第 5 章　图

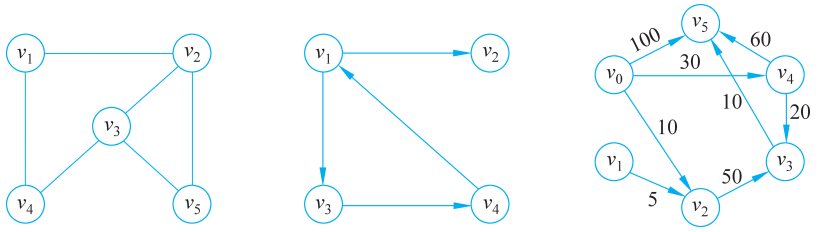
图是一种比线性结构和树更为复杂的数据结构。在线性结构中，数据元素之间是一对一的关系，即除了第一个和最后一个元素外，每个元素都有唯一的直接前驱和直接后继；在树结构中，数据元素之间是一对多的关系，即除了根结点之外，每个结点都有唯一的双亲结点，但可以有多个孩子；而在图结构中，数据元素之间可以是多对多的关系，也就是说，任意两个数据元素之间都可能相关。

图的应用领域非常广泛，如物理、化学、信息工程、计算机科学，以及数学等其他分支中[①]。

5.1　图的基本概念

图(Graph)由两个集合 V 和 E 组成，记为 $G=(V,E)$，其中，V 是顶点(Vertex)的有穷非空集合，E 是连接 V 中两个不同顶点的边(Edge)的有穷集合。对于图 G，若边集 $E(G)$ 为无向边的集合，则称该图为无向图(Undirected Graph)；若边集 $E(G)$ 为有向边的集合，则称该图为有向图(Directed Graph)。

在无向图中，用一对圆括号括起来的顶点对 (v_i,v_j) 是无序的，称为与顶点 v_i 和顶点 v_j 相关联的一条边。这条边没有特定的方向，所以 (v_i,v_j) 与 (v_j,v_i) 是同一条边。如图 5.1(a)所示，无向图 G_1 的顶点集合为 $V(G_1)=\{v_1,v_2,v_3,v_4,v_5\}$，边集合为 $E(G_1)=\{(v_1,v_2),(v_1,v_4),(v_2,v_3),(v_2,v_5),(v_3,v_4),(v_3,v_5)\}$。



(a) 无向图G_1　　(b) 有向图G_2　　(c) 有向图G_3

图 5.1　图的示例

在有向图中，用一对尖括号括起来的顶点对 $\langle v_i,v_j\rangle$ 是有序的，称为从顶点 v_i 到顶点 v_j 的一条有向边，也可以称作一条弧(Arc)，其中，v_i 为弧尾或起点，v_j 为弧头或终点。如图 5.1(b)所示，有向图 G_2 的顶点集合为 $V(G_2)=\{v_1,v_2,v_3,v_4\}$，边集合为 $E(G_2)=\{\langle v_1,v_2\rangle,\langle v_1,v_3\rangle,\langle v_3,v_4\rangle,\langle v_4,v_1\rangle\}$。

① 在离散数学中，图论是专门研究图的性质的数学分支。

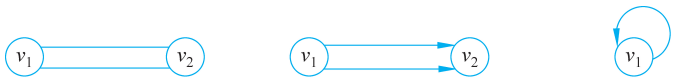
在实际应用中，边上可以标注具有某种含义的数值，表示从一个顶点到另一个顶点的距离、时间、次数、耗费等，称为权(Weight)。这种带权的图通常称为网(Network)。如图 5.1(c)所示，有向网 G_3 的顶点集合为 $V(G_3)=\{v_0,v_1,v_2,v_3,v_4,v_5\}$，边集合为 $E(G_3)=\{\langle v_0,v_2\rangle 10,\langle v_0,v_4\rangle 30,\langle v_0,v_5\rangle 100,\langle v_1,v_2\rangle 5,\langle v_2,v_3\rangle 50,\langle v_3,v_5\rangle 10,\langle v_4,v_3\rangle 20,\langle v_4,v_5\rangle 60\}$。

