C语言名题精选百则

技巧篇

冼镜光　编著

本书特点：

- ◎ 本书体例别具一格
- ◎ 囊括100则经典C语言程序设计题
- ◎ 精辟文字全面解析C语言程序设计技巧
- ◎ 免费提供完整的源程序下载

机械工业出版社
China Machine Press

ISBN 7-111-16376-1/TP·4264

责任编辑：王 平 封面设计：李 瞳

本书收集了100则C语言程序设计题，共分9类。第一类比较简单，主要希望读者了解到本书的题目、解法与其他书籍之间的差异；第二至六类分别是关于数字、组合数学或离散数学、查找、排序、字符串等方面的题目；第七类列出了一些不太容易归类的题目，如Buffon丢针问题、Dijkstra的三色旗问题等；第八类则收录了一些有趣的、娱乐性的题目，如魔方阵等；第九类题目相对较难，且多数是程序设计的名题。

本书结构别具一格，问题与解答分开介绍。前半部分，把所有问题分类收纳在一起，说明问题的内容和分析；后半部分是问题的解答。另外，每个题目还包含习题、程序和参考文献，习题部分要求读者延续解答中未完成的工作，程序则为读者提供了一个很好的参考，而参考文献是读者进行进一步学习论证的资料来源。本书内容新颖，实用性强，可作为高等院校师生学习C语言很好的参考用书。

ISBN 7-111-16376-1

9 787111 163763 >

定价：44.00元

地址：北京市百万庄大街22号 邮政编码：100037
联系电话：（010）68326294 网址：http://www.cmpbook.com
E-mail:online@cmpbook.com

C语言名题精选百则

技巧篇

冼镜光　编著

机 械 工 业 出 版 社

本书收集了100则C语言程序设计题，共分9类。第一类比较简单，主要希望读者了解到本书的题目、解法与其他书籍之间的差异；第二至六类分别是关于数字、组合数学或离散数学、查找、排序、字符串等方面的题目；第七类列出了一些不太容易归类的题目，如Buffon丢针问题、Dijkstra的三色旗问题等；第八类则收录了一些有趣的、娱乐性的题目，如魔方阵等；第九类题目相对较难，且多数是程序设计的名题。

本书在组织方式方面别具一格，问题与解答分开介绍。前半部分，把所有问题按归类的方式收纳在一起，说明问题的内容和分析；后半部分是问题的解答。另外，每个题目还包含习题、程序和参考文献，习题部分要求读者延续解答中未完成的工作，而程序为读者提供了一个很好的参考。详细完整的源程序放于网站中方便读者下载学习使用，而参考文献是读者进行进一步学习论证的资料来源。

本书内容新颖，实用性强，可作为高等院校师生学习C语言的参考书，也是C语言爱好者的自学读物。

版权声明

本书为经台湾儒林图书公司独家授权发行的中文简化字版本。本书中文简体字版在中国大陆的专有出版权属机械工业出版社所有。在没有得到本书原版出版者和本书出版者书面许可时，任何单位和个人不得擅自摘抄、复制本书的本部分或全部（包括数据和出版物）以任何方式进行传播。本书原版版权属台湾儒林图书公司。版权所有，侵权必究。

本书版权登记号：01-2003-7483　图字：01-2003-7483

图书在版编目（CIP）数据

C语言名题精选百则技巧篇 / 冼镜光编著. -北京：机械工业出版社，2005.6
ISBN 7-111-16376-1
Ⅰ.C…　Ⅱ.冼…　Ⅲ.C语言-程序设计-习题　Ⅳ.TP312-44
中国版本图书馆CIP数据核字（2005）第024545号

机械工业出版社（北京市百万庄大街22号 邮政编码 100037）
责任编辑：王　平　　版式设计：李永梅
三河市宏达印刷有限公司印刷 • 新华书店北京发行所发行
2005年7月第1版第1次印刷
787mm×1092mm 1/16 • 32印张 • 786千字
0001-5000册
定价：44.00元

凡购本图书，如有缺页、倒页、脱页，由本社发行部调换
本社购书热线电话：（010）68326294
封面无防伪标均为盗版

序

一年多前，作者就有了编写一本 C 语言问题集的念头，其中几度动笔，但终究还是把手稿悉数毁掉，始终不能确定为什么一直不能满意自己的成果，直到某天突然发现问题的症结就在于“读者要一本什么样的问题集”之后才恍然大悟，正是因为想在书中放入太多不兼容的东西，才造成数次易稿的问题。

读者究竟要一本什么样的问题集？作者思索再三，也读过一些相关的著作，发现读者不能只有一本问题集。首先，目前市面上的问题集大致可以分成两种（不以使用的语言来分）：第一种比较简单，目的是用一些程序来说明某种语言的用法，因此定位于入门学习程序语言的读者；第二种的内容比较丰富，从语言的用法一直到最后一部分的数值分析，显然定位于想深入学习的读者。但作者认为，一本这样的书对于想深入学习的读者而言，可能有一半（讲解语言的部分）没有用，而对刚入门的初学者来说，同样也有一半（数值分析的部分）没有用，即对两者都是浪费。如果书中还有基于某一科目知识的题目，那么读者所面临的就不仅仅是开发程序的问题，还有了解这门知识时的困难。

如果一本问题集会出现上述的问题，那为何不把它分开来做呢？但分开之后又要从哪一部分开始呢？环顾目前已经出版的类似书籍，其最终的目的都不过是教您如何编程而已。当然，正确性是程序的第一要点，但并不是惟一的；一旦入门且能够编写正确的程序之后，效率（Efficiency）就是另一个必须追求的目标，但这一点却鲜有程序设计的书籍探讨。这里所讲的“效率”，并不是指如何安排程序中的语句来加快速度；这样做固然可能会让程序的运行速度加快，但却不是一个治本的方法。就好像在骑自行车时，把身体练得强壮些可以骑得更快，但就长远来看，恐怕就不如一辆小小的摩托车了；换句话说，固然把程序语言学得很精通，可以迅速编写一个程序，但是如果采用了一种很慢的方法，则不如一个采用很快的方法，但没有用极佳的语言功能来编写的程序。本书列举的这种例子很多，读下去自然就会看到。所以，本书就从这一步出发，通过各个题目的讲解，教读者如何设计一个有效率的程序。书中有很多题目是读者耳熟能详的，那么是否想到过更好的做法呢？

收集在本书中的题目，有许多是知名的问题，它们散见于各类书本、教科书与期刊、杂志，有一些是学校的考题，另一些源于个人的研究，还有一些则出自名家的专栏，当然更有一些是名著的片段。在取材时，把它限定于中等程度以上，而且不包含任何数据结构的论题，因此最深的数据结构就是堆栈（Stack）、队列（Queue）以及排序中用到的堆积（Heap），对于与树状结构（Tree）、图（Graph）等有关的题目，一概不用，而留到后续的书中处理。至于那些简单的题目，一般教科书与参考书已经有很多，所以也不采录，但若采用，那么在解法上必定有独特、值得讨论的地方，或者有一种更有效率的做法。

本书绝大多数的题目都取自其他书、期刊论文，因此凡是能查出题目的来源和作者的，都在该题的参考文献中说明，以表示对原作者的敬意。如果您觉得某些题目很好，希望在自己的著作中引用，也请您能注明出处。

虽然本书是一本问题集，但并不希望读者看了题目就马上去读解答，显然没有经过深思就直接看解答不会有很大的作用，所以，本书的做法是一分为二，前半部分是问题与问题的说明，后半部分才是问题的解答与程序。另外，书中所有程序均放于网站中以方便读者下载学习使用。正因为如此，书中许多程序的主程序都没有列出来，在编程时已经考虑到这一点，这些程序的主程序只是提供一些测试资料以及显示结果而已，所有重点都放到各个函数中。本书所提供的程序免费提供读者参考、抄录，可以随意修改来配合学习，更可以提供给他人使用，但是读者不应该以它牟利，这正是公共程序（Public Domain Software）的意义；换句话说，未经本书作者允许，不能采用本书中任何程序或片段从事营利的行为。

本书是作者的一个新尝试，希望可以为学习编程的人员提供更高一层的信息与技巧，也希望您能在阅读、使用本书之余提供更多的改进意见。如果大家觉得这样的题材有趣而且实用，那么浩翰的书海中还有更多素材可以帮助我们再接再厉出版其他论题的专集。

尽管本书最后的程序设计工作只用了三个多月时间，但是过去一年多在图书馆中寻幽访胜的工作使我受益匪浅，最后要感谢妻子在整理这些资料时所付出的精力，与伴我昼夜不分甚至颠倒日夜工作的心意。

冯镜光

前　言

从某个角度来看，计算机科学是为编程服务的（吸收并分析来自各行各业、科目和范畴的问题，并且可能的话，找出一个高效率的算法来解决问题），而有了算法之后，编程就不再是一件难事了。所以解决问题的算法才是编程的关键所在，一个程序只不过是在实现它所使用的算法而已。如果程序有了问题，也就是有了 Bug，可能是对所使用的语言的语法、语意不甚了解而出了差错，或者是程序逻辑有了漏洞，其后者就是算法可能有问题；当然还有第三种可能，就是从算法到实现它的程序之间出了问题，原因是对所使用的算法了解不够而产生了误解，也可能是对语言了解不够导致使用错误而无法完全实现算法的功能。总之，要编写一个程序，除了要彻底了解所使用语言的特性之外，还需想出一个解决问题的算法。

当然首先不要被“算法”这个名词吓住了，它也并不是高不可及。事实上，有不少科学家正在孜孜不倦地研究、设计各式各样的算法。作为一个程序员，即便是一个刚学习程序设计的人，在把程序语句写下来之前，必定会先构思一番。例如，解 $ax^2+bx+c=0$ 这道一元二次方程式的题目，他可能会想到先算 b^2-4ac 是否不小于 0，如果是，再用课本中的公式来解，这就是算法。

本书并不像其他“问题集”、“问题演练”的书一样，把针对程序语言的问题与设计算法的问题，甚至于数值分析的问题都集中在一起，而使读者觉得语言问题不值一提（如果是一个编程高手），或者觉得解方程式等问题高不可攀（如果根本不懂数值分析）。本书的做法是，针对语言的问题，读者可以去读程序语言的教材、参考书，在那些书中自然会有详细的说明；对于一些有特殊背景的问题，如数值分析等也不作介绍，于是书中的问题都可以说是独立的，所无法独立的就是对问题的解法。因此，对于这一类型的题目，将在书中各处多安插几个，让读者见到它们的机会大一些。总之，本书所着重讲述的不是程序语言的问题，而在于如何设计一个解决问题的算法。

但是，算法也有好坏之分，如果在一本入门书中作者不慎选用了一个差劲的算法，而使读者对它有了第一印象，以后再想扭转就很难了。有一个很好的例子，求组合系数 C(n,r)，它的定义是：

$$C(n,r)=\frac{n!}{r!(n-r)}$$

很多作者的做法是先算出 n!、r!与 $(n-r)!$，再把 $C(n,r)$ 求出来，这是个很自然的解法，但是作者却觉得这个方法不够美妙，而用：

$$C(n,r)=C(n-1,r)+C(n-1,r-1)$$

因此，程序就可以写成递归的形式，先以递归的方法算出 $C(n-1,r)$，再用递归的方法去算 $C(n-1,r-1)$，最后把结果相加。程序比较短小，看起来也很漂亮（寥寥数行）。但又有新的问题出现，在执行程序时付出的代价可就太大了。当 n 较大，而 r 接近于 n/2 时，用

这种方法会比用第一种方法慢很多。不仅如此，许多人认为第二种方法只用加法，而第一种方法还要用乘法，因此第二种方法不但“漂亮”而且“快速”，但事实上这是个“主观”而不正确的结论。因此，就回到“有效率”的算法的课题上。

什么样的算法才算是有效率的呢？简单地说，它要快，而且使用空间要小。这里的“快”不是指 CPU 时间。因为，CPU 时间是相对的，在一部 Cray 上，上面的第二种方法可能要比在 PC 上用第一种方法快许多倍，所以不能用 CPU 时间来度量。一般而言，科学家会根据算法的程序一共进行了多少个运算（+、–、*、/、比较等），或处理了多少个数据作为评估的标准。本书不深入探讨这个问题，但在需要的地方也会稍微进行介绍。本书的程序都会选用一个较快并且一般读者都能接受的方法，有时还会介绍一个比较慢的方法与快的方法之间的差异。一般而言，我们着重于用较快也就是效率高的算法来解决问题。

总之，本书放弃了那些专门的解说程序语言用法的简单题目，而倾向于程序设计的内涵，也就是如何设计一个算法。对于有多种方法可以使用的题目，采用效率高的那一种。正因为如此，本书将不仅仅是一本入门书，还需要读者对程序语言、程序设计具有一定的基本知识。本书结合 C 语言来展开介绍，但 C 语言并不是一个很适合用来做这些实例的语言，其独特且有些怪异但却很方便的语法，往往会使一个读程序的初学者无所适从，因此在适当的地方，都作了提示。

本书的安排方式

本书共收集了 100 个中等难度以上的程序设计题目，可以粗略地分成 9 类（也许分类方式并不很适当，但这是作者觉得比较合理的分法），每一类中的题目在观念、解法上存在一些类似的地方。

第一类主要希望读者能够了解到本书的题目、解法与其他“问题集”书籍之间的差异。这几个题比较容易，但是有些程序员就是不会活用题目所给的条件，经常设计出一些“正确”但“效率”差的程序或算法。这几个题具有很好的代表性，希望能够做到本书所要求的境界，而不致于写出没有效率的程序。

第二类则进入正题，这十几个问题的对象是数字。例如，如何填数值的字谜？如何求出 2～N 的质数？如何求 x 的 n（正整数）次方？如何计算 Fibonacci 数，$f_n = f_{n-1} + f_{n-2}$？如何求组合系数 C(n,r)？如何求 n!等。乍看之下，似乎都会了，确实每一个题目都是熟悉的，因为都是程序设计课中的论题，但是在重做这些题目之前，首先看看是否能达到本书的要求。

第三类则比较少见，属于组合数学（Combinatoric Mathematics）或离散数学（Discrete Mathematics）的题目，包含了产生一个集合的所有子集、Gray 码、产生 1～n 的所有排列、产生 n 个元素中任取 r 个元素的所有取法、产生 n 个元素的所有 k 个元素的子集、n 个元素集合的所有分割（Partition）方式，以及如何把一个正整数写成若干个正整数的和等。其中有些题目，如排列方式、组合方式已经很普遍了，但有一些则不是，相信这部分对学习组合（或离散）数学的朋友会有所帮助。

第四类的主题是查找（Search），重点是在一群数中找出满足条件的元素。这一部分基本上是二分查找法（Binary Search）的各种变形，有些题目则是推广，而且难度也有一些

提高，但解法十分完美。

第五类的主题是排序（Sorting）。为方便起见，把几个有名的方法都写成了程序供读者参考。详细介绍：如何加快插入式排序（Insertion Sort）；如何在快速法中处理值相等的元素，使得排序完成之后值相等的元素仍然维持原来的顺序；如何让快速排列法在排正整数时不必递归，也不用堆栈；如何找出中位数；堆积法（Heap Sort）与它的变形；合并排序法（Merge Sort）；桶子法（Bucket Sort）等基本技巧。此外，还有不少应用到排序的程序也收集在此。例如，若数组中元素大多数相同时如何处理等。排序在计算机科学中是一个相当重要的技巧，但一般书都只介绍如何排序，本书则希望通过这些应用实例，使读者知道排序的应用不止是排列成绩、订单这些简单数据而已。

第六类是字符串（String）方面的课题。对很多人来说，这可能是最困难的一部分，因为大多数的书在这一类型的问题上都介绍不多。在该部分，通过问题与解答，介绍了Knuth–Morris–Pratt 方法与化简的 Boyer–Morre 方法，这两种方法在寻找字符串 s 中是否有一个特定的字符串时有相当杰出的表现。此外，还有几个有关部分序列（Subsequence）的问题，例如找出两个字符串的最长共同序列、把一个字符串编排成另一个字符串等。

第七类是一些不太容易归类的题目，如 Buffon 丢针问题，Dijkstra 的三色旗问题、找零钱问题、背包问题、无限位数的整数运算、最短路径问题等。这些可以算是杂项，相当富有变化，既有很单纯的（比如文字到整数的转换），又有含相当技巧性的（比如向量的分类）等。

第八类收录了一些有趣的、娱乐性的题目，如魔方阵（Magic Square）、N 后问题、武士巡逻、环游世界、一笔画、河内塔、生命游戏等。这些题目并不是广义的、两人对局的游戏，因此说成娱乐性的题目还恰当些，但也不排除以后会添加对局的游戏题目的可能性。

最后一类对读者来说是一个挑战。与第一类正好相反，这些题目都比较难，大多数是程序设计的名题。例如，要求找出圆上的一组点有没有 4 个可以组成长方形、机器人走完一段固定路程之后回到原点一共做了多少次 360° 的旋转、一组数中连续若干个的和与积的最大值、如何找出名人、如何找出投标结果中得票过半数的人、如何找出一对一函数，以及找出最长的递增部分序列等。

除了第一类的题目建议每道都做一次之外，其他的题目没有必要一题接一题地依次演练，读者可从中选择想做的题目来磨练一番。另外也需注意，分类的目的是方便把题目收集在一起，并不是指哪一类题目特别难、哪一类题目比较简单，而且同一类题目的难易程度也并不是愈在后面就愈难。作者认为，难易之分并没有绝对的标准，往往因人而异，也因人的学习背景而异。因此，如果您一下就把题目破解了，做得至少与本书的解答相当，那就恭喜了；但如果一时做不出结果，也不要灰心，多思考、多转几个弯、多变换几个招式，或许就会得到好的结果。这些思路历程就构成您以后解题、就业、工作的经验，切记千万不要看了题目就去看解答。

题目的组织方式

本书虽然也是一个问题集，但在组织方式方面却与其他同类书籍不太相同。虽然书中也是一个题目接一个题目讲解，但是问题与解答是分开的。在前半部分，把所有问题分类

收纳在一起，而问题的解答、说明与程序则是在下半部分。这种做法，就是希望能先消化了题目，经过相当程度的思考之后，觉得实在有必要或者自己也做出了结果，希望与本书的做法作个比较之后，才去翻阅解答的部分。

每一个题目都从问题的部分开始，在此说明该问题是什么，要做些什么事。接着就是问题的说明，因为做题目的人背景不同，也许问题部分过于简略而要作进一步的解说，这就是“说明”部分的工作。通常，在此会对问题作进一步的解释、分析，可能的话，还会指出适当的提示，好让读者有一个下手的地方。有些题目中，也会介绍一个大多数人都会想到的方法，分析此做法效率不高，而要求寻找更好的方法；换句话说，程序的正确性固然重要，但到了某个阶段之后，效率就是一个重要的课题，而要求的重点就是效率。

后半部分就是各个问题的解答，在解答中会把问题部分重复一次，但不会再讲说明的部分，而是立刻进入解答问题。绝大多数的解答都由解答、习题、参考文献、程序 4 部分构成，但是也有一些题目会视具体情况而有所增减。解答部分的目的，是告诉读者编写这个程序背后的想法、技巧、数据结构等要件，通常会很仔细地讲解要如何考虑这个问题，用什么数据结构会简化程序设计的工作。重要的是，绝大多数的题目到最后都会就所用的方法作一个效率上的分析，这些分析都是非正式的，在此也避免使用令人困扰的符号。另外，还可以帮助读者了解如何分析一个程序的效率进而改善自己的程序。

习题部分通常是要求延续解答中未完成的工作，例如证明一些在解答中不会做到但却有用的简单结果，就解答的效率作进一步分析或修改程序使其可以做更多的工作、多输出些有用的结果等，通常只要模仿或修改本书提供的程序就可以完成，但还是值得花点工夫去完成；另外还有极少数的题目，本身就是一个独立的具有挑战性的问题。一般而言，较难的习题书中都加上了提示，但是不用提示也应该可以解答，因为解法通常不止一个，所以提示部分仅供参考。

参考文献部分是本书的特点之一，在此可以知道这个题目是谁想出来的，解法是谁做的；如果这种方法并不是最好的，可以在什么地方找到更好的解法；也或者，有人在什么时候有过类似或其他的解法。因此通过这些参考文献，以此为起点，就可以发现更多的资料与另一个天地。

程序部分就是该题目的实际程序，它完全反映了解答部分的讨论。但需要注意，绝大多数的程序都已经写成函数的形式而由主程序调用，主程序方面只不过提供一些执行时的测试值而已，所以在这种情况下，为了节省篇幅，主程序并没有附在书上。除非可以完全不必改变而使用本书中的其他函数，所有的函数都会列出来供读者参阅。如果想知道主程序的细节，可以查看光盘中的内容。本书中的所有程序也都分类存储在光盘中，请看下面的说明。

本书中没有的内容

第一，强调本书并不讨论如何运用某一种程序语言，所以不可能在本书找到。第二，本书既然指明了“技巧篇”，所以问题就以发挥技巧为主，因此那些一眼就可以看出结果、比较没有变化的题目就排除在外了。第三，也是最重要的一点，本书的题目限定在数据结构以下，也就是说解这些题目可以不要数据结构的技巧，因而有比较广泛的读者群。事实

上，本书中所用到的技巧中最难的就是堆栈（Stack）、队列（Queue）而已，而且维数很低；倒是在第五类排序中的内容比较多，但对大多数人而言，这些都是耳熟能详的内容。至于在数据结构以上以及一些更为基本的题目，考虑在其他书中介绍。

本书下载文件

http://www.cmpbook.com/download/16376.rar

或 http://www.cmpbook.com/jk_xz.asp

目　录

序
前言

第一部分　问题

第1章　序曲……1
问题1.1　最长平台（PLATEAU.C）……1
问题1.2　支配值数目（GT_COUNT.C）……1
问题1.3　等值数目（EQ_COUNT.C）……2
问题1.4　两数组最短距离（MINDIST.C）2
问题1.5　等值首尾和（HEADTAIL.C）……3
第2章　数字问题……4
问题2.1　Armstrong数（ARMS1.C, ARMS2.C）……4
问题2.2　数字谜（TRENTE.C）……6
问题2.3　求质数（PRIME1.C）……7
问题2.4　筛法（SIEVE.C）……8
问题2.5　线性筛法（L_SIEVE.C）……9
问题2.6　因子分解（FACTOR.C）……10
问题2.7　数值自乘递归解（R_POWER.C）……10
问题2.8　数值自乘非递归解（I_POWER.C）……11
问题2.9　Fibonacci数非递归解（FIB_IT.C）……11
问题2.10　快速Fibonacci数算法（FIB_MT.C）……11
问题2.11　扩充Fibonacci数（EX_FIB.C）12
问题2.12　二项式系数加法解（CNR_ADD.C）……13
问题2.13　快速二项式系数算法（CNR_LOG.C）……15
问题2.14　快速阶乘运算（FACTLOG2.C）……15
问题2.15　更快的阶乘算法（FACTLOG.C）……17
问题2.16　连续整数的固定和（GIVENSUM.C）……17

第二部分　解答

第10章　序曲……98
问题1.1　最长平台（PLATEAU.C）……98
问题1.2　支配值数目（GT_COUNT.C）..99
问题1.3　等值数目（EQ_COUNT.C）……101
问题1.4　两数组最短距离（MINDIST.C）103
问题1.5　等值首尾和（HEADTAIL.C）.104
第11章　数字问题……106
问题2.1　Armstrong数（ARMS1.C,ARMS2.C）……106
问题2.2　数字谜（TRENTE.C）……109
问题2.3　求质数（PRIME1.C）……114
问题2.4　筛法（SIEVE.C）……118
问题2.5　线性筛法（L_SIEVE.C）……120
问题2.6　因子分解（FACTOR.C）……123
问题2.7　数值自乘递归解（R_POWER.C）……125
问题2.8　数值自乘非递归解（I_POWER.C）……127
问题2.9　Fibonacci数非递归解（FIB_IT.C）……129
问题2.10　快速Fibonacci数算法（FIB_MT.C）……130
问题2.11　扩充Fibonacci数（EX_FIB.C）133
问题2.12　二项式系数加法解（CNR_ADD.C）……136
问题2.13　快速二项式系数算法（CNR_LOG.C）……138
问题2.14　快速阶乘运算（FACTLOG2.C）……141
问题2.15　更快的阶乘算法（FACTLOG.C）……144
问题2.16　连续整数的固定和（GIVENSUM.C）……148

第 3 章　排列、组合与集合.....19
问题 3.1　列出所有子集（DIRECT.C）....19
问题 3.2　列出所有子集——字典顺序（LEXICAL.C）.....19
问题 3.3　产生 Gray 码（GRAYCODE.C）20
问题 3.4　产生所有排列——旋转法（PERMUT_R.C）.....21
问题 3.5　产生所有排列——字典顺序（PERMU_LR.C）.....22
问题 3.6　所有 K 个元素的子集(KSUBSET.C).....23
问题 3.7　集合的所有分割方式（SETPART.C）.....24
问题 3.8　整数的所有不同分割数目（INTPART#.C）.....25
问题 3.9　整数的分割方式（INTPART.C）26
第 4 章　查找.....27
问题 4.1　寻找脚码（ISEARCH.C）.....27
问题 4.2　寻找固定的和（FIXSUM.C）...27
问题 4.3　无限式查找（INF_SRCH.C）....28
问题 4.4　寻找极小值（CYCLEMIN.C）..28
问题 4.5　两个数组的中位数（MEDIAN2.C）.....29
问题 4.6　寻找中间值（B_SEARCH.C）..30
问题 4.7　3 个数组的共同元素（SEARCH3.C）.....30
问题 4.8　寻找最小与次小元素(1ST&2ND.C).....31
问题 4.9　查找矩阵（M_SEARCH.C）.....31
问题 4.10　表示成两个数平方和（TWOSQUAR.C）.....32
问题 4.11　最大方块区域（MAXSQR.C,MAXSQR2.C）.....32
第 5 章　排序.....34
问题 5.1　二分插入法（BINSERT.C）.....34
问题 5.2　Shell 法（SHELL.C）.....36
问题 5.3　快速排列法（QSORT.C）.....37
问题 5.4　保持等值的原来顺序(QSORT_L.C).....37
问题 5.5　非递归、无堆栈快速排列法（QSORT_I.C）.....38
问题 5.6　求中位数（MEDIAN1.C）.....39

第 12 章　排列、组合与集合.....152
问题 3.1　列出所有子集（DIRECT.C）..152
问题 3.2　列出所有子集——字典顺序（LEXICAL.C）.....154
问题 3.3　产生 Gray 码（GRAYCODE.C）156
问题 3.4　产生所有排列——旋转法（PERMUT_R.C）.....161
问题 3.5　产生所有排列——字典顺序（PERMU_LR.C）.....164
问题 3.6　所有 K 个元素的子集(KSUBSET.C).....166
问题 3.7　集合的所有分割方式（SETPART.C）.....170
问题 3.8　整数的所有不同分割数目（INTPART#.C）.....173
问题 3.9　整数的分割方式（INTPART.C）176
第 13 章　查找.....180
问题 4.1　寻找脚码（ISEARCH.C）.....180
问题 4.2　寻找固定的和（FIXSUM.C）.181
问题 4.3　无限式查找（INF_SRCH.C）..183
问题 4.4　寻找极小值（CYCLEMIN.C）186
问题 4.5　两个数组的中位数（MEDIAN2.C）.....188
问题 4.6　寻找中间值（B_SEARCH.C）191
问题 4.7　3 个数组的共同元素（SEARCH3.C）.....194
问题 4.8　寻找最小与次小元素（IST&2ND.C）.....196
问题 4.9　查找矩阵（M_SEARCH.C）...198
问题 4.10　表示成两个数平方和（TWOSQUAR.C）.....200
问题 4.11　最大方块区域（MAXSQR.C,MAXSQR2.C）.....202
第 14 章　排序.....209
问题 5.1　二分插入法（BINSERT.C）....209
问题 5.2　Shell 法（SHELL.C）.....212
问题 5.3　快速排列法（QSORT.C）.....215
问题 5.4　保持等值的原来顺序(QSORT_L.C).....219
问题 5.5　非递归、无堆栈快速排列法（QSORT_I.C）.....224
问题 5.6　求中位数（MEDIAN1.C）.....228

问题 5.7 堆积法（HEAPSORT.C）.....39
问题 5.8 改良的堆积法（HEAP_NEW.C）43
问题 5.9 合并法（M_SORT.C）.....44
问题 5.10 桶子法（BUCKET.C）.....44
问题 5.11 单一重复元素排序（LOT_DUP.C）
.....45
问题 5.12 均匀重复元素排序（LOG_DUP.C）
.....45
问题 5.13 堆积式合并（HEAPMERG.C）46
问题 5.14 检查数组元素是否相异
（UNIQUE.C）.....47
问题 5.15 数组中和为零的段落
（ZEROSUM.C）.....47
问题 5.16 平面上的极大点（MAXSET.C）
.....48
问题 5.17 宴会中访问数目的极大值
（MAXVISIT.C）.....48
问题 5.18 包含在其他区间中的区间
（CONTAIN.C）.....49

第 6 章 字符串.....51

问题 6.1 括号匹配问题（PARCOUNT.C）51
问题 6.2 转换成后继式写法（POLISH.C）
.....51
问题 6.3 计算前置式写法（PREFIX.C）.52
问题 6.4 Knuth-Morris-Pratt 法寻找字符串
（KMP.C）.....53
问题 6.5 Boyer-Moore 法寻找字符串（BM.C）
.....57
问题 6.6 所谓的 h-序列
（RH_SEQ.C,H_SEQ.C）.....58
问题 6.7 寻找部分序列（SUBSEQ.C）...59
问题 6.8 最长重复部分序列（MAX_REPS.C）
.....59
问题 6.9 最长共同部分序列（LCS.C）...60
问题 6.10 字符串编修（STREDIT.C）....61
问题 6.11 产生无连续重复部分的字符串
（DISTSEQ.C）.....62

第 7 章 其他问题.....63

问题 7.1 Buffon 丢针问题（BUFFON.C）63
问题 7.2 三色旗问题（FLAG.C）.....63
问题 7.3 字符串列整数的转换（X_ATOI.C）
.....64
问题 7.4 整数类型列的极值（LIMITS.C）65

问题 5.7 堆积法（HEAPSORT.C）.....231
问题 5.8 改良的堆积法（HEAP_NEW.C）235
问题 5.9 合并法（M_SORT.C）.....239
问题 5.10 桶子法（BUCKET.C）.....242
问题 5.11 单一重复元素排序（LOT_DUP.C）
.....250
问题 5.12 均匀重复元素排序（LOG_DUP.C）
.....252
问题 5.13 堆积式合并（HEAPMERG.C）259
问题 5.14 检查数组元素是否相异
（UNIQUE.C）.....266
问题 5.15 数组中和为零的段落
（ZEROSUM.C）.....267
问题 5.16 平面上的极大点（MAXSET.C）
.....269
问题 5.17 宴会中访问数目的极大值
（MAXVISIT.C）.....274
问题 5.18 包含在其他区间中的区间
（CONTAIN.C）.....278

第 15 章 字符串.....284

问题 6.1 括号匹配问题（PARCOUNT.C）284
问题 6.2 转换成后继式写法（POLISH.C）
.....286
问题 6.3 计算前置式写法（PREFIX.C）292
问题 6.4 Knuth-Morris-Pratt 法寻找字符串
（KMP.C）.....299
问题 6.5 Boyer-Moore 法寻找字符串（BM.C）
.....302
问题 6.6 所谓的 h-序列
（RH_SEQ.C,H_SEQ.C）.....306
问题 6.7 寻找部分序列（SUBSEQ.C）.309
问题 6.8 最长重复部分序列（MAX_REPS.C）
.....311
问题 6.9 最长共同部分序列（LCS.C）.315
问题 6.10 字符串编修（STREDIT.C）..320
问题 6.11 产生无连续重复部分的字符串
（DISTSEQ.C）.....325

第 16 章 其他问题.....332

问题 7.1 Buffon 丢针问题（BUFFON.C）332
问题 7.2 三色旗问题（FLAG.C）.....334
问题 7.3 字符串列整数的转换（X_ATOI.C）
.....338
问题 7.4 整数类型列的极值（LIMITS.C）340

问题 7.5　无限位数算术（ARITH.C）.....65
问题 7.6　线性表示的矩阵相乘（MATMUL.C）.....66
问题 7.7　对称表示的矩阵相乘（MATMUL_S.C）.....66
问题 7.8　找零钱问题（CHANGE.C）.....67
问题 7.9　背包问题（KNAPSACK.C）.....68
问题 7.10　最佳矩阵相乘顺序（PRODSEQ.C）.....68
问题 7.11　最短路径问题（SHORTEST.C）69
问题 7.12　产生匹配括号的字符串（PAR_GEN.C）.....70
问题 7.13　稳定伴侣问题（STABLE.C）..71
问题 7.14　单调矩阵的极值（MONO_MAX.C）.....72
问题 7.15　向量分类（CLASSIFY.C）.....73

第 8 章　游戏问题.....75

问题 8.1　奇数阶魔方阵（MAGIC_O.C）75
问题 8.2　单偶数阶魔方阵（MAGIC_SE.C）.....77
问题 8.3　双偶数阶魔方阵（MAGIC_DE.C）.....78
问题 8.4　N 后问题公式解（N_QUEEN1.C）.....80
问题 8.5　N 后问题递归解（N_QUEENR.C）.....81
问题 8.6　武士巡逻（KNIGHT.C）.....83
问题 8.7　环游世界（HAMILTON.C）.....84
问题 8.8　一笔画（EULER.C）.....85
问题 8.9　非递归河内之塔（HANOI_I.C）86
问题 8.10　生命游戏（LIFE.C）.....87

第 9 章　终曲.....89

问题 9.1　等量正负号段落（BALANCE.C）.....89
问题 9.2　寻找长方形（RECT.C）.....89
问题 9.3　多边形的直径（DIAMETER.C）90
问题 9.4　机器人旋转角度（TURNS.C）.91
问题 9.5　最大涵盖距离（MAXCOVER.C）.....92
问题 9.6　最大连续元素和（MAXSUM.C,MAXSUM1.C）.....93
问题 9.7　最大连续元素积（MAXPROD.C）.....94

问题 7.5　无限位数算术（ARITH.C）....343
问题 7.6　线性表示的矩阵相乘（MATMUL.C）.....349
问题 7.7　对称表示的矩阵相乘（MATMUL_S.C）.....352
问题 7.8　找零钱问题（CHANGE.C）....355
问题 7.9　背包问题（KNAPSACK.C）...357
问题 7.10　最佳矩阵相乘顺序（PRODSEQ.C）.....362
问题 7.11　最短路径问题（SHORTEST.C）367
问题 7.12　产生匹配括号的字符串（PAR_GEN.C）.....372
问题 7.13　稳定伴侣问题（STABLE.C）375
问题 7.14　单调矩阵的极值（MONO_MAX.C）.....379
问题 7.15　向量分类（CLASSIFY.C）....382

第 17 章　游戏问题.....388

问题 8.1　奇数阶魔方阵（MAGIC_O.C）388
问题 8.2　单偶数阶魔方阵（MAGIC_SE.C）.....391
问题 8.3　双偶数阶魔方阵（MAGIC_DE.C）.....395
问题 8.4　N 后问题公式解（N_QUEEN1.C）.....397
问题 8.5　N 后问题递归解（N_QUEENR.C）.....402
问题 8.6　武士巡逻（KNIGHT.C）.....408
问题 8.7　环游世界（HAMILTON.C）....415
问题 8.8　一笔画（EULER.C）.....421
问题 8.9　非递归河内之塔（HANOI_I.C）431
问题 8.10　生命游戏（LIFE.C）.....438

第 18 章　终曲.....445

问题 9.1　等量正负号段落（BALANCE.C）.....445
问题 9.2　寻找长方形（RECT.C）.....448
问题 9.3　多边形的直径（DIAMETER.C）450
问题 9.4　机器人旋转角度（TURNS.C）453
问题 9.5　最大涵盖距离（MAXCOVER.C）.....457
问题 9.6　最大连续元素和（MAXSUM.C,MAXSUM1.C）.....460
问题 9.7　最大连续元素积（MAXPROD.C）.....465

问题 9.8 寻找名人（SINK.C）.....94
问题 9.9 投票问题（VOTING.C）.....95
问题 9.10 寻找 1 对 1 函数（1TO1.C）...95
问题 9.11 寻找支配元素（DOMINATR.C）
.....96
问题 9.12 最长递增部分序列（LIS.C）...96

问题 9.8 寻找名人（SINK.C）.....467
问题 9.9 投票问题（VOTING.C）.....470
问题 9.10 寻找 1 对 1 函数（1TO1.C）.474
问题 9.11 寻找支配元素（DOMINATR.C）
.....477
问题 9.12 最长递增部分序列（LIS.C）.479

参考文献.....483

Download at http://www.pin5i.com/

第一部分　问　　题

第1章　序　　曲

问题 1.1　最长平台（PLATEAU.C）

已知一个已经从小到大排序的数组，这个数组中的一个平台（Plateau）就是连续的一串值相同的元素，并且这一串元素不能再延伸。例如，在 1,2,2,3,3,3,4,5,5,6 中 1,2.2,3.3.3,4,5.5,6 都是平台。试编写一个程序，接收一个数组，把这个数组中最长的平台找出来。在上面的例子中 3.3.3 就是该数组中最长的平台。

【说明】

这个程序十分简单，但是要编写好却不容易，因此在编写程序时应该考虑下面几点：

（1）使用的变量越少越好。

（2）能否只把数组的元素每一个都只查一次就得到结果？

（3）程序语句也要越少越好。

这个问题曾经困扰过 David Gries 这位知名的计算机科学家。本题与解答取自 David Gries 编写的有关程序设计的专著。

问题 1.2　支配值数目（GT_COUNT.C）

已知 f[ ]与 g[ ]两个整数数组，元素都已经从小到大排列，试编写程序算出 f[ ]中每一个元素比 g[ ]中元素大的个数的总数。换句话说，f[0]比 g[ ]中多少个元素大、f[1]比 g[ ]中多少个元素大等，这些值的总和就是所要求的答案。

例如，如果 f[ ]中有 1,3,5,7,9，而 g[ ]中有 2,3,4,7,8，比 g[0]大的有 f[1]～f[4]，比 g[1]大的有 f[2]～f[4]，比 g[2]大的有 f[2]～f[4]，比 g[3]大的是 f[4]，比 g[4]大的是 f[4]，因此答案是 4+3+3+1+1=12。

Download at http://www.pin5i.com/

【说明】

与问题 1.1 一样，需要特别注意数组 f[ ]与 g[ ]已经排序。如果问题 1.1 能够做出完美的解答，那么本题也不难，相似的方法就可以得到高效率的程序。

问题 1.3 等值数目（EQ_COUNT.C）

已知两个整数数组 f[ ]与 g[ ]，它们的元素都已经从小到大排列好，而且两个数组中的元素都各不相同。例如，f[ ]中有 1,3,4,7,9，而 g[ ]中有 3,5,7,8,10。试编写程序算出这两个数组之间有多少组相同的元素。

就上例而言，f[1]与 g[0]为 3 是第一组；f[3]与 g[2]为 7 是第二组。

【说明】

建议不要使用下面的方法来编程：

（1）固定 f[i]。

（2）对于 f[i]而言，检查 g[ ]中是否有与 f[i]相同的元素。

（3）处理下一个 f[i]，即 f[i+1]。

因为 f[ ]与 g[ ]都已经从小到大排列好，所以应该活用这一个很强的特性。一个好的程序员绝对不应用上面的笨拙方法来编写程序，这样会做太多无意义的事。

为什么呢？因为 g[ ]的元素都相异，对于 f[i]而言，最多只会找出一个与它相同的元素，最坏的情况是把 g[ ]全部查完才找出相同元素（如果采用上面的方法），如果 g[ ]中有 n 个元素，需要查 n 次；但是若 f[ ]中也有 n 个元素，那么需要把 g[ ]查 n 遍，一共作 n^2 次比较才能找出结果。

试着找出一种方法，把 f[ ]与 g[ ]各查一次就可以得到答案（记住，活用 f[ ]与 g[ ]已经从小到大排列的特性）。

做完这一题后，建议继续做下一题。

问题 1.4 两数组最短距离（MINDIST.C）

已知两个元素从小到大排列的数组 x[ ]与 y[ ]，请编写一个程序算出两个数组元素彼此之间差的绝对值中最小的一个数，此值称做数组的距离。

【说明】

如果 x[i]与 y[j]是两个元素，那么|x[i]-y[j]|就是这两个元素之间的距离，所有这些距离的极小值，称为数组的距离。比如说 x[ ]有 1,3,5,7,9，y[ ]有 2,6,8，那么最短距离就是 1，因为 x[0]与 y[0]、x[1]与 y[0]、x[2]与 y[1]、x[3]与 y[1]，还有 x[4]与 y[2]的距离都是 1。

Download at http://www.pin5i.com/

注意，如果 x[ ]与 y[ ]各有 m 与 n 个元素，那么元素之间的距离就一共有 m•n 个；事实上往往用不着这么多个值，应该活用 x[ ]与 y[ ]已经排列好的特性，不要把所有的距离都算出来。

问题 1.5 等值首尾和（HEADTAIL.C）

假设有一个数组 x[ ]，它有 n 个元素，每一个都大于零；称 x[0]+x[1]+…+x[i]为前置和（Prefix Sum），而 x[j]+x[j+1]+…+x[n-1]为后置和（Suffix Sum）。试编写一个程序，求出 x[ ]中有多少组相同的前置和与后置和。

【说明】

如果 x[ ]的元素是 3,6,2,1,4,5,2，于是 x[ ]的前置和有以下 7 个，即 3,9,11,12, 16,21,23；后置和则是 2,7,11,12,14,20,23；于是 11、12 与 23 这 3 对就是值相同的前置和与后置和，因为：

11=3+6+2（前置和）=2+5+4（后置和）

12=3+6+2+1（前置和）=2+5+4+1（后置和）

由于 23 是整个数组元素的和，因此前置和与后置和一定相同。

当然，也可以用上面的方法把前置和与后置和都先算出来（两者都是从小到大的递增序列，为什么？），再进行比较，但建议不要使用这种方法，因为它需要额外的内存。

Download at http://www.pin5i.com/

第2章 数字问题

问题 2.1 Armstrong 数（ARMS1.C,ARMS2.C）

在三位的正整数中，例如 abc，有一些可以满足 $a^3+b^3+c^3=abc$ 的条件，也就是说，各个位数的立方和正好是该数的本身，这些数称为 Armstrong 数。试编写一个程序来求出所有的三位 Armstrong 数。

【说明】

这是个常见的问题，许多课本、习题集都有。解决的基本方法就在于有没有办法从一个三位数中把它的 3 个位数取出（例如，把 379 分解成 3、7 与 9），或者在已知 3 个位数时把该三位数算出来（例如，已知百位为 5，十位为 1，个位为 9，则三位数为 519）。

另一个变形是给出一个数，证明它是否为一个 Armstrong 数。这两者类似，由自己产生的数可以永远控制在三位，但是后者却一定要去测试它是不是三位数。

如何测试一个数是否为三位数呢？有的书中介绍这样一种方法（见程序 UGLY.C），使用对数 log()以及指数 exp()函数，不仅如此，还要用四舍五入（在 C 语言中只好加上 0.5 再转型成整数）。为什么？原来只是在取出输入数据的某一个位数而已。一个简单的程序写得如此复杂，大多数初学者恐怕都很难看明白。在此给出一个提示，千万不要重蹈覆辙。

（1）如果要确认 x 是个五位数，就看 x/100000 是否为零。如果为零，那么 x 的位数不会超过五位。接着再检查 x 是否真的是五位，这就要看：

(x % 100000)/10000

是否为零，如果不为零，这个值就是 x 中从右向左数第五位数的值；如果是零，x 就最多只能有四位而已。为什么？因为 x%100000 是把 x 用 100000 去除的余数，值在 0～99999 之间，正是 x 的右边五位，再用 10000 去除，那就是第五位的值了；如果 x 是五位数，那么这个位数就一定大于 1，如程序 2-1 所示。

【程序】

程序 2-1（UGLY.C）

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

void main(void)
```

Download at http://www.pin5i.com/

```
{
    int    number;                /* the input number              */
    int    pos, ten_pow;          /* # of digits and 10^x          */
    int    arm_no;                /* computed Armstrong Number     */
    int    temp, i;               /* working variables             */
    float  expon;                 /* computed exponent             */
    char   line[100];             /* input buffer                  */

    printf("\nPlease key in number : ");/* get a number          */
    gets(line);
    number = atoi(line);

    pos = 0;                      /* count # of digits             */
    ten_pow = 1;                  /* power of 10, 10^0=1           */
    do {                          /* start counting                */
        ten_pow *= 10;            /* one more digit...             */
        pos++;
    } while (ten_pow < number);

    if (ten_pow == number)        /* all 10^x are not Arm #        */
       printf("\n%d is not an Armstrong Number.", number);
    else {                        /* not 10^x                      */
        ten_pow = number;         /* now ten_pow is a temp var     */
        arm_no  = 0;              /* initial value for Arm. #      */
        for (i = pos-1; i >= 1; i--) {      /* scan from R->L      */
            expon = exp(i*log(10.0));       /* the ugly part       */
            temp  = ten_pow / expon;        /* you do it...        */
            ten_pow %= (int)(expon + 0.5);
            arm_no  += exp(pos*log(temp)) + 0.5;
        }
        arm_no += exp(pos*log(ten_pow)) + 0.5;
        if (arm_no == number)
           printf("\n%d is an Armstrong Number.", number);
        else
           printf("\n%d is not an Armstrong Number.", number);
    }
}
```

（2）如果这两个条件都正确，则 x 是一个五位数。

（3）按照这个技巧，要找出 x 的第三位（自右往左算），那就是：

$$(x\%\ 1000)/100$$

（4）用下面的式子也可以求出 x 的第三位的值：

$$(x/100)\%10$$

Download at http://www.pin5i.com/

问题 2.2　数字谜（TRENTE.C）

在一些游戏与休闲的书刊中，经常会看到如下的数字谜：

```
      VINGT
       CINQ
+      CINQ
------------
     TRENTE
```

试编写一个程序进行破解（看下面的规则说明）。

【说明】

数字谜的规则是这样的：

（1）不同的英文字母表示不同的数字。因此，题目中最右边一行的 T、Q、E 就是不同的数。

（2）每一个数最左边一位不是 0。所以上面的谜题中，V、C、T 都不是 0。

（3）上面的题目是加，那就表示 VINGT 这个五位数与 CINQ 这个四位数的两倍相加后，会得到 TRENTE 这个六位数，并且数字的安排如谜语所示。

在解这些数字谜时，很多人都会不知不觉地用最笨拙的方法，因此可能执行好几分钟，甚至好几小时都得不到结果。就以上题为例，它一共有 9 个不同的字母，因此耗用 10 个数字符号中的 9 个，很多人就会写出下面的形式，如程序 2-2 所示。

【程序】

程序 2-2

```
for (V=1; V<=9; V++) /* V <> 0 */
     for (I=0; I<=9; I++)
          .....
          for (E=0; E<=9; E++) {
               VINGT = .....;
               CINQ  = .....;
               TRENTE    = .....;
               if (TRENTE == VINGT+CINQ+CINQ) {
                    .....
               }
          }
```

如果用这个程序，那么也许一两个小时也不一定能够算出结果，因为在最内层循环中的 if 一共要执行将近 10^9，近一亿次，所以这是最笨的方法。

通常，用一点小学算术知识就可以节省大量的计算时间。换言之，可以用四则运算来找出某个字母值的范围（如偶数、奇数、5～9、3 或 4 等，而不是 0～9），范围越小，程序

Download at http://www.pin5i.com/

的工作就越少，执行起来就越快。

不妨从这一点出发，就本题而给出一些提示：

（1）因为 3 个位数相加，最大就是 9+9+9=27，如果从这一个位数的右边一位会有进位，那么这个进位就最多是 2，所以 3 个位数再加上进位的最大值是 29，所以该位数和是 9，而进位是 2。

（2）现在看 V 那位，加上进位之后得到 TR，因为 V 最大是 9，进位最大是 2，所以 TR 的最大值是 11。因为 T 是结果的第一位，所以不能是 0，因此只好是 1；但结果前两位是 TR，TR 最大是 11，现在 T 是 1，于是 R 只能是 0。注意，如果 R 是 2～9 中任意一个，那就表示没有从 R 到 T 的进位，因而 T 就是 0 了。所以从一点看来，两个值就固定了，即 T=1，R=0。

（3）再来看 V，V 一定大于 8。为什么？千位数的进位最大是 2，要一个数加上 2 而有两位数，那么这个数不就至少是 8 吗？

（4）看个位数的 E。它是由 T 加上两个 Q 得来的，已知 T 是 1，而 2Q 一定是偶数，所以 E 是个奇数。

就用这 4 点提示，T 与 R 的循环就不必了；E 的循环有 3、5、7、9（注意，T 是 1，E 就不能用 1）；V 的循环只能用 8 与 9。还没有决定的就剩下 I、N、G、C 与 Q5 个，所以循环就有 7 层，I、N、G 与 Q 为 0～9；C 为 2～9（因为 C 是第一位，不能是 0，也不能是 1，因为 T 是 1），E 是 3、5、7、9；V 是 8 与 9，所以最内层的 if 就执行了 $10^4\times7\times4\times2$ 次而已，于是用这个方法写成的程序所花的时间大约是前面所提到的：

$$\frac{10^4\times7\times4\times2}{10^9}=\frac{7\times4\times2}{10^5}=0.056\%$$

因此可以看到节省了多少时间。

思考一下，另外是否还有其他可以节省时间的方法呢？

问题 2.3　求质数（PRIME1.C）

试编写一个程序，找出前 N（如 200）个质数。如果没有进一步要求，这不是难题。但在此希望从所知的、使用除法的方法中，用最快的办法来编写程序。

【说明】

可能最先想到的办法，就是让某个变量 i 从 2 变到 N，然后检查它是不是质数，如果是就显示出来，如果不是，就检查下一个。这是正确的做法，但却没有注意到一个小细节，因而使程序运行速度变慢。当然，2 是质数，但所有 2 的倍数都不是质数，如果令 i 从 2 变到 N，不就很冤枉地测试了 4,6,8,10,…这些数？所以第一点提示是测试 2,3,5,7,…,N，即 2 与所有奇数就足够了。同理，3 是质数，但 6,9,12,…这些 3 的倍数却不是，因此，如果能够把 2 与 3 的倍数都跳过去而不测试，任意连续的 6 个数中，就只会测试两个而已。以

Download at http://www.pin5i.com/

6n,6n+1,6n+2,6n+3,6n+4,6n+5 为例，6n,6n+2,6n+4 是偶数，又 6n+3 是 3 的倍数，所以如果把 2 与 3 的倍数都不理会，要测试的数就只留下 6n+1 与 6n+5 而已，因而工作量只是前述想法的 2/6=1/3，应该用这个方法去编写程序。

假设 i 是如上述的一个数（不是 2 与 3 的倍数），如何测试 i 是个质数呢？按照定义，i 如果是质数，也就只有 1 与 i 可以除尽自己，所以可以用 2,3,4,5,6,…,i-1 去除 i，如果都除不尽（余数不是 0），i 就是质数。观点也对，但却与上一点一样地笨拙。当 i>2 时，有哪一个数 i 可以被 i-1 除尽的？没有，为什么？如果 i 不是质数，那么 i=a•b，此地 a 与 b 既不是 i 又不是 1；正因为 a>1，a 至少是 2，因此 b 最多是 i/2 而已，去除 i 的数用不着是 2,3,4,…,i-1，而用 2,3,4,…,i/2 就可以了。不但如此，因为 i=a•b，a 与 b 不能大于 $\sqrt{i}$，为什么呢？如果 a>$\sqrt{i}$，b>$\sqrt{i}$，于是 a•b>$\sqrt{i}\cdot\sqrt{i}$=i，因此就都不能整除 i 了。如果 i 不是质数，它的因子最大就是 $\sqrt{i}$；换言之，用 2,3,4,5, …,$\sqrt{i}$ 去除 i 就行了。

但是，用 2,3,4,5,…,$\sqrt{i}$ 去除 i 也是个浪费。就像是前一段所说的，2 除不尽，2 的倍数也除不尽；同理，3 除不尽，3 的倍数也除不尽……最理想的方法就是用质数去除 i，因为在前一段的提示，i 不是 2 与 3 的倍数，所以就用 5,7,11,13,17,19,…这些比 $\sqrt{i}$ 小的质数去除 i 即可。

但问题是这些质数从何而来？这比较简单，可以准备一个数组 prime[ ]，用来存放找出来的质数，一开始它应该有 2、3 与 5。于是当 i=5,7,11,13,17,19,23,25,29,…这些不是 2 与 3 的倍数时，就用 prime[ ]中小于 $\sqrt{i}$ 的数去除 i 即可，如果都除不尽，i 就是质数，把它放入 prime[ ]中，因此 prime[ ]中的质数就会越来越多，直到满足所要的个数为止。

不妨用这段说明来编写程序，不过应注意下面几点：

（1）不要处理 2 与 3 的倍数（见第一段）。

（2）用质数去除（见第二、三段）。

（3）用 $\sqrt{i}$ 可能会有问题，为什么？有什么方法可以解决？

问题 2.4　筛法（SIEVE.C）

求质数是一个很普遍的问题，通常不外乎用数去除，到除不尽时，给定的数就是质数。但是早在 2000 年前人们就知道了一个不必用除法而找出 2～N 的质数的所有方法了。假设有一个很神奇的筛子，可以给出一个数，例如 i，这个筛子有办法把 i 的所有倍数去掉。请用这个方法求出 2～N 之间的所有质数。

这个方法称为 Eratosthenes（人名）筛法（Sieve Mothod）。

【说明】

下面通过一个例子来加强对筛法的印象，求出 2～40 之间的质数。首先，把 2～40 这些数一字排开：

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40

Download at http://www.pin5i.com/

2 是质数，所以把 2 的倍数都筛掉：

1 2 3 ~~4~~ 5 ~~6~~ 7 ~~8~~ 9 ~~10~~ 11 ~~12~~ 13 ~~14~~ 15 ~~16~~ 17 ~~18~~ 19 ~~20~~
21 ~~22~~ 23 ~~24~~ 25 ~~26~~ 27 ~~28~~ 29 ~~30~~ 31 ~~32~~ 33 ~~34~~ 35 ~~36~~ 37 ~~38~~ 39 ~~40~~

下一个数自然是质数 3，所以把 3 的倍数都筛掉：

1 2 3 ~~4~~ 5 ~~6~~ 7 ~~8~~ ~~9~~ ~~10~~ 11 ~~12~~ 13 ~~14~~ ~~15~~ ~~16~~ 17 ~~18~~ 19 ~~20~~
~~21~~ ~~22~~ 23 ~~24~~ 25 ~~26~~ ~~27~~ ~~28~~ 29 ~~30~~ 31 ~~32~~ ~~33~~ ~~34~~ 35 ~~36~~ 37 ~~38~~ ~~39~~ ~~40~~

下一个是 5，把 5 的倍数筛掉：

1 2 3 ~~4~~ 5 ~~6~~ 7 ~~8~~ ~~9~~ ~~10~~ 11 ~~12~~ 13 ~~14~~ ~~15~~ ~~16~~ 17 ~~18~~ 19 ~~20~~
~~21~~ ~~22~~ 23 ~~24~~ ~~25~~ ~~26~~ ~~27~~ ~~28~~ 29 ~~30~~ 31 ~~32~~ ~~33~~ ~~34~~ ~~35~~ ~~36~~ 37 ~~38~~ ~~39~~ ~~40~~

下一个是 7，把 7 的倍数筛掉（其实在此之前都已经筛掉了）：

1 2 3 ~~4~~ 5 ~~6~~ 7 ~~8~~ ~~9~~ ~~10~~ 11 ~~12~~ 13 ~~14~~ ~~15~~ ~~16~~ 17 ~~18~~ 19 ~~20~~
~~21~~ ~~22~~ 23 ~~24~~ ~~25~~ ~~26~~ ~~27~~ ~~28~~ 29 ~~30~~ 31 ~~32~~ ~~33~~ ~~34~~ ~~35~~ ~~36~~ 37 ~~38~~ ~~39~~ ~~40~~

再下来是 11，比 20/2 大了，所以工作停止，没有被筛掉的就是质数，它们是 2,3,5,7,11,13,17,19,23,29,31,37。

可以按照这一逻辑来编写程序，但是需注意下面几点：

（1）2 是质数，所以 2 的倍数是一定会被删除的，所以在一开始时根本没有把 2 的倍数放到筛子中的必要，这就省下了一半的空间。

（2）如果要求 2～N 之间的质数，做到 N/2 就可以停下来，因为大过 N/2 的数都不可能整除 N。

（3）程序不可以使用乘法与除法，只能用加或减，以求加快速度。

请基于这 3 项要求来编制程序。

问题 2.5 线性筛法（L_SIEVE.C）

在做筛法（Sieve）求质数的问题时，非质数的数据有很多是重复删除的。例如，如果有一个数是3×7×17×23，那么在删除 3 的倍数时会删到它，删除 7、17 与 23 的倍数时也都会去删它。看起来好像并没有多大关系，因为只是把一个值存入对应的位置而已，但是，当要求质数的范围很大时，此类有很多质因子的合数越来越多，因此程序的效率自然就会大打折扣。基于筛法的观念，编写一个程序，使得在删除非质数时“绝对”不做重复的工作。

【说明】

解这个问题的诀窍是如何安排删除的次序，使得每一个非质数都只被删除一次。

中学时学过一个因式分解定理，它说任何一个非质（合）数都可以分解成质数的连乘积。例如，$16=2^4$，$18=2\times3^2$，$691488=2^5\times3^2\times7^4$ 等。如果把因式分解中最小的质数写在最左边，有$16=2^4$，$18=2\times9$，$691488=2^5\times21609$；换句话说，把合数 n 写成$n=p^kq$，此时 q 当然是大于 p 的，因为 p 是因式分解中最小的质数。由于因式分解的惟一性，任何

Download at http://www.pin5i.com/

一个合数 n，写成 $n = p^k q$ 的方式也是惟一的。

由于 $q \geqslant p$ 的关系，因此在删除非质数时，如果已知 p 是质数，可以先删除 $p^2, p^3, p^4, \cdots$，接着找出比 p 大而没有被删除的数 q，然后删除 $pq, p^2q, p^3q, \cdots$，一直到 $pq > n$ 为止。试把这个概念编制成程序。

因为每一个非质数都只被删除一次，可想而知，这个程序的速度一定相当快。依据 Gries 与 Misra 的文章，线性的时间，也就是与 n 成正比的时间就足够了（此时要找出 2n 的质数）。

问题 2.6　因子分解（FACTOR.C）

编写一个程序，读入一个正整数，把它的所有质因子找出来。例如，如果输入的是 72，$72 = 2^3 \times 3^2$，于是质因子就有 2 与 3，如果输入的是 181944，$181944 = 2^3 \times 3^2 \times 7 \times 19^2$，所以质因子为 2、3、7 与 19。为了方便起见，$2^3 \times 3^2 \times 7 \times 19^2$ 可以用 2(3)3(2)7(1)19(2)作为输出形式，也就是说，如果分解开来有 a^b，输出时就是 a(b)。

【说明】

传统的做法是把输入值（假设是 n）用 2 去除，一直到除不尽为止。如果一共除了 i 次，就有 2^i 这一项，输出中就会出现 2(i)；接着再用 3 去除、5 去除、7 去除等，直到商数变成 1 为止。以 181944 为例，第一次用 2 除得到 93972，再除一次是 46896，第三次得到 23493，于是 2 就不能整除了。下来用 3 去除，第一次得到 7831，第二次是 2527，第三次就不能整除。对于 2527 而言，用 7 去除得到 361，再用 7 就除不尽了，其次的 11、13、15、17，也都除不尽；但 19 可以，再用 19 去除得 19；最后用 19 除，商为 1，不能再除了，因此就得到 $181944 = 2^3 \times 3^2 \times 7 \times 19^2$ 的结果。

试用这个概念来编写程序。

问题 2.7　数值自乘递归解（R_POWER.C）

如果 n 与 m 是正整数，那么 m^n 就是把 m 连乘 n 次，这是一个效率很低的方法，请写一个计算效率高的程序，并且分析程序中一共用了多少个乘法，应该以 n–1 个乘法作为设计准则。

【说明】

这是一个典型的递归设计题目，应该注意以下几点：

（1）试用分而治之（Divide.and.Conquere）的策略。

（2）注意到 x^4 可以用 x^2 自乘的关系，由此可以大量地降低乘法数目。

（3）连乘 n 次要 n–1 个乘法，能做到只要 $2\log_2 n$ 个乘法吗？

Download at http://www.pin5i.com/

问题 2.8 数值自乘非递归解（I_POWER.C）

继续求 m^n 问题（m 与 n 是正整数）。前面的提示会得到一个递归的程序，请编制一个运算效率同样高的非递归的程序。

【说明】

或许读者有自己独特的看法，但在此提供一个简单的建议，可以采用它来编写程序，当然也可以把它化简。建议是把指数 n 用二进制来看，比如若 n=13，那么 $13_{10}=1101_2=2^3+2^2+2^0$，所以求 m^n 时就相当于求 $m^{2^3+2^2+2^0}=m^{2^3}\times m^{2^2}\times m^{2^0}$；会发现二进制表示中对应那一位是 1，在 m^n 中就有那么一项。把这个观念编制成程序。

另外一个办法是可以把递归解法中每一个递归步骤的 n 提出来，看在什么时候用 $(m^k)^2$，什么时候用 $m(m^{2k})$，然后用非递归方式写出来。

了解了这些观点之后，编写这个程序就不难了。在编写完程序之后，还应该分析一下程序乘了多少次。

问题 2.9 Fibonacci 数非递归解（FIB_IT.C）

Fibonacci 数列 f_1,f_2,L,f_n 的定义是：

$$f_n=\begin{cases}1 & n=1\\ 1 & n=2\\ f_{n-1}+f_{n-2} & n>2\end{cases}$$

不用递归的方法，也不用数组，编写一个函数，它接收 n 的值，返回 f_n。

【说明】

用递归方法算 Fibonacci 数列效率是很低的，要计算很多个重复的加法，这个题目要求不用递归，不用数组，把 f_n 求出来。不过应注意下面的事项：

（1）递归方式并非全然不好，但不能直接套用公式。

（2）因为当 n>2 时，$f_n=f_{n-1}+f_{n-2}$，所以程序只保留 f_{n-1} 与 f_{n-2} 就可以算出 f_n。

问题 2.10 快速 Fibonacci 数算法（FIB_MT.C）

继续讨论 Fibonacci 数列问题。在非递归的 Fibonacci 程序中，在算 f_n 时最多不超过 n-2 个加法，编写一个速度更快的程序，或许可以用乘法。如果每一个乘法用 m 单位时间，每

Download at http://www.pin5i.com/

一个加法用 p 单位时间，于是非递归的写法因为最多有 n-2 个加法，因此最多用(n-2)p 时间。请问，改善的程序要用多少时间？假设只考虑加法与乘法而已。

【说明】

解决这个问题的技巧有不少，在此先提示一个很容易理解的方法。用矩阵来算，看下面的式子：

$$\begin{pmatrix} f_n \\ f_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \bullet \begin{pmatrix} f_{n-1} \\ f_{n-2} \end{pmatrix}, n>2$$

相信不难看出这个式子是对的，其实这只不过是把：

$$f_n = f_{n-1} + f_{n-2}$$
$$f_{n-1} = f_{n-1}$$

写成矩阵式而已。

将上式展开：

$$\begin{pmatrix} f_{n-1} \\ f_{n-2} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \left(\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} f_{n-2} \\ f_{n-3} \end{pmatrix} \right) = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^2 \begin{pmatrix} f_{n-2} \\ f_{n-3} \end{pmatrix}$$

一般而言，有：

$$\begin{pmatrix} f_n \\ f_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^i \begin{pmatrix} f_{n-i} \\ f_{n-i-1} \end{pmatrix}$$

继续展开会得到：

$$\begin{pmatrix} f_n \\ f_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{n-2} \begin{pmatrix} f_2 \\ f_1 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{n-2} \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

可以用这个观点来编写程序。

问题 2.11　扩充 Fibonacci 数（EX_FIB.C）

定义一组称为扩充的 Fibonacci 数如下：

已知 X 与 Y 两个数，于是扩充 Fibonacci 数 F_i 为

$$F_i = \begin{cases} 1 & i=0 \\ 1 & i=1 \\ X \bullet F_{i-2} + Y \bullet F_{i-1} & i>1 \end{cases}$$

因此，当 X=Y=1 时，这一组扩充的 Fibonacci 数就变成一般的 Fibonacci 数，现在请编写一个函数，接收一个 n 值，不用数组与递归的方法算出下面的结果：

$$F_0 \bullet F_1 + F_1 \bullet F_{n-1} + \cdots + F_i \bullet F_{n-1} + \cdots + F_{n-1} \bullet F_1 + F_n \bullet F_0$$

【说明】

与许多求某个式子的和的程序一样，也不能马上下手编写程序。因为数学家用的式子

Download at http://www.pin5i.com/

与编写程序的式子不同，数学家讲求式子要能表达出原意，而编程的式子则要求好算，不仅如此，还要运算得快。

如果用上面的式子，为了要用到F_0与F_n，可能要用一个数组把F_0,F_1,F_n算出来后保存起来，再两两相乘、相加。这种方法固然不错，但当 n 比较大时，内存的要求就大了，所以限定不能用数组。

因此，这里的提示是把式子换个面目，看有没有办法实现不用递归来算 Fibonacci 数的程序。

问题 2.12 二项式系数加法解（CNR_ADD.C）

编写一个程序，只用加法，求出 n 中取 r 个的组合系数C(n,r)；并且尽可能地使加法数目降低。

【说明】

有很多种办法可以编写这个程序，但是几乎所有的方法都与二项式系数C(n,r)有关，这个式子就是：

$$C(n,r)=C(n-1,r)+C(n-1,r-1)$$

于是马上就会有人提笔编写程序了，如程序 2-3 所示。

【程序】

程序 2-3

```
int cnr(int n, int r)
{
    if (n == r || r == 0)
        return 1;
    else
        return cnr(n-1,r) + cnr(n-1,r-1);
}
```

当然这个程序的确只用了加法，而且也是正确的，但这可能是所有这一类程序中最差劲的一个，为什么呢？

用$a_{n,r}$表示在算C(n,r)时所用的加法个数。很显然，从程序的 if 部分可以看出，当$n=r$以及$r=0$时没有用到加法，因此$a_{n,n}=a_{n,0}=0$；else 的部分则表示，要先算 C(n-1,r)与C(n-1,r-1)，一共用了$a_{n-1,r}$与$a_{n-1,r-1}$个加法，再把这两个结果相加，因此又多用了一个加法，综合起来就是用了$a_{n-1,r}+a_{n-1,r-1}+1$个加法：

$$a_{n,r}=\begin{cases}0 & n=r\text{或者}r=0\\ a_{n-1,r}+a_{n-1,r-1}+1 & \end{cases}$$

看$a_{n,r}$与$a_{n-1,r}+a_{n-1,r-1}$的差异，很容易就会联想到：

$$a_{n,r}=C(n,r)-1$$

Download at http://www.pin5i.com/

这个猜测是对的，可以用数学归纳法证明它。假设 r≤n/2，于是把 C(n,r)展开就可得到：

$$a_{n,r}=\frac{n\bullet(n-1)\cdots(n-r+1)}{r\bullet(r-1)\cdots3\bullet2\bullet1}-1$$
$$\approx n^{T}$$

如果 r>n/2，就在展开时把 r!去掉而剩下(n-r)!，结果是一样的；再者 C(n,r)=C(n,n-r)，所以用上面的方法来算 C(n,r)的加法次数大约与 n^r 成正比。

另外一种常见的做法是通过 Pascal 三角形来算：

```
n=0      1
n=1    1    1
n=2  1    2    1
n=3  1    3    3    1
```

因此从 n=2 起，将中间的 2 用其左上与右上两个元素相加；接下来就算 n=3 的所有资料，因此从 n=2 中的 1 与 2 算出第一个 3，再由 2 与 1 算出第二个 3，于是就可以一列接一列地把结果算出来，当算出所要的那一行的结果时，工作就可以停止了，这就是下面的片段，如程序 2-4 所示。

【程序】

程序 2-4

