Problem A. Nasta Rabbara

Sasha just finished watching yet another season of the infamous TV series "Nasta Rabbara", the best show in Berland. The action takes place in a city called Nasta Rabbara that has a population of n people.

In the beginning of the season, all residents of the city were friendly to each other and lived in peace. But as the season developed, people would start getting angry at each other. Specifically, in each of m series, one pair of people quarreled. Note that a pair of people may have quarreled several times throughout the season.

Obviously, Sasha is quite concerned about the situation in Nasta Rabbara. He picked q segments of the season and now wants to analyze them. For convenience, the segments are numbered with integers from 1 to q. The i-th segment contains episodes from the l_i-th to the r_i-th, inclusively. Note that the different segments may overlap, i.e. have common episodes.

When analyzing a given segment, Sasha watches all its episodes and decides which people are good and which are bad (only taking into account the episodes that belong to the segment). Sasha believes that during the segment, good guys should never quarrel with each other, as well as bad guys should never quarrel with each other. I.e. he would accept only a quarrel between a good and a bad guy. Though, sometimes the storyline of a segment can become very complicated, so it's not even possible to classify all characters as either good or bad.

Your task is to help Sasha understand the situation in Nasta Rabbara. For each segment of the season, Sasha needs to determine whether there exists a way to classify all its people as either good or bad, so that:

- no two good guys quarreled over this segment of the season;
- no two bad guys quarreled over this segment of the season.

Input

The first line contains three space-separated integers n, m and q ($2 \le n \le 10^5$; $1 \le m, q \le 10^5$), denoting the number of people, the number of episodes, and the number of segments Sasha is interested in correspondingly.

The i-th of the next m lines contains a pair of space-separated integers x_i, y_i ($1 \le x_i < y_i \le n$), denoting that in the i-th episode person x_i and person y_i quarrel with each other. Assume that all characters of the season are numbered by consecutive integers from 1 to n.

The j-th of the next q lines contains a pair of space-separated integers l_j, r_j ($1 \le l_j \le r_j \le m$), denoting the boundaries of the j-th segment.

Output

Print q lines. In the j-th line print the word "`Possible`" if there is a way to classify people as either good or bad considering only the episodes from the l_j-th to the r_j-th inclusive. If there is no such a way, print the word "`Impossible`". All words should be printed without quotes.

Examples

standard input	standard output
4 3 6 1 2 2 3 1 3 1 1 2 2 3 3 1 2 2 3 1 3	Possible Possible Possible Possible Possible Impossible
4 5 6 1 2 2 3 1 3 1 4 1 2 1 1 1 2 1 3 2 4 2 5 1 5	Possible Possible Impossible Possible Impossible Impossible

Note

In the first example the answer is "`Possible`" for all segments, except for the segment that forms a quarrel triangle between people 1, 2 and 3.

Problem B. Colored Blankets

Recently Polycarp has opened a blankets store and he faced with many challenges.

He has got k blankets. A blanket has two sides, each of k blankets has at most one colored side. So either both sides are uncolored or one side is colored and the other one is not. If a side is colored, one of n possible colors is used. It is known that k is divisible by n.

Polycarp wants to color all uncolored sides in such a way that:

- each blanket will have both sides colored (one color per side), the colors can be the same or different, all the used colors are from n possible colors;
- it will be possible to compose n kits with $\frac{k}{n}$ blankets in each kit that blankets in each kit are *identically colored*.

It is allowed to turn over blankets to determine that they are *identically colored*: for example, red-blue and blue-red blankets are *identically colored*. Blankets in different kits can be *identically colored*.

Input

The first line contains two integer numbers k and n ($1 \le n \le k \le 1000$) — number of blankets and colors. It is guaranteed that k is divisible by n. The second line contains a sequence of integers $c_1, c_2, \ldots, c_k$ ($1 \le c_i \le n$ or $c_i = -1$), where c_i stands for the color of the colored side of the i-th blanket or -1 if it is uncolored.

Output

If there is no solution for the problem, print "`No`" in the first line. Otherwise print a line containing "`Yes`" and k lines describing each blanket. The i-th line should contain a pair of colors (integers in the range $1, 2, \ldots, n$) used for the i-th blanket. You may print colors in pairs in any order.

