ACM数据结构

1.并查集：带权并查集 维护相对关系。 记得赋初始值

int find(int x)

{

if (x != parent[x])

{

int t = parent[x];

parent[x] = find(parent[x]);

value[x] += value[t];

}

return parent[x];

}

int px = find(x);

int py = find(y);

if (px != py)

{

parent[px] = py;

value[px] = -value[x] + value[y] + s;

}

种类并查集可以开大几倍的数组，之后x+n表示一种关系，x+2n表示一种关系，以此类推。

2.可持久化并查集（回到某一个版本）

#include<bits/stdc++.h>

#define max(x,y) ((x)>(y)?(x):(y))

#define min(x,y) ((x)<(y)?(x):(y))

#define LL long long

#define swap(x,y) (x^y?(x^=y,y^=x,x^=y):0)

#define tc() (A==B&&(B=(A=ff)+fread(ff,1,100000,stdin),A==B)?EOF:\*A++)

#define pc(ch) (pp\_<100000?pp[pp\_++]=(ch):(fwrite(pp,1,100000,stdout),pp[(pp\_=0)++]=(ch)))

#define N 200000

int pp\_=0;char ff[100000],\*A=ff,\*B=ff,pp[100000];

using namespace std;

int n,Q,tot=0,rt[N+5],a[N+5];

struct Chairman\_Tree

{

int Son[2],fa,level;

}node[N\*20];

inline void read(int &x)

{

x=0;int f=1;char ch;

while(!isdigit(ch=tc())) f=ch^'-'?1:-1;

while(x=(x<<3)+(x<<1)+ch-'0',isdigit(ch=tc()));

x\*=f;

}

inline void write(int x)

{

if(x<0) pc('-'),x=-x;

if(x>9) write(x/10);

pc(x%10+'0');

}

inline void Build(int &rt,int l,int r)//初始的建树，一开始每个节点的fa都是本身，这是并查集的基础思想

{

rt=++tot;

int mid=l+r>>1;

if(!(l^r)) {node[rt].fa=l;return;}

Build(node[rt].Son[0],l,mid),Build(node[rt].Son[1],mid+1,r);

}

inline void NewPoint(int &rt,int lst,int l,int r,int x,int fa)//新插入一个节点

{

rt=++tot;

int mid=l+r>>1;

if(!(l^r)) {node[rt].fa=fa,node[rt].level=node[lst].level;return;}//更新fa，并复制以前版本的这个节点的level

node[rt].Son[0]=node[lst].Son[0],node[rt].Son[1]=node[lst].Son[1];

if(x<=mid) NewPoint(node[rt].Son[0],node[lst].Son[0],l,mid,x,fa);

else NewPoint(node[rt].Son[1],node[lst].Son[1],mid+1,r,x,fa);

}

inline void Add\_level(int rt,int l,int r,int x)//增加一个节点的在按秩合并时的优先级

{

int mid=l+r>>1;

if(!(l^r)) {++node[rt].level;return;}

if(x<=mid) Add\_level(node[rt].Son[0],l,mid,x);

else Add\_level(node[rt].Son[1],mid+1,r,x);

}

inline int Query(int rt,int l,int r,int x)//询问x节点在某一版本下的位置

{

int mid=l+r>>1;

if(!(l^r)) return rt;

if(x<=mid) return Query(node[rt].Son[0],l,mid,x);

else return Query(node[rt].Son[1],mid+1,r,x);

}

inline int getfa(int rt,int x)//询问x节点在某一版本下的祖先

{

int fa=Query(rt,1,n,x);

return node[fa].fa^x?getfa(rt,node[fa].fa):fa;//如果x节点在该版本下的父亲等于它本身，就返回x，否则返回x的父亲在这个版本下的祖先，和经典的getfa()函数差不多

