2019春季　电子工程学院

《通信网技术基础》

第四章

网络规划设计与图论

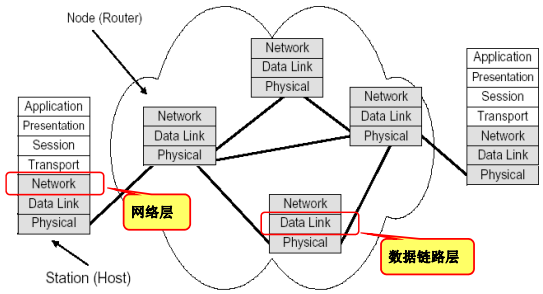
1

导语

- 为什么研究图论？

2

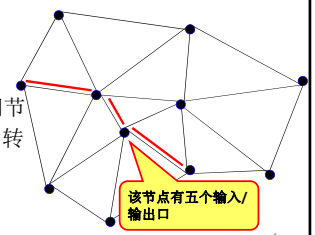
回顾：通信网络的层次化模型



3

交换和路由的概念

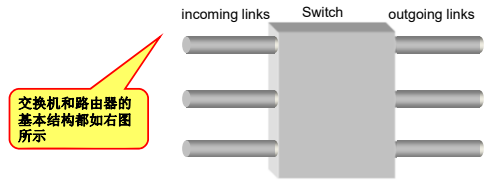
- 路由（Routing）：
  - 计算一条从数据**源节点**到**目的节点**的**路径（path）**
- 交换（Switching）
  - 把数据（包）从中间节点的一个端口（port）转发到另一个端口。



4

交换机和路由器

- 交换机：在**链路层**建立、维护、改变虚电路逻辑连接关系的设备
- 路由器：在**网络层**进行路由决策的智能交换设备

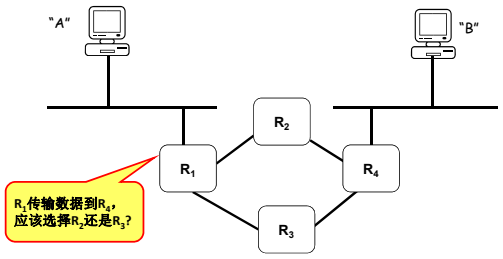


虚电路： 一种提供端对端连接关系的虚拟电路（不固定占用物理资源的链路）

5

路由问题的引出

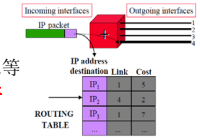
- 交换机/路由器只是在**路由表给定**的条件下执行数据包的转发
- 路由表如何得到？



6

路由的定义

- 路由的**功能**
  - 计算从源到目的节点的路径
  - 在路径沿途的交换（路由）节点上设置相应的状态信息（包括预留资源、更新路由表等）
- 路由的**元素**
  - 网络信息获取：网络拓扑、地址信息等
  - **路由算法**：计算到目的节点的所有**好**的路径
  - **信息转发**：通过交换设备将信息转发至目的节点



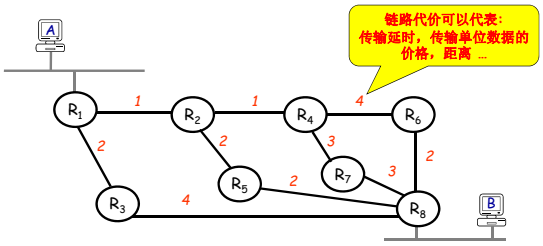
7

路由算法的性能评价

- **指标**：
  - 给定包长数据包的平均端到端延时
  - 传输带宽和链路利用率
  - 复杂度
  - 稳定性

8

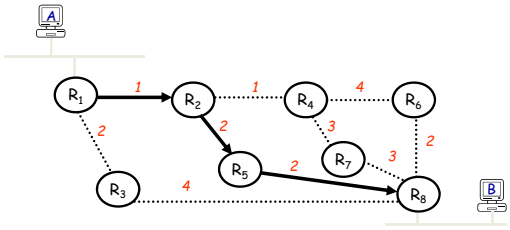
路由场景举例



目标：选择一条A到B的路径使得所经过链路的代价和最小

9

路由场景举例



这个例子的解很容易直观得到

10

2019春季　电子工程学院

第四章　网络规划设计理论基础

- 通信网的**拓扑结构**在通信网设计中是一个很重要的问题，它不但影响网的**造价和维护费用**，而且对网络的**可靠性**和网络的**控制及规划**起着重要的作用。
- 传统的网都是转接式的，包括电路转接和信息转接，是由**交换节点**和**传输线路**构成。从数学模型来说这是一个**图论**的问题。
- 在通信网的规划、设计和维护中，**可靠性**是一项重要的性能指标。

4.1 图论基础
4.2 树
4.3 路径选择算法
4.4 网络流量分配及其算法
4.5 通信网的可靠性

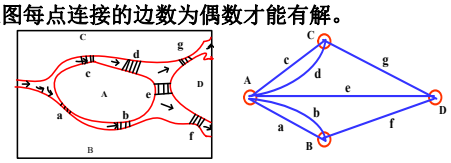
1. 图论的起源

·古普鲁士哥尼斯堡城的Pregel有七座桥将四块陆地相连。

·从任一陆地出发走遍七座桥（而且只走一次）再回到原地是否可行？

·1736年Euler（欧拉：圣彼得堡大学的数学教授）证明无解。

·连通图每点连接的边数为偶数才能有解。



4.1.1 图的概念

4.1.2 图的矩阵表示

❖图的概念

图论是专门研究人们在自然界和社会生活中遇到的包含某种二元关系的问题或系统。它把这种问题或系统抽象为点和线的集合，用点和线相互连接的图来表示，图4.1就是这样一个图，通常称为点线图。

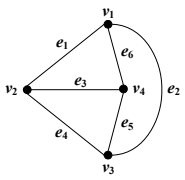


图4.1 图的概念

1. 图的定义

- 设有节点集$V=\{v_1, v_2, \ldots, v_n\}$和边集$E=\{e_1, e_2, \ldots, e_m\}$，当存在关系$R$，使$V\times V\to E$成立时，则说由节点集$V$和边集$E$组成图$G$，记为$G=(V, E)$。
- 关系R可以说成对任一边e_k，有V中的一个节点对(v_i, v_j)与之对应。图G中的V集可任意给定，而E集只是代表V中的二元关系。
  - 对$v_i\in V$，$v_j\in V$，当且仅当v_i对v_j存在某种关系时（如邻接关系）才有某一个$e_k\in E$。如果有一条边e_k与节点对(v_i, v_j)相对应，则称v_i，v_j是e_k的端点，记为$e_k=(v_i, v_j)$，称点v_i，v_j与边e_k关联，而称v_i与v_j为相邻节点。
  - 若有两条边与同一节点关联，则称这两条边为相邻边。

例4.1 图4.2中

$V=\{v1, v2, v3, v4\}$

$E=\{e1, e2, e3, e4, e5, e6\}$

其中$e1=(v1, v2)$，$e2=(v1, v3)$，

$e3=(v2, v4)$，$e4=(v2, v3)$，

$e5=(v3, v4)$，$e6=(v1, v4)$。

故可将图4.2记为$G=(V, E)$。

图中$v1$与$e1$, $e2$, $e6$关联，

$v1$与$v2$, $v3$, $v4$是相邻节点，

$e1$与$e2$, $e3$, $e4$, $e6$是相邻边。

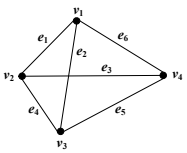
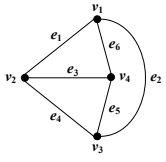
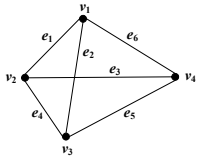


图4.2 图G

- 一个图可以用几何图形来表示，但一个图所对应的几何图形不是唯一的。
- 不难看出，图4.2表示的图与图4.1表示的图是相同的图G，说明一个图只由它的节点集V、边集E和点与边的关系所确定，而与节点的位置和边的长度及形状无关。
- 图4.1和图4.2只是一个图G的两种不同的几何表示方法。





2. 图的相关概念

- 节点：表示物理实体，用v_i表示。
- 边：节点间的连线，表示两节点间存在连接关系，用e_{ij}表示。
- 无向图：设图$G=(V, E)$，当v_i对v_j存在某种关系R等价于v_j对v_i存在某种关系R，则称G为无向图。
  - 即图G中的任意一条边e_k都对应一个无序节点对（v_i, v_j），（v_i, v_j）=（v_j, v_i）。如图4.3所示。

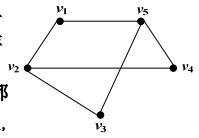


图4.3 无向图

- 有向图：设图$G=(V, E)$，当vi对vj存在关系R不等价于vj对vi存在关系R，则称G为有向图。即图G中的任意一条边都对应一个有序节点对（vi, vj），（vi, vj）≠（vj, vi）。如图4.4所示。
- 有权图：设图$G=(V, E)$，如果对它的每一条边e_k或对它的每个节点vi赋以一个实数p_k，则称图G为有权图或加权图，p_k称为权值。对于电路图，若节点为电路中的点，边为元件，则节点的权值可以为电压或电阻，边的权值为电流。对于通信网而言，节点可代表交换局，边代表链路，权值可以为长度，造价等，如图4.5所示。

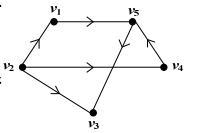


图4.4 有向图

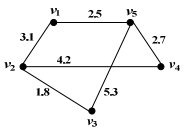


图4.5 有权图

- 自环：若与一个边er相关联的两个节点是同一个节点，则称边er为自环。
- 重边：在无向图中与同一对节点关联的两条或两条以上的边称为重边。在有向图中与同一对节点关联且方向相同的两条或两条以上的边称为重边。没有自环和重边的图称为简单图。

如图4.6（a）中，与边$e1$所关联的两个节点是同一个节点，这种边就为自环；而与$v2$、$v4$相关联的边有两条，即$e6$和$e7$，这就是重边，重边的重数也可为3或更多。

- 图4.6(b)中的$e6$和$e7$虽也同时与$v2$、$v4$相关联，但箭头方向不同，不能称为重边。在实际问题中，重边常可合并成一条边。对于一条无向边可画成两条方向相反的有向边，使有向图中没有无向边;也可将与同一对节点相关联的两条有向边合并成一条无向边。

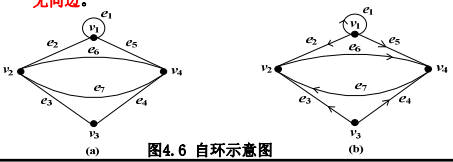


图4.6 自环示意图

- 节点的度数：与某节点相关联的边数可定义为该节点的度数，记为$d(v_i)$。如图4.6(a)中$d(v_2)=4$, $d(v_3)=2$, $d(v_1)=4$。若为有向图，用$d^+(v_i)$表示离开或从节点v_i射出的边数,即节点v_i的出度，用$d^-(v_i)$表示进入或射入节点v_i的边数即节点v_i的入度，而节点v_i的度数表示为$d(v_i)=d^+(v_i)+d^-(v_i)$。如图4.6(b)中，$d^+(v_1)=3$, $d^-(v_1)=1$, $d(v_1)=4$。

