集合(conspiracy,3s,256MB)

【算法分析】

考虑找出所有 A 集合的点两两相连、B 集合的点两两不相连、但 A，B 不一定均为空的方案，并分别判断这些方案是否满足 A，B 均不为空，满足条件则计入答案。

考虑先找出一组这样的方案：把每个点拆成点 i 和点 i+n，分别表示把 i 放入 A 集合和 B 集合。若 i 和 j 相连，那么 i 和 j 不能一起在 B 集合中，即连边：i+n->j，j+n->i。否则，i 和 j 不能一起在 A 集合中，即连边：i->j+n，j->i+n。

然后对这张 2n 个点的图求强连通分量，若 i 和 i+n 在同一个强连通分量内则无合法方案，直接输出 0。否则记 id[i]为点 i 所在强连通分量编号，将所有满足 id[i]<id[i+n] 的点放入 A 集合，其它点放入 B 集合。

至此我们得到了初始方案，考虑调整这种方案以得到其它方案。我们发现，如果初始方案中，点 u,v 都在 A 集合，那么 u,v 不可以同时被调整到 B 集合。同理，B 集合中的两个点也不能同时被调到 A 集合。也就是说，不能同时将两个同一集合的点调到另一个集合。

那么我们可以把调整方案分为 3 类：

1.在 A 集合中找一个点 u，调到 B 集合。

合法条件：所有和 u 相连的点都在 A 集合。

2.在 B 集合中找一个点 u，调到 A 集合。

合法条件：所有和 u 不相连的点都在 B 集合。

3.在 A 集合中找一个点 u，在 B 集合中找一个点 v，互换 u,v 的所属集合。

合法条件：除 v 以外与 u 相连的点都在 A 集合，且除 u 以外与 v 不相连的点都在 B 集合。

这样我们就能通过调整初始方案，得到所有合法方案了。

时间复杂度 $O(n^2)$，空间复杂度 $O(n^2)$。

【参考程序】

```
#include <bits/stdc++.h>
using namespace std;
#define pb push_back
template <class t>
inline void read(t & res)
{
    char ch;
    while (ch = getchar(), !isdigit(ch));
    res = ch ^ 48;
    while (ch = getchar(), isdigit(ch))
    res = res * 10 + (ch ^ 48);
}
const int e = 5005, o = 2 * e;
vector<int>g[o];
bool bo[e][e];
int n, neg[o], cnt[e], ans, dfn[o], low[o], vis[o], tim, stk[o], top,
id[o], a[e], b[e];
int c1[e], c2[e], tot;
```

```
inline void link(int x, int y) // 点 x 与点 y 互斥
{
    g[x].pb(neg[y]);
    g[y].pb(neg[x]);
}
inline void dfs(int u)
{
    dfn[u] = low[u] = ++tim;
    stk[++top] = u;
    vis[u] = 1;
    int i, len = g[u].size();
    for (i = 0; i < len; i++)
    {
        int v = g[u][i];
        if (!vis[v]) dfs(v);
        if (vis[v] == 1) low[u] = min(low[u], low[v]);
    }
    if (dfn[u] == low[u])
    {
        ++tot;
        do
        {
            int x = stk[top];
            vis[x] = 2;
            id[x] = tot;
        }
        while (stk[top--] != u);
    }
}
inline void nie()
{
    puts("0");
    fclose(stdin);
    fclose(stdout);
    exit(0);
}
int main()
{
    int i, j, x;
    read(n);
    for (i = 1; i <= n; i++)
    {
        read(cnt[i]);
        for (j = 1; j <= cnt[i]; j++)
```

```
        {
            read(x);
            bo[i][x] = 1;
        }
        neg[i] = i + n;
        neg[i + n] = i;
    }
    for (i = 1; i < n; i++)
    for (j = i + 1; j <= n; j++)
    if (bo[i][j]) link(neg[i], neg[j]); // 若 i 和 j 相连，连边：i+n->j，j+n->i
    else link(i, j); // 否则，连边：i->j+n，j->i+n
    for (i = 1; i <= 2 * n; i++) // 对这张 2n 个点的图求强连通分量
    if (!vis[i]) dfs(i);
    for (i = 1; i <= n; i++) // 若 i 和 i+n 在同一个强连通分量内则无合法方案
    {
        if (id[i] == id[i + n]) nie(); // 满足 id[i]<id[i+n]的点放入 A 集合，
其它点放入 B 集合
        if (id[i] < id[i + n]) c1[++c1[0]] = i;
        else c2[++c2[0]] = i;
    }
    int ans = c1[0] && c2[0];
    for (i = 1; i <= c1[0]; i++) // 求 A 集合中的每个点与同在 A 集合的几个点相
连
    for (j = 1; j <= c1[0]; j++)
    if (bo[c1[i]][c1[j]]) a[c1[i]]++;
    for (i = 1; i <= c2[0]; i++) // 求 B 集合中的每个点与同在 B 集合的几个点不
相连
    for (j = 1; j <= c2[0]; j++)
    if (!bo[c2[i]][c2[j]] && i != j) b[c2[i]]++;
    for (i = 1; i <= c1[0]; i++) // 在 A 集合中找一个点 u，调到 B 集合。
    if (a[c1[i]] == cnt[c1[i]] && c1[0] > 1) ans++;
    for (i = 1; i <= c2[0]; i++) // 在 B 集合中找一个点 u，调到 A 集合。
    if (b[c2[i]] == n - cnt[c2[i]] - 1 && c2[0] > 1) ans++;
    for (i = 1; i <= c1[0]; i++) // 在 A 集合中找一个点 u，在 B 集合中找一个点
v，互换 u,v 的所属集合。
    for (j = 1; j <= c2[0]; j++)
    if (a[c1[i]] == cnt[c1[i]] || (a[c1[i]] == cnt[c1[i]] - 1 &&
bo[c1[i]][c2[j]]))
    if (b[c2[j]] == n - cnt[c2[j]] - 1 || (b[c2[j]] == n - cnt[c2[j]] -
2 && !bo[c1[i]][c2[j]])) ans++;
    printf("%d\n", ans);
    fclose(stdin);
    fclose(stdout);
    return 0;
```

```
}
```

