# The 3rd Universal Cup


## Stage 6: Osijek

August 10-11, 2024

This problem set should contain 12 problems on 19 numbered pages.

**Based on**


Osijek Competitive Programming Camp

# Problem A. Coprime Array

| | |
|---|---|
| Input file: | `standard input` |
| Output file: | `standard output` |
| Time limit: | 1 second |
| Memory limit: | 256 megabytes |

Given two integers $s$ and $x$, find any shortest array such that the sum of its elements is $s$, and all elements are coprime to $x$.

Two integers are *coprime* if the only positive integer that divides both of them is 1.

## Input

The only line contains the integers $s$ and $x$ ($2 \le s, x \le 10^9$).

## Output

If there is no array that satisfies the condition, print a single integer $-1$.

Otherwise, the first line should contain one integer $n$ ($1 \le n \le 10^6$) — the length of the array. The next line should contain $n$ space-separated integers — the array itself. The elements of the array should not exceed $10^9$ in absolute value.

If there are multiple possible answers, print any. We have a proof that if a solution exists, then there exists a solution satisfying the constraints above.

## Examples

| standard input | standard output |
|---|---|
| 9 6 | 3<br>-7 -7 23 |
| 14 34 | 2<br>83 -69 |

# Problem B. Square Locator

| | |
|---|---|
| Input file: | `standard input` |
| Output file: | `standard output` |
| Time limit: | 1 second |
| Memory limit: | 256 megabytes |

There is a square $ABCD$ whose vertices have integer coordinates, with $A$ on the positive $y$-axis.

You are given the squared distances $AO^2$, $BO^2$, $CO^2$, $DO^2$, where $O(0,0)$ is the origin. Find the vertices of the square.

## Input

The only line of each test contains four integers $AO^2$, $BO^2$, $CO^2$, $DO^2$ ($1 \le AO^2, BO^2, CO^2, DO^2 \le 10^{18}$) — the **squares** of the distances of the vertices of the square to the origin.

## Output

Output one line containing seven space-separated integers $A_y$, $B_x$, $B_y$, $C_x$, $C_y$, $D_x$, $D_y$, representing the coordinates of the vertices of the square $A(0, A_y)$, $B(B_x, B_y)$, $C(C_x, C_y)$, $D(D_x, D_y)$.

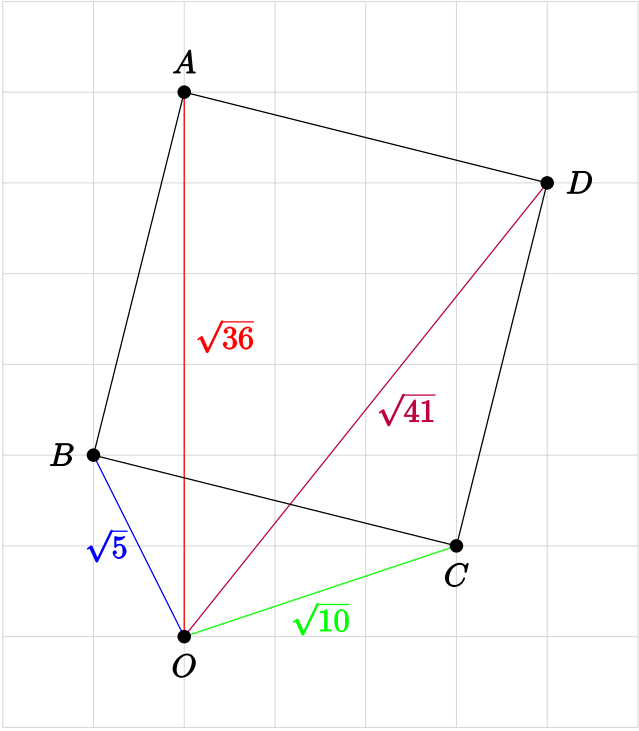
The input is given in such a way that such integers exist. If there are multiple possible answers, print any of them.

## Example

| standard input | standard output |
|---|---|
| 36 5 10 41 | 6 -1 2 3 1 4 5 |

## Note

The first test case is pictured below.



