Problem A. Area of Effect

Source file name:	areaofeffect.c, areaofeffect.cpp, areaofeffect.java, areaofeffect.py
Input:	`Standard`
Output:	`Standard`

Liam (angryneeson52) is playing his favorite tower defense game! This game involves destroying minions of his opponent while defending his own villages.
Liam's favorite attack is an Area of Effect attack. The Area of Effect attack is a perfect circle. Liam simply picks a center and a radius for the attack and every minion in or on that circle gets destroyed! Minions are small enough to be considered points.
The game isn't as simple as just destroying all minions. Liam must also avoid hitting his villages with his attacks. The attack may touch the walls of a village but must not enter the village. Villages are also perfect circles.
His attack also has a limit on its maximum radius. The attack can be reduced in radius but cannot go above the maximum.
Determine the maximum number of minions Liam can destroy in a single attack without damaging any of his own villages.

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. Each input begins with a line with 3 space-separated integers, n m r, where:

- n ($1 \leq n \leq 10$) is the number of Liam's villages.
- m ($1 \leq m \leq 2000$) is the number of opposing minions.
- r ($1 \leq r \leq 20000$) is the maximum radius of Liam's Area of Effect attack.

The next n lines will each contain 3 space-separated integers v_x v_y v_r which represent the location ($-20000 \leq v_x, v_y \leq 20000$) and radius ($1 \leq v_r \leq 20000$) of one of Liam's villages. No two villages will intersect or overlap.
The next m lines will each contain 2 space-separated integers m_x m_y which represent the location ($-20000 \leq m_x, m_y \leq 20000$) of one of the enemy minions. No two minions will occupy the same point, and no enemy minion will be inside any of Liam's villages.

Output

Output a single integer representing the maximum number of enemy minions that Liam can destroy with a single attack.

Example

Input	Output
1 3 3 0 0 1 3 3 -3 3 3 -3	1
1 5 3 0 0 1 3 3 -3 3 3 -3 3 0 0 3	3
4 10 100 0 0 3 10 0 3 10 10 3 0 10 3 0 4 0 5 0 6 5 3 5 -3 5 5 6 7 3 6 10 4 8 4	5

Problem B. Canyon Mapping

Source file name:	canyon.c, canyon.cpp, canyon.java, canyon.py
Input:	`Standard`
Output:	`Standard`

Canyons are deep ravines between escarpments or cliffs. They exist on more than just Earth. For example, Valles Marineris on Mars is a vast canyon system running along the Martian equator and is roughly the size of the United States.

Working for a prestigious mapping company, you've been tasked with making maps for such canyons. A canyon can be represented in 2D by a simple polygon outline. The maps you will be constructing will be perfectly square and axis aligned. This is due to the mapping company's desire that tops of their maps are always North. In order to increase the detail, sometimes multiple maps are used to cover just one canyon. Such a collection is called a mapping system. The mapping system of the canyon must cover the entire area of the canyon. The maps in the mapping system must also be the same scale with respect to the canyon and the same size with respect to their printing. This allows the maps to be easily compared when viewed together.

Your mapping system will have exactly k maps. You need to find a mapping system that completely covers the canyon, but each map covers as little area as possible, since a map with less area can be shown at a higher detail. All of the maps must be perfectly square, and must cover the same amount of area on the canyon. The maps can overlap. Since area is the square of side length, just output the side length for simplicity. If things go well enough, your mapping system may even be used to map Valles Marineris in the near future.

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. Each input begins with a line with two space-separated integers n ($3 \leq n \leq 2000$) and k ($1 \leq k \leq 3$), where n is the number of vertices in the polygon, and k is the number of square maps in the mapping system. The next n lines each contain two space-separated integers x y ($-20000 \leq x, y \leq 20000$). These are the coordinates of the polygon, in order. No two edges of the polygon will intersect. All points will be distinct. No three consecutive points will be collinear. The polygon will be a simple polygon which does not touch or cross itself. It will not be degenerate, and will have positive area.

Output

Output a real number rounded to exactly two decimal places, representing the minimum side length with respect to the canyon for each square map in your mapping system, where the maps are identically sized, as small as possible, and the system still covers the entire canyon.

