第3章

渐进记法

渐进符号的引入

- 确定程序的操作计数和步数有两个重要的原因：
  - 比较两个完成同一功能的程序的时间复杂性；
  - 预测随着实例特征的变化，程序运行时间的变化量。
- 操作计数和步数都不能够非常精确地描述时间复杂性。
  - 操作计数: 把注意力集中在某些“关键”的操作上，而忽略了所有其他操作。
  - 执行步数: 概念本身就不精确。

渐进符号的引入

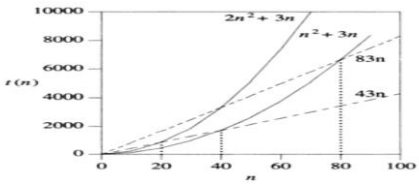
- 引入渐进符号的目的：
  - 描述大型实例特征下，时间复杂性和空间复杂性的具体表现。
  - 渐进符号O使用最普遍

渐进符号的引入

- 如果有两个程序的时间复杂性分别为:
  - $c_1n^2+c_2n$ 和 c_3n,
- 对于足够大的n，复杂性为c_3n的程序将比复杂性为$c_1n^2+c_2n$ 的程序运行得快。
- 对于比较小的n值，两者都有可能成为较快的程序(取决于c_1, c_2和c_3)。
- 如果：c_1=1, c_2=2, c_3=100, 则有
  - $c_1n^2 + c_2n \le c_3n$ n≤98
  - $c_1n^2+c_2n > c_3n$ n>98

时间函数比较示例

- $t_A(n)=n^2+3n$; $t_B(n)=43n$;
- $t_A(n)=2n^2+3n$; $t_B(n)=83n$;



- $t_A(n)=c_1n^2+c_2n+c_3$; $t_B(n)=c_4n$;

渐进符号O, Ω, Θ, o

- 设***f(n)*** 表示程序的时间复杂性或空间复杂性(***n***为实例特征)。
- **O(Big Oh)**符号给出了函数f的一个上限。
- **Ω(Omega)**符号给出了函数f的一个下限。
  Θ(Theta)符号，函数f的上限与下限相同。
- **o(Little oh)**符号。

渐进的大于、小于、等于

- 定义3-1，令p(n)和q(n)是两个非负函数

p(n) 渐进地大于 *q(n)*

当且仅当 $\lim_{n\to\infty}\frac{q(n)}{p(n)}=0$

q(n) 渐进地小于 *p(n)*

当且仅当 *p(n)*渐进的大于*q(n)*

p(n) 渐进地等于 *q(n)*

当且仅当 任何一个都不是渐进的大于另一个

例3-1

- $\lim_{n\to\infty}\frac{10n+7}{3n^2+2n+6}=0$
- $3n^2+2n+6$ 渐进地大于 $10n+7$
- $10n+7$ 渐进地小于 $3n^2+2n+6$

- $8n^4+9n^2$ 渐进地大于 $100n^3-3$
- $2n^2+3n$ 渐进地大于 $83n$
- $12n+6$ 渐进地等于 $6n+2$

f(n) 中的项

- ***f(n)***：表示程序的时间复杂性或空间复杂性（***n***为实例特征）。

因为程序的时间或空间复杂度是一个非负数，假设f对所有n都是非负值

- ***f(n)*** 一般为若干项之和
- 例：***f(n) =3n²+2n+6,***
  - 项：***3n²***， ***2n***， ***6***

*f(n)*中通常出现的项

项	名称
1	常数
logn	对数
n	线性
nlogn	n个logn
n^2	平方
n^3	立方
2^n	指数
n!	阶乘

- **1 < logn < n < nlogn < n^2 < n^3 < 2^n < n!**
  - <：渐进地小于

大O记法

- ***f(n)=O(g(n))*** (读作 “*f(n)* 是*g(n)*的大*O*”)，表示 *f(n)* 渐进地小于或等于 *g(n)*
- $3n^2+2n+6$ 渐进地大于 $10n+7$
  - $10n+7=O(3n^2+2n+6)$;
  - $3n^2+2n+6\neq O(10n+7)$
- $100n^3-3=O(8n^4+9n^2)$
- $8n^4+9n^2\neq O(100n^3-3)$
- $83n=O(2n^2+3n)$
- $12n+6=O(6n+2)$

