第七章 图

我们用树可以表示一些层级结构的数据，那么我们如何表示一个象一个地区的“交通图”这样的网状结构呢？

graph [ɡræf] n.图；图表；曲线图 v.把…绘入图表；用胶版印刷; 网络.图形；图像；图解

-graph 做词是“写”的意思。

vertex [ˈvɜrˌteks] n.顶点；（三角形或锥形的）顶点；角顶；至高点

--vert，ver,vers 旋转的意思

edge [edʒ]n.边缘；边沿；刀刃；利刃

matrix [ˈmeɪtrɪks] 矩阵；

adjacent [əˈdʒeɪs(ə)nt] adj.与…毗连的；邻近的 网络:接近的；相邻；紧接着的

--jac “扔”的意思

directed [də'rektɪd]adj. 有指导的；定向的；【数】标出(数、角、线段的)正负的。网络：导演；有向；导向

undirected [ˌʌndə'rektɪd]adj.未受指导的；不受指引的；(信等)未写姓名住址的。网络：无向；无主导；无指导的

--rect “直”的意思

--di “二”的意思

orthogonal [ɔr'θɒgənəl] adj.直角的；正交的；直交的；(统计数字)互不相关的。网络：垂直的；正交性

# 7.4 图的应用

## 7.4.1图的连通分量

### 1.问题

#### 1)问题：

如右图，如何求出图的连通分量（有几个连通子图）

连通图：无论是广度优先搜索还是深度优先搜索，仅需要调用一次搜索过程，即从任一个顶点出发，便可以遍历图中的各个顶点。

非连通图：则需要多次调用搜索过程，而每次调用得到的顶点访问序列恰为各连通分量中的顶点集。调用搜索过程的次数就是该图连通分量的个数。

|  |
| --- |
| bool visited[MAX\_VERTEX\_NUMBER] = {false}  void TraverseGraph(Graph g){  for(int vi = 0; vi<g.vexNum; vi++){  if(!visited[vi]){  Search(g,vi);//DFS,BFS  }  }  } |

**在这个算法里面：Search()调用几次，就有几个连通子图。**

A

D

B

C

I

F

E

G

A

D

B

C

I

J

H

F

E

G

J

H

## 7.4.2 最小生成树

### 1.生成树

**连通图**G的一个子图如果是一棵包含G的所有顶点的树，则该子图称为G的生成树（Spanning Tree）。在生成树中添加任意一条属于原图中的边必定会产生回路,n个顶点的生成树具有n-1条边。

用广搜和深搜都可以生成树。

### 2.最小生成树

在一个连通网（有边权的连通图）的所有生成树中，各边的代价之和最小的那棵生成树称为该连通网的最小代价生成树（Minimum Cost Spanning Tree），简称为最小生成树。

理解：1.最小生成树是对连通网来说的。

**最小生成树的重要性质如下：**

设N=(V,{E}) 是一连通网，U 是顶点集V的一个非空子集。若（u , v）是一条具有最小权值的边，其中u∈U，v∈V-U，则存在一棵包含边（u , v）的最小生成树。（看不明白）

N表示连通网

V表示顶点集合

E是边的集合

一个点到到另一个点的最小距离，那么这个边必须在这个树中。

用反证法证

假设不存在这样一棵包含边（u , v）的最小生成树。任取一棵最小生成树T，将（u , v）加入T中。根据树的性质，此时T中必形成一个包含（u , v）的回路，且回路中必有一条边（u’ , v’）的权值大于或等于（u , v）的权值。删除（u , v），则得到一棵代价小于等于T的生成树T’，且T’为一棵包含边（u , v）的最小生成树。这与假设矛盾。

### 3.最小生成树的有什么用处

前题是对边通网基础上来说的，把所有点都连起来，边权之和最小。如6个点只能选择5条边。

引例：灌溉

**问题描述：**

到了旱季农业生产的灌溉就成了一个大问题。为了保证灌溉的顺利，某县政府决定投资为各个村之间建立灌溉管道。

**输入：**

输入第1行包括一个整数N，表示某县的村庄的数量。（3≤N≤100），

第2行到N+1行为一个N×N的矩阵,表示每个村庄之间的距离。

虽然在理论上，他们是N行，每行由N个用空格分隔的数组成，实际上，他们限制在80个字符，因此，某些行会紧接着另一些行。当然，对角线将会是0，因为不会有线路从第i个村到它本身（任何两个村之间的距离都不大于100000）。

**输出：**

输出只有一个，为修建灌溉管道将所有村庄相连在一个灌溉系统里所需的最小管道长度。

**样例输入：**

4

0 4 9 21

4 0 8 17

9 8 0 16

21 17 16 0

**样例输出：**

28

### 4.最小生成树算法

#### 1）普里姆算法(prim)---加点且不构成回路。

##### (1)基本思路

此算法可以称为“加点法”，每次迭代选择代价最小的边对应的点，加入到最小生成树中。算法从某一个顶点s开始，逐渐长大覆盖整个连通网的所有顶点。

①先选择一个顶点u，把u加到U集合中

②再找出顶点u发出的最小的边

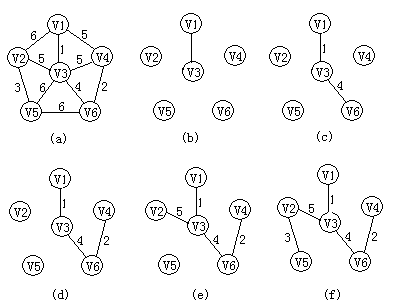
③找这到条边的另一个顶点v

④把v加到U中

⑤再从集合U中的每一个点发出的其它边中找出最小的。

重复过过程直到U=V

理解：1.是要准备一个数组,来记录顶点是否已经做为起始点用过2.关于是边集的定义，边集元素的下标是这个边的终点，当然也要记录下起点，要用到结构体。第几边，也就是终点为几号顶点的边，当然还要看一下，这个顶点是否已经加入到U集合中了。



##### (2)算法描述

|  |
| --- |
| /\*从顶点u出发，按普里姆算法构造连通网gn 的最小生成树，并输出生成树的每条边\*/  MiniSpanTree\_Prim(AdjMatrix gn, VertexData u){  k=LocateVertex(gn, u);  closedge[k].lowcost=0; /\*初始化，U={u} \*/  for (i=0;i<gn.vexnum;i++) {  if ( i!=k){/\*对V-U中的顶点i，初始化closedge[i]\*/  closedge[i].adjvex=u;  closedge[i].lowcost=gn.arcs[k][i].adj;  }  }  for (e=1;e<=gn.vexnum-1;e++){/\*找n-1条边(n= gn.vexnum) \*/  k0=Minium(closedge); /\* closedge[k0]中存有当前最小边（u0,v0）的信息\*/  u0= closedge[k0].adjvex; /\* u0∈U\*/  v0= gn.vexs[k0] /\* v0∈V-U\*/  printf(u0, v0); /\*输出生成树的当前最小边（u0,v0）\*/  closedge[k0].lowcost=0; /\*将顶点v0纳入U集合\*/  for ( i=0 ;i<vexnum;i++){ /\*在顶点v0并入U之后，更新closedge[i]\*/  if ( gn.arcs[k0][i].adj <closedge[i].lowcost){  closedge[i].lowcost= gn.arcs[k0][i].adj;  closedge[i].adjvex=v0;  }  }  }  } |

