

Algorithm - P24

代入法

Induction Method

代入法指在求解递归式时，先猜测一个界，然后通过数学归纳法（强归纳法）证明渐近界正确。

使用代入法的步骤是为：

1. 猜测递归式解的形式

2. 在利用数学归纳法（强归纳法）证明递归式渐近界正确的过程中

找到使 $T(n) = O(g(n))$ 成立的常数 C, N 组成的凭证 (C, N)

如求解 $T(n) = 2T(\lfloor n/2 \rfloor) + n$ ，首先猜测解为 $T(n) = O(n \lg n)$ ，如果成立

则在递归步聚中，假设存在常数 $C > 0$ ，使得对于任意正整数 $m < n$, $T(m) \leq Cm \lg^m$ 为真

则考虑 $T(n) = 2T(\lfloor n/2 \rfloor) + n \stackrel{(IH)}{\leq} 2[C\lfloor n/2 \rfloor \lg^{\lfloor n/2 \rfloor}] + n$ ，根据归纳假设

$$\leq Cn \lg^{(n/2)} + n = Cn \lg^n - cn + n \leq Cn \lg^n$$

可知递归情形对于任意正常数 $C > 0$, $N=1$ 成立

再考虑基础情形，如果取 $T(1)$ 为基础步聚，又 $T(1)=1$ ，

则 $T(1) = 1 \leq C \cdot 1 \cdot \lg^1 = 0$ 对于任何常数 C 均不成立。

于是保留 $T(1)=1$ ，考虑 $T(2)$ 和 $T(3)$ 作为基础步聚

有 $T(2) = 2 \cdot T(1) + 2 = 4$, $T(3) = 2 \cdot T(2) + 3 = 5$

于是有当 $C \geq 2$ 时, $T(2) \leq C \cdot 2 \cdot \lg^2$ 和 $T(3) \leq C \cdot 3 \cdot \lg^3$ 成立

于是得到凭证 $(2, 1)$ 使得解 $T(n) = O(n \lg n)$ 成立

对于一个类似的递归式，猜测一个类似的解通常是合理的。

如求解 $T(n) = 2T(\lfloor n/2 \rfloor) + 17 + n$ ，猜测解为 $T(n) = O(n \lg n)$

则在递归步聚中，假设存在常数 $C > 0$ ，使得对于任意正整数 $34 \leq m < n$, $T(m) \leq Cm \lg^m$ 为真

则考虑 $T(n) = 2T(\lfloor n/2 \rfloor + 17) + n \stackrel{(IH)}{\leq} 2C(\lfloor n/2 \rfloor + 17) \lg^{\lfloor n/2 \rfloor + 17} + n$

$$\leq Cn \lg^{\lfloor n/2 \rfloor + 17} + 34 \lg^{\lfloor n/2 \rfloor + 17} + n$$

取 $n \geq 68$ ，则有 $\lfloor n/2 \rfloor + 17 \leq \frac{3}{4}n$ ，即 $T(n) \leq Cn \lg^n - cn \lg^{\frac{3}{4}} + 34 \lg^{\frac{3}{4}n} + n$

$$= Cn \lg^{\frac{3}{4}}(c-1)n + 34 \lg^{\frac{3}{4}n} = ④(n) - ④(c \lg^n) = ④(n)$$

即存在 $c > 0$ 使得 $T(n) \leq C_n \lg^n - ④(n) \leq C_n \lg^n$

再考虑基础情形，取 $n=68$ ，有 $T(n)=k$ ，其中 k 为大于 0 的常数

则存在 $C_2 > 0$ ，使得 $T(68) = k \leq C_2 \cdot 68 \cdot \lg^{68}$

再取 $C = \max(C_1, C_2)$ 可知， C 同时满足基础步聚。

于是得到凭证 $(C, 67)$ ，使得解 $T(n) = O(n \lg n)$ 成立

所以对于子问题规模中不显著影响递归式解的低阶项，可以忽略且

可以考虑忽略后，再使用代入法进行猜测求解。

Algorithm - P25

递归 - 归纳法

代入法(细节) 注意可能出现在正确地猜测出了渐近界的情况下

在递归步聚证明时出现递归步聚无法证明的情形

如对于递归式 $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$, 猜测解为 $T(n) = O(n)$

于是假设存在一个常数 C , 使得 $0 \leq y < n$, 有 $T(y) \leq Cy$

则根据归纳假设, 有 $0 \leq \lfloor n/2 \rfloor, \lceil n/2 \rceil < n$

于是有 $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$

$$\stackrel{(IH)}{\leq} C\lfloor n/2 \rfloor + C\lceil n/2 \rceil + 1 = C(\lfloor n/2 \rfloor + \lceil n/2 \rceil) + 1$$

$\leq Cn + 1$

注意 $T(n) \leq Cn + 1$ 无法得出 $T(n) \leq cn$ 的结论, 所以归纳步聚证明失败

参考在离散数学归纳法应用时的类似情况, 可以考虑使用归纳裁入

即通过归纳法证明一个比原命题更强的命题是真, 从而完成原命题的证明,

如对于递归式 $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$, 猜测 $T(n) = O(n)$

注意这个命题是等价于 $\exists c > 0, N > 0 \forall n > N \rightarrow T(n) \leq cn$

构造一个更强的命题是 $\exists c > 0, N > 0, d \geq 1 \forall n > N \rightarrow T(n) \leq cn - d$