# Problem C. -is-this-bitset-

| | |
|---|---|
| Input file: | `standard input` |
| Output file: | `standard output` |
| Time limit: | 5 seconds |
| Memory limit: | 256 megabytes |

**Note the low memory limit!**

You are given a binary tree with $n$ nodes in it, rooted at node 1. This means that each node has at most 2 children. You are also given two arrays of $n$ integers, $a$ and $b$

The subset problem for node $i$ is defined to be the question: Can you take a subset $S$ of the ancestors of node $i$ and itself, such that $\sum_{j \in S}(a_j) = b_i$

You can do at most 5000 operations on array $a$. In one operation you choose two integers $i$ and $x$ ($1 \le i \le n$, $0 \le x \le 2 \cdot 10^6$), and set $a_i := x$.

After these operations, solve the subset sum problem for each node $i$, and output the results as a bitstring.

## Input

The first line contains the integer $n$ ($1 \le n \le 300\,000$) — the size of the binary tree The next $n-1$ lines contain two integers $u$ and $v$ ($1 \le u, v \le n$, $u \ne v$) — nodes $u$ and $v$ are connected by an edge.

It is guaranteed that these edges form a binary tree rooted at node 1

The next line contains $n$ integers $a_1, a_2, \ldots, a_n$ ($0 \le a_i \le 2 \cdot 10^6$) — the array $a$.

The last line of the input contains $n$ integers $b_1, b_2, \ldots, b_n$ ($0 \le b_i \le 2 \cdot 10^6$) — the array $b$

## Output

Output $n$ integers $a'_1, a'_2, \ldots, a'_n$ ($0 \le a'_i \le 2 \cdot 10^6$) — the new array $a'$, which is the array $a$ after the operations were done on it.

On the next line, output a bitstring of length $n$, with a 1 on position $i$, if the subset problem on node $i$ can be solved, and 0 otherwise.

## Examples

| standard input | standard output |
|---|---|
| 5<br>2 1<br>1 3<br>3 4<br>5 4<br>1 3 11 12 6<br>0 5 12 13 18 | 1 3 11 12 0<br>10110 |
| 1<br>2000000<br>2000000 | 2000000<br>1 |

## Note

In the sample output, it was decided to change the last number of array $a$ into 0.

# Problem D. Cycle Game

| | |
|---|---|
| Input file: | `standard input` |
| Output file: | `standard output` |
| Time limit: | 3 seconds |
| Memory limit: | 256 megabytes |

There's a board with $n \times m$ white squares. Jeroen plays a single-player game on this grid. Each turn he colors one square black. Two squares are adjacent if they share an edge. The squares, together with these adjacency relations form a planar graph. The game ends when a simple cycle of black squares is formed that has a non-empty interior. By drawing the edges of the graph with straight lines, between the centres of the squares we get a drawing of the planar graph. A square that does not lie on the cycle, is in the interior of the cycle, if it lies in the interior of the polygon that the edges of the cycle form, when drawn as straight line segments. Jeroen wants to produce a pretty picture on the playing board when he's done so he already thought of the moves he is going to play. If a move happens to end the game, he doesn't play it. In the input there are $k$ distinct positions $(r_i, c_i)$, the moves that Jeroen has planned out.

Your job is to figure out for each move, if it would end the game or not, and print a bitstring with 1 if the $i$-th move ends up being played, and 0 if this move would end the game, and does not get played.

## Input

The first line of the input contains three integers $n$,$m$,$k$ ($1 \le n \cdot m \le 300\,000$, $1 \le k \le n \cdot m$) The next $k$ lines each consist of two integers $r_i$, $c_i$ ($1 \le r_i \le n$, $1 \le c_i \le m$), which are the row and column of the $i$-th move that Jeroen is planning to play.

## Output

Output a single line with a string of length $k$ with a 1 on the $i$-th position if the $i$-th move will be played, and 0 if this move would end up losing, so it will not be played.

## Examples

| standard input | standard output |
|---|---|
| 4 3 7<br>2 1<br>2 2<br>2 3<br>3 1<br>3 2<br>4 1<br>4 2 | 1111111 |
| 3 3 8<br>1 1<br>1 2<br>1 3<br>2 3<br>3 3<br>3 2<br>3 1<br>2 1 | 11111110 |

# Problem E. Sum of Squares

| | |
|---|---|
| Input file: | `standard input` |
| Output file: | `standard output` |
| Time limit: | 2 seconds |
| Memory limit: | 256 megabytes |

You're given a polynomial $A(x) = a_0 + \cdots + a_n x^n$ with integer coefficients and an integer $m$.