##### (3)算法实现—用邻接矩阵存储的无向网

|  |
| --- |
| /\*  Name: 7-4-1-MatrixMiniSpanTree\_Prim.cpp  Date:  Description:  从顶点u出发，按普里姆算法构造连通网gn 的最小生成树，并输出生成树的每条边  void MiniSpanTree\_Prim(AdjMatrix gn, VertexData u){  k=LocateVertex(gn, u);  closedge[k].lowcost=0; //初始化，U={u}  for (i=0;i<gn.vexnum;i++) {  if ( i!=k){//对V-U中的顶点i，初始化closedge[i]  closedge[i].adjvex=u;  closedge[i].lowcost=gn.arcs[k][i].adj;  }  }  for (e=1;e<=gn.vexnum-1;e++){//找n-1条边(n= gn.vexnum)  k0=Minium(closedge); //closedge[k0]中存有当前最小边（u0,v0）的信息  u0= closedge[k0].adjvex; // u0∈U  v0= gn.vexs[k0] // v0∈V-U  printf(u0, v0); //输出生成树的当前最小边（u0,v0）  closedge[k0].lowcost=0; //将顶点v0纳入U集合  for ( i=0 ;i<vexnum;i++){ //在顶点v0并入U之后，更新closedge[i]  if ( gn.arcs[k0][i].adj <closedge[i].lowcost){  closedge[i].lowcost= gn.arcs[k0][i].adj;  closedge[i].adjvex=v0;  }  }  }  }  6  1 2 3 4 5 6  0 6 1 5 32767 32767  6 0 5 32767 3 32767  1 5 0 5 6 4  5 32767 5 0 32767 2  32767 3 6 32767 0 6  32767 32767 4 2 6 0  \*/  #include <iostream>  #include <cstdio>  #include <vector>  #include <cstring>  using namespace std;  struct MatrixGraph{  vector<int> vertexes;  vector<vector<int> > edges;  int vexNum = 0;  int edgeNum = 0;  };  struct CloseEdge{  int vex = -1;  int lowCost = 0;  };  void CrtMatrixGraph(MatrixGraph &g){  cin>>g.vexNum;  g.vertexes.resize(g.vexNum);  for(int i = 0; i < g.vexNum; i++){  cin>>g.vertexes[i];  }    g.edges.resize(g.vexNum);  for(int i = 0; i < g.vexNum; i++){  g.edges[i].resize(g.vexNum);  }    for(int i = 0; i < g.vexNum; i++){  for(int j = 0; j < g.vexNum; j++){  cin>>g.edges[i][j];  if(g.edges[i][j]>0){  g.edgeNum++;  }  }  }  g.edgeNum = g.edgeNum/2;  }  void OutMatrixGraph(MatrixGraph &g){  cout<<g.vexNum<<"\t"<<g.edgeNum<<endl;  for(int i = 0; i < g.vexNum; i++){  for(int j = 0; j < g.vexNum; j++){  cout<<g.edges[i][j]<<"\t";  }  cout<<endl;  }  }  CloseEdge\* closeEdge = NULL;  int Minimum(int vexcnt){  int lc= 32767;  int k0 = 0;  for(int i = 0;i< vexcnt;i++){  if(closeEdge[i].lowCost>0 && closeEdge[i].lowCost<lc ){  lc=closeEdge[i].lowCost;  k0 = i;  }  }  return k0;  }  /\*从顶点u出发，按普里姆算法构造连通网gn 的最小生成树，并输出生成树的每条边\*/  void MiniSpanTree\_Prim(MatrixGraph &g, int vi){  closeEdge=(CloseEdge\*)malloc(sizeof(CloseEdge)\*g.vexNum);  int k=vi;//LocateVertex(gn, u);  closeEdge[k].lowCost =0; /\*初始化，U={u} \*/  for (int i=0;i<g.vexNum;i++) {  // if ( i!=k){/\*对V-U中的顶点i，初始化closedge[i]\*/  closeEdge[i].vex=k;  closeEdge[i].lowCost=g.edges[k][i];  // }  printf("%d%d:%d\t",g.vertexes[k],g.vertexes[i],closeEdge[i].lowCost);  }  cout<<endl;  for (int e=1;e<=g.vexNum-1;e++){/\*找n-1条边(n= gn.vexnum) \*/  int k0=Minimum(g.vexNum); /\*找出在closeEdge数组中存的权值最小的边，也就是其它点到起始点的最小距离的那个顶点\*/  int u0= closeEdge[k0].vex; /\* 那条边的下标，就是那个其它点，那边的存的顶点就是起始点\*/  printf("%d,%d\n",g.vertexes[u0],g.vertexes[k0]); /\*输出生成树的当前最小边（u0,v0）\*/  closeEdge[k0].lowCost=0; /\*将顶点v0纳入U集合，也就是k0这条边不能再用了\*/  for (int i=0 ;i<g.vexNum;i++){ /\*在顶点v0并入U之后，更新closedge[i]\*/  if ( g.edges[k0][i] < closeEdge[i].lowCost){  closeEdge[i].lowCost= g.edges[k0][i];  closeEdge[i].vex=k0;  }  }  }  }    int main(){  const char\* fileName = "7-4-1-MatrixMiniSpanTree\_Prim.in";  if(!freopen(fileName,"r",stdin)){  cerr<<"Open file error"<<endl;  return 0;  }  MatrixGraph mg;  CrtMatrixGraph(mg);  //OutMatrixGraph(mg);  MiniSpanTree\_Prim(mg,5);  return 0;  } |

分析：以v6，下标为5的点为例，分析算法

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 |
| 0 | 0 | 6 | 1 | 5 | 32767 | 32767 |
| 1 | 6 | 0 | 5 | 32767 | 3 | 32767 |
| 2 | 1 | 5 | 0 | 5 | 6 | 4 |
| 3 | 5 | 32767 | 5 | 0 | 32767 | 2 |
| 4 | 32767 | 3 | 6 | 32767 | 0 | 6 |
| 5 | 32767 | 32767 | 4 | 2 | 6 | 0 |

1.初始化状态

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 边 | 0 | 1 | 2 | 3 | 4 | 5 |
| 起始顶点 | 5 | 5 | 5 | 5 | 5 | 5 |
| 到达点 | 0 | 1 | 2 | 3 | 4 | 5 |
| 权值lowCost | 32767 | 32767 | 4 | 2 | 6 | 0 |

2.第一次执行Minimum()返回 边权最小值的下标 3，其实也就是从5到3的距离最小。

通过把下标为3的边的权值修改为0，表示这条边已经选中了，其实也可以改成如-1等其它值。总之和没用的边能区分开来就行。

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 边 | 0 | 1 | 2 | 3 | 4 | 5 |
| 起始顶点 | 5 | 5 | 5 | 5 | 5 | 5 |
| 到达点 | 0 | 1 | 2 | 3 | 4 | 5 |
| 权值lowCost | 32767 | 32767 | 4 | **0** | 6 | 0 |

3.同时也找到一个新点3，看一下3到各个点的边

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 边（到达的点） | 0 | 1 | 2 | 3 | 4 | 5 |
| 原起始顶点 | 5 | 5 | 5 | 5 | 5 | 5 |
| 现超始点 | 3 | 3 | 3 | **3** | 3 | 3 |
| 原权值 | 32767 | 32767 | 4 | **0** | 6 | 0 |
| 现权值 | 5 | 32767 | 5 | 0 | 32767 | 2 |
| 新权值 | 5 | 32767 | 4 | 0 | 6 | 0 |
| 新起始点 | 3 | 5 | 5 | 5 | 5 | 5 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 边 | 0 | 1 | 2 | 3 | 4 | 5 |
| 起始顶点 | 3 | 5 | 5 | 5 | 5 | 5 |
| 权值 | 5 | 32767 | 4 | 0 | 6 | 0 |

