数据结构

平衡树（FHQ Treap）

区间翻转

文艺平衡树

```
#include<bits/stdc++.h>
using namespace std;

const int N=1e5+5;
```

```cpp
int n, m;

struct Node{
    int l, r;
    int v;
    int key, sz;
    int rev;

    #define ls tr[u].l
    #define rs tr[u].r
}tr[N];

int idx, root;

int add(int v){
    ++idx;
    tr[idx].v=v, tr[idx].key=rand(), tr[idx].sz=1;
    return idx;
}

void pushup(int u){
    tr[u].sz=tr[ls].sz+tr[rs].sz+1;
}

void pushdown(int u){
    if(tr[u].rev){
        swap(ls, rs);
        tr[ls].rev^=1, tr[rs].rev^=1;
        tr[u].rev=0;
    }
}

int merge(int x, int y){
    if(!x || !y) return x+y;
    if(tr[x].key>tr[y].key){
        pushdown(x);
        tr[x].r=merge(tr[x].r, y);
        pushup(x);
        return x;
    }
    else{
        pushdown(y);
        tr[y].l=merge(x, tr[y].l);
        pushup(y);
        return y;
    }
}

void split(int u, int k, int &x, int &y){
    if(!u) return x=y=0, void();
    pushdown(u);
    if(tr[ls].sz+1<=k)
        x=u, split(rs, k-tr[ls].sz-1, rs, y);
    else
        y=u, split(ls, k, x, ls);
    pushup(u);
}
```

```
void reverse(int l, int r){
    int x, y, z;
    split(root, l-1, x, y), split(y, r-l+1, y, z);
    tr[y].rev^=1;
    root=merge(x, merge(y, z));
}

void output(int u){
    if(!u) return;
    pushdown(u);
    output(ls); cout<<tr[u].v<<' '; output(rs);
}

int main(){
    srand(19260817);
    ios::sync_with_stdio(false);
    cin>>n>>m;
    for(int i=1; i<=n; i++) root=merge(root, add(i));

    while(m--){
        int l, r; cin>>l>>r;
        reverse(l, r);
    }

    output(root);

    return 0;
}
```

按值分裂

例题：普通平衡树

```
#include<bits/stdc++.h>
using namespace std;

const int N=1e5+5;

struct Node{
    int l, r;
    int key;
    int sz, v;

    #define ls tr[u].l
    #define rs tr[u].r
}tr[N];

int root, idx;

void pushup(int u){
    tr[u].sz=tr[ls].sz+tr[rs].sz+1;
}

int add(int v){
    ++idx;
```

```
    tr[idx].key=rand(), tr[idx].sz=1, tr[idx].v=v;
    return idx;
}

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);
        pushup(x);
        return x;
    }
    else{
        tr[y].l=merge(x, tr[y].l);
        pushup(y);
        return y;
    }
}

void split(int u, int val, int &x, int &y){
    if(!u) return x=y=0, void();
    if(tr[u].v<=val)
        x=u, split(rs, val, rs, y);
    else
        y=u, split(ls, val, x, ls);
    pushup(u);
}

void insert(int v){
    int x, y;
    split(root, v, x, y);
    root=merge(x, merge(add(v), y));
}

void remove(int v){
    int x, y, z;
    split(root, v, x, z);
    split(x, v-1, x, y);
    y=merge(tr[y].l, tr[y].r);
    root=merge(merge(x, y), z);
}

int val4rk(int v){
    int x, y;
    split(root, v-1, x, y);
    int res=tr[x].sz+1;
    root=merge(x, y);
    return res;
}

int rk4val(int k){
    int u=root;
    while(u){
        if(tr[ls].sz+1==k) return tr[u].v;
        else if(tr[ls].sz>=k) u=ls;
        else k-=tr[ls].sz+1, u=rs;
    }
    return -1;
}
```

```
int get_prev(int v){
    int x, y;
    split(root, v-1, x, y);
    int u=x;
    while(rs) u=rs;
    int res=tr[u].v;
    root=merge(x, y);
    return res;
}

int get_next(int v){
    int x, y;
    split(root, v, x, y);
    int u=y;
    while(ls) u=ls;
    int res=tr[u].v;
    root=merge(x, y);
    return res;
}

int main(){
    ios::sync_with_stdio(false);
    srand(131);
    int q; cin>>q;
    while(q--){
        int op, x; cin>>op>>x;
        if(op==1) insert(x);
        else if(op==2) remove(x);
        else if(op==3) cout<<val4rk(x)<<endl;
        else if(op==4) cout<<rk4val(x)<<endl;
        else if(op==5) cout<<get_prev(x)<<endl;
        else if(op==6) cout<<get_next(x)<<endl;
    }
    return 0;
}
```

字符串

Hash

```
const int N=1e6+5, B=131, P=1e9+7;

string s;
int n;

int f[N], p[N];

int get(int l, int r){
    return (f[r]-f[l-1]*p[r-l+1]%P+P)%P;
}

signed main(){
```

```
    cin>>s; int n=s.size();
    s=' '+s;
    p[0]=1;
    rep(i,1,n){
        f[i]=f[i-1]*131%P+s[i]-'a';
        p[i]=p[i-1]*131%P;
    }

    int m; cin>>m;
    while(m--){
        int l, r, ll, rr; read(l), read(r), read(ll), read(rr);
        if(get(l, r)==get(ll, rr)) puts("Yes");
        else puts("No");
    }

    return 0;
}
```

Rope

可以实现快速的插入、删除和查找字符串。

操作名称	输入文件中的格式	功能
$\text{Move}(k)$	Move k	将光标移动到第 k 个字符之后，如果 $k=0$，将光标移到文本开头
$\text{Insert}(n,s)$	Insert n s	在光标处插入长度为 n 的字符串 s，光标位置不变 $n\geq 1$
$\text{Delete}(n)$	Delete n	删除光标后的 n 个字符，光标位置不变，$n\geq 1$
$\text{Get}(n)$	Get n	输出光标后的 n 个字符，光标位置不变，$n\geq 1$
$\text{Prev}()$	Prev	光标前移一个字符
$\text{Next}()$	Next	光标后移一个字符

```
#include<bits/stdc++.h>
#include<ext/rope>
using namespace std;
using namespace __gnu_cxx;

const int N=25e5;
char s[N];

inline void reads(char *s, int len) {
    s[len]='\0'; len--;
    for(int i=0;i<=len;i++) {
        s[i]='\0';
        while(s[i]<32 || 126<s[i]) s[i]=getchar();
    }
}
```

```
rope<char> res;

int main(){
    int q; cin>>q;
    int cur=0;

    while(q--){
        int t;
        string op; cin>>op;
        if(op=="Move"){
            cin>>t; cur=t;
        }
        else if(op=="Insert"){
            cin>>t; reads(s, t);
            res.insert(cur, s);
        }
        else if(op=="Delete"){
            cin>>t;
            res.erase(cur, t);
        }
        else if(op=="Get"){
            cin>>t;
            cout<<res.substr(cur, t)<<endl;
        }
        else if(op=="Prev") cur--;
        else if(op=="Next") cur++;
    }

    return 0;
}
```

后缀数组

用 $SA[i]$ 来记录排名为 i 的非空后缀的编号，用 $Height[i]$ 来记录排名为 i 的非空后缀与排名为 $i-1$ 的非空后缀的最长公共前缀的长度。

```
#include<bits/stdc++.h>
using namespace std;

#define rep(i,a,b) for(int i=(a);i<=(b);i++)
#define dwn(i,a,b) for(int i=(a);i>=(b);i--)

const int N=1e6+5;

int n, m;
char s[N];
int sa[N], x[N], y[N], c[N], rk[N], height[N];

void get_sa(){
    rep(i,1,n) c[x[i]=s[i]]++;
    rep(i,2,m) c[i]+=c[i-1];
    dwn(i,n,1) sa[c[x[i]]--]=i;

    for(int k=1; k<=n; k<<=1){
```

```
        int num=0;
        rep(i,n-k+1,n) y[++num]=i;
        rep(i,1,n) if(sa[i]>k) y[++num]=sa[i]-k;
        rep(i,1,m) c[i]=0;
        rep(i,1,n) c[x[i]]++;
        rep(i,2,m) c[i]+=c[i-1];
        dwn(i,n,1) sa[c[x[y[i]]]--]=y[i], y[i]=0;
        swap(x, y);
        x[sa[1]]=1, num=1;
        rep(i,2,n) x[sa[i]]=(y[sa[i]]==y[sa[i-1]] && y[sa[i]+k]==y[sa[i-1]+k])?
num: ++num;
        if(num==n) break;
        m=num;
    }
}

void get_height(){
    rep(i,1,n) rk[sa[i]]=i;
    for(int i=1, k=0; i<=n; i++){
        if(rk[i]==1) continue;
        if(k) k--;
        int j=sa[rk[i]-1];
        while(i+k<=n && j+k<=n && s[i+k]==s[j+k]) k++;
        height[rk[i]]=k;
    }
}

int main(){
    scanf("%s", s+1);
    n=strlen(s+1), m='z';

    get_sa();
    get_height();

    rep(i,1,n) cout<<sa[i]<<' '; puts("");
    rep(i,1,n) cout<<height[i]<<' ';

    return 0;
}
```

