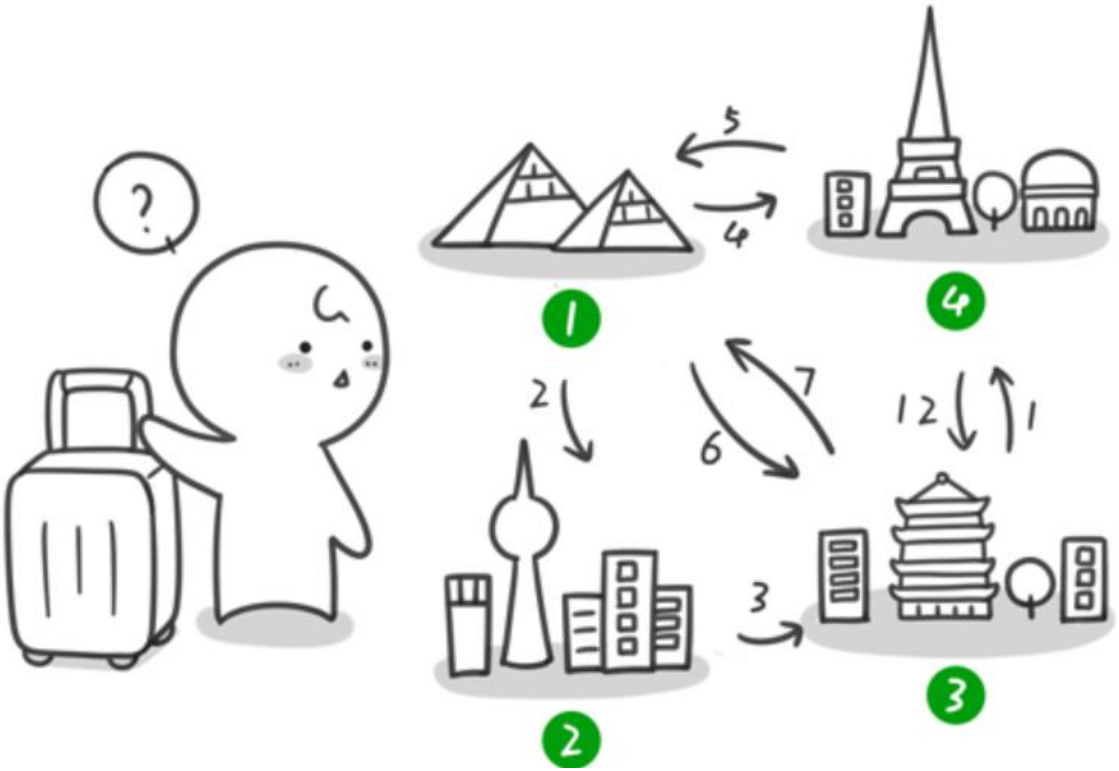
高级数据结构-Floyd（弗洛伊德）算法

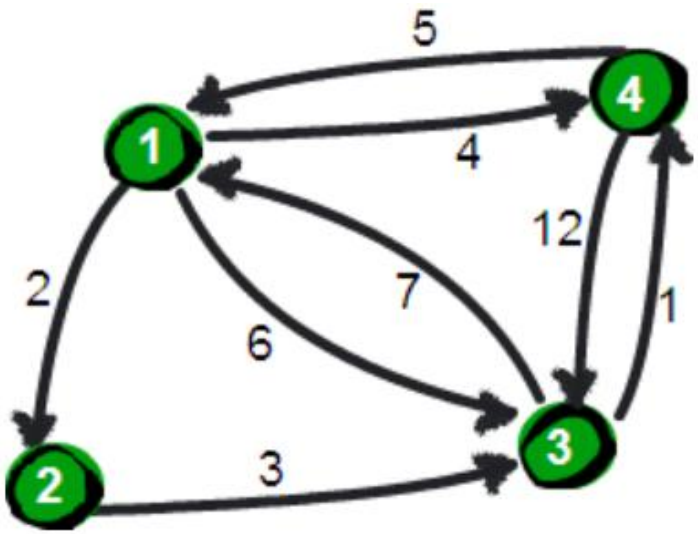
信智学院　陈琰宏

小哼准备去一些城市旅游。有些城市之间有公路，有些城市之间则没有，如下图。为了节省经费以及方便计划旅程，小哼希望在出发之前知道任意两个城市之前的最短路程。



Floyd算法

Floyd 是解决任意两点间最短路径(称为多源最短路径问题)的经典算法，可以正确处理有向图或负权的最短路径问题。



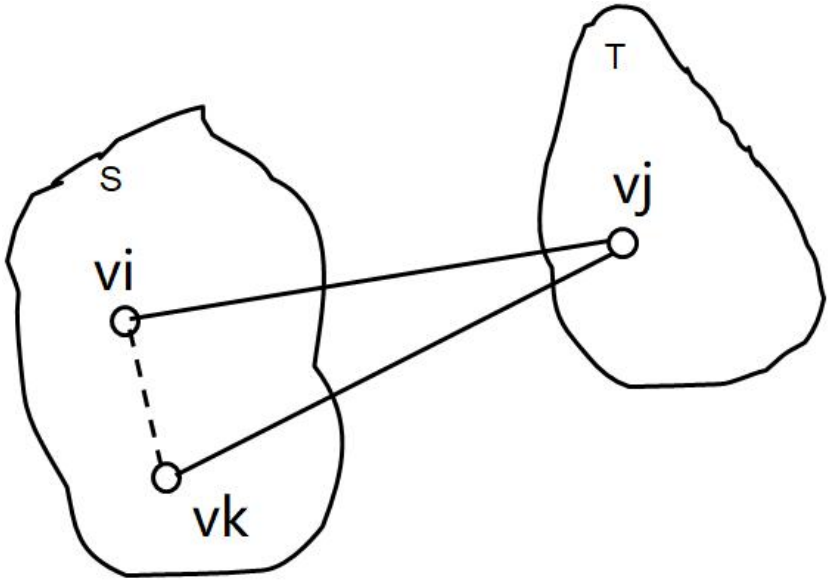
	1	2	3	4
1	0	2	6	4
2	∞	0	3	∞
3	7	∞	0	1
4	5	∞	12	0

