《计算机算法设计与分析》

第二章 图与遍历算法

马丙鹏

2020年09月17日

中国科学院大学
University of Chinese Academy of Sciences

第二章 图与遍历算法

- 2.1 图的基本概念和性质
- 2.2 图的遍历算法

2.1 图的基本概念和性质

- **1. 栈和队列**
  - 共性
    - n个元素的特殊的有序表
    - 都可以利用静态(动态)数据结构—数组(链表)实现
  - 栈的特点
    - 所有的插入和删除都在栈顶的一端进行
    - 后进先出表, 最后入栈的元素将首先被移出
  - 队列的特点
    - 所有的插入只在尾部的一端进行，所有的删除只能在前部的另外一端进行
    - 先进先出表，第一个出入队列中的元素也第一个被移出

2.1 图的基本概念和性质

- 1. 栈和队列
  - 栈的数组表示
    - 用一维数组STACKS(1:n)表示
    - 栈底: STACKS(1)
    - 第i个元素STACKS(i)
    - 栈顶指针: top

2.1 图的基本概念和性质

■ 1. 栈和队列

□栈的数组表示

栈运算: 插入一个元素

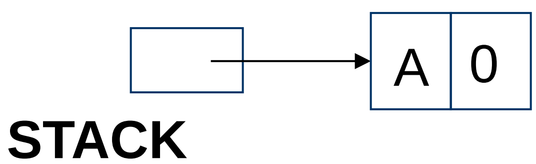
```
procedure ADD(item, STACK, n, top)
  if top ≥ n then call STACKFULL endif
  top← top +1
  STACK(top) ← item
end ADD
```

栈运算: 删除一个元素

```
procedure DELETE(item, STACK, top)
  if top ≤ 0 then call STACKEMPTY endif
  item ← STACK(top)
  top ← top-1
end DELETE
```

2.1 图的基本概念和性质

- **1. 栈和队列**
  - 栈的链接表示
    - 一种单向链接表
    - 每个结点有两个信息段, DATA和LINK
    - DATA存放数据
    - LINK指向前一节点



2.1 图的基本概念和性质

■ 1. 栈和队列

□栈的链接表表示

➢结点的插入和删除

结点的插入

call GETNODE(T)
DATA(T) ←item
LINK(T) ←STACK
STACK ← T

结点的删除

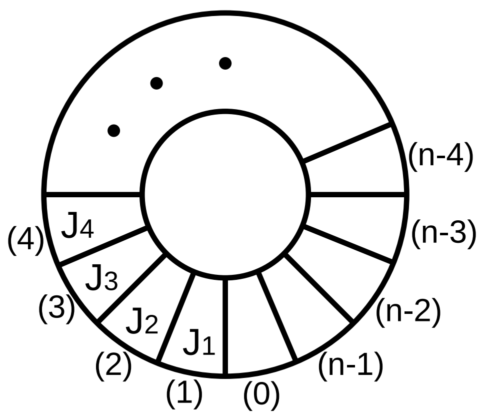
item ←DATA(STACK)
T ←STACK
STACK ←LINK(SATCK)
call RETNODE(T)

2.1 图的基本概念和性质

■ 1. 栈和队列

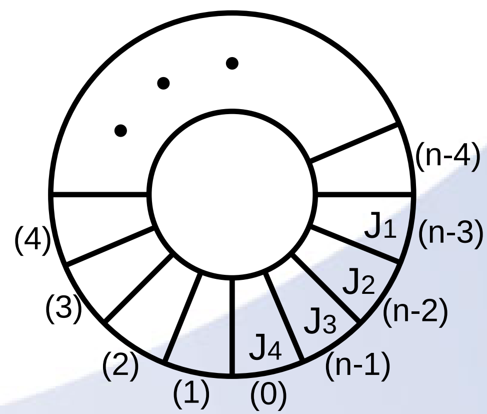
□队列的数组表示

- 当队列用数组表示时,可以当成一个环形来看待
- 插入数组按照顺时针方向将**rear**移到下一个空位置
- **Front**总指着队中第一个元素按照逆时针的前一个位置



front=0; rear=4

(a)



front=n-4; rear=0

(b)

2.1 图的基本概念和性质

■ 1. 栈和队列

□队列的数组表示

➢元素的插入和删除

队列运算: 插入一个元素

procedure ADDQ(item, Q, n, front, rear)
  rear ← (rear+1) mod n
if front=rear **then call** QUEUEFULL **endif**
  Q(rear)← item
end ADDQ

2.1 图的基本概念和性质

■ 1. 栈和队列

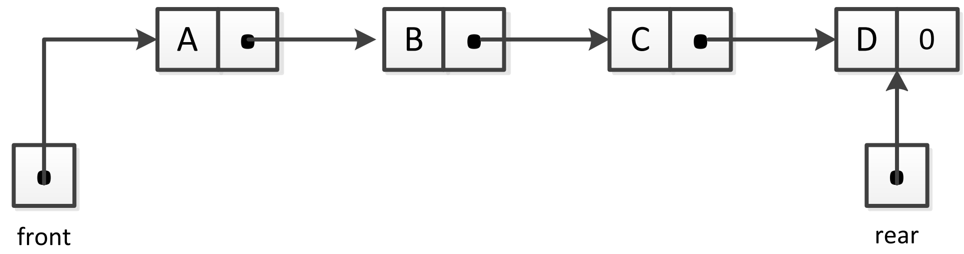
- 队列的数组表示
  - ➢元素的插入和删除

队列运算: 删除一个元素

```
procedure DELETEQ(item, Q, n, front, rear)
  if front = rear then call QUEUEEMPTY endif
   front ← (front+1) mod n
   item ← Q(front)
end DELETEQ
```

2.1 图的基本概念和性质

- 1. 栈和队列
  - 队列的链接表表示
    - 与栈的链接表表示类似
    - 指针front和rear 指示前部和尾部的位置



2.1 图的基本概念和性质

■ 2. 树

- 定义:

  树(tree)是一个或多个结点的有限集合，它使得:

  - 有一个指定为根(root)的结点
  - 剩余结点被划分成m≥0个不相交的子集合: $T_1, \ldots, T_m$,
  - 这些集合的每一个子集合又都是一棵树, 并称$T_1, \ldots,\ T_m$为根的子树。

