树上部分

LCA

倍增

```
void bfs(int root){
    depth[0]=0,depth[root]=1;  //depth[0]初始化深度为0表示不合法的深度
    queue<int> Q;Q.push(root);

    while(Q.size()){
        int t=Q.front();
        Q.pop();

        for(int i=h[t];~i;i=ne[i]){
            int j=e[i];
            if(depth[j]>depth[t]+1){
                depth[j]=depth[t]+1;

                Q.push(j);
                fa[j][0]=t;
                for(int k=1;k<=15;k++)
                    fa[j][k]=fa[fa[j][k-1]][k-1];
            }
        }
    }
}

inline int lca(int a,int b){
    if(depth[a]<depth[b]) swap(a,b);
    //从后向前遍历
    for(int i=15;i>=0;i--){
        if(depth[fa[a][i]]>=depth[b])
            a=fa[a][i];
    }

    if(a==b) return a;

    for(int i=15;i>=0;i--)
        if(fa[a][i]!=fa[b][i]){
            a=fa[a][i],b=fa[b][i];
        }
    return fa[a][0];
}
```

Tarjan离线O(n)

```
void dfs(int u,int father){ //算出每个点到根节点的距离
    for(int i=h[u];~i;i=ne[i]){
        int j=e[i];
        if(j==father) continue;
        dist[j]=dist[u]+w[i];
        dfs(j,u);
    }
}

void tarjan(int u){
    st[u]=1;
    for(int i=h[u];~i;i=ne[i]){
        int j=e[i];
        if(!st[j]){
            tarjan(j);
            fa[j]=u;
        }
    }

    for(int i=0;i<query[u].size();i++){
        int ver=query[u][i].x,id=query[u][i].y;
        if(st[ver]==2){ //如果已经被划分到了左边被遍历过的子树中
            int root=find(ver);
            res[id]=dist[u]+dist[ver]-2*dist[root];
        }
    }
    st[u]=2;
}
```

树链剖分

```
#define int LL
const int N = 100010, M = N*2;
int n,m,e[M],ne[M],h[N],idx=0,w[M];
int val[N],fa[N],son[N],top[N],sz[N],dep[N],id[N],cnt=0;
//top[u]重链的顶点，val[u]是维护的序列
struct Node{
    int l,r;
    int sum,add;
}tr[N<<2];

void add(int a,int b){
    e[idx]=b,ne[idx]=h[a],h[a]=idx++;
}

void dfs(int u,int father,int d){
    dep[u]=d,sz[u]=1,fa[u]=father;
    for(int i=h[u];~i;i=ne[i]){
        int j=e[i];
```

```
        if(j==father) continue;
        dfs(j, u, d+1);
        sz[u] += sz[j];
        if(sz[j] > sz[son[u]]) son[u]=j;
    }
}

void dfs(int u,int up){
    id[u]=++cnt,val[cnt]=w[u],top[u]=up;
    if(!son[u]) return ;
    dfs(son[u], up);
    for(int i=h[u];~i;i=ne[i]){
        int j=e[i];
        if(j==fa[u]||j==son[u]) continue;
        dfs(j, j);  //轻儿子所在重链的顶点就是j自己
    }
}

void push_up(int u){
    tr[u].sum=tr[u<<1].sum+tr[u<<1|1].sum;
}

void calc(Node &u,int add){
    u.add+=add;
    u.sum+=add*(u.r-u.l+1);
}

void push_down(int u){
    if(tr[u].add){
        calc(tr[u<<1], tr[u].add);
        calc(tr[u<<1|1], tr[u].add);
        tr[u].add=0;
    }
}

void build(int u,int l,int r){
    tr[u]={l,r};
    if(l==r){
        tr[u].add=0;
        tr[u].sum=val[l];
        return ;
    }
    int mid=l+r>>1;
    build(u<<1,l,mid), build(u<<1|1,mid+1,r);
    push_up(u);
}

void modify(int u,int l,int r,int k){
    if(tr[u].l>=l&&tr[u].r<=r){
        tr[u].add+=k;
        tr[u].sum+=k*(tr[u].r-tr[u].l+1);
        return ;
    }
    push_down(u);
    int mid=tr[u].l+tr[u].r>>1;
    if(l<=mid) modify(u<<1, l, r, k);
    if(r>mid) modify(u<<1|1, l, r, k);
    push_up(u);
```

```
}

int query(int u,int l,int r){
    if(tr[u].l>=l&&tr[u].r<=r){
        return tr[u].sum;
    }
    push_down(u);
    int mid=tr[u].l+tr[u].r>>1;
    int ans=0;
    if(l<=mid) ans+=query(u<<1, l, r);
    if(r>mid) ans+=query(u<<1|1, l, r);
    return ans;
}

void add_road(int u,int v,int k){
    while(top[u]!=top[v]){
        if(dep[top[u]]<dep[top[v]]) swap(u, v);
        modify(1, id[top[u]], id[u], k);
        u=fa[top[u]];
    }
    if(dep[u]<dep[v]) swap(u,v);
    modify(1, id[v], id[u], k);
}

void add_son(int u,int k){
    modify(1, id[u], id[u]+sz[u]-1, k);
}

int query_road(int u,int v){
    int res=0;
    while(top[u]!=top[v]){
        if(dep[top[u]] < dep[top[v]]) swap(u, v);
        res+=query(1, id[top[u]], id[u]);
        u=fa[top[u]];
    }
    if(dep[u]<dep[v]) swap(u,v);
    res+=query(1, id[v], id[u]);
    return res;
}

int query_son(int u){
    return (query(1, id[u], id[u]+sz[u]-1));
}

void solve(){
    n=read();
    rep(i,1,n) w[i]=read();
    rep(i,1,n-1){
        int u=read(),v=read();
        add(u, v); add(v, u);
    }
    dfs(1,-1,1);
    dfs(1,1);
    build(1,1,n);

    m=read();
    rep(i,1,m){
        int op=read();
```

```
        int u,v,k;
        if(op==1){
            u=read(),v=read(),k=read();
            add_road(u,v,k);
        }
        else if(op==2){
            u=read(),k=read();
            add_son(u,k);
        }
        else if(op==3){
            u=read(),v=read();
            print(query_road(u, v));
        }
        else{
            u=read();
            print(query_son(u));
        }
    }
}

// ============================================
```