算法分析

从任意节点i到任意节点j的最短路径不外乎2种可能：

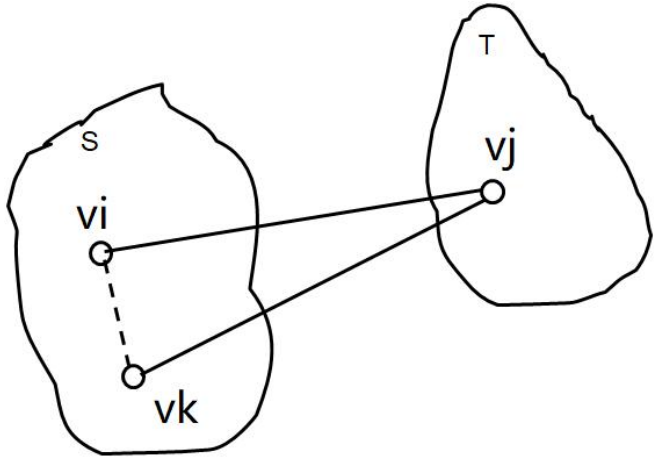
1) 直接从节点i到节点j。

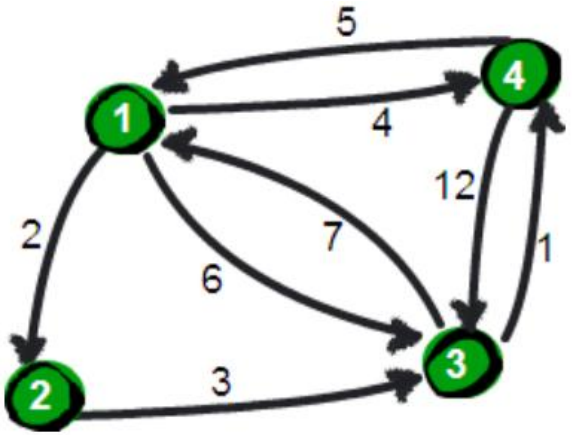
2) 从节点i经过若干个节点k到节点j。



可以证明vi到v_j的最短路径，要么是从vi到v_j的直接路径；要么是从v_i经某个顶点 v_k 再到v_j的路径。

Floyd-Warshall算法





递推公式：

初始：*dist* [i][j] = Edge[i][j]，v_i是源点

递推：*dist* [i][j] = min{ *dist* [i][j], *dist* [i][k] + dis[k][j] }, $v_k \in T$

其中v_u是当前dist[ ]最小的顶点。

算法的具体步骤

```
1  for(k=1;k<=n;k++)//经过中间K点
2      for(i=1;i<=n;i++) //源点
3      for(j=1;j<=n;j++)  //目标点
4      if(dis[i][j]>dis[i][k]+dis[k][j])
5          dis[i][j]=dis[i][k]+dis[k][j];
```

```
4 8
1 2 2
1 3 6
1 4 4
2 3 3
3 1 7
3 4 1
4 1 5
4 3 12
```

```
0    2    5    4
9    0    3    4
6    8    0    1
5    7   10    0
```

```
//Floyd算法核心语句
for(k=1;k<=n;k++)
for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
if(e[i][j]>e[i][k]+e[k][j] )
    e[i][j]=e[i][k]+e[k][j];
//输出最终的结果
for(i=1;i<=n;i++) {
    for(j=1;j<=n;j++)
        printf("%7d",e[i][j]);
    printf("\n");
}
```

1 [2896]牛的旅行

农民John的农场里有很多牧区。有的路径连接一些特定的牧区。一片所有连通的牧区称为一个牧场。但是就目前而言，你能看到至少有两个牧区不连通。现在，John想在农场里添加一条路径（注意，恰好一条）。对这条路径有这样的限制：一个牧场的直径就是牧场中最远的两个牧区的距离（本题中所提到的所有距离指的都是最短的距离）。考虑如下的两个牧场，图1是有5个牧区的牧场，牧区用“*”表示，路径用直线表示。每一个牧区都有自己的坐标：

图1所示的牧场的直径大约是12.07106，最远的两个牧区是A和E，它们之间的最短路径是A-B-E。

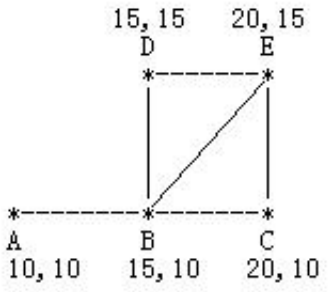


图1

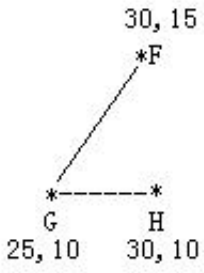


图2

[2896]牛的旅行

这两个牧场都在John的农场上。John将会在两个牧场中各选一个牧区，然后用一条路径连起来，使得连通后这个新的更大的牧场有最小的直径。注意，如果两条路径中途相交，我们不认为它们是连通的。只有两条路径在同一个牧区相交，我们才认为它们是连通的。

