实验指导书： 图论模型

【实验目的】

1、把最短路径、最大流、最小生成树、旅行商、关键路径等图论问题转化为数学规划模型，并用软件进行求解。。

【实验相关知识】

图的基本知识

8.1.1 图的相关定义

图（Graph）是由一个表示对象的非空集合和一个表示这些对象之间关系的非空集合所构成的二元组，通常用大写字母 G,H 等表示。

定义 1：一个无向图(Undirected Graph) G 是由一个非空点集 $V(G)$ 和其中元素的无序关系集合 $E(G)$ 构成，记为 $G=(V(G),E(G))$,简记为 $G=(V,E)$ 。

$V=\{v_1,v_2,\cdots,v_n\}$ 称为无向图 G 的顶点集（vertex set）或节点集(node set)，每一个元素 $v_i(i=1,2,\cdots,n)$ 称为图 G 的一个顶点（vertex）或节点（node）；$E=\{e_1,e_2,\cdots,e_m\}$ 称为无向图 G 的边集(edge set)，每一个元素 $e_k(k=1,2,\cdots,m)$（即 V 中某两个元素 v_i,v_j 的无序对）记为 $e_k=(v_i,v_j)=(v_j,v_i)$ 或 $e_k=v_iv_j=v_jv_i$，称为无向图 G 的一条边（edge）。

当一条边可表示为 $e_k=v_iv_j$ 时，称 v_i,v_j 为边 e_k 的端点，并称 v_i 与 v_j（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顶点，则称这两条边在图 G 中相邻，没有公共顶点的边称为相互独立的（independent）。

定义 2：一个有向图（digraph）D 是由一个非空点集 $V(D)$ 和其中元素的有序关系集合 $A(D)$ 构成，记为 $D=(V(D),A(D))$，简记为 $D=(V,A)$ 。

$V=\{v_1,v_2,\cdots,v_n\}$ 称为有向图 D 的顶点集或节点集，V 中的每一个元素 $v_i(i=1,2,\cdots,n)$ 称为该图的一个顶点或节点；$A=\{a_1,a_2,\cdots,a_m\}$ 称为有向图 D 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 $a_k=v_iv_j(k=1,2,\cdots,n)$，被称为该有向图的一条从 v_i 到 v_j 的弧（arc）。

当一条弧可表示为 $a_k=v_iv_j$ 时，称 v_i 为 a_k 的头（head），v_j 为 a_k 的尾（tail），或者说 v_i 到 v_j 相邻，v_j 从 v_i 相邻；称弧 a_k 为 v_i 的出弧（outgoing arc），为 v_j 的入弧（incoming arc）。

图的点集中点的数量 $|V|$ 称作图的阶（order），边集中边的数量 $|E|$ 称作图的边数(size). 当一个图的顶点集和边集都是有限集，则称该图为有限图。

定义 3：给一个图的每一条边（弧）赋予一个数字，则得到一个**赋权图**（weighted graph）。这些数字可以表示距离，花费，时间等，统称为权重(weight)。无向图和有向图都可以赋权。

例 1．下面两个图 G,D,W 分别是无向图，有向图和赋权图。

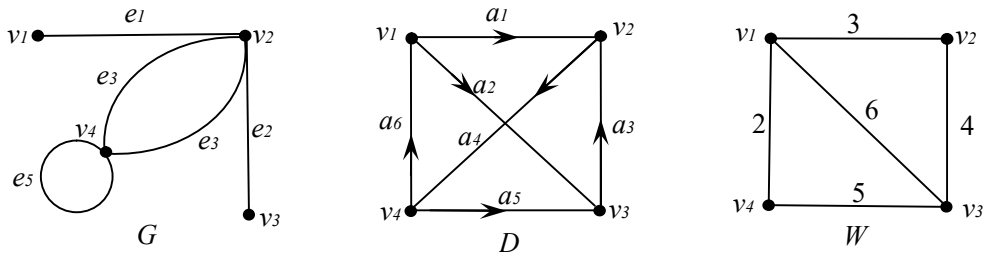


图 2 无向图、有向图和赋权图

只有一个顶点的图称为平凡图（trivial graph），除平凡图外所有图都被称为非平凡图。

如果图的一条边（或者弧）的端点为同一个点，则称这条边为环（loop）。

在无向图当中，如果两条边的端点相同，则称这两条边为关联这两点的重边（multi-edge）。有向图当中，两条弧如果端点和方向都相同才称为重边。

既没有重边也没有环的图被称为简单图（simple graph）。

8.1.2 图的顶点的度

定义 1：（1）在无向图中，与顶点 v 关联的边的数目（环算两次）称为 v 的**度**（degree），记为 $d(v)$.

度为奇数的点称为奇顶点，度为偶数的点称为偶顶点。度为 0 的点称为**孤立点**（isolated vertex），它不与任何点相邻。度为 1 的点称为叶子点（leaf）。

所有顶点度都为 k $(k=1,2,\cdots|V|-1)$ 的简单图被称为 k 阶正则图（k-regular graph）。特别地，若图的各顶点的度均等于 $|V|-1$，即任意一顶点都和其他顶点相邻，则称此图为完全图（complete graph），具有 n 个顶点的完全图记为 K_n 。

(2)在有向图中，从顶点 v 引出的边的数目称为 v 的**出度**（out degree），记为 $d^+(v)$；从顶点 v 引入的边的数目称为的**入度**(in degree)，记为 $d^-(v)$.

显然 $d^+(v)=d^-(v)$，所以 v 的度应为 $d(v)=d^+(v)+d^-(v)$ 。

若对于有向图中任意两点 v_i,v_j 之间有且仅有一条有向边，则称该图为严格有向图。

定理 1：（图论第一定理）图的顶点的度的总和等于边数的两倍。

$$\sum_{v\in V(G)} d(v)=2|E(G)|$$

显然所有顶点的度的总和为一个偶数，也就是说，在计算一个图的总度数的时候，所有的边都被计算了两次。

推论 1：在有向图中

$$\sum_{v\in V(D)} d(v)=\sum_{v\in V(D)} d^+(v)+d^-(v)=2|E(G)|\text{。}$$

推论 2：任何图中奇顶点的总数必为偶数.

例 2. 在一次聚会中，认识奇数个人的人数一定是偶数。

认识是指人与人之间相互认识，即 a 若认识 b，则 b 也认识 a。可以把聚会中的所有人看作点，把认识关系看作边，从而构造出一个无向图。每个人认识其他的人的数量要么为奇数，要么为偶数，可以分别看作该顶点的度。根据定理 1，所有认识关系的总和是一个偶数。

因为无论奇数个偶数还是偶数个偶数之和均为偶数，因此要满足总和为偶数，必然应该有偶数个奇数。

8.1.3 子图及运算

1．子图

定义 1： 设图$G=(V,E)$，$H=(V_1,E_1)$

(1) 若$V_1\subseteq V$，$E_1\subseteq E$，则称H是G的**子图**(subgraph). 特别地，若$V_1=V$，$E_1\subseteq E$，则称H为G的**生成子图**（spanning subgraph）.

(2) 设$V_1\subseteq V$，且$V_1\neq\Phi$，以V_1为顶点集、两个端点都在V_1中的图G的边为边集的图G的子图，称为G的**由 V_1 导出的子图**（induced subgraph），记为$G[V_1]$。

(3) 设$E_1\subseteq E$，且$E_1\neq\Phi$，以E_1为边集,E_1的端点集为顶点集的图 G 的子图,称为 G 的**由 E_1 导出的子图**,记为$G[E_1]$。

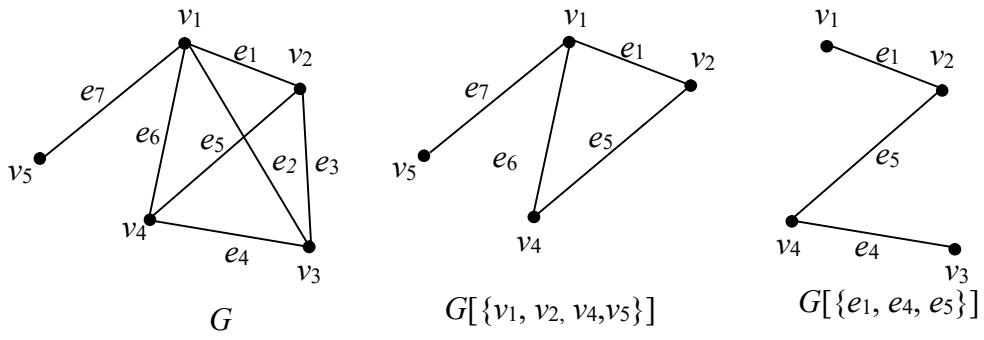


图 3 生成子图与导出子图

2．图的运算

图的运算类似于集合运算，由于图是二元集，所以较之集合运算又有一些不同。

定义 2： $G=(V,E)$，$G_1=(V_1,E_1)$，$G_2=(V_2,E_2)$为三个图

（1）两个图G_1与G_2的**和**$G_1+G_2=(V_1\bigcup V_2,E_1\bigcup E_2)$。

（2）$v\in V,e\in E$，$G-\{v\}$表示从图G中去掉点v，同时去掉和v相关联的所有的边；$G-\{e\}$表示仅去掉边e而不去掉任何的顶点。可推广到多个点和多条边的情形。

（3）两个图G_1与G_2的**差**$G_1-G_2=(V_1-V_2,E_1-E_2)$。

（4）图G具有n个顶点，其**补图（complementary）**可表示为$\overline{G}=K_n-E$,即图G的补图和它自身具有相同的顶点集，

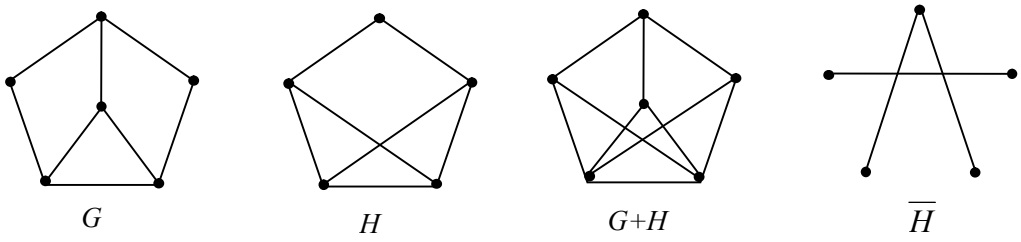


图 4 图的加法与补运算

定义 3： 两个图$G_1=(V_1,E_1)$，$G_2=(V_2,E_2)$的笛卡儿积（Cartesian product）表示为$C=G_1\times G_2$，其点集$V(C)=\{(u_i,v_j)(i=1,2,\cdots|V_1|,j=1,2,\cdots|V_2|\}$，其边集$E(C)$由如下规则确定：若两点的第一个坐标相同，第二个坐标表示的点在某个图中相邻，或者第二个坐标相同，第一个坐标表示的点在某个图中相邻，则两点在C中相邻，所有满足此规则的相邻关系构成$E(C)$。

可以看出，图的笛卡儿积仍然是一个图，其顶点的表示方法类似于平面坐标系下点的表示方法，它的顶点数量等于两个作为因子的图的顶点数量乘积，而点的相邻关系的判别规则尤其需要注意。

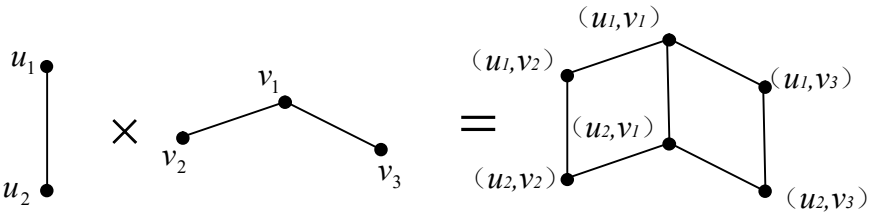


图 5 笛卡儿积

8.1.4 图的连通性

定义 1：在一个图$G=(V,E)$中，若存在一个以v_0为起点，以v_k为终点的点与边交替的序列：$v_0e_1v_1e_2v_2e_3\cdots v_k\,(k=1,2,\cdots|V|)$，则称这个序列为一条从$v_0$到$v_k$通路（walk）。在简单图中可以仅用点来表示这个序列。

图 6 中$v_1e_1v_2e_2v_3e_{10}v_7e_{11}v_6e_{11}v_7e_9v_2$是一条$v_1$到$v_2$的通路，这样的通路还有很多。只要从起点到终点的过程中经过的所有点与点，边与边顺次相邻，就构成一条通路。可见通路中允许存在重复的点和重复的边。

定义 2：若在起点v_0和终点v_k之间存在一条不经过重复的边的通路，则称其为一条v_0-v_k迹（trail）。起点和终点相同的迹称为回路（circuit）。

图 6 中$v_1e_8v_7e_4v_4e_3v_3e_{10}v_7e_9v_2$是众多$v_1-v_2$迹中的一条。可见在迹中允许重复经过点，但是不存在重复的边。

定义 3：若在起点v_0和终点v_k之间存在一条不经过重复的点的通路，则称其为一条v_0-v_k路径（path）。具有n个顶点的路径记为P_n。 起点和终点相同的路径称为圈（cycle）。圈的顶点数和边数相同，具有n个顶点的圈记为C_n， n为奇（偶）数叫做奇（偶）圈。

