Problem A. Adjustment Office

Input file: `adjustment.in`
Output file: `adjustment.out`

Garrison and Anderson are working in a company named "Adjustment Office". In competing companies workers change the reality, in this company they try to predict the future.

They are given a big square board $n \times n$. Initially in each cell (x, y) of this board the value of $x + y$ is written $(1 \le x, y \le n)$. They know that in the future there will be two types of queries on the board:

- "`R` r" — sum up all values in row r, print the result and set all values in row r to zero;
- "`C` c" — sum up all values in column c, print the result and set all values in column c to zero.

They have predicted what queries and results there will be. They need to ensure that they have correctly predicted the results. Help them by computing the results of the queries.

Input

The first line of the input contains two integers n and q $(1 \le n \le 10^6, 1 \le q \le 10^5)$ — the size of the square and the number of queries.

Each of the next q lines contains the description of the query. Each query is either "`R` r" $(1 \le r \le n)$ or "`C` c" $(1 \le c \le n)$.

Output

The output file shall contain q lines. The i-th line shall contain one integer — the result of the i-th query.

Sample input and output

adjustment.in	adjustment.out
3 7 R 2 C 3 R 2 R 1 C 2 C 1 R 3	12 10 0 5 5 4 0

Problem B. Binary vs Decimal

Input file: `binary.in`
Output file: `binary.out`

Bruce has recently got a job at NEERC (Numeric Expression Engineering & Research Center) facility, which studies and produces many kinds of curious numbers. His first assignment is to perform a study of bindecimal numbers.

A positive integer is called *bindecimal* if its decimal representation is a suffix of its binary representation; both binary and decimal representations are considered without leading zeros. For example, $10_{10} = 10\mathbf{10}_2$, thus, 10 is a bindecimal number. The numbers $1010_{10} = 111111\mathbf{0010}_2$ and $42_{10} = 1010\mathbf{10}_{10}$ are, evidently, not bindecimal.

First of all, Bruce is going to create a list of bindecimal numbers. Help him find the n-th smallest bindecimal number.

Input

The first and the only line contains one integer — n ($1 \le n \le 10\,000$).

Output

Print one integer — the n-th smallest bindecimal number in decimal notation.

Sample input and output

binary.in	binary.out
1	1
2	10
10	1100

Note

Here is a table with the first few numbers which contain only 0's and 1's in their decimal representation (it is clear that all other numbers are not bindecimal):

Decimal	Binary	Comment
1	**1**	1st bindecimal number
10	10**10**	2nd bindecimal number
11	10**11**	3rd bindecimal number
100	1100**100**	4th bindecimal number
101	1100**101**	5th bindecimal number
110	1101**110**	6th bindecimal number
111	1101**111**	7th bindecimal number
1000	111110**1000**	8th bindecimal number
1001	111110**1001**	9th bindecimal number
1010	111111**0010**	Not a bindecimal number
1011	111111**0011**	Not a bindecimal number
1100	1000100**1100**	10th bindecimal number

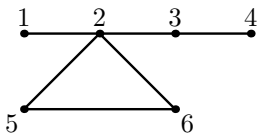
Problem C. Cactus Jubilee

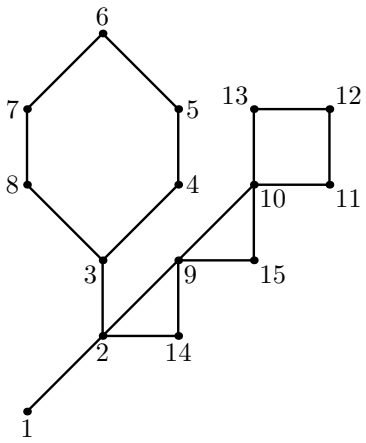
Input file: `cactus.in`
Output file: `cactus.out`

This is the 20-th Northeastern European Regional Contest (NEERC). Cactus problems had become a NEERC tradition. The first Cactus problem was given in 2005, so there is a jubilee — 10 years of Cactus.

Cactus is a connected undirected graph in which every edge lies on at most one simple cycle. Intuitively cactus is a generalization of a tree where some cycles are allowed. Multiedges (multiple edges between a pair of vertices) and loops (edges that connect a vertex to itself) are not allowed in a cactus.

You are given a cactus. Let's *move* an edge — remove one of graph's edges, but connect a different pair of vertices with an edge, so that a graph still remains a cactus. How many ways are there to perform such a move?





For example, in the left cactus above (given in the first sample), there are 42 ways to perform an edge move. In the right cactus (given in the second sample), which is the classical NEERC cactus since the original problem in 2005, there are 216 ways to perform a move.