Consider a multivariate polynomial $D(x_1, \ldots, x_m)$, defined as

$$D(x_1, \ldots, x_m) = \prod_{i=1}^{m} A(x_i) \prod_{j=1}^{i-1} (x_i - x_j).$$

Let $s$ be the sum of squares of all coefficients of $D(x_1, \ldots, x_m)$. Find $s$ modulo $10^9 + 7$.

## Input

The only line of input contains two integers $n$ ($0 \le n \le 500$) and $m$ ($0 \le m \le 10^9$).

The second line contains $n + 1$ integers $a_0, \ldots, a_n$ ($0 \le a_i < 10^9 + 7$).

It is guaranteed that $a_n \neq 0$ and $a_0 \neq 0$.

## Output

Print a single integer, the value of $s$ modulo $10^9 + 7$.

## Examples

| standard input | standard output |
|---|---|
| 2 0<br>1 2 3 | 1 |
| 2 1<br>1 2 3 | 14 |
| 2 2<br>1 2 3 | 264 |

## Note

For $A(x) = 1 + 2x + 3x^2$ and $m = 2$, we have

$$D(x_1, x_2) = -9x_1^3x_2^2 - 6x_1^3x_2 - 3x_1^3 + 9x_1^2x_2^3 - x_1^2x_2 - 2x_1^2 + 6x_1x_2^3 + x_1x_2^2 - x_1 + 3x_2^3 + 2x_2^2 + x_2.$$

Note that also $D = 1$ for $m = 0$ and $D(x_1) = 1 + 2x_1 + 3x_1^2$ for $m = 1$.

# Problem F. Alternating Cycle

| | |
|---|---|
| Input file: | `standard input` |
| Output file: | `standard output` |
| Time limit: | 2 seconds |
| Memory limit: | 256 megabytes |

You are given $n$ points in the plane, with no 3 points collinear. You can choose some non-empty subset of the points, and choose an order for this subset. If the points in your chosen ordered subset are $p_0, p_1, \ldots, p_{k-1}$, we want for $1 \leq i \leq k+1$, that the angles $\angle p_{i-1} p_i p_{i+1}$ are alternating clockwise and counterclockwise in increasing order of $i$. In the labeling of the points in the angle, you should take the indices $\mod k$. Note that $k$ must be even, otherwise due to parity it is impossible. Such an ordered subset of points is called an *alternating cycle*.

You have to find the *alternating cycle* with the smallest number of points in it, and print it to the output, or report no such cycle exists.

## Input

The first line of input contains a single integer, $n$ ($1 \leq n \leq 200\,000$) — the number of points in the input.

Each of the next $n$ lines contains the description of a point. Each line contains two integers $x$ and $y$, ($0 \leq x, y \leq 10^9$) — the coordinates of the point.

It is guaranteed that the points are distinct, and it is also guaranteed no 3 points are collinear.

## Output

Output $-1$ if there is no alternating cycle. Otherwise, output $k$, the number of points in the cycle. On the following lines output the points on the cycle in order.

On each of the following $k$ lines, print two integers $x$ and $y$, ($0 \leq x, y \leq 10^9$) — the coordinates of a point in the alternating cycle.

The points should be a subset of the input points, and they should form an alternating cycle in the order of the input.

If there are multiple solutions which achieve the minimum $k$, any of these solutions is accepted.

## Examples

| standard input | standard output |
|---|---|
| 6<br>10 15<br>20 15<br>15 23<br>0 31<br>15 0<br>30 30 | 6<br>0 31<br>10 15<br>15 0<br>20 15<br>30 30<br>15 23 |
| 4<br>0 0<br>0 1<br>1 0<br>1 1 | -1 |