2.1 图的基本概念和性质

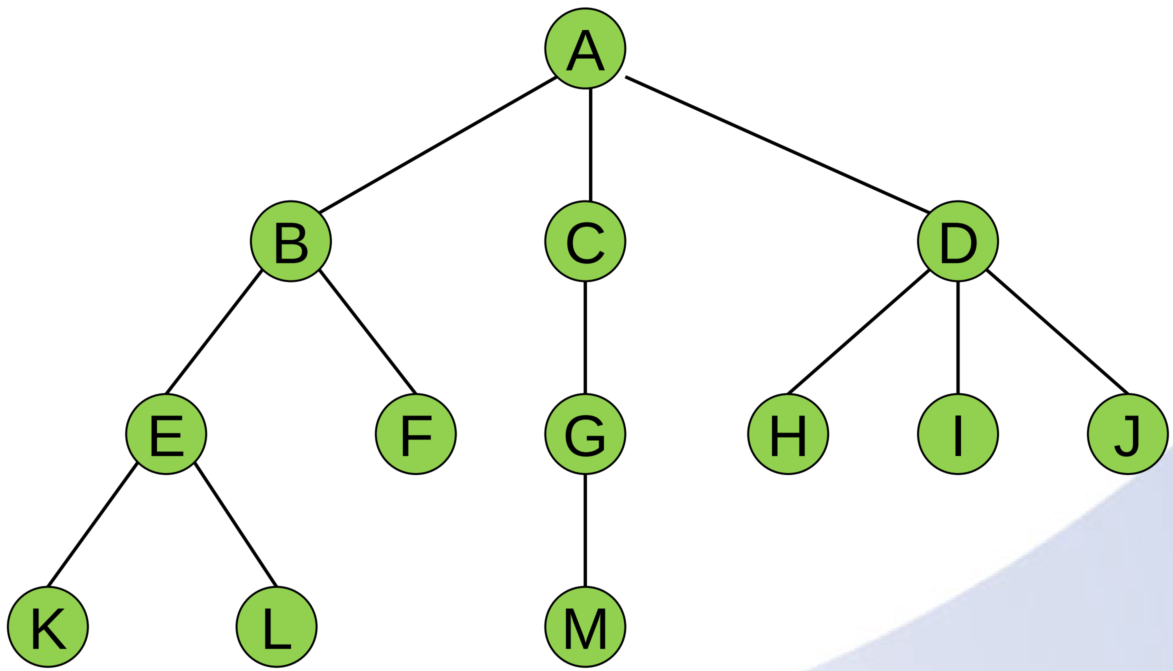
- 2. 树
  - 基本概念
    - 结点的度(degree)：一个结点的子树数
    - 树的度：树中结点度的最大值
    - 结点的级(level)(层)：设根是1级，若某结点在p级，则它的儿子在p+1级
    - 树的高度(深度)：树中结点的最大级数
    - 叶子(终端结点)：度为0的结点
    - 内结点(非终端结点)：度不为0的结点
    - 森林：m≥0个不相交树的集合

中国科学院大学
University of Chinese Academy of Sciences

2.1 图的基本概念和性质

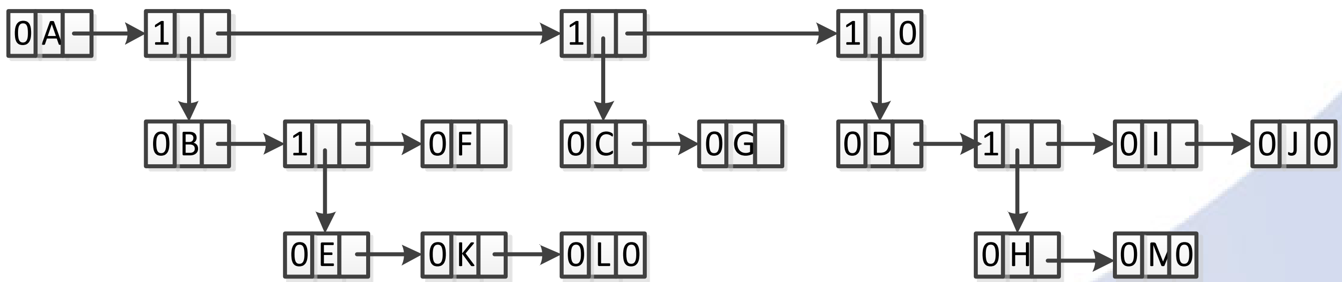
■ 2. 树

□基本概念



2.1 图的基本概念和性质

- **2. 树**
  - 树的表示方法：用链接表表示；
    - 每个结点三个信息段：**TAG, DATA, LINK**
    - **TAG=0, DATA**存数据
    - **TAG=1, DATA**存链接信息



2.1 图的基本概念和性质

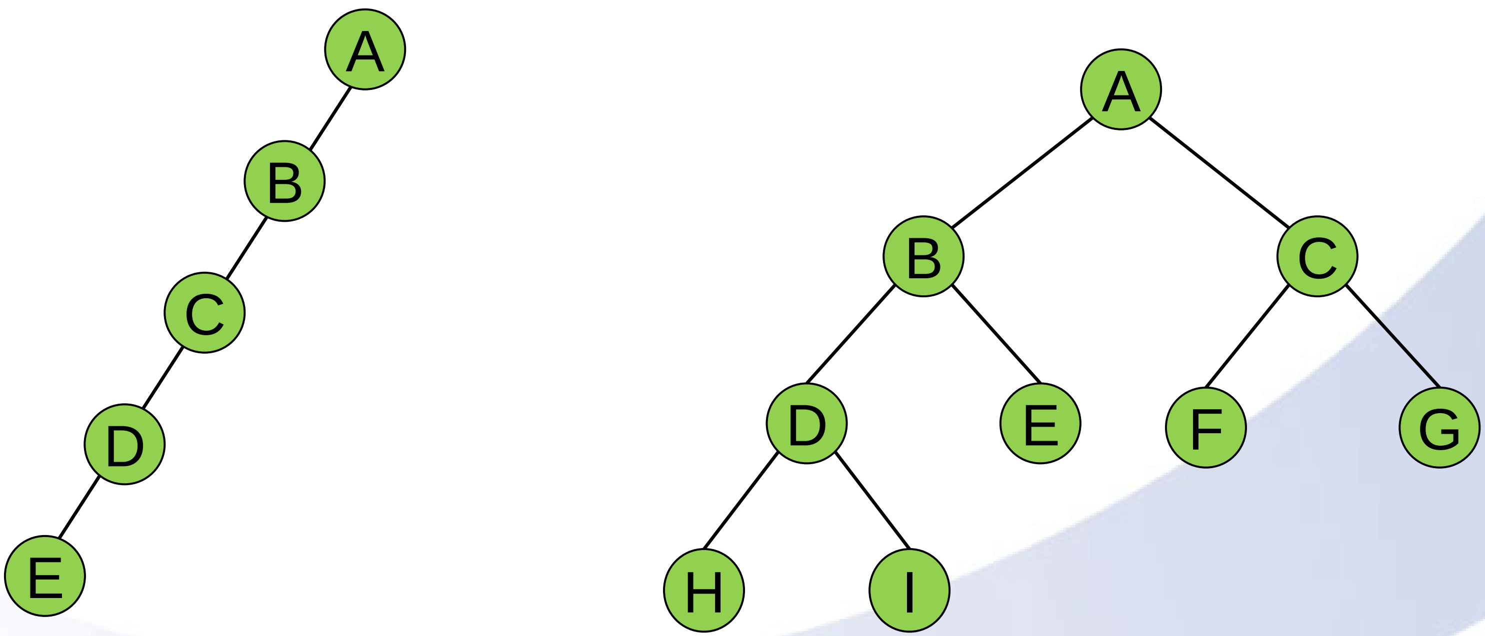
- 2. 树
  - 二元树
    - 定义2.1 二元树(binary tree)是结点的有限集合，它或者为空，或者由一个根和两棵称为左子树和右子树的不相交二元树所组成
  - 二元树性质1：
    - 引理2.1 一棵二元树第i级的最大结点数是2^{i-1}。深度为k的二元树的最大结点数为2^k-1, $k>0$。
  - 二元树和度为二的树的区别？
    - 二元树的子树有左右之分，而树的子树没有
    - 二元树允许有零个结点，而树至少一个结点

