数据結构講义

（下册）

唐策善
牛忠荣 编

USTC

中国科学技术大学六系

1981年5月

数据结构讲义下册目录

第六章　图

图(Graph)是较线性表和树更为复杂的一种数据结构。在线性表中，每个数据元素只有一个前趋和一个后继；在树形结构中，数据元素之间有明显的层次关系。且每一层的数据元素可以和它下面一层的多个元素相联结。但只能和它上面一层的一个元素相联结。在图形结构中，结点间的联系是任意的，任何一个数据元素都可以和其它元素相联结。由此可见，可以把树看成一种特殊形式的图。

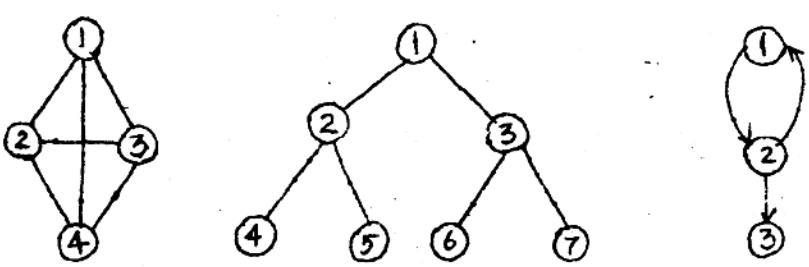
本课程不再讨论图的理论，因为它是“离散数学”的内容之一。本章主要讨论几种常用的图和常用的存贮结构及其相应的算法。首先简单介绍一些将要用到的基本术语。

6－1　基本术语

图、有向图和网　图G由两个集合V(G)和E(G)组成。记作

$$G=(V,E)$$

其中：V(G)是顶点(Vertex)的有限非空集合；E(G)是边(Edge)的有限集。且边是顶点的无序对或有序对。例如，图6－1是三个简单图的例子。



G1　　G2　　G3

图6－1　图的示例

G_1和G_2是无向图（undirected graph或graph），G_3是有向图（divected graph或digraph）。在无向图中，表示边的顶点对是无顺序的，即（v_1，v_2）和（v_2，v_1）表示同一条边。例如图6－1中的G_1就是无序对：

$$V(G_1)=\{v_1, v_2, v_3, v_4\}$$

$$E(G_1)=\{(v_1, v_2), (v_1, v_3), (v_1, v_4), (v_2, v_3), (v_2, v_4), (v_3, v_4)\}$$

在有向图中的每条边（又称弧（arc），是用有向对$\langle v_i, v_j\rangle$来表示的，v_i是弧的尾（tail）或初始点（initial node）；v_j是弧的头（head）或终端结点（termind node）。在图上画时用从尾到头的箭头表示。例如图6－1中的G_3就是有序对：

$$V(G_3)=\{v_1, v_2, v_3\}$$

$$E(G_3)=\{\langle v_1, v_2\rangle \langle v_2, v_1\rangle \langle v_2, v_3\rangle\}$$

其中$\langle v_1, v_2\rangle$和$\langle v_2, v_1\rangle$是两条不同的弧。

边（v_1，v_2）和弧$\langle v_1, v_2\rangle$的两个顶点必须是不同的。在n个顶点的无向图中，若每一个顶点和其它n－1个顶点之间都有边，则有$n\times(n-1)/2$条边。这样的图称为完全图（Completed graph）。例如G_1就是有4个顶点的完全图。在n个顶点的有向图中，边的最大数为$n\times(n-1)$。具有$n(n-1)$条弧的有向图，称为有向完全图。

有时图的边具有与它相关的数。这种与图的边相关的数叫做权（weight）。这些权可以表示从一个顶点到另一个顶点的距离或花费的代价。这类图又叫做网（Network）。