可持久化 Trie

例题：

给定一个非负整数序列 a，初始长度为 N。

有 M 个操作，有以下两种操作类型：

1. `A x`：添加操作，表示在序列末尾添加一个数 x，序列的长度 N 增大 1。
2. `Q l r x`：询问操作，你需要找到一个位置 p，满足 $l \leq p \leq r$，使得：$a[p] \oplus a[p+1] \oplus \ldots \oplus a[N] \oplus xa[p] \oplus a[p+1] \oplus \ldots \oplus a[N] \oplus x$ 最大，输出这个最大值。

```
#include<bits/stdc++.h>
```

```cpp
using namespace std;

inline int read(){
   int s=0,w=1;
   char ch=getchar();
   while(ch<'0'||ch>'9'){if(ch=='-')w=-1;ch=getchar();}
   while(ch>='0'&&ch<='9') s=s*10+ch-'0',ch=getchar();
   return s*w;
}

const int N=6e5+5, M=N*25;
int tr[M][2], idx;
int root[N];
int s[N];
int max_id[M];

// i表示当前插入数对应的下标id，k表示处理到第几位。
// p表示上一版本的位置指针，q表示最新版本的位置指针。
void insert(int i,int k,int p,int q){
    // 边界
    if(k==-1){
        max_id[q]=i;
        return;
    }
    int v=s[i]>>k&1;
    if(p) tr[q][v^1]=tr[p][v^1];
    tr[q][v]=++idx;
    insert(i,k-1,tr[p][v],tr[q][v]);
    max_id[q]=max(max_id[tr[q][1]],max_id[tr[q][0]]);
}

inline int query(int root,int c,int id){
    int p=root;
    for(int i=23;i>=0;i--){
        int v=c>>i&1;
        if(max_id[tr[p][v^1]]>=id) p=tr[p][v^1];
        else p=tr[p][v];
    }
    return s[max_id[p]]^c;
}

int main(){
    max_id[0]=-1;
    root[0]=++idx;
    insert(0,23,0,root[0]);

    int n,m;
    cin>>n>>m;
    for(int i=1;i<=n;i++){
        int k; k=read();
        s[i]=k^s[i-1];
        root[i]=++idx;
        insert(i,23,root[i-1],root[i]);
    }

    char op[2];
    while(m--){
        scanf("%s",op);
```

```
        if(*op=='A'){
            n++;
            int x; x=read();
            s[n]=s[n-1]^x;
            root[n]=++idx;
            insert(n,23,root[n-1],root[n]);
        }
        else{
            int l,r,x; l=read(), r=read(), x=read();
            l--, r--;
            printf("%d\n",query(root[r],s[n]^x,l));
        }
    }
    return 0;
}
```

AC 自动机

例题：

给定 n 个模式串 s_i 和一个文本串 t，求有多少个不同的模式串在文本串里出现过。
两个模式串不同当且仅当他们**编号**不同。

```
#include<bits/stdc++.h>
using namespace std;

#define set0(a) memset(a,0,sizeof(a))
#define rep(i,a,b) for(int i=(a);i<=(b);i++)

const int N=2e5+5;

struct AcAutomaton{
    int tr[N][26], idx;
    int cnt[N], fail[N], id[N];
    int q[N], tt, hh;

    void clear(int u){
        memset(tr[u], 0, sizeof tr[u]), id[u]=cnt[u]=fail[u]=0;
    }

    void init(){
        clear(0);
        idx=0;
        tt=-1, hh=0;
    }

    void insert(char *s, int index){
        int u=0;
        for(int i=0; s[i]; i++){
            int v=s[i]-'a';
            if(!tr[u][v]){
                 tr[u][v]=++idx;
```

```
                clear(idx);
            }
            u=tr[u][v];
        }
        id[u]=index;
    }

    void build(){
        for(int i=0; i<26; i++) if(tr[0][i]){
            fail[tr[0][i]]=0;
            q[++tt]=tr[0][i];
        }

        while(tt>=hh){
            int u=q[hh++];
            for(int i=0; i<26; i++){
                int &p=tr[u][i];
                if(p) fail[p]=tr[fail[u]][i], q[++tt]=p;
                else p=tr[fail[u]][i];
            }
        }
    }

    void query(char *s){
        int u=0;
        for(int i=0; s[i]; i++){
            u=tr[u][s[i]-'a'];
            for(int p=u; p; p=fail[p]) cnt[id[p]]++;
        }
    }
}ac;

const int M=1e6+5;
char str[155][75], tmp[M];

int main(){
    int n;
    while(cin>>n, n){
        ac.init();

        for(int i=1; i<=n; i++){
            scanf("%s", str[i]);
            ac.insert(str[i], i);
        }

        ac.build();
        scanf("%s", tmp);

        ac.query(tmp);

        int mx=0;
        rep(i,1,n) mx=max(mx, ac.cnt[i]);

        cout<<mx<<endl;
        rep(i,1,n) if(ac.cnt[i]==mx) cout<<str[i]<<endl;
    }

    return 0;
```

```
}
```

后缀自动机

例题：

给定一个只包含小写字母的字符串 S,

请你求出 S 的所有出现次数不为 1 的子串的出现次数乘上该子串长度的最大值。

```
#include<bits/stdc++.h>
using namespace std;

const int N=2e6+5, M=N<<1;

#define int long long

struct Edge{
    int to, next;
}e[M];

int h[N], idx;

void add(int u, int v){
    e[idx].to=v, e[idx].next=h[u], h[u]=idx++;
}

char str[N];

struct Node{
    int len, fa; // longest length of state, link
    int ch[26];
}node[N];
int tot=1, last=1;
int f[N];

void ins(int c){
    int p=last, np=last=++tot;
    f[tot]=1;
    node[np].len=node[p].len+1;
    for(; p && !node[p].ch[c]; p=node[p].fa) node[p].ch[c]=np;
    if(!p) node[np].fa=1;
    else{
        int q=node[p].ch[c];
        if(node[q].len==node[p].len+1) node[np].fa=q;
        else{
            int nq=++tot;
            node[nq]=node[q], node[nq].len=node[p].len+1;
            node[q].fa=node[np].fa=nq;
            for(; p && node[p].ch[c]==q; p=node[p].fa) node[p].ch[c]=nq;
        }
    }
}

int res=0;
```

```
void dfs(int u){
    for(int i=h[u]; ~i; i=e[i].next){
        int go=e[i].to;
        dfs(go);
        f[u]+=f[go];
    }
    if(f[u]>1) res=max(res, f[u]*node[u].len);
}

signed main(){
    scanf("%s", str);
    for(int i=0; str[i]; i++) ins(str[i]-'a');
    memset(h, -1, sizeof h);
    for(int i=2; i<=tot; i++) add(node[i].fa, i);
    dfs(1);

    cout<<res<<endl;

    return 0;
}
```

莫队

基础莫队

例题：

小 Z 把这 N 只袜子从 1 到 N 编号，然后从编号 L 到 R 的袜子中随机选出两只来穿。

尽管小 Z 并不在意两只袜子是不是完整的一双，甚至不在意两只袜子是否一左一右，他却很在意袜子的颜色，毕竟穿两只不同色的袜子会很尴尬。

你的任务便是告诉小 Z，他有多大的概率抽到两只颜色相同的袜子。

```
inline void read(int &x){
    int s=0; x=1;
    char ch=getchar();
    while(ch<'0' || ch>'9') {if(ch=='-')x=-1;ch=getchar();}
    while(ch>='0' && ch<='9') s=(s<<3)+(s<<1)+ch-'0',ch=getchar();
    x*=s;
}

const int N=50050;

int n, m, w[N];
struct Query{
    int id, l, r, k;
    bool operator < (const Query &o)const{
        return k==o.k? ((k&1)? r<o.r: r>o.r): k<o.k;
    }
}q[N];

ll res;
pair<ll, ll> ans[N];
```

```
int b[N];

void add(int val){
    res+=b[val];
    b[val]++;
}

void del(int val){
    res-=b[val]-1;
    b[val]--;
}

int main(){
    cin>>n>>m;
    rep(i,1,n) read(w[i]);

    const int len=sqrt(n+1);
    rep(i,1,m){
        int l, r; read(l), read(r);
        q[i]={i, l, r, l/len};
    }

    sort(q+1, q+1+m);

    for(int i=0, j=1, k=1; k<=m; k++){
        int l=q[k].l, r=q[k].r;
        while(i<r) add(w[++i]);
        while(i>r) del(w[i--]);
        while(j<l) del(w[j++]);
        while(j>l) add(w[--j]);
        ll g=__gcd(res, 1LL*(r-l+1)*(r-l)/2);
        ans[q[k].id]={res/g, 1LL*(r-l+1)*(r-l)/2/g};
    }

    rep(i,1,m) cout<<ans[i].x<<'/'<<ans[i].y<<endl;

    return 0;
}
```

