清华大学软件学院
算法分析与设计基础
2020 年春季学期

<u>作业 **7**</u>

吴佳龙　班级：软件 83　学号：2018013418　April 23, 2020

7.1. (CLRS Exercises 17.4-3)

证明. 对于删除操作，分情况讨论：

若 $\alpha_{i-1} > \frac{1}{2}, \alpha_i > \frac{1}{2}$，则

$$\begin{aligned}\hat{c}_i &= 1 + (2num_i - size_i) - (2num_{i-1} - size_{i-1}) \\ &= 1 + 2(num_{i-1} - 1) - size_{i-1} - 2num_{i-1} + size_{i-1} \\ &= 1 - 2 = -1\end{aligned}$$

若 $\alpha_{i-1} > \frac{1}{2}, \frac{1}{3} \le \alpha_i \le \frac{1}{2}$，则

$$\begin{aligned}\hat{c}_i &= 1 + (size_i - 2num_i) - (2num_{i-1} - size_{i-1}) \\ &= 1 + size_{i-1} - 2(num_{i-1} - 1) - 2num_{i-1} + size_{i-1} \\ &= 3 + 2(size_{i-1} - 2num_{i-1}) \\ &\le 3 + 2 \cdot (-1) = 1\end{aligned}$$

若 $\alpha_{i-1} > \frac{1}{2}, \alpha_i < \frac{1}{3}$，这只可能是 $size_{i-1} = num_{i-1} = 1$，则

$$\hat{c}_i = 1 + 0 - (2num_{i-1} - size_{i-1}) = 1 + 0 - 1 = 0$$

若 $\alpha_{i-1} \le \frac{1}{2}, \alpha_i \ge \frac{1}{3}$，则

$$\begin{aligned}\hat{c}_i &= 1 + (size_i - 2num_i) - (size_{i-1} - 2num_{i-1}) \\ &= 1 + size_{i-1} - 2(num_{i-1} - 1) - size_{i-1} + 2num_{i-1} \\ &\le 1 + 2 = 3\end{aligned}$$

若 $\alpha_{i-1} \le \frac{1}{2}, \alpha_i < \frac{1}{3}$，则

$$\begin{aligned}\hat{c}_i &= num_{i-1} + (size_i - 2num_i) - (size_{i-1} - 2num_{i-1}) \\ &= num_{i-1} + \frac{2}{3}size_{i-1} - 2(num_{i-1} - 1) - size_{i-1} + 2num_{i-1} \\ &= 2 + num_{i-1} - \frac{1}{3}size_{i-1} \\ &\le 2 + \frac{2}{3} \le 3\end{aligned}$$

□

7.2. (CLRS Problems 17-2)

(a) 由于不同数组之间的数不存在大小关系，我们只能遍历每一个数组并在上面进行二分查找。在数组 A_i 上进行二分查找的时间复杂度是 $\Theta(\lg 2^i) = \Theta(i)$。最坏情况下，每个数组都不为空，且在最后一个数组中找到（或任何数组中都不存在），时间复杂度是 $1 + 2 + \cdots + k = \Theta(k^2) = \Theta(\lg^2 n)$

(b) 插入算法描述为：找到最小的 i 使得 $n_i = 0$，即 A_i 为空，然后将 $A_j, j < i$ 中的元素取出，和新加入的元素 a 全部移动到 A_i。为了使 A_i 中的元素有序，我们需要进行 $i-1$ 次二路归并，先将 $\{a\}$ 与 A_0 归并，然后将 $\{a\} \cup A_0$ 与 A_1 归并，以此类推，最终将 $\{a\} \cup \bigcup_{j=0}^{i-2} A_j$ 与 A_{i-1} 归并。时间复杂度为

$$\Theta(1) + \Theta(2) + \cdots + \Theta(2^{i-1}) = \Theta(2^i)$$

最坏 最坏情况下，$n_0 = \cdots = n_{k-1} = 1$ 则 $i = k$，我们需要合并所有 $A_0, \cdots, A_{k-1}$，时间复杂度为 $\Theta(2^k) = \Theta(n)$。

