## A. Balance the Bits

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

A sequence of brackets is called balanced if one can turn it into a valid math expression by adding characters '+' and '1'. For example, sequences '(())()', '()', and '(()(()))' are balanced, while ')(', '(()', and '(()))(' are not.

You are given a binary string $s$ of length $n$. Construct two balanced bracket sequences $a$ and $b$ of length $n$ such that for all $1 \le i \le n$:

- if $s_i = 1$, then $a_i = b_i$
- if $s_i = 0$, then $a_i \ne b_i$

If it is impossible, you should report about it.

### Input

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

The first line of each test case contains a single integer $n$ ($2 \le n \le 2 \cdot 10^5$, $n$ is even).

The next line contains a string $s$ of length $n$, consisting of characters 0 and 1.

The sum of $n$ across all test cases does not exceed $2 \cdot 10^5$.

### Output

If such two balanced bracked sequences exist, output "YES" on the first line, otherwise output "NO". You can print each letter in any case (upper or lower).

If the answer is "YES", output the balanced bracket sequences $a$ and $b$ satisfying the conditions on the next two lines.

If there are multiple solutions, you may print any.

### Example

**input**

```
3
6
101101
10
1001101101
4
1100
```

**output**

```
YES
()()()
((()))
YES
()()((()))
(())()()()
NO
```

### Note

In the first test case, $a=$"()()()" and $b=$"((()))". The characters are equal in positions $1$, $3$, $4$, and $6$, which are the exact same positions where $s_i = 1$.

In the second test case, $a=$"()()((()))" and $b=$"(())()()()". The characters are equal in positions $1, 4, 5, 7, 8, 10$, which are the exact same positions where $s_i = 1$.

In the third test case, there is no solution.

## B. 3-Coloring

time limit per test: 3 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

**This is an interactive problem.**

Alice and Bob are playing a game. There is $n \times n$ grid, initially empty. We refer to the cell in row $i$ and column $j$ by $(i, j)$ for $1 \leq i, j \leq n$. There is an infinite supply of tokens that come in $3$ colors labelled $1$, $2$, and $3$.

The game proceeds with turns as follows. Each turn begins with Alice naming one of the three colors, let's call it $a$. Then, Bob chooses a color $b \neq a$, chooses an empty cell, and places a token of color $b$ on that cell.

We say that there is a **conflict** if there exist two adjacent cells containing tokens of the same color. Two cells are considered adjacent if they share a common edge.

If at any moment there is a conflict, Alice wins. Otherwise, if $n^2$ turns are completed (so that the grid becomes full) without any conflicts, Bob wins.

We have a proof that Bob has a winning strategy. Play the game as Bob and win.

The interactor is **adaptive**. That is, Alice's color choices can depend on Bob's previous moves.

## Interaction

The interaction begins by reading a single integer $n$ ($2 \leq n \leq 100$) — the size of the grid.

The turns of the game follow. You should begin each turn by reading an integer $a$ ($1 \leq a \leq 3$) — Alice's chosen color.

Then you must print three integers $b, i, j$ ($1 \leq b \leq 3, b \neq a, 1 \leq i, j \leq n$) — denoting that Bob puts a token of color $b$ in the cell $(i, j)$. The cell $(i, j)$ must not contain a token from a previous turn. If your move is invalid or loses the game, the interaction is terminated and you will receive a **Wrong Answer** verdict.

After $n^2$ turns have been completed, make sure to exit immediately to avoid getting unexpected 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.

### Hack Format

To hack, use the following format.

The first line contains a single integer $n$ ($2 \leq n \leq 100$).

The second line contains $n^2$ integers $a_1, \ldots, a_{n^2}$ ($1 \leq a_i \leq 3$), where $a_i$ denotes Alice's color on the $i$-th turn.

The interactor might deviate from the list of colors in your hack, but only if it forces Bob to lose.

### Example

**input**

```
2
1

2

1

3
```

**output**

```
2 1 1

3 1 2

3 2 1

1 2 2
```

## Note

The final grid from the sample is pictured below. Bob wins because there are no two adjacent cells with tokens of the same color.

$$\begin{matrix} 2 & 3 \\ 3 & 1 \end{matrix}$$

The sample is only given to demonstrate the input and output format. It is not guaranteed to represent an optimal strategy for Bob or the real behavior of the interactor.

# C. Travelling Salesman Problem

time limit per test: 2 seconds

memory limit per test: 256 megabytes

input: standard input

output: standard output

There are $n$ cities numbered from $1$ to $n$, and city $i$ has beauty $a_i$.

A salesman wants to start at city $1$, visit every city exactly once, and return to city $1$.

