

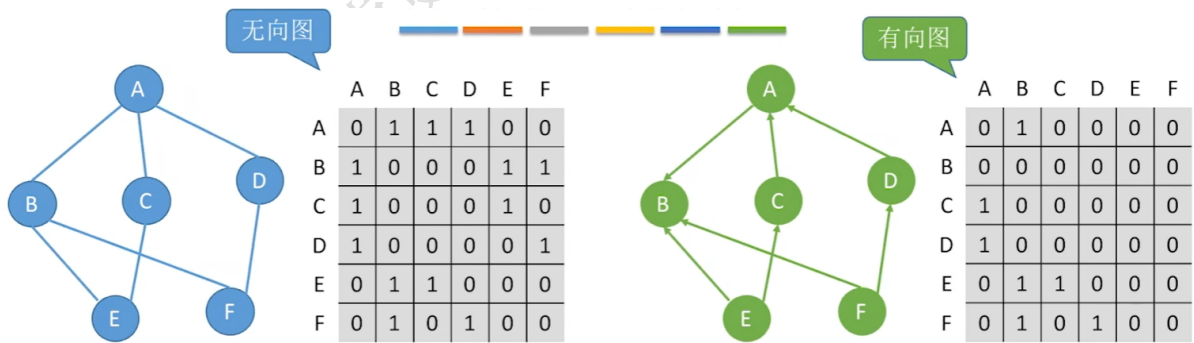
Lecture 21: 图的存储及其操作

21.1 邻接矩阵法

21.1.1 邻接矩阵

基本概念

- 定义:
 - 结点数为 n 的图 $G = (V, E)$ 的邻接矩阵 A 是 $n \times n$ 的。将 G 的顶点编号为 $v_1, v_2, \dots, v_n$, 则:
$$A[i][j] = \begin{cases} 1, & \text{若 } (v_i, v_j) \text{ 或 } \langle v_i, v_j \rangle \text{ 是 } E(G) \text{ 中的边} \\ 0, & \text{若 } (v_i, v_j) \text{ 或 } \langle v_i, v_j \rangle \text{ 不是 } E(G) \text{ 中的边} \end{cases}$$
- 性能:
 - 空间复杂度: $O(|V|^2)$
 - 只和顶点数相关, 和实际的边数无关;
- 特点:
 - 优点:
 - 适合于**存储稠密图**;
 - 无向图的邻接矩阵是对称矩阵, 可以压缩存储 (只存储上三角区/下三角区);
 - 缺点:
 - 空间复杂度高, **不适合存储稀疏图**;
- 图示:



- 总结:

知识: 邻接链表

结点连接的结点

行中非0元素数为出度

列中非0元素数为入度

简单图

对称性

4x4

结点数

边数

定义:

```
typedef struct Graph
{
    Vertex vex[MaxVertexMax];
    EdgeType edge[MaxVertexMax][MaxVertexMax];
    int vertexNum;//顶点数;
    int edgeNum;//边数;
}
```

- 第 i 个结点的度 = 第 i 行 (或第 i 列) 的 **非零元素个数**

有向图

- 第 i 个结点的 **出度** = 第 i 行的非零元素个数
- 第 i 个结点的 **入度** = 第 i 列的非零元素个数
- 第 i 个结点的度 = 第 i 行, 第 i 列的 **非零元素个数之和**
- 邻接矩阵法 **求顶点的度/出度/入度** 的事件复杂度为 $O(|V|)$

带权值

- 带权图只需要将其中值变为权值;

邻接矩阵法的性质

- 设图 G 的邻接矩阵为 A (矩阵元素为 0/1), 则 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;

计算举例

- 计算:
 - $A^2[1][4] = a_{1,1}a_{1,4} + a_{1,2}a_{2,4} + a_{1,3}a_{3,4} + a_{1,4}a_{4,4} = 1$
 - 表示: 从 A 到 D 有一条长度为 2 的路径;
- 图示:

