树套树

树状数组套树状数组

简介

一般用与维护二维矩阵，码量以及常数小，通常涉及差分。

模板题

洛谷p4514

维护一个矩阵，支持以下操作：

1. 将以 $(r_1,c_1),(r_2,c_2)$ 为顶点的矩阵内全部元素加上 k
2. 输出 $(r_1,c_1),(r_2,c_2)$ 为顶点的矩阵内全部元素的和

考虑二维差分，设矩阵元素 $a_{x,y}=\sum_{i=1}^x\sum_{j=1}^y b_{i,j}$则

\begin{equation}S_{i,j}=\sum_{x=1}^r\sum_{y=1}^c
a_{x,y}=\sum_{x=1}^r\sum_{y=1}^c\sum_{i=1}^x\sum_{j=1}^y
b_{i,j}=\sum_{i=1}^r\sum_{j=1}^c (r-i+1)(c-j+1)b_{i,j}\end{equation}

展开得

\begin{equation}S_{i,j}=(r+1)(c+1)\sum_{i=1}^r\sum_{j=1}^c b_{i,j}-
(c+1)\sum_{i=1}^r\sum_{j=1}^c ib_{i,j}-(r+1)\sum_{i=1}^r\sum_{j=1}^c
jb_{i,j}+\sum_{i=1}^r\sum_{j=1}^c ijb_{i,j}\end{equation}

考虑分别维护以下四项

\begin{equation}\sum_{i=1}^r\sum_{j=1}^c b_{i,j},\sum_{i=1}^r\sum_{j=1}^c
ib_{i,j},\sum_{i=1}^r\sum_{j=1}^c jb_{i,j},\sum_{i=1}^r\sum_{j=1}^c ijb_{i,j}\end{equation}

修改操作变为 $b_{r_1,c_1}+=v,b_{r_2+1,c_1}-=v,b_{r_1,c_2+1}-=v,b_{r_2+1,c_2+1}+=v$

查询操作变为 $S=S_{r_2,c_2}-S_{r_1-1,c_2}-S_{r_2,c_1-1}+S_{r_1-1,c_1-1}$

空间复杂度 $O(n^2)$时间复杂度 $O(q\log^2 n)$

```
#define lowbit(x) (x)&(-x)
const int MAXN=2050;
struct BIT{
    int a[4][MAXN][MAXN],n,m;
    void modify(int r,int c,int v){
        for(int i=r;i<=n;i+=lowbit(i)){
            for(int j=c;j<=m;j+=lowbit(j)){
                a[0][i][j]+=v;
```

```
                a[1][i][j]+=v*r;
                a[2][i][j]+=v*c;
                a[3][i][j]+=v*r*c;
            }
        }
    }
    void add(int r1,int c1,int r2,int c2,int v){
        modify(r1,c1,v);modify(r2+1,c1,-v);
        modify(r1,c2+1,-v);modify(r2+1,c2+1,v);
    }
    int pre_sum(int r,int c){
        int ans[4]={0};
        for(int i=r;i;i-=lowbit(i)){
            for(int j=c;j;j-=lowbit(j)){
                _for(k,0,4)
                ans[k]+=a[k][i][j];
            }
        }
        return ans[0]*(r+1)*(c+1)-ans[1]*(c+1)-ans[2]*(r+1)+ans[3];
    }
    int query(int r1,int c1,int r2,int c2){
        return pre_sum(r2,c2)-pre_sum(r1-1,c2)-
pre_sum(r2,c1-1)+pre_sum(r1-1,c1-1);
    }
}tree;
char order[1000];
int main()
{
    scanf("X %d %d\n",&tree.n,&tree.m);
    int a,b,c,d,v;
    while(~scanf("%s",order)){
        if(order[0]=='L'){
a=read_int(),b=read_int(),c=read_int(),d=read_int(),v=read_int();
            tree.add(a,b,c,d,v);
        }
        else{
            a=read_int(),b=read_int(),c=read_int(),d=read_int();
            enter(tree.query(a,b,c,d));
        }
    }
    return 0;
}
```

