7.2 图的存储结构

7.2.1 邻接矩阵

邻接矩阵是表示顶点之间相邻关系的矩阵。设G=(V,E)是具有n个顶点的图，顶点编号依次为0、1、…、n-1，则G的邻接矩阵是具有如下定义的n阶方阵A。

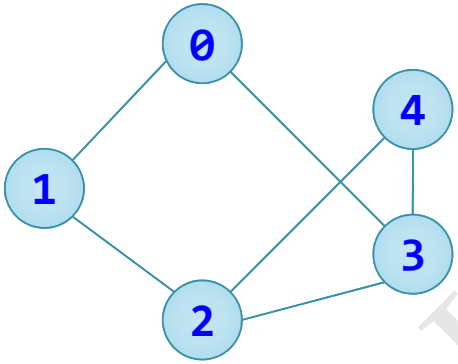
若G是不带权的图：

$$A[i][j]=\begin{cases}1 & \text{对于无向图，}(i,j)\text{或}(j,i)\in E(G);\\ & \text{对于有向图，}\langle i,j\rangle\in E(G)\\ 0 & i=j\\ 0 & \text{其他情况}\end{cases}$$

若G是带权图或网，则邻接矩阵可定义为（其中w_{ij}为边(i,j)或$<i,j>$的权）：

$$A[i][j]=\begin{cases} w_{ij} & \text{对于无向图，}(i,j)\text{或}(j,i)\in E(G)\text{；对于有向图，}<i,j>\in E(G) \\ 0 & i=j \\ \infty & \text{其他情况} \end{cases}$$

示例

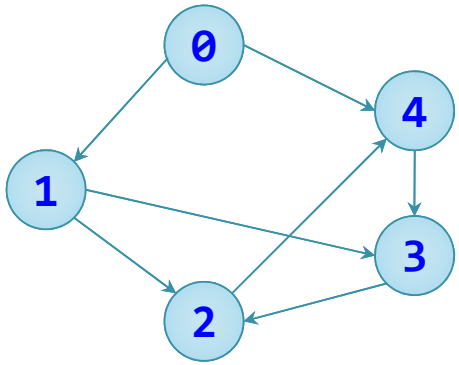


A_1=

	0	1	2	3	4
0	0	1	0	1	0
1	1	0	1	0	0
2	0	1	0	1	1
3	1	0	1	0	1
4	0	0	1	1	0

顶点1的度为2

无向图时一定为对称矩阵



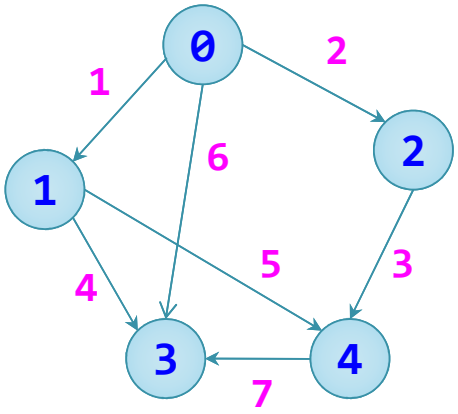
A_2=

	0	1	2	3	4
0	0	1	0	0	1
1	0	0	1	1	0
2	0	0	0	0	1
3	0	0	1	0	0
4	0	0	0	1	0

顶点1的出度为2

顶点1的入度为1

有向图时不一定为对称矩阵



A_3=

	0	1	2	3	4
0	0	1	2	6	∞
1	∞	0	∞	4	5
2	∞	∞	0	∞	3
3	∞	∞	∞	0	∞
4	∞	∞	∞	7	0

顶点1的出度为2

顶点1的入度为1

有向图时不一定为对称矩阵

图的邻接矩阵具有这样的特点：

- 对于n个顶点e条边的图采用邻接矩阵存储时占用存储空间为$O(n^2)$，与边数e无关（不考虑压缩存储），特别适合存储稠密图；
- 任何图的邻接矩阵表示是唯一的；
- 图采用邻接矩阵存储时判断两个顶点i、j之间是否有边十分容易。

