# Introduction to Trees

Dev Karan Singh (devkaran1231)
Expert at codeforces (1817)
**5 star** at codechef (2040)

# Introduction to Trees

- Basic terminologies in trees
- Properties of trees
- How to store a tree? How will the input be given?
- Traversal techniques in Trees
    - **D**epth **F**irst **S**earch (DFS)
    - **B**readth **F**irst **S**earch (BFS)
- Most Basic Dynamic programming on Trees
    - Question Link
- Given a tree, find the height of the tree.
- Given root node and another node x, print path from root to x if exists.

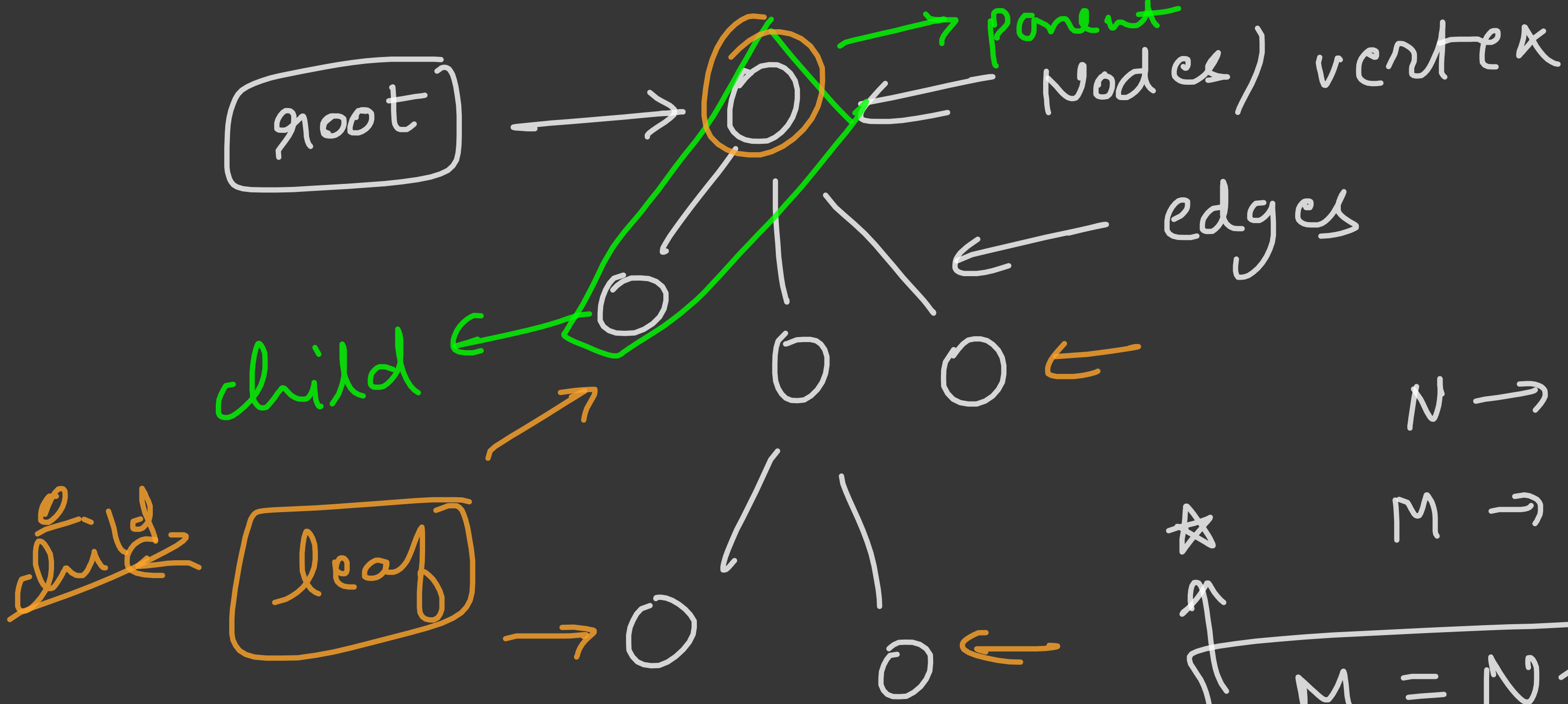
# Trees

Trees → BT/BST → DSA ×

BT/BST → one node can have at max 2 children

