https://www.luogu.com.cn/blog/bfqaq/qian-tan-shu-zhuang-shuo-zu-quan-zhi-shu

浅谈树状数组套权值树

**posted on 2020-01-31 22:58:43 | under**[**未分类**](https://www.luogu.com.cn/blog/bfqaq/#type=%E6%9C%AA%E5%88%86%E7%B1%BB)**|** 15

**浅谈树状数组套权值树**

前置芝士：权值树。

（不会权值树的同学可以阅读一下[往期日报](https://www.luogu.com.cn/blog/your-alpha1022/WeightSegmentTree-ChairmanTree)或是[我的博客](https://www.luogu.com.cn/blog/bfqaq/qian-tan-quan-zhi-xian-duan-shu)）

（本文树状数组可使用线段树等数据结构代替）

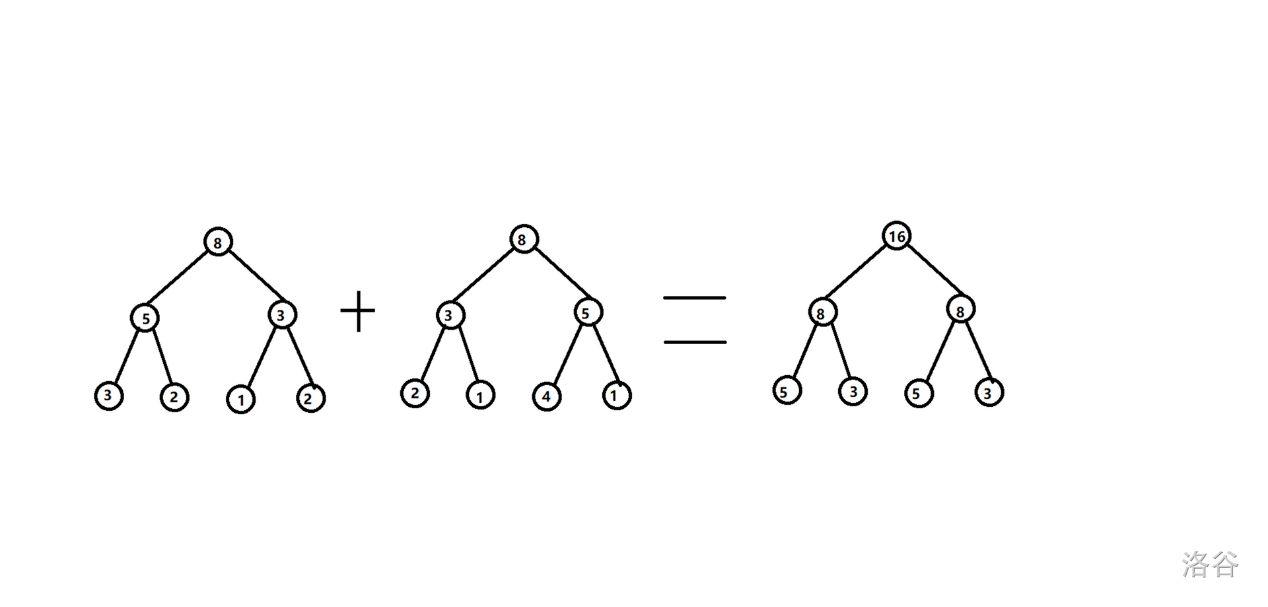
**树状数组套权值树基础操作**

（注：使用权值树的条件是值域已知。这里设最大权值为 len*len*）

众所周知，权值树是一类神奇的数据结构，他的神奇之处在于有可加性和可减性。

因为在值域不变的情况下，权值线段树的形态不发生改变，

所以我们将两棵权值树相加，也就是将他们的对应点相加（如图）。



因此，我们可以直接将权值树当作一种数据类型，用相应的数据结构维护。

模版：<https://www.luogu.com.cn/problem/P3380>

这道题最经典的做法是线段树套平衡树，但我们同样可以用树状数组套权值树来解决。

我们对序列的每一个位置开一个权值树，然后使用树状数组进行维护。

每次操作先跑一边树状数组处理出这次操作涉及到哪几棵树，

然后直接进行权值树的查询即可。

（没错就是这么简单）

**单点修改**

对于单点修改，我们先要在树状数组上处理出需要修改哪几棵树。

这步操作与树状数组的基本操作相同，只是将树状数组的加减变成了线段树的修改：

int lb(int x){

return x&(-x);