Input

The first line of the input file contains two integers n and m ($1 \le n \le 50\,000$, $0 \le m \le 50\,000$). Here n is the number of vertices in the graph. Vertices are numbered from 1 to n. Edges of the graph are represented by a set of edge-distinct paths, where m is the number of such paths.

Each of the following m lines contains a path in the graph. A path starts with an integer k_i ($2 \le k_i \le 1000$) followed by k_i integers from 1 to n. These k_i integers represent vertices of a path. Adjacent vertices in a path are distinct. Path can go to the same vertex multiple times, but every edge is traversed exactly once in the whole input file.

The graph in the input file is a cactus.

Output

Write to the output file a single integer — the number of ways to move an edge in the cactus.

Sample input and output

cactus.in	cactus.out
6 1 7 1 2 5 6 2 3 4	42
15 3 9 1 2 3 4 5 6 7 8 3 7 2 9 10 11 12 13 10 5 2 14 9 15 10	216

Problem D. Distance on Triangulation

Input file: `distance.in`
Output file: `distance.out`

You have a convex polygon. The vertices of the polygon are successively numbered from 1 to n. You also have a triangulation of this polygon, given as a list of $n - 3$ diagonals.

You are also given q queries. Each query consists of two vertex indices. For each query, find the shortest distance between these two vertices, provided that you can move by the sides and by the given diagonals of the polygon, and the distance is measured as the total number of sides and diagonals you have traversed.

Input

The first line of the input file contains an integer n — the number of vertices of the polygon ($4 \le n \le 50\,000$).

Each of the following $n-3$ lines contains two integers a_i, b_i — the ends of the i-th diagonal ($1 \le a_i, b_i \le n$, $a_i \ne b_i$).

The next line contains an integer q — the number of queries ($1 \le q \le 100\,000$).

Each of the following q lines contains two integers x_i, y_i — the vertices in the i-th query ($1 \le x_i, y_i \le n$).

It is guaranteed that no diagonal coincides with a side of the polygon, and that no two diagonals coincide or intersect.

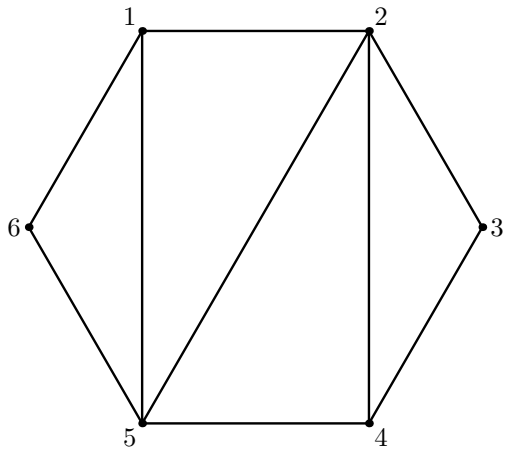
Output

For each query output a line containing the shortest distance.

Sample input and output

distance.in	distance.out
6 1 5 2 4 5 2 5 1 3 2 5 3 4 6 3 6 6	2 1 1 3 0

This is the polygon from the sample input.



Problem E. Easy Problemset

Input file: `easy.in`
Output file: `easy.out`

Perhaps one of the hardest problems of any ACM ICPC contest is to create a problemset with a reasonable number of easy problems. On Not Easy European Regional Contest this problem is solved as follows.

There are n jury members (judges). They are numbered from 1 to n. Judge number i had prepared p_i *easy problems* before the jury meeting. Each of these problems has a *hardness* between 0 and 49 (the higher the harder). Each judge also knows a very large (say infinite) number of *hard problems* (their hardness is 50). Judges need to select k problems to be used on the contest during this meeting.

They start to propose problems in the ascending order of judges numbers. The first judge takes the first problem from his list of remaining easy problems (or a hard problem, if he has already proposed all his easy problems) and proposes it. The proposed problem is selected for the contest **if its hardness is greater than or equal to the total hardness of the problems selected so far**, otherwise it is considered too easy. Then the second judge does the same etc.; after the n-th judge, the first one proposes his next problem, and so on. This procedure is stopped immediately when k problems are selected.

If all judges have proposed all their easy problems, but they still have selected less than k problems, then they take some hard problems to complete the problemset regardless of the total hardness.

Your task is to calculate the total hardness of the problemset created by the judges.

Input

The first line of the input file contains the number of judges n ($2 \le n \le 10$) and the number of problems k ($8 \le k \le 14$). The i-th of the following n lines contains the description of the problems prepared by the i-th judge. It starts with p_i ($1 \le p_i \le 10$) followed by p_i non negative integers between 0 and 49 — the hardnesses of the problems prepared by the i-th judge in the order they will be proposed.

Output

Output the only integer — the total hardness of the selected problems.

