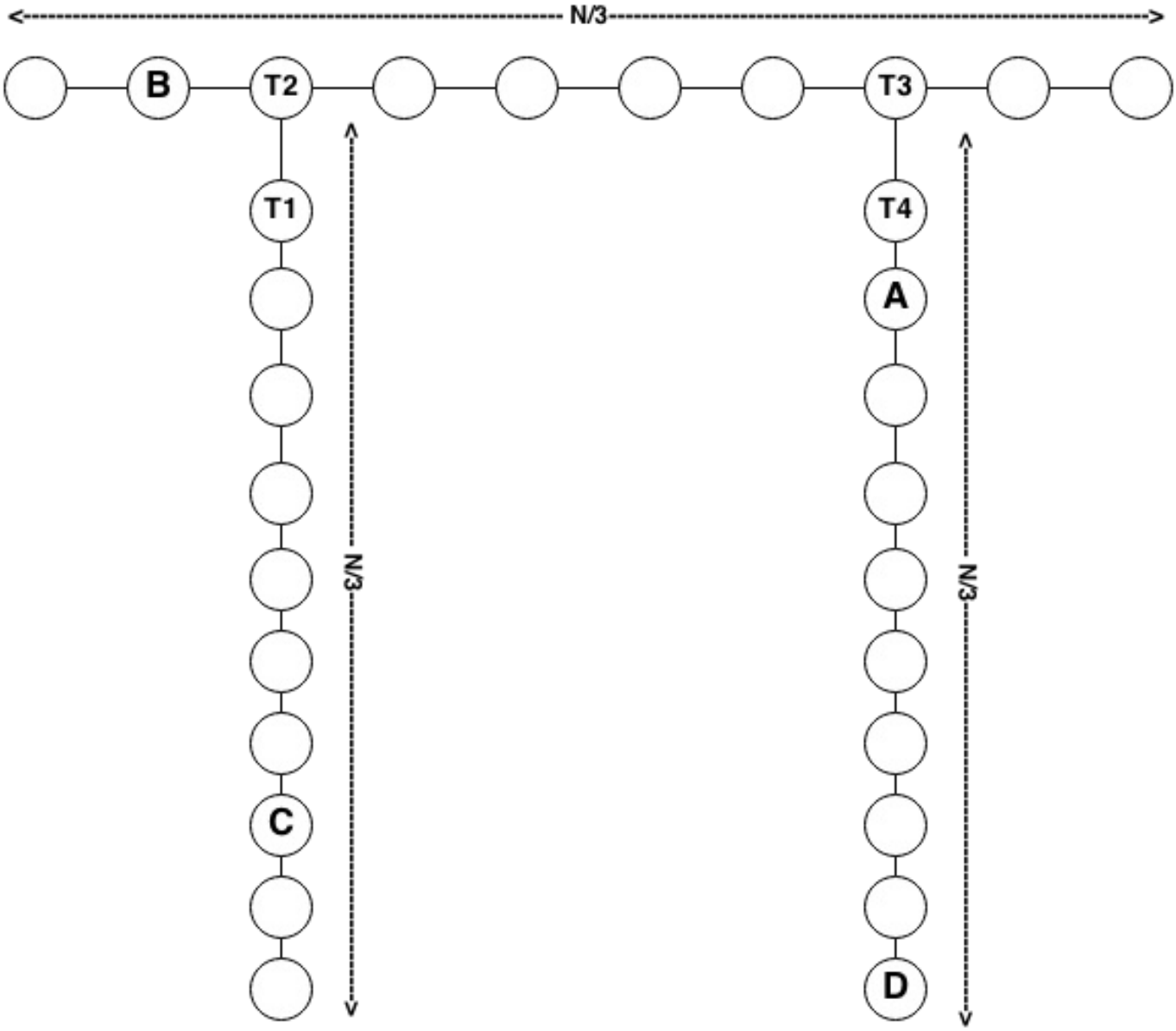
Heavy Light Decomposition

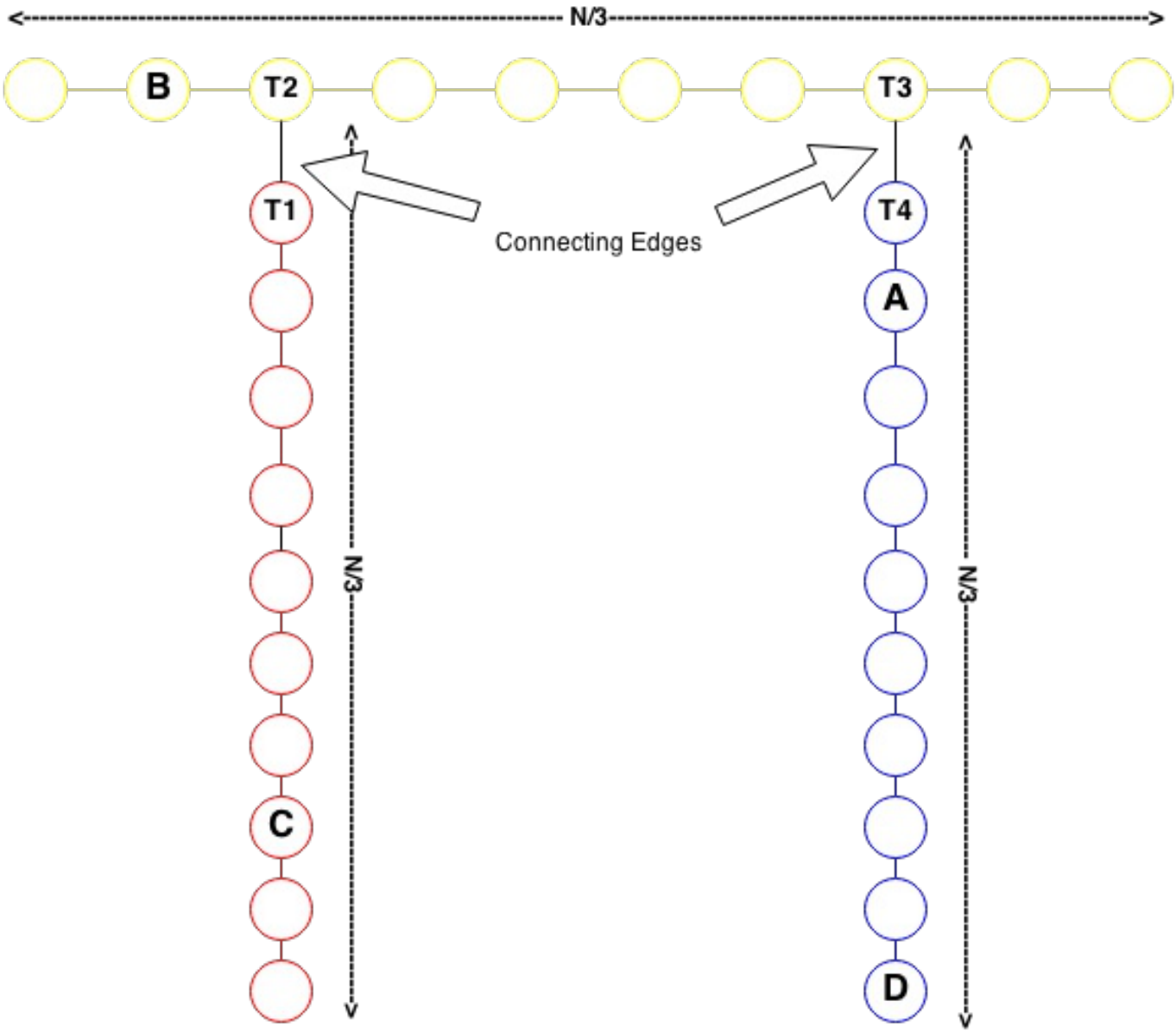
- **Balanced Binary Tree** is good, need to visit at most **2log N** nodes to reach from any node to any other node in the tree
- If a balanced binary tree with N nodes is given, then many queries can be done with O(log N) complexity. **Distance of a path, Maximum/Minimum in a path, Maximum contiguous sum** etc etc.
- **Chains** are also good. We can do many operations on array of elements with O(log N) complexity using **segment tree / BIT**.
- **Unbalanced Tree** is bad

Unbalanced Trees



Travelling from one node to other requires **O(n)** time.

What if we broke the tree in to 3 chains?



Basic Idea

We will divide the tree into **vertex-disjoint chains** (Meaning no two chains has a node in common) in such a way that to move from any node in the tree to the root node, we will have to change at most **log N chains**. To put it in another words, the **path** from **any node** to **root** can be broken into pieces such that the each piece belongs to only one chain, then we will have **no more than log N pieces**.

So what?