在渐近的意义上，g(n)是f(n)的上限

渐进复杂性分析

- *f(n)=O(g(n))*
- *f(n)=0*，*g(n)=0*
- 除*f(n)=0*以外，*g(n)*通常是
  - 令*f(n)=O(g(n))*为真的最小单位项（系数为1）
  - $f(n)=10n+7=O(3n^2+2n+6)$
  - $f(n)=10n+7=O(n)$
- $f(n)=8n^4+9n^2=O(n^4)$

  $f(n)=100n^3-3=O(n^3)$

  $f(n)=3n^2+2n+6=O(n^2)$

  $f(n)=12n+6=O(n)$

渐进复杂性分析

- 渐进复杂性分析，用**步数中渐进最大的项**来描述复杂度。
  - f(n)：步数函数
  - 步数函数中最小单位项：系数为1的各项
  - g(n)：最大项(渐进最大的项)
- 例：$f(n)=3n^2+6n\log n+7n+5$
- 最小单位项：n^2、$n\log n$、n、1
- 最大项： n^2
- $f(n)=3n^2+6n\log n+7n+5$
  $=O(n^2)$

渐进记法Ω

- ***f(n)=Ω(g(n))*** (读作 “*f(n)* 是*g(n)*的Ω”
  表示 *f(n)* **渐进地大于或等于** *g(n)*
- $f(n) = 10n + 7 = \Omega(n)$
- $f(n) = 100n^3 - 3 = \Omega(n^3)$
- $f(n) = 3n^2 + 2n + 6 = \Omega(n)$
- $f(n) = 8n^4+9n^2 = \Omega(n^3)$

在渐近的意义上，g(n)是f(n)的下界

渐进记法 Θ

- ***f(n)=Θ(g(n))*** (读作 “*f(n)* 是*g(n)*的Θ”
  表示 *f(n)* **渐进地等于** *g(n)*
- $f(n) = 10n + 7 = \Theta(n)$
- $f(n) = 100n^3 - 3 = \Theta(n^3)$
- $f(n) = 3n^2 + 2n + 6 \neq \Theta(n)$
- $f(n) = 8n^4+9n^2 \neq \Theta(n^3)$

大O记法

- 定义3-3[大O记法]：
- ***f(n)=O(g(n))*** (读作 “*f(n)* 是*g(n)*的大*O*”)，当且仅当存在正的常数*c*和n_0，使得对于所有的*n*, $n \geq n_0$，有$f(n) \leq cg(n)$。
- *g* 是*f* 的一个上限(不考虑常数因子*c*)
  - *O*表示量级(Order)。(最坏情况)
  - *O(g(n))*表示当*n*增大时， *f(n)*至多将以正比于*g(n)*的速度增长。
  - *n*足够大时， *f(n)*不大于*g(n)*的一个常数倍。

大O记法

- 定义3-3[大O记法]：
- ***f(n)=O(g(n))*** 当且仅当存在正的常数*c*和n_0，使得对于所有的*n*, $n \geq n_0$，有$f(n) \leq cg(n)$。

