比赛链接

补题情况

题目	蒋贤蒙	王赵安	王智彪
B	2	0	0
D	2	0	0
E	2	0	1
G	0	0	0
I	0	0	0
J	2	0	1
L	2	0	1

题解

B. Bipartite Blanket

题意

给定一个二分图，左部有 n 个点，右部有 m 个点和一个下界 t，每个点有一个点权。

要求选出一个点集，该点集的点权和不小于 t 且该点集是二分图匹配的一部分。问有多少选择方案。

题解

首先给定霍尔婚配定理：对点集 S，定义 $f(S)$ 表示所有与 S 相邻的点集，则 S 可以被匹配的充要条件是 $\forall T(T\subseteq S\to |T|\le |f(T)|)$。

于是可以 $O(n2^n)$ 计算出左部、右部的所有合法点集。

设 A 是左部的一个合法点集，B 是右部的一个合法点集，则有 $A\cup B$ 是二分图匹配的一部分，证明如下：

由于 A 存在匹配，所以可以令 A 中每个点向他的匹配点连一条有向边，同时对 B 进行相同操作。

得到一个有向图，由于所有点出度不超过 1，因此可以有向图可以分割成若干个环和路径。对于每个环，直接两两匹配即可。

对于每个路径，易知路径终点不属于 A,B，如果路径长度是奇数，则舍去路径终点，其他点两两匹配。

如果路径长度是偶数，则加入路径终点进行两两匹配。

于是可以记录左部所有合法点集的点权以及右部所有合法点集的点权，双指针统计答案。

```
const int MAXN=21;
int g1[MAXN],g2[MAXN],a[MAXN],b[MAXN];
```

```
int s[1<<MAXN],vis[1<<MAXN],bt[1<<MAXN],lg2[1<<MAXN];
bool ok[1<<MAXN];
#define lowbit(x) (x&(-x))
vector<int> solve(int *a,int *g,int n){
    vector<int> ans;
    int S=1<<n;
    ok[0]=1;
    ans.push_back(0);
    _for(i,1,S){
        ok[i]=1;
        s[i]=0;
        _for(j,0,n){
            if(i&(1<<j)){
                ok[i]&=ok[i^(1<<j)];
                s[i]+=a[j];
            }
        }
        vis[i]=vis[i^lowbit(i)]|g[lg2[lowbit(i)]];
        ok[i]&=(bt[vis[i]]>=bt[i]);
        if(ok[i]){
            ans.push_back(s[i]);
            //printf("%d %d\n",i,s[i]);
        }
    }
    return ans;
}
char buf[MAXN];
int main(){
    int n=read_int(),m=read_int();
    _for(i,0,n){
        scanf("%s",buf);
        _for(j,0,m){
            if(buf[j]=='1'){
                g1[i]|=1<<j;
                g2[j]|=1<<i;
            }
        }
    }
    _for(i,0,n)a[i]=read_int();
    _for(i,0,m)b[i]=read_int();
    int S=1<<MAXN;
    _for(i,1,S)
    bt[i]=bt[i^lowbit(i)]+1;
    _for(i,0,MAXN)
    lg2[1<<i]=i;
    vector<int> s1=solve(a,g1,n);
    vector<int> s2=solve(b,g2,m);
    sort(s1.begin(),s1.end());
    sort(s2.begin(),s2.end(),greater<int>());
    int t=read_int(),pos=0;
```

```
    LL ans=0;
    for(int v:s1){
        while(pos<s2.size()&&v+s2[pos]>=t)pos++;
        ans+=pos;
    }
    enter(ans);
    return 0;
}
```

D. Dancing Disks

题意

给定一个 6×6 的栈，每次可以从栈 (r,c) 顶部取若干个元素放入栈 $(r+1,c)$ 或栈 $(r,c+1)$ 的顶部。

初始时所有元素都在栈 $(1,1)$，自底向上分别是 $a_1,a_2\cdots a_n$，其中 $a_1,a_2\cdots a_n$ 构成 $1\sim n$ 的一个排列。

