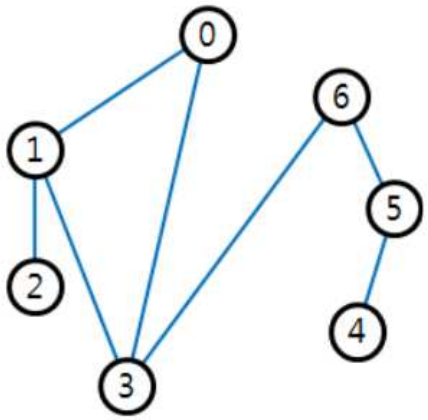
7600 Bridge Park

There is a bridge park in Yeosu city which is one of beautiful ocean cities in Korea. There are n small islands and m bridges connecting them. The islands in the park are located along a circumference. A bridge in the park is connecting exactly two islands and no two bridges cross. Let us assume an island is a point. Then each island is located at the position of a vertex of a convex polygon which is a simple polygon whose all interior angles are strictly less than 180 degrees. Also, a bridge is considered as a straight line segment connecting two points, and no two segments cross except at the end points.

All islands are connected by the bridges, so a person on an island can visit all other islands through these bridges. The figure on the right shows an example of the bridge park.



One day, when a lot of tourists visited the park, there was an accident where a bridge was broken, thus some tourists were trapped on some isolated islands for a long time. After the accident, the park management committee made a plan of adding bridges such that all islands remain connected even if one bridge is broken. Of course, additional bridges must be straight line segments and should not intersect with any other bridges except for end points. The committee wants to know the minimum number of additional bridges required in order to achieve the goal with least cost.

Let (i, j) be the bridge connecting two islands i and j. In above figure, if the bridge (6, 3) is broken, the islands are partitioned into two connected components {0, 1, 2, 3} and {4, 5, 6}. If the bridge (5, 6) is broken, the islands are partitioned into two connected components {0, 1, 2, 3, 6} and {4, 5}. If the bridge (0, 3) is broken, the islands remain connected. If two bridges (2, 3) and (3, 4) are added, all islands remain connected even if any one bridge is broken.

Given the information of islands and bridges of the park, write a program to find the minimum number of additional bridges such that all islands remain connected even if any one bridge is broken.

Input

The input file contains several test cases, each of them as described below.

Your program is to read from standard input. The input starts with a line containing two integers, n and m ($3 \le n \le 100,000$, $n-1 \le m \le 2n-3$), where n and m are the numbers of islands and bridges of the park, respectively. All islands are numbered from 0 to $n-1$ and positioned at the vertices of a convex polygon. The ordered list $(0, 1, 2, \ldots, n-1)$ is a sequence of islands visited when we traverse the boundary of the convex polygon in counterclockwise order. In the following m lines, two integers i and j ($0 \le i, j \le n-1$) are given line by line, where (i, j) represents a bridge connecting the islands i and j. Note that all islands are connected by the bridges.

Output

For each test case, the output must follow the description below.

Your program is to write to standard output. Print exactly one line which contains an integer which is the minimum number of additional bridges such that all islands remain connected even if one bridge is broken.

Sample Input

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

Sample Output

```
2
0
```

7601 Football

Football is one of the most popular sports in the world. There is a football league with n teams. A team plays against all the other teams and there is exactly one match for each pair of teams. Therefore, each team plays $n - 1$ matches. A draw is broken by the penalty shoot-out so there is no draw. After a match, if a team wins, it earns one point. If it loses, then it earns zero point.

After the match schedule is over, each team reports its point to the league office to decide the best team. The league office wants to be sure that each team made no mistake in reporting its point. Especially it wants to know whether the reported points are valid, that is, it is possible to assign those points to teams under the league rule.

Given n integers which are points reported by the teams, write a program to determine whether they are valid.

Input

The input file contains several test cases, each of them as described below.

Your program is to read from standard input. The input consists of two lines. The first line contains an integer n ($2 \leq n \leq 10,000$) where n is the number of teams. The next line contains n integers which are points reported by teams. Each integer is between zero and $n - 1$, inclusive.

Output

For each test case, the output must follow the description below.

Your program is to write to standard output. Print '`1`' if the reported points are valid. Otherwise, print '`-1`'.

Sample Input

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

Sample Output

```
1
-1
```

7602 House Rental

Acmi is a medium-sized city, through which there is a long main street from the west to the east. All facilities in Acmi, such as supermarkets, schools, train stations, bus stops, and hospitals, are located on the main street.

You are going to rent a house in Acmi that is on the main street. You think the location is the most priority in choosing your house. In particular, you have a list of k facility types and want to minimize the distance to each facility. For example, if you think supermarket is important and so it is in your list of k facility types, then you want your house to be located close to a supermarket.

If $k = 1$, then the problem is easy because you will rent a house just in front of any facility of the type you want. But in general cases where $k > 1$, this strategy may not work.

Given n facilities of k different types along the main street, you are to write a program that finds a best location of your house that minimizes the maximum among the distances to the nearest facility of each type. The location of every building along the main street is represented by an integer from $-1,000,000,000$ to $1,000,000,000$. The smaller location number a building on the street has, the more to the west it is located along the street. The distance between two buildings is considered to be exactly the difference of their location numbers. Note that there may be two or more facilities at a common location, and that there is at least one facility with each of the k facility types. Assume that there is at least one vacant house you can rent at every integral location number.

Input

The input file contains several test cases, each of them as described below.

Your program is to read from standard input. The input starts with a line containing two integers k ($1 \leq k \leq 100,000$) and n ($k \leq n \leq 1,000,000$), where n is the number of facilities along the main street and k is the number of different facility types. In the following n lines, the location of each of the n facilities and its type are given line by line in the form of two integers: the first integer represents the location between $-1,000,000,000$ and $1,000,000,000$, inclusively, along the street, and the second integer between 1 to k represents the type.