If there are multiple solutions, print any of them.

Examples

standard input	standard output
6 2 1 1 2 2 -1 2	Yes 1 2 2 1 2 2 2 2 2 1 2 2
8 4 4 -1 1 -1 4 3 -1 -1	Yes 4 1 2 1 2 1 3 1 1 4 3 1 4 1 4 1

Problem C. Component Tree

This problem is a little bit unusual. Here you are to write a program to process queries in online mode. That means that your program will be allowed to read a query only after writing answer for the previous query. 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 C++ you've got to use the `fflush(stdout)` function, in Java — call `System.out.flush()`, and in Pascal — `flush(output)`.

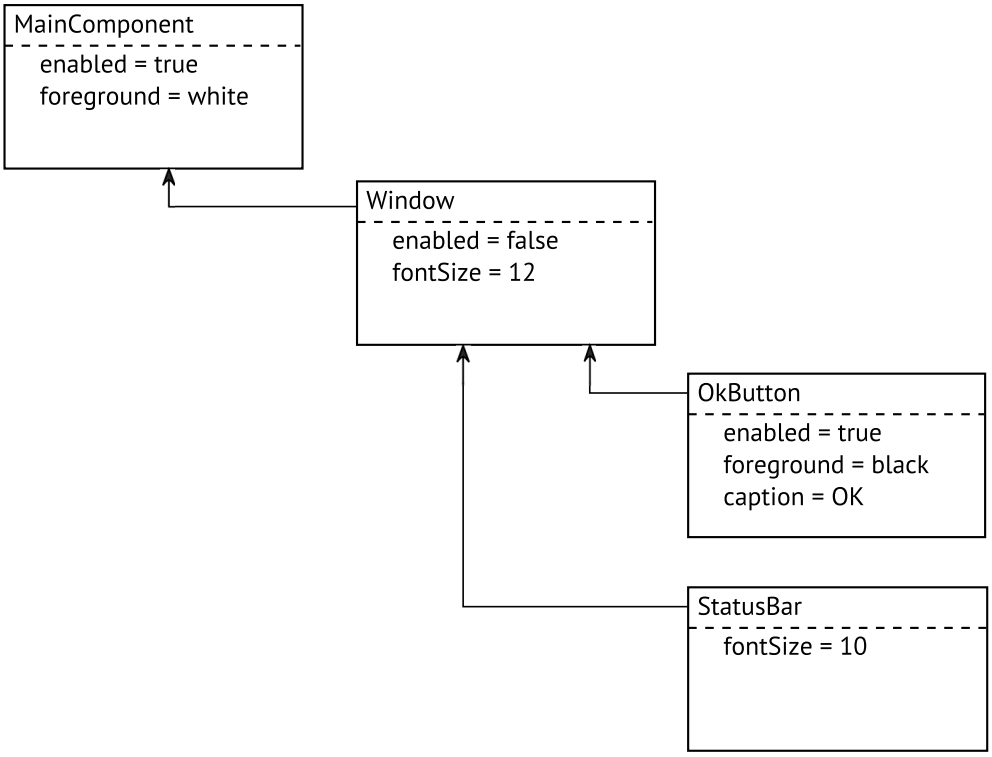
Imagine a complex application that consists of n components (the application itself is not essential in this problem, so we omit its description). All components form a tree-like hierarchy: each component except one (let's call this special component a *main component*) is contained in exactly one other component (let's call it a *parent component*). For the sake of convenience, let's assume that the components are numbered with integers from 1 to n with the main component having the number 1.

Each component has a set of properties associated with it. Each property is given in the form ‘‘`KEY=VALUE`’’. All property keys within a single component are distinct.

You are given the component tree together with all properties. Your task is to handle several queries for finding the property of a component. Each query is given as a pair (c, key), which means that you have to find the value of the property key for the component number c.

To answer the query, you have to follow this algorithm:

1. If the value of the property is defined for the component c, report this value as an answer. Otherwise, go to step 2.
2. If the component number c is not *main*, continue search in the *parent component* (set c equal to the number of *parent component*) and go to the step 1. Otherwise, the value of the property is not defined and considered to be ‘‘`N/A`’’.



