Zombit pandemic
==================

The nefarious Professor Boolean is up to his usual tricks. This time he is using social engineering to achieve his twisted goal of infecting all the rabbits and turning them into zombits! Having studied rabbits at length, he found that rabbits have a strange quirk: when placed in a group, each rabbit nudges exactly one rabbit other than itself. This other rabbit is chosen with uniform probability. We consider two rabbits to have socialized if either or both of them nudged the other. (Thus many rabbits could have nudged the same rabbit, and two rabbits may have socialized twice.) We consider two rabbits A and B to belong to the same rabbit warren if they have socialized, or if A has socialized with a rabbit belonging to the same warren as B.

For example, suppose there were 7 rabbits in Professor Boolean's nefarious lab. We denote each rabbit using a number. The nudges may be as follows:

1 nudges 2 2 nudges 1 3 nudges 7 4 nudges 5 5 nudges 1 6 nudges 5 7 nudges 3

This results in the rabbit warrens {1, 2, 4, 5, 6} and {3, 7}.

Professor Boolean realized that by infecting one rabbit, eventually it would infect the rest of the rabbits in the same warren! Unfortunately, due to budget constraints he can only infect one rabbit, thus infecting only the rabbits in one warren. He ponders, what is the expected maximum number of rabbits he could infect?

Write a function answer(n), which returns the expected maximum number of rabbits Professor Boolean can infect given n, the number of rabbits. n will be an integer between 2 and 50 inclusive. Give the answer as a string representing a fraction in lowest terms, in the form "numerator/denominator". Note that the numbers may be large.

For example, if there were 4 rabbits, he could infect a maximum of 2 (when they pair up) or 4 (when they're all socialized), but the expected value is 106 / 27. Therefore the answer would be "106/27".

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

public class Answer {
```

```
private static int publicn;
private static BigInteger[] factorials, ways;
private static BigInteger numerator;
private static int[] p;
private static BigInteger[][] bc;

public static String answer(int n) {

    /*
    Sum of all pseudo forest widths / total pseudo forests, or average width of all possible
    pseudo forests.
    */

    publicn = n;
    int nplusone = n+1;
    factorials = new BigInteger[nplusone];
    numerator = BigInteger.ZERO;
    bc = new BigInteger[nplusone][nplusone];
    ways = new BigInteger[nplusone];
    BigInteger prev = factorials[0] = BigInteger.ONE;

    //Fill factorials array.
    for(int ind = 1; ind < nplusone; ind++){

        prev = factorials[ind] = prev.multiply(BigInteger.valueOf(ind));
    }

    //Generate ways to form a network of size n when each node nudges another. A000435
    for(int i = 2; i < nplusone; i++){

        BigInteger sum = BigInteger.ZERO;
        BigInteger bigi = BigInteger.valueOf(i);

        for(int c = 1; c < i; c++){

            BigInteger firstpart = nchoosek(i, c);
            int isubc = i-c;
            BigInteger secondpart = BigInteger.valueOf(isubc).pow(isubc);
            BigInteger thirdpart = BigInteger.valueOf(c).pow(c);
            sum = sum.add(firstpart.multiply(secondpart).multiply(thirdpart));
        }
    ways[i] = sum.divide(bigi);
}

//Start experiment.
numerator = numerator.add(ways[n].multiply(BigInteger.valueOf(n)));
int looptrip = n/2, remaining = 2;

for(int i = n - 2; i > looptrip; i--){

    BigInteger rest = BigInteger.valueOf(remaining - 1).pow(remaining++);
    numerator = numerator.add(BigInteger.valueOf(i).multiply(nchoosek(n, i))
                        .multiply(ways[i]).multiply(rest));
}
//End experiment.

//Generate partitions.
int[] a = new int[nplusone];
int k = 1, y = n - 1;

while(k != 0){

    int x = a[--k] + 1;

    while(x << 1 <= y){

        a[k++] = x;
        y -= x;
    }

    int l = k + 1;

    while(x <= y){

        a[k] = x++;
        a[l] = y--;
        p = Arrays.copyOfRange(a, 0, k + 2);
        addtonumerator();
    }

    int xandy = a[k] = x + y;
    y = xandy - 1;
    p = Arrays.copyOfRange(a, 0, k+1);
    addtonumerator();
```

```
        }

        BigInteger denominator = BigInteger.valueOf(n-1).pow(n);
        BigInteger gcd = numerator.gcd(denominator);
        return numerator.divide(gcd).toString() + "/" + denominator.divide(gcd).toString();
    }

    public static void addtonumerator(){

        Arrays.sort(p);
        int maxinp = p[p.length - 1];

        if(p[0] > 1 && maxinp <= publicn/2){

            BigInteger formeachtreemult = BigInteger.ONE, waystosplitintop = BigInteger.ONE,
            maxtree = BigInteger.valueOf(maxinp), denom = BigInteger.ONE;
            int remainingrabbits = publicn;
            HashMap<Integer, Integer> treemultiplicities = new HashMap<Integer, Integer>();

            for(int i = 0; i < p.length; i++){

                int treetoexamine = p[i];
                formeachtreemult = formeachtreemult.multiply(ways[treetoexamine]);
                waystosplitintop = waystosplitintop.multiply(nchoosek(remainingrabbits,
                treetoexamine));

                remainingrabbits -= treetoexamine;
                int count = 1;

                if(treemultiplicities.containsKey(treetoexamine)){

                    count += treemultiplicities.get(treetoexamine);
                }

                treemultiplicities.put(treetoexamine, count);
            }

            Iterator<Integer> mvaluesitr = treemultiplicities.values().iterator();

            do{

                denom = denom.multiply(factorials[mvaluesitr.next()]);
            } while (mvaluesitr.hasNext());

            waystosplitintop = waystosplitintop.divide(denom);
            numerator =
            numerator.add(maxtree.multiply(waystosplitintop).multiply(formeachtreemult));
        }
    }

    public static BigInteger nchoosek(int n, int k){

        if(bc[n][k] == null){

            int nsubk = n - k;
            return bc[n][nsubk] = bc[n][k] = factorials[n].divide(factorials[k].multiply(factorials[nsubk]));

        }

        else{

            return bc[n][k];
        }
    }
}
```