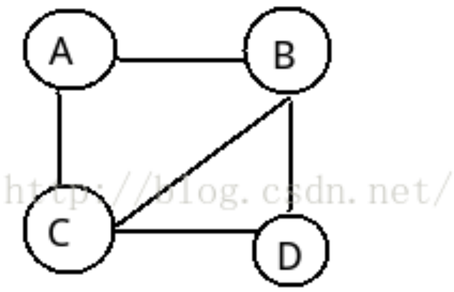
http://blog.csdn.net/gamer_gyt/article/details/51498546

一：图的分类

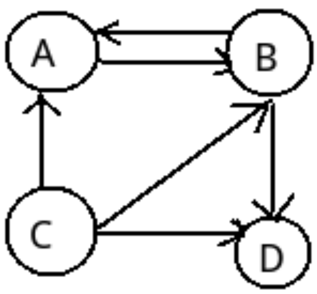
1：无向图

即两个顶点之间没有明确的指向关系，只有一条边相连，例如，A顶点和B顶点之间可以表示为 <A, B> 也可以表示为<B, A>，如下所示



2：有向图

顶点之间是有方向性的，例如A和B顶点之间，A指向了B，B也指向了A，两者是不同的，如果给边赋予权重，那么这种异同便更加显著了



==

在次基础上，根据图的连通关系可以分为

无向完全图：在无向图的基础上，每两个顶点之间都存在一条边，一个包含N个顶点的无向完全图，其总边数为N（N-1）/2

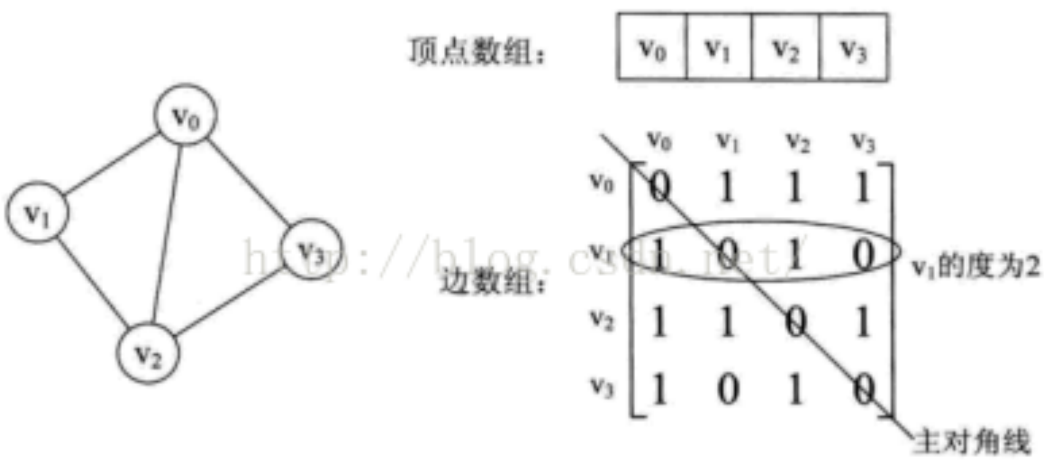
有向完全图：在有向图的基础上，每两个顶点之间都存在一条边，一个包含N个顶点

的有向完全图，其总边数为N（N-1）
连通图：针对无向图而言的，如果任意两个顶点之间是连通的，则该无向图称为连通图
非连通图：无向图中，存在两个顶点之间是不连通的，则该无向图称为非连通图
强连通图：针对有向图而言的，如果有向图中任意两个顶点之间是连通的（注意方向问题，A—>B，成立，但B—>A不一定成立），则该有向图称为强连通图

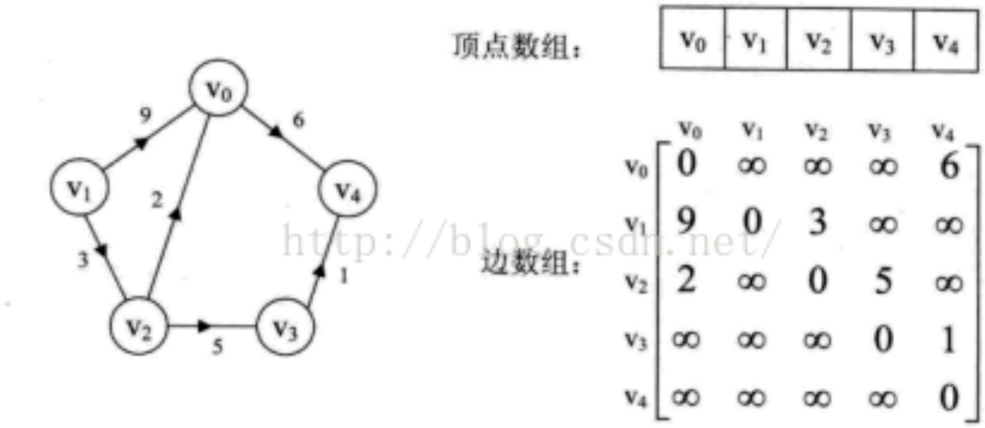
二：图的存储结构

1：邻接矩阵

使用二维数组来存储图的边的信息和权重，如下图所示的4个顶点的无向图



从上面可以看出，无向图的边数组是一个对称矩阵。所谓对称矩阵就是n阶矩阵的元满足$a_{ij} = a_{ji}$。即从矩阵的左上角到右下角的主对角线为轴，右上角的元和左下角相对应的元全都是相等的。
如果换成有向图，则如图所示的五个顶点的有向图的邻接矩阵表示如下



