算法设计与分析 Week 1

第一周作业-Solution

Lecturer: 杨启哲 Last modified: 2025 年 9 月 28 日

1. 用 True 或者 False 填空:

$f(n)$	$g(n)$	$f = O(g)$	$f = o(g)$	$f = \Omega(g)$	$f = \Theta(g)$	$f = \omega(g)$
$n^3 + 3n$	$100n^2 + 2n + 100$	F	F	T	F	T
$50n + \log n$	$10n + \log\log n$	T	F	T	T	F
$50n^2 \log n$	$10n(\log n)^3$	F	F	T	F	T
$n + \log n$	$\log^{100} n + \log n$	F	F	T	F	T
$n!$	$2^n + n^{100}$	F	F	T	F	T

2. 考虑如下算法 COUNT6:

算法: COUNT6

输入: 正整数 n

输出: 第 6 步的执行次数 count

```
1: count ← 0
2: for i ← 1 to ⌊log n⌋ do
3:     for j ← i to i + 5 do
4:         for k ← 1 to i^2 do
5:             count ← count + 1
6:         end for
7:     end for
8: end for
```

(1) 第 5 步执行了多少次?

(2) 该算法的时间复杂性是什么?

解答. 算法共有 3 层循环:

- 第一个循环共执行了 $\lfloor \log n \rfloor$ 次。
- 第二个循环每次执行 6 次。
- 第三个循环每次执行 i^2 次。

从而程序一共执行:

$$6(1 + 2^2 + 3^2 + \cdots + (\lfloor \log n \rfloor)^2) = 6 \cdot \frac{\lfloor \log n \rfloor(\lfloor \log n \rfloor + 1)(2\lfloor \log n \rfloor + 1)}{6} = \Theta((\log n)^3)$$

次 count ++ 命令。注意到输入规模是 $\log n$，因此该算法是一个时间为 $O((\log n)^3)$ 的算法，为多项式 (三次方) 算法。 □

3. 请找到两个单调递增函数 $f(n)$ 和 $g(n)$，使得 $f \neq O(g)$ 并且 $g \neq O(f)$。

解答. 考察下列两个函数 $f(n), g(n)$:

$$f(n) = \begin{cases} n^{2n}, & n\text{为偶数} \\ n^{2n+1}, & n\text{为奇数} \end{cases}, \quad g(n) = \begin{cases} n^{2n-1}, & n\text{为偶数} \\ n^{2n}, & n\text{为奇数} \end{cases},$$

注意到:

- 当 n 为奇数时，$\frac{f(n)}{g(n)} = n \to \infty$.
- 当 n 为偶数时，$\frac{g(n)}{f(n)} = n \to \infty$.

所以我们有 $f \neq O(g)$，$g \neq O(f)$。下面证两个函数都是单调递增的:

- 当 $n = 2k$ 时，我们有:

$$\begin{aligned} f(n) - f(n-1) &= (2k)^{4k} - (2k-1)^{4k-1} > 0 \\ g(n) - g(n-1) &= (2k)^{4k-1} - (2k-1)^{4k-2} > 0 \end{aligned}$$

- 当 $n = 2k+1$ 时，我们有:

$$\begin{aligned} f(n) - f(n-1) &= (2k+1)^{4k+3} - (2k)^{4k} > 0 \\ g(n) - g(n-1) &= (2k+1)^{4k+2} - (2k)^{4k-1} > 0 \end{aligned}$$

□

4. 考虑如下的一个算法 EUCLID:

算法: EUCLID

输入: 两个正整数 a, b
输出: $\gcd(a, b)$

repeat
  $r \leftarrow b \mod a$
  $b \leftarrow a$
  $a \leftarrow r$
until $a = 0$
return b

(1) 假定 $b \geqslant a$，用 b 来表示该算法的运行时间。

(2) 这个算法的时间复杂性是什么?

Remark 0.1

这个算法实际上就求最大公约数的辗转相除法，当然我们这里省去了对其正确性的证明。

解答. 该算法的时间复杂性主要取决于循环体的执行次数，假设循环体执行了 k 次，我们有:

$$\begin{aligned} a_1 &= a \\ b_1 &= b \\ a_2 &= b_1 \mod a_1 < b_1 \\ b_2 &= a_1 < a_2 \\ &\vdots \end{aligned}$$

注意到在这个过程中，每次一定有:

$$a_i \leqslant \frac{1}{2}\min\{a_{i-1}, b_{i-1}\}$$

即:

$$a_i \leqslant \frac{1}{2}a_{i-1} \leqslant \left(\frac{1}{2}\right)^{i-1} a_1 \leqslant \left(\frac{1}{2}\right)^{i-1} b$$