2.1 图的基本概念和性质

■ 2. 树

□特殊的二元树

➢斜树与完全二元树



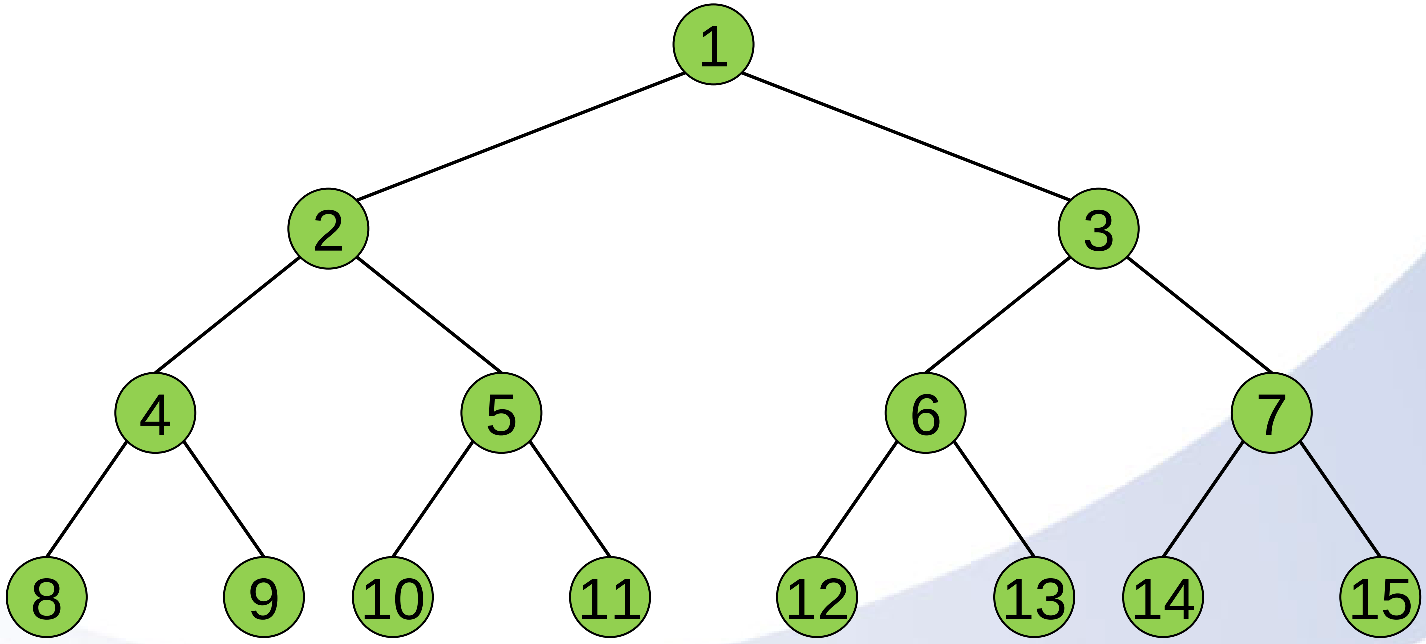
(a) (b)

