Problem A. TV

Input file:	stdin
Output file:	stdout
Time limit:	2 seconds
Memory limit:	256 megabytes
Feedback:	

Recently Berland scientists discovered new field to research. They explore how to organize TV schedule to maximize happiness of viewers. They proved that a change of happiness after viewing a TV-show depends only on this show and the previous show.

Formally, each TV-show is characterized by three values: t_i, a_i and b_i, where t_i is the type of the i-th TV-show, a_i and b_i are how happiness changes after viewing the i-th show. If the previous TV-show exists (i.e. the current show is not the first) and the previous show type equals to the type of the current show then happiness is increased by a_i. Otherwise after viewing the i-th TV-show happiness is increased by b_i.

There are n TV-shows in total numbered from 1 to n. The problem is to choose some TV-shows and order them in a sequence in a way to maximize happiness after viewing all of them one after another. Actually some TV-shows do not increase happiness, if quality of a TV-show is very low it can have negative a and/or b values. It is possible to choose empty set of TV-shows, in that case happiness will no be changed. Each show can be used at most once.

Input

The first line contains integer n ($1 \leqslant n \leqslant 600$) — the number of TV-shows. Each of the following n lines describes one show. The i-th line contains three integers: t_i, a_i, b_i ($1 \leqslant t_i \leqslant n$; $|a_i|, |b_i| \leqslant 10^5$) — the type and the values of the i-th show.

Output

The output should contain two lines. Output two integer numbers c and k in the first line, where c — maximal happiness change after viewing subset of shows in an appropriate order, k ($0 \leqslant k \leqslant n$) — the number of shows in the optimal subset. Output the sequence of TV-shows in the order that maximizes happiness in the second line. Formally, output k indices $f_1, f_2, \ldots, f_k$ ($1 \leqslant f_j \leqslant n$), where f_1 is the first show to be shown, f_2 is the second show to be shown, and so on. If there are many optimal solutions, any is acceptable.

Examples

stdin	stdout
4 1 10 5 1 5 10 2 -1 -1 3 5 1	21 3 4 2 1
8 2 3 7 2 5 2 5 -1 -1 2 10 9 4 -10 10 4 -10 10 4 -10 10 4 -10 10	60 8 8 4 2 7 1 6 3 5

Problem B. Travelling Camera Problem

Input file:	`stdin`
Output file:	`stdout`
Time limit:	4 seconds
Memory limit:	256 megabytes
Feedback:	

Programming competitions become very popular in Berland. Now Berland Broadcasting Corporation (BBC) plans to organize a TV broadcast of the All-Berland Regional Contest. The contest will be in a narrow hall stretched from the left to the right. BBC puts a long straight rail with a camera on it along the hall. The rail starts from the left wall and goes along the hall to the right wall.

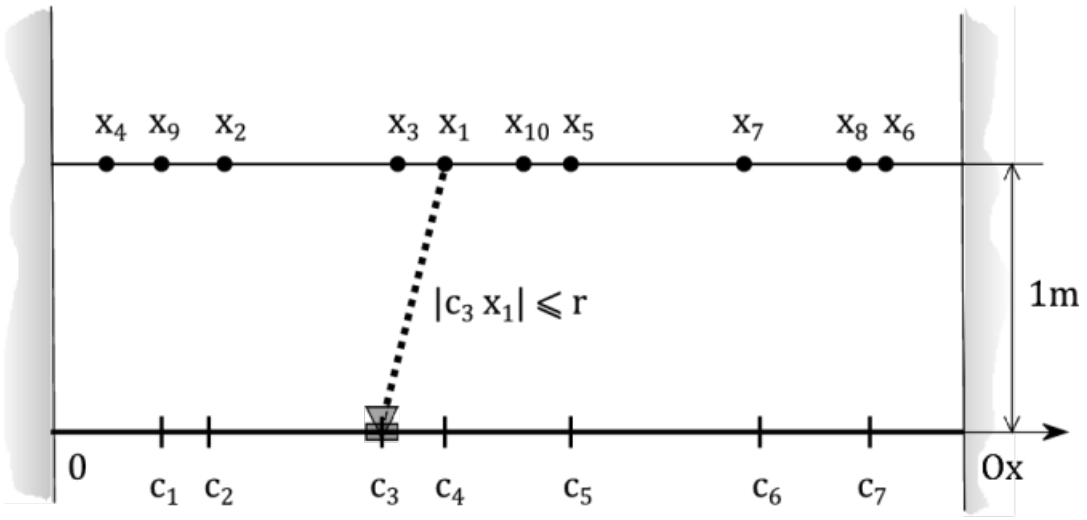
Not every position of the camera on the rail is suitable for shooting. The cameraman has chosen m shooting positions on the rail: $c_1, c_2, \ldots, c_m$, where c_i is the distance (in meters) from the i-th position to the left wall. Initially the camera will be at the position c_1.

