Problem A. Beth's Cookies

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

This is the story Alph told his schoolmate Beth. "Last vacations, I went exploring that big abandoned house in the woods above the dam lake. I was very proud at that time to be freshly admitted to Tomorrow Programming School, and so I decided to stick to firm and clear rules of exploration and to make a log of my actions.

Inside the house, there were only rooms connected by doors between them. No corridors, no hallways. Whenever I entered a room for the second, third, etc. time, I did it always through the door by which I had left it last time. Whenever I went through a door I wrote down one letter. When I moved from a room to an adjacent room which was either undiscovered yet or which I had discovered later than the current room, I wrote down F, like Forward. When I moved to a room which I had discovered earlier than the current room, I wrote down B, like Backward. By discovery of a room I mean entering the room for the first time.

I started and finished my exploration in the entrance room of the house. When I finally left the house, I had a sequence of letters, in the order I wrote them down during my exploration. Later, to make it more suitable for some future automated processing, I substituted each F by an opening bracket and each B by a closing bracket. Here is my modified sequence. Do you think it can be used for some unusual programming task?"

It took Beth only seconds to reply. "Surely, it can be used. Insert symbol `*` between each `)(` pair. Insert symbol 1 between each `()` pair. And also, insert pair of symbols +1 between each `))` pair. It will result in an arithmetic expression, and I will give you as many cookies, as will be the value of the expression, but you have to calculate it first, obviously."

Input

The first input line contains even integer N ($2 \leq N \leq 100$), the length of the sequence of bracket which resulted from Alph's exploration. The second line contains the sequence itself, without any spaces or additional symbols.

Output

Output the number of cookies Alph will receive from Beth.

Example

Input	Output
10 ((())(()))	5

Problem B. Clubbing

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

Students of Tomorrow Programming School engage in different school programming clubs. Each student is a member of some number of clubs. The clubs are supervised by the Principal Club Coach (PCC). His main occupation is to talk to club members to help them organize their activities. PCC has a fixed schedule in the form of a list of students he is going to talk to in the nearest future. He always talks to only one student at a time, he may talk to a student repeatedly at various times. Each talk takes short time interval, which is always the same. There is negligible time between subsequent talks.

Currently, the school director also needs to talk to some students in the presence of PCC, because he needs to start another state supported programming project. He is going to visit PCC's cabinet and spend some uninterrupted time there. In that time, he wants to talk to all members of at least one club. Thus, PCC defined a so-called director interval in his schedule. It is an uninterrupted sequence of his talks to students, in which all members of at least one club appear at least once.

Before he suggests an acceptable director interval, PCC at least wants to know the number of such intervals in his schedule.

Input

The first input line contains one integer N ($1 \leq N \leq 10^5$), the number of student clubs. Next N lines contain the list of club members, each line specifies one club. One club is specified by a string without spaces, in which each member is represented by a single character. All characters in the string are different. The last input line contains the schedule of PCC, in the form of nonempty string with at most 10^5 characters, each character represents one student. In all strings, each character is one of the first 17 lowercase letters in the alphabet("`a`" - "`q`").

Output

Output one integer, the number of director intervals in PCC's schedule.

Example

Input	Output
2 pid lid lidp	3
2 baf lek affleck	4

Problem C. Digitalisation

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

The students of Tomorrow Programming School are admitted to the school in the nationwide School Admission Process (SAP), which involves most schools in the region. SAP consists of two rounds.

At the beginning of the first SAP round, each student applies for two schools, while indicating his priorities. For each student there is a school which is his first priority and a school which is his second priority. All students who apply are tested in a national exam that gives each student a unique score and creates a common list CL of students sorted according to their scores in the test in descending order (the highest score is the best). CL is available to each school. Each school prefers to admit students with better score. The capacity of each school is bounded by common capacity limit C. Each school keeps a list of candidates, the capacity of the list is also C. The lists are initially empty.

