Experiment 2.1

Student Name: Nishant Kumar mehta **UID: 21BCS3402**
Branch: CSE **Section/Group: IOT - 602**
Semester: 5th **Dateof Performance: 15/09/23**
Subject Name: Advance Programming Lab **Subject Code: 21CSP - 314**

1. Aim: To implement the concept of Graphs

2. Objective:

- 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.
- Markov takes out his Snakes and Ladders game, stares at the board and wonders: "If I can always roll the die to whatever number I want, what would be the least number of rolls to reach the destination?"

  Rules The game is played with a cubic die of faces numbered to .

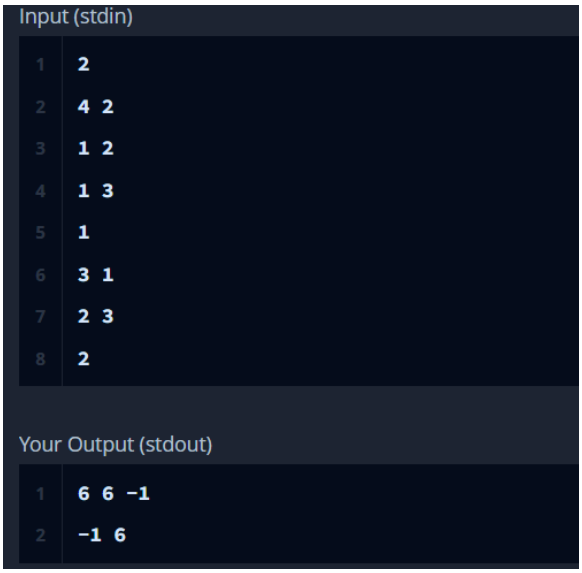
  1. Starting from square , land on square with the exact roll of the die. If moving the number rolled would place the player beyond square , no move is made.
  2. If a player lands at the base of a ladder, the player must climb the ladder. Ladders go up only.
  3. If a player lands at the mouth of a snake, the player must go down the snake and come out through the tail. Snakes go down only.

3. Program and output:

1. public static List<Integer> bfs(int n, int m, List<List<Integer>> edges, int s) {

```
List<List<Integer>> l=new ArrayList<>();
while(n-->0) l.add(new ArrayList<Integer>());
for(int i=0;i<edges.size();i++)
{
int a=edges.get(i).get(0);
int b=edges.get(i).get(1);
List<Integer> temp =  (l.get(a-1));
        if(!temp.contains(b-1))
                {
                temp.add(b-1);
                l.remove(a-1);
                l.add(a-1,temp);
                }
               temp =  (l.get(b-1));
        if(!temp.contains(a-1))
                {
                temp.add(a-1);
                l.remove(b-1);
                l.add(b-1,temp);
                }
}
int[] d=new int[l.size()];
Arrays.fill(d,0);
Boolean[] v=new Boolean[l.size()];
```

```
    Arrays.fill(v,false);
    Queue<Integer> q=new LinkedList<Integer>();
    q.add(s-1);
    v[s-1]=true;
    while(!q.isEmpty()){
       int p=q.poll();
       for(int i:l.get(p)){
          if(v[i]==false)
          {
                v[i]=true;
                q.add(i);
                d[i]=6+d[p];
          }
       }
    }
    List<Integer> r=new ArrayList<>();
    for(int i=0;i<d.length;i++){
       if(i==s-1) continue;
       else if(d[i]==0) r.add(-1);
       else r.add(d[i]);
    }
    return r;
    }
}
```



2.

```
public static int quickestWayUp(List<List<Integer>> ladders, List<List<Integer>> snakes) {
   Map<Integer,Integer> sm = new HashMap();
      Map<Integer,Integer> lm = new HashMap();
      for(List<Integer> l : ladders)
         lm.put(l.get(0),l.get(1));
      for(List<Integer> s : snakes)
         sm.put(s.get(0),s.get(1));
      Queue<Integer> q = new LinkedList();
      q.offer(1);
      int turns = 0;
      Set<Integer> set = new HashSet();
      while(!q.isEmpty()){
         int size = q.size();
         while(size-->0){
         int p = q.poll();
         if(p==100)
```

```
            return turns;
        if(set.contains(p))
            continue;
        set.add(p);
        int x = Math.min(p+6,100);
        for(int i=p+1;i<=x;i++){
            if(lm.containsKey(i)){
                q.add(lm.get(i));
            }else if(sm.containsKey(i)){
                q.add(sm.get(i));
            }else{
                q.add(i);
            }
        }
        }
        turns++;
    }
    return -1
} }
```