

算法设计与分析

第 5 章作业参考答案

作业情况

教材：算法设计技巧与分析 [沙特]M. H. Alsuwaiyel

题目编号：5.7, 5.12, 5.29, 5.31, 补充题

题目范围：归纳法

邮箱：wjyyy1@126.com

习题 5.7

对于下列给出的 8 个数的序列，用图来说明算法 RADIXSORT 的运算。

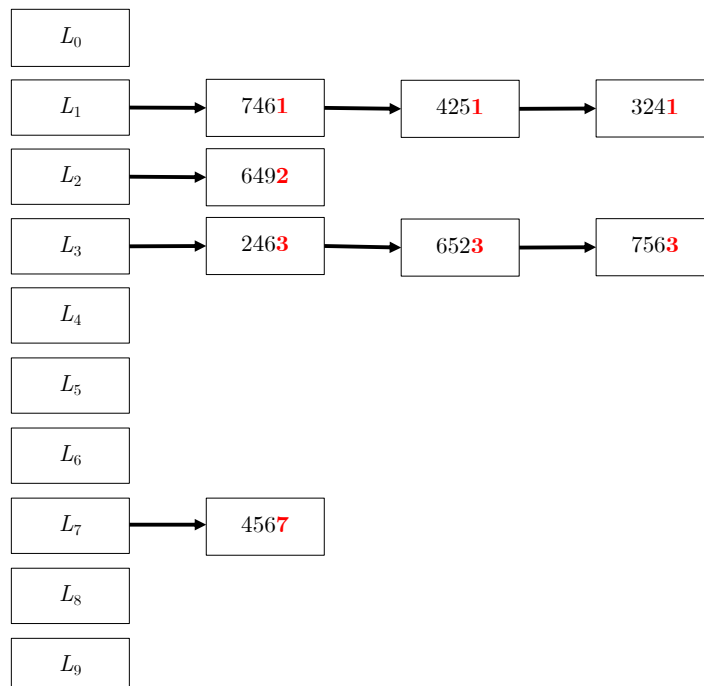
(a) 4567, 2463, 6523, 7461, 4251, 3241, 6492, 7563