【知识扩展】　　平行边与自环

本书中讨论的图都是简单图，也就是既不含平行边也不含自环(Self Loop)的图。

在无向图中，若关联一对顶点的无向边多于一条，则称这些边为平行边，如图 5.2(a)所示。在有向图中，若关联一对顶点的有向边多于一条，并且这些边的初始点和终端点相同(方向一致)，则称这些边为平行边，如图 5.2(b)所示。含平行边的图称为多重图(Multigraph)。平行边的条数称为重数。

自环是指一条边关联的两个顶点为同一个顶点，也就是说，自己到自己有一条边，如图 5.2(c)所示。

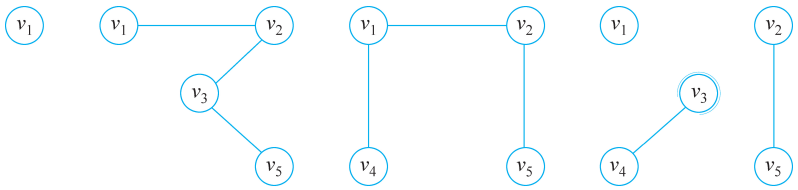


(a) 无向图的平行边　　(b) 有向图的平行边　　(c) 自环

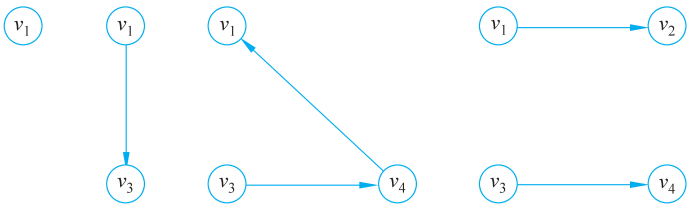
图 5.2　平行边与自环示例

下面介绍关于图的一些基本术语(用 n 表示顶点的数目，用 e 表示边的数目)。

(1) 子图(Subgraph)：假设有两个图 $G=(V,E)$ 和 $G'=(V',E')$，如果 $V'\subseteq V$ 且 $E'\subseteq E$，则称 G' 为 G 的子图。例如，图 5.3 给出了图 5.1 中 G_1 和 G_2 的一些子图示例。



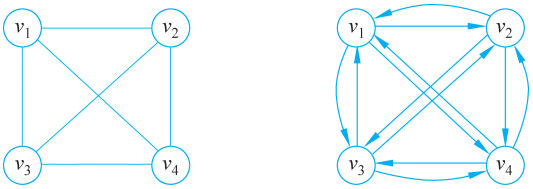
(a) G_1的子图



(b) G_2的子图

图 5.3　子图的示例

(2) 完全图(Completed Graph)：对于无向图，若具有 $n(n-1)/2$ 条边，则称为无向完全图。如图 5.4(a)所示，G_4 有 4 个顶点，6 条边。对于有向图，若具有 $n(n-1)$ 条弧，则称为有向完全图。如图 5.4(b)所示，G_5 有 4 个顶点，12 条弧。



（a）无向完全图G_4　　（b）有向完全图G_5

图 5.4　完全图的示例

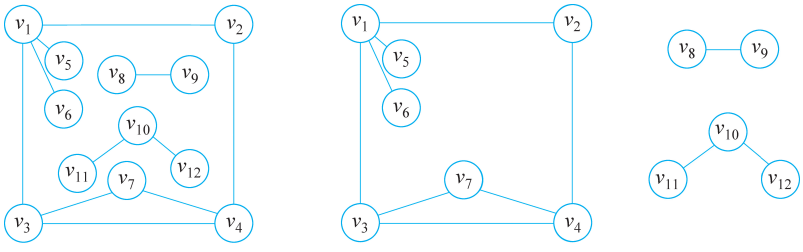
（3）稀疏图（Sparse Graph）和稠密图（Dense Graph）：有很少条边或弧（如 $e<n\log_2 n$）的图称为稀疏图，反之称为稠密图。