	A	B	C	D
A	0	1	0	0
B	1	0	1	1
C	0	1	0	1
D	0	1	1	0

21.1.2 代码实现

结构定义

```
#define MaxVertexNum 100           //顶点数目的最大值
typedef struct{
    char Vex[MaxVertexNum];        //顶点表(可以存更复杂的信息)
    int Edge[MaxVertexNum][MaxVertexNum]; //邻接矩阵, 边表
    int vexnum, arcnum;             //图的当前顶点数和边数/弧数
}MGraph;
```

```
#define MaxVertexNum 100           //顶点数目的最大值
#define INFINITY 最大的int指      //宏定义常量"无穷"
```

```
typedef char VertexType;           //顶点的数据类型
typedef int EdgeType;             //带权图中边上权值的数据类型
typedef struct{
    VertexType Vex[MaxVertexNum]; //顶点
    EdgeType Edge[MaxVertexNum][MaxVertexNum]; //边的权
    int vexnum, arcnum;           //图的当前顶点数和弧数
}MGraph;
```

21.2 邻接表法

21.2.1 图的存储

基本概念

- 图示:



邻接表法

- 概念
 - 图 G 中每一个顶点建立一个单链表 v_i , 第 i 个单链表中的结点表示依附于顶点 v_i 的边;
 - 因此单链表称之为 v_i 顶点的**边表**;
- 结构
 - 图
 - 存储顶点构成的顺序表;
 - 顶点
 - 存储顶点的值+指向的第一条边;
 - 边
 - 存储指向的顶点名+指向下一条边的指针;
- 无向图
 - 边结点的数量是 $2|E|$ 整体空间复杂度为: $O(|V| + 2|E|)$
 - 获得结点的度: 遍历当前结点的链表, 其数量即为度;
- 有向图
 - 边结点的数量是 $|E|$ 整体空间复杂度为 $O(|V| + |E|)$
 - 把所有边依次遍历一遍, 知道入度;
- 缺陷
 - 每条边对应两份冗余信息, 删除顶点, 删除边等操作事件复杂度高
- 空间复杂度

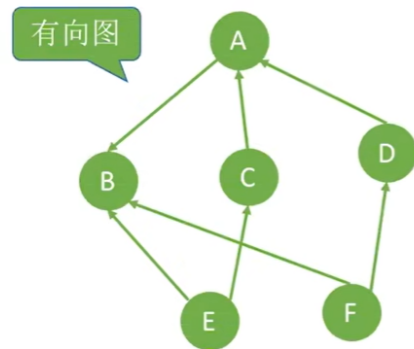
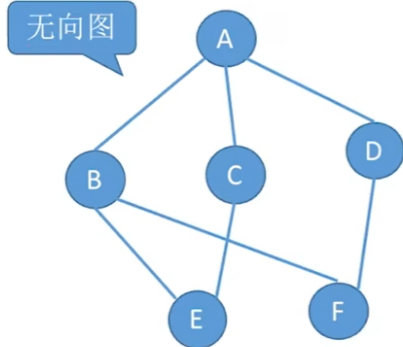
- 无向图:

$$O(|V| + 2|E|)$$

- 有向图:

$$O(|V| + |E|)$$

- 顺序+链式存储:

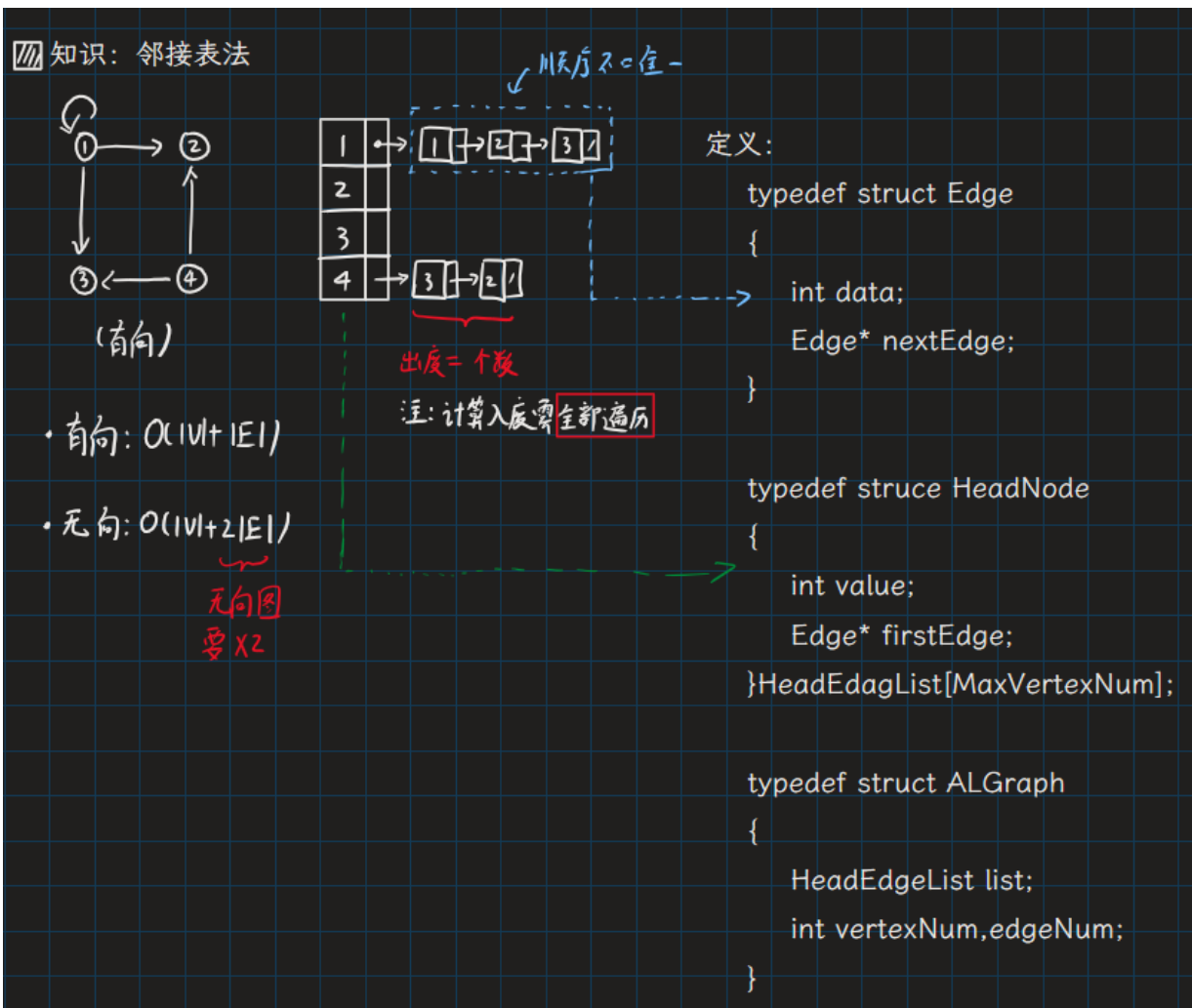


	data	*first			
0	A		→	1	→ 2 → 3 ^
1	B		→	0	→ 4 → 5 ^
2	C		→	0	→ 4 ^
3	D		→	0	→ 5 ^
4	E		→	1	→ 2 ^
5	F		→	1	→ 3 ^

	data	*first			
0	A		→	1	^
1	B	^			
2	C		→	0	^
3	D		→	0	^
4	E		→	1	→ 2 ^
5	F		→	1	→ 3 ^

- 总结:

获取笔记持续更新中...



21.2.2 代码实现

```
#define MaxVertexNum 100 //图中顶点数目的最大图
//用邻接表法存储的图
typedef struct{
    AdjList vertices; //邻接表
    int vexnum,arcnum; //图的顶点数和弧数
}ALGraph;

//顶点
typedef struct VNode{
    VertexType data; //顶点信息
    ArcNode *first; //第一条边/弧
}VNode,AdjList[MaxVertexNum];

//边/弧
typedef struct ArcNode{
    int adjvex; //边/弧指向哪个结点
    struct ArcNode *next; //指向下一条弧的指针
    //InfoType info //边权值
}ArcNode;
```

