Problem 1. GUI

Input file:	`input.txt`
Output file:	`output.txt`
Time limit:	1 second
Memory limit:	256 megabytes

Your company is developing a new window-based GUI. The existing implementations of this interface type are fatally flawed: someone else made them. Windows are rectangles made of pixels. Sides of windows are parallel to screen sides. You are to write a program determing the relative positions of windows. According to your task, the program receives positions of two windows A and B as input and produces a verdict on their relative position. The following statements are possible:

- `A in B` — all pixels of the window A are pixels of the window B;
- `B in A` — all pixels of the window B are pixels of the window A;
- `Separate` — A and B do not share any pixels;
- `Intersect` — for all other cases;

It is guaranteed that the positions of the windows A and B are not the same.

The window position is defined by the coordinates of two pixels: the upper left corner and the lower right corner. The pixel with the coordinates (0, 0) is located in the lower left corner of the screen. The coordinate values ascend from left to right and downwards.

Input

The first line of the input file contains a single integer N — the number of sets of input data for your program ($1 \le N \le 1000$). It is followed by N pairs of lines.

Each set of input data is described with two lines: the first line gives the position of the window A, and the second line gives the position of the window B. For each window there are four integers: the coordinates X_l, Y_t of the upper left pixel and the coordinates X_r, Y_b of the lower right pixel ($0 \le X_l \le X_r \le 10^5$, $0 \le Y_t \le Y_b \le 10^5$).

Output

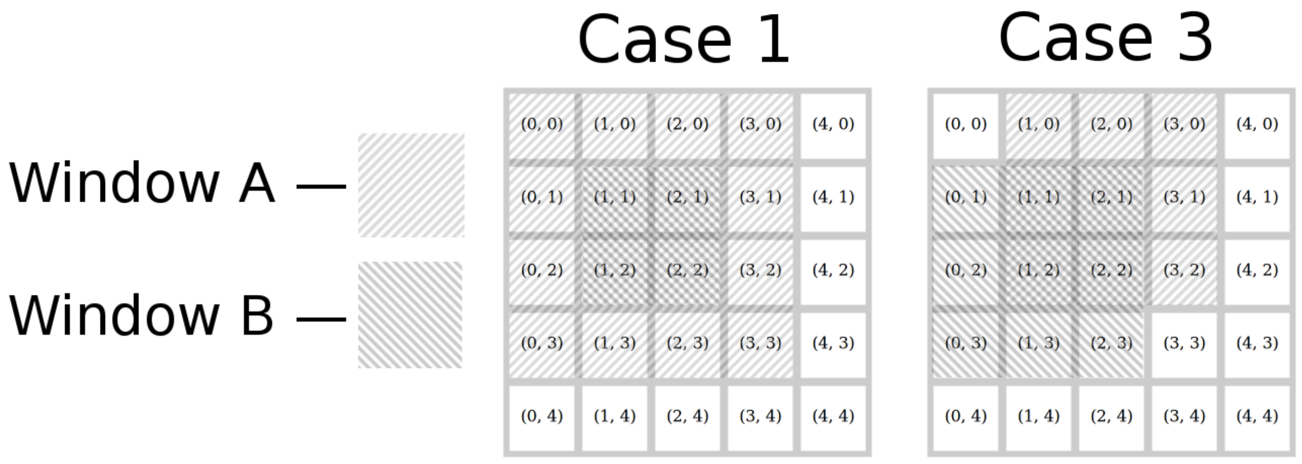
For each set of input data print a verdict on the relative position for the given windows.

Example

input.txt	output.txt

3
0 0 3 3
1 1 2 2
0 0 9 9
10 0 19 9
1 0 3 2
0 1 2 3
	B in A
	Separate
	Intersect

Illustration



Problem 2. Searching on the Cube

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

Two friends are playing a new board game they just invented. The lead thinks of a cube size and the initial position of the robot on the cube. The player then tries to find the treasure, using the "hotter / colder"principle, guiding the robot blindly.

The game field is a cube of size $N \times N \times N$; each of its sides is divided into $N \times N$ cells. According to the game rules, the lead can set the cube size N in the range between 3 and 300. The lead places a treasure in one of the $6N^2$ cells on the cube surface; the treasure stays in the same cell throughout the game. The lead also places the robot into one of the cells; the robot faces one of the four possible directions parallel to one of the cell sides.

The game is played step by step. In each step, the player gives a single command to the robot. There are four types of commands:

- `left` — the robot turns 90 degrees left, remaining in the same cell.
- `right` — the robot turns 90 degrees right, remaining in the same cell.
- `forward` — the robot moves one cell forward.
- `dig` — the robot digs in the cell where it is positioned.

