数据结构复习：

**数据结构：逻辑结构+存储结构+运算** 内存基本单位：字节

存储结构，又称物理结构——顺序 链式 索引 散列

**取址符& \*为解引用！！！访问指针指向的内存值！！！**

函数调用：单向（实参拷贝给形参） 传指针（拷贝地址 \*a） 传引用（&a）不用拷贝，**实参形参一个变量**

线性表：具有相同数据类型数据的有限序列 顺序与链式 双向链表 循环链表

（**链表增删改操作 修改指针 找i-1 画图分析 头插法**）

**栈：后进先出，只在尾部操作**

栈与递归问题——汉诺塔



应用：数制转换（低位最后输出，看到反序想到栈） 括号匹配

表达式计算（从前往后扫，遇到数，压入操作数栈，遇到操作符，压入操作符栈，**操作符优先级比栈顶高，弹出计算** 左括号优先级高右括号优先级低）

队列：特殊线性表，先进先出，插删在两端

大量移动或存在假上溢现象——循环队列解决

空表r=f 表满（r mod m）+1=f

二维数组和广义表：

对称矩阵：矩阵的压缩 上三角下三角（压缩之后为一维数组存储，下标与原始（i，j）的对应关系）

带状矩阵：非零元素集中在主对角线：按行的顺序依次存或以对角线的顺序存储

随机稀疏矩阵：三元组存储 i，j，value 结构体数组存

十字（正交链表）：在行列方向上，将非0元素链接在一起（克服三元组移动添加的不便）

广义表：

由多个原子or子表构成的有限序列

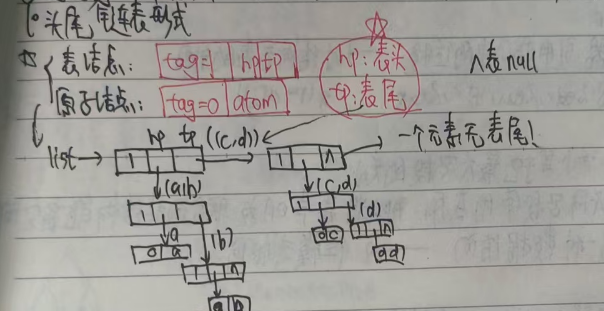
表长：第一层元素个数；表深：嵌套层数（min为1）；表头：第一个元素；表尾（除第一个元素，剩余元素构成的广义表）