则在归纳步聚中, $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$

$$\stackrel{(IH)}{\leq} (c\lfloor n/2 \rfloor - d) + (c\lceil n/2 \rceil - d) + 1$$

$$\leq cn - d + (1 - d) \leq cn - d$$

于是归纳步聚成立, 即对于常量 $c > 0, N > 0, d \geq 1$,

这个命题是蕴含着 对于常量 $c > 0, N > 0$, 当 $n > N$ 时, $T(n) \leq cn$

则有 $T(n) = O(n)$ 成立

陷阱 特别注意: 归纳步聚必须得到与归纳假设严格一致的命题形式为真

如对于递归式, $T(n) = 2T(\lfloor n/2 \rfloor) + n$, 猜测 $T(n) = O(n)$

在递归步聚中, $T(n) = 2T(\lfloor n/2 \rfloor) + n$, 又 $T(y) \leq Cy$, $0 \leq y < n$

$$\stackrel{(IH)}{\leq} 2C\lfloor n/2 \rfloor + n \leq cn + n = O(n)$$

注意 $T(n) \leq cn + n$ 无法得出 $T(n) \leq cn$ 的结论, 所以 $cn + n = O(n)$ 不正确

使用三次实数乘法计算复数 $(a+bi)(c+di) = (ac-bd) + (ad+bc)i$. 其中 $a, b, c, d \in \mathbb{R}$

`complex.Multiply((a,b), (c,d)) = j complex.Multiply := (Num a) => (a, b) → (a, a) → (a, a)`

`let e1 = (a+b)*c`

$$e1 = (a+b)c = ac+bc$$

$$e2 = b*(c+d) = bc+bd$$

$$e3 = (a-b)*d = ad-bd$$

$$total = \text{inr}(e1-e2, e2+e3)]$$

$$e1-e2 = ac-bd, e2+e3 = ab+ad$$

Algorithm - P26

算法设计与分析

第19章 归纳法

代入法换元

对于某些不易代入的情形，可以通过换元，转换成容易代入验证的递归式。

如递归式 $T(n) = 2T(\lfloor \sqrt{n} \rfloor) + \lg n$ ， $(\text{c}, 2) \geq 4 = (1, 2) \geq 4$

可以令 $m = \lg n$ ，忽略舍入误差后有 $\lfloor \sqrt{n} \rfloor = 2^{\frac{m}{2}}$

于是有 $T(2^m) = T(2^{\frac{m}{2}}) + m$ ，再令 $S(m) = T(2^m)$

则有 $S(m) = S(m/2) + m$ ，于是有 $S(m) = O(m \lg m)$

于是有 $T(2^m) = O(m \lg m)$ ，即 $T(n) = O(\lg n \cdot \lg \lg n)$

注意由于代入法整体上是使用数学归纳法验证猜测的递归式的角平。

所以面临与数学归纳法相似的问题，即无法构造解而只能验证解。

递归树 (recursive tree)，用于生成一个对递归式的猜测，然后用代入法验证猜测是否正确。

如果用递归树生成猜测时，对于分支刻画和代价求和足够仔细，则可以直接证明递归式。

递归树中每个结点表示一个子问题，子问题是对应于某次递归函数调用。

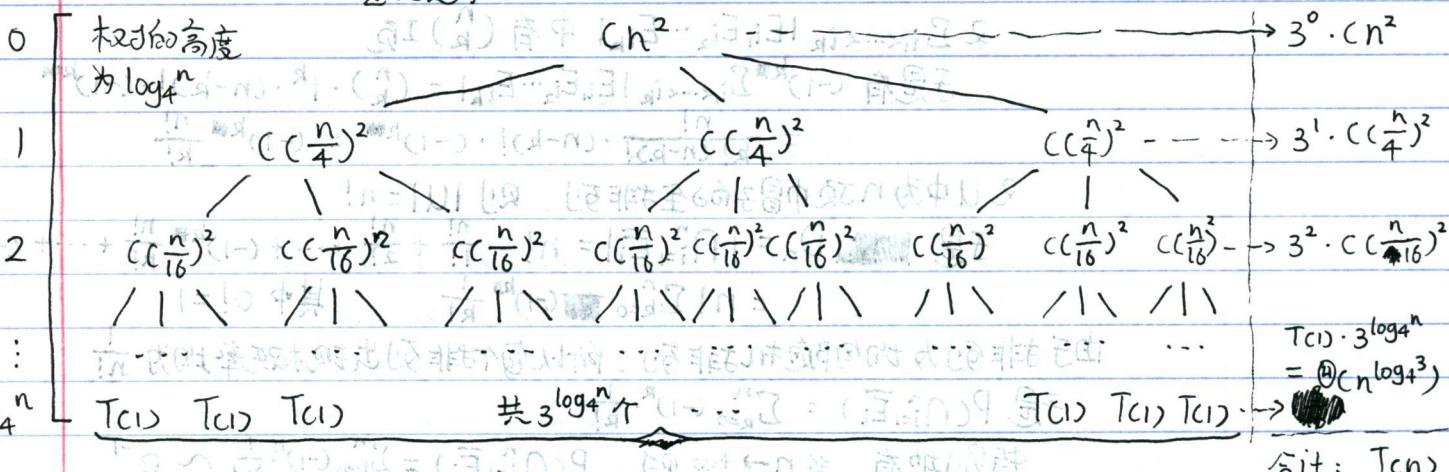
然后对树中每层的代价求和，得到树中每层的代价。

最后对树中所有层的代价求和，得到所有层次的递归树用的总代价。

如对于递归式 $T(n) = 3T(\lfloor n/4 \rfloor) + \Theta(n^2)$ ，可以设 $\Theta(n^2) = cn^2$ ，其中 $c > 0$