线段树套线段树

简介

一般用于解决树状数组套树状数组无法解决的二维偏序问题，通常涉及权值线段树、动态开点等。

模板题

洛谷p3332

维护 n 个可重集，支持以下操作：

1. 将 c 加入编号为 $[l\sim r]$ 的集合
2. 查询编号为 $[l\sim r]$ 的集合的并集中第 c 大的元素

考虑权值线段树套线段树，第一维维护每个数值，第二维维护每个集合在某个数值区间拥有的元素个数。

为防止爆内存，第二维线段树需要动态开点，同时考虑线段树标记永久化优化常数。

时空间复杂度均为 $O(q\log v\log n)$

```
const int MAXN=(5e4+5)*4,MAXM=40;
struct Node{
    int ch[2],tag;
    LL sz;
}node[MAXN*MAXM];
int lef[MAXN],rig[MAXN],root[MAXN],n,tot;
void build_2D(int k,int L,int R){
    lef[k]=L,rig[k]=R;
    if(L==R)
    return;
    int M=L+R>>1;
    build_2D(k<<1,L,M);
    build_2D(k<<1|1,M+1,R);
}
void update_1D(int &k,int lef,int rig,int L,int R){
    if(!k)k=++tot;
    if(L<=lef&&rig<=R)
    return node[k].sz+=rig-lef+1,node[k].tag++,void();
    int mid=lef+rig>>1;
    if(mid>=L)
    update_1D(node[k].ch[0],lef,mid,L,R);
    if(mid<R)
    update_1D(node[k].ch[1],mid+1,rig,L,R);
    node[k].sz=node[node[k].ch[0]].sz+node[node[k].ch[1]].sz+1LL*(rig-
lef+1)*node[k].tag;
}
void update_2D(int L,int R,int v){
    int k=1,mid;
    while(lef[k]<rig[k]){
        update_1D(root[k],1,n,L,R);
        mid=lef[k]+rig[k]>>1;
        k<<=1;
        k|=(mid<v);
```

```
    }
    update_1D(root[k],1,n,L,R);
}
LL query_1D(int k,int lef,int rig,int L,int R,int tag){
    if(!k)
    return 1LL*(min(rig,R)-max(lef,L)+1)*tag;
    if(L<=lef&&rig<=R)
    return node[k].sz+1LL*(rig-lef+1)*tag;
    int mid=lef+rig>>1;
    if(mid>=R)
    return query_1D(node[k].ch[0],lef,mid,L,R,tag+node[k].tag);
    else if(mid<L)
    return query_1D(node[k].ch[1],mid+1,rig,L,R,tag+node[k].tag);
    else
    return
query_1D(node[k].ch[0],lef,mid,L,R,tag+node[k].tag)+query_1D(node[k].ch[1],
mid+1,rig,L,R,tag+node[k].tag);
}
int query_2D(int L,int R,LL rk){
    int k=1;LL trk;
    rk--;
    while(lef[k]<rig[k]){
        trk=query_1D(root[k<<1|1],1,n,L,R,0);
        k<<=1;
        if(trk<=rk)
        rk-=trk;
        else
        k|=1;
    }
    return lef[k];
}
int main()
{
    n=read_int();
    build_2D(1,1,n);
    int q=read_int(),opt,l,r;
    LL c;
    while(q--){
        opt=read_int(),l=read_int(),r=read_int(),c=read_LL();
        if(opt&1)
        update_2D(l,r,c);
        else
        enter(query_2D(l,r,c));
    }
    return 0;
}
```

