

查找技术 1

Tan Yiqing

2025 年 12 月 8 日



1 概念

先介绍一些定义

1. 关键码：识别一个记录的某个数据项
2. 键码：关键码的值
3. 主关键码：可以唯一标识一个记录的关键码
4. 次关键码：不能唯一标识一个记录的关键码
5. 静态查找：不涉及插入和删除操作的查找，线性表用的多
6. 动态查找：涉及插入和删除操作的查找，树用的多

查找算法的时间性能通过关键码的比较次数来衡量。具体来说，定义：

平均查找长度 ASL (Average Search Length)：表示在进行多次查找操作后，平均每次查找所需的关键码比较次数。

$$ASL = \sum_{i=1}^n P_i \cdot L_i$$

其中 P_i 表示查找第 i 个记录的概率 (与算法无关)， L_i 表示查找第 i 个记录所需的关键码比较次数 (与算法有关)， n 表示线性表中的记录总数。

ASL 是衡量查找算法效率的重要指标，ASL 越小，表示查找效率越高。

2 线性表查找技术

2.1 顺序查找

顺序查找是最简单的一种查找方法。它从线性表的第一个记录开始，依次将每个记录的关键码与待查找的关键码进行比较，直到找到与待查找关键码相等的记录，或者搜索到线性表的末尾为止。

2.1.1 算法描述

没什么好描述的

2.1.2 时间复杂度分析

假设有 n 个记录，查找成功时，平均查找长度 ASL 为：

$$ASL_{success} = \frac{1 + 2 + \dots + n}{n} = \frac{n + 1}{2}$$

查找不成功时，平均查找长度 ASL 为：

$$ASL_{failure} = \frac{n + 1}{2}$$

算法复杂度为 $O(n)$ ，太慢了，但是比较好实现。

2.2 折半查找

2.2.1 使用条件

折半查找要求线性表中的记录必须按关键码有序，且采用顺序存储。

2.2.2 算法描述

每次将待查找的关键码与线性表中间位置的记录的关键码进行比较，

1. 如果相等则查找成功；
2. 如果待查找关键码小于中间位置记录的关键码，则在前半部分继续进行折半查找 ($high=mid-1$)；
3. 否则在后半部分继续进行折半查找 ($low=mid+1$)。

如果 $low > high$ ，则查找失败。

2.2.3 折半查找判定树

判定树其实是一棵二叉排序树，根节点表示待查找的关键码，左子树表示小于根节点的关键码，右子树表示大于根节点的关键码。

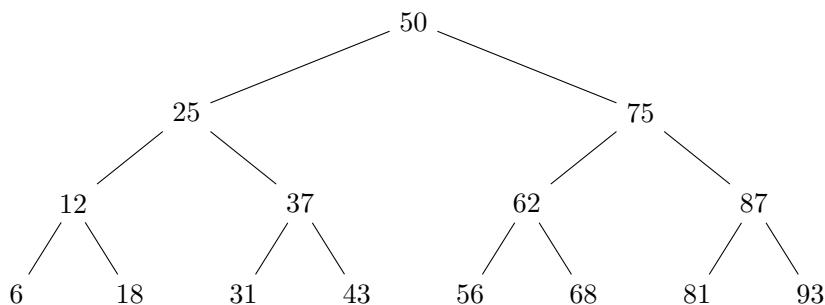


图 1: 折半查找判定树示例

2.2.4 时间复杂度分析

可以从判定树的高度来分析时间复杂度。对于 n 个记录的线性表，折半查找的判定树高度为 $\lfloor \log_2 n \rfloor + 1$ 。因此，折半查找的时间复杂度为 $O(\log n)$ 。这是比顺序查找更高效的查找方法。

3 树表查找技术

3.1 二叉排序树查找

3.1.1 二叉排序树

二叉排序树（Binary Search Tree, BST）是一种特殊的二叉树，也称二叉查找树，满足以下性质：

1. 或者是一个空树；
2. 或者满足以下性质：
 - (a) 若左子树非空，则左子树所有节点的关键码小于根节点的关键码。
 - (b) 若右子树非空，则右子树所有节点的关键码大于根节点的关键码。
 - (c) 左右子树也是二叉排序树。

二叉排序树的中序遍历结果是一个有序的序列。

3.1.2 二叉排序树的构建

构建二叉排序树的过程如下，是比较简单的：

1. 如果树为空，则将待插入的关键码作为根节点。
2. 否则，比较待插入的关键码与当前节点的关键码：
 - (a) 如果待插入关键码小于当前节点关键码，则递归地将其插入左子树。
 - (b) 如果待插入关键码大于当前节点关键码，则递归地将其插入右子树。

3.1.3 二叉排序树的删除

二叉排序树的删除比较复杂，删除节点时需要考虑三种情况：

1. 删除的节点是叶子节点，直接删除即可。
2. 删除的节点只有一个子节点，将该节点的子节点连接到其父节点。
3. 删除的节点有两个子节点，找到该节点的中序后继（右子树中最小的节点）或中序前驱（左子树中最大的节点），用该节点的关键码替换待删除节点的关键码，然后删除中序后继或前驱节点。