```
int cnr(int n, int r)
{
    int answer [MAXSIZE];
    int i, j;

    answer[0]=1;
    for (i=1; i<=n; i++)
        for (answer[i]=1, j=i-1; j>=1; j-)
            answer[j] += answer[j-1];
    return answer[r];
}
```

这个程序已经简单许多了，当 i=2 时有 1 个加法，i=3 时 2 个，i=3 时 3 个，i=n 时就要 n-1 个加法，所以总共就用了：

$$\sum_{i=2}^{n}(i-1)=\frac{n(n-1)}{2}\approx n^2$$

与 n^2 成正比的加法次数，虽然从 n^r 剧降到 n^2，但这仍然不够好，因为要求 C(n,r)，那为什么要计算比 r 大的部分呢？总而言之，n^2 加法还是太多了，事实上大约不超过 nr 个加法就已足够。

分析一下上面的程序 2-4，编写一个更快、毫无多余加法的程序。

提示：仔细看看 Pascal 三角形在算 C(n,r)时的路线，特别是递归程序经过的所在，必会有所收获，会发现程序将会出奇地简单。

Download at http://www.pin5i.com/

问题 2.13 快速二项式系数算法（CNR_LOG.C）

看下面的式子：

$$(2^p+1)^n = C(n,0)+C(n,1)2^p+C(n,2)(2^p)^2+\cdots+C(n,n)(2^p)^n$$
$$= \sum_{i=0}^{n} C(n,i)(2^p)^i$$

如果使用的语言或编译程序（如 Ada）有可以计算任意位整数加、减、乘、除的能力的话，能够从这个式子找出一个只需要与 $\log_2 n$ 成正比的乘法次数求所有 $C(n,i), i=0,1,\cdots,n$ 的办法吗？下面假设 C 语言具有此能力，来编写程序。

【说明】

假设给了一个 n，能否找出一种方法，用上式把所有 C(n,i),i=0,1,…,n 都算出来，这是主题。为了方便起见，不妨假设所有运算都可以处理任意位数的整数，而不会造成溢位；其实有一些语言，如 Ada，或是一些链接库都提供了这个能力，所以不必认为如此的假设不切实际。应用也还不少，包含有把 π 计算到一百万位，测定任何整数是不是质数以及做分解因式的程序，还有就是处理多项式；后者在计算几何学、机器人等应用上都有相当重要的地位。

提示：式子中有一个 p，它的值是多少？这是要事先决定的。此外，能在大约 $\log_2 n$ 个乘法的限制下求出所有 C(n,r)吗？此时 n 是输入，或许需要用到一些处理数元的技巧。

问题 2.14 快速阶乘运算（FACTLOG2.C）

在问题 2.13 中已经看过用与 $\log_2 n$ 个乘法成正比的技巧来算出 C(n,r)，请问有没有办法利用这个结果设计一个求 n!的程序，它只用与 $(\log_2 n)^2$ 成正比的乘法（所有有关硬件、语言的假设同问题 2.13）。

【说明】

要做到只用 $\log_2 n$ 个乘法，老手们马上就会想到分而治之，完全正确。但是，如果分而治之不在合适的地方分，用错了方法去治，常常就会得不偿失。下面是作者在某篇文章中看到的做法，如程序 2-5 所示。

【程序】

程序 2-5

```
unsigned long fact(int m, int n)
{
```

Download at http://www.pin5i.com/


```
        if (m == n)
            return m;
        else
            return fact(m, (m+n)/2)*fact((m+n)/2+1,n);
    }
```

请问这个程序快了没有？没有，不但如此，反而慢了。

事实上这个程序的想法是，把求n!的工作分成两半，如下式：

$$n!=1\bullet 2\bullet 3\cdots\left(\frac{n}{2}\right)\bullet\left(\frac{n}{2}+1\right)\cdots n$$
$$=\left[1\bullet 2\bullet 3\cdots\left(\frac{n}{2}\right)\right]\left[\left(\frac{n}{2}+1\right)\cdots n\right]$$

于是再递归调用去算$1\times 2\cdots(n/2)$，以及$(n/2+1)\cdots n$。那么它用了多少个乘法呢？用t_n表示在算n!时程序所要用到的乘法个数，因为它又分解成两半，为了方便起见，假设两半各有n/2个数，因此各需$t_{n/2}$个乘法，求出两半的结果后还要一个乘法来把两半的结果相乘才是正确的答案，所以在程序中就一共用了$2t_{n/2}+1$个乘法，亦即：

$$t_n=2t_{n/2}+1$$

如果把上式一步一步地展开，就会得到：

$$\begin{aligned}t_n&=2t_{n/2}+1\\&=2(2t_{n/2^2}+1)+1\\&=2^2t_{n/2^2}+1+2\\&=2^2(2t_{n/2^3}+1)+1+2\\&=2^3t_{n/2^3}+1+2+2^3\\&\vdots\\&=2^it_{n/2^i}+\sum_{j=0}^{i-1}2^j\end{aligned}$$

当i足够大，使$n/2^i$为1时，就会得到t_1，但根据上面的程序，只有一个数（也就是m与n相等）时，就直接返回m，所以没有用到乘法，亦即$t_1=0$。所以当$i=\log_2 n$时，$n/2^i=1$，上式变成：

$$\begin{aligned}t_n&=2^{\log_2 n}t_1+\sum_{j=0}^{\log_2 n-1}2^j\\&=\sum_{j=0}^{\log_2 n-1}2^j\\&=n-1\end{aligned}$$

注意上面倒数第二式，它是个等比级数，首项是$2^0=1$，公比是2，一共有$\log_2 n$项，所以和为n-1。

因此这个方法用了n-1个乘法，与传统的2,3,…,n一个接一个相乘所用的乘法数目一

Download at http://www.pin5i.com/

样多，不但如此，还用了递归，因此不论 n 是多少都肯定会比传统的连乘法慢！所以这个程序的尝试并不成功。

但是，想法却很接近，只要把上面显而易见的分而治之的方法动动脑筋、换个角度，马上就可以得到乘法个数与 $(\log_2 n)^2$ 成正比的程序。另外，我们知道计算所有 C(n,i) 也不过用了 $\log_2 n$ 个乘法，因此能不能用某个 C(n,i) 来帮忙算 n!呢？

问题 2.15 更快的阶乘算法（FACTLOG.C）

请仔细研究问题 2.14，编写出一个只用与 $\log_2 n$ 成正比的乘法个数，计算 n!的程序。

【说明】

问题 2.14 的解法是有进一步改良的余地的。用递归方法，通过下式计算：

$$\binom{n}{r}-\binom{n-1}{r}+\binom{n-1}{r-1}$$

一样，它也有一些被重复计算的地方，如果能够把它找出来，程序自然就可以做到乘法个数与 $\log_2 n$ 成正比的地步。仔细做一做问题 2.14 的习题（1）与习题（2），必会有所收获。

问题 2.16 连续整数的固定和（GIVENSUM.C）

编写一个程序，读入一个正整数，把所有那些连续的和为给定的正整数的正整数找出来。例如，如果输入 27，发现 2～7、8～10、13 与 14 的和是 27，这就是解答；如果输入的是 10000，应该有 18～142、297～1328、388～412、1998～2002 这 4 组。注意，不见得一定会有答案，譬如说 4、16 就无解；另外，排除只有一个数的情况，否则每一个输入就都至少有一个答案，就是它自己。

【说明】

任何人看到这个题目都会马上想到一个办法，把所有的和算出来，与输入比较。曾经看到过如下的一个解法，如程序 2-6 所示。

【程序】

程序 2-6（NOT_GOOD.C）

```
#include <stdio.h>
#include <stdlib.h>

void main(void)
```

Download at http://www.pin5i.com/


```
{
    char  line[100];
    long  sum, i, j, n;

    gets(line);
    n = atol(line);

    for (i = 1; i <= n/2; i++)
        for (sum = i, j = i + 1; j <= n/2+1; j++) {
            sum += j;
            if (sum == n)
                printf("\n%ld - %ld", i, j);
        }
}
```

它的做法是先固定一个 i、sum 变量，接着令 j 从 i+1 起变化，每次都把 j 的值加到 sum 中，因此 sum 中的值就是 i,i+1,…,j 这些连续整数的和。因此令 i 从 1 变到 n/2（n 是给定的值），而 j 从 i+1 变到 n/2+1，如果有一个和与 n 相同，就显示 i 与 j，表示 n 的值是 i 到 j 这一串连续的正整数的和。为什么 i 要从 1 到 n/2？很简单，如果 i 是 n/2，下一个数就是 n/2+1，于是这两个（连续的）数的和 n/2+(n/2+1)= n+1 就大于 n，所以 i 最多只能到 n/2；同理可以说明 j 不可以大过 n/2+1。

这个程序当然是对的，但运行太慢了！用 10000 作为输入，它一共执行了 311.71 秒，也就是 5 分钟多；但事实上，这个题目可以在不到一秒之内得出答案，而且当输入 1000000（100 万）时，也不过用 178.12 秒（3 分钟）左右而已，相比之下 NOT_COOD.C 的效率实在太低了。

问题出在什么地方？加法次数太多了。在上面的程序中，i 与 j 的关系永远满足 $1 \leqslant i \leqslant n$，$i+1 \leqslant j \leqslant n+1$，$i<j$；每一组 i 与 j 都会做一次加法，所以就一共做了大约 $n^2/8$ 个加法（这是个近似值）；当 n=10000 时，就大约是 1250 万个。

不过，一个好程序员应该研究是否有更好的方法存在，事实上就有一个，大约需要 2n 个加法就足够了，能想得出来吗？下面是几点有用的提示：

（1）如果在求和时是用 $i+(i+1)+\cdots+j$ 表示，那么 $i \leqslant n/2$；这是上面提过的。

（2）如果某个和 $i+(i+1)+\cdots+j$ 比给定的值大，那么把和变小，但仍然维持是一串连续整数的和时，拿掉 j 变成 $i+(i+1)+\cdots+(j-1)$，不如拿掉 i 变成 $(i+1)+\cdots+j$。为什么？因为 j 比 i 大，拿掉 j，和就下降太快了，不如拿掉 i，再慢慢降低（能用数学来证明吗？）

（3）如果和 $i+(i+1)+\cdots+j$ 比给定值小，加上 j+1 变成的 $i+\cdots+j+(j-1)$；道理同前。

有了这几点，编程应该不会是件难事了。

Download at http://www.pin5i.com/

第 3 章　排列、组合与集合

问题 3.1　列出所有子集（DIRECT.C）

编写一个程序，列出{1,2,3,…,n}这个集合的所有子集，包括空集合ϕ。

【说明】

列出一个集合的所有子集有很多做法，题目中并没有要求依某个特定的次序来排列，因此是不难做出来的。

因为集合中一共有 n 个元素，所以总共就会有 2^n 个子集；例如{1,2,3}有如下子集：

{}
{1}　　{2}　　{3}
{1,2}　　{1,3}　　{2,3}
{1,2,3}

问题 3.2　列出所有子集——字典顺序（LEXICAL.C）

编写一个程序，用字典顺序（Lexical Order）把一个集合的所有子集找出来。

【说明】

如果不知道何谓字典顺序，在此作一个简单的说明。假设给定的集合有 n 个元素，{1,2,3, … ,n}。又若 $\{a_1,a_2,\cdots a_i\}$ 与 $\{b_1,b_2,\cdots,b_j\}$ 是它的两个子集；如果 $a_1=b_1,a_2=b_2,\cdots,a_k=b_k$，但 $a_{k+1}<b_{i+1}$，$\{a_1,a_2,,\cdots,a_i\}$ 就排在 $\{b_1,b_2,\cdots,b_j\}$ 的前面。例如，{1,2,3,4}与{1,2,4}是两集合，它们前面的两个元素相同，但第三个不同，因此包含小的元素的集合就排在前面。请回想一下，这与字符串的比较有什么不一样呢？完全相同，惟一的差异，就是在集合中的元素要从小到大排好。

下面左边是 n=3，右边是 n=4 的结果，如表 3-1 所示。

表　3-1

n=3	n=4	n=3	n=4
{}	{}	{1,3}	{1,2,3,4}
{1}	{1}	{2}	{1,2,4}
{1,2}	{1,2}	{2,3}	{1,3}
{1,2,3}	{1,2,3}	{3}	{1,3,4}

Download at http://www.pin5i.com/

（续）

n=3	n=4	n=3	n=4
	{1,4}		{2,4}
	{2}		{3}
	{2,3}		{3,4}
	{2,3,4}		{4}

问题 3.3　产生 Gray 码（GRAYCODE.C）

编写一个程序，用 Gray 码（Gray Code）的顺序列出一个集合的所有子集。

【说明】

这个问题其实是在看有没有办法把 Gray（人名）码用程序编写出来，有了 Gray 码，找出对应的集合是件简单的事，问题 3.2 已经讲过了。

什么是 Gray 码？nbit 的 Gray 码是一连串共有 2^n 个元素的数列，每一个元素都有 nbit，而且任何相邻的两个元素之间只有 1bit 的值不同。例如，3 个 bit 的 Gray 码：

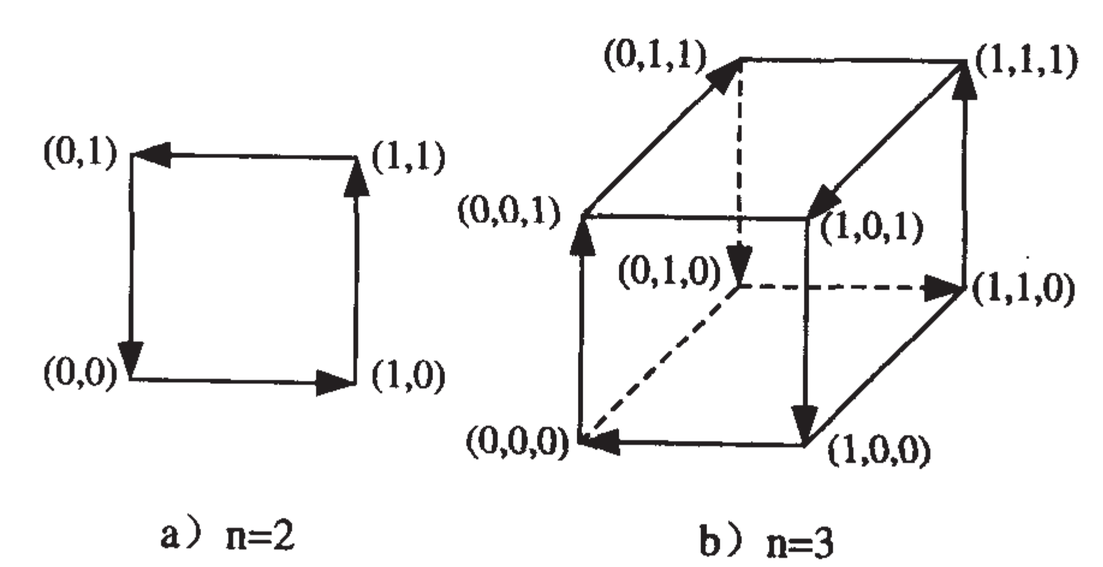
000　001　011　010　110　111　101　100

是一组 Gray 码，任何相邻两个元素都只有 1bit 值不同。但是，Gray 码却并不是惟一的，把它循环排列或是用反过来的顺序写，也会得到一组 Gray 码；比如说，如果把最后 3 个元素放到最前面去，就会得到：

111　101　100　000　001　011　010　110

也是一组 Gray 码。

Gray 码是一个很直观的几何意义，不妨把 nbit 看成是 n 度空间中一个点的坐标，因此有 2^n 个坐标点，正好是 n 维空间中的一个正立方体的 2^n 个角落。如图 3-1a 所示，当 n=2 时就是平方，是个正方形，如图 3-1b 所示，n=3 时就是个正立方体了。



a）n=2　　b）n=3

图　3-1

如果能够从原点出发把每个顶点都走一次再回到原点，且每一个顶点都不重复，那么沿途经过的点的坐标，就是一个 Gray 码。在图 3-1a 中，依箭头顺序，会得到 00,10,11,01；对图 3-1b 而言，则是 000,001,011, 010,110,111,101,100。当然，用不同的走法，就会有不同的 Gray 码，例如，在图 3-1b 中，000,100,101,001, 011,111,110,010 就是一个很好的例子。

Download at http://www.pin5i.com/

看看下面 n=4 的 Gray 码：

0000	0001	0011	0010
0110	0111	0101	0100
1100	1101	1111	1110
1010	1011	1001	1000

仔细看看哪一个位置是从 0 变到 1 的，它与左右两个邻居的关系如何，或许就有办法写成程序了。

或者，如图 3-1b 所示，沿途经过的方向有何变化，也可以编写出程序。这只是两点提示，方法不止有这两个。

问题 3.4 产生所有排列——旋转法（PERMUT_R.C）

编写一个程序，用旋转顺序（Rotation Order）列出 n 个元素的所有排列。

【说明】

旋转顺序并没有统一的定义，不过从下面 4 个元素的列法可以看出一点端倪。

1234	3124	2314
4123	4312	4231
3412	2431	1423
2341	1243	3142
[123]4	[312]4	[231]4
		[12]34

2134	3214	1324
4213	4321	4132
3421	1432	2413
1342	2143	3241
[213]4	[321]4	[132]4
		[21]34

[1]234

在上面的例子中，4 个元素经过上次旋转就会回到原来的样子，将前 3 个元素旋转一次（见上面加框的地方），得到一个新的组合，这个组合旋转 4 次，又回到原样，因而再把前 3 个元素旋转一次；新组合旋转 4 次又还原，于是前 3 个元素再旋转一次，不过 3 个元素经过 3 次旋转之后又会还原，所以这 3 个元素的前两个元素还要旋转一次，一直到两个元

Download at http://www.pin5i.com/

素的旋转也还原了，所有组合完全显现出来了。

因为 n 个元素旋转 n 次就还原，所以就旋转前 n-1 个元素，这就产生了 n(n-1)个排列；但是前 n-1 个元素经过 n-1 旋转之后又会还原，因此又得旋转前 n-2 个元素，于是就有了 n(n-1)(n-2)个排列。继续这样做，不就一共产生出 n!个不同的排列了吗？

注意，要检查有没有还原成原样不必大动手脚，因为开始时是 1234…n，所以只要 n 回到第 n 位，就表示这 n 个元素旋转了 n 次；同理，n-1 回到第 n-1 位，表示前 n-1 个元素旋转了 n-1 次后还原；依此类推，当 i 回到第 i 位时，就是前 i 位被旋转了 i 次后还原。但重要的是，这里是以大旋转来控制小旋转的，也就是说，n 次 n 个元素的旋转之后，接着是一次 n-1 个元素的旋转；n-1 次 n-1 个元素的旋转之后，接着是一次 n-2 个元素的旋转；……，i 次 i 个元素的旋转之后，是一次 i-1 个元素的旋转。

要如何把这个概念化成程序呢？请熟悉上面的例子，有可能的话，不妨试一试 5 个元素；所有的提示都讲过了，等全然明白以后，把程序编写出来。

问题 3.5　产生所有排列——字典顺序（PERMU_LR.C）

编写一个程序，用字典顺序列出 n 个元素的所有排列（Permutation）。

【说明】

下面是一个 n=4，用字典顺序列出来的所有排列，一共为 4!=24 个。

1234	2134	3124	4123
1243	2143	3142	4132
1324	2314	3214	4213
1342	2341	3241	4231
1423	2413	3412	4312
1432	2431	3421	4321

字典顺序的先后是这样定义的，如果 $a_1a_2\cdots a_n$ 与 $b_1b_2\cdots b_n$ 是 n 个元素的两个排列，于是有 $a_1=b_1, a_2=b_2, \cdots, a_i=b_i$，但 $a_{i+1}<b_{i+1}$，就是说 $a_1a_2\cdots a_n$ 在 $b_1b_2\cdots b_n$ 的前面，或者说前者较“小”；注意，自 i+2 个之后的各个元素是没有影响的。其实，这就是用来决定字符串大小的方式。举例而言，2314 与 2341，前两个元素相同，但第三个为 1<4，所以 2314 在前，2341 在后；同理，1234 与 4321 相比，1234 在前，4321 在后。

如何产生字典顺序的排列呢？据 Hall 与 Knuth 的考证，200 年（1812 年）前 Fischer 与 Kruse 在他们的一本书中就已经提过这样的方法了。从 1234…n 开始，首先从右到左找出第一对满足 $a_i<a_{i+1}$ 的 a_i 与 a_{i+1}，于是从 a_{i+1} 起在它右边的就是越来越小的；有了 a_i 之后，再一次地从右到左查一次，找出第一个满足 $a_i<a_j$ 的 a_j。接着把 a_i 到 a_j 的各个元素反过来

Download at http://www.pin5i.com/

排，就是字典顺序的下一个了。下面来看看如何找出 153642 的下一个：从右到左第一组满足 $a_i < a_{i+1}$ 的是 3 与 6，所以 a_i 就是 3。接着从右到左去找第一个 a_j，使得 $a_i < a_j$，则是 4(3<4=；最后把 3 与 4 之间的元素反过来排，因此得到 154632，这就是结果。

看另一个递归的做法，先看上面 4!=24 个排列的第一行，它们的第一个元素都是 1，第一行最末一个是以 1 起头，用字典顺序排出来的最后一个，自然是 1432。事实上，如果是 n 个元素的排列，应该会想到最后一个应该是 1n(n-1)…432。下一行是 2 开头，把 n(n-1)…432 中最小的一个与 1 互换，得到 2n(n-1)…431，但这绝对不是 1n(n-1)…432 的下一个，不过如果把后面的 n-1 个元素反过来排，就会得到 2134…(n-1)n，还是正确的顺序，于是就进入第二行。

第二行的最末应该是 2n(n-1)…431，但 1 已经在第 1 行出现过了，所以把倒数第二个元素（自然是 3）与 2 互换，得到 3n(n-1)…421，再把后面的 n-1 个元素反过来排，有 3124…(n-1)n，是不是第三行的第一个呢？

再做第三行，那是 3n(n-1)…421，因为 1 与 2 已经在第一个位置出现过，这一次把 4 换到头上去，得到 4n(n-1)…321，再反过来排，就是 4123…(n-1)n，正好是第四行。

于是就可以归纳出一个递归的做法了，从 1234…n 起，用一个递归的程序：

（1）i = n 。

（2）排到 n 的排列（递归的）。

（3）把第 i 位与第一位互换。

（4）把后面的 n-1 位反过来排。

（5）i 减去 1。

（6）回到第（2）步。

请把这个想法发展成一个递归的程序。

问题 3.6 所有 K 个元素的子集（KSUBSET.C）

编写一个程序，把一个 n 个元素的集合{1,2,3,…,n}中所有 k 个元素的子集用字典顺序列出来。

【说明】

还记得列出所有子集的程序吗？问题 3.2 已经介绍了一个把所有子集用字典顺序列出来的例子，事实上，只要把那个例子的程序稍作修饰就可以做出这个问题的答案。

如果 n=5，那么所有 3 个元素的子集用字典顺序来列就是

1	2	3
1	2	4
1	2	5

Download at http://www.pin5i.com/

1	3	4
1	3	5
1	4	5
2	3	4
2	3	5
2	4	5
3	4	5

注意在每一个子集变到下一个子集时究竟是哪一个位置有所变动；另外，特别重要的是，每一个位置都有该位置的最大值。例如，第三位可以到 5，第二位只能到 4（因为它到 5 的话，第三位就不可能放 3），而第一位（最左边）则是 3；这些最大值可以用 n（总元素个数）、k（子集的个数）以及每一个元素的位置算出来。

了解这些想法之后，程序就很好写了，不妨自己动动脑筋。

问题 3.7　集合的所有分割方式（SETPART.C）

一个集合 S 的分割（Partition），就是把 S 分解成若干个子集 $S_1, S_2, \cdots, S_p$，使得 $S_1 \cup S_2 \cup \cdots \cup S_p = S$，但对任意两个不同的子集 S_i 与 S_j 而言，$S_i \cap S_j = \phi$。换句话说，集合的分割，就是把该集合分解成若干相互分离的子集。举例而言，{1,2,3}的分割就有{1},{2},{3}；{1,2},{3}；{1,3},{2}；{2,3},{1}；{1,2,3}这 5 种。编写一个程序，读入 n，列出{1,2,3,…,n}的所有分割方式。

【说明】

一般而言，不会直接对{1,2,3,…,n}这个集合操作；聪明的做法是记录下哪一个元素在哪一个分割中，就以{1,2,3,4}为例，它的分割方式有下面的 15 种，如表 3-2 所示。

表 3-2

集　合	编　码	集　合	编　码
{1,2,3,4}	1111	{1,4},{2,3}	1221
{1,2,3},{4}	1112	{1},{2,3,4}	1222
{1,2,4},{3}	1121	{1},{2,3},{4}	1223
{1,2},{3,4}	1122	{1,4},{2},{3}	1231
{1,2},{3},{4}	1123	{1},{2,4},{3}	1232
{1,2,3},{2}	1211	{1},{2},{3,4}	1233
{1,3},{2,4}	1212	{1},{2},{3},{4}	1234
{1,3},{2},{4}	1213		

Download at http://www.pin5i.com/

在上面的写法中，右边的四位数表示 1,2,3,4 在哪一个分割中。例如，1213 表示 1 与 3 在第一个分割，2 在第二个，3 在第三个，所以分割就是{1,3},{2},{3}。再看 1212，它指出 1 与 3 在第一个分割，2 与 4 在第二个分割，因此分割本身就是{1,3},{2,4}。这个四位数就当作该分割的编码（Code），只要能找出编码本身有什么性质，分割就可以从编码重建出来。

这个编码有一个特色，在分割中从{1,2,3,4}起，总是在第一个部分保持最长的分割，而把余下来的元素放到第二个分割中，接着就按这种形式把可能的方式都列出来，再把大的分割切成较小的部分，一直到不能再切割为止。更重要的是，永远把 1 所在的分割定成第一个，所以四位数中 1 这个元素的编码值永远是 1。再看 2 这个元素，它的编码值永远不会大过 2，为什么会这样呢？如果 2 在第三个分割中，就会有：

$$\{1,\cdots\}\cdots\{2,\cdots\}\cdots$$

因此中间分割中的元素值都大于 3，但是可以把{2,…}这个部分换到前面而成为：

$$\{1,\cdots\}\{2,\cdots\}\cdots$$

因此 2 就在第二个分割中；这样做是正确的，因为分割本身并无次序性。推广开来，如果把分割中的元素从小到大排列，而且再把分割以第一个元素的顺序来排，那么 i 这个元素的编码一定小于或等于 i。

不但如此，还可以证明：第 i 个元素的编码是小于或等于 1,2,3,…,i−1 这几个元素的编码的极大值再加上 1。证明本身并不难，因为 i 在它所在的分割中，如果不是排在第一个（见 4 个元素时的{3,4}），就是排在第一个后面（见 4 个元素时的{2,3}，或{2,3,4}）。在前者，i 的编码如上述所讲的，最大就是 i，不过 i 所在的分割最多只会排在{1,…},{2,…},…,{i-1,…}后面，在这种情况下，1,2,…,i-1 的编码正好是 1,2,…,i-1，只要其中有两个合二为一，1,2,…,i-1 的编码值就会降低，所以 i 的编码值不能超过 1,2,…,i-1 的编码的极大值加上 1。如果 i 不是排在最前面，那么将 i 所在的分割中排在最前面的元素设为 j，就一定小于或等于 i-1。由上面的说明，i 的编码值是 1,2,…,j-1 的编码的极大值加上 1；更重要的是，在 j+1 到 i-1 之间的元素一定落在 1,2,…,j 所在的分割之一，但因为 i 与 j 在同一个分割中，所以 i 也就是等于 1 到 i-1 的编码极大值再加上 1 了。

用这个条件，就可以写一个很简单的程序了（建议是，把上面的说明彻底弄清楚之后再动手，不必忙着写程序）。

问题 3.8 整数的所有不同分割数目（INTPART#.C）

所谓的整数的分割（Partition of an Integer），指的就是把一个正整数写成若干个正整数的和，但这里只计较有多少种分解方式，而不计较它的内容。例如，4=3+1,2+2,2+1+1,1+1+1+1，再加上自己，就一共有 5 种分割方式。编写一个程序，输入一个正整数，输出它有多少种分割方式。

【说明】

以下是几点重要的提示：

Download at http://www.pin5i.com/

（1）要把 n 进行分割，其实不完全只针对 n，还要看分割中最大的值是什么。例如，要把 10 进行分割，若在分割中最大的值是 6，亦即 10=6+…，那么“…”的部分充其量的值是 4 而已，不仅如此，和还须等于 4；因此，如果知道“…”，亦即 4，有多少种分割方式，也正是在分割 10 时，最大值为 6 的分割方式！

（2）应该分析一下，n 这个输入值，与在分割中的极大值（如（1）中的 6）之间有什么联系，找出来，问题就解决了。

（3）不要忘了，递归是非常有效的武器。

问题 3.9　整数的分割方式（INTPART.C）

对于一个正整数 n 而言，它的一个分割（Partition），就是把 n 写成若干个正整数的和，但不计较书写的顺序。编写一个程序，输入 n，把 n 的所有分割显示出来。

【说明】

如果 n=7，那么有如下的分割。

7						
6	1					
5	2					
5	1	1				
4	3					
4	2	1				
4	1	1	1			
3	2	2				
3	2	1	1			
3	1	1	1	1		
2	1	1	1	1	1	
1	1	1	1	1	1	1

一共有 12 个，仔细观察在各个输出中前后两者的差异，并且自己做一做其他的结果（比如 n=5 时有 7 个，n=6 时有 11 个等），就不难写出程序了。

Download at http://www.pin5i.com/

第4章 查 找

问题 4.1 寻找脚码（ISEARCH.C）

已知一个整数数组 x[ ]，其中的元素彼此都不相同，而且也已经从小到大排列好。请用比较大小、相等的方式编写一个程序，找出给定的数组中是否有一个元素满足 x[i]=i 的关系。举例而言，如果 x[ ]={-2,-1,3,7,8}，x[3]=3，因此 3 就是答案。

【说明】

用笨方法，一个循环就可以找出答案，如程序 4-1 所示。

【程序】

程序 4-1

```
for (index = -1, i=0; i<n, i++)
    if (x[i] == i) {
        index = i;
        break;
    }
if (index >= 0)
    printf("\nFound at x[%d]", index);
else
    printf("\nNot Found");
```

这个程序在最坏的情况下，for 一共绕了 n 圈，作了 n 次比较，但却没有用到 x[ ]的元素已经排列好的条件。

事实上，如果输入有 n 个元素，应该可以写出与 $\log_2 n$ 次成正比的比较的程序，关键是 x[ ]的元素是排好顺序的。

问题 4.2 寻找固定的和（FIXSUM.C）

有两个数组 x[ ]与 y[ ]，各有 m 与 n 个元素，而且各个元素并没有依顺序排列；且 d 是一个已知的值。编写一个程序，看看在 x[ ]与 y[ ]中有没有满足 x[i]+y[j]=d 的元素。例如，若 x[ ]={3,7,2,4}，y[ ]={1,5,2,3}，d 为 9；那么 x[1]+y[2]与 x[3]+y[1]都合乎条件，亦即都是 9。

【说明】

当然，一个初学者很可能会编写出如程序 4-2 所示的程序。

Download at http://www.pin5i.com/

【程序】

程序 4-2

```
for (i = 0; i < m; i++)
    for (j = 0; j<n; j++)
        if (x[i] + y[j] == d)
            return
```

但是这不是一个好程序，因为在最坏的情况下（也就是根本不会有 x[i]+y[j]==d 的 i 与 j 时），一共要比较 m•n 次，因为每一个 x[i]与每一个 y[j]都相加，与 d 相比，当 m 与 n 很大时，这样写法的效率是很慢的。

因此，要求写一个次数少于 m•n 次的程序。

问题 4.3　无限式查找（INF_SRCH.C）

已知一个数组，元素个数有多少并不很清楚，但是数组元素已经依顺序从小到大排好，而且在数组最后添加了足够多的∞记号；∞表示最大的值，比数组中每一个元素都大，而且个数足够多。编写一个程序，在这个数组中找出某个给定的值。

【说明】

因为并不知道数组中有多少个元素，所以虽然数组中元素已经依顺序排列好，也不能套用二分查找法（Binary Search）。不过，二分查找法仍然是最有效率的、在已排好顺序的数组中查找数据的方法。请问，有没有办法克服有关不知道有多少元素的限制呢？

还有一点很重要，很可能不知道那个∞的值究竟是多少；惟一可以确定的就是∞比每一个数组中的值都大，不一定是该类型的最大值。例如，如果数组中的元素是 int 类型，它可能是 1,3,5,7,9,50,50,50,…，于是 50 在此就具有∞的作用。

提示：二分法还是可以用的，不过关键是要如何运用它，为了方便起见，假设∞正是该类型最大的值。

问题 4.4　寻找极小值（CYCLEMIN.C）

一个数组是以循环顺序排列的，也就是说在数组中有某个元素 i，从 x_i 开始有这样的关系，即 $x_0<x_1<x_2<\cdots<x_{i-1}$，$x_i<x_{i+1}<\cdots<x_n<x_0$。例如，8,10,14,15,2,6 这 7 个元素就是循环顺序排列的，因为从 2 开始为递增，到了最后一个元素就转换为第 1 个元素，再依顺序递增。换句话说，如果把 $x_i,x_{i+1},\cdots,x_n$ 取出，并且接到数组开头，于是就是一个从小到大的顺序（这不是个旋转的动作吗？）。编写一个程序，接收一个以循环顺序排列的数

Download at http://www.pin5i.com/

组，把它的极小值找出来，以上面数据为例，程序应该会输出 2。

【说明】

因为从 x_0 起顺序是递增的，一直到极小值出现，马上就会出现相反顺序，于是很多人马上就会想出这个做法：

```
for ( i = 1; i < n && x[i] >= x[i-1]; i++)
```

一旦这个循环停下来了，如果 i 等于 n 那就表示每一个元素都大于在它前面的那一个，因而极小值为 x[0]；但若 i<n，且 x[i]<x[i-1]，因此极小值为 x[i]。

这是个正确的做法，但效率却不够高，因为在最坏的情况下可能要做 n-1 次的比较。不过，这个数组严格说还是有顺序性的，根据这一特性应该可以找出更好、更快的方法，不妨试试看。

问题 4.5 两个数组的中位数（MEDIAN2.C）

已知两个数组 x[ ]与 y[ ]，各有 n 个元素，并且也已经以从小到大的顺序排好，编写一个程序，用少于 n 次比较找出 x[ ]与 y[ ]合并后的中位数。

【说明】

因为 x[ ]与 y[ ]都已经排好，而且各有 n 个元素，因此一般最自然的想法就是把 x[ ]与 y[ ]做合并的工作，一直合并到得出第 n 个元素时，这就是中位数了，这一共会用 n 次比较。注意，为方便起见，把中位数定成在中间的两个及其中之一，这是个方便的定义，因为 x[ ]与 y[ ]合起来有 2n 个元素，因此真正的中位数的定义应该是第 n 与第 n+1 个元素的平均数；所以为了方便，只要求找出第 n 个或第 n+1 个元素即可。

上面提到的，用合并的方法会用到与 n 成正比的比较次数，并不是所期望的做法，所希望的，是要比 n 少很多的做法。可能会想到的改善是：因为 x[ ]与 y[ ]是排好的，可以用 x[ ]的中间元素，例如 x[mid]，通过二分查找法找出它在 y[ ]数组中的位置，例如：

```
y[i] < x[mid] < y[i+1]
```

所以在合并起来之后，x[mid]前面的元素就有 mid+i+1 个，如果这个值大于 n，那么中位数就必定在这 mid+i+1 个元素内，因此 x[mid+1]～x[n-1],y[i+1]～y[n-1]都可以去掉，接着在 x[0]～x[mid],y[0]～y[i]之间找中位数就行了。但若 mid+i+1 比 n 小，在合并起来后中位数在元素多的那一半，所以把 x[0]～x[mid],y[0]～y[i]去掉，而在 x[mid+1]～x[n-1],y[i+1]～x[n-1]之间找中位数。反复这个过程，就不难找出中位数了。需要多少次比较呢？大约与 $(\log_2 n)^2$ 成正比，道理为何请读者自己证明。

事实上，问题还可以做得更好，而且也不必那么复杂，在与 $\log_2 n$ 成正比的比较次数之下就可以找出中位数的。

Download at http://www.pin5i.com/

问题 4.6 寻找中间值（B_SEARCH.C）

已知一个整数数组 x[ ]，一共有 n 个元素。对于任意两个相邻元素 x[i]与 x[i+1]而言，它们的差的绝对值都不超过 1，用数学式来写就是|x[i]-x[i+1]|≤1；而且还有 x[0]<x[n-1]。编写一个程序，在 x[ ]数组中找出它是否有某一个输入的值 a，x[0]≤a≤x[n-1]，程序一共用了多少次比较？能够做到少于 n 次吗？

【说明】

这个题目看起来有不少条件，其实它非常简单。有很多程序设计的题目原本都是很容易的，只不过常常看不清它的真面目，以致于做不出来或者是用了很笨拙的方法，这个题目就是一例。

提示没有什么必要，不过如果实在没有头绪，那就不妨用 I 作横坐标，x[i]作纵坐标，然后运用在微积分成数值分析中学过的东西。这是笔者的想法，不过有的朋友根本不这么想，而把它与每一本 C 语言教科书中都可能会有的例题产生联系，从书上马上取出那个例题，什么都不改就算是答案了。

那么把这道题放在此地有什么用处？也骗骗读者，唬一唬而已。不妨也可以拿它来唬人。记住，正因为它那么容易（但找不到窍门可就难啦！），还是要动动脑筋，不要看后面的解答。

问题 4.7 3 个数组的共同元素（SEARCH3.C）

有 3 个数组 x[ ]、y[ ]与 z[ ]，各有 x、y 与 z 个元素，而且三者都已经从小到大依序排列。编写一个程序，找出值最小的共同元素（也就是同时在 3 个数组中出现，并且值最小的元素）；但若没有共同元素，请显示合适的信息。

【说明】

通常寻找数据的工作都是给了一个数据，看看在某个数组中有没有它，但是这个问题却是另一个目的，要找出一个共同的元素。其实在实际应用中这种类型是屡见不鲜的。例如，警察知道犯人是台北市人，有大学学历，有窃盗前科，因此就可能从台北市得到这个台北人的档案，从教育部取得历年大学毕业生的档案，再从警察局取得有窃盗前科的人的档案，如果这些档案已经依人名从小到大排好了，那么找出嫌疑人的工作就与本题差不多，所不同的是：第一，只要找出第一个，而警察局可能要找出所有的；第二，用数组处理，所以得知该元素个数，而警察局收到的是档案，也许不知道各个档案有多少笔数据而已。

因为不知道要找的是什么，所以二分查找法没有多大用处，要另外想办法来解题，据我所知，最多用 3min(x,y,z)次比较就可以决定答案，更重要的是程序将不超过 10 列，所以要往深处思考。

Download at http://www.pin5i.com/

问题 4.8 寻找最小与次小元素（1ST&2ND.C）

编写一个程序，接收一个数组，找出该数组中最小与次小的元素的值。

【说明】

这是个十分简单的问题，通常会如程序 4-3 所示。

【程序】

程序 4-3

```
void first_and_second(int x[], int n, int*first, int*second)
{
    int i;

