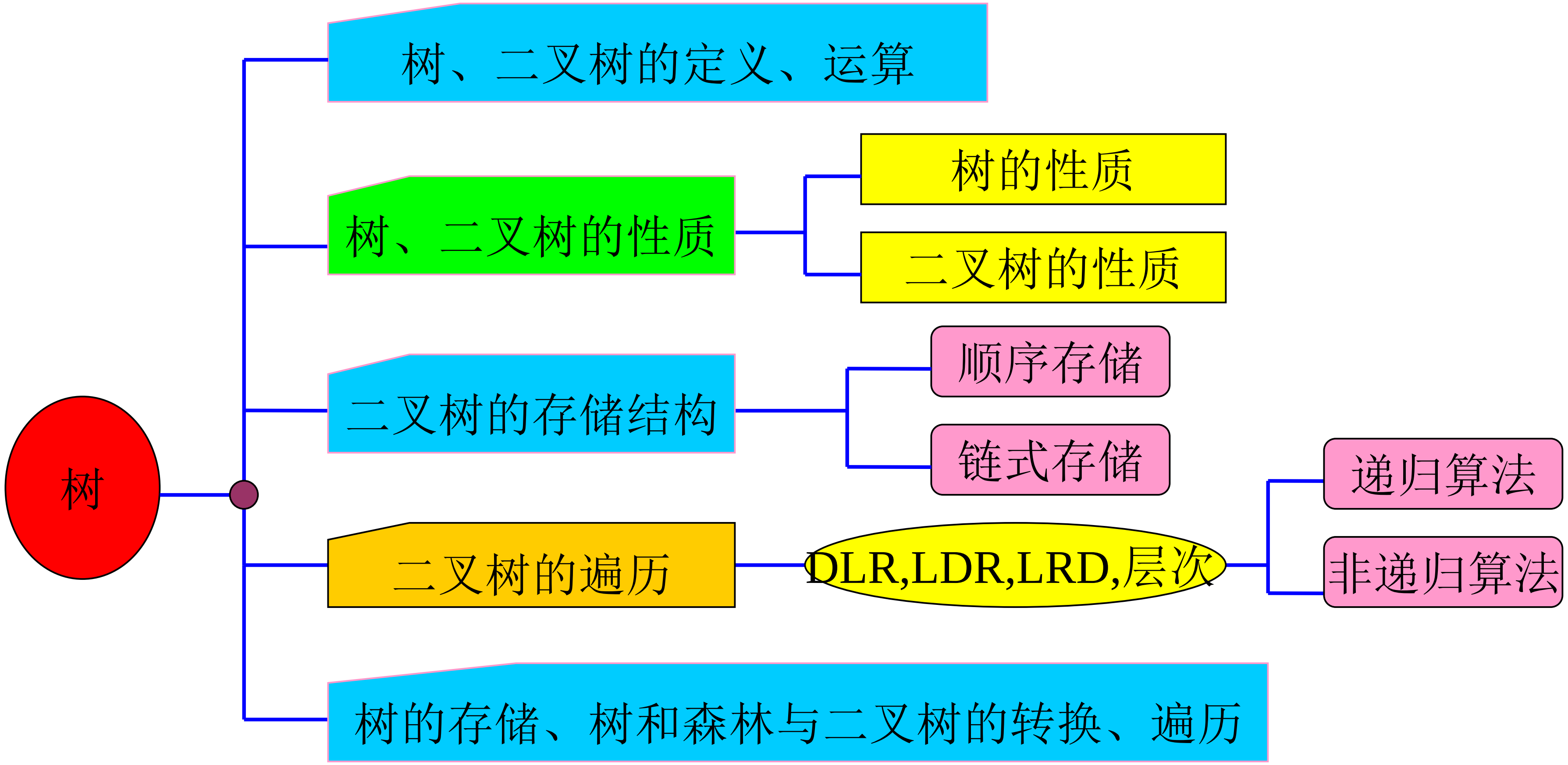
4

树与二叉树

知识点


树
树、二叉树的定义、运算
树、二叉树的性质
树的性质
二叉树的性质
二叉树的存储结构
顺序存储
链式存储
二叉树的遍历
DLR,LDR,LRD,层次
递归算法
非递归算法
树的存储、树和森林与二叉树的转换、遍历

