

# 递归时间复杂度分析

dbywsc

2025/8

# 目录

1 一些符号

2 对数的基本运算

3 主定理

4 初赛题目选讲

在之前的学习中，我们已经学会了使用大  $O$  符号分析时间复杂度，比如：

```
for(int i = 1; i <= n; i++) {  
    for(int j = 1; j <= m; j++) {  
        ans += a[i][j];  
    }  
}
```

在上面的代码中， $\text{ans} += \text{a}[i][j]$  会执行  $n \times m$  次，因此我们称其时间复杂度为  $O(nm)$ 。当一段代码执行的次数类似于  $an^3 + bn^2 + cn + T$  时，我们将其时间复杂度记做  $O(n^3)$ 。事实上，我们使用的符号  $O$  在复杂度分析中称之为 **大 O 表示法** (big-oh)，这种忽略低次项、常数项和系数的时间复杂度叫 **渐进时间复杂度**。

除了 big-oh 之外，时间复杂度还有其他的表示符号。

设一段代码执行的严格时间为  $T(n)$ ，则有以下符号：

$\Theta(n)$ ：读作 theta，用这种符号最精确的时间复杂度，即

$\Theta(n) = T(n)$ 。

$O(n)$ ：读作 big-oh，使用这种方法表示“小于等于”，即

$O(n) \leq T(n)$ 。

$o(n)$ ：读作 small-oh，使用这种方法表示“小于”，即

$o(n) < T(n)$ 。

$\Omega(n)$ ：读作 big-omega，使用这种方法表示“大于等于”，即

$\Omega(n) \geq T(n)$ 。

$\omega(n)$ ：读作 small-omega，使用这种方法表示“大于”，即

$\omega(n) > T(n)$ 。在实际做题中，我们常用  $O(n)$  估计时间；而在理论分析中，我们常用  $\Theta(n)$  和  $O(n)$  两种符号。

在讲解递归时间复杂度之前，我们需要先熟悉一下对数的使用。设  $a^b = N$ ，那么我们可以把这个式子写成  $\log_a N = b$ 。可以发现，形如  $f(x) = \log_a x$  与  $g(x) = a^x$ ，当二者的  $a$  相等时  $f(x)$  和  $g(x)$  是互逆的，我们称  $f(x)$  和  $g(x)$  互为 **反函数**。下面列出几个常用的对数：

以 10 为底的对数称之为“常用对数”，记做  $\lg x$ 。

以  $e^1$  的对数称之为自然对数，记做  $\ln x$ 。

以 2 为底的对数在算法竞赛中也很常用，**本节中简写为  $\log x$** 。

---

<sup>1</sup>即欧拉常数，约等于 2.71828182846

下面是一些对数运算的法则：

$$\log_a x + \log_a y = \log_a(xy)$$

$$\log_a x - \log_a y = \log_a\left(\frac{x}{y}\right)$$

$$\log_a x^y = y \log_a x$$

$$a^{\log_a b} = b$$

在分析递归程序的时间复杂度时，常使用**主定理**(Master Theorem)。

设一个递归程序执行的时间为  $T(n)$ ，我们在递归的过程中，将这个问题分治为了  $a$  个规模为  $\frac{n}{b}$  的子问题，每次递归额外带来的计算为  $f(n)$ ，即：

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

那么存在以下关系<sup>2</sup>：

$$T(n) = \begin{cases} \Theta(n^{\log_b a}) & f(n) = O(n^{\log_b(a)-\epsilon}) \quad \epsilon > 0 \\ \Theta(f(n)) & f(n) = \Omega(n^{\log_b(a)+\epsilon}) \quad \epsilon > 0 \\ \Theta(n^{\log_b a} \log^{k+1} n) & f(n) = \Theta(n^{\log_b a} \log^k n) \quad k \geq 0 \end{cases}$$