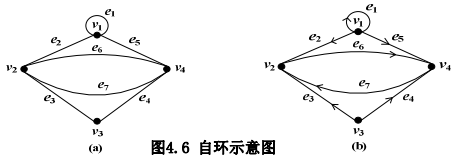


图4.6 自环示意图

- 边序列：有限条边的一种串序排列称为边序列，边序列中的各条边是首尾相连的，如图4.7中（$e1$, $e2$, $e3$, $e4$, $e5$, $e6$, $e3$）就是一个边序列。在边序列中，某条边是可以重复出现的，节点也是可以重复出现的。
- 链(chain)：没有重复边的边序列叫做链。在链中每条边只能出现一次。起点和终点不是同一节点的链叫开链。起点和终点重合的链叫闭链。通常所说的链指的是开链。链中边的数目称为链的长度。如图4.7中（e2, e3, e4）为闭链，而（e1, e2, e3, e6）为开链。

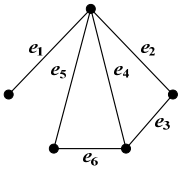


图4.7 边序列

- 径(path)：既无重复边，又无重复节点的边序列叫做径。在径中，每条边和每个节点都只出现一次。如图4.7 中（$e1$, $e2$, $e3$）即为一条径。
- 在一条径中，除了起点和终点的节点的度数为1外，其他节点的度数都是2。

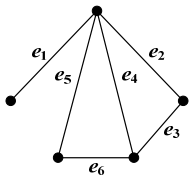


图4.7 边序列

- 链和径是不一样的，链中可以有重复的节点，而径中不能出现重复的节点，链中各节点的度数不定，而径中各节点度数是有规律的。

➢ 回路(circuit)：起点和终点重合的径称为回路（或称为圈），回路是每个节点度数均为2的连通图，如图4.7中（$e2, e3, e4$）是一回路。由定义可知，回路必为连通图。

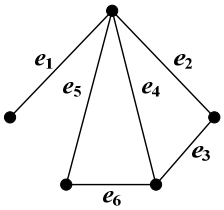
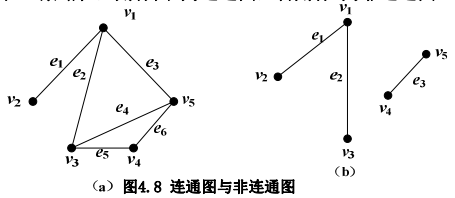


图4.7 边序列

3. 图的连通性

➢ 连通图：设图$G=(V,E)$，若图中任意两个节点之间至少存在一条路径，则称图G为连通图，否则称G为非连通图。



（a）图4.8 连通图与非连通图

在图4.8中，（a）为一连通图，（b）为一非连通图。

➢ 环路：回路间不重边的并称为环路。

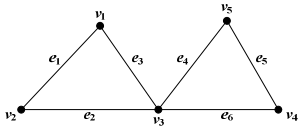


图4.9 环路

如图4.9中，（$e_1, e_2, e_6, e_5, e_4, e_3$）是环路，是回路（$e_1, e_2, e_3$）与回路（$e_4, e_5, e_6$）的不重边的并，它们有一个公共点$v_3$，是连通的。环路中每个节点的度数均为偶数。

➢ 子图、真子图和生成子图：
设有图$G=(V,E)$，$G'=(V',E')$ $V'\subseteq V$，$E'\subseteq E$，则称G'是G的子图，写成$G'\subseteq G$；若$V'\subseteq V$，$E'\subset E$，则称G'是G的真子图，写成$G'\subset G$；若$V'=V$，$E'\subseteq E$则称G'是G的生成子图。

➢ 最大连通子图：若图G'是图G的一个连通子图，但再加上一个属于原图G的任何一个其他元素，图G'就失去了连通性，成为非连通图，则图G'叫图G的最大连通子图。

- 从子图的定义可以看出，每个图都是它自己的子图。从原来的图中适当地去掉一些边和节点后得到子图。如果子图中不包含原图的所有边就是原图的真子图，若包含原图的所有节点的子图就是原图的生成子图。如图4.10中，（b）是（a）的真子图，（c）是（a）的生成子图，也是真子图。

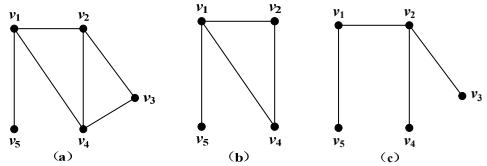


图4.10 图与子图

4.1.1 图的概念

4.1.2 图的矩阵表示

- 图的最直接的表示方法是用几何图形，且这种方法已经被广泛地应用。
- 但这种表示在数值计算和分析时有很大缺点，因此需借助于矩阵表示。这些矩阵是与几何图形一一对应的，即由图形可以写出矩阵，由矩阵也能画出图形。这样画出的图形可以不一样，但在拓扑上是一致的，也就是满足图的抽象定义。用矩阵表示的最大优点是可以存入计算机，并进行所需的运算。

1. 邻接矩阵

- 由节点与节点之间的关系确定的矩阵称为邻接矩阵。它的行和列都与节点相对应，因此对于一个有n个节点，m条边的图G，其邻接矩阵是一个$n\times n$的方阵，方阵中的每一行和每一列都与相应的节点对应，记作$C(G)=[c_{ij}]_{n\times n}$。

$$C(G)=\begin{array}{c} \\ v_1 \\ \vdots \\ v_i \\ \vdots \\ v_n \end{array}\begin{array}{c} \begin{array}{cccccc} v_1 & v_2 & \cdots & v_j & \cdots & v_n \end{array} \\ \begin{bmatrix} & & & \vdots & & \\ & & & \vdots & & \\ \cdots & \cdots & & c_{ij} & \cdots & \cdots \\ & & & \vdots & & \\ & & & \vdots & & \end{bmatrix}_{n\times n} \end{array}$$

- c_{ij}对于有向图：

$$C_{ij}=\begin{cases}1 & \text{若从}v_i\text{到}v_j\text{有边}\\ 0 & \text{若从}v_i\text{到}v_j\text{无边}\end{cases}$$

- c_{ij}对于无向图：

$$C_{ij}=C_{ji}=\begin{cases}1 & v_i\text{与}v_j\text{间有边}\\ 0 & v_i\text{与}v_j\text{间无边}\end{cases}$$

- 邻接矩阵C有如下特点：

（1）当图中无自环时，C阵的对角线上的元素都为0。若有自环，则对角线上对应的相应元素为1。

（2）有向图中，C阵中的每行上1的个数为该行所对应的节点的射出度数$d^+(v_i)$，每列上的1的个数则为该列所对应的节点的射入度数$d^-(v_i)$；无向图中，每行或每列上1的个数则为该节点的总度数$d(v_i)$。当某节点所对应的行和列均为零时说明该节点为孤立节点。

例4.2 求图4.11（a），（b）的邻接矩阵。

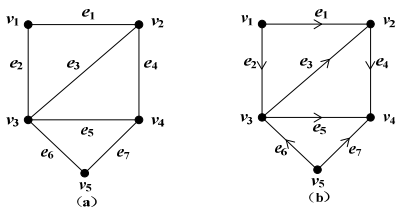


图4.11 图的矩阵表示

解： 图4.11(a)的邻接矩阵为

$$C(G_1)=\begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{array}\begin{array}{c} \begin{array}{ccccc} v_1 & v_2 & v_3 & v_4 & v_5 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix} \end{array}$$

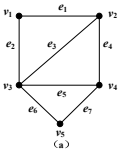
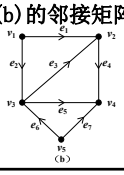


图4.11(b)的邻接矩阵为



$$C(G_2)=\begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{array}\begin{array}{c} \begin{array}{ccccc} v_1 & v_2 & v_3 & v_4 & v_5 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix} \end{array}$$

- 对于无向简单图，邻接矩阵是对称的，且对角线上的元素全为零。
- 对于有向简单图，即没有自环和同方向并行边的有向图，对角线元素也为零，但邻接矩阵不一定对称。

2．权值矩阵

- 设G为有权图，而且是具有n个节点的简单图，其权值矩阵为

$$W(G)=(w_{ij})_{n\times n}=\begin{array}{c} \\ v_1 \\ v_2 \\ \vdots \\ v_i \\ \vdots \\ v_n \end{array}\begin{array}{c} \begin{array}{cccccc} v_1 & v_2 & \cdots & v_j & \cdots & v_n \end{array} \\ \begin{bmatrix} & & \cdots & & & \\ & & \cdots & & & \\ \cdots & \cdots & \cdots & w_{ij} & \cdots & \cdots \\ & & \cdots & & & \\ & & \cdots & & & \end{bmatrix}_{n\times n} \end{array}$$

$$w_{ij}=\begin{cases} p_{ij} & v_i\text{与（到）}v_j\text{有边，}p_{ij}\text{为权值} \\ \infty & v_i\text{与（到）}v_j\text{无边} \\ 0 & i=j \end{cases}$$

- 无向简单图的权值矩阵是对称的，对角线元素全为零。
- 有向简单图的权值矩阵不一定对称，但对角线元素也全为零。

- 图4.12和图4.13的权值矩阵$W(G1)$和$W(G2)$分别为：

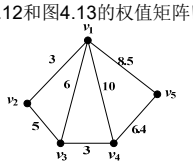


图4.12 无向图G1

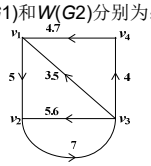


图4.13 有向图G2

$$W(G_1)=\begin{bmatrix} 0 & 3 & 6 & 10 & 8.5 \\ 3 & 0 & 5 & \infty & \infty \\ 6 & 5 & 0 & 3 & \infty \\ 10 & \infty & 3 & 0 & 6.4 \\ 8.5 & \infty & \infty & 6.4 & 0 \end{bmatrix}$$

对称

$$W(G_2)=\begin{bmatrix} 0 & 5 & \infty & \infty \\ \infty & 0 & 7 & \infty \\ 3.5 & 5.6 & 0 & 4 \\ 4.7 & \infty & \infty & 0 \end{bmatrix}$$

不对称

4.1 图论基础
4.2 树
4.3 路径选择算法
4.4 网络流量分配及其算法
4.5 通信网的可靠性

4.2.1 树的概念及性质
4.2.2 图的生成树及其求法
4.2.3 最小生成树算法

1. 定义

- 任何两节点间有且只有一条径的图称为**树**，树中的边称为**树枝**（branch）。若树枝的两个节点都至少与两条边关联，则称该树枝为**树干**；若树枝的一个节点仅与此边关联，则称该树枝为**树尖**，并称该节点为**树叶**。若指定树中的一个点为**根**，则称该树为**有根树** 。

