**可持续化线段树（主席树）**

**什么是主席树**

可持久化数据结构(Persistent data structure)就是利用函数式编程的思想使其支持询问历史版本、同时充分利用它们之间的共同数据来减少时间和空间消耗。

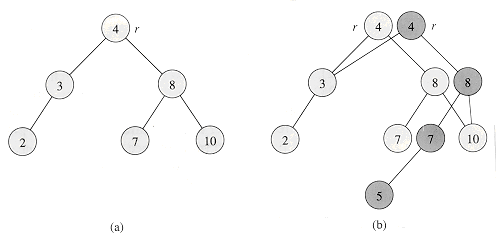
因此可持久化线段树也叫函数式线段树又叫主席树。

**可持久化数据结构**

在算法执行的过程中，会发现在更新一个动态集合时，需要维护其过去的版本。这样的集合称为是可持久的。

实现持久集合的一种方法时每当该集合被修改时，就将其整个的复制下来，但是这种方法会降低执行速度并占用过多的空间。

考虑一个持久集合S。

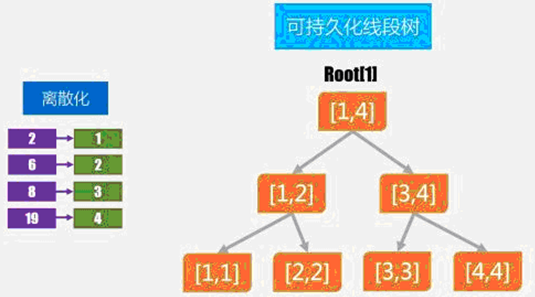


如图所示，对集合的每一个版本维护一个单独的根，在修改数据时，只复制树的一部分。

称之为可持久化数据结构。

**离散化**

但是，有一个很重要的问题！题目的空间限制肯定是有的，假设所输入的数的范围为int，你总不可能开一个大小为int的树吧？而且还要多棵线段树。此时此刻，我们就可以引入一个新知识了——离散化。看起来很高端，其实很简单，其实你脑补一下map（属于STL）就行了，或者回忆一下高中数学必修一集合那一章，有一个叫映射的东西，和离散化意思差不多（起码在这道题上的作用是一模一样的），所以不详细阐述，在源代码中会有小小的注释。

好了，目前有一个数列：{2,8,19,6}。假设我们已经离散化结束了，结果为2→1；6→2；8→3；19→4。那么以后我们进行数据的处理时，1就表示2了，2就表示6了，3就表示8了。。。是不是和映射一个意思？这样的好处在于，我们不需要依赖就弄个[1,2147483647]的线段树了，若题目规定n<=100000，则最大只需要一棵[1,100000]的线段树了。如下图（其实没有蛮多含义，真正的变化在后面）：

【若是建一棵[1,19]的线段树，你想想是多么浪费空间 = =。】

**历史版本的作用**

这么多棵线段树，我们也不可能建立多个结构体来保存。我们可以把所有线段树的节点全部放在tree结构体中，设当前有m个节点，每执行一次插入操作，新增了x个节点，则存放在tree中的第(m+1)个节点至第(m+x)个节点（当然也有别的编号方式）。同时，我们需要一个root数组，其中root[i]表示第i棵线段树的根节点的编号，是一棵值域树，表示区间[1,i]的所有值在值域树的分布。

这样我们就构建完了，来想想——为什么需要历史版本？回到我们一开始的问题，求区间第k大，假设当前询问为求[x,y]的第k大，则我们所需要用到的线段树为第x-1棵到第y棵。从根节点开始，我们将第y棵树和第x-1棵树一一对应的节点所维护的值进行相减，其所得到的数就是在所询问的[x,y]中，当前节点表示的子区间的那几个数值在整个区间中出现的次数，用t表示，即t=root[y].[1,mid]-root[x-1].[1,mid]。先判断t是否大于k，如果t大于k，那么说明在区间[x,y]内存在[1,mid]的数的个数大于k，也就是第k大的值在[1,mid]中，否则在[mid+1,r]中。（有点绕，慢慢看 →\_→）





