数据结构

线段树

```
// 区间加线段树
struct Tag{
    int lz;
    Tag(){}
    Tag(int lz):lz(lz){}
    void init(){lz=0;}
    bool empty(){return lz==0;}
    Tag operator+(const Tag&a)const{
        Tag t{};
        t.lz=lz+a.lz;
        return t;
    }
};
struct Info{
    int sum;
    Info(){}
    Info(int sum):sum(sum){}
    bool valid(){
        // 二分时判断合法性
        return sum>0;
    }
    Info operator+(const Info&a)const{
        Info t{};
        t.sum=sum+a.sum;
        return t;
    }
};
Info merge(Info info,Tag tag,int len){
    Info t{};
    t.sum=info.sum+tag.lz*len;
    return t;
}
struct Segment{
    int l,r;
    Info info;
    Tag tag;
#define ls (p<<1)
#define rs ((p<<1)|1)
#define mid ((l+r)>>1)
}seg[4*N];
void pushtag(int p,Tag tag){
    seg[p].info=merge(seg[p].info,tag,seg[p].r-seg[p].l+1);
    seg[p].tag=seg[p].tag+tag;
}
void pushup(int p){seg[p].info=seg[ls].info+seg[rs].info;}
void pushdown(int p){
    int l=seg[p].l,r=seg[p].r;
    Tag tag=seg[p].tag;
```

```
    if(l==r || tag.empty())return;
    pushtag(ls,tag);
    pushtag(rs,tag);
    seg[p].tag.init();
}
void build(int p,int l,int r){
    seg[p].l=l,seg[p].r=r;
    seg[p].tag.init();
    if(l==r){
        seg[p].info=Info(0);
        return;
    }
    build(ls,l,mid);
    build(rs,mid+1,r);
    pushup(p);
}
// 区间修改
void upd(int p,int x,int y,Tag&tag){
    int l=seg[p].l,r=seg[p].r;
    if(x<=l && y>=r){
        pushtag(p,tag);
        return;
    }
    pushdown(p);
    if(x<=mid)upd(ls,x,y,tag);
    if(y>mid)upd(rs,x,y,tag);
    pushup(p);
}
// 单点修改
void upd(int p,int x,Info&info){
    int l=seg[p].l,r=seg[p].r;
    if(l==r){
        seg[p].info=info;
        return;
    }
    pushdown(p);
    if(x<=mid)upd(ls,x,info);
    else upd(rs,x,info);
    pushup(p);
}
Info query(int p,int x,int y){
    int l=seg[p].l,r=seg[p].r;
    if(x<=l && y>=r){
        return seg[p].info;
    }
    pushdown(p);
    if(y<=mid)return query(ls,x,y);
    if(x>mid)return query(rs,x,y);
    return query(ls,x,y)+query(rs,x,y);
}
int q_first(int p,int x,int y){
    if(!seg[p].info.valid() || x>y)return -1;
    int l=seg[p].l,r=seg[p].r;
    if(l==r)return l;
    if(y<=mid)return q_first(ls,x,y);
```

```
    if(x>mid)return q_first(rs,x,y);
    int t=q_first(ls,x,y);
    if(t!=-1)return t;
    return q_first(rs,x,y);
}
```

势能线段树

```
// 区间整除/区间加/max/min/sum
struct Segment{
    int l,r;
    int mn,mx,sum;
    int add;
}seg[4*N];
void pushtag(int p,int x){
    int l=seg[p].l,r=seg[p].r;
    seg[p].mx+=x;
    seg[p].mn+=x;
    seg[p].sum+=(r-l+1)*x;
    seg[p].add+=x;
}
int cal(int x,int z){
    if(x>=0)return x/z;
    return -(-x+z-1)/z;
}
void dfs(int p,int z){
    if(cal(seg[p].mx,z)-seg[p].mx == cal(seg[p].mn,z)-seg[p].mn){
        pushtag(p,cal(seg[p].mx,z)-seg[p].mx);
        return;
    }
    pushdown(p);
    dfs(ls,z);
    dfs(rs,z);
    pushup(p);
}
void dfs_sqrt(int p){
    if((int)sqrt(seg[p].mx)-seg[p].mx == (int)sqrt(seg[p].mn)-seg[p].mn){
        pushtag(p,(int)sqrt(seg[p].mx)-seg[p].mx);
        return;
    }
    pushdown(p);
    dfs_sqrt(ls);
    dfs_sqrt(rs);
    pushup(p);
}
void div(int p,int x,int y,int z){
    int l=seg[p].l,r=seg[p].r;
    if(x<=l && y>=r){
        dfs(p,z);
        return;
    }
    pushdown(p);
    if(x<=mid)div(ls,x,y,z);
    if(y>mid)div(rs,x,y,z);
    pushup(p);
```

```
}
```

线段树二分

```
int ask(int p,int x,int y,int z){       //线段树上二分
    if(x==y)return 1;
    int l=seg[p].l,r=seg[p].r;
    if(x>mid)return ask(rs,x,y,z);      //全在左边
    if(y<=mid)return ask(ls,x,y,z);     //全在右边
    int x1=q(ls,x,mid),x2=q(rs,mid+1,y);        //左右都有，按 x,(l+r)/2,y 为分界点查询
    if(x1%z!=0&&x2%z!=0)return 0;
    if(x1%z==0&&x2%z==0)return 1;
    if(x1%z==0)return ask(rs,mid+1,y,z);        //向右查找
    return ask(ls,x,mid,z);         //向左查找
}
```

线段树分治

```
const int N=2e5+5;
#define ls (p<<1)
#define rs ((p<<1)|1)
#define mid ((l+r)>>1)
vector<pair<int,int>>vec[4*N];
stack<pair<int,int>>st;
int n,m,k;
struct Dsu{         // 可撤销并查集
    int f[2*N],siz[2*N];
    void init(){
        for(int i=1;i<=2*n;i++){
            f[i]=i;
            siz[i]=1;
        }
    }
    int find(int x){    // 不能路径压缩
        if(x==f[x])return x;
        return find(f[x]);
    }
    bool merge(int x,int y){
        x=find(x),y=find(y);
        if(x==y)return false;
        if(siz[x]>siz[y])swap(x,y);
        siz[y]+=siz[x];
        f[x]=y;
        st.emplace(x,y);
        return true;
    }
    bool check(int x){
        int u=find(x),v=find(x+n);
        return u==v;
    }
    void del(){
        auto [x,y]=st.top();
        st.pop();
```

```
        siz[y]-=siz[x];
        f[x]=x;
    }
}dsu;
// 把修改挂在线段树上
void ins(int p,int l,int r,int u,int v,int x,int y){
    if(x<=l && y>=r){
        vec[p].emplace_back(u,v);
        return;
    }
    if(x<=mid)ins(ls,l,mid,u,v,x,y);
    if(y>mid)ins(rs,mid+1,r,u,v,x,y);
}
// dfs线段树
void cal(int p,int l,int r){
    if(l>r)return;       // 注意 q=0 的情况
    int cnt=0,ok=1;
    for(auto [x,y]:vec[p]){
        if(dsu.merge(x,y+n))cnt++;
        if(dsu.merge(x+n,y))cnt++;
        if(dsu.check(x)||dsu.check(y)){
            ok=0;
            break;
        }
    }
    if(ok){
        if(l==r)cout<<"Yes"<<endl;
        else{
            cal(ls,l,mid);
            cal(rs,mid+1,r);
        }
    }
    else{
        for(int i=l;i<=r;i++)cout<<"No"<<endl;
    }
    for(int i=1;i<=cnt;i++)dsu.del();
}
void solve(){
    cin>>n>>m>>k;
    dsu.init();
    while(m--){
        int x,y,l,r;
        cin>>x>>y>>l>>r;
        l++;
        ins(1,1,k,x,y,l,r);
    }
    cal(1,1,k);
}
```

Segment Tree Beats

弱化版

```
// 区间取 max，区间求和,区间求 min
// O(nlogn)
struct Segment{
    int l,r,sum;
    int mn,sec,cnt;
    int lz;
#define ls (p<<1)
#define rs ((p<<1)|1)
#define mid ((l+r)>>1)
}seg[4*N];
void pushup(int p){
    seg[p].sum=seg[ls].sum+seg[rs].sum;
    if(seg[ls].mn<seg[rs].mn){
        seg[p].mn=seg[ls].mn;
        seg[p].cnt=seg[ls].cnt;
        seg[p].sec=min(seg[ls].sec,seg[rs].mn);
    }
    else if(seg[ls].mn>seg[rs].mn){
        seg[p].mn=seg[rs].mn;
        seg[p].cnt=seg[rs].cnt;
        seg[p].sec=min(seg[ls].mn,seg[rs].sec);
    }
    else{
        seg[p].mn=seg[ls].mn;
        seg[p].cnt=seg[ls].cnt+seg[rs].cnt;
        seg[p].sec=min(seg[ls].sec,seg[rs].sec);
    }
}
void push(int p,int lz){
    seg[p].sum+=lz*seg[p].cnt;
    seg[p].mn+=lz;
    seg[p].lz+=lz;
}
void pushdown(int p){
    int mn=min(seg[ls].mn,seg[rs].mn);
    int lz=seg[p].lz;
    if(lz==0)return;
    if(seg[ls].mn==mn)push(ls,lz);
    if(seg[rs].mn==mn)push(rs,lz);
    seg[p].lz=0;
}
void build(int p,int l,int r){
    seg[p].l=l,seg[p].r=r;
    seg[p].lz=0;
    if(l==r){
        seg[p].cnt=1;
        seg[p].mn=seg[p].sum=0;
        seg[p].sec=1e18;
        return;
    }
    build(ls,l,mid);
    build(rs,mid+1,r);
    pushup(p);
```

```
}
void upd_max(int p,int x,int y,int k){
    if(x>y)return;
    int l=seg[p].l,r=seg[p].r;
    if(k<=seg[p].mn)return;
    if(x<=l && y>=r && k<seg[p].sec){
        seg[p].sum+=(k-seg[p].mn)*seg[p].cnt;
        seg[p].lz+=(k-seg[p].mn);
        seg[p].mn=k;
        return;
    }
    pushdown(p);
    if(x<=mid)upd_max(ls,x,y,k);
    if(y>mid)upd_max(rs,x,y,k);
    pushup(p);
}
int q_sum(int p,int x,int y){
    int l=seg[p].l,r=seg[p].r;
    if(x<=l && y>=r)return seg[p].sum;
    pushdown(p);
    int ans=0;
    if(x<=mid)ans+=q_sum(ls,x,y);
    if(y>mid)ans+=q_sum(rs,x,y);
    return ans;
}
int q_min(int p,int x,int y){
    int l=seg[p].l,r=seg[p].r;
    if(x<=l && y>=r)return seg[p].mn;
    pushdown(p);
    int ans=1e18;
    if(x<=mid)ans=min(ans,q_min(ls,x,y));
    if(y>mid)ans=min(ans,q_min(rs,x,y));
    return ans;
}
```

