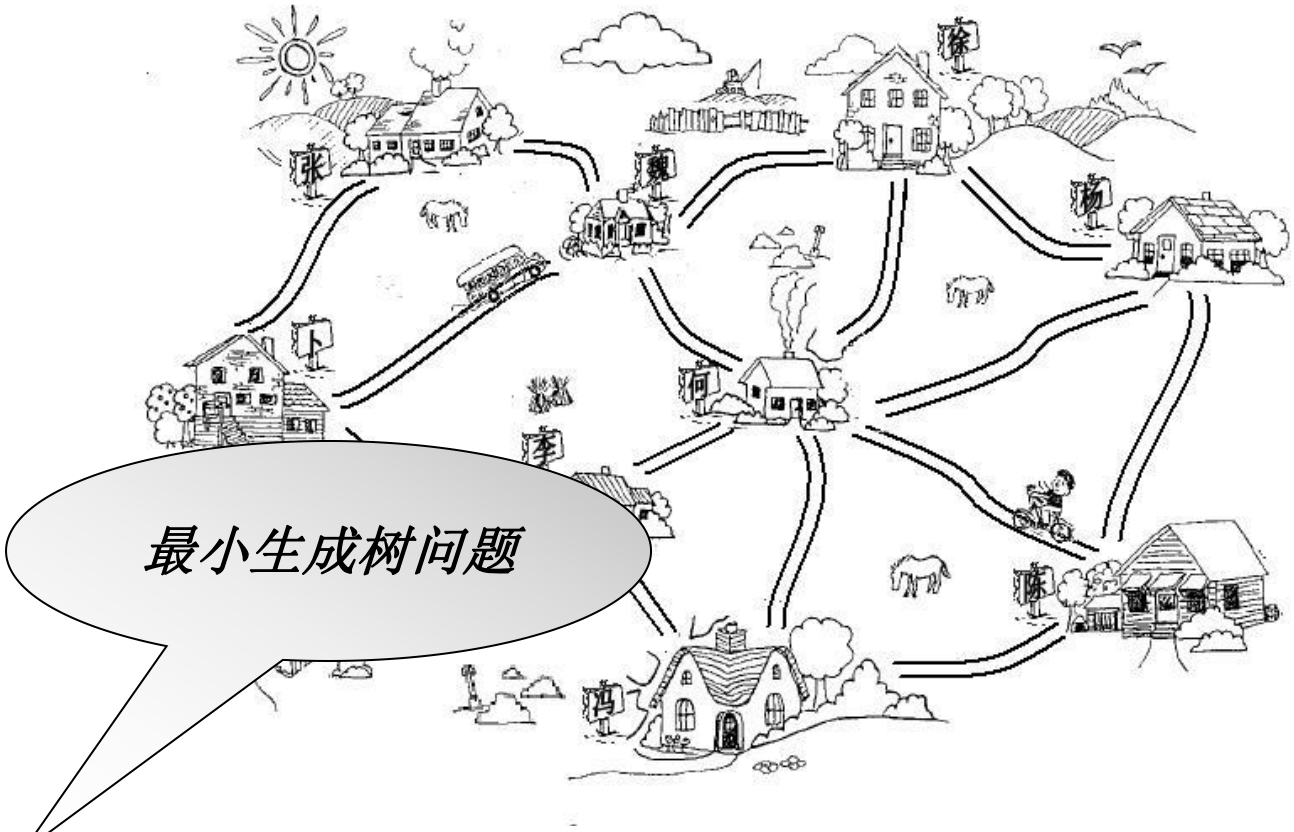
第七章 图

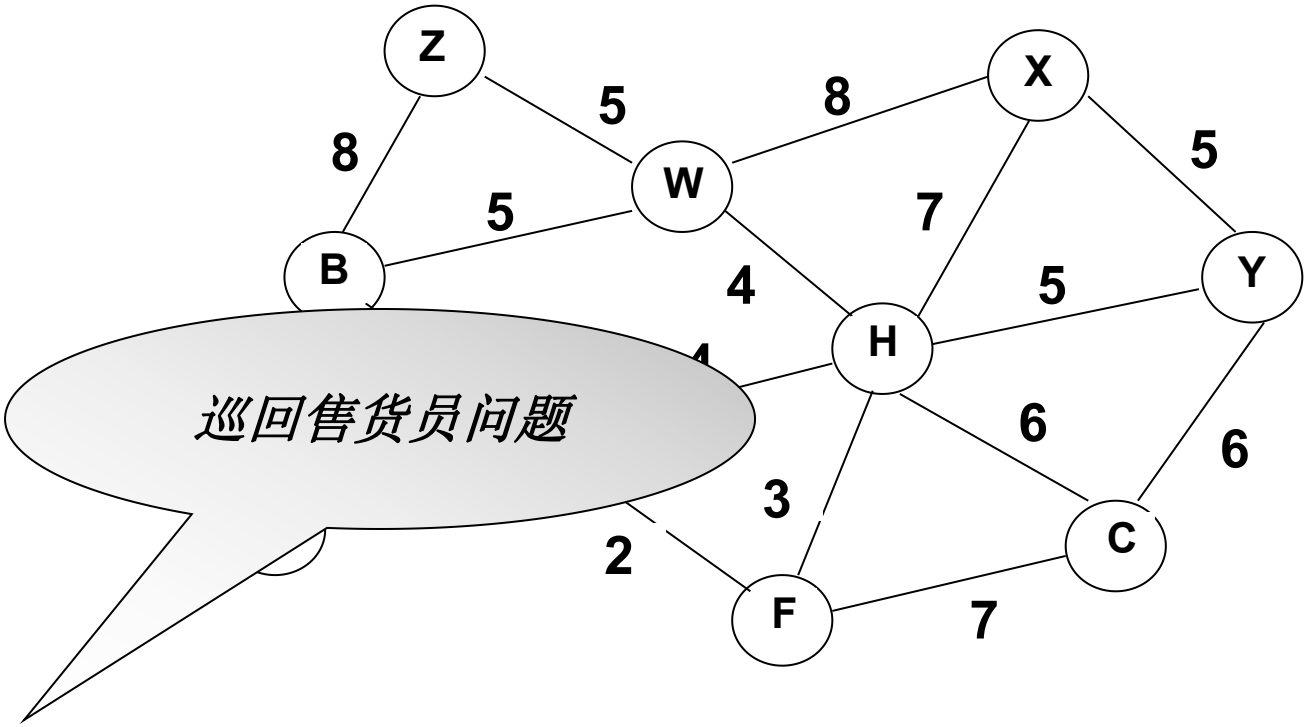
7.1 图的定义和术语
7.2 图的存储结构
7.3 图的遍历
7.4 图的连通
7.5 有向无环图及其应用
7.6 最短路径

【例】图中村与村之间的道路是一个较长远的规划目标。



[问题1] 公路村村通项目要求用最小的投入实现每个村都能够有公路通达。那么应该选择建设哪些道路可以使这个投资最小呢？（假设每条道路的建设成本已知）

【例】下图为公路规划抽象及造价预算示例图。



[问题2] 在同样的抽象图中，假设把 “造价” 的含义修改成“距离”，那么我们就可以问：要走遍每个村庄，并回到起点，该如何走才能够使得总的路程最短？

7.1 图的定义和术语

一. 图的定义

- 图(Graph)是一种比线性表和树更为复杂的数据结构。
  - 线性结构：是研究数据元素之间的一对一关系，数据元素之间有明显顺序关系
  - 树结构：是研究数据元素之间的一对多的关系，数据元素之间有明显的层次关系
  - 图结构：是研究数据元素之间的多对多的关系。任意两个元素之间可能存在关系，即结点之间的关系可以是任意的，图中任意元素之间都可能相关。

- 图的应用极为广泛，已渗入到诸如语言学、逻辑学、物理、化学、电讯、计算机科学以及数学的其它分支。

7.1 图的定义和术语

一. 图的定义

- 图是由顶点集合(vertex)及顶点间的关系集合组成的一种数据结构：

  Graph＝(V, E)

  其中，

  V ={x | x∈数据对象}，顶点集合为空的图称为空图

  E是顶点之间关系的有穷集合，包括

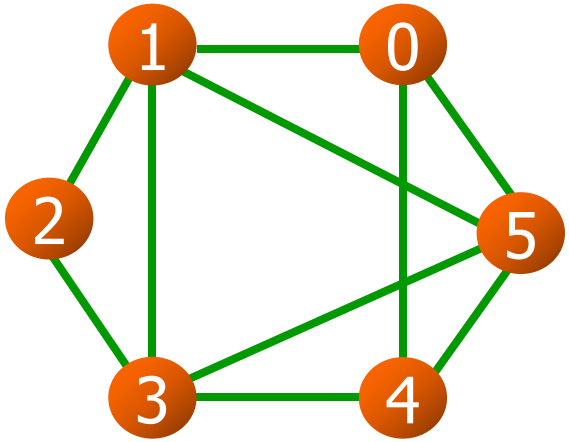
  E1 = {(x, y) | x, y ∈ V } 边的集合

  或 E2 = {<x, y> | x, y ∈ V } 弧的集合

7.1 图的定义和术语

二. 图的分类

- 无向图：用(x, y)表示两个顶点x, y之间的一条边(edge)，边是无序的。



- 例如： 图 N = { V , E } ，

其中，

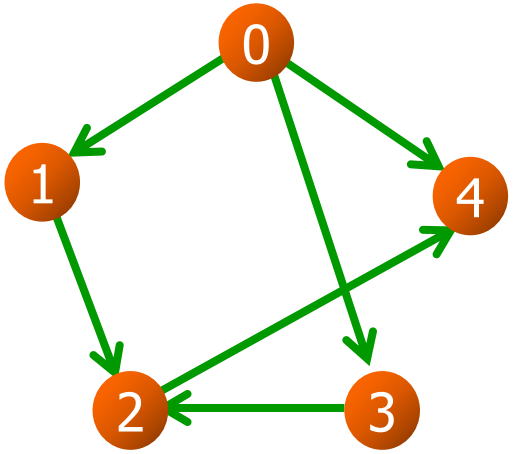
顶点集合V = {0, 1, 2, 3, 4, 5}

边的集合E = {(0, 1), (0, 4), (0, 5), (1, 2), (1, 3), (1, 5), (2, 3), (3, 4), (3, 5), (4, 5)}

