算法设计与分析作业三

作者：吴润泽　　学号：181860109

Email： 181860109@smail.nju.edu.cn

2020 年 3 月 18 日

目录

CHAPTER 8

problem 8.2

算法 假设有 5 个数 a，b，c，d，e

1. 比较 a b，将较小者放入 a，较大者放入 b
2. 比较 c d，将较小者放入 c，较大者放入 d
3. 比较 a c，将较小者放入 a，较大者放入 c，若 a 和 c 发生交换则同时交换 b 和 d，即保证 a b 和 c d 原有大小关系不变。此时有 $a < c < d, a < b$，则 a 不可能是中位数
4. 比较 b e，将较小者放入 b，较大者放入 e
5. 比较 b c，将较小者放入 b，较大者放入 c，若 b 和 c 发生交换则同时交换 d 和 e，即保证 b e 和 c d 原有大小关系不变。此时有 $b < c < d, b < e$，则 b 不可能是中位数
6. 比较 c e，将较小者放入 c,较大者放入 e,此时有 $c < d < e$,且 $c > b, c > a$，即 c 就是中位数。

problem 8.4

算法 设数组 array 元素个数为 n，不妨设 n 为奇数，阶为 k 的元素为 m；

1. 当 k 等于 $\frac{n+1}{2}$ 时，直接调用算法 A，即可找到 m；
2. 当 k 小于 $\frac{n+1}{2}$ 时，遍历数组 array，记录其元素最小值为 min，对于原数组 array，有 $n-k$ 个元素大于 m，$k-1$ 个元素小于 m。
3. 开辟新数组 temp，将前 $n-2k+1$ 个元素值赋为 min-1，并将 array 的 n 个元素放入其后。则数组 temp 中有 $n-k$ 个元素大于 m，$n-k$ 个元素小于 m。因此 m 为 temp 中位数，调用算法 A，即可找到 m。
4. 当 k 大于 $\frac{n+1}{2}$ 时，同理记录其元素最大值为 max，开辟数组 temp，在数组 temp 中前 $2k-n-1$ 个赋值为 max+1。同样使得 m 变为 temp 中位数，调用算法 A，即可找到 m。

算法 1 $KthFind$ 算法

Function $KthFind\ (array, n, k)$

  if $k == \dfrac{n+1}{2}$ **return** $A(array, n)$

  $min\_num := min(array, n), max\_num := max(array, n)$

  Let $temB[1...2n]$ *be new array.*

  for $i := 1$ *to* n **do**

    $temB[i] := A[i]$

  if $k < \dfrac{n+1}{2}$ **then**

    for $i := 1$ *to* $n-2k+1$ **do**

      $temB[i+n] := min\_num - 1$

    return $A(temB, 2n-2k+1)$

  if $k > \dfrac{n+1}{2}$ **then**

    for $i := 1$ *to* $2k-n-1$ **do**

      $temB[i+n] := max\_num - 1$

    return $A(temB, 2k-1)$

时间复杂度 寻找最值和开辟新数组的时间复杂度为 $O(n)$，添加的元素个数最多为 $n-1$ 个，数组的规模仍为 $O(n)$，而找中位数算法 A，时间复杂度为 $O(n)$，因此总的时间复杂度仍为线性。

problem 8.5

(1)

使用归并排序进行降序排列，排序结果前 k 个元素即为所求。
归并排序时间复杂度为 $O(n\log n)$，输出复杂度为 $O(k)$，总时间复杂度为 $O(n\log n)$.

1.**Function** $KthOrder1$ $(array, n, k)$
2. 对原数组使用归并排序，并按照升序排列
3. **for** $i := 1$ *to* k **do**
4. $res.add(array[i])$
5. **return** res

(2)

根据原数组建立最大堆，最大堆堆顶存储当前堆的最大值，则弹出堆顶元素 k 次，k 个元素即为所求。
根据已有序列建堆的时间复杂度为 $O(n)$，弹出堆顶元素 k 次，每次修复堆时间复杂度为 $O(\log n)$，则总的时间复杂度为 $O(n + k\log n)$.

1.**Function** $KthOrder2$ $(array, n, k)$
2. 根据原数组进行建堆，得到最大堆为 heap
3. **for** $i := 1$ *to* k **do**
4. $res.add(heap.top())$
5. $heap.pop()$\\弹出堆顶元素，并修复
6. **return** res

(3)

利用 problem 8.4 中 Kth-find 算法，找到原数组中的第 k+1 大元素 m。遍历原数组，找到其中大于 m 的元素加入结果数组 res。对 res 使用归并排序，升序排列，得到 res 即为所求。
找第 k+1 大，遍历原数组为线性时间 $O(n)$，对 res 归并排序 $O(k\log k)$，总

时间复杂度为 $O(n + k \log k)$.

Function $KthOrder3$ $(array, n, k)$

  $m := KthFind(array, n, k+1)$\\找到数组第k+1大元素

  for $i := 1$ *to* n **do**

    if $array[i] > m$ **do** $res.add(array[i])$

  对 res 数组使用归并排序，并按照升序排列

  return res

problem 8.6

(1)

使用归并排序进行排列时间复杂度为 $O(n\log n)$，遍历中位数 M 左右两侧元素，与 M 差值最小的加入 res，并移动对应侧指针，直至 res 个数为 k 即可。总时间复杂度为 $O(n\log n + k)$.

