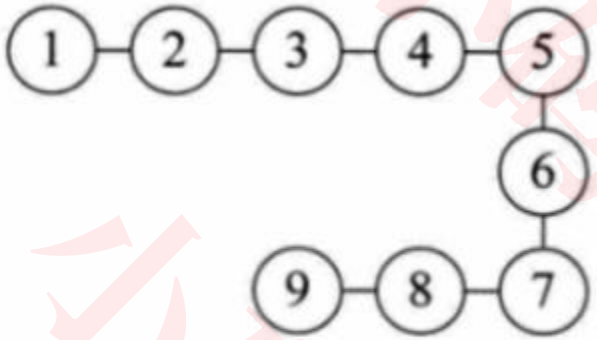
数据结构回顾

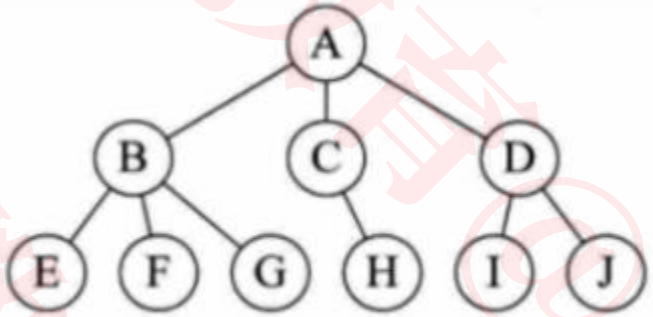


线性结构

数组、链表、

栈、队列、

哈希表

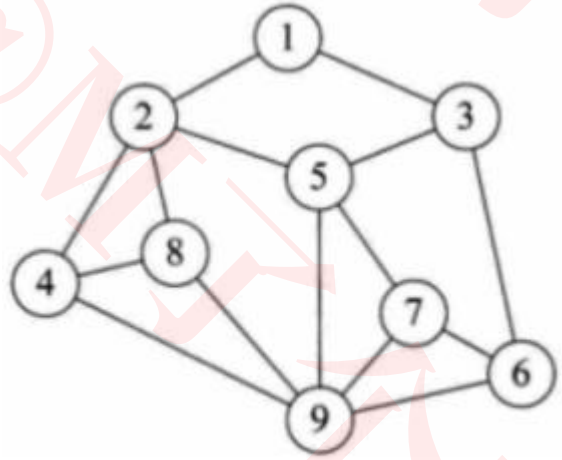


树形结构

二叉树、B树、

堆、Trie、

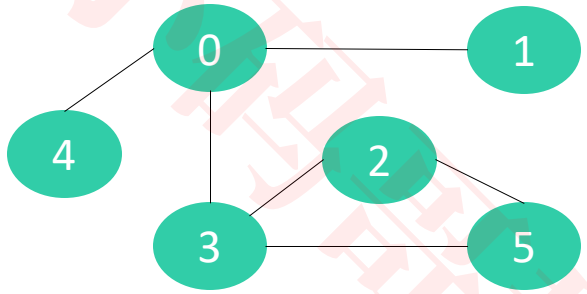
哈夫曼树、并查集

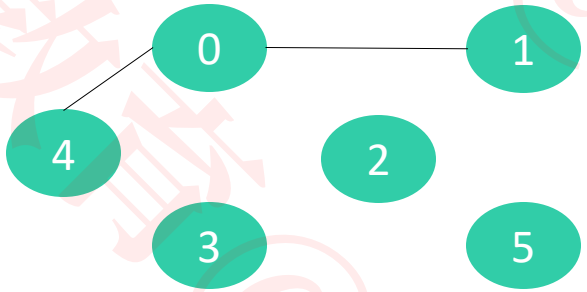


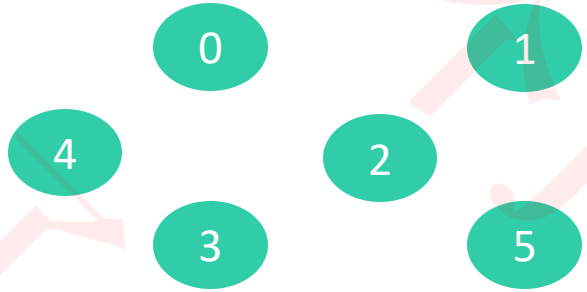
图形结构

图（Graph）

- 图由顶点（vertex）和边（edge）组成，通常表示为 G = (V, E)
  - G表示一个图，V是顶点集，E是边集
  - 顶点集V有穷且非空
  - 任意两个顶点之间都可以用边来表示它们之间的关系，边集E可以是空的



