高等学校计算机科学与技术教材

- 原理与技术的完美结合
- 教学与科研的最新成果
- 语言精炼，实例丰富
- 可操作性强，实用性突出

计算机算法与实践教程

□ 徐保民　陈旭东　李春艳　编著

清华大学出版社　●　北京交通大学出版社

高等学校计算机科学与技术教材

计算机算法与实践教程

徐保民　陈旭东　李春艳　编著

清华大学出版社
北京交通大学出版社
·北京·

内 容 简 介

本书较系统地阐述算法设计的方法、技术和应用实例。全书内容包括算法基础、常用数据结构、基本算法设计技术、贪心法、分治法、回溯法和分枝定界法，内容丰富，概念清楚，通俗易懂。全书特别注重对实际应用问题的分析和理解，全部算法实例都给出了完整的程序实现，并对算法的工作过程进行说明，使算法更加易于理解和掌握。

本书可作为高等院校各专业算法实践类课程用教材，也可以作为程序设计类课程、算法类课程和数据结构课程的辅助用书，同时还可以作为算法爱好者和参加各种程序设计比赛选手的自学用书。

图书在版编目(CIP)数据

计算机算法与实践教程/徐保民，陈旭东，李春艳编著. —北京：清华大学出版社；北京交通大学出版社，2007.8
（高等学校计算机科学与技术教材）
ISBN 978－7－81123－149－6

Ⅰ. 计…　Ⅱ. ①徐…　②陈…　③李…　Ⅲ. 电子计算机－算法理论－高等学校－教材　Ⅳ. TP301.6

中国版本图书馆 CIP 数据核字（2007）第 103160 号

责任编辑：谭文芳　　特邀编辑：田　平
出版发行：清华大学出版社　　邮编：100084　　电话：010－62776969　　http://www.tup.com.cn
　　北京交通大学出版社　　邮编：100044　　电话：010－51686414　　http://press.bjtu.edu.cn
印 刷 者：北京东光印刷厂
经　　销：全国新华书店
开　　本：185×260　　印张：12.75　　字数：322 千字
版　　次：2007 年 7 月第 1 版　　2007 年 8 月第 1 次印刷
书　　号：ISBN 978－7－81123－149－6/TP·367
印　　数：1～4 000 册　　定价：22.00 元

前　言

Berlin Ski 博士曾说:“有两种思想,像珠宝商放在天鹅绒上的宝石一样熠熠生辉,一个是微积分,另一个就是算法。微积分造就了现代科学,而算法则造就了现代世界。”

作为计算机科学与技术学科核心地位的专业必修课程之一的算法,对从事计算机系统结构、系统软件及应用软件研究和开发人员来讲都是非常重要的和必不可少的。

目前对于计算机算法介绍的教材通常有两类:一类着重介绍的是数据结构本身的实现,即数据结构与算法;另一类着重介绍的是算法设计的原理,即算法设计与分析。本教程则从算法实践的角度出发,着重介绍几种常用算法的基本思想、完整实现及其应用,使读者能用学过的理论知识解决具体、复杂的实际问题,从而达到应用型人才培养的目标。

本书以算法实践为知识单元,以期为读者提供坚实的计算机算法的应用知识。全书共分8章。

第1章　算法基础知识。主要介绍了算法的基本概念,并简要描述了算法以及算法的计算复杂性。

第2章　线性数据结构与算法。主要介绍堆和栈的结构特性及基于这些结构的一些常用算法。

第3章　非线性数据结构与算法。主要介绍树和图的结构特性及基于这些结构的一些常用算法。

第4章　递归与分治。主要介绍递归的概念和分治法的基本思想及基于递归或分治思想所解决的经典问题。

第5章　贪心法。主要介绍贪心算法的思想,利用贪心算法解决实际问题。

第6章　动态规划。主要介绍动态规划法的适用性和算法的设计要点及利用动态规划法所解决的经典问题。

第7章　回溯法。主要以若干实例为背景介绍适合于求解组合数较大的问题的回溯法的思想及其使用方法。

第8章　分枝定界。主要通过对具体问题求解过程的分析,介绍了分枝定界法的主要思想及使用范畴。

本书各章节的内容在安排上,都是先介绍一种算法的基本思想,然后从解决计算机科学与应用中出现的实际问题入手,由简到繁,描述若干经典的算法与应用。书中采用 Java 语言给出了算法的完整程序实现,所有程序全部通过作者的调试运行。每章后都附有相当数量的习题,便于加深对本章所讲述内容的理解、巩固所学的知识及加强实际动手能力。