其中， $\epsilon$  是常数。

<sup>2</sup>严格来说本节主定理描述其实并不准确。在第二种情况和第三种情况中，遇到临界情况时需要额外判断。本节忽略了这一点，但不影响做初赛的题目。

第一种情况：当  $f(n) = O(n^{\log_b(a)-\epsilon})$  时， $T(n) = \Theta(n^{\log_b a})$ 。

e.g

$$T(n) = 2T\left(\frac{n}{2}\right) + \sqrt{n}$$

$$a = 2, b = 2, \log_b a = 1, f(n) = n^{\frac{1}{2}}$$

$$f(n) = n^{\frac{1}{2}} = n^{\log_b a - \frac{1}{2}}。取 \epsilon = \frac{1}{2}，可得 T(n) = \Theta(n)$$

第二种情况：当  $f(n) = \Omega(\log_b(a) + \epsilon)$  时， $T(n) = \Theta(f(n))$ 。

e.g

$$T(n) = 2T\left(\frac{n}{2}\right) + n^2$$

$$a = b = 2, \log_b a = 1, f(n) = n^2$$

$$f(n) = n^2 = n^{\log_b a + 1}, \text{ 取 } \epsilon = 1, \text{ 可得 } T(n) = \Theta(n^2)$$

第三种情况：当  $f(n) = \Theta(n^{\log_b a} \log^k n)$  时，  
 $T(n) = \Theta(n \log_b a \log^{k+1} n)$ 。

e.g

$$T(n) = 2T\left(\frac{n}{2}\right) + n$$

$$a = b = 2, \log_b a = 1, f(n) = n$$

当  $k = 0$  时： $f(n) = n = n^{\log_b a} \times \log^0 n$ ， 可得  
 $T(n) = \Theta(n^{\log_b a} \log^{0+1} n) = n \log n$ 。

NOIP 2016 提高组选择题第 14 题：

假设某算法的计算时间表达式为递推关系式：

$$T(n) = 2T\left(\frac{n}{4}\right) + \sqrt{(n)}$$

$$T(1) = 1$$

求算法时间复杂度。

由主定理得， $a = 2, b = 4, \log_b a = \frac{1}{2}, f(n) = n^{\frac{1}{2}}$ 。

根据第三种情况，取  $k = 0$  时， $f(n) = n^{\log_b a} \times \log^0 n$ ，因此

$$T(n) = \Theta(n^{\log_b a} \log^{0+1} n) = \Theta(n^{\frac{1}{2}} \log n)^3$$
。

---

<sup>3</sup>在初赛的选择题中，选项有时用 big-oh 表示法，此时如果同时存在 big-oh 和 Theta 取更精确的 Theta，如果只有 big-oh 取 big-oh 即可

NOIP 2015 提高组选择题第十题：

假设某算法的计算时间表达式为递推关系式：

$$T(n) = T(n - 1) + n \quad (n \in \mathbb{N}^*)$$

$$T(0) = 1$$

求算法时间复杂度。这个问题不需要主定理，直接计算即可。

$$T(n) = T(n - 1) + n = T(n - 2) + (n - 1) + n$$

$$= T(n - 3) + (n - 2) + (n - 1) + n = \dots$$

$$= T(0) + 1 + 2 + \dots + (n - 1) + n$$

$$= 1 + \frac{n(n + 1)}{2}$$

如果取 Theta，则  $T(n) = \Theta(1 + \frac{n(n+1)}{2})$ ，如果取 big-oh，那么渐进时间复杂度为  $O(n^2)$ 。

NOIP 2013 提高组选择题第七题：斐波那契数列的定义如下：

$$F_1 = F_2 = 1 \quad F_n = F_{n-2} + F_{n-1} \quad (n \geq 3)$$

问用下面方式计算  $F_n$  的时间复杂度是多少：

```
int F(int n) {
    if(n <= 2) return 1;
    return F(n - 1) + F(n - 2);
}
```

- A.  $O(1)$  , B.  $O(n)$  , C.  $O(n^2)$  D.  $O(F_n)$

本题仅仅要求我们求出渐进时间复杂度，显然，计算任何一项的复杂度都不会超过  $F_n$ ，因此复杂度最大的项就是计算  $F_n$  的复杂度，因此选 D。