要求构造一个方案将所有元素转移到栈 $(6,6)$，且栈 $(6,6)$ 自底向上分别是 $n,n-1\cdots 1$，其中 $n\le 40000$。

题解

设 $f(r,c)$ 表示从 $f(r,c)$ 出发至少可以将多少个元素转移到 $(6,6)$ 且最终 $(6,6)$ 上元素有序。

显然 $f(6,6)=1$，而对 $f(6,5)$，假设元素自底向上为 $1,2$，考虑先移动 2，再移动 1。假设元素自底向上为 $2,1$，则同时移动 $2,1$。

如果有三个元素自底向上为 $3,1,2$，显然没有合法方案，所以 $f(6,5)=2$，同理 $f(5,6)=2$。

对其他情况，可以先将栈 (r,c) 的元素从小到大排序，然后依次分配 $f(i,j)$ 个元素给栈 (i,j) 处理 $(i\ge r,j\ge c,i+j\gt r+c)$，于是有

$$ f(r,c)=\sum_{i\ge r,j\ge c,i+j\gt r+c}f(i,j) $$

最后有 $f(1,1)\ge 40000$，所以一定存在合法方案。方案构造按上述方法即可。

每个元素最多移动 10 步，所以最多出现在 10 个栈，所以总元素个数为 $O(10n)$，算上排序复杂度，总时间复杂度为 $O(10n\log n)$。

```
const int MAXN=4e4+5,N=6;
int dp[N+1][N+1];
int dfs1(int r,int c){
    if(dp[r][c])return dp[r][c];
    _rep(i,r,N)_rep(j,c,N){
        if(i!=r||j!=c)
        dp[r][c]+=dfs1(i,j);
```

```
    }
    return dp[r][c];
}
int a[MAXN],temp1[MAXN],temp2[MAXN],b[N+1][N+1];
pair<int,int> p[MAXN];
stack<int> st[N+1][N+1];
void Move(int r1,int c1,int r2,int c2,int cnt){
    while(r1<r2){
        printf("%d %d D %d\n",r1,c1,cnt);
        r1++;
    }
    while(c1<c2){
        printf("%d %d R %d\n",r1,c1,cnt);
        c1++;
    }
}
void dfs2(int r,int c,int n){
    if(n==0||(r==6&&c==6))return;
    _for(i,0,n){
        temp1[i]=temp2[i]=st[r][c].top();
        st[r][c].pop();
    }
    if(r+c==11){
        if(n==1)
        Move(r,c,6,6,1);
        else{
            if(temp1[0]<temp1[1]){
                Move(r,c,6,6,2);
                st[6][6].push(temp1[1]);
                st[6][6].push(temp1[0]);
            }
            else{
                Move(r,c,6,6,1);
                Move(r,c,6,6,1);
                st[6][6].push(temp1[0]);
                st[6][6].push(temp1[1]);
            }
        }
        return;
    }
    sort(temp1,temp1+n);
    int lpos=0;
    _rep(i,r,N)_rep(j,c,N){
        if(i==r&&j==c)continue;
        int rpos=min(lpos+dp[i][j],n);
        _for(k,lpos,rpos)
        p[temp1[k]]=make_pair(i,j);
        b[i][j]=rpos-lpos;
        lpos=rpos;
    }
```

```
    _for(i,0,n){
        int r2=p[temp2[i]].first,c2=p[temp2[i]].second;
        Move(r,c,r2,c2,1);
        st[r2][c2].push(temp2[i]);
    }
    for(int i=N;i>=r;i--)for(int j=N;j>=c;j--){
        if(i==r&&j==c)continue;
        dfs2(i,j,b[i][j]);
    }
}
int main(){
    int n=read_int();
    _rep(i,1,n)
    st[1][1].push(read_int());
    dp[6][6]=1,dp[6][5]=dp[5][6]=2;
    dfs1(1,1);
    dfs2(1,1,n);
//    while(!st[6][6].empty()){
//        space(st[6][6].top());
//        st[6][6].pop();
//    }
    return 0;
}
```

