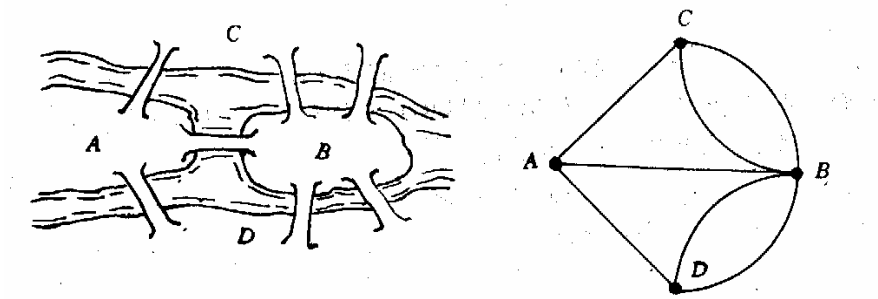
第五章　图与网络模型及方法

§1 概论

图论起源于 18 世纪。第一篇图论论文是瑞士数学家欧拉于 1736 年发表的“哥尼斯堡的七座桥”。1847 年，克希霍夫为了给出电网络方程而引进了“树”的概念。1857 年，凯莱在计数烷C_nH_{2n+2}的同分异构物时，也发现了“树”。哈密尔顿于 1859 年提出“周游世界”游戏，用图论的术语，就是如何找出一个连通图中的生成圈，近几十年来，由于计算机技术和科学的飞速发展，大大地促进了图论研究和应用，图论的理论和方法已经渗透到物理、化学、通讯科学、建筑学、生物遗传学、心理学、经济学、社会学等学科中。

图论中所谓的“图”是指某类具体事物和这些事物之间的联系。如果我们用点表示这些具体事物，用连接两点的线段（直的或曲的）表示两个事物的特定的联系，就得到了描述这个“图”的几何形象。图论为任何一个包含了一种二元关系的离散系统提供了一个数学模型，借助于图论的概念、理论和方法，可以对该模型求解。哥尼斯堡七桥问题就是一个典型的例子。在哥尼斯堡有七座桥将普莱格尔河中的两个岛及岛与河岸联结起来问题是要从这四块陆地中的任何一块开始通过每一座桥正好一次，再回到起点。当



然可以通过试验去尝试解决这个问题，但该城居民的任何尝试均未成功。欧拉为了解决这个问题，采用了建立数学模型的方法。他将每一块陆地用一个点来代替，将每一座桥用连接相应两点的一条线来代替，从而得到一个有四个“点”，七条“线”的“图”。问题成为从任一点出发一笔画出七条线再回到起点。欧拉考察了一般一笔画的结构特点，给出了一笔画的一个判定法则：这个图是连通的，且每个点都与偶数线相关联，将这个判定法则应用于七桥问题，得到了“不可能走通”的结果，不但彻底解决了这个问题，而且开创了图论研究的先河。

图与网络是运筹学（Operations Research）中的一个经典和重要的分支，所研究的问题涉及经济管理、工业工程、交通运输、计算机科学与信息技术、通讯与网络技术等诸多领域。下面将要讨论的最短路问题、最大流问题、最小费用流问题和匹配问题等都是图与网络的基本问题。

我们首先通过一些例子来了解网络优化问题。

例 1　最短路问题（SPP－shortest path problem）

一名货柜车司机奉命在最短的时间内将一车货物从甲地运往乙地。从甲地到乙地的公路网纵横交错，因此有多种行车路线，这名司机应选择哪条线路呢？假设货柜车的运行速度是恒定的，那么这一问题相当于需要找到一条从甲地到乙地的最短路。

例 2　公路连接问题

某一地区有若干个主要城市，现准备修建高速公路把这些城市连接起来，使得从其中任何一个城市都可以经高速公路直接或间接到达另一个城市。假定已经知道了任意两

个城市之间修建高速公路的成本，那么应如何决定在哪些城市间修建高速公路，使得总成本最小？

例 3　指派问题（assignment problem）

一家公司经理准备安排 N 名员工去完成 N 项任务，每人一项。由于各员工的特点不同，不同的员工去完成同一项任务时所获得的回报是不同的。如何分配工作方案可以使总回报最大？

例 4　中国邮递员问题（CPP－chinese postman problem）

一名邮递员负责投递某个街区的邮件。如何为他（她）设计一条最短的投递路线（从邮局出发，经过投递区内每条街道至少一次，最后返回邮局）？由于这一问题是我国管梅谷教授 1960 年首先提出的，所以国际上称之为中国邮递员问题。

例 5　旅行商问题（TSP－traveling salesman problem）

一名推销员准备前往若干城市推销产品。如何为他（她）设计一条最短的旅行路线（从驻地出发，经过每个城市恰好一次，最后返回驻地）？这一问题的研究历史十分悠久，通常称之为旅行商问题。

例 6　运输问题（transportation problem）

某种原材料有 M 个产地，现在需要将原材料从产地运往 N 个使用这些原材料的工厂。假定 M 个产地的产量和 N 家工厂的需要量已知，单位产品从任一产地到任一工厂的运费已知，那么如何安排运输方案可以使总运输成本最低？

上述问题有两个共同的特点：一是它们的目的都是从若干可能的安排或方案中寻求某种意义下的最优安排或方案，数学上把这种问题称为最优化或优化（optimization）问题；二是它们都易于用图形的形式直观地描述和表达，数学上把这种与图相关的结构称为网络（network）。与图和网络相关的最优化问题就是网络最优化或称网络优化（netwok optimization）问题。所以上面例子中介绍的问题都是网络优化问题。由于多数网络优化问题是以网络上的流（flow）为研究的对象，因此网络优化又常常被称为网络流（network flows）或网络流规划等。

下面首先简要介绍图与网络的一些基本概念。

§2 图与网络的基本概念

2.1 无向图

一个无向图(undirected graph) G 是由一个非空有限集合 $V(G)$ 和 $V(G)$ 中某些元素的无序对集合 $E(G)$ 构成的二元组，记为 $G=(V(G),E(G))$。其中 $V(G)=\{v_1,v_2,\Lambda,v_n\}$ 称为图 G 的顶点集（vertex set）或节点集（node set），$V(G)$ 中的每一个元素 $v_i(i=1,2,\Lambda,n)$ 称为该图的一个顶点（vertex）或节点（node）；$E(G)=\{e_1,e_2,\Lambda,e_m\}$ 称为图 G 的边集（edge set），$E(G)$ 中的每一个元素 e_k (即 $V(G)$ 中某两个元素 v_i,v_j 的无序对) 记为 $e_k=(v_i,v_j)$ 或 $e_k=v_iv_j=v_jv_i$ $(k=1,2,\Lambda,m)$，被称为该图的一条从 v_i 到 v_j 的边（edge）。

当边 $e_k=v_iv_j$ 时，称 v_i,v_j 为边 e_k 的端点，并称 v_j 与 v_i 相邻（adjacent）；边 e_k 称为与顶点 v_i,v_j 关联（incident）。如果某两条边至少有一个公共端点，则称这两条边在图 G 中相邻。

