# 第9章 应用二叉树的基本概念编程

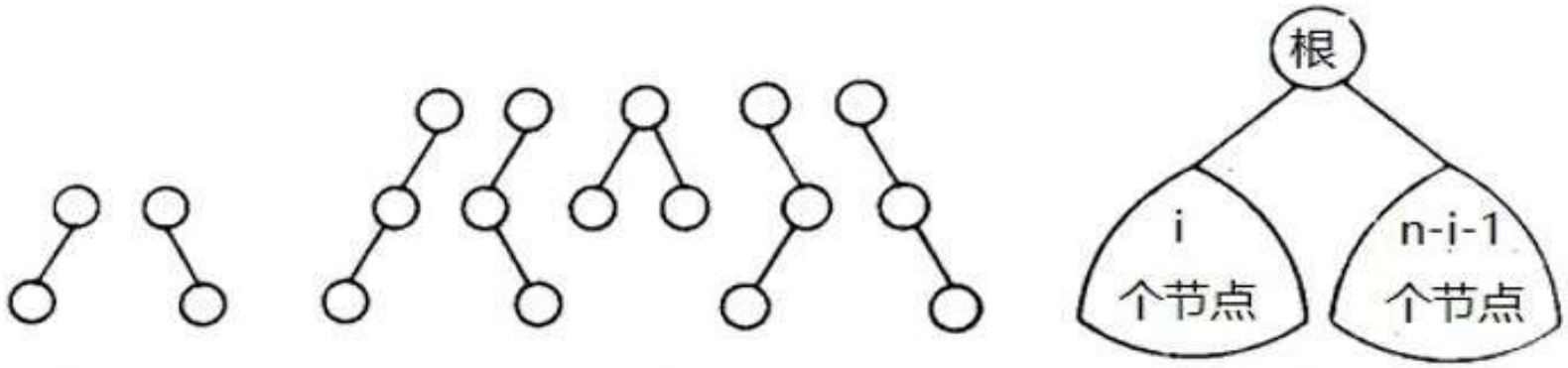
吴永辉
ICPC Asia Programming Contest 1st Training Committee – Chair
yhwu@fudan.edu.cn
WeChat：13817360465

- 9.1 普通有序树转化为二叉树的实验范例
- 9.2 应用典型二叉树的实验范例（**）
- 9.3 计算二叉树路径的实验范例（*）
- 9.4 通过遍历确定二叉树结构的实验范例（*）

# 9.2 应用典型二叉树的实验范例

- 定义**9.2.1**（二叉树， **Binary Tree**）二叉树是每个节点最多有两棵子树（左子树和右子树）的树结构。

设具有 $n$ 个节点、互不相似的二叉树的数目为 $b_n$。则 $b_0=1$ 的二叉树为空树；$b_1=1$ 的二叉树是只有一个根节点的树；$b_2=2$ 和 $b_3=5$，二叉树的形态分别如图 9.2-1 (a)和图 9.2-1 (b)所示。