于是有递归树的基本形式如图所示。总计为 $T(n) = 3T(n/4) + cn^2$

则有完整的递归树如图。



$$\text{于是有 } T(n) = cn^2 + 3^1 \cdot (cn/4)^2 + 3^2 \cdot (cn/16)^2 + \dots + (3^{(\log_4^n)-1}) cn^2 + \Theta(n^{\log_4 3})$$

$$= \sum_{i=0}^{\log_4^n-1} \left(\frac{3}{16}\right)^i \cdot cn^2 + \Theta(n^{\log_4 3}) < \sum_{i=0}^{\infty} \left(\frac{3}{16}\right)^i \cdot cn^2 + \Theta(n^{\log_4 3})$$

$$= 1/(1-3/16) \cdot cn^2 + \Theta(n^{\log_4 3}) = O(cn^2)$$

又第一次递归调用即需要 $\Theta(n^2)$ ，所以 $T(n) = \Omega(n^2)$ ，于是可知 $T(n) = \Theta(n^2)$

Algorithm - P27

8.9 - Search problem

非完全 n 叉树，对于递归式 $T(n) = 3T(\lfloor n/4 \rfloor) + cn^2$ 中，每次递归调用中的子问题具有相同规模。于是可以知道对于每一个子问题分支，都会在树的同一层触及基础情形，或者说所有表示 $T(1)$ 的叶结点者在递归树的同一层上。但是对于子问题规模划分不相等且无法通过舍入来忽略差异的情况下，则可能存在有的分支比其他分支更早地触及基础情形并终止，导致并非所有的叶结点都在同一层。即递归树虽然依旧是满 n 叉树（国外定义），指树中的结点或者是叶结点，或者恰好有 n 个子结点，但是并非完全 n 叉树（国内定义的满 n 叉树）。即树中每个内点都至少有 n 个子结点，且总的树共有 $(n^{k+1} - 1)/(n-1)$ 个结点，高度为 k 。

如对于递归式 $T(n) = T(n/3) + T(2n/3) + cn$ ，（忽略舍入的问题）

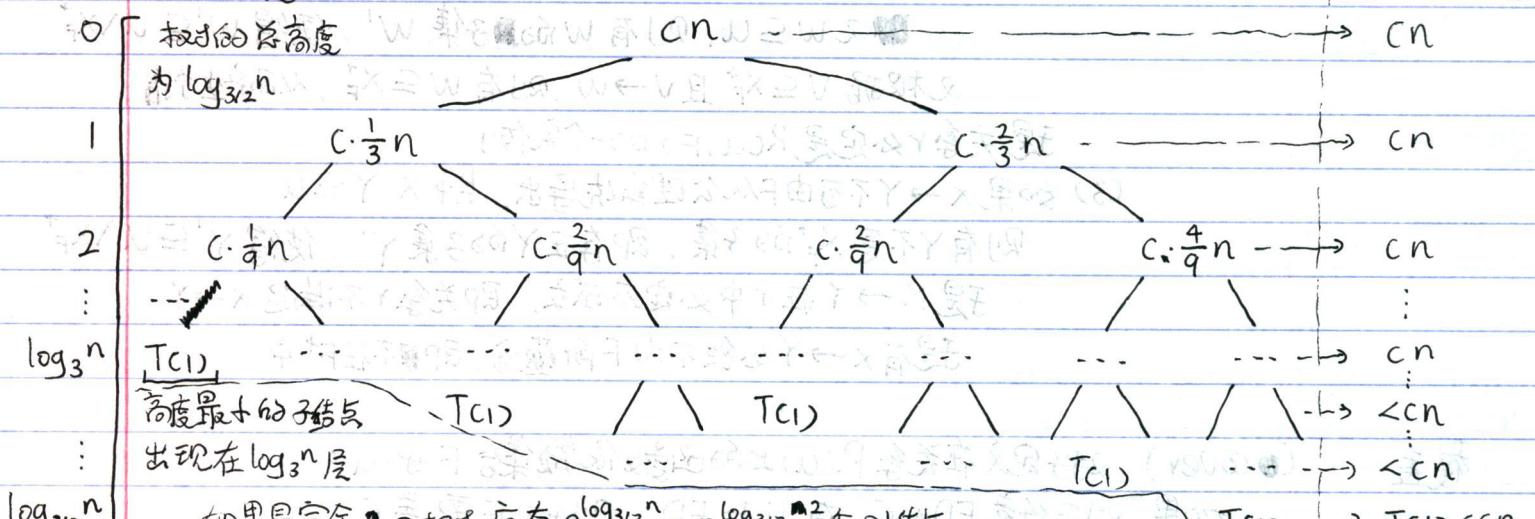
则其分解为子问题并最终到达 $T(1)$ 的过程中

有最长简单路径 $n \rightarrow \frac{2}{3}n \rightarrow (\frac{2}{3})^2 n \rightarrow \dots \rightarrow 1$ ，相当于 $k_1 = \log_{3/2} n$ ，即 $n = (\frac{3}{2})^{k_1}$ 。

而最短简单路径 $n \rightarrow \frac{1}{3}n \rightarrow (\frac{1}{3})^2 n \rightarrow \dots \rightarrow 1$ ，其中 $k_2 = \log_3 n < k_1$ 。

而其他分支的长度 k 介于两者之间，即 $k_2 < k < k_1$ 。

在递归树中表现为



如果是完全二叉树，应有 $2^{\log_{3/2} n} = n^{\log_{3/2} n^2}$ 个叶结点。

但是随着树逐渐接近最高层，缺失的叶结点越多。

合it: $O(n \lg n)$

但是仍然可以猜测 $T(n) = O(n \lg n)$ ，即 $T(n) \leq dn \lg n$ ，其中 d 为正常数。

$$\text{见 } T(n) = T(n/3) + T(2n/3) + cn \stackrel{\text{IH}}{\leq} d(\frac{n}{3} \lg \frac{n}{3}) + d(\frac{2n}{3} \lg \frac{2n}{3}) + cn$$

$$= d(\frac{n}{3} + \frac{2n}{3}) \lg n - (\frac{d}{3} \lg 3 + \frac{2d}{3} \lg 3) n + cn$$

$$= dn \lg n - (\lg 3 - \frac{2}{3})dn + cn$$

$$\leq dn \lg n$$

即有递归式 $T(n) = T(n/3) + T(2n/3) + cn$ 的解为 $T(n) = O(n \lg n)$

Algorithm - P28

ST 500

算法 - 分治法 / A

主定理 (master theorem) 用于形式化地求解递归式。

对于递归式 $T(n) = aT(n/b) + f(n)$, 其中常数 $a \geq 1, b > 1$ 且 $f(n)$ 是渐近正函数

表示将规模为 n 的原问题分解为 a 个子问题是

每个子问题的规模为 n/b ,

函数 $f(n)$ 为分解与合并子问题所需的代价

注意从技术上讲，递归式并非良好定义的，由于 n/b 可能不是整数

但是替换 $T(\lceil n/b \rceil)$ 和 $T(\lfloor n/b \rfloor)$ 为 $T(n/b)$ 并不会影响递归式的渐近性质

所以通常可以直接忽略舍入问题

对于定义在 $n \in \mathbb{N}$ 上的递归式 $T(n)$, 令 $T(1) = \Theta(1)$, $a \geq 1, b > 1$, $f(n)$ 是渐近正函数