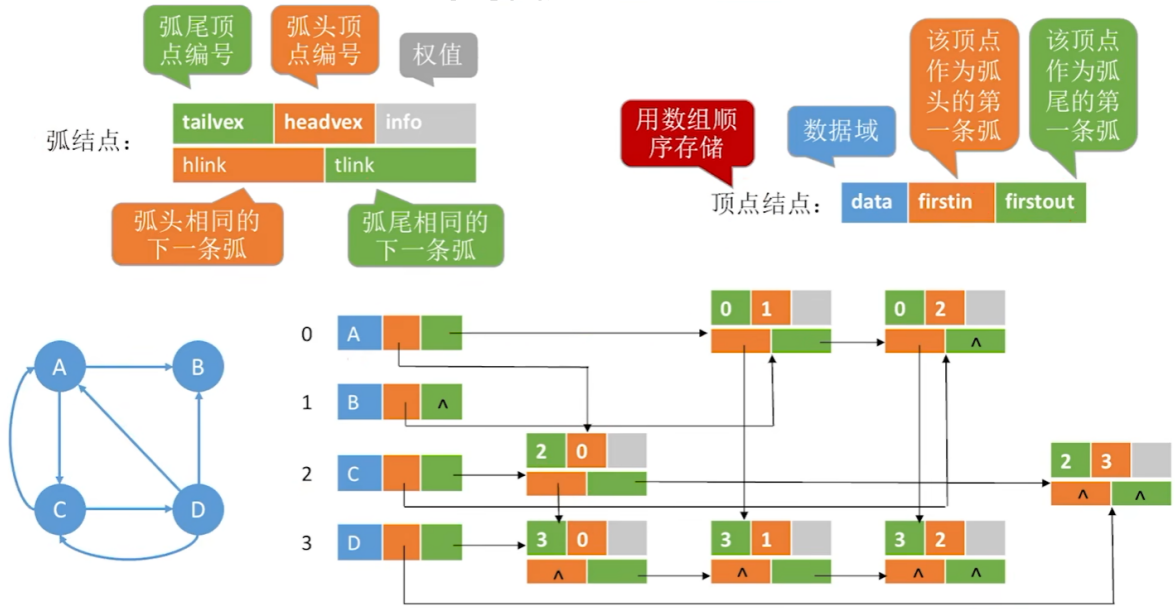
21.3 十字链表与邻接多重表

21.3.1 十字链表

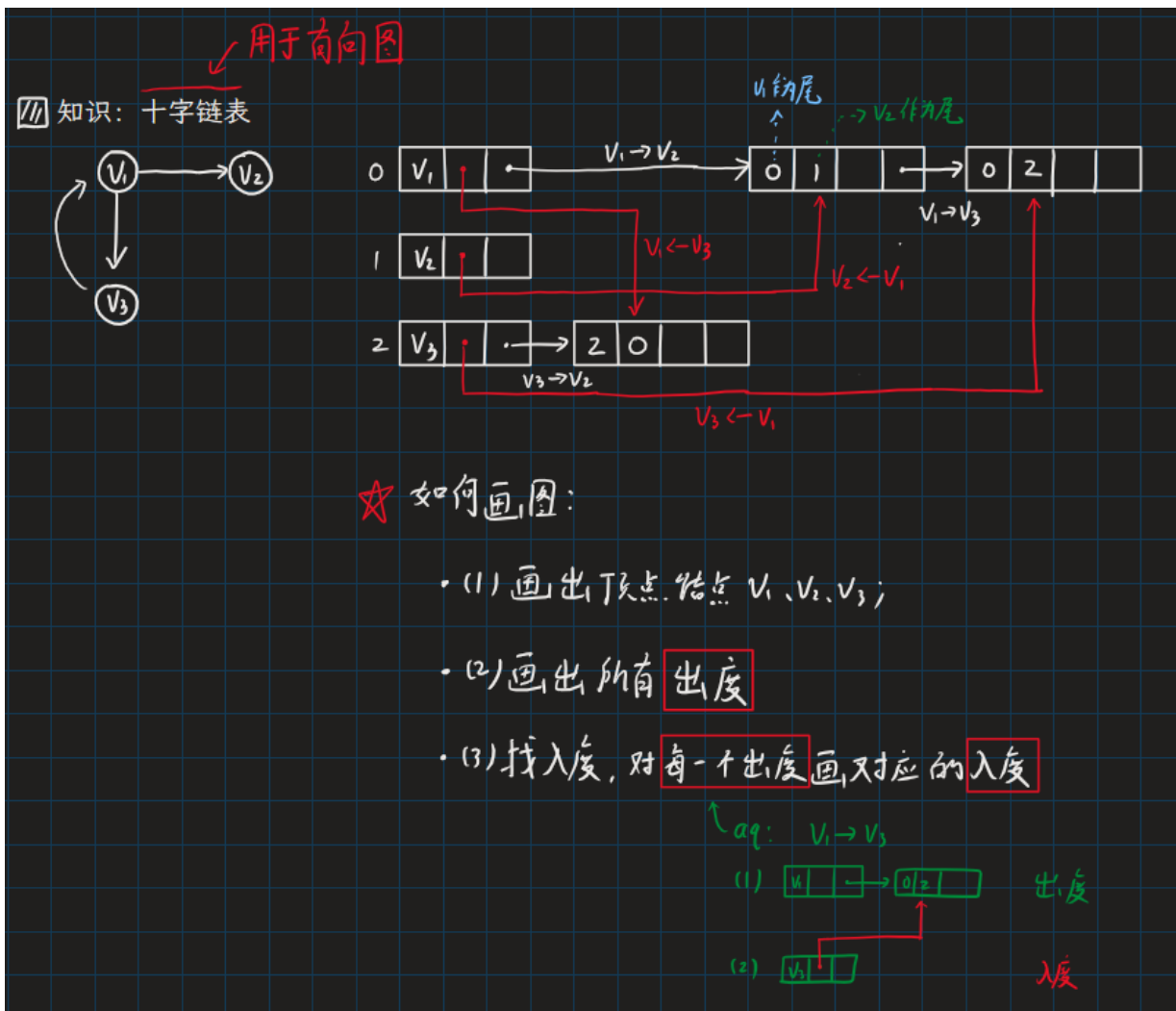
概念：十字链表

- 概念：
 - 是有向图的一种链式存储方式；
- 结构：
 - 顶点结点
 - data：存放该顶点的数据信息，如顶点名称；
 - firstin：指向以该顶点为弧头的第一个弧结点；
 - firstout：指向以该顶点为弧尾的第一个弧结点；
 - 顶点节点之间是顺序存储的；
 - 弧结点
 - tailvex：指向弧尾顶点；
 - headvex：指向弧头顶点；
 - hlink：指向弧头相同的下一个弧结点；
 - tlink：指向弧尾相同的下一个弧结点；
 - info：存放该弧的相关信息；
- 空间复杂度：
 - $O(|V| + |E|)$

