## **区间操作 文艺平衡树**

Splay的区间翻转：   
****【建树操作】****   
注意建树每次返回根节点的编号   
区分一个****节点的排名****和****这个节点的值****：这个节点的排名是它是当前数组中的第几个，用左儿子的size+1表示；这个节点的值是题目中输入的数字，在本题中是1~n；   
增加数字为1和n+2的两个哨兵节点，因为如果对区间1~x或 x~n操作，用到前后的节点就需要1和n+2。

在****main****函数中

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

data[1]=-inf; data[n+2]=inf;

rt=build\_tree(0,1,n+2);

****build\_tree:****

int build\_tree(int fa,int l,int r)

{

if (l>r) return 0;

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

int now=++sz;

key[now]=data[mid]; f[now]=fa; tag[now]=0;

ch[now][0]=build\_tree(now,l,mid-1);

ch[now][1]=build\_tree(now,mid+1,r);

pushup(now);

return now;

}

****【下传标记】****   
每到一个新节点都要pushdown

void pushdown(int x)

{

if (x && tag[x])

{

tag[ch[x][0]]^=1;

tag[ch[x][1]]^=1;

swap(ch[x][0],ch[x][1]);

tag[x]=0;

}

}

****【splay操作】****   
与普通的splay没有什么不同，比上面的goal加了一个目标goal而已

void splay(int x,int goal)*//比上面的goal加了一个目标goal而已*

{

for (int fa; (fa=f[x])!=goal; rotate(x))

if (f[fa]!=goal)

rotate((get(x)==get(fa))?fa:x);

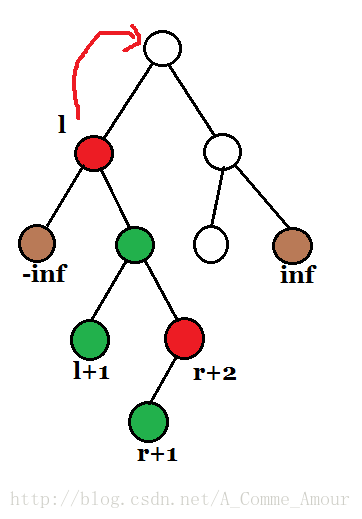
if (!goal) rt=x;

}

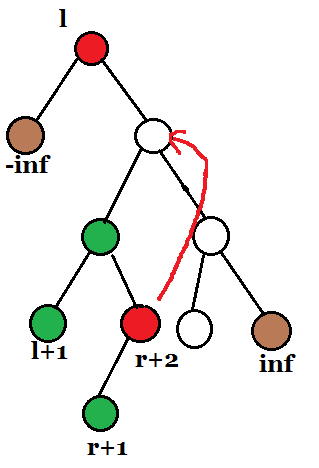
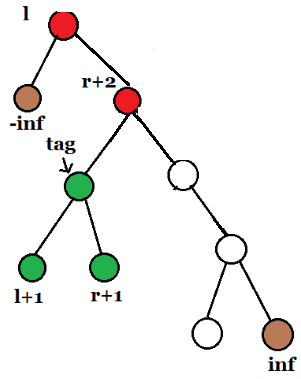
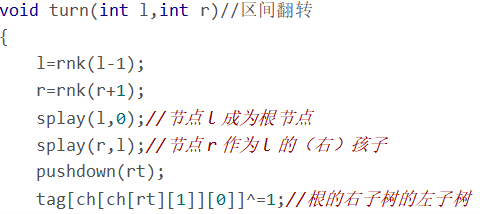
最重要的来了   
****【turn翻转区间】****   
首先，也是最重要的，我们认为伸展树中序遍历即是我们维护的序列！什么意思呢？比如有数据在数组中这样存放：a[5]={5,4,3,1,2};那么存入伸展树后，再中序遍历的结果应该还是：{5，4，3，1，2}。即下标从小到大，而不是里面的值从小到大！这是与SBT树最大的不同！   
****原理****：若要翻转[l+1, r+1]，将r+2 Splay到根，将l Splay到 r+2 的左儿子，然后[l+1, r+1]就在根节点的右子树的左子树位置了，给它打上标记（理解是否有误？）