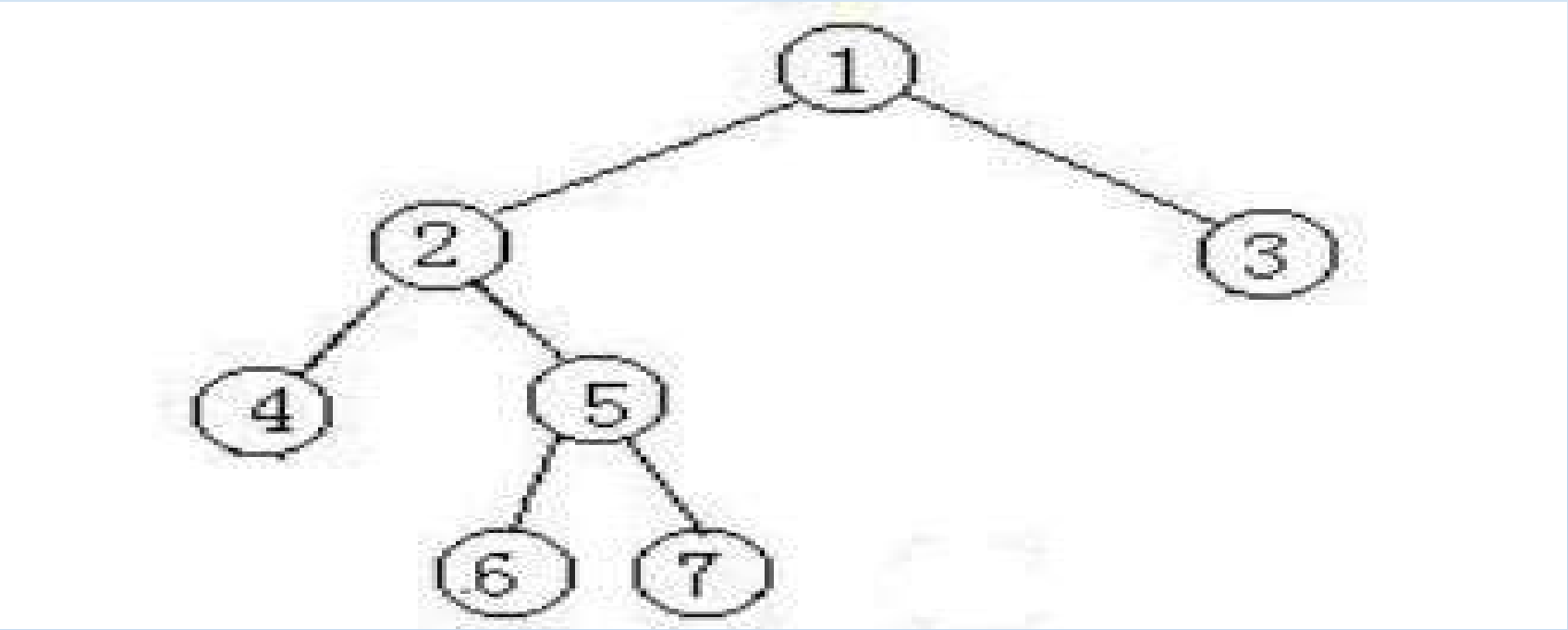

(a) n=2　　(b) n=3　　(c) 一般情形 n>1

图 9.2-1

当 $n>3$ 时，二叉树可以看成是由一个根节点、一棵具有 $i$ 个节点的左子树和一棵具有 $n-i-1$ 个节点的右子树组成，如图 9.2-1 (c)所示，其中 $0\le i\le n-1$。根据加法定理和乘法定理，$b_n=\begin{cases}1 & n=0\\ \sum_{i=0}^{n-1} b_i b_{n-i-1} & n\ge 1\end{cases}$。

对于二叉树，可以推导叶节点与节点总数的关系。设 $n$ 是二叉树的节点总数，$n_0$ 是孩子为 0 的节点数（即叶子节点数），$n_1$ 是孩子为 1 的节点数，$n_2$ 是孩子为 2 的节点数，则得公式①$n=n_0+n_1+n_2$；又因为一个孩子为 2 的节点会有 2 个子节点，一个孩子为 1 的节点会有 1 个子节点，除根节点外其他节点都有父节点，则得公式②$n=1+n_1+2*n_2$；由①、②两式把 $n_2$ 消去，得公式③：$n=2*n_0+n_1-1$。

- 定义**9.2.2**（满二叉树，**Full Binary Tree**）一棵二叉树的结点要么是叶子结点，要么它有两个子结点，这样的二叉树就是满二叉树。



# 9.2.1 Expressions

- 试题来源：**Ulm Local Contest 2007**
- 在线测试：**POJ 3367，UVA 11234，HDOJ 1805**

- 算术表达式通常被写为运算符在两个操作数之间（这被称为中缀表示法）。例如，(*x*+*y*)*(*z*–*w*)是用中缀表示法表示的算术表达式。然而，如果表达式是用后缀表示法（也称为逆波兰表示法）写的，则编写程序计算表达式更容易。在后缀表示法中，运算符写在其两个操作数后面，这两个操作数也可以是表达式。例如，“*x y+z w–**”是上面给出的算术表达式的后缀表示法。注意，在后缀表示法中不需要括号。

- 现在，假设我们使用队列而不是栈。队列也有**push**和**pop**操作，但它们的含义不同：
- **push**：在队列的末尾插入一个数字。
- **pop**：将队列首部的数字从队列中取出。
- 您是否可以重写给出的表达式，使得使用队列的算法的结果与使用栈的算法计算该表达式的结果相同？

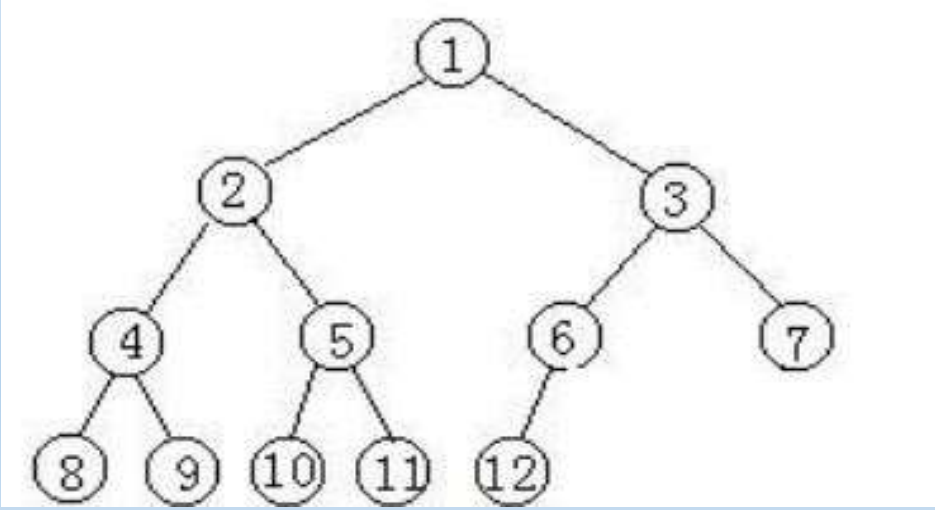
- 输入
- 输入的第一行给出一个数字$T$（$T \leq 200$）。后面的$T$行每行给出一个后缀表示法的表达式。算术运算符用大写字母表示，数字用小写字母表示。本题设定每个表达式的长度小于10000个字符。
- 输出
- 对于每个给出的表达式，当算法是用队列而不是用栈时，输出具有相等结果的表达式。为了使解唯一，运算符不允许可结合的或可交换的。

