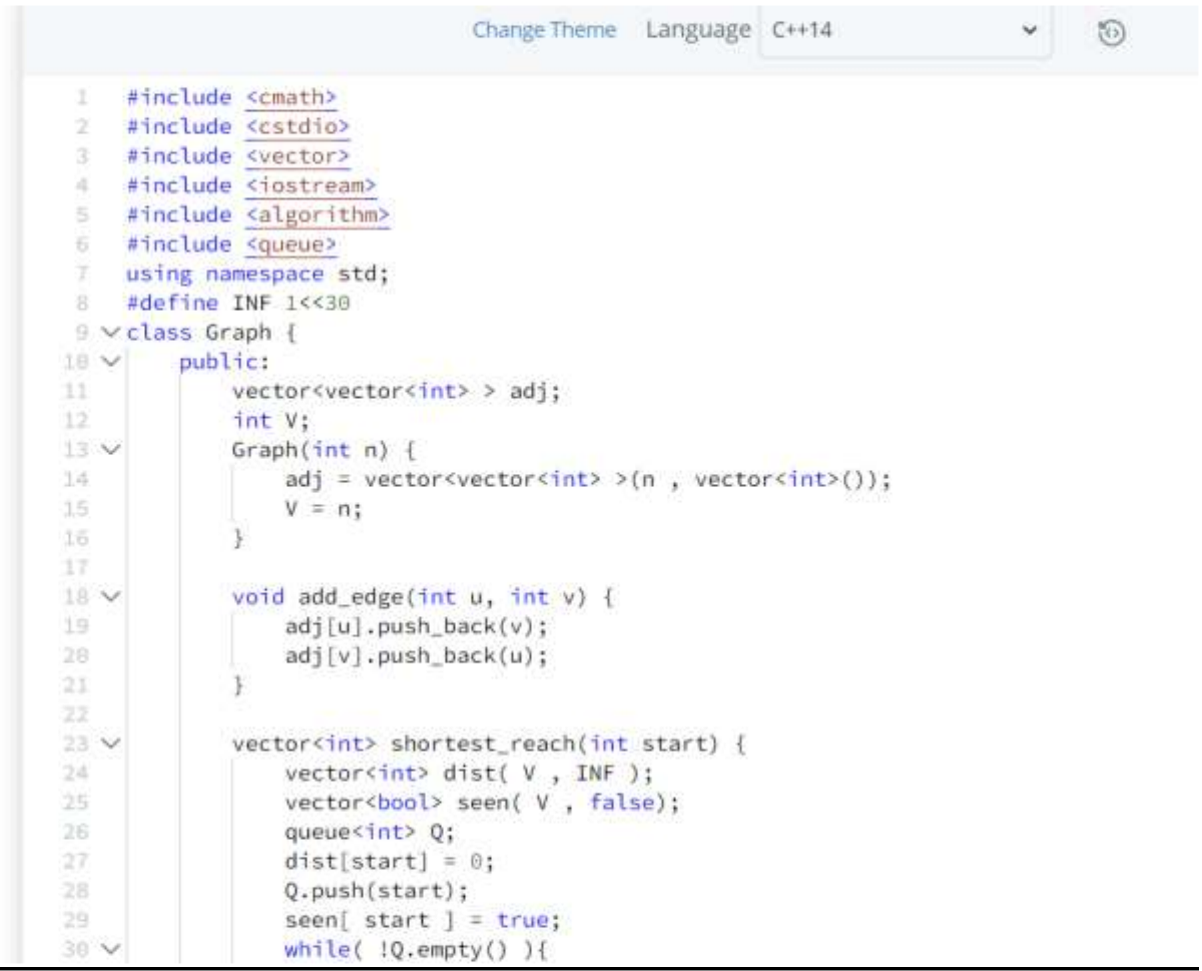
Problem 1: Breadth First Search: Shortest Reach

