**带权二分图的最佳匹配算法 - KM算法**

1. **KM算法求二分图的最佳匹配思想**

对于具有二部划分( V1, V2 )的加权完全二分图，其中 V1= { x1, x2, x3, ... , xn }， V2= { y1, y2, y3, ... , yn }，边< xi, yj >具有权值 Wi,j 。该带权二分图中一个总权值最大的完美匹配，称之为最佳匹配。

记 L(x) 表示结点 x 的标记量，如果对于二部图中的任何边<x,y>，都有 L(x)+ L(y)>= Wx,y，我们称 L 为二部图的**可行顶标**。

设 G(V,E) 为二部图， G'(V,E') 为二部图的子图。如果对于 G' 中的任何边<x,y> 满足， L(x)+ L(y)== Wx,y，我们称 G'(V,E') 为 G(V,E) 的等价子图。

**定理：设 L 是二部图 G 的可行顶标。若 根据L得到的等价子图 GL有完美匹配 M，则 M 是 G 的最佳匹配。**

证明：由于 GL 是 G 的等价子图，M 是 GL 的完美匹配，所以，M 也是 G  的完美匹配。由于对于匹配 M 的每条边 e ，都有 e∈ E( GL )，而且 M 中每条边覆盖每个顶点正好一次，所以

W( M )= Sum( W(e) ) = Sum ( L(x) ), 其中e∈ M , x∈ V

另一方面，对于 G 的任何完美匹配 M' 有

W( M' )= Sum( W(e) ) <= Sum ( L(x) ), 其中e∈ M' , x∈ V

于是 W( M )>= W( M' )，即 M 是 G 的最优匹配。

由上述定理，我们可以通过来不断修改可行顶标，得到等价子图，从而求出最佳匹配。

就像匈牙利算法一样，我们依次为每一个顶点 i 寻找增广路径，在寻找增广路经时路径上的边必须满足等价条件L(x)+ L(y)>= Wx,y. 如果寻找增广路径失败(找不到一条边满足等价条件)，我们就修改相应的可行顶标，来得到增广路径。

例如对于如下一个二分图：

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Y1** | **Y2** | **Y3** |
| **X1** | **1** | **2** | **3** |
| **X2** | **3** | **2** | **4** |
| **X3** | **2** | **3** | **5** |

若要对这个完全二分图求最佳匹配

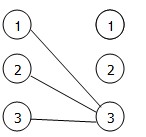
 初始化：

Lx(1)= max{ y| w(1,y), 1<= y<= 3 }= max{ 1, 2, 3 }= 3, Ly(1)= 0

Lx(2)= max{ 3, 2, 4 }= 4, Ly(2)= 0

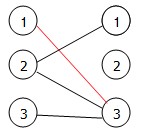
Lx(3)= max{ 2, 3, 5 }= 5, Ly(3)= 0;

我们建立等价子图( 满足 Lx(x)+ Ly(y)== W(x,y) ) 如下：



**图1**

对于该图，运用匈牙利算法对 X 部顶点 1 求增广路径(路径上的边满足等价条件)，得到一个匹配，如图( 红色代表匹配边 )：



**图2**

接着对 X 部顶点 2 按照DFS求增广路径，寻找增广路径的过程为 X 2-> Y 3-> X 1。到了X1后找不到能满足等价条件的路径了，因此增广路径寻找失败。我们把寻找增广路径失败的 DFS 的交错树中，在 X 部顶点集称之为 S， 在 Y 部的顶点集称之为 T。则 S= { 1, 2 }，T= { 3 }。现在我们就通过修改顶标值来扩大等价子图，如何修改。

1)   我们寻找一个 d 值，使得 d= min{ (x,y)| Lx(x)+ Ly(y)- W(x,y), x∈ S, y∉ T }，因些，这时 d= min{

Lx(1)+Ly(1)-W(1,1),  Lx(1)+Ly(2)-W(1,2),  Lx(2)+Ly(1)-W(2,1),  Lx(2)+Ly(2)-W(2,2) }=

