Problem A. Azulejos

Input file:	`standard input`
Output file:	`standard output`
Time limit:	10 seconds
Memory limit:	256 megabytes

Azulejo in the cathedral of Porto.
Source: Wikimedia Commons

Ceramic artists Maria and João are opening a small *azulejo* store in Porto. *Azulejos* are the beautiful ceramic tiles for which Portugal is famous. Maria and João want to create an attractive window display, but, due to limited space in their shop, they must arrange their tile samples in two rows on a single shelf. Each of João's tiles has exactly one of Maria's tiles in front of it and each of Maria's tiles has exactly one of João's tiles behind it. These hand-crafted tiles are of many different sizes, and it is important that each tile in the back row is taller than the tile in front of it so that both are visible to passers-by. For the convenience of shoppers, tiles in each row are arranged in non-decreasing order of price from left to right. Tiles of the same price may be arranged in any order subject to the visibility condition stated above.

Input

The first line of input contains an integer n ($1 \le n \le 5 \cdot 10^5$), the number of tiles in each row. The next four lines contain n integers each; the first pair of lines represents the back row of tiles and the second pair of lines represents the front row. Tiles in each row are numbered from 1 to n according to their ordering in the input. The first line in each pair contains n integers $p_1, \ldots, p_n$ ($1 \le p_i \le 10^9$ for each i), where p_i is the price of tile number i in that row. The second line in each pair contains n integers $h_1, \ldots, h_n$ ($1 \le h_i \le 10^9$ for each i), where h_i is the height of tile number i in that row.

Output

If there is a valid ordering, output it as two lines of n integers, each consisting of a permutation of the tile numbers from 1 to n. The first line represents the ordering of the tiles in the back row and the second represents the ordering of the tiles in the front row. If more than one pair of permutations satisfies the constraints, any such pair will be accepted. If no ordering exists, output `impossible`.

Examples

standard input	standard output
4	3 2 4 1
3 2 1 2	4 2 1 3
2 3 4 3	
2 1 2 1	
2 2 1 3	
2	impossible
1 2	
2 3	
2 8	
2 1	

Problem B. Beautiful Bridges

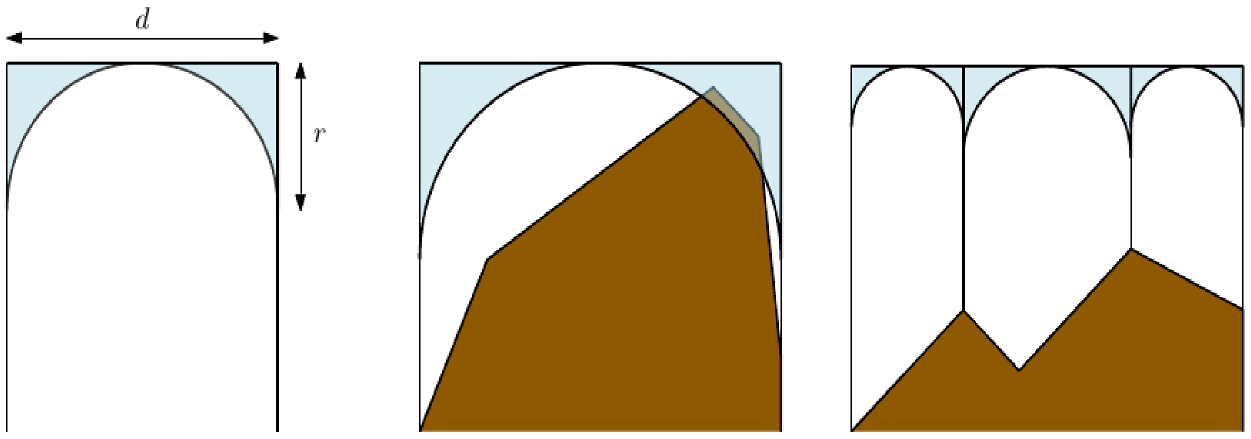
Input file:	`standard input`
Output file:	`standard output`
Time limit:	10 seconds
Memory limit:	256 megabytes

Example of a Roman arch bridge. Source: Wikimedia Commons

What connects us all? Well, it is often bridges. Since ancient times, people have been building bridges for roads, for trains, for pedestrians, and as aqueducts to transport water. It is humanity's way of not taking inconvenient geography for an answer.

The company Arch Bridges Construction (ABC) specializes in—you guessed it—the construction of arch bridges. This classical style of bridge is supported by pillars that extend from the ground below the bridge. Arches between pillars distribute the bridge's weight onto the adjacent pillars.

The bridges built by ABC often have pillars spaced at irregular intervals. For aesthetic reasons, ABC's bridges always have semicircular arches, as illustrated in Figure B.1. However, while a bridge arch can touch the ground, it cannot extend below the ground. This makes some pillar placements impossible.



(a) Consecutive pillars at distance d are connected by a semicircular arch with radius $r = d/2$. (b) An invalid pillar placement (arches cannot extend below ground). (c) A bridge corresponding to Sample Input 1.

Figure B.1: Bridge examples.

