

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 $C_n H_{2n+2}$ 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、运筹学，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。

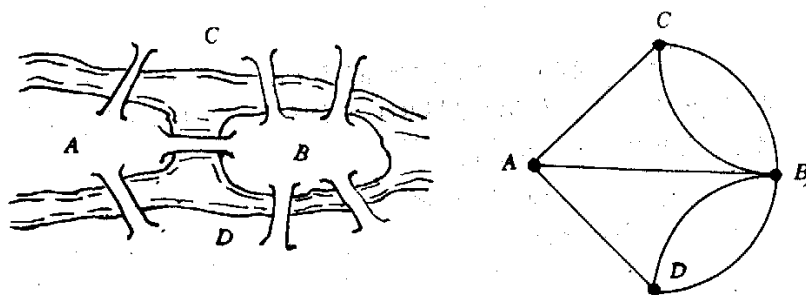


图 1 哥尼斯堡七桥问题

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 (network 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 $V(G)=\{v_1,v_2,\dots,v_n\}$ 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 $v_i (i=1,2,\dots,n)$ 称为该图的一个顶点 (vertex) 或节点 (node)； $E(G)=\{e_1,e_2,\dots,e_m\}$ 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 $e_k=v_i v_j=v_j v_i (k=1,2,\dots,m)$ ，被称为该图的一条从 v_i 到 v_j 的边 (edge)。

当边 $e_k=v_i v_j$ 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 network 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 $\nu(G)$ 表示，边数用 $|E|$ 或 $\varepsilon(G)$ 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, ν 和 ε 代替 $V(G), E(G), \nu(G)$ 和 $\varepsilon(G)$ 。

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

当弧 $a_k = v_i v_j$ 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 $V(G) = X \cup Y$ ， $X \cap Y = \Phi$ ， $|X| |Y| \neq 0$ (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：

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

(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 $1, 2, \dots, n$ 表示或编号， A 中的弧用自然数 $1, 2, \dots, m$ 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。

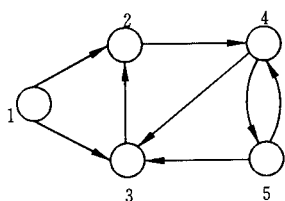


图 2 有向图

例 7 对于图 2 所示的有向图，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，则弧表表示如表 1 所示。

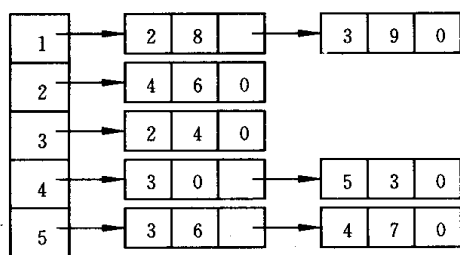
表 1

| | | | | | | | | |
|----|---|---|---|---|---|---|---|---|
| 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。

(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), (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。此时该网络图可以用前向星形表示法表示为表 2 和表 3。

表 2 节点对应的出弧的起始地址编号数组

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

表 3 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弧，可以采用反向星形（reverse star）表示法：首先存放进入节点 1 的所有弧，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* 的所有弧。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，我们一般还用一数组记录每个节点的入弧的起始地址（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表 4 和表 5。

表 4 节点对应的入弧的起始地址编号数组

| | | | | | | |
|-----------------------|---|---|---|---|---|---|
| 节点号 <i>i</i> | 1 | 2 | 3 | 4 | 5 | 6 |
| 起始地址 <i>rpoint(i)</i> | 1 | 1 | 3 | 6 | 8 | 9 |

表 5 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，将弧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。这相当于一紧凑的双向星形表示法，如表 6 所示。

表 6 两种表示法中的弧的对应关系

| | | | | | | | | |
|-------------------------|---|---|---|---|---|---|---|---|
| 反向法中弧编号 <i>j</i> | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 正向法中弧编号 <i>trace(j)</i> | 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

$W = v_0 e_1 v_1 e_2 \cdots e_k v_k$ ，其中 $e_i \in E(G)$ ， $1 \leq i \leq k$ ， $v_j \in V(G)$ ， $0 \leq j \leq k$ ， 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, \dots, 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顶点的 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 至各顶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, \dots, 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:

```
clc,clear
a=zeros(6);
a(1,2)=50;a(1,4)=40;a(1,5)=25;a(1,6)=10;
a(2,3)=15;a(2,4)=20;a(2,6)=25;
a(3,4)=10;a(3,5)=20;
a(4,5)=10;a(4,6)=25;
a(5,6)=55;
a=a+a';
a(find(a==0))=inf;
pb(1:length(a))=0;pb(1)=1;index1=1;index2=ones(1,length(a));
d(1:length(a))=inf;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];
    temp2=find(d(index1)==d(temp)-a(temp,index1));
    index2(temp)=index1(temp2(1));
end
d, index1, index2
```

我们编写的从起点 sb 到终点 db 通用的 Dijkstra 标号算法程序如下:

```
function [mydistance,mypath]=mydijkstra(a,sb,db);
% 输入: a—邻接矩阵(aij)是指i到j之间的距离, 可以是有向的
% sb—起点的标号, db—终点的标号
% 输出: mydistance—最短路的距离, mypath—最短路的路径
n=size(a,1); visited(1:n) = 0;
distance(1:n) = inf; % 保存起点到各顶点的最短距离
```

```

distance(sb) = 0; parent(1:n) = 0;
for i = 1: n-1
    temp=distance;
    id1=find(visited==1); %查找已经标号的点
    temp(id1)=inf; %已标号点的距离换成无穷
    [t, u] = min(temp); %找标号值最小的顶点
    visited(u) = 1; %标记已经标号的顶点
    id2=find(visited==0); %查找未标号的顶点
    for v = id2
        if a(u, v) + distance(u) < distance(v)
            distance(v) = distance(u) + a(u, v); %修改标号值
            parent(v) = u;
        end
    end
end
end
mypath = [];
if parent(db) ~= 0 %如果存在路!
    t = db; mypath = [db];
    while t ~= sb
        p = parent(t);
        mypath = [p mypath];
        t = p;
    end
end
end
mydistance = distance(db);
return

```

3.2 两个指定顶点之间最短路问题的数学表达式

假设有向图有 n 个顶点，现要求从顶点 1 到顶点 n 的最短路。设 $W = (w_{ij})_{n \times n}$ 为赋权邻接矩阵，其分量为

$$w_{ij} = \begin{cases} w(v_i v_j), & v_i v_j \in E \\ \infty, & \text{其它} \end{cases}$$

决策变量为 x_{ij} ，当 $x_{ij} = 1$ ，说明弧 $v_i v_j$ 位于顶点 1 至顶点 n 的路上；否则 $x_{ij} = 0$ 。其数学规划表达式为

$$\begin{aligned} & \min \sum_{v_i v_j \in E} w_{ij} x_{ij} \\ \text{s.t. } & \sum_{\substack{j=1 \\ v_i v_j \in E}}^n x_{ij} - \sum_{\substack{j=1 \\ v_j v_i \in E}}^n x_{ji} = \begin{cases} 1, & i = 1 \\ -1, & i = n \\ 0, & i \neq 1, n \end{cases} \\ & x_{ij} = 0 \text{ 或 } 1 \end{aligned}$$

例 10 在图 3 中，用点表示城市，现有 $A, B_1, B_2, C_1, C_2, C_3, D$ 共 7 个城市。点与点之间的连线表示城市间有道路相连。连线旁的数字表示道路的长度。现计划从城市 A 到城市 D 铺设一条天然气管道，请设计出最小价格管道铺设方案。

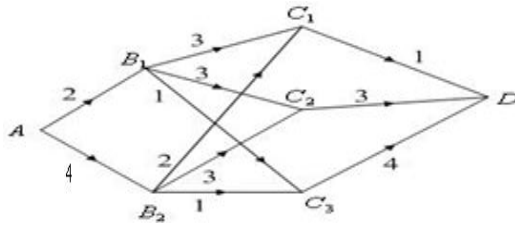


图3 7个城市间的连线图

编写 LINGO 程序如下:

```

model:
sets:
cities/A,B1,B2,C1,C2,C3,D/;
roads(cities,cities)/A B1,A B2,B1 C1,B1 C2,B1 C3,B2 C1,
B2 C2,B2 C3,C1 D,C2 D,C3 D/:w,x;
endsets
data:
w=2 4 3 3 1 2 3 1 1 3 4;
enddata
n=@size(cities); !城市的个数;
min=@sum(roads:w*x);
@for(cities(i)|i #ne#1 #and# i #ne#n:
    @sum(roads(i,j):x(i,j))=@sum(roads(j,i):x(j,i)));
@sum(roads(i,j)|i #eq#1:x(i,j))=1;
@sum(roads(i,j)|j #eq#n:x(i,j))=1;
end

```

例 11 (无向图的最短路问题) 求图 4 中 v_1 到 v_{11} 的最短路。

分析 例 10 处理的问题属于有向图的最短路问题, 本例是处理无向图的最短路问题, 在处理方式上与有向图的最短路问题有一些差别, 这里选择赋权邻接矩阵的方法编写 LINGO 程序。

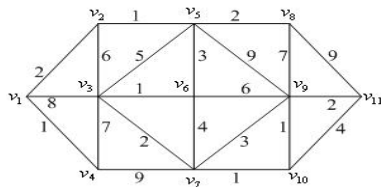


图4 无向图的最短路问题

编写 LINGO 程序如下:

```

model:
sets:
cities/1..11/;
roads(cities,cities):w,x;
endsets
data:
w=0;
enddata
calc:
w(1,2)=2;w(1,3)=8;w(1,4)=1;
w(2,3)=6;w(2,5)=1;

```

```

w(3,4)=7;w(3,5)=5;w(3,6)=1;w(3,7)=2;
w(4,7)=9;
w(5,6)=3;w(5,8)=2;w(5,9)=9;
w(6,7)=4;w(6,9)=6;
w(7,9)=3;w(7,10)=1;
w(8,9)=7;w(8,11)=9;
w(9,10)=1;w(9,11)=2;w(10,11)=4;
@for(roads(i,j):w(i,j)=w(i,j)+w(j,i));
@for(roads(i,j):w(i,j)=@if(w(i,j) #eq# 0, 1000,w(i,j)));
endcalc
n=@size(cities); !城市的个数;
min=@sum(roads:w*x);
@for(cities(i)|i #ne#1 #and# i #ne#
n:@sum(cities(j):x(i,j))=@sum(cities(j):x(j,i)));
@sum(cities(j):x(1,j))=1;
@sum(cities(j):x(j,1))=0; !不能回到顶点1;
@sum(cities(j):x(j,n))=1;
@for(roads:@bin(x));
end

```

与有向图相比较，在程序中只增加了一个语句 $\text{@sum(cities(j):x(j,1))=0}$ ，即从顶点 1 离开后，再不能回到该顶点。

求得的最短路径为 $1 \rightarrow 2 \rightarrow 5 \rightarrow 6 \rightarrow 3 \rightarrow 7 \rightarrow 10 \rightarrow 9 \rightarrow 11$ ，最短路径长度为 13。

3.3 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 $n-1$ 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 ，

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

来存放各边长度，其中：

$$a_{ii} = 0 \quad i = 1, 2, \dots, n;$$

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

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

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 $A_0, A_1, \dots, A_k, \dots, A_n$ ，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, $i, j, k = 1, 2, \dots, n$ 。

最后, 当 $k = n$ 时, A_n 即是各顶点之间的最短通路值。

例12 用Floyd算法求解例9。

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;
n=6; a=zeros(n);
a(1,2)=50;a(1,4)=40;a(1,5)=25;a(1,6)=10;
a(2,3)=15;a(2,4)=20;a(2,6)=25; a(3,4)=10;a(3,5)=20;
a(4,5)=10;a(4,6)=25; a(5,6)=55;
a=a+a'; M=max(max(a))*n^2; %M为充分大的正实数
a=a+((a==0)-eye(n))*M;
path=zeros(n);
for k=1:n
    for i=1:n
        for j=1:n
            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
```

我们使用LINGO9.0编写的FLOYD算法如下:

```
model:
sets:
nodes/c1..c6/;
link(nodes,nodes):w,path; !path标志最短路径上走过的顶点;
endsets
data:
path=0;
w=0;
@text(mydata1.txt)=@writefor(nodes(i):@writefor(nodes(j):
    @format(w(i,j),' 10.0f'),@newline(1));
@text(mydata1.txt)=@write(@newline(1));
@text(mydata1.txt)=@writefor(nodes(i):@writefor(nodes(j):
    @format(path(i,j),' 10.0f'),@newline(1));
enddata
calc:
w(1,2)=50;w(1,4)=40;w(1,5)=25;w(1,6)=10;
w(2,3)=15;w(2,4)=20;w(2,6)=25;
w(3,4)=10;w(3,5)=20;
w(4,5)=10;w(4,6)=25;w(5,6)=55;
@for(link(i,j):w(i,j)=w(i,j)+w(j,i));
@for(link(i,j) |i#ne#j:w(i,j)=@if(w(i,j)#eq#0,10000,w(i,j)));
@for(nodes(k):@for(nodes(i):@for(nodes(j):
```