**缩小空间**

其实必要的知识已经讲得差不多了，但是我们最后还要面临一个问题——加入一个数，就新建一棵线段树。我们假设有100000个数吧，且有100000次询问，试想这一大片庞大的线段树森林是要占用多大的内存？一定会MLE的（当然数据小就无所谓）。我们有什么办法缩小空间需求？我们注意到，每次我们加入一个被离散化后的数x，则从根结点开始向下更新，我们真正相对于前面一棵线段树的差异之处是很少的！设有一颗[1,4]的线段树，若当前插入值为3，则[1,4]的左儿子[1,2]没有丝毫改动！如果又新建一个，完全是浪费。这样子，我们就有一个方法缩小冗余的空间了——将没有区别的部分直接指回去！如图所示：



空间是不是小多了！是的。后面的线段树也以此类推。

**可持久化线段树**

令 T 表示一个结点，它的左儿子是 left(T)，右儿子是 right(T)。

若 T 的范围是 [L,R]，那么 left(T) 的范围是 [L,mid]，right(T) 的范围是 [mid+1,R]。

**单点更新**

我们要修改一个叶子结点的值，并且不能影响旧版本的结构。

在从根结点递归向下寻找目标结点时，将路径上经过的结点都复制一份。

找到目标结点后，我们新建一个新的叶子结点，使它的值为修改后的版本，并将它的地址返回。

对于一个非叶子结点，它至多只有一个子结点会被修改，那么我们对将要被修改的子结点调用修改函数，那么就得到了它修改后的儿子。

在每一步都向上返回当前结点的地址，使父结点能够接收到修改后的子结点。

在这个过程中，只有对新建的结点的操作，没有对旧版本的数据进行修改。

**区间查询**

从要查询的版本的根节点开始，像查询普通的线段树那样查询即可。

**延迟标记**

...

**区间第K小值问题**

有n个数，多次询问一个区间[L,R]中第k小的值是多少。

**查询[1,n]中的第K小值**

我们先对数据进行离散化，然后按值域建立线段树，线段树中维护某个值域中的元素个数。

在线段树的每个结点上用cnt记录这一个值域中的元素个数。

那么要寻找第K小值，从根结点开始处理，若左儿子中表示的元素个数大于等于K，那么我们递归的处理左儿子，寻找左儿子中第K小的数；

若左儿子中的元素个数小于K，那么第K小的数在右儿子中，我们寻找右儿子中第K-(左儿子中的元素数)小的数。

**查询区间[L,R]中的第K小值**

我们按照从1到n的顺序依次将数据插入可持久化的线段树中，将会得到n+1个版本的线段树（包括初始化的版本），将其编号为0~n。

可以发现所有版本的线段树都拥有相同的结构，它们同一个位置上的结点的含义都相同。

考虑第i个版本的线段树的结点P，P中储存的值表示[1,i]这个区间中，P结点的值域中所含的元素个数；

假设我们知道了[1,R]区间中P结点的值域中所含的元素个数，也知道[1,L-1]区间中P结点的值域中所包含的元素个数，显然用第一个个数减去第二个个数，就可以得到[L,R]区间中的元素个数。

因此我们对于一个查询[L,R]，同步考虑两个根root[L-1]与root[R]，用它们同一个位置的结点的差值就表示了区间[L,R]中的元素个数，利用这个性质，从两个根节点，向左右儿子中递归的查找第K小数即可。

**例题**

求区间第k大。【HDU 2665】

#include<cstdio>

#include<algorithm>

#define MAXN 100005

using namespace std;

int a[MAXN],b[MAXN],n,tot,root[MAXN],q,l,r,k,link[MAXN],t;

struct Node{

int ls,rs,size;

};

Node tree[MAXN\*20];

struct cmp{

bool operator () (int i,int j){

return (a[i]<a[j]);

}

};

cmp x;

void discretize(){

sort(link+1,link+n+1,x); // 以link作为中转站，对a进行排序?

for (int i=1;i<=n;i++) b[link[i]]=i;//b[x]表示第x个为第b[x]名

}

void insert(int &now,int l,int r,int x){

tree[++tot]=tree[now]; now=tot;

tree[now].size++;

if (l==r) return;

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

if (x<=mid) insert(tree[now].ls,l,mid,x);

else insert(tree[now].rs,mid+1,r,x);

}

int query(int nl,int nr,int l,int r,int k){//第k小

if (l==r) return l;

int size=tree[tree[nr].ls].size-tree[tree[nl].ls].size,mid=(l+r)>>1;

if (size>=k) return query(tree[nl].ls,tree[nr].ls,l,mid,k);

else return query(tree[nl].rs,tree[nr].rs,mid+1,r,k-size);