The reporter has prepared the plan showing how she will move tomorrow. She will move only along a line parallel to the rail on the distance 1 meter from the rail. So each her position is defined by a single number x — the distance (in meters) from the left wall.

The coverage will consist of n scenes. The first scene will be at the position x_1, so the reporter will start the live coverage there. The second scene will be at the position x_2, so she will move from x_1 to x_2 between the scenes. The third scene will be at the position x_3 and so on. In total the reporter will successively visit n positions $x_1, x_2, \ldots, x_n$, the j-th scene will be at x_j.

For sure it is a bad idea to shoot the reporter if she is too far from the camera. It should be at most r meters to the reporter at the moments of scenes.

Write a program to find the minimum total distance the camera will move tomorrow. You may assume that both the camera and the reporter move only along their lines, the maximum speed of the camera is enough to keep up with the reporter.



Input

The first line of the input contains three integer numbers m ($2 \leqslant m \leqslant 3 \cdot 10^5$), n ($2 \leqslant n \leqslant 3 \cdot 10^5$), r ($1 \leqslant r \leqslant 1000$), where m is the number of positions where the camera can shoot, n is the number of scenes and r is the maximum distance between the camera and the reporter in the moments of scenes.

The second line contains valid camera positions: m real numbers $c_1, c_2, \ldots, c_m$ ($0 \leqslant c_i \leqslant 10^6$). The numbers are given with exactly one digit after the decimal point. They are given in the strictly increasing order ($c_i < c_{i+1}$ for each $1 \leqslant i < m$).

The third line contains positions of scenes in the chronological order: n real numbers $x_1, x_2, \ldots, x_n$ ($0 \leqslant x_j \leqslant 10^6$). The numbers are given with exactly one digit after the decimal point. These numbers are not necessarily distinct.

It is guaranteed that it is possible to find at least one valid camera position for each scene.

Output

Print a single real number with at least one digit after the decimal point — the minimum total distance the camera needs to move to shoot all the scenes.

Examples

stdin
4 3 4 10.0 20.0 30.0 40.0 31.0 41.0 20.0
stdout
50.0

stdin
5 11 3 2.1 6.0 8.2 11.0 13.3 4.5 7.5 3.5 7.9 10.7 13.8 12.6 9.0 6.0 5.5 10.5
stdout
11.7

stdin
6 8 4 1.7 3.2 4.7 6.2 9.2 15.1 3.5 8.2 6.0 4.9 3.2 6.8 7.1 6.0
stdout
3.0

Note

In the first example to shoot the first scene the camera will move to the position $c_3 = 30$ from the initial position $c_1 = 10$. To shoot the second scene it will move to the position $c_4 = 40$. Finally, to shoot the third scene it will move to the position $c_2 = 20$. The total distance the camera will move is 50.

Problem C. Equivalent Cards

Input file:	`stdin`
Output file:	`stdout`
Time limit:	1 second
Memory limit:	256 megabytes
Feedback:	

Jane is playing a game with her friends. They have a deck of round cards of radius 100. Each card has a set of disjoint rectangles strictly within the bounding circle. The rectangles' vertices and the card center have integer coordinates. However, the rectangles' edges are not necessarily parallel to the axes. The value of a card depends on the rectangles in it and equals the sum of areas of all its rectangles. The cards are *equivalent* if they have same values.

The rules are simple. Each player is given a card in the beginning. Then they turn the cards face-up. If any player can spot another player's card equivalent to their own card, the player who first noticed the equivalency gives their cards to another player and draws a new card from the deck face-up. The game ends when there are no equivalent cards among players, or when a player needs to draw a card, but the deck is empty. The player with the least number of cards wins.

Of course, Jane never cheats. She also believes that her friends don't cheat as well. But the game is so dynamic, that there is no time to verify if some cards are equivalent, i.e. have the same total area of rectangles. So, if somebody makes a mistake and claims that two cards are equivalent while they are not, other players may leave it unnoticed and keep playing.

To avoid this, Jane decided to use her web-camera and write a program to find equivalent cards. She noticed that cards on the pictures from the camera taken under some angle look like ellipses and rectangles look like parallelograms, but she is not good at geometry. Also the images are scaled, shifted and rotated, so the problem seems to be too hard to Jane. She asked you to write an algorithm to find equivalent cards.

Fortunately, you know a good image processing library which does the hardest work of finding figures for you. Your task is, given the library output, find all equivalence classes and for each card tell which class it belongs to. An equivalence class is a set of cards having the same sum of areas of rectangles.

Input

The first line of the input contains n ($1 \leqslant n \leqslant 100$), where n is the number of pictures. Each picture contains a single card in it.

Then n descriptions of pictures follow. The description of a picture consists of several lines. The first two lines of the description specify an ellipse — a card boundary on the picture. The first line contains coordinates of two most distant opposite points on the ellipse (any pair of opposite points in case of a tie). The second line contains the coordinates of two closest opposite points on the ellipse (any pair of opposite points in case of a tie), the distance between them is at least 1. These four points completely determine the ellipse. The following line contains r_i ($1 \leqslant r_i \leqslant 4$) — the number of rectangles on the card. The following r_i blocks contain the coordinates of four points, a pair of coordinates per line. Each point is a corner of a corresponding parallelogram on the picture in the clockwise or counter-clockwise order.

