**暨南大学本科实验报告专用纸**

课程名称 数据结构实验 成绩评定

实验项目名称 线段树套平衡树 指导教师 刘波、郭科芩

实验项目编号 8060154914 实验项目类型 创新性 实验地点 N116

学生姓名 阮炜霖 学号 2020101603

学院 信息科学技术学院 系 计算机系 专业 网络工程

实验时间 2021 年 10 月 9 日 上 午～ 10 月 9 日 上 午

**（一）实验目的和要求；**

目的：熟练掌握线段树以及平衡树两种数据结构，并将两者结合使用解决问题。

要求：任选一种高级程序语言编写源程序，并调试通过，测试正确。

**（二）实验主要内容；**

（1） 将n个玩家的游戏得分录入系统，并使用树套树的数据结构进行维护。

（2）建立系统在线维护这些玩家的得分，并且支持以下操作：

①询问得分为k在系统中某编号范围[l,r]的排名；

②查询系统某编号范围[l,r]中排名为k的得分；

③提供编号并修改该学生的得分；

④查询得分为k在系统中某编号范围[l,r]的前一个得分（相邻比其大）

⑤查询得分为k在系统中某编号范围[l,r]的后一个得分（相邻比其小）

**（三）实验原理**

我们用一颗线段树管理下标区间[L,R]，线段树的每个节点都是一颗平衡树，并且存储了区间中的所有数的信息，这样的话，所有区间都被分成logN个线段树的节点。

①查询区间中某一个数的排名：在线段树中找到区间相应结点，然后每个节点的平衡树内查询对应数的排名并求和即可。时间复杂度**O(log^2 n)**

②查询区间内排名为k的数的值：由于这项操作在线段树上不可加。所以我们考虑转换为判定一个数是不是区间内排名为k的。这个可以在**O(log^2 n)** 的时间内通过操作1完成。那么我们考虑二分答案，就可以解决这个问题。时间复杂度**O(log^3 n)**

③单点修改：我们在线段树上找到所有覆盖这个点的区间，然后在所有区间对应的平衡树中删除原数，加入新数即可。时间时间复杂度**O(log^2 n)**

④查询区间内一个数前驱、后继：

我们只需要对于所有区间分别查询，然后答案相应的取 max/min即可。时间复杂度**O(log^2 n)**

（注：上述log表示以2为底数的对数函数）

关于复杂度证明：我们通过线段树的学习可以知道，对其区间进行查询和修改，一共是qlogn的复杂度，其中q是询问次数，n是序列元素个数。修改是单点的，但我们要找到所有覆盖到该元素的区间并在其平衡树上删除和新增元素，大概是logn个区间，因此复杂度是两个log的。查询排名、查询前驱和后继同理，因为我们访问区间的个数大概是logn个，而在平衡树上每次都需要logn（这里的n是平衡树的元素个数）,那么平均下来操作的级别也是两个log的。查询排名为k的值是三个log，是加上了二分的复杂度，因为操作的不可加性，我们只能折半得再线段树上找区间某个排名为k的值，比其他操作更慢。

**（四）实验步骤及调试分析；**

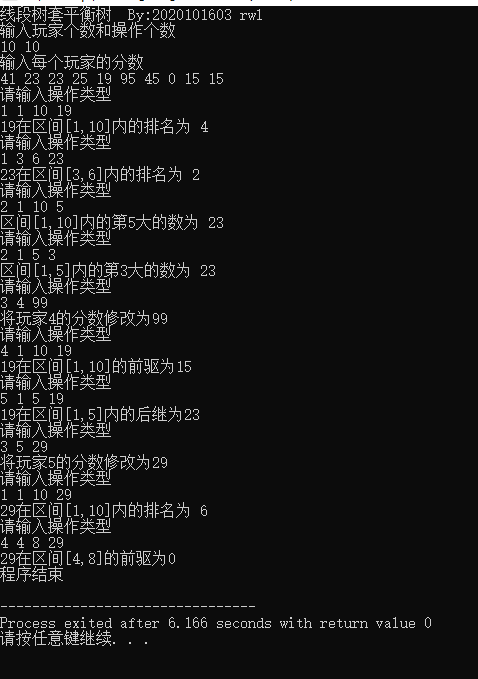
游戏中，玩家数量和得分以及查询的数量都可能会是一个相当大的数值，在本实验中上限为1e5和1e8(分数)。在这个范围内不能使用线性复杂度的操作，而平衡树可以解决插入、删除、查询排名、查询分数、查询前驱、查询后继等操作，我们可以尝试使用该数据结构来解决问题。但是单单用平衡树解决不了编号范围的限定，我们需要管理每个区间的下表来存储区间所构成平衡树的信息，因此我们尝试把线段树套用进去。这样，我们通过线段树就可以访问到每个区间，并且把每个区间都用平衡树来表示。

