Lyft Level 5 Challenge 2018 - Final Round (Open Div. 2)

A. The King's Race

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

On a chessboard with a width of n and a height of n, rows are numbered from bottom to top from 1 to n, columns are numbered from left to right from 1 to n. Therefore, for each cell of the chessboard, you can assign the coordinates (r,c), where r is the number of the row, and c is the number of the column.

The white king has been sitting in a cell with $(1,1)$ coordinates for a thousand years, while the black king has been sitting in a cell with (n,n) coordinates. They would have sat like that further, but suddenly a beautiful coin fell on the cell with coordinates (x,y)...

Each of the monarchs wanted to get it, so they decided to arrange a race according to slightly changed chess rules:

As in chess, the white king makes the first move, the black king makes the second one, the white king makes the third one, and so on. However, in this problem, **kings can stand in adjacent cells or even in the same cell at the same time**.

The player who reaches the coin first will win, that is to say, the player who reaches the cell with the coordinates (x,y) first will win.

Let's recall that the king is such a chess piece that can move one cell in all directions, that is, if the king is in the (a,b) cell, then in one move he can move from (a,b) to the cells $(a+1,b)$, $(a-1,b)$, $(a,b+1)$, $(a,b-1)$, $(a+1,b-1)$, $(a+1,b+1)$, $(a-1,b-1)$, or $(a-1,b+1)$. Going outside of the field is prohibited.

Determine the color of the king, who will reach the cell with the coordinates (x,y) first, if the white king moves first.

Input

The first line contains a single integer n ($2 \le n \le 10^{18}$) — the length of the side of the chess field.

The second line contains two integers x and y ($1 \le x,y \le n$) — coordinates of the cell, where the coin fell.

Output

In a single line print the answer "`White`" (without quotes), if the white king will win, or "`Black`" (without quotes), if the black king will win.

You can print each letter in any case (upper or lower).

Examples

input
```
4
2 3
```
output
```
White
```

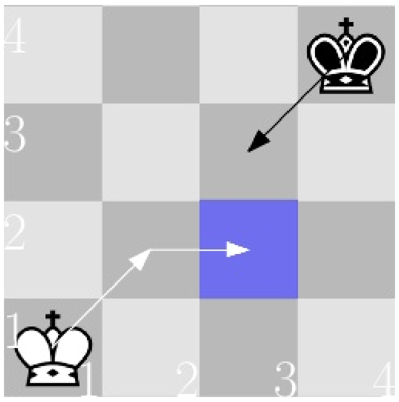
input
```
5
3 5
```
output
```
Black
```

input
```
2
2 2
```
output
```
Black
```

Note

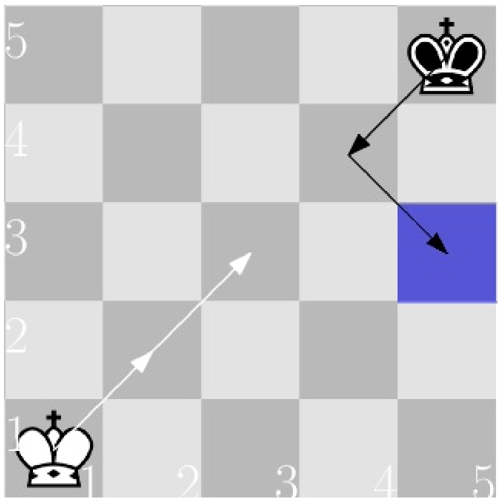
An example of the race from the first sample where both the white king and the black king move optimally:

1. The white king moves from the cell $(1,1)$ into the cell $(2,2)$.
2. The black king moves form the cell $(4,4)$ into the cell $(3,3)$.
3. The white king moves from the cell $(2,2)$ into the cell $(2,3)$. This is cell containing the coin, so the white king wins.

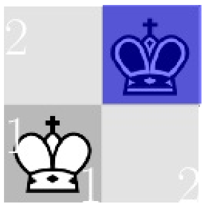


An example of the race from the second sample where both the white king and the black king move optimally:

1. The white king moves from the cell $(1,1)$ into the cell $(2,2)$.
2. The black king moves form the cell $(5,5)$ into the cell $(4,4)$.
3. The white king moves from the cell $(2,2)$ into the cell $(3,3)$.
4. The black king moves from the cell $(4,4)$ into the cell $(3,5)$. This is the cell, where the coin fell, so the black king wins.



