图的存储必须要完整、准确地反映顶点集和边集的信息。根据不同图的结构和算法，采用不同的存储方式将对程序的效率产生相当大的影响，因此所选的存储结构应适合于待求解的问题。

邻接矩阵法

所谓邻接矩阵存储，是指用一个一维数组存储图中顶点的信息，用一个二维数组存储图中边的信息（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。

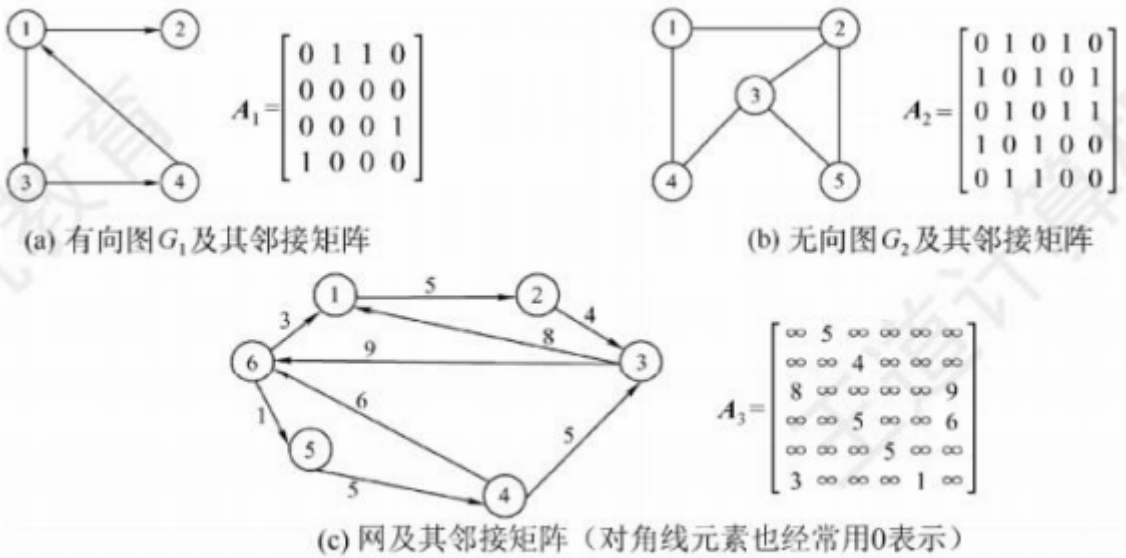
$$\text{顶点数为}n\text{的图}G=(V,E)\text{的邻接矩阵}A\text{是}n\times n\text{的，将}G\text{的顶点编号为}v_1,v_2,\ldots,v_n\text{，则}$$

$$A[i][j]=\begin{cases}1,(v_i,v_j)\text{或}<v_i,v_j>\text{是}E(G)\text{中的边}\\0,(v_i,v_j)\text{或}<v_i,v_j>\text{不是}E(G)\text{中的边}\end{cases}$$

对带权图而言，若顶点vi和vj之间有边相连，则邻接矩阵中对应存放着该边对应的权值，若顶点vi和vj不相连，则通常用0或∞来代表这两个顶点之间不存在边：

$$A[i][j]=\begin{cases}w_{ij},(v_i,v_j)\text{或}<v_i,v_j>\text{是}E(G)\text{中的边}\\0\text{或}\infty,(v_i,v_j)\text{或}<v_i,v_j>\text{不是}E(G)\text{中的边}\end{cases}$$

有向图、无向图对应的邻接矩阵示例如图所示。



(a) 有向图G_1及其邻接矩阵

(b) 无向图G_2及其邻接矩阵

(c) 网及其邻接矩阵（对角线元素也经常用0表示）

图 6.5　有向图、无向图及网的邻接矩阵

图的邻接矩阵存储结构定义如下：

```
#define MaxVertexNum 100//顶点数目的最大值
typedef char VertexType;//顶点对应的数据类型
typedef int EdgeType;//边对应的数据类型
typedef struct{
    VertexType vex[MaxVertexNum];//顶点表
    EdgeType edge[MaxVertexNum][MaxVertexNum];//邻接矩阵，边表
    int vexnum, arcnum;
}MGraph;
```