4．再次执行Minimum()返回边权最小值的下标 2，其实也就是从5到2的距离最小。通过把下标为2的边的权值修改为0，表示这条边已经选中了顶点2也选中了。

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 边（到达的点） | 0 | 1 | 2 | 3 | 4 | 5 |
| 起始顶点 | 3 | 5 | 5 | 5 | 5 | 5 |
| 原权值 | 5 | 32767 | 0 | 0 | 6 | 0 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 边（到达的点） | 0 | 1 | 2 | 3 | 4 | 5 |
| 起始顶点 | 3 | 5 | 5 | 5 | 5 | 5 |
| 现超始点 | 2 | 2 | 2 | 2 | 2 | 2 |
| 原权值 | 5 | 32767 | 0 | 0 | 6 | 0 |
| 现权值 | 1 | 5 | 0 | 5 | 6 | 4 |
| 新权值 | 1 | 5 | 0 | 0 | 6 | 0 |
| 新起始点 | 2 | 2 | 5 | 5 | 5 | 5 |

理解：其实上面表格，对理解程序帮助不大，思考起来也很复杂，辛苦一顿，先留着吧。关键是看算法思想后面的理解。

#### 2)克鲁斯卡尔算法(Kruskal)

##### (1)基本思想

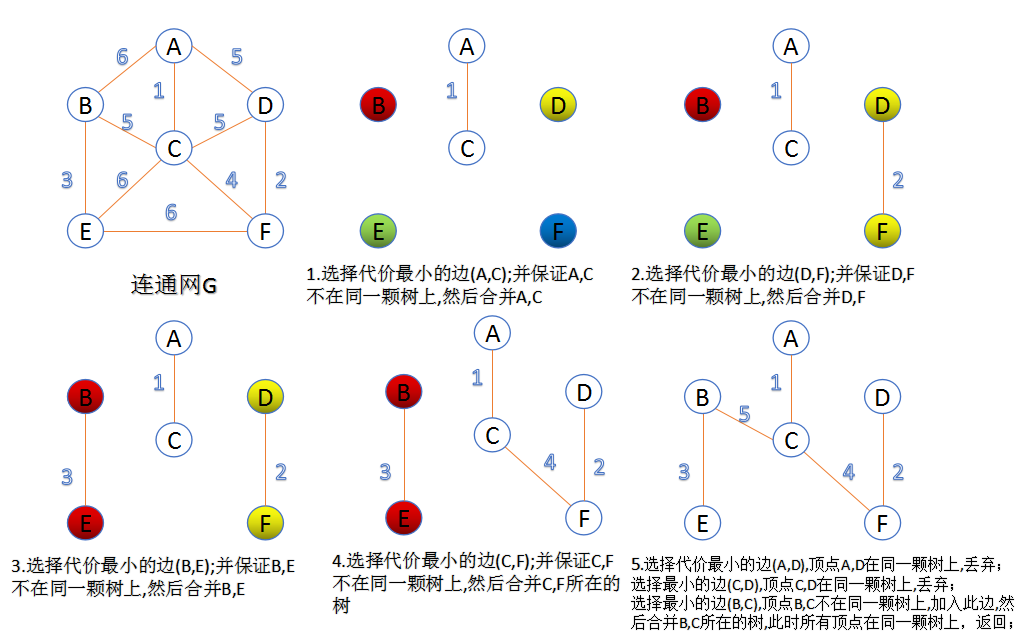
此算法可以称为“加边法”，初始最小生成树边数为0，每迭代一次就选择一条满足条件的最小代价边，加入到最小生成树的边集合里。

①把图中的所有边按代价从小到大排序；

②把图中的n个顶点看成独立的n棵树组成的森林；

③按权值从小到大选择边，所选的边连接的两个顶点ui,vi,应属于两颗不同的树，则成为最小生成树的一条边，并将这两颗树合并作为一颗树。

④重复(3),直到所有顶点都在一颗树内或者有n-1条边为止。



此算法结合并查集，很好理解。

### 5．最小生成树应用

#### 练习1 寻宝

**题目描述**

亮亮解出了卷轴隐藏的秘密，来到了一片沼泽地。这里有很多空地，而面试直通卡可能埋在任意一块空地中，好在亮亮发现了一堆木材，他可以将木材铺在两个空地之间的沼泽地上。因为亮亮不知道面试直通卡具体在哪一块空地中，所以必须要保证任意一块空地对于亮亮来说是可以抵达的。 “怎么还有鳄鱼！没办法，看来有些空地不能直接到达了。” 亮亮虽然没有洁癖，但是沼泽地实在太臭了，所以亮亮不会循环利用木材。而且木材不能拼接在一起使用，所以亮亮必须要知道在耗费木材最少的情况下，最长的那根木材至少需要多长。

**输入描述:**

第一行包含两个整数N(1≤N≤10000),M(1≤M≤1000000)。N表示公有N块空地。

接下来M行，每行包含三个整数P(1≤P≤N),Q(1≤Q≤N),K代表P,Q两个间没有鳄鱼，需要耗费K的木材。

**输出描述:**

一个整数，即耗费木材最少的情况下，最长的那根木材长度。

|  |  |
| --- | --- |
| **输入样例** | **输出样例** |
| 4 3  1 2 1  2 3 1  3 4 2 | 2 |

#### 练习2 畅通工程

**Problem Description**

省政府“畅通工程”的目标是使全省任何两个村庄间都可以实现公路交通（但不一定有直接的公路相连，只要能间接通过公路可达即可）。经过调查评估，得到的统计表中列出了有可能建设公路的若干条道路的成本。现请你编写程序，计算出全省畅通需要的最低成本。

**Input**

测试输入包含若干测试用例。每个测试用例的第1行给出评估的道路条数 N、村庄数目M ( < 100 )；随后的 N 行对应村庄间道路的成本，每行给出一对正整数，分别是两个村庄的编号，以及此两村庄间道路的成本（也是正整数）。为简单起见，村庄从1到M编号。当N为0时，全部输入结束，相应的结果不要输出。

**Output**

对每个测试用例，在1行里输出全省畅通需要的最低成本。若统计数据不足以保证畅通，则输出“?”。

**Sample Input**

3 3

1 2 1

1 3 2

2 3 4

1 3

2 3 2

0 100

**Sample Output**

3

?

#### 练习3 继续畅通工程

**Problem Description**

省政府“畅通工程”的目标是使全省任何两个村庄间都可以实现公路交通（但不一定有直接的公路相连，只要能间接通过公路可达即可）。现得到城镇道路统计表，表中列出了任意两城镇间修建道路的费用，以及该道路是否已经修通的状态。现请你编写程序，计算出全省畅通需要的最低成本。

**Input**

测试输入包含若干测试用例。每个测试用例的第1行给出村庄数目N ( 1< N < 100 )；随后的 N(N-1)/2 行对应村庄间道路的成本及修建状态，每行给4个正整数，分别是两个村庄的编号（从1编号到N），此两村庄间道路的成本，以及修建状态：1表示已建，0表示未建。

当N为0时输入结束。

**Output**

每个测试用例的输出占一行，输出全省畅通需要的最低成本。

**Sample Input**

3

1 2 1 0

1 3 2 0

2 3 4 0

3

1 2 1 0

1 3 2 0

2 3 4 1

3

1 2 1 0

1 3 2 1

2 3 4 1

0

**Sample Output**

3

1

0

#### 例4 畅通工程再续

**Problem Description**

相信大家都听说一个“百岛湖”的地方吧，百岛湖的居民生活在不同的小岛中，当他们想去其他的小岛时都要通过划小船来实现。现在政府决定大力发展百岛湖，发展首先要解决的问题当然是交通问题，政府决定实现百岛湖的全畅通！经过考察小组RPRush对百岛湖的情况充分了解后，决定在符合条件的小岛间建上桥，所谓符合条件，就是2个小岛之间的距离不能小于10米，也不能大于1000米。当然，为了节省资金，只要求实现任意2个小岛之间有路通即可。其中桥的价格为 100元/米。

**Input**

输入包括多组数据。输入首先包括一个整数T(T <= 200)，代表有T组数据。

每组数据首先是一个整数C(C <= 100),代表小岛的个数，接下来是C组坐标，代表每个小岛的坐标，这些坐标都是 0 <= x, y <= 1000的整数。

**Output**

每组输入数据输出一行，代表建桥的最小花费，结果保留一位小数。如果无法实现工程以达到全部畅通，输出”oh!”.

