承载梦想的心愿树 题解

Subtask 1

枚举每个子树，之后枚举子树内所有点，时间复杂度 $O(n^3)$。

卡得好的话甚至可以卡过 Subtask 2.

本 Subtask 的代码为 `tree_sol1.cpp`。

Subtask 2

枚举每个 u, v，并计算 $f(u, v)$。

可以发现，u 和 v 都在 i 的子树中的充要条件，就是 $lca(u, v)$ 在 i 的子树中。

因此，枚举 u, v 之后，将计算所得的结果加到 $lca(u, v)$ 上，再用一个 DFS 统计子树和，即可。

时间复杂度 $O(n^2 \log n)$ 或 $O(n^2)$。

本 Subtask 的标程为 `tree_sol2.cpp`，同时 `tree_sol5.cpp` 是使用了 ST 表求 LCA 的版本。

接下来的内容将包含大量推式子。在开始之前，请先熟背加法和乘法的交换律、分配律、结合律。

转换 $f(u, v)$，消去绝对值和最小值运算

让我们看看 Subtask 3 为我们提供了什么思路。

$b_u = b_v$ 意味着 $b_u \oplus b_v \oplus c = c$。这样，我们就把题中要求的式子转换为了 $f(u, v) = |a_u - a_v| \times min(a_u, a_v) \times c$。而 c 全局统一，所以我们只用专注于 $|a_u - a_v| \times min(a_u, a_v)$，最后输出时再乘上 c 即可。

我们还可以发现如下两个定律：

1. $f(u, u) = 0$，不产生贡献。显而易见。
2. $f(u, v) = f(v, u)$。因为绝对值、最小值运算都不在乎顺序。

因此，我们可以只计算当 $a_u > a_v$ 的贡献，之后再乘 2 即可，原式子可以变为 $(a_u - a_v) \times a_v = \color{red}{a_u a_v} - \color{blue}{a_v^2}$。注意颜色，之后会再次出现。

这个性质并不止用于 Subtask 3，它就是正解的性质。

思考如何维护这个式子。

考虑一种情况，假设这里有 $a_1 < a_2 < a_3 < a_4 < a_5 < a_6$，共 6 个数。

首先，我们已经计算了 a_1, a_2, a_4, a_5, a_6 两两计算贡献之和的结果。

现在，我们要加入 a_3 这个数字。这样，a_3 就会与其余 5 个数产生新贡献。新增的贡献为：

$a_3a_1 - a_1^2 + a_3a_2 - a_2^2 + a_4a_3 - a_3^2 + a_5a_3 - a_3^2 + a_6a_3 - a_3^2$。

> 别忘了上面的两个定律：
>
> 1. $f(u,u)=0$，不产生贡献。显而易见。
> 2. $f(u,v)=f(v,u)$。因为绝对值、最小值运算都不在乎顺序。

我们可以将原来维护好的数分成三类，第一类是小于新加入的数的（上例中共有 2 个），第二类是大于新加入的数的（上例中共有 3 个），第三类是等于新加入的数的（没有意义的）。

分别考虑前两类。

对于第一类，新加入的数会成为更大的 a_u，原来的数会成为更小的 a_v。上例中产生的贡献为：

$$\begin{aligned}&a_3a_1 - a_1^2 + a_3a_2 - a_2^2\\=&(a_3a_1+a_3a_2)-(a_1^2+a_2^2)\\=&a_3(a_1+a_2)-(a_1^2+a_2^2)\end{aligned}$$

而对于第二类，新加入的数会成为更小的 a_v，原来的数会成为更大的 a_u。上例中产生的贡献为：

$$\begin{aligned}&a_4a_3 - a_3^2 + a_5a_3 - a_3^2 + a_6a_3 - a_3^2\\=&(a_4a_3+a_5a_3+a_6a_3)-(a_3^2+a_3^2+a_3^2)\\=&a_3(a_4+a_5+a_6)-(a_3^2+a_3^2+a_3^2)\end{aligned}$$

