## BIT

struct Bit{

int n,treenum[maxn];

void init(int k){

n=k;

memset(treenum,0,sizeof treenum);

}

int lowbit(int x){

return x&(-x);

}

void add(int i,int add){

while(i<=n){

treenum[i]+=add;

i+=lowbit(i);

}

}

int Sum(int i){

int sum=0;

while(i){

sum+=treenum[i];

i-=lowbit(i);

}

return sum;

}

};

int n,m; ll treeNum[maxn][maxn][maxn];

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

ll getSum(int x,int y,int z){

ll sum=0;

for(int i=x; i>0; i-=lowbit(i))

for(int j=y; j>0; j-=lowbit(j))

for(int k=z; k>0; k-=lowbit(k))

sum+=treeNum[i][j][k];

return sum;

}

void add(int x,int y,int z,int val){

for(int i=x; i<maxn; i+=lowbit(i))

for(int j=y; j<maxn; j+=lowbit(j))

for(int k=z; k<maxn; k+=lowbit(k))

treeNum[i][j][k]+=val;

}

## 2.RMQ

int a[maxn],Max[maxn][20],Min[maxn][20];

void rmq\_init(int l,int r){

for(int i=l; i<=r; i++)

Max[i][0]=Min[i][0]=a[i];

for(int j=1; l+(1<<j)-1<=r; j++)

for(int i=l; i+(1<<j)-1<=r; i++){

Max[i][j]=max(Max[i][j-1],Max[i+(1<<(j-1))][j-1]);

Min[i][j]=min(Min[i][j-1],Min[i+(1<<(j-1))][j-1]);

}

}

int rmq\_max(int l,int r){

int k=log2(r-l+1);

return max(Max[l][k],Max[r-(1<<k)+1][k]);

}

int rmq\_min(int l,int r){

int k=log2(r-l+1);

return min(Min[l][k],Min[r-(1<<k)+1][k]);

}

## 3.线段树

//单点更新+- 查询区间和

int n;

int a[maxn];

struct Node{

int l,r;

int sum;

};

struct SegTree{

Node st[maxn\*3];

void build(int i,int l,int r){

st[i].l=l;

st[i].r=r;

if(l==r){

st[i].sum=a[l];

return ;

}

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

build(i<<1,l,mid);

build((i<<1)|1,mid+1,r);

push\_up(i);

}

void update(int i,int k,int add){

st[i].sum+=add;

if(st[i].l==st[i].r)

return;

int mid=(st[i].l+st[i].r)>>1;

if(k<=mid)

update(i<<1,k,add);

else

update((i<<1)|1,k,add);

}

int query(int i,int l,int r){

if(st[i].l==l&&st[i].r==r)

return st[i].sum;

int mid=(st[i].l+st[i].r)>>1;

if(r<=mid)

return query(i<<1,l,r);

else if(l>mid)

return query((i<<1)|1,l,r);

else

return query(i<<1,l,mid)+query((i<<1)|1,mid+1,r);

}

//回溯更新父节点

void push\_up(int i){

st[i].sum=st[i<<1].sum+st[i<<1|1].sum;

}

}st;

//区间更新为同一值 求和

struct Node{

int l,r;

int sum;

int lazy,tag;

};

int n;

ll a[maxn];

struct SegTree{

Node st[maxn\*3];

void build(int i,int l,int r){

st[i].l=l;

st[i].r=r;

st[i].lazy=st[i].tag=0;

if(l==r){

st[i].sum=1;

return ;

}

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

build(i<<1,l,mid);

build((i<<1)|1,mid+1,r);

push\_up(i);

}

void update(int i,int l,int r,int val){

if(st[i].l==l&&st[i].r==r){

st[i].lazy=1;

st[i].tag=val;

st[i].sum=(r-l+1)\*val;

return ;

}

int mid=(st[i].l+st[i].r)>>1;

if(st[i].lazy==1){

st[i].lazy=0;

update(i<<1,st[i].l,mid,st[i].tag);

update((i<<1)|1,mid+1,st[i].r,st[i].tag);

st[i].tag=0;

}

if(r<=mid)

update(i<<1,l,r,val);

else if(l>mid)

update((i<<1)|1,l,r,val);

else

update(i<<1,l,mid,val),update((i<<1)|1,mid+1,r,val);

push\_up(i);

}

//回溯更新父节点

void push\_up(int i){

st[i].sum=st[i<<1].sum+st[i<<1|1].sum;

}

}st;

struct Node{

int l,r;

ll val;

ll lazy;

};

int n;

ll a[maxn];

struct SegTree{

Node st[maxn\*3];

void build(int i,int l,int r){

st[i].l=l;

st[i].r=r;

st[i].val=0;

if(l==r){

st[i].val=a[l];

st[i].lazy=st[i].val;

return ;

}

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

build(i<<1,l,mid);

build((i<<1)|1,mid+1,r);

push\_up(i);

}

void update(int i,int l,int r,ll val){

if((st[i].lazy&val)==st[i].lazy)

return;

if(st[i].l==st[i].r){

st[i].lazy&=val;

st[i].val&=val;

return ;

}

int mid=(st[i].l+st[i].r)>>1;

if(r<=mid)

update(i<<1,l,r,val);

else if(l>mid)

update((i<<1)|1,l,r,val);

else

update(i<<1,l,mid,val),update((i<<1)|1,mid+1,r,val);

push\_up(i);

}

ll query(int i,int l,int r){

if(st[i].l==l&&st[i].r==r)

return st[i].val;

int mid=(st[i].l+st[i].r)>>1;

if(r<=mid)

return query(i<<1,l,r);

else if(l>mid)

return query((i<<1)|1,l,r);

else

return min(query(i<<1,l,mid),query((i<<1)|1,mid+1,r));

}

//回溯更新父节点

void push\_up(int i){

st[i].val=min(st[i<<1].val,st[(i<<1)|1].val);

st[i].lazy=st[i<<1].lazy|st[(i<<1)|1].lazy;

}

}st;

## 4.SBT

struct SBT{

int left,right;

int size,key;

void init(){

left=right=0;

size=1;

}

}tree[maxn];

int tot,root;

void left\_rotate(int &x){

int y=tree[x].right;

tree[x].right=tree[y].left;

tree[y].left=x;

tree[y].size=tree[x].size;

tree[x].size=tree[tree[x].left].size+tree[tree[x].right].size+1;

x=y;

}

void right\_rotate(int &x){

int y=tree[x].left;

tree[x].left=tree[y].right;

tree[y].right=x;

tree[y].size=tree[x].size;

tree[x].size=tree[tree[x].left].size+tree[tree[x].right].size+1;

x=y;

}

void maintain(int &x,int flag){

if(flag==0){

if(tree[tree[tree[x].left].left].size>tree[tree[x].right].size)

right\_rotate(x);

else if(tree[tree[tree[x].left].right].size>tree[tree[x].right].size)

left\_rotate(tree[x].left),right\_rotate(x);

else

return;

