二叉树

性质

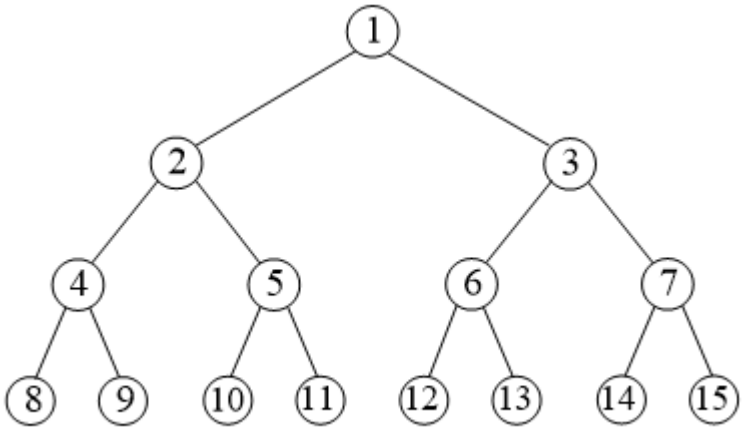
每个节点最多有两个子节点：左孩子、右孩子以它们为根的子树称为左子树、右子树。二叉树的每个节点不必全有左、右孩子，可以只有一个孩子或没有孩子，没有孩子的结点称为叶子节点。 二叉树的第i层，最多有2^{i-1}个节点

平衡二叉树

平衡二叉搜索树又被称为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***

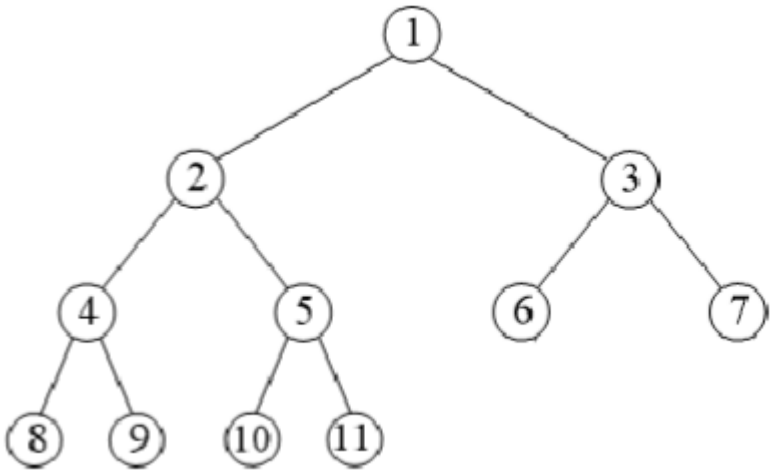
满二叉树

每一层的结点数都是满的。



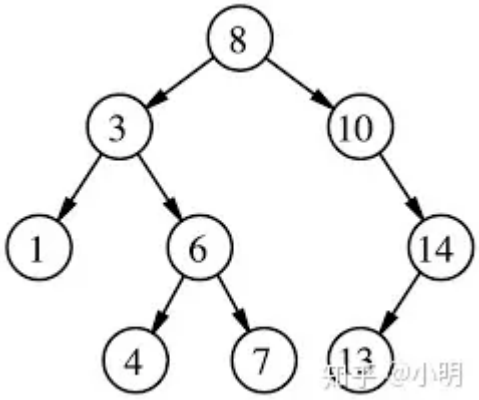
完全二叉树

完全二叉树由满二叉树转化而来，也就是将满二叉树***从最后一个节点开始删除，一个一个从后往前删除，剩下的就是完全二叉树。***



一棵有N个结点的满二叉树，树的高度是O(logN)。从根结点到叶子结点，只需要走logN步，例如N = 100万，树的高度仅有20，只需要20步就能到达100万个结点中的任意一个。