Trees → Generic Tree

Generic Tree ↓ Any number of children

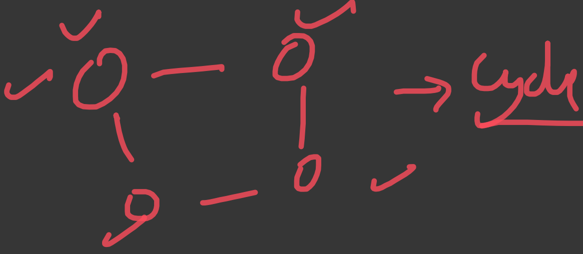


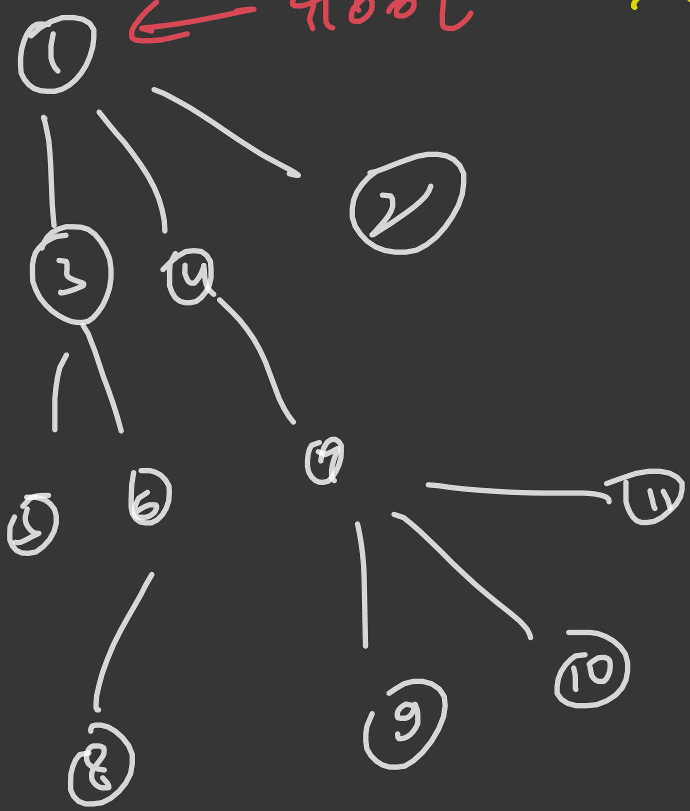
N → nodes

M → edges

☆ $$\boxed{M = N-1}$$

→ In trees we don't have cycles





root → ancestor to every node of the tree

→ subtree of 4

→ Array → store data → access it → traversal

Trees → Traversal → DFS (Depth)

→ BFS (Breadth)

visiting each element of the tree atleast once

Q How to store a tree?
How will the input be given?

No of nodes

↳ First line contains N the follows N-1 lines. Each of the N-1 lines contains u and v which means there is an edge b/w u and v