来看图片：

##### **step1**

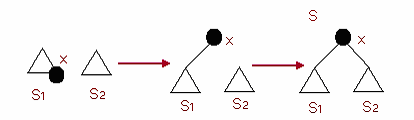
先使l旋转到根   


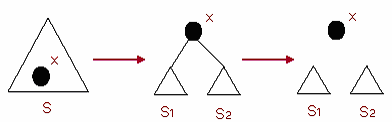
##### **step2**

使r+2旋转到根的右孩子，   
   
由于l < r+2,此时l成了r+2的左子树，那么r+2的右子树的左子树即为所求得区间，我们就可以对这棵子树随意操作了！比如删除整个区间，区间内的每个数都加上x，区间翻转，区间旋转等。   


### **其他操作**

Join(S1,S2)：将两个伸展树S1与S2合并成为一个伸展树。其中S1的所   
有元素都小于S2的所有元素。   
首先，我们找到伸展树S1 中最大的一个元素x，再通过Splay(x,S1)将x 调   
整到伸展树S1 的根。然后再将S2 作为x 节点的右子树。这样，就得到了新的   
伸展树S。如图所示



Split(x,S)：以x 为界，将伸展树S 分离为两棵伸展树S1 和S2，其中S1   
中所有元素都小于x，S2中的所有元素都大于x。   
首先执行Find(x,S)，将元素x 调整为伸展树的根节点，则x 的左子树就是   
S1，而右子树为S2。如图所示   


## **Code：**

#include<cstdio>#include<cstring>#include<algorithm>#include<iostream>

using namespace std;

const int MAXN=1000007;

const int inf=1e9;

int f[MAXN],cnt[MAXN],ch[MAXN][2],size[MAXN],key[MAXN],tag[MAXN],sz,rt;

int n,m,x,y,data[MAXN];

bool get(int x)

{

return ch[f[x]][1]==x;

}

void pushup(int x)

{

size[x]=size[ch[x][0]]+size[ch[x][1]]+1;

}

void pushdown(int x)

{

if (x && tag[x])

{

tag[ch[x][0]]^=1;

tag[ch[x][1]]^=1;

swap(ch[x][0],ch[x][1]);

tag[x]=0;

}

}

void rotate(int x)

{

int old=f[x],oldf=f[old],which=get(x);

pushdown(old); pushdown(x);*//不要忘记pushdown*

ch[old][which]=ch[x][which^1]; f[ch[old][which]]=old;

ch[x][which^1]=old; f[old]=x;

f[x]=oldf;

if (oldf) ch[oldf][ch[oldf][1]==old]=x;

pushup(old); pushup(x);

}

void splay(int x,int goal)*//比上面的goal加了一个目标goal而已*

{

for (int fa; (fa=f[x])!=goal; rotate(x))

if (f[fa]!=goal)

rotate((get(x)==get(fa))?fa:x);

if (!goal) rt=x;

}

int build\_tree(int fa,int l,int r)

{

if (l>r) return 0;

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

int now=++sz;

key[now]=data[mid]; f[now]=fa; tag[now]=0;

ch[now][0]=build\_tree(now,l,mid-1);

ch[now][1]=build\_tree(now,mid+1,r);

pushup(now);

return now;

}

int rnk(int x)

{

int now=rt;

while (1)

{

pushdown(now);

if (x<=size[ch[now][0]]) now=ch[now][0];

else

{

x-=size[ch[now][0]]+1;

if (!x) return now;

now=ch[now][1];

}

}

}

void turn(int l,int r)//区间翻转

{

l=rnk(l-1);

r=rnk(r+1);

splay(l,0);*//节点l成为根节点*

splay(r,l);*//节点r作为l的（右）孩子*

pushdown(rt);

tag[ch[ch[rt][1]][0]]^=1;*//根的右子树的左子树*

}

void write(int now)

{

pushdown(now);

if (ch[now][0]) write(ch[now][0]);

if (key[now]!=-inf && key[now]!=inf) printf("%d ",key[now]);

if (key[ch[now][1]]) write(ch[now][1]);

}

int main()

{

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

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

data[1]=-inf; data[n+2]=inf;

rt=build\_tree(0,1,n+2);

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

{

scanf("%d%d",&x,&y);

turn(x,y);

}

write(rt);

return 0;

}