Problem 2: Snakes and Ladders: The Quickest Way Up

Course Name- AP-I Course Code- 21CSP-314

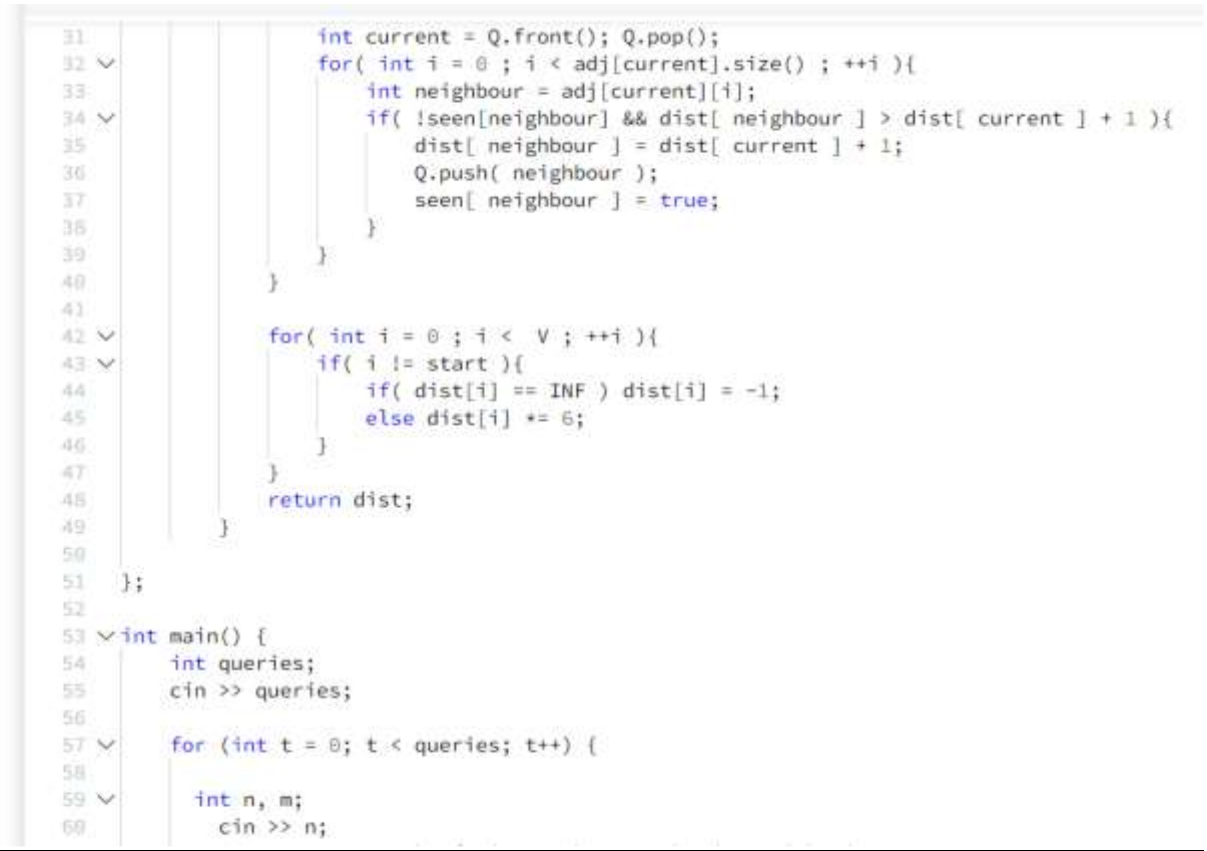
Code:

1.