线段树/树状数组套名次树

简介

用于实现区间范围的名次树操作，空间复杂度 $O(n\log n)$

模板题

洛谷p3380

维护一种数集结构，支持以下操作：

1. 查询 k 在区间内的排名
2. 查询区间内排名为 k 的值
3. 修改某一位置上的数值
4. 查询 k 在区间内的前驱
5. 查询 k 在区间内的后继

线段树套名次树版本

线段树套名次树 rank update pre suf 操作和普通线段树类似 kth 操作需要二分 $+$ rank 操作。

rank update pre suf 操作时间复杂度为 $O(\log^2 n)$ kth 操作时间复杂度 $O(\log^2 n\log v)$

```
const int MAXN=(5e4+5)*4,MAXS=MAXN*20,Inf=0x7fffffff;
template <typename T>
struct Treap{
    int pool[MAXS],top,tot;
    int root[MAXN],ch[MAXS][2],r[MAXS],sz[MAXS],cnt[MAXS];
    T val[MAXS];
    int new_node(T v){
        int id=top?pool[top--]:++tot;
        val[id]=v;r[id]=rand();sz[id]=cnt[id]=1;ch[id][0]=ch[id][1]=0;
        return id;
    }
    void push_up(int id){sz[id]=sz[ch[id][0]]+sz[ch[id][1]]+cnt[id];}
    void Rotate(int &id,int dir){
        int t=ch[id][dir^1];
        ch[id][dir^1]=ch[t][dir];ch[t][dir]=id;id=t;
        push_up(ch[id][dir]);push_up(id);
    }
    void Insert(int &id,T v){
        if(!id)
        return id=new_node(v),void();
        if(v==val[id])
```

```
        cnt[id]++;
        else{
            int dir=v<val[id]?0:1;
            Insert(ch[id][dir],v);
            if(r[id]<r[ch[id][dir]])
            Rotate(id,dir^1);
        }
        push_up(id);
    }
    void Erase(int &id,T v){
        if(!id)
        return;
        if(v==val[id]){
            if(cnt[id]>1)return    cnt[id]--,push_up(id);
            else if(!ch[id][0])pool[++top]=id,id=ch[id][1];
            else if(!ch[id][1])pool[++top]=id,id=ch[id][0];
            else{
                int d=r[ch[id][0]]>r[ch[id][1]]?1:0;
                Rotate(id,d);Erase(ch[id][d],v);push_up(id);
            }
        }
        else{
            if(v<val[id])Erase(ch[id][0],v);
            else Erase(ch[id][1],v);
            push_up(id);
        }
    }
    int Rank(int id,T v){//有多少个数严格小于v
        if(!id)return 0;
        if(v==val[id])return sz[ch[id][0]];
        else if(v<val[id])return Rank(ch[id][0],v);
        else return sz[ch[id][0]]+cnt[id]+Rank(ch[id][1],v);
    }
    T Kth(int id,int rk){
        if(!id) return -1;//第rk小的节点不存在
        if(rk>sz[ch[id][0]]+cnt[id]) return Kth(ch[id][1],rk-sz[ch[id][0]]-
cnt[id]);
        else if(rk>sz[ch[id][0]]) return val[id];
        else return Kth(ch[id][0],rk);
    }
    T Pre(int id,T v){
        int pos=id,ans=-Inf;
        while(pos){
            if(val[pos]<v){
                ans=ans<val[pos]?val[pos]:ans;
                pos=ch[pos][1];
            }
            else pos=ch[pos][0];
        }
        return ans;
```

```
    }
    T Suf(int id,T v){
        int pos=id,ans=Inf;
        while(pos){
            if(v<val[pos]){
                ans=ans<val[pos]?ans:val[pos];
                pos=ch[pos][0];
            }
            else pos=ch[pos][1];
        }
        return ans;
    }
    void insert(int root_id,T v){Insert(root[root_id],v);}
    void erase(int root_id,T v){Erase(root[root_id],v);}
    int rank(int root_id,T v){return Rank(root[root_id],v);}//如果需要，记得+1
    T kth(int root_id,int rk){return Kth(root[root_id],rk);}
