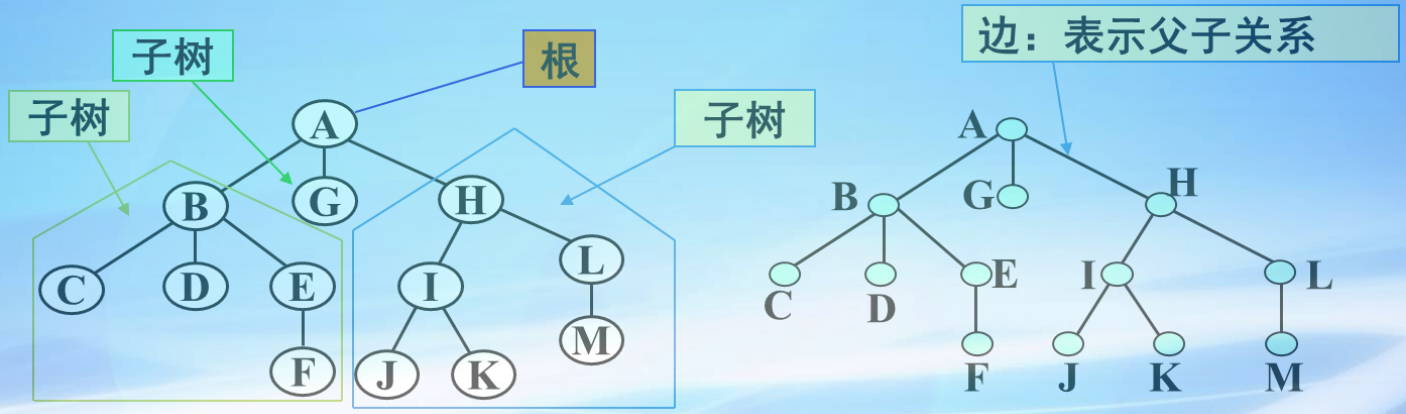
图的存储方式



学习目标和要求

1. 准确掌握图的存储方法，包括：邻接矩阵顺序存储、邻接表的存储方法与实现。

图的存储方式

- 邻接矩阵的顺序存储（邻接数组）
- 邻接矩阵的链式存储（邻接表）

图的存储方式

号名对照表

1. 由号查名

对顶点名进行编号0，1，2……n−1

顶点编号（下标）	0	1	2	…	…	n−1
name	北京	南京	上海	…	…	…
其他信息域						

号名对照表

2．由名查号

由顶点名（字符串） 查找其编号的方法：

（1）若按顶点名排序——二分法查找（快）

顶点编号（下标）	0	1	2	…	…	n−1
name	北京	南京	上海	…	…	…
其他信息域						

号名对照表

2．由名查号

由顶点名（字符串） 查找其编号的方法：

（1）若按顶点名排序——二分法查找（快）

（2）若不按顶点名排序——顺序查找（慢）

（3）建立辅助字典——散列表、检索树、平衡树

邻接矩阵的顺序存储

- 二维数组存储

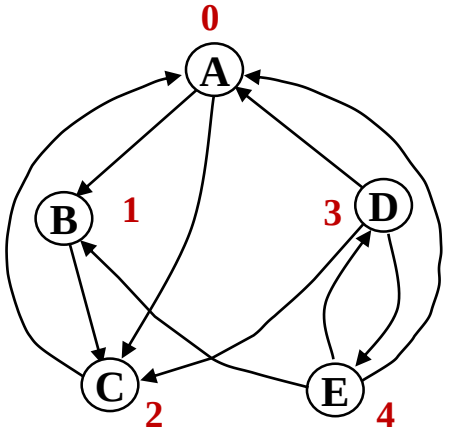
对于含有n个顶点的图，邻接矩阵是n*n的方阵，直接采用二维数组（如a[n][n]）进行存储，称为图的**邻接数组**