图的邻接矩阵类型声明：

```
// ----- 图的数组(邻接矩阵)存储表示 --------
#define INFINITY INT_MAX                    // 用整型最大值代替∞
#define MAX_VERTEX_NUM 30                   // 最大顶点个数
enum GraphKind {DG, DN, UDG, UDN};          // {有向图,有向网,无向图,无向网}
#define MAX_NAME 4                          // 顶点字符串的最大长度+1
typedef int VRType;
typedef char VertexType[MAX_NAME];

typedef struct {
    VRType adj; // 顶点关系类型。对无权图，用1或0表示相邻否；对带权图，则为权值
    InfoType *info; // 该弧相关信息的指针(可无)
} ArcCell, AdjMatrix[MAX_VERTEX_NUM][MAX_VERTEX_NUM]; // 二维数组

typedef struct Graph {
    VertexType vexs[MAX_VERTEX_NUM]; // 顶点向量
    AdjMatrix arcs;          // 邻接矩阵
    int vexnum, arcnum;  // 图的当前顶点数和弧数
    GraphKind kind;      // 图的种类标志
} MGraph;
```

图的邻接矩阵类型声明：

```
#define INF    INT_MAX          // 用整型最大值代替∞
#define MAXVEX 30               //图中最大顶点个数
typedef char VertexType[4];
                                //定义VertexType为字符串类型
typedef struct vertex
{
   int adjvex;                  //顶点编号
   VertexType data;             //顶点的信息
} VType;                        //顶点类型

typedef struct graph
{  int n, e;                    //n为实际顶点数,e为实际边数
   VType vexs[MAXVEX];          //顶点集合
   int edges[MAXVEX][MAXVEX]; //边的集合
} MGraph;                       //图的邻接矩阵类型
```

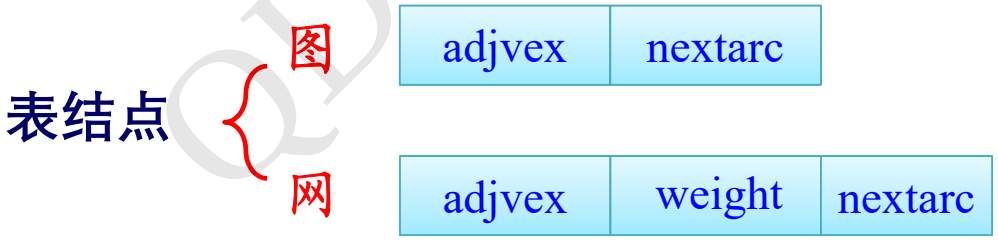
7.2.2 邻接表

- 邻接表是图的一种链式存储结构。
- 在邻接表中，对图中每个顶点建立一个单链表，把该顶点的所有相邻点串起来。
- 所有的头指针构成一个数组，称为表头结点数组，用**adjlist**表示，第**i**个单链表**adjlist[i]**中的结点表示依附于顶点**i**的边，也就是说头指针数组元素的下标与顶点编号一致。
- 表头结点的设置如下：

