

尚德机构

数据结构

主讲：王老师

学习是一种信仰！ IN LEARNING WE TRUST

SUNLAND

 全国高等教育自学考试指定教材

2012年版

计算机及应用专业 独立本科段

数据结构

含：数据结构自学考试大纲

课程代码:02331

组编 / 全国高等教育自学考试指导委员会

主编 / 苏仕华

外语教学与研究出版社

全国高等教育自学考试指定教材 2331数据结构

主 编 苏仕华

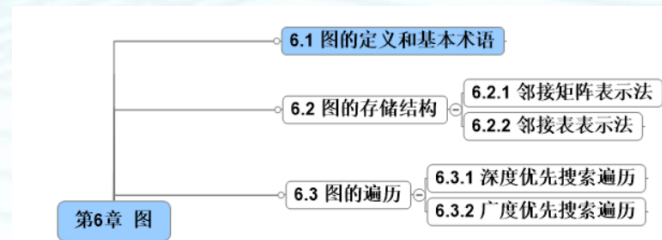
出版社 外语教学与研究出版社

出版时间 2012年3月

第6章

图

- 6.1 图的定义和基本术语
- 6.2 图的存储结构
 - 6.2.1 邻接矩阵表示法
 - 6.2.2 邻接表表示法
- 6.3 图的遍历
 - 6.3.1 深度优先搜索
 - 6.3.2 广度优先搜索
- 6.4 图的生成树和最小生成树
- 6.5 最短路径
- 6.6 拓扑排序



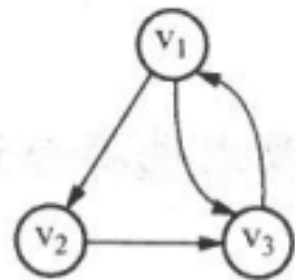
图（Graph）是一种复杂的**非线性结构**。在线性结构中，数据元素之间满足唯一的线性关系，每个数据元素（除第一个和最后一个外）只有一个直接前趋和一个直接后继；在树形结构中，数据元素之间有着明显的层次关系，并且每个元素只与上一层中一个元素（双亲结点）及下一层中多个元素（孩子结点）相关；而在图形结构中，**结点之间的关系可以是任意的，图中任意两个元素之间都可能相关**。因此，图比线性表和树形结构更为复杂。

图形结构简称为图。图 G 由两个集合 V 和 E 组成，定义为 $G=(V, E)$ ，其中 V 是顶点的有限非空集合， E 是由 V 中顶点偶对表示的边的集合。通常， $V(G)$ 和 $E(G)$ 分别表示图 G 的顶点集合和边集合。 $E(G)$ 也可以为空集，即图 G 只有顶点而没有边。

6.1第一节 图的定义和基本术语

第6章 图	6.1 图的定义和基本术语	
	6.2 图的存储结构	6.2.1 邻接矩阵表示法 6.2.2 邻接表表示法
	6.3 图的遍历	6.3.1 深度优先搜索遍历 6.3.2 广度优先搜索遍历

对于一个图 G ，若每条边都是有方向的，则称该图为有向图。在有向图中，一条有向边是由两个顶点组成的有序对，通常用尖括号表示。例如， $\langle v_i, v_j \rangle$ 就表示一条有向边，此边称为顶点 v_i 的一条出边，顶点 v_j 的一条入边；另外，称 v_i 为起始端点（或起点）， v_j 为终止端点（或终点）。因此， $\langle v_i, v_j \rangle$ 和 $\langle v_j, v_i \rangle$ 是两条不同的有向边。有向边又称为弧，边的起点称为弧尾，边的终点称为弧头。例如，图 6.1(a) 中所示的图 G_1 是一个有向图，该图的顶点集和边集分别为

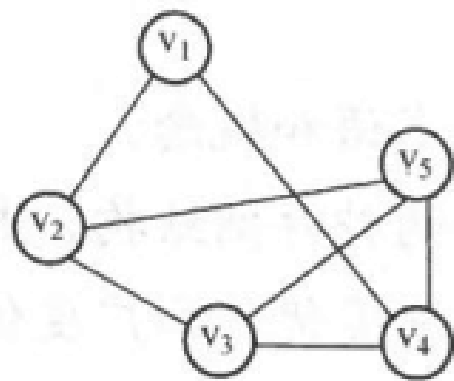


(a) 图 G_1

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

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

对于一个图 G ，若每条边都是没有方向的，则称该图为无向图。在一个无向图中，边均是顶点的无序对，通常用圆括号表示。因此， $\langle v_i, v_j \rangle$ 和 $\langle v_j, v_i \rangle$ 表示同一条边。图 6.1(b) 中所示的 G_2 就是一个无向图，此图的顶点集和边集分别为



(b) 图 G_2

$$V(G_2) = \{v_1, v_2, v_3, v_4, v_5\}$$

$$E(G_2) = \{ (v_1, v_2), (v_1, v_4), (v_2, v_3), (v_2, v_5), (v_3, v_4), (v_3, v_5), (v_4, v_5) \}$$

6.1 图的定义和基本术语



在无向图中, 若存在一条边 (v_i, v_j) , 则称顶点 v_i, v_j 为该边的两个端点, 并称它们互为邻接点, 或称 v_i 和 v_j 相邻接。在有向图中, 若 $\langle v_i, v_j \rangle$ 是一条边, 则称顶点 v_i 邻接到 v_j , 顶点 v_j 邻接于顶点 v_i 。

我们通常用 n 表示图中的顶点数, 用 e 表示图中边或弧的数目, 并且在下面的讨论中不考虑顶点到自身的边, 即若 $\langle v_i, v_j \rangle$ 或 $\langle v_j, v_i \rangle$ 是 $E(G)$ 的一条边, 则要求 $v_i \neq v_j$ 。因此, 对于无向图, e 的取值范围是 $0 \sim \frac{1}{2}n(n-1)$ 。我们将具有 $\frac{1}{2}n(n-1)$ 条边的无向图称为无向完全图。同理, 对于有向图, e 的取值范围是 $0 \sim n(n-1)$, 称具有 $n(n-1)$ 条边或弧的有向图为有向完全图。

图的基本术语

● **顶点(Vertex)**——图中的数据元素;

● **$\langle V_i, V_j \rangle$** ——有向图中,顶点 V_i 到 顶点 V_j 的边,也称**弧**;

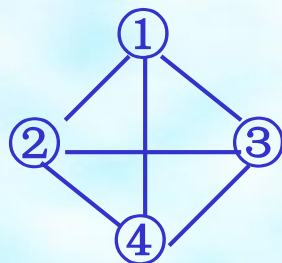
└─ **弧头** (终端点): 箭头端;

└─ **弧尾** (初始点): 无箭头端;

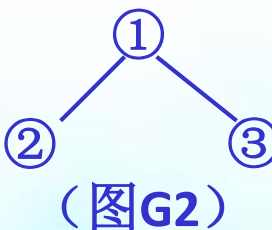
● **完全图**—— $\left\{ \begin{array}{l} \text{无向完全图: 边数} = n*(n-1)/2 \text{ 的无向图;} \\ \text{有向完全图: 边数} = n*(n-1) \text{ 的有向图} \end{array} \right.$
(顶点数 n)

● **权** ——与图中的边相关的数;

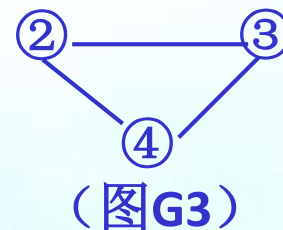
● **子图**——图 G 和 G' ,若有 $V(G') \subseteq V(G)$ 和 $E(G') \subseteq E(G)$,则称 G' 为图 G 的子图。



①
(图G1)



(图G2)



(图G3)

第6章 图	6.1 图的定义和基本术语	
	6.2 图的存储结构	6.2.1 邻接矩阵表示法 6.2.2 邻接表表示法
	6.3 图的遍历	6.3.1 深度优先搜索遍历 6.3.2 广度优先搜索遍历

6.1 图的定义和基本术语

● **邻接**——若 $(V_i, V_j) \in E(G)$ ，则称 V_i 和 V_j 互为邻接点；

● **关联**——若 $(V_i, V_j) \in E(G)$ ，则称边 (V_i, V_j) 关联于顶点 V_i 和 V_j ；

注：1) 邻接是指顶点之间的关系，而关联是指边与顶点间的关系。

2) 若弧 $\langle V_i, V_j \rangle \in E(G)$ ，则称 V_j 是 V_i 的邻接点

● **度**——

- 无向图：顶点 V_i 的度为与 V_i 相关联的边的个数；
 $D(V_i)$
- 有向图
 - 出度：顶点 V_i 的出度为以 V_i 为尾的出边数；
 $OD(V_i)$
 - 入度：顶点 V_i 的入度为以 V_i 为头的入边数；
 $ID(V_i)$
 - 度：有向图的度 = 入度 + 出度；
 $D(V_i) = OD(V_i) + ID(V_i)$

第6章 图

6.1 图的定义和基本术语

6.2 图的存储结构

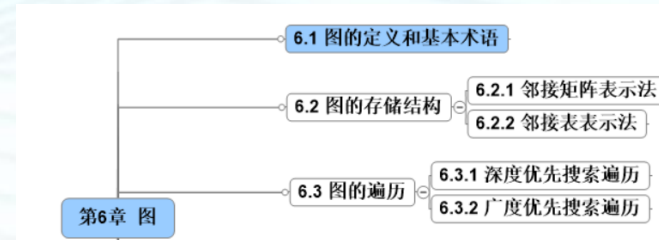
6.2.1 邻接矩阵表示法

6.2.2 邻接表表示法

6.3 图的遍历

6.3.1 深度优先搜索遍历

6.3.2 广度优先搜索遍历



注：图中边数 e 与顶点的度的关系

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

（一边带二度，两度组成一边）

● **路径**——图中，顶点 V_p 至顶点 V_q 的路径是顶点序列 $\{ V_p, V_{i_1}, V_{i_2}, \dots, V_{i_n}, V_q \}$ 且
对无向图，边 $(V_p, V_{i_1}), (V_{i_1}, V_{i_2}), \dots, (V_{i_n}, V_q) \in VR(G)$;
对有向图，弧 $\langle V_p, V_{i_1} \rangle, \langle V_{i_1}, V_{i_2} \rangle, \dots, \langle V_{i_n}, V_q \rangle \in VR(G)$;

● **路径长度**——路径上边或弧的数目；

● **简单路径**——除第一个和最后一个外，其余各顶点均不相同的路径；

第6章 图	6.1 图的定义和基本术语	
	6.2 图的存储结构	6.2.1 邻接矩阵表示法 6.2.2 邻接表表示法
	6.3 图的遍历	6.3.1 深度优先搜索遍历 6.3.2 广度优先搜索遍历

● **回路**—第一个和最后一个顶点相同的路径，也称环；

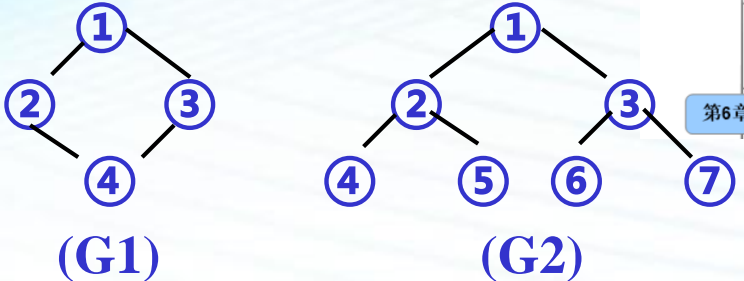
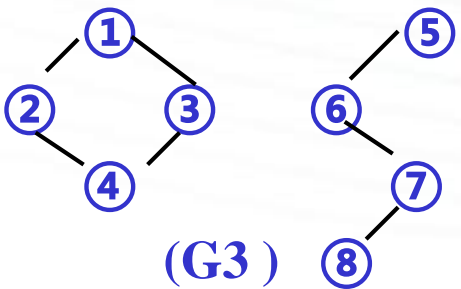
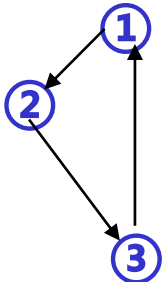
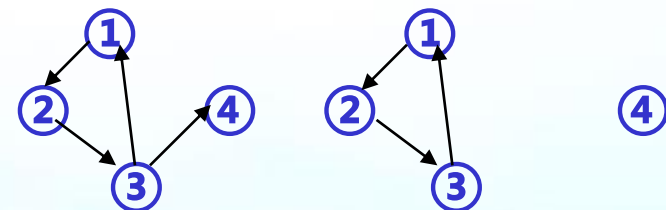
● **简单回路**—第一个和最后一个顶点相同的简单路径；

注：回路中可以有多圈，而简单回路只能有一个圈。

● **连通**——无向图中，若从顶点 V_i 到 V_j 顶点有路径，则称 V_i 和 V_j 是连通的。

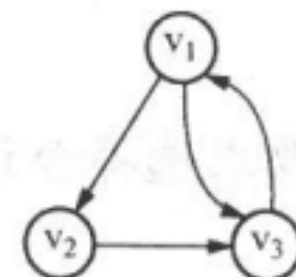
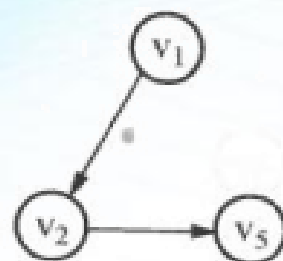
● **连通图和连通分量**