观察这两个式子，我们针对两个式子的两个颜色（共四个式子），可以发现规律。

设以下四个变量：

1. cnt_1，大于 a_u 的数有多少个？
2. sum_0，小于 a_u 的所有数的总和。
3. sum_1，大于 a_u 的所有数的总和。
4. sq_0，大于 a_u 的所有数**的平方**的总和。先平方，再求和。

则新加入 a_u，会导致贡献增加：

$$a_u sum_0 - sq_0 + a_u sum_1 - cnt_1 a_u^2$$

上面提到的数完全可以用权值线段树维护。为了卡常，请使用常数更小的树状数组代替。

同样，删除 a_u，会导致贡献减少相同的值。由于 $a_u=a_v$ 时不贡献，我们不需要考虑删除和计算贡献的先后顺序，反正都一样。

这样，我们就得出了树上启发式合并的做法，时间复杂度 $O(n\log^2 n)$。

本 Subtask 的标程为 `tree_sol3.cpp`。

分拆二进制的每一位，分别计算

让我们离开 $b_u=b_v$ 的特殊性质，根据上面提到的内容和发现的性质计算答案。

我们可以将 b_u,b_v,c 都拆成有 6 位的二进制数，每一位分别考虑。

规定最低位为第 0 位，设 $b_{u,i}$ 为 b_u 的第 i 位，c_i 为 c 的第 i 位。设当前只考虑第 i 位的情况。

题中给出的函数 $f(u,v)=|a_u-a_v|\times min(a_u,a_v)\times(b_u\oplus b_v\oplus c)$，我们稍微修改一下，变成 $f'(u,v,i)=|a_u-a_v|\times min(a_u,a_v)\times(b_{u,i}\oplus b_{v,i}\oplus c_i)$，可以得到式子，$f(u,v)=\sum_{i=0}^{5}2^i f'(u,v,i)$。

可以发现，当 $b_{u,i}\oplus b_{v,i}\oplus c_i=1$ 时，这一个二进制位才对 $f'(u,v,i)$有贡献。

这很好求，当 c_i 为 1 时，只有 $b_{u,i}=b_{v,i}$ 才能产生贡献；而当 c_i 为 0 时，只有 $b_{u,i}\neq b_{v,i}$ 才能产生贡献。

这样，我们可以开两棵树状数组，一棵树状数组只允许存储 $b_{u,i}=0$ 的数，另一棵只允许存储 $b_{u,i}=1$ 的数。这里的树状数组存储的内容上文有写。

加入点 u 时，只向有可能与 u 产生贡献的树状数组查询。

由于要枚举位，时间复杂度为 $O(6n\log^2 n)$。这个时间复杂度其实相当危险，因此我给了更大的时间限制，并将后 3 个 `Subtask` 调整为了“总和”模式~~防止同学们因为没卡过去造成惨烈挂分而问候出题人~~。

正解代码

略过快读快输部分。

```
const int MAXN=1e5;
const int MAXBIT=6; //最多 6 位
const long long MOD=998244353;
```

一些基本内容：

```
//离散化
int dcnt;
long long data[MAXN+5];
void add_data(long long x)
{
    data[++dcnt]=x;
}
void init_data()
{
    sort(data+1,data+1+dcnt);
    dcnt=unique(data+1,data+1+dcnt)-data-1;
}
int get_dataid(long long x)
{
    return lower_bound(data+1,data+1+dcnt,x)-data;
}
//基本内容
int n;
int a[MAXN+5];
int b[MAXN+5],c;
long long ans[MAXBIT][MAXN+5];  //分开每一位存储
int nowbid; //当前在枚举第几位？
long long nowans;   //当前的贡献总和，需要注意它和字面意思的答案不是一个意思
```

```
int getbit(int x,int pos)   //获取 x 的 pos 位内容
{
    return (x>>pos)&1;
}
int setbit(int x,int pos,int v)
{
    if(getbit(x,pos)!=v)
    {
        x^=(1<<pos);
    }
    return x;
}
```