- 图4.14所示为一棵**树**，v_1为**树根**，e_1，e_2，e_3，e_5 e_6等为**树干**，e_7，e_4，e_8，e_9，e_{10}，e_{11}等为**树尖**，v_6，v_7，v_8，v_{10}等为**树叶**。

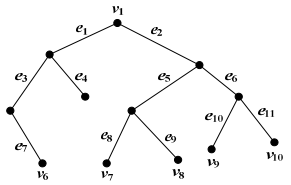


图4.14 树

2. 性质

- **树是无环的连通图，但增加一条边便可以得到一个环**。任何两节点间有径的图一定是连通图，而只有一条径就不能有环。
- **树是最小连通图**，即去掉树中的任何一条边就成为非连通图，丧失了连通性。
- **若树有m条边及n个节点，则有$m=n-1$**，即有n个节点的树共有$n-1$个树枝。
- **除了单点树外，任何一棵树中至少有两片树叶**。

4.2.1 树的概念及性质

4.2.2 图的生成树及其求法

4.2.3 最小生成树算法

1. 图的生成树

- 设G是一个连通图，T是G的一个子图且是一棵树，若T包含G的所有节点，则称T是G的一棵**生成树**，也称**支撑树**。

- 只有连通图才有生成树；反之，有生成树的图必为连通图。
- 图G的生成树上的边组成**树枝集**。生成树之外的边称为**连枝**，连枝的边集称为**连枝集**或称为**树补**。如果在生成树上加一条连枝，便会形成一个回路。若图G本身不是树，则G的生成树不止一个，而连通图至少有一棵生成树。
- 连通图G的生成树T的**树枝数**称为图G的**阶**。如果图G有n个节点，则它的阶ρ是$\rho(G)=\rho=n-1$。

- 具有n个节点、m条边的连通图，生成树T有$n-1$条树枝和$m-n+1$条连枝。
- 连枝集的连枝数称为图G的**空度**，记为μ，当G有m条边时，有

$$\mu(G)=\mu=|G-T|=m-n+1$$

显然有

$$m=\rho+\mu$$

阶　　空度

- 图的阶 ρ 表示生成树的大小，取决于 G 中的节点数。
- 图的空度表示生成树覆盖该图的程度， μ 越小，覆盖度越高， μ =0表示图 G 就是树。另一方面，空度 μ 也反映图 G 的连通程度， μ 越大，连枝数越多，图的连通性越好， μ =0表示图 G 有最低连通性，即最小连通图。

2. 生成树的求法

- **破圈法**：拆除图中的所有回路并使其保持连通，就能得到 G 的一棵生成树。
- **避圈法**：在有 n 个点的连通图 G 中任选一条边（及其节点）；选取第二、三…条边，使之不与已选的边形成回路；直到选取完 n-1条边且不出现回路，结束。

例4.3 分别用破圈法和避圈法选择图4.15（a）的一棵树。

解： 破圈法：如图4.15所示，选择回路（$v1$，$v3$，$v4$），去掉 e_1；选择回路（$v1$，$v2$，$v3$），去掉 e_3；选择回路（$v2$，$v3$，$v5$），去掉 $e6$；最后选择回路（$v3$，$v4$，$v6$，$v5$），去掉 $e9$ 依次得到图4.15（b）-（e），其中（e）为（a）的一棵生成树。

避圈法：依次选取五条边 $e3$，$e4$，$e7$，$e9$，$e8$，每一条边均不与已选边形成回路，见图4.16，最后得到 G 的又一棵生成树（e）。

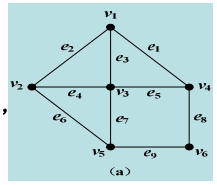


图4.15（a）

解： 破圈法：如图4.15所示，选择回路（$v1$，$v3$，$v4$），去掉 e_1；选择回路（$v1$，$v2$，$v3$），去掉 e_3；选择回路（$v2$，$v3$，$v5$），去掉 $e6$；最后选择回路（$v3$，$v4$，$v6$，$v5$），去掉 $e9$ 依次得到图4.15（b）-（e），其中（e）为（a）的一棵生成树。

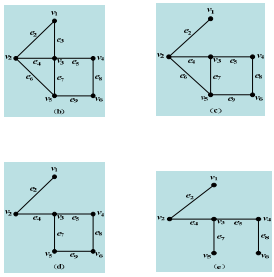


图4.15 破圈法示意图

避圈法：依次选取五条边 $e3$，$e4$，$e7$，$e9$，$e8$，每一条边均不与已选边形成回路，见图4.16，最后得到 G 的又一棵生成树（e）。

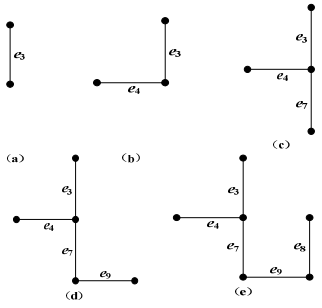


图4.16 避圈法示意图

4.2.1 树的概念及性质

4.2.2 图的生成树及其求法

4.2.3 最小生成树算法

- **最小生成树**：如果连通图G本身不是一棵树，则它的生成树就不止一棵。如果为图G加上权值，则各个生成树的树枝权值之和一般不相同，其中权值之和最小的那棵生成树为最小生成树。
- 最小生成树一般是在两种情况下提出的，一种是有约束条件下的最小生成树，另一种是无约束条件下的最小生成树。

1．无约束条件的情况

- **Kruskal算法**（克鲁斯格）

①将连通图G中的所有边按权值的非减次序排列；
②选取权值最小的边为树枝，再按①的次序依次选取不与已选树枝形成回路的边为树枝。如有几条这样的边权值相同则任选其中一条；
③对于有n个点的图直到n-1条树枝选出，结束。

这种算法的复杂性主要决定于把各边排列成有序的队列。

- **Prim算法**

① 写出图G的权值矩阵；
② 由点v_1开始，在行1中找出最小元素w_{1j}；
③ 在行1和行j中，圈去列1和列j的元素，并在这两行余下的元素中找出最小元素，如w_{jk}（如有两个均为最小元素可任选一个）；
④ 在行1、行j和行k中，圈去列1、列j和列k的元素，并在这三行余下的元素中找出最小元素；
⑤ 直到矩阵中所有元素均被圈去，即找到图G的一棵最小生成树。

例4.4 要建设连接如图4.17所示的七个城镇的线路网，任意两个城镇间的距离见表4.1，请用P算法找出线路费用最小的网路结构图（设线路费用与线路长度成正比）。

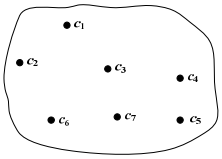


图4.17 七个城市的地图

表4.1 各城镇间的距离（km）

	C_2	C_3	C_4	C_5	C_6	C_7
C_1	8	5	9	12	14	12
C_2		9	15	17	8	11
C_3			7	9	11	7
C_4				3	17	10
C_5					8	10
C_6						9

解：这个问题可抽象为用图论求最小生成树的问题，首先列出权值矩阵

$$W=\begin{bmatrix} 0 & 8 & 5 & 9 & 12 & 14 & 12 \\ 8 & 0 & 9 & 15 & 17 & 8 & 11 \\ 5 & 9 & 0 & 7 & 9 & 11 & 7 \\ 9 & 15 & 7 & 0 & 3 & 17 & 10 \\ 12 & 17 & 9 & 3 & 0 & 8 & 10 \\ 14 & 8 & 11 & 17 & 8 & 0 & 9 \\ 12 & 11 & 7 & 10 & 10 & 9 & 0 \end{bmatrix}$$

在第一行中找出最小元素5，圈去第1行和第3行中第1列和第3列的元素，在这两行余下的元素中找到最小元素7，再圈去第1行、第3行和第4行中第1列、第3列和第4列的元素，从这3行余下的元素中找到最小元素为3，重复上述过程，依次找到7，8，8，将这些最小元素对应的边和节点全部画出就可得到一棵最小生成树，如图4.18所示。

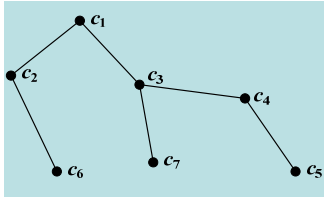


图4.18 最小费用网络结构图

所以，费用最小的网路结构网路总长度L为

L=3+5+7+7+8+8=38 km

2. 有约束条件的最小生成树

- 在设计通信网的网路结构时，经常会提出一些特殊的要求，如某交换中心或某段线路上的业务量不能过大，任意两点间经过转接的次数不能过多等，这类问题可归结为求有约束条件的最小生成树的问题。
- 关于有约束条件的最小生成树的求法目前并没有一般的有效算法，而且不同的约束条件，算法也将有区别。这里，我们只介绍一种常用的解决有约束条件的生成树的方法，即穷举法。
- 穷举法就是先把图中的所有生成树穷举出来，再按条件筛选，最后选出最短的符合条件的生成树。显然这是一种最直观的也是最繁杂的方法，虽然可以得到最佳解，但计算量往往很大。

4.1 图论基础
4.2 树
4.3 路径选择算法
4.4 网络流量分配及其算法
4.5 通信网的可靠性

4.3.1 狄克斯特拉（Dijkstra）算法
4.3.2 Warshall-Floyd算法
4.3.3 第K条最短路径选择问题

1. D算法原理

- 已知图$G=(V, E)$，将其节点集分为两组：置定节点集G_p和未置定节点集$G-G_p$。其中G_p内的所有置定节点，是指定点v_s到这些节点的路径为最短（即已完成最短路径的计算）的节点。而$G-G_p$内的节点是未置定节点，即v_s到未置定节点距离是暂时的，随着算法的下一步将进行不断调整，使其成为最短径。
- 在调整各未置定节点的最短径时，是将G_p中的节点作为转接点。具体地说，就是将G_p中的节点作为转接点，计算（v_s, v_j）的径长（$v_j \in G-G_p$），若该次计算的径长小于上次的值，则更新径长，否则，径长不变。计算后取其中径长最短者，之后将v_j划归到G_p中。当（$G-G_p$）最终成为空集，同时$G_p=G$，即求得v_s到所有其他节点的最短路径。

- w_j：表示v_s与其他节点的距离。
- 在G_p中，w_i表示上一次划分到G_p中的节点v_i到v_s的最短路径。
- 在G-G_p中，表示从v_s到v_j仅经过G_p中的节点作为转接点所求得的该次的最短路径的长度。
- 如果v_s与v_j不直接相连，且无置定节点作为转接点，则令$w_j = \infty$ 。

