# Problem A. Areas

| | |
|---|---|
| Input file: | areas.in |
| Output file: | areas.out |
| Time limit: | 1 second |

Consider $N$ different lines on the plane. They divide it to several parts, some of which are finite, some infinite. Your task in this problem is for each finite part to find its area.

## Input

The first line of the input file contains $N$ — the number of lines ($1 \le N \le 80$). Each of next $N$ lines contains four integer numbers $x_1$, $y_1$, $x_2$ and $y_2$ — the coordinates of two different points of the line.

All coordinates do not exceed $10^2$ by their absolute value.

No two lines coincide.

## Output

First output $K$ — the number of finite parts among those the lines divide the plane to.

Next $K$ lines of the output file must contain area parts sorted in non-decreasing order. You answer must be accurate up to $10^{-4}$.

Due to floating point precision losses possible, do not consider parts with area not exceeding $10^{-8}$.

## Example

| areas.in | areas.out |
|---|---|
| 5<br>0 0 1 0<br>1 0 1 1<br>1 1 0 1<br>0 1 0 0<br>0 0 1 1 | 2<br>0.5000<br>0.5000 |

# Problem B. Beloved Sons

| | |
|---|---|
| Input file: | `beloved.in` |
| Output file: | `beloved.out` |
| Time limit: | 1 second |

Once upon a time there lived a king and he had $N$ sons. And the king wanted to marry his beloved sons on the girls that they did love. So one day the king asked his sons to come to his room and tell him whom do they love.

But the sons of the king were all young men so they could not tell exactly whom they did love. Instead of that they just told him the names of the girls that seemed beautiful to them, but since they were all different, their choices of beautiful girls also did not match exactly.

The king was wise. He did write down the information that the children have provided him with and called you, his main wizard.

"I want all my kids to be happy, you know," he told you, "but since it might be impossible, I want at least some of them to marry the girl they like. So please, prepare the marriage list."

Suddenly you recalled that not so long ago the king told you about each of his sons, so you knew how much he loves him. So you decided to please the king and make such a marriage list that the king would be most happy. You know that the happiness of the king will be proportional to the square root of the sum of the squares of his love to the sons that would marry the girls they like.

So, go on, make a list to maximize the king's happiness.

## Input

The first line of the input file contains $N$ — the number of king's sons ($1 \le N \le 400$). The second line contains $N$ integer numbers $A_i$ ranging from 1 to 1000 — the measures of king's love to each of his sons.

Next $N$ lines contain lists of king's sons' preferences — first $K_i$ — the number of the girls the $i$-th son of the king likes, and then $K_i$ integer numbers — the girls he likes (all potentially beautiful girls in the kingdom were numbered from 1 to $N$, you know, beautiful girls were rare in those days).

## Output

Output $N$ numbers — for each son output the number of the beautiful girl he must marry or 0 if he must not marry the girl he likes.

Denote the set of sons that marry a girl they like by $L$, then you must maximize the value of

$$\sqrt{\sum_{i \in L} A_i^2}$$

## Example

| beloved.in | beloved.out |
|---|---|
| 4<br>1 3 2 4<br>4 1 2 3 4<br>2 1 4<br>2 1 4<br>2 1 4 | 2 1 0 4 |

# Problem C. Strange Counter

| | |
|---|---|
| Input file: | `counter.in` |
| Output file: | `counter.out` |
| Time limit: | 1 second |

The new secret counter invented in one theoretical computer science lab is the great breakthrough in the computer microschemes design. This counter consists of $N$ registers numbered from 0 to $N-1$, each of which contains 0, 1 or 2. If the number in the $i$-th register is $A_i$, the number stored in the counter is $\sum_{i=0}^{N-1} A_i 2^i$.

One can see that the same number can be stored in the counter in several different ways. For example, the number 5 can be stored in a 3-register counter as $(1, 0, 1)$ or as $(0, 2, 1)$.

The main feature of the counter is that it can add numbers that are powers of two to the number stored in the counter, only changing the value of a small number of registers. Namely, the scientists of the lab developed the scheme that allowed adding such number changing no more than four registers!