The left/right direction is defined as seen from the outside of the cube.

When the `forward` command is executed, firstly, the side of the robot's cell it looks at is determined. Then the robot moves towards this side to the adjacent cell. If after such move robot remains on the same side of the cube, its direction doesn't change. If the robot moves to the adjacent face of the cube, its direction naturally rotates by 90 degrees around the cube edge it walks through. A simple rule holds true: if after the execution of the `forward` command the robot is turned 180 degrees, and the `forward` command is executed again, the robot returns to the previous cell.

The player does not see the playing field: he or she tells the commands to the lead blindly, and the lead executes these commands. Each time a `forward` command is executed, the lead tells the player one of the three words:

- `closer` — the robot is now closer to the treasure.
- `farther` — the robot is now further from the treasure.
- `same` — the distance between the robot and the treasure has not changed.

The distance from the robot to the treasure is the euclidean distance between their cell centers, i.e. the length of the segment between these centers. Note that the distance is calculated directly through the cube and not along its surface. If this value decreases, the lead says `closer`, and if it increases — `farther`.

Write a program which will play this game as the player. You cannot make mistakes: when the `dig` command is executed, the robot must be in the treasure cell.

Interaction protocol

This is an interactive problem. Instead of file input-output, you will work with a special program — an interactor. You will interact with this program via the standard input-output streams.

Your program sends robot control commands to the output stream. Each command is defined by the words `left`, `right`, `forward` or `dig` (see description of commands above). After each `forward` command printed by your program one of the following words is sent to the input stream: `closer`, `farther` or `same` (see description of results above).

Once you print `dig`, the game is over. If at this moment the robot and the treasure are in different cells, your solution will get Wrong Answer. Also, the number of commands must not be greater than $100N$, otherwise the solution will get Wrong Answer.

Make sure you print the newline character and clear the buffer of the output stream (the `flush` command) after each printed command. Otherwise the solution may be given a Timeout verdict.

Example

stdin	stdout
farther closer closer farther closer	forward left left forward forward forward right right forward dig

Problem 3. Mirrors

Input file:	`input.txt`
Output file:	`output.txt`
Time limit:	5 seconds
Memory limit:	512 megabytes

The dream has come true! No more fleeing from the widow's rage through the corridors. From now on, Madame Gritsatsueva is the main character of the "Reflect me!" game.

According to the game rules, she is moving in a rectangular game field of square cells of size $N \times M$, constantly reflecting either from the field borders or from rotating or stationary double-sided mirrors. Each mirror is a unitary segment with its center located in the center of a game field cell. Madame herself is much smaller than a mirror or a unitary cell and thus she can be considered a point, which, at any integer moment of time, is located in the center of a cell.

Every second Madame moves to an adjacent cell, unless she reflects from the field border. If she reaches a cell with a mirror, she changes her direction according to the reflection rule, i.e. "the angle of incidence equals the angle of reflection", that is: 90 degrees if the mirror is at 45 degrees to her direction, 180 degrees if the mirror blocks her way, i.e. is perpendicular to her direction. If the mirror does not block her movement, i.e. is parallel to her movement, it doesn't affect her further moves. If the widow finds herself in a boundary cell, her direction changes to the opposite direction. In this case, she stays in the boundary cell for two adjacent moments of time (think of it this way: she moves from the center of the cell to the margin for a half of a second, reflects instantaneously and moves back from the margin for another half of a second).

Madame may start at any nonnegative moment of time and in any direction from one of the acceptable start points. Her goal is to end up in the exit point with the coordinates (r_0, c_0) as soon as she can.

Input

The first line of the input file contains six integers: N, M, K, S, r_0, c_0, where N, M — vertical and horizontal field dimensions ($1 \le N, M \le 10^9$), K — the number of possible starting points ($1 \le K \le 10^3$), S — the number of mirrors ($0 \le S \le 10^3$), r_0, c_0 — the number of row and column where the exit point is located ($1 \le r_0 \le N$, $1 \le c_0 \le M$).

The next K lines contain the coordinates of the acceptable start points, with two integers in each line r_j and c_j ($1 \le r_j \le N$, $1 \le c_j \le M$).

The remaining S lines contain descriptions of the mirrors: five integers r_i, c_i, a_i, t_i, p_i in each. Here r_i, c_i are the numbers of row and column of the cell where the i-th mirror is located ($1 \le r_i \le N$, $1 \le c_i \le M$).

The number a_i describes the position of the mirror at the start moment ($0 \le a_i \le 3$). If $a_i = 0$, then the mirror is positioned vertically, and with each increment of the value a_i by 1 it turns 45 degrees clockwise. Thus if $a_i = 1$, then the mirror segment is directed from the lower left corner to the upper right corner; if $a_i = 2$, the segment is horizontal, etc.

