A. Sign Flipping

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

You are given n integers $a_1, a_2, \ldots, a_n$, where n **is odd**. You are allowed to flip the sign of some (possibly all or none) of them. You wish to perform these flips in such a way that the following conditions hold:

1. At least $\frac{n-1}{2}$ of the adjacent differences $a_{i+1} - a_i$ for $i = 1, 2, \ldots, n-1$ are **greater than or equal to** 0.
2. At least $\frac{n-1}{2}$ of the adjacent differences $a_{i+1} - a_i$ for $i = 1, 2, \ldots, n-1$ are **less than or equal to** 0.

Find any valid way to flip the signs. It can be shown that under the given constraints, there always exists at least one choice of signs to flip that satisfies the required condition. If there are several solutions, you can find any of them.

Input

The input consists of multiple test cases. The first line contains an integer t ($1 \le t \le 500$) — the number of test cases. The description of the test cases follows.

The first line of each test case contains an integer n ($3 \le n \le 99$, n is odd) — the number of integers given to you.

The second line of each test case contains n integers $a_1, a_2, \ldots, a_n$ ($-10^9 \le a_i \le 10^9$) — the numbers themselves.

It is guaranteed that the sum of n over all test cases does not exceed 10000.

Output

For each test case, print n integers $b_1, b_2, \ldots, b_n$, corresponding to the integers after flipping signs. b_i has to be equal to either a_i or $-a_i$, and of the adjacent differences $b_{i+1} - b_i$ for $i = 1, \ldots, n-1$, at least $\frac{n-1}{2}$ should be non-negative and at least $\frac{n-1}{2}$ should be non-positive.

It can be shown that under the given constraints, there always exists at least one choice of signs to flip that satisfies the required condition. If there are several solutions, you can find any of them.

Example

input
5 3 -2 4 3 5 1 1 1 1 1 5 -2 4 7 -6 4 9 9 7 -4 -2 1 -3 9 -4 -5 9 -4 1 9 4 8 9 5 1 -9
output
-2 -4 3 1 1 1 1 1 -2 -4 7 -6 4 -9 -7 -4 2 1 -3 -9 -4 -5 4 -1 -9 -4 -8 -9 -5 -1 9

Note

In the first test case, the difference $(-4) - (-2) = -2$ is non-positive, while the difference $3 - (-4) = 7$ is non-negative.

In the second test case, we don't have to flip any signs. All 4 differences are equal to 0, which is both non-positive and non-negative.

In the third test case, $7 - (-4)$ and $4 - (-6)$ are non-negative, while $(-4) - (-2)$ and $(-6) - 7$ are non-positive.

B. Neighbor Grid

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

You are given a grid with n rows and m columns, where each cell has a non-negative integer written on it. We say the grid is **good** if for each cell the following condition holds: if it has a number $k > 0$ written on it, then exactly k of its neighboring cells have a number greater than 0 written on them. Note that if the number in the cell is 0, there is no such restriction on neighboring cells.

You are allowed to take any number in the grid and increase it by 1. You may apply this operation as many times as you want, to any numbers you want. Perform some operations (possibly zero) to make the grid good, or say that it is impossible. If there are multiple possible answers, you may find any of them.

Two cells are considered to be neighboring if they have a common edge.

Input

The input consists of multiple test cases. The first line contains an integer t ($1 \le t \le 5000$) — the number of test cases. The description of the test cases follows.

The first line of each test case contains two integers n and m ($2 \le n, m \le 300$) — the number of rows and columns, respectively.

The following n lines contain m integers each, the j-th element in the i-th line $a_{i,j}$ is the number written in the j-th cell of the i-th row ($0 \le a_{i,j} \le 10^9$).

It is guaranteed that the sum of $n \cdot m$ over all test cases does not exceed 10^5.

Output

If it is impossible to obtain a good grid, print a single line containing "NO".

Otherwise, print a single line containing "YES", followed by n lines each containing m integers, which describe the final state of the grid. This final grid should be obtainable from the initial one by applying some operations (possibly zero).

If there are multiple possible answers, you may print any of them.