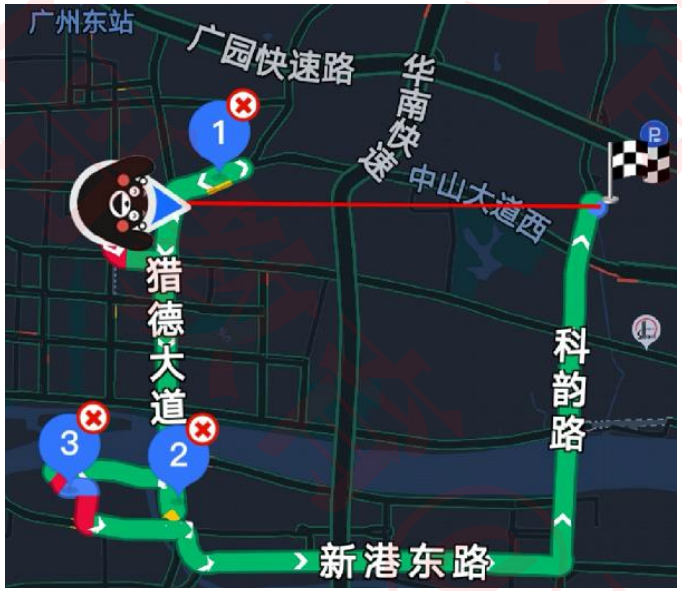


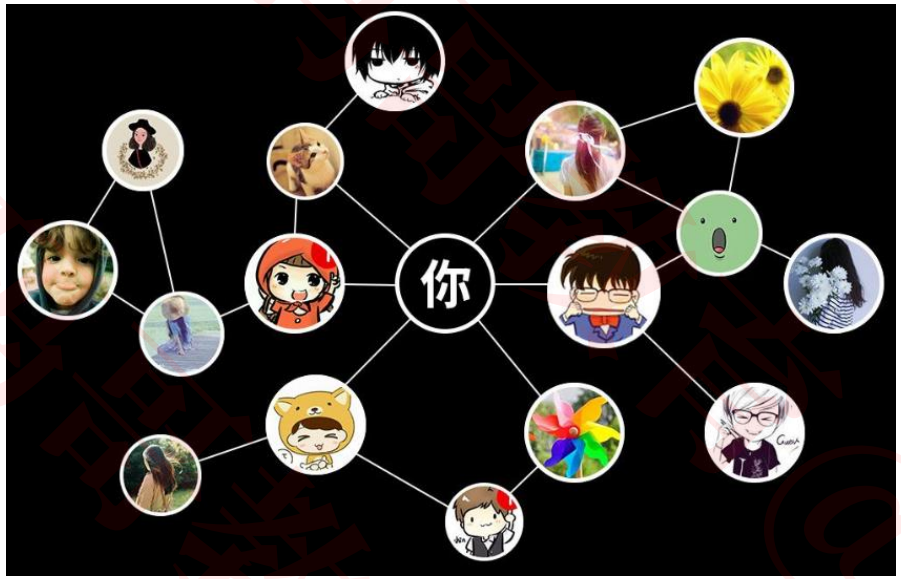


小码哥教育 SEEMYGO

图的应用举例

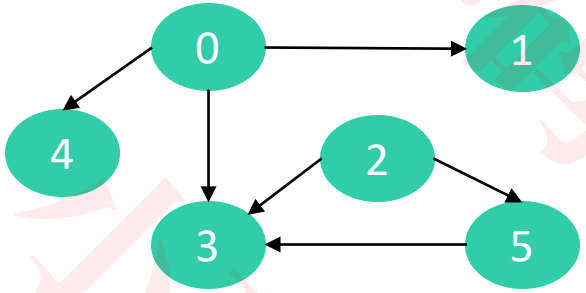
- 图结构的应用极其广泛
  - 社交网络
  - 地图导航
  - 游戏开发
  - ……



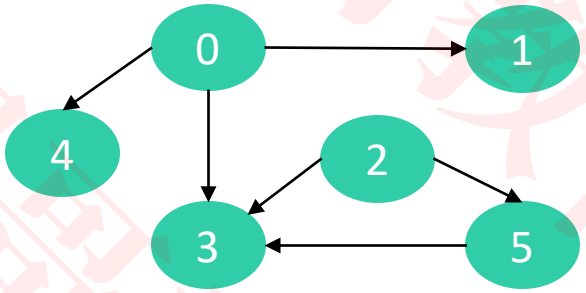


有向图（Directed Graph）

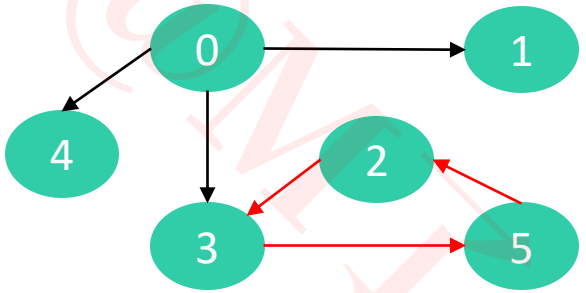
- 有向图的边是有明确方向的



- 有向无环图（Directed Acyclic Graph，简称 DAG）
  - 如果一个有向图，从任意顶点出发无法经过若干条边回到该顶点，那么它就是一个有向无环图



无环



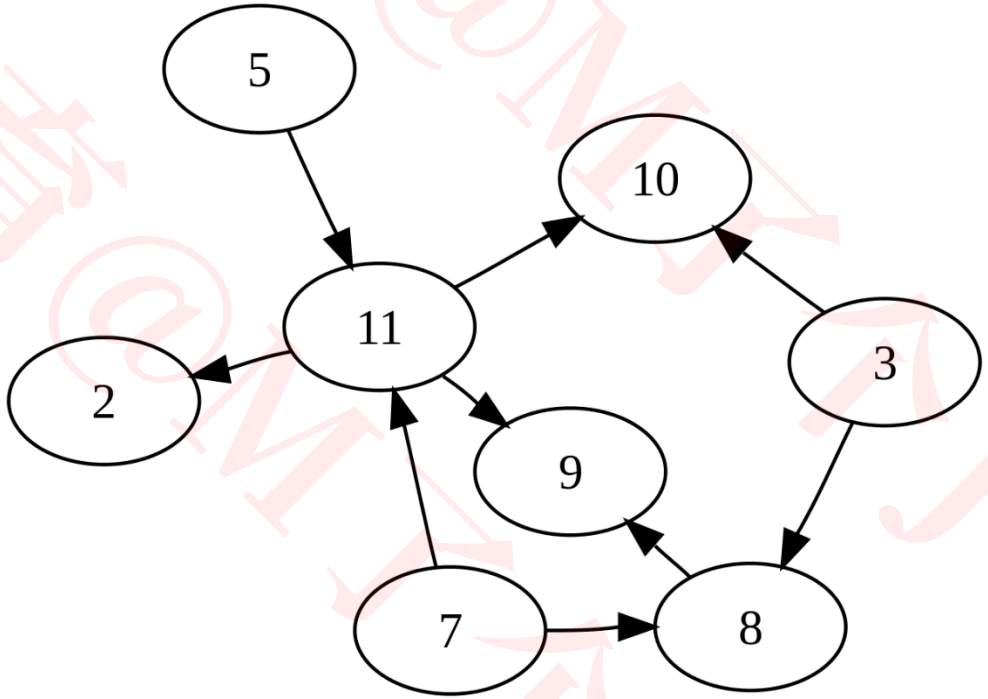
有环

出度、入度

- 出度、入度适用于有向图

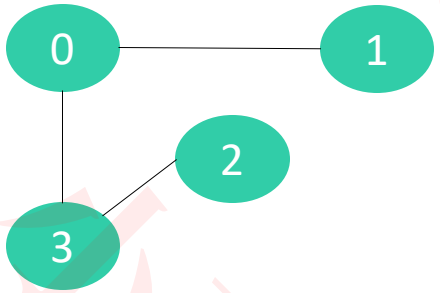
- 出度（Out-degree）
  - 一个顶点的出度为 x，是指有 x 条边以该顶点为起点
  - 顶点11的出度是3

