// DP 状态压缩 TSP

#include<iostream>

#include<cstdio>

#include<cstring>

#include<algorithm>

#include<queue>

#include<map>

#include<set>

#include<stack>

using namespace std;

typedef long long LL;

int n;

int dis[20][20];

int dp[20][1<<16];

int min(int x,int y)

{

if (x==-1) return y;

if (y==-1) return x;

if (x<=y) return x;

return y;

}

int main()

{

while(scanf("%d",&n) && n)

{

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

for(int j=0;j<=n;j++)

cin>>dis[i][j];

for(int k=0;k<=n;k++) /// floyd

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

for(int j=0;j<=n;j++)

if (i!=j && j!=k && i!=k)

dis[i][j]=min(dis[i][j],dis[i][k]+dis[k][j]);

n++;

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

dp[0][0]=0;

for(int state=0;state<(1<<n);state++) /// 2的n次方 从 0000 到 1111

{

for(int i=0;i<n;i++) /// 0 1 2 3

if (dp[i][state]!=-1) /// 优化 不可能的情况: 没有从i开始走

{

for(int j=0;j<n;j++) /// 枚举state每一位 到达的 j

if (((1<<j)&state)==0) /// == 0 可以到达 j 的state状态为 0

{

dp[j][state|(1<<j)]=min(dp[j][state|(1<<j)], dp[i][state]+dis[i][j]);/// 走 为正的 重载min

/// 到达j点 state|(1<<j) 的状态 当前集合S中的元素已经全部经历过的最短路径

cout << i << ' ' << j << ' ' << (state|(1<<j)) << ' ' << dp[j][state|(1<<j)] << endl;

}

}

}

int ans=dp[0][(1<<n)-1]; /// 到达0点并且全部为1( 访问过 )

cout<<ans<<endl;

}

}

/\*

3

0 1 10 10

1 0 1 2

10 1 0 10

10 2 10 0

样例输出 Sample Output

8

数据范围及提示 Data Size & Hint

1<=n<=15

思路：

首先题目给出了邻接矩阵，而且还说每个点可以访问多次，那么我们考虑两点间的距离便只考虑最短距离，不考虑怎么到的方式和路径 选择Floyd 算法求出两点间的最短距离

题目数据量较小 先考虑搜索 那么解答树将会是n个节点的全排列 数量达到 n！而且不好剪枝 而且这样想还有一定的问题 题目上说一个点可以经历很多次以达到最短路 那么解答树将不是排列 而是集合的形式 那么搜索就很难解决这道题 并且很明显感到有重叠子问题 那么开始考虑动归

如果用dp的话 要考虑状态数量和状态的划分和表示 那么由搜索的猜想可以感觉到需要集合 状态可能就要表示为当前集合S中的元素已经全部经历过的最短路径

并且应该记下来集合S中最后一个访问的元素是什么 因为这样的话可以方便的递推出下一个状态最终解决问题 并且题目给出两点最短距离 记下了最后一个访问的点 每一个状态就可以由上一个状态得出 那么状态转移方程：

dp [ 集合S(不包含j) ] [ 要访问的点j ] (表示集合S中的点已经访问，且最后访问的是j的最短路长度)

= min { dp [ 已经访问过的集合S‘ (S’中没有点j，也没有点i) ] [ S‘集合中最后一个访问的点是 i ] +dis [ i ] [ j ] }

其中i属于S枚举(ATT!！ i属于S 不属于S‘ 这里要多理解)就好 就可以开始递推了

并且问题的答案 就应该在dp[集合S包括除了0的所有点][最后一个访问的是0号点]

dp的实现：

首先考虑集合如何表示 如果用数组的话 那么数量是惊人的 因为每个点可以重复 并且无法估计需要的大小 集合 集合有一个性质就是无序性 集合中有 只代表有 没有顺序 这也解释了为什么上文说要记下来最后一个访问的点 那么选择用二进制来表示和压缩状态

#include <iostream>

#include <cstring>

#include <cstdio>

#include <vector>

using namespace std;

#define N 1<<16

#define nn 16

int d[nn][nn],f[N][nn];

int main () {

//状态压缩 二进制位表示对应状态 编号n对应的二进制位为1<<n 那一位上是1表示遍历过 是0表示还没有遍历

//f[s][i]表示已经遍历的元素在s（二进制数）中 i为最后遍历的编号 则f[s][i]的值表示s中的元素已经遍历完，且刚访问完i的最短路长度

int n;//节点数量

cin>>n;

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

for (int j=0;j<=n;j++)

cin>>d[i][j];//输入已经给出的两个节点的距离

for (int k=0;k<=n;k++)

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

for (int j=0;j<=n;j++){

if (i!=k&&j!=k&&i!=j){

d[i][j]=min(d[i][j],d[i][k]+d[k][j]);//Floyd 求出i到j的最短路长度记在d[i][j]中

}

}

int maxx=(1<<(n+1))-1; //最大的状态数 即(1<<(n+1))-1 对应的二进制数(n位)上每一位为1

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

for (int j=0;j<nn;j++)

f[i][j]=0x6ffffff;//初始化 所有情况的距离初始化为最大值

for (int i=0;i<=n;i++) f[0][i]=d[0][i];

初始化 s集合中没有点 只刚遍历完i点的最短路值即为d[0][i](0号点到i的最短路) (这是动归的基础起点与边界) 因为题目要求从0号点出发

也是为什么答案在f[集合s有除了0号点的所有点][0] 代表所有点全部访问完并且最后一个访问的是0号点

符合题目所说从0出发访问再回到0的最短距离

for (int s=1;s<=maxx;s++)//枚举所有状态

for (int j=0;j<=n;j++)//访问还不在s中的点j 去更新f

if (!((1<<j)&s)){//如果j在s里 不访问

for (int i=0;i<=n;i++){//从s中的点（i属于集合s） 出发去更新到j的距离

if (s&(1<<i)){//i在s中

f[s][j]=min(f[s][j],f[s^(1<<i)][i]+d[i][j]);//异或即可求出补集（或者叫集合的减法）s^(1<<i) 表示把s集合中的i去掉

}

}

}

//int mmax=0x7fffffff;

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

// mmax=min(mmax,f[maxx^(1<<i)][i]+d[i][0]);

//cout<<mmax<<"\n";//如果下面的最终结果不理解 尝试理解下这一种输出 其实两者是一致的一样的 可以帮助你理解下面

cout<<f[maxx^1][0]; //s集合中有除0号点以外的所有元素，最后一个访问的元素为0号 即为所求

return 0;

}

\*/