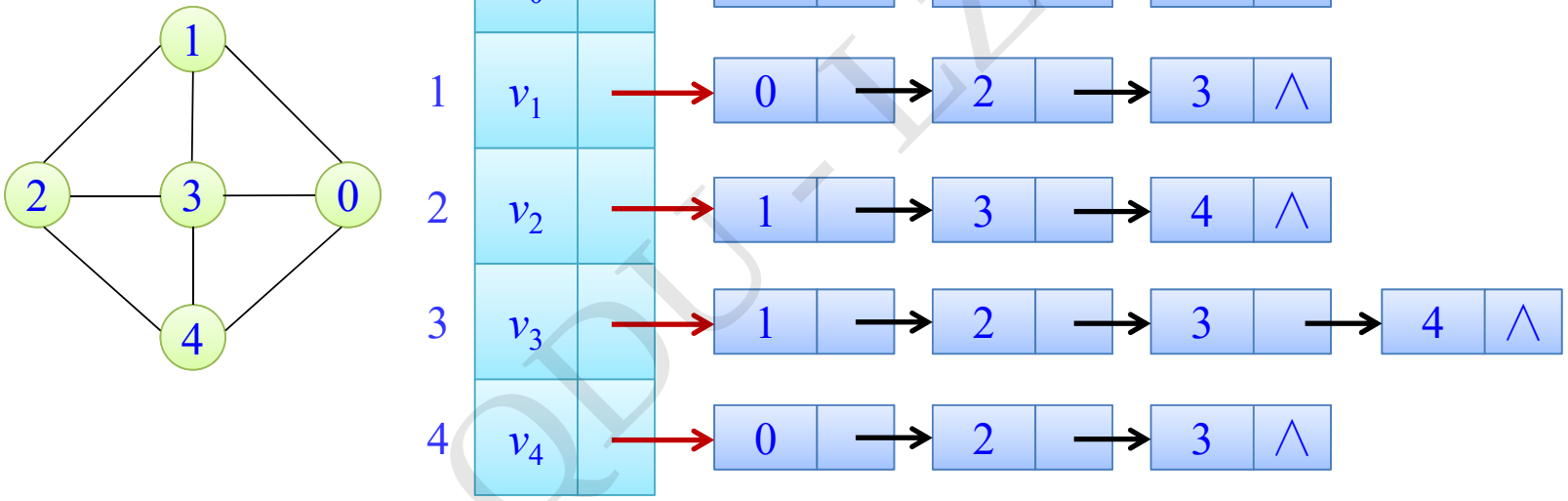
单链表中的每个结点由3个域组成：

- ✓ 顶点域adjvex（用以指示该相邻点在头结点数组中的下标）
- ✓ 权值域weight（存放对应边的权值）
- ✓ 指针域nextarc（用以指向依附于顶点i的下一条边所对应的结点）

