**实验四：区间树上重叠区间的查找算法**

**姓名：陈鸿绪 学号：PB21000224 日期：11.1**

**实验内容：**对红黑树数据结构进行修改，使其成为一颗区间数，并实现区间树上的重叠区间查找算法。要求从给定文件中构造区间树，要求插入顺序按照文本中一致。最后根据待查询区间，输出对应区间树对应结点区间，若没有则输出nil。

**实验目的：**了解红黑树基础数据结构区间树，熟悉区间树查找重叠区间算法。

**区间树的数据结构：**选择基础的数据结构为红黑树，每个结点x代表的是一个区间x.int，结点关键字选择为x.int.low。对于本次实验只涉及到的相关操作有二叉排序树插入、红黑树性质维护、区间树重叠区间查找。

**主代码解释：**

void LEFT\_ro(TNode \*p){

if(p->p!=NULL){

if(p->p->L==p) p->p->L=p->R;

else p->p->R=p->R;

}

else{

Head=p->R;

}

int a1=((p->R->L==NULL)?(-1e3):(p->R->L->max));

int a2=(p->L==NULL)?(-1e3):(p->L->max);

int a3=(p->R->R==NULL)?(-1e3):(p->R->R->max);

p->max=max(max(a1,a2),p->int\_high);

p->R->max=max(max(p->max,p->R->int\_high),a3);

TNode \*temp=p->R->L;

p->R->p=p->p;

p->R->L=p;

p->p=p->R;

p->R=temp;

if(temp) temp->p=p;

}// 左旋结点，同时维护附加信息

void RIGHT\_ro(TNode \*p){

if(p->p!=NULL){

if(p->p->L==p) p->p->L=p->L;

else p->p->R=p->L;

}

else{

Head=p->L;

}

int a1=((p->L->R==NULL)?(-1e3):(p->L->R->max));

int a2=(p->R==NULL)?(-1e3):(p->R->max);

int a3=(p->L->L==NULL)?(-1e3):(p->L->L->max);

p->max=max(max(a1,a2),p->int\_high);

p->L->max=max(max(p->max,p->L->int\_high),a3);

TNode \*temp=p->L->R;

p->L->R=p;

p->L->p=p->p;

p->p=p->L;

p->L=temp;

if(temp) temp->p=p;

}// 右旋结点，同时维护附加信息

void RB\_insert\_fixup(TNode \*z){

while(z->p->color==0){

if(z->p->p->L==z->p){

TNode \*y=z->p->p->R;

if(y!=NULL&&y->color==0){

z->p->color=1;

y->color=1;

z->p->p->color=0;

z=z->p->p;

}

else if(z==z->p->R){

z=z->p;

LEFT\_ro(z);

}

else{

z->p->color=1;

z->p->p->color=0;

RIGHT\_ro(z->p->p);

}

}

else{

TNode \*y=z->p->p->L;

if(y!=NULL&&y->color==0){

z->p->color=1;

y->color=1;

z->p->p->color=0;

z=z->p->p;

}

else if(z==z->p->L){

z=z->p;

RIGHT\_ro(z);

}

else{

z->p->color=1;

z->p->p->color=0;

LEFT\_ro(z->p->p);

}

}

Head->color=1;

if(z==Head) break;

}

}// 插入结点后维护红黑树的性质

void RB\_insert(TNode \*z){

TNode \*p=Head,\*pre;

while(true){

if(z->int\_low<=p->int\_low) pre=p,p=p->L;

else pre=p,p=p->R;

if(z->max>pre->max) pre->max=z->max;

if(p==NULL){

if((pre->int\_low)>=(z->int\_low)) {pre->L=z;}

else {pre->R=z;}

z->L=NULL,z->R=NULL,z->p=pre;

break;

}

}

RB\_insert\_fixup(z); // 二叉排序树插入后进行红黑树维护

}// 插入按照二叉排序树进行插入

void RB\_search(TNode \*z,TNode \*Head\_,int &flag){

TNode \*p=Head\_;

if(!p){

// printf("nil\n");

return ;

} // 如果是空则返回没有重叠区间

if(!((p->int\_low>z->int\_high)||(p->int\_high<z->int\_low))){

printf("[%d %d]\n",p->int\_low,p->int\_high);

flag=1;