2.1 图的基本概念和性质

■ 2. 树

□特殊的二元树

➢满二元树：深度为k且有2^k-1个结点的二元树



2.1 图的基本概念和性质

■ 2. 树

□特殊的二元树

➢完全二元树：

一棵有n个结点深度为k的二元树，当它的结点相当于深度为k的满二元树中编号为1到n的结点时，称该二元树是完全的。

➢完全二元树的叶子结点至多出现在相邻的两级上。

➢完全二元树的结点可以紧凑地存放在一个一维数组中(性质见引理2.1)。

2.1 图的基本概念和性质

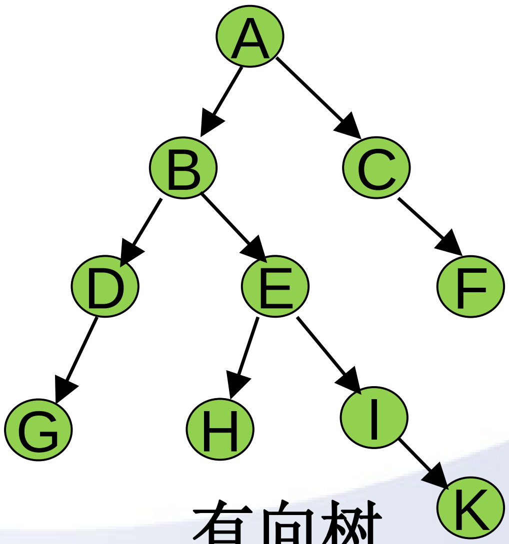
■ 2. 树

□有向树: 一棵有向树是满足下述条件的无圈有向图：

① 有一个称为根的顶点，它不是任何有向边的头；

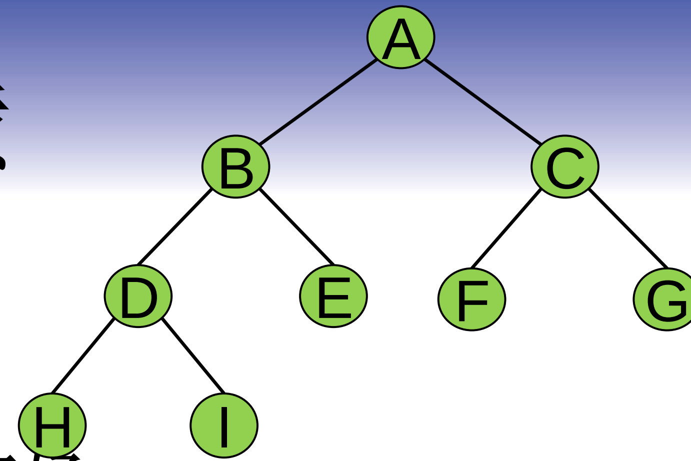
② 除了根以外，其余每个顶点的入度均为1；

③ 从根到每个顶点有一条有向路。



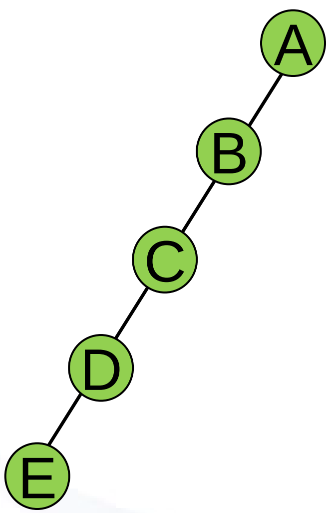
有向树

2.1 图的基本概念和性质



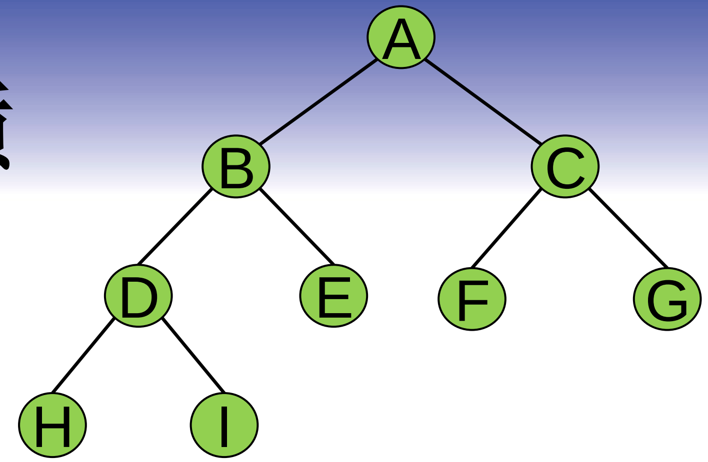
■ 2. 树

- 二元树的数组表示方法
  - ➢对于完全二元树，空间效率好；
  - ➢其他二元树，要浪费大量空间

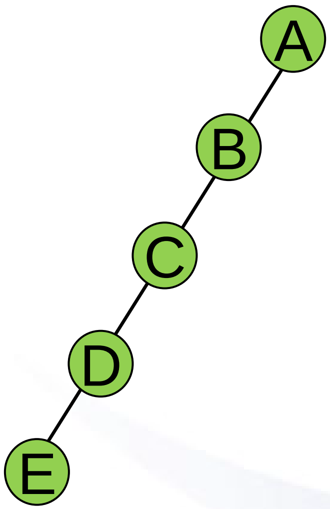


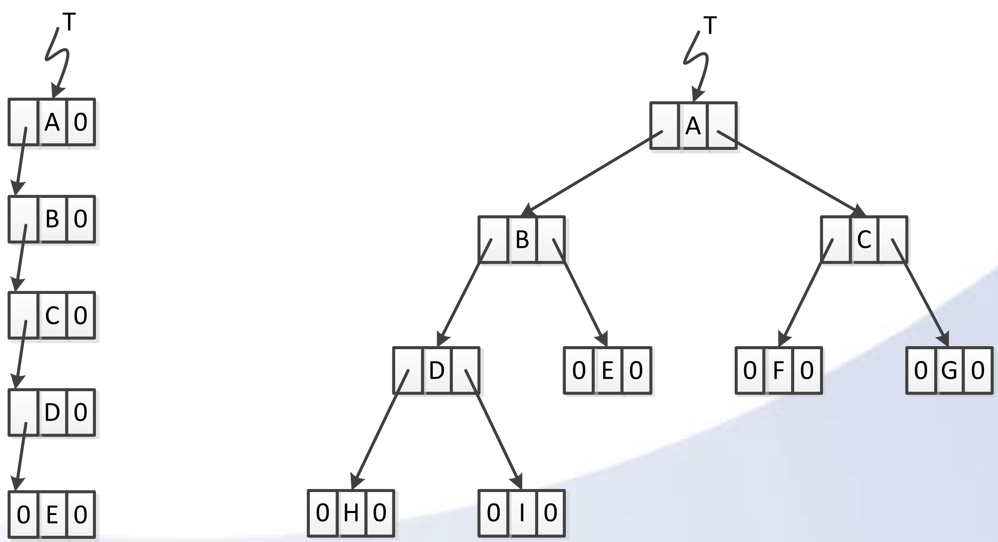
	TREE	TREE
(1)	A	A
(2)	B	B
(3)	--	C
(4)	C	D
(5)	--	E
(6)	--	F
(7)	--	G
(8)	D	H
(9)	--	I
⋮	⋮	
(16)	E	

2.1 图的基本概念和性质



- **2. 树**
  - 二元树的链表表示方法
    - 结构简单，有效。
    - 链表中每个结点有三个信息段, **LCHILD, DATA** 和**RCHILD**