J. Jazz Journey

题意

给定固定路线 $a_1,a_2\cdots a_d(1\le a_i\le n)$需要坐飞机依次经过各点。

接下来给定若 m 种机票，每张机票有一个起点 u 和终点 v 以及费用 w。

机票分为单程票和双程票，其中每张单程票只能实现一次 $u\to v$双程票可以实现一次 $u\to v$ 和一次 $v\to u$。

题解

由于只有 $d-1$ 次移动，所以不妨将所有移动按起点终点分类，其中 $u\to v,v\to u$ 视为同类。

例如，对路径 12313121，可以得到如下几类：

1. $1\to 2,1\to 2,2\to 1$
2. $2\to 3$
3. $3\to 1,1\to 3,3\to 1$

然后对每类路径，单独考虑费用。不妨设某类路径的起点终点为 u,v且 $u\to v$ 的双程票是最划算的。

考虑扫描该类型的所有移动，一旦出现 $u\to v,v\to u$ 则立刻购买 $u\to v$ 双程票，否则先保留。

最后剩余的移动一定是形如 $v\to u,v\to u\cdots v\to u,u\to v\cdots u\to v$这个时候再考虑买 $v\to u$ 的双程票划算还是单程票划算即可。

最后不要忘了把双程票当单程票用比单程票便宜的情况。时间复杂度 $O\left(d\log d\right)$

```
const LL inf=2e9;
const int MAXN=3e5+5;
map<pair<int,int>,int>mp;
struct Node{
    int u,v;
    int type;
    bool operator < (const Node &b)const{
        if(u!=b.u)
        return u<b.u;
        else if(v!=b.v)
        return v<b.v;
        else
        return type<b.type;
    }
};
map<Node,LL> cost;
vector<int> c[MAXN];
int a[MAXN];
LL ans;
void solve(int id,int u,int v){
    LL w1,w2,w3,w4;
    int cnt[2]={0,0};
    w1=cost.count(Node{u,v,0})?cost[Node{u,v,0}]:inf;
    w2=cost.count(Node{u,v,1})?cost[Node{u,v,1}]:inf;
    w3=cost.count(Node{v,u,0})?cost[Node{v,u,0}]:inf;
    w4=cost.count(Node{v,u,1})?cost[Node{v,u,1}]:inf;
    w1=min(w1,w2);
    w3=min(w3,w4);
    for(int t:c[id]){
        if(t==0){
            if(cnt[1]){
                if(w4==min(w2,w4)&&w4<w3+w1){
                    ans+=w4;
                    cnt[1]--;
                }
                else
                cnt[0]++;
            }
            else
            cnt[0]++;
        }
        else{
            if(cnt[0]){
                if(w2==min(w2,w4)&&w2<w1+w3){
                    ans+=w2;
```

```
                    cnt[0]--;
                }
                else
                cnt[1]++;
            }
            else
            cnt[1]++;
        }
    }
    int tt=min(cnt[0],cnt[1]);
    if(w2!=min(w2,w4)&&w2<w1+w3){
        ans+=1LL*tt*w2;
        cnt[0]-=tt;
        cnt[1]-=tt;
    }
    else if(w4!=min(w2,w4)&&w4<w3+w1){
        ans+=1LL*tt*w4;
        cnt[0]-=tt;
        cnt[1]-=tt;
    }
    ans+=1LL*cnt[0]*w1+1LL*cnt[1]*w3;
}
int main(){
    int n=read_int(),d=read_int();
    _for(i,0,d)a[i]=read_int();
    _for(i,1,d){
        int x=a[i-1],y=a[i];
        if(x>y)swap(x,y);
        if(mp.find(make_pair(x,y))==mp.end()){
            int t=mp.size();
            mp[make_pair(x,y)]=t;
        }
        c[mp[make_pair(x,y)]].push_back(a[i-1]>a[i]);
    }
    int m=read_int();
    _for(i,0,m){
        int u=read_int(),v=read_int();
        char type=get_char();
        LL w=read_int();
        Node t;
        if(type=='0')
        t=Node{u,v,0};
        else
        t=Node{u,v,1};
        if(cost.find(t)==cost.end())
        cost[t]=inf;
        cost[t]=min(cost[t],w);
    }
    for(map<pair<int,int>,int>::iterator
iter=mp.begin();iter!=mp.end();iter++){
        pair<int,int> temp=iter->first;
```

```
        solve(iter->second,temp.first,temp.second);
    }
    enter(ans);
    return 0;
}
```