}

else{

if(tree[tree[tree[x].right].right].size>tree[tree[x].left].size)

left\_rotate(x);

else if(tree[tree[tree[x].right].left].size>tree[tree[x].left].size)

right\_rotate(tree[x].right),left\_rotate(x);

else

return;

}

maintain(tree[x].left,0);

maintain(tree[x].right,1);

maintain(x,0);

maintain(x,1);

}

void insert(int &x,int key){

if(x==0){

x=++tot;

tree[x].init();

tree[x].key=key;

}

else{

tree[x].size++;

if(key<tree[x].key)

insert(tree[x].left,key);

else

insert(tree[x].right,key);

maintain(x,key>=tree[x].key);

}

}

//得到前驱，小于 前趋定义为小于x，且最大的数

//返回前驱的节点编号

int get\_pre(int &r,int y,int key){

if(r==0)

return y;

if(key>tree[r].key)//加个等号就是小于等于

return get\_pre(tree[r].right,r,key);

else

return get\_pre(tree[r].left,y,key);

}

//得到后继，大于 后继定义为大于x，且最小的数

//返回后继的节点编号

int get\_next(int &r,int y,int key){

if(r==0)

return y;

if(key<tree[r].key)//加个等号就是大于等于

return get\_next(tree[r].left,r,key);

else

return get\_next(tree[r].right,y,key);

}

//查找是否存在key值为k的元素

//不存在返回0，存在返回节点编号

int find(int t,int k){

while(t && k!=tree[t].key)

t=k<tree[t].key?find(tree[t].left,k):find(tree[t].right,k);

return t;

}

int get\_kth(int &x,int k){

if(k<=tree[tree[x].left].size)

return get\_kth(tree[x].left,k);

else if(k>tree[tree[x].left].size+1)

return get\_kth(tree[x].right,k-tree[tree[x].left].size-1);

return tree[x].key;

}

int get\_max(int r){

while(tree[r].right)

r=tree[r].right;

return r;

}

int get\_min(int r){

while(tree[r].left)

r=tree[r].left;

return r;

}

//删除值为key的数

int del(int &x,int key){

if(!x)

return 0;

tree[x].size--;

if(key==tree[x].key || (key<tree[x].key&&tree[x].left==0)

|| (key>tree[x].key&&tree[x].right==0)){

if(tree[x].left && tree[x].right){

int p=del(tree[x].left,key+1);

tree[x].key=tree[p].key;

return p;

}

else{

int p=x;

x=tree[x].left+tree[x].right;

return p;

}

}

else

return del(key<tree[x].key?tree[x].left:tree[x].right,key);

}

int deletemin(){

int temp=root;

if(tree[temp].left==0){

root=tree[root].right;

return tree[temp].key;

}

while(tree[tree[temp].left].left){

tree[temp].size--;

temp=tree[temp].left;

}

tree[temp].size--;

int ret=tree[tree[temp].left].key;

tree[temp].left=tree[tree[temp].left].right;

return ret;

}

//得到前驱，返回的是小于v的key值，不存在v

int Pred(int t,int v){

if(!t)

return v;

if(v<=tree[t].key)

return Pred(tree[t].left,v);

else{

int tmp=Pred(tree[t].right,v);

return v==tmp?tree[t].key:tmp;

}

}

//得到后继，返回的是大于v的key值，不存在v

int Succ(int t,int v){

if(!t)

return v;

if(v>=tree[t].key)

return Succ(tree[t].right,v);

else{

int tmp=Succ(tree[t].left,v);

return v==tmp?tree[t].key:tmp;

}

}

void inorder(int &x){

if(x==0)

return;

else{

inorder(tree[x].left);

// cout<<tree[x].key<<" ";

cout<<" "<<tree[x].key<<" "<<tree[x].size<<endl;

inorder(tree[x].right);

}

}

int main(){

char op[20];

while(true){

root=tot=0; int tmp;

cout<<"enter: ";

scanf("%s",op);

if(op[0]=='i'){

scanf("%d",&tmp);

insert(root,tmp);

inorder(root);

cout<<endl;

}

else if(op[0]=='p'){

scanf("%d",&tmp);

printf("%d\n",get\_pre(root,0,tmp));

}

else{

scanf("%d",&tmp);

printf("%d\n",get\_next(root,0,tmp));

}

}

}

## 5.KDT

//2D-求距离最近的点

namespace KDT{

const int maxn = 1e6+7,sigma\_size = 2;

int D,K=2;

struct Point{

int d[sigma\_size];

int &operator[](int k){

return d[k];

}

void read() {

for(int i=0; i<K; ++i)

scanf("%d",d+i);

}

void write() {

for(int i=0; i<K; ++i)

printf("%d%c",d[i]," \n"[i+1==K]);

}

bool operator < (const Point & a) const{

return d[D]<a.d[D];

}

}a[maxn],b[maxn];

struct Node{

Point key,maxd,mind;

Node \*ch[2];

void push\_up(){

for(int i=0; i<K; i++){

maxd[i]=max(maxd[i],ch[0]->maxd[i]);

maxd[i]=max(maxd[i],ch[1]->maxd[i]);

mind[i]=min(mind[i],ch[0]->mind[i]);

mind[i]=min(mind[i],ch[1]->mind[i]);

}

}

}pool[maxn],\*ptr,\*null;

struct qnode{

ll dis; Point p;

bool operator < (const qnode & a) const{

return dis<a.dis;

}

};

priority\_queue<qnode> que;

ll f(int x){

return 1LL\*x\*x;

}

ll dist(Point &a,Point &b){

ll ret=0;

for(int i=0; i<K; i++)

ret+=f(a[i]-b[i]);

return ret;

}

ll h(Node\* o,Point &p){

ll ret=0;

for(int i=0; i<K; i++){

if(o->mind[i]>p[i])

ret+=f(o->mind[i]-p[i]);

else if(p[i]>o->maxd[i])

ret+=f(p[i]-o->maxd[i]);

}

return ret;

}

Node\* newnode(const Point &p){

ptr->key=p;

ptr->ch[0]=ptr->ch[1]=null;

for(int i=0; i<K; i++)

ptr->maxd[i]=ptr->mind[i]=ptr->key[i];

return ptr++;

}

void init(){

ptr=pool;

null=ptr++;

null->ch[0]=null->ch[1]=null;

for(int i=0; i<K; i++){

null->mind[i]=inf;

null->maxd[i]=-inf;

}

}

void build(Node\*& o,int l,int r,int dim){

if(l>r) return;

int mid=l+r>>1;

D=dim;

nth\_element(a+l,a+mid,a+r+1);

o=newnode(a[mid]);

build(o->ch[0],l,mid-1,(dim+1)%K);

build(o->ch[1],mid+1,r,(dim+1)%K);

o->push\_up();

}

void insert(Node\*& o,Point p,int dim){

if(o==null){

o=newnode(p);

return;

}

if(p[dim]<o->key[dim])

insert(o->ch[0],p,(dim+1)%K);

else

insert(o->ch[1],p,(dim+1)%K);

o->push\_up();