# 试题解析

- 一个用中缀表示法表示的算术表达式，可以表示为一棵满二叉树，运算符为内结点，操作数为叶结点。中序遍历这棵满二叉树，即可得到用中缀表示法表示的算术表达式。本题输入给出一个后缀表示法的算术表达式，就是算术表达式所对应的满二叉树的后序遍历；要求依由下至上、从右往左的顺序遍历这棵二叉树。

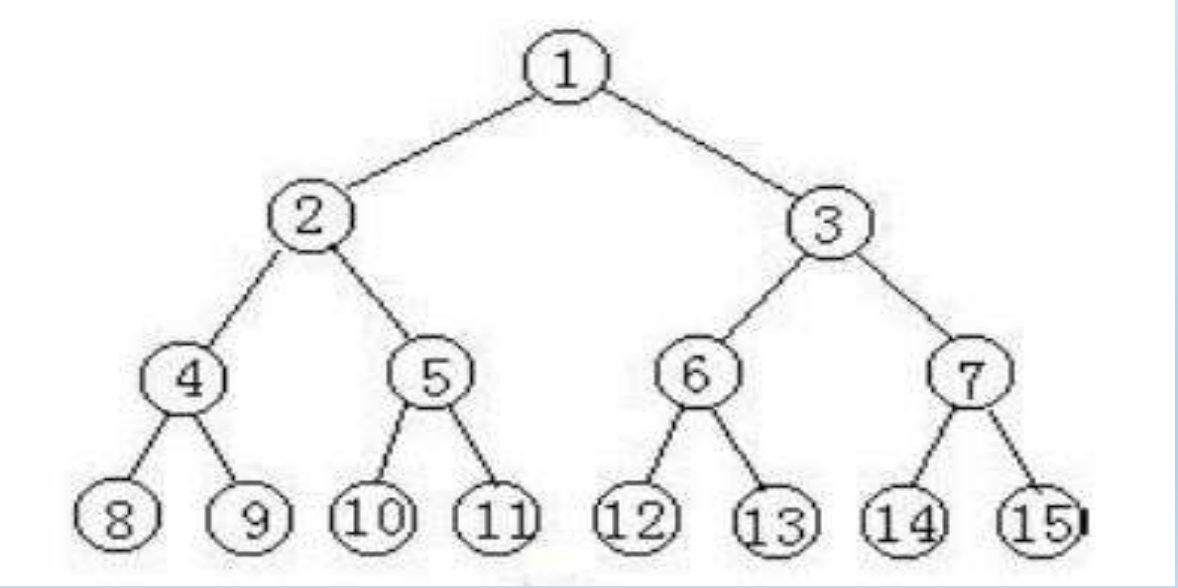
- 我们可按照下述方法“重写”这个遍历串：
- 【1】构建后序遍历对应的满二叉树
- 这个计算过程要使用栈：顺序分析后序串的每个字符：若当前字符为操作数，则赋值给当前结点，结点序号入栈；否则栈顶的两个结点序号出栈，分别作为当前结点的左右指针，当前字符（运算符）赋值给当前结点。处理完当前字符后，结点序号+1。
- 【2】从根出发，宽度优先搜索（即自上而下、由左而右顺序）满二叉树
- 这个计算过程则要使用队列：根序号进入队列；然后反复取出队首的结点序号，将结点的数据加入结果串；若结点存在左右指针的话，则将其送入队列。这个过程一直进行至队列空为止。
- 【3】对结果串进行反转操作，形成由下至上、从右往左的遍历顺序。
- 其中步骤⑵⑶，就是使用队列重写“后序遍历”。

- 定义**9.2.2**（完全二叉树，**Complete Binary Tree**）若一棵二叉树的深度为*h*，除第 *h* 层外，其它各层 (1～*h*-1) 的结点数都达到最大个数，第 *h* 层所有的结点都连续集中在最左边，则这样的二叉树被称为完全二叉树。



- 由于完全二叉树中度为1的节点数只有两种可能：0或1，由公式③：$n= 2*n_0+n_1-1$，$n_0=n/2$或$n_0=(n+1)/2$。所以，当$n$为奇数时（即$n_1=0$），$n/2$向上取整为$n_0$；当$n$为偶数时（$n_1=1$），$n_0$为$n/2$的整商。因此，可以根据完全二叉树的节点总数计算出叶节点数。

- 定义**9.2.3**（完美二叉树，**Perfect Binary Tree**）在一棵二叉树中，如果所有分支结点都存在左子树和右子树，并且所有叶子都在同一层上，这样的二叉树称为完美二叉树。



完美二叉树具有如下性质。

**定理 9.2.1** 一棵层数为 $k$ 的完美二叉树，总节点数为 $2^k-1$。

完美二叉树的总节点数一定是奇数个。例如图 9.2-4 是一棵层数 4 的完美二叉树，节点数为 $2^4-1=15$。

**定理 9.2.2** 一棵完美二叉树的第 $i$ 层上的节点数为 $2^{i-1}$。

例如，图 9.2-4 中的完美二叉树，第 2 层的节点数为 $2^{2-1}=2$。一个层数为 $k$ 的完美二叉树的叶子节点个数（位于最后一层）为 $2^{k-1}$。例如图 9.2-4 中的完美二叉树，第 4 层的叶节点数为 $2^{4-1}=8$。