In the third example, the coin fell in the starting cell of the black king, so the black king immediately wins.



B. Taxi drivers and Lyft

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

Palo Alto is an unusual city because it is an endless coordinate line. It is also known for the office of Lyft Level 5.

Lyft has become so popular so that it is now used by all m taxi drivers in the city, who every day transport the rest of the city residents — n riders.

Each resident (including taxi drivers) of Palo-Alto lives in its unique location (there is no such pair of residents that their coordinates are the same).

The Lyft system is very clever: when a rider calls a taxi, his call does not go to all taxi drivers, but only to the one that is the closest to that person. If there are multiple ones with the same distance, then to taxi driver with a smaller coordinate is selected.

But one morning the taxi drivers wondered: how many riders are there that would call the given taxi driver if they were the first to order a taxi on that day? In other words, you need to find for each taxi driver i the number a_i — the number of riders that would call the i-th taxi driver when all drivers and riders are at their home?

The taxi driver can neither transport himself nor other taxi drivers.

Input

The first line contains two integers n and m ($1 \le n, m \le 10^5$) — number of riders and taxi drivers.

The second line contains $n+m$ integers $x_1, x_2, \ldots, x_{n+m}$ ($1 \le x_1 < x_2 < \ldots < x_{n+m} \le 10^9$), where x_i is the coordinate where the i-th resident lives.

The third line contains $n+m$ integers $t_1, t_2, \ldots, t_{n+m}$ ($0 \le t_i \le 1$). If $t_i = 1$, then the i-th resident is a taxi driver, otherwise $t_i = 0$.

It is guaranteed that the number of i such that $t_i = 1$ is equal to m.

Output

Print m integers $a_1, a_2, \ldots, a_m$, where a_i is the answer for the i-th taxi driver. The taxi driver has the number i if among all the taxi drivers he lives in the i-th smallest coordinate (see examples for better understanding).

Examples

input

```
3 1
1 2 3 10
0 0 1 0
```

output

```
3
```

input

```
3 2
2 3 4 5 6
1 0 0 0 1
```

output

```
2 1
```

input

```
1 4
2 4 6 10 15
1 1 1 1 0
```

output

```
0 0 0 1
```

Note

In the first example, we have only one taxi driver, which means an order from any of n riders will go to him.

In the second example, the first taxi driver lives at the point with the coordinate 2, and the second one lives at the point with the coordinate 6. Obviously, the nearest taxi driver to the rider who lives on the 3 coordinate is the first one, and to the rider who lives on the coordinate 5 is the second one. The rider who lives on the 4 coordinate has the same distance to the first and the second taxi drivers, but since the first taxi driver has a smaller coordinate, the call from this rider will go to the first taxi driver.

In the third example, we have one rider and the taxi driver nearest to him is the fourth one.

C. The Tower is Going Home

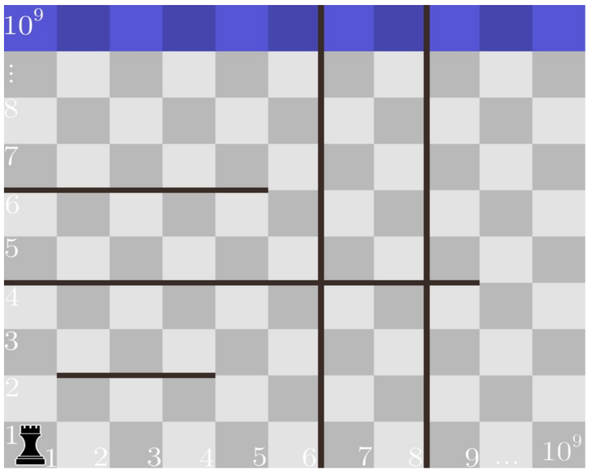
time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

On a chessboard with a width of 10^9 and a height of 10^9, the rows are numbered from bottom to top from 1 to 10^9, and the columns are numbered from left to right from 1 to 10^9. Therefore, for each cell of the chessboard you can assign the coordinates (x, y), where x is the column number and y is the row number.