带修莫队

例题：

墨墨购买了一套 N 支彩色画笔（其中有些颜色可能相同），摆成一排，你需要回答墨墨的提问。

1. `Q L R` 代表询问你从第 L 支画笔到第 R 支画笔中共有几种不同颜色的画笔。
2. `R P Col` 把第 P 支画笔替换为颜色 Col。

```
#include<bits/stdc++.h>
using namespace std;

const int N=1e4+5, S=1e6+5;

struct Query{
```

```
    int id, l, r, t;
}q[N];

struct Modify{
    int p, c;
}c[N];

int cnt[S];
int w[N], ans[N];
int n, m;
int mq, mc;
int len;

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

bool cmp(Query &a, Query &b){
    int al=get(a.l), ar=get(a.r), bl=get(b.l), br=get(b.r);
    if(al!=bl) return al<bl;
    if(ar!=br) return ar<br;
    return a.t<b.t;
}

void add(int v, int &res){
    if(!cnt[v]) res++;
    cnt[v]++;
}

void del(int v, int &res){
    if(cnt[v]==1) res--;
    cnt[v]--;
}

int main(){
    cin>>n>>m;
    for(int i=1; i<=n; i++) cin>>w[i];

    for(int i=0; i<m; i++){
        char op; int l, r; cin>>op>>l>>r;
        if(op=='Q') mq++, q[mq]={mq, l, r, mc};
        else mc++, c[mc]={l, r};
    }

    len=cbrt((double)n*mc)+1;

    sort(q+1, q+1+mq, cmp);

    for(int i=0, j=1, res=0, t=0, k=1; k<=mq; k++){
        int id=q[k].id, l=q[k].l, r=q[k].r, tm=q[k].t;
        while(i<r) add(w[++i], res);
        while(i>r) del(w[i--], res);
        while(j<l) del(w[j++], res);
        while(j>l) add(w[--j], res);

        while(t<tm){
            t++;
            if(c[t].p>=j && c[t].p<=i){
```

```
                del(w[c[t].p], res);
                add(c[t].c, res);
            }
            swap(w[c[t].p], c[t].c);
        }

        while(t>tm){
            if(c[t].p>=j && c[t].p<=i){
                del(w[c[t].p], res);
                add(c[t].c, res);
            }
            swap(w[c[t].p], c[t].c);
            t--;
        }

        ans[id]=res;
    }

    for(int i=1; i<=mq; i++) cout<<ans[i]<<endl;

    return 0;
}
```

点分治 & 点分树

点分治

给定一个有 N 个点（编号 $0, 1, \ldots, N-1$）的树，每条边都有一个权值（不超过 1000）。

树上两个节点 x 与 y 之间的路径长度就是路径上各条边的权值之和。

求长度不超过 K 的路径有多少条。

```
#pragma GCC optimize("O3")
#include<bits/stdc++.h>
using namespace std;

#define endl '\n'
#define pb push_back
#define rep(i,a,b) for(int i=(a);i<=(b);i++)
#define dwn(i,a,b) for(int i=(a);i>=(b);i--)
#define ceil(a,b) (a+(b-1))/b
#define all(x) (x).begin(), (x).end()
using vi = vector<int>;

inline void read(int &x) {
    int s=0;x=1;
    char ch=getchar();
    while(ch<'0'||ch>'9') {if(ch=='-')x=-1;ch=getchar();}
    while(ch>='0'&&ch<='9') s=(s<<3)+(s<<1)+ch-'0',ch=getchar();
    x*=s;
}

const int N=1e4+5, M=N<<1;
```

```
int n, m;

struct node{
    int to, next, w;
}e[M];

int h[N], tot;

void add(int u, int v, int w){
    e[tot].to=v, e[tot].w=w, e[tot].next=h[u], h[u]=tot++;
}

bool vis[N];

int get_sz(int u, int fa){
    int res=1;
    for(int i=h[u]; ~i; i=e[i].next){
        int go=e[i].to;
        if(go==fa || vis[go]) continue;

        res+=get_sz(go, u);
    }

    return res;
}

int get_wc(int u, int fa, int tot, int &wc){
    if(vis[u]) return 0;
    int sum=1, ms=0;
    for(int i=h[u]; ~i; i=e[i].next){
        int go=e[i].to;
        if(go==fa) continue;

        int t=get_wc(go, u, tot, wc);
        ms=max(ms, t);
        sum+=t;
    }
    ms=max(ms, tot-sum);
    if(ms<=tot/2) wc=u;
    return sum;
}

void get_dis(int u, int fa, int dis, vi &v){
    if(vis[u]) return;
    v.pb(dis);
    for(int i=h[u]; ~i; i=e[i].next){
        int go=e[i].to;
        if(go==fa) continue;

        get_dis(go, u, dis+e[i].w, v);
    }
}

int get(vi v){
    if(v.size()<2) return 0;
    sort(all(v));
    int res=0;
```

```
    rep(i,0,v.size()-1) res+=upper_bound(v.begin()+i+1, v.end(), m-v[i])-
v.begin()-i-1;
    return res;
}

int calc(int u){
    if(vis[u]) return 0;
    get_wc(u, -1, get_sz(u, -1), u);
    vis[u]=true;

    int res=0;
    vi buf; // 存储重心到其它点的距离。
    for(int i=h[u]; ~i; i=e[i].next){
        int go=e[i].to;
        vi tmp;

        get_dis(go, u, e[i].w, tmp);
        res-=get(tmp);

        buf.insert(buf.end(), all(tmp));
    }

    for(int i: buf) if(i<=m) res++;
    res+=get(buf);

    for(int i=h[u]; ~i; i=e[i].next) res+=calc(e[i].to);
    return res;
}

int main(){
    while(cin>>n>>m, n || m){
        rep(i,1,n) h[i]=-1, vis[i]=false;
        tot=0;

        rep(i,1,n-1){
            int u, v, w; read(u), read(v), read(w); u++, v++;
            add(u, v, w), add(v, u, w);
        }

        cout<<calc(1)<<endl;
    }

    return 0;
}
```

点分树

例题：

给出度数不超过 3 的树，每个点有一个属性 val，每次询问一个点 u 求所有年龄在 $[L, R]$ 之间的点到 u 这个点的**距离之和**。

```
#pragma GCC optimize("O3")
```

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

#define int long long
#define endl '\n'
#define debug(x) cerr << #x << ": " << x << endl
#define pb push_back
#define rep(i,a,b) for(int i=(a);i<=(b);i++)

#define all(x) (x).begin(), (x).end()
#define lb(a, x) distance(begin(a), lower_bound(all(a), (x)))
#define ub(a, x) distance(begin(a), upper_bound(all(a), (x)))

inline void read(int &x) {
    int s=0;x=1;
    char ch=getchar();
    while(ch<'0'||ch>'9') {if(ch=='-')x=-1;ch=getchar();}
    while(ch>='0'&&ch<='9') s=(s<<3)+(s<<1)+ch-'0',ch=getchar();
    x*=s;
}

const int N=15e4+5, M=N<<1;

int n, m, A, age[N];

struct node{
    int to, w, next;
}e[M];

int h[N], tot;

void add(int u, int v, int w){
    e[tot].to=v, e[tot].w=w, e[tot].next=h[u], h[u]=tot++;
}

bool vis[N];

struct Father{
    int u, num, dis;
};

struct Son{
    int age, dis;
    bool operator < (const Son &o)const{
        return age<o.age;
    }
};

vector<Father> f[N];
vector<Son> s[N][3];

int get_sz(int u, int fa){
    if(vis[u]) return 0;
    int res=1;
    for(int i=h[u]; ~i; i=e[i].next){
        int go=e[i].to;
        if(go==fa) continue;
        res+=get_sz(go, u);
```

```
    }
    return res;
}

int get_wc(int u, int fa, int tot, int &wc){
    if(vis[u]) return 0;
    int sum=1, ms=0;
    for(int i=h[u]; ~i; i=e[i].next){
        int go=e[i].to;
        if(vis[go] || go==fa) continue;

        int t=get_wc(go, u, tot, wc);
        ms=max(ms, t);
        sum+=t;
    }
    ms=max(ms, tot-sum);
    if(ms<=tot/2) wc=u;
    return sum;
}

void get_dis(int u, int fa, int dis, int wc, int k, vector<Son> &p){
    if(vis[u]) return;
    f[u].pb({wc, k, dis});
    p.pb({age[u], dis});
    for(int i=h[u]; ~i; i=e[i].next){
        int go=e[i].to;
        if(go==fa || vis[go]) continue;
        get_dis(go, u, dis+e[i].w, wc, k, p);
    }
}

void calc(int u){
    if(vis[u]) return;
    get_wc(u, -1, get_sz(u, -1), u);
    vis[u]=true;

    for(int i=h[u], k=0; ~i; i=e[i].next, k++){ //
        int go=e[i].to;
        if(vis[go]) continue;

        vector<Son> &p=s[u][k];
        p.pb({-1, 0}), p.pb({A+1, 0}); // 哨兵
        get_dis(go, u, e[i].w, u, k, p);

        sort(all(p));

        rep(i,1,p.size()-1) p[i].dis+=p[i-1].dis;
    }
    for(int i=h[u]; ~i; i=e[i].next) calc(e[i].to);
}

int query(int u, int l, int r){
    int res=0;
    for(auto fa: f[u]){ // 遍历与上层的重心的相关信息
        int g=age[fa.u]; // ① 展开和重心节点的讨论
        if(l<=g && g<=r) res+=fa.dis;
        rep(i,0,2){ // 展开和兄弟子树的节点的讨论
            if(i==fa.num) continue;
```

