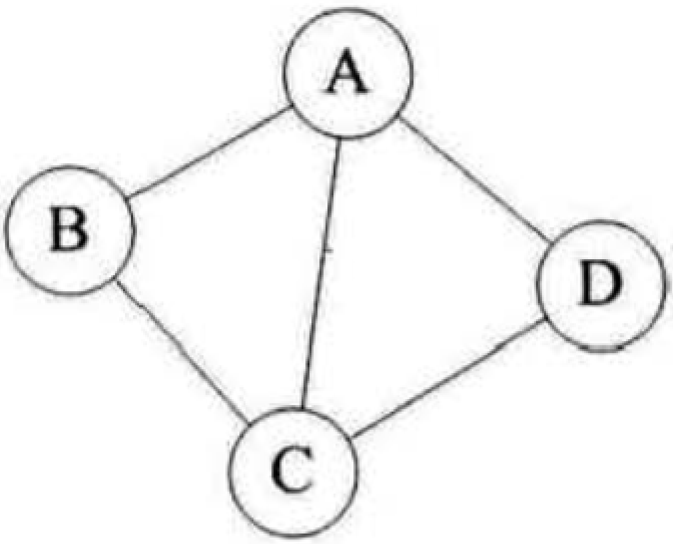
一、图

1. 图的类型

1-1. 无向图

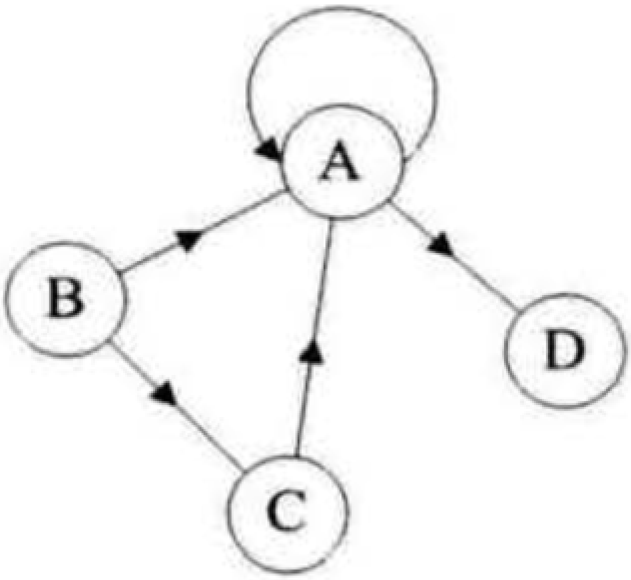
无向边：若顶点vi到vj之间的边没有方向，则称这条边为无向表，用无序偶对(vi，vj)来表示。如果图中任意两个顶点之间的边都都是无向表，则称该图为无向图。



对于上面的这个无向图，G1=(V1,{E1})，其中顶点集合V1={A,B,C,D};边集合E1={（A,B），(B,C),（C,D），（D,A），（A,C）}

1-2. 有向图

若从顶点vi到vj的边有方向，则称这条边为有向边，也称为弧。用有序偶<vi，vj>来表示，vi称为弧尾，vj称为弧头。如果图中任意两个顶点之间的边都是有向边，则称该图为有向图。

