ADA Lab 1 (Section A)

A. Fix Wiring (Easy Version)

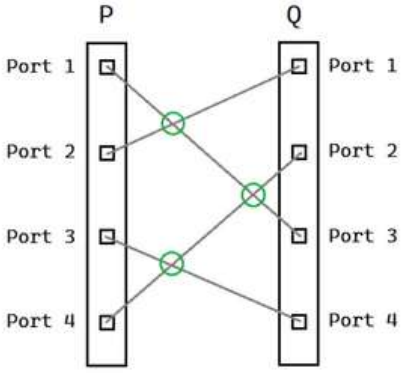
1 second, 256 megabytes

This is the easy version of the problem. The only difference between the two versions is the input constraints.

You are tasked with establishing connections between 2 networks, P and Q. Each network has many ports arranged in a vertical line. Port 1 is on top of this line, and port i is a unit distance below port $i-1$ for each $i \geq 2$. The port lines of both networks are parallel to each other.

To make the connections, you are given n wires, and n instructions, i-th instruction being of the form (p_i, q_i). This means that the i-th wire must be connected from port p_i of network P to port q_i of network Q.

For example, given $n = 4$ and instructions $(4, 2), (1, 3), (3, 4), (2, 1)$, the final connections are as follows:



After completing the circuit, you are perplexed by how complicated the circuit looks. You want to know the number of **wire intersections** in the resulting connections.

Two wires are said to *intersect* if they have different start ports, different end ports, and the two overlap at some point in their resulting connection.

In the above figure, wire overlaps have been circled. In this case, there are 3 wire overlaps.

Input

The first line is a single integer n $(1 \leq n \leq 10^3)$, the number of wires to be connected in the network.

n lines follow. The i-th line contains 2 space-separated integers p_i, q_i, $(1 \leq p_i, q_i \leq 10^9)$, denoting that the i-th write connects port p_i of line P to port q_i of line Q.

Output

Print a single integer - the number of wire overlaps observed.

input

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

output

```
3
```

input

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

output

```
6
```

In test 2, the wire intersections are as follows:

- Wire 1 intersects with wire 3.
- Wire 1 intersects with wire 6.
- Wire 2 intersects with wire 3.
- Wire 2 intersects with wire 6.
- Wire 3 intersects with wire 5.
- Wire 5 intersects with wire 6.

Note the following:

- Wire 3 does NOT intersect with wire 6. This is because, by the given definition, wires must start and end from different ports to count as an intersection.
- Wire 1 does NOT intersect with wire 5 for the same reason.

B. Fix Wiring (Hard Version)

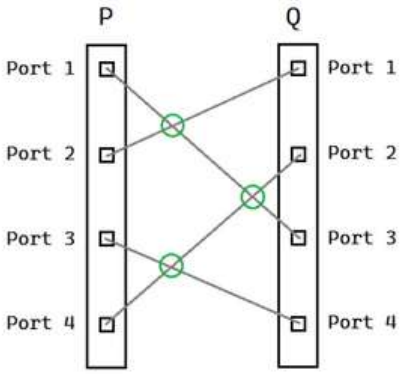
2.0 s, 256 megabytes

This is the hard version of the problem. The only difference between the two versions is the input constraints.

You are tasked with establishing connections between 2 networks, P and Q. Each network has many ports arranged in a vertical line. Port 1 is on top of this line, and port i is a unit distance below port $i-1$ for each $i \geq 2$. The port lines of both networks are parallel to each other.

To make the connections, you are given n wires, and n instructions, i-th instruction being of the form (p_i, q_i). This means that the i-th wire must be connected from port p_i of network P to port q_i of network Q.

For example, given $n = 4$ and instructions $(4, 2), (1, 3), (3, 4), (2, 1)$, the final connections are as follows:



After completing the circuit, you are perplexed by how complicated the circuit looks. You want to know the number of **wire intersections** in the resulting connections.

Two wires are said to *intersect* if they have different start ports, different end ports, and the two overlap at some point in their resulting connection.

In the above figure, wire overlaps have been circled. In this case, there are 3 wire overlaps.

