# 第4章　图　　论

图论起源于著名的柯尼斯堡七桥问题. 在柯尼斯堡的普莱格尔河上有 7 座桥将河中的岛及岛与河岸连接起来，如图 4.1 所示，$A$，$B$，$C$，$D$ 表示陆地. 问题是要从这 4 块陆地中任何一块开始，通过每一座桥正好一次，再回到起点. 然而无数次的尝试都没有成功. 欧拉在 1736 年解决了这个问题，他用抽象分析法将这个问题化为第一个图论问题，即把每一块陆地用一个点来代替，将每一座桥用连接相应的两个点的一条线来代替，从而相当于得到一个图. 欧拉证明了这个问题没有解，并且推广了这个问题，给出了对于一个给定的图可以某种方式走遍的判定法则. 这项工作使欧拉成为图论（及拓扑学）的创始人.

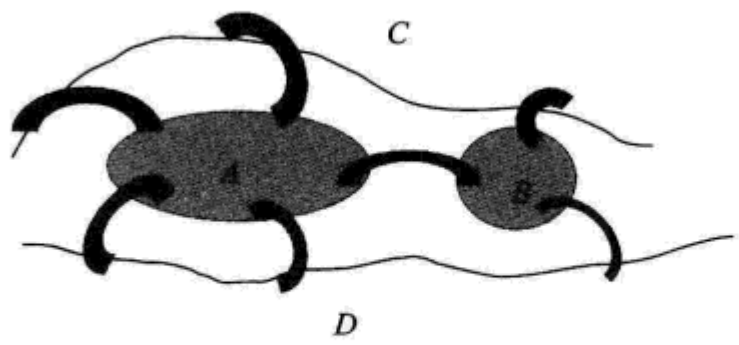

图 4.1　柯尼斯堡七桥图

1859 年，英国数学家哈密顿发明了一种游戏：用一个规则的实心十二面体，它的 20 个顶点标出世界著名的 20 个城市，要求游戏者找一条沿着各边通过每个顶点刚好一次的闭回路，即“绕行世界”. 用图论的语言来说，游戏的目的是在十二面体的图中找出一个生成圈. 这个问题后来就称为哈密顿问题. 由于运筹学、计算机科学和编码理论中的很多问题都可以化为哈密顿问题，这一问题引起广泛的注意和研究.

在图论的历史中，还有一个最著名的问题——四色猜想. 这个猜想说，在一个平面或球面上的任何地图能够只用四种颜色来着色，使得没有两个相邻的国家有相同的颜色. 每个国家必须由一个单连通域构成，而两个国家相邻是指它们有一段公共的边界，而不仅仅只有一个公共点. 每个地图可以导出一个图，其中国家都是点，当相应的两个国家相邻时这两个点用一条线来连接. 所以四色猜想是图论中的一个问题. 它对图的着色理论、平面图理论、代数拓扑图论等分支的发展起到推动作用.

在本章中，我们将重点讲解有关图论问题的各种算法.

## 4.1　图的基本概念

图论中的“图”并不是通常意义上的几何图形或物体的形状图，而是以一种抽象的形式来表达一些确定的事物之间的联系的一个数学系统. 从这个意义上说，有关事物间联系的问题，都可以从图的角度加以分析，这样，图的一些基本方法的运用是

很重要的. 数学建模竞赛中对图的方法的运用也是非常多的.

**定义 4.1** 一个有序二元组 $(V,E)$ 称为一个图,记为

$$G=(V,E)$$

其中 $V$ 称为图 $G$ 的顶点集,其元素称为顶点;$E$ 称为图 $G$ 的边集,其元素称为边.

**定义 4.2** 若将图 $G$ 的每一条边 $e$ 都对应一个实数 $F(e)$,则称 $F(e)$ 为该边的权,并称图 $G$ 为赋权图(网络),记为 $G=(V,E,F)$.

**定义 4.3** 一个 $n$ 阶赋权图 $G=(V,E,F)$ 的权矩阵为 $\boldsymbol{A}=(a_{ij})_{n\times n}$,其中

$$a_{ij}=\begin{cases}F(v_iv_j), & v_{ij}\in E\\ 0, & i=j\\ +\infty, & v_{ij}\notin E\end{cases}$$

作为阅读此章的基础,图的更多概念请读者参考参考文献《数学方法选讲》(刘承平,2002).

## 4.2 Dijkstra 算法与 Warshall-Ford 算法

最短路径问题(shortest route problem)是竞赛中常见的一类中等难度的求解问题,如 1999 年国赛的钢管运输问题,甚至有时一些看似跟最短路径问题无关的问题也可以归结为最短路径问题. 但实际问题通常由于结点数非常多,让人感觉困难重重,无法入手. 解这类题时同学们往往不得要领,不少同学想采用穷举法把所有可能的情况全部列出,再找出其中最短的那条路径;或是采用算法中的递归或深度搜索算法寻找. 这两种方法都是费时非常多的,如果城市数目过多则很可能要超时,显然这几种算法对于求最短路径这类最优解问题是不合适的,所以在数模竞赛中求解时运用得非常少,通常也不建议读者使用. 本节简要地和读者一起探讨一下该类问题,分析一下此类问题的一些算法,也使有兴趣的同学对此问题有初步了解.

在现实生活中,经常需要用到两点间的最短距离. 求两点间的最短路及最短距圈离,常用的为 Dijkstra 算法与 Warshall-Ford 算法.

### 4.2.1 Dijkstra 算法与动态规划

动态规划算法已经成为了许多求解的重要算法,只不过在很多题目中动态规划的表达式比较难写,而恰恰图论中求某两点的最短路问题如果用动态规划算法考虑则可以非常容易地找到那个表达式. 在图论问题中,实现动态规划求解某两点的最短路问题的算法即 Dijkstra 算法.

**例 4.1** 求图 4.2 中从 $P_1$ 到 $P_5$ 的最短路.

记 $d(Y,X)$ 为城市 $Y$ 与城市 $X$ 之间的直接距离(若这两个城市之间没有道路直接相连,则可以认为直接距离为 $+\infty$),用 $L(X)$ 表示城市 $P_1$ 到城市 $X$ 的最短路线的

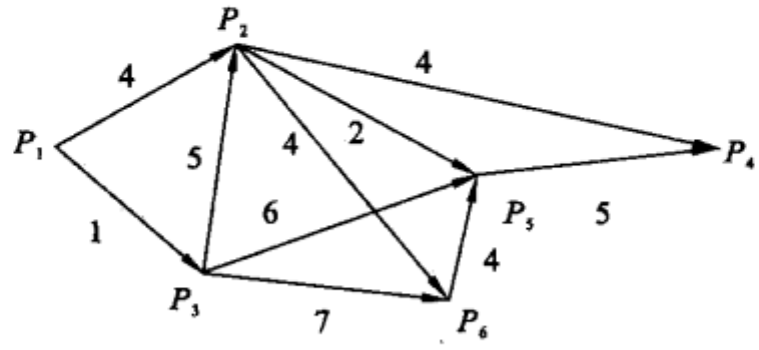


图 4.2　求 $P_1$ 到 $P_5$ 的最短路

路长，则模型可写为

$$\begin{cases} L(P_1)=0 \\ L(X)=\min\limits_{Y\neq X}\{L(Y)+d(Y,X)\}, \quad X\neq P_1 \end{cases}$$

算法适用的条件和范围：

(1) 求图 $G$ 中某一点到其他各点最短路，一般用 Dijkstra 算法.

(2) 有向图和无向图(无向图可以视为$(u,v)$,$(v,u)$同属于边集 $E$ 的有向图)均适用.

(3) 所有边权非负(任取$(i,j)\in E$ 都有 $d_{ij}\geqslant 0$).

利用 Dijkstra 算法我们也可以写出最短路问题的程序代码. 由于程序源码过长，读者请登陆华中数学建模网 http://www.shumo.cn/大学数学实验栏目下载学习.

## 4.2.2　Warshall-Ford 算法

Warshall-Ford 算法是求任意两顶点间最短距离的算法.

