树套树

线段树套线段树

虽然称为线段树套线段树，其实大多是通过树状数组套主席树来实现的(没错就是[带修主席树](http://hotpot.clatisus.com/LOTK%ef%bc%88lxh%ef%bc%89/主席树))，用以实现查询k在区间内的排名，查询区间内排名为k的数，修改某一位的值，查询某一位的前驱后继等。

线段树套平衡树

如果说，要解决区间上的问题，如最大值，区间修改等，我们选择的显然是线段树。

但是线段树不能查询第k大，不能查询一个数在区间的排名，自然也不能查询前驱和后继，这些只能交给平衡树来解决。

而涉及到区间翻转时，显然也是平衡树的范畴，这种时候我们就需要将两种树形结合起来解决这类更复杂的问题。

特别是涉及到区间翻转，分析后我们不难发现，在前驱后继□k在区间内的排名，排名为k的数的查询等操作，线段树套平衡树并不会比线段树套线段树更优。(查找排名为k的数由于二分甚至会多个log)

下面给出一道模板题[洛谷P3380 □模板】二逼平衡树（树套树）](https://www.luogu.com.cn/problem/P3380)

并没有找到带区间翻转和区间查询的题，读者如果有兴趣可以出一道（

下面给出代码(这个题卡常。。。甚至吃评测机状态，之前90分的代码突然变70分。。。然后A了)

平衡树的代码部分我就不进行讲解，用的是[平衡树](http://hotpot.clatisus.com/LOTK%ef%bc%88lxh%ef%bc%89/平衡树)的板子

insert/delete

插入操作并没有什么难度，就是在插入的这个点的这条链上的平衡树都插入这个点，删除同理

~~~cpp void insert(int x,int p,int w){

```
ins(tr[x].root,w);
if(tr[x].l==tr[x].r)return;
int mid=(tr[x].l+tr[x].r)>>1;
if(p<=mid)insert(x<<1,p,w);
else insert(x<<1|1,p,w);
```

} void Delete(int x,int p,int w){

```
del(tr[x].root,w);
if(tr[x].l==tr[x].r)return;
int mid=(tr[x].l+tr[x].r)>>1;
if(p<=mid)Delete(x<<1,p,w);
```
~~~

```
else Delete(x<<1|1,p,w);
```

} ~~~

query

关键的一步操作，我们将所要用到的平衡树的根放在一个vector里方便使用

~~~cpp vector<int> vec; void query(int x,int l,int r){

```
if(tr[x].l>r||tr[x].r<l)return;
if(l<=tr[x].l&&tr[x].r<=r){vec.push_back(tr[x].root);return;}
query(x<<1,l,r);
query(x<<1|1,l,r);
```

} ~~~

pre/suc

对于求前驱后继，我们就在每棵平衡树中求然后取最大最小即可。

~~~cpp int Pre(int k){

```
int ans=-INF;
for(auto x:vec)ans=max(ans,pre(k,x));
vec.clear();
return ans;
```

} int Suc(int k){

```
int ans=INF;
for(auto x:vec)ans=min(ans,suc(k,x));
vec.clear();
return ans;
```

} ~~~

rank

每棵平衡树求rank相加

~~~cpp int Rank(int k){

```
int ans=0;
for(auto x:vec)ans+=get_rank(k,x);
vec.clear();
```
~~~

```
return ans;
```

} ~~~

kth

这步操作比较复杂，也是复杂度最大的部分，因为我们并没有直接得到k大的方法，于是只能进行二分，然后用Rank来确定其排名，确定排名我们还需得到排名的上下界，否则易错判。

更重要的一点，要对一开始可能输入的值排序处理方便二分，不然二分增加的这个log偏大导致超时

~~~cpp int calc1(int k){

```
int ans=0;
for(auto x:vec)ans+=get_rank(k,x);
return ans;
```

} int calc2(int k){

```
int ans=0;
for(auto x:vec)ans+=get_rank2(k,x);
return ans;
```

} int b[maxn],bcnt; int get_kth(int k){

```
int L=1,R=bcnt;
while(L<=R){
    int mid=(L+R)>>1;
    int l=calc1(b[mid])+1,r=calc2(b[mid]);
    if(l<=k&&k<=r){vec.clear();return b[mid];}
    if(l>k)R=mid-1;
    if(r<k)L=mid+1;
}
```

} ~~~

完整代码