## Note

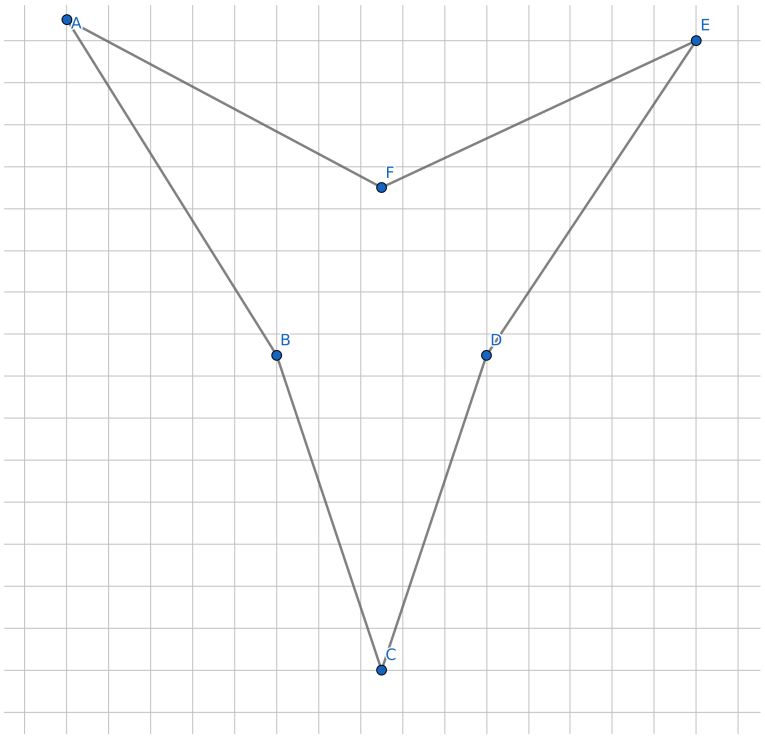



Illustration of sample 1, with a possible alternating cycle of length 6 drawn with straight line segments.

# Problem G. Touching Grass

| | |
|---|---|
| Input file: | `standard input` |
| Output file: | `standard output` |
| Time limit: | 3 seconds |
| Memory limit: | 32 megabytes |

**Note the unusual memory limit.**

Adamant is a famous person for writing educational blog posts on math and programming. One day, he finally decides that he has written too many blogs and it's time to go outside and touch some grass.

Adamant has a lawn on which $n$ grass plants grow. The grasses are indexed from 1 to $n$. We can draw the lawn and grasses on the 2D plane: the ground is the line $y = 0$, and grasses grow up vertically. Each grass can be described by two numbers $x$ and $y$, meaning it can be considered as a vertical line segment from $(x, 0)$ to $(x, y)$.

To touch grass, Adamant will move his hand $m$ times. Each time he moves his hand from the coordinates $(x_1, y_1)$ to the coordinates $(x_2, y_2)$ along a straight path. We say that his hand touches the $i$-th grass, if the segment $(x_1, y_1) - (x_2, y_2)$ touches the segment (or its endpoints) defined by the $i$-th grass.

Your task is to determine, for each time Adamant moves his hand, whether he will touch any grass. If the answer is yes, you also need to find the index of any grass he will touch.

## Input

The first line contains an integer $n$ ($1 \le n \le 8 \times 10^5$).

The $i$-th of the next $n$ lines contains two integers $x, y$ ($1 \le x, y \le 10^9$) describing the grass with index $i$. It is guaranteed that no two grasses have the same $x$.

The next line contains an integer $m$ ($1 \le m \le 3 \times 10^5$).

Each of the next $m$ lines contains four integers $x_1, y_1, x_2, y_2$ ($1 \le x_1, y_1, x_2, y_2 \le 10^9$) describing a hand movement. It is guaranteed that both $x_1$ and $x_2$ are different from the $x$-coordinate of any grass.