Output

For each test case, the output must follow the description below.

Your program is to write to standard output. Print exactly one line for the input. The line should contain an integer that represents a best location of your house for the input. If there are two or more best locations, then you must output the smallest location number among them.

Sample Input

```
1 5
-100 1
-10 1
0 1
1 1
2 1
5 5
-2 1
0 3
```

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

Sample Output

```
-100
0
0
```

7603 Independent Edges

Let G be a simple undirected graph. Two vertices of G are said to be *adjacent* if they are connected by an edge, and two edges of G are said to be *adjacent* if they share a vertex. In the graph shown in Figure 1(a), vertices 3 and 4 are adjacent because there is an edge (3,4) connecting them; edges (3,4) and (3,5) are adjacent because they share a vertex 3. A subset of vertices of G is called an *independent vertex set* if no two vertices in the subset are adjacent; also, a subset of edges of G is called an *independent edge set* if no two edges in the subset are adjacent. The size of a maximum independent vertex set, an independent vertex set of largest possible size, of G is called the *vertex independence number* and denoted by $\alpha(G)$; analogously, the size of a maximum edge independent set of G is called the *edge independence number* and denoted by $\upsilon(G)$.

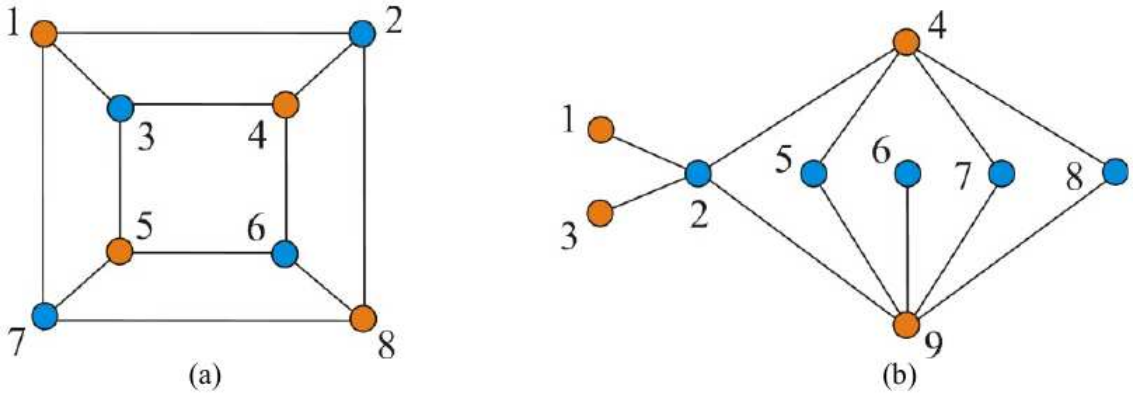


Figure 1. Two graphs G_1 and G_2 : (a) graph G_1, where $\alpha(G_1) = 4$ and $\upsilon(G_1) = 4$; (b) graph G_2, where $\alpha(G_2) = 6$ and $\upsilon(G_1) = 3$.

The INDEPENDENT EDGE SET problem is the problem of finding a maximum independent edge set of a graph. The problem is well-studied, so many algorithms for the problem have been designed and implemented, where most of the algorithms run in time polynomial to the size of the graph. Sometimes, we expect something more than the output of a computer program, so as to be sure of the correctness of the output. This motivates the study of so-called certifying algorithms.

When the user gives X as an input and the program outputs Y, the user usually has no way of knowing whether Y is a correct output on input X or it has been compromised by a bug. A *certifying algorithm* is an algorithm that produces, with each output, a *certificate* Z that the particular output has not been compromised by a bug. By inspecting the certificate, either manually or by use of a program, the user can convince him/her that the output is correct, or reject the output as buggy. The process of checking Z can be automated with a *checker*, which is an algorithm for verifying that Z proves that Y is a correct output for X.

Let us think of the certificate that should be produced by a certifying algorithm for the INDEPENDENT EDGE SET problem on a bipartite graph. Here, we say that a graph is *bipartite* if its vertex set can be divided into two disjoint sets U and V in such a way that every edge connects a vertex in U to one in V. The graph of Figure 1(a) is bipartite, where the vertex set is partitioned into two sets of orange-colored vertices and blue-colored vertices; the same follows for the graph of Figure 1(b). If the input graph is bipartite, we can utilize König's theorem which states that $\upsilon(G) + \alpha(G) = n$ for any bipartite graph G with n vertices. This suggests that if the input graph is bipartite, its maximum independent vertex set can serve as a good certificate. Once we have identified an independent edge set of size k and an independent vertex set of size $n - k$ in a bipartite graph G with n vertices, then it

is evident that $\upsilon(G) = k$ and $\alpha(G) = n - k$, i.e., the independent edge set and the independent vertex set found are both maximum possible.

Given a bipartite graph G, your task is to devise a certifying algorithm for finding a maximum independent edge set of G. The algorithm should produce, in addition to an output Y, a certificate Z which has been described above. It is assumed that the graph G has n vertices that are indexed from 1 to n. It may be easy to write a checker that determines if no two edges in Y are adjacent, and determines if no two vertices in Z are adjacent, and finally accepts the output if $|Y| + |Z| = n$.

Input

The input file contains several test cases, each of them as described below.

Your program is to read from standard input. The first line contains two positive integers n and m, respectively, representing the numbers of vertices and edges of the input graph, where $n \leq 1,000$ and $m \leq 50,000$. It is followed by m lines, each contains two positive integers u and v that represent an edge between vertex u and vertex v of the input graph. The input graph is a bipartite graph that is not necessarily connected.

Output

For each test case, the output must follow the description below.

Your program is to write to standard output. The first line should contain an integer, k, indicating the size of a maximum independent edge set of the input graph. In the following k lines, each contains an edge of the maximum independent edge set. Then, a certificate produced by your algorithm should follow: a line containing an integer, k', representing the size of the certificate is followed by a line containing the members of the certificate in ascending order.

