# 如何较为直观地打印二叉树

## 【题目】

二叉树可以用常规的三种遍历结果来描述其结构，但是不够直观，尤其是二叉树中有重复值的时候，仅通过三种遍历的结果来构造二叉树的真实结构更是难上加难，有时则根本不可能。给定一棵二叉树的头节点 head，已知二叉树节点值的类型为 32 位整型，请实现一个打印二叉树的函数，可以直观地展示树的形状，也便于画出真实的结构。

## 【难度】

尉 ★★☆☆

## 【解答】

这是一道较开放的题目，面试者不仅要设计出符合要求且不会产生歧义的打印方式，还要考虑实现难度，在面试时仅仅写出思路必然是不满足代码面试要求的。本书给出一种符合要求且代码量不大的实现，希望读者也能实现并优化自己的设计。具体过程如下：

1．设计打印的样式。实现者首先应该解决的问题是用什么样的方式来无歧义地打印二叉树。比如，二叉树如图 3-4 所示。

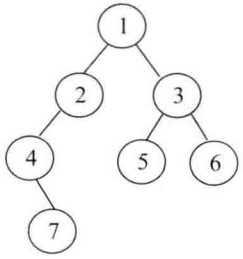


图 3-4

对如图 3-4 所示的二叉树，本书设计的打印样式如图 3-5 所示。

```
                v6v
        v3v
                ^5^
H1H
        ^2^
                        v7v
                ^4^
```

图 3-5

下面解释一下如何看打印的结果。首先，二叉树大概的样子是把打印结果顺时针旋转 90°，读者可以把图 3-4 的打印结果（也就是图 3-5 顺时针旋转 90°之后）做一下对比，两幅图是存在明显对应关系的；接下来，怎么清晰地确定任何一个节点的父节点呢？如果一个节点打印结果的前缀与后缀都有“H”(比如图 3-5 中的“H1H”)，说明这个节点是头节点，当然就不存在父节点。如果一个节点打印结果的前缀与后缀都有“v”，表示父节点在该节点所在列的前一列，在该节点所在行的下方，并且是离该节点最近的节点。比如图 3-5 中的“v3v”、“v6v”和“v7v”，父节点分别为“H1H”、“v3v”和“^4^”。如果一个节点打印结果的前缀与后缀都有“^”，表示父节点在该节点所在列的前一列，在该节点所在行的上方，并且是离该节点最近的节点。比如，图 3-5 中的“^5^”、“^2^”和“^4^”，父节点分别为“v3v”、“H1H”和“^2^”。

2．一个需要重点考虑的问题——规定节点打印时占用的统一长度。我们必须规定一个节点在打印时到底占多长。试想一下，如果有些节点的值本身的长度很短，比如“1”、“2”等，而有些节点的值本身的长度很长，比如“43323232”、“78787237”等，那么如果不规定一个统一的长度，在打印一个长短值交替的二叉树时必然会出现格式对不齐的问题，进而产生歧义。在 Java 中，整型值占用长度最长的值是 Integer.MIN_VALUE（即-2147483648），占用的长度为 11，加上前缀和后缀(“H”、“v”或“^”)之后占用长度为 13。为了在打印之后更好地区分，再把前面加上两个空格，后面加上两个空格，总共占用长度为 17。也就是说，长度为 17 的空间必然可以放下任何一个 32 位整数，同时样式还不错。至此，我们约定，打印每一个节点的时候，必须让每一个节点在打印时占用长度都为 17，如果不足，前后都用空格补齐。比如节点值为 8，假设这个节点加上“v”作为前后缀，那么实质内容为“v8v”，长度才为 3，在打印时在“v8v”的前面补 7 个空格，后面也补 7 个空格，让总

长度为 17。再如节点值为 66，假设这个节点加上“v”作为前后缀，那么实质内容为“v66v”，长度才为 4，在打印时在“v66v”的前面补 6 个空格，后面补 7 个空格，让总长度为 17。总之，如果长度不足，前后贴上几乎数量相等的空格来补齐。

3．确定了打印的样式，规定了占用长度的标准，最后来解释具体的实现。打印的整体过程结合了二叉树先右子树、再根节点、最后左子树的递归遍历过程。如果递归到一个节点，首先遍历它的右子树。右子树遍历结束后，回到这个节点。如果这个节点所在层为 l，那么先打印 l×17 个空格（不换行），然后开始制作该节点的打印内容，这个内容当然包括节点的值，以及确定的前后缀字符。如果该节点是其父节点的右孩子，前后缀为“v”，如果是其父节点的左孩子，前后缀为“^”，如果是头节点，前后缀为“H”。最后在前后分别贴上数量几乎一致的空格，占用长度为 17 的打印内容就制作完了，打印这个内容后换行。最后进行左子树的遍历过程。

直观地打印二叉树的所有过程请参看如下代码中的 printTree 方法。

```
public class Node {
        public int value;
        public Node left;
        public Node right;

        public Node(int data) {
                this.value = data;
        }
}

public void printTree(Node head) {
        System.out.println("Binary Tree:");
        printInOrder(head, 0, "H", 17);
        System.out.println();
}

public void printInOrder(Node head, int height, String to, int len) {
        if (head == null) {
                return;
        }
        printInOrder(head.right, height + 1, "v", len);
        String val = to + head.value + to;
        int lenM = val.length();
        int lenL = (len - lenM) / 2;
        int lenR = len - lenM - lenL;
        val = getSpace(lenL) + val + getSpace(lenR);
        System.out.println(getSpace(height * len) + val);
        printInOrder(head.left, height + 1, "^", len);
}

public String getSpace(int num) {
```

```
        String space = " ";
        StringBuffer buf = new StringBuffer("");
        for (int i = 0; i < num; i++) {
                buf.append(space);
        }
        return buf.toString();
    }
```

## 【扩展】

有关功能设计的面试题，其实最难的部分并不是设计，而是在设计的优良性和实现的复杂程度之间找到一个平衡性最好的设计方案。在满足功能要求的同时，也要保证在面试场上能够完成大致的代码实现，同时对边界条件的梳理能力和代码逻辑的实现能力也是一大挑战。读者可以看到本书提供的方法在完成功能的同时其代码很少，也请读者设计自己的方案并实现它。

# 二叉树的序列化和反序列化

## 【题目】

二叉树被记录成文件的过程叫作二叉树的序列化，通过文件内容重建原来二叉树的过程叫作二叉树的反序列化。给定一棵二叉树的头节点 head，并已知二叉树节点值的类型为 32 位整型。请设计一种二叉树序列化和反序列化的方案，并用代码实现。

## 【难度】

士 ★☆☆☆

## 【解答】

本书提供两套序列化和反序列化的实现，供读者参考。

方法一：通过先序遍历实现序列化和反序列化。

先介绍先序遍历下的序列化过程，首先假设序列化的结果字符串为 str，初始时 str=""。先序遍历二叉树，如果遇到 null 节点，就在 str 的末尾加上“#!”，“#”表示这个节点为空，节点值不存在，“!”表示一个值的结束；如果遇到不为空的节点，假设节点值为 3，就在 str 的末尾加上“3!”。比如图 3-6 所示的二叉树。

根据上文的描述，先序遍历序列化，最后的结果字符串 str 为：12!3!#!#!#!。