## Output

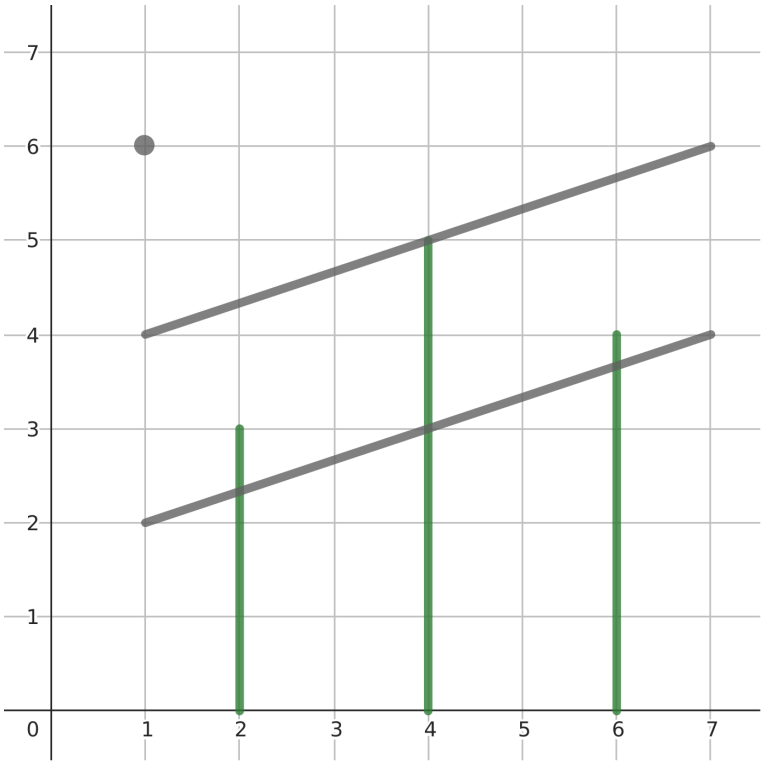
For each hand movement, print one line: the index of any grass that is touched. If there are multiple answers, print any one of them. If no grass is touched, print $-1$.

## Example

| standard input | standard output |
|---|---|
| 3<br>2 3<br>6 4<br>4 5<br>3<br>1 4 7 6<br>7 4 1 2<br>1 6 1 6 | 3<br>1<br>-1 |

## Note

Visualization of the sample:


7
6
5
4
3
2
1
0
1
2
3
4
5
6
7

# Problem H. Game Design

| | |
|---|---|
| Input file: | `standard input` |
| Output file: | `standard output` |
| Time limit: | 1 second |
| Memory limit: | 256 megabytes |

You are a game designer working on a new video game, *Outstandingly Captivating Platforming Challenge*. The game consists of $n$ levels indexed $1 \dots n$, which the player must complete in that order. In addition to the normal progression, the levels are also connected through $n$ one-way *warp portals*. A different team in your company has already completed the design of each level. They have placed one warp portal entrance and one warp portal exit in every level. Your task is to connect each entry portal to an exit portal on a **different** level such that each exit portal is also connected to only one entry portal.

However, there is an additional restriction: the player must not be able to skip ahead in the game. That is, the player must not be able to enter a portal, exit at a portal on a later level, and keep playing on that later level. In order to make this possible, the level designers have placed some exit portals in isolated locations from which the rest of the level is not accessible. That is: if an entry portal on level $u$ leads to an exit portal on some level $v > u$, then the exit portal on level $v$ must be in an isolated location.

You have already written a program to examine all allowed ways to connect each entry portal to an exit portal, in order to measure the predicted audience engagement. That program has been running for a while now, your boss is getting angry, and you want to know how long this program will take. Thus, calculate the number of allowed ways to connect each entry portal to an exit portal. Print the answer modulo 998 244 353.

## Input

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

Each test case consists of a binary string $s$ of length $n$ ($2 \le n \le 5000$). The $i$-th character of $s$ is `1` if the exit portal on level $i$ is at an isolated location, and `0` otherwise.

It is guaranteed that the sum of $n$ over all test cases doesn't exceed 5000.

## Output

For each test case, print the answer modulo 998 244 353 on a separate line.

## Example

| standard input | standard output |
|---|---|
| 4<br>0101<br>1010010010001010<br>11111<br>10100100011000010010101001001001 | 3<br>0<br>44<br>393298077 |

## Note

In the first example test, the valid configurations are $[2, 1, 4, 3]$, $[2, 4, 1, 3]$ and $[4, 1, 2, 3]$, where the $i$-th position in the array is the location of the exit portal connected to the entry portal on the $i$-th level.

In the second example test, there is no entry portal that can be connected to the exit portal on the last level.

