A. 金乌初上瑞云开

将总电子数按照填入能级顺序不断减去对应能级的最大电子数，直到剩余电子数不足以填满下一个能级为止。

```
#include<bits/stdc++.h>
using namespace std;
int n;
int s[25]={0,1,2,2,3,3,4,3,4,5,4,5,6,4,5,6,7,5,6,7};
char ch[25]=
{'?','s','s','p','s','p','s','d','p','s','d','p','s','f','d','p','s','f','d','p'};
int main(){
    ios::sync_with_stdio(0);
    cin.tie(0);cout.tie(0);
    cin>>n;
    for(int i=1;n>=1;i++){
        int num=2;
        if(ch[i]=='p')num=6;
        if(ch[i]=='d')num=10;
        if(ch[i]=='f')num=14;
        int mn=min(n,num);
        n-=mn;
        cout<<s[i]<<ch[i]<<mn<<' ';
    }
    return 0;
}
```

B. 爆竹声中旧岁裁

45pts

考虑枚举两个电梯的位置，再计算时间和，取最小。

计算价值和可以考虑等差数列优化（相邻的楼层差 1），复杂度 $O(n^2)$。

100pts

一种可行的方案是对于较小的数据枚举来找规律（或者直接瞪眼瞪出规律，再去反证）。

楼层单独的时间仅依赖于最近的电梯，那么可以考虑把 n 分成两份，分别由两个电梯管辖。

此时每份最优解显然是电梯在其中点的时候：电梯上移或下移都会使总时间更大。

本题要求字典序最小的方案，而且偶数楼层的中点不是整数层，所以向下取整。

对于初学者，奇偶不同的 n 取中点可以用不同的 `if` 区分：

```
#include <bits/stdc++.h>
#define ll long long
using namespace std;
ll n;
void work(){
    cin>>n;
```

```
    if(n==1) cout<<"1 1";
    else if(n%4==0) cout<<n/4<<' '<<n/2+n/4;
    else if(n%2==0) cout<<n/4+1<<" "<<n/2+n/4+1;
    else {
        ll m=n/2;
        if(m%2==0) cout<<m/2<<' '<<m+m/2+1;
        else cout<<m/2+1<<' '<<m+(n-m)/2;
    }
}
int main(){
    work();
    return 0;
}
```

C. 春入千门风送暖

20pts

留给暴力枚举的朋友们。

m=9

这个特殊性质经过数学证明：

- 一个正整数必然可以表示为 $3k+i$，其中 k 为非负整数，$i=0,1,2$。
- 它的平方 $9k^2+6ki+i*i$ 对 9 取模只会是 $0,1,4,7$，除了 0 之外，其它两两相加都不会是 9 的倍数。
- 那么取一个取模为 0 的，其它全部取走就是最优答案。

100pts

注意到数据范围 $3\le m\le 6\times 10^6$，也就是说 m 并不会太大，是在数组空间允许的范围内的。

我们可以发现，两个数平方和为 m 的倍数，即他们的平方分别对 m 取模后的和为 m 的倍数。

我们可以把它们平方对 m 取模存在一个数组里，用 d_i 表示取模为 i 的个数。

对于 i 和 $m-i$（取模结果），取了其中一组，另一组就一个都不能取，否则显然有和为 m，所以答案累加 d_i 与 d_{m-i} 最大值。

特别地，取模为 0 的只能取一个，否则任意两个对 m 取模为 0 的加起来显然是 m 的倍数。

此外，m 为偶数时，若 $i=\frac{m}{2}$，则 $m-i=\frac{m}{2}$，它们对应的是同一个 d_i，这种情况下只能取一个，否则任意两个的和必为 m 倍数。

```
#include <bits/stdc++.h>
#define zls rp++
#define ll long long
using namespace std;
int t,n;
ll m,a[1000005],d[6000005];
void work(){
    cin>>n>>m;
    // ↓ 多测记得清空数组
```

```
    for(int i=0;i<m;i++)d[i]=0;
    ll ans=0;
    for(int i=1;i<=n;i++){
        cin>>a[i];
        d[(a[i]*a[i])%m]++;
    }
    if(d[0])ans++;
    for(int i=1;i<=m/2;i++){
        int j=m-i;
        if(i==j&&d[i])ans++;
        else ans+=max(d[i],d[j]);
    }
    cout<<ans<<"\n";
}
int main(){
    ios::sync_with_stdio(false);
    cin.tie(0);
    cout.tie(0);
    cin>>t;
    while(t--)
    work();
    return 0;
}
```