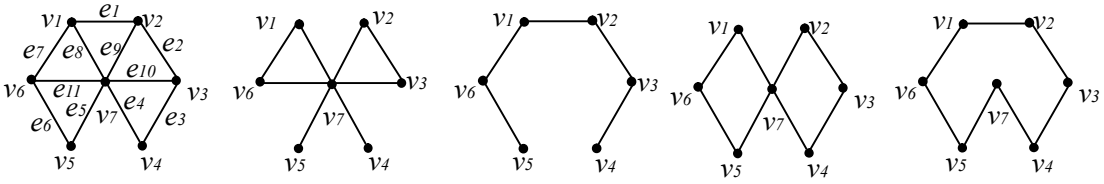


图 6　图的迹、路径、回路、圈

定义 4：在图$G=(V,E)$中，两点v_i与v_j（$v_i,v_j\in V$）之间若存在至少一条以其中一点为起点，另外一点为终点的路径，则称这两点是连通（connected）的。否则称这两点非连通（disconnected）。更一般地，若图中任意两点都是连通的，则称该图是连通图，否则就称为非连通图。非连通图总是由一些连通的分支（component）所组成。

定义 5：图$G=(V,E)$是连通图，若$G-v(v\in V)$为非连通图，则称点v为图G的割点（cut vertex）。

定义 6：图$G=(V,E)$是连通图，若$G-e(e\in E)$为非连通图，则称边e为图G的桥（bridge）。

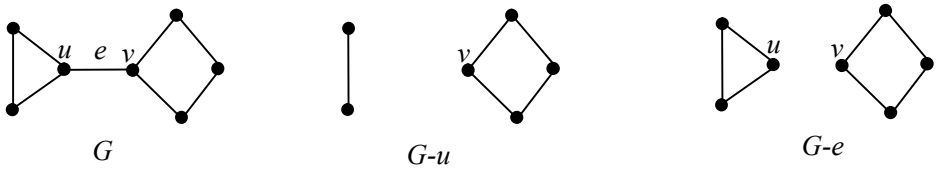


图 7 割点与桥

注意割点集并不全由割点构成，一个非完全连通图可以不存在割点，但总存在割点集。割边集同样如此，一个非平凡图不一定存在桥，但一定存在割边集。

图的连通性（connectivity）由连通度来刻画，可以分为点连通度和边连通度两个方面来度量，这两方面是不一样的。

定义 7：图$G=(V,E)$是连通图，若$G-V_1(V_1\subset V)$为非连通图，则称V_1为图G的割点集（vertex cut）。使得图$G-V_1$非连通且包含点数最少的点集V_1叫做图G的最小割点集（minimum vertex cut）。图的点连通度（记为$\kappa(G)$）等于最小割点集的势（cardinality）。

定义 8：图$G=(V,E)$是连通图，若$G-E_1(E_1\subset E)$为非连通图，则称E_1为图G的割边集（edge cut）。使得图$G-E_1$非连通且包含边数最少的边集E_1叫做图G的最小割点集（minimum edge cut）。图的边连通度（记为$\lambda(G)$）等于最小割边集的势。

显然一个图连通若存在割点，它的点连通度为1。同样，如果一个连通图包含桥，它的边连通度为1。考察一个图的连通性直观上可以通过观察它是否存在割点和桥来衡量。图的最小度（记为$\delta(G)$）、点连通度和边连通度之间满足：

$$0\le\kappa(G)\le\lambda(G)\le\delta(G)$$

当且仅当图G为非连通图时，点连通度和边连通度为0。

下面这个图可以很好的说明上面的不等式。

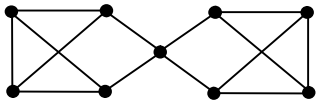


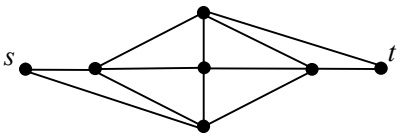
图 8 图 G 中$\kappa(G)=1,\lambda(G)=2,\delta(G)=3$

定义 9：有向图$D=(V,A)$中，$v_i,v_j\in V$。若既存在v_i到v_j的有向路径，也存在v_j到v_i的有向路径，则称此有向图为强连通的（strong）。

定义 10：图$G=(V,E)$中任意两点v_i与v_j的之间最短路径的长度叫做两点之间的距离，记为$d(v_i,v_j)$或者d_{ij}。两点之间若不存在路径（对有向图是指不存在方向一致的路径），则距离为∞。例如两个顶点分别位于非连通图的两个分支。

一般情况下，两点若相邻，则两点之间的距离为1，即一条边的长度为1；对于赋权图，两点相邻，则距离由权重来衡量。

思考一下右图中s与t之间的距离为多少？



8.1.5 一些特殊图

定义 10：$G=(V,E)$，若$V=X\bigcup Y$，$X\bigcap Y=\Phi$，同一点集X或Y中任意两个顶点都不相邻，即X中的点只能和Y中的点相邻，Y中的点也只和X中的点相邻，称G为二部图（bipartite graph）；若 X 中每一顶点皆与 Y 中一切顶点相邻，称为完全二部图，记为$K_{m,n}$，其中m,n分别为X与Y的势.

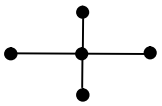
任何路径均为二部图，所有长度为偶数的圈也是二部图（为什么长度为奇数的不是？）。有时候某些图并不能直观的看出它们是二部图，但是可以保持相邻关系不变的前提下改变形状使其一目了然。

图 9 二部图

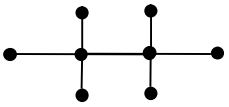
定理 2：一个图是二部图当且仅当它不存在奇圈。

定义 11：一个图中若存在一个到其余顶点距离都为 1 的顶点，则此图称为星型图（star graph）。

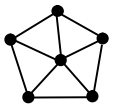
此外还有车轮图（wheel graph），双星（double star）图等等。



星



双星



车轮

8.2 图的矩阵表示

图的表示方式除了直观的点与边的表示之外，为了借助计算机技术来解决更复杂的图的问题而通常采用的方法是矩阵方式。

8.2.1 邻接矩阵

邻接矩阵（adjacency matrix）是由图中点与点之间的相邻关系的一种矩阵表示形式。

对无向图 G，其邻接矩阵为一个方阵 $A=(a_{ij})_{|V|\times|V|}$，其行列数均等于图的顶点数。其每个元素按如下规则定义：

$$a_{ij}=\begin{cases}1, & \text{若}v_i\text{与}v_j\text{相邻;}\\ 0, & \text{若}v_i\text{与}v_j\text{不相邻.}\end{cases}$$

也就是说，如果两顶点 v_i 与 v_j 之间有一条边相关联，则邻接矩阵中对应的元素为 1；否则为 0。

对赋权无向图 W，其邻接矩阵 $A=(a_{ij})_{|V|\times|V|}$，其中：

$$a_{ij}=\begin{cases}w_{ij} & \text{若}v_i\text{与}v_j\text{相邻且}w_{ij}\text{为其权；}\\ 0 & \text{若}i=j\text{；}\\ \infty & \text{若}v_i\text{与}v_j\text{不相邻。}\end{cases}$$

例 1. 右图所示的图可用邻接矩阵和赋权邻接矩阵分别表示为：

$$A=\begin{array}{c}\begin{matrix}v_1 & v_2 & v_3 & v_4 & v_5\end{matrix}\\ \begin{pmatrix}0&1&0&0&1\\1&0&1&0&1\\0&1&0&1&0\\0&0&1&0&1\\1&1&0&1&0\end{pmatrix}\end{array}\begin{matrix}v_1\\v_2\\v_3\\v_4\\v_5\end{matrix}\qquad \begin{array}{c}\begin{matrix}v_1 & v_2 & v_3 & v_4 & v_5\end{matrix}\\ \begin{pmatrix}0&3&\infty&\infty&4\\3&0&6&\infty&2\\\infty&6&0&2&\infty\\\infty&\infty&2&0&5\\4&2&\infty&5&0\end{pmatrix}\end{array}\begin{matrix}v_1\\v_2\\v_3\\v_4\\v_5\end{matrix}$$

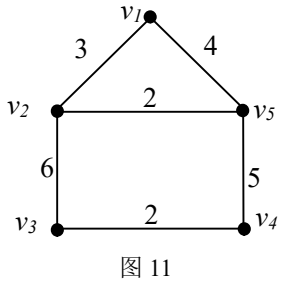


图 11

可见无向图的邻接矩阵为一个对角线全为 0 的 0-1 对称阵。

邻接矩阵的幂运算 $A^{(k)}$ $(k=1,2,\cdots)$ 中任意位置上的元素 $a_{ij}{}^{(k)}$（注意这里只是一种形式化的表示，其值由矩阵幂运算确定）表示点 i 到点 j 长度为 k 的通路的数量。例如：

$$A^3=\begin{pmatrix}2&4&2&2&4\\4&2&5&1&6\\2&5&0&4&1\\2&1&4&0&5\\4&6&1&5&2\end{pmatrix}$$

$A^3(1,5)=4$ 表示从 v_1 到 v_5 长度为 3 的通路总共有 4 条，分别为：

$$v_1v_2v_1v_5,\ \ v_1v_5v_1v_5,\ \ v_1v_5v_4v_5,\ \ v_1v_5v_2v_5$$

对有向图 D，其邻接矩阵 $A=(a_{ij})_{|V|\times|V|}$，其中：

$$a_{ij}=\begin{cases}1, & \text{若}(v_i,v_j)\in D;\\ 0, & \text{否则。}\end{cases}$$

也就是说，如果两顶点 v_i 与 v_j 之间有一条以 v_i 为头，以 v_j 为尾的有向边，则邻接矩阵中对应的元素为 1；否则为 0。

对赋权有向图 W，其邻接矩阵 $A=(a_{ij})_{|V|\times|V|}$，其中：

$$a_{ij}=\begin{cases}w_{ij} & \text{若}(v_i,v_j)\in D\text{且}w_{ij}\text{为其权;}\\ 0 & \text{若}i=j;\\ \infty & \text{若}(v_i,v_j)\notin D。\end{cases}$$

例 2. 右图所示的有向图用邻接矩阵和赋权邻接矩阵分别表示为：

$$\begin{array}{c} \begin{matrix} v_1 & v_2 & v_3 & v_4 & v_5 \end{matrix} \\ \begin{pmatrix} 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \end{pmatrix} \begin{matrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{matrix} \end{array} \qquad \begin{array}{c} \begin{matrix} v_1 & v_2 & v_3 & v_4 & v_5 \end{matrix} \\ \begin{pmatrix} 0 & \infty & 0 & 0 & 4 \\ 3 & 0 & \infty & 0 & \infty \\ 0 & 6 & 0 & 2 & 0 \\ 0 & 0 & \infty & 0 & 5 \\ \infty & 2 & 0 & \infty & 0 \end{pmatrix} \begin{matrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{matrix} \end{array}$$

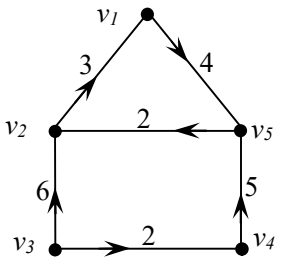


图 12

8.2.2 关联矩阵

关联矩阵（incidence matrix）是由图中点与边之间的关联关系的一种矩阵表示形式。关联矩阵的行数等于图的点数 $|V|$，列数等于边数 $|E|$。因此与邻接矩阵不同，关联矩阵未必是方阵。

对无向图，其关联矩阵 M＝$M=(m_{ij})_{|V|\times|E|}$，其中：

$$m_{ij}=\begin{cases}1 & \text{若}v_i\text{与}e_j\text{相关联;}\\ 0 & \text{若}v_i\text{与}e_j\text{不关联。}\end{cases}$$

例 3. 右图的关联矩阵为：

$$\begin{array}{c} \begin{matrix} e_1 & e_2 & e_3 & e_4 & e_5 & e_6 \end{matrix} \\ \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 \end{pmatrix} \begin{matrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{matrix} \end{array}$$

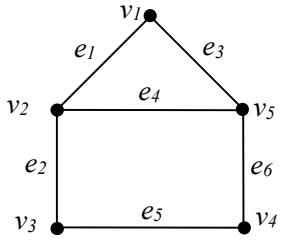


图 13

对有向图，其关联矩阵 M＝$(m_{ij})_{|V|\times|D|}$，其中：

$$m_{ij}=\begin{cases}1 & \text{若}v_i\text{是}e_j\text{的头;}\\ -1 & \text{若}v_i\text{是}e_j\text{的尾;}\\ 0 & \text{若}v_i\text{与}e_j\text{不关联。}\end{cases}$$

例 4. 右图的关联矩阵为：

$$\begin{array}{c} \begin{matrix} a_1 & a_2 & a_3 & a_4 & a_5 & a_6 \end{matrix} \\ \begin{pmatrix} -1 & 0 & 1 & 0 & 0 & 0 \\ 1 & -1 & 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 \\ 0 & 0 & -1 & 1 & 0 & -1 \end{pmatrix} \begin{matrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{matrix} \end{array}$$

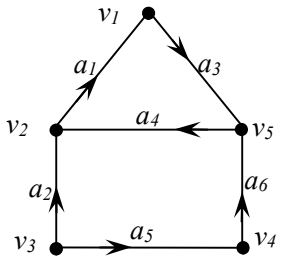


图 14

邻接矩阵和关联矩阵在表示图的时候直观简洁，但是我们不难看到，在这些矩阵中存在大量的 0。以邻接矩阵为例，在 $|V|^2$ 个元素中，仅有 $2|E|$ 个元素非 0，因此会浪费大量的存储空间。

