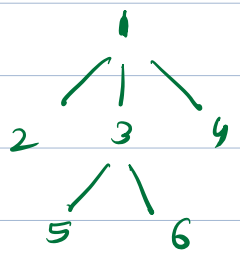
Given a tree, Q queries are asked :

Each query have 2 int x & y.

If y belongs to subtree of x or not



Q = 7

3 5 → Yes

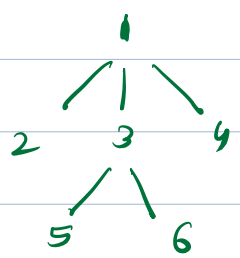
5 6 → NO

2 6 → NO

1 6 → Yes

1 2 → Yes

4 5 → NO

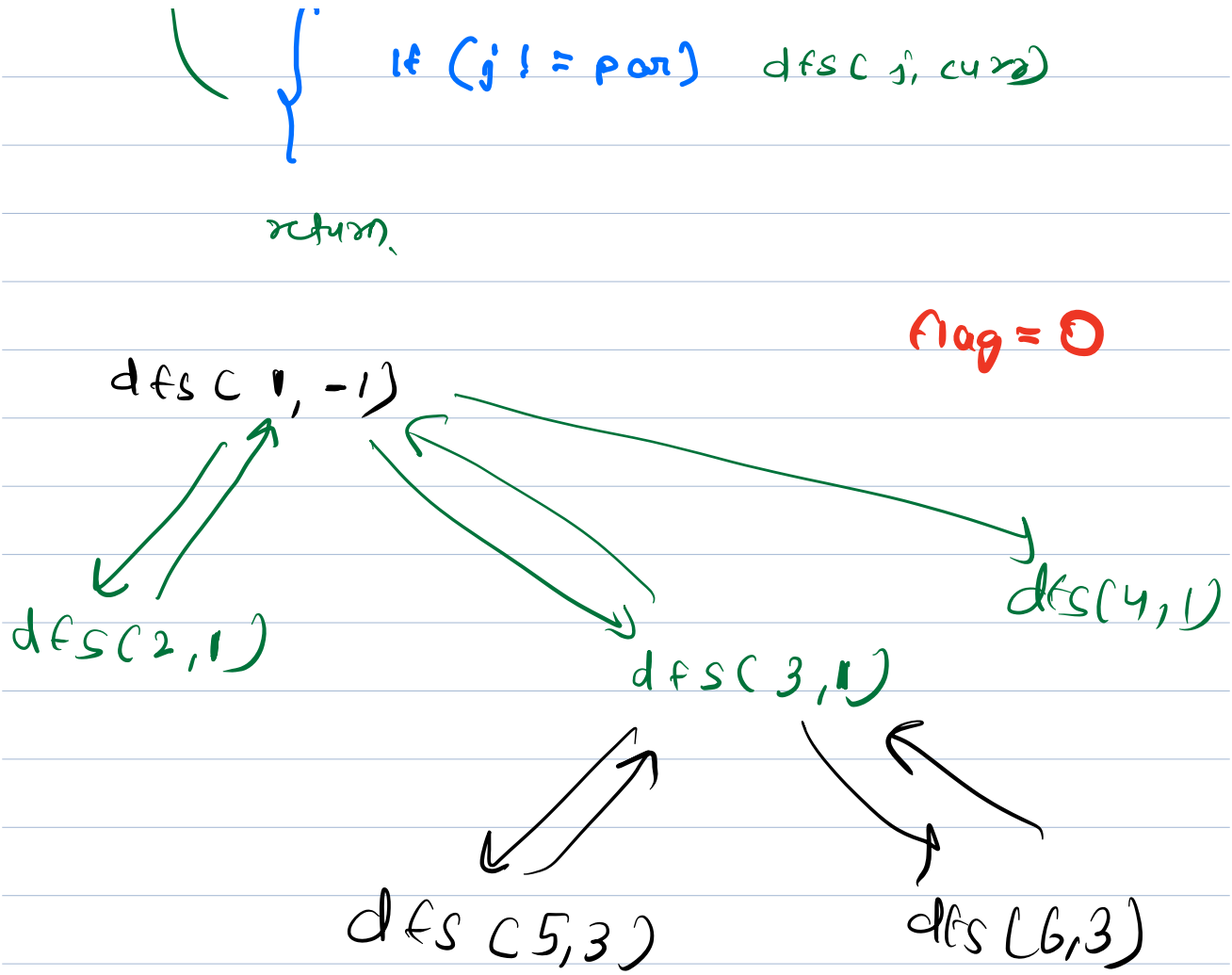


y

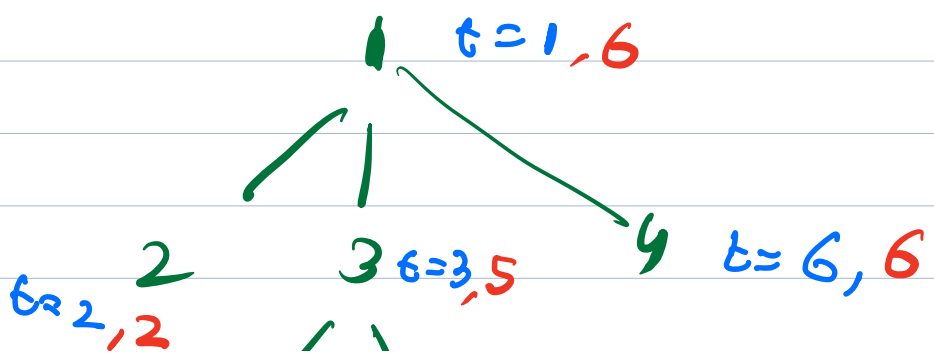
3 6 True

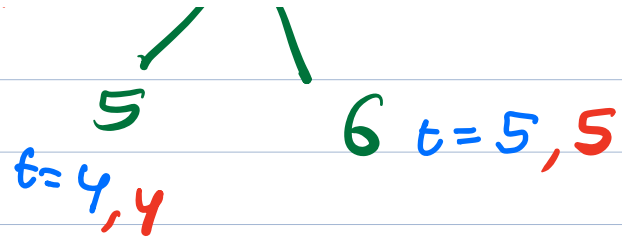
3 4 False

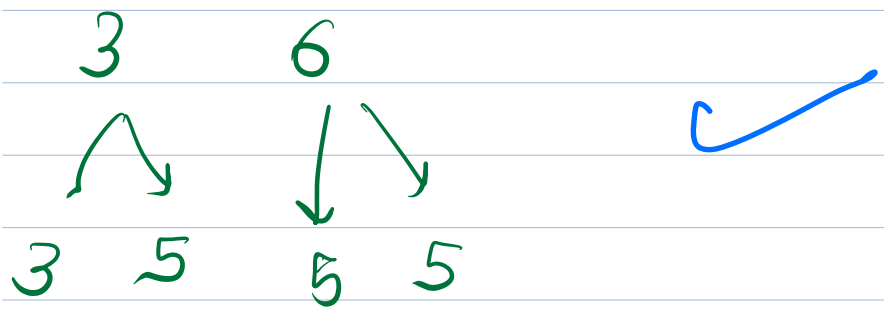
```
void dfs ( int curr, par)
{
    // curr is visiting right now

    for( auto j: adj[ curr]
```