# Problem I. Geometry Hacking

| | |
|---|---|
| Input file: | `standard input` |
| Output file: | `standard output` |
| Time limit: | 1 second |
| Memory limit: | 256 megabytes |

Jeroen made a new algorithm for checking if a point is inside a polygon:

Given a simple polygon $\mathbf{p_0}, \mathbf{p_1} \dots \mathbf{p_{n-1}}$ and a query point $\mathbf{q}$, his algorithm does the following:

- Initialize ans := 0
- Draw a ray from $\mathbf{q}$ in the positive $x$-direction.
- For all $0 \leq i < n$: Check if the closed segment $\mathbf{p_i p_{(i+1) \bmod n}}$ intersects the ray, if so, increment ans.
- For all $0 \leq i < n$ if $\mathbf{p_i}$ lies on the ray, add 1 to ans.
- If ans is odd, the algorithm says inside, otherwise outside.

He is convinced his algorithm is correct, but you have to convince him otherwise. To humiliate him, you are looking for polygons where the point $(0, 0)$ is strictly inside the polygon (so not on the boundary), but his algorithm will say that the point is outside. Consider all simple polygons with vertices on integer coordinates for which this is true, and sort this infinite list by area, breaking ties arbitrarily. Given an integer $k \leq 10^3$ in the input, output the $k$ first polygons in this list. If multiple possible answers exist, you can print any of them.

Note that two polygons $\mathbf{a}$ and $\mathbf{b}$ are considered the same polygon if the point list of $\mathbf{a}$ is a cyclic rotation of the points of $\mathbf{b}$. Two polygons are also considered the same, if the reversed point list of $\mathbf{a}$ is a cyclic rotation of $\mathbf{b}$. So in these cases, only one of $\mathbf{a}$ and $\mathbf{b}$ is included in the infinite list. Note that although placing an extra integer point on one of the edges of a polygon produces the same shape, this will be counted as a different polygon.

## Input

The only line of input contains the integer $k$ ($1 \leq k \leq 1000$) — the number of polygons you should output.

## Output

For the first $k$ polygons for which Jeroen's algorithm fails, in sorted order of area, output a description:

On the first line of the description, output $n$, the number of vertices of the polygon.

On each of the next $n$ lines, output 2 integers, $x$ and $y$ ( $|x|, |y| \leq 10^9$), the coordinates of the vertices of the polygon.

Under the constraints of the problem, it can be proven that if the smallest areas of the first $k$ valid polygons are $A_1 \leq A_2 \leq \cdots \leq A_k$, then ensuring the additional constraint that the coordinates do not exceed $10^9$ in absolute value, does not change this list of areas.

Note that polygons in the output have to be simple polygons.

## Example

| standard input | standard output |
|---|---|
| 2 | 4<br>-1 0<br>0 1<br>1 0<br>0 -1<br>3<br>0 -4<br>3 5<br>-3 5 |

## Note

**The output for the sample input is actually wrong. It is only to show the output format. These two polygons both have the point $(0, 0)$ inside, but Jeroen's algorithm does not fail, and their areas are not guaranteed to be the smallest possible $2$ areas.**

# Problem J. Non-Interactive Nim

| | |
|---|---|
| Input file: | `standard input` |
| Output file: | `standard output` |
| Time limit: | 1 second |
| Memory limit: | 256 megabytes |

*Nim* is a classical strategy game played by two players. There are $n$ piles of stones, the $i$-th of which contains $a_i$ stones. Players alternate in taking turns. On each turn, the player must pick a pile with a positive number of stones and remove a positive number of stones from it. The player who can't make a move loses.

The Blue Monster wanted to add an interactive Nim problem to the Ordinary & Common Problem Collection. In this problem, your program would have to play Nim against an opponent who always plays optimally. However, the Blue Monster is too lazy to learn how to create interactive problems. Therefore, you are asked to only make moves where the opponent has only one optimal move.

You are given $a_1, a_2, \ldots, a_n$ — an instance of Nim. It is guaranteed that if both players play optimally, then the first player loses. You will play as the first player, your opponent will play as the second player. Your opponent will always make optimal moves. You have to play in such a way that after each of your moves, there is only one move that your opponent can make such that you will lose if both players play optimally after that point.

Print a sequence of such moves or declare that this is impossible. Notice that since your opponent always makes optimal moves, you know exactly what moves your opponent will make. You do not have to minimize the length of the sequence.

## Input

The first line contains one integer $t$ ($1 \le t \le 5 \cdot 10^4$) — the number of test cases. $t$ test cases follow. Each test case is described as follows.

