Problem A. Dominoes in the Box

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

The bottom of a rectangular box has the form of $N \times M$ cm. You must tile it with dominoes of size 1×2 cm. Dominoes must completely cover the bottom of the box, and they must not overlap.

The tiling is called solid, if any line that separates the bottom of the box into two non-empty parts crosses the interior of at least one domino.

You must create a program that would construct a solid tiling of the bottom of the box.

Input

The first line of the input file contains N and M ($1 \le N, M \le 1000$).

Output

If the solid tiling exists, print the rectangular table of lowercase letters of the English alphabet ('`a`'...'`z`'). The table must contain N rows and M columns. Two equal adjacent characters denote a domino. For each character exactly one adjacent character must be equal to it.

If no solid tiling exists, create an empty output file.

Example

domino.in	domino.out
6 7	aabbcdd effgchh eiigjjk llmnnok pqmrros pqttuus

Problem B. Arcanoid

Input file:	arcanoid.in
Output file:	arcanoid.out
Time limit:	2 seconds
Memory limit:	64 mebibytes

Aino and friends watch as N-dimensional ball of radius R flies in N-dimensional cube with side length equal to D. The ball reflects from the walls by the law of light reflection:

- The trajectory before reflection, the trajectory after reflection, and the normal to the reflection surface at the point where the ball touches the wall are in the same plane;
- The angle which the trajectory of the ball makes to the normal is equal to the angle which the trajectory after reflection makes to the same normal.

But the ball flies too fast for Aino to notice the reflections from the walls. Your problem is to help Aino to find the location where the M-th touching of the wall will be. The cube is positioned in such way that one of its vertices has coordinates $(0, 0, ..., 0)$, another one has coordinates $(D, D, ..., D)$, and its edges are parallel to coordinate axes.

Input

The input file contains three lines.

The first line contains integer numbers: N ($1 \le N \le 2008$), D ($1 \le D \le 1000$), R ($1 \le R < D/2$), M ($1 \le M \le 2008$) — the dimension of the cube, the length of its edge, the radius of the ball and the number of the reflection that Aino is interested in. The second line contains N integer numbers — the initial coordinates of the center of the ball. It is guaranteed that the ball doesn't have common points with the cube's walls, and is completely inside the cube. The third line contains N integer numbers — the coordinates of the vector of the ball's velocity. All coordinates do not exceed 1000. The velocity is not equal to 0.

Output

Print one line to the output file — the coordinates of the M-th touching point. Each coordinate must be printed with at least 6 points after the decimal point. It is guaranteed that each time the ball touches the wall it touches exactly one wall.

Example

arcanoid.in	arcanoid.out
4 100 1 1 5 5 5 5 -1 -2 -3 -4	4.000000 3.000000 2.000000 0.000000

Problem C. Degree of Minimality of Spanning Tree

Input file:	span.in
Output file:	span.out
Time limit:	2 seconds
Memory limit:	64 mebibytes

The scientists of Htrae planet are studying the problem yet unsolved there — finding a minimal spanning tree in an undirected graph. Everybody know that adding one edge to a tree causes exactly one cycle to appear. Based on this fact, one of the leading mathematicians of the planet proved a criterion of minimality for spanning trees — "A spanning tree is minimal if and only if after adding any graph edge to the given tree, the added edge has the greatest weight in the corresponding cycle."

Further study of the problem didn't lead to better results. In order to help scientists it was decided to run a contest for the best algorithm. Since nobody at Htrae knows how to create the best solution, it was decided to evaluate solutions based on the minimality criteria. Consider an edge w not included into the spanning tree. Let us add it to the tree. One new cycle appears. Find the sum of weights of all edges of the cycle that have a weight greater than w has. Let this sum be y, the number of such edges be k and the edge w have weight x. In this case the goodness of the edge w is $y - kx$.

The degree of minimality of the spanning tree is the sum of goodnesses of all edges not included into the spanning tree.

Given a graph and a spanning tree in it, find the degree of minimality of the spanning tree in the graph.

Input

The first line of the input file contains n and m ($1 \le n \le 75000$, $n - 1 \le m \le 100000$) — the number of vertices and edges in the graph, respectively. Each of the following $n - 1$ lines describes one edge included into the spanning tree. Each edge is described by three numbers: x y z, where x and y are the numbers of the vertices connected by the edge and z is the weight of the edge ($0 \le z \le 10^4$). The following $m - n + 1$ lines describe edges not included into the spanning tree in the same format.