```

        tm=@smin(w(i,j),w(i,k)+w(k,j));
        path(i,j)=@if(w(i,j)#gt# tm,k,path(i,j));w(i,j)=tm));
    endcalc
end

```

我们编写的求起点 sb 到终点 db 通用的 Floyd 算法程序如下:

```

function [dist,mypath]=myfloyd(a,sb,db);
% 输入: a—邻接矩阵(aij)是指 i 到 j 之间的距离, 可以是有向的
% sb—起点的标号; db—终点的标号
% 输出: dist—最短路的距离; % mypath—最短路的路径
n=size(a,1); path=zeros(n);
for i=1:n
    for j=1:n
        if a(i,j)~=inf
            path(i,j)=j; %j 是 i 的后续点
        end
    end
end
for k=1:n
    for i=1:n
        for j=1:n
            if a(i,j)>a(i,k)+a(k,j)
                a(i,j)=a(i,k)+a(k,j);
                path(i,j)=path(i,k);
            end
        end
    end
end
dist=a(sb,db);
mypath=sb; t=sb;
while t~=db
    temp=path(t,db);
    mypath=[mypath,temp];
    t=temp;
end
return

```

§ 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 $\tau(G)$ 表示 G 的生成树的个数, 则有公式

$$\text{公式 (Caylay)} \quad \tau(K_n) = n^{n-2}.$$

$$\text{公式} \quad \tau(G) = \tau(G - e) + \tau(G \cdot e).$$

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 $\varepsilon = v - 1$ 。
- (iii) G 是树当且仅当 G 连通, 且 $\varepsilon = v - 1$ 。
- (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 $P \sim V$
 找最小边 pv , 其中 $p \in P, v \in V - P$
 $P = P + \{v\}$
 $Q = Q + \{pv\}$
 end

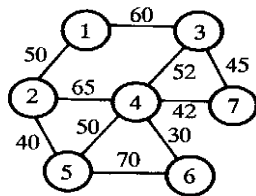


图 5 最小生成树问题

例 13 用 prim 算法求图 5 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;
a=zeros(7);
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+a';a(find(a==0))=inf;
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 (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, \dots, e_i 已选好, 则从 $E(G) - \{e_1, e_2, \dots, e_i\}$ 中选取 e_{i+1} , 使得
- ① $G[\{e_1, e_2, \dots, e_i, e_{i+1}\}]$ 中无圈, 且
 - ② $w(e_{i+1}) = \min$ 。
- (iii) 直到选得 e_{v-1} 为止。

例 14 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;
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,b]=find(a);
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=index(1,flag);v2=index(2,flag);
    if v1~=v2
        result=[result,data(:,flag)];
    end
    index(find(index==v2))=v1;
    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 中的一个对集; 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 次 ($k > 0$) 正则 2 分图, 则 G 有完美对集。

所谓 k 次正则图, 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, \dots, x_n 去做 n 件工作 y_1, y_2, \dots, 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 $V(G) = X \cup Y$, $X = \{x_1, \dots, x_n\}$, $Y = \{y_1, \dots, y_n\}$, 当且仅当 x_i 适合做工作 y_j 时, $x_i y_j \in E(G)$, 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 $yz \in M$, $S = S \cup \{z\}$, $T = T \cup \{y\}$, 转 (iii); 否则, 取可增广轨 $P(u, y)$, 令 $M = (M - E(P)) \cup (E(P) - M)$, 转 (ii)。

把以上算法稍加修改就能够用来求二分图的最大完美对集。

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 $w(x_i y_j) \geq 0$, 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 $l:V(G) \rightarrow R$, 满足 $\forall x \in X, y \in Y$,

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

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

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

$$l(y) = 0, \quad y \in Y。$$

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

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

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

$$l = \bar{l}, \quad G_l = G_{\bar{l}}。$$

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

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

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 6 (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°. $\forall v_0 \in V(G)$ ，令 $W_0 = v_0$ 。

2°. 假设迹 $W_i = v_0 e_1 v_1 \cdots e_i v_i$ 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} ：

(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 $G_i = G - \{e_1, \cdots, e_i\}$ 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°. 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 $V_0 = \{v \mid v \in V(G), d(v) = 1(\bmod 2)\}$ 。

(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, \cdots, C_k ，使得

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

$$(ii) \max_{1 \leq i \leq k} \sum_{e \in E(C_i)} w(e) = \min,$$

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

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 $C = v_1 v_2 \cdots v_n v_1$ 。

(i) 对于 $1 < i+1 < j < n$, 构造新的 Hamilton 圈:

$$C_{ij} = v_1 v_2 \cdots v_i v_j v_{j-1} v_{j-2} \cdots v_{i+1} v_{j+1} v_{j+2} \cdots v_n v_1,$$

它是由 C 中删去边 $v_i v_{i+1}$ 和 $v_j v_{j+1}$, 添加边 $v_i v_j$ 和 $v_{i+1} v_{j+1}$ 而得到的。若 $w(v_i v_j) + w(v_{i+1} v_{j+1}) < w(v_i v_{i+1}) + w(v_j v_{j+1})$, 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 $w(C)$ 的一个上界。

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, 就不对了。

例 15 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表 7。

表 7 六城市间的距离

| | 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:

```
function main
clc,clear
global a
a=zeros(6);
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=a+a'; L=size(a,1);
c1=[5 1:4 6];
[circle,long]=modifycircle(c1,L);
c2=[5 6 1:4];%改变初始圈, 该算法的最后一个顶点不动
[circle2,long2]=modifycircle(c2,L);
if long2<long
    long=long2;
    circle=circle2;
end
circle,long
%*****
%修改圈的子函数
%*****
function [circle,long]=modifycircle(c1,L);
global a
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
end
long=a(c1(1),c1(L));
for i=1:L-1
    long=long+a(c1(i),c1(i+1));
end
circle=c1;

```

6.3.3 旅行商问题的数学表达式

设城市的个数为 n , d_{ij} 是两个城市 i 与 j 之间的距离, $x_{ij} = 0$ 或 1 (1 表示走过城市 i 到城市 j 的路, 0 表示没有选择走这条路)。则有

$$\min \sum_{i \neq j} d_{ij} x_{ij}$$

s.t. $\sum_{j=1}^n x_{ij} = 1, i = 1, 2, \dots, n$, (每个点只有一条边出去)

$$\sum_{i=1}^n x_{ij} = 1, j = 1, 2, \dots, n$$
, (每个点只有一条边进去)
$$\sum_{i,j \in s} x_{ij} \leq |s| - 1, 2 \leq |s| \leq n-1, s \subset \{1, 2, \dots, n\}$$

(除起点和终点外, 各边不构成圈)

$$x_{ij} \in \{0,1\}, \quad i, j = 1, 2, \dots, n, \quad i \neq j.$$

其中 $|s|$ 表示集合 s 中元素个数。

将旅行商问题写成数学规划的具体形式还需要一定的技巧，下面的例子我们引用LINGO帮助中的一个程序。

例 16 已知 SV 地区各城镇之间距离见表 8，某公司计划在 SV 地区做广告宣传，推销员从城市 1 出发，经过各个城镇，再回到城市 1。为节约开支，公司希望推销员走过这 10 个城镇的总距离最少。

表 8 城镇之间的距离

| | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|----|----|----|----|----|----|----|
| 1 | 8 | 5 | 9 | 12 | 14 | 12 | 16 | 17 | 22 |
| 2 | | 9 | 15 | 17 | 8 | 11 | 18 | 14 | 22 |
| 3 | | | 7 | 9 | 11 | 7 | 12 | 12 | 17 |
| 4 | | | | 3 | 17 | 10 | 7 | 15 | 18 |
| 5 | | | | | 8 | 10 | 6 | 15 | 15 |
| 6 | | | | | | 9 | 14 | 8 | 16 |
| 7 | | | | | | | 8 | 6 | 11 |
| 8 | | | | | | | | 11 | 11 |
| 9 | | | | | | | | | 10 |

解 编写 LINGO 程序如下：

```

MODEL:
SETS:
  CITY / 1.. 10/: U; ! U( I) = sequence no. of city;
  LINK( CITY, CITY):
    DIST, ! The distance matrix;
    X; ! X( I, J) = 1 if we use link I, J;
ENDSETS
DATA: !Distance matrix, it need not be symmetric;
  DIST = 0      8      5      9      12      14      12      16      17      22
          8      0      9      15      17      8      11      18      14      22
          5      9      0      7      9      11      7      12      12      17
          9      15      7      0      3      17      10      7      15      18
          12      17      9      3      0      8      10      6      15      15
          14      8      11      17      8      0      9      14      8      16
          12      11      7      10      10      9      0      8      6      11
          16      18      12      7      6      14      8      0      11      11
          17      14      12      15      15      8      6      11      0      10
          22      22      17      18      15      16      11      11      10      0;
ENDDATA
!The model:Ref. Desrochers & Laporte, OR Letters,
Feb. 91;
N = @SIZE( CITY);
MIN = @SUM( LINK: DIST * X);
@FOR( CITY( K):
  ! It must be entered;
  @SUM( CITY( I) | I #NE# K: X( I, K)) = 1;
  ! It must be departed;
  @SUM( CITY( J) | J #NE# K: X( K, J)) = 1;
  ! Weak form of the subtour breaking constraints;
  ! These are not very powerful for large problems;
  @FOR( CITY( J) | J #GT# 1 #AND# J #NE# K:

```