树

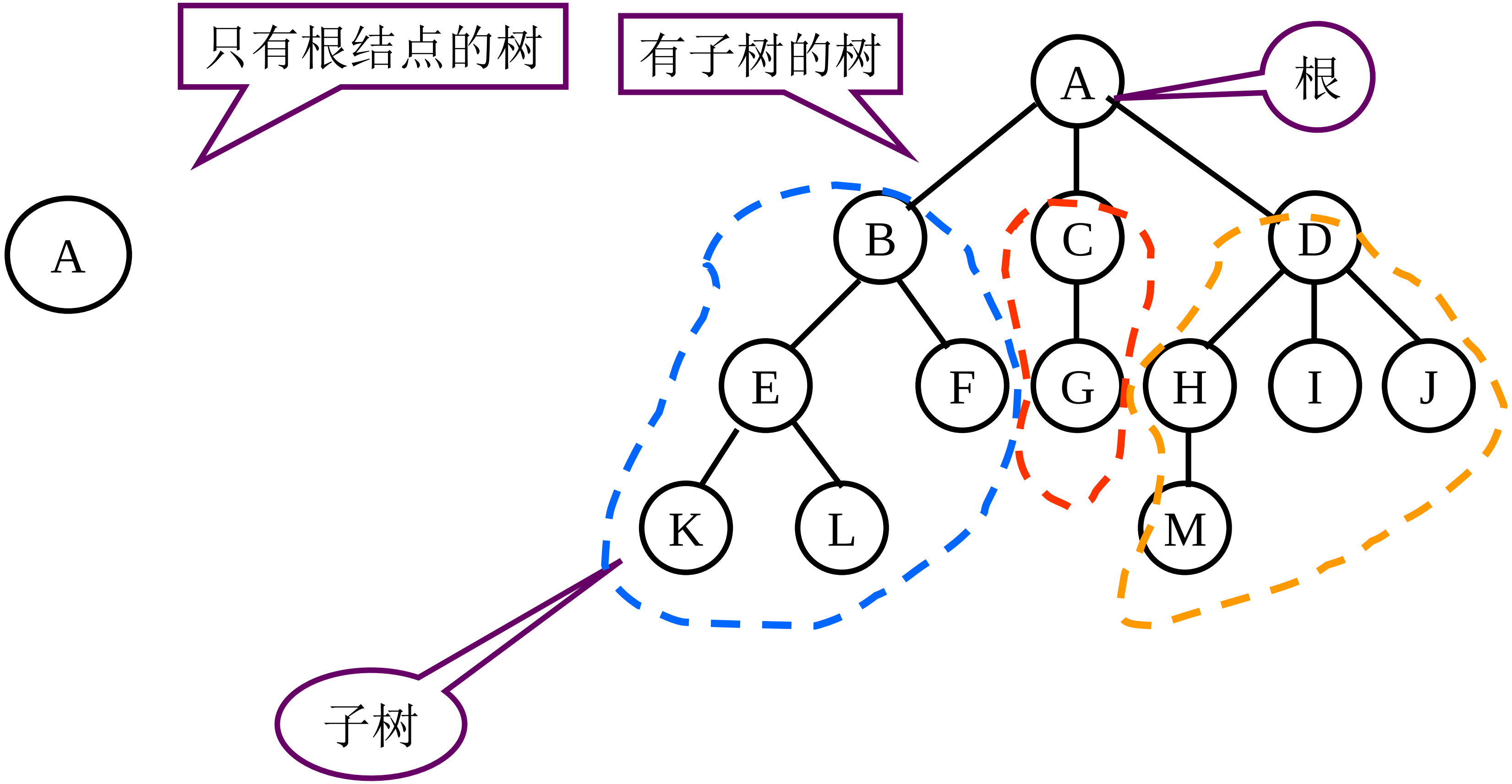
树的概念

树（Tree）是n（n≥0）个节点的有限集合T，它满足两个条件 ：

- 有且仅有一个特定的称为根（Root）的节点；
- 其余的节点可以分为m（m≥0）个互不相交的有限集合T1、T2、……、Tm，其中每一个集合又是一棵树，并称为其根的子树（Subtree）。

表示方法 ：**树形表示法、目录表示法**。

树



树

基本概念：

一个节点的子树的个数称为该节点的**度数**，一棵树的度数是指该树中节点的最大度数。

度数为零的节点称为**树叶**或**终端节点**，度数不为零的节点称为分支节点，除根节点外的分支节点称为内部节点。

一个节点的子树之根节点称为该节点的**子节点**，该节点称为它们的**父节点**，同一节点的各个子节点之间称为兄弟节点。一棵树的根节点没有父节点，叶节点没有子节点。

树

一个节点系列k_1, k_2，……, k_i, k_{i+1}，……, k_j, 并满足k_i是k_{i+1}的父节点，就称为一条从k_1到k_j的**路径**，路径的长度为j-1, 即路径中的**边数**。路径中前面的节点是后面节点的祖先，后面节点是前面节点的子孙。

