图

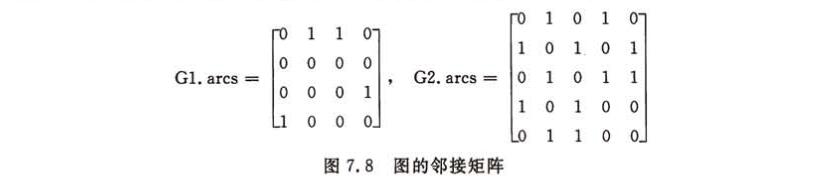
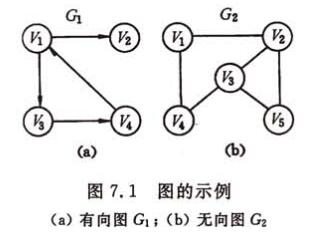
[TOC]

# # 1 图存储结构的定义

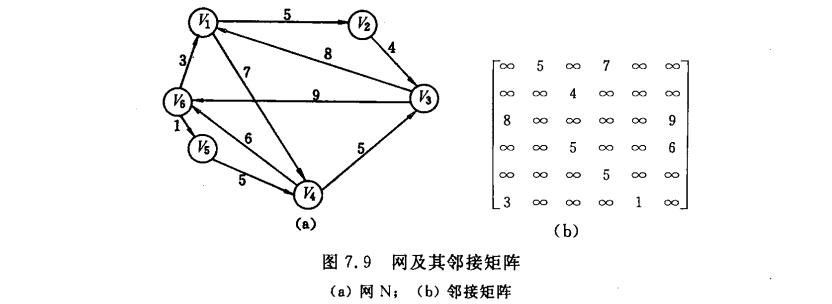
## ## 1.1 图的邻接矩阵定义法

用一个一维数组存储图中顶点（vertex）的信息，用一个二维数组存储图中边（弧arc）的信息（即各顶点之间的邻接关系）。存储顶点之间的邻接关系的二维数组称为**邻接矩阵**。

* 对于**有向图**，如果vi到vj有边，arc[vi][vj] = 1；如果vi到vj无边，arc[vi][vj] = 0。
* 对于**无向图**，如果vi到vj有边，arc[vi][vj] = arc[vj][vi] = 1；如果vi到vj无边，arc[vi][vj] = arc[vj][vi] = 0。



