图的定义与操作

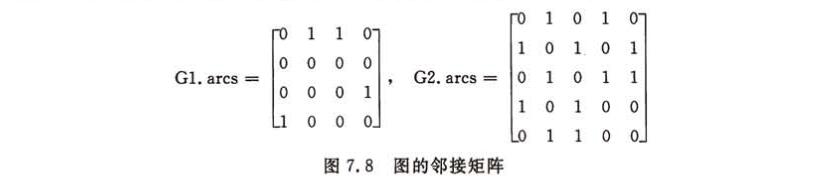
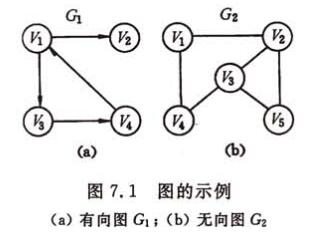
[TOC]

# # 1 图的各种存储结构的定义

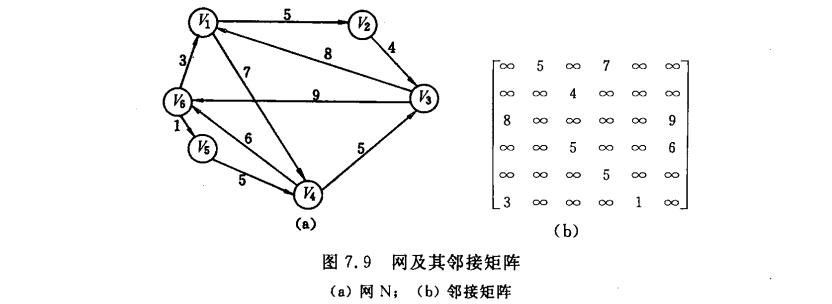
## ## 1.1 图的邻接矩阵定义法

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。



**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。**

#define MaxVertexNum 100

#define VertexType int

//## 1.1 图的邻接矩阵定义法

typedef struct{ //以邻接矩阵存储的图类型

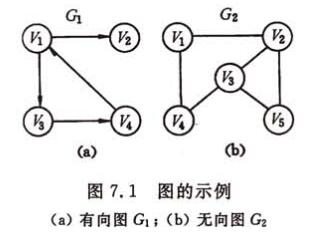
VertexType vexs[MaxVertexNum]; //顶点信息

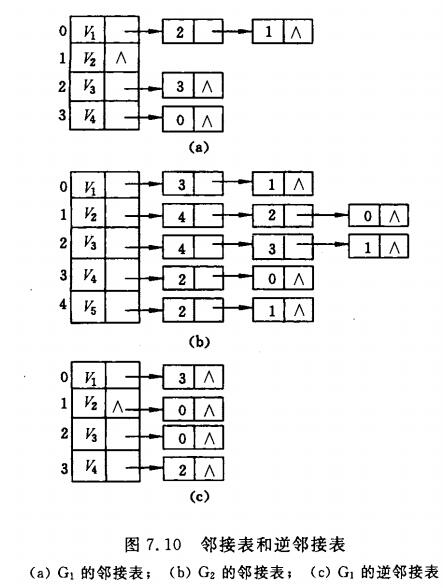
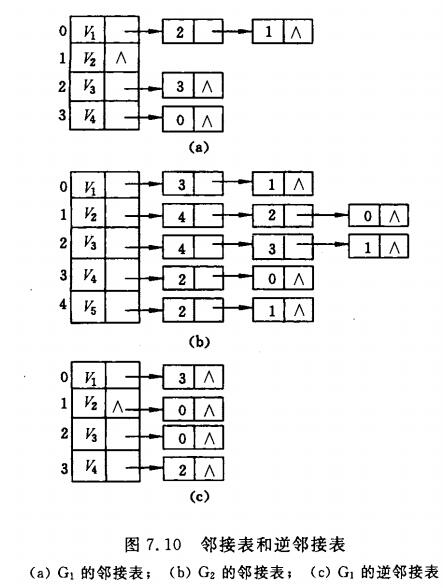
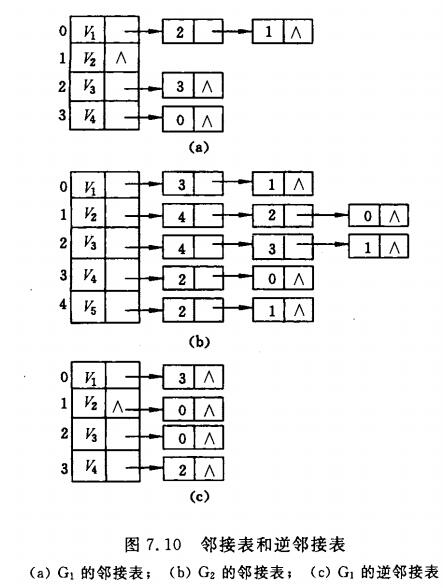
int arc[MaxVertexNum][MaxVertexNum]; //边（弧）信息

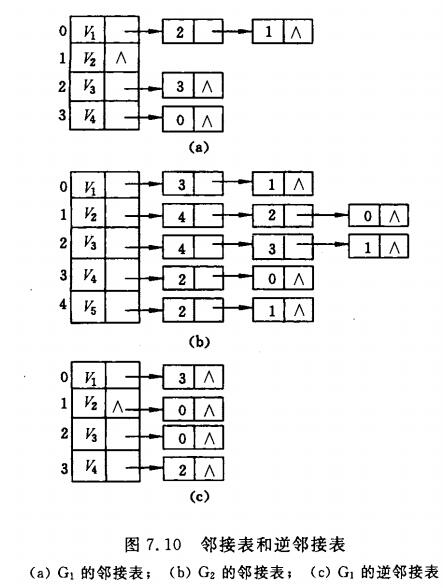
int vexnum,arcnum; //顶点数和边数

}MGraph;

## ## 1.2 图的邻接表定义法







#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 nodeindex; //弧所指顶点的位置（下标）

WeightType cost; //带权图中边权值

struct ArcNode \*nextarc; //下一条弧指针

//ArcNode \*nextarc;//Successful

}ArcNode;

typedef struct{ //顶点表（顶点）结点

VertexType data; //顶点数据

ArcNode \*firstarc; //

}VNode,AdjList[MaxVertexNum];

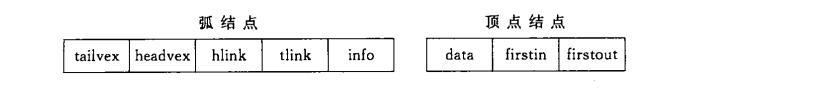
typedef struct{ //以邻接表存储的图类型

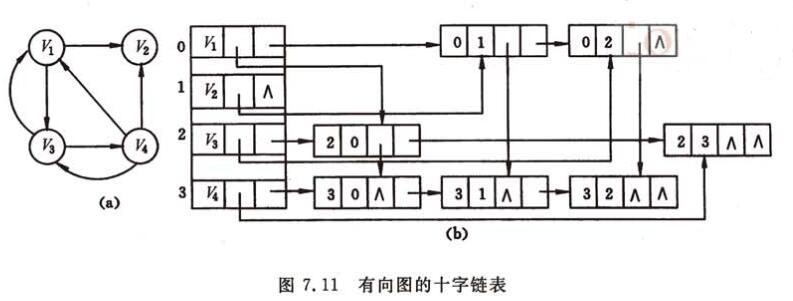
AdjList vertices; //邻接表

int vexnum,arcnum;

}ALGraph;

## ## 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