边上赋权的无向图称为赋权无向图或无向网络（undirected network）。我们对图和网络不作严格区分，因为任何图总是可以赋权的。

一个图称为**有限图**，如果它的顶点集和边集都有限。图 G 的顶点数用符号 $|V|$ 或

$n(G)$ 表示，边数用$|E|$或$e(G)$表示。

当讨论的图只有一个时，总是用G来表示这个图。从而在图论符号中我们常略去字母G，例如，分别用V,E,n 和e 代替$V(G),E(G),n(G)$和$e(G)$。

端点重合为一点的边称为**环**(loop)。

一个图称为**简单图**(simple graph)，如果它既没有环也没有两条边连接同一对顶点。

2.2 有向图

定义 一个有向图（directed graph 或 digraph）G是由一个非空有限集合V和V中某些元素的有序对集合A构成的二元组，记为$G=(V,A)$。其中$V=\{v_1,v_2,\Lambda,v_n\}$称为图$G$的顶点集或节点集，$V$中的每一个元素$v_i(i=1,2,\Lambda,n)$称为该图的一个顶点或节点；$A=\{a_1,a_2,\Lambda,a_m\}$称为图$G$的弧集（arc set），$A$中的每一个元素$a_k$(即$V$中某两个元素$v_i,v_j$的有序对) 记为$a_k=(v_i,v_j)$或$a_k=v_iv_j(k=1,2,\Lambda,n)$，被称为该图的一条从$v_i$到$v_j$的弧（arc）。

当弧$a_k=v_iv_j$时，称v_i为a_k的尾（tail），v_j为a_k的头（head），并称弧a_k为v_i的出弧（outgoing arc），为v_j的入弧（incoming arc）。

对应于每个有向图D，可以在相同顶点集上作一个图G，使得对于D的每条弧，G有一条有相同端点的边与之相对应。这个图称为D的基础图。反之，给定任意图G，对于它的每个边，给其端点指定一个顺序，从而确定一条弧，由此得到一个有向图，这样的有向图称为G的一个定向图。

以下若未指明“有向图”三字，“图”字皆指无向图。

2.3 完全图、二分图

每一对不同的顶点都有一条边相连的简单图称为完全图(complete graph)。n个顶点的完全图记为K_n。

若$V(G)=X\,\mathrm{Y}\,Y$，$X\,\mathrm{I}\,Y=\Phi$，$|X||Y|\neq 0$（这里$|X|$表示集合X中的元素个数），X中无相邻顶点对，Y中亦然，则称G为二分图(bipartite graph)；特别地，若$\forall x\in X,\forall y\in Y$，则$xy\in E(G)$，则称$G$为完全二分图，记成$K_{|X|,|Y|}$。

2.4 子图

图H叫做图G的**子图（subgraph）**，记作$H\subset G$，如果$V(H)\subset V(G)$，$E(H)\subset E(G)$。若H是G的子图，则G称为H的**母图**。

G的支撑子图（spanning subgraph，又成生成子图）是指满足$V(H)=V(G)$的子图H。

2.5 顶点的度

设$v\in V(G)$，G中与v关联的边数（每个环算作两条边）称为v的度(degree)，记作$d(v)$。若$d(v)$是奇数，称v是奇顶点(odd point)；$d(v)$是偶数，称v是偶顶点(even point)。关于顶点的度，我们有如下结果：

(i) $\sum_{v\in V}d(v)=2e$

(ii) 任意一个图的奇顶点的个数是偶数。

2.6 图与网络的数据结构

网络优化研究的是网络上的各种优化模型与算法.为了在计算机上实现网络优化的

算法，首先我们必须有一种方法（即数据结构）在计算机上来描述图与网络。一般来说，算法的好坏与网络的具体表示方法，以及中间结果的操作方案是有关系的。这里我们介绍计算机上用来描述图与网络的 5 种常用表示方法：邻接矩阵表示法、关联矩阵表示法、弧表表示法、邻接表表示法和星形表示法。在下面数据结构的讨论中，我们首先假设 $G=(V,A)$ 是一个简单有向图，$|V|=n,|A|=m$，并假设 V 中的顶点用自然数 $1,2,\Lambda\ ,n$ 表示或编号，A 中的弧用自然数 $1,2,\Lambda\ ,m$ 表示或编号。对于有多重边或无向网络的情况，我们只是在讨论完简单有向图的表示方法之后，给出一些说明。

（i）邻接矩阵表示法

邻接矩阵表示法是将图以邻接矩阵（adjacency matrix）的形式存储在计算机中。图 $G=(V,A)$ 的邻接矩阵是如下定义的：C 是一个 $n\times n$ 的 $0-1$ 矩阵，即

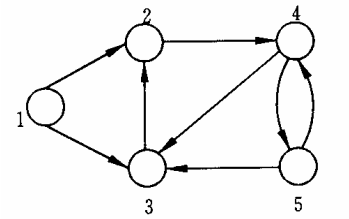
$$C=(c_{ij})_{n\times n}\in\{0,1\}^{n\times n},$$

$$c_{ij}=\begin{cases}1, & (i,j)\in A,\\ 0, & (i,j)\notin A.\end{cases}$$

也就是说，如果两节点之间有一条弧，则邻接矩阵中对应的元素为 1；否则为 0。可以看出，这种表示法非常简单、直接。但是，在邻接矩阵的所有 n^2 个元素中，只有 m 个为非零元。如果网络比较稀疏，这种表示法浪费大量的存储空间，从而增加了在网络中查找弧的时间。

例 7　对于右图所示的图，可以用邻接矩阵表示为

$$\begin{bmatrix}0&1&1&0&0\\0&0&0&1&0\\0&1&0&0&0\\0&0&1&0&1\\0&0&1&1&0\end{bmatrix}$$



同样，对于网络中的权，也可以用类似邻接矩阵的 $n\times n$ 矩阵表示。只是此时一条弧所对应的元素不再是 1，而是相应的权而已。如果网络中每条弧赋有多种权，则可以用多个矩阵表示这些权。

（ii）关联矩阵表示法

关联矩阵表示法是将图以关联矩阵（incidence matrix）的形式存储在计算机中．图 $G=(V,A)$ 的关联矩阵 B 是如下定义的：B 是一个 $n\times m$ 的矩阵，即

$$B=(b_{ik})_{n\times m}\in\{-1,0,1\}^{n\times m},$$

$$b_{ik}=\begin{cases}1, & \exists j\in V,k=(i,j)\in A,\\ -1, & \exists j\in V,k=(j,i)\in A,\\ 0, & \text{其它}.\end{cases}$$

也就是说，在关联矩阵中，每行对应于图的一个节点，每列对应于图的一条弧。如果一个节点是一条弧的起点，则关联矩阵中对应的元素为 1；如果一个节点是一条弧的终点，则关联矩阵中对应的元素为 -1；如果一个节点与一条弧不关联，则关联矩阵中对应的元素为 0。对于简单图，关联矩阵每列只含有两个非零元（一个 $+1$，一个 -1）。可以看出，这种表示法也非常简单、直接。但是，在关联矩阵的所有 nm 个元素中，只有 $2m$ 个为非零元。如果网络比较稀疏，这种表示法也会浪费大量的存储空间。但由于

关联矩阵有许多特别重要的理论性质，因此它在网络优化中是非常重要的概念。

例 8　对于例 7 所示的图，如果关联矩阵中每列对应弧的顺序为(1,2)，(1,3)，(2,4)，(3,2)，(4,3)，(4,5)，(5,3)和(5,4)，则关联矩阵表示为

$$\begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 & -1 & 0 & -1 & 0 \\ 0 & 0 & -1 & 0 & 1 & 1 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 1 & 1 \end{bmatrix}$$

同样，对于网络中的权，也可以通过对关联矩阵的扩展来表示。例如，如果网络中每条弧有一个权，我们可以把关联矩阵增加一行，把每一条弧所对应的权存储在增加的行中。如果网络中每条弧赋有多个权，我们可以把关联矩阵增加相应的行数，把每一条弧所对应的权存储在增加的行中。

