

Algorithm - P12

渐近效率 指当输入规模无限增加时，在极限中算法运行时间如何随输入规模变大而增加

渐近非负 指对于函数 $f(n)$, 当 n 足够大时, $f(n)$ 非负, 即 $\exists N > 0 \forall n (n > N \rightarrow f(n) \geq 0)$

渐近正函数 指对于函数 $f(n)$, 当 n 足够大时, $f(n)$ 为正, 即 $\exists N > 0 \forall n (n > N \rightarrow f(n) > 0)$

渐近上界 (bounded above asymptotically), 对于给定的函数 $g(n)$, 用 $O(g(n))$ 表示函数集合
即有 $O(g(n)) = \{f(n) \mid \exists c > 0, n_0 > 0 \forall n (n \geq n_0 \rightarrow 0 \leq f(n) \leq c g(n))\}$

渐近下界 (bounded below asymptotically), 对于给定的函数 $g(n)$, 用 $\Omega(g(n))$ 表示函数集合
即有 $\Omega(g(n)) = \{f(n) \mid \exists c > 0, n_0 > 0 \forall n (n \geq n_0 \rightarrow 0 \leq c g(n) \leq f(n))\}$

渐近紧确界 (asymptotically tight bound), 对于给定的函数 $g(n)$, 用 $\Theta(g(n))$ 表示函数集合
即有 $\Theta(g(n)) = \{f(n) \mid \exists c_1, c_2 > 0, n_0 > 0 \forall n (n \geq n_0 \rightarrow c_1 g(n) \leq f(n) \leq c_2 g(n))\}$

注意在以上定义中并未如离散数学中使用 $|f(n)|$, 因为要求集合中的 $f(n)$ 均为渐近非负的

等式/不等式 当渐近记号 $\Theta(g(n))$ 独立于等式(不等式)右边时, 表示一种集合的成员关系

中的 $\Theta(g(n))$, 即用 $f(n) = \Theta(g(n))$ 表示 $f(n) \in \Theta(g(n))$

而 $f(n) > \Theta(g(n))$ 表示 $\exists h(n) (h(n) = \omega(g(n)) \wedge f(n) = \Theta(h(n)))$

而 $f(n) < \Theta(g(n))$ 表示 $\exists h(n) (h(n) = o(g(n)) \wedge f(n) = \Theta(h(n)))$

而当渐近记号 $\Theta(g(n))$ 出现于表达式之中, 则代表某个无需关注名字的匿名函数 (anonymous function)

如在递归表达式 $T(n) = 2T(n/2) + \Theta(n)$,

即没有必要准确说明所有低阶项, 而理解力都包含于 $\Theta(n)$ 表示的匿名函数

则可使用渐近记号 $\Theta(g(n))$ 来消除一个等式中无关紧要的细节与混乱

而如渐近记号 $\Theta(g(n))$ 同时出现于等式或不等式的左右两侧, 如 $2h^2 + \Theta(n) = \Theta(n^2)$

表示无论如何选择等号左侧的匿名函数, 表达式所用的函数都是等式右侧函数集合的成员

或是原有的一个等号右侧的匿名函数使等式成立

渐近小于 (dominated asymptotically), 对于给定的函数 $g(n)$, 用 $o(g(n))$ 表示函数集合

即有 $o(g(n)) = \{f(n) \mid \forall c > 0 \exists n_0 > 0 \forall n (n \geq n_0 \rightarrow 0 \leq f(n) < c g(n))\}$, 即 $\lim_{n \rightarrow \infty} f(n)/g(n) = 0$

渐近大于 (dominate asymptotically), 对于给定的函数 $g(n)$, 用 $\omega(g(n))$ 表示函数集合

即有 $\omega(g(n)) = \{f(n) \mid \forall c > 0 \exists n_0 > 0 \forall n (n \geq n_0 \rightarrow 0 \leq c g(n) < f(n))\}$, 即 $\lim_{n \rightarrow \infty} f(n)/g(n) = \infty$

Algorithm - P13

渐近性质

渐近性质

传递性

$$f(n) = \Theta(g(n)) \wedge g(n) = \Theta(h(n)) \rightarrow f(n) = \Theta(h(n))$$

$$f(n) = O(g(n)) \wedge g(n) = O(h(n)) \rightarrow f(n) = O(h(n))$$

$$f(n) = \Omega(g(n)) \wedge g(n) = \Omega(h(n)) \rightarrow f(n) = \Omega(h(n))$$

$$f(n) = o(g(n)) \wedge g(n) = o(h(n)) \rightarrow f(n) = o(h(n))$$

$$f(n) = \omega(g(n)) \wedge g(n) = \omega(h(n)) \rightarrow f(n) = \omega(h(n))$$

自反性

$$f(n) = \Theta(f(n)), f(n) = O(f(n)), f(n) = \Omega(f(n))$$

对称性

$$f(n) = \Theta(g(n)) \leftrightarrow g(n) = \Theta(f(n))$$

转置对称性

$$f(n) = O(g(n)) \leftrightarrow g(n) = \Omega(f(n))$$

$$f(n) = o(g(n)) \leftrightarrow g(n) = \omega(f(n))$$

与实数比较类比

两个函数 $f(n)$ 与 $g(n)$ 的渐近比较，与实数 a 和 b 的比较有类比

$$\begin{array}{c|c|c|c|c|c} f(n) & = \Theta(g(n)) & = O(g(n)) & = \Omega(g(n)) & = o(g(n)) & = \omega(g(n)) \\ \hline a & = b & \leq b & \geq b & < b & > b \end{array}$$

