算法设计与分析 LECTURE 1

第一周作业-Solution

LECTURER: 杨启哲 LAST MODIFIED: 2024 年 9 月 25 日

1. 令数组 $A[1, \ldots 500] = 1, 2 \ldots, 500$，用算法 $BinarySearch$ 搜索下列元素时，分别执行了多少次比较运算？

   (a) -3 (b) 1 (c) 500 (d) 250

解答. 注意到 $BinarySearch$ 的 mid 更新规则为 $\lfloor (low + high)/2 \rfloor$，其中需要进行的比较次数有 $low \le high$，$j = 0$，$x = a[mid]$ 和 $x < a[mid]$，因此计算可得：

(a) 33 次
(b) 32 次
(c) 36 次
(d) 4 次

如果只考虑 $x = a[mid]$ 和 $x < a[mid]$，即不考虑循环的判断比较则计算可得：

(a) 16 次
(b) 15 次
(c) 17 次
(d) 1 次

□

Remark 0.1

一般我们只关注 $x = a[mid]$ 和 $x < a[mid]$ 的次数，因为循环体中关于 $low \le high$，$j = 0$ 的判断依赖于具体的实现，并且其不会超过 $x = a[mid]$ 和 $x < a[mid]$ 的增长速率。但这里怕同学们误解，我给了两个版本的比较次数。

2. 用 $True$ 或者 $False$ 填空：

$f(n)$	$g(n)$	$f = O(g)$	$f = o(g)$	$f = \Omega(g)$	$f = \Theta(g)$	$f = \omega(g)$
$0.2n^3 + 3n$	$100n^2 + 2n + 100$					
$50n + \log n$	$10n + \log\log n$					
$50n \log n$	$10n \log\log n$					
$n + \log n$	$\log^{100} n + \log n$					
$n!$	$2^n + n^{100}$					

解答. 计算可得：

$f(n)$	$g(n)$	$f = O(g)$	$f = o(g)$	$f = \Omega(g)$	$f = \Theta(g)$	$f = \omega(g)$
$0.2n^3 + 3n$	$100n^2 + 2n + 100$	F	F	T	F	T
$50n + \log n$	$10n + \log\log n$	T	F	T	T	F
$50n \log n$	$10n \log\log n$	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

□

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. 考虑如下算法 $COUNT7$:

算法: $COUNT7$

输入: 正整数 n
输出: 第 6 步的执行次数 $count$

1: $count \leftarrow 0$
2: for $i \leftarrow 1$ to n do
3: $\quad j \leftarrow \lfloor \frac{n}{2} \rfloor$.
4: $\quad$ while $j \geq 1$ do
5: $\quad\quad count \leftarrow count + 1$
6: $\quad\quad$ if j 是奇数 then $j \leftarrow 0$
7: $\quad\quad$ else $j \leftarrow \lfloor \frac{j}{2} \rfloor$
8: $\quad\quad$ end if
9: $\quad$ end while
10: end for

(1) 第 6 步执行了多少次?
(2) 要表示算法的时间复杂性，O 和 Θ 哪个符号更合适? 为什么?
(3) 该算法的时间复杂性是什么?

Remark 0.2

应该是第 5 步 $count + 1$ 的执行次数。

解答. 算法一共有 2 个循环，第一个循环执行了 n 次，第二个循环每次执行的次数都是一样的，假设为 k 次。k 实际上式满足最大的 $2^k \mid n$ 的值，因此 $k \leq \lfloor \log n \rfloor$，因此我们有：

(1) 第 6 步执行了 $nk \leq \lfloor \log n \rfloor$ 次。

(2) 从数学的角度来讲 Θ 是一个更好的符号，因为这不仅表示了算法至多这么快，也表示了算法至少这么快，更为精确。但大部分时候从算法的角度来说，我们给出一个算法，其实只能说明了这么快一定可以做出来，但并没有保证至少也需要这么多时间，所以对于一个具体的算法来说，我们如果可以准确分析，Θ 是个更好的符号，但对于某个具体的问题来说，即使我们给出了一个能准确分析的算法，我们暂时也只能说这个问题是在 $O(f)$ 里的，因为我们并不能保证这个问题一定需要这么多时间。

(3) 由第一问可知算法的时间复杂性为 $\Theta(n \log n)$.

□

5. 考虑元素唯一性问题：给定一个数组 $A[1, \ldots, n]$，其中的元素都是整数，请设计一个有效的算法来判断数组中是否有重复元素；并尝试分析你的算法的时间复杂性。

解答. 考虑如下的算法：

(1) 对数组 A 进行排序。

(2) 从头到尾扫描数组，如果发现有两个相邻的元素相等，则返回 $True$，否则返回 $False$.

注意到排序过后，相同的元素必然是相邻的，因此算法可以找出是否有重复元素。由于排序算法需要 $O(n \log n)$ 的时间复杂性，因此整个算法的时间复杂性为 $O(n \log n)$. □

Remark 0.3

从实际角度来说，建立哈希表是一个更好的方式，但需要一定的方式解决哈希冲突。

6. (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 0.4

这道题是去年第一次作业的一道题，答案在去年的课程主页上已经给出，所以这**不是必做的题目**。但是有兴趣的同学可以先自己思考一下，然后再去查看答案。

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：

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

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

$\square$