All coordinates are floating point numbers between -1000 and 1000, inclusively. They are given with an accuracy of exactly 8 digits after the decimal point.

Output

Print the only line containing the sequence $f_1, f_2, \ldots, f_n$ describing the equivalence classes. It should be true that $f_i = f_j$ if and only if the i-th and the j-th cards are equivalent. You may use any integer values between 1 and 100 inclusive.

Examples

stdin
3 -10.00000000 0.00000000 10.00000000 0.00000000 0.00000000 -10.00000000 0.00000000 10.00000000 2 5.00000000 5.00000000 5.00000000 6.00000000 6.00000000 6.00000000 6.00000000 5.00000000 3.00000000 2.00000000 3.00000000 1.00000000 4.00000000 1.00000000 4.00000000 2.00000000 -8.00000000 -6.00000000 8.00000000 6.00000000 6.00000000 8.00000000 -6.00000000 -8.00000000 1 1.00000000 0.00000000 0.00000000 1.00000000 -1.00000000 0.00000000 0.00000000 -1.00000000 -10.00000000 0.00000000 10.00000000 0.00000000 0.00000000 -5.00000000 0.00000000 5.00000000 1 1.00000000 1.00000000 0.00000000 1.00000000 0.00000000 -1.00000000 1.00000000 -1.00000000
stdout
1 1 2

Note

You can assume that the perspective effect on pictures from the camera is negligible. Thus, each picture of a card is the card's orthogonal projection onto a plane.

Problem D. Grumpy Cat

Input file:	`stdin`
Output file:	`stdout`
Time limit:	1 second
Memory limit:	256 megabytes
Feedback:	

This problem is a little bit unusual. Here you are to implement an interaction with a testing system. That means that you can make queries and get responses in the online mode. Please be sure to use the stream flushing operation after each query's output in order not to leave part of your output in some buffer. For example, in Pr++ you've got to use the `fflush(stdout)` function, in Java — call `System.out.flush()`, and in Pascal — `flush(output)`.

Fall is coming and it's time to elect a new governor in Cattown. After looking at the results of the past elections the citizens decided to nominate a new candidate — Grumpy Cat. They said, all the previous governors used to spend the Cattown's budget on their own needs and did nothing helpful for the town. Grumpy Cat, they said, can't steal more money than he needs to feed himself.

The administration of Cattown refused to register Grumpy as a new candidate and it caused a lot of discontent. People organized the biggest demonstration in the history of Cattown.

Your friend Eugene is a correspondent of a local newspaper. He has a work assignment to talk to demonstrants to understand their main demands. After talking with several people Eugene realized that each demonstrant either wants Grumpy Cat to be the new governor or doesn't want Grumpy Cat to be a registered candidate or just wants to take part in the demonstration and doesn't have any requirements at all. Let's call these 3 types of people grumpy-lovers, grumpy-haters and grumpy-neutral respectively. It is known that there is at least one grumpy-lover and at least one grumpy-hater among the demonstrants.

Eugene decided to find a type of each demonstrant. He can choose a group of demonstrants and ask them for a number of Grumpy Cat supporters among the group. The people of Cattown don't like journalists, reporters and correspondents. Each time Eugene asks a group of people, they proceed as follows:

1. They talk to each other to understand who is who there. For sure, all grumpy-lovers are counted as Grumpy Cat supporters and grumpy-haters are not.
2. If there are more grumpy-lovers than grumpy-haters in this group, all grumpy-neutrals express support of Grumpy Cat at the time of this survey.
3. If there are more grumpy-haters than grumpy-lovers in this group, all grumpy-neutral people do not support Grumpy Cat at the time of this survey.
4. If there are equal numbers of grumpy-haters and grumpy-lovers in this group, each grumpy-neutral demonstrant at the time of this survey decides to support or not to support independently on his own account.
5. After all grumpy-neutral people decide their position regarding Grumpy Cat, somebody tells the correspondent the number of supporters.

A fact that a grumpy-neutral person has supported or hasn't supported Grumpy Cat doesn't affect its decision in the future. Eugene can think that the surveys are completely independent.

Eugene doesn't have much time to do too many surveys. He can do at most $\lfloor \frac{2\pi n}{3} \rfloor$ surveys, where π is the ratio of a circle's circumference to its diameter and $\lfloor x \rfloor$ is x rounded down. It seems too difficult for him! Help him and write a program to interact with demonstrants to find the type of each demonstrant. Eugene knows that there is at least one grumpy-lover and there is at least one grumpy-hater among the demonstrants.

Input

To read answers to the queries your program should use standard input.

The input starts with a line containing a positive integer t — the number of testcases in the test.