```

    U( J ) >= U( K ) + X ( K, J ) -
    ( N - 2 ) * ( 1 - X( K, J ) ) +
    ( N - 3 ) * X( J, K ) ) );
! Make the X's 0/1;
@FOR( LINK: @BIN( X ) );
! For the first and last stop we know...;
@FOR( CITY( K ) | K #GT# 1:
    U( K ) <= N - 1 - ( N - 2 ) * X( 1, K );
    U( K ) >= 1 + ( N - 2 ) * X( K, 1 ) );
END

```

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

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

(iii) $D: V \rightarrow R$ 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 ，可以直接用所有弧上对应的权和顶点上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

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

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

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 个单位的流量从网络外部流入该顶点，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 $|d_i|$ 个单位的流量从该顶点流失到网络外部 (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

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

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 $L = 0$ (即所有弧 (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

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

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

$$v = v(f) = d_s = -d_t.$$

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:

$$\max v$$

$$\text{s.t. } \sum_{j:(i,j) \in A} f_{ij} - \sum_{j:(j,i) \in A} f_{ji} = \begin{cases} v, & i = s \\ -v, & i = t \\ 0, & i \neq s, t \end{cases}, \quad (4)$$

$$0 \leq f_{ij} \leq u_{ij}, \quad \forall (i, j) \in A. \quad (5)$$

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 7 (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

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

所定义的函数 f' 是 G' 中使得 $v(f') = v(f)$ 的流，这里 $f^+(v)$ 表示流出 v 的流量， $f^-(v)$ 表示流入 v 的流量（在 G 中）。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 8 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 $(s, v_2, \dots, v_{n-1}, t)$ （此轨为无向的），若 $v_i v_{i+1} \in A$ ，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

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

$$\delta = \min\{\delta_{ij}\}$$

则在这条可增广轨上每条前向弧的流都可以增加一个量 δ ，而相应的后向弧的流可减少 δ ，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。即首先给出一个初始流，这样的流是存在的，例如零流。如果存在关于它的可增广轨，那么调整该轨上每条弧上的流量，就可以得到新的流。对于新的流，如果仍存在可增广轨，则用同样的方法使流的值增大，继续这个过程，直到网络中不存在关于新得到流的可增广轨为止，则该流就是所求的最大流。

这种方法分为以下两个过程：

A. 标号过程：通过标号过程寻找一条可增广轨。

B. 增流过程：沿着可增广轨增加网络的流量。

这两个过程的步骤分述如下。

(A) 标号过程：

(i) 给发点标号为 (s^+, ∞) 。

(ii) 若顶点 x 已经标号，则对 x 的所有未标号的邻接顶点 y 按以下规则标号：

① 若 $(x, y) \in A$ ，且 $f_{xy} < u_{xy}$ 时，令 $\delta_y = \min\{u_{xy} - f_{xy}, \delta_x\}$ ，

则给顶点 y 标号为 (x^+, δ_y) ，若 $f_{xy} = u_{xy}$ ，则不给顶点 y 标号。

② $(y, x) \in A$ ，且 $f_{yx} > 0$ ，令 $\delta_y = \min\{f_{yx}, \delta_x\}$ ，则给 y 标号为 (x^-, δ_y) ，若 $f_{yx} = 0$ ，则不给 y 标号。

(iii) 不断地重复步骤 (ii) 直到收点 t 被标号，或不再有顶点可以标号为止。当 t 被标号时，表明存在一条从 s 到 t 的可增广轨，则转向增流过程 (B)。如若 t 点不能被标号，且不存在其它可以标号的顶点时，表明不存在从 s 到 t 的可增广轨，算法结束，此时所获得的流就是最大流。

(B) 增流过程

(i) 令 $u = t$ 。

(ii) 若 u 的标号为 (v^+, δ_t) ，则 $f_{vu} = f_{vu} + \delta_t$ ；若 u 的标号为 (v^-, δ_t) ，则 $f_{uv} = f_{uv} - \delta_t$ 。

(iii) 若 $u = s$ ，把全部标号去掉，并回到标号过程 (A)。否则，令 $u = v$ ，并回到增流过程 (ii)。

求网络 $N = (s, t, V, A, U)$ 中的最大流 x 的算法的程序设计具体步骤如下：

对每个节点 j ，其标号包括两部分信息

$(\text{pred}(j), \max f(j))$

该节点在可能的增广路中的前一个节点 $\text{pred}(j)$ ，以及沿该可能的增广路到该节点为止可以增广的最大流量 $\max f(j)$ 。

STEP0 置初始可行流 x (如零流)；对节点 t 标号，即令 $\max f(t) =$ 任意正值 (如 1)。

STEP1 若 $\max f(t) > 0$ ，继续下一步；否则停止，已经得到最大流，结束。

STEP2 取消所有节点 $j \in V$ 的标号，即令 $\max f(j) = 0$ ， $\text{pred}(j) = 0$ ；令 $\text{LIST} = \{s\}$ ，对节点 s 标号，即令 $\max f(s) =$ 充分大的正值。

STEP3 如果 $\text{LIST} \neq \Phi$ 且 $\max f(t) = 0$ ，继续下一步；否则：(3a) 如果 t 已经有标号 (即 $\max f(t) > 0$)，则找到了一条增广路，沿该增广路对流 x 进行增广 (增广的流量为 $\max f(t)$ ，增广路可以根据 pred 回溯方便地得到)，转 STEP1。

(3b) 如果 t 没有标号 (即 $LIST = \Phi$ 且 $\max f(t) = 0$)，则停止，已得到最大流。

STEP4 从 $LIST$ 中移走一个节点 i ；寻找从节点 i 出发的所有可能的增广弧：(4a) 对非饱和前向弧 (i, j) ，若节点 j 没有标号 (即 $\text{pred}(j) = 0$)，对 j 进行标号，即令

$$\max f(j) = \min\{\max f(i), u_{ij} - x_{ij}\}, \quad \text{pred}(j) = i,$$

并将 j 加入 $LIST$ 中。

(4b) 对非空后向弧 (j, i) ，若节点 j 没有标号 (即 $\text{pred}(j) = 0$)，对 j 进行标号，即令

$$\max f(j) = \min\{\max f(i), x_{ij}\}, \quad \text{pred}(j) = -i,$$

并将 j 加入 $LIST$ 中。

例 17 用 Ford-Fulkerson 算法计算如图 6 网络中的最大流，每条弧上的两个数字分别表示容量和当前流量。

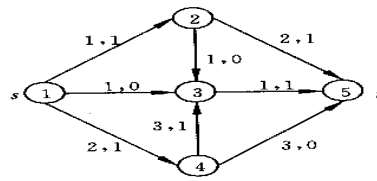


图 6 最大流问题

解 编写程序如下：

```

clc,clear
u(1,2)=1;u(1,3)=1;u(1,4)=2;u(2,3)=1;u(2,5)=2;
u(3,5)=1;u(4,3)=3;u(4,5)=3;
f(1,2)=1;f(1,3)=0;f(1,4)=1;f(2,3)=0;f(2,5)=1;
f(3,5)=1;f(4,3)=1;f(4,5)=0;
n=length(u);list=[];maxf(n)=1;
while maxf(n)>0
    maxf=zeros(1,n);pred=zeros(1,n);
    list=1;record=list;maxf(1)=inf;
    %list是未检查邻接点的标号点，record是已标号点
    while (~isempty(list))&(maxf(n)==0)
        flag=list(1);list(1)=[];
        labell=find(u(flag,:)-f(flag,:));
        labell=setdiff(labell,record);
        list=union(list,labell);
        pred(labell)=flag;
        maxf(labell)=min(maxf(flag),u(flag,labell)...
            -f(flag,labell));
        record=union(record,labell);
        label2=find(f(:,flag));
        label2=label2';
        label2=setdiff(label2,record);
        list=union(list,label2);
        pred(label2)=-flag;
        maxf(label2)=min(maxf(flag),f(label2,flag));
        record=union(record,label2);
    end
    if maxf(n)>0
        v2=n; v1=pred(v2);
        while v2~=1

```

```

        if v1>0
            f(v1,v2)=f(v1,v2)+maxf(n);
        else
            v1=abs(v1);
            f(v2,v1)=f(v2,v1)-maxf(n);
        end
        v2=v1; v1=pred(v2);
    end
end
end
end
f

```

例18 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 和 v_4 ，城市与中转站的连接以及管道的容量如图7所示，求从城市 s 到城市 t 的最大流。

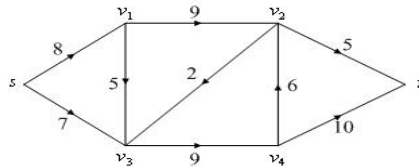


图7 最大流问题

解 使用最大流的数学规划表达式，编写LINGO程序如下：

```

model:
sets:
nodes/s,1,2,3,4,t/;
arcs(nodes,nodes)/s 1,s 3,1 2,1 3,2 3,2 t,3 4,4 2,4 t/:c,f;
endsets
data:
c=8 7 9 5 2 5 9 6 10;
enddata
n=@size(nodes); !顶点的个数;
max=flow;
@for(nodes(i)|i #ne#1 #and# i #ne# n:
    @sum(arcs(i,j):f(i,j))=@sum(arcs(j,i):f(j,i));
@sum(arcs(i,j)|i #eq# 1:f(i,j))=flow;
@sum(arcs(i,j)|j #eq# n:f(i,j))=flow;
@for(arcs:@bnd(0,f,c));
end

```

在上面的程序中，采用了稀疏集的编写方法。下面介绍的程序编写方法是利用赋权邻接矩阵，这样可以不使用稀疏集的编写方法，更便于推广到复杂网络。

```

model:
sets:
nodes/s,1,2,3,4,t/;
arcs(nodes,nodes):c,f;
endsets
data:
c=0;
@text('fdata.txt')=f;
enddata
calc:
c(1,2)=8;c(1,4)=7;
c(2,3)=9;c(2,4)=5;

```

```

c(3,4)=2;c(3,6)=5;
c(4,5)=9;c(5,3)=6;c(5,6)=10;
endcalc
n=@size(nodes); !顶点的个数;
max=flow;
@for(nodes(i)|i #ne#1 #and# i #ne# n:
    @sum(nodes(j):f(i,j))=@sum(nodes(j):f(j,i));
@sum(nodes(i):f(1,i))=flow;
@sum(nodes(i):f(i,n))=flow;
@for(arcs:@bnd(0,f,c));
end

```

§8 最小费用流及其求法

8.1 最小费用流

上面我们介绍了一个网络上最短路以及最大流的算法,但是还没有考虑到网络上流的费用问题,在许多实际问题中,费用的因素很重要。例如,在运输问题中,人们总是希望在完成运输任务的同时,寻求一个使总的运输费用最小的运输方案。这就是下面要介绍的最小费用流问题。

在运输网络 $N = (s, t, V, A, U)$ 中, 设 c_{ij} 是定义在 A 上的非负函数, 它表示通过弧 (i, j) 单位流的费用。所谓最小费用流问题就是从发点到收点怎样以最小费用输送一已知量为 $v(f)$ 的总流量。

最小费用流问题可以用如下的线性规划问题描述:

$$\begin{aligned}
 & \min \sum_{(i,j) \in A} c_{ij} f_{ij} \\
 \text{s.t.} \quad & \sum_{j:(i,j) \in A} f_{ij} - \sum_{j:(j,i) \in A} f_{ji} = d_i, \\
 & 0 \leq f_{ij} \leq u_{ij}, \quad \forall (i,j) \in A.
 \end{aligned}$$

$$\text{其中 } d_i = \begin{cases} v(f), & i = s \\ -v(f), & i = t \\ 0, & i \neq s, t \end{cases}$$

显然, 如果 $v(f) = \text{最大流 } v(f_{\max})$, 则本问题就是最小费用最大流问题。如果 $v(f) > v(f_{\max})$, 则本问题无解。

例 19 (最小费用最大流问题) (续例 18) 由于输油管道的长短不一或地质等原因, 使每条管道上运输费用也不相同, 因此, 除考虑输油管道的最大流外, 还需要考虑输油管道输送最大流的最小费用。图 8 所示是带有运费的网络, 其中第 1 个数字是网络的容量, 第 2 个数字是网络的单位运费。

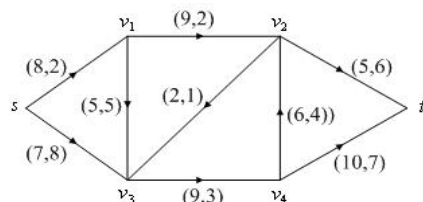


图 8 最小费用最大流问题

解 按照最小费用流的数学规划写出相应的 LINGO 程序如下:

```

model:
sets:
nodes/s,1,2,3,4,t/:d;
arcs(nodes,nodes)/s 1,s 3,1 2,1 3,2 3,2 t,3 4,4 2,4 t/:c,u,f;
endsets
data:
d=14 0 0 0 0 -14; !最大流为14;
c=2 8 2 5 1 6 3 4 7;
u=8 7 9 5 2 5 9 6 10;
enddata
min=@sum(arcs:c*f);
@for(nodes(i):@sum(arcs(i,j):f(i,j))-@sum(arcs(j,i):f(j,i))=d(i));
@for(arcs:@bnd(0,f,u));
end

```

求得最大流的最小费用是 205, 而原最大流的费用为 210 单位, 原方案并不是最优的。

类似地, 可以利用赋权邻接矩阵编程求得最小费用最大流。LINGO 程序如下:

```

model:
sets:
nodes/s,1,2,3,4,t/:d;
arcs(nodes,nodes):c,u,f;
endsets
data:
d=14 0 0 0 0 -14;
c=0; u=0;
enddata
calc:
c(1,2)=2;c(1,4)=8;
c(2,3)=2;c(2,4)=5;
c(3,4)=1;c(3,6)=6;
c(4,5)=3;c(5,3)=4;c(5,6)=7;
u(1,2)=8;u(1,4)=7;
u(2,3)=9;u(2,4)=5;
u(3,4)=2;u(3,6)=5;
u(4,5)=9;u(5,3)=6;u(5,6)=10;
endcalc
min=@sum(arcs:c*f);
@for(nodes(i):@sum(nodes(j):f(i,j))-@sum(nodes(j):f(j,i))=d(i));
@for(arcs:@bnd(0,f,u));
end

```

8.2 求最小费用流的一种方法—迭代法

这里所介绍的求最小费用流的方法叫做迭代法。这个方法是由 Busacker 和 Gowan 在 1961 年提出的。其主要步骤如下:

- (i) 求出从发点到收点的最小费用通路 $\mu(s,t)$ 。
- (ii) 对该通路 $\mu(s,t)$ 分配最大可能的流量:

$$\bar{f} = \min_{(i,j) \in \mu(s,t)} \{u_{ij}\}$$

并让通路上的所有边的容量相应减少 \bar{f} 。这时, 对于通路上的饱和边, 其单位流费用

相应改为 ∞ 。

(iii)作该通路 $\mu(s,t)$ 上所有边 (i,j) 的反向边 (j,i) 。令

$$u_{ji} = \bar{f}, \quad c_{ji} = -c_{ij}$$

(iv)在这样构成的新网络中, 重复上述步骤(i), (ii), (iii), 直到从发点到收点的全部流量等于 $v(f)$ 为止 (或者再也找不到从 s 到 t 的最小费用道路)。

下面我们编写了最小费用最大流函数 `mincostmaxflow`, 其中调用了利用 Floyd 算法求最短路的函数 `floydpath`。

求解例 19 具体程序如下 (下面的全部程序放在一个文件中):