}

int main(){

freopen("HDU2665.in","r",stdin);

freopen("HDU2665.out","w",stdout);

scanf("%d",&t);

for (int j=1;j<=t;j++){

root[0]=0; tot=0;

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

for (int i=1;i<=n;i++) { scanf("%d",&a[i]); link[i]=i; }

discretize(); // 离散化

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

root[i]=root[i-1];

insert(root[i],1,n,b[i]);

}

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

scanf("%d %d %d",&l,&r,&k);

printf("%d\n",a[link[query(root[l-1],root[r],1,n,k)]]);

}

}

return 0;

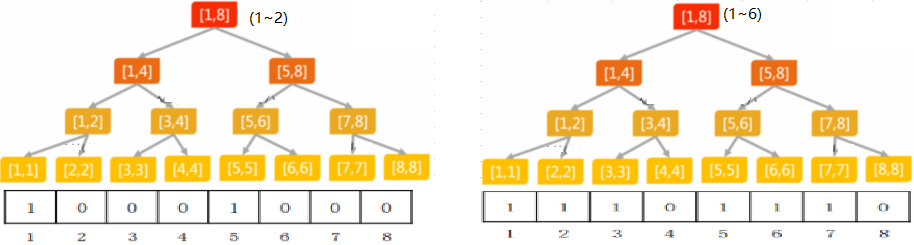
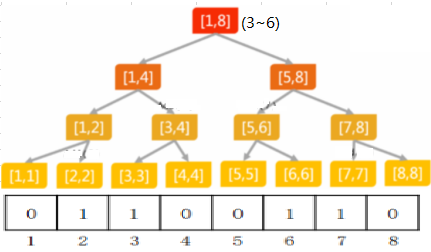
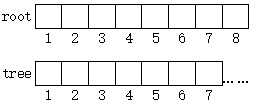
}

/\*

1

8 0

134 23 98 178 289 83 918 100\*/



可持续化线段树

算法：前缀和（区间、**可持续化**）+二分（单调性）。

数据结构：可持续化线段树（值树）

空间：nlogn（每个数，需要修改logn个节点，需要root[]：每棵树根的地址，tree[]：树的节点

作用：区别第几小（大）。

**常数优化的技巧**

一种在常数上减小内存消耗的方法：

插入值时候先不要一次新建到底，能留住就留住，等到需要访问子节点时候再建下去。

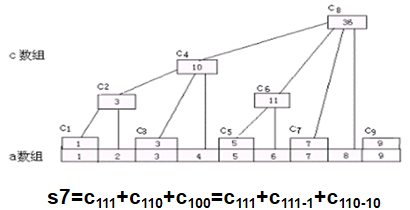
这样理论内存复杂度依然是O(Nlg^2N)，但因为实际上很多结点在查询时候根本没用到，所以内存能少用一些。

**动态第K小值**

每一棵线段树是维护每一个序列前缀的值在任意区间的个数，如果还是按照静态的来做的话，那么每一次修改都要遍历O(n)棵树，时间就是O(2\*M\*nlogn)->TLE。

考虑到前缀和，我们通过树状数组来优化，即树状数组套主席树，每个节点都对应一棵主席树，那么修改操作就只要修改logn棵树，O(nlognlogn+Mlognlogn)时间是可以的，但是直接建树要nlogn\*logn（10^7）会MLE。

我们发现对于静态的建树我们只要nlogn个节点就可以了，而且对于修改操作，只是修改M次，每次改变俩个值（减去原先的，加上现在的）也就是说如果把所有初值都插入到树状数组里是不值得的，所以我们分两部分来做，所有初值按照静态来建，内存O(nlogn)，而修改部分保存在树状数组中，每次修改logn棵树，每次插入增加logn个节点O(M\*logn\*logn+nlogn)。



# **区间动态第k大模板(树状数组套主席树)**

# **原理：**

<https://blog.csdn.net/g21wcr/article/details/86652152>

<https://blog.csdn.net/u014664226/article/details/47839973>

# **板子一：**

时空复杂度：O((n+q)\*logn\*logn)

[洛谷 P2617](https://www.luogu.org/problem/P2617)

#include<bits/stdc++.h>

using namespace std;

const int maxn=1e5+5;

struct node

{

int lc,rc,siz;

}tree[maxn\*400\*2];*//节点*

struct qury

{

int a,b,c;

