1 二叉树:由结点的有限集合构成:或者为空集/或者由一个根节点和两棵不相交的左子树和右子树的二叉树组成,结点是有次序的是有序树
2 n 个节点的二叉树共有 f(n)种(catalan 数)
3 内部结点 = 分支节点, 度数:结点子树的数目 层数:根节点层数为 0
4 满二叉树:一棵二叉树的结点或为树叶,或者有两棵非空子数,称作满二叉树
5 完全二叉树:最多只有下面两层结点度数小于 2,最下面一层结点集中在最左面连续的位置上
6 完全二叉树叶子结点只可能在最下面两层出现,根节点到各个结点的路径长度之和在具有相同节点数的二叉树中达到最小值
7 扩充二叉树:结点出现空指针->增加一个空树叶,成为满二叉树,新增加空树叶个数为原来结点个数+1
8 扩充二叉树:外部路径长度 = 内部路径长度 + 2 * n (归纳法, 对内部结点个数 n 进行归纳)
9 满二叉树定理:非空满二叉树树叶数=分支节点数+1 ==> 非空二叉树空子数数目=节点数+1
10 任意的二叉树,$n_0 = n_2 + 1$, 二叉树的第 i 层最多有 2^i 个结点
11 有 n 个节点的 k 叉树中,n(k-1) + 1 个指针是空的
12 二叉树的高度定义为树的层数(层数最大的结点的层数+1),深度定义为最长路径长度(高度减一)
13 有 n 个节点的完全二叉树高度为$[\log_2(n + 1)]$(向上取整)
14 完全二叉树按广度优先遍历编号,i 的父节点为[(i- 1) / 2],左子节点 2i+1,右子节点 2i+2
15 i 为偶数时,左兄弟 i-1,i 为奇数时,右兄弟 i+1
16 深度优先访问,前序/中序/后序 时间效率 O(n), 空间效率 O(n)(精确来说 O(树深))
17 已知先序和中序序列,可以唯一确定一棵二叉树,但只知道前序和后序不能唯一确定
18 非递归深度优先访问二叉树(书) 利用栈,要判断栈非空
19 前序 访问->右儿子入栈->指向左儿子->为空弹栈
20 中序 (入栈->指向左儿子)$_n$->出栈访问->指向右儿子
21 后序 (入栈->指向左儿子)$_n$->弹栈->指向右儿子->进栈->访问->指针置空
22 时间代价 O(n) 栈的深度与树的深度相关,最好 O(log n) 最坏 O(n)
23 广度优先:自上而下,从左到右 时间复杂度 O(n), 空间复杂度 O(n)(完全二叉树取到最大值(n+1)/2)
24 链式存储二叉树(二叉链表)(三叉链表->增加指向父节点的指针)
25 非完全二叉树的顺序存储->很多空节点 链式存储:不浪费空间,插入删除方便
26 完全二叉树的顺序存储:是最简单最节省空间的存储方式 存储结构上是线性的,逻辑结构上仍然是二叉树结构
27 二叉搜索树:或者是一棵空树,或者任何一个结点左子树结点都比他小,右子树都比他大
28 按照中序周游打印出来得到由小到大的排列,结点的值唯一
29 插入:进行一次失败的查找,再执行插入(插入算法:书)
30 删除:没有左子树->用右子树的根代替被删除的结点 有左子树->用左子树中序遍历的最后一个结点代替被删除的结点(改进后的)
31 (最小值)堆:一个关键码序列 $K_i < K_{2i+1}$ $K_i < K_{2i+2}$
32 堆中的数据局部有序,兄弟之间没有限定大小关系,堆不唯一,是一个可用数组表示的完全二叉树
33 建堆:从 n/2 - 1 的位置开始调整到树根
34 插入新元素:插入堆的最后位置,向上调整

35 移除最小值:将最后一个元素移到根节点,向下调整
36 建堆效率:O(n)(级数求和/放缩)
37 插入/删除时间代价 O(log n)
38 优先队列:用堆可以自然的实现
39 Huffman 树: 等长编码需要 $\log_k n$ 位, 可以使经常出现的字符编码较短,减少编码长度
40 扩充二叉树/外部路径长度 ==> 最小带权路径长度的二叉树称作 Huffman 树/最优二叉树
41 按权重排序,拿走权最小的两个节点,合并,重复至只剩一个元素
42 Huffman 编码:二进制编码长度最小->外部路径长度最小 任何一个字符的编码都不是另一个字符编码的前缀
43 K 叉 Huffman 树