斜堆复杂度分析报告

潘屹

上海交通大学 ACM 班 (电院 2231), 522031910741

摘要

本文对斜堆的时间复杂度进行了分析，参考 D.D. Sleator 与 R.E. Tarjan 的势能分析法 [1]，证明其归并操作的均摊时间复杂度为

$$T(n) = O(\log n).$$

进一步地，参考 B. Schoenmakers 的方式 [2][3]，能得到该操作在最坏情况下的均摊时间

$$T(n) \leq \log_{\Phi} n, \quad \Phi = \frac{1+\sqrt{5}}{2}.$$

1 原理简析

斜堆是左偏树的一个变种，由 D.D. Sleator 与 R.E. Tarjan 在 1986 年提出。其与左偏树的主要区别在于不保证左儿子高度大于等于右儿子，即在 merge 操作中，不进行左右儿子高度的判断；而是每一次将右儿子与新堆归并后，都进行左右儿子的交换操作。归并操作的 C++ 实现代码如下：

```
tnode* merge(tnode *x, tnode *y) {
    if (x == nullptr)
        return y;
    if (y == nullptr)
        return x;
    if (!Compare(x->data, y->data))
        std::swap(x, y);
    merge(x->right, y);
    std::swap(x->left, x->right);
    return x;
}
```

通过上述操作，可以对斜堆的原理进行感性理解，即，通过每次右子树归并后交换的方式，尽量维持整棵二叉树的左右子树深度稳定，进行自我调整（self-adjusting），从而使得每一次归并操作的复杂度均摊在 $O(\log n)$[1] 。下面的分析将对这一点进行严格证明。

2 势能分析法

在 D.D. Sleator 与 R.E. Tarjan 的文章中，使用了势能（Potential）分析的方式进行斜堆均摊复杂度的上界分析。势能分析是一种对数据结构进行均摊复杂度分析的常见方式，对于一个数据结构 S

[1]本文中若无特别说明，log 表示以 2 为底数的对数。

（例如，本例中的斜堆），我们可以定义其势能为 $\Phi(S)$，这是一个会随数据结构的形态而改变的值。实际上，可以进行感性理解：当一个数据结构的复杂程度（例如最大深度）增大时，其势能[2]也随之增大。因此，定义一个操作 i 的摊还时间消耗

$$a_i = t_i + (\Phi_i - \Phi_{i-1}).$$

于是在一个操作序列 $(op_i)_{i=1}^m$ 中，总时间消耗为

$$\sum_{i=1}^{m} t_i = \sum_{i=1}^{m} a_i - \Delta\Phi.$$

通常 Φ_i 是一个初值为 0 而保持非负的值，这样一来，$\sum a_i$ 就是 $\sum t_i$ 的一个上界。可以看出，势能分析法的关键在于选择适当的势能函数 Φ。下面通过势能分析法对斜堆的复杂度进行分析。

3 归并操作的复杂度上界

考虑到斜堆的插入、删除操作的本质实际上都是进行归并操作，所以在分析斜堆的时间复杂度时，只需要对归并操作进行分析即可。

定义 1 (重节点)**.** 对于一个非叶子节点 u，其子节点 v 为 u 的**重儿子**当且仅当 v 的子树大小大于 u 子树大小的一半，并称 u 为树上的**重节点**，反之则为**轻节点**。

关于轻重节点有一些显然的推论：一个节点至多只有一个重儿子；一个轻儿子的子树大小不超过其父节点子树大小的一半，从而可以导出引理。

引理 2. 对于一个非叶子节点 u 与其子树中的一个节点 v，记它们的子树大小分别为 w_u, w_v，设从 u 到 v 的路径中轻节点个数为 k（u 不计入），则有

$$k \le \log \frac{w_u}{w_v}.$$

证明. 设这条路径的轻节点为 $(x_i)_{i=1}^k$，按从 u 到 v 的路径顺序，并记它们的子树大小为 w_{x_i}，则显然有

$$w_{x_{i+1}} < \frac{w_{x_i}}{2}\ (i \le 1) \quad \text{且} \quad w_{x_1} < \frac{w_u}{2}, w_v \le w_{x_k}.$$

于是

$$w_v \le w_u 2^{-k}.$$

□

推论 3. 在节点数为 n 的二叉树的任意一条从顶向下的路径中，共有不超过 $\lfloor \log n \rfloor$ 个轻节点。

上面的内容是关于二叉树中轻重节点的基本定义，据此展开斜堆复杂度的分析，首先是最关键的势能函数定义：

定义 4. 对于一个斜堆 S，定义其势能函数 $\phi(S)$ 为其所有同时为重儿子与右儿子的节点个数。

随后通过选定的势能函数进行摊还时间估计：

[2]Here *potential* refers to its potential to increase later time cost.

定理 5. 对于归并后节点数为 n 的斜堆，其单次归并过程 i 的摊还时间满足

$$a_i = O(\log n).$$

证明. 设归并前的两个斜堆 h_1, h_2 的总结点数为 n_1, n_2，在其右路径上的重节点个数为 k_1, k_2。在一次归并过程中，时间消耗可用其右路径上的节点总数来衡量，即轻儿子与重儿子的个数之和，依引理与相关节点的定义

$$t_i \le \log n_1 + k_1 + \log n_2 + k_1 + k_2 \le 2\log n + k_1 + k_2.$$

再考虑势能变化，显然，在一次归并操作中，右路径上所有的重节点必然与左子树发生交换，进而转变为轻节点，带来 $-(k_1 + k_2)$ 的势能变化；对于轻节点转化为重节点的情况，考虑最坏（势能增加量最大）的情况，即所有路径上的轻节点都转化为重节点，此时势能变化量为

$$\Phi_i - \Phi_{i-1} \le \log n - k_1 - k_2.$$

于是有

$$a_i = t_i + \Phi_i - \Phi_{i-1} \le 3\log n = O(\log n).$$

□

这样就证明了，斜堆的归并、插入、删除操作的复杂度均为 $O(\log n)$，查询最值的操作为 $O(1)$。

4 精确时间

B. Schoenmakers 对斜堆归并操作的时间复杂度进行了更加精确的分析，其中的数学思想较为深刻；下面的内容参考其中方法，得到该操作在最坏情况下的均摊时间。

笔者还没看懂，下次再补。

参考文献

[1] Daniel Dominic Sleator and Robert Endre Tarjan. Self-adjusting heaps. *SIAM Journal on Computing*, 15(1):52–69, 1986. (document)

[2] Anne Kaldewaij and Berry Schoenmakers. The derivation of a tighter bound for top-down skew heaps. *Information Processing Letters*, 37(5):265–271, 1991. (document)

[3] Berry Schoenmakers. A tight lower bound for top-down skew heaps. *Information processing letters*, 61(5):279–284, 1997. (document)