Each testcase starts with a line containing a single integer n ($2 \leqslant n \leqslant 100$) — the number of demonstrants. The following lines will contain one integer each — the number of Grumpy Cat supporters according to the preceding survey.

The total number of demonstrants in all testcases in the test doesn't exceed 1000.

Output

The program should use the standard output to print queries. Each query describes a single survey. It should contain exactly two lines: the first line should contain g ($1 \leqslant g \leqslant n$) — the number of demonstrants in an interviewed group, the second line should contain g distinct positive integer numbers $t_1, t_2, \ldots, t_g$ ($1 \leqslant t_i \leqslant n$) — the numbers of the demonstrants in the group. The demonstrants are numbered from 1 to n.

After your program found the types of all the demonstrants it should print exactly two lines: the first line should contain the only integer `-1`, the second line should contain exactly n integer numbers $f_1, f_2, \ldots, f_n$ ($1 \leqslant f_i \leqslant 3$), where $f_i = 1$ if the i-th demonstrant is a grumpy-lover, $f_i = 2$ if the i-th demonstrant is a grumpy-hater and $f_i = 3$ in case of the i-th demonstrant is grumpy-neutral.

After the output of each line your program should execute the `flush` operation. Use single space to separate integers in a line. Each line should end with end-of-line.

The program should write queries for the succeeding testcase after printing two lines described in the second paragraph for the previous testcase. The program should terminate normally after the last testcase.

Examples

stdin	stdout
2 3 2 1 0 5 0 3	3 1 2 3 3 1 2 3 1 2 -1 1 2 3 3 2 4 3 3 5 3 1 -1 1 2 3 2 3

Note

The example illustrates only the format of interaction, you can assume that the correct answers in the example are just guessed.

Problem E. Scientific Battalion

Input file:	`stdin`
Output file:	`stdout`
Time limit:	5 seconds
Memory limit:	256 megabytes
Feedback:	

Colonel Kruglyakovski always follows modern tendencies. Recently he decided to organize scientific battalion in his army. Huge amount of work has been done to find the best of the bests. And now colonel has n clever soldiers standing in front of him in a row. So almost everything is ready for further practice. But there is one problem — the soldiers are standing ordered as usual by their height. This is totally unacceptable for a scientific battalion. So colonel decided to order them by their IQ from lower to higher.

To achieve this, Kruglyakovski performs the following procedure m times:

- He walks along the row starting from position 1 and when he doesn't like the way the soldiers are ordered he immediately stops. It is known that during the j-th walk-through he stopped in front of position p_j. Numbers p_j depend only on the mood of the colonel and don't obey any rule.
- Kruglyakovski tells that he is not satisfied with the existing order and gently asks the soldier who is currently standing at position p_j to make the order better. And loudly adds that otherwise all the scientific battalion will dig trenches for the rest of the day.
- The soldier who is currently at position p_j tries to improve the situation. He gives a command to the soldiers at positions $p_j, p_j+1, \ldots, n$ to step out of the row, sort by their IQ from lower to higher and then step back to the unoccupied positions. This would work fine for a regular battalion. But for the scientific battalion the situation is a bit trickier — each soldier doesn't want to follow the order of the person with a lower IQ. As a result, the soldiers with an IQ greater than IQ of the soldier at position p_j will not step out of the row. Thus only soldiers at positions $p_j, p_j+1, \ldots, n$ with IQ less or equal than IQ of the soldier at p_j will follow the order and will take part in the rearrangement.
- The colonel waits until the described rearrangement completes and returns to the beginning of the row. After that, the j-th walk-through ends.

Being the adjutant of the colonel you were given a task to evaluate how close each of the obtained arrangements is to the perfect one. It was decided that the suitable metric for that would be the *irregularity* of the arrangement — the number of such pairs (x, y) that $x < y$, but IQ of the soldier at position x is greater than IQ of the soldier at position y. Now you need to find out the irregularity of the initial arrangement and the irregularity of the arrangements obtained during the colonel's walk-throughs.

Input

The first line contains two integers n and m ($1 \leqslant n \leqslant 5 \cdot 10^5$, $1 \leqslant m \leqslant 5 \cdot 10^5$) — the number of soldiers and the number of walk-througs the colonel made.

The next line contains n integers $a_1, a_2, \ldots, a_n$ ($1 \leqslant a_i \leqslant 10^6$), where a_i is IQ of the soldier who initially stands at position i. The values are not necessarily distinct.

The next m lines contain one integer each. The j-th line contains number p_j ($1 \leqslant p_j \leqslant n$) — the number of the position where the colonel stopped during the walk-through number j. The positions $p_1, p_2, \ldots, p_m$ are not necessarily distinct.

Output

Print in the first line the irregularity of the initial arrangement. Then for each walk-through print in a separate line the irregularity of the arrangement which was obtained by the end of the walk-through.

Examples

stdin	stdout
4 4 120 130 140 110 2 3 4 1	3 2 1 1 0

Note

In the example the irregularity of the initial arrangement is 3, it is formed by pairs $(1, 4)$, $(2, 4)$ and $(3, 4)$.