}

void query(Node\* o,Point &p,int k){

ll cur=dist(o->key,p),dis[2]={INF,INF};

if(que.size()<k)

que.push({cur,o->key});

else if(cur<que.top().dis){

que.pop(); que.push({cur,o->key});

}

for(int i=0; i<2; i++)

if(o->ch[i]!=null)

dis[i]=h(o->ch[i],p);

bool t=dis[0]>dis[1];

for(int i=0; i<2; i++,t^=1)

if(o->ch[t]!=null&&(que.size()<k)||dis[t]<que.top().dis)

query(o->ch[t],p,k);

}

};

KDT::Node\* root;

KDT::Point\* a = KDT::a;

int n,q,&K=KDT::K;

int main() {

ios\_base::sync\_with\_stdio(0);

int t; scanf("%d", &t);

while(t--) {

scanf("%d", &n);

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

for(int j = 0; j < K; ++j)

scanf("%d", &a[i][j]);

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

KDT::b[i] = a[i];

KDT::init();

KDT::build(root, 1, n, 0);

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

while(KDT::que.size()) KDT::que.pop();

KDT::query(root, KDT::b[i], 2);

printf("%I64d\n", KDT::que.top().dis);

}

}

return 0;

}

//kD-距离最近的k个点

namespace KDT{

const int maxn = 1e6+7,sigma\_size = 5;

int D,K;

struct Point{

int d[sigma\_size];

int &operator[](int k){

return d[k];

}

void read() {

for(int i=0; i<K; ++i)

scanf("%d",d+i);

}

void write() {

for(int i=0; i<K; ++i)

printf("%d%c",d[i]," \n"[i+1==K]);

}

bool operator < (const Point & a) const{

return d[D]<a.d[D];

}

}a[maxn];

struct Node{

Point key,maxd,mind;

Node \*ch[2];

void push\_up(){

for(int i=0; i<K; i++){

maxd[i]=max(maxd[i],ch[0]->maxd[i]);

maxd[i]=max(maxd[i],ch[1]->maxd[i]);

mind[i]=min(mind[i],ch[0]->mind[i]);

mind[i]=min(mind[i],ch[1]->mind[i]);

}

}

}pool[maxn],\*ptr,\*null;

struct qnode{

ll dis; Point p;

bool operator < (const qnode & a) const{

return dis<a.dis;

}

};

priority\_queue<qnode> que;

ll f(int x){

return 1LL\*x\*x;

}

ll dist(Point &a,Point &b){

ll ret=0;

for(int i=0; i<K; i++)

ret+=f(a[i]-b[i]);

return ret;

}

ll h(Node\* o,Point &p){

ll ret=0;

for(int i=0; i<K; i++){

if(o->mind[i]>p[i])

ret+=f(o->mind[i]-p[i]);

else if(p[i]>o->maxd[i])

ret+=f(p[i]-o->maxd[i]);

}

return ret;

}

Node\* newnode(const Point &p){

ptr->key=p;

ptr->ch[0]=ptr->ch[1]=null;

for(int i=0; i<K; i++)

ptr->maxd[i]=ptr->mind[i]=ptr->key[i];

return ptr++;

}

void init(){

ptr=pool;

null=ptr++;

null->ch[0]=null->ch[1]=null;

for(int i=0; i<K; i++){

null->mind[i]=inf;

null->maxd[i]=-inf;

}

}

void build(Node\*& o,int l,int r,int dim){

if(l>r) return;

int mid=l+r>>1;

D=dim;

nth\_element(a+l,a+mid,a+r+1);

o=newnode(a[mid]);

build(o->ch[0],l,mid-1,(dim+1)%K);

build(o->ch[1],mid+1,r,(dim+1)%K);

o->push\_up();

}

void insert(Node\*& o,Point p,int dim){

if(o==null){

o=newnode(p);

return;

}

if(p[dim]<o->key[dim])

insert(o->ch[0],p,(dim+1)%K);

else

insert(o->ch[1],p,(dim+1)%K);

o->push\_up();

}

void query(Node\* o,Point &p,int k){

ll cur=dist(o->key,p),dis[2]={INF,INF};

if(que.size()<k)

que.push({cur,o->key});

else if(cur<que.top().dis){

que.pop(); que.push({cur,o->key});

}

for(int i=0; i<2; i++)

if(o->ch[i]!=null)

dis[i]=h(o->ch[i],p);

bool t=dis[0]>dis[1];

for(int i=0; i<2; i++,t^=1)

if(o->ch[t]!=null&&(que.size()<k)||dis[t]<que.top().dis)

query(o->ch[t],p,k);

}

};

KDT::Node\* root;

KDT::Point\* a = KDT::a;

int n,q,&K=KDT::K;

int main() {

ios\_base::sync\_with\_stdio(0);

while(scanf("%d%d", &n, &K)==2) {

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

for(int j=0; j<K; ++j)

scanf("%d",&a[i][j]);

KDT::init();

KDT::build(root, 1, n, 0);

scanf("%d", &q);

while(q--){

KDT::Point p; p.read();

int k; scanf("%d",&k);

KDT::query(root,p,k);

printf("the closest %d points are:\n", k);

vector<KDT::Point> v; v.reserve(k);

while(KDT::que.size()){

v.push\_back(KDT::que.top().p);

KDT::que.pop();

}

for(int i = k - 1; ~i; --i)

v[i].write();

}

}

return 0;

}

## 6.划分树

const int maxn = 1e5+7;

const int maxm = 1e6+7;

const double pi = acos(-1.0);

int tree[20][maxn];//表示每层每个位置的值

int sorted[maxn];

int toleft[20][maxn];//toleft[p][i]表示第i层从1到i有多少个数分入左边

void build(int l,int r,int dep){

if(l==r)

return ;

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

int same=mid-l+1;//表示等于中间值而且被分入左边的个数

for(int i=l; i<=r; i++)

if(tree[dep][i]<sorted[mid])

same--;

int lpos=l,rpos=mid+1;

for(int i=l; i<=r; i++){

if(tree[dep][i]<sorted[mid])//比中间的数小,分入左边

tree[dep+1][lpos++]=tree[dep][i];

else if(tree[dep][i]==sorted[mid]&&same>0){

tree[dep+1][lpos++]=tree[dep][i];

same--;

}

else//比中间值大分入右边

tree[dep+1][rpos++]=tree[dep][i];

toleft[dep][i]=toleft[dep][l-1]+lpos-l;//从1到i放左边的个数

}

build(l,mid,dep+1);

build(mid+1,r,dep+1);

}

//查询区间第k大的数，[L,R]是大区间，[l,r]是要查询的小区间

int query(int L,int R,int l,int r,int dep,int k){

if(l==r)

return tree[dep][l];

int mid=(L+R)>>1;

int cnt=toleft[dep][r]-toleft[dep][l-1];//[l,r]中位于左边的个数

if(cnt>=k){

//L+要查询的区间前被放在左边的个数

int newl=L+toleft[dep][l-1]-toleft[dep][L-1];

//左端点加上查询区间会被放在左边的个数

int newr=newl+cnt-1;

return query(L,mid,newl,newr,dep+1,k);

