Problem Description

In the heart of the Quantum Lattice, n entangled qubit resonators form the foundation of reality stabilization. Each resonator i can exist in a *ground state* (0) or an *excited state* (1). The Lattice is protected by m Chrono-Parity Safeguards—ancient protocols established by the Cosmic Weavers to prevent quantum decoherence.

Each safeguard is defined by a tuple (t, l, r, j):

- **Type A** ($t = 0$): If the segment $[l, r]$ contains an *even* number of resonators, then for every resonator $i \in [l, r]$, if i is in ground state, resonator j *must* be excited.
- **Type B** ($t = 1$): If the segment $[l, r]$ contains an *odd* number of resonators, then for every resonator $i \in [l, r]$, if j is excited, resonator i *must* be in ground state.

Your mission is to determine whether a valid state configuration exists that satisfies **all** active Chrono-Parity Safeguards. Note that a safeguard is only active when its parity condition holds; otherwise, it imposes no constraints.

Formally, given integers n and m, and m quadruples (t, l, r, j), decide if there exists a binary array $s[1..n]$ such that for every quadruple:

- If $t = 0$ and $(r - l + 1) \equiv 0 \pmod 2$, then $\forall i \in [l, r],\ s[i] = 0 \implies s[j] = 1$.
- If $t = 1$ and $(r - l + 1) \equiv 1 \pmod 2$, then $\forall i \in [l, r],\ s[j] = 1 \implies s[i] = 0$.

This problem lies at the intersection of constraint satisfaction and combinatorial parity logic, requiring deep structural insight to resolve within stringent computational limits.

Input

The first line contains two integers n and m ($1 \le n, m \le 10^5$).

Each of the next m lines contains four integers t, l, r, j ($t \in \{0, 1\}$, $1 \le l \le r \le n$, $1 \le j \le n$), describing a Chrono-Parity Safeguard.

Output

Print `YES` if a valid state assignment exists, otherwise print `NO`.

Constraints

- $1 \le n, m \le 10^5$
- $1 \le l \le r \le n$
- $1 \le j \le n$
- $t \in \{0, 1\}$ for all safeguards

Sample

Sample Input:

```
2 2
0 1 2 1
0 1 2 2
```

Sample Output:

```
YES
```

Sample Input:

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

Sample Output:

```
NO
```