2：邻接表

邻接矩阵是一种不错的图存储结构，但是对于边数相对较少的图，这种结构存在空间上的极大浪费，因此找到一种数组与链表相结合的存储方法称为邻接表。

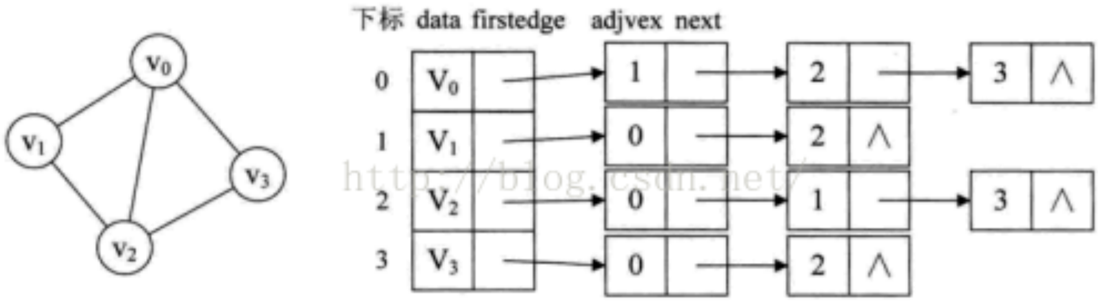
邻接表的处理方法是这样的：

（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，数组可以较容易的读取顶点的信息，更加方便。

（2）图中每个顶点Vi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，

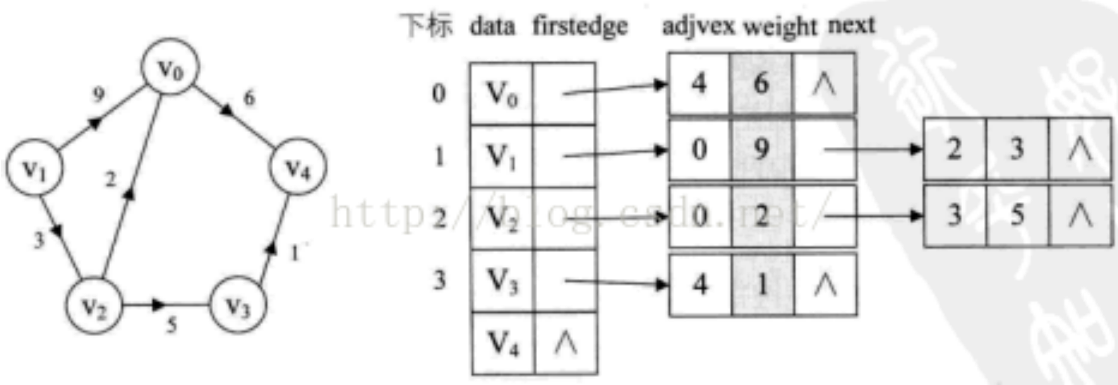
无向图称为顶点Vi的边表，有向图则称为顶点vi作为弧尾的出边表

如下为无向图的邻接表表示：



从图中可以看出，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。

有向图的邻接表表示：



3：十字链表

对于邻接表来说，计算顶点的入度是不方便的，那么有没有一种存储方式能够轻松的计算顶点的入度和出度呢，答案是肯定的

在十字链表中重新定义了节点的结构：

data	firstin	firstout

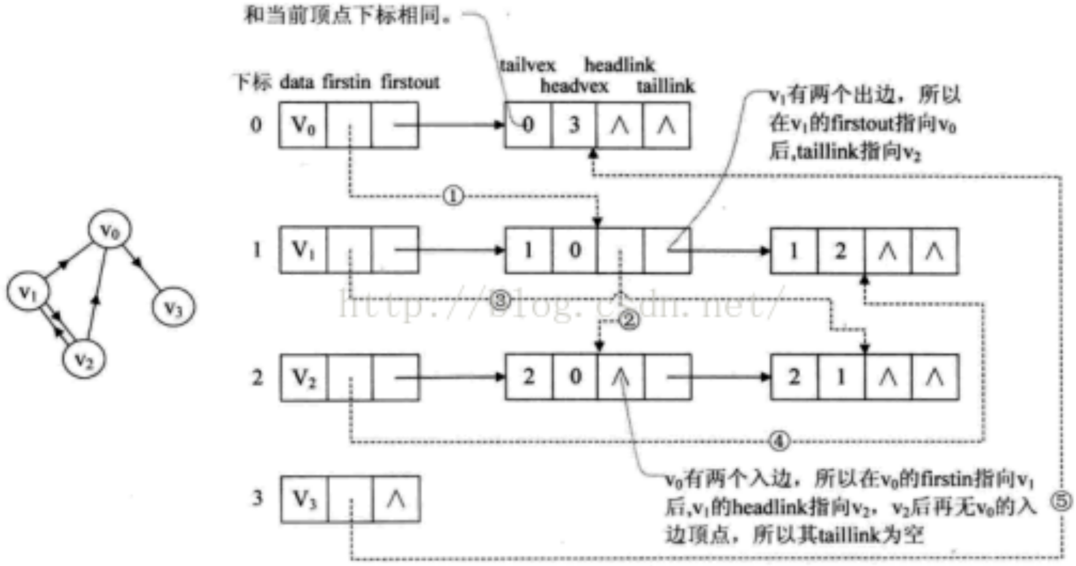
firstin表示入边表头指针，指向该顶点的入边表中第一个结点，firstout表示出边表头指针，指向该顶点的出边表中的第一个结点

