Problem A. Tea

Input file:	`tea.in`
Output file:	`tea.out`
Time limit:	2 seconds
Memory limit:	64 megabytes

In numerous departments of some huge company there are n people working. They enjoy drinking tea during the break. But they are disciplined therefore they have only one break daily. To make this break as pleasant as possible each of the employees drinks tea with one of their favorite tastes. An employee could drink tea with different tastes at different days. Varietes of tea are numbered from 1 to m for convenience.

Recently, the employees of the department bought a large set of teabags. This set contains a_1 teabags of the first variety, a_2 teabags of the second variety, …, and a_m teabags of the m-th variety. Now they want to know, how long will they have enough teabags, so that each of the employees could drink some tea of his favorite variety each day?

Each employee in the department drinks one cup of tea that brews one teabag. Teabags are not brewed again.

Input

The first line of the input file contains two integers n, m ($1 \le n, m \le 50$). The second line contains m integers $a_1, \ldots, a_m$ ($1 \le a_i \le 10^6$ for all i from 1 to m). Each of the following n lines contains description of favorite tastes of employees. The i-th line describes i-th employee in the following format: positive integer k_i — number of favorite varietes following k_i integers from 1 to m — numbers of this varietes.

Output

The sole line of the output should contain the maximum number of days during which employees will have enough sufficient number of bags.

Example

tea.in	tea.out

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

Problem B. Domino Tiling

Input file:	`dominoes.in`
Output file:	`dominoes.out`
Time limit:	2 seconds
Memory limit:	64 megabytes

Given a field of a size $n \times m$. Some of the cells of this field are busy, others are *free*. One can put *dominoes* on the field. Dominoes are pieces composed of several squares, such that each square has the size equal to the size of a field cell. A domino can be put on the field only in such a way that each square from the domino fits perfectly in a certain cell of the field.

In this problem, you have only two types of dominoes. A domino of type 1 has the size of 1×2 and costs a euro cents. A domino of type 2 has the size of 1×1 and costs b euro cents. You need to tile the field using the smallest sum of money possible.

To *tile* a field by dominoes means to put several dominoes on this field such that every square of every domino lies on some *free* field cell and every *free* field cell is covered by exactly one domino square.

You can rotate 1×2 dominoes: that is, it can be considered as either having width of 1 and height of 2, or width of 2 and height of 1.

Input

The first line of the input file contains four numbers n, m, a, b ($1 \le n, m \le 300$, a, b are integers not exceeding 1000 by the absolute value). Each of the next n lines contain m symbols: a dot "`.`" denotes a free cell, an asterisk "`*`" denotes a busy cell.

Output

Output a single number: the minimum amount of money which is needed to tile the given field.

Example

dominoes.in	dominoes.out
2 3 3 2 .** .*.	5

Problem C. Experimental treatment

Input file:	`experimental.in`
Output file:	`experimental.out`
Time limit:	2 seconds
Memory limit:	64 megabytes

After a lot of unsuccessful tries to diagnose desease of new patient, MD House decided to try new experimental treatment. During the treatment each hour Forman offered patient to choose and drink one of the two pills. It is known, that right after the n-th pill was chosen, the patient suddenly recovers. The patient remembers how many pills of each type he drinks, and Forman knows all the pairs of pills, which he offered throughout the time. Because you need to know, what pills are helped, House wants to reestablish the type of each pill, chosen by the patient. Please, help him.

Input

The first line contains the number of pills, chosen by the patient, n and the number of different types of pills, which are in the hospital, m ($1 \le n \le 1000, 2 \le m \le 1000$). Each i-th line, starting from the second till $(n+1)$-th, contains the pair of integers a_i, b_i ($1 \le a_i, b_i \le m$, $a_i \ne b_i$) — the ids of the types of pills, which were offered by Forman at $(i-1)$-th hour. The last line contains m numbers c_j — the number of chosen pills of the type j ($0 \le c_j \le n$). The types are numerated from 1.

Output

Output the sequence of n numbers, where i-th number is equal to the type of the pill, chosen at i-th hour. If there are multiple answers, output any. If there are no answers, print -1.

Example

experimental.in	experimental.out

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

3 3
1 2
1 3
2 3
1 1 0
	-1

Problem D. Maximum flow

Input file:	`maxflow.in`
Output file:	`maxflow.out`
Time limit:	2 second
Memory limit:	64 megabytes

An oriented graph is given where each edge has an integer capacity. Find the maximum flow from the vertex 1 to the vertex n.

Input

The first line of the input file contains integers n and m – the number of vertices and the number of edges, correspondingly ($2 \le n \le 100$, $1 \le m \le 1000$). The next m lines contain three non-negative integers each describing an edge – the source, the target and the capacity of the edge. The vertices are numbered starting from 1, and the capacities do not exceed 10^5.