dsu_on_tree

```
const int N=200010,M=N*2,mod=1e9+7;
vector<pair<int,int>> edge[N];
int n,m,k,sz[N],son[N],l[N],r[N],cnt,id[N],dep2[N],dep1[N];
unordered_map<int,int> mp;
LL ans=1e9;

void dfs(int u,int pre){
    l[u]=++cnt,id[cnt]=u,sz[u]=1;
    for(auto v:edge[u]){
        if(v.x==pre) continue;
        dep2[v.x]=dep2[u]+v.y;
        dep1[v.x]=dep1[u]+1;
        dfs(v.x, u);
        sz[u]+=sz[v.x];
        if(sz[son[u]] < sz[v.x]) son[u]=v.x;
    }
    r[u]=cnt;
}

void dfs(int u,int pre,int top){
    for(auto v:edge[u]){
        if(v.x==pre||v.x==son[u]) continue;
        dfs(v.x, u, 0);
    }
    if(son[u]) dfs(son[u], u, 1);

    auto ask=[&](int x)->void{
        int tmp=k+2ll*dep2[u]-dep2[x];
        if(mp.count(tmp)){
```

```
            ans=min(ans, mp[tmp]+dep1[x]-dep1[u]*2);
        }
    };

    auto add=[&](int x)->void{
        if(mp.count(dep2[x]))
            mp[dep2[x]]=min(mp[dep2[x]], dep1[x]);
        else mp[dep2[x]]=dep1[x];
    };

    for(auto v:edge[u]){
        if(v.x==son[u]||v.x==pre) continue;
        for(int i=l[v.x];i<=r[v.x];++i){
            ask(id[i]);
        }
        for(int i=l[v.x];i<=r[v.x];++i){
            add(id[i]);
        }
    }
    ask(u);
    add(u);
    if(!top) mp.clear();
}
// ============================================================
const int N=200010,M=N*2,mod=1e9+7;
int n,m,e[M],ne[M],h[N],sz[N],son[N],color[N],idx;
int sum=0,cnt[N],mx,ans[N];

void add(int a,int b){
    e[idx]=b,ne[idx]=h[a],h[a]=idx++;
}

void dfs_son(int u,int pre){
    sz[u]=1;
    for(int i=h[u];~i;i=ne[i]){
        int j=e[i];
        if(j==pre) continue;
        dfs_son(j ,u);
        if(sz[j]>sz[son[u]]) son[u]=j;
        sz[u]+=sz[j];
    }
}

void modify(int u,int pre,int op,int zson){
    int c=color[u];
    cnt[c] += op;
    if(cnt[c]>mx) mx=cnt[c], sum=c;
    else if(cnt[c]==mx) sum += c;

    for(int i=h[u];~i;i=ne[i]){
        int j=e[i];
        if(j==pre||j==zson) continue;
        modify(j, u, op, zson);
    }
}

void dfs(int u,int pre,int s){
    for(int i=h[u];~i;i=ne[i]){
```

```
        int j=e[i];
        if(j==pre||j==son[u]) continue;
        dfs(j, u, 0);
    }
    if(son[u]) dfs(son[u], u, 1);
    modify(u, pre, 1, son[u]);
    ans[u]=sum;
    if(!s) modify(u, pre, -1, 0),sum=0,mx=0;
}

//======================map版本_使用name数组来巧妙完成map的赋值
const int N=200010,M=N*2,mod=1e9+7;
int n,m,w[N],cnt[N];
LL ans[N];
vector<int> edge[N];
int sz[N],son[N],l[N],r[N],idx,id[N];
map<pii,int> e;
struct Node{
    map<int,int> mp;
    LL ans;
    void add(int x){
        ans -= mp[x]*(cnt[x]-mp[x]);
        mp[x] ++;
        ans += mp[x]*(cnt[x]-mp[x]);
    }
}tr[N];
int name[N];

void dfs(int u,int pre){
    l[u]=++idx,sz[u]=1,id[idx]=u;
    for(auto v:edge[u]){
        if(v==pre) continue;
        dfs(v, u);
        sz[u] += sz[v];
        if(sz[v]>sz[son[u]]) son[u]=v;
    }
    r[u]=idx;
}

void dfs(int u,int pre,int top){
    for(auto v:edge[u]){
        if(v==pre||v==son[u]) continue;
        dfs(v, u, 0);
    }
    if(son[u]) dfs(son[u], u, 1);

    if(son[u]) name[u]=name[son[u]];
    for(auto v:edge[u]){
        if(v==son[u]||v==pre) continue;
        for(int i=l[v];i<=r[v];++i){
            int ver=id[i];
            tr[name[u]].add(w[ver]);
        }
    }
    tr[name[u]].add(w[u]);
    ans[e[make_pair(u, pre)]]=tr[name[u]].ans;
}
```

```
void solve(){
    n=read();
    rep(i,1,n) w[i]=read(),cnt[w[i]]++,name[i]=i;
    rep(i,1,n-1){
        int u=read(),v=read();
        edge[u].push_back(v);
        edge[v].push_back(u);
        e[make_pair(u,v)]=i;
        e[make_pair(v,u)]=i;
    }
    dfs(1,0);
    dfs(1,0,1);
    rep(i,1,n-1) printf("%lld ",ans[i]);
}
```