现在请你编程找出一条连接两个不同牧场的路径，使得连上这条路径后，这个更大的新牧场有最小的直径。

输入描述：

第 1 行：一个整数N (1 ≤ N ≤ 150), 表示牧区数；

第 2 到 N+1 行：每行两个整数X，Y (0 ≤ X，Y≤ 100000)，表示N个牧区的坐标。每个牧区的坐标都是不一样的。

第 N+2 行到第 2*N+1 行：每行包括N个数字 (0或1) 表示一个对称邻接矩阵。

例如，题目描述中的两个牧场的矩阵描述如下：

A B C D E F G H

A 0 1 0 0 0 0 0 0

B 1 0 1 1 1 0 0 0

C 0 1 0 0 1 0 0 0

D 0 1 0 0 1 0 0 0

E 0 1 1 1 0 0 0 0

F 0 0 0 0 0 0 1 0

G 0 0 0 0 0 1 0 1

H 0 0 0 0 0 0 1 0　输入数据中至少包括两个不连通的牧区。

输出描述：

每组输出只有一行，包括一个实数，表示所求答案。数字保留六位小数。

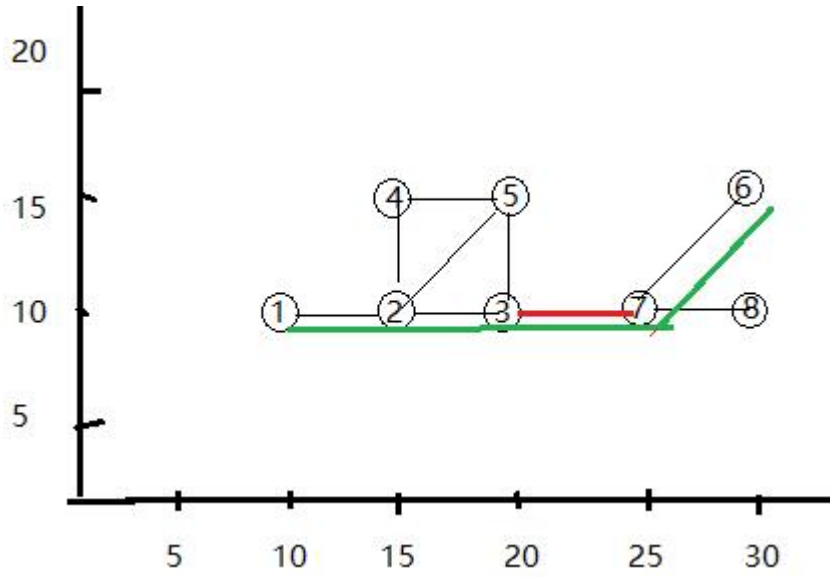
[2896]牛的旅行

样例输入：

8
10 10
15 10
20 10
15 15
20 15
30 15
25 10
30 10
01000000
10111000
01001000
01001000
01110000
00000010
00000101
00000010

样例输出：

22.071068



3和7之间添一条边
直径：1 2 3 6 7

分析

题意：在一个有n个节点的无向图中，有若干个联通块，并且至少有两个联通块之间是不连通的，节点称为牧区，联通块称为牧场，牧场的直径为这个联通块中两个节点之间路径长度的最大值，整张图的直径为各个联通块直径的最大值。

题目要求：我们添加一条边，使两个本来不连通的联通块能够连通。那么就产生了一个新的牧场和这个牧场的直径，我们需要使加入了一条边后的图的直径最小。

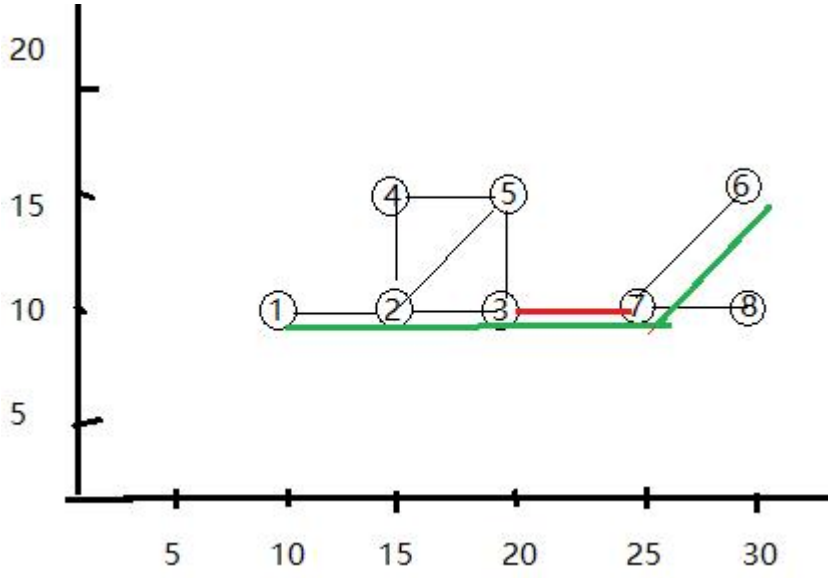
处理：我们先处理原图中各个联通块的直径，题目要求为最短距离，就可以用floyd算出各个联通节点之间的最小距离，再求以每个节点节点为端点的路径长度最大值。