Sample Input

**2**

2

10 10

20 20

3

1 1

2 2

1000 1000

Sample Output

1414.2

oh!

#### 例5 公路

**题目描述**

Flatopia岛国完全平坦。不幸的是，Flatopia没有公共高速公路。因此Flatopia的交通很困难。弗拉托利亚政府意识到了这个问题。他们计划建造一些高速公路，这样就可以在不离开高速公路系统的情况下在任何一对城镇之间行驶。

Flatopian城镇的编号从1到N.每条高速公路恰好连接两个城镇。所有高速公路都遵循直线。所有高速公路都可以在两个方向上使用。高速公路可以自由地相互交叉，但司机只能在位于两条高速公路尽头的小镇的高速公路之间切换。

Flatopian政府希望尽量减少最长的高速公路的建设时间。但是，他们希望保证每个城镇都可以从其他城镇到达公路。

**输入**

第一行输入是一个整数T，它表示跟随的测试用例数。

每种情况的第一行是整数N（3 <= N <= 500），这是村庄的数量。然后是N行，其中第i个包含N个整数，这些N个整数中的第j个是村庄i和村庄j之间的距离（距离应该是[1,65536]内的整数）。每个测试用例后都有一个空行。

**输出**

对于每个测试用例，您应输出一个包含整数的行，该整数是要构建的最长道路的长度，以便连接所有村庄，并且此值最小。

**样例输入**

1

3

0 990 692

990 0 179

692 179 0

**样例输出**

692

英文版

**Description**

The island nation of Flatopia is perfectly flat. Unfortunately, Flatopia has no public highways. So the traffic is difficult in Flatopia. The Flatopian government is aware of this problem. They're planning to build some highways so that it will be possible to drive between any pair of towns without leaving the highway system.

Flatopian towns are numbered from 1 to N. Each highway connects exactly two towns. All highways follow straight lines. All highways can be used in both directions. Highways can freely cross each other, but a driver can only switch between highways at a town that is located at the end of both highways.

The Flatopian government wants to minimize the length of the longest highway to be built. However, they want to guarantee that every town is highway-reachable from every other town.

**Input**

The first line of input is an integer T, which tells how many test cases followed.

The first line of each case is an integer N (3 <= N <= 500), which is the number of villages. Then come N lines, the i-th of which contains N integers, and the j-th of these N integers is the distance (the distance should be an integer within [1, 65536]) between village i and village j. There is an empty line after each test case.

**Output**

For each test case, you should output a line contains an integer, which is the length of the longest road to be built such that all the villages are connected, and this value is minimum.

**Sample Input**

1

3

0 990 692

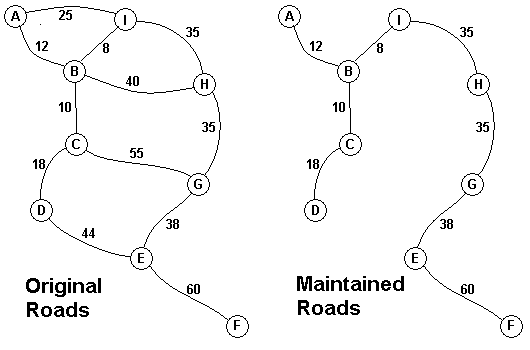
990 0 179

692 179 0

**Sample Output**

692

#### 例6 丛林之路



**Description**

The Head Elder of the tropical island of Lagrishan has a problem. A burst of foreign aid money was spent on extra roads between villages some years ago. But the jungle overtakes roads relentlessly, so the large road network is too expensive to maintain. The Council of Elders must choose to stop maintaining some roads. The map above on the left shows all the roads in use now and the cost in aacms per month to maintain them. Of course there needs to be some way to get between all the villages on maintained roads, even if the route is not as short as before. The Chief Elder would like to tell the Council of Elders what would be the smallest amount they could spend in aacms per month to maintain roads that would connect all the villages. The villages are labeled A through I in the maps above. The map on the right shows the roads that could be maintained most cheaply, for 216 aacms per month. Your task is to write a program that will solve such problems.

**Input**

The input consists of one to 100 data sets, followed by a final line containing only 0. Each data set starts with a line containing only a number n, which is the number of villages, 1 < n < 27, and the villages are labeled with the first n letters of the alphabet, capitalized. Each data set is completed with n-1 lines that start with village labels in alphabetical order. There is no line for the last village. Each line for a village starts with the village label followed by a number, k, of roads from this village to villages with labels later in the alphabet. If k is greater than 0, the line continues with data for each of the k roads. The data for each road is the village label for the other end of the road followed by the monthly maintenance cost in aacms for the road. Maintenance costs will be positive integers less than 100. All data fields in the row are separated by single blanks. The road network will always allow travel between all the villages. The network will never have more than 75 roads. No village will have more than 15 roads going to other villages (before or after in the alphabet). In the sample input below, the first data set goes with the map above.

**Output**

The output is one integer per line for each data set: the minimum cost in aacms per month to maintain a road system that connect all the villages. Caution: A brute force solution that examines every possible set of roads will not finish within the one minute time limit.

Sample Input

9

A 2 B 12 I 25

B 3 C 10 H 40 I 8

C 2 D 18 G 55

D 1 E 44

E 2 F 60 G 38

F 0

G 1 H 35

H 1 I 35

3

A 2 B 10 C 40

B 1 C 20

0

**Sample Output**

216

30

热带岛屿Lagrishan的长老们遇到了一个问题。几年前，在村庄之间的额外道路上花了一大笔外援资金。但丛林无情地超越了道路，因此大型公路网络的维护成本太高。长老理事会必须选择停止维持一些道路。左上方的地图显示了现在使用的所有道路以及维护它们的每月aacms的成本。当然，即使路线不像以前那么短，也需要在维护道路上的所有村庄之间进行某种方式。首席长老想告诉长老会，他们每月可以花费最少的金额来维持连接所有村庄的道路。在上面的地图中，村庄标有A到I. 右边的地图显示了最便宜的道路，每月216个aacms。你的任务是编写一个可以解决这些问题的程序。

**输入**

输入由1到100个数据集组成，后面跟着一个只包含0的最后一行。每个数据集都以一个只包含数字n的行开头，这个数字是村庄的数量，1 <n <27，村庄被标记使用字母表的前n个字母，大写。每个数据集都以n-1行完成，这些行以字母顺序从村庄标签开始。最后一个村庄没有线路。村庄的每条线路都以村庄标签开头，后面跟着一个从这个村庄到村庄的道路数k，后面有字母标签。如果k大于0，则该线继续每条k条道路的数据。每条道路的数据是道路另一端的村庄标签，后面是道路的aacms的每月维护费用。维护成本将是小于100的正整数。行中的所有数据字段由单个空格分隔。道路网络将始终允许所有村庄之间的旅行。该网络永远不会有超过75条道路。没有村庄将有超过15条道路通往其他村庄（在字母表之前或之后）。在下面的示例输入中，第一个数据集与上面的地图一起使用。

**输出**

每个数据集的输出为每行一个整数：维持连接所有村庄的道路系统的每月aacms的最低成本。注意：检查每一组可能的道路的强力解决方案将无法在一分钟的时间内完成。

样本输入

9

A 2 B 12 I 25

B 3 C 10 H 40 I 8

C 2 D 18 G 55

D 1 E 44

E 2 F 60 G 38

F 0

G 1 H 35

H 1 I 35

3

A 2 B 10 C 40

B 1 C 20

0

样本输出

216

30

#### 例7 品牌折扣店（outlets）

**Problem Description**

In China, foreign brand commodities are often much more expensive than abroad. The main reason is that we Chinese people tend to think foreign things are better and we are willing to pay much for them. The typical example is, on the United Airline flight, they give you Haagendazs ice cream for free, but in China, you will pay $10 to buy just a little cup.