}

else{

//r+区间后分入左边的数的个数

int newr=r+toleft[dep][R]-toleft[dep][r];

//右端点减去区间分入右边的数的个数

int newl=newr-(r-l-cnt);

return query(mid+1,R,newl,newr,dep+1,k-cnt);

}

}

int main(){

int n,m,T; cin>>T;

while(T--){

scanf("%d%d",&n,&m);

memset(tree,0,sizeof tree);

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

scanf("%d",&tree[0][i]);

sorted[i]=tree[0][i];

}

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

build(1,n,0);

int s,t,k;

while(m--){

scanf("%d%d%d",&s,&t,&k);

printf("%d\n",query(1,n,s,t,0,k));

}

}

return 0;

}

## 7．回文树

struct PalindromicTree{

//表示编号为i的节点表示的回文串在两边添加字符c以后变成的回文串的编号

int next[maxn][sigma\_size];

//表示节点i失配以后跳转不等于自身的节点i表示的回文串的最长后缀回文串

int fail[maxn];

int cnt[maxn];//表示节点i表示的本质不同的串的个数

//表示以节点i表示的最长回文串的最右端点为回文串结尾的回文串个数

int num[maxn];

int len[maxn];//表示结点i表示的回文串长度

int s[maxn];//存放添加的字符

int last;//指向上一个字符所在的节点

int n,p;//字符数组指针 节点指针

int newnode(int l){

for(int i=0; i<sigma\_size; i++)

next[p][i]=0;

cnt[p]=num[p]=0;

len[p]=l;

return p++;

}

void init(){

p=0;

//0表示偶数长度串的根和1表示奇数长度串的根

newnode(0);

newnode(-1);

last=0;

n=0;

s[n]=-1;////开头放一个字符集中没有的字符，减少特判

fail[0]=1;

}

int Getfail(int x){

while(s[n-len[x]-1]!=s[n])

x=fail[x];

return x;

}

void insert(int c){

c-='a';

s[++n]=c;

int cur=Getfail(last);//通过上一个回文串找这个回文串的匹配位置

if(!next[cur][c]){//如果这个回文串没有出现过，说明出现了一个新的本质不同的回文串

int now=newnode(len[cur]+2);

fail[now]=next[Getfail(fail[cur])][c];

next[cur][c]=now;

num[now]=num[fail[now]]+1;

}

last=next[cur][c];

cnt[last]++;

}

void count(){

//父亲累加儿子的cnt，因为如果fail[v]=u，则u一定是v的子回文串

for(int i=p-1; i>=0; i--)

cnt[fail[i]]+=cnt[i];

}

};

## 8.树分治

//距离不超过k的点对数

struct Edge{

int to,next,w;

}edge[maxn];

int head[maxn],tot;

void addedge(int u,int v,int w){

edge[tot].to=v; edge[tot].w=w; edge[tot].next=head[u];

head[u]=tot++;

}

int n,k,ans,root,num;

//子树的规模 子节点中最大的size

int size[maxn],mx[maxn],mi,dis[maxn];

int vis[maxn];//顶点是否被作为重心删除的标志

//计算子树的大小

void dfssize(int u,int fa){

size[u]=1;

mx[u]=0;

for(int i=head[u]; i!=-1; i=edge[i].next){

int v=edge[i].to;

if(v!=fa&&!vis[v]){

dfssize(v,u);

size[u]+=size[v];

if(size[v]>mx[u])

mx[u]=size[v];

}

}

}

//查找重心

void dfsroot(int r,int u,int fa){

if(size[r]-size[u]>mx[u])

mx[u]=size[r]-size[u];

if(mx[u]<mi)

mi=mx[u],root=u;

for(int i=head[u]; i!=-1; i=edge[i].next){

int v=edge[i].to;

if(v!=fa&&!vis[v])

dfsroot(r,v,u);

}

}

//计算子树中所有顶点到中心的距离

void dfsdis(int u,int d,int fa){

dis[num++]=d;

for(int i=head[u]; i!=-1; i=edge[i].next){

int v=edge[i].to;

if(v!=fa&&!vis[v])

dfsdis(v,d+edge[i].w,u);

}

}

//统计不超过k的顶点对的个数

int calc(int u,int d){

int ret=0;

num=0;

dfsdis(u,d,0);

sort(dis,dis+num);

int i=0,j=num-1;

while(i<j){

while(dis[i]+dis[j]>k&&i<j)

j--;

ret+=j-i;

i++;

}

return ret;

}

//对于顶点u所在的子树，查找重心并分割求解

void dfs(int u){

mi=n;

dfssize(u,0);

dfsroot(u,u,0);

ans+=calc(root,0);

vis[root]=1;

for(int i=head[root]; i!=-1; i=edge[i].next){

int v=edge[i].to;

if(!vis[v]){

ans-=calc(v,edge[i].w);

dfs(v);

}

}

}

void init(){

memset(head,-1,sizeof head);

memset(vis,false,sizeof vis);

tot=ans=0;

}

int main(){

while(scanf("%d%d",&n,&k)!=EOF){

if(!n) break;

init();

int u,v,w;

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

scanf("%d%d%d",&u,&v,&w);

addedge(u,v,w); addedge(v,u,w);

}

dfs(1);

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

}

return 0;

}

## 9．SplayTree

#define Key\_value ch[ch[root][1]][0]

//父结点、左右孩子、点值、子树规模

int pre[maxn],ch[maxn][2],key[maxn],size[maxn];

//子树的和 子树是否被翻转 子树是否被置为一个数

int sum[maxn],rev[maxn],same[maxn];

//数列左起最大连续和 右起最大连续和 这个数列和最大的子序列

int lx[maxn],rx[maxn],mx[maxn];

//根结点、结点数量

int root,tot1;

//内存池、内存池容量

int s[maxn],tot2;

//增量的延迟标记

int add[maxn];

int a[maxn];

int n,q;

//debug部分\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void Treavel(int x){

if(x){

Treavel(ch[x][0]);

printf("结点：%2d: 左儿子 %2d 右儿子 %2d 父结点 %2d size = %2d rev = %2d\n",x,ch[x][0],ch[x][1],pre[x],size[x],rev[x]);

Treavel(ch[x][1]);

}

}

void debug(){

printf("root:%d\n",root);

Treavel(root);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void NewNode(int &r,int father,int k){

if(tot2)

r=s[tot2--];

else

r=++tot1;

pre[r]=father;

ch[r][0]=ch[r][1]=0;

key[r]=k;

sum[r]=k;

rev[r]=same[r]=0;

lx[r]=rx[r]=mx[r]=k;

size[r]=1;

}

void Update\_Same(int r,int v){

if(!r)

return;

key[r]=v;

sum[r]=v\*size[r];

lx[r]=rx[r]=mx[r]=max(v,v\*size[r]);

same[r]=1;

}

//反转的更新

void Update\_Rev(int r){

if(!r)

return;

swap(ch[r][0],ch[r][1]);

swap(lx[r],rx[r]);

rev[r]^=1;//这里要注意，一定是异或1

}

//增加的更新

void Update\_Add(int r,int ADD){

if(!r)

return;

add[r]+=ADD;

key[r]+=ADD;

