1.4 算法和算法分析

1.4.1 算法及其描述

- 算法是对特定问题求解步骤的一种描述，它是指令的有限序列，其中每条指令表示一个或多个操作。

一个算法具有下列5个重要特性：

- **有穷性**：一个算法必须总是（对任何合法的输入值）在执行有限步之后结束，且每一步都可在有限时间内完成。有穷的概念不是纯数学的，而是在实际上是合理的，可接受的。
- **确定性**：算法中每一条指令必须有确切的含义，不会产生二义性。
- **可行性**：一个算法是能行的，即算法中描述的操作都是可以通过已经实现的基本运算执行有限次来实现的。
- **输入性**：一个算法有零个或多个输入。
- **输出性**：一个算法有一个或多个输出。

1.4.2 算法设计的要求

算法设计应满足以下几条目标：

- **正确性**：算法应当满足具体问题的需求。这是最重要也是最基本的标准。
- “正确”一词的含义，大体可分为以下4个层次：
  - ✓ a.程序不含语法错误；
  - ✓ b.程序对于几组输入数据能够得出满足规格说明要求的结果；
  - ✓ c.程序对于精心选择的典型、苛刻而带有刁难性的几组输入数据能够得出满足规格说明要求的结果；
  - ✓ d.程序对于一切合法的输入数据都能产生满足规格说明要求的结果。

- **可读性**：算法主要是为了人的阅读与交流，其次才是机器执行。可读性好有助于人对算法的理解；晦涩难懂的程序易于隐藏较多错误，难以调试和修改。为了达到这个要求，算法的逻辑必须是清晰的、简单的和结构化的。
- **健壮性**：当输入数据非法时，算法也能适当地做出反应或进行处理，而不会产生莫明其妙的输出结果。
- **高效率与低存储量需求**。通俗地说，效率指的是算法执行的时间。对于同一个问题如果有多个算法可以解决，执行时间短的算法效率高。存储量需求指算法执行过程中所需要的最大存储空间。效率与低存储量需求这两者都与问题的规模有关。

【示例1】请阅读下列两个算法：

算法1：

```
void fun1()
{  int n;
   n=2;
   while (n%2==0)
        n=n+2;
   printf("%d\n",n);
}
```

算法2：

```
void fun2()
{  int x=10,y;
   y=0;
   x=x/y;
   printf("%d,%d\n",x,y);
}
```

