ABC

Task ID: `abc`	Time limit: `0.2 sec`	Memory limit: `256 MB`

Given a string S, find the number of ways to delete some (possibly zero) characters from it to get the string "`abc`".

(Two ways of deleting characters are considered different if there is an index such that the character at that index is deleted in one case but not in the other.)

For example, for the string $S =$"`zaxbabboc`" there are five possible ways:

```
     S: zaxbabboc
way #1: .a.b....c
way #2: ....ab..c
way #3: ....a.b.c
way #4: .a....b.c
way #5: .a...b..c
```

Input

The only line of input contains the string S. S will consist of between 1 and 1 000 000 characters, inclusive. Each character of S will be a lowercase English letter.

Constraints

There are 25 batches of input, each worth 4 points. In the first 5 batches the length of S is at most 100 and in another 5 batches the length is at most 10 000.

Output

Output a single line with a single integer: the number of ways to get "`abc`".

Sample tests

input	output
`zaxbabboc`	`5`

input	output
`abc`	`1`

The only way is not to erase anything.

input	output
`cba`	`0`

input	output
`ab`	`0`

Airplane traffic

Task ID: `airplanes`	Time limit: `0.2 sec`	Memory limit: `256 MB`

In our small flat country there are n airports, numbered 0 through $n-1$. No three airports are colinear. Flight safety regulations require that each flight has to follow a straight line from one airport to another.

The Cookie monster is currently at airport 0 and wants to travel to airport 1. An airplane with a supply of cookies is supposed to travel from airport 2 to airport 3. We must ensure that they cannot meet. Formally, we need to find two separate flight schedules (one for the Cookie monster, one for the cookies) such that the corresponding polylines are completely disjoint.

Input

The first line of input contains either the word "`decide`" or the word "`solve`" (see below).

The second line of input contains the integer n ($4 \leq n$). Each of the following n lines contains two integers x_i, y_i ($0 \leq x_i, y_i \leq 10^9$): the coordinates of airports 0 through $n-1$.

Constraints

There are ten batches of input, each worth 10 points. For each $i \in \{1, \ldots, 5\}$, tests in batch $2i-1$ and batch $2i$ will be the same, except that we used "`decide`" in batch $2i-1$ and "`solve`" in batch $2i$. Maximum n will be 4, 8, 100, 2 000, and 50 000, respectively.

Output

The first line of output should contain either "`YES`" if a valid schedule exists or "`NO`" otherwise. If a schedule exists and the first word in the input is "`solve`", the rest of the output should contain one possible schedule: one line with a sequence of $\leq n-2$ airports visited by the Cookie monster, and one line with a sequence of $\leq n-2$ airports for the cookies.

Sample tests

input

```
decide
4
201 2
104 302
1 99
210 201
```

output

```
NO
```

input

```
solve
5
201 2
104 302
1 99
210 201
410 320
```

output

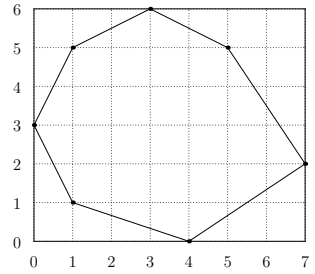
```
YES
0 4 1
2 3
```

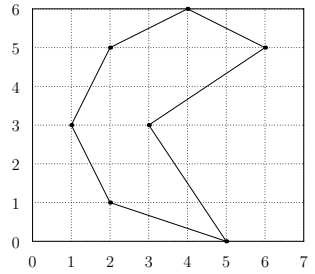
For this test case this is the only possible correct output.

Convex

Task ID: `convex`	Time limit: `0.4 sec`	Memory limit: `256 MB`

A polygon is called *convex* if each of its inner angles is less than or equal to 180°. Equivalently, a polygon P is called convex if each of its diagonals lies completely within P. (A diagonal is a line segment that connects two non-adjacent vertices of P.) For example, in the following figures the left polygon is convex whereas the right polygon is not.





You are given a set A of n points in the plane such that no three points lie on the same line. You want to draw a **convex** polygon such that each vertex of the polygon is a point from A. Your task is to compute the largest number of vertices that convex polygon can have.

Input

The first line of the input contains a single integer $3 \leq n \leq 300$. Each of the following n lines describes a single point: there are two integers $-10^9 \leq x_i, y_i \leq 10^9$ separated by a single space.

You may assume that no point will appear more than once in the input. Furthermore you may assume that no three points are colinear.

Output

Output a single line with a single integer: the largest possible number of vertices our convex polygon can have.

Sample test

input

```
6
-3 3
4 4
-4 -5
7 -5
2 -2
9 2
```

output

```
5
```

The figure on the right shows the 6 given points and one way of drawing a convex polygon with 5 vertices.

Cyclic Shift

Task ID: `cyclic`	Time limit: `0.5 sec`	Memory limit: `256 MB`

The i-th *cyclic shift* of the sequence $A = (a_0, \ldots, a_{n-1})$ is a sequence which is formed by shifting A by i positions to the left, wrapping around. Formally, for $0 \leq i < n$ the i-th cyclic shift is defined as $\mathcal{C}(A, i) = (a_i, \ldots, a_{n-1}, a_0, a_1, \ldots, a_{i-1})$.

