主席树是很简（du）单（liu）的数据结构

题目给你一个序列，每次修改后算一个新的版本，询问某个版本中某个值

我们先以Luogu P3919 【模板】可持久化数组（可持久化线段树/平衡树）作为模板讲一下主席树

主席树（可持久化线段树）

先学一下线段树qaq

主席树本名可持久化线段树，也就是说，主席树是基于线段树发展而来的一种数据结构。其前缀"可持久化"意在给线段树增加一些历史点来维护历史数据，使得我们能在较短时间内查询历史数据

不同于普通线段树的是主席树的左右子树节点编号并不能够用计算得到，所以我们需要记录下来，但是对应的区间还是没问题的。

我们注意到，对于修改操作，当前版本与它的前驱版本相比，只更改了一个节点的值，其他大多数节点的值没有变化。

能不能重复利用，以达到节省空间的目的?

——分治？没错，如果只修改了左半边，那么我们可以使用前驱版本的右半边，反之同理。

于是，我们就可以用线段树，进行修改操作时，只要当前节点的左（右）儿子没有被修改，我们就可以使用前驱版本的那个节点。

那查找呢？每次保存版本i的根节点，利用线段树的方法查找就好了。

代码实现（代码中有详细注释qaq）：

```
#include <bits/stdc++.h>
#define N 1000005
using namespace std;
inline char nc(){
    static char buf[100000],*p1=buf,*p2=buf;
    return p1==p2&&(p2=(p1=buf)+fread(buf,1,100000,stdin),p1==p2)?EOF:*p1++;
}
inline int read()
{
    register int x=0,f=1;register char ch=nc();
    while(ch<'0'||ch>'9'){if(ch=='-')f=-1;ch=nc();}
    while(ch>='0'&&ch<='9')x=(x<<3)+(x<<1)+ch-'0',ch=nc();
    return x*f;
}
inline void write(register int x)
{
    if(!x)putchar('0');if(x<0)x=-x,putchar('-');
    static int sta[20];register int tot=0;
    while(x)sta[tot++]=x%10,x/=10;
    while(tot)putchar(sta[--tot]+48);
```

```
}
struct node{
    int rt[N],t[N<<5],ls[N<<5],rs[N<<5];
    int cnt;//尾节点，插入节点用
    inline int build(register int l,register int r)
    {
        int root=++cnt;
        if(l==r)
        {
            t[root]=read();//顺带读入
            return root;
        }
        int mid=l+r>>1;
        ls[root]=build(l,mid),rs[root]=build(mid+1,r);
        return root;
    }
    inline int update(register int pre,register int l,register int
r,register int x,register int c)
    {
        int root=++cnt;
        if(l==r)
        {
            t[root]=c; //修改
            return root;
        }
        ls[root]=ls[pre],rs[root]=rs[pre];//先把子节点指向前驱结点以备复用
        int mid=l+r>>1;
        if(x<=mid)
            ls[root]=update(ls[pre],l,mid,x,c);
        else
            rs[root]=update(rs[pre],mid+1,r,x,c);
        return root;
    }
    inline void query(register int pre,register int l,register int
r,register int x)
    {
        //普通的线段树查询
        if(l==r)
        {
            write(t[pre]),puts("");
            return;
        }
        int mid=l+r>>1;
        if(x<=mid)
            query(ls[pre],l,mid,x);
        else
            query(rs[pre],mid+1,r,x);
    }
}tr;
int main()
{
    tr.cnt=0;
    int n=read(),m=read();
    tr.build(1,n);
    tr.rt[0]=1;
    for(register int i=1;i<=m;++i)
    {
        int tic=read(),opt=read();
```

```
        if(opt==1)
        {
            int pos=read(),v=read();
            tr.rt[i]=tr.update(tr.rt[tic],1,n,pos,v);
        }
        else
        {
            int pos=read();
            tr.rt[i]=tr.rt[tic];
            tr.query(tr.rt[tic],1,n,pos);
        }
    }
    return 0;
}
```

还有一种问题是求静态区间[l,r]中第k小的数

先给一个很暴力的做法：

先将区间进行排序（莫队），再用平衡树来求区间第k小

这样的复杂度是 O(nn−−√logn)O(nnlogn)

如果你有足够的卡常技巧（A了挑战），也许能卡过[Luogu P3834 【模板】可持久化线段树 1（主席树）]