设 $\boldsymbol{A}=(a_{ij})_{n\times n}$ 为赋权图 $G=(V,E,F)$ 的权矩阵，当 $v_iv_j\in E$ 时，$a_{ij}=F(v_iv_j)$，否则取 $a_{ii}=0$，$a_{ij}=+\infty$ $(i\neq j)$，$d_{ij}$ 表示从 $v_i$ 到 $v_j$ 点的距离，$r_{ij}$ 表示从 $v_i$ 到 $v_j$ 点的最短路中一个点的编号.

① 赋初值. 对所有 $i,j$，$d_{ij}=a_{ij}$，$r_{ij}=j$，$k=1$. 转向②.

② 更新 $d_{ij}$，$r_{ij}$，对所有 $i,j$，若 $d_{ik}+d_{kj}<d_{ij}$，则令 $d_{ij}=d_{ik}+d_{kj}$，$r_{ij}=k$，转向③.

③ 终止判断. 若 $d_{ii}<0$，则存在一条含有顶点 $v_i$ 的负回路，终止；或者 $k=n$ 终止；否则令 $k=k+1$，转向②.

其中最短路线可由 $r_{ij}$ 得到.

该算法的适用条件和范围：①任意两点间的最短路径；②可以适用于有负权的情况.

算法较简单，适用范围广，实践中的效果不错，虽然复杂性略高(时间复杂度 $O(n^3)$)，仍不失为一个很实用的算法.

下面用一个例子来看看 Floyd 最短路算法的 MATLAB 程序：

**例 4.2**　求图 4.3 中任意两点间的最短路.

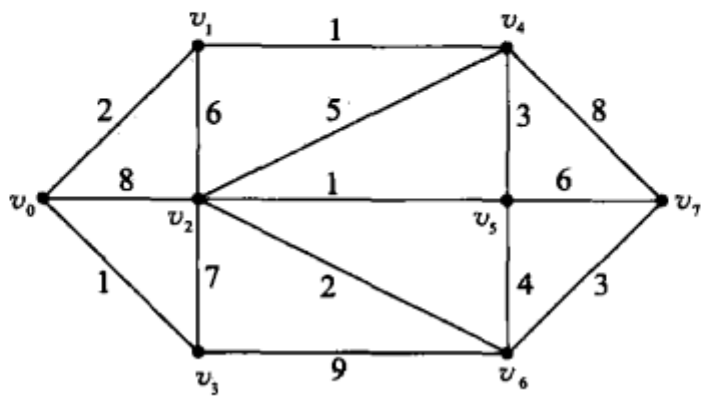


图 4.3　8 个点的连通赋权图

用 Warshall-Floyd 算法，MATLAB 程序代码如下：

```
        %floyd.m
function[D,R]=floydwarshall(A)
        %采用 floyd 算法计算图中任意两点之间最短路程,可以有负权
        %参数 D 为连通图的权矩阵
        %R 是路线矩阵
D=A;n=length(D);  %赋初值
for(i=1:n)for(j=1:n)R(i,j)=j;end;end  %赋路径初值
for(k=1:n)
    for(i=1:n)
        for(j=1:n)
    if(D(i,k)+D(k,j)<D(i,j))D(i,j)=D(i,k)+D(k,j);
%更新 dij,说明通过 k 的路程更短
    R(i,j)=k;end;end;end  %更新 rij,需要通过 k
    k  %显示迭代步数
    D  %显示每步迭代后的路长
    R  %显示每步迭代后的路径
    pd=0;for i=1:n%含有负权时
        if(D(i,i)<0)pd=1;break;end;end
%跳出内层的 for 循环,存在一条含有顶点 vi 的负回路
    if(pd==1)  fprintf('有负回路');break;end
%存在一条负回路,跳出最外层循环,终止程序
end  %程序结束
```

在命令窗口输入：

```
n=8;
A=[  0    2    8    1    Inf   Inf   Inf   Inf
     2    0    6    Inf  1     Inf   Inf   Inf
     8    6    0    7    5     1     2     Inf
     1    Inf  7    0    Inf   Inf   9     Inf
```

```
    Inf   1    5    Inf  0     3     Inf   8
    Inf   Inf  1    Inf  3     0     4     6
    Inf   Inf  2    9    Inf   4     0     3
     Inf  Inf  Inf  Inf  8     6     3     0];  %MATLAB 中,Inf 表示+∞
  [D,R]=floydwarshall(A)
```

运行结果(部分)如下:

```
D =
     0     2     7     1     3     6     9    11
     2     0     5     3     1     4     7     9
     7     5     0     7     4     1     2     5
     1     3     7     0     4     7     9    12
     3     1     4     4     0     3     6     8
     6     4     1     7     3     0     3     6
     9     7     2     9     6     3     0     3
    11     9     5    12     8     6     3     0
```

其中 D 矩阵为任意两点间的最短距离.

**评论** Dijkstra 算法与 Warshall-Ford 算法作为求解点与点之间最短距离及相应路径的算法,具有简单易懂,运行速度快的特点,在使用时要注意两者的区别,根据实际需要进行算法的选择.

## 4.3 最小生成树

**定义 4.4** 连通而无圈的图称为树(tree),常用 $T$ 表示树.

设 $G=(V,E)$是一个无向连通赋权图. $E$ 中每条边$(v_i,v_j)$的权为 $a_{ij}$. 如果 $G$ 的一个子图 $G'$是一棵包含 $G$ 所有顶点的树,则称 $G'$为 $G$ 的生成树. 或者可以由树的定义认为,任意一个连通的$(p,q)$图 $G$ 适当去掉 $q-p+1$ 条边后,都可以变成树,这棵树称为图 $G$ 的生成树.

关于树的一些结论:

(1) 设 $G$ 是连通图,且边数小于顶点数,则 $G$ 中至少有一个顶点的度为 1.

(2) 设 $G$ 是具有 $n$ 个顶点的无向连通图,$G$ 是树的充分必要条件是:$G$ 有 $n-1$ 条边.

最小生成树问题:设 $T$ 是图 $G$ 的一棵生成树,用 $F(T)$表示树 $T$ 中所有边的权数之和,$F(T)$称为该生成树 $T$ 的权(费用). 一个连通图 $G$ 的生成树一般不止一棵,在 $G$ 的所有生成树中,权数最小的生成树称为 $G$ 的最小生成树.

网络的最小生成树在实际中有广泛应用. 例如,在设计通信网络时,用图的顶点表示城市,用边$(v_i,v_j)$的权 $a_{ij}$ 表示建立城市 $v_i$ 和城市 $v_j$ 之间的通信线路所需的费用,则最小生成树就给出了建立通信网络最经济的方案.

最小生成树问题的数学表示如下(0-1 规划模型)：

$$\min z(y) = \sum_{(i,j)\in A} d_{ij}x_{ij}$$

$$\text{s.t.} \quad \sum_{j\in V} x_{1j} \geqslant 1 \quad \text{（根至少有一条边连接到其他点）}$$

$$\sum_{j\in V} x_{ij} = 1, \quad i \neq 1 \quad \text{（除根以外，每个点只有一条边进入）}$$

$$x_{ij} = 0 \text{ 或 } 1, \quad i,j \in \mathbf{N} \quad \text{（决策变量，} x_{ij} = 1 \text{ 表示连接）}$$

对一般问题，用贪婪算法得到的只是近似最优解，而不能保证得到最优解. 但用贪婪方法计算最小生成树，却可以设计出保证得到最优解的算法. 下面我们介绍的 Kruskal 避圈法就是一个基于贪婪算法的最小生成树算法.

Kruskal 避圈法是 Kruskal 在 1956 年提出的最小生成树算法，它的思路很容易理解. Kruskal 算法每次选择 $n-1$ 条边，所使用的贪婪准则是：从剩下的边中选择一条不会产生回路的具有最小权重的边加入已选择的边的集合中. 注意到所选取的边若产生回路则不可能形成一棵生成树. 设 $G=(V,E)$ 是一个连通赋权图，$V=\{1,2,\cdots,n\}$. 避圈法将图 $G$ 中的边按权数从小到大逐条考察，按不构成圈的原则加入到 $T$ 中(若有选择时，不同的选择可能会导致最后生成树的权数不同)，直到 $q(T)=n-1$ 为止，即 $T$ 的边数=$G$ 的顶点数减 1 为止.

