算法分析与设计: 第五次作业

PB20061372　朱云沁　Apr. 28, 2023

13.1-5 证明: 在一棵红黑树中, 从某结点 x 到其后代叶结点的所有简单路径中, 最长的一条至多是最短一条的 2 倍.

解: 给定任意两条从 x 到叶结点的简单路径 $\mathcal{P}_1, \mathcal{P}_2$, 设其长度分别为 l_1, l_2, 且 $l_1 \geq l_2$.

- 由红黑树性质 5 知, $\mathcal{P}_1$, $\mathcal{P}_2$ 包含相同数目的黑色结点;
- 由红黑树性质 3, 4 知, $\mathcal{P}_1$ 除黑色叶结点外, 剩余 $l_1 - 1$ 个结点中不存在相邻的红色结点. 进而, 由容斥原理, $\mathcal{P}_1$ 中红色结点的数目至多为 $\lfloor \frac{l_1}{2} \rfloor$.

因此, $\mathcal{P}_1$, $\mathcal{P}_2$ 中黑色结点数目至少为 $\lceil \frac{l_1}{2} \rceil$. 故 $l_2 \geq \lceil \frac{l_1}{2} \rceil \geq \frac{l_1}{2}$, 即 $\mathcal{P}_1$ 的长度至少是 $\mathcal{P}_2$ 的 2 倍.

13.1-7 试描述一颗含有 n 个关键字的红黑树, 使其红色内部结点个数与黑色内部结点个数的比值最大. 这个比值是多少? 该比值最小的树又是怎样呢? 比值是多少?

解: 由红黑树性质 2, 4, 5 知:

- 比值最大: 共有 3 个结点, 其中根结点为黑色, 根结点的两个孩子结点为红色, 比值为 2;
- 比值最小: 所有结点均为黑色, 比值为 0.

13.2-5 如果能够使用一系列 Right-Rotate 调用把一个二叉搜索树 T_1 变为二叉搜索树 T_2, 则称 T_1 可以右转 (right-converted) 成 T_2. 试给出一个例子, 表示两棵树 T_1 和 T_2 , 其中 T_1 不能够右转成 T_2 .然后, 证明: 如果 T_1 可以右转成 T_2 , 那么它可以通过 $O(n^2)$ 次 Right-Rotate 调用来实现右转.

解: T_1 不一定能够右转成 T_2, 例子如下:

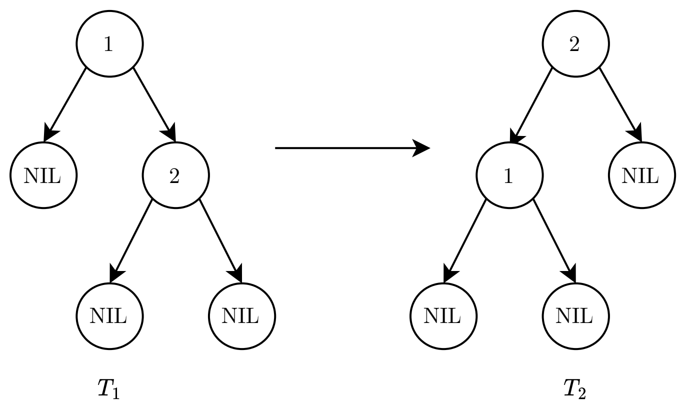

Figure 1. T_1 cannnot be right-converted to T_2.

注意到, 对 T_1 中任意结点 x, Right-Rotate(T_1, x) 为有效操作, 当且仅当 x 的左孩子存在. 换言之, 当任意结点 x 均不存在左孩子时, T_1 无法进行有效的 Right-Rotate 操作.

考虑调用 Right-Rotate(T_1, x). 我们将旋转前 x 的左子树的大小记作 $n_{x.left}$, x 的左孩子的右子树的大小记作 $n_{x.left.right}$. 旋转后, x 的左孩子的右子树成为 x 的左子树, 使得 x 的左子树的大小减少 $n_{x.left} - n_{x.left.right}$, 其余结点的左子树大小不变.

由于 T_1 中所有结点的左子树大小之和为 $O(n^2)$, 每次 Right-Rotate 调用均使该和至少减少 1 , 故至多能够执行 $O(n^2)$ 次 Right-Rotate 调用. 因此, T_1 可以通过 $O(n^2)$ 次 Right-Rotate 调用来右转成 T_2.

13.3-5 考虑一颗用 RB-Insert 插入 n 个结点而成的红黑树. 证明: 如果 $n > 1$, 则该树至少有 1 个红结点.

解: 考虑最后插入的结点 x, RB-Insert 首先将 x 染为红色. 若 $x.p$ 为黑色, 则不进行任何调整, 此时, 红黑树有红色结点 x; 若 $x.p$ 为红色, RB-Insert-Fixup 分情况对红黑树进行调整:

- 情况 1, 4 下, x 仍为红色, 待调整的结点向上传递. 此时, 红黑树有红色结点 x;
- 情况 2, 3, 5, 6 下, x 的祖父结点 y 被染为红色. 经过旋转后, y 必然不是根结点. 此时, 红黑树有红色结点 y.

综上所述, $n > 1$ 时, 红黑树至少有 1 个红结点.

14.1-5 给定 n 个元素的顺序统计树中的一个元素 x 和一个自然数 i, 如何在 $O(\lg n)$ 的时间内确定 x 在该树线性序中的第 i 个后继?

解: 伪代码如下:

Algorithm 1: 14.1-5

Input: a node x in an order-statistic tree T and a natural number i.
Output: the i-th successor of x in T.

1. r = OS-Rank(T, x)
2. **return** OS-Select(T, $r + i$)

OS-Rank 与 OS-Select 的时间复杂度均为 $O(\lg n)$, 故该算法的时间复杂度为 $O(\lg n)$.

14.1-7 说明如何在 $O(n \lg n)$ 时间内, 利用顺序统计数对大小为 n 的数组中的逆序对进行计数. (逆序对定义参考教材 P24 思考题 2-4).

解: 伪代码如下:

Algorithm 2: 14.1-7

Input: an array $A[1..n]$.
Output: the number of inversions in A.

1. Initialize an empty order-statistic tree T.
2. *count* $\leftarrow 0$
3. **for** $i \leftarrow 1$ **to** n:
4. x = OS-Insert(T, $A[i]$) // x is the newly inserted node.
5.   *count* $\leftarrow$ *count* $+ i -$ OS-Rank(T, x)
6. **return** *count*

注意到, 在 OS-Insert 中, 若新结点的关键字与其父结点的关键字相等, 则新结点总是作为其父结点的右孩子插入. 在每次迭代中, 插入新结点和计算新结点的秩均为 $O(\lg n)$ 时间. 因此, 该算法的时间复杂度为 $O(n \lg n)$.

14.3-3 请给出一个有效的算法, 对一个给定的区间 i, 返回一个与 i 重叠且具有最小低端点的区间; 或者当这样的区间不存在时返回 $T.nil$.

解: 伪代码如下:

Algorithm 3: 14.3-3

Input: an interval i.
Output: an interval j that overlaps with i and has the smallest low endpoint; or $T.nil$ if no such interval exists.

1. $x \leftarrow T.root$
2. $j \leftarrow T.nil$
3. **while** $x \neq T.nil$:
4.   **if** $x.int.low \leq i.high$ **and** $x.int.high \geq i.low$:
5. $j \leftarrow x$
6. $x \leftarrow x.left$
7.   **else if** $x.left \neq T.nil$ **and** $x.left.max \geq i.low$:
8. $x \leftarrow x.left$
9.   **else**:
10. $x \leftarrow x.right$
11. **return** j