除了常用邻接矩阵和关联矩阵表示图之外，还有一些方法可以用来表示图，例如弧表表示法、邻接表表示法等，其表示的效率甚至更高。

8.3 图的方法建模

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（如图）。假定已经知道了任意两个城市之间修建高速公路的成本（见表），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？

显然如果用图论的语言来描述这个问题就是：找到图 1 的一个生成子图，使得该生成子图是连通的，并且任意两点之间仅存在唯一路径。

这个问题可以用图论中关于最小生成树的理论来解决。最小生成树问题是图论中最基本的理论之一。

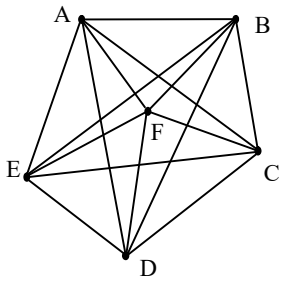


图 1. 城市交通图

E	51				
D	54	38			
C	62	106	60		
B	59	130	112	50	
A	48	40	97	100	45
	F	E	D	C	B

表 1. 修路费用表（单位：千万元）

8.3.1 图的最小生成树问题及算法

1．树及最小生成树

树是图论中一个非常重要的概念，什么叫做树呢？它与我们日常生活中简单的树有哪些不同？

定义 1：连通的无圈图叫做树（tree），记之为 T 。树中度为 1 的点称为叶子节点。

右图是具有 5 个顶点的不同的树。可以看出树的一般特征：

1）树的顶点数比边数多 1；

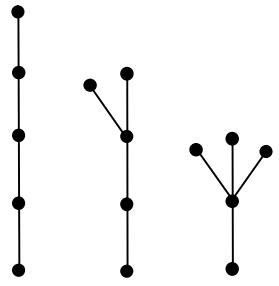


图 2 树

2）任意两点之间仅存在唯一路径；

3）除了叶子节点，每个点都是割点；

4）每条边都是桥。

树的判别可运用下面的定理。

定理 1: 一个图$G=(V,E)$若满足下列任意两个条件：

1） 连通；

2） 点数=边数+1；

3） 不存在任何的圈。

则称图G为树。

再进一步考察树的结构可以发现，去掉树的任意一条边或者去掉任意一个非叶子节点，树都会变的不连通。而任意添加一条边都会使得树出现圈。

定理 2: 树是具有最小连通性（minimum connected），同时也是具有最大无圈性(maximum acyclic)的连通图。

定义 2：若图$G=(V,E)$及树T之间满足$V(G)=V(T)$，$E(T)\subset E(G)$，则称T是G的生成树（spanning tree）。一个连通图的生成树的个数很多，用$\tau(G)$表示G的生成树的个数。有如下公式计算一个n阶完全图的生成树的数目：

$$\tau(K_n)=n^{n-2} \qquad (1)$$

这个公式被称为 Caylay 公式。因此在引言的例子当中，如果不考虑修路的成本，总共有6^4=1296 种不同的方案。

对于一般连通图，其生成树的数目的判别还没有一个一般性的方法。

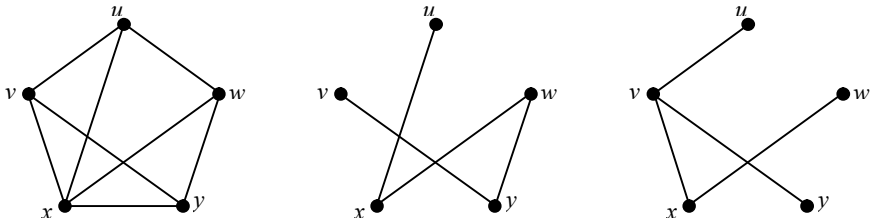


图 3 连通图及其生成树

定义 3：在一个赋权图中，则所有边的权重之和最小的生成树称为该图的最小生成树（minimum spanning tree）。找出赋权图的最小生成树的问题称为最小生成树问题。

例如在开头所举的例子当中，将所有城市连接起来的总费用最小的修路方案如下图，其最小费用为 W_{AE} + W_{AF} + W_{AB} + W_{ED} + W_{BC} =40+48+45+50+38=221。

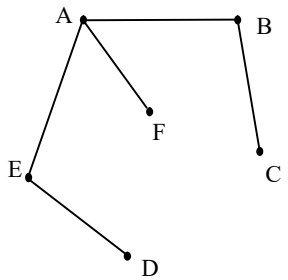


图 4 最小费用树

最小生成树问题的现实意义不言而喻。例如上面的例子当中，找到最小生成树也就找到了费用最小的筑路方案。除此之外，最小生成树问题还在电路设计，运输网络等方面有很好的运用。

解决最小生成树问题的主要方法有克鲁斯卡尔算法（Kruskal’s Algorithm）和普利姆(Prim’s Algoritm)算法。

2．克鲁斯卡尔算法

克鲁斯卡尔是近代著名的数学和语言学家。他一生积极从事研究工作，曾在贝尔实验室工作多年。

克鲁斯卡尔算法描述如下：

对于一个连通的赋权图 G，按照如下步骤构造其最小生成树 T：

1） 找出所有 G 中的权重最小的边 e_1 作为 T 的第一条边；

2） 在余下的 G 的边当中找出权重最小的边 e_2 作为 T 的第二条边；

3） 在前面选择剩余的 G 的边中找出权重最小的边 e_3 作为 T 的第三条边，且能和前面所选的边构成圈；

4）重复步骤 3）（边的下标顺序增加），直到找出 $n-1$ 条边，则得到 G 的最小生成树。

例 1．用克鲁斯卡尔算法求下图的最小生成树。

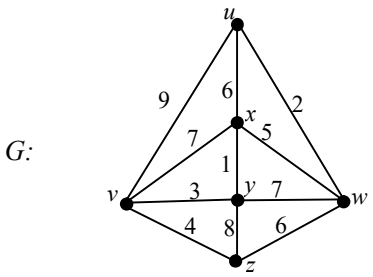


图 5

解：由于最小生成树必然包含所有顶点，因此将图 G 的顶点集作为最小生成树的点集。只需要按算法步骤找出符合条件的边即可。该最小生成树有 6 个点，5 条边。

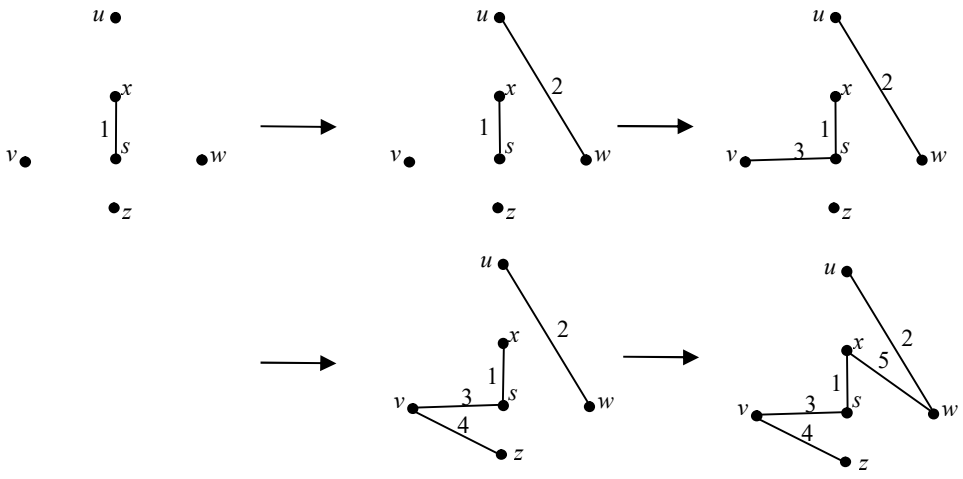


图 6 克鲁斯卡尔算法求最小生成树过程

该最小生成树的权重之和 $w(T)=\sum_{i=1}^{5} w(e_i)=15$。

可见，一条边 uv 在某一步被选中，在以后的选取过程中它将不会被再次选到，即可将这条边压缩为一点（两顶点重合）。

MATLAB 程序实现：

```
clear all;
clc;
```

```
%输入图的带权邻接矩阵，按字母顺序分别将 u ,v ,w, x, y, z 替换为 1,2,3,4,5,6 作为矩阵元素%
下标
A=zeros(6);
A(1,2)=9; A(1,3)=2; A(1,4)=6;
A(2,4)=7; A(2,5)=3; A(2,6)=4;
A(3,4)=5; A(3,5)=7; A(3,6)=6;
A(4,5)=1;
A(5,6)=8;
%找出权重非 0 的边并将其下标进行记录;
[i,j]=find(A~=0);
%将权非 0 的重赋值给行向量 B;
B=A(find((A~=0)));
%将非 0 权重的边的行列信息以及权重构造矩阵 Data;
Data=[i';j';B'];
List=Data(1:2,:);
%生成树的边树等于顶点数减 1;
Branch=max(size(A))-1;
%算法核心部分:
Solve=[];
while length(Solve)<Branch   %若生成树边数少于 n-1，则按升序依次取权重最小的边；
    temp=min(Data(3,:));
    flag=find(Data(3,:)==temp);
    flag=flag(1);
    v1=Data(1,flag);
    v2=Data(2,flag);
        if List(1,flag)~=List(2,flag)
            Solve=[Solve,Data(:,flag)];
        end
    %将已选的边压缩为一个顶点，且用标号较小的顶点来代替；
        if v1>v2
            List(find(List==v1))=v2;
        else
            List(find(List==v2))=v1;
        end
    Data(:,flag)=[];
    List(:,flag)=[];
end
Solve
Wt=sum(Solve(3,:))
```

程序输出:

```
Solve =
     4     1     2     2     3
     5     3     5     6     4
     1     2     3     4     5
Wt =
    15
```

第一,二行分别表示选中的边的顶点,第三行表示对应的权重，最小权重之和为 15。

3．普利姆算法

普利姆与克鲁斯卡尔是普林斯顿大学的校友，并于 1949 年取得博士学位。

普利姆算法的思想是，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。

普利姆算法描述如下：

对于连通的赋权图$G=(V,E)$，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。

1）初始化顶点集$P=\{v_1\}$，$v_1\in V$，边集$Q=\Phi$；

2）选中G的与顶点v_1相邻的距离最近的点v_2，$P=\{v_1,v_2\}$，$Q=\{v_1v_2\}$；

3）重复步骤2）直到$P=V$，停止；

4）$T=(P,Q)$即为所求的最小生成树。

例2. 用普利姆算法求例1中图G的最小生成树。

解：用普利姆算法构造最小生成树可以从任一点开始，不妨从u点开始，用W表示权重之和，初始化$P=\{u\}$，$Q=\Phi$，$W=0$。

1）找出与u点距离最近的相邻点w，$P=\{u,w\},Q=\{uw\},W=2$；

2）将uw看作一点（即u与w重合），再取与u或w距离最近的点x，注意此时是把与u相邻和与w相邻的点综合在一起考虑，最近距离也是与x相邻点的距离和与w相邻点的距离当中的最小值。$P=\{u,w,x\}$, $Q=\{uw,wx\},W=7$；

3）重复上述步骤，可以得到：

$P=\{u,w,x,y,v,z\},Q=\{uw,wx,xy,yv,vz\},W=15$。

即最小生成树$T=(P,Q)$。

采用普利姆算法如果选择的初始点不同，最小生成树的构造过程也是不一样的。

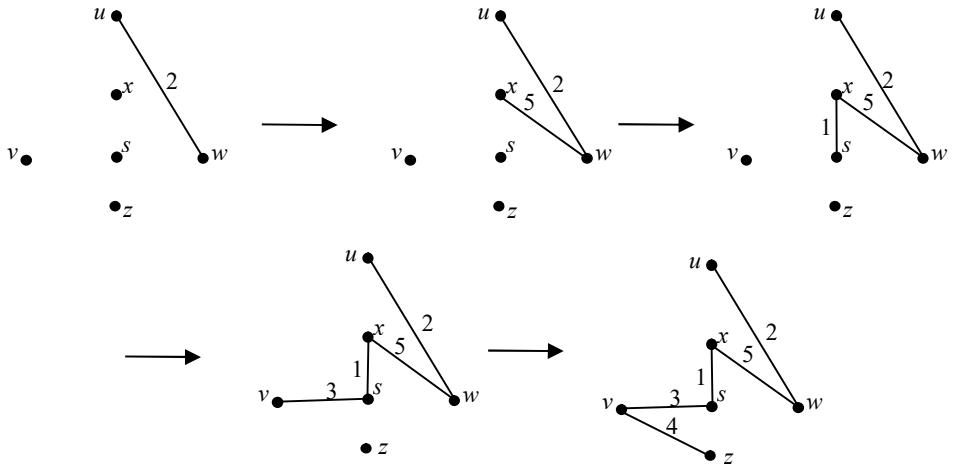


图7 普利姆算法求最小生成树过程

MATLAB程序实现：

```
clear all;
clc;
A=zeros(6);
A(1,2)=9; A(1,3)=2; A(1,4)=6;
A(2,4)=7; A(2,5)=3; A(2,6)=4;
A(3,4)=5; A(3,5)=7; A(3,6)=6;
A(4,5)=1;
A(5,6)=8;
A=A+A'; %A 为对称阵
%将带权邻接矩阵中为 0 的值用一个大于所有权重的数值代替;
A(find(A==0))=100;
%初始化，k 为计数器；
T=[];
```

```
row=1;k=0;
column=2:length(A);
%从第一行开始，分别找每一列的最小权重；用 temp 储存；
while length(T)~=length(A)-1
    k=k+1;
    temp=A(row,column);
    temp=temp(:);
    weight(k)=min(temp);
    [ir,jc]=find(A(row,column)==weight(k));
    i=row(ir(1));j=column(jc(1));
    T=[T,[i;j;weight(k)]];row=[row,j];column(find(column==j))=[];
end
T
W=sum(weight)
```