D. 马驰万里福迎来

因为题目给了，所以对于每一对的 l, r，排列的第 $l+1 \sim r-1$ 项除了顺序外没有任何变化，也不可能有任何变化。

然后我们每次枚举 l, r 改变的值决定换不换。

注意在求值的时候，一定要注意顺序和变化量。

时间复杂度 $O(nm)$。

std:

```
#include <bits/stdc++.h>
using namespace std;

long long a[1005][1005];
int p[1005];
int main(){
//    freopen("talk10.in","r",stdin);
//    freopen("talk10.out","w",stdout);
    int n,m;
    scanf("%d%d",&n,&m);
    for(int i=1;i<=n;i++){
        for(int j=1;j<=n;j++){
            scanf("%lld",&a[i][j]);
        }
    }
    for(int i=1;i<=n;i++) scanf("%d",&p[i]);
    long long ans=0;
```

```
    for(int i=1;i<=n;i++){
        for(int j=i+1;j<=n;j++){
            ans+=a[p[i]][p[j]];
        }
    }
    for(int i=1;i<=m;i++){
        int l,r;
        scanf("%d%d",&l,&r);
           long long s=a[p[r]][p[l]]-a[p[l]][p[r]];
           for(int j=l+1;j<=r-1;j++){
               s=s+a[p[r]][p[j]]-a[p[j]][p[r]];
            s=s+a[p[j]][p[l]]-a[p[l]][p[j]];
        }
           if (s<0){
               swap(p[l],p[r]);ans+=s;
           }
    }
    printf("%lld\n",ans);
    return 0;
}
```

E. 玉阶雪化梅初绽

诈骗题。

发现盒子之间的关系是一个森林，考虑转化两种操作：

- 将任意一个节点的父亲改为任意一个它的原兄弟节点。
- 将任意一个节点的父亲改为它的原父亲的父亲。

所有树的根不能变，求最后全变成链且一端为根的最小次数。

显然对于每棵树分开求，最后求和即可。

注意到一个结论：对于任意一棵未达成要求的树，都可以进行一次操作将树的深度加一。具体地，找到最长的一条一端为根的链，随意选择上面一个有兄弟的节点，进行一次第一个操作即可。

因此我们可以将答案做到总节点个数减去最大深度。

注意到每次操作最多使深度加一，因此这个还是下限，那么也就是答案。

```
#include<bits/stdc++.h>
#define ll long long
using namespace std;
int n,root,maxx,fa[200010];
vector<int> son[200010];
void dfs(int v,int sum)
{
    maxx=max(maxx,sum);
    for(int u:son[v])dfs(u,sum+1);
}
int main()
{
```

```
    //freopen("20.in","r",stdin);
    //freopen("20.out","w",stdout);
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    int T;
    cin>>T;
    while(T--)
    {
        cin>>n;
        int ans=n;
        for(int i=0;i<n;i++)son[i].clear();
        for(int i=1;i<=n;i++)
        {
            cin>>fa[i];
            son[fa[i]].push_back(i);
        }
        for(int i=1;i<=n;i++)
        {
            if(fa[i])break;
            maxx=0;
            dfs(i,1);
            ans-=maxx;
        }
        cout<<ans<<endl;
    }

    return 0;
}
```

