Problem A. Abat-jour (Division 1 Only!)

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	256 mebibytes

The great mathematician John spent hours staring at his exciting abat-jour. Lamps in it formed a regular polygon. The same old story...

The bright advertisement said that it is nice, cool, has N lamps of K brand colors and is M-regular (where suprisingly M divides N). Abat-jour is said to be M-regular if for every M consecutive lamps the next M consecutive lamps are of the same color and in the same order. John was very surprised when he had found that one lamp was of dirty grey color, absolutely no way that it was brand. So there were some problems with M-regularity too. Abat-jour was semi-M-regular, so for every M consecutive lamps the colors of the next M consecutive lamps differs in no more than one lamp. As expected thousands of semi-M-regular abat-jours with one dirty grey lamp conquered John's mind.

You were hired by a researcher to help unlift the veil over John's genius. (Yeah, this man likes to spend money on weird things). As the second task you have to calculate the number of semi-M-regular abat-jours with N lamps, exactly one dirty grey lamp, while the other have to be of one of K brand colors, in the mind of the great mathematician. And you only have to calculate it modulo 10^9+7. If two abat-jours can be turned one into another by rotation, those abat-jours are counted as identical.

Input

In the only input line you are given 3 integers N, M, K, separated by spaces ($2 \leqslant N, M, K \leqslant 500$, M divides N) — number of lamps in abat-jour, regularity number and number of brand colors respectively.

Output

Print the only number — answer to the problem.

Examples

standard input	standard output
8 2 2	16

Problem B. Winery

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	256 mebibytes

A start-up winery needed to solve a difficult problem with wine storage.

As everyone knows, wines vary mainly based on the year the grapes were harvested. This is why k categories of wine are distinguished, which differ only in their age. The first category is from 0 to a_1 years old, the age of the wines in the second category is in half-intervals $[a_1, a_1+a_2)$, the third category — $[a_1 + a_2, a_1 + a_2 + a_3)$, ..., the k-th — $[a_1 + a_2 + \ldots + a_{k-1}, a_1 + \ldots + a_k)$. To improve wine quality, an international convention dictates that a_{i+1} must be divisible by a_i.

Different categories of wine should be stored in different ways: different conditions for temperature, humidity, and so on. To offer a full range of wines, the wine company made a decision to always store all categories of wine. However, to cut costs, the winery built exactly k wine cellars for storage.

Help the owners arrange the process so that each year all the wine categories that could have been produced up to this year are always in stock. The first year, the grape harvest was very fruitful and all the wine cellars were filled with wine from that year's grapes (0 years old).

Make a plan for the company for the next n years. For each year, output which wine cellar (numbered starting from 1) the wine from the new harvest should be put in. If this cellar already has wine in it, the wine must be sold immediately.

If a year's wine does not need to be put in a cellar, output 0. In this case, the new wine will be sold at the nearest farmer's market.

Input

The first line of input has two integers n and k ($1 \le n \le 10^5$, $1 \le k \le 15$). The next line has k integers a_i ($1 \le a_i \le 10^9$). It is guaranteed that a_{i+1} is divisible by a_i.

Output

Output n integers — the numbers of the cellars where new wine should be put for the corresponding year. Output 0 if the wine for this year should be sold off. After each harvest, the cellars should contain wines from each category whose minimal age is not more than the age of the company.

Examples

standard input	standard output
3 1 1	1 1 1
3 2 1 1	1 2 1
10 2 2 4	0 1 0 1 0 2 0 2 0 1
10 3 1 2 2	1 1 2 2 3 3 1 1 2 2

Problem C. Cdecl

Input file:	standard input
Output file:	standard output
Time limit:	2 seconds
Memory limit:	256 mebibytes

Even relatively new C programmers have no trouble reading simple C declarations such as

```
int      foo[];     // foo is an array of ints
char    *foo;        // foo is a pointer to char
```

but as the declarations get a bit more involved, it's more difficult to know exactly what you're looking at.

```
char *(*(**foo[][])())[]; // ???
```

Bjorn decided to develop a simplified interpreted language based on C language — C−−.

