傅申 PB20000051
2022 秋

算法基础 作业 3

6.1–3. 证明: 在最大堆的任一子树中, 该子树所包含的最大元素在该子树的根结点上.

解: 采用反证法, 假设该子树所包含的最大元素 $A[i]$ 不在该子树的根结点上, 则最大元素的父亲也在该子树中, 有 $A[\text{PARENT}(i)] < A[i]$, 不满足最大堆性质, 矛盾. 因此该子树所包含的最大元素在该子树的根结点上.

6.1–5. 一个已排好序的数组是一个最小堆吗?

解: 因为 $\text{PARENT}(i) = \lfloor i/2 \rfloor \leqslant i$, 所以对一个已排好序的数组, 除了根以外的结点 i 都有 $A[\text{PARENT}(i)] \leqslant A[i]$, 满足最小堆性质. 因此一个已排好序的数组是一个最小堆.

6.2–3. 当元素 $A[i]$ 比其孩子的值都大时, 调用 MAX-HEAPIFY (A, i) 会有什么结果?

解: 在第 7 行结束后, $largest = i$, 不会进入第 8 行的 **if** 语句, 直接结束运行而不改变任何元素.

6.2–5. MAX-HEAPIFY 的代码效率很高, 但第 10 行中的递归调用可能例外, 它可能使某些编译器产生低效的代码. 请用循环控制结构取代递归, 重写 MAX-HEAPIFY 代码.

解: 如下

Algorithm 1: LOOP-MAX-HEAPIFY (A, i)

```
1  while i ⩽ A.heap-size
2      l = LEFT(i)
3      r = RIGHT(i)
4      if l ⩽ A.heap-size and A[l] > A[i]
5          largest = l
6      else
7          largest = i
8      if r ⩽ A.heap-size and A[r] > A[i]
9          largest = r
10     if largest = i
11         break
12     exchange A[i] with A[largest]
13     i = largest
```

6.3–3. 证明: 对于任一包含 n 个元素的堆中, 至多有 $\lceil n/2^{h+1} \rceil$ 个高度为 h 的结点.

解: 记高度为 h 的结点有 n_h 个. 使用数学归纳法, 先考虑 n_0. 因为高度为 0 的结点就是叶节点, 而堆中叶节点的个数为 $\lceil n/2 \rceil$, 所以满足

$$n_0 \leqslant \lceil n/2^{0+1} \rceil \tag{3.1}$$

现在假设 n_{h-1} 满足 $n_{h-1} \leqslant \lceil n/2^{(h-1)+1} \rceil$, 则对于 n_h, 若 n_{h-1} 为偶数, 则 $n_h = n_{h-1}/2 = \lceil n_{h-1}/2 \rceil$; 若 n_{h-1} 为奇数, 则 $n_h = \lfloor n_{h-1}/2 \rfloor + 1 = \lceil n_{h-1}/2 \rceil$. 因此

$$n_h = \left\lceil \frac{n_{h-1}}{2} \right\rceil \leqslant \left\lceil \frac{\lceil n/2^{(h-1)+1} \rceil}{2} \right\rceil = \lceil n/2^{h+1} \rceil \tag{3.2}$$

所以对所有的 h, 至多有 $\lceil n/2^{h+1} \rceil$ 个高度为 h 的结点.

6.4–4. 证明: 在最坏情况下, HEAPSORT 的时间复杂度是 $\Omega(n \lg n)$.

解: 先只考虑 HEAPSORT 的循环部分, 若每次调用 MAX-HEAPIFY 都要将根结点逐步交换到最低的结点 (比如每次都交换到 $A[A.heap\text{-}size]$), 则出现最坏情况, 比如输入数组已降序排序好. 在这种情况下, 每次每次调用 MAX-HEAPIFY 都要花费 $\Theta(\lfloor \lg i \rfloor)$ 的时间, 所以循环部分的时间复杂度为

$$\sum_{i=n}^{2} \Theta(\lfloor \lg i \rfloor) + \Theta(1) = \Theta(\lg n! + n) = \Theta(n \lg n) \tag{4.1}$$

而 BUILD-MAX-HEAP 的时间复杂度为 $O(n)$, 所以最坏情况下 HEAPSORT 的时间复杂度满足 $T(n) = \Theta(n \lg n) + O(n)$, 显然有下界 $\Omega(n \lg n)$.

6.5–2. 试说明 MAX-HEAP-INSERT $(A, 10)$ 在堆 $A = \langle 15, 13, 9, 5, 12, 8, 7, 4, 0, 6, 2, 1 \rangle$ 上的操作过程.

解: 如图 **??** 所示.

6.5–4. 在 MAX-HEAP-INSERT 的第 2 行, 为什么我们要先把关键字设为 $-\infty$, 然后又将其增加到需要的值呢?

解: 先将关键字设为 $-\infty$ 能保证调用 HEAP-INCREASE-KEY 时, 输入的数组满足最大堆性质, 是一个最大堆.