During the first walk-through the colonel stops in front of the second soldier. The soldier at position 3 doesn't take part in the rearrangement because his IQ is greater than IQ of the soldier who gives the command. Thus only the second and the fourth soldiers rearrange. So by the end of walk-through the order is $(120, 110, 140, 130)$ with irregularity 2.

After the second rearrangement the order is $(120, 110, 130, 140)$ with irregularity 1.

The third walk-through doesn't change the order, so irregularity is also 1.

After the fourth rearrangement all soldiers got sorted by their IQ, so irregularity is 0.

Problem F. Judging Time Prediction

Input file:	stdin
Output file:	stdout
Time limit:	2 seconds
Memory limit:	256 megabytes
Feedback:	

It is not easy to predict. Do you know that the modern weather prediction is marginally better than to use the previous day weather as a prediction for the next day?

This problem is about judging a submission on a programming contest like ACM-ICPC. Suppose there is a single submission for the problem containing n tests. Based on the previous statistics for each test we know t_i — the expected time to judge on it and p_i — the probability of a submission to pass the test.

There are m judging machines in the system. They are completely independent and are used to judge a submission on multiple tests at the same time. They work as follows. Each time a judging machine has no task (has just started or finished processing the previous job), it chooses the unprocessed test with the smallest number and judges submission on the chosen test. If it was judged on the i-th test, then after time t_i the verdict will be known. As it was written above, the probability that a submission passes the test is p_i.

Let's assume that the judging process starts at the same moment when all the judging machines start. So at this moment the first $\min(m, n)$ tests start on the judging machines.

You know that on ACM-ICPC contests it is not necessary to judge a submission on all tests. The judging process will be aborted at the first moment when two conditions are met at the same time:

- the solution was judged on some test (say, x) and didn't pass it,
- the solution was judged on all the tests $1, 2, \ldots, x - 1$ and passed all of them.

Naturally, if all the tests are judged and passed the judging process ends too.

Write a program to print the expected time to judge a submission. It means that you are to find the mathematical expectation of the judging time according to the specified model.

Input

The first line of the input contains two integer numbers n and m ($1 \leqslant n \leqslant 3 \cdot 10^5, 1 \leqslant m \leqslant 3 \cdot 10^5$) — the number of tests and judging machines. The following n lines contain pairs of integers t_i and real numbers p_i ($1 \leqslant t_i \leqslant 100, 0 < p_i < 1$), where t_i is the time required to judge the i-th test and p_i is the probability to pass the i-th test. The values p_i are given with no more than 4 digits after the decimal point.

Output

Print the only real number e — the expected time to judge the submission. Print it with at least 4 digits after the decimal point.

Examples

stdin	stdout
2 1 1 0.5 2 0.5	2.0000000000
2 2 1 0.5 2 0.5	1.5000000000

Problem G. Expression Evaluation

Input file:	`stdin`
Output file:	`stdout`
Time limit:	2 seconds
Memory limit:	256 megabytes
Feedback:	

Berland scientists have proven that in the nearest future each computer will have k processors! So the major problems in Berland computer science are about parallel algorithms. One of them is about expression evaluation.

You are given an expression consisting of integer numbers, variables, operation signs (addition, subtraction, multiplication) and parenthesis. Each operation is binary, no unary operation is allowed. Initially only numbers and variables are evaluated. If both operands for an operation are evaluated it is possible to evaluate the result of the operation in one time unit. Since k processors are available it is allowed to do at most k independent evaluations in the same time unit. All of them will be processed in the same moment, so each operand for each operation should be evaluated before the time unit starts.

It is not allowed to simplify/modify the expression or use some properties of operations except their priorities. For example, multiple processors unable to evaluate the expression `a+b+ ...+ z` faster than in 25 time units. Also even if different parts of the expression contain identical subexpressions, each of subexpressions should be evaluated independently. For example, if the given expression is `-4+(a+2)*(a+2)` and $k = 1$, the expression can be evaluated in 4 time units. Here are the evaluations per time units:

1. evaluate the left subexpression `a+2`,
2. evaluate the right subexpression `a+2`,
3. evaluate the subexpression `(a+2)*(a+2)`,
4. evaluate the expression `-4+(a+2)*(a+2)`.

But if the number of processors $k = 2$, the 3 time units are enough:

1. in the same time evaluate both subexpressions `a+2`,
2. evaluate the subexpression `(a+2)*(a+2)`,
3. evaluate the expression `-4+(a+2)*(a+2)`.

The order of evaluation should match operation priorities, addition and subtraction have the same priority so a chain of them is evaluated from the left to the right.

Write a program to find the minimum time to evaluate the given expression.

Input

The first line of the input contains the number of processors k ($1 \leqslant k \leqslant 10^5$). The second line contains the given non-empty expression. It consists of integer numbers, variables, operator signs (addition, subtraction, multiplication) and round brackets. The given expression is correct in a typical mathematical sense. The integer numbers are between -2147483648 and 2147483647, inclusive. The variables are words containing lowercase Latin letters. Their lengths are between 1 and 10, inclusive. The expression length doesn't exceed $3 \cdot 10^5$ characters.