Sample Input

```
8 12
1 2
2 8
8 7
7 1
3 4
4 6
6 5
5 3
1 3
2 4
5 7
6 8
9 11
4 2
4 5
4 7
4 8
9 2
9 5
9 6
9 7
9 8
```

```
3 2
2 1
```

Sample Output

```
4
2 4
8 6
7 5
1 3
4
1 4 5 8
3
3 2
5 4
9 6
6
1 3 5 6 7 8
```

7604 Memory Cell

Taeyang has to compute a very complex expression using his calculator. He wants to use the memory facility in the calculator, but there is only one memory cell since his calculator is a quite old model. However, he wants to maximize the usability of the single memory cell by finding the largest common subexpression for a given input expression.

For example, look at the following input expression,

$$1 + (2 \times 3) + 5/(1 + 2 \times 3)$$

This expression has two occurrences of 2×3, so it is a common subexpression. There are another two occurrences of $1 + 2 \times 3$. Thus this expression has two common subexpressions, 2×3 and $1 + 2 \times 3$. The largest (or longest) common subexpression is $1 + 2 \times 3$. Note that the first pair of parentheses (2×3) in the input expression is superfluous since $1 + (2 \times 3)$ is same as $1 + 2 \times 3$ by the precedence between operators. Taeyang wants to find a largest common subexpression and to store the result of it into the single memory cell. You write a program to help Taeyang to find the largest common subexpression for a given expression.

To simplify the problem, the only four types of binary operators are considered, i.e. the addition (+), the subtraction (-), the multiplication ($\times$), and the division (/). The usual precedence rules (the multiplication and the division have higher precedence over the addition and the subtraction) are applied and all the operators are assumed left-associative. Hence, the expression tree for the above expression can be depicted as in Figure 1.

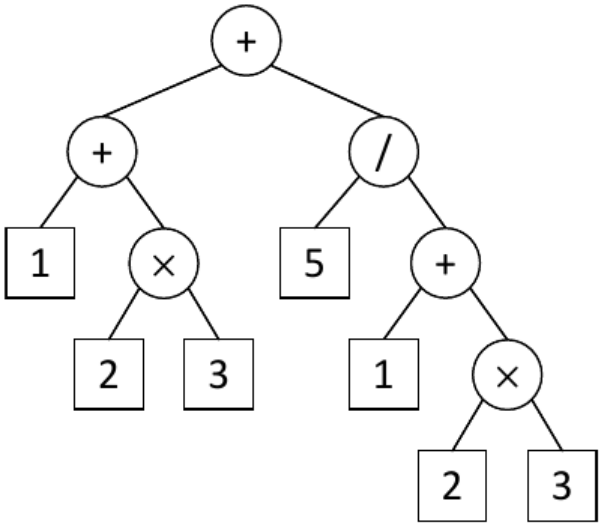


Figure 1. The expression tree for $1 + (2 \times 3) + 5/(1 + 2 \times 3)$.

When you construct the expression tree, you should apply the precedence and the associativity of operators (**Rule 1**), and also preserve the order of operands in the input expression, i.e., the order in the expression must be the same as that of in-order traversal in the tree (**Rule 2**). The expression tree is an ordered, single-rooted tree; a subexpression corresponds to a unique subtree of the expression tree whose leaf nodes represent the operands in the same order occurred in the expression. Thus a common subexpression exists if there are two disjoint, identical subtrees in the expression tree. Finally,

the common subexpression should not be overlapped (**Rule 3**). Even though Rule 3 is a natural consequence of Rule 1, this rule is specified for clarity.

The size of an expression is the number of operators and operands included in the expression, which corresponds to the number of nodes of the expression tree. Therefore, the parentheses for clarifying the bindings between operators and the operands are not counted for the size. For instance, the size of the expression $(50+50)\times((2)+3)$ is seven even though it contains many pairs of parentheses. Note that the common subexpression should be proper, i.e. the size of it is less than that of the whole expression. As a result, finding a largest common expression in a given expression is equivalent to finding two disjoint same subtrees in the corresponding expression tree satisfying Rule 1, 2, and 3 with the maximum size.

Let us consider, for example, an expression $(1+1)\times(1+1)\times(1+1)\times(1+1)\times(1+1)$. For this expression, you may think a largest common subexpression is $(1+1)\times(1+1)\times(1+1)\times(1+1)$, but its two occurrences overlap each other, so this violates Rule 3. You also think a common subexpression would be $(1+1)\times(1+1)$, but it is not because there are no two disjoint same subtrees corresponding to two disjoint occurrences of $(1+1)\times(1+1)$ in the expression tree in Figure 2. Actually $1+1$ is a unique common subexpression in the given expression, so it is the largest.