DSU on tree，作为一个比较神奇的算法，常常可以代替树剖/树上莫队来完成一些分别处理以每个根节点为问题核心关于其子树的问题。

本算法的思想就是类似于树剖，每次维护树的重儿子为根的子树，即儿子最多的子树，这样可以保证每个点呗操作一次会使得维护的节点数至少翻倍。因此每个点最多被维护logn次，总时间复杂度为O(nlogn)

模板题：

A - Lomsat gelral

大意是找到以每个点为根节点的子树的主要颜色之和

```
#define int LL
const int N=200010,M=N*2,mod=1e9+7;
int n,m,e[M],ne[M],h[N],sz[N],son[N],color[N],idx;
int sum=0,cnt[N],mx,ans[N];

void add(int a,int b){
    e[idx]=b,ne[idx]=h[a],h[a]=idx++;
}

void dfs_son(int u,int pre){
    sz[u]=1;
    for(int i=h[u];~i;i=ne[i]){
        int j=e[i];
        if(j==pre) continue;
        dfs_son(j ,u);
        if(sz[j]>sz[son[u]]) son[u]=j;
        sz[u]+=sz[j];
    }
}

void modify(int u,int pre,int op,int zson){
    int c=color[u];
    cnt[c] += op;
    if(cnt[c]>mx) mx=cnt[c], sum=c;
    else if(cnt[c]==mx) sum += c;

    for(int i=h[u];~i;i=ne[i]){
        int j=e[i];
```

```
        if(j==pre||j==zson) continue;
        modify(j, u, op, zson);
    }
}

void dfs(int u,int pre,int s){
    for(int i=h[u];~i;i=ne[i]){
        int j=e[i];
        if(j==pre||j==son[u]) continue;
        dfs(j, u, 0);
    }
    if(son[u]) dfs(son[u], u, 1);
    modify(u, pre, 1, son[u]);
    ans[u]=sum;
    if(!s) modify(u, pre, -1, 0),sum=0,mx=0;
}

void solve(){
    n=read();
    rep(i,1,n) color[i]=read();
    rep(i,1,n-1){
        int u=read(),v=read();
        add(u,v), add(v,u);
    }
    dfs_son(1,-1);
    dfs(1,-1,1);
    for(int i=1;i<=n;++i) printf("%lld ",ans[i]);
}
```