- 入度（In-degree）
  - 一个顶点的入度为 x，是指有 x 条边以该顶点为终点
  - 顶点11的入度是2

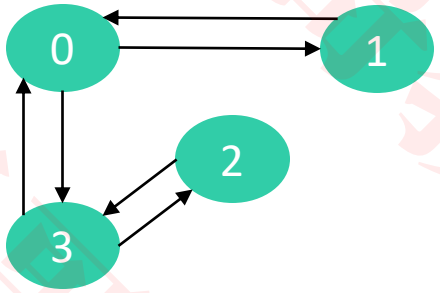


无向图（Undirected Graph）

- 无向图的边是无方向的

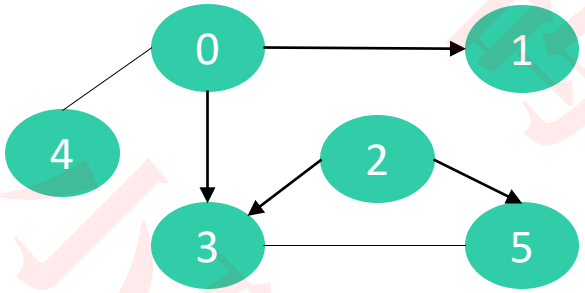


- 效果类似于下面的有向图



混合图（Mixed Graph）

- 混合图的边可能是无向的，也可能是有向的

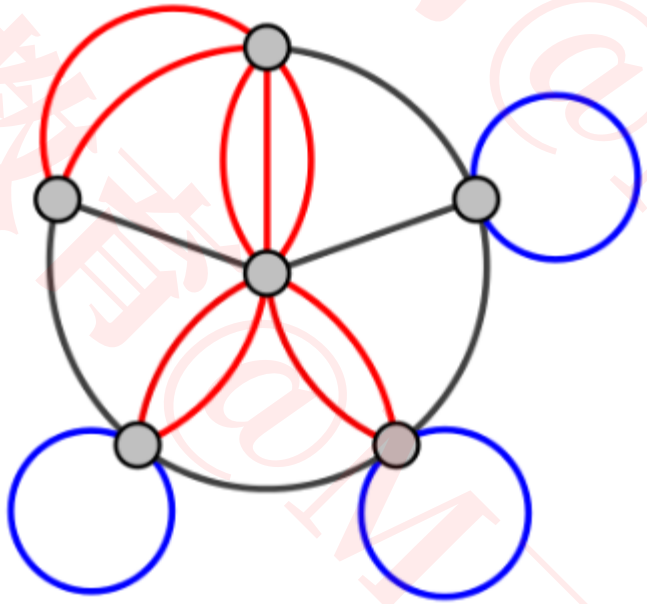


简单图、多重图

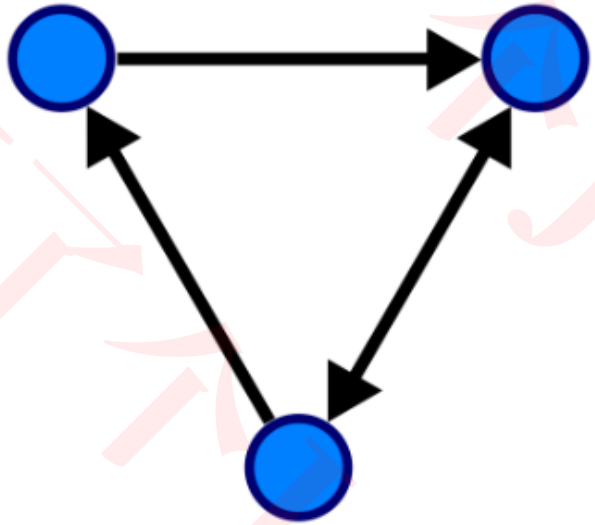
- 平行边
  - 在无向图中，关联一对顶点的无向边如果多于1条，则称这些边为平行边
  - 在有向图中，关联一对顶点的有向边如果多于1条，并且它们的的方向相同，则称这些边为平行边

- 多重图（Multigraph）
  - 有平行边或者有自环的图

- 简单图（Simple Graph）
  - 既没有平行边也不没有自环的图
  - 课程中讨论的基本都是简单图



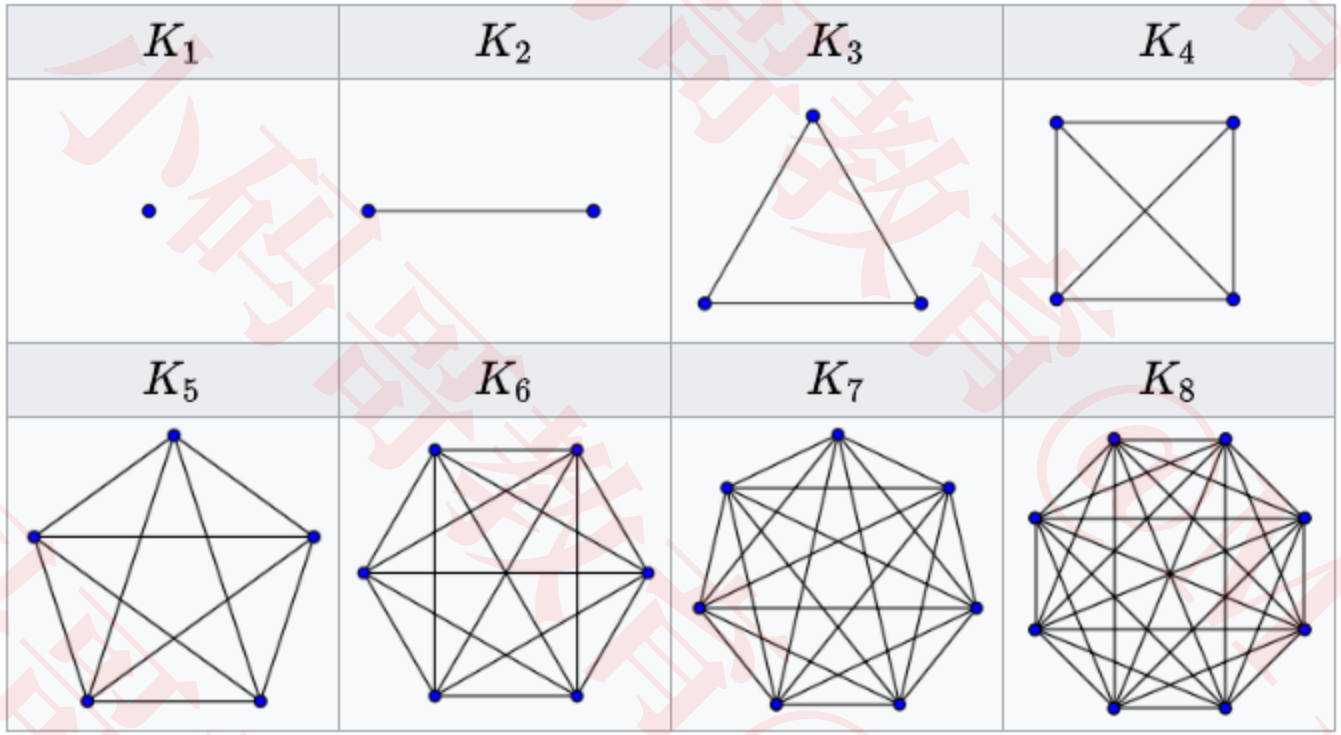
多重图



简单图

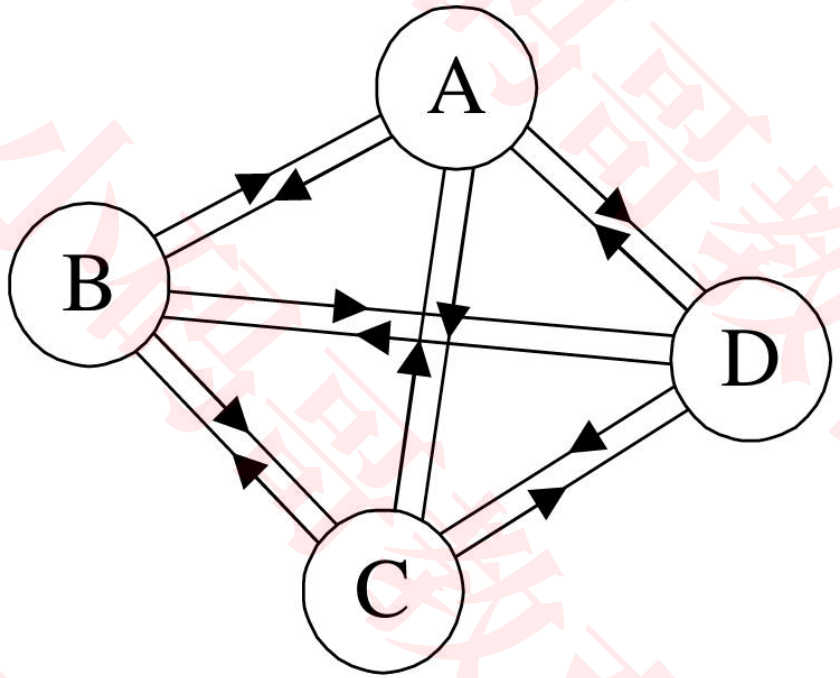
无向完全图（Undirected Complete Graph）

- 无向完全图的任意两个顶点之间都存在边
- n 个顶点的无向完全图有 $n(n-1)/2$ 条边
  - ✓ $(n-1)+(n-2)+(n-3)+\cdots+3+2+1$