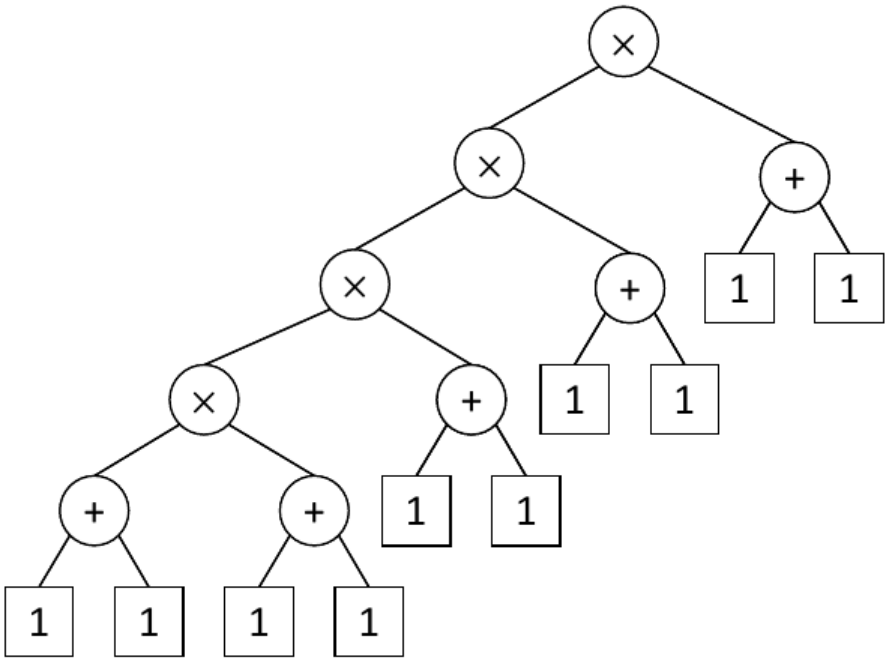


Figure 2. The expression tree for $(1+1)\times(1+1)\times(1+1)\times(1+1)\times(1+1)$.

As a final example, the common subexpression of $(1+2\times3)+5/(2\times3+1)$ should be 2×3 instead of $1+2\times3 or 2\times3+1$ because the order of the operands are different, which violates Rule 2.

Input

The input file contains several test cases, each of them as described below.

Your program is to read from standard input. The input consists of a single line containing the input expression. It consists of the operands (positive integers), the binary operators (`+`, `-`, `*`, `/`), and parentheses. The maximum length of the input line is 1,000 including the newline character. For the multiplication, the symbol '`*`' is used instead of $\times$. The operators, the operands, and the parentheses can be separated by zero or more white spaces. Assume that the input expression contains at least one common subexpression and also assume that the size of it should be greater than or equal to three.

Output

For each test case, the output must follow the description below.

Your program is to write to standard output. Print the postfix for the largest common subexpression in a single line. The postfix of an expression is the result of the post-order traversal of the expression tree. For instance, the postfix of the expression '`1 + 2 * 3`' is '`1 2 3 * +`'. In the output, the operators and the operands should be separated by a space. If there exist different largest common subexpressions, print arbitrary one of them. If your input is an invalid expression, your program should print '`ERROR`'.

Sample Input

```
1 + (2 * 3) + 5 / (1 + 2 * 3)
(1+1)*(1*1)*(1+1)*(1*1)*(1+1)
1 +(12*(3)) +(4+1)/((12)*3+1)
(1+1)*(1*1)*(1+1)//(1*1)*(1+1)
1 +(12*(3)))+(4+1)/((12)*3+1)
```

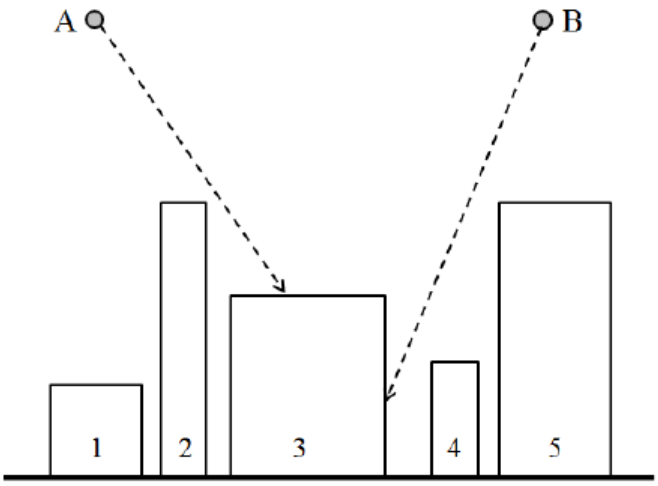
Sample Output

```
1 2 3 * +
1 1 *
12 3 *
ERROR
ERROR
```

7605 Meteorites

NSC(Naro Space Center) has observed that k meteorites are falling to our city. NSC wants to know which buildings in the city will be collided with the falling meteorites.

They are considering this problem in 2-dimensional version first. There are n buildings in the city. All buildings are represented by rectangles whose bottom sides are located on the x-axis. All rectangles are disjoint, i.e., any two distinct rectangles do not intersect each other. Let's distinguish the rectangles by the integers from 1 to n. The rectangle 1 is the leftmost rectangle. The rectangle i lies on the left of the rectangle j if $1 \leq i < j \leq n$ (See the figure below).



All meteorites are represented by points. Every meteorite is falling along a line whose slope is represented by a pair of integers (dx, dy). If a meteorite is currently located on (x, y) and its slope is (dx, dy), it is falling along the line connecting two points (x, y) and $(x + dx, y + dy)$. In the above figure, we can see that both of the meteorites A and B are rushing toward the building 3. If a meteorite touches a point of the boundary of a building, it will be immediately perished by explosion.

Given a data set of n buildings and k meteorites, write a program to compute which building will be damaged for each meteorite.

Input

The input file contains several test cases, each of them as described below.

Your program is to read from standard input. The input starts with a line containing an integer, n $(1 \leq n \leq 100,000)$, where n is the number of rectangles representing the buildings in the city. In the following n lines, each of n rectangles is given line by line in left-to-right order. All rectangles are numbered from 1 to n in order given as the input. Each rectangle is represented by three integers, x_1, x_2, and h $(1 \leq x_1 < x_2 \leq 10^9, 1 \leq h < 10^9)$, where x_1 and x_2 are the x-coordinates of the left side and the right side of the rectangle, respectively, and h is the y-coordinate of the top side of it. Note that the bottom sides of all rectangles are located on the x-axis and the rectangle i lies on the left of the rectangle j if $1 \leq i < j \leq n$.

