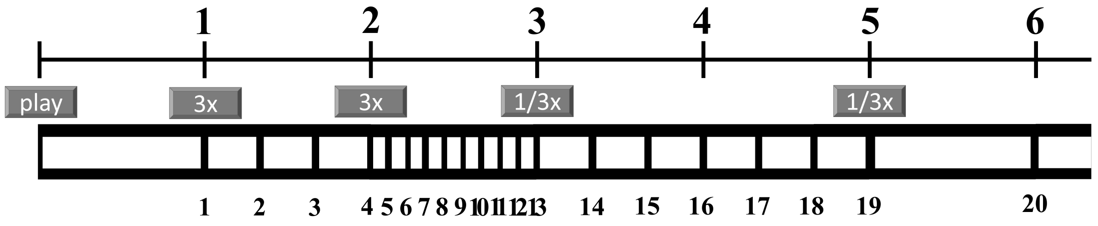
Problem A
Fast Forwarding

Time Limit: 2 seconds

Mr. Anderson frequently rents video tapes of his favorite classic films. Watching the films so many times, he has learned the precise start times of his favorite scenes in all such films. He now wants to find how to wind the tape to watch his favorite scene as quickly as possible on his video player.

When the **[play]** button is pressed, the film starts at the normal playback speed. The video player has two buttons to control the playback speed: The **[3x]** button triples the speed, while the **[1/3x]** button reduces the speed to one third. These speed control buttons, however, do not take effect on the instance they are pressed. Exactly one second after playback starts and every second thereafter, the states of these speed control buttons are checked. If the **[3x]** button is pressed on the timing of the check, the playback speed becomes three times the current speed. If the **[1/3x]** button is pressed, the playback speed becomes one third of the current speed, unless it is already the normal speed.

For instance, assume that his favorite scene starts at 19 seconds from the start of the film. When the **[3x]** button is on at one second and at two seconds after the playback starts, and the **[1/3x]** button is on at three seconds and at five seconds after the start, the desired scene can be watched in the normal speed five seconds after starting the playback, as depicted in the following chart.



Your task is to compute the shortest possible time period after the playback starts until the desired scene starts. The playback of the scene, of course, should be in the normal speed.

Input

The input consists of a single test case of the following format.

t

The given single integer t ($0 \leq t < 2^{50}$) is the start time of the target scene.

Output

Print an integer that is the minimum possible time in seconds before he can start watching the target scene in the normal speed.

Sample Input 1	Sample Output 1
`19`	`5`

Sample Input 2	Sample Output 2
`13`	`5`

Sample Input 3	Sample Output 3
`123456789098765`	`85`

Sample Input 4	Sample Output 4
`51`	`11`

Sample Input 5	Sample Output 5
`0`	`0`

Sample Input 6	Sample Output 6
`3`	`3`

Sample Input 7	Sample Output 7
`4`	`2`

Problem B
Estimating the Flood Risk

Time Limit: 2 seconds

Mr. Boat is the owner of a vast extent of land. As many typhoons have struck Japan this year, he became concerned of flood risk of his estate and he wants to know the average altitude of his land. The land is too vast to measure the altitude at many spots. As no steep slopes are in the estate, he thought that it would be enough to measure the altitudes at only a limited number of sites and then approximate the altitudes of the rest based on them.

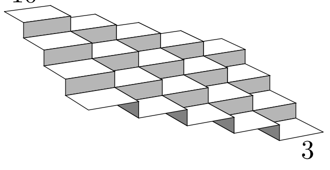
Multiple approximations might be possible based on the same measurement results, in which case he wants to know the worst case, that is, one giving the lowest average altitude.

Mr. Boat's estate, which has a rectangular shape, is divided into grid-aligned rectangular areas of the same size. Altitude measurements have been carried out in some of these areas, and the measurement results are now at hand. The altitudes of the remaining areas are to be approximated on the assumption that altitudes of two adjoining areas sharing an edge differ at most 1.

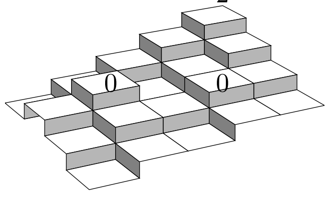
In the first sample given below, the land is divided into 5×4 areas. The altitudes of the areas at $(1, 1)$ and $(5, 4)$ are measured 10 and 3, respectively. In this case, the altitudes of all the areas are uniquely determined on the assumption that altitudes of adjoining areas differ at most 1.

In the second sample, there are multiple possibilities, among which one that gives the lowest average altitude should be considered.

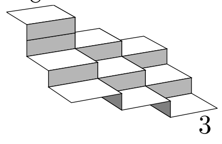
In the third sample, no altitude assignments satisfy the assumption on altitude differences.



Sample 1



Sample 2



Sample 3

Your job is to write a program that approximates the average altitude of his estate. To be precise, the program should compute the total of approximated and measured altitudes of all the mesh-divided areas. If two or more different approximations are possible, the program should compute the total with the severest approximation, that is, one giving the lowest total of the altitudes.