So when we Chinese go abroad, one of our most favorite activities is shopping in outlets. Some people buy tens of famous brand shoes and bags one time. In Las Vegas, the existing outlets can't match the demand of Chinese. So they want to build a new outlets in the desert. The new outlets consists of many stores. All stores are connected by roads. They want to minimize the total road length. The owner of the outlets just hired a data mining expert, and the expert told him that Nike store and Apple store must be directly connected by a road. Now please help him figure out how to minimize the total road length under this condition. A store can be considered as a point and a road is a line segment connecting two stores.

**Input**

There are several test cases. For each test case: The first line is an integer N( 3 <= N <= 50) , meaning there are N stores in the outlets. These N stores are numbered from 1 to N. The second line contains two integers p and q, indicating that the No. p store is a Nike store and the No. q store is an Apple store. Then N lines follow. The i-th line describes the position of the i-th store. The store position is represented by two integers x,y( -100<= x,y <= 100) , meaning that the coordinate of the store is (x,y). These N stores are all located at different place. The input ends by N = 0.

**Output**

For each test case, print the minimum total road length. The result should be rounded to 2 digits after decimal point.

**Sample Input**

4

2 3

0 0

1 0

0 -1

1 -1

0

**Sample Output**

3.41

**问题描述**

在中国，外国品牌商品往往比国外贵得多。主要原因是我们中国人倾向于认为外国的东西更好，我们愿意为他们付出很多。典型的例子是，在美国联合航空公司的航班上，他们免费为您提供Haagendazs冰淇淋，但在中国，您只需花10美元购买一小杯。

因此，当我们中国人出国时，我们最喜欢的活动之一就是在商店购物。有些人一次买几十个名牌鞋包。在拉斯维加斯，现有的网点无法满足中国人的需求。所以他们想在沙漠中建立一个新的网点。新的商店包括许多商店。所有商店都通过道路连接。他们希望尽量减少道路总长度。这些网点的所有者刚刚聘请了一位数据挖掘专家，而专家告诉他，耐克商店和Apple商店必须通过一条道路直接连接。现在请帮助他弄清楚在这种情况下如何最小化道路总长度。商店可以被视为一个点，道路是连接两个商店的线段。

**输入**

有几个测试用例。对于每个测试用例：第一行是整数N（3 <= N <= 50），这意味着出口中有N个存储。这些N个商店的编号从1到N.第二行包含两个整数p和q，表示No.p商店是Nike商店，而q商店是Apple商店。然后是N行。第i行描述了第i个商店的位置。商店位置由两个整数x，y（-100 <= x，y <= 100）表示，这意味着商店的坐标是（x，y）。这N家商店都位于不同的地方。输入以N = 0结束。

**输出**

对于每个测试用例，打印最小总道路长度。结果应舍入为小数点后的2位数。

**样本输入**

4

2 3

0 0

1 0

0 -1

1 -1

0

**样本输出**

3.41

## 7.4.3有向无环图应用--拓扑排序（Topological Sort）

### 例1—排队问题（zcmu-2153）

**Problem Description**

马上要上体育课了，上体育课之前总归是要排个队的，ly作为班长，怎么排队的问题只能由她来解决，但是马上要上课了，ly又不清楚所有人的身高，她又不好意思问每个人的身高，因为这样会显的自己很不负责，于是她只能通过肉眼观察...那么问题来了，她只能观察出两个人A和B谁高谁矮，但是她没有办法排出一个序列。

ly都已经帮你出了两次主意赢过wjw，那么现在她需要你的帮助，你可以帮她吗？

(ly会告诉你A和B谁高，如果A比B高，会用A>B来表示)

**Input**

只有一组数据，每个比较结果占一行，读取到文件结束

**Output**

若输入数据无解，则输出"No Answer!",否则从高到低输出每个人的名字，中间没有分割符

若有多种情况，输出字典序最小的答案

**Sample Input**

E>A

A>S

S>Y

**Sample Output**

EASY

### 例2—逃生(HDU-4857)

**Problem Description**

糟糕的事情发生啦，现在大家都忙着逃命。但是逃命的通道很窄，大家只能排成一行。

现在有n个人，从1标号到n。同时有一些奇怪的约束条件，每个都形如：a必须在b之前。

同时，社会是不平等的，这些人有的穷有的富。1号最富，2号第二富，以此类推。有钱人就贿赂负责人，所以他们有一些好处。

负责人现在可以安排大家排队的顺序，由于收了好处，所以他要让1号尽量靠前，如果此时还有多种情况，就再让2号尽量靠前，如果还有多种情况，就让3号尽量靠前，以此类推。

那么你就要安排大家的顺序。我们保证一定有解。

**Input**

第一行一个整数T(1 <= T <= 5),表示测试数据的个数。

然后对于每个测试数据，第一行有两个整数n(1 <= n <= 30000)和m(1 <= m <= 100000)，分别表示人数和约束的个数。

然后m行，每行两个整数a和b，表示有一个约束a号必须在b号之前。a和b必然不同。

**Output**

对每个测试数据，输出一行排队的顺序，用空格隔开。

**Sample Input**

1

5 10

3 5

1 4

2 5

1 2

3 4

1 4

2 3

1 5

3 5

1 2

**Sample Output**

1 2 3 4 5

### 1．AOV-网（Activity On Vertex）：

用顶点表示活动，用弧表示活动间的优先关系的有向无环图，称为顶点表示活动的网（Activity On Vertex Network），简称为AOV-网。

例如，一个计算机专业的学生必须学习一系列基本课程，其中有些课程是基础课，如高等数学，它独立于其他课程，而另一些课程必须在学完作为它的基础的选修课才能开始。如通常应该在学完“程序设计基础”和“离散数学”之后才开始学习“数据结构”。这种课程间的优先关系如图6-17表示。用顶点表示课程，弧表示先决条件，若课程Ci是课程Cj的先决条件，则有弧< Ci,Cj>存在。

|  |  |  |
| --- | --- | --- |
| **课程编号** | **课程名称** | **先修课程** |
| C1 | 高等数学 | 无 |
| C2 | 程序设计基础 | 无 |
| C3 | 普通物理 | C1 |
| C4 | 离散数学 | C1,C2 |
| C5 | C语言程序设计 | C2 |
| C6 | 数据结构 | C2,C4,C5 |
| C7 | 编译原理 | C5,C6 |
| C8 | 计算机原理 | C3 |
| C9 | 操作系统 | C6, C8 |