    T pre(int root_id,T v){return Pre(root[root_id],v);}
    T suf(int root_id,T v){return Suf(root[root_id],v);}
};
struct Tree{
    Treap<int> S;
    int a[MAXN],lef[MAXN],rig[MAXN];
    void build(int k,int L,int R){
        lef[k]=L,rig[k]=R;
        _rep(i,L,R)
        S.insert(k,a[i]);
        if(L==R)
        return;
        int M=L+R>>1;
        build(k<<1,L,M);
        build(k<<1|1,M+1,R);
    }
    void build(int n){build(1,1,n);}
    int Rank(int k,int L,int R,int v){
        if(L<=lef[k]&&rig[k]<=R)
        return S.rank(k,v);
        int mid=lef[k]+rig[k]>>1;
        if(mid>=R)
        return Rank(k<<1,L,R,v);
        else if(mid<L)
        return Rank(k<<1|1,L,R,v);
        else
        return Rank(k<<1,L,R,v)+Rank(k<<1|1,L,R,v);
    }
    int rank(int L,int R,int v){return Rank(1,L,R,v)+1;}
    int kth(int L,int R,int rk){
        int lef=0,rig=1e8,mid,trk,ans=0;
        while(lef<=rig){
            mid=lef+rig>>1;
            trk=rank(L,R,mid);
            if(trk<=rk){
```

```
                ans=mid;
                lef=mid+1;
            }
            else if(trk>rk)
            rig=mid-1;
        }
        return ans;
    }
    void update(int k,int pos,int v){
        S.erase(k,a[pos]);
        S.insert(k,v);
        if(lef[k]==rig[k]){
            a[pos]=v;
            return;
        }
        int mid=lef[k]+rig[k]>>1;
        if(pos<=mid)
        update(k<<1,pos,v);
        else
        update(k<<1|1,pos,v);
    }
    void update(int pos,int v){update(1,pos,v);}
    int Pre(int k,int L,int R,int v){
        if(L<=lef[k]&&rig[k]<=R)
        return S.pre(k,v);
        int mid=lef[k]+rig[k]>>1;
        if(mid>=R)
        return Pre(k<<1,L,R,v);
        else if(mid<L)
        return Pre(k<<1|1,L,R,v);
        else
        return max(Pre(k<<1,L,R,v),Pre(k<<1|1,L,R,v));
    }
    int pre(int L,int R,int v){return Pre(1,L,R,v);}
    int Suf(int k,int L,int R,int v){
        if(L<=lef[k]&&rig[k]<=R)
        return S.suf(k,v);
        int mid=lef[k]+rig[k]>>1;
        if(mid>=R)
        return Suf(k<<1,L,R,v);
        else if(mid<L)
        return Suf(k<<1|1,L,R,v);
        else
        return min(Suf(k<<1,L,R,v),Suf(k<<1|1,L,R,v));
    }
    int suf(int L,int R,int v){return Suf(1,L,R,v);}
}tree;
int main()
{
    int n=read_int(),m=read_int(),opt,l,r,pos,k,ans;
```

```
    _rep(i,1,n)
    tree.a[i]=read_int();
    tree.build(n);
    while(m--){
        opt=read_int();
        if(opt==3){
            pos=read_int(),k=read_int();
            tree.update(pos,k);
        }
        else{
            l=read_int(),r=read_int(),k=read_int();
            switch(opt){
                case 1:
                    ans=tree.rank(l,r,k);
                    break;
                case 2:
                    ans=tree.kth(l,r,k);
                    break;
                case 4:
                    ans=tree.pre(l,r,k);
                    break;
                case 5:
                    ans=tree.suf(l,r,k);
                    break;
            }
            enter(ans);
        }
    }
    return 0;
}
```

