https://www.luogu.com.cn/blog/bfqaq/qian-tan-quan-zhi-xian-duan-shu

浅谈权值线段树

**posted on 2020-01-10 22:49:56 | under**[**未分类**](https://www.luogu.com.cn/blog/bfqaq/#type=%E6%9C%AA%E5%88%86%E7%B1%BB)**|** 11

**浅谈权值线段树**

（tips：如果未学习过线段树不建议阅读）

众所周知，在信息学中，有一种神奇的数据结构叫做线段树，它可以解决许许多多的区间动态查询问题。

的确，线段树是一类神奇的数据结构。但是，如果你认为它只能解决一些区间动态查询的问题，那么就太低估它了。

（感谢[犇犇犇犇](https://www.luogu.com.cn/user/35998)提供的修改建议）

**0 目录**

* 1. 什么是权值线段树
  2. 权值树的重要性质
  3. 权值树的辅助操作
  4. 权值树的习题精选

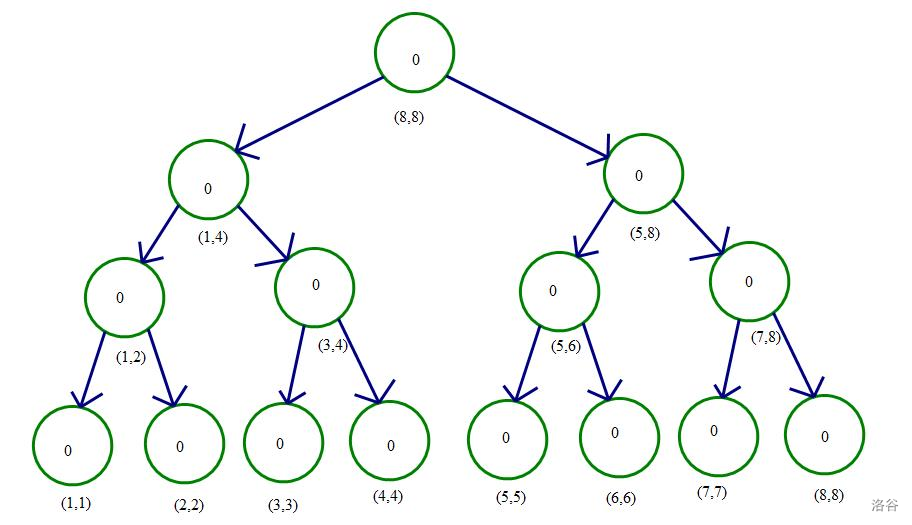
**1 什么是权值线段树**

顾名思义，权值线段树，就是对权值作为维护对像而开的线段树，即每个点上存的是区间内的对应数字的某种值（最常见的是出现次数）。

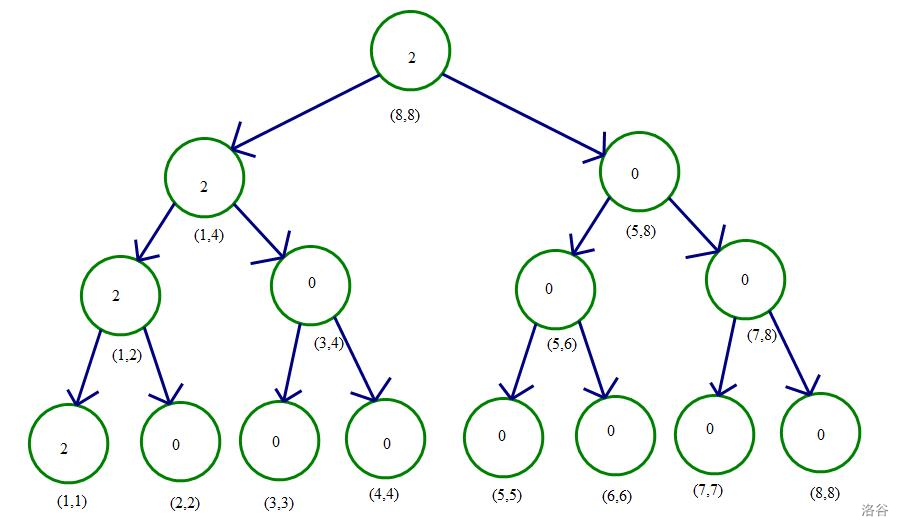