The next line contains an integer, k ($1 \leq k \leq 100,000$) which represents the number of meteorites. In the following k lines, each of k meteorites is given line by line. Each meteorite is represented by four integers, x, y, dx, and dy ($1 \leq x \leq 10^9$, $\max h < y \leq 10^9$, $-1,000 \leq dx \leq 1,000$, $-1,000 \leq dy \leq -1$), where (x, y) is the current coordinate of the meteorite, (dx, dy) represents a slope of the line along which the meteorite is falling, and $\max h$ is the highest value among the y-coordinates of the top sides of the buildings.

Output

For each test case, the output must follow the description below.

Your program is to write to standard output. Print exactly one line for each meteorite in order given as the input. The line should contain the building number which will be collided with a meteorite. If no building will be damaged, print the zero (0).

Sample Input

```
5
5 9 4
10 12 12
13 20 8
22 24 5
25 30 12
2
7 20 3 -4
27 20 -3 -7
4
10 12 6
13 20 15
22 24 2
25 30 9
4
21 18 0 -2
27 20 3 -5
11 20 20 -15
6 20 1 -5
```

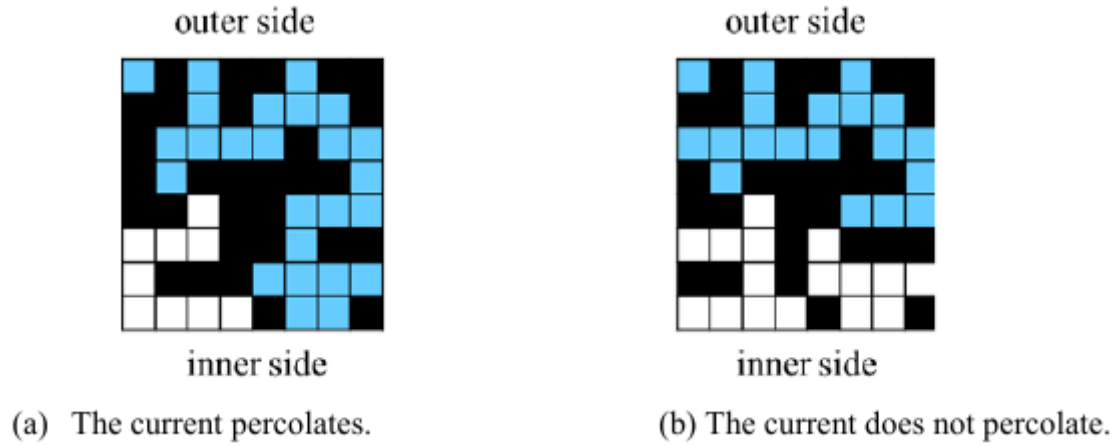
Sample Output

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

7606 Percolation

Professor Kim is in the process of developing a new fabric material such that the electrical current percolates. The cross section of the new fabric can be shown in the two-dimensional $M \times N$ grid. We assume that the top of the grid is the outer side of the fabric and the bottom is the inner side. The black colored cell insulates the current while the white cell conducts the current. The current travels from the outer side to the inner side of the fabric only through the white cells that are adjacent to each other. When the white cells are connected diagonally, the current cannot flow (see Figure 1). The professor wants to test whether the current could percolate from the outer side to the inner side through the material or not.

For example, the following Figure 1(a) shows that the current percolates through the material but Figure 1(b) does not:



(a) The current percolates. (b) The current does not percolate.

Figure 1. Two examples.

When the information on each cell is given, write a program that computes whether the current can pass through the material or not.

Input

The input file contains several test cases, each of them as described below.

Your program is to read from standard input. The first line of the input contains two integers M ($2 \leq M \leq 1,000$) and N ($2 \leq N \leq 1,000$) which represent the size of the fabric. In the following M lines, the information about each cell is given. Each line consists of N characters which composed by '`0`' or '`1`'. The character '`0`' represents the conduct cell and '`1`' represents the insulate cell.

Output

For each test case, the output must follow the description below.

Your program is to write to standard output. If the current can pass through the material from the top to the bottom, print '`YES`'. If not, print '`NO`'.

Sample Input

```
5 6
010101
010000
011101
100011
001011
8 8
11000111
01100000
00011001
11001000
10001001
10111100
01010000
00001011
```

Sample Output

```
NO
YES
```

7607 Power Supply

There is a tree T with N vertices, as a power delivery network. Each vertex of T is either a supply vertex or a demand vertex, which has a positive integer called the supply or demand, respectively. Each demand vertex can receive power from exactly one supply vertex through edges in T. That brings up a flow of power through edges. Each edge is assigned a positive integer called the capacity.

One wishes to partition T into subtrees by deleting edges from T, if necessary. Of course, T itself can be a partition. But the followings should be satisfied:

(a) Each subtree contains exactly one supply vertex whose supply is no less than the sum of all demands in the subtree.

(b) The flow of power through each edge is no more than the capacity of the edge.

Your program should answer to the question of whether T has such a partition.

For example, Figure 1(a) depicts a tree T: each supply vertex is drawn by a rectangle, each demand vertex by a circle, the supply or demand is written inside, and the capacity is attached to each edge. The tree T has a desired partition as illustrated in Figure 1(b): the deleted edges are drawn by dashed lines and each subtree is surrounded by a dotted line: the flow value is attached to each edge and the flow direction is indicated by an arrow.