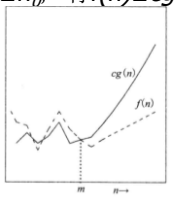


图 3-4 $g(n)$ 是 $f(n)$ 的一个上限（最多带一个常量因子 c）

线性函数

- 例 3-7

$f(n)=\mathbf{3n+2}$

当$n \geqslant n_0=2$时， $f(n)=3n+2 \leqslant 3n+n=4n$

$f(n)=\mathbf{O(n)}$

$f(n)=100n+6$,

当 $n \geqslant n_0=6$，$f(n)=100n+6 \leqslant 100n+n=101n$

$f(n)=$ **$100n+6=O(n)$**。

平方函数

- 例 3-8

$f(n)=10n^2+4n+2$

$n\geq 2,\ f(n)\leq 10n^2+5n$

$n\geq 5,\ 5n\leq n^2$

$n\geq n_0=5,$

$f(n)\leq 10n^2+n^2$

$=11n^2,$

$f(n)=O(n^2)$

$10n^2+4n+2=O(n^2)$

指数函数

- 例 3-9

$f(n)=6*2^n+n^2$

$n\geq 4,\ n^2\leq 2^n,$

$n\geq 4,\ f(n)\leq 6*2^n+2^n=7*2^n$

$6*2^n+n^2=O(2^n)$

常数函数

- 例 3-10

$f(n)=c$

$f(n)=O(1)$

最小上限

- 例 3-11

$f(n)=3n+3$

$n\geq 3,\ f(n)=\mathbf{3n+3}\leq 3n+n=4n=\mathbf{O(n)}$

$n\geq 2,\ f(n)=\mathbf{3n+3}\leq 3n^2=\mathbf{O(n^2)}$(不是最小上限)

因此常用$f(n)=\mathbf{3n+3=O(n)}$

- 语句f(n)=O (g(n)) 仅表明对于所有的$n\geq n_0$，cg(n)是f(n)的一个上限。它并未指出该上限是否为最小上限。
- 为了使语句$f(n)=O(g(n))$有实际意义，其中的g(n)应尽量地小。

Ω符号

- 定义3-4[Ω符号]:
- ***f(n)=Ω(g(n))*** 当且仅当 存在正的常数c和n_0，使得对于所有的$n,\ n\geq n_0$，有$f(n)\geq cg(n)$。
- g 是f的一个下限(不考虑常数因子c)。

$f(n)=3n+2\geq 3n=\mathbf{\Omega(n)}$

$f(n)=10n^2+4n+2\geq 10n^2=\mathbf{\Omega(n^2)}$

$f(n)=6*2^n+n^2\geq 6*2^n=\mathbf{\Omega(2^n)}$

Ω符号

- 定义3-4[Ω符号]:
- ***f(n)=Ω(g(n))*** 当且仅当 存在正的常数c和n_0，使得对于所有的$n,\ n\geq n_0$，有$f(n)\geq cg(n)$。

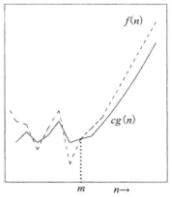


图 3-5 $g(n)$ 是 $f(n)$ 的一个下限（最多加一个常数因子）

最大下限

- f(n)=3n+2≥3n=**Ω**(n)
- f(n)=3n+2=**Ω**(1)
- 为了使语句$f(n)$=**Ω**(g(n))更有实际意义，其中的g(n)应**足够地大**。
- 使用3n+2=**Ω**(n)
- $6*2^n+n^2=\Omega(2^n)$
- $6*2^n+n^2=$**Ω**(1)
- 使用$6*2^n+n^2=\Omega(2^n)$

Θ符号

- 对于所有足够大的n（如$n\geq n_0$），g既是f的上限也是f的下限(不考虑常数因子c)。
- 定义3-5[Θ符号]：***f(n)=Θ(g(n))*** **(**读作“$f(n)$是$g(n)$的**Θ**”，当且仅当存在正常数c_1, c_2和n_0，使得对于所有的n, $n\geq n_0$, 有$c_1g(n)\leq f(n)\leq c_2g(n)$。
- 函数f介于函数g的c_1倍和c_2倍之间，除非n小于n_0。