}q[maxn]; *//离线操作*

char op[10];*//指令*

int n,m,ansx,ansy,tot,len;*//n个数 m个查询 tot个节点 len为离散化后的长度*

*//ansx ansy 用于记录 查询操作 所需要的节点个数*

int root[maxn],a[maxn],b[maxn<<1],qx[maxn],qy[maxn];*//记录根节点编号 原序列 离散化后的序列*

*// qx qy 用于记录 查询操作 所需要的节点*

inline int lowbit(int x)

{

return x&(-x);

}

inline void update(int &root,int pre,int l,int r,int val,int num)

{

root=++tot;

tree[root]=tree[pre];

tree[root].siz+=num;

if(l==r)

return;

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

if(val<=mid)

update(tree[root].lc,tree[pre].lc,l,mid,val,num);

else

update(tree[root].rc,tree[pre].rc,mid+1,r,val,num);

}

inline void add(int pos,int v)

{

int val=lower\_bound(b+1,b+len+1,a[pos])-b;

for(int i=pos;i<=n;i+=lowbit(i))

update(root[i],root[i],1,len,val,v);

}

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

{

if(l==r) return l;

int sum=0;

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

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

sum-=tree[tree[qx[i]].lc].siz;

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

sum+=tree[tree[qy[i]].lc].siz;

if(k<=sum){

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

qx[i]=tree[qx[i]].lc;

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

qy[i]=tree[qy[i]].lc;

return query(l,mid,k);

}

else{

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

qx[i]=tree[qx[i]].rc;

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

qy[i]=tree[qy[i]].rc;

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

}

}

inline void prework(){*//初始化操作 包括离散化等*

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

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

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

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

}

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

scanf("%s",op);

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

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

scanf("%d",&q[i].c);

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

q[i].c=-1,b[++len]=q[i].b;

}

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

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

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

add(i,1);

}

inline void mainwork(){*//查询操作*

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

if(q[i].c>=0){

ansx=0,ansy=0;

for(int j=q[i].b;j;j-=lowbit(j))

qy[++ansy]=root[j];

for(int j=q[i].a-1;j;j-=lowbit(j))

qx[++ansx]=root[j];

printf("%d\n",b[query(1,len,q[i].c)]);

}

else{

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

a[q[i].a]=q[i].b;

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

}

}

}

int main(){

prework();

mainwork();

return 0;

}

# **板子二：**

时空复杂度：O(n\*lgn+q\*logn\*lgn) 因为先建立了一棵静态主席树。

[ZOJ 2112](http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=2112)

#include<bits/stdc++.h>

using namespace std;

const int maxn=6e4+5;

const int M=2400005;

struct node{

int lc,rc,siz;

}tree[M];*//节点*

struct qury

{

int a,b,c;

}q[10010]; *//离线操作*

char op[10];*//指令*

int n,m,ansx,ansy,tot,len;*//n个数 m个查询 tot个节点 len为离散化后的长度*

*//ansx ansy 用于记录 查询操作 所需要的节点个数*

int srt[maxn],rt[maxn],a[maxn],b[maxn],qx[maxn],qy[maxn];*//记录静态主席树根节点编号 新增节点编号 原序列 离散化后的序列*

*// qx qy 用于记录 查询操作 所需要的节点*

inline int lowbit(int x)

{

return x&(-x);

}

inline void update(int &root,int pre,int l,int r,int val,int num)

{

root=++tot;

tree[root]=tree[pre];

tree[root].siz+=num;

if(l==r)

return;

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

if(val<=mid)

update(tree[root].lc,tree[pre].lc,l,mid,val,num);

else

update(tree[root].rc,tree[pre].rc,mid+1,r,val,num);

}

inline void add(int pos,int v)

{

int val=lower\_bound(b+1,b+len+1,a[pos])-b;

for(int i=pos;i<=n;i+=lowbit(i))

update(rt[i],rt[i],1,len,val,v);

}

inline int query(int x,int y,int l,int r,int k)

{

if(l==r) return l;

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

int sum=tree[tree[y].lc].siz-tree[tree[x].lc].siz;

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

sum-=tree[tree[qx[i]].lc].siz;

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

sum+=tree[tree[qy[i]].lc].siz;

if(k<=sum){

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

qx[i]=tree[qx[i]].lc;

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

qy[i]=tree[qy[i]].lc;

return query(tree[x].lc,tree[y].lc,l,mid,k);