Input

The first line of the input contains an integer n ($1 \le n \le 3 \cdot 10^5$) — the number of components. Then the input contains the descriptions of components 1, 2, ..., n.

Each description starts with a line containing two space-separated integers p_i and k_i ($0 \le p_i \le n$, $k_i \ge 0$) — the number of parent component and the number of properties. For the first component p_i is equal to 0. Each of the following k_i lines contains a single property in the form `KEY=VALUE`, where both `KEY` and `VALUE` are strings of Latin letters and digits with length between 1 and 10 characters each. All keys for a given component are distinct. The keys are case-sensitive, so, for example, properties with keys "`Length`" and "`length`" are considered distinct. The values are also case-sensitive.

The next line contains an integer q ($1 \le q \le 3 \cdot 10^5$) — the number of queries you have to answer. Each of the following q lines contains an integer c_j ($1 \le c_j \le n$) and a string key_j, separated by a single space, where key_j is a string of Latin letters and digits with length between 1 and 10 characters. Note that for each query, starting from the second, you will be able to read it only when you print the answer for the previous query and make a flush operation.

It is guaranteed that the total number of properties does not exceed $3 \cdot 10^5$.

Output

For each query, print a line with the value of the required property. You should make a flush operation after each line.

Examples

standard input	standard output
4 0 2 enabled=true foreground=white 1 2 enabled=false fontSize=12 2 3 enabled=true foreground=black caption=OK 2 1 fontSize=10 5 3 enabled 4 enabled 3 fontSize 4 foreground 2 caption	true false 12 white N/A
1 0 2 width=300 height=500 2 1 width 1 Width	300 N/A

Problem D. Data Center

The startup "Booble" has shown explosive growth and now it needs a new data center with the capacity of m petabytes. Booble can buy servers, there are n servers available for purchase: they have equal price but different capacities. The i-th server can store a_i petabytes of data. Also they have different energy consumption — some servers are *low voltage* and other servers are not.

Booble wants to buy the minimum number of servers with the total capacity of at least m petabytes. If there are many ways to do it Booble wants to choose a way to maximize the number of *low voltage* servers. Booble doesn't care about exact total capacity, the only requirement is to make it at least m petabytes.

Input

The first line contains two integer numbers n and m ($1 \le n \le 2 \cdot 10^5, 1 \le m \le 2 \cdot 10^{15}$) — the number of servers and the required total capacity.

The following n lines describe the servers, one server per line. The i-th line contains two integers a_i, l_i ($1 \le a_i \le 10^{10}, 0 \le l_i \le 1$), where a_i is the capacity, $l_i = 1$ if server is *low voltage* and $l_i = 0$ in the opposite case.

It is guaranteed that the sum of all a_i is at least m.

Output

Print two integers r and w on the first line — the minimum number of servers needed to satisfy the capacity requirement and maximum number of *low voltage* servers that can be bought in an optimal r servers set.

Print on the second line r distinct integers between 1 and n — the indices of servers to buy. You may print the indices in any order. If there are many solutions, print any of them.

Examples

standard input	standard output
4 10 3 1 7 0 5 1 4 1	2 1 4 2
3 13 6 1 6 1 6 1	3 3 1 2 3

Note

In the first example any pair of servers which includes the server 2 is a correct output.

Problem E. Election of a Mayor

The Berland capital is preparing for mayoral election. There are two candidates for the position: the current mayor and his rival. The rival is a serious competitor, and it's not easy for current mayor to win the election.

The candidate will be declared the winner if he wins at *more* than a half of *all* polling stations. The results of the polling stations are supplied independently. The station winner is the candidate who gets *more* than a half of the votes of this station. No candidate is declared a winner of a polling station if both candidates have got the same number of votes at this station. Similarly, no candidate is declared a winner of the election if both candidates won at the same number of polling stations.

All eligible voters are going to take part in the election and have already decided whom to give their vote to. The campaign headquarters of the current mayor has collected data from all n stations of the city, and now for every station it is known how many people will vote for the current mayor and how many people will vote for his opponent.