再处理加入一条边后产生的直径（加入边的长度+一个节点为端点的路径最大值+另一个节点为端点的路径最大值）的最小值，最后在原图的直径与新产生的直径中取大的那个即为所求的直径最小值。

分析

1. 根据坐标算出边的权值（两节点之间的距离）
2. 用floyd算出任意两点之间的最短距离
3. 算出以每个点为一端的所有路径中最长距离
4. 算出添加一条边后产生的新直径的最小值（新增边的长度加上两个端点的路径长度最大值）

5取原有的最大直径与新直径的较大值



各个点为端点的路径长度最大值

1：12.071068　1 - > 5
2：7.071068　2 -> 5
3：10　3 -> 1或 3 -> 4
4：10　4 -> 1或 4 -> 3
5：12.071068　5 - >1
6：12.071068　6 - >7
7：7.0701068　7 - >6
8：12.071068　8 - >6

```cpp
#include<bits/stdc++.h>
using namespace std;
const int N=155;
const double inf=0x3f3f3f3f; //无穷大
char g[N][N];  //存邻接矩阵（无向图）
int n;
double d[N][N],m[N]; //d 为任意两节点之间的距离，m为以每个节点为一端的所有路径中长度最大值
struct node{
    int x,y; //存每个点的坐标
}q[N];
double get(node a,node b){    //计算两个节点（有边直接相连）的距离
    double dx=a.x-b.x,dy=a.y-b.y;
    return sqrt(dx*dx+dy*dy);
}
int main(){
    cin>>n;
    for(int i=0;i<n;i++)
        cin>>q[i].x>>q[i].y; //输入每个节点的坐标
    for(int i=0;i<n;i++)
        cin>>g[i];  //输入邻接矩阵（即两个节点之间是否有边直接相连）
    for(int i=0;i<n;i++)
    {
        for(int j=0;j<n;j++)
        {
            if(i!=j)  //若i=j，则为自己到自己，即距离为0（全局变量默认为0）
            {
                if(g[i][j]=='1') d[i][j]=get(q[i],q[j]); //‘1’表示两节点之间有边，则算出这两节点之间的距离
                else d[i][j]=inf;    //‘0’表示两点之间没有边直接相连，距离为无穷大
```

```
            if(i!=j)  //若i=j，则为自己到自己，即距离为0（全局变量默认为0）
            {
                if(g[i][j]=='1') d[i][j]=get(q[i],q[j]); //‘ 1’ 表示两节点之间有边，则算出这两节点之间的距离
                else d[i][j]=inf;    //‘ 0’ 表示两点之间没有边直接相连，距离为无穷大
            }
        }
    }
    for(int k=0;k<n;k++)  //floyd 算各个节点（联通的节点）之间的最短距离
       for(int i=0;i<n;i++)
          for(int j=0;j<n;j++)
             d[i][j]=min(d[i][j],d[i][k]+d[k][j]);
    for(int i=0;i<n;i++)
       for(int j=0;j<n;j++)
          if(d[i][j]<inf)  //当i和j这两个节点之间联通时
            m[i]=max(m[i],d[i][j]); //求以i节点为一端的所有路径中长度最大值 （另一端遍历除i之外的节点）
    double ans1=-1; //表示在原图中各个牧场的直径的最大值
    for(int i=0;i<n;i++)
       ans1=max(m[i],ans1);
    double ans2=inf; //表示添加一条边产生的新的牧场的直径 （求新产生直径的最小值）
    for(int i=0;i<n;i++)
       for(int j=0;j<n;j++)
          if(d[i][j]>=inf) //添加边的前提当然是这条边原来是不存在的（即两个节点之间不连通）
            ans2=min(ans2,get(q[i],q[j])+m[i]+m[j]);//添加后产生的直径为两个节点之间的直接距离加上以两个节点为一段的路径长度最大值
    double ans=max(ans1,ans2); // 原来的直径最大值与新牧场的直径取大的那个值
    printf("%f",ans); //输出小数点后六位
    return 0;
}
```

2 [3574]sightseeing trip

题目描述：

在Adelton城有一个桑给巴尔岛的旅行社。它已决定提供它的客户，其中一些景点，观光小镇。为了赚取尽可能多的钱，该机构接受了一个精明的决定：有必要找到最短的路线，开始和结束在同一个地方。你的任务是写一个程序，找到这样的路线。

镇上有n个景点，编号从1到N和M个双向道路编号从1到M。

两个景点可以通过多条道路连接，但没有道路连接本身景点。每一条观光路线是一个有序的道路编号，道路数k大于2。观光路线是无重复景点的环路。观光路线的长度是所有路线长度之和。您的程序必须找到长度最小的观光路线，或指定它是不可能的。

输入：

第一行给出N,M。 N<=100， M<=10000

接下来M行，前两个数为结点，后一个数为权值

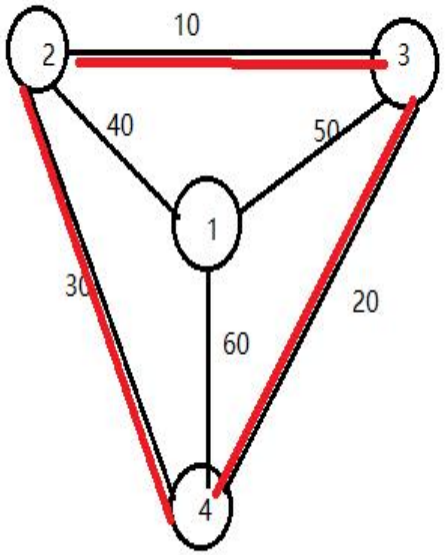
输出：

路径，没有则输出 “No solution.”