Then, the second round of SAP starts. It consists of one or more cycles of so-called update events. In a cycle, each school, one after another, performs one update event.

In an update event, the school with update list L searches CL from its end (starting with the worst score applicant) and they select the first applicant A who is not in L and who fulfills all following conditions:

- Either L is not full, or the score of A is better than the worst score of the candidates in L.
- The school with list L is either the first or the second priority of A.
- A is currently either not on a candidate list of any school or they are currently on candidate list of their second priority school.

If the school cannot select an applicant fulfilling the conditions, the update event is claimed to be empty and it is terminated. Otherwise, if L is full at the moment, the candidate with worst score in L is removed from L. Next, A is included in L. If A had been to this moment on a candidate list of another school, A is removed from that list. Finally, the update event is claimed to be valid.

If, during a cycle, at least one update event is valid, a new cycle is started all over again. The second round of SAP ends when all update events in one cycle are empty. Then, the schools admit the students which are on their respective candidate lists.

The regional statisticians want to know, how many students were admitted to their first or second priority schools. Write an appropriate program for them.

Input

The first input line contains three integers N, M, C ($2 \leq N, M \leq 10^5$; $1 \leq C \leq 100$), the total number of student applicants, the number of schools, and the common admission capacity. Schools are labeled by integers 1, 2, ..., M. Each of the following N lines contain the first and the second priority of one student, represented by the labels of the schools. The lines are sorted in descending order of the scores of the corresponding students.

Output

Output two numbers on a single line. The number of students who got to their desired school with first priority and the number of students who got to their desired school with second priority.

Example

Input	Output
9 3 4 1 2 2 3 1 3 3 2 1 2 3 2 2 3 2 3 2 1	9 0
4 2 1 1 2 1 2 1 2 1 2	1 1

Problem D. Expressions

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

Part of the training in arithmetic classes in Tomorrow Programming School consists of quick evaluation of a given arithmetic expression, in the head only. Fortunately, the expressions contain only positive integers, no brackets, and only three operations: addition, subtraction, and multiplication. Thus, the result is guaranteed to be an integer. Moreover, at introductory stages, a student has to evaluate only the parity of the result - is it odd or even?

Unfortunately, the professor presenting the expressions on the blackboard is known to be quite absent-minded. He often rewrites various numbers in an expression, even more times, while the students are already calculating. Typically, it changes the value of the whole expression, and the students have to start their calculations all over again.

The students decided to write a program which would help them. The input of the program will be the original expression, and the sequence of subsequent value changes in it. For each change, the program will calculate the value of the modified expression, possibly even without recalculating the whole expression from scratch.

Input

The first input line contains two numbers N and M ($1 \leq N, M \leq 10^5$), the number of integers in the expression, and the number of subsequent value changes in the expression. The second input line contains N token-separated integers A_1, ..., A_N ($1 \leq A_i \leq 10^9$). Each token is $+$, $-$, or $*$ (plus, minus, star (multiplication)). Each of the next M lines describe one value change in the expression. It contains two space-separated numbers X ($1 \leq X \leq N$) and Y ($1 \leq Y \leq 10^9$), which means that the X-th number in the expression was changed to value Y.

The changes are considered to be subsequent, i.e., when a next change appears on a different position, the previous change in the expression is preserved.

Output

Output $M + 1$ lines. The i-th line contains either "even" or "odd", depending on whether the input expression evaluates to an even or to an odd value after application of the first $i-1$ changes. In particular, the first output line indicates whether the expression evaluates to an even or to an odd value before any change was done. The evaluation follows standard arithmetic rule of operations precedence. Multiplication is granted higher precedence than addition and subtraction.

Example

Input	Output
6 4 11 + 22 * 33 - 44 * 55 * 66 1 2 2 3 4 5 3 5	odd even odd odd odd

Problem E. Fragmentation

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

Not so long ago, a rare type of meteorite, called mesosiderite, fell on the premises of Tomorrow Programming School and was immediately collected. The school was given the honour of organizing the cutting of the meteorite into smaller pieces, which will be exported to laboratories around the world. Swiss Precision Cutting Agency (SPCA) was hired to do the cutting job.