3.1.4 二叉排序树查找

其实和折半查找差不多：

1. 如果当前节点为空，则查找失败。
2. 如果待查找关键码等于当前节点关键码，则查找成功。
3. 如果待查找关键码小于当前节点关键码，则递归地在左子树中查找。
4. 如果待查找关键码大于当前节点关键码，则递归地在右子树中查找。

3.1.5 时间复杂度分析

取决于树的形状，在 $O(\log n)$ 到 $O(n)$ 之间变化。理想情况下，树是平衡的，时间复杂度为 $O(\log n)$ 。但在最坏情况下，树退化为链表，时间复杂度为 $O(n)$ 。

3.2 平衡二叉树查找

3.2.1 平衡二叉树

很自然地，我们希望二叉排序树的时间复杂度贴近 $O(\log n)$ ，为此引入平衡二叉树的概念。

平衡二叉树 平衡二叉树（AVL 树）是一种特殊的二叉排序树，满足以下性质：

1. 或者空树；
2. 或者具有以下性质：
 - (a) 根结点的左子树和右子树的深度最多相差 1.
 - (b) 左子树和右子树也是平衡二叉树。

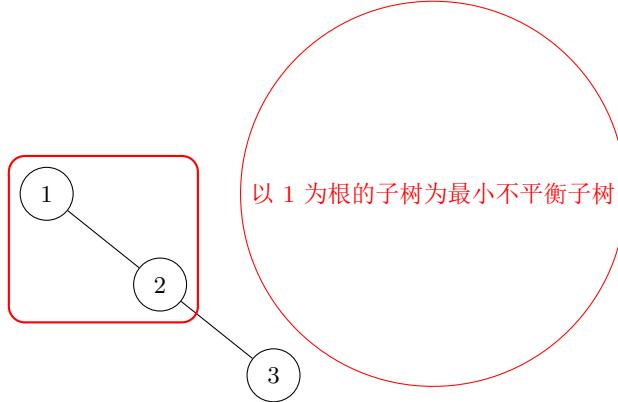
平衡因子 左子树深度减去右子树深度。

$$BF(node) = height(node.left) - height(node.right)$$

最小不平衡子树 在平衡二叉树的构造过程中，以距离插入结点最近的，且平衡因子绝对值大于 1 的结点为根结点的子树，称为最小不平衡子树。

例如，依次插入 1, 2, 3, 4 到一棵初始为空的平衡二叉树（AVL 树）：

1. 插入 1：树只有一个结点，无需调整。
2. 插入 2：2 作为 1 的右孩子，树依然平衡。
3. 插入 3：3 作为 2 的右孩子，此时 1 的平衡因子变为 -2，失衡。



此时，以结点 1 为根的子树平衡因子绝对值大于 1，且它是距离插入点 3 最近的失衡结点，所以以 1 为根的子树就是“最小不平衡子树”。只需对该子树做一次左旋，整棵树即可恢复平衡。

L 层平衡二叉树的结点数上下界

定理 1. 设一棵平衡二叉树（AVL 树）的高度为 k （共有 k 层，根为第 1 层），记该高度下所有平衡二叉树中结点数最少的为 M_k 。则：

上界：任意高度为 k 的二叉树，结点数不超过满二叉树：

$$N_k \leq 2^k - 1.$$

下界： M_k 满足

$$M_1 = 1, \quad M_2 = 2, \quad M_k = M_{k-1} + M_{k-2} + 1 \quad (k \geq 3),$$

且

$$M_k = F_{k+2} - 1,$$

其中 F_k 为 Fibonacci 数列， $F_1 = 1, F_2 = 1, F_k = F_{k-1} + F_{k-2}$ 。因此存在常数 $c > 0$ 和

$$\varphi = \frac{1 + \sqrt{5}}{2},$$

使得

$$M_k = \Omega(\varphi^k).$$

证明. 1. 上界

高度为 k 的任意二叉树，第 i 层最多 2^{i-1} 个结点：

$$N_k \leq \sum_{i=1}^k 2^{i-1} = 2^k - 1,$$

当树为满二叉树时取等号。

2. 下界的递推关系

AVL 树中，每个结点左右子树高度差不超过 1，且左右子树本身也是 AVL 树。记高度为 k 且结点数最少的 AVL 树为 T_k ，结点数为 M_k 。显然

$$M_1 = 1, \quad M_2 = 2.$$

对 $k \geq 3$, 设 T_k 根结点左右子树高度分别为 h_L, h_R 。则有

$$\max\{h_L, h_R\} = k - 1, \quad |h_L - h_R| \leq 1,$$

因此 $\{h_L, h_R\}$ 只能为 $(k - 1, k - 1)$, $(k - 1, k - 2)$ 或 $(k - 2, k - 1)$ 。

为使整棵树结点数最少, 应取高度更“不平衡”的情形, 即 $(k - 1, k - 2)$ (或对称的 $(k - 2, k - 1)$)。此时左右子树本身也应取各自高度下的最少结点数, 故