Given a ground profile and a desired bridge height h, there are usually many ways of building an arch bridge. We model the ground profile as a piecewise-linear function described by n key points $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$, where the x-coordinate of a point is the position along the bridge, and the y-coordinate is the elevation of the ground above sea level at this position along the bridge. The first and last pillars must be built at the first and last key points, and any intermediate pillars can be built only at these key points. The cost of a bridge is the cost of its pillars (which is proportional to their heights) plus the cost of its arches (which is proportional to the amount of material used). So a bridge with k pillars of heights $h_1, \ldots, h_k$ that are separated by horizontal distances $d_1, \ldots, d_{k-1}$ has a total cost of

$$\alpha \cdot \sum_{i=1}^{k} h_i + \beta \cdot \sum_{i=1}^{k-1} d_i^2$$

for some given constants α and β. ABC wants to construct each bridge at the lowest possible cost.

Input

The first line of input contains four integers n, h, α, and β, where n ($2 \leq n \leq 10^4$) is the number of points describing the ground profile, h ($1 \leq h \leq 10^5$) is the desired height of the bridge above sea level, and $\alpha, \beta (1 \leq \alpha, \beta \leq 10^4)$ are the cost factors as described earlier. Then follow n lines, the i^{th} of which contains two integers x_i, y_i ($0 \leq x_1 < x_2 < \ldots < x_n \leq 10^5$ and $0 \leq y_i < h$), describing the ground profile.

Output

Output the minimum cost of building a bridge from horizontal position x_1 to x_n at height h above sea level. If it is impossible to build any such bridge, output `impossible`.

Examples

standard input	standard output

5 60 18 2
0 0
20 20
30 10
50 30
70 20
	6460

4 10 1 1
0 0
1 9
9 9
10 0
	impossible

Problem C. Checks Post Facto

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

Your university's board game club just hosted a Checkers tournament, and you were assigned to take notes on the games. Unfortunately, while walking home, you dropped all of your papers into a puddle! Disaster! Much of what you wrote is now unreadable; all you have left are some lists of moves played in the middle of various games. Is there some way you can reconstruct what happened in those games? You had better fix things fast, or they will demote you to Tic-Tac-Toe recordkeeper!

Checkers (or English Draughts) is a well-known board game with simple rules. It is played on the dark squares of an 8×8 checkerboard. There are two players, Black and White, who alternate turns moving their pieces (all of Black's pieces are black and all of White's pieces are white). Each piece occupies a single dark square, and can be either a normal *man* or a promoted *king*. A turn consists of choosing one piece and moving it in one of two ways:

1. Shifting it diagonally to an unoccupied adjacent dark square, as shown in Figure C.1(a). This is called a *simple move*. If the piece is a man, it can move only in the two diagonal directions towards the opposing side of the board (towards the bottom for Black, the top for White). If the piece is a king, it can move in all four diagonal directions.

2. Jumping over an adjacent enemy piece to an unoccupied square immediately on the other side, then removing (*capturing*) that piece. Men can jump only in the two directions described above, while kings can jump in all four. The player can then repeat this step, continuing to jump with the same piece as long as there are properly-positioned enemy pieces to capture. Such a sequence of one or more jumps is called a jump move. Figure C.1(b) shows a *jump move* comprising three jumps.

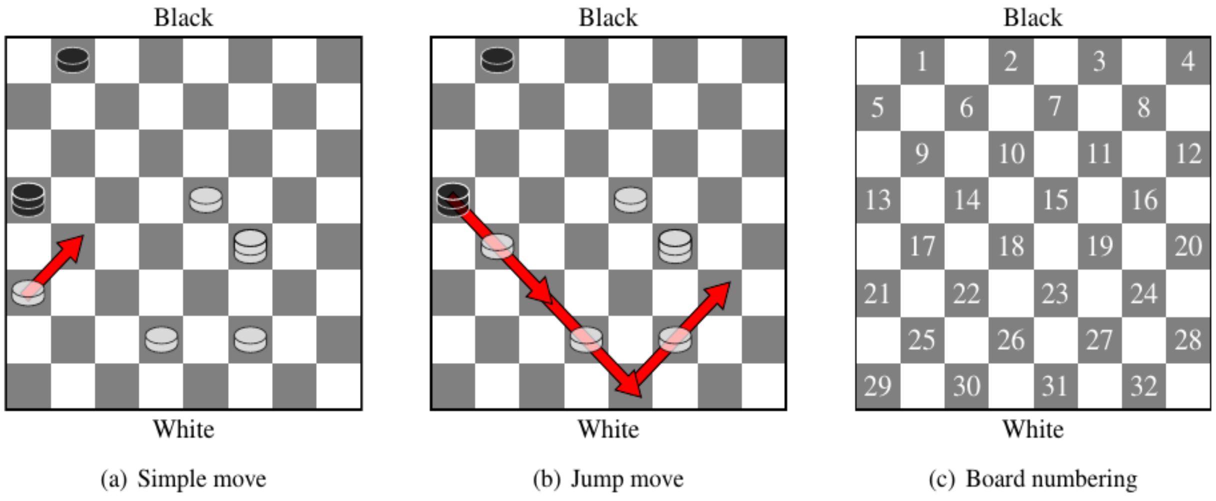


Figure C.1: (a), (b) The first two moves in Sample Input 1. The larger pieces are kings and the smaller ones are men. Black sits at the top and White at the bottom. (c) Numbering used in the input.