三分性

对任意实数 a, b , $a > b$, $a = b$, $a < b$ 三个命题有且恰有一个成立

或者 $a \geq b$, $a \leq b$ 中至少有一个命题成立

特别注意：对于函数 $f(n)$ 和 $g(n)$ 的渐近比较

$f(n) = O(g(n))$ 和 $f(n) = \Omega(g(n))$ 可能都不成立

如对于 $f(n) = n$ 和 $g(n) = n^{1+\sin n}$

由于 $1 + \sin n$ 在 $n \rightarrow \infty$ 时，始终在 $[0, 2]$ 上振荡

所以 $n^{1+\sin n}$ 始终在 $g(n) = 1$ 和 $g(n) = n^2$ 之间振荡

因此 $f(n) = O(g(n))$ 和 $f(n) = \Omega(g(n))$ 均不成立

多参数函数比较 对于给定的渐近非负的多参数函数 $g(n, m)$

定义 渐近上界 $O(g(n, m)) = \{f(n, m) \mid \exists c > 0, n_0 > 0, m_0 > 0 \forall n, m (n \geq n_0 \wedge m \geq m_0 \rightarrow 0 \leq f(n, m) \leq c g(n, m))\}$

渐近下界 $\Omega(g(n, m)) = \{f(n, m) \mid \exists c, n_0, m_0 > 0 \forall n, m (n \geq n_0 \wedge m \geq m_0 \rightarrow 0 \leq c g(n, m) \leq f(n, m))\}$

渐近紧确界 $\Theta(g(n, m)) = \{f(n, m) \mid \exists c_1, c_2, n_0, m_0 > 0 \forall n, m (n \geq n_0 \wedge m \geq m_0 \rightarrow 0 \leq c_1 g(n, m) \leq f(n, m) \leq c_2 g(n, m))\}$

向上取整/向下取整 对任意 $x \in \mathbb{R}, a, b \in \mathbb{Z}^+$, 有 $\lceil \frac{ax}{b} \rceil = \lceil \frac{x}{ab} \rceil$ 和 $\lfloor \frac{ax}{b} \rfloor = \lfloor \frac{x}{ab} \rfloor$

且有 $\lceil \frac{a}{b} \rceil \leq \frac{a+cb-1}{b}$, $\lfloor \frac{a}{b} \rfloor \geq \frac{a-cb-1}{b}$

Algorithm - P14

Stirling's approximation

对实数常量 $a > 1$ 和 b , 有 $\lim_{n \rightarrow \infty} \frac{n^b}{a^n} = 0$, 即 $n^b = o(a^n)$

即任何大于1的指数函数比任意多项式函数增长更快

$$e^x = 1 + x + \frac{1}{2!}x^2 + \frac{1}{3!}x^3 + \dots = \sum_{i=0}^{\infty} \frac{x^i}{i!}$$

则对任意 $x \in \mathbb{R}$, 有 $e^x \geq 1 + x$. 注意仅在 $x=0$ 时取得等号

而当 $|x| \leq 1$ 时, 则有近似估计 $1+x \leq e^x \leq 1+x+x^2$.

当 $x \rightarrow 0$ 时, $e^x = 1+x+x^2+o(x^2) = 1+x+O(x^2)$

考虑 $\lim_{n \rightarrow \infty} (1+\frac{x}{n})^n$, 当 $x \neq 0$ 时 $\lim_{n \rightarrow \infty} (1+\frac{x}{n})^{\frac{n}{x} \cdot x} = e^x$

取 $t = \frac{n}{x}$, 则当 $n \rightarrow \infty$ 时, $t \rightarrow \infty$

则有 $\lim_{n \rightarrow \infty} (1+\frac{x}{n})^{\frac{n}{x} \cdot x} = (\lim_{t \rightarrow \infty} (1+\frac{1}{t})^t)^x = e^x$

当 $x=0$ 时, $\lim_{n \rightarrow \infty} (1+0)^n = 1 = e^0$, 则 $\lim_{n \rightarrow \infty} (1+\frac{x}{n})^n = e^x$ 对任意 $x \in \mathbb{R}$ 成立

对数记号

定义对数记号 $\lg^k x = \underbrace{\lg(\lg(\dots\lg)}_{k \text{ 个}} x$ (取幂)

$\lg(\lg x) = \lg(\lg x)$ (复合)

对于实数 $a > 0, b > 0, c > 0$ 且 $b \neq 1$, 则有 $a^{\log_b c} = c^{\log_b a}$

证明如, $a^{\log_b c} = (b^{\log_b a})^{\log_b c} = (b^{\log_b c})^{\log_b a} = c^{\log_b a}$

当实数 $|x| < 1$ 时, $\ln(1+x)$ 存在一种简单的级数展开

$$\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \dots = \sum_{i=1}^{\infty} (-1)^{i-1} x^i / i$$

对实数 $x > -1$, 有 $\frac{x}{1-x} \leq \ln(1+x) \leq x$, 注意仅当 $x=0$ 时取得等号

的对数有界 若对实数常量 k , 有 $f(n) = O(\lg^k n)$, 则称 $f(n)$ 是多对数有界的

对任意常量 $a > 0$, 有 $\lg^a n = o(n^a)$

即任意的正多项式函数比任意多对数函数增长更快

已知 $\lim_{n \rightarrow \infty} \frac{n^b}{a^n} = 0$, 其中 $a > 1, b \in \mathbb{R}$

则以 \lg^n 代替 n , 以 2^a 代替 a , 则有 $\lim_{n \rightarrow \infty} \frac{\lg^b n}{(2^a)^{\lg^n}} = \lim_{n \rightarrow \infty} \frac{\lg^b n}{n^a} = 0$

斯特林近似公式 (Stirling's approximation), 是对阶乘 $n!$ 在 $n \rightarrow \infty$ 时的一个近似

即有 $n! = \sqrt{2\pi n} \left(\frac{n}{e}\right)^n (1 + O(\frac{1}{n}))$, 同时有 $|n!| \sim \sqrt{2\pi n} \left(\frac{n}{e}\right)^n (n \rightarrow \infty)$

对于 $n \geq 1$, 则有 $n! = \sqrt{2\pi n} \left(\frac{n}{e}\right)^n e^{an}$, 其中 $\frac{1}{12n+1} < a_n < \frac{1}{12n}$