    if (x[0] <= x[1])
        *first = x[0], *second = x[1];
    else
        *first = x[1], *second = x[0];
    for (i = 2; i < n; i++)
        if (x[i] < *first)
            *second = *first, *first = x[i];
        else if (x[i] < *second)
            *second = x[i];
}
```

它很简洁而且容易懂，重要的是，它相当快。它一共做了多少次比较呢？开始时有一次，对于 i 从 2 至 n-1 而言，最坏的情况是两次，第一次是 if，第二次在 else if，因此就一共有 1+2[(n-1)-2+1]=2n-3 次比较。

不过，希望能从另一个角度来看这个问题，能不能用“分而治之”的策略，通过递归方法来写程序呢？程序进行了多少次比较？会比上面的程序快吗？仔细想想这个问题。注意，并不期望找出最快的方法，只是建议用“分而治之”（Divide.and.Conquer）的方法而已。

问题 4.9 查找矩阵（M_SEARCH.C）

已知一个 n 列 n 行的矩阵 M，它的元素满足一个很特殊的性质，即任一元素 $M_{i,j}$ 都小于在它右边与下方的元素（如果存在的话），换言之，$M_{i,j}<M_{i,j+1}$ 且 $M_{i,j}<M_{i+1,j}$。现在又有一个值 K，编写一个程序，检查矩阵 M 中是否有 K。

Download at http://www.pin5i.com/

【说明】

这个矩阵有了一种很特殊的结构，换言之，每一列与每一行都从小到大排列，所以在做寻找的工作时就可以通过这个特殊的顺序来编写程序。

注意，虽然有 n^2 个元素，但理论上可以证明，在最坏的情况下，就可以决定 K 在不在 ***M*** 中；关键所在，就是如何模仿二分查找法的技巧在一次比较之后就尽可能去掉多余不要的元素。

问题 4.10　表示成两个数平方和（TWOSQUAR.C）

已知一个正整数 R，编写一个程序找出所有满足 $X^2+Y^2=R$ 的正整数对 X 与 Y。

【说明】

例如，如果 R 是 10000，那么因为

$$\begin{aligned}10000&=28^2+96^2\\&=60^2+80^2\end{aligned}$$

所以就有 4 组 X,Y 满足条件，也就是 28 与 96、96 与 28、60 与 80、80 与 60。

要解这个题目很简单，因为 X 与 Y 的值都小于或等于 R 的平方根，所以用两个嵌套的循环，分别以 X 与 Y 作变量，每次比较 X^2+Y^2 是否与 R 相等，如果是，就把 X 与 Y 显示出来。至少这是一个解答，但这个做法却做了太多不应该做的事，换言之，它做了 R 次比较；因为 X 与 Y 是在 1 与 $\sqrt{R}$ 之间的任意整数，所以就应该有 $\sqrt{R}\cdot\sqrt{R}=R$ 组，每一组都作了一次比较。

不过，事实上使用与 $\sqrt{R}$ 成正比的比较就可以做到，换句话说，程序可以快 $\sqrt{R}$ 倍，试一试。

提示：仔细研究例题的解答。

问题 4.11　最大方块区域（MAXSQR.C,MAXSQR2.C）

在一个矩阵中，相连在一起的一块正方形区域就叫做子区域。例如，$M_{3,5}$、$M_{3,6}$、$M_{3,7}$、$M_{4,5}$、$M_{4,6}$、$M_{4,7}$、$M_{5,5}$、$M_{5,6}$ 与 $M_{5,7}$ 就是矩阵 ***M*** 中从 $M_{3,5}$ 起的一个 3×3 的子区域（Subblock）。编写一个程序，接收一个方阵（列数与行数相同），并且再接收一个已知的值，设为 K，找出在给定方阵中值全部是 K 的最大一个子区域，并且提示出这个子区域的位置与它的大小。

【说明】

这个题目并不十分容易，但也不难。如图 4-1 所示是一个 8×8 的方阵，元素值为 0 或 1，如果 K 是 1 的话，那么最大的子区域就有好几个，但都是 3×3（如程序 4-3 所示）。

Download at http://www.pin5i.com/

对这道题而言，最简单的做法就是用两个脚码 i 与 j，对于任何一个 $M_{i,j}$，就去查以它为左上角的区域，看看最大的值全为 K 的子区域有多大，当每一个元素都查完之后，就了解最大的子区域何在了。当然，在进行检查工作时，如果是从 $M_{i,j}$ 起，需要查 $M_{i,j}$、$M_{i+1,j}$、$M_{i,j+1}$、$M_{i+1,j+1}$ 是否相同，如果是，那么容量就至少是 2；接着再查 $M_{i+1,j}$、$M_{i+2,j+1}$、$M_{i+2,j+2}$、$M_{i+1,j+2}$ 与 $M_{i,j+2}$，可以用这个方法一直扩大，直到不能扩大为止。下面就是这样的一个示意图，如图 4-2 所示。

1	1	1	0	0	0	0	0
0	1	1	1	1	0	0	0
0	0	1	1	1	1	1	0
0	0	0	1	1	1	1	1
0	1	1	1	1	1	1	1
0	1	1	1	1	0	1	1
0	0	1	1	1	0	0	0
0	0	1	1	1	1	0	0

图 4-1

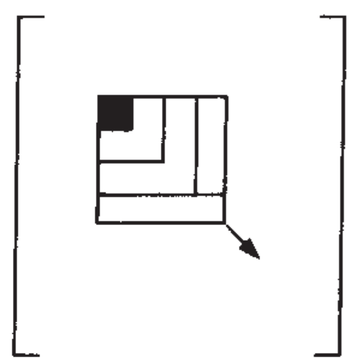


图 4-2

这是绝大多数程序员都会想到的办法，但是这个方法却要比较与 n^4 成正比的次数（n 是方阵的列数）。为什么？方阵有 n^2 个元素，每一个元素最多比较 n^2 次，所以是 $n^2 \cdot n^2 = n^4$ 次。当然，事实上不会有那么多次，会少一些，但总是与 n^4 成正比。

做这个题目的基本要求是打破比较 n^4 次的限制，例如 n^3 次，或进一步比较 n^2 次。与 n^2 成正比的比较次数是最低限度了，因为矩阵本身就有 n^2 个元素，至少每一个元素都会比一次。

Download at http://www.pin5i.com/

第5章 排 序

问题 5.1 二分插入法（BINSERT.C）

插入式排列法（Insertion Sort）是一个极为简单的方法，但它的效率却很低，因为比较次数与 n^2 成正比，n 是数据个数；而且还有大量移动数据的操作。编写一个程序，在这两点上作一些改进，使得比较次数与 $n\log_2 n$ 成正比，而且移动数据的速度加快。

【说明】

举如下例子来说明插入法的工作方式，按照从小到大顺序排列下面的一组数：

9 5 7 3 4

从第二个数起，把它之后的部分看成是未排列的部分，第一个元素是已排好的部分，然后依次把未排好部分的第一个元素取出，插到已排好部分的正确位置中，于是已排好部分的元素个数加 1，未排好部分元素个数减 1，表示成如下形式：

9 | 5 7 3 4
已排好　未排好

把未排好的第一个 5 取出，插到已排好的部分，得到如下形式：

5 9 | 7 3 4
已排好　未排好

下一步是把 7 插到已排好的部分，得到如下形式：

5 7 9 | 3 4
已排好　未排好

以此类推，插入 3 后，得到如下形式：

3 5 7 9 | 4
已排好　未排好

最后把 4 插到已排好的部分，于是已排好的部分就包括了所有元素。

3 4 5 7 9
已排好

在传统的一些教科书中一般这样来做插入工作的：如果要把 x[i] 从 x[0] 插入到 x[i-1]

Download at http://www.pin5i.com/

这一个已经排好的段落中，令 x[i] 与 x[i-1],x[i-2],…,x[1],x[0] 相比较，如果 x[j]≥x[i](0≤j≤i-1)，就把 x[j] 向后移一位，直到 x[j]<x[i]，x[i] 就放在 x[j+1] 处。这里，就以把 4 插到 3,5,7,9 中为例，开始先将 4 与 9 比较，9 比较大，所以 9 向右移一位：

x[i]=4　　3　5　7　<u>9</u>　9

上面划了底线的数据，表示是原来参与比较的数据所在地；接着 4 再与 7 比较，7 比较大，所以 7 向右移一位：

x[i]=4　　3　5　<u>7</u>　7　9

接着将 4 与 5 比较，5 还是比较大，因此：

x[i]=4　　3　<u>5</u>　5　7　9

最后将 4 与 3 比较，4 比 3 大，因此 4 就放在 3 的右边一位的位置上，所以最后结果是：

<u>3</u>　4　5　7　9

在最坏的情况下，每一个 x[i]都比较 i 次，也就是与每一个 x[i-1],x[i-2],…,x[1],x[0] 都比较过，然后才找到正确的位置，于是当 i=1,2,3,…,n-1 时，总比较次数为：

$$1+2+3+\cdots+(n-1)=\frac{n(n-1)}{2}$$

所以插入法在最坏的情况下，要有与 n^2 成正比的比较次数，也就是说，这是个 $O(n^2)$ 的程序，n 在此是数据个数。

在平均的情况下又如何呢？也就是让插入法去排一组均匀随机数，它会有多少次比较？假设正在处理 x[i]，打算把 x[i] 插到 x[0]～x[i-1] 之间，于是 x[i] 可能在的位置一共有 i+1 个；也就是说 x[0] 左边，x[0] 与 x[1] 之间，x[1] 与 x[2] 之间，…，x[i-2] 与 x[i-1] 之间，以及 x[i] 留在原地，因为数据是均匀的随机数，所以落在这 i+1 个空位中的可能性都相等，所以每一个位置的可能性都是 1/(i+1)。

将 x[i]与 x[i-1]比较一次，如果 x[i]>x[i-1] 就在原地；如果 x[i]<x[i-1]，但比较第二次时若有 x[i]>x[i-2]，x[i] 就在 x[i-1]。依此类推，x[i] 与 x[i-j] 的比较次数正好是 j 次；在最后一次与 x[0] 比较时，如果 x[i]>x[0] 就在 x[1]，但若 x[i]<x[0]，那就在 x[0] 了，因此落在 x[0] 的数据左边的比较次数是与落在 x[0] 与 x[1] 之间是相同的，都是 i 次。综合起来，平均比较次数为：

$$\begin{aligned}
&1\cdot\frac{1}{i+1}+2\cdot\frac{1}{i+1}+\cdots+i\cdot\frac{1}{i+1}+i\cdot\frac{1}{i+1}\\
&=\left(\sum_{j=1}^{i}\frac{j}{i+1}\right)+\frac{i}{i+1}\\
&=\frac{1}{i+1}\left(\sum_{j=1}^{i}j\right)+\frac{i}{i+1}\\
&=\frac{1}{i+1}\cdot\frac{i(i+1)}{2}+\frac{i}{i+1}\\
&=\frac{i}{2}+\frac{i}{i+1}
\end{aligned}$$

Download at http://www.pin5i.com/

因当 i=2,3,…,n-1 时都要做相同的工作，所以总平均次数为：

$$\sum_{i=1}^{n-1}\left[\frac{i}{2}+\frac{i}{i+1}\right]=\frac{1}{2}\sum_{i=1}^{n-1} i+\sum_{i=1}^{n-1}\frac{i}{i+1}$$

$$=\frac{1}{2}\frac{n(n-1)}{2}+\sum_{i=1}^{n-1}\frac{i}{i+1}$$

但是，通过微积分的知识：

$$\sum_{i=1}^{n-1}\frac{i}{i+1}\approx\int_{1}^{n-1}\frac{x}{x+1}dx$$

$$=\int_{2}^{n}(1-\frac{1}{y})dy \qquad (y=x+1)$$

$$=(n-2)-[\ln n-\ln 2]$$

所以总平均比较次数是：

$$\frac{n(n-1)}{4}+(n-1)-\ln n+\ln 2$$

其中最高次项是 $n^2/4$，所以平均比较次数还是与 n^2 成正比，也是 $O(n^2)$ 的。这个结论说明，插入法在最坏与平均的情况下都有与 n^2 成正比的比较次数，而这正是它速度慢、人们不爱用它的理由。

但是，插入法是可以加快的。第一，如何把数据插到已排好的一串数据中；第二，移动的动作也可以加快。改进之后，也许它仍然不能与快速排列法等相提并论，但快 3～4 倍总不是难事，知道要如何编写吗？

提示：留心一下二分查找法与 C 语言中的 <string.h> 标准链接库函数。

问题 5.2　Shell 法（SHELL.C）

在传统的教科书中，Shell 排列法都是直接引用 D.L.Shell 当年原著中的办法，在要排序的数组中，先把间隔为 n/2 的元素排好，然后把间隔为 $n/2^2$ 的元素排好，再排间隔为 $n/2^3, n/2^4, \cdots, 4, 2, 1$ 的元素，最后就是一个依顺序排列完成的结果。D.Knuth 曾经在他的著作中证明，Shell 方法要用与 $n^{1.5}$ 成正比的比较次数（当然已经比插入法等快了一些），不过最近 Sedgewick 等人却找出了更快的办法，证明可以把 Shell 方法加快到只用到 $n^{4/3}$ 成正比的比较次数。试编写一个程序，找出一组更快的方法。

【说明】

D.Knuth 很早就知道如何用 $(3^i-1)/2$ 的方式，亦即 1,4,13,…，这样的方法会比 Shell 原来的做法好一些。换言之，如果有 n 个元素，就得找出满足 $(3^i-1)/2<n$ 的最大一个 i 值，大约是 $[\log_2(2n-1)/\log_2 3]$。找到了 i 之后，就排间隔为 $(3^i-1)/2$ 的元素，再排间隔为 $(3^{i-1}-1)/2$ 的元素，…，最后排间隔为 13,4,1 的元素等。这种方法会比较快，有其直观的

Download at http://www.pin5i.com/

意义，因为若间隔为 $n/2, n/2^2, n/2^3, \cdots$ 时，就会有很多重复的比较。例如，，以 $n=8$ 为例，第一次会比较 x[0]与 x[8]，第二次间隔为 4 时会比较 x[0],x[4],x[8]，但 x[0] 与 x[8] 早就比较过，同理第三次间隔为 2，比较 x[0],x[2],x[4],x[6],x[8]，不就有更多重复的比较了吗？而这些重复了的比较数的结果在此之前就已经知道了，因而 Knuth 提出的间隔节省了时间。

近年来，Sedgewick 研究过快速排列法之后，再接再厉研究 Shell 排列法，也得到一些重要的成果，建议把间隔定为下式：

$$4^{j+1}+3\cdot 2^j+1$$

于是 Shell 方法的速度就可以降低到只用与 $n^{4/3}$ 成正比的比较次数而已，这又比 $n^{1.5}$ 进了一层，虽然 1.5 只比 1.33 多不了许多，但理论家却是相当兴奋的。

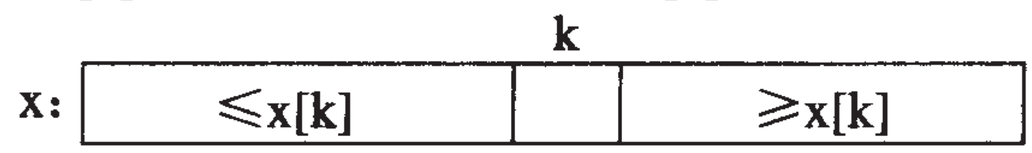
请用上述 Sedgewick 的方法来编写程序。

问题 5.3 快速排列法（QSORT.C）

读者可以找一本书，研究一下快速排列法（Quick Sort）的概念，并且写成程序。

【说明】

快速排列法是 C.A.R. Hoare 提出来的方法，它需用到递归的技巧。它的概念是这样的，有一个数组 x[ ]，它有 n 个元素。如果能够在 x[ ] 中找出一个元素 x[k]，使得在 x[k] 左边都比 x[k] 小或相等，在 x[k] 右边的则大于或等于 x[k]：



因此，只要把 x[1]～x[k-1]，x[k+1]～x[n]分别排列好，两者合并，就可以把整个 x[ ] 排好了。

所以为 x[1,…,n]排序就分成两个部分，第一部分是找出一个分界点 x[k]，满足上述的条件，第二部分是递归地去排 x[1,…,k-1] 与 x[k+1,…,n]，排好后结果就出来了。

所有快速排列法程序的差异都在第一步——如何找出分界点 x[k]。

问题 5.4 保持等值的原来顺序（QSORT_L.C）

快速排列法（Quick Sort）有一个缺点，就是相等的数据在排序完成后不会保持原来的顺序。不过，这个问题不难克服，使用连接串行的（Linked List）技巧就可以容易地克服它；编写一个程序完成这项工作。

【说明】

快速排列法虽然在一般情况下表现极为出色，却也有许多缺点，如果这些缺点不能一一克服，那么速度再快也是枉然，在这个问题中本书打算提出一个快速排列法不稳定

Download at http://www.pin5i.com/

（Unstable）的问题。所谓的一个稳定的排序方法，不但要能够把数据的大小顺序排好，而且还得维持在输入数据中值相等的各个数据的顺序。这一点看起来不重要，因为在书本上只排整数、实数或字符串而已，但在实际的应用中，却常常排一个文件的记录，这就可能会出问题了。就以电话簿为例，如果每一个记录都包含了姓名、电话号码、地址这 3 项数据，而希望只依姓名顺序排列，当然同名同姓的人不在少数，如果再按姓名排列时不能依输入顺序来处理同名同姓的人，那么在使用这个文件时就有可能出现问题。例如，本来 John Smith 在第 100 号记录，在查询时就希望他出现在那，但一经排列，这个数据很可能就脱离原有的位置，而当姓名栏相同时的顺序规则，使用起来不就是很不方便吗？

要维持值相同的数据的顺序并不十分难，这会在习题中讨论，不过却会引入很多额外的工作。不过，如果用串行来处理，那么这个问题就简单了。在程序中不妨假设输入是一个串行，在“分”的时候就把输入串行分成 3 个部分串行，分别连接那些小于、等于、大于某个值的元素，再递归调用自己去除小于、大于这两部分，排好之后，把小于、等于、大于这 3 个子串行再串回来，不就得到一个排列完成的串行了吗？如果在“分”的时候注意到加入“等于”那个串行中的顺序与输入的顺序相同，那么快速排列法就是一个稳定（Stable）的排序方法了。

问题 5.5　非递归、无堆栈快速排列法（QSORT_I.C）

许多人都批评快速排列法，但是观点却集中在快速排列法使用了递归的方法；事实上这一类的批评虽然并不完全是有的放矢，但是却不曾切中要害，换句话说，这并不是快速排列法的最严重缺点。无论如何，写一个不用递归的快速排列法却可以让不少人安心，请问能不能写一个不用递归，但也不用堆栈的快速排列法来排正整数呢？

【说明】

不用递归很简单，用数组作堆栈来仿真递归的动作即可。下面是几点有用的建议。首先，把问题集中在只排正整数上头，换句话说，输入数据全是正值，因此负的数就可以在数组中作记号用。

仔细回想在快速排列法中用 Lomuto 方法把原数组分解成两部分时的方法，一旦数组分解完了，中间的基准元素就明确了位置；换言之，它算是已经排好了。这是一个非常重要的方法，因此往后的工作中就不必处理这个元素。为了方便起见，把这个变量存回数组，往后凡是碰到负值的数，就表示是已经排好了的。

递归方法此时分成两次使用，一次做左半，一次做右半。不过本书不打算用递归，过程就要复杂些。读者不妨这样想，再加一个指标，指向被排好的元素，或是已排好部分的下一个元素，如图 5-1 所示。

因此，可以不断地分割，直到 x[0] 排好为止；然后向前移，再分割，直到 x[1] 排好，…，最后使 x[n-1] 也排好，工作就完成了。不过应注意一点，就算 x[0] 排好了，这不单是 x[0]

Download at http://www.pin5i.com/

被排好，往往在 x[1]～x[n-1]之间也有不少元素被排好了。

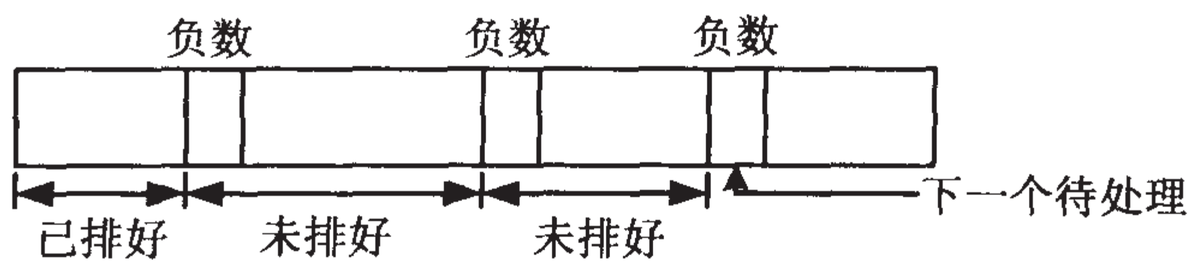


图 5-1

问题 5.6 求中位数（MEDIAN1.C）

中位数（Median）是将一组数从小到大排好后居中的数；如果这一组数的个数为奇数，那么中位数是存在的；但若个数为偶数，就没有中间的那一个数了，因此取居中两个数的平均数。例如，3,1,7,5,9 经过排列后是 1,3,5,7,9，所以中位数是 5；但 3,1,7,5,9,4，经过排列得到 1,3,4,5,7,9，所以中位数就是(4+5)/2=4。为了方便用整数计算，编写一个程序，接收一个整数数组，不必排大小而找出该数组的中位数。

【说明】

只要一排好大小，问题就简单了，所以规定不能排大小。求中位数的方法很多，有些方法中看不中用，也就是说理论上很快，但程序不太好写。不过，倒是有一个现成的方法可以用，虽然效率不是很高。这个题目的提示是，何不仔细研究一下快速排列法中叫做 split() 的程序与快速排列法的内涵呢？仔细想想就可以得到一个不错的方法。

问题 5.7 堆积法（HEAPSORT.C）

编写一个堆积排列法（Heap Sort）的程序。读者可以参考身边任何参考书，但注意一点，程序尽量编写得有效率。

【说明】

在树状结构中有一种形状特别的结构，如图 5-2 所示。

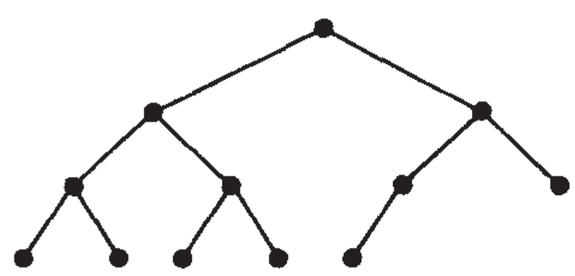


图 5-2

除了最低层与倒数第二层靠右边的节点之外，每一个节点都恰好有两个后代；而且重要的是，如果倒数第二层有某些节点不足一个后代，它们一定是在右边。如图 5-3 所示的 5

Download at http://www.pin5i.com/

个图中左边两个符合条件，右边 3 个则不符合条件。

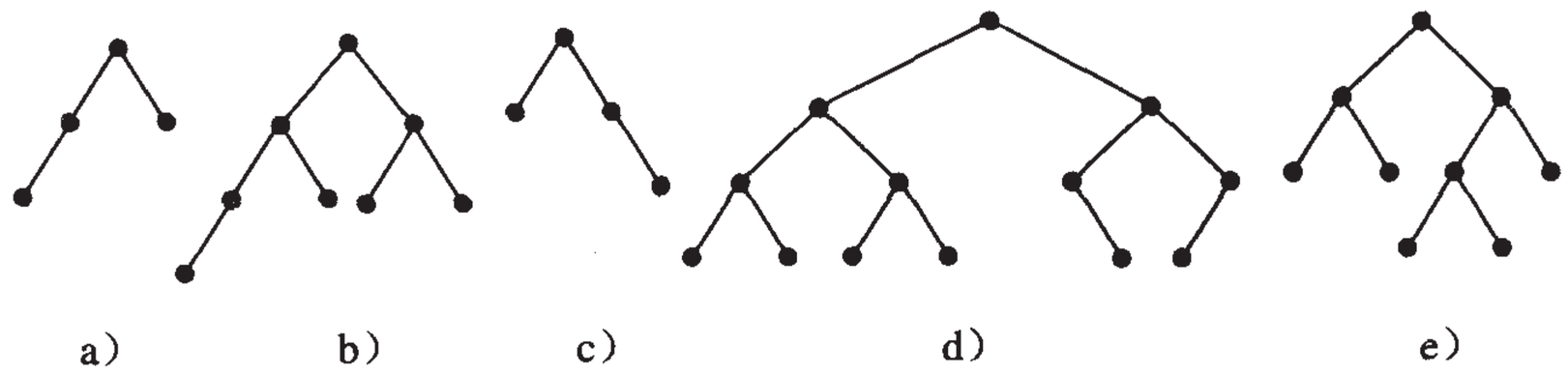


图 5-3

图 5-3c 是因为多出来的一个后代是在右边，而不是在左边；图 5-3d 是因为倒数第二层最右边的两个节点不满足条件；图 5-3e 则是倒数第二层最左边的两个没有后代，但若改成图 5-4 这样，就都是符合条件的了。

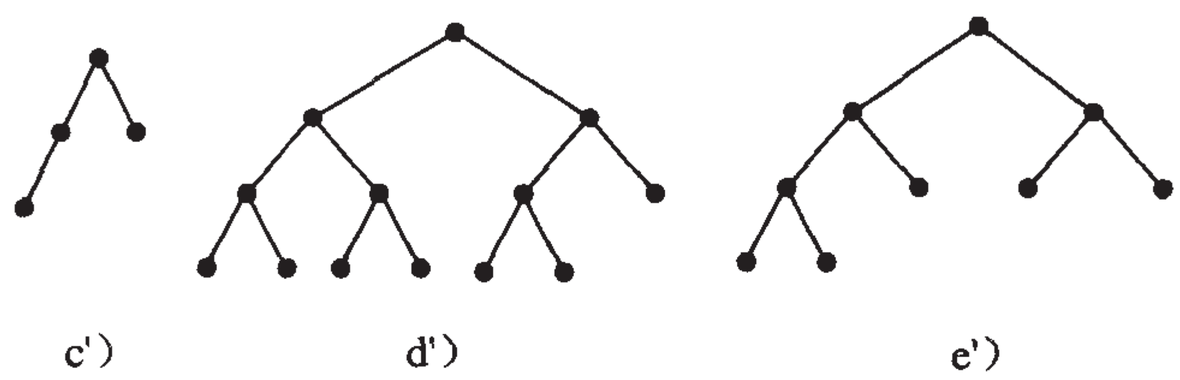


图 5-4

这种树状结构就叫做完全树状结构（Complete Tree）。

如果在完全树状结构中的每一个节点中都有一个值，而且这个值都比它的两个后代的值大（小也行，不过此处用“大”），那么这个完全树状结构就叫做一个堆积（Heap），如图 5-5 所示。

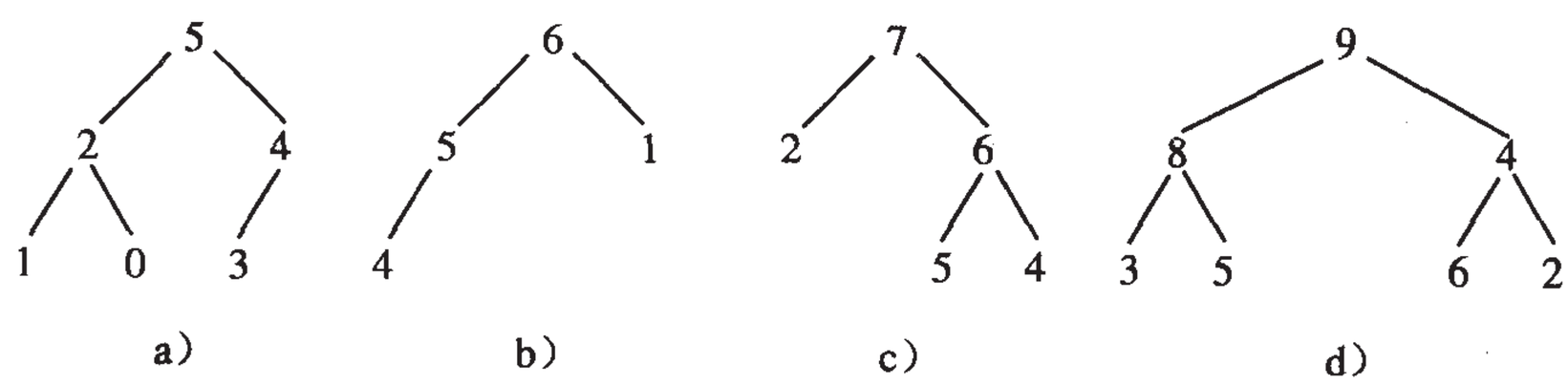


图 5-5

图 5-5a 和图 5-5b 是堆积（检查每一个节点是否都大于它的两个后代）；图 5-5c 不是堆积，因为它不是一个完全树状结构，所以每一个节点的值都比后代的大；图 5-5d 也不是堆积，显然它是个完全树状结构，原因就在于节点 4 的值不比两个后代大。

如何排出堆积来呢？其实也不难，例如，如图 5-6 所示的情况。

假设在 15 与 13 以上都满足堆积的条件，但是 10 却不是，因为 10 比 15 与 13 都小。正因为 15 与 13 以下是个堆积，要把 10 也变成堆积是不太难的。把 10 的后代中大的那个找出来，即 15，然后与 10 互换，因此 10,15,13 这三者中最大的就跑到顶上，而 10 就到了较大的后代所在的位置；不过问题就出来了，因为 15 原来是在那个位置中的堆积顶端，但现在把 10 换了下来，以 10 为顶点的部分还是个堆积吗？用图 5-6 来看显然就不是，不过问题却已经解决了一半，如图 5-7 所示。

Download at http://www.pin5i.com/

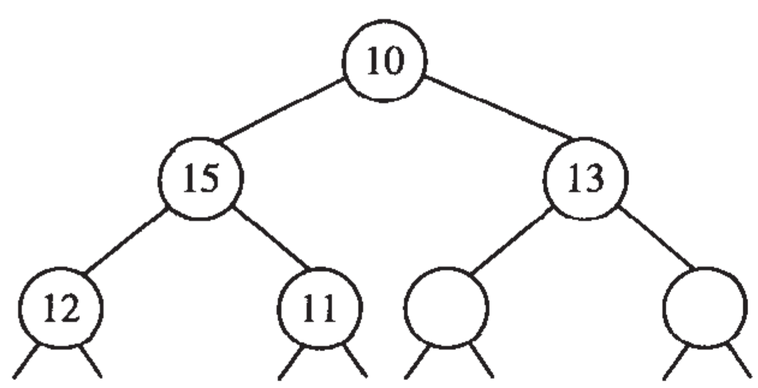


图 5-6

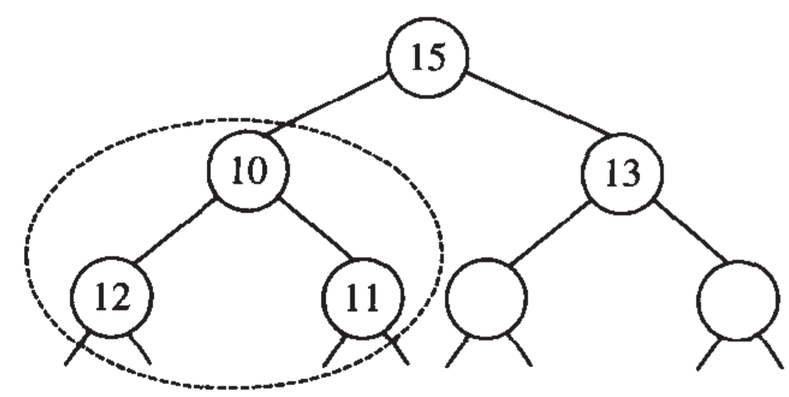


图 5-7

为什么呢？因为 13 作为顶点的部分本来就是个堆积（这是一开始时的假设），现在它上方的是 15，比它还大，所以 15 与它右边的部分是个堆积，惟一有问题的，就是原来在顶点的 10 所处的新位置，因而使虚线部分可能变成不是个堆积。

不过这没有问题，因为若 10 以下不是堆积，那么把 10 以下的部分取出来，如图 5-8 所示。

这正是同一个问题，只是换了个角度考虑而已。反复上面的操作，把 10 的后代中大的那个找出来（12），与 10 互换，因此得到如图 5-9 所示的形式。

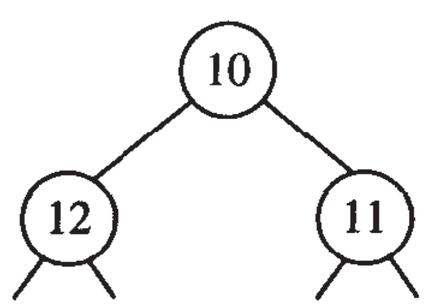


图 5-8

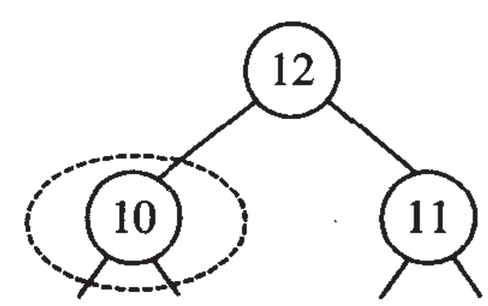


图 5-9

把它拼回原处，如图 5-10 所示。

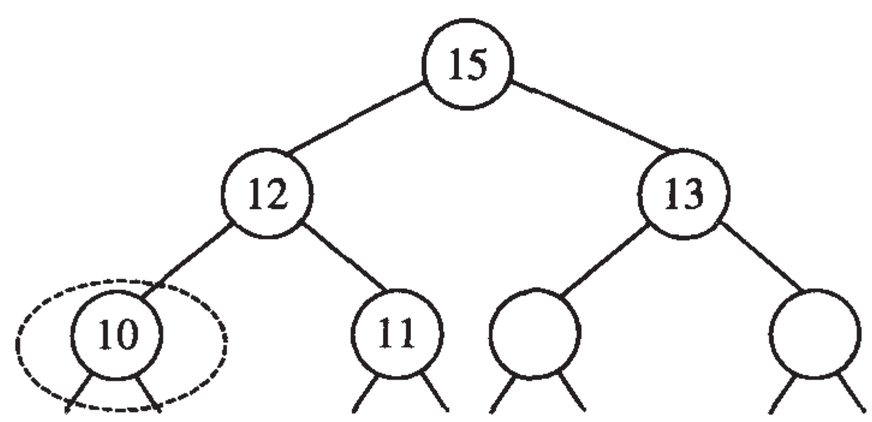


图 5-10

于是不是堆积的地方又向下移了一层，就这样一直反复地向下走，最后不外乎把 10 调到底部，而没有任何后代，10“以下”就一定是个堆积，那当然整棵树也就是堆积了；另一个可能是到了某个节点之后，10 比它的两个后代都大，于是 10 以下就是堆积。因此，就把这个完全树状结构转换成堆积了。

有了上面的观点，就掌握了如何把只有顶端不满足堆积条件，而其他节点都满足堆积条件的树还原成一个堆积。如何把一个完全树状结构变成堆积呢？这也不难，首先注意到叶部的节点根本不成问题，因为它们没有后代，于是从底向上、从右向左地处理，如图 5-11 所示。

先把最底层的 3 个元素的部分从右向左处理后，也就是说，如果它们不满足堆积条件，就用上面的方法把它们还原成堆栈。接着，向上移一层，如图 5-12 所示。

Download at http://www.pin5i.com/

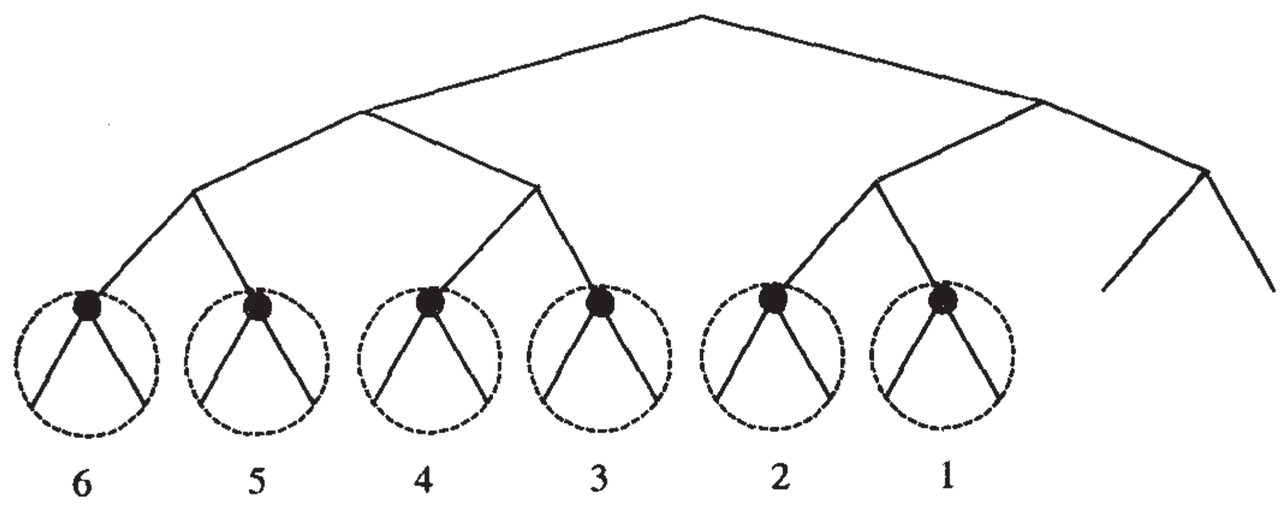


图 5-11

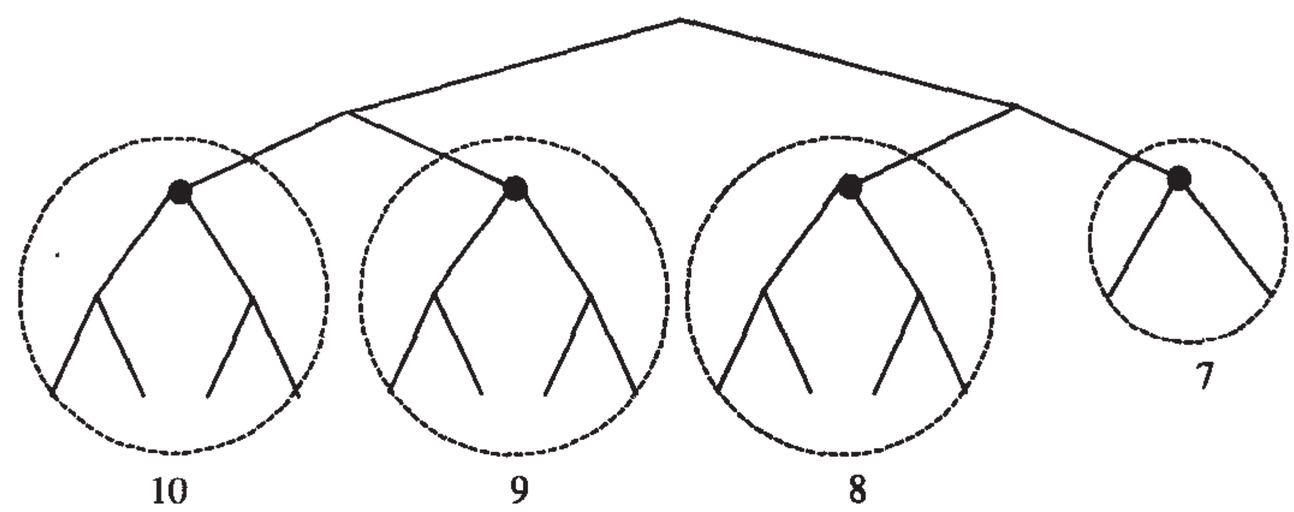


图 5-12

现在的树状结构有两层了，顶上的元素可能不满足堆积条件，但它们的后代却一定是堆积，这是上一节的结果，做完这几个部分树状结构之后，它们就都是堆积了，因而再往上一层，如图 5-13 所示。

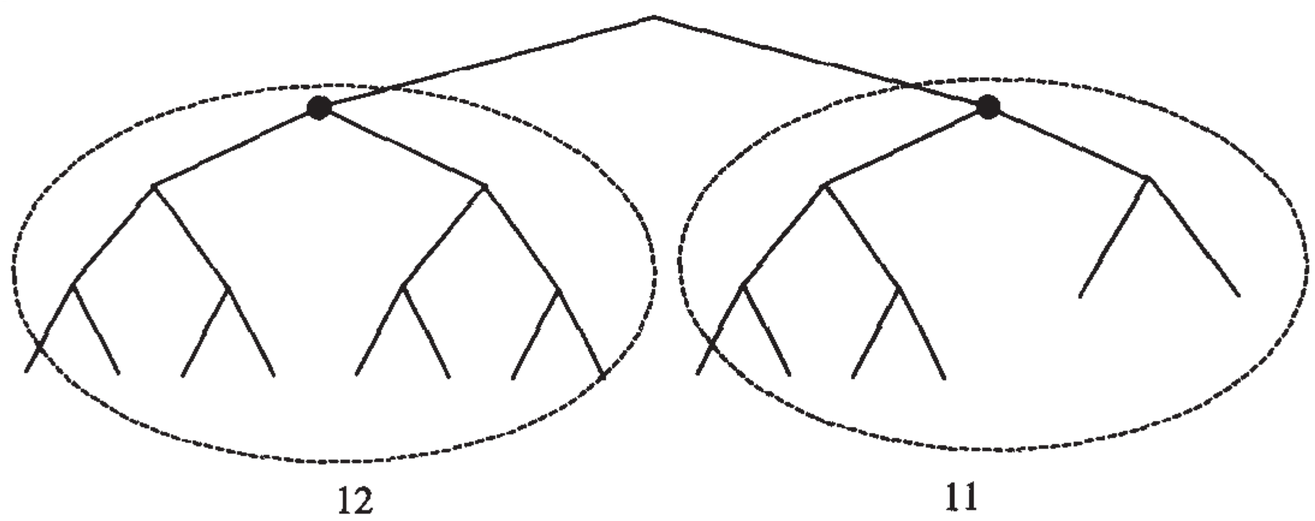


图 5-13

这只有两个（但却大了许多的）树状结构，但是仍然满足条件，亦即它的后代以下都是堆积，只有顶点本身可能不是。再把它们用上述方法还原，就留下根部可能不满足堆积条件了，最后处理根部即可。这样，就可以把一个完全没有堆积面目的完全树状结构改变成堆积。

如何用堆积排序？这是最简单的一步了，为什么呢？因为在顶上的元素是最大的，所以可以把它摘下来，因而最大值就有了，接着把堆积底部最右边的元素切断，把它的值放在顶上，因此就一方面得到一个最大的值，一方面少了一个元素；但现在还是不是堆积呢？很可能不是了，因为新放到顶上的元素极有可能破坏了堆积的结构。不过庆幸的是，这样的破坏方式并不会造成大乱，仅有一点破坏而已，其他的节点因为没有动过，所以各个节点（除了顶点之外）的下方都还是一个堆积。可以用上面的方法，把顶点再还原成堆积，

Download at http://www.pin5i.com/

于是最大的元素就又会出现在顶点，不过这个最大的元素可能就是原来数据中次大的了，所以排序的节点就排好了两个（最大与次大）。反复进行这个过程，就可以按从大到小的顺序把所有元素排好。

这就是堆积法的大概，编程时要注意以下几点：

（1）没有必要用树状结构，用数组就行了，若要达到这一点，要了解每一个元素在数组中的相对位置；有一个简单的办法是这样的：把顶点元素放在数组第一个位置，它的两个后代放在第二、第三个位置，然后就从上到下、从左到右地排下去。那么 x[i] 的后代在什么地方呢？

（2）也没有必要用额外的数组来存放排好顺序的元素，事实上可以在原地完成工作。

（3）建议编写一个名为 fix_heap() 的函数，用来接收一个元素，把该元素以下的部分还原成堆积；然后就可以通过这个函数来排序了。

（4）注意 C 语言的数组脚码从 0 开始的事实，在处理上有没有什么困难？

问题 5.8 改良的堆积法（HEAP_NEW.C）

在用堆积法排顺序时会有一个很明显的缺点，许多优秀的程序员或许都能看出来。如果现在要决定 x[i] 是不是应该在目前的位置，需要找出在堆积中 x[i] 的 x[2×i] 与 x[2×i+1]，再用这两者中大的一个（用一次比较）与 x[i] 相比（第二个比较），如果 x[i] 比较大，问题就解决了；但若 x[i] 比较小，就得把 x[2×i] 与 x[2×i+1] 中大的那一个与 x[i] 互换，再去处理在新位置的 x[i] 值。很明显地，把一个值每往下移一层就要两次比较，是不是太多了一些？请运用一项技巧来克服这一短处。

【说明】

如图 5-14 所示，它是一个堆积的某部分。在箭头所指的地方的元素不满足堆积的条件，所以找它的后代（98，81）与大的那个比，于是两次比较把 98 放到上方空位；下来再找原 98 所在地的后代，大的一个是 80，与 50 比较，比 50 大，所以再两次比较把 80 送到原 98 所在的地方；第三步，用原 80 所在地的后代中大的一个与 50 比，即 70 与 50 比，因此把 70 送到原 80 的所在地；最后拿原 70 所在的两个后代中大的那一个 55 与 50 比，还是比 50 大，所以把 55 移到原 70 所在，而由 50 占据 55 的位置，一共用了 8 次比较，是不是太多了呢？

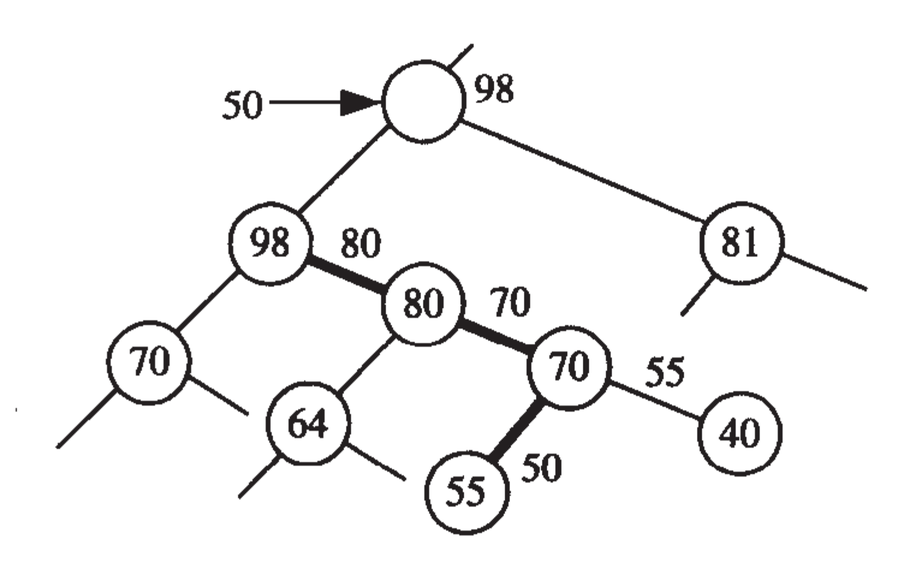


图 5-14

Download at http://www.pin5i.com/

提示：仔细看看在比较过程中所经过的路径，它有什么特色？父与子的关系如何？突破这一点，编写的程序也不一定比原来的 fix_heap()长。

问题 5.9 合并法（M_SORT.C）

合并排列法（Merge Sort）是把要排序的数据分成大约相等的两组，把这两组的顺序排好之后，通过合并的方法而合成一个全部都排好的结果。编写一个程序实现这个想法。

【说明】

这是一个很标准的分而治之（Divide and Conquer）方法的运用，假设要排序的 n 个数据是放在一个数组中，如图 5-15 所示。

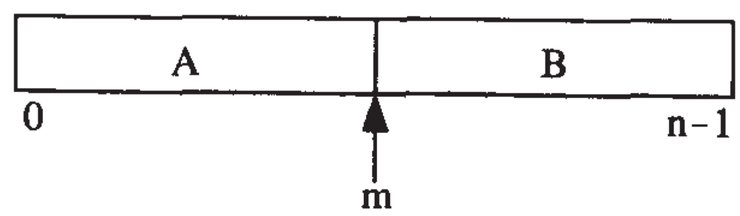


图 5-15

把它分成均等的两段，即 A 与 B，这是分的动作；接着以递归的形式把这两段排好，这是治的动作；最后，当这两部分都排好了，使用合并（Merge）的方法就可以重新合成一个全部排列完成的数组，这正是合的动作。

有了这种方法，能否写出一个又简洁又有效率的程序呢？

问题 5.10 桶子法（BUCKET.C）

快速排列法（Quick Sort）、二分插入法（Binary Insertion Sort）、堆积法（Heap Sort），合并法（Merge Sort）等，都至少要用与 $n\log_2 n$ 成正比的比较次数来排 n 个数据（当然，不一定是数值）。但如果要排的不过是一群正整数（unsigned int），有没有更快的方法？请写出这个特殊排序的程序，目的是打破 $n\log_2 n$ 的限制。

【说明】

在一些数据结构或算法的书中都会提到，任何一个“一般性”的排序方法都至少要用与 $n\log_2 n$ 成正比的比较次数。但是在实际应用中，要排序的对象却往往是一连串的正整数，因此似乎用不着“一般性”的方法才对，所以这个问题是有没有一个特殊的方法来排正整数，不但如此，还要求更快，而且打破 $n\log_2 n$ 的限制。心目中有这种方法吗？

注意一个方法上的问题，$n\log_2 n$ 这个值的来源，假设在排大小的时候只会用到比较、移动元素的操作，而不对被排列的数据作进一步的处理，因此理论家说，这就是“决

Download at http://www.pin5i.com/

策树模型”。不过现代的计算机的功能肯定不止比较、移动元素这两者，肯定还有其他的功能，所以为了打破这一限制，就不能只用“决策树模型”，而要运用进一步的技巧了。如果对数据处理一次，就认为对它作了一次比较，那么程序一共作了多少次“比较”呢？

问题 5.11 单一重复元素排序（LOT_DUP.C）

如果一个数组中有 n 个元素，但是并非每一个元素都不相同；事实上，只有大约 $\alpha\log_2 n$ 个值是不同的，其余 $n-\alpha\log_2 n$ 个值全部一样，请开发一个特别快的程序把数组的元素排好顺序。α 是个常数，为了方便起见，可以当成 1；另外，$n-\alpha\log_2 n>0$，而且 n 比 $\alpha\log_2 n$ 大很多。请问程序用了多少次比较？

【说明】

如果 n 是 16，$\log_2 n$ 就是 4，这个题目是说，在 16 个数中只有 4 个左右是不同的。下面就是一个例子：

3 7 3 3 4 3 3 3
3 3 3 9 3 3 3 3

这 16 个数中只有 3,4,7,9 是不同的，其他都与这四者中之一相同。这是一个非常特殊的排序题目，如果用快速排列法这些一般方法去排的话，就会很浪费时间，因为有太多相同的元素在内。

理论上，任何一般用途的排序方法都至少要用与 $n\log_2 n$ 成正比的比较次数，但是对这个特殊问题而言，因为有太多相同的元素在内，往往只要与 n 成比例的比较次数就可以了，有没有办法做到这一点？

问题 5.12 均匀重复元素排序（LOG_DUP.C）

如果一个数组有 n 个元素，但却只有 $\alpha\log_2 n$ 个不同的值，换句话说，数组中有 $\alpha\log_2 n$ 不同的值，每一个都重复出现了 $n/(\alpha\log_2 n)$ 次。编写一个特别快的程序把数组的元素排好顺序。假设 n 很大，比 $\alpha\log_2 n$ 大很多，α 是个常数；试问程序用了多少次比较。为了方便起见，可以把 α 当成 1。

【说明】

课本中学到的大小排列方法，其实是一种极为一般性的，对任何类型数据都适用的方法。但是对特殊的数据而言，这些一般性的方法就不见得有其长处了。什么是特殊数据？例如几乎已经排好的数组、只在某个范围中的整数，或者是有很多重复的数据等。已经看到过用 Bucket 法排正整数，或有 $\log_2 n$ 个数相异，其他完全相同的数组，都只需

Download at http://www.pin5i.com/

要大约与 n 成正比的比较次数就可以了。在本题中是另一个特殊的例子，在 n 个数据中有 $\log_2 n$ 个不同值，这意味着在最坏的情况下，n 个数据分成 $\log_2 n$ 组，每一组中的数据完全相同；最好的情况则是除了这 $\log_2 n$ 值之外，剩下的 $n-\log_2 n$ 个值都相同。后者在问题 5.11 中见过了，比较次数大概与 n 成正比，在这个问题中可以得出前者答案，因为前者包含了后者。

这个题目用不着高深的数据结构，用本书中的方法，再加上一点灵巧的心思就可以做出来。一般的排序方法都会用与 $n\log_2 n$ 成正比的比较次数，读者应该试试写一个能够达到与 $n\log_2(\log_2 n)$ 成正比的比较次数的程序。

提示：不能去排整个数组，应在合并（Merge）的动作上动动脑筋，另外，如果学过数据结构，就应该看出有一个极为适用的结构。

问题 5.13　堆积式合并（HEAPMERG.C）

已知 m 个数组，每一个数组都有 n 个元素；为了方便起见，这 m 个数组正好是 x[ ][ ] 矩阵中的 m 列。如果这 m 列中的元素都各自以从小到大的顺序排好了，编写一个程序把这 m·n 个元素合并到一个大矩阵中。不过在做这个题目时有进一步的限制，因为内存有限，大概最多只有与 m 成正比的内存可以使用。请问如何克服这一层限制？当然，程序的速度就会慢了。

【说明】

这个题目最简单的做法就是把矩阵两列两列地合并，因为每列有 n 个元素，合并 n 列大约只要与 m·n 成正比的比较次数，但内存却差不多要多出一倍，也就是 2mn，而所想克服的就是这一点，用时间来换取空间。

其实这个题目有它实用上的意义；如果把 m 个已排好顺序的数组想象成 m 个已经排列好的文件，想把这 m 个文件合成一个更大的、排好顺序的文件时，两者对比就很容易明白为什么要有那一层限制了。文件的长度是很大的，常常大到内存放不下，但文件的数目却显然不会太多，因此限制就是能够用与文件数目成正比的内存来做这件事。

假设内存中可以储存 10 000 个数，而有 1 000 000 个存放在文件中的数要排列，就可以把 1 000 000 个数每次读 10 000 个，排好后放到一个文件中；再读接下来的 10 000 个数，排好后放到第二个文件，因此共产生 100 个文件，接下来用差不多与 100 成正比的内存就可以把 100 个文件合并成一个排列完成的大文件了。当然，或许读者会想，100 个文件多不可思议。不过这没有问题，如果嫌 100 个太多，25 个如何？可以把上面的步骤分 4 次完成，在每一次都用 25 个文件，而合并成一个文件（内容为原文件的 1/4）；最后再做一次 4 个文件的合并就可以得到结果。

当然，这并不是一个好的排序的建议，但事实上的确是一个可行的做法，笔者用过的一些系统中就把这种方法在排序系统的某一环中运用过。

Download at http://www.pin5i.com/

读者能够想到一种方法来做这件事吗？

问题 5.14 检查数组元素是否相异（UNIQUE.C）

已知一个数组，编写一个程序把相同的元素去掉而只留下一个。

【说明】

这个问题几乎所有人都会做，简单极了，但正因为简单，就会常常不知不觉地接受常见的、没有效率的做法而不自知。一般做法是：假设已经给定的数组 x[ ]，元素有 n 个；再令 m 是已经找到的全不相同的元素个数，开始时 x[0] 一定可以在数组中，所以 m 为 0。

接着令 i 为 1～n-1，对于每一个 x[i] 而言，看看它是否与 x[0],x[2],…,x[m-1] 这些元素相同，如果不相同，那些相异的元素就又多了一个，因此令 m 为 m+1，把 x[i] 放到这个新的位置中；但若 x[i] 与某个在 x[0]～x[m-1] 的元素相同，那么 x[i] 就可以去掉了。所以当全部 n 个元素都处理完后，x[0]～x[m] 中就是那些相异的元素。

这种方法是正确的，但一共比较了多少次呢？在最坏的情况下，就是 n 个元素全不相同，处理到 x[i] 时 m 的值为 i-1，x[0],x[1],…,x[i-1]都不一样，一共比较了 i-1 次。因此程序令 i 从 2 变到 m 时，比较了：

$$1+2+\cdots+(i-1)+\cdots+(n-1)=\frac{n(n-1)}{2}$$

也就是近乎 n^2 次。这是个比较慢的方法，有没有办法打破这个限制，写出更快的程序？

问题 5.15 数组中和为零的段落（ZEROSUM.C）

已知一个整数数组，其中可能有正整数、负整数与 0，而且元素也不会重复出现，编写一个程序找出这个数组中是否有一个子数组元素的和为零。所谓的子数组，就是原来数组中连续的若干（至少一）个元素。例如，如果数组为 1,2,3,-5,4，那么 2,3,-5 这个子数组中元素的和就是零。

【说明】

很多程序员的做法是：如果原来数组 x[ ] 有 n 个元素，那么就准备一个 n×n 的矩阵 $A=[a_{ij}]$，其中 a_{ij} 储存了 x[i]～x[j] 的和，换言之：

$$a_{ij}=x[i]+x[i+1]+\cdots+x[j],\ i\leqslant j$$

当把 $A=[a_{ij}]$ 算出来之后，再去查有没有值为零的元素；当然还有其他的做法，有的也许不用 A 矩阵，但是还是在算 a_{ij}。这样的做法要做 $n(n-1)/2$ 次比较，因为有 $n(n-1)/2$ 个 a_{ij}，亦即 x[i]+…+x[j]，效率是很差的，不但如此，还要多出额外的空间。对于这个问题而言，程序应该做到比 $n(n-1)/2$ 还少的比较次数。

Download at http://www.pin5i.com/

问题 5.16 平面上的极大点（MAXSET.C）

在平面上如果有两个点(x,y)与(a,b)，说(x,y)支配（Dominate）了(a,b)，这就是指 x≥a 而且 y≥b；用图来看就是(a,b)座落在以(x,y)为右上角的一个无限的区域中。对于平面上的任意一个有限点集合而言，一定存在有若干个点，它们不会被集合中任何一个点所支配，这些点构成一个所谓的极大集合。编写一个程序，读入一个点的集合（比如以(x,y)这样的坐标形式），找出这个集合中的极大集合。

【说明】

如果(x,y)支配(a,b)，x≥a，y≥b，那么(a,b)就在x的左方、y的下方，如图5-16所示。

如果有n个点，那么就有可能某些点被支配，又有某些点支配其他点了，如图5-17所示。

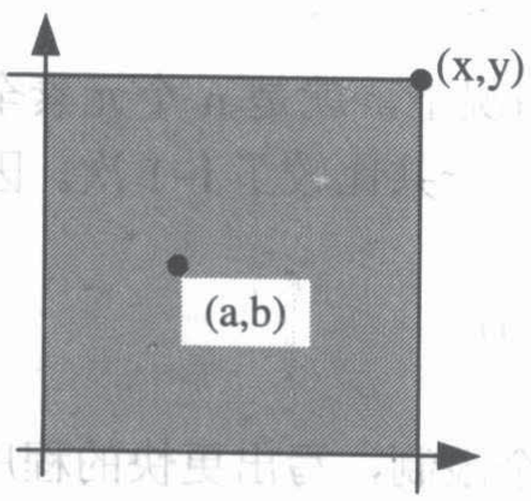


图 5-16

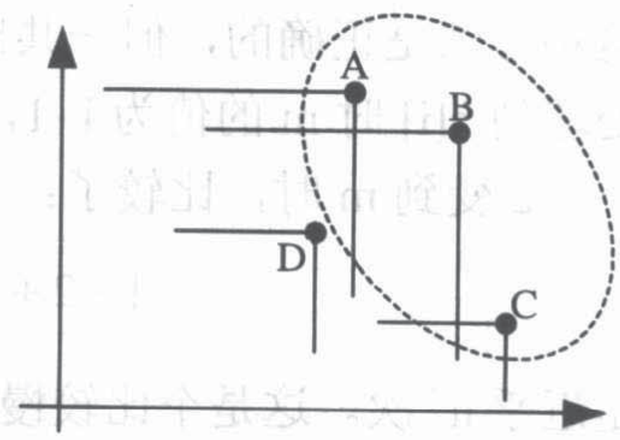


图 5-17

在图5-17中，A支配D，B也支配D，但C却什么也没支配；不但如此，A,B,C都不曾被任何点支配，所以极大集合就是{A,B,C}。

严格地说，程序是不难写的，可以先固定一个点(x,y)，然后对于每一个不是(x,y)的点(a,b)而言，检查(a,b)是否支配(x,y)，如果是，(x,y)就不可能在极大集合中；所以当每一个点都做过这一步骤之后，就知道哪些点不在极大集合里，当然其他的点就构成极大集合。但是，这种方法比较慢，大约要用到与n^2成正比的检查次数，所以要求使这个程序快一些。

在此有几点建议。第一，排大小有点用处；第二，在极大集合中的点的y坐标有什么特性？如果能够多画几个图，很快就可以想出一个漂亮而且简单的方法。

问题 5.17 宴会中访问数目的极大值（MAXVISIT.C）

在一个宴会中一共有n位来宾，按照来宾的到达与离去的登记时间，知道第i位来宾在x_i时到达，在y_i时离开，因此第i位来宾在宴会中的时间是$[x_i, y_i]$，亦即$x_i \leqslant t \leqslant y_i$中

Download at http://www.pin5i.com/

所有可能的 t，编写一个程序，读入 x_i 与 y_i，$1 \leqslant i \leqslant n$，找出同一时刻之内最多会有多少人同时在宴会场所中。

【说明】

例如，若有如此数据，如表 5-1 所示。

表 5-1

号　码	到　达	离　开
1	1	4
2	2	4
3	4	7
4	5	6
5	3	8

于是在宴会场中人数最多时是 3 个人，分别是 t=3 与 t=5 的时候；t=3 时在会场中的是{1,2,5}，而 t=5 时则是{3,4,5}；注意，在 t=4 时似乎有{1,2,3,5}在宴会场，但这是不对的，因为离开的时间 1 与 2 已经是不在会场中了，在 t=4 时，{3,5}在会场，3 刚来，但{1,2}已经离开了。另外，不能假设 x_i，亦即到达时间已经依次排好；换句话说，到达时间可能是乱七八糟的。

解这个题目有几种方法，但是为了要找出最多的在场人数，都需要与 $n\log_2 n$ 成正比的比较次数。提示很简单，时间为 t 时，若 $x_i \leqslant t \leqslant y_i$，那么 i 就在宴会场。当然，可以认定输入数据一定满足 $x_i < y_i$ 这个条件。

问题 5.18　包含在其他区间中的区间（CONTAIN.C）

已知一组区间，$[a_i, b_i]$，i=1,2,…,n；编写一个程序，找出这些区间中有哪一些会包含在其他的某个区间中。

【说明】

为了处理问题的方便，不妨假设 $a_i < b_i$，所有 a_i 与 b_i 都是整数，而且没有重复出现的区间。如果有 3 个区间[0,2]、[2,4]、[1,3]，于是没有任何一个区间会包含在第一个区间之内；但若区间是[1,3]、[0,2]、[2,3]、[3,4]，于是[2,3]就包含在[1,3]之内，但[1,3]、[0,2]、[3,4]都不包含在任何一个区间中，如图 5-18 所示。

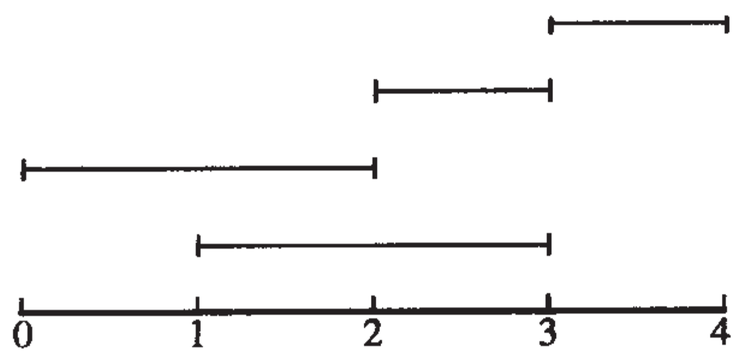


图 5-18

Download at http://www.pin5i.com/

这个题目看起来杂乱无章，但是事实上有一个很简单的解决方法。

提示：何不仔细地研究一下上一个求最多访客的例题呢？这个题目的解法与问题 5.17 的技巧多少是有一些类似的。

Download at http://www.pin5i.com/

第6章 字 符 串

问题 6.1 括号匹配问题（PARCOUNT.C）

在某个字符串中包含有左括号、右括号与其他符号；规定（与常见的数学式子一样）任何一个左括号都从内向外地与在它右边、距离最近的右括号相匹配。编写一个程序，找出无法匹配的左括号与右括号，并且在输入列下方把它们标出来。

【说明】

如果只要求找出括号是否匹配是很简单的，通常可以用一个计数用的变量，开始时是零，碰到一个左括号就加 1，碰到右括号减 1。如果处理过程中发现这个计数的值小于 0，就表示右括号多于左括号；如果处理完了，这个值还大于 0，则表示左括号比较多。不过，这个方法无法指出发生错误的左（或右）括号在什么地方，这个问题就是要写一个程序克服这一点。

下面用一个例子来看这个问题，输入：

```
((A+B)+-*/(X+Y)()((())ABC)
```

算一下就知道左括号多一个，但哪一个是多余的呢？记住问题的规则，每一个左括号都与它右边，最靠近它的右括号相匹配，所以括号的配对是(A+B),(X+Y),(),((())ABC)，所以多余的左括号在最左边，应该在最左边的左括号下方做一个记号，比如：

```
((A+B)+ -*/(X+Y)()((())ABC)
$
```

同样地，如果有多余的右括号，就用不同的记号，比如说：

```
(A+B)(C+D))+A-B(D+E)(T
    ?          $
```

用最简单的数据结构来写这个程序。

问题 6.2 转换成后继式写法（POLISH.C）

编写一个程序，读入一道正确的算术式，把它转译成反向波兰形式（Reversed Polish Form）。为了方便起见，假设整道算术式在同一列上，并且变量只有一个英文字母，不含常数（换言之，所有运算都是一个字母的变量）。目前，只需要处理+、-、*、/、(、)即可，没有正负号。

Download at http://www.pin5i.com/

【说明】

反向波兰形式是用来表示一道表达式的常见写法，是由波兰学派的逻辑学家发展出来表示逻辑式子用的。它有一个很重要的特性，就是不会用到括号。在寻常的算术式子中，运算是写在两个操作数之间，但反向波兰形式则是把运算写在对应的操作数后方。例如，a+b 写成 a b +，a+b*c 写成 a b c * +，a*b+c 写成 a b * c +。对 a+b 而言，+的操作数是 a 与 b，所以写成 a b +是很明显的；再看 a+b*c，最先算的是 b*c，因而写成 b c *，+的两个操作数是 a 与 b*c，因此就写成 a b c * +；同理，a*b+c 写成 a b * c +就不必解说了。

如果算术式子中有括号，括号中的部分对某个运算而言就是一个完整的操作数。在(a+b)*(c+d)中，a+b 的写法是 a b +，c+d 的写法是 c d +，但 a b +与 c d +又是*的操作数，因此得到 a b + c d + *。再看一例，a*(b+c)-(e+f)/g，a 与 b c +是*的操作数，所以这一部分可以写成 a b + c *，同理(e+f)/g 可以写成 e f + g /，因此整个式子就是 a b + c * e f + g / -。

从上面的说明可以看到，一旦写成反向波兰形式，括号就没有了，这是它方便的地方。但是，如何把算术式转变成反向波兰形式呢？如果仔细研究上述的结果，可以有下面的结论：

（1）反向波兰形式中变量的顺序与原来算术式中的顺序相同。

（2）运算永远出现在它的两个操作数后方。

因为上述的第一点理由，当读到一个操作数的时候，就可以把它输出，因为操作数在输入与输出中顺序相同。运算又如何呢？运算是比较麻烦的，先不管左、右括号，因为在自左向右地读取输入时，看到了一个+号，此时不但还没有见到它的右边操作数，更看不到下一个运算，于是无法决定这个+号是不是可以计算。例如 a+b*c，如果现在才读入+，就没有办法知道这个加法能不能计算，一直到*出现才能作决定（也就是不能算）；同样，看到了*也不表示它能够算，为什么呢？如果输入是 a+b*(x+y)，把 x+y 算出来之后才能算*法。换句话说，在下一个运算出现之前，并不知道目前这个运算可不可以算；回想一下先乘除后加减的规则，如果下一个运算（例如+）的优先级比现在的（例如*）低，那么就可以算现在的这个。正因为如此，用堆栈是个好方法，把不能算的运算先推到堆栈中，于是现在的这一个就在顶上，当下一个运算出现了，就看看顶上那一个与下一个的关系，如果顶上（现在）的这一个可以算，就把它输出。如果碰到左、右括号又怎么办呢？

以上就是个简单的提示。其实，可以在其他的书中找到解答，但建议不要这样做，因为在计算机刚问世时这是个难题，如果能够独立解决它而不求助外力的话，是很值得自豪的，但如果学过了这方面的知识，就不妨跳过去，做一做后面的习题。注意，以上的方法与提示并不是最好的，可以参看有关编译程序的专业书籍来寻求更好、更普遍性的解法。

问题 6.3　计算前置式写法（PREFIX.C）

课本中都会提到如何算一道用反置波兰形式（Reversed Polish Form）的表达式的计算方法，但如何计算前置式波兰形式（Prefix Polish Form）的表达式呢？为了简单起见，表达式在同一列上；只有加、减、乘、除 4 个运算；操作数只有一个数字符号。编写一个程序，

Download at http://www.pin5i.com/

接收一道前置式波兰形式的表达式，把结果求出来。

【说明】

前置式波兰形式，顾名思义，就是把运算放在它的两个操作数前方的写法，与反置（也称为后继式）波兰形式一样，它也不必用括号。于是 3+4 就写成+ 3 4，3+4*5 则是+ 3 * 4 5，而(a+b)*(c+d)就是* + a b + c d 了。再看比较复杂的，3+4+5+6+7 是+ + + + 3 4 5 6 7，但 3+(4+(5+(6+7)))则是+ 3 + 4 + 5 + 6 + 7，而 3+(4+(5+6))+7 却是+ + 3 + 4 + 5 6 7；最后 2*(3+4)/(4+5)-6/(7+8)的结果为-* 2 / + 3 4 + 4 5 / 6 + 7 8。

应该了解如何算反置波兰形式，它用一个堆栈来存放还不能计算的操作数，一直到对应的运算出现了，才取出那两个对应的操作数，算出结果，把结果放回堆栈，直到输入都用完了，结果就在堆栈中。

算前置式波兰形式也是类似的，不过比较复杂，因为不但会有还不能算的操作数，更有还不能算的运算。例如，计算+ + 3 + 4 + 5 6 7：最前面两个加号显然不能算，因为操作数还没出现，到了 3 也不能算，如此一直继续，要到达 6 了，才会有完整的两个操作数与对应的运算，因此结果为：

+ + 3 + 4 + 5 6 7 → + + 3 + 4 11 7

接着+ 4 11 就可以算了，因此有：

+ + 3 15 7

接下来+3 15 就可以算了，故有+ 18 7，最后就得了 25 这个结果。

虽然常常都只会提到后继式写法，但是前置式的写法却也是常见的。例如，汇编语言不都是先写运算（或操作）码，再写对应的运算吗？在 LISP 语言中，也是先写了一个运算，再接着写操作数的。但是，为什么它不常见呢？因为前置式有个缺点，看看+ 3 + 4 + 5 + 6 7 就不难明白，一直到最后一个符号（7）出现之前，整道式子都还不能计算，这就是前置式有，而后继式不会出现的缺点，而且把一般表达式转换成前置式，写法也复杂些（但却并不比较困难）。

虽然如此，写个程序练习如何计算前置式波兰形式却也不是件没意义的事，不妨与后继式的做法相互比较印证一番。

问题 6.4　Knuth-Morris-Pratt 法寻找字符串（KMP.C）

试开发一个效率较高的寻找部分字符串的程序。

【说明】

这是一个很常见的题目，但是大多数的解法都与下面的片段相类似（假设已知的字符串在 t[ ]中，而要找的部分字符串在 p[ ]），如程序 6-1 所示。

【程序】

程序 6-1

```
for (i = 0; i <= strlen(t)-strlen(p); i++) {
```

Download at http://www.pin5i.com/


```
        for (j = 0, k = i;j < strlen(p)&& t[k] == p[j];j++, k++)
            ;
        if ( j >= strlen(p)) return i;
    }
    return -1;
```

不过，这却不是很有效率的做法，在 t[ ]与 p[ ]不大的时候，倒也无所谓，但是当 t[ ]很大（例如在编修程序之下，去找某个字符串，于是 t[ ]就是源文件的长度）的时候，这个写法就会很慢了。为什么慢呢？原因是做了很多重复的工作。

如果 t[ ]与 p[ ]的内容如下：

t[ ] = bcbcbabcabcaabcabcabcacab

p[ ] = abcabcacab

如果用上面的片段来检查 p[ ]在不在 t[ ]中，会得到如图 6-1 所示的结果。在图中，有进行比较的部分列出了 p[ ]中的内容，而小数点的部分是没有比较的部分；所以，只要算一算不是小数点的符号有多少个，就表示进行了多少次比较，在图 6-1 中会得到 44 次。

如图 6-1 所示为 t[ ]与 p[ ]的比较，把 p[0]与某个 t[i]对齐，然后就向右边比较，一直到有一个 p[i]与 t[k]不相等了，就把 p[0]与 t[i+1]对齐，再重复这个动作。注意，把 p[0]与 t[i+1]对齐，就相当于把 t[ ]看成固定的，而把 p[ ]向右移一位，这是个重要的看法。

```
t[ ] = bcbcbabcabcaabcabcabcacab
p[ ] = a···
        a···
         a···
          a···
           a···
            abcabcac···
             a···
              a···
               abcab···
                a···
                 a···
                  ab···
                   abcabcac···
                    a···
                     a···
                      abcabcacab
```

图 6-1

为什么说图 6-1 中有很多重复的工作呢？看 p[0]与 t[5]对齐的地方，把它抄在下面：

```
t[ ]   bcbcbabcabcaabcabcabcacab
p[ ]         abcabcac···
```

当发现 a 与 c 不等时，实在是没有必要把 p[ ]向右移一位造成下面这样：

Download at http://www.pin5i.com/

t[ ] bcbcbabcabcaabcabcabcacab

p[ ] abcabcac…

接下来的几步中会发现 a 与 b 不相等，a 与 c 不相等（见图 6-1）。但是，在已经相匹配的那一段中 abcabca…（注意，最后一个 c 没有算进去，因为它与 t[ ]中对应的部分(a)不相等）有重复的部分。能看出匹配的最后哪一部分不是 abca 吗？前半段也是 abca，可以把 p[ ]向右移 3 位而得到：

t[ ] bcbcbabcabcaabcabcabcacab

p[ ] abcabcac…

abcab…

所以就不必去比开头的 abc 这 3 个符号了。

一般而言，如果 t[ ]与 p[ ]的比较已经进行到 t[k]与 p[j]，但却发现它们不相等，就有了如图 6-2 所示的把 p[ ]多移几格的情况。

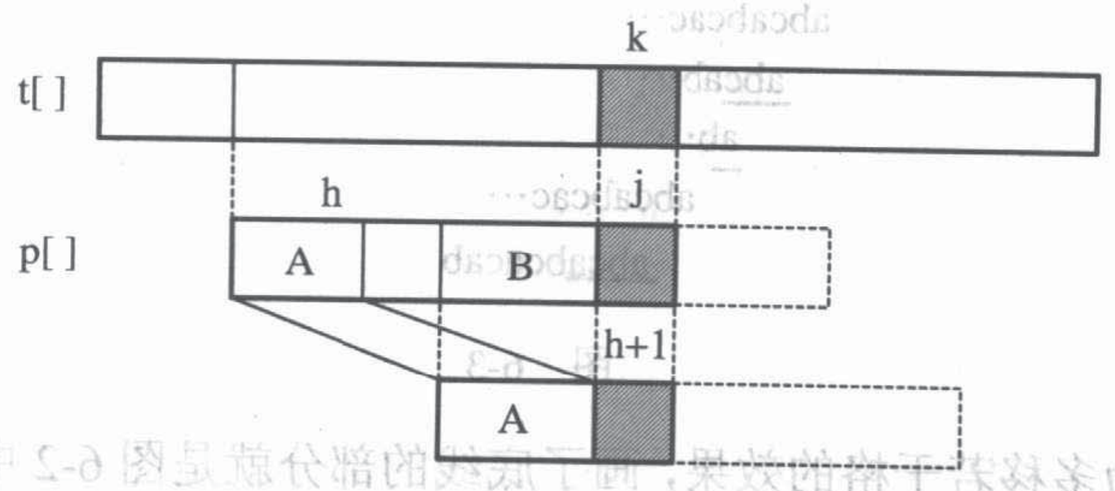


图 6-2

在图 6-2 中，p[0]～p[j-1]与 t[ ]中对应的部分是相等的，但 t[k]却不等于 p[j]，这表示为了要继续寻找部分字符串的工作，p[ ]就得右移了；前面简单方法只右移一格，但有时（如前所述）是可以一口气移许多格的。假设在 p[0]～p[j-1]这一段能吻合 t[ ]某一段的部分中，前 h+1 个元素（p[0]～p[h]）与后 h 个元素（p[j-h]～p[j-h-1]）相同，把 p[ ]的前 h+1 个元素（A）与 p[ ]的后 h+1 个元素（B）对齐，因为 A=B，所以对齐之后，前 h+1 个元素是不必再比较的，只要将 p[h+1]与 t[k]继续比较即可，如图 6-2 所示。

首先把前面的例子再做一次。前面 5 个符号的动作同前，然后自 t[5]起有：

t[ ] bcbcbabcabcaabcabcabcacab

p[ ] abcabcac…

于是 abcabca 就是已经吻合的部分，再看看它前段与后段有没有相同的地方；如果令 A=abca，B=abca（注意，有重叠没有关系）因此就把 A 部分移到 B 的位置就行了，由此得到：

t[ ] bcbcbabcabcaabcabcabcacab

p[ ] abcabcac…

abcab…

p[ ]中吻合部分（也就是 A）为 abca，下一个要比较的是 b，它不等于 t[ ]中对应的元素（a），所以故技重施，找出 abca 的前段与后段有没有相等的地方。结果发现 A=a，B=a，所以把 A 放到 B 的下方，有：

Download at http://www.pin5i.com/


```
t[ ]   bcbcbabcabcaabcabcabcacab
p[ ]           abcabcac…
                 abcab…
                     ab…
```

现在下一个要比较的是 p[1]，它是 b，还是不等于 t[ ]的对应元素，但是已吻合的部分（只有 a）已经无法找到 A 与 B 这不同的两部分了，因此只好像简单方法一样，乖乖地右移一格，接下来的工作列在图 6-3 中。

```
t[ ] = bcbcbabcabcaabcabcabcacab
p[ ] = a…
        a…
         a…
          a…
           a…
            abcabcac…
             abcab…
              ab…
               abcabcac…
                abcabcacab
```

图 6-3

如图 6-3 所示为多移若干格的效果，画了底线的部分就是图 6-2 中的 A，已经知道它与上一列的 B 部分是相同的，因此把它右移对齐而已，事实上并没有进行比较。那么图 6-3 中一共有多少次比较呢？算一算就知道了，29 次，少了 15 次。

因此，对于 p[0]～p[j-1]而言，如果找得出 A（或 B）部分所在，而且也不用很多次比较的话，就有了一个比简单方法更好的寻找字符串程序了。不过，如何找出 A 或 B 呢？假设对某个 i 而言，已经有了 p[0]～p[i]中的 A 与 B 的部分，就假设 A 是 p[0]～p[s]，而 B 则是 p[i-s]～p[i]。有了这个前提之后，用它来求出 p[i]～p[i+1]之间的 A 与 B，如图 6-4 所示。

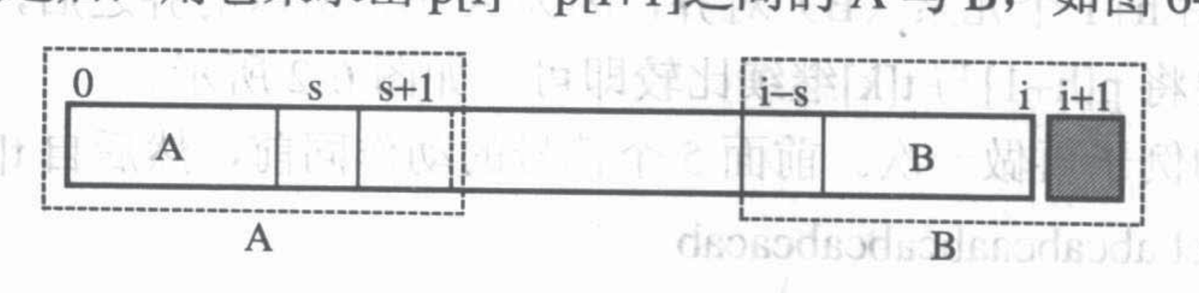


图 6-4

因为 p[0]～p[s]（A 部分）与 p[i-s]～p[i]（B 部分），依假设，它们是相同的，如果现在多了一个 p[i+1]，而且 p[i+1]与 p[s+1]相同，这就好办了，因为在 p[0]～p[i+1]中 A 是 p[0]～p[s+1]，而 B 则是 p[i-s]～p[i+1]。但是，若 p[s+1]与 p[i+1]不相等，那又怎么办呢？也不难，见图 6-5。

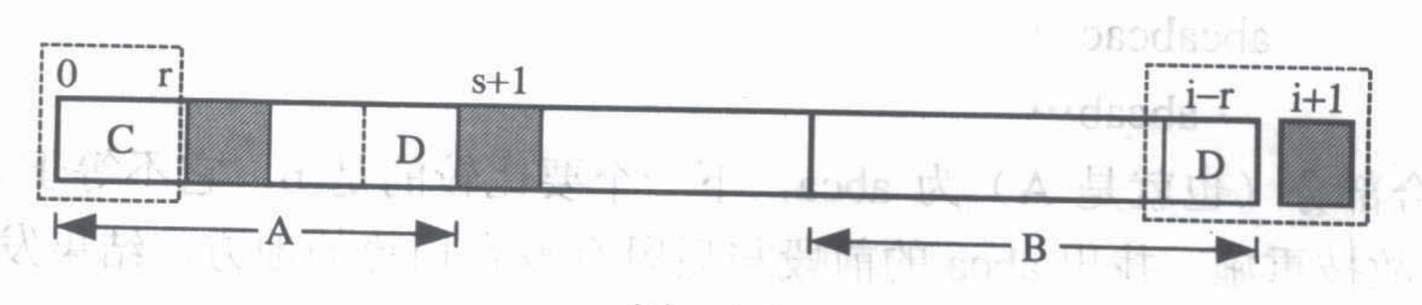


图 6-5

Download at http://www.pin5i.com/

如果 p[s+1]不等于 p[i+1]，那么在 p[0]～p[i]中的 A 与 B 部分就应该要缩短，如果真有这首尾相等的部分，就说是 C 与 D 好了；假设 C 是 p[0]～p[r]，而 D 是 p[i-r]～p[i]，然后再比较 p[r+1]与 p[i+1]就行了，于是就一直重复这个过程，到最后某个 p[r+1]与 p[i+1]相等，就是没有一个元素与 p[i+1]相等，对后者而言，就相当于右移一格从头开始了。

以前面的 abcabcacab 为例，p[0]部分是没有 A 与 B 的，因为只有一个元素；p[1]亦然，因为它与 p[0]不一样，同理可以处理 p[2]。现在看 p[3]，很明显地 A 是 p[0]，B 是 p[3]；对 p[4]而言，A 是 p[0]～p[1]，B 是 p[3]与 p[4]；p[5]时，A 是 p[0]～p[2]，B 是 p[3]～p[5]；p[6]时 A 是 p[0]，B 是 p[6]，其他的可以依次类推。

现在整个过程应该了解了，请把它写成程序。

问题 6.5　Boyer-Moore 法寻找字符串（BM.C）

根据实验与理论的分析，KMP 方法在一般情况下，并不见得会比传统的写法快多少，不过还有更好写而且平均会比 KMP 快的方法存在。这个方法差不多与 KMP 方法同时由 Boyer 与 Moore 两人发现，这个题目主要就是在探讨 Boyer-Moore 方法。

【说明】

在与 Knuth、Morris 和 Pratt 提出 KMP 方法的同时，Boyer 与 Moore 也提出另一个方法，通常都称为 BM 法；事实上，Knuth、Mooris、Pratt 等人的文章中的最后一部分就已经对 Boyer-Moore 方法作了理论上的分析。

出人意料的是，Boyer-Moore 方法（以下简称 BM 法或 BM）与常见常用的技巧习惯总是在比较时从左往右处理不大相同，BM 法是从右向左。

为了说明 BM 方法的观念，还是看个例子，如果要在：

if you wish to understand others you must …

中找 must 这个字的话；从头把 must 与上述句子对齐：

```
if you wish to understand others you must …
|
must
    |
    must
        |
        must
            |
            must
```

最先是 must 与 if y 对齐，对齐之后就从右往左比较，一开始就发现 y 与 t 不相等，不但如此，y 也不出现在 must 当中，因此很确切地说，if y 与 must 不相等，而且更重要的是，如果有与 must 相等的部分，就一定在 y 的右方，而绝对不会跨越 y（因为 y 不出现在 must 中）；因此一下子就可以把 must 向右移 4 格。接着，must 与 ou w 比较，这也是从右往左做，很明显地 t 与 w 不相等，按照上面的讨论，要有与 must 相等的部分，就一定在 w 的右边

Download at http://www.pin5i.com/

（注意，因为 w 也不出现在 must 中）。于是再往右移 4 格，因此两个比较就移动了 8 格。同样的道理，must 与 ish 比较时也造成右移 4 格的效果，这就到了 to u 与 must 比较的时候了。如果 u 与 t 不相等，must 可以向右移 4 格吗？这是 BM 方法的关键。

事实上，答案是不能，原因就在于 must 中有 u 这个字，下一步可能很快就会想到，把这两个 u 对齐如何？正是如此：

```
if you wish to understand others you must ...
   |
   must
     |
     must
      ||
       must
         |
         must
          |
          must
```

当把 to u 与 must 的 u 对齐时，如果其他的符号都依次相等，就找到部分字符串了。因此，对齐之后的结果是 und 与 must 比较，很明显地 d 与 t 不等，且 d 也不出现在 must 中，因此又可以向右移 4 格去比较 erst 与 must。

在比较 erst 与 must 时，t 与 s 是相等的，但 r 却不等于 u，也不出现在 must 中。如果原字符串含有 must 的话，就一定是在 r 的右方（不正是同样的道理吗？），所以如果 must 移到 r 的右边，就继续比较 stan 与 must。由于 n 与 t 不相等，而且 n 也不在 must 中，所以又向右移 4 格，比较 d ot 与 must；下一步是由于 o 不等于 s，于是把 must 移到 o 的右边比较 ther 与 must，继续这个步骤，很快就可以找到 must 的。

这就是 Boyer-Moore 方法的基本概念，请把它写成程序。

问题 6.6　所谓的 h-序列（RH_SEQ.C,H_SEQ.C）

这个题目是要写一个程序，接收一个字符串，辨认它是不是一个 h-序列。h-序列的定义是：第一，0 这个数字符号是一个 h-序列；第二，任何的 h-序列如果不是一个 0 的话，就是从 1 开始，后面跟着两个 h-序列。

【说明】

如果希望有一个明确的定义，就是这样：

h-seq　::= '0'

h-seq　::= '1'h-seq h-seq

这是个递归的定义，就像是 n!与 Fibonacci 数列一样，由这个定义，100 是一个 h-序列；为什么呢？因为 0 是 h-序列，所以把 100 中的两个 0 都换成 h-seq，因此就是 1h-seqh-seq，由第二个定义，它是一个 h-序列。

同理 10100 也是个 h-序列，可以把 0 先都换成 h-seq，得到 1 h-seq 1 h-seq h-seq，后面

Download at http://www.pin5i.com/

3 项可以用第二个定义换成 h-seq，于是变为 1h-seqh-seq；这 3 项就又可以化简成 h-seq，自然地就说明了 10100 是一个 h-序列。同样地，1101000 也是个 h-序列，原因是中间的 100 是个 h-seq，因而原式变成 110h-seq0，再把剩下的 0 也换成 h-seq，于是有 11h-seqh-seqh-seq，中间的 1h-seqh-seq 可以换成 h-seq，故得到 1h-seqh-seq，当然这又可以化简成 h-seq。因此 1101000 是个 h-序列。

不过，100100 却不是个 h-序列，因为只能化简到 h-seqh-seq，最左边还少一个 1；同理 0000 也不是。此外，还可以自己多加分析。

有很多方法来写这个程序，因为定义是递归的，自然地可能很快就会用递归方式写出程序，但有没有办法不用递归呢？

问题 6.7 寻找部分序列（SUBSEQ.C）

如果 s 是一个字符串，把其中（在任何位置）的符号去掉，留下来的内容就是 s 的一个子序列。例如，如果 s 的内容是“abcdefg”，去掉 b、d、f，留下“aceg”；去掉 e、f、g，留下“abcde”；去掉 a、b、c、e、g 得到“df”。于是“aceg”、“abcde”与“df”都是原来字符串“abcdefg”的子序列（Subsequence），或说是部分序列。编写一个函数，接收字符串 text[ ]与 pat[ ]，看看 pat[ ]是否为 text[ ]的子序列，并且把 pat[ ]在 text[ ]中各符号的位置记录下来。

【说明】

找部分字符串要比找子序列难，因为在部分字符串的情况下，字符串中各个符号都相邻，但找子序列时则不然。看上面提到的例子，如果 text[ ]是“abcdefg”，那么 pat[ ]为“aceg”，如图 6-6 所示。

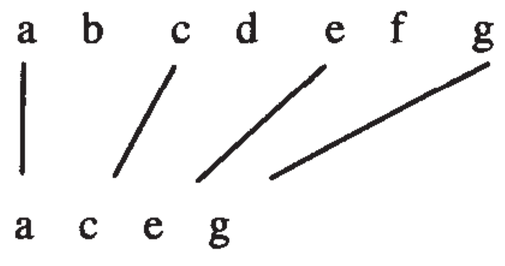


图 6-6

中间的 b、d、f 被跳过去。在写程序时要留意到这一点，程序就不难写，不必太执着于把找部分字符串的问题变化来满足要求；事实上，这是一个简单的程序。如果 text[ ]与 pat[ ]的长度分别为 m 与 n，千万不要设计出与 mn 成正比的比较次数，着眼点应放在大约 m 次就能完成的标准；当然，m≥n。

问题 6.8 最长重复部分序列（MAX_REPS.C）

如果 t 与 p 是两个字符串，把 p 中的每一个符号重复写 i 次，就得到一个新字符串 p^i，

Download at http://www.pin5i.com/

p^i是 t 的子序列吗？编写一个程序，找出最大的、使p^i是 t 的子序列的 i，如果 p 根本就不是 t 的子序列，则程序返回 0。

【说明】

如果 t 是“abcabcabcabcabcabcabcabc”，也就是把 8 个 abc 连接起来，另外 p 就是“abc”。可以看出 p 是 t 的子序列，因为 t 的头 3 个符号就是 abc；p^2，也就是“aabbcc”，是不是 t 的子序列呢？是的，下面就是一个解答，第三列则是p^3：

```
abcabcabcabcabcabcabcabc
a   ab   bc    a
a   a    ab   b   bc    c   c
```

对于p^4，亦即“aaaabbbbcccc”而言，前 4 个 a 用掉 t 中的 10 个符号，接下来的 4 个 b 就到了第 20 个符号，剩下来的只有 cabc，只能找到两个 c 了，所以令p^i仍旧是 t 的子序列的最大 i 值是 3。

可能马上就会想到，对于每一个 i=1,2,3…而言，把p^i展开，然后用问题 6.7 中 SUBSEQ.C 程序来决定p^i是否是 t 的子序列；如果不是，i 就是最大的；如果是，就把 i 加上 1，再试一次。最后总是会停下来的。例如，当p^i的长度大于 t 的长度时。当然这是一个正确的方法，但不太好。用|p|与|t|表示 p 与 t 的长度，由于p^i是把 p 的每一个符号重复 i 次，所以p^i的长度是 i|p|；正因为 i|p|≤|t|，所以 i≤|t|/|p|，亦即 i 不能超过|t|/|p|才有可能使 p 是 t 的子序列。

在问题 6.7 中说过，在处理 p 与 t 时，比较次数大约是|t|，因此用上面的做法，万一 i 的最大值正好是|t|/|p|时，比较次数就用了(|t|/|p|)·|t|；也就是$|t|^2/|p|$次了，是比较慢的。

正因为如此，知道 i 最多是|t|/|p|，最少是 1，有什么好方法可以使比较的次数降低呢？

注意：在此用p^i表示把 p 中各个符号重复写 i 次，这并不是一个标准写法，只是在这一题中用它；事实上，p^i的正常意义是把 p 接连写 i 次。

问题 6.9　最长共同部分序列（LCS.C）

如果$A=a_1a_2\cdots a_m$是一个长度为 m 的字符串，把其中的若干（可能是 0 个，也可能是 n）个符号去掉，而得到一个新字符串，这个新字符串就称为 A 的部分序列（Subsequence）。例如，若 A=abc0123，那么 b02、abc123、b3、c、abc0123、ab12 等都是 A 的部分序列。

假设给出两个字符串 A 与 B，长度分别是 m 与 n，那么 A 与 B 就含有若干共同的部分序列，至少虚字符串（或说是空字符串）就是一个共同部分序列；所谓 C 是 A 与 B 的共同部分序列，指的是 C 是 A 的部分序列，C 也是 B 的部分序列。编写一个程序，把 A 与 B 的共同部分序列中最长的那一个找出来。

这个问题一般都称为最长共同部分序列（Longest Common Subsequence）问题，简称

Download at http://www.pin5i.com/

为 LCS。

【说明】

这是一个非常有名的题目，而且是一个分支中的主要问题，这个分支称为字符串配合（String Matching），可以说是计算机科学研究领域中比较早开发的科目，目前的应用很广，从语音分析到生化都能看到这一支的踪迹，参看下面参考文献中各篇文章的介绍。

写程序的熟手或了解算法理论的朋友是不难写这个程序的，因为它不过是一个动态规划（Dynamic Programming）的应用而已。给一点提示，如果两个字符串是 $A=a_1a_2\cdots a_m$，$B=b_1b_2\cdots b_n$，考虑 a_j 与 b_j。

如果在 $a_1a_2\cdots a_{i-1}$ 与 $b_1b_2\cdots b_{j-1}$ 这两个字符串的前段中已经找到了一个长度是 k 的共同部分序列，那么会有两种可能：

（1）如果 $a_i=b_j$，于是把原来长度为 k 的共同部分序列后面补上 a_i（或 b_j，哪一个无所谓，因为两者相同），就会得到 $a_1a_2\cdots a_i$ 与 $b_1b_2\cdots b_j$ 的一个长度是 k+1 的共同部分序列。

（2）如果 $a_i\neq b_j$，分成两种情况讨论：第一，检查 $a_1a_2\cdots a_{i-1}$ 与 $b_1\cdots b_{j-1}\ b_j$ 的最长共同部分序列的长度；第二，检查 $a_1\cdots a_{i-1}\ a_i$ 与 $b_1b_2\cdots b_j$ 的最长共同部分序列的长度。最后进行适当的处置。

有了这个观点在心中，找出最长共同部分序列应该不会十分困难，但是要如何把那个序列找出来呢？这或许要好好想一想。

下面推荐的这本书是论文集，有许许多多与编修字符串方面的文章可供参考，自然也包含有最长共同序列这个问题；此外，书中还有不少这些方面的应用与说明。

R.A.Wagner and M.J.Fischer. The String-to-String Correction Problem, Journal of ACM. Vol.21(1974), pp.168～173

问题 6.10 字符串编修（STREDIT.C）

已知两个字符串 s 与 t，要研究如何把字符串 s 经由一连串编修动作变成 t。能够使用的就是插入一个符号，以及删除一个符号；把某个符号换成另一个，就可以通过先把它删除再在原地插入所需的符号来完成。编写一个程序，接收 s 与 t，找出如何才能够在最少步骤之下把 s 改成 t。

【说明】

把一个字符串修改成第一个字符串在语汇分析（Lexical Analysis）与拼字分析（Spelling Check）中有很重要的地位。例如，已知 s 这个字符串可能有问题，而在 s 中应该会出现 $t_1,t_2\cdots,t_k$ 这几个字符串其中之一，但是用哪一个比较好呢？通常会选使用最少修动而能够把 s 改出来的那一个，这一项技巧在化学中研究分子结构相当有用。

如果已知 ABCD 这个字符串，想把它改成 XBYD，一看就知道可以把 A 换成 X，C 换

Download at http://www.pin5i.com/

成 Y 就行了，这就有了 4 个动作——删除 A，插入 X，删除 C，插入 Y；但也可以把 ABCD 全部删除，再插入 XBYD，这就要 8 个动作——4 个删除，4 个插入。当然，两者相比，自然是第一个方法好些。这是一个简单的例子，但当 s 与 t 这两个字符串很长时就不那么容易看出结果了，因此这个题目就是要求编写这样的一个程序。

前面的最长部分序列（Longest Common Sequence）的技巧对本题非常有帮助，不妨先看看 LCS.C 这个程序。

问题 6.11　产生无连续重复部分的字符串（DISTSEQ.C）

编写一个程序，产生由 1,2,3 这 3 个数字符号所构成、长度为 n 的字符串，并且在字符串中对于任何一个部分字符串而言，都不会有相邻的、完全相同的部分字符串。

【说明】

这个题目对产生的字符串的规格要求相当繁琐，但是用几个例子就可以说明清楚。为了方便起见，令 n=5，于是 12321 是要求的结果，但 12123 却不是，因为在字符串中 12 与 12 是相邻而且重复的部分字符串，同理 31212 也不合要求。但 12312 合乎要求，虽然 12 重复出现了，但却不相邻。

程序并不难写，试试回溯(Backtrak)的技巧，不断地产生由 1,2,3 构成长度为 n 的字符串，并且检查它是否满足条件，可以加快程序速度的地方，就是检查所产生的字符串是否相邻的重复部分。

Download at http://www.pin5i.com/

第7章 其他问题

问题 7.1 Buffon 丢针问题（BUFFON.C）

在地面上画出距离为 L 的平行线，有一根长度为 D 的针。如果把针随意地往地面上一丢，请问压线的概率有多少？这就是有名的 Buffon（人名）丢针（Needle）问题。请写一个程序，用仿真的技巧，估计出针压线的概率。

【说明】

如图 7-1 所示，把针的一端视为一个参考点，讨论就以这个点为中心。当把针丢在地上时，这个参考点距离下方平行线有 x 长，与下方平行线成 θ 角。

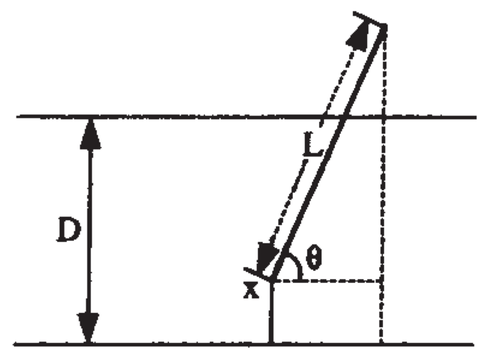


图 7-1

于是针的另一端与下方平行线的距离就是：

$$x + L\sin\theta$$

如果这个值在 0 与 D 之间，亦即：

$$0 < x + L\sin\pi < D$$

于是针就没有压线，不然针就是压线的。

因为针是随意丢的，x 与 θ 可能是 0～D 与 0～2π 之间的任何一个数。于是可以产生两个随机数，第一个在 0～D 之间，另一个在 0～2π 之间，分别表示 x 与 θ 。如果一共产生了 n 个位置，而发现 c 个是没有压线的，那么压线的概率就大概是 1-c/n。在理论上，这个问题的答案是：

$$\frac{2}{\pi}\ \frac{L}{D}$$

在写程序时不妨假定 L 与 D 都是 1，看看得到的概率是否近似于 2/π，亦即近似于 0.64。

问题 7.2 三色旗问题（FLAG.C）

假设有一个数组，它有 n 个元素，每一个不外乎是红、白、蓝 3 种颜色之一的代号，

Download at http://www.pin5i.com/

就说是 R、W、B 好了。这些元素在数组中并没有依同样颜色的元素排在一起的方式来排列，请写一个程序把这些元素排成所有蓝色在前，然后是白色，最后是红色的排列方式，不过在写程序时要满足下面的条件：

（1）不能用到额外的内存，换句话说，只能在数组之内用互换的方式完成。

（2）互换两个元素的动作要越少越好。

（3）对于每一个元素而言，测试它是红、白，还是蓝的工作，每种颜色最多只能做一次测试。

在这个限制之下，请编写一个最快的程序。

【说明】

有些人很自然地会想到把蓝色当成 1、白色当成 2、红色当成 3，然后用排大小的程序把次序排出来，自然地蓝色在前，白色居中，红色最后了；完全正确，不过却违背了规定。也就是说，互换两个元素的动作是不是太多了呢？对每一个元素测试它的颜色的行为是否违反了要求呢？对前者而言，不同的排序方法会有不同数目的互换动作。比如说，如果用选择法，每次找到 i+1～n 中最小的值，与第 i 个元素互换，那么排序的动作只不过用了 n-1 次互换。但是对第三个限制而言，就很难保证对每一个元素、每一种颜色都最多只测试一次了。

第三个条件需要仔细地说明，x[i]表示第 i 个元素。第三个条件是说，x[i]== R，x[i]==W，x[i]==B 这 3 个比较的任何一个都最多只能做一次。举个例子，如果在决定 x[i]应该放在什么地方时，上述 3 个比较都各做一次，是正确的。任取两个或任取一个各做一次，也是正确的。但 x[i]==R、x[i]==W、x[i]==R 就是错的了，因为对 R 的测试做了两次。

其实这个题目并不难，把数组从头到尾查一次就可以做出结果，那 3 个繁琐的条件，只不过是防止去用排序或“不够快”的方法来解题而已。

问题 7.3　字符串列整数的转换（X_ATOI.C）

在平常课本中的 atoi()函数范例，通常都不能查出溢位（Overflow）的情况，或者是不理会这个情况而给出错误的结果，请修改这个 atoi()程序，把转换后的结果用第二个参数返回来。例如，x_atoi(s,*t)就用 t 返回结果，而函数则指出 OVERFLOW（大于最大整数值），UNDERFLOW（小于最小整数值），NO_ERROR（正常），为了方便起见，假设使用的机器是用 2 的补码来储存整数。有没有办法查出机器是用 1 的补码还是 2 的补码？

【说明】

atoi()函数的函数值是没办法检验是否正确的，除非把它用 itoa()转换回字符串，再与原字符串比较数字符号部分是否相等。程序在得到不正确的 atoi()函数值之后还可以毫无其事地工作，除非最终结果相当“奇怪”，否则很可能就会当成正确的结果，并且接收它。正因为如此，使用 atoi()就会有潜在的危险。

Download at http://www.pin5i.com/

所以，这个题目就是要加强 atoi()的功能。例如，可以写一个 x_atoi(s,*t)函数，接收一个字符串 s，把转换结果存放在 t 中返回来，同时，再利用函数值来指出 OVERFLOW、UNDERFLOW、NO_ERROR。请问，要如何修改 atoi()这个函数呢？

问题 7.4 整数类型列的极值（LIMITS.C）

请写一个程序，求出在<limits.h>这个引入文件中所定义的、各个有关整数类型的特征值。

【说明】

或许使用的系统中的<limits.h>有更多的信息，但在这个题目中主要的对象是整数类型：char、short、int 与 long、signed 或 unsigned。要想办法，通过自己的程序找出每一个整数类型各占用多少个数元。有符号（signed）时的极大值、极小值，无符号（unsigned）时的极大值（极小值永远是 0）。不要用 sizeof，它是以 char 为基准的，而 char 占多少个数元呢？另外，在特定的情况下——也就是说只写了 char，char 类型是有符号还是无符号的呢？知道机器是用 1 的补码还是 2 的补码吗？

为了方便起见，不必把程序写得具有 100%的移植性，不妨假设符号数元在最左边，而不是采用符号数元与其他位数分离的方式。

在程序中，不能引入<limits.h>，并且把值显示出来，也不能用 sizeof，换句话说，只能靠 C 语言的基本功能来做这件事。另外更重要的是，不要假设在用 IBM PC、Mac、VAX 或任何用习惯了的机器；事实上在编写这个程序时不能以任何机器做假设。换言之，移植性固然不能做到 100%，但总是越高越好，所以不妨想到溢位时程序可能会停止执行，而失掉计算结果这件事上头；这一点很重要，一些机器有此功能，所以要小心。

问题 7.5 无限位数算术（ARITH.C）

请写一个使用十进制算术，并且可以计算任意位数精确度的正整数加法与乘法的程序。

【说明】

计算机中所存的值都只能有有限的精确度，例如 long 通常有 9～10 位，double 则大约有 15 位等，一旦计算结果越出了这个范围，值就不再是正确的了。然而在许多研究中，比如计算机代数学（Computer Algebra）、几何论证（Geometric Reasoning）等，都不能有不精确的值产生，因为产生了不精确（或近似）的值，整个推论过程甚至于结果就都不复成立。所以，计算机科学家就开发出不少这一类型具有无限精确度的软件，比如 MACSYMA、Scratchpad II、Reduce、MATHEMATICA 等都是有名的例子。

在无限精确度的加、减法中，程序本身要提供没有误差的整数运算，而究竟有多少精

Download at http://www.pin5i.com/

确度则常常由硬件或内存决定；因此，要做两个 1000 位的十进制数相乘，结果差不多会有 2000 位，一般而言并不是一件难事。

在这个题目中，要编写两个进行无限精确度算术的函数，一个是加法，一个是乘法；加法是简单的，在写乘法的程序时，千万不要直接使用手算的方式先把每个位数都乘出来，最后用加法去加，这样会使乘法函数相当没有效率，应该想想如何避免这一点。

问题 7.6　线性表示的矩阵相乘（MATMUL.C）

在 C 语言中虽然号称功能比 Pascal、Fortran 强，但是在某些方面其实却并不见得有多少改进，函数参数中的高维数组就是一个例子，如果要保持数组的结构，那么就只能在第一维留下空白而变成不完整数组，其他维度的容量还是得写出来。这一点只比 Pascal 好一点点，而与 Fortran、PL/1 或 Ada 相比还差得远。在早期 Fortran 程序员为了突破 Fortran 的限制而开发出一组很实用的技巧，本题与问题 7.7 的目的就在于此。请写 3 个函数，第一个函数把二维数组转换成一维，第二个函数把一维转换成二维，第三个函数求两个代表二维数组的一维数组运行矩阵相乘。

【说明】

要写的第一个函数不妨叫做：

