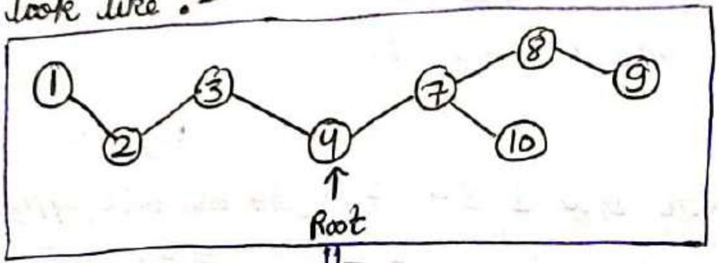
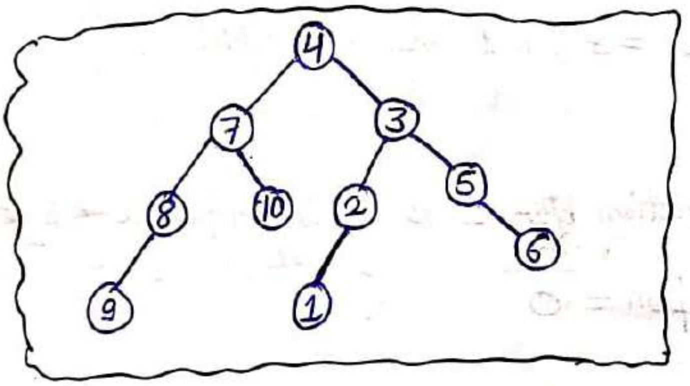
Tree Foundations & Framework

* Trees → a general modification of graph which has no cycle and it is connected.

How trees look like :-

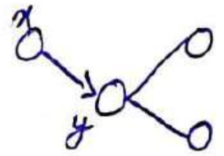


⇓



All concepts like BFS / DFS is applicable to trees. We will mostly use DFS instead of BFS because here in trees, there is no concept of shortest path but there is a concept of unique path.

Given nodes x & y, exploring y's neighbours :




```
visited [y] = 1
for (auto u : g [y])
  if (visited [u] != 1)
    DFS (u)
```

When performing DFS on tree you don't need a visited array. Because there are are no cycles, you can't accidently revisit a node.

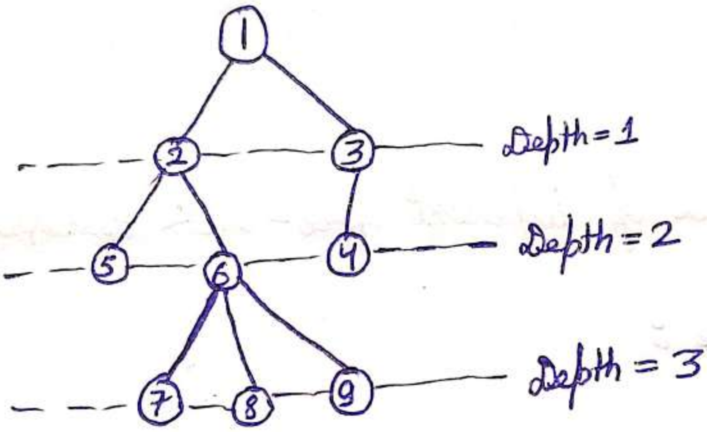
∴ visited [y] = 1

```
for (auto u : g [y])
    if (u != parent [y]) dfs (u)
```

DFS in Tree Code :-

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

int n;
vector<vector<int>> g;

void dfs(int nn, int pp) {
    cout << nn << endl;
    for (auto v : g[nn]) {
        if (v != pp) {
            dfs(v, nn);
        }
    }
}

void solve() {
    cin >> n;
    g.resize(n + 1);
    for (int i = 0; i < n - 1; i++) {
        int a, b;
        cin >> a >> b;
        g[a].push_back(b);
        g[b].push_back(a);
    }
    dfs(1, 0);
}
```

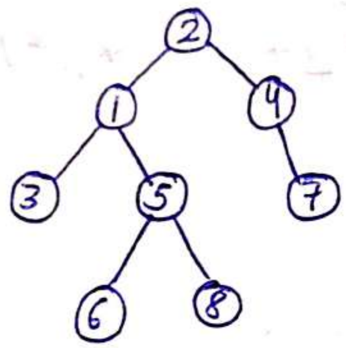
* Find distance to all node from 1.



```cpp
int n;
vector<vector<int>> g;
vector<int> depth;

void dfs(int nn, int pp, int dd) {
    depth[nn] = dd;
    for (auto v : g[nn]) {
        if (v != pp) {
            dfs(v, nn, dd + 1);
        }
    }
}

void solve() {
    cin >> n;
    g.resize(n + 1);
    depth.resize(n + 1);
    for (int i = 0; i < n - 1; i++) {
        int a, b;
        cin >> a >> b;
        g[a].push_back(b);
        g[b].push_back(a);
    }
    dfs(1, 0, 0);
}
```

• <u>Subtree</u> :-

For any particular node, it has child or its subchild.

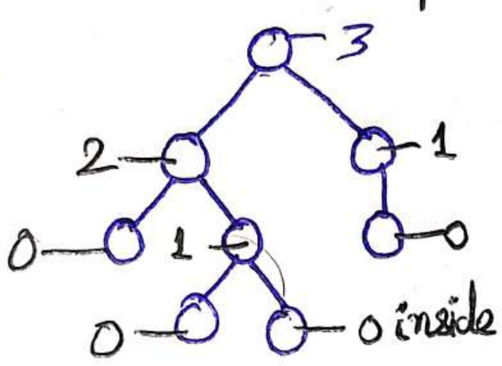


Q: Given a tree. For every node: Calculate no. of nodes in its subtree.