50分莫队+平衡树做法

```
#pragma GCC optimize("O3")
#include <bits/stdc++.h>
#define N 500005
#define M 200005
using namespace std;
inline char nc(){
    static char buf[100000],*p1=buf,*p2=buf;
    return p1==p2&&(p2=(p1=buf)+fread(buf,1,100000,stdin),p1==p2)?
EOF:*p1++;
}
inline int read()
{
    register int x=0,f=1;register char ch=nc();
    while(ch<'0'||ch>'9'){if(ch=='-')f=-1;ch=nc();}
    while(ch>='0'&&ch<='9')x=(x<<3)+(x<<1)+ch-'0',ch=nc();
    return x*f;
}
inline void write(register int x)
{
    if(!x)putchar('0');if(x<0)x=-x,putchar('-');
    static int sta[20];register int tot=0;
    while(x)sta[tot++]=x%10,x/=10;
    while(tot)putchar(sta[--tot]+48);
}
struct Splay{
    int v,fa,ch[2],sum,rec;
}tree[N];
int tot=0;
inline void update(register int x)
```

```
{

 tree[x].sum=tree[tree[x].ch[0]].sum+tree[tree[x].ch[1]].sum+tree[x].rec;
}
inline bool findd(register int x)
{
    return tree[tree[x].fa].ch[0]==x?0:1;
}
inline void connect(register int x,register int fa,register int son)
{
    tree[x].fa=fa;
    tree[fa].ch[son]=x;
}
inline void rotate(register int x)
{
    int Y=tree[x].fa;
    int R=tree[Y].fa;
    int Yson=findd(x);
    int Rson=findd(Y);
    int B=tree[x].ch[Yson^1];
    connect(B,Y,Yson);
    connect(Y,x,Yson^1);
    connect(x,R,Rson);
    update(Y),update(x);
}
inline void splay(register int x,register int to)
{
    to=tree[to].fa;
    while(tree[x].fa!=to)
    {
        int y=tree[x].fa;
        if(tree[y].fa==to)
            rotate(x);
        else if(findd(x)==findd(y))
            rotate(y),rotate(x);
        else
            rotate(x),rotate(x);
    }
}
inline int newpoint(register int v,register int fa)
{
    tree[++tot].fa=fa;
    tree[tot].v=v;
    tree[tot].sum=tree[tot].rec=1;
    return tot;
}
inline void Insert(register int x)
{
    int now=tree[0].ch[1];
    if(tree[0].ch[1]==0)
    {
        newpoint(x,0);
        tree[0].ch[1]=tot;
    }
    else
    {
        while(19260817)
        {
```

```
            ++tree[now].sum;
            if(tree[now].v==x)
            {
                ++tree[now].rec;
                splay(now,tree[0].ch[1]);
                return;
            }
            int nxt=x<tree[now].v?0:1;
            if(!tree[now].ch[nxt])
            {
                int p=newpoint(x,now);
                tree[now].ch[nxt]=p;
                splay(p,tree[0].ch[1]);
                return;
            }
            now=tree[now].ch[nxt];
        }
    }
}
inline int find(register int v)
{
    int now=tree[0].ch[1];
    while(19260817)
    {
        if(tree[now].v==v)
        {
            splay(now,tree[0].ch[1]);
            return now;
        }
        int nxt=v<tree[now].v?0:1;
        if(!tree[now].ch[nxt])
            return 0;
        now=tree[now].ch[nxt];
    }
}
inline void delet(register int x)
{
    int pos=find(x);
    if(!pos)
        return;
    if(tree[pos].rec>1)
    {
        --tree[pos].rec;
        --tree[pos].sum;
    }
    else
    {
        if(!tree[pos].ch[0]&&!tree[pos].ch[1])
            tree[0].ch[1]=0;
        else if(!tree[pos].ch[0])
        {
            tree[0].ch[1]=tree[pos].ch[1];
            tree[tree[0].ch[1]].fa=0;
        }
        else
        {
            int left=tree[pos].ch[0];
            while(tree[left].ch[1])
```

```
                left=tree[left].ch[1];
            splay(left,tree[pos].ch[0]);
            connect(tree[pos].ch[1],left,1);
            connect(left,0,1);
            update(left);
        }
    }
}
inline int arank(register int x)
{
    int now=tree[0].ch[1];
    while(19260817)
    {
        int used=tree[now].sum-tree[tree[now].ch[1]].sum;
        if(x>tree[tree[now].ch[0]].sum&&x<=used)
        {
            splay(now,tree[0].ch[1]);
            return tree[now].v;
        }
        if(x<used)
            now=tree[now].ch[0];
        else
            x-=used,now=tree[now].ch[1];
    }
}
struct query{
    int l,r,id,bl,k;
}q[M];
int a[N],blocksize=0,ans[M];
inline bool cmp(register query a,register query b)
{
    return a.bl!=b.bl?a.l<b.l:((a.bl&1)?a.r<b.r:a.r>b.r);
}
int main()
{
    int n=read(),m=read();
    blocksize=sqrt(m);
    for(register int i=1;i<=n;++i)
        a[i]=read();
    for(register int i=1;i<=m;++i)
    {
        int l=read(),r=read(),k=read();
        q[i]=(query){l,r,i,l/blocksize,k};
    }
    sort(q+1,q+m+1,cmp);
    int l=1,r=0;
    for(register int i=1;i<=m;++i)
    {
        int ll=q[i].l,rr=q[i].r;
        while(ll<l)
            Insert(a[--l]);
        while(rr>r)
            Insert(a[++r]);
        while(ll>l)
            delet(a[l++]);
        while(rr<r)
            delet(a[r--]);
        ans[q[i].id]=arank(q[i].k);
```

```
    }
    for(register int i=1;i<=m;++i)
        write(ans[i]),puts("");
    return 0;
}
```