节点的层数等于父节点的层数加一，根节点的层数定义为一。树中节点层数的最大值称为该树的**高度**或**深度**。

若树中每个节点的各个子树的排列为从左到右，不能交换，即兄弟之间是有序的，则该树称为**有序树**。一般的树是有序树。

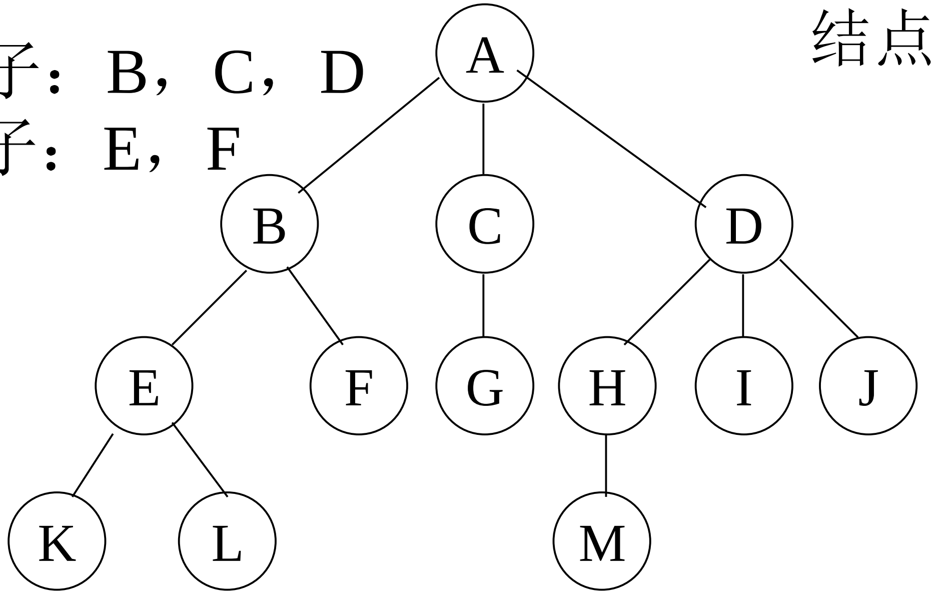
m（m≥0）棵互不相交的树的集合称为**森林**。树去掉根节点就成为森林，森林加上一个新的根节点就成为树。

树

结点A的度：3
结点B的度：2
结点M的度：0

结点A的孩子：B，C，D
结点B的孩子：E，F

树的度：3



结点I的双亲：D
结点L的双亲：E

结点B，C，D为兄弟
结点K，L为兄弟

树的深度：4

结点A的层次：1
结点M的层次：4

结点F，G为堂兄弟
结点A是结点F，G的祖先

树

树的逻辑结构 ：树中任何节点都可以有零个或多个直接后继节点（子节点），但至多只有一个直接前趋节点（父节点），根节点没有前趋节点，叶节点没有后继节点。

4.2

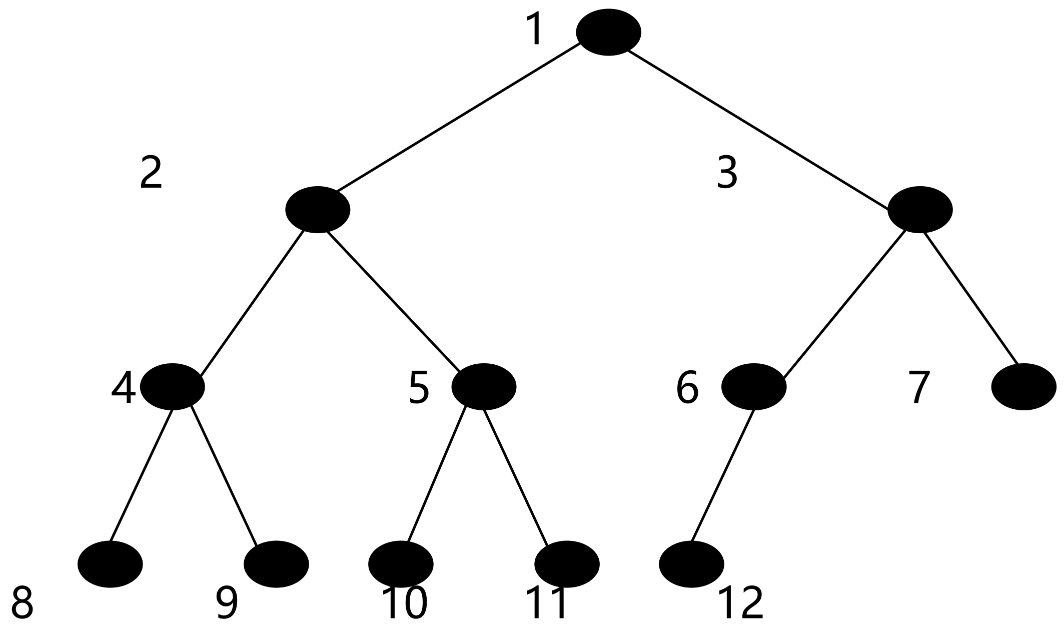
二叉树

二叉树

二叉树

二叉树的定义 ： **二叉树（Binary Tree）**是n（n≥0）个节点的有限集合，它或者是空集（n=0），或者是由一个根节点以及两棵互不相交的、分别称为左子树和右子树的二叉树组成。二叉树与普通有序树不同，二叉树严格区分左孩子和右孩子，即使只有一个子节点也要区分左右。