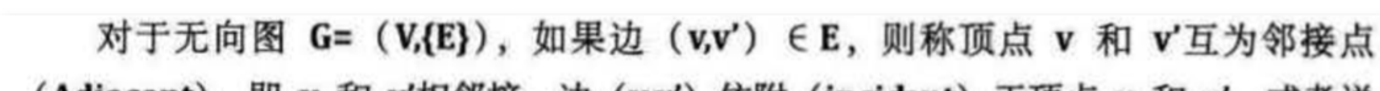


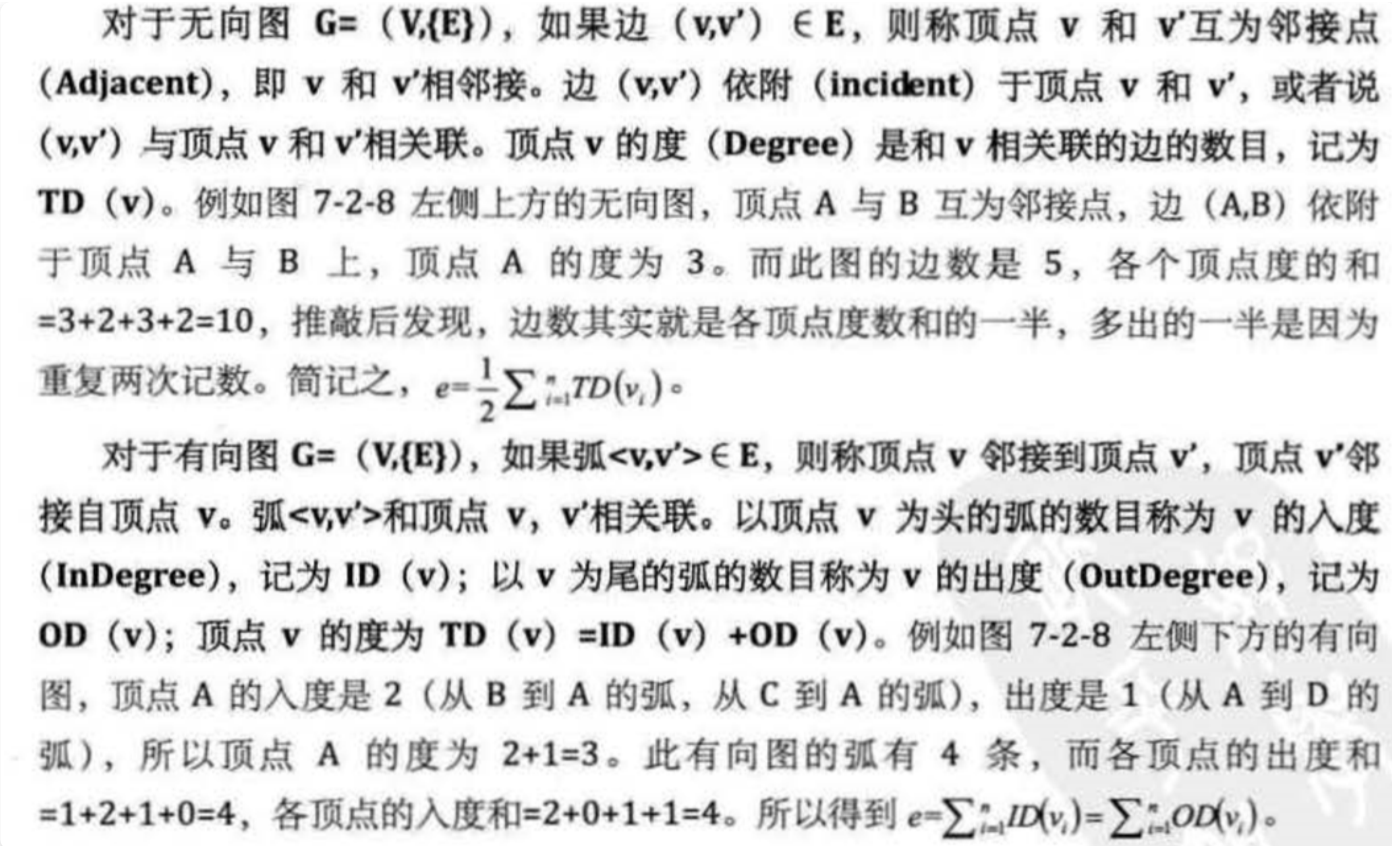
对于上面的有向图，G2=(V2,{E2}),其中顶点集合V2={A,B,C,D},弧集合E2={<A,D>,<B,A>,<C,A>,<B,C>}

1-3. 完全图

在无向图/有向图，如果任意两个顶点之间都存在边，则称该图为无向/有向完全图

2. 图的顶点与边间关系


对于无向图 G=（V,{E}），如果边（v,v'）∈E，则称顶点 v 和 v'互为邻接点


对于无向图 G=（V,{E}），如果边（v,v'）∈E，则称顶点 v 和 v'互为邻接点（Adjacent），即 v 和 v'相邻接。边（v,v'）依附（incident）于顶点 v 和 v'，或者说（v,v'）与顶点 v 和 v'相关联。顶点 v 的度（Degree）是和 v 相关联的边的数目，记为 TD（v）。例如图 7-2-8 左侧上方的无向图，顶点 A 与 B 互为邻接点，边（A,B）依附于顶点 A 与 B 上，顶点 A 的度为 3。而此图的边数是 5，各个顶点度的和=3+2+3+2=10，推敲后发现，边数其实就是各顶点度数和的一半，多出的一半是因为重复两次记数。简记之，$e=\frac{1}{2}\sum_{i=1}^{n}TD(v_i)$。

对于有向图 G=（V,{E}），如果弧<v,v'>∈E，则称顶点 v 邻接到顶点 v'，顶点 v'邻接自顶点 v。弧<v,v'>和顶点 v，v'相关联。以顶点 v 为头的弧的数目称为 v 的入度（InDegree），记为 ID（v）；以 v 为尾的弧的数目称为 v 的出度（OutDegree），记为 OD（v）；顶点 v 的度为 TD（v）=ID（v）+OD（v）。例如图 7-2-8 左侧下方的有向图，顶点 A 的入度是 2（从 B 到 A 的弧，从 C 到 A 的弧），出度是 1（从 A 到 D 的弧），所以顶点 A 的度为 2+1=3。此有向图的弧有 4 条，而各顶点的出度和=1+2+1+0=4，各顶点的入度和=2+0+1+1=4。所以得到 $e=\sum_{i=1}^{n}ID(v_i)=\sum_{i=1}^{n}OD(v_i)$。

enter description here

树中根结点到任意结点的路径是唯一的，但是图中顶点与顶点之间的路径却是不唯一的。路径的长度是路径上的边或弧的数目。

3. 连通图相关术语

在无向图G中，如果从顶点v到顶点u有路径，则称v和u是连通的。如果对于图中任意两个顶点vi、vj属于E,vi和vj都是连通的，则称G是连通图。
无向图中的极大连通子图称为连通分量。注意连通分量的概念，它强调：

- 要是子图
- 子图要是连通的
- 连通子图含有极大顶点数
- 具有极大顶点数的连通子图包含依附这些顶点的所有边

在有向图G中，如果对于每一对vi、vj，从vi到vj和从vj到vi都存在路径，则称G是强连通图。

3-1. 连通图生成树

所谓的连通图的生成树是一个极小的连通子图，它含有图中全部的n个顶点，但只有足以构成一棵树的n-1条边。
图一是普通图，但显然不是生成树，当去掉两条构成环的边后，比如图2和图三，就满足n个顶点n-1条边，都是一棵生成树。如果一个图有n个顶点和小于n条边，则是非连通图，如果它多于n-1条边，必定构成一个环。