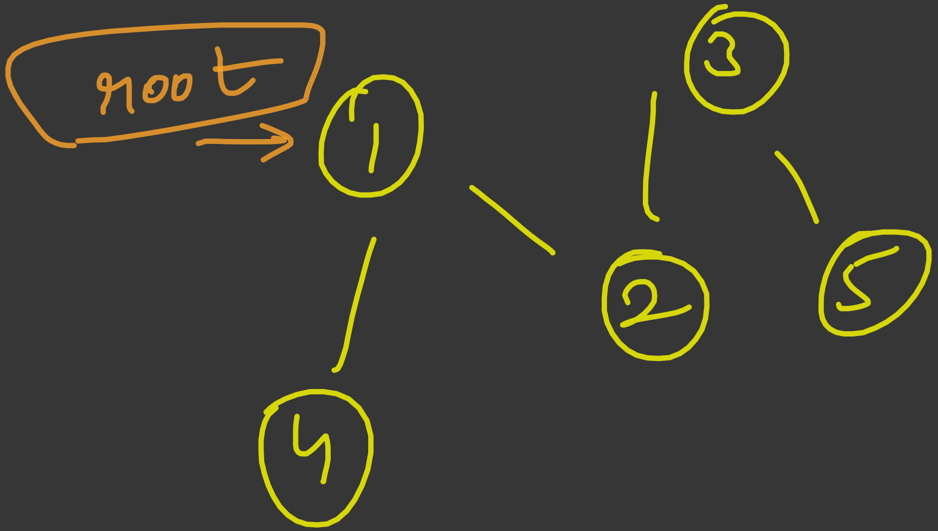
Eg 5

1 2

2 3

1 4

3 5



int arr[n+1]; → array of integers of size n+1 vector<int> adj[n+1],

→ array of vectors of size n+1

1 based indexing

$\boxed{n=5}$

0 → [ ] ✓

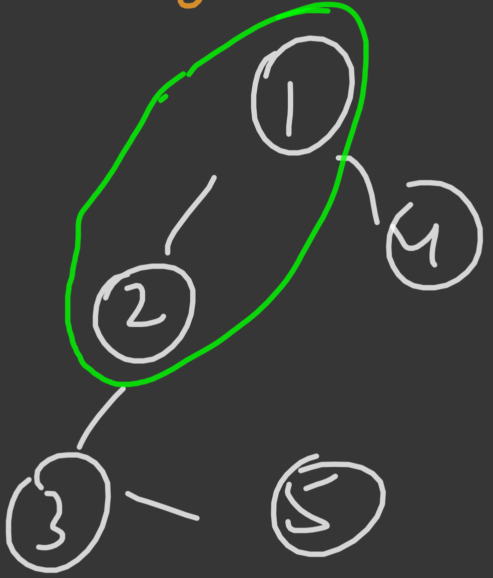
1 → [2, 4]

2 → [1, 3]

3 → [2, 5]

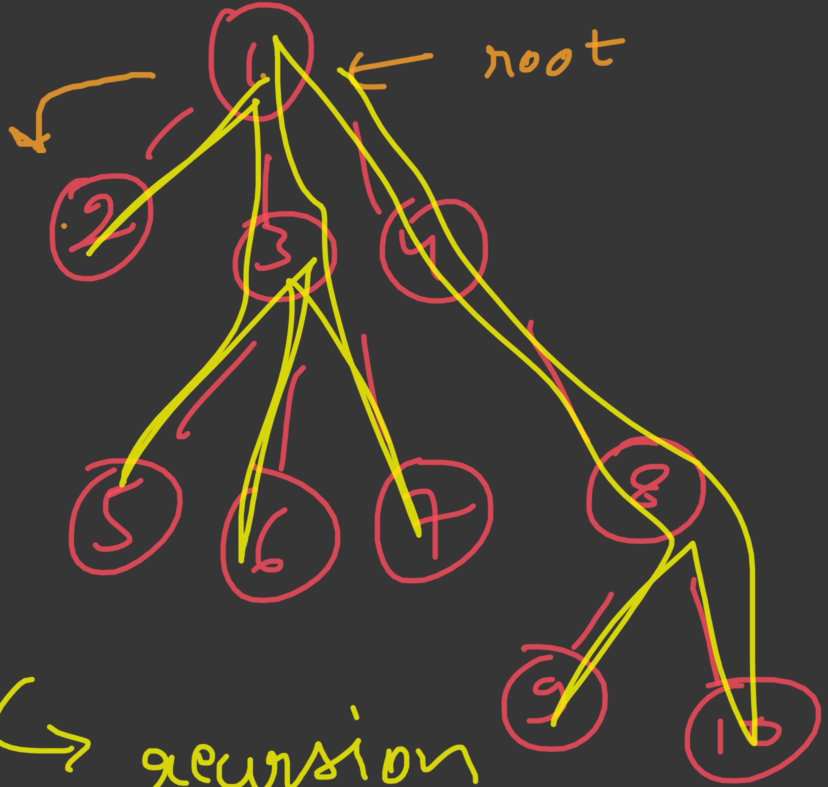
4 → [1]

5 → [3]



```
int n,
cin>>n;
for(int i=0, i<n-1, i++){
    int u,v,
    cin >>u>>v;
    adj[u].pb(v),
    adj[v].pb(u),
}
```

