**CIVIL - Civil Engineering**

*no tags*

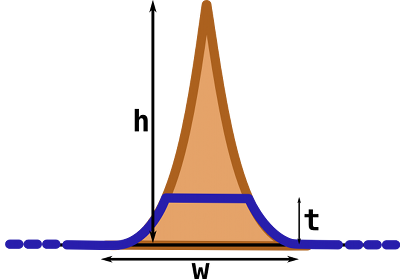
**Problem description**

Civil engineers are people who build impressive structures from concrete and steel. Throughout the world, sky scrapers get ever higher, bridges get ever wider, and tunnels get ever longer. Most recently, the Gotthard Base Tunnel just pushed the limits a bit further...



In this task, you take the role of a civil engineer who is to build a tunnel through a mountain. Unfortunately, you have a very limited budget, and must construct the cheapest possible tunnel.

Because engineers have a habit of simplifying things, we will model the mountain and the tunnel using basic geometric shapes. First of all, consider that the earth is flat, and two-dimensional. On this flat surface stands a mountain of height h and width w. Each side of the mountain is parabolic (i.e. satisfies y = ax² + bx + c for some a, b, c). You also know that the base of the mountain is smooth, which means that its steepness at the base is zero.



The tunnel is modeled as a horizontal line through the mountain. The best possible tunnel is the one which minimizes construction cost. This cost is proportional to the length of the road which leads to the tunnel, plus the length of the tunnel itself. Consider that each meter of the tunnel is a factor f times as expensive as a meter of road.

### Input

The input file consists of several test cases. Each case is given on a line by itself and consists of the three numbers h, w and f, separated by a space. All these are strictly positive floating point numbers. The input file ends with a test case where all numbers are zero (which must not be processed).

### Output

Print for each test case a single number, the optimal height t of the tunnel. Always print three digits after the decimal point. You may assume that it is always cheaper to build some tunnel than to drive over the top of the mountain.

### Example

**Input:**

1 2 1.5

9033.66 29752.4 1.56382

0 0 0 **Output:**  
0.313  
8852.956

**Problem abstraction**

**Solution**

So, we need to find the

**Source code for SPOJ**

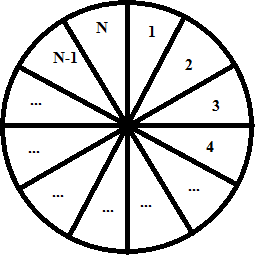
**CRCLE\_UI - Colorful Circle (EASY)**

*no tags*

**Problem description**

I take this problem from my midterm exam today, because for me and some of my friends it's interesting, so I decided to translated this problem into english and upload this problem to SPOJ. See the original problem in indonesian language [here](http://2.bp.blogspot.com/-hDQcB4cjLZQ/UKPlwjMSDeI/AAAAAAAAAPA/ZF-EdjfCKyA/s1600/2012-11-14+23.41.39.jpg).  
------------------------

Given **N** sectors where 1<**N**<101000, from a circle that sown in the picture below:



We will color each sector with **K** different colors, where 2<**K**<101000 such that each sector colored with one color and each adjacent sector must have different color. Your task is to count how many ways to color all that sectors.

**Input**

First line, there is a number **T**(0<**T**<1000) denoting number of test cases, then T lines follow.  
each line containing two integers: **N** and **K** separated by a space.

**Output**

For each test case, output number of ways to color the circle, since the number can be too large, take modulo 109+7.

**Example**

**Input:**

2

2 3

3 3

**Output:**

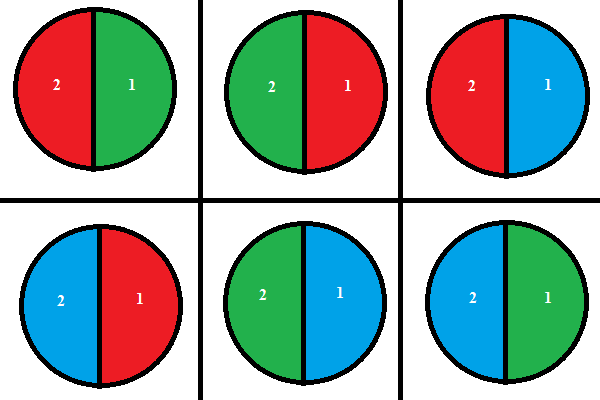
6

6

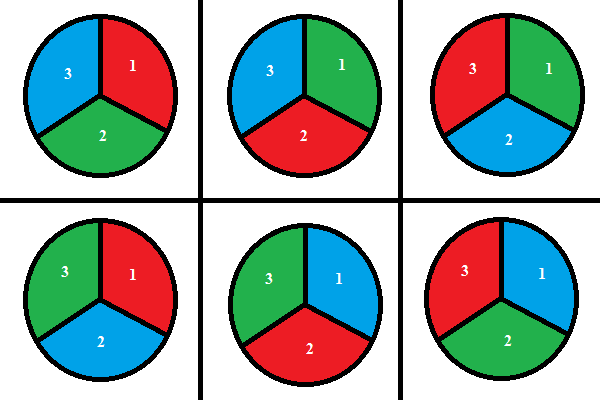
**Problem abstraction**

**Explanation:**

For the first case, we have two sectors and three colors, here is all possibilities:



For second test case, we have three sector and three colors, here is all possibilities:

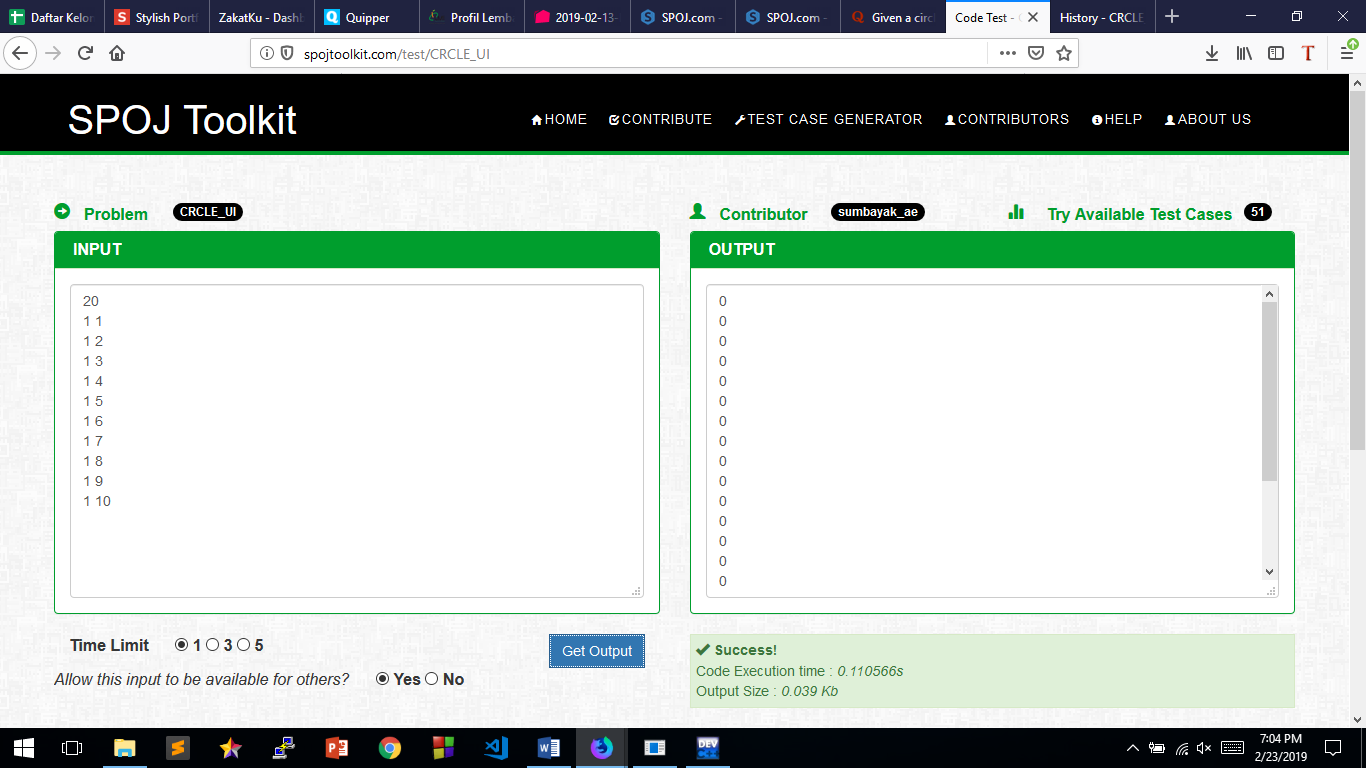


Furthermore, the pattern is changed when N > 3, because it have N-2 sector that should be count as different thing that N-1 itself, because when k < 1/2N 🡨 there should be at least 2 sector that have same colour. So there is some addition possibility result in N-2.