有向完全图（Directed Complete Graph）

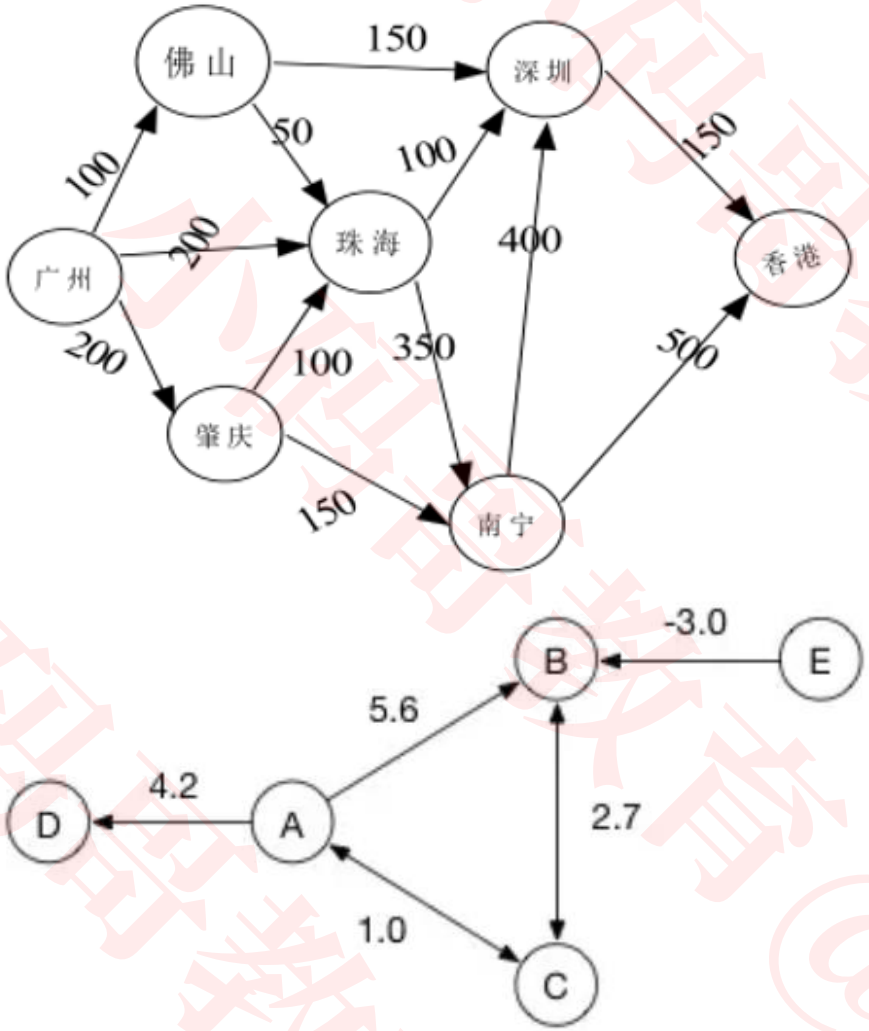
- 有向完全图的任意两个顶点之间都存在方向相反的两条边
  - n 个顶点的有向完全图有 $n(n-1)$ 条边