# # DFS → depth first search



↳ recursion

1 → 2, 3, 4

2 → X

3 → ~~1~~, 5, 6, 7

4 → ~~1~~, 8

5 → ~~3~~

6 → ~~3~~

7 → ~~3~~

8 → ~~4~~, 9, 10

9 → ~~8~~

10 → ~~8~~

1 2 3 5 6 7 4 8 9 10

valid DFS

```
void   dfs(int node, int par){
// this is the place where
     I just entered node
    cout << node << " ";
for (auto it : adj[node]){
      if(it == par) continue
      dfs(it, node);
 }

// We are about to leave
                       this node
```

dfs in subtree of it

}

```
dfs ( 1, 0 );
```

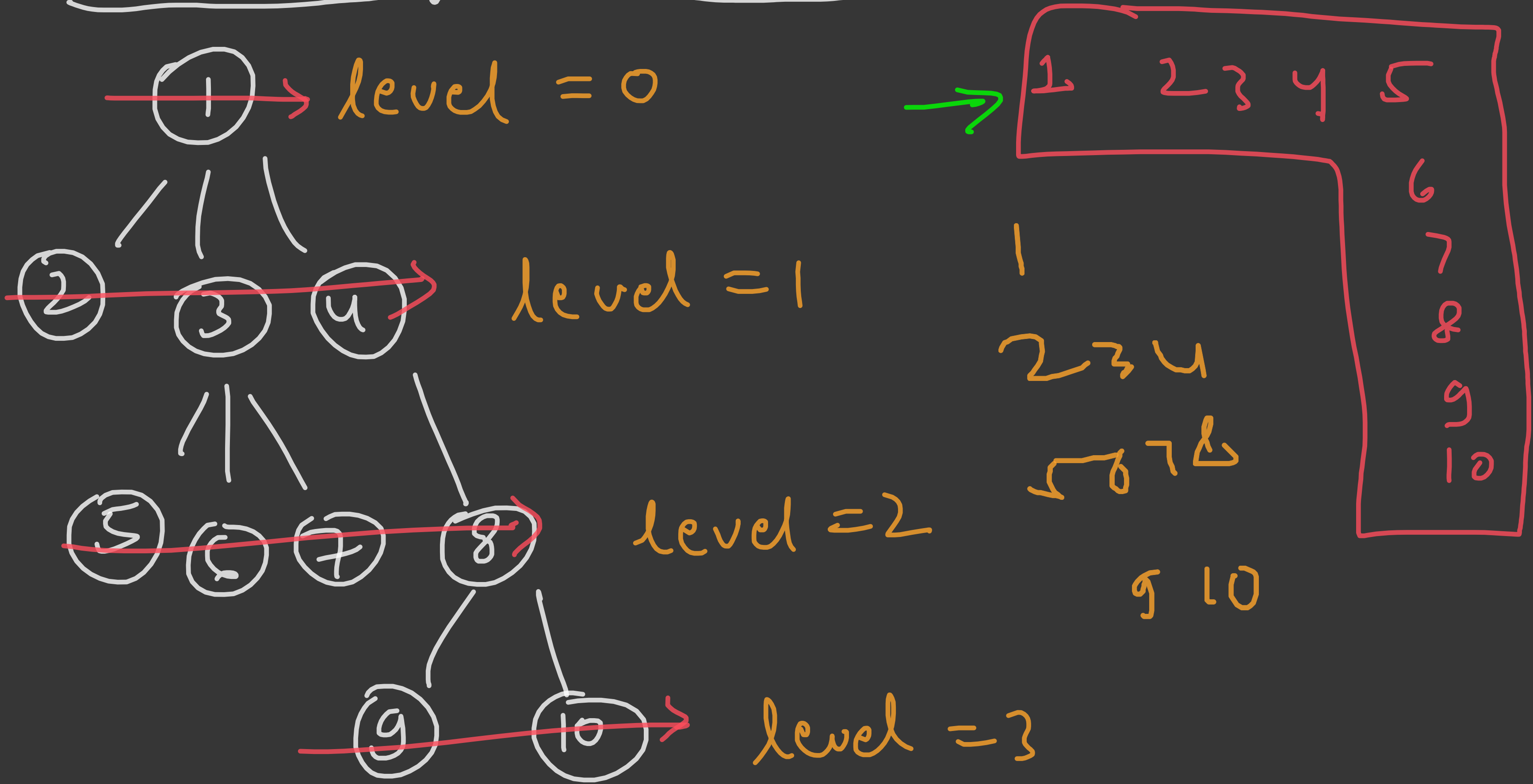