**（五）实验结果及分析；**

一开始发现，因为数据量过大，程序运行的时间会超过1秒的时限，我尝试使用快读来节省读入数据的时间。接着，因为Splay的巨大常数，所以我需要换成Treap或者FHQ Treap这一类常数更小的平衡树（红黑树因为码量原因不做考虑）。

调试过程中，线段树和平衡树的操作需要分类处理，因此使用命名空间将其单独封装起来，更方便调试。

通过实验结果，我们得出结论：该程序已实现了游戏玩家的得分录入、修改、查询排名所需得分、查询前驱和后继分数以及查询得分所在排名等操作。



**（六）附录：源程序**

//#pragma GCC optimize("Ofast", "inline", "-ffast-math")

//#pragma GCC target("avx,sse2,sse3,sse4,mmx")

#include<bits/stdc++.h>

#define inf 2147483647

//#define int long long

using namespace std;

const int N=1e5+7;

int read(){ //快读优化

int x=0,f=1;

char ch=getchar();

while(ch<'0'||ch>'9'){if(ch=='-') f=f\*-1;ch=getchar();}

while(ch>='0'&&ch<='9'){x=x\*10+ch-'0';ch=getchar();}

return x\*f;

}

int n,m,a[N];

namespace Treap{ //Treap平衡树

struct balanced{

int w,sz,num,fix,ch[2];

//分别为节点的值，子树大小，值的个数，平衡因子和儿子编号

};

int tot;

balanced tree[N\*20];

int newnode(int w){ //新建节点

++tot;

tree[tot].w=w;

tree[tot].fix=rand();

tree[tot].num=1;

tree[tot].ch[0]=tree[tot].ch[1]=0;

tree[tot].sz=1;

return tot;

}

void pushup(int p){ //维护节点的子树大小

tree[p].sz=tree[tree[p].ch[0]].sz+tree[tree[p].ch[1]].sz+tree[p].num;

}

void rotate(int &p,int d){ //旋转操作

int y=tree[p].ch[d];

tree[p].ch[d]=tree[y].ch[d^1];

tree[y].ch[d^1]=p;

pushup(p);

pushup(y);

p=y;

}

void insert(int &p, int w){ //插入节点

if (!p) p=newnode(w);

else if(tree[p].w==w) ++tree[p].num;

else{

if(tree[p].w>w){

insert(tree[p].ch[0],w);

if(tree[tree[p].ch[0]].fix>tree[p].fix) rotate(p, 0);

}else{

insert(tree[p].ch[1],w);

if(tree[tree[p].ch[1]].fix>tree[p].fix) rotate(p, 1);

}

}

pushup(p);

}

void remove(int &p, int w){ //删除节点

if(tree[p].w>w) remove(tree[p].ch[0], w);

else if(tree[p].w<w) remove(tree[p].ch[1], w);

else{

if(tree[p].num>1) --tree[p].num;

else{

if(!tree[p].ch[0]&&!tree[p].ch[1]) p=0;

else if(!tree[p].ch[0]){

rotate(p,1);

remove(tree[p].ch[0],w);

}

else if(!tree[p].ch[1]){

rotate(p,0);

remove(tree[p].ch[1],w);

}else{

if(tree[tree[p].ch[0]].fix>tree[tree[p].ch[1]].fix){

rotate(p,0);

remove(tree[p].ch[1],w);

}else{

rotate(p,1);

remove(tree[p].ch[0],w);

}

}

}

}

if (p) pushup(p);

}

int queryrank(int p,int k){ //询问平衡树上k值的最高排名

if(!p) return 0;

if(tree[p].w>k) return queryrank(tree[p].ch[0],k);

else if(tree[p].w==k) return tree[tree[p].ch[0]].sz;

else return tree[tree[p].ch[0]].sz + tree[p].num + queryrank(tree[p].ch[1], k);

