题目描述

大家在《离散数学2》中会学习到半连通的概念。对于一个有向图，我们称其是半连通的，当且仅当对于图中的任意两点 u 和 v，在点 u 和点 v 之间至少存在一条路径使得要么点 u 能到达点 v，要么点 v 能到达点 u。

例如一个 3 个点的图，图中的边为 $1 \to 2, 1 \to 3, 3 \to 2$，则该图是半连通的，因为对于点 1 和 2，1 可到达 2；对于点 1 和 3，1 可到达 3；对于点 2 和 3，3 可到达 2。但对于 4 个点的图 $2 \to 1, 3 \to 1, 4 \to 1, 3 \to 2, 4 \to 2$ 不是半连通的，因为点 3 和点 4 之间 3 无法到达 4，4 也无法到达 3。

现在我们给出一个**有向无环图**，请你判断其是不是半连通的。

输入

第一行一个正整数 T $(1 \le T \le 100)$ 表示数据组数。

接下来 T 组数据:

第一行两个正整数 n, m $(1 \le n, m \le 10^4)$，分别表示图中点的个数和边的个数。

接下来 m 行，每行两个正整数 u_i, v_i $(1 \le u_i, v_i \le n)$ 表示图中有一条 $u_i \to v_i$ 的有向边。

输出

对于每组数据，若给出的图是半连通的则输出 `YES`，否则输出 `NO`。

输入样例

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

输出样例

```
YES
NO
```

题解

在有向无环图（DAG）中，每个顶点通过有向边与其他顶点连接。由于是无环的，意味着你不能从一个顶点出发经过一系列的边后回到这个顶点，这个性质确保了可以对图进行拓扑排序。

拓扑排序 是将DAG中的所有顶点排列成线性序列，且对于每一对顶点 u 和 v，如果存在一条从 u 到 v 的边，那么在序列中 u 必然出现在 v 之前。这样的一个排序序列反映了顶点间的依赖关系。

注意拓扑排序中**只保证**：如果存在u到v的边，那么拓扑序列中u一定在v的前面

而**不能保证**：如果拓扑序列中u在v的前面，那么存在u到v的边

半连通图 的定义是：对于图中的任意两个顶点 u 和 v，要么 u 可以到达 v，要么 v 可以到达 u

对于DAG和拓扑排序，我们给出以下结论：

DAG是半连通的，当且仅当其拓扑序列中任意相邻顶点之间存在路径

下面给出证明：

- 如果DAG是半连通的，那么对于拓扑排序中的任意连续有序顶点对 $<u, v>$，节点 u 必然能够到达节点v，这意味着拓扑序列中任意相邻顶点之间存在路径。
- 如果在拓扑序列中，对于每一对连续顶点 u 和 v 都存在从 u 到 v 的边，或者存在从 v 到 u 的边，那么对于任何顶点对 $<x, y>$ ，如果它们是相邻的，那么它们之间存在路径；如果它们之间存在其他顶点，那么x可以通过一系列的边到达 y（即存在一条路径通过排序中位于 x 和 y 之间的其他顶点）。因此，DAG是半连通的。

所以问题转化为，如何判断拓扑序列是否满足任意相邻顶点之间都存在路径。

当我们利用**入度和出度**来生成拓扑序列的时候，只需要注意：当生成拓扑序列的队列中存在多于1个顶点时，这些顶点之间是平行关系，是不存在路径的。

- 如果在初始情况下将所有入度为0的顶点入队，如果存在两个及以上入度为0的顶点，那么这些顶点之间一定无法到达
- 如果在某次我们取出队列中的点，让与之相关联的顶点的入度-1，出现了两个及以上入度为0的顶点，那么这些顶点之间也是无法到达的

所以我们只需要判断在生成拓扑序列的过程中，队列中的顶点数是否始终为1(对应代码第30行)

标程代码

```
1  #include <bits/stdc++.h>
2
3  using namespace std;
```

```

int main() {
    ios::sync_with_stdio(false);
    cin.tie(0);
    int tt;
    cin >> tt;
    while (tt--) {
        int n, m;
        cin >> n >> m;
        vector<vector<int>> g(n);
        vector<int> in(n);
        for (int i = 0; i < m; i++) {
            int u, v;
            cin >> u >> v;
            --u; --v;
            g[u].emplace_back(v);
            in[v]++;
        }
        queue<int> q;
        for (int i = 0; i < n; i++) {
            if (in[i] == 0) {
                q.push(i);
            }
        }
        bool ok = true;
        while (!q.empty()) {
            if (q.size() != 1) {
                ok = false;
                break;
            }
            int s = q.front();
            q.pop();
            for (auto t : g[s]) {
                if (--in[t] == 0) {
                    q.push(t);
                }
            }
        }
        cout << (ok ? "YES" : "NO") << '\n';
    }
    return 0;
}
```