Kruskal 避圈法(以例 4.2 为例)的 MATLAB 程序代码如下：

```
n=8;A=[0 2 8 1 0 0 0 0; 2 0 6 0 1 0 0 0;8 6 0 7 5 1 2 0;1 0 7 0 0 0 9 0;
0 1 5 0 0 3 0 8;0 0 1 0 3 0 4 6;0 0 2 9 0 4 0 3;0 0 0 0 8 6 3 0];
k=1;  %记录 A 中不同正数的个数
for(i=1:n-1)for(j=i+1:n)  %此循环是查找 A 中所有不同的正数
if(A(i,j)>0)x(k)=A(i,j);  %数组 x 记录 A 中不同的正数
kk=1;  %临时变量
for(s=1:k-1)if(x(k)==x(s))kk=0;break;end;end  %排除相同的正数
k=k+kk;end;end;end
k=k-1  %显示 A 中所有不同正数的个数
for(i=1:k-1)for(j=i+1:k)  %将 x 中不同的正数从小到大排序
if(x(j)<x(i))xx=x(j);x(j)=x(i);x(i)=xx;end;end;end
T(n,n)=0;  %将矩阵 T 中所有的元素赋值为 0
q=0;  %记录加入到树 T 中的边数
for(s=1:k)if(q==n)break;end  %获得最小生成树 T,算法终止
for(i=1:n-1)for(j=i+1:n)if(A(i,j)==x(s))T(i,j)=x(s);T(j,i)=x(s);
%加入边到树 T 中
TT=T;  %临时记录 T
while(1)pd=1;  %砍掉 TT 中所有的树枝
for(y=1:n)kk=0;
for(z=1:n)if(TT(y,z)>0)kk=kk+1;zz=z;end;end  %寻找 TT 中的树枝
```

```
if(kk==1)TT(y,zz)=0;TT(zz,y)=0;pd=0;end;end  %砍掉 TT 中的树枝
if(pd)break;end;end  %已砍掉了 TT 中所有的树枝
pd=0;  %判断 TT 中是否有圈
for(y=1:n-1)for(z=y+1:n)if(TT(y,z)>0)pd=1;break;end;end;end
if(pd)T(i,j)=0;T(j,i)=0;  %假如 TT 中有圈
else q=q+1;end;end;end;end;end
%显示近似最小生成树 T,程序结束
```

另外 Prim 在 1957 年提出另一种最小生成树算法——Prim 算法，这种算法特别适用于边数相对较多，即比较接近于完全图的图. 此算法是按逐个将顶点连通的步骤进行的，它只需采用一个顶点集合. 这个集合开始时是空集，以后将已连通的顶点陆续加入到集合中去，到全部顶点都加入到集合中了，就得到所需的生成树.

设 $G=(V,E)$是一个连通赋权图，$V=\{1,2,\cdots,n\}$. 构造 $G$ 的一棵最小生成树的 Prim 算法的过程是：首先从图的任一顶点起进行，将它加入集合 $S$ 中，$S=\{1\}$，然后作如下的贪婪选择，从与之相关联的边中选出权值 $a_{ij}$ 最小的一条作为生成树的一条边，此时满足条件 $i\in S, j\in V-S$，并将该 $j$ 加入集合 $S$ 中，表示这两个顶点已被所选出的边连通了. 如此进行下去，直到全部顶点都加入到集合 $S$ 中. 在这个过程中选取到的所有边恰好构成 $G$ 的一棵最小生成树.

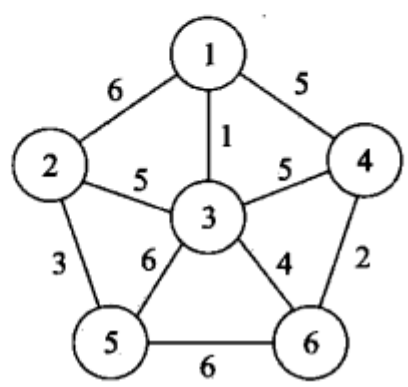

图 4.4　6 个点的连通赋权图

由于 Prim 算法中每次选取的边两端总是一个已连通顶点和一个未连通顶点，故这个边选取后一定能将该未连通点连通而又保证不会形成回路. 例如，对于图 4.4 中的赋权图，按 Prim 算法选取边的过程如图 4.5 所示.

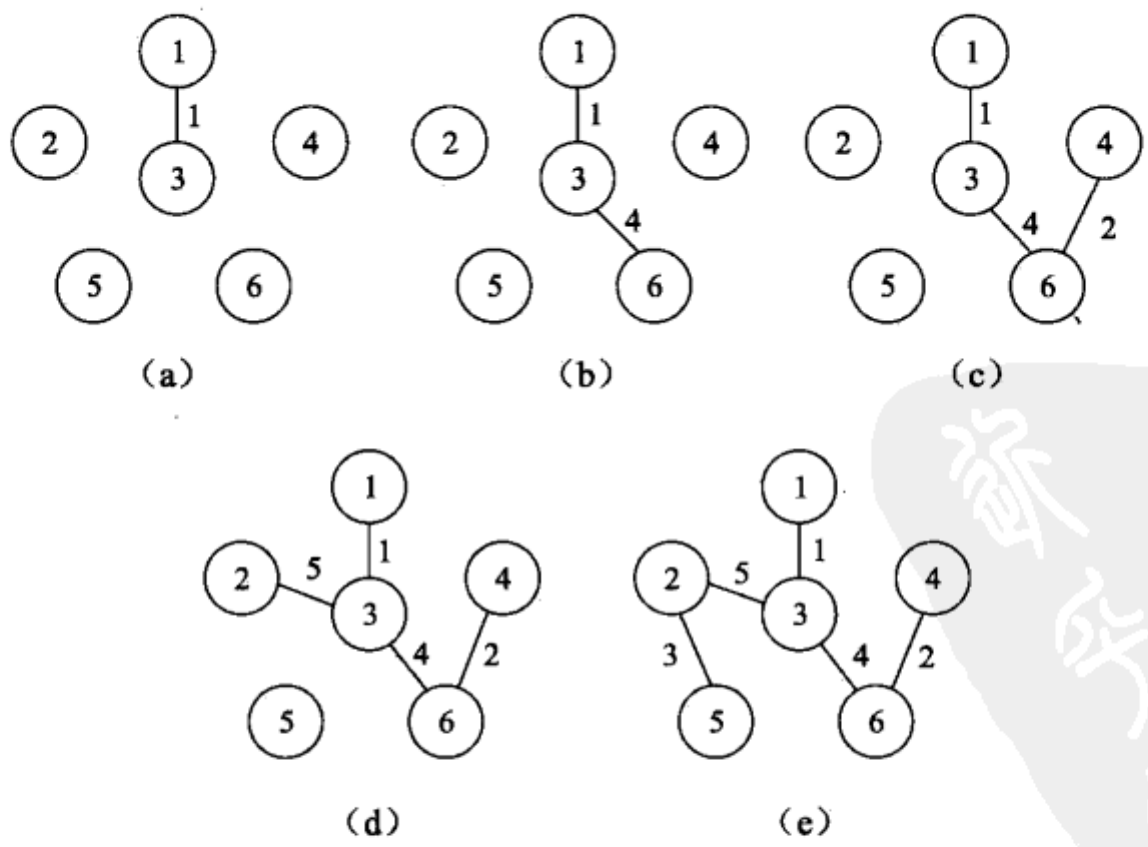

图 4.5　按 Prim 算法选取边的步骤

Prim 算法程序如下：

```
function [T,c]=tree_prim(a,n)
T=[];c=0;v=1;R=2:n;
for j=2:n
      b(1,j-1)=1;
      b(2,j-1)=j;
      b(3,j-1)=a(1,j);
end

while size(T,2)<n-1
     [q,i]= min(b(3,:));
     T(:,size(T,2)+1)=b(:,i);
     c=c+b(3,i);
     v=b(2,i);
     temp=find(R==b(2,i));
     R(temp)=[];b(:,i)=[];
     for j=1:length(R)
          d=a(v,b(2,j));
          if d<b(3,j)
               b(1,j)=v;b(3,j)=d;
          end
     end
end
```

**评论**　最小生成树的使用是非常广泛的，具有计算复杂度低的优点.

## 4.4　TSP 问题

下面讨论一下图论中著名的 TSP 问题.