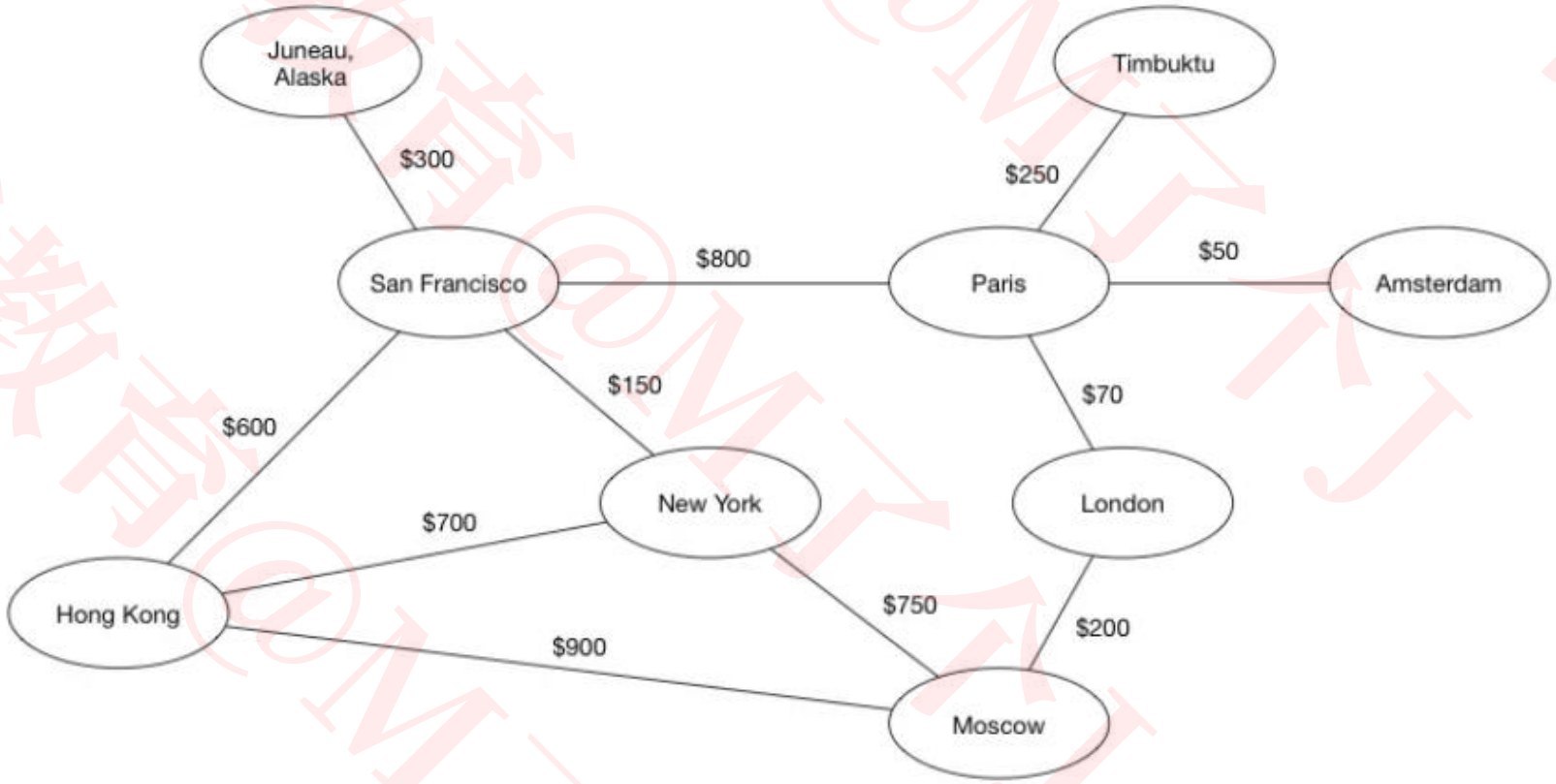


- 稠密图（Dense Graph）：边数接近于或等于完全图
- 稀疏图（Sparse Graph）：边数远远少于完全图

有权图（Weighted Graph）

- 有权图的边可以拥有权值（Weight）

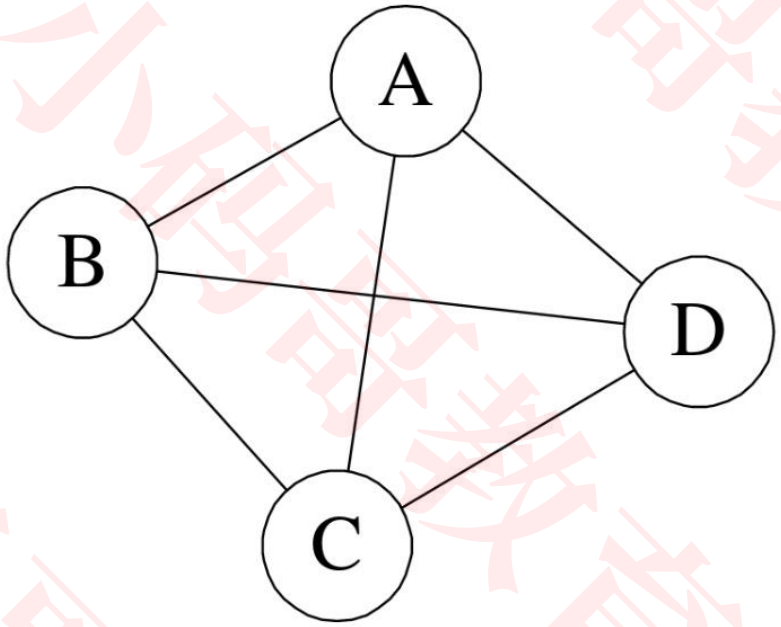


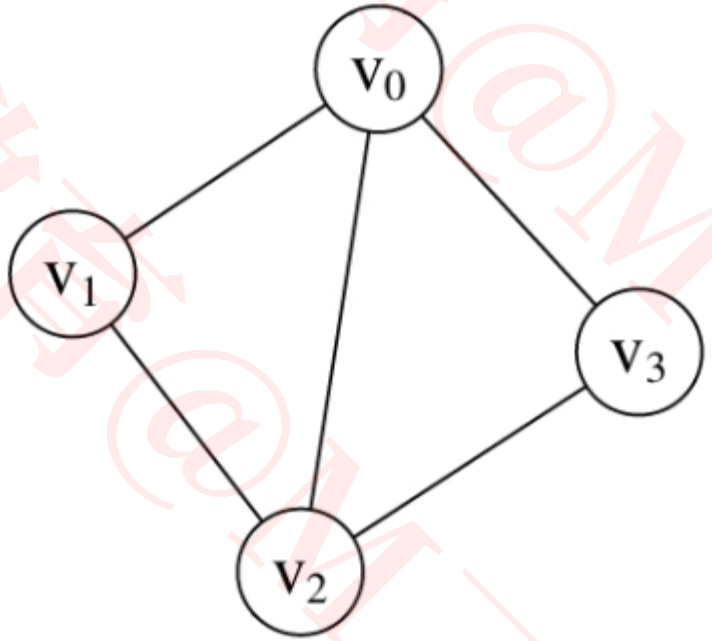


连通图（Connected Graph）

- 如果顶点 x 和 y 之间存在可相互抵达的路径（直接或间接的路径），则称 x 和 y 是连通的

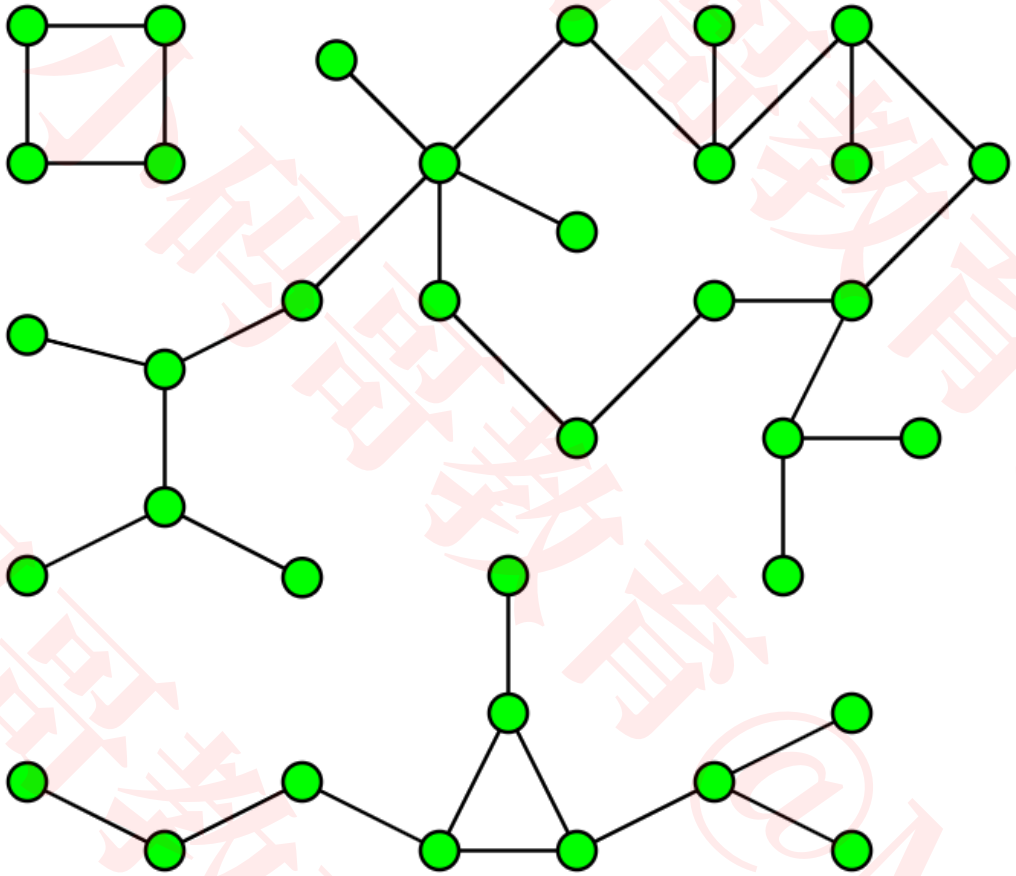
- 如果无向图 G 中任意2个顶点都是连通的，则称G为连通图





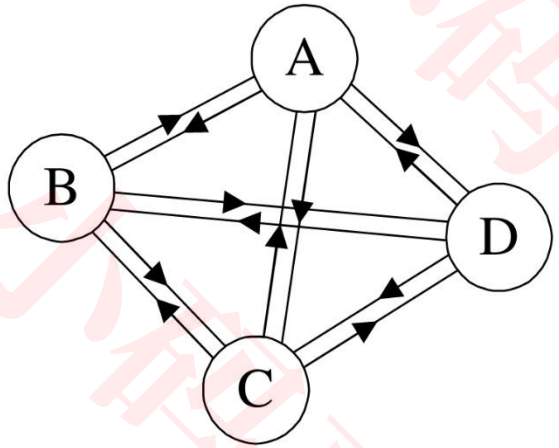
连通分量（Connected Component）

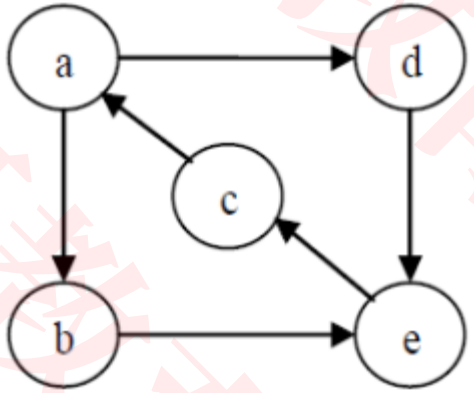
- 连通分量：无向图的极大连通子图
  - 连通图只有一个连通分量，即其自身；非连通的无向图有多个连通分量
- 下面的无向图有3个连通分量