优点：可直接从v行w列读出边<v，w>信息

缺点：存储量较大，$O(n^2)$

适用情况：边数相对较多的**有向图**

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



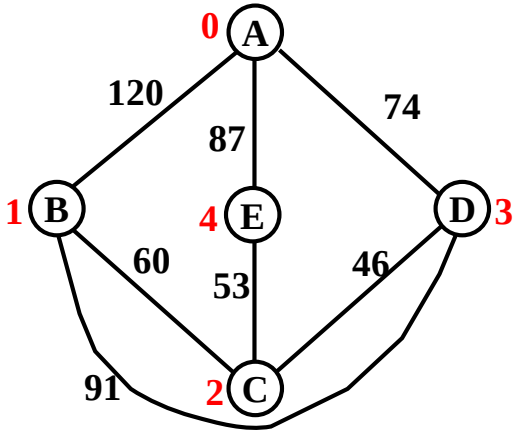
邻接矩阵的顺序存储

- 一维数组压缩存储

利用无向图（无向加权图）邻接矩阵的对称性，用一维数组压缩存储（仅存严格下三角部分）

	0	1	2	3	4	5	6	7	8	9
b	120	∞	60	74	91	46	87	∞	53	∞

	0	1	2	3	4
0	0	120	∞	74	87
1	120	0	60	91	∞
2	∞	60	0	46	53
3	74	91	46	0	∞
4	87	∞	53	∞	0