- 对于不带权的图，weight域可以不设置；对于带权图，weight域置为相应边的权值。

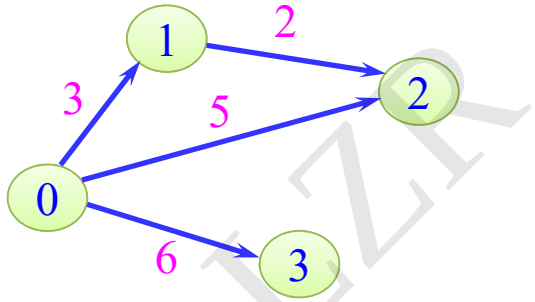


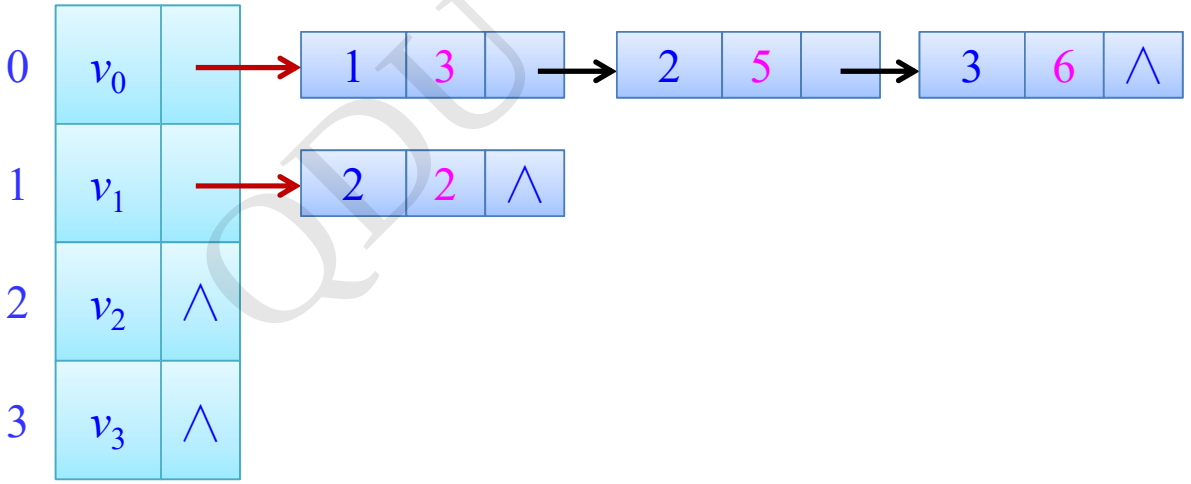
➢ 无向图的邻接表示意图



头结点　　边结点

➢ 有向图的邻接表示意图



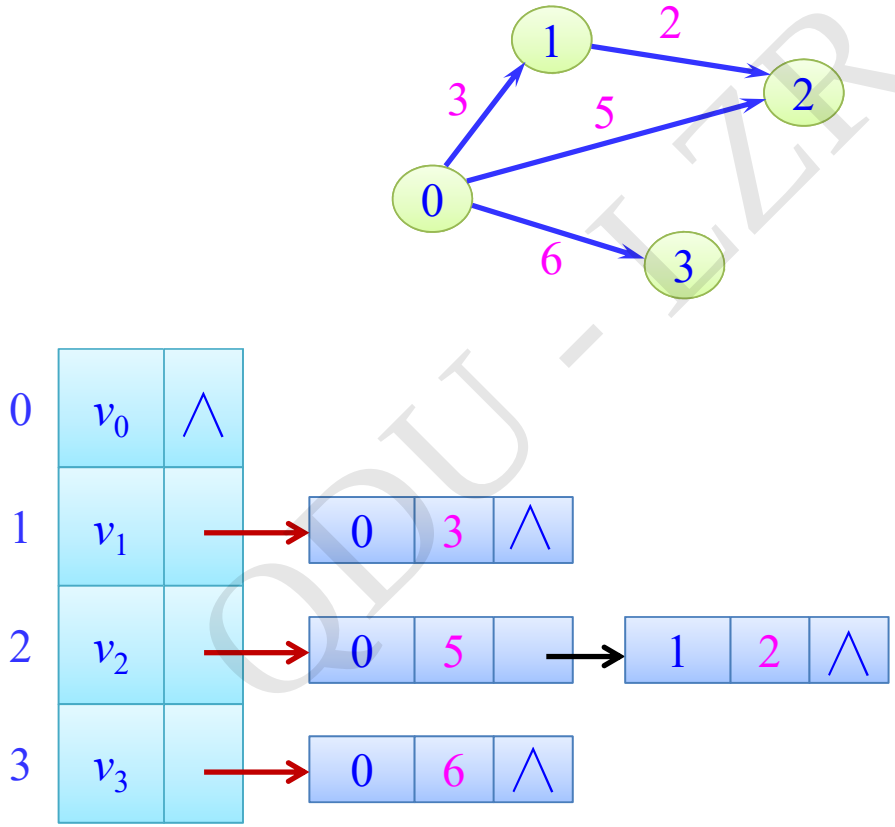


有向图的逆邻接表

- 在无向图的邻接表中，顶点v_i的度恰为第i个链表中的结点数。
- 而在有向图中，第i个链表中的结点个数只是顶点v_i的出度，为求入度，必须遍历整个邻接表。在所有链表中其邻接点域的值为i的结点的个数是顶点v_i的入度。

◆ 有时，为了便于确定顶点的入度或以顶点v_i为头的弧，可以建立一个有向图的逆邻接表，即对每个顶点v_i建立一个链接以v_i为头的弧的表。

➢ 有向图的逆邻接表示意图



图的邻接表具有这样的特点：

- 对于n个顶点e条边的图采用邻接表存储时占用存储空间为$O(n+e)$，与边数e有关，特别适合存储稀疏图；
- 图的邻接表表示不一定是唯一的，这是因为邻接表的每个单链表中，各结点的顺序是任意的；
- 图采用邻接表存储时查找一个顶点的所有相邻顶点十分容易。

一个图的邻接表存储结构的类型声明如下：

```
typedef char VertexType[10];    //VertexType为字符串类型
typedef struct edgenode
{   int adjvex;                 //相邻点序号
    int weight;                 //边的权值
    struct edgenode *nextarc;   //下一条边的顶点
} ArcNode;              //每个顶点建立的单链表中边结点的类型
typedef struct vexnode
{   VertexType data;            //存放一个顶点的信息
    ArcNode *firstarc;          //指向第一条边结点
} VHeadNode;                    //单链表的头结点类型
typedef struct
{   int n,e;                    //n为实际顶点数,e为实际边数
    VHeadNode adjlist[MAXVEX];  //单链表头结点数组
} ALGraph;                      //图的邻接表类型
```

END