表示课程之间优先关系的有向无环图

### 2.拓扑序列

在AOV网中一个有向无环图的拓扑序列是将图中的顶点排成一个线性序列，使得对于图中任意一对顶点u，v。若存在边<u,v>，则线性序列中u出现在v之前。

AOV-网的特性如下：

若vi为vj的先行活动，vj为vk的先行活动，则vi必为vk的先行活动，即先行关系具有可传递性。

不应该存在回路。

AOV-网的拓扑序列不是唯一的。

如前图的一个拓扑序列为：

c1 c2 c3 c4 c5 c6 c7 c8 c9

它的另二个拓扑序列为：

c1 c2 c3 c4 c5 c8 c6 c7 c9

c2 c5 c1 c4 c6 c7 c3 c8 c9

### 3.拓扑排序

求出拓扑序列的过程就是拓扑排序

#### 求拓扑排序的基本思想：

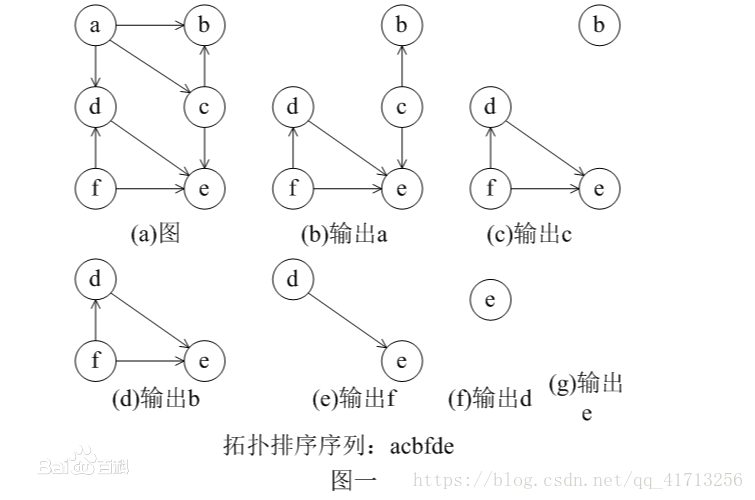
（1） 从有向图中选一个无前驱的顶点输出；

如果找不出这样的顶点，此题无解。

（2） 将此顶点和以它为起点的弧删除（把这个节点指向的节点的入度减一）；

（3） 重复（1）、（2），直到不存在无前驱的顶点；

（4） 若此时输出的顶点数小于有向图中的顶点数，则说明有向图中存在回路，不存在拓扑排序，也就是很多题目的无解的情况。否则输出的顶点的顺序即为一个拓扑序列。



#### 拓扑排序的算法实现：

|  |
| --- |
| /\*  Name:7-4-3-1-topologicalSort.cpp  Date: 07/06/17 08:14  Description:复习几个名词：  1.有向无环图dag: directed acyclic graph  2.邻接矩阵：adjacency matrix  3.邻接表：adjacency list  4.逆邻接表: inverse adjacency list  9  a b c d e f g h i  0 0 1 1 0 0 0 0 0  0 0 0 1 1 1 0 0 0  0 0 0 0 0 0 0 1 0  0 0 0 0 0 1 0 0 0  0 0 0 0 0 1 1 0 0  0 0 0 0 0 0 1 0 1  0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 1  0 0 0 0 0 0 0 0 0  \*/  #include <iostream>  #include <cstdio>  #include <vector>  #include <queue>  #include <stack>  using namespace std;  vector<vector<int> > gam;//gam:graph by adjacency matirx 利用邻接矩阵表示图  vector<vector<int> > gal;//graph by adjacency list 利用邻接表表示图,本质就是从这个点能到达那几个点，方便求出度 ，原理和树的数组表示法一样  vector<vector<int> > gial;// graph by inverse adjacency list 利用逆邻接表表示图，本质是记住这个点能从那几个点到达。求入度方便  vector<char> name;//用于存放节点的名字  vector<int> indegree;//用于存放顶点的入度  int n; //节点的个数  int temp;//临时变量 temporary  void CrtG(){  cin>>n;  gam.resize(n+1,vector<int>(n+1,0));  gal.resize(n+1);  gial.resize(n+1);  indegree.resize(n+1,0);  name.resize(n+1);  for(int i = 1; i <= n; i++){  cin>> name[i];  }  for(int i =1; i <= n; i++){  for(int j = 1; j <= n; j++){  cin>>temp;  if(temp > 0){  gam[i][j] = temp;//更新邻接矩阵，表示从i顶点可以到达j顶点  gal[i].push\_back(j);//邻接表表示法，j作为i的孩子存起来  gial[j].push\_back(i);//逆邻接表表示法，i作为j的“爹”存起来  }  }  }  }  void FindIndegreeByAM(){//通过邻接矩阵求入度 find indegree by adjacency matrix  bool dagFlag = false;  for(int c = 1; c <= n; c++){  temp = 0;  for(int r = 1; r <=n; r++){  if(gam[r][c] > 0){  temp++;  }  }  indegree[c] = temp;  if(temp == 0){  dagFlag = true;  }  }  if(!dagFlag){  cout<<"this is a direct cyclic graph"<<endl;  }  }  void FindIndegreeByAL(){//通过邻接表求入度 find indegree by adjacency list  bool dagFlag = false;  for(int k = 1; k <= n; k++){  temp = 0;  for(int i = 1; i <= n; i++){//遍历每个节点的孩子  for(int c = 0; c < gal[i].size(); c++){//children  if(gal[i][c] == k){  temp++;  }  }  }  indegree[k] = temp;  if(temp == 0){  dagFlag = true;  }  }  if(!dagFlag){  cout<<"this is a direct cyclic graph"<<endl;  }  }  void FindIndegreeByIAL(){//通过逆邻接表求入度 find indegree by inverse adjacency list  bool dagFlag = false;  for(int k = 1; k <= n; k++){  temp = 0;  temp = gial[k].size();  indegree[k] = temp;  if(temp == 0){  dagFlag = true;  }  }  if(!dagFlag){  cout<<"this is a direct cyclic graph"<<endl;  }  }  void OutG(){  for(int i =1; i <= n; i++){  for(int j = 1; j <= n; j++){  cout<<gam[i][j]<<" ";  }  cout<<endl;  }  }  void OutIndegree(){  for(int i = 1; i <= n; i++){  printf("%c:%d\n",name[i],indegree[i]);  }  }  void TopologicalSort1(){  int ans,cnt;//answer  vector<int> vis(n+1,0);  cnt = 0;  for(int i = 1; i <= n; i++){  for(int j = 1; j <= n; j++){  if(indegree[j] == 0){//找到一个入度为0的点  ans = j;  cnt++;  vis[cnt] = j;  indegree[j]--;//让他为负数，标记为不再使用  break;  }  }  for(int j = 1; j<= n; j++) {  if(gam[ans][j] > 0){  //gam[ans][j] = 0;//删除边，其实没什么用  indegree[j]--;//与相邻的边的入度减1  }  }  }  for(int i = 1; i <= cnt; i++){  printf("c%d\t",vis[i]);  }  }  void TopologicalSort2(){//利用队列技术  queue<int> q;//用与存放入度为0的节点序号  int vi;//vertex index  for(int i = 1; i <= n; i++){  if(indegree[i] == 0){  q.push(i);//将入度为零的节点压入队列  }  }  while(!q.empty()){  vi = q.front();  q.pop();  printf("c%d\t",vi);//输出入度为零的节点  for(int i = 0; i < gal[vi].size(); i++){  indegree[gal[vi][i]]--;//将vi的邻接点的入度减1  if(indegree[gal[vi][i]] == 0){  q.push(gal[vi][i]);//将入度为零的节点压入队列  }  }  }  }  void TopologicalSort3(){//利用栈技术  stack<int> s;//用与存放入度为0的节点序号  int vi;//vertex index  for(int i = 1; i <= n; i++){  if(indegree[i] == 0){  s.push(i);//将入度为零的节点压入队列  }  }  while(!s.empty()){  vi = s.top();  s.pop();  printf("c%d\t",vi);//输出入度为零的节点  for(int i = 0; i < gal[vi].size(); i++){  indegree[gal[vi][i]]--;//将vi的邻接点的入度减1  if(indegree[gal[vi][i]] == 0){  s.push(gal[vi][i]);//将入度为零的节点压入队列  }  }  }  }  int main(){  if(!freopen("7-4-3-1-topologicalSort.in","r",stdin)){  cerr<<"Open file error";  return 0;  }  CrtG();  //OutG();  //FindIndegreeByAM();  //FindIndegreeByAL();  FindIndegreeByIAL();  OutIndegree();  cout<<endl<<"============================"<<endl;  TopologicalSort1();  cout<<endl<<"============================"<<endl;  FindIndegreeByIAL();  cout<<endl<<"============================"<<endl;  OutIndegree();  cout<<endl<<"============================"<<endl;  TopologicalSort2();  cout<<endl<<"============================"<<endl;  FindIndegreeByIAL();  cout<<endl<<"============================"<<endl;  OutIndegree();  cout<<endl<<"============================"<<endl;  TopologicalSort3();  return 0;  } |

