# 1、定义

树是节点的有限集合。

相关概念：

孩子、双亲、度、叶子（终端节点）、根（非终端节点）、有序树、无序树、祖先、子孙

深度、节点深度、树深度

# 2、分类

## 二叉树

二叉树的每个结点至多只有二棵子树(不存在度大于2的结点)，二叉树的子树有左右之分，次序不能颠倒。二叉树的第i层至多有2i-1个结点；深度为k的二叉树至多有2k-1个结点；对任何一棵二叉树T，如果其终端结点数为n0，度为2的结点数为n2，则n0=n2+1。



遍历：前序遍历、中序遍历、后序遍历

### 完全二叉树

满二叉树：除最后一层无任何子节点外，每一层上的所有结点都有两个子结点。也可以这样理解，除叶子结点外的所有结点均有两个子结点。节点数达到最大值，所有叶子结点必须在同一层上。

　　满二叉树的性质：

　　1) 一颗树深度为h，最大层数为k，深度与最大层数相同，k=h;

　　2) 叶子数为2h;

　　3) 第k层的结点数是：2k-1;

　　4) 总结点数是：2k-1，且总节点数一定是奇数。



### 满二叉树

完全二叉树：若设二叉树的深度为h，除第 h 层外，其它各层 (1～(h-1)层) 的结点数都达到最大个数，第h层所有的结点都连续集中在最左边，这就是完全二叉树。

**注：**完全二叉树是效率很高的数据结构，堆是一种完全二叉树或者近似完全二叉树，所以效率极高，像十分常用的排序算法、Dijkstra算法、Prim算法等都要用堆才能优化，二叉排序树的效率也要借助平衡性来提高，而平衡性基于完全二叉树。

## 二叉查找树

二叉查找树又称为是二叉排序树（Binary Sort Tree）或二叉搜索树。二叉排序树或者是一棵空树，或者是具有下列性质的二叉树：

　　1) 若左子树不空，则左子树上所有结点的值均小于它的根结点的值；

　　2) 若右子树不空，则右子树上所有结点的值均大于或等于它的根结点的值；

　　3) 左、右子树也分别为二叉排序树；

4) 没有键值相等的节点。

### 性质

二叉查找树的性质：对二叉查找树进行中序遍历，即可得到有序的数列。

### 复杂度

二叉查找树的时间复杂度：它和二分查找一样，插入和查找的时间复杂度均为O(logn)，但是在最坏的情况下仍然会有O(n)的时间复杂度。原因在于插入和删除元素的时候，树没有保持平衡（比如，我们查找上图（b）中的“93”，我们需要进行n次查找操作）。我们追求的是在最坏的情况下仍然有较好的时间复杂度，这就是平衡查找树设计的初衷。

### 插入

　　二叉查找树的高度决定了二叉查找树的查找效率。

　　二叉查找树的插入过程如下：

　　1) 若当前的二叉查找树为空，则插入的元素为根节点;

　　2) 若插入的元素值小于根节点值，则将元素插入到左子树中;

3) 若插入的元素值不小于根节点值，则将元素插入到右子树中。

### 删除

　　二叉查找树的删除，分三种情况进行处理：

　　1) p为叶子节点，直接删除该节点，再修改其父节点的指针（注意分是根节点和不是根节点），如图a;

　　2) p为单支节点（即只有左子树或右子树）。让p的子树与p的父亲节点相连，删除p即可（注意分是根节点和不是根节点），如图b;

　　3) p的左子树和右子树均不空。找到p的后继y，因为y一定没有左子树，所以可以删除y，并让y的父亲节点成为y的右子树的父亲节点，并用y的值代替p的值；或者方法二是找到p的前驱x，x一定没有右子树，所以可以删除x，并让x的父亲节点成为y的左子树的父亲节点。如图c。







## 平衡二叉树

我们知道，对于一般的二叉搜索树（Binary Search Tree），其期望高度（即为一棵平衡树时）为log2n，其各操作的时间复杂度O(log2n)同时也由此而决定。但是，在某些极端的情况下（如在插入的序列是有序的时），二叉搜索树将退化成近似链或链，此时，其操作的时间复杂度将退化成线性的，即O(n)。我们可以通过随机化建立二叉搜索树来尽量的避免这种情况，但是在进行了多次的操作之后，由于在删除时，我们总是选择将待删除节点的后继代替它本身，这样就会造成总是右边的节点数目减少，以至于树向左偏沉。这同时也会造成树的平衡性受到破坏，提高它的操作的时间复杂度。于是就有了我们下边介绍的平衡二叉树。

　　平衡二叉树（Balanced Binary Tree）又被称为AVL树（有别于AVL算法），且具有以下性质：它是一棵空树或它的左右两个子树的高度差的绝对值不超过1，并且左右两个子树都是一棵平衡二叉树。平衡二叉树的常用算法有红黑树、AVL树等。在平衡二叉搜索树中，我们可以看到，其高度一般都良好地维持在O(log2n)，大大降低了操作的时间复杂度。

　　最小二叉平衡树的节点的公式如下：

　　F(n)=F(n-1)+F(n-2)+1

　　这个类似于一个递归的数列，可以参考Fibonacci数列，1是根节点，F(n-1)是左子树的节点数量，F(n-2)是右子树的节点数量。

### AVL树

### 红黑树

## B树

## B+树

## B\*树

## Trie树

Trie树称为字典树，又称单词查找树，Trie树，是一种树形结构，是一种哈希树的变种。典型应用是用于统计，排序和保存大量的字符串（但不仅限于字符串），所以经常被搜索引擎系统用于文本词频统计。它的优点是：利用字符串的公共前缀来减少查询时间，最大限度地减少无谓的字符串比较，查询效率比哈希树高。

　　Tire树的三个基本性质：

　　1) 根节点不包含字符，除根节点外每一个节点都只包含一个字符；

　　2) 从根节点到某一节点，路径上经过的字符连接起来，为该节点对应的字符串；

　　3) 每个节点的所有子节点包含的字符都不相同。

　　Tire树的应用：

　　1) 串的快速检索

　　给出N个单词组成的熟词表，以及一篇全用小写英文书写的文章，请你按最早出现的顺序写出所有不在熟词表中的生词。

在这道题中，我们可以用数组枚举，用哈希，用字典树，先把熟词建一棵树，然后读入文章进行比较，这种方法效率是比较高的。

　　2) “串”排序

　　给定N个互不相同的仅由一个单词构成的英文名，让你将他们按字典序从小到大输出。用字典树进行排序，采用数组的方式创建字典树，这棵树的每个结点的所有儿子很显然地按照其字母大小排序。对这棵树进行先序遍历即可。

　　3) 最长公共前缀

　　对所有串建立字典树，对于两个串的最长公共前缀的长度即他们所在的结点的公共祖先个数，于是，问题就转化为求公共祖先的问题。

# 3、代码

## 数组实现

## 链表实现

# 4、应用

压缩软件——赫夫曼树

搜索——人机对战（不断做树的搜索，找到最优的路径）