Codeforces Global Round 5

A. Balanced Rating Changes

time limit per test: 1 second
memory limit per test: 512 megabytes
input: standard input
output: standard output

Another Codeforces Round has just finished! It has gathered n participants, and according to the results, the expected rating change of participant i is a_i. These rating changes are *perfectly balanced* — their sum is equal to 0.

Unfortunately, due to minor technical glitches, the round is declared *semi-rated*. It means that all rating changes must be divided by two.

There are two conditions though:

- For each participant i, their modified rating change b_i must be integer, and as close to $\frac{a_i}{2}$ as possible. It means that either $b_i = \lfloor \frac{a_i}{2} \rfloor$ or $b_i = \lceil \frac{a_i}{2} \rceil$. In particular, if a_i is even, $b_i = \frac{a_i}{2}$. Here $\lfloor x \rfloor$ denotes rounding down to the largest integer not greater than x, and $\lceil x \rceil$ denotes rounding up to the smallest integer not smaller than x.
- The modified rating changes must be perfectly balanced — their sum must be equal to 0.

Can you help with that?

Input

The first line contains a single integer n ($2 \le n \le 13\,845$), denoting the number of participants.

Each of the next n lines contains a single integer a_i ($-336 \le a_i \le 1164$), denoting the rating change of the i-th participant.

The sum of all a_i is equal to 0.

Output

Output n integers b_i, each denoting the modified rating change of the i-th participant in order of input.

For any i, it must be true that either $b_i = \lfloor \frac{a_i}{2} \rfloor$ or $b_i = \lceil \frac{a_i}{2} \rceil$. The sum of all b_i must be equal to 0.

If there are multiple solutions, print any. We can show that a solution exists for any valid input.

Examples

input

```
3
10
-5
-5
```

output

```
5
-2
-3
```

input

```
7
-7
-29
0
3
24
-29
38
```

output

```
-3
-15
0
2
12
-15
19
```

Note

In the first example, $b_1 = 5$, $b_2 = -3$ and $b_3 = -2$ is another correct solution.

In the second example there are 6 possible solutions, one of them is shown in the example output.

B. Balanced Tunnel

time limit per test: 1 second
memory limit per test: 512 megabytes
input: standard input
output: standard output

Consider a tunnel on a one-way road. During a particular day, n cars numbered from 1 to n entered and exited the tunnel exactly once. All the cars passed through the tunnel at constant speeds.

A traffic enforcement camera is mounted at the tunnel entrance. Another traffic enforcement camera is mounted at the tunnel exit. *Perfectly balanced.*

Thanks to the cameras, the order in which the cars entered and exited the tunnel is known. No two cars entered or exited at the same time.

Traffic regulations prohibit overtaking inside the tunnel. If car i overtakes any other car j inside the tunnel, car i must be fined. However, each car can be fined at most once.

Formally, let's say that car i *definitely overtook* car j if car i entered the tunnel later than car j and exited the tunnel earlier than car j. Then, car i must be fined if and only if it definitely overtook at least one other car.

Find the number of cars that must be fined.

Input

The first line contains a single integer n ($2 \le n \le 10^5$), denoting the number of cars.

The second line contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le n$), denoting the ids of cars in order of entering the tunnel. All a_i are pairwise distinct.

The third line contains n integers $b_1, b_2, \ldots, b_n$ ($1 \le b_i \le n$), denoting the ids of cars in order of exiting the tunnel. All b_i are pairwise distinct.

Output

Output the number of cars to be fined.