完全版

```
// 区间加，区间取 min ，区间求和，区间求 max，区间求历史 max
// O(nlog^2n)
struct SegmentTreeBeats{
    int l,r,sum;
    int mx,mx_pre,sec,cnt;      // 最大 / 历史最大 / 严格次大 / 最大出现次数
    int lz1,lz2,lz1_pre,lz2_pre;    // 最大/次大的lazytag
#define ls (p<<1)
#define rs ((p<<1)|1)
#define mid ((l+r)>>1)
}seg[4*N];
void pushup(int p){
    seg[p].sum=seg[ls].sum+seg[rs].sum;
    seg[p].mx_pre=max(seg[ls].mx_pre,seg[rs].mx_pre);
    if(seg[ls].mx>seg[rs].mx){
        seg[p].mx=seg[ls].mx;
        seg[p].cnt=seg[ls].cnt;
        seg[p].sec=max(seg[ls].sec,seg[rs].mx);
    }
```

```
    else if(seg[ls].mx<seg[rs].mx){
        seg[p].mx=seg[rs].mx;
        seg[p].cnt=seg[rs].cnt;
        seg[p].sec=max(seg[rs].sec,seg[ls].mx);
    }
    else{
        seg[p].mx=seg[ls].mx;
        seg[p].cnt=seg[ls].cnt+seg[rs].cnt;
        seg[p].sec=max(seg[ls].sec,seg[rs].sec);
    }
}
void push(int p,int lz1,int lz2,int lz1_pre,int lz2_pre){
    int l=seg[p].l,r=seg[p].r;
    seg[p].sum+=lz1*seg[p].cnt;
    seg[p].sum+=lz2*(r-l+1-seg[p].cnt);

    seg[p].mx_pre=max(seg[p].mx_pre,seg[p].mx+lz1_pre);
    seg[p].mx+=lz1;

    seg[p].lz2_pre=max(seg[p].lz2_pre,seg[p].lz2+lz2_pre);
    seg[p].lz2+=lz2;
    seg[p].sec+=lz2;

    seg[p].lz1_pre=max(seg[p].lz1_pre,seg[p].lz1+lz1_pre);
    seg[p].lz1+=lz1;
}
void pushdown(int p){
    int mx=max(seg[ls].mx,seg[rs].mx);
    int
lz1=seg[p].lz1,lz2=seg[p].lz2,lz1_pre=seg[p].lz1_pre,lz2_pre=seg[p].lz2_pre;
    if(lz1==0 && lz2==0)return;
    if(seg[ls].mx==mx)push(ls,lz1,lz2,lz1_pre,lz2_pre);
    else push(ls,lz2,lz2,lz2_pre,lz2_pre);
    if(seg[rs].mx==mx)push(rs,lz1,lz2,lz1_pre,lz2_pre);
    else push(rs,lz2,lz2,lz2_pre,lz2_pre);
    seg[p].lz1=seg[p].lz1_pre=seg[p].lz2=seg[p].lz2_pre=0;
}

int a[N];
void build(int p,int l,int r){
    seg[p].l=l,seg[p].r=r;
    seg[p].lz1=seg[p].lz1_pre=0;
    seg[p].lz2=seg[p].lz2_pre=0;
    if(l==r){
        seg[p].mx=seg[p].mx_pre=seg[p].sum=a[l];
        seg[p].cnt=1;
        seg[p].sec=-1e18;
        return;
    }
    build(ls,l,mid);
    build(rs,mid+1,r);
    pushup(p);
}
void upd_add(int p,int x,int y,int k){
    int l=seg[p].l,r=seg[p].r;
```

```
    if(x<=l && y>=r){
        seg[p].sum+=k*(r-l+1);
        seg[p].mx+=k;seg[p].mx_pre=max(seg[p].mx_pre,seg[p].mx);
        seg[p].lz1+=k,seg[p].lz1_pre=max(seg[p].lz1_pre,seg[p].lz1);
        if(seg[p].sec!=-1e18){
            seg[p].sec+=k;
            seg[p].lz2+=k,seg[p].lz2_pre=max(seg[p].lz2,seg[p].lz2_pre);
        }
        return;
    }
    pushdown(p);
    if(x<=mid)upd_add(ls,x,y,k);
    if(y>mid)upd_add(rs,x,y,k);
    pushup(p);
}
void upd_min(int p,int x,int y,int k){      // 区间取min
    int l=seg[p].l,r=seg[p].r;
    if(k>=seg[p].mx)return;
    if(x<=l && y>=r && k>seg[p].sec){
        seg[p].sum-=(seg[p].mx-k)*seg[p].cnt;
        seg[p].lz1-=(seg[p].mx-k);
        seg[p].mx=k;
        return;
    }
    pushdown(p);
    if(x<=mid)upd_min(ls,x,y,k);
    if(y>mid)upd_min(rs,x,y,k);
    pushup(p);
}
int q_sum(int p,int x,int y){
    int l=seg[p].l,r=seg[p].r;
    if(x<=l && y>=r)return seg[p].sum;
    pushdown(p);
    int ans=0;
    if(x<=mid)ans+=q_sum(ls,x,y);
    if(y>mid)ans+=q_sum(rs,x,y);
    return ans;
}
int q_mx(int p,int x,int y){        // 区间最值
    int l=seg[p].l,r=seg[p].r;
    if(x<=l && y>=r)return seg[p].mx;
    pushdown(p);
    int ans=-1e18;
    if(x<=mid)ans=max(ans,q_mx(ls,x,y));
    if(y>mid)ans=max(ans,q_mx(rs,x,y));
    return ans;
}
int q_premx(int p,int x,int y){     // 区间历史最值
    int l=seg[p].l,r=seg[p].r;
    if(x<=l && y>=r)return seg[p].mx_pre;
    pushdown(p);
    int ans=-1e18;
    if(x<=mid)ans=max(ans,q_premx(ls,x,y));
    if(y>mid)ans=max(ans,q_premx(rs,x,y));
    return ans;
```

```
}
```

扫描线

```
const int N=1e6+5;
int val[N];     //离散化点对应的值
struct Seg{
    int l,r;
    int cnt,len;
    #define ls (p<<1)
    #define rs ((p<<1)|1)
    #define mid ((l+r)>>1)
}seg[4*N];
void build(int p,int l,int r){
    seg[p].l=l;
    seg[p].r=r;
    seg[p].cnt=0;
    seg[p].len=0;
    if(l==r)return;
    build(ls,l,mid);
    build(rs,mid+1,r);
}
void pushup(int p){
    int l=seg[p].l,r=seg[p].r;
    if(seg[p].cnt)seg[p].len=val[r+1]-val[l];
    else if(l==r)seg[p].len=0;
    else seg[p].len=seg[ls].len+seg[rs].len;
}
void upd(int p,int x,int y,int k){
    int l=seg[p].l,r=seg[p].r;
    if(x<=l&&y>=r){
        seg[p].cnt+=k;
        pushup(p);
        return;
    }
    if(x<=mid)upd(ls,x,y,k);
    if(y>mid)upd(rs,x,y,k);
    pushup(p);
}
struct Edge{    //扫描线
    int x,y1,y2,k;
    bool operator<(const Edge&a)const{return x<a.x;}
};
vector<int>s;       //离散化
int pos(int x){return lower_bound(s.begin(),s.end(),x)-s.begin()+1;}
void solve(){
    int n;
    cin>>n;
    vector<Edge>a;
    for(int i=1;i<=n;i++){
        int x1,x2,y1,y2;
        cin>>x1>>y1>>x2>>y2;
        a.push_back({x1,y1,y2,1});
        a.push_back({x2,y1,y2,-1});
        s.push_back(y1);s.push_back(y2);
```

```
    }
    sort(a.begin(),a.end());
    sort(s.begin(),s.end());
    s.erase(unique(s.begin(),s.end()),s.end());
    for(int i=1;i<=s.size();i++)val[i]=s[i-1];
    build(1,1,s.size());
    int ans=0,lst=0;
    for(auto i:a){
        int x=i.x,y1=i.y1,y2=i.y2,k=i.k;
        ans+=(x-lst)*seg[1].len;
        upd(1,pos(y1),pos(y2)-1,k);
        lst=x;
    }
    cout<<ans<<endl;
}
```

李超线段树

```
#define int long long
const int N=1e5+5;
#define double long double
struct node{
    int l,r;
    double k,b;
    #define mid ((l+r)>>1)
    #define ls (p<<1)
    #define rs ((p<<1)|1)
}seg[4*N];
double cal(int x,double k,double b){
    return k*x+b;
}
void build(int p,int l,int r){
    seg[p].l=l,seg[p].r=r;
    seg[p].k=0,seg[p].b=0;
    if(l==r)return;
    build(ls,l,mid);
    build(rs,mid+1,r);
}

void upd(int p,double k,double b){          // 插入直线
    int l=seg[p].l,r=seg[p].r;
    double k0=seg[p].k,b0=seg[p].b;
    if((cal(l,k,b)>cal(l,k0,b0))&&(cal(r,k,b)>cal(r,k0,b0))){
        seg[p].k=k,seg[p].b=b;
        return;
    }
    if(cal(l,k,b)<=cal(l,k0,b0)&&cal(r,k,b)<=cal(r,k0,b0))return;
    if(cal(mid,k,b)>cal(mid,k0,b0)){        // 当前直线在中点处更优
        swap(seg[p].k,k);
        swap(seg[p].b,b);
        k0=seg[p].k;
        b0=seg[p].b;
    }
    if(l==r)return;
    if(cal(l,k,b)>cal(l,k0,b0))upd(ls,k,b);
```

```
    if(cal(r,k,b)>cal(r,k0,b0))upd(rs,k,b);
}
void ins(int p,double k,double b,int x,int y){      // 插入线段
    int l=seg[p].l,r=seg[p].r;
    if(x<=l && y>=r){
        upd(p,k,b);
        return;
    }
    if(x<=mid)ins(ls,k,b,x,y);
    if(y>mid)ins(rs,k,b,x,y);
}
double q(int p,int x){
    int l=seg[p].l,r=seg[p].r;
    double ans=cal(x,seg[p].k,seg[p].b);
    if(l==r)return ans;
    if(x<=mid)ans=max(ans,q(ls,x));
    else ans=max(ans,q(rs,x));
    return ans;
}
```

线段树合并

```
const int N=1e5+5;
#define mid ((l+r)>>1)
int X[N],Y[N],Z[N],rt[N],num=0;
int ls[60*N],rs[60*N],cnt[60*N],pos[60*N];
void pushup(int p){
    cnt[p]=cnt[ls[p]]+cnt[rs[p]];
    pos[p]=min(pos[ls[p]],pos[rs[p]]);
}
void upd(int &p,int l,int r,int x,int k){
    if(!p)p=++num;
    if(l==r){
        cnt[p]+=k;
        pos[p]=l;
        return;
    }
    if(x<=mid)upd(ls[p],l,mid,x,k);
    else upd(rs[p],mid+1,r,x,k);
    pushup(p);
}
int merge(int x,int y,int l,int r){
    if(!x || !y)return x+y;
    if(l==r){
        cnt[x]+=cnt[y];
        pos[x]=l;
        return x;
    }
    ls[x]=merge(ls[x],ls[y],l,mid);
    rs[x]=merge(rs[x],rs[y],mid+1,r);
    pushup(x);
    return x;
}
int ans[N],mx;
void dfs(int x,int ff){
```

```
    for(int i=head[x];i;i=edge[i].next){
        int y=edge[i].to;
        if(y==ff)continue;
        dfs(y,x);
        rt[x]=merge(rt[x],rt[y],1,mx);
    }
    if(cnt[rt[x]])ans[x]=pos[rt[x]];
}
void solve(){
    int m;
    cin>>n>>m;
    t=log2(n);
    for(int i=1;i<n;i++){
        int u,v;
        cin>>u>>v;
        add(u,v);
        add(v,u);
    }
    init();     //lca
    for(int i=1;i<=m;i++){
        cin>>X[i]>>Y[i]>>Z[i];
        mx=max(mx,Z[i]);
    }
    for(int i=1;i<=m;i++){
        upd(rt[X[i]],1,mx,Z[i],1);
        upd(rt[Y[i]],1,mx,Z[i],1);
        upd(rt[lca(X[i],Y[i])],1,mx,Z[i],-1);
        if(f[lca(X[i],Y[i])][0])
        upd(rt[f[lca(X[i],Y[i])][0]],1,mx,Z[i],-1);
    }
    dfs(1,1);
    for(int i=1;i<=n;i++){
        cout<<ans[i]<<endl;
    }
    return;
}
```