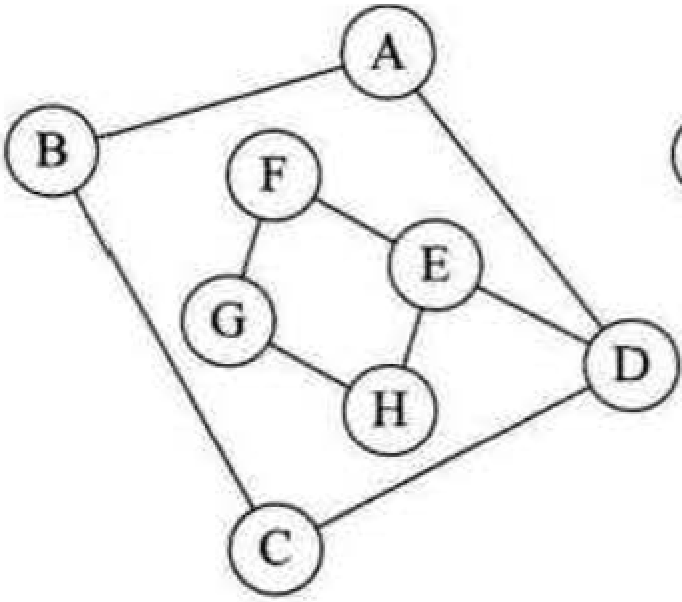


图1

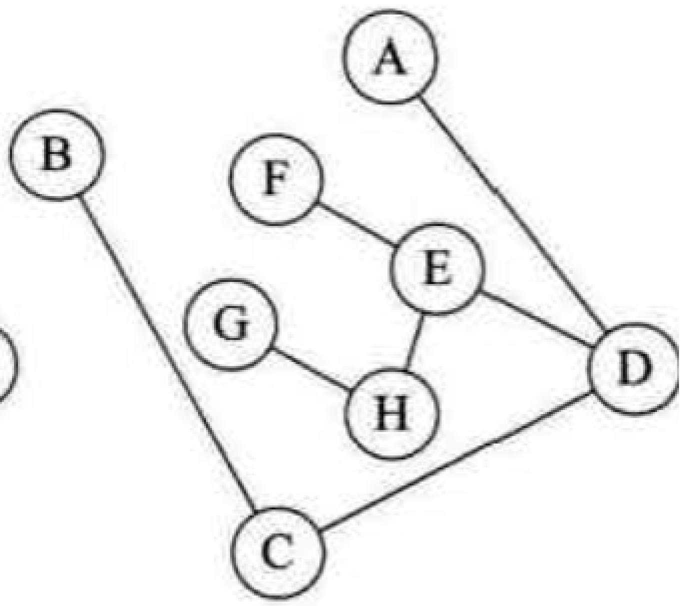


图2

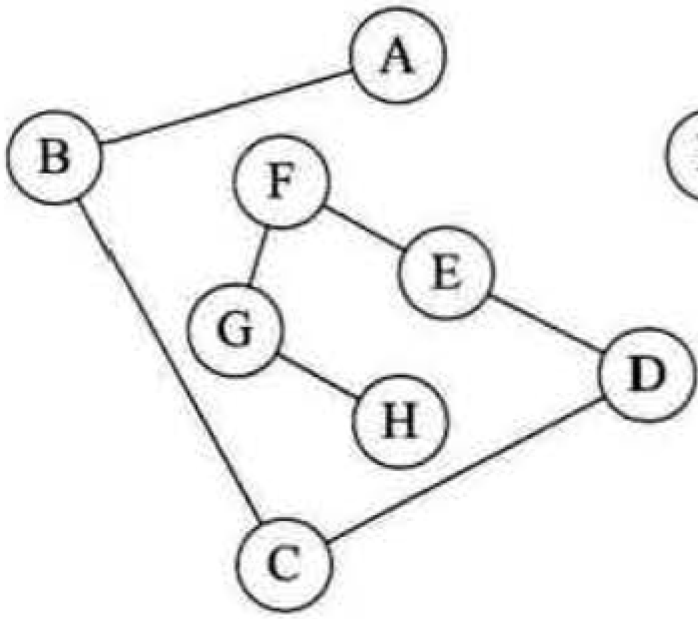


图3

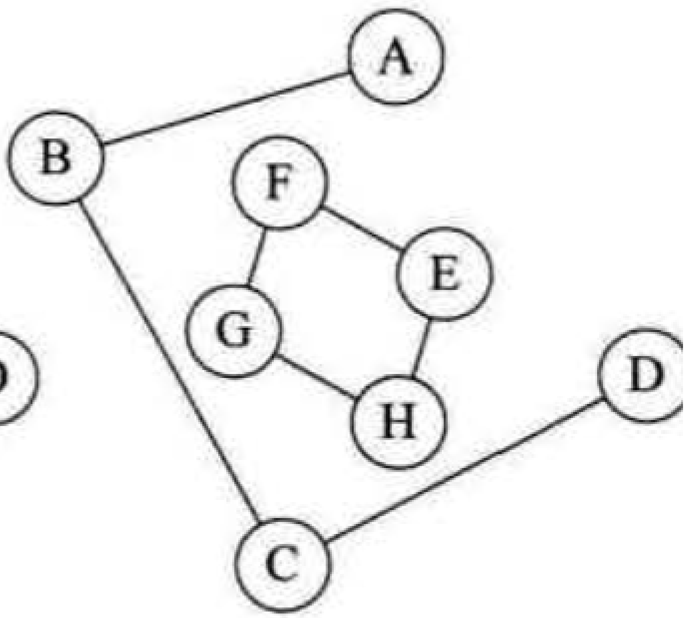


图4

3-2. 有向树

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，则是一棵有向树。所谓入度为0其实就相当于树中的根结点，其余顶点入度为1就是树中的非根结点的双亲只

有一个。**一个有向图的生成森林由若干棵有向树组成，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**

4. 图的定义和术语总结

图按照有无方向分为**无向图**和**有向图**。无向图由**顶点**和**边**构成，有向图由顶点和**弧**构成。弧有**弧尾**和**弧头**之分。

图按照边或弧的多少分**稀疏图**和**稠密图**。如果任意两个顶点之间都存在边叫**完全图**，有向的叫**有向完全图**。若无重复的边或顶点到自身的边则叫**简单图**。

图中顶点之间有**邻接点**、**依附**的概念。无向图顶点的边数叫**做度**，有向图顶点分为**入度**和**出度**。