重新定义的边表结构为：

tailvex	headvex	headlink	taillink

其中，tailvex是指弧起点在顶点表的下表，headvex是指弧终点在顶点表的下标，headlink是指入边表指针域，指向终点相同的下一条边，taillink是指边表指针域，指向起点相同的下一条边。如果是网，还可以增加一个weight域来存储权值。

比如下图，顶点依然是存入一个一维数组，实线箭头指针的图示完全与邻接表相同。就以顶点v0来说，firstout指向的是出边表中的第一个结点v3。所以，v0边表结点hearvex = 3，而tailvex其实就是当前顶点v0的下标0，由于v0只有一个出边顶点，所有headlink和taillink都是空的。



重点需要解释虚线箭头的含义。它其实就是此图的逆邻接表的表示。对于v0来说，它有两个顶点v1和v2的入边。因此的firstin指向顶点v1的边表结点中headvex为0的结点，如上图圆圈1。接着由入边结点的headlink指向下一个入边顶点v2，如上图圆圈2。对于顶点v1，它有一个入边顶点v2，所以它的firstin指向顶点v2的边表结点中headvex为1的结点，如上图圆圈3。

十字链表的好处就是因为把邻接表和逆邻接表整合在一起，这样既容易找到以v为尾的弧，也容易找到以v为头的弧，因而比较容易求得顶点的出度和入度。

而且除了结构复杂一点外，其实创建图**算法**的时间复杂度是和邻接表相同的，因此，在有向图应用中，十字链表是非常好的**数据结构**模型。

这里就介绍以上三种存储结构，除了第三种存储结构外，其他的两种存储结构比较简单

三：图的遍历

1：深度优先遍历（DFS）

它从图中某个结点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图，直至图中所有和v有路径相通的顶点都被访问到。若图中尚有顶点未被访问，则另选图中一个未曾被访问的顶点作起始点，重复上述过程，直至图中的所有顶点都被访问到为止。

基本实现思想：

（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，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。

递归实现
（1）访问顶点v；visited[v]=1；//算法执行前visited[n]=0
（2）w=顶点v的第一个邻接点；
（3）while（w存在）
        if（w未被访问）
                从顶点w出发递归执行该算法；
        w=顶点v的下一个邻接点；

非递归实现
（1）栈S初始化；visited[n]=0；
（2）访问顶点v；visited[v]=1；顶点v入栈S
（3）while(栈S非空)
        x=栈S的顶元素(不出栈)；
        if(存在并找到未被访问的x的邻接点w)
                访问w；visited[w]=1；
                w进栈;
        else
                x出栈；

2：广度优先遍历（BFS）

它是一个分层搜索的过程和二叉树的层次遍历十分相似，它也需要一个队列以保持遍历过的顶点顺序，以便按出队的顺序再去访问这些顶点的邻接顶点。

基本实现思想：
（1）顶点v入队列。
（2）当队列非空时则继续执行，否则算法结束。
（3）出队列取得队头顶点v；访问顶点v并标记顶点v已被访问。
（4）查找顶点v的第一个邻接顶点col。
（5）若v的邻接顶点col未被访问过的，则col入队列。
（6）继续查找顶点v的另一个新的邻接顶点col，转到步骤（5）。
    直到顶点v的所有未被访问过的邻接点处理完。转到步骤（2）。
广度优先遍历图是以顶点v为起始点，由近至远，依次访问和v有路径相通而且

路径长度为1，2，……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，需设置队列存储访问的顶点。

伪代码

（1）初始化队列Q；visited[n]=0；

（2）访问顶点v；visited[v]=1；顶点v入队列Q；

（3） while（队列Q非空）

    v=队列Q的对头元素出队；

    w=顶点v的第一个邻接点；

    while（w存在）

        如果w未访问，则访问顶点w；

        visited[w]=1；

        顶点w入队列Q；

        w=顶点v的下一个邻接点。

非强连通图：如果有向图中存在两个顶点之间是不连通的，则该有向图称为非强连通图