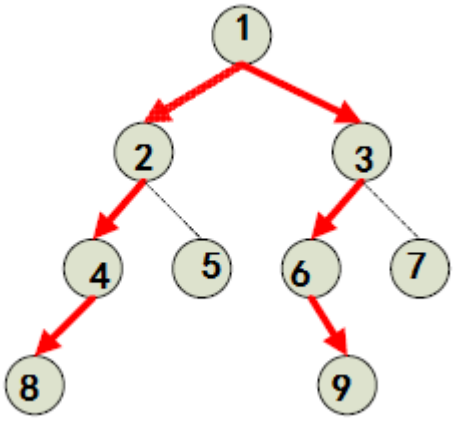
1. 二叉树习题：

(1) 指定二叉树某结点求与该结点距离最远的叶子结点。

(2) 对于有根树 T 的两个结点 u、v，最近公共祖先 LCA(T,u,v)表示一个结点 x，满足 x 是 u、v 的祖先且 x 的深度尽可能大。求一颗二叉树的最近公共祖先。

二叉树的测试案例如下图所示：



2. 根据 Huffman 算法原理 ，实现文本型文件压缩和解压缩。生成 Huffman 码表，能够压缩文件（见附件 Huffman_coding.txt ,），能够解压文件。为了便于性能比较，提供了一个大的文本压缩文件（见附件 Huffman_big.txt），请计算你的压缩时间和解压缩时间。为了平衡不同机器的差异，请将该时间除以 CPU 的频率（以 GHZ）为单位，如采用多核机器的多 CPU 同时运算，请另外给换算到单 CPU 的指标。

[选做]请将你的压缩效率与压缩时间与现有的压缩工作做一个比较，并考虑其中的差异所在。

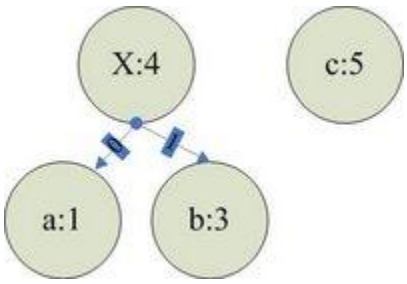
算法原理如下：

Huffman 算法是一种基于统计的压缩方法。它的本质就是对文本文件中的字符进行重新编码，对于使用频率越高的字符，其编码也越短。但是任何 2 个字符的编码，是不能出现向前包含的。也就是说字符 A 的编码的前段，不可能为字符 B 的编码。经过编码后的文本文件，主要包含 2 个部分：Huffman 码表部分和压缩内容部分。解压缩的时候，先把 Huffman 码表取出来，然后对压缩内容部分各个字符进行逐一解码，形成源文件。

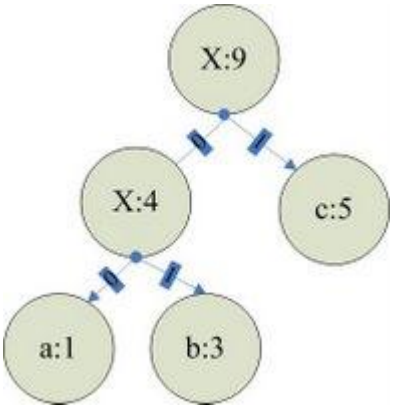
由此可见，使用 Huffman 算法的关键是形成 Huffman 码表。怎样才能生成一个"使用频率越高的字符，其编码也越短"的码表呢?这里就要用到 Huffman 树的数据结构。当把一棵 Huffman 树生成后，码表也就生成了。以下举例说明，假定我们的原始文本为"abcbbcccc"

1.扫描源文件，对字符频率进行统计。对于我们的样例，统计结果是:a:1　b:3 c:5 (按频率升序排列)

2.从上述队列中取出频率最低的 2 个节点，合并成一个频率为 2 节点频率之和的树枝节点 X，加入到原队列中，加入后，继续保持队列按频率升序排列.



3.重复步骤 2，直到队列中只有一个节点。



4.这样，我们就形成了一棵 Huffman 树。叶子节点为字符，从树根节点到叶子节点的路径即为该字符的 Huffman 编码。从一个节点导航到其左孩子，该段路径为 0，导航到右孩子，该段路径为 1.所以，a 字符的编码就是 00,b 字符的编码为 01,c 字符的编码为 1，符合"使用频率越高的字符，编码越短"的要求。

5.Huffman 码表生成后，原文本"abcbbcccc"就变成了 0001101011111 的位串，按每个字符占用 2 个 byte 计算，大小由原来的 18 个字节(9*2),共 144 个 bit,变成了 13 个 bit,2 个字节。达到了压缩的目的。

解压缩过程:

解压缩也分成 2 部分进行，首先是根据压缩文件中的 Huffman 码表，在内存中生成一棵 Huffman 树，然后，根据 Huffman 树，对压缩内容进行解压缩。比如如果压缩内容为位串 0001101011111，那么从树根节点起，因为第一个 bit 为 0，先转向左子树，第二个 bit

为 0,再转向左子树,到达叶子 a,所以解码出来的第一个字符就是 a,每次解压一个字符，都从根节点起，根据 bit 流，向左或向右转，直到到达叶子节点，也就是解压出来的字符。一直重复此过程，直到所有的字符都被解压缩。