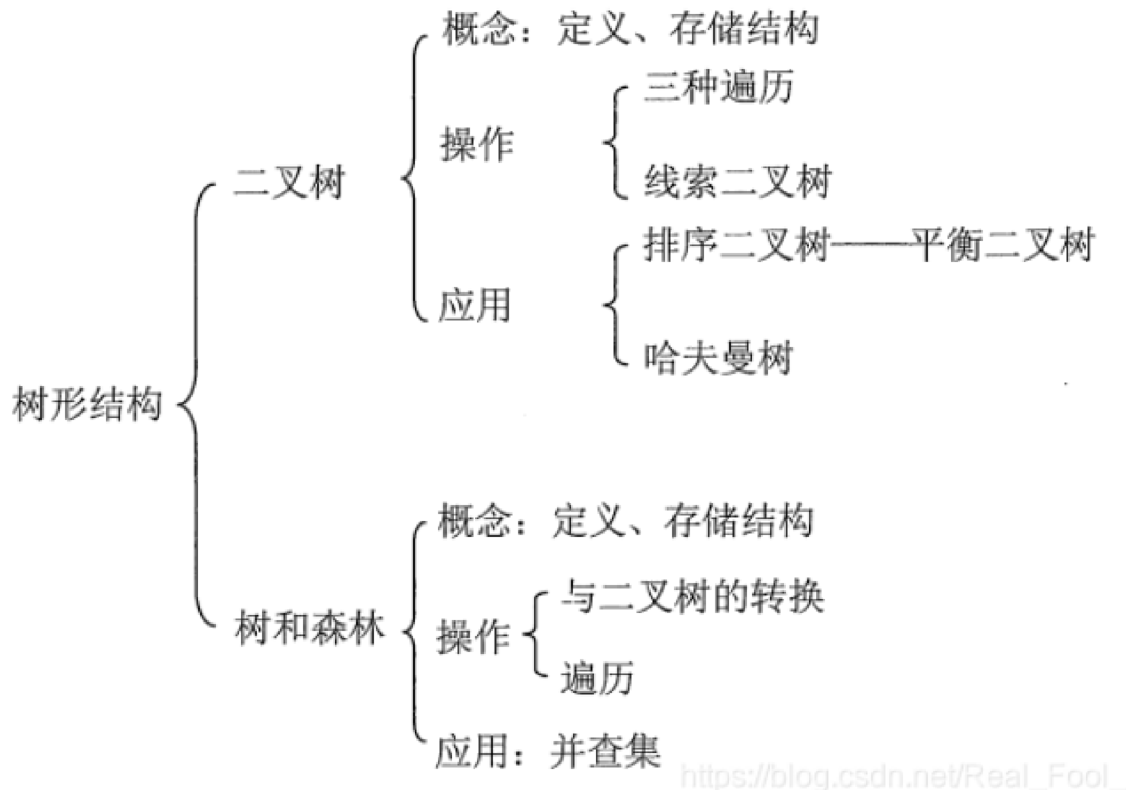
树



- **性质**

  树中的结点数等于所有结点的度数加1.
  度为m mm的树中第i ii层上至多有`m^{i-1}`个结点（i > = 1 ）
  高度为h 的m叉树至多有`(m^{h}-1)/(m-1)`个结点。
  具有n个结点的m叉树的最小高度为`log m (n*(m-1)+1)`

- **二叉树**

  每个节点至多只有两棵子树

  一些特殊的二叉树：

  - **满二叉树**

    一棵高度为h，且含有`2^{h}-1`个结点的二叉树称为满二叉树，即树中的每层都含有最多的结点。满二叉树的叶子结点都集中在二叉树的最下一层，并且除叶子结点之外的每个结点度数均为2 。可以对满二叉树按层序编号:约定编号从根结点(根结点编号为1)起，自上而下，自左向右。这样，每个结点对应一个编号,对于编号为i的结点，若有双亲，则其双亲为`i/2`,若有左孩子，则左孩子为`2i`;若有右孩子，则右孩子为`2i+1`

  - **完全二叉树**

    高度为h、有n 个结点的二叉树，当且仅当其每个结点都与高度为h 的满二叉树中编号为1~n的结点一一对应时，称为完全二叉树

    特点：

    1.若`i<=n/2`, 则结点i 为分支结点，否则为叶子结点。
    2.叶子结点只可能在层次最大的两层上出现。对于最大层次中的叶子结点，都依次排列在该层最左边的位置上。
    3.若有度为1 的结点，则只可能有一个，且该结点只有左孩子而无右孩子(重要特征)。
    4.按层序编号后，一旦出现某结点(编号为i)为叶子结点或只有左孩子，则编号大于i 的结点均

为叶子结点。
5.若n 为奇数，则每个分支结点都有左孩子和右孩子;若n 为偶数，则编号最大的分支结点(编号为`n/2`)只有左孩子，没有右孩子，其余分支结点左、右孩子都有。