举个最简单的例子，权值线段树可以用于维护一个数在一个序列中出现的次数。

比如现在有一个数组1 ,1, 2, 2, 2, 3, 4, 5 ,6,7,81,1,2,2,2,3,4,5,6,7,8

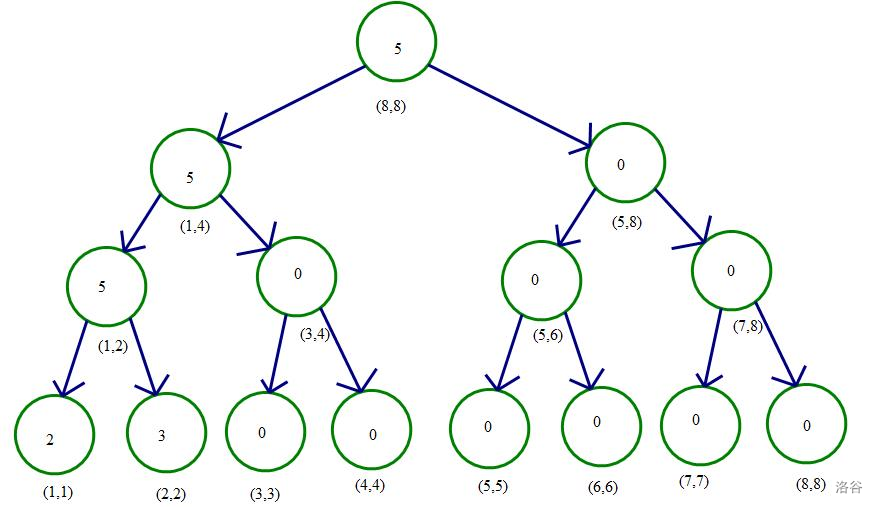
**对于每个节点，初始时个数为0**



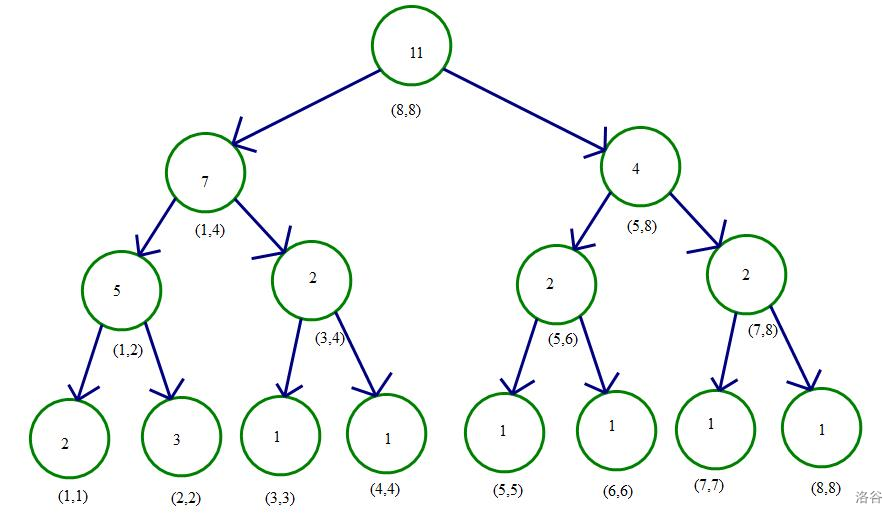
**插入1**



**插入2**



**最后**



（以上也就是一个建树的过程）

由于权值树也是线段树，所以权值树的操作也都是跟线段树一样的，时间复杂度是 \log lenlog*len* 每次操作。

（其中，len*len* 代表最大的权值，因为我们是对权值开的线段树）

听到这儿，也许有人会想到平衡树。的确，权值树与平衡树的用途基本是相同的，同样的复杂度，同样支持动态。

但是，权值树代码量小，易于调整，优势也就由此显现出来了。

由于没有专门的模版题，我们就借用平衡树的模版来解决。

[【模版】普通平衡树](https://www.luogu.com.cn/problem/P3369)

code:（由于我们只需要关心每个数之间的相对关系，因此需要用到离散化操作。 ）

#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=111111;

struct seg{

int v;