Examples

input

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

output

```
2
```

input

```
7
5 2 3 6 7 1 4
2 3 6 7 1 4 5
```

output

```
6
```

input

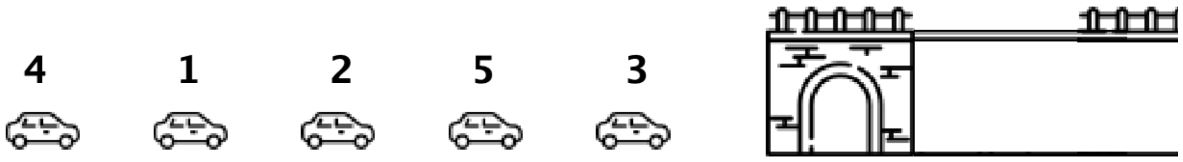
```
2
1 2
1 2
```

output

```
0
```

Note

The first example is depicted below:



Car 2 definitely overtook car 5, while car 4 definitely overtook cars 1, 2, 3 and 5. Cars 2 and 4 must be fined.

In the second example car 5 was definitely overtaken by all other cars.

In the third example no car must be fined.

C1. Balanced Removals (Easier)

time limit per test: 1 second
memory limit per test: 512 megabytes
input: standard input
output: standard output

This is an easier version of the problem. In this version, $n \le 2000$.

There are n distinct points in three-dimensional space numbered from 1 to n. The i-th point has coordinates (x_i, y_i, z_i). The number of points n is even.

You'd like to remove all n points using a sequence of $\frac{n}{2}$ snaps. In one snap, you can remove any two points a and b that have not been removed yet and form a *perfectly balanced* pair. A pair of points a and b is perfectly balanced if no other point c (that has not been removed yet) lies within the axis-aligned minimum bounding box of points a and b.

Formally, point c lies within the axis-aligned minimum bounding box of points a and b if and only if $\min(x_a, x_b) \le x_c \le \max(x_a, x_b)$, $\min(y_a, y_b) \le y_c \le \max(y_a, y_b)$, and $\min(z_a, z_b) \le z_c \le \max(z_a, z_b)$. Note that the bounding box might be degenerate.

Find a way to remove all points in $\frac{n}{2}$ snaps.

Input

The first line contains a single integer n ($2 \le n \le 2000$; n is even), denoting the number of points.

Each of the next n lines contains three integers x_i, y_i, z_i ($-10^8 \le x_i, y_i, z_i \le 10^8$), denoting the coordinates of the i-th point.

No two points coincide.

Output

Output $\frac{n}{2}$ pairs of integers a_i, b_i ($1 \le a_i, b_i \le n$), denoting the indices of points removed on snap i. Every integer between 1 and n, inclusive, must appear in your output exactly once.

We can show that it is always possible to remove all points. If there are many solutions, output any of them.

Examples

input

```
6
3 1 0
0 3 0
2 2 0
1 0 0
1 3 0
0 1 0
```

output

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

input

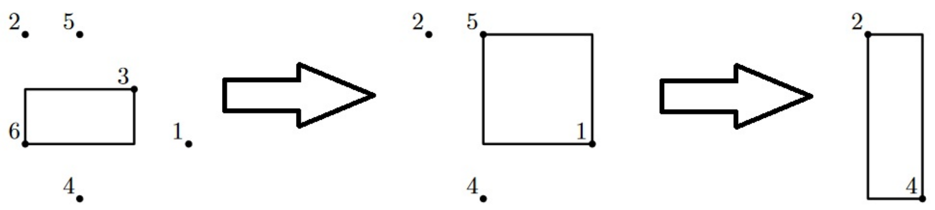
```
8
0 1 1
1 0 1
1 1 0
1 1 1
2 2 2
3 2 2
2 3 2
2 2 3
```

output

```
4 5
1 6
2 7
3 8
```

Note

In the first example, here is what points and their corresponding bounding boxes look like (drawn in two dimensions for simplicity, as all points lie on $z = 0$ plane). Note that order of removing matters: for example, points 5 and 1 don't form a perfectly balanced pair initially, but they do after point 3 is removed.



C2. Balanced Removals (Harder)

time limit per test: 1 second

memory limit per test: 512 megabytes
input: standard input
output: standard output

This is a harder version of the problem. In this version, $n \le 50\,000$.