In Checkers, captures are forced moves. If a jump move is available at the start of a player's turn, they must jump, and cannot stop jumping with that piece until it has no more possible jumps. They are free to choose which piece to jump with, and where, if there are multiple possibilities. In Figure C.1(b), Black could not have made any other move.

If a man reaches the farthest row from its player (that is, a black man reaches the bottom row or a white man reaches the top row), it is removed from the board and replaced by a king of the same color (it is said to be *promoted*), and the turn ends. A piece cannot be promoted and then jump backwards as a new king in the same turn.

Given a list of moves, find a setup of pieces such that the moves can be legally played in sequence starting from that setup. This setup may not have black men on the bottom row or white men on the top row, since they would have been promoted to kings already. You need only ensure that the rules above are obeyed; you do not need to ensure that this setup is reachable in a real game of Checkers.

Input

The first line of input contains a character c and an integer n, where $c \in \{\texttt{B}, \texttt{W}\}$ indicates which player makes the first move (Black or White respectively) and n ($1 \le n \le 100$) is the number of moves in the list. Then follow n lines, each of which describes a move in the standard Checkers notation defined below.

The dark squares are identified by numbers 1–32, as shown in Figure C.1(c). A simple move from square a to square b is written as $a - b$. A jump move starting at a and jumping to $b_1, b_2, \ldots, b_k$ is written as $a \times b_1 \times b_2 \times \cdots \times b_k$.

There is always a valid solution for the given set of moves.

Output

Output two boards side-by-side (separated by a space), giving the positions of all pieces on the board before (on the left) and after (on the right) the given moves. Use '`-`' for light squares, '`.`' for empty dark squares, lowercase '`b`' and '`w`' for black and white men, and uppercase '`B`' and '`W`' for black and white kings. If there is more than one valid solution, any one is acceptable.

Examples

standard input	standard output
`W 3`	`-b-.-.-. -b-.-.-.`
`21-17`	`.-.-.-.- .-.-.-.-`
`13x22x31x24`	`-.-.-.-. -.-.-.-.`
`19x28`	`B-.-w-.- .-.-w-.-`
	`-.-.-W-. -.-.-.-.`
	`w-.-.-.- .-.-.-.-`
	`-.-w-w-. -.-.-.-W`
	`.-.-.-.- .-.-.-.-`
`B 5`	`-.-b-.-W -.-.-.-W`
`2-7`	`b-b-.-.- .-.-.-.-`
`9x2`	`-w-.-.-. -b-.-.-.`
`32-27`	`B-w-b-.- B-w-.-.-`
`2x11x18`	`-.-.-.-. -.-W-.-.`
`5-9`	`.-.-.-.- .-.-.-.-`
	`-.-.-.-. -.-.-B-.`
	`.-.-.-B- .-.-.-.-`

Problem D. Circular DNA

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

You have an internship with a bioinformatics research group studying DNA. A single strand of DNA consists of many genes, which fall into different categories called *gene types*. Gene types are delimited by specific nucleotide sequences known as *gene markers*. Each gene type i has a unique start marker $\mathtt{s}_i$ and a unique end marker $\mathtt{e}_i$. After many dirty jobs (growing bacteria, cell extraction, protein engineering, and so on), your research group can convert DNA into a form consisting of only the gene markers, removing all the genetic material lying between the markers.

Your research group came up with the interesting hypothesis that gene interpretation depends on whether the markers of some gene types form properly nested structures. To decide whether markers of gene type i form a proper nesting in a given sequence of markers w, one needs to consider the subsequence of w containing only the markers of gene type i ($\mathtt{s}_i$ and $\mathtt{e}_i$), leaving none of them out. The following (and only the following) are considered to be properly nested structures:

- $\mathtt{s}_i\mathtt{e}_i$
- $\mathtt{s}_i N \mathtt{e}_i$, where N is a properly nested structure
- AB, where A and B are properly nested structures

Given your computing background, you were assigned to investigate this property, but there is one further complication. Your group is studying a specific type of DNA called *circular* DNA, which is DNA that forms a closed loop. To study nesting in circular DNA, it is necessary to cut the loop at some location, which results in a unique sequence of markers (the direction of reading is fixed by molecular properties). Whether a gene type i forms a proper nesting now also depends on where the circular DNA is cut. Your task is to find the cutting location that maximizes the number of gene types that form a properly nested structure. Figure D.1 shows an example corresponding to Sample Input 1. The indicated cut results in the markers for gene type 1 being properly nested.