For example, for $A = (3, 4, 5, 6)$ there are 4 different cyclic shifts:

$$\begin{aligned}\mathcal{C}(A, 0) &= (3, 4, 5, 6)\\ \mathcal{C}(A, 1) &= (4, 5, 6, 3)\\ \mathcal{C}(A, 2) &= (5, 6, 3, 4)\\ \mathcal{C}(A, 3) &= (6, 3, 4, 5)\end{aligned}$$

Note that for some sequences different cyclic shifts may produce the same outcome. For example if $B = (1, 2, 1, 2)$ then $\mathcal{C}(B, 0) = \mathcal{C}(B, 2) = (1, 2, 1, 2)$ and $\mathcal{C}(B, 1) = \mathcal{C}(B, 3) = (2, 1, 2, 1)$.

The cyclic shifts of a sequence A can be *ordered lexicographically*. The smallest element in this order is called the *lexicographically smallest cyclic shift*.

Formally, the lexicographic order is defined as follows: The sequence $X = (x_0, \ldots, x_{n-1})$ is *lexicographically smaller* than the sequence $Y = (y_0, \ldots, y_{n-1})$ if there exists an index $j \geq 0$ such that $x_j < y_j$ and for all $i < j$ we have $x_i = y_i$. We will denote this $X \prec Y$.

In the examples above we have $\mathcal{C}(A, 0) \prec \mathcal{C}(A, 1) \prec \mathcal{C}(A, 2) \prec \mathcal{C}(A, 3)$ and $\mathcal{C}(B, 0) = \mathcal{C}(B, 2) \prec \mathcal{C}(B, 1) = \mathcal{C}(B, 3)$. For the sequence $C = (3, 1, 3, 7)$ the lexicographic order of its cyclic shift is $\mathcal{C}(C, 1) \prec \mathcal{C}(C, 0) \prec \mathcal{C}(C, 2) \prec \mathcal{C}(C, 3)$. In particular, note that $\mathcal{C}(C, 0) = (3, 1, 3, 7) \prec (3, 7, 3, 1) = \mathcal{C}(C, 2)$.

Thus, the lexicographically smallest cyclic shift of A is the shift by 0, for C it is the shift by 1, and for B there are multiple shift amounts that produce the lexicographically smallest cyclic shift: 0 and 2.

We have a sequence A of n integers. Your task is to determine which of its cyclic shifts is the lexicographically smallest one. However, we will not show you the sequence. To obtain the information you need, you will have to ask questions about the sequence. For each question you choose two indices i and j ($0 \leq i, j < n$). We will then tell you which of the numbers a_i and a_j is the smaller one. (The precise description of the format of the communication is described later.) The fewer questions you ask, the better your solution is, the more points it will receive.

Interaction

Your program should communicate through standard input (stdin) and output (stdout).

As the beginning, your program should read a single integer n ($1 \leq n \leq 100\,000$) from stdin. This integer is the length of the hidden sequence $A = (a_0, a_1, \ldots a_{n-1})$.

To compare the elements a_i and a_j:

1. Write to stdout: i, a single space, j, and a newline. Then, flush stdout (explained below).

2. Read a single integer x from stdin. The value of x will be -1 if $a_i < a_j$, 0 if $a_i = a_j$, and 1 if $a_i > a_j$.

Once you know which of the cyclic shifts is the smallest one, output a single line containing a single integer k ($0 \leq k < n$) such that $\mathcal{C}(A, k)$ is the smallest cyclic shift out of all the cyclic shifts of the hidden sequence A. If there is more than one smallest cyclic shift, output an arbitrary one.

A note about communication: The communication between our grader and your solution will only work if you make sure that your output is not buffered – that is, the grader must see your question as soon as you output it. This can be forced by flushing the standard output.[1]

Example

In the following example the prefix `>` denotes that your program reads the line from stdin, and `<` denotes that your program writes the line to stdout.

```
> 3
< 0 1
> -1
< 2 1
> 1
< 0
```

The grader initially tells you the length n of the sequence A.
Your program compares a_0 and a_1.
The grader responds that a_0 is smaller *than a_1.*
Your program compares a_2 and a_1.
The grader responds that a_2 is greater *than a_1.*
Your program answers – the smallest cyclic shift is $\mathcal{C}(A, 0)$.

Note that from the two questions your program can conclude that $a_0 < a_1 < a_2$. For each such sequence the smallest cyclic shift has to be the shift by 0.

Constraints

There are 5 batches of inputs, each worth 20 points. The maximum n in the batches is 200, 500, 1000, 5000, and 10 000, respectively.

For each batch, each test case will be tested separately, and your score for the batch will be the minimum over all individual test case scores.

If your program fails to solve a test case, its score for that test case is 0. This includes the standard ways (wrong answer, time limit exceeded, runtime error), but you will also fail if you do not follow the communication protocol.