The first line of the test case contains one integer $n$ ($2 \le n \le 10^5$). The second line contains $n$ integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 10^{18}$). It is guaranteed that if both players play optimally, the first player will lose.

It is guaranteed that the sum of $n$ over all test cases doesn't exceed $10^5$.

## Output

For each test case, print the answer as follows.

If playing the game in a way that the opponent always has only one optimal move is impossible, print $-1$. Otherwise, print an integer $k$ ($1 \le k \le 100$) on the first line — the number of moves. On each of the next $k$ lines, print two integers $p$ ($1 \le p \le n$) and $x$, signifying that you will remove $x$ stones from the $p$-th pile.

It can be shown that under the constraints of this problem, if it is possible to play in a way that the opponent always has only one optimal move, then it is possible to do that and lose the game within 100 moves.

## Example

| standard input | standard output |
|---|---|
| 2<br>4<br>4 2 7 1<br>4<br>1 1 1 1 | 4<br>3 2<br>1 2<br>3 3<br>4 1<br>-1 |

## Note

In the first example test, the opponent is forced to make moves $(2, 2)$, $(3, 2)$, $(1, 1)$ and $(1, 1)$. The game will play out as follows:

| After your move | After opponent's move |
|---|---|
| $4, 2, 5, 1$ | |
| | $4, 0, 5, 1$ |
| $2, 0, 5, 1$ | |
| | $2, 0, 3, 1$ |
| $2, 0, 0, 1$ | |
| | $1, 0, 0, 1$ |
| $1, 0, 0, 0$ | |
| | $0, 0, 0, 0$ |

In the second example test, no matter what move you make, there will be three nonempty piles left, each with one stone. Clearly, all choices your opponent has are equivalent.

# Problem K. String and Nails

| | |
|---|---|
| Input file: | `standard input` |
| Output file: | `standard output` |
| Time limit: | 2 seconds |
| Memory limit: | 256 megabytes |

You are given $n$ nails mounted on a wooden board at integer coordinates. You have a piece of string that encloses all the nails and is pulled tight to form the smallest loop that encloses the points. You have to remove the nails one at a time, following this rule:

First, you pull the string tight around the remaining nails. Then, you may choose any nail that the string is pulled taut around (so the string touches this nail and makes an angle of $< 180$ degrees) and remove it. You try to repeat this process until only one point remains.

Check if you can repeatedly remove nails until only one nail is left, and if so, give such a sequence of removals.

## Input

The first line of input contains a single integer, $n$ ($1 \le n \le 200\,000$) — the number of points in the input.

Each of the next $n$ lines contains the description of a nail. Each line contains two integers $x$ and $y$, ($0 \le x, y \le 10^9$) — the coordinates of the nail on the wooden board.

It is guaranteed that no two nails occupy the exact same position on the wooden board.

## Output

Output "`YES`" if it is possible to remove all but one nail according to the rules; otherwise, print "`NO`". If the answer is "`YES`", on the following lines output the removals.

On each of the following $n - 1$ lines, print two integers $x$ and $y$, ($0 \le x, y \le 10^9$) — the coordinates of the nail that you currently want to remove.

If there are multiple solutions, any solution is accepted.

## Examples

| standard input | standard output |
|---|---|
| 3<br>1 1<br>2 4<br>3 1 | YES<br>1 1<br>2 4 |
| 1<br>1000000000 0 | YES |

# Problem L. All-You-Can-Eat

| | |
|---|---|
| Input file: | `standard input` |
| Output file: | `standard output` |
| Time limit: | 1 second |
| Memory limit: | 256 megabytes |

**This is an interactive problem.**

In an all-you-can-eat conveyor belt restaurant, meals are placed on a moving conveyor belt. The customers sit next to the conveyor belt as meals slide by. You can take any meal as it slides by you, and you can repeat that any number of times. It's a pretty good deal, but it's also easy to overeat and feel sick afterwards.

Specifically, after you sit down, meals $1, 2, \ldots, n$ will slide by. After seeing meal $i$, you judge its caloric value to be $a_i$.