Output

Print the minimum number of time units required to evaluate the given expression on k processors.

Examples

stdin
100 a+b+c+d+e+f+g+h+i+j+k+l+m+n+o+p+q+r+s+t+u+v+w+x+y+z
stdout
25

stdin
1 -4+(a+2)*(a+2)
stdout
4

stdin
2 -4+(a+2)*(a+2)
stdout
3

Note

The expression in the input satisfies the following BNF as a symbol exp.

- ⟨exp⟩ ::= ⟨number⟩ | ⟨variable⟩ | (⟨exp⟩) | ⟨exp⟩+⟨exp⟩ | ⟨exp⟩-⟨exp⟩ | ⟨exp⟩*⟨exp⟩
- ⟨number⟩ ::= integer number between -2147483648 and 2147483647
- ⟨variable⟩ ::= sequence of lowercase Latin letters

Problem H. Password Service

Input file:	stdin
Output file:	stdout
Time limit:	1 second
Memory limit:	256 megabytes
Feedback:	

Startups are here, startups are there. Startups are everywhere! Polycarp would like to have his own startup, too. His business idea is a password service. Have you noticed how many hours you spent trying to create a password?

Polycarp's service will help clients to create a password by requirements. He is thinking about a freemium business model of monetization. He doesn't know what it is, but he likes the word "freemium".

The first release of Polycarp's startup should have a simple form with two fields. The first field is for n, where n denotes that the required password can consist of only first n lowercase letters from the Latin alphabet. The second form field is for string s containing the characters '<', '>' and '='. The sign in position i denotes the comparison result of the i-th and the $(i+1)$-th character in the password. So if the length of s is l then the required password should consist of exactly $l+1$ lowercase letters.

Polycarp offers you a great position in his startup, he offers you to become the CTO. Polycarp can't offer you a great salary (just only \$1), but he will give you so many stock options that in case of IPO exit you will be a millionaire! Why not? So your task is to write a program to generate a password containing some of the first n lowercase letters of the Latin alphabet and which has s as a result of comparisons of consecutive characters.

Input

The first line of the input contains an integer number n ($1 \leqslant n \leqslant 26$), where n denotes that the required password should contain only lowercase letters from the first n letters of the Latin alphabet. The second line contains the string s (the length of s is between 1 and 5000, inclusive), where s consists of the characters '<', '>' and '='. The i-th character stands for the result of comparison of the i-th and the $(i+1)$-th characters of the password.

Output

Print the required password or -1 if it doesn't exist. You may print any answer in case of multiple answers.

Examples

stdin	stdout
5 =<>	bbdc

Problem I. Plugs and Sockets

Input file:	stdin
Output file:	stdout
Time limit:	2 seconds
Memory limit:	256 megabytes
Feedback:	

The Berland Regional Contest will be held in the main hall of the Berland State University. The university has a real international status. ThatвЃ™s why the power sockets in the main hall are not of the same type. Some power sockets use the Berland standard of 330 volts at 40 Hz, but other sockets use the Beuropean standard of 125 volts at 60 Hz.

The technical committee has n computers of three types. The computers of the first type have power plugs to plug them in Berland sockets (of 330 volts), the computers of the second type have plugs to plug them in Beuropean sockets (of 125 volts). The most universal type is the third type, they can be plugged into any socket, it doesn't matter if the socket uses the Berland standard or the Beuropean standard.

Also the computers differ by power consumption, the i-th computer consumes w_i watts per hour.

The technical committee has to solve a difficult problem. Which computers should they use and how to plug them in the order to maximize the number of plugged computers? A single socket can be used for at most one plug. If there are many ways to choose the maximum number of computers to plug, the technical committee wants to find the way to minimize the total power consumption of the chosen computers.

Input

The first line of the input contains n, a and b ($1 \leqslant n \leqslant 5000; 0 \leqslant a, b \leqslant 5000$) — the number of computers the technical committee has, the number of Berland standard sockets and the number of Beuropean standard sockets in the hall. The following n lines contain computers' descriptions, one description per line. Each description is a pair of two positive integer numbers t_i and w_i ($1 \leqslant t_i \leqslant 3; 1 \leqslant w_i \leqslant 5000$) — the type of the i-th computer and its power consumption.

Output

On the first line print the maximum number of computers that can be plugged and the required minimum total power consumption. Then print a single line for each plugged computer with two integer numbers j and f_j ($1 \leqslant j \leqslant n; 1 \leqslant f_j \leqslant a + b$) meaning that the j-th computer should be connected to the f_j-th socket. The computers are numbered from 1 to n in the order of the input and sockets are numbered from 1 to $a + b$ in such way that the first a sockets use the Berland standard and the sockets $a + 1$, $a + 2$, ..., $a + b$ use the Beuropean standard. Print the lines in any order. If there are multiple answers, print any of them.

