错题集 2

1□Ancient Distance

链接

题意

给一个有根树，在树上选择 k 个关键点(根必须选)，使得所有点到最近关键祖先(可以是自己)距离的最大值最小。

求出 k 分别为 $1\sim n$ 时答案的和。

题解

考虑贪心，假设已知最小距离为 d□发现每次取树中深度最大的点的 d 级祖先作为关键点最佳。

而选取该节点作为关键点后可以直接删去该关键点的子树，然后继续从新的树中选择深度最大的点。

考虑用树链剖分和线段树维护每个点的 d 级祖先和子树删除操作。

另一方面，发现如果将最小距离设为 d□则每次子树删除操作至少删去 d 的节点，于是操作次数为 $O(\frac nd)$□

考虑倒序枚举每个最小距离，求出该最小距离至少需要的关键点个数，然后更新该关键点个数对应的答案。

总操作数为 $O(\sum_{i=1}^{n-1} \frac ni)=O(n\log n)$□删除子树和查询祖先操作时间复杂度为 $O(\log n)$□于是总时间复杂度为 $O\left(n\log^2 n\right)$□

注意每次枚举完成可以给整棵树打上子树还原标记，不能暴力重建树，否则时间复杂度过高。

```
const int MAXN=2e5+5;
struct Edge{
    int to,next;
}edge[MAXN<<1];
int head[MAXN],edge_cnt;
void Insert(int u,int v){
    edge[++edge_cnt].to=v;
    edge[edge_cnt].next=head[u];
    head[u]=edge_cnt;
}
int d[MAXN],sz[MAXN],f[MAXN],dfs_id[MAXN],dfs_t,inv_id[MAXN];
int h_son[MAXN],mson[MAXN],p[MAXN];
void dfs_1(int u,int fa,int depth){
    sz[u]=1;f[u]=fa;d[u]=depth;mson[u]=0;
    for(int i=head[u];i;i=edge[i].next){
        int v=edge[i].to;
```

```
        dfs_1(v,u,depth+1);
        sz[u]+=sz[v];
        if(sz[v]>mson[u]){
            h_son[u]=v;
            mson[u]=sz[v];
        }
    }
}
void dfs_2(int u,int top){
    dfs_id[u]=++dfs_t;p[u]=top;inv_id[dfs_t]=u;
    if(mson[u])
    dfs_2(h_son[u],top);
    for(int i=head[u];i;i=edge[i].next){
        int v=edge[i].to;
        if(v==h_son[u])
        continue;
        dfs_2(v,v);
    }
}
int lef[MAXN<<2],rig[MAXN<<2],v[MAXN<<2],s[MAXN<<2],lazy[MAXN<<2];
int Max(int x,int y){
    if(!x||!y)return x|y;
    if(d[x]>d[y])return x;
    return y;
}
void build(int k,int L,int R){
    lef[k]=L,rig[k]=R;
    int M=L+R>>1;
    if(L==R)
    return s[k]=v[k]=inv_id[M],void();
    build(k<<1,L,M);build(k<<1|1,M+1,R);
    s[k]=v[k]=Max(v[k<<1],v[k<<1|1]);
}
void push_down(int k){
    if(lazy[k]<0){
        s[k<<1]=s[k<<1|1]=0;
        lazy[k<<1]=lazy[k<<1|1]=lazy[k];
        lazy[k]=0;
    }
    else if(lazy[k]>0){
        s[k<<1]=v[k<<1],s[k<<1|1]=v[k<<1|1];
        lazy[k<<1]=lazy[k<<1|1]=lazy[k];
        lazy[k]=0;
    }
}
void update(int k,int L,int R,int type){
    if(L<=lef[k]&&rig[k]<=R){
        if(type<0)s[k]=0;
        else s[k]=v[k];
        lazy[k]=type;
```

```
        return;
    }
    push_down(k);
    int mid=lef[k]+rig[k]>>1;
    if(mid>=L)update(k<<1,L,R,type);
    if(mid<R)update(k<<1|1,L,R,type);
    s[k]=Max(s[k<<1],s[k<<1|1]);
}
int Find_p(int pos,int dis){
    if(d[pos]<=dis)return 1;
    while(true){
        if(d[pos]-d[p[pos]]>=dis)return inv_id[dfs_id[pos]-dis];
        dis-=d[pos]-d[p[pos]]+1;
        pos=f[p[pos]];
    }
}
int n,ans[MAXN];
int Count(int dis){
    int cnt=0,p;
    while(true){
        if(!s[1])break;
        p=Find_p(s[1],dis);
        update(1,dfs_id[p],dfs_id[p]+sz[p]-1,-1);
        cnt++;
    }
    update(1,1,n,1);
    return cnt;
}
int main()
{
    while(~scanf("%d",&n)){
        edge_cnt=0,dfs_t=0;
        _rep(i,1,n)
        ans[i]=n,head[i]=0;
        _rep(i,2,n)
        Insert(read_int(),i);
        dfs_1(1,0,0);
        dfs_2(1,1);
        build(1,1,n);
        for(int i=n-1;i>=0;i--)
        ans[Count(i)]=i;
        _rep(i,2,n)ans[i]=min(ans[i],ans[i-1]);
        LL sum=0;
        _rep(i,1,n)sum+=ans[i];
        enter(sum);
    }
    return 0;
}
```