Sample input and output

easy.in	easy.out
3 8 5 0 3 12 1 10 4 1 1 23 20 4 1 5 17 49	94
3 10 2 1 3 1 1 2 2 5	354

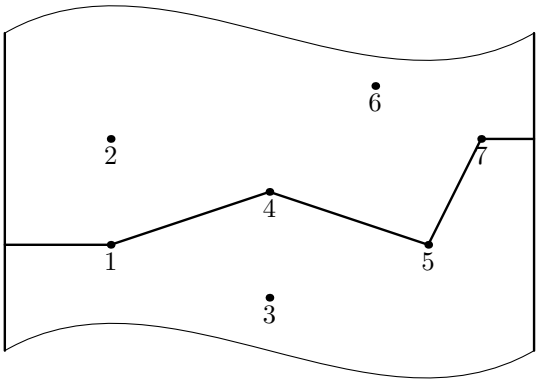
In the first example, three problems with hardnesses of 0, 1, and 1 are selected first. Then the first judge proposes the problem with hardness 3 and it is selected, too. The problem proposed by the second judge with hardness 1 is not selected, because it is too easy. Then the problems proposed by the third, the first, and the second judges are selected (their hardnesses are 5, 12 and 23). The following three proposed problems with hardness of 17, 1 and 20 are not selected, and the problemset is completed with a problem proposed by the third judge with hardness of 49. So the total hardness of the problemset is 94.

In the second example, three problems with hardnesses of 1, 1, and 2 are selected first. The second problem of the first judge (hardness 3) is too easy. The second judge is out of his easy problems, so he proposes a problem with hardness 50 and it is selected. The third judge's problem with hardness 5 is not selected. The judges decide to take 6 more hard problems to complete the problemset, which gives the total hardness of $54 + 6 \cdot 50 = 354$.

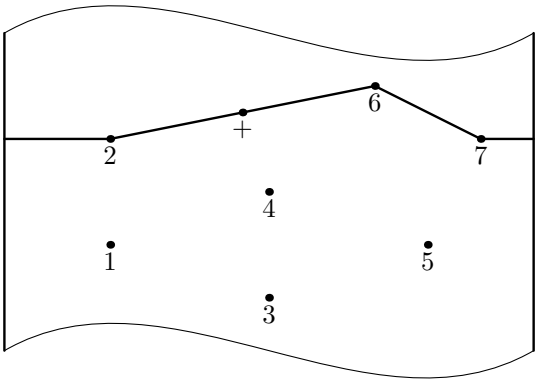
Problem F. Froggy Ford

Input file: `froggy.in`
Output file: `froggy.out`

Fiona designs a new computer game Froggy Ford. In this game, a player helps a frog to cross a river using stone fords. Frog leaps from the river's shore to the first stone ford, than to the second one and so on, until it reaches the other shore. Unfortunately, frog is pretty weak and its leap distance is quite limited. Thus, a player should choose the optimal route — the route that minimizes the largest leap required to traverse the route.



Optimal route



Optimal route with added stone

Fiona thinks that this game is not challenging enough, so she plans to add a possibility to place a new stone in the river. She asks you to write a program that determines such a location of the new stone that minimizes the largest leap required to traverse the optimal route.

Input

The first line of the input file contains two integers w — the width of the river and n — the number of stones in it ($1 \le w \le 10^9$, $0 \le n \le 1000$).

Each of the following n lines contains two integers x_i, y_i — the coordinates of the stones ($0 < x_i < w$, $-10^9 \le y_i \le 10^9$). Coordinates of all stones are distinct.

River shores have coordinates $x = 0$ and $x = w$.

Output

Write to the output file two real numbers x_+ and y_+ ($0 < x_+ < w$, $-10^9 \le y_+ \le 10^9$) — the coordinates of the stone to add. This stone shall minimize the largest leap required to traverse the optimal route. If a new stone cannot provide any improvement to the optimal route, then an arbitrary pair of x_+ and y_+ satisfying the constraints can be written, even coinciding with one of the existing stones.

Your answer shall be precise up to three digits after the decimal point.

Sample input and output

froggy.in	froggy.out
10 7 2 2 2 4 5 1 5 3 8 2 7 5 9 4	4.5 4.5

Problem G. Generators

Input file: `generators.in`
Output file: `generators.out`

Little Roman is studying *linear congruential generators* — one of the oldest and best known pseudo-random number generator algorithms. Linear congruential generator (LCG) starts with a non-negative integer number x_0 also known as *seed* and produces an infinite sequence of non-negative integer numbers x_i ($0 \le x_i < c$) which are given by the following recurrence relation:

$$x_{i+1} = (ax_i + b) \bmod c$$

here a, b, and c are non-negative integer numbers and $0 \le x_0 < c$.