2 [3574]sightseeing trip

样例输入：

4 6
1 2 40
1 3 50
1 4 60
2 3 10
2 4 30
3 4 20



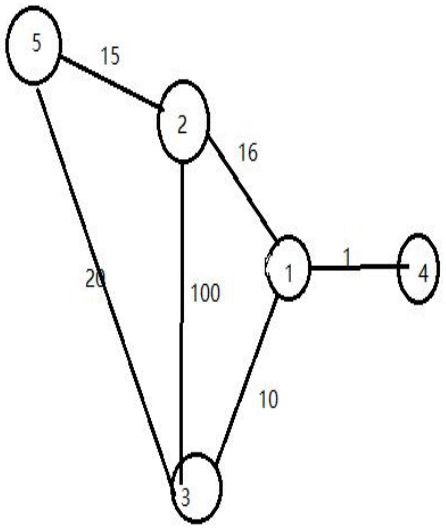
样例输出：

2 3 4

2 [3574]sightseeing trip

输入样例：

5 7
1 4 1
1 3 300
3 1 10
1 2 16
2 3 100
2 5 15
5 3 20



初始状态

	1	2	3	4	5
1	0	16	10	1	∞
2	16	0	100	∞	15
3	10	100	0	∞	20
4	1	∞	∞	0	∞
5	∞	15	20	∞	0

插入中间节点1

	1	2	3	4	5
1	0	16	10	1	∞
2	16	0	26	17	15
3	10	26	0	11	20
4	1	17	11	0	∞
5	∞	15	20	∞	0

插入中间节点2

	1	2	3	4	5
1	0	16	10	1	31
2	16	0	26	17	15
3	10	26	0	11	20
4	1	17	11	0	32
5	31	15	20	32	0

插入中间节点3

	1	2	3	4	5
1	0	16	10	1	31
2	16	0	26	17	15
3	10	26	0	11	20
4	1	17	11	0	32
5	31	15	20	32	0

插入中间节点4

	1	2	3	4	5
1	0	16	10	1	30
2	16	0	26	17	15
3	10	26	0	11	20
4	1	17	11	0	31
5	30	15	20	31	0

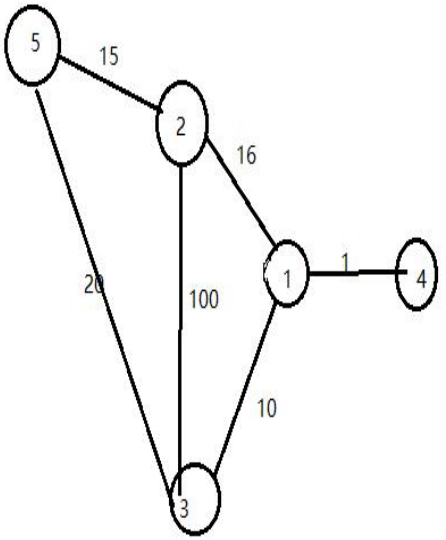
输出样例：

1 3 5 2

思路：
用floyed算法

```
void get(int i,int j)
{
    if(!pos[i][j])return;//若pos为0，表示i与j之间没有中间节点，即i与j直接相连
    get(i,pos[i][j]);//i到中间节点
    path[cnt++]=pos[i][j];//将节点存入path中
    get(pos[i][j],j); //中间节点到j
    return;
}
```

```
32  for(int k=1;k<=n;k++)
33  {
34      for(int i=1;i<k;i++)   //保证i和j都小于k，则可保证d[i][j]中i到j的路径中没有k
35      for(int j=i+1;j<k;j++)
36      {
37          if((long long)dis[i][j]+g[j][k]+g[k][i]<ans)//long long 防止爆int
38          {
39              ans=dis[i][j]+g[j][k]+g[k][i];//更新环的长度
40              cnt=0; //既然有比上一个环长度更小的环，则重新找这个环上的节点
41              path[cnt++]=i; //存节点
42              get(i,j);
43              path[cnt++]=j;
44              path[cnt++]=k;
45          }
46      }
47      for(int i=1;i<=n;i++)
48      for(int j=1;j<=n;j++)
49      {
50          if((long long)dis[i][k]+dis[k][j]<dis[i][j])//若以k为中间节点，i与j能相连
51          {
52              dis[i][j]=dis[i][k]+dis[k][j];//更新
53              pos[i][j]=k; //k为i到j的路径的中间节点
54          }
55      }
56  }
```



