关于堆排序的一种新型算法*

吴晓红
(广东中山学院计算机系，中山，528403)
郭改霞
(长沙交通学院计算机系，长沙，410076)

ON A NEW ALGORITHM OF HEAPSORT

Wu Xiao-hong
(*Department of Computer Engineering, Zhongshan College, Zhongshan, Guangdong,* 528403, *P.R.China*)
Guo Gai-xia
(*Department of Computer Engineering, Changsha Communications University, Changsha,* 410076, *P.R.China*)

Abstract

In this paper, a new algorithm of heapsort, called second order heapsort, is suggested. It is proved that the average time complexity of the algorithm is $O(n\log_2 n)$ and the speed of the algorithm, which has used optimum data processing method, is heightened 180%.

Key words: second order heapsort, optimum data processing, algorithm

§1. 引　言

排序技术无论在系统软件还是应用软件中都有很高的使用频度. 它既是计算机算法设计的基本理论之一，也是计算机应用中的一项重要技术. 对基于比较的一类排序法，理论上已证明其算法的时间复杂度下界为 $n\log_2 n$[1]. 要提高排序的效率不但要采用时间复杂度为 $O(n\log_2 n)$ 的最佳排序算法，而且在排序过程中，也要设法采用优化数据处理技术，减小数据操作中的时间耗费. 通过最佳排序算法与优化数据处理技术的结合，获取高效的排序效果. 本文对时间复杂度为 $O(n\log_2 n)$ 的堆排序算法，通过对文献 [2] 中 Heapsort1 算法的分析，在平均时间复杂度为 $O(n\log_2 n)$ 的基础上，提出了二次堆排序算法，即把 n 个数据的堆排序过程，划分为多个较小的堆排序过程. 使数据操作在较小的存储空间范围内进行，避免了由于远距离存储空间之间数据操作所引起的时间耗费. 使数据得到了优化处理，取得理想的排序效果.

* 1998 年 12 月 15 日收到.

§ 2. Heapsort1 算法分析

Heapsort1 的算法设计思想是在堆排序的重建过程中，当把堆顶元素取走后，由于能充当新的堆顶数据的必是其左、右孩子中的一个，只需作一次比较就可使大者上升一层. 重复这个过程，直至叶结点，再将当前堆的最后一个叶结点数据 (其空出的位置以存放堆顶数据) 附在当前所缺的叶结点上. 这个新附的叶结点有可能上升，这只要通过逐次与新结点作一次比较来实现. 根据算法思想，设堆的高度为 h, 则有：

引理. Heapsort1 算法的重新建堆过程中，附在所缺的叶结点上的新结点，可能上升过程中的平均比较次数不超过 2 次.

证明. 由于附在所缺叶结点上的新结点是堆的叶结点，由堆的特性可知，该结点比其新结点大而上升 1 层的概率 $p_1 \le \frac{1}{2}$, 同理，该结点比其新结点大的概率 $p \le \frac{1}{2}$. 由条件概率可知，该结点上升 2 层的概率 $p_2 \le p = \frac{1}{2^2}$. 同理推得，该结点上升 i 层的概率 $p_i \le \frac{1}{2^i}(1 \le i \le h-1)$. 每上升一次需增一次比较，可得新结点上升过程中的平均比较次数的数学期望

$$s \le \sum_{i=1}^{h-1} \frac{1}{2^i} < 1. \tag{1}$$

无论新结点是否上升，均需进行一次比较，所以新结点上升过程中的平均比较次数为 $s+1 < 2$. 证毕.

定理 1. n 个数据 Heapsort1 算法的平均时间复杂度为

$$T(n) = 2n + \sum_{i=1}^{n-1}(\lfloor \log_2 i \rfloor + 2). \tag{2}$$

证明. Heapsort1 算法重建过程中，把堆顶数据取走后，只需作一次比较就可使大者上升一层. 重复这个过程，直至叶结点，共需比较 $h = \lfloor \log_2 i \rfloor$ 次 (i 为当前堆的结点数). 由引理可知，n 个数据重新建堆过程的比较次数为

$$s = \sum_{i=1}^{n-1}(\lfloor \log_2 i \rfloor + 2). \tag{3}$$

而 n 个数据建立初始堆的比较次数为 $2n$[3], 所以 Heapsort1 算法的平均时间复杂度为 $T(n) = s + 2n = 2n + \sum_{i=1}^{n-1}(\lfloor \log_2 i \rfloor + 2)$. 证毕.