树状数组:

```
//树状数组
struct Data
{
    int cnt;
    long long sum,sum2; //sum2 即为平方和
    Data()
    {
        cnt=0;
        sum=0ll;
        sum2=0ll;
    }
    Data(int id,int c=1)
    {
        cnt=c;
        sum=data[id]*c%MOD;
        sum2=data[id]*data[id]%MOD*c%MOD;
    }
}tree[2][MAXN+5];   //设置两棵树状数组
Data operator +(Data a,Data b)  //重载运算符以方便后续编写代码
{
    Data res;
    res.cnt=a.cnt+b.cnt;
    res.sum=(a.sum+b.sum)%MOD;
    res.sum2=(a.sum2+b.sum2)%MOD;
    return res;
}
Data operator -(Data a,Data b)
{
    Data res;
    res.cnt=a.cnt-b.cnt;
    res.sum=(((a.sum-b.sum)%MOD)+MOD)%MOD;
    res.sum2=(((a.sum2-b.sum2)%MOD)+MOD)%MOD;
    return res;
}
int lowbit(int x)
{
    return x&-x;
```

```
}
void change(int tid,int x,Data val) //tid 为访问的树状数组编号
{
    while(x<=dcnt)
    {
        tree[tid][x]=tree[tid][x]+val;
        x+=lowbit(x);
    }
}
Data query(int tid,int x)
{
    Data res;
    while(x)
    {
        res=res+tree[tid][x];
        x-=lowbit(x);
    }
    return res;
}
```

定义

接下来是重点：加入或删除一个数，计算当前的答案。

```
void add(int u,int val) //加入/删除 a[u], val=1 代表加入, -1代表删除
{
    int my=getbit(b[u],nowbid); //b[u] 在第 nowbid 位上的值
    int to=my^getbit(c,nowbid)^1;   //只有 b[v] 在第 nowbid 位上值为 to 时，才能和 b[u] 产生贡献
    Data dv=Data(a[u],val); //构造加入/删除的 Data
    Data lower=query(to,a[u]-1),upper=query(to,dcnt)-query(to,a[u]);    //lower 代表小于 a[u] 的，upper 代表大于 a[u] 的
    long long diff=0;   //diff 是贡献的变化量，升高还是减少取决于 val
    diff=(diff+(data[a[u]]*lower.sum%MOD))%MOD;
    diff=(diff+(upper.sum*data[a[u]]%MOD))%MOD;
    diff=(diff-lower.sum2)%MOD;
    diff=(diff-(data[a[u]]*data[a[u]]%MOD*upper.cnt%MOD))%MOD;
    diff=(diff%MOD+MOD)%MOD;
    nowans=(nowans+val*diff+MOD)%MOD;
    change(my,a[u],dv); //执行加入/删除
}
```

接下来是树上启发式合并的板子：

```
struct Edge
{
    int to,next;
}edge[MAXN*2+5];
int edge_cnt;
int head[MAXN+5];
int sz[MAXN+5],hson[MAXN+5];
```

```cpp
void add_edge(int u,int v)
{
    edge[++edge_cnt]=(Edge){v,head[u]};
    head[u]=edge_cnt;
}
void pre(int u,int faa) //预处理重儿子
{
    sz[u]=1;
    for(int i=head[u];i;i=edge[i].next)
    {
        int v=edge[i].to;
        if(v!=faa)
        {
            pre(v,u);
            sz[u]+=sz[v];
            if(sz[hson[u]]<sz[v])
            {
                hson[u]=v;
            }
        }
    }
}
void dfs_add(int u,int faa,int val) //递归加入/删除，不负责直接计算答案
{
    add(u,val);
    for(int i=head[u];i;i=edge[i].next)
    {
        int v=edge[i].to;
        if(v!=faa)
        {
            dfs_add(v,u,val);
        }
    }
}
void dfs(int u,int faa,bool keep)   //计算答案，keep 代表是否保留贡献
{
    if(hson[u])
    {
        for(int i=head[u];i;i=edge[i].next)
        {
            int v=edge[i].to;
            if(v!=faa&&v!=hson[u])
            {
                dfs(v,u,false);
            }
        }
        dfs(hson[u],u,true);
        for(int i=head[u];i;i=edge[i].next)
        {
            int v=edge[i].to;
            if(v!=faa&&v!=hson[u])
            {
                dfs_add(v,u,1);
```

```
            }
        }
    }
    add(u,1);
    ans[nowbid][u]=nowans;
    if(!keep)
    {
        dfs_add(u,faa,-1);
    }
}
```