min{ 3+0- 1, 3+0-2,  4+0-3,  4+0-2 }= min{ 2, 1, 1, 2 }= 1。

寻找最小的 d 是为了保证修改后仍满足性质对于边 <x,y> 有 Lx(x)+ Ly(y)>= W(x,y)。

2)   然后对于顶点 x

1. 如果 x∈ S 则 Lx(x)= Lx(x)- d。

2. 如果 x∈ T 则 Ly(x)= Ly(x)+ d。

3. 其它情况保持不变。

如此修改后，我们发现对于边<x,y>，顶标 Lx(x)+ Ly(y) 的值变为

1.  Lx(x)- d+ Ly(y)+ d，  x∈ S, y∈ T。

2.  Lx(x)+ Ly(y)，  x∉ S,  y∉ T。

3.  Lx(x)- d+ Ly(y)， x∈ S, y∉ T。

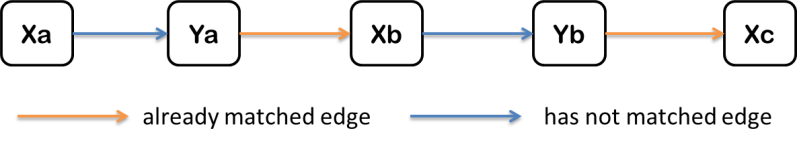
4.  Lx(x)+ Ly(y)+ d， x∉ S,  y∈ T。

易知，修改后对于任何边仍满足 Lx(x)+ Ly(y)>= W(x,y)。更重要的是：

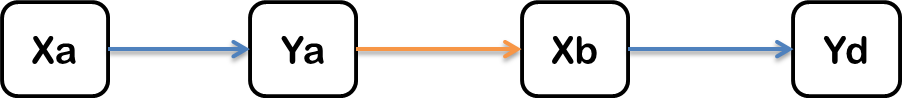
1. 根据上述1的修改，原来属于(S,T )的边不退出相等子图
2. 等价子图中至少会增加一个节点。这个节点是Y中的非T节点，且它就是造成d最小值的那个节点。比如X中的顶点a和Y中的非T顶点b使得上述d值最小 (即d = L(a) + L(b) – Wa, b)。那么根据上述3的修改，修改后L(a) + L(b) – Wa, b = 0。因此b顶点成为了等价子图中的一部分。

根据上述修改L值后，再次从顶点2开始重新寻找增广路径，这次一定能找到。

举个例子：假设X={Xa, Xb, Xc, Xd, Xe} Y = { Ya, Yb, Yc, Yd, Ye }. Xb和Ya进行了匹配，Xc已经和Yb进行了匹配。现在要为Xa寻找匹配。假设寻找匹配的过程中，增广路径到了Xc后无法继续扩展了。



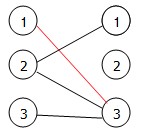
我们按照上述描述，求对{Xa, Xb, Xc} {Yc, Yd, Ye}这两个集合求最小d，假设最小d由Xb-Yd这条边得到。那么按照上述方法修改L后，Xb-Yd就会满足等价条件了 (L(b) + L(d) – WXb, Yd = 0). 我们再次从Xa开始寻找匹配，这时会成功找到下面这条增广路径：



按照匈牙利算法，将Xa与Ya匹配起来，将Xb与Yd匹配起来，从而完成了当前对Xa的匹配。

回到上述图2的例子，修改后 Lx(1)= 2, Lx(2)= 3, Lx(3)= 5, Ly(1)= 0, Ly(1)= 0, Ly(2)= 0, Ly(3)= 1。

这时 Lx(2)+Ly(1)=3+0=3= W(2,1)，在等价子图中增加了一条边，等价子图变为：



如此按以上方法，得到等价子图的完美匹配。

另外计算 d 值的时候可以进行一些优化，即在寻找增广路径的时候就求出最小d值，而没有必要在增广路径失败之后再去求最小d值。见后述文字。  
  
**2. KM算法的实现描述**（1）可行点标：每个点有一个标号，记lx[i]为X方点i的标号，ly[j]为Y方点j的标号。如果对于图中的任意边(i, j, W)都有lx[i]+ly[j]>=W，则这一组点标是**可行**的。特别地，对于lx[i]+ly[j]=W的边(i, j, W)，称为**可行边**；  
（2）KM 算法的核心思想就是通过修改某些点的标号（但要满足点标始终是可行的），不断增加图中的可行边总数，直到图中存在仅由可行边组成的完全匹配为止，此时这个 匹配一定是最佳的（因为由可行点标的的定义，图中的任意一个完全匹配，其边权总和均不大于所有点的标号之和，而仅由可行边组成的完全匹配的边权总和等于所 有点的标号之和，故这个匹配是最佳的）。一开始，求出每个点的初始标号：lx[i]=max{e.W|e.x=i}（即每个X方点的初始标号为与这个X方 点相关联的权值最大的边的权值），ly[j]=0（即每个Y方点的初始标号为0）。这个初始点标显然是可行的，并且，**与任意一个X方点关联的边中至少有一条可行边**；  
（3）然后，从每个X方点开始DFS增广。DFS增广的过程与最大匹配的Hungary算法基本相同，只是要注意两点：一是只找可行边，二是要把搜索过程中遍历到的X方点和Y方点全部记下来（可以用vst搞一下），以进行后面的修改。同时，对于增广过程中的不可行边，计算d值并update最小d值。  
（4） 增广的结果有两种：若成功（找到了增广轨），则该点增广完成，进入下一个点的增广。若失败（没有找到增广轨），则需要按照增广过程中求得的最小d值改变一些点的标号，使得图中可行边的 数量增加。方法为：将所有在增广轨中（就是在增广过程中遍历到）的X方点的标号全部减去d，所有在增广轨中的Y方点的标号全部加上d，则 对于图中的任意一条边(i, j, W)（i为X方点，j为Y方点）：  
<1>i和j都在增广轨中：此时边(i, j)的(lx[i]+ly[j])值不变，也就是这条边的可行性不变（原来是可行边则现在仍是，原来不是则现在仍不是）；  
<2>i在增广轨中而j不在：此时边(i, j)的(lx[i]+ly[j])的值减少了d，也就是原来这条边不是可行边（否则j就会被遍历到了），而现在可能是；  
<3>j在增广轨中而i不在：此时边(i, j)的(lx[i]+ly[j])的值增加了d，也就是原来这条边不是可行边（若这条边是可行边，则在遍历到j时会紧接着执行DFS(i)，此时i就会被遍历到），现在仍不是；  
<4>i和j都不在增广轨中：此时边(i, j)的(lx[i]+ly[j])值不变，也就是这条边的可行性不变。  
这 样，在进行了这一步修改操作后，图中原来的可行边仍可行，而原来不可行的边现在则可能变为可行边。  
（5）修改后，重新寻找增广路径。  
（6）以上就是KM算法的基本思路。时间复杂度为O(n3)   
  