二叉搜索树（bst）



二叉搜索树（又叫二叉查找树），它是具有下列性质的二叉树：

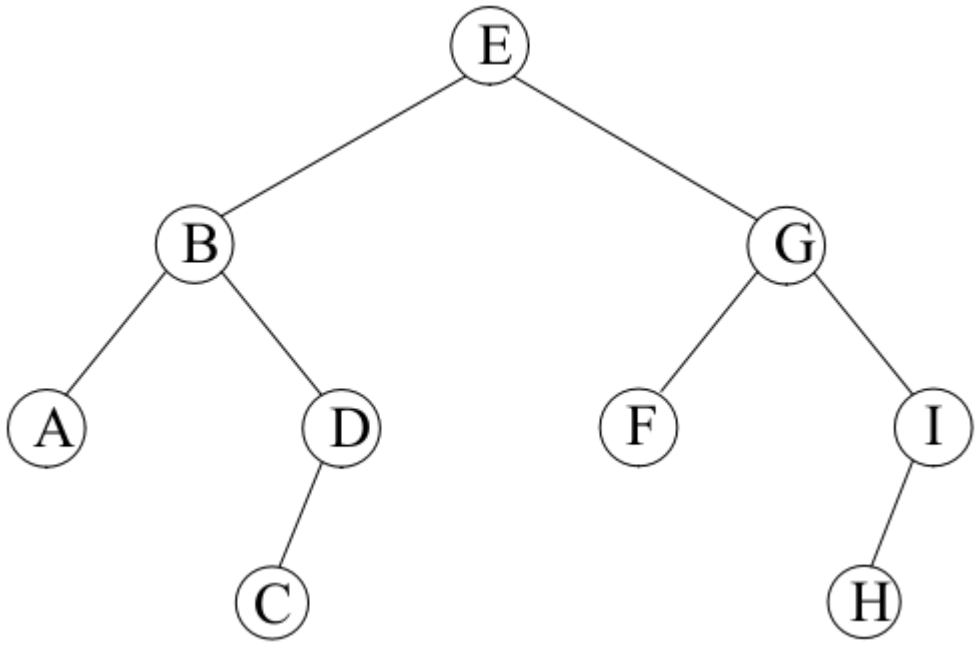
> 若左子树不空，则左子树上所有结点的值均小于它的根结点的值； 若右子树不空，则右子树上所有结点的值均大于或等于它的根结点的值； 左、右子树也分别为二叉搜索树。

总之：**二叉搜索树中，左子树都比其根节点小，右子树都比其根节点大，递归定义。**

重点来了！二叉搜索树的**中序遍历**一定是**从小到大排序**的。

> 比如上图的中序遍历结果：1，3，4，6，7，8，10，13，14

二叉树的遍历



先序遍历：按父、左儿子、右儿子的顺序访问：

EBADCGFIH

中序遍历：按左儿子、父、右儿子的顺序访问：

ABCDEFGHI

后序遍历：按左儿子、右儿子、父的顺序访问：

ACDBFHIGE

层序遍历：上下左右，一层一层的遍历

深度优先搜索（DFS）：一条路走到底

EBGADFICH

广/宽度优先搜索（BFS）：一层一层的扩展

确定一棵二叉树

唯一确定一棵二叉树的方法

在了解以何种方式能唯一确定一棵**二叉树**之前，需要先认识树的遍历方式有哪几种。

树的遍历方式

1. 先序遍历
2. 后序遍历
3. 层序遍历

二叉树的遍历方式

1. 先序遍历
2. 中序遍历
3. 后序遍历
4. 层序遍历

确定的方式

那么如何唯一确定一棵二叉树呢？这就需要结合二叉树的遍历方式来进行，主要有以下两种方式：

1. 二叉树的先序遍历+中序遍历
2. 二叉树的后序遍历+中序遍历

在记忆这两种方式时，为了避免混淆，可以这样记忆：由于中序遍历是二叉树特有的遍历方式，==所有唯一确定一棵二叉树的每一种方式中必有一个是中序遍历==，而剩下的那一种就要么是先序遍历，要么是后序遍历。

