Tree Matching

You are given a tree consisting of n nodes.

A matching is a set of edges where each node is an endpoint of at most one edge. What is the maximum number of edges in a matching?

Input

The first input line contains an integer n: the number of nodes. The nodes are numbered 1,2,...,n.

Then there are n-1 lines describing the edges. Each line contains two integers a and b: there is an edge between nodes a and b.

Output

Print one integer: the maximum number of pairs.

Constraints

1 <= n <= 2 . 10^5
1 <= a,b <= n

Example

Input:

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

Output:

```
2
```

Problem Breakdown

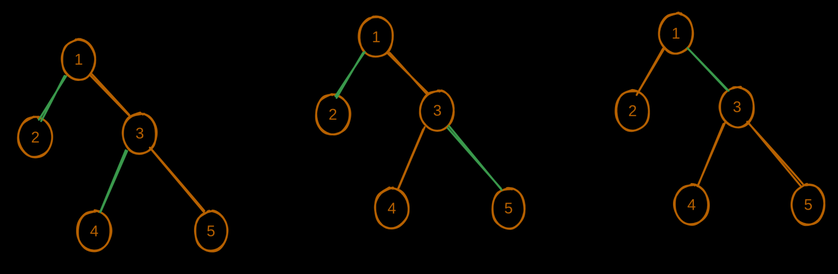
Lets understand the problem first

You're given a tree with n nodes.

We need to find:

- The maximum number of edges you can choose such that no two edges share a vertex.

That's the definition of a maximum matching on a tree.



(1, 2) and (3, 4)

(1, 2) and (3, 5)

(1, 3)

DP states definition

Let's define for each node v:

- dp[v][0]: maximum matching size in the subtree rooted at v when v is NOT matched with its parent.

- dp[v][1]: maximum matching size in the subtree rooted at v when v IS matched with one of its children.

We will compute these values via DFS bottom-up.

Transitions

Case - 1 → v is NOT matched (dp[v][0])

If v is not matched with its parent or any child, then each child's subtree is independent.

Each child u can either:

- be matched (use one of its own children), or
- not be matched.

So we take the maximum from both choices for each child:

```
dp[v][0] = Σ over all children u of v [ max(dp[u][0], dp[u][1]) ]
```

That's like "children are free".

Case - 2 → v IS matched (dp[v][1])

Now v is matched with exactly one child (say, w).

If we match v with u:

- We gain +1 edge (the edge (v,u))

- For that child u, we can only take configurations where u is not matched with its own children (dp[u][0])

- For other children (not matched with v), we can take max(dp[w][0], dp[w][1]) freely.

So we try every child as the matching partner and take the best:

```
dp[v][1] = max over all children u {
    1 + dp[u][0] + Σ over all other children w!=u [ max(dp[w][0], dp[w][1]) ]
}
```

Code

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

const int MAXN = 200005;
vector<int> adj[MAXN];
int n;
long long dp[MAXN][2];
bool vis[MAXN];

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

    long long sum = 0;
    for(int u : adj[v]) {
      if(u == parent) continue;
      dfs(u, v);
      sum += max(dp[u][0], dp[u][1]);
    }

    dp[v][0] = sum;

    long long best = 0;
    for(int u : adj[v]) {
      if(u == parent) continue;

      long long candidate = 1 + dp[u][0] + (sum - max(dp[u][0], dp[u][1]));

      best = max(best, candidate);
    }

    dp[v][1] = best;
}

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

  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);

  cout << max(dp[1][0], dp[1][1]) << "\n";

  return 0;
}
```