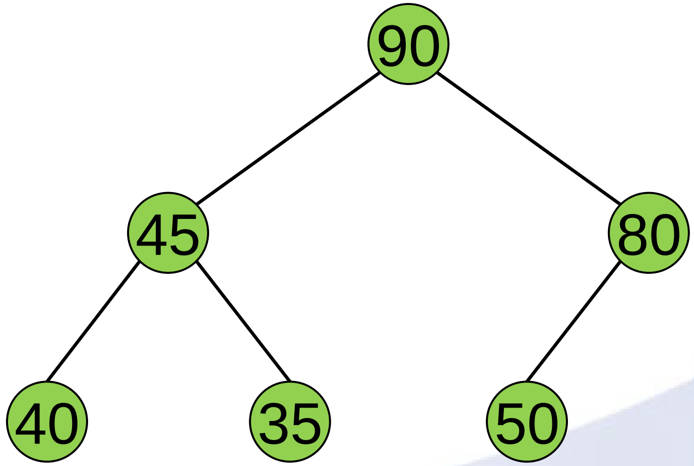


2.1 图的基本概念和性质

■ 2. 树

□ 堆：

➢ 堆是一棵完全二元树，它的每个结点的值至少和该结点的儿子们(如果存在的话)的值一样大(max-堆)(或小，min-堆)。



2.1 图的基本概念和性质

■ 2. 树

□二分检索树：

二分检索树 T 是一棵二元树，它或者为空，或者其每个结点含有一个可以比较大小的数据元素，且有：

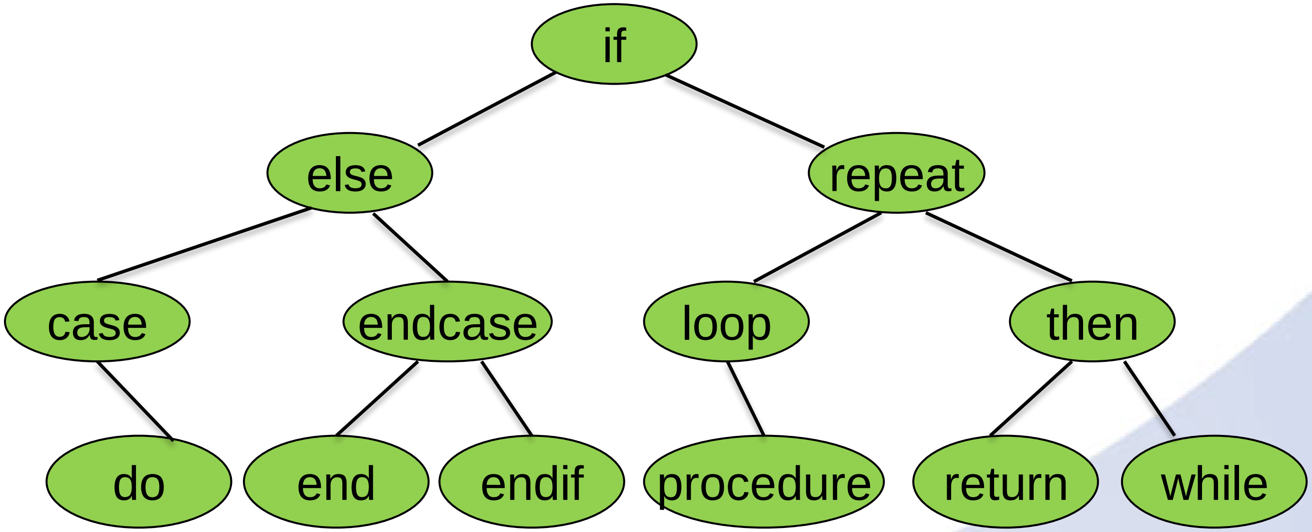
- ➢ T 的左子树的所有元素比根结点中的元素小；
- ➢ T 的右子树的所有元素比根结点中的元素大；
- ➢ T 的左子树和右子树也是二分检索树。
- ➢注：二分检索树要求树中所有结点的元素值互异

2.1 图的基本概念和性质

■ 2. 树

□二分检索树

➢按照首字母排序



2.1 图的基本概念和性质

- **3. 集合的树表示和不相交集合的合并**
  - 问题描述：
    - 给定一个全集U，该集合包含n个元素
    - 该集合包含多个不相交的子集
    - 某些应用需要实现这些不相交子集的合并、查找操作，并且这些操作最终可形成序列
    - 如何高效率实现这些操作序列?

2.1 图的基本概念和性质

- 3. 集合的树表示和不相交集合的合并
  - 集合操作举例
    - $n=10$, $U=\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$
    - $s_1=\{1, 7, 8, 9\}$; $s_2=\{2, 5, 10\}$; $s_3=\{3, 4, 6\}$
    - 合并运算：
      - ✓ $s_1 \cup s_2=\{1, 7, 8, 9, 2, 5, 10\}$
    - 查找运算：
      - ✓ 元素4包含在s_1, s_2, s_3的哪个集合中？

2.1 图的基本概念和性质

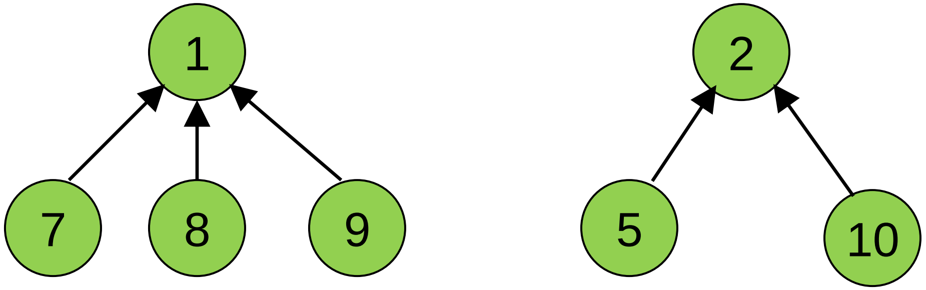
- **3. 集合的树表示和不相交集合的合并**
  - 方法一：位向量
    - s_1= {1, 0, 0, 0, 0, 0, 1, 1, 1, 0};
    - s_2= {0, 1, 0, 0, 1, 0, 0, 0, 0, 1};
    - 利用位运算可得出
      $s_1 \cup s_2$= {1, 1, 0, 0, 1, 0, 1, 1, 1, 1}
    - 缺点：
      当n很大，而每个集合的大小相对于n来说又很小时，并或者交的执行时间不是与两个集合中的元素数目成正比，而是与n成正比。

2.1 图的基本概念和性质

- 3. 集合的树表示和不相交集合的合并
  - 方法二: 集合元素表
    - $s_1=\{1, 7, 8, 9\}$;
    - $s_2=\{2, 5, 10\}$
    - 合并操作：
      无序$|s_1|*|s_2|$, 有序$|s_1|+|s_2|$,
    - 查找操作：
      与集合长度的和成正比。最坏为$|n|$

2.1 图的基本概念和性质

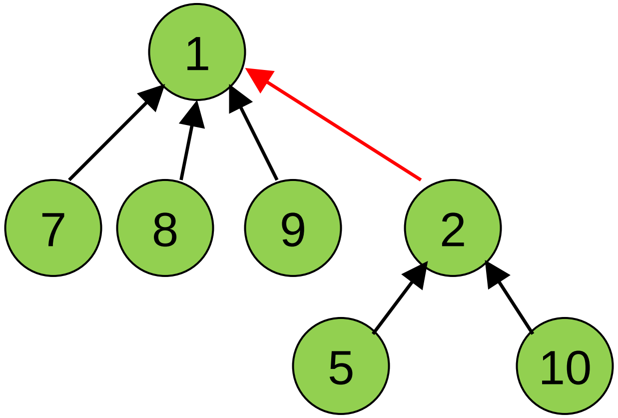
- 3. 集合的树表示和不相交集合的合并
  - 方法三——树
    - s_1和s_2可以用下面的两棵树来表示：

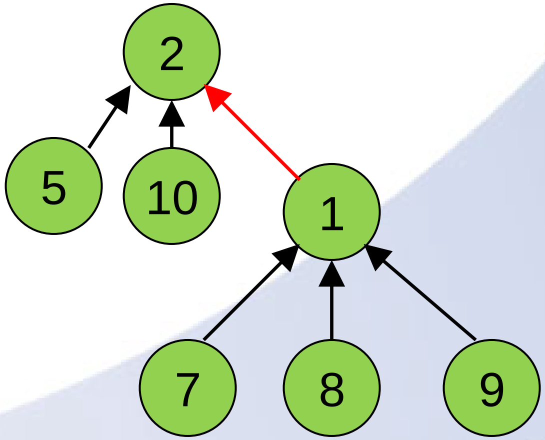


2.1 图的基本概念和性质

■ 3. 集合的树表示和不相交集合的合并

- 方法三——树
  - 求两个不相交集合的并集，一种最直接的方法就是使一棵树变成另一棵树的子树。
  - 因此，$s_1 \cup s_2$就有下面两种图的形式之一：





2.1 图的基本概念和性质

■ 3. 集合的树表示和不相交集合的合并

□数据结构

- ➢树用链接表表示，树中每个结点仅含有一个信息段，每个结点只需要设置**PARENT**信息段。
- ➢**PARENT(i)**: 存放着元素**i**在树中其父结点的指针。
- ➢根结点的**PARENT**信息段的内容为零。

