1 Input Format

line 1: n m, where n is the number of contestants and m is the number of judges
line i: Votes of judge i, where positive numbers a are votes for contestant a, and negative numbers \(-a\) are votes against contestant a
Ranges: \(2 \leq n \leq 1000\), \(1 \leq m \leq 2000\), \(a \in \{1, \ldots, n\}\)

The input always consists of exactly one test case.

2 Output Format

The literal string yes, or the literal string no

3 Model

For each contestant c, let \(x_c\) be a variable which is true if and only if contestant c advances. The vote of a judge for a contestant a can then be expressed as the literal \(x_a\), and against an contestant \(a\) as the literal \(\overline{x_a}\). Each judge has exactly two votes \((v, w)\), where \(v\) and \(w\) are each literals. The Problem then reduces to the question whether the following expression is satisfiable:

\[
\bigwedge_{(v, w) \in \text{Judges}} (v \lor w) \land (\overline{v} \Rightarrow w) \land (\overline{w} \Rightarrow v)
\]

Which is equivalent to the implication form

\[
\bigwedge_{(v, w) \in \text{Judges}} (v \Rightarrow \overline{w}) \land (\overline{v} \Rightarrow w)
\]

4 Solution

Construct an implication graph from the input: For each contestant c, create two nodes (for \(x_c\) and \(\overline{x_c}\)). For each judge, insert two edges according to the implications of the judge’s votes.

Run Tarjan’s algorithm to find the strongly connected components. Finally, check whether for any \(i\) both \(x_i\) and \(\overline{x_i}\) lie in the same scc.

Function Tarjan\((G = (V, E))\)

\[
\begin{align*}
\text{index} & \leftarrow 0 \\
S & \leftarrow \text{new stack} \\
\text{for } v \in V & \text{ do} \\
& v.\text{lowlink} \leftarrow \infty \\
& v.\text{index} \leftarrow \infty \\
\text{for } v \in V & \text{ do} \\
& \text{if } v.\text{index} = \infty \text{ then} \\
& \quad \text{TarjanDFS}(v, S, G)
\end{align*}
\]

Function TarjanDFS\((v, S, G = (V, E))\)

\[
\begin{align*}
& v.\text{index} \leftarrow \text{index} \\
& v.\text{lowlink} \leftarrow \text{index} \\
& S.\text{push}(v) \\
& \text{index} \leftarrow \text{index} + 1 \\
& \text{for each } (v, w) \in E \text{ do} \\
& \quad \text{if } w.\text{index} = \infty \text{ then} \\
& \quad \quad \text{TarjanDFS}(w, S, G) \\
& \quad \quad v.\text{lowlink} \leftarrow \min(v.\text{lowlink}, w.\text{lowlink}) \\
& \quad \text{else if } w \text{ is on the stack then} \\
& \quad \quad v.\text{lowlink} \leftarrow \min(v.\text{lowlink}, w.\text{index}) \\
& \quad \text{if } v.\text{lowlink} = v.\text{index} \text{ then} \\
& \quad \quad \text{start new scc} \\
& \quad \quad \text{repeat} \\
& \quad \quad \quad w \leftarrow S.\text{pop()} \\
& \quad \quad \quad \text{add } w \text{ to scc} \\
& \quad \quad \text{until } w = v \\
& \text{end scc}
\end{align*}
\]

5 Implementation

• Map \(-n, \ldots, -1, 1, \ldots, n\) to \(0, \ldots, 2n\) to store nodes in array
• Efficient “is on stack” check: store flag for each node
• Find a good format for SCC information (flag or set)