我们以一个简单的例题来介绍这类问题的常用解法：

**例 4.3**　（旅行推销员问题(travel salesman problem ，简称 TSP 问题)，又称货郎担问题、推销员问题、旅行商问题等)设 $A,B,C,D,E$ 5 个城市($n=5$)之间的路程由如下距离矩阵 $\boldsymbol{W}$ 给出. 假设一个推销员想从城市 $A$ 出发，问是否存在一个行程安排，使得他能不重复地遍历所有城市后回到这个城市，而且所走路程最短. 试编程输出访问城市次序.

$$\boldsymbol{W}=\begin{bmatrix} \infty & 14 & 2 & 16 & 2 \\ 14 & \infty & 25 & 1 & 3 \\ 2 & 25 & \infty & 9 & 9 \\ 16 & 1 & 9 & \infty & 6 \\ 2 & 3 & 9 & 6 & \infty \end{bmatrix}$$

旅行推销员问题，用图论来说即“已给一个 $n$ 个点的完全图，每条边都有一个长度 $d_{ij}$，求总长度最短的经过每个顶点正好一次的一条封闭回路. 这个问题可分为对称旅行商问题($d_{ij}=d_{ji}$，$\forall i,j=1,2,\cdots,n$)和非对称旅行商问题($d_{ij}\neq d_{ji}$，$\forall i,j=1,2,\cdots,n$). 旅行商问题的数学模型为

$$\min z=\sum_{(i,j)\in A} d_{ij}x_{ij}$$

$$\text{s.t.}\quad \sum_{j\in V} x_{1j}\geqslant 1 \quad \text{（根至少有一条边连接到其他点）}$$

$$\sum_{i\in V} x_{ij}=1,\quad j\neq 1 \quad \text{（除根以外，每个点只有一条边进入）}$$

$$x_{ij}=0 \text{ 或 } 1,\quad i,j\in \mathbf{N}$$

旅行商问题是一个典型的组合优化问题，TSP 问题显然是 NP 问题. 因为如果任意给出一个行程安排，可以很容易算出旅行总路程. 但是，要想知道一条总路程最小的行程是否存在，在最坏情况下，必须检查所有可能的旅行安排. 随着 $n$ 的加大，可能的路径数目与城市数目 $n$ 是成指数型增长的，这将是个天文数字.

如果只有 3 个城市 $A$，$B$ 和 $C$，互相之间都有道路相连，而且起始城市是任意的，则有 6 种访问每个城市的次序：$ABC$，$ACB$，$BAC$，$BCA$，$CAB$，$CBA$. 如果有 4 个城市(图 4.6)，则有 24 种次序，可以用阶乘来表示：$4!=4\times 3!=4\times 3\times 2\times 1=24$. 如果有 5 个城市，则有 $5!=5\times 4!=120$，类似地，有 $6!=720$ 等. 即使用计算机来计算，这种急剧增长的可能性的数目也远远超过计算资源的处理能力.

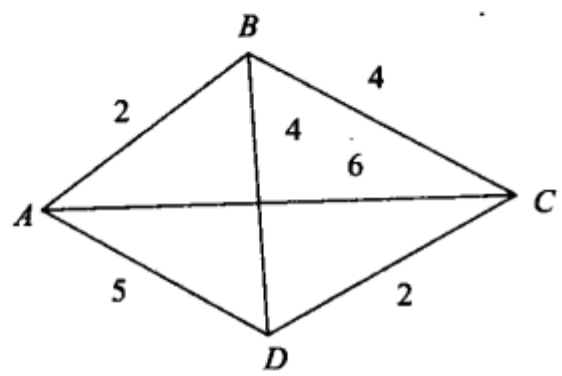

图 4.6　4 个城市的完全图

问题的关键是某些事情在实践中行不通. Cook 评论：“如果有 100 个城市，需要求出 100! 条路线的费用，没有哪一台计算机能够胜任这一任务. 打个比方，让太阳系中所有的电子以它旋转的频率来计算，就算太阳烧尽了也算不完.

1998 年，科学家们成功地解决了美国 13509 个城市之间的 TSP 问题，2001 年又解决了德国 15112 个城市之间的 TSP 问题. 但这一工程代价也是巨大的，共使用了美国 Rice 大学和 Princeton 大学之间网络互连的、由速度为 500 MHz 的 Compaq EV6 Alpha 处理器组成的 110 台计算机，所有计算机花费的时间之和为 22.6 年.

下面我们考虑用贪婪算法来处理这个问题：将城市间的距离从小到大排列有

$$d_{24}(1),d_{13}(2),d_{15}(2),d_{25}(3),d_{45}(6),d_{35}(9),d_{34}(9),d_{12}(14),d_{14}(16),d_{23}(25)$$

由于是 5 个城市，环绕一圈为 5 条边，贪婪算法求解此问题的过程是从最小边开始，依次从小到大取边加入到回路边集中，但在将 1 条边加入时不能使 1 顶点的度数超过 3，也不能形成小回路.

此问题解的过程如下：

$d_{24}(1)$；

$d_{24}(1)+d_{13}(2)$；

$d_{24}(1)+d_{13}(2)+d_{15}(2)$；

$d_{24}(1)+d_{13}(2)+d_{15}(2)+d_{25}(3)$；

$d_{24}(1)+d_{13}(2)+d_{15}(2)+d_{25}(3)+[d_{45}(6)]$；

（下标中 5 出现了 3 次，顶点 5 有三条边相连，$d_{35}(6)$放弃）

$d_{24}(1)+d_{13}(2)+d_{15}(2)+d_{25}(3)+[d_{35}(9)]$；

（下标中 5 出现了 3 次，顶点 5 有三条边相连，$d_{35}(6)$放弃）

$d_{24}(1)+d_{13}(2)+d_{15}(2)+d_{25}(3)+d_{34}(9)$.

得到一条回路 $v_1\to v_3\to v_4\to v_2\to v_5\to v_1$ 是最佳的回路.

**例 4.4** 设有 6 个城市，其坐标分别为 $a(0,0)$，$b(4,3)$，$c(1,7)$，$d(15,7)$，$e(15,4)$，$f(18,0)$. 6 个城市间的距离矩阵为

$$\boldsymbol{D}=\begin{bmatrix} \infty & 5 & 7.07 & 16.55 & 15.52 & 18 \\ 5 & \infty & 5 & 11.7 & 11.01 & 14.32 \\ 7.07 & 5 & \infty & 14 & 14.32 & 18.38 \\ 16.55 & 11.7 & 14 & \infty & 3 & 7.62 \\ 15.52 & 11.01 & 14.32 & 3 & \infty & 5 \\ 18 & 14.32 & 18.38 & 7.62 & 5 & \infty \end{bmatrix}$$

用贪婪算法，先将任两城市间的连线距离按从小到大的次序排列，然后从中逐个选择. 但有两种情况的连线应舍弃：①使任一城市的度数（连线数）超过 2 的连线必须舍弃；②在得到经过所有点的回路前就形成小回路的连线必须舍弃.

