

第 5 章 图

本章概述	图结构是一种比树结构更复杂的非线性结构,在图结构中,任意两个顶点之间都可能有关系。图结构具有极强的表达能力,可用于描述各种复杂的数据对象。图的应用十分广泛,典型的应用领域有电路分析、项目规划、鉴别化合物、统计力学、遗传学、人工智能、语言学等。本章是本课程的难点和重点。 本章由实际问题引出图结构,介绍图的定义、分类和基本术语,讨论图的邻接矩阵存储和邻接表存储,讨论图的遍历操作及具体实现,最后介绍图的 4 个经典应用:最小生成树、最短路径、拓扑排序和关键路径
教学重点	图的基本术语;图的存储表示;图的遍历;图的经典应用
教学难点	图的遍历算法;Prim 算法;Kruskal 算法;Dijkstra 算法;Floyd 算法;拓扑排序算法;关键路径算法
教学目标	(1) 解释图的定义和基本术语,设计图的抽象数据类型定义; (2) 辨析图的深度优先和广度优先遍历方法,针对图结构给出遍历结果; (3) 辨析图的邻接矩阵和邻接表存储方法,分析查找邻接点等操作的执行过程和效率,画出存储示意图,描述存储结构定义; (4) 基于图的邻接矩阵和邻接表存储,设计构造图、遍历等算法; (5) 陈述 Prim 算法和 Kruskal 算法的基本思想,说明贪心选择策略,描述算法执行过程中存储单元的变化,解释算法的程序实现,分析时间性能; (6) 陈述 Dijkstra 算法的基本思想,设计存储结构,描述求解过程中存储单元的变化,解释 Dijkstra 算法的程序实现,评价时间性能; (7) 陈述 Floyd 算法的基本思想,建立递推关系式,描述求解过程中存储单元的变化,解释 Floyd 算法的程序实现,评价时间性能; (8) 解释 AOV 网的定义,说明拓扑序列的含义,陈述拓扑排序算法的基本思想,描述算法执行过程中存储单元的变化,评价时间性能; (9) 辨析 AOV 网和 AOE 网,论证关键路径和关键活动对工程进度的影响,分析关键路径的求解思想,给出算法执行过程中存储单元的变化



本章源代码

5.1 引言



在图结构中,任意两个顶点之间都可能有关系,可用于描述各种复杂的数据对象。在实际应用中,很多问题抽象出的数据模型是图结构。下面请看两个例子。



课件 5-1

【例 5-1】 七巧板涂色问题。假设有如图 5-1(a)所示七巧板,使用至多 4 种不同颜色对七巧板涂色,要求每个区域涂一种颜色,相邻区域的颜色互不相同。

【想法——数据模型】 为了识别不同区域的相邻关系,可以将七巧板的每个区域看成一个顶点,如果两个区域相邻,则这两个顶点之间有边相连,从而将七巧板抽象为图结构,如图 5-1(b)所示。这种图结构有什么特点? 如何进行存储? 如何求得涂色方案呢?

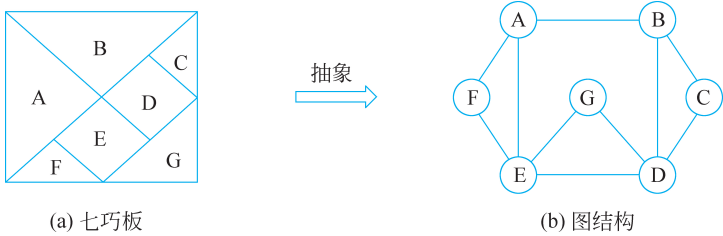


图 5-1 七巧板及其数据模型

【例 5-2】 农夫过河问题。一个农夫带着一只狼、一只羊和一筐菜,想从河一边(左岸)乘船到另一边(右岸)。由于船太小,农夫每次只能带一样东西过河。但是如果没有农夫看管,则狼会吃羊,羊会吃菜。农夫怎样过河才能把每样东西安全地送过河呢?