Every day there are fights between black and white pieces on this board. Today, the black ones won, but at what price? Only the rook survived, and it was driven into the lower left corner — a cell with coordinates $(1, 1)$. But it is still happy, because the victory has been won and it's time to celebrate it! In order to do this, the rook needs to go home, namely — on the **upper side** of the field (that is, in any cell that is in the row with number 10^9).

Everything would have been fine, but the treacherous white figures put spells on some places of the field before the end of the game. There are two types of spells:

- Vertical. Each of these is defined by one number x. Such spells create an infinite *blocking line* between the columns x and $x+1$.
- Horizontal. Each of these is defined by three numbers x_1, x_2, y. Such spells create a *blocking segment* that passes through the top side of the cells, which are in the row y and in columns from x_1 to x_2 inclusive. The peculiarity of these spells is that it is **impossible** for a certain pair of such spells to have a common point. Note that horizontal spells can have common points with vertical spells.



An example of a chessboard.

Let's recall that the rook is a chess piece that in one move can move to any point that is in the same row or column with its initial position. In our task, the rook can move from the cell (r_0, c_0) into the cell (r_1, c_1) only under the condition that $r_1 = r_0$ or $c_1 = c_0$ and there is no *blocking lines* or *blocking segments* between these cells (For better understanding, look at the samples).

Fortunately, the rook can remove spells, but for this it has to put tremendous efforts, therefore, it wants to remove the minimum possible number of spells in such way, that after this it can return home. Find this number!

Input

The first line contains two integers n and m ($0 \le n, m \le 10^5$) — the number of vertical and horizontal spells.

Each of the following n lines contains one integer x ($1 \le x < 10^9$) — the description of the vertical spell. It will create a *blocking line* between the columns of x and $x + 1$.

Each of the following m lines contains three integers x_1, x_2 and y ($1 \le x_1 \le x_2 \le 10^9$, $1 \le y < 10^9$) — the numbers that describe the horizontal spell. It will create a *blocking segment* that passes through the top sides of the cells that are in the row with the number y, in columns from x_1 to x_2 inclusive.

It is guaranteed that all spells are different, as well as the fact that for each pair of horizontal spells it is true that the segments that describe them do not have common points.

Output

In a single line print one integer — the minimum number of spells the rook needs to remove so it can get from the cell $(1, 1)$ to at least one cell in the row with the number 10^9

Examples

input

```
2 3
6
8
1 5 6
1 9 4
2 4 2
```

output

```
1
```

input

```
1 3
4
1 5 3
1 9 4
4 6 6
```

output

```
1
```

input

```
0 2
1 1000000000 4
1 1000000000 2
```

output

```
2
```

input

```
0 0
```

output

```
0
```

input

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

output

```
2
```

Note

In the first sample, in order for the rook return home, it is enough to remove the second horizontal spell.

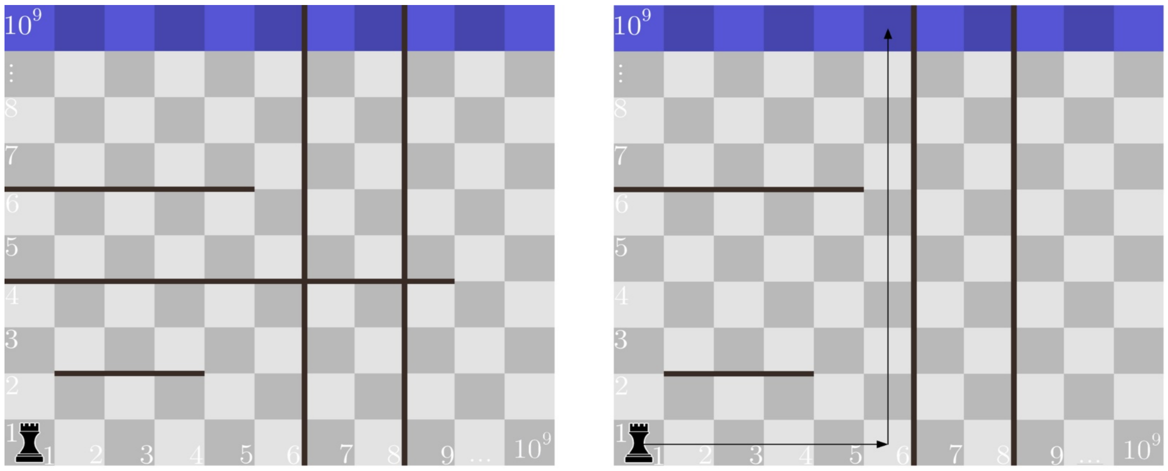


Illustration for the first sample. On the left it shows how the field looked at the beginning. On the right it shows how the field looked after the deletion of the second horizontal spell. It also shows the path, on which the rook would be going home.