距离按从小到大的次序排列：

$D_{de}(3)$，　$D_{ab}(5)$，　$D_{bc}(5)$，

$D_{ef}(5)$，　$D_{ae}(7.07)$，　$D_{df}(7.62)$，

$D_{be}(11.01)$，　$D_{bd}(11.7)$，　$D_{ed}(14)$，

$D_{bf}(14.32)$，　$D_{ce}(14.32)$，　$D_{ae}(15.52)$，

$D_{ad}(14.55)$，　$D_{af}(18.38)$，　$D_{ef}(18.38)$

按贪婪算法原则，其选择过程如下：

$D_{de}$；

$D_{de}+D_{ab}$；

$D_{de}+D_{ab}+D_{bc}$；

$D_{de}+D_{ab}+D_{bc}+D_{ef}$；

$D_{de}+D_{ab}+D_{bc}+D_{ef}+[D_{ae}]$；　（形成小回路，舍弃）

$D_{de}+D_{ab}+D_{bc}+D_{ef}+[D_{df}]$；　（形成小回路，舍弃）

$D_{de}+D_{ab}+D_{bc}+D_{ef}+[D_{be}]$；　（$b$ 顶点度数超过 2，舍弃）

$D_{de}+D_{ab}+D_{bc}+D_{ef}+[D_{bd}]$； （$b$ 顶点度数超过 2，舍弃）

$D_{de}+D_{ab}+D_{bc}+D_{ef}+D_{cd}$；

$D_{de}+D_{ab}+D_{bc}+D_{ef}+D_{cd}+[D_{bf}]$； （$b$ 顶点度数超过 2，舍弃）

$D_{de}+D_{ab}+D_{bc}+D_{ef}+D_{cd}+[D_{ce}]$； （$c$，$e$ 顶点度数超过 2，舍弃）

$D_{de}+D_{ab}+D_{bc}+D_{ef}+D_{cd}+[D_{ae}]$； （$e$ 顶点度数超过 2，舍弃）

$D_{de}+D_{ab}+D_{bc}+D_{ef}+D_{cd}+[D_{ae}]$； （$e$ 顶点度数超过 2，舍弃）

$D_{de}+D_{ab}+D_{bc}+D_{ef}+D_{cd}+[D_{ad}]$； （$d$ 顶点度数超过 2，舍弃）

$D_{de}+D_{ab}+D_{bc}+D_{ef}+D_{cd}+D_{af}$； 得到 1 条回路

最后得到的回路如图 4.7 所示的结果，总长度为 50. 不过，这不是此问题的最优解，此问题的最优解为图 4.8 所示的路径（可以用分枝定界等方法求得），总长度为 48.39. 用贪婪算法得到的结果同最优解相比只多了 3.3%.

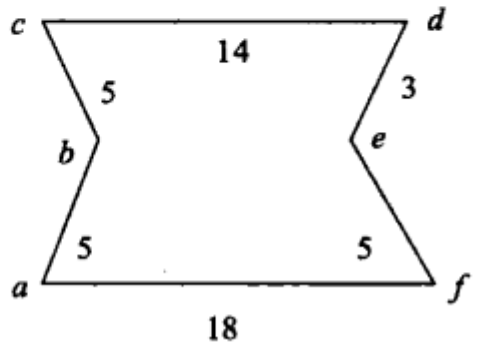


图 4.7 按贪婪算法所得的回路

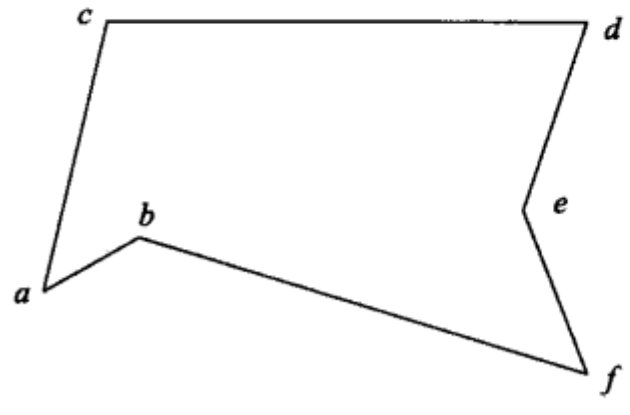


图 4.8 最优回路

TSP 问题在图论上被称为哈密顿回路问题，即指找从一点出发不重复地走过所有的结点，最后又回到原出发点的路径的问题，我们下面还跟大家介绍欧拉回路问题，它是指从一点出发不重复地走过所有的边，最后又回到原出发点的路径的问题. 两个问题的不同点在于：哈密顿回路问题是访问每个结点一次，而欧拉回路问题是访问每条边一次. 对图 $G$ 是否存在欧拉回路，欧拉已给出充分必要条件，而对图 $G$ 是否存在哈密顿回路至今仍未找到满足该问题的充分必要条件. 欧拉回路问题也称为中国邮递员问题. 中国数学家管梅谷于 1960 年首先研究并给出算法.

中国邮递员问题（著名图论问题之一）：邮递员从邮局出发送信，要求对辖区内每条街，都至少通过一次，再回邮局. 在此条件下，怎样选择一条最短路线？

算法：找总权数最小的欧拉回路，遍历图.

欧拉回路与欧拉图：通过图 $G$ 的每条边一次仅且一次，而且走遍每个结点的通路（回路），就是欧拉通路（回路）. 存在欧拉回路的图就是欧拉图. 欧拉回路要求边不能重复，结点可以重复. 笔不离开纸，不重复地走完所有的边，且走过所有结点，就是所谓的一笔画. 欧拉图的所有顶点度数为偶数，且为连通图.

欧拉图或通路的判定：

(1) 无向连通图 $G$ 是欧拉图，$G$ 不含奇数度结点($G$ 的所有结点度数为偶数).

(2) 非平凡连通图 $G$ 含有欧拉通路；$G$ 最多有两个奇数度的结点.

(3) 连通有向图 $D$ 含有有向欧拉回路；$D$ 中每个结点的入度等于出度.

连通有向图 $D$ 含有有向欧拉通路；$D$ 中除两个结点外，其余每个结点的入度等于出度，且此两点满足：一个端点入度比出度大 1；一个端点入度比出度小 1.

**例 4.5** (两只蚂蚁比赛问题)两只蚂蚁甲、乙分别处在图 4.9 中的顶点 $a$，$b$ 处，并设图中各边长度相等. 甲提出同乙比赛：从它们所在顶点出发，走过图中所有边最后到达顶点 $c$ 处. 如果它们速度相同，问谁最先到达目的地？

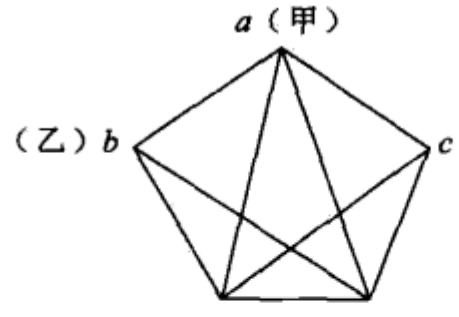


图 4.9 蚂蚁比赛

图 4.9 中有两个奇度顶点 $b$，$c$，因此存在从 $b$ 到 $c$ 的欧拉通路，蚂蚁乙走到 $c$ 只要走一条欧拉通路，边数为 9，而蚂蚁甲要想走完图中所有边到达 $c$，至少要先走一条边到达 $b$，再走一条欧拉通路，故它至少要走 10 条边到达 $c$，所以乙必胜.

**评论** 最短路径问题是现实生活中十分常见的问题，其难点在于普通算法计算复杂度太大而导致难以进行计算，故在做这类问题时，要充分考虑计算复杂度的问题，要在验证算法可行的情况下进行最优解的贪婪搜索.

## 4.5 着色问题

已知图 $G=(V,E)$，对图 $G$ 的所有顶点进行着色时，要求相邻的两顶点的颜色不一样，问至少需要几种颜色？这就是所谓的顶点着色问题.

若对图 $G$ 的所有边进行着色时，要求相邻的两条边的颜色不一样，问至少需要几种颜色？这就是所谓的边着色问题.

这些问题的提出是有实际背景的. 值得注意的是，着色模型中的图是无向图. 对于顶点着色问题，若是有限图，也可转化为有限简单图. 而边着色问题可以转化为顶点着色问题.

例如，物资储存问题：一家公司制造 $n$ 种化学制品 $A_1, A_2, \cdots, A_n$，其中有些化学制品若放在一起可能产生危险，如引发爆炸或产生毒气等，称这样的化学制品是不相容的. 为安全起见，在储存这些化学制品时，不相容的不能放在同一储存室内. 问至少需要多少个储存室才能存放这些化学制品？

作图 $G$，用顶点 $v_1, v_2, \cdots, v_n$ 分别表示 $n$ 种化学制品，顶点 $v_i$ 与 $v_j$ 相邻，当且仅当化学制品 $A_i$ 与 $A_j$ 不相容.

于是储存问题就化为对图 $G$ 的顶点着色问题，对图 $G$ 的顶点最少着色数目便是最少需要的储存室数.