图上的边或弧上带**权**则称为**网**。

图中顶点间存在**路径**，两顶点存在路径则说明是**连通**的，如果路径最终回到起始点则称为**环**，当中不重复叫**简单路径**。若任意两顶点都是连通的，则图就是**连通图**，有向则称**强连通图**。图中有子图，若子图极大连通则就是**连通分量**，有向的则称**强连通分量**。

无向图中连通且 n 个顶点 n−1 条边叫**生成树**。有向图中一顶点入度为 0 其余顶点入度为 1 的叫**有向树**。一个有向图由若干棵有向树构成**生成森林**。

enter description here

5. 图的抽象数据类型

6. 图的存储结构

从图的逻辑结构定义来看，图上任何一个顶点都可被看成是第一个顶点，任一顶点的邻接点也不存在次序关系。
比如下面四张图，其实它们是同一个图，只不过顶点的位置不同，就造成了表象上不太

一样的感觉

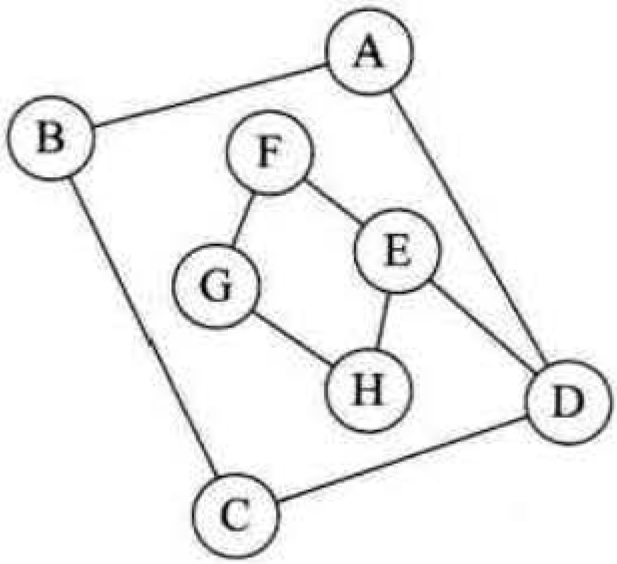


图1

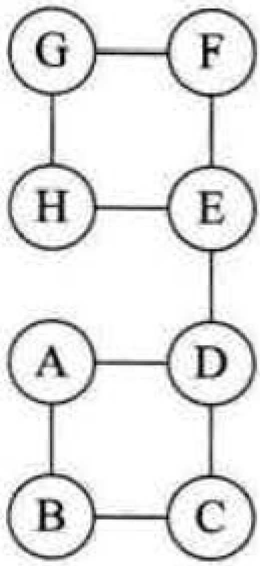
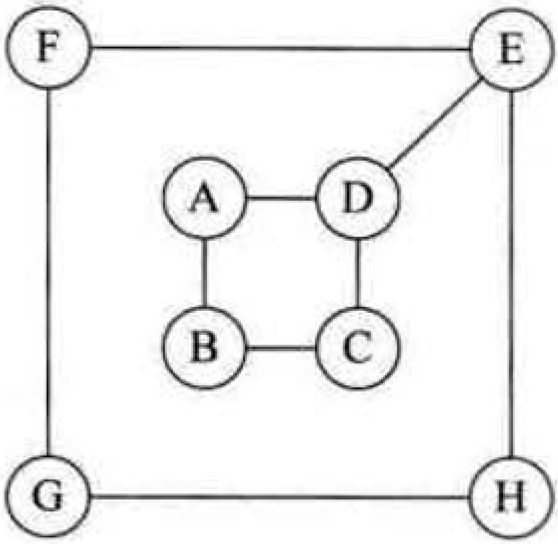
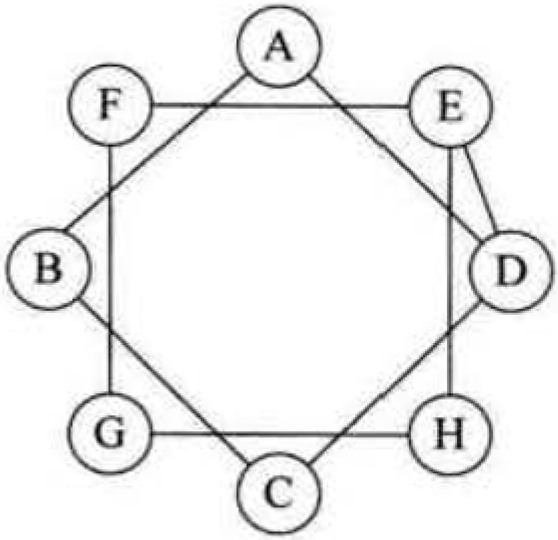