}

int querynum(int p,int k){ //询问平衡树上排名为k的值

if(tree[tree[p].ch[0]].sz+1==k) return tree[p].w;

else if(tree[tree[p].ch[0]].sz+1<k) return querynum(tree[p].ch[1], k - 1 - tree[tree[p].ch[0]].sz);

else return querynum(tree[p].ch[0],k);

}

int querypre(int p,int k){ //询问平衡树上k值的排名

if(!p) return -inf;

if(tree[p].w>=k) return querypre(tree[p].ch[0],k);

else return max(tree[p].w,querypre(tree[p].ch[1],k));

}

int querysuf(int p,int k){ //询问平衡树上k的前驱

if(!p) return inf;

if(tree[p].w<=k) return querysuf(tree[p].ch[1],k);

else return min(tree[p].w,querysuf(tree[p].ch[0],k));

}

};

namespace SEG{

struct segment{

int l,r,root;

};

segment tree[N\*8];

void build(int p,int l,int r){

tree[p].l=l;

tree[p].r=r;

for(int i=l;i<r+1;++i) Treap::insert(tree[p].root,a[i]);//对每个区间建平衡树

if(l!=r){

int mid=(l+r)/2;

build(p\*2,l,mid); //左区间建树

build(p\*2+1,mid+1,r); //右区间建树

}

}

void modify(int p, int x, int y){ //线段树单点修改

Treap::remove(tree[p].root,a[x]); //删除原数并新增

Treap::insert(tree[p].root,y);

if (tree[p].l==tree[p].r) return;

int mid=(tree[p].l+tree[p].r)/2;

if(x>mid) modify(p\*2+1,x,y); //修改右区间

else modify(p\*2,x,y); //修改左区间

}

int queryrank(int p,int l,int r,int k){

if(tree[p].l>r||tree[p].r<l) return 0;

if(tree[p].l>=l&&tree[p].r<=r) return Treap::queryrank(tree[p].root,k);

else return queryrank(p\*2,l,r,k)+queryrank(p\*2+1,l,r,k);

}

int querynum(int u,int v,int k){

int l=0,r=1e8;

while (l<r){

int mid=(l+r+1)/2;

if(queryrank(1,u,v,mid)<k) l=mid;

else r=mid-1;

}

return r;

}

int querypre(int p,int l,int r,int k){ //对每个区间查询前驱并求最大值

if(tree[p].l>r||tree[p].r<l) return -inf;

if(tree[p].l>=l&&tree[p].r<=r) return Treap::querypre(tree[p].root,k);

else return max(querypre(p\*2,l,r,k),querypre(p\*2+1,l,r,k));

}

int querysuf(int p, int l, int r, int k){ //对每个区间查询后继并求最小值

if(tree[p].l>r||tree[p].r<l) return inf;

if(tree[p].l>=l&&tree[p].r<=r) return Treap::querysuf(tree[p].root,k);

else return min(querysuf(p\*2,l,r,k), querysuf(p\*2+1,l,r,k));

}

}

signed main(){

cout<<"线段树套平衡树 By:2020101603 rwl \n";

cout<<"输入玩家个数和操作个数 \n";

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

cout<<"输入每个玩家的分数 \n";

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

SEG::build(1,1,n); //建线段树

for (int i=0;i<m;++i){

cout<<"请输入操作类型\n";

int opt=read();

if(opt==3){

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

printf("将玩家%d的分数修改为%d\n",x,y);

SEG::modify(1,x,y); //修改某一位置上的值

a[x]=y;

}else{

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

if(opt==1) printf("%d在区间[%d,%d]内的排名为 %d\n",k,l,r,SEG::queryrank(1,l,r,k)+1); //查询k在区间内的排名

else if(opt==2) printf("区间[%d,%d]内的第%d大的数为 %d\n",l,r,k,SEG::querynum(l,r,k)); //查询区间内排名为k的值

else if(opt==4) printf("%d在区间[%d,%d]的前驱为%d\n",k,l,r,SEG::querypre(1,l,r,k)); //查询k在区间内的前驱

else printf("%d在区间[%d,%d]内的后继为%d\n",k,l,r,SEG::querysuf(1,l,r,k)); //查询k在区间内的后继

}

}

cout<<"程序结束\n";

return 0;

}

**暨南大学本科实验报告专用纸(附页)**