The results have been disappointing for the current mayor, but his staff came up with a way to win the election. It was suggested to merge some pairs of polling stations in such a way that the current mayor will become the election winner. However, (for the sake of credibility), the proposed plan must comply with two conditions. Firstly, it is possible to merge only the pairs of stations with adjacent numbers (i.e., station j may be merged with either station $j-1$, or with station $j+1$). The resulting station cannot be merged again. Secondly, the number of such mergers for obvious reasons must be as few as possible.

Your task is to help the current mayor's campaign headquarters and produce such plan.

Input

The first line contains single integer n ($2 \le n \le 2 \cdot 10^5$) — the number of the polling stations.

Each of the following n lines contains two integers m_j and r_j ($0 \le m_j, r_j \le 10^5$) — the number of voters at station j who plan to vote for the current mayor and his rival correspondingly.

Output

If current mayor cannot win the election at any condition, print a single number -1 to the first line.

Otherwise, to the first line print an integer u — the minimum number of polling station mergers the current mayor needs to perform in order to win. To each of the next u lines print a pair of integers — the numbers of the merged stations. The pairs as well as the numbers within each pair can be printed in any order. If there are multiple solutions, print any of them.

Examples

standard input	standard output
7 15 8 8 10 14 14 12 13 13 12 21 10 20 30	2 1 2 6 7
2 1 5 5 1	-1
2 10 9 15 7	0

Problem F. Ilya Muromets

Силачом слыву недаром — семерых одним ударом!

From the Russian cartoon on the German fairy tale.

Ilya Muromets is a legendary bogatyr. Right now he is struggling against Zmej Gorynych, a dragon with n heads numbered from 1 to n from left to right.

Making one sweep of sword Ilya Muromets can cut at most k contiguous heads of Zmej Gorynych. Thereafter heads collapse getting rid of empty space between heads. So in a moment before the second sweep all the heads form a contiguous sequence again.

As we all know, dragons can breathe fire. And so does Zmej Gorynych. Each his head has a firepower. The firepower of the i-th head is f_i.

Ilya Muromets has time for at most two sword sweeps. The bogatyr wants to reduce dragon's firepower as much as possible. What is the maximum total firepower of heads which Ilya can cut with at most two sword sweeps?

Input

The first line contains a pair of integer numbers n and k ($1 \le n, k \le 2 \cdot 10^5$) — the number of Gorynych's heads and the maximum number of heads Ilya can cut with a single sword sweep. The second line contains the sequence of integer numbers $f_1, f_2, \ldots, f_n$ ($1 \le f_i \le 2000$), where f_i is the firepower of the i-th head.

Output

Print the required maximum total head firepower that Ilya can cut.

Examples

standard input	standard output
8 2 1 3 3 1 2 3 11 1	20
4 100 10 20 30 40	100

Problem G. FacePalm Accounting

An owner of a small company FacePalm has recently learned that the city authorities plan to offer to small businesses to participate in improving parks and garden squares. However, credible sources informed the FacePalm owner that the loss-making companies will not get such an offer. Moreover, the sources have also told how loss-making companies will be determined.

A company will be considered *loss-making* if for every k contiguous days the total income of the company is negative.

The FacePalm owner discussed the situation with his chief accountant, and they decided to change the report so that the company would look loss-making.

The company report for n days can be represented as a sequence of integers $a_1, a_2, \ldots, a_n$, where a_i is the company income in the day i (negative values correspond to losses).

The accountant can change any values in this sequence, but no updated value can become less than the smallest value in the original report — otherwise the updated report will look absolutely implausible. Besides, the accountant wants the total change of the values to be as small as possible.

We will assume that the total change of the values is $\sum_{i=1}^{n} |\tilde{a}_i - a_i|$, where $\tilde{a}_i$ is the i-th value in the updated report.

Your task is to calculate the minimum required total change of the values and provide the updated report.

Input

The first line contains integers n and k ($1 \le k \le n \le 2 \cdot 10^5$) — the number of days in the report and the number of days in the definition of loss-making company.

The second line contains n space-separated integers $a_1, a_2, \ldots, a_n$ ($-10000 \le a_i \le 10000$), where a_i is the company income in day i.