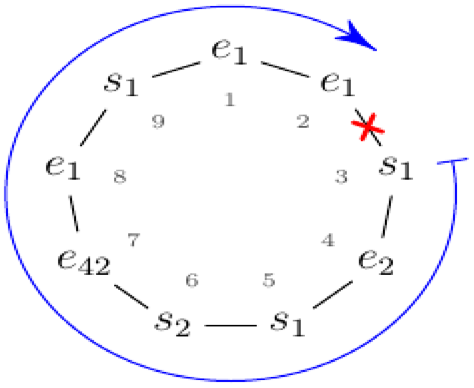


Figure D.1: Illustration of Sample Input 1 with its optimal cutting location.

Input

The first line of input contains an integer n ($1 \le n \le 10^6$), the length of the DNA. The next line contains the DNA sequence, that is, n markers. Each marker is a character c followed by an integer i, where $c \in \{\mathtt{s}, \mathtt{e}\}$ specifies whether it is a start or an end marker and i ($1 \le i \le 10^6$) is the gene type of the marker. The given DNA sequence has been obtained from the circular DNA by cutting at an arbitrary location.

Output

Output one line with two integers p and m, where p is the cutting position that maximizes the number of different gene types that form a proper nesting, and m is this maximum number of gene types. The DNA is cut just before the p^{th} input marker (for instance, the cut shown in Figure D.1 has $p = 3$). If more than one cutting position yields the same maximum value of m, output the smallest p that does so.

Examples

standard input	standard output
9 e1 e1 s1 e2 s1 s2 e42 e1 s1	3 1
8 s1 s1 e3 e1 s3 e1 e3 s3	8 2

Problem E. Dead-End Detector

Input file:	`standard input`
Output file:	`standard output`
Time limit:	5 seconds
Memory limit:	512 megabytes

A dead-end sign.
Source: Wikimedia Commons

The council of your home town has decided to improve road sign placement, especially for dead ends. They have given you a road map, and you must determine where to put up signs to mark the dead ends. They want you to use as few signs as possible.

The road map is a collection of locations connected by two-way streets. The following rule describes how to obtain a complete placement of dead-end signs. Consider a street S connecting a location x with another location. The x-entrance of S gets a dead-end sign if, after entering S from x, it is not possible to come back to x without making a U-turn. A U-turn is a 180-degree turn immediately reversing the direction.

To save costs, you have decided not to install redundant dead-end signs, as specified by the following rule. Consider a street S with a dead-end sign at its x-entrance and another street T with a dead-end sign at its y-entrance. If, after entering S from x, it is possible to go to y and enter T without making a U-turn, the dead-end sign at the y-entrance of T is redundant. See Figure E.1 for examples.

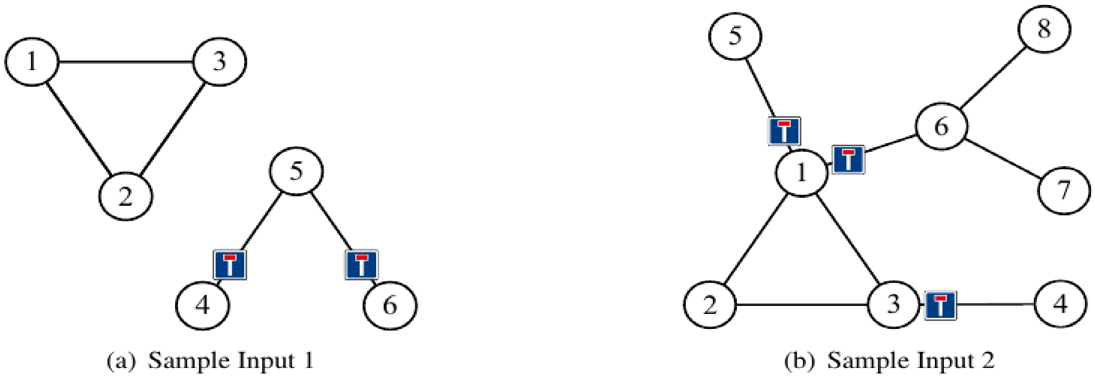


Figure E.1: Illustration of sample inputs, indicating where non-redundant dead-end signs are placed.

Input

The first line of input contains two integers n and m, where n ($1 \le n \le 5 \cdot 10^5$) is the number of locations and m ($0 \le m \le 5 \cdot 10^5$) is the number of streets. Each of the following m lines contains two integers v and w ($1 \le v < w \le n$) indicating that there is a two-way street connecting locations v and w. All location pairs in the input are distinct.

Output

On the first line, output k, the number of dead-end signs installed. On each of the next k lines, output two integers v and w marking that a dead-end sign should be installed at the v-entrance of a street connecting locations v and w. The lines describing dead-end signs must be sorted in ascending order of v-locations, breaking ties in ascending order of w-locations.