F. 紫陌烟轻柳渐栽

我们考虑一个问题，就是这题很诈骗。

很显而易见的发现你把前面一堆位置都钦定之后你要求的位数不会超过 60。

然后我们很显而易见的发现在你钦定一堆位数之后，就可以用一个很典的 trick。

然后我们发现假设你钦定完之后假设还剩 k 个数，很显然的答案为 $\sum_{i=0}^{k-1}\binom{k-1}{i}$。

然后预处理。

求值的时候记得两边丢。

最后不要忘了记录答案再算一下。

讲个笑话，作者本题最大收获为知道了 $\sum_{i=0}^{k-1}\binom{k-1}{i}=2^{k-1}$，赢！

下面是 std，可以当个乐子。

```
#include <bits/stdc++.h>
using namespace std;

long long c[65][65];
long long s[65];
long long rp=(1e18)+(5e17);
```

```cpp
int a[65];
int main(){
    ios::sync_with_stdio(false);
    s[0]=1;
    /*for(int i=1;i<=60;i++){
        c[i][0]=c[i][i]=1;
        for(int j=1;j<i;j++){
            c[i][j]=c[i-1][j-1]+c[i-1][j];
            if (c[i][j]>=rp) c[i][j]=rp;
            s[i]+=c[i][j];
            if (s[i]>=rp) s[i]=rp;
        }
        s[i]+=2;
    }*/
    for(int i=1;i<=60;i++){
        s[i]=s[i-1]*2;
        if (s[i]>=rp) s[i]=rp;
    }
    int t;
    cin>>t;
    while(t--){
        long long n,k;
        cin>>n>>k;
        if (n<=60&&s[n-1]<k){
            cout<<-1<<endl;
            continue;
        }
        k--;
        int p=0;
        for(int i=1;i<=60;i++){
            if (s[i]>k){
                p=i;
                break;
            }
        }
        int l=0,r=p+2;
        for(int j=p-1;j>=0;j--){
            if (k>=s[j]){
                k-=s[j];
                a[--r]=p-j;
            }
            else{
                a[++l]=p-j;
            }
        }
        a[++l]=p+1;
        int ans=0;
        for(int i=1;i<=p+1;i++){
            ans+=abs(a[i]-i);
        }
        cout<<ans<<endl;
    }
    return 0;
```

```
}
```

G. 杯举屠苏人共乐

个人认为还不错的一道题。

看到题可以想到，这是一棵以 0 为根的树。

然后我们考虑一个问题，如果出现一个环，那就会出现与这个环的连通块就废了。

很容易证，因为出现一个环，所以不可能有节点能连到 0 号节点，而如果某个节点属于此连通块，既然已经成环了，那么一定是这个节点的 x 在环上。证毕。

然后我们用并查集把每个环找出来丢掉，其他直接连单向边。

首先暴力做法很一眼，直接以 0 为根树形 dp，设 $dp_{i,j}$ 代表以 i 为根的子树种 j 个西瓜的最小花费。时间复杂度 $O(m^2)$。考虑优化。

我们发现一个奇妙的性质：$c_{x_i} \leq c_i$。

然后很容易想到贪心，对于每次都选能选的当中最小的，然后把这个选完之后新增能选的丢进堆。

时间复杂度 $O(m \log m)$。

std:

```cpp
#include <bits/stdc++.h>
using namespace std;

long long a[1000005];
int x[1000005];
int f[1000005];
int sz[1000005];
struct node{
    long long x;int w;
    friend bool operator<(node x,node y){
        return x.x>y.x;
    }
};
int find(int x){
    if (x==f[x]||f[x]==-1) return x;
    return f[x]=find(f[x]);
}
vector<int>p[1000005];
priority_queue<node>q;
int main(){
//    freopen("mud13.in","r",stdin);
//    freopen("mud13.out","w",stdout);
    int t;
    scanf("%d",&t);
    while(t--){
        long long n;int m;
        scanf("%lld%d",&n,&m);
        for(int i=0;i<=m;i++) p[i].clear();
        for(int i=1;i<=m;i++) scanf("%lld",&a[i]);
```

```
        for(int i=1;i<=m;i++) scanf("%d",&x[i]);
        for(int i=1;i<=m;i++) f[i]=i,sz[i]=1;
        for(int i=1;i<=m;i++){
            int u=find(i),v=find(x[i]);
            if (x[i]!=0){
                if (u==v) f[find(i)]=-1;
                else{
                    if (sz[u]<sz[v]){
                        f[u]=v;
                        sz[v]+=sz[u];
                    }
                    else f[v]=u,sz[u]+=sz[v];
                }
            }
        }
        for(int i=1;i<=m;i++){
            if (find(i)!=-1){
                p[x[i]].push_back(i);
            }
        }
        while(!q.empty()) q.pop();
        int ans=0;
        node p2;p2.w=0,p2.x=0;
        q.push(p2);
        while(!q.empty()){
            node p1=q.top();
            q.pop();
            if (n<p1.x) break;
            n-=p1.x;
            ans++;
            for(int i=0;i<(int)p[p1.w].size();i++){
                node rt;
                rt.w=p[p1.w][i];
                rt.x=a[rt.w];
                q.push(rt);
            }
        }
        printf("%d\n",ans-1);
    }
    return 0;
}
```