}

void Push\_Up(int r){

int lson=ch[r][0],rson=ch[r][1];

size[r]=size[lson]+size[rson]+1;

sum[r]=sum[lson]+sum[rson]+key[r];

lx[r]=max(lx[lson],sum[lson]+key[r]+max(0,lx[rson]));

rx[r]=max(rx[rson],sum[rson]+key[r]+max(0,rx[lson]));

mx[r]=max(0,rx[lson])+key[r]+max(0,lx[rson]);

mx[r]=max(mx[r],max(mx[lson],mx[rson]));

}

void Push\_Down(int r){

if(same[r]){

Update\_Same(ch[r][0],key[r]);

Update\_Same(ch[r][1],key[r]);

same[r]=0;

}

if(rev[r]){

Update\_Rev(ch[r][0]);

Update\_Rev(ch[r][1]);

rev[r]=0;

}

if(add[r]){

Update\_Add(ch[r][0],add[r]);

Update\_Add(ch[r][1],add[r]);

add[r]=0;

}

}

void Build(int &x,int l,int r,int father){

if(l>r)

return;

int mid=(l+r)/2;

NewNode(x,father,a[mid]);

Build(ch[x][0],l,mid-1,x);

Build(ch[x][1],mid+1,r,x);

Push\_Up(x);

}

void Init(){

root=tot1=tot2=0;

ch[root][0]=ch[root][1]=pre[root]=size[root]=same[root]=rev[root]=sum[root]=key[root]=0;

lx[root]=rx[root]=mx[root]=-inf;

NewNode(root,0,-1);

NewNode(ch[root][1],root,-1);

for(int i=0; i<n; i++)

scanf("%d",&a[i]);

Build(Key\_value,0,n-1,ch[root][1]);

Push\_Up(ch[root][1]);

Push\_Up(root);

}

void Rotate(int x,int kind){

int y=pre[x];

Push\_Down(y);

Push\_Down(x);

ch[y][!kind]=ch[x][kind];

pre[ch[x][kind]]=y;

if(pre[y])

ch[pre[y]][ch[pre[y]][1]==y]=x;

pre[x]=pre[y];

ch[x][kind]=y;

pre[y]=x;

Push\_Up(y);

}

void Splay(int r,int goal){

Push\_Down(r);

while(pre[r]!=goal){

if(pre[pre[r]]==goal){

Push\_Down(pre[r]);

Push\_Down(r);

Rotate(r,ch[pre[r]][0]==r);

}

else{

Push\_Down(pre[pre[r]]);

Push\_Down(pre[r]);

Push\_Down(r);

int y=pre[r];

int kind=ch[pre[y]][0]==y;

if(ch[y][kind]==r){

Rotate(r,!kind);

Rotate(r,kind);

}

else{

Rotate(y,kind);

Rotate(r,kind);

}

}

}

Push\_Up(r);

if(goal==0)

root=r;

}

int Get\_Min(int r){

Push\_Down(r);

while(ch[r][0]){

r=ch[r][0];

Push\_Down(r);

}

return r;

}

int Get\_Max(int r){

Push\_Down(r);

while(ch[r][1]){

r=ch[r][1];

Push\_Down(r);

}

return r;

}

int Get\_Kth(int r,int k){

Push\_Down(r);

int t=size[ch[r][0]]+1;

if(t==k)

return r;

if(t>k)

return Get\_Kth(ch[r][0],k);

else

return Get\_Kth(ch[r][1],k-t);

}

//删除根节点

void Remove(){

if(ch[root][0]==0){//没有左孩子

root=ch[root][1];

pre[root]=0;

}

else{

int m=Get\_Max(ch[root][0]);

Splay(m,root);

ch[m][1]=ch[root][1];

pre[ch[root][1]]=m;

root=m;

pre[root]=0;

Push\_Up(root);//要更新

}

}

//在第pos个数后插入tot个数

void Insert(int pos,int tot){

for(int i=0; i<tot; i++)

scanf("%d",&a[i]);

Splay(Get\_Kth(root,pos+1),0);

Splay(Get\_Kth(root,pos+2),root);

Build(Key\_value,0,tot-1,ch[root][1]);

Push\_Up(ch[root][1]);

Push\_Up(root);

}

void ADD(int l,int r,int D){

Splay(Get\_Kth(root,l),0);

Splay(Get\_Kth(root,r+2),root);

Update\_Add(Key\_value,D);

Push\_Up(ch[root][1]);

Push\_Up(root);

}

void erase(int r){

if(!r) return;

s[++tot2]=r;

erase(ch[r][0]);

erase(ch[r][1]);

}

//从第pos个数开始连续删除tot个数

void Delete(int pos,int tot){

Splay(Get\_Kth(root,pos),0);

Splay(Get\_Kth(root,pos+tot+1),root);

erase(Key\_value);

pre[Key\_value]=0;

Key\_value=0;

Push\_Up(ch[root][1]);

Push\_Up(root);

}

//从第pos个数连续开始的tot个数修改为c

void Make\_Same(int pos,int tot,int c){

Splay(Get\_Kth(root,pos),0);

Splay(Get\_Kth(root,pos+tot+1),root);

Update\_Same(Key\_value,c);

Push\_Up(ch[root][1]);

Push\_Up(root);

}

//将从第pos个数连续开始的tot 反转

void Reverse(int pos,int tot){

Splay(Get\_Kth(root,pos),0);

Splay(Get\_Kth(root,pos+tot+1),root);

Update\_Rev(Key\_value);

Push\_Up(ch[root][1]);

Push\_Up(root);

}

//求和

int Get\_Sum(int pos,int tot){

Splay(Get\_Kth(root,pos),0);

Splay(Get\_Kth(root,pos+tot+1),root);

return sum[Key\_value];

}

//得到最大和

int Get\_MaxSum(int pos,int tot){

Splay(Get\_Kth(root,pos),0);

Splay(Get\_Kth(root,pos+tot+1),root);

return mx[Key\_value];

}

//查找SplayTree中值为val的下标

int find(int r,int val){

if(!r)

return 0;

if(val==key[r])

return r;

if(val<key[r])

return find(val,ch[r][0]);

return find(val,ch[r][1]);

}

//单点的操作

void insert(int pos,int val){

int x=Get\_Kth(root,pos);

if(ch[x][1]==0)

NewNode(ch[x][1],x,val);

else{

x=Get\_Min(ch[x][1]);

NewNode(ch[x][0],x,val);

}

while(x){

Push\_Up(x);

x=pre[x];

}

n++;//!!!!!!!!