The values t_i and p_i define the time of the first rotation of the mirror 45 degrees clockwise and the period of each consecutive turn, respectively ($1 \le p_i \le 7$, $1 \le t_i \le p_i$). Thus the mirror turns in the following moments of time: t_i — the first turn, $t_i + p_i$ — the second turn, $t_i + 2p_i$ — the third turn etc. Note that at the moments t_i, $(t_i + p_i)$ the mirror has already moved to a new position, and at the moments $(t_i - 1)$, $(t_i + p_i - 1)$ it is still in the previous position. If the mirror is stationary, the last two numbers t_i and p_i in the description equal -1. It is guaranteed that all positions of starting points, mirrors and exit points are pairwise different, which includes that no mirror is located in the start point. The field orientation and row and column numbering are such that row numbers increase while moving down the field, and column numbers increase while moving right.

Output

The output file must contain a single integer: the earliest time Madame can reach the exit point r_0, c_0, or -1, if that is impossible.

Example

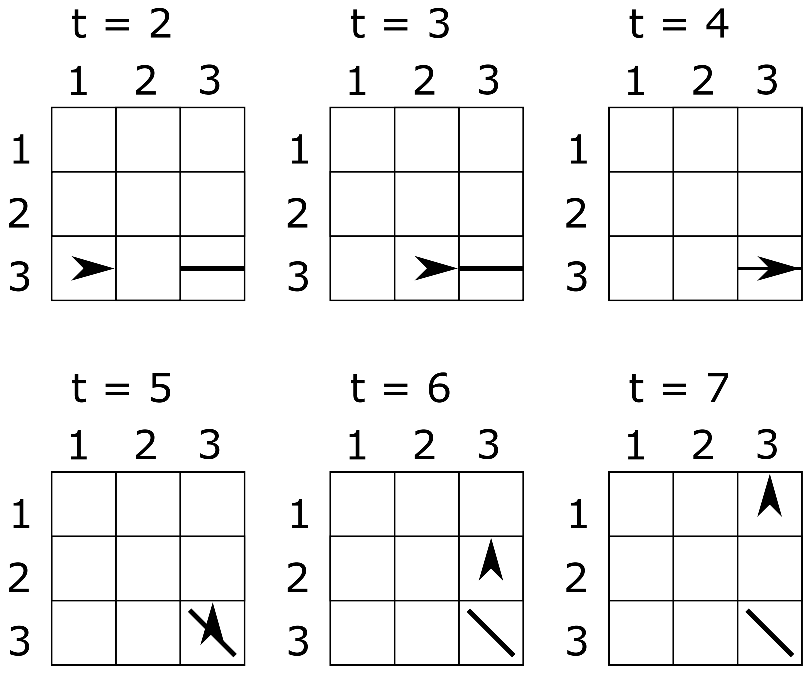
input.txt	output.txt

`3 3 1 1 1 3`
`3 1`
`3 3 2 5 7`
	`7`

`3 3 1 2 3 2`
`1 1`
`1 3 2 5 7`
`3 3 2 -1 -1`
	`-1`

Example explanation

The picture below represents the first sample. An arrow shows a position and a direction of Madame, the segment in a cell $(3, 3)$ stands for a mirror. Madame starts at the time $t = 2$ to the right. At the time $t = 4$ she passes by the mirror and at the time $t = 4.5$ reflects from the margin, after that she reflects from the rotated mirror at the time $t = 5$. At the time $t = 7$ Madame reaches the exit point.



Problem 4. Roads to cinematography

Input file:	`input.txt`
Output file:	`output.txt`
Time limit:	1 second
Memory limit:	256 megabytes

After the fiasco in Santa Carolina Mr. John First and his friends opened a new shrine of cinematograpy. Their new promotion location was chosen not far from a major railroad. Black Jack offered to build roads to the nearest stations where passenger trains stop.

Mr. John First drew a coordinate system. The origin point was their current location, and the axes X and Y were oriented to the east and north, respectively. Stations were marked as points with the coordinates $(x_i,\, y_i)$, where $i = 1, 2, \ldots, n$. The railway passes in the southeast direction, so the following inequations hold true:

$$\begin{cases} 0 \le x_1 \le x_2 \le x_3 \le \ldots \le x_{n-1} \le x_n \\ 0 \le y_n \le y_{n-1} \le y_{n-2} \le \ldots \le y_2 \le y_1 \end{cases}$$

Diana insists that each road should be oriented strictly at one of the cardinal directions, i.e. that the corresponding segment should be oriented to one of the coordinate axes. Jack says that roads must be built so that each station is connected by cinema directly or indirectly with those roads (without using the railroad) and that the distance between each station and the cinema is minimal. The distance must be minimal provided that the roads are parallel to the coordinate axes.