H. 灯悬华市夜同辉

感觉这题的思维很上位，因为想到很不简单，但是这似乎是一个很好的 trick 且并不好想。

看到此题一眼 $O(n^2)$ 做法，然后想到用数据结构优化。

经过思考，数据结构无法优化。

然后我们发现，这种奇妙的东西满足 $2 \times j = i + k$，然后我们想到映射神器 bitset。

我们先把值映射成下标。问题转化。

因为是排列，所以我们按值域处理，用两个 bitset 分别处理小的的位置和大的位置，再位移转化。

为了避免出现把 $k<j<i$ 的情况统计进去，我们考虑预处理一个 t 的 bitset 去杜绝这个问题。

我们发现这么搞好像会超时，于是我们对 t 分块再预处理即可。块长要调好。

bitset 神力常数小跑得快！

理论复杂度 $O(\frac{n^2}{w})$ 但是实际跑到了也就不到 2s。

std 的空间不超过 12 Mib.

代码短小精悍。

```
#include <bits/stdc++.h>
using namespace std;

const int N=200005;
int a[200005];
int b[200005];
bitset<N>x,y;
bitset<N>t[405];
int main(){
//    freopen("ce.in","r",stdin);
//    freopen("ce.out","w",stdout);
    ios::sync_with_stdio(false);
    int n;
    cin>>n;
    for(int i=1;i<=n;i++) cin>>a[i];
    for(int i=1;i<=n;i++) b[a[i]]=i;
    long long ans=0;
    for(int i=1;i<=200000;i++){
        t[400][i]=1;
    }
    for(int i=399;i>=1;i--){
        t[i]=t[i+1];
        for(int j=1;j<=500;j++){
            t[i][i*500+j]=0;
        }
    }
    for(int i=1;i<=n;i++) x[i]=0,y[i]=1;
    for(int i=1;i<=n;i++){
        bitset<N>p;
        p.reset();
        if (n+1-2*b[i]>=0){
            int x=b[i]-1;
            int s=(x-1)/500+1;
            p=t[s];
            for(int j=s*500;j>x;j--) p[j]=0;
        }
        else{
            int x=n-b[i];
            int s=(x-1)/500+1;
            p=t[s];
            for(int j=s*500;j>x;j--) p[j]=0;
```

```
        }
        if (n+1-2*b[i]>=0) ans+=(x&(y>>(n+1-2*b[i]))&p).count();
        else ans+=((x>>(2*b[i]-n-1))&y&p).count();
        x[b[i]]=1;
        y[n-b[i]+1]=0;
    }
    cout<<ans<<endl;
    return 0;
}
```

I. 蹄声踏破山河静

考虑不带修怎么做。

可以注意到两个区间如果存在包含关系或不交，可以通过调整使得答案不劣。

所以可以认为区间满足 l_i，r_i 分别递增。这时为了让区间尽可能覆盖更多位置，只需让 l_i 尽量小，r_i 尽量大。

所以拿出最左边的 k 个位置作为 l，最右边的 k 个位置作为 r 显然是不劣的。

（下面我们认为 $l_1=1,l_2=2,\dots,l_k=k$，$r_1=n-k+1,r_2=n-k,\dots,r_k=n$）

注意到所有区间的长度均 $>\lfloor\frac{n}{2}\rfloor$，故所有区间均过序列中点。因此考虑把区间拆成两段，$[l,r]$ 拆为 $[l,\lfloor\frac{n}{2}\rfloor]$ 和 $[1+\lfloor\frac{n}{2}\rfloor,r]$ 两个部分，这样两个部分就只分别和 l 和 r 有关了。