参与本书编写的老师,多年来从事程序设计、数据结构和算法实践类课程的教学,积累了丰富的教学经验,书中的很多内容都是他们教学经验和研究课题的总结。本书第1～6章由徐保民编写,第7、8章由李春艳编写,程序调试由徐保民和陈旭东完成。全书由陈旭东负责统编和定稿。

在本书的编写过程中，得到了清华大学出版社、北京交通大学出版社及段连平老师、谭文芳老师的大力支持和帮助。张宏勋、高强、孙丽君、连卫民、石磊等老师为本书的编写也提出了不少具体建议，在此致以诚挚的谢意。

由于作者水平所限，书中难免会存在错误和缺点，殷切期望广大读者批评指正，电子邮箱xudong _ chen@tom.com。

编　者

2007 年 7 月

目　录

第 1 章　算法基础知识 …… 1
1.1　算法简介 …… 1
1.2　算法描述形式 …… 2
1.3　算法复杂性分析 …… 3
1.3.1　时间复杂度 …… 3
1.3.2　空间复杂度 …… 5
习题 …… 6
第 2 章　线性数据结构与算法 …… 7
2.1　线性表 …… 7
2.1.1　线性表定义及特点 …… 7
2.1.2　顺序表 …… 7
2.1.3　链表 …… 10
2.1.4　数组与链表性能比较 …… 16
2.2　栈与队列 …… 17
2.2.1　栈 …… 17
2.2.2　队列 …… 19
2.3　应用举例 …… 24
2.3.1　队列应用举例 …… 24
2.3.2　栈应用举例 …… 26
习题 …… 30
第 3 章　非线性数据结构与算法 …… 33
3.1　树与二叉树 …… 33
3.1.1　树的基本概念 …… 33
3.1.2　二叉树 …… 34
3.1.3　树与二叉树的存储结构 …… 35
3.1.4　树的搜索 …… 39
3.2　图 …… 67
3.2.1　图的基本概念 …… 68
3.2.2　图的存储结构 …… 69
3.2.3　图的搜索 …… 74
3.3　应用举例 …… 77
3.3.1　课程安排问题 …… 77
3.3.2　关键路径问题 …… 80

习题 …… 84
第 4 章 递归与分治 …… 87
4.1 递归法 …… 87
4.1.1 递归思想 …… 87
4.1.2 应用举例 …… 88
4.2 分治法 …… 95
4.2.1 问题的提出 …… 95
4.2.2 分治法概述 …… 96
4.2.3 应用举例 …… 98
习题 …… 109
第 5 章 贪心法 …… 112
5.1 问题的提出 …… 112
5.2 贪心法概述 …… 112
5.3 应用举例 …… 114
5.3.1 哈夫曼编码 …… 114
5.3.2 单源最短路径 …… 118
5.3.3 最小生成树 …… 123
5.3.4 背包问题 …… 129
5.3.5 多机调度问题 …… 132
5.3.6 马踏棋盘问题 …… 135
习题 …… 138
第 6 章 动态规划 …… 141
6.1 问题的提出 …… 141
6.2 动态规划法概述 …… 142
6.3 应用举例 …… 144
6.3.1 多源最短路径 …… 144
6.3.2 背包问题 …… 146
6.3.3 图像压缩 …… 149
6.3.4 最长公共子序列问题 …… 151
习题 …… 154
第 7 章 回溯法 …… 158
7.1 问题的提出 …… 158
7.2 回溯法概述 …… 159
7.3 应用举例 …… 161
7.3.1 背包问题 …… 161
7.3.2 n 皇后问题 …… 166
7.3.3 组合问题 …… 168
7.3.4 填字游戏 …… 170
习题 …… 172