```
dfs(x, pp)
   sub[x]=1
   for(v:g[x])
     if (v!=pp) ~~dfs(v,r~~
       dfs(v,nn)
       sub[x] += sub[v]
```

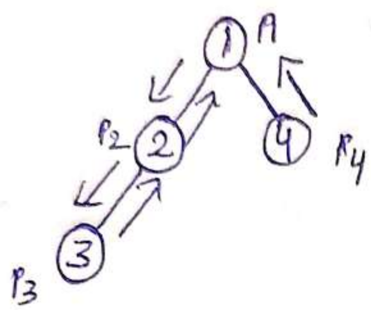
Q: Calculate how far, deep inside there is a node ?

Ex :



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

int n;
vector<vector<int>> g;
// ancestor
vector<int> depth,par;
// subtree
vector<int> subsz,subfar;

void dfs(int nn,int pp,int dd){
    depth[nn] = dd;
    par[nn] = pp;

    subsz[nn] = 1;
    subfar[nn] = 0;

    for(auto v:g[nn]){
        if(v!=pp){
            dfs(v,nn,dd+1);
            subsz[nn] += subsz[v];
            subfar[nn] = max(subfar[nn],1+subfar[v]);
        }
    }
}
```

• Pre Order traversal vs Post



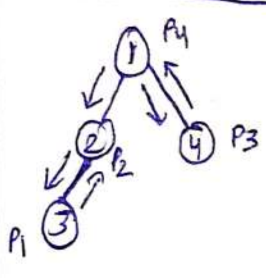
Pre order : 1 2 3 4

OR

Pre order : 1, 4, 2, 3

* Preorder traversal of a tree is not unique.

Post order Traversal :-



P_1 P_2 P_3 P_4

Post order : 3 2 4 1

```cpp
int n;
vector<vector<int>> g;

void dfs(int nn,int pp){
    cout<<nn<<endl;
    for(auto v:g[nn]){
        if(v!=pp){
            dfs(v,nn);
        }
    }
}

void solve(){
    cin>>n;
    g.resize(n+1);
    for(int i=0;i<n-1;i++){
        int a,b;
        cin>>a>>b;
        g[a].push_back(b);
        g[b].push_back(a);
    }
    dfs(1,0);
}
```

```cpp
int n;
vector<vector<int>> g;

void dfs(int nn,int pp){
    for(auto v:g[nn]){
        if(v!=pp){
            dfs(v,nn);
        }
    }
    cout<<nn<<endl;
}

void solve(){
    cin>>n;
    g.resize(n+1);
    for(int i=0;i<n-1;i++){
        int a,b;
        cin>>a>>b;
        g[a].push_back(b);
        g[b].push_back(a);
    }
    dfs(1,0);
}
```

Q: Given a tree, find out the number of different pre-order traversals psbl?

When we are deciding for a node :

a → look at levels

b → in which order you'll visit them

c → order inside subtree.

For c,

\# of ways (child 1) = 1

" " " (child 2) = 2

" " " (child 3) = 3

⋮

Depending on b,

(# child!) × (1 * 2 * 3 - - * k^{th} child ways)

```cpp
int fact[100101];
int n;
vector<vector<int>> g;
vector<int> ways;
void dfs(int nn, int pp) {
    ways[nn] = 1;
    int child = 0;

    for (auto v : g[nn]) {
        if (v != pp) {
            dfs(v, nn);
            ways[nn] *= ways[v];
            child++;
        }
    }
    ways[nn] *= fact[child_count];
}
```

Frameworks :-

1. Foundation (Basic properties)
→ DFS
→ subtree
→ Ancestors

2. Diameter
Nodes with max distance in a tree determines its diameter.

3. Center
Mid point ~~of~~ node of diameter's path. There can be more than one center.

4. Centroid
A node whose all subtrees size is less than half of the whole tree's size.

5. Contribution Technique
Edge

How to find diameter?

Diameter is the longest possible path between any two nodes in the tree.

1. Pick any random node (say node A) and run a DFS to find the node farthest from it. (B).

2. Now, run a second DFS starting from node B to find the node farthest ~~to~~ from it (C).

3. ~~A & B are di~~ The path between B and C is the diameter of the tree.

- More on Frameworks

1. Ancestral Maintanence — Data structures on Ancestors

2. Small to Large Merging / BSU on sack — Data structures on subtrees

3. DS on Paths — Binary lifting, LCA finding

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

int n;
vector<vector<int>> g;
// ancestor
vector<int> depth;

void dfs(int nn,int pp,int dd){
    depth[nn] = dd;
    for(auto v:g[nn]){
        if(v!=pp){
            dfs(v,nn,dd+1);
        }
    }
}

void solve(){
    cin>>n;
    g.resize(n+1);
    depth.resize(n+1);

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

    dfs(1,0,0);

    int x = 1;
    for(int i=1;i<=n;i++){
        if(depth[i]>depth[x])x=i;
    }

    dfs(x,0,0);

    int y = 1;
    for(int i=1;i<=n;i++){
        if(depth[i]>depth[y])y=i;
    }

    cout<<x<<" "<<y<<endl;
```