~~~cpp #include<algorithm> #include<stack> #include<ctime> #include<cstring> #include<cmath> #include<iostream> #include<iomanip> #include<cstdio> #include<queue> using namespace std; inline int read(){

```
int x=0,f=1;char ch=getchar();
while(!isdigit(ch)){if(ch=='-')f=-1;ch=getchar();}
while(isdigit(ch)){x=x*10+ch-'0';ch=getchar();}
return x*f;
```

} void write(int x){

```
if(x<0){putchar('-');write(-x);}
else {
    if(x/10)write(x/10);
    putchar(x%10+'0');
}
```

} #define ull unsigned long long const int maxn=2000005; ull hval[maxn]; int cnt; ull Rand(){

```
static ull r=1;
return (r*=998244353)%2147483647;
```

} int num[maxn],sum[maxn],son[maxn][2],val[maxn]; int newnode(int x){

```
++cnt;num[cnt]=sum[cnt]=1;
val[cnt]=x;hval[cnt]=Rand();
return cnt;
```

} void pushup(int x){

```
sum[x]=sum[son[x][0]]+sum[son[x][1]]+num[x];
```

} void Rot(int &p,int k){

```
int x=son[p][k];
son[p][k]=son[x][k^1];
son[x][k^1]=p;
pushup(p);pushup(x);
p=x;
```

} void ins(int &p,int x){

```
if(!p){p=newnode(x);return;}
++sum[p];
if(val[p]==x){++num[p];return;}
ins(son[p][x>val[p]],x);
if(hval[p]>hval[son[p][x>val[p]]])Rot(p,x>val[p]);
```

} void del(int &p,int x){

1. -sum[p];

if(val[p]==x){

```
    if(num[p]>1)--num[p];
    else if(!son[p][0]||!son[p][1])p=son[p][0]+son[p][1];
         else {Rot(p,hval[son[p][0]]>hval[son[p][1]]);del(p,x);}
}
else del(son[p][x>val[p]],x);
```

} int get_rank(int x,int root){

```
int p=root,ans=0;
while(p){
    if(val[p]==x){return ans+sum[son[p][0]];}
    else if(x>val[p]){ans+=sum[son[p][0]]+num[p];p=son[p][1];}
    else p=son[p][0];
}
return ans;
```

} int get_rank2(int x,int root){

```
int p=root,ans=0;
while(p){
    if(val[p]==x){return ans+sum[son[p][0]]+num[p];}
    else if(x>val[p]){ans+=sum[son[p][0]]+num[p];p=son[p][1];}
    else p=son[p][0];
}
return ans;
```

} const int INF=2147483647; int pre(int x,int root){

```
int p=root,ans=-INF;
while(p){
    if(val[p]>=x)p=son[p][0];
    else {ans=val[p];p=son[p][1];}
}
return ans;
```

} int suc(int x,int root){

```
int p=root,ans=INF;
while(p){
    if(val[p]<=x)p=son[p][1];
    else {ans=val[p];p=son[p][0];}
}
return ans;
```

} struct tree{

```
int l,r,root;
```

} tr[maxn]; int n,m,w[maxn]; void build(int x,int l,int r){

```
tr[x].l=l;tr[x].r=r;
if(l==r)return;
int mid=(l+r)>>1;
build(x<<1,l,mid);
build(x<<1|1,mid+1,r);
```

} void insert(int x,int p,int w){

```
ins(tr[x].root,w);
if(tr[x].l==tr[x].r)return;
int mid=(tr[x].l+tr[x].r)>>1;
if(p<=mid)insert(x<<1,p,w);
else insert(x<<1|1,p,w);
```

} vector<int> vec; void query(int x,int l,int r){

```
if(tr[x].l>r||tr[x].r<l)return;
if(l<=tr[x].l&&tr[x].r<=r){vec.push_back(tr[x].root);return;}
query(x<<1,l,r);
query(x<<1|1,l,r);
```

} int Rank(int k){

```
int ans=0;
for(auto x:vec)ans+=get_rank(k,x);
vec.clear();
return ans;
```

} int calc1(int k){

```
int ans=0;
for(auto x:vec)ans+=get_rank(k,x);
return ans;
```

} int calc2(int k){

```
int ans=0;
for(auto x:vec)ans+=get_rank2(k,x);
return ans;
```

} int b[maxn],bcnt; int get_kth(int k){

```
int L=1,R=bcnt;
while(L<=R){
    int mid=(L+R)>>1;
    int l=calc1(b[mid])+1,r=calc2(b[mid]);
    if(l<=k&&k<=r){vec.clear();return b[mid];}
    if(l>k)R=mid-1;
    if(r<k)L=mid+1;
}
```