Algorithm - P15

6.1.1 - multimap

多重函数

用记号 $f^{(i)}(n)$ 表示函数 $f(n)$ 重复 i 次作用于一个初值 n 上

形式化的定义为：对于定义于实数集 \mathbb{R} 上的函数 $f(n)$

$$f^{(i)}(n) = \begin{cases} n & i=0 \\ f(f^{(i-1)}(n)), & i>0 \end{cases}$$

应用时特别注意： $f^{(n-1)}(n)$ 是否还在 $f(n)$ 的定义域中

$\text{polyFunc} ::= (\text{Num } a) \Rightarrow (a \rightarrow a) \rightarrow a \rightarrow \text{Int} \rightarrow a$

$\text{polyFunc } n \ 0 = n$

$\text{polyFunc } f \ n \ i = \text{polyFunc } f \ (f \ n) \ (i-1)$

多重对数函数 (iterated logarithm). 又称迭代对数，记为 $\lg^* n$ ，通常使用 $\lg^* n$

迭代对数 注意对数函数对非正数无定义，即仅当 $\lg^{(i-1)} n > 0$ 时， $\lg^{(i)} n$ 有定义

定义 $\lg^* n = \min\{i \geq 0 \mid \lg^{(i)} n \leq 1\}$

($\lg^* n$) 又可形式化地定义为，对于定义在实数集 \mathbb{R} 上的函数 $\lg^* n$

$$\lg^* n = \begin{cases} 0 & n \leq 1 \\ 1 + \lg^*(\lg n), & n > 1 \end{cases}$$

以图形的求解方法如下：在 x 轴上取点 $(n, 0)$

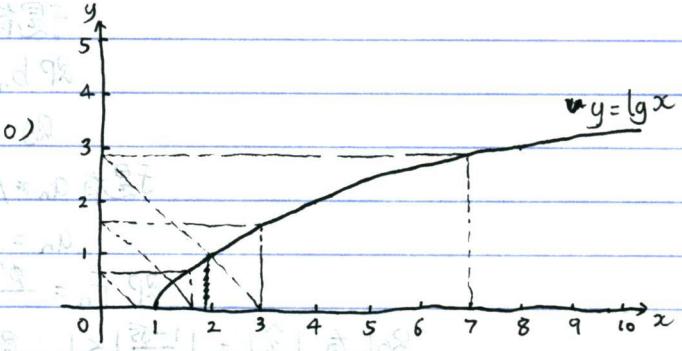
① 作 $x=n$ 交 $y=\lg n$ 于 $(n, \lg n)$ 处

② 作 $y=\lg n$ 交 y 轴于 $(0, \lg n)$ 处

③ 作 $y=\lg n - x$ 交 x 轴于 $(\lg n, 0)$

重复以上步骤直至交点在 $(-\infty, 1]$ 之间

终止时 y 轴上的交点数即为 $\lg^* n$ 的值 即有 $\lg^* 7 = 3$



注意：多重对数函数是一个增长非常慢的函数

$x \in \mathbb{R}$	$(-\infty, 1]$	$(1, 2]$	$(2, 4]$	$(4, 16]$	$(16, 65536]$	$(65536, 2^{65536}]$	\dots
$\lg^* x$	0	1	2	3	4	5	\dots

note: $\lg(2^{65536}) = 65536 \rightarrow \lg(2^{65536}) = 16$

$\text{iterLog} ::= (\text{Floating } a, \text{Ord } a) \Rightarrow a \rightarrow \text{Int}$

$\text{iterLog } n_0 =$

$\text{let func acc } n \ 0 = \text{const } \text{and } \text{func } acc \leftarrow \text{func } acc + 1 \text{ in } \text{func } acc$

$| n \leq 1 \text{ 时 } = acc \quad | n > 1 \text{ 时 } = acc + \lfloor \lg n / \lg 2 \rfloor] n \leq 1 \text{ 时返回 acc}$

$\text{func } acc + \lfloor \lg n / \lg 2 \rfloor] n > 1 \text{ 时 递归地调用}$

$\text{in func } 0 \ n_0 \text{ at exit } acc \neq n_0 \neq \text{not } \text{func } acc + 1 \ (lg n)$

有 $\text{iterLog } 65537 \rightarrow 5$ ，注意基本上很少遇到大于 2^{65536} 的数字

Algorithm - P16

已知 $F_n = \frac{\phi^n - \hat{\phi}^n}{\sqrt{5}}$

斐波那契数，定义为 $F_0 = 0, F_1 = 1, F_n = F_{n-1} + F_{n-2}, n \in \mathbb{Z}^+$ 且 $n > 1$

与黄金分割率(golden ratio) ϕ 及其共轭数(conjugate) $\hat{\phi}$ 有关

ϕ 与 $\hat{\phi}$ 是 $x^2 = x + 1$ 的两个根，有 $\phi = \frac{1+\sqrt{5}}{2}$, $\hat{\phi} = \frac{1-\sqrt{5}}{2}$

有 $F_i = \frac{\phi^i - \hat{\phi}^i}{\sqrt{5}}$, 其中 $i \in \mathbb{N}$

证明有，基础步骤聚： $\frac{\phi^0 - \hat{\phi}^0}{\sqrt{5}} = \frac{1-1}{\sqrt{5}} = 0 = F_0$

$\frac{\phi^1 - \hat{\phi}^1}{\sqrt{5}} = \frac{2\sqrt{12}}{\sqrt{5}} = 1 = F_1$

递归步骤聚：假设 $\forall n \in \mathbb{Z}^+ \text{ 且 } n > 1, F_n = F_{n-2} + F_{n-1}$,

则构造等比数列 $a_n + \alpha a_{n-1} = \beta(a_{n-1} + \alpha a_{n-2})$

于是有 $a_n = (\beta - \alpha)a_{n-1} + \beta\alpha a_{n-2}$, 即 $\beta - \alpha = 1$ 且 $\beta\alpha = 1$

