Deltix Round, Summer 2021 (Div. 1 + Div. 2)

比赛链接

E. Equilibrium

题意

给定两个序列 $\{A\},\{B\}$，每次询问一个区间 $[l,r]$。

每次操作可以在 $[l,r]$ 中选择若干对位置 $(a_1,b_1),(a_2,b_2)\cdots (a_k,b_k)$，满足 $a_1\lt b_1\lt a_2\lt b_2\lt\cdots \lt a_k\lt b_k$。

然后对每对位置 (i,j)，令 $A_i\gets A_i+1,B_j\gets B_j+1$，问使得 $A_i=B_i(l\le i\le r)$ 的最小操作数。

题解

令 $C_i=B_i-A_i$，于是上述操作的每对位置 (i,j) 影响为 $C_i\gets C_i-1,C_j\gets C_j+1$，最终需要使得 $C_i=0$。

不难发现 $\sum_{i=l}^r C_i\neq 0$ 时必然无解，因为每次操作不改变 C_i 之和。

另外 $C_i=0$ 等价于前缀和等于 0，然后上述操作的每次位置 (i,j) 对前缀和的影响为 $i\sim j-1$ 的前缀和减一，其余不变。

于是 $C[l,r]$ 的最小前缀和小于 0 时也必然无解。然后每次操作可以等效为选择 $[l,r-1]$ 的若干段前缀和减一，于是最小操作等于最大前缀和。

考虑线段树维护，时间复杂度 $O((n+q)\log n)$。

```
const int MAXN=1e5+5;
struct Node{
    LL sum,max_pre,min_pre;
    Node(int v=0){
        sum=max_pre=min_pre=v;
    }
    Node operator + (const Node &b)const{
        Node c;
        c.sum=sum+b.sum;
        c.max_pre=max(max_pre,sum+b.max_pre);
        c.min_pre=min(min_pre,sum+b.min_pre);
        return c;
    }
}s[MAXN<<2];
int a[MAXN],lef[MAXN<<2],rig[MAXN<<2];
void build(int k,int L,int R){
```

```
    lef[k]=L,rig[k]=R;
    int M=L+R>>1;
    if(L==R){
        s[k]=Node(a[M]);
        return;
    }
    build(k<<1,L,M);
    build(k<<1|1,M+1,R);
    s[k]=s[k<<1]+s[k<<1|1];
}
Node 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);
    else
    return query(k<<1,L,R)+query(k<<1|1,L,R);
}
int main(){
    int n=read_int(),q=read_int();
    _rep(i,1,n)
    a[i]=-read_int();
    _rep(i,1,n)
    a[i]+=read_int();
    build(1,1,n);
    while(q--){
        int l=read_int(),r=read_int();
        Node ans=query(1,l,r);
        if(ans.sum!=0||ans.min_pre<0)
        puts("-1");
        else
        enter(ans.max_pre);
    }
    return 0;
}
```

F. Sports Betting

题意

给定 n 个队伍的能力值，队伍间两两进行一场比赛，队伍 i 战胜队伍 j 的概率为 $\frac{a_i}{a_i+a_j}$。

定义间接战胜：如果队伍 i 战胜队伍 b_1，b_1 战胜 $b_2\cdots b_k$ 战胜 j，则称 i 间接战胜 j。

称一个队伍为冠军队伍当且仅当该队直接战胜或间接战胜所有其他队伍。问冠军队伍的期望个数。

题解

如果队伍 i 战胜队伍 j，则从点 i 向点 j 连一条边。

设 $U=\{1,2\cdots n\}$，不难发现，假设冠军队伍集合为 S 当且仅当 S 为强连通分量且 S 与 $U-S$ 的所有边方向为 $S\to U-S$。

设 $G(S,T)$ 表示 S 与 T 的所有边方向为 $S\to T$ 的概率，$f(S)$ 为 S 构成强连通分量的概率，根据容斥定理，有

$$ f(S)=1-\prod_{T\subset S,T\neq \emptyset}G(T,S-T)f(T) $$

最终答案为

$$ \sum_{S\subseteq U}G(S,U-S)f(S)|S| $$

考虑如何计算 $G(S,T)$，有

$$ G(S,T)=\prod_{i\in S}\prod_{j\in T}\frac{a_i}{a_i+a_j} $$

显然可以暴力 $O(n^2)$ 计算单个 $G(S,T)$，预处理 $F(i,T)=\prod_{j\in T}\frac{a_i}{a_i+a_j}$，则 $G(S,T)=\prod_{i\in S}F(i,T)$ 可以 $O(n)$ 计算。