(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)
0	0	…	…	2	…	1	1	1	2

2.1 图的基本概念和性质

■ 3. 集合的树表示和不相交集合的合并

□数据结构

- ➢将两个不相交的集合合并，实质上就是把一棵树的根的**PARENT**信息段置成另一棵树的根。
  - ✓合并操作U(1, 2)后：(Parent[1]=2)

(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)
2	0	…	…	2	…	1	1	1	2

```
procedure U(i, j)
  //根为i和j的两个不相交集合用它们的并来取代//
  integer i, j
  PARENT(i) ← j
end U
```

合并后树根为j

2.1 图的基本概念和性质

■ 3. 集合的树表示和不相交集合的合并

□数据结构

- 找到元素i所属集合就是找到包含元素i的集合的树根。
- U操作为常量时间，F操作则与查找元素在集合树中的层数有关。

```
procedure F(i)
  //查找包含元素i的树的根//
  integer i, j
  j ← i
  while PARENT(j)>0 do  //若此结点是根，则PARENT(j)=0 //
    j ← PARENT(j)
  repeat
  return (j)
end F
```

2.1 图的基本概念和性质

■ 3. 集合的树表示和不相交集合的合并

□U和F的性能问题——退化树

➢问题描述：有集合如下：

(1)	(2)	( )	(i)	( )	(n)
0	0	…	0	…	0

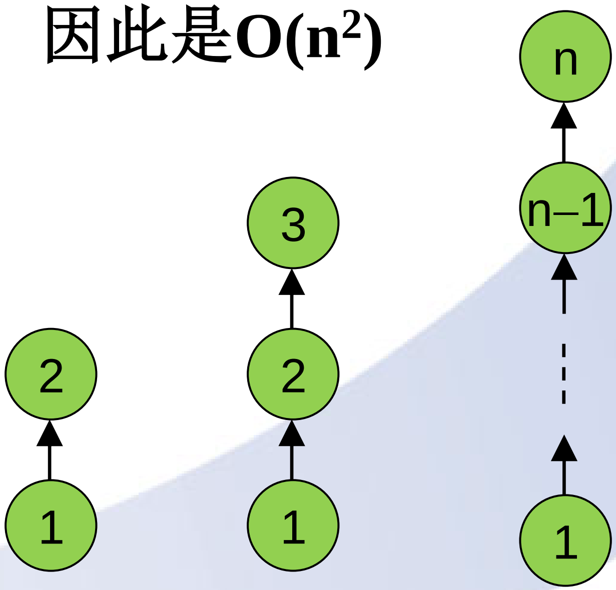
➢依次作下列操作: U(1, 2), F(1), U(2, 3), F(1), …, U(n-1, n)

➢按照算法U和F, 最终得到的树及时间耗费分析

2.1 图的基本概念和性质

■ 3. 集合的树表示和不相交集合的合并

- U和F的性能问题——退化树
  - 分析
    - ✓U: 每次都是常量时间，因此总共是O(n-1)
    - ✓F(1): 2+3+…+(n-1)，因此是$O(n^2)$
    - ✓症结？合并操作！



2.1 图的基本概念和性质

- **3. 集合的树表示和不相交集合的合并**
  - 加权规则
    - 如果树i的结点数少于树j的结点数，则使j成为i的父亲，反之，使i成为j的父亲。
    - 简化之:

      节点数少的树合并到节点数多的树中。
    - 树根i:

      PARENT(i)不再为0，而是树i中结点个数的负数形式。

2.1 图的基本概念和性质

■ 3. 集合的树表示和不相交集合的合并

□加权规则

```
Procedure  UNION(i,  j)
// PARENT(i) =-COUNT(i)//
integer i, j, x
x←PARENT(i) + PARENT(j)
if PARENT(i) > PARENT(j)
    then  PARENT(i) ←j
          PARENT(j) ←x
    else   PARENT(j) ←i
          PARENT(i)←x
endif
end UNION
```