【想法——数据模型】 在某一时刻,农夫、狼、羊和菜或者在河的左岸,或者在河的右岸,可以用 0 表示在河的左岸,用 1 表示在河的右岸,如图 5-2(a)所示。例如,1010 表示农夫和羊在河的右岸、狼和菜在河的左岸。将每一个可能的状态抽象为一个顶点,边表示状态转移发生的条件,从而将农夫过河问题抽象为一个图结构,如图 5-2(b)所示。这种图结构有什么特点? 如何进行存储? 如何求解得到过河方案呢?

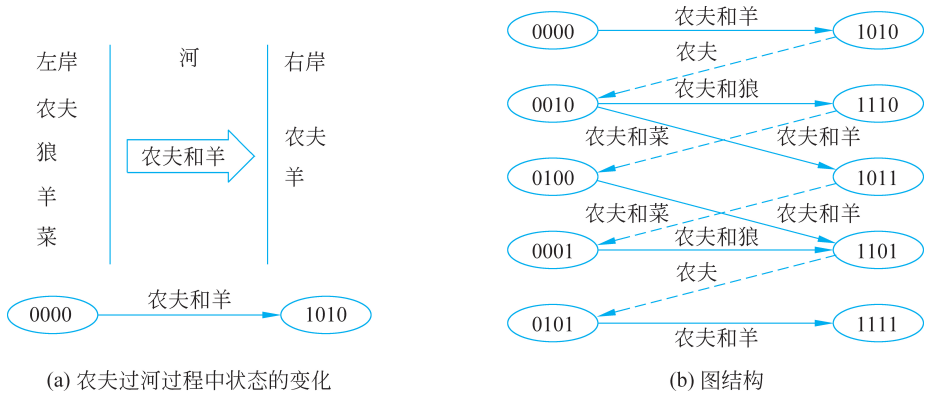


图 5-2 农夫过河问题及其数据模型

5.2 图的逻辑结构



课件 5-2

5.2.1 图的定义和基本术语

在图中常常将数据元素称为**顶点**^①(vertex)。

1. 图的定义

图(graph)是由顶点的集合和顶点之间边的集合组成^②,通常表示为

$$G = (V, E)$$

其中, G 表示一个图, V 是顶点的集合, E 是顶点之间边的集合。例如,在图 5-3(a)所示 G_1 中,顶点集合 $V = \{v_0, v_1, v_2, v_3, v_4\}$,边的集合 $E = \{(v_0, v_1), (v_0, v_3), (v_1, v_2), (v_1, v_4), (v_2, v_3), (v_2, v_4)\}$ 。

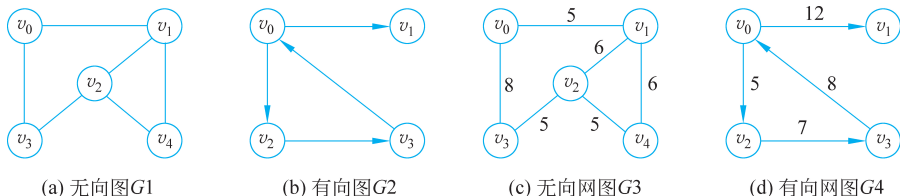


图 5-3 图的示例

在图中,若顶点 v_i 和 v_j 之间的边没有方向,则称这条边为**无向边**,用无序偶对 (v_i, v_j) 表示;若从顶点 v_i 到 v_j 的边有方向,则称这条边为**有向边**(也称为**弧**,以区别于无向边),用有序偶对 $\langle v_i, v_j \rangle$ 表示, v_i 称为**弧尾**, v_j 称为**弧头**。如果图的任意两个顶点之间的边都是无向边,则称该图为**无向图**(undirected graph),否则称该图为**有向图**(directed graph)。例如,图 5-3(a)所示是一个无向图,图 5-3(b)所示是一个有向图^③。