} void Delete(int x,int p,int w){

```
del(tr[x].root,w);
if(tr[x].l==tr[x].r)return;
```

```
int mid=(tr[x].l+tr[x].r)>>1;
if(p<=mid)Delete(x<<1,p,w);
else Delete(x<<1|1,p,w);
```

} int Pre(int k){

```
int ans=-INF;
for(auto x:vec)ans=max(ans,pre(k,x));
vec.clear();
return ans;
```

} int Suc(int k){

```
int ans=INF;
for(auto x:vec)ans=min(ans,suc(k,x));
vec.clear();
return ans;
```

} struct Query{

```
int opt,l,r,k;
```

}q[maxn]; int main(){

```
n=read();m=read();
build(1,1,n);
for(int i=1;i<=n;++i){
    w[i]=read();
    insert(1,i,w[i]);
    b[++bcnt]=w[i];
}
for(int i=1;i<=m;++i){
    q[i].opt=read();
    q[i].l=read();q[i].r=read();
    if(q[i].opt!=3)q[i].k=read();
    else b[++bcnt]=q[i].r;
}
sort(b+1,b+bcnt+1);
for(int i=1,opt,l,r,k;i<=m;++i){
    opt=q[i].opt;l=q[i].l;r=q[i].r;
    if(opt!=3)k=q[i].k;
    if(opt==1){query(1,l,r);write(Rank(k)+1);putchar('\n');}
    if(opt==2){query(1,l,r);write(get_kth(k));putchar('\n');}
    if(opt==3){Delete(1,l,w[l]);w[l]=r;insert(1,l,w[l]);}
    if(opt==4){query(1,l,r);write(Pre(k));putchar('\n');}
    if(opt==5){query(1,l,r);write(Suc(k));putchar('\n');}
}
return 0;
```

} ~~~

平衡树套线段树(替罪羊)

一开始想到平衡树套线段树会感到很奇怪，毕竟大部分的平衡树树形是会改变的，这样的话在改变的过程中同时维护线段树就很难实现。

~~那么有没有这样一棵树，又能保持平衡，又能不轻易改变树形呢?~~

答案是有，替罪羊树完美保证了树形不轻易改变的性质，在重构的同时我们也重构我们的线段树，这样我们就能实现一种带插入的区间查询结构。能够实现插入删除以及区间k大(小)的查询。

而替罪羊树上的每个点，都保存着它子树的权值线段树，一旦重构，便自底向上进行线段树合并。

这里我们给出一道例题[洛谷P4278 带插入区间K小值](https://www.luogu.com.cn/problem/P4278)

这道题由于出题人魔改了数据，这种做法只能得20分，其余点TLE+MLE□不过题目来源bzoj好像木了，仅供练习使用吧。

ins

在插入替罪羊的同时我们插入线段树，同时用栈保留祖先关系(至于什么作用我们后面再说)

~~~cpp void ins(int x,int val){

```
int p=root;
while(p){
    insert(rt[p],val,0,70000,1);
    if(sum[son[p][0]]>=x){
        sta[++top]=p;sn[top]=0;
        p=son[p][0];
    }
    else {
        x-=sum[son[p][0]]+1;
        sta[++top]=p;sn[top]=1;
        p=son[p][1];
    }
}
p=newnode();w[p]=val;
insert(rt[p],w[p],0,70000,1);
son[sta[top]][sn[top]]=p;
for(int i=top;i>=1;--i)pushup(sta[i]);
```

} ~~~

build

这里也是替罪羊的rebuild部分，我们将重建的树所包含的点存好之后，对其重建并通过线段树合并整合信息。
~~~

~~~cpp void build(int &x,int l,int r){

```
if(l>r){x=0;return;}
if(l==r){
    x=s[l];
    insert(rt[x],w[x],0,70000,1);
    return;
}
int mid=(l+r)>>1;x=s[mid];
build(son[x][0],l,mid-1);
build(son[x][1],mid+1,r);
pushup(x);
Merge(rt[x],rt[son[x][0]],rt[son[x][1]],0,70000);
insert(rt[x],w[x],0,70000,1);
```

} ~~~

merge

线段树合并