Example

input
5 3 4 0 0 0 0 0 1 0 0 0 0 0 0

```
2 2
3 0
0 0
2 2
0 0
0 0
2 3
0 0 0
0 4 0
4 4
0 0 0 0
0 2 0 1
0 0 0 0
0 0 0 0
```

output

```
YES
0 0 0 0
0 1 1 0
0 0 0 0
NO
YES
0 0
0 0
NO
YES
0 1 0 0
1 4 2 1
0 2 0 0
1 3 1 0
```

Note

In the first test case, we can obtain the resulting grid by increasing the number in row 2, column 3 once. Both of the cells that contain 1 have exactly one neighbor that is greater than zero, so the grid is good. Many other solutions exist, such as the grid

$$0\ 1\ 0\ 0$$
$$0\ 2\ 1\ 0$$
$$0\ 0\ 0\ 0$$

All of them are accepted as valid answers.

In the second test case, it is impossible to make the grid good.

In the third test case, notice that no cell has a number greater than zero on it, so the grid is automatically good.

C. Element Extermination

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

You are given an array a of length n, which initially is a **permutation** of numbers from 1 to n. In one operation, you can choose an index i ($1 \leq i < n$) such that $a_i < a_{i+1}$, and remove either a_i or a_{i+1} from the array (after the removal, the remaining parts are concatenated).

For example, if you have the array $[1, 3, 2]$, you can choose $i = 1$ (since $a_1 = 1 < a_2 = 3$), then either remove a_1 which gives the new array $[3, 2]$, or remove a_2 which gives the new array $[1, 2]$.

Is it possible to make the length of this array equal to 1 with these operations?

Input

The first line contains a single integer t ($1 \leq t \leq 2 \cdot 10^4$) — the number of test cases. The description of the test cases follows.

The first line of each test case contains a single integer n ($2 \leq n \leq 3 \cdot 10^5$) — the length of the array.

The second line of each test case contains n integers a_1, a_2, ..., a_n ($1 \leq a_i \leq n$, a_i **are pairwise distinct**) — elements of the array.

It is guaranteed that the sum of n over all test cases doesn't exceed $3 \cdot 10^5$.

Output

For each test case, output on a single line the word "YES" if it is possible to reduce the array to a single element using the aforementioned operation, or "NO" if it is impossible to do so.

Example

input

```
4
3
1 2 3
4
3 1 2 4
3
2 3 1
6
2 4 6 1 3 5
```

output

```
YES
YES
NO
YES
```

Note

For the first two test cases and the fourth test case, we can operate as follow (the bolded elements are the pair chosen for that operation):

$[1, \ , \ ] \rightarrow [\ , \ ] \rightarrow [1]$

$[3, \ , \ , 4] \rightarrow [3, \ , \ ] \rightarrow [\ , \ ] \rightarrow [4]$

$[\ , \ , 6, 1, 3, 5] \rightarrow [\ , \ , 1, 3, 5] \rightarrow [4, 1, \ , \ ] \rightarrow [4, \ , \ ] \rightarrow [\ , \ ] \rightarrow [4]$

D. Replace by MEX

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

You're given an array of n integers between 0 and n inclusive.

In one operation, you can choose any element of the array and replace it by the MEX of the elements of the array (which may change after the operation).

For example, if the current array is $[0, 2, 2, 1, 4]$, you can choose the second element and replace it by the MEX of the present elements — 3. Array will become $[0, 3, 2, 1, 4]$.

You must make the array non-decreasing, using at most $2n$ operations.

It can be proven that it is always possible. Please note that you do **not** have to minimize the number of operations. If there are many solutions, you can print any of them.

–

An array $b[1 \ldots n]$ is non-decreasing if and only if $b_1 \le b_2 \le \ldots \le b_n$.

The MEX (minimum excluded) of an array is the smallest non-negative integer that does not belong to the array. For instance:

- The MEX of $[2, 2, 1]$ is 0, because 0 does not belong to the array.
- The MEX of $[3, 1, 0, 1]$ is 2, because 0 and 1 belong to the array, but 2 does not.
- The MEX of $[0, 3, 1, 2]$ is 4 because 0, 1, 2 and 3 belong to the array, but 4 does not.

It's worth mentioning that the MEX of an array of length n is always between 0 and n inclusive.

Input
The first line contains a single integer t ($1 \le t \le 200$) — the number of test cases. The description of the test cases follows.

The first line of each test case contains a single integer n ($3 \le n \le 1000$) — length of the array.

The second line of each test case contains n integers $a_1, \ldots, a_n$ ($0 \le a_i \le n$) — elements of the array. Note that they **don't have to be distinct**.

It is guaranteed that the sum of n over all test cases doesn't exceed 1000.

Output
For each test case, you must output two lines:

The first line must contain a single integer k ($0 \le k \le 2n$) — the number of operations you perform.