（iii）弧表示法

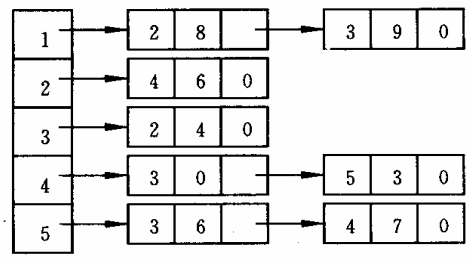
弧表表示法将图以弧表（arc list）的形式存储在计算机中。所谓图的弧表，也就是图的弧集合中的所有有序对。弧表表示法直接列出所有弧的起点和终点，共需 $2m$ 个存储单元，因此当网络比较稀疏时比较方便。此外，对于网络图中每条弧上的权，也要对应地用额外的存储单元表示。例如，例 7 所示的图，假设弧(1,2)，(1,3)，(2,4)，(3,2)，(4,3)，(4,5)，(5,3)和(5,4)上的权分别为 8，9，6，4，0，3，6 和 7，则弧表表示如下：

起点	1	1	2	3	4	4	5	5
终点	2	3	4	2	3	5	3	4
权	8	9	6	4	0	3	6	7

为了便于检索，一般按照起点、终点的字典序顺序存储弧表，如上面的弧表就是按照这样的顺序存储的。

（iv）邻接表表示法

邻接表表示法将图以邻接表（adjacency lists）的形式存储在计算机中。所谓图的邻接表，也就是图的所有节点的邻接表的集合；而对每个节点，它的邻接表就是它的所有出弧。邻接表表示法就是对图的每个节点，用一个单向链表列出从该节点出发的所有弧，链表中每个单元对应于一条出弧。为了记录弧上的权，链表中每个单元除列出弧的另一个端点外，还可以包含弧上的权等作为数据域。图的整个邻接表可以用一个指针数组表示。例如，例 7 所示的图，邻接表表示为



这是一个 5 维指针数组，每一维（上面表示法中的每一行）对应于一个节点的邻接表，如第 1 行对应于第 1 个节点的邻接表（即第 1 个节点的所有出弧）。每个指针单元的第 1 个数据域表示弧的另一个端点（弧的头），后面的数据域表示对应弧上的权。如第 1 行中的“2”表示弧的另一个端点为 2（即弧为（1,2））,“8”表示对应弧(1,2)上的权

为 8；“3”表示弧的另一个端点为 3（即弧为(1,3)），“9”表示对应弧（1，3）上的权为 9。又如，第 5 行说明节点 5 出发的弧有(5,3)、(5,4)，他们对应的权分别为 6 和 7。

对于有向图 $G=(V,A)$，一般用 $A(i)$ 表示节点 i 的邻接表，即节点 i 的所有出弧构成的集合或链表（实际上只需要列出弧的另一个端点，即弧的头）。例如上面例子，$A(1)=\{2,3\}$， $A(5)=\{3,4\}$ 等。这种记法我们在本书后面将会经常用到。

（v）星形表示法

星形（star）表示法的思想与邻接表表示法的思想有一定的相似之处。对每个节点，它也是记录从该节点出发的所有弧，但它不是采用单向链表而是采用一个单一的数组表示。也就是说，在该数组中首先存放从节点 1 出发的所有弧，然后接着存放从节点 2 出发的所有弧，依此类推，最后存放从节点 n 出发的所有弧。对每条弧，要依次存放其起点、终点、权的数值等有关信息。这实际上相当于对所有弧给出了一个顺序和编号，只是从同一节点出发的弧的顺序可以任意排列。此外，为了能够快速检索从每个节点出发的所有弧，我们一般还用一个数组记录每个节点出发的弧的起始地址（即弧的编号）。在这种表示法中，可以快速检索从每个节点出发的所有弧，这种星形表示法称为前向星形（forward star）表示法。

例如，在例 7 所示的图中，仍然假设弧（1,2），（l,3），（2,4），（3,2），（4,3），（4,5），（5,3）和（5,4）上的权分别为 8，9，6，4，0，3，6 和 7。此时该网络图可以用前向星形表示法表示如下：

节点对应的出弧的起始地址编号数组（记为 *point*）

节点号 i	1	2	3	4	5	6
起始地址 $point(i)$	1	3	4	6	7	9

记录弧信息的数组

弧编号	1	2	3	4	5	6	7	8
起点	1	1	2	3	4	4	5	5
终点	2	3	4	2	3	5	3	4
权	8	9	6	4	0	3	6	7

在数组 *point* 中，其元素个数比图的节点数多 1（即 $n+1$），且一定有 $point(1)=1$，$point(n+1)=m+1$。对于节点 i，其对应的出弧存放在弧信息数组的位置区间为

$$[point(i), point(i+1)-1],$$

如果 $point(i)=point(i+1)$，则节点 i 没有出弧。这种表示法与弧表表示法也非常相似，“记录弧信息的数组”实际上相当于有序存放的“弧表”。只是在前向星形表示法中，弧被编号后有序存放，并增加一个数组（ *point* ）记录每个节点出发的弧的起始编号。

前向星形表示法有利于快速检索每个节点的所有出弧，但不能快速检索每个节点的所有入弧。为了能够快速检索每个节点的所有入孤，可以采用反向星形（reverse star）表示法：首先存放进入节点 1 的所有孤，然后接着存放进入节点 2 的所有弧，依此类推，最后存放进入节点 n 的所有孤。对每条弧，仍然依次存放其起点、终点、权的数值等有关信息。同样，为了能够快速检索从每个节点的所有入弧，我们一般还用一个数组记录每个节点的入孤的起始地址（即弧的编号）。例如，例 7 所示的图，可以用反向星形表示法表示为如下形式：

节点对应的入弧的起始地址编号数组（记为 *rpoint*）

节点号 i	1	2	3	4	5	6

起始地址 $rpoint(i)$	1	1	3	6	8	9

记录弧信息的数组

弧编号	1	2	3	4	5	6	7	8
终点	2	2	3	3	3	4	4	5
起点	3	1	1	4	5	5	2	4
权	4	8	9	0	6	7	6	3

如果既希望快速检索每个节点的所有出弧，也希望快速检索每个节点的所有入弧，则可以综合采用前向和反向星形表示法。当然，将孤信息存放两次是没有必要的，可以只用一个数组（trace）记录一条弧在两种表示法中的对应关系即可。例如，可以在采用前向星形表示法的基础上，加上上面介绍的 *rpoint* 数组和如下的 *trace* 数组即可。这相当于一种紧凑的双向星形表示法如下所示：

两种表示法中的弧的对应关系（记为 *trace*）

反向法中弧编号 j	1	2	3	4	5	6	7	8
正向法中弧编号 $trace(j)$	4	1	2	5	7	8	3	6

对于网络图的表示法，我们作如下说明：

① 星形表示法和邻接表表示法在实际算法实现中都是经常采用的。星形表示法的优点是占用的存储空间较少，并且对那些不提供指针类型的语言（如 FORTRAN 语言等）也容易实现。邻接表表示法对那些提供指针类型的语言（如 C 语言等）是方便的，且增加或删除一条弧所需的计算工作量很少，而这一操作在星形表示法中所需的计算工作量较大（需要花费 $O(m)$ 的计算时间）。有关“计算时间”的观念是网络优化中需要考虑的一个关键因素。