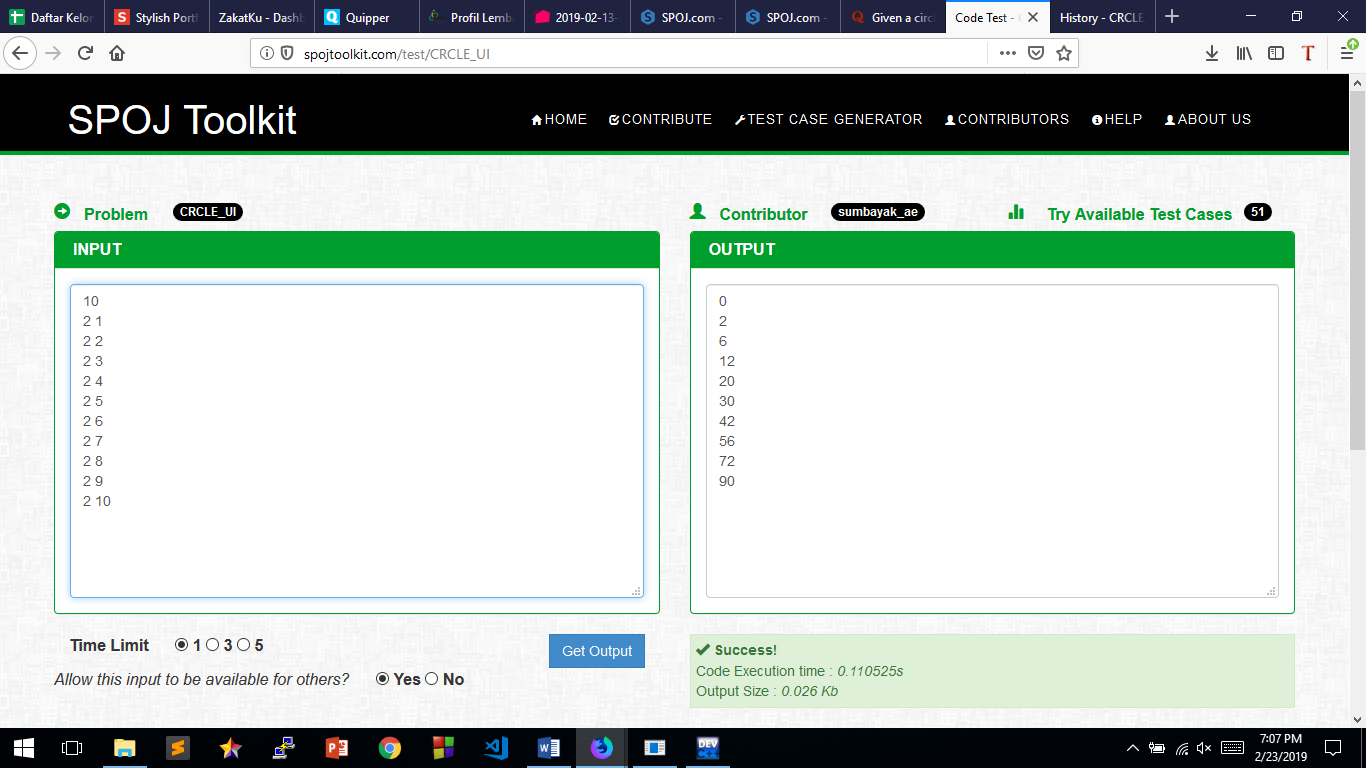
So, we can assume that this case can not be solved by permutation or combination ways. Because there are at least 2 terms - one which looks at the circles with N - 1 segments and one that looks at the circle with N - 2 segments. From this I could also determine that there are 2 or more initial conditions that would give you the solution for N = 1 and N = 2, as they don’t have a both the N - 1 and N - 2 segments circles.

**Solution**

Now here is the thing - for my calculations to make sense, I would have to make the *N*1=0, which would make the case where N = 1 always 0. That mean N=1 is a special case



This cannot be fixed within the recursion, because N = 1 truly is a special case.



Now, *N*2=(*k*−1). How did I find this out? Well, I knew that for N = 2, the solution is always *k*(*k*−1) 🡨 0, 2, 6, 12, 20, 30, 42, 56, 72, 90 have such pattern.

Because I still didn’t have a serious combinatorics class, I could mostly rely on my senses, and if there was a recursive solution, it would at least have to hold for N = 2, since it is not, by any means, a special case.

But now, how could I find out what the formula for *Nn* was? Well, I assumed that the way the two numbers of circles were looked at was that an odd offset would always multiply with (*k*−2).

This is because the way the number of combinations change is drastic when changing from an odd N to and even N and vice versa, so every change like that would yield less combinations than a change from even to even or odd to odd would. So, here is my answer for Nn:

***Nn*=(*k*−1)*Nn*−2+(*k*−2)*Nn*−1**

**Source code for SPOJ**

#include <stdio.h>

#include <stdlib.h>

unsigned long long rec(unsigned long long N, unsigned long long k) {

unsigned long long hasil;

if (N>3) {

return ((k-1)\*rec(N-2, k) + (k-2)\*rec(N-1, k)) % 1000000007;

}

else {

return k\*(k-1)\*(k-2);

}

}

int main() {

int T;

unsigned long long N, k, hasil;

scanf("%d", &T);

int i;

for (i=0;i<T;i++) {

scanf("%llu %llu", &N, &k);

if(N%2 == 0) {

if (N<2) {

hasil = 0;

printf ("%llu\n", hasil);

}

else {

printf ("%llu\n", rec(N, k));

}

}

else {

if (N<=2) {

hasil = 0;

printf ("%llu\n", hasil);

}

else {

printf ("%llu\n", rec(N, k));

}

}

}

return 0;

}