Θ符号

- 定义3-5[Θ符号]：***f(n)=Θ(g(n))*** **(**读作“$f(n)$是$g(n)$的**Θ**”，当且仅当存在正常数c_1, c_2和n_0，使得对于所有的n, $n\geq n_0$, 有$c_1g(n)\leq f(n)\leq c_2g(n)$。

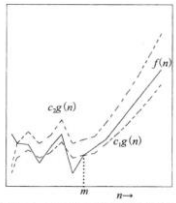


图 3-6 $g(n)$既是$f(n)$的上限也是$f(n)$的下限（最多加一个常数因子）

Θ符号

f(n)=3n+2=**Θ**(n)

$f(n)=10n^2+4n+2=$**Θ**(n^2)

$f(n)=6*2^n+n^2=$**Θ**(2^n)

对 $n\geq 16$，$\log_2 n<10*\log_2 n+4\leq 11*\log_2 n$，

$10*\log_2 n+4=\Theta(\log_2 n)$

O符号有用的结论

- 定理3-1
  如果$f(n)=a_m n^m+\cdots+a_1 n+a_0$ 且$a_m>0$，则$f(n)=O(n^m)$。
- 加法规则：

$$T(n)=T_1(n)+T_2(n)$$
$$=O(g_1(n))+O(g_2(n))$$
$$=O(\max(g_1(n),g_2(n)))$$

- 乘法规则：

$$T(n)=T_1(n)*T_2(n)$$
$$=O(g_1(n))*O(g_2(n))$$
$$=O(g_1(n)*g_2(n))$$

Ω、Θ符号有用的结论

- 定理3-3
  - 如果$f(n)=a_m n^m+\cdots+a_1 n+a_0$且$a_m>0$，则$f(n)=\Omega(n^m)$。
- 定理3-5
  - 如果$f(n)=a_m n^m+\cdots+a_1 n+a_0$且$a_m>0$，则$f(n)=\Theta(n^m)$。

小o记法

- 定义[小o符号]:
- ***f(n)=o(g(n))*** (读作 “*f(n)* 是*g(n)*的小*O*”)，当且仅当 *f(n)=O(g(n))* 且 *f(n)* ≠ Ω*(g(n))*.

常用的渐进符号标记(P74)

	$f(n)$	渐进符号
E1	c	$\Theta(1)$
E2	$\sum_{i=0}^{k} c_i n^i$	$\Theta(n^k)$
E3	$\sum_{i=1}^{n} i$	$\Theta(n^2)$
E4	$\sum_{i=1}^{n} i^2$	$\Theta(n^3)$
E5	$\sum_{i=1}^{n} i^k, k>0$	$\Theta(n^{k+1})$
E6	$\sum_{i=0}^{n} r^i, r>1$	$\Theta(r^n)$
E7	$n!$	$\Theta((n/e)^n)$
E8	$\sum_{i=1}^{n} 1/i$	$\Theta(\log n)$

+可以是O、Ω、Θ之一。

关于渐进符号的推理规则(P74)

I1	$\{f(n)=\oplus(g(n))\} \to \sum_{n=a}^{b} f(n)=\oplus\left(\sum_{n=a}^{b} g(n)\right)$
I2	$\{f_i(n)=\oplus(g_i(n)), 1\le i\le k\} \to \sum_{i=1}^{k} f_i(n)=\oplus(\max_{1\le i\le k}\{g_i(n)\})$
I3	$\{f_i(n)=\oplus(g_i(n)), 1\le i\le k\} \to \prod_{i=1}^{k} f_i(n)=\oplus\left(\prod_{i=1}^{k} g_i(n)\right)$
I4	$\{f_1(n)=O(g_1(n)), f_2(n)=\Theta(g_2(n))\} \to f_1(n)+f_2(n)=O(g_1(n)+g_2(n))$
I5	$\{f_1(n)=\Theta(g_1(n)), f_2(n)=\Omega(g_2(n))\} \to f_1(n)+f_2(n)=\Omega(g_1(n)+g_2(n))$
I6	$\{f_1(n)=O(g(n)), f_2(n)=\Theta(g(n))\} \to f_1(n)+f_2(n)=\Theta(g(n))$