（4）邻接（Adjacent）：有边/弧相连的两个顶点之间的关系。例如$(v_1,v_2)\in E(G_1)$，则称顶点 v_1 和 v_2 互为邻接点，即 v_1 和 v_2 相邻接。边(v_1,v_2)依附于顶点 v_1 和 v_2，或者说边(v_1,v_2)与顶点 v_1 和 v_2 相关联。

（5）度（Degree）：顶点 v 的度是和 v 相关联的边的数目，记为 TD(v)。例如，图 5.1(a)中 G_1 的顶点 v_3 的度是 3。对于有向图，顶点 v 的度分为入度（Indegree）和出度（Outdegree）。入度是以顶点 v 为头的弧的数目，记为 ID(v)；出度是以顶点 v 为尾的弧的数目，记为 OD(v)。顶点 v 的度为 TD(v)=ID(v)+OD(v)。例如，图 5.1(b)中 G_2 的顶点 v_1 的入度 ID(v_1)=1，出度 OD(v_1)=2，度 TD(v_1)=ID(v_1)+OD(v_1)=3。如果顶点 v_i 的度记为 TD(v_i)，则满足 $\sum_{i=1}^{n}\mathrm{TD}(v_i)=2e$，即度数之和等于边数的两倍。

（6）路径（Path）：在无向图 G 中，从顶点 v 到顶点 v'的路径是一个顶点序列$(v=v_{i,0},v_{i,1},\cdots,v_{i,m}=v')$，其中，$(v_{i,j-1},v_{i,j})\in E$，$1\leqslant j\leqslant m$。如果 G 是有向图，则路径也是有向的，顶点序列应满足$\langle v_{i,j-1},v_{i,j}\rangle\in E$，$1\leqslant j\leqslant m$。路径长度是一条路径上经过的边或弧的数目。序列中顶点不重复出现的路径称为简单路径。第一个顶点和最后一个顶点相同的路径称为回路或环（Cycle）。除了第一个顶点和最后一个顶点之外，其余顶点不重复出现的回路，称为简单回路或简单环。

（7）连通（Connected）：在图 G 中，如果从顶点 v 到顶点 v'有路径，则称 v 和 v'是连通的。如果对于无向图中任意两个顶点 $v_i,v_j\in V$，v_i 和 v_j 都是连通的，则称该无向图是连通图（Connected Graph）。图 5.1(a)中 G_1 就是一个连通图，而图 5.5(a)中的 G_6 则是非连通图，但 G_6 有三个连通分量，如图 5.5（b）所示。所谓连通分量（Connected Component），指的是无向图中的极大连通子图。



（a）无向图G_6　　（b）G_6的三个连通分量

图 5.5　无向图及其连通分量示例

在有向图中，如果对于每一对 $v_i, v_j \in V, v_i \neq v_j$，从 v_i 到 v_j 和从 v_j 到 v_i 都存在路径，则称该有向图是强连通图。有向图中的极大强连通子图称作有向图的强连通分量。例如，图 5.1(b)中的 G_2 不是强连通图，但它有两个强连通分量，如图 5.6 所示。

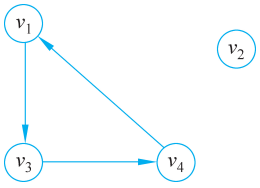


图 5.6　有向图 G_2 的两个强连通分量

(8) 生成树(Spanning Tree)：一个连通图的生成树是一个极小连通子图，它含有图中全部顶点，但只有足以构成一棵树的 $n-1$ 条边。如图 5.7 所示为 G_6 中最大的那个连通分量的一棵生成树。如果在一棵生成树上添加一条边，必定构成一个环，因为这条边使得它依附的那两个顶点之间有了第二条路径。

一棵有 n 个顶点的生成树有且仅有 $n-1$ 条边。如果一个图有 n 个顶点和小于 $n-1$ 条边，则是非连通图。如果它多于 $n-1$ 条边，则一定有环。但是，有 $n-1$ 条边的图不一定是生成树。