Roman is curious about relations between sequences generated by different LCGs. In particular, he has n different LCGs with parameters $a^{(j)}$, $b^{(j)}$, and $c^{(j)}$ for $1 \le j \le n$, where the j-th LCG is generating a sequence $x_i^{(j)}$. He wants to pick one number from each of the sequences generated by each LCG so that the sum of the numbers is the maximum one, but is not divisible by the given integer number k.

Formally, Roman wants to find integer numbers $t_j \ge 0$ for $1 \le j \le n$ to maximize $s = \sum_{j=1}^{n} x_{t_j}^{(j)}$ subject to constraint that $s \bmod k \ne 0$.

Input

The first line of the input file contains two integer numbers n and k ($1 \le n \le 10\,000$, $1 \le k \le 10^9$). The following n lines describe LCGs. Each line contains four integer numbers $x_0^{(j)}$, $a^{(j)}$, $b^{(j)}$, and $c^{(j)}$ ($0 \le a^{(j)}, b^{(j)} \le 1000$, $0 \le x_0^{(j)} < c^{(j)} \le 1000$).

Output

If Roman's problem has a solution, then write on the first line of the output file a single integer s — the maximum sum not divisible by k, followed on the next line by n integer numbers t_j ($0 \le t_j \le 10^9$) specifying some solution with this sum.

Otherwise, write to the output file a single line with the number -1.

Sample input and output

generators.in	generators.out
2 3 1 1 1 6 2 4 0 5	8 4 1
2 2 0 7 2 8 2 5 0 6	-1

In the first example, one LCG is generating a sequence 1, 2, 3, 4, 5, 0, 1, 2, ..., while the other LCG a sequence 2, 3, 2, 3, 2, ...

In the second example, one LCG is generating a sequence 0, 2, 0, 2, 0, ..., while the other LCG a sequence 2, 4, 2, 4, 2, ...

Problem H. Hypercube

Input file: `hypercube.in`
Output file: `hypercube.out`

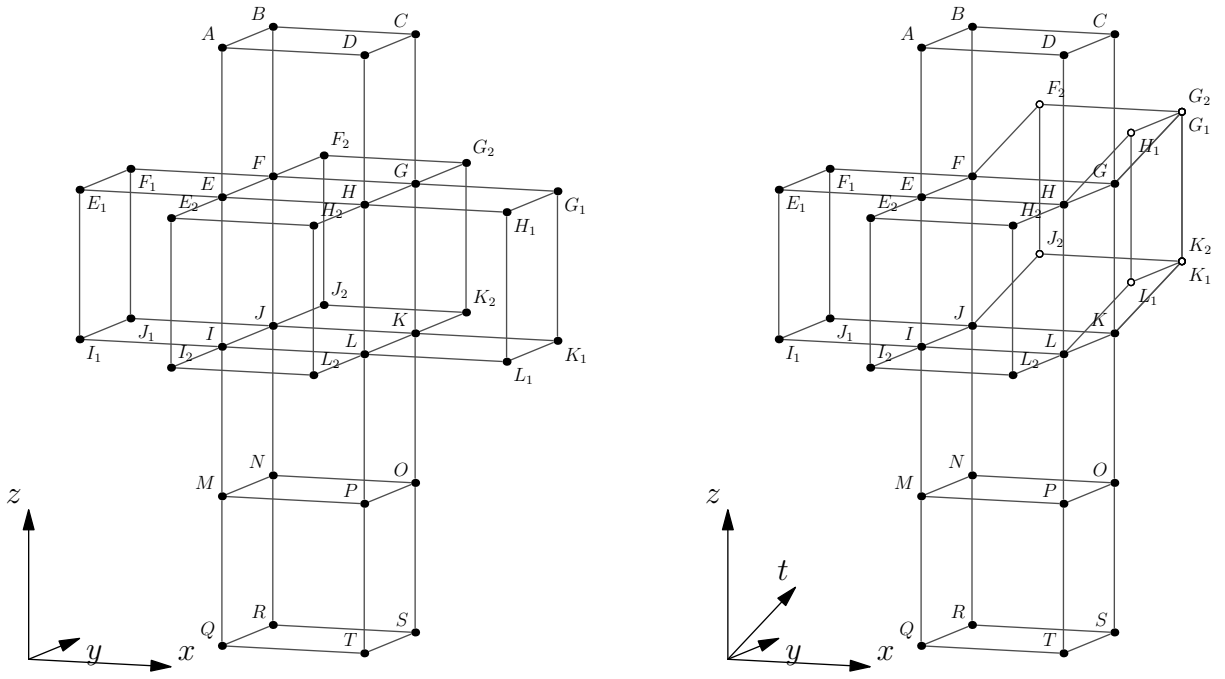
Consider a 4-hypercube also known as tesseract. A unit *solid tesseract* is a 4D figure that is equal to the convex hull of 16 points with Cartesian coordinates $(\pm\frac{1}{2}, \pm\frac{1}{2}, \pm\frac{1}{2}, \pm\frac{1}{2})$ — its vertices. It has 32 edges (1D), 24 square faces (2D), and 8 cubic 3-faces (3D) also known as *cells*. We study hollow tesseracts and define a *tesseract* as a boundary of a solid tesseract. Thus, a tesseract is a connected union of 8 solid cubes (its cells) that intersect between each other at 24 tesseract's square faces, 32 edges, and 16 vertices.