}

int Del(int pos){

int x=Get\_Kth(root,pos-1);

int y=Get\_Kth(root,pos+1);

Splay(x,0);

Splay(y,root);

x=ch[ch[root][1]][0];

pre[x]=0; ch[ch[root][1]][0]=0;

Push\_Up(ch[root][1]);

Splay(1,0);

n--;

return key[x];

}

//删除最后一个点（第n个），加入到第一个点前。

void move1(){

int x=Del(n+1);

Insert(0+1,x);

}

void move2(){

int x=Del(1+1);

Insert(n+1,x);

}

void Inorder(int r){

if(!r) return;

Push\_Down(r);

Inorder(ch[r][0]);

printf("%d ",key[r]);

Inorder(ch[r][1]);

};

## 10.分块例题

/\*

某天，Lostmonkey发明了一种超级弹力装置，为了在他的绵羊朋友面前显摆，

他邀请小绵羊一起玩个游戏。

游戏一开始，Lostmonkey在地上沿着一条直线摆上n个装置，

每个装置设定初始弹力系数ki，当绵羊达到第i个装置时，

它会往后弹ki步，达到第i+ki个装置，若不存在第i+ki个装置，

i=1，你要输出从j出发被弹几次后被弹飞，

若i=2则还会再输入一个正整数k，表示第j个弹力装置的系数被修改成k。

\*/

int n,q;

//跳出该块需要的部数 跳向下一个块的哪个点

int a[maxn],cnt[maxn],to[maxn];

//点属于第几个块/块长/块数 块左端 块右端

int belong[maxn],block,num,l[maxn],r[maxn];

//下标从1开始

void Build(){

block=(int)sqrt(n+0.5);

num=n/block;

if(n%block)

num++;

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

l[i]=(i-1)\*block+1;

r[i]=i\*block;

}

r[num]=n;//最右端端点更新

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

belong[i]=((i-1)/block)+1;

for(int i=num; i>0; i--){

for(int j=r[i]; j>=l[i]; j--){

if(j+a[j]>r[i]){

cnt[j]=1; to[j]=min(n+1,j+a[j]);

}

else{

cnt[j]=cnt[j+a[j]]+1;

to[j]=min(n+1,to[j+a[j]]);

}

}

}

}

void modify(int pos,int val){

a[pos]=val;

for(int i=pos; i>=l[belong[pos]]; i--){

if(i+a[i]>r[belong[pos]]){

cnt[i]=1;

to[i]=min(n+1,i+a[i]);

}

else{

cnt[i]=cnt[i+a[i]]+1;

to[i]=min(n+1,to[i+a[i]]);

}

}

}

int query(int pos){

int ans=0;

for(int i=pos; i<=n; i=to[i])

ans+=cnt[i];

return ans;

}

int main(){

while(scanf("%d",&n)!=EOF){

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

scanf("%d",&a[i]);

Build();

scanf("%d",&q);

int op,pos,v;

for(int i=0; i<q; i++){

scanf("%d",&op);

if(op==1){

scanf("%d",&pos); pos++;

printf("%d\n",query(pos));

}

else{

scanf("%d%d",&pos,&v); pos++;

modify(pos,v);

}

}

}

return 0;

}

## 11.树链剖分

struct Edge{

int to,next;

}edge[maxn\*2];

int head[maxn],tot;

int size[maxn];//保存以x为根的子树节点个数

int dep[maxn]; //保存当前节点的深度

int fa[maxn]; //保存当前节点的父亲

int son[maxn]; //保存重儿子

int top[maxn]; //保存当前节点的所在链的顶端节点

int p[maxn]; //保存树中每个节点剖分后的新编号

int fp[maxn]; //fp和p数组相反

int pos;

void init(){

tot=0; pos=1;

memset(head,-1,sizeof head);

memset(son,-1,sizeof son);

}

void addedge(int u,int v){

edge[tot].to=v; edge[tot].next=head[u]; head[u]=tot++;

}

//记录所有的重边

void dfs1(int u,int father,int d){

dep[u]=d;

fa[u]=father;

size[u]=1;

for(int i=head[u]; i!=-1; i=edge[i].next){

int v=edge[i].to;

if(v!=father){

dfs1(v,u,d+1);

size[u]+=size[v];

if(son[u]==-1||size[v]>size[son[u]])

son[u]=v;

}

}

}

//连重边成重链

void dfs2(int u,int tp){

top[u]=tp;

p[u]=pos++;

fp[p[u]]=u;

if(son[u]==-1)

return;

dfs2(son[u],tp);

for(int i=head[u]; i!=-1; i=edge[i].next){

int v=edge[i].to;

if(v!=son[u]&&v!=fa[u])

dfs2(v,v);

}

}

int c[maxn];

int n;

int lowbit(int x){

return x&(-x);

}

int Sum(int i){

int res=0;

while(i>0){

res+=c[i];

i-=lowbit(i);

}

return res;

}

void add(int i,int val){

while(i<=n){

c[i]+=val;

i+=lowbit(i);

}

}

void change(int u,int v,int val){

int f1=top[u],f2=top[v];

int tmp=0;

while(f1!=f2){

if(dep[f1]<dep[f2]){

swap(f1,f2);

swap(u,v);

}

add(p[f1],val);

add(p[u]+1,-val);

u=fa[f1]; f1=top[u];

}

if(dep[u]>dep[v])

swap(u,v);

add(p[u],val);

add(p[v]+1,-val);

}

int a[maxn];

int main(){

int u,v,m,pp;

while(scanf("%d%d%d",&n,&m,&pp)!=EOF){

init();

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

scanf("%d",&a[i]);

while(m--){

scanf("%d%d",&u,&v);

addedge(u,v); addedge(v,u);

}

dfs1(1,0,0);

dfs2(1,1);

memset(c,0,sizeof c);

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

add(p[i],a[i]);

add(p[i]+1,-a[i]);

}

char op[20];

while(pp--){

scanf("%s",op);

if(op[0]=='Q'){

scanf("%d",&u);

printf("%d\n",Sum(p[u]));

}

else{

int tmp1,tmp2,tmp3;

scanf("%d%d%d",&tmp1,&tmp2,&tmp3);

if(op[0]=='D')

tmp3=-tmp3;

change(tmp1,tmp2,tmp3);

}

}

}

return 0;

}

//查询最大值 取反 改单边值

struct Edge{

int to,next;

}edge[maxn\*2];

int head[maxn],tot;

int size[maxn];//保存以x为根的子树节点个数

int dep[maxn]; //保存当前节点的深度

int fa[maxn]; //保存当前节点的父亲

int son[maxn]; //保存重儿子

int top[maxn]; //保存当前节点的所在链的顶端节点

int p[maxn]; //保存树中每个节点剖分后的新编号

int fp[maxn]; //fp和p数组相反

int pos;

void init(){

tot=0; pos=1;

memset(head,-1,sizeof head);

memset(son,-1,sizeof son);

}

void addedge(int u,int v){

edge[tot].to=v; edge[tot].next=head[u]; head[u]=tot++;

}

//记录所有的重边

void dfs1(int u,int father,int d){

dep[u]=d;

fa[u]=father;

size[u]=1;

for(int i=head[u]; i!=-1; i=edge[i].next){

int v=edge[i].to;

if(v!=father){

dfs1(v,u,d+1);

size[u]+=size[v];

if(son[u]==-1||size[v]>size[son[u]])

son[u]=v;