1. 在简单应用中，可直接用二维数组作为图的邻接矩阵（顶点信息等均可省略）。
2. 当邻接矩阵的元素仅表示相应边是否存在时，EdgeType可用值为0和1的枚举类型。

3. 无向图的邻接矩阵是对称矩阵，对规模大的邻接矩阵可采用压缩存储。

4. 邻接矩阵表示法的空间复杂度为O(n^2)，其中n为图的定点数|V|。

图的邻接矩阵存储表示法具有以下特点：

无向图的邻接矩阵一定是一个对称矩阵（并且唯一）。因此，在实际存储邻接矩阵时只需存储上（或下）三角矩阵的元素。
对于无向图，邻接矩阵的第i行（或第i列）非零元素（或非∞元素）的个数正好是顶点i的度TD(vi)。
对于有向图，邻接矩阵的第i行非零元素（或非∞元素）的个数正好是顶点i的出度OD(vi)；第i列非零元素（或非∞元素）的个数正好是顶点i的入度ID(vi)。
用邻接矩阵存储图，很容易确定图中任意两个顶点之间是否有边相连。但是，要确定图中有多少条边，则必须按行、列对每个元素进行检测，所花费的时间代价很大。
稠密图（即边数较多的图）适合采用邻接矩阵的存储表示。
设图G的邻接矩阵为A，A^n的元素A^n[i][j]等于由顶点i到顶点j的长度为n的路径的数目。

邻接表法

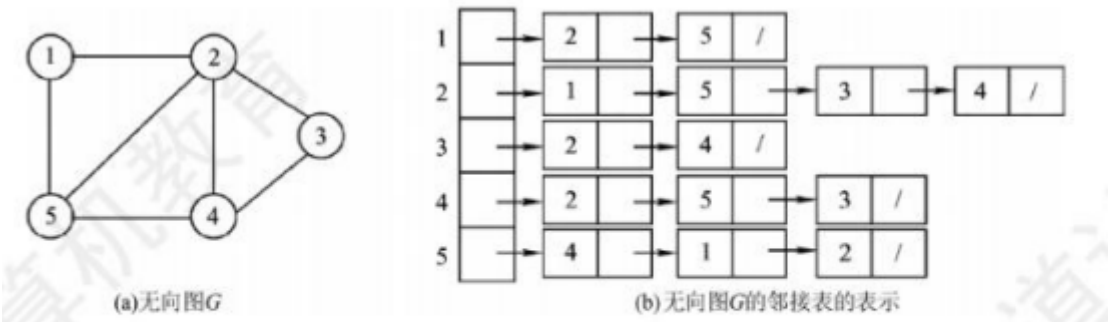
当一个图为稀疏图时，使用邻接矩阵法显然会浪费大量的存储空间，而图的邻接表法结合了顺序存储和链式存储方法，大大减少了这种不必要的浪费。

所谓邻接表，是指对图G中的每个顶点vi建立一个单链表，第i个单链表中的结点表示依附于顶点vi的边（对于有向图则是以顶点vi为尾的弧），这个单链表就称为顶点vi的边表（对于有向图则称为出边表）。边表的头指针和顶点的数据信息采用顺序存储，称为顶点表，所以在邻接表中存在两种节点：顶点表结点和边表结点，如图所示。