线段树空间回收

```
int tot=0,cnt=0,rubs[N];
void Delete(int &now){
    ls[now]=0;
    rs[now]=0;
    sum[now]=0;
    rubs[++tot]=now;
    now=0;
}
int New(){
    if(tot) return rubs[tot--];
    return ++cnt;
}
```

主席树

```
const int N=2e5+5;
int cnt,root[N];
struct Seg{
    int l,r,sum;
}seg[40*N];
void ins(int l,int r,int pre,int &now,int p){
    seg[++cnt]=seg[pre];
    now=cnt;
    seg[now].sum++;
    if(l==r)return;
    int mid=(l+r)>>1;
    if(p<=mid)ins(l,mid,seg[pre].l,seg[now].l,p);
    else ins(mid+1,r,seg[pre].r,seg[now].r,p);
}
int query(int l,int r,int L,int R,int x){       //区间第k小
    if(l==r)return l;
    int mid=(l+r)>>1;
    int tmp=seg[seg[R].l].sum-seg[seg[L].l].sum;
    if(x<=tmp)return query(l,mid,seg[L].l,seg[R].l,x);
    return query(mid+1,r,seg[L].r,seg[R].r,x-tmp);
}
int n,m,a[N];
void solve(){
    cin>>n>>m;
    for(int i=1;i<=n;i++)cin>>a[i];
    for(int i=1;i<=n;i++){
        ins(1,n,root[i-1],root[i],a[i]);
    }
    while(m--){
        int l,r,k;
        cin>>l>>r>>k;
        int num=query(1,n,root[l-1],root[r],k);
        cout<<num<<endl;
    }
}
```

二维数点

```
const int N=2e6+5,M=(1ll<<32)+5;      //M为值域
int n,m,cnt,root[N];
struct Seg{
    int l,r,sum;
}seg[32*N];
void ins(int l,int r,int pre,int &now,int p,int val){
    seg[++cnt]=seg[pre];
    now=cnt;
    seg[now].sum+=val;
    if(l==r)return;
    int mid=(l+r)>>1;
    if(p<=mid)ins(l,mid,seg[pre].l,seg[now].l,p,val);
    else ins(mid+1,r,seg[pre].r,seg[now].r,p,val);
}
```

```
int query(int L,int R,int l,int r,int x,int y){
    if(x<=l&&y>=r)return seg[R].sum-seg[L].sum;
    int ans=0,mid=(l+r)/2;
    if(x<=mid)ans+=query(seg[L].l,seg[R].l,l,mid,x,y);
    if(y>mid)ans+=query(seg[L].r,seg[R].r,mid+1,r,x,y);
    return ans;
}
struct node{    //离线下来加点的过程
    int x,y,val;
    bool operator<(const node&a)const{
        if(x==a.x)return y<a.y;
        return x<a.x;
    }
}a[N];
int b[N];   //用于二分查找x位置
void solve(){
    cin>>n>>m;
    for(int i=1;i<=n;i++){
        cin>>a[i].x>>a[i].y>>a[i].val;
    }
    sort(a+1,a+1+n);
    for(int i=1;i<=n;i++){
        b[i]=a[i].x;
        ins(1,M,root[i-1],root[i],a[i].y,a[i].val);
    }
    while(m--){
        int x1,y1,x2,y2;
        cin>>x1>>y1>>x2>>y2;
        int id1=lower_bound(b+1,b+n+1,x1)-b-1;
        int id2=upper_bound(b+1,b+n+1,x2)-b-1;
        if(id1>=id2)cout<<0<<endl;
        else cout<<query(root[id1],root[id2],1,M,y1,y2)<<endl;
    }
}
```

平衡树

fhq Treap（单点）

```
const int N=2e5+5;
struct node{
    int l,r;
    int val,key;
    int siz;
}tr[N];
mt19937_64 Rnd(random_device{}());
int id,rt;
inline int newnode(int val){
    tr[++id].val=val;
    tr[id].key=Rnd();
    tr[id].siz=1;
    return id;
}
inline void update(int p){tr[p].siz=tr[tr[p].l].siz+tr[tr[p].r].siz+1;}
void split(int p,int val,int&x,int&y){  //按值分裂，使小于等于val的数位于x上
```

```
    if(p==0){
        x=y=0;
        return;
    }
    if(tr[p].val<=val){
        x=p;
        split(tr[p].r,val,tr[x].r,y);
    }
    else{
        y=p;
        split(tr[p].l,val,x,tr[y].l);
    }
    update(p);
}
void split_siz(int p,int siz,int&x,int&y){      //按大小分裂,使x的大小为siz
    if(!p){
        x=y=0;
        return;
    }
    if(tr[tr[p].l].siz+1<=siz){
        x=p;
        split_siz(tr[p].r,siz-tr[tr[p].l].siz-1,tr[x].r,y);
    }
    else{
        y=p;
        split_siz(tr[p].l,siz,x,tr[y].l);
    }
    update(p);
}
int merge(int x,int y){
    if(!x||!y)return x+y;
    if(tr[x].key>tr[y].key){
        tr[x].r=merge(tr[x].r,y);
        update(x);
        return x;
    }
    else{
        tr[y].l=merge(x,tr[y].l);
        update(y);
        return y;
    }
}
int x,y,z;
void ins(int val){      //插入
    split(rt,val,x,y);
    rt=merge(merge(x,newnode(val)),y);
}
void del(int val){      //删除
    split(rt,val,x,z);
    split(x,val-1,x,y);
    y=merge(tr[y].l,tr[y].r);       //只删除一个值
    rt=merge(merge(x,y),z);
}
int get_rank(int val){      //查询排名
    split(rt,val-1,x,y);
```

```
    int ans=tr[x].siz+1;
    rt=merge(x,y);
    return ans;
}
int get_kth(int k){        //第k小，相同值重复计数
    int p=rt;
    while(p){
        if(tr[tr[p].l].siz+1==k)break;
        if(tr[tr[p].l].siz>=k){
            p=tr[p].l;
        }
        else{
            k-=tr[tr[p].l].siz+1;
            p=tr[p].r;
        }
    }
    return tr[p].val;
}
int pre(int val){        //前驱
    split(rt,val-1,x,y);
    int p=x;
    while(tr[p].r)p=tr[p].r;
    int ans=tr[p].val;
    rt=merge(x,y);
    return ans;
}
int nxt(int val){        //后继
    split(rt,val,x,y);
    int p=y;
    while(tr[p].l)p=tr[p].l;
    int ans=tr[p].val;
    rt=merge(x,y);
    return ans;
}
```

文艺平衡树（区间）

```
const int N=2e5+5;
struct node{
    int l,r;
    int val,key;
    int siz;
    int rev;
}tr[N];
mt19937_64 Rnd(random_device{}());
int id,root;
inline int newnode(int val){
    tr[++id].val=val;
    tr[id].key=Rnd();
    tr[id].siz=1;
    return id;
}
inline void update(int p){tr[p].siz=tr[tr[p].l].siz+tr[tr[p].r].siz+1;}
inline void pushdown(int p){        //懒标记
    swap(tr[p].l,tr[p].r);
```

```
    tr[tr[p].l].rev^=1;
    tr[tr[p].r].rev^=1;
    tr[p].rev=0;
}
void split(int p,int siz,int&x,int&y){      //按大小分裂,使x的大小为siz
    if(!p){
        x=y=0;
        return;
    }
    if(tr[p].rev)pushdown(p);
    if(tr[tr[p].l].siz+1<=siz){
        x=p;
        split(tr[p].r,siz-tr[tr[p].l].siz-1,tr[x].r,y);
    }
    else{
        y=p;
        split(tr[p].l,siz,x,tr[y].l);
    }
    update(p);
}
int merge(int x,int y){
    if(!x||!y)return x+y;
    if(tr[x].key>tr[y].key){
        if(tr[x].rev)pushdown(x);
        tr[x].r=merge(tr[x].r,y);
        update(x);
        return x;
    }
    else{
        if(tr[y].rev)pushdown(y);
        tr[y].l=merge(x,tr[y].l);
        update(y);
        return y;
    }
}
void ins(int x){        //有序插入
    root=merge(root,newnode(x));
}
void rev(int l,int r){      //区间翻转
    int x,y,z;
    split(root,l-1,x,y);
    split(y,r-l+1,y,z);
    tr[y].rev^=1;
    root=merge(merge(x,y),z);
}
void midord(int p){         //中序遍历
    if(!p)return;
    if(tr[p].rev)pushdown(p);
    midord(tr[p].l);
    cout<<tr[p].val<<' ';
    midord(tr[p].r);
}
int query(int p,int x){      //二分第k小
    if(tr[tr[p].l].siz==x-1)return tr[p].val;
    else if(tr[tr[p].l].siz>=x)return query(tr[p].l,x);
```

```
    else return query(tr[p].r,x-tr[tr[p].l].siz-1);
}
```

树状数组

一维

```
int a[N],t[N],n;
void add(int x,int y){
    for(int i=x;i<N;i+=i&-i)
        t[i]+=y;
}
int query(int x){
    int ans=0;
    for(int i=x;i>0;i-=i&-i)ans+=t[i];
    return ans;
}
```

一维区间修改+查询

```
int t1[N],t2[N];
void add(int*t,int x,int y){
    while(x<N){
        t[x]+=y;
        x+=x&-x;
    }
}
void add(int l,int r,int x){    // 区间修改
    add(t1,l,x);
    add(t1,r+1,-x);
    add(t2,l,x*(l-1));
    add(t2,r+1,-x*r);
}
void add(int x,int y){      // 差分数组上单点修改，用于初始化
    add(t1,x,y);
    add(t2,x,(x-1)*y);
}
int query(int*t,int x){
    int ans=0;
    while(x){
        ans+=t[x];
        x-=x&-x;
    }
    return ans;
}
int query(int l,int r){     // 区间查询
    int ans=0;
    ans+=r*query(t1,r);
    ans-=(l-1)*query(t1,l-1);
    ans+=query(t2,l-1);
    ans-=query(t2,r);
    return ans;
}
```

二维

```
const int N=5e3+5;
int t[N][N];
void add(int x,int y,int z){        //单点修改
    while(x<N){
        int tmp=y;
        while(tmp<N){
            t[x][tmp]+=z;
            tmp+=tmp&-tmp;
        }
        x+=x&-x;
    }
}
int query(int x,int y){          //矩阵查询
    int ans=0;
    while(x){
        int tmp=y;
        while(tmp){
            ans+=t[x][tmp];
            tmp-=tmp&-tmp;
        }
        x-=x&-x;
    }
    return ans;
}
```