② 当网络不是简单图，而是具有平行弧（即多重弧）时，显然此时邻接矩阵表示法是不能采用的。其他方法则可以很方便地推广到可以处理平行弧的情形。

③ 上述方法可以很方便地推广到可以处理无向图的情形，但由于无向图中边没有方向，因此可能需要做一些自然的修改。例如，可以在计算机中只存储邻接矩阵的一半信息（如上三角部分），因为此时邻接矩阵是对称矩阵。无向图的关联矩阵只含有元素 0 和 $+1$，而不含有 -1，因为此时不区分边的起点和终点。又如，在邻接表和星形表示法中，每条边会被存储两次，而且反向星形表示显然是没有必要的，等等。

2.7 轨与连通

$W = v_0 e_1 v_1 e_2 \Lambda\ e_k v_k$，其中 $e_i \in E(G)$，$1 \le i \le k$，$v_j \in V(G)$，$0 \le j \le k$，e_i 与 v_{i-1}, v_i 关联，称 W 是图 G 的一条道路(walk)，k 为路长，顶点 v_0 和 v_k 分别称为 W 的起点和终点，而 $v_1, v_2, \Lambda\ , v_{k-1}$ 称为它的内部顶点。

若道路 W 的边互不相同，则 W 称为迹(trail)。若道路 W 的顶点互不相同，则 W 称为轨(path)。

称一条道路是闭的，如果它有正的长且起点和终点相同。起点和终点重合的轨叫做圈(cycle)。

若图 G 的两个顶点 u, v 间存在道路，则称 u 和 v 连通(connected)。u, v 间的最短轨的长叫做 u, v 间的距离。记作 $d(u, v)$。若图 G 的任二顶点均连通，则称 G 是连通图。

显然有：

(i) 图P是一条轨的充要条件是P是连通的，且有两个一度的顶点，其余顶点的度为 2；

(ii) 图C是一个圈的充要条件是C是各顶点的度均为 2 的连通图。

§3 应用—最短路问题

3.1 两个指定顶点之间的最短路径

问题如下：给出了一个连接若干个城镇的铁路网络，在这个网络的两个指定城镇间，找一条最短铁路线。

以各城镇为图G的顶点，两城镇间的直通铁路为图G相应两顶点间的边，得图G。对G的每一边e，赋以一个实数$w(e)$—直通铁路的长度，称为e的权，得到赋权图G。G的子图的权是指子图的各边的权和。问题就是求赋权图G中指定的两个顶点u_0, v_0间的具最小权的轨。这条轨叫做u_0, v_0间的最短路，它的权叫做u_0, v_0间的距离，亦记作$d(u_0, v_0)$。

求最短路已有成熟的算法：迪克斯特拉（Dijkstra）算法，其基本思想是按距u_0从近到远为顺序，依次求得u_0到G的各顶点的最短路和距离，直至v_0（或直至G的所有顶点），算法结束。为避免重复并保留每一步的计算信息，采用了标号算法。下面是该算法。

(i) 令$l(u_0)=0$，对$v \neq u_0$，令$l(v)=\infty$，$S_0=\{u_0\}$，$i=0$。

(ii) 对每个$v \in \bar{S}_i$（$\bar{S}_i = V \setminus S_i$），用

$$\min_{u \in S_i} \{l(v), l(u)+w(uv)\}$$

代替$l(v)$。计算$\min_{v \in \bar{S}_i}\{l(v)\}$，把达到这个最小值的一个顶点记为$u_{i+1}$，令$S_{i+1}=S_i \cup \{u_{i+1}\}$。

(iii). 若$i=|V|-1$，停止；若$i<|V|-1$，用$i+1$代替i，转(ii)。

算法结束时，从u_0到各顶点v的距离由v的最后一次的标号$l(v)$给出。在v进入S_i之前的标号$l(v)$叫 T 标号，v进入S_i时的标号$l(v)$叫 P 标号。算法就是不断修改各项点的 T 标号，直至获得 P 标号。若在算法运行过程中，将每一顶点获得 P 标号所由来的边在图上标明，则算法结束时，u_0至各项点的最短路也在图上标示出来了。

例 9　某公司在六个城市c_1, c_2, Λ, c_6中有分公司，从c_i到c_j的直接航程票价记在下述矩阵的(i, j)位置上。（∞表示无直接航路），请帮助该公司设计一张城市c_1到其它城市间的票价最便宜的路线图。

$$\begin{bmatrix} 0 & 50 & \infty & 40 & 25 & 10 \\ 50 & 0 & 15 & 20 & \infty & 25 \\ \infty & 15 & 0 & 10 & 20 & \infty \\ 40 & 20 & 10 & 0 & 10 & 25 \\ 25 & \infty & 20 & 10 & 0 & 55 \\ 10 & 25 & \infty & 25 & 55 & 0 \end{bmatrix}$$

用矩阵 $a_{n\times n}$（n 为顶点个数）存放各边权的邻接矩阵，行向量 pb、$index_1$、$index_2$、d 分别用来存放 P 标号信息、标号顶点顺序、标号顶点索引、最短通路的值。其中分量

$$pb(i)=\begin{cases}1 & \text{当第}i\text{顶点已标号}\\ 0 & \text{当第}i\text{顶点未标号}\end{cases};$$

$index_2(i)$ 存放始点到第 i 点最短通路中第 i 顶点前一顶点的序号；

$d(i)$ 存放由始点到第 i 点最短通路的值。

求第一个城市到其它城市的最短路径的 Matlab 程序如下：

```
clear;
clc;
M=10000;
a(1,:)=[0,50,M,40,25,10];
a(2,:)=[zeros(1,2),15,20,M,25];
a(3,:)=[zeros(1,3),10,20,M];
a(4,:)=[zeros(1,4),10,25];
a(5,:)=[zeros(1,5),55];
a(6,:)=zeros(1,6);
a=a+a';
pb(1:length(a))=0;pb(1)=1;index1=1;index2=ones(1,length(a));
d(1:length(a))=M;d(1)=0;temp=1;
while sum(pb)<length(a)
    tb=find(pb==0);
    d(tb)=min(d(tb),d(temp)+a(temp,tb));
    tmpb=find(d(tb)==min(d(tb)));
    temp=tb(tmpb(1));
    pb(temp)=1;
    index1=[index1,temp];
    index=index1(find(d(index1)==d(temp)-a(temp,index1)));
    if length(index)>=2
        index=index(1);
    end
    index2(temp)=index;
end
d, index1, index2
```

3.2 每对顶点之间的最短路径

计算赋权图中各对顶点之间最短路径，显然可以调用 Dijkstra 算法。具体方法是：每次以不同的顶点作为起点，用 Dijkstra 算法求出从该起点到其余顶点的最短路径，反复执行 n 次这样的操作，就可得到从每一个顶点到其它顶点的最短路径。这种算法的时间复杂度为 $O(n^3)$。第二种解决这一问题的方法是由 Floyd R W 提出的算法，称之为 Floyd 算法。

假设图 G 权的邻接矩阵为 A_0，

$$A_0=\begin{bmatrix} a_{11} & a_{12} & \Lambda & a_{1n} \\ a_{21} & a_{22} & \Lambda & a_{2n} \\ \mathrm{M} & \mathrm{M} & \Lambda & \mathrm{M} \\ a_{n1} & a_{n2} & \Lambda & a_{nn} \end{bmatrix}$$

来存放各边长度，其中：

$a_{ii}=0 \quad i=1,2,\Lambda\ ,n$；

$a_{ij}=\infty$ i,j之间没有边，在程序中以各边都不可能达到的充分大的数代替；

$a_{ij}=w_{ij}$ w_{ij}是i,j之间边的长度，$i,j=1,2,\Lambda\ ,n$。

对于无向图，A_0是对称矩阵，$a_{ij}=a_{ji}$。

Floyd 算法的基本思想是：递推产生一个矩阵序列$A_0,A_1,\Lambda\ ,A_k,\Lambda\ ,A_n$，其中$A_k(i,j)$表示从顶点$v_i$到顶点$v_j$的路径上所经过的顶点序号不大于$k$的最短路径长度。