邻接矩阵的顺序存储

❖ 一维数组压缩存储

无向图边存取方法

假设：邻接矩阵（对角线元素全为0的对称矩阵）$A_{n\times n}$，行列号对应顶点编号0至n-1（n是顶点个数），用一维数组b[m]存储矩阵A

第i行存储i个元素（i=0，1，2……n-1），

数组元素总数m=1+2+3+……+(n-1)=n(n-1)/2

对于任意i和j（1≤i≤n-1，0≤j<i），第i行第j列元素a_{ij}存储于b[k]，这里k=1+2+……+(i-1)+j=i*(i-1)/2+j

找到a_{ij}，也就相当于找到对称元素a_{ji}

邻接矩阵的顺序存储

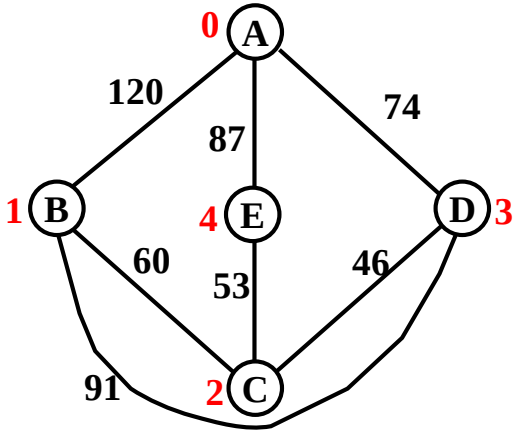
- 一维数组压缩存储

无向图边存取方法：k=i*(i-1)/2+j

$Loc(a_{31})=3*(3-1)/2+1=4$

	0	1	2	3	4	5	6	7	8	9
b	120	∞	60	74	91	46	87	∞	53	∞

	0	1	2	3	4
0	0				
1	120	1	2		
2	∞	60	3	4	5
3	74	91	46	6	7	8	9
4	87	∞	53	∞	



邻接矩阵的顺序存储

❖ 存储效率分析

邻接数组存储法（无论是否压缩）是边集的一种顺序存储方式

优点：简单，极易在图中查找、插入、删除一条边

缺点：存储n个顶点的图，要占用$O(n^2)$个存储单元

（无论图中实际含多少条边）

图的读入、存储空间的初始化等需要花费$O(n^2)$个单位时间

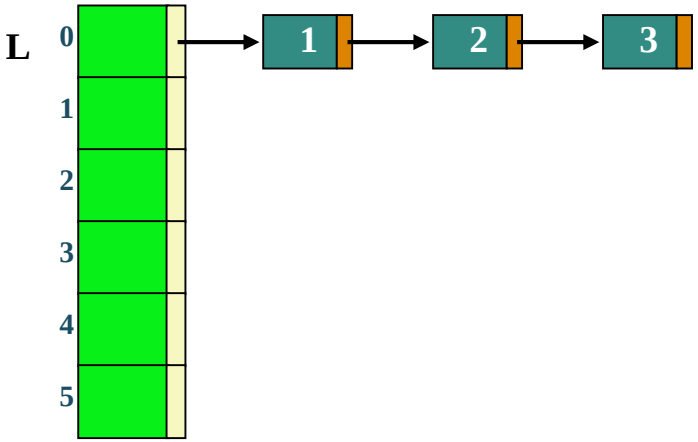
对于边数$m<<n^2$时的稀疏图是不经济的

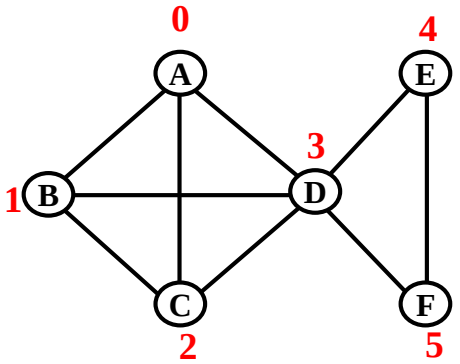
存储稀疏图最好采用邻接表法

邻接表存储

❖ 邻接表的概念

顶点v的所有邻接点组成的表，称为v的邻接表L[v]



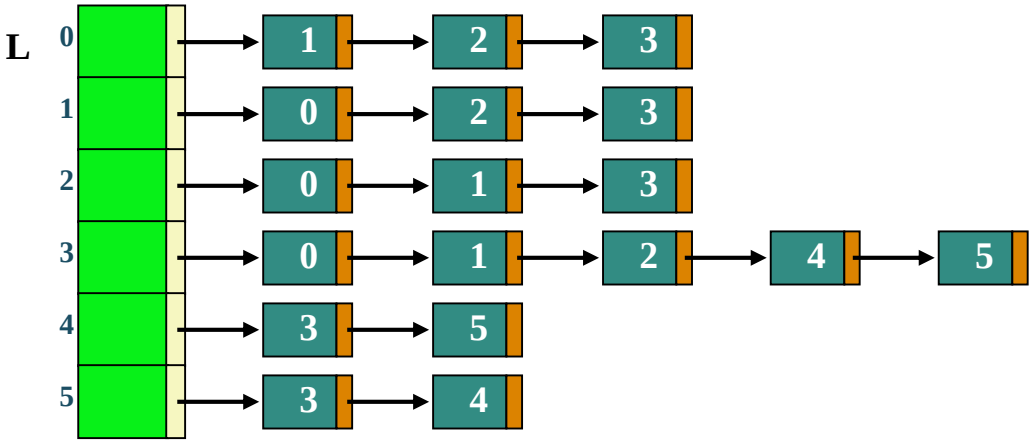


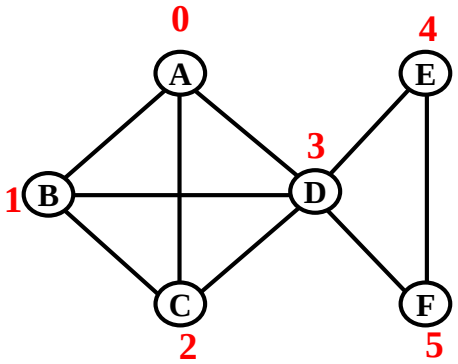
邻接表存储

❖ 邻接表的概念

顶点v的所有邻接点组成的表，称为v的邻接表L[v]