* 对于**带权图**（网），若顶点vi和vj之间有边相连，则邻接矩阵中对应项存放着对应的权值；若不相连，则用∞来表示两个顶点之间不存在的边。



#define MaxVertexNum 100

#define VertexType int

//## 1.1 图的邻接矩阵定义法

typedef struct{

VertexType vexs[MaxVertexNum]; //顶点信息

int arc[MaxVertexNum][MaxVertexNum]; //边（弧）信息

int vexnum,arcnum; //顶点数和边数

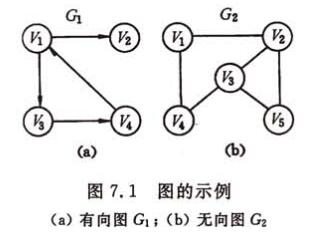
}MGraph; //以邻接矩阵存储的图类型

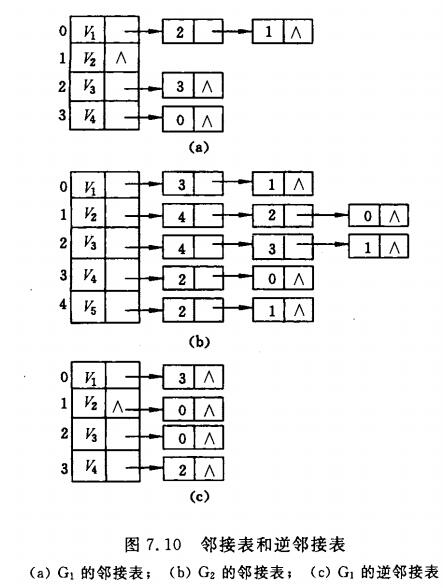
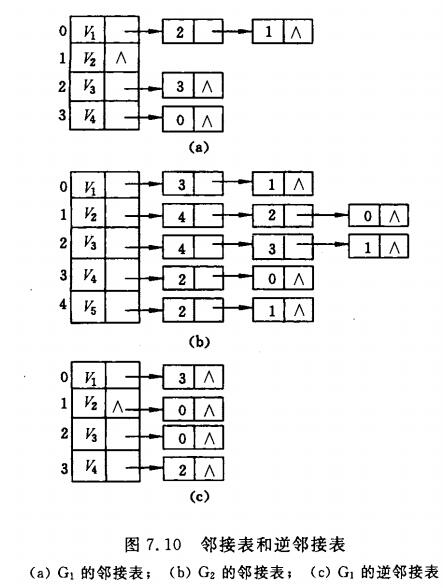
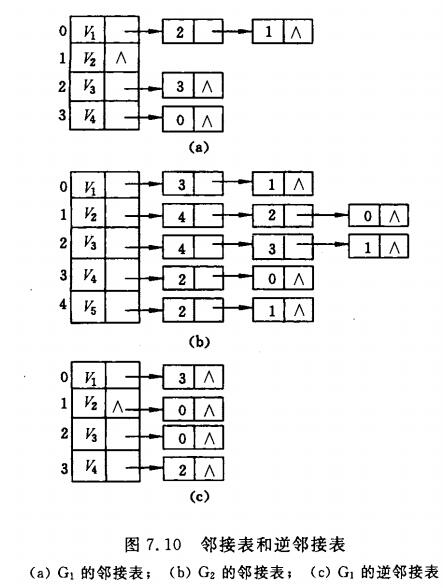
**图的邻接矩阵存储表示法的特点：**

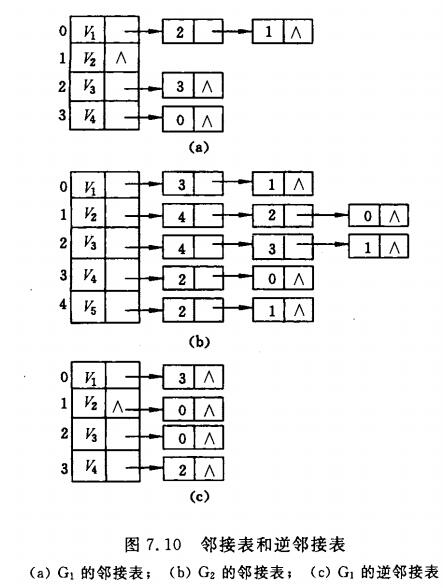
1. 无向图的邻接矩阵一定是个**对称矩阵**，并且唯一。因此，在实际存储邻接矩阵只需存储上（下）三角矩阵的元素即可。
2. 对于无向图，邻接矩阵的第i行（或第i列）非零元素的个数正好是第i个顶点的度。对于有向图，邻接矩阵的第i行（或第i列）非零元素的个数正好是第i个顶点的出度（或入度）。
3. **容易确定图中任意两个顶点是否有边**，但要确定图中有多少条边则按行（列）对每个元素检测，**时间开销很大**。
4. **空间复杂度为O(n^2)，**适合存储**稠密图。**
5. **非带权图的邻接矩阵n次方值，就是对应顶点间路径长度为n的路径条数。**

## ## 1.2 图的邻接表定义法

邻接表法是对图G中的每个顶点vi，将所有邻接于vi的顶点链成一个单链表，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(firstarc)和顶点信息采用顺序存储，称为**顶点表**。所以，在邻接表中存在两种结点：**顶点表（头）结点**和**边表结点**。