2. D算法实现流程

某一个节点到所有节点的最短距离

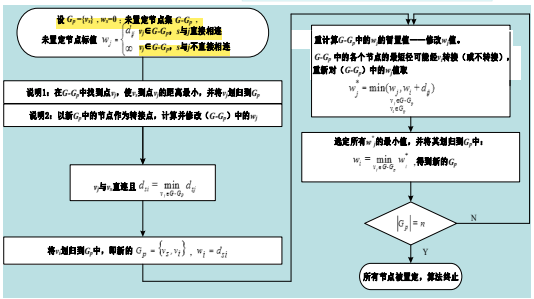


图4.19 D算法流程图

例4.5 用D算法求图4.20中v_1到其他各节点的最短路径。

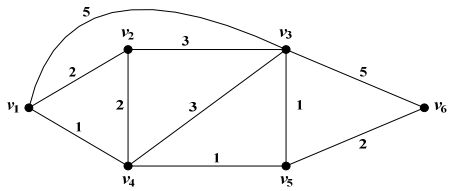


图4.20 D算法例题图

解：计算过程及结果列于表4.2及表4.3中。最终路径图如图4.21所示。

表4.2 D算法计算过程

迭代次数 j	v_1 v_2 v_3 v_4 v_5 v_6	置定节点	w_i	G_i
0	(0) 2 5 1 ∞ ∞	v_1	$w_1=0$	$\{v_1\}$
1	2 5 ① ∞ ∞	v_4	$w_4=1$	$\{v_1,v_4\}$
2	② 4 2 ∞	v_2	$w_2=2$	$\{v_1,v_4,v_2\}$
3	4 ② ∞	v_5	$w_5=2$	$\{v_1,v_4,v_2,v_5\}$
4	③ 4	v_3	$w_3=3$	$\{v_1,v_4,v_2,v_5,v_3\}$
5	④	v_6	$w_6=4$	$\{v_1,v_4,v_2,v_5,v_3,v_6\}$

表4.3 v1到其他各节点的最短路径和径长

节点	v_1	v_2	v_3	v_4	v_5	v_6
最短路径	(v_1)	(v_2)	(v_4,v_3)	(v_4)	(v_4,v_5)	(v_4,v_5,v_6)
径长	0	2	3	1	2	4

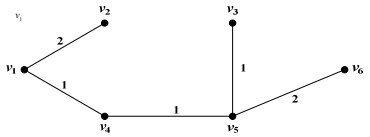


图4.21 v1到其他各节点的最短路径

链路状态路由：LSR

- Link State Routing
  - 每个节点维护到所有邻居节点的距离
  - 该距离信息(link state)被广播到网络中的所有节点
  - 每个节点独立计算并维护路由表
- 优点
  - 更加稳定、且比距离矢量法的收敛速度更快
  - 容易发现网络拓扑，易于网络维护
  - 易于实现源路由(source-routing)及Quality-of-service routing (multiple route tables)

70

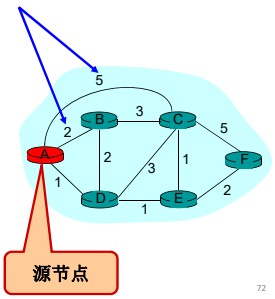
典型的 LSR: Dijkstra's Algorithm

- 基本原理:
  - 每个节点维护从源端S迄今为止建立的最短路径信息 (*previous hop, length*)
  - 每一步选择局部最优路径，最终收敛到全局最优 *(chooses local optimum at each step, which leads to global optimum)*
- 算法:
  - 初始化:起点 (S,0)，其它节点 (*, ∞)
  - 重复下列操作，直到所有节点均已完成
    - 选择一个拥有最短路径的未完成节点 U
    - 标记节点U为完成
    - 针对该节点的所有未完成邻居节点
      - *length*(V) = min[*length*(V), *length*(U) + *d*(U,V)]
      - and correspondingly update *previous hop*

71

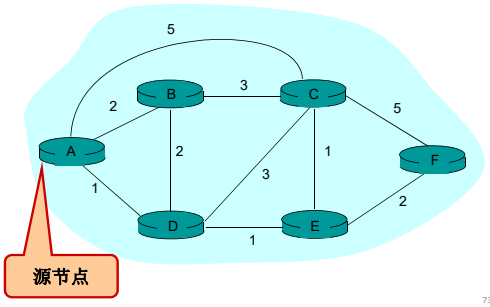
一些标记

- c(i,j): 从节点 *i* 到节点 *j*的链路代价; ≥ 0；若两者间无链路，则代价无穷大
- D(v): 当前从源节点到节点v的路径代价
- p(v): 从源节点到节点v的所选路径上，v的相邻节点
- S: 源节点到该集合S中所有节点的最小路径代价已知（即已经完成的节点）。



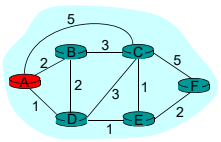
72

Dijkstra算法：举例



Dijkstra算法：举例

Step	start S	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1						
2						
3						
4						
5						

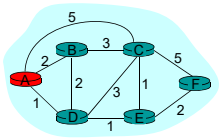


```
1  Initialization:
2   S = {A};
3   for all nodes v
4     if v adjacent to A
5       then D(v) = c(A,v);
6       else D(v) = ∞;
…
```

红色的节点代表“Done”

Dijkstra算法：举例

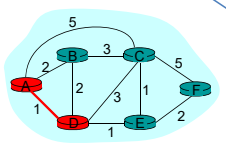
Step	start S	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1						
2						
3						
4						
5						



```
…
8  Loop
9   find w not in S s.t. D(w) is a minimum;
10  add w to S;
11  update D(v) for all v adjacent
      to w and not in S:
12  If D(w) + c(w,v) < D(v) then
13    D(v) = D(w) + c(w,v); p(v) = w;
14  until all nodes in S;
```

Dijkstra算法：举例

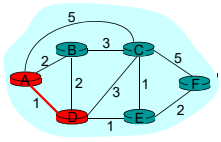
Step	start S	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1	AD					
2						
3						
4						
5						



```
…
8  Loop
9   find w not in S s.t. D(w) is a minimum;
10  add w to S;
11  update D(v) for all v adjacent
      to w and not in S:
12  If D(w) + c(w,v) < D(v) then
13    D(v) = D(w) + c(w,v); p(v) = w;
14  until all nodes in S;
```

Dijkstra算法：举例

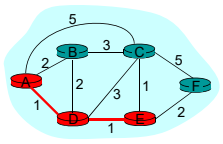
Step	start S	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1	AD		4,D		2,D	
2						
3						
4						
5						



```
…
8  Loop
9   find w not in S s.t. D(w) is a minimum;
10  add w to S;
11  update D(v) for all v adjacent
      to w and not in S:
12  If D(w) + c(w,v) < D(v) then
13    D(v) = D(w) + c(w,v); p(v) = w;
14  until all nodes in S;
```

Dijkstra算法：举例

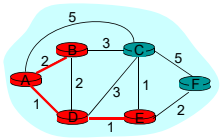
Step	start S	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1	AD		4,D		2,D	
2	ADE		3,E			4,E
3						
4						
5						



```
…
8  Loop
9   find w not in S s.t. D(w) is a minimum;
10  add w to S;
11  update D(v) for all v adjacent
      to w and not in S:
12  If D(w) + c(w,v) < D(v) then
13    D(v) = D(w) + c(w,v); p(v) = w;
14  until all nodes in S;
```

Dijkstra算法：举例

Step	start S	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1	AD		4,D		2,D	
2	ADE		3,E			4,E
3	ADEB					
4						
5						

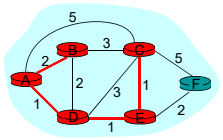



```
…
8  Loop
9    find w not in S s.t. D(w) is a minimum;
10   add w to S;
11   update D(v) for all v adjacent
       to w and not in S:
12   If D(w) + c(w,v) < D(v) then
13     D(v) = D(w) + c(w,v); p(v) = w;
14 until all nodes in S;
```

79

Dijkstra算法：举例

Step	start S	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1	AD		4,D		2,D	
2	ADE		3,E			4,E
3	ADEB					
4	ADEBC					
5						

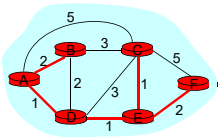



```
…
8  Loop
9    find w not in S s.t. D(w) is a minimum;
10   add w to S;
11   update D(v) for all v adjacent
       to w and not in S:
12   If D(w) + c(w,v) < D(v) then
13     D(v) = D(w) + c(w,v); p(v) = w;
14 until all nodes in S;
```

80

Dijkstra算法：举例

Step	start S	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1	AD		4,D		2,D	
2	ADE		3,E			4,E
3	ADEB					
4	ADEBC					
5	ADEBCF					

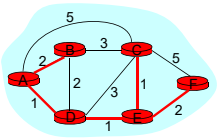



```
…
8  Loop
9    find w not in S s.t. D(w) is a minimum;
10   add w to S;
11   update D(v) for all v adjacent
       to w and not in S:
12   If D(w) + c(w,v) < D(v) then
13     D(v) = D(w) + c(w,v); p(v) = w;
14 until all nodes in S;
```

81

Dijkstra算法：举例

Step	start S	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1	AD		4,D		2,D	
2	ADE		3,E			4,E
3	ADEB					
4	ADEBC					
5	ADEBCF					

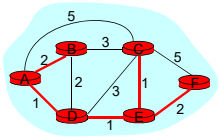


To determine path A → C (say), work backward from C via p(v)

82

路由转发表

- 针对节点A执行完毕 Dijkstra 算法后得到一系列从A到达所有节点的最短路径
- 由此可以得到A节点的**路由转发表**



Destination	Link
B	(A,B)
C	(A,D)
D	(A,D)
E	(A,D)
F	(A,D)

83

4.3.1 狄克斯特拉（Dijkstra）算法

4.3.2 Warshall-Floyd算法

4.3.3 第K条最短路径选择问题

1. F算法的基本思路

- F算法使用距离矩阵和路由矩阵。
- 距离矩阵是一个$n\times n$矩阵，以图G的n个节点为行和列。记为$W=[w_{ij}]_{n\times n}$，w_{ij}表示图G中v_i和v_j两点之间的路径长度。
- 路由矩阵是一个$n\times n$矩阵，以图G的n个节点为行和列。记为$R=[r_{ij}]_{n\times n}$，其中r_{ij}表示v_i至v_j经过的转接点（中间节点）。

➢ F算法的思路是首先写出初始的W阵和R阵，接着按顺序依次将节点集中的各个节点作为中间节点，计算此点距其他各点的径长，每次计算后都以求得的与上次相比较小的径长去更新前一次较大径长，若后求得的径长比前次径长大或相等则不变。以此不断更新和，直至W中的数值收敛。