程序输出:

```
T =
     1     3     4     5     2
     3     4     5     2     6
     2     5     1     3     4
W =
    15
```

比较上述两种算法可以看出他们之间虽然构造最小生成树的过程是不同的，但是结果却是一致的，可以说明两种方法都是有效的构造最小生成树的方法。

8.3.2 图的最短路问题及算法

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。

将公路网络用一个图G来描述，显然甲乙两地应该是其中两个顶点，连接不同城市的公路以及它长度就构成了带权边。因此G为赋权图，而求从甲到乙的最短路线问题就转换为求赋权图中指定的两个顶点u_0, v_0间的具最小权的路径的问题。这条路径叫做u_0, v_0间的最短路，它的权叫做u_0, v_0间的距离，记作$d(u_0, v_0)$。

根据不同情况，又可以将最短路问题分为：固定起点（或终点）分别到其他顶点的最段路问题、固定起点和终点的最短路问题等；

求最短路径的有很多成熟的算法，例如迪克斯特拉（Dijkstra）算法、Floyd 算法等。

1．迪克斯特拉算法

迪克斯特拉算法的基本思想是：按从近到远的顺序依次求得u_0到G的其余各顶点的最短路和距离，最终到达v_0时算法结束。主要采用标号修正算法（Label-Correcting Algorithm）对每一步运算进行记录，可避免重复并且能保留每一步的计算信息。

迪克斯特拉算法描述为：

用$l(v)$表示u_0到v点的距离，S表示路径顺次经过的点集；

(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 $v\in\overline{S}_i$（$\overline{S}_i=V-S_i$），用 $\min\limits_{u\in S_i}\{l(v),l(u)+w(uv)\}$ 代替 $l(v)$。计算 $\min\limits_{v\in\overline{S}_i}\{l(v)\}$，把达到这个最小值的一个顶点记为 u_{i+1}，令 $S_{i+1}=S_i\bigcup\{u_{i+1}\}$。

(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 u_i 的距离由 v 的最后一次的标号 $l(u_i)$ 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 标号时途经的边在图上标明，在算法结束时 u_0 到各顶点的最短路就在图上标示出来了。

例 3. 求下图中 u_1 到各顶点 u_i 的最小距离 $d(u_1,u_i)$ 以及最短路 $l(u_i)$。

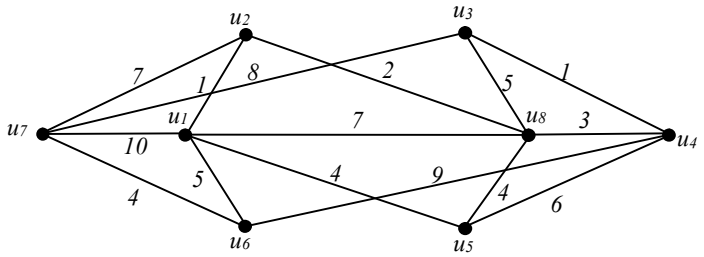


图 8

解：（1）$S_0=\{u_1\},\overline{S_0}=\{u_2,u_3,u_4,u_5,u_6,u_7,u_8\}$,因为 u_1 仅与 u_2，u_5，u_6，u_7，u_8 相邻，与 u_3，u_4 不相邻,取最小权重（不相邻用 ∞ 表示）：

$$\min\{1,\infty,\infty,4,5,10,7\}=1$$

$$d(u_1,u_2)=1,\quad l(u_2)=(u_1,u_2);$$

（2）$S_1=\{u_1,u_2\},\overline{S_1}=\{u_3,u_4,u_5,u_6,u_7,u_8\}$，两个距离集合分别表示 u_1 到除 u_2 其余点的距离和 u_1 经过 $l(u_2)$ 到其余点的距离。

$$\min\{\{\infty,\infty,4,5,10,7\}\bigcup\{1+\infty,1+\infty,1+\infty,1+\infty,1+7,1+2\}\}=3,$$

$$d(u_1,u_8)=3,\quad l(u_8)=l(u_2)+\{u_8\}=(u_1,u_2,u_8)$$

（3）$S_2=\{u_1,u_2,u_8\},\overline{S_2}=\{u_3,u_4,u_5,u_6,u_7\}$

$$\min\{\{\infty,\infty,4,5,10\}\bigcup\{1+\infty,1+\infty,1+\infty,1+\infty,$$

$$1+7\}\bigcup\{3+5,3+3,3+4,3+\infty,3+\infty\}\}=4;$$

$$d(u_1,u_5)=4,\quad l(u_5)=(u_1,u_5);$$

(4) $S_3=\{u_1,u_2,u_5,u_8\},\overline{S_3}=\{u_3,u_4,u_6,u_7\}$

$$\min\{\{\infty,\infty,5,10\}\bigcup\{1+\infty,1+\infty,1+\infty,$$

$$1+7\}\bigcup\{4+\infty,4+6,4+\infty,4+\infty\}$$

$$\bigcup\{3+5,3+3,3+4,3+\infty\}\}=5$$

$$d(u_1,u_6)=5,\quad l(u_6)=(u_1,u_6)$$

(5) $S_4=\{u_1,u_2,u_5,u_6,u_8\},\overline{S_4}=\{u_3,u_4,u_7\}$

$$\min\{\{\infty,\infty,10\}\cup\{1+\infty,1+\infty,1+7\}$$
$$\cup\{4+\infty,4+6,4+\infty\}\cup\{5+\infty,5+9,5+4\}$$
$$\cup\{3+5,3+3,3+\infty\}\}=6$$

$$d(u_1,u_4)=6,\quad l(u_4)=l(u_8)+\{u_4\}=(u_1,u_2,u_8,u_4)$$

(6) $S_5=\{u_1,u_2,u_4,u_5,u_6,u_8\},\overline{S_5}=\{u_3,u_7\}$

$$\min\{\{\infty,10\}\cup\{1+\infty,1+7\}\cup\{6+1,6+\infty\}$$
$$\cup\{4+\infty,4+\infty\}\cup\{5+\infty,5+4\}\cup\{3+5,3+\infty\}\}=7$$

$$d(u_1,u_3)=7,\quad l(u_3)=l(u_4)+\{u_3\}=(u_1,u_2,u_8,u_4,u_3)$$

(7) $S_6=\{u_1,u_2,u_3,u_4,u_5,u_6,u_8\},\overline{S_6}=\{u_7\}$

$$\min\{\{10\}\cup\{1+7\}\cup\{7+8\}\cup\{6+\infty\}$$
$$\cup\{4+\infty\}\cup\{5+4\}\cup\{3+\infty\}\}=8$$

$$d(u_1,u_7)=8,\quad l(u_7)=l(u_6)+\{u_7\}=(u_1,u_6,u_7)$$

故从 u_0 到其余各点的最短距离分别为：$l(u_2)=(u_1,u_2)$，$l(u_3)=(u_1,u_2,u_8,u_4,u_3)$，$l(u_4)=(u_1,u_2,u_8,u_4)$，$l(u_5)=(u_1,u_5)$，$l(u_6)=(u_1,u_6)$，$l(u_7)=(u_1,u_6,u_7)$，$l(u_8)=(u_1,u_2,u_8)$。

根据该算法思想可以写出 MATLAB 程序：

```
clc;clear all;close all;
%输入带权邻接矩阵，Inf 表示不相邻，有时也用一个足够大的正数来替代。
A =[ 0 1 Inf Inf 4 5 10 7;
     0 0 Inf Inf Inf Inf 7 2;
     0 0 0 1 Inf Inf 8 5;
     0 0 0 0 6 9 Inf 3;
     0 0 0 0 0 Inf Inf 4;
     0 0 0 0 0 0 4 Inf;
     0 0 0 0 0 0 0 Inf
     0 0 0 0 0 0 0 0];
W=A+A';
%D 的第一行是最短路径的值，第二行是每次求出的最短路径的终点,S 用于存放路径终点和长度
D=[];T=[];S=[];
%初始化
i=1;m=length(W);
S(1,1)=i;V=1:m;V(i)=[];D=[0;i];
```

```
n=2;[mD,nD]=size(D);
while ~isempty(V) %程序中的 ti 为临时变量
    [td,j]=min(W(i,V));
    tj=V(j);
    for k=2:nD
        [t1,jj]=min(D(1,k)+W(D(2,k),V));
        t2=V(jj);T(k-1,:)=[t1,t2,jj];
    end
        t=[td,tj,j;T];[t3,t4]=min(t(:,1));
    if t3==td
         S(1:2,n)=[i;t(t4,2)];
    else
         t5=find(S(:,t4)~=0);
         t6=length(t5);
        if   D(2,t4)==S(t6,t4)
            S(1:t6+1,n)=[S(t5,t4);t(t4,2)];
         else
            S(1:3,n)=[i;D(2,t4);t(t4,2)];
        end;
    end
    D=[D,[t3;t(t4,2)]];V(t(t4,3))=[];
    [mD,nD]=size(D);n=n+1;
end;
S,D
```

程序输出结果为：

S =

1	1	1	1	1	1	1	1
0	2	2	5	6	2	2	2
0	0	8	0	0	8	8	7
0	0	0	0	0	4	4	0
0	0	0	0	0	0	3	0

D =

0	1	3	4	5	6	7	8
1	2	8	5	6	4	3	7

2．弗洛伊德（Floyd）算法

如果要计算赋权图中各对顶点两两之间的最短路径，这个问题等价于将每一个顶点分别作为起点来计算其到其它点的最短路径，最后可以确定出所有顶点间相互距离最短的路径。解决这个问题的方法之一是反复调用迪克斯特拉算法，由于总共需要执行 n 次这样的操作，因此这种算法的时间复杂度为 $O(n^3)$ 。

任意两点间最短距离的另一种方法是由 Floyd. R.W 提出的，称之为 Floyd 算法。这种算法可以有效的减少运算，提高效率。

假设图 G 权的邻接矩阵为 A_0

$$A_0=\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \cdots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix}$$

来存放各边长度，其中：

$a_{ii}=0 \quad i=1,2,\cdots,n$；

$a_{ij}=\infty$ $\quad i,j$之间没有边（程序中可以所有权重都不可达到的充分大的数来代替）；

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

对于无向图，A_0是对称矩阵，$a_{ij}=a_{ji}$。

Floyd 算法的基本思想：递推产生一个矩阵序列$A_0,A_1,\cdots,A_k,\cdots,A_n$，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，$i,j,k=1,2,\cdots,n$。

最后，当$k=n$时，A_n即是各顶点之间的最短路径的值。

例 4. 下图是六个城市$c_1,c_2,\cdots,c_6$公路交通图：每条边权表示从c_i到c_j的直达的路程，若无直达（即两城市不相邻）则用∞表示，试求出任意两城市间的最短路线。

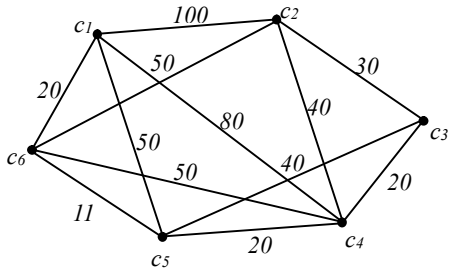


图 9

解：根据图写出其带权邻接矩阵为：

$$A=\begin{pmatrix} 0 & 100 & \infty & 80 & 50 & 20 \\ 100 & 0 & 30 & 40 & \infty & 50 \\ \infty & 30 & 0 & 20 & 40 & \infty \\ 80 & 40 & 20 & 0 & 20 & 50 \\ 50 & \infty & 20 & 10 & 0 & 110 \\ 20 & 50 & \infty & 50 & 110 & 0 \end{pmatrix}$$

根据弗洛伊德算法，可写出其MATLAB程序，其中矩阵A最初表示带权邻接矩阵，输出的时候表示的是最短距离矩阵。Path用于存放两点达到最小距离所经过的顶点序号：

```
clear all;close all;
clc;
B=[0 100 Inf 80 50 20
  zeros(1,2) 30 40 Inf 50
  zeros(1,3) 20 40 Inf
  zeros(1,4) 20 50
```

```
    zeros(1,5),110
    zeros(1,6)];
    A=B+B';
  m=length(A);
  Path=zeros(m);
  for k=1:m
     for i=1:m
        for j=1:m
           if A(i,j)>A(i,k)+A(k,j)
              A(i,j)=A(i,k)+A(k,j);
              Path(i,j)=k;
           end
        end
     end
  end
  A, Path
```

程序输出为：

A =

0	70	90	70	50	20
70	0	30	40	60	50
90	30	0	20	40	70
70	40	20	0	20	50
50	60	40	20	0	70
20	50	70	50	70	0

Path =

0	6	5	5	0	0
6	0	0	0	4	0
5	0	0	0	0	4
5	0	0	0	0	0
0	4	0	0	0	1
0	0	4	0	1	0

例如 A(2,5)=60,Path(2,5)=4 表示 c_2，c_5 之间的最短距离为 60，且路径为 c_2, c_4, c_5；A(3,4)=20,Path(3,4)=0 表示 c_3，c_4 之间的最短距离为 20，两城市相邻所以不经过别的点。