}

inline void connect(int v,int x,int y)//在版本v中连接x和y，将他们放入一个集合中

{

int fx=getfa(rt[v],x),fy=getfa(rt[v],y);//先求出版本v中它们的祖先

if(!(fx^fy)) return;//如果祖先相同，就退出函数

if(node[fx].level<node[fy].level) swap(fx,fy);//如果x的优先级小于y的优先级，就交换x和y

NewPoint(rt[v],rt[v-1],1,n,node[fy].fa,node[fx].fa);//将优先级小的节点的父亲连向优先级大的节点的父亲

if(!(node[fx].level^node[fy].level)) Add\_level(rt[v],1,n,node[fx].fa);//如果它们的优先级相同，就将它们合并后的祖宗的优先级加1

}

int main()

{

register int i;

for(read(n),read(Q),Build(rt[0],i=1,n);i<=Q;++i)//先建一棵树，然后进行操作

{

int op,x,y;read(op),read(x);

if(op^2) read(y),rt[i]=rt[i-1];

switch(op)

{

case 1:connect(i,x,y);break;//在当前版本下连接x和y

case 2:rt[i]=rt[x];break;//将当前版本还原回曾经的版本x

case 3:pc(getfa(rt[i],x)^getfa(rt[i],y)?'0':'1'),pc('\n');break;//若当前版本下x和y的父亲相同，输出1，否则输出0

}

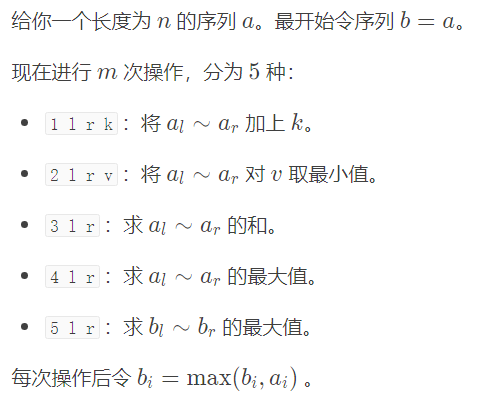
}

return fwrite(pp,1,pp\_,stdout),0;

}

矩形计数的本质是数线段个数。

线段树标记永久化：顺序无关（有交换律）

3.区间最值与历史最值（线段树模板3）

C区间加，时间戳加QH查询不加时间，B时间戳置为t

int n,m,op,l,r,k,v;

struct segment\_tree{

ll sum;

int l,r,maxa,cnt,se,maxb;

int add1,add2,add3,add4;

}s[2000005];

inline void push\_up(int p)

{

s[p].sum=s[p\*2].sum+s[p\*2+1].sum;

s[p].maxa=max(s[p\*2].maxa,s[p\*2+1].maxa);

s[p].maxb=max(s[p\*2].maxb,s[p\*2+1].maxb);

if(s[p\*2].maxa==s[p\*2+1].maxa)

{

s[p].se=max(s[p\*2].se,s[p\*2+1].se);

s[p].cnt=s[p\*2].cnt+s[p\*2+1].cnt;

}

else if(s[p\*2].maxa>s[p\*2+1].maxa)

{

s[p].se=max(s[p\*2].se,s[p\*2+1].maxa);

s[p].cnt=s[p\*2].cnt;

}

else

{

s[p].se=max(s[p\*2].maxa,s[p\*2+1].se);

s[p].cnt=s[p\*2+1].cnt;

}

}

void build(int l,int r,int p)

{

s[p].l=l,s[p].r=r;

if(l==r)

{

s[p].sum=s[p].maxa=s[p].maxb=read();

s[p].cnt=1,s[p].se=-2e9;

return;

}

int mid=(l+r)/2;

build(l,mid,p\*2);

build(mid+1,r,p\*2+1);

push\_up(p);

}

inline void change(int k1,int k2,int k3,int k4,int p)

{

s[p].sum+=1ll\*k1\*s[p].cnt+1ll\*k2\*(s[p].r-s[p].l+1-s[p].cnt);

s[p].maxb=max(s[p].maxb,s[p].maxa+k3);

s[p].maxa+=k1;

if(s[p].se!=-2e9)s[p].se+=k2;

s[p].add3=max(s[p].add3,s[p].add1+k3);

s[p].add4=max(s[p].add4,s[p].add2+k4);

s[p].add1+=k1,s[p].add2+=k2;

}

inline void push\_down(int p)