子图　假设有两个图G和G′，且满足下述条件：

$$V(G')\subseteq V(G)$$

和

$$E(G') \subseteq E(G)$$

则称G′为G的子图（Subgraph）。例如，图6－2是子图的一些例子。

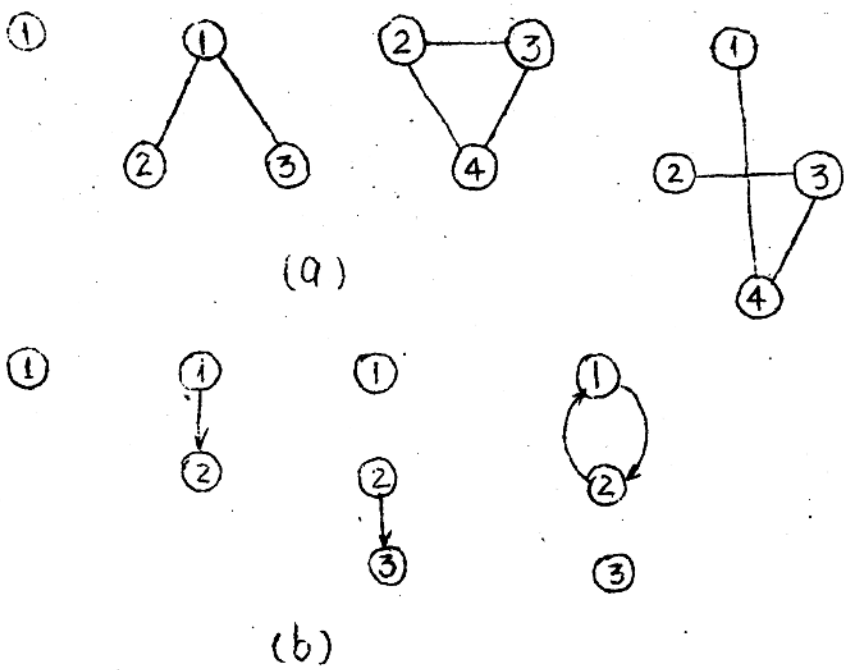


图6－2　子图示例

（a）　G_1的一些子图。　（b）　G_3的一些子图

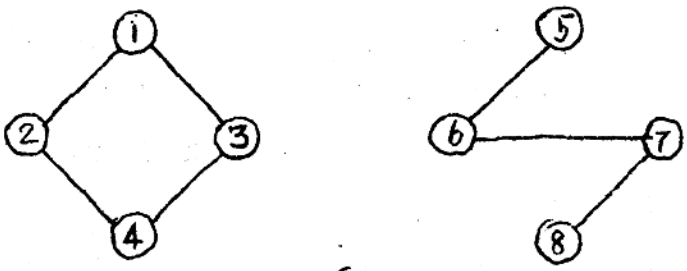
度、入度和出度　若(v_1, v_2)是E(G)中的一条边，则称顶点v_1和v_2是邻接的（adjacent）。边(v_1, v_2)是依附（incident）于顶点v_1和v_2。顶点的度（Degree）是依附于该顶点的边的数目。例如，G_1中顶点v_1的度是3。若G是有向图，则顶点v的入度（Indegree）是以v为头的弧的数目；出度（Outdegree）是以v为尾的弧的数目。入度和出度两者之和恰是它的度。例如，G_3中顶点v_2的入度ID(v_2)为1，出度OD(v_2)为2，度

$TD(v_3)$ 为3。若图G中有n个顶点。e条边。且每个顶点 v_i 的度为 $TD(v_i)$。则有

$$e=\frac{1}{2}\sum_{i=1}^{n} TD(v_i)$$

路径和环路　在图G中从顶点 v_p 到顶点 v_q 的路径（Path）是顶点序列（v_p，v_{i1}，v_{i2}，……，v_{in}，v_q）。且（v_p，v_{i1}），（v_{i1}，v_{i2}），……，（v_{in}，v_q）是E(G)中的边。若G是有向图。则路径也是有向的。由弧$<v_p, v_{i1}>$，$<v_{i1}, v_{i2}>$，……$<v_{in}, v_q>$组成。路径长度是路径上边的数目。除了第一个和最后一个顶点之外。序列中其余顶点各不相同的路径。称为简单路径。第一个顶点和最后一个顶点相同的路径和简单路径。称为环路（Cycle）和简单环路。