各顶点邻接表总称为图的邻接表（adjacency lists）





邻接表存储

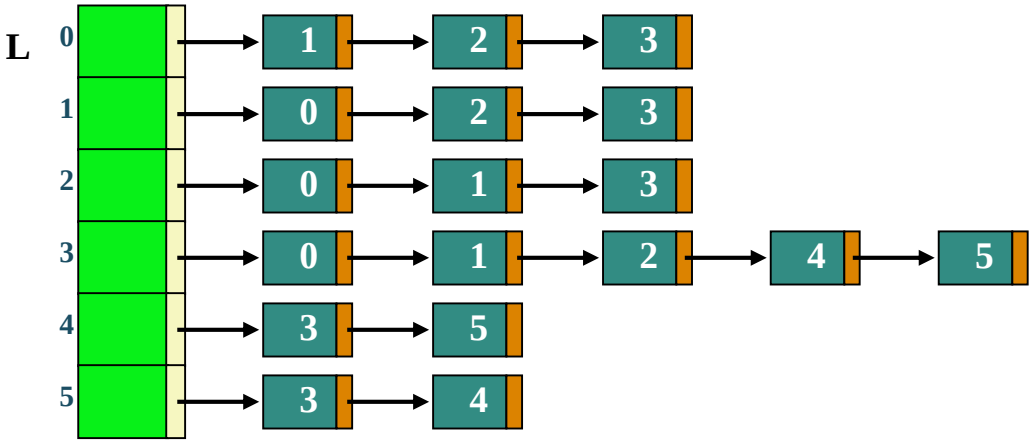
❖ 邻接表的概念

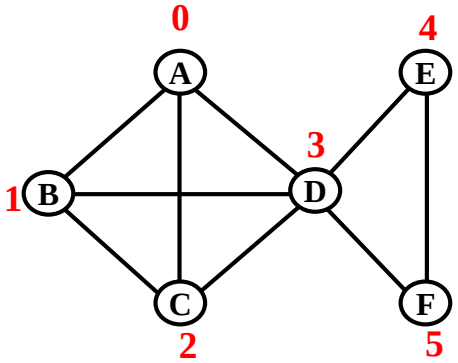
顶点v的所有邻接点组成的表，称为v的邻接表L[v]

各顶点邻接表总称为图的邻接表（adjacency lists）

邻接表采用链式存储（即邻接链表）

L[v]中每个结点对应图中一条边，称为边结点





邻接表存储

- 邻接表的概念

  顶点v的所有邻接点组成的表，称为v的邻接表L[v]

  各顶点邻接表总称为图的邻接表（adjacency lists）

  邻接表采用链式存储（即邻接链表）

  L[v]中每个结点对应图中一条边，称为边结点

对有向图，若存在边<v，w>，则w处于L[v]中

对无向图，若存在边(v，w)，则w处于L[v]中，同时v也处于L[w]中

邻接表存储

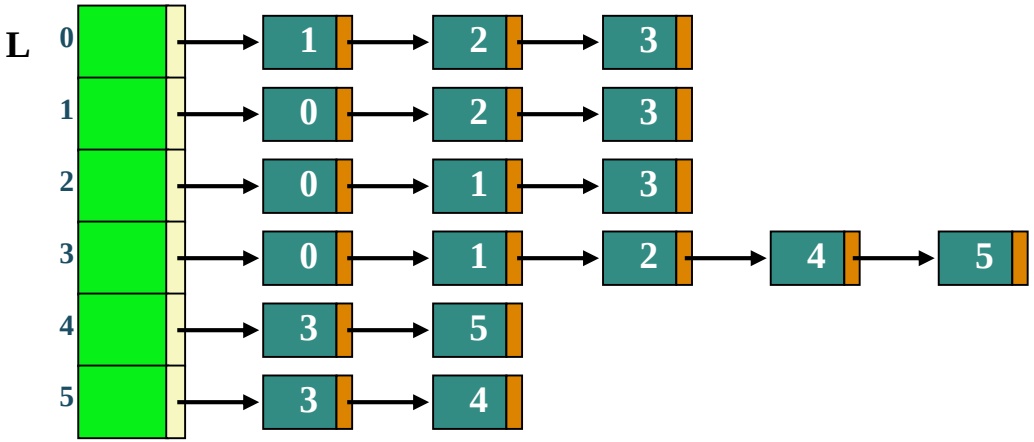
❖ 邻接表的概念

n个顶点的邻接链表构成一个链表组

使用一个表头指针数组将n个链表结合在一起

邻接表通常设计成单向链表形式

根据需要，也可设计成双向的、或其他形式的链表



邻接表存储

❖ 边结点的结构

加权图的边结点类型定义

含有邻接点名称（编号）域、边的长度（耗费）域和指向下一个边结点的指针域

边结点的类型定义：

```
typedef  struct edge_node
{
    int  adjacent;     // 邻接点名称域
    cost_type  cost;  // 边的耗费域
    struct edge_node  *next; //指各下一个边结点
}edge_node, *Eptr;
```