第 8 章　分枝定界 …… 176
8.1　问题的提出 …… 176
8.2　分枝定界法概述 …… 176
8.3　应用举例 …… 177
8.3.1　背包问题 …… 178
8.3.2　旅行商问题 …… 188
习题 …… 192
参考文献 …… 194

第 1 章　算法基础知识

算法不仅是计算机科学的一个分支，它更是计算机科学的核心。本章主要介绍有关算法的一些基本概念、算法的描述形式及算法复杂性分析等内容。

1.1　算法简介

计算机的问世是 20 世纪人类最伟大的发明之一，它把人类社会带进了信息技术时代，而算法是计算机科学的重要基础，就像算盘一样，人们需要为计算机编制各种各样的“口诀”即算法(Algorithm)，才能使其工作。

对于计算机科学来讲，算法的概念是至关重要的，但是，究竟什么是算法？至今还没有一个完全统一的定义。一般而言，算法是指解决那些适合于计算机实现的问题的方法或过程。严格地讲，算法是对特定问题求解步骤的一种描述。例如，计算机用于解决数值计算，如科学计算中的数值积分、解线性方程等的计算方法，就是数值计算的算法；用于解决非数值计算，如用于管理、文字处理、图像图形等的排序、分类、查找，就是非数值计算的算法。

尽管算法并不给出问题的精确解，只是说明怎样才能得到解，但是，算法通常都是由有限个操作组成的。这些操作包括加、减、乘、除、判断、赋值等，并按顺序、分支、循环等结构组织在一起。

作为一个算法，应该具备如下 5 个特性。

(1) 输入性

一个算法要具有 0 个或多个外部量作为算法的输入。这些外部量通常体现为算法中的一组变量。有些输入量需要在算法执行过程中输入。从表面上看，有些算法好像没有输入量，实际上是输入量已被嵌入到算法之中。

(2) 输出性

一个算法必须具有一个或多个输出，以反映算法对输入数据加工后的结果。没有输出的算法是毫无意义的。

(3) 确定性

算法的每一个步骤必须具有确定的定义，即每一步要执行的动作是确定的，是无二义性的。在任何条件下，算法只有唯一的一条执行路径，即对于相同的输入得出的输出结果也是相同的。

(4) 有穷性

对于任何合法的输入值，算法必须在执行有限个步骤之后结束，并且每一步都可以在有限的时间内完成。

(5) 可行性

算法中描述的操作都是可以通过已经实现的基本运算的有限次执行来实现，即算法的具体实现应该能够被计算机执行。

假如现在有一个问题:依据如下分段函数,要求对用户输入的一个自变量 x 的值,给出相应的函数值。

$$f(x)=\begin{cases}x+2, & x>0\\ 2, & x=0\\ x-2, & x<0\end{cases}$$

实现分段函数求值功能的过程如下。

步骤 1,输入自变量 x 的值。

步骤 2,依据自变量 x 的值进行判断:

- 如果 $x>0$,执行 $x+2\Rightarrow f(x)$操作;
- 如果 $x<0$,执行 $x-2\Rightarrow f(x)$操作;
- 如果 $x=0$,执行 $2\Rightarrow f(x)$操作。

其中,$A\Rightarrow B$ 表示将表达式 A 的值赋给变量 B。

步骤 3,输出步骤 2 计算的结果。

显然,上述描述满足算法的 5 个重要特征,因此,该描述就是一个算法。

在现实社会中,不同的人对于同一问题会有不同的看法或解决方法。同样,在计算机领域,对于同一问题存在多种算法也是很自然的事情。例如对于一批数据的排序问题,就存在多种排序方法。判断一个算法的好坏主要依据如下 4 个标准。

(1) 正确性

正确性是设计一个算法的首要条件,如果一个算法不正确,其他方面就无从谈起。一个正确的算法是指在合理的数据输入下,能在有限的时间内得出正确的结果。

(2) 可读性

算法主要是为了人的阅读与交流,其次才是让计算机执行,因此算法应该易于人的理解;另一方面,晦涩难读的算法易于隐藏较多错误而使实现该算法的程序的调试工作变得更加困难。

(3) 健壮性

算法应当具备检查错误和对错误进行适当处理的能力。一般而言,处理错误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) 效率