{

int maxn=max(s[p\*2].maxa,s[p\*2+1].maxa);

if(s[p\*2].maxa==maxn)

change(s[p].add1,s[p].add2,s[p].add3,s[p].add4,p\*2);

else change(s[p].add2,s[p].add2,s[p].add4,s[p].add4,p\*2);

if(s[p\*2+1].maxa==maxn)

change(s[p].add1,s[p].add2,s[p].add3,s[p].add4,p\*2+1);

else change(s[p].add2,s[p].add2,s[p].add4,s[p].add4,p\*2+1);

s[p].add1=s[p].add2=s[p].add3=s[p].add4=0;

}

void update\_add(int p)

{

if(l>s[p].r||r<s[p].l)return;

if(l<=s[p].l&&s[p].r<=r)

{

s[p].sum+=1ll\*k\*s[p].cnt+1ll\*k\*(s[p].r-s[p].l+1-s[p].cnt);

s[p].maxa+=k;

s[p].maxb=max(s[p].maxb,s[p].maxa);

if(s[p].se!=-2e9)s[p].se+=k;

s[p].add1+=k,s[p].add2+=k;

s[p].add3=max(s[p].add3,s[p].add1);

s[p].add4=max(s[p].add4,s[p].add2);

return;

}

push\_down(p);

update\_add(p\*2),update\_add(p\*2+1);

push\_up(p);

}

void update\_min(int p)

{

if(l>s[p].r||r<s[p].l||v>=s[p].maxa)return;

if(l<=s[p].l&&s[p].r<=r&&s[p].se<v)

{

int k=s[p].maxa-v;

s[p].sum-=1ll\*s[p].cnt\*k;

s[p].maxa=v,s[p].add1-=k;

return;

}

push\_down(p);

update\_min(p\*2),update\_min(p\*2+1);

push\_up(p);

}

ll query\_sum(int p)

{

if(l>s[p].r||r<s[p].l)return 0;

if(l<=s[p].l&&s[p].r<=r)return s[p].sum;

push\_down(p);

return query\_sum(p\*2)+query\_sum(p\*2+1);

}

int query\_maxa(int p)

{

if(l>s[p].r||r<s[p].l)return -2e9;

if(l<=s[p].l&&s[p].r<=r)return s[p].maxa;

push\_down(p);

return max(query\_maxa(p\*2),query\_maxa(p\*2+1));

}

int query\_maxb(int p)

{

if(l>s[p].r||r<s[p].l)return -2e9;

if(l<=s[p].l&&s[p].r<=r)return s[p].maxb;

push\_down(p);

return max(query\_maxb(p\*2),query\_maxb(p\*2+1));

}

int main()

{

n=read(),m=read();

build(1,n,1);

while(m--)

{

op=read(),l=read(),r=read();

if(op==1)k=read(),update\_add(1);

else if(op==2)v=read(),update\_min(1);

else if(op==3)write(query\_sum(1));

else if(op==4)printf("%d\n",query\_maxa(1));

else printf("%d\n",query\_maxb(1));

}

return 0;

}

区间修改的可持久化线段树（需要标记永久化）

int root[maxn<<1],wh[maxn];

//wh记录每个时间戳对应哪个版本

int a[maxn];

struct Chairman\_Tree{

long long dat[maxn<<6],tag[maxn<<6];

int ls[maxn<<6],rs[maxn<<6],cnt;

#define ls(x) ls[x]

#define rs(x) rs[x]

inline void update(int node){

dat[node]=dat[ls(node)]+dat[rs(node)];

}

void build(int l,int r,int &node){

node=++cnt;

if(l==r){

dat[node]=a[l];

return;

}

int mid=l+r>>1;

build(l,mid,ls(node));

build(mid+1,r,rs(node));

update(node);

}//初始建树

void add(int L,int R,int l,int r,int &ne,int ol,int d){

ne=++cnt;

dat[ne]=dat[ol]+1ll\*(min(R,r)-max(L,l)+1)\*d;

tag[ne]=tag[ol];

if(L<=l&&R>=r){

tag[ne]+=(long long)d;

ls[ne]=ls[ol],rs[ne]=rs[ol];//继承子节点

return;