图示：



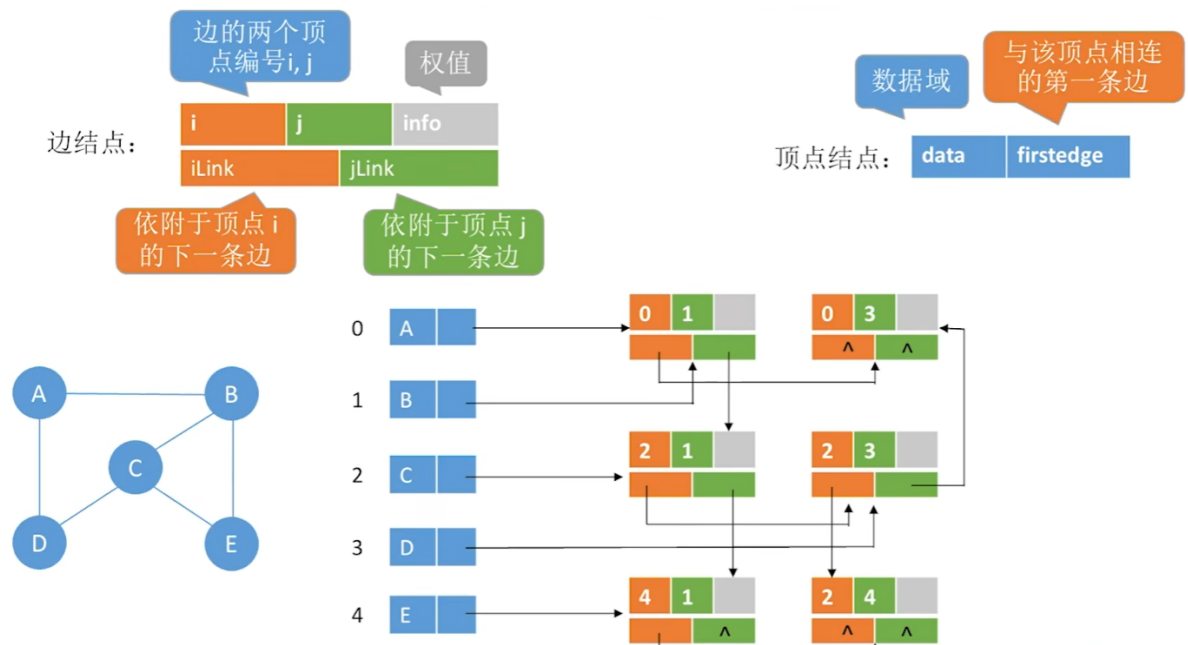
总结：



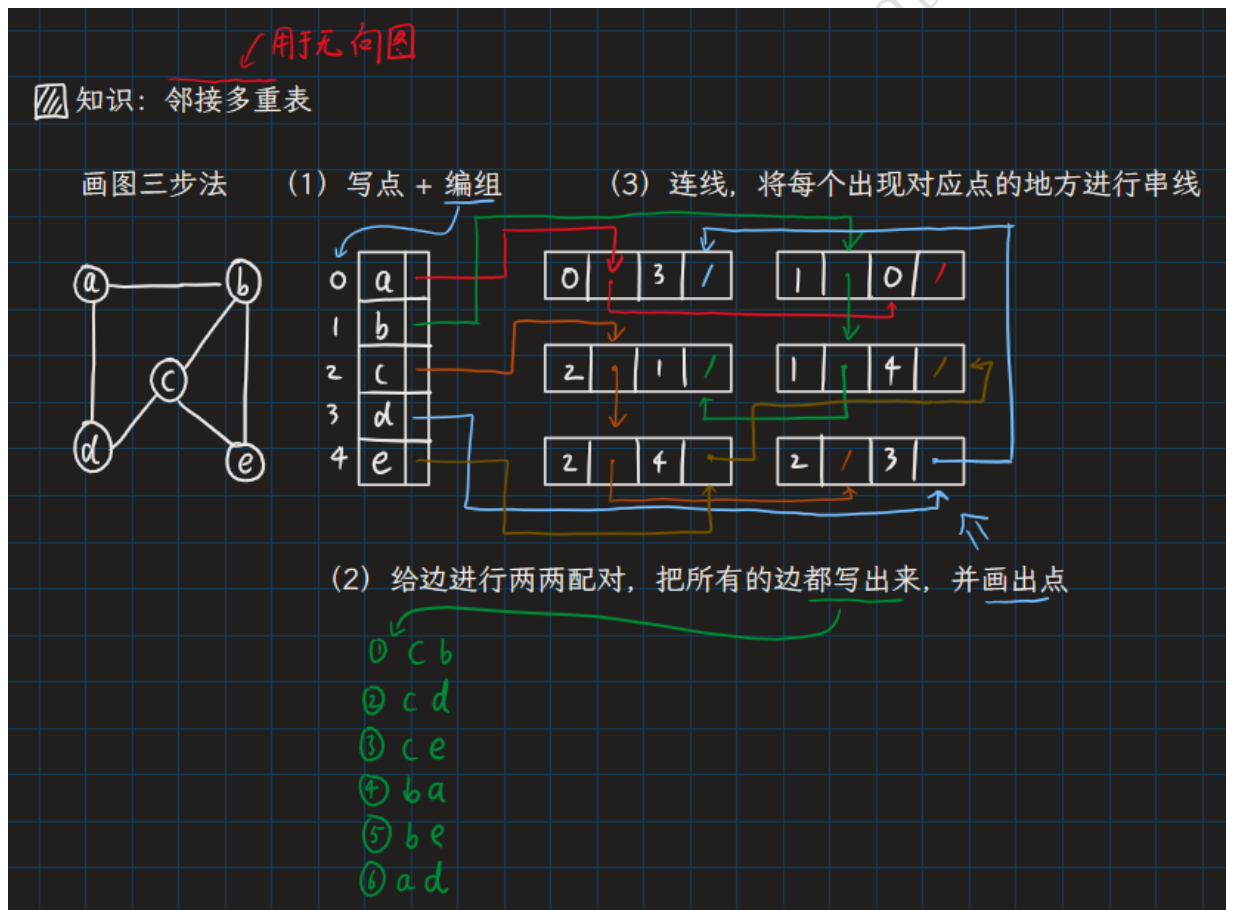
21.3.2 邻接多重表

概念：邻接多重表

- 概念：
 - 是无向图的一种链式存储方式；
- 空间复杂度
 - $O(|V| + |E|)$
- 优点：
 - 删除边，删除结点等操作很方便
- 注意：
 - 邻接多重表只适用于存储无向图
- 图示：



总结:



21.3.3 总结

	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 时间复杂度为 $O(V)$	找有向图的入边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
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

21.4 图的基本操作

图的基本操作

- `Adjacent(G, x, y)`: 判断图 G 是否存在边 $\langle x, y \rangle$ 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 $\langle x, y \rangle$ 不存在，则向图 G 中删除该边
- `RemoveEdge(G, x, y)`: 若无向边 (x, y) 或有向边 $\langle x, y \rangle$ 存在，则从图 G 中删除该边。
- `FirstNeighbor(G, x)`: 求图中顶点 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 顶点 y 是顶点 x 的一个邻接点，返回除 y 之外顶点 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 $\langle x, y \rangle$ 对应的权值。