You write a part of the interpreter, which defines declaration by some expression which used onle one declared name.

I.e. variable `tmp` must have declaration `char (*tmp[])` if we want to store expression `(*((**(foo[10][4]))()))` in it.

Thus, you need to describe a variable whose type is exactly the type of expression.

Declared variable is identified by name. In addition to one variable name, a declaration is composed of one "basic type" ('`int`', '`char`' or '`void`') and zero or more "derived types". A declaration can have exactly one basic type, and it's always on the far left of the expression.

The "basic type" are augmented with "derived types", and C−− has three of them:

- `*` — pointer to "..."

  This is denoted by the familiar * character, and it should be self evident that a pointer always has to point to something.

- `[]` — array of "..."

  "Array of" should be undimensioned – `[]`, because the sizes don't really play significantly into reading a declaration. It can be only at right side from array name.

- `()` — function returning "..."

  This is usually denoted by a pair of parentheses together – `()`. We'll note that parentheses used to represent "function returning" are different than those used for grouping: grouping parens surround the variable name, while "function returning" parens are always on the right.

The "array of" `[]` and "function returning" `()` type operators have higher precedence than "pointer to" `*`, and this leads to some fairly straightforward rules for decoding.

The correct declaration of the variable in the language C−− satisfies the following grammar

```
decl : type_spec declarator
type_spec : 'void' | 'char' | 'int'
declarator : pointer direct_declarator | direct_declarator
pointer : '*' | '*' pointer
direct_declarator : id | '(' declarator ')' | direct_declarator '[' ']'
                        | direct_declarator '(' ')'
```

where `id` is the identifier, consisting of Latin letters, digits and the underscore character and does not start on the digit.

Semantics of the declaration:

1. Find the identifier. This is name which are declared. On a piece of paper, write "declare identifier as".
2. Look to the right. If there is nothing there, or there is a right parenthesis ")", goto step 4.
3. You are now positioned either on an array (`[]`) or function (`()`) descriptor. There may be a sequence of these, ending either with an unmatched right parenthesis or the end of the declarator. For each such descriptor, reading from left to right: if an empty array "`[]`", write "array of" if a function "`()`", write "function returning" Stop at the unmatched parenthesis or the end of the declarator, whichever comes first.
4. Return to the starting position and look to the left. If there is nothing there, or there is a left parenthesis '(', goto step 6.
5. You are now positioned on a pointer descriptor, '*'. There may be a sequence of these to the left, ending either with an unmatched left parenthesis '(' or the start of the declarator. Reading from right to left, for each pointer descriptor write "pointer to". Stop at the unmatched parenthesis or the start of the declarator, whichever is first.
6. At this point you have either a parenthesized expression or the complete declarator. If you have a parenthesized expression, consider it as your new starting point and return to step 2.
7. Write down the type specifier. Stop.

So, the name are declared as an "array of" or "function returning" or "pointer to" some type.

For example, `foo` is "array of " "array of" "pointer to" "pointer to" "function returning" "pointer to" "array of" "pointer to" char.

The correct expression to the variable given by the following grammar:

```
exp  : unary_exp
unary_exp  : postfix_exp | *unary_exp
postfix_exp  : primary_exp | postfix_exp '[' number']' | postfix_exp '(' ')'
primary_exp  : id | '(' exp ')'
```

where `number` is integer. So a postfix `[]` or `()` have higher priority than `*`. But other parens (surrounding `id`) have higher priority than `[]` or `()`.

There are semantics of the expression:

1. if we have expression `exp` as `id`, then `exp` has type of `id`
2. if we have expression `exp` of some type, then `exp` and `(exp)` have same type
3. if we have expression `exp` as "pointer to" "sometype", then `*exp` has "sometype" type
4. if we have expression `exp` as "array of" "sometype", then `exp[number]` has "sometype" type
5. if we have expression `exp` as "function returning" "sometype", then `exp()` has "sometype" type

Note, that expression and declaration are correct, and in C– function may returned array and other function.

Your task is to print the correct variable declaration in language C−−, whose type exactly matches the type of expression.

Input

The first line of input contains a string with a correct variable declaration.

The second line contains a string with a correct expression.