}

//……

void add(int o,int v){

for(int i=o;i<=n;i+=lb(i)) change(rt[i],1,len,a[o],v);

}

紧接着就是权值树的修改操作，毫无改动的放上去：

void pushup(int o){

t[o].v=t[t[o].ls].v+t[t[o].rs].v;

}

void change(int &o,int l,int r,int k,int v){

if(!o) o=++tot;

if(l==r){

t[o].v+=v;

return ;

}

int mid=l+r>>1;

if(k<=mid) change(t[o].ls,l,mid,k,v);

else change(t[o].rs,mid+1,r,k,v);

pushup(o);

}

当然，此处有两种写法。一种是像我一样，在每次改一棵树。

也可以先将所有要修改的根节点记录下，一起修改。

在接下来的查询操作中，我使用的就是第二种写法。

**查询操作**

在这儿以查询kth为例：

同样的，先预处理出设计到的树。与上面不同的是，我们将它记录下来：

int find\_num(int l,int r,int k){

cnt=num=0;

for(int i=r;i;i-=lb(i)){

tem[++cnt]=rt[i];

}

for(int i=l-1;i;i-=lb(i)){

tmp[++num]=rt[i];

}

return query\_num(1,len,k);

}

然后，我们一起查询，同权值树kth查询。先统计左子树的个数，判断在左子树还是右子树，然后递归寻找。

int query\_num(int l,int r,int k){

if(l==r) {

return l;

}

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

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

for(int i=1;i<=num;i++) sum-=t[t[tmp[i]].ls].v;

//统计左子树的个数

if(k<=sum){

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

for(int i=1;i<=num;i++) tmp[i]=t[tmp[i]].ls;

//进入左子树,所有的根全部进入左子树

return query\_num(l,mid,k);

}

else{

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

for(int i=1;i<=num;i++) tmp[i]=t[tmp[i]].rs;

//进入右子树,所有的根全部进入右子树

return query\_num(mid+1,r,k-sum);

}

}

其他的查询操作亦是如此，在此不一一解释，直接贴代码：

#include <bits/stdc++.h>

using namespace std;

inline int read(){

register int x=0;

register bool f=0;

register char c=getchar();

while(c<'0'||c>'9'){

if(c=='-') f=1;

c=getchar();

}

while(c>='0'&&c<='9'){

x=(x<<3)+(x<<1)+c-48;

c=getchar();

}

return f?-x:x;

}

const int maxn=50005;

int len=0;

const int inf=2147483647;

struct seg{

int v,ls,rs;

}t[maxn\*100];

int rt[maxn],n,m,tot,tem[maxn],tmp[maxn],cnt,num;

int lsh[maxn<<1],a[maxn];

struct cz{

int a,b,c,d;

}q[maxn];

int lb(int x){

return x&(-x);

}

void pushup(int o){

t[o].v=t[t[o].ls].v+t[t[o].rs].v;

}

void change(int &o,int l,int r,int k,int v){

if(!o) o=++tot;

if(l==r){

t[o].v+=v;

return ;

}

int mid=l+r>>1;

if(k<=mid) change(t[o].ls,l,mid,k,v);

else change(t[o].rs,mid+1,r,k,v);

pushup(o);

}

void add(int o,int v){

for(int i=o;i<=n;i+=lb(i)) change(rt[i],1,len,a[o],v);

}

int query\_num(int l,int r,int k){

if(l==r) {

return l;

}

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

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

for(int i=1;i<=num;i++) sum-=t[t[tmp[i]].ls].v;

if(k<=sum){

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

for(int i=1;i<=num;i++) tmp[i]=t[tmp[i]].ls;

return query\_num(l,mid,k);