}

else{

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

qx[i]=tree[qx[i]].rc;

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

qy[i]=tree[qy[i]].rc;

return query(tree[x].rc,tree[y].rc,mid+1,r,k-sum);

}

}

inline void prework()*//初始化操作 包括离散化等*

{

tot=len=rt[0]=0;

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

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

{

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

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

}

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

{

scanf("%s",op);

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

if(op[0]=='Q') *//查询*

scanf("%d",&q[i].c);

else if(op[0]=='C') *//修改*

q[i].c=-1,b[++len]=q[i].b;

}

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

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

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

{

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

update(srt[i],srt[i-1],1,len,val,1);

}

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

rt[i]=srt[0];

}

inline void mainwork()*//查询操作*

{

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

{

if(q[i].c>=0)

{

ansx=0,ansy=0;

for(int j=q[i].b;j;j-=lowbit(j))

qy[++ansy]=rt[j];

for(int j=q[i].a-1;j;j-=lowbit(j))

qx[++ansx]=rt[j];

printf("%d\n",b[query(srt[q[i].a-1],srt[q[i].b],1,len,q[i].c)]);

}

else

{

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

a[q[i].a]=q[i].b;

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

}

}

}

int main(){

int t;

scanf("%d",&t);

while(t--)

{

prework();

mainwork();

}

return 0;

}

**动态第K大 (树状数组套主席树)**

在静态主席树的基础上，外面套一层树状数组

主席树就是利用前缀和的性质

利用树状数组维护前缀和的功能，可以做到 log^2 的复杂度进行单点修改

#include<cstdio>

#include<cstring>

#include<algorithm>

#include<iostream>

#include<cmath>

#include<stack>

#include<queue>

using namespace std;

typedef long long ll;

const int maxn = 100010;

int n,m,q;

int a[maxn],b[maxn\*2],qa[maxn],qb[maxn],qc[maxn];

int rt[maxn\*600],lc[maxn\*600],rc[maxn\*600],sz[maxn\*600],tot;

int xx[maxn],yy[maxn],totx,toty;

char op[10];

void modify(int &i,int o,int l,int r,int p,int k){

i=++tot;

lc[i]=lc[o],rc[i]=rc[o],sz[i]=sz[o]+k;

if(l==r) return;

int mid=(l+r)/2;

if(p<=mid) modify(lc[i],lc[i],l,mid,p,k);

else modify(rc[i],rc[i],mid+1,r,p,k);

}

void add(int x,int k){

int p=lower\_bound(b+1,b+1+q,a[x])-b;

for(int i=x;i<=n;i+=i&(-i)){

modify(rt[i],rt[i],1,q,p,k);

}

}

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

if(l==r) return l;

int sum=0;

for(int i=1;i<=toty;i++) sum+=sz[lc[yy[i]]];

for(int i=1;i<=totx;i++) sum-=sz[lc[xx[i]]];

int mid=(l+r)/2;

if(sum>=k){

for(int i=1;i<=totx;i++) xx[i]=lc[xx[i]];

for(int i=1;i<=toty;i++) yy[i]=lc[yy[i]];

return query(l,mid,k);

}else{

for(int i=1;i<=totx;i++) xx[i]=rc[xx[i]];

for(int i=1;i<=toty;i++) yy[i]=rc[yy[i]];

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

}

}

ll read(){ ll s=0,f=1; char ch=getchar(); while(ch<'0' || ch>'9'){ if(ch=='-') f=-1; ch=getchar(); } while(ch>='0' && ch<='9'){ s=s\*10+ch-'0'; ch=getchar(); } return s\*f; }

int main(){

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

for(int i=1;i<=n;i++) a[i]=read(),b[++q]=a[i];

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

scanf("%s",op);

qa[i]=read(),qb[i]=read();

if(op[0]=='Q') qc[i]=read();

else b[++q]=qb[i];

}

sort(b+1,b+1+q);

q=unique(b+1,b+1+q)-b-1;

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

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

if(qc[i]){

totx=toty=0;

for(int j=qa[i]-1;j>0;j-=j&(-j)) xx[++totx]=rt[j];

for(int j=qb[i];j>0;j-=j&(-j)) yy[++toty]=rt[j];

int ans=query(1,q,qc[i]);

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

}else{

add(qa[i],-1);

a[qa[i]]=qb[i];

add(qa[i],1);

}

}

return 0;

}