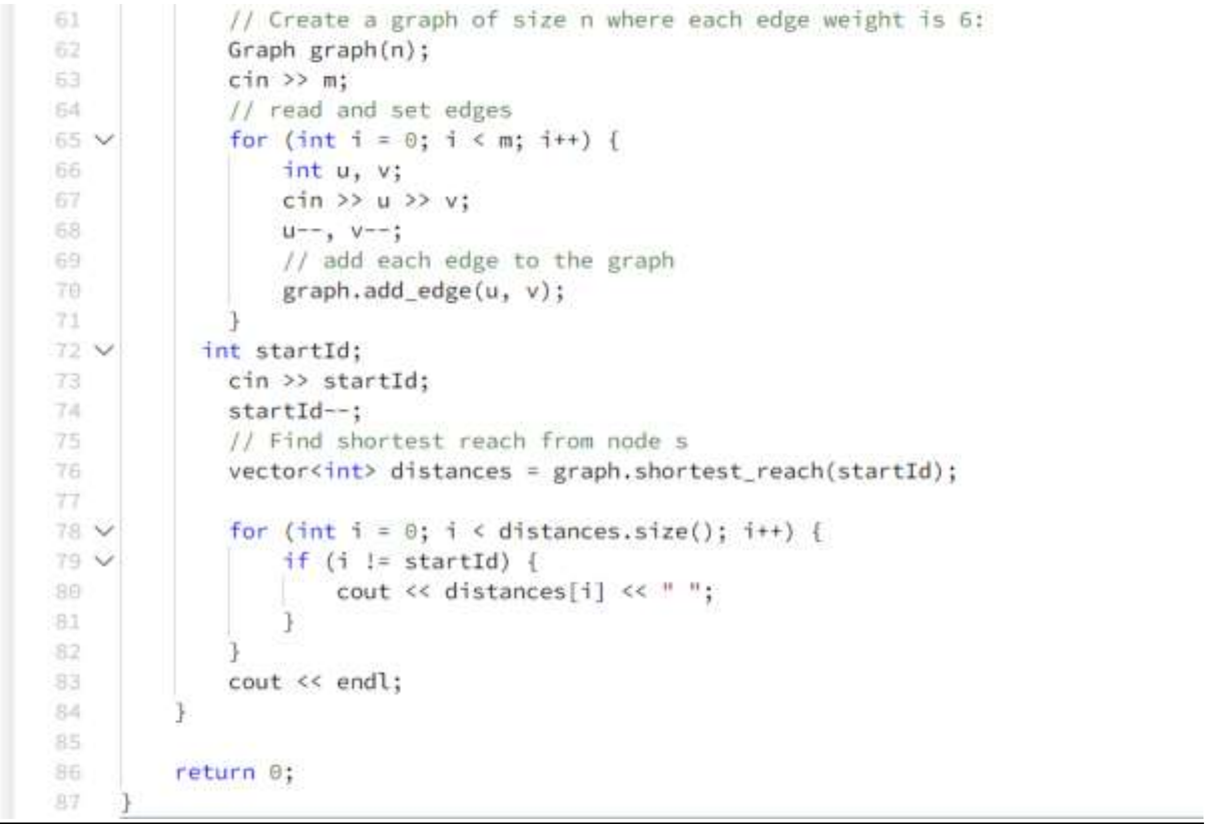


```
#include <cmath>
#include <cstdio>
#include <vector>
#include <iostream>
#include <algorithm>
#include <queue>
using namespace std;
#define INF 1<<30
class Graph {
    public:
        vector<vector<int> > adj;
        int V;
        Graph(int n) {
            adj = vector<vector<int> >(n , vector<int>());
            V = n;
        }

        void add_edge(int u, int v) {
            adj[u].push_back(v);
            adj[v].push_back(u);
        }

        vector<int> shortest_reach(int start) {
            vector<int> dist( V , INF );
            vector<bool> seen( V , false);
            queue<int> Q;
            dist[start] = 0;
            Q.push(start);
            seen[ start ] = true;
            while( !Q.empty() ){
```