洗牌(wash,1s,128MB)

【算法分析】

由于题目保证有解，则字符要么出现4次，要么仅出现2次。

假设某种字符出现4次，位置分别为$[a,b,c,d]$，则该字符的分配方案为$[(a,b),(c,d)]$或$[(a,c),(b,d)]$（“()”内为同属于一个原字符串的字符），两种分配方案对立；

若某种字符仅出现 2 次，位置分别在$[u,v]$，则可以将它看作是每两个字符重合，分配方案只有1种：$[(u,u'),(v,v')]$。但为了方便，同样可以将之拆成两个对立的分配方案（实际上两个方案完全一样）：$[(u,u'),(v,v')]$和$[(u,u'),(v,v')]$。

现在，对于每种出现的字符，都有两种互相对立的分配方案。

对于某两种字符的某种分配方案 $[(x_0,x_1),(x'_0,x'_1)]$和$[(y_0,y_1),(y'_0,y'_1)]$，它们起冲突的条件无非是“出现在两个原串的同一位置的两组字符的相对位置不同”，即：

1. (x_0,x_1)和(y_0,y_1)冲突
2. (x_0,x_1)和(y'_0,y'_1)冲突
3. (x'_0,x'_1)和(y_0,y_1)冲突
4. (x'_0,x'_1)和(y'_0,y'_1)冲突