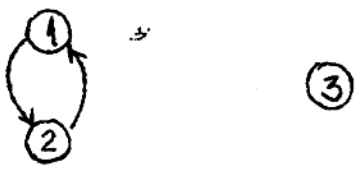
连通图和图的连通分量　在无向图G中。若从 v_1 到 v_2 有路径。则称 v_1 和 v_2 是连通的。若对于V(G)中每一对不同的顶点 v_i 和 v_j 都连通。则称G是连通图（Connected graph）。如图6—1中 G_1 和 G_2 是连通图。而图6－3中 G_4 是非连通图。但 G_4 有两个连通分量（Connected Components）。所谓连通分量指的是无向图中的极大连通子图。



G_4

图6－3　具有两个连通分量的图

在有向图中。若对于V(G)中每一对不同的顶点都存在一条从v_i到v_j和从v_j到v_i的路径。则称G是强连通图。有向图中的极大强连通子图是它的强连通分量。例如，G_3不是强连通图。但它有两个强连通分量。如图6－4所示。



6－4　G_3的强连通分量

6－2　图的存贮结构

按理。图亦可采用多重链表作为存贮结构。每个顶点包含一个数据域和若干个指针域。其中数据域给出图中相应顶点的有关信息。每个指针指向与其有联系的结点。但是。由于图在结构上较树更为复杂。故用这种多重链表的存贮结构不便于进行图的各种运算。因此。通常需要针对具体的图形和将作的运算来灵活取用存贮结构。其中较常用的有三种：用二维数组表示的邻接矩阵（Adjacency Matrices）和邻接表（Adjacency Lists）和邻接多重表（adjacency multilist）。

邻接矩阵

设G=(V，E)是有$n\geq 1$个顶点的图。则G的邻接矩阵是具有下列性质的n阶方阵：

$$A(i,j)=\begin{cases}1 & \text{若}(v_i,v_j)\text{或}\langle v_i,v_j\rangle\in E(G)\\ 0 & \text{反之}\end{cases}$$

例如。图 G_1 和 G_2 的邻接矩阵如图 6－5所示。

$$A_1=\begin{bmatrix}0&1&1&1\\1&0&1&1\\1&1&0&1\\1&1&1&0\end{bmatrix}\qquad A_2=\begin{bmatrix}0&1&0\\1&0&1\\0&0&0\end{bmatrix}$$

图 6－5　G_1 的邻接矩阵 A_1 和 G_2 的邻接矩阵 A_2

无向图的邻接矩阵是对称的。有向图的邻接矩阵则不一定对称。用邻接矩阵表示一个有 n 个顶点的图需要 n^2 个存贮单元。对于无向图。可考虑其对称性。仅需存入下三角（或上三角）矩阵。

借助于邻接矩阵容易判定任意两个顶点之间是否有边相连。并容易求得各个顶点的度。对无向图而言。顶点 v_i 的度是矩阵中第 i 行元素之和。即

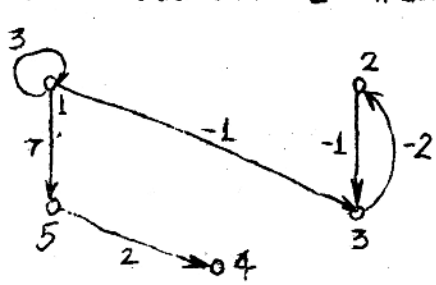
$$TD(v_i)=\sum_{j=1}^{n}A(i,j)$$

对有向图而言。第 i 行元素之和为顶点 v_i 的出度 $OD(v_i)$。第 i 列元素之和是顶点 v_i 的入度 $ID(v_i)$。

网的邻接矩阵可定义为：

$$A(i,j)=\begin{cases}W_{ij} & \text{若 } (v_i,v_j) \text{ 或} \langle v_i,v_j\rangle \in E(G)\\ 0 & \text{反之}\end{cases}$$