TC → $O(e+v) \rightarrow O(v-1+v)$

$\hookrightarrow O(2v)$

SC → $O(N)$

↳ when? → skew tree

# Breadth first search →



1

2 3 4

5 6 7 8

9 10

1

2

3

4

5

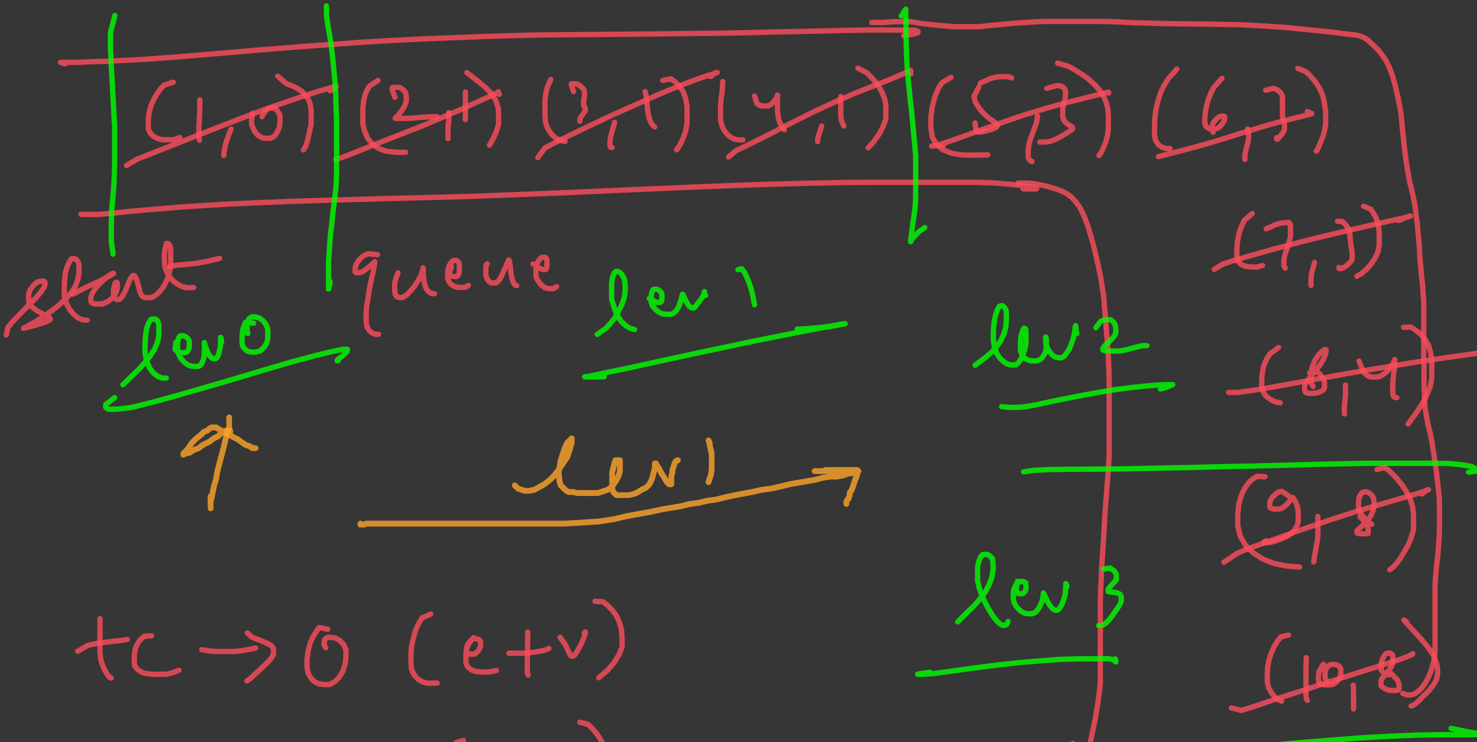
6

7

8

9

10

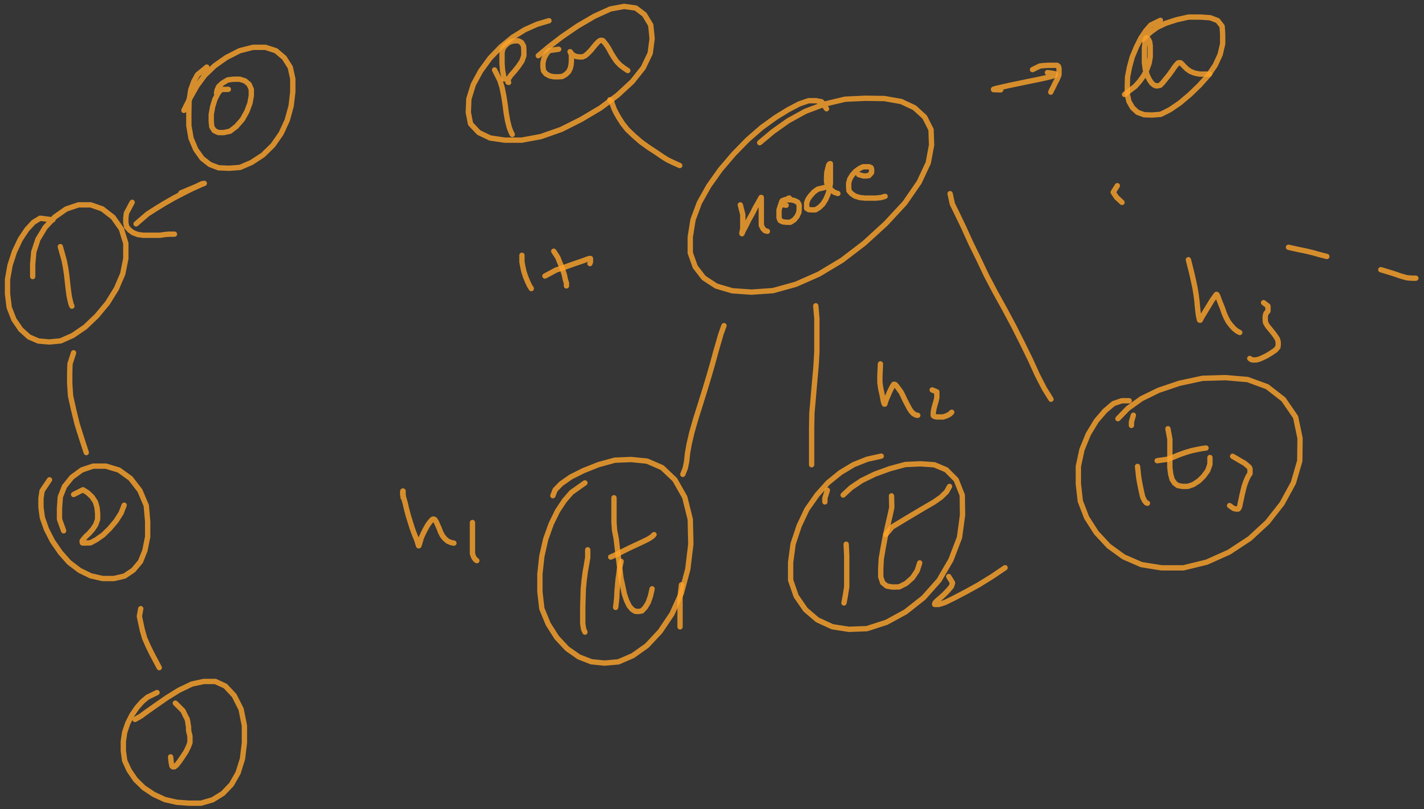


tc → O (e+v)

→ O (2N)

sc → O(N)