7.1 图的定义和术语

二. 图的分类

- 有向图：用<x, y>表示从x到y的一条弧(Arc)，且称x为弧尾，y为弧头，<x, y>是有序的。



- 例如：　图 N = { V , E } ，

其中:

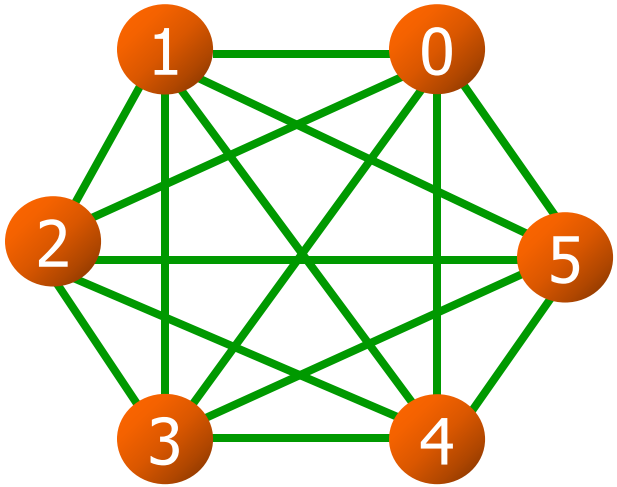
顶点集合V = {0, 1, 2, 3, 4}

边的集合E = {<0, 1>, <0, 3>, <0, 4>, <1, 2>, <2, 4>,
<3, 2> }

7.1 图的定义和术语

二. 图的分类

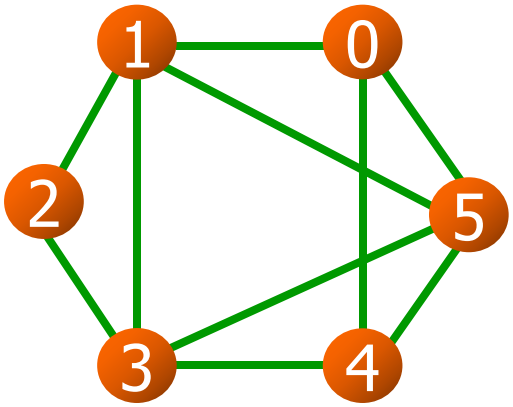
- **无向完全图**：如果无向图有 $\frac{1}{2}n(n-1)$ 条边，则称为无向完全图



7.1 图的定义和术语

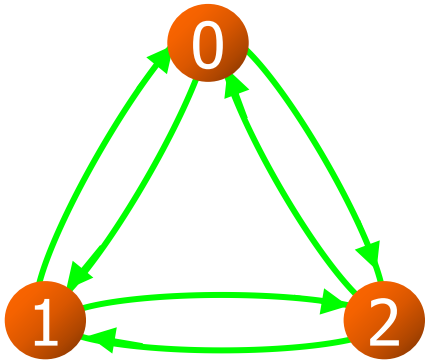
二. 图的分类

- 无向图的术语
  - **邻接点**：如果(x, y)∈E, 称x, y互为邻接点，即x, y相邻接
  - **依附、相关联**：边(x, y)依附于顶点x和y，或者说边(x, y)与顶点x和y相关联
  - **顶点的度**：和顶点相关联的边的数目，记为TD(x)



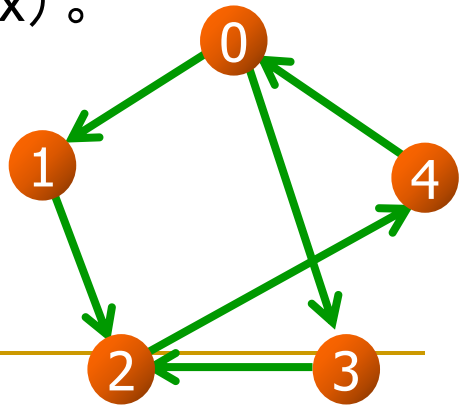
7.1 图的定义和术语

二. 图的分类



- **有向完全图**：有n(n-1)条弧的有向图。

- 有向图的术语
  - **邻接**：如果<x, y>∈E, 称x邻接到y, 或y邻接自x。
  - **相关联**：弧<x, y>与x, y相关联。
  - **入度**：以顶点为头的弧的数目，记为ID(x)。
  - **出度**：以顶点为尾的弧的数目，记为OD(x)。
  - **顶点的度**：TD(x)=ID(x)+OD(x)



7.1 图的定义和术语

二. 图的分类

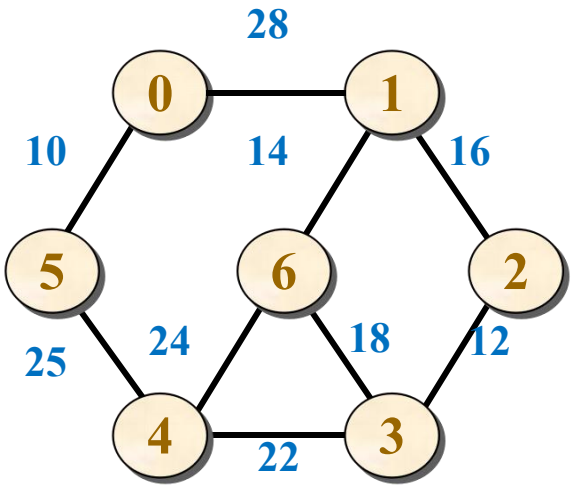
- 只有很少边或弧（e<nlogn）的图称为稀疏图，反之称为稠密图。
- 若图中有n个顶点，有e条边或弧，则满足如下关系：

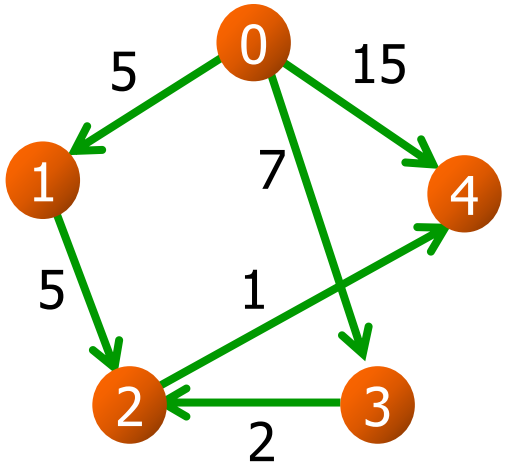
$$e = \frac{1}{2}\sum_{i=1}^{n} TD(v_i)$$

7.1 图的定义和术语

二. 图的分类

- 带权的图称为网
- 权：与图的边或弧相关的数值。





练习

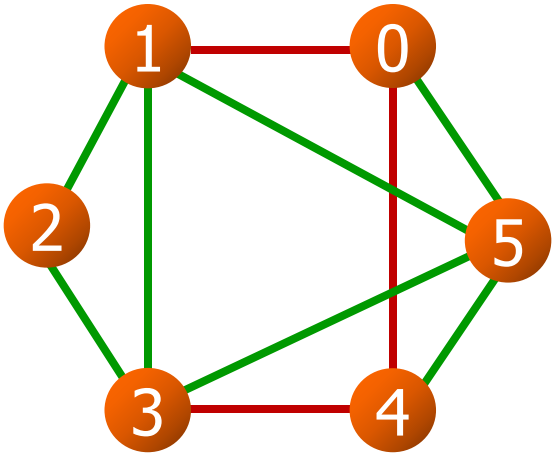
1. 设无向图G=(V,E)，其中V={a,b,c,d,e}，
   E={(a,b),(a,d),(b,c),(c,d),(d,e),(e,b),(e,c)}
   - 请画出该图
   - 求各顶点的度

2. 设一有向图G=(V,E)，其中V={a,b,c,d,e}，E={<a,b>,<a,d>,<b,a>,<c,b>,<c,d>,<d,e>,<e,a>,<e,b>,<e,c>}
   - 请画出该有向图
   - 求各顶点的入度、出度和度

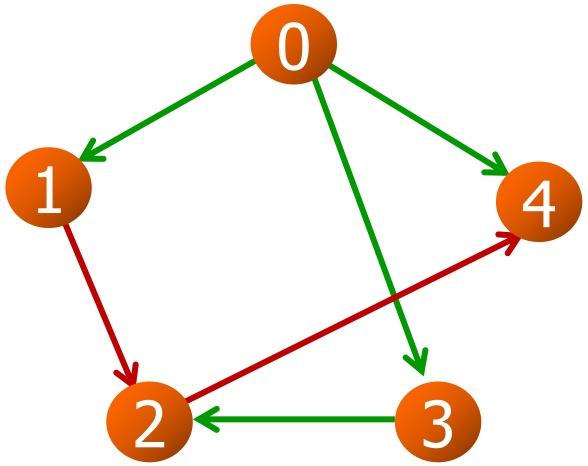
7.1 图的定义和术语

三. 图的连通

- 路径：是一个从顶点x到y的顶点序列(x, v_{i1}, v_{i2}, …, v_{im}, y)，其中,(x, v_{i1}),…(v_{ij-1}, v_{ij}),…(v_{im}, y)皆属于E。
- 路径上边或弧的数目称为该路径的长度。