Otherwise, let q be the number of questions your program needs to ask, and let n be the sequence length. Your score for this particular test case is determined by the formula $20\sqrt[4]{\min(1, 3n/q)}$.

Note that any solution that always asks at most $3n$ questions will score 100 points.

[1] If you are using C++ streams, `cout << endl` flushes the output. You can also do an explicit `cout << flush`. In C/C++ you should do a `fflush(stdout)` after each `printf`. In FreePascal do a `flush(output)` after each `writeln`. In Python use `sys.stdout.flush()`. In Java, `System.out` should flush automatically, but it also has `System.out.flush()`.

Dog search

Task ID: dog	Time limit: 0.1 sec	Memory limit: 256 MB

Everybody knows word search puzzles: you are given a grid of letters and a set of words, and the goal is to locate each of the words somewhere in the grid.

Most word search puzzles are easy for a human: you just look for the letter with the fewest occurrences and go from there. Still, word search puzzles can be pretty difficult, as illustrated by the famous "dog search" puzzle shown below on the left.

```
DGOODDODGOODDO
ODOOGGGDODGOGG
OGOGDOODGOODDD
DGDOOOGGOOGDGO
OGDGOGDGOGGOGD
DDDGDDODOOGDOO
ODGOGGDOOGGOOD
```

Can you find the word DOG anywhere in this grid (either horizontally, vertically, or diagonally)? If you cannot, or if it took you way too long, it's probably better to write a program that will do it for you. More precisely, for a given grid your program should count all occurrences of the word DOG.

Input

The first line of the input contains two positive integers r and c ($1 \le r, c \le 400$): the number of rows and the number of columns in the grid. Each of the following r lines contains a string of c letters. Each of the letters will be D, O, or G.

Constraints

There are four batches of input, each worth 25 points:

- In the first batch, each occurrence of DOG is horizontal, and from left to right.
- In the second batch, each occurrence of DOG is horizontal in either direction.
- In the third batch, each occurrence of DOG is horizontal or vertical.
- In the fourth batch, there may be occurrences in all eight directions.

Output

Output a single line with one integer – the total number of DOGs in the given grid.

Sample tests

input

```
1 5
DOGOD
```

output

```
2
```

Two overlapping dogs.

input

```
7 14
(the above grid)
```

output

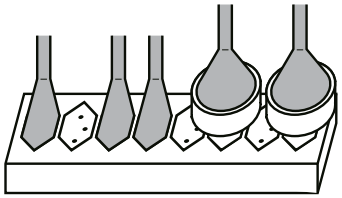
```
1
```

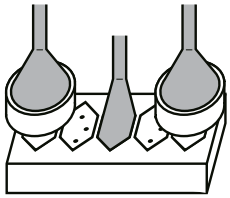
Hint: the dog hidden in the above grid goes downwards and to the right.

Electricity

Task ID: `electricity`	Time limit: `0.2 sec`	Memory limit: `256 MB`

Davos camp participants often face the problem how to connect all their laptops to the power network. They have only one power strip. The power strip has n Swiss sockets in a row. Some participants have European plugs and use adapters. The European plug is wider than the Swiss one, thus if the European plug is plugged in the i-th socket of the strip, the adjacent sockets $i-1$ and $i+1$ cannot be used.





There are s participants who use a laptop with the Swiss plug, and e participants who are using an European plug. Help them to find a way how to plug all their laptops in the power strip if possible.

Input

The first and the only line of the input contains the three integers n, s, and e separated by single spaces ($1 \leq n \leq 1\,000$, $0 \leq s, e \leq 1\,000$).

Output

Output a single line containing a string of n characters. The characters represent the power sockets on the power strip, in order. Character "`S`" represents a socket with a Swiss plug, "`E`" a socket with an European plug, and "`.`" represents an empty socket.

If there is more than one solution, output an arbitrary one. If there is no solution, output the string "`Impossible`" (quotes are for clarity).

Sample tests

input	output
`8 3 2`	`S.SS.E.E`

input	output
`5 1 2`	`E.S.E`

input	output
`4 0 3`	`Impossible`

The first two sample tests correspond to the pictures in the problem statement.

A new farm

Task ID: `farm`	Time limit: `2.0 sec`	Memory limit: `256 MB`

You have found a piece of land for sale. The land is divided into $n \times n$ squares. Each square can be bought separately. You want to buy *exactly* d squares and build a farm. Your farm has to be 4-connected. That is, you have to be able to travel between any two of its d squares without leaving the farm and without walking diagonally.

You have a map of the area. For each square (i, j) the map gives its average elevation $e_{i,j}$. The squares with a smaller elevation are more fertile. Your goal is to avoid having squares that are not fertile. More precisely, you want $\max e_{i,j}$ to be as small as possible. (The maximum is taken over all squares that belong to your farm.)

Input

The first line of input contains the integers n and d: the size of the grid and the desired size of your farm. Each of the next n lines contains n space-separated integers: the elevations $e_{i,j}$ in row major order. You may assume that $1 \le d \le n^2$ and that $1 \le e_{i,j} \le 10^9$ for all i, j.