二叉树



二叉树

二叉树的性质：

二叉树第i（i≥1）层上的节点最多为2^{i-1}个。

深度为k（k≥1）的二叉树最多有2^k-1个节点。

在任意一棵二叉树中，树叶的数目比度数为2的节点的数目多一。

总节点数为各类节点之和：$n = n_0 + n_1 + n_2$

总节点数为所有子节点数加一：$n = n_1 + 2*n_2 + 1$

故得：$n_0 = n_2 + 1$ ；

满二叉树 ：深度为k（k≥1）时有2^k-1个节点的二叉树。

完全二叉树 ：只有最下面两层有度数小于2的节点，且最下面一层的叶节点集中在最左边的若干位置上。

具有n个节点的完全二叉树的深度为

（log2n）＋1或『log2(n+1)。

二叉树

二叉树的存储：

顺序存储结构：完全二叉树节点的编号方法是从上到下，从左到右，根节点为1号节点。设完全二叉树的节点数为n，某节点编号为i

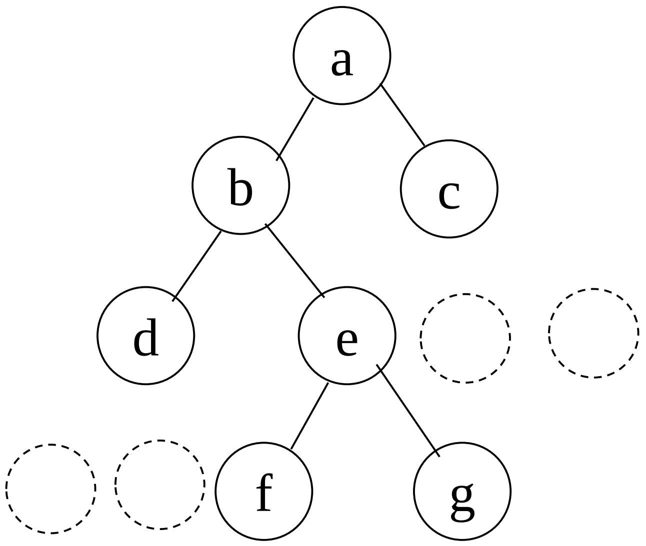
- 当i＞1（不是根节点）时，有父节点，其编号为i/2;
- 当2*i≤n时，有左孩子，其编号为2*i ,否则没有左孩子，本身是叶节点;
- 当2*i＋1≤n时，有右孩子，其编号为2*i+1 ,否则没有右孩子;
- 当i为奇数且不为1时，有左兄弟，其编号为i-1,否则没有左兄弟;
- 当i为偶数且小于n时，有右兄弟，其编号为i＋1,否则没有右兄弟;