```
            vector<Son> &p=s[fa.u][i];
            if(!p.size()) continue; // 空树跳过
            int L=lb(p, Son({l, -1}));
            int R=lb(p, Son({r+1, -1}));
            res+=(R-L)*fa.dis+p[R-1].dis-p[L-1].dis;
        }
    }

    rep(i,0,2){ // 展开和子节点的讨论
        vector<Son> &p=s[u][i];
        if(!p.size()) continue; // 空树跳过
        int L=lb(p, Son({l, -1}));
        int R=lb(p, Son({r+1, -1}));
        res+=p[R-1].dis-p[L-1].dis;
    }

    return res;
}

signed main(){
    memset(h, -1, sizeof h);
    cin>>n>>m>>A;
    rep(i,1,n) read(age[i]);

    rep(i,1,n-1){
        int u, v, w; read(u), read(v), read(w);
        add(u, v, w), add(v, u, w);
    }

    calc(1);

    int res=0;
    rep(i,1,m){
        int u, l, r; read(u), read(l), read(r);
        l=(l+res)%A, r=(r+res)%A;
        if(l>r) swap(l, r);
        res=query(u, l, r);
        cout<<res<<endl;
    }

    return 0;
}
```

LCT

例题：

给定 n 个点，编号从 1 到 n，其中第 i 个点的初始权值为 a_i。

现在要对这些点进行 m 次操作，操作共分为以下 4 种：

- `0 x y`，表示询问点 x 到点 y 之间的路径上的所有点（包括两端点）的权值的异或和。保证 x 和 y 之间存在连通路径。
- `1 x y`，表示在点 x 和点 y 之间增加一条边 (x,y)。注意：**如果两点已经处于连通状态，则无视该操作。**

- `2 x y`，表示删除边 (x,y)。注意：**如果该边不存在，则无视该操作。**
- `3 x w`，表示将点 x 的权值修改为 w。

```
#include<bits/stdc++.h>
using namespace std;

inline void read(int &x) {
    int s=0;x=1;
    char ch=getchar();
    while(ch<'0'||ch>'9') {if(ch=='-')x=-1;ch=getchar();}
    while(ch>='0'&&ch<='9') s=(s<<3)+(s<<1)+ch-'0',ch=getchar();
    x*=s;
}

const int N=1e5+5;

int n, m;

struct Node{
    int s[2], p, v;
    int sum, rev;
}tr[N];
int stk[N];

void pushrev(int x){
    swap(tr[x].s[0], tr[x].s[1]);
    tr[x].rev^=1;
}

void pushup(int x){
    tr[x].sum=tr[tr[x].s[0]].sum^tr[x].v^tr[tr[x].s[1]].sum;
}

void pushdown(int x){
    if(tr[x].rev){
        pushrev(tr[x].s[0]), pushrev(tr[x].s[1]);
        tr[x].rev=0;
    }
}

bool isroot(int x){
    return tr[tr[x].p].s[0]!=x && tr[tr[x].p].s[1]!=x;
}

void rotate(int x){
    int y=tr[x].p, z=tr[y].p;
    int k=tr[y].s[1]==x;
    if(!isroot(y)) tr[z].s[tr[z].s[1]==y]=x;
    tr[x].p=z;
    tr[tr[x].s[k^1]].p=y, tr[y].s[k]=tr[x].s[k^1];
    tr[y].p=x, tr[x].s[k^1]=y;
    pushup(y), pushup(x);
}

void splay(int x){
    int top=0, r=x;
    stk[++top]=r;
```

```
    while(!isroot(r)) stk[++top]=r=tr[r].p;
    while(top) pushdown(stk[top--]);

    while(!isroot(x)){
        int y=tr[x].p, z=tr[y].p;
        if(!isroot(y))
            if((tr[y].s[1]==x) ^ (tr[z].s[1]==y)) rotate(x);
            else rotate(y);
        rotate(x);
    }
}

void access(int x){ // 建立从原树的根到 x 的路径，同时 x 成为（splay）根节点。
    int z=x;
    for(int y=0; x; y=x, x=tr[x].p){
        splay(x);
        tr[x].s[1]=y, pushup(x);
    }
    splay(z);
}

// !
void makeroot(int x){ // x 成为原树的根节点
    access(x);
    pushrev(x);
}

int findroot(int x){ // 找到 x 所在原树的根节点，再将原树根节点转到 splay 的根节点
    access(x);
    while(tr[x].s[0]) pushdown(x), x=tr[x].s[0];
    splay(x);
    return x;
}

void split(int x, int y){ // x, y 路径用 splay 维护起来，splay 根为 y
    makeroot(x);
    access(y);
}

void link(int x, int y){ // 若 x, y 不连通，连边
    makeroot(x);
    if(findroot(y)!=x) tr[x].p=y;
}

void cut(int x, int y){ // x, y 间若直接连边，删除之
    makeroot(x); // 让 y 一定是 x 的后继
    if(findroot(y)==x && tr[y].p==x && !tr[y].s[0]){
        tr[x].s[1]=tr[y].p=0;
        pushup(x);
    }
}

int main(){
    cin>>n>>m;
    for(int i=1; i<=n; i++) read(tr[i].v);

    while(m--){
        int op, x, y; read(op), read(x), read(y);
```

```
        if(op==0){
            split(x, y);
            cout<<tr[y].sum<<'\n';
        }
        else if(op==1) link(x, y);
        else if(op==2) cut(x, y);
        else if(op==3){
            splay(x);
            tr[x].v=y;
            pushup(x);
        }
    }
    return 0;
}
```

树上启发式合并

例题：

节点有颜色的树，需要统计每个棵子树内出现次数最多的颜色之和。

```
// Problem: E. Lomsat gelral
// Contest: Codeforces - Educational Codeforces Round 2
// URL: https://codeforces.com/problemset/problem/600/E
const int N=1e5+5, M=N<<1;

int n, w[N];

struct Edge{
    int to, next;
}e[M];

int h[N], tot;

void add(int u, int v){
    e[tot].to=v, e[tot].next=h[u], h[u]=tot++;
}

int sz[N], son[N];

void dfs(int u, int fa){
    sz[u]=1;
    for(int i=h[u]; ~i; i=e[i].next){
        int go=e[i].to;
        if(go==fa) continue;
        dfs(go, u);
        sz[u]+=sz[go];
        if(sz[go]>sz[son[u]]) son[u]=go;
    }
}

int res[N];
int cnt[N], ans, mx;

void upd(int u, int fa, int d){
```

```
    cnt[w[u]]+=d;
    if(d==1){
        if(cnt[w[u]]>mx) mx=cnt[w[u]], ans=w[u];
        else if(cnt[w[u]]==mx) ans+=w[u];
    }

    for(int i=h[u]; ~i; i=e[i].next){
        int go=e[i].to;
        if(go==fa) continue;
        upd(go, u, d);
    }
}

void dfs2(int u, int fa, bool del){
    for(int i=h[u]; ~i; i=e[i].next){
        int go=e[i].to;
        if(go==fa || go==son[u]) continue;
        dfs2(go, u, 1);
    }
    if(son[u]) dfs2(son[u], u, 0);

    cnt[w[u]]++;
    if(cnt[w[u]]>mx) mx=cnt[w[u]], ans=w[u];
    else if(cnt[w[u]]==mx) ans+=w[u];
    for(int i=h[u]; ~i; i=e[i].next){
        int go=e[i].to;
        if(go==fa || go==son[u]) continue;
        upd(go, u, 1);
    }
    res[u]=ans;

    if(del) upd(u, fa, -1), mx=ans=0;
}

signed main(){
    memset(h, -1, sizeof h);
    cin>>n;
    rep(i,1,n) read(w[i]);

    rep(i,1,n-1){
        int u, v; read(u), read(v);
        add(u, v), add(v, u);
    }

    dfs(1, -1);
    dfs2(1, -1, 0);

    rep(i,1,n) cout<<res[i]<<' ';
    cout<<endl;

    return 0;
}
```

线段树

主席树

例题：

查询区间 k 小数。

```
const int N=1e5+5, INF=1e9+10;

int n, m;

struct Node{
    int l, r;
    int cnt;
    #define ls tr[u].l
    #define rs tr[u].r
}tr[N*35];

int idx, root[N];

void pushup(int u){
    tr[u].cnt=tr[ls].cnt+tr[rs].cnt;
}

void insert(int &q, int p, int l, int r, int x){
    q=++idx;
    tr[q]=tr[p];
    if(l==r) return tr[q].cnt++, void();

    int mid=l+r>>1;
    if(x<=mid) insert(tr[q].l, tr[p].l, l, mid, x);
    else insert(tr[q].r, tr[p].r, mid+1, r, x);
    pushup(q);
}

int query(int p, int q, int l, int r, int k){
    if(l==r) return l;
    int mid=l+r>>1;
    int cnt=tr[tr[q].l].cnt-tr[tr[p].l].cnt;
    if(cnt>=k) return query(tr[p].l, tr[q].l, l, mid, k);
    else return query(tr[p].r, tr[q].r, mid+1, r, k-cnt);
}

int main(){
    cin>>n>>m;
    for(int i=1; i<=n; i++){
        int x; cin>>x;
        insert(root[i], root[i-1], -INF, INF, x);
    }

    while(m--){
        int l, r, k; cin>>l>>r>>k;
        cout<<query(root[l-1], root[r], -INF, INF, k)<<endl;
    }

    return 0;
}
```

扫描线

例题：求面积并

```
const int N=1e5+5;

int n;
struct Node{
    int l, r;
    int cnt;
    double len;