~~~cpp void Merge(int &x,int son0,int son1,int l,int r){

```
if(!x)x=newtr();
sum(x)=sum(son0)+sum(son1);
if(l==r)return;
int mid=(l+r)>>1;
if(sum(ls(son0))|sum(ls(son1)))Merge(ls(x),ls(son0),ls(son1),l,mid);
if(sum(rs(son0))|sum(rs(son1)))Merge(rs(x),rs(son0),rs(son1),mid+1,r);
```

} ~~~

del

在对树拍扁重建时对线段树上的点回收，否则消耗空间过于巨大

~~~cpp void del(int &x){

```
if(!x)return;
del(ls(x));del(rs(x));
Q.push(x);x=0;
```

} ~~~

其余操作没什么好说的，基本就是线段树和替罪羊树的操作，这里根据题涉条件讲两个操作
~~~

关于区间提取

在替罪羊树上提取我们所需的点的区间，如果是完全包含，则保存这点的权值线段树，单点包含则保存该点并继续递归子树。

~~~cpp void query(int x,int l,int r){

```
if(!x)return;
int L=sum[son[x][0]],R=sum[x];
if(l==1&&r==R){vectr.push_back(rt[x]);return;}
if(l<=L+1&&r>=L+1)vecp.push_back(x);
if(r<=L)query(son[x][0],l,r);
else if(l>L+1)query(son[x][1],l-L-1,r-L-1);
else {
    if(l<=L)query(son[x][0],l,L);
    if(R>L+1)query(son[x][1],1,r-L-1);
}
```

} ~~~

关于得到答案

这里我们采取在区间上二分的方式□L□R应时刻保持与区间一致(否则会影响单点的判断)

~~~cpp #define unt unsigned int int solve(int l,int r,int k){

```
query(root,l,r);k--;
int L=0,R=70000;
unt n=vectr.size();
while(L<R){
    int mid=(L+R)>>1;
    int tot=0;
    for(auto x:vectr)tot+=sum(ls(x));
    for(auto x:vecp)if(L<=w[x]&&w[x]<=mid)++tot;
    if(tot>k){
        for(unt i=0;i<n;++i)vectr[i]=ls(vectr[i]);
        R=mid;
    }
    else {
        for(unt i=0;i<n;++i)vectr[i]=rs(vectr[i]);
        L=mid+1;k-=tot;
    }
}
vecp.clear();vectr.clear();
return L;
```

} ~~~

下面给出~~TLE+MLE~~完整代码

~~~cpp #include<algorithm> #include<stack> #include<ctime> #include<cstring> #include<cmath> #include<iostream> #include<iomanip> #include<cstdio> #include<queue> using namespace std; inline int read(){

```
int x=0,f=1;char ch=getchar();
while(!isdigit(ch)){if(ch=='-')f=-1;ch=getchar();}
while(isdigit(ch)){x=x*10+ch-'0';ch=getchar();}
return x*f;
```

} void write(int x){

```
if(x<0){putchar('-');write(-x);}
else {
    if(x/10)write(x/10);
    putchar(x%10+'0');
}
```

} const int maxn=20000005; const int maxm=70005; int n,root,sum[maxm]; int cnt; int son[maxm][2]; *线段树 #define ls(x) tr[x].ls #define rs(x) tr[x].rs #define sum(x) tr[x].sum struct tree{ int sum,ls,rs; }tr[maxn]; int rt[maxm],cntt; int w[maxm]; int s[maxm],tp; queue<int> Q; void pushup(int nw){ sum[nw]=sum[son[nw][0]]+sum[son[nw][1]]+1; } int tr_get(){ if(Q.empty())return ++cntt; int t=Q.front();Q.pop(); return t; } int newtr(){ int t=tr_get(); ls(t)=rs(t)=sum(t)=0; return t; } void Merge(int &x,int son0,int son1,int l,int r){ if(!x)x=newtr(); sum(x)=sum(son0)+sum(son1); if(l==r)return; int mid=(l+r)»1; if(sum(ls(son0))|sum(ls(son1)))Merge(ls(x),ls(son0),ls(son1),l,mid); if(sum(rs(son0))|sum(rs(son1)))Merge(rs(x),rs(son0),rs(son1),mid+1,r); } void insert(int &x,int val,int l,int r,int d){ if(!x)x=newtr(); sum(x)+=d; if(l==r)return; int mid=(l+r)»1; if(val⇐mid)insert(ls(x),val,l,mid,d); else insert(rs(x),val,mid+1,r,d); } void build(int &x,int l,int r){ if(l>r){x=0;return;} if(l==r){ x=s[l]; insert(rt[x],w[x],0,70000,1); return; } int mid=(l+r)»1;x=s[mid]; build(son[x][0],l,mid-1); build(son[x][1],mid+1,r); pushup(x); Merge(rt[x],rt[son[x][0]],rt[son[x][1]],0,70000); insert(rt[x],w[x],0,70000,1); } void del(int &x){ if(!x)return; del(ls(x));del(rs(x)); Q.push(x);x=0; }* 替罪羊 int newnode(){