效率是指算法执行时所需计算机资源的多少,包括运行时间和存储空间两方面的要求。运行时间和存储空间都与问题的规模有关。存储空间指的是算法执行过程中所需的最大存储空间。

在设计一个算法时,要从上述 4 个方面综合考虑。同时还要考虑到算法的使用频率及所使用机器的软硬件环境等因素,这样才能设计出一个好的算法。

1.2 算法描述形式

算法的描述形式多种多样,不同的算法描述形式对算法的质量有一定的影响。描述同一个算法可以采用自然语言、流程图、盒图、伪代码、程序设计语言等。常用的描述算法方法有如下 3 种。

(1) 自然语言描述法

最简单的描述算法的方法是使用自然语言。用自然语言来描述算法的优点是简单且便于人们对算法的理解和阅读;缺点是不够严谨,易产生歧义。当算法比较复杂且包含很多转移分支时,用自然语言描述就不是那么直观清晰了。

(2) 算法框图法

使用程序流程图、盒图等算法描述工具来描述算法。其特点是简洁、明了,便于理解和交流。

(3) 伪码语言描述法

用上述两种方法描述的算法并不能够直接在计算机上执行。为了解决理解与执行之间的矛盾,人们常常使用一种称为伪码语言的描述方法来对算法进行描述。伪码语言介于高级程序设计语言和自然语言之间,它忽略高级程序设计语言中一些严格的语法规则与描述细节,因此它比程序设计语言更容易描述和被人理解,而比自然语言或算法框图更接近程序设计语言。

当然,大部分的算法最终是需要通过能够向计算机发送一系列命令的程序来实现的。所谓"程序"是指对所要解决问题的各个对象和处理规则的描述,或者说是数据结构和算法的描述,因此有人说,数据结构 + 算法 = 程序。

程序与算法不同。程序可以不满足算法的第 4 个特性。例如操作系统,它是在无限循环中执行的程序,因而不是算法。然而可把操作系统的各种任务看作一些单独的问题,每一个问题由操作系统中的一个子程序通过特定的算法实现,该子程序得到输出结果后便终止。

在解决实际问题时,要针对问题的不同特性,要考虑利用什么样的程序语言才是最合适的。在本书中,采用 Java 语言给出算法的完整实现。

1.3　算法复杂性分析

算法复杂性的度量主要是针对运行该算法所需要的计算机资源的多少。当算法所需要的资源越多,该算法的复杂性越高;反之,当算法所需要的资源越少,算法的复杂性越低。

对于任意给定的一个问题,设计出复杂性尽可能低的算法是在设计算法时追求的重要目标之一;而当给定的问题存在多种算法时,选择其中复杂性最低的算法是选用算法时遵循的重要准则。因此,算法的复杂性分析对算法的设计或选用具有重要的指导意义和实用价值。

1.3.1　时间复杂度

通常,对于一个算法的复杂性分析主要是对算法效率的分析,包括衡量其运行速度的时间效率及衡量其运行时所需要占用空间大小的空间效率。

对于早期的计算机来说,时间与空间都是极其珍贵的资源。由于硬件技术的发展大大提高了计算机的存储容量,使得存储容量的局限性对于算法的影响大大降低。但是时间效率并没有得到相应程度的提高。因此,算法的时间效率或算法时间复杂度是算法分析中的关键所在。

对于算法的时间效率的计算,通常是抛开与计算机硬件、软件有关的因素,仅考虑实现该算法的高级语言程序。一般而言,对程序执行的时间复杂度的分析是分块进行的,先分析程序中的语句,再分析各程序段,最后分析整个程序的执行复杂度。通常以渐进式的大 O(希腊字母 Omicron,奥米克戎)形式来表示算法的时间复杂度。渐进式的大 O 形式表示时间复杂度

的主要运算规则有如下 2 种。

(1) 求和规则

$$O(f(n))+O(g(n))=O(\max\{f(n),g(n)\})$$

其中,$f(n)$和 $g(n)$表示与 n 有关的一个函数。

(2) 乘法规则

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

$$O(cf(n))=O(f(n)),\ c \text{ 是一个正数。}$$

假设 $T(n)$是问题规模 n(n 为整数)的函数,算法的时间复杂度可以定义为 $O(f(n))$,记作:

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

由于随着问题规模 n 的增长,算法执行时间的增长率和 $f(n)$的增长率相同,因此 $T(n)$也被称为算法的时间复杂度。

为了便于比较同一问题的不同算法的效率问题,通常的做法是从算法中选取一种对于所研究问题来说是基本运算的原子操作,以该基本操作重复执行的次数作为算法的时间度量单位。

例如,对于如下的矩阵相乘算法:

```
for ( i=0; i<n; i++){
   for( j=0; j<n; j++){
          c[i][j]=0;
          for( k=0; k<n; k++){
                 c[i][j]=c[i][j]+a[i][k]*b[k][j];
          }
}
}
```

第 1 个 for 语句的执行次数为 $n+1$;第 2 个 for 语句的执行次数为 $n(n+1)$;语句 c[i][j]=0 的执行次数是 n^2;第 3 个 for 语句的执行次数为 $n^2(n+1)$;语句 c[i][j]=c[i][j]+a[i][k]*b[k][j] 的执行次数是 n^3。该算法的时间复杂度为所有语句的执行次数的总和,即:

$$T(n)=n+1+n(n+1)+n^2+n^2(n+1)+n^3=2n^3+3n^2+2n+1$$

如果根据渐进式大 O 规则,则该算法的时间复杂度为 $O(n^3)$。

在某些情况下,算法所包含的基本操作重复执行的次数会随着问题的输入数据集不同而不同。例如下面所示的冒泡排序算法的时间复杂度会随着变量 n 的取值不同而不同。

```
void bubble_sort(int a[]){
   i=0;
   do{
          change=false;
          for( j=0; j<n-i-1; j++){
                 if( a[j]>a[j+1] ) {
                        temp=a[j];
                        a[j]=a[j+1];
```

```
                    a[j+1]=temp;
                    change=true;
                }
            }
        i++;
    }while(i<n && change);
}
```

一般情况下，当讨论时间复杂度时均指最坏情况下的时间复杂度。且时间复杂度考虑的只是对于问题规模 n 的增长率，此时，只需计算出它关于 n 的增长率或阶即可。如上面所讲的冒泡排序算法时间复杂度为：$T(n)=O(n(n-1)/2)=O(n^2)$。

对于给定算法的实现即程序的时间复杂度的分析主要考虑可执行语句的情况，下面是对于程序进行时间复杂度分析的一些基本规则。

① 对于简单的输入、输出及赋值语句，执行时间为 $O(1)$。

② 对于顺序结构，需要执行一系列语句所用时间可采用渐进式大 O 的求和规则来进行计算。

③ 对于选择结构，它的时间复杂度主要体现在对条件判断后所执行的语句上。如下列判断语句：

```
if(c){
  s1;
}
else{
  s2;
}
```

判断语句所需要的执行时间为：$O(1)$。

执行语句所需要的执行时间为：$O(\max\{T(s1),T(s2)\})$。

④ 对于循环结构，它的时间复杂度主要体现在循环体语句及循环条件判断语句上，可以利用渐进式大 O 的乘积规则来进行计算。

⑤ 对于复杂的算法，可以将它分割成几个容易估算的部分，然后利用渐进式大 O 的求和规则与乘法规则计算整个算法的时间复杂度。

需要指出的是：顺序结构、选择结构和循环结构是算法的三种基本结构。在进行算法的时间复杂度分析时，重点是对这三种结构的语句进行分析。

1.3.2　空间复杂度

一般情况下，一个算法所占用的存储空间包括算法自身、算法的输入、算法的输出及实现算法的程序在运行时所占用空间的总和。

由于算法的输入和输出所占用的空间基本上是一个确定的值，它们不会随着算法的不同而不同。而算法自身所占用的空间与实现算法的语言和所使用的语句密切相关，例如程序越短，它所占用的空间就越少。一个算法在运行过程中所占用的空间，特别是算法临时开辟的存储空间单元则是由算法策略及该算法所处理的数据量决定的。因此，对于一个算法的空间复

杂度的衡量主要考虑的是算法在运行过程中所需要的存储空间的大小。

假设 $S(n)$是问题规模 n(n 为整数)的函数,可以定义算法的空间复杂度为 $O(f(n))$,记作:

$$S(n)=O(f(n))$$

与时间复杂度 $T(n)$一样,$S(n)$也被称为算法的空间复杂度。

如冒泡排序算法中,其空间复杂度为 $S(n)=O(n)$。

除非特别说明,空间复杂度也是以最坏情况来分析的。

习题

1. 简单描述算法的定义以及算法的特性。

2. 简述算法评估的方法与度量指标。

3. 算法的三种基本结构是什么?