图2



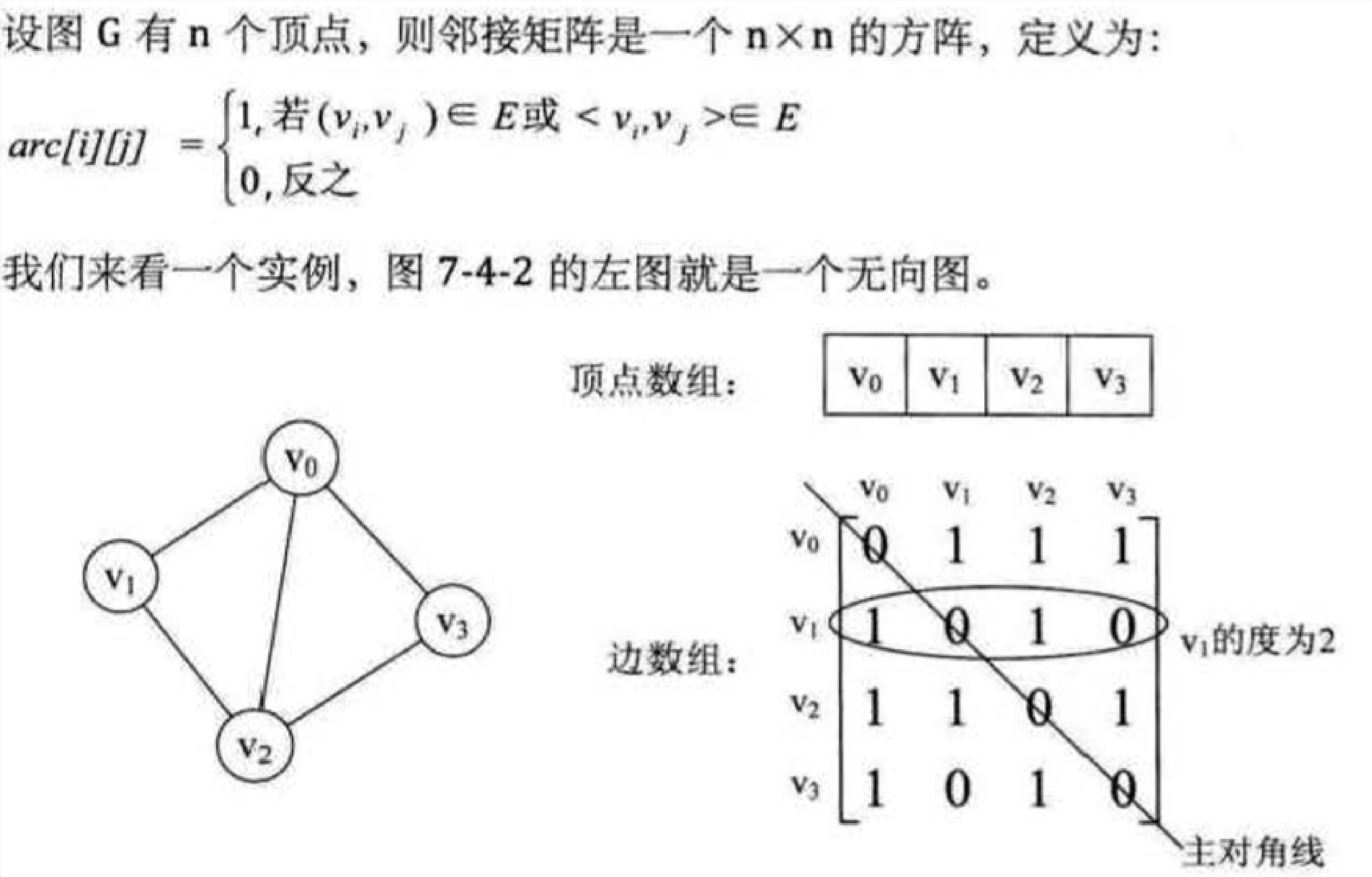


任意两个顶点之间都可能存在联系，因此无法以数 据元素在内存中的物理位置来表示元素之间的关系，也就是说，图不可能用简单的顺 序存储结构来表示。而多重链表的方式，即以一个数据域和多个指针域组成的结点表 示圈中的一个顶点，尽管可以实现图结构，但其实在树中，我们也已经讨论过，这是 有问题的。如果各个顶点的度数相差很大，按度数最大的顶点设计结点结构会造成很 多存储单元的浪费，而若按每个顶点自己的度数设计不同的顶点结构，又带来操作的 不便。