In the second sample, in order for the rook to return home, it is enough to remove the only vertical spell. If we tried to remove just one of the horizontal spells, it would not allow the rook to get home, because it would be blocked from above by one of the remaining horizontal spells (either first one or second one), and to the right it would be blocked by a vertical spell.

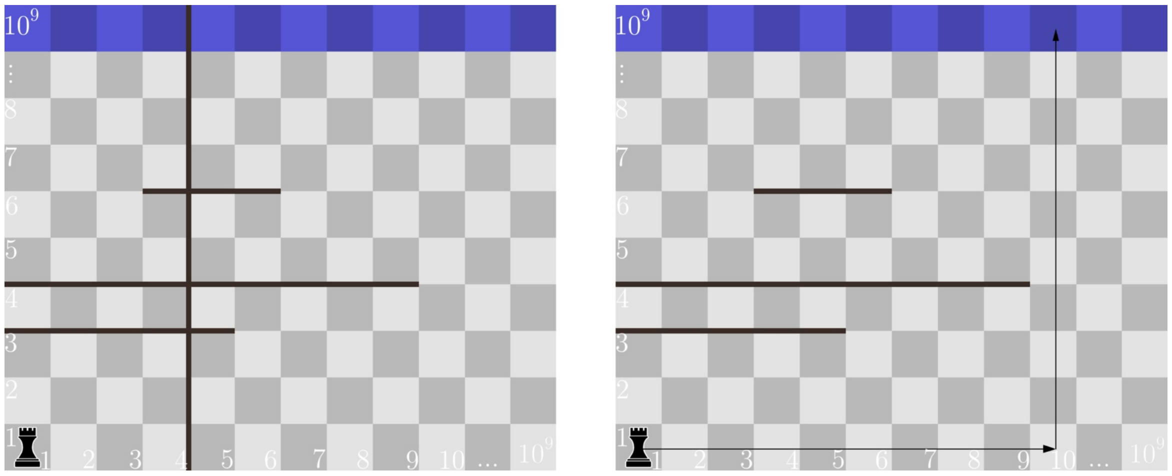


Illustration for the second sample. On the left it shows how the field looked at the beginning. On the right it shows how it looked after the deletion of the vertical spell. It also shows the path, on which the rook would be going home.

In the third sample, we have two horizontal spells that go through the whole field. These spells can not be bypassed, so we need to remove both of them.

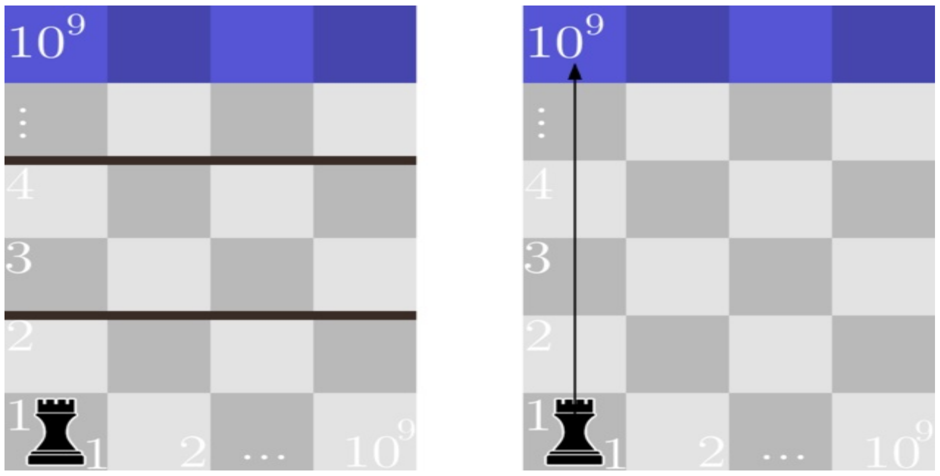


Illustration for the third sample. On the left it shows how the field looked at the beginning. On the right it shows how the field looked after the deletion of the horizontal spells. It also shows the path, on which the rook would be going home.