SPCA operates various cutting machines. Each machine can cut the meteorite or a piece of the meteorite into a number of smaller pieces, each of which weighs exactly the same. The number of resulting smaller pieces is the characteristic of that particular machine and cannot be changed. SPCA is quite sure that each of their machines can process any number of meteorite pieces in one day. A piece of meteorite produced by a machine cut cannot be cut again by the same machine in the same day, because of possible cross-contamination and ensuing loss of precision.

Each day, there is exactly one cutting machine available. The order of available machines in successive days is fixed. SPCA created a list of days in the near future and assigned to each day the characteristic of the machine available that day. SPCA calls this list a cutting schedule. The cutting process will take one or more days to complete, before a satisfactory number of meteorite pieces is obtained.

There are a few additional rules governing the cutting and exporting process.

- All laboratories demand that they all receive the same total weight of meteorite pieces.
- No meteorite piece can be shared among laboratories.
- Each piece of the meteorite must go to some laboratory, there should be no leftovers.
- At the end of each day in the cutting process, the weights of all meteorite pieces must be the same, to simplify cutting management.
- The whole cutting process has to be completed in a period of successive days, which should not be interrupted by any day without cutting.

The school has been given the cutting schedule and it is up to them to choose when to perform the actual cutting. A cutting period can be any sequence of consecutive days in the cutting schedule. Clearly, some cutting periods are favourable and some are not. A cutting period is favourable if a cutting process that starts in the first day of the cutting period would produce meteorite pieces, which satisfy the laboratories' demands at the end of the last day of the cutting period.

The school needs a program that can decide for a cutting period whether it is favourable or not.

Input

The first input line contains integer N ($1 \leq N \leq 10^5$), the number of days in the cutting schedule. The next line contains the cutting schedule in the form of integer sequence $a_1, a_2, \ldots, a_N$ ($1 \leq a_i \leq 10^6$). The value a_i represents the characteristic of the machine available on i-th day in the cutting schedule. The next line contains integer Q, ($1 \leq Q \leq 10^5$), the number of following queries. Each of the next Q lines represents one query and it contains three integers s_i, t_i, k ($1 \leq s_i \leq t_i \leq N$, $1 \leq k \leq 10^6$), the first day of the cutting period, the last day of the cutting period, and the number of laboratories involved.

Output

For each input query output, on a separate line, `Yes` if the cutting period specified by the query is favourable, otherwise output `No`.

Example

Input	Output
8 2 3 6 12 4 8 16 4 8 2 4 72 1 8 7 1 4 16 1 4 32 4 4 6 5 5 4 2 5 864 2 5 1296	Yes No Yes No Yes Yes Yes No

Problem F. Golem Coordinated Derby

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

Robotic labs in Tomorrow Programming School produce minirobots in big numbers. To perform various complex tasks, robots often form teams. Before the task begins, a team measures its strength. The robots in the team select one robot among themselves to be the team captain. Next, the robots arrange themselves in one row behind the captain. Each robot refers to the captain the value of the greatest common divisor of its own height and the height of the neighbour robot standing directly in front of it. That value predicts the strength of the bond between these robots when they perform the task. The captain totals all received values and claims the total to be the strength of the team.

The height of each robot is always expressed in centimeters and it is an integer ranging from 1 to 20. The strength of the team depends on the order of the robots in the row behind the captain. Also note, that a selection of the captain also influences the team strength. Any robot in a team can be selected as its captain.

The robots in a team always tend to maximize the team strength by selecting an appropriate captain and positioning themselves appropriately in the row. However, that is not an easy exercise for the robots, because checking all their possible arrangements is often beyond their computational scope.