}t[maxn<<3];

void pushup(int o){

t[o].v=t[o<<1].v+t[o<<1|1].v;

}

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

if(l==r){

t[o].v+=v;

return ;

}

int mid=l+r>>1;

if(q<=mid) change(o<<1,l,mid,q,v);

else change(o<<1|1,mid+1,r,q,v);

pushup(o);

}

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

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

return t[o].v;

}

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

if(ql<=mid) ans+=query\_rnk(o<<1,l,mid,ql,qr);

if(qr>mid) ans+=query\_rnk(o<<1|1,mid+1,r,ql,qr);

return ans;

}

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

if(l==r){

return l;

}

int mid=l+r>>1;

if(t[o<<1].v>=q) return query\_num(o<<1,l,mid,q);

else return query\_num(o<<1|1,mid+1,r,q-t[o<<1].v);

}

int lsh[maxn<<2],tot,n;

struct \_node{

int opt,val;

}node[maxn<<2];

int main(){

n=read();

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

node[i].opt=read();

node[i].val=read();

if(node[i].opt==4) continue;

lsh[++tot]=node[i].val;

}

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

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

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

if(node[i].opt!=4) node[i].val=lower\_bound(lsh+1,lsh+tot+1,node[i].val)-lsh;

if(node[i].opt==1) change(1,1,tot,node[i].val,1);

if(node[i].opt==2) change(1,1,tot,node[i].val,-1);

if(node[i].opt==3){

if(node[i].val==1){

puts("1");

continue;

}

printf("%d\n",query\_rnk(1,1,tot,1,node[i].val-1)+1);

}

if(node[i].opt==4){

printf("%d\n",lsh[query\_num(1,1,tot,node[i].val)]);

}

if(node[i].opt==5){

int rk=query\_rnk(1,1,tot,1,node[i].val-1);

printf("%d\n",lsh[query\_num(1,1,tot,rk)]);

}

if(node[i].opt==6){

int rk=query\_rnk(1,1,tot,1,node[i].val)+1;

printf("%d\n",lsh[query\_num(1,1,tot,rk)]);

}

}

return 0;

}

其中，change*change* 是单点修改，query\\_rnk*query*\_*rnk* 就是一个正常的区间求和。

而 query\\_num*query*\_*num* 是权值树特有的操作，也就是查询第 q*q* 大。

其操作原则就是：如果左子树有大于 q*q* 个数个从左子树查询，否则查询右子树并减去左子树的个数之和。

其原理就是二分，如果小于等于 mid*mid* 的数超过 q*q* 个，那么一定在左子树中，反之一定在右子树中。

对比两种算法：



上面的是平衡树(使用Splay),下面的是权值树。

显然，不管是时间，空间还是码量，都是权值树更优。

不过，权值树的空间优势仅在离线或是值域较小的情况下有优势。当值域较大时，由于不能动态开点，空间复杂度需要增加到 \operatorname O(V)O(*V*)（设 V*V* 为值域）。

