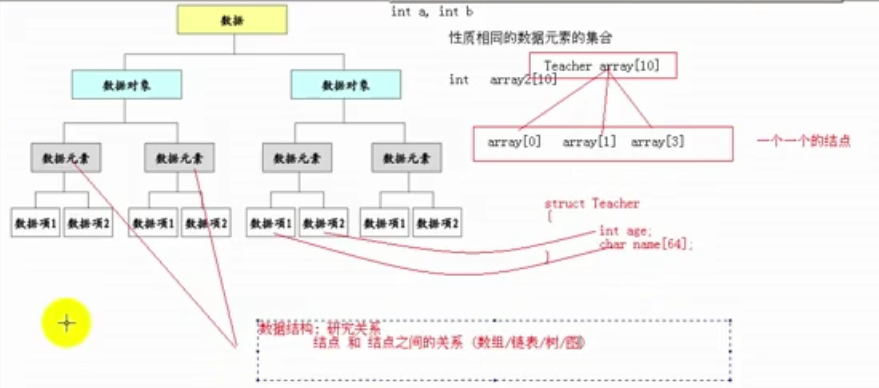
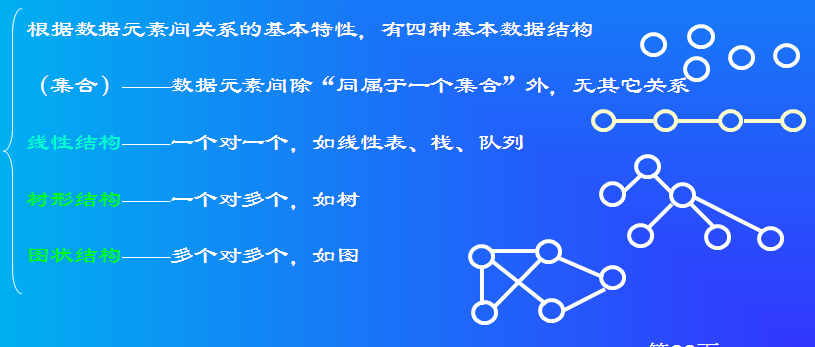
# 【基础知识】