现假设存在一棵二叉树，其先序、中序和后序遍历的结果如下：

1. 先序遍历：ABDEGCFH
2. 中序遍历：DBGEACFH
3. 后序遍历：DGEBHFCA

在介绍如何利用遍历方式唯一确定一棵二叉树之前，需要重点强调：

无论利用什么方式来唯一确定一棵二叉树，其本质都是通过两种遍历结果来不断递归地**确定根结点和划分左右子树的过程！**

先序遍历+中序遍历

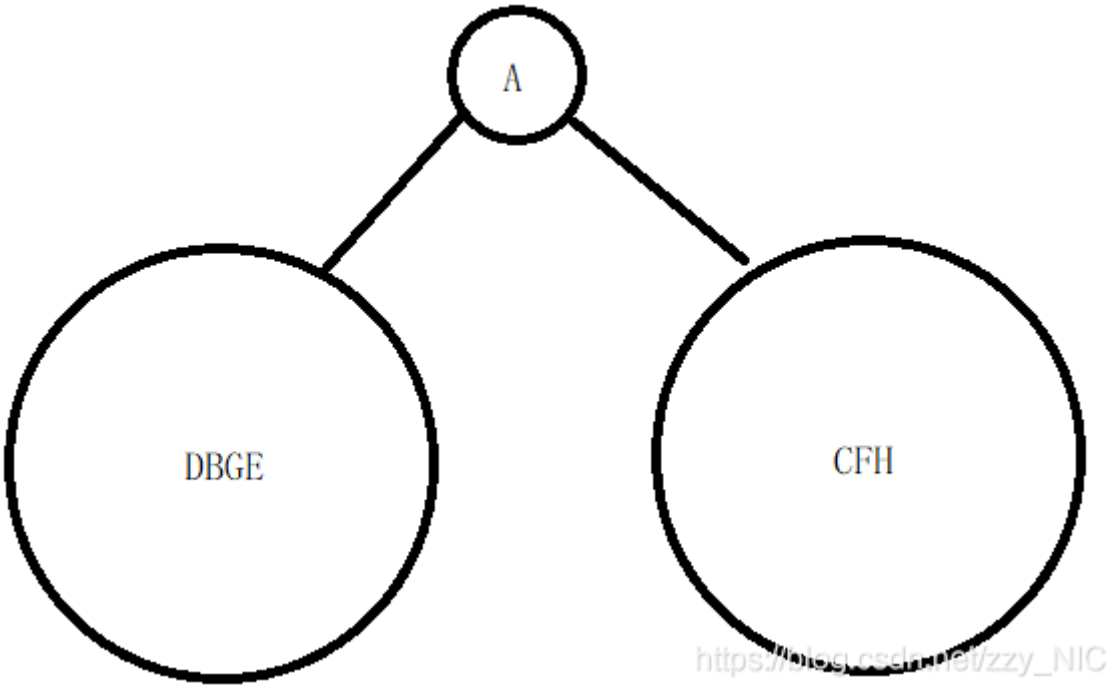
先序遍历+中序遍历的要义是：利用先序遍历确定根结点，再利用中序遍历划分左右子树

步骤一：分析整棵二叉树

对于先序遍历，其遍历方式是一个结点需要先访问自己，再访问其左子树，接着才是其右子树，故先序遍历结果中的第一个元素一定是根结点，根据上述先序遍历可以知道是A。

接着利用得到的A，在中序遍历结果中找到A所在的位置，然后便可以将A左侧的所有元素归属到根结点的左子树，A右侧的所有元素归属到根结点的右子树，而之所以这样做，是因为对于中序遍历，其遍历方式是一个结点先访问其左子树，再访问自己，接着才是其右子树，所以A前面的元素一定来自A的左子树，A后面的一定来自A的右子树。