   - **二叉搜索树（BST）**

     对于每一个节点，左孩子的值都要小于该节点，右孩子的值都要大于该节点

   - **二叉平衡树（AVL）**

     任意节点的左子树和右子树的深度之差不超过1

- **二叉树性质**

  1.任意一棵树，若结点数量为m,则边的数量为`n一1`。
  2.非空二叉树上的叶子结点数等于度为2的结点数幼加1，即`n。=n2十1`。
  3.非空二叉树上第k层上至多有`2^{k-1}`个结点(k≥1).
  4.高度为h的二叉树至多有`2^{h}-1`个结点(h≥1)。
  5.对完全二叉树按从上到下、从左到右的顺序依次编号1,2.*，n,则有以下关系:
  。i>1时，结点的双亲的编号为`i/2`，即当为偶数时，它是双亲的左孩子；当为奇数时，它是双亲的右孩子。
  。当`2i<n`时，结点的左孩子编号为`2i`,否则无左孩子。
  。当`2i十1≤n`时，结点的右孩子编号为`2i十1`，否则无右孩子。
  。结点所在层次（深度）为`{log 2 i}十1`。
  6.具有n个(m>0)结点的完全二叉树的高度为`{log 2 n}十1`。

- **遍历二叉树**

  1.前序遍历： 根---左---右

  2.中序遍历： 左---根---右

  3.后序遍历： 左---右---根

  4.层次遍历： 每一层从左到右读取

- **二叉搜索树（BST）**

  对于每一个节点，左孩子的值都要小于该节点，右孩子的值都要大于该节点

  对于BST，理想情况下（树为完全二叉树或者AVL树等），插入、查找、删除的复杂度均为`O(log n)`,最坏情况下，退化成一条链表，复杂度变为`O(n)`

- **二叉平衡树(AVL)**

  任意节点的左子树和右子树的深度之差不超过1（平衡因子）

  当某一节点失衡时，需要通过左旋和右旋操作来使二叉树平衡

  查找、插入、删除操作均为`O(log n)`，左旋右旋操作`O(1)`

  大致上，一颗AVL树的高度最多为`1.44log(N+2)-1.328`

  在高度为h的AVL树中，最少节点数`S(h)`由`S(h)=S(h-1)+S(h-2)+1`给出

- **伸展树（splay tree）**

  每一次查找节点后将该节点移动至根节点

  保证从空树开始任意连续对树操作M次最多花费`O(Mlog N)`时间

- **B树和B+树**

  因为磁盘和内存读写速度有明显的差距，磁盘中存储的数据需要先读取到内存中才能进行高速的检索。而数据库当中存储着海量的数据，光是数据库索引就有可能占据几个GB甚至更大的空间。当我们要查找数据的时候，显然不可能把整个索引树读到内存中。因此，我们只能以索引树的节点为基本单元，每次把单一节点从磁盘读取到内存当中，进行后续操作。

  如果磁盘当中的索引树是一棵平衡二叉树，查找的时候，在最坏情况下，磁盘I/O的次数等于索引树的高度。

  为了减少磁盘I/O，我们需要把原本“瘦高”的树结构变得“矮胖”，让每一个节点承载更多的元素，拥有更多的孩子。

  B树和B+树，就是这样的数据结构，因此它们非常适合做数据库和文件系统的索引。

  B树单一节点拥有的最多子节点个数，称为B树的**阶**，一颗`m`阶B树有如下特征：

  1.根节点至少有两个子节点。
  2.每个中间节点都包含`k-1`个元素（也被称为关键字）和`k`个孩子（子树），其中`ceil（m/2）<=k<=m`。
  3.每一个叶子节点都包含`k-1`个元素，其中`m/2<=k<=m`。
  4.所有的叶子节点都位于同一层。
  5.每个节点中的元素从小到大排列，节点当中任意元素的左子树都小于他，右子树都大于他

  插入操作：先找到对应位置插入，（节点未满就先进入节点，否则进入子树），未溢出无需调整，否则节点内第`ceil(m/2)`个节点上移到父节点，左右元素分裂为两棵子树

  删除操作：删除非叶节点上元素，类似BST，直接前/后驱替换==》转换成删除叶节点元素，，，对于叶节点元素，若删除后未下溢出则无需调整，否则，，，1.跟左右兄弟借：邻兄进入父节点，父亲下来该节点，，，2.不够借：父下移到左，然后右并过来，注意父节点可能下溢出，同理操作

B+树

叶节点层为链表结构，可快速通过头指针访问，非叶节点可帮助快速导航到叶节点**（B+树节点元素个数和子树分支个数相同）**

和B树区别：

B树：所有节点的关键字都有指向对应记录的指针，，，对于节点，最多m个分支，m-1个元素，，，顺序查找或者范围查找只能在中序遍历，效率低下

B+树：叶节点包含全部关键字及指向对应记录的指针，非叶节点只做叶结点的索引，，，对于节点，最多m个分支m个元素，，，B+树兼顾顺序查找和随即查找，方便范围查找