算法设计与分析

蒋婷婷

上节课回顾

- 算法研究的重要性
- 理论上和现实可计算性
- 计算复杂性理论
- 算法复杂度

函数渐近的界

设 f 和 g 是定义域为自然数集 N上的函数

(1) $f(n)=O(g(n))$

若存在正数c和n_0使得对一切$n\geq n_0$有$0\leq f(n)\leq cg(n)$

(2) $f(n)=\Omega(g(n))$

若存在正数c和n_0使得对一切$n\geq n_0$有$0\leq cg(n)\leq f(n)$

(3) $f(n)=o(g(n))$

对所有正数$c>0$存在n_0使得对一切$n\geq n_0$有$0\leq f(n)<cg(n)$

(4) $f(n)=\omega(g(n))$.

对所有正数$c>0$存在n_0使得对一切$n\geq n_0$有$0\leq cg(n)<f(n)$

(5) $f(n)=\Theta(g(n)) \Leftrightarrow f(n)=O(g(n))$ 且 $f(n)=\Omega(g(n))$

(6) $O(1)$表示常数函数

函数渐近的界的基本性质

定理1.1 设 f 和 g 是定义域为自然数集 N上的函数.

(1) 如果 $\lim\limits_{n\to\infty}\dfrac{f(n)}{g(n)}$ 存在，并且等于某个常数$c>0$，那么

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

(2) 如果$\lim\limits_{n\to\infty}\dfrac{f(n)}{g(n)}=0$，那么

$f(n)=o(g(n))$.

(3) 如果 $\lim\limits_{n\to\infty}\dfrac{f(n)}{g(n)}=+\infty$, 那么

$f(n)=\omega(g(n))$.

证明定理1.1(1)

(1) 根据极限定义，对于给定的正数$\varepsilon=c/2$, 存在某个n_0，只要$n\geq n_0$，就有

$$|\frac{f(n)}{g(n)}-c|<\varepsilon \Rightarrow c-\varepsilon<\frac{f(n)}{g(n)}<c+\varepsilon$$

$$\Rightarrow \frac{c}{2}<\frac{f(n)}{g(n)}<\frac{3c}{2}<2c$$

对所有的$n\geq n_0$，$f(n)\leq 2cg(n)$. 从而推出 $f(n)=O(g(n))$

对所有的$n\geq n_0$，$f(n)\geq (c/2)g(n)$，从而推出 $f(n)=\Omega(g(n))$, 于是 $f(n)=\Theta(g(n))$

函数渐近的界的基本性质

定理1.2 设 f, g, h是定义域为自然数集合的函数，

(1) 如果 $f=O(g)$且 $g=O(h)$，那么 $f=O(h)$.

(2) 如果 $f=\Omega(g)$且 $g=\Omega(h)$，那么 $f=\Omega(h)$.

(3) 如果 $f=\Theta(g)$和 $g=\Theta(h)$，那么 $f=\Theta(h)$.

定理1.3 假设 f 和g是定义域为自然数集合的函数，若对某个其它的函数 h, 我们有$f=O(h)$和$g=O(h)$，那么

$$f+g=O(h).$$

推论 假设 f 和g是定义域为自然数集合的函数，且满足 $g=O(f)$，那么 $f+g=\Theta(f)$.

基本函数类

阶的高低

至少指数级：$2^n, 3^n, n!, \ldots$

多项式级：$n, n^2, n\log n, n^{1/2}, \ldots$

对数多项式级：$\log n, \log^2 n, \ldots$

$$2^{2^n}, \quad n!, \quad n2^n, \quad (3/2)^n, \quad (\log n)^{\log n} = \Theta(n^{\log\log n}),$$

$$n^3, \quad \log(n!) = \Theta(n\log n), \quad n = 2^{\log n},$$

$$\log^2 n, \quad \log n, \quad \sqrt{\log n}, \quad \log\log n,$$

$$n^{1/\log n} = \Theta(1)$$

例题

例1 设 $f(n)=\frac{1}{2}n^2-3n$，证明 $f(n)=\Theta(n^2)$.

$$\lim_{n\to+\infty}\frac{f(n)}{n^2}=\lim_{n\to+\infty}\frac{\frac{1}{2}n^2-3n}{n^2}=\frac{1}{2}$$

根据定理1.1有 $f(n)=\Theta(n^2)$.

