```
class QueueEntry(object):

    def __init__(self, v = 0, dist = 0):

        self.v = v

        self.dist = dist


'''This function returns minimum number of
dice throws required to. Reach last cell
from 0'th cell in a snake and ladder game.
move[] is an array of size N where N is
no. of cells on board. If there is no
snake or ladder from cell i, then move[i]
is -1. Otherwise move[i] contains cell to
which snake or ladder at i takes to.'''

def getMinDiceThrows(move, N):



    # The graph has N vertices. Mark all

    # the vertices as not visited

    visited = [False] * N



    # Create a queue for BFS

    queue = []



    # Mark the node 0 as visited and enqueue it

    visited[0] = True



    # Distance of 0't vertex is also 0

    # Enqueue 0'th vertex

    queue.append(QueueEntry(0, 0))



    # Do a BFS starting from vertex at index 0

    qe = QueueEntry() # A queue entry (qe)

    while queue:

        qe = queue.pop(0)
```

```
v = qe.v # Vertex no. of queue entry

# If front vertex is the destination
# vertex, we are done
if v == N - 1:
    break

# Otherwise dequeue the front vertex
# and enqueue its adjacent vertices
# (or cell numbers reachable through
# a dice throw)
j = v + 1
while j <= v + 6 and j < N:

    # If this cell is already visited,
    # then ignore
    if visited[j] is False:

        # Otherwise calculate its
        # distance and mark it
        # as visited
        a = QueueEntry()
        a.dist = qe.dist + 1
        visited[j] = True

        # Check if there a snake or ladder
        # at 'j' then tail of snake or top
        # of ladder become the adjacent of 'i'
        a.v = move[j] if move[j] != -1 else j
```

```
                queue.append(a)

            j += 1

    # We reach here when 'qe' has last vertex
    # return the distance of vertex in 'qe
    return qe.dist

# driver code
N = 30
moves = [-1] * N

# Ladders
moves[2] = 21
moves[4] = 7
moves[10] = 25
moves[19] = 28

# Snakes
moves[26] = 0
moves[20] = 8
moves[16] = 3
moves[18] = 6


print("Min Dice throws required is {0}".
       format(getMinDiceThrows(moves, N)))
```