## 7.4.4最短路径

### 1．最短路径

在网中（边带权的图），求点A到点B的所有路径中，边的权值之和最短的那一条路径。这条路径就称为两点之间的最短路径，并称路径上的第一个顶点为源点（Source），最后一个顶点为终点（Destination）。

v0 v1 v2 v3 v4 v5

0 50 10 \* 45 \*

\* 0 15 \* 10 \*

20 \* 0 15 \* \*

\* 20 \* 0 35 \*

\* \* \* 30 0 \*

\* \* \* 3 \* 0

V1

50

20

10

10

30

V0

V4

V2

V3

V5

45

15

35

3

15

20

### 2．用迪杰斯特拉（Dijkstra）求最短路径算法--求一个顶点到其他各顶点的最短路径

对于图G=(V, E)，将图中的顶点分成两组：

第一组S：已求出的最短路径的终点集合（开始为{v0}）。

第二组V－S：尚未求出最短路径的顶点集合（开始为V－{v0}的全部结点）。

算法将按最短路径长度的递增顺序逐个将第二组的顶点加入到第一组中，直到所有顶点都被加入到第一组顶点集S为止。

[定理]：

下一条最短路径或者是弧（v0，vx），或者是中间经过S中的某些顶点，而后到达vx的路径。

[证明]：

可用反证法：假设下一条最短路径上有一个顶点vy不在S中，即此路径为（v0，…，vy，…， vx）。显然，（v0，…，vy）的长度小于（v0，…，vy，… ，vx）的长度，故下一条最短路径应为（v0，…，vy），这与假设的下一条最短路径（v0，…，vy，…， vx）相矛盾！因此，下一条最短路径上不可能有不在S中的顶点vy，即假设不成立。

[理解]最小距离替换法，假如我要求出v0到v1的最短距离

V1

50

20

10

10

30

V0

V4

V2

V3

V5

45

15

35

3

15

20

第一步，先找出从v0到各个点的距离的最小值。v0,v2 = 10;S = {v0}

第二步，再找出从v2到各个点的距离的最小值。v2,v3 = 15；v2,v0就不能再找了。S = {v0,v2}

也就是说：从v0出发，通过v2我可以到达v3。原来v0到v3是没有路径的，这样就相当找到了一条路径v0,v3 = v0,v2+v2+v3 = 25;

重复第一步：v0到各个点的距离的最小值v0,v3 = 25;

重复第二步：v3么保个点的距离的最小值v3,v1 = 20; S = {v0,v2,v3}

也就是说：从v0出发，通过v3我可以到达v1。这样就相当找到了一条路径v0,v3,v1 = v0,v3+v3,v1 = 45;和原来的50比较，新路径更近。将v0,v1更新为45;查找结束。

程序实现

|  |
| --- |
| /\*  Name:7-4-4-1-ShortestPathByDijkstraWithAdjacencyMatrix.cpp  Date:  Description:  \*/  #include <iostream>  #include <cstdio>  #include <vector>  #define INFINITY 32767  using namespace std;  struct MatrixGraph{  vector<int> vertexes;  vector<vector<int> > arcs;  int arcNum = 0;  int vexNum = 0;  };  void CrtMatrixGraph(MatrixGraph &g){  cin>>g.vexNum;  g.vertexes.resize(g.vexNum);  g.arcs.resize(g.vexNum);  for(int i = 0; i < g.vexNum; i++){  cin>>g.vertexes[i];  g.arcs[i].resize(g.vexNum);  }    for(int i = 0; i < g.vexNum; i++){  for(int j = 0; j < g.vexNum; j++){  cin>>g.arcs[i][j];  if(g.arcs[i][j] < INFINITY){  g.arcNum++;  }  }  }  }  void OutMatrixGraph(MatrixGraph &g){  cout<<g.vexNum<<"\t"<<g.arcNum<<endl;  for(int i = 0; i < g.vexNum; i++){  for(int j = 0; j < g.vexNum; j++){  cout<<g.arcs[i][j]<<"\t";  if(g.arcs[i][j] < INFINITY){  g.arcNum++;  }  }  cout<<endl;  }  }  void ShortestPahtDijkstra(MatrixGraph &g,int v){  int dist[g.vexNum];//dist[i]中存放顶点i的当前最短路径长度  int path[g.vexNum];//path[i]中存放顶点i的当前最短路径  int s[g.vexNum];//s为已找到最短路径的终点集合  int mindis,i,j,u,pre;  for(i = 0; i < g.vexNum; i++){  dist[i]= g.arcs[v][i]; /\*初始化距离\*/  s[i]=0; /\*辅助数组s[]置初值为0\*/  if(g.arcs[v][i]<INFINITY){  path[i]=v;//初始化每一条弧的起始点  }else{  path[i]=-1;  }  }  s[v]=1;  path[v]=0;  for(i=0;i<g.vexNum;i++){  mindis=INFINITY;  u=-1;  for(j=0;j<g.vexNum;j++){  if(s[j]==0 && dist[j]<mindis){  u=j;  mindis=dist[j];  }  }    if(u!=-1){  s[u]=1;  for(j=0;j<g.vexNum;j++)  if(s[j]==0)  if(g.arcs[u][j]<INFINITY && dist[u]+g.arcs[u][j]<dist[j]){  dist[j]=dist[u]+g.arcs[u][j];  path[j]=u;  }  }  }    printf("\nDijkstra算法求解如下：\n");  for(i=0;i<g.vexNum;i++){  if(i!=v){  printf("\n%d->%d:",v,i);  if(s[i]==1){  printf("路径长度为%2d,",dist[i]);  pre=i;  printf("路径逆序为：");  while(pre!=v){  printf("%d,",pre);  pre=path[pre];  }  printf("%d\n",pre);  }else{  printf("不存在路径\n");  }  }  }  }  int main(){  const char \*fileName = "7-4-4-1-ShortestPathByDijkstraWithAdjacencyMatrix.in";  if(!freopen(fileName,"r",stdin)){  cout<<"Open file error!";  return 0 ;  }  MatrixGraph mg;  CrtMatrixGraph(mg);    //OutMatrixGraph(mg);  ShortestPahtDijkstra(mg,0);  return 0;  } |

### 3．佛罗依德算法：求任意一对顶点间的最短路径

1)算法描述

弗洛伊德（Floyed）算法仍是从图的带权邻接矩阵cost出发，其基本方法如下。

如果从vi到vj有边，则从vi到vj一条长度为cost[i][j]的路径。该路径不一定是最短路径，尚需进行n次试探。

(-1)将vi到vj 的最短的路径长度初始化为g.arcs[i][j]，然后进行如下n次比较和修正：

(0)首先考虑路径(vi ,v0,vj)是否存在。如果存在，则比较其路径长度。取长度较短者为从vi到vj的中间顶点的序号不大于0的最短路径。

(1)假如在路径上再增加一个顶点v1，即如果(vi ,… ,v1)和(v1,… ,vj)分别是当前找到的中间顶点的序号不大于0的最短路径，那么，(vi ,…,v1,,…vj)就有可能是从vi到vj中间顶点的序号不大于1的最短路径。将它和已经得到的从vi到vj中间顶点的序号不大于0的最短路径相比较，从中选出中间顶点的序号不大于1的最短路径之后，再增加一个顶点v2，继续进行试探，依此类推，…，直至经过n次比较，最后求得的必是vi到vj的最短路径。