↗
针对无向图而言

		定 义	例
无向图	连通图	图中每对顶点间都连通; $V_i \sim V_j$	 (G1) (G2)
	连通分量	图中极大的连通子图 (再扩大一点就不连通)	 (G3) (G3不是连通图, 但它有两个连通分量)
有向图	强连通图	图中任意一对顶点 V_i 和 V_j 都有顶点 V_i 到顶点 V_j 的路径, 也有从 v_j 到 v_i 的路径, 两个顶点间双向连通。	
	强连通分量	有向图的极大强连通子图。	 (G4) 图G4的两个强连通分量

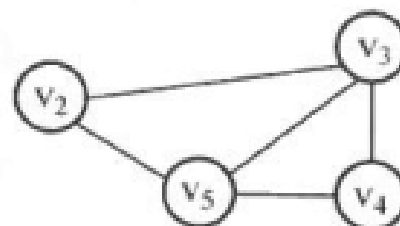
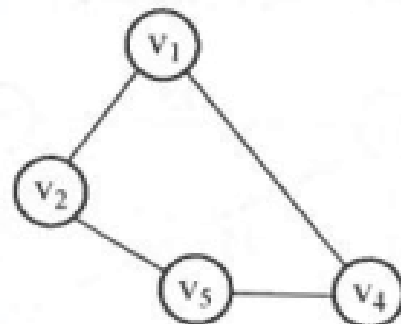
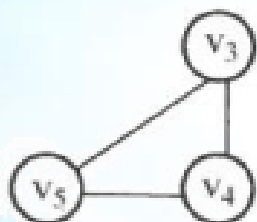
6.1 图的定义和基本术语

第6章 图	6.1 图的定义和基本术语	
	6.2 图的存储结构	6.2.1 邻接矩阵表示法 6.2.2 邻接表表示法
	6.3 图的遍历	6.3.1 深度优先搜索遍历 6.3.2 广度优先搜索遍历

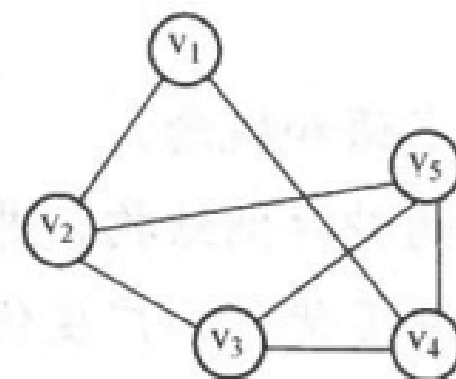


(a) 图 G_1 的子图

(a) 图 G_1



(b) 图 G_2 的子图



(b) 图 G_2

图 6.2 图 6.1 的子图

6.1 图的定义和基本术语

第6章 图

6.1 图的定义和基本术语

6.2 图的存储结构

6.2.1 邻接矩阵表示法

6.2.2 邻接表表示法

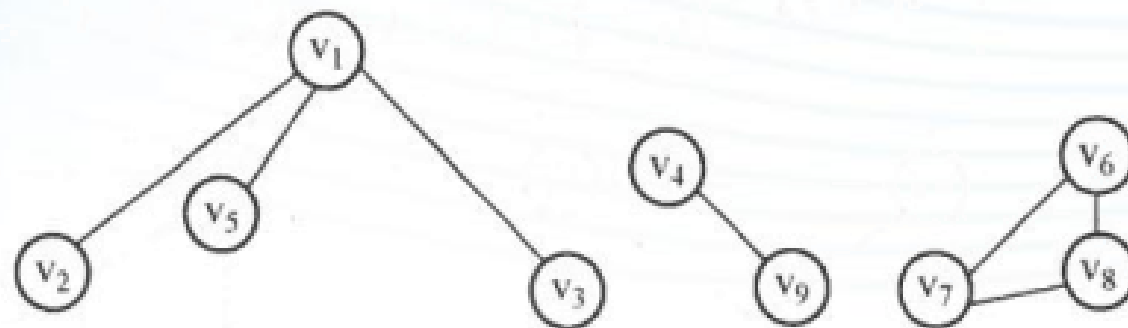
6.3 图的遍历

6.3.1 深度优先搜索遍历

6.3.2 广度优先搜索遍历



(a) 非连通图G3



(b) 图 (a) 的三个连通分量

图 6.3 非连通图及图的连通分量

6.1 图的定义和基本术语

第6章 图	6.1 图的定义和基本术语	6.2.1 邻接矩阵表示法
	6.2 图的存储结构	6.2.2 邻接表表示法
	6.3 图的遍历	6.3.1 深度优先搜索遍历
		6.3.2 广度优先搜索遍历

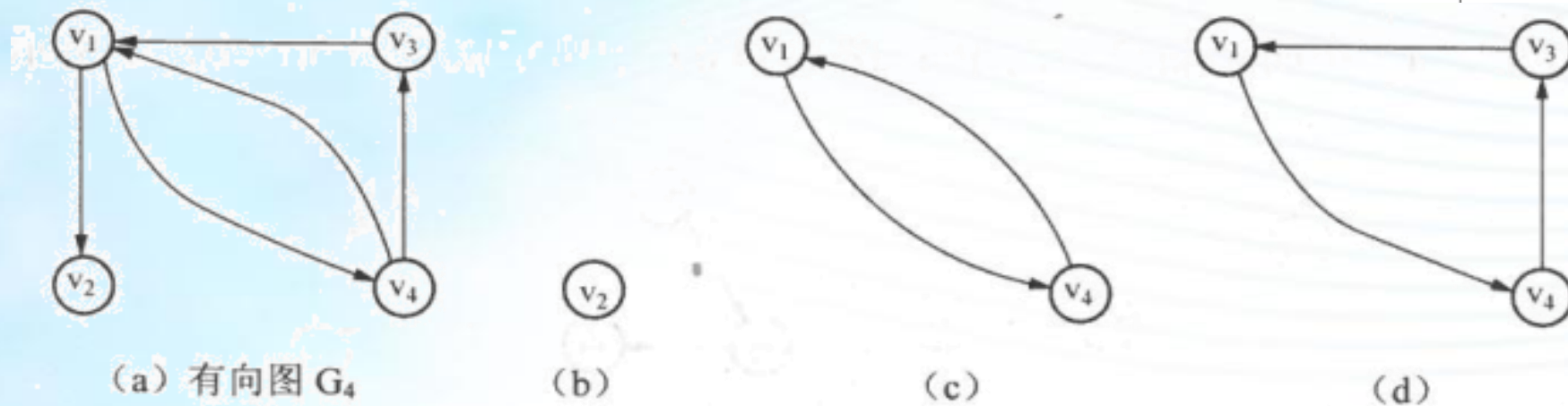


图 6.4 有向图及其强连通分量示意图

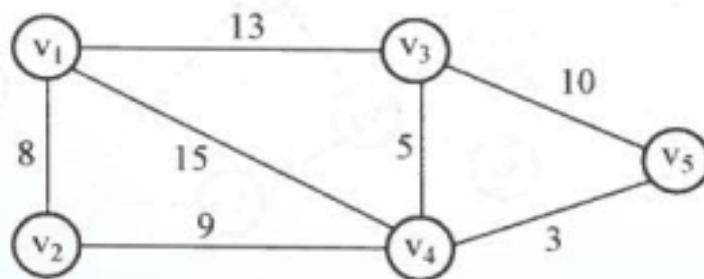


图 6.5 带权图示例 G_5

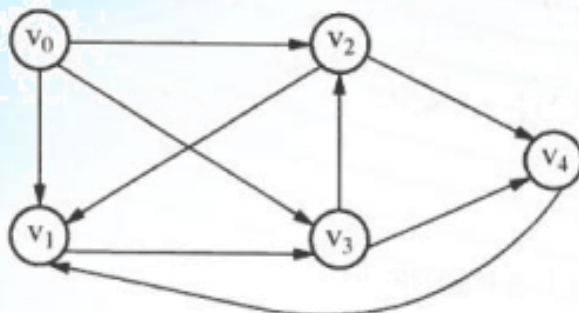
图的存储结构又称图的存储表示。图的存储表示方法很多，这里主要介绍两种最常用的方法，即**邻接矩阵和邻接表表示法**。为了适应C语言的描述，从本节起，假定图的顶点序号从0开始，即图G的顶点集 $V(G)=\{v_0, v_1, \dots, v_{n-1}\}$ 。

6.2.1 邻接矩阵表示法

$$A[i][j] = \begin{cases} 1 & \text{若 } (v_i, v_j) \text{ 或 } \langle v_i, v_j \rangle \text{ 是 } E(G) \text{ 的边} \\ 0 & \text{若 } (v_i, v_j) \text{ 或 } \langle v_i, v_j \rangle \text{ 不是 } E(G) \text{ 的边} \end{cases}$$



(a) 无向图 G_6

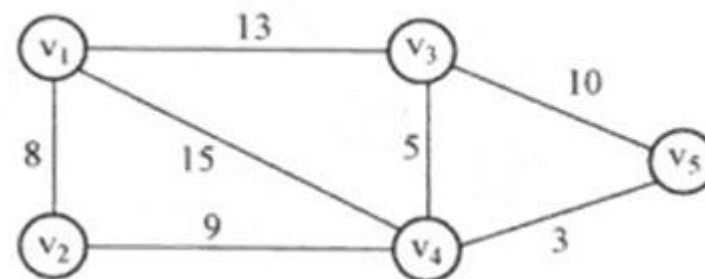


(b) 有向图 G_7

图 6.6 无向图和有向图

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

图 6.7 图的邻接矩阵

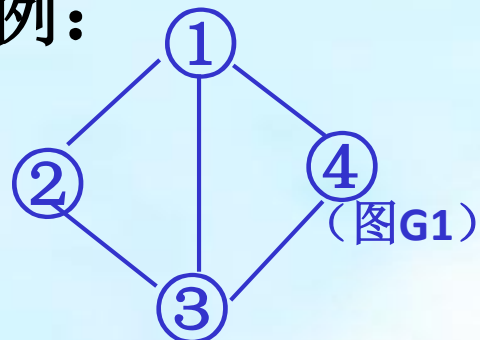


$$A_3 = \begin{bmatrix} \infty & 8 & 13 & 15 & \infty \\ 8 & \infty & \infty & 9 & \infty \\ 13 & \infty & \infty & 5 & 10 \\ 15 & 9 & 5 & \infty & 3 \\ \infty & \infty & 10 & 3 & \infty \end{bmatrix}$$

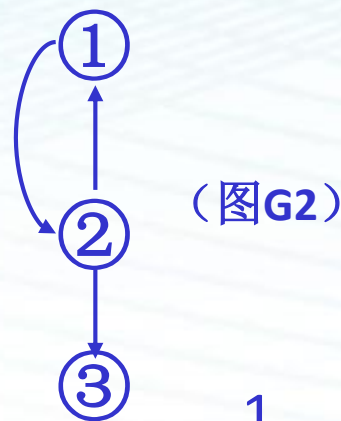
图 6.8 带权图的邻接矩阵

6.2.1 邻接矩阵表示法

▲例:



$$G1 = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix}$$



$$G2 = \begin{matrix} & \begin{matrix} 1 & 2 & 3 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \end{matrix} & \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \end{matrix}$$

6.2 图的存储结构

6.2.1 邻接矩阵表示法

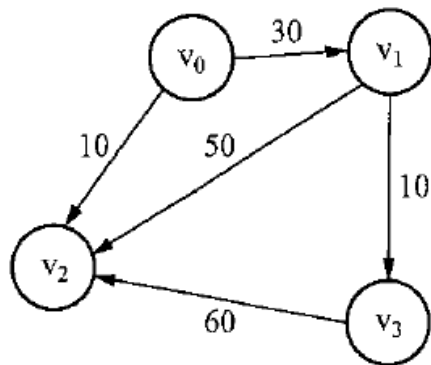
6.2.2 邻接表表示法

▲结论:

- (1) 无向图的邻接矩阵是**对称**的;
($\because (V_i, V_j) \in E(G), \text{ 则 } (V_j, V_i) \in E(G)$)
- (2) 从邻接矩阵容易判断任意两顶点间是否有边相联;
容易求出各顶点的度;
无向图: 顶点 V_i 的度 $D(V_i)$ =矩阵中第 i 行或第 i 列元素之和
有向图: $OD(V_i)$ =矩阵中第 i 行元素之和
 $ID(V_i)$ =矩阵中第 i 列元素之和