在图中,**权**(weight)通常是对边赋予的有意义的数值量^④,在实际应用中,权可以有具体的含义。例如,对于城市交通线路图,边上的权表示该条线路的长度或者等级;对于工程进度图,边上的权表示活动所需的时间,等等。边上带权的图称为**带权图**或**网图**(network graph)。例如,图 5-3(c)所示是一个无向网图,(d)所示是一个有向网图。

2. 图的基本术语

(1) 邻接、依附。

在无向图中,对于任意两个顶点 v_i 和 v_j ,若存在边 (v_i, v_j) ,则称顶点 v_i 和 v_j 互为**邻接点**^⑤(adjacent),同时称边 (v_i, v_j) **依附**(adhere)于顶点 v_i 和 v_j 。

① 有些教材将图的顶点称为**节点**(node)。新华字典对结点的定义是直线或曲线的终点或交点;对节点的定义是电路中连接三个或三个以上的点,物体的分段或两段之间连接的部分;对顶点的定义是角的两条边的交点。英文的结点和节点都是 node,都表示交叉点、集结点。

② $V = \Phi$ 的图称为空图, $V \neq \Phi$ 但 $E = \Phi$ 的图称为零图。在数据结构中,空表、空栈、空队列、空树、空二叉树、空图、零图等均表示一种可能的状态,一般作为条件判断。

③ 若图 $G = (V, E)$ 中同时包含无向边和有向边,则称为混合图。将每条无向边等效地替换为对称的一对有向边,可将混合图转化为有向图。

④ 在图中,权可以是边的属性也可以是顶点的属性。本书只讨论边上带权的图,且权值是非负整数的情况。

⑤ 在线性结构中,数据元素之间的逻辑关系表现为前驱——后继;在树结构中,结点之间的逻辑关系表现为双亲——孩子;在图结构中,顶点之间的逻辑关系表现为邻接。

在有向图中,对于任意两个顶点 v_i 和 v_j ,若存在弧 $\langle v_i, v_j \rangle$,则称顶点 v_i 邻接到 v_j ,顶点 v_j 邻接自 v_i ,同时称弧 $\langle v_i, v_j \rangle$ 依附于顶点 v_i 和 v_j 。在不致混淆的情况下,通常称 v_j 是 v_i 的邻接点。

(2) 顶点的度、入度、出度。

在无向图中,顶点 v 的**度**(degree)是指依附于该顶点的边的个数,记为 $TD(v)$ 。在具有 n 个顶点 e 条边的无向图中,有式(5-1)成立:

$$\sum_{i=0}^{n-1} TD(v_i) = 2e \quad (5-1)$$

在有向图中,顶点 v 的**入度**(in-degree)是指以该顶点为弧头的弧的个数,记为 $ID(v)$;顶点 v 的**出度**(out-degree)是指以该顶点为弧尾的弧的个数,记为 $OD(v)$ 。在具有 n 个顶点 e 条边的有向图中,有式(5-2)成立:

$$\sum_{i=0}^{n-1} ID(v_i) = \sum_{i=0}^{n-1} OD(v_i) = e \quad (5-2)$$

(3) 无向完全图、有向完全图。

在无向图中,如果任意两个顶点之间都存在边,则称该图为**无向完全图**(undirected complete graph)。含有 n 个顶点的无向完全图有 $n \times (n-1)/2$ 条边。

在有向图中,如果任意两个顶点之间都存在方向互为相反的两条弧,则称该图为**有向完全图**(directed complete graph)。含有 n 个顶点的有向完全图有 $n \times (n-1)$ 条边。