Output

Output one number — the degree of minimality of the given spanning tree.

Example

span.in	span.out
7 10 1 2 5 1 3 6 3 4 3 3 5 2 5 6 1 6 7 1 6 2 2 7 3 7 7 4 2 5 2 4	11
3 3 1 2 2 1 3 2 2 3 1	2

Problem D. Parking

Input file:	parking.in
Output file:	parking.out
Time limit:	2 seconds
Memory limit:	64 mebibytes

Parking areas in Beerland are extremely popular, so sometimes it is quite impossible to put a car there. For simplicity let us consider parking as a rectangular field of size $N \times M$, let each car occupy two adjacent cells (vertically or horizontally). Some cells in the parking are occupied by guards, so moving a car through such cells is impossible.

You can ask a worker on the parking move any car one cell forward or one cell backwards (provided that the corresponding cell is free). You can move any car several times. Your goal is to put your car to the parking. In order to do so you must have two adjacent free cells in the parking. You don't care whether you will be actually able to *drive* to the corresponding location, you can use a helicopter to put your car there.

Input

The first line of the input file contains two numbers N and M ($1 \le N, M \le 500$) — the size of the parking. The following N lines contain M characters each and describe the parking. Each character is either '`.`' — free space, or '`X`' — a guardian. The next line contains K — the number of cars on the parking. Each of the following K lines describes a car, the car is described by four numbers x_1, y_1, x_2, y_2, ($1 \le x_1, x_2 \le N$, $1 \le y_1, y_2 \le M$), where (x_1, y_1) and (x_2, y_2) are the coordinates of the cells occupied by the car. The top-left cell has coordinates $(1, 1)$, the bottom-right — (N, M).

Output

If it is possible to put a car to the parking, print "`Yes`". In the other case print "`No`". If it is possible to put a car, the next line must contain the number of movements, and then the movements, one on a line. Each movement is described by numbers x_1, y_1, x_2, y_2, ($1 \le x_1, x_2 \le N$, $1 \le y_1, y_2 \le M$), where (x_1, y_1) are the coordinates of the free cell where the car moves to, and (x_2, y_2) are the coordinates of the cell adjacent to (x_1, y_1) which is already occupied by the car in question.

Example

parking.in	parking.out
3 6 ....XX XXX.XX XXX... 3 1 2 1 3 1 4 2 4 3 4 3 5	Yes 3 3 6 3 5 3 4 2 4 1 4 1 3

Problem E. Coloring Cacti

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

> A *cactus* is any member of the succulent plant family Cactaceae, native to the Americas.
> Cacti come in a wide range of shapes and sizes.
>
> From Wikipedia — free encyclopedia

Cactus is a connected undirected graph such that each edge in it belongs to at most one cycle. You must color vertices of a cactus into minimal number of colors so that no edge connects two vertices of the same color.

Input

The first line of the input file contains N and M ($1 \le N \le 50000, 0 \le M \le 10000$), where N is the number of vertices in the cactus. All edges of the cactus are described by M paths, each edge belongs to exactly one such path. The following M lines contain descriptions of these paths. Each description starts with k_i ($2 \le k_i \le 1000$) — the number of vertices in the path. It is followed by k_i numbers, each of them from 1 to N, they describe vertices along the path. Any two vertices are connected by at most one edge. No edge connects a vertex to itself.

Output

The first line of the output file must contain K — the minimal number of colors. The second line must contain N numbers, the i-th of these numbers must be a_i ($1 \le a_i \le K$) — the color of the i-th vertex. If there are several optimal colorings, output any one.

Example

coloring.in	coloring.out
1 0	1 1
4 1 5 1 2 3 4 1	2 1 2 1 2
3 1 4 1 2 3 1	3 1 2 3

Problem F. Cacti strike back

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

As you might remember, a *cactus* is a connected undirected graph with each edge belonging to at most one simple cycle. Given a weighed graph with each connected component being a cactus, color each of its vertices to one of K colors in such a way that the sum of weights of edges connecting vertices of different colors is minimized. You must use each color at least once.