带权图(网)的邻接矩阵

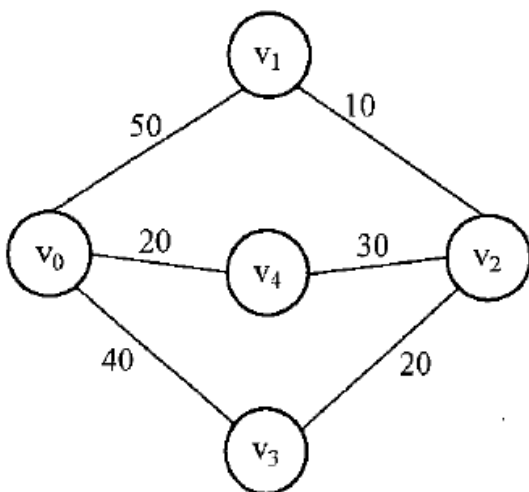
$$A[i][j] = \begin{cases} w_{ij} & \text{若 } (v_i, v_j) \text{ 或 } \langle v_i, v_j \rangle \in E(G) \text{ (} w_{ij} \text{ 为边或弧的权)} \\ \infty & v_i, v_j \text{ 间无边或弧} \end{cases}$$



a)

$$\begin{pmatrix} \infty & 30 & 10 & \infty \\ \infty & \infty & 50 & 10 \\ \infty & \infty & \infty & \infty \\ \infty & \infty & 60 & \infty \end{pmatrix}$$

b)



$$\begin{pmatrix} \infty & 50 & \infty & 40 & 20 \\ 50 & \infty & 10 & \infty & \infty \\ \infty & 10 & \infty & 20 & 30 \\ 40 & \infty & 20 & \infty & \infty \\ 20 & \infty & 30 & \infty & \infty \end{pmatrix}$$

图的邻接矩阵表示，除了需要用二维数组存储顶点之间相邻关系的邻接矩阵外，通常还需要使用一个具有 n 个元素的一维数组来存储顶点信息，其中下标为 i 的元素存储顶点 v_i 的信息。

因此，图的邻接矩阵表示的存储结构定义如下：

```
#define MaxVertexNum 50                                //最大顶点数
typedef struct {
    VertexType vexs[MaxVertexNum];                      //顶点数组，类型假定为char型
    Adjmatrix arcs[MaxVertexNum][MaxVertexNum];         //邻接矩阵，假定为int型
} MGraph;
```

6.2.1 邻接矩阵表示法

由于无向图的邻接矩阵是对称的，可采用压缩存储仅存储主对角线以下的元素。建立一个无向图的算法如下：

```
void CreateMGraph(MGraph * G, int n, int e)
{ //采用邻接矩阵表示法构造无向图G, n、e表示图的当前顶点数和边数
    int i, j, k, w;
    scanf("%d, %d", &n, &e); //读入顶点数和边数
    for(i=0; i<n; i++) //输入顶点信息
        scanf("%c", & G->vexs[i]);
    for(i=0; i<n; i++)
        for(j=0; j<n; j++)
            G->arcs[i][j]=INT_MAX; //初始化邻接矩阵元素为无穷大
    for(k=0; k<e; k++) { //读入e条边，建立邻接矩阵
        scanf("%d, %d, %d", &i, &j, &w); //读入一条边的两端顶点序号i、j及边上的权w
        G->arcs[i][j]=w;
        G->arcs[j][i]=w; //置矩阵对称元素权值
    }
}
```

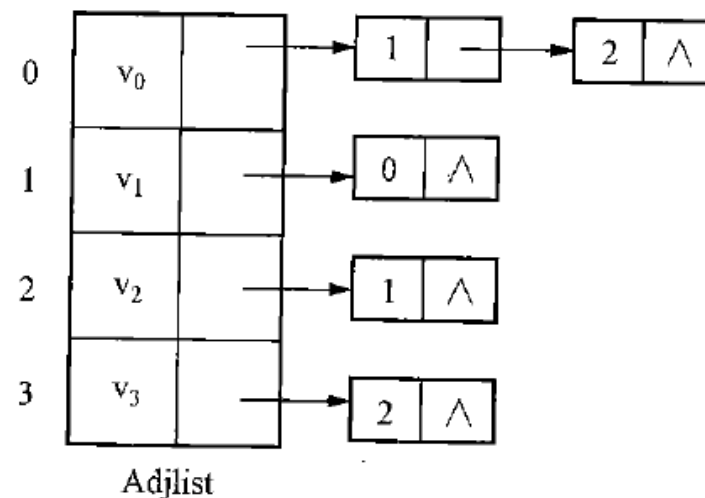
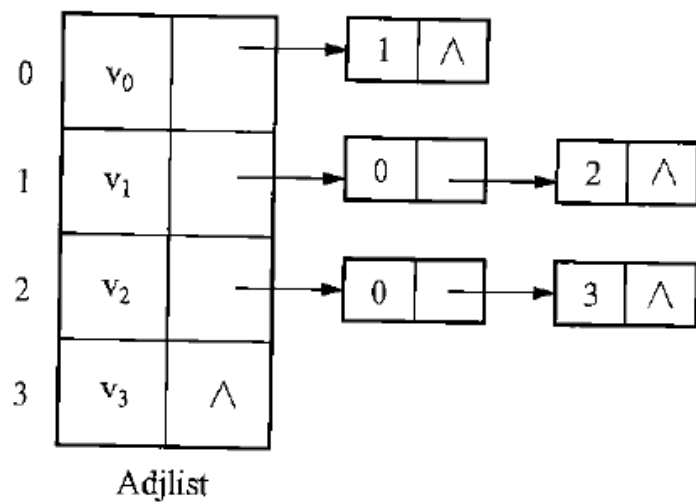
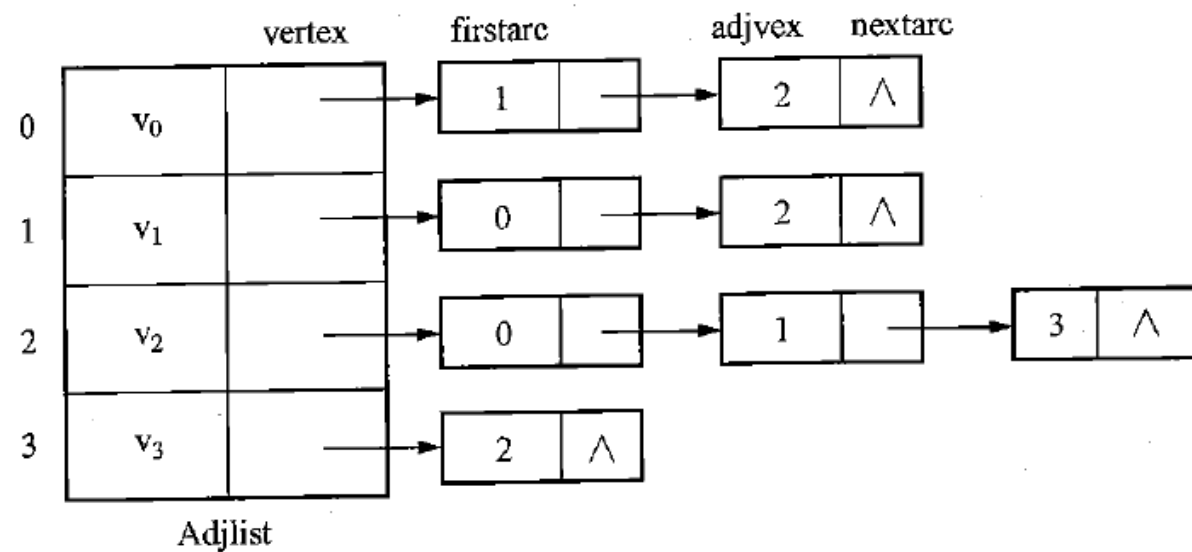
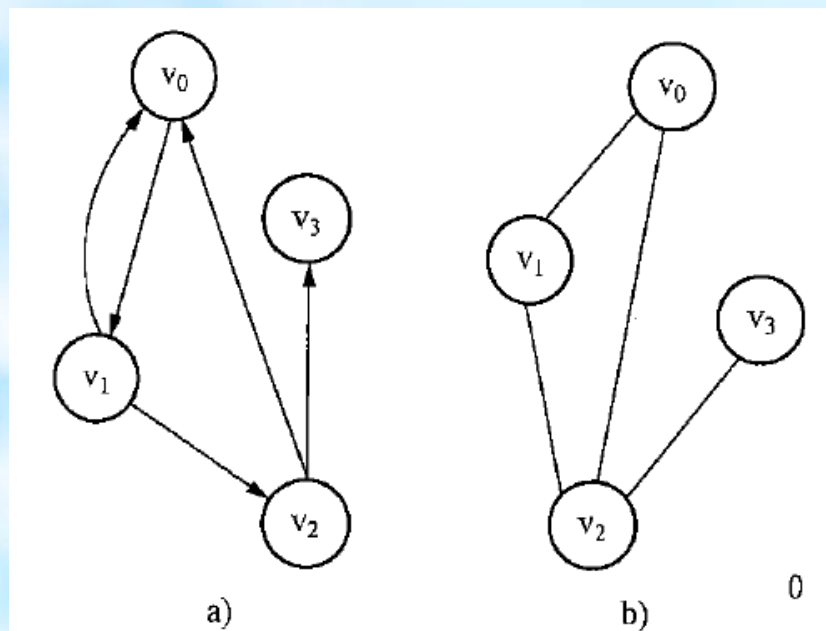
上述算法的执行时间是 $O(n^2+e+n)$ ，其中 $O(n^2)$ 是初始化邻接矩阵所耗费的时间。因此，该算法的时间复杂度应为 $O(n^2)$ 。

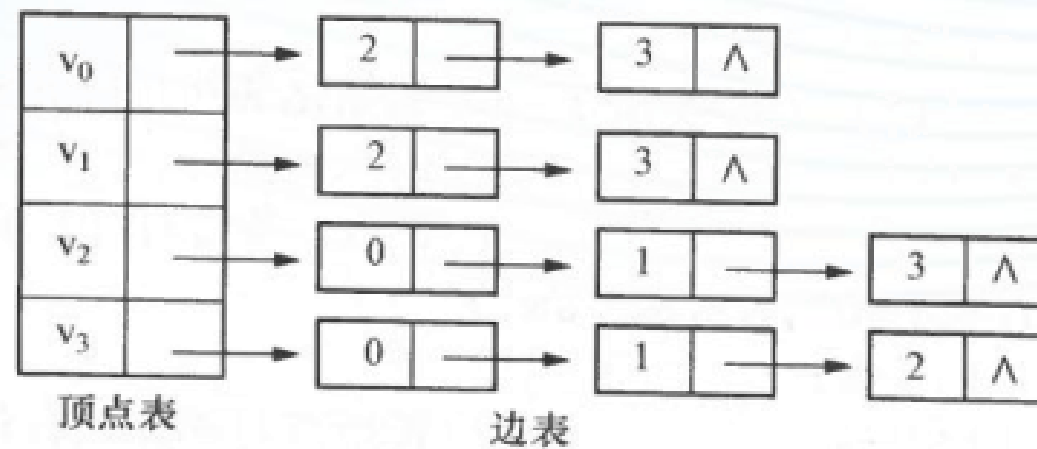
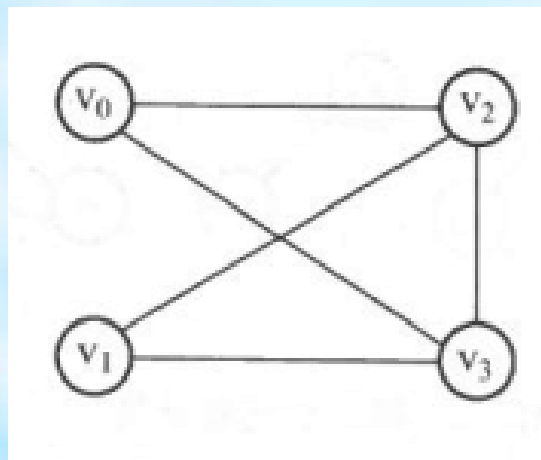
邻接表是图的一种链式存储结构。这种存储表示法类似于树的孩子链表表示法。