6-1. 邻接矩阵

考虑到图是由顶点和边或弧两部分组成。合在一起比较困难，那就很自然地考虑 到分两个结构来分别存储。顶点不分大小、主次，所以用一个一维数组来存储是很不 错的选择。而边或弧由于是顶点与顶点之间的关系， 一维搞不定，那就考虑用一个二 维数组来存储。于是我们的邻接矩阵的方案就诞生了。

图的邻接矩阵（Adjacency Matrix）存储方式是用两个数组来表示圈。一个一维 数组存储圈中顶点信息，一个二维数组（称为邻接矩阵)存储图中的边或弧的信息



enter description here

我们可以设置两个数组，顶点数组为 vertex[4]={ vo，v1， v2， v3]，边数组 arc[4][4]为
上如图这样的一个矩阵。简单解释一下，对于矩阵的主对角钱的值，即 arc[0][0]、 arc[l][l]、 arc[2][2] 、 arc[3][3] ，全为 0 是因为不存在顶点到自身的边，比 如 Vo 到 Vo. arc[O][l]=l 是因为 Vo到 VI 的边存在，而 arc[1][3]=0 是因为 VI 到 V3的边 不存在。 并且由于是无向图， VI 到V3的边不存在，意味着 V3 到V1的边也不存在。所 以无向固的边数组是一个对称矩阵。
有了这个矩阵，我们就可以很容易地知道图中的信息:

- .我们要判定任意两顶点是否有边无边就非常容易了
- 我们要知道某个顶点的度，其实就是这个顶点 VI在邻接矩阵中第 i 行(或第 i 列)的元素之相。比如顶点 VI 的度就是 1+0+1+0=2

- 求顶点 Vi 的所有邻接点就是将矩阵中第 1 行元素扫描一遍， arc[i]Ul为 1 就是邻接点

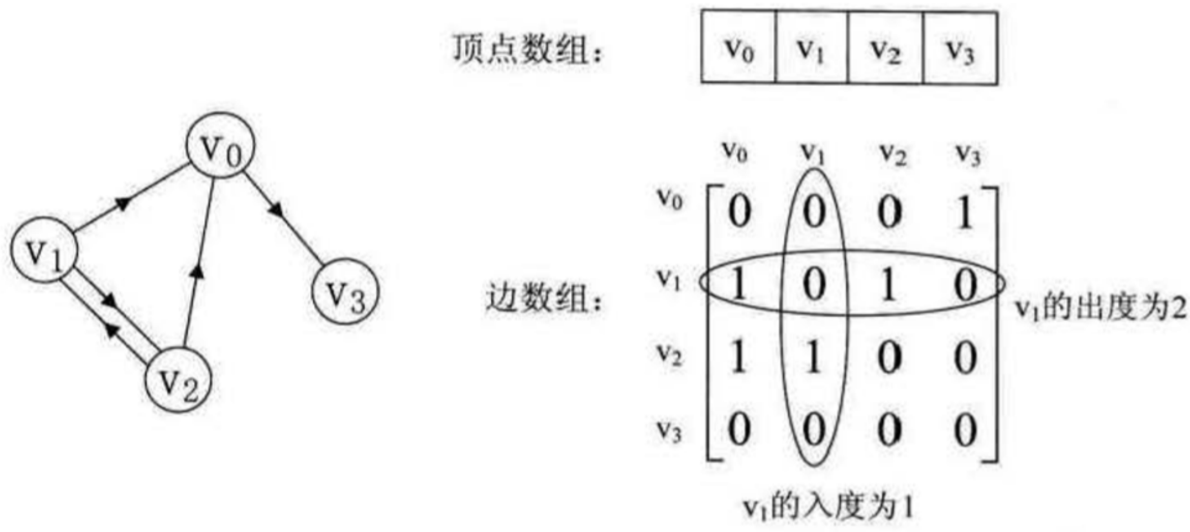


图 7-4-3

顶点数组为 vertex[4]={ v_0, v_1, v_2, v_3}，弧数组 arc[4][4]为图 7-4-3 右图这样的一个矩阵。主对角线上数值依然为 0。但因为是有向图，所以此矩阵并不对称，比如由 v_1 到 v_0 有弧，得到 arc[1][0]=1，而 v_0 到 v_1 没有弧，因此 arc[0][1]=0。