Examples

stdin	stdout
5 1 2 1 2 1 1 3 10 2 20 2 15	3 26 2 1 5 2 3 3

Problem J. Contest, Another Contest and Train

Input file:	`stdin`
Output file:	`stdout`
Time limit:	2 seconds
Memory limit:	256 megabytes
Feedback:	

Congratulations, you've made it to the Godeforces Open World Finals this year! Now you are in the train en route to the finals. The only problem is that Coogle Code Jam Online Round 3 is starting right now. And you're not going to miss that one as well!

So you face the following challenge: you need to solve the contest on the train using your laptop and having only intermittent internet connection via the mobile phone. Lucky for you, you had a stable connection right at the moment the contest started and you were able to download the statements of all n problems.

For each problem i you know the score s_i you will receive for solving it and the number of minutes t_i you need to write a solution for it. You don't need a connection while writing a solution. You need exactly one minute to submit any problem, and you do need a connection during this minute. Thus, you need exactly $t_i + 1$ minutes to write a solution and submit it.

You also know that during the contest there will be exactly m "time windows" during which you will have stable internet connection. The j-th "window" spans from the l_j-th minute up to the r_j-th minute. You can submit solutions only in the minutes which are in the stable internet connection "time windows".

You need to find a strategy which will bring you the best score possible. You may write and submit solutions in an arbitrary order. For instance, you can first solve several problems and then submit them one by one. Also at any time you can interrupt your work on a current problem and submit another one you prepared earlier. You can change the work you are doing only between minutes, so each minute you are doing one type of work: writing some solution, submitting some solution or doing nothing. Obviously, you can't submit a problem before you write a solution for it.

Input

The input starts with a line containing a pair of integers n and m ($1 \leqslant n \leqslant 30$, $1 \leqslant m \leqslant 1000$), where n is the number of problems in the problemset and m is the number of "windows" during which you will have internet connection. The problems are numbered from 1 to n.

The following n lines contain the descriptions of problems, one description per line. The i-th description contains two integers s_i, t_i ($1 \leqslant s_i, t_i \leqslant 10^4$), where s_i is the score you will receive for solving problem i, and t_i is time in minutes you need to write a solution for the i-th problem.

Then follow m lines describing "windows", one description per line. The j-th description contains two integers l_j and r_j ($0 \leqslant l_j < r_j \leqslant 10^4$), where l_j is the time when the j-th "window" starts and r_j is the time when the j-th "window" ends, so in the j-th "window" you can submit a solution in minutes $l_j, l_j + 1, \ldots, r_j - 1$. The contest minutes are numbered from 0. You may submit several solutions during a single "window". No two "windows" share a minute.

Output

Print two integers c and k on the first line of the output, where c is the maximum score possible and k is the number of problems you need to solve in order to achieve the score. On the following k lines describe the problems you will solve. Each line should contain two integers — the number of problem in the problemset and the time at which you are going to submit a solution for it. Print the problems in the order you are going to submit them. If there are several solutions, print any of them.

Examples

stdin	stdout
3 2 1 1 2 2 3 3 0 1 3 5	3 1 3 4
6 3 2 3 1 2 3 2 4 2 3 3 4 2 8 10 2 3 14 15	14 4 6 2 5 8 4 9 3 14

Problem K. Road Work

Input file:	stdin
Output file:	stdout
Time limit:	2 seconds
Memory limit:	256 megabytes
Feedback:	

There are a lot of water pipes under the main street of City S. The pipes and the pavement are very old and have to be repaired. The main street consists of n parts, which are sequentially numbered from 1 to n.

For each part of the main street the city community services have determined the number of days during which this part can stay without repair. For part i this number is equal to d_i. It means that if part i is not repaired by the end of day d_i, a geyser starts spurting up from under this part.

A geyser on the main street is not the thing the city authorities want to happen. That's why they have found money for road work and are going to hire several repair brigades, equipped with heavy machinery. At the beginning of day number 1 each brigade starts working on its own part. In a single day each brigade repairs exactly one part of the street (it is impossible for two or more brigades to work on the same part). At night a brigade moves to one of the neighboring parts. It is not allowed to move to the part that is already repaired (no matter by which brigade). It means that each brigade either always moves from part i to part $i + 1$, or always moves from part i to part $i - 1$. If a brigade completes the task assigned by authorities, it stops working at the end of a day and never moves further. It is possible that different brigades will eventually repair different number of road parts.

Your task is to help the city authorities to save money. Find out the minimum number of brigades that is enough to hire in order to repair all parts of the street in time. Please, provide also the work schedule for the brigades.

Input

The first line contains integer n ($1 \leqslant n \leqslant 3 \cdot 10^5$) — the number of parts of the main street. The second line contains n integers $d_1, d_2, \ldots d_n$, separated by spaces ($1 \leqslant d_i \leqslant 10^6$).