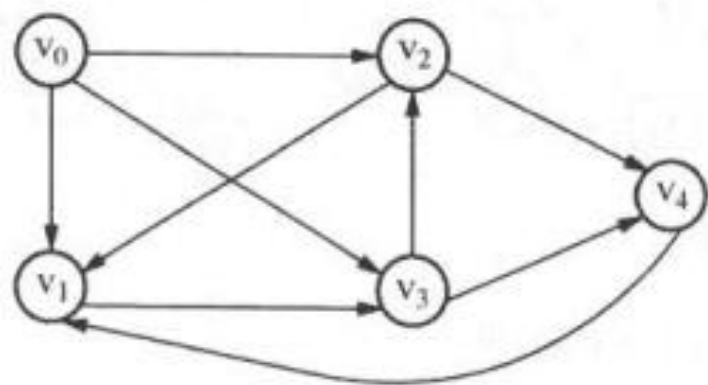
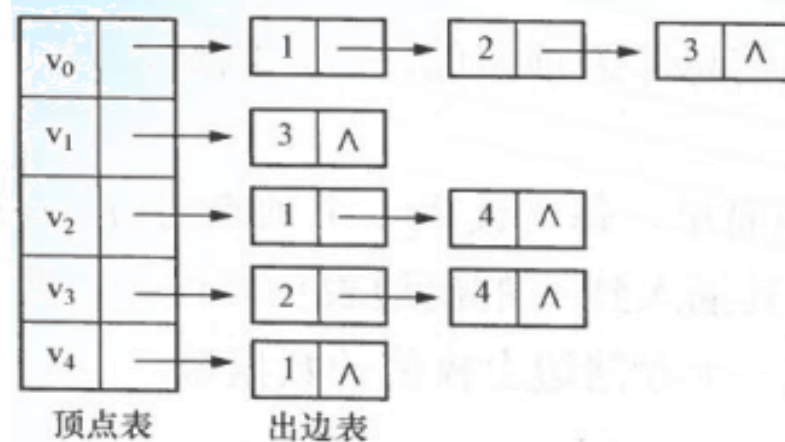
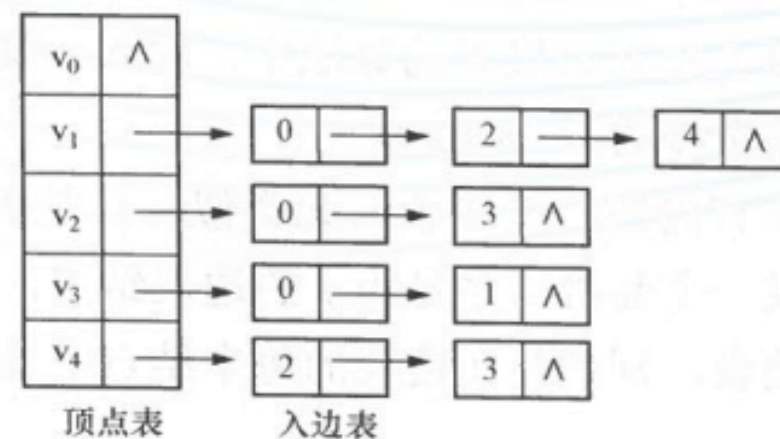
图的邻接表存储结构定义如下：

```
#define MaxVertexNum 20
typedef char VertexType;
typedef struct node {           //边表结点类型
    int adjvex;                //顶点的序号
    struct node * next;        //指向下一条边的指针
} EdgeNode;
typedef struct vnode {          //顶点表结点
    VertexType vertex;         //顶点域
    EdgeNode * link;           //边表头指针
} VNode, Adjlist[MaxVertexNum]; //邻接表
typedef Adjlist ALGraph;       //定义为图类型
```

例：G1、G2的邻接表。



图 6.9 无向图 G_6 的邻接表

(b) 有向图 G_7 (a) 图 G_7 的邻接表(b) 图 G_7 的逆邻接表图 6.10 图 G_7 的邻接表和逆邻接表

6.2.2 邻接表表示法

6.2 图的存储结构

6.2.1 邻接矩阵表示法

6.2.2 邻接表表示法

下面给出一个无向图邻接表的建表算法：

```
void CreateGraph(ALGraph GL, int n, int e)
{ //n为顶点数, e为图的边数
  int i, j, k; EdgeNode *p;
  for(i=0; i<n; i++) {
    GL[i].vertex=getchar( );
    GL[i].link=NULL;
  }
  for(k=0; k<e; k++) {
    scanf("%d, %d", &i, &j);
    p=(EdgeNode * ) malloc (sizeof (EdgeNode) ); //生成新的边表结点
    p->adjvex=j;
    p->next=GL[i].link;
    GL[i].link=p;
    p=(EdgeNode * ) malloc (sizeof(EdgeNode) ); //生成新的边表结点
    p->adjvex=i;
    p->next=GL[j].link;
    GL[j].link=p;
  }
}
```

建立有向图的邻接表与此类似，只是更加简单，每当读入一个顶点对 $\langle i, j \rangle$ 时，仅需要生成一个邻接点序号为 j 的边表结点，将其插入到 v_i 的出边表头即可。若要建立网络的邻接表，则需要在边表的每个结点中增加一个存储边上权值的数据域。

在以上建立邻接表的算法中，输入的顶点信息即为顶点的序号，因此建立邻接表的时间复杂度为 $O(n+e)$ 。

结论:

- 1) n 个顶点、 e 条边的无向图, 则其邻接表的表头结点数为 n ,
链表结点总数为 $2e$;
- 2) 对于无向图, 第 i 个链表的结点数为顶点 V_i 的度;
对于有向图, 第 i 个链表的结点数为顶点 V_i 的出度;
- 3) 在边稀疏时, 邻接表比邻接矩阵省单元;
- 4) 邻接表表示在检测边数方面比邻接矩阵表示效率要高。

6.3 图的遍历

▲遍历的含义及方法:

● **图的遍历**——从图G中某一顶点v出发，顺序访问各顶点一次。

● 方法:

为克服顶点的重复访问，设立辅助数组visited[n]。

visited[i]= $\begin{cases} 1 & \text{顶点i已被访问过} \\ 0 & \text{顶点i未被访问过} \end{cases}$

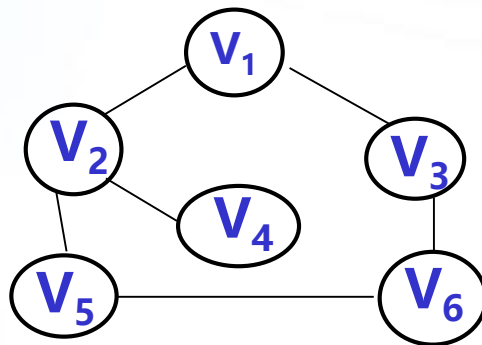
遍历方法 $\begin{cases} \text{深度优先搜索遍历} \\ \text{广度优先搜索遍历} \end{cases}$

6.3.1 连通图的深度优先搜索 (DFS)

一、过程

从图 $G(V,E)$ 中任一顶点 V_i 开始, 首先访问 V_i , 然后访问 V_i 的任一未访问过的邻接点 V_j , 再以 V_j 为新的出发点继续进行深度优先搜索, 直到所有顶点都被访问过。**类似于树的前序（先根）遍历。**

二、例:



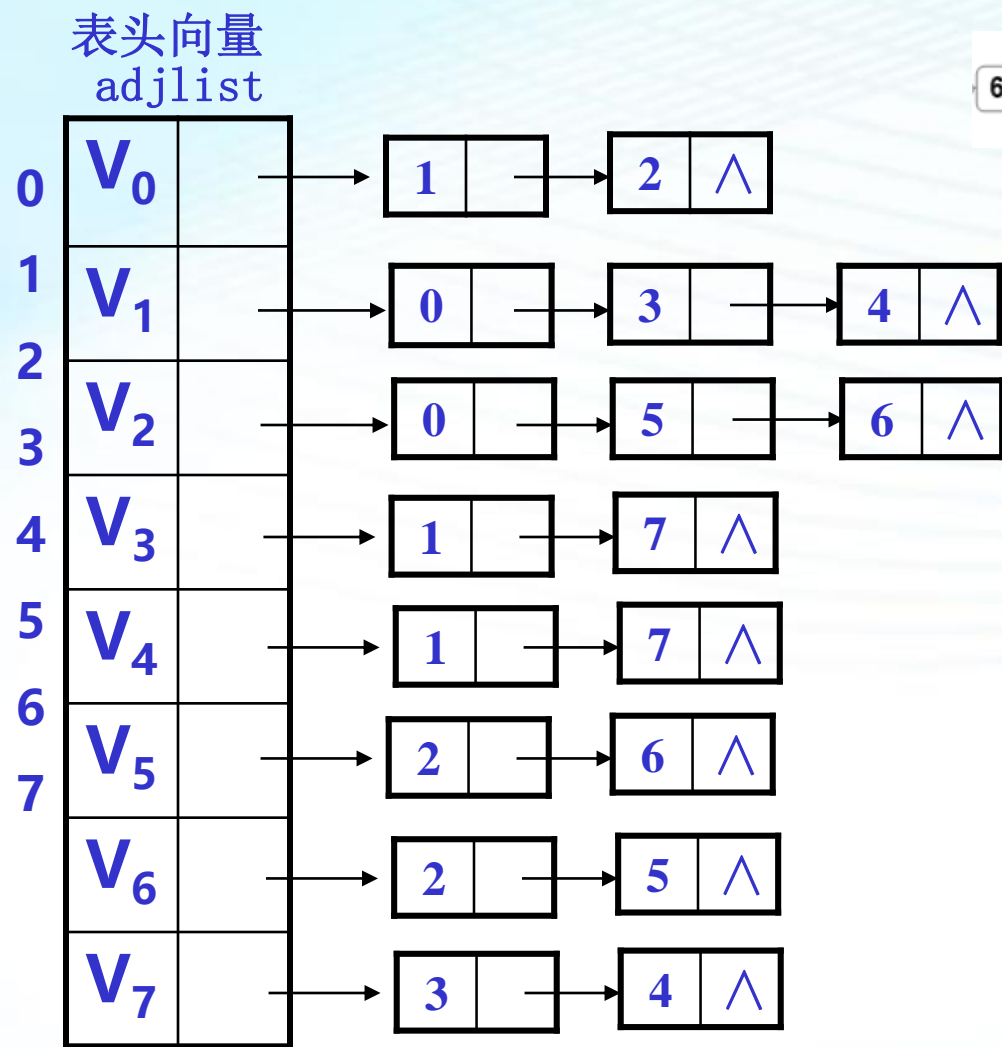
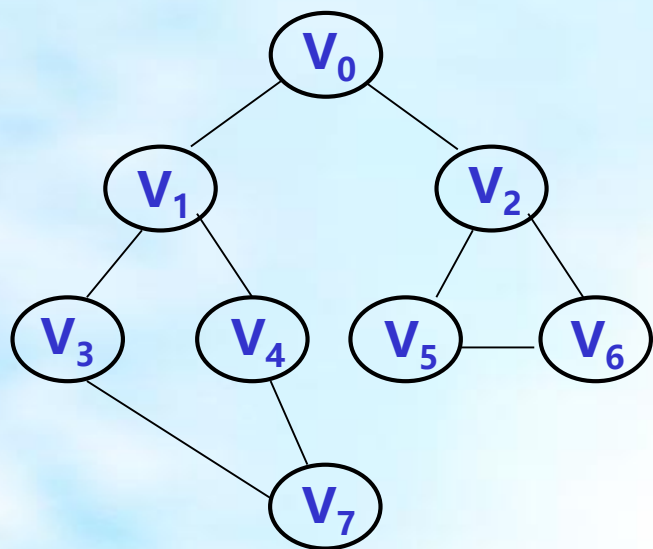
从 V_1 出发, DFS:
 $V_1, V_2, V_4, V_5, V_3, V_6$

三、算法:

▲分析:

- a、为克服顶点的重复访问, 设立一标志向量visited [n];
- b、图可用邻接矩阵或邻接表表示;
- c、**DFS**规则具有递归性, 故需用到栈。

6.3.1 深度优先搜索遍历



6.3 图的遍历

6.3.1 深度优先搜索遍历

6.3.2 广度优先搜索遍历

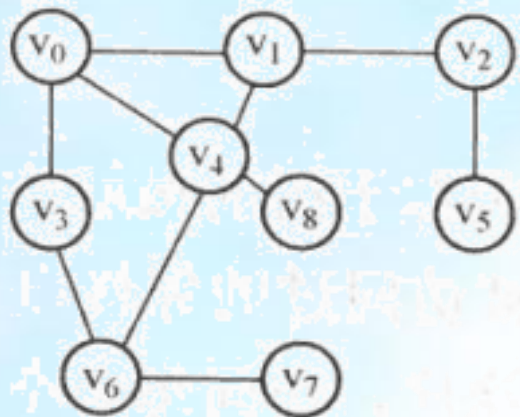
从 V_0 出发,深度优先搜索: $V_0, V_1, V_3, V_7, V_4, V_2, V_5, V_6$ 。

6.3.1 深度优先搜索遍历

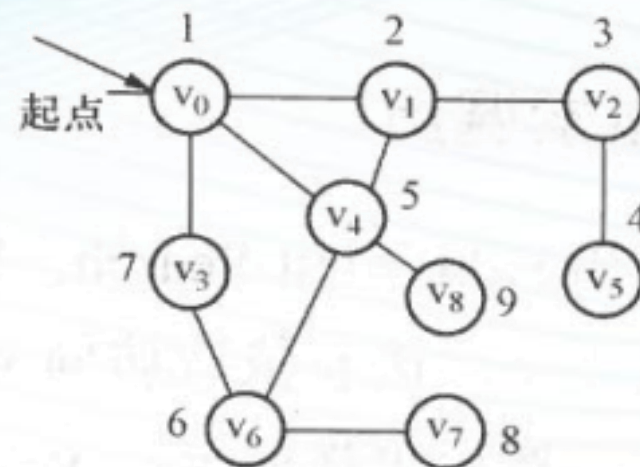
6.3 图的遍历

6.3.1 深度优先搜索遍历

6.3.2 广度优先搜索遍历



(a) 无向图 G_8



(b) G_8 的深度优先搜索遍历过程示意

图 6.11 无向图及其深度优先搜索示意图

深度优先搜索遍历图 G_8 的顶点访问序列： $v_0, v_1, v_2, v_5, v_4, v_6, v_3, v_7, v_8$ 。

(1) 以邻接矩阵为存储结构的深度优先搜索遍历算法

$O(n^2)$

```
int visited[20];  
void DFS(MGraph G, int i, int n)  
{    //从顶点vi出发, 深度优先搜索遍历图G(邻接矩阵结构)  
    int j;  
    printf("v%d→", i);           //假定访问顶点vi以输出该顶点的序号代之  
    visited[i]=1;                 //标记vi已访问过  
    for(j=0; j<n; j++)           //依次搜索vi的每个邻接点  
        if(G.arcs[i][j]==1 && !visited[j])  
            DFS(G, j, n);         //若(vi,vj)∈E(G),且vj未被访问过, 则从开始递归调用  
}
```