4. 举例说明什么是算法的空间复杂度?

5. 试列举算法复杂度分别为 $O(n^2)$和 $O(n\log_2 n)$ 的算法。并解释由阶 $O(n^2)$改进为阶 $O(n\log_2 n)$的根本原因是什么?

6. 给出下面算法的时间复杂度

```
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];
   }
```

7. 给出下面算法的时间复杂度

```
int fact(int n){
if (n<1)
        return 1;
else
        return(n * fact(n-1));
}
```

8. 利用伪码语言描述:求两个正整数 m 和 n 的最大公约数的算法。

9. 利用伪码语言描述:将保存在数组中的若干元素从小到大进行排序的算法。

10. 利用伪码语言描述:将两个分别装有果汁和酒的杯子进行互换的算法。

11. 有三个牧师和三个野人过河,只有一条能装下两个人的船,在河的任何一方或者船上,如果野人的人数大于牧师的人数,那么牧师就会有被吃掉的危险。试用伪码语言描述出一种安全的渡河方案。

12. 利用伪码语言描述收割白菜问题:菜园四周种了 n 棵白菜,并按顺时针方向由 1 到 n 编号。收割时,从编号为 1 的白菜开始,按顺时针方向每隔两棵白菜收割一棵,直到全部收割完毕。最后按收割顺序输出白菜的编号。

第 2 章　线性数据结构与算法

由于算法是作用在数据上的，因此数据的组织形式即数据结构在很大程度上影响着算法的设计和实现。本章重点介绍几种常见的线性数据结构，包括栈和队列，同时给出若干常用的基于上述线性数据结构的算法实现。

2.1　线性表

线性表(Linear List)是最常用且比较简单的一种数据结构，其主要特点是数据元素有序地存放在一起。

2.1.1　线性表定义及特点

线性表是由有限个数据元素组成的有序集合，每个数据元素由一个或多个数据项组成。例如构成 26 个英文字母的字母表(A,B,C,…,Z)是一个线性表，表中每个元素由单个字符组成数据项。

对于非空的线性表而言，它具有如下 4 个特点：

① 表中有且仅有一个开始结点；

② 表中有且仅有一个终端结点；

③ 除了开始结点和终端结点外，其他每个元素前面均有且仅有一个称为直接前趋的数据元素，它的后面均有且仅有一个称为直接后继的数据元素；

④ 虽然不同线性表的数据元素可以是各种各样的，但是同一线性表中的数据元素必须具有相同的数据类型。

线性表上的操作主要有插入、删除和查找三种。

常见的线性表的存储结构是顺序表(Sequential List)和链表(Linked List)。

2.1.2　顺序表

在计算机内存中存放线性表的最简单和最常用的方式是用一组地址上连续的存储单元依次存储线性表中的每个元素，即将线性表中的数据元素按其逻辑顺序依次存放到一个连续的内存区域，使线性表中相邻的元素存放在地址相邻的物理存储单元中，这种存储结构称为线性表的顺序存储结构，或简称为顺序表。

假设线性表的每个元素占用 L 个存储单元，并以其所占的第一个单元的存储地址 $\mathrm{LOC}(a_1)$ 作为数据元素位置，则线性表中第 $i+1$ 个数据元素的存储位置 $\mathrm{LOC}(a_{i+1})$ 与第 i 个数据元素的存储位置 $\mathrm{LOC}(a_i)$ 之间满足如下关系：

$$\mathrm{LOC}(a_{i+1}) = \mathrm{LOC}(a_i) + L$$

线性表中任意一个数据元素 a_i 的存储位置与线性表中第一个数据元素 a_1 的存储位置之

间的关系可以表示为：

$$LOC(a_i) = LOC(a_1) + (i-1) \times L$$

由于每一个数据元素的存储位置都和线性表中第一个数据元素的存储位置相差一个与数据元素在线性表中的位序成正比的常数。因此，只要确定了存储线性表的起始位置，线性表中任一数据元素都可以随机存取，所以线性表的顺序存储结构是一种随机存取的存储结构。

