Pastebin: <http://paste.ubuntu.com/24563569/>

#define NODES 100105  
#define LOG\_NODES 20  
#define callLeft s,mid,nd+nd  
#define callRight mid+1,e,nd+nd+1  
**using** **namespace** std;  
**int** n;  
**int** par[NODES], p[NODES][LOG\_NODES];  
**int** depth[NODES];  
**int** subsize[NODES];  
vector <**int**> adj[NODES];  
  
**int** chainNo=0, sz=0;  
**int** chainHead[NODES];  
**int** baseArray[NODES];  
**int** posInBase[NODES];  
**int** chainInd[NODES];  
**int** segTree[4\*NODES];  
  
**void** dfs(**int** u){  
 subsize[u]=1;  
 **for**(**int** i=0; i<adj[u].size(); i++){  
 **int** v=adj[u][i];  
 **if**(v!=par[u]){  
 depth[v]=1+depth[u];  
 par[v]=u;  
 dfs(v);  
 subsize[u]+=subsize[v];  
 }  
 }  
}  
  
**void** lca\_precal(){  
 **for**(**int** i=1; i<=n; i++){  
 p[i][0]=par[i];  
 }  
 **for**(**int** j=1; (1<<j)<n; j++){  
 **for**(**int** i=1; i<=n; i++){  
 **if**(p[i][j-1]!=-1){  
 p[i][j]=p[p[i][j-1]][j-1];  
 }  
 }  
 }  
}  
  
**void** lca\_init(**int** root){  
 memset(par, -1, **sizeof**(par));  
 memset(p, -1, **sizeof**(p));  
 depth[root]=0;  
 dfs(root);  
 lca\_precal();  
}  
  
**int** lca\_query(**int** u, **int** v){  
 **int** lg;  
 **if**(depth[u]<depth[v]){  
 swap(u, v);  
 }  
 **for**(lg=1; (1<<lg)<=depth[u]; lg++);  
 lg--;  
 **for**(**int** i=lg; i>=0; i--){  
 **if**(depth[u]-(1<<i)>=depth[v]){  
 u=p[u][i];  
 }  
 }  
 **if**(u==v) **return** u;  
 **for**(**int** i=lg; i>=0; i--){  
 **if**(p[u][i]!=-1 && p[u][i]!=p[v][i]){  
 u=p[u][i];  
 v=p[v][i];  
 }  
 }  
 **return** par[u];  
}  
  
**void** build(){  
 memset(segTree, 0, **sizeof**(segTree));  
}  
  
**void** update(**int** s, **int** e, **int** nd, **int** pos, **int** val){  
 **if**(s>pos || e<pos) **return**;  
 **if**(s==e){  
 baseArray[s]+=val;  
 segTree[nd]+=val;  
 **return**;  
 }  
 **int** mid=(s+e)>>1;  
 update(callLeft, pos, val);  
 update(callRight, pos, val);  
 segTree[nd]=max(segTree[nd+nd], segTree[nd+nd+1]);  
}  
  
**int** query(**int** s, **int** e, **int** nd, **int** l, **int** r){  
 **if**(s>r || e<l) **return** 0;  
 **if**(s>=l && e<=r) **return** segTree[nd];  
 **int** mid=(s+e)>>1;  
 **return** max(query(callLeft, l, r), query(callRight, l, r));  
}  
  
**void** HLD(**int** node){  
 **if**(chainHead[chainNo]==-1){  
 chainHead[chainNo]=node;  
 }  
 chainInd[node]=chainNo;  
 posInBase[node]=sz;  
 baseArray[sz]=0;  
 sz++;  
 **int** sc=-1;  
 **for**(**int** i=0; i<adj[node].size(); i++){  
 **if**(adj[node][i]!=par[node] && subsize[adj[node][i]]>subsize[sc]){  
 sc=adj[node][i];  
 }  
 }  
 **if**(sc!=-1){  
 HLD(sc);  
 }  
 **for**(**int** i=0; i<adj[node].size(); i++){  
 **if**(adj[node][i]!=par[node] && sc!=adj[node][i]){  
 chainNo++;  
 HLD(adj[node][i]);  
 }  
 }  
}  
  
**int** query\_up(**int** u, **int** v){  
 assert(depth[u]>=depth[v]);  
 **int** uchain, vchain=chainInd[v], ans=-1;  
 **while**(1){  
 uchain=chainInd[u];  
 **if**(uchain==vchain){  
 ans=max(ans, query(0, sz-1, 1, posInBase[v], posInBase[u]));  
 **break**;  
 }  
 ans=max(ans, query(0, sz-1, 1, posInBase[chainHead[uchain]], posInBase[u]));  
 u=par[chainHead[uchain]];  
 }  
 **return** ans;  
}  
  
**void** HLD\_init(**int** root){  
 memset(chainHead, -1, **sizeof**(chainHead));  
 lca\_init(root);  
 HLD(root);  
 build();  
}  
  
**int** HLD\_query(**int** u, **int** v){  
 **int** lca=lca\_query(u, v);  
 **return** max(query\_up(u, lca), query\_up(v, lca));  
}  
  
**void** HLD\_update(**int** u, **int** val){  
 update(0, sz-1, 1, posInBase[u], val);  
}