2. F算法实现流程

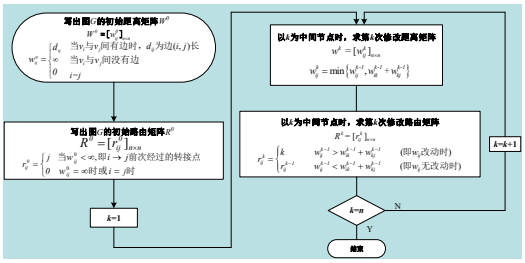


图4.22 F算法流程图

例4.6 用F算法求图4.23 中任意点之间的最短路径。

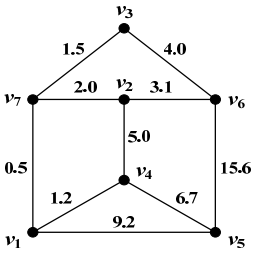


图4.23 F算法例题图

解：计算结果如下：

（1）初始化距离矩阵W^0和路由矩阵R^0。

$$W^0=\begin{array}{c}\\ v_1\\ v_2\\ v_3\\ v_4\\ v_5\\ v_6\\ v_7\end{array}\begin{array}{c}\begin{array}{ccccccc}v_1 & v_2 & v_3 & v_4 & v_5 & v_6 & v_7\end{array}\\ \begin{bmatrix}0&\infty&\infty&1.2&9.2&\infty&0.5\\ \infty&0&\infty&5&\infty&3.1&2\\ \infty&\infty&0&\infty&\infty&4&1.5\\ 1.2&5&\infty&0&6.7&\infty&\infty\\ 9.2&\infty&\infty&6.7&0&15.6&\infty\\ \infty&3.1&4&\infty&15.6&0&\infty\\ 0.5&2&1.5&\infty&\infty&\infty&0\end{bmatrix}\end{array}$$

$$R^0=\begin{array}{c}\\ v_1\\ v_2\\ v_3\\ v_4\\ v_5\\ v_6\\ v_7\end{array}\begin{array}{c}\begin{array}{ccccccc}v_1 & v_2 & v_3 & v_4 & v_5 & v_6 & v_7\end{array}\\ \begin{bmatrix}0&0&0&4&5&0&7\\ 0&0&0&4&0&6&7\\ 0&0&0&0&0&6&7\\ 1&2&0&0&5&0&0\\ 1&0&0&4&0&6&0\\ 0&2&3&0&5&0&0\\ 1&2&3&0&0&0&0\end{bmatrix}\end{array}$$

（2）依次以$v_1, v_2, \ldots, v_7$为中间节点修改W阵和R阵，结果如下：

$$W^1=\begin{bmatrix}0&\infty&\infty&1.2&9.2&\infty&0.5\\ \infty&0&\infty&5&\infty&3.1&2\\ \infty&\infty&0&\infty&\infty&4&1.5\\ 1.2&5&\infty&0&6.7&\infty&1.7\\ 9.2&\infty&\infty&6.7&0&15.6&9.7\\ \infty&3.1&4&\infty&15.6&0&\infty\\ 0.5&2&1.5&1.7&9.7&\infty&0\end{bmatrix}\quad R^1=\begin{bmatrix}0&0&0&4&5&0&7\\ 0&0&0&4&0&6&7\\ 0&0&0&0&0&6&7\\ 1&2&0&0&5&0&1\\ 1&0&0&4&0&6&1\\ 0&2&3&0&5&0&0\\ 1&2&3&1&1&0&0\end{bmatrix}$$

之前为：无穷

$$W^2=\begin{bmatrix}0&\infty&\infty&1.2&9.2&\infty&0.5\\ \infty&0&\infty&5&\infty&3.1&2\\ \infty&\infty&0&\infty&\infty&4&1.5\\ 1.2&5&\infty&0&6.7&8.1&1.7\\ 9.2&\infty&\infty&6.7&0&15.6&9.7\\ \infty&3.1&4&8.1&15.6&0&5.1\\ 0.5&2&1.5&1.7&9.7&5.1&0\end{bmatrix}\quad R^2=\begin{bmatrix}0&0&0&4&5&0&7\\ 0&0&0&4&0&6&7\\ 0&0&0&0&0&6&7\\ 1&2&0&0&5&2&1\\ 1&0&0&4&0&6&1\\ 0&2&3&2&5&0&2\\ 1&2&3&1&1&2&0\end{bmatrix}$$

之前为：无穷

4-6→4-2-6
5+3.1=8.1

$W^3=W^2$，$R^3=R^2$

2+3.1=5.1
7-6→7-2-6

4.3.2 Warshall-Floyd算法

$W^4=\begin{bmatrix}0&\infty&\infty&1.2&9.2&\infty&0.5\\\infty&0&\infty&5&\infty&3.1&2\\\infty&\infty&0&\infty&\infty&4&1.5\\1.2&5&\infty&0&6.7&8.1&1.7\\9.2&\infty&\infty&6.7&0&15.6&9.7\\\infty&3.1&4&8.1&15.6&0&5.1\\0.5&2&1.5&1.7&9.7&5.1&0\end{bmatrix}$ $R^4=\begin{bmatrix}0&0&0&4&5&0&7\\0&0&0&4&0&6&7\\0&0&0&0&0&6&7\\1&2&0&0&5&2&1\\1&0&0&4&0&6&1\\0&2&3&2&5&0&2\\1&2&3&1&1&2&0\end{bmatrix}$

$W^3=W^5=W^4$，$R^5=R^4=R^3$

4.3.2 Warshall-Floyd算法

$W^6=\begin{bmatrix}0&6.2&13.3&1.2&7.9&9.3&0.5\\6.2&0&7.1&5&11.7&3.1&2\\13.3&7.1&0&12.1&18.8&4&1.5\\1.2&5&12.1&0&6.7&8.1&1.7\\7.9&11.7&18.8&6.7&0&14.8&8.4\\9.3&3.1&4&8.1&14.8&0&5.1\\0.5&2&1.5&1.7&8.4&5.1&0\end{bmatrix}$ $R^6=\begin{bmatrix}0&4&6&4&4&4&7\\4&0&6&4&4&6&7\\6&6&0&6&6&6&7\\1&2&6&0&5&2&1\\4&4&6&4&0&4&4\\4&2&3&2&4&0&2\\1&2&3&1&4&2&0\end{bmatrix}$

W_34→3-6-4
4+8.1=12.1

4.3.2 Warshall-Floyd算法

$W^7=\begin{bmatrix}0&2.5&2&1.2&7.9&5.6&0.5\\2.5&0&3.5&3.7&10.4&3.1&2\\2&3.5&0&3.2&9.9&4&1.5\\1.2&3.7&3.2&0&6.7&6.8&1.7\\7.9&10.4&9.9&6.7&0&13.5&8.4\\5.6&3.1&4&6.8&13.5&0&5.1\\0.5&2&1.5&1.7&8.4&5.1&0\end{bmatrix}$ $R^7=\begin{bmatrix}0&7&7&4&4&7&7\\7&0&7&7&7&6&7\\7&7&0&7&7&6&7\\1&7&7&0&5&7&1\\4&7&7&4&0&7&4\\7&2&3&7&7&0&2\\1&2&3&1&4&2&0\end{bmatrix}$

从W^7和R^7可以找到任何节点间最短径的径长和路由。

4.3 路径选择算法

4.3.1 狄克斯特拉（Dijkstra）算法

4.3.2 Warshall-Floyd算法

4.3.3 第K条最短路径选择问题

4.3.3 第K条最短路径选择问题

- 最短路径通常为信息传输的首选路由，如果该路由上有业务量溢出或发生故障，就要寻找迂回路由。迂回路由应依次选择次最短路径，第三条最短路径等，这就是研究第K条最短路径要解决的问题。

4.3.3 第K条最短路径选择问题

- 第*K*条最短路径可分为两类：一类是两点之间边分离的第*K*条最短路径；一类是两点之间点分离的第K条最短路径。在这里，边分离径是指无公共边但有公共点的径，如图4.24中P_1和P_2所示，点分离径是指除了起点和终点外无公共点的径，如图4.24中的P_1和P_3所示。
- 第一类的求法是将最短路径中的所有边去掉，用D算法在剩下的图中求出次最短路径，再依照此方法求出第三条最短路径，等等；
- 第二类的求法是将最短路径中的所有节点去掉，在剩下的图中求出次最短路径，同样依照此方法求出其他最短路径。当剩下的图中两点间不存在路径时，结束。

- P1：x→v1→v2→y
- P2：x→v3→v2→v4→y
- P3：x→v5→v6→y

边分离

点分离

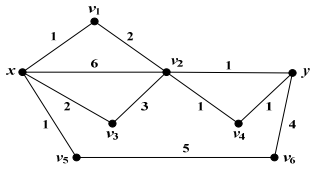


图4.24 边分离径与点分离径

4.1 图论基础
4.2 树
4.3 路径选择算法
4.4 网络流量分配及其算法
4.5 通信网的可靠性

4.4.1 流量分配的相关概念 平均流量
4.4.2 网络最大流算法——标号法
4.4.3 最佳流问题

1. 网络的定义

➢ 设N=（V,E）为有向图，它有两个非空不相交的节点子集X，Y。X中的节点称为源，Y中的节点称为宿，其他节点称为中间节点，在边集E（N）上定义一个取非负整数值的函数C，则称N为一个网络。

➢ 函数C称为N的容量函数，函数C在边eij=（v_i,v_j）的值叫边eij的容量，记为C（e_{ij}）或C（i,j）。一般来说C（i,j）≠C（j,i）。

2. 单源单宿网络

➢ 满足以下条件的网络叫单源单宿网络。

- 网络中有且只有一个节点，其$d^-(v)=0$，称该节点为源（发送节点）。 入度为0
- 网络中有且只有一个节点，其$d^+(v)=0$，称该节点为宿（接收节点）。 出度为0

➢ 设N为有一个源x和一个宿y的网络，f是定义在边集E（N）上的一个实数函数，V1，V2是V的子集，用（V1,V2）表示起点在V1中，终点在V2中的边的集合.

$$f(v_1,v_2)=\sum_{e\in(v_1,v_2)}f(e)$$

$$f(v_1,v_2)=\sum_{e\in(v_1,v_2)}f(e)$$

3. 流的概念

- 通过网络*N*的边（*i*, *j*）的实际流量叫作这条边的流，记为f_{ij}，各边的流的集合叫网络*N*的流。
- 从源发出的实际的流量（或宿收到的总流量）*F*叫网络的总流量。

4. 可行流的定义

- 设*f*是定义在边集*E*（*N*）上的一个整数值函数，若满足

  (1) 对所有的 $e \in E(N), then 0 \le f(e) \le$ C(e)

  (2) 对所有的中间顶点*i*有 f（i,V）=f（V,i）

  则称*f*是网络*N*的一个可行流（flow），*f*（*x*,*V*）称为可行流*f*的值，记作*f*（*x*,*y*）。

  f（*i*,*V*）叫流出顶点*i*的流;

  f（*V*, *i*）叫流入顶点*i*的流。
- 每一个网至少有一个流——零流。

- 在图4.25中所示的网络*N*中,每条边旁的第一个数是边的容量, 第二个数是边的流。例如*C*（*x*，1）=8，*f*（*x*，1）=4，*C*（1，2）=5，*f*（1，2）=1等等。