Output

Output a single number – the maximum flow from the vertex 1 to the vertex n.

Examples

maxflow.in	maxflow.out

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

Problem E. Evacuation

Input file:	`evacuation.in`
Output file:	`evacuation.out`
Time limit:	2 seconds
Memory limit:	64 megabytes

In T minutes the army of Loky will attack Earth. Avengers can't have time to prevent portals in New-York from opening, so Captain America decided to evacuate all citizens He only need to check, if everybody could be in safe position before the start of invasion.

The surroundings of New-York could be represened as the number of small cities, connected between themselfs with one-directional roads. Each road is characterized by its length and capacity. The length of the ro l means, that if you enter it at time t, the car will be at the end l minutes, in $t + l$. The capacity of road s means, that each minute, at most s cars could enter it. When you get to the city, each car could enter some other road or it could stop there for any time, and then move again.

Captain America have already decided, in which city the citizens need to be. Also, he knows, how many cars there are in the city. Now, he needs to check, if all citizens could evacuate before invasion, and if so, the minimal time which evacuation could take, and the minimal number of cars, which will be late, otherwise.

Input

The first line contains four integers n, m, K and T ($1 \le n \times T \le 10\,000$, $1 \le m, K \le 10\,000$) — the number of cities around New-York, the number of roads, the number of cars and the time to the invasion, respectively. Next m lines contain the description of roads between cities.

Each road is described with four integers u, v, l s ($1 \le u, v \le n$, $u \ne v$, $1 \le s \le 3\,000$, $1 \le l \le 200$) — in-point, out-point, its length and capacity, respectively.

There could be only at most one road between two cities. New-York has index 1, and the safe city has index n. In time 0 all cars are in New-York.

Output

If all citizen could get to the safe city in no bigger than T minutes, print the minimal number of minutes, to achieve that. Otherwise, print the minimal number of cars, which will be late.

Example

evacuation.in	evacuation.out

5 5 10 10
1 2 2 2
2 3 1 1
2 4 1 1
4 5 2 4
3 5 2 4
	9

Problem F. Path Covering

Input file:	`paths.in`
Output file:	`paths.out`
Time limit:	2 seconds
Memory limit:	64 megabytes

Given an oriented acyclic graph. Determine the minimum number of non-intersecting paths which cover all vertices.

Two paths are non-intersecting if they don't have common vertices.

Input

The first line of the input file contains n and m, the number of vertices and the number of edges of the graph, correspondingly ($2 \le n \le 1000$, $0 \le m \le 10^5$). The next m lines contain two integers each describing an edge: the source and the target vertices of the edge.

Output

Output a single number k, the minimum number of non-intersecting paths to cover all vertices.

Examples

paths.in	paths.out

3 3
1 3
3 2
1 2
	1

Problem G. Molecule

Input file:	`molecule.in`
Output file:	`molecule.out`
Time limit:	1 second
Memory limit:	64 megabytes

Arthur and Leonard play in the following game. In some cells of rectangle Arthur draws one of the chemical elements '`H`', '`O`', '`N`' and '`C`', after that Leonard need to draw lines between symbols in neighbouring cells, such that he will get correct molecules. Unfortunately, Arthur likes drawing a huge amount of symbols, while Leonard can't even understand whether it is possible to draw the lines in described way. Help him to answer on that question.

In this problem the lines between chemical elements create a correct molecule, if they satisfy following conditions:

- each line connect elements, written in cells, which are neighbours by side;
- between each pair of elements there is at most one line;
- each element should have exactly the defined number of adjacent lines (1 for H, 2 for O, 3 for N, 4 for C);
- empty cells don't have adjacent lines; and
- at least one cell contains an element inside.

Input

The first line contains two integer numbers n and m ($1 \le n, m \le 50$) – the sizes of the rectangle. Then n lines follows, with exactly m symbols in each, defining the distribution of elements in the rectangle; empty cells are denoted by symbol "`.`".

Output

The sole line of the output should contain one word "`Valid`", if it possible to draw lines with described conditions, and "`Invalid`" otherwise.

Example

molecule.in	molecule.out

`3 4`
`HOH.`
`NCOH`
`OO..`
	`Valid`

`3 4`
`HOH.`
`NCOH`
`OONH`
	`Invalid`

Problem H. Agrarian Reform

Input file:	`agrarian.in`
Output file:	`agrarian.out`
Time limit:	2 seconds
Memory limit:	256 megabytes

The King of Squaredom is planning the agrarian reform. The Squaredom has the form of rectangle of $m \times n$ squares. Squares are identified by pairs (x, y) where x ranges from 1 to m, and y ranges from 1 to n. Each square is either occupied by a peasant's house, or contains a swamp, or is a field. The King would like to assign peasants to fields, so that each peasant was assigned to exactly one field, and each field was assigned as most one peasant.