树状数组套名次树版本

树状数组套名次树 rank、update 操作和普通树状数组类似、kth 操作需要二分 $+$ rank 操作、pre、suf 操作需要 $\text{rank}+\text{kth}$、

rank、update 操作时间复杂度为 $O(\log^2 n)$、pre、suf、kth 操作时间复杂度 $O(\log^2 n\log v)$、

```
const int MAXN=5e4+5,MAXS=MAXN*20,Inf=0x7fffffff;
template <typename T>
struct Treap{
    int pool[MAXS],top,tot;
    int root[MAXN],ch[MAXS][2],r[MAXS],sz[MAXS],cnt[MAXS];
    T val[MAXS];
    int new_node(T v){
        int id=top?pool[top--]:++tot;
        val[id]=v;r[id]=rand();sz[id]=cnt[id]=1;ch[id][0]=ch[id][1]=0;
        return id;
```

```
    }
    void push_up(int id){sz[id]=sz[ch[id][0]]+sz[ch[id][1]]+cnt[id];}
    void Rotate(int &id,int dir){
        int t=ch[id][dir^1];
        ch[id][dir^1]=ch[t][dir];ch[t][dir]=id;id=t;
        push_up(ch[id][dir]);push_up(id);
    }
    void Insert(int &id,T v){
        if(!id)
        return id=new_node(v),void();
        if(v==val[id])
        cnt[id]++;
        else{
            int dir=v<val[id]?0:1;
            Insert(ch[id][dir],v);
            if(r[id]<r[ch[id][dir]])
            Rotate(id,dir^1);
        }
        push_up(id);
    }
    void Erase(int &id,T v){
        if(!id)
        return;
        if(v==val[id]){
            if(cnt[id]>1)return    cnt[id]--,push_up(id);
            else if(!ch[id][0])pool[++top]=id,id=ch[id][1];
            else if(!ch[id][1])pool[++top]=id,id=ch[id][0];
            else{
                int d=r[ch[id][0]]>r[ch[id][1]]?1:0;
                Rotate(id,d);Erase(ch[id][d],v);push_up(id);
            }
        }
        else{
            if(v<val[id])Erase(ch[id][0],v);
            else Erase(ch[id][1],v);
            push_up(id);
        }
    }
    int Rank(int id,T v){//有多少个数严格小于v
        if(!id)return 0;
        if(v==val[id])return sz[ch[id][0]];
        else if(v<val[id])return Rank(ch[id][0],v);
        else return sz[ch[id][0]]+cnt[id]+Rank(ch[id][1],v);
    }
    T Kth(int id,int rk){
        if(!id) return -1;//第rk小的节点不存在
        if(rk>sz[ch[id][0]]+cnt[id]) return Kth(ch[id][1],rk-sz[ch[id][0]]-
cnt[id]);
        else if(rk>sz[ch[id][0]]) return val[id];
        else return Kth(ch[id][0],rk);
```

```
    }
    T Pre(int id,T v){
        int pos=id,ans=-Inf;
        while(pos){
            if(val[pos]<v){
                ans=ans<val[pos]?val[pos]:ans;
                pos=ch[pos][1];
            }
            else pos=ch[pos][0];
        }
        return ans;
    }
    T Suf(int id,T v){
        int pos=id,ans=Inf;
        while(pos){
            if(v<val[pos]){
                ans=ans<val[pos]?ans:val[pos];
                pos=ch[pos][0];
            }
            else pos=ch[pos][1];
        }
        return ans;
    }
    int Count(int id,T v){
        int pos=id;
        while(pos){
            if(v<val[pos])
            pos=ch[pos][0];
            else if(val[pos]<v)
            pos=ch[pos][1];
            else
            return cnt[pos];
        }
        return 0;
    }
    void insert(int root_id,T v){Insert(root[root_id],v);}
    void erase(int root_id,T v){Erase(root[root_id],v);}
    int rank(int root_id,T v){return Rank(root[root_id],v);}//如果需要，记得+1
    T kth(int root_id,int rk){return Kth(root[root_id],rk);}
    T pre(int root_id,T v){return Pre(root[root_id],v);}
    T suf(int root_id,T v){return Suf(root[root_id],v);}
    int count(int root_id,T v){return Count(root[root_id],v);}
};
#define lowbit(x) x&(-x)
struct Tree{
    Treap<int> S;
    int n,a[MAXN];
    void build(int pos,int v){
        while(pos<=n){
            S.insert(pos,v);
            pos+=lowbit(pos);
```

```
        }
    }
    void build(int n){
        this->n=n;
        _rep(i,1,n)
        build(i,a[i]);
    }
    int Rank(int L,int R,int v){
        int ans=0,pos1=L-1,pos2=R;
        while(pos1){
            ans-=S.rank(pos1,v);
            pos1-=lowbit(pos1);
        }
        while(pos2){
            ans+=S.rank(pos2,v);
            pos2-=lowbit(pos2);
        }
        return ans;
    }
    int rank(int L,int R,int v){return Rank(L,R,v)+1;}
    int kth(int L,int R,int rk){
        int lef=0,rig=1e8,mid,trk,ans=0;
        while(lef<=rig){
            mid=lef+rig>>1;
            trk=rank(L,R,mid);
            if(trk<=rk){
                ans=mid;
                lef=mid+1;
            }
            else if(trk>rk)
            rig=mid-1;
        }
        return ans;
    }
    void update(int pos,int v){
        int t=pos;
        while(t<=n){
            S.erase(t,a[pos]);
            S.insert(t,v);
            t+=lowbit(t);
        }
        a[pos]=v;
    }
    int count(int L,int R,int v){
        int ans=0,pos1=L-1,pos2=R;
        while(pos1){
            ans-=S.count(pos1,v);
            pos1-=lowbit(pos1);
        }
        while(pos2){
```

```
            ans+=S.count(pos2,v);
            pos2-=lowbit(pos2);
        }
        return ans;
    }
    int pre(int L,int R,int v){
        int rk=Rank(L,R,v);
        if(rk)
        return kth(L,R,rk);
        else
        return -Inf;
    }
    int suf(int L,int R,int v){
        int rk=rank(L,R,v)+count(L,R,v);
        if(rk<=R-L+1)
        return kth(L,R,rk);
        else
        return Inf;
    }
}tree;
int main()
{
    int n=read_int(),m=read_int(),opt,l,r,pos,k,ans;
    _rep(i,1,n)
    tree.a[i]=read_int();
    tree.build(n);
    while(m--){
        opt=read_int();
        if(opt==3){
            pos=read_int(),k=read_int();
            tree.update(pos,k);
        }
        else{
            l=read_int(),r=read_int(),k=read_int();
            switch(opt){
                case 1:
                    ans=tree.rank(l,r,k);
                    break;
                case 2:
                    ans=tree.kth(l,r,k);
                    break;
                case 4:
                    ans=tree.pre(l,r,k);
                    break;
                case 5:
                    ans=tree.suf(l,r,k);
                    break;
            }
            enter(ans);
        }
    }
```

```
    return 0;
}
```

