1.时间复杂性

2.线性表： 公式化（静态分配）/链表（动态分配）

数组/矩阵

堆栈

队列

散列

3.二叉树

优先队列

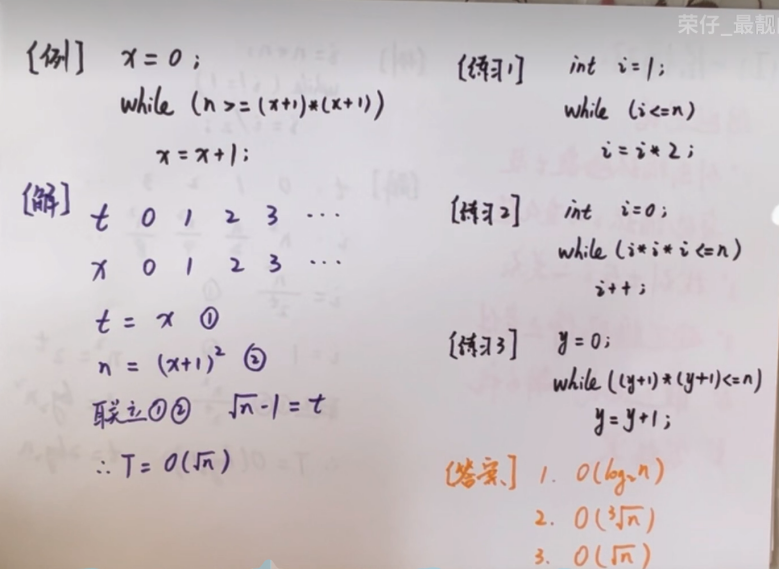
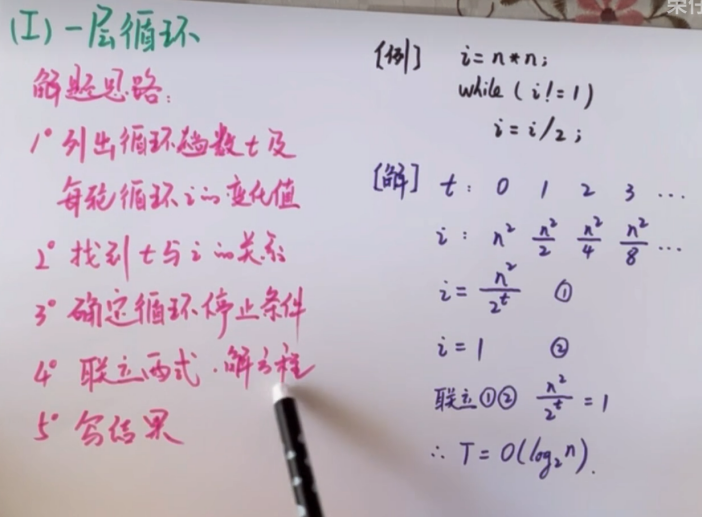
搜索树 BST AVL

4.图

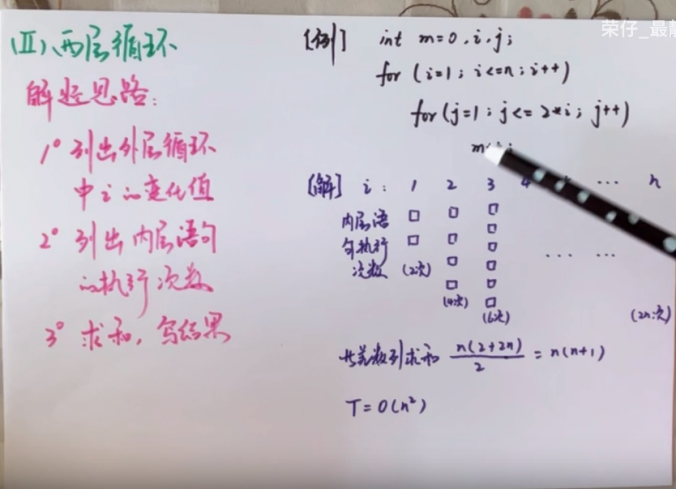
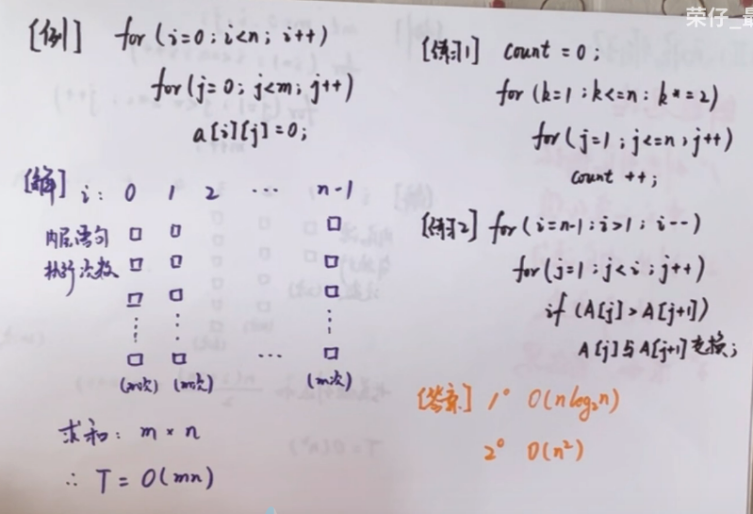
5.排序/查找