1到3有路径
(1,0,4,3)

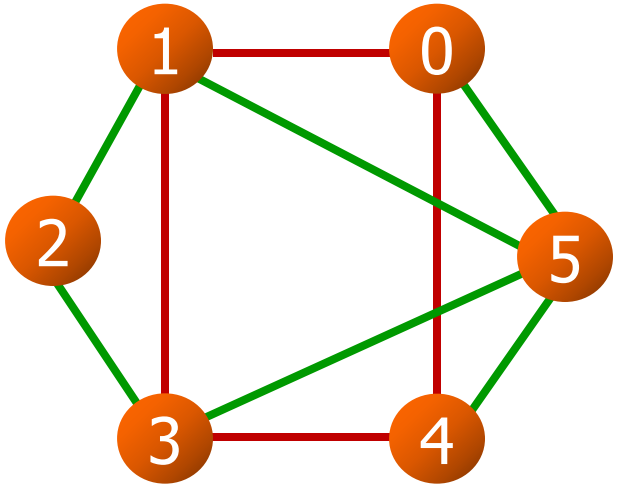


1到4有路径
(1,2,4)

7.1 图的定义和术语

三. 图的连通

- **回路或环**：路径的开始顶点与最后一个顶点相同，即路径中$(x, v_{i1}, v_{i2}, \cdots, v_{im}, y)$有x=y。
- **简单路径**：路径的顶点序列中，顶点不重复出现。



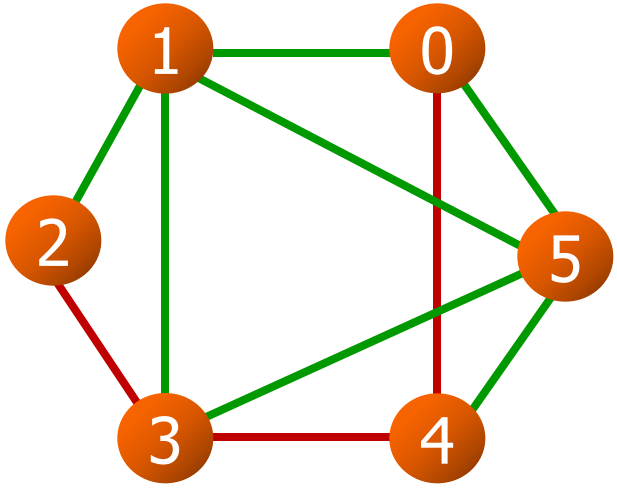
1到3是简单路径
(1,0,4,3)

1到1构成环
(1,0,4,3,1)

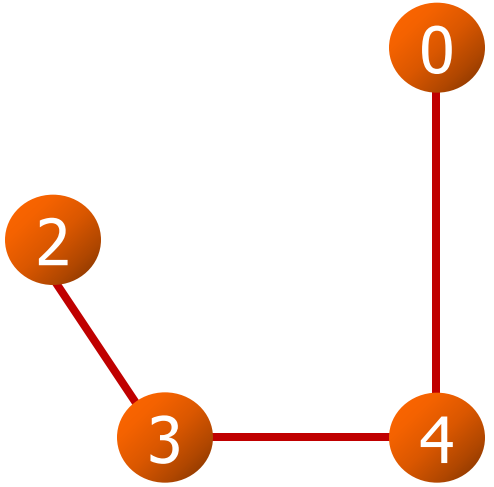
7.1 图的定义和术语

三. 图的连通

- 子图：设有两个图 G＝(V, E) 和 G’＝(V’, E’)。若 V’ ⊆ V 且 E’ ⊆ E, 称图G’是图G的子图。



图G

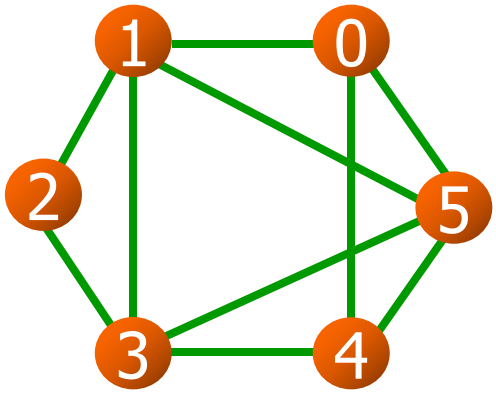


G的一个子图G’

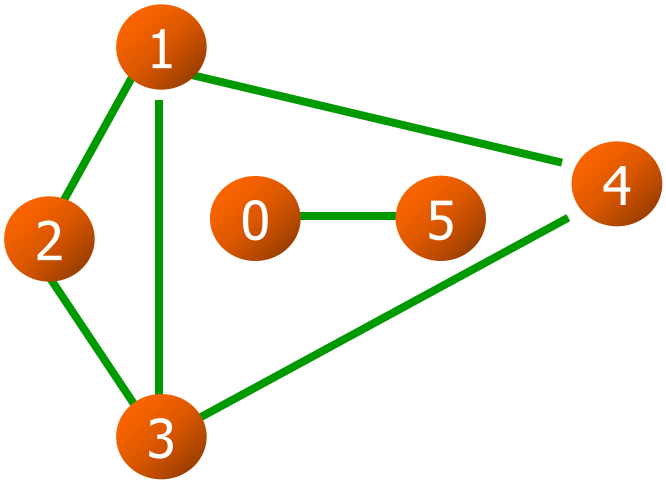
7.1 图的定义和术语

三. 图的连通

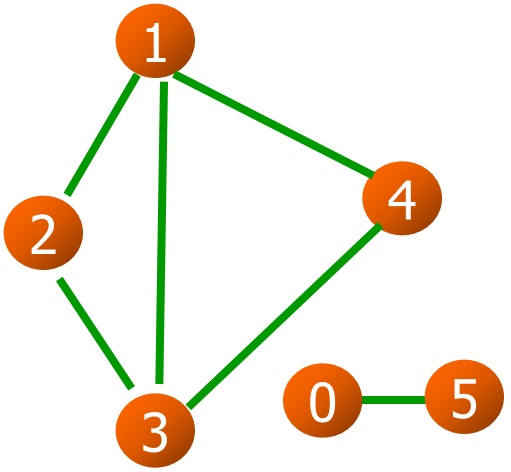
- **连通**：如果顶点x到y有路径，称x和y是连通的。
- **连通图**：任意两个顶点都连通的无向图。
- **连通分量**：是指无向图中的极大连通子图（这里极大指的是对子图再增加图G中的其它顶点，子图就不再连通了）。



连通图G1



非连通图G2

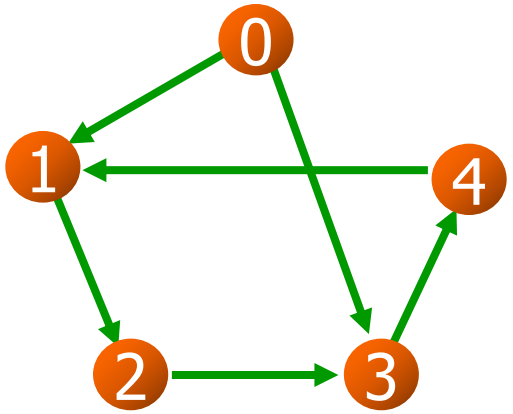


G2的两个连通分量

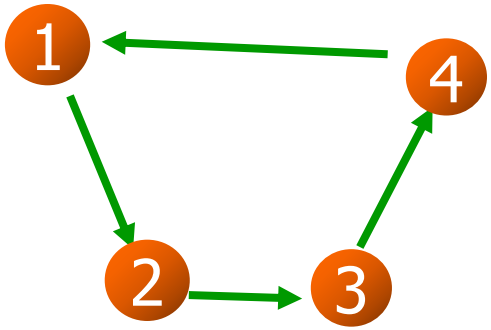
7.1 图的定义和术语

三. 图的连通

- **强连通**：有向图中两个顶点能够**相互到达**，称其强连通。
- **强连通图**：有向图中任意两顶点都存在路径。
- **强连通分量**：有向图中的极大强连通子图。



非强连通图G

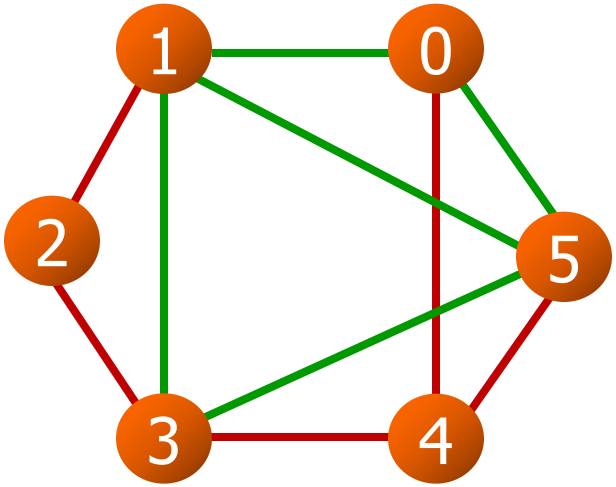


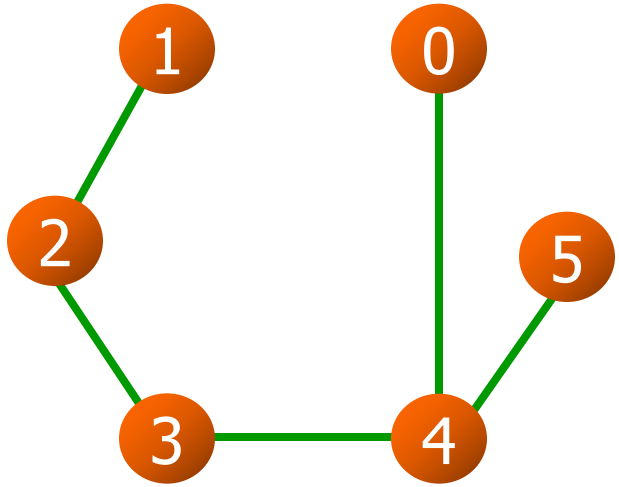
G的一个强连通分量

7.1 图的定义和术语

四. 图的生成树

- 生成树：一个连通图的生成树是一个极小连通子图，它含有图中全部n个顶点，但只有足以构成一棵树的n-1条边。
- 特点：一颗有n个结点的生成树有且仅有n-1条边。