二叉树

有n个节点的完全二叉树可以用有n+1个元素的数组进行顺序存储，节点号和数组下标一一对应，下标为零的元素不用。

利用以上特性，可以从下标获得节点的逻辑关系。不完全二叉树通过添加虚节点构成完全二叉树，然后用数组存储，这要浪费一些存储空间。

二叉树



a	b	c	d	e	0	0	0	0	f	g

浪费空间

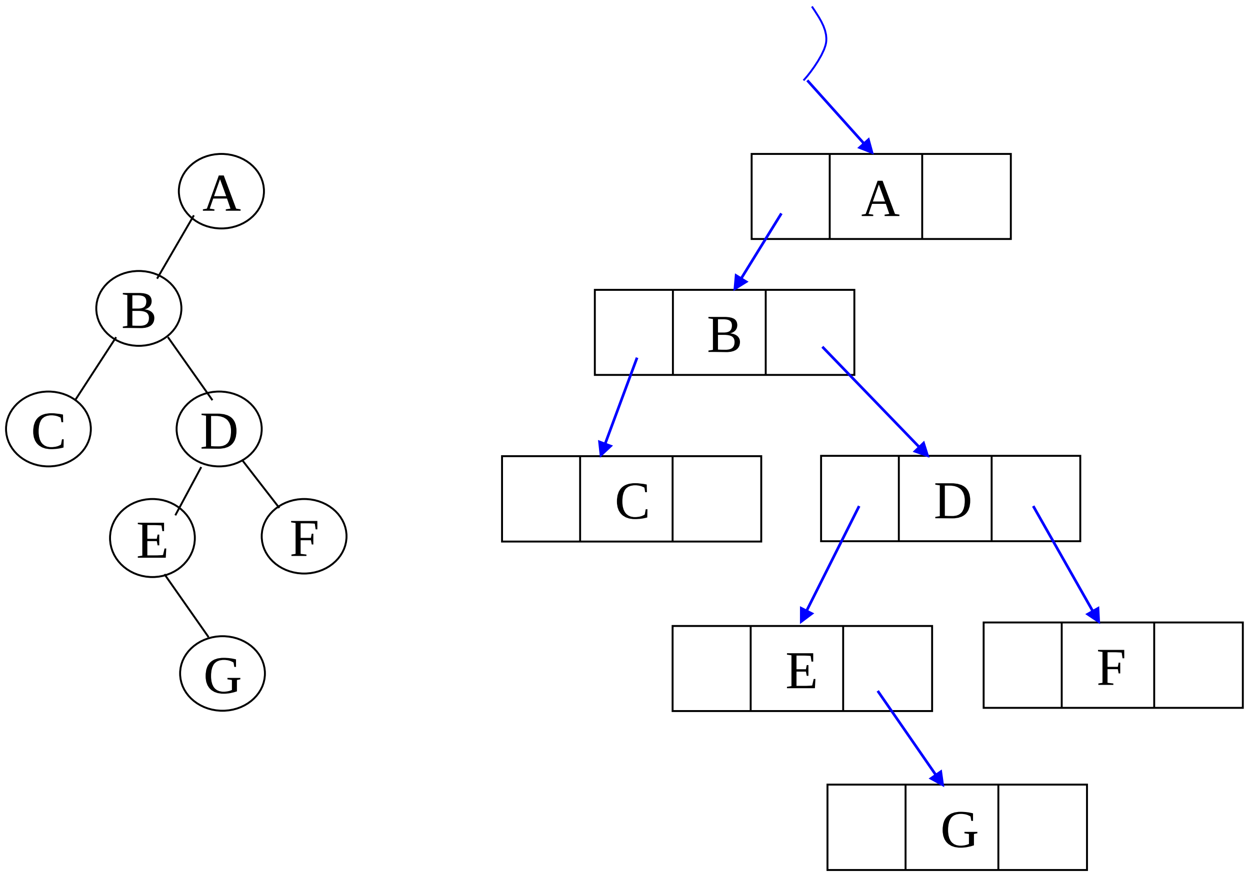
二叉树

链式存储结构 ：

```
typedef  int  data_t ;             /*定义数据类型*/
typedef  struct  node_t;           /*定义二叉树节点的内部结构*/
{
   data_t data ;           /*数据域*/
   struct node_t *lchild ,*rchild ;       /*指向左孩子和右孩子的指针*/
} bitree_t ;                       /*二叉树节点类型*/
bitree_t *root ;           /*定义指向二叉树的指针*/
```

二叉树由根节点指针决定。

二叉树



4.3

二叉树的遍历

二叉树的遍历

二叉树的遍历

遍历 ：沿某条搜索路径周游二叉树，对树中的每一个节点访问一次且仅访问一次。

“遍历”是任何类型均有的操作，对**线性结构**而言，只有一条搜索路径(因为每个结点均只有一个后继)，故不需要另加讨论。而二叉树是**非线性结构**，每个结点有两个后继，则存在如何遍历即按什么样的搜索路径进行遍历的问题。