计算时用迭代公式：

$$A_k(i,j)=\min(A_{k-1}(i,j),A_{k-1}(i,k)+A_{k-1}(k,j))$$

k是迭代次数，$i,j,k=1,2,\Lambda\ ,n$。

最后，当$k=n$时，A_n即是各顶点之间的最短通路值。

例10 用Floyd算法求解例1。

矩阵path用来存放每对顶点之间最短路径上所经过的顶点的序号。Floyd算法的Matlab程序如下：

```
clear;
clc;
M=10000;
a(1,:)=[0,50,M,40,25,10];
a(2,:)=[zeros(1,2),15,20,M,25];
a(3,:)=[zeros(1,3),10,20,M];
a(4,:)=[zeros(1,4),10,25];
a(5,:)=[zeros(1,5),55];
a(6,:)=zeros(1,6);
b=a+a';path=zeros(length(b));
for k=1:6
    for i=1:6
        for j=1:6
            if b(i,j)>b(i,k)+b(k,j)
                b(i,j)=b(i,k)+b(k,j);
                path(i,j)=k;
            end
        end
    end
end
b, path
```

§4 树

4.1 基本概念

连通的无圈图叫做树，记之为T。若图G满足$V(G)=V(T)$，$E(T)\subset E(G)$，则称T是G的生成树。图G连通的充分必要条件为G有生成树。一个连通图的生成树的个数很多，用$t(G)$表示G的生成树的个数，则有公式

公式 (Caylay) $t(K_n)=n^{n-2}$。

公式 $t(G)=t(G-e)+t(G\cdot e)$。

其中$G-e$表示从G上删除边e，$G\cdot e$表示把e的长度收缩为零得到的图。

树有下面常用的五个充要条件。

定理 1 （i）G是树当且仅当G中任二顶点之间有且仅有一条轨道。

（ii）G是树当且仅当G无圈，且$e=n-1$。

（iii）G是树当且仅当G连通，且$e=n-1$。

（iv）G是树当且仅当G连通，且$\forall e\in E(G)$，$G-e$不连通。

（v）G是树当且仅当G无圈，$\forall e\notin E(G)$，$G+e$恰有一个圈。

4.2 应用—连线问题

欲修筑连接n个城市的铁路，已知i城与j城之间的铁路造价为C_{ij}，设计一个线路图，使总造价最低。

连线问题的数学模型是在连通赋权图上求权最小的生成树。赋权图的具有最小权的生成树叫做最小生成树。

下面介绍构造最小生成树的两种常用算法。

4.2.1 prim 算法构造最小生成树

设置两个集合P和Q,其中P用于存放G的最小生成树中的顶点，集合Q存放G的最小生成树中的边。令集合P的初值为$P=\{v_1\}$（假设构造最小生成树时，从顶点v_1出发），集合Q的初值为$Q=\Phi$。prim 算法的思想是，从所有$p\in P$，$v\in V-P$的边中，选取具有最小权值的边pv，将顶点v加入集合P中，将边pv加入集合Q中，如此不断重复，直到$P=V$时，最小生成树构造完毕，这时集合Q中包含了最小生成树的所有边。

prim 算法如下：

（i）$P=\{v_1\}$，$Q=\Phi$；

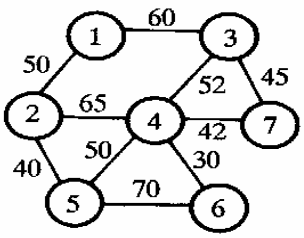
（ii）while $P\sim=V$

$pv=\min(w_{pv},p\in P,v\in V-P\}$

$P=P+\{v\}$

$Q=Q+\{pv\}$

end



例 11 用 prim 算法求右图的最小生成树。

我们用$result_{3\times n}$的第一、二、三行分别表示生成树边的起点、终点、权集合。Matlab 程序如下：

```
clc;clear;
M=1000;
a(1,2)=50; a(1,3)=60;
a(2,4)=65; a(2,5)=40;
a(3,4)=52;a(3,7)=45;
```

```
a(4,5)=50; a(4,6)=30;a(4,7)=42;
a(5,6)=70;
a=[a;zeros(2,7)];
a=a+a';a(find(a==0))=M;
result=[];p=1;tb=2:length(a);
while length(result)~=length(a)-1
    temp=a(p,tb);temp=temp(:);
    d=min(temp);
    [jb,kb]=find(a(p,tb)==d);
    j=p(jb(1));k=tb(kb(1));
    result=[result,[j;k;d]];p=[p,k];tb(find(tb==k))=[];
end
result
```

4.2.1 Kruskal 算法构造最小生成树

科茹斯克尔（Kruskal）算法是一个好算法。Kruskal 算法如下:

(i)选 $e_1 \in E(G)$，使得 $w(e_1) = \min$ 。

(ii)若 e_1, e_2, Λ, e_i 已选好，则从 $E(G) - \{e_1, e_2, \Lambda, e_i\}$ 中选取 e_{i+1}，使得

① $G[\{e_1, e_2, \Lambda, e_i, e_{i+1}\}]$ 中无圈，且

② $w(e_{i+1}) = \min$ 。

(iii)直到选得 e_{n-1} 为止。

例 12 用 Kruskal 算法构造例 3 的最小生成树。

我们用 $index_{2\times n}$ 存放各边端点的信息，当选中某一边之后，就将此边对应的顶点序号中较大序号 u 改记为此边的另一序号 v，同时把后面边中所有序号为 u 的改记为 v。此方法的几何意义是：将序号 u 的这个顶点收缩到 v 顶点，u 顶点不复存在。后面继续寻查时，发现某边的两个顶点序号相同时，认为已被收缩掉，失去了被选取的资格。

Matlab 程序如下：

```
clc;clear;
M=1000;
a(1,2)=50; a(1,3)=60;
a(2,4)=65; a(2,5)=40;
a(3,4)=52;a(3,7)=45;
a(4,5)=50; a(4,6)=30;a(4,7)=42;
a(5,6)=70;
[i,j]=find((a~=0)&(a~=M));
b=a(find((a~=0)&(a~=M)));
data=[i';j';b'];index=data(1:2,:);
loop=max(size(a))-1;
result=[];
while length(result)<loop
    temp=min(data(3,:));
    flag=find(data(3,:)==temp);
    flag=flag(1);
    v1=data(1,flag);v2=data(2,flag);
    if index(1,flag)~=index(2,flag)
        result=[result,data(:,flag)];
```

```
    end
    if v1>v2
        index(find(index==v1))=v2;
    else
        index(find(index==v2))=v1;
    end
    data(:,flag)=[];
    index(:,flag)=[];
end
result
```

§5　匹配问题

定义　若 $M \subset E(G)$，$\forall e_i, e_j \in M$，e_i 与 e_j 无公共端点（$i \neq j$），则称 M 为图 G 的一个对集；M 中的一条边的两个端点叫做在对集 M 中相配；M 中的端点称为被 M 许配；G 中每个顶点皆被 M 许配时，M 称为完美对集；G 中已无使 $|M'| > |M|$ 的对集 M'，则 M 称为最大对集；若 G 中有一轨，其边交替地在对集 M 内外出现，则称此轨为 M 的交错轨，交错轨的起止顶点都未被许配时，此交错轨称为可增广轨。