当然，如果使用压缩 trie 等方式，则可以将空间缩小为 \operatorname O(n)O(*n*)，但在本文不作讨论。

**2 权值树的重要性质**

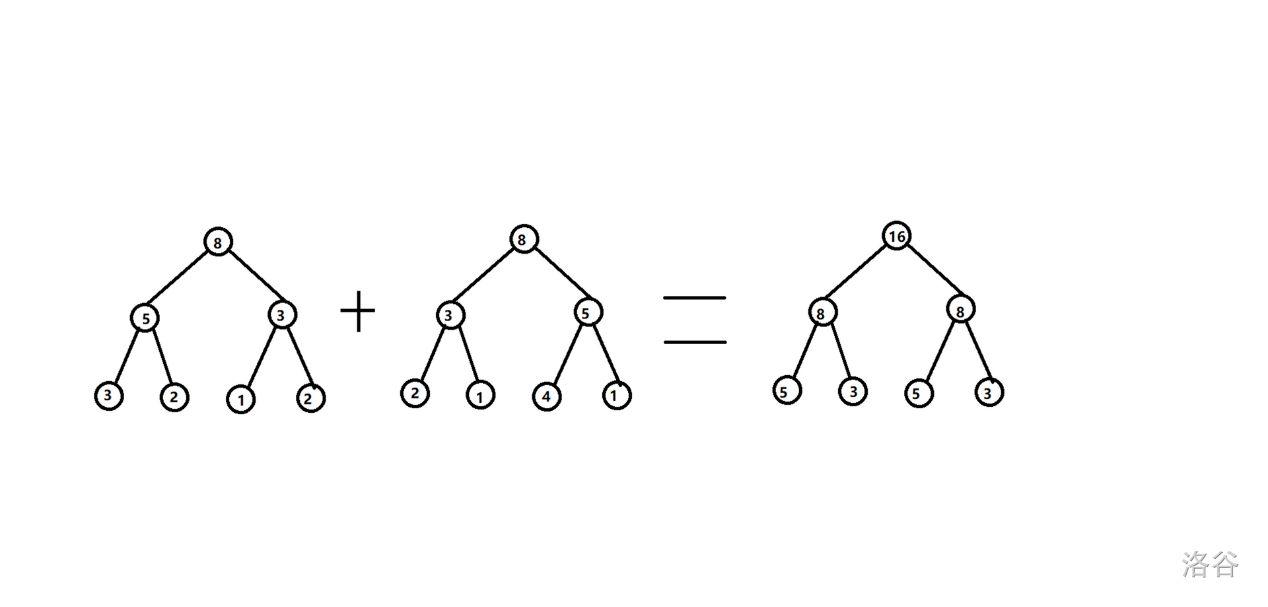
我们知道，对于一棵线段树而言，如果它的总长度不变，那么它的形态是不会改变的。

也就是说，在 len*len* 不变的情况下，权值树的形态是不会改变的。

这样一来，我们就可以对权值树进行加减法操作。

对于权值树 A,B*A*,*B*，若 A,B*A*,*B* 形态相同，则我们可以直接合并这两棵权值树，合并的方式就是对应节点相加。

显然，加出来的树依然是一棵权值线段树（如图）。



同样的，权值树亦可以相减，减出来的树依然是权值树。

这个性质非常的重要，接下的的前缀和/树状数组就要基于这一性质。

**3 权值树的辅助操作**

* 前缀和

由于权值线段树的可合并性，因此，我们可以对权值线段树进行静态前缀和。

这样一来，我们就可以从全局问题升级到维护区间。

[P3834 【模板】可持久化线段树 1（主席树）](https://www.luogu.com.cn/problem/P3834)

这个题是一个典型的静态问题。我们可以直接使用权值树的前缀和解决。

对于每个点，建立一棵权值树，第 k*k* 棵树维护 [1,k][1,*k*] 区间的出现次数。

然后查询操作就是这样的：

我们真正的需要的树为 [l,r][*l*,*r*] 的树（也就是要知道 [l,r][*l*,*r*] 区间内各数的出现次数 ），

而要得到这棵树，我们只需要将 [1,r][1,*r*] 的树与 [1,l-1][1,*l*−1] 的树相减即可。

（因为第 r*r* 棵树维护 [1,r][1,*r*] 区间的出现次数，第 l-1*l*−1 棵树维护 [1,l-1][1,*l*−1] 中的出现次数，两者相减就是我们想要的区间了）