Unfortunately after the recent experiments in the neighbouring physics lab, involving the creation of the artificial black hole, the theoretical computer science laboratory was accidently destroyed. However, the supercomputer project that the counter was designed for is still on, so you were asked to reinvent the counter.

## Input

The first line of the input file contains $N$ — the number of registers in the counter ($1 \le N \le 1\,000$). Initially all registers contain zeroes. The second line contains $M$ — the number of additions you have to make ($1 \le M \le 10\,000$). The third line contains $M$ integer numbers ranging from 0 to $N-1$. Number $i$ means that you must add $2^i$ to the number in the counter. There sum of all numbers added to the register does not exceed $2^N - 1$.

## Output

Output file must contain $M$ lines. Each line must contain the changes made adding the corresponding number to the counter.

The first number in each line must be $K$ — the number of registers changed ($1 \le K \le 4$). $K$ pairs must follow — for each changed register first output its number and then the new value.

## Example

| counter.in | counter.out |
|---|---|
| 5<br>6<br>0 0 0 1 0 0 | 1 0 1<br>1 0 2<br>2 0 1 1 1<br>1 1 2<br>1 0 2<br>3 0 1 1 1 2 1 |

# Problem D. Data Transmission

| | |
|---|---|
| Input file: | `data.in` |
| Output file: | `data.out` |
| Time limit: | 1 second |

Recently one known microprocessor productioner has developed the new type of microprocessors that can be used in difficult mathematical calculations. The processor contains $N$ so called *nodes* that are connected by $M$ *channels*. Data organized in packets, pass from *source node* to *target node* by channels and are processed by the intermediate nodes.

Each node has its *level* that determines the type of work this node does. The source node has level 1 while the target node has level $L$. For data to be correctly processed each packet of it must pass in order all nodes with levels from 1 to $L$ — that is, first it must be processed by the source node, after that by some node of level 2, so on, and finally by the target node.

Nodes can process as much data as they are asked to, however channels can only transmit the limited amount of data in a unit of time. For synchronization reasons, any data can only be transmitted from a node with level $i$ to some node with level $i + 1$ and cannot be transmitted between nodes which levels differ by more than one or from a node of higher level to a node of lower level. Nodes are so fast that they can process data packet immediately, so as soon as it reaches the node it is ready to be transmitted to the node of the next level.

No data should stall in any node and no node can produce its own data, so each unit of time the number of packets coming to any node except source and target, must be equal to the number of packets leaving this node.

The scheme of data transmission that satisfies the conditions provided is called the *data flow*. Data flow is called *blocking* if there is no way to increase the value of the data flow just increasing the amount of data passing by some channels (however, there may be the way to increase it, *decreasing* the amount of data for some channels and increasing for other ones).

Your task is to find some blocking data flow for the microprocessor given its scheme. Note, that you need not find the *maximal* possible data flow, just any blocking one.

## Input

The first line of the input file contains three integer numbers — $N$, $M$ and $L$ ($2 \le N \le 1\,500$, $1 \le M \le 300\,000$, $2 \le L \le N$). Let nodes be numbered from 1 to $N$. The second line contains $N$ integer numbers, $i$-th of them is the level $l_i$ of the $i$-th node ($1 \le l_i \le L$). Only one node has level 1, that is the source node, and only one node has level $L$ — that is the target node.

Next $M$ lines describe channels, each lines contains three integer numbers $a$, $b$ and $c$ — nodes connected by this channel and its capacity in packets per unit of time ($1 \le a, b \le N$, $l_b = l_a + 1$, $1 \le c \le 10^6$).

Two nodes can be connected by at most one channel.

## Output

Output the description of the data flow found. Output file must contain $M$ lines, they must correspond to channels and contain the amount of data transmitted by the channel in a unit of time. Channels must be listed in the order they are specified in the input file.