试问它们存在的问题。

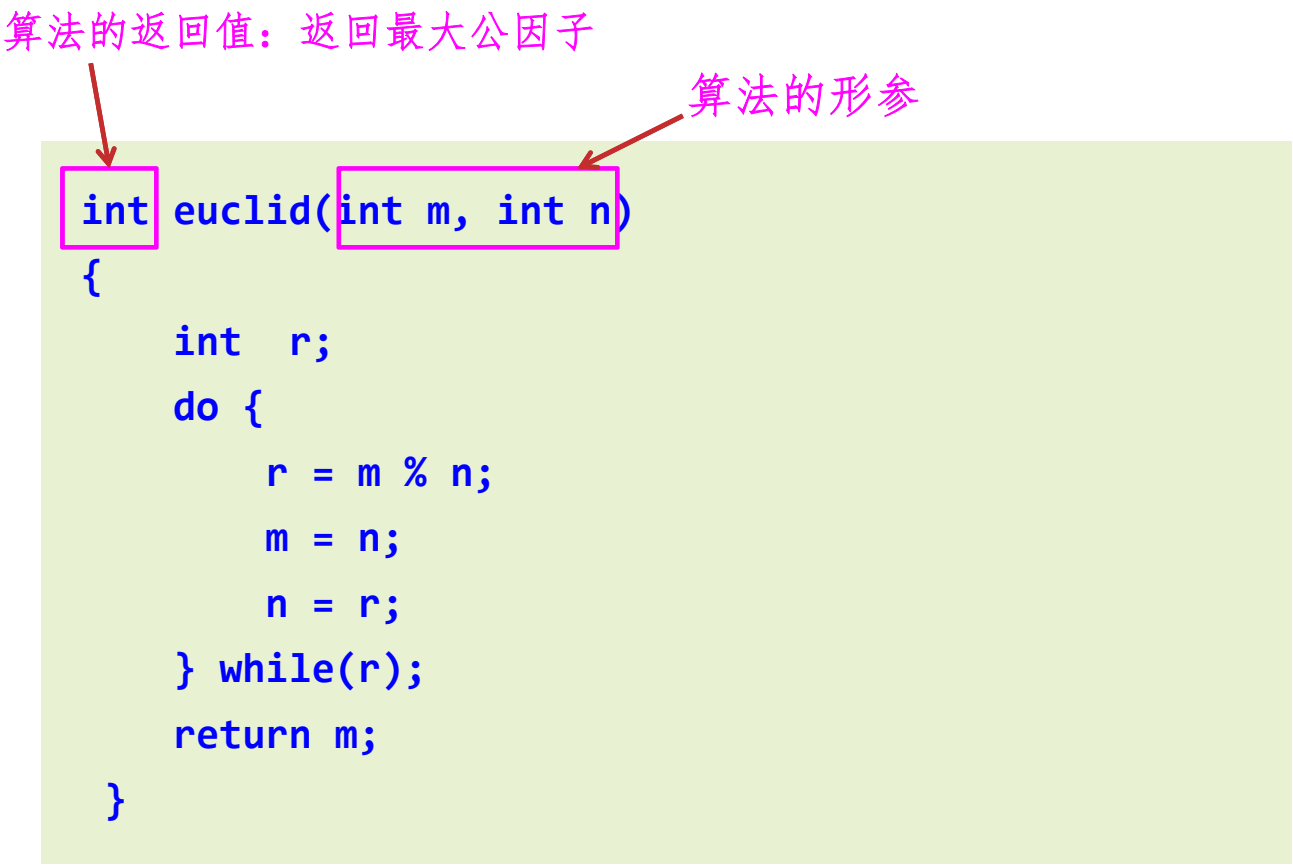
算法1是一个死循环，违反了算法的有穷性特性。

算法2出现除零错误，违反了算法的可行性特性。

算法如何描述？

- 描述算法的方式可以有多种形式。如采用某个高级语言，或者采用自然语言，或者混合形式。
- 本书采用C/C++语言来描述算法的实现。通常采用C/C++函数来描述算法。

【示例2】以设计求两个数m,n的最大公因子的算法为例说明C/C++语言描述算法的一般形式，该算法如下所示。




```
int euclid(int m, int n)
{
    int  r;
    do {
        r = m % n;
        m = n;
        n = r;
    } while(r);
    return m;
 }
```

1.4.3 算法效率的度量

- 计算机资源主要包括计算时间和内存空间。
- 算法分析是分析算法占用计算机资源的情况。所以算法分析的两个主要方面是分析算法的时间复杂度和空间复杂度。
- 算法分析的目的不是分析算法是否正确或是否容易阅读，主要是考察算法的时间和空间效率，以求改进算法或对不同的算法进行比较。

有两种衡量算法效率的方法：

- 事后统计的方法。
- 事前分析估算的方法。

（1）事后统计法存在的缺点：

- ✓ 一是必须先运行依据算法编制的程序；
- ✓ 二是所得时间的统计量依赖于计算机的硬件、软件等环境因素，有时容易掩盖算法本身的优劣。

- 因此人们常常采用另一种事前分析估算的方法。

(2) 事前分析估算的方法

一个用高级程序语言编写的程序在计算机上运行时所消耗的时间取决于下列因素：

① 依据的算法选用何种策略；

② 问题的规模，例如求100以内还是1000以内的素数；

③ 书写程序的语言，对于同一个算法，实现语言的级别越高，执行效率就越低；

④ 编译程序所产生的机器代码的质量；

⑤ 机器执行指令的速度。

- 这表明使用绝对的时间单位衡量算法的效率是不合适的。
- 撇开这些与计算机硬件、软件有关的因素，可以认为一个特定算法“运行工作量”的大小，只依赖于问题的规模(通常用整数量*n*表示），或者说，它是问题规模的函数。

1．算法时间复杂度分析

- 一个算法是由控制结构（顺序、分支和循环3种）和原操作（指固有数据类型的操作）构成的。
- 算法的运行时间取决于两者的综合效果。

例如，如下素数的判断算法prime，其中形参为*n*。

```
void prime(int n)
/* n是一个正整数 */
{
    int i = 2 ;
    while((n % i) != 0 && i * 1.0 < sqrt(n))
        i++;
    if(i * 1.0 > sqrt(n))
        printf("%d 是一个素数\n", n);
    else
        printf("%d 不是一个素数\n", n);
}
```

- 算法的执行时间主要与问题规模n有关。例如，整数n的大小、数组的元素个数、矩阵的阶数等都可作为问题规模。
- 所谓一个语句的频度，即指该语句在算法中被重复执行的次数。
- 算法中所有语句的频度之和记做$f(n)$，它是问题规模n的函数。
- 当问题规模n趋向无穷大时，$f(n)$的数量级(Order)称为渐进时间复杂度，简称为时间复杂度，记作$T(n)=O(f(n))$。

“O”的含义是为$f(n)$找到了一个上界$g(n)$，其严格的数学定义是：

$f(n)$的数量级表示为$O(g(n))$，是指存在常量$c\neq0$和n_0（为一个足够大的常量），使得

$$\lim_{n\to n0}\frac{|f(n)|}{|g(n)|}=c\neq 0$$

成立，记为$f(n)=O(g(n))$。

算法的时间量度记作：$T(n)=O(f(n))$

- 它表示随问题规模n的增大，算法执行时间的增长率和$f(n)$的增长率相同，称做算法的渐近时间复杂度（asymptotic time complexity），简称时间复杂度。