算是dsu on tree的模板了，先一遍DFS与处理出所有节点的重儿子，然后第二遍dfs的时候先dfs所有的非重儿子,再dfs所有的重儿子来处理答案。有一个优化小技巧，预处理的时候顺便与处理出dfs序，这样就不需要递归修改，改为直接循环修改，这样可以减小代码常熟。

dsu on tree 算是一个越来越板的套路，基本上只要改一改板子，转化一下问题就可以了。

B - Tree Requests

求每一个点特定数量的字符是否可以构成回文串。也是非常的板的一道题目，能不能构成回文串取决于奇数个字符的数量是否不超过2.因此直接再dsu on tree的时候维护一下每个点特定深度下的26个字母的个数即可。

```
#define N 500010
int n,m;
vector<int> edge[N];
int sz[N],dep[N],son[N];
vector<pii> qry[N];
int ans[N];
int cnt[N][27];
int l[N],id[N],r[N],idx=0;
char ch[N];

inline void dfs(int u,int pre){
    sz[u]=1,dep[u]=dep[pre]+1;l[u]=++idx,id[idx]=u;
    for(auto v:edge[u]){
        if(v==pre) continue;
        dfs(v, u);
        sz[u] += sz[v];
        if(sz[son[u]]<sz[v]) son[u]=v;
```

```
    }
    r[u]=idx;
}

inline void add(int u,int d){
    cnt[dep[u]][ch[u]-'a']+=d;
}

void dfs(int u,int pre,int top){
    for(auto v:edge[u]){
        if(v==pre||v==son[u]) continue;
        dfs(v, u, 0);
    }
    if(son[u]) dfs(son[u], u, 1);

    for(auto v:edge[u]){
        if(v==pre||v==son[u]) continue;
        for(int i=l[v];i<=r[v];++i)
            add(id[i], 1);
    }
    add(u, 1);

    for(auto q:qry[u]){
        int tmp=0;
        for(int i=0;i<26;++i){
            if(cnt[q.x][i]&1) tmp++;
        }
        ans[q.y]=(tmp<2);
    }

    if(!top) {
        for(int i=l[u];i<=r[u];++i){
            add(id[i], -1);
        }
    }
}

void solve(){
    n=read(),m=read();
    rep(i,2,n) {
        int x=read();
        edge[x].push_back(i);
        edge[i].push_back(x);
    }
    scanf("%s",ch+1);
    rep(i,1,m){
        int x=read(),h=read();
        qry[x].push_back({h,i});
    }
    dfs(1,0);
    dfs(1,0,1);
    rep(i,1,m) puts(ans[i]?"Yes":"No");
}
```

C - Blood Cousins Return

求对于每个点相对深度下不同名字的孙子的数量。将相对深度转化为绝对深度，借助set<string>来维护对应深度的名字数量

```
#define N 500010
int n,m,x,dep[N],sz[N],son[N],l[N],r[N],id[N],idx,ans[N];
string name[N];
set<string> ds[N];
vector<int> edge[N];
vector<pii> qry[N];

void dfs_init(int u,int d){
    l[u]=++idx,id[idx]=u,sz[u]=1,dep[u]=d;
    son[u]=-1;
    for(auto v:edge[u]){
        dfs_init(v, d+1);
        sz[u] += sz[v];
        if(son[u]==-1||sz[v] > sz[son[u]]) son[u]=v;
    }
    r[u]=idx;
}

void dfs(int u,int top){
    for(auto v:edge[u]){
        if(v==son[u]) continue;
        dfs(v, 0);
    }
    if(son[u]!=-1) dfs(son[u], 1);
    for(auto v:edge[u]){
        if(v==son[u]) continue;
        for(int i=l[v];i<=r[v];++i){
            ds[dep[id[i]]].insert(name[id[i]]);
        }
    }
    ds[dep[u]].insert(name[u]);
    for(auto q:qry[u]){
        int k=q.x,id=q.y;
        ans[id]=ds[k].size();
    }
    if(!top){
        for(int i=l[u];i<=r[u];++i)
            ds[dep[id[i]]].erase(name[id[i]]);
    }
}

void solve(){
    cin >> n;
    rep(i,1,n){
        cin >> name[i] >> x;
        edge[x].push_back(i);
    }
    dfs_init(0,0);
    cin >> m;
    rep(i,1,m){
        int v,k;
        cin >> v >> k;
```

```
        qry[v].push_back({k+dep[v],i});
    }
    dfs(0,1);
    for(int i=1;i<=m;++i) print(ans[i]);
}
```

