2020ICPC·南京站

比赛链接

M.Monster Hunter

链接

题意

给定以 1 为根的 n 阶的点权树，假定可以用无费用无限制删去其中 k 个结点，问删除剩余结点的最小费用。

其中，对剩余的所有结点，删除它之前需要删除它的父结点，同时删除它的费用为它的点权 $+$ 当前的所有儿子结点的费用的和。

要求输出 $k=0,1\cdots n$ 时的答案。

题解

易知确认无费用无限制删除的结点后答案已经固定。考虑 $dp(u,k,0/1)$ 表示以结点 u 的子树中有代价删除 k 个结点的最小费用。

其中 $dp(u,k,0)$ 表示无代价删除 u 结点 $dp(u,k,1)$ 表示有代价删除 u 结点。不难得到状态转移方程

$$ dp(u,i+j,0)=\min(dp(u,i+j,0),dp(u,i,0)+\min(dp(v,j,0),dp(v,j,1))) $$

$$ dp(u,i+j,1)=\min(dp(u,i+j,0),dp(u,i,1)+\min(dp(v,j,0),dp(v,j,1)+w_v)) $$

注意优化转移方式，结合代码，不难发现转移过程等效于枚举每个点对的 LCA所以复杂度为 $O(n^2)$

```
const int MAXN=2e3+5;
const LL Inf=1e18;
LL dp[MAXN][MAXN][2];
int head[MAXN],edge_cnt,w[MAXN];
struct Edge{
    int to,next;
}edge[MAXN];
void Insert(int u,int v){
    edge[++edge_cnt]=Edge{v,head[u]};
    head[u]=edge_cnt;
}
int sz[MAXN];
void dfs(int u){
```

```
    dp[u][0][0]=0;
    dp[u][1][1]=w[u];
    sz[u]=1;
    for(int i=head[u];i;i=edge[i].next){
        int v=edge[i].to;
        dfs(v);
        for(int j=sz[u];j>=0;j--)
        _rep(k,1,sz[v]){
dp[u][j+k][0]=min(dp[u][j+k][0],dp[u][j][0]+min(dp[v][k][0],dp[v][k][1]));
dp[u][j+k][1]=min(dp[u][j+k][1],dp[u][j][1]+min(dp[v][k][0],dp[v][k][1]+w[v
]));
        }
        sz[u]+=sz[v];
    }
}
int main()
{
    int T=read_int();
    while(T--){
        int n=read_int();
        _rep(i,1,n)_rep(j,1,n)dp[i][j][0]=dp[i][j][1]=Inf;
        edge_cnt=0;
        _rep(i,1,n)head[i]=0;
        _rep(i,2,n)Insert(read_int(),i);
        _rep(i,1,n)w[i]=read_int();
        dfs(1);
        for(int i=n;i;i--)
        space(min(dp[1][i][0],dp[1][i][1]));
        enter(0);
    }
    return 0;
}
```

J.Just Another Game of Stones

链接

题意

给定一个长度为 n 的序列，接下来 q 个操作。

操作 1 为区间 $\max$

操作 2 为给定数量分别和序列 $[l,r]$ 对应相等的石头堆，以及一个额外的数量为 x 的石头堆，进行石头游戏。

对每个操作 2，询问先手时第一步有几种操作可以保证必胜。

题解

对于一堆异或和为 S 的石头堆，每个堆当且仅当取 $a_i-(S\oplus a_i)$ 个石头才能保证余下石头异或和为 0。

这需要保证 $a_i-(S\oplus a_i)\gt 0$，若 $S=0$，显然无解。

不难发现若 a_i 的二进制表示在 S 的最高位为 1 则 $a_i\gt (S\oplus a_i)$，相反则有 $a_i\lt (S\oplus a_i)$。