}

else{

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

for(int i=1;i<=num;i++) tmp[i]=t[tmp[i]].rs;

return query\_num(mid+1,r,k-sum);

}

}

int find\_num(int l,int r,int k){

cnt=num=0;

for(int i=r;i;i-=lb(i)){

tem[++cnt]=rt[i];

}

for(int i=l-1;i;i-=lb(i)){

tmp[++num]=rt[i];

}

return query\_num(1,len,k);

}

int query\_rnk(int l,int r,int k){

if(l==r) {

return 0;

}

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

if(k<=mid){

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

for(int i=1;i<=num;i++) tmp[i]=t[tmp[i]].ls;

return query\_rnk(l,mid,k);

}

else{

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

for(int i=1;i<=num;i++) sum-=t[t[tmp[i]].ls].v,tmp[i]=t[tmp[i]].rs;

return sum+query\_rnk(mid+1,r,k);

}

}

int find\_rnk(int l,int r,int k){

cnt=num=0;

for(int i=r;i;i-=lb(i)){

tem[++cnt]=rt[i];

}

for(int i=l-1;i;i-=lb(i)){

tmp[++num]=rt[i];

}

return query\_rnk(1,len,k)+1;

}

int find\_pri(int l,int r,int k){

int rk=find\_rnk(l,r,k)-1;

if(rk==0) return 0;

return find\_num(l,r,rk);

}

int find\_nxt(int l,int r,int k){

if(k==len) return len+1;

int rk=find\_rnk(l,r,k+1);

if(rk==r-l+2) return len+1;

return find\_num(l,r,rk);

}

signed main(){

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

tot=cnt=num=0;

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

a[i]=read();

lsh[++len]=a[i];

}

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

q[i].a=read();q[i].b=read();q[i].c=read();

if(q[i].a!=3) q[i].d=read();

else lsh[++len]=q[i].c;

if(q[i].a==4 || q[i].a==5) lsh[++len]=q[i].d;

}

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

len=unique(lsh+1,lsh+len+1)-lsh-1;

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

a[i]=lower\_bound(lsh+1,lsh+1+len,a[i])-lsh;

add(i,1);

}

lsh[0]=-inf;

lsh[len+1]=inf;

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

if(q[i].a==3){

add(q[i].b,-1);

a[q[i].b]=lower\_bound(lsh+1,lsh+1+len,q[i].c)-lsh;

add(q[i].b,1);

}

if(q[i].a==1){

q[i].d=lower\_bound(lsh+1,lsh+1+len,q[i].d)-lsh;

printf("%d\n",find\_rnk(q[i].b,q[i].c,q[i].d));

}

if(q[i].a==2){

printf("%d\n",lsh[find\_num(q[i].b,q[i].c,q[i].d)]);

}

if(q[i].a==4){

q[i].d=lower\_bound(lsh+1,lsh+1+len,q[i].d)-lsh;

printf("%d\n",lsh[find\_pri(q[i].b,q[i].c,q[i].d)]);

}

if(q[i].a==5){

q[i].d=lower\_bound(lsh+1,lsh+1+len,q[i].d)-lsh;

printf("%d\n",lsh[find\_nxt(q[i].b,q[i].c,q[i].d)]);

}

}

return 0;