There are n distinct points in three-dimensional space numbered from 1 to n. The i-th point has coordinates (x_i, y_i, z_i). The number of points n is even.

You'd like to remove all n points using a sequence of $\frac{n}{2}$ snaps. In one snap, you can remove any two points a and b that have not been removed yet and form a *perfectly balanced* pair. A pair of points a and b is perfectly balanced if no other point c (that has not been removed yet) lies within the axis-aligned minimum bounding box of points a and b.

Formally, point c lies within the axis-aligned minimum bounding box of points a and b if and only if $\min(x_a, x_b) \le x_c \le \max(x_a, x_b)$, $\min(y_a, y_b) \le y_c \le \max(y_a, y_b)$, and $\min(z_a, z_b) \le z_c \le \max(z_a, z_b)$. Note that the bounding box might be degenerate.

Find a way to remove all points in $\frac{n}{2}$ snaps.

Input

The first line contains a single integer n ($2 \le n \le 50\,000$; n is even), denoting the number of points.

Each of the next n lines contains three integers x_i, y_i, z_i ($-10^8 \le x_i, y_i, z_i \le 10^8$), denoting the coordinates of the i-th point.

No two points coincide.

Output

Output $\frac{n}{2}$ pairs of integers a_i, b_i ($1 \le a_i, b_i \le n$), denoting the indices of points removed on snap i. Every integer between 1 and n, inclusive, must appear in your output exactly once.

We can show that it is always possible to remove all points. If there are many solutions, output any of them.

Examples

input

```
6
3 1 0
0 3 0
2 2 0
1 0 0
1 3 0
0 1 0
```

output

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

input

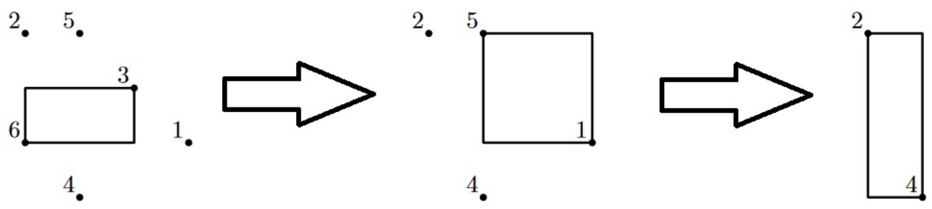
```
8
0 1 1
1 0 1
1 1 0
1 1 1
2 2 2
3 2 2
2 3 2
2 2 3
```

output

```
4 5
1 6
2 7
3 8
```

Note

In the first example, here is what points and their corresponding bounding boxes look like (drawn in two dimensions for simplicity, as all points lie on $z = 0$ plane). Note that order of removing matters: for example, points 5 and 1 don't form a perfectly balanced pair initially, but they do after point 3 is removed.



D. Balanced Playlist

time limit per test: 2 seconds
memory limit per test: 512 megabytes
input: standard input
output: standard output

Your favorite music streaming platform has formed a *perfectly balanced* playlist exclusively for you. The playlist consists of n tracks numbered from 1 to n. The playlist is automatic and cyclic: whenever track i finishes playing, track $i+1$ starts playing automatically; after track n goes track 1.

For each track i, you have estimated its *coolness* a_i. The higher a_i is, the cooler track i is.

Every morning, you choose a track. The playlist then starts playing from this track in its usual cyclic fashion. At any moment, you remember the maximum coolness x of already played tracks. Once you hear that a track with coolness **strictly** less than $\frac{x}{2}$ (no rounding) starts playing, you turn off the music immediately to keep yourself in a good mood.

For each track i, find out how many tracks you will listen to before turning off the music if you start your morning with track i, or determine that you will never turn the music off. Note that if you listen to the same track several times, every time must be counted.

Input

The first line contains a single integer n ($2 \le n \le 10^5$), denoting the number of tracks in the playlist.

The second line contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 10^9$), denoting coolnesses of the tracks.