8.3.3 图的匹配及应用

在实际生活和工作中，总会涉及到一些将两类不同的对象进行配对的问题。例如将不同工作分配给不同的人员来完成，以期达到最优的效率；舞会中尽可能多的让男士和女士结成舞伴从而避免冷场等。这些问题可以用图来刻画并用图论的相关理论进行研究。

1. 图的匹配

定义 1： 若 $M \subset E(G)$，$\forall e_i, e_j \in M$，e_i 与 e_j 相互独立（$i \neq j$），则称 M 为图 G 的一

个匹配（matching）；M 中的一条边的两个顶点叫做在对集 M 中相配(matched)；M 中的顶点称为被 M 匹配；G 中每个顶点皆被 M 匹配时，M 称为完美匹配；对于图 G 的任意匹配 M'，若匹配 M 若满足 $|M|>|M'|$，则 M 称为最大匹配。显然完美匹配必定是最大匹配，反之则不尽然。

从一定意义上来讲，图的匹配就是它的一个相互独立的边的集合。

7 名大学生 *Ben(B), Don(D), Felix(F), June(J), Kim(K), Lilly(L), Maria(M)*即将大学毕业面临找工作的问题，学校的就业中心提供了会计（*a*），咨询师（*c*），编辑（*e*），程序员（*p*），记者（*r*），秘书（*s*）和教师（*t*）7 个职位供他们选择，每个学生根据自己的喜好分别对这些职位提出了申请：

B: c, e;　　*D: a, c, p, s ,t;*　　*F: c, r;*　　*J: c ,e, r;*

K: a ,e, p ,s;　　*L: e ,r;*　　*M: p, r, s, t.*

在一人一职的前提下，就业中心能否给每个学生都安排到他们所申请的职位呢？

在这个例子中，很自然的可以将学生和工作分别看作两个点的集合：

$U=\{B,D,F,J,K,L,M\}$，$W=\{a,c,e,p,r,s,t\}$

用学生对工作的申请情况表示集合间点的相邻关系，则可以得到一个二部图。

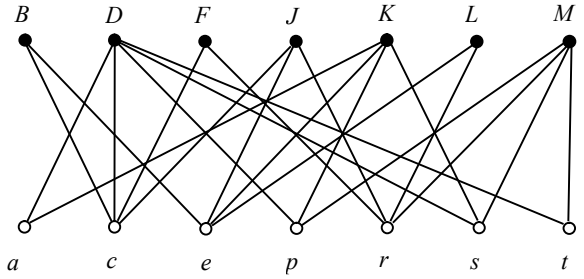


图 10

看起来好像 7 个学生，7 份工作，恰好每个人都可以每个人都安排一个，但是仔细考虑这个问题可以知道 *B,F,J,L* 四个人申请的是 3 份工作，所以至少对他们 4 人来说，至少有一个人不能得到他们申请的工作，从而 7 个学生不是每个人都能得到自己申请的工作。

定义 2：若二部图的两个点集为 U 和 W（不妨令 $|U|<|W|$），非空集合 X（$X\subseteq U$）当中元素 x 相邻的点的集合称为 x 的邻集，记为 $N(x)$；X 中所有元素的相邻点构成的集合称为 X 的邻集，记为 $N(X)$，显然有 $N(X)\subseteq W$。

若邻集满足 $|N(X)|>|X|$，则称集合 X 是可邻的（neighborly）。对于 U 的每个非空子集 X 如果都可邻，则称集合 U 是可邻的。例如在上面的例子中 $N(B)=\{c,e\}$。若记 $X=\{B,F,J,L\}$，$N(x)=\{c,e,r\}$，由于 $N(X)=3, X=4$ 不满足 $|N(X)|>|X|$，因此 $|X|$ 不可邻的。

引入邻集的意义在于对二部图的匹配情况进行判定。

定理 1：二部图 G 的两个点集分别为 U,W 且满足 $r=|U|<|W|$。G 包含一个匹配 $|M|=r$ 的

充要条件是U可邻。

推论：若G是k（$k>0$)正则二部图，则G有完美匹配。

由定理 1 以及推论可以得到图论中的一个很著名的定理，称为婚姻定理（marriage theorem）。

定理 2：在r个姑娘和s个小伙子构成的群体中$(1\le r\le s)$，在相识的男女之间总共能发生r个婚姻的充要条件是对任意$1\le k\le r$，每k位姑娘所构成的群体都至少要认识k个小伙子。

定理 2 是定理 1 的一种更形象化的描述，大家可以对照着进行理解，需要强调的是，任意非平凡图都存在匹配。

定义 3：若G中有一路径P，其边交替地在匹配M内外出现，则称P为M的交错路径（alternating path）;交错路径的起止顶点都未被匹配时，此交错路径称为可增广路径（augmenting path）。

若把可增广路径上在M外的边纳入匹配，把M内的边从匹配中删除，则被匹配的顶点数增加 2，匹配中的边数增加 1。

图的极大（maximal）匹配就是按照当前的匹配方法，不可能再添加更多边数的匹配；而最大（maximum）匹配是指在所有的匹配中包含边数最多的匹配，当然也是所有极大匹配中边数最多的匹配。

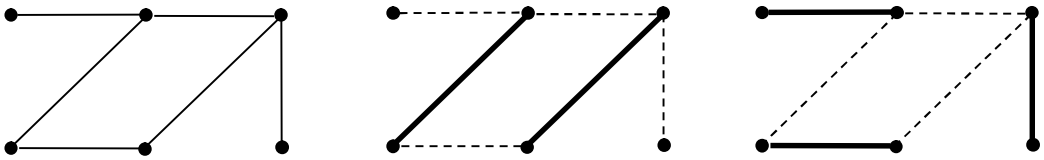


图 11

有两个重要的概念与匹配关系紧密：

1）独立边数

图G的最大独立边集，也就是包含图G中互不相邻的边最多的集合，的势称为G的独立边数（edge independence number），记为$\beta_1(G)$。显然如果M是G的最大匹配，则$\beta_1(G)=|M|$。

一个阶为n的图G存在完美匹配的充要条件是n为偶数且$\beta_1(G)=\dfrac{n}{2}$。

对于圈C_n、完全图K_n和完全二部图$K_{r,s}$（$1<r<s$）分别有：

$$\beta_1(C_n)=\beta_1(K_n)=\left\lfloor\frac{n}{2}\right\rfloor,\quad \beta_1(K_{r,s})=r。$$

2）边覆盖数

图的一个顶点和与之相关联的边称为相互覆盖（cover）。一个没有孤立点的图的边覆盖（edge cover）是一个覆盖了所有顶点的边集。最小边覆盖是覆盖所有顶点且包含边数最少的边集。

图G的最小边覆盖的势称为图的边覆盖数（edge covering number），记为$\alpha_1(G)$，不包

含孤立点的图才具有边覆盖数。

对于圈C_n、完全图K_n和完全二部图$K_{r,s}$（$1<r<s$），分别有：

$$\alpha_1(C_n)=\alpha_1(K_n)=\left\lceil\frac{n}{2}\right\rceil,\quad \beta_1(K_{r,s})=s。$$

因此

$$\alpha_1(C_n)+\beta_1(C_n)=n,\quad \alpha_1(K_n)+\beta_1(K_n)=n,\quad \alpha_1(K_{r,s})+\beta_1(K_{r,s})=r+s$$

记号$\lfloor x\rfloor$和$\lceil x\rceil$分别表示不超过x的最大整数和不小于x的最小整数。例如$\lfloor 2.5\rfloor=2$，$\lceil 2.5\rceil=3$。

可见图的独立边数和边覆盖数满足如下定理：

定理 3：对于每个阶为n且不含有孤立点的图G，总满足：

$$\alpha_1(G)+\beta_1(G)=n。$$

例 5．确定下图 G 的独立边数和边覆盖数。

G：

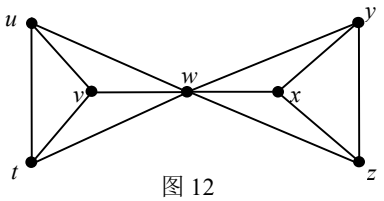


图 12

解：因为图 G 的阶为 7，因此$\beta_1(G)=\left\lfloor\frac{7}{2}\right\rfloor=3$。其一个最大独立边集为$\{tu,wx,yz\}$。

根据定理 3，有$\alpha_1(G)=n-\beta_1(G)=7-3=4$。其一个边覆盖集为$\{tu,vw,wx,yz\}$。

2．指派问题：

人员指派问题是用匹配来解决实际问题的典型例子。问题描述为：m 个工作人员$x_1,x_2,\cdots,x_m$去做n件工作$y_1,y_2,\cdots,y_n$，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是二部图，顶点集划分为$V(G)=X\bigcup Y$，$X=\{x_1,\cdots,x_m\}$，$Y=\{y_1,\cdots,y_n\}$，当且仅当x_i适合做工作y_i时，$x_iy_i\in E(G)$，求G中的最大匹配。

最大匹配可以用定理 1 来进行判别，但是要给出具体的图的最大匹配，必须借助一些算法来实现，埃德门兹(Edmonds)提出的匈牙利算法是解决这个问题的一个算法。

匈牙利算法的基本思想是：从G 的任意匹配M 开始,对X 中所有M 的非饱和点, 寻找M的可增广路径 若不存在可增广路径, 则M 为最大匹配; 若存在增广路径P, 则将P 中M 与非M 的边互换得到比M 多一条边的匹配M_1, 再对M_1 重复上述过程。

算法步骤如下：

设$G=(X,Y,E)$为二部图, 其中$X=\{x_1,x_2,\cdots,x_m\}$，$Y=\{y_1,y_2,\cdots,y_n\}$。

（1）从G中任意取定一个初始匹配M。

（2）若M饱和X-S的所有点, 则M是二部图G的最大匹配。否则取X中未被M匹配的任意顶点u，记$S=\{u\}$，$T=\Phi$。

（3）记$N(S)=\{v|u\in S,uv\in E\}$。若$N(S)=T$, 转向（2）；否则取$y\in N(S)-T$。

（4）若y是被M的匹配的, 设$yz\in M$，$S=S\cup\{z\}$，$T=T\cup\{y\}$，转（3）；否则，取可增广路径$P(u,y)$，令$M=(M-E(P))\cup(E(P)-M)$，转（2）。

例 6. 求下图的一个最大匹配。

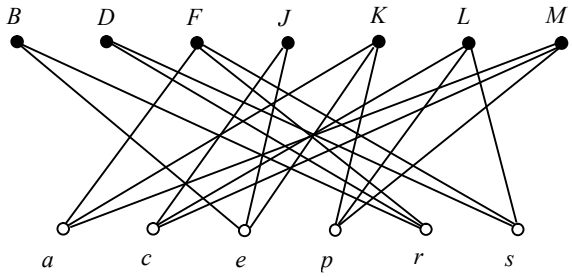
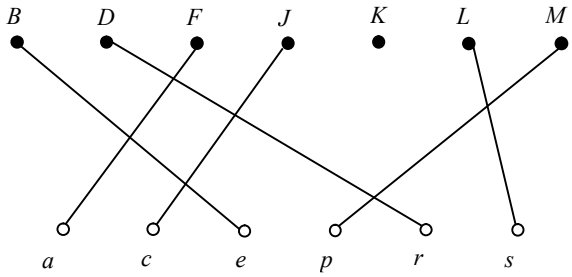


图 13

解：这是一个二部图，两个点集分别为$X=\{B,D,F,J,K,L,M\},Y=\{a,c,e,p,r,s\}$。

（为了简单起见，我们可以分别将X和Y中的元素记为自然数）由于分别在X和Y中的两个点一旦被配对以后，在后面的匹配中将不再出现，可以得到如下匹配：

M:



由于$|Y|=|M|=6$，因此上述匹配就是一个最大匹配。

把图的两个点集分别表示为行和列，可以得到一个$|X|$行$|Y|$列的邻接矩阵：

$$A=\begin{pmatrix}0&0&1&0&1&0\\0&0&0&0&1&1\\1&0&0&0&1&1\\0&1&1&0&0&0\\1&0&1&1&0&0\\0&1&0&1&0&1\\1&1&0&1&0&0\end{pmatrix}$$