```
void twoD_to_oneD(int TwoD[ ] [MAXSIZE], int m, int n, int OneD[ ]);
```

它接收一个二维数组 TwoD[ ][ ]（不知道有多少列，但内存中有 MAXSIZE 行），目前已经有了 m 列 n 行数据，再把这个数组以列为主的次序存入一个一维数组中，因此工作做完之后，OneD[ ]中有 m・n 个元素，前 n 个是 TwoD[ ][ ]的第一列，其次的 n 个为第二列等。注意，TwoD[ ][ ]与 OneD[ ][ ]可能在同一个位置。

第二个函数是：

```
void oneD_to_twoD(int OneD[ ], int TwoD[ ] [MAXSIZE], int m, int n);
```

它把一维数组 OneD[ ]还原成原来的二维数组。注意，OneD[ ]与 TwoD[ ][ ]可能在同一个位置。

第三个函数为：

```
void matmul (int a[ ], int b[ ], int c[ ], int L, int M, int N);
```

a[ ]、b[ ]都是一个二维数组转换成一维数组的形式，各有 L 列 M 行、M 列 N 行，请把 a[ ]与 b[ ]的乘积求出存入 c[ ]（L 列 N 行）。

问题 7.7　对称表示的矩阵相乘（MATMUL_S.C）

问题 7.6 的矩阵是一般形式的，但若是个对称矩阵，转换成线性形式就可以省下差不

Download at http://www.pin5i.com/

多一半的内存位置，这就不止是方便的问题了。对一个 n 列 n 行的对称矩阵（它一定是方阵）而言，如果用二维数组方式储存，就需要 m•n 个元素，但是因为是对称的缘故，储存上半（或下半）三角形的部分就足够了，所以就只需要 n(n+1)/2 个位置而已。请对照上一题，写出 StwoD_to_oneD()、oneD_to_StwoD()以及 Smatmul()这 3 个函数，但对象全都是对称矩阵。

【说明】

因为对称矩阵一定是方阵，所以上一题的 3 个函数中的矩阵的列、行数目的参数就可以合二为一，如下所示：

```
void StwoD_to_oneD(int TwoD[ ][MAXSIZE], int n, int OneD[ ]);
void oneD_to_StwoD(int OneD[ ], int TwoD[ ][MAXSIZE], int n);
void Smatmul (int a[ ], int b[ ], int c[ ], int n);
```

前两个函数与问题 7.6 是相似的，但第三个函数 Smatmul()就要小心些，因为在一维的对称矩阵中，有一半的元素并没有储存起来，而是靠另一半（因为对称的关系）记录下来的，因此在相乘时，除了第一列与最后一行之外，其他的列与行都不完全，要如何把它补完全呢？

问题 7.8 找零钱问题（CHANGE.C）

某个国家一共发行了 $a_1,a_2,a_3,\cdots,a_k$ 种不同面额的钞票，为了方便起见，假设 $a_1<a_2<\cdots<a_k$。现在手上有 n，请问要如何把 n 兑换成 $a_1,a_2,a_3,\cdots,a_k$ 这些钞票，使得所用钞票的量为最少。

【说明】

先说一个最常见的例子，如果有 1 元、5 元、10 元 3 种，而又有 107 元要兑换，于是 $a_1=1$，$a_2=5$，$a_3=10$，$n=107$。兑换方式很简单，用面额最大的去除，那就是最大面额钞票的张数，比如说 $n/a_3=107/10=10$，亦即 10 元的 10 张；除过之后就会有余数 7，再用次大面额的钞票去除，得商数 1（7/5=1），余数 2，所以 5 元的一张；再把余数用第三大面额去除（2/1）得商数 2，余数 0，于是 1 元的两张，但余数为 0，于是就没有剩下来的钱了，因此最后结果是 10 元 10 张，5 元 1 张，1 元两张。假若 $n=78$，那就是 10 元 7 张，5 元 1 张，1 元 3 张，一般书中就是这样讲的，不过对 1 元、5 元、10 元这个例子而言，倒也是正确。

如果钞票的面额是 1 元、3 元、4 元（奇怪的数字，是吧？），要兑换 10 元呢？用上面的方法，有 4 元两张（10/4=2，余 2）、1 元两张，一共用了 4 张钞票。但若用两张 3 元、1 张 4 元，也兑出了 10 元，却只用了 3 张钞票。

这个例子说明，寻常所用的方法固然可以兑换钞票，但钞票张数却不是最少的；此外问题是，请写一个程序，输入 $a_1,a_2\cdots,a_k$ 钞票面额，以及 n 个欲兑换的钱数，输出钞票张数最少的兑换方式。

Download at http://www.pin5i.com/

问题 7.9 背包问题（KNAPSACK.C）

如果有 n 个对象，它们的计量单位各为 $k_1,k_2,\cdots,k_n$；现在有一个可以容纳 K 单位的背包，请写一个程序，找出有没有办法在 1,2,…,n 中选 p 个元素出来，使得计量单位的总和为 K，可以刚好把背包装满。这就是有名的背包问题（Knapsack Problem）。

【说明】

就以计量单位分别是 1,2,4,7,10,12,13,15,17 这 9 个对象来看好了，若背包的容量是 27 单位，可以取 4,10 与 13，因此 4+10+13=27；但也可以取 2,12,13，因此 27=2+12+13；更可以取 10 与 17，于是 27=10+17；当然还有其他的可能解答。不过，这个题目却是不一定有解的。例如，若对象的计量单位是 1,5,10,15, 20；而背包是 14，物件中比 14 小的有 1,5,10，无论如何选择，都不可能加出一个 14 来的。

就像上一题找零钱的问题一样，可以用动态规划的技巧来解这个问题，不过会复杂一些，但基本道理是完全相同的。

问题 7.10 最佳矩阵相乘顺序（PRODSEQ.C）

已知一连串的矩阵，$A_0,A_1,\cdots,A_{n-1}$，要计算它们的连乘积 $A_0\times A_1\times\cdots\times A_{n-1}$；请写一个程序，读入各个矩阵的维度的容量，找出一个使乘法数目为最少的计算方式。

【说明】

一个 m 列 n 行的矩阵与另一个 n 列 k 行的矩阵相乘，如果使用传统的方式，一共要使用 mnk 个乘法，原因是将第 i 列（n 个元素）与第 j 列（也有 n 个元素）两两相乘时一共要 n 个乘法，$n-1$ 个加法；但是因为相乘的结果是一个 m 列 k 行的矩阵，上述的行为就要重复做 mk 次，因此就一共有 mnk 个乘法，$mk(n-1)$ 个加法。而加法比乘法快许多，所以一般都只计算乘法的个数。

在求 3 个矩阵 A,B,C 的连乘时，数学公式可以写成 $(A\cdot B)\cdot C=A\cdot(B\cdot C)$，换言之，先乘前两个与先乘后两个的结果是一样的，但在程序工作时效率却可能不尽相同。如果 A,B,C 分别是 a 列 b 行、b 列 c 行、c 列 d 行的矩阵，那么 $A\cdot B$ 要 abc 个乘法，结果是个 a 列 c 行的矩阵；再把这个结果与 C 这个 c 列 d 行矩阵相乘，这要 acd 个乘法，所以求 $(A\cdot B)\cdot C$ 就需要 $abc+acd=ac(b+d)$ 个乘法。先乘 $B\cdot C$ 就要用 bcd 个乘法，得出一个 b 列 d 行的矩阵，然后 A 与它相乘，要 abd 个乘法，所以 $A\cdot(B\cdot C)$ 就要用 $bcd+abd=bd(a+c)$ 个乘法。一般而言，$ac(b+d)\neq bd(a+c)$，所以就得选择值比较少的那一个方式。举个例子，如果 a=5、b=2、c=3、d=4，于是 $ac(b+d)=90$，而 $bd(a+c)=64$，于是 $A\cdot(B\cdot C)$ 的乘法个数比较少，效率比 $(A\cdot B)\cdot C$ 的高；反之，如果 a=4、b=3、c=2、d=5，$ac(b+d)=64$，

Download at http://www.pin5i.com/

但 $bd(a+c)=90$，于是（$A \cdot B$）$\cdot C$ 就比较快。因此，当一连串矩阵连乘时，选择一个合适的计算顺序往往可以使程序的效率提高很多。就以上例来看，快的方法只不过用了慢的方法的 71%左右的时间而已。

如果输入的是一连串数组的维度容量，比如说 $d_0, d_1, \cdots, d_n$，d_0 与 d_1 是第 0 个矩阵的列数与行数，d_1 与 d_2 是第一个矩阵的列数与行数，…，d_i 与 d_{i+1} 是第 i 个矩阵的列数与行数，最后，第 $n-1$ 个矩阵的列数与行数是 d_{n-1} 与 d_n。有办法找出使乘法数目最低的计算方式吗？

问题 7.11　最短路径问题（SHORTEST.C）

如果有一个矩阵 $D=[d_{ij}]$，其中 $d_{ij}>0$ 表示 i 地到 j 地的距离。若 i 与 j 之间无路可通，d_{ij} 就是无穷大（∞）；又 $d_{ij}=0$。请写一个程序，通过这个距离矩阵，把任意两个城市之间的最短距离与其行经路径找出来。

【说明】

有 4 个城市如图 7-2 所示，彼此之间有路相通的都用箭头标了出来，在旁边的就是那一条路的距离；例如，1 到 2 的距离是 5，2 到 1 却是 50。很明显地从 1 到 4 的最短距离是 10，沿途经过 1→2→4，而不是走 2→3→4，因为距离是 35。

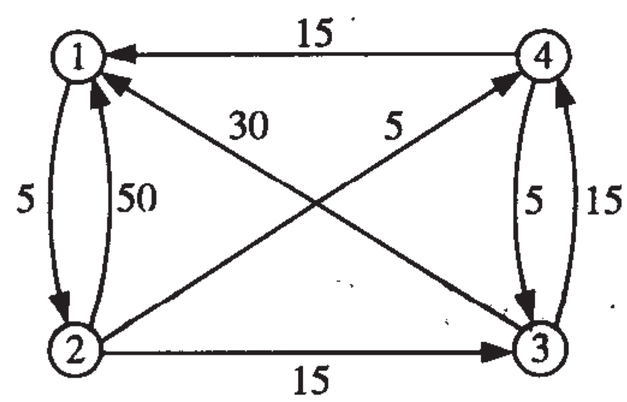


图　7-2

解法仍然是用动态规划。可以这样想，对于任何一对城市 i 与 j 而言，查每一个其他的顶点 k，看看 i→…→k，再加上 k→…→j 的距离是否比目前 i→j 的距离还少，如果是，就把 i→j 的距离改成中途经过 k 的距离。要如何找出 i 到 k 的行经路径呢？

在输入距离矩阵时建议采用下面的形式：

```
4
1    2    5
2    1   50
2    3   15
2    4    5
3    1   30
```

Download at http://www.pin5i.com/

3	4	15
4	1	15
4	3	5
0	0	0

输入 SHORTEST.IN1

在此，第一列是指出有多少个城市(4)；第二列以后各有 3 个数；第一与第二个是某条路的起点与终点，第三个数则是那一条路的距离，最后用 3 个 0 作结束。上面的数据正好对应着前面的图，城市的编号是 1～n（城市总数），不从 0 开始是避免与最后 1 列的 3 个 0 有任何冲突。

所以，最好写 3 个函数，第一个读入数据，第二个算出最短路径，第三个把路径显示出来。

为了方便，如图 7-3 所示对应着另一个输入文件 SHORTEST.IN2。可以执行程序，看看结果，确认已经了解了问题。

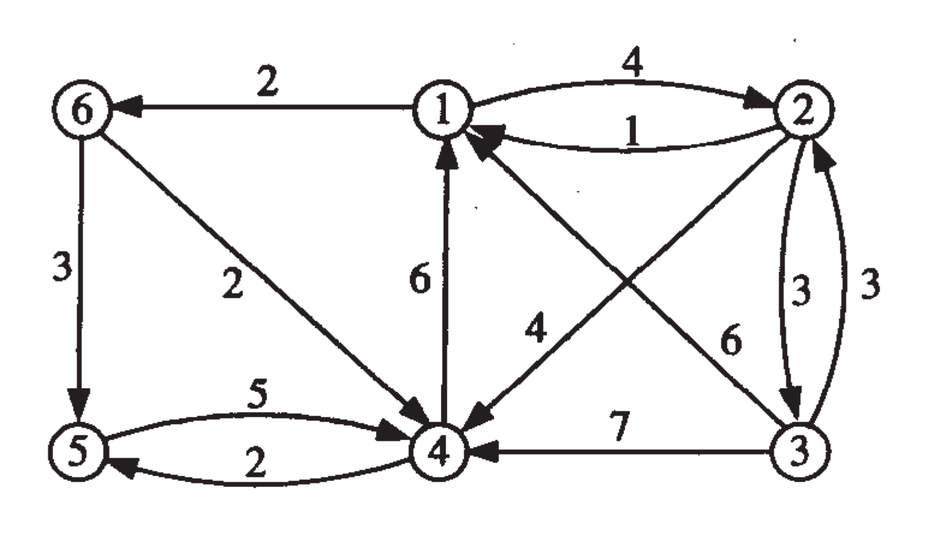


图 7-3

问题 7.12 产生匹配括号的字符串（PAR_GEN.C）

请写一个程序，输入一个正整数的值，然后列出所有由 n 个左括号与 n 个右括号正确地组成的字符串；当然，正确的左、右括号一定个数一样多，所以输入的值要是个偶数。

【说明】

所谓由括号正确地组成的字符串，指的是如果有一个左括号，那么在它的右边就一定有一个与它相匹配的右括号。例如，()()、(())，就是仅有的两个有 4 个符号的，由括号正确地组成的字符串；()()()、()(())、(())()、((()))则是 4 个有 6 个符号，由括号正确地组成的字符串。正因为有一个左（或右）括号就一定有相对应的右（或左）括号，左、右括号成双出现，因此输入就一定要是偶数，奇数是不可能的。当输入 n 之后，在字符串中左、右括号的个数就各是 n/2 个。

如何产生这样的字符串呢？下面就是一个有用的想法：如果在工作过程中已经产生了若干左、右括号，为了要把产生的行为完成，还欠 R 个左括号、L 个右括号，那么有没有

Download at http://www.pin5i.com/

办法找出产生下一个括号时 L 与 R 的关系呢？记住，递归是一个不容忽视的利器。

问题 7.13 稳定伴侣问题（STABLE.C）

有 n 个男孩 $m_1, m_2, \cdots, m_n$ 与 n 个女孩 $w_1, w_2, \cdots, w_n$。每一个男孩 m_i 都依照他喜爱这 n 个女孩的程度列成一张表，最喜欢的女孩排在第 1 位，最不喜爱的女孩排在第 n 位；同样地，每一个女孩 w_i 也依照她喜爱 n 个男孩的程度列成一张表。请写一个程序，把每一个男孩与女孩的喜爱表格读入，并且把男孩与女孩一一配对，使得：如果 m_p 与 w_q 在一对的话，那么第一，对 m_p 的喜爱表格中排在 w_q 之前的女孩而言，她的伴侣在她的表格中一定排在 m_p 之前；第二，对 w_q 的喜爱表格中排在 m_p 之前的男孩而言，他的伴侣在他的表格中一定排在 w_q 之前。这就是稳定伴侣（Stable Marriage）问题。

【说明】

这个问题恐怕要多说一些才能讲得明白。先讲不稳定（Unstable）的情况：如果 m 的女伴记成 $P_M(m)$，而 w 的男伴写成 $P_W(w)$。如果有一对男孩与女孩 m 与 w，他们不是伴侣（用上面的记号，m 的伴侣是 $P_M(m)$，w 的伴侣是 $P_W(w)$），但 m 比较中意 w 而不是 $P_M(m)$，则同时 w 比较中意 m 而不是 $P_W(w)$ 的时候，m 与 w 就一定心不甘情不愿了。例如，如果 A 与 B 是男孩，X 与 Y 是女孩，A 比较中意 X，B 比较中意 Y，A 与 Y、B 与 X 结伴，那么 A 与 Y 心目中的情人都不是对方，B 与 X 亦然，这就是不稳定的状况。稳定伴侣的问题，就是要在喜爱的表格中配出最合适、稳定的伴侣，而不是“乔太守乱点鸳鸯谱”制造对对怨偶。

看一个完整的例子。假设男孩与女孩都用编号 1,2,3,4，其喜爱表格如表 7-1 所示。

表 7-1

男孩					女孩				
1:	2	4	1	3	1:	2	1	4	3
2:	3	1	4	2	2:	4	3	1	2
3:	2	3	1	4	3:	1	4	3	2
4:	4	1	3	2	4:	2	1	4	3

表格中指出，男孩 1 最中意的女孩是 2，其次是 4 与 1，最后是 3；对于女孩 3 而言，她最中意 1，其次是 4 与 3，最后是 2，对于这两份喜欢表格而言，稳定伴侣是如表 7-2 所示。

表 7-2

男孩	女孩
1	4
2	3
3	2
4	1

Download at http://www.pin5i.com/

第 1 号男孩与第 4 号女孩配对，他们是不是怨偶？在第 1 号男孩喜欢表格中，排在第 4 号女孩之前的是第 2 号；再看第 4 号女孩，在她的喜爱表格中排在 1 之前的是第 2 号男孩，于是都符合了稳定的条件。用同样的方法可以查证其他的 3 对伴侣。

在编写程序时，可以用男孩为主，对女孩求婚，而由女孩由她的喜爱表格来决定接受还是不接受；如果可以接受，就不妨先接受，等到有更中意的男孩求婚了，就中止上一个约定（不是现实世界，不必太拘泥），另结新欢。数据证明，一定可以找出稳定伴侣的结果的。

在常见的问题中，计算机择友就是一个类似的问题，但要求却比较松些，因为男女双方不必列出对“所有”对象的喜爱程度，而且也有可能出现喜爱程度相同的情形，当然解题的方法就复杂了。

问题 7.14　单调矩阵的极值（MONO_MAX.C）

在这个题目中讲一个所谓的单调（Monotone）矩阵的概念。如果一个矩阵中的第 i 列上的极大值所在的那一行记为 i_j；$i<j$ 时，$i_j \leqslant k_j$，也就是在上方的列的极大值会座落在下方的列的极大值的左边，满足这个条件的矩阵就叫做一个单调矩阵。请写一个程序，接收一个单调矩阵，找出这个单调矩阵中每一列的极大值。请估计一下程序用了多少次比较。

【说明】

这是个简单、有趣，但是却有其他用途的题目，看一个例子来澄清问题的本意，在下面的 5×5 矩阵中：

3	2	1	1	0
1	3	1	2	1
2	3	1	0	2
3	3	5	4	1
2	3	4	5	6

第 1、2、3、4、5 列的极大值分别在第 1、2、2、3、5 行，因此上一列的极大值都在本列极大值的左上方，这正是条件。题目就在于找出每一列上的极大值，使程序输出 3,3,3,5,6。注意，只知道矩阵满足单调的条件，但却不知道极大值在何处，后者正是程序的任务。要写出程序并不难，固定列的号码，在那一列上找极大值就行了，程序要比较 m·n 次，m 与 n 是矩阵的列数与行数，而程序却没有发挥“单调”这一特性。聪明一些的程序员马上会想到这一点，从最下方一列开始做，找出极大值之后把它在那一行记住，就说在第 q 行好了，然后在处理上一列时，就在第 1～q 个元素中找极大值即可。这是个不错的想法，但还是不够好，因为还是做了差不多与 n^2 成正比的比较次数（比如说极大值都在对角线），能突破这一点吗？

Download at http://www.pin5i.com/

提示：简单得很，想想“分而治之”的策略。

问题 7.15 向量分类（CLASSIFY.C）

已知一组元素 $X_1, X_2, \cdots, X_m$，其中每一个都是有 n 个量的向量，亦即 $X_i = [x_{i1}, x_{i2}, \cdots, x_{in}]$，n 是个固定值。因为这些量，$x_{ij}$ 并不一定是数值，所以比较大小有时不见得有意义，只能假设对它们作比较相等或不相等的动作。编写一个程序，定出合理的数据结构，处理这些 $X_1, X_2, \cdots, X_m$，把它们相同的归在一类；X_i 与 X_j 相同，就指的是 $x_{i1} = x_{j1}$，$x_{i2} = x_{j2}, \cdots, X_{in} = X_{jn}$。在写程序时，不妨假定 x_{ij} 已经被编码成整数，但不能比大小。

【说明】

这是一个在实用中常见的题目，只不过换成一个容易了解的说明。一般而言，x_{ij} 不一定是一个值，而可能是一个函数，当把 i 与 j 有关的数据代入该函数时，就会得到一个分类用的值（亦即此处的 x_{ij}），但这个值却没有任何次序或大小的关系，所以用排大小的方式就完全没有意义了。

另外一点，在应用中也不能像排序时一样，通过移动元素而把相同的向量归在同一类，因此这些想法都使编程者有了如下的限制：

（1）向量中元素之间只能有相等或不等，而不能比大小。

（2）不能移动这些向量，因为它们很可能并不是一个值。

因此，有没有办法把这些向量分类呢？下面是几点可能有用的提示：

（1）为了方便起见，不妨把向量 $X_i = [x_{i1}, x_{i2}, \cdots, x_{in}]$ 想象成一个 m 列 n 行矩阵中的第 i 列，所以 x_{ij} 就正好是第 i 列第 j 行中的值；若不是一个值也无妨，可以把它想象成一个指向函数的指针。

（2）因为不能移动各列，所以只好移动指向各列的指针。

（3）因为值没有大小的关系，所以排大小的技巧派不上用场，得另想它法。

（4）分类结果一般都遵守这样的规定：$X = [x_{i1}, x_{i2}, \cdots, x_{in}]$ 与 $Y = [y_{i1}, y_{i2}, \cdots, y_{in}]$ 是两个向量，若 $x_{i1} \neq y_{i1}$，那么在输入中有某个在 Y 之前的向量，它的第 1 个分量是 x_{i1} 时，X 就排在 Y 前面；若 $x_{i1} = y_{i1}$，$x_{i2} = y_{i2}$，…，那么 $x_{in} = y_{in}$，但是 $x_{i,j+1} \neq y_{i,j+1}$，如果在输入中有某个向量的前 j 个分量与 X 以及 Y 的相同，而且第 j+1 个分量是 $x_{i,j+1}$，不但如此，该向量是出现在 Y 之前，那么 X 就在 Y 的前面。例如，若输入是：

$$X = [1,2,3]$$
$$Y = [2,3,4]$$
$$Z = [1,2,4]$$

那么输出是：

Download at http://www.pin5i.com/

$$X=[1,2,3]$$
$$Y=[1,2,4]$$
$$Z=[2,3,4]$$

对于 Z 与 Y 而言，因为在 Y 之前有一个向量（X）的第一个分量与 Z 的第一个分量相同，所以 Z 就排到前面去。

这些条件看来复杂，但事实上却都是很自然的，因为在分类时有点像字符串顺序。在写程序时，不妨先不理会最后一点，等到程序正确无误之后再作进一步的尝试。注意，不要排出顺序，而是列出哪一个向量在哪一类，因此对上面的数据而言，X,Y,Z 分别在第 1、3、2 类。

Download at http://www.pin5i.com/

第8章 游戏问题

问题 8.1 奇数阶魔方阵（MAGIC_O.C）

这个题目是填魔方阵（Magic Square）。所谓的 n 阶魔方阵，就是把 1～n^2 个连续的正整数填到一个 m・n 的方阵中，使得每一列的和、每一行的和，以及两个对角线的和都相等。在这个题目中，只填奇数阶的魔方阵，因为它好填，而且也一定会成功；如果不知道如何填法，请看下面的说明。

【说明】

下面就是一个 5 阶的魔方阵，各列、各行与两条对角线（17、5、13、21、9）与（15、14、13、12、11）的和都是 65。

17	24	1	8	15
23	5	7	14	16
4	6	13	20	22
10	12	19	21	3
11	18	25	2	9

在填表的时候，永远采用“往右上角移”的法则（可以用表 8-1 对照一番，看看是否如此），因此会碰到下面两个情况之一，如图 8-1 所示。

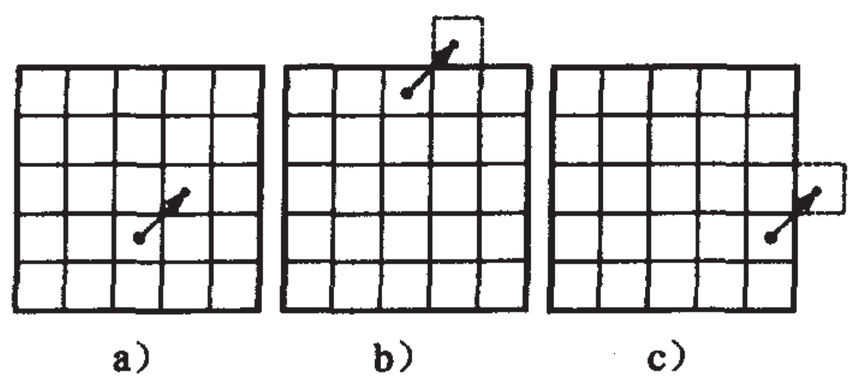


图 8-1

如图 8-1a 所示是第一种情况，往右上角移之后还在表格内，这是最简单的一种；如果现在是在第 i 列第 j 行，下一个（在右上角）的格子就会在第 i-1 列第 j+1 行。如图 8-1b 所示是第二种情况，会从上方移出表格的边缘；如图 8-1c 所示是第三种情况，则是移出右方的边缘。注意，因为是向右上角移，所以永远不会移出左方与下方的边缘。

如果是越出上方边缘，不妨把表格想成上下方是粘起来的，于是自右上方移出，就等于移到右边第一行的最下方一格；于是，若有 n 列，现在是第 i 列第 0 行，于是下一个格

Download at http://www.pin5i.com/

子就是第 n-1 列第 j+1 行（注意，C 语言的脚码是 0～n-1），如图 8-2 所示。

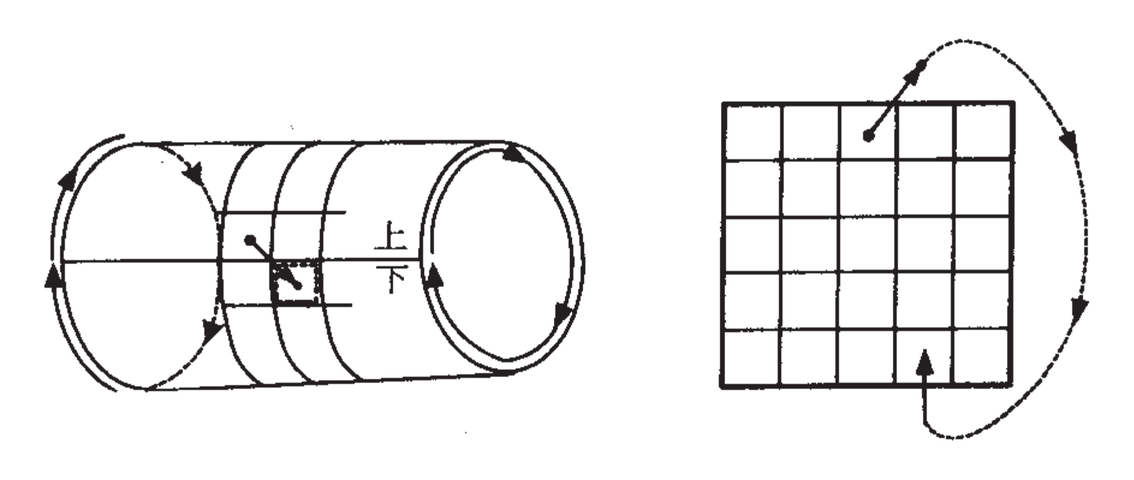


图 8-2

如果从右边移出边界，可以把表格想象成左右相连，所以向右移的话，就会移到上一列最左边的那一格，如果有 n 行，现在是在第 i 列第 n-1 行，于是下一格就是第 i-1 列第 0 行了，如图 8-3 所示。

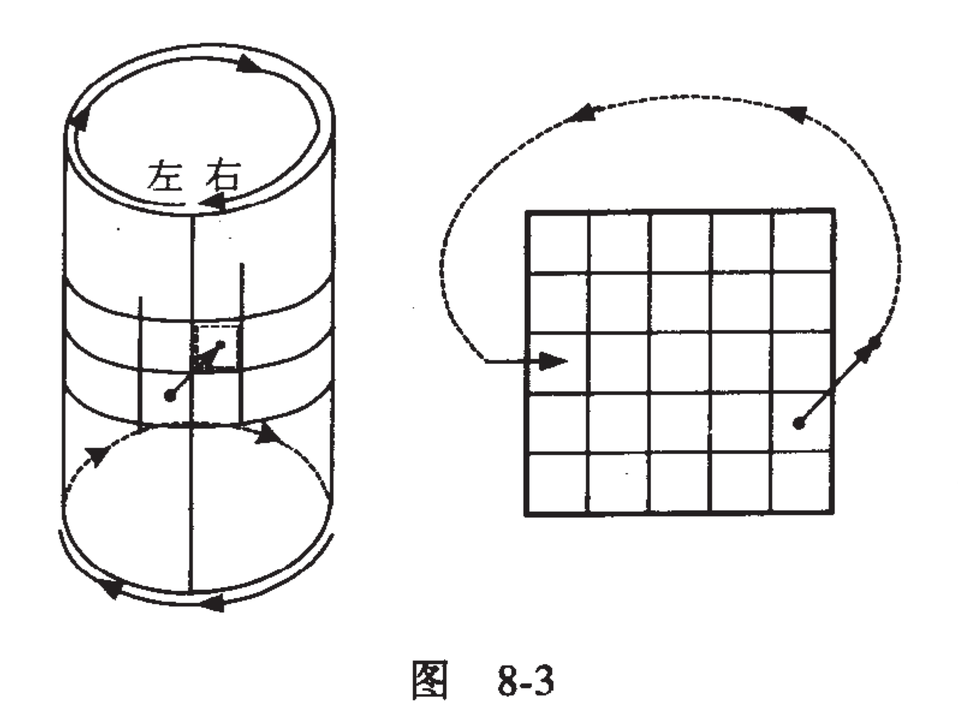


图 8-3

了解移动方法之后，就可以填表了。事实上，还有一条规则，下面很快就会讲到。在填表时是从第 0 列的中央位置，也就是第 n/2 行开始的，在那个位置上填写 1。于是就依上面的移动方式，每移动一格，就把下一个数填进去，因此若 n 是 5，填完 5 个数之后得到如图 8-4 所示情况。

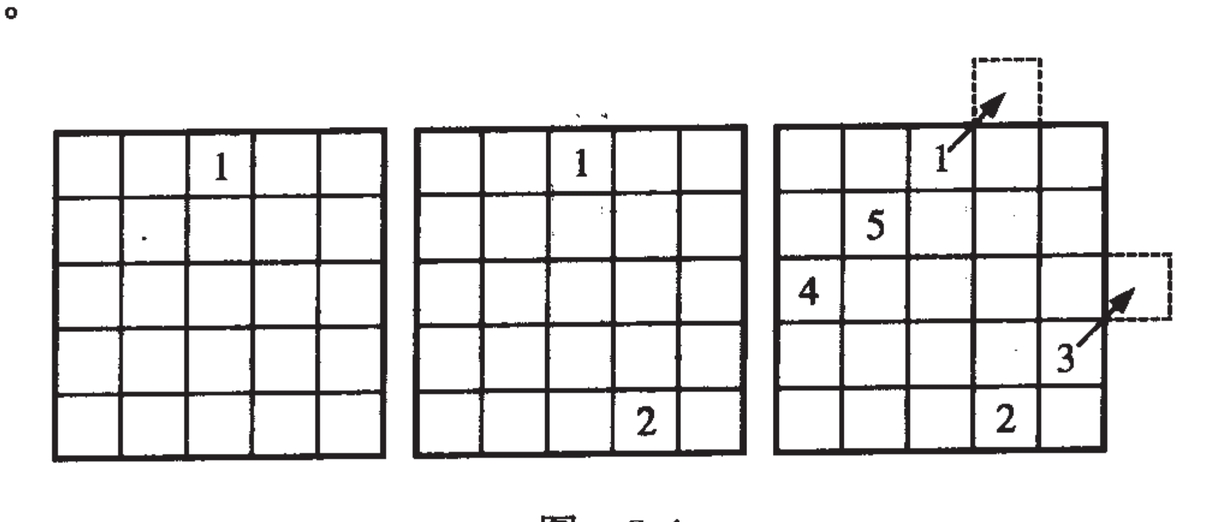


图 8-4

如果要填 6 的话，就会把原来的 1 盖过去了，因此就有了一条新规则——每填完 n 个数之后，就往下移一格，再继续填表，如图 8-5 所示。

再填完 5 个，又会有相碰的情况，所以往下移一格再填表，最终就会得到最前面的 5 个魔方阵了。

在数学上可以证明，n（奇数）阶魔方阵在填表时，每隔 n 个就一定会相碰一次，所以就总共会相碰 n 次。换言之，就要下移 n 次了。这样，就学会如何填奇数阶魔方阵了，请

Download at http://www.pin5i.com/

把程序写出来。

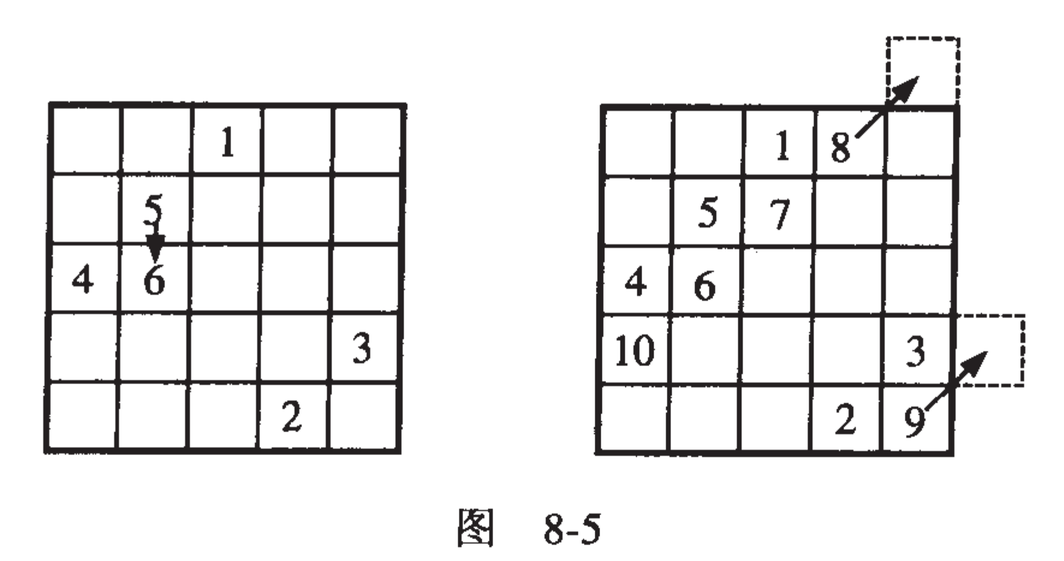


图 8-5

问题 8.2 单偶数阶魔方阵（MAGIC_SE.C）

会填奇数阶魔方阵之后，填 2(2m+1)阶的魔方阵也就不是一件难事了。请写一个程序填任何 2(2m+1)阶的魔方阵，如果不知道方法，请看下面的说明。

【说明】

2(2m+1)阶的方阵可以等分成 4 个部分，每一个部分的长与宽都是 2m+1：

	2m+1	2m+1
2m+1	A	C
2m+1	D	B

于是 A,B,C,D 这 4 个部分就都是奇数阶的。接着先填入一些准备用的数值：

（1）在 A 中以奇数阶魔方阵的方式填入 $1\sim(2m+1)^2$。

（2）在 B 中以奇数阶魔方阵的方式填入 $(2m+1)^2+1\sim 2(2m+1)^2$。

（3）在 C 中以奇数阶魔方阵的方式填入 $2(2m+1)^2+1\sim 3(2m+1)^2$。

（4）在 D 中以奇数阶魔方阵的方式填入 $3(2m+1)^2+1\sim 4(2m+1)^2$。

接下来，这 4 个部分都要动点手脚才能达到魔方阵的境界，这也可以分成以下几个步骤来处理。

（1）在 A 部分中间那一列的中央那一格起向左一共取 m 格，把它的内容与 D 的对应部分互换。

（2）在 A 部分，除了中央列之外（中央列在上一步已经处理过了），每一列的头 m 个元素与 D 的对应部分互换。

（3）在 C 部分，每一列的倒数 m-1 个元素（可能为 0 个）与 B 的对应部分互换。

（4）做完这几步，就会得到一个 2(2m+1) 阶的魔方阵了。以 m=1，亦即 2(2m+1)=6 阶魔方阵为例，做完上面的前 4 步，得到

Download at http://www.pin5i.com/

8	1	6	26	19	24
3	5	7	21	23	25
4	9	2	22	27	20
35	28	33	17	10	15
30	32	34	12	14	16
31	36	29	13	18	11

接着，后 3 步的第 1 步是在 A 部分中央那一列正中央那一格起，向左取 m=1 个格子出来，与 D 的对应部分互换，即 5 与 32 互换。第 2 步是把除了中央列外的其他列上前 m 个元素与 D 的对应部分互换，即 8 与 35、4 与 31 互换。最后的第 3 步不必做，由于 m-1=0 的缘故，就等于没有作用。最后的结果如下。

35	1	6	26	19	24
3	32	7	21	23	25
31	9	2	22	27	20
8	28	33	17	10	15
30	5	34	12	14	16
4	36	29	13	18	11

阶数是 2(2m+1)的魔方阵，一般都叫做单偶数阶（Singly Even）魔方阵。

问题 8.3　双偶数阶魔方阵（MAGIC_DE.C）

偶数阶的魔方阵是个难题，但阶数是 4 的倍数（4m）时却很简单，约略比奇数阶的复杂些而已。请写一个程序来填 4m 阶（m>0）的魔方阵。如果不知道这个方法，请看下面的说明。

【说明】

4m 阶的魔方阵可以等分成 4 个部分，每一个部分的长与宽都是 2m：

	2m	2m
2m	A	C
2m	D	B

看 P2m •2m 方阵。在这个方阵中的 $4m^2$ 个元素中要在某些元素上面做个记号，方法是：在偶数列（0,2,4,…）上的偶数行都做上记号；但对奇数列而言，则都在奇数行上做记号。

Download at http://www.pin5i.com/

因此，若 m=2，A 这一部分（*就是记号）如下：

A =

*		*	
	*		*
*		*	
	*		*

接着，把 A 的记号以 A 与 C 的分隔线为准，以对称的方式转到 C 部分，因此 A 与 C 就有了记号。下一步，则是以 A 与 D 的分隔线为准，以对称的方式把 A 与 C 中的记号分别转到 D 与 B 去，所以最后的结果为：

*		*			*		*
	*		*	*		*	
*		*			*		*
	*		*	*		*	
*		*			*		*
	*		*	*		*	
*		*			*		*
	*		*	*		*	

再其次，就是把 1～$(4m)^2$ 这些个连续的整数填到方阵中。填数字的顺序是：自上而下一列一列地处理，在同一列上，则是从左向右一格一格地处理。不过，因为填数字的方式，处理前 2m 列就够了，后 2m 列是由前 2m 列得来的。数字的选择如下：

（1）如果处理到的方格没有记号，就令下一个值为 i，填在那里，接着找出那个以方阵中心为准与目前这一格对称的方格，把$(4m)^2$-i+1 填到那。

（2）如果处理到的方格有记号，就找出那个以方阵中心为准的对称点，令下一个值为 i，填在那里，并且把$(4m)^2$-i+1 填在目前的这一格上。

如果按照这个方法去填 8×8 的方阵，再以上述的记号为准，把 1～64 填到方阵中，会得到：

64	2	62	4	5	59	7	57
9							
							56
8	58	6	60	61	3	63	1

Download at http://www.pin5i.com/

事实上，记号的做法并不一定如上述那么呆板，只要做到（在 A 部分）每一列与每一行上面都有 m 个记号就行了。因此就 2m=4 而言，有：

	*	*	
*			*
	*	*	
*			*

	*		*
*		*	
	*	*	
*			*

*	*		
		*	*
*	*		
		*	*

这几个都是合理的做法。

写程序的时候应注意脚码的运用方式，以及避免多余的运算与记号。这一类型的魔方阵一般叫做双偶数阶（DoubleEven Order）魔方阵。

问题 8.4　N 后问题公式解（N_QUEEN1.C）

8 后问题（Eight Queen Problem）是指在一个 8×8 的西洋棋盘上要如何放置 8 个皇后棋且不会互相吃到对方；皇后棋可以吃掉任何它所在的那一列、那一行，以及那两条对角线（米字型）上的任何棋子。信不信有一道公式对于每一个 n>3 而言，在 n×n 棋盘上都可以找出“一个”n 后问题的解？请依下面的说明写一个程序，读入 n，把一组解显示出来。

【说明】

不相信也不行，因为的确有此公式存在。C.F.Gauss 这位数学奇才把 8 后问题数学化，想要找出一个可以得到所有解的公式，但却没有成功，不过后人借助计算机的帮助，倒是找到了一组公式。对于 n>3 而言，在 n×n 棋盘上的某一组解可以用该公式表示出来。这一组公式可以分成 4 个部分，在下面，k 是正整数，换句话说，k=1,2,3,…，而 n 是棋盘的列数与行数。

（1）当 n 是 6k-2 或 6k 的形式时，解答是由下面两组坐标点构成：

$$A_1=\{(2i,i)\mid 1\leqslant i\leqslant \frac{1}{2}n\}$$

$$A_2=\left\{(2i-1,\frac{1}{2}n+i)\mid 1\leqslant i\leqslant \frac{1}{2}n\right\}$$

（2）当 n 是 6k-1 或 6k+1 的形式时，解答是由下面两组坐标点构成：

$$B_1=\{(2i,i)\mid 1\leqslant i\leqslant \frac{1}{2}(n-1)\}$$

$$B_2=\left\{(2i-1,\frac{1}{2}(n-1)+i)\mid 1\leqslant i\leqslant \frac{1}{2}(n+1)\right\}$$

（3）当 n 是 6k+3 的形式时，解答是由 3 组坐标点构成：

$$C_1=\left\{(1,n-1),(2,\frac{1}{2}(n-1)),(3,n)\right\}$$

Download at http://www.pin5i.com/

$$C_2=\{(2i+2,i)\mid 1\leqslant i\leqslant \frac{1}{2}(n-3)\}$$

$$C_3=\left\{(2i+3,\frac{1}{2}(n-1)+i)\mid 1\leqslant i\leqslant \frac{1}{2}(n-3)\right\}$$

（4）当 n 是 6k+8 的形式时，解答是由下列 4 组坐标点构成：

$$D_1=\left\{(2i-1,\frac{1}{2}n-2+i)\mid 1\leqslant i\leqslant 3\right\}$$

$$D_2=\{(2,n),(4,2),(6,n-1)\}$$

$$D_3=\{(2i+5,i+1)\mid 1\leqslant i\leqslant \frac{1}{2}(n-3)\}$$

$$D_4=\left\{(2i+6,\frac{1}{2}n+1+i)\mid 1\leqslant i\leqslant \frac{1}{2}(n-3)\right\}$$

（5）当 n=8 时，任选一组 8 后问题的解答就行了，使用：

$$D_1'=\{(2i,i)\mid 1\leqslant i\leqslant 4\}$$

$$D_2'=\{(1,6),(3,5),(5,8),(7,7)\}$$

以上的所有坐标点都从 1 算起。

问题 8.5　N 后问题递归解（N_QUEENR.C）

请写一个程序，读入一个值 n 表示棋盘的大小，然后求出 n×n 格棋盘上放 n 个皇后棋且不会相互吃掉对方的所有解答。

【说明】

这是广义的 N 后问题，因为所要求的是“所有”解答，而不单是其中的一组（见上题），对大多数会运用递归的人来说，这个题目反而容易做些。这一类型题目的解法通常要用到回溯（Backtrack）的技巧——不管用递归还是不用递归都是如此，虽然会浪费时间，但多半会找到解答。

回溯的技巧通常都以下面的面目出现，如程序 8-1 所示。

【程序】

程序 8-1

```
S := {目前可以用得到的位置;
WHILE S <> {} DO
 BEGIN
     取出 S 中一个元素，令为 s;
     IF s 可以安全使用 THEN
         BEGIN
             定出 s 已经使用;
             IF 还可以从 S 再进一步 THEN
```

Download at http://www.pin5i.com/


```
                    用 S 调用自己
                ELSE IF 已经有了答案 THEN
                    显示答案;
                把 s 定成没有在使用;
            END;
        END;
```

采用这种方法（并非是最好的方法），用 S 来记录在目前可以使用的一些位置，接下来的 WHILE…DO 中就一个接一个地检查它们。如果查到的是 s，那么第一件事就是看看可不可使用 s；如果可以，就把 s 标上已经使用，而它很可能就是解答之一。接着如果从 s 起还可以再向前走，于是就用 s 作目前的立足点，调用上面的程序进入下一层的工作。如果到了 s 就已经有了一个解答，那就把它显示出来。如果只要一个结果，这就是停止工作的时候了；但若要其他答案，就得继续执行。

无论如何，当 IF…THEN…ELSE 做完之后，那就表示 s 以下的内容都已经做过了，就可以试下一个 s。首先把目前的 s 定成没有在使用，这一点十分重要，因为若不做这一步，在往后试其他位置时就可能会发生问题。做完之后，回到 WHILE…DO 开头，取出下一个 s；就这样，一直循环到 S 变成空集合为止，如图 8-6 所示。

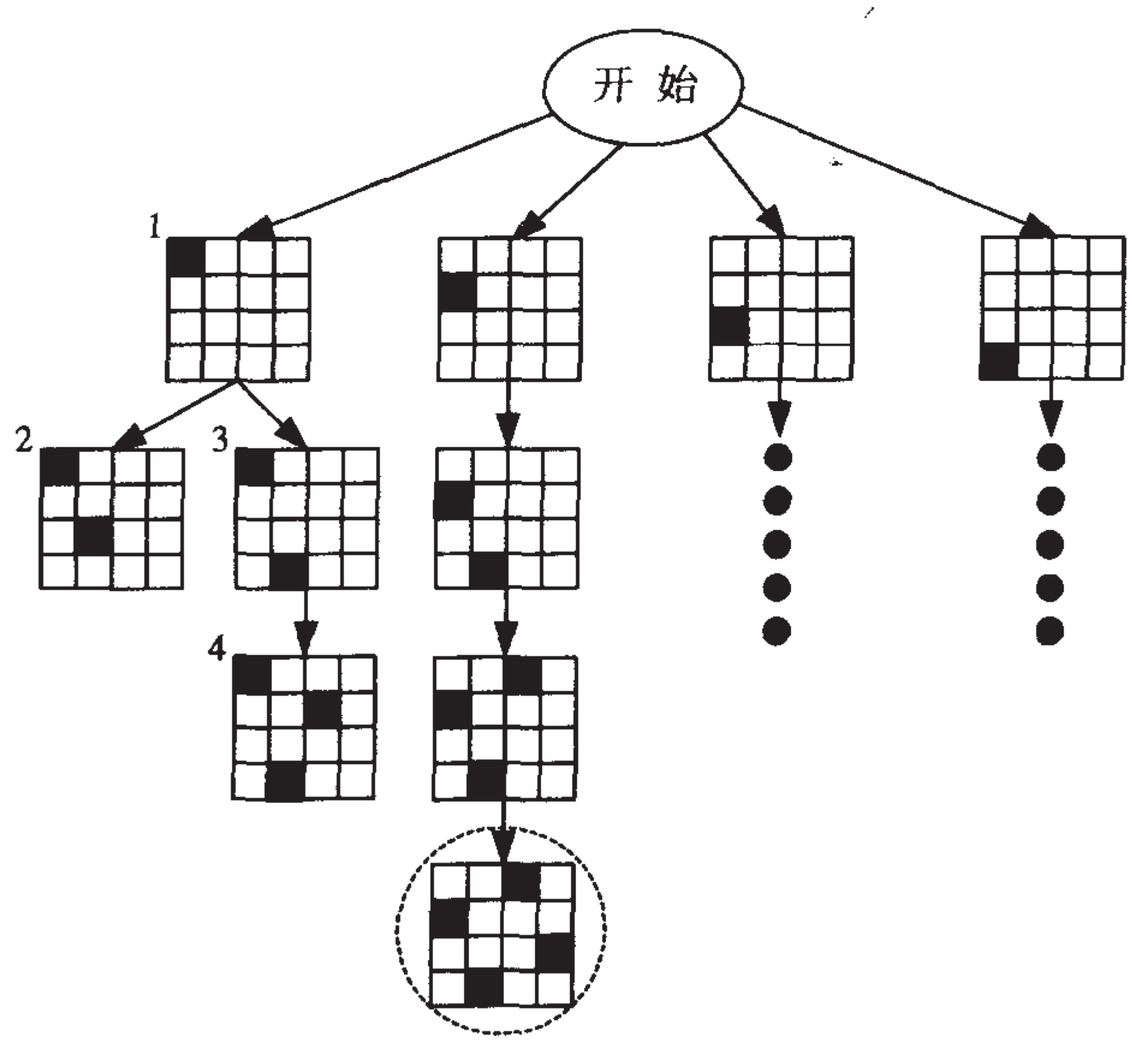


图 8-6

以 4×4 的棋盘为例，上面就是工作过程。在同一列上的棋盘就是在同一层次中的递归。每一个 S 集合都是{1,2,3,4}，表示棋盘上的 4 列，而“可以再进一步？”的意义则是向右移一行。在开始的时候，主程序或其他程序给上面的片段的信息是“从第一行开始”，因此就到达 1 的所在，自然(1,1)这一格没有问题，而且还可以右移，于是就调用自己去处理下一行，这就进到 2。进入第二层递归之后，S 还是{1,2,3,4}，但 1 不能用，因为与上一层所放位置的棋子互相干扰；2 也不能用，因为在对角线，但 3 可以了，因而把棋子放在(3,2)位置，接着再调用自己，把第三行交给第三层的递归去处理。

Download at http://www.pin5i.com/

第三层递归在试完 1,2,3,4 之后发现无一可用，因此回到第二层，这就是 IF…THEN…ELSE 的下一行叙述所在，把原来放下去的数据——亦即(3,2)定成没有使用，再取出 S 中的下一个元素；由于原来是 s=3，现在就去试 s=4，这就是图 8-6 中标了 3 的地方。接着把棋子放到(4,2)所在，又调用下一层（第三层）递归，得到 4 位置的结果，但是从该位置进入第四层递归时，S={1,2,3,4}中就没有可用的元素了，所以就倒退回第三层；第三层中 s=3,4 都不能用，前者与对角线(1,1)或(4,2)冲突，后者则与(4,2)冲突，但因为这两个都试完了，所以只能回到第二层。在第二层，s=4 也是最后一个，因而回到第一层。在第一层，现在 s=1，不妨试下一个 s=2；把上面的动作再反复，就会得到一个解答；其他的部分在图中没有画出来，但相信应该有能力完成它。

现在，看完此例，请将这个观点转化成程序。要如何测试不在同一列、同一行、同一（正、反）对角线呢？

问题 8.6 武士巡逻（KNIGHT.C）

在西洋棋中的武士（Knight）与象棋的马相似，走的都是 L 形的路，请写一个非递归的程序，输入一个表示棋盘的大小值 n，再输入一个起点的坐标，找出一条从第一格起可以让武士棋走完 n^2 格而不重复的路径。请问，找出的路径会回到起点吗？什么时候一定做不到？

【说明】

如图 8-7 所示说明了武士棋的可能走法。

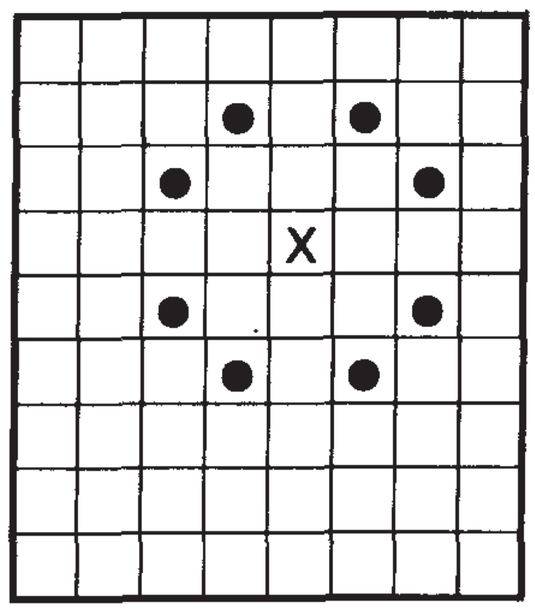


图 8-7

图 8-7 中×表示目前的位置，下一步则可以到达任何一个圆点的所在，只要它在棋盘上。如果输入一个起点，请问：从此点起，一个武士棋有没有办法把棋盘上的每一格都走遍而不重复。一个更难的问题是，武士棋能否回到出发点？这就是所谓的武士巡逻问题（Knight Tour Problem），但这个题目并不要求武士棋一定能够回到起点。

与 8 后问题一样，用回溯的技巧是不难写成一个递归程序的，但却不希望用递归方法，请试试看。

Download at http://www.pin5i.com/

问题 8.7　环游世界（HAMILTON.C）

假设有一张地图，城市与城市之间有道路相连，请写一个程序，在这份地图中找出一条把每一个城市都只走一次而不重复，并且还会回到起点的路线；如果找不出这条路线，也请报告这个现象。这个问题就叫做 Hamilton 循环（Hamilton Cycle）问题。

【说明】

地图如图 8-8 所示。

它有 10 个城市（1～10），很明显地它就有一条每一个城市都走一次，而且也只走一次的路径，不但如此，还会回到起点。不过并不是每一个图都有此等路径，下面就是这样的一个例子，如图 8-9 所示。

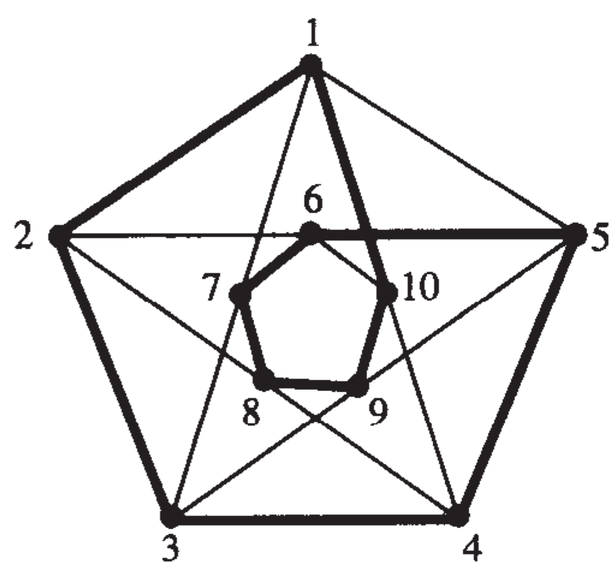


图　8-8

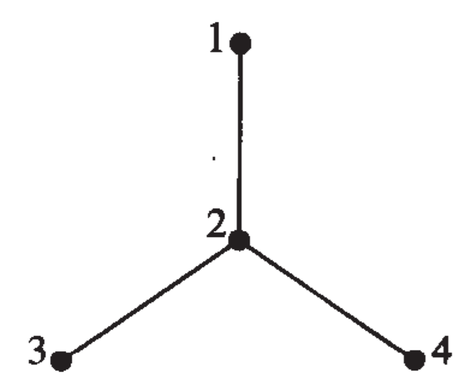


图　8-9

如果从 1 起到达 2 之后，不论是到 3 还是到 4，都无法回头到达另一个点。

这个问题并没有快速的解法，应该试一试用递归（或非递归）的回溯技巧。另外，假设一个合适的输入方式是很重要的；在本书的光盘中有两个文件 HAM.IN1 与 HAM.IN2，可以帮助进行测试，前者有题目中所提的路径，后者则无。文件的输入格式是这样的：第一列上是一个值，代表一共有多少个城市；往后的每列都有两个数，分别表示在一条路上两端的城市，最后用两个 0 作结束。于是，上面没有环绕一周的路径的图，就可以写成如下的输入：

```
4
1    2
2    3
2    4
0    0
```

还有一点值得提醒：在图 8-8 中若有这样环绕一周的路径的话，从哪一个城市出发实

Download at http://www.pin5i.com/

在没有差异；但若没有此等路径存在，从哪一个城市出发也都不会有的。在程序中仔细想想，用什么方法才能表示城市与城市之间相通的关系呢？

最后 HAM.IN1 与 HAM.IN2 如图 8-10 及图 8-11 所示。

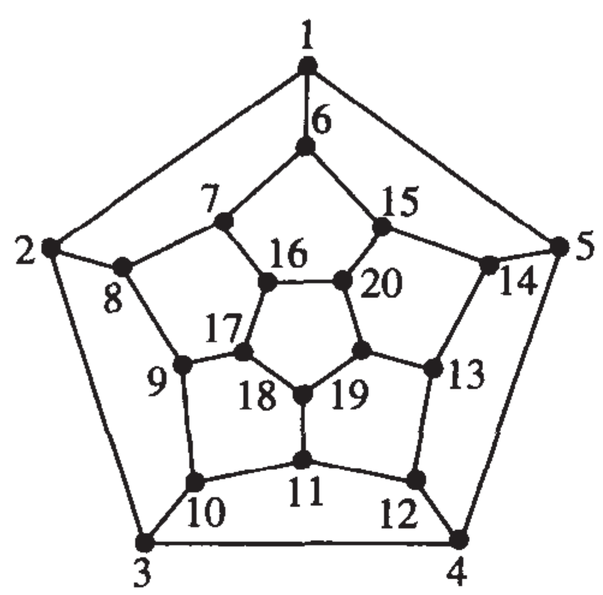


图 8-10

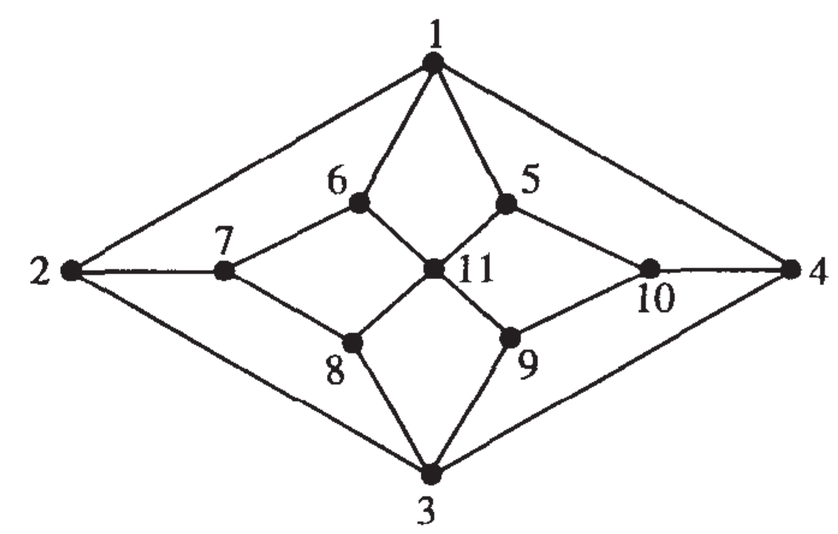


图 8-11

问题 8.8 一笔画（EULER.C）

请设计一个可以画一笔画的程序，在程序中应该想到要如何设计合理的数据结构，以及如何表示一笔画的动作。

【说明】

一笔画起源于 1736 年 Leonhard Euler 对 Konigsberg 7 桥问题的解答。东普鲁士古城 Konigsberg 被一条叫做 Pregel 的河横穿而过，因而为了交通需要而建了 7 座桥，它们的位置如图 8-12 所示。

城中的人一直想找出有没有办法把每一条桥都走过，但却都只经过一次。在 1736 年 Euler 的文章中，他把河的两岸、河中两个小岛用 4 个点代替，而把连接用的桥通过两点之间的线段来表示，如图 8-13 所示。

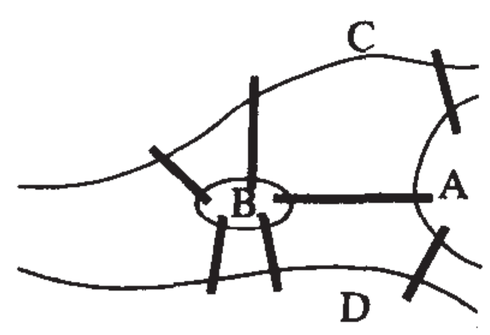


图 8-12

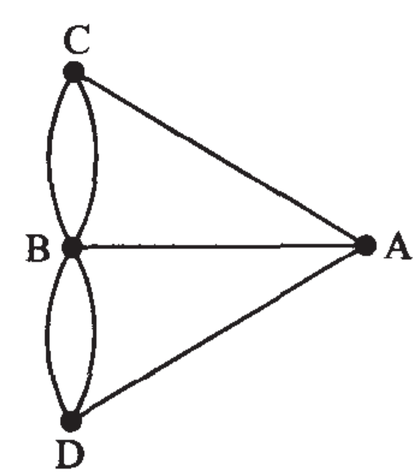


图 8-13

Euler 指出，研究这个图就足够了。对于一个点而言，如果有奇数（或偶数）条线经过它，这个点就叫做奇点（或偶点）。Euler 证明（其实并不完全）：

（1）如果图中没有奇点，那么就一定可以从任何一个点出发，走遍每一条边而回到起点，并且每一条边都只经过一次。下面就是一个例子，如图 8-14 所示。

Download at http://www.pin5i.com/

图中 A、B、C、G 有 4 条边经过，E 与 F 有两条边，D 有 6 条边，都是偶数，可以从任一点出发而回到起点，而且每一条边都只经过一次；比如 A→B→D→G→F→E→G→D→C→A→B→D→C→A 就是一种走法。

（2）如果图中有两个奇点（奇点一定有偶数个），那么从其中 1 个奇点出发，一定可以把每一条边都走一次而不重复，最后回到另一个奇点。图 8-15 就是一个例子，C 与 E 就是仅有的两个奇点，走法如下：C→D→F→A→E→C→B→A→D→B→E。

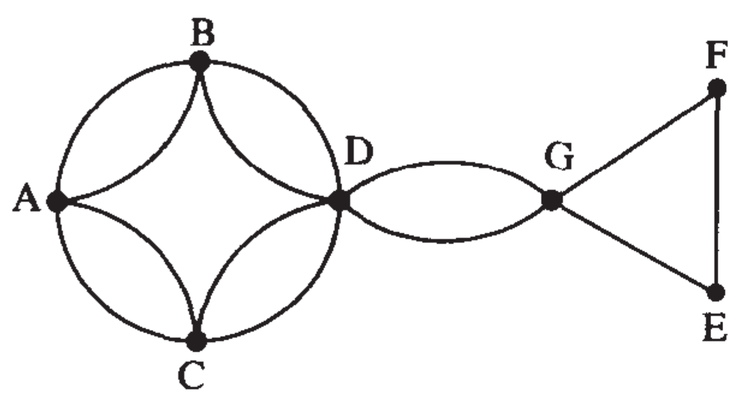


图 8-14

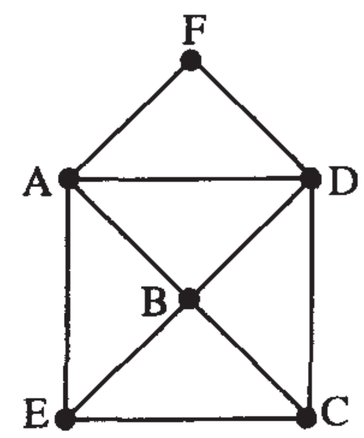


图 8-15

（3）如果奇点数目大于 2，就不可能有一笔画的走法。正因为如此，7 桥问题就是不可解的，因为 A、C 与 D 各有 3 条边，而 B 有一条，因此全都是奇点。

有多少条边经过某一个点，通常都叫做那一个点的次数。当走一笔画时，如果经过一个次数是偶数的点，那么进入那一个点之后是一定有一条边可以离开的，原因就是进入该点用一条边，但该点是个偶点，用了这一条边之后就是个奇点，所以就一定会有一条边可以出去（1,3,5,7…是奇数）。这是在写作程序时的一个有用的观念。但是若进入的是一个奇点那就有困难了，因为进入时用了一条边，剩下来的就可能有 0（0,2,4,…算是偶数）条边，就无从离开。好在因为只有两个奇点时才能画一笔画，所以从其中一个奇点走到另一个奇点去之后，两者各少了一条边，剩下来的不就全都是偶点了吗？

请用这个观点来编写程序；在程序中或许要有一个数据结构，记录点与点之间的连接情况，而且，也要记录下每一个点的次数，另外参考一下 HAMILTON.C 这个找环游世界的路径的程序或许会有点帮助。

问题 8.9　非递归河内之塔（HANOI_I.C）

河内之塔（Towers of Hanoi）是一个在入门书籍中常见的例题或习题，它是说：有 3 根柱子，1、2 与 3，在柱子 1 上串了从上到下编号是 1,2,…,n 的圆片，号码小的圆片也小。问题是，请写一个程序，把柱子 1 上的圆片搬到柱子 3 去。在搬的时候有 3 个要求：第一，每次只能搬一个圆片；第二，要搬的圆片得从某个柱子取出，并且放到另一根柱子上；第三，任何时刻、任何柱子上的圆片，从上到下都是从小到大排列。书上的解法都是递归的，请写一个非递归且不用堆栈来仿真的程序。

Download at http://www.pin5i.com/

【说明】

如果用教科书中的解法，那么递归是个非常好且效率非常高的技巧，程序大致如程序 8-2 所示。

【程序】

程序 8-2