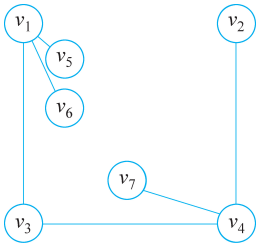
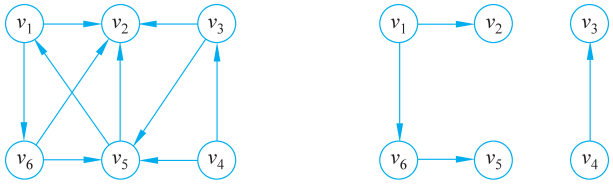


图 5.7　图 G_6 的最大的连通分量的一棵生成树

如果一个有向图恰有一个顶点的入度为 0，其余顶点的入度均为 1，则该有向图是一棵有向树。一个有向图的生成森林是由若干棵有向树组成的，含有图中全部顶点，但只有足以构成若干棵不相交的有向树的弧(如图 5.8 所示)。



(a) 有向图G_7　　(b) G_7的生成森林

图 5.8　有向图及其生成森林示例

5.2 图的存储结构

对图进行描述的时候，“顶点的位置”和“邻接点的位置”都只是相对的概念。因为从图的逻辑结构的定义来看，图上任一顶点都可以作为第一个顶点，任一顶点的邻接点之间也不存在次序关系。

为了用计算机存储和操作方便，需要人为地将图中顶点按某种顺序排列[①]起来。因此，所谓“顶点在图中的位置”指的是该顶点在这个人为的随意排列中的位置（或序号）。同样，也可以对某个顶点的所有邻接点进行排队，在这个排队中自然形成了第 1 个、第 2 个……直至第 k 个邻接点。

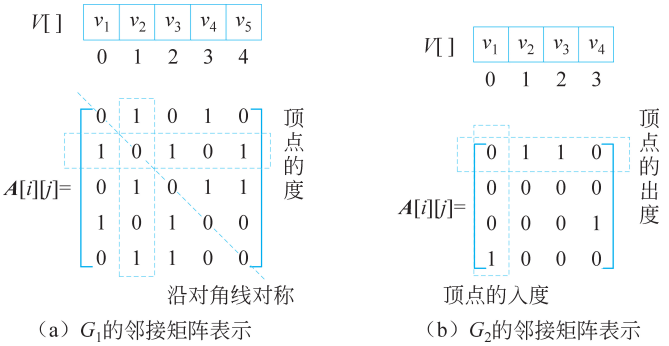
5.2.1 邻接矩阵

邻接矩阵（Adjacency Matrix）表示法是一种数组表示法（顺序表示法），使用两个数组分别存储数据元素（顶点）的信息和数据元素之间的关系（边或弧）的信息。

设 $G(V,E)$ 是具有 n 个顶点的图，先用一个一维数组存储这 n 个顶点的信息（按照某种顺序排列），然后用一个二维数组（即 G 的邻接矩阵）存储图中顶点之间的邻接关系，该矩阵是具有如下性质的 n 阶方阵：

$$\boldsymbol{A}[i][j]=\begin{cases}1, & (v_i,v_j)\text{ 或}\langle v_i,v_j\rangle \in E\\ 0, & \text{其他}\end{cases} \tag{5-1}$$

例如，如图 5.1 所示的 G_1 和 G_2 的邻接矩阵如图 5.9 所示。



（a）G_1的邻接矩阵表示　　（b）G_2的邻接矩阵表示

图 5.9　图的邻接矩阵示例

无向图的邻接矩阵是对称矩阵，第 i 行或第 i 列的元素之和就是第 i 个顶点的度。如图 5.9(a)所示，在 G_1 的邻接矩阵中，第 2 行或第 2 列的元素之和为 3，说明第 2 个顶点（即 v_2）的度为 3。