Output

Output n integers $c_1, c_2, \ldots, c_n$, where c_i is either the number of tracks you will listen to if you start listening from track i or -1 if you will be listening to music indefinitely.

Examples

input

```
4
11 5 2 7
```

output

```
1 1 3 2
```

input

```
4
```

```
3 2 5 3
```

output

```
5 4 3 6
```

input

```
3
4 3 6
```

output

```
-1 -1 -1
```

Note

In the first example, here is what will happen if you start with...

- track 1: listen to track 1, stop as $a_2 < \frac{a_1}{2}$.
- track 2: listen to track 2, stop as $a_3 < \frac{a_2}{2}$.
- track 3: listen to track 3, listen to track 4, listen to track 1, stop as $a_2 < \frac{\max(a_3, a_4, a_1)}{2}$.
- track 4: listen to track 4, listen to track 1, stop as $a_2 < \frac{\max(a_4, a_1)}{2}$.

In the second example, if you start with track 4, you will listen to track 4, listen to track 1, listen to track 2, listen to track 3, listen to track 4 again, listen to track 1 again, and stop as $a_2 < \frac{max(a_4, a_1, a_2, a_3, a_4, a_1)}{2}$. Note that both track 1 and track 4 are counted twice towards the result.

E. Balanced Binary Search Trees

time limit per test: 3 seconds
memory limit per test: 512 megabytes
input: standard input
output: standard output

Recall that a binary search tree is a rooted binary tree, whose nodes each store a key and each have at most two distinguished subtrees, left and right. The key in each node must be greater than any key stored in the left subtree, and less than any key stored in the right subtree.

The depth of a vertex is the number of edges on the simple path from the vertex to the root. In particular, the depth of the root is 0.

Let's call a binary search tree *perfectly balanced* if there doesn't exist a binary search tree with the same number of vertices that has a strictly smaller sum of depths of its vertices.

Let's call a binary search tree with integer keys *striped* if both of the following conditions are satisfied for every vertex v:

- If v has a **left** subtree whose root is u, then the parity of the key of v is **different** from the parity of the key of u.
- If v has a **right** subtree whose root is w, then the parity of the key of v is the **same** as the parity of the key of w.

You are given a single integer n. Find the number of perfectly balanced striped binary search trees with n vertices that have distinct integer keys between 1 and n, inclusive. Output this number modulo $998\,244\,353$.

Input

The only line contains a single integer n ($1 \le n \le 10^6$), denoting the required number of vertices.

Output

Output the number of perfectly balanced striped binary search trees with n vertices and distinct integer keys between 1 and n, inclusive, modulo $998\,244\,353$.

Examples

input

```
4
```

output

```
1
```

input

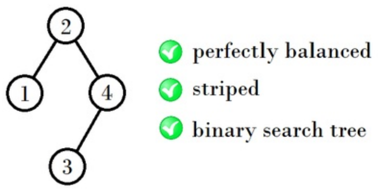
```
3
```

output

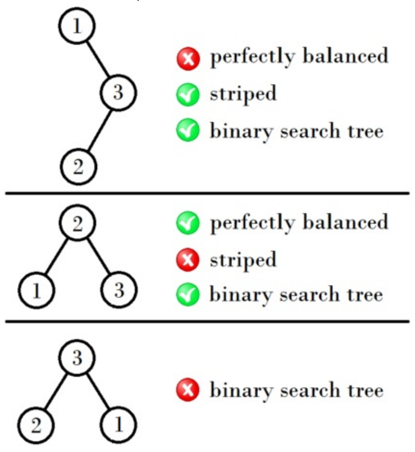
```
0
```

Note

In the first example, this is the only tree that satisfies the conditions:



In the second example, here are various trees that don't satisfy some condition:



F. Balanced Domino Placements

time limit per test: 2 seconds
memory limit per test: 512 megabytes
input: standard input
output: standard output

Consider a square grid with h rows and w columns with some dominoes on it. Each domino covers exactly two cells of the grid that share a common side. Every cell is covered by at most one domino.