Input

The input consists of a single test case of the following format.

> w d n
> x_1 y_1 z_1
> $\vdots$
> x_n y_n z_n

Here, w, d, and n are integers between 1 and 50, inclusive. w and d are the numbers of areas in the two sides of the land. n is the number of areas where altitudes are measured. The i-th line of the following n lines contains three integers, x_i, y_i, and z_i satisfying $1 \le x_i \le w$, $1 \le y_i \le d$, and $-100 \le z_i \le 100$. They mean that the altitude of the area at (x_i, y_i) was measured to be z_i. At most one measurement result is given for the same area, i.e., for $i \ne j$, $(x_i, y_i) \ne (x_j, y_j)$.

Output

If all the unmeasured areas can be assigned their altitudes without any conflicts with the measured altitudes assuming that two adjoining areas have the altitude difference of at most 1, output an integer that is the *total* of the measured or approximated altitudes of all the areas. If more than one such altitude assignment is possible, output the minimum altitude total among the possible assignments.

If no altitude assignments satisfy the altitude difference assumption, output `No`.

Sample Input 1

```
5 4 2
1 1 10
5 4 3
```

Sample Output 1

```
130
```

Sample Input 2

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

Sample Output 2

```
-14
```

Sample Input 3

```
3 3 2
1 1 8
3 3 3
```

Sample Output 3

```
No
```

Sample Input 4

```
2 2 1
1 1 -100
```

Sample Output 4

```
-404
```

Problem C
Wall Painting
Time Limit: 6 seconds

Here stands a wall made of a number of vertical panels. The panels are not painted yet.

You have a number of robots each of which can paint panels in a single color, either red, green, or blue. Each of the robots, when activated, paints panels between a certain position and another certain position in a certain color. The panels painted and the color to paint them are fixed for each of the robots, but which of them are to be activated and the order of their activation can be arbitrarily decided.

You'd like to have the wall painted to have a high *aesthetic value*. Here, the aesthetic value of the wall is defined simply as the sum of aesthetic values of the panels of the wall, and the aesthetic value of a panel is defined to be:

- 0, if the panel is left unpainted.
- The bonus value specified, if it is painted only in a single color, no matter how many times it is painted.
- The penalty value specified, if it is once painted in a color and then overpainted in one or more different colors.

Input

The input consists of a single test case of the following format.

$$\begin{array}{l} n\ m\ x\ y \\ c_1\ l_1\ r_1 \\ \vdots \\ c_m\ l_m\ r_m \end{array}$$

Here, n is the number of the panels ($1 \leq n \leq 10^9$) and m is the number of robots ($1 \leq m \leq 2 \times 10^5$). x and y are integers between 1 and 10^5, inclusive. x is the bonus value and $-y$ is the penalty value. The panels of the wall are consecutively numbered 1 through n. The i-th robot, when activated, paints all the panels of numbers l_i through r_i ($1 \leq l_i \leq r_i \leq n$) in color with color number c_i ($c_i \in \{1, 2, 3\}$). Color numbers 1, 2, and 3 correspond to red, green, and blue, respectively.

Output

Output a single integer in a line which is the maximum achievable aesthetic value of the wall.

Sample Input 1

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

Sample Output 1

```
70
```

Sample Input 2

```
26 3 9 7
1 11 13
3 1 11
3 18 26
```

Sample Output 2

```
182
```

Sample Input 3

```
21 10 10 5
1 10 21
3 4 16
1 1 7
3 11 21
3 1 16
3 3 3
2 1 17
3 5 18
1 7 11
2 3 14
```

Sample Output 3

```
210
```

Sample Input 4

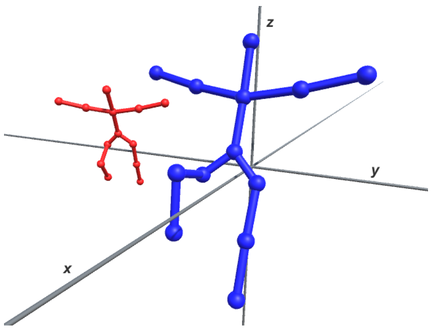
```
21 15 8 7
2 12 21
2 1 2
3 6 13
2 13 17
1 11 19
3 3 5
1 12 13
3 2 2
1 12 15
1 5 17
1 2 3
1 1 9
1 8 12
3 8 9
3 2 9
```

Sample Output 4

```
153
```

Problem D
Twin Trees Bros.
Time Limit: 3 seconds

To meet the demand of ICPC (International Cacao Plantation Consortium), you have to check whether two given *trees* are *twins* or not.



Example of two trees in the three-dimensional space.