Constraints

There are five batches of input, each worth 20 points. Maximum values of n for individual batches are 100, 200, 1000, 1000, and 1500. In the first batch you may assume that $d \le 2$. In the third batch you may assume that $d \le n$.

Caution: this is a huge input and output. Either use C/C++ `printf/scanf` or improve the performance of C++ streams by including `std::ios_base::sync_with_stdio(false);` in the beginning of main.

Output

The first line of output should be the smallest possible value of $\max e_{i,j}$ a farm can have. The next n lines of output should contain one possible layout of the optimal farm, using '`.`' for squares you should not buy and '`*`' for squares you should buy.

Sample tests

input

```
5 6
3 4 1 2 4
3 1 2 4 6
6 9 1 1 2
1 7 9 4 3
1 1 1 1 6
```

output

```
2
..**.
.**..
..**.
.....
.....
```

input

```
3 8
10 20 30
40 90 50
60 70 80
```

output

```
80
***
*.*
***
```

Apples and oranges

Task ID: `fruit`	Time limit: `0.3 sec`	Memory limit: `256 MB`

Everyone always says "don't mix apples and oranges", so we did just that. Afterwards, we placed all our apples and oranges into a single row.

We like looking at oranges. We would like to see c consecutive oranges somewhere in our row. If there already are c consecutive oranges, we are happy. Otherwise, we will have to eat some apples in order to create c consecutive oranges somewhere.

Write a program that will compute the smallest number of apples we have to eat.

Input

The first line of input contains two integers n and c: the total number of fruits in our row, and the number of consecutive oranges we want to see.

The second line of input contains a string of n uppercase letters, each of them an `A` (apple) or an `O` (orange). It is guaranteed that there will be at least c `O`s.

Constraints

There are five batches of input, each worth 20 points:

- In the first batch: $1 \leq c \leq n \leq 1000$.
- In the second batch: $1 \leq c \leq n \leq 100,000$ and $c = 2$.
- In the third batch: $1 \leq c \leq n \leq 100,000$ and $c \leq 10$.
- In the fourth batch: $1 \leq c \leq n \leq 100,000$.
- In the fifth batch: $1 \leq c \leq n \leq 1,000,000$.

Output

Output a single line with one integer – the smallest number of apples we need to eat in order to produce c consecutive oranges somewhere in the row.

Sample tests

input

```
14 3
OOAAAOAAAOAOAO
```

output

```
2
```

We get three consecutive oranges by eating the last two apples.

input

```
7 2
AAOOOAA
```

output

```
0
```

We already have two consecutive oranges.

Highways

Task ID: `highways`	Time limit: `3.0 sec`	Memory limit: `256 MB`

In Absurdistan there are n cities (numbered 1 through n). There are m bidirectional roads, each connecting a pair of cities. The country is not necessarily connected by those roads, and all roads are in an awful condition. The Grand Vizier wants you to *rebuild* some of the *existing* old roads into new modern highways. For some obscure reason, his only request is that for each city the number of highways that enter the city must be odd.[2]

Input

The first line of input contains an integer t ($t \leq 100$): the number of test cases that follow.

Each test case starts with a single line with the integers n and m ($1 \leq n$, $0 \leq m$). Then, m lines follow. Each of those lines contains two cities that are connected by an existing road. Each pair of cities is connected by at most one direct edge.

Constraints

There are ten inputs, each worth 10 points. Within an input the total number of cities will never exceed 10^6, and the total number of roads will never exceed $2 \cdot 10^6$. The maximum values of n and m in the individual test cases will gradually grow, from $(\max n, \max m) = (10, 10)$ in input #1 and $(12, 66)$ in input #3, up to $(100\,000, 200\,000)$ in input #10.

Caution: this is a huge input and output. Either use C/C++ `printf/scanf` or improve the performance of C++ streams by including `std::ios_base::sync_with_stdio(false);` in the beginning of main.

Output

The output for each test case is either a single line with the word "`NONE`" (quotes for clarity) if no solution exists, or the description of one valid set of highways. The description starts with a line with a single positive integer k: the number of highways. Then, k lines must follow, each containing the numbers of two cities connected by a highway.

[2] If you really need to know: The neighboring country of Loonia once built highways that connected the entire country in such a way that each city had an even number of highways. However, the famous car racer Eonhard Leuler came to Loonia, found a tour that used each highway once, and drove along the tour in a cycle until all highways fell apart. To spite Leuler, the Grand Vizier came up with the exactly opposite condition.