Let's call a placement of dominoes on the grid *perfectly balanced* if no row and no column contains a pair of cells covered by two different dominoes. In other words, every row and column may contain no covered cells, one covered cell, or two covered cells that belong to the same domino.

You are given a perfectly balanced placement of dominoes on a grid. Find the number of ways to place zero or more extra dominoes on this grid to keep the placement perfectly balanced. Output this number modulo $998\,244\,353$.

Input

The first line contains three integers h, w, and n ($1 \le h, w \le 3600$; $0 \le n \le 2400$), denoting the dimensions of the grid and the number of already placed dominoes. The rows are numbered from 1 to h, and the columns are numbered from 1 to w.

Each of the next n lines contains four integers $r_{i,1}, c_{i,1}, r_{i,2}, c_{i,2}$ ($1 \le r_{i,1} \le r_{i,2} \le h$; $1 \le c_{i,1} \le c_{i,2} \le w$), denoting the row id and the column id of the cells covered by the i-th domino. Cells $(r_{i,1}, c_{i,1})$ and $(r_{i,2}, c_{i,2})$ are distinct and share a common side.

The given domino placement is perfectly balanced.

Output

Output the number of ways to place zero or more extra dominoes on the grid to keep the placement perfectly balanced, modulo $998\,244\,353$.

Examples

input

```
5 7 2
3 1 3 2
4 4 4 5
```

output

```
8
```

input

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

output

```
1
```

input

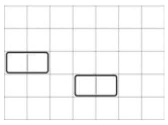
```
23 42 0
```

output

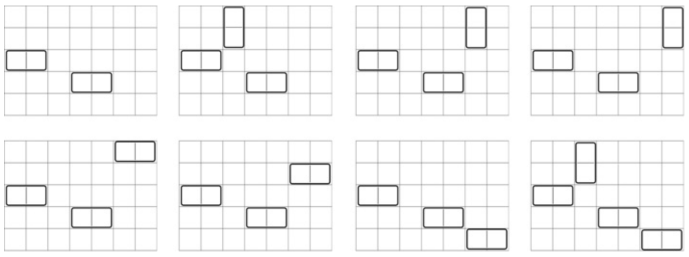
```
102848351
```

Note

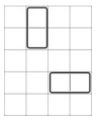
In the first example, the initial grid looks like this:



Here are 8 ways to place zero or more extra dominoes to keep the placement perfectly balanced:



In the second example, the initial grid looks like this:



No extra dominoes can be placed here.

G. Balanced Distribution

time limit per test: 2 seconds
memory limit per test: 512 megabytes
input: standard input
output: standard output

There are n friends living on a circular street. The friends and their houses are numbered clockwise from 0 to $n-1$.

Initially person i has a_i stones. The friends want to make the distribution of stones among them *perfectly balanced*: everyone should possess the same number of stones.

The only way to change the distribution of stones is by conducting *meetings*. During a meeting, people from exactly k consecutive houses (remember that the street is circular) gather at the same place and bring all their stones with them. All brought stones may be redistributed among people attending the meeting arbitrarily. The total number of stones they possess before the meeting and after the meeting must stay the same. After the meeting, everyone returns to their home.

Find a way to make the distribution of stones perfectly balanced conducting as few meetings as possible.

Input

The first line contains two integers n and k ($2 \le k < n \le 10^5$), denoting the number of friends and the size of each meeting.

The second line contains n integers $a_0, a_1, \ldots, a_{n-1}$ ($0 \le a_i \le 10^4$), denoting the number of stones people initially have.

The sum of all a_i is divisible by n.

Output

Output the minimum number of meetings m ($m \ge 0$), followed by m descriptions of meetings in chronological order.