The term *tree* in the graph theory means a connected graph where the number of edges is one less than the number of nodes. ICPC, in addition, gives three-dimensional grid points as the locations of the tree nodes. Their definition of two trees being *twins* is that, there exists a geometric transformation function which gives a one-to-one mapping of all the nodes of one tree to the nodes of the other such that for each edge of one tree, there exists an edge in the other tree connecting the corresponding nodes. The geometric transformation should be a combination of the following transformations:

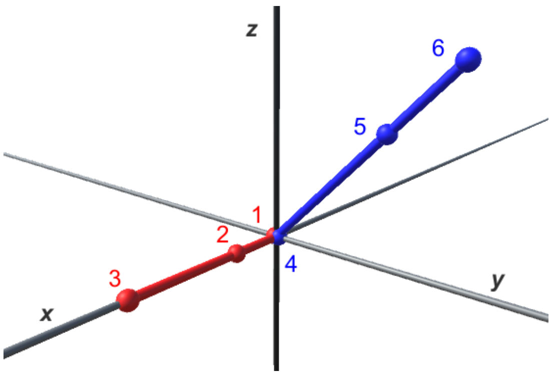
- translations, in which coordinate values are added with some constants,
- uniform scaling with positive scale factors, in which all three coordinate values are multiplied by the same positive constant, and
- rotations of any amounts around either $x-$, $y-$, and $z-$axes.

Note that two trees can be twins in more than one way, that is, with different correspondences of nodes.

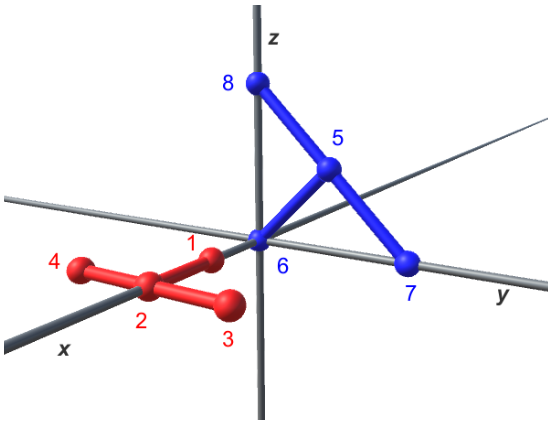
Write a program that decides whether two trees are twins or not and outputs the number of different node correspondences.

Hereinafter, transformations will be described in the right-handed xyz-coordinate system.

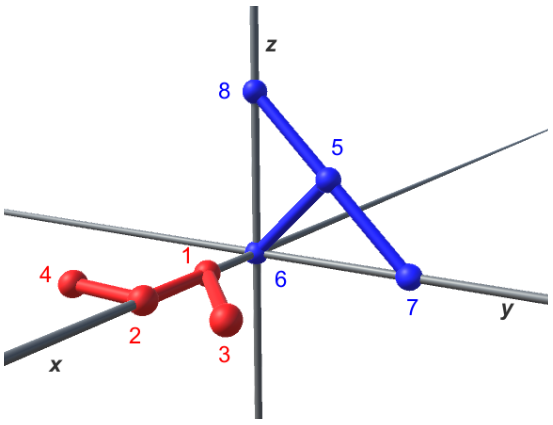
Trees in the sample inputs 1 through 4 are shown in the following figures. The numbers in the figures are the node numbers defined below.



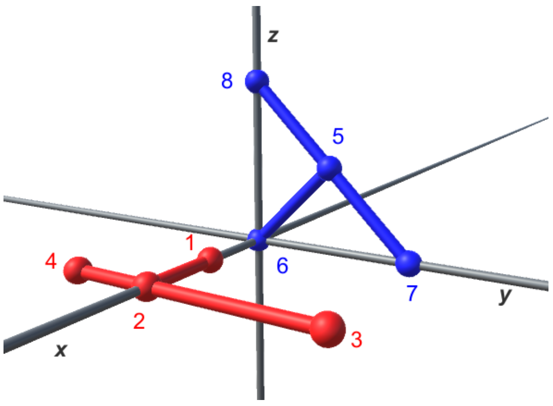
Sample Input 1



Sample Input 2



Sample Input 3



Sample Input 4

For the sample input 1, each node of the red tree is mapped to the corresponding node of the blue tree by the transformation that translates $(-3, 0, 0)$, rotates $-\pi/2$ around the z-axis, rotates $\pi/4$ around the x-axis, and finally scales by $\sqrt{2}$. By this mapping, nodes #1, #2, and #3 of the red tree at $(0, 0, 0)$, $(1, 0, 0)$, and $(3, 0, 0)$ correspond to nodes #6, #5, and #4 of the blue tree at $(0, 3, 3)$, $(0, 2, 2)$, and $(0, 0, 0)$, respectively. This is the only possible correspondence of the twin trees.

For the sample input 2, red nodes #1, #2, #3, and #4 can be mapped to blue nodes #6, #5, #7, and #8. Another node correspondence exists that maps nodes #1, #2, #3, and #4 to #6, #5, #8, and #7.

For the sample input 3, the two trees are *not* twins. There exist transformations that map nodes of one tree to distinct nodes of the other, but the edge connections do not agree.

For the sample input 4, there is no transformation that maps nodes of one tree to those of the other.