Head（（a，b），（c，d））=（a，b） **表头为元素**

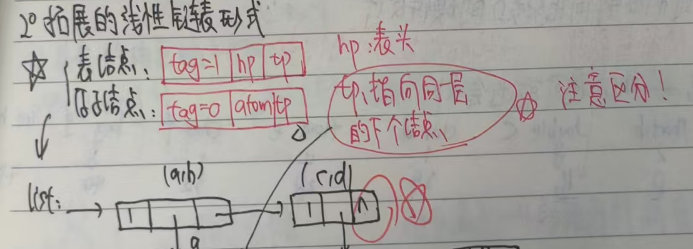
Tail（（a，b），（c，d））=（（c，d））  **表尾为广义表**

广义表中元素有表有元素，所以——表节点和原子节点

**两种存储结构：**



**注意标红 hp表头 tp表尾**



树：

除根节点外，每个节点仅有一个前驱节点（父节点）

节点的度：节点拥有的子树数目 **树的度：各节点度的max** 兄弟：同一父节点

二叉树是有序树 内部节点：除根节点之外的分支节点 堂兄弟：双亲在同一层

满二叉树与完全二叉树：

完全二叉树是指除了最后一层外，其他层的节点数都达到最大值（即前 h−1 层是满的）；而最后一层的节点从左到右连续填充​​

二叉树的性质：终端结点数n0，度为2的结点数 n0=n2+1

进入分支节点数n0+n1+n2-1=离开分支节点数n1+2n2

**通过分支数证明是一个常见的手段！**

可以用数组存树：data lc rc

或者说发过来存每个节点的父节点（根唯一）

Data A B C D E parent 0 -1 -2 -3 -4 0代表根节点

链式存储：每个节点：lc data rc 三叉链表：lc data parent rc

遍历：  
**先序的特点是能知道根 中序能区分左右子树 后序也是知道根（和先序反过来）**

画树 先序abdec 中序dbeac 层序遍历能确定根，作用同先序

线索树：

不用遍历，找到节点的直接前趋和后继

——利用树中的空指针：n个节点，2n个指针，但仅有n-1个指针用来存孩子地址

引入两个标志位 ltag rtag，0的话lc（rc）域指向左（右）还在；1的话指向前驱/后继

哈夫曼树：最优二叉树，带权最短路最小 每次选两个小的合并

哈夫曼编码 不等长编码（不能从中间截，进行解码） 编码总长最小 左0右1 字符出现次数作为权值 前缀编码

初始n个叶子结点，构成的二叉树2n-1个节点，n-1次合并，生成ht[i]，ht[i+1]……

**修改parent[i]=i，根据parent是否为0判断是否为根节点，每次选择根节点构造！**

**由parent进行回溯，最后的编码路径取反即可**

树和二叉树的转换：所有兄弟连，每个节点只保留与长子连线，左转45度

特点是无右子树，左支是孩子右支是兄弟

森林到二叉树：每棵树转为二叉，从最后一棵树开始，后一课子树做前一颗子树跟的右子

**树的遍历**：先序：先根后子树 后序：依次后根遍历根的每一个子树，再根（**同二叉树中序**，**树没有中序遍历！**）

森林遍历：先序是逐棵先序，中序是逐棵后序，森林没有后序遍历

串：

模式匹配与KMP