例题

例2　PrimalityTest(n)

输入：n, n为大于 2 的奇整数
输出：true 或者false

1．$s \leftarrow \sqrt{n}$
2．for $j \leftarrow$2 to s
3．　if j 整除 n
4．　then return false
5．return true

假设计算 $\sqrt{n}$可以在$O(1)$时间完成，可以写$O(\sqrt{n})$,
不能写$\Theta(\sqrt{n})$

多项式时间的算法

- 多项式时间的算法

  时间复杂度函数为$O(p(n))$的算法，其中$p(n)$是n的多项式

- 不是多项式时间的算法

  不存在多项式 $p(n)$使得该算法的时间复杂度为$O(p(n))$

  包含指数时间甚至更高阶的算法

多项式函数与指数函数

时间复杂度函数	问题规模					
	10	20	30	40	50	60
n	10^{-5}	$2*10^{-5}$	$3*10^{-5}$	$4*10^{-5}$	$5*10^{-5}$	$6*10^{-5}$
n^2	10^{-4}	$4*10^{-4}$	$9*10^{-4}$	$16*10^{-4}$	$25*10^{-4}$	$36*10^{-4}$
n^3	10^{-3}	$8*10^{-3}$	$27*10^{-3}$	$64*10^{-3}$	$125*10^{-3}$	$216*10^{-3}$
n^5	10^{-1}	3.2	24.3	1.7 分	5.2 分	13.0 分
2^n	.001 秒	1.0 秒	17.9 分	12.7 天	35.7 年	366 世纪
3^n	.059 秒	58 分	6.5 年	3855 世纪	$2*10^8$ 世纪	$1.3*10^{13}$ 世纪

表中默认单位为秒

多项式函数与指数函数

时间复杂度函数	1小时可解的问题实例的最大规模		
	计算机	快100倍的计算机	快1000倍的计算机
n	N_1	$100\ N_1$	$1000\ N_1$
n^2	N_2	$10\ N_2$	$31.6\ N_2$
n^3	N_3	$4.64\ N_3$	$10\ N_3$
n^5	N_4	$2.5\ N_4$	$3.98\ N_4$
2^n	N_5	$N_5 + 6.64$	$N_5 + 9.97$
3^n	N_6	$N_6 + 4.19$	$N_6 + 6.29$

10亿次/秒机器求解的问题

- 快速排序算法给10万个数据排序, 运算量约为 $10^5 \times \log_2 10^5 \approx 1.7 \times 10^6$, 仅需$1.7 \times 10^6 / 10^9 = 1.7 \times 10^{-3}$秒.
- Dijkstra算法求解1万个顶点的图的单源最短路径问题, 运算量约为$(10^4)^2 = 10^8$, 约需$10^8 / 10^9 = 0.1$秒.
- 回溯法解100个顶点的图的最大团问题, 运算量为 $100 \times 2^{100} \approx 1.8 \times 10^{32}$, 需要$1.8 \times 10^{32} / 10^9 = 1.8 \times 10^{21}$秒 $= 5.7 \times 10^{15}$年, 即5千7百万亿年!
- 1 分钟能解多大的问题. 1分钟60秒, 这台计算机可做用快速排序算法可给2×10^9(即, 20亿)个数据排序, 用Dijkstra算法可解2.4×10^5个顶点的图的单源最短路径问题. 而用回溯法一天只能解41个顶点的图的最大团问题