Input

The input consists of a single test case of the following format.

$$\begin{array}{l} n \\ x_1\ y_1\ z_1 \\ \vdots \\ x_n\ y_n\ z_n \\ u_1\ v_1 \\ \vdots \\ u_{n-1}\ v_{n-1} \\ x_{n+1}\ y_{n+1}\ z_{n+1} \\ \vdots \\ x_{2n}\ y_{2n}\ z_{2n} \\ u_n\ v_n \\ \vdots \\ u_{2n-2}\ v_{2n-2} \end{array}$$

The input describes two trees. The first line contains an integer n representing the number of nodes of each tree ($3 \leq n \leq 200$). Descriptions of two trees follow.

Description of a tree consists of n lines that give the vertex positions and $n-1$ lines that show the connection relation of the vertices.

Nodes are numbered 1 through n for the first tree, and $n+1$ through $2n$ for the second tree.

The triplet (x_i, y_i, z_i) gives the coordinates of the node numbered i. x_i, y_i, and z_i are integers in the range between -1000 and 1000, inclusive. Nodes of a single tree have distinct coordinates.

The pair of integers (u_j, v_j) means that an edge exists between nodes numbered u_j and v_j ($u_j \neq v_j$). $1 \leq u_j \leq n$ and $1 \leq v_j \leq n$ hold for $1 \leq j \leq n-1$, and $n+1 \leq u_j \leq 2n$ and $n+1 \leq v_j \leq 2n$ hold for $n \leq j \leq 2n-2$.

Output

Output the number of different node correspondences if two trees are twins. Output a zero, otherwise.

Sample Input 1

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

Sample Output 1

```
1
```

Sample Input 2

```
4
1 0 0
2 0 0
2 1 0
2 -1 0
1 2
2 3
2 4
0 1 1
0 0 0
0 2 0
0 0 2
5 6
5 7
5 8
```

Sample Output 2

```
2
```

Sample Input 3

```
4
1 0 0
2 0 0
2 1 0
2 -1 0
1 2
1 3
2 4
0 1 1
0 0 0
0 2 0
0 0 2
5 6
5 7
5 8
```

Sample Output 3

```
0
```

Sample Input 4

```
4
1 0 0
2 0 0
2 2 0
2 -1 0
1 2
2 3
2 4
0 1 1
0 0 0
0 2 0
0 0 2
5 6
5 7
5 8
```

Sample Output 4

```
0
```

Sample Input 5

```
3
0 0 0
0 0 1
0 0 2
1 2
1 3
10 4 6
0 0 0
5 2 3
4 5
5 6
```

Sample Output 5

```
1
```

Sample Input 6

```
4
0 0 0
1 3 3
-1 5 5
-10 2 2
1 2
1 3
1 4
1 1 6
0 0 0
-1 -1 10
-10 -10 4
5 6
6 7
6 8
```

Sample Output 6

```
1
```

Sample Input 7

```
7
0 0 0
1 0 0
-1 0 0
0 1 0
0 -1 0
0 0 1
0 0 -1
1 2
1 3
1 4
1 5
1 6
1 7
0 0 0
2 0 0
-2 0 0
0 2 0
0 -2 0
0 0 2
0 0 -2
8 9
8 10
8 11
8 12
8 13
8 14
```

Sample Output 7

```
24
```

Problem E
Reordering the Documents
Time Limit: 4 seconds

Susan is good at arranging her dining table for convenience, but not her office desk.

Susan has just finished the paperwork on a set of documents, which are still piled on her desk. They have serial numbers and *were* stacked in order when her boss brought them in. The ordering, however, is not perfect now, as she has been too lazy to put the documents slid out of the pile back to their proper positions. Hearing that she has finished, the boss wants her to return the documents immediately in the document box he is sending her. The documents should be stowed in the box, of course, in the order of their serial numbers.

The desk has room just enough for two more document piles where Susan plans to make two temporary piles. All the documents in the current pile are to be moved one by one from the top to either of the two temporary piles. As making these piles too tall in haste would make them tumble, not too many documents should be placed on them. After moving *all* the documents to the temporary piles and receiving the document box, documents in the two piles will be moved from their tops, one by one, into the box. Documents should be in reverse order of their serial numbers in the two piles to allow moving them to the box in order.

For example, assume that the pile has six documents #1, #3, #4, #2, #6, and #5, in this order from the top, and that the temporary piles can have no more than three documents. Then, she can form two temporary piles, one with documents #6, #4, and #3, from the top, and the other with #5, #2, and #1 (Figure E.1). Both of the temporary piles are reversely ordered. Then, comparing the serial numbers of documents on top of the two temporary piles, one with the larger number (#6, in this case) is to be removed and stowed into the document box first. Repeating this, all the documents will be perfectly ordered in the document box.