有向图讲究入度与出度，顶点 v_1 的入度为 1，正好是第 v_1 列各数之和。顶点 v_1 的出度为 2，即第 v_1 行的各数之和。

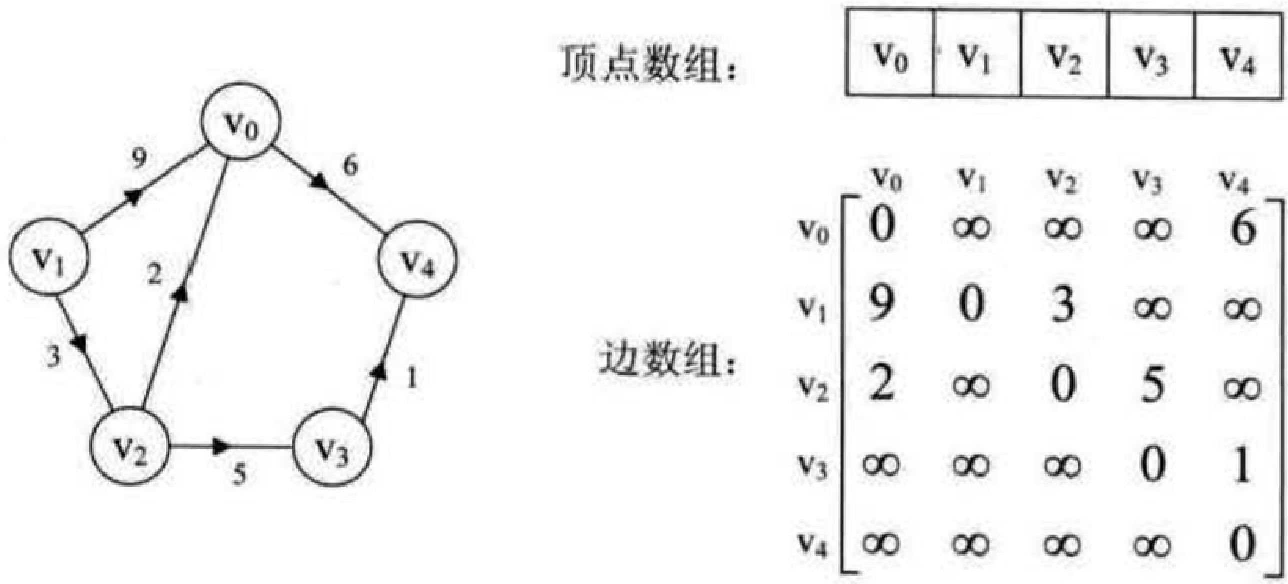
enter description here

在图的术语中，我们提到了网的概念，也就是每条边上带有权的图叫做网。那么 这些权值就需要存下来，如何处理这个矩阵来适应这个需求呢?我们有办法
设图 G 是网图，有 n 个顶点，则邻接矩阵是一个 n X n 的方阵，定义为:

$$arc[i][j]=\begin{cases} W_{ij}, 若 (v_i,v_j) \in E 或 < v_i,v_j > \in E \\ 0, \quad 若 i = j \\ \infty, \quad 反之 \end{cases}$$

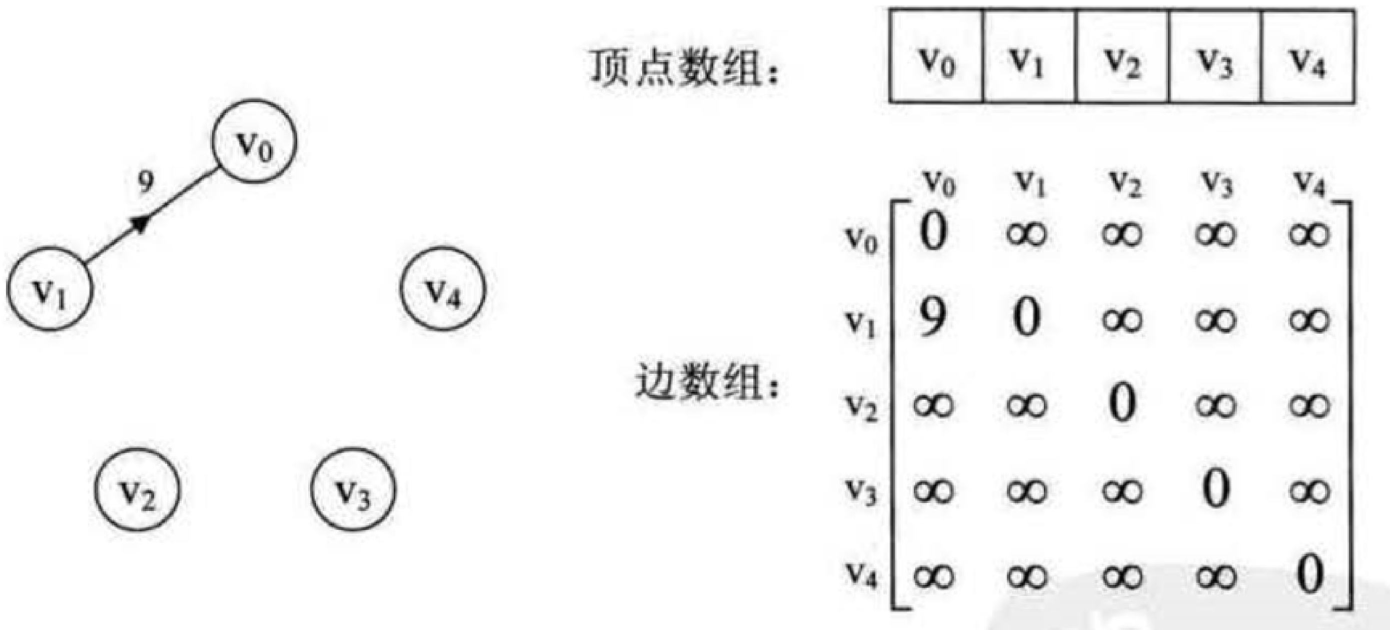
这里 Wi 表示 (VI,VJ)或<VI.Vj>上的权值。∞表示一个计算机允许的、大于所有边 上权值的值，也就是一个不可能的极限值。有同学会间，为什么不是 0 呢?原因在于 权值 Wj 大多数情况下是正值，但个别时候可能就是 0，甚至有可能是负值。因此必须 要用