Let's cut a tesseract along 17 of its 24 faces, so that it still remains connected via 7 faces that were left intact. Unfold the tesseract into a 3D hyperplane by rotating its constituting cubes along the faces that were left intact until all its cells lie in the same 3D hyperplane. The result is called a *3-net* of a tesseract. This process is a natural generalization of how a 3D cube is cut and unfolded onto a 2D plane to produce a 2-net of a cube that consists of 6 squares.

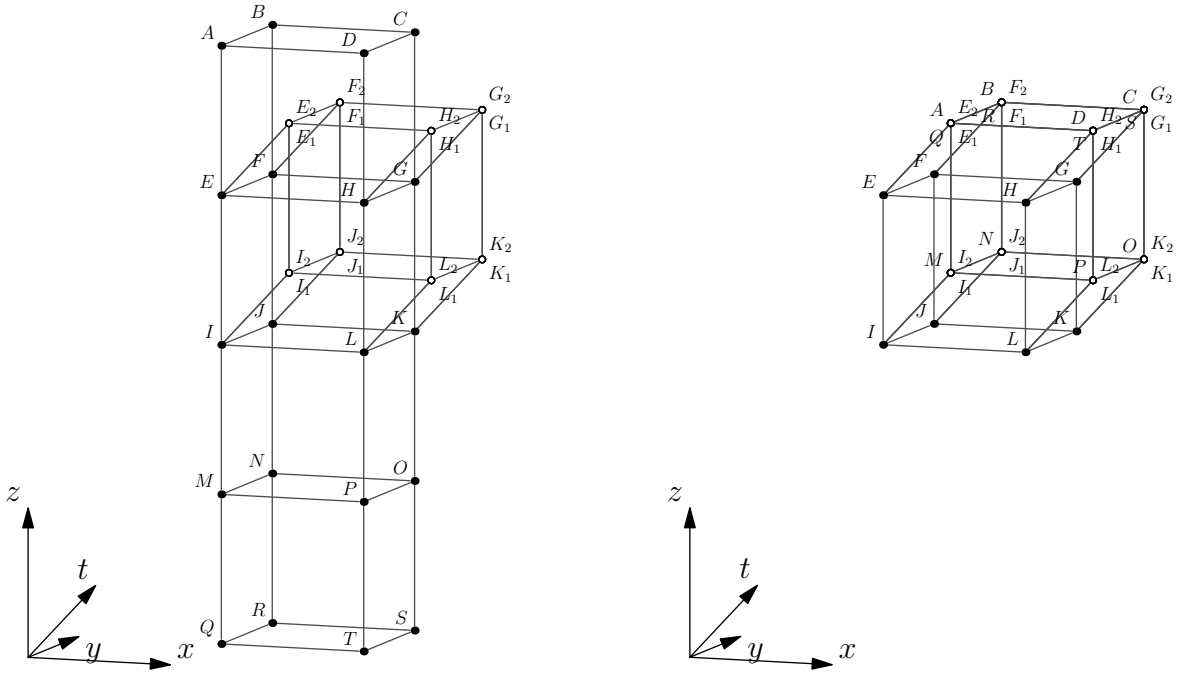
In this problem you are given a tree-like 8-polycube in 3D space also known as *octocube*. An octocube is a collection of 8 unit cubical cells joined face-to-face. More formally, intersection of each pair of cubical cells constituting an octocube is either empty, a point, a unit line (1D), or a unit square (2D). The given octocube is tree-like in the following sense. Consider an *adjacency graph* of the octocube — a graph with 8 vertices corresponding to its 8 cells. There is an edge in the adjacency graph between pairs of adjacent cells. Two cells of an octocube are called *adjacent* when their intersection is a square. Cells that intersect at a point or a line are not considered adjacent. An octocube is called *tree-like* when its adjacency graph is a tree.

Your task is to determine whether the given tree-like octocube constitutes a 3-net of a tesseract. That is, whether this octocube being put onto a hyperplane in 4D space can be folded in 4D space along the squares of intersection between its cells into a tesseract.