Sample test

input

```
3
2 1
1 2
3 0
6 5
1 2
1 3
2 3
3 4
5 6
```

output

```
1
1 2
NONE
4
1 3
3 2
3 4
5 6
```

Increasing

Task ID: `increasing`	Time limit: `0.2 sec`	Memory limit: `256 MB`

Let $D_{n,k}$ be a list of all *increasing* sequences of n integers whose elements are between 1 and k. For example:

$$D_{3,4} = (1,2,3),\ (1,2,4),\ (1,3,4),\ (2,3,4)$$
$$D_{2,5} = (1,2),\ (1,3),\ (1,4),\ (1,5),\ (2,3),\ (2,4),\ (2,5),\ (3,4),\ (3,5),\ (4,5)$$

Within each $D_{n,k}$ the individual sequences have a fixed order: the *lexicographic order*. This order is used in both examples above. Formally, the lexicographic order is defined as follows:

The sequence $a_0, \ldots, a_{n-1}$ is *lexicographically smaller* than the sequence $b_0, \ldots, b_{n-1}$ if there exists an index $j \geq 0$ such that $a_j < b_j$ and for all $i < j$ we have $a_i = b_i$.

For example, $(3, 7, 12)$ is lexicographically smaller than $(3, 8, 9)$. Hence, in $D_{3,20}$ the sequence $(3, 7, 12)$ would appear sooner than $(3, 8, 9)$. The sequence $(3, 7, 12)$ is also smaller than $(4, 5, 6)$.

You are given n, k, and a sequence S which is one of the sequences in $D_{n,k}$. Your task is to find the index of S in $D_{n,k}$. (The first sequence in $D_{n,k}$ has index 1.)

For example, in $D_{3,2}$ the index of $(1, 2, 4)$ is 2 and the index of $(2, 3, 4)$ is 4.

Input

The first line contains a single integer n. The second line contains a single integer k ($1 \leq n \leq k \leq 200\,000$). The third line contains n integers separated by a single space. Each integer is between 1 and k and they form an *increasing* sequence S as described above.

Constraints

There are 10 batches of input, each worth 10 points. In the first 3 batches k is at most 100.

Output

Let i be the index of the given sequence S in $D_{n,k}$. Note that this number can be *very large*.

Output a single line containing the value (i mod $1\,000\,000\,007$), that is, the remainder when i is divided by $1\,000\,000\,007$.

Sample test

input	output
5 8 1 2 3 4 8	4

The first four sequences in $D_{5,8}$ *are* $(1, 2, 3, 4, 5)$, $(1, 2, 3, 4, 6)$, $(1, 2, 3, 4, 7)$, *and* **(1,2,3,4,8)**.

Inverting rectangles

Task ID: `invert`	Time limit: `1.5 sec`	Memory limit: `256 MB`

You once had a nice $n \times n$ bitmap with black and white pixels only. Sadly, that was before your nephew opened the image in an editor and changed it completely.

Your nephew made multiple changes to the image, but he always used the same tool: the "invert". This tool works as follows: The user selects a rectangular region of the bitmap. All white pixels in the selected region become black and vice versa.

Luckily, the editor logged all actions your nephew took. Recover the original image.

Input

The first line of input contains the integer n: the size of the bitmap. Both rows and columns of the bitmap are numbered 1 through n, starting in the top left corner. The next n lines describe the current state of the ruined bitmap. More precisely, each of these n lines contains n space-separated 0s and 1s.

The following line of input contains the integer a: the number of actions performed by the nephew. Each of the remaining a lines describes one action, in chronological order, oldest action first. Each action is described by four integers x_1, y_1, x_2, y_2 such that $1 \leq x_1 \leq x_2 \leq n$ and $1 \leq y_1 \leq y_2 \leq n$. These represent the top-left and the bottom-right pixel of the inverted rectangle. (Note that the x coordinate is the column.)

Constraints

There are five batches of input, each worth 20 points. Maximum values of n for individual batches are 100, 200, 500, 1000, and 1000. Maximum values of a for individual batches are 100, 1000, 10^5, 10^5, and 10^6

Caution: this is a huge input and output. Either use C/C++ `printf`/`scanf` or improve the performance of C++ streams by including `std::ios_base::sync_with_stdio(false);` in the beginning of main.

Output

Output n lines, each with n space-separated 0s and 1s: the original image.

Sample test

input

```
4
1 1 0 1
0 0 1 0
1 0 1 1
0 1 0 1
2
3 1 4 3
2 3 3 4
```

output

```
1 1 1 0
0 0 0 1
1 1 1 0
0 0 1 1
```

Starting from the current, damaged state we undo the invert "2 3 3 4" to obtain the bitmap below. Then we undo "3 1 4 3" to obtain the bitmap above.

```
1 1 0 1
0 0 1 0
1 1 0 1
0 0 1 1
```

Mnemonic

Task ID: `mnemonic`	Time limit: `0.4 sec`	Memory limit: `256 MB`

Even though Mouse Plouffe is not as well known as Mouse Gauss or Mouse Euler, he is still a very respectable mathematician. One of his most useful contributions is definitely the On-Line Encyclopedia of Integer Sequences, which you probably found useful when solving programming tasks at home. However, in 1975 Mouse Plouffe became famous over the world for a totally different accomplishment: He was able to recite the first 4096 digits of the mathematical constant π.