从而循环至多执行 $k = \lceil \log_2 b \rceil$ 次，因此该算法的时间复杂性为 $O(\log b)$，注意到其输入数字也是按照二进制存储的，因此该算法是一个多项式时间算法。 □

5. (鸡蛋掉落) 假设现在有一幢 N 层高的楼和一些鸡蛋。对于这些鸡蛋来说，存在一层楼 T, 使得当这些鸡蛋从 T 层楼或更高的楼层摔落下去时鸡蛋会碎，反之鸡蛋则不会碎。你现在的目标是在下述条件下设计算法找到这个楼层 T:

(1) 你只有 1 个鸡蛋，但有 T 次机会。
(2) 你有 $\log N$ 个鸡蛋和 $\log N$ 次机会。
(3) 你有 $\log T$ 个鸡蛋和 $2\log T$ 次机会。
(4) 你有 2 个鸡蛋和 $2\sqrt{N}$ 次机会。
(5) 你有 2 个鸡蛋和 $c\sqrt{T}$ 次机会，这里 c 是一个与 T, N 无关的常数。

Remark 0.2

这是个很经典的问题，但我还是希望大家先自己尝试思考一下。

解答. 这道题的关键在于 N 和 T 的分别。

(1) 算法非常简单，从 1 楼开始，逐层向上扔鸡蛋，直到鸡蛋碎了为止。这样最多需要 T 次。
(2) 基于二分思想的算法，从 $\lfloor \frac{N}{2} \rfloor$ 层楼开始扔鸡蛋，有两种情况:
   - 鸡蛋碎了，说明 T 在 1 到 $\lfloor \frac{N}{2} \rfloor$ 之间，此时剩余 $\log N - 1$ 个鸡蛋和 $\log N - 1$ 次机会，重复上述二分过程。

- 鸡蛋没碎，说明 T 在 $\lfloor \frac{N}{2} \rfloor + 1$ 到 N 之间，此时剩余 $\log N - 1$ 个鸡蛋和 $\log N - 1$ 次机会，重复上述二分过程。

(3) 依旧基于二分算法，但现在要求的是 $\log T$，因此我们需要先确定 T 的大小。

  (i) 从 $1, 2, 2^2, \ldots$ 楼层开始扔鸡蛋，直到鸡蛋碎了为止，此时 T 在 2^{k-1} 到 2^k 之间，其中 k 为扔鸡蛋的次数。

  (ii) 仿照第二问的算法，在 $2^{k-1} \sim 2^k$ 之间的楼层找到确切的 T.

  显然算法的第一步需要消耗 $\log T$ 次机会和 1 个鸡蛋，因此由第二问的结论可知，一共至多消耗 $\log T$ 个鸡蛋和 $2 \log T$ 次机会。

(4) 鸡蛋变少了，次数变多了。因此我们可以设计如下的算法:

  (i) 从 $\lfloor \sqrt{n} \rfloor, 2\lfloor \sqrt{n} \rfloor, \ldots$ 逐层向上扔鸡蛋，直到鸡蛋碎了为止。这样最多需要 $\lfloor \sqrt{n} \rfloor$ 次。

  (ii) 第一步确定 T 的范围在 $k\lfloor \sqrt{n} \rfloor$ 到 $(k+1)\lfloor \sqrt{n} \rfloor$ 之间，因此我们可以在这个范围内逐层向上扔鸡蛋，直到鸡蛋碎了为止。这样最多需要 $\lfloor \sqrt{n} \rfloor$ 次。

  从而我们可以用 2 个鸡蛋和 $2\sqrt{n}$ 次机会找到 T。

(5) 和第 3 问的想法类似，想要做到 2 次机会和 $c \log T$ 次尝试，我们首先要确定 T 的范围，因此我们可以设计如下的算法，其中令 S_i 表示 $1 + 2 + \cdots + i$ 的和:

  (i) 从 $1 + S_1, 1 + S_2, \ldots$ 逐层向上扔鸡蛋，直到鸡蛋碎了为止。这样最多需要 k 次。

  (ii) 从 $1 + S_{k-1} + 1$ 层开始逐逐层向上扔鸡蛋，直到鸡蛋碎了为止，此时便找到了相应的 T.

  注意到在第一个鸡蛋碎的时候我们有:

$$2 + \frac{(k-1)k}{2} = 1 + S_{k-1} + 1 \leqslant T \leqslant 1 + S_k$$

  从而我们有 $k \leqslant \sqrt{2} \cdot (\sqrt{T} - 1)$，因此令 $c = 2\sqrt{2}$ 即满足要求。

$\square$