Roads cost money, and Mr. John First wants the total length of the roads to be minimal, provided that Diana's and Jack's conditions are met. The money they'd save could be spent on new films.

Input

The first line of the input file contains an integer n — the number of stations ($1 \le n \le 500$). Each of the following n lines contains two integers x_i, y_i — the coordinates of a station ($0 \le x_i, y_i \le 10^6$).

It is guaranteed that the sequence (x_i) is nonstrictly ascending and the sequence (y_i) is nonstrictly descending. No two stations are located in the same point. No station is located in the origin point.

Output

In the first line of the output file, print two integers: k — the number of roads to be built ($1 \le k \le 2000$) and A — the total length of these roads. Each of the following k lines must describe a road.

A road description must contain four integers x_1, y_1, x_2, y_2 — the coordinates of the end points of the road ($x_1 \ge x_2$, $y_1 \ge y_2$). One of the following two statements must hold true: either $x_1 = x_2$, or $y_1 = y_2$.

It is guaranteed that an optimal layout exists which meets the limitations of the output data format. If several solutions are possible, print any of them.

Example

input.txt	output.txt

3
0 9
2 4
5 1
	4 16
	0 9 0 4
	2 4 0 4
	5 1 0 1
	0 4 0 0

Problem 5. Geometric solver

Input file:	`input.txt`
Output file:	`output.txt`
Time limit:	1 second
Memory limit:	256 megabytes

Kedas is taking part in the development of a new geometric solver. His assigned task is to implement the removal of excessive constraints on a two-dimensional sketch. Help Kedas solve the task.

For this problem, assume there are only N line segments $L_1, L_2, \ldots, L_n$ in the sketch and a set of constraints imposed. Each *constraint* is a statement about some objects of the sketch. There are four types of constraints:

- `hor` i — the segment L_i is horizontal (the Y-coordinates of its endpoints are the equal).
- `vert` i — the segment L_i is vertical (the X-coordinates of its endpoints are the equal).
- `par` i j — the segments L_i and L_j are parallel.
- `perp` i j — the segments L_i and L_j are perpendicular.

A *solution* of the sketch is a placing of all $2N$ endpoints of all sketch segments on the coordinate plane, such that:

1. all sketch constraints are satisfied, i.e. the corresponding statements are true.
2. each segment has a non-zero length, i.e. its endpoints do **not** coincide.

The direction of a segment is considered irrelevant.

Excessive constraint can negatively affect the work of numerical algorithms of the geometric solver, so they need to be removed to the possible extent. A subset of sketch constraints is called *excessive* if its removal does not affect the set of solutions of the sketch. If the set of solutions of the sketch is empty, the sketch is called *inconsistent*, otherwise it is called *consistent*.

In this problem, find whether the sketch is consistent, and if it is, find the excessive set of constraints of the maximum size.

Input

The first line of the input file contains two integers: n — the number of line segments in the sketch ($1 \le n \le 10^5$), m — the number of constraints in the sketch ($0 \le m \le 10^5$). Each of the following m lines describes a single constraint.

Each constraint is described in the way shown above in the problem statement: first comes the word (`hor`, `vert`, `par` and `perp`) defining the type, followed by the space-separated number i of the first argument. If the type equals `par` or `perp`, then it is followed by another space-separated number j of the second argument. In this case $1 \le i \ne j \le n$.

Segments are numbered from 1 to n, constraints are numbered from 1 to m in the order they are described. The constraints may have completely identical descriptions.

Output

In the first line of the output file, print the word `consistent`, if the sketch is consistent, or the word `inconsistent`, if the sketch is inconsistent. If the sketch is inconsistent, do **not** print anything else.

If the sketch is consistent, the second line must contain a single integer k — the number of constraints in the discovered excessive set ($0 \le k \le m$). The third line must contain k different integers — the numbers of constraints in this set in any order.

If there are several possible answers, print any of them.

Examples

input.txt	output.txt
2 4 vert 1 hor 2 vert 1 perp 2 1	consistent 2 4 1
2 4 vert 1 hor 2 vert 1 par 2 1	inconsistent

Example explanation

In the first example, there are two vertical constraints imposed on the segment L_1; apparently, one of them can be discarded without affecting the solution set. The perpendicularity of the segments L_1 and L_2 is also excessive, because one of the segments is vertical and the other is horizontal. If three constraints are removed, it is always possible to provide a solution where one of the segments is **not** parallel to the coordinate axes, i.e. the set of solutions of the sketch increases.