7.1 图的定义和术语

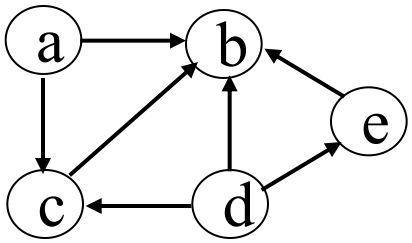
四. 图的生成树

- 无向图生成树的结论
  - 一棵有n个顶点的生成树有且仅有n-1条边
  - 如果一个图有n个顶点和小于n-1条边，则是非连通图
  - 如果多于n-1条边，则一定有环
  - 有n-1条边的子图不一定是生成树

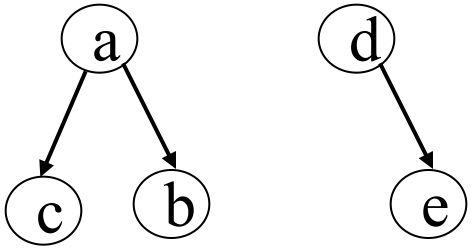
7.1 图的定义和术语

四. 图的生成树

- 有向图的生成森林
  - 有向树是只有一个顶点的入度为0，其余顶点的入度均为1的有向图。
  - 有向图的生成森林由若干棵有向树组成，含有图中全部顶点，但是只有足以构成若干棵不相交的有向树的弧。



(a) 有向图



(b) 有向树及生成森林

练习

1. 已知无向图G=(V,E)，其中V={a,b,c,d,e,f,g}，

E={(a,b),(a,c),(c,d), (d,e), (d,f),

(b,f),(e,f),(f,g), (g,c)}

① 请画出该图的连通分量；

② 请画出该图的生成树。

7.2 图的存储结构

- 图的结构复杂，任意两个顶点之间都可能存在联系，因此无法以数据元素在存储空间中的物理位置来表示元素之间的联系，即图没有顺序映像的存储结构。
- 需要设计恰当的**结点结构**和表示**边或弧的结构**。

7.2 图的存储结构

一. 邻接矩阵（数组表示法）

- 用数组分别存储顶点和边或弧
- 邻接矩阵，Adjacency Matrix，记录图中各顶点之间关系的二维数组
- 对于不带权的图，以1表示两顶点存在边(或弧)(相邻接)，以0表示两顶点不邻接，即

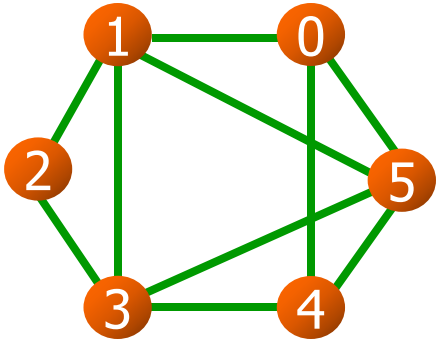
$$A[i][j] = \begin{cases} 1 & \text{如果}(i,j)\in E \text{ 或 } \langle i,j\rangle\in E \\ 0 & \text{其它} \end{cases}$$

7.2 图的存储结构

一. 邻接矩阵

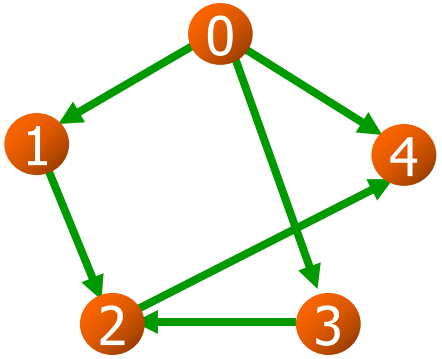
- 无向图的邻接矩阵

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 & 0 \end{pmatrix}$$



有向图的邻接矩阵

$$\begin{pmatrix} 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$



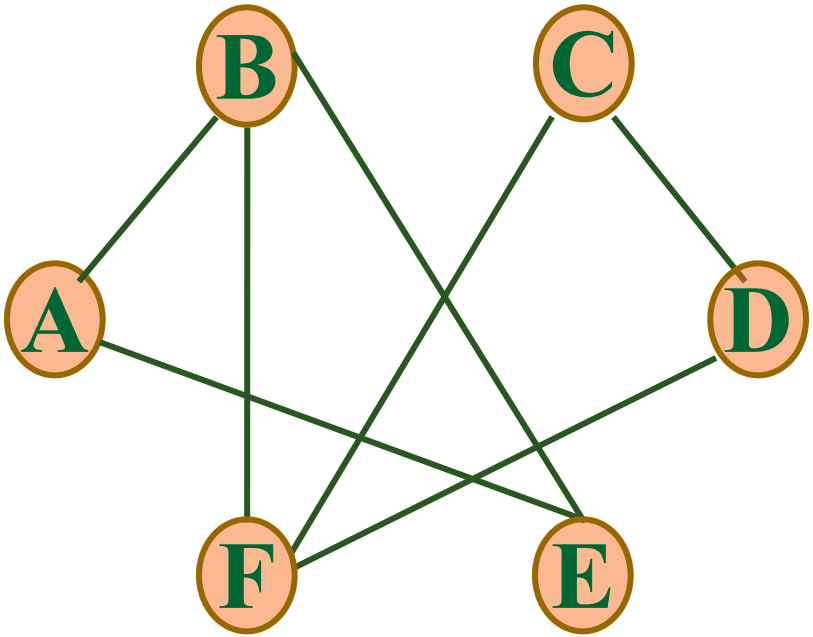
7.2 图的存储结构

一. 邻接矩阵

- 邻接矩阵的性质：
  - 无向图的邻接矩阵是对称的
  - 其第i行1的个数或第i列1的个数，等于顶点i的度TD(i)
  - 有向图的邻接矩阵可能是不对称的
  - 其第i行1的个数等于顶点i的出度OD(i)，第j列1的个数等于顶点j的入度ID(j)
  - 适合稠密图的存储，存储空间$O(n^2)$

练习

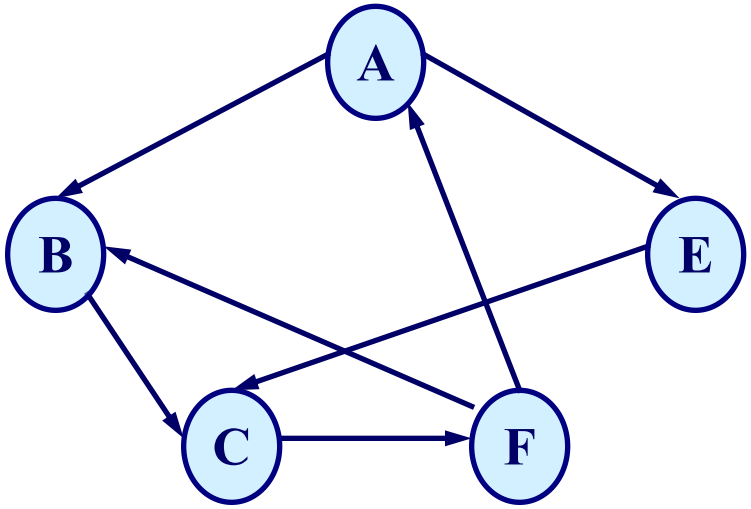
写出下图的邻接矩阵表示。



$$\begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 \end{pmatrix}$$

练习

写出下图的邻接矩阵表示。



$$\begin{pmatrix} 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

7.2 图的存储结构

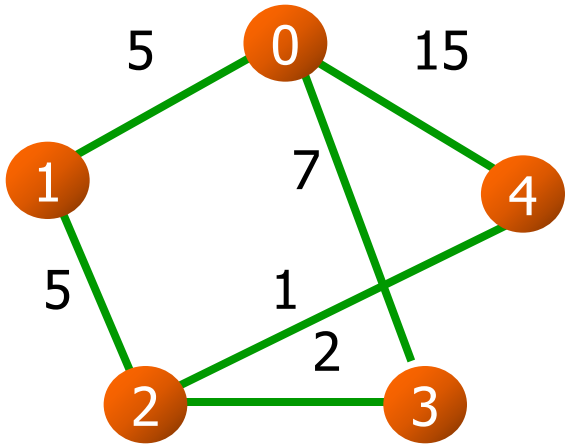
一. 邻接矩阵

- 网络：即有权值的图。在网络中，两个顶点如果不邻接，则被视为距离（即权值）为无穷大；如果邻接，则两个顶点之间存在一个距离值

$$A[i][j] = \begin{cases} w_{i,j} & \text{如果}(i,j)\in E \text{ 或 } \langle i,j\rangle \in E \\ \infty & \text{其它} \end{cases}$$

