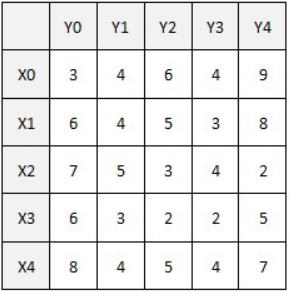
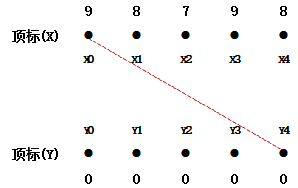
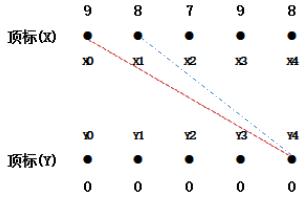
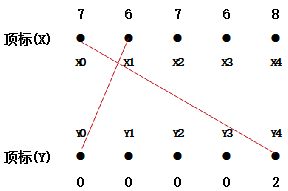
**[带权的二分图的最优匹配KM算法](http://blog.csdn.net/jarily/article/details/8617352)**

算法把权重转换成标杆，X集跟Y集的每个顶点各有一个标杆值，初始情况下权重全部放在X集上。由于每个顶点都将至少会有一个匹配点，贪心算法必然优先选择该顶点上权重最大的边（最理想的情况下，这些边正好没有交点，于是我们自然得到了最佳匹配）。最初的二分子图为：（可以看到初始化时X标杆为该顶点上的最大权重，而Y标杆为0）



从X0找增广路径，找到X0Y4；从X1找不到增广路径，也就是说，必须往二分子图里边添加新的边，使得X1能找到它的匹配，同时使权重总和添加最大。由于X1通往Y4而Y4已经被X0匹配，所以有两种可能，一个是为X0找一个新的匹配点并把Y4让给X1，或者是为X1找一个新的匹配点，现在我们将要看到标杆的作用了。根据传统的算法描述，能够进入二分子图的边的条件为L(x)+L(y)>=weight(xy)。当找不到增广路径时，对于搜索过的路径上的XY点，设该路径上的X顶点集为S，Y顶点集为T，对所有在S中的点xi及不在T中的点yj，计算d=min{(L(xi)+L(yj)-weight(xiyj))}，从S集中的X标杆中减去d，并将其加入到T集中的Y的标杆中，由于S集中的X标杆减少了，而不在T中的Y标杆不变，相当于这两个集合中的L(x)+L(y)变小了，也就是，有新的边可以加入二分子图了。从贪心选边的角度看，我们可以为X0选择新的边而抛弃原先的二分子图中的匹配边，也可以为X1选择新的边而抛弃原先的二分子图中的匹配边，因为我们不能同时选择X0Y4和X1Y4，因为这是一个不合法匹配，这个时候，d=min{(L(xi)+L(yj)-weight(xiyj))}的意义就在于，我们选择一条新的边，这条边将被加入匹配子图中使得匹配合法，选择这条边形成的匹配子图，将比原先的匹配子图加上这条非法边组成的非法匹配子图的权重和（如果它是合法的，它将是最大的）小最少，即权重最大了。好绕口的。用数学的方式表达，设原先的不合法匹配（它的权重最大，因为我们总是从权重最大的边找起的）的权重为W，新的合法匹配为W’，d为min{W-W’i}。在这个例子中，S={X0, X1}，Y={Y4}，求出最小值d=L(X1)+L(Y0)-weight(X1Y0)=2，得到新的二分子图：