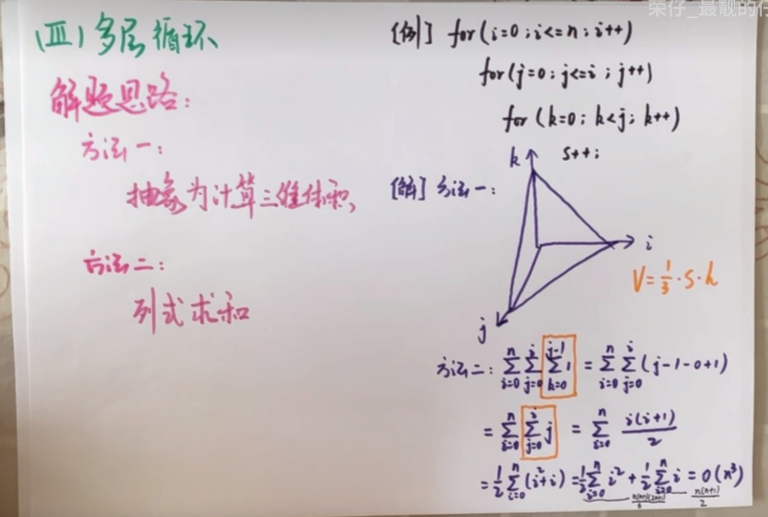
时间复杂度：

单层循环：（找t和n之间关系）



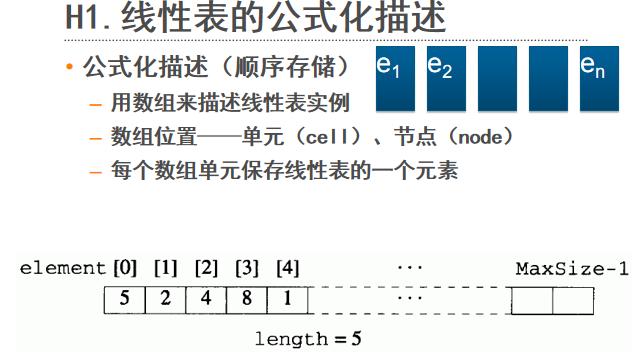
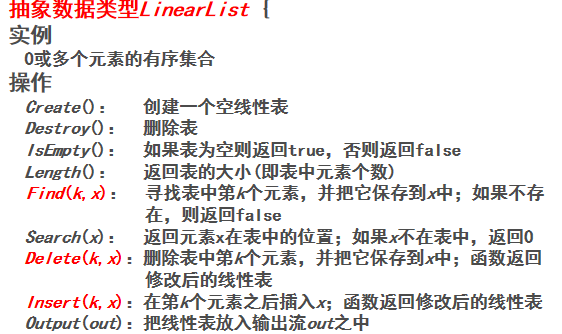
双层循环：（计算内部循环总步数和）



多层循环：（列式逐层计算）

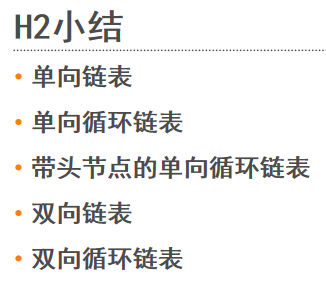
线性表：

公式化：



链表：

删除/插入操作：指针变化（前驱指向后继）



数组和矩阵：

多维数组存储： int store [u1][u2]

