第1章 算法概述

内容

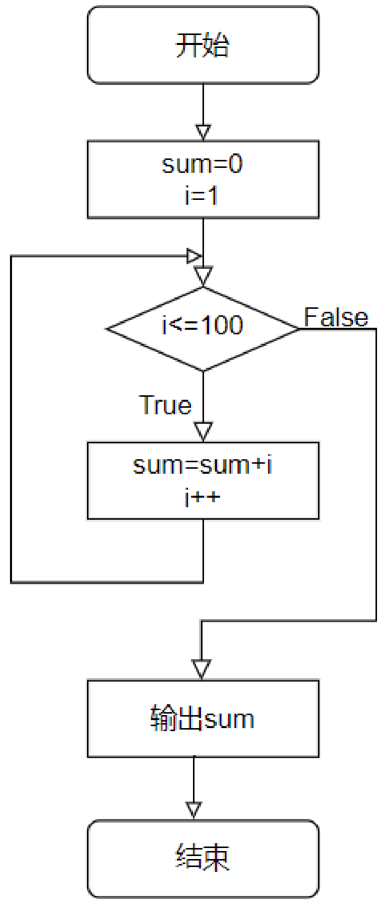
- 1.1 算法与程序
- 1.2 算法复杂性分析

算法(Algorithm)

- 算法是指解决问题的一种方法或一个过程。
- 算法是若干指令的有穷序列，满足性质：
- (1)**输入**：有外部提供的量作为算法的输入。
- (2)**输出**：算法产生至少一个量作为输出。
- (3)**确定性**：组成算法的每条指令是清晰，无歧义的。
- (4)**有限性**：算法中每条指令的执行次数是有限的，执行每条指令的时间也是有限的。

算法表示方式

- 自然语言
- **流程图**
- N-S流程图
- **伪代码**
- 计算机语言

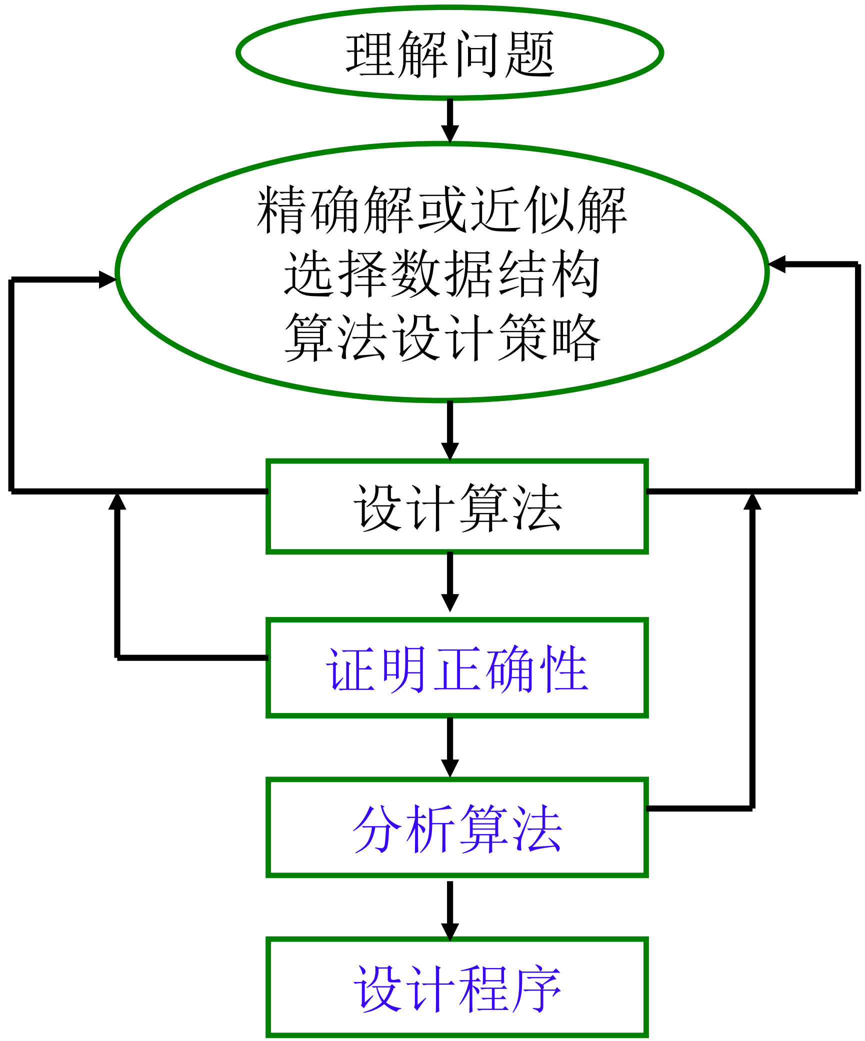



```
sum=0
for i=1 to 100
     sum=sum+i
print sum
```

