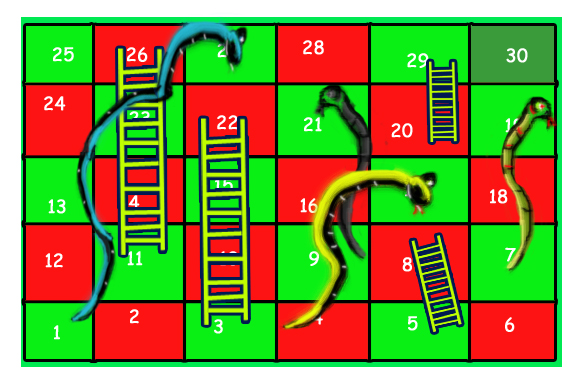
Given a snake and ladder board, find the minimum number of dice throws required to reach the destination or last cell from source or 1st cell. Basically, the player has total control over outcome of dice throw and wants to find out minimum number of throws required to reach last cell.  
If the player reaches a cell which is base of a ladder, the player has to climb up that ladder and if reaches a cell is mouth of the snake, has to go down to the tail of snake without a dice throw.



For example, consider the board shown, the minimum number of dice throws required to reach cell 30 from cell 1 is 3.   
Following are the steps:  
a) First throw two on dice to reach cell number 3 and then ladder to reach 22   
b) Then throw 6 to reach 28.   
c) Finally through 2 to reach 30.  
There can be other solutions as well like (2, 2, 6), (2, 4, 4), (2, 3, 5).. etc.

Solution:

// Java program to find minimum number of dice

// throws required to reach last cell from first

// cell of a given snake and ladder board

import java.util.LinkedList;

import java.util.Queue;

public class SnakesLadder

{

// An entry in queue used in BFS

static class qentry

{

int v;// Vertex number

int dist;// Distance of this vertex from source

}

// 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.

static int getMinDiceThrows(int move[], int n)

{

int visited[] = new int[n];

Queue<qentry> q = new LinkedList<>();

qentry qe = new qentry();

qe.v = 0;

qe.dist = 0;

// Mark the node 0 as visited and enqueue it.

visited[0] = 1;

q.add(qe);

// Do a BFS starting from vertex at index 0

while (!q.isEmpty())

{

qe = q.remove();

int v = qe.v;

// 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)

for (int j = v + 1; j <= (v + 6) && j < n; ++j)

{

// If this cell is already visited, then ignore

if (visited[j] == 0)

{

// Otherwise calculate its distance and

// mark it as visited

qentry a = new qentry();

a.dist = (qe.dist + 1);

visited[j] = 1;

// Check if there a snake or ladder at 'j'

// then tail of snake or top of ladder

// become the adjacent of 'i'

if (move[j] != -1)

a.v = move[j];

else

a.v = j;

q.add(a);

}

}

}

// We reach here when 'qe' has last vertex

// return the distance of vertex in 'qe'

return qe.dist;

}

public static void main(String[] args)

{

// Let us construct the board given in above diagram

int N = 30;

int moves[] = new int[N];

for (int i = 0; i < N; i++)

moves[i] = -1;

// 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;

System.out.println("Min Dice throws required is " +

getMinDiceThrows(moves, N));

}

}

**Output:**

Min Dice throws required is 3

Time complexity of the above solution is O(N) as every cell is added and removed only once from queue. And a typical enqueue or dequeue operation takes O(1) time.