```
int t=++cnt;
sum[t]=1;
son[t][0]=son[t][1]=0;
return t;
```

} void dfs(int x){

```
if(son[x][0])dfs(son[x][0]);
s[++tp]=x;del(rt[x]);
if(son[x][1])dfs(son[x][1]);
```

} int sta[maxm],top; bool sn[maxm]; void rebuild(int &nw){tp=0;dfs(nw);build(nw,1,tp);} void get_reb(int &nw,int p,int d){

```
if(nw==p)return;
if(max(sum[son[nw][0]],sum[son[nw][1]])>sum[nw]*0.75)rebuild(nw);
```
~~~

```
else get_reb(son[nw][sn[d]],p,d+1);
```

} int kth(int x){

```
int p=root;
while(true){
    if(sum[son[p][0]]>=x)p=son[p][0];
    else if(sum[son[p][0]]+1==x)return p;
         else {x-=sum[son[p][0]]+1;p=son[p][1];}
}
```

} void change(int x,int pv,int nv){

```
int p=root;
while(true){
    insert(rt[p],pv,0,70000,-1);
    insert(rt[p],nv,0,70000,1);
    if(sum[son[p][0]]>=x)p=son[p][0];
    else if(sum[son[p][0]]+1==x){w[p]=nv;return;}
         else {x-=sum[son[p][0]]+1;p=son[p][1];}
}
```

} void ins(int x,int val){

```
int p=root;
while(p){
    insert(rt[p],val,0,70000,1);
    if(sum[son[p][0]]>=x){
        sta[++top]=p;sn[top]=0;
        p=son[p][0];
    }
    else {
        x-=sum[son[p][0]]+1;
        sta[++top]=p;sn[top]=1;
        p=son[p][1];
    }
}
p=newnode();w[p]=val;
insert(rt[p],w[p],0,70000,1);
son[sta[top]][sn[top]]=p;
for(int i=top;i>=1;--i)pushup(sta[i]);
```

} vector<int> vectr,vecp; void query(int x,int l,int r){

```
int L=sum[son[x][0]],R=sum[x];
if(l==1&&r==R){vectr.push_back(rt[x]);return;}
if(l<=L+1&&r>=L+1)vecp.push_back(x);
if(r<=L)query(son[x][0],l,r);
else if(l>L+1)query(son[x][1],l-L-1,r-L-1);
```

```
else {
    if(l<=L)query(son[x][0],l,L);
    if(R>L+1)query(son[x][1],1,r-L-1);
}
```

} #define unt unsigned int int solve(int l,int r,int k){

```
query(root,l,r);k--;
int L=0,R=70000;
unt n=vectr.size();
while(L<R){
    int mid=(L+R)>>1;
    int tot=0;
    for(auto x:vectr)tot+=sum(ls(x));
    for(auto x:vecp)if(L<=w[x]&&w[x]<=mid)++tot;
    if(tot>k){
        for(unt i=0;i<n;++i)vectr[i]=ls(vectr[i]);
        R=mid;
    }
    else {
        for(unt i=0;i<n;++i)vectr[i]=rs(vectr[i]);
        L=mid+1;k-=tot;
    }
}
vecp.clear();vectr.clear();
return L;
```

} int main(){

```
n=read();
for(int i=1;i<=n;++i){newnode();w[i]=read();s[i]=i;}
build(root,1,n);
int q=read();char s[4];
int lst=0;
for(int i=1,x,y,k;i<=q;++i){
    scanf("%s",s);
    x=read()^lst;y=read()^lst;
    if(s[0]=='Q'){k=read()^lst;write(lst=solve(x,y,k));putchar('\n');}
    if(s[0]=='M'){int p=kth(x);change(x,w[p],y);}
    if(s[0]=='I'){ins(x-1,y);get_reb(root,cnt,1);top=0;}
}
return 0;
```

} ~~~

平衡树套线段树(非旋treap)

根据我们之前的思路，非旋treap的树形也不会轻易改变，因此我们可以采取非旋treap来实现，
在split,merge之后用线段树合并来对所需要的信息进行维护

这是一个坑，待填