程序(Program)

- 程序是算法用某种程序设计语言的具体实现。
- 程序可以不满足算法的性质(4)。
  - 例如操作系统，是一个在无限循环中执行的程序，因而不是一个算法。
  - 操作系统的各种任务可看成是单独的问题，每一个问题由操作系统中的一个子程序通过特定的算法来实现。该子程序得到输出结果后便终止。

问题求解(Problem Solving)



内容

- 1.1 算法与程序
- 1.2 算法复杂性分析

1.2 算法复杂性分析

- 算法复杂性 = 算法所需要的计算机资源
  - 算法复杂性信赖于要解的问题的规模（N）、算法的输入（I）和算法本身（A）的函数
  - $C = F(N, I, A)$
- 算法的时间复杂性 $T(N, I, A)$ ➡ $T(N, I)$
- 算法的空间复杂性 $S(N, I, A)$ ➡ $S(N, I)$

时间复杂性

- 算法在一台抽象的计算机上运行所需要的时间。

$$T(N,I)=\sum_{i=1}^{k} t_i e_i(N,I)$$

  - 其中，元运算有k 种，记为$O_1, O_2, \ldots, O_k$
  - 每执行一次这些元运算所需要时间分别为$t_1, t_2, \ldots, t_k$
  - 用到元运算O_i 的次数为 $e_i(N,I)$

三种情况下的时间复杂性

- 只能在规模为N的某些或某类有代表性的合法输入中统计相应的e_i
- 最坏情况下的时间复杂性

$$T_{\max}(N)=\max_{I\in D_N}T(N,I)=\max_{I\in D_N}\sum_{i=1}^{k}t_ie_i(N,I)=\sum_{i=1}^{k}t_ie_i(N,I^*)=T(N,I^*)$$

- 最好情况下的时间复杂性

$$T_{\min}(N)=\min_{I\in D_N}T(N,I)=\min_{I\in D_N}\sum_{i=1}^{k}t_ie_i(N,I)=\sum_{i=1}^{k}t_ie_i(N,\widetilde{I})=T(N,\widetilde{I})$$

- 平均情况下的时间复杂性

$$T_{\text{avg}}(N)=\sum_{I\in D_N}P(I)T(N,I)=\sum_{I\in D_N}P(I)\sum_{i=1}^{k}t_ie_i(N,I)$$

其中D_N是规模为N的合法输入的集合；
$P(I)$是在算法的应用中出现输入I的概率。

算法的渐近复杂性

- 当$N \to \infty$时，$T(N) \to \infty$
- 如果存在$\tilde{T}(N)$，使得当$N \to \infty$时有$\frac{T(N)-\tilde{T}(N)}{T(N)} \to 0$，那么，$\tilde{T}(N)$是算法 A 当 $N \to \infty$ 的渐近复杂性。
- 在数学上，$\tilde{T}(N)$是$T(N)$当$N \to \infty$时的渐近表达式，是$T(N)$略去低阶项留下的主项，比$T(N)$简单
- 例：