1.**Function** $KthNear1$ $(array, n, k)$

2. 对原数组使用归并排序

3. $l := \frac{n+1}{2} - 1,\ r := \frac{n+1}{2} + 1$

4. **for** $i := 1$ *to* k **do**

5. **if** $array[l] + array[r] > 2M$ **do** $res.add(array[r++])$

6. **else** $res.add(array[l--])$

7. **return** res

(2)

利用 problem8.4 中的查找中位数算法，得到中位数为 M，时间复杂度为 $O(n)$。 将原数组分为大于 M 和小于 M 的两数组 L,S，时间复杂度为 $O(n)$。 与 problem 8.5(3) 同样思想，找到 L 的前 k 小 (查找第 k+1 小，即第 n-k-1 大)LK 和 S 的前 k 大元素 SK，时间复杂度为 $O(k)$。对 LK 进行归并排序按照升序排列，对 SK 进行归并排序按照降序排列，时间复杂度为 $O(k\log k)$。之后遍历 LK 和 SK 找到与 M 最接近的 K 个元素即可。

Function $KthNear2\ (array, n, k)$
　$M := A(array, n)$\\找到中位数 M
　划分原数组为大于 M 和小于 M 两部分:L[1..n1],S[1..n2]
　$large := KthFind(L, n1, n1-k-1)$\\找到L数组第k+1小元素
　$small := KthFind(S, n2, k+1)$\\找到L数组第k+1大元素
　找到 L 数组小于 large 的 k 个元素, 并升序排列, 得到 LK
　找到 S 数组大于 small 的 k 个元素, 并降序排列, 得到 SK
　$l := 0,\ r := 0$
　for $i := 1$ *to* k **do**
　　if $LK[l] + SK[r] > 2M$ **do** $res.add(SK[r++])$
　else $res.add(LK[l--])$
　return res

problem 8.8

算法设计　使用两个堆，大根堆 q1 维护较小值，小根堆维护较大值，令大根堆 q2 元素个数为 m，小根堆元素个数为 n：

使得小根堆的堆顶是较大数中最小的，大根堆的堆顶是较小数中最大的；

将大于大根堆堆顶的数放小根堆，小于等于大根堆堆顶的数放大根堆；

对于大根堆的堆顶元素，有 n 个元素比该元素大，$m-1$ 个元素比该元素小；

对于小根堆的堆顶元素，有 m 个元素比该元素小，$n-1$ 个元素比该元素大；

在维护 $|m-n| \leq 1$ 之后，当 $m = n$ 时，两堆顶元素均为中位数，当 $m \neq n$ 时，元素个数较多的堆顶元素即为当前中位数；

易知插入和删除的时间复杂度均为 $O(\log n)$，查找中值为常数。

查找中值

　if $(q1.size() + q2.size())\%2 == 1$ **then**
　　if $q1.size() > q2.size()$ **do** $mid := q1.top()$
　　else $mid := q2.top()$
　else $mid := (q1.top + q2.top())/2$

插入操作

1. **if** $input > q1.top()$ **do** $q2.push(input)$
2. **else** $myq1.push(input)$\\\\大根堆放较小数，小根堆放较大数
3. **while** $|q1.size() - q2.size()| > 1$ **do**
4. **if** $q1.size() > q2.size()$ **do** $q2.push(q1.pop())$
4. **else** $q1.push(q2.pop())$

删除操作

1. **if** $(q1.size() + q2.size())\%2 == 1$ **then**
2. **if** $q1.size() > q2.size()$ **do** $q1.pop()$
3. **else** $q2.pop()$
4. **else** $q2.pop()$\\\\当为偶数时任意弹出一个

problem 8.9

(1)

对中位数 x_k，设 n 为奇数，有 $\frac{n+1}{2} - 1$ 个元素小于 x_k，$\frac{n+1}{2} - 1$ 个元素大于 x_k，则 $\sum_{x_i > x_k} \frac{1}{n} = \sum_{x_i < x_k} \frac{1}{n} = \frac{1}{2} - \frac{1}{2n} < \frac{1}{2}$. 对 n 为偶数，同样成立。

(2)

建立以 (w,x) 为元素的结构体数组 ori，w 为权重，x 为其对应下标。对其进行归并排序，时间复杂度为 $O(n \log n)$。遍历 ori，对权重进行累加，当权重和大于 $\frac{1}{2}$ 时，对应结构体元素的 x 即为加权中位数，时间复杂度为 $O(n)$。因此总的时间复杂度为 $O(n \log n)$。

```
Function KthNear1 (ori, n)
    对结构体数组 ori 使用归并排序
    cur_weight := 0\\当前权重
    for i := 1 to n do
        if cur_weight + ori[i].w > 1/2 do
            return ori[i].x
        else \\更新权重和权重
            cur_weight := cur_weight + ori[i].w
```

(3)

参考 BFRPT 算法 假设划分的权值和为 tar, 初始 $tar = \frac{1}{2}$

1. 将所有元素分成 $\lceil \frac{n}{5} \rceil$ 组，每组 5 个元素
2. 根据problem 8.2 可知，比较 6 次找到 5 个元素中位数
3. 递归使用 BFRPT 来找出各组中位数的中位数，记为 m
4. 基于 m 对元素进行划分，假设有 x 个元素小于 m，n-x-1 个元素大于 m
5. 计算 x 个元素的权值和 T，若权值和 T=tar，则 m 为加权平均数
6. 若权值和 T>tar, 则在 x 个元素中递归寻找中位数，tar 值不变
7. 若权值和 T<tar, 则在 n-x-1 个元素中递归寻找中位数，tar=tar-T
8. 时间复杂度 $T(n) = T(n/5) + T(7n/10) + O(n) = \Theta(n)$，满足要求。

CHAPTER 9

problem 9.4

problem 9.6

problem 9.8

problem 9.12