In one particular such restaurant, there are two types of seats:

- *Facing the direction opposite of the direction of the conveyor belt.* In that case, you can see all items that will reach you in the future, and thus can plan your choices accordingly (assume there are no other customers in the restaurant).
- *Facing the direction of movement of the conveyor belt.* In that case, you can only see one item at a time: when meal $i$ next to you, you must decide whether you want it or not, with no knowledge of what the caloric values of the meals that come after it are. Once meal $i + 1$ slides by, meal $i$ is already too far to reach.

Therefore, it is harder to optimize your lunch when facing the direction of the conveyor belt. You did, however, come up with an unethical hack: you can get rid of a meal you have picked by discreetly sliding it on the table of another customer.

In order to prevent overeating, you have set the following rules for yourself.

- You will face the direction of movement of the conveyor belt.
- Once you start eating, you won't be able to take any new meals from the conveyor belt.
- At any time, the total caloric value of meals on your table must not exceed 1000.

Now you have the opposite problem — you worry that if you follow all the rules, you will leave the restaurant hungry. You decided that you'll be happy as long as the total caloric value of meals you eat is at least 60% of what you'd be able to get if you were to face the opposite direction but still followed the other rules.

More formally, let $x$ be the maximum possible sum of a subsequence of $a_1, a_2, \ldots, a_n$ that doesn't exceed 1000. You'll be happy if the total caloric value of meals you eat is at least $0.6x$.

Implement a strategy that ensures you'll always be happy. The interactor is **adaptive**, meaning that the sequence of caloric values is not necessarily decided in advance: it may depend on the choices your program makes.

## Input

The first line contains a single integer $t$ ($1 \le t \le 10^4$) — the number of test cases.

## Interaction Protocol

The interaction between your program and the jury's program begins with reading a single integer $n$ ($1 \le n \le 10^4$) — the number of meals that will slide by.

The following will then happen $n$ times:

- Read a single integer $a_i$ ($0 \le a_i \le 1000$) from the input — the caloric value of the $i$-th meal.
- Print a line of the form $k\ t_1\ t_2\ \cdots\ t_k$, ($0 \le k \le i$, $1 \le t_j \le i$) indicating that you want to discard the meals with indices $t_1, t_2, \ldots, t_k$. All of the items must be on your table at that time.
- Print a line containing either the word "`TAKE`" (if you want to add meal $i$ on your table) or "`IGNORE`" (if you want to leave it on the conveyor belt). In either case, the word must be printed without quotes.

After the $n$-th iteration, the sum of caloric values on your table must be at most $0.6x$, where $x$ is defined above.

It is guaranteed that the sum of $n$ over all test cases is at most $10^4$.

If your program makes an invalid query at any time (i.e. you print a line that doesn't conform to the input specification, you have more than 1000 total value on the table at a time, you try to discard an item that is not on the table, or a test case ends without you being happy), the interactor will immediately terminate. If your program keeps reading input after that, it may receive an arbitrary verdict, as it will keep reading from a closed stream. To prevent this, always check if the input stream is still open, i.e. instead of writing

```
int ai;
cin >> ai;
```

write

```
int ai;
if (!(cin >> ai))
    exit(0);
```

to instantly terminate your program if the interactor terminates. This way, you will receive Wrong Answer if you make an invalid query.

After printing a query do not forget to output end of line and flush the output. To do this, use:

- `fflush(stdout)` or `cout.flush()` in C++;
- `stdout.flush()` in Python.

## Example

| standard input | standard output |
|---|---|
| 1<br>5<br>10<br><br><br>13<br><br><br>450<br><br><br>585<br><br><br>465 | <br><br><br>0<br>TAKE<br><br>0<br>TAKE<br><br>0<br>TAKE<br><br>2 1 3<br>TAKE<br><br>0<br>IGNORE |

## Note

The example section shows one possible interaction between your program and the judge. After the third item, you have items $1, 2, 3$ on your table, with caloric values $10, 13$ and $450$ respectively. When the fourth item arrives, if you want to take it, you have to discard some items, as $10+13+450+585 = 1058 > 1000$. In this example, your program has decided to discard items 1 and 3, so you have exactly 598 afterwards. You also ignore the final item.

At the end of the process, you have total caloric value 598 on the table. If you had faced the opposite direction, you could have had as much as $10 + 13 + 450 + 465 = 938$. As $\frac{598}{938} \approx 0.637 > 0.6$, this is a valid solution.