}

}

}

//连重边成重链

void dfs2(int u,int tp){

top[u]=tp;

p[u]=pos++;

fp[p[u]]=u;

if(son[u]==-1)

return;

dfs2(son[u],tp);

for(int i=head[u]; i!=-1; i=edge[i].next){

int v=edge[i].to;

if(v!=son[u]&&v!=fa[u])

dfs2(v,v);

}

}

struct Node{

int l,r;

int mx;

int mn;

int lazy;

};

int n;

struct SegTree{

Node st[maxn\*3];

void build(int i,int l,int r){

st[i].l=l;

st[i].r=r;

st[i].mx=st[i].mn=st[i].lazy=0;

if(l==r){

return ;

}

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

build(i<<1,l,mid);

build((i<<1)|1,mid+1,r);

push\_up(i);

}

//回溯更新父节点

void push\_up(int i){

st[i].mx=max(st[i<<1].mx,st[(i<<1)|1].mx);

st[i].mn=min(st[i<<1].mn,st[(i<<1)|1].mn);

}

void push\_down(int i){

if(st[i].l==st[i].r)

return;

if(st[i].lazy){

st[i<<1].mx=-st[i<<1].mx;

st[i<<1].mn=-st[i<<1].mn;

swap(st[i<<1].mx,st[i<<1].mn);

st[(i<<1)|1].mx=-st[(i<<1)|1].mx;

st[(i<<1)|1].mn=-st[(i<<1)|1].mn;

swap(st[(i<<1)|1].mx,st[(i<<1)|1].mn);

st[i<<1].lazy^=1; st[(i<<1)|1].lazy^=1;

st[i].lazy=0;

}

}

void update(int i,int k,int val){

if(st[i].l==k&&st[i].r==k){

st[i].mx=st[i].mn=val;

st[i].lazy=false;

return ;

}

push\_down(i);

int mid=(st[i].l+st[i].r)>>1;

if(k<=mid)

update(i<<1,k,val);

else

update((i<<1)|1,k,val);

push\_up(i);

}

void update2(int i,int l,int r){

if(st[i].l==l&&st[i].r==r){

st[i].mx=-st[i].mx;

st[i].mn=-st[i].mn;

swap(st[i].mx,st[i].mn);

st[i].lazy^=1;

return;

}

push\_down(i);

int mid=(st[i].l+st[i].r)>>1;

if(r<=mid)

update2((i<<1),l,r);

else if(l>mid)

update2((i<<1)|1,l,r);

else{

update2((i<<1),l,mid);

update2((i<<1)|1,mid+1,r);

}

push\_up(i);

}

void ng\_update(int u,int v){

int f1=top[u],f2=top[v];

int tmp=0;

while(f1!=f2){

if(dep[f1]<dep[f2]){

swap(f1,f2);

swap(u,v);

}

update2(1,p[f1],p[u]);

u=fa[f1]; f1=top[u];

}

if(u==v)

return;

if(dep[u]>dep[v])

swap(u,v);

return update2(1,p[son[u]],p[v]);

}

int query(int i,int l,int r){

if(st[i].l==l&&st[i].r==r)

return st[i].mx;

push\_down(i);

int mid=(st[i].l+st[i].r)>>1;

if(r<=mid)

return query(i<<1,l,r);

else if(l>mid)

return query((i<<1)|1,l,r);

else

return max(query(i<<1,l,mid),query((i<<1)|1,mid+1,r));

push\_up(i);

}

int find(int u,int v){

int f1=top[u],f2=top[v];

int tmp=-inf;//!!!

while(f1!=f2){

if(dep[f1]<dep[f2]){

swap(f1,f2);

swap(u,v);

}

tmp=max(tmp,query(1,p[f1],p[u]));

u=fa[f1]; f1=top[u];

}

if(u==v)

return tmp;

if(dep[u]>dep[v])

swap(u,v);

return max(tmp,query(1,p[son[u]],p[v]));

}

}st;

int e[maxn][3];

int main(){

int T; scanf("%d",&T);

while(T--){

init();

scanf("%d",&n);

for(int i=0; i<n-1; i++){

scanf("%d%d%d",&e[i][0],&e[i][1],&e[i][2]);

addedge(e[i][0],e[i][1]);

addedge(e[i][1],e[i][0]);

}

dfs1(1,0,0); dfs2(1,1);

st.build(1,0,pos-1);

for(int i=0; i<n-1; i++){

if(dep[e[i][0]]>dep[e[i][1]])

swap(e[i][0],e[i][1]);

st.update(1,p[e[i][1]],e[i][2]);

}

char op[20];

int u,v;

while(scanf("%s",op)==1){

if(op[0]=='D')

break;

scanf("%d%d",&u,&v);

if(op[0]=='Q')

printf("%d\n",st.find(u,v));

else if(op[0]=='C')

st.update(1,p[e[u-1][1]],v);

else

st.ng\_update(u,v);

}

}

return 0;

}

//HDU6074

int n,m,w;

struct Edge{

int to,next;

}edge[maxn\*2];

int head[maxn],tot;

int size[maxn];//保存以x为根的子树节点个数

int dep[maxn]; //保存当前节点的深度

int f[maxn]; //保存当前节点的父亲

int son[maxn]; //保存重儿子

int top[maxn]; //保存当前节点的所在链的顶端节点

int same[maxn],fa[maxn],cnt[maxn];

ll cost[maxn];

void init(){

tot=0;

memset(head,-1,sizeof head);

memset(son,-1,sizeof son);

}

void addedge(int u,int v){

edge[tot].to=v; edge[tot].next=head[u]; head[u]=tot++;

}

//记录所有的重边

void dfs1(int u){

size[u]=1;

for(int i=head[u]; i!=-1; i=edge[i].next){

int v=edge[i].to;

if(v!=f[u]){

f[v]=u; dep[v]=dep[u]+1;

dfs1(v);

size[u]+=size[v];

if(son[u]==-1||size[v]>size[son[u]])

son[u]=v;

}

}

}

//连重边成重链

void dfs2(int u,int tp){

top[u]=tp;

if(son[u]==-1)

return;

dfs2(son[u],tp);

for(int i=head[u]; i!=-1; i=edge[i].next){

int v=edge[i].to;

if(v!=son[u]&&v!=f[u])

dfs2(v,v);

}

}

int lca(int x,int y){

for(; top[x]!=top[y]; x=f[top[x]])

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

swap(x,y);

return dep[x]<dep[y]?x:y;

}

int Same(int x){

if(same[x]==x)

return x;

return same[x]=Same(same[x]);

}

int find(int x){

if(fa[x]==x)

return x;

return fa[x]=find(fa[x]);

}

void merge(int x,int y){

x = find(x); y = find(y);

if(x==y)

return;

fa[x]=y;

cnt[y]+=cnt[x];

cost[y]+=cost[x]+w;

}

void go(int x,int y){

while(true){

x=Same(x);

if(dep[x]<=dep[y])

return;

merge(x,f[x]);

same[x]=f[x];

}

}

void chain(int x,int y){

int l = lca(x,y);

go(x,l); go(y,l);

}

struct E{

int a,b,c,d,w;