```
void tower(int n, int start, int mid, int end)
{
   if (n == 1)
        print("Move disk %d from %d to %d",n, start, end);
   else {
        tower(n-1, start, end, mid);
        print("Move disk %d from %d to %d",n, start, end);
        tower(n-1, mid, start, end);
   }
}
```

这个观点是，先把在 start 柱子上的 n-1 个圆片搬到 mid 柱子上去（用 end 柱子作中继站），于是在 start 柱子上就留下 3 在最下方，也就是最大的一个圆片第 n 号，把它从 start 搬到 end 去（见 printf）；现在的情况是，最大的已经到了目的地，但在上方的第 1～n-1 号还停留在 mid 柱子上；第三步就是把 mid 上的 n-1 个圆片搬到 end 柱子上，而用 start 柱作中继站。当然，start、mid、end 就是题目中的 3 根柱子，n 是要搬的圆片数目。

递归的解不但漂亮，而且容易懂；不过了解了递归解法之后，能够由它而发展出一个非递归的解吗？当然，用堆栈来仿真并不是所期望的，应该把递归动作中在什么时候搬那一个圆片，从何处搬到何处这一层关系弄清楚，那么问题就不难了。

问题 8.10 生命游戏（LIFE.C）

请写一个模拟生命游戏（Game of Life）的程序；如果不知道所谓生命游戏，请看下面的说明。

【说明】

生命游戏是一个很简单，但却是很有趣的程序习题。在一个四周都可以延伸到无限的棋盘上的某个格子中会有一个有机体。每一个有机体在时间 t 时，会依照环绕着它的 8 个邻居的特性而决定在时间 t+1 时是否能生存下去。如果某一格在时间 t 时：

（1）有一个有机体，但是它的邻居少于或等于 1 个，或者是大于 3 个，那就会因为不够稠密或太过稠密，这个有机体在时间 t+1 时就会死亡；换言之，在 t+1 时间，那一格中不会存在有机体。下面就是几个在时间 t+1 时会死亡的例子，如图 8-16 所示。

Download at http://www.pin5i.com/

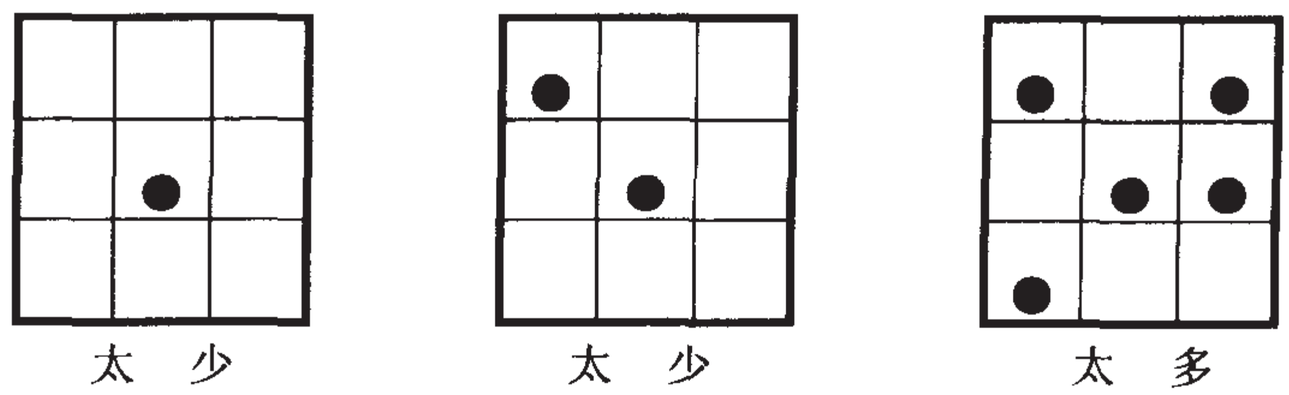


图 8-16

（2）有有机体在该处，当只有 2 个或 3 个邻居时，就可以继续生存。下面就是几个可以生存的例子，如图 8-17 所示。

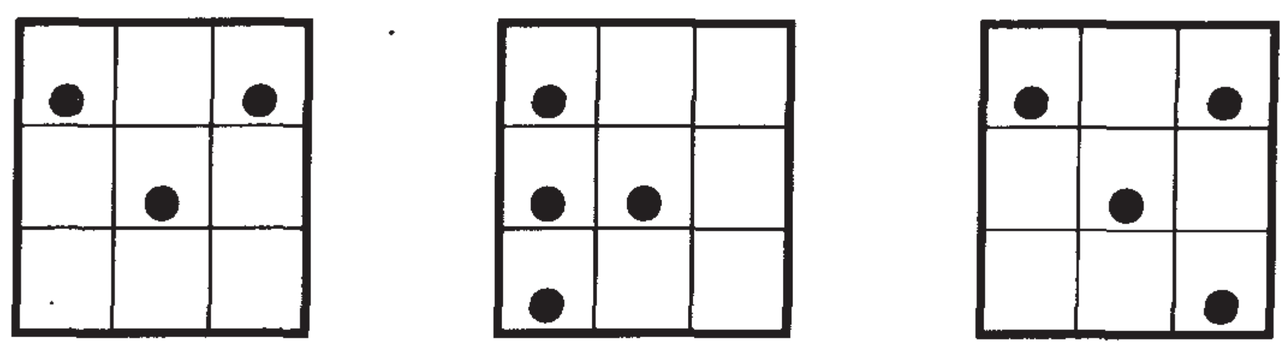


图 8-17

（3）如果没有有机体，而那一格恰好有 3 个邻居时，当时间为 t+1 时，那一格就会生出一个有机体。下面就是几个可以从“无”到“有”的例子，如图 8-18 所示。

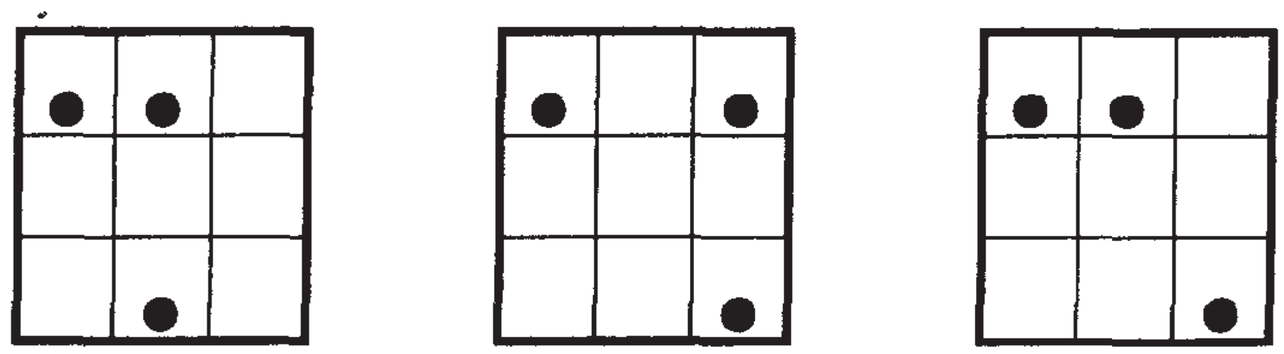


图 8-18

（4）其他情形不会造成新的有机体出生。

程序要输入棋盘的大小（毕竟内存有限，不能表示无限的棋盘，不是吗？），并且输入最大的时间值，比如 N，然后输入一个在开始时的有机体分布情况，最后显示出时间为 2,3,…,N 时棋盘上的状态。

一般而言，最终的结果不外乎 4 种。第一种是不到时间 N 时，所有有机体全都死光；第二种是到了某个时间，棋盘就不再发生变化，生者生存，这就叫做稳定（Stable）状态；第三种则是到了某个时间之后，棋盘上的分布方式就产生循环，也就是说，棋盘上有某几种状况周而复始地以周期性方式出现；第四，也就是最后一种，它不是上述三种情况之一。程序应该可以查出在时间 N 之前棋盘进入稳定状态，或者是有机体全都死，第三种情况是不容易找出来的（为什么？）。

Download at http://www.pin5i.com/

第9章 终　　曲

问题 9.1　等量正负号段落（BALANCE.C）

已知一个数组，其中有正数、0、负数，请写一个函数，找出这个数组中最长的而且正数与负数个数相同的部分数组的长度。

【说明】

在数组中的 0 既不是正数，又不是负数。如果数组的元素为 1、1、-1、-2、0、1、3、-1、-2 与 1 这 10 个数，于是前 9 个数的这一段就是所要的，它有 4 个正数、4 个负数。注意，它不可能是后 9 个数，负数比正数少了一个；自然也不可能是整个数组，因为数组中负数的个数比正数个数多 1。

对初学程序写作的朋友而言，这个题目并不难，因为他们多半会想到用两个矩阵 ***P*** 与 ***N***，P_{ij} 与 N_{ij} 表示输入数组中 i～j（含）的正数个数与负数个数，求得 P_{ij} 与 N_{ij} 并不难，把输入数据从 i 起到 j 只查一遍就行了。求出 ***P*** 与 ***N*** 之后，就把 P_{ij} 与 N_{ij} 一个接一个地比过去，一旦发现 $P_{ij}=N_{ij}$，就把 P_{ij}（或 N_{ij}）的值与已经记住的极大值相比，并且更正极大值的内容，当所有 P_{ij} 与 N_{ij} 都比完之后，就有了 $P_{ij}=N_{ij}$ 的极大值，这就是该部分数组的长度了。然而这个方法却很慢，应该是入门者的写法。一般的程序员马上就可以看出 ***P*** 与 ***N*** 只要一半，上半或下半三角形都行；有一点门道的或许用 ***P*** 或 ***N*** 就足够了，因为可以令 P_{ij} 是正数个数与负数个数之差，到时查 P_{ij} 为 0 的地方；再好一点的程序员或许发现两列就足够了。您是哪一类呢？

提示：任何人在想一个程序时往往是越想越简捷的，上面已经给了很好的提示，重点就在于如何记录正数与负数个数的改变。

问题 9.2　寻找长方形（RECT.C）

在圆周上有若干个点，已经知道这些点与点之间的弧长，依圆弧顺序排列，请写一个程序，找出这些点中有没有 4 个可以围成长方形，为了方便计算，弧长均为正整数。

【说明】

在图 9.1a 中，弧长是 1、3、1、1、4、1、2、1、2、2、2，很明显地可以得到若干长方形，不过图中只画出了两个而已，事实上，它一共有 6 个。不过在图 9-1b，却没有办法

Download at http://www.pin5i.com/

找出 4 个点围成长方形。程序的任务，就是先从简单的开始，回答“有”或“没有”就足够了。

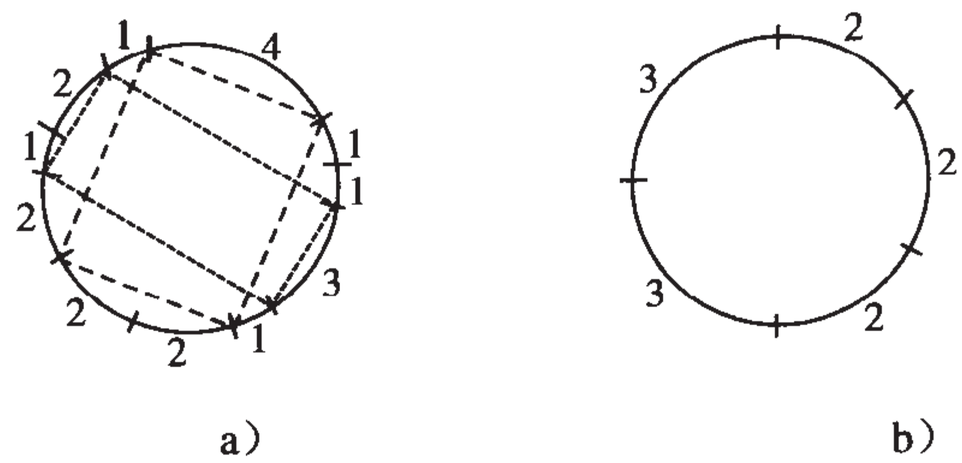


图 9-1

注意，因为给的是弧长，所以“点在圆上”这一事实是不重要的，程序只需要在弧长上动动脑筋就可以了，与圆无关。事实上，有很多问题是可以转化成与原来的面目完全不同的。那么，这个问题要转换成哪一个熟悉的技巧呢？如果把本书前面的题目都浏览或做过一次，就应该发现有相似的题目。另外，这不是一个所谓计算几何学（Computational Germetry）的问题，因为它简单到只能放在此地当作一个小题目做做而已——如果看得出诀窍的话。

问题 9.3　多边形的直径（DIAMETER.C）

假设平面上的点 $p_0, p_1, p_2, \cdots, p_{n-1}$ 围成一个多边形，$\overline{p_0p_1}, \overline{p_1p_2}, \cdots, \overline{p_{n-2}p_{n-1}}, \overline{p_{n-1}p_0}$ 为这个 n 边多边形的边。这个多边形的直径（Diameter）的定义是这样的：找出多边形的两个点 p_i 与 p_j，$0 \leqslant i < j < n$，使得 $\overline{p_ip_{i+1}} + \overline{p_{i+1}p_{i+2}} + \cdots + \overline{p_{j-1}p_j}$ 与 $\overline{p_jp_{j+1}} + \overline{p_{n-2}p_{n-1}} + \overline{p_{n-1}p_0} + \overline{p_0p_1} + \cdots + \overline{p_{i-1}p_i}$ 的差的绝对值为最小，于是 p_i 与 p_j 之间的连线就是该多边形的直径。当然，直径可能不止一条。把 $\overline{p_0p_1}, \overline{p_1p_2}, \cdots, \overline{p_{n-2}p_{n-1}}, \overline{p_{n-1}p_0}$ 的长度记成 $d_0, d_1, \cdots, d_{n-1}$ 等，找出两个点 i 与 j，$0 \leqslant i < j < n$，使得 p_i 与 p_j 的连线为该多边形的直径。

【说明】

如图 9-2 所示，各边的边长都已经标在边线旁边。

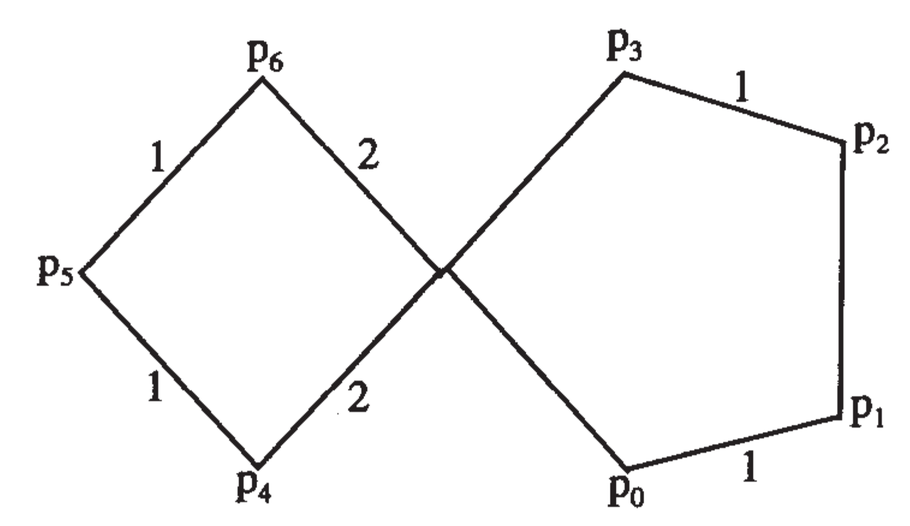


图 9-2

Download at http://www.pin5i.com/

很明显地，p_0,p_1,p_2,p_3,p_4 这一段折线的总长为 5，而 p_4,p_5,p_6,p_0 这一段为 4，差的绝对值为 1，因此 p_0 与 p_4 的连线是一条直径；同理，p_1 与 p_5 的连线也是直径，因为 p_0,p_1,p_2,p_3,p_4 这一段折线长度为 5，而 p_5,p_6,p_0,p_1 的长度为 4，差的绝对值也是 1；另外 p_1 与 p_4 的连线也是直径。还有其他的直径吗？自己找找看。因此，当程序收到输入 1、1、1、2、1、1、2 时，至少要能报告出 p_0,p_4；p_1,p_5；p_1,p_4；…这些直径其中的一组。

提示：在此之前已经做过许多个类似的题目了，这个题目并不难。首要工作，就是看看如何表示差的绝对值；也要注意到，两个决定直径的点把多边形分成两部分，这两部分边的总长正好就是整个多边形边长之和。

问题 9.4 机器人旋转角度（TURNS.C）

平面上有 n 个点，$p_1,p_2,\cdots,p_n$，有一个机器人目前放在 p_1 上，而且面向 X 轴的正方向，这个机器人会从 p_1 出发，沿途依次经过 $p_1,p_2,\cdots,p_n$ 这些点，最后回到 p_i 并且面向 X 轴的正方向。当机器人到达某个点 p_i 时，它会以逆时针方向旋转，使自己面对 p_{i+1}，再沿直线方向从 p_i 走到 p_{i+1}。编写一个程序，读入 $p_1,p_2,\cdots,p_n$ 的坐标值，求出这个机器人总共做了多少次 360° 的旋转。

【说明】

下面是一个有 4 个点的例子，如图 9-3 所示。

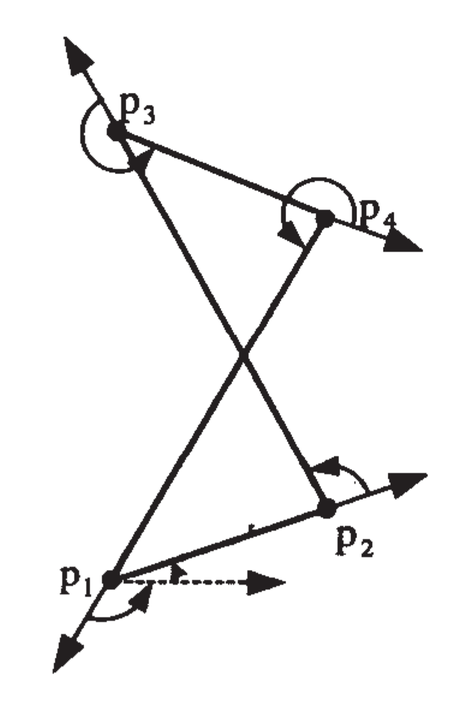


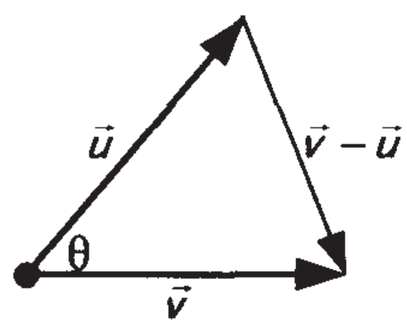
图 9-3

从图 9-3 可以看出机器人一共转了两次 360° 的旋转；图中的箭头指出机器人在 p_i 时所面对的方向，而弧型的箭头则指出旋转方向与大小。可以发现，从 p_1 到 p_2 面对 p_3 时，旋转度数还不到 180°，到了 p_3 要转成面对 p_4 时就做了一次大于 180° 的旋转，不过总度数还是不足 360°，一直到达 p_4 而旋转成面对 p_1 时才是第一次超过 360°，然后到了 p_1 而旋转成面对 X 轴时，正好满足两个 360°。注意，所有旋转都是逆时针方向。

有些朋友必定会觉得这个题目很容易，因为沿 p_i 到 p_{i+1} 的方向是 $\vec{u}=p_{i+1}\rightarrow p_i$，而从 p_{i+1} 到 p_{i+2} 的方向是 $\vec{v}=p_{i+2}\rightarrow p_{i+1}$，于是求这两个向量（Vector）$\vec{u}$ 与 $\vec{v}$ 的夹角，就会知道

Download at http://www.pin5i.com/

在 p_{i+1} 这个点转多少度，由此在每一点计算一次，且累计下来，这样就可以求出旋转的总度数，再用 360° 去除，不就求出有多少个 360° 的旋转了吗？不错，想法完全正确，但是却太慢了，因为要算出 $\vec{u}$ 与 $\vec{v}$ 的夹角，先要算出 $|\vec{u}|$、$|\vec{v}|$ 与 $|\vec{v}-\vec{u}|$，再用余弦定律才能求出夹角 θ，如图 9-4 所示。



$$\theta = \arccos\left[\frac{|\vec{u}|^2 + |\vec{v}|^2 - |\vec{v}-\vec{u}|^2}{2|\vec{u}||\vec{v}|}\right]$$

图 9-4

因此，$\vec{u}$ 与 $\vec{v}$ 之间的旋转角度就是 θ 或 $2\pi-\theta$，视旋转方向是从 $\vec{v}$ 到 $\vec{u}$ 还是从 $\vec{u}$ 到 $\vec{v}$ 而定。用了多少个运算呢？连三角函数都用上了，当 n 很大，而且又在没有算术处理单元的机器之下运行时，速度之慢可想而知。

所以，要求是，把问题仔细想一想，用最省时间的办法把程序写出来。特别是，当坐标都是整数时（比如荧光幕上的点坐标），程序能够不用实数，而单靠整数进行运算吗？

问题 9.5　最大涵盖距离（MAXCOVER.C）

已知一个数组 x[ ]，说 x[j]是 i～j 之间的极大点，指的就是对于任一在 i 与 j 之间（含 i 与 j）的 k 而言，如果 x[k]小于等于 x[j]的绝对值 k，则 x[j]的绝对值涵盖（Cover）住[i,j]这个区间。请写一个程序，找出最长的、被涵盖的区间。

【说明】

如果数组的元素是 1,-3,2,1,-2,3,5,2,-4,3。对于第一个 1 而言，它只涵盖了自己，故长度为 1；对于-3 而言，它的绝对值为 3，自然地就涵盖住 1 与-3，故长度为 2；对于 2 来说，1,-3,2 都小于 2，故涵盖的长度为 3。如果继续看下去，5 所涵盖的长度最长（值为 7），涵盖住 1,-3,2,1,-2,3,5。请问要如何编写一个高效率的程序呢？

事实上这是一个非常简单的程序，建议从右往左处理比较合理些，原因是被涵盖住的区间是在涵盖它的元素的左边，因此往左找不到比较合理。程序主体部分应该要做到 10 列之内写完，不要有多余的部分。

这个题目最好从右往左处理，根据这个提示，最有可能会想到下面的写法，它从右向左处理，而且也不足 10 列，如程序 9-1 所示。

【程序】

程序 9-1

```
for (i = n-1, max_len = 1; i >=1; i--) {
    for (j = i-1; j >= 0 && x[j] <= abs(x[i]; j--)
```

Download at http://www.pin5i.com/


```
        max_len = MAX(max_len, i-j);
    }
```

但是这个程序却不够快（虽然够短），如果 x[ ]中的元素是从小到大排好了的，对 i 而言，那么中间的循环就会绕 i 圈，因为 i 可以为 1～n-1，所以循环就一共绕了 1+2+…+(n-1)=n(n-1)/2 圈，程序就变为与 n^2 成正比的速度了。并不期望做出这样的结果，事实上希望做到与 n 成正比。

问题 9.6 最大连续元素和（MAXSUM.C,MAXSUM1.C）

已知数组 x[ ]储存了一组整数，请写一个程序，找出在数组中连续元素的和中最大的一个。举例而言，如果有数组 1,2,-6,3,-2,4,-1,3,2,-4，那么连续的元素的和有 1+2=3，1+2+(-6)=-3，2+(-6)=-4，…，但值最大的就是 3+(-2)+4+(-1)+3+2 这一段，值为 9。规定：和为负值时就定成 0，所以结果永远不为负。这个题目通常叫做最大连续元素和（Maximum Consecutive Sum）问题。

【说明】

这个问题有个很简单的解法，但效率很低，一般采用效率很高的方法的程序比使用不好的方法的要长。

一般人看到这个问题的自然反应就是用两个循环，如程序 9-2 所示。

【程序】

程序 9-2

```
For ( i = maxsum = 0; i < n; i++) {
    sum = 0;
    for (j = 1; j >= 0; j--) {
        sum += x[j];
        if (sum > maxsum)
            maxsum = sum
    }
}
```

想法是很简单的，当固定在某个 x[i]之后，把 x[i],x[i-1]+x[i],x[i-2]+ x[i-1]+x[i],…,x[0]+x[1]+…+x[i]求出来，一边算，一边找出极大值，这就是上面片段所隐含的意义。

但是，当处理到 i 时要查 i 个元素，所以要把 n 个数都处理完，就要查 1+2+3+…+i+…+n=n(n+1)/2、与 n^2 成正比的元素了。如果仔细看就会发现做了很多重复的工作，为什么？当 i 是 2 时，算了 x[2],x[1]+x[2],x[0]+x[1]+x[2]，但当 i 进到下一个位置变成 3 时，又算 x[3],x[2]+x[3],x[1]+x[2] +x[3],x[0]+x[1]+x[2]+x[3]，于是 x[1]+x[2]与 x[0]+x[1]+ x[2]不都是重复算了吗？因此，程序如果能够避免这些重复的部分，就一定会非常快。

Download at http://www.pin5i.com/

问题 9.7　最大连续元素积（MAXPROD.C）

一个整数数组中有正数、负数与 0，请写一个程序，找出这个数组中连续的元素相乘的极大值。

【说明】

这是问题 9.6 的一个变化，不过却要多点心思才行，原因就是它可能有负数，而且也可能有 0。为了方便起见，如果找不出正的极大值时，程序就返回 1 作极大值；其实这是用来化简程序用的，自然可以不必这么做。于是，在这个条件下，若数组元素是 1,0 时，极大值就是 1；同理，若元素为 0,0 时（元素自然可以重复），极大值也是 1。那么若元素为 1,-1 时，极大值又是多少呢？还是 1，原因就不是上述的条件了，而是(-1)×(-1)=1。

提示：注意负数与正数相乘会变小，但负数与负数相乘却会变大的事实。

问题 9.8　寻找名人（SINK.C）

假设有一群人，而您是新闻记者，想要找出这群人中的“名人”。何谓名人？就是人家都认得他，而他却不认识别人的那一个人。您在访问时只能用这样的问题：当您访问张三时，您只能问他“您认不认得李四？”、“您认不认得王五？”等。换句话说，您在访问 A 时（A 可以任您挑选，要访问多少个 A 与问 A 多少个问题都行）只能问 A：“您认不认得 X？”。当然，被访问者只会回答“认得”与“不认得”，而不会有其他答案。请问，您从访问而得的答案中要如何找出谁是名人，或这一群人中根本没有名人呢？这个问题通常叫做名人问题（Celebrity），或水槽问题（Sink）。

【说明】

为了做题方便，假设这一群人是 1,2,3,…,n，又用一个矩阵 k[ ][ ]来记录谁认识谁。例如，k[i][j]是真时，就表示 i 认得 j。注意一点，i 认得 j 并不表示 j 也认得 i，就像是我认得我们主席，但我们的主席肯定不认得我一样。因此这位新闻记者的工作，就是从 k[i][j]的关系找出谁是名人，或者是根本没有名人。

回想一下，“名人”者，就是他被所有人认得，但不认得所有人；如果 i 是名人，因为他不认得所有人，所以 k[i][ ]，也就是第 i 列全是“假”。注意，不管自己认不认得自己，为了方便起见，定成不认得是没有妨害的。但每个人（除了名人自己之外）都认得名人，所以第 i 行 k[ ][i]的元素（除了 k[i][i]之外）都是“真”。因此有了 k[ ][ ]矩阵之后，可以查是否有某列 p 全是“假”，而第 p 行上除了 k[p][p]之外全是“真”，于是 p 就是名人。不过如果这样做的话就至少要问 n^2 个问题，可要口干舌燥的；事实上，问与 n 成正比（比如说不超过 3n）个问题就足够了。如果 n=100，n^2=10000，但 3n 不过是 300，相差很大。

Download at http://www.pin5i.com/

请用这个想法写一个足够快速的程序，但不要问 n^2 个问题！另外，这个问题也是一个很好的例子，有时用不着把所给数据全部用到就可以解出问题的。

问题 9.9 投票问题（VOTING.C）

这个问题一般叫做过半数（Majority）问题，也有人称为投票（Voting）问题，但以前者的说法居多。假设有 1,2,…,n 等 n 个人参加投票，他们只能圈选一个人，但是却可以选任何一人，甚至于可以选不在这 n 个人中的另一个都行。例如，如果 5 个人投票，分别投给 1、8、1、100、1；1、3、5 都投给 1，2 投给 8，4 投给 100。问题是，写一个程序接收这些投票结果，看看有没有人的得票过半数。就以上例来看，1 的得票数就过半数了。

【说明】

这是个名题，也与其他问题一样有简单但效率低的解法，但也有较有技巧且效率高的做法。简单做法是，把投票结果先从小到大排好，把投给同一个人的票就收在一起，接着从头查起，看看一连串相同的值的个数，如果有过半数的，那个人就当选了。用上面的例子来看，从小到大排好是 1,1,1,8,100，很显然 1 有 3 个，过半数了，因此 1 会当选。

这是一个大家都会想到的想法，但事实上，排大小的这一步是不必要的，能开发出这样的程序吗？不过要记住一点，不能对候选人的数目作任何假设，因为如果假定候选人是 1,2,…,k，那么这个题目就简单至极而不会变成名题了。为什么呢？准备一个有 k 个元素的数组，然后……（一定马上就知道是怎么回事了吧！）。

问题 9.10 寻找 1 对 1 函数（1TO1.C）

如果 $S=\{0,1,2,3,\cdots,n\}$ 是一个集合，$f:S\rightarrow S$ 是个从 S 映射到 S 的函数，请写一个程序，找出 S 的一个部分 S'集合，使得 $f:S'\rightarrow S'$ 是一个 1 对 1 的映射函数，而且 S'的元素个数为最多。

【说明】

这是一个很不好说明，但却很有意思的题目。用一个例子来加强印象。例如，$S=\{0,1,2,3,4,5,6\}$，f 函数为 $f(0)=2$，$f(1)=f(2)=0$，$f(3)=f(4)=4$，$f(5)=3$，$f(6)=5$，如图 9-5 所示。

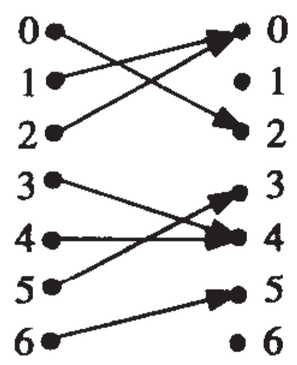


图 9-5

Download at http://www.pin5i.com/

令 S'={4}，那么 f：S'→S'是一个 1 对 1 的函数；但若令 S'={0,2,4}，f：S'→S'也是个 1 对 1 函数，不过后者有 3 个元素，所以程序的输出就是{0,2,4}，而不是{4}。

如何表示一个函数？可以用一个数组，令为 f[ ]，于是 f[0]、f[1]、f[2]、f[3]、f[4]、f[5]、f[6]中分别储存 3、1、1、5、5、4、6；要找出 f(i)时，使用 f[i]就行了。在报告结果时，既可以用另一个数组来存放 S'，也可以用一个数组记录 S 中哪一个元素要保留下来。程序并不十分难，而且还有一个非常快的写法。

问题 9.11　寻找支配元素（DOMINATR.C）

已知一个整数数组 x[ ]，对于任何一个元素 x[i]而言，在 x[i]右边，而且值不比 x[i]小的元素（亦即 j>i 而且 x[j]>=x[i]的 x[j]）都叫做 x[i]的右支配元素（Right Dominator）。在 x[i]所有的右支配元素中最靠近 x[i]的那一个，就叫做最靠近的右支配元素（Right Nearest Dominator）。当然，不见得每一个元素都有支配元素的，如最大元素就没有。问题是，编写一个程序，接收一个数组，把数组中每一个元素最靠近的右支配元素找出来。

【说明】

看个例子比较容易明了这个问题的要求。如果数组的内容是 2,1,3,5,4；那么 3,5,4 都是 2 的右支配元素，但 3 是最靠近的一个，所以 3 是 2 的最靠近右支配元素。同理 1 与 3 的最靠近右支配元素分别是 3 与 5。5 与 4 都没有最靠近右支配元素；对 5 而言，右边没有元素比它大；就 4 而言，右边没有元素了。

看起来程序并不难写，因为可以 x[1],x[2],…,x[i],…一个接一个查过去；当查到 x[i]时，就查 x[i+1],x[i+2],…，于是第一次找出比 x[i]大或相等的元素，就是 x[i]的最靠近右支配元素了。但是这样的做法效率并不高。举例而言，如果一个数组的元素是从大到小排好的，但是最后一个元素是整个数组的极大值（例如，5,4,3,2,1,6）。若数组有 n 个元素，处理 x[0]时要比较 n-1 次，处理 x[1]时比较 n-2 次，…，一般而言，处理 x[i]时要比较 n-i 次。所以就一共用了：

$$(n-1)+(n-2)+\cdots+3+2+1=n(n-1)/2$$

差不多是 n^2 次的比较。

因此，挑战是，能不能写出比较次数与 n 成正比的程序？

问题 9.12　最长递增部分序列（LIS.C）

已知一个整数数组，在这个数组中的一个递增序列，就是把若干元素删除后所留下的元素按从小到大的顺序排列。举例而言，如果数组是 1,3,5,7,8,2,9,4,10,6，那么 1,5,8,9,10 是一个递增序列，3,5,7,8,9,10 也是一个递增序列，1,3,5,7,8,9,10 也是递增序列。简单地说，

Download at http://www.pin5i.com/

可以依数组元素的次序选出若干元素，使得从小到大排好，那么这就是个递增序列。例如3,8；2,9；7,10；2,4,6；…。但是 9,4,10；5,7,8,2,9 等却不是递增序列。问题是，写一个程序接收一个整数数组，找出这个数组中最长的递增序列的长度。以上面的例子来看，1,3,5,7,8,9,10 是最长的递增序列，所以程序输出 7 的结果。

【说明】

递增序列的意义已如上述，程序要如何写？看起来似乎很难下手，但总是有迹可寻的；为什么呢？递增序列不会凭空出现，一定是先有了几个，然后又出现一个新元素，看看这个新元素能不能加到已有的递增序列后面，如果可以，已有的递增序列的长度就多了 1；但若不能添加，那要做些什么事呢？那就要自己想了。

先看个例子，还是以 1,2,4,3,6 为例。假设目前已经发现了如下的几个递增序列：

1, 2, 4, 6 （长度为 4）

1, 2, 3 （长度为 3）

现在新来的元素是 5，因此 5 可以补在 1,2,3 后面而得到 1,2,3,5（长度为 4）；那么 5 对 1,2,4,6 的关系又如何？这就要自己想了。

这个问题的起源可能是组合数学（Combinatoric Mathematics）。组合数学中有一个定理，它说 n^2+1 个数中一定有一个长度至少为 n 的递增或递降序列。此处不过是要求写一个程序把最长的递增序列的长度找出来而已，会写这个程序之后，递降序列也不过是同一回事。

Download at http://www.pin5i.com/

第二部分　解　　答

第10章　序　　曲

问题 1.1　最长平台（PLATEAU.C）

已知一个已经从小到大排列好的数组，说这个数组中的一个平台（Plateau），就是连续的一串值相同的元素，并且这一串元素不能再延伸。例如，在 1,2,2,3,3,3,4,5,5,6 中 1,2.2,3.3.3,4,5.5,6 都是平台。试编写一个程序，接收一个数组，把这个数组中最长的平台找出来。在上面的例子中 3.3.3 是该数组中最长的平台。

【解答】

在程序中可以用一个变量来记录平台的长度（注意平台长度至少为 1）。又因为数组是已经排成从小到大的顺序的，所以如果现在设平台长度是 L，然后处理到第 i 个元素，如图 10-1 所示。

图　10-1

下面就比较相隔为 L+1 的两个元素，x[i]与 x[i-L]：如果它们相等，那么因为数组是排好顺序的，就可以确定 x[i]与 x[i-L]之间的元素完全相同，平台的长度就增加 1。就这个观点来看，平台长度是不断增加的，所以程序结束时，这就是最长平台的长度了。如果 x[i]与 x[i-L]不相等，就根本不必做任何事，因为这指出从 x[i]起往左算的平台长度不足 L。其他的部分可以就程序自行研究。程序中，变量 length 就是此处的 L。

【习题】

（1）改写程序，指出最长平台的位置。

（2）编写一个程序，接收一个已经从小到大排好的数组，把所有的平台与它的位置都找出来。

（3）如果给定的数组没有依序排好，请问这个程序会在什么地方出问题？请写一个程

Download at http://www.pin5i.com/

序可以找出所有平台与它们的位置，但给定的数组却不一定要排列好。程序还能够比较 n 次就找出答案吗？为什么？

（4）在 David Greis 的书中（第 301 页）提到，Michael Griffiths 曾经用递归的方法解过这个题目；试着也写一个递归的程序解决它。

【参考文献】

D.Gries. *The Science of Programming*, Springer-Verlag, 1981

【程序】

程序 10-1（PLATEAU.C）

```
/* ------------------------------------------------------------ */
/* FUNCTION longest_plateau :                                   */
/*   Given a sorted (increasing) array, this function           */
/* computes the length of the longest plateau.  A plateau       */
/* is a consective segment of an array with equal elements      */
/* For example, if x[] contains 1, 2, 3, 3, 4, 4, 5, 5, 5      */
/* and 6, then we have six plateaus, namely 1, 2, 3-3, 4-4      */
/* 5-5-5 and 6.  Therefore the length of the longest one        */
/* is 3.                                                        */
/*   This is the simplest program which I have ever seen.       */
/*                                                              */
/* Copyright Ching-Kuang Shene      June/30/1989                */
/* ------------------------------------------------------------ */
int longest_plateau(int x[], int n)
{
    int  length = 1;         /* plateau length >= 1.            */
    int  i;
    for (i = 1; i < n; i++)
        if (x[i] == x[i-length])
            length++;
    return length;
}
```

问题 1.2　支配值数目（GT_COUNT.C）

已知 f[ ]与 g[ ]两个整数数组，元素都已经从小到大排列，请写一个程序，算出 f[ ]中比 g[ ]中元素大的对数。换句话说，f[0]比 g[ ]中多少个元素大，f[1]比 g[ ]中多少个元素大等，这些值的总和就是要求的答案。

举个例子，如果 f[ ]中有 1,3,5,7,9，而 g[ ]中有 2,3,4,7,8。那么比 g[0]大的有 f[1]～f[4]，

Download at http://www.pin5i.com/

比 g[1]大的有 f[2]～f[4]，比 g[2]大的有 f[2]～f[4]，比 g[3]大的是 f[4]，比 g[4]大的是 f[4]，所以答案是 4+3+3+1+1=12。

【解答】

这个题目的诀窍只有一点：当发现 f[i]＞g[j]时，f[i],f[i+1],…,f[m-1]都会大于 g[j]，因为 f[ ]是从小到大排列的；此处 m 是 f[ ]的元素个数，所以，当 f[i]＞g[j]时，比 g[j]大的 f[ ]元素就有 m-i 个。

于是程序就从 f[0]与 g[0]比起，在比较过程中只有两种情况需要处理：

（1）f[i]<=g[j]：既然 f[i]小于等于 g[j]，可试着用 f[i+1]来比较。换言之，f[ ]的脚码加上 1。

（2）f[i]＞g[j]：这是满足要求的部分，所以在 f[ ]中就有 m-i 个元素比 g[j]大；接着比较 f[i]与 g[j+1]，看看 f[i]是否比 g[j]的下一个元素大。换句话说，把 g[ ]的脚码加上 1。

当 f[ ]或 g[ ]没有元素了，就可以停止工作。如图 10-2 所示就是比较数组 f[1,3,5,7,9,]与 g[2,3,4,7,8]的情况，中间横线指出进行比较的元素。

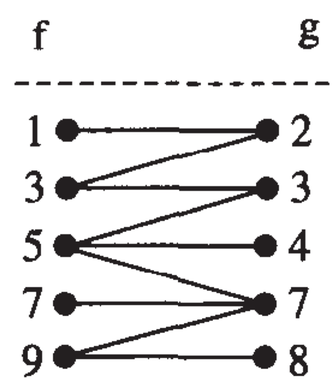


图 10-2

本题出自 M.Rem 的程序设计专栏，这是一个很有趣而且颇具启发性的专栏，有兴趣的朋友不妨下一点工夫。

【习题】

（1）请证明这个程序最多只会比较 2n 次，如果 f[ ]与 g[ ]都有 n 个元素。

（2）如果 f[ ]与 g[ ]中有相同元素时，这个程序还正确吗？为什么？

（3）修改程序，把在 g[ ]中小于每一个 f[i]的元素都显示出来。

【参考文献】

M.Rem. Small Programming Exercise 2. *Science of Computer Programming*, Vol. 3(1983), pp.313～319

【程序】

程序 10-2（GT_COUNT.C）

```
/* ------------------------------------------------------------------ */
/* FUNCTION dominance_count :                                          */
/*   Given two sorted (increasing) arrays, this function               */
/* computes the number of pairs that satisfies f[i]>g[j].              */
/* This function is similar to coincidence_count().                    */
```

Download at http://www.pin5i.com/


```
/*                                                                    */
/* Copyright Ching-Kuang Shene      June/30/1989                      */
/* ------------------------------------------------------------------ */

int  dominance_count(int f[], int g[], int m, int n)
{
     int  index_f, index_g;
     int  count;

     count = index_f = index_g = 0;
     while (index_f < m && index_g < n)
         if (f[index_f] <= g[index_g])
             index_f++;
         else
             index_g++, count += m - index_f;
     return count;
}
```

问题 1.3 等值数目（EQ_COUNT.C）

已知两个整数数组 f[ ]与 g[ ]，它们的元素都已经从小到大排列，而且两个数组中的元素都各自不同。例如，f[ ]中可能有 1,3,4,7,9，而 g[ ]有 3,5,7,8,10。

请写一个程序，算出这两个数组彼此之间有多少组相同的数据。就以上例而言，f[2]与 g[1]为 3 是第一组；f[4]与 g[3]为 8 是第二组。

【解答】

当比较 f[i]与 g[j]时，不外乎有下面 3 种情形之一：

（1）f[i]<g[j]，于是 f[i]与 g[j]就不是相等的一对，如果在 f[ ]中有与 g[j]相等的元素，那么一定在 f[i]的后面，亦即是 f[i+1],f[i+2],…之一，所以下一次应该比较 f[i+1]与 g[j]。注意：f[ ]与 g[ ]都已从小到大排好才有此结论。

（2）f[i]>g[j]，同理，下一次应该比较 g[j+1]与 f[i]。

（3）f[i]==g[j]，这样就找出了相等的一对，下一次应该比较 f[i+1]与 g[j+1]。注意：因为 f[ ]（或 g[ ]）中的元素都相异，才有这样的结论。

在开始时，当然比较 f[0]与 g[0]。当比较进行到有一方的元素都用完了，工作就可以停止了（为什么？）。例如，假设 f[ ]为 1,3,4,7,9；g[ ]为 3,5,7,8,10，下面的横线从上到下就表示比较的顺序，如图 10-3 所示。

程序中 coincidence_count()函数接收 f[ ]与 g[ ]两个数组，把相同的元素对应的数目当成函数值返回来；index_f 与 index_g 就分别是 f[ ]与 g[ ]数组的脚码；count 变量是用来计算到目前为止有多少对是相同的。

Download at http://www.pin5i.com/

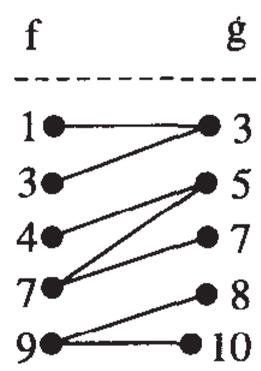


图 10-3

注意，f[ ]与 g[ ]中的元素都只查一次，虽然在查的时候可能比较不止一次；但是不论如何，f[ ]与 g[ ]中的元素被拿来比较的总次数绝对不超过 2n 次，为什么？自己想一想。所以，这是一个效率很高的方法。

【习题】

（1）请修改程序，把相同的元素的脚码显示出来。

（2）如果 f[ ]或 g[ ]中有相同的元素，但仍然是从小到大排列，这个程序还能正常运行吗？问题何在？修改程序，使得可以处理这个问题。例如，如果 f[ ]是 1,3,3,5,7 而 g[ ]是 1,3,3,8,9，于是就有 4 对 3 是相同的：f[0]与 g[0]，f[1]与 g[1]，f[1]与 g[2]，f[2]与 g[1]。程序的效率还是一样好吗？

（3）为什么一方的元素用完了就可以停止工作呢？

（4）请证明这个程序的比较次数绝对不会超过 2n 次，n 是 f[ ]与 g[ ]中的元素个数。

【程序】

程序 10-3（EQ_COUNT.C）

```
/* ------------------------------------------------------------ */
/* FUNCTION  coincidence_count :                                */
/*   Given two sorted (increasing) arrays, this function        */
/* determines how many equal pairs there are.  Note that        */
/* 'sorted' is a very strong requirement.                       */
/*                                                              */
/* Copyright Ching-Kuang Shene          June/30/1989            */
/* ------------------------------------------------------------ */

int coincidence_count(int f[], int g[], int m, int n)
{
    int  index_f, index_g;   /* subscripts for f and g          */
    int  count;              /* equal pair counter              */

    count = index_f = index_g = 0;
    while (index_f < m && index_g < n)
        if (f[index_f] < g[index_g])    /* if f[] is less       */
           index_f++;                   /* then move f          */
        else if (f[index_f] > g[index_g]) /* if f[] >           */
```

Download at http://www.pin5i.com/


```
            index_g++;                          /* then move g      */
        else
            count++, index_f++, index_g++;  /* EQUAL            */
    return count;
}
```

问题 1.4 两数组最短距离（MINDIST.C）

已知两个元素从小到大排列的数组 x[ ]与 y[ ]，请写一个程序算出两个数组元素彼此之间差的绝对值中最小的一个，这叫做数组的距离。

【解答】

注意一点，如果 x[i]与 y[j]的距离是 d，而且 x[i]>y[j]，那么排在 y[j]前面的元素与 x[i]的距离一定大于 d。为什么呢？如果 y[k]排在 y[j]前面，则 y[k]<y[j]，因此就得到 d=x[i] -y[j]<x[i]-y[k]。

当程序运行的过程中发现了 x[i]>y[j]，就可以确定 i 而不断增加 j，于是距离就会越来越小，一直到会变成 y[k]>x[i]为止。但是，反过来，又可以固定 k 而一直增加 i，距离又会下降，程序就是这样写成的。

引入文件<limits.h>中的 INT_MAX 指出 int 类型的极大值，用它来定出变量 minimum 的初值。

【习题】

（1）这个程序可以写成函数来帮助解下一个相等的前置和与后置和的问题，虽然解法不会很好。请改写程序试一试。

（2）如果有一个数组没依顺序排列，程序还能正常工作吗？如果两个都没排好又如何呢？请克服这个问题。

（3）请修改程序，找出造成最短距离的那两个元素的位置与它们的值。

（4）本题程序中的 if 段落与 else 都有两行叙述，如果分别把 index_x++与 index_y++写到它上面设定的叙述中，会有相同的结果吗？

【程序】

程序 10-4（MINDIST.C）

```
/* ------------------------------------------------------------------ */
/* FUNCTION min_distance :                                            */
/*   Given two sorted (increasing) arrays, this function              */
/* computes the minimum distance between these two arrays             */
/* by using the techniques employed in the previous                   */
/* examples.                                                          */
/*                                                                    */
```

Download at http://www.pin5i.com/


```
/* Copyright Ching-Kuang Shene      June/30/1989                    */
/* ------------------------------------------------------------     */
#include <limits.h>

#define  min(x, y)     ((x) < (y) ? (x) : (y))

int  min_distance(int x[], int y[], int m, int n)
{
    int  minimum = INT_MAX;      /* INT_MAX is from limits.h     */
    int  index_x = 0, index_y = 0;

    while (index_x < m && index_y < n)
        if (x[index_x] >= y[index_y]) {
            minimum = min(minimum, x[index_x]-y[index_y]);
            index_y++;
        }
        else {
            minimum = min(minimum, y[index_y]-x[index_x]);
            index_x++;
        }
    return minimum;
}
```

问题 1.5　等值首尾和（HEADTAIL.C）

假设有一个数组 x[ ]，它有 n 个元素，每一个都大于 0，称 x[0]+x[1] +…+x[i]是个前置和（Prefix Sum），而 x[j]+x[j+1]+…+x[n-1]则是后置和（Suffix Sum）。请写一个程序，求出 x[ ]中有多少组相同的前置和与后置和。

【解答】

既然不用额外的数组，前置和与后置和又不能省略而不算，就只能用两个指针，一个指针从头到尾走，并且记录下到目前为止的前置和；再用另一个指针从尾到头走，并且记录下到目前为止的后置和。接着，比较目前的前置和与后置和，如果相等，就得到了一组，然后增加往后走的指针，降低往回走的指针，继续工作。如果前置和大于后置和，这就表示要增加后置和才能比较长短，所以就降低往回走的指针，更新后置和，再继续；至于后置和大于前置和的情况则是相对称的，就省略了，请自行阅读程序。

【习题】

（1）这个程序要求所有元素都大于零，是否有此必要？如果没有这个条件，程序还能运行吗？如果可以，请证明它；如果不行，请改写程序。

Download at http://www.pin5i.com/

(2)请改写程序,把造成前置和与后置和相同的指针连同这个相等的值一并显示出来。

【程序】

程序 10-5 (HEADTAIL.C)

```
/* ----------------------------------------------------------------- */
/* FUNCTION head_tail :                                              */
/*   Given an array with positive elements, this routine             */
/* counts how many equal prefix and suffix sums there are            */
/* techniques employed here are all the same as the two              */
/* previous examples.                                                */
/*                                                                   */
/* Copyright Ching-Kuang Shene          June/30/1989                 */
/* ----------------------------------------------------------------- */

int  head_tail(int  x[], int n)
{
     int  prefix    = 0, suffix    = 0;
     int  prefix_idx = 0, suffix_idx = n-1;
     int  count = 0;

     while (suffix_idx >= 0 && prefix_idx <= n-1)
         if (prefix < suffix)            /* prefix too small     */
             prefix += x[prefix_idx++];
         else if (prefix > suffix)       /* suffix too small     */
             suffix += x[suffix_idx--];
         else {                          /* get an equal pair:   */
             count++;                    /* increase count and   */
             prefix += x[prefix_idx++]; /* advance pref.        */
             suffix += x[suffix_idx--]; /* and suffix           */
         }
     return count;
}
```

Download at http://www.pin5i.com/

第11章 数字问题

问题 2.1 Armstrong 数（ARMS1.C,ARMS2.C）

在 3 位的正整数中，比如 abc，有一些可以满足 $a^3+b^3+c^3=abc$ 的条件，也就是说，各个位数的立方和正好是该数的本身，这些数就叫做 Armstrong 数。试写一个程序求出所有的 3 位 Armstrong 数。

【解答】

至少有两个方法可以做出结果，两个做法大同小异，不同之处就是如何凑出一个 Armstrong 数而已。

第一个方法，用 number 来表示一个 3 位数，因此它的值就在 100～999 之间，所以用一个 for 循环来产生就是最佳选择。有了 number 之后，把它的 3 个位数取出来，令百位、十位、个位数分别是 p、q 与 r；接着再测试这 3 个数的立方和是否等于 number，如果是，就显示 number 的值；如果不是，就处理下一个 number。于是写成的程序就是 ARMS1.C。

第二个方法比较复杂些（但是距 UGLY.C 还差得远），输入 3 个位数，再由 3 个位数组成一个数，正好与第一个程序相反。3 个位数（百位、十位、个位）还是用 p、q 与 r 表示；另外，用 p100 与 q10 储存了 p*100 与 q*10 的值，换言之，如果 p 与 q 分别是 2 与 3，那么 p100 与 q10 就分别是 200 与 30，于是 p100+q10+r 就是以 p 为百位、q 为十位、r 为个位的 3 位数了。p_cube、q_cube 与 r_cube 分别储存 p、q 与 r 的立方；它们的和放在 cube_sum 中，而用 p、q 与 r 组成的 3 位数则放在 number 中。如果 number 与 cube_sum 相等，number 就是个 Armstrong 数。用这个观点写成的程序就是 ARMS2.C。注意，对于 p_cube 与 q_cube 的计算都放在合适的位置，而不是放在最内层的循环中。

3 位数的 Armstrong 数只有 4 个，即 153、370、371 与 407。

【习题】

（1）请看 UGLY.C 程序，如果 pos 变量表示输入数据 number 的位数（不一定 3 位），第一，请问在 do…while 循环执行完后 pos 是否正确。第二，请仔细研究在 else 的段落中的叙述做了什么事，有没有办法用本题的技巧来化简它？第三，这个程序在输入两位数、一位数或更多位数时会不会有正确的反应？

（2）有了这个解法，会不会求那些 4 位数的 4 次方和正好等于该数的所有 4 位数呢（解答：1634,8208,9474）？能不能写出一个一般性的程序。输入一个 1 表示位数，然后求出所有数中 n 个位数的 n 次方和正好等于该数的数呢？能不用 exp()、log()这些函数吗？数据类型该注意些什么？

Download at http://www.pin5i.com/

（3）请看下面的式子：

$$(p+1)^3 = p^3 + 3p^2 + 3p + 1$$
$$(p+1)^2 = p^2 + 2p + 1$$

换句话说，$1^3, 2^3, 3^3, 4^3, \cdots$这些数用不着每次都自乘，当手上有 p, p^2, p^3 之后，下一个 p（也就是 $p+1$）的立方可以用目前的 p^3, p^2, p 算出来。例如，如果有 $p=5, p^2=25, p^3=125$，那么 6^3 就是：

$$\begin{aligned} 6^3 &= (5+1)^3 \\ &= 5^3 + 3 \cdot 5^2 + 3 \cdot 5 + 1 \\ &= 125 + 3 \cdot 25 + 15 + 1 \\ &= 216 \end{aligned}$$

用这个技巧改写 ARMS2.C，使得不必每次都算立方，而改用加法，程序会不会快一些？

（4）请把 UGLY.C 改成可以求出 3 位数的 Armstrong 数，而不是只处理一个；然后，与 ARMS1.C、ARMS2.C，以及习题（3）的建议方式作一个时间上的比较。哪一个最快？哪一个最慢？并且说明理由。注意，最好把计算部分重复做 100 次或更多，才能算出精确的时间。

【程序】

程序 11-1（1）（ARMS1.C）

```
/* ------------------------------------------------------------------ */
/* PROGRAM Armstrong Number Search :                                  */
/*   This program searches all 3-digit Armstrong Numbers              */
/* and thus numbers in the range of 0-99 are not counted.             */
/*                                                                    */
/* Copyright Ching-Kuang Shene          June/29/1989                  */
/* ------------------------------------------------------------------ */

#include <stdio.h>
#include <time.h>

void main(void)
{
        int  p, q, r;                /* three digits                  */
        int  number;
        int  count = 0;

        printf("\nArmstrong Number Search");
        printf("\n=======================");
        printf("\n\nCount  Number");
        printf(  "\n-----  ------");

        for (number = 100; number <= 999; number++) {
            p = number/100;          /* extract left most digit */
```

Download at http://www.pin5i.com/

```
            q = (number%100)/10;    /* the middle digit          */
            r = number%10;          /* the right most one        */
            if (p*p*p+q*q*q+r*r*r == number) /* Armstrong # ?   */
                printf("\n%3d%9d", ++count, number);
        }
        printf("\n\nThere are %d 3-digit Armstrong Numbers.", count);
    }
```

程序 11-1（2）（ARMS2.C）

```
/* ------------------------------------------------------------ */
/* PROGRAM Armstrong Number Search :                            */
/*   This program searches all 3-digit Armstrong Numbers        */
/* and thus numbers in the range of 0-99 are not counted.       */
/*                                                              */
/* Copyright Ching-Kuang Shene          June/29/1989            */
/* ------------------------------------------------------------ */

#include <stdio.h>

void main(void)
{
        int  p, q, r;                   /* three digits          */
        int  p_cube, q_cube, r_cube;    /* their cubes           */
        int  p100, q10;                 /* their position values*/
        int  number;                    /* the computed number   */
        int  cube_sum;                  /* the sum of the cubes  */
        int  count = 0;                 /* counter               */

        printf("\nArmstrong Number Search");
        printf("\n=======================");
        printf("\n\nCount  Number");
        printf( "\n-----  ------");

        for (p = 1, p100 = 100; p <= 9; p++, p100+=100) {
            p_cube = p*p*p;
            for (q = q10 = 0; q <= 9; q++, q10+=10) {
                    q_cube = q*q*q;
                    for (r = 0; r <= 9; r++) {
                        r_cube   = r*r*r;
                        number   = p100 + q10 + r;
                        cube_sum = p_cube + q_cube + r_cube;
                        if (number == cube_sum)
                            printf("\n%3d%9d", ++count, number);
                    }
            }
        }
        printf("\n\nThere are %d 3-digit Armstrong Numbers.", count);
    }
```

Download at http://www.pin5i.com/

问题 2.2 数字谜（TRENTE.C）

在一些游戏与消遣的书刊中，常常会看到如图 11-1 所示的数字谜。

```
      VINGT
       CINQ
+      CINQ
------------
     TRENTE
```

图 11-1

请写一个程序破解它。

【解答】

程序在 TRENTE.C 中，对应着每一个位数都有一个 for 循环，在 for 循环中通常有一道叙述，用来测试该 for 循环的变量是否与在此之前的其他 for 循环变量相同，如果相同，就不执行该循环与该循环之下的所有内容。变量名称与题目中使用的字母相同，因此不应该会有任何阅读上的困难。问题的解答如图 11-2 所示。

```
      94851
       6483
+      6483
------------
     107817
```

图 11-2

不过，如果仔细些，问题中还可以发现不少线索。依照问题的分析，初步结果如图 11-3 所示。

```
      VING1
       CINQ
+      CINQ
------------
     10ENTE
```

图 11-3

另外，V 为 8 或 9；E 是奇数。下面一一把线索找出来，多用一条线索，程序又会加快一些。

（1）线索 1：C+N+N 一定有进位。如果 C+N+N 再加上个位数的进位不会造成进位的话，那么若个位数有进位，则 G、N 都是 0，这样相加才会是 1，但这不可能，因为 G=N 了。如果个位数没有进位，那么 G+N+N=1，就一定是 N=0，G=1；因为若 N 不是零，2N 就大于 1，G+N+N 就大于 1。于是 N 与 R 相同，G 与 T 相同，不合要求，因此十位数一定有进位。

（2）线索 2：十位的进位一定是 2。由线索 1 知，十进制一定有进位，但问题说明中曾经指出，进位最多是 2。如果进位是 1，看百位数，它的值是由 1+N+I+I 算出来的，个位数是 N，因此有 1+N+2I=x・10+N（x = 0,1 或 2）；化简它，有 1+2I=x・10，不过 x・10 是

Download at http://www.pin5i.com/

偶数，但 2I 是偶数，故 I+2I 是奇数，引起矛盾，因此得到十位的进位（到百位）是 2。

（3）线索 3：I 为 4 或 9。看百位数，它有从十位而来的进位 2（线索 2），因此百位数是 2+N+2I=x •10+N，x = 0,1 或 2 分别表示百位进到千位的值。上式消去 N，得到 2I=x •10-2。当 x=0（没有进位）时，I=-1，不可能；当 x=1（进 1 到千位）时 I=4；当 x=2（进 2 到千位）时，I=9。在原来程序中 I 不知道范围，所以只能让它从 0 变到 9，但由此知，I=4 或 9，所以工作量一下就少了 1/5。

（4）线索 4：I=4。如果 I 不是 4，而是 9，由线索 3 知百位到千位的进位是 2，因此千位以上的式子如图 11-4 所示。

```
     2     （进位）
    89
     C
+    C
-----------
   10E
```

图 11-4

注意，此时 V 已经固定是 8，原因是 V 可以是 8 或 9，但 I 已经是 9，V 就只能是 8 了。正因为 V=8，千位到万位的进位就是 2，因为 2+9+2C≥23（不是 20，因为 E=0 与 R 重复；也不是 21，因为 E=1 与 T 重复；也不是 22，因为 E 是奇数），所以 C≥6，但因为 8 与 9 都用过了，于是 C=6 或 C=7。

当 C=6 时，E=3，代回个位数，有 Q=1(1+1+1=3)或 Q=6(1+6 +6=13)，但前者与 T 重复，后者与 C 重复，因此 C 不能是 6。

当 C=7 时，E=5，代回个位数，有 Q=2(1+2+2=5)或 Q=7(1+7 +7=15)，但后者与 C 重复。如果 Q=2，注意十位数，如图 11-5 所示。

```
      2 2 2 0 0   （进位）
-----------------
      8 9 N G 1
        7 9 N 2
+       7 9 N 2
-----------------
    1 0 5 N 1 5
```

图 11-5

因此只有 3、4 与 6 这 3 个数还可以用；但 N=3,4 或 6 都不能使十位可以进 2 到百位，因此与线索 2 矛盾，Q=2 也是不对的，自然 C=7 也是不对的了。这样 C=6 与 C=7 也都不能成立，于是原来的假设 I=9 也是不对的，所以 I 只能是 4（好长的证论，希望有简单的方法）。

当 I=4 时，由线索 3，百位会进 1 到千位，如图 11-6 所示。

```
      ? 1 2 ? ?
-----------------
      V 4 N G 1
        C 4 N Q
+       C 4 N Q
-----------------
    1 0 E N 1 E
```

图 11-6

Download at http://www.pin5i.com/

现在，问题又简化了，E=3,5,7,9；V=8 或 9；C=2～9，N=0～9。循环总共绕 4×2×8×10＝640 圈而已。

（5）线索 5：V=9。如果 V 不是 9，那就是 8 了。当 V=8 时，C 只能是 9，因为 1+4+2C ≥23（见线索 4 的说明）；于是 E=3，代回个位数，Q=1 或 Q=6。Q=1 不可能，因为与 T 重复；Q=6 呢？也不可能，因为个位数是 1+6+6=13，因此进 1 到十位，由线索 2 知十位会进 2 到百位，于是十位的加法是 1+G+2N=21，故 G+2N=20，那么 G 是个偶数。到目前为止，偶数已经用了 4、6(=Q)与 8，G=2，但这却会使 N=9，与 V 重复，所以 Q 不能是 6。这指出 V=8 是不正确的，因此 V=9。如图 11-7 所示。

```
        1 2 ??（进位）
      94NG1
       C4NQ
  +    C4NQ
    10EN1 E
```

图 11-7

（6）线索 6：C=6。由上式，千位进 1 到万位就可以了，但 E 是大于等于 3 的奇数，因此 1+4+2C≥13，C≥4。于是 C 的可能性有 4、6 与 8。C 不能是 8，因为这会使千位进 2 到万位。C=6 呢？如果 C=6，那么 E=5，代回个位数，Q=2(1+2+2 =5)或者是 Q=7(1+7+7=15)。

如果 Q=2，个位不会进位到十位去，因此对十位而言，有 G+2N= 21，因此 G 是奇数。现在能用的奇数只有 3 与 7（T=1,E=5,V=9），如果 G=3，则 N=6 与 C 相同，不合；如果 G=7，N 也就是 7，G 与 N 重复，所以 Q 不能是 2。

如果 Q=7，个位就进 1 到十位，十位的加法就是 1+G+2N=21，G+2N=20，故 G 为偶数。现在偶数能用的只有 2 与 8（R=0,I=4, C=6）。当 G=2 时，N=9 与 V 重复；当 G=8 时，N=6，与 C 重复。因此 Q 也不能是 7。

两者合并，知道 C=6 是对的，故 C=6，如图 11-8 所示。

```
        112 ??（进位）
      94NG1
       64NQ
  +    64NQ
    107N1 7
```

图 11-8

（7）结论，Q=3，G=5，N=8。注意，现在还留下的只有 2、3、5、8 四个数没有用到。

看个位数，Q 只能是 3 或 8；如果 Q=8，个位会进 1 到十位，十位的加法是 1+G+2N=21，G+2N=20，故 G 是偶数，因此 G=2（Q=8）。当 G=2 时，N=9 与 V 重复，矛盾，所以 Q=3。

因为 Q=3 个位数就不会进位给十位，于是十位部分的加法为 G+2N=21，因此 G 是奇数。现在可用的数只有 2,5,8，故 G=5。

当 G=5 时，N 就只可能是 8 了。

以上是从头到尾的推论，可以在任何一个步骤停止，都会有进一步的化简程序的线索，使程序变快。下面的习题请运用推理能力多做化简的工作。

Download at http://www.pin5i.com/

【习题】

1.
```
  SEND
+ MORE
------
 MONEY
```

2.
```
   RED
×  FOR
------
DANGER
```

3.
```
  LPBRR
  TLSRR
+ SBRRR
------
 RLMPL
```

4.
```
  EIN
  EIN
  EIN
+ EIN
------
 VIER
```

5.
```
  HOW
   IS
+ HIS
------
  SIS
```

此处 sow 是一个完全平方，也就是说，sow 等于某个数的平方。

下面的问题中“*”表示一个任意的数字，不同位置的可能相同，而且也可能与英文字母或明确标出来的数字相同；但是，不论如何，最左边一位不能为 0。

6.
```
  THIS
×   IS
------
 **TOO
HARD *
------
******
```

7.
```
   ABC
×  BAC
------
  ****
  **A
***B
------
******
```

8.
```
   ***
×  *2*
------
   ***
  ****
 *8**
------
 **9*2*
```

【习题解答】

1.
```
  9567
+ 1085
------
 10652
```

2.
```
   481
×  364
------
175084
```

3.
```
  46388
  14288
+ 23888
------
  84564
```

4.
```
  821
  821
  821
+ 821
------
 3284
```

Download at http://www.pin5i.com/

5.
```
      461
       39
  +   439
  -------
      939
```
sow=961=13^2

6.
```
      4379
  ×     79
  -------
     39411
    30653
  -------
    345941
```

7.
```
      286
  ×   826
  -------
     1716
     572
   2288
  -------
   236236
```

8.
```
      620
  ×   321
  -------
      620
     1240
    1860
  -------
   199020
```

【程序】

程序 11-2（TRENTE.C）

```
/* ------------------------------------------------------------ */
/* PROGRAM Number Puzzle :                                      */
/*   This is first solution of the number puzzle.  It is        */
/* written under the guidelines in my book.  For each           */
/* digit, there corresponds a for loop and an if statement      */
/* to prevent duplicated digits.  Read it carefully.            */
/*                                                              */
/* Copyright Ching-Kuang Shene          June/30/1989            */
/* ------------------------------------------------------------ */