Input

The first line of the input contains three integer numbers N ($1 \le N \le 10000$), M ($M \ge 0$) and K ($1 \le K \le N, K \le 1000$). N is the amount of vertices, M is the amount of edges, K is the amount of colors. The next M lines describe the edges: each line contains three integer numbers a, b, c ($1 \le a, b \le N$, $0 \le c \le 10000$), where a and b are vertices connected by the edge, and c is its weight. Any two vertices are connected by at most one edge. No edge connects a vertex to itself.

Output

The first line of the output file must contain the sum of weights of edges that would connect the vertices of different colors. The second line must contain N numbers, the i-th of these numbers must be the color of the i-th vertex in the optimal coloring.

Example

cacti.in	cacti.out
2 0 1	0 1 1
3 3 2 1 2 1 2 3 2 1 3 1	2 1 2 2

Problem G. Filter Tables

Input file:	`iptables.in`
Output file:	`iptables.out`
Time limit:	2 seconds
Memory limit:	64 mebibytes

Linux operating system is equipped with a layer 3 network subsystem called *iptables*, responsible for filtering all IP packets, either incoming, outcoming or forwarded. You are to implement basic functionality of a similar subsystem.

A packet to be processed is described by a string

`recv IN=`⟨*itf*⟩ `OUT=`⟨*itf*⟩ `SRC=`⟨*ip*⟩ `DST=`⟨*ip*⟩ `PROTO=`⟨*protocol*⟩ `SPT=`⟨*port*⟩ `DPT=`⟨*port*⟩ `LEN=`⟨*size*⟩

Here

- *itf*: Interface through which the packet has been input or has to be output. Interfaces are denoted `eth0`–`eth15`. Input packets have empty output interface, output packets have empty input interface, and forwarded packets have non-empty input and output interfaces.
- *ip*: IP address in standard four octet notation: $x_1.x_2.x_3.x_4$, where $0 \le x_i \le 255$. IP address might be thought of as an unsigned 32-bit integer, with most significant byte x_1 written first.
- *protocol*: Protocol description — one of TCP, UDP, ICMP for the purpose of this task.
- *port*: Port number, integer ranging from 1 to 65535.
- *size*: Packet size in bytes, integer ranging from 1 to 2000.

Iptables system is based on the notion of a *chain*. A *chain* is a sequence of *rules* and a default *policy*. Each *rule* is a set of conditions and an action. There are three predefined chains, called `INPUT`, `OUTPUT` and `FORWARD`, used for input, output and forwarded packets, respectively. User-defined chains might also be present, but we neglect this possibility in this task. One should remember, therefore, that the next paragraph describes only a very simplified version of true `iptables` functionality.

Each packet is processed as follows. First, corresponding standard chain is chosen. Then the packet is checked against each rule of this chain, in order of their creation. If the packet matches the rule condition, its action is executed. Action can be *final* (`ACCEPT`, `DROP`, `REJECT`) — then this packet is not processed further. We consider only two *non-final* actions `LOG` — it doesn't affect the packet (just adds a line into system log file), and `RETURN` — for standard chains it means that the chain default policy is to be applied immediately to the packet being processed.

If a packet traverses the whole of a standard chain, chain default policy (always a final action) is applied unconditionally.

For each rule of each chain, as well as for default policies, packet count and total size counters are kept. They are increased whenever rule action or default policy are invoked for a packet.

At the very beginning all standard chains are empty, have zero counters and their default policy is `ACCEPT`.

For solving this task one has to check where a given ip address *ip* enters into a subnetwork *net*/*mask*, where *net* is given in four-octet IP address notation, and *mask* is an integer ranging from 0 to 32. Then an IP address *ip* enters into given subnetwork if after representing *ip* by an unsigned 32-bit integer and cleaning its $32 - mask$ least significant bits it equals *net*.

One is allowed to manipulate chains as follows:

- `iptables -F` *chain* — clean chain named *chain*. All rules from this chain are dropped, all counters are reset, default policy is set to `ACCEPT`.
- `iptables -P` *chain target* — set *target* default policy for chain *chain*. Policy counters are reset to zero.
- `iptables -Z` *chain* — reset to zero all rule and policy counters in given chain.
- `iptables -A` *chain* [`-i` *interface*] [`-o` *interface*] [`-p` *proto*] [`--sport` *port*] [`--dport` *port*] [`-s` [`!`] *ip* | *net/mask*] [`-d` [`!`] *ip* | *net/mask*] `-j` *target* — add rule into given chain. Parameters can be given in any order.