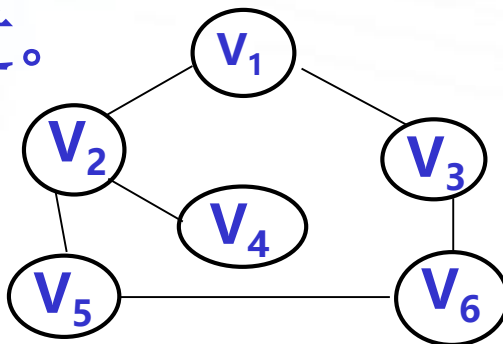
(2) 以邻接表为存储结构的深度优先搜索遍历算法 $O(n+e)$

```
int visited[20];
void DFS1(ALGraph G, int i)
{ //从顶点vi出发，深度优先搜索遍历图G(邻接表结构)
    EdgeNode * p; int j;
    printf("v%d→", i);           //假定访问顶点vi以输出该顶点的序号代之
    visited[i]=1;                 //标记vi已访问过
    p=G[i].link;                  //取vi邻接表的表头指针
    while(p!=NULL) {              //依次搜索vi的每个邻接点
        j=p->adjvex;              //j为vi的一个邻接点序号
        if(!visited[j])
            DFS1(G, j);           //若(vi,vj)∈E(G),且vj未被访问过，则从开始递归调用
        p=p->next;               //使P指向vi的下一个邻接点
    }                             //End_while
}
```

6.3.2 广度优先搜索法 (BFS)

一、过程

从图 $G(V,E)$ 中某一点 V_i 出发, 首先访问 V_i 的所有邻接点 ($V_{i1}, V_{i2}, \dots, V_{it}$), 然后再顺序访问 $V_{i1}, V_{i2}, \dots, V_{it}$ 的所有未被访问过的邻接点..., 此过程直到所有顶点都被访问过。



从 V_1 出发, BFS:

$V_1, V_2, V_3, V_4, V_5, V_6$

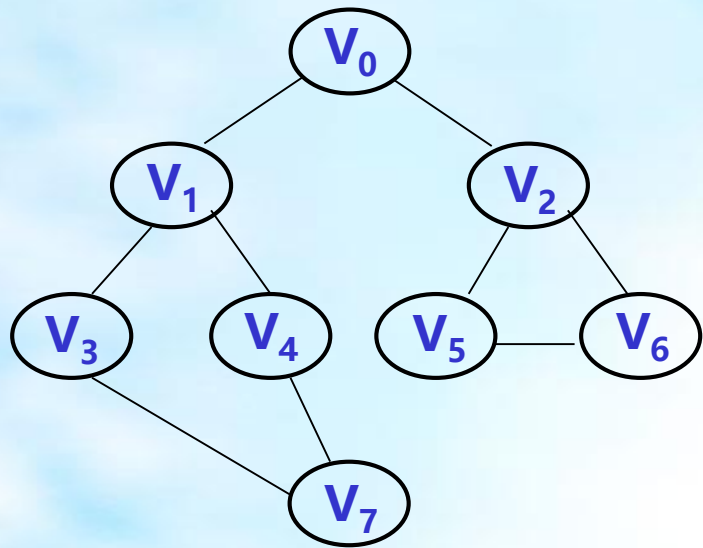
二、例:

三、算法:

▲分析:

- a、为克服顶点的重复访问, 设立一标志向量 **visited [n]**;
- b、图可用邻接矩阵或邻接表表示;
- c、顶点的处理次序——先进先出, 故需用到-----队列

6.3.2 广度优先搜索遍历



表头向量
adjlist

0	V₀	→	1	→	2	∧		
1	V₁	→	0	→	3	→	4	∧
2	V₂	→	0	→	5	→	6	∧
3	V₃	→	1	→	7	∧		
4	V₄	→	1	→	7	∧		
5	V₅	→	2	→	6	∧		
6	V₆	→	2	→	5	∧		
7	V₇	→	3	→	4	∧		

6.3 图的遍历

6.3.1 深度优先搜索遍历

6.3.2 广度优先搜索遍历

从**V₀**出发，广度优先搜索：**V₀**、 **V₁**、 **V₂**、 **V₃**、 **V₄**、 **V₅**、 **V₆**、 **V₇**。

6.3.2 广度优先搜索遍历

6.3 图的遍历

6.3.1 深度优先搜索遍历

6.3.2 广度优先搜索遍历

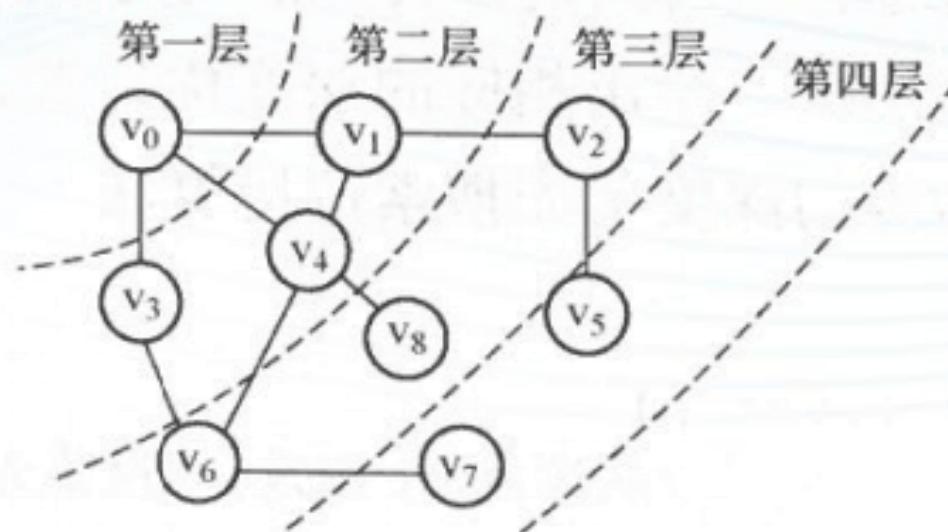
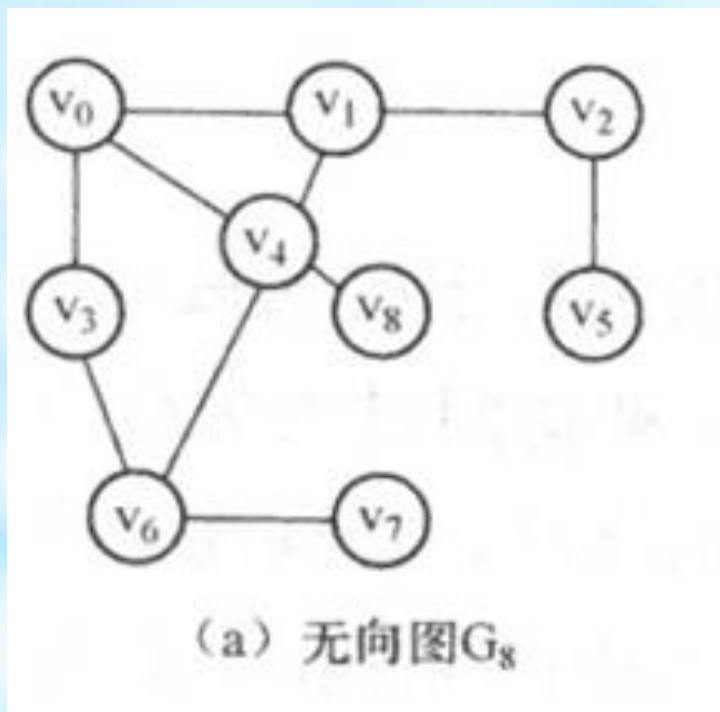
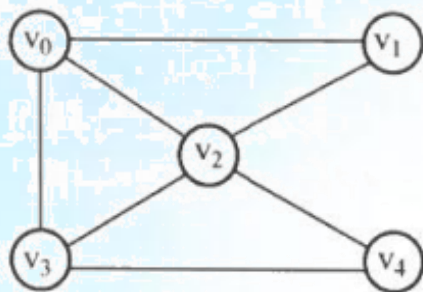


图 6.12 广度优先搜索遍历示意图

以 v_0 为出发点的广度优先搜索遍历序列： $v_0, v_1, v_3, v_4, v_2, v_6, v_8, v_5, v_7$ 。

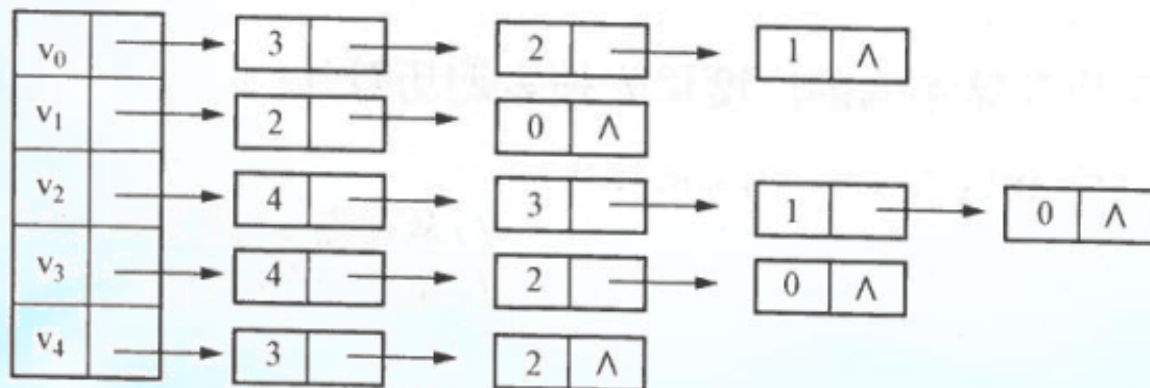
【例6.1】假设有如图6.13所示的图 G_9 ，试写出该图的邻接矩阵和邻接表以及该图在邻接矩阵存储表示下从顶点 v_3 开始搜索所得的深度优先（DFS）和广度优先（BFS）遍历序列。

(a) 图 G_9

$$A = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix}$$

(b)

图 6.13 一个无向图及其邻接矩阵

图 6.14 图 G_9 的邻接表

6.3.2 广度优先搜索遍历

6.3 图的遍历

6.3.1 深度优先搜索遍历

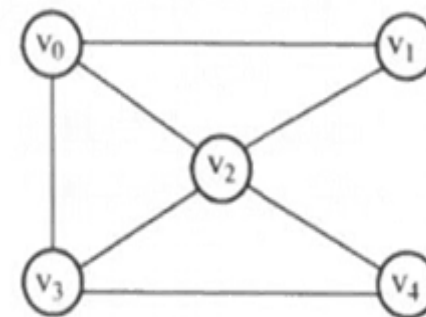
6.3.2 广度优先搜索遍历

(3) 图的深度优先搜索和广度优先搜索遍历序列分两种情况讨论。

一种是在邻接矩阵表示的顺序存储结构上，图 G_9 的两种遍历序列如下：

DFS序列为： v_3, v_0, v_1, v_2, v_4

BFS序列为： v_3, v_0, v_2, v_4, v_1



另一种是在以邻接表表示的链式存储结构上，如图6.14所示的邻接表，从顶点 v_3 出发的DFS和BFS遍历序列如下：

DFS序列为： v_3, v_4, v_2, v_1, v_0

BFS序列为： v_3, v_4, v_2, v_0, v_1

● **生成树**——含有该连通图的全部顶点的一个**极小连通子图**。

若连通图 G 的顶点个数为 n ，则 G 的生成树的边数为 **$n-1$** 。

{ G 的子图 G' 边数大于 $n-1$,则 G' 中一定**有环**。
 G 的子图 G' 边数小于 $n-1$,则 G' 中一定**不连通**。

一棵具有 n 个顶点的生成树有仅有 $n-1$ 条边，但有 $n-1$ 条边的图不一定是生成树。同一个图可以有不同生成树，例如对于图6.15(a)而言，图6.15(b)、图6.15(c)所示都是它的生成树。

