Experiment 2.1

Student Name: UTKARSH JOSHI **UID: 21BCS9158**
Branch: CSE **Section/Group: 802/A**
Semester: 5th **Date of Performance:14/08/23**
Subject: Advanced Programming **SubjectCode: 21CSP-314**

Aim: To implement the code on graph on hacker rank .

Objective: To understand the concept of graph .

Breadth First Search: Shortest Reach

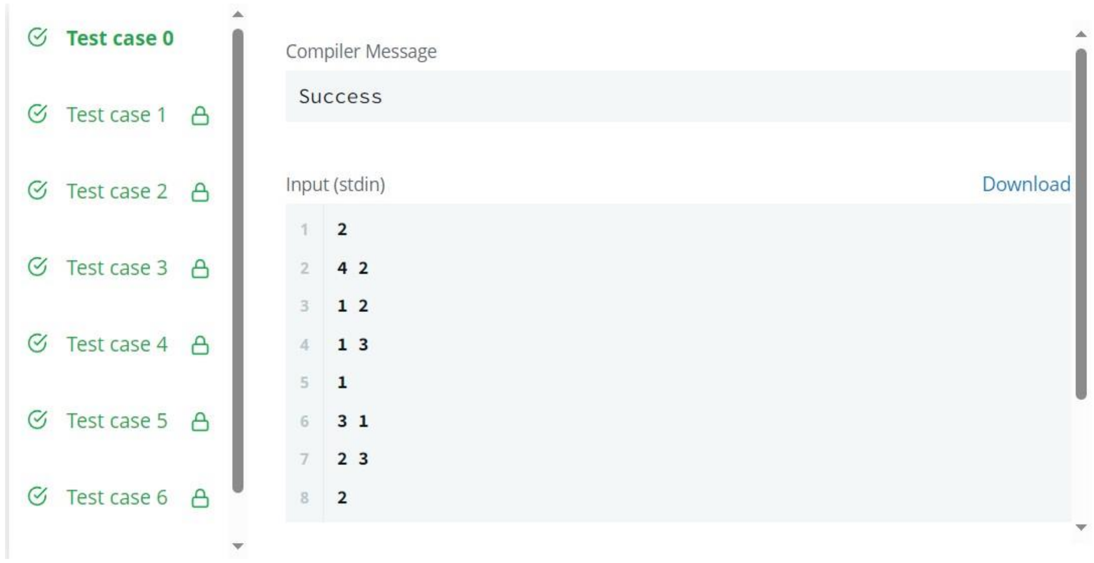
Source Code

```
#include <bits/stdc++.h>
 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() ){
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 ] > dis t[
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;
}
```

Output



Test case 0
Test case 1
Test case 2
Test case 3
Test case 4
Test case 5
Test case 6

Compiler Message

```
Success
```

Input (stdin) Download

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

Snakes and Ladders: The Quickest Way Up Source code

```
#include <bits/stdc++.h>
 using namespace
std;
 vector<pair<int, int>>
ladders; vector<pair<int, int>>
snakes; vector<int> distances;
 int HandleSnake(int x){     auto iter = find_if(snakes.begin(),
snakes.end(), [x](pair<int,  int> snake){          return
snake.first == x;
    });            if (iter ==
snakes.end())          return
x;
            return iter-
>second;
```

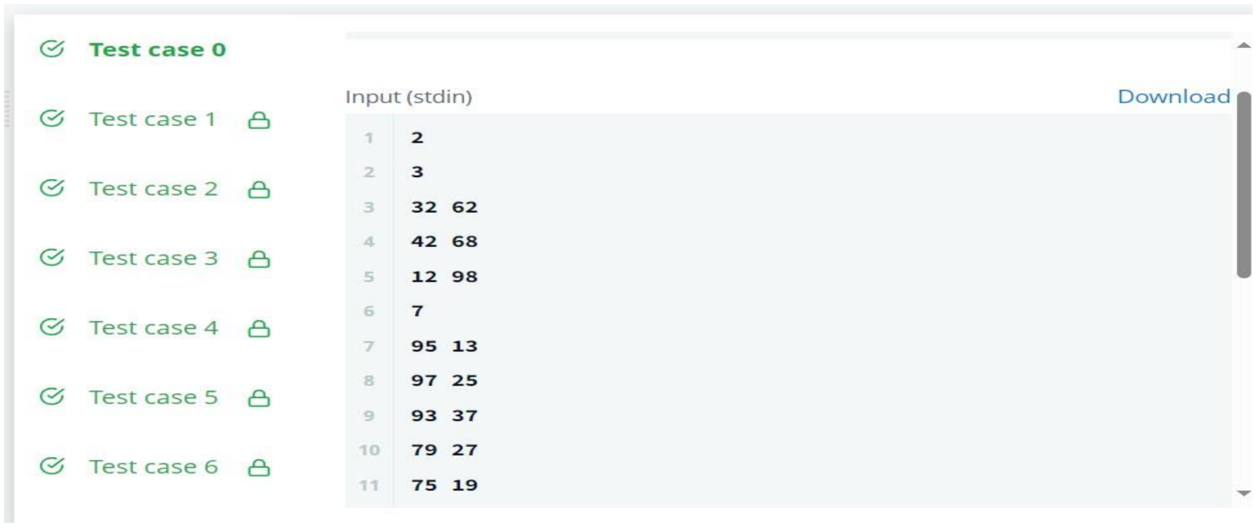
```
}  int HandleLadder(int x){      auto iter =
find_if(ladders.begin(), ladders.end(), [x](pair<in t, int>
ladder){         return ladder.first == x;
    });
    if (iter == ladders.end())
return x;
         return iter-
>second;
}  int
BFS(){
    distances.clear();
distances.resize(101, INT_MAX);
queue<int> q;     q.emplace(1);
distances.at(1) = 0;
while(!q.empty()){
                 int curr =
q.front();
        q.pop();
                          for(int roll = 1;
roll <= 6 ; ++roll){

auto dest = curr + roll;
if (dest > 100)
continue;
                         dest =
HandleSnake(dest);               dest =
HandleLadder(dest);
                         if (distances.at(dest) >
distances.at(curr)+1){
                     distances.at(dest) =
distances.at(curr)+1;
                q.emplace(dest);
            }
        }      }          auto result =
distances.back();     result = (result ==
INT_MAX) ? -1 : result;     return result;
}  void
DoTestCase(){
         int num_ladders;     cin >>
num_ladders;     ladders.resize(num_ladders);
for(auto& ladder : ladders){         cin >>
ladder.first >> ladder.second;      }
int num_snakes;      cin >> num_snakes;
snakes.resize(num_snakes);
     for(auto& snake : snakes){         cin
>> snake.first >> snake.second;
    }          cout <<
BFS() << endl;
}  int main() {      int t;
cin >> t;       for(int i = 0 ; i
< t ; ++i){

        DoTestCase();

}      return
0;
}
```

Output



```
Test case 0
Test case 1
Test case 2
Test case 3
Test case 4
Test case 5
Test case 6
```

Input (stdin) Download

```
2
3
32 62
42 68
12 98
7
95 13
97 25
93 37
79 27
75 19
```