(b) 16543, 25895, 18674, 98256, 91428, 73234, 16597, 73195

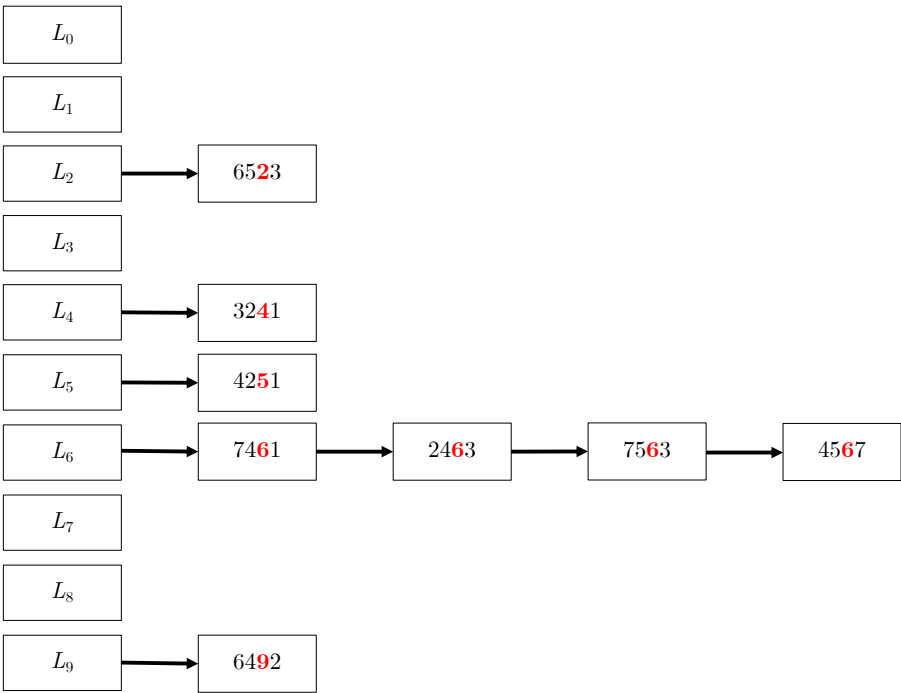
答案：

(a) 共 4 位数字，因此迭代 4 次。

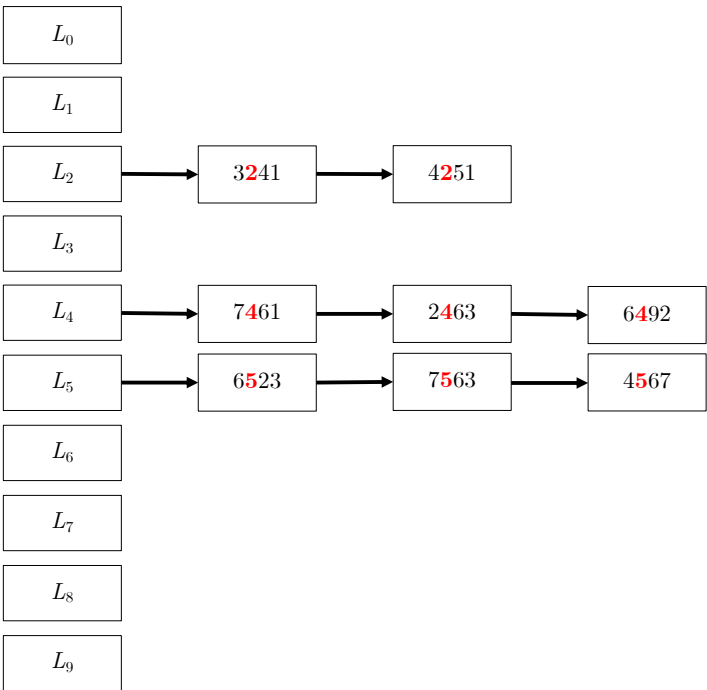
第一次迭代： $L = [4567, 2463, 6523, 7461, 4251, 3241, 6492, 7563]$



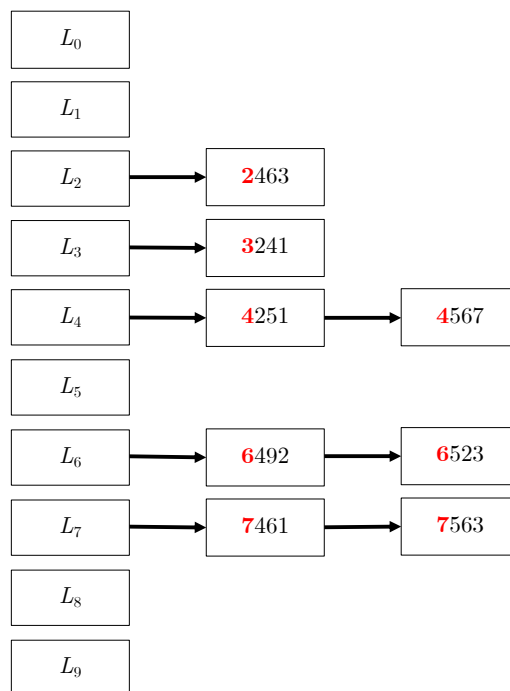
第二次迭代: $L = [7461, 4251, 3241, 6492, 2463, 6523, 7563, 4567]$



第三次迭代: $L = [6523, 3241, 4251, 7461, 2463, 7563, 4567, 6492]$



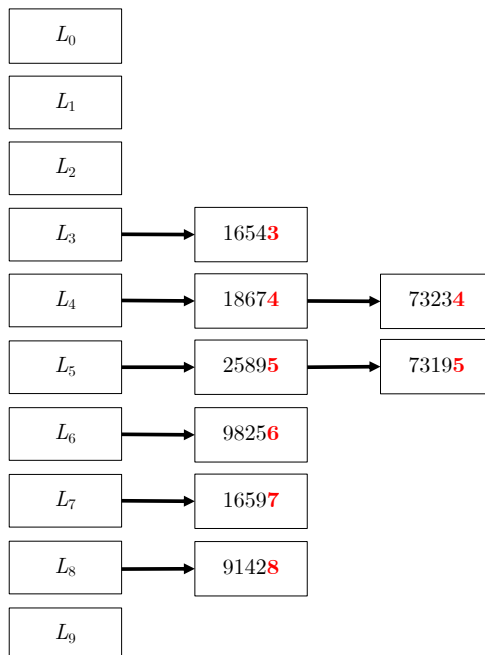
第四次迭代: $L = [3241, 4251, 7461, 2463, 6492, 6523, 7563, 4567]$



