斐波那契堆报告

孔冰玉，苏雨峰，张嘉恒，徐晓骏

2016 年 4 月 10 日

1 功能介绍

斐波那契堆是一种可合并堆，可用于实现合并优先队列。我们的斐波那契堆实现了以下功能：

- $\mathcal{O}(1)$ 时间内返回所有元素的最小（大）值。
- $\mathcal{O}(1)$ 时间内插入一个元素。
- $\mathcal{O}(1)$ 时间内合并两个斐波那契堆。
- $\mathcal{O}(\log n)$ 均摊时间内删除最小（大）元素。
- $\mathcal{O}(\log n)$ 均摊时间内删除任意指定元素。

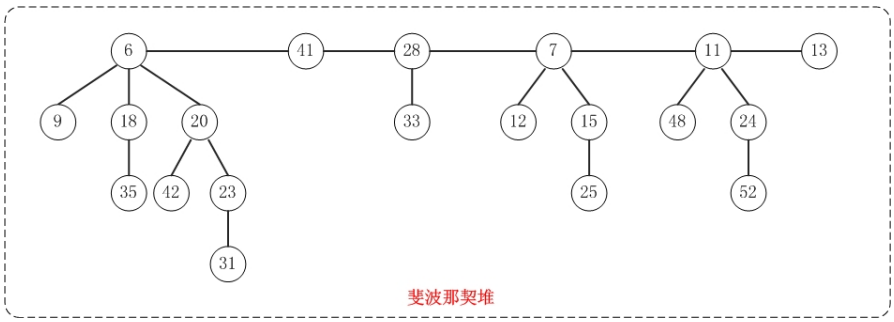

Figure 1: 斐波那契堆示意图

2 原理说明

斐波那契堆由一组满足堆性质的有根树组成，这些有根树满足两个性质：1. 若进行删除操作，每一个非根节点不能被删除两次或两次以上的孩子。2. 进行“维护”操作后，堆中不会有度数相同的两棵树。每个操作的实现原理如下：

2.1 返回最小值

维护一个指针指向根链表中的最小值，因此查找最小值操作仅需返回该值即可。

2.2 插入元素

斐波那契堆的插入操作是“懒惰”的，即插入之后不会维护各项性质。只需将新元素看作一棵树直接插入到根链表中，并维护最小值指针即可。

2.3 合并斐波那契堆

与插入操作类似，合并两个堆只需将一个堆的根链表并入第二个堆中，并维护最小值指针。

2.4 取出最小元素 (pop 操作)

pop 操作较为复杂。首先需要将最小元素从根节点的链表中剥离，并把最小元素的子树串联进根链表中。然后要对斐波那契堆的根链表做“维护”操作，使得堆中不存在度数 (节点数量) 相等的两棵树，即：扫描根链表，每当遇到两棵度数相等的树，就使其中根节点元素较大的一棵成为另一棵的子树。

2.5 删除指定元素

使用 map 维护一个从元素值到堆中节点位置的映射，这样就可以迅速定位指定元素的位置。删除操作分为三步。1. 将该元素从原位置取出，并加入根链表中，并将该节点的元素值置为最小。2. 对其原位置的父节点递归进行“级联剪切”，保证每个非根节点不会被删除两次孩子。具体的实现原理为：维护一个标记 mark 记录其是否已被删除过孩子。级联剪切中，若 mark 为 0，则置为 1；若 mark 为 1，则将该节点从原位置取出加入根链表，并对原来的父节点递归进行级联剪切。3. 执行 pop 操作，即删除了指定的元素节点。

3 正确性分析

由于每个操作都保证堆中的树满足堆性质，且每次都会维护指针指向根链表中的最小值，因此可以保证每次返回的最小值都是正确的。

4 时空复杂度证明

4.1 返回最大（小）值

由于斐波那契堆的根节点直接指向了最大（小）值，所有上述操作的时间复杂度为 $\mathcal{O}(1)$.