二叉树的遍历

由于二叉树的递归性质，遍历算法也是递归的。三种基本的遍历算法如下：

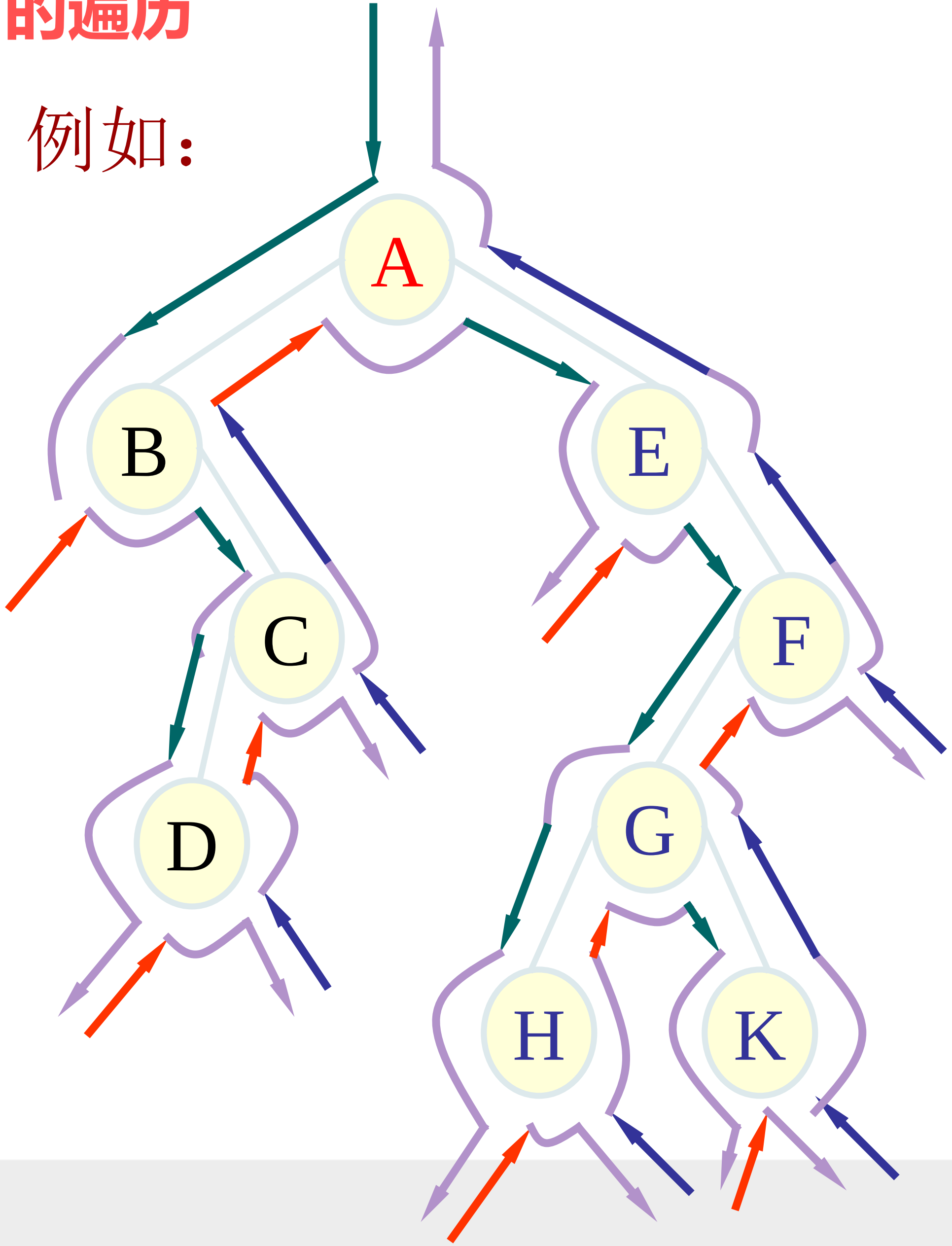
先访问树根，再访问左子树，最后访问右子树；

先访问左子树，再访问树根，最后访问右子树；

先访问左子树，再访问右子树，最后访问树根；

二叉树的遍历

例如：



先序序列：

A B C D E F G H K

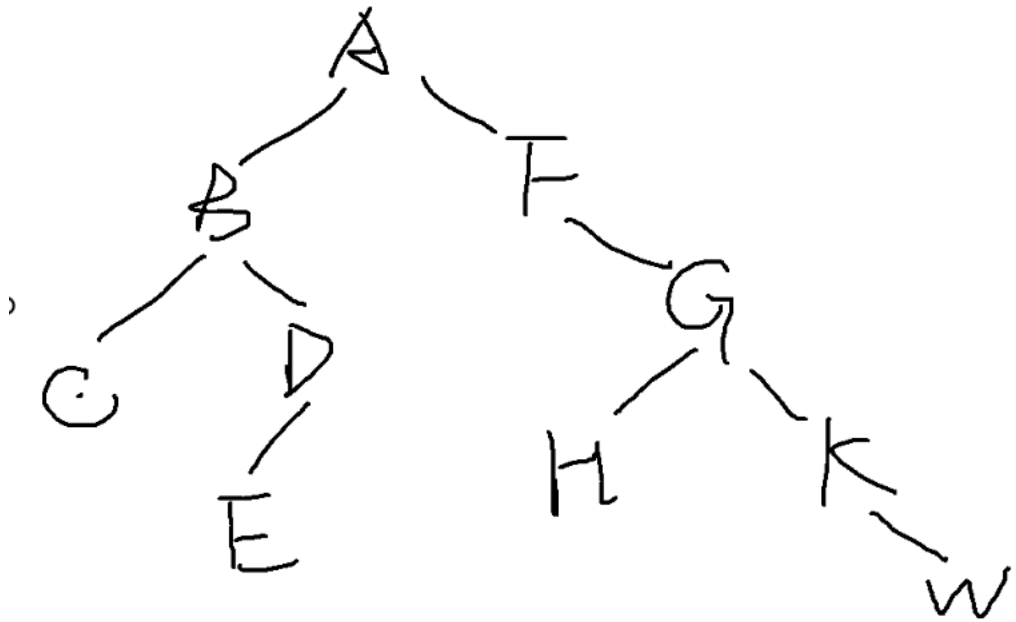
中序序列：

B D C A E H G K F

后序序列：

D C B H K G F E A

二叉树的遍历

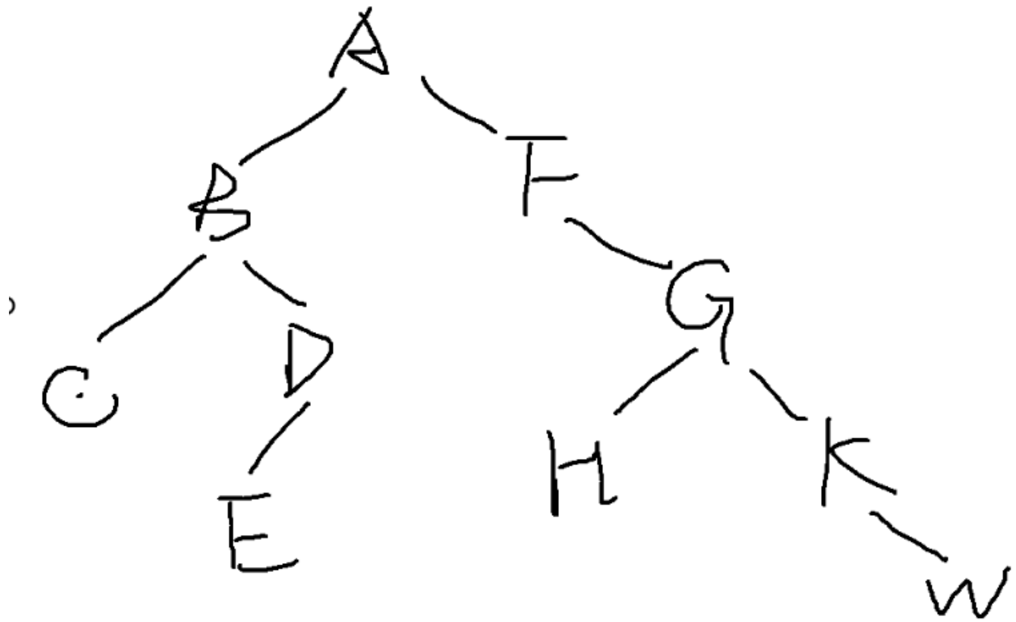


先序序列：

中序序列：

后序序列：

二叉树的遍历



先序序列：

A B C D E F G H K W

中序序列：

C B E D A F H G K W

后序序列：

C E D B H W K G F A

二叉树的遍历

先序序列：