2□Greetings Souvenir

链接

题意

给定一棵大小为n的树，且树上的每个节点有一个权值 a_i□现在要给每个节点确定一个数值 b_i□

定义 $val_i=b_i\ast$(i 子树内有多少个点的权值 a_i 等于 b_i)□要求所有点 val_i 的 mex 尽可能大。

题解

乱搞题，首先考虑每个 a□假如有 c_i 个节点权值为 a□对所有 $b_i=a$ 的点，其 val_i 的可能取值只有 $a,2a\cdots c_ia$ 共 c_i 个。

于是所有可能的取值只有 $\sum_{i=1}^n c_i=n$ 个。

在每个点 i 与其可能取值的 val_i 间连一条边，时空间复杂度 $O(n^2)$□为防止爆空间，本题用 bitset 存边。

最后考虑进行二分图匹配，从小到大给所有可能值匹配，如果匹配失败则立即结束，时间复杂度 $O(n^3)$□

考虑乱搞，强行对匈牙利算法使用当前弧优化，于是时间复杂度降为 $O(n^2)$□

```
const int MAXN=2e4+5;
struct Edge{
    int to,next;
}edge[MAXN];
int head[MAXN],edge_cnt;
void Insert(int u,int v){
    edge[++edge_cnt]=Edge{v,head[u]};
    head[u]=edge_cnt;
}
bitset<MAXN>e[MAXN];
int dfn[MAXN],dfr[MAXN],dfs_t;
void p_dfs(int u){
    dfn[u]=++dfs_t;
    for(int i=head[u];i;i=edge[i].next)
    p_dfs(edge[i].to);
    dfr[u]=dfs_t;
}
int n,col[MAXN],cnt[MAXN];
int match[MAXN],cur[MAXN];
bool dfs(int u){
    cur[u]++;
```

```
    for(int &i=cur[u];i<=n;i++){
        if(!e[u][i])continue;
        if(!match[i]||dfs(match[i]))
        return match[i]=u,true;
    }
    return false;
}
int main()
{
    n=read_int();
    _rep(i,2,n)
    Insert(read_int(),i);
    p_dfs(1);
    _rep(i,1,n)
    col[dfn[i]]=read_int();
    _rep(i,1,n){
        _rep(j,dfn[i],dfr[i])
        cnt[col[j]]++;
        _rep(j,1,n){
            if(cnt[j]&&cnt[j]*j<=n)
            e[cnt[j]*j].set(i);
            cnt[j]=0;
        }
    }
    int ans=1;
    _for(i,1,n){
        if(dfs(i))
        ans++;
        else
        break;
    }
    enter(ans);
    return 0;
}
```

3□Interval

链接

题意

给定序列 $a_1,a_2\cdots a_n$□定义函数 $F(l,r)=a_l\text{&} a_{l+1}\text{&} \cdots a_r$□定义不可重集 $S(l,r)=\{l\le a\le b\le r,F(a,b)\}$□

接下来若干询问，每次询问 $|S(l,r)|$□强制在线。

题解 1

首先，对固定的 r□易知 $F(l,r)$ 至多有 $O(\log v)$ 个取值，因为 $F(l,r)-F(l-1,r)=0$ 或 $F(l,r)$ 中的某个不为 0 的位。