## Example

| data.in | data.out |
|---|---|
| 6 7 4<br>1 2 3 4 3 2<br>1 2 3<br>2 3 3<br>3 4 4<br>1 6 4<br>6 3 2<br>5 4 3<br>6 5 4 | 3<br>3<br>4<br>4<br>1<br>3<br>3 |

# Problem E. Strong Defence

| | |
|---|---|
| Input file: | `defence.in` |
| Output file: | `defence.out` |
| Time limit: | 1 second |

The Chief of the Galactic Empire has recently received some bad news from his spies. The Dark Lord is preparing to attack the Empire. His fleet of spaceships is ready for the first hyperjump.

It is well known that travelling in space is very simple. You just start from some star and make a series of hyperjumps to other stars. You can only jump from one star to another if they are connected with a special hyperjump tunnel, which is bidirectional, thus allowing to make a jump from one star that it connects to another. Of course, the tunnels are designed in such a way that there is the way to get from each star to any other one.

However, there is the way to block the hyperjump — to do this one must put a special battleship in the corresponding hypertunnel.

Of course, the Chief would like to block all hyperpaths from the star where the headquaters of the Dark Lord are located to the star where the capital of the Galactic Empire is. The resources of the Empire are almost unbounded, so it is easy to create as many battleships as needed. Unfortunately, there is one problem.

Each hyperjump blocking battleship must have a special crystal on board which allows him to stay in the hyperspace. There is a number of types of such crystals. The problem is that there is the way to destroy all battleships carrying some particular type of crystal.

Although it is known, that for each crystal type there is the way to destroy battleships powered by this crystal, there is hope that not all of those are known to Dark Lord engineers. So the Chief would like to use blocking ships in such a way that the following conditions are satisfied:

- for each crystal type, if all ships with other crystal types are destroyed, battle ships with this crystal type block hypertunnels in such a way, that there is no path from Dark Lord's star to Empire Capital star;
- the number of different crystal types used in ships is maximal possible;
- no two ships block the same hypertunnel.

You may consider that there is the unlimited number of crystal types available and crystals of each type available.

## Input

The first line of the input file contains $N$ — the number of stars in the Galaxy($2 \le N \le 400$), $M$ – the number of tunnels, $S$ and $T$ — numbers of stars where Dark Lord headquaters and Empire Capital are located respectively ($S \ne T$).

Next $M$ lines contain two integer numbers each — the numbers of the stars the corresponding tunnel connects. No tunnel connects a star to itself, no two stars are connected with more than one tunnel.

## Output

First output $L$ — the number of crystal types used. After that output $L$ lines, for each crystal type output first $K_i$ — the number of battleships with this crystal used, and then $K_i$ numbers, identifying the hypertunnels blocked by the corresponding battleship. The tunnels are numbered starting from 1, as they are given in the input file.

## Example

| defence.in | defence.out |
|---|---|
| 4 4 1 4<br>1 2<br>1 3<br>2 4<br>3 4 | 2<br>2 1 2<br>2 3 4 |

# Problem F. Weird Dissimilarity

| | |
|---|---|
| Input file: | `dissim.in` |
| Output file: | `dissim.out` |
| Time limit: | 1 second |

The issue of this problem is to find out how far two strings over alphabet $\Sigma$ are, with respect to one weird definition of dissimilarity. For any two characters $c_1$ and $c_2$ from $\Sigma$ consider *dissimilarity* $d(c_1, c_2)$ of $c_1$ from $c_2$ — non-negative integer number. If we take two strings $\alpha$ and $\beta$ of equal length $l$, *distance* from $\alpha$ to $\beta$ is $dist(\alpha, \beta) = \sum_{i=1}^{l} d(\alpha[i], \beta[i])$.