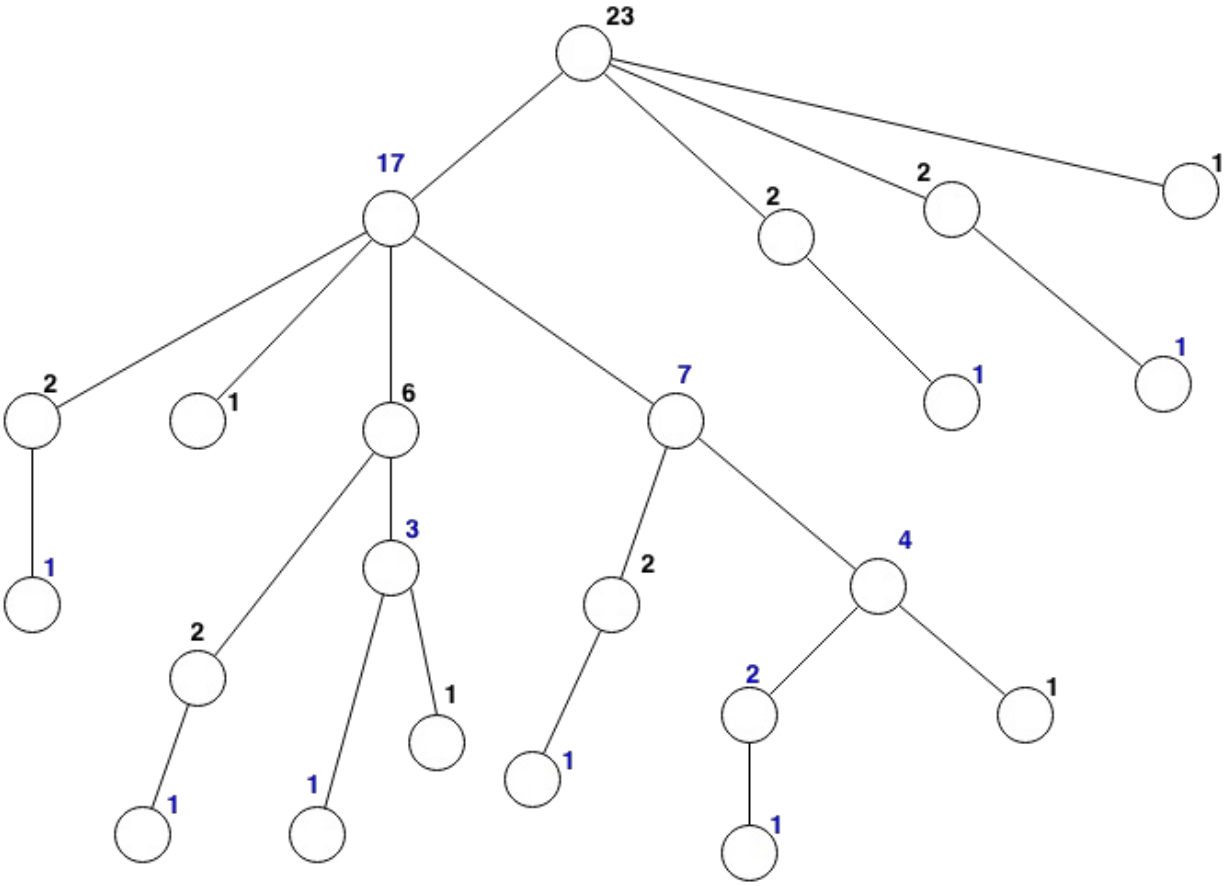
Now the path from any **node A to any node B** can be broken into two paths: **A to LCA(A, B) and B to LCA(A, B)**.

Time Complexity

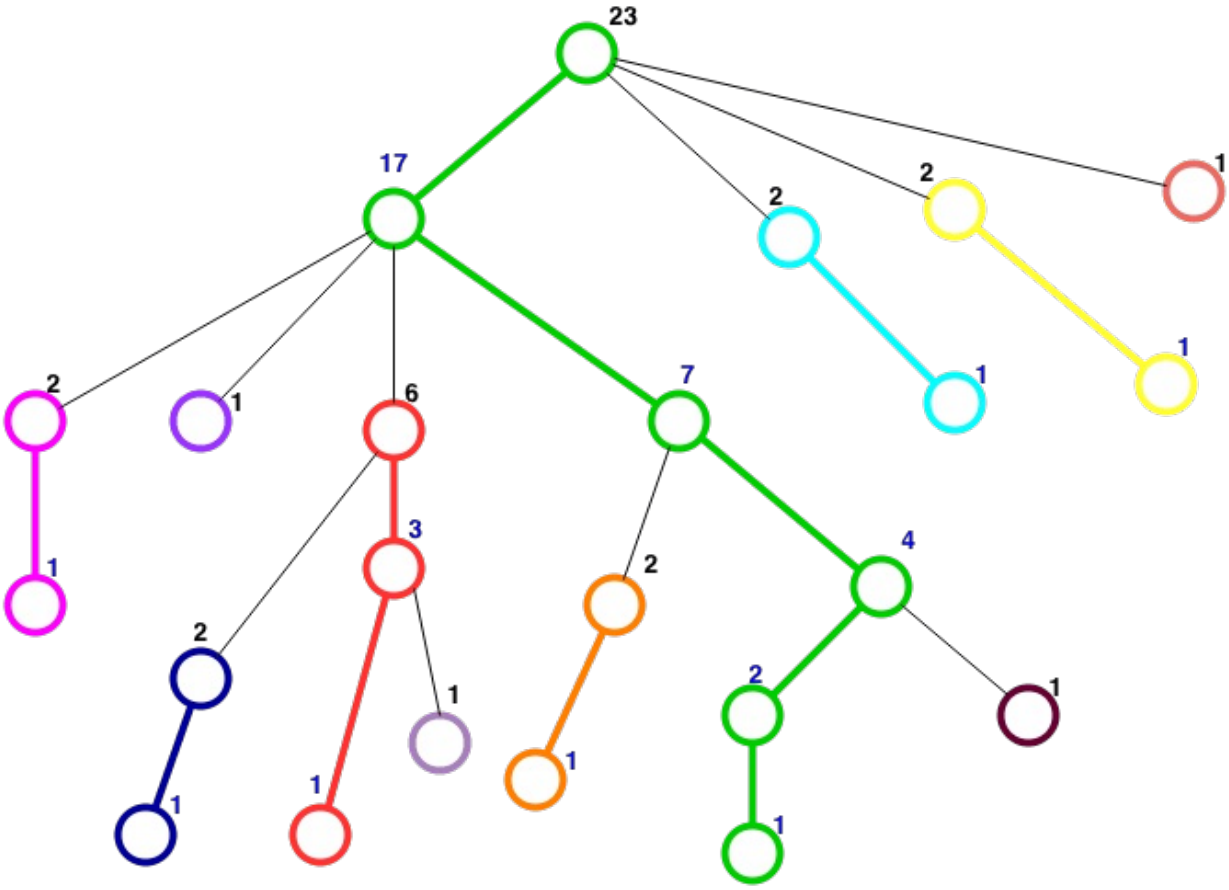
We already know that queries in **each chain** can be answered with **O(log N)** complexity and there are **at most log N chains** we need to consider per path. So on the whole we have **O(log^2 N) complexity** solution.