Example

Input	Output
4 1 1 1 5 1 5 5 4 2	4.00
6 3 -8 -8 0 -1 8 -8 1 0 0 10 -1 0	9.00
16 2 0 0 3 0 3 3 6 3 8 0 10 4 10 10 8 10 8 6 6 10 6 11 5 9 4 7 3 11 2 1 0 4	9.00

Problem C. Magic Checkerboard

Source file name:	checkerboard.c, checkerboard.cpp, checkerboard.java, checkerboard.py
Input:	`Standard`
Output:	`Standard`

Consider an $n \times m$ checkerboard. On each cell of the checkerboard, place a positive integer. The values in each column of the checkerboard must be in strictly increasing order from top to bottom, and the values in each row of the checkerboard must be in strictly increasing order from left to right.

1	2	3	4
3	4	5	6
5	6	7	8
7	8	9	10

A Magic Checkerboard has an additional constraint. The cells that share only a corner must have numbers of different parity (Even vs Odd). Note that the following checkboard is invalid, because 2 and 4 share only a corner and have the same parity:

1	2
4	6

The first 4×4 example is a valid Magic Checkboard. Given a partially filled magic checkboard, can you fill the remaining locations on the checkboard, so that the sum of all values is as small as possible?

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. Each input starts with a line with two space-separated integers n and m ($1 \leq n, m \leq 2000$), representing the number of rows (n) and the number of columns (m) of the checkerboard. Each of the next n lines will contain m space-separated integers c ($0 \leq c \leq 2000$), representing the contents of the checkerboard. Zero is used for cells without numbers that you must fill in. You may use any positive integers to fill in the cells without numbers, so long as you form a valid Magic Checkerboard. You are not limited to numbers ≤ 2000, and the numbers are not required to be unique.

Output

Output a single integer representing the minimum sum possible by replacing the 0 cells with positive integers to form a valid Magic Checkerboard. Output -1 if it is not possible to replace the 0 cells to meet the constraints of a Magic Checkerboard.

Example

Input	Output
4 4 1 2 3 0 0 0 5 6 0 0 7 8 7 0 0 10	88
4 4 1 2 3 0 0 0 5 6 0 4 7 8 7 0 0 10	-1
4 4 1 2 3 0 0 0 5 6 0 4 7 8 7 0 0 10	18

Problem D. Extensive Or

Source file name:	extensiveor.c, extensiveor.cpp, extensiveor.java, extensiveor.py
Input:	`Standard`
Output:	`Standard`

Consider a very large number R in a compressed format. It is compressed as a binary string s, and an integer k. Start with the empty string, and append s to it k times to get the binary representation of R. The string s is guaranteed to have a leading 1. Now, with R, solve the following problem: How many sets of n distinct integers are there such that each integer is between 0 and R -1, inclusive, and the XOR of all those integers is equal to zero? Since this number can get very large, return it modulo $10^9 + 7$.
As a reminder, XOR is Exclusive Or. The XOR of two numbers is done bitwise. Using $\oplus$ for XOR:

$$0 \oplus 0 = 0$$

$$0 \oplus 1 = 1$$

$$1 \oplus 0 = 1$$

$$1 \oplus 1 = 0$$

XOR is associative, so $a \oplus (b \oplus c) = (a \oplus b) \oplus c$.

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. Each input consists of exactly two lines. The first line has two space-separated integers n ($3 \leq n \leq 7$) and k ($1 \leq k \leq 100000$), where n is the number of distinct integers in the sets, and k is the number of times to repeat the string s in order to build R. The second line contains only the string s, which will consist of at least 1 and at most 50 characters, each of which is either 0 or 1. s is guaranteed to begin with a 1.

Output

Output a single line with a single integer, indicating the number of sets of n distinct integers that can be formed, where each integer is between 0 and R - 1 inclusive, and the XOR of the n integers in each set is 0. Output this number modulo $10^9 + 7$.

Example

Input	Output
3 1 100	1
3 1 100	1978
5 100 1	598192244

Problem E. Primal Partitions

Source file name:	primalpartitions.c, primalpartitions.cpp, primalpartitions.java, primalpartitions.py
Input:	`Standard`
Output:	`Standard`

The Math department has challenged the Computer Science department at your University to solve a difficult puzzle in Discrete Mathematics. With the pride of your Computer Science department at stake, you must solve this puzzle!

In this puzzle, you are given a sequence of n positive integers. The sequence must be partitioned into k consecutive contiguous regions, with at least 1 integer in each region. After finding a partition, it is scored in a strange way. In each region, you must find the largest prime number that divides every number in that region. The Math department reminds you a prime number is an integer greater than 1 where its only divisors are 1 and itself. If you cannot find such a prime, your score for that region is 0. Your total score for the partition is the minimum over all regions.

