Lowest Common Ancestor

source: HackerRank,

Naive Approach $O(N)$:

```
#define mx_nodes 100
bool visited[mx_nodes];
int parent[mx_nodes];
vector<int> tree[mx_nodes];
int root_node = 0;
/*GetParents() function traverses the tree and computes the parent array such that
The pre-order traversal begins by calling GetParents(root_node,-1) */
void GetParents(int node , int par){
    for(int i = 0 ; i < (int) tree[node].size() ; ++i){
        /*As this is a pre-order traversal of the tree the parent of the current    node has already been
        if(tree[node][i] != par){
            parent[tree[node][i]] = node ;
            GetParents(tree[node][i] , node) ;
        }
    }
}
/*Computes the LCA of nodes u and v . */
int LCA(int u , int v){
    GetParents(root_node, -1);
    /*traverse from node u uptil root node and mark the vertices encountered along the path */
    int lca ;
    while(1){
        visited[u] = true;
        if(u == root_node) break;
        u = parent[u];
    }
    /*Now traverse from node v and keep going up untill we dont hit a node that is in the path of node u t
    while(1){
        if(visited[v]){
            /*Intersection of paths found at this node.*/
            lca = v;    break ;
        }
        v = parent[v] ;
    }
    return lca ;
}
int main() {
    int n; cin >> n;
    for (int i = 0, x, y; i < n-1; ++i) { cin >> x >> y; tree[x].push_back(y); }
    cout << LCA(4, 6) << endl;
}
```

Now the parents of all the nodes can be computed without calling the `GetParents(root_node, -1)` function simply like this, if there's no extra condition:

```
  for (int i = 0, x, y; i < n-1; ++i) {
      cin >> x >> y;
      tree[x].push_back(y);
      parent[y] = x;
  }
```

Time Complexity: $O(h)$ where h is the maximum distance of the root from a leaf. In case the tree is very **unbalanced** such that every node of the tree has exactly one child, the structure of the tree becomes linear and hence $h = N$ where N is the number of nodes in the tree, making the algorithm $O(N)$.