于是只需要维护区间的二进制各位中的 1 的个数即可，时间复杂度 $O((n+q)\log n\log v)$。

```
const int MAXN=2e5+5,MAXB=30;
int a[MAXN];
int
lef[MAXN<<2],rig[MAXN<<2],minv[MAXN<<2],minc[MAXN<<2],secv[MAXN<<2],lazy[MA
XN<<2];
struct Node{
    int bitnum[MAXB];
    Node(int v=0){
        _for(i,0,MAXB)
        bitnum[i]=(v>>i)&1;
    }
    Node operator + (const Node &b)const{
        Node c;
        _for(i,0,MAXB)
        c.bitnum[i]=bitnum[i]+b.bitnum[i];
        return c;
    }
    Node operator += (const Node &b){
        _for(i,0,MAXB)
        bitnum[i]+=b.bitnum[i];
        return *this;
    }
}sum[MAXN<<2];
void push_up(int k){
    sum[k]=sum[k<<1]+sum[k<<1|1];
    if(minv[k<<1]==minv[k<<1|1]){
        minv[k]=minv[k<<1];
        minc[k]=minc[k<<1]+minc[k<<1|1];
        secv[k]=min(secv[k<<1],secv[k<<1|1]);
    }
    else if(minv[k<<1]<minv[k<<1|1]){
        minv[k]=minv[k<<1];
        minc[k]=minc[k<<1];
        secv[k]=min(secv[k<<1],minv[k<<1|1]);
    }
    else{
        minv[k]=minv[k<<1|1];
        minc[k]=minc[k<<1|1];
```

```
        secv[k]=min(minv[k<<1],secv[k<<1|1]);
    }
}
void push_tag(int k,int v){
    if(v<=minv[k])return;
    _for(i,0,MAXB){
        if((minv[k]>>i)&1)
        sum[k].bitnum[i]-=minc[k];
        if((v>>i)&1)
        sum[k].bitnum[i]+=minc[k];
    }
    minv[k]=lazy[k]=v;
}
void push_down(int k){
    if(~lazy[k]){
        push_tag(k<<1,lazy[k]);
        push_tag(k<<1|1,lazy[k]);
        lazy[k]=-1;
    }
}
void build(int k,int L,int R){
    lef[k]=L,rig[k]=R,lazy[k]=-1;
    int M=L+R>>1;
    if(L==R){
        sum[k]=Node(a[M]);
        minv[k]=a[M];
        secv[k]=1<<MAXB;
        minc[k]=1;
        return;
    }
    build(k<<1,L,M);
    build(k<<1|1,M+1,R);
    push_up(k);
}
void update(int k,int L,int R,int v){
    if(minv[k]>=v)return;
    if(L<=lef[k]&&rig[k]<=R&&secv[k]>v)
    return push_tag(k,v);
    push_down(k);
    int mid=lef[k]+rig[k]>>1;
    if(mid>=L)update(k<<1,L,R,v);
    if(mid<R)update(k<<1|1,L,R,v);
    push_up(k);
}
Node query_sum(int k,int L,int R){
    if(L<=lef[k]&&rig[k]<=R)return sum[k];
    push_down(k);
    int mid=lef[k]+rig[k]>>1;
    Node s=Node();
    if(mid>=L)s+=query_sum(k<<1,L,R);
```

```
    if(mid<R)s+=query_sum(k<<1|1,L,R);
    return s;
}
int main()
{
    int n=read_int(),q=read_int();
    _rep(i,1,n)a[i]=read_int();
    build(1,1,n);
    while(q--){
        int op=read_int(),l=read_int(),r=read_int(),x=read_int();
        if(op==1)
        update(1,l,r,x);
        else{
            Node temp=query_sum(1,l,r)+Node(x);
            int maxb=-1;
            for(int i=MAXB-1;i>=0;i--){
                if(temp.bitnum[i]&1){
                    maxb=i;
                    break;
                }
            }
            if(maxb==-1)
            enter(0);
            else
            enter(temp.bitnum[maxb]);
        }
    }
    return 0;
}
```