对于每个 l 和 r 处理出拆完后那一段区间的最大值（分别记为 $pmax$ 和 $smax$），现在一个区间 $[l_i,r_i]$ 的贡献即为 $\max(pmax_i,smax_i)$。

考虑拆掉 $\max$。显然 $pmax_i$ 关于 i 递减，$smax_i$ 关于 i 递增，于是 $\max$ 取 $pmax$ 必然是一段前缀，二分找到分界点即可。

可以在预处理 $pmax$ 和 $smax$ 后单次 $O(\log n)$ 地回答询问。

考虑带修怎么做，考虑一个位置 p 的 a_p 本质上即是在用 a_p 对 $pmax$ 的一段前缀或 $smax$ 的一段后缀 $\mathrm{checkmax}$，修改一个位置即是撤销之前的操作，加入一个新的 $\mathrm{checkmax}$ 操作。

加入操作是容易的，线段树上二分出 $\le a_p$ 的段区间覆盖即可，而删除是困难的，上线段树分治即可。

时间复杂度 $O(n\log^2 n)$。

```
#include<bits/stdc++.h>
#define ll long long

#define INF 214748364711

using namespace std;
ll n,q;
ll a[200005];
ll st[200005];
struct px
{
    ll p,v;
};
vector<px>opt[800005];
```

```
struct tree
{
    ll sum;
    int son[2],cover,maxn,minn;
};
struct SGT
{
    tree tr[7200005];
    int rt,cot;
    #define ls(id) tr[id].son[0]

    #define rs(id) tr[id].son[1]

    void cover(int& id,ll val,int l,int r)
    {
        ll yid=id;
        tr[id=++cot]=tr[yid];
        tr[id].cover=tr[id].maxn=tr[id].minn=val;
        tr[id].sum=(r-l+1)*val;
    }
    void pushdown(int &id,int l,int r)
    {
        ll yid=id;
        tr[id=++cot]=tr[yid];
        if(tr[id].cover==-INF)return;
        ll mid=l+r>>1;
        cover(ls(id),tr[id].cover,l,mid);
        cover(rs(id),tr[id].cover,mid+1,r);
        tr[id].cover=-INF;
    }
    void pushup(ll id)
    {
        tr[id].maxn=max(tr[ls(id)].maxn,tr[rs(id)].maxn);
        tr[id].sum=tr[ls(id)].sum+tr[rs(id)].sum;
        tr[id].minn=min(tr[ls(id)].minn,tr[rs(id)].minn);
    }
    void build(int &id,ll l,ll r)
    {
        if(!id)id=++cot;
        tr[id].cover=tr[id].maxn=tr[id].minn=-INF;
        if(l==r)
        {
            tr[id].sum=-INF;
            return;
        }
        ll mid=l+r>>1;
        build(ls(id),l,mid);
        build(rs(id),1+mid,r);
        pushup(id);
    }
    void find(int &id,ll l,ll r,ll val,ll mr)
    {
        if(l>mr)return;
```

```
        if(tr[id].maxn<val&&r<=mr)
        {
            cover(id,val,l,r);return;
        }
        pushdown(id,l,r);
        ll mid=l+r>>1;
        if(tr[ls(id)].minn<val)find(ls(id),l,mid,val,mr);
        find(rs(id),1+mid,r,val,mr);
        pushup(id);
    }
    ll query(int &id,ll l,ll r,ll ml,ll mr)
    {
        if(ml>mr)return 0;
        if(ml<=l&&r<=mr)return tr[id].sum;
        pushdown(id,l,r);
        ll mid=l+r>>1,res=0;
        if(ml<=mid)res+=query(ls(id),l,mid,ml,mr);
        if(mr>mid)res+=query(rs(id),1+mid,r,ml,mr);
        pushup(id);
        return res;
    }
    #undef ls
    #undef rs
}tr[2];
struct xp
{
    ll l,r;
    px s;
}s[400005];
ll tot;
ll cot;
ll ask[200005];
#define ls(id) id*2