}

**复杂度分析**

对于每次操作，我们需要在树状数组上预处理出我们需要修改哪些数，

显然这一步的复杂度是 \log nlog*n*。

然后在权值树中查询，这一步的复杂度同权值树，为 \log lenlog*len*。

所以总复杂度就是 \operatorname{O}((n+m)\log n\log len)O((*n*+*m*)log*n*log*len*)。

对于空间复杂度，我们需要使用动态开点。

对于每次插入和修改操作，树状数组预处理涉及到 \log nlog*n* 棵树，

然后每一步修改只涉及到一条链，也就是只需要 \log lenlog*len* 的空间。

在最坏情况下，空间的复杂度为 \operatorname{O}((n+m)\log n\log len)O((*n*+*m*)log*n*log*len*)，

当然实际情况远小于这一上界。

**几句闲话**

相比线段树套平衡树，这一算法的优势是时间复杂度小（因为树套树有一个操作是 \log^3log3 的复杂度）且码量较小。

而劣势是在空间复杂度上多一个 \loglog，且在未知值域的情况下（比如这题）无法在线。

应该说，两者各有各的优势。



（上面是线段树树套平衡树树，下面树状数组套权值树）

另外，如果我们将题中"在序列中的位置"看成一个维度，将"数值大小"看成另一个维度，

那么我们可以理解为本题的查询排名操作就是一个动态的二维偏序。

**一些例题**

* [陌上花开](https://www.luogu.com.cn/problem/P3810)

我们刚才提到，树状数组套权值树可以在动态的情况下维护二维偏序。

回想曾经的静态二维偏序题，我们的做法是直接使用排序消掉一个维度，然后权值树维护另一个维度。

那么同样，我们可以用排序消除掉其中某个维度的影响，然后直接用权值树来跑一个动态二维偏序。

用树状数组对其中某一维度进行维护，再用权值树维护另一维度。

#include<bits/stdc++.h>

#define qwq while(1) puts("qwq");

using namespace std;

inline int read(){

register int x=0;

register bool f=0;

register char c=getchar();

while(c<'0'||c>'9'){

if(c=='-') f=1;

c=getchar();

}

while(c>='0'&&c<='9'){

x=(x<<3)+(x<<1)+c-48;

c=getchar();

}

return f?-x:x;

}

const int maxn=100005;

int n,k,ans[maxn],tot,root[maxn<<2],tmp[maxn<<2],cnt,f[maxn];

struct \_node{

int x,y,z,id;

friend bool operator <(\_node aa,\_node bb){

if(aa.z==bb.z){

if(aa.y==bb.y) return aa.x<bb.x;

return aa.y<bb.y;

}

return aa.z<bb.z;

}

}node[maxn];

struct \_tree{

int ls,rs,v;

}t[50000000];

int lb(int x){

return x&-x;

}

void pushup(int o){

t[o].v=t[t[o].ls].v+t[t[o].rs].v;

}

void change(int &o,int l,int r,int w,int v){

if(!o) o=++tot;

if(l==r){

t[o].v+=v;

return ;

}

int mid=l+r>>1;

if(w<=mid) change(t[o].ls,l,mid,w,v);

else change(t[o].rs,mid+1,r,w,v);

pushup(o);

}

void add(int o,int w,int v){

for(int i=o;i<=k;i+=lb(i)) change(root[i],1,k,w,v);

}

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

if(l==r) {

int res=0;

for(int i=1;i<=cnt;i++) res+=t[tmp[i]].v;

return res;

}

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

if(w<=mid){

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

return query(l,mid,w);

}

else{

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

return sum+query(mid+1,r,w);

}

}

int find(int r,int w){

cnt=0;

for(int i=r;i;i-=lb(i))

tmp[++cnt]=root[i];

return query(1,k,w);

}

int main(){

n=read();k=read();

for(int i=1;i<=k;i++) root[i]=++tot;

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

node[i].x=read();node[i].y=read();node[i].z=read();node[i].id=i;

}

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

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

int j=i;

while(node[j].z==node[j+1].z) add(node[j].x,node[j].y,1),j++;

add(node[j].x,node[j].y,1);

j=i;

while(node[j].z==node[j+1].z) f[node[j].id]=find(node[j].x,node[j].y),j++;

f[node[j].id]=find(node[j].x,node[j].y);

i=j+1;

}

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

ans[f[i]]++;

}

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

printf("%d\n",ans[i]);

}

return 0;

}

* [[CQOI2011]动态逆序对](https://www.luogu.com.cn/problem/P3157)