In the fourth sample, we have no spells, which means that we do not need to remove anything.

In the fifth example, we can remove the first vertical and third horizontal spells.

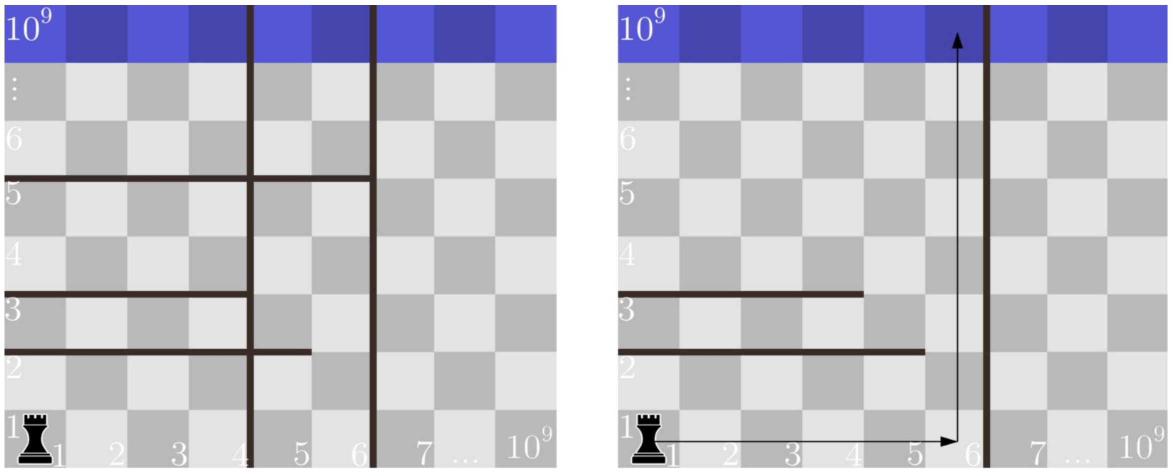


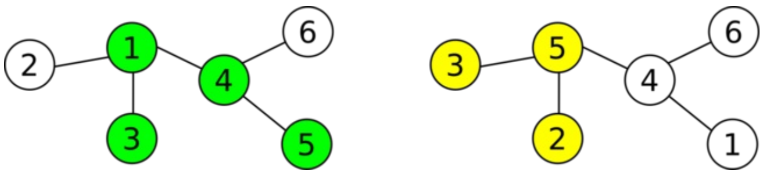
Illustration for the fifth sample. On the left it shows how the field looked at the beginning. On the right it shows how it looked after the deletions. It also shows the path, on which the rook would be going home.

D. Intersecting Subtrees

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

You are playing a strange game with Li Chen. You have a tree with n nodes drawn on a piece of paper. All nodes are unlabeled and distinguishable. Each of you **independently** labeled the vertices from 1 to n. Neither of you know the other's labelling of the tree.

You and Li Chen each chose a subtree (i.e., a connected subgraph) in that tree. Your subtree consists of the vertices labeled $x_1, x_2, \ldots, x_{k_1}$ in **your labeling**, Li Chen's subtree consists of the vertices labeled $y_1, y_2, \ldots, y_{k_2}$ in **his labeling**. The values of $x_1, x_2, \ldots, x_{k_1}$ and $y_1, y_2, \ldots, y_{k_2}$ are known to both of you.



The picture shows two labelings of a possible tree: yours on the left and Li Chen's on the right. The selected trees are highlighted. There are two common nodes.

You want to determine whether your subtrees have at least one common vertex. Luckily, your friend Andrew knows both labelings of the tree. You can ask Andrew at most 5 questions, each of which is in one of the following two forms:

- A x: Andrew will look at vertex x in your labeling and tell you the number of this vertex in Li Chen's labeling.
- B y: Andrew will look at vertex y in Li Chen's labeling and tell you the number of this vertex in your labeling.

Determine whether the two subtrees have at least one common vertex after asking some questions. If there is at least one common vertex, determine one of your labels for any of the common vertices.

Interaction

Each test consists of several test cases.

The first line of input contains a single integer t $(1 \le t \le 100)$ — the number of test cases.