均摊 采用 accounting method 分析均摊复杂度，令每次插入的均摊花费 $\hat{c}_i = \Theta(k)$。由于每个元素会被插入一次，最多只会被从 A_i 移动到 A_j（$i < j$）不超过 $k-1$ 次，而元素被插入或者移动会对时刻 t 的 c_t 产生 $\Theta(1)$ 的贡献，因此满足 $\sum_{t=1}^{n} \hat{c}_t \geq \sum_{t=1}^{n} c_t$ 。因此每次插入的均摊复杂度是 $O(k) = O(\lg n)$

(c) 删除算法描述为：找到需要删除的元素所在的数组 A_i，找到最小的 j 使得 $n_j = 1$。若 $j \neq i$ ，则从 A_j 中取出一个元素与 A_i 中要删除的元素互换并调整 A_i 有序。然后删除待删除的元素，将 A_j 中的剩余元素移动到 $A_0, \cdots, A_{j-1}$（注意 A_j 是有序的，分裂可以在线性时间完成）。

最坏 最坏情况下，$i = k-1, j = k-2$，找到 A_i 需要 $\Theta(\lg^2 n)$，找到 A_j 需要 $\Theta(\lg n)$，调整 A_i 有序需要 $\Theta(n)$，将 A_j 的元素分裂需要 $\Theta(n)$。总的时间复杂度是 $\Theta(n)$。

均摊 如果在上述的最坏情况下，交替进行删除和插入，则每次操作的时间复杂度都是 $\Theta(n)$，这说明在同时存在插入和删除操作的情况下，均摊复杂度也不会好于 $\Theta(n)$。

7.3. (CLRS Problems 19-3)

(a) 如果 $k < x.key$ 则，直接调用 FIB-HEAP-DECREASE-KEY，均摊复杂度 $\Theta(1)$；
若 $k = x.key$，不进行任何操作，复杂度 $\Theta(1)$；
若 $k > x.key$ 删除 x 结点并加入一个新的值为 k 的结点，均摊复杂度为 $O(\lg n) + \Theta(1) = O(\lg n)$

(b) 修改势函数为：$\phi(H) = t(H) + 2m(H) + 2size(H)$，因为对于原先的所有操作，操作前后 size 只可能变化不超过 1，因此，原先所有操作的均摊复杂度的分析过程与结果不变。

修改数据结构：对于所有的叶节点，建立一个链表（不需要有序）。在之前所有操作中进行 CUT 和 LINK 的时候只要检查是否造成了某些结点成为叶节点或非叶节点，在链表中加入和删除即可，只需要常数时间，因此之前所有操作均摊复杂度仍不变。

FIB-HEAP-PRUNE 的实现为，从叶子链表中取出前若干个，从堆中删去，伪代码为：

```
FIB-HEAP-PRUNE(H, r):
    q=min(r, H.n)
    for i = 1..q:
        let x = head of leaf list and remove from it
        y = x.p
```

```
        if y==NIL:
            remove x from root list
        else:
            CUT(H,x,y)
            CASCADING−CUT(H,y)
```

之所以需要 CASCADING-CUT，是为了保持 Lemma 19.1 中的性质依然成立，所以仍然有 $D(n) = O(\lg n)$，原先操作的均摊复杂度不变。

最坏情况下，每一次都需要执行 CASCADING-CUT。FIB-HEAP-PRUNE 的均摊复杂度为常数时间：

$$
\begin{aligned}
\hat{c} &= c_{cut} + c_{else} + [t(H') + 2m(H') + 2(size(H) - q)] - [t(H) + 2m(H) + 2size(H)] \\
&= c_{else} - q + [c_{cut} + (t(H') + 2m(H')) - (t(H) + 2m(H))] \\
&= O(q) - 2q + O(q) = O(1)
\end{aligned}
$$

这里 c_{cut} 表示 CUT 和 CASCADING-CUT 的实际复杂度，根据课上的分析，一次 CASCADING-CUT 的均摊复杂度是 $O(1)$ 的，因此 q 次是 $O(q)$ 的；c_{else} 表示其他所有操作的实际复杂度，也是 $O(q)$ 的。