权值线段树套名次树版本

转换一下思路，考虑外层维护权值，内层维护位置。那么 rank 操作查询 $0\sim v-1$ 区间的满足条件的点的个数。

kth 操作类似在线段树上跑类似名次树的操作update 操作先删除后插入，其他操作基于这两个基础操作即可得到。

空间复杂度 $O\left(n\log v\right)$所有操作时间复杂度 $O\left(\log n\log v\right)$

```
const int MAXN=5e4+5,MAXS=MAXN*40,Inf=0x7fffffff;
template <typename T>
struct Treap{
    int pool[MAXS],top,tot;
    int root[MAXS],ch[MAXS][2],r[MAXS],sz[MAXS],cnt[MAXS];
    T val[MAXS];
    int new_node(T v){
        int id=top?pool[top--]:++tot;
        val[id]=v;r[id]=rand();sz[id]=cnt[id]=1;ch[id][0]=ch[id][1]=0;
        return id;
    }
    void push_up(int id){sz[id]=sz[ch[id][0]]+sz[ch[id][1]]+cnt[id];}
    void Rotate(int &id,int dir){
        int t=ch[id][dir^1];
        ch[id][dir^1]=ch[t][dir];ch[t][dir]=id;id=t;
        push_up(ch[id][dir]);push_up(id);
    }
    void Insert(int &id,T v){
        if(!id)
        return id=new_node(v),void();
        if(v==val[id])
        cnt[id]++;
        else{
            int dir=v<val[id]?0:1;
            Insert(ch[id][dir],v);
            if(r[id]<r[ch[id][dir]])
            Rotate(id,dir^1);
        }
        push_up(id);
    }
    void Erase(int &id,T v){
        if(!id)
        return;
        if(v==val[id]){
            if(cnt[id]>1)
```

```
            return    cnt[id]--,push_up(id);
            else if(!ch[id][0])
            pool[++top]=id,id=ch[id][1];
            else if(!ch[id][1])
            pool[++top]=id,id=ch[id][0];
            else{
                int d=r[ch[id][0]]>r[ch[id][1]]?1:0;
                Rotate(id,d);Erase(ch[id][d],v);push_up(id);
            }
        }
        else{
            if(v<val[id])
            Erase(ch[id][0],v);
            else
            Erase(ch[id][1],v);
            push_up(id);
        }
    }
    int Rank(int id,T v){//有多少个数严格小于v
        if(!id)
        return 0;
        if(v==val[id])
        return sz[ch[id][0]];
        else if(v<val[id])
        return Rank(ch[id][0],v);
        else
        return sz[ch[id][0]]+cnt[id]+Rank(ch[id][1],v);
    }
    T Kth(int id,int rk){
        if(!id)
        return -1;//第rk小的节点不存在
        if(rk>sz[ch[id][0]]+cnt[id])
        return Kth(ch[id][1],rk-sz[ch[id][0]]-cnt[id]);
        else if(rk>sz[ch[id][0]])
        return val[id];
        else
        return Kth(ch[id][0],rk);
    }
    T Pre(int id,T v){
        int pos=id,ans=-Inf;
        while(pos){
            if(val[pos]<v){
                ans=ans<val[pos]?val[pos]:ans;
                pos=ch[pos][1];
            }
            else
            pos=ch[pos][0];
        }
        return ans;
    }
    T Suf(int id,T v){
```

```
        int pos=id,ans=Inf;
        while(pos){
            if(v<val[pos]){
                ans=ans<val[pos]?ans:val[pos];
                pos=ch[pos][0];
            }
            else
            pos=ch[pos][1];
        }
        return ans;
    }
    void insert(int root_id,T v){Insert(root[root_id],v);}
    void erase(int root_id,T v){Erase(root[root_id],v);}
    int rank(int root_id,T v){return Rank(root[root_id],v);}
    T kth(int root_id,int rk){return Kth(root[root_id],rk);}
    T pre(int root_id,T v){return Pre(root[root_id],v);}
    T suf(int root_id,T v){return Suf(root[root_id],v);}
    bool empty(int root_id){return sz[root_id]==0;}
};
const int MAXV=1e8;
struct Tree{
    Treap<int> S;
    int root,a[MAXN],pool[MAXS],top,tot,lson[MAXS],rson[MAXS];
