Independent Set

There is a tree with N vertices, numbered 1,2,…,N. For each i (1≤i≤N−1), the i-th edge connects Vertex x_i and y_i.

Taro has decided to paint each vertex in white or black. Here, it is not allowed to paint two adjacent vertices both in black.

Find the number of ways in which the vertices can be painted, modulo 10^9 +7.

Input

Input is given from Standard Input in the following format:

N
x_1 y_1
x_2 y_2
.
.
.
x_n-1 y_n-1

Output

Print the length of the longest directed path in G.

Constraints

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

Example

Input:
3
1 2
2 3

Output:

5

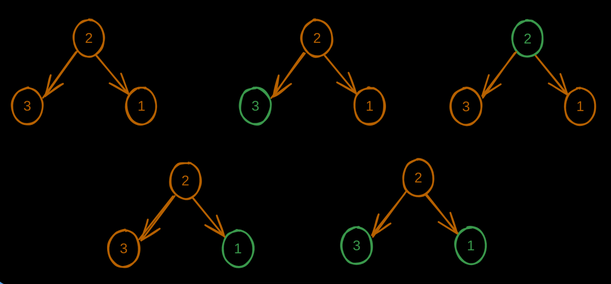
Problem Breakdown

Lets understand the problem first

We are given a tree with N vertices.

Each vertex can be white or black, but:
- No two adjacent vertices can both be black.

We must count the number of valid colorings, and output the answer mod 1e9 + 7.



We need to count all independent sets in a tree.

An independent set is a set of vertices where no two vertices are directly connected by an edge.

Each valid black-white coloring corresponds to one independent set (the black nodes form the set).

Define DP states

```
dp[v][0] = number of valid colorings of subtree rooted at v,
           when v is painted white.

dp[v][1] = number of valid colorings of subtree rooted at v,
           when v is painted black.
```

Transition logic (bottom-up DP)

Case 1 → v is white:

If the current node is white,
then its children can be either black or white (no restriction).

So for every child u:

```
Total ways = dp[u][0] + dp[u][1]
```

We multiply over all children:

```
dp[v][0] = ∏ (dp[u][0] + dp[u][1])   for all children u
```

Case 2 → v is black:

If the current node is black,
then none of its children can be black (because adjacent blacks not allowed).

So each child must be white:

```
dp[v][1] = ∏ (dp[u][0])   for all children u
```

Base case

At a leaf node:

- dp[v][0] = 1 (it can be white)

- dp[v][1] = 1 (it can also be black)

Code

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

const int MOD = 1e9 + 7;
const int MAXN = 100005;

vector<int> adj[MAXN];
long long dp[MAXN][2]; // dp[v][0] = white, dp[v][1] = black

void dfs(int v, int parent) {
  dp[v][0] = dp[v][1] = 1; // base case

  for(int u : adj[v]) {
    if(u == parent) continue;

    dfs(u, v);

    // If v is white -> children can be white or black
    dp[v][0] = (dp[v][0] * (dp[u][0] + dp[u][1]) % MOD) % MOD;

    // If v is black -> children must be white
    dp[v][1] = (dp[v][1] * dp[u][0]) % MOD;
  }
}

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

  int n;
  cin >> n;
  for(int i = 0; i < n - 1; ++i) {
    int a, b;
    cin >> a >> b;
    adj[a].push_back(b);
    adj[b].push_back(a);
  }

  dfs(1, 0);

  long long ans = (dp[1][0] + dp[1][1]) % MOD;

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