```
function mainexample19
clear;clc;
global M num
c=zeros(6);u=zeros(6);
c(1,2)=2;c(1,4)=8;c(2,3)=2;c(2,4)=5;
c(3,4)=1;c(3,6)=6;c(4,5)=3;c(5,3)=4;c(5,6)=7;
u(1,2)=8;u(1,4)=7;u(2,3)=9;u(2,4)=5;
u(3,4)=2;u(3,6)=5;u(4,5)=9;u(5,3)=6;u(5,6)=10;
num=size(u,1);M=sum(sum(u))*num^2;
[f,val]=mincostmaxflow(u,c)

%求最短路径函数
function path=floydpath(w);
global M num
w=w+((w==0)-eye(num))*M;
p=zeros(num);
for k=1:num
    for i=1:num
        for j=1:num
            if w(i,j)>w(i,k)+w(k,j)
                w(i,j)=w(i,k)+w(k,j);
                p(i,j)=k;
            end
        end
    end
end
if w(1,num) ==M
    path=[];
else
    path=zeros(num);
    s=1;t=num;m=p(s,t);
    while ~isempty(m)
        if m(1)
            s=[s,m(1)];t=[t,t(1)];t(1)=m(1);
            m(1)=[];m=[p(s(1),t(1)),m,p(s(end),t(end))];
        else
            path(s(1),t(1))=1;s(1)=[];m(1)=[];t(1)=[];
        end
    end
end
end
```

```

%最小费用最大流函数
function [flow,val]=mincostmaxflow(rongliang,cost,flowvalue);
%第一个参数：容量矩阵；第二个参数：费用矩阵；
%前两个参数必须在不通路处置零
%第三个参数：指定容量值（可以不写，表示求最小费用最大流）
%返回值 flow 为可行流矩阵,val 为最小费用值
global M
flow=zeros(size(rongliang));allflow=sum(flow(1,:));
if nargin<3
    flowvalue=M;
end
while allflow<flowvalue
    w=(flow<rongliang).*cost-((flow>0).*cost);
    path=floydpath(w);%调用 floydpath 函数
    if isempty(path)
        val=sum(sum(flow.*cost));
        return;
    end
    theta=min(min(path.*(rongliang-flow)+(path.*(rongliang-flow)==0).*M));
    theta=min([min(path'.*flow+(path'.*flow==0).*M),theta]);
    flow=flow+(rongliang>0).*(path-path').*theta;
    allflow=sum(flow(1,:));
end
val=sum(sum(flow.*cost));

```

§ 9 计划评审方法和关键路线法

计划评审方法（program evaluation and review technique, PERT）和关键路线法（critical path method, CPM）是网络分析的重要组成部分，它广泛地用于系统分析和项目管理。计划评审与关键路线方法是在 20 世纪 50 年代提出并发展起来的，1956 年，美国杜邦公司为了协调企业不同业务部门的系统规划，提出了关键路线法。1958 年，美国海军武装部在研制“北极星”导弹计划时，由于导弹的研制系统过于庞大、复杂，为找到一种有效的管理方法，设计了计划评审方法。由于 PERT 与 CPM 既有着相同的目标应用，又有很多相同的术语，这两种方法已合并为一种方法，在国外称为 PERT/CPM，在国内称为统筹方法（scheduling method）。

9.1 计划网络图

例 20 某项目工程由 11 项作业组成（分别用代号 A, B, \dots, J, K 表示），其计划完成时间及作业间相互关系如表 9 所示，求完成该项目的最短时间。

表 9 作业流程数据

| 作业 | 计划完成时间（天） | 紧前作业 | 作业 | 计划完成时间（天） | 紧前作业 |
|-----|-----------|--------|-----|-----------|-----------|
| A | 5 | — | G | 21 | B, E |
| B | 10 | — | H | 35 | B, E |
| C | 11 | — | I | 25 | B, E |
| D | 4 | B | J | 15 | F, G, I |
| E | 4 | A | K | 20 | F, G |
| F | 15 | C, D | | | |

例 20 就是计划评审方法或关键路线法需要解决的问题。

9.1.1 计划网络图的概念

定义 称任何消耗时间或资源的行动称为作业。称作业的开始或结束为事件，事件本身不消耗资源。

在计划网络图中通常用圆圈表示事件，用箭线表示工作，如图 9 所示，1, 2, 3 表示事件，A, B 表示作业。由这种方法画出的网络图称为计划网络图。



图 9 计划网络图的基本画法

虚工作用虚箭线“……→”表示。它表示工时为零，不消耗任何资源的虚构工作。其作用只是为了正确表示工作的前行后继关系。

定义 在计划网络图中，称从初始事件到最终事件的由各项工作连贯组成的一条路为路线。具有累计作业时间最长的路线称为关键路线。

由此看来，例 20 就是求相应的计划网络图中的关键路线。

9.1.2 建立计划网络图应注意的问题

(1) 任何作业在网络中用唯一的箭线表示，任何作业其终点事件的编号必须大于其起点事件。

(2) 两个事件之间只能画一条箭线，表示一项作业。对于具有相同开始和结束事件的两项以上的作业，要引进虚事件和虚作业。

(3) 任何计划网络图应有唯一的最初事件和唯一的最终事件。

(4) 计划网络图不允许出现回路。

(5) 计划网络图的画法一般是左到右，从上到下，尽量作到清晰美观，避免箭头交叉。

9.2 时间参数

9.2.1 事件时间参数

(1) 事件的最早时间

事件 j 的最早时间用 $t_E(j)$ 表示，它表明以它为始点的各工作最早可能开始的时间，也表示以它为终点的各工作的最早可能完成时间，它等于从始点事件到该事件的最长路线上所有工作的工时总和。事件最早时间可用下列递推公式，按照事件编号从小到大的顺序逐个计算。

设事件编号为 $1, 2, \dots, n$ ，则

$$\begin{cases} t_E(1) = 0 \\ t_E(j) = \max_i \{t_E(i) + t(i, j)\} \end{cases} \quad (6)$$

其中 $t_E(i)$ 是与事件 j 相邻的各紧前事件的最早时间， $t(i, j)$ 是作业 (i, j) 所需的工时。

终点事件的最早时间显然就是整个工程的总最早完工期，即

$$t_E(n) = \text{总最早完工期} \quad (7)$$

(2) 事件的最迟时间

事件 i 的最迟时间用 $t_L(i)$ 表示，它表明在不影响任务总工期条件下，以它为始点的工作的最迟必须开始时间，或以它为终点的各工作的最迟必须完成时间。由于一般情

况下，我们都把任务的最早完工时间作为任务的总工期，所以事件最迟时间的计算公式为：

$$\begin{cases} t_L(n) = \text{总工期 (或 } t_E(n)) \\ t_L(i) = \min_j \{t_L(j) - t(i, j)\} \end{cases} \quad (8)$$

其中 $t_L(j)$ 是与事件 i 相邻的各紧后事件的最迟时间。

公式 (8) 也是递推公式，但与 (6) 相反，是从终点事件开始，按编号由大至小的顺序逐个由后向前计算。

9.2.2 工作的时间参数

(1) 工作的最早可能开工时间与工作的最早可能完工时间

一个工作 (i, j) 的最早可能开工时间用 $t_{ES}(i, j)$ 表示。任何一件工作都必须在其所有紧前工作全部完工后才能开始。工作 (i, j) 的最早可能完工时间用 $t_{EF}(i, j)$ 表示。它表示工作按最早开工时间开始所能达到的完工时间。它们的计算公式为：

$$\begin{cases} t_{ES}(1, j) = 0 \\ t_{ES}(i, j) = \max_k \{t_{ES}(k, i) + t(k, i)\} \\ t_{EF}(i, j) = t_{ES}(i, j) + t(i, j) \end{cases} \quad (9)$$

这组公式也是递推公式。即所有从总开工事件出发的工作 $(1, j)$ ，其最早可能开工时间为零；任一工作 (i, j) 的最早开工时间要由它的所有紧前工作 (k, i) 的最早开工时间决定；工作 (i, j) 的最早完工时间显然等于其最早开工时间与工时之和。

(2) 工作的最迟必须开工时间与工作的最迟必须完工时间

一个工作 (i, j) 的最迟开工时间用 $t_{LS}(i, j)$ 表示。它表示工作 (i, j) 在不影响整个任务如期完成的前提下，必须开始的最晚时间。

工作 (i, j) 的最迟必须完工时间用 $t_{LF}(i, j)$ 表示。它表示工作 (i, j) 按最迟时间开工，所能达到的完工时间。它们的计算公式为：

$$\begin{cases} t_{LF}(i, n) = \text{总完工期 (或 } t_{EF}(i, n)) \\ t_{LS}(i, j) = \min_k \{t_{LS}(j, k) - t(i, j)\} \\ t_{LF}(i, j) = t_{LS}(i, j) + t(i, j) \end{cases} \quad (10)$$

这组公式是按工作的最迟必须开工时间由终点向始点逐个递推的公式。凡是进入总完工事件 n 的工作 (i, n) ，其最迟完工时间必须等于预定总工期或等于这个工作的最早可能完工时间。任一工作 (i, j) 的最迟必须开工时间由它的所有紧后工作 (j, k) 的最迟开工时间确定。而工作 (i, j) 的最迟完工时间显然等于本工作的最迟开工时间与工时的和。

由于任一个事件 i （除去始点事件和终点事件），既表示某些工作的开始又表示某些工作的结束。所以从事件与工作的关系考虑，用公式 (9)，公式 (10) 求得的有关工作的时间参数也可以通过事件的时间参数公式 (6)，公式 (8) 来计算。如工作 (i, j) 的最早可能开工时间 $t_{ES}(i, j)$ 就等于事件 i 的最早时间 $t_E(i)$ 。工作 (i, j) 的最迟必须完工时间等于事件 j 的最迟时间。

9.2.3 时差

工作的时差又叫工作的机动时间或富裕时间，常用的时差有两种。

(1) 工作的总时差

在不影响任务总工期的条件下，某工作 (i, j) 可以延迟其开工时间的最大幅度，叫做该工作的总时差，用 $R(i, j)$ 表示。其计算公式为：

$$R(i, j) = t_{LF}(i, j) - t_{EF}(i, j) \quad (11)$$

即工作 (i, j) 的总时差等于它的最迟完工时间与最早完工时间的差。显然 $R(i, j)$ 也等于该工作的最迟开工时间与最早开工时间之差。

(2) 工作的单时差

工作的单时差是指在不影响紧后工作的最早开工时间条件下，此工作可以延迟其开工时间的最大幅度，用 $r(i, j)$ 表示。其计算公式为：

$$r(i, j) = t_{ES}(j, k) - t_{EF}(i, j) \quad (12)$$

即单时差等于其紧后工作的最早开工时间与本工作的最早完工时间之差。

9.3 计划网络图的计算

以例 20 的求解过程为例介绍计划网络图的计算方法。

9.3.1 建立计划网络图

首先建立计划网络图。按照上述规则，建立例 20 的计划网络图，如图 10 所示。

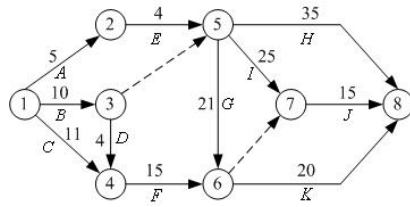


图 10 例 20 的计划网络图

9.3.2 写出相应的规划问题

设 x_i 是事件 i 的开始时间，1 为最初事件， n 为最终事件。希望总的工期最短，即极小化 $x_n - x_1$ 。设 t_{ij} 是作业 (i, j) 的计划时间，因此，对于事件 i 与事件 j 有不等式

$$x_j \geq x_i + t_{ij}$$

由此得到相应的数学规划问题

$$\begin{aligned} & \min x_n - x_1 \\ \text{s.t. } & x_j \geq x_i + t_{ij}, \quad (i, j) \in A, \quad i, j \in V \\ & x_i \geq 0, \quad i \in V \end{aligned}$$

其中 V 是所有的事件集合， A 是所有的作业集合。

9.3.3 问题求解

用 LINGO 软件求解例 20。

解 编写 LINGO 程序如下：

```
model:
sets:
events/1..8/:x;
operate(events,events)/1 2,1 3,1 4,2 5,3 4,3 5,4 6,5 6,
5 7,5 8,6 7,6 8,7 8/:t;
endsets
```