行主映射：优先以行存储

map(i,j)=u2\*i+j （获取[i][j]位置的数在实际存储行主次序的序号（从0开始））

map(i,j,k)=u3\*u2\*i+u3\*j+k （三维数组）

列主映射：优先以列存储

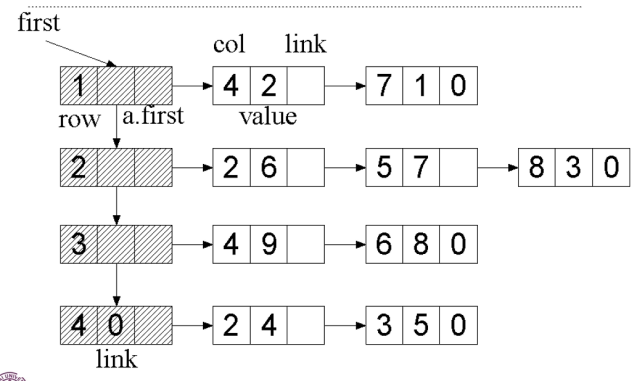
矩阵：

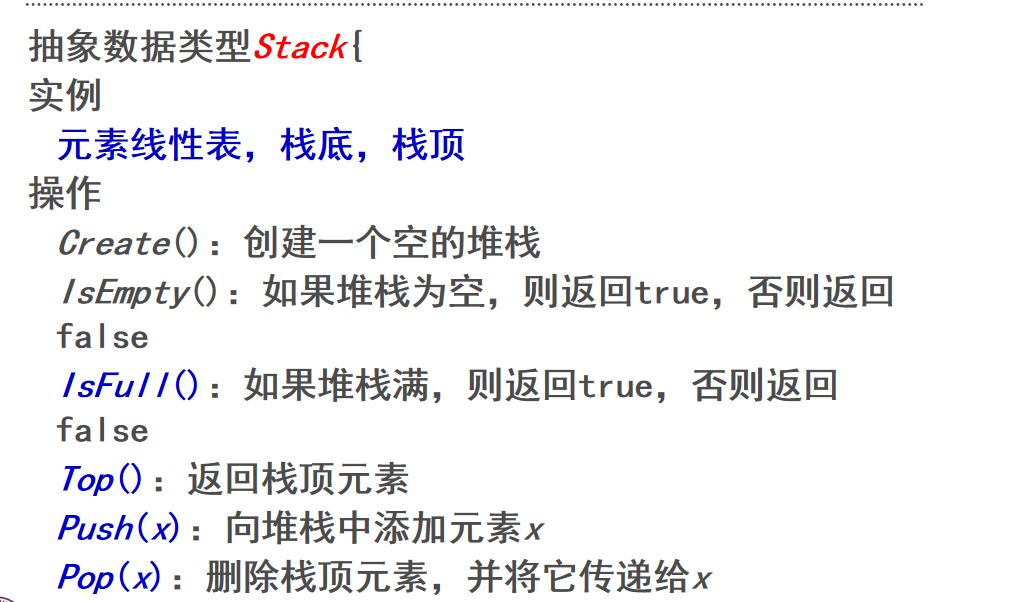
方阵

对角矩阵

上/下三角矩阵

稀疏矩阵：数组：一维数组存储非零元

 链表：

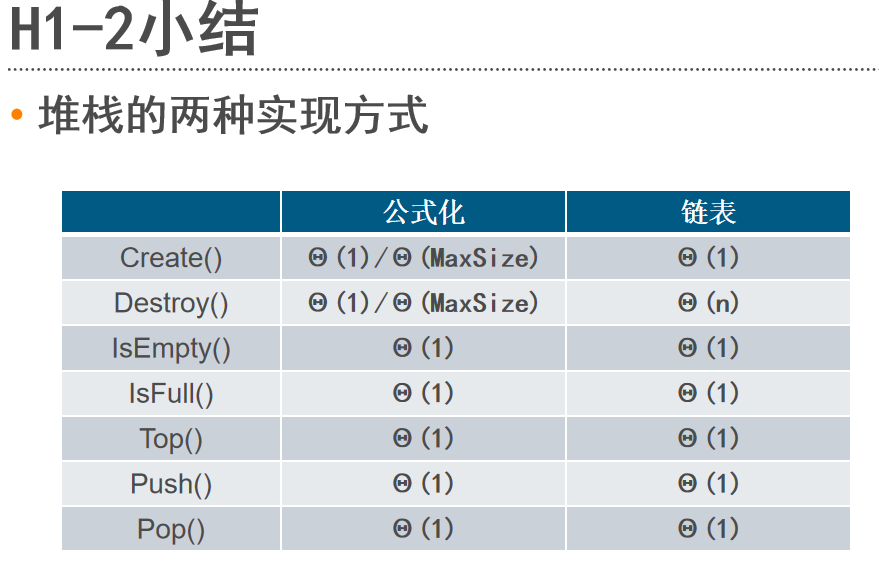
堆栈：先进后出



栈底- 栈顶

Stack[0] stack[top]

