**[左偏树](http://www.byvoid.com/blog/leftist-tree/" \o "Permanent Link to 左偏树)**

[計算機科學](http://www.byvoid.com/blog/cate/computing-science/" \o "查看 計算機科學 的全部文章) [Add comments](http://www.byvoid.com/blog/leftist-tree/#respond)246 views

**[可并堆与左偏树]**

我们最常用的二叉堆，是最常用的优先队列，它可以在O(logN)内实现插入和删除最小值操作。但是对于合并两个有序的优先队列，二叉堆就显得力不从心了。

左偏树是一种**可并堆(Mergeable Heap)**，意思是可以在O(logN)时间内完成两个堆的合并操作。左偏树(Leftist Tree)，或者叫左倾树，左式树，左式堆(Leftist Heap)，左堆。顾名思义，它好象是向左偏的，实际上它是一种趋于非常不平衡的二叉树结构，但却能够实现对数级的合并时间复杂度。

**[左偏树的定义]**

左偏树是一棵二叉树，每个节点具有四个属性：左子树(left)，右子树(right)，键值(key)，**零路径长**(npl)。其中节点i的零路径长的定义为**从i到i子树中最近的没有两非空个子节点的节点的路径的长度**。空节点的零路径长为为-1，只有一个非空子节点的节点的零路径长为0。左偏树的节点之间除了满足堆序以外，还应满足**节点左子节点的零路径长不小于右子节点的零路径长**。

根据上述定义，很容易得出：对于非空节点i，满足i.npl=i.right.npl+1。还有一个性质，一棵节点数为N的左偏树，根节点的零路径长最大为⎣log(N+1)⎦-1，即O(logN)，证明略。

**[左偏树的合并]**

左偏树的一切操作都是在合并的基础上进行的，所以要先讨论合并。

当合并两个左偏树节点a,b时，要求是这两个节点是没有包含关系的。假设a.key<b.key(如果不是这样则交换a,b)，只需对a.right和b合并，合并后的结果作为新的a.right。然后更新a.npl=a.right.npl+1。

可以证明，一次合并的时间复杂度为O(log(A.size) + log(B.size))。

**[左偏树的插入和删除]**

对已有的左偏树a插入新的值p，只需把p构建为一个只有一个元素左偏树节点b，然后合并a,b即可。

删除左偏树的最小节点，只需把根节点删除，然后合并两个子树，作为新的根节点。

**[左偏树的构建]**

可以用一个队列，使左偏树的构建为O(N)。具体方法为

1. 把所有元素作为一个单独左偏树节点放入队列；
2. 不断取出两个队首的左偏树，合并这两个左偏树，然后放入队尾；

当队列中只剩下一个左偏树，算法结束。可以证明，时间复杂度为O(N)。

**[对左偏树的比较]**

左偏树可以实现二叉堆的一切功能，而且还能实现二叉堆不易实现的合并，个人认为实际编程中左偏树更有理性，不容易错。但左偏树的算法时间常数要大于二叉堆，所以不能完全代替之。

和平衡树相比，左偏树采取了与平衡树完全相反的构造策略。平衡树为了实现所有元素的快速查找，使节点尽量趋于平衡。而左偏树的目的是实现快速的查询最小值与合并操作，恰恰要让节点尽量向左偏。**最优的平衡树，恰恰是最差的左偏树，而最优的左偏树，恰恰是平衡树退化的结果。**

斜堆、二项堆、斐波那契堆也是可并堆实现的有效方法，而且二项堆、斐波那契堆实际中会比左偏树更快，但是在时间与编程复杂度的性价比上，左偏树有着绝对的优势。

[*BYVoid*](http://www.byvoid.com/) *原创讲解，转载请注明。*