Special Child : Among all child nodes of a node, the one with **maximum sub-tree size** is considered as Special child. Each non leaf node has exactly one Special child.

Special Edge : For each **non-leaf node**, the edge **connecting the node with its Special child** is considered as Special Edge.



Each node has its sub-tree size written on top.
Each non-leaf node has exactly one special child whose sub-tree size is colored.
Colored child is the one with maximum sub-tree size.



Each Chain is represented with different color.
Thin Black lines represent the connecting edges. They connect 2 chains.

Algorithm

```
HLD(curNode, Chain):
    Add curNode to curChain
    If curNode is LeafNode: return                    //N
othing left to do
    sc := child node with maximum sub-tree size       //s
c is the special child
    HLD(sc, Chain)                                    //E
xtend current chain to special child
    for each child node cn of curNode:                //F
or normal childs
```

```
        if cn != sc: HLD(cn, newChain)                  //A
s told above, for each normal child, a new chain starts
```

dfs()

```
void dfs(int cur, int prev, int level = 0){
    //pa[cur][0] will be its immediate parent
    pa[cur][0] = prev;
    //setting the depth of the current node
    depth[cur] = level;
    //Initially set the subtree size of every node as 1
    //Add the subtree size of every child node of cur nod
e in its size
    subsz[cur] = 1;
    for (int i = 0; i < adj[cur].size(); i++){
        if (adj[cur][i] != prev){
            otherEnd[ind[cur][i]] = adj[cur][i];
            dfs(adj[cur][i], cur, level + 1);
            subsz[cur] += subsz[adj[cur][i]];
        }
    }
}
```

LCA()

```
int LCA(int p, int q){
    if (depth[q] > depth[p])swap(p, q);
    int diff = depth[p] - depth[q];
```

```
    for (int i = 0; i < LN; i++){
        if ((diff >> i) & 1)
            p = pa[p][i];
    }
    //Now p and  q are at same level
    if (p == q)return p;
    for (int i = LN - 1; i >= 0; i--){
        if (pa[p][i] != -1 && pa[p][i] != pa[q][i]){
            p = pa[p][i];
            q = pa[q][i];
        }
    }
    return pa[p][0];
}
```