Course Name- AP-I Course Code- 21CSP-314



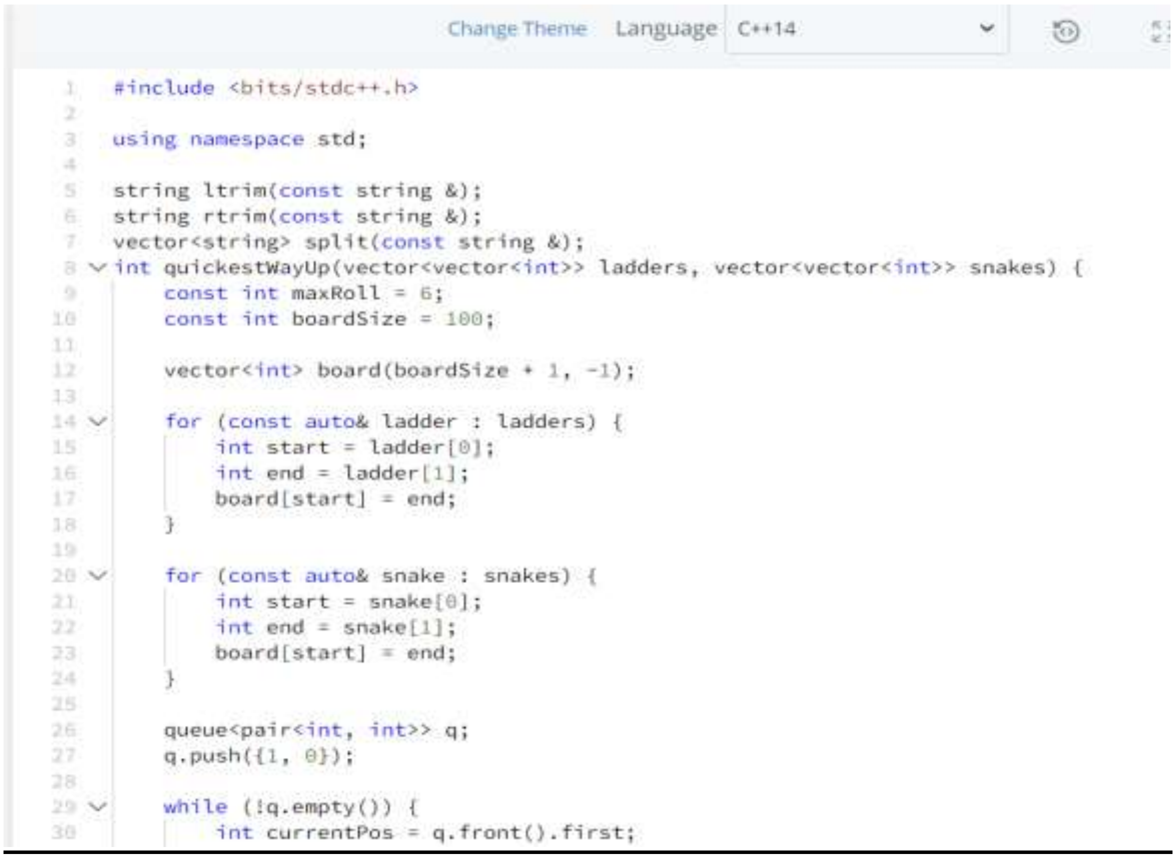
```
                int current = Q.front(); Q.pop();
                for( int i = 0 ; i < adj[current].size() ; ++i ){
                    int neighbour = adj[current][i];
                    if( !seen[neighbour] && dist[ neighbour ] > dist[ current ] + 1 ){
                        dist[ neighbour ] = dist[ current ] + 1;
                        Q.push( neighbour );
                        seen[ neighbour ] = true;
                    }
                }
            }

            for( int i = 0 ; i <  V ; ++i ){
                if( i != start ){
                    if( dist[i] == INF ) dist[i] = -1;
                    else dist[i] += 6;
                }
            }
            return dist;
        }

};

int main() {
    int queries;
    cin >> queries;

    for (int t = 0; t < queries; t++) {

      int n, m;
        cin >> n;
```