}

int mid=l+r>>1;

if(L<=mid)add(L,R,l,mid,ls(ne),ls(ol),d);

else ls(ne)=ls(ol);//不包含在这个区间内节点，直接继承过来

if(R>mid)add(L,R,mid+1,r,rs(ne),rs(ol),d);

else rs(ne)=rs(ol);

}

long long ask(int L,int R,int l,int r,int node){

if(L<=l&&R>=r)return dat[node];

int mid=l+r>>1;

long long ans=0;

if(L<=mid)ans=ask(L,R,l,mid,ls(node));

if(R>mid)ans+=ask(L,R,mid+1,r,rs(node));

return ans+1ll\*(min(R,r)-max(L,l)+1)\*tag[node];

}

}ct;

int main(){

int n=read(),m=read(),t=0,now=0;

//now是当前的时间戳，t是当前的版本

for(register int i=1;i<=n;++i)

a[i]=read();

ct.build(1,n,root[0]);

while(m--){

char s[2];

scanf("%s",s);

if(s[0]=='C'){

wh[++now]=++t;

int l=read(),r=read(),d=read();

ct.add(l,r,1,n,root[t],root[t-1],d);

}

else if(s[0]=='Q'){

int l=read(),r=read();

printf("%lld\n",ct.ask(l,r,1,n,root[t]));

}

else if(s[0]=='H'){

int l=read(),r=read(),k=read();

printf("%lld\n",ct.ask(l,r,1,n,root[wh[k]]));

}

else now=read(),root[++t]=root[wh[now]];

}

return 0;

}

点分治求路径长度小于等于k的个数（容斥原理）

void getrt(ll u,ll pa){

siz[u]=1;

maxp[u]=0;

for(int i=head[u];i;i=nex[i]){

ll y=to[i];

if(y==pa||vis[y]){

continue;

}

getrt(y,u);

siz[u]+=siz[y];

maxp[u]=max(maxp[u],siz[y]);

}

maxp[u]=max(maxp[u],sum-siz[u]);

if(maxp[u]<maxp[rt]){

rt=u;

}

}

void calc(int u,ll fa){

++num;

rem[num]=dis[u];

for(int i=head[u];i;i=nex[i]){

int v=to[i];

if(vis[v]||v==fa){

continue;

}

dis[v]=val[i]+dis[u];

calc(v,u);

}

}

ll work(){

if(num==0){

return 0;

}

sort(rem+1,rem+1+num);

for(int i=1;i<=num;++i){

cout<<rem[i]<<" ";

}cout<<endl;

ll l=1,r=num;

ll tmp=0;

while(l<=r){

if(rem[l]+rem[r]<=k){

tmp+=r-l;

l++;

}

else{

r--;

}

}

return tmp;

}

void solve(int u){

num=0;

vis[u]=1;

dis[u]=0;

calc(u,0);

ans+=work();

for(int i=head[u];i;i=nex[i]){

ll y=to[i];

if(vis[y]){

continue;

}

num=0;

dis[y]=val[i];

calc(y,u);

ans-=work();

sum=siz[y];

maxp[rt=0]=n;

getrt(y,0);

solve(y);

}

}

int main(){

n=read();

for(int i=1;i<=n-1;++i){

ll x,y,z;

x=read();

y=read();

z=read();

add(x,y,z);

add(y,x,z);

}

k=read();

maxp[rt]=sum=n;

getrt(1,0);

solve(rt);

printf("%lld",ans);

return 0;

}

线段树二分求异或值前k大的数对和

inline ll Add(ll x,ll y) {return ((x+y)%mod+mod)%mod;}

inline ll pow(ll x,ll p) {

ll res=1;

for(;p;p>>=1) {if(p&1) res=res\*x%mod;x=x\*x%mod;}

return res;

}

inline void insert(int val,int id) {

if(!rt) rt=++tot,l[rt]=1,r[rt]=n;

int p=rt;++size[rt];

for(register int d=31;d>=0;--d) {

int &nxt=ch[p][val>>d&1];

if(!nxt) {nxt=++tot;l[nxt]=id;}

p=nxt;r[p]=id;++size[p];