HLD()

```
ChainNo --> Chain number of current chain.
chainHead[i] --> Chain head of the chain in which i-th ve
rtex is present.
chainPos[i]  --> Position of i-th node in its chain.
chainInd[i] --> Index of chain in which i-th node is pres
ent.
chainSize[chainNo] --> Size of chain with chain number ch
ainNo.
baseArray[] --> array on which segment tree will operate,
 all the nodes in the same chain will be adjacent in base
 array
```

```
posInBase[i] --> position of i-th node in base array


int chainNo=0,chainHead[N],chainPos[N],chainInd[N],chainS
ize[N];
void hld(int cur,, int prev) {
    //If we have encountered this chain for the first tim
e
    if(chainHead[chainNo] == -1) chainHead[chainNo]=cur;
    //This node will be present in current chain
    chainInd[cur] = chainNo;
    //Position of current node in chain will be present s
ize of chain
    chainPos[cur] = chainSize[chainNo];
    //Increment size of chain
    chainSize[chainNo]++;
    // Position of this node in baseArray which we will u
se in Segtree
    //Initially ptr was 0
    posInBase[curNode] = ptr;
    //Insert the cost/value of node in the base array
    baseArray[ptr++] = cost;

    //Find the maximum sized subtree and its index
    int ind = -1,mx_sz = -1;
    for(int i = 0; i < adj[cur].sz; i++) {
            //make sure we are not calling HLD for the pa
rent node
```

```
            if(adj[cur][i] != prev && subsize[ adj[cur][i
] ] > mx_sz) {
                    mx_sz = subsize[ adj[cur][i] ];
                    ind = i;
        }
    }

    //continue the HLD for present chain with special chi
ld as new new member
    //of current chain
    if(ind >= 0) hld( adj[cur][ind], cur);

    for(int i = 0; i < adj[cur].sz; i++) {
        //If the child is not special, start a new chain
        if(i != ind) {
            chainNo++;
            hld( adj[cur][i], cur );
        }
    }
}
```

query_up()

```
//v is ancestor of u
//query the chain in which 'u' is present till head of th
at chain, then move to next chain up
//we do that till u and v are in the same chain and then
we query from u to v in the same chain :D
```

```
int query_up(int u, int v){
    if (u == v)return 0;
    int uchain, vchain = chainInd[v], ans = -1;
    while (1){
        //If both u and v are in same chain, query that c
hain and we are done
        uchain = chainInd[u];
        if (uchain == vchain){
            if (u == v)break;
            ans = max(ans,query_tree(1, 0, sz, posInBase[
v] + 1, posInBase[u]));
            break;
        }
        //else, query the u-chain from u to its head and
then change the chain
        //by calling query from parent[u] to v
        ans = max(ans,query_tree(1, 0, sz, posInBase[chai
nHead[uchain]], posInBase[u]));
        u = chainHead[uchain];
        u = pa[u][0];//move to parent of u, we are moving
 a chain up

        //keep on doing this untill u and v are in same c
hain
    }
    return ans;
}
```

query()

```
int query(int p, int q){
    int lca = LCA(p, q);
    //path from p to q is divided into path from p to lca
(p,q) and q to lca(p,q)
    int ans1 = query_up(p, lca);
    int ans2 = query_up(q, lca);

    //If max query
    int ans = max(ans1,ans2);

    //If sum query
    int ans = ans1 + ans2 - cost[LCA(p,q)];
    return ans;
}
```

Time Complexity:

Build Time --> **O(n)**

Query Time --> **O(logn * logn)**

Spoj QTREE

http://www.spoj.com/problems/QTREE/

```
#include<bits/stdc++.h>
typedef long long ll;
```

```
#define fast std::ios::sync_with_stdio(false);std::cin.ti
e(false)
#define endl "\n"
//#define abs(a) a >= 0 ? a : -a
#define ll long long int
#define mod 1000000007
#define Endl endl
using namespace std;

const int N = 10000 + 10;
const int MAX = 1e6 + 10;
const int LN = 14;
vector<int> adj[N], costs[N], ind[N];
int n, sz, arr[N], tree[N], depth[N], pa[N][20], otherEnd
[N], subsz[N];
int chainNo, chainHead[N], chainInd[N], posInBase[N];
int st[4 * N];

void build(int node, int l, int r){
    if (l == r){
        st[node] = arr[l];
        return;
    }
    int mid = (l + r) >> 1;
    build(2 * node, l, mid);
    build(2 * node + 1, mid + 1, r);
    st[node] = (st[2 * node] > st[2 * node + 1]) ? st[2 *
 node] : st[2 * node + 1];
```

```
}

void update(int node, int l, int r, int i, int val){
    if (l > r || l > i || r < i)return;
    if (l == i && r == i){
        st[node] = val;
        return;
    }
    int mid = (l + r) >> 1;
    update(2 * node, l, mid, i, val);
    update(2 * node + 1, mid + 1, r, i, val);
    st[node] = (st[2 * node] > st[2 * node + 1]) ? st[2 *
 node] : st[2 * node + 1];
}

int query_tree(int node, int l, int r, int i, int j){
    if (l > r || l > j || r < i){
        return -1;
    }
    if (l >= i && r <= j){
        return st[node];
    }
    int mid = (l + r) >> 1;
    return max(query_tree(2 * node, l, mid, i, j), query_
tree(2 * node + 1, mid + 1, r, i, j));
}

//v is ancestor of u
```

```
//query the chain in which 'u' is present till head of th
at chain, then move to next chain up
//we do that till u and v are in the same chain and then
we query from u to v in the same chain :D
int query_up(int u, int v){
    if (u == v)return 0;
    int uchain, vchain = chainInd[v], ans = -1;
    while (1){
        uchain = chainInd[u];
        if (uchain == vchain){
            if (u == v)break;
            ans = max(ans,query_tree(1, 0, sz, posInBase[
v] + 1, posInBase[u]));
            break;
        }
        ans = max(ans,query_tree(1, 0, sz, posInBase[chai
nHead[uchain]], posInBase[u]));
        u = chainHead[uchain];
        u = pa[u][0];//move to parent of u, we are moving
 a chain up
    }
    return ans;
}

int LCA(int p, int q){
    if (depth[q] > depth[p])swap(p, q);
    int diff = depth[p] - depth[q];
    for (int i = 0; i < LN; i++){
```

```
        if ((diff >> i) & 1)
            p = pa[p][i];
    }
    //Now p and  q are at same level
    if (p == q)return p;
    for (int i = LN - 1; i >= 0; i--){
        if (pa[p][i] != -1 && pa[p][i] != pa[q][i]){
            p = pa[p][i];
            q = pa[q][i];
        }
    }
    return pa[p][0];
}

int query(int p, int q){
    int lca = LCA(p, q);
    //path from p to q is divided into path from p to lca
(p,q) and q to lca(p,q)
    int ans = query_up(p, lca);
    int temp = query_up(q, lca);
    if (temp > ans)ans = temp;
    return ans;
}

void change(int i, int val){
    int p = otherEnd[i];
    update(1, 0, sz, posInBase[p], val);
}
```

```
void HLD(int cur, int cost, int prev){
    if (chainHead[chainNo] == -1){
        chainHead[chainNo] = cur;
    }
    chainInd[cur] = chainNo;
    posInBase[cur] = sz;
    arr[sz++] = cost;

    int sc = -1, sc_cost;
    for (int i = 0; i < adj[cur].size(); i++){
        if (adj[cur][i] != prev){
            if (sc == -1 || subsz[sc] < subsz[adj[cur][i]
]){
                sc = adj[cur][i];
                sc_cost = costs[cur][i];
            }
        }
    }
    if (sc != -1)HLD(sc, sc_cost, cur);

    for (int i = 0; i < adj[cur].size(); i++){
        if (adj[cur][i] != prev && sc != adj[cur][i]){
            //new chain at each normal node :)
            chainNo++;
            HLD(adj[cur][i], costs[cur][i], cur);
        }
    }
```

```
}

void dfs(int cur, int prev, int level = 0){
    pa[cur][0] = prev;
    depth[cur] = level;
    subsz[cur] = 1;
    for (int i = 0; i < adj[cur].size(); i++){
        if (adj[cur][i] != prev){
            otherEnd[ind[cur][i]] = adj[cur][i];
            dfs(adj[cur][i], cur, level + 1);
            subsz[cur] += subsz[adj[cur][i]];
        }
    }
}

int main(){
    int t;
    scanf("%d", &t);
    while (t--){
        //printf("\n");
        sz = 0;
        scanf("%d", &n);
        for (int i = 0; i < n; i++){
            adj[i].clear();
            costs[i].clear();
            ind[i].clear();
            chainHead[i] = -1;
            for (int j = 0; j < LN; j++)pa[i][j] = -1;
```

```
        }
        for (int i = 1; i < n; i++){
            int a, b, c;
            scanf("%d %d %d", &a, &b, &c);
            --a;--b;
            adj[a].push_back(b);
            adj[b].push_back(a);
            ind[a].push_back(i - 1);
            ind[b].push_back(i - 1);
            costs[a].push_back(c);
            costs[b].push_back(c);
        }
        chainNo = 0;
        dfs(0, -1);//setup subtree size, depth and parent
 for each node;
        //cout << "dfs over !!!" << endl;
        HLD(0, -1, -1);//decompose the tree and create th
e baseArray
        //cout << "HDL over" << endl;
        build(1, 0, sz);
        //cout << "Build over" << endl;
        //for (int i = 0; i < n; i++)cout << "edge : " <<
 i << " " << otherEnd[i]<<"   "<<posInBase[i] << endl;

        //bottom up DP code for LCA:
        for (int j = 1; j < LN; j++){
            for (int i = 0; i < n; i++){
                if (pa[i][j - 1] != -1)pa[i][j] = pa[pa[i
```

```
][j - 1]][j - 1];
            }
        }
        char ch[20];
        while (1){
            scanf("%s", ch);
            if (ch[0] == 'D')break;
            int a, b;
            scanf("%d %d", &a, &b);
            //cout << ch << " " << a << " " << b << Endl;
            if (ch[0] == 'Q')
                printf("%d\n", query(a - 1, b - 1));
            else
                change(a - 1, b);
        }
    }
    return 0;
}
```