#include <stdio.h>

void main(void)
{
     long  I, N, R=0, E, V, G, C, Q, T=1;
     long  IN, VINGT, CINQ, TRENTE;
     long  sum;

     printf("\nNumber Puzzle\n");
     printf("\n   VINGT");
     printf("\n    CINQ");
     printf("\n+)  CINQ");
     printf("\n--------");
     printf("\n  TRENTE\n");

     for (V=8; V<=9; V++)
       for (I=0; I<=9; I++)
         if (I != V && I!=T)
           for (N=0; N<=9; N++)
```

Download at http://www.pin5i.com/


```
if (N!=I && N!=V && N!=T) {
   IN = I*10 + N;
   for (G=0; G<=9; G++)
     if (G!=N && G!=I && G!=V && G!=T && G!=R)
        for (C=2; C<=9; C++)
          if (C!=G && C!=N && C!=I && C!= V && C!=T && C!=R)
             for (Q=2; Q<=9; Q++)
               if (Q!=C && Q!=G && Q!=N && Q!=I && Q!=V
                     && Q!=T && Q!=R)
                for (E=3; E<=9; E+=2)
                if (E!=Q && E!=C && E!=G && E!=N && E!=I
                                 && E!=V && E!=T && E!=R) {
                TRENTE=((((T*10+R)*10+E)*10+N)*10+T)*10+E;
                VINGT=((V*100+IN)*10+G)*10+T;
                CINQ =(C*100+IN)*10+Q;
                sum=VINGT+CINQ+CINQ;
                if (sum==TRENTE) {
               printf("\n\nThe answer is :\n");
               printf("\n%8ld", VINGT);
               printf("\n%8ld", CINQ);
               printf("\n+)%6ld", CINQ);
               printf("\n--------");
               printf("\n%8ld", TRENTE);
                                }
                                }
            }
      }
```

问题 2.3　求质数（PRIME1.C）

请写一个程序，找出前 N（比如说 200）个质数。这不是难题，如果没有进一步要求的话。不过，希望就所知的、使用除法的方法中，用最快的办法来编写程序。

【解答】

程序的结构本身并不复杂，只解释几个重点而已。

第一，看如何跳过 2 与 3 的倍数。在问题说明中，看过在 6n，6n +1，6n +2，6n +3，6n +4 与 6n +5 中，只有 6n +1 与 6n +5 要做测试，看看是否是质数，看看这两个数的相对位置。

如图 11-9 所示，上一组最后一个要测试的值，与这一组第一个要测试的值的距离是 2；而同一组中要测试的两个值之间的距离是 4。所以从 5 起，依次加 2、加 4、加 2、加 4、…就可以得到 5,7,11,13,17,19,…这些要测试的值了。但是 2+4=6，可以用一个变量（程序中是 gap），令它的值在开始时为 2，下一个值就是 6-gap（就是 4），再下次 6-gap 不就是 2 了吗？这就是程序中 gap=6-gap 的意义。

Download at http://www.pin5i.com/

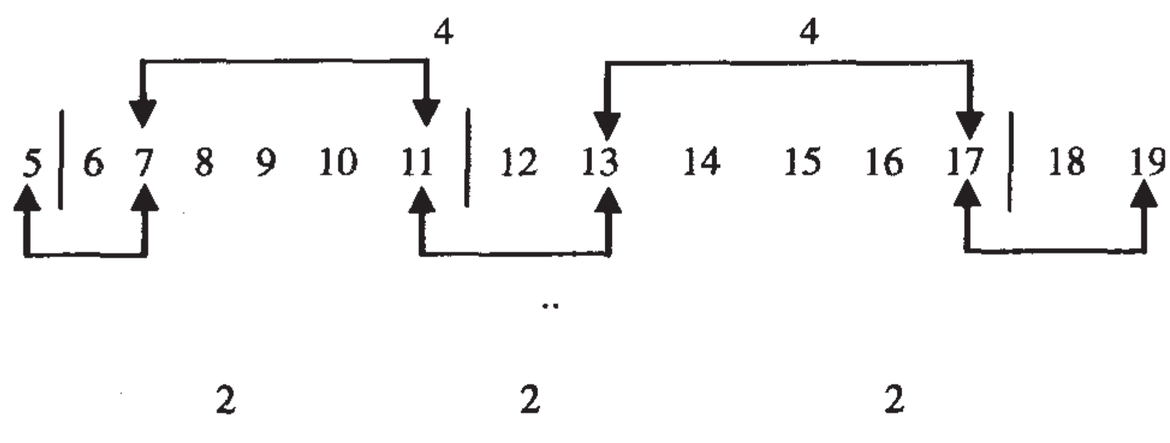


图 11-9

第二，不能比较是否小于$\sqrt{i}$，原因是对某些 i 而言，$\sqrt{i}$可能并不精确。举个例子，如果 i=121，在数学上$\sqrt{i}$自然是 11，但是计算机算出来的结果很可能是 10.999999，于是去除 i 的数就是 2,3,5,7 而不含 11，因此变成 121 是个质数了。解决之道很简单，不要用开方，用平方即可。例如，如果 p·q≤i，就用 p 去除，这就一定不会出现上面的问题。不但如此，平方的运算时间也一定会比开方有效率。

【程序说明】

在程序中，prime[ ]是用来存放找出来的质数的，一开始时就有了 2,3,5 这 3 个最基本的质数（都小于 6）。gap 的初值是 2，它的值依次是 2,4,2,4,…，在解答中见过了。Count 计算到目前为止一共找到了多少个质数，2,3,5 已经放入 prime[ ]，所以 count 的值是 3。于是当 count 比 MAXSIZE 小的时候，就产生下一个要检查的数，存入 may_be_prime，进入 for 循环做检查工作；如果是个质数，那么 for 循环中的 if 就永远是假，is_prime 的值就不曾被改变，因此就把它存入 prime[ ]中，这就是程序的基本结构。

【习题】

（1）笔者认为，使程序中跳过 2,3,5 的倍数也是可能的，请构想一个可以达成目标的做法。如果想出来了，请问它有多复杂，值不值得写到程序中？如果用这个办法，程序会快多少（当然会快的）？

（2）程序中把 5 放到 prime[ ]中，请问这是不是多此一举？为什么？如果不这样做，有什么好建议？

（3）有人说，is_prime 这个变量是多余的，其实也是如此，有没有办法修改这个程序，使它不用 is_prime 呢？程序会比较容易读吗？

（4）笔者曾在一本书中见到一个程序（BAD.C），它是用来求 1～100 之间的质数用的，但该书的作者要求“尽可能地使循环的执行次数为最少”。请问，这位作者的要求达到没有？为什么？请列出所有的改良建议。

（5）请修改程序，让它可以求出 1～N（输入）的所有质数。请问，如果程序又改成读入 M 与 N（M<N=，求出 M～N 之间的所有质数时，程序的工作量可以少一些吗？为什么？

【参考文献】

这是所有用除法来求质数的方法的基本类型，由此出发，还有各式各样的方法，速度越来越快；单单针对本题的材料，可以参看 1986 年 1 月与 1986 年 2 月份，在《微电脑时代》杂志上 Micro 随笔专栏中有关《求质数》这两篇文章；另外，下面几个方法（在 Micro

Download at http://www.pin5i.com/

随笔中介绍了一部分）也是很有趣的。

1 B.A.Chartres. Algorithm 310, Prime Number Generator, *Communications of ACM*. Vol.10(1967), pp.569

2 B.A.Chartres. Algorithm 311, Prime Number Generator, *Communications of ACM*. Vol.10(1967), pp.570

3 R.C.Singleton. Algorithm 356, A Prime Number Generator, *Communications of ACM*. Vol.12(1969),pp.563

4 R.C.Singleton. Algorithm 357, Prime Number Generator, *Communications of ACM*. Vol.12(1969), pp.563～564

【程序】

程序 11-3（BAD.C）

```
/* ------------------------------------------------------------ */
/* PROGRAM prime number generator :                             */
/*   This program generates first MAXSIZE-1 primes by           */
/* using the conventional division method, but multiples        */
/* of 2 and 3 are not tested.  Therefore it is faster.          */
/*                                                              */
/* Copyright Ching-Kuang Shene          July/02/1989            */
/* ------------------------------------------------------------ */

#include <stdio.h>

#define  MAXSIZE    100
#define  YES          1
#define  NO           0

void main(void)
{
     int  prime[MAXSIZE];          /* array to contains primes    */
     int  gap = 2;                 /* increasing gap = 2 and 4    */
     int  count = 3;               /* no. of primes               */
     int  may_be_prime = 5;        /* working  variable           */
     int  i, is_prime;

     prime[0] = 2;                 /* Note that 2, 3 and 5 are    */
     prime[1] = 3;                 /* primes.                     */
     prime[2] = 5;
     while (count < MAXSIZE)  {    /* loop until prime[] full     */
         may_be_prime += gap;      /* get next number             */
         gap = 6 - gap;            /* switch to next gap          */
         is_prime = YES;           /* suppose it were a prime     */
         for (i = 2; prime[i]*prime[i] <= may_be_prime && is_prime; i++)
```

Download at http://www.pin5i.com/


```
        if (may_be_prime % prime[i] == 0)  /* NO            */
            is_prime = NO;                 /* exit          */
    if (is_prime)                     /* if it is a prime...  */
        prime[count++] = may_be_prime;     /* save it       */
}

printf("\nPrime Number Generation Program");
    printf("\n==============================\n");
    printf("\nFirst %d Prime Numbers are :\n", count);
    for (i = 0; i < count; i++) {
        if (i % 10 == 0) printf("\n");
        printf("%5d", prime[i]);
    }
}
```

【程序】

程序 11-4（PRIME1.C）

```
#include <stdio.h>
#include <math.h>                /* for sqrt()                  */

#define MAXSIZE  100
#define YES        1
#define NO         0

void main(void)
{
    int    prime[MAXSIZE];
    int    count = 1;
    int    i, j, is_prime;
    double sqrt_i;

    prime[0] = 2;
    for (i = 3; i < MAXSIZE; i++) {
        is_prime = YES;
        sqrt_i   = sqrt((double) i);
        for (j = 0; prime[j] <= sqrt_i && is_prime; j++)
            if (i % prime[j] == 0)
                is_prime = NO;
        if (is_prime)
            prime[count++] = i;
    }

    printf("\nPrime Numbers between 1 and %d are\n", MAXSIZE);
    for (i = 0; i < count; i++) {
        if (i % 10 == 0) printf("\n");
        printf("%5d", prime[i]);
    }
```

Download at http://www.pin5i.com/


```
        printf("\n\nThere are %d Prime Numbers in total", count);
    }
```

问题 2.4　筛法（SIEVE.C）

求质数，是一个很普遍的问题，通常不外乎用数去除，到了某个地步除不尽了，给定的数就是质数，要不然则是合数。但是早在 2000 年前人们就知道了一个不必用除法而找出 2～N 之间的所有质数的所有方法了。假设有一个很神奇的筛子，可以给它一个数 i，这个筛子有办法把 i 的所有倍数去掉。请用这个方法求出 2～N 之间的所有质数。

这个方法叫做 Eratosthenes（人名）筛法（Sieve Mothod）。

【解答】

用一个数组 x[0],x[1],…来储存 3,5,7,11,…这些奇数，因此 x[i]中所储存的就是 2i+3，或者是写成 i+i+3。在开始的时候，把每一个元素都定成“没有筛掉”，然后再逐次把合数去掉。

如何把合数筛掉呢？办法其实很容易，把 x[0],x[1],…一个接一个地查过去，如果 x[i]没有被筛掉，那它一定是个质数（为什么？），但 x[i]对应的是 2i+3，因此把 2i+3 的所有倍数筛掉。2i+3 的倍数在 x[ ]数组中的什么地方呢？首先，知道 2i+3 的倍数可以写成(2i+3)j，此处 j>1（如果 j=1，就把自己筛掉了）；不过，因为 2i+3 是个奇数，而且 x[ ]中所有的元素又是奇数，所以要筛掉的是 2i+3 的奇数倍数，换言之，即 3(2i+3),5(2i+3),7(2i+3),…；这些奇数又是 2n+1 的形式，因此要删掉的其实是(2n+1)(2i+3)这些数，n=1,2,3,…。化简这个式子，得到：

$$\begin{aligned}(2n+1)(2i+3) &= 2n(2i+3)+2i+3 \\ &= 2[n(2i+3)+i]+3\end{aligned}$$

所以当知道 x[i]所对应的数(2i+3)是个质数之后，所要筛掉的数在 x[ ]中对应的位置是 n(2i+3)+i，n=1,2,3,…。当 n=1 时，这个式子是(2i+3)+i，n=2 时就是(2i+3)+i 加上(2i+3)，n=3 时再加上 2i+3；所以可以用(2i+3)+i 作初值，筛掉 x[ ]中那一个数，然后就每次加上 2i+3，筛掉对应的数等，一直到无数可筛为止。如果数组有 N 个元素，那么最后一个元素 x[N-1]就对应着 2・N+3，因此就可以求出 2 的 2・N+3 之间的所有质数了。

在程序中 sieve[ ]就对应着 x[ ]，MAXSIZE 对应着 n。一开始每一个元素存入 KEPT，表示没有筛掉。

中间的大 for 循环就把 sieve[ ]的元素一个接一个查过去，一旦查到 sieve[i]是没被筛掉的，那么 i+i+3 就是个质数，把它存放在变量 prime 中，count 是用来记录到目前为止找到了多少个质数，在开始时，2 是质数，所以 count 为 1。接下来的 while 循环中，k 就是用来指出要筛掉 prime 的倍数，前面提过，k 从 prime+i 起，每一次增加 prime 这个值，并且把 sieve[k]筛掉。到 sieve[ ]中元素都处理完后，值为 KEPT 的就是没有被筛掉的元素；程序中剩余的部分把结果显示出来。

【习题】

（1）为什么在程序运行过程中，若 sieve[i]为 KEPT，2i+3 就是一个质数呢？

Download at http://www.pin5i.com/

（2）请分析一下程序一共查过了多少个元素。

提示：如果 sieve[ ]有 n 个元素，在做到第 i 个时，要筛掉 2i+3 的倍数，这有多少个？当然这与 i 有关，当把所有与 i 对应的量都加起来之后，结果就做出来了。

（3）不处理 2 的倍数可以少处理一半的元素；因此有人建议，连 3 的倍数也不处理了，速度不就更快了吗？2 与 3 的倍数是 6 个一组的：6n,6n+1,6n+2,6n+3,6n+4,6n+5；6n,6n+2,6n+4 是 2 的倍数，6n+3 是 3 的倍数，所以 6 个数中可能会是质数的就只剩下 6n+1, 6n+5 而已，原来的 2/6＝1/3。换句话说，处理 1/3 的元素就行了。请把这个观点写成程序。

【程序】

程序 11-5（SIEVE.C）

```
/* ------------------------------------------------------------ */
/* FUNCTION sieve :                                             */
/*   This program uses Eratosthenes Sieve method to             */
/* generate all primes between 2 and MAXSIZE*2+3.  This         */
/* is a very simple yet elegant method to generate primes       */
/*                                                              */
/* Copyright Ching-Kuang Shene          July/01/1989            */
/* ------------------------------------------------------------ */

#include <stdio.h>
#define  MAXSIZE      200
#define  DELETED        1
#define  KEPT           0

void main(void)
{
     int  sieve[MAXSIZE+1];          /* the sieve array          */
     int  count = 1;                 /* no. of primes counter    */
     int  prime;                     /* a generated prime        */
     int  i, k;                      /* working variable         */

     printf("\nEratosthenes Sieve Method");
     printf("\n========================");
     printf("\n\nPrime Numbers between 2 and %d\n", MAXSIZE*2+3);

     for (i = 0; i <= MAXSIZE; i++=/* set all items to be        */
         sieve[i] = KEPT;            /* kept in the sieve        */

     for (i = 0; i <= MAXSIZE; i++=/* for each i, it             */
         if (sieve[i] == KEPT) {     /* corresponds to 2i+3      */
             prime = i + i + 3;      /* if it is not sieved      */
             count++;                /* prime=2i+3.              */
             for (k = prime + i; k <= MAXSIZE; k += prime=
```

Download at http://www.pin5i.com/


```
                sieve[k] = DELETED;     /* screen multiple      */
            }

        printf("\n%6d", 2);             /* output part.         */
        for (i = 0, k = 2; i <= MAXSIZE; i++=
            if (sieve[i] == KEPT) {
                if (k > 10) {
                    printf("\n");
                    k = 1;
                }
                printf("%6d", 2*i+3);
                k++;
            }
        printf("\n\nThere are %d primes in total.", count);
    }
```

问题 2.5 线性筛法（L_SIEVE.C）

在做筛法（Sieve）求质数的问题时，在删除非质数的数据时，有很多是重复删除的。例如，如果有一个数是 3×7×17×23，那么在删除 3 的倍数时会删除它，删除 7、17 与 23 的倍数时也都会删掉它。看起来好像并没有多大关系，因为只是把一个值存入对应的位置而已，但是，当要求质数的范围很大时，这一类型有很多质因子的合数就会越来越多，因此程序的效率自然地就会大打折扣。基于筛法的观点，请写一个程序，在删除非质数时“绝对”不做重复的工作。

【解答】

删除的技巧并不复杂，从 2 开始，先删除 $2^2,2^3,2^4,\cdots$，接着删除 $2\cdot3,2^2\cdot3,2^3\cdot3,\cdots$（注意，3 并没有被删除），再删除 $2\cdot5,2^2\cdot5,2^3\cdot5,\cdots$；再来是 $2\cdot7,2^2\cdot7,2^3\cdot7,\cdots$；$2\cdot9,2^2\cdot9,2^3\cdot9,\cdots$。一般而言，当发现 p 是一个质数时，就先删除 $p^2,p^3,p^4,\cdots$这一串合数。正因为 p 是个质数，$p^2,p^3,p^4,\cdots$不可能被其他数整除，所以在此之前不可能被删除。接着，去找出比 p 大，但没有被删除的一个数，令为 q；再去删除 $p\cdot q,p^2\cdot q,p^3\cdot q,p^4\cdot q,\cdots$；因为 p 是质数，q 是一个在此之前没有删除的数，因此 $p^i\cdot q$（$i=1,2,\cdots$）在此之前也没有被删除过。为什么会这样呢？简单地说，如果 $p^i\cdot q$ 在此之前已经被删除了，那么依照上面所讲的删除方式，一定有一个质数 a（a<p=存在使得 a^kb 正好就是 $p^i\cdot q$，这样在删除 a 的倍数时就会把 a^kb 删掉的。不过，$a^kb=p^i\cdot q$，a<p，这是不可能的，因为 a 与 p 都是质数，彼此是互质（不能整除对方）的，所以 p^i 就一定整除 b 才行，于是 $q=a^k(b/p^i)$，q 是 a^k 的倍数，在删除 a^k 的倍数时 q 早就被删除了才对，这与前面所说“q 是比 p 大的第一个没有被删除的数”相矛盾。因此 $p^i\cdot q$（$i=1,2,\cdots$）都是第一次被删除，换句话说，删除的动作绝不重复。

如果要找出 2～n 的所有质数，那么 p 最大不能超过 $\sqrt{n}$，亦即 $p^2\leqslant n$。同样的道理，任何一个 $p^i\cdot q$ 也都不能大过 n，所以删除的工作可以很简单地写成如程序 11-6 所示。

Download at http://www.pin5i.com/

【程序】

程序 11-6

```
for (p = 2; p*p <= n), NEXT(p))
     for  (q = p; p*P <= n; NEXT(q))
          for (m=p*q; m <= n; m=p*m)
               REMOVE(m);
```

在此 p 从 2 开始，每次删除完 p 的倍数之后，就用 NEXT()来找出比 p 大但未曾被删除的下一个数，它一定是质数（为什么？）；找到了 p 之后，令 q 从 p 起，删除 $pq, p^2q, p^3q, \cdots$，一直到 $p^i \cdot q > n$ 为止，之后再找出下一个没有被删除的数作 q。如此一直循环，直到找出的 q 满足 $p \cdot q > n$ 为止。这个程序的核心简单，但要如何写出来呢？

做法是用一个双向连接串行（Double Linked List），在开始的时候第 i 个元素的 previous[i]是 i-1（指向上一个），next[i]是 i+1（指向下一个），但 previous[2]没有上一个，next[n]没有下一个，用 NULL 表示。程序中的 INITIAL()集体指令做的就是这件事。这样，要删除某一个数，例如 m，是很简单的，只要让 m 的上一个（previous[m]）元素的下一个指针（next[previous[m]]）指向 m 的下一个元素（next[m]），再把 m 的下一个元素的上一个指针（previous next [m]））指向 m 的上一个元素就行了。这个集体指令叫做 REMOVE()。最后 NEXT[x]是把 x 定成 x 的下一个元素，这是最简单的一个，因为只要取出 x 的下一个，即 next[x]，再存回 x 就大功告成。

有了这 3 个集体指令之后，从开始到把 2～n 的质数找出来，就只不过是 4 列程序而已，可以参看下面的 L_SIEVE.C。

【习题】

（1）这个程序花了一半的时间去删除 2～n 之间的偶数，这是多余的，因为除了 2 之外，所有偶数都不是质数。请修改此程序，让它不处理 2 之外的偶数，这就省下一半的时间（2～n 之间有一半元素是偶数）。提示：参考上一题的技巧。

（2）此处所用的内存量很明显地比上一题的传统筛法程序大许多，把它省下一半。首先不要 previous[ ]，只留下 next[ ]。对于 next[i]而言，如果 i-1 并没有被删除，i 的上一个是 i-1；但若 i-1 已经被删除了，用 next[i-1]来储存原来的 previous[i]。换句话说，i 的上一个就是 i-1 与 next[i-1]中值较小的那一个。请问，还有什么边界条件（Boundary Condition）要考虑，并且把这个概念写成程序。这个技巧在习题（1）中能不能适用呢？

（3）在这一题看看如何修改本问题而同时求出 2～n 之间每一个数的因式分解，这是 Dexter Kozen 提出来的方法。因为每一个合数 r 都可以写成 $r=p^i \cdot q$ 的形式，p 是 r 的因式分解中最小的质数。当用本题求 2～n 的质数时，一旦决定要删除 r 了，就有 $p^i \cdot q$，可以修改程序算出 p,i 与 q，再准备 3 个数组，即 P[ ]、EMP[ ]与 Q[ ]，在 P[r]、EXP[r]与 Q[r]中分别储存 p、i 与 q；储存的动作只会做一次。如果在程序执行之前，在 P[ ]中存入一个奇怪的值（比如 0），于是在程序执行完后，如果 P[t]的值并未被改变，表示 t 不能写成 $p^i \cdot q$ 的形式，亦即不能分解，所以 t 是质数。如果原来 P[t]的值已经被改变了，就可以去找 Q[t]，再用 Q[t]作脚码去查 P[Q[t]]而得到 Q[t]的分解方式，因此一路追踪，就可以把 t 分解完成。

Download at http://www.pin5i.com/

请把这个概念写成程序。

【参考文献】

本题的解法出自 D.Gries 与 J.Misra 的文章，习题（3）亦然。

1 D.Gries and J.Misra. A Linear Sieve Algorithm for Finding Prime Numbers, *Communications of ACM*. Vol.21(1978), pp.999～1003

此外，还可以参阅下面几篇有趣的作品：

2 H.G.Mairson. Some New Upper Bounds on the Generation of Prime Numbers, *Communications of ACM*. Vol.20(1977), pp.664～669

3 P.Pritchard. A Sublinear Additive Sieve for Finding Prime Numbers, *Communications of ACM*. Vol.24(1981), pp.18～23

P.Pritchard 还有另外的几篇文章，但却超越了本书的范围，在此就不列出来了。另外，在《微电脑时代》杂志 1986 年 1 月与 2 月份的“Micro 随笔专栏”中，求质数的部分有几个老方法的介绍，但速度都不是很快。

4 洗镜光．求质数．微电脑时代．1986

但是，在真正的质数研究中，不用上面任何一个办法来检定质数，因为在这种研究中，一个数动辄一两百位，因此这些办法不是无能为力，就是内存用量过大，而且测试一个数是否是质数还是一个在 NP∩Co-NP 中的难题。不过，数学家与计算机科学家还是开发了很多方法来做这件事，但是所用的技巧都太过深奥，而无法在此介绍，有兴趣的读者请参看下面的这两本书，以及书中所列的参考文献。

5 D.M.Bressoud. *Factorization and Primality Testing*, Springer-Verlag, 1989

6 P.Ribenboim. *The Book of Prime Number Records*, 2nd.ed., Springer-Verlag, 1989

【程序】

程序 11-7（L_SIEVE.C）

```
/* ------------------------------------------------------------------ */
/* PROGRAM linear sieve program of Gries and Misra :                  */
/*   This program finds all prime numbers between 2 and               */
/* n, the input, by using Gries and Misra linear sieve                */
/* method.  This method does not use division, instead                */
/* multiplications are used.                                          */
/*                                                                    */
/* Copyright Ching-Kuang Shene          July/10/1989                  */
/* ------------------------------------------------------------------ */

#include <stdio.h>
#include <stdlib.h>

#define  MAXSIZE      1000
#define  NEXT(x)      x = next[x]
#define  REMOVE(x)   { next[PREVIOUS[x]] = next[x]
                       previous[next[x]] = previous[x];          \
```

Download at http://www.pin5i.com/


```
                    }
#define   INITIAL(n)  { unsigned long i;                          \
                        for (i = 2; i <= n; i++)                  \
                            previous[i] = i-1, next[i] = i+1;     \
                         previous[2] = next[n] = NULL;            \
                    }

void main(void)
{
     unsigned long  previous[MAXSIZE+1];     /* prev. pointer    */
     unsigned long  next[MAXSIZE+1];         /* next pointer     */
     unsigned long  prime, fact, i, mult;
     unsigned long  n;
     unsigned long  count = 0;
     char           line[100], dummy;

     printf("\nLinear Sieve Program");
     printf("\n====================");
     printf("\n\nPrime Numbers between 2 to --> ");
     gets(line);
     n = strtoul(line, &dummy, 10);

     INITIAL(n);                             /* initial the set      */
     for (prime = 2; prime*prime <= n; NEXT(prime))
         for (fact = prime; prime*fact <= n; NEXT(fact))
             for (mult = prime*fact; mult <= n; mult *= prime)
                 REMOVE(mult);               /* remove multiples     */

     for (i = 2; i != NULL; NEXT(i)) {  /* display result        */
         if (count % 8 == 0)  printf("\n");
         printf("%6ld", i);
         count++;
     }
     printf("\n\nThere are %ld Prime Numbers in Total", count);
}
```

问题 2.6　因子分解（FACTOR.C）

请写一个程序，读入一个正整数，把它的所有质因子找出来。例如，输入是 72，$72 = 2^3 \cdot 3^2$，于是质因子就有 2 与 3；如果输入是 181944，那么因为 $181944 = 2^3 \cdot 3^2 \cdot 7 \cdot 19^2$，所以质因子为 2、3、7 与 19。为了方便起见，$2^3 \cdot 3^2 \cdot 7 \cdot 19^2$ 可以用 2(3)3(2)7(1)19(2)作输出，也就是说，如果分解开来有 a^b，在输出中就是 a(b)。

Download at http://www.pin5i.com/

【解答】

用一个数不断地去除另一个数的动作可以用 for 循环很漂亮地写出来，就是说要用 k 去除 work，一直到除不尽或被除数变成 1 为止。可以写成：

```
for (i=0; work % k == 0 && work > 1; work /= k, i++)
```

变量 i 是用来计算整除了多少次的，work 是要被除的数，至于 k 则是固定的除数，自然 i 的值在原始时为 0。每一次循环时，都用 k 去除 work，看余数是不是 0，而且被除数是否大于 1；如果是，就进入循环主体，但是因为循环主体空无一物，这就相当于做到增加值的部分，把 work 被 k 除的商求出来，当作下一次除法的被除数，并且因为已经整除了一次，i 的值也加上 1。这就是用一个数反复地除另一个数，直到除不尽时的做法。了解这一点之后，令 k=2,3,5,7,9,11,…这样反复地执行上面的循环，并且把 k 与 i（前者是因子，i 是指数）显示出来，一直到 k 大过 work 为止即可。注意，正整数被 2 除可以用右移 1 位完成，而余数部分正是最右边的那一位的值，这是下面 FACTOR.C 程序中第一个 for 循环的意义。

【习题】

（1）程序中都有 work>1 的测试，究竟这有没有必要？请说明理由。

（2）在第 2 个 for 循环中，分别用 k=3,5,7,9,11,13,15,…去除，但事实上在用 3 去除时就已经去掉了所有 3 的倍数，所以 9 与 15 是不可能整除 work 的，做了多余的事。请修改这个程序，使得只会用质数去除，9,15,…等就不会出现了。

（3）请问，这个程序所输出的因子为什么都是质数？试证明它。

（4）在程序中，其实 factors [ ]与 exps[ ]这两个用来存放因子与对应的指数的数组是不必要的，为什么？请修改程序，不用这两个数组，但效果完全相同。

【程序】

程序 11-8（FACTOR.C）

```
/* ------------------------------------------------------------ */
/* PROGRAM factorization of positive integers :                 */
/*   Given an unsigned long integer, this program finds         */
/* all prime factor by using traditional division method.       */
/*                                                              */
/* Copyright Ching-Kuang Shene      July/10/1989                */
/* ------------------------------------------------------------ */

#include <stdio.h>
#include <stdlib.h>

#define  MAXSIZE 100
#define  SAVE_FACTOR(fact, exp) { if (exp > 0)                  \
                                      factors[count] = fact,    \
                                      exps[count++]  = i;       \
                                 }
```

Download at http://www.pin5i.com/


```
void main(void)
{
    unsigned long  factors[MAXSIZE];    /* stores factors      */
    unsigned long  exps[MAXSIZE];       /* stores exps         */
    unsigned long  n, work;
    int            count = 0;           /* factor counter      */
    int            i, k;
    char           line[100], *dummy;

    printf("\nFactorization by Division Program");
    printf("\n================================");
    printf("\n\nInput a positive integer --> ");
    gets(line);
    n = strtoul(line, &dummy, 10);

    for (i=0,work=n; (work & 0x01UL)==0 && work>1; work>>=1,i++)
        ;                               /* extract divisor 2   */
    SAVE_FACTOR(2, i);                  /* save it and its exp.*/

    for (k = 3; k <= work; k += 2) {    /* for k=3,5,7,9,…     */
        for (i = 0; work % k == 0 && work > 1; work /= k, i++)
            ;                           /* extract divisor k   */
        SAVE_FACTOR(k, i);              /* save it and its exp.*/
    }

    printf("\n%ld = ", n);              /* display result.     */
    for (i = 0; i < count; i++)
        printf("%ld(%ld)", factors[i], exps[i]);
}
```

问题 2.7 数值自乘递归解（R_POWER.C）

如果 n 与 m 是正整数，那么 m^n 就是把 m 连乘 n 次，这是一个很没有效率的方法。试编写一个更有效率的程序，并且分析程序中一共用了多少个乘法。应该以少量 n-1 个乘法作为设计准则。

【解答】

关键所在就是下面的定义式：

$$m^2=\begin{cases}1 & n=0\\ (m^k)^2 & n=2k=\text{（偶数）}\\ m\cdot m^{2k} & n=2k+1=\text{（奇数）}\end{cases}$$

Download at http://www.pin5i.com/

所以在一个递归程序中分成 3 部分计算就可以了：第一部分看 n 是否为 0；第二部分看 n 是否是偶数，如果是，就求 m 的 n/2 次方（递归），算出来的结果再平方就是答案，换言之 $m^n=(m^{n/2})^2$；当 n 是个奇数时，例如 $n=2k+1$，$m^{2k+1}=m\cdot m^{2k}$，递归调用自己去算 m^{2k}，得出结果后与 m 相乘就是最终结果了。

【分析】

（1）每一次分成两部分（m^k 或 m^{2k}）之后，都要用一个乘法把 m^km^k 或 mm^{2k} 算出来；在最坏的情况下，就是 n 是奇数，$m^n=m\cdot m^{2k}$，因此在算 m^{2k} 时只少了 1。不过，m^{2k} 却是偶数了，下一层递归就会去算 m^k，再算 $(m^k)^2$。

（2）因此递归的层次最多 $2\cdot\log_2 n$ 层，自然乘法的个数就是这个数目。

【习题】

（1）请用手算模拟求 $2^{13},2^{15},2^{17}$。

（2）请算一算程序计算 $2^7,2^8,2^9,\cdots,2^{14}$ 所花的时间（或许在程序中多做几次才能得到正确的时间值），请问在时间上有没有什么异常的情况？能解释吗？

（3）（续上题）请以求 $2^{11},2^{13},2^{15}$（手算）的步骤来解答上一题的异常情形。

（4）请把这个程序改成不用递归的写法；千万不要用堆栈来仿真。

（5）如果把每一次递归调用时的 n 值输出来，也许可以帮助解决第（4）题；但是这些值与 n 的二进制表示法有什么关联呢？

（6）请扩充这个程序，使得 m 与 n 可以是负整数或 0，如何处理那些不正常的情况？

【参考文献】

有关更详尽的 m^n 的解法，请参看信息传真中笔者的一篇文章。

1 冼镜光．乘幂、组合系数与阶乘——有效率地解决问题之二，信息传真微电脑版．Vol.8, (1989), No.11, pp.140～153

【程序】

程序 11-9（R_POWER.C）

```
/* ------------------------------------------------------------ */
/* FUNCTION recursive_power :                                   */
/*   Given two unsigned long data, m and n, this routine        */
/* computes m^n with recursive technique so that number         */
/* of multiplications can be reduced to O(log(n)) instead       */
/* of O(n).  Here we require m and n be positive integers       */
/* but it is a easy matter to extend it to all integers.        */
/*                                                              */
/* Copyright Ching-Kuang Shene          July/01/1989            */
/* ------------------------------------------------------------ */

unsigned long  recursive_power(unsigned long m, unsigned long n)
{
     unsigned long temp;
```

Download at http://www.pin5i.com/


```
    if (n == 0)                    /* m^0 = 1                        */
        return 1;
    else if (n & 0x01UL == 0) {/* if power is even then              */
        temp = recursive_power(m, n >> 1);
        return temp * temp;        /* m^(2k) = (m^k)^2               */
    }
    else                           /* otherwise, m^n=m*m^(n-1)       */
        return m * recursive_power(m, n-1);
}
```

问题 2.8 数值自乘非递归解（I_POWER.C）

续求 m^n 问题（m 与 n 是正整数）。前面的提示会得到一个递归的程序，请开发一个非递归但效率相同的程序。

【解答】

先以求 m^{45} 为例，$45_{10}=101101_2$，所以：

$$m^{45}=m^{(1.2^5+0.2^4+1.2^3+1.2^2+0.2^1+1.2^0)}$$
$$=(m^{2^5})(m^{2^3})(m^{2^1})(m^{2^0})$$

在 $45_{10}=101101_2$ 中，值为 1 的那一位（比如第 i 位，从右往左为 0,1,2,3,…），在 m 的指数中就有 m^{2^i} 那一项。正因为这个缘故，可以依次地把 $m^{2^0},m^{2^1},m^{2^2},\cdots,m^{2^i},m^{2^{i+1}},\cdots$ 求出来，这正好对应着 n 的二进制表示中从右向左的顺序，如果在 n 的二进制表示中第 i 位是 1，就把 m^{2^i} 乘到一个用来保存最终结果的变量中（初值为 1），因此当 n 的每一位都查完之后，就有了答案了。

如何计算 $m^{2^0},m^{2^1},\cdots,m^{2^i},m^{2^{i+1}},\cdots$ 这些数呢？请看下式：

$$m^{2^{i+1}}=m^{2^i\cdot 2}=\left(m^{2^i}\right)^2$$

在知道了 m^{2^i} 的前提下，把 m^{2^i} 求平方就可以得到 $m^{2^{i+1}}$。但是 $m^{2^0}=m^1=m$，所以求这些 m^{2^i} 只不过是求 m 的自乘而已，换句话说，一开始时值为 m，下一次是 m^2，再下一次为 $m^2\cdot m^2=m^4$，接下来是 $m^4\cdot m^4=m^8$，…。然后就要看看如何求出 n 的二进制表示了。这倒是大家都知道的“常识”，把 n 用 2 去除的余数，就是最右边的一位，再用 2 去除上一次得到的，商就是下一位，……。

在程序中，用 temp 来保存最终结果，它的初值是 1。在程序中 m 与 n 会被毁，所以它们不是用指针来传递。在 while 循环中，就测试 n 的值，如果 n 是奇数，表示 n 最右边的数为 1，于是 m 目前的值乘到 temp 中。不管如何，都把 m 求平方，把 n 减半，所以 m 的值就依次是 $m,m^2,m^4,m^8,\cdots$，对应着从右向左，每一位的位置值；而 n 就相当于表示出它的二进制值。就以 n=45 为例，下面就是循环执行时的状态，如表 11-1 所示。

Download at http://www.pin5i.com/

表 11-1

n	m	temp	说　明
45	m	1	初值
22	m^2	m	n 为奇数，乘入 temp
11	m^4	m	n 为偶数，temp 不变
5	m^8	m^5	n 为奇数，更新 temp
2	m^{16}	m^{13}	n 为奇数，更新 temp
1	m^{32}	m^{13}	n 为偶数，temp 不变
0	m^{64}	m^{45}	n 为奇数，更新 temp

【分析】

这个程序中的 while 循环最多只会绕 $\log_2 n+1$ 次，原因是每绕一次，n 的值就减半，所以最多 $\log_2 n+1$ 次之后 n 就会是 0（为什么呢？）。每绕一次时，最多做两次乘法，一次是 m 的自乘，一次是把 m 乘入 temp，所以最多做 $2(\log_2 n+1)$ 次乘法，所以速度会比传统连乘的方式快。就以 m^{45} 为例，传统连乘方式要乘 $45-1=44$ 次，但此处只乘了 14 次。

【习题】

（1）这个程序的乘法数目还可以再降低，其实保持完整的 $m,m^2,m^4,m^8,\cdots$ 是没多大必要的，请以此为出发点来改良程序。

（2）请用手算方式列出计算 $2^{13},2^{15},2^{16}$ 的过程，是不是这个程序也会得到算 2^{15} 时要多几个乘法呢？请问，一般而言，当 n 是多少时会比较慢？

（3）如果用的硬件系统会检测出整数溢位（Overflow），那么这个程序可能会有问题，因为求出 m^{45} 之后，程序还会多算 m^{64}，这个 m^{64} 就可能溢位了。请修改这个程序，克服这一问题。

【程序】

程序 11-10（I_POWER.C）

```
/* ------------------------------------------------------------ */
/* FUNCTION iterative_power :                                   */
/*   Given two unsigned long data, m and n, this routine        */
/* computes m^n with iterative technique and the number         */
/* of multiplications is still of order O(log(n)) but           */
/* recursion is no more required.                               */
/*                                                              */
/* Copyright Ching-Kuang Shene          July/02/1989            */
/* ------------------------------------------------------------ */

unsigned long iterative_power(unsigned long m, unsigned long n)
{
     unsigned long  temp = 1;
```

Download at http://www.pin5i.com/


```
while (n > 0) {                 /* if there exists 1 bits...    */
    if (n & 0x01UL == 1)        /* the right most one ?         */
        temp *= m;              /* YES, the result get a 'm'    */
    m *= m;                     /* anyway, compute m^k          */
    n >>= 1;                    /* throw away this bit          */
}
return temp;
}
```

问题 2.9 Fibonacci 数非递归解（FIB_IT.C）

Fibonacci 数列 $f_1, f_2, \cdots, f_n$ 的定义是这样的：

$$f_n = \begin{cases} 1 & n=1 \\ 1 & n=2 \\ f_{n-1}+f_{n-2} & n>2 \end{cases}$$

请不要用递归的方法，也不用数组，写一个函数，它接收 n 的值，传回 f_n。

【解答】

如果程序保存了 f_{i-1} 与 f_{i-2}，那么要算 f_i 时，把 f_{i-1} 与 f_{i-2} 相加即可，这样程序就有了 f_{i-2}、f_{i-1} 与 f_i；下一个循环要算 $f_{i+1}=f_i+f_{i-1}$，所以就得保留 f_i 与 f_{i-1} 而把 f_{i-2} 丢掉。程序中，f_0 与 f_1 分别代表 f_{i-2} 与 f_{i-1}，而 temp 则是 f_i，于是当 f_i 算出来后就要调整 f_0 与 f_1 的值，让新的 f_0 是 f_{i-1}，新的 f_1 是 f_i。

【习题】

（1）如果用传统的递归办法，也就是如程序 11-11 所示。

【程序】

程序 11-11

```
unsigned long fibonacci (int n)
{
    if (n <= 2)
        return 1;
    else
        return fibonacci(n-1)+fibonacci(n-2);
}
```

请用 n=13 作输入，用手算方式看看有哪些计算是重复做过的。

（2）（续上题）请问，算 f_n 时一共用了多少个加法，请导出一条公式。

（3）k 阶的 Fibonacci 数 $f_1^k, f_2^k, \cdots, f_i^k, \cdots$ 如下式定义：

$$f_1^k = f_2^k = \cdots = f_k^k = 1$$

$$f_i^k = f_{i-1}^k + f_{i-2}^k + \cdots + f_{n-k+1}^k \quad (i>k)$$

Download at http://www.pin5i.com/

换言之，f_j^k 是前 k 个 f_j^k 的和，因此当 k=2 时就是一般的 Fibonacci 数列。请不要用递归，或许可以用数组，写一个函数 fibK(n,k)，接收 n 与 k，算出 f_j^k。

（4）为什么函数值要用 unsigned long 而不用 int 呢？请说出看法。

【程序】

程序 11-12（FIB_IT.C）

```
/* ------------------------------------------------------------ */
/* FUNCTION fibonacci :                                         */
/*   This program uses two aux. variables to hold two           */
/* intermideate results and thus achieves linear time.          */
/*                                                              */
/* Copyright Ching-Kuang Shene      July/02/1989                */
/* ------------------------------------------------------------ */

unsigned long  fibonacci(int n)
{
     unsigned long  f0, f1, temp;
     int  i;

     if (n <= 2)
         return 1;
     else {
         for (f0 = f1 = 1, i = 3; i <= n; i++) {
             temp = f0 + f1;     /* temp = f(n-2)+f(n-1)        */
             f0   = f1;          /* f0   = f(n-2)               */
             f1   = temp;        /* f1   = f(n-1)               */
         }
         return f1;
     }
}
```

问题 2.10　快速 Fibonacci 数算法（FIB_MT.C）

续 Fibonacci 数列问题。在非递归的 Fibonacci 程序中，在算 f_n 时最多不超过 n-2 个加法，请编写一个速度更快的程序，或许可以用乘法。如果每一个乘法用 m 单位时间，每一个加法用 p 单位时间，于是非递归的写法因为最多有 n-2 个加法，因此最多用(n-2)p 时间。请问，改良程序要用多少时间？假设只考虑加法与乘法而已。

【解答】

如果把矩阵：

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$$

Download at http://www.pin5i.com/

连续乘 n-2 次，那么就不能算是一个好程序员，因为浪费了宝贵的 CPU 时间，其实求该矩阵的 n-2 次方，不就与算 x^{n-2} 是一样的吗？换言之，令上面的矩阵是 M，那么：

$$M^n = \begin{cases} \text{单位矩阵} & n=0 \\ (M^k)^2 & n=2k \\ M\cdot(M^{2k}) & n=2k+1 \end{cases}$$

因此用递归的方式，就像求 x^n 那样，可以最多用 $2\log_2 n$ 次矩阵相乘就可以得到 m^n 了。

一般而言，2×2 的矩阵相乘要有 8 个乘法，4 个加法（用其他算法可以少一些），因此最多乘 $2\log_2 n$ 次的总时间最多不超过 $2\log_2 n\,(8m+4p)$。当 n 非常大之后，$n > \log_2 n$；因为 8m+4p 是一个与 n 无关的常数，所以当 n 非常大时，此方法所用的时间就小于(n-2)p，所以速度是会比较快的。

【程序说明】

程序中有一个叫做 matrix_power()的程序，它会求出一个 2×2 矩阵的 n 次方之后，放在另一个矩阵中返回来。因为 2×2 的矩阵相乘非常简单，所以不使用专一矩阵的方式，而把矩阵元素写在参数中，并且直接相乘。

Matrix_power()的写法与用分而治之策略求 x^n 是完全一样的，请参考求乘幂的问题的说明。

【习题】

（1）请扩充这个方法，让它可以算出 k 阶 Fibonacci 数。

（2）（续上题）请评估一下上一个习题程序的时间值，当 k 愈来愈大时，计算时间会以什么方式增加；这个方法与上一个问题中习题（3）的做法相比又如何呢？

（3）在上一题与本题的解答中分别有两个计算 Fibonacci 数的程序，一个很短、很简单，但另一个却很长、很复杂，请用比较大的 n，并且通过反复计算 1000 次的方式，看看哪一个程序比较快。

【程序】

程序 11-13（FIB_MT.C）

```
/* ------------------------------------------------------------ */
/* FUNCTION fib_mt :                                            */
/*   Probably, this is the most natural way to speed up         */
/* to speedup Fibonacci number generation.  Almost all          */
/* books on algorithm design and analysis mention this          */
/* technique in some details.  This method converts the         */
/* computation of Fibonacci number to a sequence of             */
/* matrix multiplication as discussed in my article.            */
/*                                                              */
/* Copyright Ching-Kuang Shene          June/14/1989            */
/* ------------------------------------------------------------ */
```

Download at http://www.pin5i.com/


```
void matrix_power(unsigned long, unsigned long,
                  unsigned long, unsigned long, int,
                  unsigned long *, unsigned long *,
                  unsigned long *, unsigned long *);


unsigned long  fibonacci(int  n)
{
     unsigned long  a, b, c, d;

     if (n <= 2)
         return 1;
     else {  /* A^(n-2) of a matrix A        */
         matrix_power(1UL, 1UL, 1UL, 0UL, n-2, &a, &b, &c, &d);
         return a + b;
     }
}


/* ------------------------------------------------------------    */
/* FUNCTION matrix_power :                                          */
/*   This function accepts a matrix X and computes                  */
/* X^(n-2) for n >= 3 using the power computation method,           */
/* where X is defined as follows                                    */
/*                                                                  */
/*   +-          -+  +-          -+(n-2)                            */
/*   | aa    bb  |   | a     b   |                                  */
/*   |           | = |           |                                  */
/*   | cc    dd  |   | c     d   |                                  */
/*   +-          -+  +-          -+                                 */
/* ------------------------------------------------------------    */

void matrix_power(unsigned long a,   unsigned long b,
                  unsigned long c,   unsigned long d, int n,
                  unsigned long *aa, unsigned long *bb,
                  unsigned long *cc, unsigned long *dd)
{
     unsigned long  xa, xb, xc, xd;

     if (n == 1)
         *aa = a, *bb = b, *cc = c, *dd = d;
     else if (n & 0x01 == 1) {  /* n odd: X^(2k+1)=X*X^(2k)     */
         matrix_power(a, b, c, d, n-1, &xa, &xb, &xc, &xd);
         *aa = a*xa + b*xc;
         *bb = a*xb + b*xd;
         *cc = c*xa + d*xc;
```

Download at http://www.pin5i.com/


```
        *dd = c*xb + d*xd;
    }
    else {              /* n even: X^(2k)=(X^(2k))^2          */
        matrix_power(a, b, c, d, n>>1, &xa, &xb, &xc, &xd);
        *aa = xa*xa + xb*xc;
        *bb = xa*xb + xb*xd;
        *cc = xc*xa + xd*xc;
        *dd = xc*xb + xd*xd;
    }
}
```

问题 2.11 扩充 Fibonacci 数（EX_FIB.C）

定义一组叫做扩充的Fibonacci数如下——已知X与Y两个数，于是扩充Fibonacci数F_i为：

$$F_i = \begin{cases} 1 & i=0 \\ 1 & i=1 \\ X \cdot F_{i-2} + Y \cdot F_{i-1} & i>1 \end{cases}$$

因此，当X=Y=1时，这一组扩充的Fibonacci数就变成一般的Fibonacci数了，现在请写一个函数，接收一个n值，不用数组与递归的方法算出下面的结果：

$$F_0 \cdot F_n + F_1 \cdot F_{n-1} + \cdots + F_i \cdot F_{n-i} + \cdots + F_{n-1} \cdot F_1 + F_n \cdot F_0$$

本题出自 Jan L.A.van de Snepscheut。

【解答】

为了方便起见，令：

$$A_n = F_0 \cdot F_n + F_1 \cdot F_{n-1} + \cdots + F_{n-1} \cdot F_1 + F_n \cdot F_0$$

自然$A_0 = F_0 \cdot F_0 = 1$，$A_1 = F_0 \cdot F_1 + F_1 \cdot F_0 = 1+1=2$。那么$A_{n+1}$是多少呢？简单，一系列冗长的演算而已。

$$\begin{aligned} A_{n+1} &= F_0 \cdot F_{n+1} + F_1 \cdot F_n + \cdots + F_n \cdot F_1 + F_{n+1} \cdot F_0 \\ &= \left(\sum_{i=0}^{n-1} F_i \cdot F_{n+1-i} \right) + F_n \cdot F_1 + F_{n+1} \cdot F_0 \\ &= \left(\sum_{i=0}^{n-1} F_i (X \cdot F_{n-1-i} + Y \cdot F_{n-i}) \right) + F_n + F_{n+1} \end{aligned}$$

注意，上式最后两项因为$F_0 = F_1 = 1$而化简了，中间那一项则是用了F_{n+1-i}的定义式。接下来：

$$A_{n+1} = X \cdot \left(\sum_{i=0}^{n-1} F_i \cdot F_{n-1-i} \right) + Y \cdot \left(\sum_{i=0}^{n-1} F_i \cdot F_{n-i} \right) + F_n + F_{n+1}$$

但是，因为$\sum_{i=0}^{n-1} F_i \cdot F_{(n-1)-i}$正好就是$A_{n-1}$，所以上式化简成：

Download at http://www.pin5i.com/

$$A_{n+1}=X\cdot A_{n-1}+Y\cdot\left(\sum_{i=0}^{n-1}F_i\cdot F_{n-i}\right)+F_n+F_{n+1}$$

再看括号中的内容：

$$\begin{aligned}\sum_{i=0}^{n-1}F_i\cdot F_{n-i}&=\left(\sum_{i=0}^{n-1}F_i\cdot F_{n-i}\right)+F_n\cdot F_0-F_n\cdot F_0\\&=\left(\sum_{i=0}^{n-1}F_i\cdot F_{n-i}+F_n\cdot F_0\right)-F_n\cdot F_0\\&=\left(\sum_{i=0}^{n}F_i\cdot F_{n-i}\right)-F_n\\&=A_n-F_n\end{aligned}$$

最后得到：

$$A_{n+1}=X\cdot A_{n-1}+Y\cdot(A_n-F_n)+F_n+F_{n+1}$$

于是，就可以令 $n=2,3,\cdots$增加，每次保存 F_{n-1}、F_n，以及 A_{n-1} 与 A_n。用 F_{n-1} 与 F_n 来算 F_{n+1}，有了 F_{n+1} 之后再用 A_{n-1}、A_n、F_n 与 F_{n+1} 算 A_{n+1} 就行了。这才是写程序用的式子。

【程序说明】

程序中用 ext_fibonacci(n)算出 A_n。它是前面不用递归方式求 Fibonacci 数程序的一个补充，仍然用 f_1 与 f_0 来保存 F_{n-1} 与 F_n，最多用 a_1 与 a_0 保存 A_{n-1} 与 A_n，其他都没有差异，请自己研究程序。

【习题】

（1）请问在给了 n 之后这个程序一共用了多少个乘法，多少个加法？

（2）请用数组的功能改写这个程序，请问效率会有所改善吗？请证实此想法；如果没有，用数组岂不没多大意义？

（3）这个题目不过是 Fibonacci 数的扩充，而用矩阵可以得到很高效率的程序，那么有没有办法用矩阵的方式来解这个题目呢？提示：请考虑下式并化简。

$$\begin{aligned}A_{n+1}&=X\cdot A_{n-1}+Y\cdot(A_n-F_n)+F_n+F_{n+1}\\&=X\cdot A_{n-1}+Y\cdot A_n+(Y-1)F_n+F_{n+1}\end{aligned}$$

这个解法来自 Jan Tijmen Udding。

（4）请仔细研究一下程序，或使用的式子，找出可以省略下来的运算，并且改良此程序（这是一个极为简单的练习，应该试一试）。

【参考文献】

这个题目取自 M.Rem 的专栏。据 M.Rem 说，它源自 San L.A van de Snepscheut。

1 M.Rem. Small Programming Exercise, *Science of Computer Programming*. 1983(3): pp.217～222

另外，有关 Fibonacci 数的计算方面的详细讨论与其他细节，请参看：

2 冼镜光. 计算 Fibonacci 数——有效率地解决问题之三，信息传真微电脑版. 1989(8): No.21, pp.146～163

Download at http://www.pin5i.com/

【程序】

程序 11-14（EX_FIB.C）

```
/* ----------------------------------------------------------------- */
/* FUNCTION ext_fibonacci :                                          */
/*   This program extends the Fibonacci number a little              */
/* bit (see the accompanied text please.  This is a                  */
/* modification of the iterative version of Fibonacci                */
/* Number computation.  But you can also use the more                */
/* efficient matrix version.                                         */
/*                                                                   */
/* Copyright Ching-Kuang Shene      July/02/1989                     */
/* ----------------------------------------------------------------- */

unsigned long ext_fibonacci(int n, int x, int y)
{
     unsigned long  f0, f1;
     unsigned long  a0, a1;
     unsigned long  temp1, temp2;
     int  i;

     if (n == 0)
         return 1;
     else if (n == 1)
         return 2;
     else {
         for (f0 = f1 = 1, a0 = 1, a1 = 2, i = 2; i <= n; i++) {
             temp1   = x*f1 + y*f0;
             f0  = f1;
             f1  = temp1;
             temp2   = x*a0 + y*(a1 - f0) + f0 + f1;
             a0  = a1;
             a1  = temp2;
         }
         return a1;
     }
}
```

Download at http://www.pin5i.com/

问题 2.12　二项式系数加法解（CNR_ADD.C）

请写一个程序，只用加法，求出 n 中取 r 个的组合系数 C(n,r)，并且尽可能地使加法数目降低。

【解答】

只要把用递归方法算 C(n,r) 时所经过的途径找出来（不过不去计较经过多少次），那么答案就出来了。以求 C(8,3)为例，递归的程序会最先从 C(8,3)起，先是递归调用 C(7,2)与 C(7,3)，在 C(7,2)会调用 C(6,1)与 C(6,2)，在 C(7,3)调用 C(6,2)，C(6,3)，如果把递归调用的路径仔细走一次，就会发现它正好经过图 11-10 中有画线段的部分。

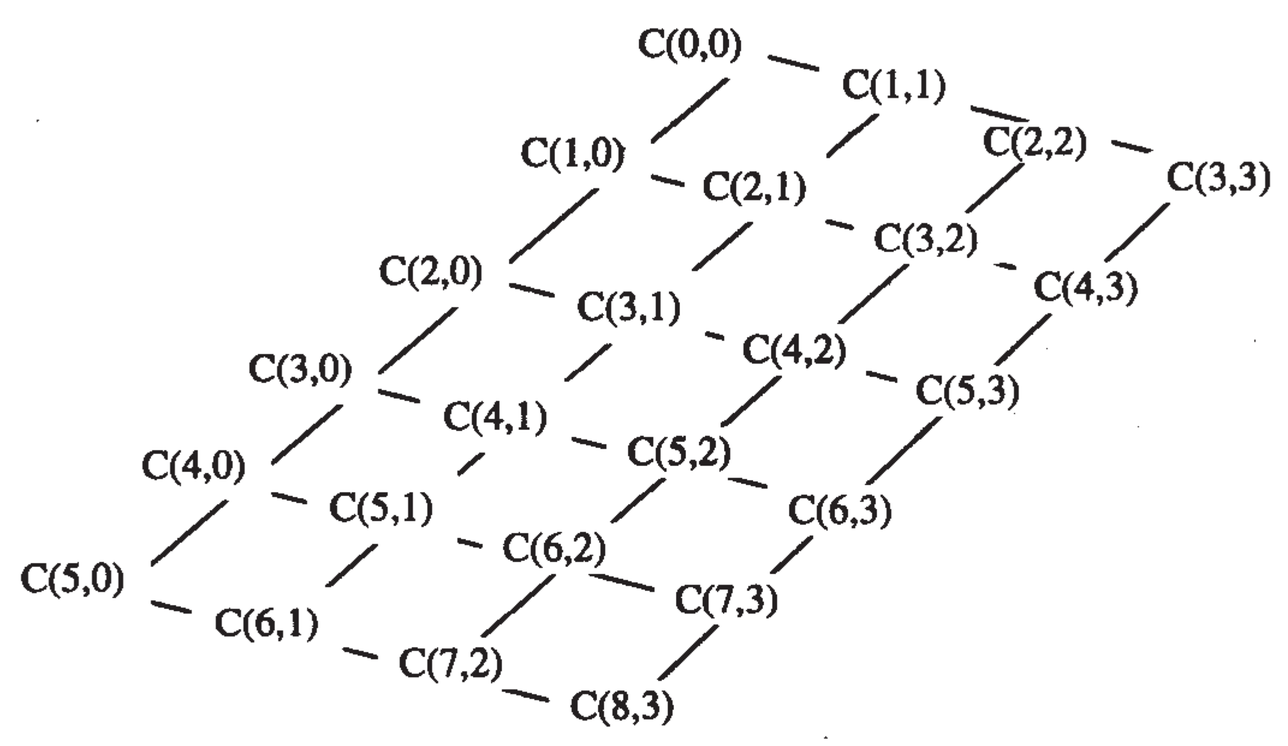


图　11-10

对于每一个 C(n,r) 而言，如果它左上角与右上角的两个元素都算出来了，那么就可以算出 C(n,r)。所以，在图 11-10 中可以从左上到右下一列一列地算，或者是从右上到左下一行一行地算；换句话说，可以算出 C(2,1),C(3,2),C(4,3)，再算 C(3,1),C(4,2),C(5,3),…；但也可以先算 C(2,1),C(3,1),C(4,1),C(5,1),C(6,1)，再算 C(3,2),C(4,2),…,C(7,2)等。

此处打算用一列一列的方式。在开始时准备一个数组 c[ ]，其容量至少有 r，它的内容全是 1，代表图 11-10 中的 C(0,0),C(1,1),…,C(r,r)。接下来，对每一个 c[i]而言，1≤i≤r，令 c[i]为 c[i]+c[i-1]，即为 $C(2,1)=C(1,0)+C(1,1)$，$C(3,2)=C(2,1)+C(2,2)$。注意，在算 C(2,1)时，它要用到 C(1,0)，这个值是 1，目前在 c[0]；也要用到 C(1,1)，现在在 c[1]，因此把 c[0]与 c[1]相加再放回 c[1]，于是 c[1]中的值就变成 C(1,1)了。

程序 CNR_ADD.C 就是用这个观点写成的，在一开始把一个足够大的数组 c[ ]定成 1，然后，就把计算的工作循环做 n-r 次，正好是图 11-10 中的第一列,第二列,…,第 n-r 列的计算；对每一列而言，自第一个元素起把它与前一个元素相加。因此计算完后，c[r]中就是 C(n,r) 的值。

Download at http://www.pin5i.com/

【习题】

（1）请问 CNR_ADD.C 中一共用了多少个加法？是 nr 个吗？还是(n-1)(r-1)个？还是其他，请证明结果。

（2）请以 C(8,3)为例，用手算追踪一下程序执行的动作，来证实的确了解此处所说的方法。

（3）这个程序有一个缺点，基本上可能会做一些重复的工作，这个潜在的问题就是 r 可能会大于 n/2。为什么 r≥n/2 时就有重复的工作呢？请修改程序，把这个重复的工作消除。

（4）请改写程序，一行一行地算。请问一行一行地算与一列一列地算在加法个数上有没有差异？内存需求呢？请问哪一种算法比较好，为什么？

【程序】

程序 11-15（CNR_ADD.C）

```
/* ------------------------------------------------------------ */
/* FUNCTION cnr :                                               */
/*   Given unsigned integers n and r, this function             */
/* computes C(n,r), the Binomial Coefficient of n and r         */
/* by using addition operation only.  The method used           */
/* calculates C(n,r) with exactly (n-1)*(r-1) additions         */
/* and this is fewer than many known techniques.                */
/*                                                              */
/* Copyright Ching-Kuang Shene          July/13/1989            */
/* ------------------------------------------------------------ */


#define  MAXSIZE  100

unsigned long cnr(int n, int r)
{
     unsigned long c[MAXSIZE];
     int i, j;
     for (i = 0; i <= r; i++)          /* initial c[] to all 1's  */
         c[i] = 1UL;
     for (i = 1; i <= n-r; i++)        /* compute C(n,r) by add.  */
         for (j = 1; j <= r; j++)      /* but compute only        */
             c[j] += c[j-1];           /* those needed.           */
     return c[r];
}
```

Download at http://www.pin5i.com/

问题 2.13 快速二项式系数算法（CNR_LOG.C）

看看下面的式子：

$$(2^p+1)^n = C(n,0)+C(n,1)2^p+C(n,2)(2^p)^2+\cdots+C(n,n)(2^p)^n$$
$$=\sum_{i=0}^{n} C(n,i)(2^p)^i$$

如果使用的语言或编译程序（比如说 Ada）有可以计算任意位整数加、减、乘、除的能力的话，能够从这个式子看出一个只用与 $\log_2 n$ 成正比的乘法次数求所有 $C(n,i)$，$i=0,1,\cdots,n$ 的办法吗？在此假定 C 语言有此能力，请写成程序。

【解答】

如果能够先选择好一个合适的 p 值，那么这个问题是易如反掌的。回想一下在课本中所学的，一个正整数（用二进制表示）如果乘以 2^p，那就相当于把那一个数向左移 p 位。如果已经有了 $C(n,0)$，$C(n,1)$，$\cdots$，$C(n,n)$ 这 n+1 个值，$C(n,i)(2^p)^i$ 就相当于把 $C(n,i)$ 向左移 p^i 位，因此 $(2^p+1)^n$ 就是这些移位后的值的总和。用图来看是很容易明白的，在图 11-3 中有斜线的部分表示存放了 $C(n,i)$ 这些值的地方，在它右边的部分是移动的数元数目，正好就是 $(2^p)^i$ 中的 p^i。一般而言，相邻的两个数有些数元会重叠，于是相加时，这些数元就不能直接抄到结果去，而要直接相加。如果把 p 选得足够大，而使得各个 $C(n,i)$ 值所占用的位置是相互错开的，就像如图 11-11 所示。

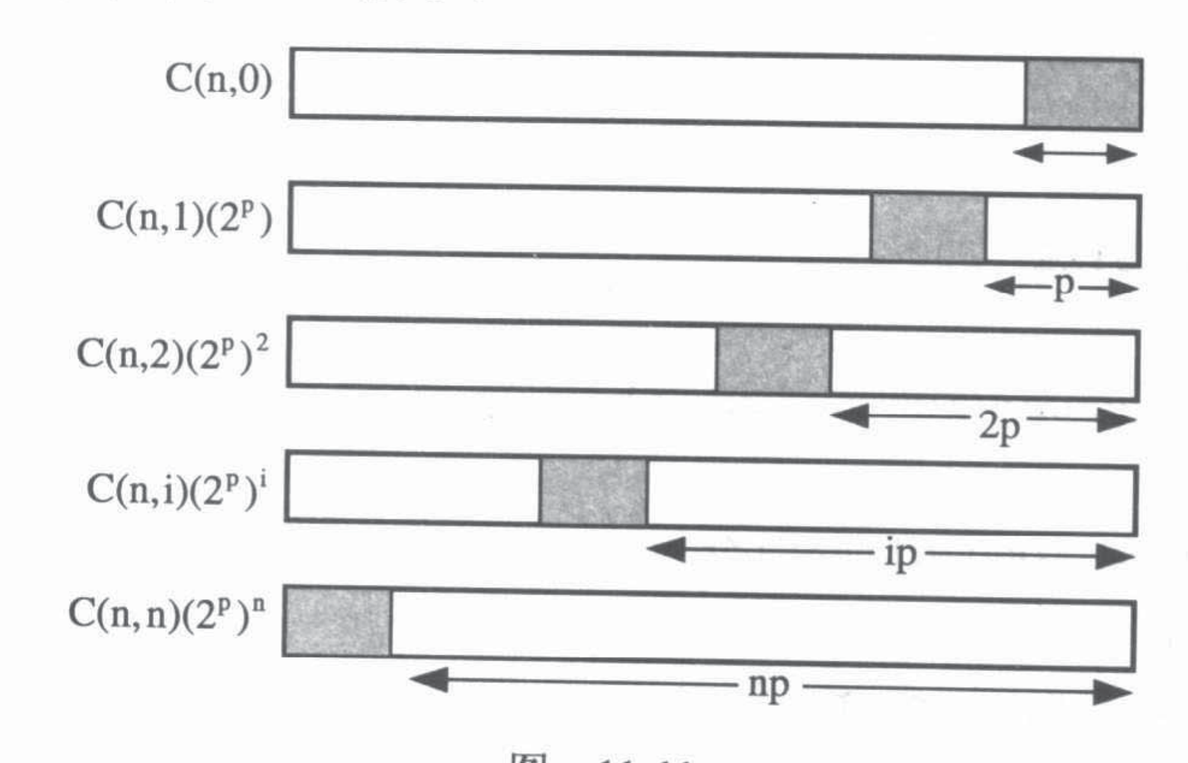


图 11-11

因此把各个值相加的时候就不过是把所有的结果用数元的 OR（|）结合起来而已，所以结果就是如图 11-12 所示。

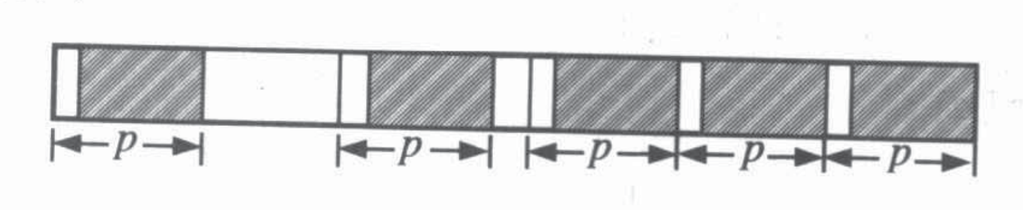


图 11-12

Download at http://www.pin5i.com/

而这正好就是$(2^p+1)^n$的结果。

于是问题就有解答了，如果已经知道了 p，求出2^p只要一步，因为在从右往左算第 p 个数元中存入一个 1，有了2^p之后再加上 1，于是2^p+1的值多半就像是 10000....0001。把这个值叫做 m，于是m^n可以用$\log_2 n$个乘法算出来。有了这个结果之后，把结果右边的 p 个数元取出，它是$C(n,n)=C(n,0)$，再向左取 p 个数元，是$C(n,n-1)=C(n,1)$，再次是 p 个数元，则是$C(n,n-2)=C(n,2)$。一直这样往左处理，每次取 p 个数元，于是依次就有了$C(n,0),C(n,1),C(n,2),\cdots$。这不就是用$\log_2 n$个乘法算出结果了吗？

但是，如何决定 p？下面是一个很粗略的估计。先看下式：

$$\begin{aligned}2^n &= (1+1)^n \\ &= \sum_{i=0}^{n} C(n,i)\cdot 1^i\cdot 1^{n-i} \\ &= C(n,0)+C(n,1)+\cdots+C(n,n)\end{aligned}$$

上式说明2^n大于每一个$C(n,i)$。但是存放2^n要 n 个数元，所以设 n 个数元就可以容纳任何一个$C(n,i)$。于是如果把 p 定成 n，每一个$C(n,i)(2^p)^i$就不会有重叠的位置，于是每 n 位一组的结果就是正确的$C(n,i)$的值了。

【程序说明】

在程序中，为了 C 语言的方便，用 unsigned long 来代替任何位数的整数类型。变量 x 的值就是2^n+1，而 mask 则是2^n-1。事实上它正是在最右边有连续 n 个 1 的一个数，要用它来每次取出 n 个数元作$C(n,i)$的结果。

首先，程序用了 power()函数来计算$(2^n+1)^n$；power()是在前面讲过的、用$\log_2 n$个乘法求m^n的函数。接着计算结果用每 n 位一组把值取出，放到 result[ ]数组中返回给调用它的函数。程序中惟一的一个 for 做的就是这件事。有没有想到过那么简单、漂亮，而且快速的解法呢？

【习题】

（1）有一个叫做 Stirling 公式的式子可以用来算出当 n 很大时 n!的近似值，形式如下：

$$n!\approx\sqrt{2\pi n}\,n^n e^{-n}$$

请用这个式子为 p 求出一个比较精确的估计。当算出结果之后，就会发现用 n 做 p 的估计还不十分离谱呢。

（2）如果机器中 unsigned long 有 32 个数元，请问 n 最多是多少才能得到所有正确的$C(n,i)$值？如果有 k 个数元时，n 又可以是多少呢？请归纳出一道 k 与 n 的关系式，最好是把 n 表示成 k 的函数。要考虑到$n+1$个$C(n,i)$中有将近一半是相同的这种情况。

（3）请写一个程序，使用此处的做法与 CNR_ADD.C 作一个速度上的比较，姑且不理会它们可能会造成溢位的现象。例如，可以用求$C(n,n/2)$来做测试，因为这是各个$C(n,i)$，$i=1,\cdots,n$中值最大的一个，试一试 n=10,20,…,100 甚至更大时，哪一个程序比较快。

（4）为什么上一题说姑且不理会溢位而进行比较呢？是否认为算出来的时间（不是答案）具有代表性？请说明理由。

Download at http://www.pin5i.com/

提示：假设两个 k 位数相加、相乘分别要用 a_k、m_k 的时间，与 n 成正比的加法就用了 $\alpha a_k n$ 时间，与 $\log_2 n$ 成正比的乘法则用了 $\beta m_k \log_2 n$ 时间。后者有没有可能比前者快？如果 k 是个在程序执行之前就固定好的常数又如何？

（5）请问，在实用上此方法与 ADD_CNR.C 相比，有什么缺点，又可能有什么优点？

【参考文献】

这些计算所有组合系数的方法是 1979 年由 Adi Shamir 提出来的。

1 A.Shamir. Factoring Numbers in O(log n) Arithmetic Steps, *Information Processing Letters*. 1979(8), pp.28～31

除了求组合系数之外，这篇文章还谈到如何用 O(logn)个乘法算 n!，以及如何分解因子的技巧，大三、大四程度的朋友估计可以读得懂它。另外，在笔者的一篇专栏中也写过这一类方法的介绍与比较，内容与此有些重复，不过可以作为参考。

2 冼镜光．乘幂、组合系数与阶乘——有效率地解决问题之二，信息传真微电脑版．Vol.8(1989), No.11, pp.140～153

【程序】

程序 11-16（CNR_LOG.C）