}

}

inline int getVal(ll &k) {

int res=0;

for(register int i=1;i<=n;++i) tmp[i]=rt;

for(register int d=31;d>=0;--d) {//若当前取1的个数为cnt,cnt>k

int x=0;

ll cnt=0;

for(register int i=1;i<=n;++i) cnt+=size[ch[tmp[i]][(a[i]>>d&1)^1]];

if(cnt>=k) res|=(1<<d),x=1;//cnt<k

else k-=cnt,x=0;

for(register int i=1;i<=n;++i) tmp[i]=ch[tmp[i]][(a[i]>>d&1)^x];

}

return res;

}

inline ll ask(int l,int r,int val) {

if(!l||!r) return 0;

ll res=0;

for(register int d=31;d>=0;--d) res=Add(res,(ll)(sum[d][r][(val>>d&1)^1]-sum[d][l-1][(val>>d&1)^1])\*((1ll<<d)%mod)%mod);

return res;

}

inline ll getSum(ll k) {

int val=getVal(k);

ll res=val\*k%mod;

for(register int i=1;i<=n;++i) tmp[i]=rt;

for(register int d=31;d>=0;--d) {

int x=val>>d&1;

if(!x) {for(register int i=1;i<=n;++i) res=Add(res,ask(l[ch[tmp[i]][(a[i]>>d&1)^1]],r[ch[tmp[i]][(a[i]>>d&1)^1]],a[i]));}

for(register int i=1;i<=n;++i) tmp[i]=ch[tmp[i]][(a[i]>>d&1)^x];

}

return res;

}

signed main() {

n=read();

ll k=read()\*2ll;//25\*1e8=2.5\*1e9

for(register int i=1;i<=n;++i) a[i]=read();

std::sort(a+1,a+1+n);

for(register int i=1;i<=n;++i) insert(a[i],i);

for(register int i=1;i<=n;++i) {

for(register int d=31;d>=0;--d) {

sum[d][i][0]=sum[d][i-1][0];

sum[d][i][1]=sum[d][i-1][1];

++sum[d][i][a[i]>>d&1];

}

}

printf("%lld\n",getSum(k)\*pow(2,mod-2)%mod);

return 0;

}

等差数列max*a*−min*a*≤(*r*−*l*+*k*)*d* 且 [*l*,*r*] 不含相同的数且 [*l*,*r*] 模 *d* 均相同。

异或01trie树可持久化，动态区间异或最大值（记得初始建树）

inline void insert(const int i,const int k,const int p,const int q) {

if (k<0) { latest[q]=i; return; }

int c=s[i]>>k&1;

if (p) trie[q][c^1]=trie[p][c^1];

trie[q][c]=++tot;

insert(i,k-1,trie[p][c],trie[q][c]);

latest[q]=max(latest[trie[q][0]],latest[trie[q][1]]);

}

inline int query(const int now,const int val,const int k,const int L) {

if (k<0) return s[latest[now]]^val;

int c=val>>k&1;

if (latest[trie[now][c^1]]>=L) return query(trie[now][c^1],val,k-1,L);

else return query(trie[now][c],val,k-1,L);

}

int main() {

read(n),read(m),latest[0]=-1,root[0]=++tot;

insert(0,23,0,root[0]);

for (re int i=1,x;i<=n;++i) {

read(x),s[i]=s[i-1]^x,root[i]=++tot;

insert(i,23,root[i-1],root[i]);

}

for (re int i=1,x,l,r;i<=m;++i) {

char op=gc(); while (!isalpha(op)) op=gc();

if (op=='A') {

read(x),root[++n]=++tot,s[n]=s[n-1]^x;

insert(n,23,root[n-1],root[n]);

} else {

read(l),read(r),read(x);

print(query(root[r-1],s[n]^x,23,l-1)),putc('\n');

}

}

树链剖分分分分

#define Rint register int

#define mem(a,b) memset(a,(b),sizeof(a))

#define Temp template<typename T>

using namespace std;

typedef long long LL;

Temp inline void read(T &x){

x=0;T w=1,ch=getchar();

while(!isdigit(ch)&&ch!='-')ch=getchar();

if(ch=='-')w=-1,ch=getchar();

while(isdigit(ch))x=(x<<3)+(x<<1)+(ch^'0'),ch=getchar();

x=x\*w;