D - Blood Cousins

计算给定点v的k级表兄的数量，我们可以转化为v的k级祖先的相对深度为k的子孙的数量，答案直接是-1即可。

```
#define N 200010
vector<int> edge[N];
int n,m,sz[N],dep[N],son[N];
int f[20][N],ans[N];
vector<pii> qry[N];
int l[N],r[N],id[N],idx=0;
int cnt[N];

void dfs_init(int u,int fa,int d){
    sz[u]=1,dep[u]=d,son[u]=-1;
    l[u]=++idx,id[idx]=u;
    for(auto v:edge[u]){
        f[0][v]=u;
        dfs_init(v, u, d+1);
        sz[u] += sz[v];
        if(son[u]==-1||sz[son[u]] < sz[v])
            son[u]=v;
    }
    r[u]=idx;
}

void dfs(int u,int top){
    for(auto v:edge[u]){
        if(v==son[u])
            continue;
        dfs(v, 0);
    }
    if(son[u]!=-1) dfs(son[u], 1);
    for(auto v:edge[u]){
        if(v==son[u]) continue;
        for(int i=l[v];i<=r[v];++i)
            cnt[dep[id[i]]] ++;
    }
    cnt[dep[u]] ++;
    for(auto v:qry[u]){
        int dp=v.x,id=v.y;
        ans[id]=cnt[dp]-1;
    }
    if(!top){
        for(int i=l[u];i<=r[u];++i)
            cnt[dep[id[i]]] --;
    }
}

int get(int u,int k){
    for(int i=19;i>=0;--i)
        if(k>>i&1)
```

```
            u=f[i][u];
    return u;
}

void solve(){
    n=read();
    rep(i,1,n){
        int x=read();
        edge[x].push_back(i);
    }
    dfs_init(0,0,0);
    for(int i=1;i<=19;++i)
        for(int j=0;j<=n;++j){
            f[i][j]=f[i-1][f[i-1][j]];
        }
    m=read();
    rep(i,1,m){
        int v=read(),p=read();
        int t=get(v,p);
        if(t) qry[t].push_back({dep[t]+p,i});
    }
    dfs(0,1);
    for(int i=1;i<=m;++i) printf("%d ",ans[i]);
}
```

G - Tree and Queries

这道题要求给定节点的子树中数量不少于给定数的颜色的个数。首先可以对每个点作为根节点来处理子树中的问题，可以用DSU on Tree来解决，其次由于我们不方便直接求得每个颜色对应的个数，因此我们还需要用树状数组来维护每一个个数的颜色数量。