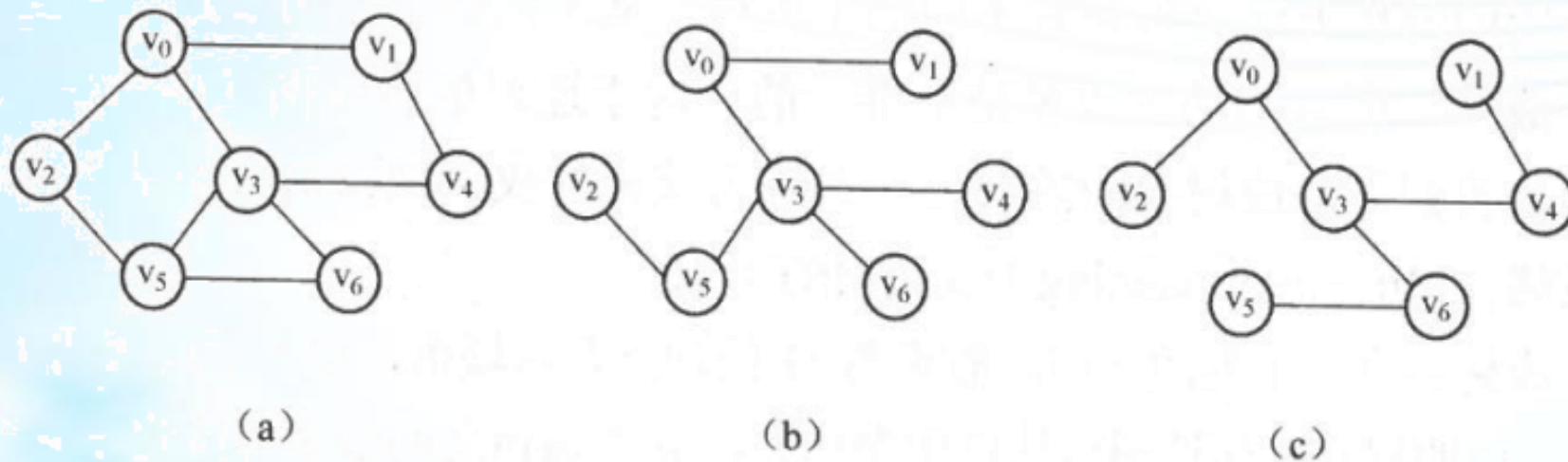


图 6.15 生成树示例

通常，我们把由深度优先搜索所得的生成树称之为深度优先生成树，简称为DFS生成树；而由广度优先搜索所得的生成树称之为广度优先生成树，简称为BFS生成树。例如，从图 G_8 的顶点 v_0 出发，所得的DFS生成树和BFS生成树如图6.16所示。

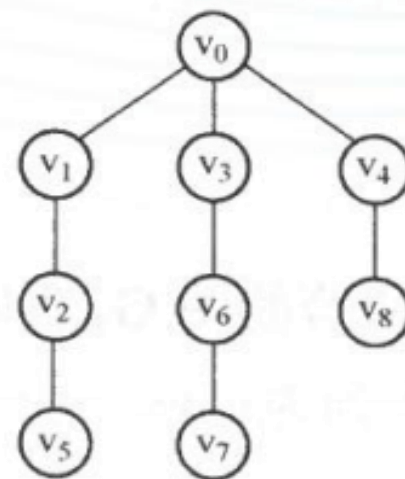
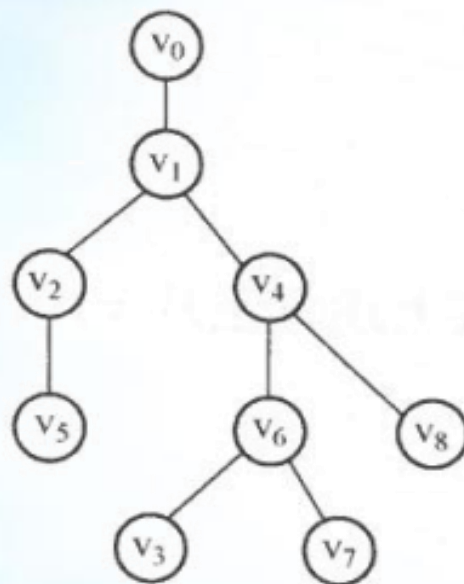
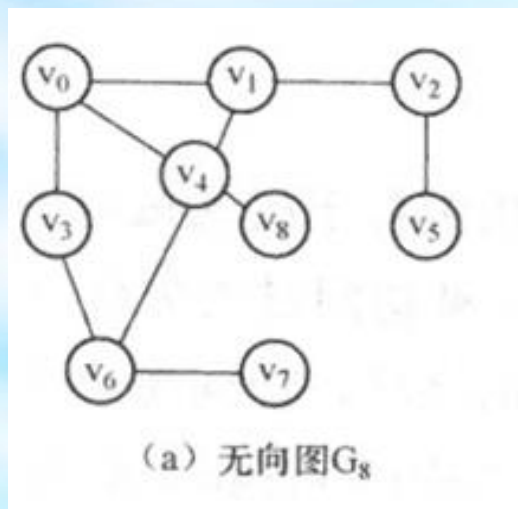
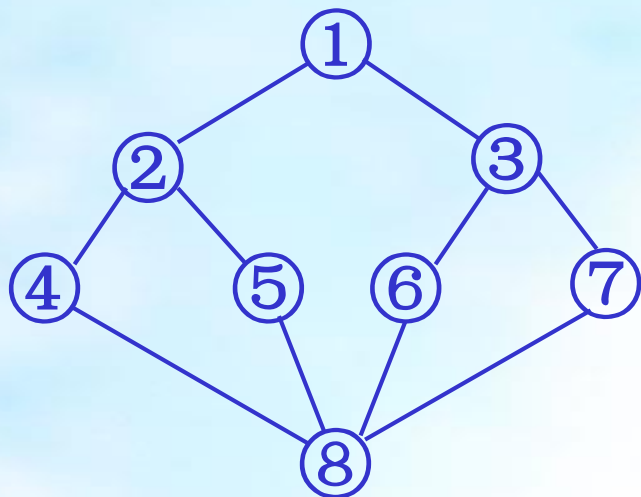
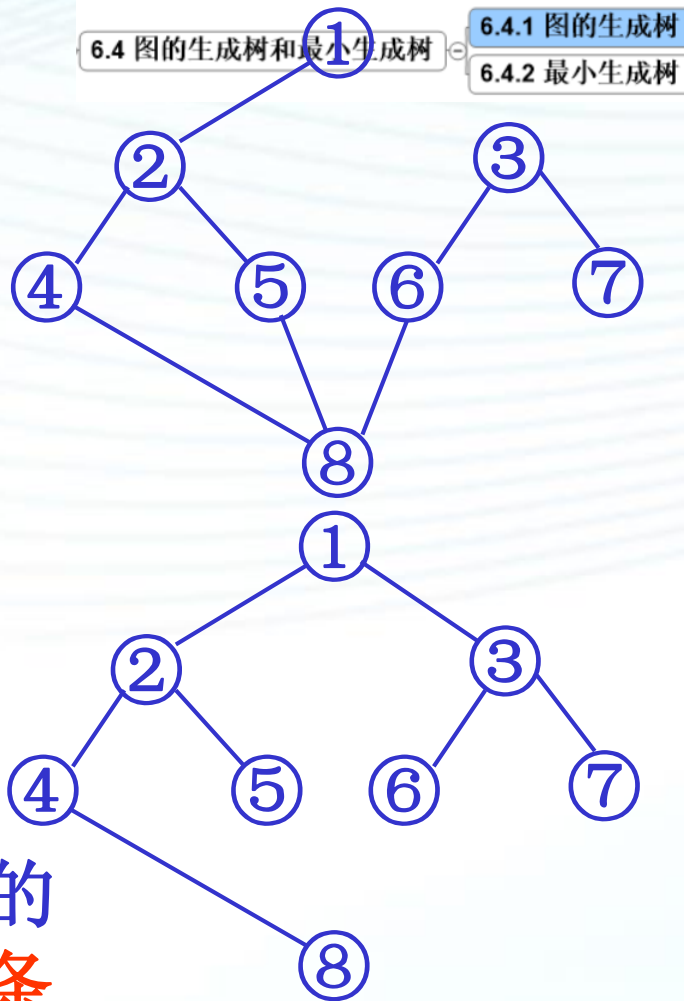


图 6.16 图 G_8 的 DFS 和 BFS 生成树



其深度优先生成树为:



注: ●生成树 **G'** 是图 **G** 的**极小**连通子图。
即 **$V(G)=V(G')$** , **G'** 是连通的, 且在 **G** 的所有连通子图中**边数最少** (**n** 个顶点, **$n-1$** 条边)。

●图的生成树**不是唯一的**。

二、最小生成树★

1、问题的提出:

●通讯网: 网中 n 个顶点—— n 个城市
两顶点间的边——两城市间线路
边的权——架设相应线路的费用

●问题1: n 个城市间的通讯网, 至少要多少条线路? ($n-1$)

n 个城市间最少的可行的通讯线路就是一棵生成树

●问题2: 选择怎样的 $n-1$ 条线路, 使总费用最少?

网上问题: 取 $n-1$ 条边, 并使边权总和为最少。

← 最小生成树问题

2、最小生成树定义

给定一个带权图，构造带权图的一棵生成树，
使树中所有边的权总和为最小。

3、最小生成树的构造算法

Prim普里姆算法和kruskal克鲁斯卡尔算法

普里姆算法的时间复杂度是 $O(n^2)$ ，与网中边数无关。

克鲁斯卡尔算法的时间复杂度为 $O(e \log e)$

【例6.3】利用普里姆算法，给出求图6.17(a)所示的无向网络的最小生成树的过程。

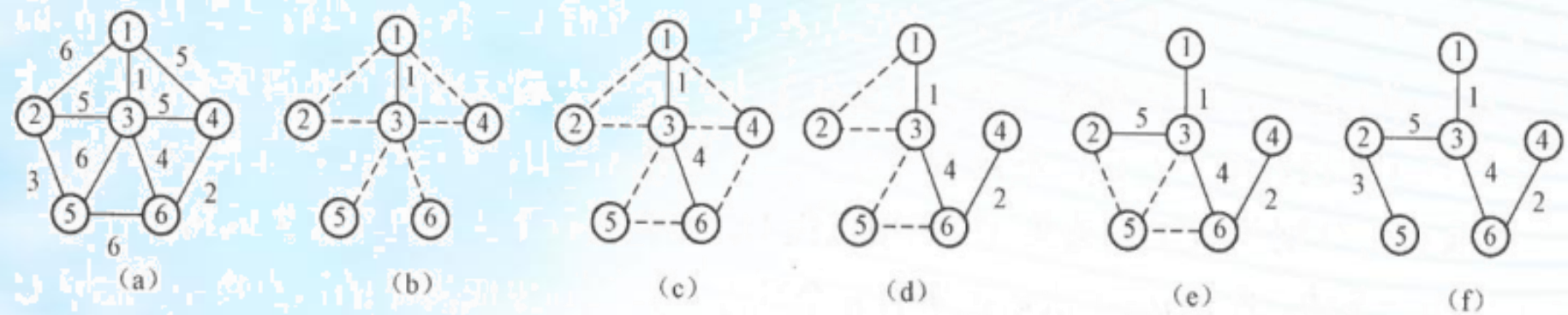
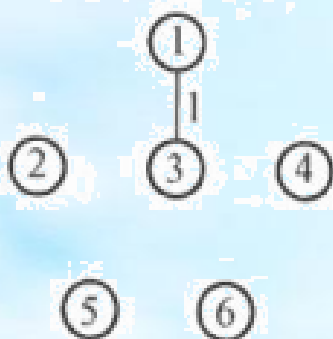
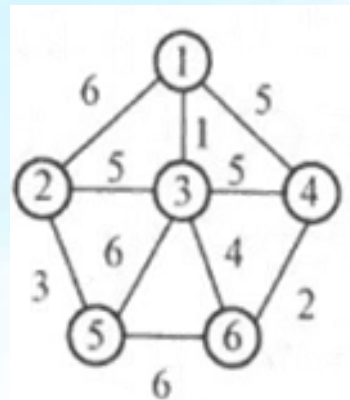
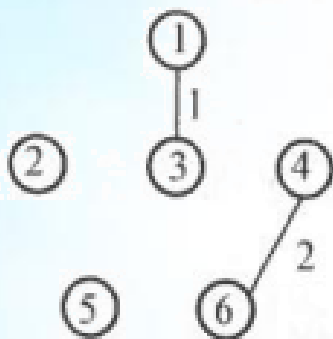


图 6.17 最小生成树的构造过程示意图

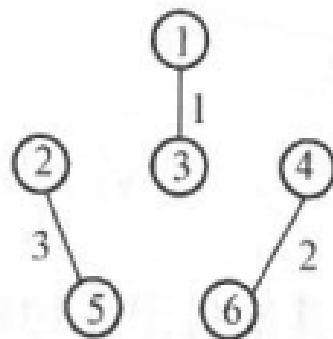
<div>minedge</div> <div>v</div>	2	3	4	5	6	U	V-U	说 明
ver lowcost	① 6	① 1	① 5			{1}	{2, 3, 4, 5, 6}	U (1, 3) 边最短
ver lowcost	③ 5		① 5	③ 6	③ 4	{1, 3}	{2, 4, 5, 6}	U (3, 6) 边最短
ver lowcost	③ 5		⑥ 2	③ 6	0	{1, 3, 6}	{2, 4, 5}	U (6, 4) 边最短
ver lowcost	③ 5		0	③ 6	0	{1, 3, 6, 4}	{2, 5}	U (3, 2) 边最短
ver lowcost	0		0	② 3	0	{1, 3, 6, 4, 2}	{5}	U (2, 5) 边最短
ver lowcost	0		0	0	0	{1, 3, 6, 4, 2, 5}	ϕ	