```
/* ------------------------------------------------------------ */
/* FUNCTION cnr :                                               */
/*   A special function to calculate all C(n,r)'s given n.      */
/* It employs a very special trick on bit operations.  NOTE     */
/* that depend on the number of bits for unsigned long type     */
/* this function might returns wrong answer.  The rule of       */
/* thumb is that if unsigned long has k bits then the max.      */
/* of n, the input, should be no more than following :          */
/*                                                              */
/*   -1 + sqrt(1 + 4*k)                                         */
/*   ------------------                                         */
/*            2                                                 */
/*                                                              */
/* Thus for k=32 as many microprocessor has, then the max.      */
/* of n is 5.  But this should not be viewd as a restriction    */
/* because the same algorithm can be used in more advanced      */
/* non-numeric applications.                                    */
/*                                                              */
/* Copyright Ching-Kuang SHENE          Feb/24/1989             */
/* ------------------------------------------------------------ */

unsigned long  power(unsigned long, unsigned long);
```

Download at http://www.pin5i.com/


```
void  cnr(int n, int answer[])
{
     unsigned long  x    = (1 << n) + 1;      /* 2^n + 1          */
     unsigned long  mask = (1 << n) - 1;      /* 2^n - 1, n 1's   */
     unsigned long  result;
     int     i;

     result = power(x, (unsigned long) n);  /* (2^n+1)^n        */

     for (i = 0; i <= n; i++, result >>= n) /* retrieve data    */
         answer[i] = result & mask;
}


/* ----------------------------------------------------------      */
/* FUNCTION power :                                                 */
/*   This is the function called 'iterative_power' in               */
/* file I_POWER.C of this book.                                     */
/* ----------------------------------------------------------      */
unsigned long power(unsigned long m, unsigned long n)
{
     unsigned long  temp = 1;

     while (n > 0) {                /* if there exists 1 bits...    */
         if (n & 0x01UL == 1)       /* the right most one ?         */
             temp *= m;             /* YES, the result get a 'm'    */
         m *= m;                    /* anyway, compute m^k          */
         n >>= 1;                   /* throw away this bit          */
     }
     return temp;
}
```

问题 2.14　快速阶乘运算（FACTLOG2.C）

在问题 2.13 中已经看到过与 $\log_2 n$ 个乘法成正比的技巧来算出 C(n, r)。请问，有没有办法利用这个结果设计一个求 n!的程序，它只用与 $(\log_2 n)^2$ 成正比个的乘法。所有有关硬件、语言的假设同上题。

【解答】

解这个问题还是要“分而治之”容易些，如果 n 是个偶数，于是：

Download at http://www.pin5i.com/

$$n! = \frac{n!}{(\frac{n}{2})!(\frac{n}{2})!} \cdot \left[\left(\frac{n}{2}\right)!\right]^2$$
$$= C(n, n/2) \cdot \left[\left(\frac{n}{2}\right)!\right]^2$$

于是算 n!一下就降成算 (n/2)!，因此当 n 是偶数时，可以用 CNR_LOG.C 这个程序去算出 C(n, n/2)，这用了与 $\log_2 n$ 成正比个乘法，就说是 $\alpha\log_2 n$ 个好了。接着再递归调用自己去算 (n/2)!。如果 n 是奇数，那么可以模仿求 m^n 的技巧，把奇数的因子提出来：

$$n! = n \cdot (n-1)!$$

于是 $n-1$ 就是个偶数，把它的结果用上面方法算出，再补乘一个 n，结果就得到了。对于任何一个 n 而言，最多是一开始是奇数，所以用第一个式子（一个乘法），使 $n-1$ 是偶数；到了偶数之后就可以得到它的一半；最坏的情况是，一半之后又是个奇数，所以又用第二式算，算完了再用第一式。但是，这最多只会重复 $2\log_2 n$ 次，所以用了 $2\log_2 n$ 个乘法；在偶数的部分算 C(i, i/2) 时要用大约 $\alpha\log_2 n$ 个乘法，因此上面的程序最多用 $(2\log_2 n)(\alpha\log_2 n) = 2\alpha(\log_2 n)^2$，也就是与 $(\log_2 n)^2$ 成正比个的乘法，就求出了 n!。

【习题】

（1）请用手算，利用上面的程序逻辑，求出 15!与 26!；不过不必把答案算出来，主要是用括号表达出程序运行时的计算次序，并且算出它用了多少个乘法。如果会的话，画成树状结构或许会清楚些。

（2）请仔细研究这个程序，它究竟有没有被重复计算的地方，请说明理由。

（3）如果不去测试 n 是奇数还是偶数，而直接用：

$$n! = C(n, n/2) \cdot (n - n/2)!$$

来计算，请问会不会有什么差异？会不会比较快呢？请仔细分析这两个做法。

【程序】

程序 11-17（FACTLOG2.C）

```
/* ------------------------------------------------------------ */
/* FUNCTION  factorial :                                        */
/*   This function uses a divide-and-conquer method to          */
/* compute n!, but employs the bit manipulation technique       */
/* calculate C(n,n/2).  Thus this still maintain O(logN)        */
/* feature, but because there are O(logN) steps in              */
/* decomposing n! the actual complexity is therefore            */
/* O((logN)^2).                                                 */
/*                                                              */
/* Copyright Ching-Kuang SHENE          Feb/25/1989             */
/* ------------------------------------------------------------ */

unsigned long  cn2(unsigned long);
```

Download at http://www.pin5i.com/


```
unsigned long  factorial(unsigned long n)
{
     unsigned long  temp;

     if (n == 1)                          /* if n = 1, n! = 1        */
         return 1;
     else if (n & 0x01UL == 1)            /* if n odd, n!=n*(n-1)!   */
         return n * factorial(n - 1);
     else {
         temp = factorial(n >> 1);    /* if n even, use C(*,*)   */
         return cn2(n) * temp * temp;/* blend up.                */
     }
}


/* ----------------------------------------------------------------- */
/* FUNCTION cn2 :                                                    */
/*   This function is adopted from a special routine using           */
/* bit manipulation technique to compute C(n,r).  See notes          */
/* therein please.                                                   */
/* ----------------------------------------------------------------- */

unsigned long  power(unsigned long, unsigned long);

unsigned long  cn2(unsigned long n)
{
     unsigned long  x    = (1 << n) + 1; /* 2^n + 1                  */
     unsigned long  mask = (1 << n) - 1; /* 2^n - 1, n 1's           */

     return (power(x, n) >> ((n >> 1)*n)) & mask;
}

/* ----------------------------------------------------------------- */
/* FUNCTION power :                                                  */
/*   Given two unsigned long data, m and n, this routine             */
/* computes m^n with iterative technique and the number              */
/* of multiplications is still of order O(log(n)) but                */
/* recursion is no more required.                                    */
/* ----------------------------------------------------------------- */

unsigned long power(unsigned long m, unsigned long n)
```

Download at http://www.pin5i.com/


```
{
    unsigned long  temp = 1;

    while (n > 0) {                 /* if there exists 1 bits...    */
        if (n & 0x01UL == 1)        /* the right most one ?         */
    temp *= m;                      /* YES, the result get a 'm'    */
        m *= m;                     /* anyway, compute m^k          */
        n >>= 1;                    /* throw away this bit          */
    }
    return temp;
}
```

问题 2.15　更快的阶乘算法（FACTLOG.C）

请仔细研究问题 2.14，开发出一个只用与 $\log_2 n$ 成正比的乘法个数、计算 n!的程序。

【解答】

上一个程序（FACTLOG2.C）在算 C(n,n/2)时的确有重复计算的现象。例如，在算 29!时，依次会有下面的推论。

29!=29•28!
=29[C(28,14)•(14!)2]
=29•C(28,14)•(14!)2
=29•C(28,14)•[C(14,7)•(7!)2]2
=29•C(28,14)•C(14,7)2•(7!)4
=29•C(28,14)•C(14,7)2• (7•6!)4
=29•C(28,14)•C(14,7)2•74•(6!)4
=29•C(28,14)•C(14,7)2•74•[C(6,3)•(3!)2]4
=29•C(28,14)•C(14,7)2•74•C(6,3)4•(3!)8
=29•C(28,14)•C(14,7)2•74•C(6,3)4•38•(2!)8
=29•C(28,14)•C(14,7)2•74vC(6,3)4•38•C(2,1)8•[(1!)2]8

所以一共算了 C(28,14),C(14,7),C(6,3),C(2,1)这 4 个组合系数（注意，它们都有 C(n, n/2)的形式）。但 C(n, n/2) 是如何算的呢？看前面讲解算 C(n, r) 的题目，就会发现它是通过求 $(2^n+1)^n$ 而得来的。于是在算 x^{28} 时有：

$$
\begin{aligned}
x^{28} &= (x^{14})^2 \\
&= [(x^7)^2]^2 \\
&= ((x \cdot x^6)^2)^2 \\
&= ((x \cdot (x^2)^3)^2)^2
\end{aligned}
$$

Download at http://www.pin5i.com/

因此算 x^{28} 时要用到 x^{14}，算 x^{14} 时用到 x^6，而算 x^6 时用到 x^2。如果把 x 换成 2^n+1，那么在算 x^{14} 时不正好是求 $(2^n+1)^{14}$，于是就算出了 C(14,i) 的各个值，每一个值占 n 位。

由这一特性，看到了一线转机，当要求 C(n, n/2) 而用 $(2^n+1)^n$ 来计算时，在计算过程中会测试目前的 x^k 中 k 是否是偶数，如果是，就把 x^k 的暂时性结果的中央 n 位取出来，因为 x^k 的值（也就是 $(2^n+1)^k$ 的值）正好是所有的 C(k,0)，C(k,1)，…，C(k, k)，所以取出中央 n 位不就是 C(k, k/2) 吗？这些值就存放在一个数组中。以后，当用递归求 (n/2)!，再用到 C(k, k/2) 时就可以从该数组取出来了。

在程序 FACTLOG.C 中，factorial()就是用来算 n!的。它先调用一个经过修改过的 power()函数去递归地计算 $(2^n+1)^n$ 的值，并且保存某些中间结果；这些结果存放在一个叫做 cnrs[ ]的数组中，然后调用 factor()函数，用上一题的方法求 n!，不过差异就是这一次不用每次都调用 cnr()函数了，当要用到某个 C(k, k/2) 时，就从 cnrs[ ]数组中把它取出来。

修改过的 power()函数是用递归写成的，不过用反复的方法也不会有任何困难。它是用来算 x^m 的，每次当 m 是偶数时，就把 x^m 的结果中间的 n 个数元取出来，存放到 cnrs[]数组的下一个位置。程序其他的地方就简单了，不打算就此作进一步说明。不过，这个程序只进行了与 $\log_2 n$ 成正比个乘法而已。为什么？用 power()去算 $(2^n+1)^n$ 用了 $\alpha\log_2 n$ 个乘法（亦即与 $\log_2 n$ 成正比），用 factor()算 C(n, n/2) 与 (n/2)! 递归地算阶乘时，只会有大约 $\log_2 n$ 次的递归调用，每一次的乘法数目是个常数（奇数时一次，偶数时两次），所以最多不过是 $\alpha\log_2 n$ 个乘法，综合起来，也不过是 $(\alpha+2)\log_2 n$ 个乘法而已，因此程序只用了与 $\log_2 n$ 成正比个乘法。

【习题】

（1）请以 29!为例，用手算追踪上面的程序；请问 cnrs[]中的内容有什么？

（2）请问，被存到 cnrs[]中的是哪些 C(k, k/2)。换句话说，能否找出在求 $(2^n+1)^n$ 时的 k 与那些 C(k, k/2) 的关系。程序中已经写得很明显了，但是此处所期望的是能够证实它。

（3）请用反复方式的 power()函数（参看 I_POWER.C）来取代此处的递归写法，要如何修正这个 power()函数呢？

（4）对已知的 n 而言，请仔细研究 power()与 factor()函数，请问它们最多会用到多少个乘法？这个情况在何时发生？于是整个程序最多会用多少个乘法？假设程序其他部分在每一个乘法时都要有 α 时间的准备工作（比如递归与其他叙述），每一个乘法要 β 时间，于是这个程序一共用了多少时间？又若用传统的连乘法，对每一个乘法而言，要有 γ 的准备时间（比如循环），于是连乘方法要多少时间？$\log_2 n$ 法在什么情况下会比连乘法快，也就是说 n 至少要是多少？

【参考文献】

请参考问题 2.13 讲求 C(n, r) 时的文献说明。此外，Peter B.Borwein 在 1985 年也提出了另一个不同的方法。在他的文章中指出，如果 M(n) 是有 n 个（十进制）数相乘所用的乘法个数，那么求 n!的乘法个数就与 $\log_2\log_2 n \cdot M(n\log_2 n)$ 成正比。他的方法不容易说明，但对无限精确度的算术相当有用，有兴趣的朋友可参看：

Download at http://www.pin5i.com/

P.B.Borwein. On the Complexity of Calculating Factorials, Journal of Algorithms. 1985(6), pp.376～380

【程序】

程序 11-18（FACTLOG.C）

```
/* ------------------------------------------------------------ */
/* FUNCTION factorial :                                         */
/*   This is an O(logN) function for calculating n!.  It        */
/* combines the O(logN) power concept and divide-and-conquer    */
/* technique.  NOTE that the improvement is that all            */
/* C(n,n/2) needed are computed along with the (2^n+1)^2.       */
/* Thus if we can save these C(n,n/2)'s it is not necessary     */
/* to call cnr(*,*) each time when C(n,n/2) is computed.        */
/* NOTE also this function works fine for n <= 7 only on a      */
/* 32 bit unsigned long computer and for these small numbers    */
/* this function is slower than the others.  But remember       */
/* that the meaning of O(logN) is for large N's, this means     */
/* we will have bigger speed gain when n is getting larger.     */
/*                                                              */
/* Copyright Ching-Kuang SHENE          Feb/25/1989             */
/* ------------------------------------------------------------ */

unsigned long  mask;              /* mask for shifting bits      */
unsigned cnrs[100];               /* storage for C(n,n/2)'s      */
int        number;                /* index for cnrs[]            */
int        p_size;                /* partition size in C(n,n/2)  */

unsigned long  factor(unsigned long);
unsigned long  power(unsigned long, unsigned long);
/* ------------------------------------------------------------ */
/* FUNCTION factorial :                                         */
/*   Control routine of n! calculation.                         */
/* ------------------------------------------------------------ */

unsigned long  factorial(unsigned long n)
{
     unsigned long  x = (1 << n) + 1;

     number = 0;                  /* set index to init. pos.     */
     mask   = (1 << n) - 1;       /* set up mask for shifting    */
     p_size = n;                  /* set up partition size       */
     (void) power(x, n);          /* now computing (2^n+1)^n     */
```

Download at http://www.pin5i.com/


```
    number = 0;                 /* reset index to the start     */
    return factor(n);           /* then compute n!              */
}


/* ----------------------------------------------------------    */
/* FUNCTION factor :                                             */
/*   This is the working routine for n!.  The idea is            */
/*                                                               */
/*          +-  1                    if n = 1                    */
/*          |                                                    */
/*   n!  =  +   n * (n-1)!           if n is odd                 */
/*          |                                                    */
/*          +-  C(n,n/2)*[(n/2)]!    if n is even                */
/*                                                               */
/* Before factor() is called n_power() is used to compute        */
/* (2^n+1)^n and during the course of this computation, it       */
/* stores all relevant C(n,n/2) into array cnrs[].  Thus         */
/* C(n,n/2)'s are available from array C(n,n/2) here.            */
/* ----------------------------------------------------------    */

unsigned long  factor(unsigned long n)
{
    unsigned long  temp;

    if (n == 1)
        return 1;
    else if (n & 0x01UL == 1)
        return n * factor(n - 1);
    else {
        temp = factor(n >> 1);
        return cnrs[number++] * temp * temp;
    }
}


/* ----------------------------------------------------------    */
/* FUNCTION power :                                              */
/*   The modified power function to compute (2^n+1)^n in         */
/* recursive mode.  During its computation, n_power() will       */
/* save C(n,n/2) into array cnrs[].                              */
/* ----------------------------------------------------------    */
```

Download at http://www.pin5i.com/


```
unsigned long  power(unsigned long n, unsigned long m)
{
     unsigned long  temp;

     if (m == 1)
         temp = n;
     else if (m & 0x01UL != 0)
         temp = n * power(n, m-1);
     else {
         temp = power(n, m >> 1);
         temp *= temp;
         cnrs[number++] = (temp >> ((m >> 1)*p_size)) & mask;
     }
     return temp;
}
```

问题 2.16　连续整数的固定和（GIVENSUM.C）

编写一个程序，读入一个正整数，把所有连续的、和为给定的正整数的正整数找出来。例如，如果输入是 27，于是发现 2～7、8～10、13 与 14 的和是 27，这就是解答。如果输入是 10000，应该有 18～142、297～1328、388～412、1998～2002 这 4 组。注意，不见得一定会有答案，例如 4、16 就无解；另外，排除只有一个数的情况，否则，每一个输入就都至少有一个答案，就是它自己。

【解答】

这个题目的解法很多，但都不外是在题目说明中的提示的变形，在此所采用的方法是这样的：

先求出一个和，sum 是 1+2+…+i；并且 sum 是第一个大于等于给定的值（令为 n）的和。例如，若 n 为 27，那么 sum 就是 1+2+3 +4+5+6+7=28。

于是，若和 i+(i+1)+…+j 比 n 大，就把和去掉小的那一端而变为(i+1) +…+j。如果和比 n 小，就加上(j+1)而变成 i+(i+1) +…+j+ (j+1)。

如果和等于 n，就把 i 与 j 显示出来，因为 i+…+j 是一组解；接着把和变大，亦即加上 j+1 而成为 i+(i+1)+…+j+(j+1)。

上述两个步骤就一直反复执行，直到 i 比 n/2 大为止（见第一点提示）。

在表 11-2 中就是用这个观点来求出 27 的连续整数和的例子。可以发现它一共用了 22 个加法。注意一点，虽然不能确定何时增加 i，但 i 一定小于或等于 n/2，所以增加 i 的次数最多是 n/2 次。再看 j，一开始 i 是 1，而 j 一直增加到和大于等于 n 为止，所以最多也是

Download at http://www.pin5i.com/

n/2 次；如果往后再有增加，j 也只会到 n/2+1 止，所以对 j 的增加也是最多 n/2 次。因此，一共有 n/2+n/2=n 次增加 i 或 j 的次数。每一次增加 i 或 j，都会把和加上对应的 j 值或扣除 i 值，所以对和而言最多只有 n 次加减运算，总共就最多不超过 2n 次加减法。与 VERY_BAD 相比，那真是天壤之别了，而这也正是本程序会快的主要原因。

表 11-2

i～j	和	比较结果
1～7	28	>
2～7	27	=
2～8	35	>
3～8	33	>
4～8	30	>
5～8	26	<
5～9	35	>
6～9	30	>
7～9	24	<
7～10	34	>
8～10	27	=
8～11	38	>
9～11	30	>
10～11	21	<
10～12	33	>
11～12	23	<
11～13	36	>
12～13	25	<
12～14	39	>
13～14	27	=
13～15	42	>

【习题】

（1）在程序中，第一个 for 在比较 sum 与 GIVEN（也就是上述的 n）时是用<，而不是<=，为什么？请解释原因。

（2）请看下面的 UNSAVE.C 程序（如程序 11-19 所示），它是在一本书中找到的，程序作者声称该程序可以解决这些问题，其使用的概念与我们的一样。不过，笔者认为这个程序有问题，能否在不执行它的前提下找出问题何在并且改正它吗？

程序 11-19（UNSAVE.C）

```
#include  <stdio.h>
#include  <stdlib.h>
void main(void)
```

Download at http://www.pin5i.com/


```
{
        long  left, right;
        long  sum;
        long  GIVEN;
        char  line[100];

        gets(line);
        GIVEN = atol(line);

        for (sum = 0, left = right = 1; left < GIVEN/2+1; right++) {
             sum += right;
             while (sum > GIVEN) {
                sum -= left;
                left++;
             }
             if (sum == GIVEN)
                printf("\n%ld - %ld", left, right);
        }
}
```

（3）如果要求的是把一个数的连乘积求出来，如何写这个程序呢？例如，给了 20，程序应该报告出 4×5，210 就应该报告 5×6×7 等。

（4）如果有一个正整数数组，元素都已经从小到大排好了。编写一个程序，读入一个值，看看数组中有多少组连续的元素可以加出给定的值。如果数组的元素正好是 1,2,3,4,…，那么这个习题就化成程序 11-20 了。

【程序】

程序 11-20（GIVENSUM.C）

```
/* ------------------------------------------------------------ */
/* PROGRAM consecutive sum :                                    */
/*   Given a positive integer, find all a's and b's such        */
/* that the sum from a to b equals the given number.            */
/*                                                              */
/* Copyright Ching-Kuang Shene          July/03/1989            */
/*                                                              */
/* ------------------------------------------------------------ */

#include  <stdio.h>
#include  <stdlib.h>

void main(void)
{
```

Download at http://www.pin5i.com/


```
    long  left, right;          /* left and right bound        */
    long  sum;                  /* computed sum                */
    long  GIVEN;                /* the given number            */
    int   count = 0;            /* solution counter            */
    char  line[100];            /* input buffer                */
    printf("\nConsecutive sum to a fixed given number");
    printf("\n=====================================\n");
    printf("\nYour number (> 0) please ---> ");
    gets(line);
    GIVEN = atol(line);

    for (sum = 0, right = 1; sum < GIVEN; sum += right, right++)
        ;

    for (left = 1, right--; left <= GIVEN/2; )
        if (sum > GIVEN)            /* if sum too large        */
            sum -= (left++);        /* reduce from left        */
        else {
            if (sum == GIVEN) {     /* if equal, print it      */
                printf("\n%ld = sum from %ld to %ld",
                    GIVEN, left, right);
                count++;            /* and increase count      */
            }                       /* if sum too small        */
            sum += (++right);       /* increase from right     */
        }

    if (count > 0)
        printf("\n\nThere are %d solutions in total.", count);
    else
        printf("\n\nSorry, there is NO solution at all.");
}
```

Download at http://www.pin5i.com/

第 12 章　排列、组合与集合

问题 3.1　列出所有子集（DIRECT.C）

请写一个程序，列出{1,2,3,…,n}这个集合的所有子集，包括空集合ϕ。

【解答】

要把各个元素直接取出来排列是不太容易做到的，所以要另寻它法。下面介绍一个很容易编写程序的做法。

在集合中一共有 n 个元素，对于每一个元素而言，它可以“属于”或“不属于”某个子集。例如，如果有 4 个元素的集合{1,2,3,4}，而又有一个子集{2,4}，那么 1 与 3 不在子集中，而 2 与 4 在子集中；如果用 1 表示在子集内，用 0 表示不在，于是{2,4}就对应着二进制数 0101。反过来，如果是 1011，就表示 1,3,4 在子集中，因而该子集就是{1,3,4}。因此，很明显地，一个有 n 个元素的集合的子集与一个 n 位的二进制数有一对一的关系。为了容易理解，列出{1,2,3,4}所有元素与 4 位的二进制数的对应，如表 12-1 所示。

表　12-1

集　　合	二进制数	集　　合	二进制数
{}	0000	{2,3}	0110
{1}	1000	{2,4}	0101
{2}	0100	{3,4}	0011
{3}	0010	{1,2,3}	1110
{4}	0001	{1,2,4}	1101
{1,2}	1100	{1,3,4}	1011
{1,3}	1010	{2,3,4}	0111
{1,4}	1001	{1,2,3,4}	1111

所有子集共有 2^4 个，不正好就是二进制数为 0 到 1111_2=15 吗？所以，只要能够产生这些二进制数，就可以根据哪几位为 1 把对应的子集找出来。问题是，如何产生二进制数？

产生二进制数并不难，一来可以应用机器的 unsigned 类型，二来可以自己动手做；为了能够多给一些例子，此处采用后者，而把前者留在习题中。二进制的加法如图 12-1 所示。

```
   0        0        1        1
 + 0      + 1      + 0      + 1
 ---      ---      ---      ----
   0        1        1       10
```

图　12-1

Download at http://www.pin5i.com/

所以，如果有一个二进制数，把它加上 1 的效果就相当于从那个数最右边一位起找出第一个 0 的位置，于是这个 0 右边的位数会是 1，如图 12-2 所示。

```
  xxxxx 0 1 1 1 1...1
+                   1
-------------------
  xxxxx 1 0 0 0 0...0
```

图 12-2

当把这个数加上 1 时，0 右边的位数全变成 0，而那个 0 则变成 1，在 0 左边的全都不变；这个说法当 0 本身在最右边时也是对的，希望读者能够了解这一点。所以把一个二进制数加 1 就是：从右往左查，如果遇到的位数是 1，就把它变成 0，再接着处理下一位；但若遇到 0，就把那个位数变成 1，停止工作。

当得到一个二进制数之后，就去查是 1 的是在哪几位，这个意思就是说对应着哪几位的元素在一个子集中，所以把位数显示出来即可。

【习题】

（1）程序是用 char 数组写成的，上面提到，如果用 unsigned 类型（int 或 long），那么加 1 的动作就可以省略，而只需要找出哪几个 bit 是 1。请用这个概念改写程序；请比较一下用数组、unsigned 类型各有什么优劣。

（2）如果集合不是用数字，而是用英文字母 A,B,C,D,…，有没有办法用最少次数的改变来列出所有子集？

【程序】

程序 12-1（DIRECT.C）

```
/* ------------------------------------------------------------ */
/* PROGRAM Subset Listing :                                     */
/*   This program lists all subsets, including empty set,       */
/* of a given set with elements {1,2,...,n} by using a          */
/* direct generation method.                                    */
/*                                                              */
/* Copyright Ching-Kuang Shene          July/04/1989            */
/* ------------------------------------------------------------ */

#include  <stdio.h>
#include  <stdlib.h>

#define   MAXSIZE   20
#define   LOOP      1

void main(void)
{
```

Download at http://www.pin5i.com/


```
    char digit[MAXSIZE];
    int  i, j;
    int  n;
    char line[100];

    printf("\nDirect Generation of All Subsets of a Set");
    printf("\n=========================================");
    printf("\n\nNumber of Elements in the Given Set --> ");
    gets(line);
    n = atoi(line);

    /* ---You'd better check to see if n is too large---      */

    for (i = 0; i < n; i++)     /* clear all digits to 0       */
       digit[i] = '0';

    printf("\n{}");             /* outpout empty set {}        */
    while (LOOP) {
       for (i = 0; i < n && digit[i] == '1'; digit[i] = '0', i++)
           ;                    /* find first 0 position       */
       if (i == n)              /* if none, all pos. are 1     */
          break;                /* thus all elem. are in set   */
       else
          digit[i] = '1';       /* now add one to this pos     */

       for (i = 0; i < n && digit[i] == '0'; i++)
          ;                     /* find first 1 position       */
       printf("\n{%d", i+1};    /* show its numner and         */
       for (j = i + 1; j < n; j++) /* others                    */
          if (digit[j] == '1')
             printf(",%d", j + 1);
       printf(")");
    }
}
```

问题 3.2　列出所有子集——字典顺序（LEXICAL.C）

请写一个程序，用字典顺序（Lexical Order）把一个集合的所有子集找出来。

【解答】

事实上，这是一个十分简单的程序，除了空集合之外，最“小”的一个集合就是{1}，

Download at http://www.pin5i.com/

再下一个就是包含 1，而且再加上 1 的下一个元素（1+1=2）的集合，即{1,2}；下一个元素自然就是含有 1 与 2，并且还有 2 的下一个元素（2+1=3）的集合{1,2,3}了。就这样一直到包含了所有元素为止，亦即{1,2,3,…,n}。下一个集合是谁？绝不是{1,2,3,…,n-1}，因为它比{1,2,3,…,n}小，事实上应该是{1,2,3,…,n-2,n}。为什么呢？在{1,2,3,…,n-1,n}与{1,2,3,…,n-2,n}之间，前 n-2 个元素完全相同，但是 n-1<n，这不就证实了以上说法了吗？

由于以上原因，因此可以用一个数组 set[ ] 来存放集合的元素，一开始时 set[0]=1，表示{1}；另外，用一个变量 position，指出目前最右边的位置何在，在开始时自然就是 1。接下来，就陆续地产生下一个集合了。注意，目前集合中最右边的元素是 set[position]，如果它的值比 n 小，那就表示还可以加进一个元素，就像是从{1,2,3}加上一个元素变成{1,2,3,4}一样（n>4）。这倒是容易做，下一个元素在 set[position+1]，因此在那存入 set[position+1]这个值就行了；同时 position 也向右移一位。如果目前最右边元素 set[position]已经是 n，因而不能再增加元素了。例如，当 n=4 时，如果有{1,3,4}，那自然不能像前面所说的加入一个 5。这时看最右边元素的位置，亦即 position，是不是在第一位（如果 n=6，而现在的集合是{6}），如果不在第一位，那就可以往回移一位，并且把那个位置的值加上 1。例如，如果现在有{1,3,4}，而 n=4；最右边（4）的位置不是在第一位，因而退回一位，等于是{1,3}；但这是不对的，因为{1,3}比{1,3,4}“小”，要做得比{1,3,4}大，把 3 加上 1 而变成{1,4}就行了。如果最右边（4）的位置是在第一位，那么程序就完成了。

【习题】

（1）有 n 个元素的集合的子集个数有 2^n 个，为什么前面所有的程序都不用一个 for 从 1 数到 2^n，而用 break 离开循环呢？（提示：想一想当 n=50 或 100 时会有什么后果）

（2）这个程序稍加改动就可以求出 n 个元素集合中元素个数不超过 m(m≤n)的所有子集，请把它写出来。

（3）这个程序也可以改成求出 n 个元素的集合中元素个数恰好是 m(m≤n)的所有子集，请把它写出来；请验查一下是否恰好有 C(n,m) 个？

注意：在编写（2）与（3）两题时，切不可以把所有子集都求出来，看看元素的个数，如果在所要的范围，就提出该集合。这样的写法是不能接受的，虽然正确；应该在编写本程序的概念上动动脑筋，只产生所要的集合，而不产生任何多余的部分才是正途。

（4）请写一个程序，把 n 个元素的集合的子集，用字典顺序的反顺序列出来（注意，不能保存各个子集），然后用反顺序列出来，因为当 n 很大时内存就不够用了；试一试了解上面所讲的观点，直接把程序列出来。

【程序】

程序 12-2（LEXICAL.C）

```
/* ------------------------------------------------------------------ */
/* PROGRAM All Possible Subset by Lexical Order :                     */
/*   This program generates all subsets by lexical order.             */
/*                                                                    */
/* Copyright Ching-Kuang Shene          July/05/1989                  */
/* ------------------------------------------------------------------ */
```

Download at http://www.pin5i.com/


```
#include <stdio.h>
#include <stdlib.h>

#define  MAXSIZE     20
#define  LOOP         1

void main(void)
{
     int  set[MAXSIZE];
     int  n, i;
     int  position;
     char line[100];

     printf("\nAll Possible Subsets Generation by Lexical Order");
     printf("\n==========================================");
     printf("\n\nNumber of Elements in the Set --> ");
     gets(line);
     n = atoi(line);
     printf("\n{}");                    /* the empty set            */
     position     = 0;                  /* start from the 1st pos.  */
     set[position] = 1;                 /* it gets a '1'            */
     while (LOOP) {                     /* loop until done...       */
        printf("\n{%d", set[0]};        /* print one result         */
        for (i = 1; i <= position; i++)
            printf(",%d", set[i]);
        printf("}");

        if (set[position] < n) {        /* this pos. can be inc     */
           set[position+1] = set[position] + 1;    /* YES           */
           position++;                  /* inc. next pos.           */
        }
        else if (position != 0)         /* NO, the 1st pos?         */
           set[--position]++;           /* backup and increase      */
        else                            /* NO, the 1st pos and can  */
           break;                       /* not be inc. JOB DONE!    */
     }
}
```

问题 3.3　产生 Gray 码（GRAYCODE.C）

请写一个程序，用 Gray 码（Gray Code）的顺序列出一个集合的所有子集。

Download at http://www.pin5i.com/

【解答】

把 n=4 的 Gray 码写出来：

0000 0001 0011 0010 0110 0111 0101 0100

1100 1101 1111 1110 1010 1011 1001 1000

如果位数是从右往左算，最右边一位是 1，就会发现从第一个数到第二个数是第一位变了，第二个数到第三个数是第二位变了，把所有位置写下来有：

1 2 1 3 1 2 1 4 1 2 1 3 1 2 1

于是马上就可以归纳出：第奇数个的数（0000 算是第 0 个）永远是改变上一个值最右边一位而得来的，可能是 0 变成 1，也可能是 1 变成 0；总之，变成了相反的值。

再看第偶数个的数（0011,0110,0101,1100,1111,1010,1001），它们是从改变上一个值的 2,3,2,4,2,3,2 这几位而来；再仔细看看，它们的邻位（1,2,1,3,1,2,1）值都是 1，而且是从右边算过来的第一个 1。因此：第偶数个的数（0000 不计）的值是从改变上一个值中右边算来第一个 1 的左边邻位而得来。

以上是对 n=4 的研究发现，何不看看 n=5 的情况呢？n=5 的 Gray 码是：

00000	00001	00011	00010	00110	00111	00101	00100
01100	01101	01111	01110	01010	01011	01001	01000
11000	11001	11011	11010	11110	11111	11101	11100
10100	10101	10111	10110	10010	10011	10001	10000

请验证一下。

事实上，Gray 码有一个很漂亮的递归解法。如果要求 n 位的 Gray 码，那就先求 n-1 位的；假设 G_{n-1} 表示所有 n-1 位的 Gray 码，于是 n 位的 Gray 码可以用下面方法造出来。

（1）把 G_{n-1} 中各个元素左边都加一个 0。

（2）把 G_{n-1} 中各个元素的顺序反过来排列，并且在前面都加一个 1。

（3）上面两组数据依顺序连接起来就是 Gray 码 G_n。

例如，求 G_4；另外，用 $\overline{G_i}$ 表示把 G_i 用逆序排列的结果。因此为了要算 G_4，先算 G_3，但算 G_3 时又要算 G_2，算 G_2 时用到 G_1。G_1 是最简单的，只有一位，所以 $G_1=\{0,1\}$，有了 G_1 算 G_2 如下：

$$
\begin{aligned}
G_2 &= 0G_1 \cup 1\overline{G_1} \\
&= 0\left\{\begin{matrix}0\\1\end{matrix}\right\} \cup 1\left\{\begin{matrix}1\\0\end{matrix}\right\} \\
&= \{00,01,11,10\}
\end{aligned}
$$

有了 G_2，算 G_3：

$$
\begin{aligned}
G_3 &= 0G_2 \cup 1\overline{G_2} \\
&= 0\left\{\begin{matrix}00\\01\\11\\10\end{matrix}\right\} \cup 1\left\{\begin{matrix}10\\11\\01\\00\end{matrix}\right\}
\end{aligned}
$$

Download at http://www.pin5i.com/

$$= \{000, 001, 011, 010, 110, 111, 101, 100\}$$

于是再算 G_4 就不会有什么问题了：

$$G_4 = 0G_3 \cup 1\overline{G_3}$$

$$= 0\left\{\begin{matrix}000\\001\\011\\010\\110\\111\\101\\100\end{matrix}\right\} \cup 1\left\{\begin{matrix}100\\101\\111\\110\\010\\011\\001\\000\end{matrix}\right\}$$

$$= \{0000, 0001, 0011, 0010, 0110, 0111, 0101, 0100$$
$$1100, 1101, 1111, 1110, 1010, 1011, 1001, 1000\}$$

用这个技巧就不难算出 $G_5, G_6, \cdots$。

初学者也许会问，为什么把 G_i 反过来排列再补上一个 1 呢？道理很简单：先看 G_n，每一个相邻的元素都只有一位不同，因此在左方填一个 0，结果也是只有一位不同，同样地，在左方填一个 1 所得来的结果中，也只会有一位不一样；G_n 的最后一个与第一个也只有一位不同，但若把 $0G_n$ 与 $1G_n$ 相接，那么 $0G_n$ 的最后一个与 $1G_n$ 的第一个就有两个位不一样了，所以把 $0G_n$ 与 $1\overline{G_n}$ 相接，在结果中，G_n 的最后一个与 $\overline{G_n}$ 的第一个完全一样（因为 $\overline{G_n}$ 是把 G_n 反过来排的），因此就只有在最左边的 0 与 1 不同，这样就有了一组 n 位的 Gray 码了。如果有兴趣，不妨用数学归纳法证明前述的两条规则所产生的，正好就是上述的 G_n。

不过，程序并不打算用递归的写法，不用递归也并不困难，看看前述的那两条规则就行了。因为奇数位与偶数位是交错出现的，而且一开始最右边一位是第 0 位，也就是偶数位，所以可以用一个变量 even，最先赋予其值为 1，当处理完一位之后就把它改为 1-even，因为 even 的值就是 1,0,1,0,…地变化；当值为 1 时，表示偶数位，值为 0 时是奇数位。如果目前是偶数位，就把最右边的位值改过来（0 变成 1，1 变成 0）；如果是奇数位，就从右向左找，找出第一个值为 1 的位，把它左边的那一位的值改变。做完之后，把修订过的结果显示出来。

为了程序编写方便，各个位的存放顺序，亦即 digit[0],digit[1], …,digit[n-1] 正好是第 0 位（最左边）,第 1 位,…,第 n-1 位（最左边），所以在显示位的值时要用相反的顺序，且在取出对应的元素来显示时，顺序也是相反的，仔细看看显示 Gray 码与集合用的 for 循环就不难明白。程序的输出有两部分，左半是 Gray 码，右半是对应的集合。

【习题】

（1）记得河内之塔（Towers of Hanoi）这个游戏吗？有 3 个、4 个与 5 个圆片，请用手算方式找出被搬动的圆片号码与 Gray 码有什么关联。请写一个程序，用 Gray 码来解河内之塔这个问题，这样就有了不用递归、没有堆栈的直接解法了。

（2）找一本书看看何谓中国九连环游戏（Chinese Ring Puzzle），把各个环的移动顺序

Download at http://www.pin5i.com/

与环的号码写下来，这一组号码与 Gray 码有什么关系。于是，又有了一个不用递归来解九连环游戏的解法。

（3）是否可以用递归的方法来写 Gray 码程序？会容易些吗？

【附注】

Gray 全名叫做 Frank Gray，是贝尔电话实验室研究物理学家，1969 年过世，这里所提的 Gray 码是他在 1940 年研究出来的，用来在 PCM（Pusle Code Modulation）方法传送信号时避免错误。1953 年 3 月 17 日，Gray 取得美国专利，这是 Gray 码第一次出版的日期。许多书都不曾提及 Gray 其人其事，以上就是一点补充。

读者如果有兴趣，不妨看一看下面的证明，用那两个法则所产生的结果正好就是 G_n，可以用数学归纳法。

当 n=1 时，只有一位，即 0；它是偶数，把最右边那一位的值改变，就得到 1，所以 $G_1=\{0,1\}$。

如果比 n-1 小或等于的 G_i 都可以用那两条规则造出来，要证明 G_n 亦然，对于 G_n 而言，有 n 位，对这 n 位的右边 n-1 个而言（最右边假定是 0），用两条规则可以造出 G_{n-1}；Gray 码从 000…0 到 100…0，都只有 n-1 位，于是在最左边加上一个 0，这是 G_n 的前半部分。注意，前半部分有 2^{n-1} 个元素，与 G_{n-1} 元素个数一样多。现在考虑这两半的最后一个元素（n 个位）0100…0，它应该是第 2^n-1 个（注意从 0 起算的事实），所以用规则二，把最右边的那一个 1 的左边邻居的值改变，得到 1100…0，于是往后的规则正好就产生 G_n 的后一（也就是最左边一位是 1）半了。所以，两条规则可以完全产生 Gray 码。现在，可以放心使用这里的程序了。

【程序】

程序 12-3（GRAYCODE.C）

```
/* ------------------------------------------------------------------ */
/* PROGRAM Subset Generation by Gray Code :                           */
/*   This program generates all possible subsets of an                */
/* elements set with Gray Code.  Simply speaking, the Gray            */
/* code of n bits is a sequence of n-bit strings such that            */
/* any two consecutive two bit strings have exactly one               */
/* position in difference.  For example, for n = 3, the               */
/* gray code generated is the sequence 000,001,010,011,               */
/* 010,110,111,101 and 100.  Therefore each item of this              */
/* sequence corresponds one subset as in the direct method            */
/*                                                                    */
/* Copyright Ching-Kuang Shene      July/05/1989                      */
/* ------------------------------------------------------------------ */

#include <stdio.h>
#include <stdlib.h>
```

Download at http://www.pin5i.com/


```
#define   MAXSIZE        20
#define   YES             1
#define   LOOP            1

#define   FLIP_DIGIT(x)   x = ((x) == '0' ? '1' : '0')
#define   FLIP(x)         x = (1 - (x))

void main(void)
{
     char  digit[MAXSIZE];
     int   position[MAXSIZE];
     int   even;
     int   n;
     int   i, count = 0;
     char  line[100];

     printf("\nAll Subset Listing by Gray Code");
     printf("\n================================");
     printf("\n\nNumber of Elements in Set Please --> ");
     gets(line);
     n = atoi(line);

     printf("\n");        /* initialization                          */
     for (i = 0; i < n; i++) {
        digit[i] = '0';
        printf("0");
     }
     printf(" : {}\n");  /* the empty set                            */

     even = YES;
     while (LOOP)  {
        if (even)                      /* for even positions:0,2,..*/
           FLIP_DIGIT(digit[0]);       /* flip the 1st bit         */
        else {                         /* for odd positions...     */
           for (i = 0; i < n && digit[i] == '0'; i++)
               ;                       /* find the first 1 bit     */
           if (i == n-1) break;        /* if it is the last..      */
           FLIP_DIGIT(digit[i+1]);     /* NO, flip its nbr         */
        }
        for (count = 0, i = n - 1; i >= 0; i--) {
           printf("%c", digit[i]);     /* print the bits           */
           if (digit[i] == '1')        /* and collect pos          */
```

Download at http://www.pin5i.com/


```
        position[count++] = i + 1;
    }
    printf(" : {%d", position[count-1]};
    for (i = count - 2; i >= 0; i--)   /* print pos        */
        printf(",%d", position[i]);
    printf(")\n");
    FLIP(even);              /* next will be odd(or even)   */
        }
    }
```

问题 3.4 产生所有排列——旋转法（PERMUT_R.C）

请写一个程序，用旋转顺序（Rotation Order）来列出 n 个元素的所有排列。

【解答】

回想一下在问题说明中提过的 n 个元素的排列，如果从 1,2,3,…,n 起旋转，n 次之后就会还原；换句话说，可以测试第 n 位，看它是否是 n 就知道还原了没有。如果还原了，旋转一次前 n-1 位，如果又还原了，就旋转前 n-2 位等；用 p 表示目前要旋转的是前 p 位，如程序 12-4 所示。

【程序】

程序 12-4

```
while（还有位数可以旋转）{
        p=第 n 位;
        旋转 1 到第 p 位;
    while（第 p 位等于 p 而且 p 至少是第二位）{
        p--;
        旋转 1 到第 p 位;
    }
}
```

大循环是用来掌握整个 n 位的旋转的，所以在大循环里 p 就是第 n 位的位置；经过旋转之后到达内循环，在内循环中先看看第 p 位是不是 p，如果是，就表示还原了，因为把 p 减去 1，接着旋转时就是前 p-1 位了，再次旋转之后很可能又还原，因此 p 又向左移了一位，一直到 p 到达第一位为止，这时就无法左移，程序自然就结束了。不过，也有可能到某个 p 经过旋转之后并没有还原，于是又再度回到大循环，开始大旋转。

例如，如果在大循环中经过旋转后得到 43125，因为第 5 位是 5，所以大循环还原；进入内循环之后，就进行前 4 位的旋转，得到 31245，不过第 4 位等于 4，因此还不能离开内循环；下一次是前 3 个元素的旋转，得到 12345；这一次第 3 位是 3，仍然不能离开，因此

Download at http://www.pin5i.com/

开始前两位的旋转，得到 21345。终于第 2 位不是 2，于是离开内循环而进入大循环，因此大循环就会产生下一个排列，也就是 52134,45213,34521,13452,21345；现在第 5 位还原，……。

程序 PERMUT_R.C 就是用这个方法写成的；注意，C 语言的数组是从 0 算起的，因此第 i 位，脚码就是 i-1，而不是 i。程序中的变量 position，就是上面的 p，于是 position 值是 0～n-1 之间。另外，为了方便，ROTATE 是定成一个集体指令而非函数，当然写成函数亦无不可，但是在执行时间上却不一定快速。

【习题】

（1）有没有办法把这个旋转法写成递归程序？试试看。

（2）程序中，小旋转都是以左边第一位为基准的，如果要求把所有的旋转都放到右边去，程序要做什么样的改变？请把程序写出来。

（3）下面的方法是由 G.Langdon 在 1967 年提出的，如程序 12-5 所示，请了解其操作方式，并且写成程序。

【程序】

程序 12-5

```
/* 脚码是 1 到 n                                    */
/* p[]中是排列                                      */
/* q[]工作用                                        */

把 p 转到 q 中;
输出结果;
i = n;
do {
     旋转 1 到 i;
     if (p[i] == q[i])
          i = n;
     else
          i--;
     输出结果;
} while (i >= 1);
```

【程序】

程序 12-6（PERMUT_R.C）

```
/* ------------------------------------------------------------ */
/* PROGRAM permutation by rotation :                            */
/*   Give an integer n, this program generates all              */
/* possible permutations by using rotation operations.          */
/*                                                              */
/* Copyright Ching-Kuang Shene      July/08/1989                */
/* ------------------------------------------------------------ */
```

Download at http://www.pin5i.com/


```
#include <stdio.h>
#include <stdlib.h>

#define  MAXSIZE  20
#define  ROTATE(p)  {  int  i, temp;                          \
                       temp = perm[p];                        \
                       for (i = p-1; i >= 0; i--)             \
                            perm[i+1] = perm[i];              \
                       perm[0] = temp;                        \
                    }
void main(void)
{
     int  perm[MAXSIZE];
     int  position;
     int  n;
     int  i;
     char line[100];

     printf("\nPermutation by Rotation Method");
     printf("\n==============================");
     printf("\n\nNumber of Elements --> ");
     gets(line);
     n = atoi(line);

     for (i = 0; i < n; i++)     /* initialize to 1,2,...,n     */
         perm[i] = i + 1;

     position = n - 1;
     while (position != 0) {     /* if still have positions..   */
         printf("\n");           /* display result              */
         for (i = 0; i < n; i++)
             printf("%d ", perm[i]);

         position = n - 1;       /* starts from the last pos    */
         ROTATE(position);       /* rotate them.                */
         while (perm[position]==position+1 && position!=0) {
             position--;         /* if last pos are equal and   */
             ROTATE(position);   /* not zero, rotate again      */
         }
     }
}
```

Download at http://www.pin5i.com/

问题 3.5 产生所有排列——字典顺序（PERMU_LR.C）

请写一个程序，用字典顺序列出 n 个元素的所有排列（Permutation）。

【解答】

先讲递归的做法，n 个元素一共有 n!个排列，而 1～n 都有可能出现在第一个位置，因为要用字符串顺序来排，所以这 n!个排列是以 1 开头的(n-1)!个在前，接着是以 2 开始的(n-1)!个，…，最后是以 n 开头的(n-1)!个。就用以 i 开头的为例，i 已经在第 1 位，自第 2 位起到第 n 位止中不会有 i，因此以 i 开始的字典顺序以 i 1 2 3 … (i-1) (i+1) … (n-1) n 起，而以 i n (n-1) … (i+1) (i-1) … 3 2 1 结束。如果用递归方式，以字典顺序来排 n-1 个元素的程序，不过是把该(n-1)!排列（其中不含 i）排出来，接在 i 的后面而已。

当以 i 起头的(n-1)!个排完后，如何产生以 i+1 起头，字典顺序的下一个呢？再套用上面的说法，这一组(n-1)!个排列以(i+1) 1 2 3 … (i-1) i (i+2) … n 起，到(i+1) n (n-1) … (i+2) i (i-1) … 3 2 1 止。可以把 i 起头的最后一个中的 i+1（它在倒数第 i 位）与 i 互换，因此得到：

(i+1k) n (n-1) … (i+2) i (i-1) … 3 2 1

再把后面的 n-1 个元素反转：

(i+1) 1 2 3 … (i-1) i (i+2) … (n-1) n

这不就是(i+1)起头的第一个元素吗？

在写程序时要注意：因为要处理在后面的 n-1 个元素，所以用两个变量 L 与 R 来指出要处理的部分；如果用 permu[ ]数组来存放一个排列的话，那么程序的架构就如程序 12-7 所示。

【程序】

程序 12-7

```
i = R;
while (1) {
      排列 perm[L+1]到 permu[R]的部分;
      把 perm[L]与 perm[i]互换;
      把 perm[L+1]到 perm[R]反过来排;
      i--;
}
```

当然，这还缺少递归的终止条件（以上面来看是永远不会停止的）；不过这很简单，当递归调用时，若 L 到 R 之间至少有 3 个元素，就做下去，而当只有两个元素时就不必递归调用了，此时接下来的两步正好就是两个元素的排列（两个元素的排列只有两种，一种是输入，另一种是两个元素互换——反过来排）。由此，递归程序就可以写出来，这就是程序 PERMU_LR.C。

Download at http://www.pin5i.com/

【习题】

（1）在递归程序中 R 的值永远不变，因为它可以定成外部变量，因此 wells()函数的参数就变成只有一个 L。请问 R 这个值是多少？接着请把程序的递归去掉，转换成非递归的程序。

（2）请证明 Fischer-Kruse 的方法是正确的；也就是说，如果已知一个排列，然后从右向左找出一个 a_i 满足 $a_i<a_{i+1}$；再从右向左找出一个 j，使得 $a_i<a_j$（如果找不出来又如何？），接着把 a_i～a_j 的元素反过来排，这就是给定排列在字典顺序中的下一个。

（3）本题的顺序是以字典顺序从小到大排的，请修改 Fischer-Kruse 以及递归程序，使它可以用反字典顺序排列。

（4）请把 Fischer-Kruse 方法写成非递归的形式。

【参考文献】

列出所有排列的方法很多，大多数重要的作品可以在下面这篇重要文献中找出来：

R.Sedgewich. Permutation Generation Methods. Computing Surveys, 1977(9), pp.137～164

对于初学者而言，这是一篇不可不读的好文章。

【程序】

程序 12-8（PERMUT_R.C）

```
/* ------------------------------------------------------------ */
/* PROGRAM lexical order permutations:                          */
/*   Give an integer n, this program generates all              */
/* permutations of n elements in lexical order. In this         */
/* program, a variable called last in used to record            */
/* a divided  boundary such that to its right some pos are      */
/* swaped.  See comments in the proge am please.                */
/*                                                              */
/* Copyright Ching-Kuang Shene      July/08/1989                */
/* ------------------------------------------------------------ */

#include  <stdio.h>
#include  <stdlib.h>

#define   MAXSIZE   20
#define   SWAP(x,y) {int  t; t = x; x = y; y = t; }

void main(void)
{
     int  perm[MAXSIZE];
     int  last;
     int  n;
```

Download at http://www.pin5i.com/


```
        int  left, right i, j;
        char line[100];

        printf("\nPermutation in Lexical Order");
        printf("\n==============================");
        printf("\n\nNumber of Elements --> ");
        gets(line);
        n = atoi(line);

        for (i = 0; i < n; i++) /* initialize to 1,2,...,n          */
           perm[i] = i + 1;

        for (last = 0; last >=0; ){
           printf("\n");    /* display result                       */
           for (i = 0; i < n; i++=
               printf("%d ", perm[i]);

           /* find the right most one with  p[i])< p[i+1]=          */

           for (last=n-2; last>=0 && perm[last]>perm[last+1];last--)

           /* find the smallest j, s, t,  p[last])<= p[j]=          */

              for (j=n-1; perm[last]>perm[j];j--)
              SWAP(perm[last], perm[j]);  /* reverse last part       */
              for (left=last+i, right=n-1; left<right; left++, right--)
                  SWAP(perm[left], right[j]);
        }
    }
```

问题 3.6　所有 K 个元素的子集（KSUBSET.C）

请写一个程序，把一个 n 个元素的集合{1,2,3…,n}中所有 k 个元素的子集用字典顺序列出来。

【解答】

先列出{1,2,3,4,5}中 3 个元素的所有子集：

{1,2,3}
{1,2,4}
{1,2,5}

Download at http://www.pin5i.com/

{1,3,4}
{1,3,5}
{1,4,5}
{2,3,4}
{2,3,5}
{2,4,5}
{3,4,5}

在开始时是{1,2,3}，接着会发现最右边的数逐渐增加，到 n(=5)为止，然后它的左边位数进 1，右边的位数则从下一个值开始；这个动作就像是一个里程表，各个位数不断增加，所不同的是在例子中，从左向右的位数永远保持从小到大的位数。

正因为如此，只需要掌握在什么时候哪一位该进位就行了。能够产生出来的集合中最“大”的一个{3,4,5}，最后一位是最大元素，左边一位是 5-1=4，再左一位是 4-1(=5-2)；依次类推，如果原来的集合有 n 个元素，要取出的是 k 个元素的集合，那么最“大”的集合就是：

$$\{n-k+1,\cdots,n-2,n-1,n\}$$

因此最后一位的值只能变到 n，之后就退回一位，但右边第二位只能变到 n-1，…。

如果仔细看看上面的例子，不难发现如下事实：

（1）如果最后一位的值小于 n，那么再加 1 的位置永远是最后一位。

（2）如果从右边算来，每一位都等于它的最大值（最右边是 n，右边第二位是 n-1,…），那么再加 1 的位置是从右边往左边算，第一个小于极大值的位置。

以{2,3,4}为例，最右边一位(4)小于 n=5，所以加 1 的位置是最后一位，因而得到{2,3,5}；再看{2,3,5}，从右往左看，最后一位已经到达最大的值，但倒数第二位却不是，所以倒数第二位加 1，得到{2,4,5}；到了{2,4,5}之后，从右往左算，第一位不到它的极大值，所以第一位加 1，得到{3,4,5}。最后，每一位都到了极大值，怎么办？事情做完了归纳起来，从{1,2,3,…,k}这个子集起，每次从右往左查，查出第一个值不到该位极大值的位置，把它加 1，不但如此，它的右边各位也得依次增加。例如，如果有{3,4,7,8}，若 4 那一位应该加 1，而得到{3,5,x,x}，再把右边各位从 5 起，每一位以 1 的值递增，最后得到{3,5,6,7}。

不过每次都要从右往左查是一件很麻烦的事，有没有别的办法直接定出要加 1 的是哪一位呢？很简单，不过又要作进一步的思考而已，主要关键是第二点。做法是，使用一个变量（position），它一开始时是 k-1（注意，C 语言中的数组从零算起，而要求出 k 个元素的所有子集），表示要加 1 的是第 k 位。在往后的工作中，如果第 k 位的值小于 n，position 的值就维持在 k-1，在这种情况下，被加 1 的永远是第 k 位。但是，若第 k 位的值已经到达了 n，就不能再加 1 了，就把 position 减 1，表示 position 左边那一位要加 1。看起来有点奇怪，但是举个例子就不难明白。例如，n=5，k=3，现在有：

$$\{1,3,\underline{4}\}$$

加了底线的地方就是 position 的位置，因为第 k 位小于 n，自然 position 就在那，加了 1 之后变成：

$$\{1,\underline{3},5\}$$

Download at http://www.pin5i.com/

但第 k 位变成 5 了，造成要加 1 的是旁边的那一位，只要把 position 减 1 即可，因此把那一位右边的依次以 1 递增，所以{1,3,5}的下一个集合是：

{1,4,5}

接着，再看第 k 位，还是等于 n，所以 position 再减 1，有：

{1,4,5}

加 position 指出的那一位加上 1，并且调整在它右边的各位，就得到：

{2,3,4}

但是这样一来，k 位又小于 n 了，所以 position 又转回第 k 位，下面就是从头开始算一次的 position 的位置：

1 2 3
1 2 4
1 2 5
1 3 4
1 3 5
1 4 5
2 3 4
2 3 5
2 4 5
3 4 5

程序就是用以上的观点写成的，用 set[ ]来储存 k 个元素的集合；再次提醒，C 语言中的数组都是自 0 算起的。

【习题】

（1）请用每一次都从左向右地找出 position 的值的方式（也就是解答中的两个事实）来改写程序。

（2）请写一个程序，把一个有 n 个元素的集合的所有子集依元素个数从小到大，而相同元素的子集用字典顺序排出来；以 3 个元素的集合来看，就是{},{1},{2},{3},{1,2},{1,3},{2,3},{1,2,3}。

（3）如果 a 是一个有 n 个元素的集合的元素，请写一个程序，列出所有包含 a，以及所有不包含 a，并且元素个数都有 k 个的子集；请问，这各有多少个？这个数与 n、k 之间的关系如何？

【程序】

程序 12-9（KSUBSET.C）

```
/* ------------------------------------------------------------------ */
/* PROGRAM Lexical Order of k-Subsets :                               */
/*   Given a n-element set {1,2,3,...,n} and a k <= n,                */
/* this program lists all k-element subsets of {1,2,..,n}             */
/* in lexical order.                                                  */
```

Download at http://www.pin5i.com/


```
/*                                                                    */
/* Copyright Ching-Kuang Shene      July/08/1989                     */
/* ------------------------------------------------------------------ */

#include  <stdio.h>
#include  <stdlib.h>

#define   MAXSIZE   20
#define   LOOP       1

void main(void)
{
     int  set[MAXSIZE];
     int  n;
     int  k;
     int  position;
     int  i, j;
     char line[100];

     printf("\nAll k Elements Subsets from an Elements Set");
     printf("\n==========================================");
     printf("\n\nNumber of Elements in the Set --> ");
     gets(line);
     n = atoi(line);
     printf("\n\nNumber of Elements in Subset ---> ");
     gets(line);
     k = atoi(line);

     for (i = 0; i < k; i++)     /* initialize the subset to     */
         set[i] = i + 1;         /* {1,2,3,...,k}                */
     printf("\n{%d", set[0]};    /* display it                   */
     for (j = 1; j < k; j++)
         printf(",%d", set[j]);
     printf(")");

     position = k - 1;           /* 'position' indicates the     */
     while (LOOP) {          /* pos to be increased              */
         if (set[k-1] == n)      /* if last pos is full, then    */
             position--;         /* move position left           */
         else                    /* or if not full, then the     */
             position = k - 1;   /* pos is always the last       */
         set[position]++;        /* so add one to the pos.       */
         for (i = position+1; i < k; i++)   /* add inc. all      */
             set[i] = set[i-1] + 1;         /* pos to its right */

         printf("\n{%d", set[0]};           /* displayit.       */
```

Download at http://www.pin5i.com/


```
            for (j = 1; j < k; j++)
                printf(",%d", set[j]);
            printf(")");

            if (set[0] >= n-k+1) break;          /* if 1st pos full */
        }
    }
```

问题 3.7 集合的所有分割方式（SETPART.C）

一个集合 S 的分割（Partition）就是把 S 分解成若干个子集 $S_1, S_2, \cdots, S_p$，使得 $S_1 \cup S_2 \cup \cdots \cup S_p = S$，但对任意两个不同的子集 S_i 与 S_j 而言，$S_i \cap S_j = \phi$。换句话说，集合的分割，就是把该集合分解成若干相互分离的子集。举例来说，{1,2,3}的分割就有{1},{2},{3}；{1,2},{3}；{1,3},{2}；{2,3},{1}；{1,2,3}这 5 种。请写一个程序，读入 n，列出{1,2,3,…,n}的所有分割方式。

【解答】

回想一下在问题说明中的一个条件。如果要找出{1,2,3,…,n}的所有分割方式，再用 c_i 表示 i 的编码值，即 i 在哪一个分割中，那么 c_i 就小于或等于 1,2,…,i-1 的编码极大值再加上 1，用式子来看就是：

$$c_i \geqslant \max(c_1, c_2, \cdots, c_{i-1}) + 1$$

所以，当处理到 i 这个元素时，只要把 $c_1, c_2, \cdots, c_{i-1}$ 的极大值找出来，再加上 1；然后令 c_i 的值从 1 变到这个值，就可得到 c_i 可能所在的分割了。不过不能这么冒失，因为当 c_i 得到了一个合适的值时，要用这个值，以及 $c_1, c_2, \cdots, c_{i-1}$ 来算出 c_{i+1} 可以在什么地方；这个程序就一直反复，直到 c_n 处理完为止；接着回溯到 c_{n-1}，让它有下一个值，再处理 c_n。当 c_{n-1} 已经不能再增加时，回溯到 c_{n-2}，给它下一个值，这是一个回溯技巧的应用，当然也可以用递归。

c_1 一定永远在第一个分割，因此 c_1 永远是 1；c_2 最多可以是 $\max(c_1)+1=2$，因此 c_2 的值可以是 1 或 2。不过总是在求极大值是很麻烦的，不如先求出来，然后在需要时，再更正它，这正是程序 SETPACT.C 的写法。

程序中有两个数组，它是用 malloc()函数取得内存位置的；code[ ]就是 c_i，而 maxi[ ]则是极大值；换言之，maxi[i]就是 code[0],code[1],…,code[i-1]的极大值再加 1。在一开始时，所有元素在同一个分割中，于是每一个 code[i]都是 1，而在这种情况下，所有 maxi[i]就都是 2，因为 code[0]～code[1]的值都是 1，极大值自然是 1，再加上 1 就得到 2。

接下来的 while 循环就一直循环，直到回溯的动作回到开头，也就是处理 i=0 的情况为止，因为它一定在第一个分割，于是就可以停止操作了。每一次，都从右往左查，找出第一个 code[i]与 maxi[i]不相等的 i；因为若相等，就表示 i 的编码已经到达了最大值，不能增

Download at http://www.pin5i.com/

加了，只有在不相等时才能增加，当 for 循环停止时的那一个 i 就有可能是可以再度递增的。然而，还要多做一步，因为 i 可能是 0，也就是说回溯（也就是 for 循环的动作）到第一个元素了，那么工作就结束了，第一个元素只能在第一个分割。但若 i 不是 0，就可以把 i 的编码值加上 1，并且由此而更正 i+1,i+2,…, n-1 的极大值；更正的程序不难，在 i 与 i 以前的 maxi[ ]与 code[ ]都不曾改变，因此从 i+1 以后的 code[ ]值就都变成从 1 起，而 maxi[ ]值则是与 maxi[i]相同或加 1，这是由 code[i]目前（加上 1 之后）的值是否与 maxi[i]相同而定。如果相同，那么 code[0]～code[i]的极大值就是 code[i]，自然地在它后面的 i+1 就要把 code[i]的极大值加上 1 了；但若不同，它们的极大值就与 maxi[i]相同。

这样，就讲完了一个不太简单的题目，可以通过计算 n=4,5 这些比较小的值来了解其中的奥妙。

【习题】

（1）程序的输出只有各个元素的编码值，请多加一个函数，把编码值与实际的分割（也就是子集）一起显示出来。

（2）在问题证明中讲过了各个分割中的元素排法，如果 i 的编码是 c_i，请证明 $c_i \geqslant \max(c_1, c_2, \cdots, c_{i-1}) + 1$。虽然也提到了如何做这件事，但却相当不严谨，应该尝试进行一个严格的论证。

（3）为什么在程序中当 code[i]=maxi[i]时，i 就是下一个可以增加值的元素，请严格地证明这一点。

（4）为什么当 code[i]增加成下一个值之后，在 i 以后的 maxi[ ]值要用程序中的方式改变？请尝试作出一个严格的论证。

【参考文献】

集合的分割与 Gray 码也有相当的关系，可参看以下书籍。

1 R.Kaye. A Gray Code for Set partitions. *Information Processing Letters*, 1976(5), pp.171～173

比较近期的作品则有：

2 C.Er. A Fast Algorithm for Generating Set Partitions. *The Computer Journal*, 1988(31), pp.283～284

3 B.Djokic, M.Miyakawa, S.Sekiguchi, I.Semba and I.Stojmenovic. A Fast Iterative Algorithm for Generating Set Partitions. *The Computer Journal*, 1989(32), pp.281～282

M.C.Er 的方法是递归的，并且号称比当时的所有方法都要快。第三篇则是更新的方法，不过与此处的解答相差不大，但它们有一个改良的技巧。与他们所作的比较，方法比 M.C.Er 的方法快 20%左右。有兴趣的朋友不妨读一读上列的文章，不过这些程序有点校对上的问题，应该试试把它更正过来。

【程序】

程序 12-10（SETPART.C）

```
/* ------------------------------------------------------------------ */
/* FUNCTION  set_partition :                                          */
```

Download at http://www.pin5i.com/


```
/*   Given an integer n, this function will lists all          */
/* possible partitions of {1,2,3,...,n}.                        */
/*                                                              */
/* Copyright Ching-Kuang Shene      July/24/1989                */
/* ---------------------------------------------------------    */

#include <stdio.h>
#include <stdlib.h>                /* for malloc() */

#define ALWAYS  1

void display(int *, int);

void set_partition(int n)
{
     int  *code, *maxi;            /* arrays for code[], maxi[]   */
     int  i, j;

     code = (int *) malloc(sizeof(int)*n);  /* get memory        */
     maxi = (int *) malloc(sizeof(int)*n);

     for (i = 0; i < n; i++=    /* initialization                 */
        code[i] = 1, maxi[i] = 2;

     while (ALWAYS) {           /* loop until done.               */
        display(code, n);       /* display one partition          */
        for (i = n-1; code[i] == maxi[i]; i--)
           ;                    /* find next 'increasible'        */
        if (i > 0) {            /* found ?                        */
           code[i]++;           /* YES, update                    */
           for (j = i + 1; j < n; j++= {
        code[j] = 1;
        maxi[j] = maxi[i]+((code[i]==maxi[i]) ? 1 : 0);
           }
           }
           else                 /* NOT FOUND, done.               */
        break;
        }
        free(code);
        free(maxi);
     }
```

Download at http://www.pin5i.com/