取 $\alpha > 0$ 且 $\beta > 0$, 则有 $\alpha = \frac{\sqrt{5}-1}{2}, \beta = \frac{\sqrt{5}+1}{2}$

于是有 $a_2 + \alpha a_1 = 1 + \alpha = \beta, a_{n+1} + \alpha a_n = \beta^{n-1}(a_2 + \alpha a_1) = \beta^n$

于是有 $\frac{a_{n+1}}{\beta^{n+1}} + \frac{\alpha}{\beta} \cdot \frac{a_n}{\beta^n} = \frac{1}{\beta}$, 则取数列 $\{b_n\}$, 有 $b_n = \frac{a_n}{\beta^n}$

则有 $b_{n+1} + \frac{\alpha}{\beta} b_n = \frac{1}{\beta}$, 设有 $\lambda \in \mathbb{R}$, $b_{n+1} + \lambda = -\frac{\alpha}{\beta}(b_n + \lambda)$

于是有 $\lambda = -\frac{1}{\alpha + \beta} > 0$, 又 $\alpha + \beta \neq 0$, 于是有 $\{b_n + \lambda\}$ 是等比数列

即 $b_n + \lambda = \left(\frac{-\alpha}{\beta}\right)^{n-1}(b_1 + \lambda)$, 又 $b_1 = \frac{a_1}{\beta} = \frac{1}{\beta}$

则 $a_n / \beta^n = \left(-\frac{\alpha}{\beta}\right)^{n-1}\left(\frac{1}{\beta} + \lambda\right) - \lambda$

于是有 $a_n / \beta^n = [\beta^n - (-\alpha)^n] / \beta^n (\alpha + \beta)$

$a_n = \left[\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right] / \sqrt{5}$

即 $F_n = \frac{\phi^n - \hat{\phi}^n}{\sqrt{5}}$, 对于任何 $n \in \mathbb{Z}^+$ 成立

另外有 $|\hat{\phi}| = |\frac{1-\sqrt{5}}{2}| < 1$, 即有 $|\hat{\phi}|^i < 1$

于是 $\frac{|\hat{\phi}|^i}{\sqrt{5}} < \frac{1}{\sqrt{5}} < \frac{1}{2}$, 而 $F_{ni} = \frac{\phi^i}{\sqrt{5}} - \frac{\hat{\phi}^i}{\sqrt{5}} < \frac{\phi^i}{\sqrt{5}} - \frac{\hat{\phi}^i}{\sqrt{5}} + 1 = \frac{\phi^i}{\sqrt{5}} + \frac{1}{2} + \left(\frac{1}{2} - \frac{\hat{\phi}^i}{\sqrt{5}}\right)$

又 F_{ni} 为整数, $\frac{1}{2} - \frac{\hat{\phi}^i}{\sqrt{5}} > 0$

则 $F_{ni} = \lfloor \frac{\phi^i}{\sqrt{5}} + \frac{1}{2} \rfloor$, 其中 $i \in \mathbb{Z}^+$, 即 F_i 为 $\phi^i / \sqrt{5}$ 最接近的整数

(Ω 无穷), 用 $f(n) = \underline{\Omega}(g(n))$ 表示存在正常量 c , 使得有无穷多个整数 n , 有 $0 \leq cg(n) \leq f(n)$

即有 $f(n) = \underline{\Omega}(g(n)) \leftrightarrow \exists c > 0 \text{ 有无穷多个 } n \in \mathbb{Z}^+, 0 \leq cg(n) \leq f(n)$

对于三分性, 用 $\underline{\Omega}(g(n))$ 替代 $\underline{\Omega}(g(n))$, 则有对任意两个函数 $f(n)$ 和 $g(n)$

有 $f(n) = O(g(n))$ 和 $f(n) = \underline{\Omega}(g(n))$ 至少满足其一

证明有, 对任意函数 $f(n)$ 和 $g(n)$, 若有 $f(n) = O(g(n)) \vee f(n) = \underline{\Omega}(g(n))$ 为真

则有 $f(n) \neq O(g(n)) \wedge f(n) \neq \underline{\Omega}(g(n))$ 为假

$\Leftrightarrow (f(n) = O(g(n)) \Leftrightarrow \exists c > 0 \exists N > 0 \forall n (n > N \rightarrow f(n) \leq cg(n))$

$\Leftrightarrow f(n) \neq O(g(n)) \Leftrightarrow \forall c > 0 \forall N > 0 \exists n (n > N \wedge f(n) > cg(n))$

即如果 $f(n) \neq O(g(n))$ 则必有 $f(n) = \underline{\Omega}(g(n))$, 于是两者必有其一

Algorithm - P17

O'

指与大O记号稍有不同的定义，通常用于更广泛的数学问题。

$$O'(g(n)) = \{f(n) \mid \text{存在正常量 } C, n_0, \text{使得对任意 } n \geq n_0, 0 \leq |f(n)| \leq Cg(n)\}$$

$$\text{即有 } f(n) = O'(g(n)) \iff |f(n)| = O(g(n))$$

如果 $g(n)$ 为渐近非负的函数，且有 $f(n) = \Omega(g(n))$ ，则 $f(n)$ 也是渐近非负的函数。

$$\text{则 } f(n) = \Omega(g(n)) \wedge f(n) = O'(g(n)) \rightarrow f(n) = \Theta(g(n))$$

但是注意其逆命题不成立。

\tilde{O}

称为软O记号，用于表示忽略对数因子的大O记号。

$$\tilde{O}(g(n)) = \{f(n) \mid \text{存在正常量 } C, k, n_0, \text{使得对任意 } n \geq n_0, 0 \leq f(n) \leq Cg(n) \cdot \lg^k n\}$$

$$\text{即 } f(n) = \tilde{O}(g(n)) \iff \exists c > 0, k > 0, n_0 > 0 \forall n (n \geq n_0 \rightarrow 0 \leq f(n) \leq c \lg^k n \cdot g(n))$$