ans：126
节点：1 2 3

ans：61
节点：1 2 3 5

```cpp
#include<bits/stdc++.h>
using namespace std;
const int N=105;
const int inf=0x3f3f3f3f;
int g[N][N],dis[N][N],path[N],cnt,pos[N][N];
//g表示原图，dis表示最短路径图
// path存最小环的各个点 ，pos表示节点i到节点j 的路径中经过的点
void get(int i,int j)
{
    if(!pos[i][j])return;//若pos为0，表示i与j之间没有中间节点，即i与j直接相连
    get(i,pos[i][j]);//i到中间节点
    path[cnt++]=pos[i][j];//将节点存入path中
    get(pos[i][j],j); //中间节点到j
    return;
}
int main()
{
    int n,m;
    cin>>n>>m;
    memset(g,0x3f,sizeof(g));//将g初始化 （最大值）
    for(int i=1;i<=n;++i)
      g[i][i]=0;  //i到i的距离为0
    for(int i=1;i<=m;++i)
    {
        int x,y,z;
        cin>>x>>y>>z;
        g[x][y]=min(g[x][y],z); //防止输入不同的距离
        g[y][x]=g[x][y];
    }
    memcpy(dis,g,sizeof(g));//将g复制给d
    int ans=inf;//表示环的总长度，初始为无穷大
    for(int k=1;k<=n;k++)
    {
        for(int i=1;i<k;i++)   //保证i和j都小于k，则可保证d[i][j]中i到j的路径中没有k
        for(int j=i+1;j<k;j++)
        {
            if((long long)dis[i][j]+g[j][k]+g[k][i]<ans)//long long 防止爆int
            {
                ans=dis[i][j]+g[j][k]+g[k][i];//更新环的长度
                cnt=0; //既然有比上一个环长度更小的环，则重新找这个环上的节点
                path[cnt++]=i; //存节点
                get(i,j);
                path[cnt++]=j;
                path[cnt++]=k;
            }
        }
        for(int i=1;i<=n;i++)
        for(int j=1;j<=n;j++)
        {
            if((long long)dis[i][k]+dis[k][j]<dis[i][j])//若以k为中间节点，i与j能相连
            {
                dis[i][j]=dis[i][k]+dis[k][j];//更新
                pos[i][j]=k; //k为i到j的路径的中间节点
            }
        }
    }
    if(ans==inf)//表示没有环
    printf("No solution.");
    else{
        for(int i=0;i<cnt;i++)
        printf("%d ",path[i]); //输出
    }

    return 0;
}
```

3 [1261]观光旅游

输入描述： 学校里面有N个景点。两个景点之间可能直接有道路相连，用Dist[I，J]表示它的长度；否则它们之间没有直接的道路相连。这里所说的道路是没有规定方向的，也就是说，如果从I到J有直接的道路，那么从J到I也有，并且长度与之相等。学校规定：每个游客的旅游线路只能是一个回路（好霸道的规定）。也就是说，游客可以任取一个景点出发，依次经过若干个景点，最终回到起点。一天，Xiaomengxian决定到湖南师大附中旅游。由于他实在已经很累了，于是他决定尽量少走一些路。于是他想请你——一个优秀的程序员——帮他求出最优的路线。怎么样，不是很难吧？（摘自《郁闷的出纳员》）

输入中有多组数据。请用SeekEof判断是否到达文件结束。 对于每组数据： 第一行有两个正整数N，M，分别表示学校的景点个数和有多少对景点之间直接有边相连。（N< =100,M< =10000） 以下M行，每行三个正整数，分别表示一条道路的两端的编号，以及这条道路的长度。

输出描述： 对于每组数据，输出一行： 如果该回路存在，则输出一个正整数，表示该回路的总长度；否则输出“No solution.”（不要输出引号）

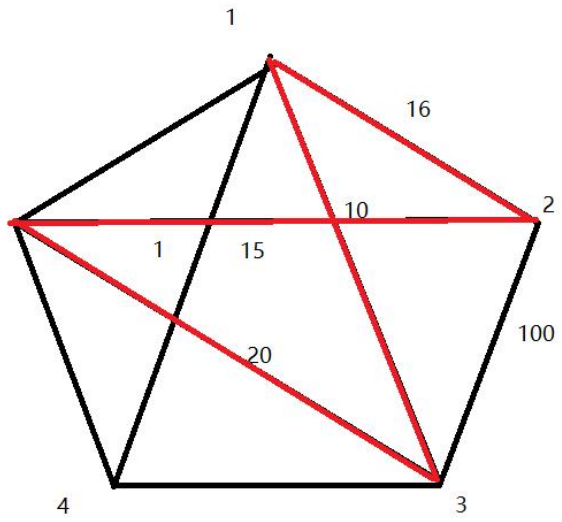
3 [1261]观光旅游

样例输入：

```
5 7
1 4 1
1 3 300
3 1 10
1 2 16
2 3 100
2 5 15
5 3 20
4 3
1 2 10
1 3 20
1 4 30
```

样例输出：

```
61
No solution.
```



1 3 5 2

10+20+15+16 = 61

[1261] 观光旅游

分析：

根据题目了解，题目要求寻找最小环。

因为题目数据N(<=100)较小，可以用邻接矩阵建立无向图，然后用Floyd算法走一遍图，找到起点和终点相同的最小路，即最小环，用ans记录走一遍后的最小环总长度。