这些起冲突的分配方案构成的二元关系可以转化为 2-SAT 问题，建图跑 Tarjan 求解即可。

时间复杂度$O(n+m)$，其中n,m分别为建成的图中的点数和边数。

【参考程序】

```
//潘恩宁
#include <bits/stdc++.h>
#define For(i, a, b) for (int i = a, bb = b; i <= bb; ++i)
#define Rof(i, a, b) for (int i = a, bb = b; i >= bb; --i)
#define s64 long long
#define oppo(x) (x <= m ? x + m : x - m)

using std :: vector;
using std :: swap;

template <class T>
inline void get(T &res)
{
    char ch;
    bool bo = false;
    while (ch = getchar(), !isdigit(ch))
        if (ch == '-') bo = true;

    res = ch ^ 48;
    while (ch = getchar(), isdigit(ch))
```

```
        res = (res << 1) + (res << 3) + (ch ^ 48);

    if (bo) res = ~ res + 1;
    return;
}

template <class T>
inline void _put(T x)
{
    if (x > 9) _put(x / 10);
    putchar(x % 10 + '0');
    return;
}

template <class T>
inline void put(T x, char ch)
{
    if (x < 0)
    {
        putchar('-');
        x = ~ x + 1;
    }
    _put(x);
    putchar(ch);
    return;
}

const int MaxN = 2005, MaxM = 5e5 + 5;
int adj[MaxN], nxt[MaxM], go[MaxM], cap[MaxM], P;
int dfn[MaxN], low[MaxN], stk[MaxN], bel[MaxN], tim, top, num;
int apr[MaxN], l1[MaxN], r1[MaxN], l2[MaxN], r2[MaxN], n, m;
bool ans[MaxN];
vector <int> d[MaxN >> 1];

inline void Link(int x, int y)
{
    nxt[++P] = adj[x], adj[x] = P, go[P] = y;
    return;
}

inline bool cross(int x1, int x2, int y1, int y2)
{
    return x1 < y1 && y2 < x2 || y1 < x1 && x2 < y2;
}
```

```
inline bool check(int a, int b) //判断 a,b 两种分配是否矛盾
{
    return cross(l1[a], r1[a], l1[b], r1[b])
        || cross(l1[a], r1[a], l2[b], r2[b])
        || cross(l2[a], r2[a], l1[b], r1[b])
        || cross(l2[a], r2[a], l2[b], r2[b]);
}

inline void ckmin(int &x, const int &y)
{
    if (x > y) x = y;
    return;
}

inline void Tarjan(int u)
{
    dfn[u] = low[u] = ++tim;
    stk[++top] = u;

    int v;
    for (int e = adj[u]; e; e = nxt[e])
    if (!dfn[v = go[e]])
    {
        Tarjan(v);
        ckmin(low[u], low[v]);
    }
    else if (!bel[v])
        ckmin(low[u], dfn[v]);
    if (dfn[u] == low[u])
    {
        num++;
        do
            bel[stk[top]] = num;
        while (stk[top--] ^ u);
    }
    return;
}

int main()
{
    freopen("wash.in", "r", stdin), freopen("wash.out", "w", stdout);

    get(n);
```

```
    int x;
    For(i, 1, n)
    {
        get(x);
        if (!apr[x]) apr[x] = ++m;
        d[apr[x]].push_back(i);
    }
    For(i, 1, m)
    if (d[i].size() == 2) //只出现 2 次的字符只有一种分配方法
    {
        l1[i] = l1[i + m] = l2[i] = l2[i + m] = d[i][0];
        r1[i] = r1[i + m] = r2[i] = r2[i + m] = d[i][1];
    }
    else
    {
        l1[i] = l1[i + m] = d[i][0];
        l2[i] = r1[i + m] = d[i][1];
        r1[i] = l2[i + m] = d[i][2];
        r2[i] = r2[i + m] = d[i][3];
    }
    For(i, 1, (m << 1) - 1)
    For(j, i + 1, m << 1)
    if (i + m ^ j && check(i, j))
    {
        Link(i, oppo(j));
        Link(j, oppo(i));
    }
    For(i, 1, m << 1)
    if (!dfn[i]) Tarjan(i);
    For(i, 1, m) //Tarjan 给连通块的编号恰为拓扑序逆序
        ans[r1[bel[i] < bel[i + m] ? i : i + m]] = ans[r2[i]] = true;
    For(i, 1, n)
        putchar(ans[i] ? '1' : '0');
    putchar('\n');

    fclose(stdin), fclose(stdout);
    return 0;
}
```

袋子(florida,3s,256MB)

【算法分析】

不妨假设$D(A) \le D(B)$。

考虑枚举$D(B)$，显然$D(B)$的值有$O(n^2)$种情况。

然后在$D(B)$确定的情况下，二分$D(A)$，问题转化为在$D(A)$和$D(B)$都确定的情况下，是

否存在一种方案使得A集合的代价不超过D(A)，B集合的代价不超过D(B)。

对于任意两个元素i, j有三种情况：

（1） 无限制。

（2） 不能同时在A集合内，但可以同时在B集合内。

（3） 不能在同一个集合内。

于是这就是一个经典的 2-SAT 问题了，把每个元素i拆成两个点i和¬i后，对于任意两个元素,分上面三种情况连边后 Tarjan 求强连通分量，判断是否存在i和¬i属于同一个强连通分量即可。时间复杂度$O(n^4 \log n)$。

为了优化这个算法，我们需要分析性质。思考D(B)的取值是否只有O(n)种？

把给出的矩阵$T_{i,j}$建成图（(i, j)边权为$T_{i,j}$）之后求一遍最大生成树，然后把最大生成树进行黑白染色。

可以通过分类讨论证明：在$D(A) \le D(B)$的前提下，如果B集合中既有黑点又有白点，那么D(B)一定是最大生成树上的边权之一，否则B集合只能是全体黑点或全体白点。