When remembering all those digits, one usually uses a mnemonic system. In this task we use the following one: Create a sentence, in which the length of the i-th word is the i-th digit of the number you want to remember. The digit 0 is encoded by a word of length ten and no word in the sentence can be longer than ten. In order to remember the first eight digits of π you can, for example, remember the sentence: "May I have a large container of coffee". The words are of lengths $3, 1, 4, 1, 5, 9, 2$ and 6 respectively.

In this problem you are given a very long sentence. Your task is to reconstruct the number which is encoded by the sentence.

Input

The first line of the input contains a single integer $1 \leq N \leq 100\,000$ – the number of words in the sentence. The following line contains exactly N words, separated by a single space each. Every character of every word is either a lowercase or uppercase letter of English alphabet. The input contains no punctuation characters. You may assume that no word is longer than ten characters and that the first word will always be shorter than ten characters.

Constraints

For testcases, which are worth at least 50% of the points, it holds that $n \leq 100$.

Output

Output a single line with a single integer: the number which is encoded by the input sentence.

Sample tests

input

```
8
May I have a large container of coffee
```

output

```
31415926
```

input

```
7
In Heuried I detected an enormous X
```

output

```
2718281
```

Morpher

Task ID: `morpher`	Time limit: `0.3 sec`	Memory limit: `256 MB`

In this task we are using an alphabet that consists of the first n uppercase letters of the English alphabet. A *morpher* is a function f that is defined on letters of our alphabet. For each letter, f returns a string that consists of at least two letters. An example of a morpher for $n = 3$: $f(\texttt{A}) = \texttt{ABC}$, $f(\texttt{B}) = \texttt{AB}$, $f(\texttt{C}) = \texttt{BC}$.

Once we define a morpher, we can use it to transform strings. The image of a string is obtained by concatenating the images of its letters. For example, when using the morpher defined above, $f(\texttt{BAB}) = f(\texttt{B}) + f(\texttt{A}) + f(\texttt{B}) = \texttt{ABABCAB}$.

As the output uses the same alphabet as the input, it is possible to use the same morpher on a given string several times in a row. For example, $f(f(\texttt{C})) = f(\texttt{BC}) = \texttt{ABBC}$.

You are given a morpher f, a string s, an integer k and an index p. Imagine that we started with the string s and applied the morpher f exactly k times in a row. Return the p-th character (1-based index) of the final string.

Input

The first line of the input contains three integers n, k, and p separated by a single space ($1 \le n \le 26$, $0 \le k \le 10^9$, $1 \le p \le 10^9$). The second line contains the string s. The following n lines describe the morpher. The i-th of these lines contains the string m_i that replaces the i-th letter of English alphabet when applying the morpher.

Each string in the input consists of at most 50 characters. Each of those characters is one of the first n uppercase English letters. The string s has at least 1 character. Each of the strings m_i has at least 2 characters.

Constraints

There are 50 inputs, each worth 2 points. In the first 10 inputs $k \le 1\,000$ and $p \le 1\,000$. In the following 15 inputs $k \le 10^9$, $p \le 1\,000$. For the next 15 inputs $k \le 1\,000$, $p \le 10^9$. The last 10 inputs contain inputs up to the given maximum.

Output

Output a single uppercase English character as described above. If the resulting string has fewer than p characters, output the character "`-`" instead.

Sample tests

input	output
3 1 4 ABC ABC AB BC	A

The morpher transforms `ABC` *to* `ABCABBC`.
The fourth letter of `ABCABBC` *is* `A`.

input	output
3 1 8 ABC ABC AB BC	-

The string `ABCABBC` *is too short.*

Parentheses

Task ID: `parentheses`	Time limit: `0.4 sec`	Memory limit: `256 MB`

A string s of `(` and `)` characters is called *balanced* if it satisfies the following two conditions:

- The number of `(` and `)` in s is equal.
- In every prefix of s the number of `(` is greater than or equal to the number of `)`.

Hence, for instance, `()()(())` is well balanced but `())(()` is not – the prefix `())` does not satisfy the second condition.

Given is a string r consisting of n characters, each of which is either `(` or `)`. Your goal is to select from r the maximum number of characters such that, when read from left to right, they form a balanced string s.

Input

The first line of the input contains a single integer n ($1 \leq n \leq 1\,000\,000$) – the length of r. The following line contains r: a string of n characters $r_1, \ldots, r_n$, each of which is either `(` or `)`.

Constraints

There are five batches of input, each worth 20 points. In the first three batches we guarantee that n is at most 20, 100 and 5000 respectively.

Output

Let s be the longest balanced string that can be selected from r as described above. Then there exist indices $1 \leq j_i < \cdots < j_k \leq n$ such that $s = r_{j_1} \ldots r_{j_k}$ and k is the length of s.

The first line of your output should contain a single integer k – the length of s. The second line should contain the indices $j_1, j_2, \ldots, j_k$ separated by a single space. I.e., the second line contains the positions of brackets in r which if read from left to right form the longest possible balanced string s.

If there is more than one optimal solution, output an arbitrary one. If $k = 0$, the second line should be empty.