For each testcase, your program should interact in the following format.

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

Each of the next $n-1$ lines contains two integers a_i and b_i $(1 \le a_i, b_i \le n)$ — the edges of the tree, indicating an edge between node a_i and b_i according to your labeling of the nodes.

The next line contains a single integer k_1 $(1 \le k_1 \le n)$ — the number of nodes in your subtree.

The next line contains k_1 distinct integers $x_1, x_2, \ldots, x_{k_1}$ $(1 \le x_i \le n)$ — the indices of the nodes in your subtree, according to your labeling. It is guaranteed that these vertices form a subtree.

The next line contains a single integer k_2 $(1 \le k_2 \le n)$ — the number of nodes in Li Chen's subtree.

The next line contains k_2 distinct integers $y_1, y_2, \ldots, y_{k_2}$ $(1 \le y_i \le n)$ — the indices (according to Li Chen's labeling) of the nodes in Li Chen's subtree. It is guaranteed that these vertices form a subtree according to Li Chen's labelling of the tree's nodes.

Test cases will be provided one by one, so you must complete interacting with the previous test (i.e. by printing out a common node or -1 if there is not such node) to start receiving the next one.

You can ask the Andrew two different types of questions.

- You can print "A x" ($1 \le x \le n$). Andrew will look at vertex x in your labeling and respond to you with the number of this vertex in Li Chen's labeling.
- You can print "B y" ($1 \le y \le n$). Andrew will look at vertex y in Li Chen's labeling and respond to you with the number of this vertex in your labeling.

You may only ask at most 5 questions per tree.

When you are ready to answer, print "C s", where s is your label of a vertex that is common to both subtrees, or -1, if no such vertex exists. Printing the answer does not count as a question. Remember to flush your answer to start receiving the next test case.

After printing a question do not forget to print 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.

If the judge responds with -1, it means that you asked more queries than allowed, or asked an invalid query. Your program should immediately terminate (for example, by calling `exit(0)`). You will receive `Wrong Answer`; it means that you asked more queries than allowed, or asked an invalid query. If you ignore this, you can get other verdicts since your program will continue to read from a closed stream.

Hack Format

To hack, use the following format. Note that you can only hack with one test case.

The first line should contain a single integer t ($t = 1$).

The second line should contain a single integer n ($1 \le n \le 1\,000$).

The third line should contain n integers $p_1, p_2, \ldots, p_n$ ($1 \le p_i \le n$) — a permutation of 1 to n. This encodes the labels that Li Chen chose for his tree. In particular, Li Chen chose label p_i for the node you labeled i.

Each of the next $n - 1$ lines should contain two integers a_i and b_i ($1 \le a_i, b_i \le n$). These edges should form a tree.

The next line should contain a single integer k_1 ($1 \le k_1 \le n$).

The next line should contain k_1 distinct integers $x_1, x_2, \ldots, x_{k_1}$ ($1 \le x_i \le n$). These vertices should form a subtree.

The next line should contain a single integer k_2 ($1 \le k_2 \le n$).

The next line should contain k_2 distinct integers $y_1, y_2, \ldots, y_{k_2}$ ($1 \le y_i \le n$). These vertices should form a subtree in Li Chen's tree according to the permutation above.

Examples

input

```
1
3
1 2
2 3
1
1
1
2
2
1
```

output

```
A 1
B 2
C 1
```

input

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

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

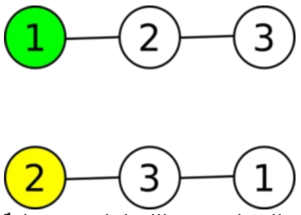
output

```
B 2
C 1
A 1
C -1
```

Note

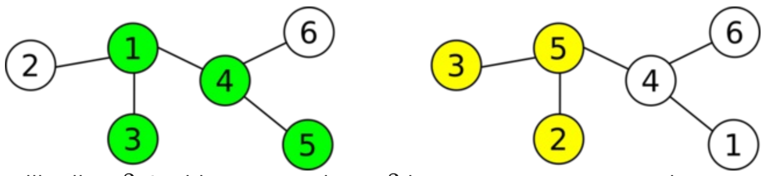
For the first sample, Li Chen's hidden permutation is $[2, 3, 1]$, and for the second, his hidden permutation is $[5, 3, 2, 4, 1, 6]$ for both cases.