查找表与查找树：

描述查找路径的二叉树——判定树 左子树小于等于 右子树大于……

静态最优查找树：

原则：概率大的先被访问，两边节点被访问的概率之和尽可能的相等

PH=∑WiHi W为概率 H为层次

次优查找树：

将关键值按值大小从小到大排序，对于每个点，计算其右边和左边的概率差值（这一点使用前缀和实现），找到差值最小的点i 在i的左右相邻处找权值大的，确定根

动态查找：二叉查找树BST（允许增删）

平衡二叉树AVL 平衡因子h左-h右的绝对值

插入/生成过程——各种旋转

删除：p为叶子，修改双亲即可 p只有左/右孩，用子树代替 p有双孩，找到中序后继.前驱q，q复制于p，删q

**B树：平衡的多路查找树，应用于文件系统 对于m叉 key用来划分区间**

除根外所有非叶结点**最少m/2（上取整）子树**

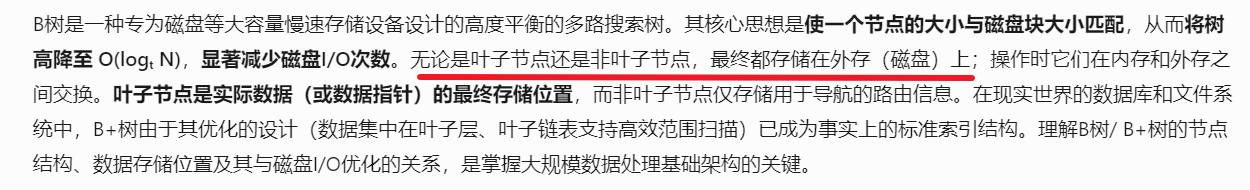
插入：若keynum+1=m，以key[m/2（上取整）]为划分点

删除： 不越下界正常 keynum=[m/2]（上取整）-1

向兄弟借：兄弟转上去，根转上来 兄弟没有：根脱下来一个，和兄弟该处一起合并

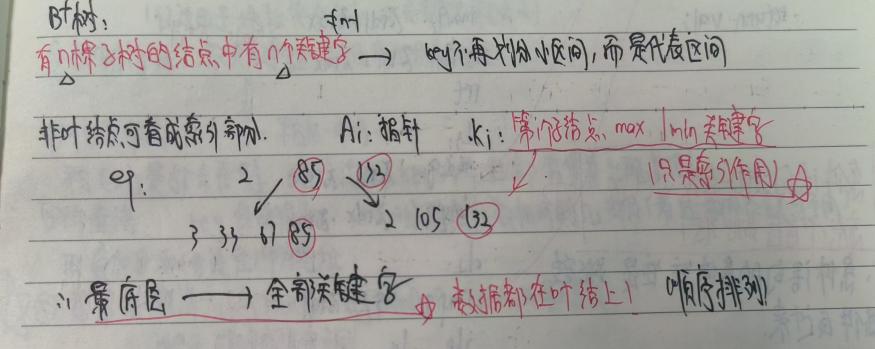
**非叶节点结构：n，A0,K1,A1,K2,……, k关键字 A为指向子树的指针**

比如：3（关键字个数） <49的 49 49-58的 58 58-72的 72 >72的



增删改查：分裂与合并

**B+树：有n个子树的结点中有n个关键字，key不再划分区间，而是代表区间**



键树：字符树

哈希表：一个有限的连续的地址空间（也称哈希表的值域），存储容纳的记录

同义词：在同一地址出现冲突的关键字 （冲突解决了可能又有新冲突）

**装填因子：**α=n/m n为记录数，m为表长 特殊方法可使α>1

**直接定址法**——hash(key) = a \* key + b

**数字分析法**——取关键字的若干位作为哈希地址

**平方取中法**——将关键字key平方后，取结果的中间几位作为哈希地址

**折叠法**—— 将长的关键字分割成长度相等的几部分（最后一部分长度可以略短），然后将这些部分叠加（或进行某种其他运算如异或XOR）起来，得到的结果（或取结果的末几位）作为哈希地址

**除留余数法**——对关键字 key 除以哈希表的大小 m，取余数作为哈希地址：hash(key) = key % m。

随机数法——选随机数

处理冲突：

开放定址法——在哈希表中再找别的位置（典型代表是线性探测再散列：这个地址不行就看地址+1行不行，**可能会有聚集现象**，数据聚在一堆；或者二次探测再散列+1^2 -1^2 +2^2 -2^2……）