(4) 稠密图、稀疏图。

称边数很少的图为**稀疏图**(sparse graph),反之,称为**稠密图**①(dense graph)。

(5) 路径、路径长度、回路。

在无向图 $G=(V,E)$ 中,顶点 v_p 到 v_q 之间的**路径**(path)是一个顶点序列 $v_p=v_{i_0}v_{i_1}\cdots v_{i_m}=v_q$,其中, $(v_{ij-1}, v_{ij}) \in E (1 \leq j \leq m)$;如果 G 是有向图,则 $\langle v_{ij-1}, v_{ij} \rangle \in E (1 \leq j \leq m)$ 。路径上边的数目称为**路径长度**(path length)。第一个顶点和最后一个顶点相同的路径称为**回路**(circuit)。显然,在图中路径可能不唯一,回路也可能不唯一。

(6) 简单路径、简单回路。

在路径序列中,顶点不重复出现的路径称为**简单路径**(simple path)。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**(simple circuit)。通常情况下,路径指的都是简单路径,回路指的都是简单回路。

(7) 子图。

对于图 $G=(V,E)$ 和 $G'=(V',E')$,如果 $V' \subseteq V$ 且 $E' \subseteq E$,则称图 G' 是 G 的**子图**②(subgraph)。图 5-4 给出了子图的示例,显然,一个图可以有多个子图。

(8) 连通图、连通分量。

在无向图中,若顶点 v_i 和 $v_j (i \neq j)$ 之间存在路径,则称 v_i 和 v_j 是连通的。若任意顶点 v_i 和 $v_j (i \neq j)$ 之间均有路径,则称该图是**连通图**(connected graph)。例如,图 5-4(a)所示是连通图,图 5-5(a)所示是非连通图。非连通图的极大连通子图称为**连通分量**(connected

① 稀疏和稠密本身就是模糊的概念,稀疏图和稠密图常常是相对而言的。显然,最稀疏图的边数是 0,最稠密图是完全图,边数达到最多。

② 通俗地说,子图是原图的一部分,是由原图中一部分顶点和这些顶点之间的一部分边构成的图。

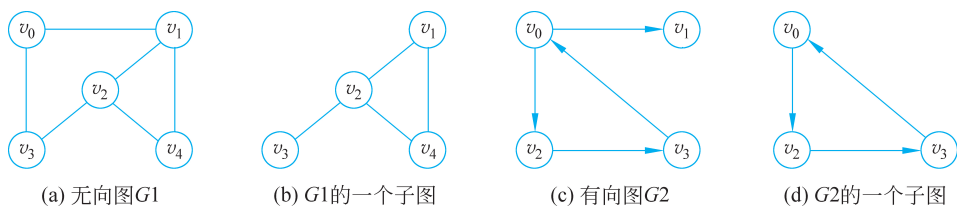


图 5-4 子图的例子

component), 极大的含义是指子图在满足连通的条件下, 包括所有连通的顶点以及与这些顶点相关联的所有边。图 5-5(a) 所示非连通图有两个连通分量, 如图 5-5(b) 所示。

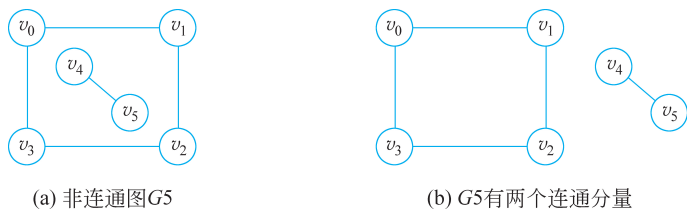


图 5-5 非连通图及连通分量

(9) 强连通图、强连通分量

在有向图中, 对任意顶点 v_i 和 v_j ($i \neq j$), 若从顶点 v_i 到 v_j 均有路径, 则称该有向图是 **强连通图** (strongly connected graph)。图 5-6(a) 所示是强连通图, 图 5-6(b) 所示是非强连通图。非强连通图的极大强连通子图^①称为 **强连通分量** (strongly connected component)。图 5-6(b) 所示非强连通图有两个强连通分量, 如图 5-6(c) 所示。