The third line contains a variable name, which type is required to describe.

The length of each declaration does not exceed $300,000$ symbols. The length of third string does not exceed 1000 symbols. There may be several additional spaces in expression.

Output

Print one line with a declaration for the required variable. This declaration must satisfy the grammar and must have the same type as expression. If there are several declarations with the same type (distinguishing by grouping parentheses) output any of them. The length of a declaration must not exceed $1,000,000$ symbols.

Examples

standard input	standard output
char *(*(**foo [][]) ())[] (*((* *(foo[10][4]))())) bar	char (*bar[])

Problem D. Matrix (Division 1 Only!)

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	256 mebibytes

Square matrix $N \times N$ containing only zeroes and ones is given. Change exactly L elements of the matrix so that it's determinant will be odd number. You can change only 0 to 1 or 1 to 0, and the same element can't be changed more than once.

Input

First line contains two integers N and L. $3 \le N \le 3000$, $1 \le L \le N$. N following lines contain strings of exactly N digits 0 or 1 without spaces — rows of binary matrix $N \times N$.

Output

If there is no solution, print one line "`-1 -1`". Otherwise output should contain L lines with two integers in each line: row and column of the changed element ($1 \le row, column \le N$). Rows are numbered from the top. If there are multiple solutions, print any of them.

Examples

standard input	standard output
3 3 100 001 010	1 2 2 2 1 3
3 1 110 110 110	-1 -1

Problem E. Stable network

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	256 mebibytes

A network of N computers needs to be set up so that if either of any two computers fails, information can still be transferred directly between any two remaining computers, directly or through other working computers.

Write a program that determines the smallest number of connections that can be made between computers to achieve this.

Input

The input is a single number N — the number of computers that need to be connected in the network ($4 \le N \le 10\,000$).

Output

Output a single number — the smallest number of connections between computers that can be used to set up the desired network.

Examples

standard input	standard output
4	6

Problem F. Dice Roll

Input file:	`standard input`
Output file:	`standard output`
Time limit:	3 seconds
Memory limit:	256 mebibytes

Lexa and Serega are playing the following game. First of all, Serega pays Lexa 6 Euro. Then he throws the m-side dice sequentially no more than N times (the sides of the dice have numbers $1, \ldots, m$ written on them; all sides have the same probability). At any moment he can see the results of all his rolls and decide to stop if he want (he can not stop before the first roll though). After he stops (either because he decided to stop or because all of the N rolls are finished), arithmetic mean of the roll results is calculated (that is, $X = (a_1 + \cdots + a_k)/k$, where k is the number of rolls and a_i are the results of the rolls). Then Lexa gives Serega X Euro and the game ends.

Serega wants to minimize his loss. In order to do it he wants to maximize the average result at the end of the game. Help him to do it.

Input

The only line of the input contains two space-separated integers N (the maximum number of rolls) and m (the number of dice sides). Limits are: $1 \leqslant N \leqslant 100$, $1 \leqslant m \leqslant 10^4$.

Output

The first line of the output should contain single floating-point number X — the maximum average result of the game (if Serega plays optimally). Answers with absolute or relative error less than 10^{-9} are considered correct.

Examples

standard input	standard output
3 6	4.0185185185

Problem G. GIT (Division 1 Only!)

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	256 mebibytes

An agile software company uses GIT as a versioning system for it's source code.

GIT system allows developers to commit their changes to a central repository. Each commit is atomic and successful, adding new changes to a code in the repository. Every commit has a unique timestamp when it's occurred. Thus the state of repository linearly changes in time with developers' commits.

At the start of the software development process GIT repository has one predefined branch called "master" which contains all the code. At any time a developer can make a new branch out of the existing one (this process is called branching) by copying all state and contents of the existing branch to a new one. After that each developer can make commits to that branch instead of the default "master" — all changes are stored in that branch only and do not affect "master" branch or any other branches. That technique is usually used for feature development.

After feature development is complete, a developer can merge his branch B back to "master" or any other already existing branch A. We'll assume that each merge is atomic and fully succeeds without any conflicts. Branch A is then considered to have all commits from branch A and all commits from branch B by the time when merge is completed. After merging both branches continue to exist.