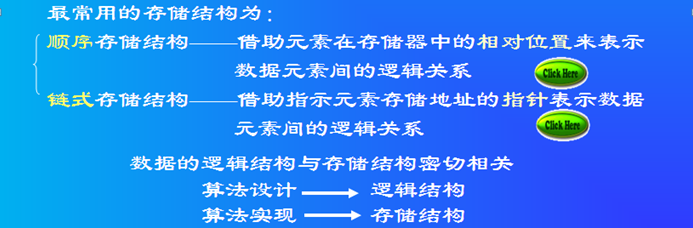


数据的逻辑结构：



数据的物理结构：（存储结构）

顺序、链式、索引、散列



常用数据的运算：



算法：

可以没有输入，但一定有输出，有限步自动终止，采用大O计法

1-时间复杂度：越往下复杂度月高

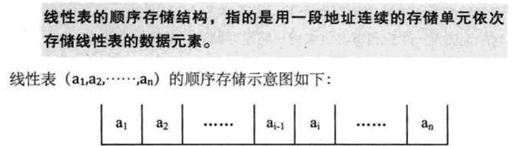


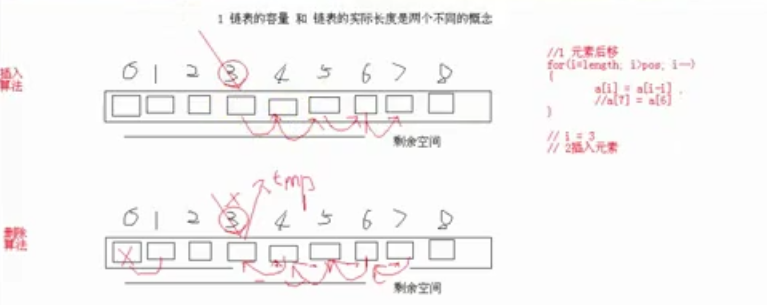
2-空间复杂度：计算算法的存储空间实现（变量定义占的空间）

# 【线性结构】

元素、前驱、后继，线性表能够逐项访问和顺序存取

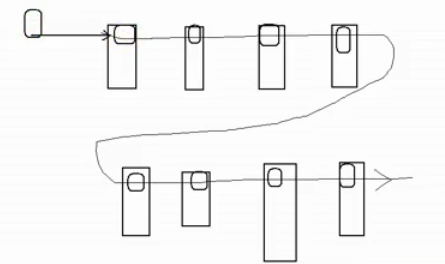
1-顺序存储：（顺序表）

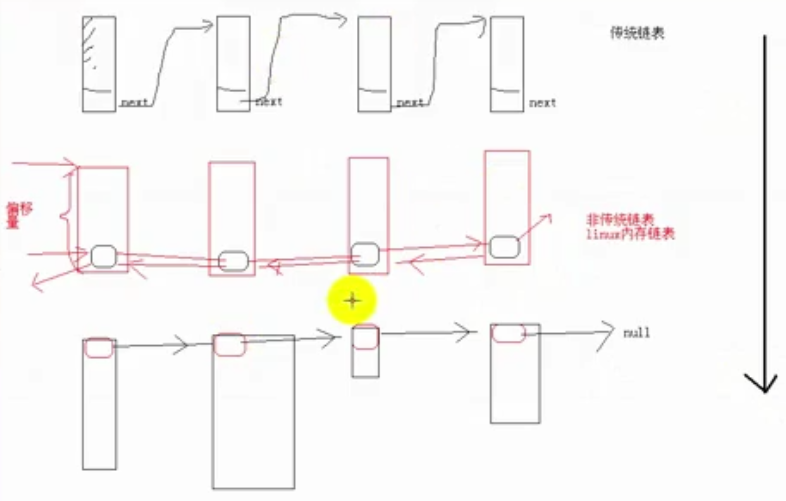


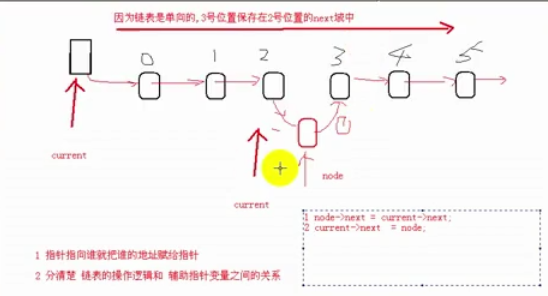


2-链式存储：（链表）

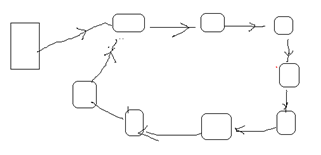
单链表的演变：



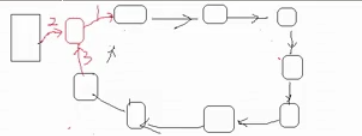




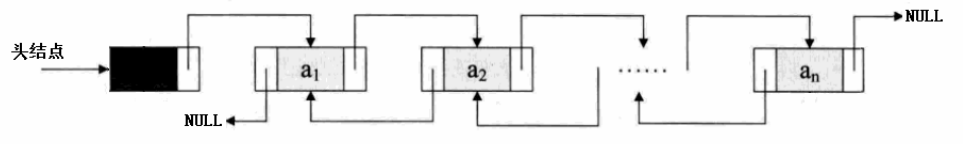
循环链表：

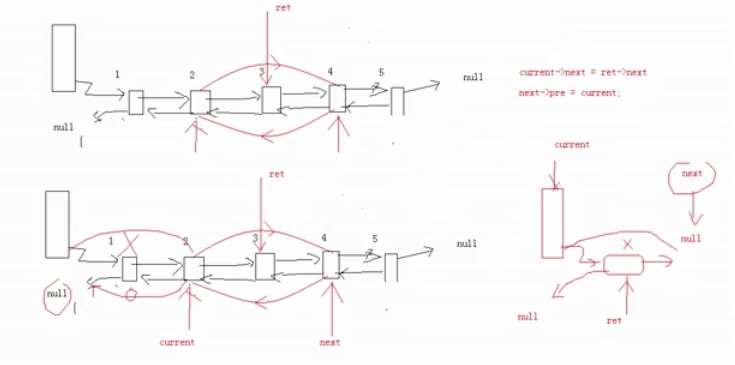
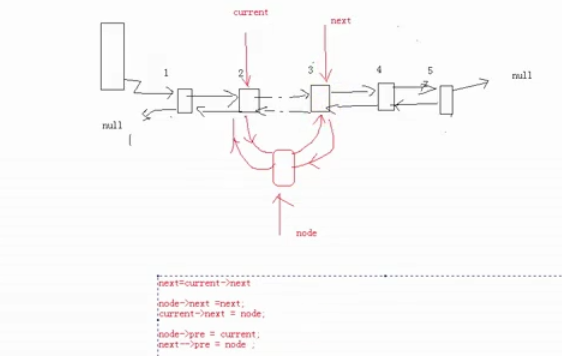