又如，时间表问题：现有 $m$ 个工作人员 $x_1, x_2, \cdots, x_m$，操作 $n$ 种设备 $y_1, y_2, \cdots,$

$y_n$. 设工作人员 $x_i$ 使用设备 $y_j$ 的时间为 $a_{ij}$，假定使用的时间均以单位时间计算，矩阵 $\boldsymbol{A}=(a_{ij})_{n\times n}$ 称为工作要求矩阵. 假定每一个工作人员在同一时间只能使用一种设备，某一种设备在同一时间里只能为一个工作人员所使用. 问应如何合理安排，使得在尽可能短的时间里满足工作人员的要求？这就是要求在工作人员与设备之间找到一个对应. 在同一时间内，工作人员 $x_i$ 使用设备 $y_j$ 对应一条从 $x_i$ 到 $y_j$ 的边，问题变为对所得的二部图 $G$ 的边着色问题，有相同的颜色的边可以安排在同一时间里.

这些都可以转化为着色模型. 下面我们来介绍着色的方法.

对图 $G=(V,E)$ 的顶点进行着色所需最少的颜色数目用 $\chi(G)$ 表示，称为图 $G$ 的色数.

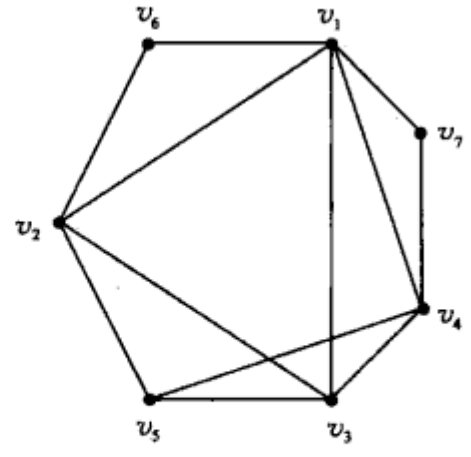

图 4.10　着色问题

**定理 4.1**　若图 $G=(V,E)$，$d=\max\{d(v)\mid v\in V\}$，则 $\chi(G)\leqslant d+1$.

这个定理给出了色数的上界. 着色算法目前还没有找到最优算法.

**例 4.6**　将图 4.10 中顶点进行着色，找到着色最少所需颜色数.

**模型建立**　引入 0-1 变量 $x_{ik}$，当 $v_i$ 着第 $k$ 种颜色时，$x_{ik}=1$；否则，$x_{ik}=0$. 设颜色种数为 $x$，建立如下模型：

$$\min x$$

$$\text{s. t.}\begin{cases}\sum\limits_{k=1}^{\Delta+1}x_{ik}=1, & i=1,2,\cdots,n\\ x_{ik}+x_{jk}\leqslant 1, & v_iv_j\in E\\ x\geqslant\sum\limits_{k=1}^{\Delta+1}kx_{ik}, & i=1,2,\cdots,n\\ x_{ik}=0\text{ 或 }1, & i=1,2,\cdots,n;k=1,2,\cdots,\Delta+1\\ x\geqslant 0\end{cases}$$

下面给出一种近似算法——最大度数优先的 Welsh-Powell 算法. 这个算法也是一个贪婪算法，算法给出了一个较好的着色方法，但不是最有效的方法，即所用的颜色数不一定是最少的，但在许多问题上，它还是有效的.

最大度数优先的 Welsh-Powell 算法如下：

设 $G=(V,E)$，$V=\{v_1,v_2,\cdots,v_n\}$，且不妨假设 $d(v_1)\geqslant d(v_2)\geqslant\cdots\geqslant d(v_n)$，$c_1$，$c_2,\cdots,c_n$ 为 $n$ 种不同的颜色.

① 令有序集 $C_i=\{c_1,c_2,\cdots,c_n\}$ $(i=1,2,\cdots,n;j=1)$. 转向②.

② 给 $v_j$ 着 $C_j$ 的第一个颜色 $C_{j1}$. 当 $j=n$ 时，停；否则，转向③.

③ $\forall k>j$，若 $v_k$ 和 $v_j$ 相邻，则令 $C_k=C_k\backslash\{C_{j1}\}$. $j=j+1$，转向②.

利用 Welsh-Powell 算法，我们也可以写出上述问题的 MATLAB 程序：

```
function[a1,b1]=sortd(n,a,b)  %排序
[m,n]=size(a);
for h=1:n-1
  for j=1:n-h
   if a(j)<a(j+1)
       t1=a(j);a(j)=a(j+1);a(j+1)=t1;
      t2=b(j);b(j)=b(j+1);b(j+1)=t2;
   end
  end
end
a1=a;
b1=b;
function welsh(n,e)
totalcolor=0;  %e为邻接矩阵
temp=zeros(1,n);  %临时矩阵
for (i=1:n) temp=e(i,:);t(i)=sum(temp);temp=0;q(i)=i;c(i)=i;end
%t记录vi的度—邻接矩阵行和,q记录顶点标号,c:颜色集
[t,q]=sortd(n,t,q);  %按度排序
[m,n]=size(e);
bepaint=zeros(m,n);  %初始化着色矩阵
for i=1:n bepaint(i,:)=c; end
i=1;
     for j=1:n  %顶点-行
        pcivj=min(bepaint(j,:));bepaint(j,i)=pcivj;  %求行最小值
          for k=j+1:n
              if(e(j,k)==1)bepaint(k,i)=bepaint(k,i)+inf;end
          end
           pcivj=0;i=i+1;
     end
bepaint
```

命令窗口输入：

```
e=[0 1 1 1 0 1 1
   1 0 1 0 1 1 0
   1 1 0 1 1 0 0
   1 0 1 0 1 0 1
   0 1 1 1 0 0 0
```

```
    1 1 0 0 0 0 0
    1 0 0 1 0 0 0];
welsh(7,e)
```

**评论** 着色问题也是现实生活中常见的一类问题，在使用时应当注意根据计算规模的大小选择合适的算法，在求解小规模问题时适宜采用线性规划模型进行全局最优解的求解，而在求解大规模问题时应当采用 Welsh-Powell 算法进行求解，以保证算法的可行性.

## 4.6 最大流问题

**定义 4.5** 设 $G=(V,E)$为有向图，在 $V$ 中指定一点称为发点(记为 $v_s$)，另一点称为收点(记为 $v_t$)，其余点称为中间点. 对每一条边 $v_iv_j \in E$，对应一个非负实数 $C_{ij}$，称为它的容量. 这样的 $G$ 称为容量网络，简称网络，记为 $G=(V,E,C)$.

**定义 4.6** 网络 $G=(V,E,C)$中任一条边 $v_iv_j$ 有流量 $f_{ij}$，称集合 $f=\{f_{ij}\}$为网络 $G$ 上的一个流.

满足下述条件的流 $f$ 称为可行流：

(1)(限制条件)对每一边 $v_iv_j$，有 $0 \leqslant f_{ij} \leqslant C_{ij}$.

(2)(平衡条件)对于中间点 $v_k$，有 $\sum f_{ik} = \sum f_{kj}$，即中间点 $v_k$ 的输入量等于输出量.

如果 $f$ 是可行流，则对收、发点 $v_t$，$v_s$ 有 $\sum f_{si} = \sum f_{jt} = W_f$，即从 $v_s$ 点发出的物质总量等于 $v_t$ 点输入的量. $W_f$ 称为网络流 $f$ 的总流量.

上述概念可以这样来理解，如 $G$ 是一个运输网络，则发点 $v_s$ 表示发送站，收点 $v_t$ 表示接收站，中间点 $v_k$ 表示中间转运站，可行流 $f_{ij}$ 表示某条运输线上通过的运输量，容量 $C_{ij}$ 表示某条运输线能承担的最大运输量，$W_f$ 表示运输总量.

可行流总是存在的，如所有边的流量 $f_{ij}=0$ 就是一个可行流(称为零流).

所谓最大流问题就是在容量网络中，寻找流量最大的可行流.

实际问题中，一个网络会出现下面两种情况：

(1) 发点和收点都不止一个. 解决的方法是再虚设一个发点 $v_s$ 和一个收点 $v_t$，发点 $v_s$ 到所有原发点边的容量都设为无穷大，所有原收点到收点 $v_t$ 边的容量都设为无穷大.