Sample tests

input	output
10)())())())	6 2 5 6 7 8 10

The sample output corresponds to the balanced string $s =$ `(())()`. *Note that for this particular balanced string* s *there are multiple ways how to choose its characters from* r. *The one that is given in sample output is illustrated below.*

```
          1
index: 1234567890
    r: )())())())
    s:  (  ())( )
```

Prefixes and suffixes of strings

Task ID: `presuf`	Time limit: `0.85 sec`	Memory limit: `1024 MB`

Given a string w, let $\varphi(w)$ be the number of strings x such that w starts with x and at the same time w ends with x. (I.e., x is both a prefix and a suffix of w.) For example, for $w =$ `ababa` we have $\varphi(w) = 3$. The three valid xs are `a`, `aba`, and `ababa`.

Given a string w of length n such that the letters of w are $\overline{w_0 w_1 \ldots w_{n-1}}$, the string $w[i : j]$ is the string $\overline{w_i w_{i+1} \ldots w_{j-1}}$. In particular, $w[0 : n] = w$.

You are given a string s of length n. Compute the value of the following sum:

$$\sum_{0 \le i < j \le n} \varphi(\, s[i : j]\,)$$

(In words: we compute φ for each contiguous substring of s, and we sum all the results.)

Input

The first line of input contains a single line with a single string s. The string will not be empty and will consist of lowercase English letters only.

Constraints

There are five batches of input, each worth 20 points. The maximum length of s for the individual batches is 50, 1 000, 4 000, 50 000, and 100 000.

Output

The only line of output should contain the value of the sum.

Sample test

input	output
`ababa`	`22`

To understand the sample test, it may be helpful to note that $\varphi($`ababa`$) = 3$, $\varphi($`abab`$) = 2$, $\varphi($`baba`$) = 2$, $\varphi($`aba`$) = 2$, $\varphi($`bab`$) = 2$, and for any other substring x of `ababa` we have $\varphi(x) = 1$.

Reversals

Task ID: `reversals`	Time limit: `N/A`	Memory limit: `N/A`

You are given two sequences S_1 and S_2, each of length n. Each of these sequences contains each number in the range $1, \ldots, n$ exactly once.

Your task is to transform the sequence S_1 into S_2. During the transformation you will make several steps. In each step you will choose 1-based indices $b \leq e$ into the sequence, and reverse the segment formed by elements at indices b through e.

Formally, the reversal with parameters b and e will transform the sequence $S = (a_1, \ldots, a_n)$ into $S' = (a_1, \ldots, a_{b-1}, \boldsymbol{a_e, a_{e-1}, \ldots, a_{b+1}, a_b}, a_{e+1}, \ldots, a_n)$.

Obviously, it is always possible to transform S_1 into S_2. Your program will be graded according to the number or reversals it needs. The fewer reversals the better!

Submission

This is an output only task! All the input files of this task are available to you. You can find them online next to this task description. The file names are in the form `x.00.in`. Each file contains a single input which is of the form described in the Input section below.

For each input file `x.00.in` your program should produce a corresponding output file named `x.00.out`. The format of the output is given in the Output section below.

Once you have prepared all the output files which you want to submit, you have to pack them into a single zip file and submit it. To produce a zip file in the command line, first make sure that you are in the directory which contains the output files. Then run the following command: "`zip zipname.zip *.out`" (where `zipname` can be any file name).

Input

The first line of the input contains a single integer n ($1 \leq n \leq 500$) as described above. Each of the second and third line contains a permutation of 1 through n. Numbers within a line are separated by single spaces. The second line is the sequence S_1 and the third line is the sequence S_2.

Output

The first line of your output should contain a single integer $k \leq 5\,000$ – the number of reversals your solution needs to change S_1 into S_2. Each of the following k lines should contain two integers b_i and e_i separated by a single space ($1 \leq b_i \leq e_i \leq n$). These k lines describe a sequence of reversals that changes the given S_1 into the given S_2. Note that the reversals must be output in the order in which they should be performed.

Sample tests

input

```
5
5 2 1 4 3
4 5 1 3 2
```

output

```
3
2 4
1 2
4 5
```

We start with the sequence $S_1 = (5, 2, 1, 4, 3)$. *In the first step we reverse elements at (1-based) positions 2 through 4, obtaining the sequence* $(5, \mathbf{4, 1, 2}, 3)$. *In the second step we reverse the first two elements, and in the third step we reverse the last two elements. That gives us the desired sequence* $S_2 = (4, 5, 1, 3, 2)$.

Grading

Your score for each test case will be a real number in $[0, 1]$. Your score for the entire task will be the sum of test case scores, multiplied by 100/16. (Thus, the score will be between 0 and 100, inclusive.) You only get a non-zero score for a test case if you submit a valid output file with a correct (not necessarily optimal) solution.

Your score for a test case is determined as follows: For each test case we have a number w such that we are certain that there is no solution with fewer than w reversals. Let y be the number of reversals you used. Then your score will be computed as follows:

$$score \;=\; 2^{1-y/w} \;=\; \frac{2}{2^{y/w}}$$

Note: For some test cases we are not certain whether there is a solution that uses w reversals. Hence, it may be impossible to score 100 points. Our best solution scores about 97.5 points.