# 9.2.2 Subtrees

- 试题来源：**BestCoder Round #61 (div.2)**
- 在线测试：**HDOJ 5524**

- 有$N$个节点的完全二叉树，问有多少种子树所包含的节点数量不同。
- **输入**
- 输入有多个测试用例，不超过1000个。
- 每个测试用例一行，给出一个整数$N$（$1 \le N \le 10^{18}$）。
- **输出**
- 对于每个测试用例，输出一行，给出不同节点数的子树有多少种。

# 试题解析

- 一棵完全二叉树有可能是完美二叉树；否则完全二叉树的子树有一棵是完美二叉树，另一棵是完全二叉树。
- 一棵完美二叉树的不同结点的子树的个数就是它的层数，因为满二叉树的子树依然是满二叉树。
- 而对于一棵完全二叉树，但不是完美二叉树，其不同节点数的子树的数目是最大的完美二叉子树的深度+最大的非完美二叉子树对深度的贡献+1（该树本身）。

根据给定节点数量，可轻易得知左右子树的形态，然后递归求解。由于 $n$ 的数量太大，要用 long long。具体方法：

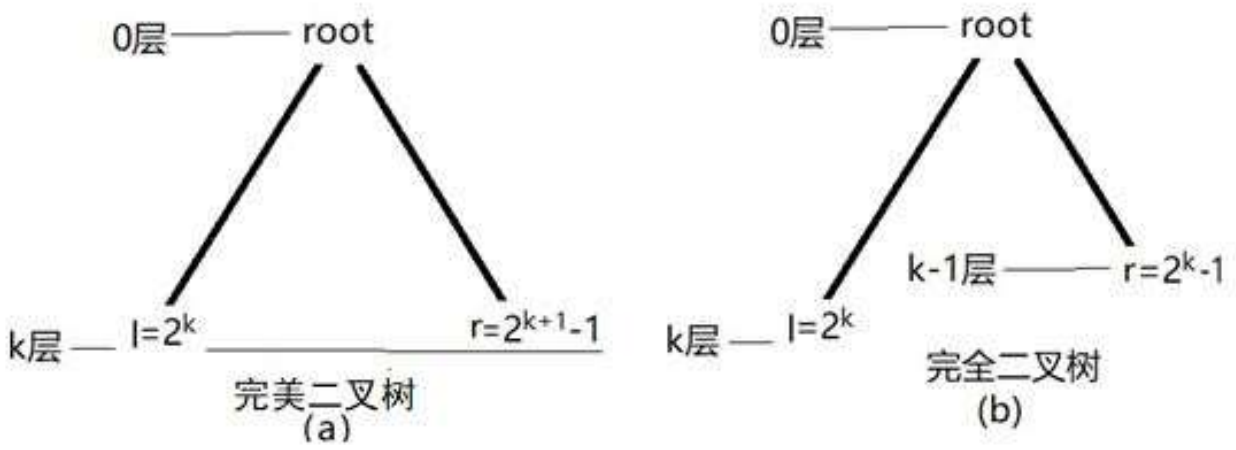



图 9.2-5

- 们通过递归过程*find*(*x*)计算完全二叉子树的棵数*ans*和最大完美二叉子树的深度*maxn*：
- 从子根节点*x*出发，向左方向扩展一条深度为*k*的左分枝，末端节点*l*的编号为$2^k$；向右方向扩展一条右分枝，末端节点*r*的编号为$2^k$-1或者$2^{k+1}$-1。
- 如果*l*≤*r*（*r*=*l*时为单根完美二叉树），则说明以*root*为根的子树为完美二叉树（如图9.2-5（a）），根据其深度*k*调整目前为止所有满二叉子树的最大深度*maxn*=*max*{*maxn*，*k*}；
- 如果*l*>*r*，则说明以*x*为根的子树为完全二叉树（图9.2-5（b））。分别递归*find*(2**x*)和*find*(2**x*+1)，完全二叉子树的棵数*ans*+1。
- 递归结束后，*ans*+*maxn*+1即为含不同节点数的子树种数(由于深度是从0开始计算，所以最后结果+1)。

