Flight Routes Check

There are n cities and m flight connections.

Your task is to check if you can travel from any city to any other city using the available flights.

Input

The first input line has two integers n and m: the number of cities and flights.
The cities are numbered 1,2,...,n.

After this, there are m lines describing the flights.
Each line has two integers a and b: there is a flight from city a to city b.

All flights are one-way flights.

Output

Print "YES" if all routes are possible, and "NO" otherwise.

In the latter case also print two cities a and b such that you cannot travel from city a to city b.

If there are several possible solutions, you can print any of them.

Constraints

1 <= n <= 10^5
1 <= m <= 2.10^5
1 <= a,b <= n

Example

Input:
```
4 5
1 2
2 3
3 1
1 4
3 4
```

Output:
```
NO
4 2
```

Problem Breakdown

Lets understand the problem first

Given a directed graph with n nodes and m edges,
determine if every node is reachable from every other node (i.e., graph is strongly connected).

We must:

Print YES if it's strongly connected.

Otherwise, print NO and a pair of nodes a b such that a can't reach b.

Intuition and Core Idea

What does strongly connected mean?

For every pair of nodes u and v, there must exist a path from u to v and a path from v to u.

Let's simplify this:

If node 1 can reach every node, and

Every node can reach node 1,

then the graph is strongly connected.

✅ So we can test both directions using DFS/BFS.

Step-by-Step Approach Using DFS

Step 1: DFS from node 1 on the original graph

Check if all nodes are reachable from node 1.

Step 2: Reverse the graph

Flip all edges. If there was an edge from u → v, make it v → u.

original graph

reversed graph

Step 3: DFS again from node 1 on the reversed graph

Check if all nodes can reach node 1.

Step 4: Output

If either DFS fails to cover all nodes, print NO and the first failing node.

Otherwise, print YES.

Why reverse graph?

To simulate: "Can every node reach node 1?" → same as "Can node 1 reach every node in reversed graph?"

Algorithm

```cpp
#include <bits/stdc++.h>
using namespace std;

void dfs(int node, vector<vector<int>>& graph, vector<bool>& visited) {
    if (visited[node]) return;
    visited[node] = true;

    for (int neighbor : graph[node]) {
        if (!visited[neighbor]) {
            dfs(neighbor, graph, visited);
        }
    }
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    int n, m;
    cin >> n >> m;

    vector<vector<int>> graph(n + 1);
    vector<vector<int>> reversed_graph(n + 1);

    for (int i = 0; i < m; ++i) {
        int u, v;
        cin >> u >> v;
        graph[u].push_back(v);
        reversed_graph[v].push_back(u);
    }

    vector<bool> visited(n + 1, false);
    dfs(1, graph, visited);

    for (int i = 1; i <= n; ++i) {
        if (!visited[i]) {
            cout << "NO\n";
            cout << "1 " << i << "\n";
            return 0;
        }
    }

    fill(visited.begin(), visited.end(), false);
    dfs(1, reversed_graph, visited);

    for (int i = 1; i <= n; ++i) {
        if (!visited[i]) {
            cout << "NO\n";
            cout << i << " 1\n";
            return 0;
        }
    }

    cout << "YES\n";
    return 0;
}
```

Simple DFS traversal to mark all nodes reachable

Reading input and creating
graph and reversed graph as well

Checking in original graph
weather we can go from node 1 to
all the other nodes

if not we print
1 -> i

Again we check weather we can go
reach node 1 from other nodes as well

if not print
i -> 1

Time Complexity

O(n+m)

Space Complexity

O(n+m)