一个不可能的值来代表不存在



6-2. 邻接表

邻接矩阵是不错的一种图存储结构，但是我们也发现，对于边数相对顶点较少的 图，这种结构是存在对存储空间的极大浪费的。比如说，如果我们要处理图 7牛5 这 样的稀疏有向图，邻接矩阵中除了 arc[l][0]有权值外，没有其他弧，其实这些存储空 间都浪费掉了

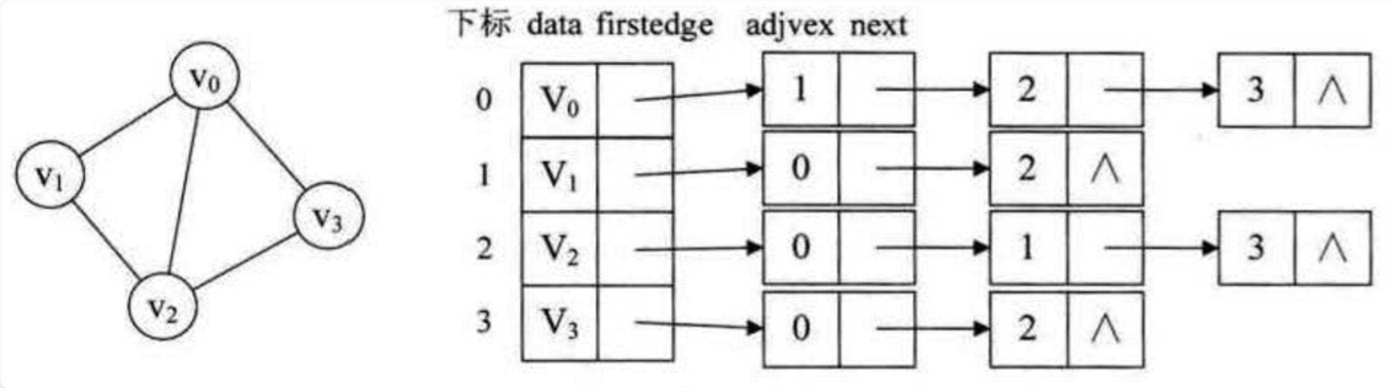


将数组和链表相结合的存储方法称为邻接表

邻接表的处理办法是这样：

1. 图中顶点用一个一维数组存储，当然，顶点也可以用单链表来存储，不过数组 可以较容易地读取顶点信息，更加方便。另外，对于顶点数组中，每个数据元 素还需要存储指向第一个邻接点的指针，以便于查找该顶点的边信息

2. 图中每个顶点 Vj 的所有邻接点构成一个线性表，由于邻接点的个数不定，所以 用单链表存储，无向图称为顶点 Vj 的边表，有向图则称为顶点 Vj 作为弧尾的 出边表



enter description here

从图中我们知道，顶点表的各个结点由 data 和 firstedge 两个域表示，data 是数据域，存储顶点的信息，firstedge 是指针域，指向边表的第一个结点，即此顶点的第一个邻接点。边表结点由 adjvex 和 next 两个域组成。adjvex 是邻接点域，存储某顶点的邻接点在顶点表中的下标，next 则存储指向边表中下一个结点的指针。比如 v_1 顶点与 v_0、v_2 互为邻接点，则在 v_1 的边表中，adjvex 分别为 v_0 的 0 和 v_2 的 2。

6-3. 十字链表

6-4. 邻接多重表

6-5. 边集数组

7. 图的遍历

从圈中某一顶点出发访遍圈中其余顶点， 且使每一个顶点仅被访问一次，这一过程就叫做固的遍历 。
树的遍历有四种方案，应该说都还好，毕竟根结点只有一个，遍历都是 从图发起，其余所有结点都只有一个双亲。可图就复杂多了，因为图的任一顶点都可 能和其余的所有顶点相邻接，极有可能存在沿着某条路径搜索后，又回到原顶点，而 有些顶点却还没有遍

历到的情况。因此我们需要在遍历过程中把访问过的顶点打上标 记，以避免访问多次而不自知。具体办法是设置一个访问数组 visited[n]， D 是图中顶 点的个数，初值为 0， 访问过后设置为 10 这其实在小说中常常见到，一行人在迷宫 中迷了路，为了避免找寻出路时屡次重复，所以会在路口用小刀刻上标记。

7-1. 深度优先遍历

深度优先遍历也有称为深度优先搜索，简称为DFS。
深度优先遍历本质上就是一个递归的过程，也像是一棵树的前序遍历。**它从图中某个顶点 v 出发，访问此顶点，然后从 v 的未被访问的邻接点出发 深度优先遍历圈，直至图中所有和 v 有路径相通的顶点都被访问到**
//TODO

7-2. 广度优先遍历

广度优先遍历，又称为广度优先搜索，简称BFS。
//TODO

8. 最小生成树

我们把构造连通网的**最小代价**生成树成为最小生成树。找连通网的最小生成树，经典算法有：普里姆算法和克鲁斯卡尔算法。

8-1. 普里姆算法

//TODO

8-2. 克鲁斯卡尔算法

//TODO

9. 最短路径

9-1. 迪杰斯特拉(Dijkstra ）算法

这是一个按路径长度递增的次序产生最短路径的算法。它的思路大体是这样的:
//TODO

9-2. 弗洛伊德(Floyd)算法

//TODO