定理 2. $T(n) = O(n \log_2 n)$.

证明. 由定理 1, 可有

$$\begin{aligned} T(n) &\le 2n + \sum_{i=1}^{n-1}(\lfloor \log_2 i \rfloor + 2) \\ &= 4n - 2 + (n-1)\lfloor \log_2 n \rfloor. \end{aligned} \tag{4}$$

可见当 $n \ge 16$ 时，便有

$$T(n) \le 2n\lfloor \log_2 n \rfloor. \tag{5}$$

因而得证定理结论.

由此定理可知 Heapsort1 算法具有较优的平均时间复杂度.

§ 3. 二次堆排序 (TWHEAPSORT) 算法

3.1 算法思想

为实现 n 个数据的排序过程, 可在技术上对数据进行优化处理, 设对数据的地址只作较少操作的存储空间 BUF. BUF 中能存放 k 个数, 把 n 个数据分成 $m=\lceil n/k\rceil$ 个段, 对每一段中的 k 个数据用 Heapsort1 算法 (按开值) 进行排序, 并在每一段的最后一个数据中求得最大值 MAX. 然后对这 m 个有序段, 各取其第一个数据, 把这 m 个数据, 用 Heapsort1 算法进行第二次堆排序, 其中, 堆顺序为新结点数据不大于孩子结点数据. 每当把堆顶数据取走后, 取该堆顶数据所在段的后续的后续数据放入空缺结点. 若某一段中的数据取完, 则用数据 $MAX+1$ 放入空缺结点, 从堆顶取走的数据, 按次序放入另一存储空间.

3.2 算法描述

算法中使用 3 个存储空间 r, h 和 d, 其中, r 用来存放 n 个被排数据, h 用来存放第二次堆排序的 m 个数据和每一段数据的起止指针, d 用来存放 n 个被排序好了的数据. 对 r 和 n 个数据按升序排序的 TWHEAPSORT 算法描述如下.

```
procedure twheapsort;
    begin
       k←min(⌊BUF/sizeof(r[1])⌋, n); m← ⌈n/k⌉; /* 确定 k 和 m*/
       max ←r[t]; /*max 取初值 */
       i←0
while i< m-1 do /* 对 m-1 个数据段中的数据排序 */
    begin
       ptr←r+i * k /* ptr 指向某一数据段 */
       Heapsort1(ptr,L,k) /* 用 Heapsort1 对 k 数据排序 */
       d[i]←h[1].key; /* 转储堆顶数据 */
    end;
    if h[1].left<h[1].right then/* 取后续数据和起止指针 */
    begin
       x.left←h[1].left − 1;
       x.key←r[x.left];
       x.right←h[1].right;
    end;
    else x.key← max;
       j←1;
while j≤m do/* 左右子孩子作一次比较, 大者上开一层, 进行 ⌊log2 m⌋ 次 */
    begin
       h[j]←h[s];
```

```
        j←s;
    end;
s← ⌊j/2⌋;
while s≥1 and h[s].dey>x.key do
    begin /* 新附结点上升 */
        h[j]←h[s];
        j←s;
        s← ⌊s/2⌋;
    end;
        h[j]←x;
        j←i+1;
    end /* 算法结束时 d[1]-d[n] 为排序好了的数据 */
```

3.3 算法分析

定理 3. n 个数据的 TWHEAPSORT 算法的平均时间复杂度为

$$T(n,m)=2n+m\sum_{i=1}^{k-1}(\lfloor\log_2 i\rfloor+2)+2m+\sum_{j=1}^{m-1}(\lfloor\log_2 j\rfloor+2), \tag{6}$$

这里 $n=mk$.