若把可增广轨上在 M 外的边纳入对集，把 M 内的边从对集中删除，则被许配的顶点数增加 2，对集中的“对儿”增加一个。

1957 年，贝尔热(Berge)得到最大对集的充要条件：

定理 2　M 是图 G 中的最大对集当且仅当 G 中无 M 可增广轨。

1935 年，霍尔(Hall)得到下面的许配定理：

定理 3　G 为二分图，X 与 Y 是顶点集的划分，G 中存在把 X 中顶点皆许配的对集的充要条件是，$\forall S \subset X$，则 $|N(S)| \geq |S|$，其中 $N(S)$ 是 S 中顶点的邻集。

由上述定理可以得出：

推论 1：若 G 是 k（$k > 0$）正则 2 分图，则 G 有完美对集。

所谓 k 正则图，即每顶点皆 k 度的图。

由此推论得出下面的婚配定理：

定理 4　每个姑娘都结识 $k(k \geq 1)$ 位小伙子，每个小伙子都结识 k 位姑娘，则每位姑娘都能和她认识的一个小伙子结婚，并且每位小伙子也能和他认识的一个姑娘结婚。

人员分派问题等实际问题可以化成对集来解决。

人员分派问题：工作人员 x_1, x_2, Λ, x_n 去做 n 件工作 y_1, y_2, Λ, y_n，每人适合做其中一件或几件，问能否每人都有一份适合的工作？如果不能，最多几人可以有适合的工作？

这个问题的数学模型是：G 是二分图，顶点集划分为 $V(G) = X \text{ Y} Y$，$X_1 = \{x_1, \Lambda, x_n\}$，$Y_1 = \{y_1, \Lambda, y_n\}$，当且仅当 x_i 适合做工作 y_i 时，$x_i y_i \in E(G)$，求 G 中的最大对集。

解决这个问题可以利用 1965 年埃德门兹(Edmonds)提出的匈牙利算法。

匈牙利算法：

（i）从 G 中任意取定一个初始对集 M。

（ii）若 M 把 X 中的顶点皆许配，停止，M 即完美对集；否则取 X 中未被 M 许配的一顶点 u，记 $S = \{u\}$，$T = \Phi$。

（iii）若 $N(S) = T$，停止，无完美对集；否则取 $y \in N(S) - T$。

（iv）若 y 是被 M 许配的，设 $yz\in M$， $S=S\,\Upsilon\{z\}$， $T=T\,\Upsilon\{y\}$，转（iii）；否则，取可增广轨 $P(u,y)$，令 $M=(M-E(P))\,\Upsilon\,(E(P)-M)$，转（ii）。

把以上算法稍加修改就能够用来求二分图的最大对集。

最优**分派问题**：在人员分派问题中，工作人员适合做的各项工作当中，效益未必一致，我们需要制定一个分派方案，使公司总效益最大。

这个问题的数学模型是：在人员分派问题的模型中，图 G 的每边加了权 $w(x_iy_j)\geq 0$，表示 x_i 干 y_j 工作的效益，求加权图 G 上的权最大的完美对集。

解决这个问题可以用库恩—曼克莱斯（Kuhn-Munkres）算法。为此，我们要引入可行顶点标号与相等子图的概念。

定义 若映射 $l:V(G)\to R$，满足 $\forall x\in X, y\in Y$，

$$l(x)+l(y)\geq w(x,y),$$

则称 l 是二分图 G 的可行顶点标号。令

$$E_l=\{xy\mid xy\in E(G), l(x)+l(y)=w(xy)\},$$

称以 E_l 为边集的 G 的生成子图为相等子图，记作 G_l。

可行顶点标号是存在的。例如

$$l(x)=\max_{y\in Y} w(xy),\quad x\in X;$$

$$l(y)=0,\qquad\qquad y\in Y。$$

定理 5 G_l **的**完美对集即为 G 的权最大的完美对集。

Kuhn-Munkres 算法

（i）选定初始可行顶点标号 l，确定 G_l，在 G_l 中选取一个对集 M。

（ii）若 X 中顶点皆被 M 许配，停止，M 即 G 的权最大的完美对集；否则，取 G_l 中未被 M 许配的顶点 u，令 $S=\{u\}$， $T=\Phi$。

(iii)若 $N_{G_l}(S)\supset T$，转（iv）；若 $N_{G_l}(S)=T$，取

$$a_l=\min_{x\in S, y\notin T}\{l(x)+l(y)-w(xy)\},$$

$$\bar{l}(v)=\begin{cases} l(v)-a_l, & v\in S\\ l(v)+a_l, & v\in T\\ l(v), & \text{其它}\end{cases},$$

$l=\bar{l}$， $G_l=G_{\bar{l}}$。

（iv）选 $N_{G_l}(S)-T$ 中一顶点 y，若 y 已被 M 许配，且 $yx\in M$，则 $S=S\,\Upsilon\{z\}$，$T=T\,\Upsilon\{y\}$，转（iii）；否则，取 G_l 中一个 M 可增广轨 $P(u,y)$，令

$$M=(M-E(P))\,\Upsilon\,(E(P)-M),$$

转（ii）。

其中 $N_{G_l}(S)$ 是 G_l 中 S 的相邻顶点集。

§6 Euler 图和 Hamilton 图

6.1 基本概念

定义 经过 G 的每条边的迹叫做 G 的 Euler 迹；闭的 Euler 迹叫做 Euler 回路或 E

回路；含 Euler 回路的图叫做 Euler 图。

直观地讲，Euler 图就是从一顶点出发每边恰通过一次能回到出发点的那种图，即不重复地行遍所有的边再回到出发点。

定理 7 （i）G 是 Euler 图的充分必要条件是 G 连通且每顶点皆偶次。

（ii）G 是 Euler 图的充分必要条件是 G 连通且 $G=\bigcup_{i=1}^{d} C_i$，C_i 是圈，$E(C_i)\cap E(C_j)=\Phi(i\neq j)$。

（iii）G 中有 Euler 迹的充要条件是 G 连通且至多有两个奇次点。

定义 包含 G 的每个顶点的轨叫做 Hamilton(哈密顿)轨；闭的 Hamilton 轨叫做 Hamilton 圈或 H 圈；含 Hamilton 圈的图叫做 Hamilton 图。

直观地讲，Hamilton 图就是从一顶点出发每顶点恰通过一次能回到出发点的那种图，即不重复地行遍所有的顶点再回到出发点。

6.2 Euler 回路的 Fleury 算法

1921 年，Fleury 给出下面的求 Euler 回路的算法。

Fleury 算法：

1^o. $\forall v_0\in V(G)$，令 $W_0=v_0$。

2^o. 假设迹 $W_i=v_0e_1v_1\Lambda\ e_iv_i$ 已经选定，那么按下述方法从 $E-\{e_1,\Lambda\ ,e_i\}$ 中选取边 e_{i+1}：

（i）e_{i+1} 和 v_i 相关联；

（ii）除非没有别的边可选择，否则 e_{i+1} 不是 $G_i=G-\{e_1,\Lambda\ ,e_i\}$ 的割边(cut edge)。(所谓割边是一条删除后使连通图不再连通的边)。