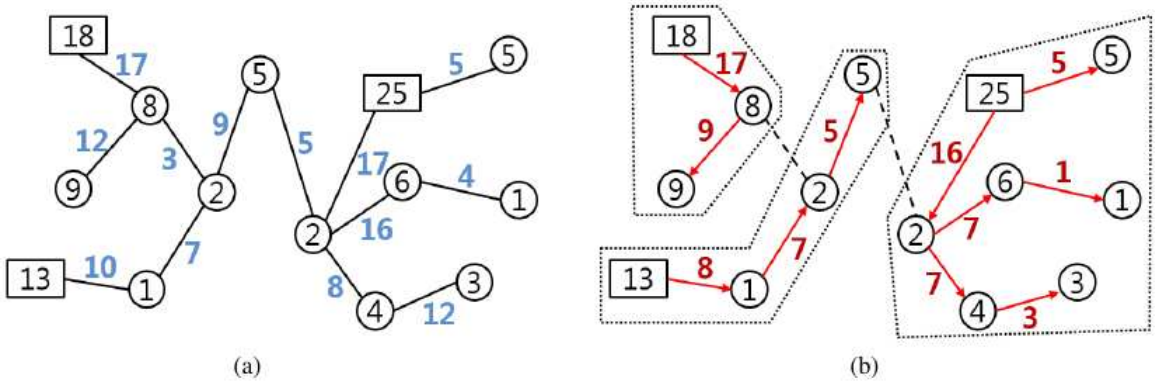


Figure 1. Illustration of a partition.

Input

The input file contains several test cases, each of them as described below.

Your program is to read from standard input. The first line of the input contains one integer N to represent the number of vertices in the tree T ($1 \le N \le 300,000$). The vertices are represented by integers 1, 2, ..., N. The i-th line of the next N lines contains two integers a and b, where if $a = 0$, then the vertex i is a supply vertex with the supply b, and if $a = 1$, then the vertex i is a demand vertex with the demand b ($i = 1, \ldots, N$, $a = 0$ or 1, and $1 \le b \le 10^9$). Also each of the next $N - 1$ lines contains three integers x, y, and z to represent one edge in T, joining two vertices x, y with the capacity z ($1 \le x, y \le N$, $1 \le z \le 10^9$).

Output

For each test case, the output must follow the description below.

Your program is to write to standard output. Print either '`0`' or '`1`' in the line. If there is a desired partition from T, then print '`1`'. Otherwise, print '`0`'.

Sample Input

```
4
1 3
1 1
0 8
1 4
2 1 3
3 2 8
2 4 4
4
1 3
1 1
0 8
1 4
2 1 3
3 2 7
2 4 4
14
0 18
0 13
1 9
1 8
1 1
1 2
1 5
1 2
1 6
1 4
0 25
1 5
1 1
1 3
4 1 17
4 3 12
6 4 3
5 6 7
5 2 10
7 6 9
7 8 5
12 11 5
8 11 17
9 8 16
9 13 4
10 8 8
14 10 12
```

Sample Output

```
1
0
```

```
1
```

7608 Robot

A robot receives a series of instructions, and moves in a square region S along the direction parallel to x-axis or y-axis that the instructions tell. The lower left corner of S is $(0, 0)$, and the upper right corner of S is (M, M). Initially, the robot is located at $(0, 0)$, and heads to the east direction.

An instruction is a pair of *action* and *value*, where the action is the type that the robot acts at the current robot position, and the value is a parameter with which the robot acts. There are two actions, `TURN` and `MOVE`. The instruction '`TURN 0`' makes the robot turn 90 degrees to the left at its current position, and '`TURN 1`' makes the robot turn 90 degrees to the right at its current position. The instruction '`MOVE` d' makes the robot move d units to the direction the robot currently heads. The distance d is a positive integer.

An instruction is valid if the position that the robot stays after executing the instruction is inside or boundary of S. In other words, the instruction is not valid if it makes the robot leave S completely. A series of the instructions is valid if the instructions are all valid.

We suppose that a robot gets a series of instructions, (`MOVE 6, TURN 0, MOVE 5, TURN 0, MOVE 2, TURN 0, MOVE 2, TURN 0, MOVE 4, TURN 0, MOVE 3, MOVE 2`) as in the left figure below, then it finally reaches at $(8, 8)$. For another series of the instructions, (`MOVE 10, TURN 0, MOVE 2, TURN 0, MOVE 5, TURN 1, MOVE 5, TURN 1, MOVE 2, TURN 1, MOVE 3, TURN 0, TURN 0, MOVE 6`) in the middle figure, the robot will be at $(7, 10)$. However, the final series of the instructions, (`MOVE 5, TURN 0, MOVE 4, TURN 1, MOVE 2, TURN 1, MOVE 5`) in the right figure, makes the robot move out of S, i.e., enter the exterior of S, so this series is not valid.

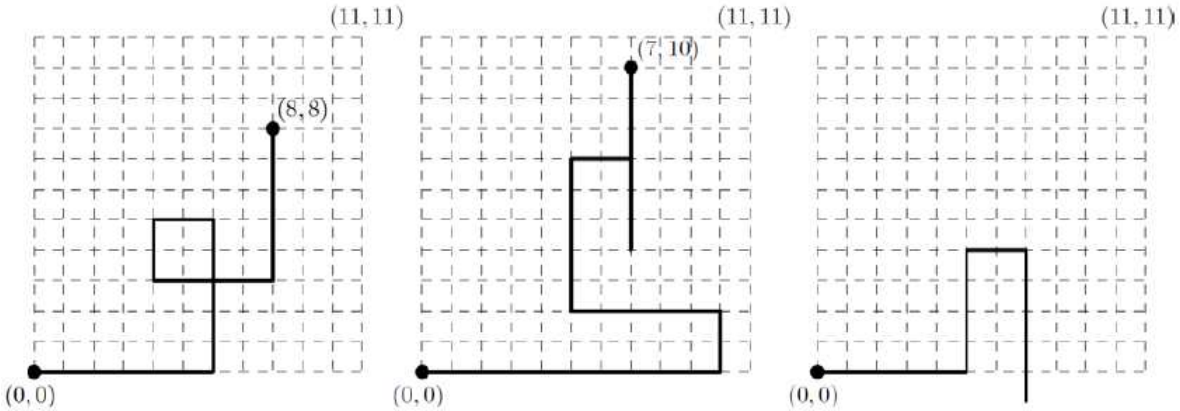


Figure 1. Three routes of the robot for three different series of instructions where $M = 11$.