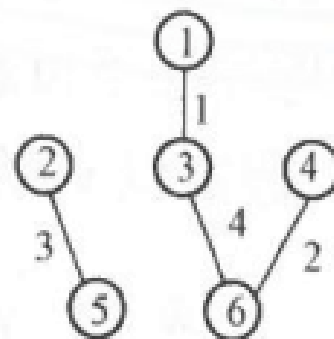
(a)



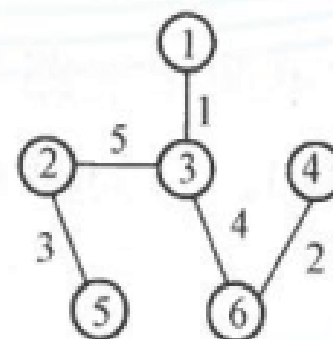
(b)



(c)



(d)



(e)

图 6.18 克鲁斯卡尔最小生成树的生成过程

6.5第五节 最短路径



最短路径问题的提法很多，在这里仅讨论单源最短路径问题：从某个源点 S 到 G 中其余各顶点的最短路径。对于求多源点的最短路径问题，可以用每个顶点作为源点调用一次单源最短路径问题算法予以解决。

迪杰斯特拉（Dijkstra）提出了按路径长度递增的顺序产生诸顶点的最短路径算法，称之为迪杰斯特拉算法。

第6章 图	6.3 图的遍历
	6.4 图的生成树和最小生成树
	6.5 最短路径
	6.6 拓扑排序

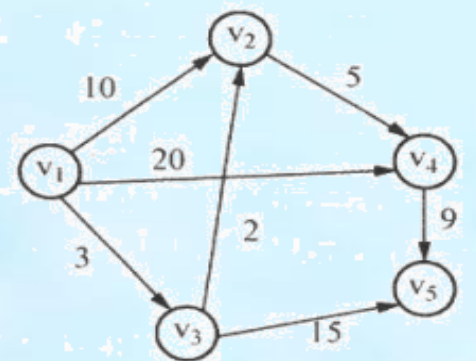


图 6.19 有向图 G_{10}

∞	10	3	20	∞
∞	∞	∞	5	∞
∞	2	∞	∞	15
∞	∞	∞	∞	9
∞	∞	∞	∞	∞

图 6.20 图 G_{10} 的带权邻接矩阵

表 6.3 迪杰斯特拉算法中 D 数组的变化情况

终点	从 v_1 到各终端的 D 值和最短路径的求解过程			
	$i=2$	$i=3$	$i=4$	$i=5$
D[2] v_2	10 (v_1, v_2)	5 (v_1, v_3, v_2)		
D[3] v_3	3 (v_1, v_3)			
D[4] v_4	20 (v_1, v_4)	20 (v_1, v_4)	10 (v_1, v_3, v_2, v_4)	
D[5] v_5	∞ (v_1, v_5)	18 (v_1, v_3, v_5)		
v_j	v_3	v_2	v_4	v_5
S	{ v_1, v_3 }	{ v_1, v_2, v_3 }	{ v_1, v_2, v_3, v_4 }	{ v_1, v_2, v_3, v_4, v_5 }

真题演练

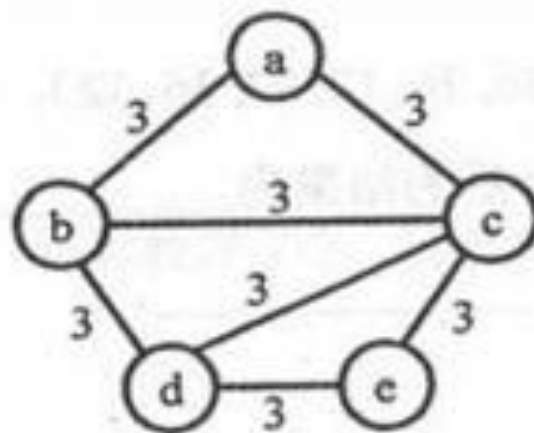
如果无向图G的最小生成树T中含有边(a,b)和(a,c),则下列选项中,一定不在T中的边是 ()。

A: (b,c)

B: (b,d)

C: (c,d)

D: (c,e)



真题演练

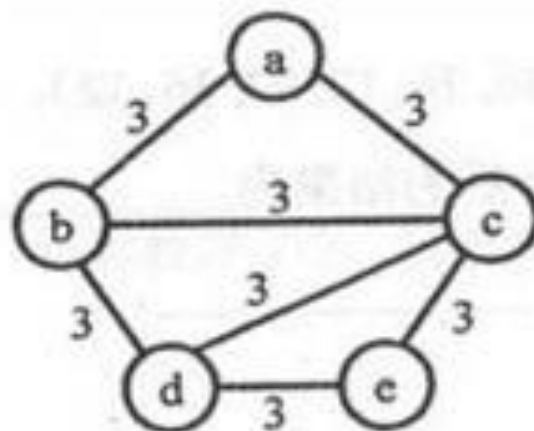
如果无向图G的最小生成树T中含有边(a,b)和(a,c),则下列选项中,一定不在T中的边是 ()。

A: (b,c)

B: (b,d)

C: (c,d)

D: (c,e)



答案: C

真题演练

下列关于有向带权图G的叙述中，错误的是（ ）。

- A:图G的任何一棵生成树都不含有回路
- B:图G生成树所含的边数等于顶点数减1
- C:图G含有回路时无法得到拓扑序列
- D:图G的最小生成树总是唯一的

真题演练

下列关于有向带权图G的叙述中，错误的是（ ）。

- A:图G的任何一棵生成树都不含有回路
- B:图G生成树所含的边数等于顶点数减1
- C:图G含有回路时无法得到拓扑序列
- D:图G的最小生成树总是唯一的

答案：D

真题演练

设带权连通图 G 中含有 $n(n > 1)$ 个顶点 e 条边。下列关于 G 的最小生成树的叙述中，正确的是（ ）。

- A:生成树中一定含有权值最小的 e 条边
- B:生成树中可能含有权值最小的 $n+1$ 条边
- C:生成树中一定含有权值最小的 n 条边
- D:生成树中可能含有权值最小的 $n-1$ 条边

真题演练

设带权连通图 G 中含有 $n(n > 1)$ 个顶点 e 条边。下列关于 G 的最小生成树的叙述中，正确的是（ ）。

- A:生成树中一定含有权值最小的 e 条边
- B:生成树中可能含有权值最小的 $n+1$ 条边
- C:生成树中一定含有权值最小的 n 条边
- D:生成树中可能含有权值最小的 $n-1$ 条边

答案：D

真题演练

设带权连通图 G 中含有 n ($n > 1$) 个顶点, 下列关于 G 的最小生成树 T 的叙述中, 正确的是 ()。

A: T 中可能含有回路

B: T 中含有图 G 的所有边

C: T 是唯一的, 且含有 $n-1$ 条边

D: T 可能不唯一, 但权一定相等

真题演练

设带权连通图 G 中含有 n ($n > 1$) 个顶点, 下列关于 G 的最小生成树 T 的叙述中, 正确的是 ()。

A: T 中可能含有回路

B: T 中含有图 G 的所有边

C: T 是唯一的, 且含有 $n-1$ 条边

D: T 可能不唯一, 但权一定相等

答案: D

真题演练

迪杰斯特拉(Dijkstra)算法的功能是 ()。

A:求图中某顶点到其他顶点的最短路径

B:求图中所有顶点之间的最短路径

C:求图的最小生成树

D:求图的拓扑排序序列

真题演练

迪杰斯特拉(Dijkstra)算法的功能是 ()。

A:求图中某顶点到其他顶点的最短路径

B:求图中所有顶点之间的最短路径

C:求图的最小生成树

D:求图的拓扑排序序列

答案: A

真题演练

在带权图的最短路径问题中，路径长度是指（ ）。

A:路径上的顶点数

B:路径上的边数

C:路径上的顶点数与边数之和

D:路径上各边的权值之和

真题演练

在带权图的最短路径问题中，路径长度是指（ ）。

A:路径上的顶点数

B:路径上的边数

C:路径上的顶点数与边数之和

D:路径上各边的权值之和

答案：D

6.6 拓扑排序

第6章 图	6.3 图的遍历
	6.4 图的生成树和最小生成树
	6.5 最短路径
	6.6 拓扑排序

一、问题的提出

图可以描述一个工程或系统的进行过程。



- 所有活动完成整个工程完成
- 活动间有一定的先后关系

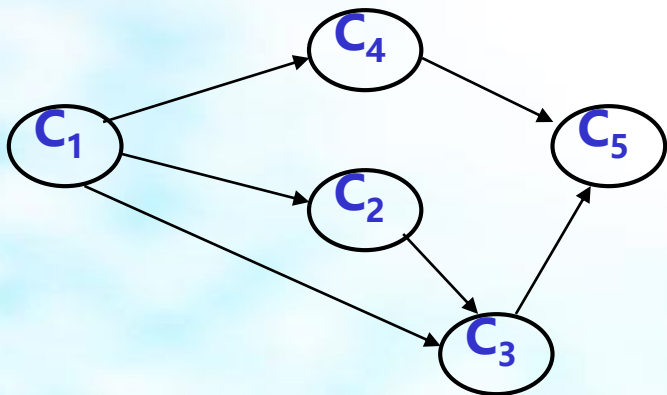
例：假定程序设计专业的学员要学5门课

这里 工程 \longrightarrow 完成5门课的学习

子工程 \longrightarrow 学习一门课程

课程编号	课程名称	先决条件
------	------	------

C_1	程序设计基础	无
C_2	离散数学	C_1
C_3	数据结构	C_1, C_2
C_4	汇编语言	C_1
C_5	高级语言	C_3, C_4



图中：

- 顶点表示课程；
- 有向边表示先决条件。

第6章 图

6.3 图的遍历

6.4 图的生成树和最小生成树

6.5 最短路径

6.6 拓扑排序

- 定义:若有向图**G**中, 顶点表示活动或任务, 边表示活动或任务之间的优先关系, 则称**G**为**AOV网络**, 即**顶点表示活动的网络(Activity on vertex network)**。

- 定义: 在一个**AOV网**中,
称**i**为**j**的前趋 \iff **i**到**j**有一条有向路径
称**i**为**j**的直接前趋 \iff 有边**<i,j>**

AOV网不能出现回路(否则自己以自己为前提, 不合理)

引出

如何检测**AOV网**中有无回路?

出现

拓扑排序

6.6第六节 拓扑排序

课程编号	课程名称	先修课程
C1	计算机基础	无
C2	高等数学	无
C3	数据结构	C4, C7
C4	算法语言	C1
C5	操作系统	C3, C6
C6	计算机原理	C9
C7	离散数学	C1, C2
C8	编译技术	C3, C4
C9	普通物理	C2

第6章 图	6.3 图的遍历
	6.4 图的生成树和最小生成树
	6.5 最短路径
	6.6 拓扑排序

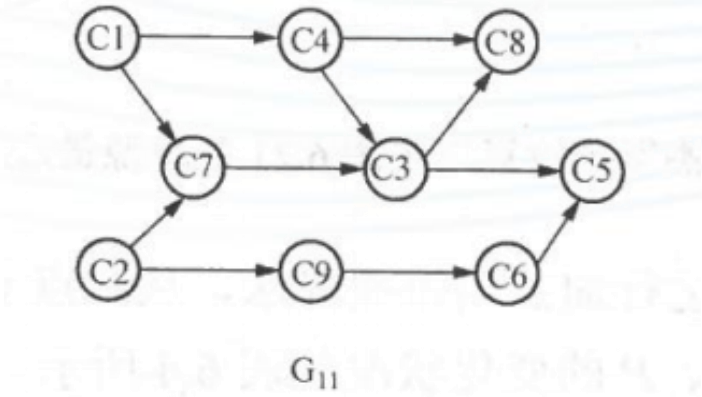


图 6.23 课程表的 AOV 网

AOV网的拓扑序列不是唯一的。例如，对图进行拓扑排序，至少可得到如下两个拓扑序列：
C1、C4、C2、C7、C3、C8、C9、C6、C5和C2、C9、C6、C1、C7、C4、C3、C5、C8。