RB\_search(z,p->L,flag);

RB\_search(z,p->R,flag);

}// 如果有重叠区间，让指示flag为1表明存在重叠区间，同时打印，再进入下一次。

else if(p->int\_low>z->int\_high){

int a1=(p->L==NULL)?-1e3:p->L->max;

int a2=(p->R==NULL)?-1e3:p->R->max;

if(!(p->L)) return;

// printf("nil\n");

if(z->int\_low>a1) return ; // 如果待查询区间的下界大于当前结点左孩子的附加//max则直接return

else RB\_search(z,p->L,flag); //否则到说明在以左结点为根的子树上可能存在重叠//区间

return;

}// 如果待查询区间在当前结点的左侧无重叠

else if(p->int\_high<z->int\_low){

int a1=(p->L==NULL)?-1e3:p->L->max;

int a2=(p->R==NULL)?-1e3:p->R->max;

if(a1>=z->int\_low) RB\_search(z,p->L,flag);// 判别待查询区间的下界是否比左结点//的附加max小，如果小则必然存在重叠区间在以左结点为根的子树中

if(a2>=z->int\_low) RB\_search(z,p->R,flag);// 判别待查询区间的下界是否比右结点//的附加max小，如果小则可能存在重叠区间在以右结点为根的子树中

return ;

}

return;

}// 查找重叠区间

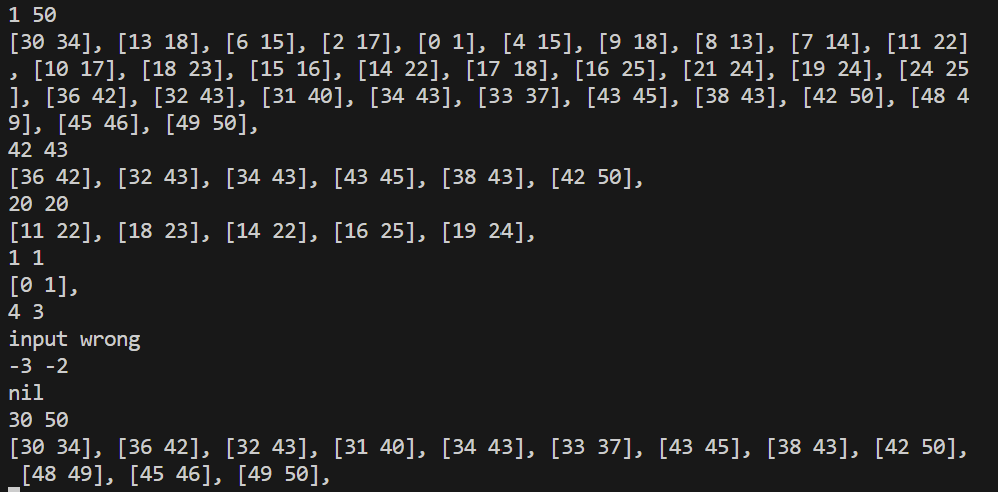
**算法时间复杂度定性分析：**

分析以上代码实现的算法，可以发现对于一个有重叠的区间，花费了最多O(log(n))查找（从根到叶子结点），假设总共有k个返回重叠区间，则会花费kO(log(n))时间复杂度，即O(klog(n))，然而注意到查询的时候至多遍历整个区间树，所以总的时间复杂度在O( min{ klog(n) , n } )。

实际上，在某些特殊极端情形，在算法中有可能存在某些路径，我们只能说明可能存在重叠区间，所以在这种情况下会导致总时间复杂度会增加，但是考虑一般非极端的平均情形，时间复杂度的增加并不能对原量级产生本质影响，所以可以近似认为该算法总时间复杂度达到O( min{ klog(n) , n } )。

**算法测试结果：**

选取若干对数据，得到如下测试结果：



根据验证得到该结果确实为准确结果。

**实验过程中遇到的困难及收获：**由于上次实验已经完成了红黑树插入算法，所以本次实验只需要将红黑树结点的数据部分改为区间即可。然后再加入待查找区间查找程序即可。一开始在理解实验内容的时候出现了差错，误认为需要查询完全包含或者被完全包含区间，而非重叠区间，最后在助教的指正下意识到是需要查找重叠区间。本次实验加深了我对红黑树、区间树的理解。