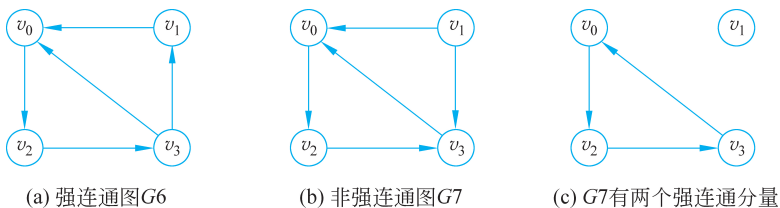


图 5-6 强连通图、非强连通图及强连通分量

5.2.2 图的抽象数据类型定义

图是一种与具体应用密切相关的数据结构, 其基本操作往往随应用不同而有很大差别。下面给出一个图的抽象数据类型定义的例子, 简单起见, 基本操作仅包含图的遍历, 针对具体应用, 需要重新定义其基本操作。

```
ADT Graph
DataModel
    顶点的集合和顶点之间边的集合
Operation
    CreateGraph
```

^① 此处极大的含义同连通分量。

```

    输入: n 个顶点 e 条边
    功能: 图的建立
    输出: 构造一个含有 n 个顶点 e 条边的图

DestroyGraph
    输入: 无
    功能: 图的销毁
    输出: 释放图占用的存储空间

DFTTraverse
    输入: 遍历的起始顶点 v
    功能: 从顶点 v 出发深度优先遍历图
    输出: 图的深度优先遍历序列

BFTTraverse
    输入: 遍历的起始顶点 v
    功能: 从顶点 v 出发广度优先遍历图
    输出: 图的广度优先遍历序列

endADT

```

5.2.3 图的遍历操作

图的**遍历**(traverse)是从图中某顶点出发,对图中所有顶点访问^①一次且仅访问一次。由于图结构本身的复杂性,所以图的遍历操作比较复杂,须解决的关键问题如下。

① 在图中,没有一个确定的开始顶点,任意一个顶点都可作为遍历的起始顶点,如何选取遍历的起始顶点?

② 从某个顶点出发可能到达不了所有其他顶点,例如非连通图,从一个顶点出发,只能访问它所在连通分量上的所有顶点,如何才能遍历图的所有顶点?

③ 由于图中可能存在回路,某些顶点可能会被重复访问,如何避免遍历不会因回路而陷入死循环?

④ 在图中,一个顶点可以和其他多个顶点相邻接,当这样的顶点访问过后,如何选取下一个要访问的顶点?

问题①的解决:既然图中没有确定的开始顶点,就可以从图中任一顶点出发,不妨将顶点进行编号,先从编号小的顶点开始。在图中,由于任何两个顶点之间都可能存在边,顶点没有确定的先后次序,所以,顶点的编号不唯一。在图的存储实现上,一般采用一维数组存储图的顶点信息,因此,可以用顶点的存储位置(即下标)表示该顶点的编号。为了和 C++ 语言的数组保持一致,顶点的编号从 0 开始。

问题②的解决:要遍历图中所有顶点,只需多次重复从某一顶点出发进行图的遍历。以下仅讨论从某一顶点出发遍历图的问题。

问题③的解决:为了在遍历过程中区分顶点是否已被访问,设置一个访问标志数组 `visited[n]` (n 为图中顶点的个数),其初值为未被访问标志 0,如果某顶点 i 已被访问,则将该顶点的访问标志 `visited[i]` 置为 1。

^① 此处访问的含义同树的遍历中访问的含义。不失一般性,在此将访问定义为输出顶点的数据信息。

问题④的解决：这就是遍历次序的问题。图的遍历通常有深度优先遍历和广度优先遍历两种方式,这两种遍历次序对无向图和有向图都适用。

深度优先遍历^①(depth-first traverse)类似于树的先序遍历。从图中某顶点 v 出发进行深度优先遍历的基本思想如下。

- ① 访问顶点 v ;
 - ② 从 v 的未被访问的邻接点中选取一个顶点 w ,然后从 w 出发进行深度优先遍历;
 - ③ 重复上述两步,直至图中所有和 v 有路径相通的顶点都被访问到。
- 显然,深度优先遍历图是一个递归过程,算法思想用伪代码描述如下^②:

算法: DFTraverse
输入: 顶点的编号 v
输出: 无

```
1.访问顶点  $v$ ; 修改标志  $visited[v]=1$ ;  
2.  $w$ =顶点  $v$  的第一个邻接点;  
3. while ( $w$  存在)  
    3.1 if ( $w$  未被访问) 从顶点  $w$  出发递归执行该算法;  
    3.2  $w$ =顶点  $v$  的下一个邻接点;
```

图 5-7 给出了对无向图进行深度优先遍历的过程示例。在访问 v_0 后选择未曾访问的邻接点 v_1 ,访问 v_1 后选择未曾访问的邻接点 v_4 。由于 v_4 没有未曾访问的邻接点,递归返回到顶点 v_1 ,选择未曾访问的邻接点 v_2 。以此类推,得到深度优先遍历序列为 $v_0 v_1 v_4 v_2 v_3 v_5$ 。

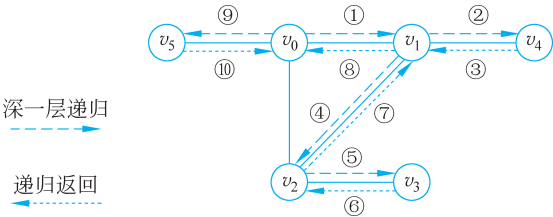


图 5-7 无向图的深度优先遍历示例



动画视频

广度优先遍历^③(breadth-first traverse)类似于树的层序遍历。从图中某顶点 v 出发进行广度优先遍历的基本思想如下。

- ① 访问顶点 v ;
- ② 依次访问 v 的各个未被访问的邻接点 $v_1, v_2, \dots, v_k$;
- ③ 分别从 $v_1, v_2, \dots, v_k$ 出发依次访问它们未被访问的邻接点,直至图中所有与顶点 v 有路径相通的顶点都被访问到。

广度优先遍历以顶点 v 为起始点,由近至远,依次访问和 v 有路径相通且路径长度为 $1, 2, \dots$ 的顶点。为了使“先被访问顶点的邻接点”先于“后被访问顶点的邻接点”被访问,设

① 深度优先遍历算法由约翰·霍普克洛夫特和罗伯特·陶尔扬发明。当他们的研究成果在 ACM 上发表以后,引起学术界很大的轰动,深度优先遍历算法在信息检索、人工智能等领域得到成功应用。

② 该算法不依赖于图的存储结构,不涉及具体的实现细节,仅描述算法的基本思路。读者要学习并掌握这种用伪代码描述顶层算法(或算法思想)的方法。

③ 广度优先遍历算法由美国计算机科学家 Edward F. Moore 在 1950 年发明,此外他还发明了有限状态自动机。



动画视频

置队列存储已被访问的顶点。例如,对图 5-8 所示有向图进行广度优先遍历,访问 v_0 后将 v_0 入队;将 v_0 出队并依次访问 v_0 的未曾访问的邻接点 v_1 和 v_2 ,将 v_1 和 v_2 入队;将 v_1 出队并访问 v_1 的未曾访问的邻接点 v_4 ,将 v_4 入队;重复上述过程,得到顶点访问序列 $v_0 v_1 v_2 v_4 v_3$,图 5-9 给出了广度优先遍历过程中队列的变化。