```

data:
t=5 10 11 4 4 0 15 21 25 35 0 20 15;
enddata
min=x(8)-x(1);
@for(operate(i,j):x(j)>x(i)+t(i,j));
end

```

计算结果给出了各个项目的开工时间，如 $x_1 = 0$ ，则作业 A, B, C 的开工时间均是第 0 天； $x_2 = 5$ ，作业 E 的开工时间是第 5 天； $x_3 = 10$ ，则作业 D 的开工时间是第 10 天；等等。每个作业只要按规定的时间开工，整个项目的最短工期为 51 天。

尽管上述 LINGO 程序给出相应的开工时间和整个项目的最短工期，但统筹方法中许多有用的信息并没有得到，如项目的关键路径、每个作业的最早开工时间、最迟开工时间等。

例 21（续例 20）求例 20 中每个作业的最早开工时间、最迟开工时间和作业的关键路径。

解 为了得到每个作业的最早开工时间、作业的关键路线等，将目标函数改为 $\sum_{i \in V} x_i$ ，即作业的开始时间尽量早，这样就可以得到作业的最早开工时间。再引进作业对应弧上的松弛变量 s_{ij} ，且 $s_{ij} = x_j - x_i - t_{ij}$ ， $(i, j) \in A$ ，这样就可以得到作业的最迟开工时间，记 y_i 表示事件 i 的最迟开工时间。当最早开工时间与最迟开工时间相同时，就得到项目的关键路径。

编写 LINGO 程序如下：

```

model:
sets:
events/1..8/:x,z;
operate(events,events)/1 2,1 3,1 4,2 5,3 4,3 5,4 6,5 6,
5 7,5 8,6 7,6 8,7 8/:s,t,m,c,y;
endsets
data:
t=5 10 11 4 4 0 15 21 25 35 0 20 15;
m=5 8 8 3 4 0 15 16 22 30 0 16 12;
c=0 700 400 450 0 0 0 600 300 500 0 500 400;
d=49;
@text(txt2.txt)=x,z;
enddata
min=mincost+sumx;
mincost=@sum(operate:c*y);
sumx=@sum(events:x);
@for(operate(i,j):s(i,j)=x(j)-x(i)+y(i,j)-t(i,j));
n=@size(events);
x(1)=0;
x(n)<d;
@for(operate:@bnd(0,y,t-m));
z(n)=x(n);
@for(events(i)|i#lt#n:z(i)=@min(operate(i,j):z(j)-t(i,j)+y(i,j)));
end

```

最迟开工时间的分析需要用到松弛变量 s_{ij} ，当 $s_{ij} > 0$ 时，说明还有剩余时间，对

应作业的工期可以推迟 s_{ij} 。例如, $s_{78} = 1$, 作业 (7, 8) (J) 的开工时间可以推迟 1 天, 即开工时间为 36。再如 $s_{46} = 2$, 作业 (4, 6) (F) 可以推迟 2 天开始, $s_{14} = 3$, 作业 (1, 4) (C) 可以推迟 3 天开始, 但由于作业 (4, 6) (F) 已能够推迟 2 天, 所以, 作业 (1, 4) (C) 最多可推迟 5 天。

由此, 可以得到所有作业的最早开工时间和最迟开工时间, 如下表所示, 方括号中第 1 个数字是最早开工时间, 第 2 个数字是最迟开工时间。

表 10 作业数据

| 作业 (i, j) | 开工时间 | 计划完成时间(天) | 作业 (i, j) | 开工时间 | 计划完成时间(天) |
|---------------|---------|-----------|---------------|---------|-----------|
| $A(1,2)$ | [0,1] | 5 | $G(5,6)$ | [10,10] | 21 |
| $B(1,3)$ | [0,0] | 10 | $H(5,8)$ | [10,16] | 35 |
| $C(1,4)$ | [0,5] | 11 | $I(5,7)$ | [10,11] | 25 |
| $D(3,4)$ | [10,12] | 4 | $J(7,8)$ | [35,36] | 15 |
| $E(2,5)$ | [5,6] | 4 | $K(6,8)$ | [31,31] | 20 |
| $F(4,6)$ | [14,16] | 15 | | | |

从上表可以看出, 当最早开工时间与最迟开工时间相同时, 对应的作业在关键路线上, 因此可以画出计划网络图中的关键路线, 如图 11 粗线所示。关键路线为 $1 \rightarrow 3 \rightarrow 5 \rightarrow 6 \rightarrow 8$ 。

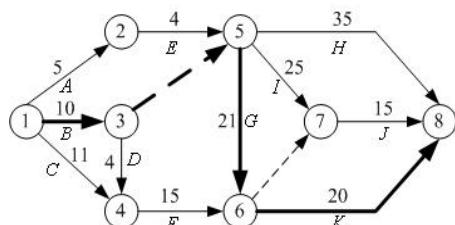


图 11 带有关键路线的计划网络图

9.3.4 将关键路线看成最长路

如果将关键路线看成最长路, 则可以按照求最短路的方法 (将求极小改为求极大) 求出关键路线。

设 x_{ij} 为 0-1 变量, 当作业 (i, j) 位于关键路线上取 1, 否则取 0。数学规划问题写成:

$$\begin{aligned} & \max \sum_{(i,j) \in A} t_{ij} x_{ij} \\ \text{s.t.} \quad & \sum_{\substack{j=1 \\ (i,j) \in A}}^n x_{ij} - \sum_{\substack{j=1 \\ (j,i) \in A}}^n x_{ji} = \begin{cases} 1, & i=1 \\ -1, & i=n \\ 0, & i \neq 1, n \end{cases} \\ & x_{ij} = 0 \text{ 或 } 1, (i, j) \in A \end{aligned}$$

例 22 用最长路的方法, 求解例 20。

解 按上述数学规划问题写出相应的 LINGO 程序。

model:

sets:

```

events/1..8/:d;
operate(events,events)/1 2,1 3,1 4,2 5,3 4,3 5,4 6,5 6,
5 7,5 8,6 7,6 8,7 8/:t,x;
endsets
data:
t=5 10 11 4 4 0 15 21 25 35 0 20 15;
d=1 0 0 0 0 0 0 -1;
enddata
max=@sum(operate:t*x);
@for(events(i):@sum(operate(i,j):x(i,j))-@sum(operate(j,i):x(j,i))=d(i));
end

```

求得工期需要 51 天，关键路线为 1→3→5→6→8。

9.4 关键路线与计划网络的优化

例 23(关键路线与计划网络的优化) 假设例 20 中所列的工程要求在 49 天内完成。为提前完成工程，有些作业需要加快进度，缩短工期，而加快进度需要额外增加费用。下表列出例 20 中可缩短工期的所有作业和缩短一天工期额外增加的费用。现在的问题是，如何安排作业才能使额外增加的总费用最少。

表 11 工程作业数据

| 作业 (i, j) | 计划完成 时间 (天) | 最短完成 时间 (天) | 缩短 1 天增加 的费用 (元) | 作业 (i, j) | 计划完成 时间 (天) | 最短完成 时间 (天) | 缩短 1 天增加 的费用 (元) |
|--------------|----------------|----------------|---------------------|--------------|----------------|----------------|---------------------|
| B(1,3) | 10 | 8 | 700 | H(5,8) | 35 | 30 | 500 |
| C(1,4) | 11 | 8 | 400 | I(5,7) | 25 | 22 | 300 |
| E(2,5) | 4 | 3 | 450 | J(7,8) | 15 | 12 | 400 |
| G(5,6) | 21 | 16 | 600 | K(6,8) | 20 | 16 | 500 |

例 23 所涉及的问题就是计划网络的优化问题，这时需要压缩关键路径来减少最短工期。

9.4.1 计划网络优化的数学表达式

设 x_i 是事件 i 的开始时间， t_{ij} 是作业 (i, j) 的计划时间， m_{ij} 是完成作业 (i, j) 的最短时间， y_{ij} 是作业 (i, j) 可能减少的时间， c_{ij} 是作业 (i, j) 缩短一天增加的费用，因此有

$$x_j - x_i \geq t_{ij} - y_{ij} \quad \text{且} \quad 0 \leq y_{ij} \leq t_{ij} - m_{ij}$$

设 d 是要求完成的天数，1 为最初事件， n 为最终事件，所以有 $x_n - x_1 \leq d$ 。而问题的总目标是使额外增加的费用最小，即目标函数为 $\min \sum_{(i,j) \in A} c_{ij} y_{ij}$ 。由此得到相应的数学

规划问题：

$$\begin{aligned}
& \min \sum_{(i,j) \in A} c_{ij} y_{ij} \\
& \text{s.t.} \quad x_j - x_i + y_{ij} \geq t_{ij}, \quad (i, j) \in A, \quad i, j \in V \\
& \quad \quad x_n - x_1 \leq d
\end{aligned}$$

$$0 \leq y_{ij} \leq t_{ij} - m_{ij}, (i, j) \in A, i, j \in V$$

9.4.2 计划网络优化的求解

用 LINGO 软件求解例 23, 程序如下:

```

model:
sets:
events/1..8/:x;
operate(events,events)/1 2,1 3,1 4,2 5,3 4,3 5,4 6,5 6,
5 7,5 8,6 7,6 8,7 8/:t,m,c,y;
endsets
data:
t=5 10 11 4 4 0 15 21 25 35 0 20 15;
m=5 8 8 3 4 0 15 16 22 30 0 16 12;
c=0 700 400 450 0 0 0 600 300 500 0 500 400;
d=49;
enddata
min=@sum(operate:c*y);
@for(operate(i,j):x(j)-x(i)+y(i,j)>t(i,j));
n=@size(events);
x(n)-x(1)<d;
@for(operate:@bnd(0,y,t-m));
end

```

作业(1,3)(B)压缩 1 天的工期, 作业(6,8)(K)压缩 1 天工期, 这样可以在 49 天完工, 需要多花费 1200 元。

如果需要知道压缩工期后的关键路径, 则需要稍复杂一点的计算。

例 24 (续例 23) 用 LINGO 软件求解例 23, 并求出相应的关键路径、各作业的最早开工时间和最迟开工时间。

解 为了得到作业的最早开工时间, 仍在目标函数中加入 $\sum_{i \in V} x_i$, 记 z_i 表示事件 i

的最迟开工时间, 其它处理方法与前面相同。

写出相应的 LINGO 程序如下:

```

model:
sets:
events/1..8/:x,z;
operate(events,events)/1 2,1 3,1 4,2 5,3 4,3 5,4 6,5 6,
5 7,5 8,6 7,6 8,7 8/:s,t,m,c,y;
endsets
data:
t=5 10 11 4 4 0 15 21 25 35 0 20 15;
m=5 8 8 3 4 0 15 16 22 30 0 16 12;
c=0 700 400 450 0 0 0 600 300 500 0 500 400;
d=49;
@text(txt2.txt)=x,z;
enddata
min=mincost+sumx;
mincost=@sum(operate:c*y);
sumx=@sum(events:x);
@for(operate(i,j):s(i,j)=x(j)-x(i)+y(i,j)-t(i,j));
n=@size(events);
x(1)=0;

```

```

x(n) < d;
@for(operate:@bnd(0,y,t-m));
z(n)=x(n);
@for(events(i)|i#t#n:z(i)=@min(operate(i,j):z(j)-t(i,j)+y(i,j)));
end

```

计算出所有作业的最早开工时间和最迟开工时间，见表 12。

表 12 作业数据

| 作业 (<i>i, j</i>) | 开工时间 | 实际完成时间(天) | 作业 (<i>i, j</i>) | 开工时间 | 实际完成时间(天) |
|--------------------|---------|-----------|--------------------|---------|-----------|
| A(1,2) | [0,0] | 5 | G(5,6) | [9,9] | 21 |
| B(1,3) | [0,0] | 9 | H(5,8) | [9,14] | 35 |
| C(1,4) | [0,4] | 11 | I(5,7) | [9,9] | 25 |
| D(3,4) | [9,12] | 4 | J(7,8) | [34,34] | 15 |
| E(2,5) | [5,5] | 4 | K(6,8) | [30,30] | 19 |
| F(4,6) | [13,15] | 15 | | | |

当最早开工时间与最迟开工时间相同时，对应的作业就在关键路线上，图 12 中的粗线表示优化后的关键路线。从图 5-10 可以看到，关键路线不止一条。

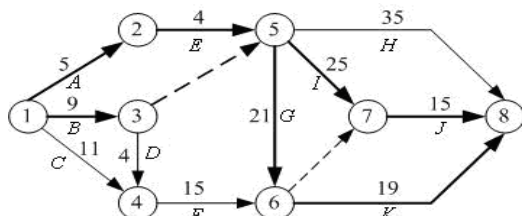


图 12 优化后的关键路线图

9.5 完成作业期望和实现事件的概率

在例 20 中，每项作业完成的时间均看成固定的，但在实际应用中，每一工作的完成会受到一些意外因素的干扰，一般不可能是完全确定的，往往只能凭借经验和过去完成类似工作需要的时间进行估计。通常情况下，对完成一项作业可以给出三个时间上的估计值：最乐观值的估计值 (*a*)，最悲观的估计值 (*b*) 和最可能的估计值 (*m*)。

设 t_{ij} 是完成作业 (*i, j*) 的实际时间 (是一随机变量)，通常用下面的方法计算相应的数学期望和方差。

$$E(t_{ij}) = \frac{a_{ij} + 4m_{ij} + b_{ij}}{6} \quad (13)$$

$$\text{var}(t_{ij}) = \frac{(b_{ij} - a_{ij})^2}{36} \quad (14)$$

设 *T* 为最短工期，即

$$T = \sum_{(i,j) \in \text{关键路线}} t_{ij} \quad (15)$$

由中心极限定理，可以假设 *T* 服从正态分布，并且期望值和方差满足

$$\bar{T} = E(T) = \sum_{(i,j) \in \text{关键路线}} E(t_{ij}) \quad (16)$$

$$S^2 = \text{var}(T) = \sum_{(i,j) \in \text{关键路线}} \text{var}(t_{ij}) \quad (17)$$

设规定的工期为 d ，则在规定的工期内完成整个项目的概率为

$$P\{T \leq d\} = \Phi\left(\frac{d - \bar{T}}{S}\right) \quad (18)$$

@psn(x) 是 LINGO 软件提供的标准正态分布函数，即

$$\text{@psn}(x) = \Phi(x) = \int_{-\infty}^x \frac{1}{\sqrt{2\pi}} e^{-t^2/2} dt \quad (19)$$

例 25 已知例 20 中各项作业完成的三个估计时间如下表所示。如果规定时间为 52 天，求在规定时间内完成全部作业的概率。进一步，如果完成全部作业的概率大于等于 95%，那么工期至少需要多少天？

表 13 作业数据

| 作业 (i, j) | 估计时间 (天) | | | 作业 (i, j) | 估计时间 (天) | | |
|--------------|----------|----|----|--------------|----------|----|----|
| | a | m | b | | a | m | b |
| A(1,2) | 3 | 5 | 7 | G(5,6) | 18 | 20 | 28 |
| B(1,3) | 8 | 9 | 16 | H(5,8) | 26 | 33 | 52 |
| C(1,4) | 8 | 11 | 14 | I(5,7) | 18 | 25 | 32 |
| D(3,4) | 2 | 4 | 6 | J(7,8) | 12 | 15 | 18 |
| E(2,5) | 3 | 4 | 5 | K(6,8) | 11 | 21 | 25 |
| F(4,6) | 8 | 16 | 18 | | | | |

解 对于这个问题采用最长路的编写方法。

按公式 (13) 和公式 (14) 计算出各作业的期望值与方差，再由期望时间计算出关键路线。从而由公式 (16) 和公式 (17) 得到关键路线的期望与方差的估计值，再利用分布函数 @psn(x)，计算出完成作业的概率与完成整个项目的时间。

写出相应的 LINGO 程序如下：