$T(n) = aT(n/b) + f(n)$, 其中 n/b 可解释为 $\lceil n/b \rceil$ 或 $\lfloor n/b \rfloor$

则有三种主要情况：

1. 如果存在常数 $\epsilon > 0$, 有 $f(n) = O(n^{\log_b a - \epsilon})$,

则有 $T(n) = \Theta(n^{\log_b a})$

2. 如果 $f(n) = \Theta(n^{\log_b a})$, 则有 $T(n) = \Theta(n^{\log_b a} \cdot \lg n)$

3. 如果存在常数 $\epsilon > 0$, 有 $f(n) = \Omega(n^{\log_b a + \epsilon})$

且对于常数 $c < 1$, 和常数 $N > 0$, 对任意 $n > N$, 有 $a f(n/b) \leq c f(n)$

则有 $T(n) = \Theta(f(n))$

注意：对于形如 $f(n) = O(n^{\log_b a - \epsilon})$ 和 $f(n) = \Omega(n^{\log_b a + \epsilon})$, 其中常数 $\epsilon > 0$

称为多项式意义上的小于和大于 (Polynomially smaller/larger)

asymptotically smaller/larger than $n^{\log_b a}$ by a factor of n^ϵ

或者可以形式化地定义为：对于常数 $\epsilon > 0$,

$n^{\log_b a} / f(n) = \Omega(n^\epsilon)$ 称为多项式意义上的小于

$f(n) / n^{\log_b a} = \Omega(n^\epsilon)$ 称为多项式意义上的大于

注意：在第3种情形中，还需要额外满足正则条件 (regularity condition)

即存在一个正常数对 (c, N) , 其中 $c < 1, N > 0$

有 $a f(n/b) \leq c f(n)$ ，实际上指子问题本身的分解和合并不能是新的递归式

注意：在3种情况之间存在空隙，即无法使用主定理的情形

如 $T(n) = 2T(n/2) + n \lg n$, $n \lg n / n^{\log_2 2} = \lg n$

即不存在 $\epsilon > 0$, 使得 $\lg n = \Omega(n^\epsilon)$, 于是无法应用主定理

Algorithm - P29

例题 - 归纳法

主定理证明

分为两个部分，假定 $T(n)$ 仅定义在常数 $b > 1$ 的幂上，即仅对 $n = 1, b^1, b^2, \dots$ 定义，并且将分析扩展到任意正整数。

注意：在一个局限的域上使用渐近符号时，应小心避免得到错误结论。

如对于递归式 $T(n) = \begin{cases} n, & n=1,2,4,8,\dots \\ n^2, & \text{otherwise} \end{cases}$ 由 $T(2^i)$ 得出 $T(n) = O(n)$ 但总体上界应为 $T(n) = O(n^2)$