In the first sample, there is a tree with three nodes in a line. On the top, is how you labeled the tree and the subtree you chose, and the bottom is how Li Chen labeled the tree and the subtree he chose:



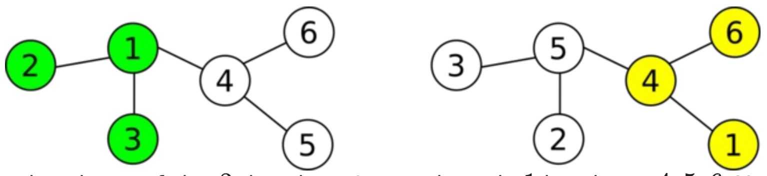
In the first question, you ask Andrew to look at node 1 in your labelling and tell you the label of it in Li Chen's labelling. Andrew responds with 2. At this point, you know that both of your subtrees contain the same node (i.e. node 1 according to your labeling), so you can output "C 1" and finish. However, you can also ask Andrew to look at node 2 in Li Chen's labelling and tell you the label of it in your labelling. Andrew responds with 1 (this step was given with the only reason — to show you how to ask questions).

For the second sample, there are two test cases. The first looks is the one from the statement:



We first ask "B 2", and Andrew will tell us 3. In this case, we know 3 is a common vertex, and moreover, any subtree with size 3 that contains node 3 must contain node 1 as well, so we can output either "C 1" or "C 3" as our answer.

In the second case in the second sample, the situation looks as follows:



In this case, you know that the only subtree of size 3 that doesn't contain node 1 is subtree $4, 5, 6$. You ask Andrew for the label of node 1 in Li Chen's labelling and Andrew says 5. In this case, you know that Li Chen's subtree doesn't contain node 1, so his subtree must be consist of the nodes $4, 5, 6$ (in your labelling), thus the two subtrees have no common nodes.

E. Optimal Polygon Perimeter

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

You are given n points on the plane. The polygon formed from all the n points is **strictly convex**, that is, the polygon is convex, and there are no three collinear points (i.e. lying in the same straight line). The points are numbered from 1 to n, in clockwise order.

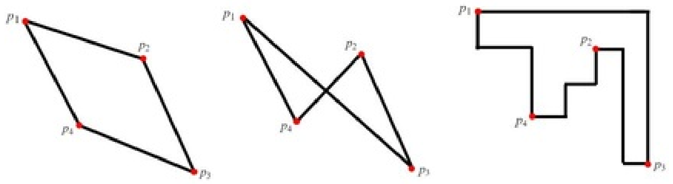
We define the distance between two points $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$ as their Manhattan distance:

$$d(p_1, p_2) = |x_1 - x_2| + |y_1 - y_2|.$$

Furthermore, we define the perimeter of a polygon, as the sum of Manhattan distances between all adjacent pairs of points on it; if the points on the polygon are ordered as $p_1, p_2, \ldots, p_k$ $(k \geq 3)$, then the perimeter of the polygon is $d(p_1, p_2) + d(p_2, p_3) + \ldots + d(p_k, p_1)$.

For some parameter k, let's consider all the polygons that can be formed from the given set of points, having **any** k vertices, such that the polygon is **not** self-intersecting. For each such polygon, let's consider its perimeter. Over all such perimeters, we define $f(k)$ to be the maximal perimeter.

Please note, when checking whether a polygon is self-intersecting, that the edges of a polygon are still drawn as straight lines. For instance, in the following pictures:



In the middle polygon, the order of points (p_1, p_3, p_2, p_4) is not valid, since it is a self-intersecting polygon. The right polygon (whose edges resemble the Manhattan distance) has the same order and is not self-intersecting, but we consider edges as straight lines. The correct way to draw this polygon is (p_1, p_2, p_3, p_4), which is the left polygon.

Your task is to compute $f(3), f(4), \ldots, f(n)$. In other words, find the maximum possible perimeter for each possible number of points (i.e. 3 to n).

Input

The first line contains a single integer n ($3 \leq n \leq 3 \cdot 10^5$) — the number of points.

Each of the next n lines contains two integers x_i and y_i ($-10^8 \leq x_i, y_i \leq 10^8$) — the coordinates of point p_i.