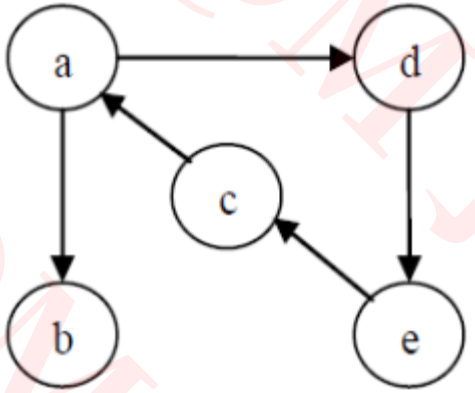


强连通图（Strongly Connected Graph）

- 如果有向图 G 中任意2个顶点都是连通的，则称G为强连通图



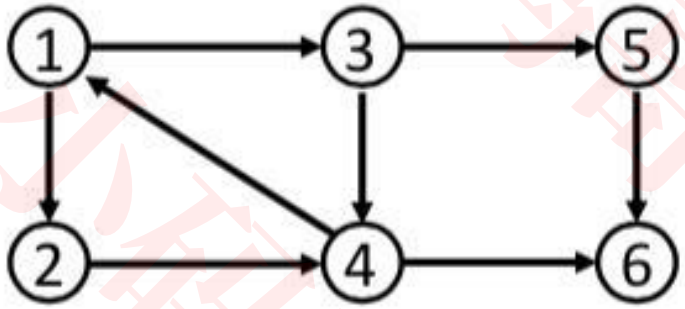




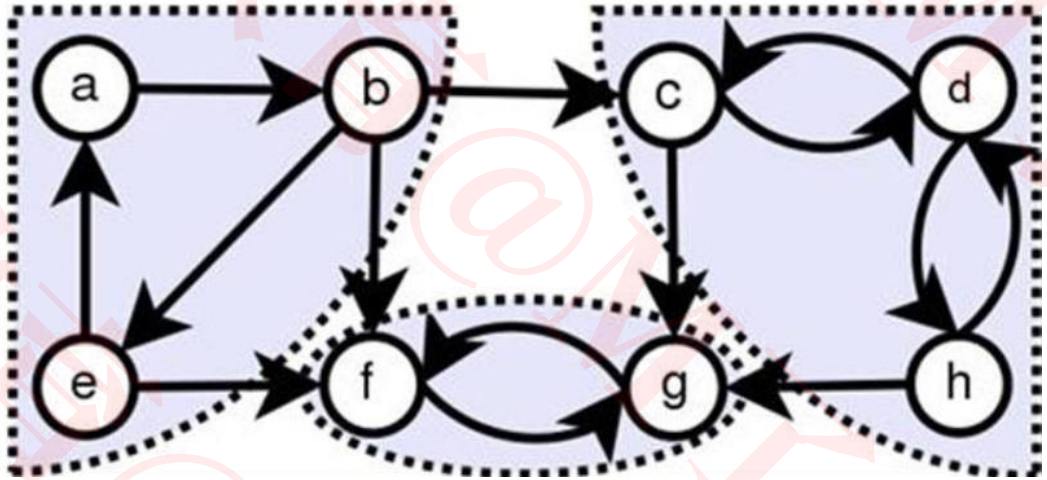
不是强连通图

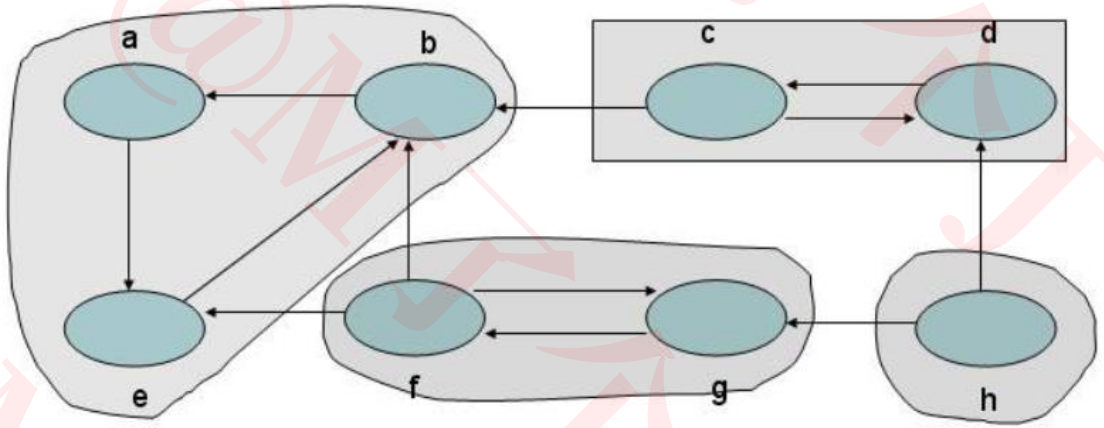
强连通分量（Strongly Connected Component）

- 强连通分量：有向图的极大强连通子图
- 强连通图只有一个强连通分量，即其自身；非强连通的有向图有多个强连通分量



强连通分量：{1,2,3,4}、{5}、{6}



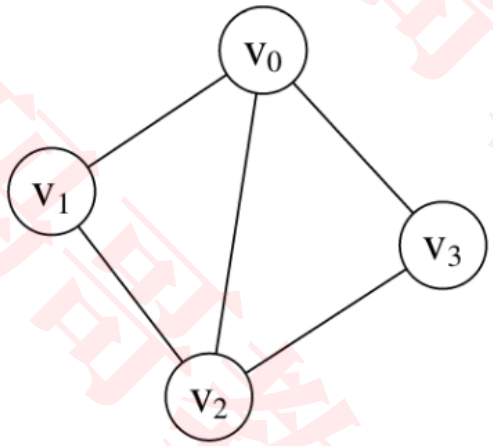


图的实现方案

- 图有2种常见的实现方案
  - 邻接矩阵（Adjacency Matrix）
  - 邻接表（Adjacency List）

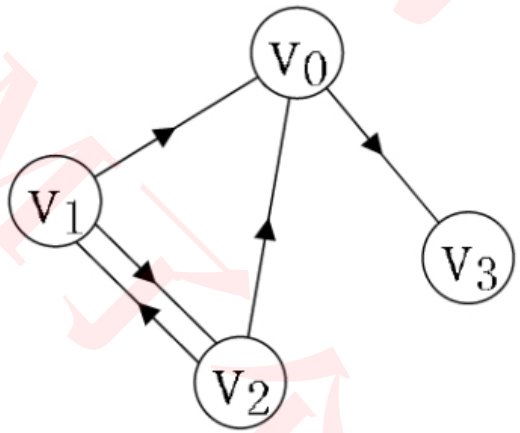
邻接矩阵（Adjacency Matrix）

- 邻接矩阵的存储方式