而有向图的邻接矩阵不一定是对称的，第 i 行的元素之和是第 i 个顶点的出度，第 i 列的元素之和是第 i 个顶点的入度。如图 5.9(b)所示，在 G_2 的邻接矩阵中，第 1 行的元

① 这个排列可以是任意顺序且和关系 VR 无关。

素之和为 2，说明第 1 个顶点（即 v_1）的出度为 2；第 1 列的元素之和为 1，说明第 1 个顶点（即 v_1）的入度为 1。

网（带权图）的邻接矩阵则是具有如下性质的 n 阶方阵：

$$\boldsymbol{A}[i][j]=\begin{cases} w_{ij}, & (v_i,v_j)\text{ 或}\langle v_i,v_j\rangle \in E \\ 0, & i=j \\ \infty, & \text{其他} \end{cases} \tag{5-2}$$

其中，w_{ij} 表示边上的权值；∞ 表示计算机允许的、大于所有边上权值的数。例如，如图 5.1 所示的 G_3 的邻接矩阵如图 5.10 所示。

$V[\,]$

v_0	v_1	v_2	v_3	v_4	v_5
0	1	2	3	4	5

$$A[i][j]=\begin{bmatrix} 0 & \infty & 10 & \infty & 30 & 100 \\ \infty & 0 & 5 & \infty & \infty & \infty \\ \infty & \infty & 0 & 50 & \infty & \infty \\ \infty & \infty & \infty & 0 & \infty & 10 \\ \infty & \infty & \infty & 20 & 0 & 60 \\ \infty & \infty & \infty & \infty & \infty & 0 \end{bmatrix}$$

图 5.10　有向网 G_3 的邻接矩阵表示

邻接矩阵的优点：

（1）便于判断两个顶点之间是否有边。可以根据 $\boldsymbol{A}[i][j]$ 的值来判断。

（2）便于计算各个顶点的度。即第 i 行或第 i 列的有效元素的个数。

邻接矩阵的缺点：

（1）不便于增加和删除顶点。需要改变邻接矩阵的大小，效率较低。

（2）不便于统计边的数目。需要扫描整个邻接矩阵，时间复杂度为 $O(n^2)$。

（3）空间复杂度高。空间复杂度为 $O(n^2)$，对于稀疏图而言尤其浪费空间。

【知识扩展】　　边 集 数 组

除了邻接矩阵表示法，还有一种不常见的数组表示法——边集数组表示法。该方法使用两个一维数组来存储图的信息。其中，存储（顶点）信息的数组与邻接矩阵表示法是一样的；存储边（或弧）信息的数组则不同，每个数据元素由三个数据项组成，即一条边的起点下标（begin）、终点下标（end）和权值（weight）。例如，如图 5.1(c)所示的 G_3 的边集数组如图 5.11 所示。

可以看出，边集数组表示法不适合对顶点的相关操作，例如，计算某个顶点的度（需要扫描整个边集数组，效率低）；它更适合对边依次进行处理的操作，例如，依据权值对边进行排序（5.4 节介绍的克鲁斯卡尔算法）。

5.2.2　邻接表

邻接表（Adjacency List）是一种把顺序和链式结合起来的存储结构。在邻接表中，需

V[ ]

0	1	2	3	4	5
v_0	v_1	v_2	v_3	v_4	v_5

E[ ]

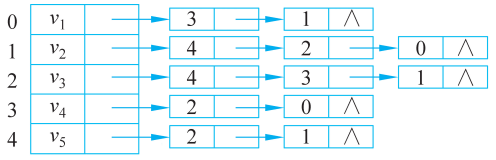
	begin	end	weight
0	0	2	10
1	0	4	30
2	0	5	100
3	1	2	5
4	2	3	50
5	3	5	10
6	4	3	20
7	4	5	60

图 5.11　有向网 G_3 的边集数组表示

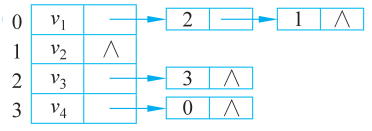
要申请一个大小为 n（与顶点数目一致）的一维数组，存放的数据元素由两个数据项组成，即顶点 v_i 的信息（如顶点的名称等）和一个指针。该指针指向为顶点 v_i 建立的一个无头结点的单链表。