```

model:
sets:
events/1..8/:d;
operate(events,events)/1 2,1 3,1 4,2 5,3 4,3 5,4 6,5 6,
5 7,5 8,6 7,6 8,7 8/:a,m,b,et,dt,x;
endsets
data:
a=3 8 8 3 2 0 8 18 18 26 0 11 12;
m= 5 9 11 4 4 0 16 20 25 33 0 21 15;
b=7 16 14 5 6 0 18 28 32 52 0 25 18;
d=1 0 0 0 0 0 0 -1;
limit=52;
enddata
@for(operate:et=(a+4*m+b)/6;dt=(b-a)^2/36);
max=tbar;
tbar=@sum(operate:et*x);

```

```

@for(events(i):@sum(operate(i,j):x(i,j))-@sum(operate(j,i):x(j,i))=d(i));
s^2=@sum(operate:dt*x);
p=@psn((limit-tbar)/s);
@psn((days-tbar)/s)=0.95;
end

```

求得关键路线的时间期望为 51 天, 标准差为 3.16, 在 52 天完成全部作业的概率为 62.4%, 如果完成全部作业的概率大于等于 95%, 那么工期至少需要 56.2 天。

§ 10 钢管订购和运输

10.1 问题描述

要铺设一条 $A_1 \rightarrow A_2 \rightarrow \dots \rightarrow A_{15}$ 的输送天然气的主管道, 如图 14 所示。经筛选后可以生产这种主管道钢管的钢厂有 S_1, S_2, \dots, S_7 。图中粗线表示铁路, 单细线表示公路, 双细线表示要铺设的管道(假设沿管道或者原来有公路, 或者建有施工公路), 圆圈表示火车站, 每段铁路、公路和管道旁的阿拉伯数字表示里程(单位 km)。

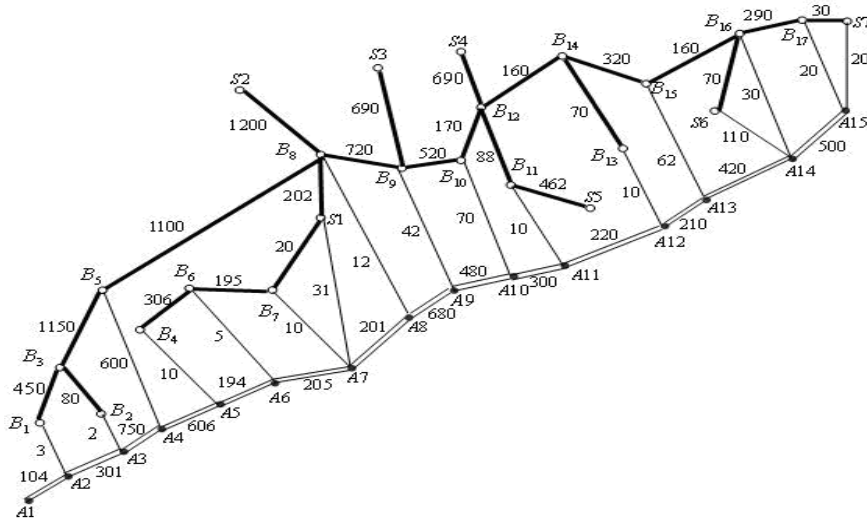


图 14

为方便计, 1km 主管道钢管称为 1 单位钢管。

一个钢厂如果承担制造这种钢管, 至少需要生产 500 个单位。钢厂 S_i 在指定期限内能生产该钢管的最大数量为 s_i 个单位, 钢管出厂销价 1 单位钢管为 p_i 万元, 见表 14; 1 单位钢管的铁路运价见表 15。

表 14

| i | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|-------|-----|-----|------|------|------|------|------|
| s_i | 800 | 800 | 1000 | 2000 | 2000 | 2000 | 3000 |
| p_i | 160 | 155 | 155 | 160 | 155 | 150 | 160 |

表 15 单位钢管的铁路运价

| 里程(km) | ≤ 300 | 301~350 | 351~400 | 401~450 | 451~500 |
|--------|------------|---------|---------|---------|---------|
| | | | | | |

| | | | | | |
|--------|---------|---------|---------|---------|----------|
| 运价(万元) | 20 | 23 | 26 | 29 | 32 |
| 里程(km) | 501~600 | 601~700 | 701~800 | 801~900 | 901~1000 |
| 运价(万元) | 37 | 44 | 50 | 55 | 60 |

1000km 以上每增加 1 至 100km 运价增加 5 万元。公路运输费用为 1 单位钢管每公里 0.1 万元（不足整公里部分按整公里计算）。钢管可由铁路、公路运往铺设地点（不只是运到点 A_1, A_2, \dots, A_{15} ，而是管道全线）。

(1) 请制定一个主管道钢管的订购和运输计划，使总费用最小（给出总费用）。

(2) 请就 (1) 的模型分析：哪个钢厂钢管的销价的变化对购运计划和总费用影响最大，哪个钢厂钢管的产量的上限的变化对购运计划和总费用的影响最大，并给出相应的数字结果。

(3) 如果要铺设的管道不是一条线，而是一个树形图，铁路、公路和管道构成网络，请就这种更一般的情形给出一种解决办法，并对图 15 按 (1) 的要求给出模型和结果。

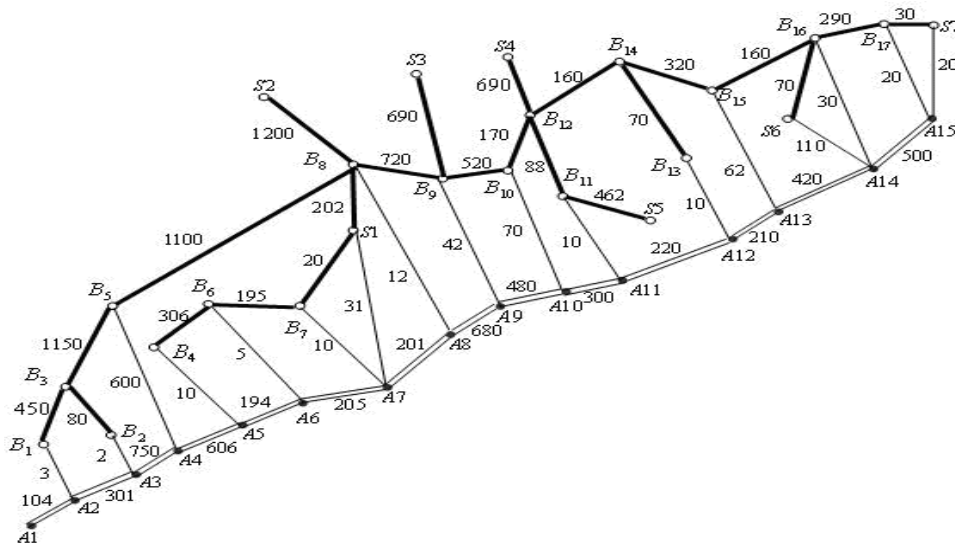


图 15

10.2 模型的建立与求解

记第 i 个钢厂的最大供应量为 s_i ，从第 i 个钢厂到铺设节点 j 的订购和运输费用为 c_{ij} ；用 l_j 表示管道第 j 段需要铺设的钢管量。 x_{ij} 是从钢厂 i 运到节点 j 的钢管量， y_j 是从节点 j 向左铺设的钢管量， z_j 是从节点 j 向右铺设的钢管量。

根据题中所给数据，我们可以先计算出从供应点 S_i 到需求点 A_j 的最小购运费 c_{ij} （即出厂售价与运输费用之和），再根据 c_{ij} 求解总费用，总费用应包括：订购费用（已包含在 c_{ij} 中），运输费用（由各厂 S_i 经铁路、公路至各点 A_j ， $i=1,2,\dots,7$ ， $j=1,2,\dots,15$ ），铺设管道 $A_j A_{j+1}$ ($j=1,2,\dots,14$) 的运费。

10.2.1 运费矩阵的计算模型

购买单位钢管及从 S_i ($i = 1, 2, \dots, 7$) 运送到 A_j ($j = 1, 2, \dots, 15$) 的最小购运费用 c_{ij} 的计算:

① 计算铁路任意两点间的最小运输费用

构造铁路距离赋权图 $G_1 = (V, E_1, W_1)$, 其中

$V = \{S_1, \dots, S_7, A_1, \dots, A_{15}, B_1, \dots, B_{17}\}$, 各顶点的编号见图 14; $W_1 = (w_{ij}^1)_{39 \times 39}$,

$$w_{ij}^1 = \begin{cases} d_{ij}^1, & i, j \text{ 之间有铁路直接相连} \\ +\infty, & i, j \text{ 之间没有铁路直接相连} \end{cases} \quad (d_{ij}^1 \text{ 表示 } i, j \text{ 两点之间的铁路路程。})$$

由于铁路运费不是连续的, 故不能直接用 Floyd 算法来计算最小运输费用。但可以用 Floyd 算法来计算任意两点间的最短铁路距离值, 再依据题中的铁路运价表, 来计算最小运输费用。这就巧妙的避开铁路运费不是连续的问题。最终计算出铁路任意两点间的最小运输费用 c_{ij}^1 。其中, 路径值无穷大时的费用也为无穷大。

② 计算公路任意两点间的最小运输费用

构造公路距离赋权图 $G_2 = (V, E_2, W_2)$, 其中 V 同上, $W_2 = (w_{ij}^2)_{39 \times 39}$,

$$w_{ij}^2 = \begin{cases} d_{ij}^2, & i, j \text{ 之间有公路直接相连} \\ +\infty, & i, j \text{ 之间没有公路直接相连} \end{cases} \quad (d_{ij}^2 \text{ 表示 } i, j \text{ 两点之间的公路路程。})$$

依据题中“公路运输费用为 1 单位钢管每公里 0.1 万元 (不足整公里部分按整公里计算)”, 计算出公路任意两点间的最小运输费用 c_{ij}^2 。路径值为无穷大时的费用也为无穷大。

③ 计算任意两点间的最小运输费用

由于可以用铁路、公路交叉运送, 所以任意相邻两点间的最小运输费用为铁路、公路两者最小运输费用的最小值。

构造铁路公路的混合赋权完全图 $G = (V, E, W)$, $W = (\tilde{c}_{ij})_{39 \times 39}$, 其中 $\tilde{c}_{ij} = \min(c_{ij}^1, c_{ij}^2)$ 。

对图 G 应用 Floyd 算法, 就可以计算出 S_i ($i = 1, 2, \dots, 7$) 到 A_j ($j = 1, 2, \dots, 15$) 的最小运送费用 \bar{c}_{ij} (单位: 万元) 见表 16。

表 16 最小运费计算结果

| | | | | | | | | | | | | | | |
|-------|-------|-------|-------|-----|-------|-----|-------|-------|-----|-----|-----|-------|-----|-----|
| 170.7 | 160.3 | 140.2 | 98.6 | 38 | 20.5 | 3.1 | 21.2 | 64.2 | 92 | 96 | 106 | 121.2 | 128 | 142 |
| 215.7 | 205.3 | 190.2 | 171.6 | 111 | 95.5 | 86 | 71.2 | 114.2 | 142 | 146 | 156 | 171.2 | 178 | 192 |
| 230.7 | 220.3 | 200.2 | 181.6 | 121 | 105.5 | 96 | 86.2 | 48.2 | 82 | 86 | 96 | 111.2 | 118 | 132 |
| 260.7 | 250.3 | 235.2 | 216.6 | 156 | 140.5 | 131 | 116.2 | 84.2 | 62 | 51 | 61 | 76.2 | 83 | 97 |
| 255.7 | 245.3 | 225.2 | 206.6 | 146 | 130.5 | 121 | 111.2 | 79.2 | 57 | 33 | 51 | 71.2 | 73 | 87 |
| 265.7 | 255.3 | 235.2 | 216.6 | 156 | 140.5 | 131 | 121.2 | 84.2 | 62 | 51 | 45 | 26.2 | 11 | 28 |
| 275.7 | 265.3 | 245.2 | 226.6 | 166 | 150.5 | 141 | 131.2 | 99.2 | 76 | 66 | 56 | 38.2 | 26 | 2 |

任意两点间的最小运输费用加上出厂售价, 得到单位钢管从任一个 S_i ($i = 1, 2, \dots, 7$) 到 A_j ($j = 1, 2, \dots, 15$) 的购买和运送最小费用 c_{ij} 。

10.2.2 总费用的数学规划模型

分析题目可以知道约束条件应包括：

钢厂产量约束：上限和下限（如果生产的话）；

运量约束： x_{ij} 对 i 求和等于 z_j 加 y_j ；

y_{j+1} 与 z_j 之和等于 $A_j A_{j+1}$ 段的长度 l_j 。

由 A_j 向 $A_j A_{j-1}$ 段铺设管道的运输总路程为 $1 + \dots + y_j = y_j(y_j + 1)/2$ ；

由 A_j 向 $A_j A_{j+1}$ 段铺设管道的运输总路程为 $1 + \dots + z_j = z_j(z_j + 1)/2$ 。

根据以上条件可以建立模型如下：

$$\min \sum_{i=1}^7 \sum_{j=1}^{15} c_{ij} x_{ij} + \frac{0.1}{2} \sum_{j=1}^{15} (z_j(z_j + 1) + y_j(y_j + 1)) \quad (20)$$

$$\text{s. t. } \sum_{j=1}^{15} x_{ij} \in \{0\} \cup [500, s_i] \quad i = 1, 2, \dots, 7 \quad (21)$$

$$\sum_{i=1}^7 x_{ij} = z_j + y_j \quad j = 1, 2, \dots, 15 \quad (22)$$

$$y_{j+1} + z_j = l_j \quad j = 1, 2, \dots, 14 \quad (23)$$

$$x_{ij} \geq 0, z_j \geq 0, y_j \geq 0 \quad i = 1, 2, \dots, 7, j = 1, 2, \dots, 15 \quad (24)$$

$$y_1 = 0, z_{15} = 0 \quad (25)$$

10.2.3 Lingo 程序

使用计算机求解上述数学规划时，需要对约束条件 (20) 进行处理。我们引进 0-1 变量

$$f_i = \begin{cases} 1, & \text{钢厂 } i \text{ 生产} \\ 0, & \text{钢厂 } i \text{ 不生产} \end{cases}$$

把约束条件 (20) 转化为

$$500f_i \leq \sum_{j=1}^{15} x_{ij} \leq s_i f_i, \quad i = 1, 2, \dots, 7 \quad (26)$$

利用 Lingo 求得总费用的最小值为 127.8632 亿。Lingo 程序如下：

model:

sets:

!nodes 表示节点集合;

nodes /S1,S2,S3,S4,S5,S6,S7,A1,A2,A3,A4,A5,A6,A7,A8,A9,A10,A11,A12,A13,A14,A15,
B1,B2,B3,B4,B5,B6,B7,B8,B9,B10,B11,B12,B13,B14,B15,B16,B17/;

!c1(i,j)表示节点 i 到 j 铁路运输的最小运价 (万元),c2(i,j)表示节点 i 到 j 公路运输的费