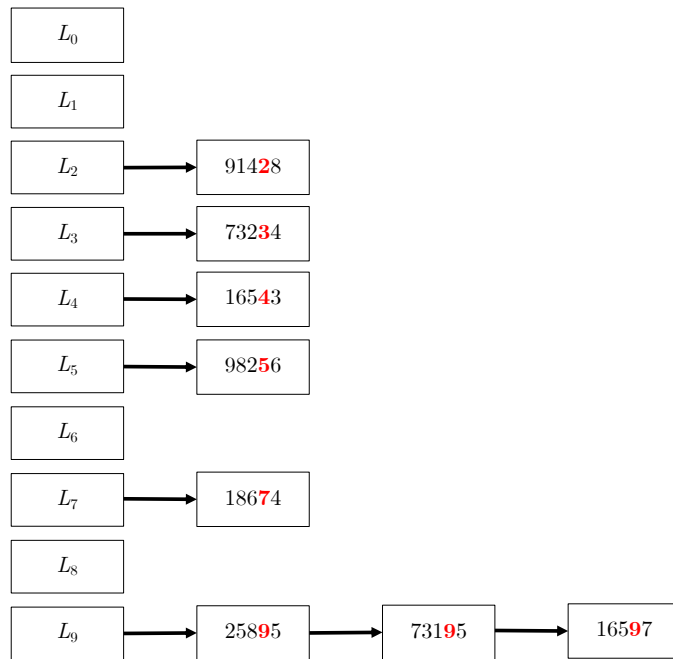
最终非降序排列的 $L = [2463, 3241, 4251, 4567, 6492, 6523, 7461, 7563]$

(b) 共 5 位数字，因此迭代 5 次。

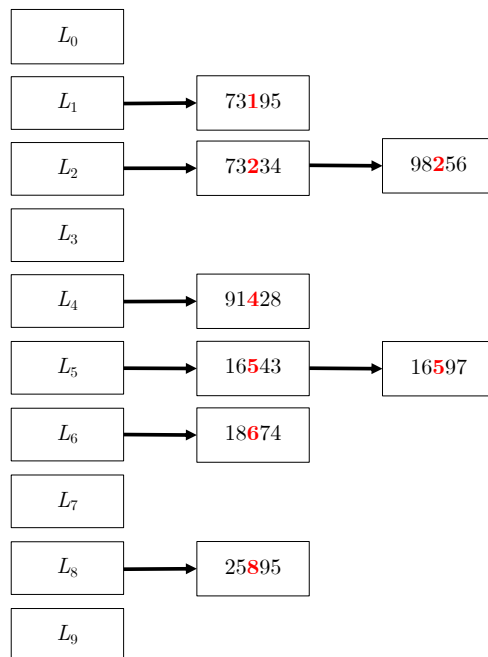
第一次迭代: $L = [16543, 25895, 18674, 98256, 91428, 73234, 16597, 73195]$