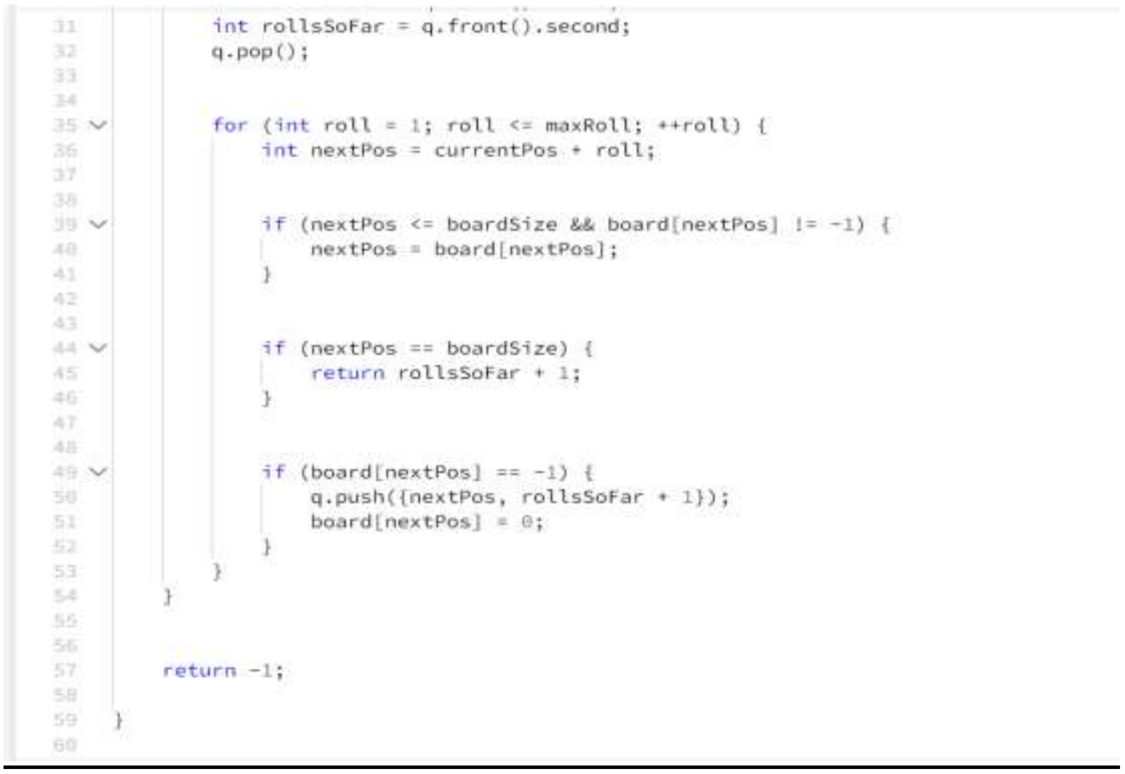
```
        // Create a graph of size n where each edge weight is 6:
        Graph graph(n);
        cin >> m;
        // read and set edges
        for (int i = 0; i < m; i++) {
            int u, v;
            cin >> u >> v;
            u--, v--;
            // add each edge to the graph
            graph.add_edge(u, v);
        }
      int startId;
        cin >> startId;
        startId--;
        // Find shortest reach from node s
        vector<int> distances = graph.shortest_reach(startId);

        for (int i = 0; i < distances.size(); i++) {
            if (i != startId) {
                cout << distances[i] << " ";
            }
        }
        cout << endl;
    }

    return 0;
}
```

Course Name- AP-I Course Code- 21CSP-314

2.



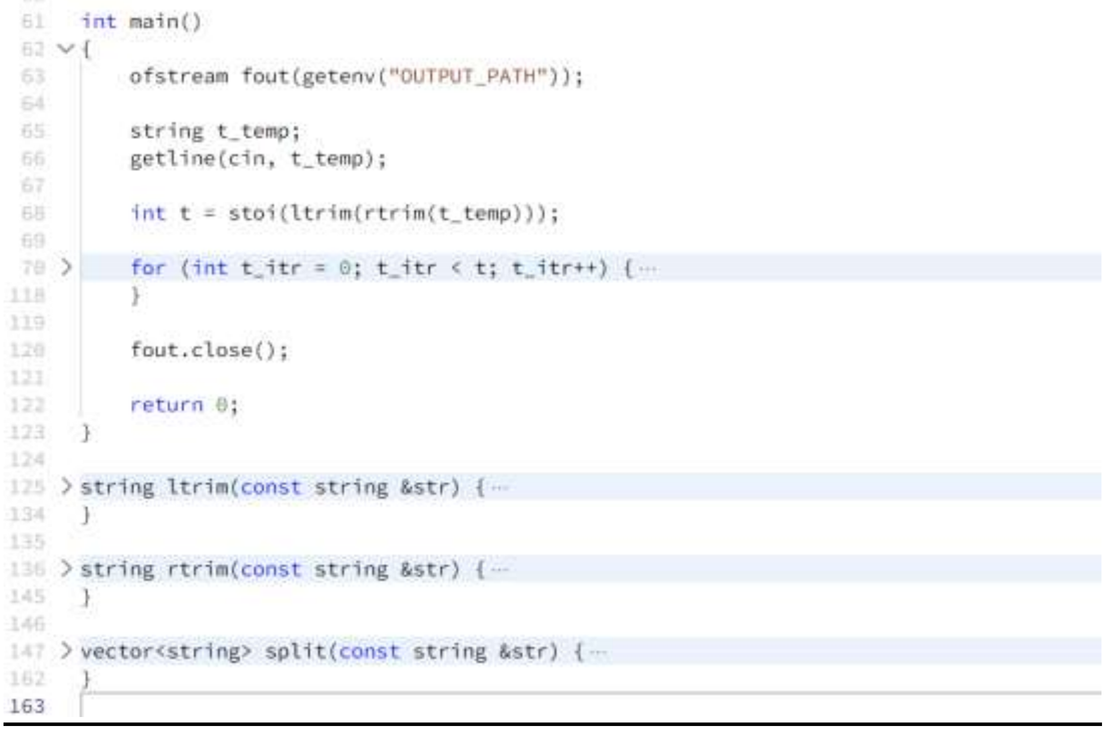
```cpp
#include <bits/stdc++.h>

using namespace std;

string ltrim(const string &);
string rtrim(const string &);
vector<string> split(const string &);
int quickestWayUp(vector<vector<int>> ladders, vector<vector<int>> snakes) {
    const int maxRoll = 6;
    const int boardSize = 100;

    vector<int> board(boardSize + 1, -1);

    for (const auto& ladder : ladders) {
        int start = ladder[0];
        int end = ladder[1];
        board[start] = end;
    }

    for (const auto& snake : snakes) {
        int start = snake[0];
        int end = snake[1];
        board[start] = end;
    }

    queue<pair<int, int>> q;
    q.push({1, 0});

    while (!q.empty()) {
        int currentPos = q.front().first;
```



```cpp
        int rollsSoFar = q.front().second;
        q.pop();


        for (int roll = 1; roll <= maxRoll; ++roll) {
            int nextPos = currentPos + roll;


            if (nextPos <= boardSize && board[nextPos] != -1) {
                nextPos = board[nextPos];
            }


            if (nextPos == boardSize) {
                return rollsSoFar + 1;
            }


            if (board[nextPos] == -1) {
                q.push({nextPos, rollsSoFar + 1});
                board[nextPos] = 0;
            }
        }
    }


    return -1;

}
```