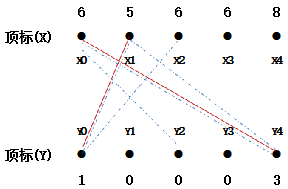
 

重新为X1寻找增广路径，找到X1Y0，可以看到新的匹配子图的权重为9+6=15，比原先的不合法的匹配的权重9+8=17正好少d=2。

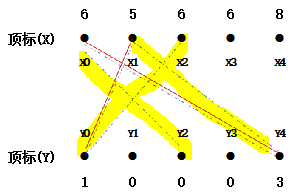
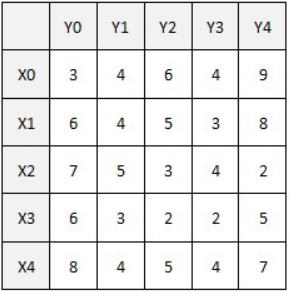
接下来从X2出发找不到增广路径，其走过的路径如蓝色的路线所示。形成的非法匹配子图：X0Y4，X1Y0及X2Y0的权重和为22。在这条路径上，只要为S={X0，X1，X2}中的任意一个顶点找到新的匹配，就可以解决这个问题，于是又开始求d。

d=L(X0)+L(Y2)-weight(X0Y2)=L(X2)+L(Y1)-weight(X2Y1)=1.

新的二分子图为：



重新为X2寻找增广路径，如果我们使用的是深搜，会得到路径：X2Y0->Y0X1->X1Y4->Y4X0->X0Y2，即奇数条边而删除偶数条边，新的匹配子图中由这几个顶点得到的新的权重为21；如果使用的是宽搜，会得到路径X2Y1，另上原先的两条匹配边，权重为21。假设我们使用的是宽搜，得到的新的匹配子图为：



接下来依次类推，直到为X4找到一个匹配点。

KM算法是通过给每个顶点一个标号（叫做顶标）来把求最大权匹配的问题转化为求完备匹配的问题的;

设顶点Xi的顶标为A[i]，顶点Yi的顶标为B[i]，顶点Xi与Yj之间的边权为w[i,j];

在算法执行过程中的任一时刻，对于任一条边(i,j)，A[i]+B[j]>=w[i,j]始终成立;

初始A[i]为与Xi相连的边的最大边权，B[j]=0;

KM算法的正确性基于以下定理：

设G(V,E)为二分图,G'(V,E')为该二分图的子图;

如果对于G'中的任何边<x,y>满足， A(x)+ B(y)==W[x,y];

则称G'(V,E')为G(V,E)的等价子图或相等子图(是G的生成子图);

若由二分图中所有满足A[i]+B[j]=w[i,j]的边(i,j)构成的子图（称做相等子图）有完备匹配;

那么这个完备匹配就是二分图的最大权匹配;

因为对于二分图的任意一个匹配，如果它包含于相等子图;

那么它的边权和等于所有顶点的顶标和；

如果它有的边不包含于相等子图，那么它的边权和小于所有顶点的顶标和(即不是最优匹配);

所以相等子图的完备匹配一定是二分图的最大权匹配;

相等子图包含原图的所有的点，相等子图一定可以找到完备匹配;

相等子图的完备匹配只需加一些虚拟点可以扩充为完美匹配(记为M);

完美匹配是包含了所有点的匹配，那么所有点的顶点的标号值都包括进来了;

虽然有些点是0，在这个状态下，把相等子图的标号一一对应的标到原图上去;

原图的任意一个匹配最多只能包含原图的所有顶点;

即任何匹配的权和不可能超过所有标号的和，所以M的和必然是最优的;

#include <cstdio>

#include <string.h>

#include <vector>

#include <algorithm>

using namespace std;

int const MAX = 1000;

int const inf = 0x3fffffff;

int w[MAX][MAX];

int link[MAX];//代表当前与Y集合中配对的X集合中的点

int visx[MAX], visy[MAX];

int lx[MAX], ly[MAX];

int n, m;//代表X和Y中元素的个数

int can(int t)

{

visx[t] = 1;

for(int i = 1; i <= m; i++){

if(!visy[i] && lx[t] + ly[i] == w[t][i]){//这里“lx[t]+ly[i]==w[t][i]”决定了这是在相等子图中找增广路的前提，非常重要

visy[i] = 1;

if(link[i] == -1 || can(link[i])){

link[i] = t;

return 1;