}

#define mid ((l+r)>>1)

#define lson rt<<1,l,mid

#define rson rt<<1|1,mid+1,r

#define len (r-l+1)

const int maxn=200000+10;

int n,m,r,mod;

//见题意

int e,beg[maxn],nex[maxn],to[maxn],w[maxn],wt[maxn];

//链式前向星数组，w[]、wt[]初始点权数组

int a[maxn<<2],laz[maxn<<2];

//线段树数组、lazy操作

int son[maxn],id[maxn],fa[maxn],cnt,dep[maxn],siz[maxn],top[maxn];

//son[]重儿子编号,id[]新编号,fa[]父亲节点,cnt dfs\_clock/dfs序,dep[]深度,siz[]子树大小,top[]当前链顶端节点

int res=0;

//查询答案

inline void add(int x,int y){//链式前向星加边

to[++e]=y;

nex[e]=beg[x];

beg[x]=e;

}

//-------------------------------------- 以下为线段树

inline void pushdown(int rt,int lenn){

laz[rt<<1]+=laz[rt];

laz[rt<<1|1]+=laz[rt];

a[rt<<1]+=laz[rt]\*(lenn-(lenn>>1));

a[rt<<1|1]+=laz[rt]\*(lenn>>1);

a[rt<<1]%=mod;

a[rt<<1|1]%=mod;

laz[rt]=0;

}

inline void build(int rt,int l,int r){

if(l==r){

a[rt]=wt[l];

if(a[rt]>mod)a[rt]%=mod;

return;

}

build(lson);

build(rson);

a[rt]=(a[rt<<1]+a[rt<<1|1])%mod;

}

inline void query(int rt,int l,int r,int L,int R){

if(L<=l&&r<=R){res+=a[rt];res%=mod;return;}

else{

if(laz[rt])pushdown(rt,len);

if(L<=mid)query(lson,L,R);

if(R>mid)query(rson,L,R);

}

}

inline void update(int rt,int l,int r,int L,int R,int k){

if(L<=l&&r<=R){

laz[rt]+=k;

a[rt]+=k\*len;

}

else{

if(laz[rt])pushdown(rt,len);

if(L<=mid)update(lson,L,R,k);

if(R>mid)update(rson,L,R,k);

a[rt]=(a[rt<<1]+a[rt<<1|1])%mod;

}

}

//---------------------------------以上为线段树

inline int qRange(int x,int y){

int ans=0;

while(top[x]!=top[y]){//当两个点不在同一条链上

if(dep[top[x]]<dep[top[y]])swap(x,y);//把x点改为所在链顶端的深度更深的那个点

res=0;

query(1,1,n,id[top[x]],id[x]);//ans加上x点到x所在链顶端 这一段区间的点权和

ans+=res;

ans%=mod;//按题意取模

x=fa[top[x]];//把x跳到x所在链顶端的那个点的上面一个点

}

//直到两个点处于一条链上

if(dep[x]>dep[y])swap(x,y);//把x点深度更深的那个点

res=0;

query(1,1,n,id[x],id[y]);//这时再加上此时两个点的区间和即可

ans+=res;

return ans%mod;

}

inline void updRange(int x,int y,int k){//同上

k%=mod;

while(top[x]!=top[y]){

if(dep[top[x]]<dep[top[y]])swap(x,y);

update(1,1,n,id[top[x]],id[x],k);

x=fa[top[x]];

}

if(dep[x]>dep[y])swap(x,y);

update(1,1,n,id[x],id[y],k);

}

inline int qSon(int x){

res=0;

query(1,1,n,id[x],id[x]+siz[x]-1);//子树区间右端点为id[x]+siz[x]-1

return res;

}

inline void updSon(int x,int k){//同上

update(1,1,n,id[x],id[x]+siz[x]-1,k);