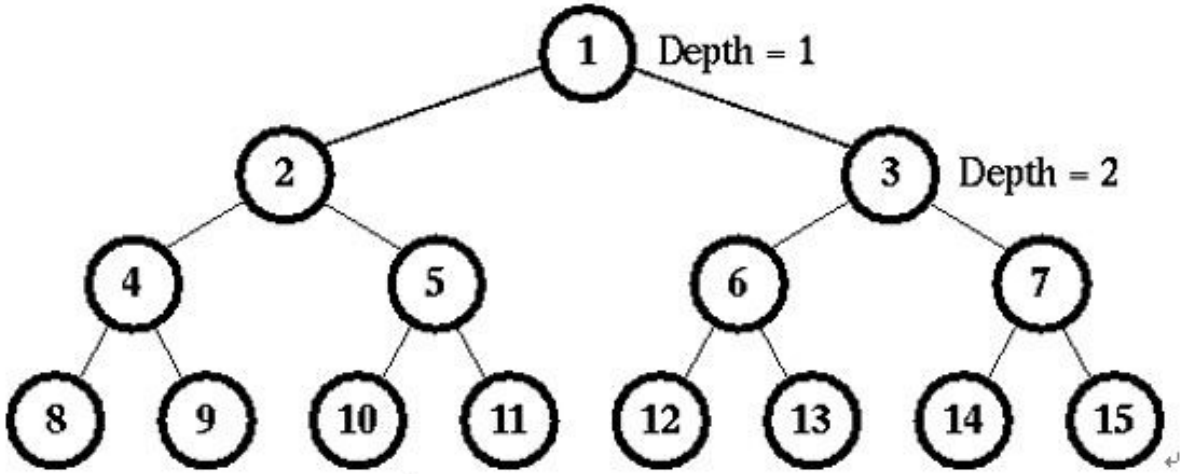
# 9.3 计算二叉树路径的实验范例

# 9.3.2 Dropping Balls

- 试题来源：**1998 Taiwan Collegiate Programming Contest, group A**
- 在线测试：**UVA 679**

- 从完美二叉树（Fully Binary Tree，FBT）的根一个接一个地向下落*K*个球。每次被落下的球到一个内节点后，球还会继续向下落下去，要么沿着左子树的路径，要么沿着右子树的路径，直到它落到了FBT的一个叶节点上。为了确定球的移动方向，在每个内节点都设置了一个标志，该标志有两个值，分别为false或true。最初，所有的标志都是false。当球落到一个内节点时，如果该节点上的标志当前值为false，则首先切换该标志的值，即从false切换到true，然后球向该节点的左子树继续向下落；否则，切换此标志的值，即从true切换到false，然后球向节点的右子树继续向下落。而且，FBT的所有节点都按顺序编号，先对深度为1的节点从1开始编号，然后对深度为2的节点开始编号，依此类推；任何深度上的节点都是从左到右编号。

例如，图 9.3-1 表示具有节点号 1, 2, 3, ……, 15 的最大深度为 4 的 FBT。初始时，所有节点的标志都被设置为 false，因此第一个落下的球将在节点 1、节点 2 和节点 4 处切换标志值，最终在节点 8 停止。第二个落下的球将在节点 1、节点 3 和节点 6 处切换标志值，并在节点 12 停止。很明显，第三个落下的球将在节点 1、节点 2 和节点 5 处切换标志值，然后在节点 10 停止。




最大深度为 4 并且顺序编号的节点的 FBT 的例子

- 本题给出一些测试用例，每个测试给出两个值。第一个值是*D*，FBT的最大深度；第二个值是*I*，第*I*个球被落下。本题设定*I*的值不会超过给出的完美二叉树的叶节点总数。
- 请您编写一个程序，计算每个测试用例的球最终停在的叶节点的位置*P*。
- 对于每个测试用例，两个参数*D*和*I*的范围如下：$2 \le D \le 20$，$1 \le I \le 524288$。

- 输入
- 输入有$l$+2行。
- 第1行，给出$l$，测试用例数。
- 第2 行，给出$D_1$ $I_1$；表示第1个测试用例，用一个空格分隔的两个十进制数。
- … …
- 第$k$+1行，给出$D_k$ $I_k$；表示第$k$个测试用例；
- 第$l$+1行，给出$D_l$ $I_l$；表示第$l$个测试用例；
- 第$l$+2行，给出-1，表示输入结束。

- 输出
- 输出*l*行。
- 第1行，第1个测试用例，球最终停在的叶节点的位置*P*；
- …… 
- 第*k*行，第*k*个测试用例，球最终停在的叶节点的位置*P*；
- …… 
- 第*l*行，第*l*个测试用例，球最终停在的叶节点的位置*P*。

# 试题解析

- 首先，要说明一点，对于满二叉树（Fully Binary Tree），国内外定义不同，国内的教材中，满二叉树就是完美二叉树。而本题来自台湾。为了使得本书定义上保持一致，“Fully Binary Tree”被译为完美二叉树。
- 简述题意：给出一棵深度为*D*的完美二叉树和*I*个球，初始所有的节点标志都是false，如果节点标志是false，则球向左走，否则球向右走，小球接触节点后，该节点状态被置反。给定完美二叉树的深度*D*和小球编号*I*，问第*I*个小球最终会落到哪个叶子节点。

- 如果直接模拟$I$个小球的下落过程，结果会超时，因为$I$的上限为$2^{19}$，每个小球最多下落19层，每组测试总下落次数可高达$2^{19}*19$次，而测试数据最多有1000组。
- 对深度为$D$的完美二叉树，按自上而下、由左而右顺序依次给节点编号1到$2^D-1$，即编号为$k$的内节点，其左儿子和右儿子分别编号为$2k$和$2k+1$。