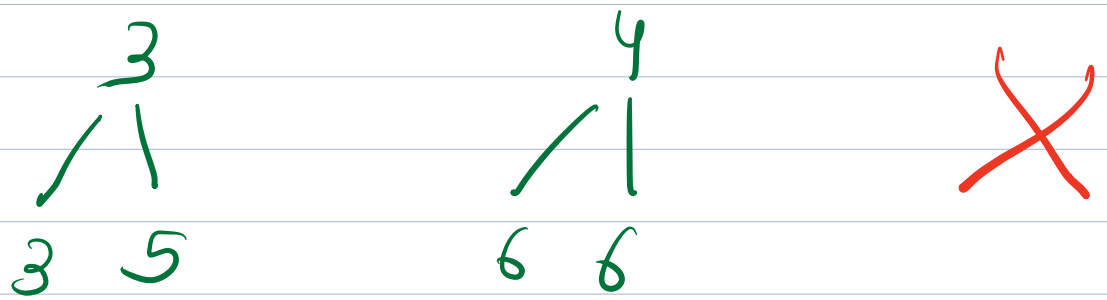

if (j != par) dfs(j, cur)
return
flag = 0
dfs(1, -1)
dfs(2,1)
dfs(4,1)
dfs(3,1)
dfs(5,3)
dfs(6,3)

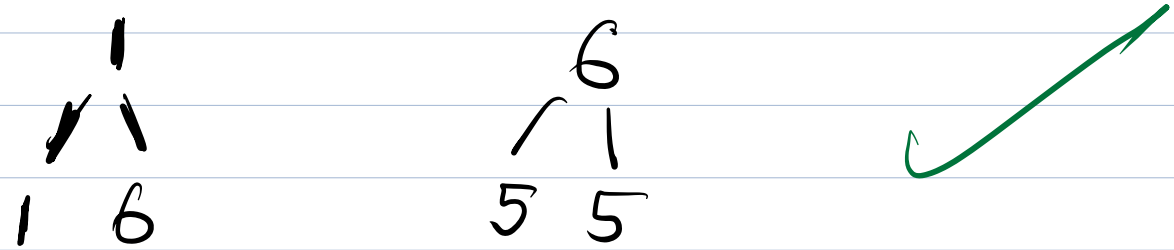
t = 0


t=1,6
2
t=2,2
3 t=3,5
4
t=6,6









Once start time & end time are recorded

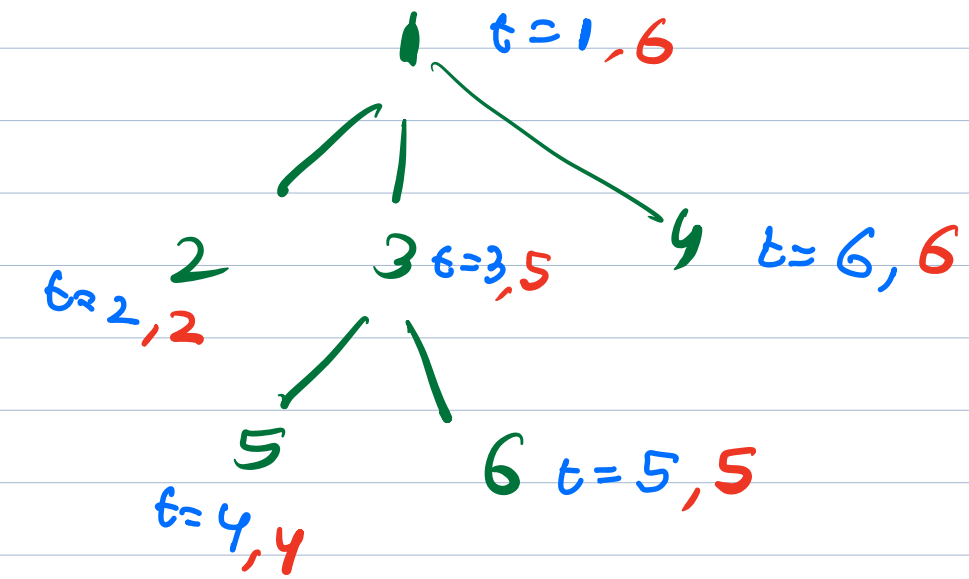
Each query can be answered in O(1) time.

```
int timer
void dfs ( int curr, int par, intime[], outtime[])
{
    timer++
    intime [curr] = timer
    for ( auto j : adj [curr]
    {
        if ( j != par)  dfs(j, curr, intime, outtime)
    }
    outime [curr] = timer
}
```