合并后集合中元素个数

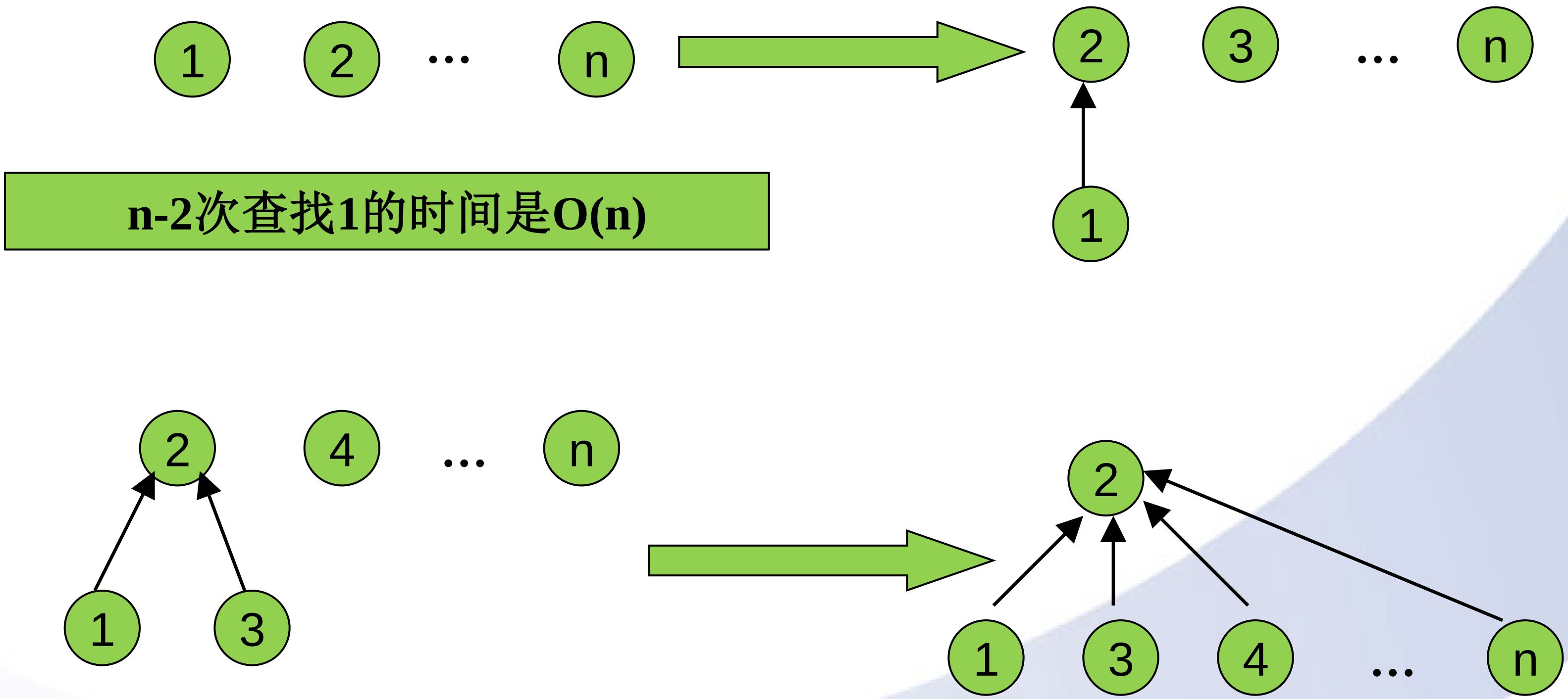
若树j的结点个数多，以j为根

若树i的结点个数多或两棵树结点个数相同，以i为根

2.1 图的基本概念和性质

■ 3. 集合的树表示和不相交集合的合并

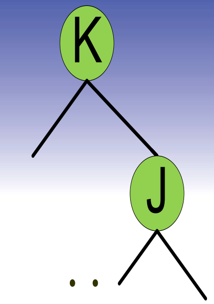
□加权规则



2.1 图的基本概念和性质

- 3. 集合的树表示和不相交集合的合并
  - 加权规则
    - 分析：
      - ✓Union(1, 2), F(1), Union(2, 3), F(1), …, Union(n-1, n)
      - ✓Union合并的开销较u要大，但仍然是常量时间
      - ✓每次查找1耗费时间为2，常量时间，则执行n-2次查找耗费时间为O(n)
      - ✓注意: 本例的查找耗时不是最坏情况
      - ✓最坏情况由引理2.3给出

2.1 图的基本概念和性质



■ 3. 集合的树表示和不相交集合的合并

引理2.3 设T是一棵由算法Union所产生的有n个节点的树。在T中没有节点的级数会大于(logn的下界+1)

证明：

- n=1，显然引理为真；
- i为T的级数，假设当$i \le n-1$时，引理为真，现证对于i=n，引理也为真；
  - ✓ 令k和j是形成树T的最后一次合并，即Union(k, j)；
  - ✓ 用count()表示数的节点数，假设count(j)=m，那么count(k)=n-m；
  - ✓ 不失一般性，可假设$1 \le m<n/2$，则有 $m \le n-m$；
  - ✓ 那么经Union合并后，j的父亲为k；则T的级数：

  1) 等于k的级数：log(n-m)的下界+1 $\le$ (logn的下界+1)

  2) 或者等于(j的级数+1)：（logm的下界+2) $\le$ (log(n/2)的下界+2) $\le$ (logn的下界+1)
- 得证

2.1 图的基本概念和性质

- 3. 集合的树表示和不相交集合的合并
  - 压缩规则
    - 引理2.3表明，对于算法Union所产生的n个结点的树，执行一次查找的最差时间至多是O(logn)
    - 如果要处理的合并和查找序列含有n次合并和m次查找，则最坏时间变成了O(mlogn)
    - 还可以继续优化！
    - 如果j是由i到它的根的路径上的一个结点，则置PARENT(j) $\leftarrow$ root(i)。
    - 更快的平均查找时间，适用于频繁查找操作。

中国科学院大学
University of Chinese Academy of Sciences

2.1 图的基本概念和性质

```
procedure FIND(i)
    integer i, j, k
    j←i
    while  PARENT(j)>0 do
        j←PARENT(j)
    repeat
    k←i
    while k≠j do
        t←PARENT(k)
        PARENT(k) ←j
        k←t
    repeat
    return(j)
end FIND
```

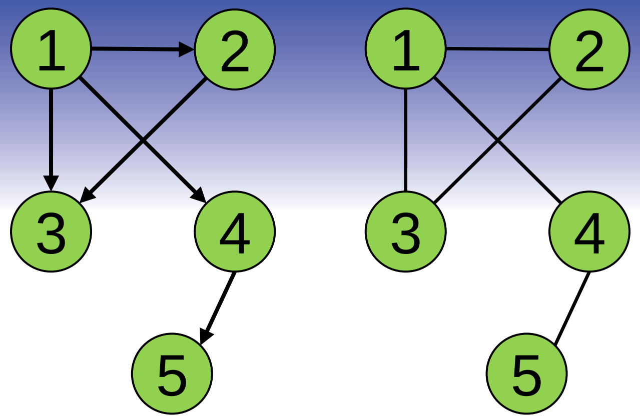
j：树根结点

找到结点i所在树的树根j

k：由i到j的路径上经过的结点

从结点i到树根j的路径上所有结点的PARENT(k)都变为j

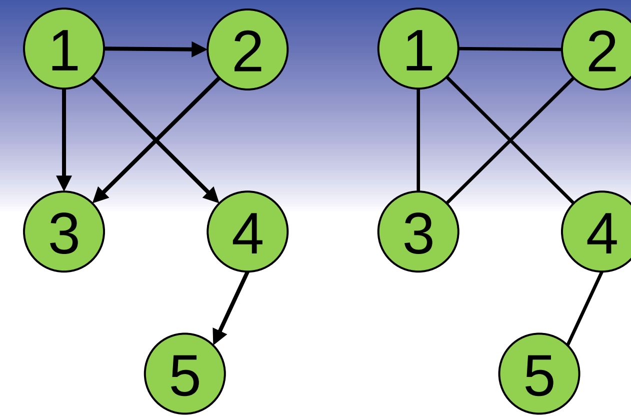
2.1 图的基本概念和性质



■ 4. 图

- **定义:**
  - 图G由称之为结点V和边E的两个集合组成，记为G=(V, E)。其中，V是一个有限非空的结点集合; E是结点对偶的集合，E的每一对偶表示G的一条边。
- **基本概念:**
  - 无向图：边的表示(i, j)
  - 有向图：边的表示<i, j>
  - 成本：带有成本的图称为网络
  - 邻接: 无向图中，如果存在边(i, j)，则称i, j相邻接

2.1 图的基本概念和性质



■ 4. 图

□基本概念：

- 结点的度(出度/入度)：邻接点的数目
- 路径：由结点v_p到v_q的一条路(path)是结点v_p， v_{i1}， v_{i2}， …， v_{im}， v_q的一个序列，它使得(v_p， v_{i1})，(v_{i1}， v_{i2})， …， (v_{im}， v_q)是E(G)的边。
- 路的长度：组成路的边数
- 简单路径：除了第一和最后一个结点可以相同以外，其它所有结点都不同。

2.1 图的基本概念和性质

■ 4. 图

□基本概念

- 环：第一个和最后一个结点相同的简单路。
- 连通图：在无向图中，如果每对结点之间都存在一条路，则称该图是连通的。
- 子图：是由G的结点集V的子集(记为V_B)和边集E中连接V_B中结点的边的子集所组成的图。
- 连通分图：一个图的最大连通子图。
- 有向图的强连通性：在有向图中，如果对于每一对结点i和j，既存在一条从i到j的路，又存在一条从j到i的路，则称该有向图是强连通的。