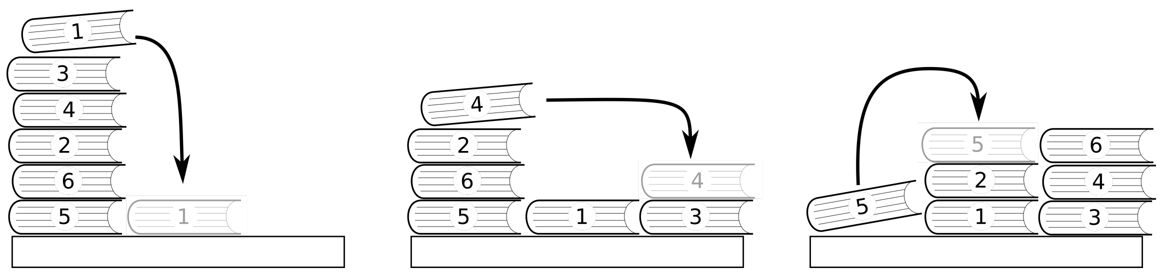


Figure E.1. Making two temporary piles

Susan is wondering whether the plan is actually feasible with the documents in the current pile and, if so, how many different ways of stacking them to two temporary piles would do. You are asked to help Susan by writing a program to compute the number of different ways, which should be zero if the plan is not feasible.

As each of the documents in the pile can be moved to either of the two temporary piles, for n documents, there are 2^n different choice combinations in total, but some of them may disturb the reverse order of the temporary piles and are thus inappropriate.

The example described above corresponds to the first case of the sample input. In this case, the last two documents, #5 and #6, can be swapped their destinations. Also, exchanging the roles of two temporary piles totally will be OK. As any other move sequences would make one of the piles higher than three and/or make them out of order, the total number of different ways of stacking documents to temporary piles in this example is $2 \times 2 = 4$.

Input

The input consists of a single test case of the following format.

n m
$s_1 \dots s_n$

Here, n is the number of documents in the pile ($1 \le n \le 5000$), and m is the number of documents that can be stacked in one temporary pile without committing risks of making it tumble down ($n/2 \le m \le n$). Numbers s_1 through s_n are the serial numbers of the documents in the document pile, from its top to its bottom. It is guaranteed that all the numbers 1 through n appear exactly once.

Output

Output a single integer in a line which is the number of ways to form two temporary piles suited for the objective. When no choice will do, the number of ways is 0, of course.

If the number of possible ways is greater than or equal to $10^9 + 7$, output the number of ways modulo $10^9 + 7$.

Sample Input 1	Sample Output 1
6 3 1 3 4 2 6 5	4

Sample Input 2	Sample Output 2
6 6 1 3 4 2 6 5	8

Sample Input 3	Sample Output 3
4 4 4 3 1 2	0

Problem F
Halting Problem

Time Limit: 3 seconds

A unique law is enforced in the Republic of Finite Loop. Under the law, programs that never halt are regarded as viruses. Releasing such a program is a cybercrime. So, you want to make sure that your software products always halt under their normal use.

It is widely known that there exists no algorithm that can determine whether an arbitrary given program halts or not for a given arbitrary input. Fortunately, your products are based on a simple computation model given below. So, you can write a program that can tell whether a given program based on the model will eventually halt for a given input.

The computation model for the products has only one variable x and $N + 1$ states, numbered 1 through $N + 1$. The variable x can store any integer value. The state $N + 1$ means that the program has terminated. For each integer i ($1 \leq i \leq N$), the behavior of the program in the state i is described by five integers a_i, b_i, c_i, d_i and e_i (c_i and e_i are indices of states).

On start of a program, its state is initialized to 1, and the value of x is initialized by x_0, the input to the program. When the program is in the state i ($1 \leq i \leq N$), either of the following takes place in one execution step:

- if x is equal to a_i, the value of x changes to $x + b_i$ and the program state becomes c_i;
- otherwise, the value of x changes to $x + d_i$ and the program state becomes e_i.

The program terminates when the program state becomes $N + 1$.

Your task is to write a program to determine whether a given program eventually halts or not for a given input, and, if it halts, to compute how many steps are executed. The initialization is not counted as a step.

Input

The input consists of a single test case of the following format.

$$\begin{array}{l} N\ x_0 \\ a_1\ b_1\ c_1\ d_1\ e_1 \\ \vdots \\ a_N\ b_N\ c_N\ d_N\ e_N \end{array}$$

The first line contains two integers N ($1 \le N \le 10^5$) and x_0 ($-10^{13} \le x_0 \le 10^{13}$). The number of the states of the program is $N+1$. x_0 is the initial value of the variable x. Each of the next N lines contains five integers a_i, b_i, c_i, d_i and e_i that determine the behavior of the program when it is in the state i. a_i, b_i and d_i are integers between -10^{13} and 10^{13}, inclusive. c_i and e_i are integers between 1 and $N+1$, inclusive.

Output

If the given program eventually halts with the given input, output a single integer in a line which is the number of steps executed until the program terminates. Since the number may be very large, output the number modulo $10^9 + 7$.

Output -1 if the program will never halt.