线段树可以 $O(\log n)$ 快速计算 $F(l,r)$□考虑对 l 二分，可以 $O(\log^2 n)$ 得到对固定的 r□$F(l,r)$ 的所有可能取值。

接下来考虑维护答案，发现 $S(l,i-1),S(l,i)$ 具有高度相似性，于是考虑使用可持久化线段树维护。

线段树的每个版本 i 维护所有 $S(l,i)(1\le l\le i)$ 的答案，而版本 i 只需要在版本 $i-1$ 的基础上稍微修改即可。

线段树相邻版本的差异来自 $F(l,i)(1\le l\le i)$□$F(l,i)(1\le l\le i)$ 至多有 $O(\log v)$ 个取值，考虑依次插入线段树。

而又有 $S(l,i)\subset S(l-1,i)$□于是考虑差分维护 $S(l,i)(1\le l\le i)$□令 $S(l,i)=\sum_{j=l}^i d_{j,i}$□

对所有相同的值 v□设该值最后一次出现位置为 $last$□于是有 $1 \le l\le last,v\in S(l,i)$□

于是对所有不同的值，考虑维护每个值对 $d_{last,i}$ 的贡献即可，时空间复杂度 $O(n\log n\log v)$

```
const int MAXN=1e5+5,MAXM=30;
int a[MAXN],lef[MAXN<<2],rig[MAXN<<2],s[MAXN<<2];
void push_up(int k){
    s[k]=s[k<<1]&s[k<<1|1];
}
void build(int k,int L,int R){
    lef[k]=L,rig[k]=R;
    int M=L+R>>1;
    if(L==R)return s[k]=a[M],void();
    build(k<<1,L,M);build(k<<1|1,M+1,R);
    push_up(k);
}
int query(int k,int L,int R){
    if(L<=lef[k]&&rig[k]<=R)return s[k];
    int mid=lef[k]+rig[k]>>1;
    if(mid>=R)return query(k<<1,L,R);
    else if(mid<L)return query(k<<1|1,L,R);
    return query(k<<1,L,R)&query(k<<1|1,L,R);
}
struct Node{
    int ch[2],v;
}node[MAXN*MAXM*MAXM];
int root[MAXN],tot;
void update(int &k1,int k2,int lef,int rig,int pos,int v){
    k1=++tot;
    node[k1]=node[k2];
    node[k1].v+=v;
    if(lef==rig)return;
```

```
    int mid=lef+rig>>1;
    if(mid>=pos)update(node[k1].ch[0],node[k2].ch[0],lef,mid,pos,v);
    else update(node[k1].ch[1],node[k2].ch[1],mid+1,rig,pos,v);
}
int query(int k,int lef,int rig,int L,int R){
    if(!k)return 0;
    if(L<=lef&&rig<=R)return node[k].v;
    int mid=lef+rig>>1;
    if(mid>=R)return query(node[k].ch[0],lef,mid,L,R);
    else if(mid<L)return query(node[k].ch[1],mid+1,rig,L,R);
    return
query(node[k].ch[0],lef,mid,L,R)+query(node[k].ch[1],mid+1,rig,L,R);
}
unordered_map<int,int> last;
int main()
{
    int n=read_int();
    _rep(i,1,n)a[i]=read_int();
    build(1,1,n);
    _rep(i,1,n){
        root[i]=root[i-1];
        int cur=-1,lef,rig,mid,pos=i+1;
        while(true){
            lef=1,rig=pos-1,pos=0;
            while(lef<=rig){
                mid=lef+rig>>1;
                if(query(1,mid,i)!=cur){
                    lef=mid+1;
                    pos=mid;
                }
                else
                rig=mid-1;
            }
            if(!pos)
            break;
            cur=query(1,pos,i);
            if(last.count(cur)){
                if(last[cur]<pos){
                    update(root[i],root[i],1,n,last[cur],-1);
                    update(root[i],root[i],1,n,pos,1);
                    last[cur]=pos;
                }
            }
            else{
                update(root[i],root[i],1,n,pos,1);
                last[cur]=pos;
            }
        }
    }
    int q=read_int(),l,r,lastans=0;
    while(q--){
```

```
        l=(read_int()^lastans)%n+1,r=(read_int()^lastans)%n+1;
        if(l>r)swap(l,r);
        enter(lastans=query(root[r],1,n,l,r));
    }
    return 0;
}
```