主函数:

```
int main()
{
    freopen("tree.in","r",stdin);
    #ifndef debug
    freopen("tree.out","w",stdout);
    #endif
    read(n);
    for(int i=1,u,v;i<n;i++)
    {
        read(u);read(v);
        add_edge(u,v);
        add_edge(v,u);
    }
    for(int i=1;i<=n;i++)
    {
        read(a[i]);
        add_data(a[i]);
    }
    init_data();
    for(int i=1;i<=n;i++)
    {
        a[i]=get_dataid(a[i]);
    }
    for(int i=1;i<=n;i++)
    {
        read(b[i]);
    }
    read(c);
    pre(1,0);
    for(nowbid=0;nowbid<MAXBIT;nowbid++)
    {
        dfs(1,0,false);
    }
    for(int i=1;i<=n;i++)
    {
        long long anssum=0;
        for(int j=0;j<MAXBIT;j++)
        {
            anssum=(anssum+(ans[j][i]*(1ll<<j)%MOD))%MOD;    //计算每一位的总和
```

```
        }
        write(anssum*2%MOD);    //别忘了乘 2
        putchar('\n');
    }
    return 0;
}
```

完整标程为 `tree_sol4.cpp` 。

线段树合并

上面的树上启发式合并的做法时间复杂度其实波动很大，当遇到最坏情况（二叉树）时，时间复杂度会被打满为 $O(n\log n\times 6\log n)$，而最好情况（链）时，时间复杂度为 $O(n\times 6\log n)$。

这里介绍线段树合并做法。

程序的基本内容就不介绍了，请自行阅读上方链接的 OI Wiki。这里只介绍合并线段树节点。

假设这里有 $a_1<a_2<a_3<a_4$ 共 4 个数。现在，我们已经维护好了 a_1,a_2 的贡献和 a_3,a_4 的贡献，即将合并这两个线段树上的节点。

经过转换，我们可以得到式子 $a_ua_v-a_v^2$（上文有提到）。

因此，新增了四对贡献为：

$$\begin{aligned}&a_3a_1-a_1^2+a_3a_2-a_2^2+a_4a_1-a_1^2+a_4a_2-a_2^2\\=&a_3a_1+a_3a_2+a_4a_1+a_4a_2-(a_1^2+a_2^2+a_1^2+a_2^2)\\=&(a_3+a_4)(a_1+a_2)-2(a_1^2+a_2^2)\end{aligned}$$

如何维护新增的贡献自然就呼之欲出了。

这是合并代码：

```
void update(int o)
{
    int lc=tree[o].lc,rc=tree[o].rc;    //权值线段树，左节点比右节点更小
    for(int i=0;i<=1;i++)
    {
        tree[o].data[i]=tree[lc].data[i]+tree[rc].data[i];
    }
    tree[o].res=(tree[lc].res+tree[rc].res)%MOD;
    for(int i=0;i<=1;i++)
    {
        int j=i^getbit(c,nowbid)^1; //j 代表能和 i 产生贡献的位的值
        tree[o].res=(tree[o].res+(tree[lc].data[i].sum*tree[rc].data[j].sum%MOD))%MOD;
        tree[o].res=(tree[o].res-
(tree[lc].data[i].sum2*tree[rc].data[j].cnt%MOD)+MOD)%MOD;
    }
}
```