Your task is to find the maximum possible score. The Math department will win if their score is better, so be sure to find the maximum each time!

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. Each input begins with a line with two space-separated integers n ($1 \leq n \leq 20000$) and k ($1 \leq k \leq \min(100, n)$), where n is the number of positive integers in the original sequence, and k is the number of regions in the partition. The next line contains n space-separated integers v ($1 \leq v \leq 1000000$). This is the sequence to be partitioned, in order.

Output

Output a single integer representing the maximum score possible partitioning the sequence of n positive integers into k regions.

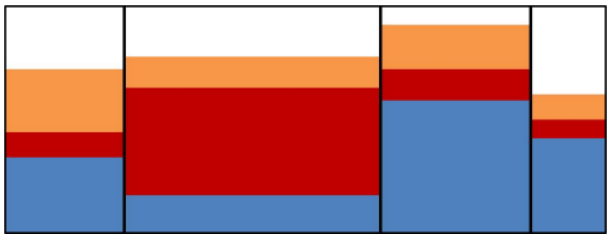
Example

Input	Output
5 3 10 5 4 8 3	2
5 3 10 11 12 13 14	0

Problem F. Sand Art

Source file name:	sandart.c, sandart.cpp, sandart.java, sandart.py
Input:	`Standard`
Output:	`Standard`

At art shows, it is very common to have booths where children can create their very own sand art. This art is typically made by taking a jar or bottle and filling it with layers of different colors of sand. Instead of a bottle, this year a new container is being used for decorating! The container is a glass box!
The box has a 2D rectangular face and a thickness of exactly 1 unit. Inside the glass box, $n-1$ vertical dividers are placed to separate it into n sections. In the example below, the box is divided into 4 sections using 3 dividers: Sometimes the children want certain amounts of each color to be in each section of their



work of art. They specify a minimum and maximum for each combination of section and sand color. Your task is to help them find how balanced they can make the artwork. This is done by minimizing the difference between the sand heights in the section with the highest overall sand level and the section with the lowest overall sand level.

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. Each input begins with a single line with 4 space-separated integers, n m w h, where:

- n ($2 \leq n \leq 200$) is the number of sections
- m ($1 \leq m \leq 200$) is the number of colors of sand
- w, h ($1 \leq w, h \leq 5000$) are the width and height of the box (it always has a depth of 1)

The next line will contain m space-separated real numbers (with at most 3 decimal places) v ($0 < v \leq w \cdot h$), which represent the volumes of each color of sand. It is not necessary to use all of the sand, but the minimums for each section must be satisfied.
The next line will have n - 1 space-separated real numbers with at most 3 decimal places) x ($0 < x < w$) which represent the distance from the left wall of each divider. The xs are guaranteed to be sorted in increasing order.
The next n lines will each have m space-separated real numbers (with at most 3 decimal places) min ($0 \leq min \leq w \cdot h$). The j^{th} element of the i^{th} row is the minimum amount of sand color j to be put in section i.
The next n lines will each have m space-separated real numbers (with at most 3 decimal places) max ($0 \leq max \leq w \cdot h$). The j^{th} element of the i^{th} row is the maximum amount of sand color j to be put in section i, and $min_{ij} \leq max_{ij}$.

Output

Output a real number rounded to exactly 3 decimal places representing the minimum difference possible between the maximum and minimum heights of sand in the sections. A distribution of sand will always be possible that satisfies the constraints in the input.

Example

Input	Output
2 2 5 5 2.0 2.0 4.0 1.0 0.0 0.0 1.0 1.0 0.0 0.0 2.0	0.750
2 2 5 5 2.0 2.0 4.0 1.0 0.0 0.0 1.0 1.5 0.0 0.0 2.0	0.625
2 5 11 10 3.0 4.0 4.0 9.0 2.0 4.0 2.0 2.0 1.0 0.5 0.25 0.0 2.0 0.0 4.0 1.0 2.0 2.0 3.0 4.0 0.75 0.0 2.1 0.0 5.1 1.1	0.266

Problem G. String Stretching

Source file name:	stretching.c, stretching.cpp, stretching.java, stretching.py
Input:	`Standard`
Output:	`Standard`

Start with a string p. Now, create a new string s, like this: Start with the empty string, and insert p. Then, choose some position in the string (including, possibly, the very beginning or the very end), and insert p again. And again. And again.
For example, suppose p is "**hello**". Starting with the empty string, a string s might be generated like this (each new insertion of p is in **bold**):

1.
2. **hello**
3. h**hello**ello
4. hhelloel**hello**lo
5. hhe**hello**lloelhellolo

So, after 5 steps, the string s is **hhehellolloelhellolo**.
Given the final string s, find the shortest string p which could have generated s. If there's more than one with the shortest length, find the one that comes first alphabetically.