图 6.6　顶点表和边表结点结构

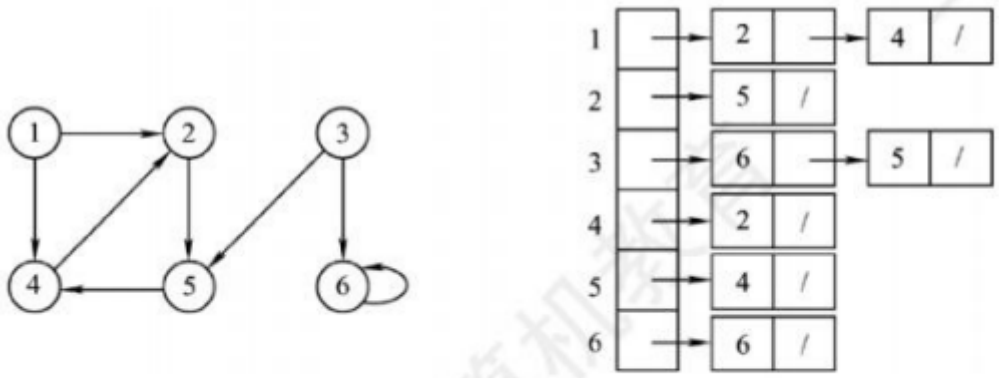
顶点表结点由两个域组成：顶点域（data）存储顶点vi的相关信息，边表头指针域（firstarc）指向第一条边的边表结点。边表结点至少由两个域组成：邻接点域（adjvex）存储与头节点顶点vi邻接的顶点编号，指针域（nextarc）指向下一条边的边表结点。

无向图和有向图的邻接表的实例分别如图所示。



(a)无向图G　　(b)无向图G的邻接表的表示

图 6.7　无向图邻接表表示法实例



(a)有向图 G　　(b)有向图 G 的邻接表的表示

图 6.8　有向图邻接表表示法实例

```
#define MaxVertexNum 100//图中顶点数目的最大值
typedef struct ArcNode{//边表结点
    int adjvex;//该弧所指向的顶点的位置
    struct ArcNode *nextarc;//指向下一条弧的指针
    //InfoType info;//网的边权值
}ArcNode;
typedef struct VNode{//顶点表结点
    VertexType data;//顶点信息
    ArcNode *firstarc;//指向第一条依附该顶点的弧的指针
}VNode,AdjList[MaxVertexNum];
typedef struct{
    AdjList vertices;//邻接表
    int vexnum, arcnum;//图的顶点数和弧数
}ALGraph;//ALGraph是以邻接表存储的图类型
```

图的邻接表存储方法具有以下特点：

若G为无向图，则所需的存储空间为O(|V|+2|E|)；若G为有向图，则所需的存储空间为O(|V|+|E|)。前者的倍数2是因为在无向图中，每条边在邻接表中出现了两次。
对于稀疏图（即边数较少的图），采用邻接表表示将极大地节省存储空间。

在邻接表中，给定一个顶点，能很容易地找出它的所有邻边，因为只需要读取它的邻接表。在邻接矩阵中，相同的操作则需要扫描一行，花费的时间为O(n)。但是，若要确定给定的两个顶点间是否存在边，则在邻接矩阵中可以立刻查到，而在邻接表中则需要在相应结点对应的边表中查找另一结点，效率较低。

在无向图的邻接表中，求某个顶点的度只需计算其邻接表中的边表结点个数。在有向图的邻接表中，求某个顶点的出度只需计算其邻接表中的边表结点个数；但求某个顶点x的入度则需遍历全部的邻接表，统计邻接点（adjvex）域为x的边表结点个数。

图的邻接表表示并不唯一，因为在每个顶点对应的边表中，各边结点的链接次序可以是任意的，它取决于建立邻接表的算法及边的输入次序。

十字链表

十字链表是有向图的一种链式存储结构。在十字链表中，有向图的每条弧用一个结点（弧结点）来表示，每个顶点也用一个结点（顶点结点）来表示。两种结点的结构如下所示。

弧结点中有5个域：tailvex和headvex两个域分别指示弧尾和弧头这两个顶点的编号；头链域hlink指向弧头相同的下一个弧结点；尾链域tlink指向弧尾相同的下一个弧结点；info域存放该弧的相关信息。这样，弧头相同的弧在同一个链表上，弧尾相同的弧也在同一个链表上。

