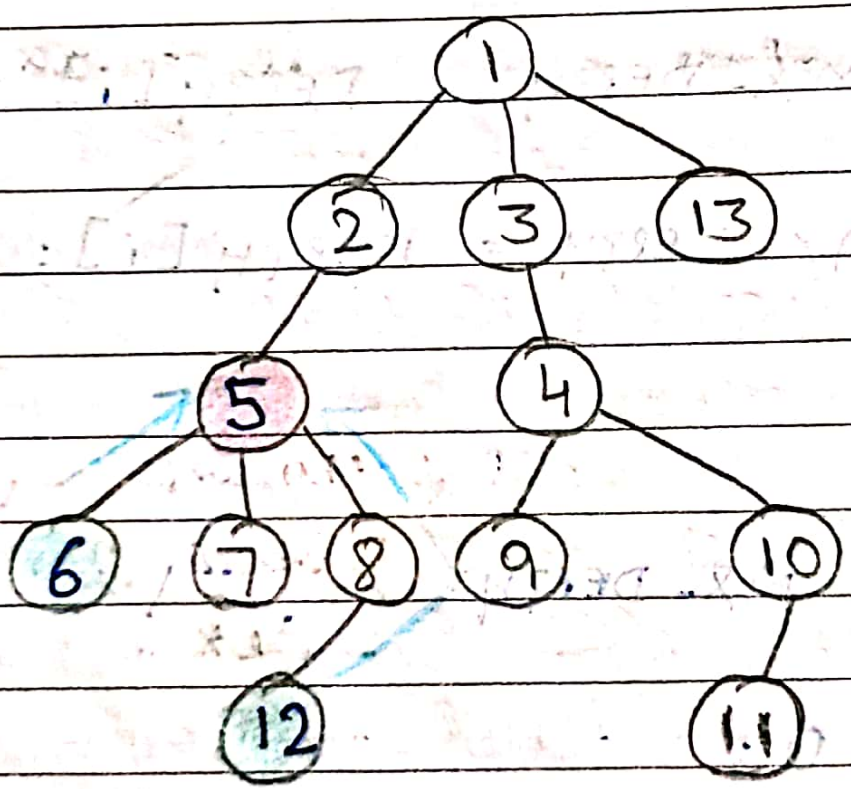
LONEST COMMON ANCESTOR (LCA) IN A TREE

LCA → Lowest common parent for two child nodes.



e.g: LCA for (6) & (12) => (5)
" " (6) & (8) => (5)
" " (7) & (11) => (1)
" " (6) & (5) => (5)

APPROACH: T.C: O(n)

First we will find the path of both the given children.

e.g: 6 → 1, 2, 5, 6
12 → 1, 2, 5, 8, 12

And then we will find the last common element, which is 5 here.

CODE

```
CONST INT N = 1e5+10;
VECTOR<INT> G[N];
-> To store parent of each node.
INT PAR[N];

VOID DFS (INT VERTEX, INT P=-1)
{
    PAR[VERTEX] = P; // storing parent as value and vertex as index.
    FOR (INT CHILD : G[VERTEX])
    {
```

```
            IF (CHILD == P) CONTINUE;
            DFS (CHILD, VERTEX);
        }
}
```

→ Returning path vector

```
VECTOR <INT> PATH (INT V)
{
    VECTOR <INT> ANS;
    WHILE (V != -1)
    {
        ANS.PB (V);
        V = PAR [V];
    }
    REVERSE (ANS.BEGIN(), ANS.END());

    RETURN ANS;
}

INT MAIN()
{
    INT n;
    CIN >> n;

    FOR (INT i=0; i<n-1; i++)
    {
        INT V1, V2;
        CIN >> V1 >> V2;
        g[V1].PB (V2);
        g[V2].PB (V1);
    }
```

→ calling DFS function to store all the parents

```
DFS(1);

INT x, y; // children to find LCA
CIN >> x >> y;
VECTOR<INT> PATH_x = PATH(x);
// calculating path of x (child)
VECTOR<INT> PATH_y = PATH(y);
// calculating path of y (child)
```

→ calculating children with lower path (lower array size of path)

```
INT MN_LN = MIN(PATH_x.SIZE(),
                PATH_y.SIZE());

INT LCA = -1;
→ calculating LCA
FOR(INT i=0; i<= MN_LN; i++)
{
    IF (PATH_x[i] == PATH_y[i]) LCA=
                                PATH_x[i];
    ELSE BREAK;
}

COUT << LCA << "\n";

RETURN 0;
}
```

Page No.
Date

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
6 12

OUTPUT:

5