练习

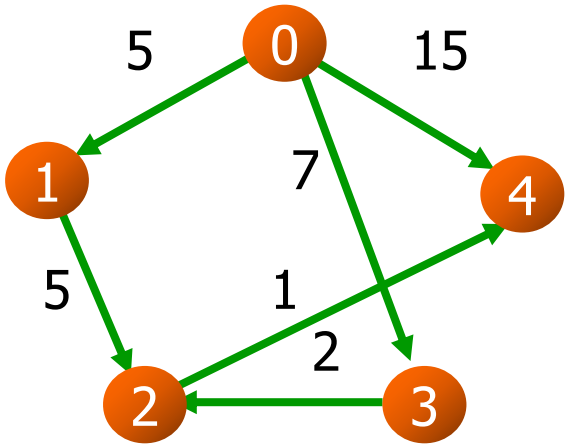
- 已知无向网N={V,E}，V={0,1,2,3,4}，E={(0,1,5)，(0,3,7)，(0,4,15)，(1,2,5)，(2,4,1)，(3,2,2)}，E中每个三元组的第三个元素表示权。
  - 画出该网
  - 计算每个结点的度
  - 写出该网的邻接矩阵



$$\begin{pmatrix} \infty & 5 & \infty & 7 & 15 \\ 5 & \infty & 5 & \infty & \infty \\ \infty & 5 & \infty & 2 & 1 \\ 7 & \infty & 2 & 1 & \infty \\ 15 & \infty & \infty & \infty & \infty \end{pmatrix}$$

练习

- 有向网N={V,E}，V={0,1,2,3,4}，E={<0,1,5>，<0,3,7>，<0,4,15>，<1,2,5>，<2,4,1>，<3,2,2> }，E中每个三元组的第三个元素表示权。
  - 画出该网
  - 计算每个结点的入度、出度、度
  - 写出该网的邻接矩阵



$$\begin{pmatrix} \infty & 5 & \infty & 7 & 15 \\ \infty & \infty & 5 & \infty & \infty \\ \infty & \infty & \infty & \infty & 1 \\ \infty & \infty & 2 & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \end{pmatrix}$$

7.2 图的存储结构

二. 邻接表

- 邻接表，Adjacency List，是图的一种链式存储结构
  - 在邻接表中，每个顶点设置一个单链表，其每个表结点都是依附于该顶点的边（或以该顶点为尾的弧）
  - 边(弧)的结点结构
    - adjvex; // 该边(弧)所指向的顶点的位置
    - nextarc;// 指向下一条边(弧)指针
    - info; // 该边(弧)相关信息，如权值

adjvex	nextarc	info

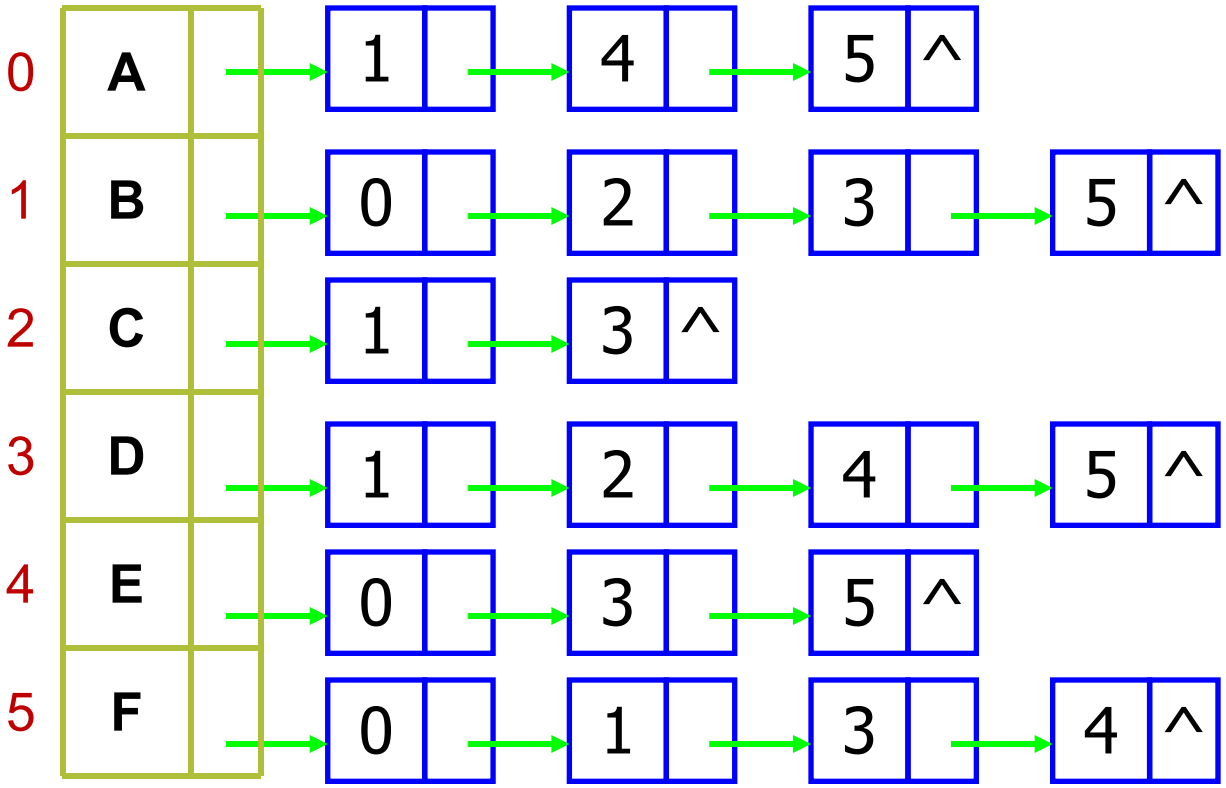
  - 顶点的结点结构
    - data; // 顶点信息
    - firstarc;//指向第一条依附该顶点的边(弧)

