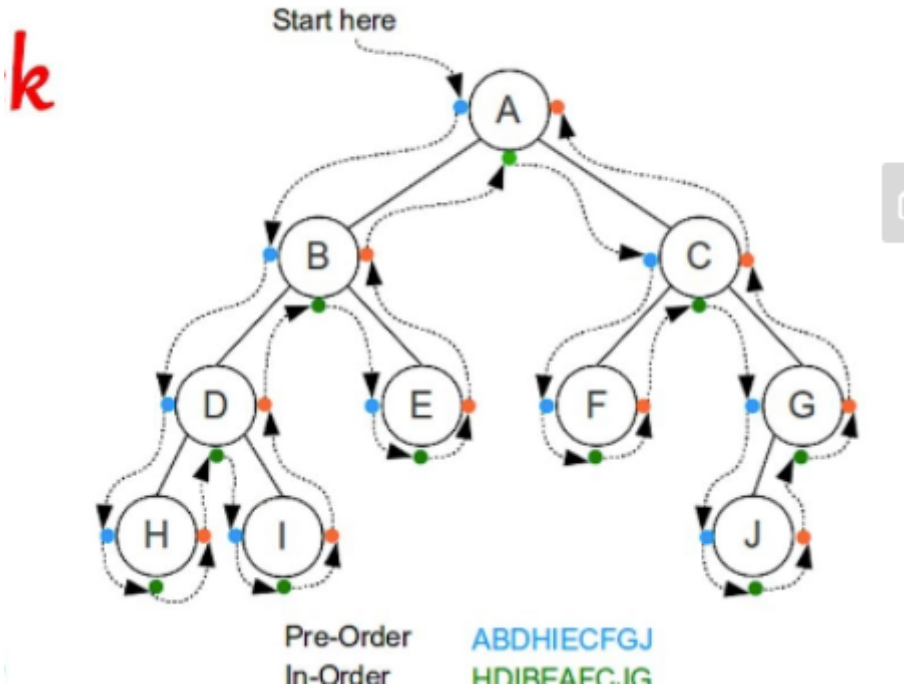
二叉树遍历

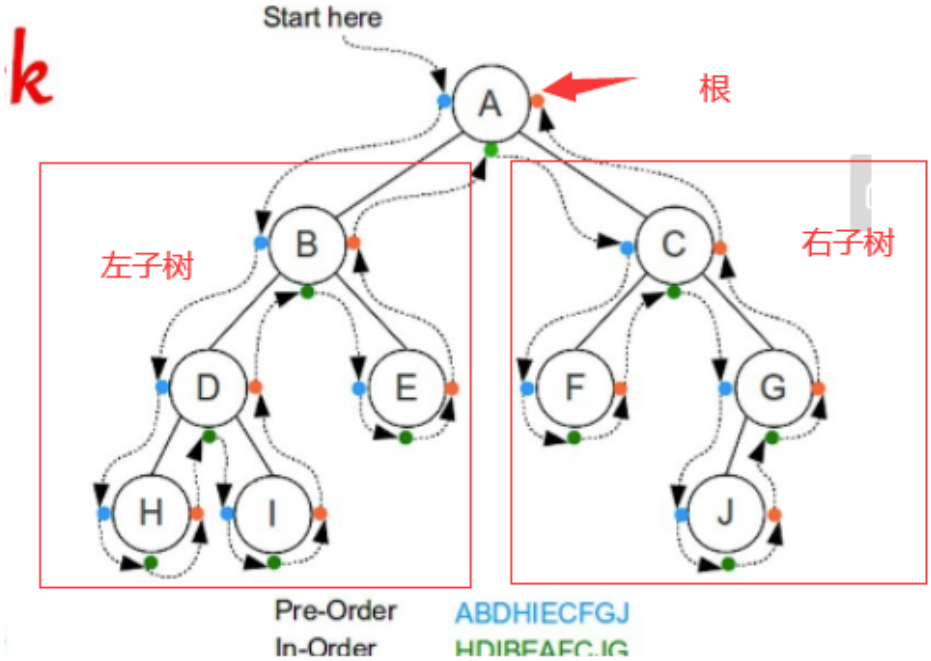
> 归思想实现



先序遍历：

先序遍历是指先遍历根节点，再遍历左子树，最后遍历右子树

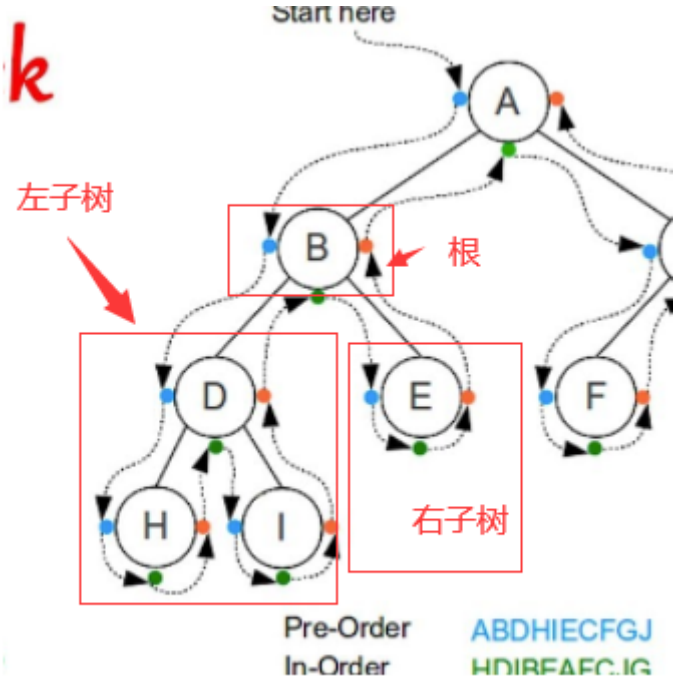
第一步：遍历A->A的左子树->A的右子树



最终遍历为A->[BDHIE]A的左子树->[CFGJ]A的右子树

第二步: 由于A的左子树没有遍历，所以要遍历A的左子树

我们把A的左子树看作是一个待遍历的二叉树，那么遍历图如下：



最终遍历为A->B->[DHI]B的左子树->E->[CFGJ]A的右子树

第三步：B的左子树没有遍历所以需要遍历B的左子树

我们把B的左子树看作是一个待遍历的二叉树，那么明显的按照先序次序遍历这个二叉树的话遍历次序是D->H->I

最终遍历为A->B->D->H->I->E->[CFGJ]A的右子树

至此A的左子树遍历结束，开始遍历A的右子树，遍历方式跟遍历A的左子树一致

需要理解的是，遍历根节点的时候，一定是可以取到根节点的具体值的，因为此时的根节点是一个单独的节点，而遍历左子树或者右子树的时候，左子树/右子树可能是一个二叉树结构，按照一个二叉树单独遍历。

二叉树遍历框架：

```
void traverse(TreeNode root) {
    // 前序遍历
    handle(root);    // 在此位置处理节点既是前序遍历
    traverse(root.left)
    // 中序遍历
    handle(root);  // 在此位置处理节点既是中序遍历
    traverse(root.right)
    // 后序遍历
handle(root);  // 在此位置处理节点既是后序遍历
}
```

如何理解递归遍历：

> traverse(root)：表示正在遍历一个根节点为root的树
>
> traverse(root.left)： 表示遍历根节点为root.left的树，而根节点为root.left的树既是root的左子树，这句话的意思就是遍历root的左子树，并且需要确定的是这样调用结束之后，root的左子树就已经遍历完成了。
>
> traverse(root.right)： 表示遍历根节点为root.righ的树，而根节点为root.righ的树既是root的右子树，这句话的意思就是遍历root的右子树，并且需要确定的是这样调用结束之后，root的右子树就已经遍历完成了。
>
> handle(root); 标识正在访问root节点，而这个root是traverse(root）中的root，root已经是确定的子树的根节点了，程序走到这里的时候一定是访问到了这个节点，如果访问到了这节点，那么该节点之前的所有操作就已经处理了。比如：中序遍历的时候，假设root为整个数的root，那么访问到这个节点的时候，traverse(root.left)肯定已经处理了，即root的左子树肯定已经处理过了。