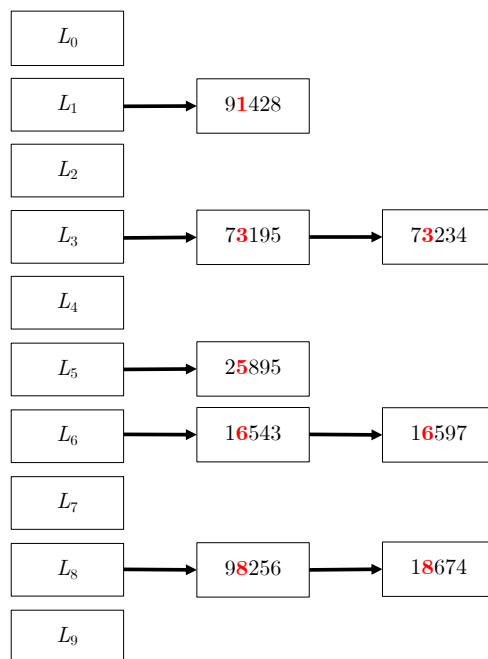
第二次迭代: $L = [16543, 18674, 73234, 25895, 73195, 98256, 16597, 91428]$



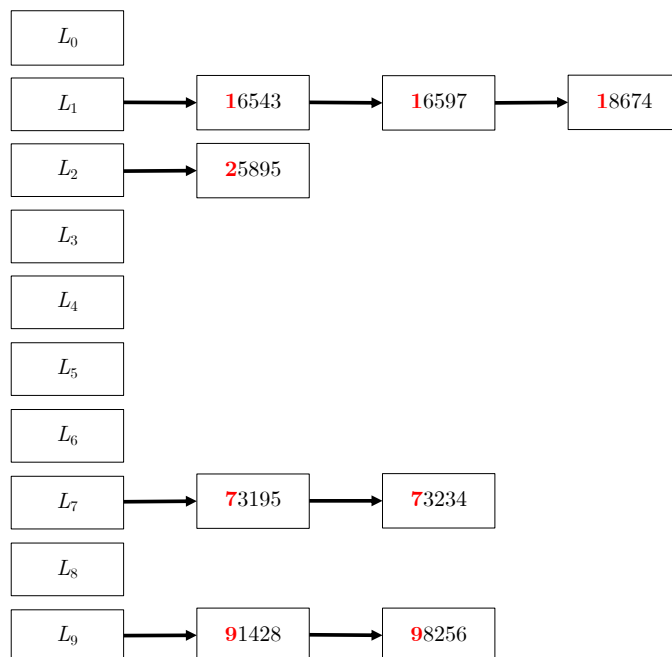
第三次迭代: $L = [91428, 73234, 16543, 98256, 18674, 25895, 73195, 16597]$



第四次迭代: $L = [73195, 73234, 98256, 91428, 16543, 16597, 18674, 25895]$



第五次迭代: $L = [91428, 73195, 73234, 25895, 16543, 16597, 98256, 18674]$



最终非降序排列的 $L = [16543, 16597, 18674, 25895, 73195, 73234, 91428, 98256]$

解析: 表就是链表/列表, 每个表初始时是空的, 对于表中元素, 按照插入顺序进行排列。从低位到高位, 按照当前位从小到大依次完成排序, 就可以在 $O(nk)$ 的时间

内得到结果。多次连接这个表的过程很像蜘蛛纸牌。

习题 5.12

一种称为 *bucket* 桶排序的方法按如下方式工作：令 $A[1 \cdots n]$ 是一个在合理的范围内的 n 个数的序列，例如 $1 \cdots m$ 中所有的数，这里 m 是一个与 n 相比不太大的数，这些数被分到 k 个桶中，第 1 个桶放 1 到 $\lfloor m/k \rfloor$ 间的数，第 2 个桶放 $\lfloor m/k \rfloor + 1$ 到 $\lfloor 2m/k \rfloor$ 间的数，如此等等。现在将每个桶里的数用其他的排序算法排序，如算法 INSERTIONSORT，试分析算法的运行时间。

答案： $O(n^2/k)$ 或 $O(n^2/k + n)$

解析：将 n 个数划分到 k 个桶里所用的时间为 $O(n)$ ；此时第 i 个桶中的所有数都比第 $i+1$ 个桶里所有的数都要小 ($i < k$)。那么这时候我们只需要对每个桶的内部进行排序就可以了。

而由于「 m 是一个与 n 相比不太大的数」，所以每个桶内的数的数量可以认为是 $O(n/k)$ 的规模。而对桶内部的元素排序的时间复杂度为 $O(n^2/k^2)$ ，一共排序 k 次，所以总的复杂度为 $O(n^2/k)$ 。