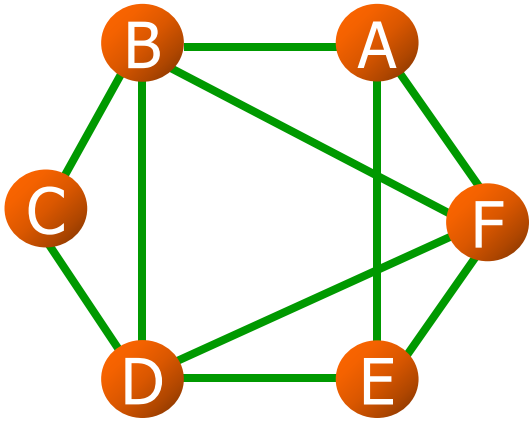
data	firstarc

7.2 图的存储结构

二. 邻接表

- 无向图的存储示意图

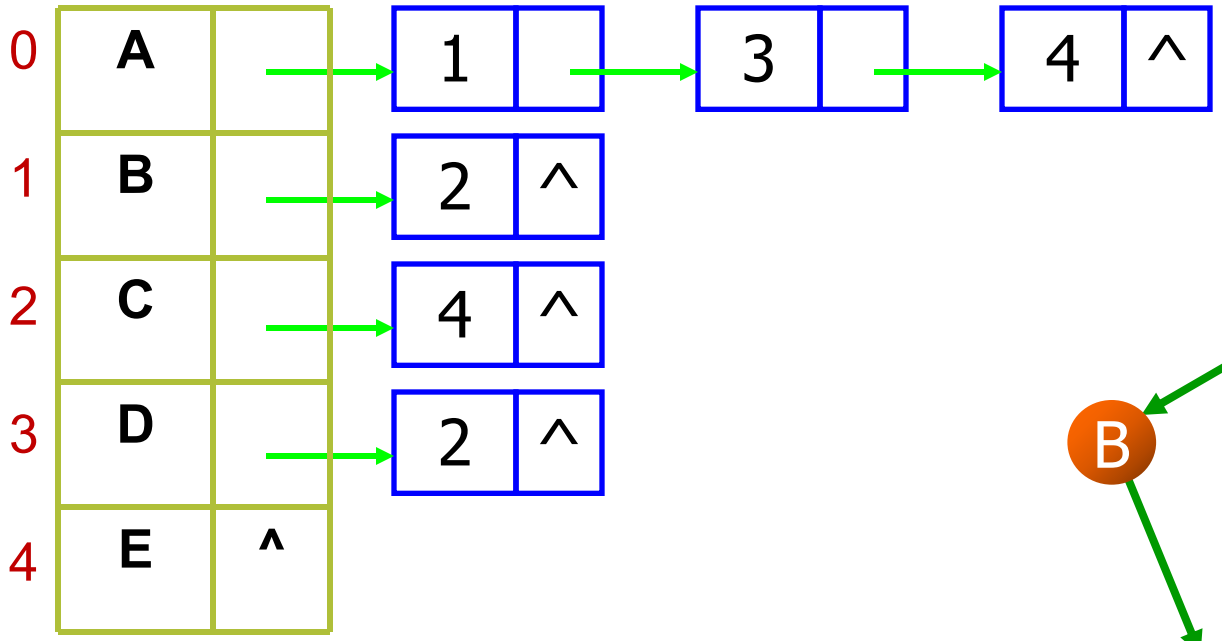


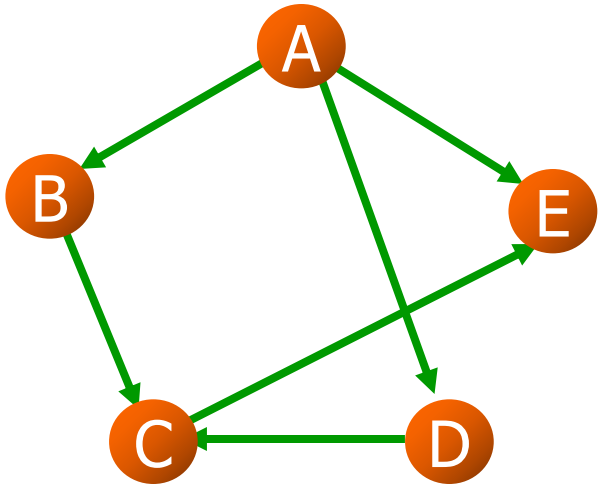


7.2 图的存储结构

二. 邻接表

- 有向图的存储示意图

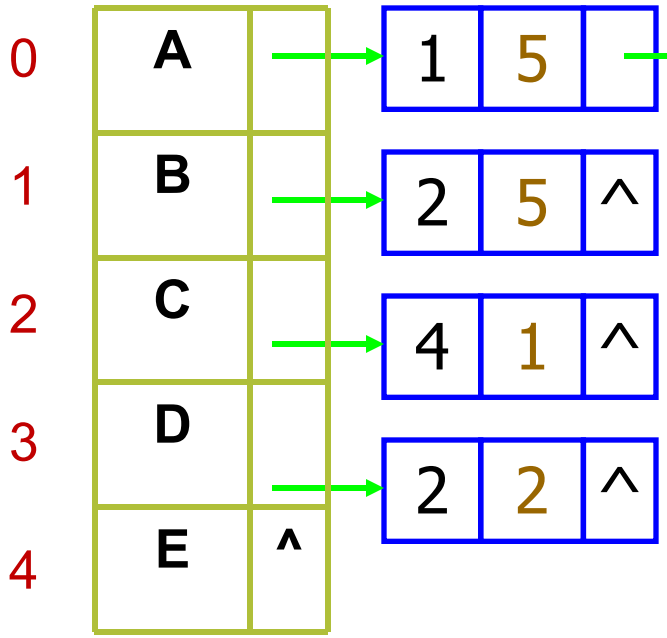


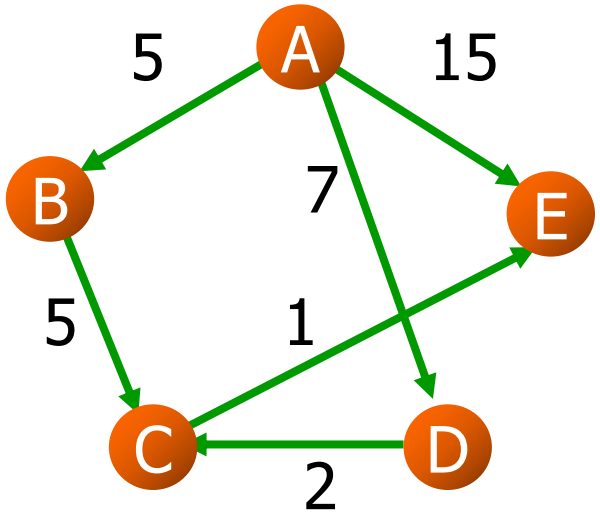


7.2 图的存储结构

二. 邻接表

- 网的存储示意图

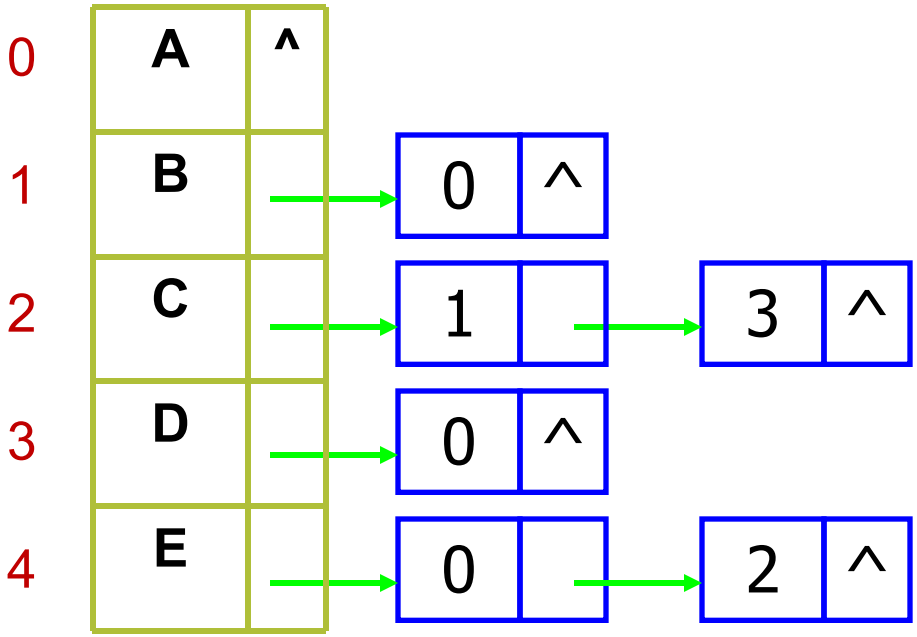


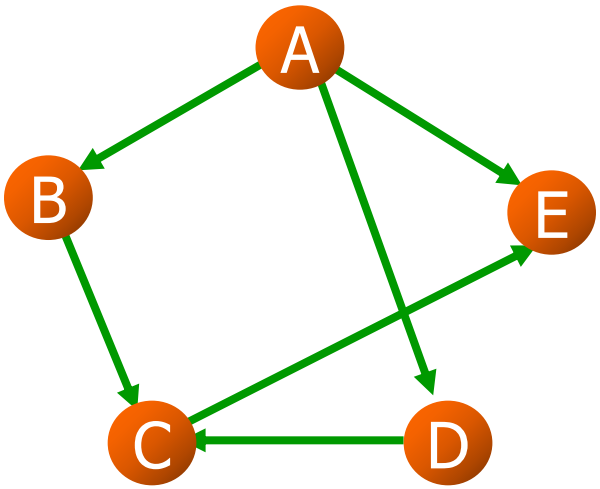


7.2 图的存储结构

二. 邻接表

- 有向图的逆邻接表，弧的箭头向内





7.2 图的存储结构

二. 邻接表

- 对于有向图的邻接表，其第i个链表中结点的个数只是该顶点的出度；逆邻接表则表示入度
- 要判定两个顶点i和j是否有边（或弧），必须搜索整个第i个和第j个链表，没有邻接矩阵方便

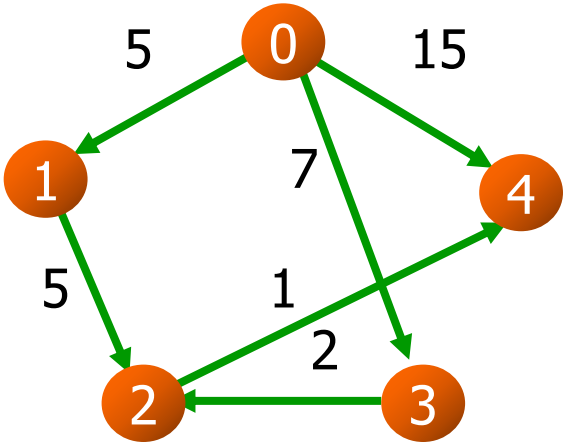
7.2 图的存储结构

二. 邻接表

- 邻接表
  - 优点：能直接得到与顶点i相邻接的其它顶点；适合稀疏图的存储，存储空间O(n+e)
  - 要搜索整个表才能知道两个顶点间是否有边或者弧
- 邻接矩阵
  - 优点：直接知道两顶点间是否有连线；适合稠密图的存储，存储空间$O(n^2)$
  - 缺点：要搜索才能得到与顶点i相邻接的其它顶点

练习

- 有向网N={V,E}，V={0,1,2,3,4}，E={<0,1,5>，<0,3,7>，<0,4,15>，<1,2,5>，<2,4,1>，<3,2,2> }，E中每个元组的第三个元素表示权。
  - 画出该网
  - 计算每个结点的入度、出度
  - 写出该网的邻接表和逆邻接表



7.3 图的遍历

一. 遍历的含义

- 从图的某一顶点开始，访问图中其余顶点，且使每一个顶点仅被访问一次
- 图的遍历主要方式
  - 深度优先搜索 **DFS** - Depth_First Search
  - 广度优先搜索 **BFS** - Broadth_First Search

7.3 图的遍历

二. 深度优先搜索

- 图中可能存在回路，且图的任一顶点都可能与其它顶点相通，在访问完某个顶点之后可能会沿着某些边又回到了曾经访问过的顶点。
- 为了避免重复访问，可设置一个辅助数组 visited [n]

标志顶点是否被访问过， visited [i]初始值设为FALSE，一旦顶点v_i被访问过，将visited [i] 设为TRUE。

7.3 图的遍历

二. 深度优先搜索

- 图的深度优先搜索过程：

（1）从图中某个初始顶点v出发，首先访问该顶点；然后选择v的一个相邻且没有被访问过的顶点w，再从w出发进行深度优先搜索，以此类推，直到图中所有与顶点v有路径相通的顶点都被访问过为止；

（2）如果图中仍有未被访问的顶点，则以该顶点为起始点，重复上述过程，直到图中所有顶点都被访问过为止。

7.3 图的遍历

二. 深度优先搜索

- 算法流程：
  - 所有顶点访问标志visited[]设置为FALSE
  - 从某顶点v_0开始，设v=v_0

    1. 如果visited[v]==FALSE，则访问该顶点v，并将顶点v压入栈中，且设visited[v]=TRUE

    2. 如果找到当前顶点的一个新的相邻顶点w

    （即visited[w] == FALSE），设v=w，重复步骤1

    3. 否则(说明当前顶点的所有相邻顶点都已被访问过，或者当前顶点没有相邻顶点)，如果当前顶点是v_0，退出；否则返回上一级顶点(即从栈中弹出一个顶点)，重复步骤2。