It is guaranteed that at least one value of a_i is negative.

Output

In the first line print the required minimum total change. In the second line print the corresponding updated report. No value in the updated report can be less than the minimum value of the original report.

If there are multiple solutions, print any of them.

Examples

standard input	standard output
5 4 3 -3 -1 1 2	1 2 -3 -1 1 2
8 3 2 1 -3 -2 4 -3 0 2	3 1 1 -3 -2 2 -3 0 2
4 2 -2 1 -2 1	0 -2 1 -2 1
3 3 -5 6 10	12 -5 -5 9

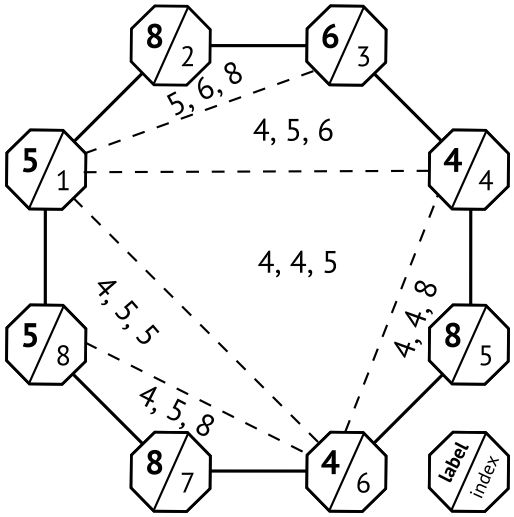
Problem H. Minimal Agapov Code

Triangulation is a decomposition of a polygon into a set of triangles that the triangle vertices are in the vertices of the polygon. The triangles in a triangulation can't have internal common points, and they should cover all the polygon. It is easy to see that triangulation of n-gon always consists of $n-2$ triangles.

You are given a regular polygon with n vertices. The vertices are labeled with integer numbers. *Agapov code* of its triangulation is constructed as follows:

- write a code for each triangle: a sequence of labels of a triangle vertices in non-decreasing order;
- concatenate all triangle codes in such order that the resulting sequence is lexicographically minimal.

A sequence $a = \langle a_1, a_2, \ldots, a_m \rangle$ is lexicographically less than $b = \langle b_1, b_2, \ldots, b_m \rangle$ if there is such index i ($1 \le i \le m$) that $a_i < b_i$ and $a_j = b_j$ for all j ($1 \le j < i$).



Triangle codes are written in triangles, the *Agapov code* is $\langle 4, 4, 5, 4, 4, 8, 4, 5, 5, 4, 5, 6, 4, 5, 8, 5, 6, 8 \rangle$.

Given a labeled regular polygon write a program to find the triangulation with lexicographically minimal *Agapov code*.

Input

The first line contains integer n ($3 \le n \le 5 \cdot 10^5$). The second line contains n space-separated integers: $c_1, c_2, \ldots, c_n$ ($1 \le c_i \le n$), where c_i denotes the label on the i-th vertex. Vertices are indexed clockwise.

Output

Print $n - 2$ lines describing the required triangulation, one line per triangle. In each line print three integers: indices of triangle vertices in order of non-decreasing of their labels. Lines should be ordered in such a way that concatenation of triangle codes in order of output will give the required lexicographically minimal *Agapov code*.

If there are multiple solutions, print any of them.

Examples

standard input	standard output
8 5 8 6 4 8 4 8 5	4 6 1 6 4 5 6 8 1 4 1 3 6 8 7 1 3 2

Problem I. Sale in GameStore

A well-known Berland online games store has announced a great sale! Buy any game today, and you can download more games for free! The only constraint is that the total price of the games downloaded for free can't exceed the price of the bought game.

When Polycarp found out about the sale, he remembered that his friends promised him to cover any single purchase in GameStore. They presented their promise as a gift for Polycarp's birthday.

There are n games in GameStore, the price of the i-th game is p_i. What is the maximum number of games Polycarp can get today, if his friends agree to cover the expenses for any single purchase in GameStore?

Input

The first line of the input contains a single integer number n ($1 \le n \le 2000$) — the number of games in GameStore. The second line contains n integer numbers $p_1, p_2, \ldots, p_n$ ($1 \le p_i \le 10^5$), where p_i is the price of the i-th game.