例如。图 6－6 列出了网和它的邻接矩阵。

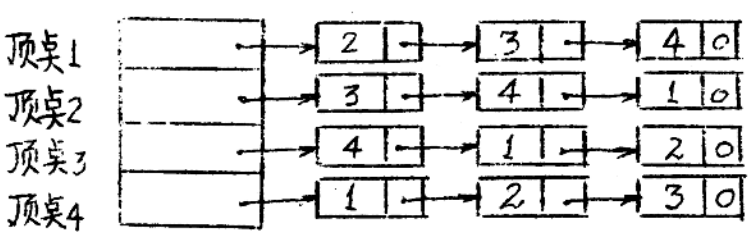


$$A=\begin{bmatrix} 3 & 0 & 0 & 0 & 7 \\ 0 & 0 & -1 & 0 & 0 \\ -1 & -2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 \end{bmatrix}$$

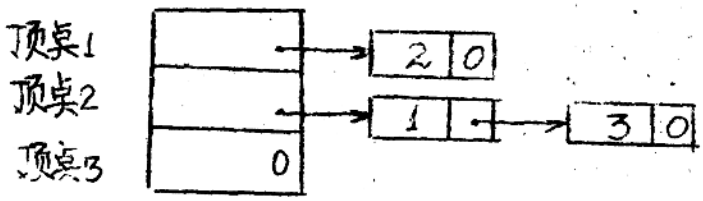
图 6－6　　网N和它的邻接矩阵

邻接表

在邻接表这种存贮结构中。对图中每个顶点建立一个链表。第1个链表中的结点是依附于顶点 v_1 的边（对有向图是以顶点 v_1 为尾的边）。每个结点由两个域组成：顶点域（vertex）。用以指示与顶点 v_1 邻接的点的序号，链域（link）。用以指向下一条边。（对于网。还需要在结点中增加一个存放权值的域）。每个链表设一表头结点。这些表头结点本身以向量形式存贮。以便随机访问任一顶点的链表。例如。G_1 和 G_3 的邻接表分别如图 6－7（a）和（b）所示。



（a）



（b）

（a）G_1 的邻接表　　（b） G_3 的邻接表

图 6－7　　邻接表示例

若无向图中有n个顶点和e条边，则它的邻接表需n个头结点和2e个表结点。每个表结点有两个（对于网则有三个）域。显然，在边稀疏的情况下，用邻接表比邻接矩阵节省存贮单元。

在无向图的邻接表中，第i个链表中的结点数即为顶点v_i的度。而对于有向图，则只是顶点v_i的出度；为了求入度，必须对整个邻接表扫描一遍，找到顶点域的值为v_i的结点的个数。显然，这是很麻烦的。为了便于确定顶点的入度，我们可以另建立一个逆邻接表，即对每个顶点v_i建立一个链结以v_i为头的边的表。图6－8画出了G_3的逆邻接表。也可以象稀疏矩阵一样，建立一个正交链表，表中每个结点表示一条弧$\langle v_i, v_j \rangle$。它有四个域：尾域v_i；头域v_j；链接以v_i为尾的边的指针域；链接以v_j为头的边的指针域。这样做实际上是将邻接矩阵看成是一个稀疏矩阵，以正交表作为它的存贮结构。