完整标程为 `tree_sol6.cpp` 。

该做法复杂度不明。

> 显然，对于两颗满的线段树，合并操作的复杂度是 $O(n\log n)$ 的。但实际情况下使用的常常是权值线段树，总点数和 n 的规模相差并不大。并且合并时一般不会重复地合并某个线段树，所以我们最终增加的点数大致是 $n\log n$ 级别的。这样，总的复杂度就是$O(n\log n)$ 级别的。当然，在一些情况下，可并堆可能是更好的选择。
>
> —OI Wiki

考虑到线段树比树状数组的常数大很多，线段树合并和树上启发式合并，哪个跑得更快还真不好说。

这是数据生成程序（节选）：

```
#subid 为 Subtask 编号
#ptid 为测试点编号，7号测试点作为样例下发，不会在线评测
if ptid==2 or subid==4:
    tree=Graph.chain(n)
elif ptid==1:
    tree=Graph.tree(n)
elif ptid==2:
    tree=Graph.chain(n)
elif ptid==3 or ptid==7:
    tree=Graph.binary_tree(n)
elif ptid==4:
    tree=Graph.tree(n,0.2,0.6)
elif ptid==5:
    tree=Graph.tree(n,0.4,0.4)
elif ptid==6:
    tree=Graph.tree(n,0.6,0.2)
```

参数的意义可以点击这里查阅。

这样的数据，运行结果如何呢？

笔者在出题时编写了自动对拍程序，这是对拍过程中程序回报的运行时间（节选）：

```
......
Check sub  5 pt  1 with target tree_sol4.exe 1969ms ok 98627 numbers
Check sub  5 pt  2 with target tree_sol4.exe  725ms ok 98249 numbers
Check sub  5 pt  3 with target tree_sol4.exe 2322ms ok 94676 numbers
Check sub  5 pt  4 with target tree_sol4.exe  626ms ok 92496 numbers
Check sub  5 pt  5 with target tree_sol4.exe  678ms ok 96442 numbers
Check sub  5 pt  6 with target tree_sol4.exe  588ms ok 95139 numbers
Check sub  5 pt  7 with target tree_sol4.exe 2552ms ok 99355 numbers
......
Check sub  5 pt  1 with target tree_sol6.exe 1596ms ok 98627 numbers
Check sub  5 pt  2 with target tree_sol6.exe 1420ms ok 98249 numbers
Check sub  5 pt  3 with target tree_sol6.exe 1842ms ok 94676 numbers
Check sub  5 pt  4 with target tree_sol6.exe 1093ms ok 92496 numbers
Check sub  5 pt  5 with target tree_sol6.exe 1738ms ok 96442 numbers
Check sub  5 pt  6 with target tree_sol6.exe 1242ms ok 95139 numbers
Check sub  5 pt  7 with target tree_sol6.exe 1414ms ok 99355 numbers
```

其中 `tree_sol4.exe` 是树上启发式合并，`tree_sol6.exe` 是线段树合并。

同时，oiClass 给出的评测结果显示，对于 Subtask 5，线段树合并做法的时间在 687~1129ms 之间，没有断崖式数据；而树上启发式合并做法，三分之二测试点在 359~425ms 之间，三分之一测试点在 1323~1595ms 之间，中间有断崖。

另外还需要注意的是，线段树合并做法的空间消耗至少 315MB，树上启发式合并做法最多消耗 22MB。

> 使用何种方法完成题目，取决于你的选择。

总结

本题是一道需要推式子和思维的题目。考虑到本题的难点集中在思维而不是模拟，所以放在 T3。

版权信息

题解：广州市铁一中学 邓子君