算法设计与分析 Week 2

第二次作业-solution

Lecturer: 杨启哲 Last modified: 2025 年 10 月 9 日

1. 当输入由下列区间中的 n 个正整数组成时，以 n 为大小说明算法 $\mathrm{RadixSort}$ 的时间复杂性。

   - $[1, \ldots, n^2]$.
   - $[1, \ldots, 3^n]$.
   - $[1, \ldots, 2^{n!}]$.

   解答. 只需要计算至多需要多少位表示数即可。

   (1) 当数据范围在 $1 \sim n^2$ 时，其用来表示的数位至多为 $\lceil 2 \log n \rceil$，从而算法的时间复杂性为 $O(n \log n)$.
   (2) 当数据范围在 $1 \sim 3^n$ 时，其用来表示的数位至多为 $\lceil \log 3 \rceil n$，从而算法的时间复杂性为 $\Theta(n^2)$.
   (3) 当数据范围在 $1 \sim 2^{n!}$ 时，其用来表示的数位至多为 $n!$，从而算法的时间复杂性为 $\Theta(n \cdot n!)$.

   □

2. 用归纳法设计一个递归算法，求在 $A[1, \ldots, n]$ 中 n 个实数的平均值。

   解答. 假设 $A[1, \ldots, n]$ 中前 k 个实数的平均值为 $Ave(k)$，则有:

   $$Ave(k) = \frac{k-1}{k} Ave(k-1) + \frac{1}{k} A[k] = Ave(k-1) + \frac{1}{k}(A[k] - Ave(k-1))$$

   我们可以据此设计一个简单的递归算法:

   算法: 计算 n 个实数的平均值

   输入: 数组 $A[1, \ldots, n]$
   输出: n 个实数的平均值

   ```
   Result ← 0
   for i = 1 to n do
       Result ← Ave(k − 1) + (1/k)(A[k] − Ave(k − 1))
   end for
   return Result
   ```

   □

Remark 0.1

我直接改写成了 For 循环的形式，出这道题的目的是希望大家有一个对归纳法的基本了解，即解决一个输入规模为 n 的问题时，可以通过解决输入规模为 $n-1$ 的问题来解决。本题所希望完成的思路就是通过前 k 个的平均数来计算前 $k+1$ 个的平均数。一般而言，这样的思路在设计递归算法时是非常有用的，因为其往往蕴含了正确性的证明。

3. 对于一个含有 n 个元素的整数数组，记 M 是其中元素的最大与最小值之差，即：

$$M = \max_i A[i] - \min_i A[i].$$

请给出一个 $O(n+M)$ 时间的算法对该数组进行排序。

解答. 我们构造如下的一个算法：

算法: 特殊的排序

输入: 数组 $A[1,\dots,n]$
输出: 排好序的数组

1: 遍历一遍 A，找出 $\min_i A[i]$ 和 $\max_i A[i]$
2: 计算 $M = \max_i A[i] - \min_i A[i]$
3: 准备 $M+1$ 个空表 $B_0, B_1, \dots, B_M$
4: **for** $i = 1$ to n **do**
5: 　　将 $A[i]$ 放进 $B_{A[i]-\min_i A[i]}$ 中
6: **end for**
7: 将 $B_0, B_1, \dots, B_M$ 中的数按顺序放进数组 A 中
8: **return** A

注意到，在整个算法过程中：

(1) 第一步寻找最大值和最小值需要 $O(n)$ 的时间。
(2) 第四步的循环需要 $O(n)$ 的时间。
(3) 第七步的构造排好序的数组 A 需要遍历所有的表，因此需要 $O(M)$ 的时间。

因此该算法的时间复杂性为 $O(n+M)$. □

Remark 0.2

该问题又展示了一个不需要通过比较来进行排序的例子，因此该算法的复杂度会跟数据的范围有关。

4. 给定一个含有 n 个元素的数组，注意到数组中的某些元素是重复的，即这些元素在数组中出现不止一次。给出一个算法，移除数组中重复的元素，要求算法的时间复杂度为 $O(n \log n)$。

解答. 我们可以先对数组进行排序，然后再扫描一遍数组，将重复的元素移除。具体来说，假设排序后的数组为 $A'[1,\ldots,n]$，我们可以维护一个指针 i,k，其初始值分别为 $1,2$，然后从左到右扫描数组：

- 当 $A'[i] \neq A'[k]$ 时，我们将 $A'[i+1]$ 赋值为 $A'[k]$，并将 i,k 向右移动一位。
- 当 $A'[i] = A'[k]$ 时，我们将 k 向右移动一位，

算法在 $k > n$ 时结束，并返回此时 $A'[1,\ldots,i]$ 作为处理好的无重复元素的数组。最终，数组的前 k 个元素即为移除重复元素后的结果。算法的伪代码如下：