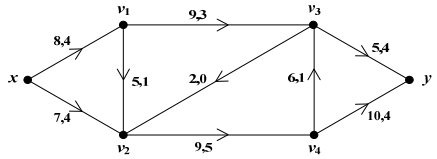


图4.25 流

5. 最大流定义

- 设*f*是网络*N*的一个流，如果不存在*N*的流*f'*，使*f'*(*x*,*y*)> *f*(*x*,*y*)，则称*f*为最大流，最大流的值记作f_{max}。
- 网络流量的讨论主要是要找出它的一个最大流，为此，我们先来讨论割的概念。

6. 割与割集

- 割的定义：设*N*=(*V*,*E*)是只有一个源*x*和一个宿*y*的网络，*V*1是*V*(*N*)的一个子集，*x*∈*V*1，*y*∈ $\overline{V1}$，（*V*1， $\overline{V1}$ ）表示起点在*V*1，终点在$\overline{V1}$的边的集合,把这些边的集合称为*N*的一个割，记作*K*。割中边的容量之和叫做割*K*的容量，用 *C* (*V*1, $\overline{V1}$) 或用*C*(*K*)表示割K的容量，于是表示。

$$C(K) = \sum_{e \in K} C(e)$$

- 由割的定义可知，网络*N* 的一个割是分离源和宿的弧的集合。
- 割的方向取从源到宿的方向。

- 割与割集的概念有区别，割*K*是按有向图来定义的，而割集则是按无向图来定义的。
  - 在图4.25中割（*V*1, $\overline{V1}$ ）={(*v*1,*v*3),(*v*2,*v*4)};
  - 而由节点子集*V*1和确定的割集是{(*v*1,*v*3),(*v*3,*v*2),(*v*2,*v*4)}，把N的割（*V*1, $\overline{V1}$ ）的全部边删去，自x到y将不存在任何有向链。
  - 我们取割的方向为从v_x到v_y的方向。
- 最小割：设 *N*为一个网，*K*是*N*的一个割，若不存在*N*的割*K'*使*C*(*K'*)<*C*(*K*)，则称*K*是*N*的最小割，其容量记为$C_{min}(K)$。
- 对于任何网络*N*，有$f_{max} \le C_{min}(K)$。

7．最大流最小割定理

- 在任何网络中，最大流的值等于最小割的容量，即$f_{max}=C_{min}(K)$。

8．前向边和反向边

- 在图的割集中，与割方向一致的边叫做前向边，与割方向相反的边叫反向边。

9．饱和边、非饱和边、零流边和非零流边

- 若$f(i,j)=C(i,j)$，则称边(vi,vj)为饱和边，若$f(i,j)<C(i,j)$，则称此边为非饱和边。若$f(i,j)=0$则称边eij为零流边，否则，为非零流边。

10．路、可增广路与不可增广路

- 路：设N为一个网络，N中相异节点$v1$，$v2,\ldots,vn$，对任意的$i(i=1,2,\ldots,n)$，（$vi,vi+1$）或（$vi+1,vi$）是N的一条边，且二者不能同时出现。这些节点的序列形成一条从x到y的道路，称为N中从x到y的一条路。在网络的路中可包含前向边，也可包含反向边。

- 可增广路与不可增广路：若从x到y的一条路中，所有前向边都未饱和，所有反向边都是非零流量的，则这条路称可增广路（可增流路）。若从x到y的一条路中，有一条前向边为饱和的或有一条反向边为零流量的，则这条路称为不可增广路。

4.4.1 流量分配的相关概念

4.4.2 网络最大流算法——标号法

4.4.3 最佳流问题

1．标号法基本思想

- 基本思想：从某初始可行流出发，在网络中寻找可增广路，若找到一条可增广路，则在满足可行流的条件下，沿该可增广路增大网络的流量，直到网络中不再存在可增广路。若网络中不存在可增广路，则网络中的可行流就是所求的最大流。

- 使用标号法求解网络最大流的过程如下：

（1）从任一个初始可行流出发，如零流。

（2）标号寻找一条从x到y点的可增广路。

（3）求解增广量：对于可增广路，总可能使其所有前向边都增加一个正整数ε，所有的反向边都减ε，而同时保持全部边的流量为正值且不超过边的容量，也不影响其他路上的边的流量，但却使网络的流F增加了ε。当x到y的全部路都为不可增广路时，F就不能再增加了，即F达到最大值。增流量ε用如下方法确定：

若可增广路上的边（i,j）的可增流量为

$$\varepsilon_{ij}=\begin{cases}c_{ij}-f_{ij} & (i,j)\text{为前向边}\\ f_{ij} & (i,j)\text{为反向边}\end{cases}$$

则此可增广路的增流量ε为$\varepsilon=\min\{\varepsilon_{ij}\}$。

可增广路径上的所有边中可增流量的最小值

（4）增广过程：前向边增加ε流量，反向边减ε。

（5）如增广后仍是可行流，则转到第（2）步；否则，已得到最大流。

- 上面讨论的主要是针对网络中某一条可增广路，来求解最大流。对于一个网络而言，应用最大流最小割定理来求解。

2. 标号算法步骤

➢ 标号法分为两个过程：其一是标记过程，用来寻找可增广路，同时可确定集合$V1$，此过程只需对每个节点检查一次，就能找到一条可增广路；其二是增广路的流的增加。

➢ 标记过程中，每一个节点给三种不同的记号。

- 第一个记号是下标i，即要检查的节点$i \in V1$的下标；
- 第二个记号用“+”或“-”来标记，若$C(i,j)-f(i,j)>0$则记为“+”号，若$f(j,i)>0$则记为“-”号；
- 第三个记号则用来说明有关弧上所能增大的流值。

➢ 标号算法如下：

初始x标记：(x, +, ε(i))

- 第一步：标记过程

（1）源x标记为$(x,+,\varepsilon(j))$，其中$\varepsilon(j)=\min\{\varepsilon(i),C(i,j)-f(i,j)\}$，之后称$j$已标记，未检查。

（2）任选一个已标记未检查的节点i，若节点j与i关联且尚未标记，则当

①$(i,j)\in E, C(i,j)> f(i,j)$时，将$j$标上$(i,+,\varepsilon(j))$，其中$\varepsilon(j)=\min\{\varepsilon(i), C(i,j)- f(i,j)\}$,之后称$j$已标记，未检查。

②$(j,i)\in E, f(j,i)>0$时，将j标上$(i,-,\varepsilon(j))$，其中$\varepsilon(j)=\min\{\varepsilon(i), f(j,i)\}$，之后称$j$已标记，未检查。

③ 与节点i关联的节点都被标记后，将i的第二个记号“+”或“-”用一个小圆圈圈起来，称i已被标记且被检查。

（3）重复（2）直到宿y被标记，或者直至不再有节点可以被标记。

- 第二步：增广过程

q是变量

（1）令$z=y$；目的（宿）节点

（2）如果z的标记为$(q,+,\varepsilon)$，把$f(q,z)$增加$\varepsilon(y)$；如果z的标记为$(q,-,\varepsilon)$，把$f(z,q)$减小$\varepsilon(y)$；

（3）如果$q=x$，把全部标记去掉，回到标记过程。否则，令$z=q$，回到（2）。

- 上一跳的源节点q作为本跳的目的节点进行增广；
- 逐级考虑前一跳的增广

例4.7 求图4.26所示网络的最大流。

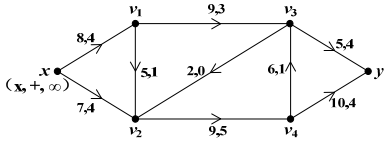
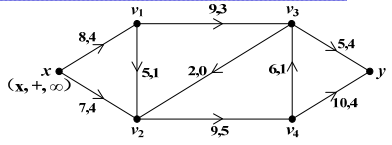


图4.26



解：第一步：标记过程

（1）源x标记成$(x,+,\infty)$。

（2）考察与x关联的节点$v1$和$v2$（为简单起见，下面我们用节点的下标表示该节点）：对节点1，$(x,1)\in E$且$C(x,1)=8$，$f(x,1)=4$，所以$\varepsilon(1)=\min\{\infty,8-4\}=4$。于是节点1标记成$(x,+,4)$；对于节点2，$\varepsilon(2)=\min\{\infty,7-4\}=3$。所以节点2标记成$(x,+,3)$。

v1 v2 均被标记

与x关联的节点均被标记，故x标记中的记号“+”用圆圈圈起来，即x被标记且被检查，如图4.27所示。

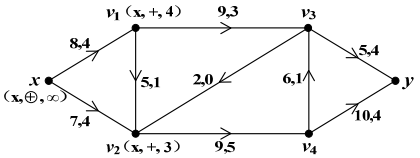


图4.27

继续上面的过程，直到宿y被标记，节点3标记为(1,+,4)，节点1被标记且被检查。节点4标记为(2,+,3)，节点2被标记且被检查。宿y被标记为（4,+,3），节点3和4被标记，被检查，如图4.28所示。

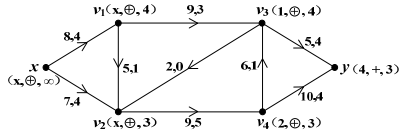


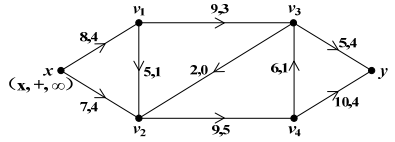
图4.28

第二步：增广过程

由标记过程找到一条可增广路：x，v2，v4，y

(1) 令z=y。

(2) y的标记为（4，+，3），所以把边（4，y）上的流值增加ε（y）=3，依次把边（2,4），（x,2）上的流值增加3。



(3) 去掉全部标记，得一网络如下图4.29，再回到标记过程。

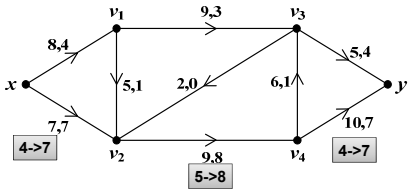
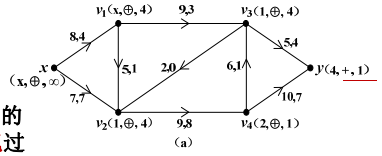


图4.29

增流ε（y）=3

对图4.29所示的网络，由标记过程和增广过程得图4.30（a）和（b）。



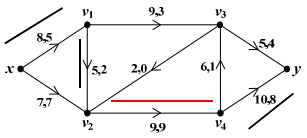
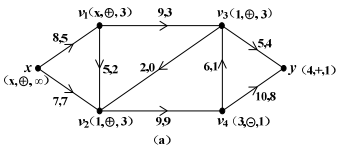


图4.30

(4) 去掉全部标记，得一网络，再回到标记过程。

对图4-30(b)所示的网络，由标记和增广过程可得图4.31(a)和(b)。



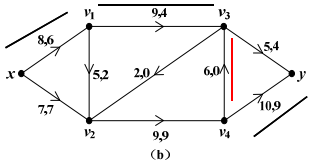
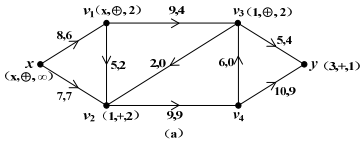