3^o. 当第 2 步不能再执行时，算法停止。

6.3 应用

6.3.1 邮递员问题

中国邮递员问题

一位邮递员从邮局选好邮件去投递，然后返回邮局，当然他必须经过他负责投递的每条街道至少一次，为他设计一条投递路线，使得他行程最短。

上述中国邮递员问题的数学模型是：在一个赋权连通图上求一个含所有边的回路，且使此回路的权最小。

显然，若此连通赋权图是 Euler 图，则可用 Fleury 算法求 Euler 回路，此回路即为所求。

对于非 Euler 图，1973 年，Edmonds 和 Johnson 给出下面的解法：

设 G 是连通赋权图。

（i）求 $V_0=\{v\mid v\in V(G),d(v)=1(\mathrm{mod}\,2)\}$。

（ii）对每对顶点 $u,v\in V_0$，求 $d(u,v)$（$d(u,v)$ 是 u 与 v 的距离，可用 Floyd 算法求得）。

（iii）构造完全赋权图 $K_{|V_0|}$，以 V_0 为顶点集，以 $d(u,v)$ 为边 uv 的权。

（iv）求 $K_{|V_0|}$ 中权之和最小的完美对集 M。

（v）求 M 中边的端点之间的在 G 中的最短轨。

（vi）在（v）中求得的每条最短轨上每条边添加一条等权的所谓“倍边”（即共端点

共权的边）。

（vii）在（vi）中得的图G'上求 Euler 回路即为中国邮递员问题的解。

多邮递员问题：

邮局有$k(k \geq 2)$位投递员，同时投递信件，全城街道都要投递，完成任务返回邮局，如何分配投递路线，使得完成投递任务的时间最早？我们把这一问题记成 kPP。

kPP 的数学模型如下：

$G(V,E)$是连通图，$v_0 \in V(G)$，求G的回路$C_1,\Lambda\ ,C_k$，使得

（i）$v_0 \in V(C_i)$，$i=1,2,\Lambda\ ,k$，

（ii）$\max\limits_{1\leq i\leq k} \sum\limits_{e\in E(C_i)} w(e) = \min$，

（iii）$\bigcup\limits_{i=1}^{k} E(C_i) = E(G)$

6.3.2 旅行商（TSP）问题

一名推销员准备前往若干城市推销产品，然后回到他的出发地。如何为他设计一条最短的旅行路线（从驻地出发，经过每个城市恰好一次，最后返回驻地）？这个问题称为旅行商问题。用图论的术语说，就是在一个赋权完全图中，找出一个有最小权的 Hamilton 圈。称这种圈为最优圈。与最短路问题及连线问题相反，目前还没有求解旅行商问题的有效算法。所以希望有一个方法以获得相当好（但不一定最优）的解。

一个可行的办法是首先求一个 Hamilton 圈C，然后适当修改C以得到具有较小权的另一个 Hamilton 圈。修改的方法叫做改良圈算法。设初始圈$C = v_1v_2\Lambda\ v_nv_1$。

（i）对于$1<i+1<j<n$，构造新的 Hamilton 圈:

$$C_{ij} = v_1v_2\Lambda\ v_iv_jv_{j-1}v_{j-2}\Lambda\ v_{i+1}v_{j+1}v_{j+2}\Lambda\ v_nv_1,$$

它是由C中删去边v_iv_{i+1}和v_jv_{j+1}，添加边v_iv_j和$v_{i+1}v_{j+1}$而得到的。若$w(v_iv_j)+w(v_{i+1}v_{j+1})<w(v_iv_{i+1})+w(v_jv_{j+1})$，则以$C_{ij}$代替$C$，$C_{ij}$叫做$C$的改良圈。

（ii）转（i），直至无法改进，停止。

用改良圈算法得到的结果几乎可以肯定不是最优的。为了得到更高的精确度，可以选择不同的初始圈，重复进行几次算法，以求得较精确的结果。

这个算法的优劣程度有时能用 Kruskal 算法加以说明。假设C是G中的最优圈。则对于任何顶点v，$C-v$是在$G-v$中的 Hamilton 轨，因而也是$G-v$的生成树。由此推知：若T是$G-v$中的最优树，同时e和f是和v关联的两条边，并使得$w(e)+w(f)$尽可能小，则$w(T)+w(e)+w(f)$将是$w(C)$的一个下界。

这里介绍的方法已被进一步发展。圈的修改过程一次替换三条边比一次仅替换两条边更为有效；然而，有点奇怪的是，进一步推广这一想法，就不利了。

例 13 从北京（Pe）乘飞机到东京(T)、纽约(N)、墨西哥城(M)、伦敦(L)、巴黎(Pa)五城市做旅游，每城市恰去一次再回北京，应如何安排旅游线，使旅程最短？各城市之间的航线距离如下表：

	L	M	N	Pa	Pe	T
L		56	35	21	51	60
M	56		21	57	78	70
N	35	21		36	68	68
Pa	21	57	36		51	61

Pe	51	78	68	51		13
T	60	70	68	61	13	

解：编写程序如下：

```
clc,clear
a(1,2)=56;a(1,3)=35;a(1,4)=21;a(1,5)=51;a(1,6)=60;
a(2,3)=21;a(2,4)=57;a(2,5)=78;a(2,6)=70;
a(3,4)=36;a(3,5)=68;a(3,6)=68;
a(4,5)=51;a(4,6)=61;
a(5,6)=13;
a(6,:)=0;
a=a+a';
c1=[5 1:4 6];
L=length(c1);
flag=1;
while flag>0
        flag=0;
    for m=1:L-3
        for n=m+2:L-1
            if a(c1(m),c1(n))+a(c1(m+1),c1(n+1))<a(c1(m),c1(m+1))+a(c1(n),c1(n+1))
                flag=1;
                c1(m+1:n)=c1(n:-1:m+1);
            end
        end
    end
end
sum1=0;
for i=1:L-1
    sum1=sum1+a(c1(i),c1(i+1));
end
circle=c1;
sum=sum1;
c1=[5 6 1:4];%改变初始圈，该算法的最后一个顶点不动
flag=1;
while flag>0
        flag=0;
    for m=1:L-3
        for n=m+2:L-1
            if a(c1(m),c1(n))+a(c1(m+1),c1(n+1))<...
                    a(c1(m),c1(m+1))+a(c1(n),c1(n+1))
                flag=1;
                c1(m+1:n)=c1(n:-1:m+1);
            end
        end
    end
end
sum1=0;
for i=1:L-1
    sum1=sum1+a(c1(i),c1(i+1));
end
if sum1<sum
```

```
    sum=sum1;
    circle=c1;
end
circle,sum
```

§7 最大流问题

7.1 最大流问题的数学描述

7.1.1 网络中的流

定义 在以V为节点集，A为弧集的有向图$G=(V,A)$上定义如下的权函数：

（i）$L:A\to R$为孤上的权函数，弧$(i,j)\in A$对应的权$L(i,j)$记为l_{ij}，称为孤(i,j)的容量下界（lower bound）；

（ii）$U:A\to R$为弧上的权函数，弧$(i,j)\in A$对应的权$U(i,j)$记为u_{ij}，称为孤(i,j)的容量上界，或直接称为容量（capacity）；

（iii）$D:V\to R$为顶点上的权函数，节点$i\in V$对应的权$D(i)$记为d_i，称为顶点i的供需量（supply / demand）；

此时所构成的网络称为流网络，可以记为

$N=(V,A,L,U,D)$。

由于我们只讨论V,A为有限集合的情况，所以对于弧上的权函数L,U和顶点上的权函数D，可以直接用所有孤上对应的权组成的有限维向量表示，因此L,U,D有时直接称为权向量，或简称权。由于给定有向图$G=(V,A)$后，我们总是可以在它的弧集合和顶点集合上定义各种权函数，所以流网络一般也直接简称为网络。

在流网络中，弧(i,j)的容量下界l_{ij}和容量上界u_{ij}表示的物理意义分别是：通过该弧发送某种“物质”时，必须发送的最小数量为l_{ij}，而发送的最大数量为u_{ij}。顶点$i\in V$对应的供需量d_i则表示该顶点从网络外部获得的“物质”数量（$d_i<0$时），或从该顶点发送到网络外部的“物质”数量（$d_i>0$时）。下面我们给出严格定义。