Sample Input 1

```
2 0
5 1 2 1 1
10 1 3 2 2
```

Sample Output 1

```
9
```

Sample Input 2

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

Sample Output 2

```
-1
```

Sample Input 3

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

Sample Output 3

```
-1
```

Sample Input 4

```
13 3
15 -10 4 4 2
19 0 4 4 3
23 -20 4 4 1
6 1 2 1 5
6 17 3 -1 6
3 2 9 4 7
7 0 9 4 8
11 -5 9 4 6
6 1 7 1 10
6 5 8 -8 11
1 2 11 1 12
6 1 14 2 13
6 -4 11 1 12
```

Sample Output 4

```
26
```

Problem G
Ambiguous Encoding
Time Limit: 2 seconds

A friend of yours is designing an encoding scheme of a set of characters into a set of variable length bit sequences. You are asked to check whether the encoding is ambiguous or not. In an encoding scheme, characters are given distinct bit sequences of possibly different lengths as their codes. A character sequence is encoded into a bit sequence which is the concatenation of the codes of the characters in the string in the order of their appearances. An encoding scheme is said to be ambiguous if there exist two different character sequences encoded into exactly the same bit sequence. Such a bit sequence is called an "ambiguous binary sequence".

For example, encoding characters "`A`", "`B`", and "`C`" to `0`, `01` and `10`, respectively, is ambiguous. This scheme encodes two different character strings "`AC`" and "`BA`" into the same bit sequence `010`.

Input

The input consists of a single test case of the following format.

$$\begin{array}{l} n \\ w_1 \\ \vdots \\ w_n \end{array}$$

Here, n is the size of the set of characters to encode ($1 \le n \le 1000$). The i-th line of the following n lines, w_i, gives the bit sequence for the i-th character as a non-empty sequence of at most 16 binary digits, 0 or 1. Note that different characters are given different codes, that is, $w_i \ne w_j$ for $i \ne j$.

Output

If the given encoding is ambiguous, print in a line the number of bits in the shortest ambiguous binary sequence. Output zero, otherwise.

Sample Input 1	Sample Output 1
3 0 01 10	3

Sample Input 2

```
3
00
01
1
```

Sample Output 2

```
0
```

Sample Input 3

```
3
00
10
1
```

Sample Output 3

```
0
```

Sample Input 4

```
10
1001
1011
01000
00011
01011
1010
00100
10011
11110
0110
```

Sample Output 4

```
13
```

Sample Input 5

```
3
1101
1
10
```

Sample Output 5

```
4
```

Problem H
Parentheses Editor
Time Limit: 2 seconds

You are working with a strange text editor for texts consisting only of open and close parentheses. The editor accepts the following three keys as editing commands to modify the text kept in it.

- '`(`' appends an open parenthesis ('`(`') to the end of the text.
- '`)`' appends a close parenthesis ('`)`') to the end of the text.
- '`-`' removes the last character of the text.

A balanced string is one of the following.

- "`()`"
- "`(`X`)`" where X is a balanced string
- "XY" where both X and Y are balanced strings

Initially, the editor keeps an empty text. You are interested in the number of balanced substrings in the text kept in the editor after each of your key command inputs. Note that, for the same balanced substring occurring twice or more, their occurrences should be counted separately. Also note that, when some balanced substrings are inside a balanced substring, both the inner and outer balanced substrings should be counted.

Input

The input consists of a single test case given in a line containing a number of characters, each of which is a command key to the editor, that is, either '`(`', '`)`', or '`-`'. The number of characters does not exceed 200 000. They represent a key input sequence to the editor.

It is guaranteed that no '`-`' command comes when the text is empty.

Output

Print the numbers of balanced substrings in the text kept in the editor after each of the key command inputs are applied, each in one line. Thus, the number of output lines should be the same as the number of characters in the input line.

Sample Input 1

```
(()())---)
```

Sample Output 1

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

Sample Input 2

```
()--()()----)(()()))
```

Sample Output 2

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

Problem I
One-Way Conveyors
Time Limit: 2 seconds

You are working at a factory manufacturing many different products. Products have to be processed on a number of different machine tools. Machine shops with these machines are connected with conveyor lines to exchange unfinished products. Each unfinished product is transferred from a machine shop to another through one or more of these conveyors.

As the orders of the processes required are not the same for different types of products, the conveyor lines are currently operated in two-way. This may induce inefficiency as conveyors have to be completely emptied before switching their directions. *Kaizen* (efficiency improvements) may be found here!

Adding more conveyors is too costly. If all the required transfers are possible with currently installed conveyors operating in fixed directions, no additional costs are required. All the required transfers, from which machine shop to which, are listed at hand. You want to know whether all the required transfers can be enabled with all the conveyors operated in one-way, and if yes, directions of the conveyor lines enabling it.