问题的复杂度分析

多项式时间可解的问题与难解的问题

- **多项式时间可解的问题 *P***

  存在着解***P*** 的多项式时间的算法

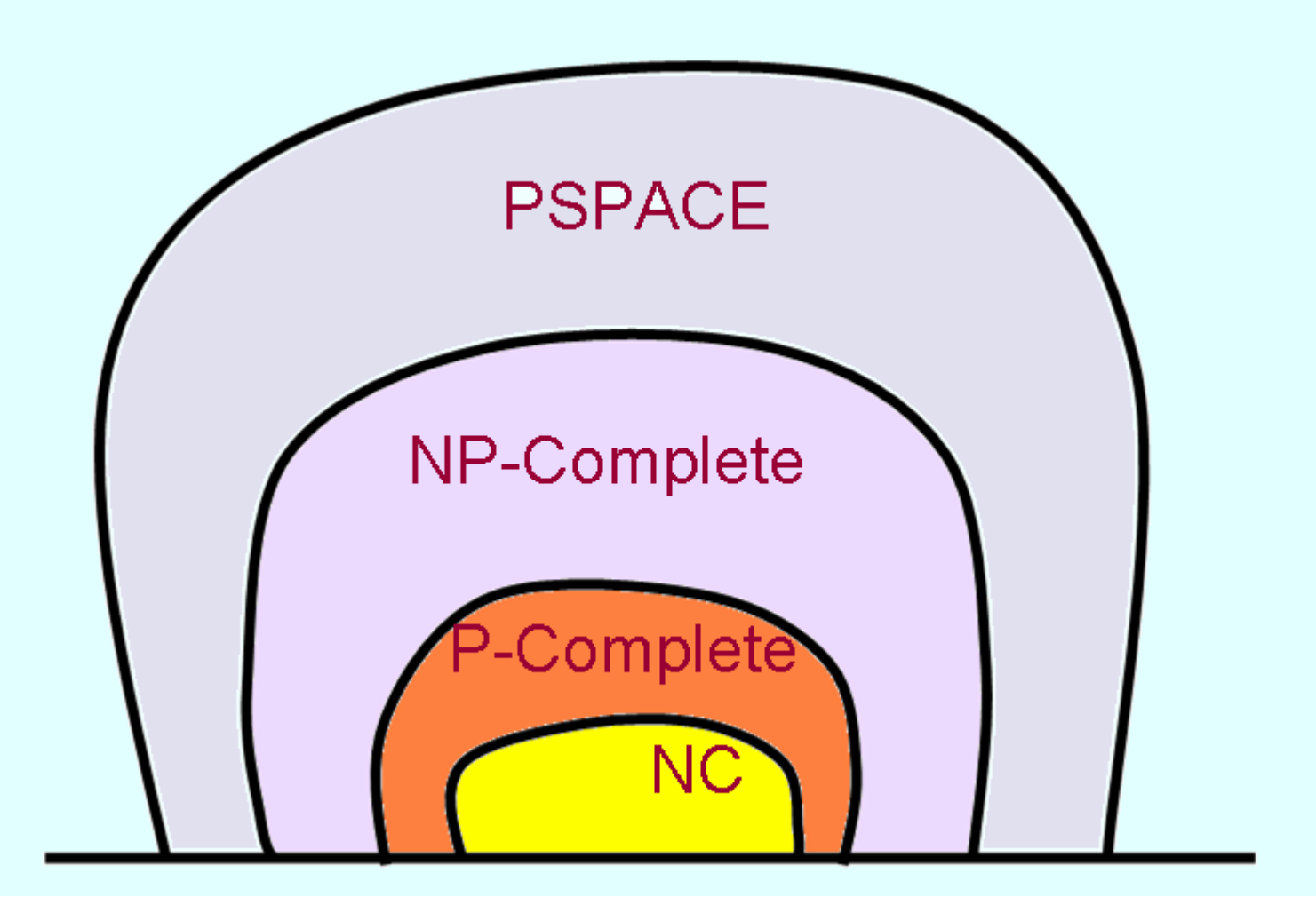
- **难解的问题*P***

  不存在解***P*** 的多项式时间的算法

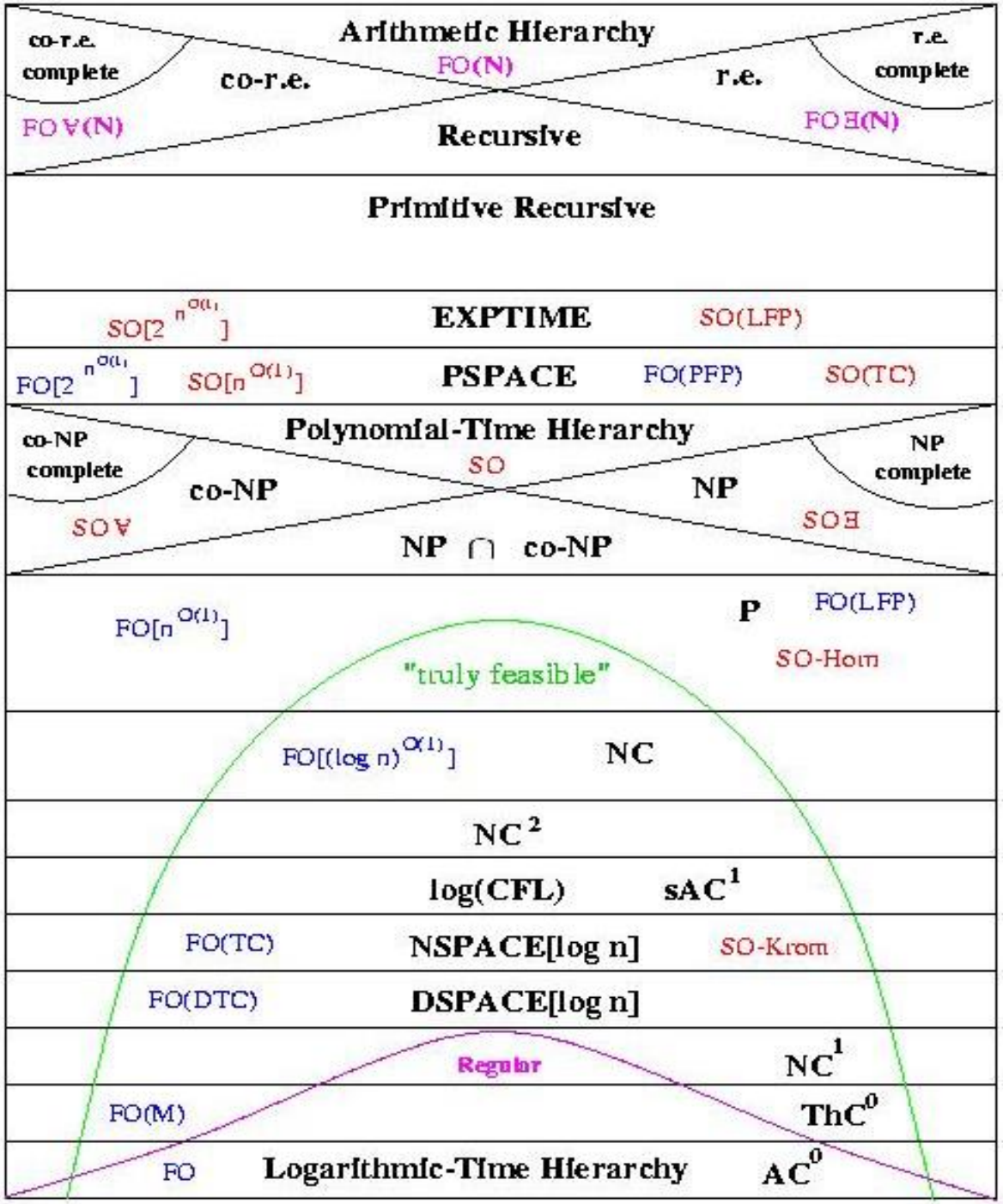
- 实际上可计算的问题

  多项式时间可解的问题

不同复杂性类的基本层次结构



计算复杂性类的谱系


Arithmetic Hierarchy
co-r.e. complete
co-r.e.
FO(N)
r.e.
r.e. complete
FO∀(N)
FO∃(N)
Recursive
Primitive Recursive
SO[2^n^O(1)]
EXPTIME
SO(LFP)
FO[2^n^O(1)]
SO[n^O(1)]
PSPACE
FO(PFP)
SO(TC)
Polynomial-Time Hierarchy
co-NP complete
co-NP
SO
NP