进一步，考虑将前 $\frac n2$ 个点染黑，其余点染白，记黑点集为 U_1，白点集为 U_2，于是有

$$ G(S,T)=G(S\cap U_1,T\cap U_1)G(S\cap U_1,T\cap U_2)G(S\cap U_2,T\cap U_1)G(S\cap U_2,T\cap U_2) $$

考虑 $O(n2^n)$ 预处理出下述四种情况

$$ G_{0/1,0/1}(S,T)=G(S,T)(S\subseteq U_1/S\subseteq U_2,T\subseteq U_1/T\subseteq U_2) $$

于是可以 $O(1)$ 计算 $G(S,T)$，总时间复杂度为 $O\left(3^n\right)$。

```
const int MAXN=15,MAXB=8,mod=1e9+7;
int quick_pow(int n,int k){
    int ans=1;
    while(k){
        if(k&1)ans=1LL*ans*n%mod;
        n=1LL*n*n%mod;
        k>>=1;
    }
    return ans;
}
int f[1<<MAXN],a[MAXN],p[MAXN][MAXN],g[4][1<<MAXB][1<<MAXB];
void cal(int p1,int n1,int p2,int n2,int g[1<<MAXB][1<<MAXB]){
    static int temp[MAXB][1<<MAXB];
    int s1=1<<n1,s2=1<<n2;
    _for(i,0,n1){
```

```
        _for(j,0,s2){
            temp[i][j]=1;
            _for(k,0,n2){
                if(j&(1<<k))
                temp[i][j]=1LL*temp[i][j]*p[i+p1][k+p2]%mod;
            }
        }
    }
    _for(i,0,s1)_for(j,0,s2){
        g[i][j]=1;
        _for(k,0,n1){
            if(i&(1<<k))
            g[i][j]=1LL*g[i][j]*temp[k][j]%mod;
        }
    }
}
int cal2(int i,int j,int m){
    int mk=(1<<m)-1;
    LL t=1LL*g[0][i&mk][j&mk]*g[1][i&mk][j>>m]%mod;
    t=t*g[2][i>>m][j&mk]%mod*g[3][i>>m][j>>m]%mod;
    return t;
}
int cal3(int i){
    int ans=0;
    while(i){
        ans+=i&1;
        i>>=1;
    }
    return ans;
}
int main(){
    int n=read_int();
    if(n==1){
        puts("1");
        return 0;
    }
    _for(i,0,n)
    a[i]=read_int();
    _for(i,0,n){
        _for(j,0,n)
        p[i][j]=1LL*a[i]*quick_pow(a[i]+a[j],mod-2)%mod;
    }
    int m=n/2;
    cal(0,m,0,m,g[0]);
    cal(0,m,m,n-m,g[1]);
    cal(m,n-m,0,m,g[2]);
    cal(m,n-m,m,n-m,g[3]);
    int S=(1<<n)-1,ans=0;
    _rep(i,1,S){
        f[i]=1;
```

```
        for(int j=(i-1)&i;j!=i;j=(j-1)&i)
        f[i]=(f[i]-1LL*f[j]*cal2(j,i^j,m))%mod;
        ans=(ans+1LL*f[i]*cal2(i,i^S,m)%mod*cal3(i))%mod;
    }
    enter((ans+mod)%mod);
    return 0;
}
```

G. Gates to Another World

题意

给定编号为 $0\sim 2^n-1$ 的点，u 与 v 连边当且仅当 u,v 的编号二进制表示仅有一位不同。

接下来 m 个操作，操作分两种：

1. 删去区间 $[l,r]$ 的所有点
2. 询问 a,b 两点是否连通

数据保证每个点最多被删除一次，且每次询问的两点一定存在。

题解

首先考虑一种暴力做法，给第 i 个点一个权值 a_i，表示第 i 个点在第 a_i 次操作后被删除，如果第 i 个点直到最后也没被删除则 $a_i=m+1$。

定义每个边的权值为与该边相邻的两个点权值的最小值。然后逆序处理操作，根据边权从大到小加边同时处理询问，于是并查集维护连通性即可。

时间复杂度 $O(\text{Na}\times n2^n)$，其中 $O(\text{Na})$ 为并查集复杂度。

不难发现 $[t2^k,(t+1)2^k-1]$ 的所有点如果 a_i 值相等，那他们一定被加入后就立刻连通，于是可以当作一个点处理。

考虑线段树优化建图，把每个删点 $[l,r]$ 分为 $O(n)$ 个二的幂次区间，这样最多只有 $O(nm)$ 个点。

再考虑连边，可以分治处理，假设当前处理到区间 $[t2^k,(t+1)2^k-1]$，可以将点分为左区间的点和右区间的点，只考虑左右区间之间的连边。

然后不难发现只有左右区间编号差 2^{k-1} 的点正好连一条边，于是可以双指针处理连边，然后继续递归左右区间连边即可。

不难发现边数是 $O(n^2m)$，于是总时间复杂度 $O(\text{Na}\times n^2m)$。

进一步考虑优化，发现 $[t2^k,t2^k+x],[t2^k-1-x,t2^k-1](x\lt 2^k)$ 的所有点也是连通的，于是可以猫树优化建图，点数最多 $O(m)$。