Input

The input consists of a single test case of the following format.

$$\begin{array}{l} n\ m \\ x_1\ y_1 \\ \vdots \\ x_m\ y_m \\ k \\ a_1\ b_1 \\ \vdots \\ a_k\ b_k \end{array}$$

The first line contains two integers n ($2 \le n \le 10\,000$) and m ($1 \le m \le 100\,000$), the number of machine shops and the number of conveyors, respectively. Machine shops are numbered 1 through n. Each of the following m lines contains two integers x_i and y_i ($1 \le x_i < y_i \le n$), meaning that the i-th conveyor connects machine shops x_i and y_i. At most one conveyor is installed between any two machine shops. It is guaranteed that any two machine shops are connected through one or more conveyors. The next line contains an integer k ($1 \le k \le 100\,000$), which indicates the number of required transfers from a machine shop to another. Each of the following k lines contains two integers a_i and b_i ($1 \le a_i \le n$, $1 \le b_i \le n$, $a_i \ne b_i$), meaning that transfer from the machine shop a_i to the machine shop b_i is required. Either $a_i \ne a_j$ or $b_i \ne b_j$ holds for $i \ne j$.

Output

Output "`No`" if it is impossible to enable all the required transfers when all the conveyors are operated in one-way. Otherwise, output "`Yes`" in a line first, followed by m lines each of which describes the directions of the conveyors. All the required transfers should be possible with the conveyor lines operated in these directions. Each direction should be described as a pair of the machine shop numbers separated by a space, with the start shop number on the left and the end shop number on the right. The order of these m lines do not matter as far as all the conveyors are specified without duplicates or omissions. If there are multiple feasible direction assignments, whichever is fine.

Sample Input 1

```
3 2
1 2
2 3
3
1 2
1 3
2 3
```

Sample Output 1

```
Yes
1 2
2 3
```

Sample Input 2

```
3 2
1 2
2 3
3
1 2
1 3
3 2
```

Sample Output 2

```
No
```

Sample Input 3

```
4 4
1 2
1 3
1 4
2 3
7
1 2
1 3
1 4
2 1
2 3
3 1
3 2
```

Sample Output 3

```
Yes
1 2
2 3
3 1
1 4
```

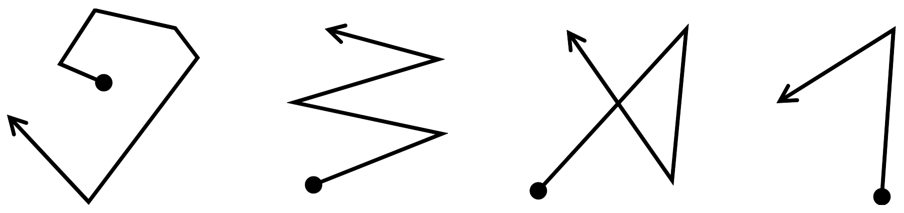
Problem J
Fun Region

Time Limit: 2 seconds

Dr. Ciel lives in a planar island with a polygonal coastline. She loves strolling on the island along spiral paths. Here, a path is called *spiral* if both of the following are satisfied.

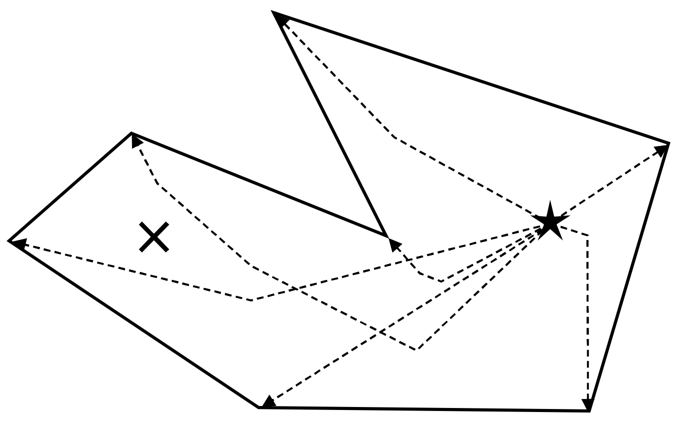
- The path is a simple planar polyline with no self-intersections.
- At all of its vertices, the line segment directions turn clockwise.

Four paths are depicted below. Circle markers represent the departure points, and arrow heads represent the destinations of paths. Among the paths, only the leftmost is spiral.



Dr. Ciel finds a point *fun* if, for all of the vertices of the island's coastline, there exists a spiral path that starts from the point, ends at the vertex, and does not cross the coastline. Here, the spiral path may touch or overlap the coastline.

In the following figure, the outer polygon represents a coastline. The point ⋆ is fun, while the point × is not fun. Dotted polylines starting from ⋆ are valid spiral paths that end at coastline vertices.