算法: 移除重复元素

输入: 数组 $A[1,\ldots,n]$
输出: 移除重复元素后的数组

```
对数组 A 进行排序，得到排序后的数组 A'
i ← 1, k ← 2
for k = 2 to n do
    if A'[i] ≠ A'[k] then
        A'[i+1] ← A'[k]
        i ← i+1, k ← k+1
    else
        k ← k+1
    end if
end for
return A'[1,...,i]
```

该算法的时间复杂性为：

(1) 排序需要 $O(n\log n)$ 的时间。
(2) 扫描数组需要 $O(n)$ 的时间。

因此总的时间复杂性为 $O(n\log n)$. □

5. 请给出一个 $O(n)$ 的算法，其可以将 n 个 0 到 n^3-1 的数进行排序。

(Hint: 考虑一下 n 进制？)

解答. 我们可以构造一个 n 进制的基数排序。注意到 n^3-1 的数用 n 进制表示，其最多有 3 位，因此我们可以据此构造一个 3 轮的排序。算法伪代码如下：

算法: 排序

输入: 数组 $A[1,\ldots,n]$，其中每个数的范围在 $0 \sim n^3-1$
输出: 排好序的数组

```
将 A 放进列表 L 中
```

2: **for** $i = 1$ to 3 **do**
3: 准备 n 个空表 $B_0, B_1, \ldots, B_{n-1}$
4: **while** L 不为空 **do**
5: 将 L 中的数 x 取出
6: 将 x 放进 B_k 中，其中 k 是 x 在 n 进制下的第 i 位数字，即：$k = \frac{x \mod n^i}{n^{i-1}}$.
7: **end while**
8: 将 $B_0, \ldots B_{n-1}$ 中的数按顺序放进 L 中
9: **end forreturn** L

□

6. 设 $A[1, \ldots, n]$ 和 $B[1, \ldots, n]$ 是两个已按升序排列的互不相同的整数所组成的数组，给出一个有效的算法找出在 A 和 B 中第 $\frac{n}{4}$ 小的整数，这里不妨认为 n 是 4 的倍数，你的算法的运行时间是多少?

解答. 一个非常直接的算法是直接将两个数组合并，然后取出第 $\frac{n}{4}$ 个数。该算法的时间复杂性为 $O(n)$。这里我们尝试设计一个 $O(\log n)$ 的算法，特别的我们做一下推广，将寻找第 $\frac{n}{4}$ 小的数推广为寻找第 k 小的数。

算法的基本思路是通过二分来确定具体的位置。具体来说，考虑数组 $A[\frac{n}{2}]$，通过二分可以找到其在 $B[n]$ 中的具体位置，比如 $B[j] < A[\frac{n}{2}] < B[j+1]$.，此时我们知道 $A[\frac{n}{2}]$ 在这 $2n$ 个数中排在第 $\frac{n}{2} + j$ 的位置，此时可以对比 $\frac{n}{2} + j$ 和 k 的大小:

- 若 $\frac{n}{2} + j = k$，则 $A[\frac{n}{2}]$ 即为所求。
- 若 $\frac{n}{2} + j < k$，则所求的数在 $A[\frac{n}{2} + 1, \ldots, n]$ 和 $B[j+1, \ldots, n]$ 中，我们可以递归地在这两个子数组中寻找第 $k - (\frac{n}{2} + j)$ 小的数。
- 若 $\frac{n}{2} + j > k$，则所求的数在 $A[1, \ldots, \frac{n}{2} - 1]$ 和 $B[1, \ldots, j]$ 中，我们可以递归地在这两个子数组中寻找第 k 小的数。

该算法的伪代码如下:

算法: 寻找第 k 小的数

输入: 数组 $A[1, \ldots, n]$ 和 $B[1, \ldots, n]$，以及整数 k
输出: A 和 B 中第 k 小的数
1: $\mathrm{Find}(A[1, \ldots, n], B[1, \ldots, n], k)$
过程: $\mathrm{Find}(A[1, \ldots, n], B[1, \ldots, m], k)$
2: **if** $k = 1$ **then**
3: **return** $\min(A[1], B[1])$
4: **end if**
5: **if** $n < m$ **then**
6: 交换 A 和 B, $mid \leftarrow \frac{m}{2}$
7: **else**

```
    mid ← n/2
end if
在 B 中二分查找 A[mid] 的位置，设为 j，即 B[j] < A[mid] < B[j + 1]
if mid + j = k then
    return A[mid]
else if mid + j < k then
    return Find(A[mid + 1, ..., n], B[j + 1, ..., m], k − mid − j)
else
    return Find(A[1, ..., mid − 1], B[1, ..., j], k)
end if
```

该算法的时间复杂性为 $O(\log n)$，因为每次递归时，数组 A 和 B 的规模都至少减了 $\frac{1}{4}$。 □