Name- Manish Singh Barolia UID- 21BCS5712

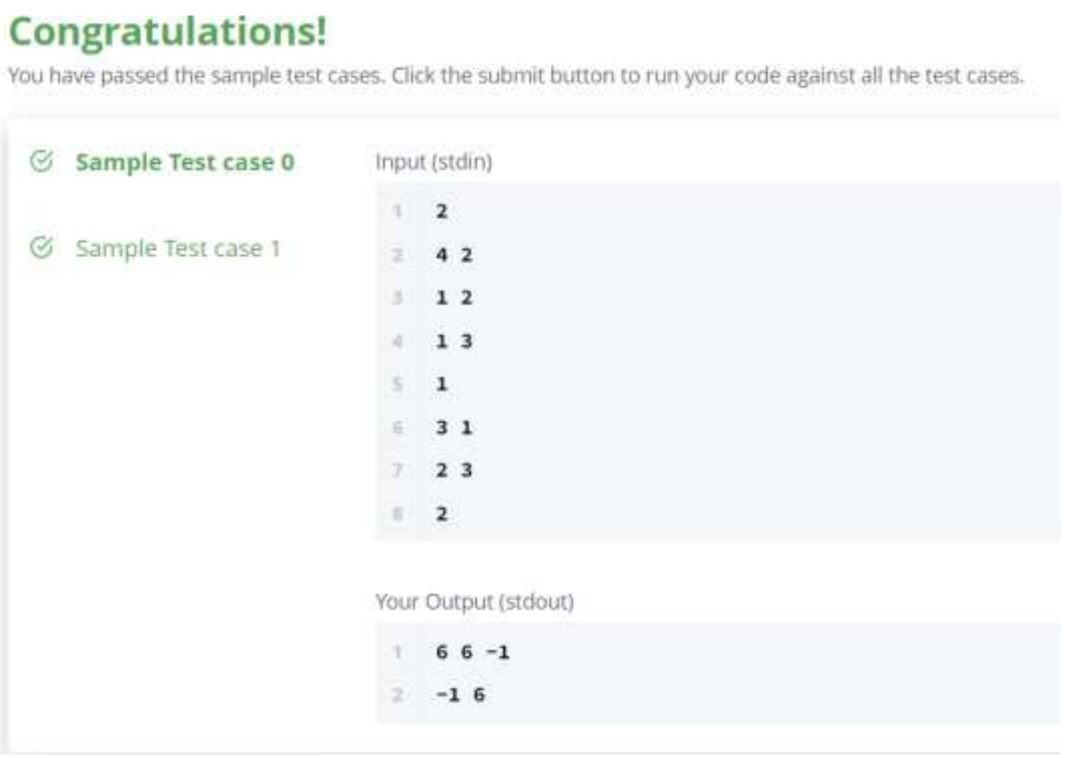
Course Name- AP-I   Course Code- 21CSP-314



```
int main()
{
    ofstream fout(getenv("OUTPUT_PATH"));

    string t_temp;
    getline(cin, t_temp);

    int t = stoi(ltrim(rtrim(t_temp)));

    for (int t_itr = 0; t_itr < t; t_itr++) {···
    }

    fout.close();

    return 0;
}

string ltrim(const string &str) {···
}

string rtrim(const string &str) {···
}

vector<string> split(const string &str) {···
}
```