图4.31

对图4.31(b)所示的网络，由标记和增广过程可得图4.32(a)和(b)。



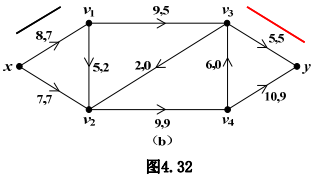


图4.32

最后得到如图4.33所示的网络。

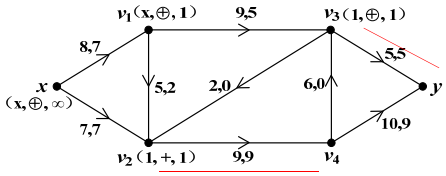


图4.33

最大流最小割{（v2,v4），（v3,y）}

- 从以上整个过程来看，当一个节点被标记，被检查后，在之后的过程中就完全可以不再考虑，所以这种标法是有效的。
- 某一节点*n*得到标记，则表示自*x*（*源节点*）到*n*之间的路为一条可增广路的前面一段。自*x*到*n*可能存在许多这种路，但只要找到一条就足够。
- 如果*y*被标记，则说明自*x*至*y*存在一条可增广的路，流值的改变则可按*ε(y)*来确定。

3．最大流最小割定理的推广

➢ 多源多宿网络

- 若一个网络有多个源和多个宿，*vx*1, *vx*2,…，*vxm*；*vy*1, *vy*2, …,*vyn* .对这种网设置两个节点*vx*和 *vy*；*vx*和 *vy*作为新源和宿，连接新的边（*vx*, *vx*1），（*vx*, *vx*2），…，（*vx*, *vxm*）和（*vy*1, *vy*），（*vy*2, *vy*），…，（*vyn*, *vy*），指定其容量均为∞，将原图化为单源单宿网。
- 求得最大流后，通过*vxi*的信息均由*vx*发，经由*vyi*的信息均由*vy*收。如图4.34所示。

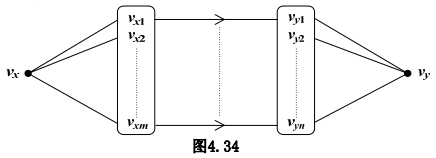


图4.34

➢ 节点容量问题

当节点的转接能力有限时，可把该节点分成两个节点，一个与所有射入边相连，另一个与所有射出边相连，再在这两节点间加一条有向边，边的容量可标为节点的容量。如图4.35所示。

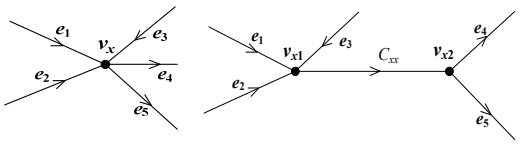


图4.35

➢无向图情况

一般的无向图是指双通路，所以边容量实际上是正向容量，也是反向容量。

这时可把一条无向边换成两条有向边，即一条是正向的有向边，另一条是反向的有向边，然后可按有向图计算。

4.4.1 流量分配的相关概念

4.4.2 网络最大流算法——标号法

4.4.3 最佳流问题

最后得到如图4.33所示的网络。

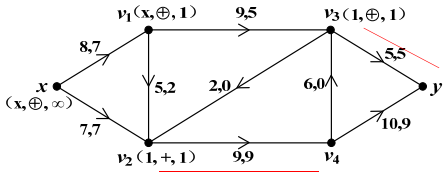


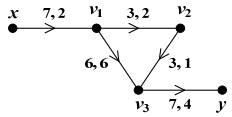
图4.33

最大流最小割{（v2,v4），（v3,y）}

➢最佳流问题，就是给定网结构$G(V,E)$，边容量C_{ij}，边费用a_{ij}以及总流量F_{xy}，要求费用：Ø $=\sum_{ij} a_{ij} \cdot f_{ij}$ 最小。

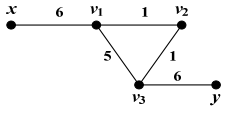
➢负价环算法 （N算法）

图4.36（a）中x到y间有两条径，即x，$v1$，$v2$，$v3$，y和x，$v1$，$v3$，y。每条边上数字代表各边的容量c_{ij}和费用a_{ij}。图4.36（b）是一组可行流，总流量F_{xy}=6，总费用是69。图4.36(c)给出了各边上流量改变的可能性以及改变单位流量所需的费用。此图称为对于(b)中可行流而得的补图。以e_{12}为例，它是非饱和边，流量尚可增加$C12-f12=2$，所需单位费用为+2。另一方面，流量也可减少$f12=1$，不致破坏非负性，所需单位费用就是-2，因减流就减小费用。这两种可能改变的流量用补图中两条附有两个数字的有向边来表示，前面的数字代表可增流值，后面的数字代表单位流量所需的费用。



(a) C_{ij}，a_{ij}容量和费用

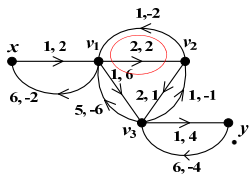
图4.36（a）中x到y间有两条径，即x，$v1$，$v2$，$v3$，y和x，$v1$，$v3$，y。每条边上数字代表各边的容量c_{ij}和费用a_{ij}。



(b) f_{ij}可行流

图4.36（b）是一组可行流，总流量F_{xy}=6，总费用是69。

2*6+2*1+6*5+1*1+4*6=69



(c) 补图和负价环

- 图4.36(c)给出了各边上流量改变的可能性以及改变单位流量所需的费用。此图称为对于(b)中可行流而得的补图。以e_{12}为例，它是非饱和边，流量尚可增加$C12-f12=2$，所需单位费用为+2。
- 另一方面，流量也可减少$f12=1$，不致破坏非负性，所需单位费用就是-2，因减流就减小费用。
- 这两种可能改变的流量用补图中两条附有两个数字的有向边来表示，前面的数字代表可增流值，后面的数字代表单位流量所需的费用。

- 补图上若存在一个有向环，环上各边的a_{ij}之和是负数，则称此环为负价环。
- 沿负价环方向增流，并不破坏环上诸节点的流量连续性，也不破坏各边的非负性和有限性，结果得到一个F_{xy}不变的可行流，其总费用将有所降低。
- 图4.36(c)中的（$v1$，$v2$，$v3$，$v1$）环是一个负价环，取环中的容量最小值作为可增流的值，此时为2；这负价环的单位流量费用是2+1-6=-3。因为可增流值为2，所以可节省费用为-3×2= -6。把F_{xy}的费用从69降到63。新的可行流如图4.36(d)所示。

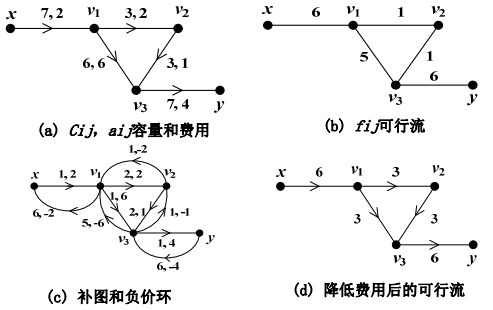


图4.36 负价环法求最佳流过程

➢ 负价环法的步骤可归纳如下：

（1）在图上找任一满足总流量*Fxy*的可行流。

（2）做补图。对所有边*eij*，若*Cij*> *fij*，做边*e'ij*，其容量为*C'ij*=*Cij*- *fij*，费用为*aij*，若*fij* >0,再作*e'ji*，其容量为*C'ij*= *fij*，费用为- *aij*。

（3）在补图上找负价环。若无负价环，算法终止。若有，沿着这个负价环*C*方向使各边增流，增流量为 min *C'ij* 。

（4）修改原图的边流量，得新的可行流，返回第二步。

4.1 图论基础
4.2 树
4.3 路径选择算法
4.4 网络流量分配及其算法
4.5 通信网的可靠性

4.5.1 可靠性定义及相关概念

4.5.2 多部件系统可靠性计算

4.5.3 工程中采用的可靠性指标

1．通信网可靠性定义

➢ 通信网可靠性定义为网络在给定条件下和规定时间内，完成规定功能，并能把其业务质量参数保持在规定值以内的能力。当网络丧失了这种能力时就是出了故障，由于网络出现故障的随机性，所以研究网络的可靠性要使用概率论和数理统计的知识。

- 通信网的可靠性可以用可靠度、不可靠度、平均故障间隔时间以及平均故障修复时间来描述。

2．可靠度*R*(*t*)

➢ 可靠度是系统在给定条件下和规定时间内完成所要求功能的概率，用*R*(*t*)表示，若用一非负随机变量*x*表示系统的故障间隔时间，则*R*(*t*)定义为

$$R(t) = P(x > t)$$

间隔时间大于t时概率

即系统在时间间隔[0，t]内不发生故障（运行）的概率。

➢ 不可靠度：从可靠度*R*(*t*)与故障间隔时间分布函数*F*(*t*)的关系可以看出，*F*(*t*)即为系统的不可靠度。

- 为了得到系统的可靠度，必须首先知道该系统的故障间隔时间分布函数。这里只研究指数分布函数，它可以表示为

$$F(t)=P(x\le t)=1-e^{-\lambda t} \qquad t\ge 0 \qquad \lambda>0$$

- 式中 λ为系统在单位时间内发生故障的概率，称故障率。为了研究问题的简便，假设与时间无关，则根据可靠度定义

$$R(t)=P(x>t)=1-F(t)=e^{-\lambda t}$$

3．系统的平均故障间隔时间（MTBF）

➢ 平均故障间隔时间（MTBF，Mean Time Between Failures）是两个相邻故障间的时间的平均值。

- 系统故障间隔时间x的概率密度函数为

$$f(t)=\frac{dF(t)}{dt}=\lambda R(t)=\lambda e^{-\lambda t}$$

- 系统的平均故障间隔时间为

$$\mathrm{MTBF}=\int_0^\infty tf(t)dt=\int_0^\infty R(t)dt$$

- 为λ常量时，$\mathrm{MTBF}=1/\lambda$ $R(t)=P(x>t)=1-F(t)=e^{-\lambda t}$

- MTBF是表征网络可靠性的重要参量。定性地说，MTBF越大，系统越可靠。若 λ为常量，MTBF与 λ一样，都可以用来充分描述系统的可靠性。
- 系统在平均寿命到达时尚能运行的概率为e-1=0.368。这说明有些系统可能在MTBF达到前出故障，即故障间隔时间短于MTBF；另一些相同的系统，故障间隔时间可能大于MTBF。