Spoj QTREE2

http://www.spoj.com/problems/QTREE2/

```
#include<bits/stdc++.h>
typedef long long ll;
#define fast std::ios::sync_with_stdio(false);std::cin.ti
e(false)
```

```
#define endl "\n"
//#define abs(a) a >= 0 ? a : -a
#define ll long long int
#define mod 1000000007
#define Endl endl
using namespace std;
const int N = 200000 + 10;
const int MAX = 1e6 + 10;
const int LN = 15;
vector<int> adj[N], costs[N], ind[N];
int n, sz, arr[N], tree[N], depth[N], pa[N][20], otherEnd
[N], subsz[N];
int chainNo, chainHead[N], chainInd[N], posInBase[N];
int st[4 * N];

void build(int node, int l, int r){
    if (l == r){
        st[node] = arr[l];
        return;
    }
    int mid = (l + r) >> 1;
    build(2 * node, l, mid);
    build(2 * node + 1, mid + 1, r);
    st[node] = st[2 * node] + st[2 * node + 1];
}

int query_tree(int node, int l, int r, int i, int j){
    if (l > r || l > j || r < i){
```

```
        return 0;
    }
    if (l >= i && r <= j){
        return st[node];
    }
    int mid = (l + r) >> 1;
    return (query_tree(2 * node, l, mid, i, j) + query_tr
ee(2 * node + 1, mid + 1, r, i, j));
}

int query_up(int u,int v){
    if (u == v)return 0;
    int uchain, vchain = chainInd[v], ans = 0;//uchain an
d vchain contains the chain number
    while (1){
        uchain = chainInd[u];
        if (uchain == vchain){
            if (u == v)break;
            ans += query_tree(1, 0, sz, posInBase[v] + 1,
 posInBase[u]);//please note that we are doing +1 because
 arr[v] contains the edge length

                //between node 'v-1' and 'v'. So by doing
arr[posInBase[v] + 1] we get the

                //first edge between v and v + 1 and so fo
rth and so on upto u.
            break;
```

```
        }
        ans += query_tree(1, 0, sz, posInBase[chainHead[u
chain]], posInBase[u]);
        u = chainHead[uchain];
        u = pa[u][0];
    }
    return ans;
}

int LCA(int u, int v){
    if (depth[u] < depth[v])swap(u, v);
    int diff = depth[u] - depth[v];
    for (int i = 0; i < LN; i++){
        if ((diff >> i) & 1){
            u = pa[u][i];
        }
    }
    if (u == v)return u;
    //Now u and v are at the same level
    for (int i = LN - 1; i >= 0; i--){
        if (pa[u][i] != pa[v][i]){
            u = pa[u][i];
            v = pa[v][i];
        }
    }
    return pa[u][0];
}
```

```
int query(int u, int v){
    int lca = LCA(u, v);
    int ans = query_up(u, lca);
    ans += query_up(v, lca);
    //cout << "query:   " << u << "  " << v << "   lca :
" << lca << endl;
    return ans;
}

void HLD(int node, int prev, int cost){
    //cout << "head :  " << chainNo << " " << node<<"   "
<<chainHead[chainNo] << endl;
    if (chainHead[chainNo] == -1){
        chainHead[chainNo] = node;
    }
    chainInd[node] = chainNo;
    posInBase[node] = sz;
    arr[sz++] = cost;

    int sc = -1, ncost;
    for (int i = 0; i < adj[node].size(); i++){
        if (adj[node][i] != prev){
            if (sc == -1 || subsz[sc] < subsz[adj[node][i
]]){
                sc = adj[node][i];
                ncost = costs[node][i];
            }
        }
```

```
    }
    if (sc != -1){
        HLD(sc, node, ncost);
    }
    for (int i = 0; i < adj[node].size(); i++){
        if (adj[node][i] != prev && adj[node][i] != sc){
            chainNo++;
            HLD(adj[node][i], node, costs[node][i]);
        }
    }
    return;
}

void dfs(int node, int prev, int level){
    pa[node][0] = prev;
    depth[node] = level;
    subsz[node] = 1;
    for (int i = 0; i < adj[node].size(); i++){
        if (adj[node][i] != prev){
            otherEnd[ind[node][i]] = adj[node][i];
            dfs(adj[node][i], node, level + 1);
            subsz[node] += subsz[adj[node][i]];
        }
    }
}

int getkth(int p, int q, int k){
    int lca = LCA(p, q), d;
```

```
    if (lca == p){
        d = depth[q] - depth[p] + 1;
        swap(p, q);//we want p to be at higher depth.....
so swap p and q if p is at lower depth i.e. it is the lca
        k = d - k + 1;//decide 'k' accordingly i.e. k wil
l now become total distance minus k as we have now change
 our p(which was originally q)
    }
    else if (q == lca);//do nothing if q is lca
    //case when neither p and q are lca
    else{
        d = depth[p] + depth[q] - 2 * depth[lca] + 1;
        /*
        d denotes the total dist between the nodes p and
q. it will be = dist(p,lca) + dist(lca,q) - 1
        = (depth[p] - depth[lca] + 1) + (depth[q] - depth
[lca] + 1) - 1
        = depth[p] + depth[q] - 2 * depth[lca] + 1
        */
        if (k > depth[p] - depth[lca] + 1){//case when 'k
' will be between lca and q i.e. dist b/w lca and 'p' is
less than k
            k = d - k + 1;//change 'k' accordingly
            swap(p, q);//swap p and q  as we want to calc
ulate the dist from 'p' only.
        }
    }
    //Now we have set starting node as 'p' and changed k
```

```
accordingly such that the kth node between 'p'
    //and 'q' will always lie between 'p' and lca(p,q) at
 a dist 'k' from p
    //Also dist(p,lca) > k
    //cout << "p : " << p << "  q : " << q << "   k : " <
< k << "   lca : " << lca << endl;
    k--;//decrement k as k = 1 will indicate p itself.
    for (int i = LN - 1; i >= 0; i--){
        if ((1 << i) <= k){//if k is greater than or equa
l to 2^i than we can move up by that much nodes
            p = pa[p][i];//p will become 2^i th ancestor
of p
            k -= (1 << i);//we will move 2^i nodes up and
 k will be decreased by that amount
        }
    }
    return p;
}

int main(){
    int t, a, b, c, x, y;
    scanf("%d", &t);
    while (t--){
        scanf("%d", &n);
        for (int i = 0; i < n; i++){
            ind[i].clear();
            adj[i].clear();
            costs[i].clear();
```

```
            chainHead[i] = -1;
            for (int j = 0; j < LN; j++)pa[i][j] = -1;
        }
        for (int i = 1; i < n; i++){
            scanf("%d %d %d", &a, &b, &c);
            --a; --b;
            adj[a].push_back(b);
            adj[b].push_back(a);
            costs[a].push_back(c);
            costs[b].push_back(c);
            ind[a].push_back(i - 1);
            ind[b].push_back(i - 1);
        }
        //cout << "chaiHead : "; for (int i = 0; i < n; i
++)cout << chainHead[i] << "  "; cout << endl;
        chainNo = 0;
        dfs(0, -1, 0);
        HLD(0, -1, -1);
        //cout<<"arr : ";for (int i = 0; i <= sz; i++)cou
t << arr[i] << "  "; cout << endl;
        build(1, 0, sz);
        for (int j = 1; j < LN; j++){
            for (int i = 0; i < n; i++){
                if (pa[i][j - 1] != -1)pa[i][j] = pa[pa[i
][j - 1]][j - 1];
            }
        }
        //cout << "sz :  " << sz << endl;
```

```
        //for (int i = 0; i <= chainNo; i++)cout << "chai
n : " << i << "  " << chainHead[i] << endl;
        char ch[20];
        while (1){
            scanf("%s", ch);
            if (ch[1] == 'O')break;
            if (ch[0] == 'D'){
                scanf("%d %d", &a, &b);
                printf("%d\n", query(a-1, b-1));
            }
            else if (ch[0] == 'K'){
                scanf("%d %d %d", &a, &b, &c);
                printf("%d\n", getkth(a - 1, b - 1, c) +
1);
            }
        }
    }
    return 0;
}
```