习题 5.29

用图来说明算法 MAJORITY 对下列数组的运算。

- (a)

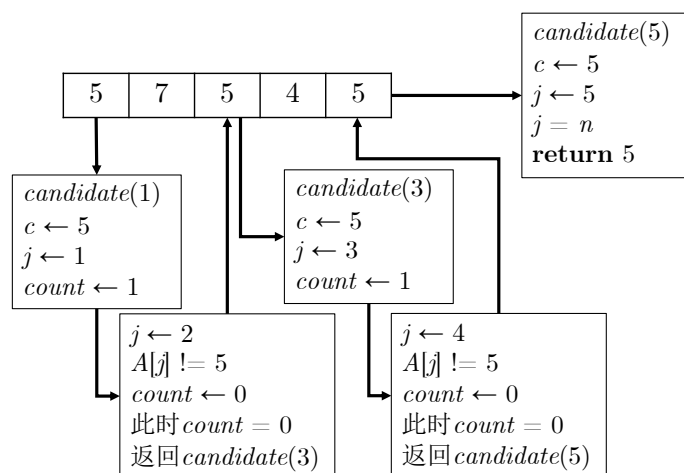
5	7	5	4	5
---	---	---	---	---
- (b)

5	7	5	4	8
---	---	---	---	---
- (c)