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. Each input consists of a single line with a single string s. The string will consist of only lower case letters, and will be at least 1, and at most 200, characters long.

Output

Output a single line with the string p, which is the shortest possible string that could generate s.

Example

Input	Output
`hhehellolloelhellolo`	`hello`

Problem H. Vending Machine

Source file name:	vending.c, vending.cpp, vending.java, vending.py
Input:	`Standard`
Output:	`Standard`

Sleazy Bob has happened upon a vending machine. After watching enough people buy tasty snacks, Bob has realized that the vending machine is broken!
Here's what Sleazy Bob observes:

1. A person tries to buy a snack.
2. The vending machine then checks to see if there are any left of that snack.
3. If there are any left, the machine charges the person for that snack.
4. If the machine successfully charges the person, it then gives the person a *different* snack! Possibly no snack at all, if the machine is out of the different snack!

Sleazy Bob notices that, although the machine is broken, it is at least consistent. Whenever a customer buys a snack from position i, the machine vends from position $f(i)$, where f is some predictable function. Now, Bob wants to make a profit from the machine. He wants to buy some snacks from the machine, and then turn around and sell the snacks he gets for the market price of the snack. This may be different from the vending price. If a cheap snack is at i, and an expensive snack is at $f(i)$, he could rake in the cash! Assuming Bob can always find buyers for his snacks, what is the maximum net gain that Bob can obtain by buying some number, possibly zero, of snacks and turning around and selling them later? You may assume that Bob has enough money from other shady activities to cover the cost of buying any number of snacks from the vending machine.

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. Each input begins with a line with a single integer n ($1 \leq n \leq 100000$), which is the number of snack positions in the machine. Each of the next n lines contains 4 space-separated integers, f p m s, which describe a snack position in the machine, in order from 1 to n, where:

- f ($1 \leq f \leq n$) is the value of $f(i)$. That is, it is the position from which the machine will vend when Bob buys from this position.
- p ($1 \leq p \leq 1000000$) is the price Bob must pay to buy from this position.
- m ($1 \leq m \leq 1000000$) is the market price of the snack at this position
- s ($1 \leq s \leq 1000000$) is the number of snacks at this position.

Output

Output a single line with a single integer, indicating the maximum profit Sleazy Bob can get from his nefarious abuse of the broken vending machine.

Example

Input	Output
hhehellolloelhellolo	hello

Problem I. Rainbow Zamboni

Source file name:	zamboni.c, zamboni.cpp, zamboni.java, zamboni.py
Input:	`Standard`
Output:	`Standard`

Pacman is cleaning his ice rink! The rink, like everything in PacMan's universe, wraps! In other words, if you go off the edge of the rectangular rink to the right, you'll end up in the same row on the left side of the rink. Go off of the left side of the rink, and you'll end up on the right. Going up off the top of the rink causes you to end up in the same column but on the bottom. Go off of the bottom, and you'll end up all the top.
What makes this zamboni special is that when it cleans the ice, it leaves a color on the ice! At the start, the ice is colorless (white). Whenever the zamboni is on a cell of ice, it overwrites the color on that cell. Each color is represented by a capital letter of the alphabet. The colors are in alphabetical order. The zamboni switches to the next letter in the alphabet at each step, and wraps at the end of the alphabet. If the current color is P, the next color is Q. If the current color is Z, the next color is A.
PacMan uses a very specific algorithm to clean the ice:

$stepSize \leftarrow 1$
loop $numSteps$ times
    Move $stepSize$ steps in the current direction
    Rotate the direction of the zamboni 90 degrees clockwise
    Switch to the next color
    $stepSize \leftarrow stepSize + 1$
end loop

The zamboni is originally facing up. Knowing where the zamboni starts, the size of the rink, and the number of steps taken, determine what the ice looks like after the zamboni has completed its run.