注意此矩阵与一般图的邻接矩阵的差别。

MATLAB 程序实现：

```
clc;clear all;close all;
  A=[ 0 0 1 0 1 0
      0 0 0 0 1 1
      1 0 0 0 1 1
      0 1 1 0 0 0
      1 0 1 1 0 0
      0 1 0 1 0 1
      1 1 0 1 0 0];
[m,n]=size(A);
M=zeros(m,n);
%找到一个初始匹配 M
for i=1:m
      for j=1:n
            if   A(i,j)~=0
                 M(i,j)=1;
                 A(i,:)=0;
                 A(:,j)=0;
            end
       end
  end
while(1)
    %记录 X 中点的标号和标记*
     for i=1:m
         x(i)=0;
    end
    for i=1:n
        y(i)=0;
    end
    for i=1:m
        p=1;
%寻找 X 中 M 的所有非饱和点
        for j=1:n
            if M(i,j)==1
                p=0;
            end;
        end
        if (p)
            x(i)=-n-1;
        end;
      end
%将 X 中 M 的所有非饱和点都给以标号 0  和标记*, 程序中用 n+1 表示 0  标号,
%标号为负数时表示标记*
```

```
    p=0;
while(1)
       xi=0;
    for i=1:m
        if x(i)<0
           xi=i;
        break;
        end;
    end
%假如 X 中存在一个既有标号又有标记*的点, 则任取 X 中既有标号又有标记*的点 xi
    if xi==0
        p=1;
    break;
    end
%假如 X 中所有有标号的点都已去掉了标记*, 算法终止
    x(xi)=x(xi)*(-1); %去掉 xi 的标记*
    k=1;
    for j=1:n
      if A(xi,j)&y(j)==0
           y(j)=xi;
           yy(k)=j;
           k=k+1;
      end;
    end
%对与 xi 邻接且尚未给标号的 yj 都给以标号 i
if k>1
     k=k-1;
for j=1:k
       p1=1;
    for i=1:m
        if M(i,yy(j))==1
             x(i)=-yy(j);
             p1=0;
        break;
     end;
end
%将 yj 在 M 中与之邻接的点 xk (即 xkyj∈M), 给以标号 j 和标记*
if p1==1
     break;
     end;
end
if p1==1
     k=1;
     j=yy(j);
```

```
%yj 不是 M 的饱和点
while(1)
      P(k,2)=j;
      P(k,1)=y(j);
      j=abs(x(y(j)));
%任取 M 的一个非饱和点 yj, 逆向返回
    if(j==n+1)
      break;
    end
%找到 X 中标号为 0 的点时结束, 获得 M-增广路径 P
       k=k+1;
end
for i=1:k
      if M(P(i,1),P(i,2))==1
           M(P(i,1),P(i,2))=0;
%将匹配 M 在增广路径 P 中出现的边去掉
      else
         M(P(i,1),P(i,2))=1;
      end;
      end
%将增广路径 P 中没有在匹配 M 中出现的边加入到匹配 M 中
    break;
  end;
 end;
end
  if p==1
        break;
  end;
end
%假如 X 中所有有标号的点都已去掉了标记*, 算法终止，显示最大匹配 M
M
```

程序结束程序输出结果：

```
M =
     0     0     1     0     0     0
     0     0     0     0     1     0
     1     0     0     0     0     0
     0     1     0     0     0     0
     0     0     0     1     0     0
     0     0     0     0     0     1
     0     0     0     0     0     0
```

观察矩阵 M 的构成，可见在其中每一行和一列中仅有一个元素非 0，表示 X 中每个点仅能和 Y 中不多于一个点相匹配。例如 $M(1,3)=1$，表示 X 中标号为 1 的点和 Y 中标号为 3 的点是匹配 M 中的一条边。

此结果仅是所有最大匹配中的一个，要得到不同的结果，可以改变 X 以及 Y 标号的顺序得到不同的矩阵进行运算。不同的具体问题的程序可以改变输入矩阵来实现。

3．最优指派

最优**指派问题**：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$w(x_i y_j)\geq 0$，表示x_i干y_j工作的效益，求赋权图G上的权重之和最大的最大匹配，也称为最佳匹配。

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

定义 4: 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是二部图G的可行顶点标号。令$E_l=\{xy\,|\,xy\in E(G), l(x)+l(y)=w(xy)\}$，称以$E_l$为边集的$G$的生成子图为相等子图，记作$G_l$。

定理 4: 如果l是图G的一个可行顶点标号，$M\subseteq G_l$且是X到Y的一个完整匹配，则M即为从X到Y的最佳匹配。

Kuhn-Munkres 算法

设$G=(X,Y,E)$，每条边xy有权$w(xy)$：

（1）选定初始可行顶点标号l，通常取

$$\begin{cases} L(x)=\max\{w(x,y)\,|\,y\in Y\}, & x\in X \\ L(y)=0, & y\in Y \end{cases}$$

在相等子图G_l中选取一个匹配M。

（2）若X中顶点皆被M匹配，停止，M即G的权最大的最大匹配；否则，取G_l中未被M匹配的顶点u，令$S=\{u\}$，　$T=\Phi$。

(3)若$N_{G_l}(S)\supset T$，转（4）；若$N_{G_l}(S)=T$，取

$$\alpha_l=\min_{x\in S, y\in T^c}\{l(x)+l(y)-w(xy)\}$$

其中T^c表示T在Y中的补集，建立一个新的标号l'

$$l'(v)=\begin{cases} l(v)-\alpha_l, & v\in S \\ l(v)+\alpha_l, & v\in T \\ l(v), & \text{其它} \end{cases}$$

注意到$\alpha_l>0$，且$N_{G_{l'}}(s)\neq T$。用l'与$G_{l'}$分别替换l与G_l。

（4）选$N_{G_l}(S)-T$中一顶点y，若y已被M匹配，且$yx\in M$，则$S=S\bigcup\{z\}$，$T=T\bigcup\{y\}$，转(3)；否则，取G_l中一个M可增广路径$P(u,y)$，令$M=(M-E(P))\bigcup(E(P)-M)$，转(2)。

其中$N_{G_l}(S)$是G_l中S的相邻顶点集。

例 7. 求矩阵A所对应的一个最佳覆盖。

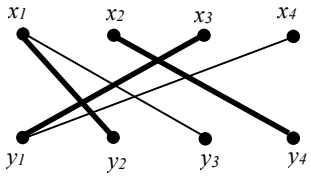
$$A=\begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{array}\begin{array}{c} \begin{array}{cccc} y_1 & y_2 & y_3 & y_4 \end{array} \\ \begin{pmatrix} 4 & 5 & 5 & 1 \\ 2 & 2 & 4 & 6 \\ 4 & 2 & 3 & 3 \\ 5 & 0 & 2 & 1 \end{pmatrix} \end{array}$$

解：该矩阵表示的是一个完全二部图$K_{4,4}$，欲求出最佳覆盖，即权重之和最大的匹配，则

必须先利用匈牙利算法求出一个最大匹配，显然 $K_{4,4}$ 是存在完美匹配的。因此该题目就是求一个权重之和最大的完美匹配。计算过程如下：

(1)取初始可行顶点标记 l,作出相等子图 G_1 并给出初始匹配 $M_1=\{x_1y_2,x_2y_4,x_3y_1\}$：

L(y) / L(x)	0	0	0	0
5	4	[5]	[5]	1
6	2	2	4	[6]
4	[4]	2	3	3
5	[5]	0	2	1



相等子图 G_1 与匹配 M_1

初始可行顶点标记取 $L(x)$ 为每行最大元素，$L(y)=0$，它们分别是和矩阵行列维数相等的向量。“[ ]”中的元素表示 $L(x)+L(y)=w(xy)$。

（2）显然此匹配非完美匹配，选择一个非饱和的点 $\{x_4\}$

$$S=\{x_4\},T=\Phi$$

（3）$y_1\in N_{G_l}(S)-T$；

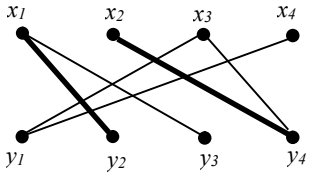
（4）$x_3y_1\in M$，故 $S=\{x_3,x_4\}$，$T=\{y_1\}$；

（5）$N_{G_l}(S)=T$，须调整标记，计算：

$$\alpha_1=\min\{L(x)+L(y)-w(xy)\,|\,x\in\{x_3,x_4\},y\in\{y_2,y_3,y_4\}\}$$
$$=\min\{4-1,4-3,4-3,5-0,5-2,5-1\}=1$$
$$L_1(x)=(5,6,3,4)\quad L_1(y)=(1,0,0,0)$$

从而得到新的可行点标记 l_1 和相等子图 G_2，并给出新的匹配 M_2：

L1(y) / L1(x)	1	0	0	0
5	4	[5]	[5]	1
6	2	2	4	[6]
3	[4]	2	[3]	[3]
4	[5]	0	2	1



相等子图 G_2 与匹配 M_2

（6）$y_3\in N_{G_2}(S)-T$，$P=x_4y_1x_3y_3$ 为交错路径，故

$$M_2=M_1\oplus P=\{x_1y_2,x_2y_3,x_3y_3,x_4y_1\}$$

由于 M_2 中所有点均饱和，所以当前匹配即为最佳匹配，最大权重之和为：

$$w(M_2)=w(x_1y_2)+w(x_2y_3)+w(x_3y_3)+w(x_4y_1)=19$$

MATLAB 程序实现：

```
function cnum = min_line_cover(Edge)    % function cnum = min_line_cover(Edge)
    [r_cov,c_cov,M,stepnum] = step2(Edge);
    [c_cov,stepnum] = step3(M,length(Edge));
    [M,r_cov,c_cov,Z_r,Z_c,stepnum] = step4(Edge,r_cov,c_cov,M);
    cnum = length(Edge)-sum(r_cov)-sum(c_cov);
```

算法函数 Hung_Al.m

```
function [Matching,Cost] = Hung_Al(Matrix)
   Matching = zeros(size(Matrix));
   % 找出每行和每列相邻的点数
      num_y = sum(~isinf(Matrix),1);
      num_x = sum(~isinf(Matrix),2);
   % 找出每行和每列的孤立点数
      x_con = find(num_x~=0);
      y_con = find(num_y~=0);
   %将矩阵压缩、重组
      P_size = max(length(x_con),length(y_con));
      P_cond = zeros(P_size);
      P_cond(1:length(x_con),1:length(y_con)) = Matrix(x_con,y_con);
      if isempty(P_cond)
         Cost = 0;
         return
      end
% 确保存在完美匹配，计算矩阵边集
         Edge = P_cond;
         Edge(P_cond~=Inf) = 0;
         cnum = min_line_cover(Edge);
         Pmax = max(max(P_cond(P_cond~=Inf)));
         P_size = length(P_cond)+cnum;
         P_cond = ones(P_size)*Pmax;
         P_cond(1:length(x_con),1:length(y_con)) = Matrix(x_con,y_con);
%主函数程序,此处将每个步骤用 switch 命令进行控制调用步骤函数
exit_flag = 1;
   stepnum = 1;
   while exit_flag
      switch stepnum
         case 1
            [P_cond,stepnum] = step1(P_cond);
         case 2
            [r_cov,c_cov,M,stepnum] = step2(P_cond);
         case 3
            [c_cov,stepnum] = step3(M,P_size);
         case 4
            [M,r_cov,c_cov,Z_r,Z_c,stepnum] = step4(P_cond,r_cov,c_cov,M);
         case 5
            [M,r_cov,c_cov,stepnum] = step5(M,Z_r,Z_c,r_cov,c_cov);
         case 6
            [P_cond,stepnum] = step6(P_cond,r_cov,c_cov);
         case 7
```

```
            exit_flag = 0;
        end
    end
Matching(x_con,y_con) = M(1:length(x_con),1:length(y_con));
Cost = sum(sum(Matrix(Matching==1)));
%下面是 6 个步骤函数 step1~step6
%步骤 1：找到包含 0 最多的行，从该行减去最小值
function [P_cond,stepnum] = step1(P_cond)
    P_size = length(P_cond);
        for ii = 1:P_size
        rmin = min(P_cond(ii,:));
        P_cond(ii,:) = P_cond(ii,:)-rmin;
    end
    stepnum = 2;
%步骤 2：在 P-cond 中找一个 0，并找出一个以该数 0 为星型的覆盖
function [r_cov,c_cov,M,stepnum] = step2(P_cond)
%定义变量 r-cov，c-cov 分别表示行或列是否被覆盖
    P_size = length(P_cond);
    r_cov = zeros(P_size,1);
    c_cov = zeros(P_size,1);
    M = zeros(P_size);
        for ii = 1:P_size
        for jj = 1:P_size
            if P_cond(ii,jj) == 0 && r_cov(ii) == 0 && c_cov(jj) == 0
                M(ii,jj) = 1;
                r_cov(ii) = 1;
                c_cov(jj) = 1;
            end
        end
    end
    % 重初始化变量
    r_cov = zeros(P_size,1);
    c_cov = zeros(P_size,1);
    stepnum = 3;
%步骤 3：每列都用一个 0 构成的星型覆盖，如果每列都存在这样的覆盖，则 M 为最大匹配
function [c_cov,stepnum] = step3(M,P_size)
    c_cov = sum(M,1);
    if sum(c_cov) == P_size
        stepnum = 7;
    else
        stepnum = 4;
    end
%步骤 4：找一个未被覆盖的 0 且从这出发点搜寻星型 0 覆盖。如果不存在，转步骤 5，否%
则转步骤 6
```

```
function [M,r_cov,c_cov,Z_r,Z_c,stepnum] = step4(P_cond,r_cov,c_cov,M)
P_size = length(P_cond);
zflag = 1;
while zflag
        row = 0; col = 0; exit_flag = 1;
        ii = 1; jj = 1;
        while exit_flag
            if P_cond(ii,jj) == 0 && r_cov(ii) == 0 && c_cov(jj) == 0
                row = ii;
                col = jj;
                exit_flag = 0;
            end
            jj = jj + 1;
            if jj > P_size; jj = 1; ii = ii+1; end
            if ii > P_size; exit_flag = 0; end
        end
       if row == 0
           stepnum = 6;
           zflag = 0;
           Z_r = 0;
           Z_c = 0;
        else
           M(row,col) = 2;
           if sum(find(M(row,:)==1)) ~= 0
               r_cov(row) = 1;
               zcol = find(M(row,:)==1);
               c_cov(zcol) = 0;
            else
               stepnum = 5;
               zflag = 0;
               Z_r = row;
               Z_c = col;
            end
        end
end
%步骤 5：
function [M,r_cov,c_cov,stepnum] = step5(M,Z_r,Z_c,r_cov,c_cov)
   zflag = 1;
   ii = 1;
   while zflag
      rindex = find(M(:,Z_c(ii))==1);
      if rindex > 0
         ii = ii+1;
         Z_r(ii,1) = rindex;
```

```
        Z_c(ii,1) = Z_c(ii-1);
      else
        zflag = 0;
      end
        if zflag == 1;
        cindex = find(M(Z_r(ii),:)==2);
        ii = ii+1;
        Z_r(ii,1) = Z_r(ii-1);
        Z_c(ii,1) = cindex;
      end
  end
  for ii = 1:length(Z_r)
    if M(Z_r(ii),Z_c(ii)) == 1
      M(Z_r(ii),Z_c(ii)) = 0;
    else
      M(Z_r(ii),Z_c(ii)) = 1;
    end
  end
  r_cov = r_cov.*0;
  c_cov = c_cov.*0;
  M(M==2) = 0;
  stepnum = 3;
% 步骤 6：
function [P_cond,stepnum] = step6(P_cond,r_cov,c_cov)
a = find(r_cov == 0);
b = find(c_cov == 0);
minval = min(min(P_cond(a,b)));
P_cond(find(r_cov == 1),:) = P_cond(find(r_cov == 1),:) + minval;
P_cond(:,find(c_cov == 0)) = P_cond(:,find(c_cov == 0)) - minval;
stepnum = 4;
```