For all $i \neq j$, a flight from city $i$ to city $j$ costs $\max(c_i, a_j - a_i)$ dollars, where $c_i$ is the price floor enforced by city $i$. Note that there is no absolute value. Find the minimum total cost for the salesman to complete his trip.

## Input

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

The $i$-th of the next $n$ lines contains two integers $a_i$, $c_i$ ($0 \le a_i, c_i \le 10^9$) — the beauty and price floor of the $i$-th city.

## Output

Output a single integer — the minimum total cost.

### Examples

**input**

```
3
1 9
2 1
4 1
```

**output**

```
11
```

**input**

```
6
4 2
8 4
3 0
2 3
7 1
0 1
```

**output**

```
13
```

## Note

In the first test case, we can travel in order $1 \to 3 \to 2 \to 1$.

- The flight $1 \to 3$ costs $\max(c_1, a_3 - a_1) = \max(9, 4-1) = 9$.
- The flight $3 \to 2$ costs $\max(c_3, a_2 - a_3) = \max(1, 2-4) = 1$.
- The flight $2 \to 1$ costs $\max(c_2, a_1 - a_2) = \max(1, 1-2) = 1$.

The total cost is $11$, and we cannot do better.

## D. Flip the Cards

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

There is a deck of $n$ cards. The $i$-th card has a number $a_i$ on the front and a number $b_i$ on the back. Every integer between $1$ and $2n$ appears exactly once on the cards.

A deck is called sorted if the front values are in **increasing** order and the back values are in **decreasing** order. That is, if $a_i < a_{i+1}$ and $b_i > b_{i+1}$ for all $1 \le i < n$.

To flip a card $i$ means swapping the values of $a_i$ and $b_i$. You must flip some subset of cards (possibly, none), then put all the cards in any order you like. What is the minimum number of cards you must flip in order to sort the deck?

## Input

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

The next $n$ lines describe the cards. The $i$-th of these lines contains two integers $a_i, b_i$ ($1 \le a_i, b_i \le 2n$). Every integer between $1$ and $2n$ appears exactly once.

## Output

If it is impossible to sort the deck, output "-1". Otherwise, output the minimum number of flips required to sort the deck.

### Examples

**input**

```
5
3 10
6 4
```

```
1 9
5 8
2 7
```

**output**

```
2
```

**input**

```
2
1 2
3 4
```

**output**

```
-1
```

**input**

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

**output**

```
-1
```

**Note**

In the first test case, we flip the cards $(1,9)$ and $(2,7)$. The deck is then ordered $(3,10),(5,8),(6,4),(7,2),(9,1)$. It is sorted because $3<5<6<7<9$ and $10>8>4>2>1$.

In the second test case, it is impossible to sort the deck.

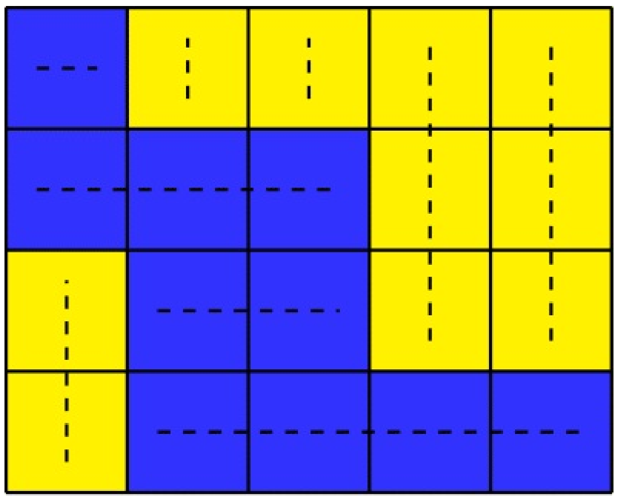
## E. 2-Coloring

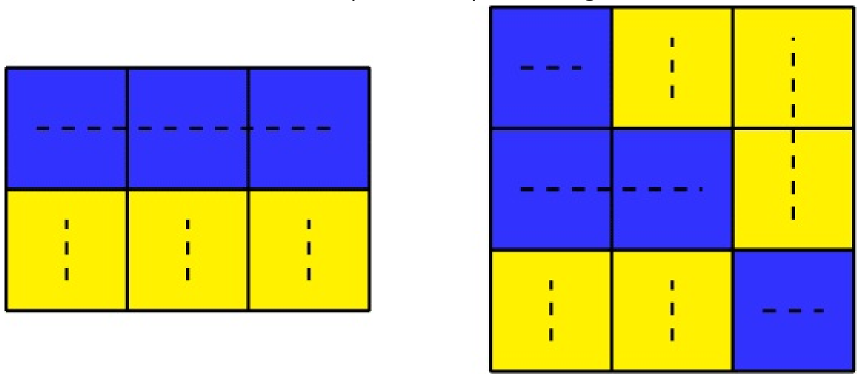
time limit per test: 3 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

There is a grid with $n$ rows and $m$ columns. Every cell of the grid should be colored either blue or yellow.