链表表示：（栈顶作为首节点）

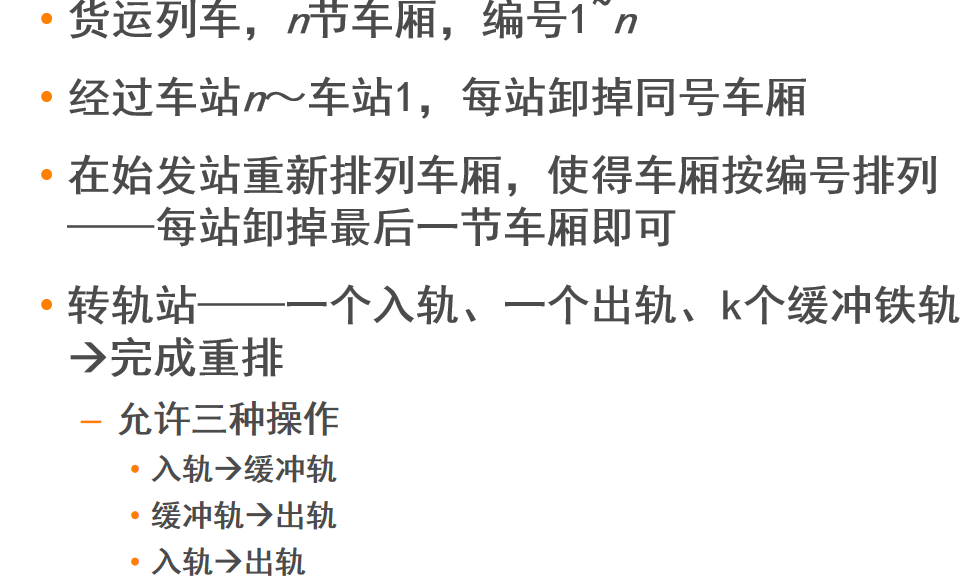


括号匹配问题：

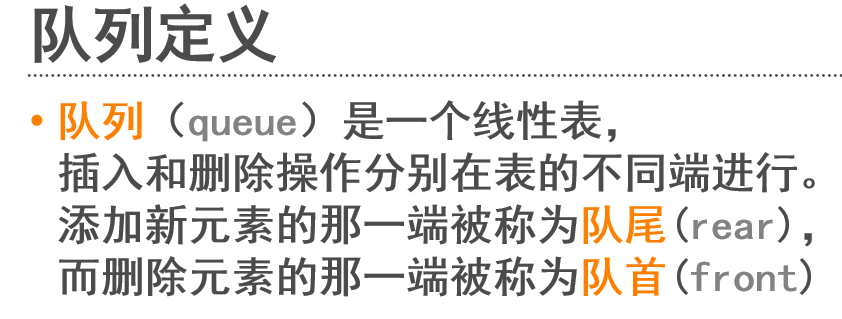
表达式左括号入栈

遇右括号pop出与之匹配

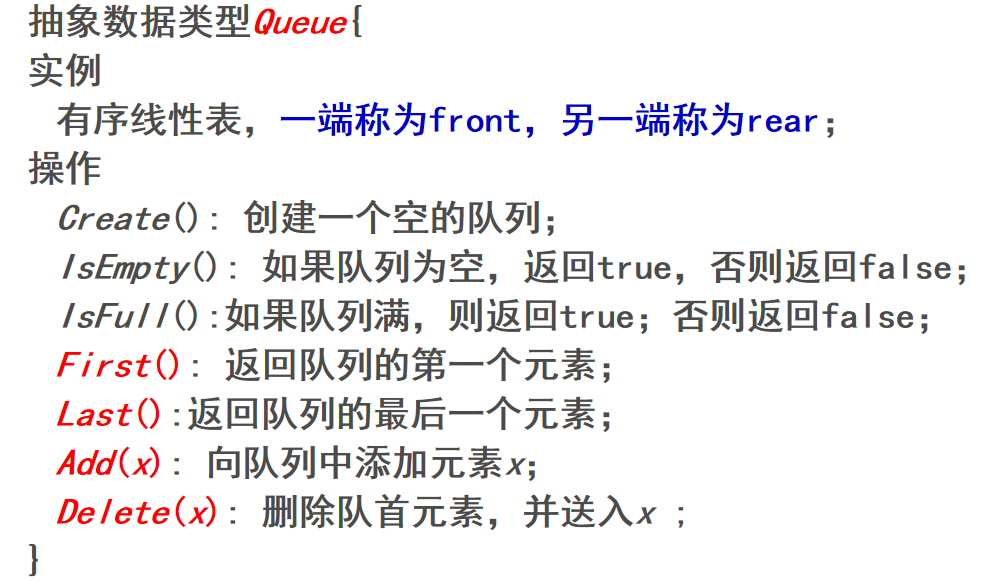
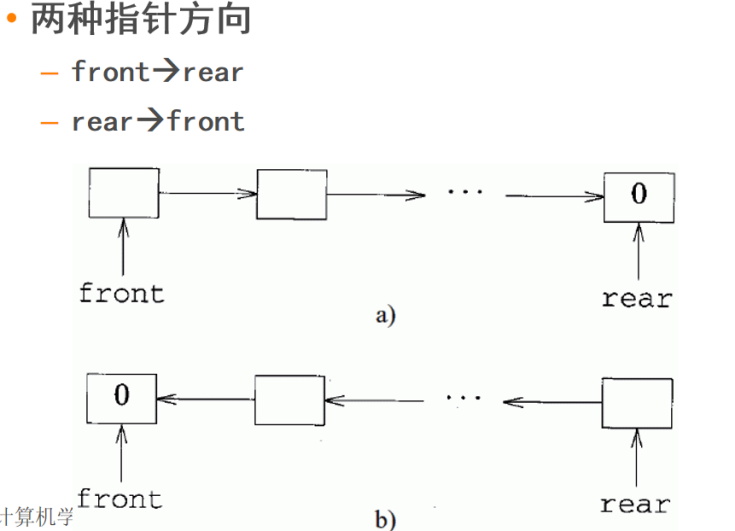
车厢重排问题： 栈底大 栈顶小 依顺序弹出

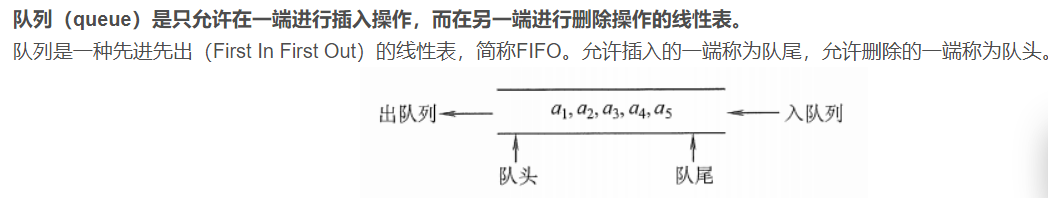
 不符合顺序的停在栈中

队列：先进先出

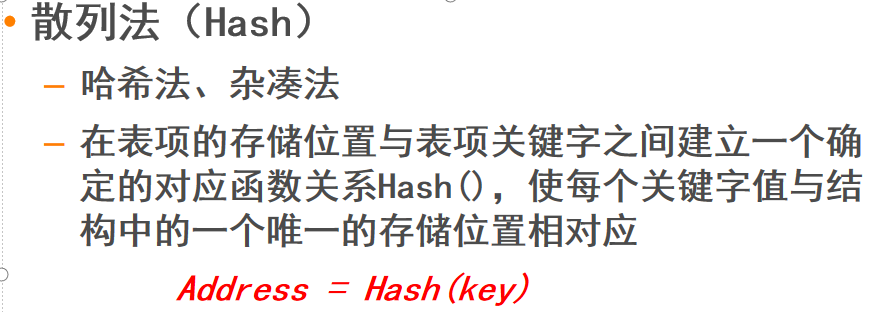


公式化： 数组中 链表：





散列：



构造哈希函数：

1.直接定址：线性计算

Hash（key）=a\*key+b

2.数字分析：key值中易区分的几位

1. 平方取中法：取关键字平方后的中间几位为哈希地址

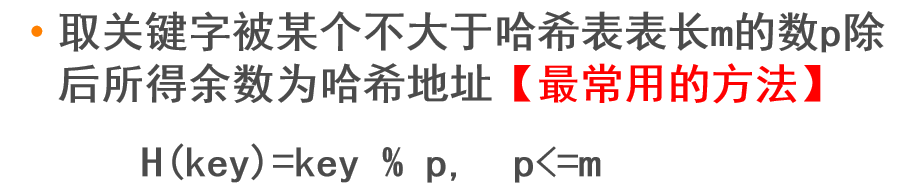
4.折叠法：将关键字分割成位数相同的几部分（最后一部分的位数可以不同），取这几部分的叠加和（舍去进位）作为哈希地址