总的时间复杂度为O(nlog2n)

```
// #define int LL
const int N=200010,M=N*2,mod=1e9+7;
int n,m,col[N],sz[N],dep[N],son[N],id[N],l[N],r[N],idx,ans[N];
vector<int> edge[N];
vector<pii> qry[N];
int cnt[N];
struct BIT{
    #define lowbit(x) ((x)&(-x))
    int tr[100005],n;
    void resize(int _n=100000){n=_n;}
    void add(int x,int d){
        for(;x<=n;x+=lowbit(x)) tr[x]+=d;
    }
    int ask(int x){
        int ans=0;
        for(;x;x-=lowbit(x)) ans += tr[x];
        return ans;
    }
}T;

void dfs_init(int u,int pre){
    l[u]=++idx,id[idx]=u;
    dep[u]=dep[pre]+1,sz[u]=1;
    for(auto v:edge[u]){
```

```
        if(v==pre) continue;
        dfs_init(v, u);
        sz[u] += sz[v];
        if(sz[son[u]] < sz[v]) son[u]=v;
    }
    r[u]=idx;
}

void dfs(int u,int pre,int up){
    for(auto v:edge[u]){
        if(v!=pre&&v!=son[u])
            dfs(v, u, 0);
    }
    if(son[u]) dfs(son[u], u, 1);

    for(auto v:edge[u]){
        if(v==pre||v==son[u])
            continue;
        for(int i=l[v];i<=r[v];++i){
            if(cnt[col[id[i]]])
                T.add(cnt[col[id[i]]], -1);
            cnt[col[id[i]]] ++;
            T.add(cnt[col[id[i]]], 1);
        }
    }
    if(cnt[col[u]])
        T.add(cnt[col[u]], -1);
    cnt[col[u]] ++;
    T.add(cnt[col[u]], 1);

    for(auto tmp:qry[u]){
        int x=tmp.x,id=tmp.y;
        ans[id] = T.ask(T.n);
        if(x>1) ans[id] -= T.ask(x-1);
    }

    if(!up){
        for(int i=l[u];i<=r[u];++i){
            T.add(cnt[col[id[i]]], -1);
            cnt[col[id[i]]] --;
            if(cnt[col[id[i]]])
                T.add(cnt[col[id[i]]], 1);
        }
    }
}

void solve(){
    n=read(),m=read();
    rep(i,1,n)
        col[i]=read();
    rep(i,1,n-1){
        int u=read(),v=read();
        edge[u].push_back(v);
        edge[v].push_back(u);
    }
    dfs_init(1,0);
    rep(i,1,m){
        int x=read(),t=read();
```

```
        qry[x].push_back({t,i});
    }
    dfs(1,0,1);
    for(int i=1;i<=m;++i)
        print(ans[i]);
}
```

L. Listing Tedious Paths

A tree is a graph that is connected (there exists a path between any two of its vertices), undirected (the edges of the graph have no direction), and acyclic (there are no cycles).

A colorful tree is a tree in which each of its vertices has a specific color.

A tedious path is a path in the tree such that both the initial and final vertices have the same color, and there is no vertex or edge that appear more than once in the path. Note that the color of the intermediate vertices, if there are any, are irrelevant.

Given a colorful tree, with NN vertices, your task is calculate, for each of the edges, the number of tedious paths that go through that edge.

```
const int N=200010,M=N*2,mod=1e9+7;
int n,m,w[N],cnt[N];
LL ans[N];
vector<int> edge[N];
int sz[N],son[N],l[N],r[N],idx,id[N];
map<pii,int> e;
struct Node{
    map<int,int> mp;
    LL ans;
    void add(int x){
        ans -= mp[x]*(cnt[x]-mp[x]);
        mp[x] ++;
        ans += mp[x]*(cnt[x]-mp[x]);
    }
}tr[N];
int name[N];

void dfs(int u,int pre){
    l[u]=++idx,sz[u]=1,id[idx]=u;
    for(auto v:edge[u]){
        if(v==pre) continue;
        dfs(v, u);
        sz[u] += sz[v];
        if(sz[v]>sz[son[u]]) son[u]=v;
    }
    r[u]=idx;
}

void dfs(int u,int pre,int top){
    for(auto v:edge[u]){
        if(v==pre||v==son[u]) continue;
        dfs(v, u, 0);
    }
    if(son[u]) dfs(son[u], u, 1);

    if(son[u]) name[u]=name[son[u]];
```

```
    for(auto v:edge[u]){
        if(v==son[u]||v==pre) continue;
        for(int i=l[v];i<=r[v];++i){
            int ver=id[i];
            tr[name[u]].add(w[ver]);
        }
    }
    tr[name[u]].add(w[u]);
    ans[e[make_pair(u, pre)]]=tr[name[u]].ans;
}

void solve(){
    n=read();
    rep(i,1,n) w[i]=read(),cnt[w[i]]++,name[i]=i;
    rep(i,1,n-1){
        int u=read(),v=read();
        edge[u].push_back(v);
        edge[v].push_back(u);
        e[make_pair(u,v)]=i;
        e[make_pair(v,u)]=i;
    }
    dfs(1,0);
    dfs(1,0,1);
    rep(i,1,n-1) printf("%lld ",ans[i]);
}
```