可由此分别定义 $\tilde{\Omega}$ 记号与 $\tilde{\Theta}$ 记号，对于渐近非负函数 $g(n)$

$$\tilde{\Omega}(g(n)) = \{f(n) \mid \text{存在正常量 } c, k, n_0, \text{使得对任意 } n \geq n_0, 0 \leq c \cdot \lg^k n \cdot g(n) \leq f(n)\}$$

$$\tilde{\Theta}(g(n)) = \{f(n) \mid \text{存在正常量 } c_1, k_1, c_2, k_2, n_0, \text{使得对任意 } n \geq n_0, 0 \leq c_1 \lg^{k_1} n \cdot g(n) \leq f(n) \leq c_2 \lg^{k_2} n \cdot g(n)\}$$

于是参考大O记号、大Ω记号与大Θ记号的关系可知

$$f(n) = \tilde{O}(g(n)) \wedge f(n) = \tilde{\Omega}(g(n)) \iff f(n) = \tilde{\Theta}(g(n))$$

多重函数

对于定义在实数集上的任意单调递增函数 $f(n)$ 应用重复操作符 *

$$\text{对于给定的常量 } c \in \mathbb{R}, f_c^*(n) = \min \{i \geq 0 \mid f^{(i)}(n) \leq c\}$$

$$\text{即有正整数 } i = f_c^*(n), \text{使得 } f^{(i)}(n) \leq c \wedge f^{(i+1)}(n) > c$$

于是有值 $f_c^*(n)$ 为缩小其参数 n 至小于或等于 c 所需要函数 f 重复应用的数目

注意：这个函数 $f(n)$ 并不需要是良定义的。

但是注意需要满足两个条件，对于给定的函数 $f(n)$ 和常量 c

在 $n > c$ 时，应有 $f(n) < n$ ，即对于 $k \in \mathbb{N}$, $f^{(k)}(n) > f^{(k+1)}(n)$

$\lim_{k \rightarrow \infty} f^{(k)}(n)$ 的极限或者存在且小于等于 c ，或者为负无穷大

或者存在一个正整数 k 使得 $f^{(k)}(n) \leq c$ ，且 $f^{(k+1)}(n)$ 无定义

如令函数 $f(n) = \sqrt{n}$ ，则 $f_2^*(n) = \Theta(\lg \lg n)$

$$\text{但是 } f_1^*(n) = \begin{cases} 0, & n \leq 1 \\ \infty, & n > 1 \end{cases} \text{ 由于 } \forall n > 1, \sqrt{n} > 1$$

iterFunc :: (Num a, Ord a) => (a → a) → a → a → Int

iterFunc f c n =

let func acc n

$$\quad | n \leq c = acc$$

$$\quad | \text{otherwise} = func(acc + 1)(f n)$$

in func 0 n

Algorithm - P18

在使用分治策略递归地求解一个问题时，每层递归有三个步骤：

分解 (divide) 步骤将问题递归分为一些子问题，子问题的形式与原问题一样，但规模更小。

解决 (conquer) 步骤递归地求解子问题，当子问题是足够小时，则无需递归直接求解。

合并 (combine) 步骤将子问题的解组合成原问题的解。

分治策略面对的问题分类：

递归情况 (recursive case)，指当问题规模足够大时，需要递归地求解。

基本情况 (base case)，指当问题规模足够小，而不再需要递归地求解。

注意：有时候子问题并非与原问题形式一样而规模更小的问题，而是形式不同的子问题。

而是形式不同的子问题，将对这些子问题是的求解看作合并步骤的一部分。

递归式 (recurrence)，指一个等式或不等式，通过更小的输入上的函数值来描述更大的输入上的函数。

$$\text{如 MERGE-SORT 过程的最坏情况运行时间 } T(n) = \begin{cases} \Theta(1), & n=1 \\ 2T(n/2) + \Theta(n), & n>1 \end{cases}$$

另外，递归算法可能将问题划分为规模不同的子问题。

$$\text{如递归式 } T(n) = T(2n/3) + T(n/3) + \Theta(n)$$

划分的子问题也可能不是原问题是规模的一个固定比例。

如对于线性搜索的递归版本，有 $T(n) = T(n-1) + \Theta(1)$

这里蕴含了循环倍数在遇输入上与递归倍数相同。

所以如 Haskell 即没有循环倍数而全部以递归倍数实现。

求解递归式的三种方法，得出算法的 Θ 或 O 表示界的方法。

代入法，指先猜测一个界，然后用数学归纳法证明这个渐近界是正确的。

递归树法，将递归式转换为有根树，用结点表示不同层次的递归调用产生的代价，然后采用边界和技术解决。

注意：通常内点表示递归情况，内点的值表示分解与合并所需的代价。

而叶结点表示基础情况，叶结点的值表示求解该基础情况所需的代价。

主方法 (master method)，用于求解形如 $T(n) = aT(n/b) + f(n)$ 的递归式。

其中参数 $a \geq 1$ ，表示将原问题分解为 a 个与原问题是形式一样的子问题。

$b > 1$ ，表示分解而成的子问题是原问题规模为原问题是 $1/b$ 。

$f(n)$ 为一个不同于递归问题是的函数，用于描述分解与合并所需的代价。

虽然主方法是一个单独的方法，但可以与代入法和递归树法相结合。

另外递归式也可能是不等式的形式，如 $T(n) \leq 2T(n/2) + \Theta(n)$

Algorithm - P19

递归式细节
通常在计算和表示递归式时，会忽略一些递归式声明和求解的技术细节。
如对于 MERGE-SORT，当 n 为奇数时，子问题是向分解为 $T(n/2)$ 和 $T((n-1)/2)$ 。
最坏情形即 MERGE-SORT 的最坏情形运行时间的递归式为 $T(n) = T(n/2) + T((n-1)/2) + \Theta(n)$ 。
另外通常会忽略边界条件的细节，由于对于足够大的输入规模 n ，通常有 $T(n) = \Theta(n)$ 。
即表示递归式时会去掉 n 很小时的显式描述，并不影响递归式的增长阶。
由于 $T(n)$ (n 足够大时) 的值虽然改变，递归式的精确值，但改变幅度不超过一个常数因子。