$$\text{左子树结点数} = M_{k-1}, \quad \text{右子树结点数} = M_{k-2}.$$

再加上根结点 1 个:

$$M_k = M_{k-1} + M_{k-2} + 1, \quad k \geq 3.$$

3. 与 Fibonacci 数列的关系

Fibonacci 数列定义为

$$F_1 = 1, \quad F_2 = 1, \quad F_k = F_{k-1} + F_{k-2} \quad (k \geq 3).$$

声称

$$M_k = F_{k+2} - 1, \quad k \geq 1.$$

用归纳法证明:

当 $k = 1, 2$ 时:

$$M_1 = 1 = F_3 - 1, \quad M_2 = 2 = F_4 - 1.$$

设对 $k - 1, k - 2$ 成立, 即 $M_{k-1} = F_{k+1} - 1$, $M_{k-2} = F_k - 1$, 则

$$\begin{aligned} M_k &= M_{k-1} + M_{k-2} + 1 \\ &= (F_{k+1} - 1) + (F_k - 1) + 1 \\ &= F_{k+1} + F_k - 1 = F_{k+2} - 1. \end{aligned}$$

故对一切 $k \geq 1$ 成立。

4. 漐近下界

Fibonacci 数列有通项公式 (Binet 公式):

$$F_k = \frac{\varphi^k - \psi^k}{\sqrt{5}},$$

其中

$$\varphi = \frac{1 + \sqrt{5}}{2}, \quad \psi = \frac{1 - \sqrt{5}}{2}, \quad |\psi| < 1.$$

因此存在常数 $c_1, c_2 > 0$, 对充分大的 k 有

$$c_1 \varphi^k \leq F_k \leq c_2 \varphi^k,$$

即 $F_k = \Theta(\varphi^k)$, 特别地 $F_k = \Omega(\varphi^k)$ 。由

$$M_k = F_{k+2} - 1$$

可知, M_k 与 F_{k+2} 同阶, 故

$$M_k = \Omega(\varphi^{k+2}) = \Omega(\varphi^k).$$

综上, 高度为 k 的 AVL 树的最少结点数满足

$$M_k = F_{k+2} - 1 = \Omega(\varphi^k),$$

最多结点数为 $2^k - 1$, 命题得证。 □

n 个结点平衡二叉树的层数上下界

定理 2. 设一棵平衡二叉树 (AVL 树) 有 n 个结点, 高度 (层数) 为 k 。则其高度满足

$$c_1 \log n \leq k \leq c_2 \log n$$

其中 $c_1, c_2 > 0$ 为与 n 无关的常数。也就是说, 平衡二叉树的高度是 $k = \Theta(\log n)$ 。

证明. 记高度为 k 的 AVL 树的最少结点数为 M_k , 最多结点数为 N_k 。

由前一条定理我们已经知道:

$$M_k = F_{k+2} - 1, \quad M_k = \Omega(\varphi^k), \quad N_k \leq 2^k - 1,$$

其中 $\varphi = \frac{1 + \sqrt{5}}{2} > 1$, F_k 为 Fibonacci 数列。

1. 利用下界 M_k 求高度的上界

对任意高度为 k 的 AVL 树, 有

$$n \geq M_k.$$

而 $M_k = \Omega(\varphi^k)$, 即存在常数 $c > 0$, 以及充分大的 k_0 , 使得对 $k \geq k_0$:

$$M_k \geq c \varphi^k.$$

于是

$$n \geq M_k \geq c \varphi^k,$$

从而

$$\varphi^k \leq \frac{n}{c}, \quad k \leq \log_\varphi \left(\frac{n}{c} \right) = O(\log n).$$

这给出了 AVL 树高度关于结点数的一个上界: 存在常数 $c_2 > 0$, 使得

$$k \leq c_2 \log n.$$

2. 利用上界 N_k 求高度的下界

另一方面, 对任意高度为 k 的二叉树, 有

$$n \leq N_k \leq 2^k - 1.$$

因此

$$n + 1 \leq 2^k, \quad k \geq \log_2(n + 1).$$

从而存在常数 $c_1 > 0$, 使得

$$k \geq c_1 \log n.$$

3. 合并两侧不等式

综合 (1) 与 (2), 得到: 对结点数为 n 的任意 AVL 树, 其高度 k 满足

$$c_1 \log n \leq k \leq c_2 \log n,$$

即

$$k = \Theta(\log n).$$

定理得证。 □

事实上, n 个结点的平衡二叉树的高度下界更精确地为完全二叉树的情况:

$$k \geq \lfloor \log_2 n \rfloor + 1.$$