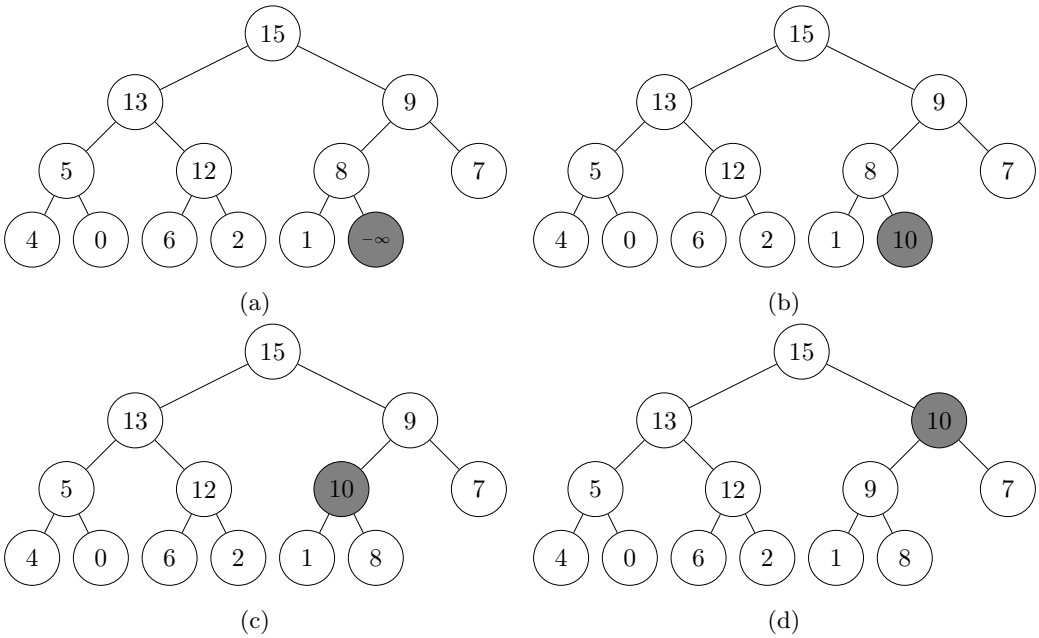

图 2.1: MAX-HEAP-INSERT(A, 10) 的操作过程

7.2–2. 当数组 A 的所有元素都具有相同值时, QUICKSORT 的时间复杂度是什么?

解: 所有元素都具有相同值时, PARTITION 会将数组分为长度为 0 和 $n-1$ 的两个子数组, 即最坏情况, 时间复杂度为 $\Theta(n^2)$.

7.2–5. 假设快速排序每一层所做的划分的比例都是 $1-\alpha:\alpha$, 其中 $0<\alpha\leqslant 1/2$ 且是一个常数. 试证明: 在相应递归树中, 叶节点的最小深度大约是 $-\lg n/\lg a$, 最大深度大约是 $-\lg n/\lg(1-\alpha)$. (无需考虑整数舍入问题).

解: 递归树深度最小的叶结点对应每次都取较小的子问题, 即取长度比例为 α 的子数组, 就有 $1\approx\alpha^k n\Rightarrow k\approx\log_\alpha\dfrac{1}{n}=-\lg n/\lg a$. 同理, 递归树深度最大的叶结点对应每次都取较大的子问题, 即取长度比例为 $1-\alpha$ 的子数组, 就有 $1\approx(1-\alpha)^k n\Rightarrow k\approx\log_{1-\alpha}\dfrac{1}{n}=-\lg n/\lg(1-\alpha)$.

7.4–5. 当输入数据已经 "几乎有序" 时, 插入排序速度很快. 在实际应用中, 我们可以利用这一特点来提高快速排序的速度. 当对一个长度小于 k 的子数组调用快速排序时, 让它不做任何排序就返回. 当上层的快速排序调用返回后, 对整个数组运行插入排序来完成排序过程. 试证明: 这一排序算法的期望时间复杂度为 $O(nk+n\lg(n/k))$. 分别从理论和实践的角度说明我们应该如何选择 k?

解: 因为修改后的快速排序在对长度小于 k 的子数组不作操作就返回, 因此递归树的高度将减小 $\lg k$, 对应期望时间复杂度为 $O(n(\lg n - \lg k)) = O(n\lg(n/k))$. 而插入排序对快速排序返回的数组排序时, 等价于分别对 n/k 个长度为 k 的子数组进行插入排序, 对应时间复杂度为 $nO(k^2)/k = O(nk)$. 因此, 总的时间复杂度为 $O(nk + n\lg(n/k))$.

从理论的角度, 要让 $nk + n\lg(n/k)$ 尽可能小. 对 k 取导数得到 $n(1 - 1/k\ln 2) = 0$, 取 $k = 1/\ln 2$ 时能得到最小值. 从实践的角度, 排序算法的快慢不仅需要考虑复杂度, 还要考虑存储层次等其他因素, 因此 k 的选取需要经过实验确定.

8.1–4. 假设现有一个包含 n 个元素的待排序序列. 该序列由 n/k 个子序列组成, 每个子序列包含 k 个元素. 一个给定子序列中的每个元素都小于其后继子序列中的所有元素, 且大于其前驱子序列中的所有元素. 因此, 对于这个长度为 n 的序列的排序转化为对 n/k 个子序列的 k 个元素的排序. 试证明: 这个排序问题中所需比较次数的下界是 $\Omega(n\lg k)$. (*提示:* 简单地将每个子序列的下界进行合并是不严谨的.)

解: 输入数据可能的排列数有 $(k!)^{n/k}$ 种, 因此一个比较排序算法的比较次数 h 满足

$$(k!)^{n/k} \leqslant 2^h \Rightarrow h \geqslant \frac{n}{k}\lg k! = \frac{n}{k}\Omega(k\lg k) = \Omega(n\lg k) \tag{4.1}$$