Minion's bored game
==========

There you have it. Yet another pointless "bored" game created by the bored minions of Professor Boolean.

The game is a single player game, played on a board with n squares in a horizontal row. The minion places a token on the left-most square and rolls a special three-sided die.

If the die rolls a "Left", the minion moves the token to a square one space to the left of where it is currently. If there is no square to the left, the game is invalid, and you start again.

If the die rolls a "Stay", the token stays where it is.

If the die rolls a "Right", the minion moves the token to a square, one space to the right of where it is currently. If there is no square to the right, the game is invalid and you start again.

The aim is to roll the dice exactly t times, and be at the rightmost square on the last roll. If you land on the rightmost square before t rolls are done then the only valid dice roll is to roll a "Stay". If you roll anything else, the game is invalid (i.e., you cannot move left or right from the rightmost square).

To make it more interesting, the minions have leaderboards (one for each n,t pair) where each minion submits the game he just played: the sequence of dice rolls. If some minion has already submitted the exact same sequence, they cannot submit a new entry, so the entries in the leader-board correspond to unique games playable.

Since the minions refresh the leaderboards frequently on their mobile devices, as an infiltrating hacker, you are interested in knowing the maximum possible size a leaderboard can have.

Write a function answer(t, n), which given the number of dice rolls t, and the number of squares in the board n, returns the possible number of unique games modulo 123454321. i.e. if the total number is S, then return the remainder upon dividing S by 123454321, the remainder should be an integer between 0 and 123454320 (inclusive).

n and t will be positive integers, no more than 1000. n will be at least 2.

Test cases
==========

Inputs:
    (int) t = 1
    (int) n = 2

Output:
   (int) 1

Inputs:
   (int) t = 3
   (int) n = 2
Output:
   (int) 3

```
//Author: Neil VonHoltum
package com.google.challenges;
import java.math.BigInteger;

public class Answer {

    private static BigInteger[] board;
    private static int rolls;

    public static int answer(int t, int n) {

        if(t+1 < n){

            return 0;
        }

        board = new BigInteger[n];
        board[1] = board[0] = BigInteger.ONE;
        rolls = t-1;
        int size = 2;
        int last = n-1, secondtolast = n-2;

        while(size < n){

            board[size] = board[size - 1];
            arrange(size);
            size++;
        }

        while(rolls > 0){

            board[last] = board[secondtolast].add(board[last]);
            arrange(last);
        }
```

```
        return board[last].mod(BigInteger.valueOf(123454321)).intValue();
    }

    public static void arrange(int looptrip){

        BigInteger left = board[0], now, boardprevi;
        boardprevi = left;

        for(int i = 0, previ = i++; i < looptrip; previ = i++){

            now = board[i];
            board[previ] = boardprevi.add(now);
            boardprevi = board[i] = left.add(now);
            left = now;
        }

        rolls--;
    }
}
```