顶点结点中有3个域；data域存放该顶点的数据信息，如顶点名称；firstin域指向以该顶点为弧头的第一个弧结点；firtout域指向以该顶点为弧尾的第一个弧结点。

下图为有向图的十字链表法。

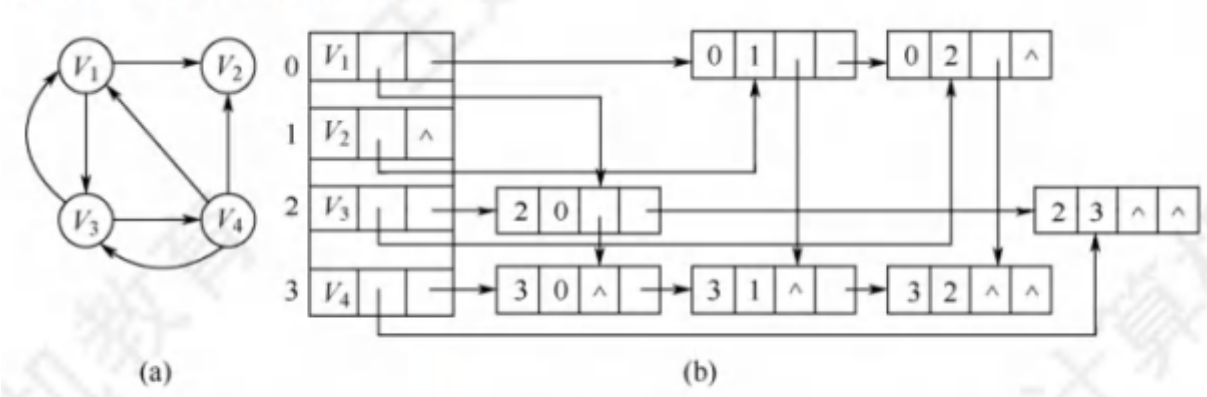


图 6.9　有向图的十字链表表示（弧结点省略 info 域）

注意，顶点结点之间是顺序存储的，弧结点省略了info域。

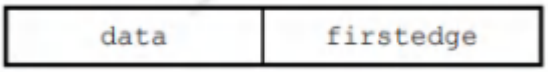
在十字链表中，既容易找到Vi为尾的弧，也容易找到Vi为头的弧，因而容易求得顶点的出度和入度。图的十字链表表示不是唯一的，但一个十字链表表示唯一确定一个图。

邻接多重表

邻接多重表是无向图的一种链式存储结构。在邻接表中，容易求得顶点和边的各种信息，但在邻接表中求两个顶点之间是否存在边而对边执行删除等操作时，需要分别在两个顶点的边表中遍历，效率极低。与十字链表类似，在邻接多重表中，每条边用一个结点表示，其结构如下。

其中，ivex和jvex这两个域指示该边依附的两个顶点的编号；ilink域指向下一条依附于顶点ivex的边；jlink域指向下一条依附于顶点jvex的边，info域存放该边的相关信息。

每个顶点也用一个结点表示，它由如下所示的两个域组成。



其中，data域存放该顶点的相关信息，firstedge域指向第一条依附于该顶点的边。

在邻接多重表中，所有依附于同一顶点的边串联在同一链表中，因为每条边依附于两个顶点，所以每个边结点同时链接在两个链表中。对于无向图而言，其邻接多重表和邻接表的差别仅在于，同一条边在邻接表中用两个结点表示，而在邻接多重表中只有一个结点。