我们先考虑简化的问题：我们要询问整个区间内的第K小。这样我们对值域建线段树，每个节点记录这个区间所包含的元素个数，建树和查询时的区间范围用递归参数传递，然后用二叉查找树的询问方式即可：即如果左边元素个数sum>=K，递归查找左子树第K小，否则递归查找右子树第K - sum小，直到返回叶子的值。

现在我们要回答对于区间[l, r]的第K小询问。如果我们能够得到一个插入原序列中[1, l - 1]元素的线段树，和一颗插入了[1, r]元素的线段树，由于线段树是开在值域上，区间长度是一定的，所以结构也必然是完全相同的，我们可以直接对这两颗线段树进行相减，得到的是相当于插入了区间[l ,r]元素的线段树。注意这里利用到的区间相减性质，实际上是用两颗不同历史版本的线段树进行相减：一颗是插入到第l-1个元素的旧树，一颗是插入到第r元素的新树。

这样相减之后得到的是相当于只插入了原序列中[l, r]元素的一颗记录了区间数字个数的线段树。直接对这颗线段树按照BST的方式询问，即可得到区间第k小。

这种做法是可行的，但是我们显然不能每次插入一个元素，就从头建立一颗全新的线段树，否则内存开销无法承受。事实上，每次插入一个新的元素时，我们不需要新建所有的节点，而是只新建增加的节点。也就是从根节点出发，先新建节点并复制原节点的值，然后进行修改即可。

这样我们我们每到一个节点，只需要修改左儿子或者右儿子其一的信息，一直递归到叶子后结束，修改的节点数量就是树高，也就是新建了不超过树高个节点，内存开销就可以承受了。

注意我们对root[0]也就是插入了零个元素的那颗树，记录的左右儿子指针都是0，这样我们就可以用这一个节点表示一个任意结构的空树而不需要显式建树。这是因为对于这个节点，不管你再怎么递归，都是指向这个节点本身，里面记录的元素个数就是零。

```
#include <bits/stdc++.h>
#define N 200005
using namespace std;
inline char nc(){
    static char buf[100000],*p1=buf,*p2=buf;
    return p1==p2&&(p2=(p1=buf)+fread(buf,1,100000,stdin),p1==p2)?EOF:*p1++;
}
inline int read()
{
    register int x=0,f=1;register char ch=getchar();
    while(ch<'0'||ch>'9'){if(ch=='-')f=-1;ch=getchar();}
    while(ch>='0'&&ch<='9')x=(x<<3)+(x<<1)+ch-'0',ch=getchar();
    return x*f;
}
inline void write(register int x)
{
    if(!x)putchar('0');
    static int sta[20];register int tot=0;
```

```
    while(x)sta[tot++]=x%10,x/=10;
    while(tot)putchar(sta[--tot]+48);
}
int n,q,m,cnt=0;
int a[N],b[N],T[N];
int sum[N<<5],ls[N<<5],rs[N<<5];
inline int build(register int l,register int r)
{
    int root=++cnt;
    sum[root]=0;
    int mid=l+r>>1;
    if(l<r)
        ls[root]=build(l,mid),rs[root]=build(mid+1,r);
    return root;
}
inline int update(register int pre,register int l,register int r,register
int x)
{
    int root=++cnt;
    ls[root]=ls[pre],rs[root]=rs[pre],sum[root]=sum[pre]+1;
    int mid=l+r>>1;
    if(l<r)
    {
        if(x<=mid)
            ls[root]=update(ls[pre],l,mid,x);
        else
            rs[root]=update(rs[pre],mid+1,r,x);
    }
    return root;
}
inline int query(register int u,register int v,register int l,register int
r,register int k)
{
    if(l>=r)
        return l;
    int x=sum[ls[v]]-sum[ls[u]];
    int mid=l+r>>1;
    if(x>=k)
        return query(ls[u],ls[v],l,mid,k);
    else
        return query(rs[u],rs[v],mid+1,r,k-x);
}
int main()
{
    n=read(),q=read();
    for(register int i=1;i<=n;++i)
        b[i]=a[i]=read();
    sort(b+1,b+n+1);
    m=unique(b+1,b+n+1)-b-1;
    T[0]=build(1,m);
    for(register int i=1;i<=n;++i)
    {
        int t=lower_bound(b+1,b+m+1,a[i])-b;
        T[i]=update(T[i-1],1,m,t);
    }
    while(q--)
    {
        int l=read(),r=read(),k=read();
```

```
        int t=query(T[l-1],T[r],1,m,k);
        write(b[t]),puts("");
    }
}
```