邻接表存储

❖ 边结点的结构

0/1图的边结点类型定义

含有邻接点名称(编号)域和指向下一个边结点的指针域

边结点的类型定义：

```
typedef  struct edge_node
{
    int  adjacent;     // 邻接点名称域
    struct edge_node  *next; //指各下一个边结点
}edge_node, *Eptr;
```

邻接表存储

❖ 表头（顶点）结点的结构

含有顶点名称域和指向邻接表首结点的指针域

必要时，可增加其他域

顶点结点的类型定义：

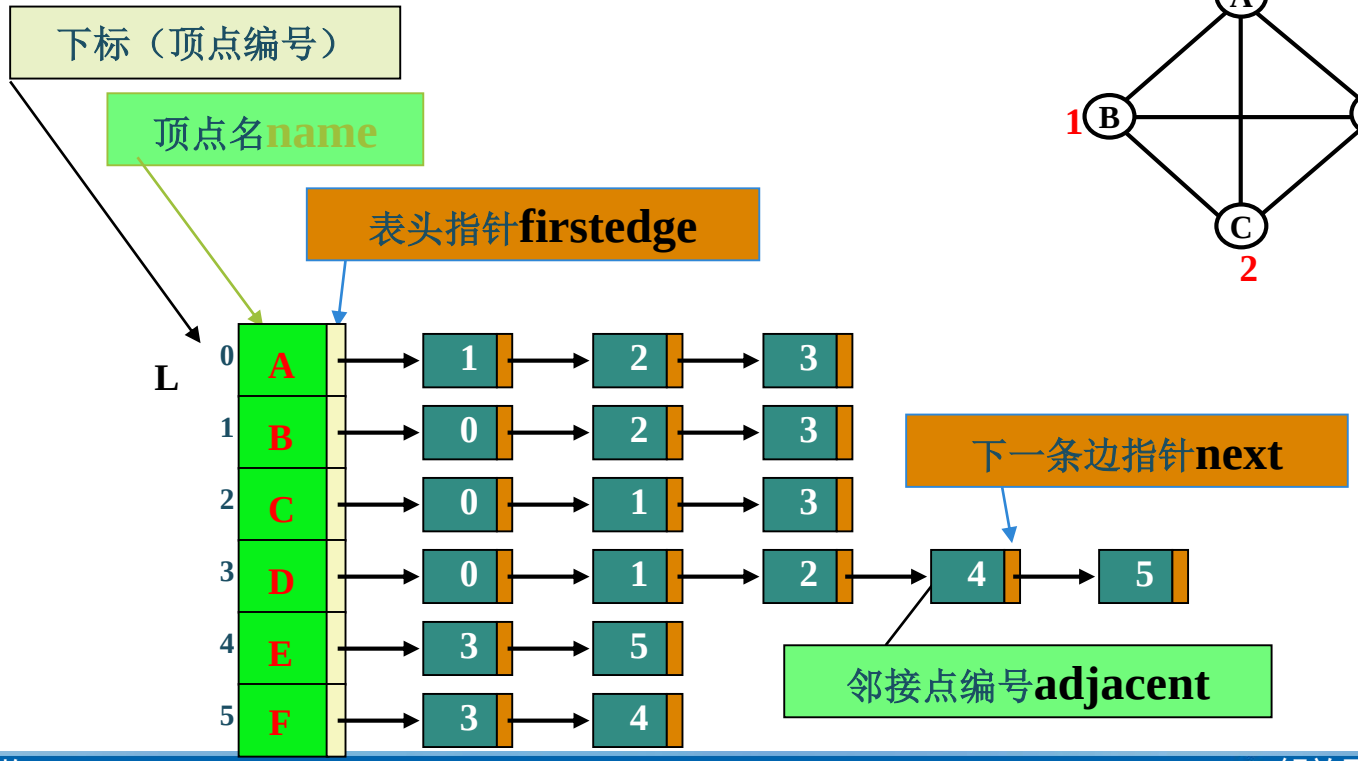
```
typedef  struct head_node
{
    Vname_type  name ; //顶点名，用于由号查名
    Eptr  firstedge ; //邻接表的首指针
}hnode;
```

