算法与复杂性　作业三

516021910528 - SHEN Jiamin

2020 年 3 月 9 日

1. k 为正常数，证明 $\log_m n = o(n^k)$

证明

1. 当 $0 < m < 1$ 时，$\forall c > 0, n > 1, k > 0$

$$\log_m n < 0 < cn^k$$

2. 当 $m > 1$ 时，令

$$f(x) = \frac{\log_m x}{x^k}$$
$$f'(x) = -\frac{k}{\ln m}\frac{1}{x^{k+2}}$$

由 $k > 0$，$\ln m > 0$ 可知，当 $x > 0$ 时，$f'(x) < 0$，$f(x)$ 单调递减

$\forall c > 0, \exists N = m^c > 0$，使得当 $n > N$ 时

$$f(n) < f(N) = \frac{\log_m m^c}{m^{ck}} = \frac{c}{m^{ck}}$$

由 $ck > 0$，$m > 1$ 可知，$m^{ck} > 1$

$$c \cdot m^{ck} > c$$
$$c > \frac{c}{m^{ck}} = f(N) > f(n) = \frac{\log_m n}{n^k}$$
$$\log_m n < cn^k$$

所以

$$\log_m n = o(n^k)$$

□

2. 寻找单调递增函数 $f(n)$ 和 $g(n)$，使得 $f(n) = O(g(n))$ 和 $g(n) = O(f(n))$ 都不成立

即寻找单调递增函数 $f(n)$ 和 $g(n)$，使得 $\forall c > 0, \forall N > 0$

$$\exists n_1 > N, f(n_1) > cg(n_1), \quad \exists n_2 > N, g(n_2) > cf(n_2)$$

令 $k = \left\lfloor \frac{\lfloor \log_2 n \rfloor}{2} \right\rfloor \in \mathbb{N}$

$$f(n) = \begin{cases} 2^n & , 2^{2k} \leq n < 2^{2k+1} \\ 2^{2^{2k+1}} & , 2^{2k+1} \leq n < 2^{k+2} \end{cases}$$

$$g(n) = \begin{cases} 2^{2^{2k}} & , 2^{2k} \leq n < 2^{k+1} \\ 2^n & , 2^{2k+1} \leq n < 2^{2k+2} \end{cases}$$

则，当 $2^{2k} \leq n < 2^{2k+1}$ 时，

$$\frac{f(n)}{g(n)} = \frac{2^n}{2^{2^{2k}}} = 2^{(n-2^{2k})} \tag{1}$$

当 $2^{2k+1} \leq n < 2^{2k+2}$ 时，

$$\frac{g(n)}{f(n)} = \frac{2^n}{2^{2^{2k+1}}} = 2^{(n-2^{2k+1})} \tag{2}$$

$\forall c > 0, N > 0$，一定存在一个足够大的 $m > 0$，使得 $c < 2^{2^{2m}}$ 且 $N < 2^{2m}$

- 由 (1) 式可知，当 $n = 2^{2m+3} - 1 > N$ 时，$2^{2m+2} < n < 2^{2m+3}$，

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

- 由 (2) 式可知，当 $n = 2^{2m+4} - 1 > N$ 时，$2^{2m+3} < n < 2^{2m+4}$，

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

综合可知，对于上述 $f(n), g(n)$，$f(n) = O(g(n))$ 和 $g(n) = O(f(n))$ 都不成立

3. 假设解决同一个问题的两个算法 A1 和 A2 的时间复杂性分别为 $O(n^3)$ 和 $O(n)$，如果为这两个算法分别编写程序并在同样的环境下运行，算法 A2 的程序一定比算法 A1 的程序运行得快吗？为什么？

不一定。程序的运行速度除了取决于时间复杂度，还取决于输入的规模以及程序的质量。在时间复杂度中，除了主项，还包括被忽略的常数以及其他项可能影响实际的运行时间。

如算法 A1 的程序可能运行的时间为 n^3，而算法 A2 的程序可能运行时间为 $256n$。此时，若输入规模 $n < 16$，则算法 A1 比算法 A2 运行得更快。

4. 考虑以下冒泡排序算法。

算法 1 BubbleSort

输入: n 个元素的数组 $A[1 \dots n]$

输出: 按非降序排列的数组 $A[1 \dots n]$

```
i ← 1, sorted ← false
while i ≤ n − 1 ∧ sorted ≠ true do
    sorted ← true
    for j ← n downto i + 1 do
        if A[j] < A[j-1] then
            SWAP(A[j], A[j − 1])
            sorted ← false
        end if
    end for
    i ← i + 1
end while
```

(a) 元素比较的最少次数是多少？何时达到最小值？

当数组本身为非降序时可达到最小值。此时外层循环只执行 1 次，内层循环只执行 $n-1$ 次。所以元素比较次数最少为 $n-1$ 次。

(b) 元素比较的最多次数是多少？何时达到最大值？

当数组本身为严格升序时可达到最大值。此时外层循环执行 $n-1$ 次，内层循环共执行次数为

$$\sum_{i=1}^{n-1}(n-i) = \frac{n(n-1)}{2}$$

所以元素比较的最多次数为 $\frac{n(n-1)}{2}$ 次。

(c) 可以用 O,Ω,Θ 表示算法的运行时间吗？

- 最好情况：

$$n-1 = O(n) = \Omega(n) = \Theta(n)$$

- 最坏情况：

$$\frac{1}{2}(n^2-n) = O(n^2) = \Omega(n^2) = \Theta(n^2)$$