所以应用渐近符号在有限值域上，应绝对清楚应用环境。

对于递归式 $T(n) = aT(n/b) + f(n)$ ，假定 n 是常数 $b > 1$ 的幂，且 b 不一定是整数。

主定理证明过程分为三个引理：

① 求解主递归式的问题归约为一个求和表达式的求值。

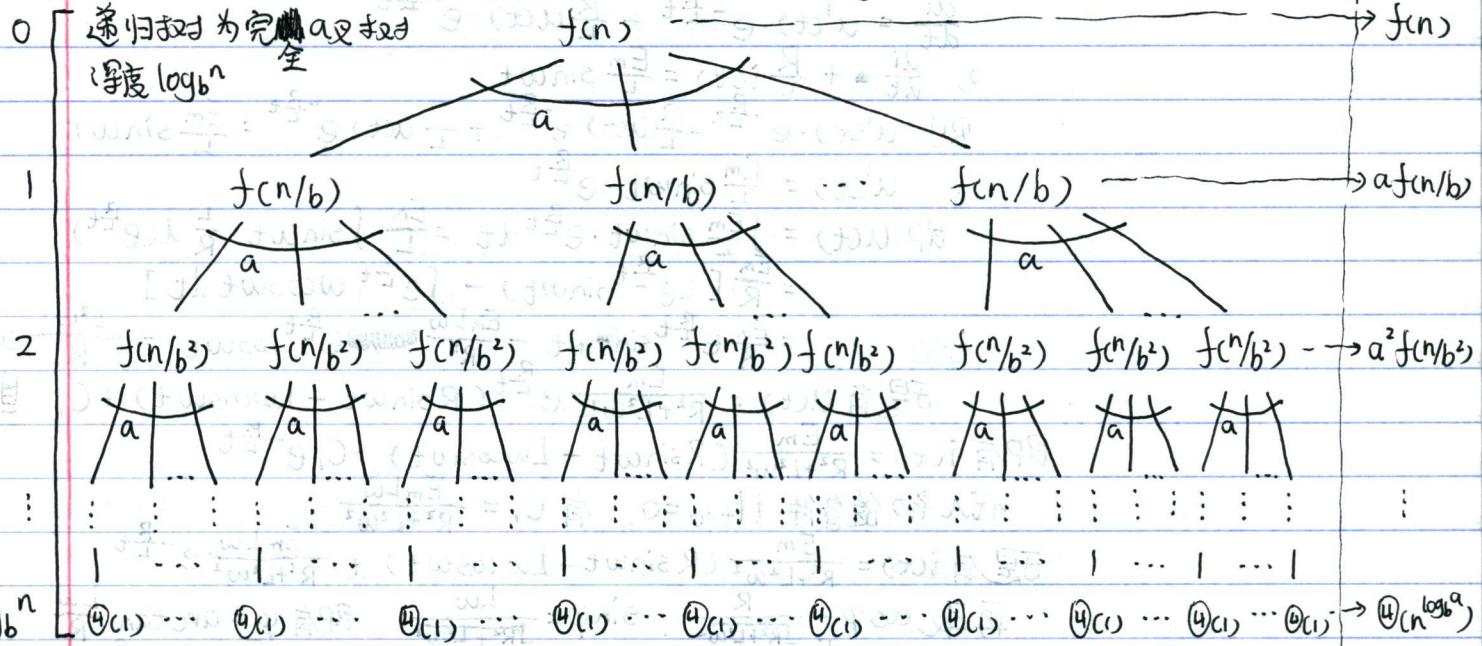
② 确定这个求和式的界。

③ 将上述两个引理合而为一，以证明在 n 为 b 的幂的情况下主定理。

引理 1 对于常数 $a \geq 1$ 和 $b > 1$ ，定义在 b 的幂上的递归式 $T(n) = \begin{cases} \Theta(1), & n=1 \\ aT(n/b) + f(n), & n=b^i, i \in \mathbb{Z}^+ \end{cases}$ 其中 $f(n)$ 是定义在 b 的幂上的非负函数。

则有 $T(n) = \Theta(n^{\log_b a}) + \sum_{j=0}^{\log_b n} a^j f(n/b^j)$

可以将 a 看作一个正整数，便于做出可视化递归树。



递归树共有 $a^{\log_b n}$ 个叶结点，每个叶结点代价为 $\Theta(1)$ ，总代价为 $\Theta(a^{\log_b n}) = \Theta(n^{\log_b a})$

每一层有 a^j 个内结，每个内结的分解与合并代价为 $f(n/b^j)$ ，总代价为 $a^j f(n/b^j)$

于是递归式转化为求和式 $T(n) = \Theta(n^{\log_b a}) + \sum_{j=0}^{\log_b n - 1} a^j f(n/b^j)$

递归树同时表达了递归式的三种代价分布情况。

由叶结点决定 / 均匀分布在所有层次上 / 由根结点 ($f(n)$) 决定

Algorithm - P30

引理2 对于定义在 b 的幂上的函数 $g(n) = \sum_{j=0}^{\log_b n - 1} a_j f(n/b^j)$, 其中常数 $a_j \geq 1$, $f(n)$ 是定义在 b 的幂上的非负函数

则有 1. 如果有常数 $\varepsilon > 0$, 使得 $f(n) = O(n^{\log_b a - \varepsilon})$, 则 $g(n) = O(n^{\log_b a})$

2. 如果 $f(n) = \Theta(n^{\log_b a})$, 则 $g(n) = \Theta(n^{\log_b a} \lg n)$

3. 如果有常数 $\varepsilon < 1$ 和 $N > 0$, 使得当 $n > N$ 时, $a f(n/b) \leq c f(n)$, 则 $g(n) = \Theta(f(n))$

证明过程有: 对于情况1, 有 $f(n) = O(n^{\log_b a - \varepsilon})$, 其中 $\varepsilon > 0$

则有 $f(n/b^j) = O(n^{\log_b a - \varepsilon}(n/b^j)^{\log_b a - \varepsilon})$

$$\text{于是有 } g(n) = O\left(\sum_{j=0}^{\log_b n - 1} a_j (\frac{n}{b^j})^{\log_b a - \varepsilon}\right)$$

$$= O(n^{\log_b a - \varepsilon} \cdot \sum_{j=0}^{\log_b n - 1} (\frac{a}{b^{\log_b a - \varepsilon}})^j)$$

$$= O(n^{\log_b a - \varepsilon} \cdot \sum_{j=0}^{\log_b n - 1} (a/b^{\varepsilon})^j)$$

$$= O(n^{\log_b a - \varepsilon} \cdot \sum_{j=0}^{\log_b n - 1} (b^{-\varepsilon})^j)$$

$$= O(n^{\log_b a - \varepsilon} \cdot (\frac{b^{-\varepsilon}}{b^{-\varepsilon} - 1}))$$

$$= O(n^{\log_b a - \varepsilon} \cdot \frac{n^{\varepsilon}}{b^{\varepsilon} - 1}) \quad \text{又其中 } b, \varepsilon \text{ 皆为常数}$$

$$= O(n^{\log_b a - \varepsilon} \cdot O(n^\varepsilon)) = O(n^{\log_b a})$$

于是可知情况1成立.

对于情况2, 有 $f(n) = \Theta(n^{\log_b a})$, 则 $f(n/b^j) = \Theta((n/b^j)^{\log_b a})$

$$\text{于是 } g(n) = \Theta\left(\sum_{j=0}^{\log_b n - 1} a_j (\frac{n}{b^j})^{\log_b a}\right)$$

$$= n^{\log_b a} \cdot \Theta\left(\sum_{j=0}^{\log_b n - 1} (\frac{a}{b^{\log_b a}})^j\right)$$

$$= n^{\log_b a} \cdot \Theta\left(\sum_{j=0}^{\log_b n - 1} 1\right)$$

$$= n^{\log_b a} \cdot \Theta(\log_b n) * \Theta(c \log_b n) = \Theta(c \log_b n / \log_b)$$

$$= \Theta(n^{\log_b a} \lg n)$$

于是可知情况2成立.