(2) 网络中除了边有容量外，点也有容量. 解决的方法是将所有有容量的点分成两个点，如点 $v$ 有容量 $C_v$，将点 $v$ 分成两个点 $v'$ 和 $v''$，令 $C(v'v'')=C_v$.

最大流问题的数学模型表示如下：

$$\max\ z = v_f$$

$$\text{s.t.}\quad \sum_{\substack{j\in V\\(i,j)\in A}} f_{ij} - \sum_{\substack{j\in V\\(j,i)\in A}} f_{ji} = \begin{cases} v_f, & i = s \\ -v_f, & i = t \\ 0, & i \neq s,t \end{cases}$$

$$0 \leqslant f_{ij} \leqslant C_{ij},\quad (i,j) \in A$$

**例 4.7** 现需要将城市 $s$ 的石油通过管道运送到城市 $t$，中间有 4 个中转站 $v_1$，$v_2$，$v_3$，$v_4$，城市与中转站的连接以及管道的容量如图 4.11 所示，求从城市 $s$ 到城市 $t$ 的总的最大运输量.

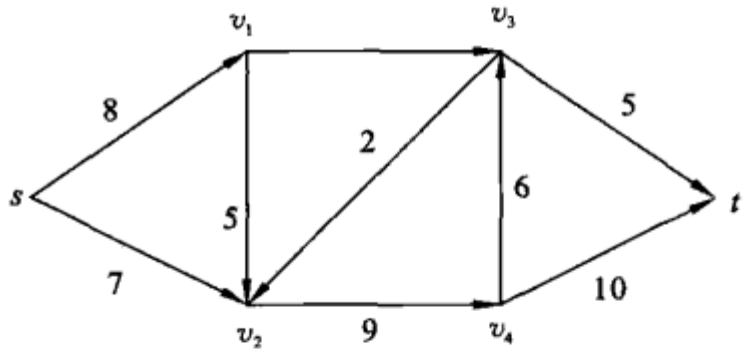


图 4.11 管道运输

最大流的 Ford-Fulkerson 标号算法如下：

(1) 标号过程.

① 给发点 $v_s$ 以标号$(+,\delta_s)$，$\delta_s=+\infty$.

② 选择一个已标号的点 $x$，对于 $x$ 的所有未给标号的邻接点 $y$，按下列规则处理：

当 $yx\in E$ 且 $f_{yx}>0$ 时，令 $\delta_y=\min\{f_{yx},\delta_x\}$，并给 $y$ 以标号$(x-,\delta_y)$；

当 $xy\in E$ 且 $f_{xy}<C_{xy}$ 时，令 $\delta_y=\min\{C_{xy}-f_{xy},\delta_x\}$，并给 $y$ 以标号$(x+,\delta_y)$.

③ 重复②直到收点 $v_t$ 被标号或不再有点可标号时为止. 若 $v_t$ 得到标号，说明存在一条可增广链，转(2) 调整过程；若 $v_t$ 未得到标号，标号过程已无法进行时，说明 $f$ 已经是最大流.

(2) 调整过程.

④ 决定调整量 $\delta=\delta_{vt}$，令 $u=v_t$.

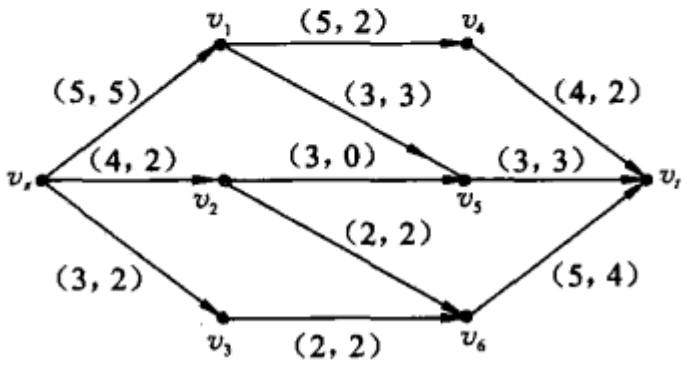


图 4.12 求所示网络最大流

⑤ 若 $u$ 点标号为$(v+,\delta_u)$，则以 $f_{vu}+\delta$ 代替 $f_{vu}$；若 $u$ 点标号为$(v-,\delta_u)$，则以 $f_{vu}-\delta$ 代替 $f_{vu}$.

⑥ 若 $v=v_s$，则去掉所有标号转(1)重新标号；否则令 $u=v$，转⑤.

算法终止后，令已有标号的点集为 $S$，则割集$(S,S^c)$为最小割，从而 $W_f=C(S,S^c)$.

**例 4.8** 求图 4.12 所示网络的最大流.

利用 Ford-Fulkerson 标号法求最大流算法的 MATLAB 程序代码如下：

```
n=8;C=[0  5  4  3  0  0  0  0
       0  0  0  0  5  3  0  0
       0  0  0  0  0  3  2  0
```

```
        0   0  0  0  0  0  2  0
        0   0  0  0  0  0  0  4
        0   0  0  0  0  0  0  3
        0   0  0  0  0  0  0  5
        0   0  0  0  0  0  0  0];  %弧容量
for(i=1:n)for(j=1:n)f(i,j)=0;end;end  %取初始可行流 f 为零流
for(i=1:n)No(i)=0;d(i)=0;end  %No,d 记录标号
while(1)
    No(1)=n+1;d(1)=Inf;  %给发点 vs 标号
    while(1)pd=1;  %标号过程
        for(i=1:n)if(No(i))  %选择一个已标号的点 vi
            for(j=1:n)if(No(j)==0&f(i,j)<C(i,j))
            %对于未给标号的点 vj,当 vivj 为非饱和弧时
                No(j)=i;d(j)=C(i,j)-f(i,j);pd=0;
                if(d(j)>d(i))d(j)=d(i);end
            elseif(No(j)==0&f(j,i)>0)
            %对于未给标号的点 vj,当 vjvi 为非零流弧时
                No(j)=-i;d(j)=f(j,i);pd=0;
                if(d(j)>d(i))d(j)=d(i);end;end;end;end;end
        if(No(n)|pd)break;end;end  %若收点 vt 得到标号或者无法标号,终止标号过程
        if(pd)break;end  %vt 未得到标号,f 已是最大流,算法终止
        dvt=d(n);t=n;  %进入调整过程,dvt 表示调整量
        while(1)
            if(No(t)>0)f(No(t),t)=f(No(t),t)+dvt;  %前向弧调整
            elseif(No(t)<0)f(No(t),t)=f(No(t),t)-dvt;end  %后向弧调整
            if(No(t)==1)for(i=1:n)No(i)=0;d(i)=0;end;break;end
            %当 t 的标号为 vs 时,终止调整过程
            t=No(t);end;end;  %继续调整前一段弧上的流 f
wf=0;for(j=1:n)wf=wf+f(1,j);end  %计算最大流量
f  %显示最大流
wf  %显示最大流量
No  %显示标号,由此可得最小割,程序结束
```

**评论** 在处理结点个数较少的问题时,可采用线性规划模型,在解决节点数目较多的问题时,则应当采用 Ford-Fulkerson 标号算法.

## 4.7 最小费用流问题

这里我们要进一步探讨不仅要使网上的流达到最大,或者达到要求的预定值,而

且还要使运输流的费用是最小的，这就是最小费用流问题.

最小费用流问题的一般提法：已知网络$G=(V,E,C)$，每条边$v_iv_j\in E$除了已给容量$C_{ij}$外，还给出了单位流量的费用$b_{ij}(\geqslant 0)$. 所谓最小费用流问题就是求一个总流量已知的可行流$f=\{f_{ij}\}$使得总费用$b(f)=\sum\limits_{v_iv_j\in E}b_{ij}f_{ij}$最小.

特别地，当要求$f$为最大流时，此问题即为最小费用最大流问题.

最小费用流问题的数学表示如下：

$$\min z=\sum_{(i,j)\in A}b_{ij}f_{ij}$$

$$\text{s.t.}\quad \sum_{\substack{j\in V\\(i,j)\in A}}f_{ij}-\sum_{\substack{j\in V\\(j,i)\in A}}f_{ji}=\begin{cases}v_f, & i=s\\ -v_f, & i=t\\ 0, & i\neq s,t\end{cases}$$

$$0\leqslant f_{ij}\leqslant C_{ij},\quad (i,j)\in A$$

(1) 设网络$G=(V,E,C)$，取初始可行流$f$为零流，求解最小费用流问题的迭代步骤如下.