点分治

点分治的题目大体的模板都是一样的，只需要改一改分治的部分。我们考虑本题，开一个桶，对于以u为根的树，长度为k的路径有三种，分别是两个端点都在子树内的，横跨根节点端点分别在两棵子树内的，一个端点为根节点的。所以我们开一个桶表示非当前子树路径内的长度为i的边是否存在。然后每次遍历完一个节点后记得将桶清空。

```
const int N=200010,M=N*2,mod=1e9+7,INF=1e9;
int e[M],ne[M],h[N],w[M],idx;
int wc,sz[N],maxp[N],st[N],dis[N],sum;
int t1[N],t2[N],n,m,k;

void add(int a,int b,int c){
    e[idx]=b,w[idx]=c,ne[idx]=h[a],h[a]=idx++;
}

void get_wc(int u,int pre){
    sz[u]=1;
    maxp[u]=0;
    for(int i=h[u];~i;i=ne[i]){
        int j=e[i];
        if(j==pre||st[j]) continue;
        get_wc(j, u);
        sz[u] += sz[j];
        maxp[u]=max(maxp[u], sz[j]);
    }
```

```
    maxp[u]=max(maxp[u], sum-sz[u]);
    if(maxp[u]<maxp[wc]) wc=u;
}

void get_dist(int u,int pre,int dis,int &q){
    t2[q++]=dis;
    for(int i=h[u];~i;i=ne[i]){
        int j=e[i];
        if(j==pre||st[j]) continue;
        get_dist(j, u, dis+w[i], q);
    }
}

void calc(int u){ //对于以u为重心形态的树进行分治
    int p=0;
    for(int i=h[u];~i;i=ne[i]){
        int j=e[i];
        if(st[j]) continue;
        int q=0;
        get_dist(j, u, w[i], q);
        for(int i=0;i<q;++i){
            /*对于每个子树进行计算*/
        }
        for(int i=0;i<q;++i){
            t1[p++]=t2[i];/* 合并conquer*/
        }
    }
    /*for(int i=0;i<p;++i) clear; 清空当前层*/
}

void divide(int u){
    st[u]=true;
    calc(u);
    for(int i=h[u];~i;i=ne[i]){
        int j=e[i];
        if(st[j]) continue;
        sum = sz[j];
        maxp[wc=0]=INF;
        get_wc(j, 0); //找到重心
        divide(wc);   //树上分治递归
    }
}

void solve(){
    memset(h,-1,sizeof h);
    n=read(),m=read();
    for(int i=1;i<n;++i){
        int u=read(),v=read(),w=read();
        add(u,v,w), add(v,u,w);
    }
    maxp[wc]=sum=n;
    get_wc(1,0);
    divide(wc);
    // solve....
}
```

树上莫队

树上莫队

欧拉序

lca.

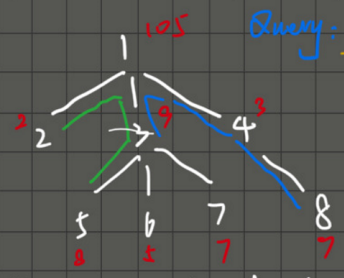
莫队

树上莫队

例：一个N个节点的树，1~N，M次询问，u,v.

u → v 路径上有多少种不同权值。

$\begin{cases} N \le 40000 \\ M \le 10^5 \\ 各点权值 < 2^{31}-1 \end{cases}$



通用做法：将树变成序列来出行正面处理，转化为欧拉序

欧拉序：1 2 2 3 5 5 6 6 7 7 3 4 8 8 4 1

刚进来写一次，最后出时写一次

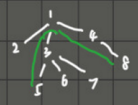
f做法：① 用 fi[] 和 la[] 分别记录每个点第一次出现和最后一次出现的位置。

② 对询问 x,y，若 fi[x] > fi[y] 则交换使其满足 fi[x] < fi[y]

③ 1. 若 lca(x,y) = x，欧拉序列中会对应 [fi[x], fi[y]] 中只出现一次的点就对应 x 到 y 的路径。

2. lca(x,y) ≠ x，欧拉序列中对应 [la[x], fi[y]] 中只出现一次的和 lca(x,y)

即求欧拉序中只出现1次的数有多少不同。



莫队维护：cnt[] res st[]：记录每个数出现多少次。

```
add(x):
    st[x] ^= 1
  if(st[x] == 0)
    cnt[x]--;
    if(cnt[x] == 0)
      res--
  else
      if(cnt[x]==0)
          res++;
      cnt[x]++;
```

增加和减少是个函数。

思路整理：

① 对所有点权值离散化

② 求树的欧拉序列和 fi[], la[]

③ 求 lca，用倍增法 f[u][i] u的第2^i的祖先

④ 将树上询问变成欧拉序中的询问

⑤ 莫队维护 { cnt[] 权值出现次数 ; st[] 结点出现次数 ; res.