  - 用递归方式实现

7.3 图的遍历

二. 深度优先搜索

- 算法实现：

```
bool visited[MAX_VERTEX_NUM];    // 访问标志数组
Status (* VisitFunc)(int v);     // 函数变量

void DFSTraverse(Graph G, Status (*Visit)(int v)) {  // 算法7.4
 // 对图G作深度优先遍历。
 int v;
 VisitFunc = Visit; // 使用全局变量VisitFunc，使DFS不必设函数指针参数
 for (v=0; v<G.vexnum; ++v) visited[v] = false; // 访问标志数组初始化
 for ( v=0; v<G.vexnum; ++v )
   if (!visited[v]) DFS(G, v);          // 对尚未访问的顶点调用DFS
}
```

7.3 图的遍历

二. 深度优先搜索

- 算法实现：

```
void DFS(Graph G, int v) {  // 算法7.5
   // 从第v个顶点出发递归地深度优先遍历图G。
   int w;
   visited[v] = true;   VisitFunc(v);  // 访问第v个顶点
   for ( w=FirstAdjVex(G, v);  w>=0;  w=NextAdjVex(G, v, w) )
      if (!visited[w])   // 对v的尚未访问的邻接顶点w递归调用DFS
         DFS(G, w);
}
```

7.3 图的遍历

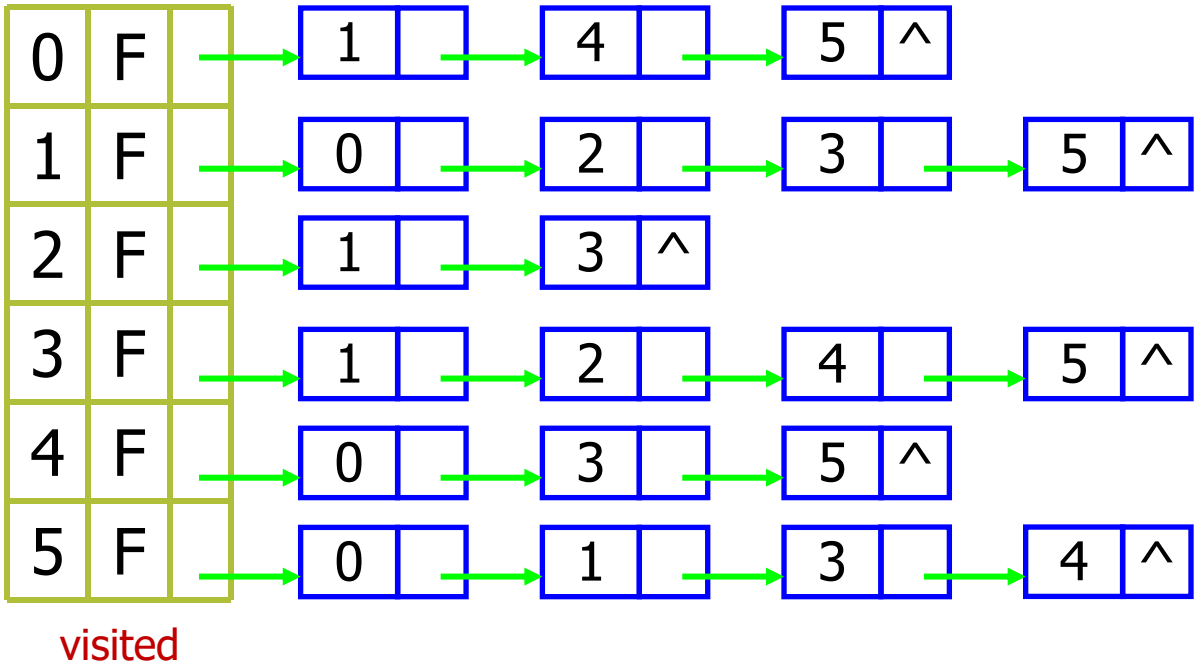
二. 深度优先搜索

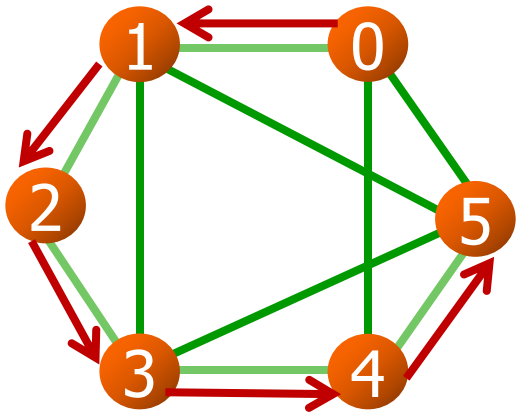
- 特点：
  - 图的深度优先搜索是树的先根遍历的推广。
  - 图的深度优先遍历的过程实质上是对每个顶点查找其邻接点的过程，其计算工作量取决于图所采用的的存储结构：
    - 若用邻接矩阵存储，其时间复杂度为$O(n^2)$
    - 若用邻接表存储，其时间复杂度为$O(n+e)$

7.3 图的遍历

二. 深度优先搜索

- 采用以下邻接表存储结构时，DFS次序为0,1,2,3,4,5

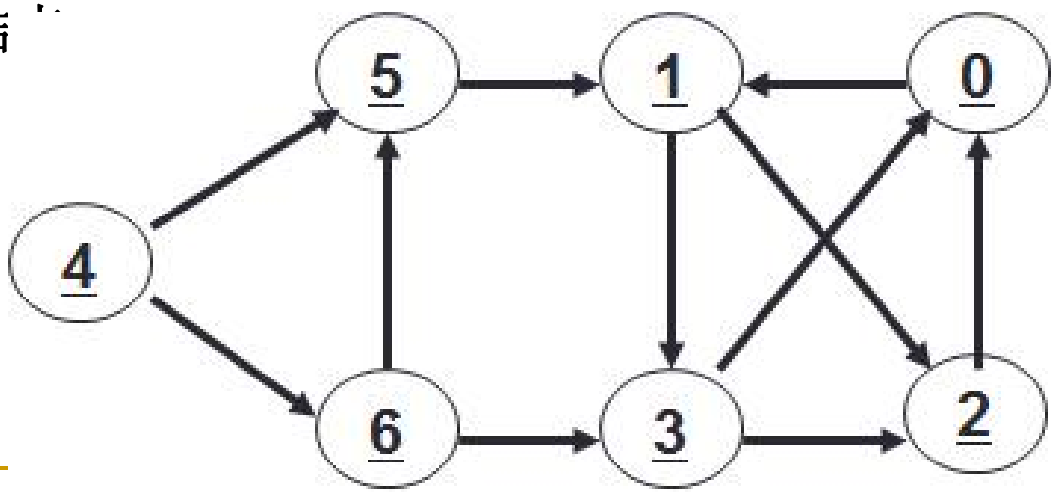




7.3 图的遍历

二. 深度优先搜索

- 有向图的DFS推导：0 1 2 3 4 5 6
  - 从0开始访问，接着只能访问1
  - 从1到2，2无相邻未访问结点，跳转回1
  - 从1到3，3无相邻未访问结点，跳转回1
  - 1无相邻未访问结点，跳转回0
  - 0无相邻未访问结点，跳转到未访问新结点4
  - 从4到5，5无相邻未访问结点，跳转回4
  - 从4到6，6无相邻未访问结
  - 全部结点已访问，结束



7.3 图的遍历

三. 广度优先搜索

- **广度优先搜索**，是一种层次遍历搜索方法，过程：

（1）访问初始顶点v，接着访问它的所有未被访问过的邻接顶点v_1, v_2，……，v_t；按照v_1, v_2，……，v_t的次序，访问每一个顶点的所有未被访问的邻接顶点。依次类推，直到图中所有和初始点v相连通的顶点都被访问为止。

（2）重复上述过程，直到图中所有顶点都被访问为止。

- 若顶点x先于顶点y被访问，则顶点x的邻接点也先于顶点y的邻接点被访问，所以广度优先搜索邻接点的过程具有先进先出的特征，可以使用队列存储刚刚访问的顶点。

7.3 图的遍历

三. 广度优先搜索

- 广度优先搜索算法
  - 所有顶点访问标志visited[]设置为FALSE
  - 从某顶点v_0开始，访问v_0，visited[v_0]=TRUE，将v_0插入队列Q

1. 如果队列Q不空，则从队列Q头上取出一个顶点v，否则结束

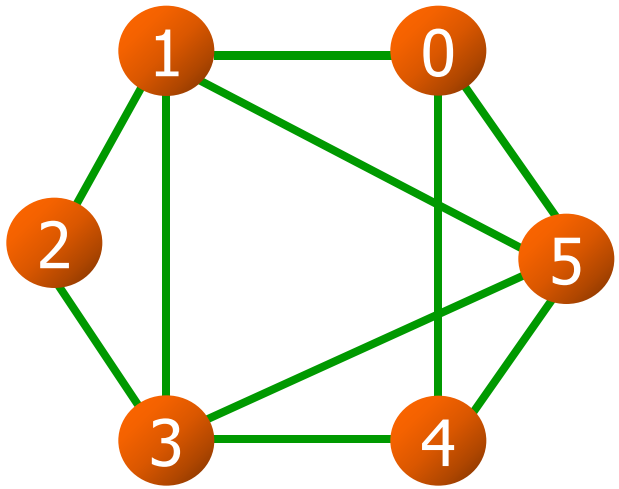
2. 依次找到顶点v的所有相邻顶点v'，如果visited[v']==FALSE，则访问该顶点v'，然后将v'插入队列Q，并使visited[v']=TRUE，