You are given two strings $\lambda$ and $\mu$. Consider all possible pairs of strings $\alpha$ and $\beta$ of equal length over $\Sigma$, such that $\lambda$ is a subsequence of $\alpha$ and $\mu$ is a subsequence of $\beta$ (string $\omega$ of length $n$ is a subsequence of a string $\xi$ of length $m$ if there exist $1 \le i_1 < i_2 < \ldots < i_n \le m$ such that $\omega[j] = \xi[i_j]$ for all $1 \le j \le n$). Choose among them $\hat{\alpha}$ and $\hat{\beta}$ such that $dist(\hat{\alpha}, \hat{\beta})$ is minimal possible. *Dissimilarity* of $\lambda$ from $\mu$ is defined as $D(\lambda, \mu) = dist(\hat{\alpha}, \hat{\beta})$.

Your task is to find the dissimilarity of $\lambda$ from $\mu$ and to provide $\hat{\alpha}$ and $\hat{\beta}$ such that $D(\lambda, \mu) = dist(\hat{\alpha}, \hat{\beta})$.

## Input

The first line of the input file contains $\Sigma$ — several different characters that form the alphabet for the strings we consider ($1 \le |\Sigma| \le 200$, all characters have ASCII code greater than space). Next two lines contain $\lambda$ and $\mu$ respectively. Length of each of the given strings does not exceed 2000. Next $|\Sigma|$ lines contain $|\Sigma|$ non-negative integer numbers each, $j$-th number of $i$-th line contains dissimilarity of $i$-th character from $j$-th.

## Output

On the first line of the output file print $D(\lambda, \mu)$. On the second and third lines of the output file print $\hat{\alpha}$ and $\hat{\beta}$, such that $D(\lambda, \mu) = dist(\hat{\alpha}, \hat{\beta})$, $\lambda$ is a subsequence of $\hat{\alpha}$ and $\mu$ is a subsequence of $\hat{\beta}$. Length of each of $\hat{\alpha}$ and $\hat{\beta}$ must not exceed 4000.

## Example

| dissim.in | dissim.out |
|---|---|
| ab<br>ab<br>ba<br>2 1<br>4 1 | 4<br>aba<br>bba |

# Problem G. PL/Cool

| | |
|---|---|
| Input file: | `plcool.in` |
| Output file: | `plcool.out` |
| Time limit: | 2 seconds |

The new IMB compiler "Unvisual Age for PL/Cool" is planned for release next month. Your task in this problem is to write the interpreter for the PL/Cool language so that the compiler programmers could test their product.

Program on PL/Cool can contain expressions. Each expression can contain integer constants and variables. The following operations are allowed: `+` (add), `-` (substract), `*` (multiply), `/` (divide), `%` (modulo), and `^` (raise). All operations have their usual meaning and priority (`[^]` > `[*]` = `[/]` = `[%]` > `[+]` = `[-]`), all operations except `^` are evaluated from left to right, `^` is evaluated from right to left. Parenthesis can be used to change the order of evaluation, as usually. Unary minus and plus are allowed, their priority in this case is the highest.

Divide operation acts like integer division: first both operands are replaced with their absolute values, next integer division is performed, remainder is dropped, and finally the sign of the quotient is set equal to the sign of the true quotient of the operands. Taking modulo is perfomed the same way, except that the quotient is dropped and remainder is kept. For example, $(80 + 4 * 75)/(56 - 2\text{^}2\text{^}3) = -1$.

Program may contain variables. Variable name starts with a letter, which may be followed by letters and digits. Variable names are case insensitive. Maximal name length is 10 characters.

PL/Cool has two operators: `print` and `define`. Program is the sequence of the operators, one on a line.

Operator `print` has the following syntax: `print <expression>`

Here expression is any expression that may contain variables and integer constants. Operator `print` prints the value of the expression.

Operator `define` has the following syntax:

```
define <operand1> <operand2>
```

Here `<operand1>` and `<operand2>` are either variables or non-negative integer constants. After execution of the `define` operator, all occurences of the `<operand1>` are replaced with the `<operand2>`. Define operator can be used even to change the value of the integer constants, for example after

```
define 2 4
```

the value of $2 + 2$ is 8.