Input

The first input line contains one integer N ($2 \leq N \leq 10^5$), the number of robots in the team. The second line contains N space-separated integers A_i ($1 \leq A_i \leq 20$), the list of heights of all robots in the team.

Output

Output a single integer, the maximum strength of the team specified in the input.

Example

Input	Output
7 2 3 12 4 6 4 3	22

Problem G. Hamster

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

Hamster Joe is one of the beloved pets of Tomorrow Programming School. The biology department entrusted the school's "Central Robot Endowed with Smashing AI" (CRESAI) with the job of projecting and building a playpen for Joe. This playpen will be situated in a newly designated animal space within the department. This space is a flat horizontal area, tiled with uniformly sized square tiles of unit side length, that come together to create a rectangular grid marked by grooves where tiles meet. The playpen will be bordered by unit-length wall segments, each rising from the groove between two adjacent tiles, ensuring that both ends of every segment align with tile corners.

Joe can move from his current tile to any adjacent tile if the tiles are not separated by a wall segment. Joe cannot jump or crawl over any wall segment and Joe also cannot squeeze himself between two adjacent wall segments, or smash himself through them. Thus, when the locations of the wall segments are chosen appropriately, they form an enclosure, from which Joe would not be able to escape.

Surprisingly, CRESAI was not up to the task. It positioned each wall segment of unit length correctly, in the sense that each segment's base now sits in a groove and its ends coincide with the corners of a tile. However, it seems that most of the wall segments were positioned randomly so that the existence of any enclosure of any shape is not guaranteed.

To fix the problem at least temporarily, biologists are looking for the minimum number of additional wall segments of unit length, which, when installed at appropriately selected unused grooves, would create an enclosure of any shape or size. In the resulting layout, some of the wall segments originally installed by CRESAI may remain useless.

Input

The first line contains one integer M ($1 \leq M \leq 10^5$), the number of wall segments installed by CRESAI. Next, there are M lines, each describing one wall segment installed by CRESAI. A wall segment is described by four integers x_1, y_1, x_2, y_2, where x_1, y_1 are the coordinates of one end of the wall segment and, x_2, y_2 are the coordinates of the other end of the segment. The axes of the system of coordinates are parallel to the grooves and the coordinates of each grove intersection are integers. For all wall segment coordinates, it holds $-10^9 \leq x_1, y_1, x_2, y_2 \leq 10^9$ and $|x_1 - x_2| + |y_1 - y_2| = 1$.

Output

Output one line with the minimum number of additional wall segments which would create an enclosure of any size or shape, from which Joe would not be able to escape.

Example

Input	Output
9 0 0 0 1 0 1 1 1 1 1 1 2 1 2 0 2 0 2 -1 2 -1 2 -2 2 -2 2 -2 1 -2 1 -2 0 -2 0 -1 0	1

Explanation

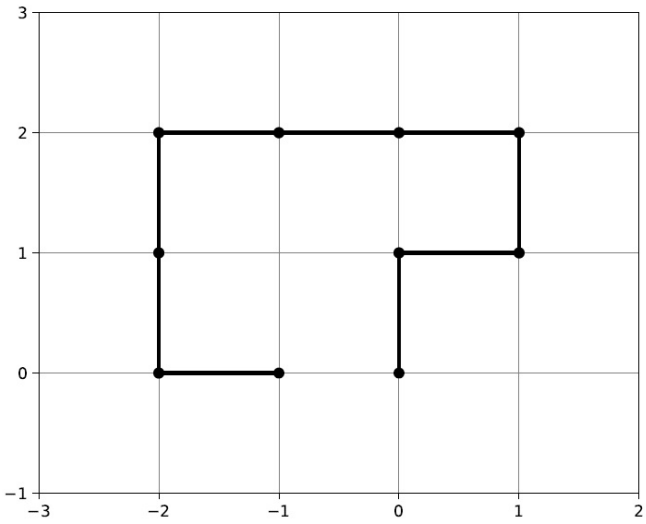


Figure 1: Rendering of the example input.

