算法设计与分析 Week 2

第二周作业-Solution

Lecturer: 杨启哲 Last modified: 2024 年 11 月 14 日

截止日期 2023 年 9 月 25 日晚 24：00

1. 用图来说明寻找多数元素的算法 $Majority$ 对下列数组的运算：

   (1) $A = [5, 7, 5, 4, 4]$.

   (2) $A = [2, 4, 1, 4, 4, 4, 6, 4]$.

   解答.

   (1) 对于数组 $A = [5, 7, 5, 4, 4]$，算法的运算过程如下：
   - 第一次 $count = 0$ 时，算法扫到 7，因此算法继续调用 $candidate(3)$.
   - 第二次 $count = 0$ 时，算法扫到 4(第 4 个位置)，因此算法继续调用 $candidate(5)$.
   - 调用 $candidate(5)$ 时，返回 4 作为多数元素的备选，但检查一遍以后发现 4 不是多数元素，因此返回不存在。

   (2) 对于数组 $A = [2, 4, 1, 4, 4, 4, 6, 4]$，算法的运算过程如下：
   - 第一次 $count = 0$ 时，算法扫到 4(第 2 个位置)，因此算法继续调用 $candidate(2)$.
   - 第二次 $count = 0$ 时，算法扫到 4(第 4 个位置)，因此算法继续调用 $candidate(5)$.
   - 调用 $candidate(5)$ 时，$count$ 不会小于 0，因此返回 $A[5]$ 即 4 作为多数元素的备选，并且检查一遍以后发现 4 是多数元素，因此返回 4.

   □

2. 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 $RadixSort$ 的时间复杂性。
   - $[1, \ldots, n^2]$.
   - $[1, \ldots, 2^{2^n}]$.

   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 $1 \sim 2^{2^n}$ 时，其用来表示的数位至多为 2^n，从而算法的时间复杂性为 $\Theta(n \cdot 2^n)$.

   □

3. 请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 $A[1,...,n]$ 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 $A[1,...,n]$
输出: n 个实数的平均值

```
1: Result ← 0
2: for i = 1 to n do
3:     Result ← Ave(k − 1) + (1/k)(A[k] − Ave(k − 1))
4: end for
5: 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。

4. 令 $A[1,...,n]$ 是一个有 n 个整数的已排序的数组，x 是整数。请设计一个 $O(n)$ 时间的算法来判断 A 中是否存在两个数的和等于 x。

解答. 数组已经排好序，因此我们可以维护两个指针，分别指向数组的头和尾，然后不断地向中间移动，直到找到两个数的和等于 x 或者找不到为止。具体来说，初始情况为：

$$A[1] + A[n]$$

其有三种情况：

- 当 $A[1] + A[n] < x$ 时，我们可以将 $A[1]$ 向右移动。
- 当 $A[1] + A[n] > x$ 时，我们可以将 $A[n]$ 向左移动。
- 当 $A[1] + A[n] = x$ 时，我们找到了两个数的和等于 x，返回 True.

其关键在于，假设指针已经移动过一些位置，不妨令当前为 $A[i] + A[j]$，则处在左面的指针不需要再往左移动，处在右面的指针不需要再往右移动，从而整个移动次数不会超过 n。算法的伪代码如下：

算法: 判断是否存在两数和为 n

输入： 数组 $A[1,\dots,n]$ 和整数 x

输出： 是否存在两个数的和等于 x, 若存在返回 True，否则返回 False

```
left ← 1, right ← n
while left < right do
    if A[left] + A[right] = x then
        return True
    else if A[left] + A[right] < x then
        left ← left + 1
    else
        right ← right − 1
    end if
end while
return False
```

我们对算法的正确性再做一些严格的说明：

Lemma 0.2

设第 k 次循环开始时 left 和 right 的值为 i_k, r_k，则对任意的 $j_1 < j_2$，我们有 $i_{j_1} \leqslant i_{j_2} < j_{j_2} \leqslant j_{j_1}$。

解答. 由算法流程自然可得。 □

Lemma 0.3

设第 k 次循环开始时 left 和 right 的值为 i_k, r_k，并且 $A[i_k] + A[j_k] \neq x$，则对任意的 $i < i_k, j > j_k$，由 $A[i] + A[j] \neq x$.

解答. 反设存在 $A[i] + A[j] = x$，则对于 i 存在 $r^{(i)}$ 使得 $A[i] + A[r^{(i)}] < x$，即 $(i, r^{(i)})$ 是某次循环开始时的 left 和 right 的值。同理对于 j 存在 $l^{(j)}$ 使得 $A[l^{(j)}] + A[j] > x$，即 $(l^{(j)}, j)$ 是某次循环开始时的 left 和 right 的值。由上述引理可知，一共存在两个情况：

- $i \leqslant l^{(j)} < j \leqslant r^{(i)}$，则我们有 $A[i] + A[r^{(i)}] > A[i] + A[j] = x$，矛盾。
- $l^j < i < r^{(i)} < j$，则我们有 $A[l^{(j)}] + A[j] < A[i] + A[J] = x$，矛盾。

□

□

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
for i = 1 to 3 do
    准备 n 个空表 B_0, B_1, ..., B_{n-1}
    while L 不为空 do
        将 L 中的数 x 取出
        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 = (x mod n^i) / n^{i-1}.
    end while
    将 B_0, ... B_{n-1} 中的数按顺序放进 L 中
end forreturn L
```

□