Algorithms Tutorial 4 Worksheet 3

For doing this problem, we use BFS.

First we create an array called S-L, of size N^2. If for cell i, there is no snakes or ladder on it, then S-L[i] = -1 else S-L[i] contains the value to which the snake or the ladder points to.

Define func Game (int S-L[], int N)
{ Initialise boolean array visited of size N^2, with all elements initially ~~false~~.
Initialise queue q.
∵ We are ~~the~~ at the first cell, mark visited[0] true.
Then enqueue the first cell along with its distance from first cell (here it is 0)

While (q is not empty)
{ int frontelement = q.front.
If frontelement == destination cell 0,
break;
If that is not the case then,
~~enqueue~~ enqueue its 6 neighbours (since there are 6 dice throws).

for (int i = frontelement + 1 ; i <= frontelement + 6 ; i++)
{ If that cell is visited (visited[i] = true)
{break;}
else, visit the cell
{ int dist = distance of frontelement + 1. ← from start node.
mark this new cell as visited.
~~If no snakes or ladders.~~
~~int~~

Define int vertex.

```
            If no snakes or ladders,
            {vertex = value of latest cell (i)}
              else
            {vertex = move[i]}
          Push vertex and dist into the queue as one
            node.
        }
      }
    }
    return
    return. q.distance.   q.dist.
}
```

we can create the board in the main function easily.

Time Complexity => O(n) since every cell added and removed only once from queue at max. Also, adding / removing takes O(1) time