关键字位数多

移位叠加法：把各部分的最后一位对齐相加

间界叠加法（分界叠加法）：各部分不折断，沿各部分的分界来回折叠，然后对齐相加

1. 除留余数法：



处理冲突：

线性开型寻址法（线性探测法）

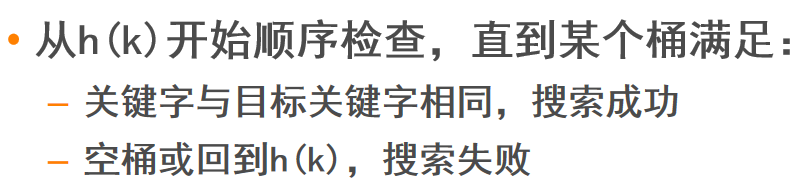
二次探测法

双散列法（再哈希法）

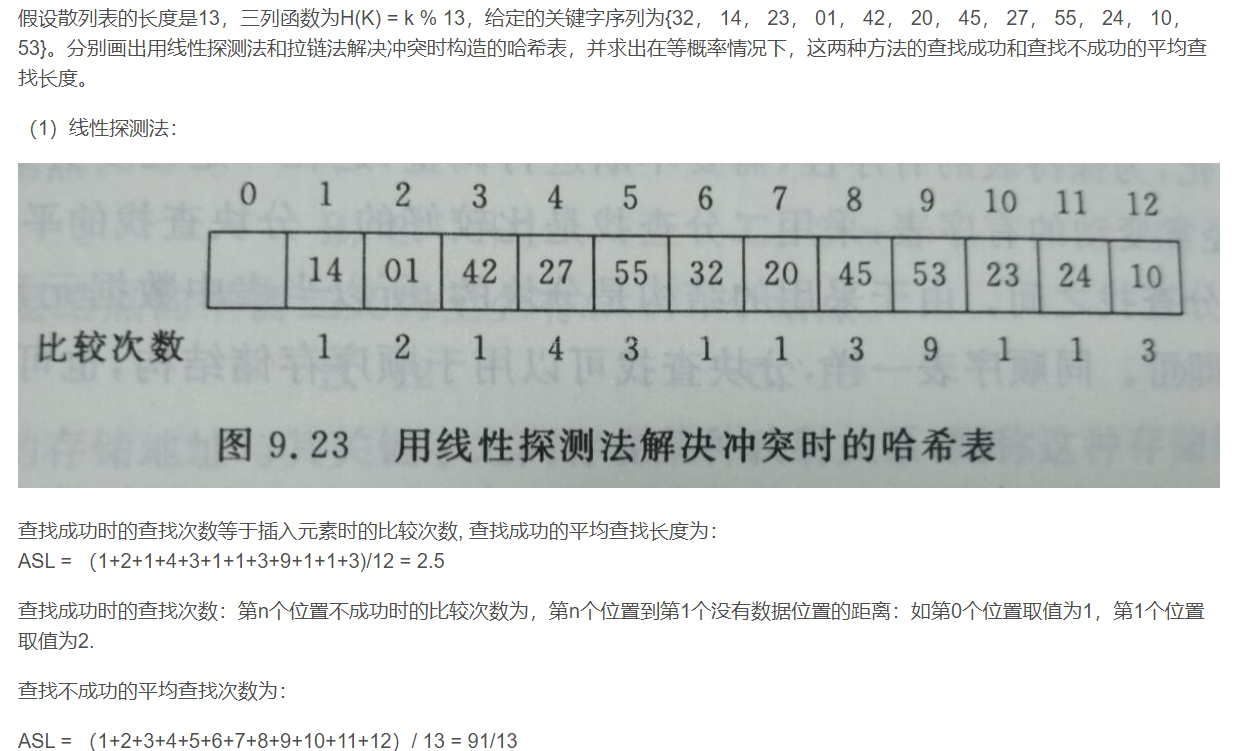
链表法（拉链法、链地址法）

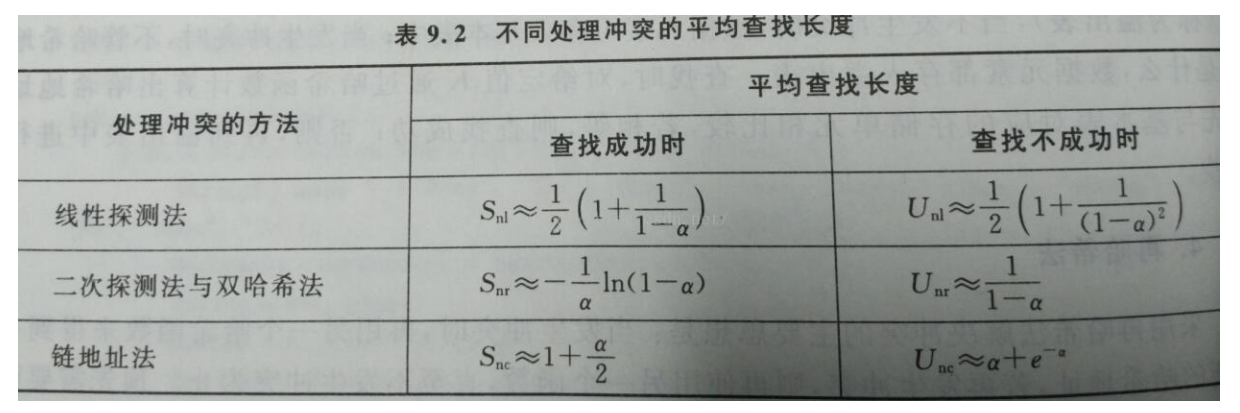


线性探测：



例题：





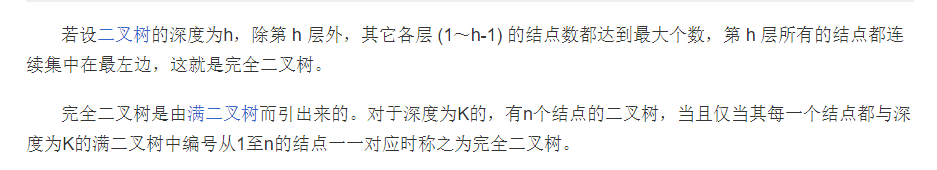
树：

度：节点孩子数

森林：有根树的集合

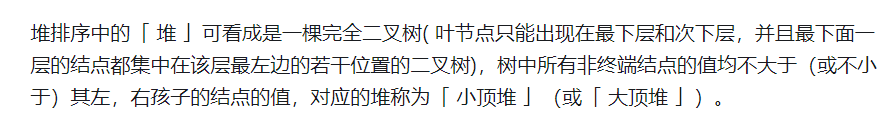


完全二叉树：