The i-th description must consist of an integer s_i ($0 \le s_i < n$), followed by k non-negative integers $b_{i,0}, b_{i,1}, \ldots, b_{i,k-1}$ ($b_{i,j} \ge 0$). Such a description denotes a meeting of people $s_i, (s_i+1) \bmod n, \ldots, (s_i+k-1) \bmod n$, and $b_{i,j}$ denotes the number of stones person $(s_i+j) \bmod n$ must have after the i-th meeting. The sum of $b_{i,j}$ must match the total number of stones owned by these people before the i-th meeting.

We can show that a solution exists for any valid input, and any correct output contains at most 10^7 non-whitespace characters.

Examples

input

```
6 3
2 6 1 10 3 2
```

output

```
3
2 7 3 4
5 4 4 2
1 4 4 4
```

input

```
11 4
1 0 1 0 0 4 4 2 4 3 3
```

output

```
3
3 2 2 2 2
8 2 2 2 5
10 2 2 2 2
```

Note

In the first example, the distribution of stones changes as follows:

- after the first meeting: $2\ 6\ \mathbf{7}\ \mathbf{3}\ \mathbf{4}\ 2$;
- after the second meeting: $\mathbf{4}\ \mathbf{2}\ 7\ 3\ 4\ \mathbf{4}$;
- after the third meeting: $4\ \mathbf{4}\ \mathbf{4}\ \mathbf{4}\ 4\ 4$.

In the second example, the distribution of stones changes as follows:

- after the first meeting: $1\ 0\ 1\ \mathbf{2}\ \mathbf{2}\ \mathbf{2}\ \mathbf{2}\ 2\ 4\ 3\ 3$;
- after the second meeting: $\mathbf{5}\ 0\ 1\ 2\ 2\ 2\ 2\ 2\ \mathbf{2}\ \mathbf{2}\ \mathbf{2}$;
- after the third meeting: $\mathbf{2}\ \mathbf{2}\ \mathbf{2}\ 2\ 2\ 2\ 2\ 2\ 2\ 2\ \mathbf{2}$.

H. Balanced Reversals

time limit per test: 2 seconds
memory limit per test: 512 megabytes
input: standard input
output: standard output

You have two strings a and b of equal even length n consisting of characters `0` and `1`.

We're in the endgame now. To finally make the universe *perfectly balanced*, you need to make strings a and b equal.

In one step, you can choose any prefix of a of even length and reverse it. Formally, if $a = a_1 a_2 \ldots a_n$, you can choose a positive even integer $p \le n$ and set a to $a_p a_{p-1} \ldots a_1 a_{p+1} a_{p+2} \ldots a_n$.

Find a way to make a equal to b using at most $n+1$ reversals of the above kind, or determine that such a way doesn't exist. The number of reversals doesn't have to be minimized.

Input

The first line contains a single integer t ($1 \le t \le 2000$), denoting the number of test cases.

Each test case consists of two lines. The first line contains a string a of length n, and the second line contains a string b of the same length ($2 \le n \le 4000$; $n \bmod 2 = 0$). Both strings consist of characters `0` and `1`.

The sum of n over all t test cases doesn't exceed 4000.

Output

For each test case, if it's impossible to make a equal to b in at most $n+1$ reversals, output a single integer -1.

Otherwise, output an integer k ($0 \le k \le n+1$), denoting the number of reversals in your sequence of steps, followed by k even integers $p_1, p_2, \ldots, p_k$ ($2 \le p_i \le n$; $p_i \bmod 2 = 0$), denoting the lengths of prefixes of a to be reversed, in chronological order.

Note that k doesn't have to be minimized. If there are many solutions, output any of them.

Example

input

```
4
0100011011
1101011000
10101010
10101010
0011
1001
100011
110010
```

output

```
3
6 4 10
0

-1
7
2 6 2 6 2 2 6
```

Note

In the first test case, string a changes as follows:

- after the first reversal: **`100010`**`1011`;
- after the second reversal: **`0001`**`101011`;
- after the third reversal: **`1101011000`**.