但是要注意，主席树在不做额外处理时只能查询静态的区间k大（小）值。

接下来，我们就考虑动态区间k小值。如果我们要对区间进行修改的话，一个简单的主席树已经无法实现了。

如果对原来的节点直接修改的话，会造成不可名状的运行错误（有兴趣的同学可以结合上面插入代码想一想为什么），

空间和时间也无法接受（我们需要把后面所有树都更改一下），但我们在做树套树的时候，可以做类似的操作，那么主席树是不是应该也套些什么呢?

主席树上的点，储存的都是在一段权值区间内的数据个数，我们必须要维护数据个数才可以通过相减得到一段区间的权值线段树。

而现在有了修改，对于这个修改的维护，朴素的做法有2种：O(1)查询，O(n)维护（扫一遍），和O(n)查询（现场算）和O(1)维护。

这两种做法都不是很忧，所以我们考虑利用快捷维护前缀和的[树状数组]解决这个问题，即所谓"树状数组套主席树"

```
#include <bits/stdc++.h>
#define N 100005
#define M 40000005
using namespace std;
inline int read()
{
    register int x=0,f=1;register char ch=getchar();
    while(ch<'0'||ch>'9'){if(ch=='-')f=-1;ch=getchar();}
    while(ch>='0'&&ch<='9')x=(x<<3)+(x<<1)+ch-'0',ch=getchar();
    return x*f;
}
inline void write(register int x)
{
    if(!x)putchar('0');if(x<0)x=-x,putchar('-');
    static int sta[25];int tot=0;
    while(x)sta[tot++]=x%10,x/=10;
    while(tot)putchar(sta[--tot]+48);
}
int a[N];
int n,m;
int root[N],ls[M],rs[M],c[M];
int tot=0;
int xx[40],yy[40];
int v,d;
inline int lowbit(register int x)
{
    return x&(-x);
}
inline void update(register int &now,register int l,register int r)
{
    if(now==0)
```

```
        now=++tot;
    c[now]+=d;
    if(l==r)
        return;
    int mid=l+r>>1;
    if(v<=mid)
        update(ls[now],l,mid);
    else
        update(rs[now],mid+1,r);
}
inline void change()
{
    int x=read(),b=read();
    d=-1,v=a[x];
    for(register int i=x;i<=n;i+=lowbit(i))
        update(root[i],0,1e9);
    d=1,v=b;
    for(register int i=x;i<=n;i+=lowbit(i))
        update(root[i],0,1e9);
    a[x]=b;
}
inline int query()
{
    int x=read(),y=read(),k=read();
    --x;
    x^=y^=x^=y;
    int t1=0,t2=0;
    for(register int i=x;i>=1;i-=lowbit(i))
        xx[++t1]=root[i];
    for(register int i=y;i>=1;i-=lowbit(i))
        yy[++t2]=root[i];
    int l=0,r=1e9;
    while(l<r)
    {
        int temp=0;
        for(register int i=1;i<=t1;++i)
            temp+=c[ls[xx[i]]];
        for(register int i=1;i<=t2;++i)
            temp-=c[ls[yy[i]]];
        if(k<=temp)
        {
            for(register int i=1;i<=t1;++i)
                xx[i]=ls[xx[i]];
            for(register int i=1;i<=t2;++i)
                yy[i]=ls[yy[i]];
            r=l+r>>1;
        }
        else
        {
            for(register int i=1;i<=t1;++i)
                xx[i]=rs[xx[i]];
            for(register int i=1;i<=t2;++i)
                yy[i]=rs[yy[i]];
            k-=temp;
            l=(l+r>>1)+1;
        }
    }
    return l;
```

```
}
int main()
{
    n=read(),m=read();
    for(register int i=1;i<=n;++i)
    {
        v=read();
        a[i]=v,d=1;
        for(register int j=i;j<=n;j+=lowbit(j))
            update(root[j],0,1e9);
    }
    while(m--)
    {
        char ch=getchar();
        while(ch!='C'&&ch!='Q')
            ch=getchar();
        if(ch=='Q')
            write(query()),puts("");
        else
            change();
    }
    return 0;
}
```

标签: 主席树