NP complete
SO∀
SO∃
NP ∩ co-NP
FO[n^O(1)]
P
FO(LFP)
"truly feasible"
SO-Horn
FO[(log n)^O(1)]
NC
NC^2
log(CFL)
sAC^1
FO(TC)
NSPACE[log n]
SO-Krom
FO(DTC)
DSPACE[log n]
Regular
NC^1
FO(M)
ThC^0
FO
Logarithmic-Time Hierarchy
AC^0

算法及计算复杂性理论的拓广

- 算法
  - 概率算法
  - 近似算法
  - 在线算法
  - 分布式算法
- 计算复杂性
  - 概率Turing机与概率复杂性
  - 近似求解的复杂性
  - 参数复杂性
  - 计数复杂性
  - 通信复杂性

计算理论的发展

- 新的理论
  - 新的计算模型

    算法和计算复杂度理论的基础是Turing机模型，现在有许多非Turing机的计算模型，如DNA计算，量子计算等
  - 新的环境

    正在提出的云计算等新的计算环境，不再是传统的顺序算法、并行算法或者分布式算法等，更加重视协同与博弈，可能需要研究关于刻画独立个体群体行为的新的计算理论、模型和分析方法
- 在软件开发中算法好坏有时不是最重要的因素，更重要的可能是标准化、质量保证、实现成本等

数学基础：对数函数

符号：

$$\log n = \log_2 n, \quad (\lg n = \log_{10} n)$$

$$\log^k n = (\log n)^k$$

$$\log\log n = \log(\log n)$$

性质：

$$\log_b n = o(n^\alpha) \quad \alpha > 0$$

$$a^{\log_b n} = n^{\log_b a}$$

$$\log_k n = \Theta(\log_l n)$$

阶乘

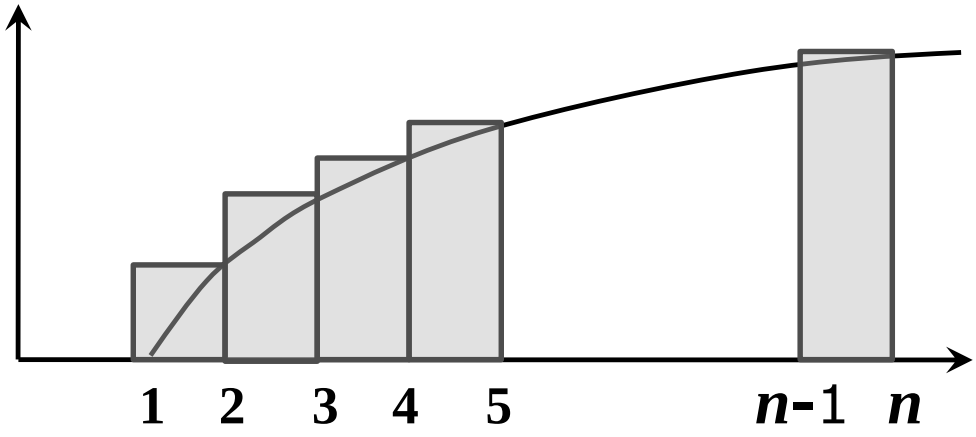
Stirling公式 $n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n\left(1+\Theta\left(\frac{1}{n}\right)\right)$