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. Each input consists of a single line with 5 space-separated integers, r c i j n, where:

- r ($1 \leq r \leq 2000$) is the number of rows of the ice rink.
- c ($1 \leq c \leq 2000$) is the number of columns of the ice rink.
- i ($1 \leq i \leq r$) is the starting row of the zamboni.
- j ($1 \leq j \leq c$) is the starting column of the zamboni.
- n ($1 \leq n \leq 10^{18}$) is the number of steps ($numSteps$ in PacMan's algorithm)

Note that row 1 is the top row, and column 1 is the left column.

Output

Output the state of the ice rink after the zamboni has completed its run, as a $r \times c$ grid of characters. Each character should be a capital letter (A-Z) representing the color of the ice, or a dot (.) if the ice is white, or an 'at' sign (@) if the cell is the final location of the Zamboni.

Example

Input	Output
5 5 3 3 4	....BBC ..A.C ....C @DDDD
5 5 3 3 7	EG... E@BBC EGA.C EG..C FGFFF

Problem J. Zig Zag Nametag

Source file name:	zigzag.c, zigzag.cpp, zigzag.java, zigzag.py
Input:	`Standard`
Output:	`Standard`

When ninjas go to conferences they wear fake nametags. One ninja in particular wants to impress his Sensei. His Sensei chooses a new favorite number every day. The pupil wants to put a name on his nametag that encodes his Sensei's favorite number! This name will consist of only lower case letters. He assigns a value to each letter, based on its position in the alphabet (e.g. a = 1, b = 2, ..., z = 26). Then, he encodes the Sensei's number by adding up the absolute values of the differences of every consecutive pair of letters. For example, the string `azxb` has the value of:

$$|a - z| + |z - x| + |x - b| = |1 - 26| + |26 - 24| + |24 - 2| = 49$$

The name that the ninja will write on his nametag is the shortest string that encodes to his Sensei's favorite number. If there's more than one string of the shortest length, he'll choose the one that comes first alphabet- ically. Given the Sensei's favorite number, k, find the string that the ninja should put on his nametag.

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. Each input consists of a single line with a single integer k ($1 \leq k \leq 1000000$), which is the Sensei's favorite number. There will always be a name that encodes to the Sensei's number.

Output

Output a single line with a string of lower case letters, which is the name that the ninja should put on the nametag to impress the Sensei.

Example

Input	Output
1	ab
19	at
77	aoazb

Problem K. Contest Score

Source file name: contestscore.c, contestscore.cpp, contestscore.java, contestscore.py
Input: Standard
Output: Standard

PC^2 SCOREBOARD			
Rank	Name	Solved	Points
1	RobotU	BuggyBot	42
2	MonstersInc	BuggyBot	43

You are participating in the Association for Computing Machinery's Intercollegiate Programming Competition (ACM ICPC). You must complete a set of n problems. Since you are an experienced problem solver, you can read a problem and accurately estimate how long it will take to solve it, in a negligible amount of time. But you can only keep the details from k problems straight at any given time.

Let t_i be the time it will take to solve the ith problem. Your strategy for the contest is as follows:

1. Read the first k problems.
2. Choose a problem that you have read that will take the shortest time to solve (if there are ties, choose any of them arbitrarily).
3. Solve the problem, and read the next unread problem (if there is any).
4. If there are still unsolved problems, go back to step 2.

You want to calculate your total penalty time for the contest. Your penalty time for the contest is defined by the sum of submission times for all the problems. Given the ordering of problems in the contest, compute the penalty time of your strategy.

Input

The first line of input contains two space-separated integers n and k ($1 \leq k \leq n \leq 300$).

The ith line of the next n lines contains a single integer t_i ($1 \leq t_i \leq 10^6$).

Output

Print, on a single line, a single integer repesenting the total penalty time.

Example

Input	Output
4 3 1 3 2 1	14

Problem L. Equality

Source file name:	equality.c, equality.cpp, equality.java, equality.py
Input:	`Standard`
Output:	`Standard`

You are grading an arithmetic quiz. The quiz asks a student for the sum of the numbers. Determine if the student taking the quiz got the question correct.

Input

The first and the only line of input contains a string of the form:

$a + b = c$

It is guaranteed that a, b, and c are single-digit positive integers. The input line will have exactly 9 characters, formatted exactly as shown, with a single space separating each number and arithmetic operator.

Output

Print, on a single line, `YES` if the sum is correct; otherwise, print `NO`.

Example

Input	Output
`1 + 2 = 3`	`YES`
`2 + 2 = 5`	`NO`