The second line must contain k integers $x_1, \ldots, x_k$ ($1 \le x_i \le n$), where x_i is the index chosen for the i-th operation.

If there are many solutions, you can find any of them. Please remember that it is **not** required to minimize k.

Example

input
5 3 2 2 3 3 2 1 0 7 0 7 3 1 3 7 7 9 2 0 1 1 2 4 4 2 0 9 8 4 7 6 1 2 3 0 5

output
0 2 3 1 4 2 5 5 4 11 3 8 9 7 8 5 9 6 4 1 2 10 1 8 1 9 5 2 4 6 3 7

Note
In the first test case, the array is already non-decreasing ($2 \le 2 \le 3$).

Explanation of the second test case (the element modified by each operation is colored in red):

- $a = [2, 1, 0]$; the initial MEX is 3.
- $a = [2, 1, \color{red}{3}]$; the new MEX is 0.
- $a = [\color{red}{0}, 1, 3]$; the new MEX is 2.
- The final array is non-decreasing: $0 \le 1 \le 3$.

Explanation of the third test case:

- $a = [0, 7, 3, 1, 3, 7, 7]$; the initial MEX is 2.
- $a = [0, \color{red}{2}, 3, 1, 3, 7, 7]$; the new MEX is 4.
- $a = [0, 2, 3, 1, \color{red}{4}, 7, 7]$; the new MEX is 5.
- $a = [0, 2, 3, 1, \color{red}{5}, 7, 7]$; the new MEX is 4.
- $a = [0, 2, 3, \color{red}{4}, 5, 7, 7]$; the new MEX is 1.
- The final array is non-decreasing: $0 \le 2 \le 3 \le 4 \le 5 \le 7 \le 7$.

E. Inversion SwapSort

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

Madeline has an array a of n integers. A pair (u, v) of integers forms an inversion in a if:

- $1 \le u < v \le n$.
- $a_u > a_v$.

Madeline recently found a magical paper, which allows her to write two indices u and v and swap the values a_u and a_v. Being bored, she decided to write a list of pairs (u_i, v_i) with the following conditions:

- all the pairs in the list are distinct and form an inversion in a.
- all the pairs that form an inversion in a are in the list.
- Starting from the given array, if you swap the values at indices u_1 and v_1, then the values at indices u_2 and v_2 and so on, then after all pairs are processed, the array a will be sorted in **non-decreasing order**.

Construct such a list or determine that no such list exists. If there are multiple possible answers, you may find any of them.

Input
The first line of the input contains a single integer n ($1 \le n \le 1000$) — the length of the array.

Next line contains n integers $a_1, a_2, \ldots, a_n$ $(1 \le a_i \le 10^9)$ — elements of the array.

Output

Print -1 if no such list exists. Otherwise in the first line you should print a single integer m ($0 \le m \le \dfrac{n(n-1)}{2}$) — number of pairs in the list.

The i-th of the following m lines should contain two integers u_i, v_i ($1 \le u_i < v_i \le n$).

If there are multiple possible answers, you may find any of them.

Examples

input
3 3 1 2
output
2 1 3 1 2

input
4 1 8 1 6
output
2 2 4 2 3

input
5 1 1 1 2 2
output
0

Note

In the first sample test case the array will change in this order $[3, 1, 2] \rightarrow [2, 1, 3] \rightarrow [1, 2, 3]$.

In the second sample test case it will be $[1, 8, 1, 6] \rightarrow [1, 6, 1, 8] \rightarrow [1, 1, 6, 8]$.

In the third sample test case the array is already sorted.

F. Integer Game

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

This is an interactive problem.

Anton and Harris are playing a game to decide which of them is the king of problemsetting.

There are three piles of stones, initially containing a, b, and c stones, where a, b, and c are **distinct** positive integers. On each turn of the game, the following sequence of events takes place:

- The first player chooses a positive integer y and provides it to the second player.
- The second player adds y stones to one of the piles, with the condition that **he cannot choose the same pile in two consecutive turns**.

The second player loses if, at any point, two of the piles contain the same number of stones. The first player loses if 1000 turns have passed without the second player losing.

Feeling confident in his skills, Anton decided to let Harris choose whether he wants to go first or second. Help Harris defeat Anton and become the king of problemsetting!

Input

The first line of input contains three distinct positive integers a, b, and c ($1 \leq a, b, c \leq 10^9$) — the initial number of stones in piles 1, 2, and 3 respectively.

Interaction

The interaction begins by reading the integers a, b and c.