For example, look at the leftmost picture below. It shows a wire-frame of the tree-like octocube. Rotate cell $GHLKG_1H_1L_1K_1$ around a plane $GHLK$ and cell $FGKJF_2G_2K_2J_2$ around a plane $FGKJ$ at angle 90 degrees in 4-th dimension outside of the original hyperplane. As a result, point G_1 joins with G_2 and K_1 joins with K_2. The face GKK_2G_2 is glued to face GKK_1G_1. The result is shown on the right. The 4-th dimension is orthographically projected onto the 3 shown in perspective. The points that have moved out of the original hyperplane are marked with hollow dots.



Rotate $EFJIE_1F_1J_1I_1$ around $EFJI$ and $EHLIE_2H_2L_2I_2$ around $EHLI$. The result is shown on the following picture on the left. The remaining steps are as follows. Rotate $MNOPQRST$ around $MNOP$, then rotate both $MNOPQRST$ and $IJKLMNOP$ around $IJKL$ and rotate $ABCDEFGH$ around $EFGH$. The last step is to glue all faces that meet together to get a tesseract that is shown on the right.



Input

The first line of the input file contains there integers m, n, k — the width, the depth, and the height of the box that contains the given octocube ($1 \le m, n, k \le 8$). The following k groups of lines describe rectangular slices of the box from top to bottom. Each slice is described by n rows with m characters each. The characters on a line are either '`.`', denoting an empty space, or '`x`', denoting a unit cube. The input file is guaranteed to describe a tree-like octocube.

Output

Write to the output file a single word "Yes" if the given octocube can be folded into a tesseract or "No" otherwise.

Sample input and output

hypercube.in	hypercube.out
3 3 4 ... .x. ... .x. xxx .x. ... .x. ... ... .x. ...	Yes
8 1 1 xxxxxxxx	No

Problem I. Iceberg Orders

Input file: `iceberg.in`
Output file: `iceberg.out`

You are working for Metagonia stock exchange. Recently traders in Metagonia had heard about Iceberg orders traded on London stock exchange and asked your employer to add such functionality as well. A *stock exchange* is an engine that receives *orders* and generates *trades*.

An *iceberg order* is a quintuple of integers (ID, T, P, V, TV). Each order has an identifier ID (unique among all orders), type T (which is equal to either $BUY = 1$ or $SELL = 2$), price P, total remaining volume V and tip volume TV. For each order, exchange additionally keeps track of its current volume CV and priority PR. There is also a global priority counter of the exchange GP. An *order book* of the exchange is a set of orders.

Trades that are generated by the exchange are quadruples of integers (BUY_ID, $SELL\_ID$, P, V). Each trade has BUY_ID and $SELL\_ID$ — identifiers of matching BUY and $SELL$ orders, trade price P, and trade volume V.

When an order is received by the exchange it is matched against orders currently on the order book. This is done as follows. Suppose an order a is received with $T_a = SELL$. Among all orders currently on the order book we look for an order b such that $T_b = BUY$ and $P_b \geq P_a$. We select such an order b with the largest price, and if there are several — one with the smallest priority. If there is such an order b, then a trade t is generated with $BUY\_ID_t = ID_b$ and $SELL\_ID_t = ID_a$ at trade price $P_t = P_b$ with trade volume $V_t = \min(V_a, CV_b)$. V_a, V_b, and CV_b are all decreased by trade volume. If $V_b = 0$ after this, then the order b is removed from the order book. If $CV_b = 0$ (but $V_b > 0$) then we set current volume of order b to $CV_b = \min(V_b, TV_b)$, set $PR_b = GP$, and increment GP. We continue these operations of selecting b and generating trades until either $V_a = 0$ or there are no more orders b on the order book which satisfy the condition. In the latter case, we add order a to the order book with $CV_a = \min(V_a, TV_a)$ and $PR_a = GP$, and then increment GP. When the process of matching the order a is finished with several trades between the same pair of orders a and b (and there can be lots of them!), they are all united into a single trade with the volume equal to the sum of individual trade volumes.

If $T_a = BUY$ we are looking for an order b with $T_b = SELL$ and $P_b \leq P_a$ and select such an order b with the smallest price and the smallest priority among those. The rest of the matching process is as described above, with trades having $BUY\_ID_t = ID_a$, $SELL\_ID_t = ID_b$, $P_t = P_b$, and $V_t = \min(V_a, CV_b)$.

Initially the order book is empty. You are presented with several orders that are received by the exchange one by one. You need to print generated trades and the order book state after all orders are processed.

Hint: The priority and GP are introduced in the problem statement only for the purpose of a formal description of the algorithm. The actual implementation does not have to keep track of priority. Typically, an exchange simply keeps a priority-ordered list of orders of each type at each price in its order book.