连边也用分治加双指针处理，不过注意如果存在跨左右区间的点需要同时分配到左右区间递归。

最终边数是 $O(nm)$，于是总时间复杂度 $O(\text{Na}\times nm)$。

```
const int MAXL=50,MAXM=5e4+5;
struct opt{
    LL a,b;
    int t;
    bool operator < (const opt &o)const{
        return a<o.a;
    }
};
vector<opt> upd,nodes;
void Insert(opt node,LL vl,LL vr){
    if(node.a==node.b)
    nodes.push_back(node);
    else{
        LL vm=vl+vr>>1;
        if(node.a<=vm&&node.b>vm){
            nodes.push_back(opt{node.a,vm,node.t});
            nodes.push_back(opt{vm+1,node.b,node.t});
        }
        else if(node.b<=vm)
        Insert(node,vl,vm);
        else
        Insert(node,vm+1,vr);
    }
}
vector<pair<int,int> > edges[MAXM];
void build(int ql,int qr,LL vl,LL vr){
    if(ql==qr)
    return;
    LL vm=vl+vr>>1,len=vm-vl+1;
    int qm1=ql,qm2;
    while(qm1<qr&&nodes[qm1+1].a<=vm)qm1++;
    qm2=qm1+(nodes[qm1].b>vm?0:1);
    int pos1=ql,pos2=qm2;
    while(pos1<=qm1&&pos2<=qr){
        edges[min(nodes[pos1].t,nodes[pos2].t)].emplace_back(pos1,pos2);
        if(nodes[pos1].b+len<nodes[pos2].b)
        pos1++;
        else if(nodes[pos1].b+len>nodes[pos2].b)
        pos2++;
        else
        pos1++,pos2++;
    }
    build(ql,qm1,vl,vm);
    build(qm2,qr,vm+1,vr);
}
int p[MAXM<<2];
int Find(int x){
    return p[x]==x?x:p[x]=Find(p[x]);
}
```

```
pair<LL,LL> query[MAXM];
vector<LL> mp;
char buf[MAXL];
int main(){
    int n=read_int(),m=read_int();
    LL maxv=(1LL<<n)-1;
    _for(i,0,m){
        scanf("%s",buf);
        LL l=read_LL(),r=read_LL();
        if(buf[0]=='b'){
            upd.push_back(opt{l,r,i});
            query[i]=make_pair(-1LL,-1LL);
        }
        else
        query[i]=make_pair(l,r);
    }
    query[m]=make_pair(-1LL,-1LL);
    sort(upd.begin(),upd.end());
    LL pos1=0;
    int pos2=0;
    while(pos1<=maxv){
        if(pos2<upd.size()){
            if(upd[pos2].a==pos1){
                Insert(upd[pos2],0,maxv);
                pos1=upd[pos2++].b+1;
            }
            else{
                Insert(opt{pos1,upd[pos2].a-1,m},0,maxv);
                pos1=upd[pos2].a;
            }
        }
        else{
            Insert(opt{pos1,maxv,m},0,maxv);
            pos1=maxv+1;
        }
    }
    build(0,nodes.size()-1,0,maxv);
    _for(i,0,nodes.size()){
        p[i]=i;
        mp.push_back(nodes[i].b);
    }
    stack<int> ans;
    for(int i=m;i>=0;i--){
        for(pair<int,int> pr:edges[i]){
            int x=Find(pr.first),y=Find(pr.second);
            if(x!=y)
            p[x]=y;
        }
        if(query[i].first!=-1){
            int p1=lower_bound(mp.begin(),mp.end(),query[i].first)-
mp.begin();
```

```
            int p2=lower_bound(mp.begin(),mp.end(),query[i].second)-
mp.begin();
            p1=Find(p1);
            p2=Find(p2);
            if(p1==p2)
            ans.push(1);
            else
            ans.push(0);
        }
    }
    while(!ans.empty()){
        enter(ans.top());
        ans.pop();
    }
    return 0;
}
```