}

}

}

return 0;

}

int km()

{

int sum = 0;

memset(ly, 0, sizeof(ly));

for(int i = 1; i <= n; i++){//把各个lx的值都设为当前w[i][j]的最大值

lx[i] = -inf;

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

if(lx[i] < w[i][j])

lx[i] = w[i][j];

}

}

memset(link, -1, sizeof(link));

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

while(1){

memset(visx, 0, sizeof(visx));

memset(visy, 0, sizeof(visy));

if(can(i))//如果它能够形成一条增广路径，那么就break

break;

int d = inf;//否则，后面应该加入新的边,这里应该先计算d值

for(int j = 1; j <= n; j++)//对于搜索过的路径上的XY点，设该路径上的X顶点集为S，Y顶点集为//T，对所有在S中的点xi及不在T中的点yj

if(visx[j])

for(int k = 1; k <= m; k++)

if(!visy[k])

d = min(d, lx[j] + ly[k] - w[j][k]);

if(d == inf)

return -1;//找不到可以加入的边，返回失败（即找不到完美匹配）

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

if (visx[j])

lx[j] -= d;

for(int j = 1; j <= m; j++)

if(visy[j])

ly[j] += d;

}

}

for(int i = 1; i <= m; i++)

if(link[i] > -1)

sum += w[link[i]][i];

return sum;

}

算法改进：

给每个Y顶点一个"松弛量"函数slack;

每次开始找增广路时初始为无穷大;

在寻找增广路的过程中，检查(i,j)时，如果它不在相等子图中;

则让slack[j]=min(原值,A[i]+B[j]-W[i,j]);

这样在修改顶标时，取所有的不在交错树中的Y顶点的slack值中的最小值作为d值即可;

算法过程：

①初始化可行顶标的值;

②用匈牙利算法寻找完备匹配;

③若未找到完备匹配则修改可行顶标的值;

④重复②③直到找到相等子图的完备匹配;

#include<iostream>

#include<cstring>

#include<cstdlib>

#include<cstdio>

#include<climits>

#include<algorithm>

using namespace std;

const int N = 1000;

const int INF = 0xffffff;

int w[N][N];//权值

int lx[N],ly[N]; //顶标

int linky[N];//记录与i匹配的顶点

int visx[N],visy[N];

int slack[N];//松弛量

int nx,ny;//二分图两边的顶点数

void init(){

memset(linky,-1,sizeof(linky));//记录与i匹配的顶点

memset(ly,0,sizeof(ly));///初始化顶标y为0

for(int i = 0; i < nx; i++)

for(int j = 0,lx[i] = -INF; j < ny; j++){

if(w[i][j] > lx[i])

lx[i] = w[i][j];///初始化顶标x为与顶点Xi关联的边的最大权

}

}

bool find(int x) { //匈牙利算法

visx[x] = true;

for(int y = 0; y < ny; y++) {

if(visy[y])

continue;

int t = lx[x] + ly[y] - w[x][y];//若t==0，则为最大权匹配；

if(t==0) {

visy[y] = true;

if(linky[y]==-1 || find(linky[y])) {

linky[y] = x;

return true; //找到增广轨

}

}

else if(slack[y] > t)

slack[y] = t;

}

return false; //没有找到增广轨（说明顶点x没有对应的匹配，与完备匹配(相等子图的完备匹配)不符）

}

int KM(){ //返回最优匹配的值

init();

for(int x = 0; x < nx; x++) {

for(int i = 0; i < ny; i++)

slack[i] = INF;//松弛函数初始化为无穷大

while(1) {

memset(visx,0,sizeof(visx));

memset(visy,0,sizeof(visy));

if(find(x)) //找到增广轨，退出

break;

int d = INF;

for(int i = 0; i < ny; i++){//没找到，对l做调整(这会增加相等子图的边)，重新找

if(!visy[i] && d > slack[i])

d = slack[i];