$$T(N) = 3N^2 + 4N\log N + 7$$

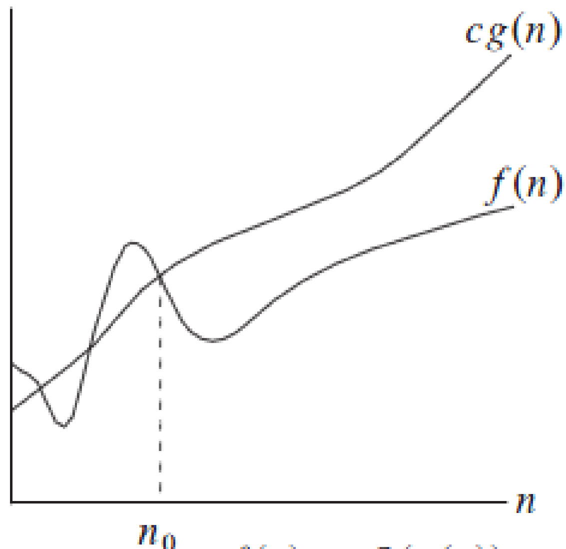
$$\tilde{T}(N) = 3N^2 \Longleftarrow \frac{T(N)-\tilde{T}(N)}{T(N)} = \frac{4N\log N + 7}{3N^2 + 4N\log N + 7} \to 0$$

渐近分析的记号

- 在以下的讨论中，$f(n)$和 $g(n)$是定义在正数集上的正函数。

渐近上界记号O

- 如果存在正的常数c和自然数 n_o，使得当$n \geqslant n_o$时有$f(n) \leqslant cg(n)$，则称函数 $f(n)$ 当n充分大时上有界，且 $g(n)$ 是它的一个上界，记为 $f(n) = \mathrm{O}(\mathrm{g(n)})$



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

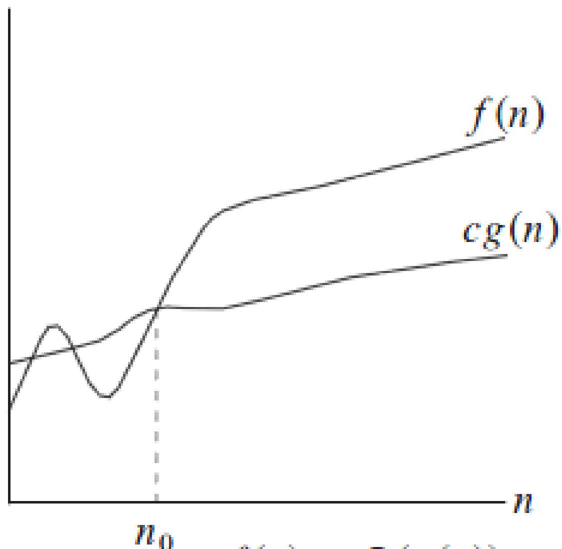
例子

- $3n = O(n)$
- $n+1024 = O(n)$
- $2n^2+11n-10 = O(n^2)$
- $n^2 = O(n^3)$

对于一个算法，得到的上界的阶越低，则评估就越精确，结果就越有价值。

渐近下界记号 Ω

- 如果存在正的常数 c 和自然数 n_o，使得当 $n \geqslant n_o$ 时有 $f(n) \geqslant cg(n)$，则称函数 $f(n)$ 当 n 充分大时下有界，且 $g(n)$ 是它的一个下界，记为 $f(n) = \Omega(g(n))$



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

渐近下界记号Ω

- 对于一个算法，得到的下界的阶越高，则评估就越精确，结果就越有价值。

同阶记号 Θ

- 当且仅当 $f(n) = \mathrm{O}(\mathrm{g}(\mathrm{n}))$ 且 $f(n) = \Omega(g(n))$ 时，

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

例子

- $f(n)=2n+1$， $g(n)=n$， 证明$f(n)=\Theta(g(n))$

- $f(n)=O(g(n))$
- 即证明：存在常数$c>0$，自然数n_0，使得当$n\geq n_0$时，有$f(n)\leq cg(n)$。
- 证明：

$2n+1\leq cn \Rightarrow (c-2)n-1\geq 0$

$c>2, n_0=\lceil 1/(c-2)\rceil$

例：$c=3, n_0=1$

- $f(n)=\Omega(g(n))$
- 即证明：存在常数$c>0$，自然数n_0，使得当$n\geq n_0$时，有$f(n)\geq cg(n)$。
- 证明：

$2n+1\geq cn \Rightarrow (2-c)n+1\geq 0$

$c\leq 2, n_0=\lceil -1/(2-c)\rceil=1$

例：$c=2, n_0=1$

END