2	4	1	4	4	4	6	4
---	---	---	---	---	---	---	---

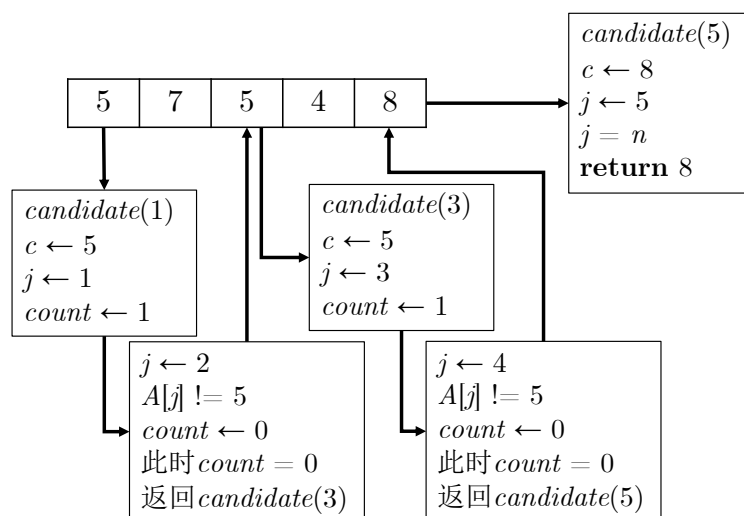
答案：

(a) *candidate*(1) 的过程如图所示：



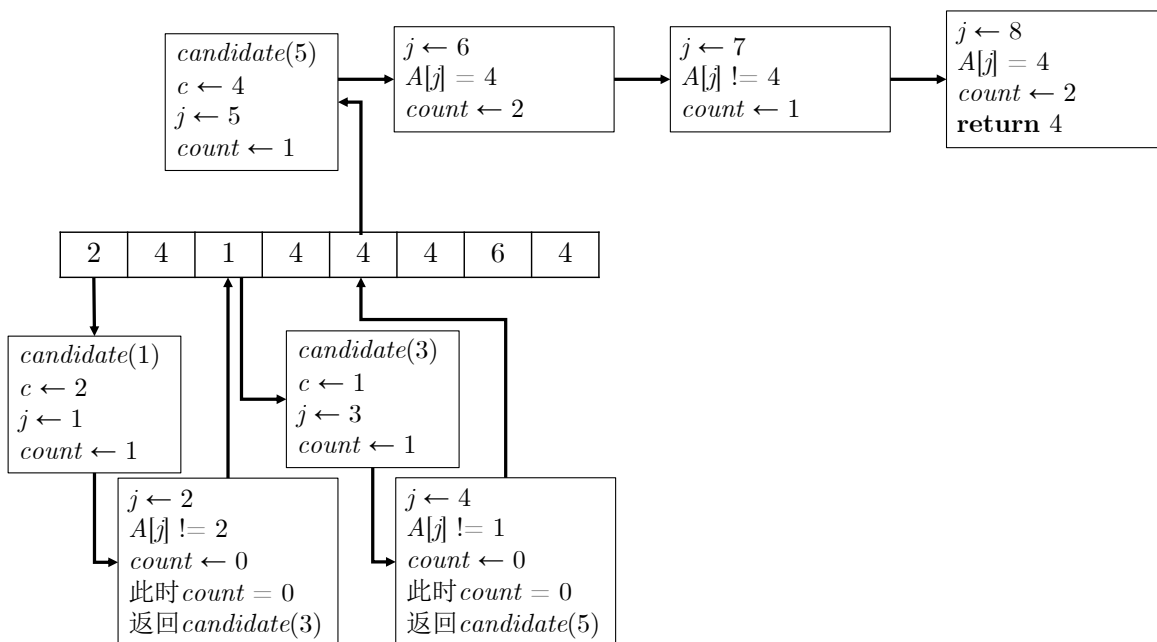
最后验证发现 5 出现了 3 次，是多数元素。

(b) $candidate(1)$ 的过程如图所示:



最后验证发现 5 出现了 2 次，不是多数元素。

(c) $candidate(1)$ 的过程如图所示:



最后验证发现 4 出现了 5 次，是多数元素。

解析: 在观察结论 5.1 中, 可以知道如果一个数是多数元素, 它出现的次数 $k > \lfloor n/2 \rfloor$, 那么每次去除两个不同元素时, 它的 $k' \geq k - 1$, 而 $n' = n - 2$, 仍然有 $k' > \lfloor n'/2 \rfloor$, 所以 MAJORITY 算法是有效的。只需要用图模拟出算法 5.9 即可。

如果一个数是全局多数元素 (在下标 $[1, n]$), 那么根据上述分析, 它也一定是局

部多数元素（在下标 $[3, n]$ ），可以使用归纳法。而因为返回值可能只是局部多数元素，还需要一次 $O(n)$ 的验证，而由于 *candidate* 过程是 $O(n)$ 的，所以这样做并不会增加复杂度。

习题 5.31

对下面的说法，证明其为正确或错误：

如果算法 MAJORITY 里过程 *candidate* 的第 7 步中 $j = n$ 并且 $count = 0$ ，那么 c 是多数元素。

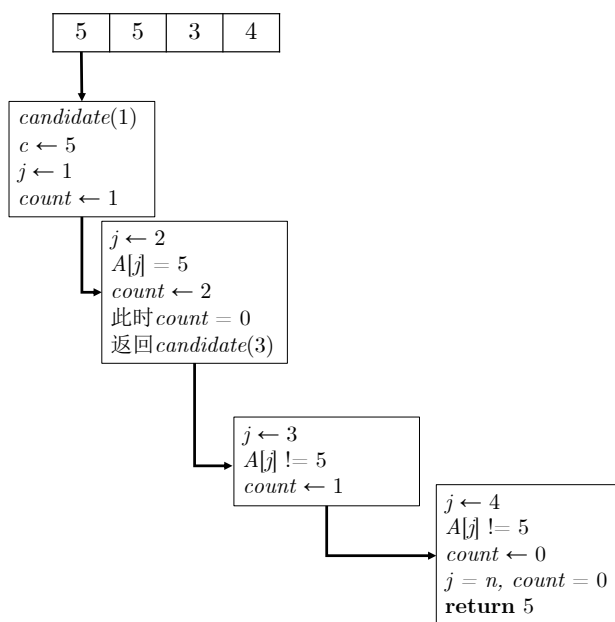
答案：错误。

证明：当 $j = n$ 且 $count = 0$ 时，容易想到，最后一次 *candidate*(i) 的区间 $[i, n]$ 中，元素 c 必然在其中一半的位置出现。考虑习题 5.29 的解析中的推断，元素 c 只在一半的位置出现，那么一定存在一个长为 m （ m 为偶数）的序列，其中元素 c 出现了 $m/2$ 次。