Output

Print the maximum number of games Polycarp can get today.

Examples

standard input	standard output
5 5 3 1 5 6	3
2 7 7	2

Note

In the first example Polycarp can buy any game of price 5 or 6 and download games of prices 1 and 3 for free. So he can get at most 3 games.

In the second example Polycarp can buy any game and download the other one for free.

Problem J. Getting Ready for VIPC

Eulampius is preparing himself to the Very Important Programming Contest (VIPC). There is a lot of time before the contest, and Eulampius hopes to make headway in his problem solving skills.

Eulampius decided to train by taking part in practice contests. He made a list of n contests that will take place before VIPC. Each contest in the list is characterized by three parameters: the day d_i of the contest, the minimum skill l_i required to take part in the contest and the maximum skill h_i that the participant can reach after taking part in the contest.

The participant skill becomes equal to h_i if he starts taking part in the contest when he's fully rested. If his fatigue equals t by the beginning of the contest, then the participant skill becomes equal to $h_i - t$ after the contest, even if $h_i - t$ is negative or less than his skill before the contest.

Participating in a contest takes much energy, that's why Eulampius can take part in at most one contest during a day. Participating in a contest increases Eulampius' fatigue by Δt (same for all contests).

Each day Eulampius doesn't participate in any contest, his fatigue t becomes equal to $\lfloor t/2 \rfloor$ (divided by two rounded down).

Initially, Eulampius' skill is s, and his fatigue t equals zero. Your task is to determine the maximum skill Eulampius can reach before VIPC, no matter what his fatigue will be.

Input

The first line contains three integers n, s, Δt ($1 \le n \le 2 \cdot 10^5$, $0 \le s \le 10^6$, $0 \le \Delta t \le 10^4$) — the number of contests, Eulampius' initial skill, the amount of fatigue added after a contest.

Each of the next n lines contains three integers d_i, l_i, h_i ($1 \le d_i \le 10^6, 0 \le l_i < h_i \le 10^6$) — the day of the i-th contest, the minimum skill needed to take part in the i-th contest and the maximum skill that can be reached after taking part in the i-th contest.

The contests are ordered by their dates, so $d_i \le d_{i+1}$ ($i = 1, 2, \ldots, n-1$).

Output

In the first line print two integers c and m — the maximum skill Eulampius can reach and the number of contests he needs to take part in.

In the second line print m space-separated integers — the indices of the contests in the order of participation. The contests are indexed starting from 1 in order of their appearance in the input.

If there are multiple solutions, print any of them.

Examples

standard input	standard output
6 12 3 5 8 22 6 22 43 7 40 65 8 40 65 10 63 100 11 62 97	96 4 1 2 4 6
4 5 3 1 5 10 2 5 15 3 15 46 4 10 42	43 2 2 3
4 5 6 1 5 10 2 5 15 3 15 46 4 10 42	41 2 1 4

Note

In the first example the maximum skill that Eulampius can reach is 96:

1. Eulampius' initial skill is 12 and his fatigue is 0.
2. On the day 5 he participates in the first contest. Now his skill is $22 - 0 = 22$ and his fatigue is $0 + 3 = 3$.
3. On the day 6 he participates in the second contest. Now his skill is $43 - 3 = 40$ and his fatigue is $3 + 3 = 6$.
4. On the day 7 he does not participate in any contest. His skill is still 40 and his fatigue is $\lfloor \frac{6}{2} \rfloor = 3$.
5. On the day 8 he participates in the fourth contest. Now his skill is $65 - 3 = 62$ and his fatigue is $3 + 3 = 6$.
6. On the day 9 he does not participate in any contest. His skill is still 62 and his fatigue is $\lfloor \frac{6}{2} \rfloor = 3$.
7. On the day 10 he does not participate in any contest. His skill is still 62 and his fatigue is $\lfloor \frac{3}{2} \rfloor = 1$.
8. On the day 11 he participates in the sixth contest. Now his skill is $97 - 1 = 96$ and his fatigue is $1 + 3 = 4$.