Given a side length M of a square, a series of n (≥ 1) instructions, and a robot at an initial position $(0, 0)$ of S heading to the east, write a program to output the final position the robot reaches after executing n instructions one by one.

Input

The input file contains several test cases, each of them as described below.

Your program is to read from standard input. The input starts with a line containing two integers, M and n ($1 \leq M \leq 1,000$, $1 \leq n \leq 1,000$), where M is the side length of the square S, i.e., its upper right corner has coordinates $(x, y) = (M, M)$, and n is the number of instructions the robot will execute. Each of the following n lines contains a single instruction. Each instruction is represented by

a pair of '`TURN`' and '*dir*' or by a pair of '`MOVE`' and '*d*', where *dir* is '`0`' or '`1`' and *d* is a positive integer no more than 1,000. Note that the robot is initially at $(0, 0)$ of S and heads to the east direction.

Output

For each test case, the output must follow the description below.

Your program is to write to standard output. Print exactly one line. If the series of the instructions given in the input is valid, the line should contain two non-negative integers, which are x-coordinate and y-coordinate, separated by a space, of the position that the robot reaches just after executing all the instructions. If the series is not valid, then the line should contain a single integer '`-1`'.

Sample Input

```
11 14
MOVE 10
TURN 0
MOVE 2
TURN 0
MOVE 5
TURN 1
MOVE 5
TURN 1
MOVE 2
TURN 1
MOVE 3
TURN 0
TURN 0
MOVE 6
11 7
MOVE 5
TURN 0
MOVE 4
TURN 1
MOVE 2
TURN 1
MOVE 5
2 7
MOVE 2
TURN 0
MOVE 3
TURN 0
MOVE 2
TURN 0
MOVE 2
```

Sample Output

```
7 10
-1
-1
```

7609 Room

A simple polygon in the plane is called *monotone* if every vertical line intersects the interior of the polygon in at most one line segment, and is called *orthogonal* if every edge of the polygon is either horizontal or vertical. Since many actual room layouts in floor plans are orthogonal and monotone, it is natural to represent them by orthogonal and monotone polygons in the plane.

We consider the following problem of reconstructing a polygon from a set of data points obtained from an orthogonal and monotone room layout. The interior of the room layout is connected. There is exactly one data point for each edge of the room layout and it lies in the interior of the edge. Each data point consists of its position (the x-coordinate and the y-coordinate) and the orientation of the edge (H for a horizontal edge and V for a vertical edge) where it lies. Your program is to compute the length of the boundary of the orthogonal and monotone polygon that can be reconstructed from input set of data points.

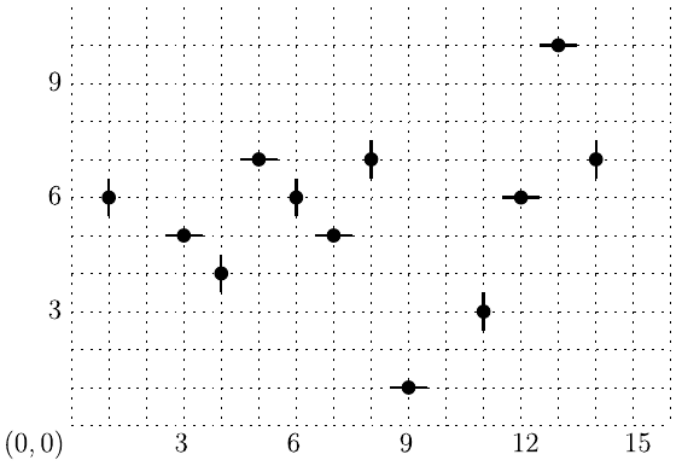


Figure 1. 12 data points in the plane: (9, 1, H), (3, 5, H), (6, 6, V), (5, 7, H), (14, 7, V), (11, 3, V), (7, 5, H), (13, 10, H), (12, 6, H), (1, 6, V), (4, 4, V), (8, 7, V).

For example, consider an input consisting of the following 12 data points as shown in Figure 1. Each data point is shown as a point with a short segment specifying the orientation of the edge where the point lies. Then, one can reconstruct an orthogonal and monotone polygon from the data points as shown in Figure 2. Note that there is no other polygon that can be reconstructed from the data points.

The polygon shown in Figure 2 has edges of lengths 2, 5, 2, 2, 5, 6, 4, 3, 5, 7, 4, 3, in clockwise order along the boundary from the leftmost vertical edge. Therefore, if these 12 data points of Figure 1 are given as input of your program, then your program must output 48 as the answer.

Your program is to compute the length of the boundary of the orthogonal and monotone polygon that can be reconstructed from an input set of data points.

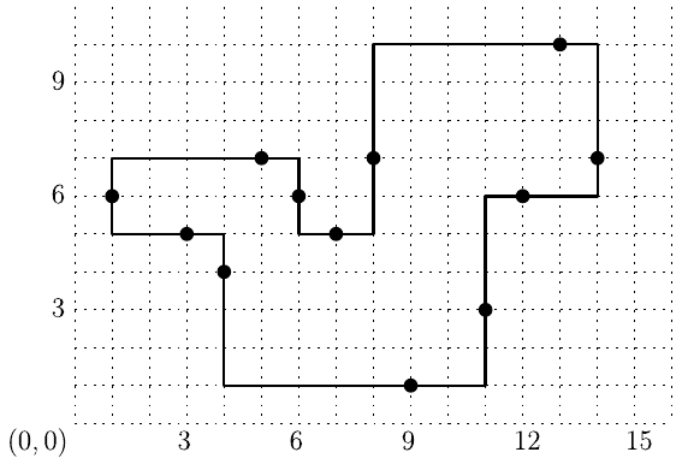


Figure 2. An orthogonal and monotone polygon reconstructed from the data points in Figure 1.