我们考虑第 i*i* 个数插入时贡献的逆序对个数，就是位置在 i*i* 前面且比 a\_i*ai*​ 大的数的个数。

然后删除第 i*i* 个数的时候减少的逆序对个数就是在 i*i* 前面且大于 a\_i*ai*​ 的数与在 i*i* 后面且小于 a\_i*ai*​ 的数的个数之和。

我们可以将位置作为一个维度，值作为一个维度，这样这道题就变成了一个二维问题。

直接用树状数组套权值树解决。

#include <bits/stdc++.h>

#define int long long

using namespace std;

inline int read(){

register int x=0;

register bool f=0;

register char c=getchar();

while(c<'0'||c>'9'){

if(c=='-') f=1;

c=getchar();

}

while(c>='0'&&c<='9'){

x=(x<<3)+(x<<1)+c-48;

c=getchar();

}

return f?-x:x;

}

const int maxn=100005;

const int inf=2147483647;

struct seg{

int v,ls,rs;

}t[maxn\*85];

int rt[maxn],n,m,tot,cnt,num,ans,pos[maxn];

int a[maxn];

int lb(int x){

return x&(-x);

}

void pushup(int o){

t[o].v=t[t[o].ls].v+t[t[o].rs].v;

}

void change(int &o,int l,int r,int k,int v){

if(!o) o=++tot;

if(l==r){

t[o].v+=v;

return;

}

int mid=l+r>>1;

if(k<=mid) change(t[o].ls,l,mid,k,v);

else change(t[o].rs,mid+1,r,k,v);

pushup(o);

}

void add(int o,int v){

for(int i=o;i<=n;i+=lb(i)) change(rt[i],1,n,a[o],v);

}

int query(int o,int l,int r,int ql,int qr){

if(ql>qr) return 0;

if(ql<=l&&r<=qr){

return t[o].v;

}

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

if(ql<=mid) sum+=query(t[o].ls,l,mid,ql,qr);

if(qr>mid) sum+=query(t[o].rs,mid+1,r,ql,qr);

return sum;

}

int find(int l,int r,int ql,int qr){

int sum=0;

for(int i=r;i;i-=lb(i)){

sum+=query(rt[i],1,n,ql,qr);

}

for(int i=l-1;i;i-=lb(i)){

sum-=query(rt[i],1,n,ql,qr);

}

return sum;

}

signed main(){

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

tot=cnt=num=0;

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

a[i]=read();add(i,1);

ans+=find(1,i-1,a[i]+1,n);

pos[a[i]]=i;

}

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

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

int q=read();

ans-=find(1,pos[q]-1,q+1,n);

ans-=find(pos[q]+1,n,1,q-1);

add(pos[q],-1);

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

}

return 0;

}

刚才的题都是比较纯粹的题，但在实际的应用中还有很多的题目，是用数据结构来配合某种算法。

比如下面的这道题：

* [[HEOI2016/TJOI2016]序列](https://www.luogu.com.cn/problem/P4093)

这是一个动态规划题，是求一个最长的，且在所有变化中都不下降的子序列。

我们不妨用 a\_i,f\_i,g\_i,dp\_i*ai*​,*fi*​,*gi*​,*dpi*​ 分别表示其原本的值，变化中最大去到的值，变化中最小取到的值和以这一位结尾的最不下降子序列长度。

大致的做法与经典的 LIS*LIS* 一样，只不过条件变化成为我们要找到一个 j*j* 满足以下两个条件：

f(x)=\left\{ \begin{aligned} f\_j \le a\_i \\ a\_j\le g\_i \end{aligned} \right.*f*(*x*)={*fj*​≤*ai*​*aj*​≤*gi*​​

且在所有满足条件的 j*j* 中 dp\_j*dpj*​ 最大。

那么我们可以考虑用树状数组套权值树来维护：

树状数组维护 a*a* 然后查询时查的范围是 [1,g\_i][1,*gi*​]；

权值树维护 f*f* 然后查询时查的范围是 [1,a\_i][1,*ai*​]；

权值树中记录的量就是最大的 dp*dp* 值。

#include <bits/stdc++.h>

using namespace std;

inline int read(){

register int x=0;

register bool f=0;

register char c=getchar();

while(c<'0'||c>'9'){

if(c=='-') f=1;

c=getchar();