The King asked his Minister of Agronomy to prepare the list of peasants. After that he would assign them to fields. The Private Counselor of the King has found out the algorithm the King will use to assign peasants to fields.

The King would look through the peasants in order they are listed by the Minister of Agronomy. For each peasant he would find the closest to his house field that has no peasant assigned to it yet. That field would be assigned to this peasant. If there are several such fields, the field which has the smallest x will be chosen, if there are still several such fields, the field which has the smallest y among them will be chosen. The distance between squares (x_1, y_1) and (x_2, y_2) is $|x_1 - x_2| + |y_1 - y_2|$.

The Minister of Agronomy would like to order peasants in such a way that the sum of distances between peasant and the field he is assigned to for all peasants were as small as possible. Help him to find such order.

Input

The first line of the input file contains four integer numbers: m, n, k and s — the size of the field, the number of peasants, and the number of swamps, respectively ($1 \le m, n \le 20$, $1 \le k \le mn/2$, $0 \le s \le mn - 2k$). The following k lines contain coordinates of squares where peasants live, the i-th of these lines contains two integer numbers x_i, y_i ($1 \le x_i \le m$, $1 \le y_i \le n$). No two peasants live in the same square.

The following s lines contain coordinates of squares containing swamps.

Output

Output k numbers — the order in which the Minister of Agronomy should order peasants so that the King assigned them to the fields in the optimal way.

Example

agrarian.in	agrarian.out

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

Problem I. Teams

Input file:	`teams.in`
Output file:	`teams.out`
Time limit:	2 seconds
Memory limit:	64 megabytes

The main contest of "SWEERC" is coming soon. Teams came from n different universities; from each university exactly two teams came. The team already have taken place, when the organizers was discovered that some teams from the same university sit very close to each other. The organizers have serious problems with rearranging participants of the olympiad.

Tables at which teams sit are placed in a row. The distance between two adjacent tables equals 10 meters. The organizers want to make the minimal distance between workplaces of two teams from the same school as bigger as possible.

The organizers should move to a new location all the equipment of a team. Therefore the organizers want to move team in a way that the sum of distances between old and new workplaces of teams is as small as possible.

For example, consider two teams from universities $1, 2, 3$ and 4 are involved into competition. Consider the initial distribution is $1, 3, 2, 2, 1, 4, 4, 3$ (for each workplace the id of the university of the team is specified). In this case the teams from university

2 sit too close, as well as the teams from university 4. If jury rearranges teams in the following order $1, 3, 2, 4, 1, 3, 2, 4$, the distance between two teams from the same school is not less than 40 meters. The greater distance could not be achieved.

For this example the sum of the distances between the old and new positions of all teams is $0 + 0 + 0 + 20 + 0 + 20 + 30 + 10 = 80$ meteres, it is minimal for the initial location of the teams.

Given the initial location of the teams your task is rearrange teams in a way to maximize the minimal distance between teams from the same school. Moreover, new arrangement of teams should be chosen from all possible arrangements, such that the sum of distances between old and new positions is minimal as possible.

Input

The first line of the input file contains a single number n — the number of teams ($1 \leq n \leq 100$). The second line contains the sequence $a_1, a_2, \ldots, a_{2n}$ — the initial location of teams. For each team the number of schools which team represents is specified. It is guaranteed that sequence contains only numbers from 1 to n and each number occurs exactly twice.

Output

The sole line of the output should contain: how to rearrange teams to maximize the minimal distance between teams from the same school. Moreover, new arrangenment of teams should be chosen from the all possible arrangements, such that the sum of distances between old and new positions is minimal as possible. If there are several optimal solutions, output any of them.

Example

`teams.in`	`teams.out`

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

Problem J. Minimum cost maximum flow

Input file:	`mincost.in`
Output file:	`mincost.out`
Time limit:	2 seconds
Memory limit:	64 megabytes

An oriented graph is given, where each edge has a capacity and a cost. Find the minimum cost maximum flow from the vertex 1 to the vertex n.

Input

The first line of the input file contains n and m, the number of vertices and the number of edges correspondingly ($2 \leq n \leq 100$, $1 \leq m \leq 1000$). The next m lines contain four non-negative integers each which describe an edge: the source vertex, the target vertex, the capacity and the cost. The vertices are numbered starting from 1. Capacities and costs do not exceed 10^5.

