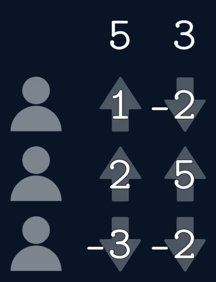
Inconspicuous Hacking

Handout

Tim Gerlach Max Mündlein

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: $2 \le \texttt{n} \le 1000$, $1 \le \texttt{m} \le 2000$, $a \in \{1, \ldots, n\}$



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:

$$\bigwedge_{(v,w)\,\in\,\text{Judges}} (v \lor w)$$

Which is equivalent to the implication form

$$\bigwedge_{(v,w)\,\in\,\text{Judges}} (\overline{v} \implies w) \land (\overline{w} \implies v)$$

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))
    index ← 0
    S ← new stack
    for v ∈ V do
        v.lowlink ← ∞
        v.index ← ∞
    for v ∈ V do
        if v.index = ∞ then
            TarjanDFS(v, S, G)

Function TarjanDFS(v, S, G = (V, E))
    v.index ← index
    v.lowlink ← index
    S.push(v)
    index ← index + 1
    for each (v, w) ∈ E do
        if w.index = ∞ then
            TarjanDFS(w, S, G)
            v.lowlink ← min(v.lowlink,
             w.lowlink)
        else if w is on the stack then
            v.lowlink ← min(v.lowlink, w.index)
    if v.lowlink = v.index then
        start new scc
        repeat
            w ← S.pop()
            add w to scc
        until w = v
        end scc
```

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)