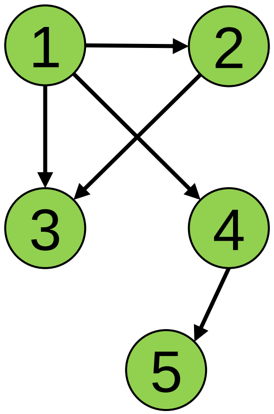
2.1 图的基本概念和性质

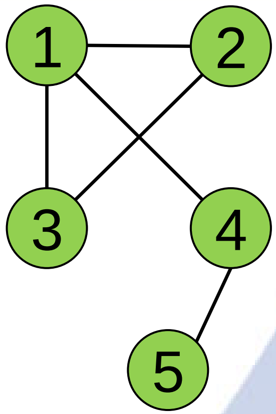
■ 4. 图

□图的表示方法

➢邻接矩阵 $A=(a_{ij})_{n\times n}$

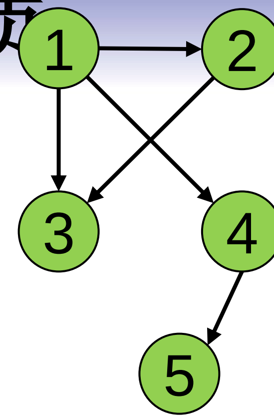
$$a_{ij}=\begin{cases}1 & \text{if } (v_i,v_j)\in G\\ 0 & \text{otherwise}\end{cases}$$

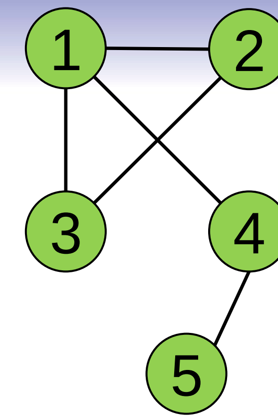




$$\begin{array}{c c} & \begin{array}{ccccc}1 & 2 & 3 & 4 & 5\end{array}\\ \begin{array}{c}(1)\\(2)\\(3)\\(4)\\(5)\end{array} & \begin{bmatrix}0&1&1&1&0\\0&0&1&0&0\\0&0&0&0&0\\0&0&0&0&1\\0&0&0&0&0\end{bmatrix}\end{array} \qquad \begin{array}{c} \begin{array}{ccccc}1 & 2 & 3 & 4 & 5\end{array}\\ \begin{bmatrix}0&1&1&1&0\\1&0&1&0&0\\1&1&0&0&0\\1&0&0&0&1\\0&0&0&1&0\end{bmatrix}\end{array}$$

2.1 图的基本概念和性质

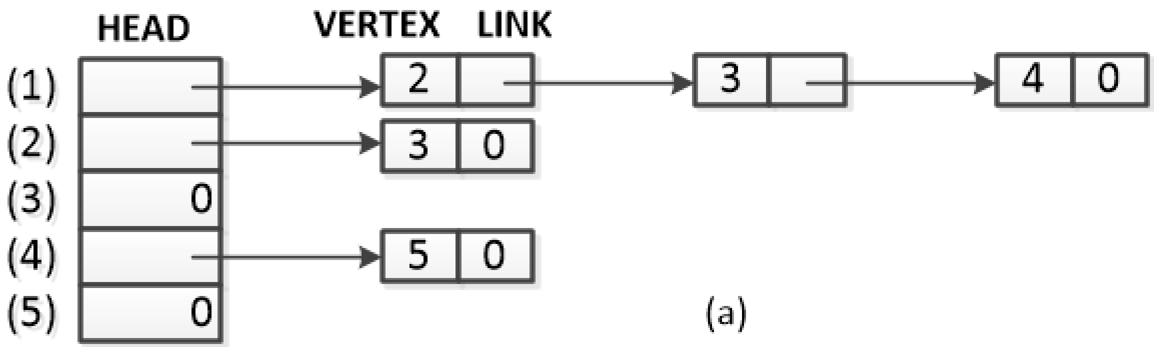




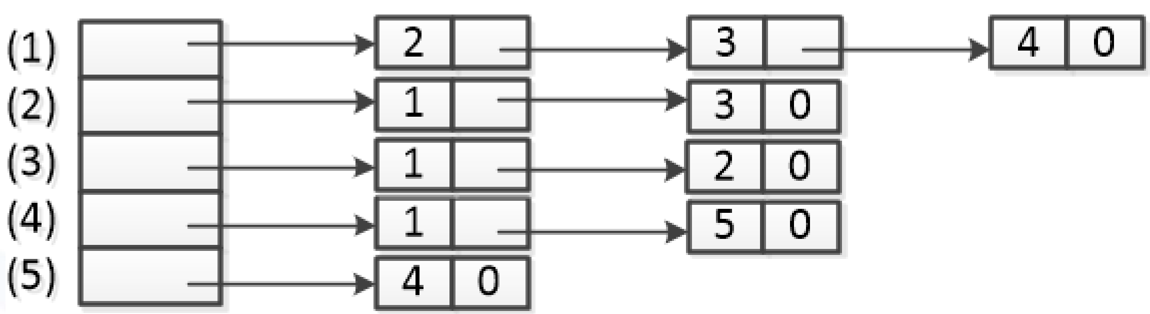
■ 4. 图

□图的表示方法

➢链接表



(a)



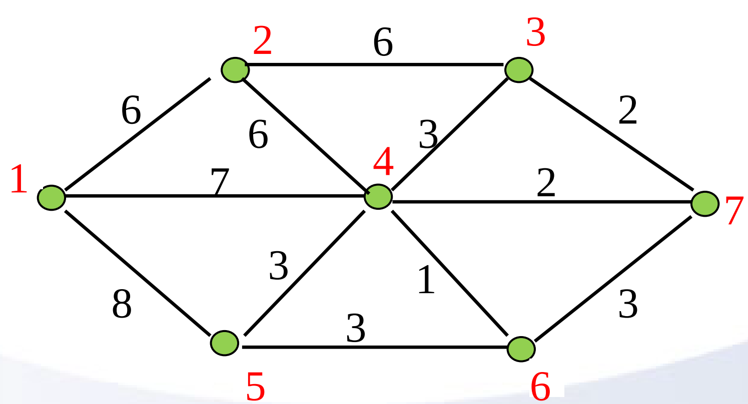
(b)

2.1 图的基本概念和性质

■ 4. 图

□赋权图

- ➢赋权图是将图的每个边都赋予一个权值，表示成本、效益值、容量、流量等。
- ➢赋权图的邻接矩阵的 (i, j) - 元素为连结顶点 **i，j**的权值。当顶点 **i，j**没有边连结时，可根据问题需要，取**0**或 ∞ 等。



一个赋权图

$$A=\begin{pmatrix} \infty & 6 & \infty & 7 & 8 & \infty & \infty \\ 6 & \infty & 6 & 6 & \infty & \infty & \infty \\ \infty & 6 & \infty & 3 & \infty & \infty & 2 \\ 7 & 6 & 3 & \infty & 3 & 1 & 2 \\ 8 & \infty & \infty & 3 & \infty & 3 & \infty \\ \infty & \infty & \infty & 1 & 3 & \infty & 3 \\ \infty & \infty & 2 & 2 & \infty & 3 & \infty \end{pmatrix}$$

End