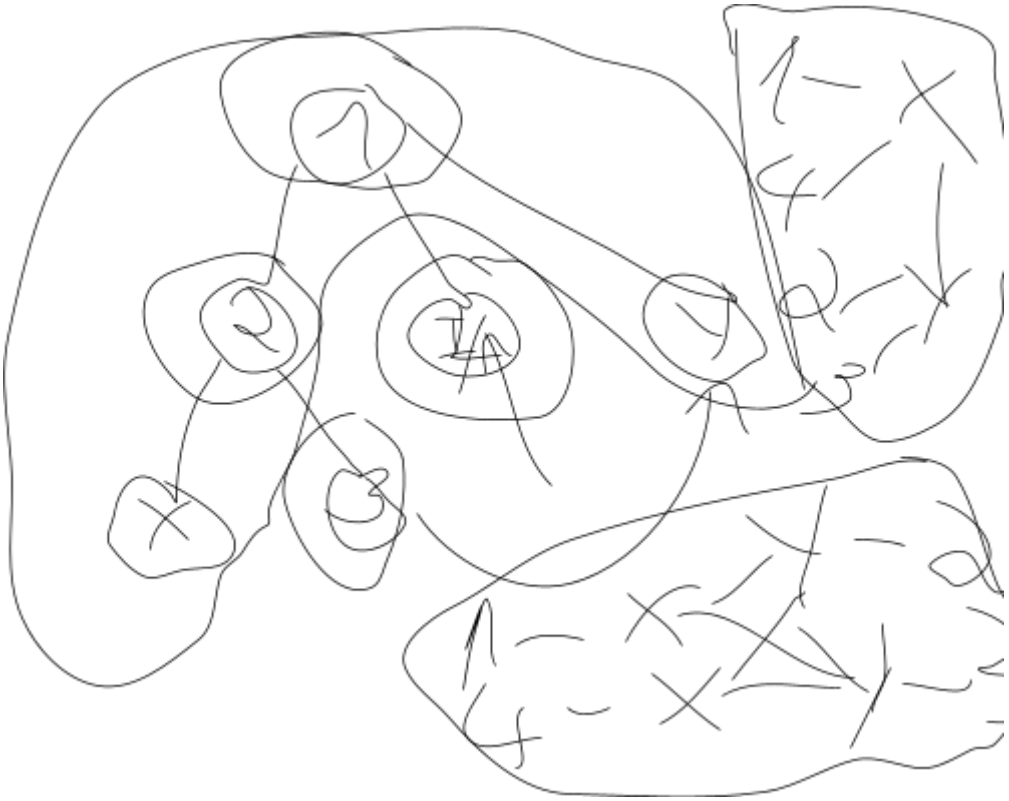
Tree is a connected graph having n nodes and n - 1 edges.
For every node i and node j, there is only 1 distinct shortest path between them.
Diameter is the longest path in a tree;
Leaf is a node don’t have any child.

P1:



Most distant relatives of some node i only can be either x or y

=> how do we check that some nodes belong to the same tree?

Connect i -> p[i], p[i] -> i

After that, we just need to count the number of connected components

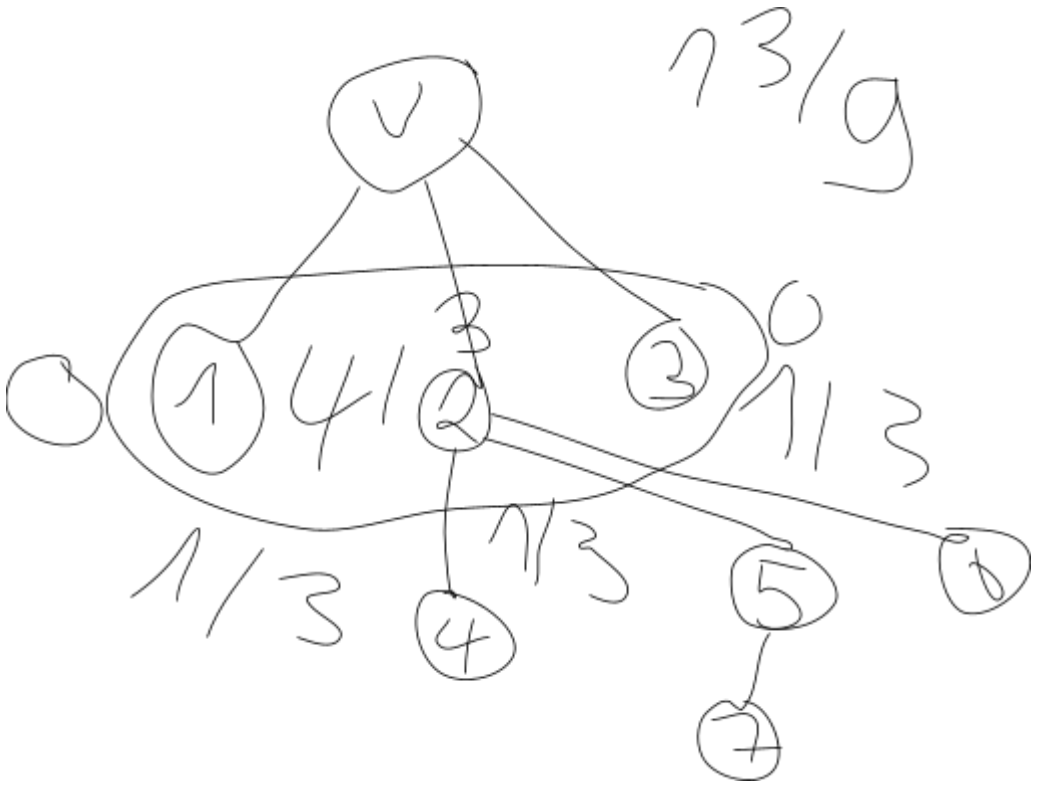
All nodes in this tree will be connected to x or y, that its belong to the same connected components when we traverse from every node in this tree.

```
vector<int> adj[maxn];

adj[i].push_back(p[i]);
adj[p[i]].push_back(i);
Int trees = 0;
For (int i=1;i<=n;i++) if (!visited[i]) dfs(i), trees++;

Cout <<trees;
```

P2:
Expected value =

Dynamic Programming

1. Base case and final results
2. Dp configuration(number of states, number of dimensions in dp representation, the definition of dp, which value dp store)
3. Update formula



We need to find the expected length of v' children and then use these values to find the expected length of v

Dp[u] = the expected length when the horse start from u
Base case, u is leaf => dp[u] = 0;

Int dp[maxn];

Void

Cout <<fixed<<setprecision(6)<<dfs(1,0);