#define MaxVertexNum 100

#define VertexType int

#define WeightType double

//## 1.2 图的邻接表定义法

typedef struct ArcNode{ //弧（边表）结点

int adjvex; //弧所指顶点的位置（下标）

WeightType cost; //带权图的边权值(info)

struct ArcNode \*nextarc; //下一条弧指针

//ArcNode \*nextarc;//Successful

}ArcNode;

typedef struct{ //顶点表（顶点/头）结点

VertexType data; //顶点数据

ArcNode \*firstarc; //指向第一条依附该顶点的弧指针

}VNode,AdjList[MaxVertexNum];

typedef struct{

AdjList vertices; //邻接表

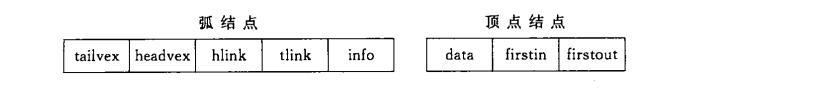
int vexnum,arcnum; //顶点数和边数

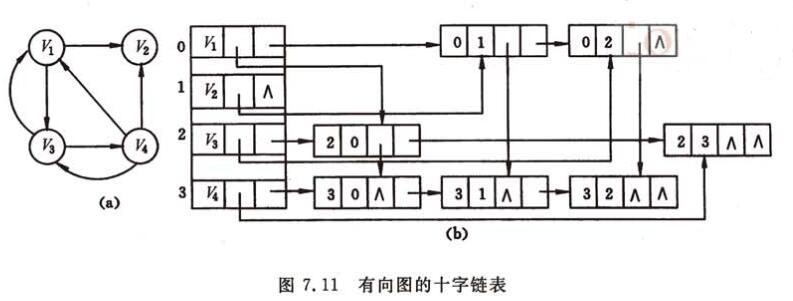
}ALGraph; //以邻接表存储的图类型

**图的邻接表存储表示法的特点：**

1. 对于图G=(V,E)，如果G为**无向图**，则所需存储空间为**O(|V|+2|E|)**；如果G为**有向图**，则所需存储空间为**O(|V|+|E|)**。
2. 对于**稀疏图**，采用邻接表表示将大大节省存储空间。
3. 求某个顶点的所有邻边，只需读取它的邻接表。其**效率O(ViE)**一般比邻接矩阵O(n)所实现的优。
4. 确定两个顶点是否有边，需在邻接表相应的边表中查找另一个结点。其**效率O(ViE)不如**邻接矩阵O(1)所实现的。
5. 在有向图的邻接表中，求某个顶点的**出度**只需计算其边表结点个数；但求其**入度**，则需要遍历全部边表，因此可以设计**逆邻接表**实现快速求其入度。
6. 图的邻接表**表示并不唯一，但可表示一个确定的图**。因为每个顶点的边结点链接次序可以是任意的，取决于建立邻接表的算法和边的输入次序。

## ## 1.3 有向图的十字链表定义法





#define MaxVertexNum 100

#define VertexType int

#define InfoType double

//## 1.3 有向图的十字链表定义法

typedef struct XArcNode{

int tailvex,headvex;

struct XArcNode \*hlink,\*tlink;

//XArcNode \*hlink,\*tlink;//Successful

//InfoType info;

}XArcNode;

typedef struct{

VertexType data;

XArcNode \*firstin,\*firstout;

}XVNode;

typedef struct{

XVNode xlist[MaxVertexNum];

int vernum,arcnum;

}XLGraph;

## ## 1.4 无向图的邻接多重表定义法

# # 2 图的遍历

## ## 2.1 深度优先搜索

## ## 2.2广度优先搜索

# # 3 图的基本应用

## ## 3.1 最小生成树

### ### 3.1.1 Prim算法

### ### 3.1.2 Kruskal算法

## ## 3.2 最短路径

### ### 3.2.1 Dijkstra算法

### ### 3.2.2 Floyd-Warshall算法

## ## 3.3 拓扑排序

## ## 3.4 关键路径

# # 4 图的实例