3. 重复1，2

7.3 图的遍历

三. 广度优先搜索

- 无向图举例



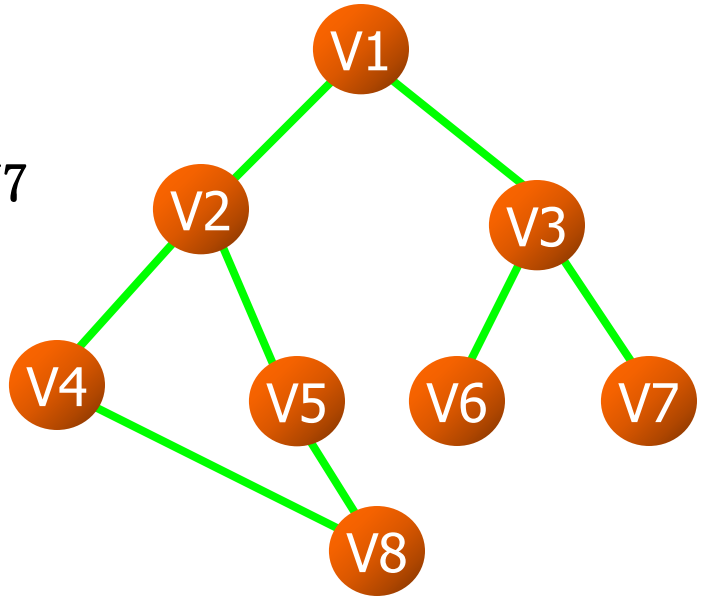
- 0入队
- 0出队，1、4、5入队，
  队列：1 4 5
- 1出队，2、3入队，
  队列：4 5 2 3
- 4 5 2 3 依次出队，结束

BFS为： 0, 1, 4, 5, 2, 3

7.3 图的遍历

三. 广度优先搜索

- 无向图举例
  - V1入队
  - V1出队，V2、V3入队，队列：V2 V3
  - V2出队，V4、V5入队，队列：V3 V4 V5
  - V3出队，V6、V7入队，队列：V4 V5 V6 V7
  - V4出队，V8入队，队列：V5 V6 V7 V8
  - V5 V6 V7 V8依次出队，结束

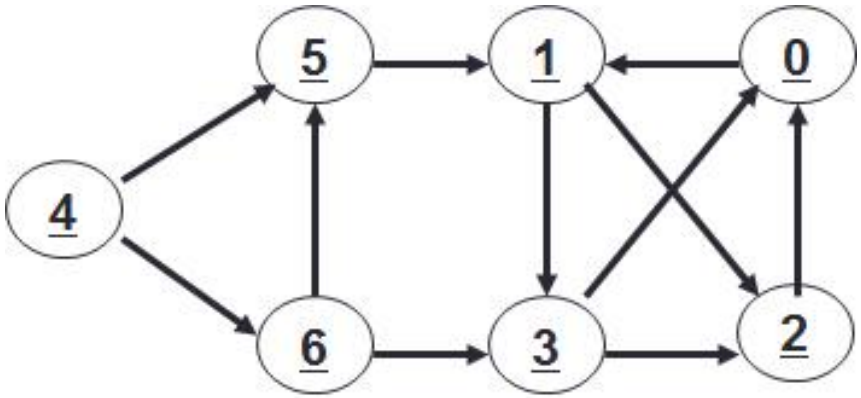


BFS为：
v1, v2, v3, v4, v5, v6, v7, v8

7.3 图的遍历

三. 广度优先搜索

- 有向图举例



- 0入队
- 0出队，1入队，队列：1
- 1出队，2、3入队，队列：2 3
- 2 3依次出队
- 搜索未访问结点，4入队
- 4出队，5、6入队，队列：5 6
- 5 6依次出队，结束

BFS为 0, 1, 2, 3, 4, 5, 6

7.3 图的遍历

三. 广度优先搜索

- BFS算法实现

```
void BFSTraverse(Graph G, Status (*Visit)(int v )) {//  算法7.6
  // 按广度优先非递归遍历图G。使用辅助队列Q和访问标志数组visited。
  QElemType v,w;
  queue Q;
  QElemType u;
  for (v=0; v<G.vexnum; ++v) visited[v] = FALSE;
  InitQueue(Q);                          // 置空的辅助队列Q
  for (v=0;  v<G.vexnum;  ++v)
    if (!visited[v]) {                   // v尚未访问
      visited[v] = TRUE;  Visit(v); // 访问v
      EnQueue(Q, v);                     // v入队列
      while (!QueueEmpty(Q)) {
        DeQueue(Q, u);                   // 队头元素出队并置为u
```

（续下页）

7.3 图的遍历

三. 广度优先搜索

- BFS算法实现

```
（接上页）
for (w=FirstAdjVex(G, u);  w>=0;  w=NextAdjVex(G, u, w))
          if (!visited[w]) {          // u的尚未访问的邻接顶点w入队列Q
            visited[w] = TRUE;  Visit(w);
            EnQueue(Q, w);
          }//if
      }//while
    }//if
} // BFSTraverse
```

7.3 图的遍历

二. 广度优先搜索

- 特点：
  - 图的广度优先搜索类似于树的层次遍历的过程。
  - 图的广度优先遍历的过程实质上也是对每个顶点查找其邻接点的过程，其计算工作量与深度优先搜索遍历相同，两者不同仅在于对顶点的访问顺序不同。

7.3 图的遍历

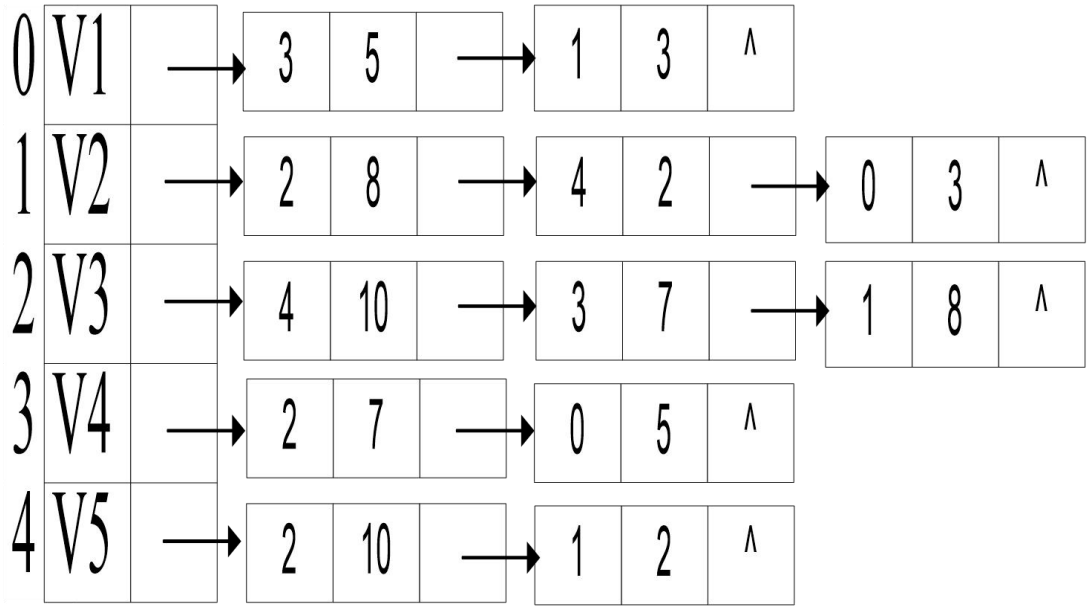
- 可以看出无论是深度优先遍历还是广度优先遍历,其实质都是通过边或弧找邻接点的过程,只是访问的顺序不同。
- 两者的时间复杂度相同,取决于采取的存储结构，若用邻接矩阵为$O(n^2)$,若用邻接表则为$O(n+e)$

7.3 图的遍历

- 如果图为连通图，则从该图的任意一个顶点开始执行一次深度优先遍历或广度优先遍历，即可访问该连通图的所有顶点。
- 如果图为非连通图，则依次从未访问过的顶点开始执行深度优先遍历或广度优先遍历，直至所有的顶点均被访问。
- 事实上执行一次深度或广度优先可以遍历一个连通分量。图中有多少个连通分量，就调用多少次深度或广度优先遍历。

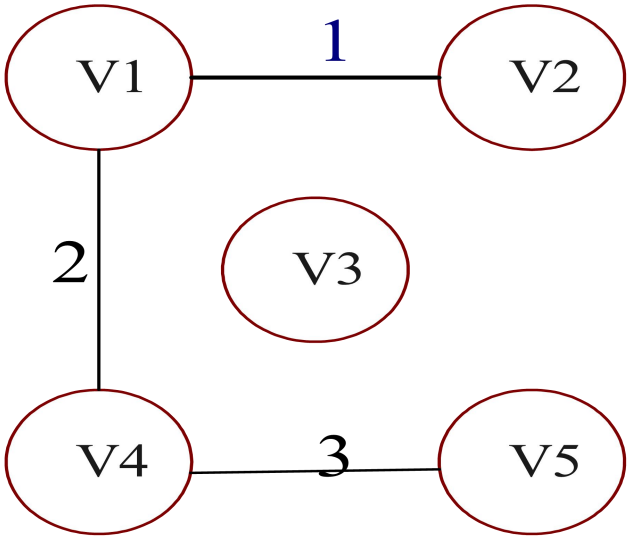
7.3 图的遍历

练习1：假设无向网G的邻接表表示如下图，分别写出深度优先遍历和广度优先遍历的顶点序列。



7.3 图的遍历

练习2：假设用邻接表存储，下图中边上序号表示边输入顺序(链表头插入)，画出该图邻接表，写出其深度优先遍历序列和广度优先遍历序列。



无向图G5