对于情况3, 有 $a f(n/b) \leq c f(n)$, 又 $f(n)$ 是定义在 b 的幂上的非负函数

由于 $g(n) = \sum_{j=0}^{\log_b n - 1} a_j f(n/b^j)$ 至少有一项为 $f(n)$, 且每一项均非负

$$\text{且 } g(n) = \sum_{j=0}^{\log_b n - 1} f(n/b^j)$$

则对于 $n \leq N$ 的情况, 由于皆可以视为常数, 于是归并进一个 $O(1)$ 的项

而对于 $n > N$ 的情况, 有 $f(n/b) \leq (c/a) f(n)$.

迭代 j 次, 则有 $f(n/b^j) \leq (c/a)^j f(n)$

$$\text{即 } a^j f(n/b^j) \leq c^j f(n)$$

$$\text{于是 } g(n) = \sum_{j=0}^{\log_b n - 1} a^j f(n/b^j) \leq \sum_{j=0}^{\log_b n - 1} c^j f(n) + O(1), \text{ 又 } c < 1$$

$$\leq f(n) \sum_{j=0}^{+\infty} c^j + O(1)$$

$$= (\frac{1}{1-c}) f(n) + O(1)$$

$$= O(f(n))$$

合并可知情况3中 $g(n) = \Theta(f(n))$ 成立.

合并可得引理2的证明.

Algorithm - P31

0+9 = 100% of 100%

引理3 对于常数 $a \geq 1, b > 1$, $f(n)$ 是定义在 b 的幂上的非负函数

$T(n)$ 是定义在 b 的幂上的递归式 $\left\{ \begin{array}{l} ④(c_1) \text{ if } n = 1 \\ ⑤ T(n) = aT(n/b) + f(n), n = b^i, i \in \mathbb{Z}^+ \end{array} \right.$

则有：1. 如果存在常数 $\epsilon > 0$, 有 $f(n) = O(n^{\log_b a - \epsilon})$, 则 $T(n) = O(n^{\log_b a})$

2. 如果 $f(n) = O(n^{\log_b a})$, 则 $T(n) = O(n^{\log_b a} \lg n)$

3. 如果存在常数 $\epsilon > 0$, 有 $f(n) = \Omega(n^{\log_b a + \epsilon})$

且存在常数 $c < 1, N > 0$, 使得当 $n > N$ 时, $a f(n/b) \leq c f(n)$

则 $T(n) = O(f(n))$

$$\begin{aligned} \text{证明过程有, } T(n) &= aT(n/b) + f(n) = a^2T(n/b^2) + af(n/b) + f(n) \\ &= \dots = a^{\log_b n}T(1) + a^{\log_b n-1}f(b) + \dots + af(n/b) + f(n) \\ &= n^{\log_b a}O(c_1) + \sum_{j=0}^{\log_b n-1} a^j f(n/b^j) \end{aligned}$$

则根据引理2, 令 $g(n) = \sum_{j=0}^{\log_b n-1} a^j f(n/b^j)$

对于情况1, 存在常数 $\epsilon > 0$, 使得 $f(n) = O(n^{\log_b a - \epsilon})$

则 $g(n) = O(n^{\log_b a})$

于是 $T(n) = n^{\log_b a}O(c_1) + \sum_{j=0}^{\log_b n-1} a^j f(n/b^j)$

$$= O(n^{\log_b a}) + O(n^{\log_b a}) = O(n^{\log_b a})$$

对于情况2, $f(n) = O(n^{\log_b a})$

则 $g(n) = O(n^{\log_b a} \lg n)$

于是 $T(n) = O(n^{\log_b a}) + O(n^{\log_b a} \lg n) = O(n^{\log_b a} \lg n)$

对于情况3, 存在常数 $\epsilon > 0$, 使得 $f(n) = \Omega(n^{\log_b a + \epsilon}) = \omega(n^{\log_b a})$

且存在常数 $c < 1, N > 0$, 使得当 $n > N$ 时, $a f(n/b) \leq c f(n)$

则 $g(n) = O(f(n))$

于是 $T(n) = O(n^{\log_b a}) + O(f(n)) = O(f(n))$

于是可知引理3成立,

进一步地合并3个引理可以得到主定理成立的结论

特别地对于情况3, 常数 $c < 1, N > 0$ 的存在蕴含了 $\epsilon > 0$ 的存在

基础步骤: 对于 $n = b^{\lceil \log_b N \rceil}$, $f(n)$ 和 $n^{\log_b a + \epsilon}$ 都是常数, 即 $f(n) = \Omega(n^{\log_b a + \epsilon})$

递归步骤: 假设对于 $N < x < n$, $f(x) = \Omega(x^{\log_b a + \epsilon})$ 成立,

则 $f(n) \geq \frac{a}{c} f(n/b)$

$\stackrel{(IH)}{\geq} \frac{a}{c} \cdot d \cdot (\frac{n}{b})^{\log_b a + \epsilon}$, 其中 d 是使 (d, N) 为 $f(n) = \Omega(n^{\log_b a + \epsilon})$

$\geq \frac{d}{c b^\epsilon} \cdot \frac{a}{a} \cdot n^{\log_b a + \epsilon}$, 又 $b > 1, c < 1$

则取 $\epsilon > 0$ 使 $c b^\epsilon > 1$ 即可使不等式成立

依据强归纳法可知, 存在 $\epsilon > 0$ 使得 $f(n) = \Omega(n^{\log_b a + \epsilon})$

Algorithm - P32

主定理的取整 对于主定理的证明和应用，需要从 b 的幂扩展至所有整数。
即需要考虑递归式 $T(n) = aT(\lceil n/b \rceil) + f(n)$
和 $T(n) = aT(\lfloor n/b \rfloor) + f(n)$

其中 $T(n) = aT(\lceil n/b \rceil) + f(n)$ 的下界 和 $T(n) = aT(\lfloor n/b \rfloor) + f(n)$ 的上界
可由 $\lfloor n/b \rfloor \leq n/b \leq \lceil n/b \rceil$ 方便地得到