4.2 插入一个元素

首先我们先来介绍一下在数据机构复杂度分析中很重要的一个概念，叫做势能法摊还分析。

势能法的工作方式如下。我们将对一个初始数据结构 D_0 执行 n 个操作。对每个

$i = 0, 1, \ldots n$, 令 c_i 为第 i 个操作的实际代价，令 D_i 为在数据结构 D_{i-1} 上执行第 i 个操作得到的结果数据结构。势函数 Φ 将每个数据结构 D_i 映射到一个实数 $\Phi(D_i)$，此值即为关联到数据结构 D_i 的势。第 i 个操作的摊还代价 $\hat{c_i}$ 用势函数定义为：

$$\hat{c_i} = c_i + \Phi(D_i) - \Phi(D_{i-1})$$

因此，每个操作的摊还代价等于实际代价加上此操作引起的势能变化。于是有 n 个操作的总摊还代价为：

$$\sum_{i=1}^{n} \hat{c_i} = \sum_{i=1}^{n} c_i + \Phi(D_n) - \Phi(D_0)$$

如果能定义一个势函数 Φ 使得 $\Phi(D_n) \geq \Phi(D_0)$，则总摊还代价 $\sum_{i=1}^{n} \hat{c_i}$ 给出了总实际代价 $\sum_{i=1}^{n} c_i$ 的一个上界。实际中，我们不是总能知道将要执行多少个操作。因此，如果对所有 i，我们要求 $\Phi(D_i) \geq \Phi(D_0)$，则可以像核算法一样保证总能提前支付。

了解了势函数的概念，接下来我们来看插入一个元素的时间复杂度，这里利用势能法均摊分析。首先引入斐波那契堆的势函数：

$$\Phi(H) = t(H) + 2m(H)$$

其中 $t(H)$ 代表 H 中根链表中树的数目，$m(H)$ 表示 H 中已经标记的节点数目。那么对于插入操作，势增加量为 1，而且实际代价为 $\mathcal{O}(1)$, 因此摊还代价为 $\mathcal{O}(1)$.

4.3 合并两个斐波那契堆

同样利用斐波那契堆的势函数。合并两个堆过程中势函数变化为：

$$\Phi(H) - (\Phi(H_1) + \Phi(H_2)) = (t(H) + 2m(H) - (t_1(H) + 2m(H_1) + t_2(H) + 2m(H_2))) = 0$$

所以其摊还代价等于它的实际代价 $\mathcal{O}(1)$.

4.4 删除最大（小）元素

为了分析这个操作的复杂度，我们先来分析斐波那契堆中最大度数的界 $D(n)$. 假设 x 是斐波那契堆中的任意节点，它的度数为 k，设 $y_!, y_2, \ldots, y_k$ 是它的孩子，并且以链入 x 的顺序排列，则有 $y_i.degree \geq i - 2 (i \geq 2)$. 这是由于当 y_i 链入 x 时，$y_i.degree = x.degree \geq i-1$, 之后其最多失去一个孩子，于是有 $y_i.degree \geq i-2 (i \geq 2)$. 还有斐波那契数自身的性质：

$$F_{k+2} = 1 + \sum_{i=0}^{k} F_i$$

$$F_{k+2} \geq \phi^k$$

其中 ϕ 为黄金分割率。

下面接着证明，x 为斐波那契堆中任意节点，$k = x.degree$，则用 $size(x) \geq F_{k+2}$。记 s_k 为度数为 k 的节点最小 $size$, 则有：

$$size(x) \geq s_k \geq 2 + \sum_{i=2}^{k} s_{i-2}$$

利用数学归纳法可证得 $s_k \geq F_{k+2}$，综上有：

$$size(x) \geq \phi^k$$

则可得到：

$$D(n) = \mathcal{O}(lgn)$$

对于删除最大（小）元素，根据我们实现算法步骤，总实际工作量为 $\mathcal{O}(D(n) + t(H))$，而在删除之前势为 $t(H) + 2m(H)$，删除之后最多有 $D(n) + 1$ 个根留下并且该过程没有任何节点被标记。于是摊还代价最多为：

$$\mathcal{O}(D(n) + t(H)) + D(n) + 1 + 2m(H) - t(H) - 2m(H) = \mathcal{O}(D(n)) = \mathcal{O}(lgn)$$

4.5 删除指定任意元素

可以将这个过程分为两步，第一步将该元素的变为整个斐波那契堆的最小值，然后通过一些列操作把它变成新的最小节点。这个过程的摊还时间是 $\mathcal{O}(1)$ 的，第二步删除最小元素，这个过程的摊还时间是 $\mathcal{O}(lgn)$, 于是整个过程的摊还时间是 $\mathcal{O}(lgn)$

5 具体实现方式和细节

具体实现中，我们使用一个双链表来存储所有的根节点，并且每个节点的所有孩子也用双链表来存储。这样，插入元素与合并两个堆的操作就变为了在双链表中插入一个节点与合并两个双链表的操作，都可以简单地在 $\mathcal{O}(1)$ 时间内完成。

对于删除后的维护操作，我们将根链表中的树根据度数大小从小到大排序并扫描，每遇见两个度数大小相等的树就合并，这样即可完成维护。