Problem H. Movers

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

Department of successful computing in Tomorrow Programming School is famous for stockpiling large quantities of desks and monitors in their labs, it serves well to the whole community.

Any time a meeting is being held in a lab, the number of desks and monitors in it should be sufficient to serve all participants. In case of necessity, additional desks and/or monitors may be borrowed from neighbouring labs. In extreme cases, all desks and all monitors from all neighbouring labs may be brought in. Thus, available desks and monitors in a lab are exactly those desks and monitors which are in the lab itself and in all its neighbour labs. Desks or monitors are never transported from more distant labs, it is deemed ineffective and accident prone. After a meeting, all borrowed desks and monitors are returned back to their original labs, before any other meeting starts.

The desired configuration for a meeting is when the number of available desks and monitors in the lab is equal. Often, the number of available desks in a lab is either smaller or bigger than the number of available monitors, and that creates specific problems for the maintenance staff each time.

The inflow of desks and monitors to the department is rapid. Frequently a shipment of desks and monitors arrives to the department and its contents is added to various labs. It is added immediately or immediately after the end of the current meeting.

To plan the meetings, and equipment maintenance as well, it is important to know how many desks and monitors are available in any lab at any moment. The department needs a program that can process two kinds of queries. The first kind of query specifies a number of desks or monitors that have just been added to a particular lab. The second kind of query asks for a relation between the number of available desks and monitors in a particular lab.

Input

The first input line contains three integers N, M, Q ($1 \le N \le 10^5$, $0 \le M \le 10^5$, $0 \le Q \le 10^5$), the number of labs, the number of pairs of labs which are neighbours to each other, and the number of queries. Labs are labeled by integers 1, 2, ..., N. The second line contains N space-separated integers D_i ($0 \le D_i \le 100$), the number of desks in lab i. The third line contains N space-separated integers E_i ($0 \le E_i \le 100$), the number of monitors in lab i. Next M lines contain space-separated unique pairs of distinct integers a_i, b_i, ($1 \le a_i, b_i \le N$), the labels of pairs of neighbour labs. Next Q lines contain the queries, one query per line. Each query is provided in one of the two formats.

1. `add <count> desk/monitor <label>` - query increases the number of desks or monitors by the given `<count>` in a lab with specified `<label>`.
2. `check <label>` - query asks for a relation between the number of available desks and available monitors in a lab with specified `<label>`.

One `add` query increases the number of desks or monitors in a lab by at most 100.

Output

Output Q lines, one for each query of type check, in the order they appear in the input. Each line contains one of the strings "`desks`", "`monitors`", or "`same`", depending on whether there are more available desks or more available monitors in the lab specified by the query, or whether their numbers are equal.

Example

Input	Output
4 5 8 1 1 1 0 2 0 2 0 1 2 2 3 3 4 4 1 1 3 check 2 add 2 desk 1 check 2 add 1 monitor 3 check 1 check 2 check 3 check 4	monitors desks same same same monitors

Problem I. Natatorium

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

Tomorrow Programming School Department of Arithmetics is going to build a new pool in the basement of their building. The pool will serve mainly for testing of innovative floating point arithmetic algorithms, especially their floating properties in different kinds of liquids. Occasionally, the members of the staff will be allowed to the pool as well. To make the pool optimally fit for human needs, the department has to decide on the pool dimensions. The pool will be rectangular, it must not be a square. Its exact surface area C is predetermined by the choice of the relevant algorithms to be tested there. The company they hired to build the pool offers the list of possible pool side lengths. They have studied the demand in detail and they guarantee that a pool with the given area can be built using the lengths in the list. However, the particular choice is up to the department.

As the presented list is relatively long, the department also hired a junior programmer who is going to find out which lengths from the list can be chosen. Before he started his job, the programmer had made interesting observations. First, all available pool side lengths are primes. Second, the surface area C of the pool is a product of two distinct primes. He thinks this information may be helpful to find appropriate side lengths reasonably quickly.