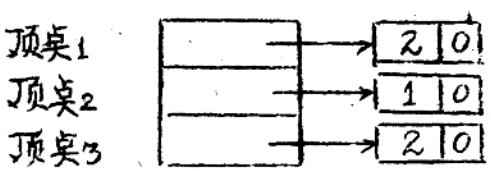


图 6－8　G_3 的逆邻接表

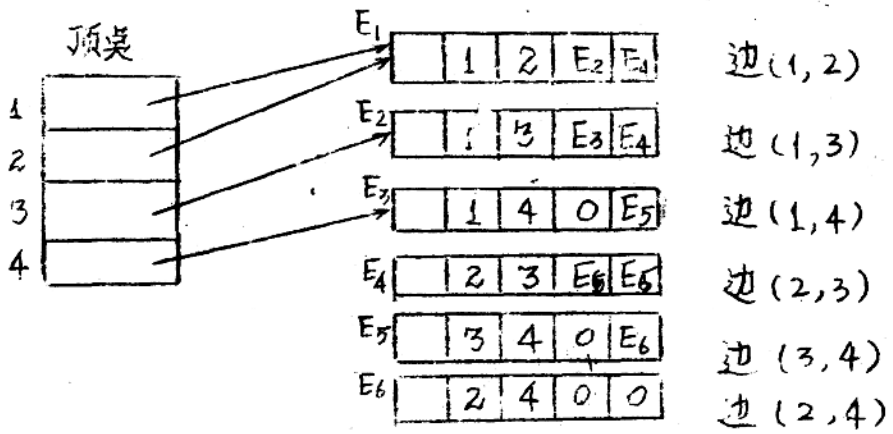
邻接多重表

在无向图的邻接表中可以看到。每一条边（v_1，v_j）有两个结点一个在v_1的链表中；一个在v_j的链表中。在较复杂的问题中。有时要对被搜索过的边作记号。从而需要同时找到表示一条边的两个结点。这样。我们就需用邻接多重表（Adjacency Multilists）作为图的存贮结构。

在邻接多重表中。每条边用一个结点表示。它由如下所示的五个域组成：

mark	v_1	v_j	v_1—Link	v_j—Link

其中：mark为标志域。用以标记该条边是否被搜索过；v_1和v_j为该边依附的两个顶点；v_1—Link 可以指向下一条依附于v_1的边；v_j—Link 用以指向下一条依附于v_j的边。在这种表中。除了增加一个标志域外，所需的存贮量和邻接表所需的相同。例如。G_1的邻接多重表。如图6－9所示。



顶点1：$E_1 \rightarrow E_2 \rightarrow E_3$

顶点2：$E_1 \rightarrow E_4 \rightarrow E_5$

顶点3：$E_2 \rightarrow E_4 \rightarrow E_6$

顶点4：$E_3 \rightarrow E_5 \rightarrow E_6$

图 6－9　G_1 的邻接多重表

6－3　遍历图和求连通分量

和树的遍历类似。在此，我们希望从图中某一点出发访问图中其余的顶点。若给定的图是连通图，则从图中任意一点出发，顺着某些边可以访问到该图中所有的顶点，且使每一个顶点仅被访问一次。这一过程就叫做遍历图（Tra--versing graph）。

然而图的遍历要比树的遍历复杂得多。由于图的任一顶点都可能和其余的顶点相邻接，故在访问了某个顶点之后，可能顺着某条边又访问到已被访问过的顶点。例如，对图 6－1中的 G_1，它的每个顶点都和其余三个顶点相邻接。在访问了 v_1、v_2 和 v_3 之后，顺着边（v_3，v_1）又访问到 v_1。因此，在遍历图的过程中，必须记下每个被访问到的顶点，以免同一顶点被访问多次。为此，我们可以对每个顶点设一个辅助函数 VISITED(W)，它的初始状态为零，一旦顶点 W 被访问，就将它置成1。

下面我们介绍两种遍历图的方法：深度优先检索和广度优先检索。

深度优先检索

无向图的深度优先检索按下列步骤进行：

首先访问初始顶点v。下一步选出一个邻接到v且未被访问过的顶点w。然后从w开始深度优先检索。当检索一直达到这样的顶点，使得与它邻接的顶点都已访问过了。于是便回到上次访问过的顶点，若此顶点还有一个未被访问过且又邻接到它上面的顶点w，就从w进行深度优先检索。当从任一个已访问过的顶点再也到达不了尚未访问过的顶点时，检索就结束了。这过程最好用递归的方式来编：

```
PROCEDURE DFS(V)
    〈给出一个具有n个顶点的无向图G=(V,E)和初始值为
      0的数组VISITED(n)。此算法访问从v开始的所有能到
      到的顶点。G和VISITED是全程量〉
    VISITED(V)←1
    FOR each vertex W adjacent to v DO
      IF VISITED(W)=0 THEN CALL DFS(W)
    END
END DFS
```