单链表的每个结点包括邻接点域（adjvex）和链域（nextarc）两部分。其中，邻接点域指示与 v_i 邻接的顶点的位置（在数组中的下标），链域指向单链表的下一个结点。如有必要，还可以再设置一个数据域（info），用来存储与边（依附于 v_i 和该邻接点）相关的信息，如权值等。

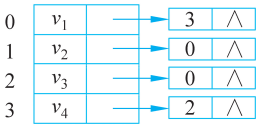
例如，如图 5.12(a)和图 5.12(b)所示分别为图 5.1 中 G_1 和 G_2 的邻接表。



（a）G_1的邻接表



（b）G_2的邻接表



（c）G_2的逆邻接表

图 5.12　邻接表和逆邻接表的示例

在无向图的邻接表中，顶点 v_i 的度正是其对应的单链表中的结点个数；而在有向图中，其对应的单链表中的结点个数只是顶点 v_i 的出度。若要计算有向图中某个顶点的入度，必须遍历整个邻接表中的所有单链表的结点，统计 v_i（在数组中的下标）出现的次数。有时，为了便于确定顶点的入度，可以建立一个有向图的逆邻接表，即对每个顶点 v_i 建立一个单链表，存放与 v_i 邻接且作为弧尾的顶点（v_i 作为弧头）。例如，如图 5.12(c)所示为有向图 G_2 的逆邻接表。

需要特别注意的是，**一个图的邻接矩阵表示是唯一的，但其邻接表表示并不唯一**。这是因为在邻接表表示中，单链表各结点的链接次序取决于建立邻接表的算法，以及顶点对

(边依附的两个顶点)的输入次序。

邻接表的优点：

(1) 便于增加和删除顶点。

(2) 便于统计边的数目。对于一个具有 n 个顶点 e 条边的图 G,时间复杂度为 $O(n+e)$。

(3) 空间效率高。对于一个具有 n 个顶点 e 条边的图 G,空间复杂度为 $O(n+e)$,适合表示稀疏图。对于稠密图,考虑到邻接表中要附加链域,因此常采取邻接矩阵表示法。

邻接表的缺点：

(1) 不便于判断顶点之间是否有边。要判定 v_i 和 v_j 之间是否有边,就需扫描 v_i 和 v_j 对应的单链表,最坏情况下的时间复杂度为 $O(e)$。

(2) 不便于计算有向图各个顶点的度。在有向图的邻接表中,顶点 v_i 的出度是其对应的单链表中的结点个数,若要计算入度则比较困难,需要遍历所有单链表的结点。与之相反,采用逆邻接表表示,计算有向图顶点的入度比较容易,但计算出度又困难了。

5.2.3 邻接多重表和十字链表

对于无向图来说,任意一条边 (v_i,v_j) 依附的两个顶点 v_i 和 v_j 是相互邻接的。所以在邻接表中,v_j 的下标会出现在 v_i 的单链表中,而 v_i 的下标也会出现在 v_j 的单链表中,这会给一些关于图的操作带来不便。例如,对已被搜索过的边 (v_i,v_j) 作记号或删除该边等,不仅需要在 v_i 的单链表中找到存储 v_j 下标的结点,还要在 v_j 的单链表中找到存储 v_i 下标的结点。在处理这种操作的问题时,采用邻接多重表(Adjacency Multilist)存储无向图更为适宜。

在邻接多重表中,需要申请一个大小为 n(与顶点数目一致)的一维数组,存放的数据元素由两个数据项组成,即顶点 v_i 的信息(如顶点的名称等)和一个指针。该指针用来指向依附于顶点 v_i 的边结点。