Be faster than the hired programmer and solve the problem first.

Input

The first input line contains integer C ($1 \leq C \leq 10^{18}$), the area of the pool surface. The second line contains integer M ($1 \leq M \leq 2 \cdot 10^5$), the number of pool side lengths offered by the company. The third line contains a list of space-separated unique primes $P_1, \ldots, P_M$ ($1 < P_i < 10^9$), representing the offered pool side lengths.

All measurements are expressed in the same units.

Output

Output a single line with two space-separated pool side lengths from the offered list, which can be used to build the pool. The first side length has to be smaller than the second one.

Example

Input	Output
15 5 7 2 5 11 3	3 5

Problem J. Proglute

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

Baroque opera geeks in Tomorrow Programming School are developing a new string instrument suitable for their innovative performances. The instrument, named programmed lute, or proglute in short, consists of flat circular body, with N pegs situated on the body perimeter, and labeled by integers from 1 to N.

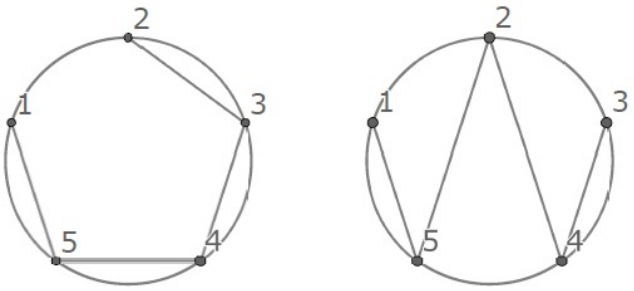
Each string on the instrument is stretched between two different pegs and runs across the proglute's body without crossing other strings. To enhance complex resonance effects, the developers decided to attach two strings to all but two pegs, called principal pegs. Only one string is attached to each principal peg. To support glissando effects, the strings are arranged in such way that it is possible for a musician to touch the string at one principal peg and then slide the finger along all strings on the instrument to the other principal peg. While sliding, musician does not remove the finger from a string, and skips from a string to another one only at their common end peg.

To build the instrument, there are many ways to arrange the strings on the proglute. Different arrangement would result in different musical properties of the instrument. The developers want to know the number of all possible arrangements of strings on the proglute. They introduced the following notions.

- The characteristic of a string is an unordered pair of labels of pegs at the string ends.
- The characteristic of a proglute is the set of all its string characteristics.
- Two strings arrangements on proglute are considered to be different when their corresponding characteristics are different.

Calculate the number of different string arrangements on the proglute.

The figure below shows two possible solutions for a proglute with five pegs:



Input

The input consists of single line with an integer N ($2 \leq N \leq 1000$), the number of the pegs on the proglute perimeter.

Output

Output a single number equal to the number of mutually different arrangements of the strings on the proglute mod $10^9 + 7$.

Example

Input	Output
5	20
666	61847156

Problem K. Screamers in the Storm

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

Students visiting the Multidimensional Data Cabinet at Tomorrow Programming School may get access to multidimensional hyperspheres stored in air-conditioned helium cupboards in the cabinet. Before exploring a chosen sphere, a student has to prove he/she is knowledgeable enough to handle the sphere. There are spheres of various dimensions and various integer radiuses, each in a separate locked cupboard. The dimension and the radius of the sphere are written on the cupboard label. To unlock the cupboard, the student has to enter a specific value into the lock mechanism. The value depends on the dimension and the radius of the sphere. It is equal to the sum of absolute values of all coordinates of all integer points which lie inside or on the surface of the sphere, when the center of the sphere is exactly in the center of coordinates. The sum has to be entered modulo $10^9 + 7$.

An integer point in a d-dimensional Euclidean space is a point which all d coordinates are integers.

The Cabinet manager is planning to obtain more spheres in the future, and you know that calculations which help to unlock some high-dimensional spheres may be quite tedious to perform by hand.