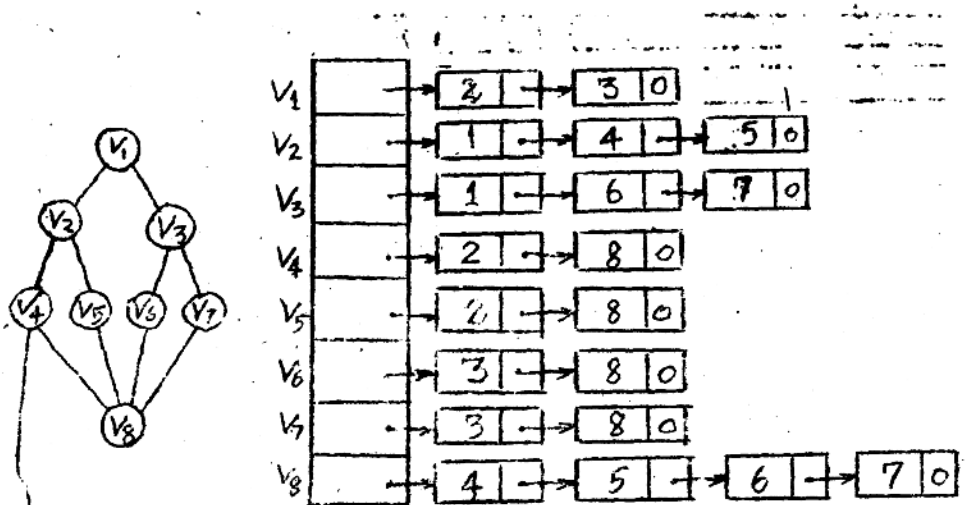
在G是用邻接表表示的情况下。邻接至v的顶点w可以通过跟随在其后的一连串链来确定之。既然算法DFS对邻接表中的每个结点至多只检验一次。现有2e个表结点。所以完成检索的时间应为O(e)。如果G是用邻接矩阵表示。那么决定所有邻接至v的顶点的时间是O(n)。因为至多有n个顶点要访问。所以总的时间为$O(n^2)$。

图6-10(a)的G_5是用图6-10(b)的邻接表表示的。假如深度优先检索从v_1开始。那么G_5的所有顶点被访问的次序是v_1，v_2，v_4，v_8，v_5，v_6，v_3，v_7。容易验证。

DFS(v_1)访问了所有连接至 v_1 的顶点。因此。所有被访问过的顶点。连同 G_6 中关联这些顶点的所有边。形成了 G 的一个连通分量。

宽度优先检索

从顶点 v 开始。先作出已访问过的标记。接着下一步访问邻接到 v 的所有顶点。然后再访问邻接这些顶点的顶点。如此等等。如对图 6—10(a)来说。从顶点 v_1 开始作宽度优先检索。则首先访问 v_1 。然后 v_2 。 v_3 。下一次访问 v_4 。 v_5 。 v_6 和 v_7 。最后是 v_8 。详细的情况见下面的算法 BFS 。



(a) (b)

(a) 图 G_6 (b) G_6 的邻接表

图 6－10 G_6 和它的邻接表

```
PROCEDURE BFS(V)
    {图G的宽度优先检索从顶点V开始执行。凡已访问过的顶点
     标记成VISITED(i)=1。图G和VISITED是全程量。
     VISITED初值为0}
  VISITED(v)←1
  initialize Q with vertex v in it  {Q是队列}
  WHILE Q not empty DO
    CALL DELETEQ(V, Q)  {从队列中删去}
    FOR all vertices w adjacent to v DO
      IF VISITED(w)=0  {加W至队列}
        THEN{CALL ADDQ(W, Q);VISITED(W)←1}
                              {对W作标记}
    END
  END
END BFS
```

由于每个被访问的顶点进入队列只有一次。所以WHILE循环至多重复n次。假如用邻接矩阵对于每被访问顶点来说。FOR循环为O(n)次。所以总的次数是$O(n^2)$。如用邻接表时。FOR循环总的次数是$d_1+\cdots+d_n=O(e)$。其中d_i是v_i的度数。所有被访问过的顶点。和关联在这些顶点上的所有边。一道形成G的连通分量。

下面来看图遍历的二个简单应用：求图的连通分量和求连通图的生成树。