在边结点中有 4 个域：两个顶点域(ivex 和 jvex)分别指示边 (v_i,v_j) 依附的两个顶点的位置(在数组中的下标);指针域(ilink)指向下一个边结点,该结点存储下一条依附于 v_i 的边;指针域(jlink)指向下一个边结点,该结点存储着下一条依附于 v_j 的边。如有必要,还可以再设置两个域：数据域(info),用来存储与该边相关的信息,如权值等;标志域(mark),用来标记该边是否被搜索过。

例如,如图 5.1 所示的无向图 G_1 的邻接多重表如图 5.13 所示。在邻接多重表中,所有依附于同一顶点的边串联在同一链表中。只要从顶点 v_i 出发,就可以沿着相应的指针找出依附于该顶点的所有边。由于每条边都依附于两个顶点,导致每个边结点被同时链接在两个链表中,因此得名“邻接多重表”。

对于有向图来说,用邻接表来计算顶点的出度容易,计算其入度困难。如果采用逆邻接表表示,计算入度是容易了,但计算出度又困难了。所以,把邻接表与逆邻接表结合起来,就成了有向图的一种存储结构——十字链表(Orthogonal List)。

在十字链表中,需要申请一个大小为 n(与顶点数目一致)的一维数组,存放的数据元素由三个数据项组成,即顶点 v_i 的信息(如顶点的名称等)和两个指针(firstin 和 firstout)。这两个指针用来指向依附于顶点 v_i 的弧结点：一个是以 v_i 为弧头,另一个是

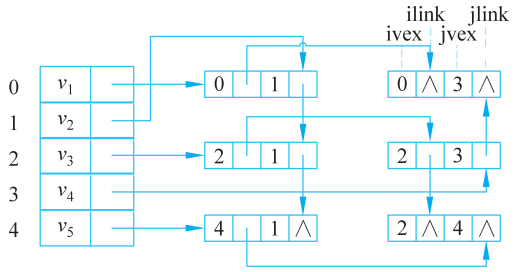


图 5.13　无向图 G_1 的邻接多重表

以 v_i 为弧尾。

在弧结点中有 4 个域：尾域(tailvex)和头域(headvex)分别指示弧尾和弧头这两个顶点的位置(在数组中的下标)；同头域(hlink)指向下一个弧结点，该结点存储弧头相同的下一条弧；同尾域(tlink)指向下一个弧结点，该结点存储弧尾相同的下一条弧。如有必要，还可以再设置一个数据域(info)，用来存储与该弧相关的信息，如权值等。

例如，如图 5.1 所示的 G_2 的十字链表如图 5.14 所示。从顶点 v_i 对应的 firstout 指针出发，沿着 tlink 指针依次相连的各个弧结点，其表示的弧都是以 v_i 为弧尾。这条链表中的弧结点的个数，就是顶点 v_i 的出度。从顶点 v_i 对应的 firstin 指针出发，沿着 hlink 指针依次相连的各个弧结点，其表示的弧都是以 v_i 为弧头。这条链表中的弧结点的个数，就是顶点 v_i 的入度。

从同一个顶点出发，弧头相同的弧在一个链表上，弧尾相同的弧在另一个链表上。两个链表形成了一个类似十字交叉的路径，因此得名“十字链表”。注意：弧结点所在的链表并非循环链表，结点之间的相对位置也是自然形成的，不一定按照顶点序号排列。

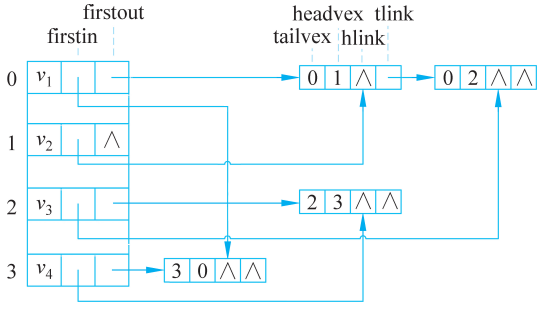


图 5.14　有向图 G_2 的十字链表

5.3　图的遍历

与树的遍历类似，图的遍历(Traversing Graph)是从图中某一顶点出发访遍图中其余顶点，且使每个顶点仅被访问一次。然而，图的遍历要比树的遍历复杂得多。因为图的任一顶点都可能和其余的顶点相邻接。所以在访问了某个顶点之后，可能会沿着某条路径又回到该顶点上。例如，如图 5.1(a)所示的 G_1，由于图中存在回路，因此在访问了 v_1、