Interesting sequence

Task ID: `sequence`	Time limit: `0.1 sec`	Memory limit: `256 MB`

Peter found an interesting sequence of integers in the Online Encyclopedia of Integer Sequences (`http://oeis.org`). The n-th element a_n in this sequence is the maximal integer k such that n^k divides $n!$.

For example, let's compute a_{12}. We have $12! = 479\,001\,600$ and that equals $12^5 \times 1925$. Clearly, 12^5 divides $12!$ but 12^6 does not. Hence, $a_{12} = 5$.

Peter would like to calculate the elements of this sequence. Help him and write a program that generates for the given n the element a_n.

Input

The input contains a single integer n ($2 \leq n \leq 10^9$).

Constraints

There are 50 batches of inputs, each worth 2 points. In the first 5 batches $n \leq 10$, the next 5 batches have $n \leq 20$, in the following 10 batches $n \leq 1\,000$ and the remaining 30 batches contain inputs up to the given maximum.

Output

Output a single integer, the n-th element in the sequence described above.

Sample tests

input	output
12	5

input	output
45	10

$45! = 45^{10} \times 3\,513\,069\,352\,126\,303\,700\,056\,339\,599\,631\,973\,351\,424$

Sum of permutations

Task ID: `sumperm`	Time limit: `0.15 sec`	Memory limit: `256 MB`

You are given an n-digit integer x. Your task is to find the sum of all $n!$ numbers, which are formed by permuting the digits of x.

Note that two different permutations of digits can result in the same number. In such a case you should count the number multiple times – once for each permutation that produced it. Also note that if x contains the digit 0, some permutations will produce numbers with leading zeros. This is allowed.

Input

The only line of the input contains a single integer x ($1 \leq x \leq 10^{18}$). You may assume that x has no leading zeros.

Constraints

There are 25 batches of input, each worth 4 points. In the first 10 batches $1 \leq x \leq 999$ and in the next 15 batches $1 \leq x \leq 999\,999\,999$.

Output

Let s be the sum of all the permutations of digits of x, as described above. Note that this number can be *very large*. Output a single line containing the value (s mod 366 239), that is, the remainder when s is divided by 366 239.

Sample tests

input	output
123	1332

The 6 numbers which are formed by permuting digits 1, 2, and 3 are 123, 132, 213, 231, 312, and 321. Their sum is 1332.

input	output
122	1110

The sum of the 6 permutations is 122 + 122 + 212 + 212 + 221 + 221 = 1110.

input	output
103	888

Here the permuted numbers are 013, 031, 103, 130, 301, and 310. Their sum is 888.

Two ropes

Task ID: `tworopes`	Time limit: `0.2 sec`	Memory limit: `256 MB`

Peter likes topology. In his free time he often experiments with ropes. His friend Bob wanted to make him a new toy, so he took two ropes: a red one and a black one. He wove the two ropes together in a complicated fashion. At the very end, he tied the ends of the red rope together, and then tied the ends of the black rope together, thus forming two rings. Bob then asked Peter to separate the two rings without cutting either of them or untying the ends.

You are given the position of the ropes in 3D space. To help you, Peter already arranged the two ropes into a special configuration: Each rope is currently a simple closed polyline. Additionally, the entire red rope lies in the plane $x = 0$ and the entire black rope lies in the plane $y = 0$. Write a program that will check whether the two ropes can be separated or not.

Input

The first line contains two integers n and m ($3 \leq n, m \leq 50$) – the number of vertices on the red polyline and the black polyline, respectively. Next come the n rows decribing the red rope. Each of these lines contains three integers x_i, y_i, z_i ($-1000 \leq x_i, y_i, z_i \leq 1000$) – coordinates of vertices of the polyline, in order. The following m lines describe the black rope using the same format. Note that the last point of each polyline is also connected to the first point, and that all segments of each polyline are straight line segments. You may assume the following:

- each vertex of the red polyline has $x_i = 0$ and each vertex of the black polyline has $y_i = 0$
- each polyline is simple (i.e., does not overlap/intersect/touch itself)
- the two polylines are disjoint (i.e., have no common points)

Constraints

Your solution will be tested on 10 batches of tests, each worth 10 points. In batch #1, $n = m = 3$. In batch #2, $n = m = 4$. In batch #3, $n = 3$ and $m \leq 50$. In batch #4, $n = 4$ and $m \leq 50$. In batches #5 to #7 each polyline is the boundary of a convex polygon.

Output

Output the string "`YES`" if it is possible to separate the ropes and "`NO`" if it is not.

Sample tests

input

```
4 4
0 1 0
0 1 3
0 -1 3
0 -1 0
1 0 1
1 0 2
-1 0 2
-1 0 1
```

output

```
YES
```

input

```
4 4
0 1 0
0 1 2
0 -1 2
0 -1 0
1 0 1
1 0 3
-1 0 3
-1 0 1
```

output

```
NO
```