  `-i` *interface* IN-interface of packet must have given value.

  `-o` *interface* OUT-interface of packet must equal given value.

  `-p` *proto* packet protocol must be equal to *proto*.

  `--sport` *port* packet SPT must coincide with *port*.

  `--dport` *port* packet DPT must coincide with *port*.

  `-s [!]` *ip* `|` *net/mask* only packets with SRC equal to *ip*, or lying in subnetwork *net/mask*, match. Exclamation mark means negation of condition.

  `-d [!]` *ip* `|` *net/mask* only packets with DEST equal to *ip*, or lying in subnetwork *net/mask*, match. Exclamation mark means negation of condition.

  `-j` *target* action.
- `iptables -vL` — output all chain counters in order INPUT, OUTPUT, FORWARD. Standard chain header: "`Chain` *chain* `(policy` *policy* *pkts* `packets,` *bytes* `bytes)`". Here *chain* is chain name (one of `INPUT`, `OUTPUT`, `FORWARD`), *policy* — default policy, *pkts* and *bytes* is the quantity and total size of packets, to which default policy has been applied. Statistics table for all rules of given chain follows. Column names are: "`pkts bytes target proto in out source destination sport dport`". All columns are right-justified by spaces: `source` and `destination` are 22 characters wide, all other columns are 8 characters wide. For each chain rule the quantity and total size of matched packets, rule action, protocol, input and output interfaces, source, destination, source and destination ports are output. If a parameter wasn't specified in a rule, one should output `anywhere` for IP address fields, and `any` for other fields. If a subnet mask *net/mask* with *mask* = 32 is output, then the "`/32`" part is omitted. Follow the sample output as close as possible.

Input

Input file contains packet descriptions and `iptables` commands, defined above, as well as empty lines.

Output

For each statistics command output all three statistics tables, followed by an empty line.

Example

iptables.in

```
iptables -A INPUT -p TCP --dport 22 -j ACCEPT
iptables -A INPUT -p TCP --dport 445 -j LOG
iptables -A INPUT -p ICMP -s 192.168.101.0/24 -j ACCEPT
iptables -A INPUT -j DROP
iptables -A FORWARD -d 212.193.33.130/32 --dport 80 -j ACCEPT
iptables -A FORWARD -s ! 192.168.101.0/24 -j REJECT
iptables -P FORWARD DROP
iptables -vL
recv IN=eth0 OUT= SRC=212.193.32.1 DST=192.168.0.1 PROTO=TCP SPT=6023 DPT=80 LEN=60
recv IN=eth0 OUT=eth1 SRC=212.193.32.1 DST=192.168.0.1 PROTO=TCP SPT=6023 DPT=80 LEN=40
iptables -vL
recv IN=eth0 OUT= SRC=212.193.32.1 DST=212.193.33.130 PROTO=TCP SPT=6023 DPT=80 LEN=55
recv IN=eth1 OUT= SRC=192.168.101.13 DST=192.168.101.1 PROTO=TCP SPT=7013 DPT=445 LEN=12
recv IN=eth0 OUT= SRC=192.168.101.13 DST=192.168.101.1 PROTO=ICMP SPT=7013 DPT=445 LEN=10
iptables -vL
```

