1.4 算法和算法分析

1.4.1 算法

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，其中每一条指令表示一个或多个操作。一个算法必须满足以下五个重要特性：

1．有穷性　2．确定性　3．可行性

4．输入　5．输出

1．**有穷性**　对于任意一组合法输入值，在执行有穷步骤之后一定能结束，即：算法中的每个步骤都能在有限时间内完成。

2．**确定性**　对于每种情况下所应执行的操作，在算法中都有确切的规定，使算法的执行者或阅读者都能明确其含义及如何执行。并且在任何条件下，算法都只有一条执行路径，不允许有二义性。

例如下面这句话：‘张三对李四讲，他的儿子考上了大学。’

确切含义？

3．可行性　算法中的所有操作都必须足够基本，都可以通过已经实现的基本操作运算有限次实现之。

4．有输入　一个算法有零个或多个的输入，这些输入取自于某个特定的对象的集合。

5．有输出　一个算法有一个或多个的输出，它是与“输入”有确定关系的量值，是算法进行信息加工后得到的结果。

1.4.2 算法设计的要求

设计算法时，通常应考虑达到以下目标：

1．正确性

2. 可读性

3．健壮性

4．高效率与低存储量需求

1．正确性

首先，算法应当满足以特定的“规格说明”方式给出的需求。

其次，对算法是否“正确”的理解可以有以下四个层次：

a．程序中不含语法错误；

b．程序对于几组输入数据能够得出满足要求的结果；

c．程序对于精心选择的、典型、苛刻且带有刁难性的几组输入数据能够得出满足要求的结果；

d．程序对于一切合法的输入数据都能得出满足要求的结果；

通常以第 **c** 层意义的正确性作为衡量一个算法是否合格的标准。

2. 可读性

算法主要是为了人的阅读与交流，其次才是为计算机执行，因此算法应该易于人的理解；另一方面，晦涩难读的程序易于隐藏较多错误而难以调试。

3．健壮性

当输入的数据非法时，算法应当恰当地作出反映或进行相应处理，而不是产生莫名奇妙的输出结果。并且，处理出错的方法不应是中断程序的执行，而应是返回一个表示错误或错误性质的值，以便在更高的抽象层次上进行处理。

4．高效率与低存储量需求

通常，效率指的是算法执行时间；存储量指的是算法执行过程中所需的最大存储空间，两者都与问题的规模有关。

1.4.3 算法效率的度量

算法执行时间需通过依据该算法编制的程序在计算机上运行时所消耗的时间来度量。通常有两种度量程序执行时间的方法。

事后统计法：

缺点：1．必须执行程序

2．其它因素掩盖算法本质

事前分析估算法：

一个用高级语言编写的程序在计算机上运行时所消耗时间取决于下列因素：

1．算法选用的策略

2．问题的规模

3．编写程序的语言

4．编译程序产生的机器代码的质量

5．计算机执行指令的速度

撇开与计算机硬件、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。

一般情况下，算法中基本操作重复执行的次数是问题规模 n 的某个函数f(n)，算法的时间度量记作

$$T(n) = O(f(n))$$

它表示随着问题规模 n 的增长，算法执行时间的增长率和 f（n）的增长率相同，称作算法的（渐近）时间复杂度（Asymptotic Time Complexity），简称**时间复杂度**。

如何估算算法的时间复杂度？

算法 = 控制结构 + 原操作 （固有数据类型的操作）

算法的执行时间 =

∑ 原操作（i）的执行次数×原操作（i）的执行时间

算法的执行时间与原操作执行次数之和成正比

从算法中选取一种对于所研究的问题来说是基本操作的原操作，以该基本操作在算法中重复执行的次数作为算法运行时间的衡量准则。

例一：两个矩阵相乘

```
void mult(int a[], int b[], int& c[] ) {
  // 以二维数组存储矩阵元素，c 为 a 和 b 的乘积
   for (i=1; i<=n; ++i)
      for (j=1; j<=n; ++j) {
         c[i,j] = 0;
         for (k=1; k<=n; ++k)
            c[i,j] += a[i,k]*b[k,j];
      } //for
} //mult
```

基本操作：**乘法操作**

时间复杂度：$O(n^3)$

例二：选择排序

```
void select_sort(int&a[], int n) {
  // 将 a 中整数序列重新排列成自小至大有序的整数序列。
  for ( i = 0;  i< n-1;  ++i ) {
    j = i ;    // 选择第 i 个最小元素
    for ( k = i+1;  k < n;  ++k )
      if (a[k] < a[j] )  j = k;
      if ( j != i )  a[j] ←→ a[i]
    }
} // select_sort
```

基本操作：**比较（数据元素）操作**

时间复杂度：**O（n^2）**

1.4.4 算法的存储空间需求

算法的空间复杂度定义为:

$$S(n) = O(g(n))$$

表示随着问题规模 n 的增大，算法运行所需存储量的增长率与 g(n) 的增长率相同。

算法的存储量包括:
1．输入数据所占空间
2．程序本身所占间
3．辅助变量所占空间

若**输入数据**所占空间只取决于问题本身，**和算法无关**，则只需要分析除输入和程序之外的**辅助变量**所占**额外空间**。

若所需额外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**。

若所需存储量依赖于特定的输入，则通常按最坏情况考虑。