除最后一层外 结点都达到最大值

大根堆：



（大） 根节点值大于其左右孩子

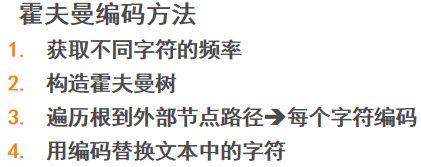
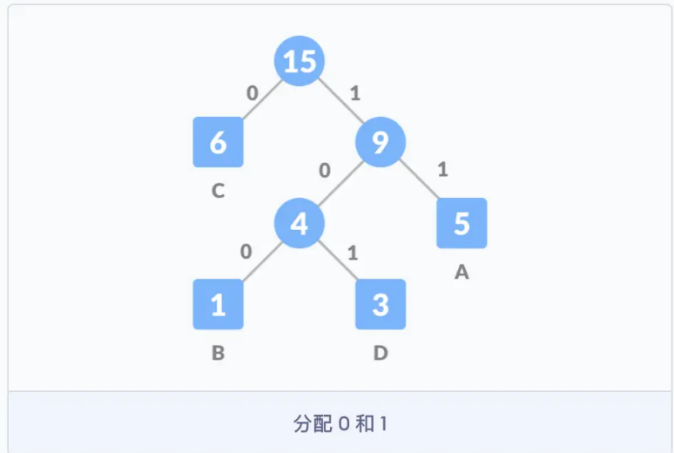
初始化：将完全二叉树转化为大根堆

数组有n个元素 [1:n] 从i=[n/2]的元素开始，检查以其为根节点的子树是否为大根堆，若不是则调整，然后依次调整后续i-1，i-2，直至根节点

**霍夫曼编码：**

高频率短码 低频率长码

任一编码不是其他编码的前缀



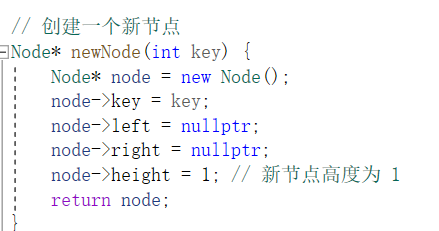
**霍夫曼树**（最佳二叉树）：



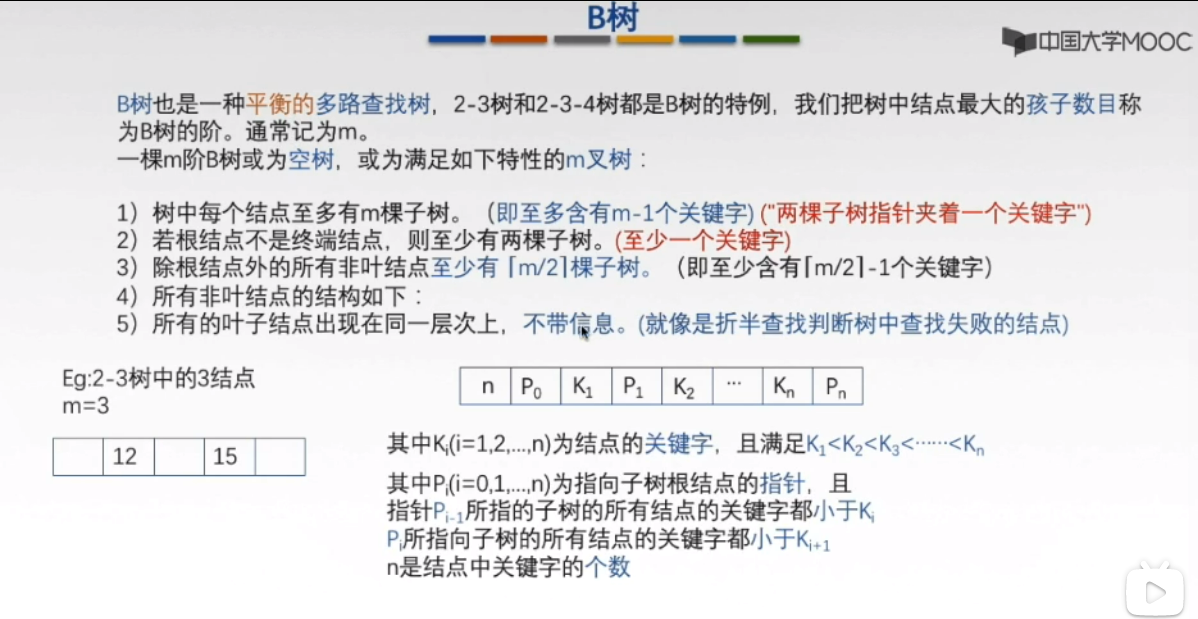
将频率最低的两两**合并**（作为一个父节点的两个子节点，父节点权值为子节点之和）