After reading the integers, print a single line containing either "`First`" or "`Second`", denoting who you want to play as (as first or second correspondently).

On each turn, the first player (either you or the judge) must print a positive integer y ($1 \leq y \leq 10^{12}$).

Then, the second player must print 1, 2, or 3, indicating which pile should have y stones added to it. From the second turn onwards, the pile that the second player chooses must be different from the pile that they chose on the previous turn.

If you are playing as `Second` and complete 1000 turns without losing, or if you are playing as `First` and the judge has determined that it cannot make a move without losing, the interactor will print 0 and will finish interaction. This means that your program is correct for this test case, and you should exit immediately.

If you are playing as `First` and complete 1000 turns without winning, or if you are playing as `Second` and print a move that makes two piles have the same number of stones, or if you output an invalid move as either player, the interactor will print -1 and will finish interaction. You will receive a **Wrong Answer** verdict. Make sure to exit immediately to avoid getting other verdicts.

After printing something do not forget to output end of line and flush the output. Otherwise, you will get **Idleness limit exceeded**. To do this, use:

- `fflush(stdout)` or `cout.flush()` in C++;
- `System.out.flush()` in Java;
- `flush(output)` in Pascal;
- `stdout.flush()` in Python;
- see documentation for other languages.

In this problem, hacks are disabled.

Example

input
5 2 6 3 0
output
First 2 3

Note

In the sample input, the piles initially have 5, 2, and 6 stones. Harris decides to go first and provides the number 2 to Anton. Anton adds 2 stones to the third pile, which results in 5, 2, and 8.

In the next turn, Harris chooses 3. Note that Anton cannot add the stones to the third pile since he chose the third pile in the previous turn. Anton realizes that he has no valid moves left and reluctantly recognizes Harris as the king.

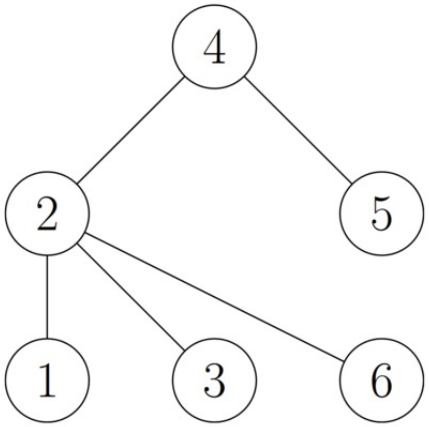
G. Tree Modification

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

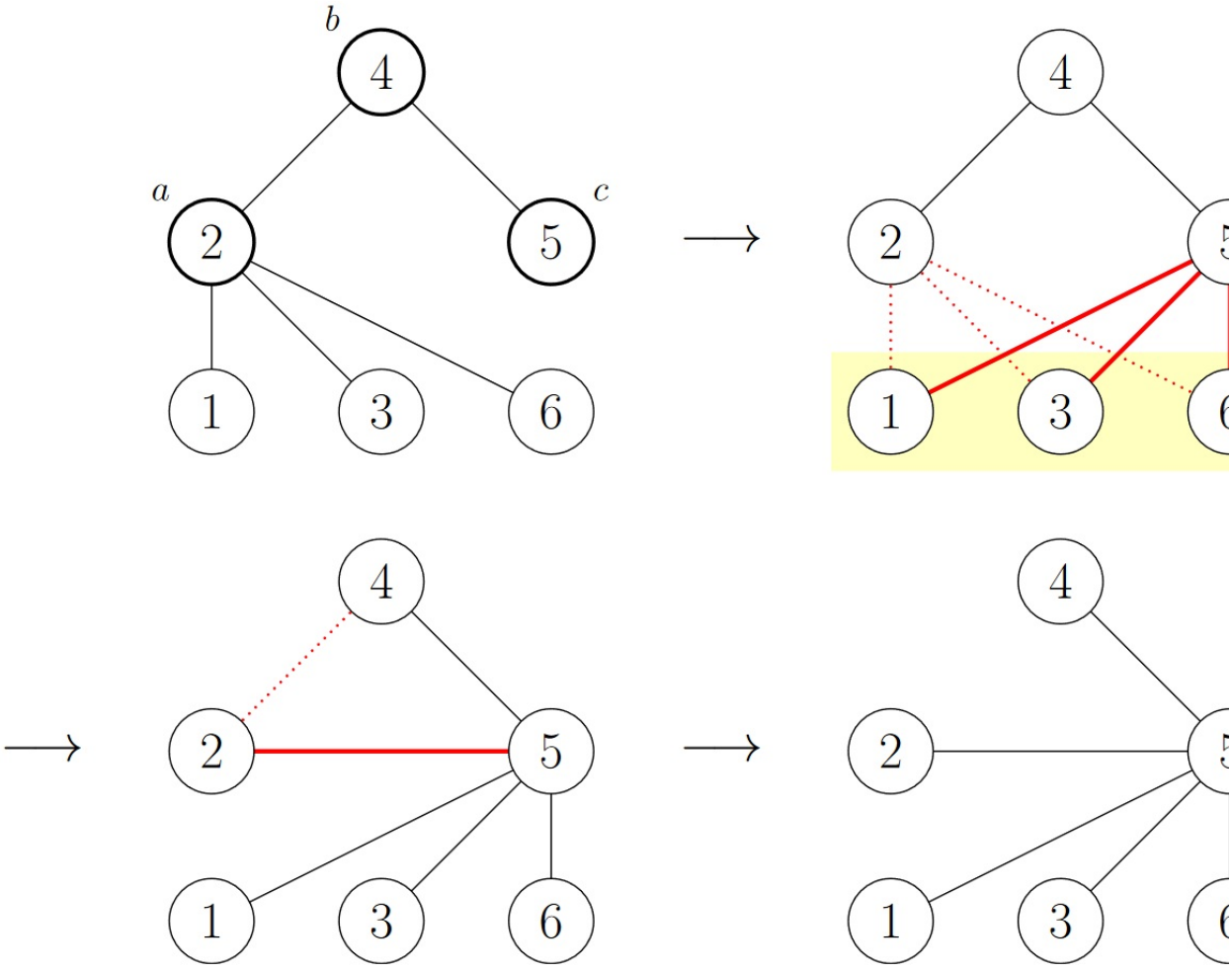
You are given a tree with n vertices. You are allowed to modify the structure of the tree through the following multi-step operation:

1. Choose three vertices a, b, and c such that b is adjacent to both a and c.
2. For every vertex d **other than** b that is adjacent to a, remove the edge connecting d and a and add the edge connecting d and c.
3. Delete the edge connecting a and b and add the edge connecting a and c.

As an example, consider the following tree:



The following diagram illustrates the sequence of steps that happen when we apply an operation to vertices 2, 4, and 5:



It can be proven that after each operation, the resulting graph is still a tree.

Find the minimum number of operations that must be performed to transform the tree into a star. A star is a tree with one vertex of degree $n - 1$, called its center, and $n - 1$ vertices of degree 1.

Input

The first line contains an integer n ($3 \leq n \leq 2 \cdot 10^5$) — the number of vertices in the tree.

The i-th of the following $n - 1$ lines contains two integers u_i and v_i ($1 \leq u_i, v_i \leq n$, $u_i \neq v_i$) denoting that there exists an edge connecting vertices u_i and v_i. It is guaranteed that the given edges form a tree.

Output

Print a single integer — the minimum number of operations needed to transform the tree into a star.

It can be proven that under the given constraints, it is always possible to transform the tree into a star using at most 10^{18} operations.

Examples

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

input
4 2 4 4 1 3 4
output
0

Note

The first test case corresponds to the tree shown in the statement. As we have seen before, we can transform the tree into a star with center at vertex 5 by applying a single operation to vertices 2, 4, and 5.

In the second test case, the given tree is already a star with the center at vertex 4, so no operations have to be performed.

H. Set Merging

time limit per test: 4 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

You are given a permutation $a_1, a_2, \ldots, a_n$ of numbers from 1 to n. Also, you have n sets $S_1, S_2, \ldots, S_n$, where $S_i = \{a_i\}$. Lastly, you have a variable cnt, representing the current number of sets. Initially, $cnt = n$.

We define two kinds of functions on sets:

$f(S) = \min\limits_{u \in S} u$;

$g(S) = \max\limits_{u \in S} u$.

You can obtain a new set by merging two sets A and B, if they satisfy $g(A) < f(B)$ (Notice that the old sets do not disappear).

Formally, you can perform the following sequence of operations:

- $cnt \leftarrow cnt + 1$;
- $S_{cnt} = S_u \cup S_v$, you are free to choose u and v for which $1 \leq u, v < cnt$ and which satisfy $g(S_u) < f(S_v)$.

You are required to obtain some specific sets.

There are q requirements, each of which contains two integers l_i,r_i, which means that there must exist a set S_{k_i} (k_i is the ID of the set, you should determine it) which equals $\{a_u \mid l_i \leq u \leq r_i\}$, which is, the set consisting of all a_i with indices between l_i and r_i.

