```cpp
#include <bits/stdc++.h>
using namespace std;
#define ll long long

#define left p<<1 , l , (l+r)>>1
#define right p<<1|1 , ((l+r)>>1)+1 , r
```

LCA by segment tree complexity O(log(n)):

```cpp
struct LCA{
    vector<int> first , occur , depth , seg;
    vector<vector<int>> adj;

    LCA(vector<vector<int>> adj , int n , int root){
        this->adj = adj;
        depth.assign(n+1 , 0);
        first.assign(n+1 , 0);
        occur.reserve(n*2);
        dfs(root , 0);
        seg.assign((occur.size()*4)+100 , 0);
        build(1 , 0 , occur.size()-1);
    }

    void dfs(int u , int parent , int d = 0){
        depth[u] = d;
        first[u] = occur.size();
        occur.push_back(u);
        for(auto v : adj[u])
            if(v≠parent){
                dfs(v , u , d+1);
                occur.push_back(u);
            }
    }

    int build(int p , int l , int r){
        if(l==r) return seg[p] = occur[l];
        int c1 = build(left) , c2 = build(right);
        if(depth[c1] < depth[c2]) seg[p] = c1; else seg[p] = c2;
        return seg[p];
    }

    int query(int u , int v){
        int l = first[u] , r = first[v];
        if(l > r) swap(l , r);
        return query(l , r , 1 , 0 , occur.size()-1);
    }

    int query (int i , int j , int p , int l , int r){
        if(j<l  ||  r<i) return -1;
        if(i⩽l && r⩽j) return seg[p];
        int c1 = query(i , j , left) , c2 = query(i , j , right);
        if(c1==-1) return c2; if(c2==-1) return c1;
        return depth[c1] < depth[c2] ? c1 : c2;
    }
};
```