- $f(n)=5n^3-2n^2+3n-100$，则$T(n)=O(n^3)$，因为

$$\lim_{n\to n0}\frac{|f(n)|}{|n^3|}=5$$

- $T(n)\neq O(n^4)$，$\lim_{n\to n0}\frac{|f(n)|}{|n^4|}=\frac{5}{n}$，当$n$足够大时，值趋于0。
- $T(n)\neq O(n^2)$，$\lim_{n\to n0}\frac{|f(n)|}{|n^2|}=5n$，当$n$足够大时，值趋于∞。

- 定理：若$f(n)=a_m n^m+a_{m-1}n^{m-1}+\ldots+a_1 n+a_0$是一个$m$次多项式，则$T(n)=O(n^m)$

- 算法中基本运算语句的频度$f(n)$与$T(n)$同数量级。一般地，常用最深层循环内的语句中的原操作的执行频度(重复执行的次数)来表示。

- 一个没有循环的算法中基本运算次数与问题规模n无关，记作$O(1)$，也称作常量阶。
- 一个只有单循环的算法中基本运算次数与问题规模n的增长呈线性增大关系，记作$O(n)$，也称线性阶。
- 常用的还有平方阶$O(n^2)$、立方阶$O(n^3)$、对数阶$O(\log_2 n)$、指数阶$O(2^n)$等等。

不同数量级对应的值存在着如下关系：

$O(1)<O(\log_2 n)<O(n)<O(n\log_2 n)<O(n^2)<O(n^3)<O(2^n)<O(n!)$

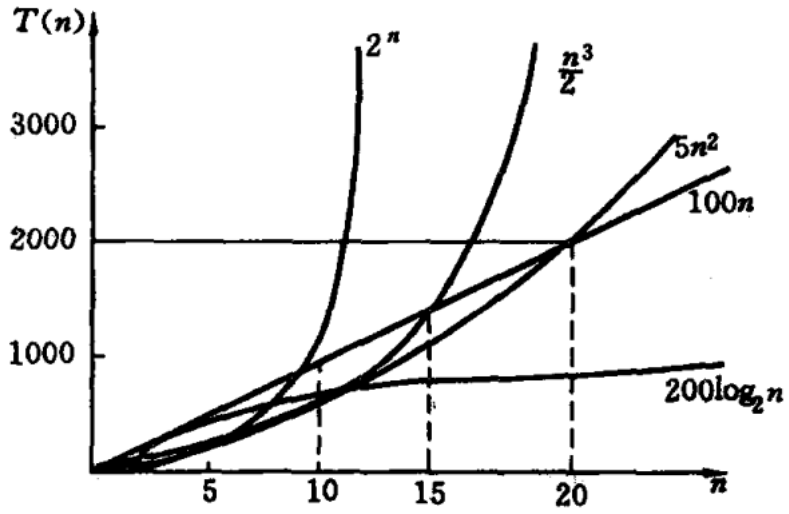


图 1.7　常见函数的增长率

对于求1+2+…+*n*值，不同的算法设计，如算法1不如算法2好。

```
int Sum1(int n)
{  int i, s;
   if (n<=0) return 0;
   s=0;
   for (i=1;i<=n;i++)
     s+=i;
   return s;
}
```

算法1

```
int Sum2(int n)
{  if (n<0) return 0;
   else return n*(n+1)/2;
}
```

算法2

因为算法1的时间复杂度为O(*n*)，而算法2的时间复杂度为O(1)。

1.4.4 算法空间复杂度分析

- 一个上机执行的程序除了需要存储空间来寄存本身所用指令、常数、变量和输入数据外，也需要一些对数据进行操作的工作单元和存储一些为实现计算所需信息的辅助空间，如形参所占空间和局部变量所占空间等。
- 对算法进行存储空间分析时，只考察局部变量所占空间。
- 算法的临时空间一般也作为问题规模n的函数，以数量级形式给出，记作：$S(n) = O(g(n))$，其中“O”的含义与时间复杂度中的含义相同。

```
int max(int a[],int n)
{
   int i,maxi=0;
   for (i=1;i<=n;i++)
      if (a[i]>a[maxi])
         maxi=i;
   return a[maxi];
}
```

- ✓ 函数体内分配的变量空间为临时空间，不计形参占用的空间；
- ✓ 这里仅计i、maxi变量的空间，其空间复杂度为O(1)。

- 若算法所需临时空间相对于输入数据量来说是常数，则称此算法为原地工作或就地工作。
- 若所需临时空间依赖于特定的输入，则通常按最坏情况来考虑。

END