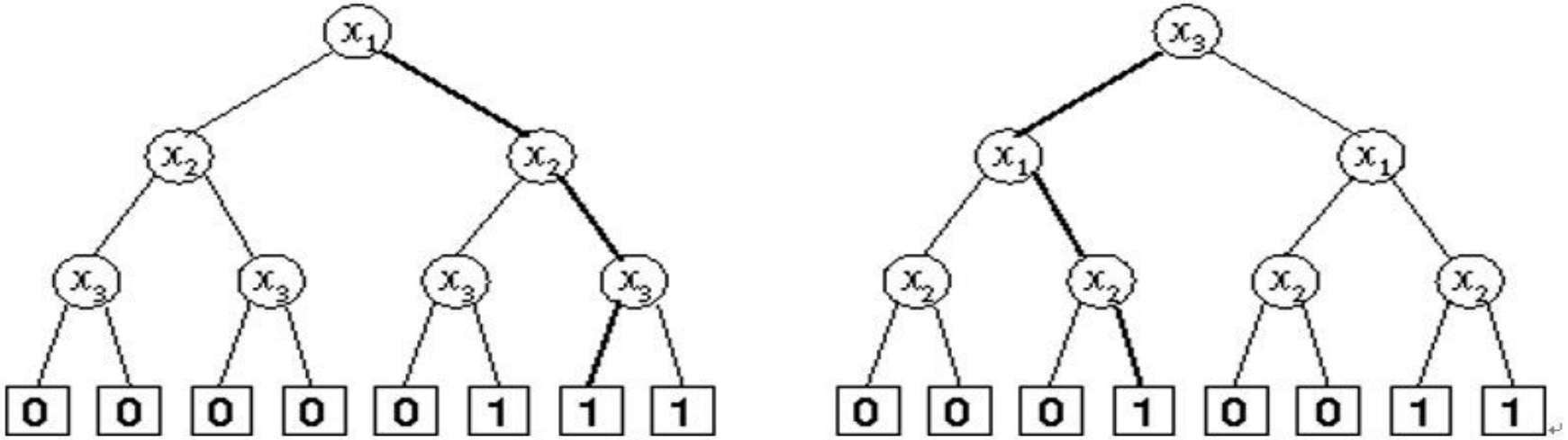
- 经过分析和模拟后，可以发现，对于当前测试用例给出的*I*，如果*I*是奇数，则小球在根节点时，落向左子树，并且是第(*I*+1)/2个落向左子树的小球；如果*I*是偶数，则小球在根节点时，落向右子树，并且是第*I*/2个落向右子树的小球。因此，小球最后落入的节点的编号与第*I*个小球，以及*I*的奇偶性有关。
- 例如，对于第7个小球，因为7是奇数，所以小球落向左子树，到达节点2；往下相当于第(7+1)/2=4个小球，落向右子树，到达节点5；再往下，则相当于第2个小球，落向右子树，到达节点11。
- 所以，本题直接模拟小球的下落过程即可得出答案：直接对于第*I*个小球仅进行一趟次数为*D*的循环，每次循环计算小球在当前层次落向左右子树以及落入下一层的节点编号，最终求出第*I*个小球停在*D*层的叶节点编号。

# 9.3.3 S-Trees

- 试题来源：**ACM Mid-Central European Regional Contest 1999**
- 在线测试：**POJ 1105，UVA 712**

- 在变量集$X_n=\{x_1, x_2, \ldots, x_n\}$上的一棵奇怪树（Strange Tree (S-tree)，称为S-树）是一棵二叉树，表示布尔函数$f$:{0,1}->{0,1}。S-树的每条路径都从根节点开始，由$n$+1个节点组成。每个S-树的节点都有一个深度，是它自身和根节点之间的节点数量（根节点的深度为0）。深度小于$n$的节点称为非终端节点。所有非终端节点都有两个子节点：右子节点和左子节点。每个非终端节点用变量集$X_n$中的某一变量$x_i$标记。所有深度相同的非终端节点都用相同的变量标记，不同深度的非终端节点用不同的变量标记。因此，相应于根，有一个唯一的变量$x_{i1}$；相应于深度为1的节点有唯一的变量$x_{i2}$；以此类推。变量序列$x_{i1}, x_{i2}, \ldots, x_{in}$被称为变量排序。深度为$n$的节点称为终端节点，它们没有孩子，被标记为0或1。注意，变量顺序和终端节点上0和1的分布足以完全地描述S-树。

如前所述，每个 S-树表示一个布尔函数 $f$。如果有一个 S-树和变量 $x_1, x_2, \ldots, x_n$ 的值，那么很容易计算出 $f(x_1, x_2, \ldots, x_n)$是什么：从根开始。现在重复如下过程：如果当前所在的节点被标记为变量 $x_i$，则取决于变量的值是 1 还是 0，分别走向其右孩子或左孩子。一旦到达终端节点，它的标签将给出函数的值。




函数 $x_1$ and ($x_2$ or $x_3$)的 S-树

图 9.3-3

在图 9.3-3 中，给出了两个表示相同布尔函数 $f(x_1, x_2, x_3)=x_1$ and ($x_2$ or $x_3$)的 S-树。对于左边的树，变量顺序是 $x_1, x_2, x_3$，对右边的树，变量顺序是 $x_3, x_1, x_2$。