Problem K. Treeland

Treeland is an ancient country located on the territory of the modern Berland many many years ago. It is a well-known fact that Treeland consisted of n cities connected with $n - 1$ bidirectional roads. It was possible to travel from any city to any other one along the roads. The cities were numbered from 1 to n.

Nowadays we do not know what cities were connected by roads directly, but recently archaeologists have found new information that can help to reconstruct the road network of Treeland.

They found an ancient library with the diaries of a well-known traveler Biklouho-Baclay. The diaries contain interesting information: for each city i there is a list of all cities in the order of non-decreasing distance from i. The distances are calculated assuming that each road has length 1, so a distance between cities is the minimum number of roads to travel between them.

Formally, for each city i there is a list $C_i = \langle c_{i,1}, c_{i,2}, \ldots, c_{i,n} \rangle$ containing a permutation of cities (numbers from 1 to n) in such order that $d(i, c_{i,j}) \le d(i, c_{i,j+1})$ for every $j = 1 \ldots n - 1$, where $d(x, y)$ is a distance between cities x and y. Obviously, $c_{i,1} = i$ for each i. The cities that are equidistant from i can be listed in any order.

Your task is to restore a possible road network consistent with the found lists.

Input

The input consists of several test cases. The first line contains integer number t ($1 \le t \le 1000$) — the number of test cases.

Then t blocks follow, each describing a single test case. The first line of a block contains integer number n ($2 \le n \le 2000$) — the number of cities. The following n lines contain lists C_i, one list per line.

It is guaranteed that the required road network exists for each test case. The sum of n over all test cases doesn't exceed 2000.

Output

For each test case print $n - 1$ lines describing roads. Each line should contain a pair of cities connected with a road. You may print the roads and cities in a pair in any order. Print a blank line after the output of a test case.

If there are many solutions, print any of them.

Examples

standard input	standard output
3 2 1 2 2 1 5 1 4 5 3 2 2 3 4 1 5 3 4 2 5 1 4 3 1 5 2 5 4 3 1 2 3 1 3 2 2 1 3 3 1 2	2 1 2 3 3 4 5 4 4 1 2 1 3 1

Problem L. Useful Roads

Berland capital contains n junctions and m roads connecting pairs of junctions. All roads are one-way. As you probably know Berland has two misfortunes: fools and roads.

It is one month before mayoral election. So it is time for current government to make the city better. To show that they care both about the infrastructure and about the budget, the government decided to repair only *useful* roads.

Current mayor thinks that a road from a junction u to a junction v is useful if there exists a simple path from City Hall to some junction containing the road (u, v). A path is called simple if it does not have any repeated junctions, i.e. contains each junction at most once. The City Hall is located at the junction 1.

Help Ministry of Road Transport and Highways to find all useful roads in the city.

Input

The input contains several test cases. Each test case starts with a line containing two integers n, m ($2 \le n \le 2 \cdot 10^5; 1 \le m \le 2 \cdot 10^5$) — the number of junctions and the number of roads in the city. The following m lines contain road descriptions, one description per line. A description consists of a pair of space-separated integers u_i, v_i ($1 \le u_i, v_i \le n; u_i \ne v_i$) meaning that the i-th road leads from the junction u_i to the junction v_i. The junctions are numbered from 1 to n. The City Hall is located at the junction 1.

It is guaranteed that there is at most one road between a pair of junctions in each direction.

There is a blank line after each test case. The sum of n over all test cases in the input doesn't exceed $2 \cdot 10^5$. The same is true for m: the sum of m over all test cases in the input doesn't exceed $2 \cdot 10^5$.

Output

Print two lines for each test case. On the first line print the number of useful roads. The second line should contain the indices of useful roads in ascending order. The roads are indexed from 1 to m in order of appearance in the input. Leave the second line empty if there are no useful roads in the city.

Examples

standard input	standard output
5 7 1 2 5 2 2 3 3 4 4 5 2 4 4 2 2 1 1 2	5 1 3 4 5 6 1 1

Problem M. Variable Shadowing

In computer programming, *variable shadowing* occurs when a variable declared within a certain scope has the same name as a variable declared in an outer scope. The outer variable is said to be shadowed by the inner variable, and this can lead to a confusion. If multiple outer scopes contain variables with the same name, the variable in the nearest scope will be shadowed.