    int New(){
        int k=top?pool[top--]:++tot;
        lson[k]=rson[k]=0;
        return k;
    }
    void Del(int &k){
        pool[++top]=k;
        k=0;
    }
    void Insert(int &k,int lef,int rig,int v,int pos){
        if(!k)k=New();
        S.insert(k,pos);
        if(lef==rig)
        return;
        int mid=lef+rig>>1;
        if(v<=mid)
        Insert(lson[k],lef,mid,v,pos);
        else
        Insert(rson[k],mid+1,rig,v,pos);
    }
    void insert(int pos,int v){Insert(root,0,MAXV,v,pos);}
    void build(int n){
        _rep(i,1,n)
        insert(i,a[i]);
    }
    void Erase(int &k,int lef,int rig,int v,int pos){
        S.erase(k,pos);
```

```
        if(lef==rig){
            if(S.empty(k))
            Del(k);
            return;
        }
        int mid=lef+rig>>1;
        if(v<=mid)
        Erase(lson[k],lef,mid,v,pos);
        else
        Erase(rson[k],mid+1,rig,v,pos);
        if(S.empty(k))
        Del(k);
    }
    void erase(int pos,int v){Erase(root,0,MAXV,v,pos);}
    int Rank(int k,int lef,int rig,int v,int pos1,int pos2){
        if(!k)
        return 0;
        if(rig<=v)
        return S.rank(k,pos2)-S.rank(k,pos1);
        int mid=lef+rig>>1;
        if(mid>=v)
        return Rank(lson[k],lef,mid,v,pos1,pos2);
        else
        return
Rank(lson[k],lef,mid,v,pos1,pos2)+Rank(rson[k],mid+1,rig,v,pos1,pos2);
    }
    int rank(int L,int R,int v){return Rank(root,0,MAXV,v-1,L,R+1)+1;}
    int kth(int L,int R,int rk){
        if(rk==0)
        return -Inf;
        else if(rk>R-L+1)
        return Inf;
        int k=root,lef=0,rig=MAXV,mid,trk;
        R++;rk--;
        while(lef<rig){
            trk=S.rank(lson[k],R)-S.rank(lson[k],L);
            mid=lef+rig>>1;
            if(rk>=trk)
            rk-=trk,k=rson[k],lef=mid+1;
            else
            k=lson[k],rig=mid;
        }
        return lef;
    }
    void update(int pos,int v){
        erase(pos,a[pos]);
        insert(pos,v);
        a[pos]=v;
    }
    int count(int L,int R,int v){
        int k=root,lef=0,rig=MAXV,mid;
```

```
        while(lef!=rig){
            if(!k)
            return 0;
            mid=lef+rig>>1;
            if(v<=mid){
                k=lson[k];
                rig=mid;
            }
            else{
                k=rson[k];
                lef=mid+1;
            }
        }
        return S.rank(k,R+1)-S.rank(k,L);
    }
    int pre(int L,int R,int v){return kth(L,R,rank(L,R,v)-1);}
    int suf(int L,int R,int v){return kth(L,R,rank(L,R,v)+count(L,R,v));}
}tree;
int main()
{
    int n=read_int(),m=read_int(),opt,l,r,pos,k,ans;
    _rep(i,1,n)
    tree.a[i]=read_int();
    tree.build(n);
    while(m--){
        opt=read_int();
        if(opt==3){
            pos=read_int(),k=read_int();
            tree.update(pos,k);
        }
        else{
            l=read_int(),r=read_int(),k=read_int();
            switch(opt){
                case 1:
                    ans=tree.rank(l,r,k);
                    break;
                case 2:
                    ans=tree.kth(l,r,k);
                    break;
                case 4:
                    ans=tree.pre(l,r,k);
                    break;
                case 5:
                    ans=tree.suf(l,r,k);
                    break;
            }
            enter(ans);
        }
    }
    return 0;
```

```
}
```