- 以变量值赋值（Variable Values Assignment (VVA)）给出变量$x_1$, $x_2$, ..., $x_n$的值($x_1 = b_1, x_2 = b_2, ..., x_n = b_n$)，其中$b_1, b_2, ..., b_n$在{0, 1}中取值。例如( $x_1 = 1, x_2 = 1, x_3 = 0$)是对$n = 3$的一个有效的VVA，对于上述样例函数，结果值为$f(1, 1, 0) = 1$ and (1 or 0) = 1，相应的路径在图中以粗体表示。
- 请您编写一个程序，给出一棵S-树和一些VVA，按上述方法计算$f(x_1, x_2, ..., x_n)$。

- 输入
- 输入给出若干S-树，以及相关的VVA的测试用例。每个测试用例的第一行给出一个整数$n$，$1 \le n \le 7$，即S-树的深度。接下来的一行，给出S-树的变量顺序。这一行的格式是$x_{i1}\ x_{i2}\ \ldots\ x_{in}$（$n$个完全不同的、用空格分隔的字符串，）。因此，对于$n=3$，变量顺序$x_3, x_1, x_2$，则这一行如下所示：
- x3 x1 x2
- 接下来的一行给出了0和1在终端节点上的分布，给出$2^n$个字符（每个字符是0或1），后面跟着换行符；这些字符按照它们在S-树中出现的顺序给出，第一个字符对应于S-树的最左边的终端节点，最后一个字符对应于其最右边的终端节点。
- 再接下来的一行给出一个整数$m$，即VVA的数量；后面给出$m$行VVA。在$m$行中的每一行给出$n$个字符（每个字符是0或1），后跟一个换行符。不管S树的变量顺序如何，第一个字符是$x_1$的值，第二个字符是$x_2$的值，依此类推。例如，一行中给出110相应于 VVA ($x_1 = 1, x_2 = 1, x_3 = 0$)。
- 输入以n=0开头的测试用例终止。程序不用处理该测试用例。

- 输出
- 对于每个S-树，输出一行“S-Tree #*j*:”，其中*j*是S-树的编号。然后输出一行，对每个给定*m* VVA，输出$f(x_1, x_2, ..., x_n)$，其中*f*是S-树定义的函数。
- 在每个测试用例之后输出一个空行。

# 试题解析

- 本题给出一棵完美二叉树，每个非叶节点层用一个变量$x_i$表示。有$m$条从根节点开始的路线，0表示向左孩子走，1表示向右孩子走，问$m$条路上最终节点的值。
- 把询问的序列可以看作二进制数，例如000看成十进制的0，010看成十进制的2。假设我们把最后的叶子节点存在字符数组$a$[]中，则可以发现，分别把询问序列中的每个二进制数转换成十进制数$s$，则$a[s]$对应当前询问要访问的叶节点值。

# 9.4 通过遍历确定二叉树结构的实验范例

# 9.4.2 Tree

- 试题来源：**ACM Central American Regionals 1997**
- 在线测试：**UVA 548**

- 给出一棵二叉树，请您计算二叉树中这样的叶节点的值：该节点是从二叉树的根到所有的叶节点中具有的最小路径值的终端节点。一条路径的值是该路径上节点的值之和。

- 输入
- 输入给出一棵二叉树的描述：该二叉树的中序和后序遍历的序列。程序将从输入中读取两行（直到输入结束）。第一行给出树的中序遍历的值的序列，第二行给出树的后序遍历的值的序列。所有值都大于0且小于500。本题设定，任何二叉树没有超过25个节点或少于1个节点。
- 输出
- 对于每棵树的描述，请您输出具有最小路径值的叶节点的值。如果存在多条路径具有相同的最小值的情况，您可以选择任何合适的终端节点。

# 试题解析

- 简述题意：本题输入一个二叉树的中序和后序遍历的序列，输出二叉树的一个叶节点，要求该叶节点到根的路径值最小。

- 本题的解题算法：
- 步骤1：根据中序遍历和后序遍历构造对应二叉树。方法如下：
- 由后序遍历确定当前子树的根节点，再在中序遍历中找到子根位置，则中序遍历中根的左侧是其左子树的中序遍历，根的右侧是其右子树的中序遍历；若左子树的长度为*len*，则当前的后序遍历的前*len*个数据是左子树的后序遍历序列。右子树也是同理。
- 步骤2：对二叉树进行深度优先搜索*dfs*(*r*, *m*)，其中递归参数*r*为当前节点，*m*为根至*r*的路径值。显然，递归到叶节点*r*时，*m*保存的是*r*到根节点的路径值。
- 设记录最佳方案的全局变量*ans*和*pos*，*ans*存储到目前为止叶节点至根的最小路径值，*pos*存储这个叶结点。当递归至叶节点*r*时，如果*m*小于*ans*，则*ans*调整为*m*，*pos*调整为当前的叶节点*r*。显然，递归结束后的*ans*和*pos*即为本题要求的最佳方案。

# 9.4.3 Parliament

- 试题来源：**Quarterfinal, Central region of Russia, Rybinsk, October 17-18 2001**
- 在线测试：**Ural 1136**

- MMMM州选出了一个新的议会。在议会注册过程中，每位议员都会获得其唯一的正整数的议员证号。数字是随机给出的；在数字序列中，两个相邻的数字之间可以隔着几个数。议会中的椅子排列呈一个树状结构。当议员们进入礼堂时，他们按下列顺序就座。他们中的第一个人坐上主席的位子。接下来的每一位代表如果议员证号小于主席的证号，就向左走，否则就向右走；然后，坐上空位子，并宣布自己是子树的主席。如果子树主席的座位已被人占了，则坐上座位的算法以相同的方式继续：代表根据子树主席的证号向左或向右移动。