循环链表插入元素的头插法和尾插法：尾插法无特别，头插法需注意要改变三个指针的指向



双向链表：



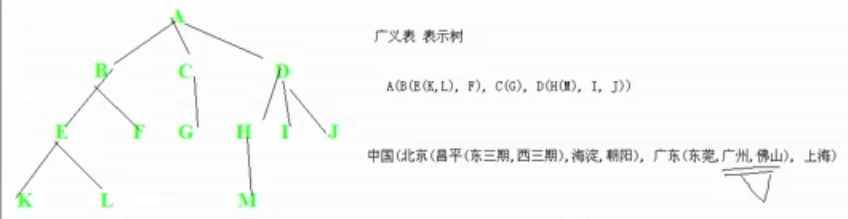


栈：先进后出，单边操作

队列：先进先出，两头操作

# 【非线性结构】

**非线性结构**：一个直接前驱，但可能有多个直接后继

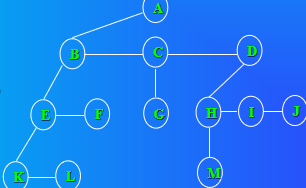
**概念**：根结点（root，无前驱）；叶子（无后继，终端结点）；子树（根结点去除后的每个集合）；森林（不相交的树的集合，如上图去除A根的子树）；双亲（parent，直接前驱结点）；孩子（child，直接后继结点）；兄弟（sibling，同双亲的同层结点）；堂兄弟（双亲位于同一层的结点的子结点）；祖先（从根到该结点所经分支的所有结点）；子孙（该结点下层子树中的结点）；分支结点（树根外的结点）；结点的度（几个直接后继）；树的度（Max{结点的度}）；树的深度（层）

**存储方式：**

1-顺序：按二叉树的结点“自上而下、从左至右”编号，用一组连续的存储单元存储。若是完全/满二叉树则可以做到唯一复原，所以非完全/满二叉树，需补缺。

2-链式：

**表示法**：图形表示（如上图左）、广义表示（如上图右）、左孩子－右兄弟表示法（左为长，如下）



**二叉树结构**：



**性质**：树是递归定义的，树是n个互不相同的集合；

性质1: 在二叉树的第k层上至多有2k-1个结点

性质2: 深度为k的二叉树至多有2k-1个结点

性质3: 对于任何一棵二叉树，若2度的结点数: K个，则叶子数: K＋1个

性质4：一棵深度为k 的二叉树，且有2k -1个结点的二叉树（满二叉树）

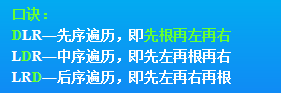


完全二叉树：只有最后一层叶子不满，且全部集中在左边。满二叉树是完全二叉树的一个特例

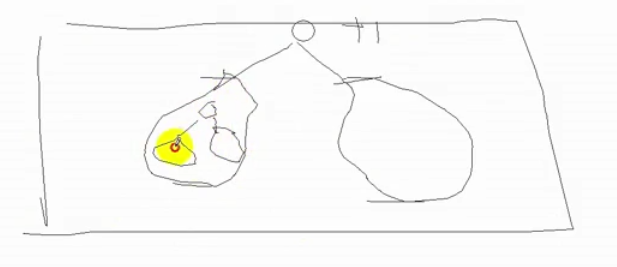
性质5：有n个结点的完全二叉树的深度必为log2n向下取整，再＋1

性质6: 对完全二叉树，若从上至下、从左至右编号，则编号为i 的结点，其左孩子编号必为2i，其右孩子编号必为2i＋1；其双亲的编号必为i/2（i＝1 时为根,除外）

**遍历**：递归



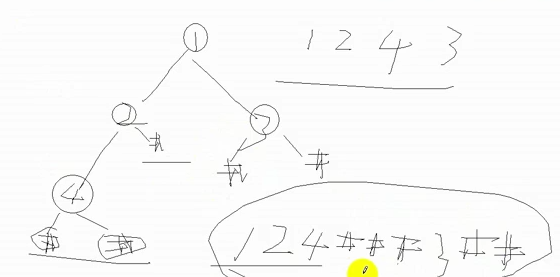
遍历方法——牢记一种约定，对每个结点的查看都是“先左后右” 。递归求树高：



**二叉树创建：**

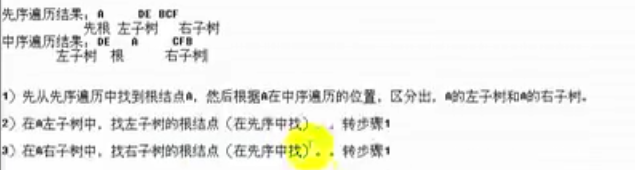
1-#法：需要插入#来辅助确定唯一的二叉树。（单独的中序、先序、后序遍历均不能确定一棵二叉树）

若采用先序结果如下：1 2 4## # 3##

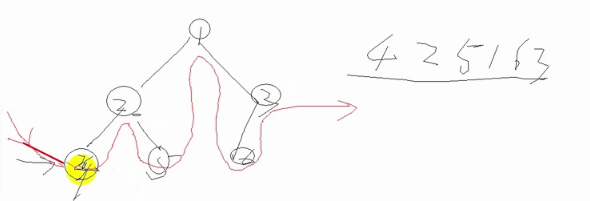
****

两个##前一个树必是某个根节点

2-先序+中序遍历法：可以唯一确定一棵二叉树。（中序+其他任意一种遍历可完成唯一确定）



3-线索二叉树



链表串起来。