$e^{-\lambda t}$ where t=1/λ

4.5.1 可靠性定义及相关概念

4.5.2 多部件系统可靠性计算

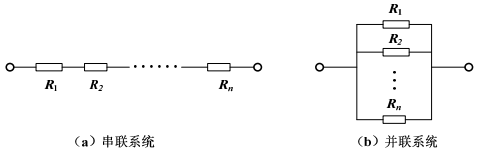
4.5.3 工程中采用的可靠性指标

1．串联系统

➢ 当若干个具有指数故障间隔时间分布函数的部件串联时，可以很方便地求出该串联系统的可靠度。

- 图4.37（a）是n个部件串联的系统，各部件相互独立，当n个部件有一个失效时，该串联系统就发生故障。设各个部件的寿命为x_i，可靠度为$R_i(t)$，i=1，2，…，n。该串联系统的故障间隔时间x是n个部件的故障间隔时间x_i中最小值，即

$$x=\min(x_1,x_2,\cdots,x_n)$$



（a）串联系统　　（b）并联系统

图4.37 串联系统与并联系统

- 该串联系统的可靠度根据定义，有：

$$R(t)=P\{x>t\}=P\{\min(x_1,x_2,\cdots,x_n)>t\}=P\{x_1>t,x_2>t,\cdots,x_n>t\}=\prod_{i=1}^{n}R_i(t)$$

当 $R_i(t)=e^{-\lambda_i t}$ 时，

$$R(t)=\prod_{i=1}^{n}e^{-\lambda_i t}=\exp(-t\sum_{i=1}^{n}\lambda_i)$$

4.5.2 多部件系统可靠性计算

- 串联系统的平均故障间隔时间

$$\text{MTBF}=\int_0^\infty R(t)dt=\int_0^\infty \exp(-t\sum_{i=1}^n\lambda_i)dt=\frac{1}{\sum_{i=1}^n\lambda_i}$$

当$\lambda_1=\lambda_2=\cdots=\lambda_n=\lambda$时，

$$R(t)=\exp(-n\lambda t)$$

$$\text{MTBF}=\frac{1}{n\lambda}$$

4.5.2 多部件系统可靠性计算

2．并联系统

- 图4.37（b）是*n*个部件并联的系统，各部件相互独立，当*n*个部件全部失效时该并联系统才发生故障。各部件的故障间隔时间和可靠度分别为x_i和$R_i(t)$，i=1，2，…，n。
- 并联系统的故障间隔时间

$$x=\max(x_1,x_2,\cdots,x_n)$$

4.5.2 多部件系统可靠性计算

- 可靠度

$$R(t)=P\{\max(x_1,x_2,\cdots,x_n)>t\}=1-P\{\max(x_1,x_2,\cdots,x_n)\le t\}$$

$$=1-P\{x_1\le t,x_2\le t,\cdots,x_n\le t\}=1-\prod_{i=1}^n[1-R_i(t)]$$

当 $R_i(t)=e^{-\lambda_i t}$时

$$R(t)=1-\prod_{i=1}^n(1-e^{-\lambda_i t})$$

- 系统的平均故障间隔时间

$$\text{MTBF}=\int_0^\infty R(t)dt=\int_0^\infty[1-\prod_{i=1}^n(1-e^{-\lambda_i t})]dt$$

4.5.2 多部件系统可靠性计算

当 $\lambda_1=\lambda_2=\cdots=\lambda_n=\lambda$ 时

$$R(t)=1-(1-e^{-\lambda t})^n$$

$$\text{MTBF}=\int_0^\infty R(t)dt=\int_0^\infty[1-(1-e^{-\lambda t})^n]dt$$

令$y=1-e^{-\lambda t}$，$dy=\lambda e^{-\lambda t}dt$，$dt=\frac{1}{\lambda e^{-\lambda t}}dy=\frac{1}{\lambda}\cdot\frac{1}{1-y}dy$。

当t=0时，y=0；当 $t\to\infty$ 时，y=1，故有

$$\text{MTBF}=\frac{1}{\lambda}\int_0^1\frac{1-y^n}{1-y}dy=\frac{1}{\lambda}\int_0^1[1+y+y^2+\cdots+y^{n-1}]dy=\frac{1}{\lambda}[1+\frac{1}{2}+\frac{1}{3}+\cdots+\frac{1}{n}]=\sum_{i=1}^n\frac{1}{i\lambda}$$

4.5.2 多部件系统可靠性计算

3．复杂系统的可靠度

- 一般的系统并不只是由多部件串联或并联组成的，而是串并混合或更复杂的系统，这些系统的可靠度都可通过等效系统的方法用串联、并联系统可靠度的计算方法得到。
- 下面通过图4.38(a)所示系统的可靠度求解说明复杂系统可靠度的求解方法。

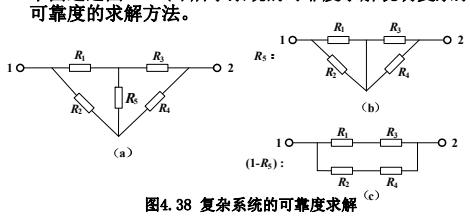


图4.38 复杂系统的可靠度求解

4.5.2 多部件系统可靠性计算

- 要求解图4.38(a)中1与2两节点之间的可靠度，可以分别考虑$R5$的运行状态和失效状态，若该部件运行，系统可等效为图(b)，即此部件相当于短路，其概率为$R5$；若该部件失效，系统可等效为图(c)，即此部件相当于开路，其概率为1-$R5$。
- 已知串联系统的可靠度

$$R(t)=R_1(t)\times R_2(t)\times\cdots\times R_n(t)=\prod_{i=1}^n R_i(t)$$

- 并联系统的可靠度

$$R(t)=1-\prod_{i=1}^n[1-R_i(t)]=1-\prod_{i=1}^n F_i(t)$$

4.5.2 多部件系统可靠性计算

- 故可得该复杂系统的可靠度

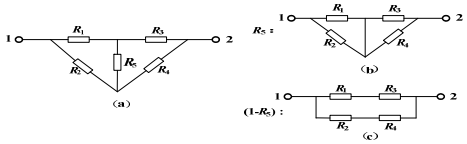
$$R = R_5(1-F_1F_2)(1-F_3F_4)+(1-R_5)[1-(1-R_1R_3)(1-R_2R_4)]$$

$$= R_5[1-(1-R_1)(1-R_2)][1-(1-R_3)(1-R_4)]+(1-R_5)[1-(1-R_1R_3)(1-R_2R_4)]$$

$$= R_5(R_1+R_2-R_1R_2)(R_3+R_4-R_3R_4)(1-R_5)(R_1R_3+R_2R_4-R_1R_2R_3R_4)$$

$$= R_1R_3+R_2R_4-R_1R_2R_3R_4+R_5(R_1R_4+R_2R_3$$

$$-R_1R_3R_4-R_2R_3R_4-R_1R_2R_3-R_1R_2R_4+2R_1R_2R_3R_4)$$



4.5 通信网的可靠性

4.5.1 可靠性定义及相关概念

4.5.2 多部件系统可靠性计算

4.5.3 工程中采用的可靠性指标

4.5.3 工程中采用的可靠性指标

➢ 上面讨论的系统可靠性都假定系统为不可修复系统，而实际通信网中的绝大多数部件属于可修复系统。这时，可靠性概念中包含着维修性，可靠性可用有效度*A*、MTBF和平均故障修复时间（MTTR：Mean Time to Restoration）来表述。

4.5.3 工程中采用的可靠性指标

➢ 用*x*表示系统的故障间隔时间，用*y*表示部件出现故障后的修复时间，设*x*，*y*均服从指数分布，即

$$P\{x \le t\} = 1-e^{-\lambda t} \qquad t \ge 0 \quad \lambda > 0$$

$$P\{y \le t\} = 1-e^{-\mu t} \qquad t \ge 0 \quad \mu > 0$$

式中 λ，μ，分别为系统的故障率和修复率。

4.5.3 工程中采用的可靠性指标

- 若 $R(t+\Delta t)$ 是在 $t+\Delta t$ 时系统正常运行的概率，有两种情况可到达运行状态，即*t*时在运行，*t*到 $t+\Delta t$ 之间不出故障，以及*t*时已失效，*t*到 $t+\Delta t$ 之间能修复。这样可得：

$$R(t+\Delta t) = R(t)(1-\lambda\Delta t)+[1-R(t)]\mu\Delta t$$

令 $\Delta t \to 0$，整理后得

$$R'(t) = \mu-(\lambda+\mu)R(t)$$

这是非齐次常微分方程，可以求出它的通解为

$$R(t) = \frac{\mu}{\lambda+\mu}+Ce^{-(\lambda+\mu)t} \qquad \cdots\cdots(4.38)$$

4.5.3 工程中采用的可靠性指标

当 λ 和 μ 为常量时，若 t=0时刻系统处于正常运行状态，即 $R(0)$=1，则可得常数

$$C = \frac{\lambda}{\lambda+\mu}$$

方程(4.38)的通解

$$R(t) = \frac{\mu}{\lambda+\mu}+\frac{\lambda}{\lambda+\mu}e^{-(\lambda+\mu)t} \qquad \cdots\cdots(4.40)$$

若 t=0时刻系统处于故障状态，即 $R(0)$=0，可得常数

$$C = -\frac{\mu}{\lambda+\mu}$$

4.5.3 工程中采用的可靠性指标

方程（4.38）的通解

$$R(t)=\frac{\mu}{\lambda+\mu}(1-e^{-(\lambda+\mu)t}) \quad \cdots\cdots(4.42)$$

当$t\to\infty$时，式（4.40）和（4.42）均成为

$$R=\lim_{t\to\infty}R(t)=\frac{\mu}{\lambda+\mu}$$

这就是系统的稳态可靠度，在工程中称其为系统有效度，用A表示，即

$$A=\frac{\mu}{\lambda+\mu} \quad \cdots\cdots(4.44)$$

4.5.3 工程中采用的可靠性指标

由前知系统的平均故障间隔时间为

$$\mathrm{MTBF}=1/\lambda$$

同理可知平均故障修复时间为

$$\mathrm{MTTR}=1/\mu$$

则式(4.44)为

$$A=\frac{\mathrm{MTBF}}{\mathrm{MTBF}+\mathrm{MTTR}}$$

4.5.3 工程中采用的可靠性指标

同理，系统不可利用度为

$$U=1-A=\frac{\lambda}{\lambda+\mu}=\frac{\mathrm{MTTR}}{\mathrm{MTBF}+\mathrm{MTTR}}$$

***U*=1-*A*为在规定的时间和条件内系统丧失规定功能的概率，称为系统不可利用度。**

- **可见系统的有效度为可靠性与维修性两者的综合。**
- **为了提高通信系统的可靠性，在工程中可采用主备用设备转换等冗余技术。**

谢谢

本讲回顾及补充

- **交换**：把数据（包）从交换节点的一个端口（port）转发到另一个端口。
  - 电路交换：**直接对电路进行交换，面向连接，主要用于电话网**
  - 分组交换：**将分组数据包从一个链接交换转发到另一个链接，无连接，主要用于数据网**
- **路由**：计算从数据源节点到目的节点的**路径（path）**
  - **链路状态路由(LSR：Link State Routing)算法**

167