根据上述分析，可以先画出如下图片：

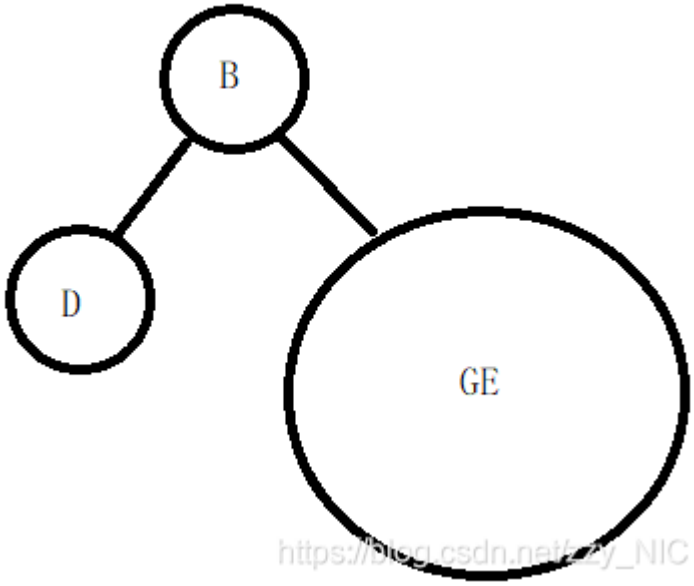


步骤二：分析A的左子树，即包含DBGE的这棵二叉树。

这时可以把包含DBGE这四个元素的先序遍历部分和中序遍历部分单独提取出来：

1. 部分先序遍历：BDEG
2. 部分中序遍历：DBGE

现在单独观察包含DBGE的这棵树和它对应的部分先序遍历、部分中序遍历，可以看到接下来的分析方式又与步骤一是一模一样的了，即先确定这个树（仅包含DBGE的这棵树）的根结点，再确定它的两棵子树，现在根结点是部分先序遍历的第一个元素，即B，根据B，在部分中序遍历中划分左右子树，即左子树包含D，右子树包含GE，得到的图片如下：



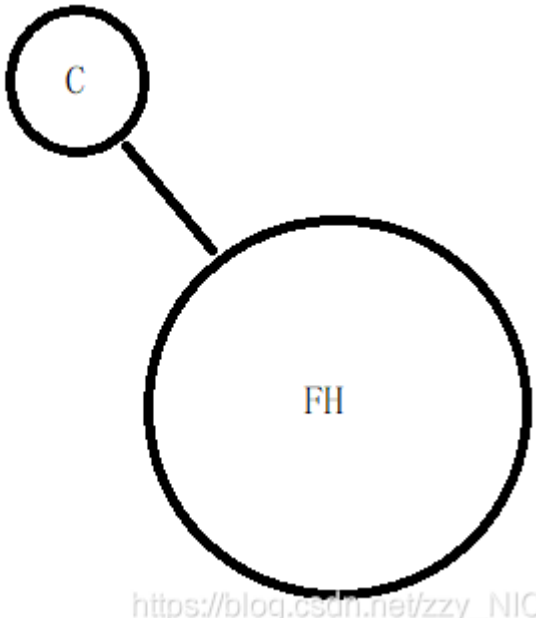
可以看到这时B的左子树只有一个结点，所以可以认为已经确定好了，那么就只需要按照步骤一的方式对B的右子树进行同样的分析即可。