Formally, a declared variable shadows another declared variable if the following conditions are met simultaneously:

- the other variable is declared in outer scope and before (in terms of position in program source code) the declaration of the first variable,
- the other variable is nearest among all variables satisfying the condition above.

Here is an example containing exactly one variable shadowing:

```
/* Prints a+max(b,c) */
int main() {
    int a, b, c;
    cin >> a >> b >> c;
    if (b > c) {
        int a = b; // <-- variable 'a' shadows outer 'a'
        int x = c;
        b = x;
        c = a;
    }
    int x = a + c; // <-- no shadowing here
    cout << x << endl;
}
```

Variable shadowing is permitted in many modern programming languages including C++, but compilers can warn a programmer about variable shadowing to avoid possible mistakes in a code.

Consider a trivial programming language that consists only of scopes and variable declarations. The program consists of lines, each line contains only characters '`{`', '`}`', '`a`' ... '`z`' separated by one or more spaces.

- *Scopes.* A scope (excluding global) is bounded with a pair of matching curly brackets '`{`' and '`}`'. A scope is an inner scope relative to another scope if brackets of the first scope are enclosed by brackets of the second scope.
- *Variables.* A variable declaration in this language is written just as a name of the variable. In addition all variables are lowercase Latin letters from '`a`' to '`z`' inclusive (so there are at most 26 variable names). A variable is declared in each scope at most once.

Given a syntactically correct program (i.e. curly brackets form a *regular bracket sequence*), write an analyzer to warn about each fact of variable shadowing. Warnings should include exact positions of shadowing and shadowed variables. Your output should follow the format shown in the examples below.

Input

The first line contains integer n ($1 \le n \le 50$) — the number of lines in the program. The following n lines contain the program. Each program line consists of tokens '`{`', '`}`', '`a`' ... '`z`' separated by one or more spaces. The length of each line is between 1 and 50 characters. Each program line contains at least one non-space character.

The curly brackets in the program form a *regular bracket sequence*, so each opening bracket '`{`' has uniquely defined matching closing bracket '`}`' and vice versa. A variable is declared in a scope at most once. Any scope (including global) can be empty, i.e. can contain no variable declarations.

Output

For each fact of shadowing write a line in form "`r1:c1: warning: shadowed declaration of ?, the shadowed position is r2:c2`", where "`r1:c1`" is the number of line and position in line of shadowing declaration and "`r2:c2`" is the number of line and position in line of shadowed declaration. Replace '`?`' with the letter `'a' ... 'z'` — the name of shadowing/shadowed variable. If multiple outer scopes have variables named as the shadowing variable, the variable in the nearest outer scope is shadowed.

Print warnings in increasing order of `r1`, or in increasing order of `c1` if values `r1` are equal. Leave the output empty if there are no variable shadowings.

Examples

standard input
1 { a { b { a } } } b
standard output
1:11: warning: shadowed declaration of a, the shadowed position is 1:3

standard input
1 { a { a { a } } }
standard output
1:7: warning: shadowed declaration of a, the shadowed position is 1:3 1:11: warning: shadowed declaration of a, the shadowed position is 1:7

standard input

7
a b {
 a { a }
 c b {
  c
 } }
{ a d z
} z

standard output
2:2: warning: shadowed declaration of a, the shadowed position is 1:1 2:6: warning: shadowed declaration of a, the shadowed position is 2:2 3:6: warning: shadowed declaration of b, the shadowed position is 1:3 4:3: warning: shadowed declaration of c, the shadowed position is 3:2 6:3: warning: shadowed declaration of a, the shadowed position is 1:1

standard input
1 { a } { b } a { a } { a } { a x { a b x } } b x
standard output
1:17: warning: shadowed declaration of a, the shadowed position is 1:13 1:23: warning: shadowed declaration of a, the shadowed position is 1:13 1:29: warning: shadowed declaration of a, the shadowed position is 1:13 1:35: warning: shadowed declaration of a, the shadowed position is 1:29 1:39: warning: shadowed declaration of x, the shadowed position is 1:31