A coloring of the grid is called stupid if every row has exactly one segment of blue cells and every column has exactly one segment of yellow cells.

In other words, every row must have at least one blue cell, and all blue cells in a row must be consecutive. Similarly, every column must have at least one yellow cell, and all yellow cells in a column must be consecutive.


An example of a stupid coloring.


Examples of clever colorings. The first coloring is missing a blue cell in the second row, and the second coloring has two yellow

segments in the second column.

How many stupid colorings of the grid are there? Two colorings are considered different if there is some cell that is colored differently.

## Input

The only line contains two integers $n$, $m$ ($1 \le n, m \le 2021$).

## Output

Output a single integer — the number of stupid colorings modulo $998244353$.

### Examples

**input**

```
2 2
```

**output**

```
2
```

**input**

```
4 3
```

**output**

```
294
```

**input**

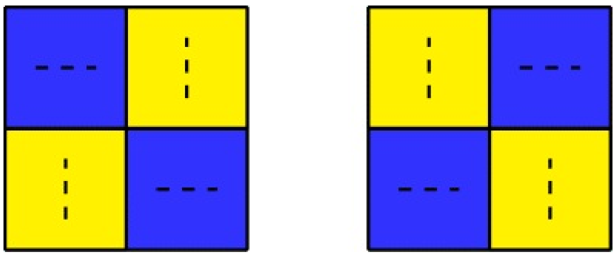
```
2020 2021
```

**output**

```
50657649
```

## Note

In the first test case, these are the only two stupid $2 \times 2$ colorings.



## F. Balance the Cards

time limit per test: 3 seconds

memory limit per test: 256 megabytes

input: standard input

output: standard output

A balanced bracket sequence is defined as an integer sequence that can be built with the following rules:

- The empty sequence is balanced.
- If $[a_1, \dots, a_n]$ and $[b_1, \dots, b_m]$ are balanced, then their concatenation $[a_1, \dots, a_n, b_1, \dots, b_m]$ is balanced.
- If $x$ is a positive integer and $[a_1, \dots, a_n]$ is balanced, then $[x, a_1, \dots, a_n, -x]$ is balanced.

The positive numbers can be imagined as opening brackets and the negative numbers as closing brackets, where matching brackets must have the same type (absolute value). For example, $[1, 2, -2, -1]$ and $[1, 3, -3, 2, -2, -1]$ are balanced, but $[1, 2, -1, -2]$ and $[-1, 1]$ are not balanced.

There are $2n$ cards. Each card has a number on the front and a number on the back. Each integer $1, -1, 2, -2, \dots, n, -n$ appears exactly once on the front of some card and exactly once on the back of some (not necessarily the same) card.

You can reorder the cards however you like. You are **not** allowed to flip cards, so numbers cannot move between the front and back. Your task is to order the cards so that the sequences given by the front numbers and the back numbers are both balanced, or report that it is impossible.

## Input

The first line contains a single integer $n$ ($1 \le n \le 2 \cdot 10^5$) — the number of bracket types, and half the number of cards.

The next $2n$ lines describe the cards. The $i$-th of these lines contains two integers $a_i$, $b_i$ ($-n \le a_i, b_i \le n$, $a_i \ne 0$, $b_i \ne 0$) — the numbers on the front and back of the $i$-th card, respectively. Every integer $1, -1, 2, -2, \dots, n, -n$ appears exactly once as $a_i$ and exactly once as $b_i$.

## Output

On the first line, output "YES" if it's possible to reorder these cards to satisfy the condition. Otherwise, output "NO". You can print each letter in any case (upper or lower).

If it is possible, on the next $2n$ lines output the cards in an order so that the front and back are both balanced. If there are multiple solutions, you may output any.

**Examples**

**input**

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

**output**

```
YES
1 3
4 -3
-4 4
-1 -4
5 1
3 -1
2 2
-2 5
-3 -5
-5 -2
```

**input**

```
2
1 1
-1 2
2 -1
-2 -2
```

**output**

```
NO
```

## Note

In the first test case, the front numbers create the balanced sequence $[1, 4, -4, -1, 5, 3, 2, -2, -3, -5]$ and the back numbers create the balanced sequence $[3, -3, 4, -4, 1, -1, 2, 5, -5, -2]$.

In the second test case, the cards are given in an order so that the front numbers are balanced, but the back numbers create the unbalanced sequence $[1, 2, -1, -2]$. If we swapped the second and third cards, we would balance the back numbers and unbalance the front numbers. But there is no order that balances both.