}

inline void dfs1(int x,int f,int deep){//x当前节点，f父亲，deep深度

dep[x]=deep;//标记每个点的深度

fa[x]=f;//标记每个点的父亲

siz[x]=1;//标记每个非叶子节点的子树大小

int maxson=-1;//记录重儿子的儿子数

for(Rint i=beg[x];i;i=nex[i]){

int y=to[i];

if(y==f)continue;//若为父亲则continue

dfs1(y,x,deep+1);//dfs其儿子

siz[x]+=siz[y];//把它的儿子数加到它身上

if(siz[y]>maxson)son[x]=y,maxson=siz[y];//标记每个非叶子节点的重儿子编号

}

}

inline void dfs2(int x,int topf){//x当前节点，topf当前链的最顶端的节点

id[x]=++cnt;//标记每个点的新编号

wt[cnt]=w[x];//把每个点的初始值赋到新编号上来

top[x]=topf;//这个点所在链的顶端

if(!son[x])return;//如果没有儿子则返回

dfs2(son[x],topf);//按先处理重儿子，再处理轻儿子的顺序递归处理

for(Rint i=beg[x];i;i=nex[i]){

int y=to[i];

if(y==fa[x]||y==son[x])continue;

dfs2(y,y);//对于每一个轻儿子都有一条从它自己开始的链

}

}

int main(){

read(n);read(m);read(r);read(mod);

for(Rint i=1;i<=n;i++)read(w[i]);

for(Rint i=1;i<n;i++){

int a,b;

read(a);read(b);

add(a,b);add(b,a);

}

dfs1(r,0,1);

dfs2(r,r);

build(1,1,n);

while(m--){

int k,x,y,z;

read(k);

if(k==1){

read(x);read(y);read(z);

updRange(x,y,z);

}

else if(k==2){

read(x);read(y);

printf("%d\n",qRange(x,y));

}

else if(k==3){

read(x);read(y);

updSon(x,y);

}

else{

read(x);

printf("%d\n",qSon(x));

树上主席树板子

void build(P &x,ll l,ll r){

x.sum=0;

if(l==r){

return;

}

ll mid=(l+r)>>1;

x.l=++tot;

build(node[x.l],l,mid);

x.r=++tot;

build(node[x.r],mid+1,r);

}

void insert(P pas,P &x,ll l,ll r,ll z){

x.sum=pas.sum+1;

if(l==r){

return;

}

ll mid=(l+r)>>1;

if(z<=mid){

x.l=++tot;

insert(node[pas.l],node[x.l],l,mid,z);

x.r=pas.r;

}

if(z>mid){

x.r=++tot;

ins

ert(node[pas.r],node[x.r],mid+1,r,z);

x.l=pas.l;

}

}

void dfs(ll x,ll fa){

root[x]=++tot;

insert(node[root[fa]],node[root[x]],1,N,w[x]);

f[x]=fa;

dep[x]=dep[fa]+1;

siz[x]=1;

ll maxson=-1;

for(int i=head[x];i;i=nex[i]){

int y=to[i];

if(y==fa){

continue;

}

dfs(y,x);

siz[x]+=siz[y];

if(siz[y]>maxson){

son[x]=y;

maxson=siz[y];

}

}

}

void dfs2(ll x,ll topf){

top[x]=topf;

if(!son[x]){

return;

}

dfs2(son[x],topf);

for(int i=head[x];i;i=nex[i]){

ll y=to[i];

if(y==f[x]||y==son[x]){

continue;

}

dfs2(y,y);

}

}

ll LCA(ll x,ll y){

while(top[x]!=top[y]){

if(dep[top[x]]<dep[top[y]]){

swap(x,y);

}

x=f[top[x]];

}

if(dep[x]>dep[y]){

swap(x,y);

}

return x;

}

ll ask(P x,P y,P z,P d,ll l,ll r,ll k){

if(l==r){

return l;

}

ll res=node[x.l].sum+node[y.l].sum-node[z.l].sum-node[d.l].sum;

ll mid=(l+r)>>1;

if(res>=k){

return ask(node[x.l],node[y.l],node[z.l],node[d.l],l,mid,k);

}else{

return ask(node[x.r],node[y.r],node[z.r],node[d.r],mid+1,r,k-res);