Outcome-

Problem 1 outcome



Congratulations!

You have passed the sample test cases. Click the submit button to run your code against all the test cases.

Input (stdin)

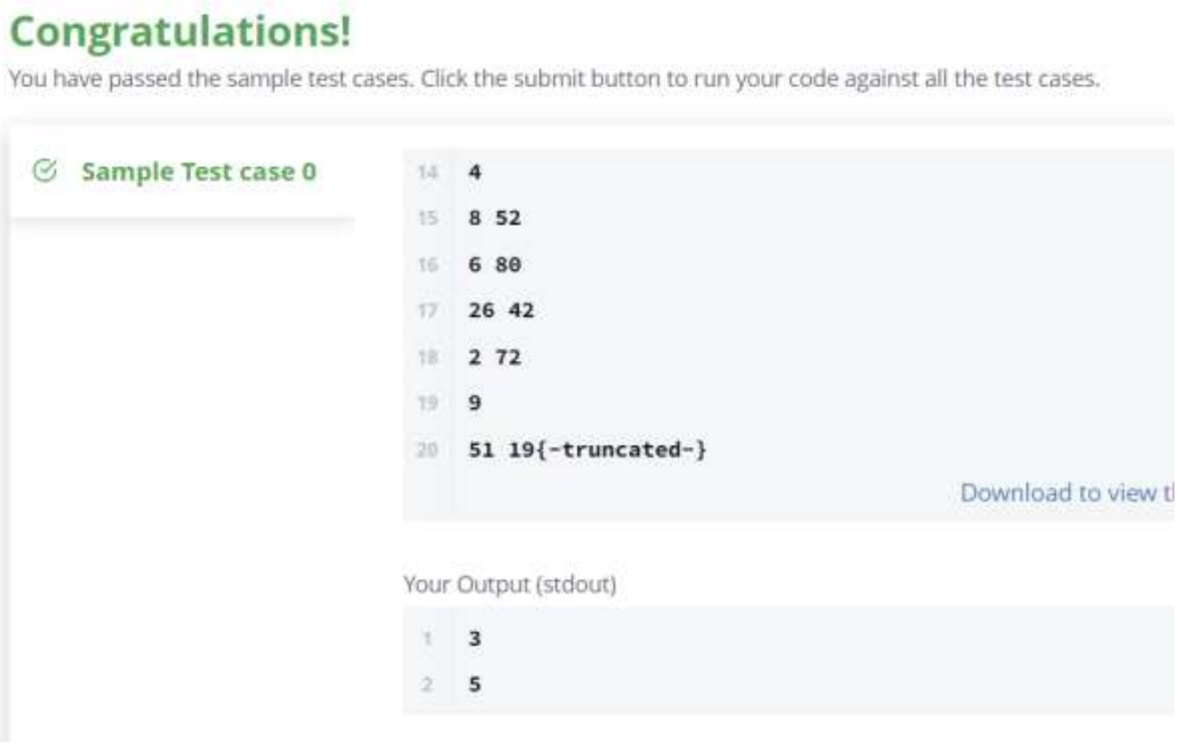
```
2
4 2
1 2
1 3
1
3 1
2 3
2
```

Your Output (stdout)

```
6 6 -1
-1 6
```

Course Name- AP-I Course Code- 21CSP-314

Problem 2 Outcome



Congratulations!

You have passed the sample test cases. Click the submit button to run your code against all the test cases.

Sample Test case 0

```
4
8 52
6 80
26 42
2 72
9
51 19{-truncated-}
```

Download to view t

Your Output (stdout)

```
3
5
```

Learning Outcomes-

1. Learnt about the graph data structure.
2. Learnt about bradth first search.
3. Learnt to traverse the graph using bfs.