#define rs(id) id*2+1

void insert(ll id,ll l,ll r,ll ml,ll mr,px s)
{
    if(ml<=l&&r<=mr)
    {
        opt[id].emplace_back(s);return;
    }
    ll mid=l+r>>1;
    if(ml<=mid)insert(ls(id),l,mid,ml,mr,s);
    if(mr>mid)insert(rs(id),1+mid,r,ml,mr,s);
}
void solve(ll id,ll l,ll r)
{
    ll yrt[2]={tr[0].rt,tr[1].rt},cot[2]={tr[0].cot,tr[1].cot};
    for(auto it:opt[id])
    {
        if(it.p<=n/2)
        {
```

```
            if(tr[0].tr[tr[0].rt].minn>=it.v)continue;
            ll r=it.p;
            tr[0].find(tr[0].rt,1,n/2,it.v,r);
        }
        else
        {
            if(tr[1].tr[tr[1].rt].minn>=it.v)continue;
            ll r=min(n/2,n-it.p+1);
            tr[1].find(tr[1].rt,1,n/2,it.v,r);
        }
    }
    if(l==r)
    {
        ll uid=l;
        ll r=ask[uid];
        ll l=1,res=0;
        while(l<=r)
        {
            ll mid=l+r>>1;
            if(tr[0].query(tr[0].rt,1,n/2,mid,mid)>tr[1].query(tr[1].rt,1,n/2,ask[uid]-
mid+1,ask[uid]-mid+1))
            res=mid,l=mid+1;
            else r=mid-1;
        }
        ask[uid]=tr[0].query(tr[0].rt,1,n/2,1,res)+tr[1].query(tr[1].rt,1,n/2,1,ask[uid]-
res);
    }
    else
    {
        ll mid=l+r>>1;
        solve(ls(id),l,mid);
        solve(rs(id),1+mid,r);
    }
    for(ll i=0;i<2;++i)
    tr[i].rt=yrt[i],tr[i].cot=cot[i];
}
int main()
{
    ios::sync_with_stdio(0);cin.tie(0);cout.tie(0);
    cin>>n>>q;
    for(ll i=1;i<=n;++i)st[i]=1,cin>>a[i];
    ll opt;
    for(ll i=1;i<=q;++i)
    {
        cin>>opt;
        if(opt==1)
        {
            ll p,v;
            cin>>p>>v;
            s[++tot].l=st[p];
            s[tot].r=cot;s[tot].s=(px){p,a[p]};
            st[p]=cot+1;
            a[p]+=v;
```

```
        }
        else
        {
            ll k;
            cin>>k;
            ask[++cot]=k;
        }
    }
    for(ll i=1;i<=n;++i)
    {
        s[++tot].l=st[i];
        s[tot].r=cot;
        s[tot].s=(px){i,a[i]};
    }
    for(ll i=1;i<=tot;++i)
        if(s[i].l<=s[i].r)insert(1,1,cot,s[i].l,s[i].r,s[i].s);
    tr[0].build(tr[0].rt,1,n/2);
    tr[1].build(tr[1].rt,1,n/2);
    solve(1,1,cot);
    for(ll i=1;i<=cot;++i)cout<<ask[i]<<'\n';
}
```

J. 愿借骏心行远志

注意到我们本质上是要在知道我们当前有哪些钥匙，有哪些箱子的情况下判断是否有解，构造最小化方案只需要我们枚举当前开哪个箱子后判断剩下箱子是否有解即可。

考虑你最开始有一把钥匙，每个箱子内恰好有一把钥匙的情况。

把钥匙的种类视作点，一个箱子视作开箱子类型钥匙和打开后获得钥匙之间的边，题意就变成了指定点出发欧拉路径。

但是如果一个箱子里有多把钥匙，问题就不是欧拉路了，考虑寻找一个图有解的情况：

- 当每类钥匙的总数大于等于需要使用的数量，且可以获取需要用的每种钥匙（上图从最开始拥有钥匙对应节点 BFS 可达），我们声称这是必然有解的。

必要性是显然的，充分性就是我们要说明始终存在一个可以打开的箱子，打开后不影响连通性。

你当前图满足：

对于任意需要的钥匙类型 s，存在一个钥匙种类序列 $k_1, k_2, \ldots, k_m$，其中 k_1 为你当前手上拥有的钥匙，$k_m = s$，k_{i+1} 可以通过 k_i 打开的箱子获得。