}

}

int main(){

n=read();

m=read();

for(int i=1;i<=n;++i){

a[i]=read();

b[i]=a[i];

}

for(int i=1;i<=n-1;++i){

ll x,y;

x=read();

y=read();

add(x,y);

add(y,x);

}

sort(b+1,b+1+n);

N = unique(b+1,b+1+n)-(b+1);

for(int i=1;i<=n;++i){

w[i]=lower\_bound(b+1,b+1+N,a[i])-b;

}

root[0]=++tot;

build(node[tot],1,N);dfs(1,0);dfs2(1,1);

for(int i=1;i<=m;++i){

ll u,v,k,lc;

u=read();

v=read();

k=read();

u=u^ans;

lc=LCA(u,v);

ll tmp = ask(node[root[u]],node[root[v]],node[root[lc]],node[root[f[lc]]],1,N,k);

ans=b[tmp];

printf("%lld\n",ans);

}

树套树板子（动态区间第k小）

const int MAX=10005;

struct segment\_tree{int v;int ls,rs;}t[MAX\*400];//线段树开nlogn大小

struct operation{bool b;int l,r,k;int pos,t;}q[MAX];//因为要离散花所以要把所有数据输进来离线搞

int n,m,a[MAX],o[MAX<<1],rt[MAX],len,tot,temp[2][20],cnt[2];

char opt;

void Modify(int &now,int l,int r,int pos,int val)

{

if (!now) now=++tot;

t[now].v+=val;

if (l==r) return;

int mid=l+r>>1;

if (pos<=mid) Modify(t[now].ls,l,mid,pos,val);

else Modify(t[now].rs,mid+1,r,pos,val);

}

void prepare\_Modify(int x,int val)

{

int k=lower\_bound(o+1,o+len+1,a[x])-o;

for (int i=x;i<=n;i+=i&-i) Modify(rt[i],1,len,k,val);//处理出需要修改哪log棵主席树

}

int Query(int l,int r,int k)

{

if (l==r) return l;

int mid=l+r>>1,sum=0;

for (int i=1;i<=cnt[1];i++) sum+=t[t[temp[1][i]].ls].v;

for (int i=1;i<=cnt[0];i++) sum-=t[t[temp[0][i]].ls].v;

if (k<=sum)

{

for (int i=1;i<=cnt[1];i++) temp[1][i]=t[temp[1][i]].ls;

for (int i=1;i<=cnt[0];i++) temp[0][i]=t[temp[0][i]].ls;

return Query(l,mid,k);

}

else

{

for (int i=1;i<=cnt[1];i++) temp[1][i]=t[temp[1][i]].rs;

for (int i=1;i<=cnt[0];i++) temp[0][i]=t[temp[0][i]].rs;

return Query(mid+1,r,k-sum);

}

}

int prepare\_Query(int l,int r,int k)

{

memset(temp,0,sizeof(temp));//同修改，处理出需要进行相减操作的是哪log棵主席树

cnt[0]=cnt[1]=0;

for (int i=r;i;i-=i&-i) temp[1][++cnt[1]]=rt[i];

for (int i=l-1;i;i-=i&-i) temp[0][++cnt[0]]=rt[i];

return Query(1,len,k);

}

int main()

{

ios::sync\_with\_stdio(false);

cin>>n>>m;

for (int i=1;i<=n;i++) cin>>a[i],o[++len]=a[i];

for (int i=1;i<=m;i++)

{

cin>>opt;

q[i].b=(opt=='Q');

if (q[i].b) cin>>q[i].l>>q[i].r>>q[i].k;

else cin>>q[i].pos>>q[i].t,o[++len]=q[i].t;

}

sort(o+1,o+len+1);

len=unique(o+1,o+len+1)-o-1;//离散 —— 排序 + 去重

for (int i=1;i<=n;i++) prepare\_Modify(i,1);

for (int i=1;i<=m;i++)

{

if (q[i].b) printf("%d\n",o[prepare\_Query(q[i].l,q[i].r,q[i].k)]);

else

{

prepare\_Modify(q[i].pos,-1);

a[q[i].pos]=q[i].t;

prepare\_Modify(q[i].pos,1);

}

}

return 0;

}