```

用邻接矩阵, c(i,j)表示节点 i 到 j 的最小运价, path 标志最短路径上走过的顶点;
link(nodes, nodes): w, c1,c2,c,path1,path;
supply/S1..S7/:S,P,f;
need/A1..A15/:b,y,z; !y 表示每一点往左铺的量, z 表示往右铺的量;
linkf(supply, need):cf,X;
endsets
data:
S=800 800 1000 2000 2000 2000 3000;
P=160 155 155 160 155 150 160;
b=104,301,750,606,194,205,201,680,480,300,220,210,420,500,0;
path1=0; path=0; w=0; c2=0;
! 以下是格式化输出计算的中间结果和最终结果;
@text(MiddleCost.txt)=@writefor(supply(i): @writefor(need(j): @format(cf(i,j),' 6.1f' )),
@newline(1));
@text(Train_path.txt)=@writefor(nodes(i):@writefor(nodes(j):@format(path1(i,j),'5.0f' )),
@newline(1));
@text(Final_path.txt)=@writefor(nodes(i):@writefor(nodes(j):@format(path(i,j),'5.0f' )),
@newline(1));
@text(FinalResult.txt)=@writefor(supply(i):@writefor(need(j):@format(x(i,j),'5.0f' )),
@newline(1) );
@text(FinalResult.txt)=@write(@newline(1));
@text(FinalResult.txt)=@writefor(need:@format(y,'5.0f' ));
@text(FinalResult.txt)=@write(@newline(2));
@text(FinalResult.txt)=@writefor(need:@format(z,'5.0f' ));
enddata
calc:
!输入铁路距离邻接矩阵的上三角元素;
w(1,29)=20;w(1,30)=202;w(2,30)=1200;w(3,31)=690;w(4,34)=690;w(5,33)=462;
w(6,38)=70;w(7,39)=30;w(23,25)=450;w(24,25)=80;w(25,27)=1150;w(26,28)=306;
w(27,30)=1100;w(28,29)=195;w(30,31)=720;w(31,32)=520;w(32,34)=170;w(33,34)=88;
w(34,36)=160;w(35,36)=70;w(36,37)=320;w(37,38)=160;w(38,39)=290;
@for(link(i,j): w(i,j) = w(i, j)+w(j,i) ); !输入铁路距离邻接矩阵的下三角元素;
@for(link(i,j)|i#ne#j: w(i,j)=@if(w(i,j) #eq# 0, 20000,w(i,j))); ! 无铁路连接, 元素为充分大的数;
!以下就是最短路计算公式 (Floyd-Warshall 算法) ;
@for(nodes(k):@for(nodes(i):@for(nodes(j):tm=@smin(w(i,j),w(i,k)+w(k,j));
path1(i,j)=@if(w(i,j)#gt# tm,k,path1(i,j));w(i,j)=tm));
!以下就是按最短路 w 查找相应运费 C1 的计算公式;
@for(link|w#eq#0: C1=0);
@for(link|w#gt#0 #and# w#le#300: C1=20);
@for(link|w#gt#300 #and# w#le#350: C1=23);
@for(link|w#gt#350 #and# w#le#400: C1=26);
@for(link|w#gt#400 #and# w#le#450: C1=29);
@for(link|w#gt#450 #and# w#le#500: C1=32);
@for(link|w#gt#500 #and# w#le#600: C1=37);
@for(link|w#gt#600 #and# w#le#700: C1=44);
@for(link|w#gt#700 #and# w#le#800: C1=50);
@for(link|w#gt#800 #and# w#le#900: C1=55);
@for(link|w#gt#900 #and# w#le#1000: C1=60);

```

```

@for(link|w#gt#1000: C1= 60+5*@floor(w/100-10)+@if(@mod(w,100)#eq#0,0,5) );
!输入公路距离邻接矩阵的上三角元素;
c2(1,14)=31;c2(6,21)=110;c2(7,22)=20;c2(8,9)=104;c2(9,10)=301;c2(9,23)=3;
c2(10,11)=750;c2(10,24)=2;c2(11,12)=606;c2(11,27)=600;c2(12,13)=194;c2(12,26)=10;
c2(13,14)=205;c2(13,28)=5;c2(14,15)=201;c2(14,29)=10;c2(15,16)=680;c2(15,30)=12;
c2(16,17)=480;c2(16,31)=42;c2(17,18)=300;c2(17,32)=70;c2(18,19)=220;c2(18,33)=10;
c2(19,20)=210;c2(19,35)=10;c2(20,21)=420;c2(20,37)=62;c2(21,22)=500;c2(21,38)=30;
c2(22,39)=20;
@for(link(i,j): c2(i,j) = c2(i,j)+c2(j,i)); !输入公路距离邻接矩阵的下三角元素;
@for(link(i,j):c2(i,j)=0.1*c2(i,j)); ! 距离转化成费用;
@for(link(i,j)|i#ne#j: c2(i,j) =@if(c2(i,j)#eq#0,10000,c2(i,j) )); !无公路连接, 元素为充分大的数;
@for(link: C= @if(C1#le#C2,C1,C2)); ! C1 和 C2 矩阵对应元素取最小;
@for(nodes(k):@for(nodes(i):@for(nodes(j):tm=@smin(C(i,j),C(i,k)+C(k,j));
path(i,j)=@if(C(i,j)#gt# tm,k,path(i,j));C(i,j)=tm));
@for(link(i,j)|i #le# 7#and# j#ge#8 #and# j#le# 22:cf(i,j-7)=c(i,j)); !提取下面二次规划模型需要的 7×15 矩阵;
endcalc
[obj]min=@sum(linkf(i,j):(cf(i,j)+p(i))*x(i,j))+0.05*@sum(need(j):y(j)^2+y(j)+z(j)^2+z(j));
! 约束;
@for(supply(i):[con1]@sum(need(j):x(i,j))<= S(i)*f(i));
@for(supply(i):[con2]@sum(need(j):x(i,j)) >= 500*f(i));
@for(need(j):[con3] @sum(supply(i):x(i,j))=y(j)+z(j));
@for(need(j)|j#NE#15:[con4] z(j)+y(j+1)=b(j));
y(1)=0; z(15)=0;
@for(supply: @bin(f));
@for(need: @gin(y));
end

```

10.3 管道为树形图时的模型

当管道为树形图时, 建立与上面类似的非线性规划模型

$$\min \sum_{i=1}^7 \sum_{j=1}^{21} c_{ij} x_{ij} + 0.05 \sum_{j=1}^{21} \sum_{(jk) \in E} (y_{jk}^2 + y_{jk}) \quad (27)$$

$$\text{s. t. } 500 f_i \leq \sum_{j=1}^{21} x_{ij} \leq s_i f_i, \quad i = 1, 2, \dots, 7 \quad (28)$$

$$\sum_{i=1}^7 x_{ij} = \sum_{(jk) \in E} y_{jk}, \quad j = 1, 2, \dots, 21 \quad (29)$$

$$y_{jk} + y_{kj} = l_{jk}, \quad x_{ij}, y_{jk} \geq 0 \quad (30)$$

其中 (jk) 是连接 A_j, A_k 的边, E 是树形图的边集, l_{jk} 是从 A_j 到 A_k 的长度, y_{jk} 是由 A_j 沿 (jk) 铺设的钢管数量。

用 Lingo 求解得最小费用为 140.6631 亿元。Lingo 程序如下:

model:

sets:

-114-