1
100
2
3
end
worst case

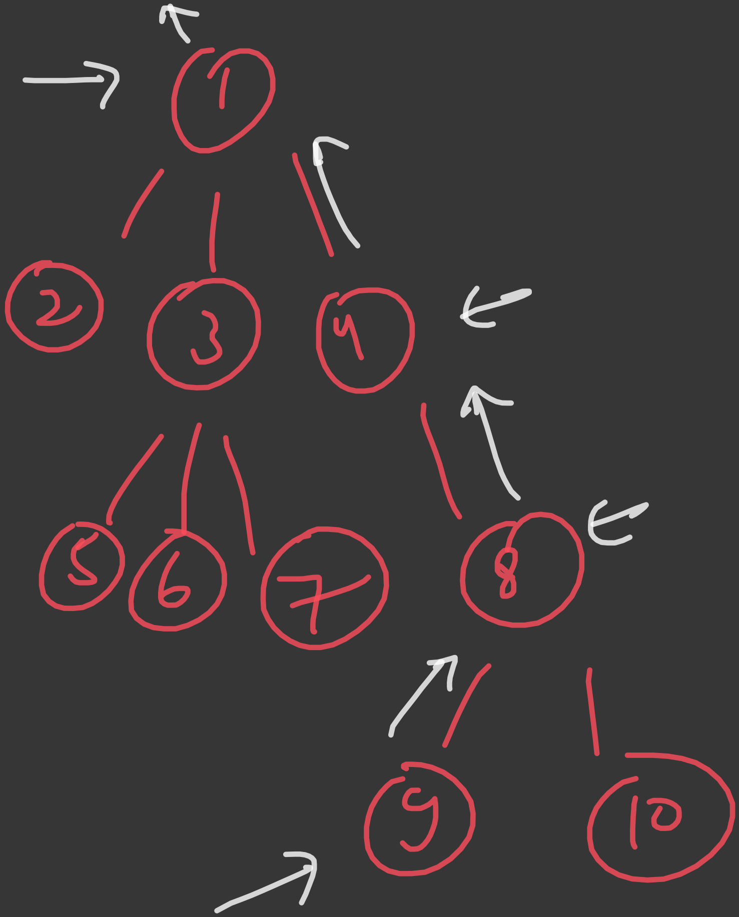

0
1
2
j
par
node
h
1+
h1
it
it2
h2
it3
h3

bool dfs(node) → true → x has been found in subtree of node

→ false

↓

not found in subtree



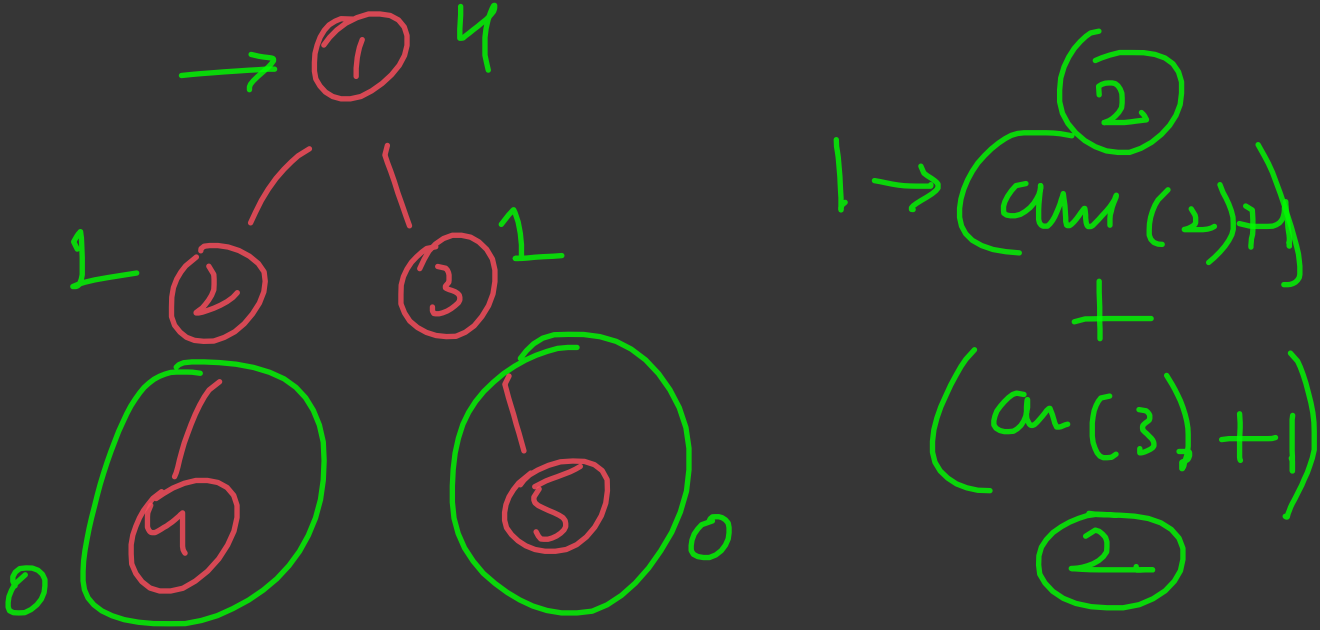
path → 1 4 8 9