    #define ls(u) u<<1
    #define rs(u) u<<1|1
}tr[N<<2];

vector<double> w;
int find(double x){
    return lower_bound(all(w), x)-begin(w);
}

void pushup(int u){
    if(tr[u].cnt) tr[u].len=w[tr[u].r+1]-w[tr[u].l];
    else tr[u].len=tr[ls(u)].len+tr[rs(u)].len;
}

void build(int u, int l, int r){
    tr[u]={l, r, 0, 0};
    if(l==r) return;
    int mid=l+r>>1;
    build(ls(u), l, mid), build(rs(u), mid+1, r);
}

void upd(int u, int l, int r, int k){
    if(l<=tr[u].l && tr[u].r<=r) return tr[u].cnt+=k, pushup(u), void();
    int mid=tr[u].l+tr[u].r>>1;
    if(l<=mid) upd(ls(u), l, r, k);
    if(mid<r) upd(rs(u), l, r, k);
    pushup(u);
}

struct Query{
    double l, r, x;
    int k;

    bool operator < (const Query &o)const{
        return x<o.x;
    }
}q[N];
int tot;

int main(){
    int cs;
```

```
    while(cin>>n, n){
        w.clear();

        tot=0;
        rep(i,1,n){
            double x1, y1, x2, y2; scanf("%lf%lf%lf%lf", &x1, &y1, &x2, &y2);
            w.pb(y1), w.pb(y2);
            q[++tot]={y1, y2, x1, 1};
            q[++tot]={y1, y2, x2, -1};
        }
        sort(q+1, q+1+tot);

        sort(all(w));
        w.erase(unique(all(w)), end(w));

        build(1, 0, w.size()+5);

        double res=0;
        rep(i,1,tot){
            if(i>1) res+=(q[i].x-q[i-1].x)*tr[1].len;
            upd(1, find(q[i].l), find(q[i].r)-1, q[i].k);
        }

        printf("Test case #%d\n", ++cs);
        printf("Total explored area: %.2lf\n\n", res);
    }
    return 0;
}
```

线段树合并

例题

给出一颗树，每个点都有一个权值，最后对于每个点，输出在它的子树中，有多少个点的权值比它大。

```
inline void read(int &x) {
    int s=0;x=1;
    char ch=getchar();
    while(ch<'0'||ch>'9') {if(ch=='-')x=-1;ch=getchar();}
    while(ch>='0'&&ch<='9') s=(s<<3)+(s<<1)+ch-'0',ch=getchar();
    x*=s;
}

const int N=1e5+5, M=N<<1;

vi nums;

int find(int x){
    return lb(nums, x)+1;
}

int w[N], n;
int res[N];
```

```
struct Edge{
    int to, next;
}e[M];

int h[N], tot;

void add(int u, int v){
    e[tot].to=v, e[tot].next=h[u], h[u]=tot++;
}

struct Node{
    int l, r;
    int cnt;

    #define ls tr[u].l
    #define rs tr[u].r
}tr[N*22];

int root[N], idx;

void pushup(int u){
    tr[u].cnt=tr[ls].cnt+tr[rs].cnt;
}

void insert(int &u, int l, int r, int x){
    if(!u) u=++idx;
    if(l==r) return tr[u].cnt++, void();
    int mid=l+r>>1;
    if(x<=mid) insert(ls, l, mid, x);
    else insert(rs, mid+1, r, x);
    pushup(u);
}

int merge(int u, int v){
    if(!u) return v;
    if(!v) return u;
    ls=merge(ls, tr[v].l);
    rs=merge(rs, tr[v].r);
    pushup(u);
    return u;
}

int query(int u, int l, int r, int nl, int nr){
    if(!u) return 0;
    if(nl<=l && r<=nr) return tr[u].cnt;
    int mid=l+r>>1, res=0;
    if(nl<=mid) res+=query(ls, l, mid, nl, nr);
    if(mid<nr) res+=query(rs, mid+1, r, nl, nr);
    return res;
}

void dfs(int u, int fa){
    insert(root[u], 1, n, find(w[u]));
    for(int i=h[u]; ~i; i=e[i].next){
        int go=e[i].to;
        if(go==fa) continue;
        dfs(go, u);
        root[u]=merge(root[u], root[go]);
```

```
    }
    if(find(w[u])+1<=n) res[u]=query(root[u], 1, n, find(w[u])+1, n);
}

int main(){
    read(n);
    rep(i,1,n) read(w[i]), nums.pb(w[i]);

    sort(all(nums));
    nums.erase(unique(all(nums)), nums.end());

    memset(h, -1, sizeof h);
    rep(i,2,n){
        int fa; read(fa);
        add(fa, i), add(i, fa);
    }

    dfs(1, -1);

    rep(i,1,n) cout<<res[i]<<endl;
    return 0;
}
```

线段树优化建图

```
// Problem: B. Legacy
// Contest: Codeforces - Codeforces Round #406 (Div. 1)
// URL: https://codeforces.com/problemset/problem/786/B
// Memory Limit: 256 MB
// Time Limit: 2000 ms
//
// Powered by CP Editor (https://cpeditor.org)

int n, q, S;

struct Edge{
    int to, next;
    ll w;
}e[M];

int h[N], tot;

void add(int u, int v, ll w){
    e[tot].to=v, e[tot].w=w, e[tot].next=h[u], h[u]=tot++;
}

#define ls u<<1
#define rs u<<1|1

int id[N];
int idx, tmp;
int rtL, rtR;
void build(int u, int l, int r, int op){
```

```
    if(op==-1){
        id[u]=++idx;
        int mid=l+r>>1;
        build(ls, l, mid, 0), build(rs, mid+1, r, 1);
        rtL=ls, rtR=rs;
        return;
    }
    if(l==r){
        id[u]=++tmp;
        return;
    }

    int mid=l+r>>1;
    id[u]=++idx;
    build(ls, l, mid, op), build(rs, mid+1, r, op);
    if(!op) add(id[u], id[ls], 0), add(id[u], id[rs], 0);
    else add(id[ls], id[u], 0), add(id[rs], id[u], 0);
}

void link(int u, int l, int r, int cur, int nl, int nr, ll w, int op){
    if(nl<=l && r<=nr){
        if(!op) add(cur, id[u], w);
        else add(id[u], cur, w);
        return;
    }
    int mid=l+r>>1;
    if(nl<=mid) link(ls, l, mid, cur, nl, nr, w, op);
    if(mid<nr) link(rs, mid+1, r, cur, nl, nr, w, op);
}

bool vis[N];
ll d[N];
priority_queue<pli, vector<pli>, greater<pli>> pq;

#define x first
#define y second

void dijk(){
    rep(i,1,idx) d[i]=1e18;
    d[S]=0;
    pq.push({d[S], S});

    while(pq.size()){
        auto t=pq.top(); pq.pop();
        int u=t.y;
        if(vis[u]) continue;
        vis[u]=true;

        for(int i=h[u]; ~i; i=e[i].next){
            int go=e[i].to;
            if(d[go]>e[i].w+d[u]){
                d[go]=e[i].w+d[u];
                pq.push({d[go], go});
            }
        }
    }

    rep(i,n+1,n<<1) cout<<(d[i]==INF? -1: d[i])<<' ';
```

```
    cout<<endl;
}

int main(){
    memset(h, -1, sizeof h);
    cin>>n>>q>>S;
    idx=n<<1;
    build(1, 1, n<<1, -1);
    rep(i,1,n) add(i, i+n, 0);

    while(q--){
        int op; scanf("%d", &op);
        int u, v;
        ll w;
        if(op==1){
            scanf("%d%d%lld", &u, &v, &w);
            add(u+n, v, w);
        }
        else if(op==2){
            int l, r; scanf("%d%d%d%lld", &u, &l, &r, &w);
            link(rtL, 1, n, u, l, r, w, 0);
        }
        else if(op==3){
            int l, r; scanf("%d%d%d%lld", &u, &l, &r, &w);
            link(rtR, n+1, n<<1, u, l+n, r+n, w, 1);
        }
    }


    dijk();

    return 0;
}
```

线段树标记维护

例题：序列操作

lxhgww 最近收到了一个 01 序列，序列里面包含了 n 个数，下标从 0 开始。这些数要么是 0，要么是 1，现在对于这个序列有五种变换操作和询问操作：

- `0 l r` 把 $[l,r]$ 区间内的所有数全变成 0
- `1 l r` 把 $[l,r]$ 区间内的所有数全变成 1
- `2 l r` 把 $[l,r]$ 区间内的所有数全部取反，也就是说把所有的 0 变成 1，把所有的 1 变成 0
- `3 l r` 询问 $[l,r]$ 区间内总共有多少个 1
- `4 l r` 询问 $[l,r]$ 区间内最多有多少个连续的 1