Examples

standard input	standard output

6 5
1 2
1 3
2 3
4 5
5 6
	2
	4 5
	6 5

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

Problem F. Directing Rainfall

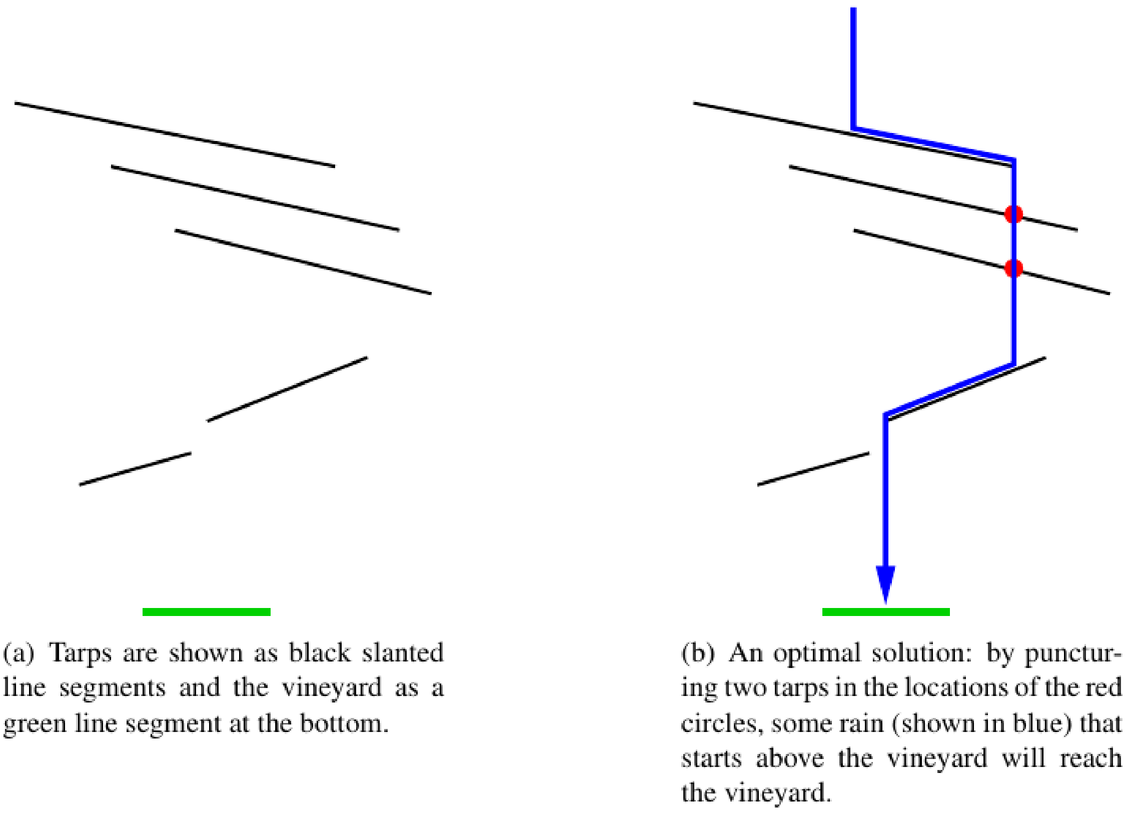
Input file:	`standard input`
Output file:	`standard output`
Time limit:	15 seconds
Memory limit:	256 megabytes

Porto and the nearby Douro Valley are famous for producing port wine. Wine lovers from all over the world come here to enjoy this sweet wine where it is made. The International Consortium of Port Connoisseurs (ICPC) is organizing tours to the vineyards that are upstream on the Douro River. To make visits more pleasurable for tourists, the ICPC has recently installed sun tarps above the vineyards. The tarps protect tourists from sunburn when strolling among the vines and sipping on a vintage port.

Unfortunately, there is a small problem with the tarps. Grapes need sunlight and water to grow. While the tarps let through enough sunlight, they are entirely waterproof. This means that rainwater might not reach the vineyards below. If nothing is done, this year's wine harvest is in peril!

The ICPC wants to solve their problem by puncturing the tarps so that they let rainwater through to the vineyards below. Since there is little time to waste before the rainy season starts, the ICPC wants to make the minimum number of punctures that achieve this goal.

We will consider a two-dimensional version of this problem. The vineyard to be watered is an interval on the x-axis, and the tarps are modeled as line segments above the x-axis. The tarps are slanted, that is, not parallel to the x- or y-axes (see Figure F.1 for an example). Rain falls straight down from infinitely high. When any rain falls on a tarp, it flows toward the tarp's lower end and falls off from there, unless there is a puncture between the place where the rain falls and the tarp's lower end—in which case the rain will fall through the puncture instead. After the rain falls off a tarp, it continues to fall vertically. This repeats until the rain hits the ground (the x-axis).


(a) Tarps are shown as black slanted line segments and the vineyard as a green line segment at the bottom.

(b) An optimal solution: by puncturing two tarps in the locations of the red circles, some rain (shown in blue) that starts above the vineyard will reach the vineyard.

Figure F.1: Illustration of Sample Input 1.

For legal reasons you have to ensure that at least some of the rain that reaches the vineyard originated from directly above the vineyard. This is to prevent any vineyard from stealing all their rain from neighboring vineyards (see the second sample input for an example).