Input

The first line of the input contains the number of orders n ($1 \leq n \leq 50\,000$). Each of the following n lines represent an order. Each order is given by a space-separated quintuple $ID\ T\ P\ V\ TV$, where $1 \leq ID \leq 1\,000\,000$, $T = 1$ for BUY and $T = 2$ for $SELL$, $1 \leq P \leq 100\,000$ and $1 \leq TV \leq V \leq 1\,000\,000\,000$.

Output

For each order print all trades generated by processing this order, in ascending order of pairs (BUY_ID, $SELL\_ID$), each trade on its own line. Each trade shall be printed as a space-separated quadruple of integers $BUY\_ID\ SELL\_ID\ P\ V$. It is guaranteed that the total number of trades would not exceed 100 000. Print a blank line after all trades, followed by the order book. Each order that is still on the book shall be printed as a space-separated **sextuple** $ID\ T\ P\ V\ TV\ CV$, sorted first by P and then by PR.

Sample input and output

iceberg.in	iceberg.out
7 42 1 100 200 20 239 1 100 50 50 1111 1 101 30 15 1234 1 100 300 15 4321 2 99 125 25 5678 1 101 30 30 8765 2 101 100 20	42 4321 100 30 239 4321 100 50 1111 4321 101 30 1234 4321 100 15 5678 8765 101 30 42 1 100 170 20 10 1234 1 100 285 15 15 8765 2 101 70 20 20

In the sample input the first four orders have $T = BUY$. Assuming that at the beginning GP at the exchange was equal to 1, after receiving these orders the order book looks in the following way when ordered according to the matching rules from the problem statement for $T_b = BUY$ orders (first by the largest price, then by the smallest priority):

ID	T	P	V	TV	CV	PR
1111	1	101	30	15	15	3
42	1	100	200	20	20	1
239	1	100	50	50	50	2
1234	1	100	300	15	15	4

The fifth order (with $ID_a = 4321$) has $T_a = SELL$, $P_a = 99$, and $V_a = 125$ and is eligible for match with all of the above four orders given in the above table. First, it matches twice with the order 1111 with the highest price of 101 for a total trade volume of 30, removes it from the order book, bumps GP to 6 in the process, and decreases V_a to 95. Then, there are three other orders at price 100. One matching pass through them produces three trades for a total volume of 85 (volume 20 with order 42, volume 50 with order 239, removes it from the book, volume 15 with order 1234), bumps GP to 8, and decreases V_a to 10. The remaining orders in the book are shown below:

ID	T	P	V	TV	CV	PR
42	1	100	180	20	20	6
1234	1	100	275	15	15	7

One last match with the order 42 produces a trade with a volume of 10 (for a total volume of 30 for matches with order 42) and the order 4321 is done ($V_a = 0$). Four corresponding total trades for order 4321 are printed in the sample output. The remaining order book is:

ID	T	P	V	TV	CV	PR
42	1	100	180	20	10	6
1234	1	100	275	15	15	7

The sixth BUY order (with $ID = 5678$) is added to the order book (GP becomes 9):

ID	T	P	V	TV	CV	PR
5678	1	101	30	30	30	8
42	1	100	180	20	10	6
1234	1	100	275	15	15	7

The last, seventh order (with $ID = 8765$), can be matched only with the order 5678 due to price condition, generates a trade with volume of 30, order 5678 is removed from the order book, while order 8765 is added. Now the order book has both BUY and $SELL$ orders:

ID	T	P	V	TV	CV	PR
42	1	100	180	20	10	6
1234	1	100	275	15	15	7
8765	2	101	70	20	20	9

Problem J. Jump

Input file: standard input
Output file: standard output

Consider a toy interactive problem OneMax which is defined as follows. You know an integer n and there is a hidden bit string S of length n. The only thing you may do is to present the system a bit string Q of length n, and the system will return the number OneMax(Q) — the number of bits which coincide in Q and S at the corresponding positions. The name of OneMax problem stems from the fact that this problem is simpler to explain when $S = 111\ldots11$, so that the problem turns into maximization (Max) of the number of ones (One).

When n is even, there is a similar (but harder) interactive problem called Jump. The simplest way to describe the Jump is by using OneMax:

$$\textsc{Jump}(Q) = \begin{cases} \textsc{OneMax}(Q) & \text{if } \textsc{OneMax}(Q) = n \text{ or } \textsc{OneMax}(Q) = n/2; \\ 0 & \text{otherwise.} \end{cases}$$

Basically, the only nonzero values of OneMax which you can see with Jump are n (which means you've found the hidden string S) and $n/2$.

Given an even integer n — the problem size, you have to solve the Jump problem for the hidden string S by making interactive Jump queries. Your task is to eventually make a query Q such that $Q = S$.