定义 对于流网络$N=(V,A,L,U,D)$，其上的一个**流**（flow）f是指从N的弧集A到R的一个函数，即对每条弧(i,j)赋予一个实数f_{ij}（称为弧(i,j)的流量）。如果流f满足

$$\sum_{j:(i,j)\in A} f_{ij}-\sum_{j:(j,i)\in A} f_{ji}=d_i,\quad \forall i\in V, \tag{1}$$

$$l_{ij}\le f_{ij}\le u_{ij},\quad \forall (i,j)\in A, \tag{2}$$

则称f为可行流（feasible flow）。至少存在一个可行流的流网络称为可行网络（feasible network）．约束（1）称为流量守恒条件（也称流量平衡条件），约束（2）称为容量约束。

可见，当$d_i>0$时，表示有d_i个单位的流量从该顶点流出，因此顶点i称为供应点（supply node）或源（source），有时也形象地称为起始点或发点等；当$d_i<0$时，表示有$|d_i|$个单位的流量流入该点（或说被该顶点吸收），因此顶点i称为需求点（demand node）或汇（sink），有时也形象地称为终止点或收点等；当$d_i=0$时，顶点i称为转

运点（transshipment node）或平衡点、中间点等。此外，根据（1）可知，对于可行网络，必有

$$\sum_{i\in V} d_i = 0 \qquad (3)$$

也就是说，所有节点上的供需量之和为 0 是网络中存在可行流的必要条件。

一般来说，我们总是可以把 $L \neq 0$ 的流网络转化为 $L = 0$ 的流网络进行研究。所以，除非特别说明，以后我们总是假设 $L = 0$（即所有孤 (i, j) 的容量下界 $l_{ij} = 0$），并将 $L = 0$ 时的流网络简记为 $N = (V, A, U, D)$。此时，相应的容量约束（2）为

$$0 \le x_{ij} \le u_{ij}, \quad \forall (i, j) \in A \text{。}$$

定义 在流网络 $N = (V, A, U, D)$ 中，对于流 f，如果

$$f_{ij} = 0, \quad \forall (i, j) \in A,$$

则称 f 为零流，否则为非零流。如果某条弧 (i, j) 上的流量等于其容量（$f_{ij} = u_{ij}$），则称该弧为饱和弧（saturated arc）；如果某条弧 (i, j) 上的流量小于其容量（$f_{ij} < u_{ij}$），则称该弧为非饱和弧；如果某条弧 (i, j) 上的流量为 0（$f_{ij} = 0$），则称该弧为空弧（void arc）。

7.1.2 最大流问题

考虑如下流网络 $N = (V, A, U, D)$：节点 s 为网络中唯一的源点，t 为唯一的汇点，而其它节点为转运点。如果网络中存在可行流 f，此时称流 f 的流量（或流值，flow value）为 d_s（根据（3），它自然也等于 $-d_t$），通常记为 v 或 $v(f)$，即

$$v = v(f) = d_s = -d_t \text{。}$$

对这种单源单汇的网络，如果我们并不给定 d_s 和 d_t（即流量不给定），则网络一般记为 $N = (s, t, V, A, U)$。最大流问题（maximum flow problem）就是在 $N = (s, t, V, A, U)$ 中找到流值最大的可行流（即最大流）。我们将会看到，最大流问题的许多算法也可以用来求解流量给定的网络中的可行流。也就是说，当我们解决了最大流问题以后，对于在流量给定的网络中寻找可行流的问题，通常也就可以解决了。

因此，用线性规划的方法，最大流问题可以形式地描述如下：

$$\max v$$

$$\text{s.t.} \quad \sum_{j:(i,j)\in A} x_{ij} - \sum_{j:(j,i)\in A} x_{ji} = \begin{cases} v, & i = s \\ -v, & i = t \\ 0, & i \neq s, t \end{cases},$$

（4）

$$0 \le x_{ij} \le u_{ij}, \quad \forall (i, j) \in A. \qquad (5)$$

定义 如果一个矩阵 A 的任何子方阵的行列式的值都等于 0，1 或 -1，则称 A 是全幺模的(totally unimodular TU，又译为全单位模的)，或称 A 是全幺模矩阵。

定理 8（整流定理） 最大流问题所对应的约束矩阵是全幺模矩阵。若所有弧容量均为正整数，则问题的最优解为整数解。

最大流问题是一个特殊的线性规划问题。我们将会看到利用图的特点，解决这个问题的方法较之线性规划的一般方法要方便、直观得多。

7.1.3 单源和单汇运输网络

实际问题往往是多源多汇网络，为了计算的规格化，可将多源多汇网络G化成单源单汇网络G'。设X是G的源，Y是G的汇，具体转化方法如下：

（i）在原图G中增加两个新的顶点x和y，令其分别为新图G'中之单源和单汇，则G中所有顶点V成为G'之中间顶点集。

（ii）用一条容量为∞的弧把x连接到X中的每个顶点。

（iii）用一条容量为∞的弧把Y中的每个顶点连接到y。

G和G'中的流以一个简单的方式相互对应。若f是G中的流，则由

$$f'(a)=\begin{cases} f(a), & \text{若}a\text{是}G\text{的弧} \\ f^{+}(v)-f^{-}(v), & \text{若}a=(x,v) \\ f^{-}(v)-f^{+}(v), & \text{若}a=(v,y) \end{cases}$$

所定义的函数f'是G'中使得$v(f')=v(f)$的流。反之，G'中的流在G的弧集上的限制就是G中具有相同值的流。

7.2 最大流和最小割关系

设$N=(s,t,V,A,U)$，$S\subset V$，$s\in S$，$t\in V-S$，则称$(S,\bar{S})$为网络的一个割，其中$\bar{S}=V-S$，$(S,\bar{S})$为尾在S，头在$\bar{S}$的弧集，称

$$C(S,\bar{S})=\sum_{\substack{(i,j)\in A\\ i\in S,\,j\in\bar{S}}}u_{ij}$$

为割$(S,\bar{S})$的容量。

定理 9　f是最大流，$(S,\bar{S})$是容量最小的割的充要条件是

$v(f)=C(S,\bar{S})$。

在网络$N=(s,t,V,A,U)$中，对于轨$(s,v_2,\Lambda,v_{n-1},t)$（此轨为无向的），若$v_iv_{i+1}\in A$，则称它为前向弧；若$v_{i+1}v_i\in A$，则称它为后向弧。

在网络N中，从s到t的轨P上，若对所有的前向弧(i,j)都有$f_{ij}<u_{ij}$，对所有的后向弧(i,j)恒有$f_{ij}>0$，则称这条轨P为从s到t的关于f的可增广轨。

令

$$d_{ij}=\begin{cases} u_{ij}-f_{ij}, & \text{当（}i,j\text{）为前向弧} \\ f_{ij}, & \text{当（}i,j\text{）为后向弧} \end{cases},$$

$$d=\min\{d_{ij}\}$$

则在这条可增广轨上每条前向弧的流都可以增加一个量d，而相应的后向弧的流可减少d，这样就可使得网络的流量获得增加，同时可以使每条弧的流量不超过它的容量，而且保持为正，也不影响其它弧的流量。总之，网络中f可增广轨的存在是有意义的，因为这意味着f不是最大流。

7.3 最大流的一种算法—标号法

标号法是由 Ford 和 Fulkerson 在 1957 年提出的。用标号法寻求网络中最大流的基本思想是寻找可增广轨，使网络的流量得到增加，直到最大为止。