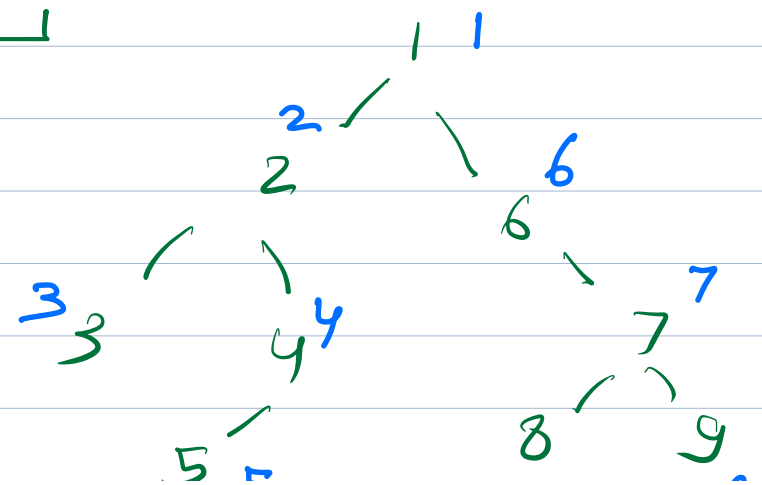
```
timer = 0
int intime [N], out time [N]
dfs ( 1, -1, intime, outime)
for ( i = 0 ;  i < q  ; i++)
{
    x  y .
    if ( intime[y] ≥ intime [x] && outime[y] ≤ outime[x])
        cout << "yes" ;
    else
        cout << "No" ;
}
```

TC : O(V+E+Q)

SC : O(V)



1, 2, 3, 5, 6, 4



8 9

1, 2, 3, 4, 5, 6, 7, 8, 9

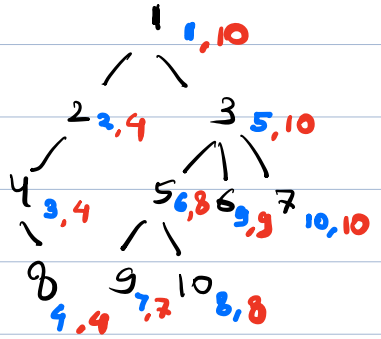
Q. Given a tree A nodes numbered from 1 to A, root at 1. Initially every one is 1.

Q queries x

Type1 All nodes in subtree of x, make them 1

Type2 All nodes in subtree of x, make them 0.

Type3 Count no of nodes in subtree of x, which are one.



1 2 3 4 5 6 7 8 9 10

1, 2, 4, 8, 3, 5, 9, 10, 6, 7

2 5

3 3 ⇒ 3

3 5 ⇒ 0

2 6

1 1

3 9 ⇒ 5

2 4

3 1 ⇒ 8

update curr subtree

update (intime[curr] + 1, outime[curr])

→ Tree flattening
→ segment tree
→ lazy propagation

```
int timer

void dfs ( int curr, int par, intime[], outtime[])
{
    timer++
    intime [curr] = timer
    for ( auto j : adj [curr]
    {
        if ( j != par)   dfs(j, curr, intime, outime)
    }
    outime [curr] = timer .
}

void Build ( idx, start, end)
{
    if ( start == end)
    {
        segtree[idx] = 0
        return
    }
    mid = (start + end) /2
    Build ( 2*idx +1, start, mid)
    Build ( 2 idx + 2, mid+1, end)
    segtree [ idx] = segtree[2idx+1] + segtree[ 2idx+2]
                                                + 2
}

// Update
```

// Query.

```
timer = 0
int intime[N], outtime[N]
dfs(1, -1, intime, outtime)
Build(0, 0, N-1)
for(i = 0; i < q; i++)
{
    if(type1)
    {
        update(0, 0, N-1, intime[x]+1, outtime[x], 1)
    }
    else if(type2)
    {
        update(0, 0, N-1, intime[x]+1, outtime[x], 0)
    }
    else
    {
        print(query(0, 0, N-1, intime[x]+1, outtime[x]))
    }
}
```

TC: $O(V+E + Q \log V + V) : O(V+E+Q\log V)$

Sc : O(V + V + 4V) ⇒ O(V)