步骤三：分析A的右子树，即包含CFH的这颗二叉树。

这时可以把包含CFH这三个元素的先序遍历部分和中序遍历部分单独提取出来：

1. 部分先序遍历：CFH
2. 部分中序遍历：CFH

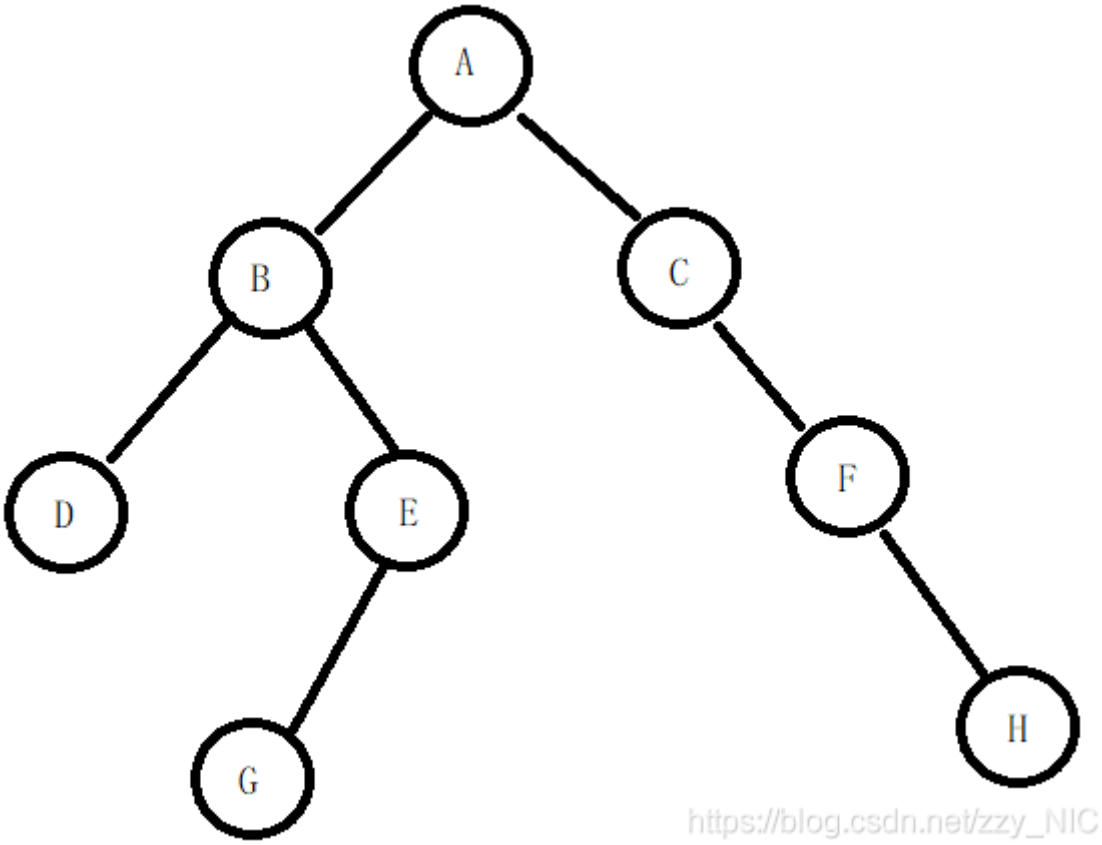
现在单独观察包含CFH的这棵树和它对应的部分先序遍历、部分中序遍历，可以看到接下来的分析方式又与步骤一是一模一样的了，即先确定这个树（仅包含CFH的这棵树）的根结点，再确定它的两棵子树，现在根结点是部分先序遍历的第一个元素，即C，根据C，在部分中序遍历中划分左右子树，即左子树为空，右子树包含FH，得到的图片如下：



可以看到这时B的左子树没有一个结点，所以可以认为已经确定好了，那么就只需要按照步骤一的方式对C的右子树进行同样的分析即可。

步骤四：得到一颗完整的二叉树。

根据上述步骤，完整地走一遍流程，便可以得到如下的一棵二叉树：



后序遍历+中序遍历

后序遍历+中序遍历的要义是：利用后序遍历确定根结点，再利用中序遍历划分左右子树

其实后序遍历+中序遍历唯一确定一棵二叉树的方法与先序遍历+中序遍历唯一确定一棵二叉树的方法本质上是一样的，唯一不同在于，利用后序遍历+中序遍历唯一确定一棵二叉树的方法在**每次确定根结点时**，需要对后序遍历的结果**从后往前看**，比如对于上述例子中的后序遍历：DGEBHFCA，第一步确定根结点时，需要取最后一个元素，因为后序遍历是最后才访问根结点，所以此时根结点的位置就是最后一个，而其他步骤就与先序遍历+中序遍历唯一确定一棵二叉树的方法是一样的了。