Input

The input file contains several test cases, each of them as described below.

Your program is to read from standard input. The first line contains an integer, n ($4 \leq n \leq 500,000$), where n is the number of data points. In the following n lines, the data points are given one by one. Each line contains two integers that are the x-coordinate and the y-coordinate of a data point followed

by an indicator of '0' or '1'. Each integer is between -2^{30} and $2^{30} - 1$, inclusively. The indicator is '0' if the edge where the data point lies is horizontal, and '1' otherwise.

Output

For each test case, the output must follow the description below.

Your program is to write to standard output. Print the length of the boundary of an orthogonal and monotone polygon that is reconstructed from the input. If there is no orthogonal and monotone polygon that can be reconstructed from input, print '-1'.

Sample Input

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

Sample Output

```
48
-1
36
```

7610 Rounding

Professor Park conducts and records the N data values from his experiment every day. The data values are recorded in real numbers, which rounded up to the first decimal place. The data of M days are shown on a two dimensional table with each row sum and each column sum. Every experimental data values including row sums and column sums, however, must be in integers to be published. Each number can be up or down to the nearest integer so that the sum of the rounded elements in each row (column) equals the row (column) sum. More formally, if x is a real number which represents an experimental data value or a row sum or a column sum, you can replace x by $\lfloor x \rfloor$ or $\lceil x \rceil$. If this operation, i.e. feasible rounding, is possible, the new table is called feasibly rounded table.

For examples, refer to the following tables:

	1	2	3	row sum
1	4.3	6.7	7.1	18.1
2	9.2	3.0	0.2	12.4
3	4.0	7.7	1.3	13.0
column sum	17.5	17.4	8.6	

(a) Original Table

	1	2	3	row sum
1	4	7	7	18
2	9	3	0	12
3	4	7	2	13
column sum	17	17	9	

(b) A feasibly rounded table

Figure 1. First example of the feasible rounding.

	1	2	3	row sum
1	0.4	0.4	0.4	1.2
2	0.5	0.5	0.5	1.5
column sum	0.9	0.9	0.9	

(a) Original Table

	1	2	3	row sum
1	0	0	1	1
2	1	1	0	2
column sum	1	1	1	

(b) A feasibly rounded table

Figure 2. Second example of the feasible rounding.

Given an original table, write a program that finds a feasibly rounded table.

Input

The input file contains several test cases, each of them as described below.

Your program is to read from standard input. The first line of the input contains two integers M ($2 \leq M \leq 200$) which represents the total experiment days and N ($2 \leq N \leq 200$) which represents the experimental data values for each day. In the following M lines, all experimental data values and row sums are given in real numbers to first decimal place. The i-th line consists of N real numbers which represent the experimental data values of the i-th day ($1 \leq i \leq M$) and the i-th row sum. The next line consists of N real numbers which represent column sums. The experimental data values are between 0.0 to 1,000.0, inclusively.

Output

For each test case, the output must follow the description below.

Your program is to write to standard output. Print a feasibly rounded table including each row sum and column sum. It is known that the feasibly rounded table can be obtained always.

Sample Input

```
3 3
4.3 6.7 7.1 18.1
9.2 3.0 0.2 12.4
4.0 7.7 1.3 13.0
17.5 17.4 8.6
2 3
0.4 0.4 0.4 1.2
0.5 0.5 0.5 1.5
0.9 0.9 0.9
```

Sample Output

```
4 7 7 18
9 3 0 12
4 7 2 13
17 17 9
0 0 1 1
1 1 0 2
1 1 1
```

7611 Virus

A malicious computer virus is to spread through a tree network. The network forms a binary tree and the infection or protection of nodes will be done as in the following process:

(1) At time 0, the virus infects the root.

(2) At each subsequent time step, only a single uninfected node can be protected by a computer vaccine. Also, at each subsequent time step, the virus spreads from infected nodes to all their unprotected child nodes. Once infected or protected, a node remains so during the whole process.

(3) When the virus can be no longer spread, the process ends.

In this problem, we would like to minimize the number of infected nodes when the process ends. For example, consider the network in Figure 1.

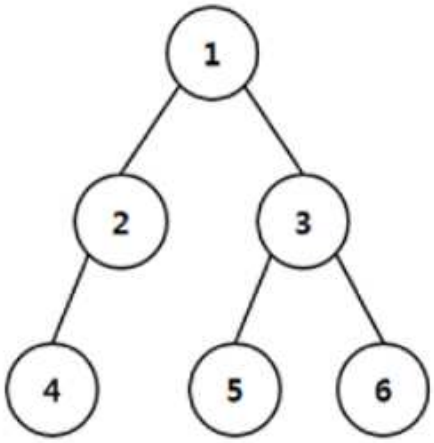


Figure 1. Illustration of a computer network.

At time 0, the virus infects the root of network, node 1. At time 1, suppose node 3 is protected, then the virus infects node 2. Subsequently, suppose node 4 is protected at time 2, then the virus can be no longer spread. When the process ends, the number of infected nodes is 2. If node 2 is protected instead of node 3 at time 1, the virus infects node 3 and the number of infected nodes cannot be smaller than 2. So, the minimum possible number of infected nodes is 2 in this example.

Given a network of binary tree, you are to write a program that computes the minimum possible number of infected nodes.

Input

The input file contains several test cases, each of them as described below.

Your program is to read from standard input. The input starts with a line containing a positive integer n ($1 \leq n < 2^{20}$) which represents the number of nodes in the network. Then follow n lines; the i-th line describes the node i with two integers representing the left child and the right child of the node. It is assumed that nodes are numbered from 1 to n. The root of network is node 1, and an empty node is represented by '0'.

Output

For each test case, the output must follow the description below.

Your program is to write to standard output. Print exactly one line which contains the minimum possible number of infected nodes.

Sample Input

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

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

Sample Output

```
2
2
2
```