}

while(c>='0'&&c<='9'){

x=(x<<3)+(x<<1)+c-48;

c=getchar();

}

return f?-x:x;

}

void write(int x){

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

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

putchar('0'+x%10);

}

const int maxn=100005;

int len=100000;

const int inf=2147483647;

struct seg{

int v,ls,rs;

}t[maxn\*100];

int n,m,tot,ans,rt[maxn],f[maxn],g[maxn],a[maxn];

int lb(int x){

return x&(-x);

}

void pushup(int o){

t[o].v=max(t[t[o].ls].v,t[t[o].rs].v);

}

void change(int &o,int l,int r,int q,int v){

if(!o) o=++tot;

if(l==r){

t[o].v=max(t[o].v,v);

return ;

}

int mid=l+r>>1;

if(q<=mid) change(t[o].ls,l,mid,q,v);

else change(t[o].rs,mid+1,r,q,v);

pushup(o);

}

void add(int o,int v){

for(int i=a[o];i<=n;i+=lb(i)) change(rt[i],1,len,f[o],v);

}

int query(int o,int l,int r,int q){

if(l==r) {

return t[o].v;

}

int mid=l+r>>1;

if(q<=mid) return query(t[o].ls,l,mid,q);

else return max(t[t[o].ls].v,query(t[o].rs,mid+1,r,q));

}

int find(int o,int v){

int res=0;

for(int i=o;i;i-=lb(i)) res=max(res,query(rt[i],1,len,v));

return res;

}

signed main(){

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

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

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

int x=read(),v=read();

f[x]=max(f[x],v);

g[x]=min(g[x],v);

}

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

int res=find(g[i],a[i])+1;

ans=max(res,ans);

add(i,res);

}

write(ans);

return 0;

}

最后，我们再来看一道有一点难（du）度（liu）的题目。

* [[Ynoi2016]镜中的昆虫](https://www.luogu.com.cn/problem/P4690)

首先，区间染色，想到珂朵莉树。但显然这题数据不可能随机。

接着我们来思考：在无修改的情况下（也就是[HH的项链](https://www.luogu.com.cn/problem/P1972)）我们是使用一个线段树或树状数组维护前驱的。

那么这道题的解法就是综合这两种思想：

我们考虑一个颜色，显然，我们只需要将这种颜色记录一次。

在静态中，我们是通过维护前驱完成的，那么在动态中，我们同样维护前驱。

记一个点左侧最靠右且与之颜色相同的点为 pre*pre*，若无则为 00，

那么我们只需要统计 i\in[l,r]*i*∈[*l*,*r*] 且 pre\_i\in[0,l)*prei*​∈[0,*l*) 的点，因为这些点是这个区间中这个颜色的点中最靠左的那一个。

(这是个很显然的事实，因为如果这个点的前驱满足 l\le pre\_i*l*≤*prei*​ 那么 pre\_i*prei*​ 这一个点也属于 [l,r][*l*,*r*] 且与之同色，并且更靠左)

将点在序列中的位置作为一维，前驱作为另一维，用树状数组套权值树维护。

在修改的时候，我们用珂朵莉树的思想，用 set*set* 将同色的点结合成一个段，显然，一个段内的点出了最左侧的之外前驱均为 i-1*i*−1。

每次修改的时候，我们将左右个端点所在的段拆分，然后暴力取出中间的段修改并将之合并为一个段。

采用摊还法，set*set* 内初始有 n*n* 个段，修改时产生的段个数可以看成 \operatorname{O}(m)O(*m*) （左右端点断开后新增的段），

那么修改次数的复杂度可以看成均摊的 \operatorname{O}(n+m)=\operatorname{O}(n)O(*n*+*m*)=O(*n*) （n,m*n*,*m* 同阶），单次修改复杂度 \operatorname{O}(\log^2 n)O(log2*n*)，总复杂度 \operatorname{O}(n\log^2 n)O(*n*log2*n*)。