构造一种 $m = n$ 的情形，并使得 *candidate*(n) 时出现 $j = n$ 且 $count = 0$ 的情况，如下图所示：

5	5	3	4
---	---	---	---

那么有如下过程，其中 *candidate*(1) 直接执行到 $j = 4$ ，然后返回 5。（全程有 $c = 5$ ）



而 5 只出现了 2 次，不是多数元素。

补充题

一个数列的首项 $a_1 = 0$, 后续奇数项和偶数项的计算公式分别为 $a_{2n} = a_{2n-1} + 2$, $a_{2n+1} = a_{2n-1} + a_{2n} - 1$, 设计一个递归算法求数列第 n 项。

答案: 定义函数 $Calculate(n)$, 即可递归计算。

Algorithm 1: 计算数列第 n 项

Input: n

Output: $a_n \leftarrow Calculate(n)$


```

1  $Calculate(n)$ 
2 过程  $Calculate(n)$ 
3   if  $n = 1$  then return 0
4   if  $n \% 2 = 0$  then
5       return  $Calculate(n - 1) + 2$ 
6   else
7       return  $Calculate(n - 1) + Calculate(n - 2) - 1$ 
8   end
9 end

```

解析: 这个算法的时间复杂度较高, 可以定义一个函数, 将已经计算出的数列项进行存储, 降低时间复杂度, 方法如下以供参考。算法第 8 行“if $f[n] \neq 0$ ”时返回, 是因为考虑到当 $n > 1$ 时, 都有 $a_n > 0$ 的前提, 所以可以用 f 值是否为 0 来判断是否已经计算过该值。

这也类似于 3 月 30 日课上所讲的斐波那契数列求法。



4.1.1 斐波那契序列 (Fibonacci)

一种有效算法

- 采取**自底向上**的方式递推求值, 并把**中间结果**存储起来以便以后用来计算所要求的解。利用这种技术可以设计出特别有效的算法来解决许多组合最优化问题。
- 对于Fibonacci 函数来说, 如果从 f_1 开始自底向上地计算直至到 f_n , 只需要 $\Theta(n)$ 时间和 $\Theta(1)$ 空间。和上面的方法相比, 可以很大程度地降低时间复杂度。

2023/3/30
Algorithms Design Techniques and Analysis
10

Algorithm 2: 优化后的计算数列第 n 项

Input: n
Output: $a_n \leftarrow \text{Calculate}(n)$

```
1 for  $i \leftarrow 1$  to  $n$  do
2   |  $f[i] \leftarrow 0$ 
3 end
4  $f \leftarrow []$ 
5  $\text{Calculate}(n)$ 
6 过程  $\text{Calculate}(n)$ 
7   if  $n = 1$  then return  $0$ 
8   if  $f[n] \neq 0$  then return  $f[n]$ 
9   if  $n \% 2 = 0$  then
10    |  $f[n] \leftarrow \text{Calculate}(n - 1) + 2$ 
11    | return  $f[n]$ 
12  else
13    |  $f[n] \leftarrow \text{Calculate}(n - 1) + \text{Calculate}(n - 2) - 1$ 
14    | return  $f[n]$ 
15  end
16 end
```

总结

题目编号: 5.7, 5.12, 5.29, 5.31, 补充题

日期: 2023 年 3 月 30 日

批改人: 王骏骁

邮箱: wjyyy1@126.com

习题 5.7: 部分同学排版不够清晰, 行和列的关系看不清楚。题目中的逗号是千分号, 不是分隔数字。

习题 5.12: 注意使用的算法是 INSERTIONSORT, 插入排序的复杂度为 $O(n^2)$ 。其中 m 可以认为和 n 同阶, 但 k 仍是参数。此外考虑将每个数分配到桶中的时间复杂度, 可以写 $O(n^2/k + n)$ 。

习题 5.29: (b) 题注意返回无解。

补充题: 部分同学化简公式后漏了倍数, 即 $a_{2n+1} = 2a_{2n-1} + 1$ 。有些同学没有写补充题。