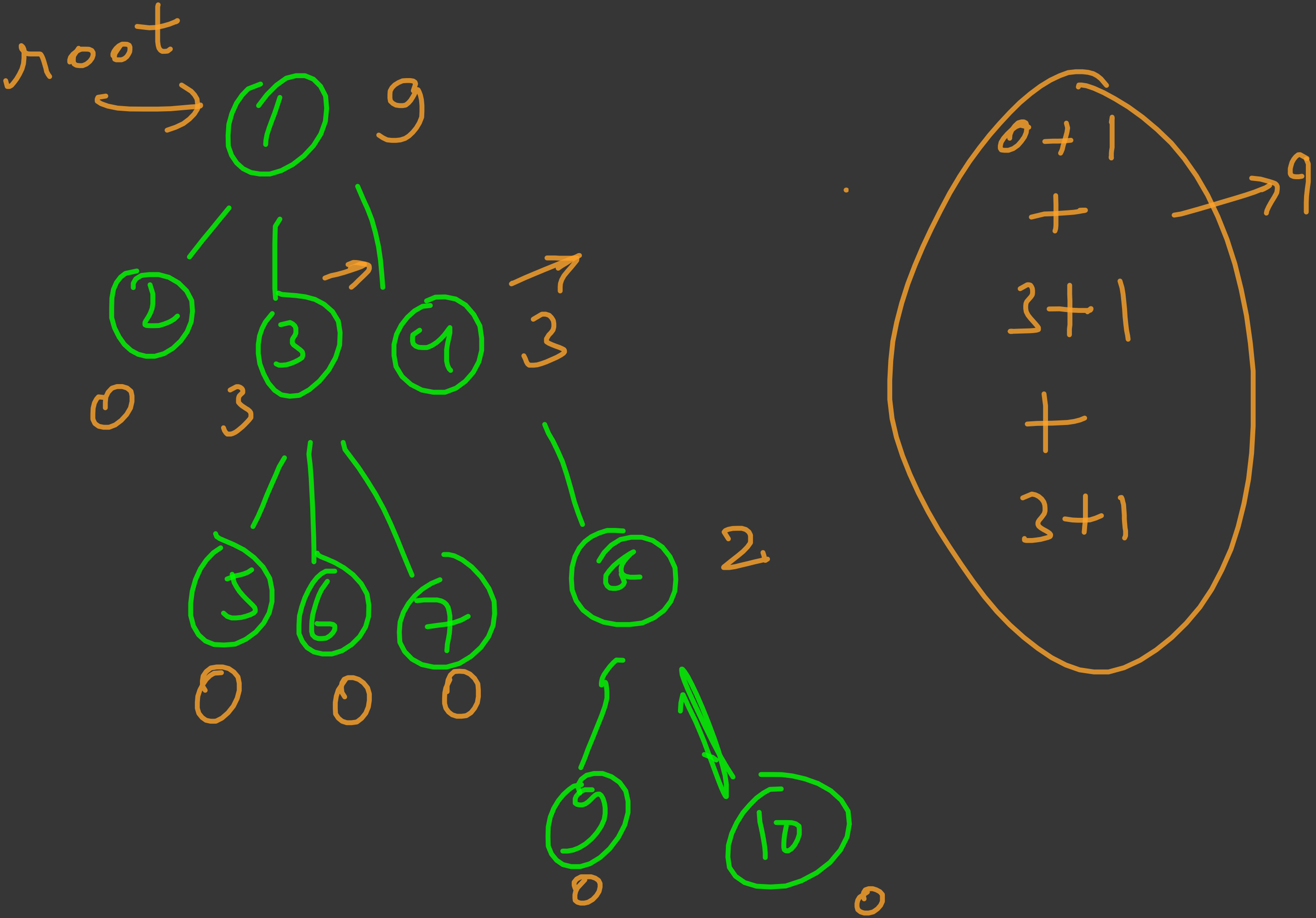
# Subordinates

```
5
1 1 2 3
```

→

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


4
1
1
2
3
1
1
4
5
0
0
2
1 → (ans(2)+1)
+
(an(3)+1)
2


root
1
9
2
0
3
3
4
3
5
0
6
0
7
0
8
2
9
0
10
0
0+1
+
3+1
+
3+1
9