Input

The first line of input contains three integers l, r and n, where (l, r) $(0 \leq l < r \leq 10^9)$ is the interval representing the vineyard and n $(0 \leq n \leq 5 \cdot 10^5)$ is the number of tarps. Each of the following n lines describes a tarp and contains four integers x_1, y_1, x_2, y_2 , where (x_1, y_1) is the position of the tarp's lower end and (x_2, y_2) is the position of the higher end $(0 \leq x_1, x_2 \leq 10^9, x_1 \neq x_2$, and $0 < y_1 < y_2 \leq 10^9)$. The x-coordinates given in the input (l, r, and the values of x_1 and x_2 for all tarps) are all distinct. The tarps described in the input will not intersect, and no endpoint of a tarp will lie on another tarp.

Output

Output the smallest number of punctures that need to be made to get some rain falling from above the vineyard to the vineyard.

Examples

standard input	standard output

10 20 5
32 50 12 60
30 60 8 70
25 70 0 80
15 30 28 40
5 20 14 25
	2

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

Problem G. First of Her Name

Input file:	`standard input`
Output file:	`standard output`
Time limit:	10 seconds
Memory limit:	512 megabytes

In the Royal Family, names are very important! As the Royal Historian you have been charged with analyzing the patterns in the names of the Royal Ladies in the realm.

There have been n Royal Ladies, for convenience numbered from 1 to n. The name of each Lady is an uppercase letter concatenated with the name of her mother. The exception is the Lady numbered 1, the founder of the Royal Family, whose name is just a single uppercase letter.

For example, ENERYS could be the mother of AENERYS (as the name AENERYS consists of the single uppercase letter 'A' concatenated with ENERYS, which is her mother's name). Similarly, AENERYS could be the mother of DAENERYS and YAENERYS.

You are given the description of all the Royal Ladies. Your task is to determine, for certain interesting strings s, the number of Royal Ladies for whom s is a prefix of their name.

For example, consider Sample Input 1 below, with a Royal Line that goes straight from the founder S to AENERYS (through YS, RYS, ERYS, NERYS and ENERYS), with each Lady having exactly one daughter. Then AENERYS has two daughters—DAENERYS and YAENERYS, with the latter having one daughter, RYAENERYS.

In such a family, RY is a prefix of the names of two ladies: RYS and RYAENERYS. E is a prefix of the names of ERYS and ENERYS. N is a prefix only of NERYS's name, while S is a prefix only of the name of the founder, S. AY is not a prefix of any Royal Lady's name.

Input

The first line of input contains two integers n and k, where n ($1 \leq n \leq 10^6$) is the total number of Royal Ladies and k ($1 \leq k \leq 10^6$) is the number of query strings.

Then follow n lines describing the Royal Ladies. The i^{th} of these lines describes the Royal Lady numbered i, and contains an uppercase letter c_i ('A' – 'Z') and an integer p_i , where c_i is the first letter of the name of Lady i, and p_i ($p_1 = 0$ and $1 \leq p_i < i$ for $i > 1$) is the number of her mother (or 0, in the case of the First Lady). All the names are unique.

The remaining k lines each contain one nonempty query string, consisting only of uppercase letters. The sum of the lengths of the query strings is at most 10^6.

Output

Output k lines, with the i^{th} line containing the number of Royal Ladies who have the i^{th} query string as a prefix of their name.

Example

standard input	standard output
10 5	2
S 0	2
Y 1	1
R 2	1
E 3	0
N 4	
E 5	
A 6	
D 7	
Y 7	
R 9	
RY	
E	
N	
S	
AY	

Problem H. Hobsons' trains

Input file:	`standard input`
Output file:	`standard output`
Time limit:	5 seconds
Memory limit:	256 megabytes

Mr. Hobson has retired from running a stable and has invested in a more modern form of transport, trains. He has built a rail network with n stations. However, he has retained his commitment to free the passenger from the burden of too many choices: from each station, a passenger can catch a train to exactly one other station. Such a journey is referred to as a *leg*. Note that this is a one-way journey, and it might not be possible to get back again.

Hobson also offers exactly one choice of ticket, which allows a passenger to travel up to k legs in one trip. At the exit from each station is an automated ticket reader (only one, so that passengers do not need to decide which to use). The reader checks that the distance from the initial station to the final station does not exceed k legs.

Each ticket reader must be programmed with a list of valid starting stations, but the more memory this list needs, the more expensive the machine will be. Help Hobson by determining, for each station A, the number of stations (including A) from which a customer can reach A in at most k legs.