二维矩阵修改+查询

```
const int N=2050;
struct node{
    int t[N][N];
    void add(int x,int y,int z){
        while(x<N){
            int tmp=y;
            while(tmp<N){
                t[x][tmp]+=z;
                tmp+=tmp&-tmp;
            }
            x+=x&-x;
        }
    }
    int query(int x,int y){          //矩阵查询
        int ans=0;
        while(x){
            int tmp=y;
            while(tmp){
                ans+=t[x][tmp];
                tmp-=(tmp&-tmp);
            }
            x-=x&-x;
        }
        return ans;
    }
```

```
}d,di,dj,dij;   // 差分
void Add(int x,int y,int z){    // 差分数组单点修改
    d.add(x,y,z);
    di.add(x,y,z*x);
    dj.add(x,y,z*y);
    dij.add(x,y,z*x*y);
}
int q(int x,int y){     // 前缀查询
    int ans=0;
    ans+=d.query(x,y)*(x*y+x+y+1);
    ans-=di.query(x,y)*(y+1);
    ans-=dj.query(x,y)*(x+1);
    ans+=dij.query(x,y);
    return ans;
}
void add(int x1,int y1,int x2,int y2,int z){
    Add(x1,y1,z);
    Add(x1,y2+1,-z);
    Add(x2+1,y1,-z);
    Add(x2+1,y2+1,z);
}
int query(int x1,int y1,int x2,int y2){
    int ans=0;
    ans+=q(x2,y2);
    ans-=q(x1-1,y2);
    ans-=q(x2,y1-1);
    ans+=q(x1-1,y1-1);
    return ans;
}
void solve(){
    int n,m,op;
    cin>>n>>m;
    while(cin>>op){
        int x1,x2,y1,y2,z;
        if(op==1){
            cin>>x1>>y1>>x2>>y2>>z;
            add(x1,y1,x2,y2,z);
        }
        else{
            cin>>x1>>y1>>x2>>y2;
            cout<<query(x1,y1,x2,y2)<<endl;
        }
    }
}
```

笛卡尔树

```
// 笛卡尔树，按下标是一个二叉搜索树，按 val 是一个小根堆
struct Descartes{
    int fa,l,r;
}dk[N];
int a[N],rt,n;
void build(){       // 对排列 a 构建笛卡尔树（小根堆）
    deque<int>dq;
    for(int i=1;i<=n;i++){
```

```
        int lst=0;
        while(!dq.empty() && a[dq.back()]>a[i]){        // < 小根堆    |    > 大根
堆
            lst=dq.back(),dq.pop_back();
        }
        if(lst)dk[i].l=lst,dk[lst].fa=i;
        if(dq.empty())rt=i;
        else dk[dq.back()].r=i,dk[i].fa=dq.back();
        dq.push_back(i);
    }
}
```

序列分块

```
const int N=2e5+5;
int L[N],R[N],pos[N];
int siz,id;
int a[N],sum[N],lazy[N];
void add(int l,int r,int x){
    int pos1=pos[l],pos2=pos[r];
    if(pos1==pos2){
        for(int i=l;i<=r;i++)a[i]+=x;
        sum[pos1]+=x*(r-l+1);
        return;
    }
    for(int i=pos1+1;i<pos2;i++)lazy[i]+=x;
    for(int i=l;i<=R[pos1];i++)a[i]+=x;
    for(int i=L[pos2];i<=r;i++)a[i]+=x;
    sum[pos1]+=x*(R[pos1]-l+1);
    sum[pos2]+=x*(r-L[pos2]+1);
}
int query(int l,int r){
    int pos1=pos[l],pos2=pos[r],ans=0;
    if(pos1==pos2){
        for(int i=l;i<=r;i++)ans+=a[i];
        ans+=lazy[pos1]*(r-l+1);
        return ans;
    }
    for(int i=pos1+1;i<pos2;i++){
        ans+=sum[i]+lazy[i]*(R[i]-L[i]+1);
    }
    for(int i=l;i<=R[pos1];i++)ans+=a[i];
    for(int i=L[pos2];i<=r;i++)ans+=a[i];
    ans+=lazy[pos1]*(R[pos1]-l+1);
    ans+=lazy[pos2]*(r-L[pos2]+1);
    return ans;
}
void solve(){
    int n,m;
    cin>>n>>m;
    siz=sqrt(n),id=siz;
    for(int i=1;i<=id;i++){
        L[i]=(i-1)*siz+1;
        R[i]=i*siz;
    }
```

```
    if(R[id]<n){
        id++;
        L[id]=R[id-1]+1;
        R[id]=n;
    }
    for(int i=1;i<=n;i++)cin>>a[i];
    for(int i=1;i<=id;i++){
        for(int j=L[i];j<=R[i];j++){
            pos[j]=i;
            sum[i]+=a[j];
        }
    }
}
```

树分块（随机撒点）

```
mt19937 Rnd(random_device{}());
bitset<30005> bit[N];   // 路径信息
int t[N];       // 上方最近的关键点编号
void solve(){
    int n,m;
    cin>>n>>m;
    // 随机撒 k 个点作为关键点
    int k=sqrt(n)-1;    // 视情况直接将根作为关键点
    vis[1]=1;
    while(k--){
        int p=Rnd()%n+1;
        while(vis[p])p=Rnd()%n+1;
        vis[p]=1;
    }
    for(int i=1;i<=n;i++)cin>>a[i];
    for(int i=1;i<n;i++){
        int u,v;
        cin>>u>>v;
        G[u].push_back(v);
        G[v].push_back(u);
    }
    // 树剖 lca
    dfs1(1);
    dfs2(1,1);
    for(int i=1;i<=n;i++){
        if(vis[i]){
            // 对于每个关键点，暴力往上跳，以及路径上信息
            int now=i;
            while(true){
                bit[i][a[now]]=true;
                now=f[now];
                if(now==0)break;
                // 记录最近的关键点编号
                if(vis[now]){
                    bit[i][a[now]]=true;    // 视情况统计上面关键点的信息
                    t[i]=now;
                    break;
                }
            }
```

```
        }
    }
    bitset<30005>now;
    while(m--){
        int u,v;
        cin>>u>>v;
        int lc=lca(u,v);
        // 先向上跳到不超过 lca 的第一个关键点
        // 统计单点信息
        while(u && !vis[u] && dep[u]>=dep[lc]){
            now[a[u]]=true;
            u=f[u];
        }
        // 如果上面的关键点不超过 lca，往上跳关键点
        // 统计这段路径已经预处理好的信息
        while(t[u] && dep[t[u]]>=dep[lc]){
            now|=bit[u];
            u=t[u];
        }
        // 从最接近 lca 的关键点往上跳
        // 统计单点信息
        while(u && dep[u]>=dep[lc]){
            now[a[u]]=true;
            u=f[u];
        }
        // 统计路径另一端
        while(v && !vis[v] && dep[v]>=dep[lc]){
            now[a[v]]=true;
            v=f[v];
        }
        while(t[v] && dep[t[v]]>=dep[lc]){
            now|=bit[v];
            v=t[v];
        }
        while(v && dep[v]>=dep[lc]){
            now[a[v]]=true;
            v=f[v];
        }
        int ans=now.count();
        cout<<ans<<endl;
    }
}
```

莫队

普通莫队

```
const int N=1e6+10;
int cnt[N],pos[N];
int ans[N],a[N];
int res=0;
void Add(int x){
    res-=cnt[a[x]]*cnt[a[x]];
    cnt[a[x]]++;
    res+=cnt[a[x]]*cnt[a[x]];
```

```
}
void Del(int x){
    res-=cnt[a[x]]*cnt[a[x]];
    cnt[a[x]]--;
    res+=cnt[a[x]]*cnt[a[x]];
}
struct node{
    int l,r,id;
}q[N];
void solve(){
    int n,m,k;
    cin>>n>>m>>k;
    int siz=sqrt(n);    // 块大小最佳为 n/sqrt(m)
    for(int i=1;i<=n;i++){
        cin>>a[i];
        pos[i]=i/siz;
    }
    for(int i=0;i<m;i++){
        cin>>q[i].l>>q[i].r;
        q[i].id=i;
    }
    sort(q,q+m,[&](node a,node b){
        if(pos[a.l]!=pos[b.l])return a.l<b.l;
        if(pos[a.l]&1)return a.r<b.r;
        return a.r>b.r;
    });
    int l=1,r=0;
    for(int i=0;i<m;i++){
        while(r<q[i].r)Add(++r);
        while(l>q[i].l)Add(--l);
        while(r>q[i].r)Del(r--);
        while(l<q[i].l)Del(l++);
        ans[q[i].id]=res;
    }
    for(int i=0;i<m;i++)cout<<ans[i]<<endl;
}
```

回滚莫队

```
#define int long long
const int N=2e5+5;
int n,m;
int a[N],L[N],R[N],pos[N];
int ans[N];
int cnt[N],tmp[N];

struct node{
    int l,r,id;
}q[N];

void solve(){
    cin>>n>>m;
    int siz=sqrt(n),tot=n/siz;
    for(int i=1;i<=tot;i++){
        L[i]=(i-1)*siz+1;
```

```
        R[i]=i*siz;
    }
    if(R[tot]<n){
        tot++;
        L[tot]=R[tot-1]+1;
        R[tot]=n;
    }
    for(int i=1;i<=tot;i++){
        for(int j=L[i];j<=R[i];j++){
            pos[j]=i;
        }
    }
    vector<int>v;   // 离散化
    for(int i=1;i<=n;i++){
        cin>>a[i];
        v.push_back(a[i]);
    }
    sort(v.begin(),v.end());
    v.erase(unique(v.begin(),v.end()),v.end());
    for(int i=1;i<=n;i++)a[i]=lower_bound(v.begin(),v.end(),a[i])-v.begin()+1;
    for(int i=1;i<=m;i++){
        cin>>q[i].l>>q[i].r;
        q[i].id=i;
    }
    sort(q+1,q+m+1,[&](node aa,node bb){        // 不能奇偶优化
        if(pos[aa.l]!=pos[bb.l])return pos[aa.l]<pos[bb.l];
        return aa.r<bb.r;
    });
    int l=R[1]+1,r=R[1];
    int now=1,lst=0,mx=0;
    for(int i=1;i<=m;i++){
        if(pos[q[i].l]==pos[q[i].r]){          //块内直接暴力
            int res=0,mx=0;
            for(int j=q[i].l;j<=q[i].r;j++){
                int val=a[j];
                tmp[val]++;
                if(tmp[val]>mx){
                    mx=tmp[val];
                    res=val;
                }
                else if(tmp[val]==mx)
                    res=min(res,val);
            }
            for(int j=q[i].l;j<=q[i].r;j++){
                int val=a[j];
                tmp[val]--;
            }
            ans[q[i].id]=res;
            continue;
        }
        int bb=pos[q[i].l];
        if(bb!=now){         //进入新块
            while(r>R[now])cnt[a[r--]]--;    // 先滚回来
            lst=0,mx=0;       // 右边的答案
            now=bb;
```

```
            l=R[now]+1,r=R[now];    // 更新端点
        }
        while(r<q[i].r){      //右边add
            r++;
            int val=a[r];
            cnt[val]++;
            if(cnt[val]>mx){
                mx=cnt[val];
                lst=val;
            }
            else if(cnt[val]==mx){
                lst=min(lst,val);
            }
        }
        int res=lst,mx2=mx;          //左边临时答案
        while(l>q[i].l){            //左边add
            l--;
            int val=a[l];
            tmp[val]++;
            if(tmp[val]+cnt[val]>mx2){
                mx2=tmp[val]+cnt[val];
                res=val;
            }
            else if(tmp[val]+cnt[val]==mx2){
                res=min(res,val);
            }
        }
        while(l<R[now]+1){      //回滚
            tmp[a[l]]--;
            l++;
        }
        ans[q[i].id]=res;
    }
    for(int i=1;i<=m;i++){
        cout<<v[ans[i]-1]<<endl;
    }
}
```