则考虑 $T(n) = aT(\lceil n/b \rceil) + f(n)$ 的上界的证明

有递归树用的参数序列如 $n, \lceil n/b \rceil, \lceil \lceil n/b \rceil/b \rceil, \lceil \lceil \lceil n/b \rceil/b \rceil/b \rceil, \dots$

于是有递归的定义如 $n_j = \begin{cases} n & j=0 \\ \lceil n_{j-1}/b \rceil & j>0 \end{cases}$ 于是有

$$\lceil n_{j-1}/b \rceil < n_{j-1}/b + 1$$

即有 $n_0 = n, n_1 < n_0/b + 1, n_2 < n/b^2 + 1/b + 1, \dots$

于是有 $n_j < n/b^j + \sum_{i=0}^{j-1} \frac{1}{b^i} < n/b^j + \sum_{i=0}^{\infty} \frac{1}{b^i} = \frac{n}{b^j} + \frac{b}{b-1}$

令 $j = \lceil \log_b n \rceil$ ，即 $\lceil \log_b n \rceil > \log_b n - 1, b^{\lceil \log_b n \rceil} > b^{\log_b n - 1}$

$$\begin{aligned} n_j &< \frac{n}{b^j} + \frac{b}{b-1} = n/b^{\lceil \log_b n \rceil} + \frac{b}{b-1} < n/b^{\log_b n - 1} + \frac{b}{b-1} \\ &= n/n \cdot b^{-1} + \frac{b}{b-1} = b + \frac{b}{b-1} = O(1) \end{aligned}$$

于是可知在递归树高度为 $\lceil \log_b n \rceil$ 的层时，最多为常数规模

则可知在 n 为任意整数的情形下，
递归树与 n 为 b 的幂的递归树的差别仅在于高度为 $\lceil \log_b n \rceil$ ，即有 $\lceil \log_b n \rceil + 1$ 层

于是有递归式 $T(n) = \Theta(n^{\log_b a}) + \sum_{j=0}^{\lceil \log_b n \rceil - 1} a^j f(n_j)$

对于求和式 $\sum_{j=0}^{\lceil \log_b n \rceil - 1} a^j f(n_j)$ 的分析可参照 b 的幂的情形

对于情形 1，即如果有 $\epsilon > 0$ ，使得 $f(n) = \Omega(n^{\log_b a + \epsilon})$

且存在常数 $c < 1$ ，使得对于足够大的 n 有 $a f(n) \leq c f(n)$

如果取 $n > b + \frac{b}{b-1}$ ，有常数 $c < 1$ 使不等式成立，则有 $a^j f(n/b^j) \leq c^j f(n)$

于是可用与 n 为 b 的幂次相似的方法证明 $T(n) = \Theta(f(n))$

对于情形 2，即假设设有 $f(n) = \Theta(n^{\log_b a})$

且如果有 $f(n_j) = O(n^{\log_b a / a_j}) = O((n/b^j)^{\log_b a})$ ，则可以利用 n 为 b 的幂次的方法

由于 $j \leq \lceil \log_b n \rceil$ ，即 $b^j/n \leq 1$ ，则对于足够大的 n_j

$$\begin{aligned} f(n_j) &\leq c \left(\frac{n}{b^j} + \frac{b}{b-1} \right)^{\log_b a} = c \cdot \left[\frac{n}{b^j} \cdot \left(1 + \frac{b}{n(b-1)} \right) \right]^{\log_b a} \\ &\leq c \cdot \left(\frac{n}{b^j} \right)^{\log_b a} \cdot \left(1 + \frac{b}{b-1} \right)^{\log_b a} \end{aligned}$$

又 $c \cdot \left(1 + \frac{b}{b-1} \right)^{\log_b a}$ 为常数

则 $f(n_j) \leq O((n/b^j)^{\log_b a})$

进而可得到 当 $f(n) = \Theta(n^{\log_b a})$ 时， $T(n) = \Theta(n^{\log_b a} \lg n)$

对于情形 1，可以利用与情形 2 相似的方法得到

于是得到主定理扩展至向上取整形式的证明。

可对 $T(n) = aT(\lceil n/b \rceil) + f(n)$ 利用类似方法以得到向下取整形式的证明

Algorithm - P33

How to find it?

GF - Generating Function

$$F(z) = \sum_{i=0}^{\infty} f_i z^i = 0 + z + z^2 + 2z^3 + 3z^4 + 5z^5 + 8z^6 + \dots$$

(称为斐波那契生成函数(generating function), 是一种形式幂级数)

$$F(z) = z + zF(z) + z^2 F(z)$$

$$\text{证明过程有, } z + zF(z) + z^2 F(z) = z + 0 + z^2 + z^3 + 2z^4 + \dots \\ + 0z^2 + z^3 + z^4 + \dots$$

$$= z + \sum_{i=2}^{\infty} (f_{i-1} + f_{i-2}) z^i$$

$$= f_0 \cdot z^0 + f_1 \cdot z^1 + \sum_{i=2}^{\infty} f_i z^i$$

$$= F(z)$$

$$F(z) = \frac{z}{1-z-z^2} = \frac{z}{(1-\phi z)(1-\hat{\phi} z)} = \frac{1}{\sqrt{5}} \left(\frac{1}{1-\phi z} - \frac{1}{1-\hat{\phi} z} \right), \text{ 其中 } \phi = \frac{\sqrt{5}+1}{2}, \hat{\phi} = \frac{1-\sqrt{5}}{2}$$