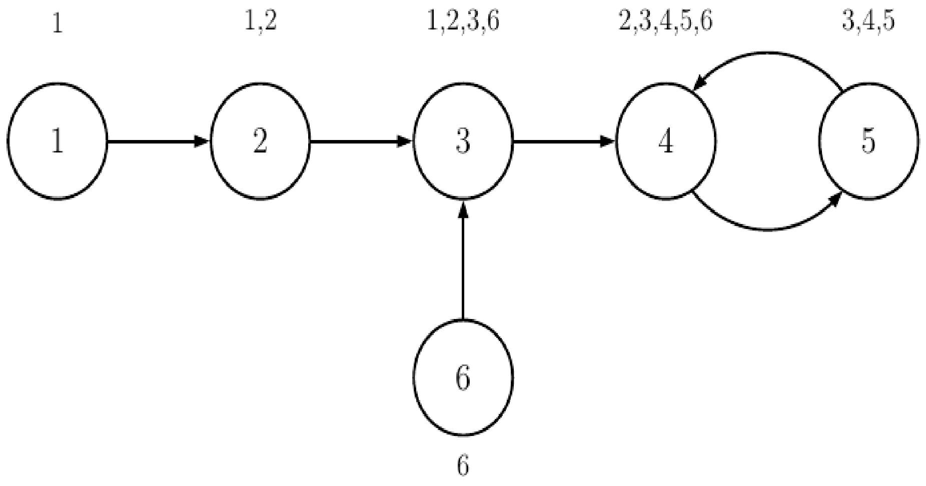


Figure H.1: Illustration of Sample Input 1. Each circle represents a station. The numbers outside the circles are the station numbers loaded into the ticket readers when $k = 2$.

Input

The first line of input contains two integers n and k, where n ($2 \le n \le 5 \cdot 10^5$) is the number of stations and k ($1 \le k \le n - 1$) is the maximum number of legs that may be traveled on a ticket. Then follow n lines, the i^{th} of which contains an integer d_i ($1 \le d_i \le n$ and $d_i \ne i$), the station which may be reached from station i in one leg.

Output

Output n lines, with the i^{th} line containing the number of stations from which station i can be reached in at most k legs.

Examples

standard input	standard output

6 2
2
3
4
5
4
3
	1
	2
	4
	5
	3
	1

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

Problem I. Karel the Robot

Input file:	`standard input`
Output file:	`standard output`
Time limit:	10 seconds
Memory limit:	256 megabytes

Did you know that the word "robot" is almost 100 years old? It was first introduced in 1920, in the science-fiction theatrical play R.U.R., written by Karel Čapek. As a tribute to this Czech writer, an educational programming language was named Karel many years later at Stanford University. Your task is to implement an interpreter of a simplified version of this programming language.

The Karel programming language controls a robot named Karel, who lives in a grid of unit squares. Some of the squares are free, while others contain a barrier. Karel always occupies one of the free squares and faces one of the four cardinal directions. The two basic commands are "move forward" and "turn left". The language also provides simple conditional and looping statements. The main educational potential of the language lies in the possibility of defining new procedures for more complex tasks.

Our simplified version of the language can be described by the following grammar:

```
<program>   :=  "" | <command> <program>
<command>   :=  "m" | "l" | <proc-call> |
                "i" <condition> "(" <program> ")(" <program> ")" |
                "u" <condition> "(" <program> ")"
<condition> :=  "b" | "n" | "s" | "e" | "w"
<proc-call> :=  <uppercase-letter>
<proc-def>  :=  <uppercase-letter> "=" <program>
```

There are five types of commands:

- `m` ("move forward") advances Karel's position by one grid square in its current heading, unless there is a barrier, in which case the command has no effect.
- `l` ("turn left") makes Karel turn left 90 degrees.
- `X` where `X` is any uppercase letter, invokes the procedure named `X`.
- `i` ("if") followed by a single-letter condition, and two programs in parentheses. If the condition is satisfied, the first program is executed. Otherwise, the second program is executed.
- `u` ("until") followed by a single-letter condition, and a program in parentheses. If the condition is satisfied, nothing is done. Otherwise, the program is executed and then the command is repeated.

A condition can be either '`b`', which is satisfied if and only if there is a barrier in the next square in Karel's current heading, or one of the four directional letters '`n`', '`s`', '`e`', or '`w`', which is satisfied if and only if Karel's current heading is north, south, east, or west, respectively.

For instance, a simple program `ub(m)` can be understood to mean: "keep moving forward until there is a barrier", while `un(l)` means "turn to the north". A procedure definition `R=lll` defines a new procedure 'R' which effectively means "turn right".

Input

The first line of input contains four integers r, c, d, and e, where r and c ($1 \leq r, c \leq 40$) are the dimensions of the grid in which Karel lives, d ($0 \leq d \leq 26$) is the number of procedure definitions, and e ($1 \leq e \leq 10$) is the number of programs to be executed.

Then follow r lines describing the grid (running north to south), each containing c characters (running west to east), each character being either '.' (denoting a free square) or '#' (denoting a barrier). All squares outside this given area are considered barriers, which means Karel may never leave the area.

Each of the next d lines contains a procedure definition, associating a procedure name (one uppercase letter) with a program forming the procedure body. No procedure name is defined more than once. Procedure bodies may contain invocations of procedures that have not yet been defined.

The last $2e$ lines describe the programs to be executed. Each such description consists of a pair of lines. The first line of each pair contains two integers i and j and a character h, where i $(1 \le i \le r)$ is the row and j $(1 \le j \le c)$ is the column of Karel's initial position, and $h \in \{\texttt{n}, \texttt{s}, \texttt{e}, \texttt{w}\}$ represents Karel's initial heading. It is guaranteed that the initial position is a free square. The second line of each pair contains a program to be executed from that initial position.

All procedure bodies and all programs to be executed are at least 1 and at most 100 characters long, syntactically correct, and only contain invocations of procedures that are defined. The lines with procedure definitions and programs to be executed contain no whitespace characters.