In the second example, the sketch is inconsistent, because the constraint 4 states that the segments L_1 and L_2 are parallel, while the constraints 1 and 2 stipulate that they are perpendicular.

Problem 6. Monsters

Input file:	`input.txt`
Output file:	`output.txt`
Time limit:	5 seconds
Memory limit:	256 megabytes

Vasya has been a computer game fan since early childhood. Back in elementary school, he dreamed of getting a job in the gaming industry. Unfortunately, he never learned to code. Neither he learned to build 3D models. He didn't bother to learn to play music, either, because music school is so mind-numbingly boring! He hates working with clients — or having to deal with any people, for that matter. He can't write a plot and doesn't speak any foreign languages. Yet he's always been sure that it's not about skills — it's about loving what you do. That's why he's decided a long time ago to become a game tester — because he does love playing!

His dream has come true. He's been sitting in a tiny two by two by two cubicle, playing the same stupid browser game over and over again, for the three hundred seventy fifth time. In this game, Vasya controls a character running through levels and fighting N monsters. Vasya wins the game when he beats all those monsters. Levels in the game are designed in such manner that Vasya can fight only one moster at a time, and the order in which he fights these monsters is strictly pre-defined by the game developer. This must have been the developer's idea of «non-linear gameplay».

Vhen Vasya encounters a monster, he pulls out his sword and begins to hack the monster. The monster begins to chew Vasya. There are no extra skills, protection or other features of this sort in the game, because it would make the game way too complicated for the target audience. After a minute or so of rhythmic hacking and chewing, the duel comes to an end. The end is defined solely by two numerical parameters: the awesomeness score of Vasya's sword and the size of the monster's teeth. If the teeth are strictly larger than the sword, the monster chews Vasya, and, consequently, Vasya loses. Otherwise Vasya beats the monster, and the monster falls apart, yielding a new sword as some kind of a reward for Vasya's troubles. Vasya can pick this sword instead of his own or just leave it there. Given that for each sword its awesomeness score is known as a number, it's clear when a new sword must be picked (and otherwise), even though every now and then Vasya begins to doubt that the target audience is endowed with enough intelligence to figure it out. Afterwards, Vasya's wounds heal completely, and he's back on his way to meet the next monster.

The reason for so many sessions of such a worthless game is that Vasya's colleagues can't think of a smarter method to fix the gameplay balance. The real reason behind this might be that they simply enjoy seeing Vasya suffer — at least in Vasya's mind this theory is beginning to look more and more viable. Initially, Vasya gets a sword with an awesomeness score of X, defined in the config file of the game. The awesomeness score of the sword falling out of the i-th monster is defined by the integer S_i in the config file. The problem is that the teeth size of each monster is **not** strictly fixed. The config file contains only an integer M_i — the maximum possible value of teeth size of the i-th monster. The actual teeth size of a monster is generated randomly when the new character is created at the very beginning of the game. It (the teeth size of the i-th monster) can be any integer between 1 and M_i, inclusive.

It looks like the random number generation in this game exists for the sole purpose of marketing the game as one having a high «replayability» value. Every now and then someone changes the numbers in the config file in an attempt to fix the balane, which doesn't make testing easier at all — quite the opposite. Vasya can't code but he's got a vague feeling that his suffeing can be automated. He's asking for help: write a program that will evaluate the probability of winning the game based on the config file.

In this problem, you must determine the number of ways to generate monster teeth in such a way that Vasya can win the game. Since the number of such ways can be very large, find the remainder of dividing this number by $(10^9 + 7)$. In addition, K changes are performed in the config file in succession, and you must find the answer again after every one of these changes. Each change either raises the awesomeness score S_i of one of the swords or raises M_i — the upper limit for the teeth size of one of the monsters.

Input

The first line of the input file contains three integers: N — the number of monsters in the game $(1 \le N \le 10^5)$, K — the number of changes in the game config file $(0 \le K \le 10^5)$ and X — the awesomeness score of the sword Vasya gets initially $(1 \le X \le 10^9)$.

The second line contains N integers M_i, each defining the maximum teeth size of one of the monsters $(1 \le M_i \le 10^9)$. The third line contains N integers S_i — the awesomeness scores of swords falling out of the monsters $(1 \le S_i \le 10^9)$

The next K lines of the input file each contain a description of a single change in the config file. For each change, three integers are defined: t_j — type of change$(0 \le t_j \le 1)$, k_j — number of monster changed $(1 \le k_j \le N)$, and V_j — new parameter value$(2 \le V_j \le 10^9)$. If $t_j = 0$, then the maximum teeth size M_{k_j} of the k_j-th monster rises to V_j. If $t_j = 1$, then the awesomeness score S_{k_j} of the sword falling out of the k_j-th monster rises to V_j.