  - 一维数组存放顶点信息
  - 二维数组存放边信息
- 邻接矩阵比较适合稠密图
  - 不然会比较浪费内存



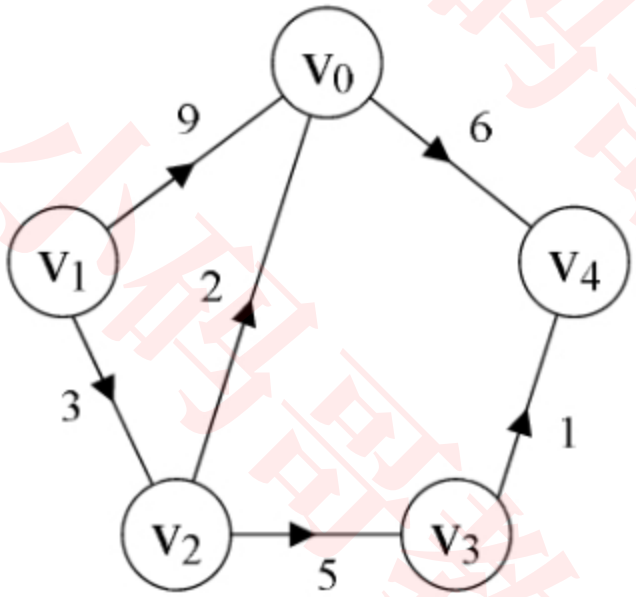
顶点数组			
v_0	v_1	v_2	v_3

边数组	v_0	v_1	v_2	v_3
v_0	0	1	1	1
v_1	1	0	1	0
v_2	1	1	0	1
v_3	1	0	1	0



边数组	v_0	v_1	v_2	v_3
v_0	0	0	0	1
v_1	1	0	1	0
v_2	1	1	0	0
v_3	0	0	0	0

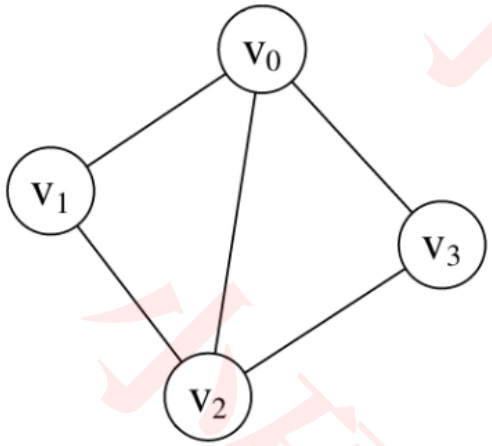
邻接矩阵 – 有权图

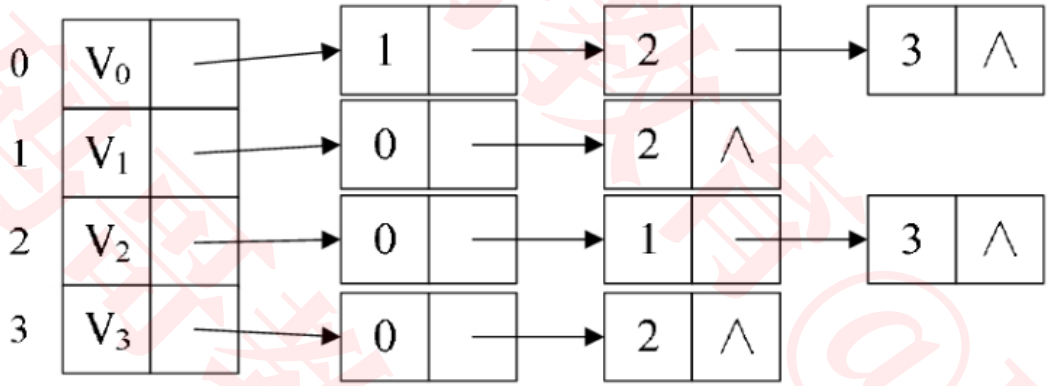


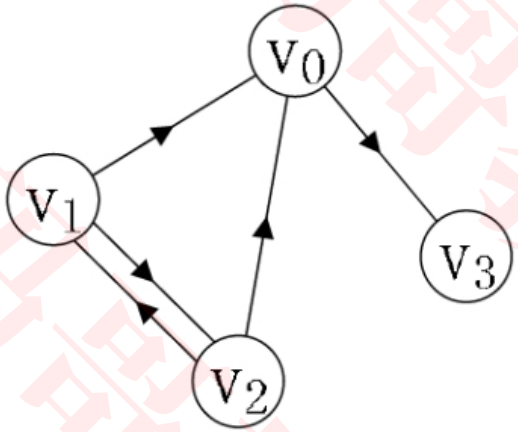
顶点数组				
v_0	v_1	v_2	v_3	v_4

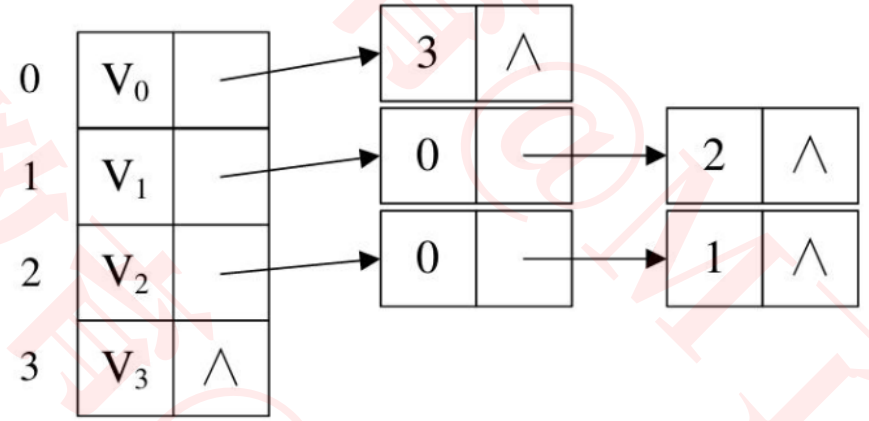
边数组					
	v_0	v_1	v_2	v_3	v_4
v_0	∞	∞	∞	∞	6
v_1	9	∞	3	∞	∞
v_2	2	∞	∞	5	∞
v_3	∞	∞	∞	∞	1
v_4	∞	∞	∞	∞	∞