证明. 根据前述 TWHEAPSORT 算法的思想, 我们将 n 个数据分成 m 个数据段, 每段中含 k 个数据, 因而其时间复杂度 $T(n,m)$ 由两个部分组成:

(1) 对含有 k 个数据的 m 个数据段, 用 Heapsort1 算法进行排序, 其平均时间复杂度为 T_1. 由定理 1 可得

$$T_1=m\{2k+\sum_{i=1}^{k-1}(\lfloor\log_2 i\rfloor+2)\}=2n+m\sum_{i=1}^{k-1}(\lfloor\log_2 i\rfloor+2). \tag{7}$$

(2) 对 m 个有序数据段的第一个数据再次用 Heapsort1 算法进行排序, 其时间复杂度为

$$T_2=2m+\sum_{j=1}^{m-1}(\lfloor\log_2 j\rfloor+2), \tag{8}$$

因而有

$$T(n,m)=T_1+T_2=2n+m\sum_{i=1}^{k-1}(\lfloor\log_2 i\rfloor+2)+2m+\sum_{j=1}^{m-1}(\lfloor\log_2 j\rfloor+2).$$

证毕.

类似定理 2, 我们也可得到 (证明从略).

定理 4. 固定 m, 则有 $T(n,m)=O(n\log_2 n)$.

进一步, 我们还可以得到:

定理 5. 固定 $m>1$, 则有

$$T(n)-T(n,m)\to\infty(n\to\infty). \tag{9}$$

证明. 根据定理 1 和定理 3, 有

$$T(n)-T(n,m)=\sum_{i=k}^{n-1}\lfloor\log_2 i\rfloor-(m-1)\sum_{i=1}^{k-1}\lfloor\log_2 i\rfloor-2m-\sum_{j=1}^{m-1}\lfloor\log_2 j\rfloor$$
$$>\sum_{i=n-m+1}^{n-1}\lfloor\log_2 i\rfloor-2m-\sum_{j=1}^{m-1}\lfloor\log_2 j\rfloor. \tag{10}$$

注意到 m 是固定的, 令 $n\to\infty$, 便易得定理结论.

上述两个定理告诉我们, TWHEAPSORT 算法的平均时间复杂度的阶数并不比 Heapsort1 降低, 但随着数据量的增加, 前者明显比后者越来越节省时间. 这是因为前者采用了优化的数据处理技术, 即把 n 个数据的堆排序, 划分成多个较小的堆的排序过程, 从而使数据操作在较小的存储空间范围内进行. 避免了由于近距离存储空间之间数据操作所引起的时间耗费, 使排序过程时间较少, 提高了排序的效率. 下一节的实验数据将进一步证实这里的结论.

§ 4. 实验与结论

为验证 TWHEAPSORT 算法优化数据处理技术的作用. 我们在 586 微机上用 Heapsort1 算法和 TWHEAPSORT 算法分别对相同的几组数据进行排序, 在使用 TWHEAPSORT 算法时, 取 $m=20$. 将测得的有关数据列于表 1, 其中 t_1 为 Heapsort1 算法所耗费的时间, t 为 TWHEAPSORT 算法所耗费的时间.

表 1

数据量 n	$T(n)$	$T(n,20)$	t_1	t
1000	11975	7848	1.110	0.319
2000	25951	17568	2.396	0.830
3000	40903	27988	3.731	1.324
4500	63807	44488	5.824	2.011
6000	87807	61868	7.984	2.808

实验结果说明了上述结论是正确的. TWHEAPSORT 算法的排序速度比 Heapsort1 提高 180%. 这主要得益于 TWHEAPSORT 算法在数据处理过程中, 减少了时间耗费, 使排序运行在最佳状态.

可见, 排序效率不但取决于算法的优劣, 而且算法的实现技术也很重要 [4]. 本文就是通过对其实现过程中数据处理优化而改进了算法的排序效率.

参 考 文 献

[1] D. E 克努著, 管纪文等译, 计算机程序设计技巧, 国防工业出版社, **8**(1984), 119–159.
[2] 顾洲, 渚宇章, 堆整序的改进算法及其复杂性分析, 计算机学报, B **4**(1990).
[3] 卢开澄, 算法与复杂性, 高等教育出版社, **5**(1995), 155–160.
[4] 唐开山, 大数据量处理中的 c 指针及运算效率, 微计算机应用, **18**:1(1997), 40–42.