Output

For each program execution, output the final position of Karel after the complete program is executed from the respective initial position. Follow the format used to describe initial positions, that is, two numbers and a directional character. If a particular execution never terminates, output `inf` instead.

Example

standard input	standard output
`4 8 5 7`	`1 1 w`
`.......#`	`inf`
`..#....#`	`1 1 w`
`.###...#`	`2 4 s`
`.....###`	`4 4 e`
`R=lll`	`1 4 e`
`G=ub(B)`	`inf`
`B=ub(m)lib(l)(m)`	
`H=ib()(mmHllmll)`	
`I=III`	
`1 1 w`	
`G`	
`1 1 e`	
`G`	
`2 2 n`	
`G`	
`2 6 w`	
`BR`	
`4 1 s`	
`ib(lib()(mmm))(mmmm)`	
`1 1 e`	
`H`	
`2 2 s`	
`I`	

Problem J. Miniature Golf

Input file:	`standard input`
Output file:	`standard output`
Time limit:	6 seconds
Memory limit:	256 megabytes

A group of friends has just played a round of miniature golf. Miniature golf courses consist of a number of holes. Each player takes a turn to play each hole by hitting a ball repeatedly until it drops into the hole. A player's score on that hole is the number of times they hit the ball. To prevent incompetent players slowing down the game too much, there is also an upper limit l (a positive integer) on the score: if a player has hit the ball l times without the ball dropping into the hole, the score for that hole is recorded as l and that player's turn is over. The total score of each player is simply the sum of their scores on all the holes. Naturally, a lower score is considered better.

There is only one problem: none of the players can remember the value of the integer l. They decide that they will not apply any upper limit while playing, allowing each player to keep playing until the ball drops into the hole. After the game they intend to look up the value of l and adjust the scores, replacing any score on a hole that is larger than l with l.

The game has just finished, but the players have not yet looked up l. They wonder what their best possible ranks are. For this problem, the *rank* of a player is the number of players who achieved an equal or lower total score after the scores are adjusted with l. For example, if the adjusted scores of the players are 3, 5, 5, 4, and 3, then their ranks are 2, 5, 5, 3 and 2 respectively.

Given the scores of the players on each hole, determine the smallest possible rank for each player.

Input

The first line of input contains two integers p and h, where p ($2 \le p \le 500$) is the number of players and h ($1 \le h \le 50$) is the number of holes. The next p lines each contain h positive integers. The j^{th} number on the i^{th} of these lines is the score for player i on hole j, and does not exceed 10^9.

Output

Output a line with the minimum possible rank for each player, in the same order as players are listed in the input.

Examples

standard input	standard output

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

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

Problem K. Traffic Blights

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

Cars! Where do they come from? Where do they go? Nobody knows. They appear where roads have been built, as if out of nowhere. Some say that no two cars are alike. Some say that if you look closely, you can see the pale ghosts of miserable humans inside them, trapped forever—particularly in the morning and late afternoon. What scientific eye could frame their fearful symmetry?

Well, yours, hopefully. As part of your government's Urban Traffic Control department, you are trying to write a paper on local traffic congestion. It is too dangerous to observe cars in the wild, of course, but you have been given some data on the traffic lights along your town's Main Street, and you would like to do some theoretical calculations about how well-synchronized they are.

Main Street is set out on a line, with traffic lights placed at various points along it. Each traffic light cycles between red and green with a fixed period, being red for r seconds, then green for g seconds, then red for r seconds, and so on. The values of r and g may be different for different traffic lights. At time 0, all the lights have just turned red.

Assume that an "ideal" car mystically appears at the west end of Main Street at a uniformly random real-valued time in the interval $[0, 2019!]$ (where $k!$ is the product of the first k positive integers), driving eastwards at a slow crawl of 1 meter/second until it hits a red light. What is the probability that it will make it through all the lights without being forced to stop? If it does hit a red light, which one is it likely to hit first?

Write a program to answer these questions.

Input

The first line of input contains an integer n ($1 \leq n \leq 500$), the number of traffic lights. Each of the following n lines contains three integers x, r, and g describing a traffic light, where x ($1 \leq x \leq 10^5$) is the position of the light along Main Street in meters, and r and g ($0 \leq r, g$ and $1 \leq r + g \leq 100$) are the durations in seconds of the red and green portions of the light's period (so the light is red from time 0 to r, from time $r + g$ to $2r + g$, and so on).

The west end of Main Street is at position 0, and the lights are listed in order of strictly increasing position.

Output

For each of the n lights, output a line containing the probability that this light will be the first red light an "ideal" car hits. Then output a line containing the probability that an "ideal" car makes it all the way without stopping. Your answers should have an absolute error of at most 10^{-6}.

Examples

standard input	standard output

4
1 2 3
6 2 3
10 2 3
16 3 4
	0.4
	0
	0.2
	0.171428571429
	0.228571428571

6
4 1 5
9 8 7
13 3 5
21 5 7
30 9 1
2019 20 0
	0.166666666667
	0.466666666667
	0.150000000000
	0.108333333333
	0.091666666667
	0.016666666667
	0.000000000000