如最上面那张图，我们要求第 66 小的数。

我们先查根节点，为 1111，表示一共 1111 个数。

左孩子权值为 77，表示 [1,4][1,4] 一共 77 个数，而我们需要第 66 小的数，7>67>6，所以我们要找的数一定在左孩子上。故递归左子树。

这时，我们发现左孩子为 55，所以前 55 小的数都在区间 [1,2][1,2] 上，所以我们要找的数在右孩子上。

因为左孩子已经有 55 个数了，我们要找右子树上的 6-5=16−5=1 小的数。故递归右子树。

同理递归，一直找到叶节点。我们发现找到了33。

然后再同区间第 k*k* 大的操作即可。

int query(int o1,int o2,int l,int r,int q){

if(l==r){

return l;

}

int mid=l+r>>1,tmp=t[t[o2].ls].v-t[t[o1].ls].v;

if(tmp>=q) return query(t[o1].ls,t[o2].ls,l,mid,q);

else return query(t[o1].rs,t[o2].rs,mid+1,r,q-tmp);

}

但是，这个时候，一个新的问题出现了。

每一个权值树的空间都是 {\displaystyle Θ(n)}Θ(*n*) 级别的，那么总复杂度就是 Θ(n^2)Θ(*n*2)。显然，这是无法接受的。

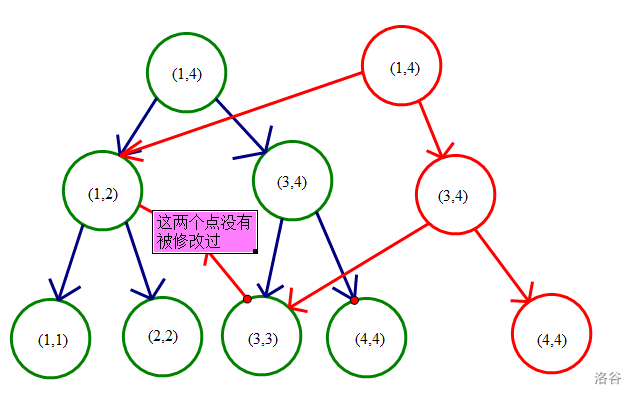
那么我们怎么解决呢？

* 动态开点

所谓动态开点，就是动态的分配内存。

显然，当我们修改一个权值时，只会对它和它的直属父辈节点产生影响（也就是一条链）。

那么，这个时候，我们只需要对这一条链进行分配内存，而剩下的节点可以借用原节点（如图所示）。



代码如下：

void pushup(int o){

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

}

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

if(!o) o=++cnt;

if(l==r){

t[o].v+=v;

return ;

}

int mid=l+r>>1;

if(q<=mid){

t[o].rs=t[lsto].rs;

t[o].ls=++cnt;

t[t[o].ls]=t[t[lsto].ls];

change(t[lsto].ls,t[o].ls,l,mid,q,v);

}

else{

t[o].ls=t[lsto].ls;

t[o].rs=++cnt;

t[t[o].rs]=t[t[lsto].rs];

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

}

pushup(o);

}

这样做之后，空间就会被降到 Θ(n logn)Θ(*nlogn*) 级别，就可以接受了。

于是这题的代码如下：（一般对于 100000100000 的数据只需要开 32\times maxn32×*maxn* 左右的空间即可，但不放心可以开的更大）

#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=111111;

struct seg{

int v,ls,rs;

}t[maxn<<8];

int rt[maxn<<2],cnt;

void pushup(int o){

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

}

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

if(!o) o=++cnt;

if(l==r){

t[o].v+=v;

return ;

}

int mid=l+r>>1;

if(q<=mid){

t[o].rs=t[lsto].rs;

t[o].ls=++cnt;

t[t[o].ls]=t[t[lsto].ls];

change(t[lsto].ls,t[o].ls,l,mid,q,v);

}

else{

t[o].ls=t[lsto].ls;

t[o].rs=++cnt;

t[t[o].rs]=t[t[lsto].rs];

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

}

pushup(o);

}

int query(int o1,int o2,int l,int r,int q){

if(l==r){

return l;