最大子数组问题 (maximum subarray)
对于输入的实数数组，寻找具有最大的和的连续子数组。

即输入一个 n 个实数的数组 $A[1..n]$ ，假设其中元素不全为负数。

输出一个数组 $A[i..j]$ ，使得在所有 $A[i..j]$ ($1 \leq i \leq j \leq n$) 中， $A[i..j]$ 的和最大。
暴力求解方法为，简单地尝试每一个非空连续子数组。

即需要检查 $\binom{n}{2} + n = \Theta(n^2)$ 个子数组，即检查每一对下标组合和每一个单独元素。

最大子数组
如果数组 A [13, -3, -25, 20, -3, -16, -23, 18, 20, -7, 12, -5, -22, 15, -4, 7, X]

`maxSubarray lst = (maxSubarray :: [Int] -> (Int, Int, Int))`

`let func l left right tl tr add asum` 如果一个输入数组中不全为负数

`| null | = (left, right, asum)` 则输出最大子数组的首尾元素

`| otherwise p = max (asum, func l left right tl tr add asum)` 都是正数

`let tadd = add + head l` 且任意负数及左边的元素和为正数

`in if tadd <= 0 then func (tail l) left right (tr+1) (tr+1) 0 asum`

`else if tadd >= asum func (tail l) left right tl (tr+1) tadd asum`

`else func (tail l) tl tr tl (tr+1) tadd tadd`

`in func 0 0 1 1 0 0`

`def maxSubarray(lst):`

`left, right, temp_l, temp_r, add, sum = 0, 0, 1, 1, 0, 0`

`for i in lst:`

`tadd = add + i`

`if tadd <= 0:` 如果当前子数组小于 0，则不可能是最大子数组的一部分

`temp_l, temp_r, add = temp_r + 1, temp_r + 1, 0` 重置临时子数组的左右下标与和

`elif tadd <= sum:` 如果和大于等于当前最大和 sum，则临时子数组增长一位

`temp_r, add = temp_r + 1, tadd`

`else:` 如果 tadd > sum，则更新当前最大子数组的下标和 sum，临时子数组增长一位

`left, right, temp_r, add, sum = temp_l, temp_r, temp_r + 1, tadd, tadd`

`return (left, right, sum)`

Algorithm - P20

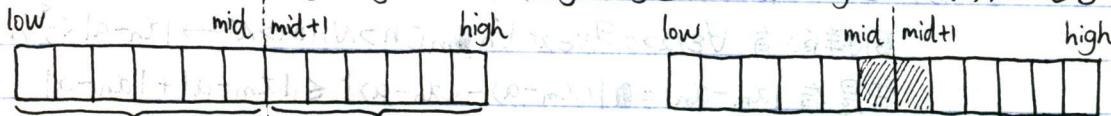
最大子数组的分治解法，假定求数组 $A[low..high]$ 中的最大子数组，且数组的中央位置 $mid = \lfloor (low+high)/2 \rfloor$

则 $A[low..high]$ 的最大子数组 $A[i..j]$ 必定属于三种情况之一

有 $low \leq i \leq j \leq mid$ ，即 $A[i..j]$ 是子数组 $A[low..mid]$ 的子数组

有 $mid < i \leq j \leq high$ ，即 $A[i..j]$ 是子数组 $A[mid+1..high]$ 的子数组

有 $low \leq i \leq mid < j \leq high$ ，即 $A[i..j]$ 跨越 $A[low..high]$ 的中点，即包含 $A[mid..mid+1]$



完全在 $A[low..mid]$ 中 完全在 $A[mid+1..high]$ 中 跨越 $A[low..high]$ 的中点，包含 $A[mid..mid+1]$

于是可以递归地求出 $A[low..mid]$ 和 $A[mid+1..high]$ 的最大子数组

再求出 $A[low..high]$ 中包含 $A[mid..mid+1]$ 的最大子数组

比较 3 个子数组的和，可以得到 $A[low..high]$ 的最大子数组

注意第三种情形并非原问题的规模更小的实例，因为加入了包含 $A[mid..mid+1]$ 的限制条件

对于求 $A[low..high]$ 中包含 $A[mid..mid+1]$ 的最大子数组，可以在 $\Theta(n)$ 时间内完成

FIND-MAX-CROSSING-SUBARRAY(A, low, mid, high)

$$left-sum = -\infty$$

$$sum = 0$$

for $i = mid$ down to low

$$sum = sum + A[i]$$

if $sum > left-sum$

$$left-sum = sum$$

$$max-left = i$$

$$right-sum = -\infty$$

$$sum = 0$$

for $j = mid+1$ to $high$

$$sum = sum + A[j]$$

if $sum > right-sum$

$$right-sum = sum$$

$$max-right = j$$

return ($max-left$, $max-right$, $left-sum + right-sum$)

def find_max_crossing_subarray(clst, low, mid, high):

$$left-sum, sum, i = float(-inf), 0, mid$$

while $i \geq low$:

$$sum += clst[i]$$

if $sum > left-sum$:

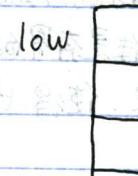
$$left-sum, max-left = sum, i$$

$$i -= 1$$

从 $A[mid]$ 开始，依序遍历至 $A[low..mid]$

并从中找出包含 $A[mid]$ 的最大连续子数组

则循环了 $mid - low + 1$ 次



从 $A[mid+1]$ 开始，依序遍历至 $A[mid+1..high]$

并从中找出包含 $A[mid+1]$ 的最大连续子数组

则循环了 $high - (mid + 1) + 1$ 次

可知 $A[low..high]$ 中包含 $A[mid..mid+1]$ 的最大连续子数组必是两个数组的合并

总计循环次数为 $high - low + 1$ 次

所以算法是 $\Theta(n)$

$$right-sum, sum, j = float(-inf), 0, mid+1$$

while $j \leq high$:

$$sum += clst[j]$$

if $sum > right-sum$:

$$right-sum, max-right = sum, j$$

$$j += 1$$

return ($max-left$, $max-right$, $left-sum + right-sum$)

Algorithm - P21

贪心法

最大子数数组的 分解：将原问题分解为，求完全位于前半部分或后半部分的最大子数数组

分治解法

以及 跨越数组中点的最大子数数组

求解：递归地求解完全位于前半部分或后半部分的最大子数数组

合并：首先求解跨越数组中点的最大子数数组。

然后通过比较三种情形的最大子数数组的和，求出原问题的解

FIND-MAXIMUM-SUBARRAY(A, low, high)

if low == high

return (low, high, A[low])

else mid = (low + high) / 2

(left_low, left_high, left_sum) =

FIND-MAXIMUM-SUBARRAY(A, low, mid)

(right_low, right_high, right_sum) =

FIND-MAXIMUM-SUBARRAY(A, mid+1, high)

(cross_low, cross_high, cross_sum) =

FIND-MAX-CROSSING-SUBARRAY(A, low, mid, high)

if left_sum >= right_sum and left_sum >= cross_sum

return (left_low, left_high, left_sum)

elseif right_sum >= left_sum and right_sum >= cross_sum

return (right_low, right_high, right_sum)

else return (cross_low, cross_high, cross_sum)

于是综合可得算法的运行时间 $T(n) = \begin{cases} \Theta(n), & n=1 \\ \Theta(n \lg n), & n>1 \end{cases}$

则时间复杂度为 $\Theta(n \lg n)$

def find_maximum_subarray(lst, low, high):

if low == high : return (low, high, lst[low])

else: mid = (low + high) // 2

left_low, left_high, left_sum = find_maximum_subarray(lst, low, mid)

right_low, right_high, right_sum = find_maximum_subarray(lst, mid+1, high)

(cross_low, cross_high, cross_sum) = find_max_crossing_subarray(lst, low, mid, high)

if left_sum >= right_sum and left_sum >= cross_sum :

return (left_low, left_high, left_sum)

elif right_sum >= cross_sum : 注意当 left_sum >= right_sum

return (right_low, right_high, right_sum)

else: (mid_low, mid_high) 中寻找一个量 right_sum > cross_sum

return (cross_low, cross_high, cross_sum)

\rightarrow right_sum > left_sum

Algorithm - P22

矩阵乘法 对于 $n \times n$ 的方阵 $A = (a_{ij})$ 和 $B = (b_{ij})$, 其中 $i, j = 1, 2, \dots, n$. 定义 $A \cdot B$ 为 AB 的乘积 $C = (c_{ij})$, $C = AB$

有对任意 $1 \leq i, j \leq n$, $c_{ij} = \sum_{k=1}^n a_{ik} \cdot b_{kj}$

标准算法 SQUARE-MATRIX-MULTIPLY($n \times n$ 方阵 A, B , A, B 具有相同大小)

$n = A.\text{rows}$

let $C = \text{new } n \times n \text{ matrix}$

for $i = 1$ to n

 for $j = 1$ to n

$c_{ij} = 0$

 for $k = 1$ to n

$c_{ij} += a_{ik} \cdot b_{kj}$

 return C

调用矩阵的属性 $\text{rows} \rightarrow n$

并新建一个 $n \times n$ 矩阵

对于矩阵 C 的每个元素 c_{ij}

有 $c_{ij} = \sum_{k=1}^n a_{ik} \cdot b_{kj}$

三层循环每一层都有 n 步

循环体执行需要常量时间 $\Theta(1)$

标准算法的时间复杂度为 $\Theta(n^3)$

简单分治算法: 假定 A, B 为 $n \times n$ 矩阵, 其中 n 为 2 的幂, 即 $n = 2^k$, $k \in \mathbb{Z}^+$

且 A 和 B 都可以划分为 $\frac{n}{2} \times \frac{n}{2}$ 的矩阵, $C = AB$ 也可以划分为 $\frac{n}{2} \times \frac{n}{2}$ 的矩阵

即 $C = AB$ 可以转换为 $\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \cdot \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$

即可以定义递归算法, 传入两个 $n \times n$ 的矩阵 A 和 B , 返回 $n \times n$ 的矩阵 $C = AB$

SQUARE-MATRIX-MULTIPLY-RECURSIVE($n \times n$ 方阵 A, B , n 为 2 的幂)