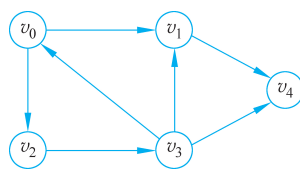


图 5-8 一个有向图

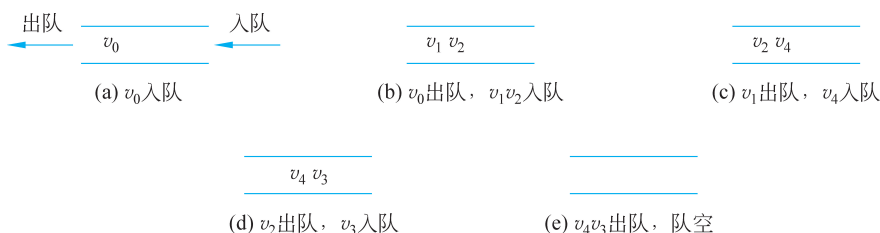


图 5-9 广度优先遍历过程中队列的变化

广度优先遍历的算法思想用伪代码描述如下：

```

算法: BFTraverse
输入: 顶点的编号 v
输出: 无
1. 队列 Q 初始化;
2. 访问顶点 v; 修改标志 visited[v]=1; 顶点 v 入队列 Q;
3. while (队列 Q 非空)
3.1 v=队列 Q 的队头元素出队;
3.2 w=顶点 v 的第一个邻接点;
3.3 while (w 存在)
3.3.1 如果 w 未被访问,则
访问顶点 w; 修改标志 visited[w]=1; 顶点 w 入队列 Q;
3.3.2 w=顶点 v 的下一个邻接点;
    
```



课件 5-3

5.3 图的存储结构及实现



图是一种复杂的数据结构,任意两个顶点之间都可能存在边,所以无法通过顶点的存储位置反映顶点之间的邻接关系,因此图没有顺序存储结构。从图的定义可知,一个图包括两部分:顶点的信息以及顶点之间边的信息。无论采用什么方法存储图,都要完整、准确地表示这两方面的信息。一般来说,图的存储结构应根据具体问题的要求来设计,下面介绍两种常用的存储结构——邻接矩阵和邻接表。

5.3.1 邻接矩阵

1. 邻接矩阵的存储结构

图的**邻接矩阵**(adjacency matrix)存储也称数组表示法,用一个一维数组存储图中的顶点,用一个二维数组存储图中的边(即各顶点之间的邻接关系),存储顶点之间邻接关系的二

维数组称为邻接矩阵。设图 $G=(V,E)$ 有 n 个顶点,则邻接矩阵是一个 $n \times n$ 的方阵,定义为

$$\text{edge}[i][j] = \begin{cases} 1 & \text{若 } (v_i, v_j) \in E \text{ 或 } \langle v_i, v_j \rangle \in E \\ 0 & \text{否则} \end{cases} \quad (5-3)$$

若 G 是网图,则邻接矩阵定义为

$$\text{edge}[i][j] = \begin{cases} w_{ij} & \text{若 } (v_i, v_j) \in E \text{ 或 } \langle v_i, v_j \rangle \in E \\ 0 & \text{若 } i = j \\ \infty & \text{否则} \end{cases} \quad (5-4)$$

其中, w_{ij} 表示边 (v_i, v_j) 或弧 $\langle v_i, v_j \rangle$ 上的权值; ∞ 表示一个计算机允许的、大于所有边上权值的数。图 5-10 所示为一个无向图及其邻接矩阵存储示意图,图 5-11 所示为一个有向网图及其邻接矩阵存储示意图。

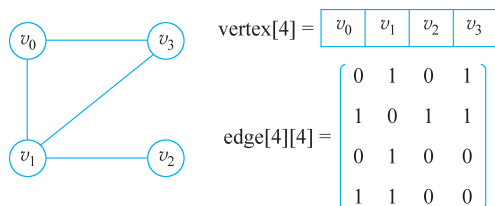


图 5-10 无向图及其邻接矩阵存储示意图

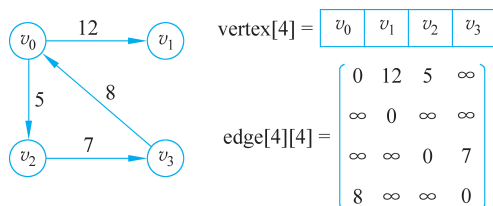


图 5-11 有向网图及其邻接矩阵存储示意图

显然,无向图的邻接矩阵一定是对称矩阵,而有向图的邻接矩阵则不一定对称。在图的邻接矩阵存储中容易实现下述基本操作:

① 对于无向图,顶点 i 的度等于邻接矩阵中第 i 行(或第 i 列)非零元素的个数。对于有向图,顶点 i 的出度等于邻接矩阵中第 i 行非零元素的个数;顶点 i 的入度等于邻接矩阵中第 i 列非零元素的个数。

② 判断顶点 i 和 j 之间是否存在边,只需测试邻接矩阵中相应位置的元素 $\text{edge}[i][j]$,若其值为 1,则有边;否则,顶点 i 和 j 之间不存在边。

③ 查找顶点 i 的所有邻接点,扫描邻接矩阵的第 i 行,若 $\text{edge}[i][j]$ 的值为 1,则顶点 j 是顶点 i 的邻接点。

2. 邻接矩阵的实现

将图的抽象数据类型定义在邻接矩阵存储结构下用 C++ 的类实现,下面给出邻接矩阵存储的类定义,其中成员变量实现图的邻接矩阵存储,成员函数实现图的基本操作。

```
const int MaxSize = 10;           //图中最多顶点个数
template <typename DataType>
class MGraph
{
public:
    MGraph(DataType a[ ], int n, int e); //构造函数
    ~MGraph();                          //析构函数
    void DFSTraverse(int v);            //深度优先遍历图
```

```
void BFTraverse(int v);           //广度优先遍历图
private:
    DataType vertex[MaxSize];    //存放图中顶点的数组
    int edge[MaxSize][MaxSize];  //存放图中边的数组
    int vertexNum, edgeNum;      //图的顶点数和边数
};
```

下面讨论邻接矩阵基本操作的实现。

(1) 构造函数——图的建立。

建立一个含有 n 个顶点 e 条边的图,顶点信息由参数 $a[n]$ 给出,边的信息由键盘输入,假设建立无向图,算法用伪代码描述如下:

算法: 构造函数 MGraph
 输入: 顶点的数据信息 $a[n]$, 顶点个数 n , 边的条数 e
 输出: 图的邻接矩阵存储

1. 存储图的顶点个数和边的条数;
2. 将顶点信息存储在一维数组 $vertex$ 中;
3. 初始化邻接矩阵 $edge$;
4. 依次输入每条边并存储在邻接矩阵 $edge$ 中:
 - 4.1 输入边依附的两个顶点的编号 i 和 j ;
 - 4.2 将 $edge[i][j]$ 和 $edge[j][i]$ 的值置为 1;

建立无向图邻接矩阵的构造函数定义如下。

```
template <typename DataType>
MGraph<DataType>:: MGraph(DataType a[ ], int n, int e)
{
    int i, j, k;
    vertexNum = n; edgeNum = e;
    for (i = 0; i < vertexNum; i++)           //存储顶点
        vertex[i] = a[i];
    for (i = 0; i < vertexNum; i++)           //初始化邻接矩阵
        for (j = 0; j < vertexNum; j++)
            edge[i][j] = 0;
    for (k = 0; k < edgeNum; k++)             //依次输入每一条边
    {
        cin >> i >> j;                       //输入边依附的两个顶点的编号
        edge[i][j] = 1; edge[j][i] = 1;      //置有边标志
    }
}
```

(2) 析构函数——图的销毁。

图的邻接矩阵存储是静态存储分配,在图变量退出作用域时自动释放所占内存单元,因此,图的邻接矩阵存储无须销毁,析构函数为空。