}

int mid=l+r>>1,tmp=t[t[o2].ls].v-t[t[o1].ls].v;

if(tmp>=q) return query(t[o1].ls,t[o2].ls,l,mid,q);

else return query(t[o1].rs,t[o2].rs,mid+1,r,q-tmp);

}

int lsh[maxn<<2],tot,n,m,node[maxn<<2];

int main(){

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

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

node[i]=read();

lsh[i]=node[i];

}

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

tot=unique(lsh+1,lsh+1+n)-lsh-1;

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

node[i]=lower\_bound(lsh+1,lsh+tot+1,node[i])-lsh;

change(rt[i-1],rt[i],1,tot,node[i],1);

}

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

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

printf("%d\n",lsh[query(rt[l-1],rt[r],1,tot,q)]);

}

return 0;

}

不论是修改还是操作，时间复杂度都同线段树。

由于不需要建树，而每一次修改只需改一条链，所以每次消耗空间复杂度为 \log lenlog*len*，总空间复杂度 n\log len*n*log*len*。

* 树状数组/线段树

由于权值树可以加减，那么我们也可以用树状数组或是线段树来维护。

在这里我就以树状数组为例：

我们先建立一个树状数组，以维护每一个节点的数据；

然后再在每个点建立权值树（配合动态开点）

* [P3380 【模板】二逼平衡树（树套树）](https://www.luogu.com.cn/problem/P3380)

在修改的时候从树状数组找到需要修改的节点：

void add(int o,int v){

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

}

然后在对应的权值树上进行修改：

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);

}

查询的时候也是一样，在此以查询第 k*k* 小为例子

先从树状数组预处理出需要查询的节点。

此处我用 tem*tem* 记录需要加的节点，用 tmp*tmp* 记录需要减去的节点。

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);

}

紧接着进行查询,具体做法与权值树相同：

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;

//所有节点全部进入左子树

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;

//所有节点全部进入右子树

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

}

}

剩余的操作大致做法相同。

code：

#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;

}

同样是对比（平衡树使用Treap）：



此处权值树的常数优势就更为明显了。但相应的，权值树的空间劣势在逐渐暴露。有兴趣深入学习权值线段树树套树的同学可以参考[这篇日报](https://www.luogu.com.cn/blog/bfqaq/qian-tan-shu-zhuang-shuo-zu-quan-zhi-shu)，在本文不作深入展开。

**4 权值树的习题精选**

（限于篇幅，此处习题不给出讲解）

**杂题**

* [P1600 天天爱跑步](https://www.luogu.com.cn/problem/P1600)
* [P1972 [SDOI2009]HH的项链](https://www.luogu.com.cn/problem/P1972)
* [P4113 [HEOI2012]采花](https://www.luogu.com.cn/problem/P4113)
* [P2468 [SDOI2010]粟粟的书架](https://www.luogu.com.cn/problem/P2468)
* [P3168 [CQOI2015]任务查询系统](https://www.luogu.com.cn/problem/P3168)
* [P3157 [CQOI2011]动态逆序对](https://www.luogu.com.cn/problem/P3157)
* [P3810 【模板】三维偏序（陌上花开）](https://www.luogu.com.cn/problem/P3810)

**扫描线**

* [P1856 [USACO5.5]矩形周长Picture](https://www.luogu.com.cn/problem/P1856)
* [P5490 【模板】扫描线](https://www.luogu.com.cn/problem/P5490)
* [P1904 天际线](https://www.luogu.com.cn/problem/P1904)

**二维偏序**

* [P2163 [SHOI2007]园丁的烦恼](https://www.luogu.com.cn/problem/P2163)
* [P5816 [CQOI2010]内部白点](https://www.luogu.com.cn/problem/P5816)
* [P3431 [POI2005]AUT-The Bus](https://www.luogu.com.cn/problem/P3431)
* [P1502 窗口的星星](https://www.luogu.com.cn/problem/P1502)
* [P4390 [BOI2007]Mokia 摩基亚](https://www.luogu.com.cn/problem/P4390)