查找：

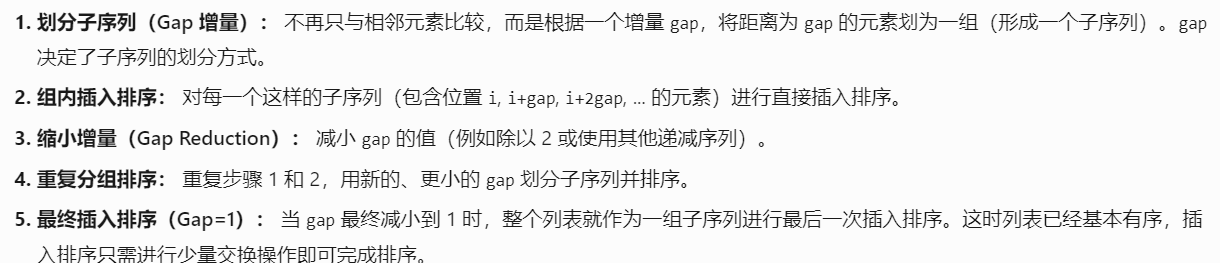
链地址法：为每个哈希地址建立一个单链表，存储所有的同义词记录

再哈希法；建立一个缓冲区……

排序：

插入排序：核心为插入 前面有序后面无序 **稳定排序**

希尔排序： **不是稳定排序**



交换排序

快排：2个指针，一次找对一个位置

（就平均而言，快排被认为是内部排序中最好的 **非稳定排序**）

选择排序：

每次从待排序列中选择min，然后和无序序列第一个交换（也就是有序序列最后一个）

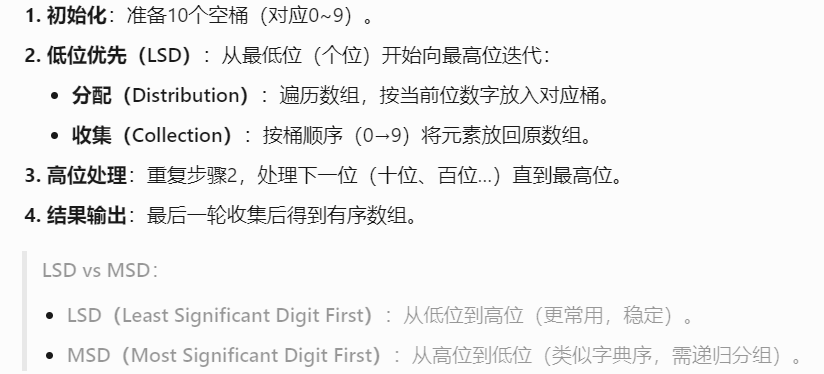
堆排序：小跟堆根小 大根堆根大

2i与2i+1，数组顺序存储 根与孩比，交换，直到叶节点 **不稳定排序**

归并排序：

从长为1开始，**稳定排序**

基数排序：基于分配和回收 时间复杂度可达n



堆排序：依次把以i=n/2（下取整），n/2-1……1为根的子树调整为堆

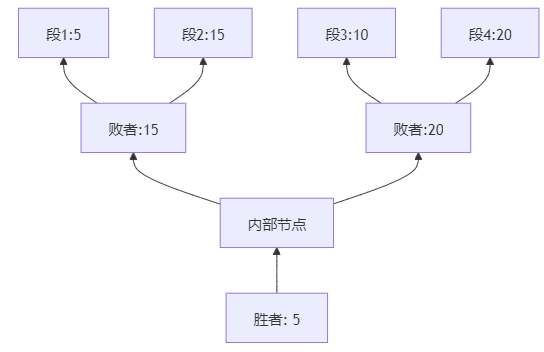
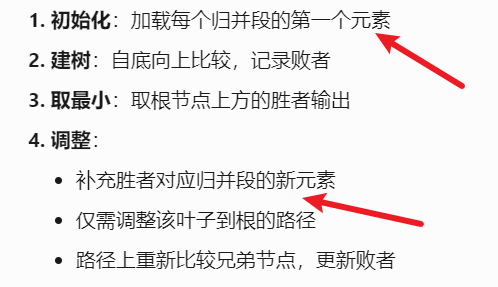
i=n/2（下取整）是最后一个非叶节点

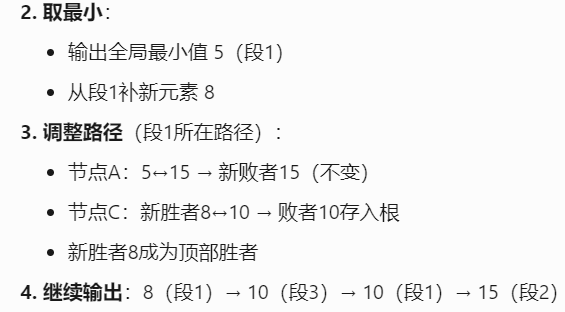
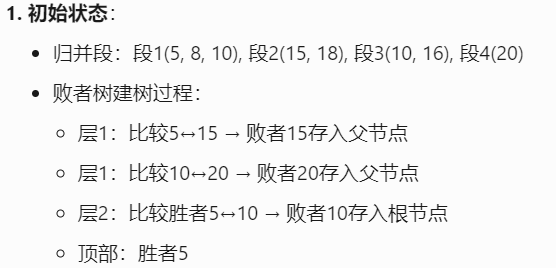
外排：