考虑你手上的某种钥匙 a，如果你当前拥有数量已经超过需要数量了，显然无影响，否则：

你现在有一个序列 $ka_1, ka_2, \ldots, ka_{m_a}$ 可以获取一个 a，我们钦定 $\forall 1 < i < m_a, ka_i \neq a$。

- 若 $ka_1 \neq a$，则有：

  对于需要的 $\forall c$ 钥匙，你现在有一个钥匙种类序列 $kc_1, kc_2 \ldots, kc_{m_c} = c$ 可以获取 c，如果序列 kc 中没有 a，a 使用后不影响连通性，否则你考虑找到 j 最大的 kc_j 使得 $\exists i, ka_i = kc_j$，若你使用了 a，你现在可以使用 $ka_1, ka_2, \ldots, ka_{i-1}, kc_j, kc_{j+1}, \ldots, kc_{m_c}$ 获取 c。

- 若 $ka_1 = a$，则有：

  直接用 a 获取 ka_2，你发现 $ka_1 \neq a$ 论证仍然成立。

故上述条件充分。

做就直接枚举当前位开哪个箱子，去掉这个箱子后在剩余图上 BFS 即可，时间复杂度 $O(tn^2k)$。

```
#include<bits/stdc++.h>
#define ll long long
using namespace std;
const int MAXN=200;
ll t;
ll k,n;
ll kv[205],cot[205],sum[205];
bool use[205],vis[205];
struct px
{
    ll nd;
    vector<ll>h;
}a[205];
struct ed
{
    ll v,next;
}edge[605];
ll head[205],cnt;
void add(ll u,ll v)
{
    edge[++cnt].v=v;edge[cnt].next=head[u];head[u]=cnt;
}
bool check()
{
    memset(head,0,sizeof(head));cnt=0;
    for(ll i=1;i<=MAXN;++i)
        if(kv[i])
        add(0,i);
    memset(vis,0,sizeof(vis));
    for(ll i=1;i<=n;++i)
        if(!use[i])
            for(auto v:a[i].h)
                add(a[i].nd,v);
    queue<ll>k;
    k.emplace(0);vis[0]=1;
    while(!k.empty())
    {
        ll id=k.front();k.pop();
        for(ll i=head[id];i;i=edge[i].next)
        {
            ll v=edge[i].v;
            if(!vis[v])
            vis[v]=1,k.emplace(v);
        }
    }
    for(ll i=1;i<=MAXN;++i)
        if(cot[i]&&!vis[i])
        return 0;
    return 1;
```

```
}
int main()
{
    ios::sync_with_stdio(0);cin.tie(0);cout.tie(0);
    cin>>t;
    for(ll sb=1;sb<=t;++sb)
    {
        memset(use,0,sizeof(use));
        memset(cot,0,sizeof(cot));
        memset(kv,0,sizeof(kv));
        memset(sum,0,sizeof(sum));
        cin>>k>>n;
        ll ls;
        while(k--)
        cin>>ls,++kv[ls],++sum[ls];
        for(ll i=1;i<=n;++i)
        {
            cin>>a[i].nd;
            ++cot[a[i].nd];
            ll s,ls;
            cin>>s;
            a[i].h.clear();
            while(s--)cin>>ls,a[i].h.emplace_back(ls),++sum[ls];
        }
        bool ko=0;
        for(ll i=1;i<=MAXN;++i)
            if(cot[i]>sum[i])
            {
                ko=1;break;
            }
        if(ko)
        {
            cout<<"No\n";continue;
        }
        vector<ll>res;
        for(ll sb=1;sb<=n;++sb)
        {
            bool ok=0;
            for(ll i=1;i<=n;++i)
            {
                if(!use[i]&&kv[a[i].nd])
                {
                    --kv[a[i].nd];use[i]=1;
                    for(auto it:a[i].h)++kv[it];
                    --cot[a[i].nd];
                    if(check())
                    {
                        res.emplace_back(i);
                        ok=1;
                        break;
                    }
                    ++cot[a[i].nd];
                    use[i]=0;
```

```
                    for(auto it:a[i].h)--kv[it];
                    ++kv[a[i].nd];
                }
            }
            if(!ok)
            {
                ko=1;break;
            }
        }
        if(ko)
        {
            cout<<"No\n";continue;
        }
        cout<<"Yes\n";
        for(auto it:res)cout<<it<<' ';
        cout<<'\n';
    }
}
```