主程序 exaple3.m

```
clc;
clear all;
A=[ 4 5 5 1
    2 2 4 6
    4 2 3 3
    5 0 2 1];
 [M,cost]=Hung_Al(A)
```

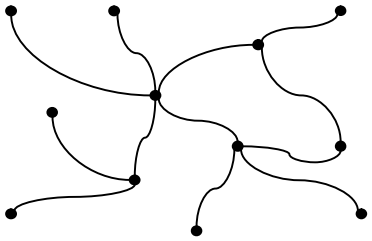
程序输出结果：

```
M =
     0     1     0     0
     0     0     0     1
     0     0     1     0
```

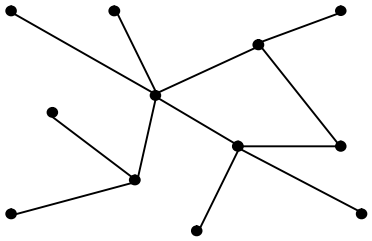
```
     1     0     0     0
cost =
    19
```

8.3.4 图的覆盖及应用

在某城区为监控和指挥交通，需要设置交通岗亭。如何设置尽可能少的交通岗亭，以监控该城区所有街道？



街区图



模拟图

图 14

首先假设在该城区不存在孤岛，即所有街道均连通，可以将街道交叉口看作顶点，街道看作边，因此可以用一个连通图来进行模拟。岗亭设置显然应该设置在道路交叉口的位置才能监控尽可能多的路口。

该问题的数学模型是：在一个连通图$G(V,E)$中，如何选择顶点集X($X \subseteq V$),使得X包含顶点数量最少并且这些顶点可以和E中所有的边相关联。

定义 1：一个图$G(V,E)$的顶点集X($X \subseteq V$）中如果任意两个顶点都不相邻，则称该顶点集是**独立**的。

定义 2：图$G(V,E)$的**独立点数**(vertex independence number)为包含点数最多的独立点集的势，记为$\beta(G)$。包含$\beta(G)$个顶点的独立点集称为**最大独立点集**(maximum independent set)。

对于圈C_n、完全图K_n和完全二部图$K_{r,s}$（$1<r<s$）分别有：

$$\beta(C_n)=\left\lfloor \frac{n}{2} \right\rfloor,\ \beta(K_n)=1,\ \beta(K_{r,s})=s。$$

定义 3：设$G(V,E)$为连通图，$X \subseteq V$，若$\forall e \in E$，e都与X中某个顶点关联，则称X是G的一个**点覆盖**（vertex cover），或者直接称为**覆盖**(cover)。若X是一个覆盖，但X的子集都不是覆盖，则称它是一个极小覆盖。包含顶点数最少的覆盖称为**最小覆盖**（minimum vertex cover）。最小覆盖的势称为图G的**点覆盖数**(vertex covering number)，记为$\alpha(G)$。

有时候也把最小点覆盖称为最小控制集。

对于圈C_n、完全图K_n和完全二部图$K_{r,s}$（$1<r<s$），分别有：

$$\alpha(C_n)=\left\lceil\frac{n}{2}\right\rceil,\ \alpha(K_n)=n-1,\ \alpha(K_{r,s})=r。$$

观察可得：

$$\alpha(C_n)+\beta(C_n)=n,\quad \alpha(K_n)+\beta(K_n)=n,\quad \alpha(K_{r,s})+\beta(K_{r,s})=r+s$$

可见图的独立点数和点覆盖数满足如下定理：

定理 1：对于每个阶为n且不含有孤立点的图G，总满足：

$$\alpha(G)+\beta(G)=n。$$

例 8．确定下图 G 的独立点数和点覆盖数。

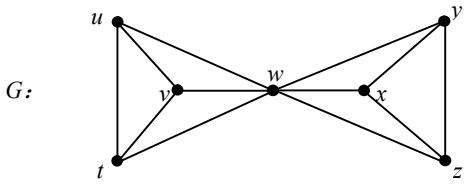


图 15

解：图G的阶为 7，由于w点与所有顶点均相邻，因此$G-w$得到两个K_3分别为：$\langle\{u,v,t\}\rangle$和$\langle\{x,y,z\}\rangle$，所以图G的独立点数为$\beta(G)=2$，即在两个K_3中分别取一点所构成的点集都是最大独立点集。根据定理 4 有$\alpha(G)=5$，例如$\{t,u,w,y,z\}$就是一个最小点覆盖。

求解图的最小点覆盖没有精确的算法，即该问题通常不能求出精确解，只能通过一些算法得到近似的解。常用的算法有逻辑算法和启发式算法。

1. 逻辑算法

逻辑算法是一种代数方法，首先定义逻辑代数运算的“和”与“乘”。

逻辑“和”称为“或”运算，运算相当于集合中的并集$\bigcup$运算，常采用“+”号来直观表示。例如$\{a\}+\{a,b\}=\{a,b\}$。

逻辑“乘”运算相当于集合中的交集$\bigcap$运算，通常用“$\bullet$”来表示。例如$\{a\}\bullet\{a,b\}=\{a\}$。

用逻辑算法求最小覆盖是基于极小覆盖的如下性质：对于覆盖K中的顶点v，当且仅当$v\in K,N(v)\notin K$或者$v\notin K,N(v)\in K$时，K为极小覆盖，找出所有极小覆盖中包含顶点最少的就是最小覆盖。

逻辑算法的步骤如下：

（1） 将每个顶点v_i与其邻集$N(v_i)$表示为逻辑和运算$v_i+N(v_i)$；

（2） 将所有顶点与其对应的逻辑预算求乘积，记为$\prod_i(v_i+N(v_i))$。

（3） 将（2）中的逻辑运算式化简。

例 9．求下图的最小点覆盖。

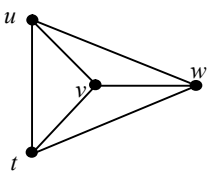


图 16

解：列出逻辑运算式并进行逻辑运算：

$$(\{u\}+\{w,v,t\})(\{v\}+\{w,u,t\})(\{t\}+\{u,v,w\})$$

$$=(\{u,v\}+\{w,u,t\}+\{w,v,t\}+\{w,u,v,t\})(\{t\}+\{u,v,w\})$$

$$=(\{u,v\}+\{w,u,t\}+\{w,v,t\})(\{t\}+\{u,v,w\})$$

$$=\{u,v,t\}+\{u,v,w\}+\{w,u,t\}+\{w,u,v,t\}+\{w,v,t\}+\{w,u,v,t\}$$

$$=\{u,v,t\}+\{u,v,w\}+\{w,u,t\}+\{w,v,t\}$$

注意这个图是一个K_4，其点覆盖数为 3，即最小点覆盖应有 3 个顶点。根据逻辑运算的结果可知图中任意 3 点构成的集合都是一个最小点覆盖。

2．启发式算法：

逻辑算法的计算复杂度随着顶点数量的增加呈指数级增长，但是在实际问题中往往顶点数量都比较庞大，用逻辑算法将非常困难。对于这种情况一般采用启发式的近似算法。

按照经验来讲，一般是把度比较大的顶点作为覆盖的点，因为它能覆盖更多的边。启发式算法过程如下：

设G为连通图，用K表示所求的最小覆盖，初始$K=\Phi$。

（1）若$V(G)=\Phi$，则无最小覆盖，算法终止。否则取$u\in V(G)$，使

$$\deg(u)=\max\{\deg(v)|v\in V(G)\}$$

（2）令$K=K\bigcup\{u\}$，$V(G)=V(G)-\{u,N(u)\}$，转（1）。

例 10．求下图的一个极小点覆盖。

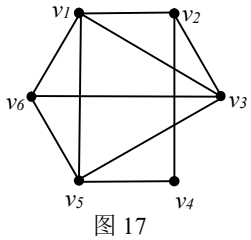
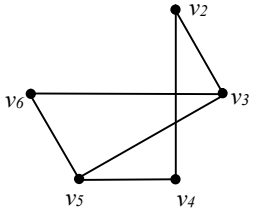


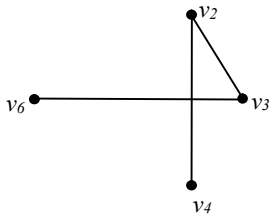
图 17

解：按照启发式算法，依次取度最大的顶点，并去掉和该点相关联的所有边，以及由于去掉相应的边产生的孤立点，直到所有边都被去掉为止。记初始覆盖集$K=\Phi$；

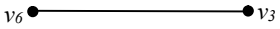
首先找度最大的点，不妨取v_1，$K=\{v_1\}$去掉v_1和与v_1关联的边，得到



在余下的点中再取度最大的点，这里取v_5，$K=\{v_1,v_5\}$，去掉和v_5以及与v_5关联的边得到



再取v_2，$K=\{v_1,v_5,v_2\}$，去掉v_2和与v_2关联的边得到，注意由于去掉v_2产生了孤立点v_4也一并去掉。



最后取v_6，$K=\{v_1,v_5,v_2,v_6\}$，去掉v_6和与v_6关联的边，所有边均被去掉，说明K即为原图的一个极小点覆盖，K中的顶点可以覆盖所有的边。

MATLAB 程序如下：

```
clc;
clear;
A= [0 1 1 0 1 1
    1 0 1 1 0 0
    1 1 0 0 1 1
    0 1 0 0 1 0
    1 0 1 1 0 1
    1 0 1 0 1 0];
 [m,n]=size(A);
 k=zeros(1,m);
 sum1=[];h=[];
 for o=1:m
     sum1=[];
   for p=1:m
     t=sum(A(p,:));
     sum1=[sum1,t];
   end
 %从小到大排序点的度,k1 为排序后的结果，index 表示 k1 中元素在原序列中的位置
    [k1,index]=sort(sum1);
 %将点按照度从大到小排序，b 为排序后的索引
    l=length(index);
   for ii=1:l
      b(ii)=index(l-ii+1);
   end
 %依次度最大的点，并删除其相邻的边
     A(b(1),:)=0;
     A(:,b(1))=0;
     h=[h,b(1)]
```

```
    A
   if A==zeros(m)
    break
   end
end
fprintf('A maximal cover is');
h
```

程序输出结果：

```
A maximal cover is
h =
     5     3     2     6
```

3．利用关联矩阵求极小覆盖：

由于点与边的覆盖关系，而关联矩阵可以很恰当的描述这种关系，因此可以利用关联矩阵来求图的极小点覆盖，这里我们以上面的例子来进行说明。

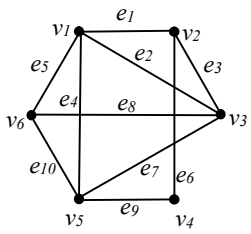


图 18

根据图首先写出相应的关联矩阵 R，初始化覆盖 $K=\Phi$。

$$R=\begin{pmatrix}1&1&0&1&1&0&0&0&0&0\\1&0&1&0&0&1&0&0&0&0\\0&1&1&0&0&0&1&1&0&0\\0&0&0&0&0&1&0&0&1&0\\0&0&0&1&0&0&1&0&1&1\\0&0&0&0&1&0&0&1&0&1\end{pmatrix}\begin{matrix}v_1\\v_2\\v_3\\v_4\\v_5\\v_6\end{matrix} \qquad (1)$$

1）在（1）中取恰好有两个 1 的列中 1 所在的行，例如 v_1 行，令 $v_1\in K$。划去 v_1 行及 v_1 行中元素 1 所在的列，得到

$$\begin{pmatrix}1&1&0&0&0\\1&0&1&0&0\\0&1&0&1&0\\0&0&1&1&1\\0&0&0&0&1\end{pmatrix}\begin{matrix}v_2\\v_3\\v_4\\v_5\\v_6\end{matrix} \qquad (2)$$

2）在（2）中取恰好有两个 1 的列中 1 所在的行，例如 v_6 行，令 $v_6\in K$。划去 v_6 行及

v_6 行中元素 1 所在的列，得到

$$\begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix} \begin{matrix} v_2 \\ v_3 \\ v_4 \\ v_5 \end{matrix} \qquad (3)$$

3）在（3）中取恰好有两个 1 的列中 1 所在的行，例如 v_5 行，令 $v_5 \in K$ 。划去 v_5 行及 v_5 行中元素 1 所在的列，得到

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{matrix} v_2 \\ v_3 \\ v_4 \end{matrix} \qquad (4)$$