另外，我们在维护全局 set*set* 的同时也对每个颜色各开一个 set*set* 一起操作，这样会更方便。

#include <bits/stdc++.h>

using namespace std;

inline int read(){

register int x=0;

register bool f=0;

register char c=getchar();

while(c<'0'||c>'9'){

if(c=='-') f=1;

c=getchar();

}

while(c>='0'&&c<='9'){

x=(x<<3)+(x<<1)+c-48;

c=getchar();

}

return f?-x:x;

}

void write(int x){

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

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

putchar('0'+x%10);

}

const int maxn=400005;

int len=0;

const int inf=2147483647;

struct node{

int l,r,x;

//node():l(0),r(0),x(0){}

friend bool operator <(node a,node b){

return a.l<b.l;

}

}tp;

struct seg{

int v,ls,rs;

}t[maxn\*50];

int rt[maxn],n,m,tot,tem[maxn],tmp[maxn],cnt,num;

int lsh[maxn<<1],a[maxn],pre[maxn];

struct cz{

int a,b,c,d;

}q[maxn];

set<node> s[maxn],al;

set<int> now;

set<node>:: iterator it;

set<int>:: iterator \_it;

int lb(int x){

return x&(-x);

}

void pushup(int o){

t[o].v=t[t[o].ls].v+t[t[o].rs].v;

}

void change(int &o,int l,int r,int k,int v){

if(!o) o=++tot;

if(l==r){

t[o].v+=v;

return ;

}

int mid=l+r>>1;

if(k<=mid) change(t[o].ls,l,mid,k,v);

else change(t[o].rs,mid+1,r,k,v);

pushup(o);

}

void add(int o,int v){

for(int i=o;i<=n;i+=lb(i)) change(rt[i],0,n,pre[o],v);

}

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

if(l==r) {

return 0;

}

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

if(k<=mid){

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

for(int i=1;i<=num;i++) tmp[i]=t[tmp[i]].ls;

return query(l,mid,k);

}

else{

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

for(int i=1;i<=num;i++) sum-=t[t[tmp[i]].ls].v,tmp[i]=t[tmp[i]].rs;

return sum+query(mid+1,r,k);

}

}

int find(int l,int r,int k){

cnt=num=0;

for(int i=r;i;i-=lb(i)){

tem[++cnt]=rt[i];

}

for(int i=l-1;i;i-=lb(i)){

tmp[++num]=rt[i];

}

return query(0,n,k);

}

void split(int x){

tp=(node){x,0,0};

it=al.upper\_bound(tp);--it;

if(it->l==x) return;

tp=\*it;

al.erase(tp);s[tp.x].erase(tp);

node tp1=(node){tp.l,x-1,tp.x};

node tp2=(node){x,tp.r,tp.x};

al.insert(tp1);al.insert(tp2);

s[tp.x].insert(tp1);

s[tp.x].insert(tp2);

}

void update(int l,int r,int x){

if(l!=1) split(l);

if(r+1<=n) split(r+1);

now.insert(x);

tp=(node){l,0,0};

it=al.lower\_bound(tp);

while(it->l!=r+1){

tp=\*it;now.insert(tp.x);

if(tp.l>l&&pre[tp.l]!=tp.l-1){

add(tp.l,-1);

pre[tp.l]=tp.l-1;

add(tp.l,1);

}

al.erase(tp);s[tp.x].erase(tp);

tp=(node){l,0,0};

it=al.lower\_bound(tp);

if(it==al.end()) break;

}

tp=(node){l,0,0};

it=s[x].lower\_bound(tp);--it;

add(l,-1);pre[l]=it->r;add(l,1);

tp=(node){l,r,x};

al.insert(tp);s[x].insert(tp);

for(\_it=now.begin();\_it!=now.end();++\_it){

tp=(node){r,0,0};

it=s[\*\_it].upper\_bound(tp);

if(it!=s[\*\_it].end()){

l=it->l;

tp=(node){l,0,0};

it=s[\*\_it].lower\_bound(tp);--it;

add(l,-1);pre[l]=it->r;add(l,1);

