Tree Rotation

一. 考察内容：

线段树 线段树合并 动态开点 贪心 动态规划

二. 题目分析：

[题目大意]

在一棵具有n个叶子的完全二叉树上，每个叶子节点都有一个1-n权值（不重复），你可以对于任何一个非叶子结点进行“旋转”操作，这个操作会使左右儿子的顺序调换，这个操作不限次数，随后先序遍历整棵树，将访问到的有权值的节点记录下来，最小化“记录的逆序对个数”。

[写题思路]

观察题目，我们会得出一个结论：无论如何旋转某个节点的子树，都不会影响该子树之外的逆序对个数，例如一个序列：3 7 2 9 0 4 5 1，其中2 9 0 4属于一个子树，这样旋转这棵子树，只会改变“2”，“9”，“0”，“4”之间的位置关系，而不会改变“2 9 0 4”与前面、后面的位置关系。

我们考虑对于每个节点，记录下他的子树权值都有哪些，例：若vis[3]=true，则该子树有权值为3的节点。随后我们考虑在树上转移这个状态，很显然，每个节点的vis等于其子节点的vis“或”值。对于这个节点，我们考虑计算其子节点的左儿子中每个数大于右儿子中输的个数（即逆序对个数）来判断是否旋转这个节点，并把这两个儿子相互产生的逆序对个数加入答案，最后将这两个节点的vis值合并到父节点。

直到最后，我们把所有节点都合并到根节点，算法完成。

但是这样的时间复杂度是不令人满意的，每次合并的时间复杂度为O(n)，一共需要合并2(n-1)次，复杂度在$O(n^2)$以上，我们考虑用线段树代替这个vis数组，用线段树合并实现数组的合并，并且用动态开点减少内存消耗，即可实现上述算法。

另外一点需要注意的是，对于每个节点，求子节点逆序对时可以在线段树合并的过程中顺便完成。

三. 代码实现：

```
#define _CRT_SECURE_NO_DEPRECATE
/************************
*创建时间：2018 08 28
*文件类型：源代码文件
*题目来源：BZOJ
*当前状态：已通过
*备忘录：线段树 线段树合并 贪心 动态开点
先把子树换好，再换根
*作者：HtBest
************************/
#include <stdio.h>
#include <iostream>
#include <string>
#include <stdlib.h>
#include <string.h>
#include <algorithm>
#include <queue>
// #include <sys/wait.h>
// #include <sys/types.h>
// #include <unistd.h>
using namespace std;
#define MAXN 400000
```

```
int n,fa[MAXN],lson[MAXN],rson[MAXN],v[MAXN],_dfsx,cnt;
long long ans;
struct NODE
{
    int lson,rson,v;
    void reset(){lson=rson=v=0;}
    void merge(NODE a,NODE b)
    {
        v=a.v+b.v;
    }
}o[20000000];
struct segmentTree
{
    int root;
    void getid(int &x)
    {
        x=++cnt;
        o[x].reset();
    }
    void update(int &x,int l,int r,int p)//将一个节点更新为1
    {
        if(!x)getid(x);
        if(l==r)o[x].v=1;
        else
        {
            int m=(l+r)>>1;
            if(p<=m)update(o[x].lson,l,m,p);
            else update(o[x].rson,m+1,r,p);
            o[x].merge(o[o[x].lson],o[o[x].rson]);
        }
    }
    int query(int x,int l,int r,int p)//查询一个节点
    {
        if(!x)return 0;
        if(l==r)return o[x].v;
        else
        {
            int m=(l+r)>>1;
            if(p<=m)return query(o[x].lson,l,m,p);
            else return query(o[x].rson,m+1,r,p);
        }
    }
    void update(int x){update(root,1,200000,x);}
    int query(int x){return query(root,1,200000,x);}
}seg[MAXN];
/* Variable explain:
n:叶子数
fa[i]:i节点的父亲
lson[i]:i节点的左儿子
rson[i]:i节点的右儿子
v[i]:i节点的权值
_dfsx:dfs序标记
cnt:统计线段树节点使用情况
o[i]:线段树节点
ans:答案
seg[i]:第i个节点的权值线段树
*/
int merge(int a,int b,long long &ans1,long long &ans2)
{
    if(!a||!o[a].v)return b;
    if(!b||!o[b].v)return a;
    if(o[a].lson||o[a].rson||o[b].lson||o[b].rson)
    {
        ans1+=(long long)o[o[a].lson].v*o[o[b].rson].v;
        ans2+=(long long)o[o[b].lson].v*o[o[a].rson].v;
        o[a].lson=merge(o[a].lson,o[b].lson,ans1,ans2);
        o[a].rson=merge(o[a].rson,o[b].rson,ans1,ans2);
        o[a].merge(o[o[a].lson],o[o[a].rson]);
    }
```

```
    return a;
}
void addt(int a,int b,int c)
{
    fa[b]=fa[c]=a;
    lson[a]=b,rson[a]=c;
}
int dfs1()
{
    int ls1,dfsx=++_dfsx,l,r;
    scanf("%d",&ls1);
    if(ls1){v[dfsx]=ls1;return dfsx;}
    l=dfs1();
    r=dfs1();
    addt(dfsx,l,r);
    return dfsx;
}
void dfs2(int a)
{
    // printf("%d\n",a);
    long long ls1=0,ls2=0;
    if(!lson[a]||!rson[a])return;
    dfs2(lson[a]);
    dfs2(rson[a]);
    seg[a].root=merge(seg[lson[a]].root,seg[rson[a]].root,ls1,ls2);
    ans+=min(ls1,ls2);//这两句话有很大的差别，在本机，下面那个才可以AC，上面这个会RE，BZOJ上都可以AC
    // ans+=ls1<ls2?ls1:ls2;
}
void read()
{
    // freopen(".in","r",stdin);
    // freopen(".out","w",stdout);
    scanf("%d",&n);
    dfs1();
    return;
}
int main()
{
    read();
    for(int i=1;i<=_dfsx;++i)if(v[i])seg[i].update(v[i]);
    dfs2(1);
    printf("%lld\n",ans);
    return 0;
}
```

<题目跳转> <查看代码>