树状数组套权值线段树

简介

功能类似线段树/树状数组套名次树。

其实权值线段树和名次树在功能上有许多相同点，权值线段树码量相对较少，但空间复杂度多一个 $O(\log v)$

模板题

洛谷p3759

该题等价于维护一个数组，支持一下操作：

1. 修改某个点的点权
2. 询问满足位置在 $[l_1,r_1]$ 且权值在 $[l_2,r_2]$ 范围内的点个数和点权和。

树状数组的每个节点的权值线段树维护区间 $[x-\text{lowbit}(x)+1,x]$ 的权值分布，每个权值记录点个数和点权和。

空间复杂度 $O(n\log n\log v)$单次操作时间复杂度 $O(\log n\log v)$

```
const int MAXN=5e4+5,MAXM=400,mod=1e9+7;
struct Node{
    int ch[2],sz,sum;
}node[MAXN*MAXM];
struct Ans{
    int sz,sum;
    Ans operator + (const Ans &b){
        Ans c;
        c.sz=sz+b.sz;
        c.sum=sum+b.sum;
        if(c.sum>=mod)c.sum-=mod;
        return c;
    }
    Ans operator - (const Ans &b){
        Ans c;
        c.sz=sz-b.sz;
        c.sum=sum-b.sum;
        if(c.sum<0)c.sum+=mod;
        return c;
    }
    void operator += (const Ans &b){
        sz+=b.sz;
        sum+=b.sum;
```

```
        if(sum>=mod)sum-=mod;
    }
    void operator -= (const Ans &b){
        sz-=b.sz;
        sum-=b.sum;
        if(sum<0)sum+=mod;
    }
    Ans(int sz=0,int sum=0):sz(sz),sum(sum){}
};
#define lowbit(x) (x)&(-x)
int n,root[MAXN],tot;
int a[MAXN],b[MAXN],c[MAXN],d[MAXN];
void add(int pos,int v){
    while(pos<=n){
        c[pos]+=v;
        if(c[pos]>=mod)
        c[pos]-=mod;
        d[pos]++;
        pos+=lowbit(pos);
    }
}
Ans sum(int pos){
    int s1=0,s2=0;
    while(pos){
        s2+=c[pos];
        if(s2>=mod)
        s2-=mod;
        s1+=d[pos];
        pos-=lowbit(pos);
    }
    return Ans(s1,s2);
}
void update_1D(int &k,int lef,int rig,int pos,int v1,int v2){
    if(!k)k=++tot;
    node[k].sz+=v2,node[k].sum=(node[k].sum+v1)%mod;
    if(lef==rig)
    return;
    int mid=lef+rig>>1;
    if(mid>=pos)
    update_1D(node[k].ch[0],lef,mid,pos,v1,v2);
    else
    update_1D(node[k].ch[1],mid+1,rig,pos,v1,v2);
}
void update_2D(int k,int pos,int v1,int v2){
    while(k<=n){
        update_1D(root[k],1,n,pos,v1,v2);
        k+=lowbit(k);
    }
}
Ans query_1D(int k,int lef,int rig,int L,int R){
```

```
    if(!k)
    return Ans(0,0);
    if(L<=lef&&rig<=R)
    return Ans(node[k].sz,node[k].sum);
    int mid=lef+rig>>1;
    if(mid>=R)
    return query_1D(node[k].ch[0],lef,mid,L,R);
    else if(mid<L)
    return query_1D(node[k].ch[1],mid+1,rig,L,R);
    return
query_1D(node[k].ch[0],lef,mid,L,R)+query_1D(node[k].ch[1],mid+1,rig,L,R);
}
Ans query_2D(int L,int R,int ql,int qr){
    if(ql>qr)
    return Ans(0,0);
    int pos1=L-1,pos2=R;
    Ans re=Ans(0,0);
    while(pos1){
        re-=query_1D(root[pos1],1,n,ql,qr);
        pos1-=lowbit(pos1);
    }
    while(pos2){
        re+=query_1D(root[pos2],1,n,ql,qr);
        pos2-=lowbit(pos2);
    }
    return re;
}
int main()
{
    n=read_int();
    int q=read_int(),lef,rig;
    LL ans=0;Ans temp;
    _rep(i,1,n)
    a[i]=read_int(),b[i]=read_int();
    for(int i=n;i;i--){
        add(a[i],b[i]);
        temp=sum(a[i]-1);
        ans=(ans+temp.sum+1LL*b[i]*temp.sz)%mod;
        update_2D(i,a[i],b[i],1);
    }
    while(q--){
        lef=read_int();
        rig=read_int();
        if(lef==rig){
            enter(ans);
            continue;
        }
        if(lef>rig)
        swap(lef,rig);
        temp=query_2D(lef+1,rig-1,a[lef]+1,n)-
query_2D(lef+1,rig-1,1,a[lef]-1);
```

```
        ans=(ans+temp.sum+1LL*b[lef]*temp.sz)%mod;
        temp=query_2D(lef+1,rig-1,1,a[rig]-1)-
query_2D(lef+1,rig-1,a[rig]+1,n);
        ans=(ans+temp.sum+1LL*b[rig]*temp.sz)%mod;
        if(a[lef]<a[rig])
        ans=(ans+b[lef]+b[rig])%mod;
        else
        ans=(ans-b[lef]-b[rig])%mod;
        enter((ans+mod)%mod);
        update_2D(lef,a[lef],-b[lef],-1);update_2D(rig,a[rig],-b[rig],-1);
        swap(a[lef],a[rig]);swap(b[lef],b[rig]);
        update_2D(lef,a[lef],b[lef],1);update_2D(rig,a[rig],b[rig],1);
    }
    return 0;
}
```