A B D E F G C H I J

中序序列：

D B F E G A H C I J

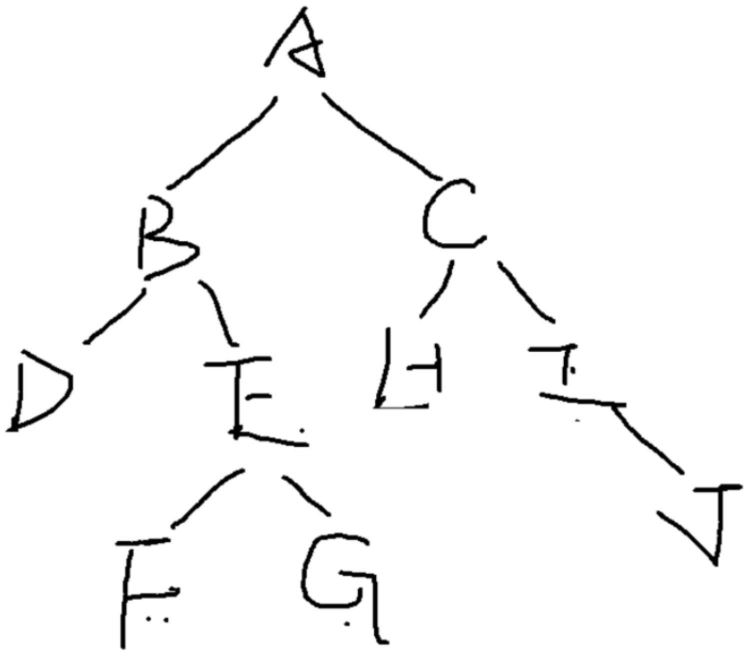
二叉树的遍历

先序序列：

A B D E F G C H I J

中序序列：

D B F E G A H C I J



二叉树的遍历

先序遍历算法

若二叉树为空树，则空操作；否则

访问根结点

先序遍历左子树

先序遍历右子树

二叉树的遍历

中序遍历算法

若二叉树为空树，则空操作；否则

中序遍历左子树

访问根结点

中序遍历右子树

二叉树的遍历

后序遍历算法

若二叉树为空树，则空操作；否则

后序遍历左子树

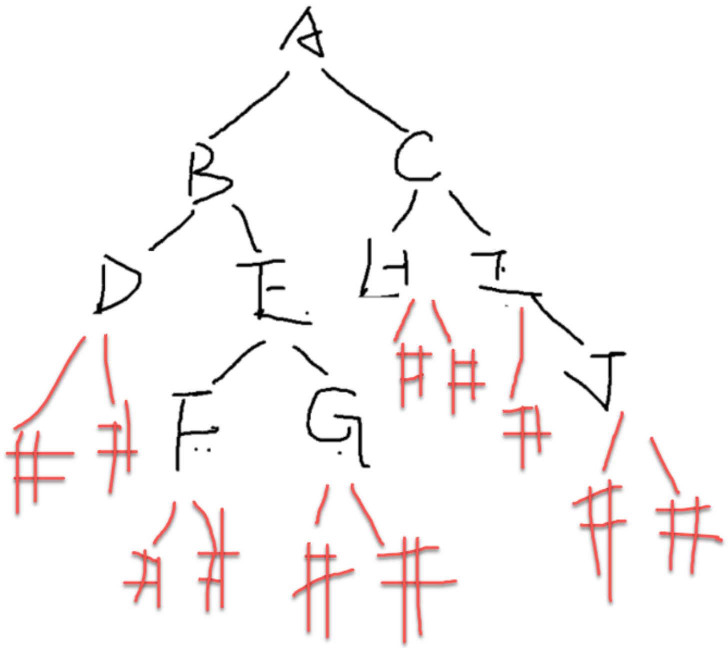
后序遍历右子树

访问根结点

二叉树的遍历

遍历的路径相同，均为从根节点出发，逆时针沿二叉树的外缘移动，每个节点均经过三次。按不同的次序访问可得不同的访问系列，每个节点有它的逻辑前趋（父节点）和逻辑后继（子节点），也有它的遍历前趋和遍历后继（要指明遍历方式）。