带修莫队

```
// O(n^(3/5))
int pos[N],a[N],ans[N],cnt[N],sum=0;
struct Query{
    int l,r,tim,id;
    bool operator<(const Query&aa)const{
        if(pos[l]!=pos[aa.l])return l<aa.l;
        if(pos[r]!=pos[aa.r])return r<aa.r;
        return tim<aa.tim;
    }
}q[N];
struct Change{
    int pos,val;
}c[N];
void del(int x){
    cnt[a[x]]--;
```

```
    sum-=cnt[a[x]]==0;
}
void add(int x){
    cnt[a[x]]++;
    sum+=cnt[a[x]]==1;
}
void upd(int x,int now){
    if(q[now].l<=c[x].pos && q[now].r>=c[x].pos){    // 当前修改的位置位于查询区间内
        cnt[a[c[x].pos]]--;
        sum-=cnt[a[c[x].pos]]==0;
        cnt[c[x].val]++;
        sum+=cnt[c[x].val]==1;
    }
    swap(a[c[x].pos],c[x].val);
}
void solve(){
    int n,m;
    cin>>n>>m;
    int siz=(int)pow(n,2.0/3);
    for(int i=1;i<=n;i++){
        cin>>a[i];
        pos[i]=i/siz;
    }
    int cnt_q=0,cnt_c=0;    // 查询/修改的次数
    for(int i=1;i<=m;i++){
        string op;
        int x,y;
        cin>>op>>x>>y;
        if(op=="Q"){
            q[++cnt_q].tim=cnt_c;
            q[cnt_q].l=x,q[cnt_q].r=y;
            q[cnt_q].id=cnt_q;
        }
        else c[++cnt_c].pos=x,c[cnt_c].val=y;
    }
    sort(q+1,q+cnt_q+1);
    int l=1,r=0,now=0;
    for(int i=1;i<=cnt_q;i++){
        while(l<q[i].l)del(l++);
        while(l>q[i].l)add(--l);
        while(r<q[i].r)add(++r);
        while(r>q[i].r)del(r--);
        while(now<q[i].tim)upd(++now,i);
        while(now>q[i].tim)upd(now--,i);
        ans[q[i].id]=sum;
    }
    for(int i=1;i<=cnt_q;i++)cout<<ans[i]<<endl;
}
```

树上莫队（欧拉序）

```
int a[N],cnt[N];
int st[N],ed[N],c[N],now;
//  st,ed 表示欧拉序，c 表示欧拉序对应的点序号
vector<int>G[N];
```

```
int siz[N],top[N],son[N],f[N],dep[N];
void dfs1(int x){
    siz[x]=1,dep[x]=dep[f[x]]+1;
    st[x]=++now,c[now]=x;
    for(auto y:G[x]){
        if(y==f[x])continue;
        f[y]=x;
        dfs1(y);
        siz[x]+=siz[y];
        if(siz[y]>siz[son[x]])son[x]=y;
    }
    ed[x]=++now,c[now]=x;
}
void dfs2(int x,int id){
    top[x]=id;
    if(son[x])dfs2(son[x],id);
    for(auto y:G[x]){
        if(y==f[x]||y==son[x])continue;
        dfs2(y,y);
    }
}
int lca(int u,int v){
    while(top[u]!=top[v]){
        if(dep[top[u]]<dep[top[v]])v=f[top[v]];
        else u=f[top[u]];
    }
    return (dep[u]<dep[v])?u:v;
}

int pos[N],vis[N],ans[N],sum=0;
void add(int x){
    cnt[a[x]]++;
    if(cnt[a[x]]==1)sum++;
}
void del(int x){
    cnt[a[x]]--;
    if(cnt[a[x]]==0)sum--;
}
void cal(int x){    // 记个 vis 标记，每次访问的时候 ^1
    if(vis[x])del(x);
    else add(x);
    vis[x]^=1;
}
struct node{
    int l,r,fa,id;
}q[N];
void solve(){
    int n,m;
    cin>>n>>m;
    for(int i=1;i<=n;i++)cin>>a[i];
    for(int i=1;i<n;i++){
        int u,v;
        cin>>u>>v;
        G[u].push_back(v);
        G[v].push_back(u);
```

```
    }
    dfs1(1);
    dfs2(1,1);
    int B=sqrt(n);
    for(int i=1;i<=2*n;i++)pos[i]=i/B;       // 欧拉序，要 *2
    for(int i=1;i<=m;i++){
        int u,v;
        cin>>u>>v;
        if(st[u]>st[v])swap(u,v);
        int fa=lca(u,v);     // lca 单独计算贡献，如果已经被计算过，则不计算
        if(fa==u){
            q[i].l=st[u],q[i].r=st[v];
            q[i].fa=0,q[i].id=i;
        }
        else{
            q[i].l=ed[u],q[i].r=st[v];
            q[i].fa=fa,q[i].id=i;
        }
    }
    sort(q+1,q+m+1,[&](node a,node b){
        if(pos[a.l]!=pos[b.l])return a.l<b.l;
        if(pos[a.l]&1)return a.r<b.r;
        return a.r>b.r;
    });
    int l=1,r=0;
    for(int i=1;i<=m;i++){
        while(r<q[i].r)cal(c[++r]);
        while(l>q[i].l)cal(c[--l]);
        while(r>q[i].r)cal(c[r--]);
        while(l<q[i].l)cal(c[l++]);
        if(q[i].fa)cal(q[i].fa);
        ans[q[i].id]=sum;
        if(q[i].fa)cal(q[i].fa);
    }
    for(int i=1;i<=m;i++)cout<<ans[i]<<endl;
}
```

二次离线莫队

珂朵莉树（ODT）

```
struct ODT{
    int l,r,x;
    ODT(int l,int r,int x):l(l),r(r),x(x){}
    ODT(int l):l(l){}
    bool operator <(const ODT &a)const{
        return l<a.l;
    }
};
set<ODT>odt;
set<ODT>::iterator split(int pos){
    auto it=odt.lower_bound(ODT(pos));
    if(it!=odt.end()&&(it->l)==pos){
        return it;
    }
```

```
    it--;
    int l=it->l,r=it->r,x=it->x;
    odt.erase(it);
    odt.insert(ODT(l,pos-1,x));
    return odt.insert(ODT(pos,r,x)).first;
}
void merge(int l,int r,int x){     //区间赋值
    auto it2=split(r+1);
    auto it1=split(l);
    for(auto it=it1;it!=it2;it++){
        /*
        区间操作
        int x=it->x,ll=it->l,rr=it->r;
        ans+=x*(rr-ll+1);   //求区间和
        */
    }
    odt.erase(it1,it2);
    odt.insert(ODT(l,r,x));
}
```

树状数组套线段树

```
int rt[N],n,cnt;
struct Seg{
    int ls,rs;
    int mx;
}seg[100*N];
void ins(int&p,int x,int y,int l,int r){
    if(!p)p=++cnt;
    seg[p].mx=max(seg[p].mx,y);
    if(l==r)return;
    int mid=(l+r)>>1;
    if(x<=mid)ins(seg[p].ls,x,y,l,mid);
    else ins(seg[p].rs,x,y,mid+1,r);
}
int q(int p,int x,int y,int l,int r){
    if(p==0 || x>y)return 0;
    if(x<=l && y>=r)return seg[p].mx;
    int ans=0,mid=(l+r)>>1;
    if(x<=mid)ans=max(ans,q(seg[p].ls,x,y,l,mid));
    if(y>mid)ans=max(ans,q(seg[p].rs,x,y,mid+1,r));
    return ans;
}
void upd(int now,int x,int y){
    while(now<=n){
        ins(rt[now],x,y,1,n);
        now+=now&-now;
    }
}
int query(int now,int x){
    int ans=0;
    while(now){
        ans=max(ans,q(rt[now],1,x,1,n));
        now-=now&-now;
    }
```

```
    return ans;
}
//---------------------------
int add[N],del[N],cnt1,cnt2;
int query(int l,int r,int k){
    if(l==r)return l;
    int sum=0;
    for(int i=1;i<=cnt1;i++)sum+=seg[seg[add[i]].ls].sum;
    for(int i=1;i<=cnt2;i++)sum-=seg[seg[del[i]].ls].sum;
    if(sum>=k){
        for(int i=1;i<=cnt1;i++)add[i]=seg[add[i]].ls;
        for(int i=1;i<=cnt2;i++)del[i]=seg[del[i]].ls;
        return query(l,mid,k);
    }
    else{
        for(int i=1;i<=cnt1;i++)add[i]=seg[add[i]].rs;
        for(int i=1;i<=cnt2;i++)del[i]=seg[del[i]].rs;
        return query(mid+1,r,k-sum);
    }
}
int query_kth(int x,int y,int k){       // 动态区间第 k 小
    cnt1=0,cnt2=0;      // 把 log 个主席树存起来，同时进行二分
    while(y)add[++cnt1]=rt[y],y-=y&-y;
    x--;
    while(x)del[++cnt2]=rt[x],x-=x&-x;
    return query(1,M,k);
}
```