$n = A.\text{rows}$

let $C = \text{new } n \times n \text{ matrix}$

if $n == 1$ then $C_{11} = a_{11} \cdot b_{11}$

else partition A, B, C as $\frac{n}{2} \times \frac{n}{2}$

分解步骤聚

时间复杂度 $\Theta(1)$

则复杂度为 $\Theta(1)$

如果以下标表示,

则复杂度为 $\Theta(n^2)$

$C_{11} = \text{SQUARE-MATRIX-MULTIPLY-RECURSIVE}(A_{11}, B_{11})$

+ $\text{SQUARE-MATRIX-MULTIPLY-RECURSIVE}(A_{12}, B_{21})$

$C_{12} = \text{SQUARE-MATRIX-MULTIPLY-RECURSIVE}(A_{11}, B_{12})$

+ $\text{SQUARE-MATRIX-MULTIPLY-RECURSIVE}(A_{12}, B_{22})$

$C_{21} = \text{SQUARE-MATRIX-MULTIPLY-RECURSIVE}(A_{21}, B_{11})$

+ $\text{SQUARE-MATRIX-MULTIPLY-RECURSIVE}(A_{22}, B_{21})$

$C_{22} = \text{SQUARE-MATRIX-MULTIPLY-RECURSIVE}(A_{21}, B_{12})$

+ $\text{SQUARE-MATRIX-MULTIPLY-RECURSIVE}(A_{22}, B_{22})$

return $C = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$

基础情形

分解为 $T(n/2)$ 的子问题

根据算法不同, 如果复制为新矩阵

解决步骤聚

共 8 次递归地调用

每次 $T(n/2)$ 的复杂度

矩阵加法

时间复杂度为 $\Theta(\frac{n^2}{4})$

递归式 $T(n) =$

$\begin{cases} \Theta(1), & n=1 \\ 8T(n/2) + \Theta(n^3), & n>1 \end{cases}$

于是有时间复杂度

$\Theta(8^{lg n}) = \Theta(n^3)$

Algorithm - P23

Strassen 算法，相对于矩阵乘法的标准算法和分治算法，Strassen 算法尽量用矩阵加减法代替矩阵乘法。

- 主要步骤聚：1. 将输入矩阵 A, B 和输出矩阵 C 分解为 $\frac{n}{2} \times \frac{n}{2}$ 的 3 矩阵
- 2. 创建 10 个 $\frac{n}{2} \times \frac{n}{2}$ 矩阵 S_1, S_2, \dots, S_{10} ，皆由上一步中划分矩阵的和与差得到
- 3. 创建 7 个 $\frac{n}{2} \times \frac{n}{2}$ 矩阵 P_1, P_2, \dots, P_7 ，皆由上两步中矩阵的乘积得到
- 4. 通过对矩阵 P_1, \dots, P_7 的不同组合的加减运算，得到子矩阵 $C_{11}, C_{12}, C_{21}, C_{22}$

	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}
A_{11}		+				+				+
A_{12}		+					+			
A_{21}			+							-
A_{22}			+		+		-			
B_{11}				-			+			+
B_{12}	+									+
B_{21}				+					+	
B_{22}	-					+		+		
$B_{12} - B_{22}$	$A_{11} + A_{12}$	$A_{21} + A_{22}$	$B_{21} - B_{11}$	$A_{11} + A_{22}$	$B_{11} + B_{22}$	$B_{11} - B_{22}$	$A_{12} - A_{21}$	$B_{21} + B_{22}$	$A_{11} - A_{21}$	$B_{11} + B_{12}$

$$P_1 = A_{11} \cdot S_1 = A_{11}B_{12} - A_{11}B_{22}, \quad P_2 = S_2 \cdot B_{22} = A_{11}B_{22} + A_{12}B_{22}$$

$$P_3 = S_3 \cdot B_{11} = A_{21}B_{11} + A_{22}B_{11}, \quad P_4 = A_{22} \cdot S_4 = A_{22}B_{21} - A_{22}B_{11}$$

$$P_5 = S_5 \cdot S_6 = A_{11}B_{11} + A_{11}B_{22} + A_{22}B_{11} + A_{22}B_{22}$$

$$P_6 = S_7 \cdot S_8 = A_{12}B_{21} + A_{12}B_{22} - A_{22}B_{21} - A_{22}B_{22}, \quad C_{11} = P_5 + P_4 - P_2 + P_6, \quad C_{12} = P_1 + P_2$$

$$P_7 = S_9 \cdot S_{10} = A_{11}B_{11} + A_{11}B_{12} - A_{21}B_{11} - A_{21}B_{12}, \quad C_{21} = P_3 + P_4, \quad C_{22} = P_5 + P_1 - P_3 - P_7$$

procedure STRASSEN($n \times n$ 矩阵 A, B)
 于是有递归式 $T(n) = \begin{cases} \Theta(1), & n=1 \\ 7T(n/2) + \Theta(n^2), & n>1 \end{cases}$

$n = A.length$, let $C = new nxn$ 矩阵

if $n == 1$ then $C_{11} = a_{11} * b_{11}$

else 划分 A, B, C 为 $\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}, \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}, \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$

$S_1 = B_{12} - B_{22}$, $S_2 = A_{11} + A_{12}$

$S_3 = A_{21} + A_{22}$, $S_4 = B_{21} - B_{11}$

$S_5 = A_{11} + A_{22}$, $S_6 = B_{11} + B_{22}$, $S_7 = A_{12} - A_{21}$

$S_8 = B_{21} + B_{22}$, $S_9 = A_{11} - A_{21}$

$S_{10} = B_{11} + B_{12}$

基础情形，时间复杂度为 $\Theta(n)$

可知 $T(n) = \Theta(7^{lg n})$

$= \Theta(n^{lg 7})$

分解步骤聚，时间复杂度为 $\Theta(n)$

$\times 2 < \lg 7 < 3$

10 次 $\frac{n}{2} \times \frac{n}{2}$ 矩阵加减法，所以 Strassen 算法

时间复杂度为 $\Theta(n^2)$ 是优于分治算法的

$P_1 = \text{STRASSEN}(A_{11}, S_1)$, $P_2 = \text{STRASSEN}(S_2, B_{22})$ 7 次递归调用 STRASSEN 算法

$P_3 = \text{STRASSEN}(S_3, B_{11})$, $P_4 = \text{STRASSEN}(A_{22}, S_4)$ 时间复杂度为 $7T(n/2)$

$P_5 = \text{STRASSEN}(S_5, S_6)$, $P_6 = \text{STRASSEN}(S_7, S_8)$, $P_7 = \text{STRASSEN}(S_9, S_{10})$

$C_{11} = P_5 + P_4 - P_2 + P_6$, $C_{12} = P_1 + P_2$, $C_{21} = P_3 + P_4$, $C_{22} = P_5 + P_1 - P_3 - P_7$ 合并步骤聚，8 次 $\frac{n}{2} \times \frac{n}{2}$ 矩阵加减法

return $C = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$ 可见复杂度 $\Theta(n^2)$