}

for(int i = 0; i < nx; i++){//修改x的顶标

if(visx[i])

lx[i] -= d;

}

for(int i = 0; i < ny; i++){//修改y的顶标

if(visy[i])

ly[i] += d;

else

slack[i] -= d;//修改顶标后，不在交错树中的y顶点的slack值都要减去d；

}

}

}

int result = 0;

for(int i = 0; i < ny; i++){

if(linky[i]>-1)

result += w[linky[i]][i];

}

return result;

}

int main(){

//freopen("C:\\Users\\Administrator\\Desktop\\kd.txt","r",stdin);

while(~scanf("%d%d",&nx,&ny)){

if(!nx||!ny)

break;

int a,b,c;

while(scanf("%d%d%d",&a,&b,&c),a+b+c){

w[a][b]=c;

}

printf("%d\n",KM());

}

return 0;

}

例题

# URAL1076.Trash（垃圾）——最佳匹配

1076. Trash

Time limit: 1.0 second

Memory limit: 64 MB

You were just hired as CEO of the local junkyard.One of your jobs is dealing with the incoming trash and sorting it for recycling.The trash comes every day in N containers and each of these containers contains certain amount of each of the N types of trash. Given the amount of trash in the containers find the optimal way to sort the trash. Sorting the trash means putting every type of trash in separate container. Each of the given containers has infinite capacity. The effort for moving one unit of trash from container i to j is 1 if i ≠ j otherwise it is 0.You are to minimize the total effort.

Input

The first line contains the number N (1 ≤ N ≤ 150), the rest of the input contains the descriptions of the containers.The (1 + i)-th line contains the description of the i-th container the j-th amount (0 ≤ amount ≤ 100) on this line denotes the amount of the j-th type of trash in the i-th container.

Output

You should write the minimal effort that is required for sorting the trash.

Sample

input

4

62 41 86 94

73 58 11 12

69 93 89 88

81 40 69 13

Output

650

问题分析：

     有n个垃圾桶，每个垃圾桶内有n种垃圾，现在是要你把垃圾分类，到每一个垃圾桶最后都只能装一种垃圾，从一个垃圾桶里把垃圾移到另一个垃圾桶会有所消耗，且都是单位消耗。。

     那么，要总消耗最少，就要原来每个垃圾桶各自保留的垃圾总和要最多，这么看来，问题就可以转化为带权二分图的最佳匹配问题。

      求出这个最佳匹配  k   最后 ans = s（所有垃圾总和）-k

#include <cstdio>

#include <string.h>

#include <algorithm>

using namespace std;

const int MAX = 200;

const int inf = 0x3f3f3f3f;

int w[MAX][MAX];

int lx[MAX], ly[MAX];

int visx[MAX], visy[MAX];

int link[MAX];

int n;

int can(int t)

{

visx[t] = 1;

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

if(!visy[i] && lx[t] + ly[i] == w[t][i]){

visy[i] = 1;

if(link[i] == -1 || can(link[i])){

link[i] = t;

return 1;

}

}

}

return 0;

}

int km(){

int d;

int sum = 0;

memset(ly, 0, sizeof(ly));

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

lx[i] = -inf;

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

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

}

memset(link, -1, sizeof(link));

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

while(1){

memset(visx, 0, sizeof(visx));

memset(visy, 0, sizeof(visy));

if(can(i)) break;

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

if(visx[j])

for(int k = 1; k <= n; k++)

if(!visy[k])

d = min(d, lx[j] + ly[k] - w[j][k]);

if(d == inf)

return -1;

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

if(visx[j])

lx[j] -= d;

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

if(visy[j])

ly[j] += d;

}

}

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

if(link[i] > -1)

sum += w[link[i]][i];

return sum;

}

int main(){

int ans = 0;

scanf("%d", &n);

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

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

scanf("%d", &w[i][j]);

ans += w[i][j];

}

ans -= km();

printf("%d\n", ans);

return 0;

}