直到构成一棵树，给连线的**左右侧分配0/1**，即可得出编码

节点的建立：



B树：



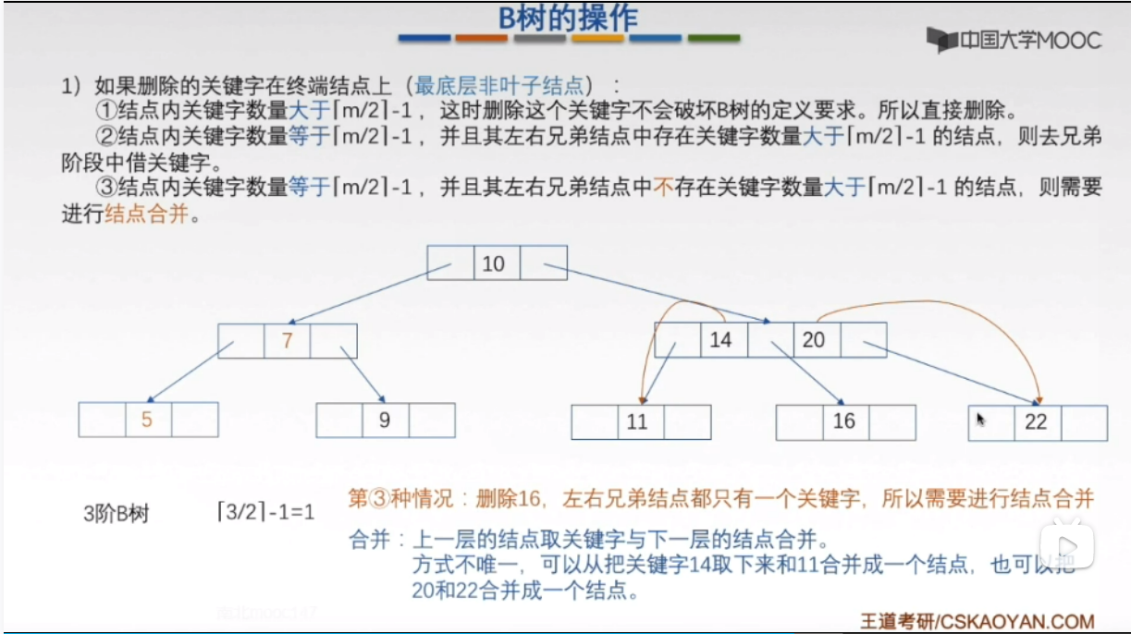
关键字：

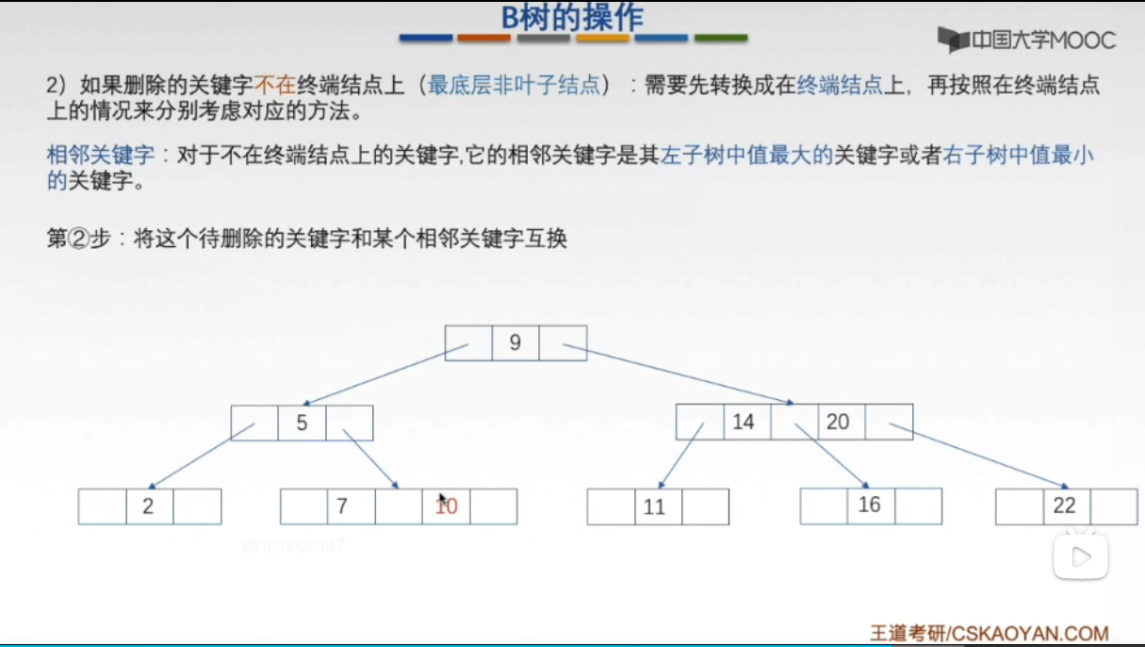
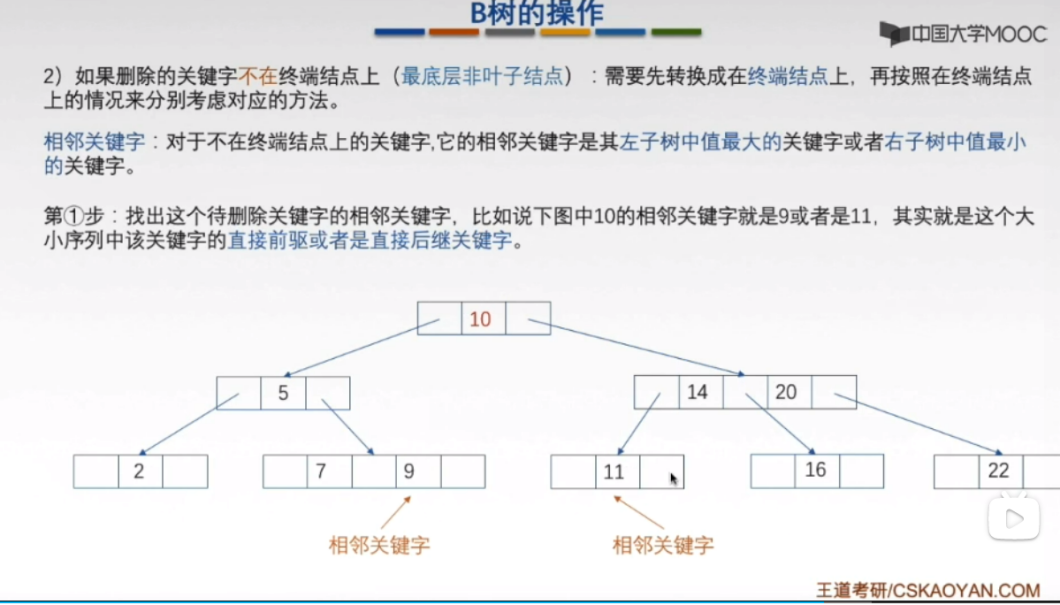
至少：[m/2]-1