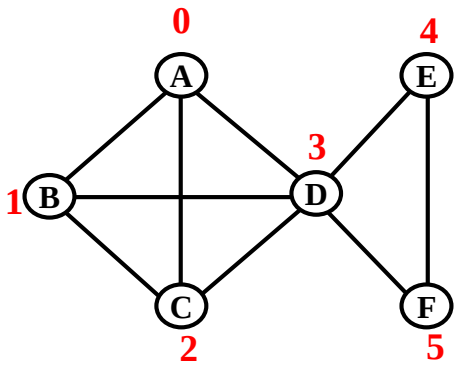
表头结点组定义为：

```
hnode   L[n];   //n为具体顶点数
```

邻接表存储

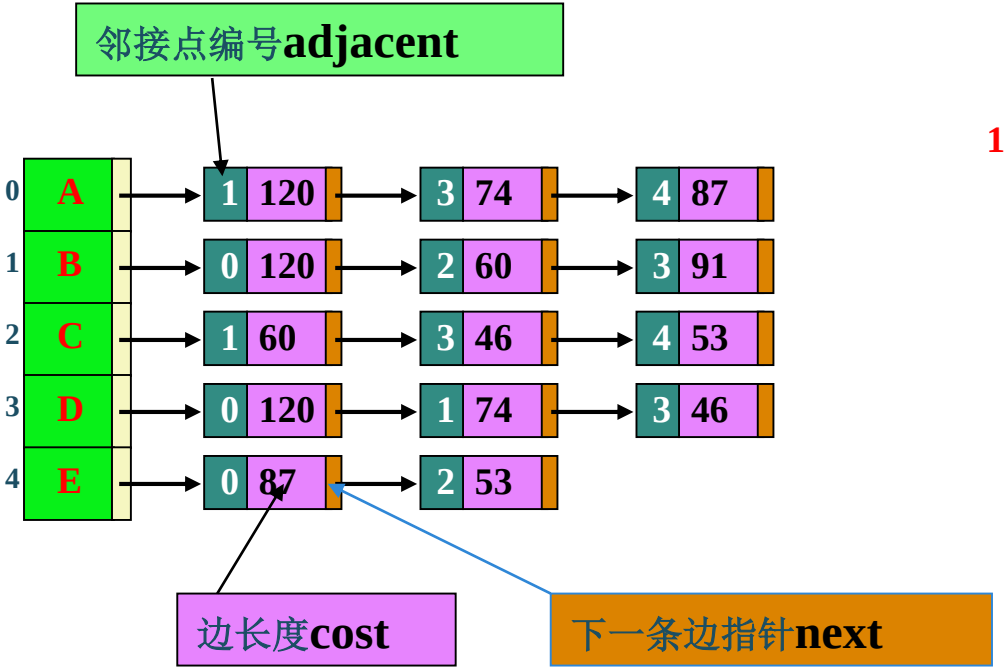
❖ 示例：无向图邻接表

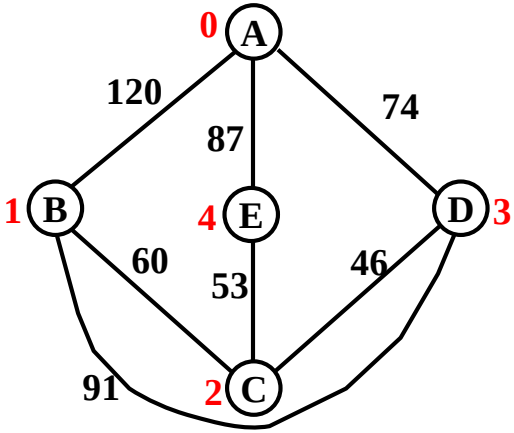




邻接表存储

❖ 示例：无向加权图邻接表





小结

- 邻接矩阵的顺序存储——邻接数组
  - 二维数组存储
  - 一维数组压缩存储
- 邻接矩阵的链式存储——邻接表

The End, Thank You!