```
const int N=1e5+5;

int n, m;
int w[N];
int tag1[N<<2], tag2[N<<2];

struct Tree{
    int l, r, len;
```

```
    int sum, con[2], pre[2], suf[2]; // longest connected, prefix, suffix

    #define ls(u) u<<1
    #define rs(u) u<<1|1
}tr[N<<2];

Tree merge(Tree a, Tree b){
    Tree res={
        a.l, b.r, a.len+b.len,
        a.sum+b.sum
    };
    if(!a.sum) res.pre[0]=a.len+b.pre[0];
    else res.pre[0]=a.pre[0];
    if(a.sum==a.len) res.pre[1]=a.len+b.pre[1];
    else res.pre[1]=a.pre[1];

    if(!b.sum) res.suf[0]=b.len+a.suf[0];
    else res.suf[0]=b.suf[0];
    if(b.sum==b.len) res.suf[1]=b.len+a.suf[1];
    else res.suf[1]=b.suf[1];

    rep(i,0,1) res.con[i]=max(max(a.con[i], b.con[i]), a.suf[i]+b.pre[i]);

    return res;
}

void F(int u, int op){
    if(op!=2){
        int t=op;
        tr[u].sum=t*tr[u].len;
        tr[u].con[t]=tr[u].pre[t]=tr[u].suf[t]=tr[u].len;
        tr[u].con[t^1]=tr[u].pre[t^1]=tr[u].suf[t^1]=0;
        tag1[u]=op, tag2[u]=0;
    }
    else{
        tag2[u]^=1;
        tr[u].sum=tr[u].len-tr[u].sum;
        swap(tr[u].con[0], tr[u].con[1]);
        swap(tr[u].pre[0], tr[u].pre[1]);
        swap(tr[u].suf[0], tr[u].suf[1]);
    }
}

void pushdown(int u){
    if(~tag1[u]){
        F(ls(u), tag1[u]), F(rs(u), tag1[u]);
        tag1[u]=-1;
    }
    if(tag2[u]){
        F(ls(u), 2), F(rs(u), 2);
        tag2[u]=0;
    }
}

void build(int u, int l, int r){
    tag1[u]=-1;
    if(l==r){
        int v=w[l];
```

```
        tr[u]={l, r, 1, v};
        tr[u].con[0]=tr[u].pre[0]=tr[u].suf[0]=v^1;
        tr[u].con[1]=tr[u].pre[1]=tr[u].suf[1]=v;
        return;
    }
    int mid=l+r>>1;
    build(ls(u), l, mid), build(rs(u), mid+1, r);
    tr[u]=merge(tr[ls(u)], tr[rs(u)]);
}

void modify(int u, int l, int r, int op){
    if(tr[u].l>=l && tr[u].r<=r){
        F(u, op);
        return;
    }
    pushdown(u);
    int mid=tr[u].l+tr[u].r>>1;
    if(l<=mid) modify(ls(u), l, r, op);
    if(r>mid) modify(rs(u), l, r, op);
    tr[u]=merge(tr[ls(u)], tr[rs(u)]);
}

Tree query(int u, int l, int r){
    if(tr[u].l>=l && tr[u].r<=r) return tr[u];
    pushdown(u);
    int mid=tr[u].l+tr[u].r>>1;
    Tree L={-1}, R={-1}; //
    if(l<=mid) L=query(ls(u), l, r);
    if(r>mid) R=query(rs(u), l, r);
    if(L.l==-1) return R; if(R.l==-1) return L;
    return merge(L, R);
}

int main(){
    read(n), read(m);
    rep(i,1,n) read(w[i]);

    build(1, 1, n);

    rep(i,1,m){
        int op, l, r; read(op), read(l), read(r); l++, r++;
        if(op<=2) modify(1, l, r, op);
        else{
            Tree t=query(1, l, r);
            if(op==3) cout<<t.sum<<endl;
            else if(op==4) cout<<t.con[1]<<endl;
        }

        if(i==5){
            Tree t=query(1, l, r);
            debug(t.con[1]);
        }
    }
    return 0;
}
```

势能线段树

例题：

第一行一个整数 n，代表数列中数的个数。

第二行 n 个正整数，表示初始状态下数列中的数。

第三行一个整数 m，表示有 m 次操作。

接下来 m 行每行三个整数 `k l r`。

- $k=0$ 表示给 $[l,r]$ 中的每个数开平方（下取整）。
- $k=1$ 表示询问 $[l,r]$ 中各个数的和。

```
inline void read(int &x) {
    int s=0;x=1;
    char ch=getchar();
    while(ch<'0'||ch>'9') {if(ch=='-')x=-1;ch=getchar();}
    while(ch>='0'&&ch<='9') s=(s<<3)+(s<<1)+ch-'0',ch=getchar();
    x*=s;
}

const int N=1e5+5;

int n, q, w[N];

struct Node{
    int l, r;
    int mx, sum;

    #define ls(u) u<<1
    #define rs(u) u<<1|1
}tr[N<<2];

void pushup(int u){
    tr[u].mx=max(tr[ls(u)].mx, tr[rs(u)].mx);
    tr[u].sum=tr[ls(u)].sum+tr[rs(u)].sum;
}

void build(int u, int l, int r){
    tr[u]={l, r, w[l], w[r]};
    if(l==r) return;
    int mid=l+r>>1;
    build(ls(u), l, mid), build(rs(u), mid+1, r);
    pushup(u);
}

void upd(int u, int l, int r){
    if(tr[u].l==tr[u].r) return tr[u].mx=tr[u].sum=sqrt(tr[u].sum), void();
    int mid=tr[u].l+tr[u].r>>1;
    if(l<=mid && tr[ls(u)].mx>1) upd(ls(u), l, r);
    if(mid<r && tr[rs(u)].mx>1) upd(rs(u), l, r);
    pushup(u);
}

int query(int u, int l, int r){
```

```
    if(l<=tr[u].l && tr[u].r<=r) return tr[u].sum;
    int mid=tr[u].l+tr[u].r>>1, res=0;
    if(l<=mid) res+=query(ls(u), l, r);
    if(mid<r) res+=query(rs(u), l, r);
    return res;
}

signed main(){
    read(n);
    rep(i,1,n) read(w[i]);
    read(q);

    build(1, 1, n);

    while(q--){
        int op, l, r; read(op), read(l), read(r);
        if(l>r) swap(l, r);
        if(!op) upd(1, l, r);
        else cout<<query(1, l, r)<<endl;
    }
    return 0;
}
```

分块

例题：

1. 若第一个字母为 `M`，则紧接着有三个数字 L, R, W。表示对闭区间 $[l,r]$ 内所有英雄的身高加上 W。
2. 若第一个字母为 `A`，则紧接着有三个数字 L, R, C。询问闭区间 $[l,r]$ 内有多少英雄的身高大于等于 C。

```
// Problem: P2801 教主的魔法
// Contest: Luogu
// URL: https://www.luogu.com.cn/problem/P2801

const int N=1e6+6, M=1010;

int n, q;
int len;

struct Node{
    int x, w;
    bool operator < (const Node &o)const{
        return w<o.w;
    }
}e[N];

int get(int x){
    return (x+len-1)/len;
}

int getL(int id){
    return (id-1)*len+1;
}
```

```
int getR(int id){
    return min(id*len, n);
}

int add[M];

int query(int l, int r, int k){
    int res=0;
    if(get(l)==get(r)){
        int id=get(l);
        if(add[id]){
            rep(i,getL(id),getR(id)) e[i].w+=k;
            add[id]=0;
        }
        rep(i,getL(id),getR(id)) if(e[i].x>=l && e[i].x<=r && e[i].w>=k) res++;
        return res;
    }
    int id=get(l);
    if(add[id]){
        rep(i,getL(id),getR(id)) e[i].w+=k;
        add[id]=0;
    }
    rep(i,getL(id),getR(id)) if(e[i].x>=l && e[i].x<=getR(id) && e[i].w>=k)
res++;
    // debug(res);
    id=get(r);
    if(add[id]){
        rep(i,getL(id),getR(id)) e[i].w+=k;
        add[id]=0;
    }
    rep(i,getL(id),getR(id)) if(e[i].x>=getL(id) && e[i].x<=r && e[i].w>=k)
res++;
    // debug(res);
    int L=get(l)+1, R=get(r)-1;
    rep(id,L,R){
        int l=getL(id), r=getR(id);
        while(l<r){
            int mid=l+r>>1;
            if(e[mid].w>=k-add[id]) r=mid;
            else l=mid+1;
        }
        res+=(e[l].w>=k-add[id]? getR(id)-l+1: 0);
    }

    return res;
}

void upd(int l, int r, int k){
    if(get(l)==get(r)){
        int id=get(l);
        rep(i,getL(id),getR(id)) if(e[i].x>=l && e[i].x<=r) e[i].w+=k;
        sort(e+getL(id), e+getR(id)+1);
        return;
    }
    int id=get(l);
    rep(i,getL(id),getR(id)) if(e[i].x>=l && e[i].x<=getR(id)) e[i].w+=k;
    sort(e+getL(id), e+getR(id)+1);
```