关于⊕的推理规则（⊕∈ {*O*, *Ω*, *Θ*}）

复杂性分析举例：

```
template<class T>
T sum(T a[], int n)
  {//计算a[0:n - 1]中元素之和
  T theSum=0;
  stepCount++; //对应于theSum=0
  for (int i=0; i<n; i++) {
      stepCount++; //对应于for语句
      theSum +=a[i];
      stepCount++; //对应于赋值语句
      }
  stepCount++; //对应于最后一个for语句
  stepCount++; //对应于return语句
  return theSum;
  }
```

步数: 2n+3

$t_{sum}(n)= 2n+3=\Theta(n)$

函数sum(程序1-30)的渐进复杂性

语句	s/e	频率	总步数
T sum(T a[], int n)	0	0	Θ(0)
{	0	0	Θ(0)
T theSum=0;	1	1	Θ(1)
for(int i=0;i<n;i++)	1	n+1	Θ(n)
theSum +=a[i];	1	n	Θ(n)
return theSum;	1	1	Θ(1)
}	0	0	Θ(0)

$t_{sum}(n)= \Theta(\max(g_i(n))) = \Theta(n)$

$\{f_i(n)=\oplus(g_i(n)), 1\le i\le k\} \to \sum_{i=1}^{k} f_i(n)=\oplus(\max_{1\le i\le k}\{g_i(n)\})$

顺序搜索的渐进复杂性

语 句	s/e	频度	步数
int SequentialSearch(T a[],T& x, int n)	0	0	Θ (0)
{	0	0	Θ (0)
int i;	1	1	Θ (1)
for(i=0;i<n&&a[i]!=x;i++)	1	Ω (1), O(n)	Ω (1), O(n)
if(i==n) return -1;	1	1	Θ (1)
return i;	1	Ω (0), O(1)	Ω (0), O(1)
}	0	0	Θ (0)

$t_{SequentialSearch}(n)=\Omega(1)$

$t_{SequentialSearch}(n)=O(n)$

函数sequentialSearch的渐进复杂性

求排列(程序1-32)的渐进复杂性

程序1-32

```
template<class T>
void permutations(T list[], int k, int m)
{ //生成list[k:m]的所有排列方式 , 输出前缀是 list[0:k-1] 后缀是 list[k:m]的所有排列方式
  int i;
  if (k = =m) {// list[k:m]只有一个排列
     copy(list,list+m+1,ostream_iterator<T>(cout, " ")) ;
          cout << endl;
       }
  else // list[k:m ]有多个排列方式，递归地产生这些排列方式
       for (i=k; i <= m; i++) {
            swap (list[k], list[i]);
            permutations (list, k+1, m);
            swap (list[k], list[i]);
          }
}
```

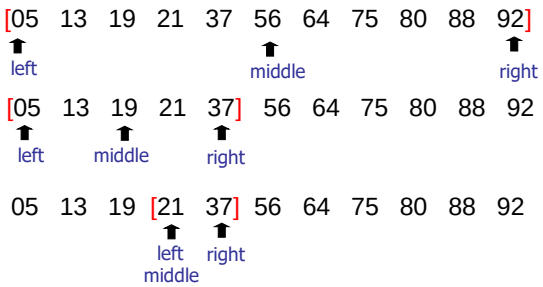

求排列(程序1-32)的渐进复杂性

- 假定m=n-1。
- k=m:所需要的时间为cn(c是一个常数)。
  - $t_{permutations}(k,m) = t_{permutations}(m,m) = cn$
- k<m:执行else语句，
  - for循环将被执行**m-k+1**次
  - 每次循环所花费的时间：$dt_{permutations}(k+1,m)$，d是一个常数.
  - $t_{permutations}(k,m) = d(m-k+1)t_{permutations}(k+1,m)$。使用置换的方法，可以得到：
- $t_{permutations}(0,m) = \Theta((m+1)*(m+1)!) = \Theta(n*n!)$，其中n≥1。