目前，几乎所有的高级程序设计语言都把数组定义为一种最基本的数据类型，而且编译程序为数组所分配的空间通常是一片连续的存储区域。因此，利用高级程序设计语言所提供的数组类型来实现顺序表是很自然的事情了。

一个数组是由一组相同类型的数据元素的固定集合。它们在物理内存上连续存放。对数组元素的访问是通过数组的下标进行的。所谓下标就是指该元素相对于数组第一个元素的偏移量，通常是 0 与 $n-1$ 之间的一个整数。通过下标，对数组中任何一个位置的元素的访问时间都是一样的。

在 Java 语言中，线性表的顺序存储结构可以描述为：

```
class ArrayLinearList{
    int size;//数组中已有元素个数
    Object element[ ];//数据元素
    ……//各种方法的定义
}
```

对于顺序表可以执行的操作很多，下面主要介绍对于用数组描述的顺序表所进行的空间分配、元素插入、元素删除和元素查找操作。

(1) 空间分配

在 Java 语言中，数组属于复合数据类型，在使用它之前，必须为其分配空间。空间分配算法的描述如下。

```
//参数 initialSize 为用户所申请的空间大小
public ArrayLinearList(int initialSize){
    if (initialSize<1)//抛出异常
            throw new IllegalArgumentException("容量必须大于 1");
    element=new Object [initialSize];//分配空间
}
```

(2) 插入操作

顺序表的插入操作是指在具有 size 个元素的线性表的第 i 个元素前插入一个新的元素 x，使线性表的长度增加 1。

插入算法的基本思路是：当 $1 \leqslant i \leqslant$ size 时，将第 size 个到第 i 个元素依次后移，然后进行插入。如成功插入返回 true，否则返回 false。

插入算法的描述如下。

```
public boolean insert _ SqList(int i, Object x){
    int j;
    int n=element.length;//获取表的容量
```

```
    if((i<1) || (i>size)) {
            System.out.println("插入元素位置不对");
            return false;
     }
    if(n==size){
            System.out.println("顺序表已满,无法插入");
            return false;
    }
    for (j=size-1;j>=i;j--)
            element[j+1]=element [j];//元素依次后移
    element[i]=x;//插入元素
    size++;
    return true;
}
```

如果顺序表未满,而且插入位置正确,算法将从最后一个元素开始后移,直到第 i 个元素,然后进行插入。如果顺序表已满,该算法只是简单地退出。在实际应用中,算法应该能动态调整顺序表的容量,这样就能减少算法出现异常的概率。

动态调整顺序表容量的基本思路是:重新创建一个容量较大的数组,将原数组中已有内容复制到新数组中。

调整顺序表容量算法的描述如下。

```
//n 表示原数组 a 的长度,newLength 为数组的新容量值
public Object [] changeArrayLength(Object [] a,int newLength){
        int n=a.length;
        //确保新的容量值合理
        if (n>newLength)
                throw new IllegalArgumentException("新的容量值太小");
        //创建一个新数组,与原数组类型相同,但容量不同
        Object [] newArray=(Object [])
                Array.newInstance(a.getClass().getComponentType(), newLength);
        //将原数组 a 中从 0 开始的内容复制到新数组 newArray 中的
        //从 0 开始的位置,n 为要复制的数组元素的数量
        System.arrayCopy(a, 0, newArray, 0, n);
        return newArray;
}
```

(3) 删除操作

顺序表的删除操作是指在具有 size 个元素的线性表中,将第 i 个元素删除,使线性表的长度减 1。

删除算法的基本思路是:当 $1 \leqslant i \leqslant$ size 时,将第 i 个元素到第 size 个元素依次前移。如成功删除返回 true,否则返回 false。

删除算法的描述如下。