Important Resources:

I would suggest all the readers to go through anudeep's bolg as most of the content have been taken from there:

https://blog.anudeep2011.com/heavy-light-decomposition/

Problems to try:

SPOJ – QTREE – http://www.spoj.com/problems/QTREE/

SPOJ – QTREE2 – http://www.spoj.com/problems/QTREE2/

SPOJ – QTREE3 – http://www.spoj.com/problems/QTREE3/

SPOJ – QTREE4 – http://www.spoj.com/problems/QTREE4/

SPOJ – QTREE5 – http://www.spoj.com/problems/QTREE5/

SPOJ – COT – http://www.spoj.com/problems/COT/

SPOJ – COT2 – http://www.spoj.com/problems/COT2/

SPOJ – COT3 – http://www.spoj.com/problems/COT3/

SPOJ – GOT – http://www.spoj.com/problems/GOT/

SPOJ – GRASSPLA – http://www.spoj.com/problems/GRASSPLA/

SPOJ – GSS7 – http://www.spoj.com/problems/GSS7/

CODECHEF – RRTREE – http://www.codechef.com/problems/RRTREE

CODECHEF – QUERY – http://www.codechef.com/problems/QUERY

CODECHEF – QTREE – http://www.codechef.com/problems/QTREE

CODECHEF – DGCD – http://www.codechef.com/problems/DGCD

CODECHEF – MONOPLOY – http://www.codechef.com/problems/MONOPLOY