iptables.out

```
Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
    pkts   bytes  target   proto      in     out               source           destination   sport   dport
       0       0  ACCEPT     TCP     any     any             anywhere             anywhere     any      22
       0       0     LOG     TCP     any     any             anywhere             anywhere     any     445
       0       0  ACCEPT    ICMP     any     any      192.168.101.0/24            anywhere     any     any
       0       0    DROP     any     any     any             anywhere             anywhere     any     any
Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
    pkts   bytes  target   proto      in     out               source           destination   sport   dport
Chain FORWARD (policy DROP 0 packets, 0 bytes)
    pkts   bytes  target   proto      in     out               source           destination   sport   dport
       0       0  ACCEPT     any     any     any             anywhere         212.193.33.130   any      80
       0       0  REJECT     any     any     any    ! 192.168.101.0/24            anywhere     any     any

Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
    pkts   bytes  target   proto      in     out               source           destination   sport   dport
       0       0  ACCEPT     TCP     any     any             anywhere             anywhere     any      22
       0       0     LOG     TCP     any     any             anywhere             anywhere     any     445
       0       0  ACCEPT    ICMP     any     any      192.168.101.0/24            anywhere     any     any
       1      60    DROP     any     any     any             anywhere             anywhere     any     any
Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
    pkts   bytes  target   proto      in     out               source           destination   sport   dport
Chain FORWARD (policy DROP 0 packets, 0 bytes)
    pkts   bytes  target   proto      in     out               source           destination   sport   dport
       0       0  ACCEPT     any     any     any             anywhere         212.193.33.130   any      80
       1      40  REJECT     any     any     any    ! 192.168.101.0/24            anywhere     any     any

Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
    pkts   bytes  target   proto      in     out               source           destination   sport   dport
       0       0  ACCEPT     TCP     any     any             anywhere             anywhere     any      22
       1      12     LOG     TCP     any     any             anywhere             anywhere     any     445
       1      10  ACCEPT    ICMP     any     any      192.168.101.0/24            anywhere     any     any
       3     127    DROP     any     any     any             anywhere             anywhere     any     any
Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
    pkts   bytes  target   proto      in     out               source           destination   sport   dport
Chain FORWARD (policy DROP 0 packets, 0 bytes)
    pkts   bytes  target   proto      in     out               source           destination   sport   dport
       0       0  ACCEPT     any     any     any             anywhere         212.193.33.130   any      80
       1      40  REJECT     any     any     any    ! 192.168.101.0/24            anywhere     any     any
```

Problem H. ATM

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

Imagine that you are an ordinary student. And you are very hungry. Today is a holiday for all students — the day of getting the stipend. There are very long queues for ATM.

It is known that you have S roubles on your account. ATM can dispense notes of 50, 100 and 500 roubles. When dispensing cash, ATM at first maximimizes the number of 500-rouble notes, then maximizes the number of 100-rouble notes.

It seems easy, but life of an ordinary student is affected very much by trolleybuses used to reach your university from your home. In these trolleybuses, it is impossible to get change from any note but 50-rouble one. Your favorite shaurma stall also won't accept 500-rouble notes.

So your task is to get all money from your account, but maximizing the number of 50-rouble notes first, the number of 100-rouble ones next. The only problem is the long queue behind you, they'll become furious if you will use ATM more than N times.

Input

Input file contains two integers: S ($0 \le S \le 10^9$), S is always divisible by 50, and N ($0 < N \le 10^5$). S is the amount of money on your account and N is the maximal possible number of times you can use the ATM.

Output

Output the integer K ($0 \le K \le N$) on the first line of output. K is the number of times you should use the ATM. Then output K lines — $i+1$-th line must contain a_i ($0 \le a_i \le S$) — the amount of money to withdraw during i-th operation. All a_i must be divisible by 50 and their sum must be equal to S.

Example

cash.in	cash.out
0 100000	0
100 1	1 100
100 2	2 50 50

Problem I. Monsters

Input file:	monsters.in
Output file:	monsters.out
Time limit:	2 seconds
Memory limit:	64 mebibytes

Good news from Berland! New part of the famous game "The Heroes of Knout and Cakes" has appeared. In one of the missions you can help main hero Mesher to deliver cakes. But there are N greedy monsters on his path. Monsters want to steal cakes from our hero.

Fortunately, Mesher took with him his favourite three-volume edition of a famous author and can use it for defense, because some parts of this book will affect anybody. To complete his mission Mesher has to defeat monsters. Battle occurs by steps. In the beggining of each move, all monsters take some cakes away from Mesher (k-th monster takes p_k cakes). Then, Mesher reads one chapter from the book to one monster. This chapter affects only this monster, because other monsters don't hear it. Once some monster hears g_k chapters from the book, it becomes peaceful and stops taking cakes away. In one move Mesher can read only one chapter.

Your task is to find minimal amount of cakes that Mesher will lose.

Input

First line of input file contains single integer number $1 \le N \le 100$ — amount of monsters. Then N lines follow, k-th line contains two integer numbers g_k, p_k $1 \le g_k, p_k \le 10000$ — k-th monster characteristics.

Output

First line of output file must contain minimal number of cakes. In second line print a permutation of integers $1 \dots N$ — the order in which Mesher must destroy monsters. If there are multiple solutions, output any.

Example

monsters.in	monsters.out
3 4 1 1 2 3 3	22 2 3 1