**3. 求二分图的最小匹配**只需把权值取反，变为负的，再用KM算出最大权匹配，取反则为其最小权匹配。  
  
**4. 代码**  
#include <stdio.h>  
#include <string.h>  
#define M 310  
#define inf 0x3f3f3f3f  
  
int n,nx,ny;  
int link[M],lx[M],ly[M],slack[M];    //lx,ly为顶标，nx,ny分别为x点集y点集的个数  
int visx[M],visy[M],w[M][M];  
  
int DFS(int x)  
{  
    visx[x] = 1;  
    for (int y = 1;y <= ny;y ++)  
    {  
        if (visy[y])  
            continue;  
        int t = lx[x] + ly[y] - w[x][y];  
        if (t == 0)       //  
        {  
            visy[y] = 1;  
            if (link[y] == -1||DFS(link[y]))  
            {  
                link[y] = x;  
                return 1;  
            }  
        }  
        else if (slack[y] > t)  //不在相等子图中slack 取最小的  
            slack[y] = t;  
    }  
    return 0;  
}  
int KM()  
{  
    int i,j;  
    memset (link,-1,sizeof(link));  
    memset (ly,0,sizeof(ly));  
    for (i = 1;i <= nx;i ++)            //lx初始化为与它关联边中最大的  
        for (j = 1,lx[i] = -inf;j <= ny;j ++)  
            if (w[i][j] > lx[i])  
                lx[i] = w[i][j];  
  
    for (int x = 1;x <= nx;x ++)  
    {  
        for (i = 1;i <= ny;i ++)  
            slack[i] = inf;  
        while (1)  
        {  
            memset (visx,0,sizeof(visx));  
            memset (visy,0,sizeof(visy));  
            if (DFS(x))     //若成功（找到了增广轨），则该点增广完成，进入下一个点的增广  
                break;  //若失败（没有找到增广轨），则需要改变一些点的标号，使得图中可行边的数量增加。  
                        //方法为：将所有在增广轨中（就是在增广过程中遍历到）的X方点的标号全部减去一个常数d，  
                        //所有在增广轨中的Y方点的标号全部加上一个常数d  
            int d = inf;  
            for (i = 1;i <= ny;i ++)  
                if (!visy[i]&&d > slack[i])  
                    d = slack[i];  
            for (i = 1;i <= nx;i ++)  
                if (visx[i])  
                    lx[i] -= d;  
            for (i = 1;i <= ny;i ++)  //修改顶标后，要把所有不在交错树中的Y顶点的slack值都减去d  
                if (visy[i])  
                    ly[i] += d;  
                else  
                    slack[i] -= d;  
        }  
    }  
    int res = 0;  
    for (i = 1;i <= ny;i ++)  
        if (link[i] > -1)  
            res += w[link[i]][i];  
    return res;  
}  
int main ()  
{  
    int i,j;  
    while (scanf ("%d",&n)!=EOF)  
    {  
        nx = ny = n;  
      //  memset (w,0,sizeof(w));  
        for (i = 1;i <= n;i ++)  
            for (j = 1;j <= n;j ++)  
                scanf ("%d",&w[i][j]);  
        int ans = KM();  
        printf ("%d\n",ans);  
    }  
    return 0;  
}