按此方法，可以同时求得各对顶点之间的最短路径。

现定义一个n阶方阵序列，这个序列依次有如下状态：A-1 ，A0 ，…，Ak ，…，An-1，其中：

A-1[i][j]=cost[i][j] (0≤i≤n-1，0≤j≤n-1)

Ak+1[i][j]=min{ Ak+1[i][k+1]+ Ak[k+1][j]} (0≤k≤n-2)

从上述计算公式可见：Ak[i][j]是从vi到vj的中间顶点的序号不大于k的最短路径的长度；An-1[i][j]就是从vi到vj的最短路径。

算法分析

A[i][j]初始状态

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 |
| 0 | 0 | 50 | 10 |  | 45 |  |
| 1 |  | 0 | 15 |  | 10 |  |
| 2 | 20 |  | 0 | 15 |  |  |
| 3 |  | 20 |  | 0 | 35 |  |
| 4 |  |  |  | 30 | 0 |  |
| 5 |  |  |  | 3 |  | 0 |

第一次迭代 k = 0;

d[0][0] d[0][0] + d[0][0]

d[0][1] d[0][0] + d[0][1]

d[0][2] d[0][0] + d[0][2]

d[0][3] d[0][0] + d[0][3]

d[0][4] d[0][0] + d[0][4]

d[0][5] d[0][0] + d[0][5]

由于d[0][0]无边，A无变化

d[1][0] d[1][0] + d[0][0]

d[1][1] d[1][0] + d[0][1]

d[1][2] d[1][0] + d[0][2]

d[1][3] d[1][0] + d[0][3]

d[1][4] d[1][0] + d[0][4]

d[1][5] d[1][0] + d[0][5]

由于d[1][0]无边，A无变化

d[2][0] d[2][0] + d[0][0]

d[2][1] d[2][0] + d[0][1] = 20 + 50 = 70

d[2][2] d[2][0] + d[0][2]

d[2][3] d[2][0] + d[0][3]

d[2][4] d[2][0] + d[0][4] =20 + 45 = 65

d[2][5] d[2][0] + d[0][5]

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 |
| 0 | 0 | 50 | 10 |  | 45 |  |
| 1 |  | 0 | 15 |  | 10 |  |
| 2 | 20 | 70  （2，0，1） | 0 | 15 | 65  （2，0，4） |  |
| 3 |  | 20 |  | 0 | 35 |  |
| 4 |  |  |  | 30 | 0 |  |
| 5 |  |  |  | 3 |  | 0 |

d[3][0] d[3][0] + d[0][0]

d[3][1] d[3][0] + d[0][1]

d[3][2] d[3][0] + d[0][2]

d[3][3] d[3][0] + d[0][3]

d[3][4] d[3][0] + d[0][4]

d[3][5] d[3][0] + d[0][5]

d30无边 无变化

d[4][0] d[4][0] + d[0][0]

d[4][1] d[4][0] + d[0][1]

d[4][2] d[4][0] + d[0][2]

d[4][3] d[4][0] + d[0][3]

d[4][4] d[4][0] + d[0][4]

d[4][5] d[4][0] + d[0][5]

d40无边 无变化

d[5][0] d[5][0] + d[0][0]

d[5][1] d[5][0] + d[0][1]

d[5][2] d[5][0] + d[0][2]

d[5][3] d[5][0] + d[0][3]

d[5][4] d[5][0] + d[0][4]

d[5][5] d[5][0] + d[0][5]

d50无边 无变化

V1

50

20

10

10

30

V0

V4

V2

V3

V5

45

15

35

3

15

20

**第二次迭代 k = 1;**

d[0][0] d[0][1] + d[1][0]

d[0][1] d[0][1] + d[1][1]

d[0][2] d[0][1] + d[1][2]

d[0][3] d[0][1] + d[1][3]

d[0][4] d[0][1] + d[1][4]

d[0][5] d[0][1] + d[1][5]

由无变化

d[1][0] d[1][1] + d[1][0]

d[1][1] d[1][1] + d[1][1]

d[1][2] d[1][1] + d[1][2]

d[1][3] d[1][1] + d[1][3]

d[1][4] d[1][1] + d[1][4]

d[1][5] d[1][1] + d[1][5]

由于d[1][1]为0，A无变化

d[2][0] d[2][1] + d[1][0]

d[2][1] d[2][1] + d[1][1]

d[2][2] d[2][1] + d[1][2]

d[2][3] d[2][1] + d[1][3]

d[2][4] d[2][1] + d[1][4]

d[2][5] d[2][1] + d[1][5]

d[3][0] d[3][1] + d[1][0]

V1

50

20

10

10

30

V0

V4

V2

V3

V5

45

15

30

3

15

20

35

d[3][1] d[3][1] + d[1][1]

d[3][2] d[3][1] + d[1][2] 20 +15 = 35

d[3][3] d[3][1] + d[1][3]

d[3][4] d[3][1] + d[1][4] 20 + 10 = 30

d[3][5] d[3][1] + d[1][5]

V1

50

20

10

10

30

V0

V4

V2

V3

V5

45

15

30

3

15

20

35

d[4][0] d[4][1] + d[1][0]

d[4][1] d[4][1] + d[1][1]

d[4][2] d[4][1] + d[1][2]

d[4][3] d[4][1] + d[1][3]

d[4][4] d[4][1] + d[1][4]

d[4][5] d[4][1] + d[1][5]

d41无边

d[5][0] d[5][1] + d[1][0]

d[5][1] d[5][1] + d[1][1]

d[5][2] d[5][1] + d[1][2]

d[5][3] d[5][1] + d[1][3]

d[5][4] d[5][1] + d[1][4]

d[5][5] d[5][1] + d[1][5]

d51无边 无变化

**第三次迭代 k = 2;**

d[0][0] d[0][2] + d[2][0]

V1

50

20

10

10

30

V0

V4

V2

V3

V5

45

15

30

3

15

20

35

35

35

d[0][1] d[0][2] + d[2][1] 1

d[0][2] d[0][2] + d[2][2]

d[0][3] d[0][2] + d[2][3] 10 + 15 = 35

d[0][4] d[0][2] + d[2][4]

d[0][5] d[0][2] + d[2][5]

增加了一条边

d[1][0] d[1][2] + d[2][0] 15 + 20 = 35

d[1][1] d[1][2] + d[2][1]

d[1][2] d[1][2] + d[2][2]

d[1][3] d[1][2] + d[2][3] 35

d[1][4] d[1][2] + d[2][4]

d[1][5] d[1][2] + d[2][5]

增加了2条边

d[2][0] d[2][2] + d[2][0]

d[2][1] d[2][2] + d[2][1]

d[2][2] d[2][2] + d[2][2]

d[2][3] d[2][2] + d[2][3]

d[2][4] d[2][2] + d[2][4]

d[2][5] d[2][2] + d[2][5]

无变化

d[3][0] d[3][2] + d[2][0] 55

d[3][1] d[3][2] + d[2][1]

d[3][2] d[3][2] + d[2][2]

d[3][3] d[3][2] + d[2][3]

d[3][4] d[3][2] + d[2][4]

V1

50

20

10

10

30

V0

V4

V2

V3

V5

45

15

30

3

15

20

35

35

35

d[3][5] d[3][2] + d[2][5]

d[4][0] d[4][2] + d[2][0]

d[4][1] d[4][2] + d[2][1]

d[4][2] d[4][2] + d[2][2]

d[4][3] d[4][2] + d[2][3]

d[4][4] d[4][2] + d[2][4]

d[4][5] d[4][2] + d[2][5]

d41无边

d[5][0] d[5][2] + d[2][0]

d[5][1] d[5][2] + d[2][1]

d[5][2] d[5][2] + d[2][2]

d[5][3] d[5][2] + d[2][3]

d[5][4] d[5][2] + d[2][4]

d[5][5] d[5][2] + d[2][5]

d52

无边 无变化