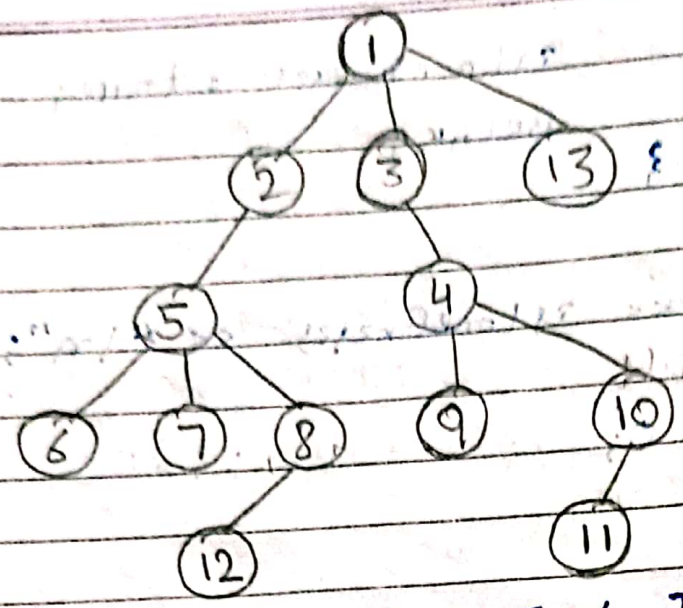
How to DFS in a Tree | Height & Depth

NOTE: Same code of DFS for graphs will also work fine here, but we don't have cycles here in trees, so, we don't need visited array, and to handle child going back on visited parent, we can simply pass parent also in DFS.



	1	2	3	4	5	6	7	8	9	10	11	12	13
Depth (d)	0	1	1	2	2	3	3	3	3	3	4	4	1
Height (h)	4	3	3	2	2	0	0	1	0	1	0	0	0

APPROACH FOR DEPTH (D)

First of all depth of a ~~leaf~~ root node in a tree is 0.

After that we will compute: ~~depth~~

depth [child] = depth [parent] + 1

e.g.: depth [1] = 0 (We know that)

depth [2] = depth [1] = 1 => 1

It is computed while going up to down ~~APPRO~~ in DFS.

APPROACH FOR HEIGHT (H)

First of all height of all the leaf nodes will be 0.

After that we will compute :

height [vertex] = max (height [vertex] , height [child] + 1)

e.g: Let's suppose we have to find height for vertex (5).

∴ → height [5] = max (height [5], height [6] + 1)

height [5] → 0 initially

height [6] + 1 → 0 initially

⇒ height [5] = 1

→ height [5] = max (height [5], height [7] + 1)

height [5] → 1 now

height [7] + 1 → 0 initially

⇒ height [5] = 1

→ height [5] = max (height [5], height [8] + 1)

height [5] → 1 now

height [8] + 1 → 1 initially

∴ height [5] = 2

It is computed while going down to up in DFS.

-> Maximum capacity of nodes

```
CONST INT N = 1e5+10;
```

-> Adjacency list

```
VECTOR<INT> G[N];
```

-> Arrays to store depth and height (already initialized with 0 (as global)).

```
INT DEPTH[N], HEIGHT[N];

VOID DFS (INT VERTEX, INT PAR=0)
{
```

CODE

→ Take action on vertex after entering the vertex.

```
FOR (INT CHILD : G[VERTEX])
{
    → Take action on child before entering the child node
    IF (CHILD == PAR) CONTINUE; // skipping visited node (parent)
    DEPTH[CHILD] = DEPTH[VERTEX] + 1;
    DFS (CHILD, VERTEX);

    → Take action on child after exiting child node.
    HEIGHT[VERTEX] = MAX (HEIGHT[VERTEX], HEIGHT[CHILD] + 1);
}
→ Take action on vertex before exiting the vertex.
}

INT MAIN()
{
    INT n; → Taking no. of nodes as input
    CIN >> n;

    → Creating Adjacency list representation of tree
    FOR (INT i=0; i < n-1; i++)
    {
        INT V1, V2;
        CIN >> V1 >> V2;
```

```
    g[v1].pb(v2);
    g[v2].pb(v1);
}

dfs(1);  -> calling the function

-> Viewing the height and depth array.
cout << "D" << " " << "H" << "\n";
for(int i=1; i<=n; i++)
{
    cout << depth[i] << " " << height[i]
         << "\n";
}

return 0;
}
```

INPUT:

```
13
1 2
1 3
1 13
2 5
3 4
5 6
5 7
5 8
8 12
4 9
4 10
10 11
```

OUTPUT:

```
D H
0 4
1 3
1 3
2 2
2 2
3 0
3 0
3 1
3 0
3 1
4 0
4 0
1 0
```