v_2、v_3、v_4 之后，沿着边(v_1，v_4)又访问到了 v_1。

为了避免同一顶点被访问多次，在遍历图的过程中必须记下每个已访问过的顶点。为此，设置一个辅助数组 visited[ ]，其 n 个分量的初始值均为 0 或者“false”，一旦访问了顶点 v_i，立即将其对应的 visited[i]置为 1 或者“true”。

根据搜索策略的不同，图的遍历算法通常分为两种：深度优先搜索和广度优先搜索。它们对无向图和有向图都适用，本节主要以无向图为例进行探讨。

5.3.1 深度优先搜索

深度优先搜索(Depth First Search，DFS)的策略和走迷宫一样，沿着一条路径一直走下去，直到无法行进时，就回退到刚刚访问过的顶点，继续寻找其他路径。对于一个连通图，深度优先搜索遍历的过程如下。

(1) 从图中某个顶点 v_i 出发，访问 v_i。

(2) 寻找刚访问过的顶点的第一个未被访问的邻接点[①]，访问该顶点。以该顶点为新顶点，重复此步骤，直至刚访问过的顶点没有未被访问的邻接点为止。

(3) 返回前一个访问过的且仍有未被访问的邻接点的顶点，找出该顶点的下一个未被访问的邻接点，访问该顶点。

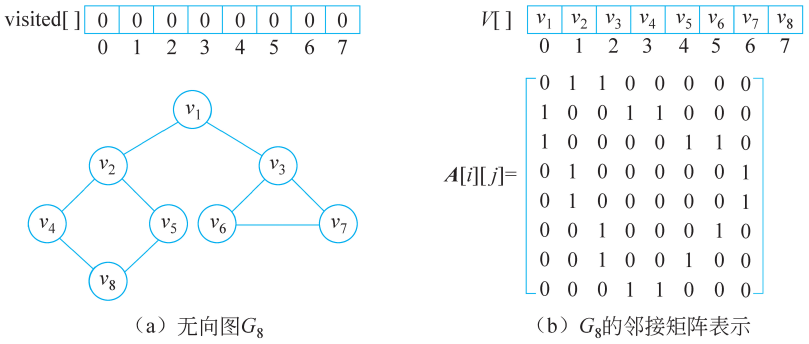
(4) 重复步骤(2)和(3)，直至图中所有顶点都被访问过，搜索结束。

例如，对如图 5.15(a)所示的无向图 G_8 进行深度优先搜索遍历，存储结构是如图 5.15(b)所示的邻接矩阵，具体过程分析如下。

(1) 从图中某个顶点出发，可以选择从 v_1 出发，先访问 v_1。

(2) 在访问了顶点 v_1 之后，选择第一个未被访问的邻接点 v_2，访问 v_2。以 v_2 为新顶点，重复此步骤，访问 v_4、v_8、v_5。在访问了 v_5 之后，由于 v_5 的邻接点都已被访问，此步骤结束。如图 5.15(c)所示，以带箭头的实线表示遍历时的访问路径，以顶点旁的数字表示访问次序，将已访问顶点对应的 visited[i]置为 1。

(3) 搜索从 v_5 回退到 v_8，由于 v_8 的邻接点都已被访问，继续回退到 v_4，同样还要回退到 v_2，直至 v_1。如图 5.15(d)所示，以带箭头的虚线表示回溯的路径。



(a) 无向图G_8　　(b) G_8的邻接矩阵表示

图 5.15　无向图 G_8 的深度优先搜索遍历

① 哪一个邻接点是第一个未被访问，要依据图的存储结构而定。