In the end you must ensure that $cnt \leq 2.2 \times 10^6$. Note that you **don't** have to minimize cnt. It is guaranteed that a solution under given constraints exists.

Input

The first line contains two integers n, q $(1 \leq n \leq 2^{12}, 1 \leq q \leq 2^{16})$ — the length of the permutation and the number of needed sets correspondently.

The next line consists of n integers $a_1, a_2, \cdots, a_n$ ($1 \leq a_i \leq n$, a_i are pairwise distinct) — given permutation.

i-th of the next q lines contains two integers l_i, r_i $(1 \leq l_i \leq r_i \leq n)$, describing a requirement of the i-th set.

Output

It is guaranteed that a solution under given constraints exists.

The first line should contain one integer cnt_E $(n \leq cnt_E \leq 2.2 \times 10^6)$, representing the number of sets after all operations.

$cnt_E - n$ lines must follow, each line should contain two integers u, v $(1 \leq u, v \leq cnt')$, where cnt' is the value of cnt before this operation), meaning that you choose S_u, S_v and perform a merging operation. In an operation, $g(S_u) < f(S_v)$ must be satisfied.

The last line should contain q integers $k_1, k_2, \cdots, k_q$ $(1 \leq k_i \leq cnt_E)$, representing that set S_{k_i} is the ith required set.

Please notice the large amount of output.

Examples

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

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

Note

In the first sample:

We have $S_1 = \{1\}, S_2 = \{3\}, S_3 = \{2\}$ initially.

In the first operation, because $g(S_3) = 2 < f(S_2) = 3$, we can merge S_3, S_2 into $S_4 = \{2, 3\}$.

In the second operation, because $g(S_1) = 1 < f(S_3) = 2$, we can merge S_1, S_3 into $S_5 = \{1, 2\}$.

In the third operation, because $g(S_5) = 2 < f(S_2) = 3$, we can merge S_5, S_2 into $S_6 = \{1, 2, 3\}$.

For the first requirement, $S_4 = \{2, 3\} = \{a_2, a_3\}$, satisfies it, thus $k_1 = 4$.

For the second requirement, $S_6 = \{1, 2, 3\} = \{a_1, a_2, a_3\}$, satisfies it, thus $k_2 = 6$

Notice that unused sets, identical sets, outputting the same set multiple times, and using sets that are present initially are all allowed.

I. Cubic Lattice

time limit per test: 4 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

A cubic lattice L in 3-dimensional euclidean space is a set of points defined in the following way:

$$L = \{u \cdot r_1 + v \cdot r_2 + w \cdot r_3\}_{u,v,w \in}$$

Where $r_1, r_2, r_3 \in$ ^{3} are some integer vectors such that:

- r_1, r_2 and r_3 are pairwise orthogonal:

$$r_1 \cdot r_2 = r_1 \cdot r_3 = r_2 \cdot r_3 = 0$$

  Where $a \cdot b$ is a dot product of vectors a and b.
- r_1, r_2 and r_3 all have the same length:

$$|r_1| = |r_2| = |r_3| = r$$

You're given a set $A = \{a_1, a_2, \ldots, a_n\}$ of integer points, i-th point has coordinates $a_i = (x_i; y_i; z_i)$. Let $g_i = \gcd(x_i, y_i, z_i)$. It is guaranteed that $\gcd(g_1, g_2, \ldots, g_n) = 1$.
You have to find a cubic lattice L such that $A \subset L$ and r is the maximum possible.

Input

First line contains single integer n ($1 \le n \le 10^4$) — the number of points in A.

The i-th of the following n lines contains integers x_i, y_i, z_i ($0 < x_i^2 + y_i^2 + z_i^2 \le 10^{16}$) — coordinates of the i-th point.

It is guaranteed that $\gcd(g_1, g_2, \ldots, g_n) = 1$ where $g_i = \gcd(x_i, y_i, z_i)$.

Output

In first line output a single integer r^2, the square of maximum possible r.

In following 3 lines output coordinates of vectors r_1, r_2 and r_3 respectively.

If there are multiple possible answers, output any.

Examples

input
2 1 2 3 1 2 1
output
1 1 0 0 0 1 0 0 0 1

input
1 1 2 2
output
9 2 -2 1 1 2 2 -2 -1 2

input
1 2 5 5
output
9 -1 2 2 2 -1 2 2 2 -1