Note that substitution is performed recursively, until some undefined constant is met, for example the following sequence of operators prints 8 and 10:

```
define 2 4
define 3 2
print 3+3
define 4 5
print 3+3
```

An attempt to redefine some variable or constant is ignored. An attempt to make a definition that leads to circular dependence is also ignored. For example, the following sequence prints 4 and 4, because the last two definitions are ignored.

```
define 2 4
define 3 2
define 2 5
print 3
define 4 2
print 3
```

If some identifier is used in the expression that is not yet defined to be substituted by some integer constant, its value is set to zero. For example, the following operator prints 3:

```
print x + 3
```

## Input

Input file contains the program on PL/Cool. All numbers in the expressions, including numbers that occur during expression evaluation, do not exceed $10^9$ by their absolute value. The total number of `define` operators does not exceed 30000, the total number of `print` operators does not exceed 2000, the length of each line does not exceed 200 characters.

In case of division denominator is never zero, zero is not raised to zero power and power exponent is always non-negative.

## Output

Output file must contain the output of all print operators in the order they are executed, one on a line.

## Example

| plcool.in | plcool.out |
|---|---|
| print (80+4*75)/(56-2^2^3)<br>print 30/-1<br>print 31%-5<br>print 0^3<br>print 2+2<br>define 2 3<br>print 2+2<br>define 3 x<br>print 2+2<br>define x 2<br>print 2+2<br>define 2 5<br>print 2+2<br>define x 7<br>print 2+2 | -1<br>-30<br>-1<br>0<br>4<br>6<br>0<br>0<br>0<br>14 |

# Problem H. Royal Federation

| | |
|---|---|
| Input file: | `royal.in` |
| Output file: | `royal.out` |
| Time limit: | 1 second |

The king of Fooland has recently decided to reorganize his kingdom. Inspired by the democracy processes in neighbouring countries, he decided to convert his kingdom into Royal Federation. The Royal Federation would consist of several provinces, each headed by its governor.

There are $N$ cities in his kingdom, numbered from 1 to $N$. Some cities are connected by roads. Roads are designed in such a way, that for each city there is exactly one way to get to any other city by the roads, not passing through any city more than once.

To prevent wastes for maintaining too small provinces, each province must contain at least $B$ cities. However, to keep governments effective, each province must contain at most $3B$ cities.

Each province must have its governer headquaters in some city. This city may be outside the province itslef, but one must be able to get to the city with governer headquaters of his province in such a way, that all intermediate cities that he visits on his way belong to his province (and only the terminal city may be from another province).

One city may contain headquaters for several provinces.

Help the king to see his plans fulfilled.

## Input

The first line of the input file contains two integer numbers — $N$ and $B$ ($1 \le N \le 10\,000$, $1 \le B \le N$). Next $N - 1$ lines contain descriptions of roads, each line contains two integer numbers — the cities the road connects.

## Output

If it is impossible to fulfil king's plans of reorganization, output 0 on the first line of the output file. In the other case output $K$ — the number of provinces in your plan of the Royal Federation. After that output $N$ integer numbers ranging from 1 to $K$ — for each city output the number of the province it belongs to.

Finally output $K$ integer numbers — the cities where the capitals of the provinces must be located in,

## Example

| royal.in | royal.out |
|---|---|
| 8 2<br>1 2<br>2 3<br>1 8<br>8 7<br>8 6<br>4 6<br>6 5 | 3<br>2 1 1 3 3 3 3 2<br>2 1 8 |

# Problem I. Two Cylinders

| | |
|---|---|
| Input file: | `twocyl.in` |
| Output file: | `twocyl.out` |
| Time limit: | 2 seconds |

In this problem your task is very simple.

Consider two infinite cylinders in three-dimensional space, of radii $R_1$ and $R_2$ respectively, located in such a way that their axes intersect and are perpendicular.

Your task is to find the volume of their intersection.

## Input

Input file contains two real numbers $R_1$ and $R_2$ ($1 \le R_1, R_2 \le 100$);

## Output

Output the volume of the intersection of the cylinders. Your answer must be accurate up to $10^{-4}$.

## Example

| twocyl.in | twocyl.out |
|---|---|
| 1 1 | 5.3333 |