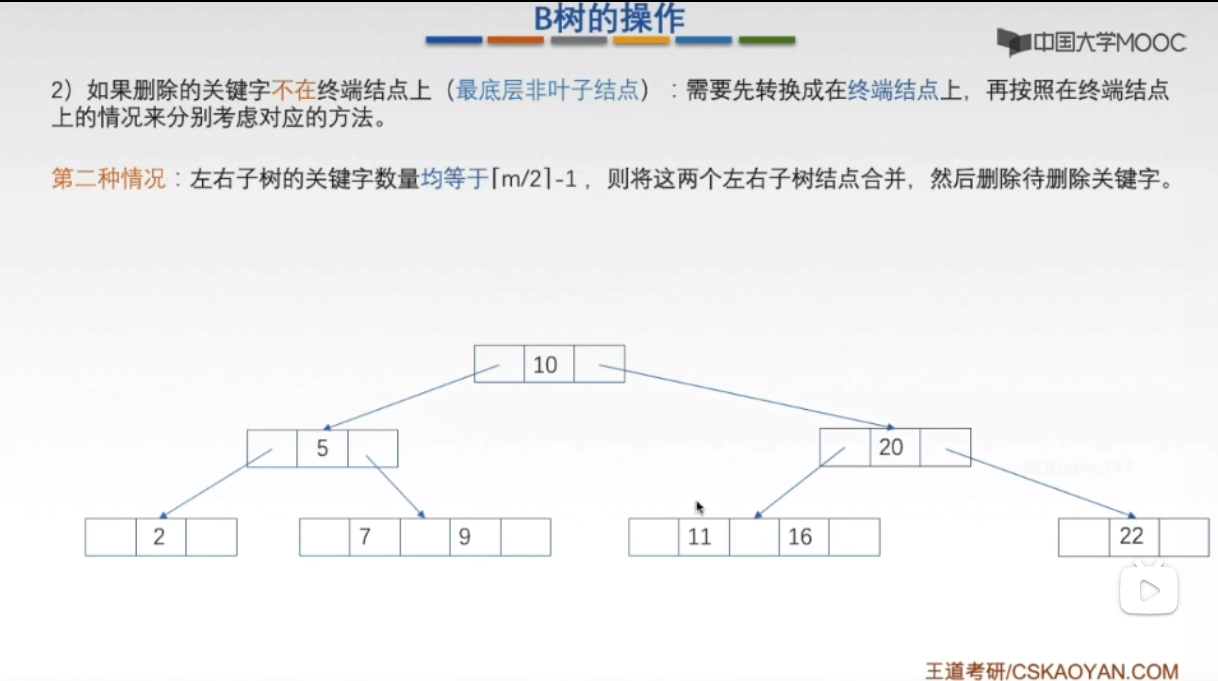
至多：m-1

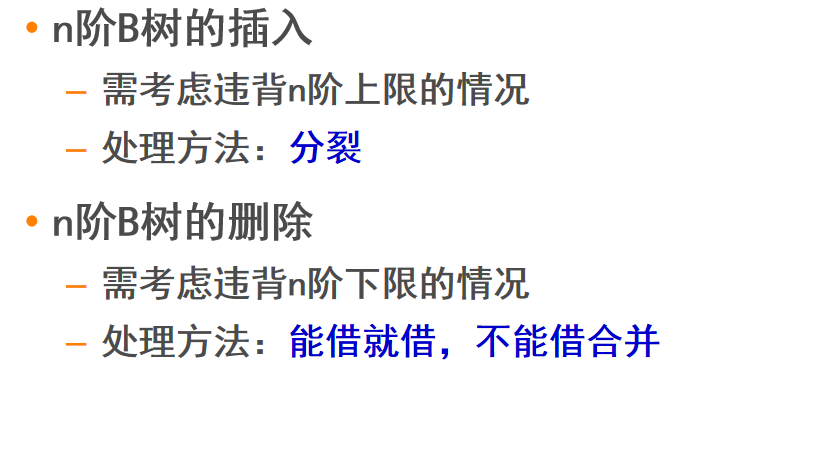
[m/2]向上取整

删除：









B+树：