Output

In the first line print the minimum number of brigades that is enough. Then for each brigade in a separate line print two integers: the number of the part where the brigade starts working, and the number of the part where it finishes working. If there are several optimal schedules, output any of them.

Examples

stdin	stdout
8 3 4 1 1 3 2 1 3	4 3 1 4 5 6 6 7 8

Problem L. Stock Trading Robot

Input file:	stdin
Output file:	stdout
Time limit:	1 second
Memory limit:	256 megabytes
Feedback:	

CyberTrader is an all-in-one trading solution for investment banks, quantitative hedge funds and proprietary trading groups. It has only one drawback — it is not implemented yet.

You are working on implementing a simple algorithm to buy/sell shares. It should work as follows. Initially a robot has d dollars and doesn't have any shares. The robot's behaviour is defined by two positive integer numbers a and b, their role is explained below.

Starting from the second day, every day the robot analyzes a new share price comparing it with the previous share price. If the price increases the robot buys shares — it buys as many shares as it can but not more than x. Actually, x is not a constant and depends on the number of consecutive increases: $x = a$ for the first increase, $x = 2a$ for two increases in a row, and so on, i.e. $x = ka$ for k consecutive increases. Surely, the robot can buy only non-negative integer number of shares and the number depends on the money it has and on x.

If the price decreases the robot sells shares — it sells as many shares as it has but not more than y. Actually, y is not a constant and depends on the number of consecutive decreases: $y = b$ for the first decrease in a row, $y = 2b$ for two decreases in a row, and so on, i.e. $y = kb$ for k consecutive decreases.

If the price doesn't change the robot does not buy or sell any shares.

Write a program for the robot to simulate the above algorithm.

Input

The first line of the input contains four positive integers n, d, a and b ($1 \leqslant n, d \leqslant 10^5$, $1 \leqslant a, b \leqslant 10$), where n is the number of days to simulate the algorithm. The following line contains sequence of positive integers $p_1, p_2, \ldots, p_n$ ($1 \leqslant p_i \leqslant 10^5$), where p_i is the share price on the i-th day.

It is guaranteed that there will be no overflow of the 32-bit signed integer type, so feel free to use type `int` (in C++ or Java) to store the number of dollars and shares.

Output

Print the number of dollars and the number of shares the robot will have after n days.

Examples

stdin	stdout
5 10 1 2 1 2 3 4 5	2 3
4 3 1 1 1 2 100000 99999	100000 0

Note

In the first test the robot will buy one share on the second day and two shares on the third day.

In the second test the robot will buy one share for \$2 and later will sell it for \$99999. Since there were \$3 initially, the total sum at the end will be \$100000.

Problem M. Winnie-the-Pooh Needs Help

Input file:	`stdin`
Output file:	`stdout`
Time limit:	4 seconds
Memory limit:	256 megabytes
Feedback:	

You know Winnie-the-Pooh has only sawdust in his head, that's why he isn't afraid of anything. To tell the truth he isn't afraid of anything except *heffalumps* and *woozles*. If he sees a woozle sitting on a trail, Winnie-the-Pooh will not use the trail and will find a roundabout route.

There are n meadows in the Hundred Acre Wood, and they are connected with m one-way trails. The j-th trail connects two distinct meadows a_j, b_j and it is possible to walk it in the direction from a_j to b_j. If Winnie-the-Pooh wants to go from the a_j-th meadow to the b_j-th meadow but sees a woozle on the j-th trail, he will go round the trail by going the shortest route connecting a_j and b_j which doesn't go through the j-th trail.

Winnie-the-Pooh is very curious and cautious at the same time. That is why for each trail j he wants to know the length of the shortest route connecting a_j and b_j which doesn't use the j-th trail. Winnie-the-Pooh is just a toy bear and he can't write a program to find the required value for each trail. Help him and write a program for him.

Input

The input contains one or more testcases. Each testcase starts with a line containing a pair of integers n and m ($2 \leqslant n \leqslant 900; 1 \leqslant m \leqslant 150000$), where n is the number of meadows and m is the number of trails. The meadows are numbered from 1 to n. The following m lines contain the descriptions of trails, one description per line. The j-th description contains two integers a_j, b_j ($1 \leqslant a_j, b_j \leqslant n; a_j \neq b_j$), where a_j and b_j are meadows connected by the j-th trail. It is possible to walk the j-th trail in the direction from a_j to b_j but not vice versa. For each pair (a, b) there is at most one trail from a to b.

The total number of meadows in all testcases doesn't exceed 900, the total number of trails doesn't exceed 150000.

Output

For each testcase print the sequence $f_1, f_2, \ldots, f_m$ on a single line. The value f_j stands for the number of trails in a shortest route which starts at a_j, finishes at b_j and avoids the j-th trail. Print `0` if there is no such route.

Examples

stdin	stdout
3 3 1 2 2 3 1 3 3 3 1 2 2 1 1 3 3 6 1 2 2 1 1 3 3 1 3 2 2 3	0 0 2 0 0 0 2 2 2 2 2 2