线段树套平衡树

```
const int N=5e4+5;
mt19937 Rnd(random_device{}());
struct Treap{       // 平衡树
    static int id;      // 要在后面初始化
    static struct node{
        int l,r;
        int val,key;
        int siz;
    }fhq[100*N];
    inline static int newnode(int val){
        id++;
        fhq[id].val=val;
        fhq[id].key=Rnd();
        fhq[id].siz=1;
        return id;
    }
    inline static void update(int p)
{fhq[p].siz=fhq[fhq[p].l].siz+fhq[fhq[p].r].siz+1;}
    int root;
    void clear(){root=0;}
    void split(int p,int val,int&x,int&y){  //按值分裂，使小于等于val的数位于x上
        if(p==0){
            x=y=0;
            return;
        }
```

```
        if(fhq[p].val<=val){
            x=p;
            split(fhq[p].r,val,fhq[x].r,y);
        }
        else{
            y=p;
            split(fhq[p].l,val,x,fhq[y].l);
        }
        update(p);
    }
    int merge(int x,int y){
        if(!x||!y)return x+y;
        if(fhq[x].key>fhq[y].key){
            fhq[x].r=merge(fhq[x].r,y);
            update(x);
            return x;
        }
        else{
            fhq[y].l=merge(x,fhq[y].l);
            update(y);
            return y;
        }
    }
    int x,y,z;
    void ins(int val){      //插入
        split(root,val,x,y);
        root=merge(merge(x,newnode(val)),y);
    }
    void del(int val){      //删除
        split(root,val,x,z);
        split(x,val-1,x,y);
        y=merge(fhq[y].l,fhq[y].r);       //只删除一个值
        root=merge(merge(x,y),z);
    }
    int getrank(int val){       //查询排名
        split(root,val-1,x,y);
        int ans=fhq[x].siz+1;
        root=merge(x,y);
        return ans;
    }
    int getnum(int k){       //第k小，相同值重复计数
        int p=root;
        while(p){
            if(fhq[fhq[p].l].siz+1==k)break;
            if(fhq[fhq[p].l].siz>=k){
                p=fhq[p].l;
            }
            else{
                k-=fhq[fhq[p].l].siz+1;
                p=fhq[p].r;
            }
        }
        return fhq[p].val;
    }
    int pre(int val){       //前驱
```

```
        split(root,val-1,x,y);
        if(!x)return INT_MIN;
        int p=x;
        while(fhq[p].r)p=fhq[p].r;
        int ans=fhq[p].val;
        root=merge(x,y);
        return ans;
    }
    int nxt(int val){       //后继
        split(root,val,x,y);
        if(!y)return INT_MAX;
        int p=y;
        while(fhq[p].l)p=fhq[p].l;
        int ans=fhq[p].val;
        root=merge(x,y);
        return ans;
    }
    #define ls (p<<1)
    #define rs ((p<<1)|1)
    #define mid ((l+r)>>1)
}seg[4*N];      // 线段树，用来划分区间
int Treap::id=0;
Treap::node Treap::fhq[100*N];
int a[N];
void build(int p,int l,int r){
    seg[p].clear();
    for(int i=l;i<=r;i++)seg[p].ins(a[i]);
    if(l==r)return;
    build(ls,l,mid);
    build(rs,mid+1,r);
}
int get_rank(int p,int l,int r,int x,int y,int k){  //返回比k小的数的个数
    if(x<=l&&y>=r)return seg[p].getrank(k)-1;
    int ans=0;
    if(x<=mid)ans+=get_rank(ls,l,mid,x,y,k);
    if(y>mid)ans+=get_rank(rs,mid+1,r,x,y,k);
    return ans;
}
int get_num(int p,int x,int y,int k){          // 查询k在区间的排名
    int L=0,R=INT_MAX;
    while(L<R){
        int md=(L+R+1)>>1;
        if(get_rank(1,1,n,x,y,md)<k)L=md;
        else R=md-1;
    }
    return L;
}
void upd(int p,int l,int r,int x,int y){        // 单点修改，记得在主函数里修改
    seg[p].del(a[x]);
    seg[p].ins(y);
    if(l==r)return;
    if(x<=mid)upd(ls,l,mid,x,y);
    else upd(rs,mid+1,r,x,y);
}
int get_pre(int p,int l,int r,int x,int y,int k){       // 查询某个值在区间的前驱
```

```
    if(x<=l&&y>=r)return seg[p].pre(k);
    int ans=INT_MIN+1;
    if(x<=mid)ans=max(ans,get_pre(ls,l,mid,x,y,k));
    if(y>mid)ans=max(ans,get_pre(rs,mid+1,r,x,y,k));
    return ans;
}
int get_nxt(int p,int l,int r,int x,int y,int k){       // 查询某个值在区间的后继
    if(x<=l&&y>=r)return seg[p].nxt(k);
    int ans=INT_MAX;
    if(x<=mid)ans=min(ans,get_nxt(ls,l,mid,x,y,k));
    if(y>mid)ans=min(ans,get_nxt(rs,mid+1,r,x,y,k));
    return ans;
}
```

树上DS

树的重心

```
vector<int>rt;      //找树的两个重心
void getrt(int x,int f){
    siz[x]=1;
    maxp[x]=0;
    for(int i=head[x];i;i=edge[i].next){
        int y=edge[i].to;
        if(y==f)continue;
        getrt(y,x);
        siz[x]+=siz[y];
        maxp[x]=max(maxp[x],siz[y]);
    }
    maxp[x]=max(maxp[x],n-siz[x]);
    if(maxp[x]<=n/2)rt.push_back(x);
}
```

树哈希

```
//有根树直接哈希，无根树求出两个重心，取两个哈希值的较小值作为树哈希
const int mod=998244353,base=13331;
int siz[N];
int HashTree(int x,int f){
    vector<int>v;
    siz[x]=1;
    for(int i=head[x];i;i=edge[i].next){
        int y=edge[i].to;
        if(y==f)continue;
        v.push_back(HashTree(y,x));
        siz[x]+=siz[y];
    }
    if(siz[x]==1)return 1;
    int ans=0;
    sort(v.begin(),v.end());
    for(auto i:v)ans=(ans*base+i)%mod;
    return ans*siz[x];
}
```

点分治

```
const int N=1e5+5,K=1e7+5;
int n,m,sum,vis[N],siz[N],maxp[N],rt;
void getrt(int x,int f){        //找重心
    siz[x]=1;
    maxp[x]=0;
    for(int i=head[x];i;i=edge[i].next){
        int y=edge[i].to;
        if(vis[y]||y==f)continue;
        getrt(y,x);
        siz[x]+=siz[y];
        if(siz[y]>maxp[x])maxp[x]=siz[y];
    }
    maxp[x]=max(maxp[x],sum-siz[x]);
    if(maxp[x]<maxp[rt])rt=x;
}
int ans[N],q[N],dis[N],tmp[K],judge[K],cnt;
void getdis(int x,int f){       //计算距离
    tmp[++cnt]=dis[x];
    for(int i=head[x];i;i=edge[i].next){
        int y=edge[i].to;
        if(vis[y]||y==f)continue;
        dis[y]=dis[x]+edge[i].val;
        getdis(y,x);
    }
}
void cal(int x){        //统计答案
    queue<int>qq;
    for(int i=head[x];i;i=edge[i].next){
        int y=edge[i].to;
        if(vis[y])continue;
        dis[y]=edge[i].val;
        cnt=0;
        getdis(y,y);
        for(int i=1;i<=cnt;i++){
            for(int j=1;j<=m;j++){
                if(q[j]>=tmp[i])ans[j]|=judge[-tmp[i]+q[j]];
            }
        }
        for(int i=1;i<=cnt;i++){
            judge[tmp[i]]=1;
            qq.push(tmp[i]);
        }
    }
    while(!qq.empty()){
        judge[qq.front()]=0;
        qq.pop();
    }
}
void dvd(int x){    //分治
    vis[x]=1;
    judge[0]=1;
    cal(x);
    for(int i=head[x];i;i=edge[i].next){
```

```
        int y=edge[i].to;
        if(vis[y])continue;
        rt=0;
        sum=siz[y];
        getrt(y,y);
        getrt(rt,rt);
        dvd(rt);
    }
}
void solve(){
    cin>>n>>m;
    for(int i=1;i<n;i++){
        int u,v,x;
        cin>>u>>v>>x;
        add(u,v,x);
        add(v,u,x);
    }
    for(int i=1;i<=m;i++)cin>>q[i];

    maxp[0]=sum=n;
    getrt(1,1);
    getrt(rt,rt);
    dvd(rt);

    for(int i=1;i<=m;i++)cout<<ans[i]<<endl;
    return;
}
```

点分树

```
const int N=2e5+5;

vector<int>G1[N],G2[N];
int siz[N],vis[N],maxp[N];
int sum,rt;

void get_rt(int x,int f){       // 点分治部分
    siz[x]=1;
    maxp[x]=0;
    for(auto y:G1[x]){
        if(y==f || vis[y])continue;
        get_rt(y,x);
        siz[x]+=siz[y];
        maxp[x]=max(maxp[x],siz[y]);
    }
    maxp[x]=max(maxp[x],sum-siz[x]);
    if(maxp[x]<maxp[rt])rt=x;
}
void dvd(int x){
    vis[x]=1;
    for(auto y:G1[x]){
        if(vis[y])continue;
        rt=0,sum=siz[y];
        get_rt(y,y);
        get_rt(rt,rt);
```

```
        G2[x].push_back(rt);        // 建树
        dvd(rt);
    }
}

int fa[N],dep[N],son[N],top[N];         // 处理点分树lca
void dfs1(int x){
    siz[x]=1,dep[x]=dep[fa[x]]+1;
    for(auto y:G1[x]){
        if(y==fa[x])continue;
        fa[y]=x;
        dfs1(y);
        siz[x]+=siz[y];
        if(siz[y]>siz[son[x]])son[x]=y;
    }
}
void dfs2(int x,int id){
    top[x]=id;
    if(son[x])dfs2(son[x],id);
    for(auto y:G1[x]){
        if(y==fa[x] || y==son[x])continue;
        dfs2(y,y);
    }
}
int lca(int u,int v){
    while(top[u]!=top[v]){
        if(dep[top[u]]<dep[top[v]])v=fa[top[v]];
        else u=fa[top[u]];
    }
    return dep[u]<dep[v]?u:v;
}
int dis(int u,int v){
    return dep[u]+dep[v]-2*dep[lca(u,v)];
}
int f[N],n,m;
void dfs(int x){
    for(auto y:G2[x]){
        f[y]=x;
        dfs(y);
    }
}

struct SegmentTree{         // 维护点分树信息
    int ls,rs;
    int sum1,sum2;
#define mid ((l+r)>>1)
}seg[100*N];
int rot[N],a[N],cnt;
void ins1(int&p,int pre,int l,int r,int x,int y){
    if(!p)p=++cnt;
    seg[p]=seg[pre];
    seg[p].sum1+=y;
    if(l==r)return;
    if(x<=mid)ins1(seg[p].ls,seg[pre].ls,l,mid,x,y);
    else ins1(seg[p].rs,seg[pre].rs,mid+1,r,x,y);
```

```
}
void ins2(int&p,int pre,int l,int r,int x,int y){
    if(!p)p=++cnt;
    seg[p]=seg[pre];
    seg[p].sum2+=y;
    if(l==r)return;
    if(x<=mid)ins2(seg[p].ls,seg[pre].ls,l,mid,x,y);
    else ins2(seg[p].rs,seg[pre].rs,mid+1,r,x,y);
}
int q_sum1(int p,int l,int r,int x,int y){
    if(!p)return 0;
    if(x<=l && y>=r)return seg[p].sum1;
    int ans=0;
    if(x<=mid)ans+=q_sum1(seg[p].ls,l,mid,x,y);
    if(y>mid)ans+=q_sum1(seg[p].rs,mid+1,r,x,y);
    return ans;
}
int q_sum2(int p,int l,int r,int x,int y){
    if(!p)return 0;
    if(x<=l && y>=r)return seg[p].sum2;
    int ans=0;
    if(x<=mid)ans+=q_sum2(seg[p].ls,l,mid,x,y);
    if(y>mid)ans+=q_sum2(seg[p].rs,mid+1,r,x,y);
    return ans;
}

void upd(int x,int y){
    int now=x;
    while(now){
        int d=dis(x,now);
        ins1(rot[now],rot[now],0,n,d,y);
        if(f[now]){
            d=dis(x,f[now]);
            ins2(rot[now],rot[now],0,n,d,y);
        }
        now=f[now];
    }
}
int query(int x,int y){
    int ans=0,now=x;
    while(now){
        int d=dis(x,now);
        if(d<=y)ans+=q_sum1(rot[now],0,n,0,y-d);
        if(f[now]){
            d=dis(x,f[now]);
            if(d<=y)ans-=q_sum2(rot[now],0,n,0,y-d);
        }
        now=f[now];
    }
    return ans;
}
void solve(){
    cin>>n>>m;
    for(int i=1;i<=n;i++)cin>>a[i];
    for(int i=1;i<n;i++){
```

```
        int u,v;
        cin>>u>>v;
        G1[u].push_back(v);
        G1[v].push_back(u);
    }
    dfs1(1);
    dfs2(1,1);

    rt=0,maxp[0]=sum=n;
    get_rt(1,1);
    get_rt(rt,rt);

    int root=rt;
    dvd(rt);
    dfs(root);

    for(int i=1;i<=n;i++){
        upd(i,a[i]);
    }
    int ans=0;
    while(m--){
        int op,x,y;
        cin>>op>>x>>y;
        x^=ans,y^=ans;
        if(op==0){
            ans=query(x,y);
            cout<<ans<<endl;
        }
        else{
            upd(x,y-a[x]);
            a[x]=y;
        }
    }
}
```