```
    id=get(r);
    rep(i,getL(id),getR(id)) if(e[i].x>=getL(id) && e[i].x<=r) e[i].w+=k;
    sort(e+getL(id), e+getR(id)+1);

    int L=get(l)+1, R=get(r)-1;
    rep(id,L,R) add[id]+=k;
}

void out(){
    rep(i,1,n) cerr<<e[i].w<<' ';
    cerr<<endl;
}

int main(){
    cin>>n>>q;
    len=sqrt(n+1);
    rep(i,1,n) read(e[i].w), e[i].x=i;

    rep(id,1,get(n)) sort(e+getL(id), e+getR(id)+1);

    while(q--){
        char ch; cin>>ch;
        int l, r, v; read(l), read(r), read(v);
        if(ch=='A') cout<<query(l, r, v)<<endl;
        else upd(l, r, v);
    }
    return 0;
}
```

KDT

例题 1：平面最近点对

```
const int N=5e5+5, INF=0x3f3f3f3f;

struct Point{
    int x[2];
};

struct Node{
    int l, r;
    Point P;
    int L[2], R[2], sz;

    #define ls tr[u].l
    #define rs tr[u].r
}tr[N];

int n;
int idx, root;

inline void pushup(int u){
    auto &L=tr[ls], &R=tr[rs];
    tr[u].sz=L.sz+R.sz+1;
```

```
    rep(i,0,1){
        tr[u].L[i]=min(tr[u].P.x[i], min(L.L[i], R.L[i]));
        tr[u].R[i]=max(tr[u].P.x[i], max(L.R[i], R.R[i]));
    }
}

Point pt[N];

int build(int l, int r, int k){
    if(l>r) return 0;
    int mid=l+r>>1;
    int u=++idx;

    nth_element(pt+l, pt+mid, pt+r+1, [&](Point a, Point b){
        return a.x[k]<b.x[k];
    });
    tr[u].P=pt[mid];

    ls=build(l, mid-1, k^1), rs=build(mid+1, r, k^1);
    pushup(u);
    return u;
}

ll res=4e18;

inline ll Dis(Point a, Point b){
    ll dx=a.x[0]-b.x[0], dy=a.x[1]-b.x[1];
    return dx*dx+dy*dy;
}

inline ll H(Node t, Point p){
    auto sqr=[](int x)->ll{return 1LL*x*x;};

    ll x=p.x[0], y=p.x[1];
    ll res=0;
    if(x<t.L[0]) res+=sqr(x-t.L[0]);
    if(x>t.R[0]) res+=sqr(x-t.R[0]);
    if(y<t.L[1]) res+=sqr(y-t.L[1]);
    if(y>t.R[1]) res+=sqr(y-t.R[1]);
    return res;
}

void query(int u, Point p){
    if(!u) return;
    if(tr[u].P.x[0]!=p.x[0] || tr[u].P.x[1]!=p.x[1]) res=min(res, Dis(tr[u].P,
p));
    ll LV=4e18, RV=4e18;
    if(ls) LV=H(tr[ls], p);
    if(rs) RV=H(tr[rs], p);

    if(LV<RV){
        if(LV<res) query(ls, p);
        if(RV<res) query(rs, p);
    }
    else{
        if(RV<res) query(rs, p);
        if(LV<res) query(ls, p);
```

```
    }
}

int main(){
    cin>>n;
    // init
    tr[0].L[0]=tr[0].L[1]=INF;
    tr[0].R[0]=tr[0].R[1]=-INF;

    rep(i,1,n){
        int x, y; read(x), read(y);
        pt[i]={x, y};
    }
    sort(pt+1, pt+1+n, [](Point a, Point b){
        return a.x[0]==b.x[0]? a.x[1]<b.x[1]: a.x[0]<b.x[0];
    });

    root=build(1, n, 0);
    rep(i,1,n) query(root, pt[i]);
    cout<<res<<endl;

    return 0;
}
```

例题 2:

你有一个 $N \times N$ 的棋盘，每个格子内有一个整数，初始时的时候全部为 0，现在需要维护两种操作:

- 将格子`x`,`y`里的数字加上 A。
- 查询子矩形权值和。

```
const int N=5e5+5, INF=0x3f3f3f3f;

struct Point{
    int x[2], w;
};

struct Node{
    int l, r;
    Point P;
    int L[2], R[2], sum, sz;

    #define ls tr[u].l
    #define rs tr[u].r
}tr[N];

int n;

int idx, root;
int buf[N], tot;
int add(){
    if(!tot) return ++idx;
    return buf[tot--];
}
```

```
void pushup(int u){
    auto &L=tr[ls], &R=tr[rs];
    tr[u].sum=tr[u].P.w+L.sum+R.sum, tr[u].sz=L.sz+R.sz+1;

    rep(i,0,1){
        tr[u].L[i]=min(tr[u].P.x[i], min(L.L[i], R.L[i]));
        tr[u].R[i]=max(tr[u].P.x[i], max(L.R[i], R.R[i]));
    }
}

const double Al=0.75;

Point pt[N];

void getSeq(int u, int cnt){
    if(ls) getSeq(ls, cnt);
    buf[++tot]=u, pt[tr[ls].sz+1+cnt]=tr[u].P;
    if(rs) getSeq(rs, cnt+tr[ls].sz+1);
}

int rebuild(int l, int r, int k){
    if(l>r) return 0;
    int mid=l+r>>1;
    int u=add();

    nth_element(pt+l, pt+mid, pt+r+1, [&](Point a, Point b){
        return a.x[k]<b.x[k];
    });
    tr[u].P=pt[mid];

    ls=rebuild(l, mid-1, k^1), rs=rebuild(mid+1, r, k^1);
    pushup(u);
    return u;
}

void maintain(int &u, int k){
    if(tr[u].sz*Al<tr[ls].sz || tr[u].sz*Al<tr[rs].sz)
        getSeq(u, 0), u=rebuild(1, tot, k);
}

void insert(int &u, Point p, int k){
    if(!u){
        u=add();
        tr[u].l=tr[u].r=0;
        tr[u].P=p, pushup(u);
        return;
    }
    if(p.x[k]<=tr[u].P.x[k]) insert(ls, p, k^1);
    else insert(rs, p, k^1);
    pushup(u);
    maintain(u, k);
}

bool In(Node t, int x1, int y1, int x2, int y2){
    return t.L[0]>=x1 && t.R[0]<=x2 && t.L[1]>=y1 && t.R[1]<=y2;
}

bool In(Point p, int x1, int y1, int x2, int y2){
```

```
    return p.x[0]>=x1 && p.x[0]<=x2 && p.x[1]>=y1 && p.x[1]<=y2;
}

bool Out(Node t, int x1, int y1, int x2, int y2){
    return t.R[0]<x1 || t.L[0]>x2 || t.R[1]<y1 || t.L[1]>y2;
}

int query(int u, int x1, int y1, int x2, int y2){
    if(In(tr[u], x1, y1, x2, y2)) return tr[u].sum;
    if(Out(tr[u], x1, y1, x2, y2)) return 0;

    int res=0;
    if(In(tr[u].P, x1, y1, x2, y2)) res+=tr[u].P.w;
    res+=query(ls, x1, y1, x2, y2)+query(rs, x1, y1, x2, y2);
    return res;
}

int main(){
    cin>>n;
    // init
    tr[0].L[0]=tr[0].L[1]=INF;
    tr[0].R[0]=tr[0].R[1]=-INF;

    int res=0, op;
    while(cin>>op, op!=3){
        if(op==1){
            int x, y, k; read(x), read(y), read(k);
            insert(root, {x^res, y^res, k^res}, 0);
        }
        else{
            int x1, y1, x2, y2; read(x1), read(y1), read(x2), read(y2);
            write(res=query(root, x1^res, y1^res, x2^res, y2^res));
            puts("");
        }
    }

    return 0;
}
```

CDQ 分治

例题 1:

给定 n 个元素（编号 $1 \sim n$），其中第 i 个元素具有 a_i, b_i, c_i 三种属性。

设 $f(i)$ 表示满足以下 4 个条件:

1. $a_j \leq a_i$
2. $b_j \leq b_i$
3. $c_j \leq c_i$
4. $j \neq i$

的 j 的数量。

对于 $d \in [0, n)$，求满足 $f(i) = d$ 的 i 的数量。