}

}

now.clear();

}

signed main(){

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

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

a[i]=read();lsh[++len]=a[i];

}

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

q[i].a=read();q[i].b=read();q[i].c=read();

if(q[i].a==1){

q[i].d=read();

lsh[++len]=q[i].d;

}

}

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

len=unique(lsh+1,lsh+len+1)-lsh-1;

tp=(node){0,0,0};

for(int i=1;i<=len;i++)s[i].insert(tp);

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

a[i]=lower\_bound(lsh+1,lsh+1+len,a[i])-lsh;

it=s[a[i]].end();it--;pre[i]=it->l;

add(i,1);

tp=(node){i,i,a[i]};

al.insert(tp);s[a[i]].insert(tp);

}

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

if(q[i].a==1){

q[i].d=lower\_bound(lsh+1,lsh+len+1,q[i].d)-lsh;

update(q[i].b,q[i].c,q[i].d);

}

else{

write(find(q[i].b,q[i].c,q[i].b));

puts("");

}

}

return 0;

}

**最后附带几个小技巧**

由于树状数组空间较大，而且实际上在运行过程中，有的节点已经被删除却任然占用空间，

这就加重了空间的负担。

事实上，一个树状数组真正在使用的空间应该只有 n \log n \log len*n*log*n*log*len* 个，而我们需要开出 (n+m)\log n \log len(*n*+*m*)log*n*log*len* 的空间。碰到空间常数不够的情况就很麻烦。

在此给出一种小方法：

int st[maxn],top,cnt；

int nnd(){

return top?st[top--]:++cnt;

}

int del(int o){

if(t[o].v==0&&t[o].rs==0&&t[o].ls==0){

st[++top]=o;

return 0;

}

return o;

}

用一个队列或是栈来分配新空间并维护已经无效的空间，将其回收。在一些卡空间常数的题目上会发挥作用（例如[这题](https://www.luogu.com.cn/problem/CF1093E)）。

另外，当我们需要在一段区间内插入一个数时，树状数组套权值树就会非常的麻烦（比如[[ZJOI2013]K大数查询](https://www.luogu.com.cn/problem/P3332)与[[HNOI2015]接水果](https://www.luogu.com.cn/problem/P3242)）。

此时我们可以内外反向，用权值树来套普通线段树。

另外，为了节省空间，内层的树在区间加时不应下放，而应该使用标记永久化。

//普通树

inline void pushup(int o){

t[o].sum=t[t[o].ls].sum+t[t[o].rs].sum;

}

inline void change(int &o,int l,int r,int ql,int qr,int k){

if(!o) o=++num;

if(ql<=l && qr>=r){

t[o].tag+=k;

t[o].sum+=k\*(r-l+1);

return ;

}

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

if(ql<=mid) change(t[o].ls,l,mid,ql,qr,k);

if(qr>mid) change(t[o].rs,mid+1,r,ql,qr,k);

pushup(o);

}

inline int query(int o,int l,int r,int k){

if(l==r)return t[o].sum;

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

if(k<=mid) return t[o].tag+query(t[o].ls,l,mid,k);

if(k>mid) return t[o].tag+query(t[o].rs,mid+1,r,k);

}

//权值树

inline void change(int o,int l,int r,int k,int v,int ql,int qr){

change(rt[o],1,n,ql,qr,k);

if(l==r) return;

int mid=l+r>>1;

if(v<=mid) change(o<<1,l,mid,k,v,ql,qr);

else change(o<<1|1,mid+1,r,k,v,ql,qr);

}

inline int query(int o,int l,int r,int k,int v){

if(l==r) return l;

int mid=l+r>>1,sum=query(rt[o<<1],1,n,k);

if(v<=sum) return query(o<<1,l,mid,k,v);

else return query(o<<1|1,mid+1,r,k,v-sum);

}