$n! = o(n^n)$， $n! = \Omega(2^n)$

$\log(n!) = \Theta(n\log n)$

$$\log(n!) = \sum_{k=1}^{n} \log k$$

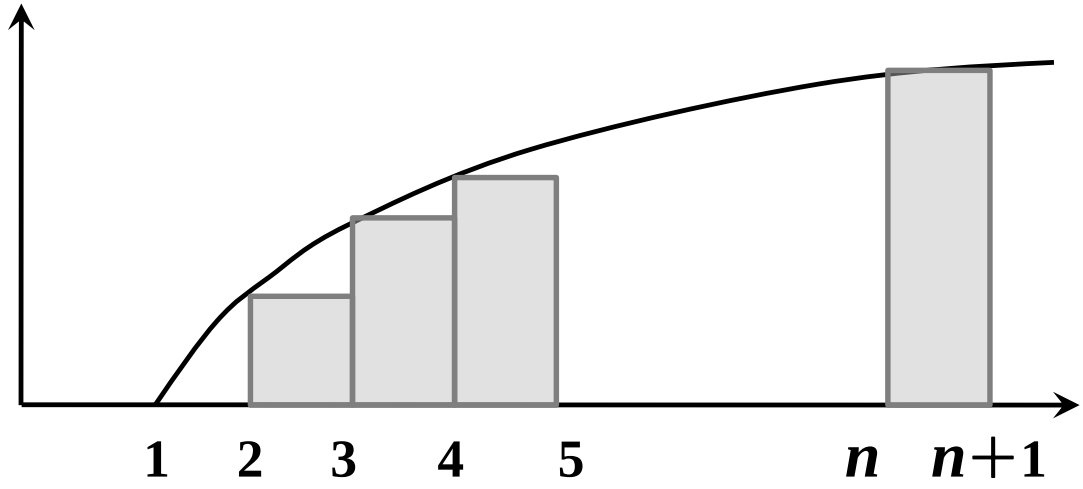
$$\geq \int_{1}^{n} \log\, x dx$$



$$= \log e(n\ln n - n + 1)$$

$$= \Omega(n\log n)$$

阶乘（续）



$$\log(n!) = \sum_{k=1}^{n} \log k \le \int_{2}^{n+1} \log x dx = O(n \log n)$$

取整函数

$\lfloor x \rfloor$：表示小于等于x的最大的整数

$\lceil x \rceil$：表示大于等于x的最小的整数

取整函数具有下述性质：

(1) $x-1<\lfloor x \rfloor \le x \le \lceil x \rceil < x+1$

(2) $\lfloor x+n \rfloor = \lfloor x \rfloor + n$, $\lceil x+n \rceil = \lceil x \rceil + n$，其中$n$为整数

(3) $$\left\lceil \frac{n}{2} \right\rceil + \left\lfloor \frac{n}{2} \right\rfloor = n$$

(4) $$\left\lceil \frac{\lceil n/a \rceil}{b} \right\rceil = \left\lceil \frac{n}{ab} \right\rceil, \quad \left\lfloor \frac{\lfloor n/a \rfloor}{b} \right\rfloor = \left\lfloor \frac{n}{ab} \right\rfloor$$

求和公式

基本求和公式

$$\sum_{k=1}^{n} a_k = \frac{n(a_1 + a_n)}{2}, \quad \{a_k\}\text{为等差数列}$$

$$\sum_{k=0}^{n} aq^k = \frac{a(1-q^{n+1})}{1-q}, \quad \sum_{k=0}^{n} x^k = \frac{1-x^{n+1}}{1-x},$$

$$\sum_{k=0}^{\infty} aq^k = \frac{a}{1-q} \quad (q<1)$$

$$\sum_{k=1}^{n} \frac{1}{k} = \ln n + O(1)$$

估计和式上界的方法

放大法：

1. $\sum_{k=1}^{n} a_k \le n a_{\max}$

2. 假设存在常数 $r<1$，使得 对一切 $k \ge 0$成立 $\frac{a_{k+1}}{a_k} \le r$， 则

$$\sum_{k=0}^{n} a_k \le \sum_{k=0}^{\infty} a_0 r^k = a_0 \sum_{k=0}^{\infty} r^k = \frac{a_0}{1-r}$$