证明过程有: 由于 $F(z) = z + zF(z) + z^2 F(z)$

$$(1-z-z^2) F(z) = z$$

又对于 $z \in \mathbb{C}$ 有 $1-z-z^2 \neq 0$

$$\text{于是 } F(z) = \frac{z}{1-z-z^2}$$

$$\text{又 } (-\phi) \cdot (-\hat{\phi}) = \frac{(-1-\sqrt{5})(\sqrt{5}-1)}{4} = -1$$

$$(-\phi) + (-\hat{\phi}) = \frac{(-1-\sqrt{5}) + (\sqrt{5}-1)}{2} = -1$$

$$\text{于是 } 1-z-z^2 = (1-\phi z)(1-\hat{\phi} z)$$

$$\text{又 } \phi - \hat{\phi} = \frac{(1+\sqrt{5}) - (1-\sqrt{5})}{2} = \sqrt{5}$$

$$\text{于是 } F(z) = \frac{z}{1-z-z^2} = \frac{z}{(1-\phi z)(1-\hat{\phi} z)}$$

$$= \frac{1}{\sqrt{5}} \cdot \frac{(\phi - \hat{\phi}) z}{(1-\phi z)(1-\hat{\phi} z)} = \frac{1}{\sqrt{5}} \left(\frac{1}{1-\phi z} - \frac{1}{1-\hat{\phi} z} \right)$$

$$F(z) = \sum_{i=0}^{\infty} \frac{1}{\sqrt{5}} (\phi^i - \hat{\phi}^i) z^i, \text{ 即 } f_i = \frac{1}{\sqrt{5}} (\phi^i - \hat{\phi}^i)$$

证明过程有: 根据级数展开有

$$\frac{1}{1-\phi z} = 1 + \phi z + \phi^2 z^2 + \dots = \sum_{i=0}^{\infty} (\phi z)^i$$

$$\frac{1}{1-\hat{\phi} z} = 1 + \hat{\phi} z + \hat{\phi}^2 z^2 + \dots = \sum_{i=0}^{\infty} (\hat{\phi} z)^i$$

$$\text{于是有 } F(z) = \frac{1}{\sqrt{5}} \left(\frac{1}{1-\phi z} - \frac{1}{1-\hat{\phi} z} \right)$$

$$= \sum_{i=0}^{\infty} \frac{1}{\sqrt{5}} (\phi^i - \hat{\phi}^i) z^i, \text{ 比较系数得 } f_i = \frac{1}{\sqrt{5}} (\phi^i - \hat{\phi}^i)$$

$$f_i = \lfloor \frac{\phi^i}{\sqrt{5}} + \frac{1}{2} \rfloor, \text{ 即 } \phi^i/\sqrt{5} \text{ 舍入至最接近的整数, 其中 } i > 0$$

证明过程有, $|\hat{\phi}| = |\frac{\sqrt{5}-1}{2}| < 1$, 于是有 对于任意正整数 i , 有 $|\hat{\phi}|^i = |\hat{\phi}|^i < 1$

$$\text{于是有 } |\hat{\phi}|^i/\sqrt{5} < 1/\sqrt{5} < \frac{1}{2}, \text{ 而 } f_i = \phi^i/\sqrt{5} - \hat{\phi}^i/\sqrt{5} < \phi^i/\sqrt{5} + \frac{1}{2} + (\frac{1}{2} - \hat{\phi}^i/\sqrt{5})$$

又 f_i 为整数, 且 $\frac{1}{2} - \hat{\phi}^i/\sqrt{5} > 0$ 特别地有 $f_0 = 0 = \lfloor \frac{1}{\sqrt{5}} + \frac{1}{2} \rfloor$

则有 $f_i = \lfloor \frac{\phi^i}{\sqrt{5}} + \frac{1}{2} \rfloor$, 即 $\phi^i/\sqrt{5}$ 舍入至最接近的整数, 其中 $i > 0$, 对 $i=0$ 也成立

Algorithm - P34

芯片检测

对于 n 片可能完全一样的集成电路芯片，原理上可以相互检测，以找到一块好的芯片。

假设有测试夹具可以同时检测两块芯片。

两块芯片都会检测另一块并报告另一块是好或坏。

好的芯片总是给出正确的报告，但坏的芯片并不总是给出错误的报告。

即可能的结果包括：

芯片 A	芯片 B	结论
B 是好的	A 是好的	或者两块都好，或者两块都坏
B 是好的	A 是坏的	至少有一块是坏的
B 是坏的	A 是好的	至少有一块是坏的
B 是坏的	A 是坏的	至少有一块是坏的

可能

注意：如果存在超过 $n/2$ 块芯片是坏的，且坏芯片会合谋欺骗。

即坏芯片会始终报告好芯片是坏的，而坏芯片是好的。

则可能的结果只剩两种，

两块都报告另一块是好的，结论是或者两块都好，或者两块都坏。

两块都报告另一块是坏的，结论是必有一块好的和一块坏的。

但是采用逐对检测的策略只能将芯片划分为好坏两类，则无法判断哪一类是好的。

如果有超过 $n/2$ 的芯片是好的，且坏芯片不会合谋。

则采用逐对检测以找到一块好的芯片，进行 $\lfloor n/2 \rfloor$ 次检测可将问题规模减半。
算法过程为：

当 $n=1$ 或 $n=2$ 时，根据假设有其中全部是好的芯片。

当 $n=3$ 时，至多有一片坏芯片。于是如果返回 AB 都好，则可确信 AB 都好。

任取两块检测，如果报告为都是好的，则从中任取一片。

如果报告为一好一坏，则取第三片。

当 $n \geq 4$ 时，假设其中有 k 块好的， $n-k$ 块坏的。则有 $k > n-k$

两两比较则有， m 对报告都是好的， l 对报告其他。

则从样本中去掉 l 对芯片，然后从 m 对中每对去掉一片。

则剩余的 m (或 $m+1$) 片中有半数以上是好的。

重复执行直到角及基础情形。

于是可以定义递归式 $T(n) = \begin{cases} \Theta(1), & n=1, 2, 3 \\ T(n/2) + \lfloor n/2 \rfloor, & n > 3 \end{cases}$

符合主定理的情况3，于是有 $T(n) = \Theta(\lfloor n/2 \rfloor) = \Theta(n)$