普通二叉树的输入



输入顺序

ABD##EF##G##CH##I#J##

先序序列：

A B D E F G C H I J

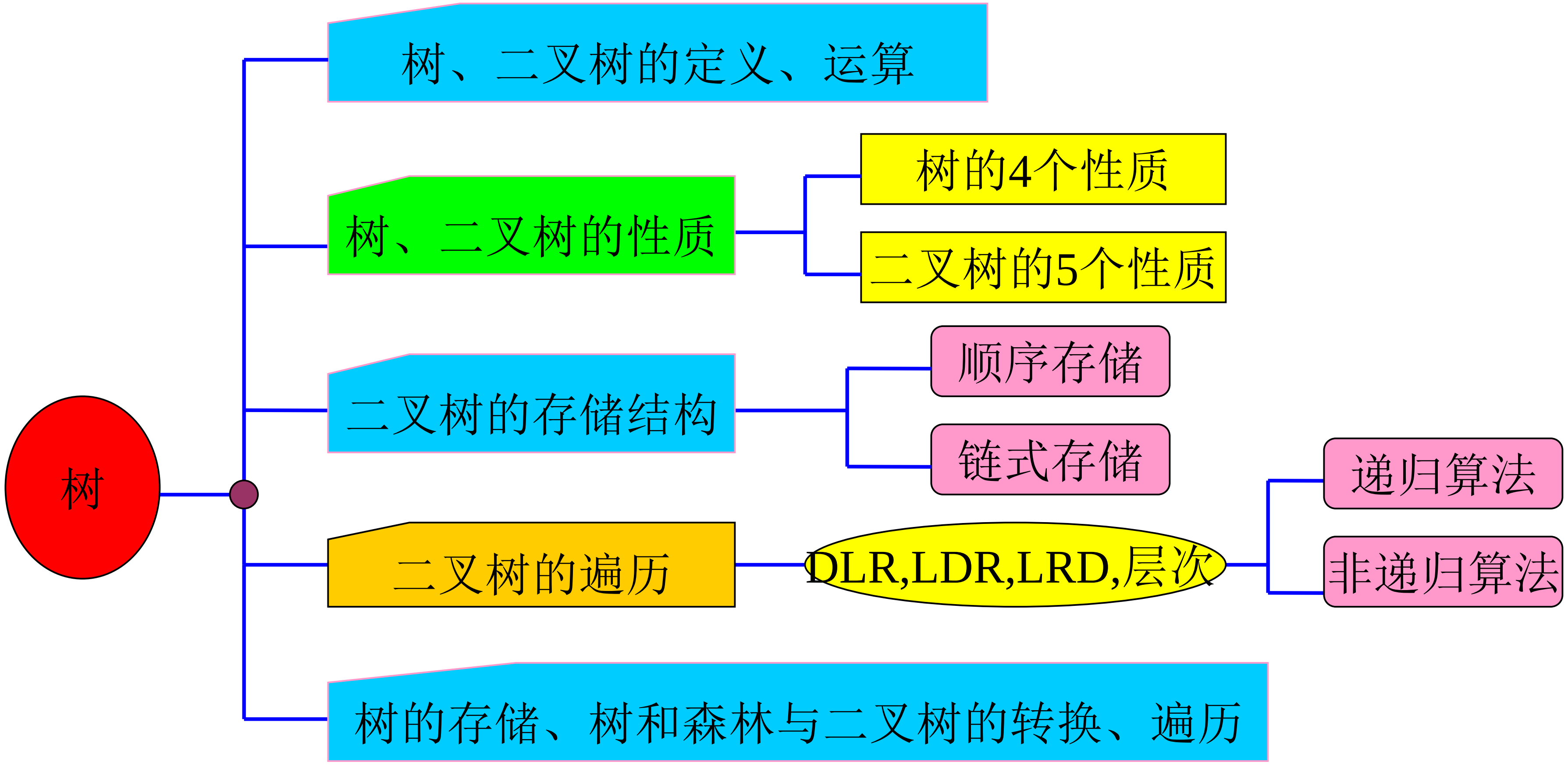
中序序列：

D B F E G A H C I J

后序序列：

D F G E B H J I C A

小结


树
树、二叉树的定义、运算
树、二叉树的性质
树的4个性质
二叉树的5个性质
二叉树的存储结构
顺序存储
链式存储
二叉树的遍历
DLR,LDR,LRD,层次
递归算法
非递归算法
树的存储、树和森林与二叉树的转换、遍历

海量视频 贴身学习

超多干货 实时更新

THANKS

— 谢谢 —