Swap (Hard - Level 1000)

Let's play with sequence of non negative integer. Given two sequence of **n** non negative integers (a_1, a_2 ... a_n) and (b_1, b_2 ... b_n). Both sequence has maximum element less than **k**, max(a_1, a_2 ... a_n) < **k** and max(b_1, b_2 ... b_n) < **k**. The game rule is you can edit both sequence with this operation: swap a_i and b_i with 1 ≤ **i** ≤ **n**, and the goal is to make sequence **a** and **b** become increasing sequence: $a_i \le a_j$ if and only if **i** ≤ **j** and $b_i \le b_j$ if and only if **i** ≤ **j**. But not all initial sequence **a** and **b** can be solved.

For example (2, 0) and (0, 1) is a pair of sequence that can't be solved:

- If you don't swap any element, you have (2, 0) and (0, 1), but sequence (2, 0) is not increasing.
- If you swap first element only, then the pair become like this (0, 0) and (2, 1), sequence (2, 1) is not increasing.
- If you swap second element only, then the pair become like this (2, 1) and (0, 0), again (2, 1) is not increasing.
- If you swap both element, then the pair become like this (0, 1) and (2, 0), again (2, 0) is not increasing

So it's impossible to solve if initial sequence is (2, 0) and (0, 1), because all possible move can't make both sequence become increasing.

Now given **n** and **k**, your task is to compute number of different pair of initial sequence (**a**, **b**) that can be solved with game described above.

Input

First line there is an integer **T** denoting number of test case, then **T** test cases follow.

For each case, there are two integers **n** and **k** written in one line, separated by a space.

Output

For each case, output number of different pair of initial sequence (**a**, **b**), since the answer can be large, output the answer modulo 10^9+7.

Constraints

0 < **T** ≤ 10^4

0 < min(**n**, **k**) ≤ 1000

0 < max(**n**, **k**) < 10^{1000}

Example

Input:
6

2 1
1 2
1 3
2 2
3 2
2 3

Output:
1
4
9
11
26
46

Explanation

Here is list of all possible pair of initial sequence (**a**, **b**) on each case:

Case 1: {[(0, 0), (0, 0)]}

Case 2: {[(0), (0)], [(0), (1)], [(1), (0)], [(1), (1)]}

Case 3: {[(0), (0)], [(0), (1)], [(0), (2)], [(1), (0)], [(1), (1)], [(1), (2)], [(2), (0)], [(2), (1)], [(2), (2)]}

Case 4: {[(0, 0), (0, 0)], [(0, 0), (0, 1)], [(0, 0), (1, 1)], [(0, 1), (0, 0)], [(0, 1), (0, 1)], [(0, 1), (1, 0)], [(0, 1), (1, 1)], [(1, 0), (0, 1)], [(1, 1), (0, 0)], [(1, 1), (0, 1)], [(1, 1), (1, 1)]}

Case 5: {[(0, 0, 0), (0, 0, 0)], [(0, 0, 0), (0, 0, 1)], [(0, 0, 0), (0, 1, 1)], [(0, 0, 0), (1, 1, 1)], [(0, 0, 1), (0, 0, 0)], [(0, 0, 1), (0, 0, 1)], [(0, 0, 1), (0, 1, 0)], [(0, 0, 1), (0, 1, 1)], [(0, 0, 1), (1, 1, 0)], [(0, 0, 1), (1, 1, 1)], [(0, 1, 0), (0, 0, 1)], [(0, 1, 0), (1, 0, 1)], [(0, 1, 1), (0, 0, 0)], [(0, 1, 1), (0, 0, 1)], [(0, 1, 1), (0, 1, 1)], [(0, 1, 1), (1, 0, 0)], [(0, 1, 1), (1, 0, 1)], [(0, 1, 1), (1, 1, 1)], [(1, 0, 0), (0, 1, 1)], [(1, 0, 1), (0, 1, 0)], [(1, 0, 1), (0, 1, 1)], [(1, 1, 0), (0, 0, 1)], [(1, 1, 1), (0, 0, 0)], [(1, 1, 1), (0, 0, 1)], [(1, 1, 1), (0, 1, 1)], [(1, 1, 1), (1, 1, 1)]}

Case 6: {[(0, 0), (0, 0)], [(0, 0), (0, 1)], [(0, 0), (0, 2)], [(0, 0), (1, 1)], [(0, 0), (1, 2)], [(0, 0), (2, 2)], [(0, 1), (0, 0)], [(0, 1), (0, 1)], [(0, 1), (0, 2)], [(0, 1), (1, 0)], [(0, 1), (1, 1)], [(0, 1), (1, 2)], [(0, 1), (2, 2)], [(0, 2), (0, 0)], [(0, 2), (0, 1)], [(0, 2), (0, 2)], [(0, 2), (1, 0)], [(0, 2), (1, 1)], [(0, 2), (1, 2)], [(0, 2), (2, 0)], [(0, 2), (2, 1)], [(0, 2), (2, 2)], [(1, 0), (0, 1)], [(1, 0), (0, 2)], [(1, 1), (0, 0)], [(1, 1), (0, 1)], [(1, 1), (0, 2)], [(1, 1), (1, 1)], [(1, 1), (1, 2)], [(1, 1), (2, 2)], [(1, 2), (0, 0)], [(1, 2), (0, 1)], [(1, 2), (0, 2)], [(1, 2), (1, 1)], [(1, 2), (1, 2)], [(1, 2), (2, 1)], [(1, 2), (2, 2)], [(2, 0), (0, 2)], [(2, 1), (0, 2)], [(2, 1), (1, 2)], [(2, 2), (0, 0)], [(2, 2), (0, 1)], [(2, 2), (0, 2)], [(2, 2), (1, 1)], [(2, 2), (1, 2)], [(2, 2), (2, 2)]}

Other Info

Test case (**n** and **k**) is generated randomly using this rule:

- Probability that **n**>**k** or **n**<=**k** is ~50% each.
- Maximum **n** and **k** is random log-uniform.
- Minimum **n** and **k** is random uniform.

Click here if you want to know my program speed and other detail.

Explanation about my Algorithm complexity:

> My 3.8KB of C code with O(min(**n**, **k**)^3) complexity got AC in 32.17s.
>
> Other submission like O(min(**n**, **k**)^4) in fast language, and O(min(**n**, **k**)^3) in slow language is all TLE. That's why this problem has "Hard" label.

Sorry for slow language user, I think it's impossible to solve this problem unless O(min(**n**, **k**)^2) exists. I recommend to try Medium version first, or learn fast language :-P

About complexity, I've proved using math that no algorithm with complexity better than O(min(**n**, **k**)^2), this is the lower bound. My best algorithm for now is O(min(**n**, **k**)^3), this is the upper bound. So the optimal algorithm lies between that lower and upper bound. I still don't have proof that my algo is optimal, so there is possibility that there is an algorithm that better than O(min(**n**, **k**)^3).

Btw, if I found around O(min(**n**, **k**)^2) by myself, I'll set "Extreme" version (level 10000+) of this problem. But if there is someone who solve this problem in around O(min(**n**, **k**)^2), of course he/she has honor to set "Extreme" version of this problem.

Time limit ~3× my program top speed.

See also: Another problem added by Tjandra Satria Gunawan