PROBLEM 5

Consider the following card game. You are given k packs of n cards numbered $1, 2, \cdots, n$. Number i is printed on the right side of the card numbered i. Thus, for each i, there are exactly k cards with number i printed on their right sides. You schuffle the nk cards sufficiently and divide them into n packs, each of which consists of k cards piled up in a certain order. Then, you arrange the n packs in a line, and call the i-th pack from the left on the line the i-th *Mountain*. See Fig. 1.

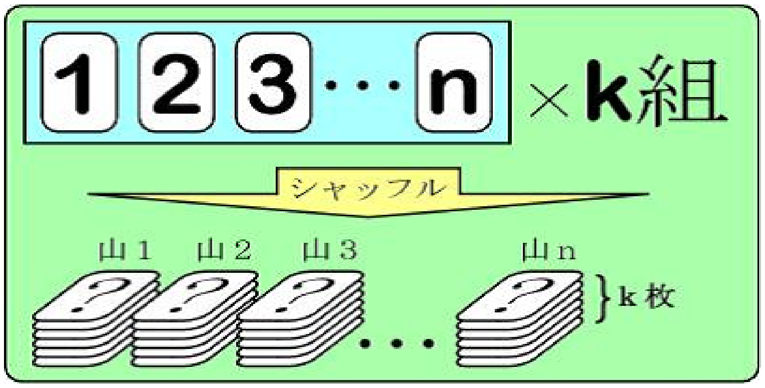

Figure 1: The initial configuration of a game.

You start a game at Mountain 1. Draw the topmost card of the Mountain. Whenever a card is drawn from a Mountain during a game, it will not be put back to the Mountain again. If the number printed on the drawn card is i, then draw the topmost card of Mountain i next time. If, as a result, j is printed on the drawn card, then draw the topmost card of Mountain j next time, and so on. Repeat such process of drawing a card from the designated Mountain, until no card remains in the mountain from which you have to draw its topmost card at that time. Then, you win the game if every Mountain is empty (i.e., no card remains in any Mountain), otherwise you lose. This ends the *first stage* of a game.

Whenever you lose a game, you may continue the game as the *second stage*, as follows. Only those Mountains with at least one remaining cards will be used in the second stage. The numbering for such Mountains remain unchanged. You start the second stage by drawing the topmost card from the leftmost Mountain among the remaining Mountains. Continue the game just in the same way as in the first stage. Like the first stage, if no card remains on the Mountain you have to draw a card next, then the second stage is over. You win in case every Mountain is empty, and you lose otherwise.

For each game, you will be given an integer m. You have to go to the second stage if m is 1 and you have lost the first stage, otherwise the game is over just after the first stage.

Initial configurations for games differ depending on shuffling the cards. There are three possibilities for each game: you win in the first stage, you lose in the first stage and win in the second stage, or you lose both in the first and second stages, depending on the initial configuration. We assume that each possible initial configuration appears with the same probability. Under the assumption we like to find the probability p to win a game.

Your task is to write a program to compute and output the probability p, satisfying the following 2 conditions.

1. p should be written in decimal form with exactly r digits under the decimal point. If $p \times 10^K$ is an integer for a sufficiently large positive integer K, then the digits under the

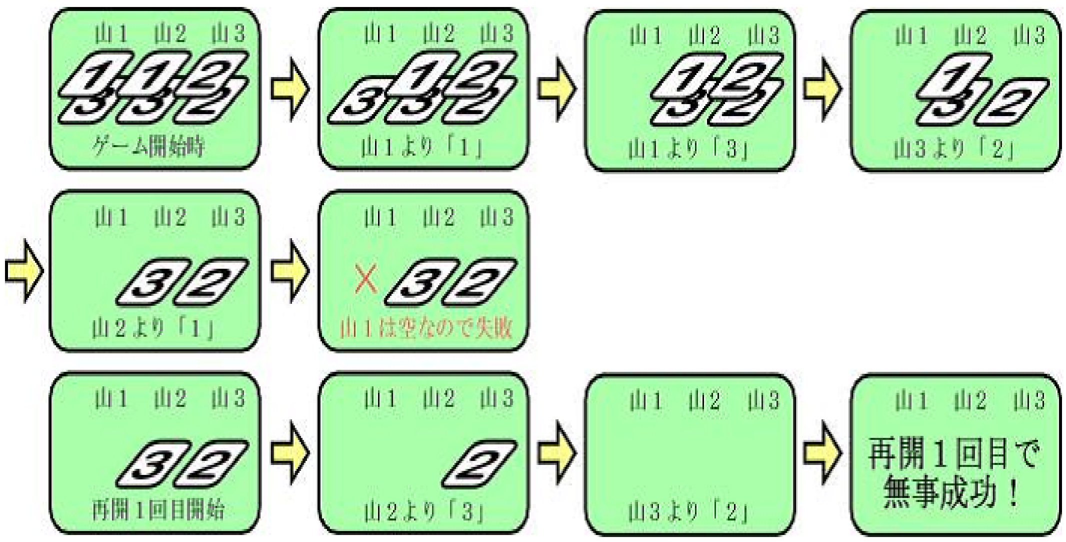

Figure 2: A sequence of moves in a game.

decimal point contain $000\cdots$ at the tail. In this case, the output should contain $00\cdots00$ at the tail so that the number of digits after the decimal point is exactly r. For example, $p = \frac{3}{8} = 0.375$ should be written as 0.37500 if $r = 5$, and 0.37 if $r = 2$. Similarly, if $p = 1.0$ and $r = 3$, then p should be written as 1.000.

2. For example, $0.150000\cdots$ can also be represented as $0.1499999\cdots$. In this case, $0.15000\cdots$ should be used.

INPUT

The first line of any input file contains integers n, k, m and r in this order, separated by a space character, where $1 \leq n \leq 10000$, $1 \leq k \leq 100$, m is either 0 or 1, and $1 \leq r \leq 10000$.

OUTPUT

Each output file should contain a single line containing the probability p followed by the Return code.

EXAMPLE

Example input 1	Example input 2	Example input 3
2 1 0 5	3 1 1 3	2 2 1 3

Example output 1	Example output 2	Example output 3
0.50000	0.833	1.000