于是D(B)可能的取值为该图最大生成树上的所有边权，以及最大生成树进行黑白染色之后，黑点两两之间的最大边权和白点两两之间的最大边权，一共最多$n + 1$种。

于是只需枚举这些D(B)可能的取值即可。复杂度$O(n^3 \log n)$，可以通过此题。

【参考程序】

```
// 陈栉旷
#include <cmath>
#include <cstdio>
#include <cstring>
#include <iostream>
#include <algorithm>

inline int read()
{
    int res = 0; bool bo = 0; char c;
    while (((c = getchar()) < '0' || c > '9') && c != '-');
    if (c == '-') bo = 1; else res = c - 48;
    while ((c = getchar()) >= '0' && c <= '9')
        res = (res << 3) + (res << 1) + (c - 48);
    return bo ? ~res + 1 : res;
}

template <class T>
inline T Max(const T &a, const T &b) {return a > b ? a : b;}

template <class T>
inline T Min(const T &a, const T &b) {return a < b ? a : b;}

typedef long longll;

const int N = 255, M = N << 1, L = N * N * 4, INF = 2147483647;
```

```
int n, m, a[N][N], fa[N], ecnt, nxt[L], adj[M], go[L], val[N], tot,
dfn[M], low[M], times, top, stk[M], bel[M];
bool col[N], ins[M];

void add_edge(int u, int v)
{
    nxt[++ecnt] = adj[u]; adj[u] = ecnt; go[ecnt] = v;
    nxt[++ecnt] = adj[v]; adj[v] = ecnt; go[ecnt] = u;
}

void egde_dda(int u, int v)
{
    nxt[++ecnt] = adj[u]; adj[u] = ecnt; go[ecnt] = v;
}

void dfs(int u, int fu)
{
    for (int e = adj[u], v; e; e = nxt[e])
        if ((v = go[e]) != fu)
            col[v] = col[u] ^ 1, dfs(v, u);
}

struct edge
{
    int u, v, w;
} eg[L];

inline bool comp(edge a, edge b)
{
    return a.w>b.w;
}

int cx(int x)
{
    if (fa[x] != x) fa[x] = cx(fa[x]);
    return fa[x];
}

bool zm(int x, int y)
{
    int ix = cx(x), iy = cx(y);
    if (ix != iy) return fa[iy] = ix, 1;
    return 0;
}
```

```
void scc(int u)
{
    dfn[stk[++top] = u] = low[u] = ++times;
    ins[u] = 1;
    for (int e = adj[u], v = go[e]; e; e = nxt[e], v = go[e])
        if (!dfn[v])
        {
            scc(v);
            low[u] = Min(low[u], low[v]);
        }
        else if (ins[v]) low[u] = Min(low[u], dfn[v]);
    if (dfn[u] == low[u])
    {
        bel[u] = ++tot; int v; ins[u] = 0;
        while (v = stk[top--], v != u) bel[v] = tot, ins[v] = 0;
    }
}

bool check(int A, int B)
{
    ecnt = times = tot = 0;
    for (int i = 1; i<= (n << 1); i++) adj[i] = dfn[i] = 0;
    for (int i = 1; i<= n; i++)
        for (int j = i + 1; j <= n; j++)
        {
            if (a[i][j] > A) egde_dda(i, n + j), egde_dda(j, n + i);
            if (a[i][j] > B) egde_dda(n + i, j), egde_dda(n + j, i);
        }
    for (int i = 1; i<= (n << 1); i++)
        if (!dfn[i]) scc(i);
    for (int i = 1; i<= n; i++) if (bel[i] == bel[n + i]) return 0;
    return 1;
}

int jiejuediao(int B)
{
    int l = 0, r = B;
    while (l <= r)
    {
        int mid = l + r >> 1;
        if (check(mid, B)) r = mid - 1;
        else l = mid + 1;
    }
```

```
    return l == B + 1 ?INF : l + B;
}

void work()
{
    m = 0;
    for (int i = 1; i<= n; i++) fa[i] = i;
    for (int i = 1; i<= n; i++)
        for (int j = i + 1; j <= n; j++)
            eg[++m] = (edge) {i, j, a[i][j] = read()};
    std::sort(eg + 1, eg + m + 1, comp);
    ecnt = tot = 0;
    for (int i = 1; i<= n; i++) adj[i] = 0;
    for (int i = 1; i<= m; i++)
        if (zm(eg[i].u, eg[i].v))
            add_edge(eg[i].u, eg[i].v), val[++tot] = eg[i].w;
    val[n] = val[n + 1] = 0;
    col[1] = 0; dfs(1, 0);
    for (int i = 1; i<= n; i++)
        for (int j = i + 1; j <= n; j++)
            if (col[i] == col[j])
                val[n + col[i]] = Max(val[n + col[i]], a[i][j]);
    int ans = INF;
    for (int i = 1; i<= n + 1; i++)
        ans = Min(ans, jiejuediao(val[i]));
    printf("%d\n", ans);
}

int main()
{
    freopen("florida.in", "r", stdin);
    freopen("florida.out", "w", stdout);

    while (~scanf("%d", &n)) work();
    return 0;
}
```