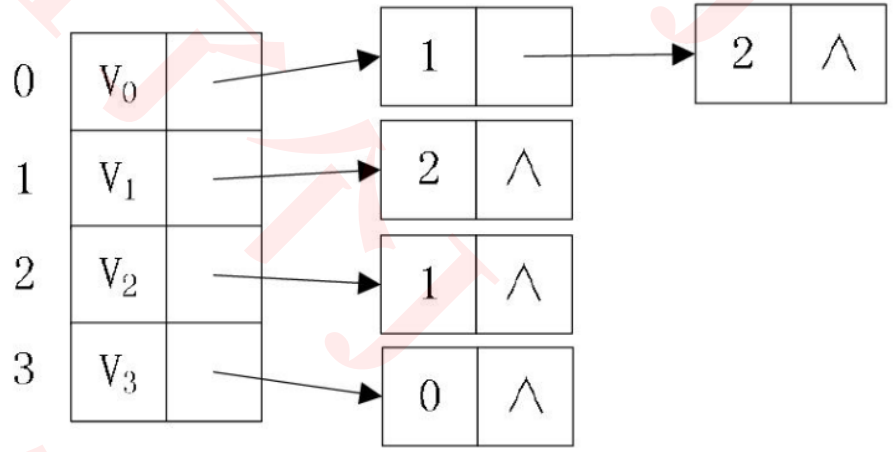
邻接表（Adjacency List）





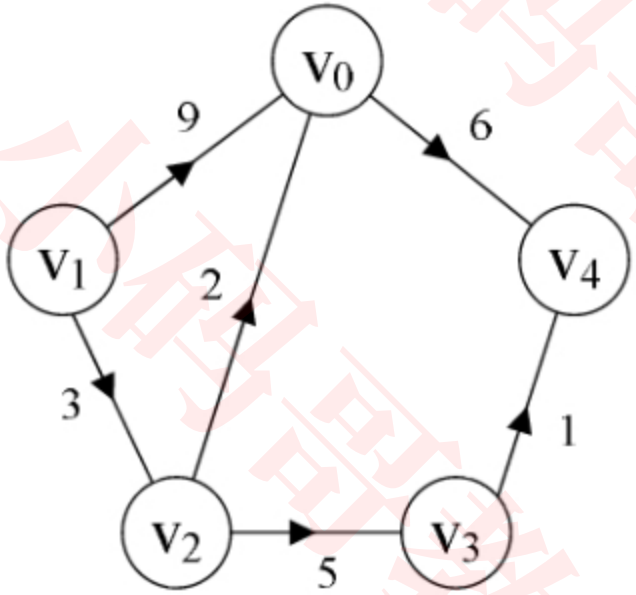


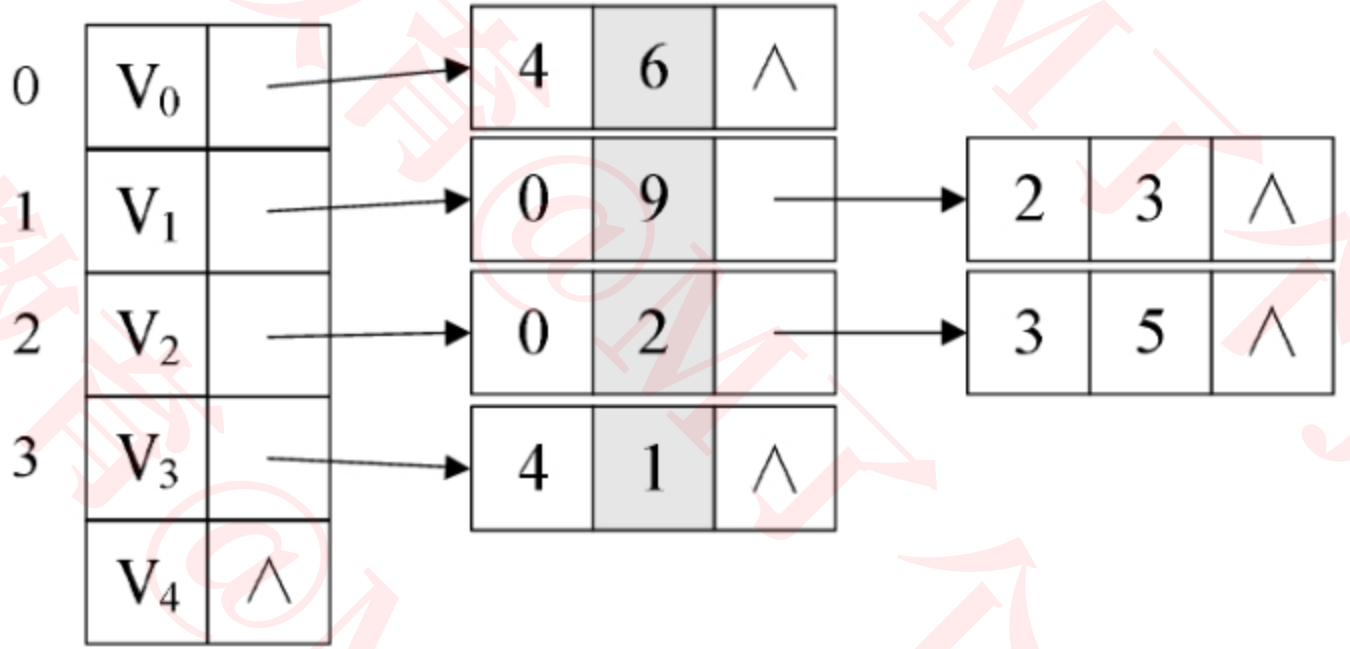




逆邻接表

邻接表 – 有权图





小码哥教育 SEEMYGO

图的基础接口

```
int verticesSize();
int edgesSize();

void addVertex(V v);
void removeVertex(V v);

void addEdge(V fromV, V toV);
void addEdge(V fromV, V toV, E weight);
void removeEdge(V fromV, V toV);
```

小码哥教育 SEEMYGO

顶点的定义

```
private static class Vertex<V, E> {
    V value;
    Set<Edge<V, E>> inEdges = new HashSet<>();
    Set<Edge<V, E>> outEdges = new HashSet<>();
    Vertex(V value) {
        this.value = value;
    }
    @Override
    public boolean equals(Object obj) {
        return Objects.equals(value, ((Vertex<V, E>) obj).value);
    }
    @Override
    public int hashCode() {
        return value == null ? 0 : value.hashCode();
    }
}
```

小码哥教育 SEEMYGO

边的定义

```
private static class Edge<V, E> {
    Vertex<V, E> from;
    Vertex<V, E> to;
    E weight;
    public boolean equals(Object obj) {
        Edge<V, E> edge = (Edge<V, E>) obj;
        return from.equals(edge.from) && to.equals(edge.to);
    }
    public int hashCode() {
        return from.hashCode() * 31 + to.hashCode();
    }
}
```