Software company had already released several versions of it's product and had a number of branches when it figured out a severe bug is present in one of the commits in GIT system. You are about to find all branches that contain that bug now.

Input

Input contains a log of commands given to a GIT system until now. In the first line of input an integer N is given — the total number of GIT commands logged ($1 \le N \le 2000$). Command C_i is considered to be issued and completed at timestamp i.

On the next N lines of input GIT command lines are given. They can be in one of the following formats:

- commit (branch-name) — a commit to a branch with name "branch-name";
- branch (new-branch) from (existing-branch) — a new branch with name "new-branch" is created from "existing-branch";
- merge (from-branch) to (to-branch) — a branch with name "from-branch" is merged into "to-branch"

Branch names contain letters a-z and digits 0-9 only and their length does not exceed 16 characters.

The next line of input contains one integer K — a number of test cases to solve ($1 \le K \le 2000$). On the next K lines of input a single integer B is given — a timestamp of a developer commit that contains severe bug.

Output

For each test case from 1 to K output a single line with all branch names in alphabetical order (digits are considered to be "smaller" than letters for sorting purposes) delimited by one space character that now contain given severe bug.

Examples

standard input	standard output
5 commit master branch feature1 from master commit feature1 merge feature1 to master commit feature1 3 1 3 5	feature1 master feature1 master feature1

Problem H. Three Paths

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	256 mebibytes

The Forgotten Kingdom consists of several towns connected by bidirectional roads, each road has non-negative length. The citizens of the Kingdom strongly dislike ambiguity, and therefore the road system satisfies the following property: for each two cities there exists *exactly* one shortest path between them.

One fine day the citizens of the Kingdom completely forgot the map of the road system, including the number of cities, the number of roads and the lengths of the roads. Fortunately, they still remember the shortest paths for three pairs of cities. However, they are not sure if they remember these paths correctly. So, you task is to help them determine if for a given three paths there exists a road system such that these paths are unique shortest paths with respect to the road system.

Input

Input consists of three lines describing the three paths mentioned above. Each line starts with n_i — the number of cities in the path, followed by space-separated list of city numbers $a_{i,1}, \ldots, a_{i,n}$ in the order they appear on the path ($1 \leqslant n_i \leqslant 100\,000$; $1 \leqslant a_{i,j} \leqslant 10^9$, equal numbers correspond to the same cities, different numbers — to different cities).

Output

Output should contain one line with the word "`Yes`" if there exists such a road system, and "`No`" otherwise.

Examples

standard input	standard output
5 4 5 6 7 8 5 1 5 9 13 17 5 3 7 10 13 9	Yes
5 10 20 30 40 50 5 10 21 32 40 53 5 29 30 31 32 33	No

Problem I. Flatland Holidays

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	256 mebibytes

After analyzing the weekends and official holidays around New Year and May 1 in Russia, the president of Flatland came to the conclusion that he could dramatically optimize citizens' days off work. The main thing is that they would not have to work more than 6 days in a row in a calendar year. He instructed the Ministry of Labor to develop a schedule for moving the weekend days (Saturdays and Sundays) to other days, so that when combined with the official Flatland holidays, citizens could have as many days off as possible in a row.

You may recall that if an official Flatland holiday falls on a weekend (Saturday or Sunday), the day off for the holiday is automatically moved to the first working day after the holiday. But by the president's order, any day off, whether it coincides with a holiday or not, can be moved to any working day. Holidays, however, are never moved.

Write a program that will help the Ministry of Labor make the required schedule for moving weekend days in the coming year. Holidays and weekends in the past year or the coming year do not need to be considered. You need to maximize the number of days off in a row just for a single year.

Input

The first line of input contains two integers — the year number Y ($2012 \le Y \le 2050$) and the number of the day of the week for January 1 in this year W ($1 \le W \le 7$) from Monday to Sunday, respectively.

The second line has the number of yearly holidays N ($0 \le N \le 366$) in Flatland. Each of the following N lines has the date of a holiday in the format `DD.MM`. The holidays are listed in chronological order. All the dates are different and are correct for the given year.