① 构造有向赋权图$G_f=(V,E_f,F)$，对于任意的$v_iv_j\in E$，$E_f$，$F$的定义如下：

当$f_{ij}=0$时，$v_iv_j\in E_f$，$F(v_iv_j)=b_{ij}$；

当$f_{ij}=C_{ij}$时，$v_jv_i\in E_f$，$F(v_jv_i)=-b_{ij}$；

当$0<f_{ij}<C_{ij}$时，$v_iv_j\in E_f$，$F(v_iv_j)=b_{ij}$，$v_jv_i\in E_f$，$F(v_jv_i)=-b_{ij}$. 然后转向②.

② 求出有向赋权图$G_f=(V,E_f,F)$中发点$v_s$到收点$v_t$的最短路$\mu$，若最短路$\mu$存在，转向③；否则$f$是所求的最小费用最大流，停止.

③ 增流. 同求最大流的方法一样，重述如下：

令$\delta_{ij}=\begin{cases}C_{ij}-f_{ij}, & v_iv_j\in\mu^+\\ f_{ij}, & v_iv_j\in\mu^-\end{cases}$，$\delta=\min\{\delta_{ij}\mid v_iv_j\in\mu\}$，重新定义流$f=\{f_{ij}\}$为

$$f_{ij}=\begin{cases}f_{ij}+\delta, & v_iv_j\in\mu^+\\ f_{ij}-\delta, & v_iv_j\in\mu^-\\ f_{ij}, & \text{其他}\end{cases}$$

如果$W_f$大于或等于预定的流量值，则适当减少$\delta$值，使$W_f$等于预定的流量值，那么$f$是所求的最小费用流，停止；否则转向①.

(3) 求解含有负权的有向赋权图$G=(V,E,F)$中某一点到其他各点最短路的Ford算法.

当$v_iv_j\in E$时，记$w_{ij}=F(v_iv_j)$，否则取$w_{ii}=0$，$w_{ij}=+\infty(i\neq j)$. $v_1$到$v_i$的最短路长记为$\pi(i)$，$v_1$到$v_i$的最短路中$v_i$的前一个点记为$\theta(i)$. Ford算法的迭代步骤：

① 赋初值$\pi(1)=0$，$\pi(i)=+\infty$，$\theta(i)=i$ $(i=2,3,\cdots,n)$.

② 更新 $\pi(i),\theta(i)$. 对于 $i=2,3,\cdots,n$ 和 $j=1,2,\cdots,n$，如果 $\pi(i)<\pi(j)+w_{ji}$，则令 $\pi(i)=\pi(j),\theta(i)=j$.

③ 终止判断. 若所有的 $\pi(i)$都无变化，停止；否则转向②.

在算法的每一步中，$\pi(i)$都是从 $v_1$ 到 $v_i$ 的最短路长度的上界. 若不存在负长回路，则从 $v_1$ 到 $v_i$ 的最短路长度是 $\pi(i)$ 的下界，经过 $n-1$ 次迭代后 $\pi(i)$ 将保持不变. 若在第 $n$ 次迭代后 $\pi(i)$ 仍在变化时，说明存在负长回路.

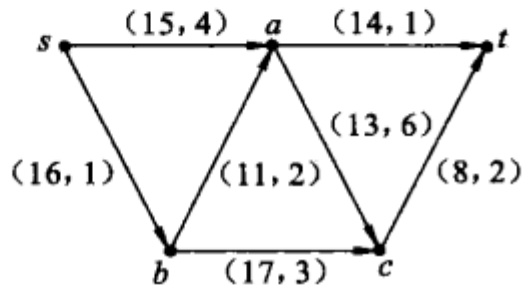

图 4.13　求 $s$ 到 $t$ 的最小费用最大流

**例 4.9**　在图 4.13 所示运输网络上，求 $s$ 到 $t$ 的最小费用最大流，括号内为$(C_{ij},b_{ij})$.

求最小费用最大流算法的 MATLAB 程序代码如下：

```
n=5;C=[0    15   16   0   0
       0    0    0    13  14
       0    11   0    17  0
       0    0    0    0   8
       0    0    0    0   0];  %弧容量
b=[0    4    1    0    0
   0    0    0    6    1
   0    2    0    3    0
   0    0    0    0    2
   0    0    0    0    0];  %弧上单位流量的费用
wf=0;wf0=Inf;  %wf 表示最大流量,wf0 表示预定的流量值
for(i=1:n)for(j=1:n)f(i,j)=0;end;end  %取初始可行流 f 为零流
while(1)
    for(i=1:n)for(j=1:n)if(j~=i)a(i,j)=Inf;end;end;end  %构造有向赋权图
    for(i=1:n)for(j=1:n)if(C(i,j)>0&f(i,j)==0)a(i,j)=b(i,j);
        elseif(C(i,j)>0&f(i,j)==C(i,j))a(j,i)=-b(i,j);
        elseif(C(i,j)>0)a(i,j)=b(i,j);a(j,i)=-b(i,j);end;end;end
    for(i=2:n)p(i)=Inf;s(i)=i;end  %用 Ford 算法求最短路,赋初值
    for(k=1:n)pd=1;  %求有向赋权图中 vs 到 vt 的最短路
        for(i=2:n)for(j=1:n)if(p(i)>p(j)+a(j,i))p(i)=p(j)+a(j,i);
          s(i)=j;pd=0;end;end;end
        if(pd)break;end;end  %求最短路的 Ford 算法结束
    if(p(n)==Inf)break;end
    %不存在 vs 到 vt 的最短路,算法终止.注意在求最小费用最大流时构造有向赋权图
     中不会含负权回路,所以不会出现 k=n
```

```
    dvt=Inf;t=n;  %进入调整过程,dvt表示调整量
    while(1)  %计算调整量
        if(a(s(t),t)>0)dvtt=C(s(t),t)-f(s(t),t);  %前向弧调整量
        elseif(a(s(t),t)<0)dvtt=f(t,s(t));end  %后向弧调整量
        if(dvt>dvtt)dvt=dvtt;end
        if(s(t)==1)break;end  %当t的标号为vs时,终止计算调整量
        t=s(t);end  %继续调整前一段弧上的流f
    pd=0;if(wf+dvt>=wf0)dvt=wf0-wf;pd=1;end
    %如果最大流量大于或等于预定的流量值
    t=n;while(1)  %调整过程
        if(a(s(t),t)>0)f(s(t),t)=f(s(t),t)+dvt;  %前向弧调整
        elseif(a(s(t),t)<0)f(t,s(t))=f(t,s(t))-dvt;end  %后向弧调整
        if(s(t)==1)break;end  %当t的标号为vs时,终止调整过程
        t=s(t);end
    if(pd)break;end  %如果最大流量达到预定的流量值
    wf=0;for(j=1:n)wf=wf+f(1,j);end;end  %计算最大流量
zwf=0;for(i=1:n)for(j=1:n)zwf=zwf+b(i,j)*f(i,j);end;end  %计算最小费用
f  %显示最小费用最大流
wf  %显示最小费用最大流量
zwf  %显示最小费用,程序结束
```

运行结果如下：

```
f=
     0     6    16     0     0
     0     0     0     0    14
     0     8     0     8     0
     0     0     0     0     8
     0     0     0     0     0
wf=22
zwf=110
```

**结果分析**　最小费用最大流量为22，最小费用为110. $f$ 矩阵为最小费用最大流.

## 4.8　二部图的匹配及应用

**定义 4.7**　设 $X,Y$ 都是非空有限顶点集，且 $X\cap Y=\varnothing$，$E\subset\{xy|x\in X,y\in Y\}$，称 $G=(X,Y,E)$ 为二部图. 如果 $X$ 中的每个点都与 $Y$ 中的每个点邻接，则称 $G=(X,Y,E)$ 为完备二部图. 若 $F:E\to\mathbf{R}^+$，则称 $G=(X,Y,E,F)$ 为二部赋权图.