It is guaranteed that with each change the older parameter value is strictly smaller than the new value.

Output

Print $K+1$ integers. The first number is a solution to the problem with the initial config file setup (i.e. before any changes). The k-th of the remaining numbers must be equal to the soluition of the problem after applying the first k changes to the config file.

Example

input.txt	output.txt

5 3 2
4 2 3 5 7
3 2 4 3 2
1 3 5
0 2 3
0 2 10
	192
	300
	450
	450

6 0 1
1000 1000 1000 1000 1000 1000
1000 1000 1000 1000 1000 1000
	993000007

Example explanation

In the first example before any changes the game proceeds as follows: Vasya encounters the first monster while carrying a sword with an awesomeness score of $X = 2$. The monster teeth size must be 1 or 2 for Vasya to win. Upon beating the monster, Vasya finds a new sword with an awesomeness score of 3 and picks it instead of the old one. The second monster has a teeth size of 1 or 2, and Vasya beats it in any case. The monster yields a sword with an awesomeness score of 2, which Vasya **doesn't** pick. The teeth size of the third monster is no larger than 3, and Vasya beats it in any case, harvesting a sword with an awesomeness score of 4. The last two monster are beatable only if their teeth size is within the range of 1 to 4, inclusively, and they yield weaker, useless swords. The result is $2 \times 2 \times 3 \times 4 \times 4 = 192$ variants of generating a winnable game.

Problem 7. Regular expressions

Input file:	`input.txt`
Output file:	`output.txt`
Time limit:	10 seconds
Memory limit:	256 megabytes

Pavel doesn't have a college education. He's a natural programmer. He loves brief, laconic bits of code more than anything else in the world. One-liners are his favorite! Striving for shorter code he often makes use of regular expressions. There is nothing more creative and enjoyable than working on yet another regular expression in the middle of the night.

But bummer. Pavel's colleague, Simon, is often unhappy with Pavel's code. Simon never uses regular expressions in his code. He's just jealous! If he knew a thing about regular expressions, he wouldn't complain.

Once Pavel was careless to remark that Simon is not creative enough to build complex regular expressions. Simon replied, quite sacrilegiously, that there was nothing creative about regular expressions. Their argument evolved into a competition: they decided to find out who would be the fastest to write a regular expression matching a defined set of strings. Simon wants to win by creating a program which would solve this problem automatically, thus proving the lack of creative component in the process. Their friend Peter, a bioinformatician, will be the judge.

Simon glanced at the syntaxis of Perl regular expressions and was horrified: is this what programmers have done to a simple and elegant concept? Simon graduated from the mathematics department of the university and has always thought that a regular expression is the following:

1. `0` (zero character) — a regular expression that matches no string.
2. `a`, `g`, `t`, `c` (letter) — a regular expression matching precisely one string: a single-letter string with the same letter as that in the expression.
3. If `R` and `P` are regular expressions, then `(R|P)` is also a regular expression. It matches all strings which match at least one of the expressions `R` and `P`.
4. If `R` and `P` are regular expressions, then `(RP)` is also a regular expression. It matches all strings which can be split in two in such a way that the first part matches the expression `R`, and the second part matches the expression `P`.
5. If `R` is a regular expression, then `(R*)` is also a regular expression. It matches all strings which can be split into $k \geq 0$ parts so that each part matches the expression `R`.

For instance, the regular expression `(a*)` matches any string containing only letters `a`, including an empty string. The regular expression `(0*)` only matches an empty string (which can be split into zero parts matching the expression `0`). The regular expression `(a|(g(tc)))` matches two strings: `a` and `gtc`. Note that it is forbidden to omit or add extra brackets to regular expressions: it must contain strictly those pairs of brackets which appear during its construction according to the rules described above.

Simon wants a flawless victory by building the shortest matching regular expression. Help Simon. Write a program which finds the shortest regular expression for a set of strings, such that all these strings match the expression.

Input

The first line contains an integer T — the number of tests. It is followed by test descriptions.

The first line of a test description contains a single positive integer N — the number of strings in the set. Each of the following N lines begins with a nonnegative integer L — the length of the string from the set, followed by the string itself, separated by a space. The string only contains lowercase latin letters `a`, `g`, `t`, `c`. Note that a string in the set can be empty. In this case the line in the file will only contain the digit 0 and a space symbol.

The total number of strings in all sets is not greater than 2 000. The sum of lengths of all strings in all sets is not greater than 2 000.