```

! nodes 表示节点集合;
nodes
/S1,S2,S3,S4,S5,S6,S7,A1,A2,A3,A4,A5,A6,A7,A8,A9,A10,A11,A12,A13,A14,A15,A16,A
17,A18,A19,A20,B1,B2,B3,B4,B5,B6,B7,B8,B9,B10,B11,B12/;
!c1(i,j)表示节点 i 到 j 铁路运输的最小单位运价 (万元) ,c2(i,j)表示节点 i 到 j 公路运输
的邻接权重矩阵, c(i,j)表示节点 i 到 j 的最小单位运价, path 标志最短路径上走过的顶
点;
link(nodes, nodes): w, c1,c2,c,path1,path;
supply/S1..S7/:s,p,f;
need/A1..A21/:b,y,z;!y 表示每一点往节点编号小的方向铺设量, z 表示往节点编号大的
方向铺设量;
linkf(supply, need):cf,x;
special/1..3/:sx; ! 铺设节点 9, 11, 17 往最大编号节点方向的铺设量;
endsets
data:
s=800 800 1000 2000 2000 2000 3000;
p=160 155 155 160 155 150 160;
b=104,301,750,606,194,205,201,680,480,300,220,210,420,500,42,10,130,190,260,100,0;
path1=0; path=0; w=0; c2=0;
! 以下是格式化输出计算的中间结果和最终结果;
@text(MiddleCost.txt)=@writefor(supply(i): @writefor(need(j): @format(cf(i,j),' 6.1f' )),
@newline(1));
@text(Train_path.txt)=@writefor(nodes(i):@writefor(nodes(j):@format(path1(i,j),'5.0f')),
@newline(1));
@text(Final_path.txt)=@writefor(nodes(i):@writefor(nodes(j):@format(path(i,j),'5.0f')),
@newline(1));
@text(FinalResult.txt)=@writefor(supply(i):@writefor(need(j):@format(x(i,j),'5.0f')),
@newline(1) );
@text(FinalResult.txt)=@write(@newline(1));
@text(FinalResult.txt)=@writefor(need:@format(y,'5.0f'));
@text(FinalResult.txt)=@write(@newline(2));
@text(FinalResult.txt)=@writefor(need:@format(z,'5.0f'));
enddata
calc:
!输入铁路距离邻接矩阵的上三角元素;
w(28,30)=450;w(29,30)=80;w(30,32)=1150;w(31,33)=306;w(33,34)=195;w(1,34)=20;
w(1,35)=202;w(32,35)=1100;w(2,35)=1200;w(23,35)=720;w(3,23)=690;w(23,36)=520;
w(36,37)=170;w(4,37)=690;w(5,24)=462;w(24,37)=88;w(25,37)=160;w(25,26)=70;
w(25,27)=320;w(27,38)=160;w(6,38)=70;w(38,39)=290;w(7,39)=30;
@for(link(i,j): w(i,j) = w(i, j)+w(j,i) );!输入铁路距离邻接矩阵的下三角元素;
@for(link(i,j)|i#ne#j: w(i,j)=@if(w(i,j) #eq# 0, 20000,w(i,j))); ! 无铁路连接, 元素为充分
大的数;
!以下就是最短路计算公式 (Floyd-Warshall 算法) ;
@for(nodes(k):@for(nodes(i):@for(nodes(j):tm=@smin(w(i,j),w(i,k)+w(k,j));
path1(i,j)=@if(w(i,j)#gt# tm,k,path1(i,j));w(i,j)=tm));
!以下就是按最短路 w 查找相应运费 C1 的计算公式;
@for(link|w#eq#0: C1=0);
@for(link|w#gt#0 #and# w#le#300: C1=20);
@for(link|w#gt#300 #and# w#le#350: C1=23);

```



```

@for(link|w#gt#350 #and# w#le#400: C1=26);
@for(link|w#gt#400 #and# w#le#450: C1=29);
@for(link|w#gt#450 #and# w#le#500: C1=32);
@for(link|w#gt#500 #and# w#le#600: C1=37);
@for(link|w#gt#600 #and# w#le#700: C1=44);
@for(link|w#gt#700 #and# w#le#800: C1=50);
@for(link|w#gt#800 #and# w#le#900: C1=55);
@for(link|w#gt#900 #and# w#le#1000: C1=60);
@for(link|w#gt#1000: C1= 60+5*@floor(w/100-10)+@if(@mod(w,100)#eq#0,0,5) );
!输入公路距离邻接矩阵的上三角元素;
c2(8,9)=104;c2(9,10)=301;c2(10,11)=750;c2(11,12)=606;c2(12,13)=194;c2(13,14)=205;
c2(14,15)=201;c2(15,16)=680;c2(16,17)=480;c2(16,23)=42;c2(17,18)=300;c2(18,19)=220;
c2(18,24)=10;c2(19,20)=210;c2(20,21)=420;c2(21,22)=500;c2(24,25)=130;c2(24,26)=190;
c2(26,27)=260;c2(6,27)=100;c2(9,28)=3;c2(10,29)=2;c2(11,32)=600;c2(12,31)=10;
c2(13,33)=5;c2(14,34)=10;c2(1,14)=31;c2(15,35)=12;c2(17,36)=70;c2(19,26)=10;
c2(20,27)=62;c2(6,21)=110;c2(21,38)=30;c2(22,39)=20;c2(7,22)=20;
@for(link(i,j): c2(i,j) = c2(i,j)+c2(j,i)); !输入公路距离邻接矩阵的下三角元素;
@for(link(i,j):c2(i,j)=0.1*c2(i,j)); ! 距离转化成费用;
@for(link(i,j)|i#ne#j: c2(i,j) =@if(c2(i,j)#eq#0,10000,c2(i,j) ); ! 距离转化成费用;
@for(link: C= @if(C1#le#C2,C1,C2)); !邻接矩阵的对角线元素变为 0;
@for(nodes(k):@for(nodes(i):@for(nodes(j):tm=@smin(C(i,j),C(i,k)+C(k,j));
path(i,j)=@if(C(i,j)#gt# tm,k,path(i,j));C(i,j)=tm));
@for(link(i,j)|i #le# 7 #and# j#ge#8 #and# j#le# 27:cf(i,j-7)=c(i,j)); !提取下面二次规划模型需要的 7×21 矩阵;
@for(supply(i):cf(i,21)=c(i,6));
endcalc
[obj]min=@sum(linkf(i,j):(cf(i,j)+p(i))*x(i,j))+0.05*@sum(need(j):y(j)^2+y(j)+z(j)^2+z(j))+
0.05*@sum(special:sx^2+sx);
! 约束;
@for(supply(i):[con1]@sum(need(j):x(i,j))<= s(i)*f(i));
@for(supply(i):[con2]@sum(need(j):x(i,j) )>= 500*f(i));
@for(need(j)|j#ne#9 #and# j#ne#11 #and# j#ne#17:[con3] @sum(supply(i):x(i,j))=y(j)+z(j));
y(9)+z(9)+sx(1)=@sum(supply(i):x(i,9)); y(11)+z(11)+sx(2)=@sum(supply(i):x(i,11));
y(17)+z(17)+sx(3)=@sum(supply(i):x(i,17));
@for(need(j)|j #le# 14:(z(j)+y(j+1))=b(j));
@for(need(j)|j#ge#19 #and# j#le#20:z(j)+y(j+1)=b(j));
sx(1)+y(16)=42; sx(2)+y(17)=10; sx(3)+y(19)=190; z(17)+y(18)=130;
y(1)+z(15)+z(16)+z(18)+z(21)=0;
@for(supply: @bin(f)); @for(need: @gin(y));
end

```

习 题 五

1. 一只狼、一头山羊和一箩卷心菜在河的同侧。一个摆渡人要将它们运过河去，但由于船小，他一次只能运三者之一过河。显然，不管是狼和山羊，还是山羊和卷心菜，都不能在无人监视的情况下留在一起。问摆渡人应怎样把它们运过河去？

2. 北京 (Pe)、东京(T)、纽约(N)、墨西哥城(M)、伦敦(L)、巴黎(Pa)各城市之间的航线距离如表 16。

表 16 六城市间的航线距离

| | 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 | |

由上述交通网络的数据确定最小生成树。

3. 某台机器可连续工作 4 年，也可于每年末卖掉，换一台新的。已知于各年初购置一台新机器的价格及不同役龄机器年末的的处理价如表 17 所示。又新机器第一年运行及维修费为 0.3 万元，使用 1-3 年后机器每年的运行及维修费用分别为 0.8, 1.5, 2.0 万元。试确定该机器的最优更新策略，使 4 年内用于更换、购买及运行维修的总费用为最省。

表 17

| j | 第一年 | 第二年 | 第三年 | 第四年 |
|-----------------|-----|-----|-----|-----|
| 年初购置价 | 2.5 | 2.6 | 2.8 | 3.1 |
| 使用了 j 年的机器处理价 | 2.0 | 1.6 | 1.3 | 1.1 |

4. 某产品从仓库运往市场销售。已知各仓库的可供量、各市场需求量及从 i 仓库至 j 市场的路径的运输能力如表 18 所示（表中数字 0 代表无路可通），试求从仓库可运往市场的最大流量，各市场需求能否满足？

表 18

| 仓库 i \ 市场 j | 1 | 2 | 3 | 4 | 可供量 |
|-----------------|----|----|----|----|-----|
| A | 30 | 10 | 0 | 40 | 20 |
| B | 0 | 0 | 10 | 50 | 20 |
| C | 20 | 10 | 40 | 5 | 100 |
| 需求量 | 20 | 20 | 60 | 20 | |

5. 某单位招收懂俄、英、日、德、法文的翻译各一人，有 5 人应聘。已知乙懂俄文，甲、乙、丙、丁懂英文，甲、丙、丁懂日文，乙、戊懂德文，戊懂法文，问这 5 个人是否都能得到聘书？最多几个得到聘书，招聘后每人从事哪一方面翻译工作？

6. 表 19 给出某运输问题的产销平衡表与单位运价表。将此问题转化为最小费用最大流问题，画出网络图并求数值解。

表 19

| 产地 \ 销地 | 1 | 2 | 3 | 产量 |
|---------|----|----|----|----|
| A | 20 | 24 | 5 | 8 |
| B | 30 | 22 | 20 | 7 |
| 销量 | 4 | 5 | 6 | |

7. 求图 16 所示网络的最小费用最大流，弧旁数字为 (c_{ij}, u_{ij}) 。

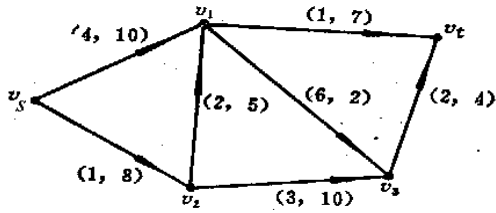


图 16 最小费用最大流

8. 某公司计划推出一种新型产品，需要完成的作业由表 20 所示。

表 20

| 作业 | 名称 | 计划完成时间(周) | 紧前作业 | 最短完成时间(周) | 缩短 1 周的费用(元) |
|----|----------|-----------|------|-----------|--------------|
| A | 设计产品 | 6 | — | 4 | 800 |
| B | 市场调查 | 5 | — | 3 | 600 |
| C | 原材料订货 | 3 | A | 1 | 300 |
| D | 原材料收购 | 2 | C | 1 | 600 |
| E | 建立产品设计规范 | 3 | A, D | 1 | 400 |
| F | 产品广告宣传 | 2 | B | 1 | 300 |
| G | 建立产品生产基地 | 4 | E | 2 | 200 |
| H | 产品运输到库 | 2 | G, F | 2 | 200 |

(1) 画出产品的计划网络图；

(2) 求完成新产品的最短时间，列出各项作业的最早开始时间、最迟开始时间和计划网络的关键路线；

(3) 假定公司计划在 17 周内推出该产品，各项作业的最短时间和缩短 1 周的费用如上表所示，求产品在 17 周内上市的最小费用；

(4) 如果各项作业的完成时间并不能完全确定，而是根据以往的经验估计出来的，其估计值如表 21 所示。试计算出产品在 21 周内上市的概率和以 95% 的概率完成新产品上市所需的周数。

表 21

| 作业 | A | B | C | D | E | F | G | H |
|--------|----|---|---|---|---|---|---|---|
| 最乐观的估计 | 2 | 4 | 2 | 1 | 1 | 3 | 2 | 1 |
| 最可能的估计 | 6 | 5 | 3 | 2 | 3 | 4 | 4 | 2 |
| 最悲观的估计 | 10 | 6 | 4 | 3 | 5 | 5 | 6 | 4 |

9. 已知下列网络图有关数据如表 22，设间接费用为 15 元/天，求最低成本日程。

表 22

| 工作代号 | 正常时间 | | 特急时间 | |
|------|--------|--------|--------|--------|
| | 工时 (d) | 费用 (元) | 工时 (d) | 费用 (元) |
| ①→② | 6 | 100 | 4 | 120 |
| ②→③ | 9 | 200 | 5 | 280 |
| ②→④ | 3 | 80 | 2 | 110 |
| ③→④ | 0 | 0 | 0 | 0 |
| ③→⑤ | 7 | 150 | 5 | 180 |
| ④→⑥ | 8 | 250 | 3 | 375 |

| | | | | |
|-----|---|-----|---|-----|
| ④→⑦ | 2 | 120 | 1 | 170 |
| ⑤→⑧ | 1 | 100 | 1 | 100 |
| ⑥→⑧ | 4 | 180 | 3 | 200 |
| ⑦→⑧ | 5 | 130 | 2 | 220 |