树链剖分

```
int dep[N],son[N],a[N],f[N],top[N],siz[N],id[N],nw[N],cnttt;
//a-原树的值，id-线段树下标，nw-线段树单点值，cnttt-线段树建点
//线段树建点时，注意用nw[]，别写成a[]
void dfs1(int x){
    dep[x]=dep[f[x]]+1;
    siz[x]=1;
    for(int i=head[x];i;i=edge[i].next){
        int y=edge[i].to;
        if(y==f[x])continue;
        f[y]=x;
        dfs1(y);
        siz[x]+=siz[y];
        if(siz[y]>siz[son[x]])son[x]=y;
    }
}
void dfs2(int x,int pos){
    top[x]=pos;
    id[x]=++cnttt;
```

```
    nw[cnttt]=a[x];
    if(son[x])dfs2(son[x],pos);
    for(int i=head[x];i;i=edge[i].next){
        int y=edge[i].to;
        if(y==f[x]||y==son[x])continue;
        dfs2(y,y);
    }
}
int q_tree(int x){       //查询子树
    return q(1,id[x],id[x]+siz[x]-1);
}
int q_link(int x,int y){     //查询链
    int ans=0;
    while(top[x]!=top[y]){
        if(dep[top[x]]<dep[top[y]])swap(x,y);
        ans+=q(1,id[top[x]],id[x]);
        x=f[top[x]];
    }
    ans+=q(1,min(id[x],id[y]),max(id[x],id[y]));
    return ans;
}
void upd_tree(int x,int z){      //修改子树
    upd(1,id[x],id[x]+siz[x]-1,z);
}
void upd_link(int x,int y,int z){    //修改链
    while(top[x]!=top[y]){
        if(dep[top[x]]<dep[top[y]])swap(x,y);
        upd(1,id[top[x]],id[x],z);
        x=f[top[x]];
    }
    upd(1,min(id[x],id[y]),max(id[x],id[y]),z);
}
```

动态 dp

```
vector<int>G[N];
int siz[N],son[N],dep[N],a[N],top[N],ed[N],fa[N],dfn[N],cntt;
int id[N];     //   dfn = i 对应的原树的点
void dfs1(int x){
    siz[x]=1;
    dep[x]=dep[fa[x]]+1;
    for(auto y:G[x]){
        if(y==fa[x])continue;
        fa[y]=x;
        dfs1(y);
        siz[x]+=siz[y];
        if(siz[y]>siz[son[x]])son[x]=y;
    }
}
void dfs2(int x,int pos){
    top[x]=pos;
    dfn[x]=++cntt;
    id[cntt]=x;
    if(son[x])dfs2(son[x],pos);
    else ed[pos]=dfn[x];     // 标记链的终点的 dfn 序
```

```
    for(auto y:G[x]){
        if(y==fa[x] || y==son[x])continue;
        dfs2(y,y);
    }
}

int f[N][2],g[N][2];
void dfs3(int x){       // 处理初始 dp 值
    f[x][1]=g[x][1]=a[x];
    if(son[x])dfs3(son[x]);
    if(son[x]){
        f[x][0]+=max(f[son[x]][0],f[son[x]][1]);
        f[x][1]+=f[son[x]][0];
    }
    for(auto y:G[x]){
        if(y==fa[x] || y==son[x])continue;
        dfs3(y);
        f[x][0]+=max(f[y][0],f[y][1]);
        f[x][1]+=f[y][0];
        g[x][0]+=max(f[y][0],f[y][1]);
        g[x][1]+=f[y][0];
    }
}
struct Matrix{
    int val[2][2];
    Matrix(){memset(val,0,sizeof(val));}
    Matrix operator*(const Matrix&o)const{
        Matrix ans;
        for(int i=0;i<2;i++){
            for(int j=0;j<2;j++){
                for(int k=0;k<2;k++){       // (max,+) 卷积
                    ans.val[i][j]=max(ans.val[i][j],val[i][k]+o.val[k][j]);
                }
            }
        }
        return ans;
    }
};
struct Segment{
    int l,r;
    Matrix dp;
#define ls (p<<1)
#define rs ((p<<1)|1)
#define mid ((l+r)>>1)
}seg[4*N];
void pushup(int p){
    seg[p].dp=seg[ls].dp*seg[rs].dp;
}
void build(int p,int l,int r){
    seg[p].l=l,seg[p].r=r;
    if(l==r){
        int pos=id[l];
        seg[p].dp.val[0][0]=g[pos][0],seg[p].dp.val[0][1]=g[pos][0];
        seg[p].dp.val[1][0]=g[pos][1],seg[p].dp.val[1][1]=-1e18;
        return;
```

```
    }
    build(ls,l,mid);
    build(rs,mid+1,r);
    pushup(p);
}
Matrix query(int p,int x,int y){
    int l=seg[p].l,r=seg[p].r;
    if(x<=l && y>=r)return seg[p].dp;
    if(y<=mid)return query(ls,x,y);
    if(x>mid)return query(rs,x,y);
    return query(ls,x,y)*query(rs,x,y);
}
void add(int p,int x,int y,int z){
    int l=seg[p].l,r=seg[p].r;
    if(l==r){
        seg[p].dp.val[0][0]+=y,seg[p].dp.val[0][1]+=y;
        seg[p].dp.val[1][0]+=z;
        return;
    }
    if(x<=mid)add(ls,x,y,z);
    else add(rs,x,y,z);
    pushup(p);
}
void upd(int x,int y){      // 树上单点修改
    int cost1=0,cost2=y-a[x];
    a[x]=y;
    while(true){
        if(top[x]==1){      // 到达最顶端，不会对父亲产生贡献
            add(1,dfn[x],cost1,cost2);
            return;
        }
        // 减去原来的贡献，再加上修改后的贡献
        Matrix pre=query(1,dfn[top[x]],ed[top[x]]);
        add(1,dfn[x],cost1,cost2);
        Matrix now=query(1,dfn[top[x]],ed[top[x]]);
        cost1=max(now.val[0][0],now.val[1][0])-max(pre.val[0][0],pre.val[1][0]);
        cost2=now.val[0][0]-pre.val[0][0];
        x=fa[top[x]];
    }
}
void solve(){
    int n,m;
    cin>>n>>m;
    for(int i=1;i<=n;i++)cin>>a[i];
    for(int i=1;i<n;i++){
        int u,v;
        cin>>u>>v;
        G[u].push_back(v);
        G[v].push_back(u);
    }
    dfs1(1);
    dfs2(1,1);
    dfs3(1);
    build(1,1,n);
    for(int i=1;i<=m;i++){
```

```
        int x,y;
        cin>>x>>y;
        upd(x,y);
        Matrix ans=query(1,dfn[1],ed[1]);
        cout<<max(ans.val[0][0],ans.val[1][1])<<endl;
    }
}
```

启发式合并

```
int f[N],siz[N],son[N],a[N],ans[N],sum,cnt[N];
void dfs1(int x){
    siz[x]=1;
    for(int i=head[x];i;i=edge[i].next){
        int y=edge[i].to;
        if(y==f[x])continue;
        f[y]=x;
        dfs1(y);
        siz[x]+=siz[y];
        if(siz[y]>siz[son[x]])son[x]=y;
    }
}
int flag,maxn;
void cal(int x,int k){
    for(int i=head[x];i;i=edge[i].next){
        int y=edge[i].to;
        if(y==f[x]||y==flag)continue;
        cal(y,k);
    }
    cnt[a[x]]+=k;
    if(cnt[a[x]]==maxn){
        sum+=a[x];
    }
    else if(cnt[a[x]]>maxn){
        maxn=cnt[a[x]];
        sum=a[x];
    }
}
void dfs2(int x,bool ok){
    for(int i=head[x];i;i=edge[i].next){
        int y=edge[i].to;
        if(y==f[x]||y==son[x])continue;
        dfs2(y,0);
    }
    if(son[x])dfs2(son[x],1);
    flag=son[x];
    cal(x,1);
    ans[x]=sum;
    if(!ok){
        flag=0;
        cal(x,-1);
        sum=0;
        maxn=0;
    }
}
```

LCT

```
//  每一条实链用一颗以深度为关键字的 splay 维护
//  只合并不分离时，可以用并查集代替 findroot 判断连通性
struct LCT{
    #define ls (tr[p].son[0])
    #define rs (tr[p].son[1])
    struct node{
        int son[2],fa;
        int rev,sum,val;
    }tr[N];
    inline int pos(int p){
        return tr[tr[p].fa].son[1]==p;
    }
    inline bool isroot(int p){
        return tr[tr[p].fa].son[0]!=p&&tr[tr[p].fa].son[1]!=p;
    }
    // 更新信息
    inline void pushup(int p){
        tr[p].sum=(tr[ls].sum^tr[rs].sum^tr[p].val);
    }
    inline void rev(int p){     // 操作
        tr[p].rev^=1;
        swap(ls,rs);
    }
    inline void pushdown(int p){
        if(tr[p].rev==0)return;
        if(ls)rev(ls);
        if(rs)rev(rs);
        tr[p].rev=0;
    }
    inline void pushall(int p){
        if(!isroot(p))pushall(tr[p].fa);
        pushdown(p);
    }
    inline void rotate(int p){
        int fa=tr[p].fa,gfa=tr[fa].fa,x=pos(p);
        if(!isroot(fa))tr[gfa].son[pos(fa)]=p;
        tr[p].fa=gfa;
        tr[fa].son[x]=tr[p].son[x^1];
        tr[tr[fa].son[x]].fa=fa;
        tr[p].son[x^1]=fa;
        tr[fa].fa=p;
        pushup(fa);
        pushup(p);
    }
    inline void splay(int p){
        pushall(p);
        while(!isroot(p)){
            if(!isroot(tr[p].fa)){
                if(pos(tr[p].fa)==pos(p))rotate(tr[p].fa);
                else rotate(p);
            }
            rotate(p);
```

```
        }
    }
    inline void access(int p){      // 打通到原树根节点的实链
        for(int y=0;p;p=tr[p].fa){      // y为上一个splay的根
            splay(p);       // 伸展+连边
            rs=y;
            pushup(p);
            y=p;
        }
    }
    inline void makeroot(int p){
        access(p);
        splay(p);
        rev(p);
    }
    inline int findroot(int p){
        access(p);
        splay(p);
        pushdown(p);
        // 找当前实链上深度最小的点，就是根
        while(ls){
            pushdown(p);
            p=ls;
        }
        splay(p);
        return p;
    }
    inline void split(int x,int y){     // 拆出x和y间的路径
        makeroot(x);            // 换根+打通+伸展
        access(y);
        splay(y);
    }
    inline void link(int x,int y){      // 连一条边
        makeroot(x);
        if(findroot(y)==x)return;
        tr[x].fa=y;
    }
    inline void cut(int x,int y){       // 断开两点
        makeroot(x);
        if(findroot(y)!=x||tr[y].fa!=x||tr[y].son[0])return;
        tr[y].fa=tr[x].son[0]=0;
        pushup(y);
    }
    inline void upd(int x,int y){      // 单点修改值
        splay(x);
        tr[x].val=y;
        pushup(x);
    }
    inline int q_sum(int x,int y){      // 查询一条链的和
        split(x,y);
        return tr[y].sum;
    }
}lct;
```