The set of points is guaranteed to be convex, all points are distinct, the points are ordered in clockwise order, and there will be no three collinear points.

Output

For each i ($3 \leq i \leq n$), output $f(i)$.

Examples

input

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

output

```
12 14
```

input

```
3
0 0
0 2
2 0
```

output

```
8
```

Note

In the first example, for $f(3)$, we consider four possible polygons:

- (p_1, p_2, p_3), with perimeter 12.
- (p_1, p_2, p_4), with perimeter 8.
- (p_1, p_3, p_4), with perimeter 12.
- (p_2, p_3, p_4), with perimeter 12.

For $f(4)$, there is only one option, taking all the given points. Its perimeter 14.

In the second example, there is only one possible polygon. Its perimeter is 8.

F. Deduction Queries

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

There is an array a of 2^{30} integers, indexed from 0 to $2^{30} - 1$. Initially, you know that $0 \leq a_i < 2^{30}$ ($0 \leq i < 2^{30}$), but you do not know any of the values. Your task is to process queries of two types:

- `1 l r x`: You are informed that the **bitwise xor** of the subarray $[l, r]$ (ends inclusive) is equal to x. That is, $a_l \oplus a_{l+1} \oplus \ldots \oplus a_{r-1} \oplus a_r = x$, where $\oplus$ is the bitwise xor operator. In some cases, the received update contradicts past updates. In this case, you should **ignore** the contradicting update (the current update).
- `2 l r`: You are asked to output the bitwise xor of the subarray $[l, r]$ (ends inclusive). If it is still impossible to know this value, considering all past updates, then output -1.

Note that the queries are **encoded**. That is, you need to write an **online** solution.

Input

The first line contains a single integer q ($1 \le q \le 2 \cdot 10^5$) — the number of queries.

Each of the next q lines describes a query. It contains one integer t ($1 \le t \le 2$) — the type of query.

The given queries will be **encoded** in the following way: let $last$ be the answer to the last query of the second type that you have answered (initially, $last = 0$). If the last answer was -1, set $last = 1$.

- If $t = 1$, three integers follow, l', r', and x' ($0 \le l', r', x' < 2^{30}$), meaning that you got an update. **First, do the following**:
$$l = l' \oplus last,\ r = r' \oplus last,\ x = x' \oplus last$$
and, if $l > r$, swap l and r.

  This means you got an update that the bitwise xor of the subarray $[l, r]$ is equal to x (notice that you need to ignore updates that contradict previous updates).

- If $t = 2$, two integers follow, l' and r' ($0 \le l', r' < 2^{30}$), meaning that you got a query. **First, do the following**:
$$l = l' \oplus last,\ r = r' \oplus last$$
and, if $l > r$, swap l and r.

  For the given query, you need to print the bitwise xor of the subarray $[l, r]$. If it is impossible to know, print -1. **Don't forget to change the value of $last$.**

It is guaranteed there will be at least one query of the second type.

Output

After every query of the second type, output the bitwise xor of the given subarray or -1 if it is still impossible to know.

Examples

input

```
12
2 1 2
2 1 1073741822
1 0 3 4
2 0 0
2 3 3
2 0 3
1 6 7 3
2 4 4
1 0 2 1
2 0 0
2 4 4
2 0 0
```

output

```
-1
-1
-1
-1
5
-1
6
3
5
```

input

```
4
1 5 5 9
1 6 6 5
1 6 5 10
2 6 5
```

output

```
12
```

Note

In the first example, the real queries (without being encoded) are:

- `12`
- `2 1 2`
- `2 0 1073741823`
- `1 1 2 5`
- `2 1 1`
- `2 2 2`
- `2 1 2`
- `1 2 3 6`
- `2 1 1`
- `1 1 3 0`
- `2 1 1`
- `2 2 2`
- `2 3 3`

- The answers for the first two queries are -1 because we don't have any such information on the array initially.
- The first update tells us $a_1 \oplus a_2 = 5$. Note that we still can't be certain about the values a_1 or a_2 independently (for example, it could be that $a_1 = 1, a_2 = 4$, and also $a_1 = 3, a_2 = 6$).
- After we receive all three updates, we have enough information to deduce a_1, a_2, a_3 independently.

In the second example, notice that after the first two updates we already know that $a_5 \oplus a_6 = 12$, so the third update is contradicting, and we ignore it.