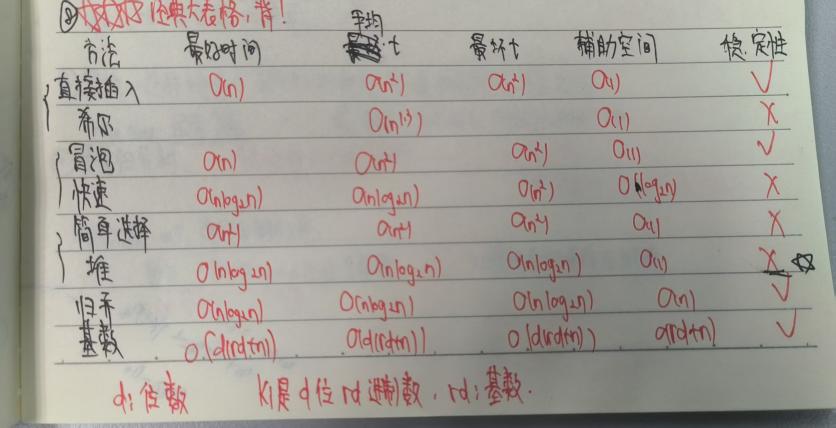
通过将数据分割成内存可处理的小块进行内部排序，然后利用归并排序的思想将所有有序片段合并成一个全局有序文件。其性能瓶颈主要在磁盘 I/O 次数​

优化方法：败者树 置换-选择排序 最佳归并树 B站

完全二叉树，叶子节点存储数据，内部节点记录"失败者"





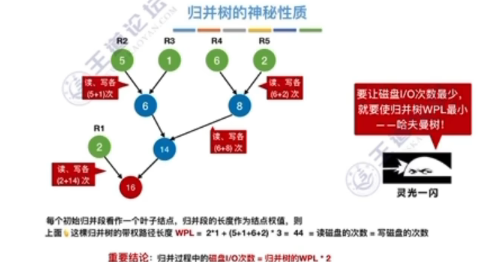


**基数排序不同于其他，不是基于比较的算法**



内存开一个区域，每次出一个min到归并段，当内存工作区所有元素都小于归并段末尾，开新一个归并段 其目的是通过减少归并段个数减少IO

最佳归并树：数字代表块的个数 哈夫曼树 注意最后带权路径长度WPL×2（读磁盘数=写磁盘数 IO=两者和）



[外部排序](https://www.bilibili.com/video/BV1V94y1q7cC?vd_source=9e2d7d85e20db70d93c3d3d4273f4b65)

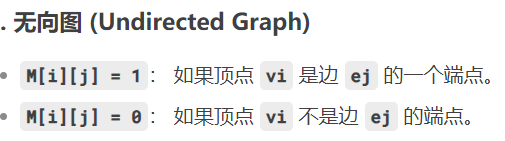
图：

稀疏图：e<nlogn 子图 连通图——无向图 强连通图——有向图

邻接矩阵（略）

邻接表

关联矩阵（一维坐标为节点，二维为边 存节点与边）



每条边 ej 连接两个顶点，所以在关联矩阵 M 中，每条边对应的列恰好有两个 1（一个对应边的起点，一个对应边的终点）。

图的遍历：深度优先与广度优先

连通图的生成树：（生成树一般不唯一）

最小生成树

Prim算法：两个集合 初始V为V0 U为V1-5 然后找跨越这两个集合的边，找权重min，更新集合 **n^2复杂度 适合稠密图！**

Kruskal算法：所有边升序排序，依次加入最小生成树且不要产生环路 **eloge 适合稀疏图**

有向无换图DAG与拓扑排序——入度为0优先算法（一个栈or队列，存deg=0的待输出点）

最短路径算法：

Dijkstra算法：和prim一样维护两个集合，从源点出发找权值最小的弧，过这个点更新其他点，入集合，继续

记录路径——记录父节点

Floyd算法：三重循环——

任意两点i和j的最短路径有两种可能：

不经过k：保持原路径长度

经过k：路径分解为i→k和k→j两段

For(int k=0;k<n;k++)

For(int v=0;v<n;v++)

For(int w=0;w<n;w++)