如果ans没变化，说明不存在环，输出“No solution.”，反之，输出ans（最小环的总长度）

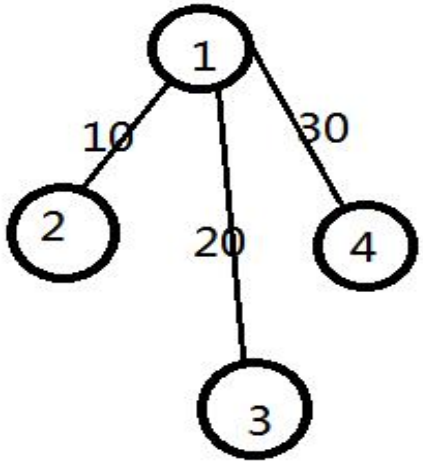
[1261] 观光旅游

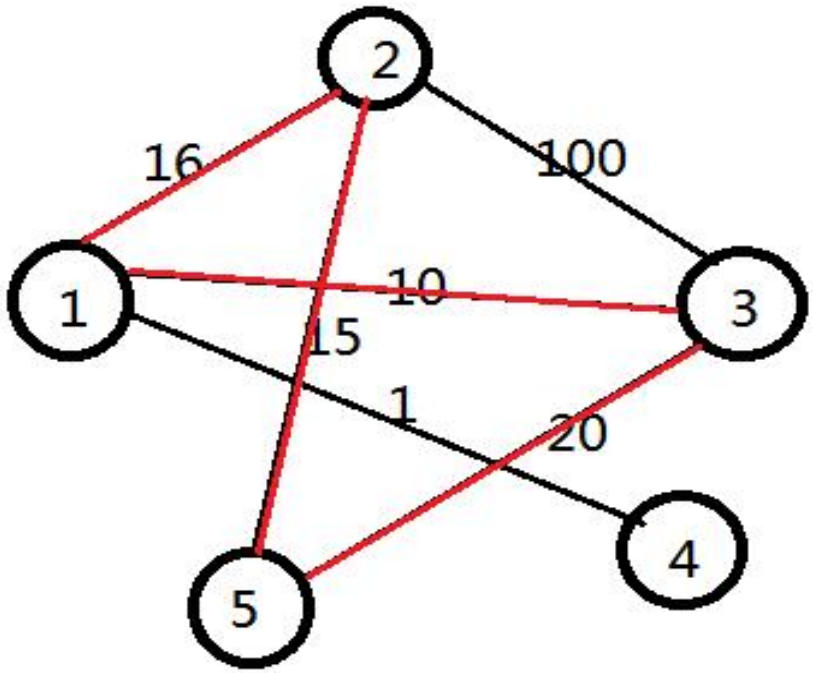
输入样例:

```
5 7
1 4 1
1 3 300
3 1 10
1 2 16
2 3 100
2 5 15
5 3 20
4 3
1 2 10
1 3 20
1 4 30
```

输出样例:

```
61
No solution.
```

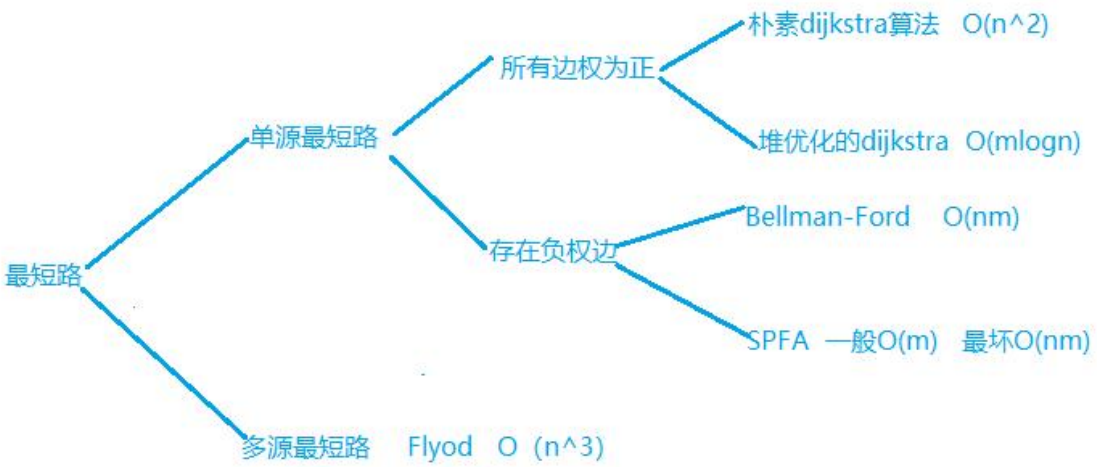




[1261] 观光旅游

```
#include <bits/stdc++.h>
using namespace std;
const int M=1005;
const int inf=1000005;
int n,m,x,y,d;
int dis[M][M],map[M][M],ans;
int main()
{
    while(scanf("%d %d",&n,&m)!=EOF)
    {
        int k;
        for(int i=1;i<=n;i++)
        for(int j=1;j<=n;j++)
            map[i][j]=inf;//每组数据初始值为无穷大
        for(int i=1;i<=m;i++)
        {
            scanf("%d %d %d",&x,&y,&d);
            map[x][y]=d;
            map[y][x]=d;
        }//建立邻接矩阵
```

最短路总结



https://blog.csdn.net/qq_44622401/article/details/104466457?depth_1-utm_source=distribute.pc_relevant.none-task&utm_source=distribute.pc_relevant.none-task

lyod

首先说一下多源最短路把。Floyd算是这几个算法里面最简单无脑的方法了，就是三重循环，暴力遍历所有的点。它的实现思想：从任意节点i到任意节点j的最短路径不外乎2种可能，1是直接从i到j，2是从i经过若干个节点k到j。所以，算法假设Dis(i,j)为节点u到节点v的最短路径的距离，对于每一个节点k，算法检查Dis(i,k) + Dis(k,j) < Dis(i,j)是否成立，如果成立，证明从i到k再到j的路径比i直接到j的路径短，便设置Dis(i,j) = Dis(i,k) + Dis(k,j)，这样一来，当遍历完所有节点k，Dis(i,j)中记录的便是i到j的最短路径的距离。

下面上这个算法实现的一个基本模板，具体代码要根据具体问题进行更改

```
for(int k=1;k<=n;k++){//代表中转顶点从1到n
        for(int i=1;i<=n;i++){
            for(int j=1;j<=n;j++){
                if(dis[i][j]>dis[i][k]+dis[k][j]){
                    dis[i][j]=dis[i][k]+dis[k][j];

