$D(i)$为根的左子树的内部路径长。但是在原树中，所有这些节点都要加深一度。同样的结论对于右子树也成立。因此我们得到递推关系

$$D(N) = D(i) + D(N-i-1) + N - 1$$

如果所有子树的大小都等可能地出现，这对于二叉查找树是成立的(因为子树的大小只依赖于第一个插入到树中的元素的相对的秩(rank))，但对二叉树不成立，那么 $D(i)$和 $D(N-i-1)$的平均值都是$(1/N)\sum_{j=0}^{N-1} D(j)$。于是

$$D(N) = \frac{2}{N}\left[\sum_{j=0}^{N-1} D(j)\right] + N - 1$$

在第 7 章将遇到并求解这个递推式，得到的平均值为 $D(N) = O(N \log N)$。因此任意节点预期的深度为 $O(\log N)$。作为一个例子，图 4-26 所示随机生成的 500 个节点的树的节点期望深度为 9.98。

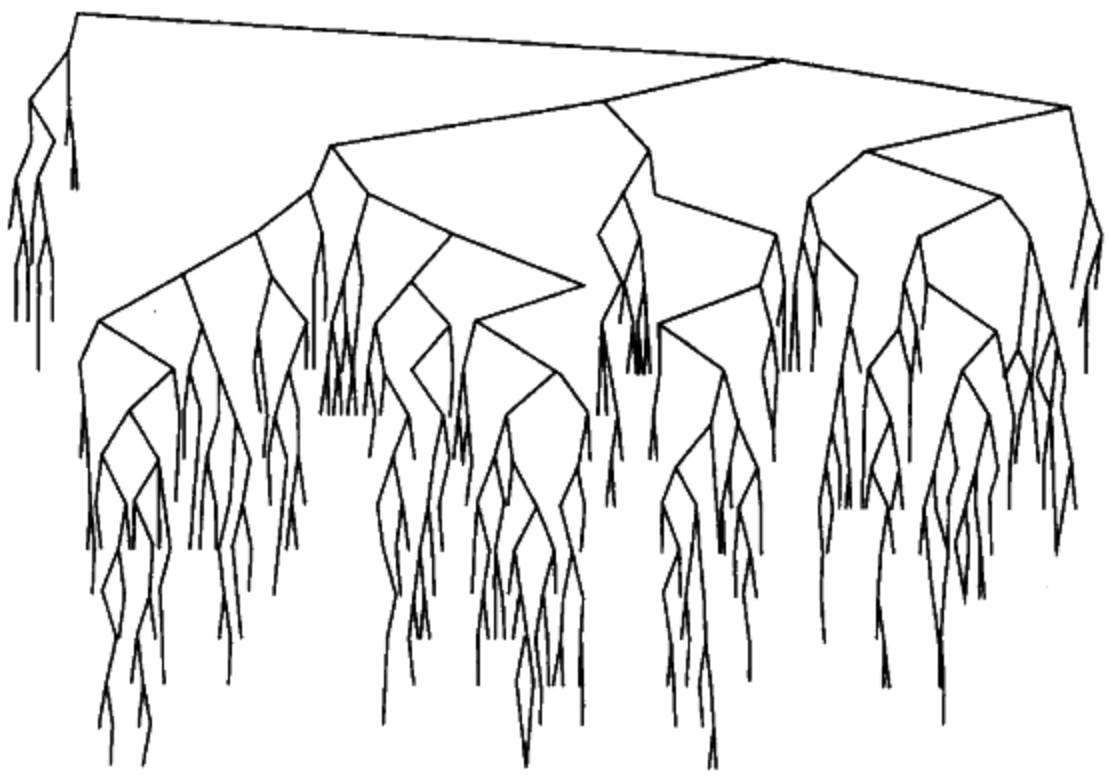

图 4-26　一棵随机生成的二叉查找树

由这个结果似乎可以立即看出上一节讨论的所有操作的平均运行时间是 $O(\log N)$，但这并不完全正确。原因在于删除操作，我们并不清楚是否所有的二叉查找树都是等可能出现的。特别是上面描述的删除算法有助于使得左子树比右子树深度深，因为我们总是用右子树的一个节点来代替删除的节点。这种方法的准确的效果仍然是未知的，但它似乎只是理论上的悬念。业已证明，如果我们交替插入和删除 $\Theta(N^2)$次，那么树的期望深度将是 $\Theta(\sqrt{N})$。在 25 万次随机 insert/remove 对操作后，图 4-26 中右沉的树看起来明显地不平衡(平均深度 = 12.51)，见图 4-27。

在删除操作中，我们可以通过随机选取右子树的最小元素或左子树的最大元素来代替被删除的元素以消除这种不平衡问题。这种做法明显消除了上述偏向并使树保持平衡，但是，没有人实际上证明过这一点。无论如何，这种现象似乎主要是理论上的问题，因为对于小的树上述效果根本不明显，甚至更奇怪。如果使用 $o(N^2)$对 insert/remove 操作，那么树似乎可以得到平衡！

上面的讨论主要是说明，决定“平均”意味着什么一般是极其困难的，可能需要一些假设，这些假设可能合理，也可能不合理。不过，在没有删除或是使用懒惰删除的情况下，我们可以断言：上述那些操作的平均运行时间都是 $O(\log N)$。除像上面讨论的一些个别情形外，这个结果与实际观察到的情形是非常一致的。