Output

Print precisely T regular expressions, one per line. Each regular expression must be an answer to a corresponding test. If for any test there is no regular expression matched by all strings from the set, print the word `Impossible` instead of the answer.

Example

input.txt	output.txt
3 2 1 a 3 gtc 1 0 3 1 g 2 gg 3 ggg	(a|(g(tc))) (0*) (g*)

Commentary

The sixth line of the input data in the example zero is followed by a space symbol.

Problem 8. WSO-2017 soccer team

Input file:	`input.txt`
Output file:	`output.txt`
Time limit:	6 seconds
	12 seconds (for Java)
Memory limit:	256 megabytes

Have you heard about the most important sports event of the year in Siberia — the "Wide-Siberian Sports Olympiad (WSO)"? This year, Pasha is planning to bring a soccer team to this competition. Pasha has N friends whom he has numbered with integers from 1 to N; Pasha plans to pick his team from his friends. For the i-th friend, Pasha has easily defined two values: a_i — the score for their attacking skills and d_i — the score for their defensive skills.

To pick his team for the "WSO2017", Pasha has planned M training sessions. Each session is described with a pair of integers l_j and r_j ($1 \le l_j < r_j \le N$) — the left and right borders of the interval for the numbers of players whom he wants to bring to this particular session. In other words, all players with the number no less than l_j and no greater than r_j will be invited to the session j. In each session, all participating players are divided **evenly** into defenders, midfielders and forwards. Pasha has chosen the values l_j and r_j for all sessions in such a way that the number of participating players in each session is divisible by three.

Pasha knows that if the i-th friend plays in defense, the quality of his game equals to d_i, and if he plays as a forward, it equals to a_i; if he plays in the midfield, his game quality equals to $\frac{1}{2}(a_i + d_i)$. For every session, Pasha wants to know the maximum possible total value of game quality achievable by the optimal distribution of players into defenders, midfielders and forwards.

Your task is to write a program which uses the known values a_i and d_i for all players as well as known values l_j and r_j for all sessions to find out the maximum possible total quality value for each session.

Input

The first line of the input file contains a single integer N — the number of Pasha's friends ($3 \le N \le 128\,000$).

The second line contains N integers a_i — the values of their attacking skills ($0 \le a_i \le 10^9$).

The third line contains N integers d_i — the values of their defensive skills ($0 \le d_i \le 10^9$).

The fourth line of the output file contains a single integer M — the number of planned sessions ($1 \le M \le 100\,000$).

Then M lines follow, each describing a session with two integers l_j and r_j — the left and right border of the player numbers invited to this session ($1 \le l_j < r_j \le N$). It is guaranteed that the number of players invited to any session is divisible by three.

Output

For each session, the output file must contain a single real number — the maximum possible total quality of the game. The error of the printed values must not be greater than 10^{-2}.

Example

input.txt	output.txt
11	32.5
1 4 3 2 4 6 7 2 3 5 1	46.5
9 3 5 2 5 9 2 4 5 8 3	20.500
5	32
1 6	30.5
2 10	
5 7	
6 11	
3 8	

Problem 9. Primitive divisors

Input file:	`input.txt`
Output file:	`output.txt`
Time limit:	1 second
Memory limit:	256 megabytes

Victor, a top coder, loves the Internet. Every day he registers on many new websites, and every time he is asked to think of a new password. Victor wrote a password generator, which produces a new password when given a prime number. All he has to do now is to learn how to generate various prime numbers.

Victor has a favorite number q. To obtain the prime numbers, he decided to use prime divisors of $q^n - 1$, increasing the exponent n daily. However, to get rid of duplicate prime numbers, he chooses on the n-th day only those numbers which he has not encountered earlier, i.e. those that were not the prime divisors of $q^m - 1$ for each $m < n$.

Help our coder to write a program that finds acceptable prime divisors for the given q and n. Since Victor's password generator only accepts relatively small numbers, he needs prime numbers not greater than 10^7.

Input

The only line of the input file contain two integers q and n ($2 \le q \le 10^9$, $1 \le n \le 10^9$).

Output

The first line of the output file must contain the number of the desired divisors of the number $q^n - 1$. The second line must contain these divisors in ascending order, separated by space characters. All these numbers must be prime, not greater than 10^7 and must not occur for the lower values of n.

Example

input.txt	output.txt

`3 4`	`1`
	`5`

`2 6`	`0`

Example explanation

In the first example $q^n - 1 = 80 = 2^4 \cdot 5$. The numbers 2 and 5 are thus prime divisors. However, 2 divides $3^1 - 1$, and 5 does not divide the numbers $3^1 - 1$, $3^2 - 1$ and $3^3 - 1$, so Victor can take only one prime divisor — the number 5. In the second example, $2^6 - 1 = 63 = 3^2 \cdot 7$. No prime divisor fits, because 3 divides $2^2 - 1$, and 7 divides $2^3 - 1$.