Interaction protocol

First, the testing system tells the length of the bit string n. Then, your solution asks the queries and the system answers them as given by the Jump definition. When a solution asks the query Q such that $Q = S$, the system answers n and terminates, so if your solution, after reading the answer n, tries reading or writing anything, it will fail.

The limit on the number of queries is $n + 500$. If your solution asks a $(n + 501)$-th query, then you will receive the "Wrong Answer" outcome. You will also receive this outcome if your solution terminates too early.

If your query contains wrong characters (neither `0`, nor `1`), or has a wrong length (not equal to n), the system will terminate the testing and you will receive the "Presentation Error" outcome.

You will receive the "Time Limit Exceeded" outcome and other errors for the usual violations.

Finally, if everything is OK (e.g. your solution finds the hidden string) on every test, you will receive the "Accepted" outcome, in this case you will have solved the problem.

Input

The first line of the input stream contains an even number n ($2 \le n \le 1000$). The next lines of the input stream consist of the answers to the corresponding queries. Each answer is an integer — either 0, $n/2$, or n. Each answer is on its own line.

Output

To make a query, print a line which contains a string of length n which consists of characters `0` and `1` only. Don't forget to put a newline character and to flush the output stream after you print your query.

Sample input and output

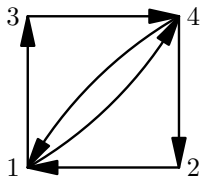
standard input	standard output
2 1 0 1 2	01 11 10 00

Problem K. King's Inspection

Input file: `king.in`
Output file: `king.out`

King Karl is a responsible and diligent ruler. Each year he travels across his country to make certain that all cities are doing well.

There are n cities in his country and m roads. In order to control the travelers, each road is unidirectional, that is a road from city a to city b can not be passed from b to a.



Karl wants to travel along the roads in such a way that he starts in the capital, visits every non-capital city exactly once, and finishes in the capital again.

As a transport minister, you are obliged to find such a route, or to determine that such a route doesn't exist.

Input

The first line contains two integers n and m ($2 \le n \le 100\,000$, $0 \le m \le n + 20$) — the number of cities and the number of roads in the country.

Each of the next m lines contains two integers a_i and b_i ($1 \le a_i, b_i \le n$), meaning that there is a one-way road from city a_i to city b_i. Cities are numbered from 1 to n. The capital is numbered as 1.

Output

If there is a route that passes through each non-capital city exactly once, starting and finishing in the capital, then output $n + 1$ space-separated integers — a list of cities along the route. Do output the capital city both in the beginning and in the end of the route.

If there is no desired route, output "`There is no route, Karl!`" (without quotation marks).

Sample input and output

king.in	king.out
4 6 1 4 4 1 4 2 2 1 3 4 1 3	1 3 4 2 1
4 3 1 4 1 4 2 2	There is no route, Karl!

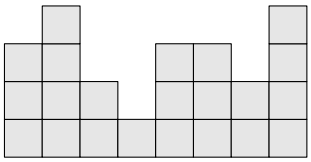
Problem L. Landscape Improved

Input file: `landscape.in`
Output file: `landscape.out`

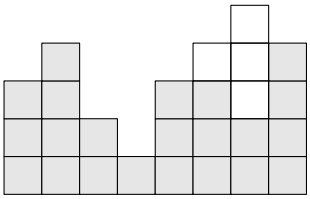
Louis L Le Roi-Univers has ordered to improve the landscape that is seen from the royal palace. His Majesty prefers to see a high mountain.

The Chief Landscape Manager is going to raise a mountain for Louis. He represents a landscape as a flat picture on a grid of unit squares. Some of the squares are already filled with rock, while others are empty. This greatly simplifies the design. Unit squares are small enough, and the landscape seems to be smooth from the royal palace.

The Chief Landscape Manager has a plan of the landscape — the heights of all rock-filled columns for each unit of width. He is going to add at most n square units of stones atop of the existing landscape to make a mountain with as high peak as possible. Unfortunately, piles of stones are quite unstable. A unit square of stones may be placed only exactly on top of the other filled square of stones or rock, moreover the squares immediately to the bottom-left and to bottom-right of it should be already filled.



Existing landscape



Improved landscape

Your task is to help The Chief Landscape Manager to determine the maximum height of the highest mountain he can build.

Input

The first line of the input file contains two integers w — the width of the existing landscape and n — the maximum number of squares of stones to add ($1 \le w \le 100\,000$, $0 \le n \le 10^{18}$).

Each of the following w lines contains a single integer h_i — the height of the existing landscape column ($1 \le h_i \le 10^9$).

Output

The output file shall contain the single integer — the maximum possible landscape height after at most n unit squares of stones are added in a stable way.

Sample input and output

landscape.in	landscape.out
8 4 3 4 2 1 3 3 2 4	5
3 100 3 3 3	4