例 3-24 折半搜索 (Binary Search)

- 在有序数组a中查找元素x

搜索过程示例



查找x=21的过程(查找成功)

搜索过程示例

查找x=85的过程(查找失败)

程序3-1 折半搜索 (Binary Search)

```
template<class T>
int binarySearch(T a[], const T& x, int n)
{//在有序数组a中查找元素x
 //如果存在，就返回元素x的位置，否则返回-1
int left=0;   //left指向数据段的左端
int right=n-1 ;//right指向数据段的右端
while (left≤right) {
    int middle=(left+right)/2;//数据段的中间
    if (x= =a[middle]) return middle;
    if (x > a[middle]) left=middle + 1;
    else right=middle-1 ;
    }
return -1; //没有找到x
```

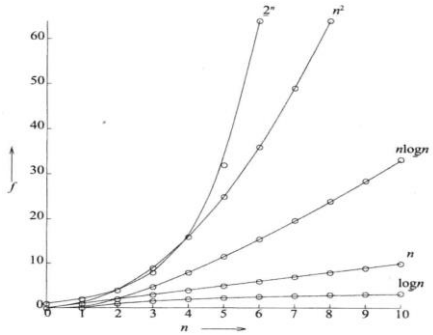
最坏情况下，时间复杂性：

O(logn)

3.5 实际复杂性

logn	n	nlogn	n^2	n^3	2^n
0	1	0	1	1	2
1	2	2	4	8	4
2	4	8	16	64	16
3	8	24	64	512	256
4	16	64	256	4096	65536
5	32	160	1024	32768	4294967296

实际复杂性



实际复杂性

	f(n)						
n	n	$n\log_2 n$	n^2	n^3	n^4	n^{10}	2^n
10	.01μs	.03μs	.1μs	1μs	10μs	10s	1μs
20	.02μs	.09μs	.4μs	8μs	160μs	2.84h	1ms
30	.03μs	.15μs	.9μs	27μs	810μs	6.83d	1s
40	.04μs	.21μs	1.6μs	64μs	2.56ms	121d	18m
50	.05μs	.28μs	2.5μs	125μs	6.25ms	3.1y	13d
100	.10μs	.66μs	10μs	1ms	100ms	3171y	$4*10^{13}$y
10^3	1μs	9.96μs	1ms	1s	16.67m	$3.17*10^{13}$y	$32*10^{283}$y
10^4	10μs	130μs	100ms	16.67m	115.7d	$3.17*10^{23}$y	
10^5	100μs	1.66ms	10s	11.57d	3171y	$3.17*10^{33}$y	
10^6	1ms	19.92ms	16.67m	31.71y	$3.17*10^7$y	$3.17*10^{43}$y	

1 微秒（μs）=10^{-6} 秒　1 毫秒（ms）=10^{-3} 秒

s= 秒　m= 分钟　h= 小时　d= 天　y= 年

图 3-15　在一台每秒 1 000 000 000 条指令的计算机上的运行时间

第4章

性能测量

性能测量

- 性能测量（performance measurement）主要关注于得到一个程序实际需要的空间和时间。
- 空间密切相关：
  - 特定的编译器
  - 编译器选项
  - 执行程序的计算机
- 不能精确地测量一个程序运行时所需要的空间
- 程序的运行时间：使用C++函数clock()

时间测量

- 测量程序(以排序为例)，需要
  - 确定实例特征n的一组值
  - 对于实例特征n的每一个值，设计测试数据
    - 可以人工设计或借助计算机设计相应的测试数据
  - 编写程序，测量运行时间
    - 为了提高测量的精确度，对于实例特征的每一个值，可以重复求解若干次。
    - 实际测量时间包括：排序的时间、额外时间(每次对a初始化等）