```

Dijkstra

这个算法应该是大部分人刚接触最短路时最先学到的算法。Dijkstra其实是基于贪心的思想，采用了广度优先搜索的策略，以起始点为中心向外层层扩展，直到扩展到终点为止。。

原理：
设G=(V,E)是一个带权有向图，把图中顶点集合V分为两组，第一组为已求出最短路径的顶点集合（用S表示，初始时S中只有一个源点，以后每求得一条最短路径，就将加入到集合S中，直到全部顶点都加入到S中，算法就结束了），
第二组为其余未确定最短路径的顶点集合（用U表示），按最短路径的的递增次序依次把第二组中的顶点加入S中。在加入的过程中，总保持从源点v到S中各个顶点的最短路径长度不大于从源点v到U中任何路径的长度。
此外，每个顶点对应一个距离，S中的顶点的距离就是从v到此顶点的最短路径长度，U中的顶点的距离，是从v到此顶点只包括S中的顶点为中间顶点的当前路径的最短长度。

```cpp
#include<bits/stdc++.h>
using namespace std;
const int N=510;
int dist[N],g[N][N],n,m;
bool st[N];
int dijkstra()
{
    memset(dist,0x3f,sizeof dist);
    dist[1]=0;
    for(int i=0;i<n-1;i++)
    {
        int t=-1;
        for(int j=1;j<=n;j++)
            if(!st[j]&&(t==-1||dist[j]<dist[t]))
                t=j;
        st[t]=true;
        for(int j=1;j<=n;j++)
            dist[j]=min(dist[j],dist[t]+g[t][j]);
    }
    if(dist[n]==0x3f3f3f3f) return -1;
    return dist[n];
}
int main()
{
    memset(g,0x3f,sizeof g);
    cin>>n>>m;
    while(m--)
    {
        int a,b,c;
        cin>>a>>b>>c;
        g[a][b]=min(g[a][b],c);
    }
    cout<<dijkstra();
}
```

优化版的Dijkstra

优化原理：
优化Dijkstra有两个版本，一个是用优先队列优化，一个是用堆来进行优化。
首先我们要分析哪里需要优化，为什么这样优化？
先回答第一个问题：从上面Dijkstra的原理和代码我们可以看出来。我们每次在找最近的点的时候（就是代码中每次给t赋值的这个过程）上面这个过程的代码是O（n^2）的，那么我们可以考虑用一种存储形式，让他每次直接返回一个最小的，就达成了优化。
然后第二个问题：小根堆和优先队列（设置出队顺序）正好符合这个要求。

```cpp
#include<bits/stdc++.h>
using namespace std;
typedef pair<int,int>PII;
const int N=1e5+10;
int n,m,h[N],w[N],e[N],ne[N],idx,dist[N];
bool st[N];
void add(int a,int b,int c)
{
    e[idx]=b,w[idx]=c,ne[idx]=h[a],h[a]=idx++;
}
int dijkstra()
{
    memset(dist,0x3f,sizeof dist);
    dist[1]=0;
    priority_queue<PII,vector<PII>,greater<PII>>heap;
    heap.push({0,1});
    while(heap.size())
    {
        auto t=heap.top();
        heap.pop();
        int ver=t.second,distance=t.first;
        if(st[ver]) continue;
        st[ver]=true;
        for(int i=h[ver];i!=-1;i=ne[i])
        {
            int j=e[i];
            if(dist[j]>distance+w[i])
            {
                dist[j]=distance+w[i];
                heap.push({dist[j],j});
            }
        }
    }
    if(dist[n]==0x3f3f3f)   return -1;
    return dist[n];
}
```

```cpp
int main()
{
    cin>>n>>m;
    memset(h,-1,sizeof h);
    while(m--)
    {
        int a,b,c;
        cin>>a>>b>>c;
        add(a,b,c);
    }
    cout<<dijkstra()<<endl;
}
```

Bellman-Ford

这个算法应该算是除了Flyod之外最简单且最容易理解的算法了。
Bellman-Ford算法的优点是可以发现负圈，缺点是时间复杂度比Dijkstra算法高。

算法流程：
（1）初始化：将除起点s外所有顶点的距离数组置无穷大 d[v] = INF, d[s] = 0
（2）迭代：遍历图中的每条边，对边的两个顶点分别进行一次松弛操作，直到没有点能被再次松弛
（3）判断负圈：如果迭代超过V-1次，则存在负圈

根据上面这个流程可以知道，我们只需要把输入数据按边存储，然后依次遍历完就ok了，算是一个比较傻的算法，这里就不多做解释了。
然后这个算法因为是按边进行处理的，所以有一个优势，就是当题目限制了最短路能有几条边的时候，只有这个算法可以使用。下面我们上一道例题 题目代码来源 AcWing 853. 有边数限制的最短路

SPFA算法就是用优先队列优化后的Bellman—Ford算法，所以原理我们就不在赘述了。
这个算法算是使用最为普遍的，很多问题不管是正权图还是带负权图，都可以用SPFA算法。
优化原理：
Bellman—Ford算法的时间复杂度比较高，原因在于Bellman—Ford算法要递推n次，每次递推，扫描所有的边，在递推n次的过程中很多判断是多余的，SPFA算法是Bellman—Ford算法的一种队列实现，减少了不必要的冗余判断。
大致流程：
用一个队列来进行维护。初始时将源点加入队列。每次从队列中取出一个顶点，并与所有与它相邻的顶点进行松弛，若某个相邻的顶点松弛成功，则将其入队。重复这样的过程直到队列为空时算法结束。