求和实例

例 求和

(1) $\sum_{k=1}^{n-1}\frac{1}{k(k+1)}$

(2) $\sum_{t=1}^{k}t2^{t-1}$

解

(1) $$\sum_{k=1}^{n-1}\frac{1}{k(k+1)}=\sum_{k=1}^{n-1}(\frac{1}{k}-\frac{1}{k+1})$$

$$=\sum_{k=1}^{n-1}\frac{1}{k}-\sum_{k=1}^{n-1}\frac{1}{k+1}=\sum_{k=1}^{n-1}\frac{1}{k}-\sum_{k=2}^{n}\frac{1}{k}=1-\frac{1}{n}$$

求和实例

$$(2)\quad \sum_{t=1}^{k} t2^{t-1} = \sum_{t=1}^{k} t(2^t - 2^{t-1})$$

$$= \sum_{t=1}^{k} t2^t - \sum_{t=1}^{k} t2^{t-1}$$

$$= \sum_{t=1}^{k} t2^t - \sum_{t=0}^{k-1} (t+1)2^t$$

$$= \sum_{t=1}^{k} t2^t - \sum_{t=0}^{k-1} t2^t - \sum_{t=0}^{k-1} 2^t$$

$$= k2^k - (2^k - 1)$$

$$= (k-1)2^k + 1$$

实例

例 估计 $\sum_{k=1}^{n}\frac{k}{3^k}$ 的上界.

解 由 $a_k=\frac{k}{3^k},\quad a_{k+1}=\frac{k+1}{3^{k+1}}$

得 $$\frac{a_{k+1}}{a_k}=\frac{1}{3}\frac{k+1}{k}\le\frac{2}{3}$$

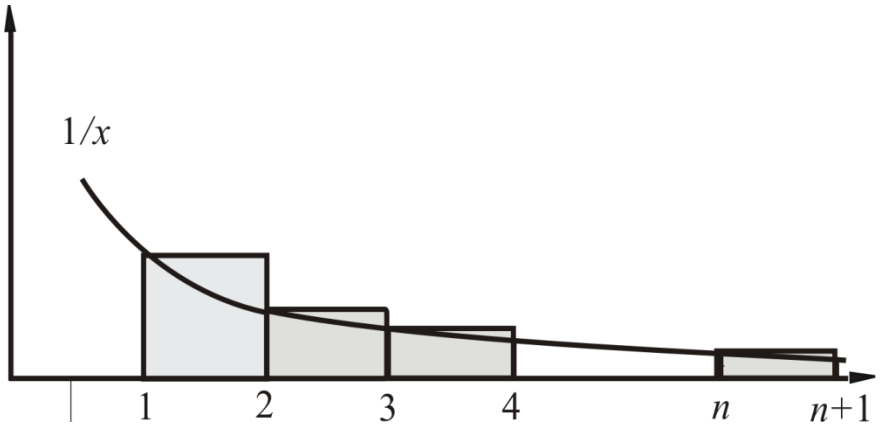
$$\sum_{k=1}^{n}\frac{k}{3^k}\le\sum_{k=1}^{\infty}\frac{1}{3}(\frac{2}{3})^{k-1}=\frac{1}{3}\frac{1}{1-\frac{2}{3}}=1$$

估计和式渐近的界

估计 $\sum_{k=1}^{n}\frac{1}{k}$ 的渐近的界.

$$\sum_{k=1}^{n}\frac{1}{k}\geq\int_{1}^{n+1}\frac{dx}{x}$$

$$=\ln(n+1)$$



$$\sum_{k=1}^{n}\frac{1}{k}=\frac{1}{1}+\sum_{k=2}^{n}\frac{1}{k}$$

$$\leq 1+\int_{1}^{n}\frac{dx}{x}$$

$$=\ln n+1$$