```
const int N = 1e5 + 5, M = 2e5 + 5;
int n, m;
struct data
{
    int a, b, c, cnt, res;

    bool operator<(const data &o) const
    {
        if (a != o.a)
            return a < o.a;
        if (b != o.b)
            return b < o.b;
        return c < o.c;
    }

    bool operator==(const data &o) const
    {
        return a == o.a && b == o.b && c == o.c;
    }
} e[N], tmp[N];
int tot;
int tr[M];

int lowbit(int x) { return x & -x; }

void add(int p, int k)
{
    for (; p < M; p += lowbit(p))
        tr[p] += k;
}

int query(int p)
{
    int res = 0;
    for (; p; p -= lowbit(p))
        res += tr[p];
    return res;
}

void cdq(int l, int r)
{
    if (l >= r)
        return;
    int mid = l + r >> 1;
    cdq(l, mid), cdq(mid + 1, r);
    // 这里定义左区间对应的指针为 j，右区间对应的指针为 i.
    for (int j = l, i = mid + 1, k = l; k <= r; k++)
        if (i > r || j <= mid && e[j].b <= e[i].b)
            add(e[j].c, e[j].cnt), tmp[k] = e[j++]; // 如果说右区间的指针已经走到边界，或者左区间的b值比较小。
        else
            e[i].res += query(e[i].c), tmp[k] = e[i++];
    for (int j = l; j <= mid; j++)
        add(e[j].c, -e[j].cnt); // 恢复桶
    for (int k = l; k <= r; k++)
```

```
        e[k] = tmp[k]; // 完成排序，复制
}

int ans[M];

int main()
{
    cin >> n >> m;
    rep(i, 1, n)
    {
        int a, b, c;
        cin >> a >> b >> c;
        e[i] = {a, b, c, 1};
    }
    sort(e + 1, e + 1 + n);
    rep(i, 1, n) if (e[i] == e[tot]) e[tot].cnt++;
    else e[++tot] = e[i];
    cdq(1, tot);
    rep(i, 1, tot) ans[e[i].res + e[i].cnt - 1] += e[i].cnt;
    rep(i, 0, n - 1) cout << ans[i] << '\n';
    return 0;
}
```

例题 2：

二维 LIS

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

struct Node{
    int x, y, z;
}w[N];

int tr[N];

int lowbit(int x){
    return x&-x;
}

void upd(int x, int val){
    for(; x<N; x+=lowbit(x)) tr[x]=max(tr[x], val);
}

int query(int x){
    int res=0;
    for(; x; x-=lowbit(x)) res=max(res, tr[x]);
    return res;
}

void resume(int x){
    for(; x<N; x+=lowbit(x)) tr[x]=0;
}
```

```
int f[N];

int ql[N], qr[N];
void divi(int l, int r){
    if(l==r) return;
    int mid=l+r>>1;
    divi(l, mid);

    int cntl=0, cntr=0;
    rep(i,l,mid) ql[++cntl]=i;
    rep(i,mid+1,r) qr[++cntr]=i;

    auto cmp=[&](int a, int b){
        return w[a].y<w[b].y;
    };

    sort(ql+1, ql+cntl+1, cmp), sort(qr+1, qr+cntr+1, cmp);

    for(int i=1, j=1; j<=cntr; ){
        if(i<=cntl && w[ql[i]].y<w[qr[j]].y || j>cntr) upd(w[ql[i]].z,
f[ql[i]]), i++;
        else f[qr[j]]=max(query(w[qr[j]].z-1)+1, f[qr[j]]), j++;
    }
    // resume the buc
    rep(i,1,cntl) resume(w[ql[i]].z);
    divi(mid+1, r);
}

vector<int> di;
int find(int x){
    return lower_bound(all(di), x)-begin(di)+1;
}

int n;

int main(){
    int T; cin>>T;
    while(T--){
        cin>>n;
        rep(i,1,n){
            w[i].x=1;
            read(w[i].y);
        }
        di.clear();
        rep(i,1,n) read(w[i].z), di.pb(w[i].z);

        sort(all(di));
        di.erase(unique(all(di)), end(di));
        rep(i,1,n) w[i].z=find(w[i].z);

        rep(i,1,n) f[i]=1;
        divi(1, n);

        int res=1;
        rep(i,1,n) res=max(res, f[i]);
        cout<<res<<endl;
    }
    return 0;
```

```
}
```

整体二分

例题 1：

给定一个含有 n 个数的序列 $a_1, a_2 \dots a_n$，需要支持两种操作：

- `Q l r k` 表示查询下标在区间 $[l, r]$ 中的第 k 小的数。
- `C x y` 表示将 a_x 改为 y。

```
const int N=3e5+5;

int n, m;
int t, w[N];
int res[N];

struct Msg{
    int id, x, y, z, ty;
}q[N], lq[N], rq[N];

int tr[N];
int lb(int x){
    return x&-x;
}

void upd(int x, int k){
    for(; x<N; x+=lb(x)) tr[x]+=k;
}

int query(int x){
    int res=0;
    for(; x; x-=lb(x)) res+=tr[x];
    return res;
}

void divi(int l, int r, int st, int ed){
    if(st>ed) return;
    if(l==r){
        rep(i,st,ed) if(q[i].id) res[q[i].id]=l;
        return;
    }

    int lt=0, rt=0;
    int mid=l+r>>1;
    rep(i,st,ed){
        if(!q[i].ty){
            int cnt=query(q[i].y)-query(q[i].x-1);
            if(q[i].z<=cnt) lq[++lt]=q[i];
            else q[i].z-=cnt, rq[++rt]=q[i];
        }
        else{
            if(q[i].y<=mid) upd(q[i].x, q[i].z), lq[++lt]=q[i];
```

```
            else rq[++rt]=q[i];
        }
    }

    rep(i,st,ed) if(q[i].ty && q[i].y<=mid) upd(q[i].x, -q[i].z);
    rep(i,1,lt) q[st-1+i]=lq[i];
    rep(i,1,rt) q[st-1+i+lt]=rq[i];
    divi(l, mid, st, st+lt-1), divi(mid+1, r, st+lt, ed);
}

bool vis[N];

int main(){
    cin>>n>>m;
    rep(i,1,n){
        int x; read(x); w[i]=x;
        q[++t]={0, i, x, 1, 1};
    }

    rep(i,1,m){
        char op; cin>>op;
        if(op=='Q'){
            int l, r, k; read(l), read(r), read(k);
            q[++t]={i, l, r, k, 0};
            vis[i]=true;
        }
        else{
            int x, y; read(x), read(y);
            q[++t]={0, x, w[x], -1, 1};
            w[x]=y;
            q[++t]={0, x, w[x], 1, 1};
        }
    }

    divi(0, 1e9+5, 1, t);
    rep(i,1,m) if(vis[i]) cout<<res[i]<<'\n';

    return 0;
}
```

例题 2：

Byteotian Interstellar Union 有 N 个成员国。

现在它发现了一颗新的星球，这颗星球的轨道被分为 M 份（第 M 份和第 1 份相邻），第 i 份上有第 A_i 个国家的太空站。

这个星球经常会下陨石雨，BIU 已经预测了接下来 K 场陨石雨的情况。

BIU 的第 i 个成员国希望能够收集 P_i 单位的陨石样本。

你的任务是判断对于每个国家，它需要在第几次陨石雨之后，才能收集足够的陨石。

```
const int N=6e5+50;
```

```
int n, m, K;
int p[N], cnt[N];

struct Rain{
    int l, r, val;
}ra[N];

vector<int> b[N];
int t;
int res[N];

struct Node{
    int id, x;
}q[N], lq[N], rq[N];

ll tr[N];
int lb(int x){
    return x&-x;
}

void upd(int x, int k){
    for(; x<N; x+=lb(x)) tr[x]+=k;
}

ll query(int x){
    ll res=0;
    for(; x; x-=lb(x)) res+=tr[x];
    return res;
}

void divi(int l, int r, int st, int ed){
    if(st>ed) return;
    if(l==r){
        rep(i,st,ed) res[q[i].id]=l;
        return;
    }

    int mid=l+r>>1;
    int lt=0, rt=0;

    rep(i,l,mid){
        int l=ra[i].l, r=ra[i].r, val=ra[i].val;
        upd(l, val), upd(r+1, -val);
    }

    rep(i,st,ed){
        ll cnt=0;
        for(auto j: b[q[i].id]){
            ll t=query(j)+query(j+m);
            cnt+=t;
            if(cnt>=q[i].x) break;
        }

        if(cnt>=q[i].x) lq[++lt]=q[i];
        else q[i].x-=cnt, rq[++rt]=q[i];
    }

    rep(i,l,mid){
```

```
        int l=ra[i].l, r=ra[i].r, val=ra[i].val;
        upd(l, -val), upd(r+1, val);
    }

    rep(i,1,lt) q[st+i-1]=lq[i];
    rep(i,1,rt) q[st+i-1+lt]=rq[i];
    divi(l, mid, st, st+lt-1), divi(mid+1, r, st+lt, ed);
}

int main(){
    cin>>n>>m;
    rep(i,1,m){
        int x; read(x);
        b[x].pb(i);
    }
    rep(i,1,n) read(p[i]);

    cin>>K;
    rep(i,1,K){
        int l, r, d; read(l), read(r), read(d);
        if(l>r) r+=m;
        ra[i]={l, r, d};
    }
    rep(i,1,n) q[i]={i, p[i]};

    divi(1, K+1, 1, n);

    rep(i,1,n){
        if(res[i]>K) puts("NIE");
        else cout<<res[i]<<endl;
    }

    return 0;
}
```