Write a program that calculates the value necessary to unlock a cupboard with a sphere in the cabinet.

Input

The input contains two integers D, R ($1 \leq D \leq 50$, $1 \leq R \leq 50$), the dimension and the radius of the sphere inside a cupboard.

Output

Print the value which unlocks the cupboard with the given sphere, modulo $10^9 + 7$.

Example

Input	Output
1 6	42
3 5	2850

Problem L. Wall

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

Tomorrow Programming School is going to decorate the front wall in the entrance hall with a distinctive pattern based on a visually appealing output of an elementary cellular automaton. The designers are going to study the outputs of several cellular automata and choose the one they like the most.

Your task is to write a program that replicates the output of an automaton.

Now, we describe elementary cellular automata and how they work. An elementary cellular automaton is a system consisting of a row of adjacent cells, and a rewriting rule. Each cell is always in one of two states: 0 or 1. The sequence of states of all cells, in the order of cells in the row, is called a generation.

The automaton operates in cycles. In one cycle, the current generation is taken as input and a new generation is calculated by applying the automaton rule to each cell in the current generation. At the end of a cycle, the current generation is replaced by the new generation. Then, another iteration of the cycle can start. The cycle can be repeated for any number of times.

It is assumed that the leftmost and the rightmost cells are also adjacent to other unused cells, which are always in state 0. This assumption ensures the automaton rule can be applied in the same way to all cells in the row. The unused cells do not appear in any input or output.

The state of a particular cell in the new generation is determined by its own state and the state of its two adjacent cells in the current generation, and by the automaton rule.

Let us consider a triple of cells consisting of a particular cell C, and its left and right adjacent cell. There are $2 \cdot 2 \cdot 2 = 8$ possible states, in which this triplet can be in the current generation: 0 or 1 in the left adjacent cell, 0 or 1 in the cell C itself, 0 or 1 in the right adjacent cell. Explicitly, all possible states of this triple of cells may be written as a sequence S of eight binary numbers: $S = (111, 110, 101, 100, 011, 010, 001, 000)$. The middle digit denotes the state of C, and the first and the third digits denote the states of the left and the right adjacent cell to C, respectively.

The automaton rule assigns one bit, 0 or 1, to each of the eight binary numbers in S. The rule is then coded as an 8-bit vector of the bits assigned to binary numbers in S. The sequence S itself is not coded in the automaton rule.

Application of the rule on cell C consists of identifying the binary number in S which corresponds to the states of C and its two adjacent cells in the current generation. The state of C in the next generation is determined by the bit value assigned by the automaton rule to that particular binary number.

There are $2^8 = 256$ different automatons, distinguished by their rule codes. In the input of this problem, the 8-bit vectors representing automaton rules are coded in decimal.

Input

The first line of the input contains two integers, R and K ($0 \le R \le 255$, $1 \le K \le 200$), the automaton rule coded in decimal and the number of generations, respectively. The second line of the input defines the first generation of the automaton. The cell states are coded by characters, with '`.`' coding 0 and '`X`' coding 1. The row of cells in the first generation is coded as a single string without spaces. The width of the row is at least 1 cell and it does not exceed 250 cells.

Output

The output contains the following K generations produced by the given automaton. The generations are coded and formatted in the same way as the first generation in the input. Each generation occupies one line, there are no empty lines in the output.

Example

Input	Output
128 5 XXXXXXXXXXXX	.XXXXXXXXXX. ..XXXXXXXX.. ...XXXXXX... ....XXXX.... .....XX.....
30 10 .....X.....	.....XXX..... .....XX..X..... .....XX.XXX..... .....XX..X...X..... .....XX.XXX.XXX..... .....XX..X....X..X..... ....XX.XXX..XXXXX.... ...XX..X...XXX.....X... ..XX.XXX.XX..X...XXX.. .XX..X....X.XXX.XX..X.