}e[maxn];

bool cmp(const E &a,const E &b){

return a.w<b.w;

}

int main(){

int T; scanf("%d",&T);

while(T--){

init();

scanf("%d%d",&n,&m);

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

int u,v; scanf("%d%d",&u,&v);

addedge(u,v); addedge(v,u);

}

dfs1(1);dfs2(1,1);

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

same[i]=fa[i]=i;

cnt[i]=1; cost[i]=0;

}

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

scanf("%d%d%d%d%d",&e[i].a,&e[i].b,&e[i].c,&e[i].d,&e[i].w);

sort(e+1,e+m+1,cmp);

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

w=e[i].w;

chain(e[i].a,e[i].b);

chain(e[i].c,e[i].d);

merge(e[i].a,e[i].c);

}

printf("%d %lld\n",cnt[find(1)],cost[find(1)]);

}

return 0;

}

## 12．LCT

int n,q;

int ch[maxn][2],pre[maxn],key[maxn];

int add[maxn],rev[maxn],Max[maxn];

bool rt[maxn];

void Update\_Add(int r,int d){

if(!r)

return ;

key[r]+=d;

add[r]+=d;

Max[r]+=d;

}

void Update\_Rev(int r){

if(!r)

return;

swap(ch[r][0],ch[r][1]);

rev[r]^=1;

}

void push\_down(int r){

if(add[r]){

Update\_Add(ch[r][0],add[r]);

Update\_Add(ch[r][1],add[r]);

add[r]=0;

}

if(rev[r]){

Update\_Rev(ch[r][0]);

Update\_Rev(ch[r][1]);

rev[r]=0;

}

}

void push\_up(int r){

Max[r]=max(max(Max[ch[r][0]],Max[ch[r][1]]),key[r]);

}

void Rotate(int x){

int y=pre[x],kind=ch[y][1]==x;

ch[y][kind]=ch[x][!kind];

pre[ch[y][kind]]=y;

pre[x]=pre[y];

pre[y]=x;

ch[x][!kind]=y;

if(rt[y])

rt[y]=false,rt[x]=true;

else

ch[pre[x]][ch[pre[x]][1]==y]=x;

push\_up(y);

}

//P函数先将根结点到r的路径上所有的结点的标记逐级下放

void P(int r){

if(!rt[r])

P(pre[r]);

push\_down(r);

}

void Splay(int r){

P(r);

while(!rt[r]){

int f=pre[r],ff=pre[f];

if(rt[f])

Rotate(r);

else if( (ch[ff][1]==f)==(ch[f][1]==r) )

Rotate(f),Rotate(r);

else

Rotate(r),Rotate(r);

}

push\_up(r);

}

int Access(int x){

int y=0;

for(; x; x=pre[y=x]){

Splay(x);

rt[ch[x][1]]=true,rt[ch[x][1]=y]=false;

push\_up(x);

}

return y;

}

//判断是否是同根(真实的树，非splay)

bool judge(int u,int v){

while(pre[u])

u=pre[u];

while(pre[v])

v=pre[v];

return u==v;

}

//使r成为它所在的树的根

void mroot(int r){

Access(r);

Splay(r);

Update\_Rev(r);

}

//调用后u是原来u和v的lca,v和ch[u][1]分别存着lca的2个儿子

//(原来u和v所在的2颗子树)

void lca(int &u,int &v){

Access(v),v=0;

while(u){

Splay(u);

if(!pre[u])

return;

rt[ch[u][1]]=true;

rt[ch[u][1]=v]=false;

push\_up(u);

u=pre[v=u];

}

}

//link(a,b) : 如果a,b不在同一颗子树中

//则通过在a,b之间连边的方式，连接这两颗子树

void link(int u,int v){

if(judge(u,v)){

puts("-1");

return;

}

mroot(u);

pre[u]=v;

}

//如果a,b在同一颗子树中，且a!=b

//则将a视为这颗子树的根以后，切断b与其父亲结点的连接

void cut(int u,int v){

if(u==v||!judge(u,v)){

puts("-1");

return;

}

mroot(u);

Splay(v);

pre[ch[v][0]]=pre[v];

pre[v]=0;

rt[ch[v][0]]=true;

ch[v][0]=0;

push\_up(v);

}

//ADD(a,b,w): 如果a,b在同一颗子树中，则将a,b之间路径上所有点的点权增加w

void ADD(int u,int v,int w){

if(!judge(u,v)){

puts("-1");

return;

}

lca(u,v);

Update\_Add(ch[u][1],w);

Update\_Add(v,w);

key[u]+=w;

push\_up(u);

}

//修改点权

void change(int u,int k){

Access(u);

key[u]=k;

push\_up(u);

}

//query(a,b): 如果a,b在同一颗子树中，返回a,b之间路径上点权的最大值

void query(int u,int v){

if(!judge(u,v)){

puts("-1");

return;

}

lca(u,v);

printf("%d\n",max(max(Max[v],Max[ch[u][1]]),key[u]));

//边权：printf("%d\n",max(Max[v],Max[ch[u][1]]));

}

//query(a,b): 如果a,b在同一颗子树中，返回a,b之间路径上点权的最大值

void query(int u,int v){

if(!judge(u,v)){

puts("-1");

return;

}

lca(u,v);

printf("%d\n",max(max(Max[v],Max[ch[u][1]]),key[u]));

//点权：printf("%d\n",max(max(Max[v],Max[ch[u][1]]),key[u]));

//边权：printf("%d\n",max(Max[v],Max[ch[u][1]]));

//点权和： printf("%d\n",sum[v]+sum[ch[u][1]]+key[u]);

}

struct Edge{

int to,next;

}edge[maxn\*2];

int head[maxn],tot;

void addedge(int u,int v){

edge[tot].to=v; edge[tot].next=head[u];

head[u]=tot++;

}

void dfs(int u){

for(int i=head[u]; i!=-1; i=edge[i].next){

int v=edge[i].to;

if(pre[v]!=0)

continue;

pre[v]=u;

dfs(v);

}

}

void init(){

tot=0;

memset(head,-1,sizeof head);

for(int i=0; i<=n; i++){

pre[i]=0;

ch[i][0]=ch[i][1]=0;

rev[i]=0;

add[i]=0;

rt[i]=true;

}

}

int main(){

while(scanf("%d",&n)!=EOF){

init();

Max[0]=-inf;

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

int u,v; scanf("%d%d",&u,&v);

addedge(u,v);

addedge(v,u);

}

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

scanf("%d",&key[i]);

Max[i]=key[i];

}

scanf("%d",&q);

pre[1]=-1;

dfs(1);

pre[1]=0;

int op;

while(q--){

scanf("%d",&op);

if(op==1){

int x,y; scanf("%d%d",&x,&y);

link(x,y);

}

else if(op==2){

int x,y; scanf("%d%d",&x,&y);

cut(x,y);

}

else if(op==3){

int w,x,y; scanf("%d%d%d",&w,&x,&y);

ADD(x,y,w);

}

else{

int x,y; scanf("%d%d",&x,&y);

query(x,y);

}

}

printf("\n");

}

return 0;

}