算法的时间复杂度为 $O(n+e)$

6.6第六节 拓扑排序

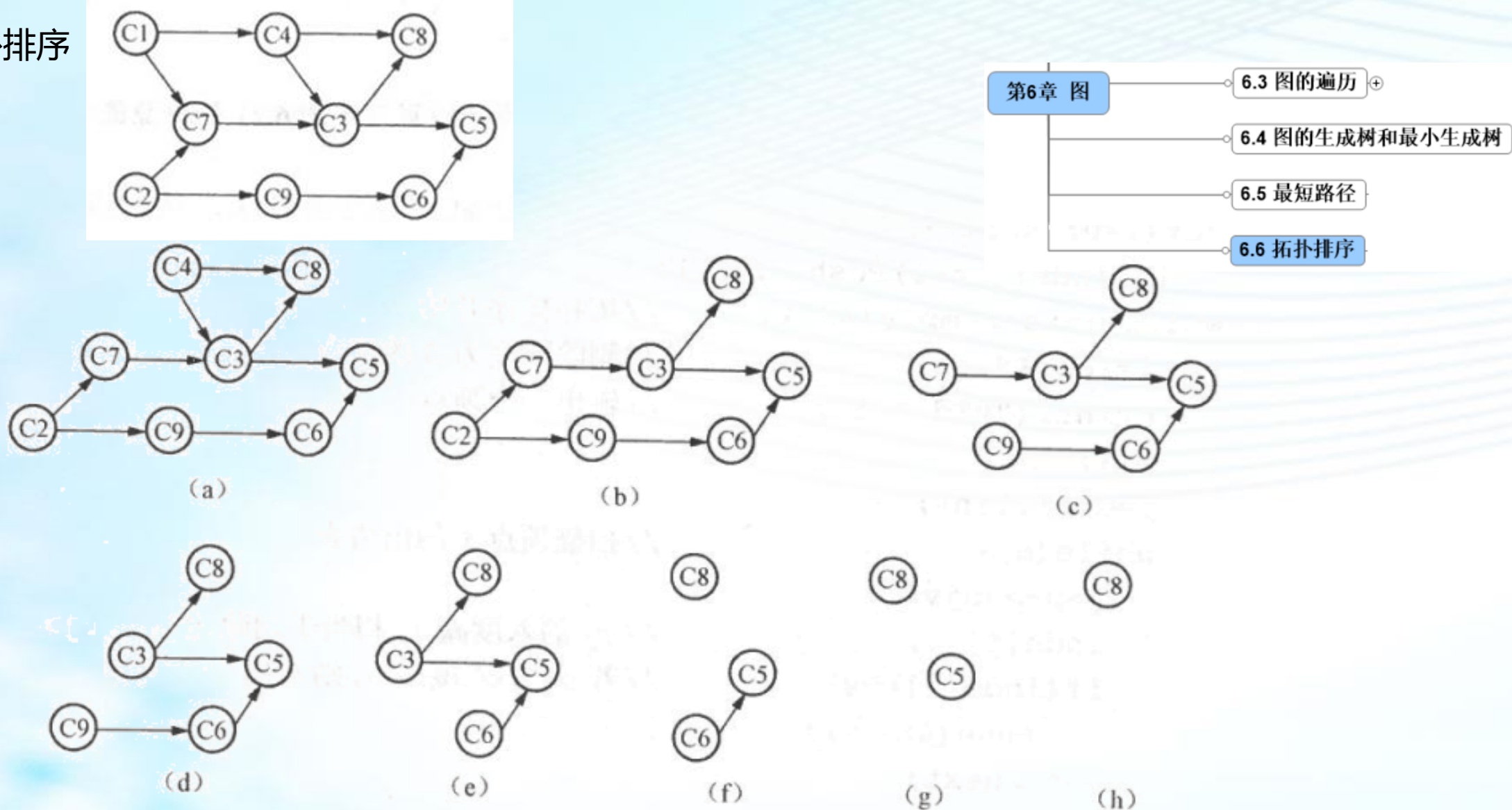
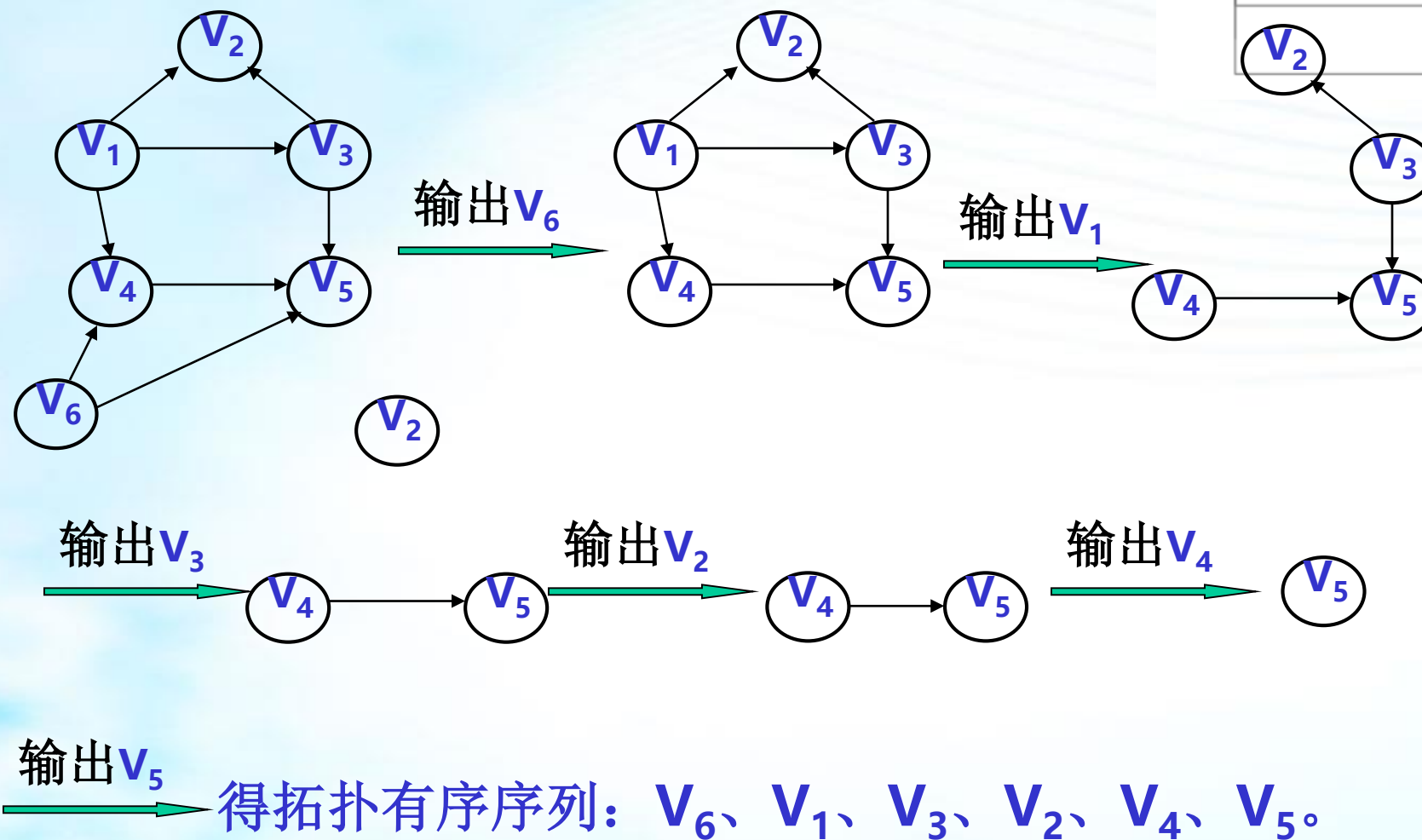


图 6.24 拓扑排序过程

AOV网的拓扑序列不是唯一的。例如，对图进行拓扑排序，至少可得到如下两个拓扑序列：
C1、C4、C2、C7、C3、C8、C9、C6、C5和C2、C9、C6、C1、C7、C4、C3、C5、C8。

6.6第六节 拓扑排序

例:



第6章 图	6.3 图的遍历
	6.4 图的生成树和最小生成树
	6.5 最短路径
	6.6 拓扑排序

真题演练

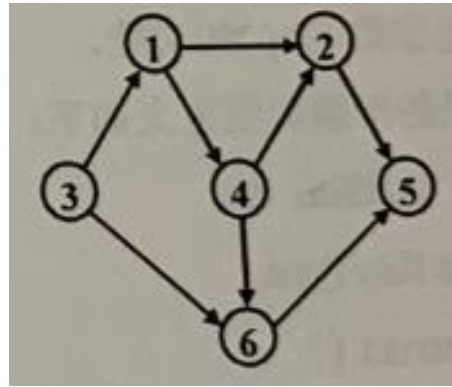
对题图所示的有向图进行拓扑排序。下列选项中能够得到的拓扑序列是（ ）。

A:3,1,2,4,5,6

B:3,1,2,4,6,5

C:3,1,4,2,5,6

D:3,1,4,2,6,5



真题演练

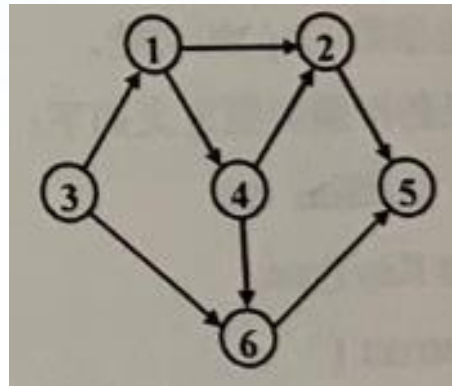
对题图所示的有向图进行拓扑排序。下列选项中能够得到的拓扑序列是（ ）。

A:3,1,2,4,5,6

B:3,1,2,4,6,5

C:3,1,4,2,5,6

D:3,1,4,2,6,5



答案：D

真题演练

若用邻接矩阵存储有向图，矩阵中主对角线以下的元素均为零，则关于该图拓扑排序序列的结论是（ ）。

A:存在，且唯一

B:存在，且不唯一

C:存在.可能不唯一

D:无法确定是否存在

真题演练

若用邻接矩阵存储有向图，矩阵中主对角线以下的元素均为零，则关于该图拓扑排序序列的结论是（ ）。

A:存在，且唯一

B:存在，且不唯一

C:存在.可能不唯一

D:无法确定是否存在

答案：C

真题演练

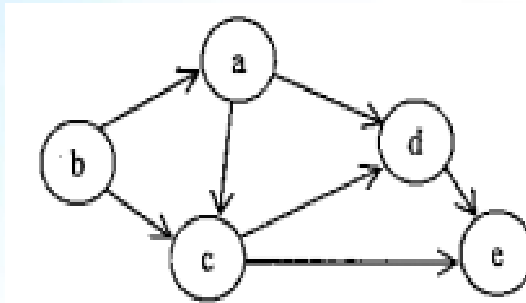
对下图进行拓扑排序，可以得到的拓扑序列是（ ）。

A:a b c d e

B:b a c d e

C:b c a d e

D:a b d c e



真题演练

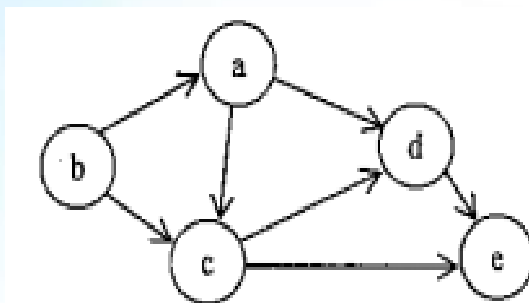
对下图进行拓扑排序，可以得到的拓扑序列是（ ）。

A:a b c d e

B:b a c d e

C:b c a d e

D:a b d c e



答案：B

真题演练

可进行拓扑排序的图只能是（ ）。

A:有向图

B:无向图

C:有向无环图

D:无向连通图

真题演练

可进行拓扑排序的图只能是（ ）。

A:有向图

B:无向图

C:有向无环图

D:无向连通图

答案：C

真题演练

下列关于有向无环图G的拓扑排序序列的叙述中，正确的是（ ）。

A:存在且唯一

B:存在且不唯一

C:存在但可能不唯一

D:无法确定是否存在

真题演练

下列关于有向无环图G的拓扑排序序列的叙述中，正确的是（ ）。

A:存在且唯一

B:存在且不唯一

C:存在但可能不唯一

D:无法确定是否存在

答案：C

真题演练

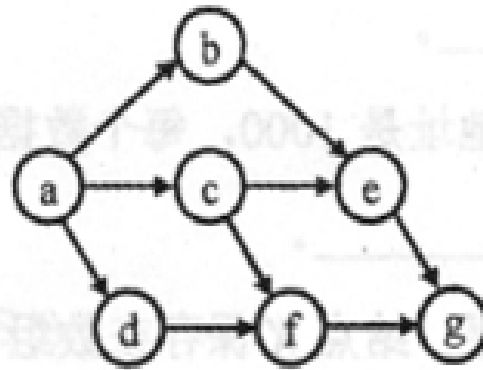
已知有向图G如下所示，G的拓扑序列是（ ）。

A:a, b, e, c, d, f, g

B:a, c, b, f, d, e, g

C:a, c, d, e, b, f, g

D:a, c, d, f, b, e, g



真题演练

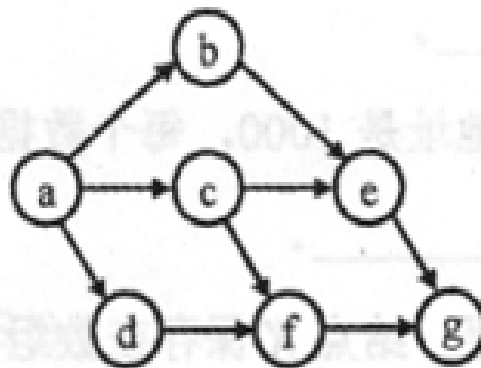
已知有向图G如下所示，G的拓扑序列是（ ）。

A:a, b, e, c, d, f, g

B:a, c, b, f, d, e, g

C:a, c, d, e, b, f, g

D:a, c, d, f, b, e, g



答案：D

本章总结





祝大家顺利通过考试！