C. Dance Party

题意

给定 $n\times 2$ 的二分图。对左部每个点，仅和右部 k_i 个点不连边。求二分图最大匹配。

题解

设 $k=\max_{i=1}^n k_i$

先进行预匹配，每个左部点任选一个还未被匹配且有连边的右部点匹配，可以用 set 维护所有未匹配的右部点，时间复杂度 $O(nk\log n)$

接下来剩下的未匹配的点一定不超过 k 个，对每个点考虑匈牙利算法匹配， 总时间复杂度为 $O(km)$

$O(m)\sim O(n^2)$考虑优化。假定现在需要对点 i 进行匈牙利算法，将右部与点 i 不相邻的点染黑，其余右部点染白。

对除点 i 以外的左部点，仅保留与黑点相关的连边，这样 $O(m)\sim O(nk)$总时间复杂度 $O\left(nk^2\right)$ 足以通过此题。

关于算法的正确性，假设在原图上存在一条从 i 出发的增广路，且增广路上除了 i 以外有其他点的失配边指向白点。

找到增广路上的最后一个白点，直接将 i 的失配边指向该点然后保留原增广路的剩余部分也可以一条增广路。

同时该增广路上除了 i 其他点的失配边都指向黑点。因此只要原图存在一条从 i 出发的增广路则只保留与黑点相关的连边也可以得到一条增广路。

```
const int MAXN=3e4+5,MAXK=105;
struct Edge{
    int to,next;
}edge[MAXN*MAXK];
int head[MAXN],edge_cnt;
void Insert(int u,int v){
    edge[++edge_cnt]=Edge{v,head[u]};
    head[u]=edge_cnt;
}
bitset<MAXN> bt[MAXN];
vector<int> g[MAXN];
namespace KM{
    set<int> s;
    int match[MAXN],vis[MAXN];
    bool dfs(int u,int k){
        if(vis[u]==k)
        return false;
        vis[u]=k;
        for(int i=head[u];i;i=edge[i].next){
```

```
            int v=edge[i].to;
            if(!match[v]||dfs(match[v],k))
            return match[v]=u,true;
        }
        return false;
    }
    bool get_pair(int n){
        _rep(u,1,n)
        s.insert(u);
        vector<int> vec;
        _rep(u,1,n){
            bool flag=true;
            for(int v:s){
                if(!bt[u][v]){
                    match[v]=u;
                    s.erase(v);
                    flag=false;
                    break;
                }
            }
            if(flag)
            vec.push_back(u);
        }
        for(int i:vec){
            mem(head,0);
            edge_cnt=0;
            _rep(u,1,n){
                for(int v:g[i]){
                    if(!bt[u][v])
                    Insert(u,v);
                }
            }
            _rep(v,1,n){
                if(!bt[i][v])
                Insert(i,v);
            }
            if(!dfs(i,i))
            return false;
        }
        return true;
    }
}
int ans[MAXN];
int main()
{
    int n=read_int();
    _rep(u,1,n){
        int k=read_int();
        while(k--){
            int v=read_int();
            g[u].push_back(v);
```

```
            bt[u][v]=1;
        }
    }
    if(KM::get_pair(n)){
        _rep(i,1,n)
        ans[KM::match[i]]=i;
        _rep(i,1,n)
        space(ans[i]);
    }
    else
    puts("-1");
    return 0;
}
```