L. Lost Logic

题意

给定 n 个变量以及三组解，要求构造若干条形如 $(!)x_i\to (!)x_j$ 的限制，使得方程只有给定的三组解。

题解

将所有变量在三组解中的取值分为 8 类，分别是 $(0,0,0),(0,0,1)\cdots (1,1,1)$。

首先对于 $(0,0,0)$ 类变量，构造约束 $x\to !x$。类似处理 $(1,1,1)$ 类变量。

对于同属于 $(0,0,1)$ 类变量 x,y。构造约束 $x\to y,!x\to !y$。

于是现在自由变量最多只有 6 个。然后处理对偶的自由变量，如 $x\in (0,0,1),y\in (1,1,0)$。构造约束 $x\to !y,!x\to y$。

现在只剩下最多 3 个自由变量了。不难发现如果有 3 个自由变量则无法进一步构造约束。

当有两个自由变量时，不妨考虑 $x\in (0,0,1),y\in (1,0,0)$ 的情况，只要去除 $x=1,y=1$ 的情况即可，可以构造约束 $x\to !y$。其他情况类似。

最后只剩下不超过一个自由变量，显然该变量任意取值后所有其他变量都取值固定，正好满足三组解。

```
const int MAXN=55;
int a[3][MAXN],b[2][2];
vector<int> c[2][2][2];
vector<pair<int,int> >ans;
int main(){
    int n=read_int();
    _for(i,0,3)_rep(j,1,n)
    a[i][j]=read_int();
    _rep(i,1,n)
    c[a[0][i]][a[1][i]][a[2][i]].push_back(i);
    vector<int> var;
    _for(i,0,2)_for(j,0,2)_for(k,0,2){
        if(c[i][j][k].size()==0)continue;
        if(i==j&&i==k){
            for(int t:c[i][j][k]){
                if(i==0)
                ans.push_back(make_pair(t,t+n));
                else
```

```
                ans.push_back(make_pair(t+n,t));
            }
        }
        else{
            int head=*c[i][j][k].begin();
            for(int t:c[i][j][k]){
                if(t==head)continue;
                ans.push_back(make_pair(head,t));
                ans.push_back(make_pair(head+n,t+n));
            }
        }
    }
    _for(i,0,2)_for(j,0,2){
        if(i==0&&j==0)continue;
        if(c[0][i][j].size()&&c[1][!i][!j].size()){
            int t1=*c[0][i][j].begin();
            int t2=*c[1][!i][!j].begin();
            ans.push_back(make_pair(t1,t2+n));
            ans.push_back(make_pair(t1+n,t2));
        }
        if(c[0][i][j].size())
        var.push_back(*c[0][i][j].begin());
        else if(c[1][!i][!j].size())
        var.push_back(*c[1][!i][!j].begin());
    }
    if(var.size()<3){
        if(var.size()==2){
            int t1=var[0],t2=var[1];
            int d1=2-a[0][t1]-a[1][t1]-a[2][t1],d2=2-a[0][t2]-a[1][t2]-
a[2][t2];
            ans.push_back(make_pair(t1+(1-d1)*n,t2+d2*n));
        }
        enter(ans.size());
        for(pair<int,int> t:ans){
            if(t.first>n){
                t.first-=n;
                putchar('!');
            }
            printf("x%d -> ",t.first);
            if(t.second>n){
                t.second-=n;
                putchar('!');
            }
            printf("x%d\n",t.second);
        }
    }
    else
    puts("-1");
    return 0;
}
```