Output

Output the single number – the cost of the minimum cost maximum flow from the vertex 1 to the vertex n. It is guaranteed that the answer does not exceed $2^{63} - 1$, and the graph does not contain cycles with negative costs.

Examples

`mincost.in`	`mincost.out`

`4 5`
`1 2 1 2`
`1 3 2 2`
`3 2 1 1`
`2 4 2 1`
`3 4 2 3`
	`12`

Problem K. The flow in network

Input file:	`flow.in`
Output file:	`flow.out`
Time limit:	2 seconds
Memory limit:	64 megabytes

In many problems on graph theory, the concepts of *network* and *flow* are widely used,

Network is the directed graph $G = (V, E)$, where for each edge $(u, v) \in V$ the value $c(u, v) \geq 0$ is given, named *capacity* of the edge. In the case $(u, v) \notin V$ it is convenient to think that $c(u, v) = 0$. In the network there are two special vertices: *source* s and *sink* t.

The *flow* in network G is the function $f : V \times V \to \mathcal{R}$, for which three propery hold:

- *Capacity constraint* — $\forall u, v \in V \; f(u, v) \leq c(u, v)$;
- *Antisymmetry* — $\forall u, v \in V \; f(u, v) = -f(v, u)$;
- *Conservation of flows* — $\forall u \in V \setminus \{s, t\} \; \sum_{v \in V} f(u, v) = 0$.

You are given a network and some function on the pairs of vertices. Check that this function is the flow in the given network.

Input

The first line contains the number of vertices in network N ($2 \leq N \leq 100$). The vertices in the network have ids from 1 to N.

The next N lines contain the capacity of edges. Each such line contains N numbers. The j-th number in $i + 1$-th line defines $c(i, j)$. All capacities are non-negative and are not bigger than 10^4. It is guaranteed that $c(i, i) = 0$.

A single blank line follows.

The next N lines contain the values of the function f in the same format. These values don't exceed 10^4 by the absolute value.

The source of the network is the vertex 1, and the sink is the vertex N.

Output

The first line of the output shoud contain `YES`, if the function is a flow, and `NO`, otherwise.

Examples

flow.in	flow.out

4
0 1 3 0
0 0 0 2
0 0 0 4
0 0 0 0
0 1 2 0
-1 0 0 1
-2 0 0 2
0 -1 -2 0
	YES

4
0 1 3 0
0 0 0 2
0 0 0 4
0 0 0 0
0 2 1 0
-2 0 0 2
-1 0 0 1
0 -2 -1 0
	NO

Problem L. Game

Input file:	`game.in`
Output file:	`game.out`
Time limit:	2 seconds
Memory limit:	64 megabytes

The King of Byteland holds an annual intellectual game. By the rules of the game all players are split into two teams, and each player is given a list of clues. During the game, all participants could exchange their clues using the following rules: each player P_1 of the first team could choose the player P_2 of the other team and ask him for his clue, which the P_1 don't know. If there are multiple such clues, P_2 could tell him any of them. Each player of the first team could ask for the clue exactly one person of the second team, but in the meantime it is possible, that the person from the second team is asked by many players. The first team wins, if they could get all the clues. Help the captain to know, if his team could get all the clues, not depending on the answers of the players in the second team.

Input

The first line contains three integer numbers n, m $(1 \leq n, m \leq 500)$ and k $(1 \leq k \leq 5000)$ – the sizes of teams and the number of clues, respectively. Each of the next $n+m$ lines contains the information about clues, which is held at the start of the game in the following format. The first number in the line corresponds to the number of the clues of the player, and next numbers are the ids of the clues – natural numbers, not bigger than k.

Output

If the first team could collect all the clues, in the first line print "1". On the second line print n integers, for each player of the first team specify, whon from the second team he should ask. If there are multiple answers, print any of them. If the first team couldn't get all the clues, then in the sole line print "2".

Example

game.in	game.out

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

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

Problem M. Bipartite Matching

Input file:	`matching.in`
Output file:	`matching.out`
Time limit:	2 seconds
Memory limit:	64 megabytes

Given a bipartite unweighed graph. You are asked to find a maximum bipartite matching.

Input

The first line of the input file contains three integers n, m k $(1 \leq n, m \leq 200$, $1 \leq k \leq n \times m)$ — the number of vertices in the first and the second parts of the graph (more precisely, each part has n vertices), and the number of graph edges, correspondingly.

k lines follow, each of them contains two numbers a_i and b_i, which denotes an edge going from the vertex a_i from the first part of the graph to the vertex b_i of the second part of the graph. In each part, vertices are numbered starting from 1.

Output

Output a single number: the maximum number of edges in the bipartite matching.

Examples

matching.in	matching.out

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