K. 一年胜景自今催

把矩阵视作一个网格，每个格子有权值。

考虑对于 $i+j$ 为奇数的格子，将其左上格点与右下格点连边权为 $A_{i,j}$ 的边。对于 $i+j$ 为偶数的格子，将其左下格点与右上格点连边权为 $A_{i,j}$ 的边。

可以注意到，对于任意两个不能同时被选的格子，代表其的边一定存在一个公共格点。

所以现在问题相当于选一些边，使得选择的边的端点不重复，最大化选择的边的边权最大值。

发现建出来的图也是一个网格图，所以也是是一个二分图，把偶数行点放入左部点，奇数行点放入右部点。直接费用流跑二分图最大匹配即可。

复杂度 $O((nm)^3)$，常数极小。

```
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
namespace MCMF {
    constexpr ll V = 100010, E = 1000010;
    constexpr ll inf = 0x3f3f3f3f3f3f3f3f;
    ll sze = 1, mx, hd[V], dis[V], prv[V], inq[V], mf[V];
    struct Edge { ll v, f, c, nxt; } e[E];
    void add_once(ll u, ll v, ll f, ll c) {
        e[++sze] = {v, f, c, hd[u]}, hd[u] = sze;
    }
    void add(ll u, ll v, ll f, ll c) {
        add_once(u, v, f, c);
        add_once(v, u, 0, -c);
        mx = max(mx, max(u, v));
    }
    bool spfa(ll s, ll t) {
        for (ll i = 0; i <= mx; i++) dis[i] = -inf;
        queue<ll> q; dis[s] = 0, mf[s] = inf, q.push(s), inq[s] = 1;
```

```
        while (!q.empty()) {
            ll u = q.front(); q.pop(), inq[u] = 0;
            for (ll i = hd[u]; i; i = e[i].nxt) {
                auto [v, f, c, nxt] = e[i];
                if (f && dis[v] < dis[u] + c) {
                    dis[v] = dis[u] + c, prv[v] = i, mf[v] = min(mf[u], f);
                    if (!inq[v]) q.push(v), inq[v] = 1;
                }
            }
        }
        return dis[t] > -inf;
    }
    ll EK(ll s, ll t) {
        ll cost = 0;
        while (spfa(s, t)) {
            if (dis[t] < 0) break;
            cost += mf[t] * dis[t];
            for (ll u = t; u != s; u = e[prv[u] ^ 1].v) {
                e[prv[u]].f -= mf[t], e[prv[u] ^ 1].f += mf[t];
            }
        }
        return cost;
    }
}
ll n, m;
ll id(ll x, ll y) {
    return (x - 1) * (m + 1) + y;
}
int main() {
    cin >> n >> m;
    for (ll i = 1; i <= n; i++) {
        for (ll j = 1, x; j <= m; j++) {
            cin >> x;
            ll u = (i + j) % 2 ? id(i, j) : id(i, j + 1);
            ll v = (i + j) % 2 ? id(i + 1, j + 1) : id(i + 1, j);
            if (i % 2 == 0) swap(u, v);
            MCMF::add(u, v, 1, x);
        }
    }
    for (ll i = 1; i <= n + 1; i += 2) {
        for (ll j = 2; j <= m + 1; j += 2) {
            MCMF::add(0, id(i, j), 1, 0);
        }
    }
    ll t = id(n + 1, m + 1) + 1;
    for (ll i = 2; i <= n + 1; i += 2) {
        for (ll j = 1; j <= m + 1; j += 2) {
            MCMF::add(id(i, j), t, 1, 0);
        }
    }
    cout << MCMF::EK(0, t) << '\n';
    return 0;
}
```