下图为无向图的邻接多重表表示法。邻接多重表的各种基本操作的实现和邻接表类似。

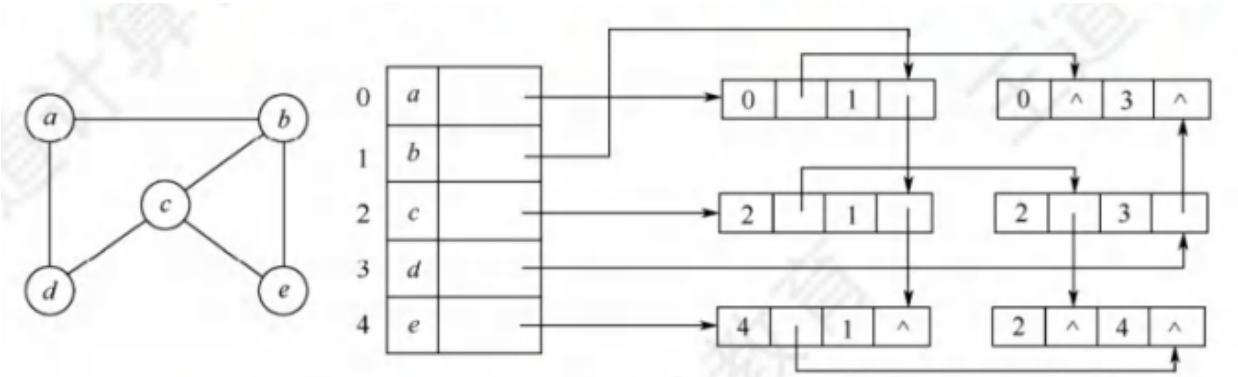


图 6.10　无向图的邻接多重表表示（边结点省略 info 域）

图的四种存储方式的总结如下表所示。

	邻接矩阵	邻接表	十字链表	邻接多重表
空间复杂度	O(|V|^2)	无向图: O(|V|+2|E|) 有向图: O(|V|+|E|)	O(|V|+|E|)	O(|V|+|E|)
找相邻边	遍历对应行或列的时间复杂度为O(|V|)	找有向图的入度必须遍历整个邻接表	很方便	很方便
删除边或顶点	删除边很方便，删除顶点需要大量移动数据	无向图中删除边或顶点都不方便	很方便	很方便

	邻接矩阵	邻接表	十字链表	邻接多重表
适用于	稠密图	稀疏图和其他	只能存有向图	只能存无向图
表示方式	唯一	不唯一	不唯一	不唯一

图的基本操作

图的基本操作是独立于图的存储结构的。而对于不同的存储方式，操作算法的具体实现会有着不同的性能。在设计具体算法的实现时，应考虑采用何种存储方式的算法效率会更高。

图的基本操作主要包括（仅抽象地考虑，所以忽略各变量的类型）：

Adjacent(G,x,y)	判断图是否存在边<x,y>或(x,y)。
Neighbors(G,x)	列出图G中与结点x邻接的边。
InsertVertex(G,x)	在图G中插入顶点x。
DeleteVertex(G,x)	从图G中删除顶点x。
AddEdge(G,x,y)	若无向边(x,y)或有向边<x,y>不存在，则向G中添加该边。
RemoveEdge(G,x,y)	若无向边(x,y)或有向边<x,y>存在，则从图G中删除该边。
FirstNeighbor(G,x)	求图G中顶点x的第一个邻接点，若有则返回顶点号。若x没有邻接点或图中不存在x，则返回-1。
NextNeighbor(G,x,y)	假设图G中顶点y是顶点x的一个邻接点，返回除y外顶点x的下一个邻接点的顶点号，若y是x的最后一个邻接点，则返回-1。
Get_edge_value(G,x,y)	获取图G中边(x,y)或<x,y>对应的权值。
Set_edge_value(G,x,y,v)	设置图G中边(x,y)或<x,y>对应的权值为v。

此外，还有图的遍历算法：按照某种方式访问图中的每个顶点且仅访问一次。图的遍历算法包括深度优先遍历和广度优先遍历。