Input

The first line is a single integer n $(1 \leq n \leq 2 \times 10^5)$, the number of wires to be connected in the network.

n lines follow. The i-th line contains 2 space-separated integers p_i, q_i, $(1 \leq p_i, q_i \leq 10^9)$, denoting that the i-th write connects port p_i of line P to port q_i of line Q.

Output

Print a single integer - the number of wire overlaps observed.

input

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

output

```
3
```

input

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

output

```
6
```

In test 2, the wire intersections are as follows:

- Wire 1 intersects with wire 3.
- Wire 1 intersects with wire 6.
- Wire 2 intersects with wire 3.
- Wire 2 intersects with wire 6.
- Wire 3 intersects with wire 5.
- Wire 5 intersects with wire 6.

Note the following:

- Wire 3 does NOT intersect with wire 6. This is because, by the given definition, wires must start and end from different ports to count as an intersection.
- Wire 1 does NOT intersect with wire 5 for the same reason.

C. Sloppy Shuffles 1

1 second, 256 megabytes

This is Part 1 of the problem. The output format and input constraints are different for both problems. It is advised to treat the two as different problems.

Given two binary strings A and B, A is a *sloppy shuffle* of B (equivalently, B is a *sloppy shuffle* of A) if and only if at least one of the following is true:

- A is equal to B
- A can be partitioned into binary strings A_1, A_2 of equal length, B can be partitioned into binary strings B_1, B_2 of equal length, and at least one of the following is true:
  - A_1 is a sloppy shuffle of B_1 and A_2 is a sloppy shuffle of B_2
  - A_1 is a sloppy shuffle of B_2 and A_2 is a sloppy shuffle of B_1

You have access to string B, and want to make changes to it such that B becomes a sloppy shuffle of A. To do so, you can perform several (possibly zero) *bit flips* on B. That is, for any bit B_i in B, you can change B_i from 0 to 1 or from 1 to 0.

What is the minimum number of bit flips you must perform on B to make it a sloppy shuffle of A?

Input

The first line contains a single integer T $(1 \le T \le 10)$, the number of test cases.

The first line of each test case is a single integer n $(1 \le n \le 10^3)$, the length of strings A and B.

The second and third lines of each test case contain the strings A and B, respectively.

Strings A and B consist of the characters "`0`" and "`1`" only.

Output

In the first line for each test case, print a single integer, the minimum number of bit flips required.

In the second line for each test case, print the modified string B, which is a sloppy shuffle of A.

In case there are multiple correct solutions, you may print any.

input

```
3
4
1101
0111
8
10100110
10011110
7
0001101
1110010
```

output

```
0
0111
1
10010110
7
0001101
```

In the first case, we can split A to get $A_1 = 11$, $A_2 = 01$, and split B to get $B_1 = 01$, $B_2 = 11$. Since $A_1 = B_2$ and $A_2 = B_1$, B is already a *sloppy shuffle* of A.

D. Sloppy Shuffles 2

1 second, 256 megabytes

This is Part 2 of the problem. The output format and input constraints are different for both problems. It is advised to treat the two as different problems.

Given two binary strings A and B, A is a *sloppy shuffle* of B (equivalently, B is a *sloppy shuffle* of A) if and only if at least one of the following is true:

- A is equal to B
- A can be partitioned into binary strings A_1, A_2 of equal length, B can be partitioned into binary strings B_1, B_2 of equal length, and at least one of the following is true:
  - A_1 is a sloppy shuffle of B_1 and A_2 is a sloppy shuffle of B_2
  - A_1 is a sloppy shuffle of B_2 and A_2 is a sloppy shuffle of B_1

Determine if A is a *sloppy shuffle* of B.

Input

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

The first line of each test case is a single integer n $(1 \le n \le 2 \times 10^5)$, the length of strings A and B.

The second and third lines of each test case contain the strings A and B, respectively.

Strings A and B consist of the characters "`0`" and "`1`" only.

The sum of n over all test cases does not exceed 2×10^5

Output

Print "`YES`" is A is a *sloppy shuffle* of B, and "`NO`" otherwise.

The output is case-insensitive. "`Yes`", "`yeS`", "`YeS`" are all acceptable outputs.

input

```
3
4
1101
0111
8
10100110
10011110
7
0001101
1110010
```

output

```
YES
NO
NO
```