- `Set_edge_value(G, x, y, v)`: 设置图 G 中边 (x, y) 或 $\langle x, y \rangle$ 对应的权值为 v 。

Adjacent (G, x, y)

- 判断图 G 是否存在边 $\langle x, y \rangle$ 或 (x, y)
- 邻接矩阵（有+无向图）：
 - $O(1)$
- 邻接表（有+无向图）：
 - $O(|V|)$

Neighbors (G, x)

- 列出图 G 中与结点 x 邻接的边
- 邻接矩阵（无向图）：
 - $O(|V|)$
- 邻接表（无向图）：
 - $O(1)$ 至 $O(|V|)$
- 邻接矩阵（有向图）
 - $O(|V|)$
- 邻接表（有向图）

- 出边: $O(1)$ 至 $O(|V|)$
- 入边: $O(|E|)$

InsertVertex

- 在图 G 中插入顶点 x
- 邻接矩阵 (无向图)
 - $O(1)$
 - 图示:

	data		A	B	C	D	E	F	X
0	A	A	0	1	1	1	0	0	0
1	B	B	1	0	0	0	1	1	0
2	C	C	1	0	0	0	1	0	0
3	D	D	1	0	0	0	0	1	0
4	E	E	0	1	1	0	0	0	0
5	F	F	0	1	0	1	0	0	0
6	X	X	0	0	0	0	0	0	0

- 邻接表 (无向图)
 - $O(1)$

DeleteVertex (G, x)

- 从图 G 中删除顶点 x
- 邻接矩阵 (有向+无向)
 - $O(|V|)$
- 邻接表 (无向)
 - $O(1)$ 至 $O(|V|)$
- 邻接表 (有向)
 - 出边: $O(1)$ 至 $O(|V|)$
 - 入边: $O(|E|)$

AddEdge (G, x, y)

- 若无向边 (x, y) 或有向边 $\langle x, y \rangle$ 不存在, 则向图 G 中添加该边
- 都是 $O(1)$