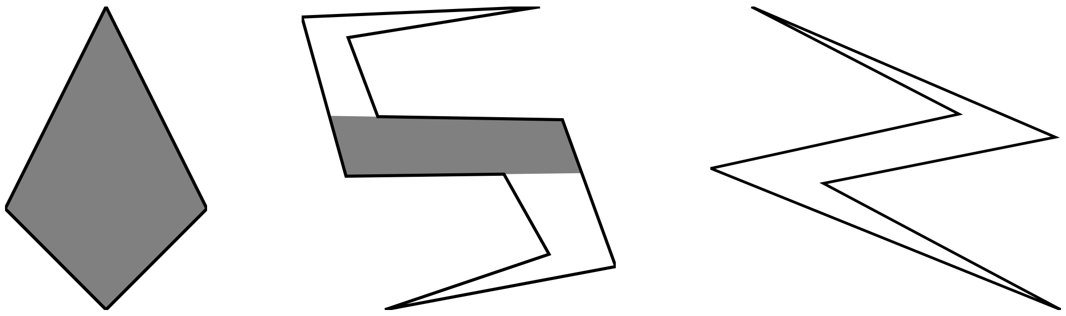


Figure J.1. Samples 1, 2, and 3.

We can prove that the set of all the fun points forms a (possibly empty) connected region, which we call the *fun region.* Given the coastline, your task is to write a program that computes the area size of the fun region.

Figure J.1 visualizes the three samples given below. The outer polygons correspond to the island's coastlines. The fun regions are shown as the gray areas.

Input

The input consists of a single test case of the following format.

$$\begin{array}{l} n \\ x_1\ y_1 \\ \vdots \\ x_n\ y_n \end{array}$$

n is the number of vertices of the polygon that represents the coastline ($3 \leq n \leq 2000$). Each of the following n lines has two integers x_i and y_i, which give the coordinates (x_i, y_i) of the i-th vertex of the polygon, in counterclockwise order. x_i and y_i are between 0 and 10 000, inclusive. Here, the x-axis of the coordinate system directs right and the y-axis directs up. The polygon is simple, that is, it does not intersect nor touch itself. Note that the polygon may be non-convex. It is guaranteed that the inner angle of each vertex is not exactly 180 degrees.

Output

Output the area of the fun region in one line. Absolute or relative errors less than 10^{-7} are permissible.

Sample Input 1	**Sample Output 1**

4
10 0
20 10
10 30
0 10
	300.00

Sample Input 2

```
10
145 269
299 271
343 193
183 139
408 181
356 324
176 327
147 404
334 434
102 424
```

Sample Output 2

```
12658.3130191
```

Sample Input 3

```
6
144 401
297 322
114 282
372 178
197 271
368 305
```

Sample Output 3

```
0.0
```

Problem K
Draw in Straight Lines

Time Limit: 3 seconds

You plan to draw a black-and-white painting on a rectangular canvas. The painting will be a grid array of pixels, either black or white. You can paint black or white lines or dots on the initially white canvas.

You can apply a sequence of the following two operations in any order.

- Painting pixels on a horizontal or vertical line segment, single pixel wide and two or more pixel long, either black or white. This operation has a cost proportional to the length (the number of pixels) of the line segment multiplied by a specified coefficient in addition to a specified constant cost.
- Painting a single pixel, either black or white. This operation has a specified constant cost.

You can overpaint already painted pixels as long as the following conditions are satisfied.

- The pixel has been painted at most once before. Overpainting a pixel too many times results in too thick layers of inks, making the picture look ugly. Note that painting a pixel with the same color is also counted as overpainting. For instance, if you have painted a pixel with black twice, you can paint it neither black nor white anymore.
- The pixel once painted white should not be overpainted with the black ink. As the white ink takes very long to dry, overpainting the same pixel black would make the pixel gray, rather than black. The reverse, that is, painting white over a pixel already painted black, has no problem.

Your task is to compute the minimum total cost to draw the specified image.

Input

The input consists of a single test case. The first line contains five integers n, m, a, b, and c, where n ($1 \leq n \leq 40$) and m ($1 \leq m \leq 40$) are the height and the width of the canvas in the number of pixels, and a ($0 \leq a \leq 40$), b ($0 \leq b \leq 40$), and c ($0 \leq c \leq 40$) are constants defining painting costs as follows. Painting a line segment of length ℓ costs $a\ell + b$ and painting a single pixel costs c. These three constants satisfy $c \leq a + b$.

The next n lines show the black-and-white image you want to draw. Each of the lines contains a string of length m. The j-th character of the i-th string is '#' if the color of the pixel in the i-th row and the j-th column is to be black, and it is '.' if the color is to be white.

Output

Output the minimum cost.

Sample Input 1

```
3 3 1 2 3
.#.
###
.#.
```

Sample Output 1

```
10
```

Sample Input 2

```
2 7 0 1 1
###.###
###.###
```

Sample Output 2

```
3
```

Sample Input 3

```
5 5 1 4 4
..#..
..#..
##.##
..#..
..#..
```

Sample Output 3

```
24
```

Sample Input 4

```
7 24 1 10 10
###...###..#####....###.
.#...#...#.#....#..#...#
.#..#......#....#.#.....
.#..#......#####..#.....
.#..#......#......#.....
.#...#...#.#.......#...#
###...###..#........###.
```

Sample Output 4

```
256
```