Trie

Trie

```
const int N=1e6+5;
int trie[N][300];
int cnt[N];
int id=0;
string value[N];
void clear(){
    for(int i=0;i<=id;i++){
        cnt[i]=0;
        for(int j=0;j<300;j++)trie[i][j]=0;
    }
    id=0;
}
void ins(string s){
    int p=0;
    for(int i=0;i<s.length();i++){
        int x=s[i];
        if(trie[p][x]==0)trie[p][x]=++id;
        p=trie[p][x];
    }
    cnt[p]++;
    value[p]=s;
}
int query(string s){
    int p=0;
    for(int i=0;i<s.length();i++){
        int x=s[i];
        if(trie[p][x]==0)return 0;
        p=trie[p][x];
    }
    return cnt[p];
}
void dfs(int p){
    if(p!=0&&cnt[p])cout<<value[p]<<endl;
    for(int i=0;i<300;i++){
        if(trie[p][i])dfs(trie[p][i]);
    }
}
```

01 Trie

```
const int N=5e5+5;
int id;
int cnt[N*32][2];       // 维护每个节点的出现次数
int trie[N*32][2];
void clear(){
    for(int i=0;i<=id;i++)trie[i][0]=trie[i][1]=cnt[i][0]=cnt[i][1]=0;
    id=0;
}
void ins(int x){    // 插入
    int p=0;
```

```
    for(int i=32;i>=0;i--){
        int v=((x>>i)&1);
        if(!trie[p][v])trie[p][v]=++id;
        cnt[p][v]++;
        p=trie[p][v];
    }
}
void del(int x){    // 删除
    int p=0;
    for(int i=32;i>=0;i--){
        int v=((x>>i)&1);
        cnt[p][v]--;
        p=trie[p][v];
    }
}
int query(int x){   //查询某个数 能异或出的最大值
    int p=0,ans=0;
    for(int i=32;i>=0;i--){
        int v=((x>>i)&1);
        if(trie[p][v^1]&&cnt[p][v^1]){
            p=trie[p][v^1];
            ans|=(1ll<<i);
        }
        else{
            p=trie[p][v];
        }
    }
    return ans;
}
// 查询 x 能异或出来的第 k 大值
int query(int p,int now,int x,int k){
    if(now==-1)return 0;
    int v=(x>>now)&1;
    int s=tr[tr[p].son[v^1]].cnt;
    if(s>=k)return (1ll<<now)+query(tr[p].son[v^1],now-1,x,k);
    else return query(tr[p].son[v],now-1,x,k-s);
}
```

可持久化01 Trie

```
int rt[N],id=0;
struct Trie{
    int son[2],cnt;
}trie[50*N];
void ins(int&p,int pre,int bit,int x){
    p=++id;
    trie[p]=trie[pre];
    trie[p].cnt++;
    if(bit<0)return;
    int v=(x>>bit)&1;
    ins(trie[p].son[v],trie[pre].son[v],bit-1,x);
}
int q(int x,int y,int bit,int z){       // 查询 z 能异或出的最大值
    if(bit<0)return 0;
    int v=(z>>bit)&1;
```

```
    if(trie[trie[y].son[v^1]].cnt-trie[trie[x].son[v^1]].cnt)
        return (1<<bit)+q(trie[x].son[v^1],trie[y].son[v^1],bit-1,z);
    return q(trie[x].son[v],trie[y].son[v],bit-1,z);
}
```

ST表

一维

```
const int N=1e6+5;
int st[N][25];
int Log[N];
void init(){
    Log[0]=-1;
    for(int i=1;i<N;i++){
        if(i&(i-1))Log[i]=Log[i-1];
        else Log[i]=Log[i-1]+1;
    }
    for(int j=1;j<=Log[N-5];j++){
        for(int i=1;(i+(1<<j)-1)<N;i++){
            st[i][j]=max(st[i][j-1],st[i+(1<<(j-1))][j-1]);
        }
    }
}
int q(int l,int r){
    int k=Log[r-l+1];
    // int k=__lg(r-l+1);
    return max(st[l][k],st[r-(1<<k)+1][k]);      //x+2^k-1=r
}
```

二维

```
const int N=2005;
int st[N][N][20];
int Log[N];
void init(){
    Log[0]=-1;
    for(int i=1;i<N;i++){
        if(i&(i-1))Log[i]=Log[i-1];
        else Log[i]=Log[i-1]+1;
    }
    for(int k=1;k<=Log[N-5];k++){
        for(int i=1;i+(1<<k)-1<N;i++){
            for(int j=1;j+(1<<k)-1<N;j++){
                int t1,t2,t3,t4;
                t1=st[i][j][k-1];
                t2=st[i+(1<<(k-1))][j][k-1];
                t3=st[i][j+(1<<(k-1))][k-1];
                t4=st[i+(1<<(k-1))][j+(1<<(k-1))][k-1];
                st[i][j][k]=max({t1,t2,t3,t4});
            }
        }
    }
}
```

```
int ask_max(int x,int y,int len){
    int k=Log[len];
    int t1,t2,t3,t4;
    t1=st[x][y][k];
    t2=st[x][y+len-(1<<k)][k];
    t3=st[x+len-(1<<k)][y][k];
    t4=st[x+len-(1<<k)][y+len-(1<<k)][k];
    return max({t1,t2,t3,t4});
}
void solve(){
    for(int i=1;i<=n;i++)
    for(int j=1;j<=m;j++){
        cin>>st[i][j][0];
    }
    init();
}
```

CDQ分治

```
struct node{
    int a,b,c;
    int cnt,ans;
    bool operator !=(const node&aa)const{
        return a!=aa.a||b!=aa.b||c!=aa.c;
    }
}s1[100005],s2[100005];     //去重前后的数组
bool cmpa(node a,node b){
    if(a.a==b.a){
        if(a.b==b.b)return a.c<b.c;
        return a.b<b.b;
    }
    return a.a<b.a;
}
bool cmpb(node a,node b){
    if(a.b==b.b)return a.c<b.c;
    return a.b<b.b;
}
int t[200005];  //权值树状数组，储存第三维c=i的元素个数，每次使用后清空
int ans[200005];
int n,k;
void add(int x,int k){
    while(x<200005){
        t[x]+=k;
        x+=(x&(-x));
    }
}
int ask(int x){
    int ans=0;
    while(x){
        ans+=t[x];
        x-=(x&(-x));
    }
    return ans;
}
void CDQ(int l,int r){
```

```
    if(l==r)return ;
    int mid=(l+r)/2;
    CDQ(l,mid);
    CDQ(mid+1,r);
    sort(s2+l,s2+mid+1,cmpb);        //按第二维排序，降维
    sort(s2+mid+1,s2+r+1,cmpb);
    int i=l,j;
    for(j=mid+1;j<=r;j++){
        while(i<=mid&&s2[i].b<=s2[j].b){        //如果第二维不下降
            add(s2[i].c,s2[i].cnt);
            i++;
        }
        s2[j].ans+=ask(s2[j].c);
    }
    for(int j=l;j<i;j++){
        add(s2[j].c,-s2[j].cnt);
    }
}
int main(){
    cin>>n>>k;
    for(int i=1;i<=n;i++){
        cin>>s1[i].a>>s1[i].b>>s1[i].c;
    }
    sort(s1+1,s1+n+1,cmpa);      //按第一维排序，降维
    int m=0,cnt=0;
    for(int i=1;i<=n;i++){
        cnt++;
        if(s1[i]!=s1[i+1]){
            s2[++m]=s1[i];
            s2[m].cnt=cnt;
            cnt=0;
        }
    }
    CDQ(1,m);
    for(int i=1;i<=m;i++)
        ans[s2[i].ans+s2[i].cnt-1]+=s2[i].cnt;
    for(int i=0;i<n;i++){
        cout<<ans[i]<<endl;
    }
    return 0;
}
```

可删堆

```
struct Heap{        // 可删堆
    priority_queue<int>a,b;
    inline void push(int x){a.push(x);}
    inline void pop(int x){b.push(x);}
    inline void pop(){
        while(!b.empty()&&a.top()==b.top())a.pop(),b.pop();
        a.pop();
    }
    inline int top(){
        while(!b.empty()&&a.top()==b.top())a.pop(),b.pop();
        return a.top();
```

```
    }
    inline bool empty(){
        while(!b.empty()&&a.top()==b.top())a.pop(),b.pop();
        return a.empty();
    }
};
```

左偏树（可并堆）

```
const int N=1e5+10;
struct node{
    int l,r,fa;
    int val,dis;
}ltt[N];
#define ls ltt[x].l
#define rs ltt[x].r
int merge(int x,int y){ //合并两个堆
    if(!x||!y)return x+y;
    if(ltt[x].val>ltt[y].val||(ltt[x].val==ltt[y].val&&x>y))swap(x,y);
    rs=merge(rs,y);
    ltt[rs].fa=x;
    if(ltt[ls].dis<ltt[rs].dis)swap(ls,rs);
    ltt[x].dis=ltt[rs].dis+1;
    return x;
}
void pop(int x){        //删除堆顶元素
    ltt[x].val=-1;
    ltt[ls].fa=ls;
    ltt[rs].fa=rs;
    ltt[x].fa=merge(ls,rs);
}
int find(int x){return ltt[x].fa==x?x:ltt[x].fa=find(ltt[x].fa);}   //并查集
void solve(){
    int n,m;
    cin>>n>>m;
    ltt[0].dis=-1;
    for(int i=1;i<=n;i++){
        cin>>ltt[i].val;
        ltt[i].fa=i;
    }
    while(m--){
        int op,x,y;
        cin>>op>>x;
        if(op==1){
            cin>>y;
            if(ltt[x].val==-1||ltt[y].val==-1)continue;
            x=find(x),y=find(y);
            if(x==y)continue;
            ltt[x].fa=ltt[y].fa=merge(x,y);
        }
        else{
            if(ltt[x].val==-1){
                cout<<-1<<endl;
                continue;
            }
```

```
            x=find(x);
            cout<<ltt[x].val<<endl;
            pop(x);
        }
    }
    return;
}
```

SOSdp

前缀和

```
for(int i=0;i<n;i++)
for(int j=0;j<(1<<n);j++)
    if((j>>i)&1) f[j]+=f[j-(1<<i)];
```

后缀和

```
for(int i=0;i<n;i++)
for(int j=0;j<(1<<n);j++)
    if(!((j>>i)&1)) f[j]+=f[j+(1<<i)];
```

前缀差分

```
for(int i=0;i<n;i++)
for(int j=0;j<(1<<n);j++)
    if((j>>i)&1) f[j]-=f[j-(1<<i)];
```

后缀差分

```
for(int i=0;i<n;i++)
for(int j=0;j<(1<<n);j++)
    if(!((j>>i)&1)) f[j]-=f[j+(1<<i)];
```