Longest Path

There is a directed graph G with N vertices and M edges. The vertices are numbered 1,2, ...,N, and for each i (1 ≤ i ≤ M), the i-th directed edge goes from Vertex x_i to y_i.
G does not contain directed cycles.

Find the length of the longest directed path in G.
Here, the length of a directed path is the number of edges in it.

Input

Input is given from Standard Input in the following format:

N M
x1 y1
x2 y2
:
:
xn yn

Output

Print the length of the longest directed path in G.

Constraints

2 <= N <= 2 . 10^5
1 <= M <= 2 . 10^5
1 <= x_i,y_i <= n

Example

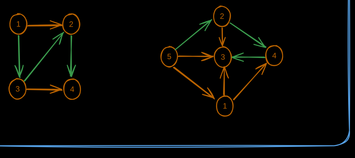
Input:
```
4 5
1 2
1 3
3 2
2 4
3 4
```
Output:
```
3
```

Problem Breakdown

Lets understand the problem first

You are given a Directed Acyclic Graph (DAG) with N vertices and M edges.
Each edge is directed x → y.

You need to find the length of the longest directed path in the DAG.
(A path's length = number of edges.)



Since the graph has no cycles,
we can order its nodes topologically —
so that every edge goes from an earlier node → a later node.

This property makes DAGs perfect for DP,
because we can process nodes in that topological order —
when we visit a node, we already know all its predecessors' results.

Define DP state

```
dp[v] = length of the longest path that ends at node v
```

That means:

- dp[v] = 0 if no edges go into v (it's a start node)
- otherwise dp[v] = max(dp[u] + 1) for all edges u → v

Transition formula

For every directed edge u → v:

```
dp[v] = max(dp[v], dp[u] + 1)
```

This means:

If we can reach v from u, then a path ending at u
can be extended by one more edge to reach v.

Algorithm overview

We can do this using Topological Sort + DP:

1. Build adjacency list.
2. Compute indegree of every node.
3. Perform topological sort using Kahn's algorithm (queue).
4. For each node in topo order:

- For every edge u → v,
  update dp[v] = max(dp[v], dp[u] + 1).

5. The answer = max(dp[i]) for all nodes.

Code

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

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

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

  vector<vector<int>> adj(n + 1);
  vector<int> indeg(n + 1, 0);
  vector<int> dp(n + 1, 0);

  for(int i = 0; i < m; ++i) {
    int x, y;
    cin >> x >> y;
    adj[x].push_back(y);
    indeg[y]++;
  }

  //Topological Sort
  queue<int> q;
  for(int i = 1; i <= n; ++i) {
    if(indeg[i] == 0) q.push(i);
  }

  vector<int> topo;
  while(!q.empty()) {
    int u = q.front(); q.pop();
    topo.push_back(u);
    for(int v : adj[u]) {
      indeg[v]--;
      if(indeg[v] == 0) q.push(v);
    }
  }

  // DP in Topological order
  for(int u : topo) {
    for(int v : adj[u]) {
      dp[v] = max(dp[v], dp[u] + 1);
    }
  }

  int ans = 0;
  for(int i = 1; i <= n; ++i) {
    ans = max(ans, dp[i]);
  }

  cout << ans << "\n";
}
```

Complexity

Building Graph -> O(m)

Topological Sort -> O(n + m)

DP propagation -> O(n + m)

Total -> O(n + m)

Memory -> O(n + m)