Codeforces Round #712 (Div. 2)

A. Déjà Vu

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

A palindrome is a string that reads the same backward as forward. For example, the strings "z", "aaa", "aba", and "abccba" are palindromes, but "codeforces" and "ab" are not. You hate palindromes because they give you déjà vu.

There is a string s. You **must** insert **exactly one** character 'a' somewhere in s. If it is possible to create a string that is **not** a palindrome, you should find one example. Otherwise, you should report that it is impossible.

For example, suppose $s =$ "cbabc". By inserting an 'a', you can create "acbabc", "cababc", "cbaabc", "cbabac", or "cbabca". However "cbaabc" is a palindrome, so you must output one of the other options.

Input

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

The only line of each test case contains a string s consisting of lowercase English letters.

The total length of all strings does not exceed $3 \cdot 10^5$.

Output

For each test case, if there is no solution, output "NO".

Otherwise, output "YES" followed by your constructed string of length $|s| + 1$ on the next line. If there are multiple solutions, you may print any.

You can print each letter of "YES" and "NO" in any case (upper or lower).

Example

input

```
6
cbabc
ab
zza
ba
a
nutforajaroftuna
```

output

```
YES
cbabac
YES
aab
YES
zaza
YES
baa
NO
YES
nutforajarofatuna
```

Note

The first test case is described in the statement.

In the second test case, we can make either "aab" or "aba". But "aba" is a palindrome, so "aab" is the only correct answer.

In the third test case, "zaza" and "zzaa" are correct answers, but not "azza".

In the fourth test case, "baa" is the only correct answer.

In the fifth test case, we can only make "aa", which is a palindrome. So the answer is "NO".

In the sixth test case, "anutforajaroftuna" is a palindrome, but inserting 'a' elsewhere is valid.

B. Flip the Bits

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

output: standard output

There is a binary string a of length n. In one operation, you can select any prefix of a with an **equal** number of 0 and 1 symbols. Then all symbols in the prefix are inverted: each 0 becomes 1 and each 1 becomes 0.

For example, suppose $a = 0111010000$.

- In the first operation, we can select the prefix of length 8 since it has four 0's and four 1's: $[01110100]00 \to [10001011]00$.
- In the second operation, we can select the prefix of length 2 since it has one 0 and one 1: $[10]00101100 \to [01]00101100$.
- It is illegal to select the prefix of length 4 for the third operation, because it has three 0's and one 1.

Can you transform the string a into the string b using some finite number of operations (possibly, none)?

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 ($1 \le n \le 3 \cdot 10^5$) — the length of the strings a and b.

The following two lines contain strings a and b of length n, consisting of symbols 0 and 1.

The sum of n across all test cases does not exceed $3 \cdot 10^5$.

Output

For each test case, output "YES" if it is possible to transform a into b, or "NO" if it is impossible. You can print each letter in any case (upper or lower).

Example

input

```
5
10
0111010000
0100101100
4
0000
0000
3
001
000
12
010101010101
100110011010
6
000111
110100
```

output

```
YES
YES
NO
YES
NO
```

Note

The first test case is shown in the statement.

In the second test case, we transform a into b by using zero operations.

In the third test case, there is no legal operation, so it is impossible to transform a into b.

In the fourth test case, here is one such transformation:

- Select the length 2 prefix to get 100101010101.
- Select the length 12 prefix to get 011010101010.
- Select the length 8 prefix to get 100101011010.
- Select the length 4 prefix to get 011001011010.
- Select the length 6 prefix to get 100110011010.

In the fifth test case, the only legal operation is to transform a into 111000. From there, the only legal operation is to return to the string we started with, so we cannot transform a into b.

C. 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.

D. 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 $1 \le i, j \le n$. 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 $b \ne a$, 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 ($2 \le n \le 100$) — the size of the grid.

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

Then you must print three integers b, i, j $(1 \le b \le 3, b \ne a, 1 \le i, j \le n)$ — 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 $(2 \le n \le 100)$.

The second line contains n^2 integers $a_1, \ldots, a_{n^2}$ $(1 \le a_i \le 3)$, 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.

E. 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 $i \ne j$, 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.

F. 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.