图 9.4-3 展示按以下议员证号 10、5、1、7、20、25、22、21、27 的顺序进入礼堂的议员的座位示例：

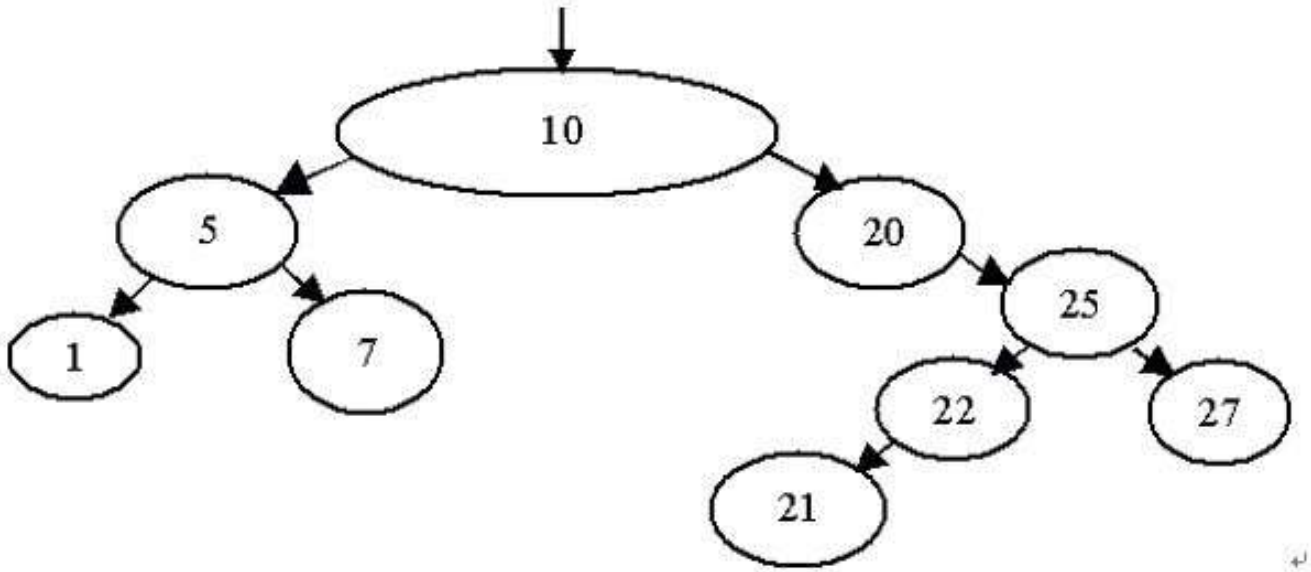



图 9.4-3

- 议会在第一次会议上决定以后不改变座次，也确定了议员发言的顺序。如果会议的次数是奇数，那么议员们的发言顺序如下：左子树、右子树和主席。如果一个子树有不止一个议员，那么他们的发言顺序也是一样的：子树的左子树、子树的右子树和子树主席。如果会议次数是偶数，发言顺序就不同：右子树、左子树和主席。对于给定的示例，奇数次会议的发言顺序为1、7、5、21、22、27、25、20、10；而偶数次会议的发言顺序为27、21、22、25、20、7、1、5、10。
- 给出奇数次会议的发言顺序，请您确定偶数次会议的发言顺序。

- 输入
- 输入的第一行给出*N*，表示议员总数。接下来的几行给出*N*个整数，表示按奇数次会议的发言顺序的议员证号码。
- 议员总数不超过3000人。议员证号码不超过65535。
- 输出
- 输出给出偶数次会议的发言顺序的议员证号码。

# 试题解析

- 由议员座位图可以看出，左子树上的键值都比根小，右子树的键值都比根大，所以这是一棵二叉搜索树。试题给出了这棵二叉搜索树的后序遍历（即奇数次会议的发言顺序），要求输出这棵二叉树的“右子树-左子树-根”的遍历（即偶数次会议的发言顺序）。
- 我们用数组$a$[]存储后序遍历，其中$a[i]$为奇数次会议中第$i$个发言的议员证号码。显然，数组最后一个元素$a[n]$即为根。则在数组$a$[]中第一个大于$a[n]$的元素$a[i]$（$1 \le i \le n-1$）即为根的右儿子，$a[i..n-1]$是右子树的后序遍历区间；而$a[1..i-1]$ 是后序遍历区间的左子树区间，其中$a[i-1]$为$a[n]$的左儿子。

- 以试题的样例为例，后序遍历的最后一个数10是树根的键值，后序遍历中第一个比10大的数是 21，则以21为首的后缀是右子树的后序遍历。而后序遍历中以21左邻的数字5为尾的前缀是根的左子树的后序遍历，而5就是10 的左儿子的键值，10的左邻数字20就是10 的右儿子的键值……；以此类推，1是5的左儿子，7是5的右儿子；对于10的右子树也一样。
- 本题可用递归函数求解，首先确定左、右子树的范围，然后输出右子树，再输出左子树，最后输出根。