// 匈牙利算法求二分图最大匹配

bool state[100048];int result[100048];

vector<int> v[100048];

bool find(int cur)

{

int i;

for (i=0;i<v[cur].size();i++)

//state数组指的是如果之前曾试图改变v[cur][i]的归属

//因为还在find函数内，所以必然是已经失败了的，所以

//不用再浪费时间了

if (!state[v[cur][i]])

{

state[v[cur][i]]=true;

if (!result[v[cur][i]] || find(result[v[cur][i]]))

{

result[v[cur][i]]=cur;

return true;

}

}

}

int main ()

{

int i,ans=0;

for (i=1;i<=n;i++)

{

memset(state,false,sizeof(state));

if (find(i)) ans++;

}

}

//KM算法求二分图最大权匹配

//bfs+slack优化,O(n^3)

const int MAXN=400;

int n,e,nl,nr;

int linkx[MAXN+48],linky[MAXN+48],slack[MAXN+48],pre[MAXN+48];

bool vx[MAXN+48],vy[MAXN+48];int lx[MAXN+48],ly[MAXN+48];

int ga[MAXN+48][MAXN+48];

//ga[i][j]表示左边的第i个人和右边的第j个人匹配的权值

//linkx[i]表示左边的第i个人和右边的哪个人匹配，未匹配为0

//linky[i]表示右边的第i个人和左边的哪个人匹配，未匹配为0

//lx[i]和ly[i]是左右的顶标，vx[i]和vy[i]表示左右当前是否访问

//pre[u]是存储路径的辅助数组，find\_route的实现可以精妙的修改增广路上的匹配

inline void find\_route(int &u)

{

for (;u;swap(u,linkx[pre[u]]))

linky[u]=pre[u];

}

int q[MAXN+48],head,tail;

inline void bfs(int u)

{

head=tail=1;q[1]=u;vx[u]=true;int t;

for (;;)

{

while (head<=tail)

{

u=q[head++];

for (register int v=1;v<=n;v++)

{

if (vy[v] || (t=lx[u]+ly[v]-ga[u][v])>slack[v]) continue;

pre[v]=u;

if (!t)

{

if (!linky[v]) {find\_route(v);return;}

vy[v]=true;vx[linky[v]]=true;q[++tail]=linky[v];

}

else slack[v]=t;

}

}

int d=INF;

for (register int v=1;v<=n;v++) if (!vy[v] && slack[v]<d) d=slack[v],u=v;

for (register int i=1;i<=n;i++)

{

if (vx[i]) lx[i]-=d;

if (vy[i]) ly[i]+=d; else slack[i]-=d;

}

if (!linky[u]) {find\_route(u);return;}

vy[u]=true;vx[linky[u]]=true;q[++tail]=linky[u];

}

}

inline void KM()

{

memset(linkx,0,sizeof(linkx));memset(linky,0,sizeof(linky));

for (register int i=1;i<=n;i++) lx[i]=-INF,ly[i]=0;

for (register int i=1;i<=n;i++)

for (register int j=1;j<=n;j++)

lx[i]=max(lx[i],ga[i][j]);

for(register int i=1;i<=n;i++)

{

for (register int j=1;j<=n;j++) vx[j]=vy[j]=false,slack[j]=INF;

bfs(i);

}

}