树上莫队的经典套路：

- 对于值域较大的进行离散化
- 在求解欧拉序的同时处理出每个点在序列中第一次出现和最后一次出现的位置
- bfs预处理出lca所需数组并在读入询问的时候根据lca判断询问在序列中的起始点
- 对询问排序后进行莫队，每次正常莫队区间完需要判断是否应该加入最近公共祖先
- 例题

给定一棵 N 个节点的树，节点编号从 1 到 N，每个节点都有一个整数权值。

现在，我们要进行 M 次询问，格式为 u v，对于每个询问你需要回答从 u 到 v 的路径上（包括两端点）共有多少种不同的点权值。

```
const int N = 1e5+100;
int n,m,k,Eular[N],dx,fi[N],la[N];
int e[N],ne[N],h[N],idx=0,val[N];
vector<int> alls;
int depth[N],fa[N][20];
int cnt[N],st[N],res,len,ans[N];
struct Ques{
    int l,r,id,p;
}q[N];

int get(int x){
    return x/len;
}

void add_edge(int a,int b){
    e[idx]=b,ne[idx]=h[a],h[a]=idx++;
}

void dfs(int u, int pre){
    Eular[++dx]=u,fi[u]=dx;
    for(int i=h[u];~i;i=ne[i]){
        int j=e[i];
        if(j==pre) continue;
        dfs(j, u);
    }
    Eular[++dx]=u,la[u]=dx;
}

queue<int> Q;
void bfs(){
    memset(depth,0x3f,sizeof depth);
    depth[0]=0,depth[1]=1;
    Q.push(1);

    while(Q.size()){
        auto t=Q.front(); Q.pop();

        for(int i=h[t];~i;i=ne[i]){
            int j=e[i];
            if(depth[j]>depth[t]+1){
                depth[j]=depth[t]+1;

                Q.push(j);
                fa[j][0]=t;
                for(int k=1;k<=16;++k)
```

```
                    fa[j][k]=fa[fa[j][k-1]][k-1];
            }
        }
    }
}

int lca(int a,int b){
    if(depth[a]<depth[b]) swap(a,b);
    for(int i=16;i>=0;--i){
        if(depth[fa[a][i]]>=depth[b])
            a=fa[a][i];
    }

    if(a==b) return a;

    for(int i=16;i>=0;--i){
        if(fa[a][i]!=fa[b][i])
            a=fa[a][i],b=fa[b][i];
    }
    return fa[a][0];
}

void add(int x,int& res){
    st[x]^=1;
    if(st[x]==0){
        cnt[val[x]] --;
        if(cnt[val[x]] == 0)
            res--;
    }
    else{
        if(cnt[val[x]]==0)
            res++;
        cnt[val[x]] ++;
    }
}

void solve(){
    n=read(),m=read();
    rep(i,1,n) val[i]=read(),alls.push_back(val[i]);
    rep(i,1,n-1){
        int u=read(),v=read();
        add_edge(u,v), add_edge(v,u);
    }
    sort(alls.begin(),alls.end());
    alls.erase(unique(alls.begin(),alls.end()),alls.end());
    for(int i=1;i<=n;++i) val[i]=lower_bound(alls.begin(),alls.end(),val[i])-alls.begin()+1;

    dfs(1,-1); //处理出欧拉序列
    bfs();  //预处理lca

    rep(i,1,m){
        int a=read(),b=read();
        if(depth[a]>depth[b]) swap(a,b);
        int p=lca(a,b);
        if(p==a){
            q[i].l=fi[a],q[i].r=fi[b];
            q[i].id=i;
        }
```

```
        else {
            q[i].l=la[a],q[i].r=fi[b];
            q[i].id=i,q[i].p=p;
        }
    }

    len=sqrt(dx);
    sort(q+1,q+1+m,[](Ques &A,Ques& B){
        int i=get(A.l), j=get(B.l);
        if(i!=j) return i<j;
        return A.r < B.r;
    });

    for(int k=1,i=0,j=1,res=0;k<=m;++k){
        int id=q[k].id, l=q[k].l,r=q[k].r,p=q[k].p;
        while(i<r) add(Eular[++ i], res);
        while(i>r) add(Eular[i --], res);
        while(j<l) add(Eular[j ++], res);
        while(j>l) add(Eular[-- j], res);
        if(p) add(p, res);
        ans[id] = res;
        if(p) add(p, res);
    }
    rep(i,1,m) print(ans[i]);
}
```