4）在（4）中取恰好有两个 1 的列中 1 所在的行，例如 v_2 行，令 $v_2 \in K$ 。划去 v_2 行及 v_2 行中元素 1 所在的列，得到空矩阵，过程结束。最小覆盖为 $K = \{v_1, v_2, v_5, v_6\}$ 。

8.3.5 图的遍历问题

1．边的遍历-中国邮差问题

一名邮差负责投递某个街区的邮件。如何为他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？这一问题是我国管梅谷教授首先提出的，所以称之为中国邮差问题(Chinese Postman Problem)。

若街区的街道用边表示，街道长度用边的权重表示，邮局与街道交叉口用点表示，可以将这个问题描述为：在一个赋权无向连通图中，寻找一个回路（或者叫巡回），并且使得该回路的边权之和最小，这样的回路称为最佳回路。

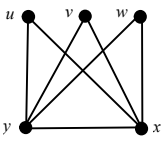
为解决这个问题，先来看下面相关的图论知识。

1）．相关定义

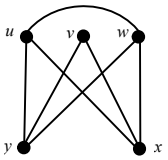
定义 1：连通图 $G = (V, E)$ 中，一条恰好遍历所有边一次的迹 T 称为欧拉迹（Eulerian trail）。相同地，一条恰好遍历所有边一次的回路 C 称为欧拉回路（Eulerian circuit）。包含欧拉回路的图称为欧拉图（Eulerian graph）。

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

欧拉回路可以看做是欧拉迹起点和终点相同的情形。而欧拉迹可以看做欧拉回路去掉一条边以后得到。



C: u-x-w-y-v-x-y-u



T: u-x-w-y-v-x-y-u

图 19 欧拉图与非欧拉图

关于欧拉回路和欧拉迹有如下的定理：

定理 1：（1）G 是 Euler 图的充分必要条件是 G 连通且每顶点皆偶次。

（2）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，

$E(C_i)\bigcap E(C_j)=\Phi(i\neq j)$。

（3）G 中有欧拉迹的充要条件是 G 连通且恰好有两个奇顶点，欧拉迹以其中一个奇点开始，到另一奇点结束。

一个有趣的现象：凡是可以“一笔画”的图形或包含欧拉回路，或包含欧拉迹。

2）Fleury 算法

1921 年，Fleury 给出求欧拉回路的算法，基本思想是：从任一点出发，每当访问一条边时，先进行检查可供选择的边，如果可供选择的边不止一条，则不选桥作为访问边，直到没有可选择的边为止。

算法步骤描述如下：

（1）$\forall v_0\in V(G)$，令 $T_0=v_0$。

（2）假设迹 $T_i=v_0e_1v_1\cdots e_iv_i$ 已经选定，那么按下述方法从 $E-\{e_1,\cdots,e_i\}$ 中选取边 e_{i+1}：

① e_{i+1} 和 v_i 相关联；

② 除非没有别的边可选择，否则 e_{i+1} 不能是 $G_i=G-\{e_1,\cdots,e_i\}$ 的桥。

（3）当第（2）步不能再执行时，算法停止。

3）邮差问题的求解

中国邮差问题的数学模型已经可以完全用图的模型来概括，因此也可以用图的相关知识来求解。分下面三种情形：

若此问题对应的连通赋权图 G 是欧拉图，则可用 Fleury 算法求欧拉回路。

而对于非欧拉图，则其任何一个回路必然经过某些边多于一次。解决这类问题的一般方法是在一些点对之间引入重边（重边与它的平行边具有相同的权），使得原图成为欧拉图，但要求添加的重复边的权之和最小。

若 G 正好有两个奇顶点，则可用 Edmonds 和 Johnson 给出的算法来解决，该算法描述如下：

设 G 是连通赋权图，$u,v\in V$ 为仅有的两个奇顶点。

（1）用 Floyd 算法求出 u 与 v 的最短距离 $d(u,v)$ 以及最短路径 P。

（2）在 P 的各相邻点之间依次添加等的平行边得到 $G'=G\bigcup P$ 。

（3）用 Fleury 算法求 G' 的欧拉回路即为中国邮差问题的解。

若 G 的奇顶点有 $2n$ $(n\geq 2)$ 个，则采用 Edmonds 最小匹配算法，该算法的基本思想是：先将奇点进行最佳匹配，再沿点对之间的最短路径添加重边可得到 G^* 为欧拉图，G^* 的最佳回路即为原图的最佳回路。算法步骤为：

（1）求出 G 中所有奇点之间的最短路径和距离。

（2）以 G 的所有奇点为顶点集，构造一完全图，边上的权为两端点在原图 G 中的最短距离，将此图记为 G' 。

（3）求出G'中的最小理想匹配M，得到奇顶点的最佳匹配。

（4）在G中沿配对顶点之间的最短路径添加重边得到欧拉图G^*。

（5）用 Fleury 算法求出G^*的欧拉回路，这就是G的最佳回路。

例 11. 若某街区示意图如下图所示，求一条最佳邮差巡回。

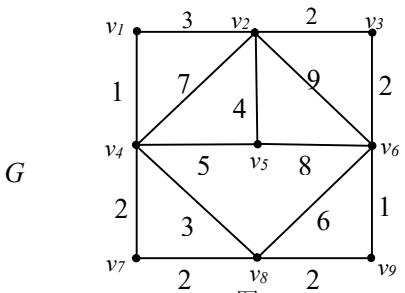


图 20

解：首先观察该图中有 4 个奇点分别为：$\deg(v_2)=\deg(v_4)=\deg(v_6)=5$，$\deg(v_5)=3$。

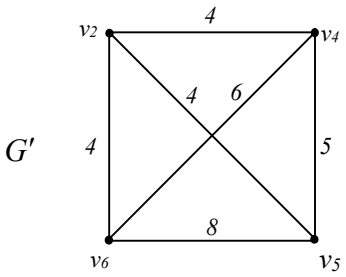
分别求出这 4 个点之间的最短距离和最短路径（可用 Floyd 算法求解）：

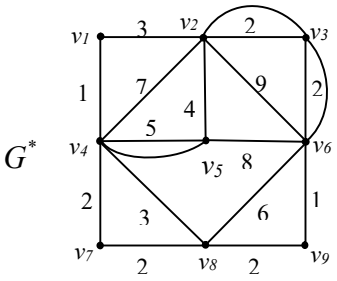
$d(v_2,v_4)=4$，$P:v_2v_1v_4$；　　$d(v_2,v_5)=4$，$P:v_2v_5$；

$d(v_2,v_6)=4$，$P:v_2v_3v_6$；　　$d(v_4,v_5)=5$，$P:v_4v_5$；

$d(v_4,v_6)=6$，$P:v_4v_8v_9v_6$；　　$d(v_5,v_6)=8$，$P:v_5v_6$；

以v_2,v_4,v_5,v_6为顶点，以它们之间的距离为权构造完全图G'。





求出G'中的权重之和最小的完美匹配$M=\{v_2v_6,v_4v_5\}$（可用匈牙利算法）。

在G中沿着v_2到v_6的最短路径$v_2v_3v_6$顺次添加重边，沿着v_4到v_5的最短路径v_4v_5添加重边可得到G^*。

G^*显然为欧拉图，可用 Fleury 方法求出其欧拉回路即为所求的最佳邮差回路。

若邮局有$k(k\geq 2)$位邮差，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？我们把这一问题称为多邮差问题。

多邮差问题的数学模型如下：

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

（i）$v_0\in V(C_i)$，$i=1,2,\cdots,k$，

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

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

显然必须尽量均匀的分配任务给每个邮差，即每个邮差所走的路线尽量要均等才能使总任务完成的时间最早。

2．点的遍历-旅行商问题

一名推销员准备前往若干城市推销产品。如何为设计一条最短的旅行路线使得他从驻地出发，经过每个城市恰好一次，最后返回驻地？这个问题就是著名的旅行商问题（Traveling Salesman Problem）或叫 TSP 问题。

1）相关定义

定义 1：图 $G=(V,E)$ 中，一条恰好遍历所有顶点一次的路径被称为汉密尔顿路径（Hamiltonian Path）。同样地，一条恰好遍历所有顶点一次的圈 C 称为汉密尔顿圈（Hamiltonian cycle）。

包含汉密尔顿圈的图称为汉密尔顿图。

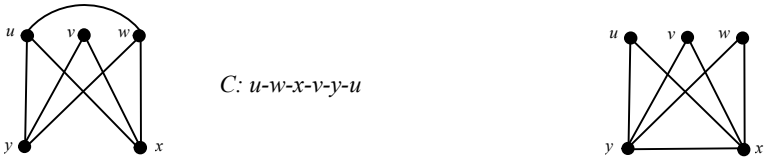


图 21 汉密尔顿图与非汉密尔顿图

对于汉密尔顿图的判别不如欧拉图那样容易，因此只能根据定义判别。同时也只有一些充分或者必要的条件，还未找到有效的充要条件。

2）旅行商问题的求解

假设每个城市都可以直达其它城市，将每个城市用点表示，将城市之间的交通里程用带权的边表示，则旅行商问题可以用图来建立模型。其描述为：在一个赋权完全图中，找出一个有最小权的汉密尔顿圈。称这种圈为最优圈。

由于汉密尔顿图的判别缺少有效的方法，因此旅行商问题也没有一个确切的方法来求解，也就是说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的解，但未必是最优解。

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。这种修改的方法叫做**改良圈算法**。由于每次一般是用两条较小权的边替换权较大的两条边，因此又叫**二边逐次修正法**。算法描述如下：

设初始圈 $C=v_1v_2\cdots v_nv_1$。

（1）对于 $1<i+1<j<n$，构造新的 Hamilton 圈:

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

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。

（2）转（1），直至无法改进，停止。

用二边逐次修正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克鲁斯卡尔算法加以说明。假设 C 是 G 中的最优圈。则对于任何顶点 v，$C-v$ 是在 $G-v$ 中的汉密尔顿路径，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。

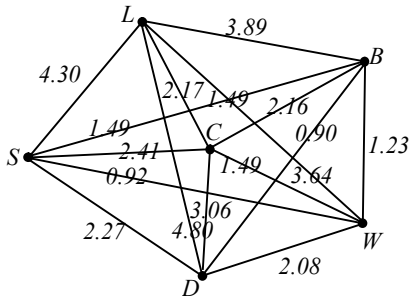
二边逐次修正法现在已被进一步发展，圈的修改过程一次替换三条边比一次仅替换两条边更有效。但是并不是一次替换的边越多越好。

例 1．某公司派推销员从北京（B）乘飞机到上海(S)、拉萨(L)、成都(C)、大连(D)、武汉(W)五城市做产品推销，每城市恰去一次再回北京，应如何安排飞行路线，使旅程最短？

各城市之间的航线距离如下表（单位：10^3 km）：

	B	S	L	C	D	W
B	0	1.49	3.89	2.16	0.90	1.23
S	1.49	0	4.30	2.41	2.27	0.92
L	3.89	4.30	0	2.17	4.80	3.64
C	2.16	2.41	2.17	0	3.06	1.49
D	0.90	2.27	4.80	3.06	0	2.08
W	1.23	0.92	3.64	1.49	2.08	0

解：假设每个城市之间都有直飞的航线，则可以将该问题转换为在一个 6 阶完全图中如何找到一个最优 H 圈的问题。用点表示城市，带权边表示航线距离，可以得到如下的图：



由于完全图中任意两点都是相邻的，因此以任意一个顶点作为起点以及终点且遍历所有顶点都可以得到一个圈。不妨令初始圈为$C_0:LSDCWBL$，权重之和$W_0=16.24$.

下面对该圈进行修正：

找初始圈C_0中两条不相邻的边 DC 和 WB，有$W_{DC}+W_{WB}>W_{DB}+W_{CW}$，故用 SC 和 DW 替换 SD 与 CW，得到新圈$C_1:LSCDWBL$，权重之和$W_1=14.34$；

重复上述步骤，直到找到比较满意的圈为止。

由于该算法简单，仅需要进行重复操作，因此可以借助计算机来完成。下面的 MATLAB 程序可以帮助我们进行运算。

MATLAB 验证程序：

```
%注意该程序需要多次运行，每次可以给出一个不同的 C0，最后算出最小权重的圈即为最
%优 H 圈
clc;
clear all;
close all;
a=[0 1.49 3.89 2.16 0.90 1.23
    0 0 4.30 2.41 2.27 0.92
```

```
        zeros(1,3) 2.17 4.80 3.64
        zeros(1,4) 3.06 1.49
        zeros(1,5) 2.08
        zeros(1,6)];
a=a+a';
c0=[5 1:4 6];
L=length(c0);
flag=1;
while flag>0
          flag=0;
     for m=1:L-3
         for n=m+2:L-1
            if a(c0(m),c0(n))+a(c0(m+1),c0(n+1))
                                  <a(c0(m),c0(m+1))+a(c0(n),c0(n+1))
                      flag=1;
                c0(m+1:n)=c0(n:-1:m+1);
            end
        end
   end
end
sum=0;
for i=1:L-1
    sum=sum+a(c0(i),c0(i+1));
end
sum
```

经过多次运算可以记录下每个不同圈的权重之和，最后将最小的一个作为近似最优圈，但是当顶点数量比较多的时候，这种方法的计算将是非常困难的：顶点数量为n时，所有可能构成圈的组合数有$n!$种，$n=6$时，需要进行720次运算，$n=20$时，运算次数将达到惊人的2432902008176640000次，显然需要找到一种更科学有效的方法。