Problem 10. Tickets

Input file:	`input.txt`
Output file:	`output.txt`
Time limit:	1 second
Memory limit:	256 megabytes

Kisa and Osya are sitting on their chairs and playing a curious game. Each of them is holding a tram ticket with a six-digit number from 000000 to 999999. They take turns to tell the digits on their tickets, from left to right, and after each pair the loser — the one whose digit is smaller — grants another one's wish.

Print the number of wishes each of them grants.

Input

The first line contains the number of Osya's ticket, and the second line contains the number of Kisa's ticket. Each number consists of six digits in the range from `0` to `9`. The number may have leading zeroes.

Output

In the first line output the number of wishes Osya will grant; in the second line output the number of wishes Kisa will grant.

Example

input.txt	output.txt
013936 132946	3 1

Example explanation

Osya loses with the first, second and fifth digit; Kisa loses with the third digit.

Problem 11. Logarithm smoothing

Input file:	`input.txt`
Output file:	`output.txt`
Time limit:	1 second
Memory limit:	256 megabytes

The life path of Vasisualiy Lokhankin is all curves. It looks like a logarithm (well, multiplied by a coefficient). In his free time, brooding about the tragic fate of the Russian intelligentsia and the great truth, he decided to take a closer look at his trajectory. In fact, he decided to substitute it with something more adapted for the masses - that is, a polyline. Naturally, he wants the polyline to be as similar to the initial logarithm as possible. On the other hand, the number of segments in the polyline must not be greater than n.

Let's take a more formal look at this problem. A function defined on the interval $[a, b]$ is a polyline with no more than n segments if its graph can be split into n parts, each of the parts being a line segment.

Let $B_n[a, b]$ denote the class of continuous polylines defined in the interval $[a, b]$ and having no more than n segments. The distance between two functions $f(x)$ and $g(x)$ on the segment $[a, b]$ is the value:

$$D(f, g) = \max_{x \in [a,b]} |f(x) - g(x)|$$

Find a polyline from the class $B_n[a, b]$ with the minimum distance to the function $f(x) = c \ln x$.

Input

The first line of the input file contains a single integer m — the number of test tasks($1 \le m \le 20$).

It is followed by m lines each containing four numbers n, c, a and b, where n is the number of segments in the polyline($1 \le n \le 20$), c is the factor multiplying the logarithm ($1 \le c \le 10^4$), a, b are the left and right endpoints of the segment in question, respectively ($10^{-2} \le a < b \le 10^2$). n and c are integers, and a, b are real numbers given with no more than two digits after the decimal point.

Output

The output file must contain m lines each containing a single real number — the minimum possible deviation of the polyline with no more than n segments from the logarithm, the answer to the corresponding test task. The absolute or relative error of your answer must not be greater than 10^{-8}.

Example

input.txt	output.txt

2
1 1 1 10
1 1 0.5 7
	0.309517460
	0.398765993

Problem 12. Outer space signals

Input file:	`input.txt`
Output file:	`output.txt`
Time limit:	1 second
Memory limit:	256 megabytes

The outer space research group is still busy searching for extraterrestrial intelligence. A while ago, two signals came from the outer space, and presumably they weren't just noise. The researchers named the signals X and Y. They encoded both signals as lines consisting of lowercase Latin letters.

Recently, the researchers got lucky again — they received yet another signal. They named the signal Z and also encoded it as a string of lowercase Latin letters. They are facing a task: to find out if Z is essentially different or it is just a combination of the already known signals X and Y.

The signal Z is considered to be a combination of the signals X and Y if and only if it contains nonintersecting occurrences of the signals X and Y. In other words, if Z contains two nonoverlapping substrings, one of each equals X and the other equals Y. If X contains Y or vice versa, substrings are considered overlapping.

Input

The first line of the input file contains the number of test cases (from 1 to 1000). The test cases are described in the following lines, three lines per each test case. The first line of test case contains representation of the signal Z. The following two lines of test case contain representations of the signals X and Y. Each of these strings is non-empty.

The total length of all lines in all test cases does not exceed 10^6.

Output

For each test case output "`YES`" (quotes for clarity) on a separate line if the signal Z is a combination of the signals X and Y, otherwise output "`NO`" (quotes for clarity).

Examples

input.txt	output.txt
3 abacada aba ada abacada aba aca abacada aba aaa	YES NO NO

Example explanation

In the second test case both strings X and Y are presented in the line Z as substrings. However, these substrings overlap: the third character of the string Z belongs to both of them.