Output

Output a single number — the maximum possible number of days that the people of Flatland can have off in a row during the specified year, if weekend days are moved such that the number of working days in a row will never be more than 6 in this year.

Examples

standard input	standard output
2012 7 1 01.01	63

Problem J. Cyclic Number

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	256 mebibytes

Input file contains one integer (its decimal representation has N digits). We cyclically shift digits of this number by $0,\ 1,\ 2,\ \ldots\ , N-1$ digits and multiply each of resulting numbers by its first digit. Calculate the sum of all N products.

Input

The only line contains one positive decimal integer without leading zeroes with not more than $250,000$ digits **(for division 1) or** $25,000$ **digits (for division 2).**

Output

The only line of output should contain the answer.

Examples

standard input	standard output
22	88
123	1521

Problem K. Building (Division 2 Only!)

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	256 mebibytes

The local building firm needs your help. They are building an apartment building where the walls are prefabricated and lifted in place using cranes. The building firm has located n possible locations for cranes, and needs to choose some of these so that the center of each wall can be reached by at least one crane. The cranes are quite expensive, so they want to use as few of them as possible. A crane can reach a wall if the wall's center is at most a distance r away.

The house that is to be built is rectangular with a length l and width w.

Find the minimum number of cranes required to reach the center of all four walls.

Input

The first line of input contains four space-separated positive integers l, w, n and r, all at most 30. l and w denote the length and width of the house, n denotes the number of possible crane locations, and r denotes the reaching distance of each crane.

This is followed by n lines, each containing two integers x and y ($-100 \le x_i, y_i \le 100$), denoting a possible location for a crane. The coordinate system has its origin in the center of the building and the x-coordinate along the length of the house.

Output

Print one integer, the minimum number of cranes required to reach all wall segments, or "`Impossible`" if not all wall segments can be reached.

Example

standard input	standard output
4 2 3 3 1 -2 4 0 -1 2	2

Problem L. Coffee or beer? (Division 2 Only!)

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	256 mebibytes

Your company built magical dispencer with coffee/beer switch in it (up for coffee and down for beer). We will divide the office population into those who prefer the coffee and those who prefer the beer.

Now, there are several possible policies that one could use, here are a few:

1. When you leave, always leave the switch up;
2. When you leave, always leave the switch down;
3. When you leave, always leave the switch as you would like to find it;

So, a person may have to toggle the switch prior to using the dispencer and, depending on policy, may need to toggle it before leaving.

Your task is to evaluate these different policies. For a given sequence of people's preferences, you are supposed to calculate how many times switch was toggled for each policy.

Input

The first and only line of input contains a string of characters 'U' and 'D', indicating that a person in the sequence wants the switch up (for coffee) or down (for beer). The string has length at least 2 and at most 1000.

The first character indicates the initial position of the switch, and the following $n-1$ characters indicate how a sequence of $n-1$ people will use the switch. You should compute the total number of switch toggles needed for each of the three policies described above.

Output

Output three numbers, each on a separate line, the total number of switch adjustments for each policy.

Example

standard input	standard output
UUUDDUDU	6 7 4

Problem M. Decoding (Division 2 Only!)

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	256 mebibytes

Alice and Bob have devised a code that encodes UPPERCASE words by shifting their letters forward.

Shifting a letter by S positions means to go forward S letters in the alphabet. For example, shifting `B` by $S = 3$ positions gives `E`. However, sometimes this makes us go past `Z`, the last letter of the alphabet. Whenever this happens we wrap around, treating `A` as the letter that follows `Z`. For example, shifting `Z` by $S = 2$ positions gives `B`.

New code depends on a parameter K and also varies depending on the position of each letter in the word. For the letter at position P , Alice and Bob use the shift value of $S = 3P + K$.

Write a program for Bob that will decode messages sent by Alice.

Input

The input will be two lines. The first line will contain the positive integer K ($K < 10$), which is used to compute the shift value. The second line of input will be the word, which will be a sequence of uppercase characters of length at most 20.

Output

The output will be the decoded word of uppercase letters.

Example

standard input	standard output
3 FXAB	ZOOM