```
/* ----------------------------------------------------------------- */
/* FUNCTION  display :                                               */
/*   This function displays the code of the partition.               */
/* ----------------------------------------------------------------- */

void  display(int *code, int n)
{
      int  i;

      printf("\n");
      for (i = 0; i < n; i++=
         printf("%3d", *(code+i));
}
```

问题 3.8 整数的所有不同分割数目（INTPART#.C）

所谓的整数的分割（Partition of an Integer），指的就是把一个正整数写成若干个正整数的和，不过只计较有多少种分解方式，而不计较它的内容。比如说，4=3+1,2+2,2+1+1,1+1+1+1，再加上自己，就一共有 5 种分割方式。请写一个程序，输入一个正整数，输出它不同分割方式的数目。

【解答】

用 p(n,m)表示在 n 的分割中极大值最多为 m 的分割数目，亦即分割具有 n=m+…,n=m+m+…,n=m+m+m+…这一类的形式。于是 n 与 m 的关系不外乎以下几种情况。

（1）m 为 1：因为 n 的分割中的极大值为 1，而且分割中又完全是正整数，所以整个分割就都是 1。以 4 为例，就是 4=1+1+1+1。于是有 p(n,1)=1。

（2）n 为 1：因为要分割的数本身就是 1，这已经是正整数中最小的值了，所以不论如何分割，也不论 m 为何值，1 一定小于 m，p(1,m)=1。

这两点就构成了递归调用的终止条件。在递归程序中，永远记住要找出能够令递归终止而返回结果或控制权的条件。

（1）n=m：也就是 n 的分割中，极大值不超过 n 的个数。当然，极大值为 n 的分割只有一个，也就是它自己。换言之，在 4 的分割中，4=4 就是极大值正好是 n 的情况．但除了这一个之后，极大值就不可能为 n 了，最多是 n-1 而已。因此求出 n 的分割中极大值是 n-1 的分割个数，再加上 n=n 这一个分割，就是 p(n,n)。换句话说，$p(n,n)=1+p(n,n-1)$。

（2）n<m：要把 n 分割，但极大值最大可以为 m，而且 m>n，这是不可能的事，因为把 n 分割时，在 n=n 的情况下就已经是最大可能了。所以说，在 n 的分割中，如果要求极大值最多可以到 m(>n)，这是不切实际的，因为分割中不可能出现 n+1,n+2,…m 这些值。

Download at http://www.pin5i.com/

换言之，$p(n,m)=p(m,n)$。

（3）n>m：这是最常见的可能性了。在 n 的分割中，如果极大值最大可以为 m，那么这不外乎分割中有一个或若干个 m，或者是根本就没有 m。比如，在 4 的分割中，说极大值最多可以到 2，那么因为 4=2+2=2+1+1=1+1+1+1，于是有 m=2 的就是 2+2 与 2+1+1，完全没有 m=2 的则是 1+1+1+1。如果 n 的分割中根本没有 m，那么它的个数不就与 n 的分割中最大可以到 m-1 的个数（亦即 p(n,m-1)）相同吗？如果 n 的分割中至少有一个 m，亦即 n = m + …，（注意，…部分可能有 m），于是 n-m=…，所以…的部分正好是 n-m 的一个分割，它的极大值最多可以到 m；因为 n-m 的分割中极大值最多可以到 m 的有 p(n-m,m)个，再加上 p(n,m-1)，这就是所有的情况，亦即 p(n,m)=p(n,m-1)+p(n-m,m)。

这样就把所有的可能性都考虑到了。在程序中，compute()函数对应着 p(n,m)，变量 number 是 n，maximum 则是 m。它就依上述的 5 个情况分析，并且递归调用自己来求值。

那么 n 的分割数目是多少呢？是 p(n,n)，这是说 n 的分割中极大值最多是 n 个数，自然地 n 的分割中不可能有大于 n 的情况。由上述的第（3）点，用 compute(n,n)就可以得到结果了。但是，求 n 的分割只用到 n 一个参数而已，为什么要那么麻烦地写成 compute(n,n)的两个呢？所以写一个额外的函数 int_part_no()来做这件工作，把这件琐事处理掉（想想看，万一有一天在使用 compute()时只给一个参数会有什么后果？）。

【习题】

（1）请把这个程序改写成非递归的形式。

提示：可以用一个表格来做这件事，如 table[n][m]，就存放了 p(n, m) 的值，这个表格的计算次序如何？

（2）一个集合的分割，就是把该集合分解成若干个两两分离的非空集合，并且这些子集的合集等于已知的集合。例如，{3,2,1}的分割有{1},{2},{3}；{1},{2,3}；{1,2},{3}；{1,3},{2}；{1,2,3}这 5 种．请写一个程序，读入集合的元素个数，输出这个集合会有多少种不同的分割方式，这个数叫做 Bell 数。

提示：与例题一样，此处的集合分割中，分解成几个部分的数目是非常重要的，因此分解成 1 个，2 个，…，到 n 个（假设集合有 n 个元素）部分的不同做法都知道了，加起来就是所有的分割方式，想法与此处的程序大同小异。

（3）请问在程序中一共用了多少个加法，递归调用了多少次？为了要算 p(n,m)，有必要求出所有 p(i,j)的值吗？此地 1≤i≤n，1≤j≤m。

（4）用 G(n)表示在有 n 位的二进制数中没有相邻的两个 1 的二进制数个数。比如，当 n=3 时，000,001,010,011,100,101,110,111 这 8 个数中只有 000,001,010,100,101 这 5 个是没有相邻为 1 的，故 G(3)=5。请写一个程序，输入 n，输出 G(n)的值。

提示：不应该 000…000～111…111 一个个地测试，请试一试“分而治之”的策略，把 n 位二进制数分成两部分，分别处理这两部分，并且小心地合并．

Download at http://www.pin5i.com/

【程序】

程序 12-11（INTPART#.C）

```
/* ----------------------------------------------------------- */
/* FUNCTION  int_part_no :                                     */
/*   Given an integer n, this function computes the            */
/* number of partitions of the input integer.  NOTE that       */
/* this function computes the number only and does not         */
/* generate the corresponding partition.  For the second       */
/* purpose, see program INTPART.C please.                      */
/*                                                             */
/* Copyright Ching-Kuang Shene      July/23/1989               */
/* ----------------------------------------------------------- */

unsigned long  compute(int, int);
unsigned long  int_part_no(int);

unsigned long  int_part_no(int n)
{
     return compute(n, n);    /* convert to another form      */
}

/* ----------------------------------------------------------- */
/* FUNCTION  compute :                                         */
/*   The computation routine.  It accepts number-the           */
/* number to be partitioned, and maximum-the maximum           */
/* value in any partition of number, then returns the          */
/* number of partitions subject to number and maximum.         */
/* ----------------------------------------------------------- */

unsigned long  compute(int number, int maximum)
{
     if (number == 1 || maximum == 1)
        return 1;
     else if (number < maximum)
        return compute(number, number);
     else if (number == maximum)
        return 1 + compute(number, maximum-1);
     else
        return compute(number,maximum-1) +
           compute(number-maximum,maximum);
}
```

Download at http://www.pin5i.com/

问题 3.9 整数的分割方式（INTPART.C）

对于一个正整数 n 而言，它的一个分割（Partition），就是把 n 写成若干个正整数的和，但不计较书写的顺序。请写一个程序，输入 n，把 n 的所有分割显示出来。

【解答】

把正整数分割成若干正整数的和并不是一个十分困难的问题。如果回头看问题说明中把 7 分解的例子，就不难发现从上一个分割方式到下一个分割式中，永远是最右边或是旁边的一连串 1 的位数有所改变。其实在问题说明中的排列方式，正是所谓的反字符串顺序，就以 4、3 为例，它的下一个是多少？因为最右边位数的 3 还有可以分解的余地，但比 3 小的下一个分解是 2，3 就分解成 2+1，故 7=4+3=4+2+1；到了 4+2+1 之后，1 自然不能分解了，但 1 左边的 2 还可以，所以把 2 分解成 1+1，故得到 7=4+1+1+1。接着，下一个可以再度分解的是 4，4 可以分解成 3，因此 7=3+x，后面的 x 是多少呢？自然它们的和是 7-3=4，但此地不能用 4，因为在此之前已经用过了，而用比 3 小的才行，要比 3 小，那只有 2，7=3+2+2，于是接下来的是 7=3+2+1+1,…。这就是解题的基本想法。

不过为了要把题目解得漂亮，要有进一步的、好的数据结构才行。为什么呢？因为在分解成正整数的和时，可能会有很多重复的部分，特别是要分解的数很大时就更是如此。10=4+3+3=3+2+2+2+1=…就是一个例子，因此用两个数组来做这件事。partition[ ]中储存了分解出来的、不同的正整数，而 mult[ ]则用来记录对应的整数的重复次数；以 10=3+2+2+1+1+1 为例，partition[ ]中依次为 3、2、1，而 mult[ ]中则是 1、2、3，因为 3 有 1 次，2 有两次，而 1 有 3 次，为了说明方便起见，不用 partition[0]与 mult[0]。

在开始时，如果要分解的数是 n，那么 partition[1]就是 n，而 mult[1]为 1，表示只出现一次，接下来就要把上面所讲的观点转化成程序了。用 part_no 记录下 n 在目前一共分解成多少个不同的整数，依照上面的说法，可以分成两个部分来说：

（1）最后一个值不是 1：就以 25=5+4+4+3+3+3+3 为例，partition[ ]依次为 5,4,3，而 mult[ ]则是 1,2,4；因为最后一个数(3)不是 1，所以可以把它分离出来，减去 1 就是下一个数（2=3-1）：

$$25=5+4+4+3+3+3+(2+\cdots)$$

但因为分离出来的是 3，它的下一个较小的值是 2，那么有多少个 2 呢？很简单，3/2=1 个 2，所以 partition[ ]变成 5,4,3,2，mult[ ]则是 1,2,3,1；但是 3 不能被 2 整除，还有剩余的数（也就是商数，它必定小于 2），所以就加在最后面，因此得到：

$$25=5+4+4+3+3+3+2+1$$

总而言之，如果 partition[part_no]，也就是最后一个数值不为 1，就可以把它取一个出来，叫做 sum，表示取出之后必须要加上 sum 的值才能得到和为 n。其次，下一个数要比 partition[part_no]小，亦即值为 partition[part_no]-1，把这个值存到 size 中；这代表在分割中的下一个值。但是要注意一点，当 mult[part_no]为 1 时，取出这个值之后原来的数就没有

Download at http://www.pin5i.com/

了（例如，在 16=5+4+4+3 中取了 3 之后，3 这个部分就不存在了），所以当 mult[part_no]不是 1 时，可以多一个数；但当 mult[part_np]为 1 时，则不能多一个数。

因为 size 代表下一个值，所以把它存到 part_no 所指出的下一个位置中，它能够重复多少次呢？前面说过，sum/size 次；但若 size 不能整除 sum，多出来的余数部分就是一个新的值。例如，对 25 而言，如果 partition[ ]中是 5,4,3，mult[ ]为 1,2,4；于是 sum 为 3，表示从最后的 4 个 3 中取了一个 3 出来，size 是 2，因为比 3 小的下一个值是 2。由于取了一个 3，故 mult[ ]的新值为 1,2,3，多出来的这个 sum 可以被 size 分成 3/2=1 组，所以 partition[ ]为 5,4,3,2，mult[ ]则是 1,2,3,1；不过正因为 sum 不能被 size 整除，还余 1，所以这个 1 就形成一个新的部分，把它加到分割中，有 partition[ ]为 5,4,3,2,1，mult[ ]为 1,2,3,1,1。但若 25=11+11+3，partition[ ]为 11,3，mult[ ]为 1；因此 sum=3，size 是 2；但因为 3 被取出之后，在原来分割中就不存在了，所以新产生的分割（sum/size）就像它原来的位置，即结果中 partition[ ]为 11,11,2,1，mult[ ]是 2,1,1。

（2）如果最后一个值是 1：这些 1 一定在最后面，也就是 partition [part_ no]，有多少个则是在 mult[part_no]中，因为 mult[part_no]指出有多少个 1，因此也正是由 1 所构成的部分的总和。因为 1 已经不能再分解了，所以在求下一个分解方式时，这些 1 都要去掉，而下一个值是从最后的一个不是 1 的值中取一个出来，再减去 1；于是 sum 就是 mult[part_no]+1，而 size 则是 partition[part_no-1]-1；为什么 sum 是 mult[part_no]+1，而不是 mult[part_no]呢？原因是，sum 是加上的值而凑成 n 用的，因为下一个可以用的值是 partition[part_no-1]-1，少了 1，所以就要在 sum 中多加一个 1 来弥补了。例如 14=4+3+3+1+1+1+1，partition[ ]是 4,3,1，mult[ ]为 1,2,4，自然 sum 为 4，但因为最后一个不是 1 而为 2，所以取出来又少了 1，因此在 sum 中加上 1 变成 1 作弥补：

$$
\begin{aligned}
14 &= 4+3+3+1+1+1+1 \\
&= 4+3+3+(4) \\
&= 4+3+(2+1)+(4) \\
&= 4+3+2+(5)
\end{aligned}
$$

于是 sum 是 5，size 为 2，其他工作就与上一点相同了，从 sum/size 求出 sum 可以用 size 分成多少组，加上从上一个值中取出的一组，就一共有 sum/size+1。于是新的 partition[ ]是 4,3,2,1，mult[ ]为 1,1,3,1。

程序 INTPART.C 是完全按照上面的观点写成的，只不过是上面求 sum 与 size 的工作合二为一而已，所以不应该会有任何困难。为了彻底地了解这个方法，建议找一个比较大的数（比如 8），从头开始用手算几个结果，就不难熟悉整个的处理过程。

【习题】

此处所列的是反字典顺序，能否修改这个程序用字典顺序列出结果呢？换言之，把 5 用 1+1+1+1+1,2+1+1+1,2+2+1,3+1+1,3+2,4+1 这 6 个顺序列出来。

【程序】

程序 12-12（INTPART.C）

```
/* ------------------------------------------------------------------ */
/* PROGRAM  integer partition :                                       */
```

Download at http://www.pin5i.com/


```
/*   Given a positive integer n, this program lists all     */
/* partition of n in anti-lexical order.                    */
/*                                                          */
/* Copyright Ching-Kuang Shene      July/21/1989            */
/* ---------------------------------------------------------- */

#include <stdio.h>
#include <stdlib.h>              /* for atoi()              */

#define  MAXSIZE  20

void display(int [], int [], int);

void main(void)
{
     int partition[MAXSIZE+1]; /* the actuall partition       */
     int mult[MAXSIZE+1];      /* multiplicity                */
     int part_no;              /* no. of parts                */
     int sum, size, remainder, count;
     int n;
     char line[100];

     printf("\nPartition of Integer");
     printf("\n====================");
     printf("\n\nInput a Positive Integer --> ");
     gets(line);
     n = atoi(line);

     partition[1] = n;          /* at the beginning, we have   */
     mult[1]      = part_no = count = 1; /* only one part.      */
     display(partition, mult, part_no);

     do {                              /* size one sum in 'sum'   */
        sum  = (partition[part_no]==1) ? mult[part_no--]+1 : 1;
        size = partition[part_no] - 1;  /* dec. size           */
        if (mult[part_no] != 1)      /* last part with mul=1    */
           mult[part_no++]--;        /* NO, cut this part       */
        partition[part_no] = size;  /* set new part=size        */
        mult[part_no]      = sum/size + 1;   /* fill other      */
        if ((remainder = sum % size) != 0) {
           partition[++part_no] = remainder;
           mult[part_no]
        }
```

Download at http://www.pin5i.com/


```
        count++;
        display(partition, mult, part_no);
    } while (mult[part_no] != n);
    printf("\n\nThere are %d partitions in anti-lexical order",
            count);
}


/* ----------------------------------------------------------- */
/* FUNCTION display :                                          */
/*   This routine displays the given partition.                */
/* ----------------------------------------------------------- */

void  display(int partition[], int mult[], int part_no)
{
     int  i, j;

     printf("\n");
     for (i = 1; i <= part_no; i++)      /* for each part       */
        for (j = 1; j <= mult[i]; j++)  /* and its mult.       */
           printf("%3d", partition[i]); /* show them           */
}
```

Download at http://www.pin5i.com/

第13章 查 找

问题 4.1 寻找脚码（ISEARCH.C）

已知一个整数数组 x[ ]，其中的元素彼此都不相同，而且也已经从小到大排列好。请用比较大小、相等的方式写一个程序，找出给定的数组中是否有一个元素满足 x[i]=i 的关系。举例而言，如果 x[ ]有 2,1,3,7,8，x[3]=3，因此 3 就是答案。

【解答】

如果要在一个已经排列好的数组中查找元素，那多半会用到二分查找法，不过这个题目有点不一样，因为“不知道”要找的数据的值，因此要有点变化才行。首先，要注意两种情况：

（1）如果 x[i]>i，那么对于比 i 大的任何一个脚码 k 而言，都会得到 x[k]>k，因此一旦发现了某个 i 满足 x[i]>i 时，满足 x[j]=j 的 j 就一定只会发生在 j<i 的所在。

（2）如果 x[i]<i，那么对于比 i 小的任何一个脚码 k 而言，都会得到 x[k]<k，于是一旦发现了某个 i 满足 x[i]<i 时，满足 x[j]=j 的 j 就一定只会发生在 j>1 的所在。

基于上述两种情况，程序就可以找出中间的元素，令它的脚码为 middle，比较 x[middle]与 middle 是否相等，相等的话就找到解答方法了。如果不相等，就看 x[middle]>middle 是否成立，如果成立，那么满足 x[i]=i 就一定在 middle 的左边；反之，则在右边。所以这是一个标准的二分查找法（Binary Search）的变形。

程序要作多少次比较呢？在 while 中每一个循环都比较两次，一次比相等，一次比大于。在最坏的情况下，while 一共会绕 $\log_2 n$ 圈，因为每一次要查找的部分都减半，所以总共会用 $2\log_2 n$ 次比较。这一项分析与二分查找法完全相同，肯定会比在问题说明中的方法快。

【习题】

（1）请证明上述的两种情况。

（2）这个程序是基于数组中元素完全相异的前提下做出来的，请问当有重复的元素，但仍然从小到大排好时，这个程序还能够正常运行吗？为什么？如果不能正常运行，请改写这个程序。

【程序】

程序 13-1（ISEARCH.C）

```
/* ------------------------------------------------------------------------ */
/* FUNCTION index_search :                                                  */
```

Download at http://www.pin5i.com/


```
/*   Given a sorted (increasing) array, this function          */
/* determines if there exists an index i such that x[i]=i.     */
/* It could be solved by a linear scan, but a modified         */
/* binary search would be much more efficient.                 */
/*                                                             */
/* Copyright Ching-Kuang Shene          July/01/1989           */
/* ----------------------------------------------------------- */

int  index_search(int x[], int n)
{
     int  first = 0;
     int  last  = n-1;
     int  middle, index;

     index = -1;
     while (first <= last) { /* a modified binary search    */
     middle = (first + last) / 2;
     if (x[middle] == middle) {
          index = middle;
          break;
     }
     else if (x[middle] > middle)
          last = middle - 1;
     else
          first = middle + 1;
     }
     return index;
}
```

问题 4.2　寻找固定的和（FIXSUM.C）

有两个数组 x[ ]与 y[ ]，各有 m 与 n 个元素，而且各个元素没有依顺序排列；d 是一个已知的值。请写一个程序，看看在 x[ ]与 y[ ]中有没有满足 x[i]+y[j]=d 的元素。例如，若 x[ ]为 3,7,2,4，y[ ]为 1,5,2,3，d 为 9；那么 x[1]+y[2]与 x[3]+y[1]都合乎条件，也即都是 9。

【解答】

解题的关键还是在二分查找法，但是因为两个数组都没有依顺序排列，要选其中之一排好才行。假设 x[ ]已经排好，排 x[ ]一般要用 $cm\log_2 m$ 次比较，在此 c 是一个与 m 无关的常数。排好之后，对于任何一个 y[j]而言，都可以在 x[ ]中查找有没有 d-y[j]这个数了，

Download at http://www.pin5i.com/

如果找到了某个 x[i]==d-y[j]，那么就有 x[i]+y[j]==d 的关系。因为 x[ ]有 m 个元素，对于每一个 y[j]而言，在 x[ ]中的二分查找法要用 $\log_2 m$ 次比较，但很可能所有 y[j]都比较后才能得到根本没有 x[i]+y[j]==d 成立的结论，这就一共比较了 $n\log_2 m$ 次。加上排序的比较次数，就一共比了 $(cm+n)\log_2 n$ 次。一般而言，当 m 与 n 比较大时，m・n 也会比 $(cm+n)\log_2 m$ 大，所以此程序的效率要比一般的写法高。

【习题】

（1）如果也将 y[ ]排序，比较的次数会不会降低？为什么？

（2）程序 fix_sum()只报告有没有 x[i]+y[j]==d 这样的条件成立而已，请修改程序，连同满足上述关系的 i 与 j 都显示出来。

（3）请修改程序，找出所有满足 x[i]+y[j]==d 的 i 与 j。请问在这一要求下，比较次数有没有增加？换言之，程序效率是否降低？

【程序】

这里用到了快速排列法 QSORT.C，不过并没有把它附在报表中，事实上程序的主要部分只有 fix_sum()程序。

程序 13-2（FIXSUM.C）

```
/* ------------------------------------------------------------ */
/* FUNCTION fix_sum :                                           */
/*   Given two int arrays x[] and y[], and an integer           */
/* given, this function checks to see if there is a pair        */
/* x[i] and y[j] such that x[i] + y[j] = given.                 */
/*   Solution is extremly simple.  Sort one of the given        */
/* array and do a sequence of binary searches.                  */
/*   NOTE that this function needs quicksort() routines.        */
/*                                                              */
/* Copyright Ching-Kuang Shene          July/02/1989            */
/* ------------------------------------------------------------ */

#define  YES   1
#define  NO    0

void  sort(int [], int);

int fix_sum(int x[], int y[], int m, int n, int given)
{
    int  first;
    int  last;
    int  middle;
    int  i;
```

Download at http://www.pin5i.com/


```
sort(x, m);                /* sort array x[]                */

for (i = 0; i < n; i++)  /* for each y[j], do a bin.     */
     for (first=0, last=m-1; first <= last; ) {
          middle = (first + last)/2;
          if (x[middle] + y[i] == given)
               return YES;
          else if (x[middle] + y[i] > given)
               last  = middle - 1;
          else
               first = middle + 1;
     }
     return NO;
}
```

问题 4.3　无限式查找（INF_SRCH.C）

已知一个数组，元素个数有多少并不很清楚，但是数组元素已经依顺序从小到大排好，而且在数组最后添加了足够多的∞记号；∞表示最大的值，比数组中每一个元素都大，而且个数足够多。试编写一个程序，在这个数组中找出某个给定的值。

【解答】

在使用二分查找的技巧时，要有左边的边界到右边的边界。当然，如果输入值是放在数组中，左边的边界是 0，但右边就不知道了。不过因为数组在最后是放了一连串∞，所以很多人就想到一个方法，从头查到尾，想要找出∞这个值。但这很危险：第一，不知道∞的真正的值为何，只知道它比数组中其他数都大，所以没有办法确定∞的位置；第二，纵使有办法认出∞，比如它在第 k 个位置，因而就对 0 与 k 这一段数据进行二分查找，但是这可能会浪费时间，因为数组中不是∞的元素可能十分多。

在此提供一个做法，它并非是最好的，但效率却与二分查找法相当。第一步是找出一个合理的、距离不太长的范围给二分查找法使用。想法是这样的，先看给定的值 GIVEN 是否是 x[0]，如果不是，就去查 x[1]与 x[2]，如果不在这个范围中，则查 x[3]～x[6]，若还不在，再查 x[7]～x[14]…。换句话说，永远在[start,end）这个区间中查有没有 GIVEN，因为 x[ ]是排好的，这相当于

```
x[start] <= GIVEN && GIVEN < x[end]
```

如果不在，就把范围推到 end 开始的区域。这些区域的长度自 $1=2^0$ 起，长度分别为 2、2^2、2^3、2^4 等，所以查找的范围愈来愈大。在程序中，bound()函数就做这样的工作，用 delta 表示区间中元素的个数，start 是左边起点，而 end 则是终点的下一个，所以用的区间是[start,end），而不是[start,end]。很自然地，如果 start 已知，end 就是 start+delta。在一开

Download at http://www.pin5i.com/

始时，start 是 0，delta 是 1，因此范围是[0,1)。接下来，delta 要加倍，因此自己相加，再把 start 移到 end 的位置，这就是下一个区间了。可以一再重复，直到[start,end)中包含了 GIVEN 为止。因为∞比所有的数都大，所以这一定可以做得到，除非 GIVEN 比∞还大，但这样就没有意义了。

例如，下面是一组数：

1、3、7、9、15、18、24、35、79、101、184、300、464、570、894、900、1111、∞、∞、……

想找出有没有 1000。在表 13-1 中列出寻找过程中的各个变量值。

表 13-1

delta	start	end	[start,end)
1	0	1	1
2	1	3	3、7
4	3	7	9、15、18、24
8	7	15	35、79、101、84、300、464、570、894
16	15	31	900、1111、∞、∞

到了[15,31)的范围就包含了 1000 的值，换句话说，值在[15,30]之间（如果有的话），接着再用二分查找法对[15,30]这个范围查找就行了。

程序中 inf_search()是主控函数，它用 bound()找出 GIVEN 所在的界限，然后用 search()在这段范围中进行二分查找。

【习题】

（1）另一个常见的写法是在 bound()函数中用 Fibonacci 数 1,1,2,3,5,8,13,…，而不是 $2^0,2^1,2^2,2^3,\cdots$，请把程序改成这个做法，看看程序效率有何差异。

（2）如果数组中有 n 个元素，请问这个程序一共作了多少个比较？

（3）为了方便起见，把∞定成该类型的极大值，如果∞只是比数组中每一个元素都大，而又不知道这个值是多少，这个程序还能够正常运行吗？为什么？如果不能，用什么方法克服？程序效率会不会大打折扣？请举一个程序不能正常运行的例子（如果程序真的不能正常运行的话）。

（4）如果数组中除了∞之外，其他元素都不相同；在这种情况下有没有办法找出∞的值是什么？需要很多次比较吗？请在这个条件下重做第（3）题。

【程序】

程序 13-3（INF_SRCH.C）

```
/* ------------------------------------------------------------ */
/* FUNCTION infinite search :                                   */
/*   Given a sorted infinite array ended with unknown           */
/* number of INFINITEs, this function perform an extended       */
/* binary search to find a particular input key.                */
```

Download at http://www.pin5i.com/


```
/*                                                                  */
/* Copyright Ching-Kuang Shene          July/16/1989               */
/* ------------------------------------------------------------     */

#define  NOT_FOUND -1

/* ------------------------------------------------------------     */
/*                  function prototypes                             */
/* ------------------------------------------------------------     */

void  bound(int [], int, int *, int *);
int   search(int [], int, int, int);

/* ------------------------------------------------------------     */
/* FUNCTION inf_search :                                            */
/*   The control routine.                                           */
/* ------------------------------------------------------------     */

int  inf_search(int x[], int GIVEN)
{
    int  left, right;

    bound(x, GIVEN, &left, &right);          /* dound               */
    return search(x, GIVEN, left, right);    /* then search         */
}

/* ------------------------------------------------------------     */
/* FUNCTION bound :                                                 */
/*   Given a sorted infinite array and a key GIVEN, this            */
/* function returns two subscripts, namely start and end,           */
/* which bound the given key.                                       */
/* ------------------------------------------------------------     */

void  bound(int x[], int GIVEN, int *start, int *end)
{
   int  delta = 1;                   /* interval length              */

   *start  = 0;                      /* starting from the 1st pos    */
   *end = *start + delta;            /* interval ends before *end    */
   while (!(x[*start] <= GIVEN && GIVEN < x[*end]== {
        delta += delta;              /* if [start,end] can not       */
        *start = *end;               /* bound, then double length    */
```

Download at http://www.pin5i.com/


```
            *end   = *start+delta;    /* and try again.                  */
        }
        (*end)--;                     /* returns the true bound          */
    }

    /* ------------------------------------------------------------------ */
    /* FUNCTION search :                                                  */
    /*   The traditional binary search function.                          */
    /* ------------------------------------------------------------------ */

    int  search(int x[], int GIVEN, int low, int high)
    {
        int  mid;

        while (low <= high) {
            mid = (low + high)/2;
            if (GIVEN < x[mid])
                high = mid - 1;
            else if (GIVEN > x[mid])
                low = mid + 1;
            else
                return mid;
        }
        return NOT_FOUND;
    }
```

问题 4.4　寻找极小值（CYCLEMIN.C）

说一个数组是以循环顺序排列的，这就是说在数组中有某个元素 i，自 x_i 起有这样的关系 $x_0 < x_1 < x_2 < \cdots < x_{i-1}, x_i < x_{i-1} < \cdots < x_n < x_0$；比如说，8,10,14,15,2,6 这 6 个元素就是循环顺序排列的，因为自 2 起为递增，到了最后一个元素就转折为第一个元素，再依顺序递增。换句话说，如果把 $x_i, x_{i+1}, \cdots, x_n$ 取出，并且接到数组开头，就是一个从小到大的排列顺序（这就是个旋转的动作）。请写一个程序，接收一个以循环顺序排列的数组，把它的极小值找出来；以上面数据为例，程序应该会输出 2。

【解答】

解决的办法还是二分查找法，也许会质疑这个数组并没有完全依顺序排列。不错，但不要以为二分查找法只能做那么一点事，事实上，只要能够把问题剖成两部分，而有办法判定解答在其中的一部分的话，这就是个二分查找法。例如，现在处理 x_L 与 x_R 之间的

Download at http://www.pin5i.com/

元素（含两个端点），取中间元素 x_M， $M=(L+R)/2$。于是就会出现下面两种情况：

（1）$x_L \leqslant x_M$：换句话说，x_M 比左边的元素大，因为从左到右是递增的，一直到极小值出现才不升反降，但 $x_L \leqslant x_M$ 明确指出 x_L 到 x_M 是递增，因而极小元素不会在 L 与 M 之间，因此下一个 L 就是 M+1。

（2）$x_L > x_M$：因为从 x_L 起是递增，一直到极小值出现才会降下来，然后自极小值起才再度上升，而且 $x_L \geqslant x_R$。现在有 $x_L > x_M$，这表示 L 与 M 中一定有极小值才会出现这个现象，所以下一个 R 是 M。

就这样一直反复，总会出现 L=R 的时候，于是 x_L 就是极小值，L 就是它的所在。

在程序 CYCLEMIN.C 中，left、right 与 mid 分别取代此地为 L、R 与 M，而且 cycle_min() 函数只返回极小值的脚码，而不是返回极小值，所以要找出极小值，就要用该脚码取出数组中对应的元素。

【习题】

（1）在程序中只要加一行叙述就可以找出最大值，再加一行叙述则可以求出中位数，请试试看。

（2）请证明极小值永远在 x_L 与 x_R 之间。

（3）程序一共作了多少次比较，请求出一个确切的公式。

【程序】

程序 13-4（CYCLEMIN.C）

```
/* ------------------------------------------------------------ */
/* FUNCTION  cyclic_min :                                       */
/*   Given a sorted array stored in a cyclic fashion,           */
/* this function locates the minimum with a modified            */
/* binary search.                                               */
/*                                                              */
/* Copyright Ching-Kuang Shene          July/26/1989            */
/* ------------------------------------------------------------ */

int  cyclic_min(int  x[], int n)
{
    int  left  = 0;
    int  right = n - 1;
    int  mid;

    while (left < right= {       /* if there is a gap :          */
        mid=(left + right)/2;    /* get the middle item          */
        if (x[mid] < x[right]=   /* if < the right most          */
            right = mid;         /* then chop right part         */
        else                     /* if >= the left part          */
            left  = mid + 1;     /* then chop right part         */
```

Download at http://www.pin5i.com/


```
        }
        return left;          /* answer is the item                    */
    }
```

问题 4.5　两个数组的中位数（MEDIAN2.C）

已知两个数组 x[ ]与 y[ ]，各有 n 个元素，并且已经以从小到大的顺序排好，请写一个程序，用少于 n 次比较找出 x[ ]与 y[ ]合并后的中位数。

【解答】

其实在问题说明中的 $(\log_2 n)^2$ 方法已经差不多可以说是涉及到重点所在了，它之所以到不了 $\log_2 n$ 的境界，是因为每次所去掉的、不可能包含了中位数的元素个数不够多。为了要去掉足够多的、不适合的元素，要把那个办法动一点手脚才行。想法是，一次要去掉将近一半的元素（就像是二分搜寻法一样），大约 $\log_2 n$ 次之后，就可以得到结果了。

首先，拿 x[mid_x]与 y[mid_y]这两个在 x[ ]与 y[ ]中央的元素比较。如果 x[mid_x]比 y[mid_y]小，那么在合并后的结果中 x[mid_x]就排在 y[mid_y]前面，而且中位数一定在这两者之间。原因是在 x[ ]中，在 x[mid_x]之前有 n/2 个元素，同理，y[ ]中在 y[mid_y]之前也有 n/2 个元素，所以合并后至少会包含了所有小于等于 x[mid_x]与 y[mid_y]的元素，这样就有了 n 个。但因为 x[mid_x]小于等于 y[mid_y+1]，所以 x[ ]中比 x[mid_x]大的部分也有若干元素落在 y[mid_y]的前面，所以合并后在 y[mid_y]之前就有不止 n 个元素。换句话说，中位数一定在 y[mid_y]之前。正因为如此，在 y[ ]中比 y[mid_y]大的元素中就不可能含有中位数了，所以 y[mid_y+1]～y[n-1]的元素就可以去掉，只留下 y[0]～y[mid_y]，这样 y[ ]中就被去掉了差不多一半的元素。

再看 x[mid_x]。在 x[ ]中比它小的不足 n/2 个元素，因为它小于等于 y[mid_y]，因此在 y[ ]中最多也就只有 y[0]～y[mid_y-1]这 n/2－1 个元素会小于 x[mid_x]，所以比 x[mid_x]小的元素最多也不过有 2(n/2-1)=n-2 个而已，因此中位数不会比 x[mid_x]小。换句话说，中位数不会出现在 x[ ]中比 x[mid_x]小的部分。所以 x[ ]中 x[0]～x[mid_x-1]这些元素也可以去掉，因而 x[ ]中也几乎去掉了一半的元素。

综合起来，如果 x[mid_x]小于等于 y[mid_y]，那么中位数就只会在 x[mid_x]～x[n-1]，或 y[0]～y[mid_y]之间，有一半元素被去掉了，同样的讨论也可以应用在 x[mid_x]比 y[mid_y]大的情况，只不过把 x[ ]与 y[ ]的角色互换而已。因为每次可以去掉一半，所以最后一定会在 x[ ]与 y[ ]中各剩一个，这些就是中位数了，一共需要 $\log_2 n$ 次比较。然而，这是理想的状况，因为在程序语言中的整数运算，不会每次都可以正好去掉一半，而在求中间的位置 mid_x 与 mid_y 时也正好在一半，因此要留心一个问题，就是到最后 x[ ]与 y[ ]中可能会留下两个相邻的元素，即 x[mid_x]与 x[mid_x+1]，或 y[mid_y]与 y[mid_y+1]，所以要特别处理。

请看程序 MEDIAN2.C，用 first_X、last_X、first_Y 与 last_Y 分别记录下 x[ ]与 y[ ]中

Download at http://www.pin5i.com/

可能包含有中位数的段落，而 mid_X 与 mid_Y 则分别指出这两个段落的中点；另外，count 记录到目前为止，去掉比中位数小的值的总数。于是在 while 循环中，用 x[mid_x]与 y[mid_y]比较，依照上述方法去掉 x[ ]与 y[ ]中的一半元素，并且把比中位数小而又被去掉的元素个数加到 count 中。注意，固然不知道真正中位数的值是多少，但因为在 x[mid_x]小于等于 y[mid_y]时，比 x[mid_x]小的值一定也小于中位数，所以把这些元素个数加到 count 中，就可以累计得出到目前为止删除了多少个比中位数小的元素。

当 first-X 与 last_X，以及 first-Y 与 last_Y 的差小于等于 1 时，x[ ]与 y[ ]中就最多各剩下两个元素，把剩下的元素取出放到数组 z[ ]中，再用 sort()排好顺序，因而中位数一定在 z[ ]中。很简单，已经删除了 count 个比中位数小的元素，而中位数应该在第 n 位，不足的部分自然在 z[ ]中，因为还少 n-count 个，而 z[ ]数组脚码从 0 起，因此中位数就在 z[n-count-1]中。

在程序中多了一个 sort()的程序，会不会影响到 $\log_2 n$ 次比较的特性呢？看看 sort()，如果 z[ ]中有 4 个元素，就有 6 次比较，3 个与 2 个元素时，分别用 3 次与 1 次比较就够了，因此与 n 无关。当然可以说一共作了最多 $\log_2 n+6$ 次比较，但是当 n 很大时，6 这个值没多大影响，所以就说成 $\log_2 n$ 也无妨。不过当 n 比较小时，就会使用比较多的比较次数了。但应注意，所谓的效率问题，只有在 n 比较大时才有意义。以 n=128 为例，$\log_2 n$ 是 7，因此问题说明中的合并方法要有 128 次比较，二分查找法要 $7^2=49$ 次比较，但这里却只有 $7+6=13$ 次比较，优劣就相当明显了。

【习题】

（1）请把 $(\log_2 n)^2$ 次比较的方法写成程序，与这里的程序就时间上作一比较。

（2）请把这里的程序扩充成 x[ ]与 y[ ]的元素个数不同，请问程序要作多少次比较？

（3）请举出一个例子，说明 while 中如果改成 first_X==last_X，以及 first_Y ==last_Y 是不智之举。请问这样做会有什么后果？

（4）程序中不要 sort()可不可以？因为并不需要把 z[ ]中元素排出顺序。修改程序，修改后会比较快吗？程序会比较简单吗？

【参考文献】

笔者最先是在 S.Baase 的书中第 3 章习题 3.15 看到这个问题的，但在思考时马上就发现他的结果并不是最好的，而得到这里的解法。但不久之后，在读 Michael Ian Shamos 的一篇文章时，发现他也证明了 $\log_2 n$ 次比较可以做出结果，并且指出这个解法来自 D.Jefferson，笔者没见过 Jefferson 的做法，但相信与这里的做法差不多。另外，Shamos 的解法与这里相同。

1 S.Baase. Compute Algorithms, Introduction to Design and Analysis. 2nd ed., 1988

2 M.I.Shamos. Geometry and Statistics: Problems at the Interface, in Algorithms and Complexity. New Directions and Recent Results, edited by J.F.Traub, Academic Press, 1976

【程序】

程序 13-5（MEDIAN2.C）

```
/* ------------------------------------------------------------------ */
/* FUNCTION  median :                                                 */
```

Download at http://www.pin5i.com/


```
/*   Given two sorted arrays x[] and y[] with n items,          */
/* this function picks up the median of the merged array        */
/* by using a binary search in log(n) comparisons.              */
/*                                                              */
/* Copyright Ching-Kuang Shene          Aug/14/1989             */
/* ------------------------------------------------------------ */

void  sort(int [], int);

int  median(int x[], int y[], int n)
{
    int  first_X = 0;              /* lower element of x[]        */
    int  first_Y = 0;              /* lower element of y[]        */
    int  last_X  = n-1;            /* higher element of x[]       */
    int  last_Y  = n-1;            /* higher element of y[]       */
    int  count   = 0;              /* # of smaller items elim.    */
    int  mid_X, mid_Y;             /* middle element pointers     */

    int  number;                   /* # of elements left          */
    int  z[4];                     /* working array               */

    while ((last_X - first_X > 1) || (last_Y - first_Y > 1)) {
        mid_X = (first_X + last_X)/2;    /* get mid ptrs          */
        mid_Y = (first_Y + last_Y)/2;
        if (x[mid_X] <= y[mid_Y]) {
            count   += (mid_X - first_X);/* inc. count            */
            first_X =  mid_X;     /* elim. lower half x[]         */
            last_Y  =  mid_Y;     /* elim. higher half x[]        */
        }
        else {
            count   += (mid_Y - first_Y);
            first_Y =  mid_Y;     /* elim. lower half y[]         */
            last_X  =  mid_X;     /* elim. higher half x[]        */
        }
    }

    for (number = 0; first_X <= last_X; first_X++)
        z[number++] = x[first_X];    /* collect remainder         */
    for ( ; first_Y <= last_Y; first_Y++)
        z[number++] = y[first_Y];

    sort(z, number);     /* sort them                             */
```

Download at http://www.pin5i.com/


```
    return z[n-count-1];     /* pick up appropriate item          */
}

/* ------------------------------------------------------------     */
/* FUNCTION  sort :                                                 */
/*   Special routine to sort small arrays with 2, 3 and             */
/* 4 elements.                                                      */
/* ------------------------------------------------------------     */

#define  SWAP(x, y)  { temp = x; x = y; y = temp; }

void  sort(int z[], int n)
{
    int  temp;

    switch (n) {
        case 4 : if (z[0] >= z[3])  SWAP(z[0], z[3]);
                 if (z[1] >= z[3])  SWAP(z[1], z[3]);
                 if (z[2] >= z[3])  SWAP(z[2], z[3]);
        case 3 : if (z[0] >= z[2])  SWAP(z[0], z[2]);
                 if (z[1] >= z[2])  SWAP(z[1], z[2]);
        case 2 : if (z[0] >= z[1])  SWAP(z[0], z[1]);
    }
}
```

问题 4.6　寻找中间值（B_SEARCH.C）

已知一个整数数组 x[ ]有 n 个元素。对于任意两个相邻元素 x[i]与 x[i+1]而言，它们的差的绝对值都不超过 1，用数学式来写就是$|x[i]-x[i+1]|\leqslant 1$，而且 $x[0]<x[n-1]$。请写一个程序，在 x[ ]数组中找出它是否有某一个输入的值 a，$x[0]\leqslant a\leqslant x[n-1]$，程序一共作了多少次比较？能够做到少于 n 次吗？

【解答】

只要把任何一个进行二分查找（Binary Search）的程序找出来，原封不动就是解答。原因请看下面的说明。

记得在数学课本中有一个定理是这样说的：如果 f(x) 在 [a, b] 中是连续函数，而且 f(a) 与 f(b) 异号，那么在 [a, b] 中 f(x) 就一定至少会有一个根。也就是说，在 [a, b] 中有一个 α 满足 f(a) = 0 的关系，如图 13-1a 所示。

Download at http://www.pin5i.com/

这个定理可以推广，如果 f(x) 在 [a, b] 中连续，而且 f(a) ≤ f(b)，于是对于任何一个 c，f(a)≤c≤f(b)，在[a,b]中一定至少会有一个值 α 使 f(α) = c；这个说法就是课本上的中间值定理（Intermediate Value Theorem）。注意，说至少有一个，而并没有指出是哪一个。中间值定理是求根的定理的推广，如果把 X 轴平移到 y = c 这条水平直线上，那就等于求 [a, b] 之间的根，如图 13-1b 所示。

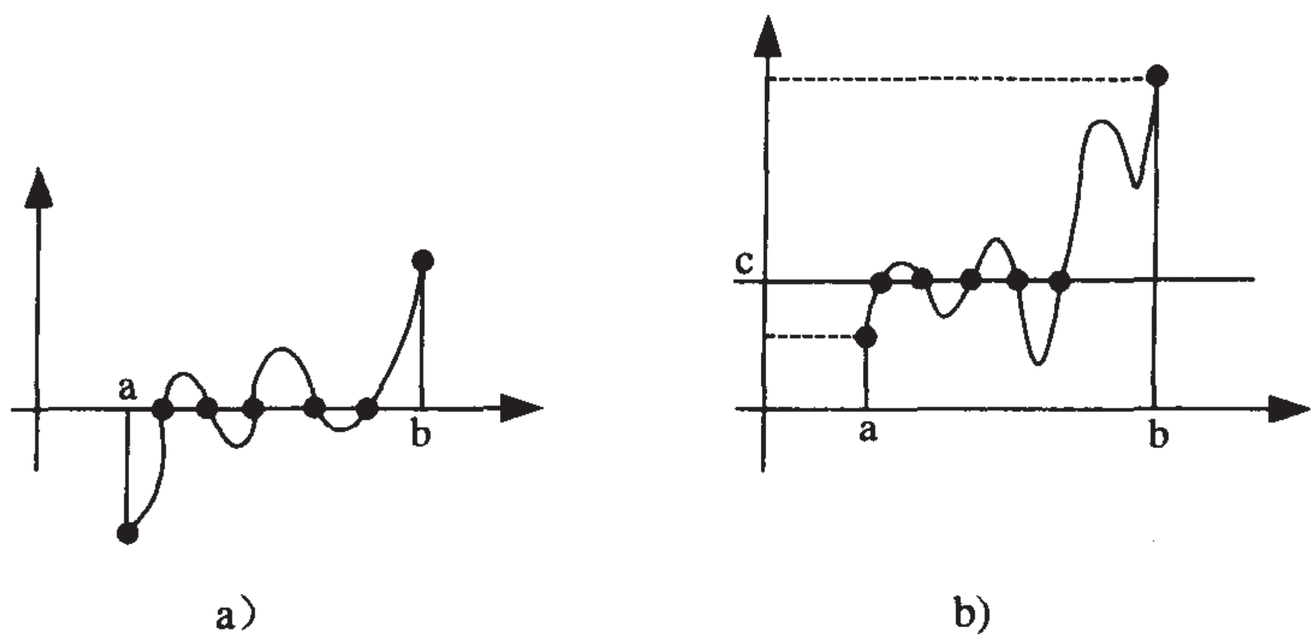


图 13-1

如何求[a,b]之间 f (x)=0 的根呢？都知道一个二分法查找（可见二分查找法不是计算机科学的专利），取 c=(a+b)/2，看 f (c)的值的符号，如果与 f (a)相同，那就一定与 f (b)相异，因此根就在[c,b]之间，因而可以不必理会[a,c]；如果 f (c)符号与 f (a)相异，那么根就在[a,c]之间，于是就不必管(c,b)。不管是哪种情况，原区间的长度都减半，如果一再反复这个过程，当区间的长度愈来愈小时，就愈来愈接近其中的一个根。

如果 f (a)与 f (b)并非异号，而想求[a,b]中哪一个点 α 会使 f(α) = c 时，这里 c 在 f (c)与 f (b)之间，做法也是一样的。还是可以用求根的方法，只是要考虑 f(a) − c 与 f(b) − c 而已。因为 c 在 f (a)与 f (b)之间，所以 f(a) − c 与 f(b) − c 异号，取 a 与 b 的中点 d = (a + b)/2，看看 f(d) − c 与哪一个异号，因而照样可以找到一个 α 使 f(α) − c = 0，那就是 f(α) = c 了。

那么题目与上面的讨论有何关联呢？只要把 0、1、2、3、… n − 1 想成是 [0, n − 1] 中的点，函数值 f(i) 就是 x[i]，而且区间只有整数，函数值也是整数，就会明白了。或许有一个疑虑，这个函数并不连续，不过那是对实数而言；对整数来说，|x[i]-x[i-1]|≤1 的关系就相当于连续的条件，因为 x[i-1]的下一个点只能是 x[i-1]+1、x[i-1]或 x[i-1]-1 其中之一，换句话说，x[0]～x[n-1]这 n 个点在整数的角度看是连续不断的，所以也就相当于连续的条件。

如果放开这一点不谈，也可以证明 x[ ]数组中一定含有 x[0]～x[n-1]之间的每一个值（习题），所以用二分搜寻法还是有用的。所以若要找的值是 key，下面的条件要成立：

```
key >= x[0] && key <= x[n-1]
```

然后用二分搜寻法即可，请注意，只能找 x[0]～x[n-1]之间的值，原因也可以从中间值定理来说明，也可以直接证明它。

前面提到过，f (x)在[a,b]连续，f (a)与 f (b)异号，那么 f (x)=0 在[a,b]之间一定有一个根，就是所谓的 Bolzano 定理，是由 Bernard Bolzano（一位天主教的神父，1781—1848 年）发现的。

Download at http://www.pin5i.com/

【习题】

（1）如果 x[ ]满足本题的特性，而且 c 的值在 x[ ]与 x[n-1]之间，请用 x[ ]的特性证明 x[ ]中必定有 c。

（2）请证明，如果 x[i]<c，那么 c 一定是 x[i+1]、…、x[n-1]其中之一；又若 x[i]>c，则 c 一定是 x[0]、…、x[i-1]其中之一。如果会证明这一题与上一题，就相当于证明了 Bolzano 定理，即中间值定理在整数区间中的形式。

（3）在处理文本文件的时候，常常要把一列交给某一个不是写的函数做分析的工作，只有那个函数返回 OK 时才能正常运行，如果返回 ERROR，就表示那一列出现了问题。不过那个程序却很笨，无法告诉问题出在那一列的具体地点。请问，要如何在自己的程序中加上一些内容，通过该函数找出是哪一个符号出了问题呢？假设已经知道了问题出现在哪一列。

【参考文献】

二分查找法不用进一步说明，所以就不必列举参考文献，不妨查阅任何一本标准教科书。Balzano 定理的讨论并不一定要看专著，有兴趣的读者不妨读一读下面这本名著。

R.Courant and H.Robbins. *What is Mathematics*. Oxford University Press, 1941

【程序】

程序 13-6（B_SEARCH.C）

```
/* ------------------------------------------------------------ */
/* FUNCTION  bolzano_search :                                   */
/*   This is the discretized Bolzano algorithm for              */
/* continue equation solving.  It is simply a bineary           */
/* search, but the key to be searched must be in between        */
/* the range of two end points.                                 */
/*                                                              */
/* Copyright Ching-Kuang Shene          July/28/1989            */
/* ------------------------------------------------------------ */

#define   ERROR   -1

int  bolzano_search(int x[], int n, int key)
{
    int  low  = 0;
    int  high = n-1;
    int  mid;

    if(key > x[n-1] || key < x[0]= /* reject illegal in        */
        return ERROR;
```

Download at http://www.pin5i.com/


```
        while (low <= high= {   /* simply a binary search          */
            mid = (low + high)/2;
            if (key > x[mid])
                low  = mid + 1;
            else if (key < x[mid])
                high = mid - 1;
            else
                return mid;      /* search succeeds always          */
        }
    }
```

问题 4.7　3 个数组的共同元素（SEARCH3.C）

有 3 个数组 x[ ]、y[ ]与 z[ ]，各有 x、y 与 z 个元素，而且三者都已经从小到大依序排列。请写一个程序，找出值最小的共同元素（也就是同时在 3 个数组中出现，并且值最小的元素），若没有共同元素，请显示合适的信息。

【解答】

因为 3 个数组 x[ ]、y[ ]和 z[ ]已经从小到大依序排列，这是一个很好的已知条件，充分利用可以开发出效率很高的程序。

假设从头开始到达 x[i]、y[j]与 z[k]时，这 3 个元素并不完全相同（可能其中有两个相同，也可能完全不同），那么为了要找出是否有三者一样的元素时，下一步该如何进行？下面就是简单的分析。

（1）如果 x[i]<y[j]：因为 y[j]比较大，所以在 x[ ]中如果要有与 y[j]相等的元素，那就一定排在 x[i]后面，因此依自然的做法就是将 i 加 1，移到 x[ ]的下一个元素。

（2）如果 x[i]≥y[j]，但 y[j]<z[k]：由第一个条件，x[i]可能等于 y[j]，但第二个条件，y[j]比 z[k]小，又假设在 x[i]、y[j]与 z[k]之前没有相等的元素，所以如果有三者相同的元素时，一定就会排在 y[j]后面，因此把 j 加 1，移到 y[ ]的下一个元素。

（3）如果 x[i]≥y[j]，且 y[j]≥z[k]，但是 z[k]<x[i]：这就失去了三者相等的机会，根据上面讲述的理由，只能把 k 加上 1。

（4）若 z[k]≥x[i]，那么 x[i]、y[j]与 z[k]三者相同，换句话说，就找到了 3 个相同的元素了。

在程序中，while 循环的作用是控制着使 3 个脚码*XX、*YY、*ZZ 不超过 x[ ]、y[ ]与 z[ ]的范围，只要有一个脚码超过范围，函数值就返回 NOT_FOUND。循环中的 if…else 叙述就是上面的程序逻辑，如果有 3 个相同的元素，函数值会传回 FOUND，而且*XX、*YY、*ZZ 正是那 3 个相同的元素在 x[ ]、y[ ]与 z[ ]中的脚码。

Download at http://www.pin5i.com/

【习题】

（1）有一个想法是，把 x[i]、y[j]与 z[k]三者中最小的脚码加 1，换句话说，如果 y[j]比 x[i]与 z[k]小，就把 j 加 1。请问用这个方法写成的程序与上述解答相比，哪一个效率高一些？这个想法正确吗？请证明它。

（2）请证明解法最多只用了 3min(X,Y,Z)次比较。在什么情况下会有那么多比较呢？

（3）在前面看到过一个类似的程序——EQ_COUNT.C，它处理两个数组的情形，此处的程序是处理 3 个数组，能否写出一个处理 4 个数组的程序？

（4）程序在处理数组时，已知元素的数量，但若改成处理顺序文件，这个元素的数量就不知道了。请改写这个程序，让它能处理顺序文件。

（5）如果数组 x[ ]、y[ ]与 z[ ]并没有依顺序排列，程序能正常运行吗？请举出一个反例。如果不能正常运行的话，如何克服这一层困难？

【参考文献】

笔者是从 D.Gries 的名著中看到这个题目的：

1 D.Gries. *The Science of Programming*. Springer-Verlag, 1981

Gries 把它叫做 Welfare Crook 问题，并且这个问题是 1972 年在德国 Marktoberdorf 举行国际夏令研习时，由 E.Dijkstra 的同事 Wim Feijen 提出的一个习题。请参看：

2 F.L.Bauer and K.Samelson(eds.). *Language Hierarchies and Interfaces*. Lecture Notes in Computer Science, No.46, Springer-Verlag, 1976

【程序】

程序 13-7（SEARCH3.C）

```
/* ------------------------------------------------------------ */
/* FUNCTION search :                                            */
/*   Given three sorted arrays, this functions locates          */
/* the first common element common to all three arrays          */
/* (if any).                                                    */
/*                                                              */
/* Copyright Ching-Kuang Shene          July/17/1989            */
/* ------------------------------------------------------------ */

#define  FOUND   1
#define  NOT_FOUND   0
int  search(int x[], int y[], int z[], int X, int Y, int Z,
                                        int *XX, int *YY, int *ZZ)
{
    *XX = *YY = *ZZ = 0;
    while (*XX < X && *YY < Y && *ZZ < Z)
        if (x[*XX] < y[*YY])
            (*XX)++;
```

Download at http://www.pin5i.com/


```
        else if (y[*YY] < z[*ZZ])
            (*YY)++;
        else if (z[*ZZ] < x[*XX])
            (*ZZ)++;
        else
            return FOUND;
    return NOT_FOUND;
}
```

问题 4.8 寻找最小与次小元素（IST&2ND.C）

请写一个函数，接收一个数组，找出该数组中最小与次小的元素的值。

【解答】

“分而治之”的策略是：把 n 个元素剖成差不多相等的两半，各有 n/2 个元素。于是递归地找出左、右两半的最小与次小值，就有了 4 个值，F1 与 S1 表示左半的最小与次小值，而 F2 与 S2 则是右半的最小与次小值。如何“合”呢？记住 F1<S1,F2<S2 这两个条件，因此可以这样做：

（1）在 F1<F2 的情况下，F1 自然是最小值，但 F2 却不一定是次小的，但可以确定的是 S2 绝对不会是次小值，因为它比 F2 大。所以次小值就是 F2 与 S1 其中之一，再次比较就可以找出来。

（2）在 F1≥F2 的情况下，F2 当然就是最小值；次小值就是 F1 与 S2 其中之一，再次比较就会得到答案。

于是，在合的部分永远要用两次比较。如果令 T(n) 表示 n 个元素的比较次数，因为要分成两半，各 n/2 个元素，因此各要 T(n/2) 次比较，所以在“分”的动作中就用了 2T(n/2) 次，再加上“合”的动作中有两次，所以就得到：

$$T(n) = 2T(n/2) + 2$$

把它多展开几项，继而得到：

$$\begin{aligned} T(n) &= 2T(n/2) + 2 \\ &= 2[2T(n/2^2) + 2] + 2 \\ &= 2^2 T(n/2^2) + 2 + 2^2 \\ &\vdots \\ &= 2^i T(n/2^i) + 2 + 2^2 + \cdots + 2^i \end{aligned}$$

当分到只有两个元素时，一次比较就可以定出先后（只有一个元素时不用比较，所以不去管它），即 $T(2)=1$。当 $i=\log_2 n-1$ 时，令 $n/2^i=2$，得：

$$\begin{aligned} T(n) &= 2^{\log_2 n-1} T(2) + 2 + 2^2 + \cdots + 2^{\log_2 n-1} \\ &= 2^{\log_2 n-1} + 2 + 2^2 + \cdots + 2^{\log_2 n-1} \end{aligned}$$

Download at http://www.pin5i.com/

$$= n/2 + n - 1$$
$$= 1.5n - 2$$

上式中的右边是个等比级数，结果是 $n-2$。很明显地，$1.5n-2$ 次比较要比 $2n-3$ 次少。但用分而治之的方法快吗？不快。难道理论有错吗？也不是的，原因是只算了比较的次数而已，程序执行时还有其他因素会影响执行的时间。

假定在分而治之的方法中，处理递归与其他工作与比较次数成正比，于是可以说分而治之的技巧在函数中每一次用了 2α 的时间，用上面的计算技巧，可以得到：

$$t(n) = 2^{\log_2 n-1} t(\alpha) + \alpha[2 + 2^2 + \cdots + 2^{\log_2 n-1}]$$
$$= \alpha(1.5n - 2)$$

式中 $t(n)$ 是 n 个元素时所用的时间。同理，问题说明中的简单方法则用了 $(2n-3)\beta$ 的时间，这里 β 是该方法中为了进行一次比较所用的时间。β 远小于 α，因为 β 的时间不过是一个简单的循环与设定叙述，但 α 却包含了递归的动作。有了这两个式子，不妨分析一下，看 n 要在什么情况之下采用分而治之的方法比较快。

程序本身是很简单的，并没有用到特殊的技巧，不过却用了一个常见的手法。因为“分而治之”时常常要用到数组中某个部分，所以原来的函数调用方式（只传达共享多少个元素）就不很妥当了，因此加上一个极端简单的函数做这一转换的工作，本书中不少题目都用了这个方法（比如快速排列法），应该不难了解。

【习题】

（1）请修改程序，可以找出最小、次小及第三小的元素，用分而治之与问题说明中的方法各做一次，哪一个比较快？各用了多少次比较？

（2）请发展一个更快的程序，理论上最好的方法是 $n + \log_2 n$ 次，能否做到？

（3）请写一个程序，同时找出一个数组中的极大与极小值，程序一共有多少次比较？

【程序】

程序 13-8（IST&2ND.C）

```
/* ------------------------------------------------------------ */
/* FUNCTION  first_second :                                     */
/*   This is a little bit faster function for computing         */
/* the first and the second elements at the same time.          */
/* It employs a recursive  method to implement divide-and-      */
/* conquer technique.  Number of comparisons is about           */
/* 1.5n-2, where n is the number of elements.  The fastest      */
/* algorithm use n+log(n) comparisons only, while the           */
/* conventional one requires 2n-3 comparisons.                  */
/*                                                              */
/* Copyright Ching-Kuang Shene          July/27/1989            */
/* ------------------------------------------------------------ */

#include <limits.h>
```

Download at http://www.pin5i.com/


```
void  f_and_s(int [], int, int, int *, int *);

void  first_second(int x[], int n, int *first, int *second)
{
     f_and_s(x, 0, n-1, first, second);
}

/* ----------------------------------------------------------------- */
/* FUNCTION  f_and_s :                                               */
/*   This function accepts the input array x[] and the               */
/* left and right bounds, then returns the smallest and              */
/* the second smallest elements in this range by using               */
/* divide-and-conquer method.                                        */
/* ----------------------------------------------------------------- */

void  f_and_s(int x[], int left, int right, int *f, int *s)
{
    int  mid;
    int  F1, F2;              /* returned smallest items             */
    int  S1, S2;              /* returned second smallest            */

    if (left > right)         /* range empty ?                       */
        *f = *s = INT_MAX;    /* YES, return INT_MAX                 */
    else if (left == right)  /* exactly one item ?                  */
        *f = x[left], *s = INT_MAX;  /* return it and inf            */
    else {
        mid = (left + right) /2; /* now cut from middle             */
        f_and_s(x, left, mid, &F1, &S1);     /* left                 */
        f_and_s(x, mid+1, right, &F2, &S2); /* right                 */
        if (F1 < F2=          /* pick 1st and 2nd items.             */
            *f = F1, *s = (S1 < F2) ? S1 : F2;
        else
            *f = F2, *s = (S2 < F1) ? S2 : F1;
    }
}
```

问题 4.9　查找矩阵（M_SEARCH.C）

已知有 n 列 n 行的矩阵 ***M***，它的元素满足一个很特殊的性质，即对于任一元素 $M_{i,j}$ 而言，都会小于在它右边与下方的元素（如果存在的话），换句话说，$M_{i,j} < M_{i,j+1}$，而且 $M_{i,j} < M_{i+1,j}$，现在又有一个值 K，请写一个程序，检查矩阵 ***M*** 中是否存在 K。

Download at http://www.pin5i.com/

【解答】

以下的解法源自 Eric S.Rosenthal。

在问题说明中提示过，这个矩阵中的每一列、每一行都是从小到大排列好的，对于 K 这个值而言，*M*矩阵中小于或等于它的元素所在的区域就像是楼梯的形状（如图 13-2 所示）。

因为每一列每一行的元素都已经排好了，所以分界线上的元素就是那一个区域中的那一列或那一行的最大值，但是该区域是由小于等于 K 的元素组成的，所以若 K 在矩阵中，那么 K 就一定会出现在这条分界线上。正因为如此，程序就只要沿着分界线走一次，看看有没有 K 出现就会得到解答。例如，图 13-3 所示是一个 4×4 矩阵，满足题目的要求，现在 K 是 5，于是就有了分界线（图 13-3 中粗线）。

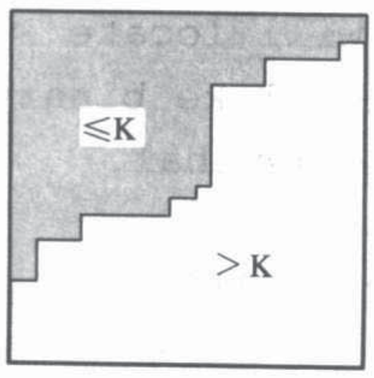


图 13-2

1	3	4	6
2	4	5	8
3	6	7	8
5	7	8	9

图 13-3

因此，走分界线就相当于依次把 4、5、4、3、5（自右上到左下），或 5、3、4、5、4（自左下到右上）查一次就可以找出 5 了。

但是由于并不知道分界线在哪里，不妨从右上角开始，拿那个元素与 K 相比，如果该元素比 K 大，就在大于 K 的区域，于是就往左移，再这样一直往左移，到了小于 K 时，这就进入了小于等于 K 的区域，就可以往下移了。一直往下移，也可能会进入大于 K 的区域，就又往左移，这样就可以查出结果了，如图 13-4 所示说明了如何找出 11 的步骤。

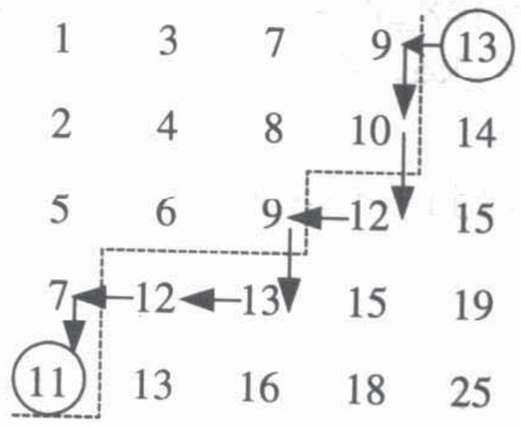


图 13-4

在程序中 i 与 j 分别指出目前在哪一列哪一行，依据那个元素如大于 key 则左移，如小于等于 key 则下移。

【习题】

（1）请证明大于 K 与小于等于 K 的分割方式是正确的，不会有其他的情况出现。

（2）请证明程序在最坏的情况下，最多只会进行 2n－1 次比较。

（3）请修改程序，使它能够找出与 K 相同的位置。

（4）请将这个问题推广到三阶数组，即 $M_{i,j,k}$ 比 $M_{i+1,j,k}$ 小，$M_{i,j+1,k}$ 比 $M_{i,j,k+1}$ 小，请问比较次数最多是多少次？

Download at http://www.pin5i.com/

【参考文献】

1 M.Harrison. Problem 80-3, *Journal of Algorithms*. Vol.1(1980), pp.108

2 E.S.Rosenthal. Solution of Problem 80-3, *Journal of Algorithms*. Vol.3(1982), pp.365

【程序】

程序 13-9（M_SEARCH.C）

```
/* ------------------------------------------------------------ */
/* FUNCTION matrix_search :                                     */
/*   Given a matrix with a special property that for each       */
/* m[i][j] we have m[i][j] < m[i,j+1] and < m[i+1][j],          */
/* also given is a key value, this function will locate         */
/* this value in the matrix m[][] in a fashion like binary      */
/* search.  This method follow from Eric S. Rosenthal.          */
/*                                                              */
/* Copyright Ching-Kuang Shene          July/02/1989            */
/* ------------------------------------------------------------ */

#define  MAXSIZE  10                        /* matrix size       */

typedef  int MATRIX[MAXSIZE][MAXSIZE];      /* matrix type       */

void matrix_search(MATRIX mat, int n, int key, int *index_x, int *index_y)
{
    int  i = 0, j = n-1;      /* start from upper right          */

    for (*index_x = *index_y = -1; i < n && j >= 0; =
        if (mat[i][j] < key=          /* element too small ?     */
            i++;                      /* move down one row.      */
        else if (mat[i][j] > key)     /* element to large ?      */
            j--;                      /* move left one column    */
        else {
            *index_x = i;             /* found, record locations */
            *index_y = j;
            return;                   /* and return              */
        }
    return;
}
```

问题 4.10　表示成两个数平方和（TWOSQUAR.C）

已知一个正整数 R，请写一个程序，找出所有满足 $X^2 + Y^2 = R$ 的正整数对 X 与 Y。

Download at http://www.pin5i.com/

【解答】

假设有一个矩阵，它的第 i 列第 j 行的元素 $M_{i,j}$ 所存储的正是 i^2+j^2，因此这个矩阵的元素满足问题 4.9 的条件（为什么？），如果用问题 4.9 的例子，在这个矩阵中找 R，就可以解决问题了。不过上面讲到的矩阵是无限的，因为 i 与 j 都没有限制，然而就像问题说明中所提到的，i 与 j 都一定小于或等于 $\sqrt{R}$，所以 $\sqrt{R}$ 列 $\sqrt{R}$ 行的矩阵就足够了。基于此，比较 $2\sqrt{R}-1$ 次就可以找出答案，这样比用 R 次比较的方法快 $\sqrt{R}$ 倍。

如果进一步思考，就可能发现 *M* 矩阵根本就是多余的，因为每次使用 $M_{i,j}$ 时，只要把 i^2+j^2 算出来即可。因为只会比较 $2\sqrt{R}-1$ 次，所以 i^2+j^2 也只会算 $2\sqrt{R}-1$ 次，但如果用矩阵的话，因为有 $\sqrt{R}$ 列 $\sqrt{R}$ 行，还要算 $\sqrt{R}\cdot\sqrt{R}=R$ 次，所以就用这样的方式写程序。

【习题】

（1）事实上，计算工作可以再减少一半，大概 $\sqrt{R}$ 次就足够了，因为如果 X 与 Y 是一组解的话，Y 与 X 也一定是一组解。请修改程序完成这件事。

（2）问题 4.9 的习题中要求把矩阵中查找元素的问题推广到三阶矩阵，这个问题解答的同时也就可以解决 $X^2+Y^2+Z^2=R$ 这个问题了。

（3）数论中有一个很有名的定理，它说任何一个自然数都可以表示成最多 4 个平方的和，请用上面的技巧写一个程序，接收一个正整数，把它表示成平方和（当然不会多过 4 个）。

【参考文献】

这个问题的来源大概已经不可考证了，很多教科书中都有这个题目，但是解法都不高明，大约用的都是在问题说明中的方法。这里的解法取自 P.Backhouse 的一篇文章，他最先发现这个题目与上一题的关联，因此就轻而易举地得到一个很漂亮的解法。

R.Backhouse. Writing a Number as the Sum of Two Squares: A New Solution, *Information Processing Letters*, 1982(14), pp.15～17

【程序】

程序 13-10（TWOSQUAR.C）

```
/* ------------------------------------------------------------ */
/* PROGRAM  twosquare :                                         */
/*   Given an integer, this program computes all sums of        */
/* two squares of this given number by using the concept        */
/* discussed in the previous program (M_SEARCH.C).              */
/*                                                              */
/* Copyright Ching-Kuang Shene          Aug/28/1989             */
/* ------------------------------------------------------------ */

#include <stdio.h>     /* for I/O functions                     */
#include <stdlib.h>    /* for atoi()                            */
#include <math.h>      /* for sqrt()                            */
```

Download at http://www.pin5i.com/


```
void main(void)
{
    int given;          /* the given number                    */
    int row, column;    /* row and column indicators           */
    int count;          /* number of solutions                 */
    char line[100];
    printf("\nRepresenting a Given Number as the Sum of Two Squares");
    printf("\n==================================================\n");
    printf("\nAn Integer Please ---> ");
    gets(line);
    given = atoi(line);
    printf("\nCount     X     Y");
    if (row*row + column*column == given) {
    printf("\n-----  -----  -----");

    row  = 1;           /* starts from far enough              */
    column   = (int) (sqrt((double) given) + 0.5);
    count = 0;          /* so solution yet                     */
     while (row <= given && column > 0= /* scan down...        */
        if (row*row + column*column == given){
            count++;
            printf("\n%5d%7d%7d", count, row, column);
            row++;
            column--;
        }
        else if (row*row + column*column > given)
            column--;
        else
            row++;

    if (count == 0)
        printf("\n\nSorry, NO ANSWER found.");
    else
        printf("\n\nThere are %d possible answers.");
}
```

问题 4.11　最大方块区域（MAXSQR.C,MAXSQR2.C）

在一个矩阵中，相连在一起的一块正方形区域就叫做一个子区域，如 $M_{3,5}$、$M_{3,6}$、$M_{3,7}$、$M_{4,5}$、$M_{4,6}$、$M_{4,7}$、$M_{5,5}$、$M_{5,6}$ 与 $M_{5,7}$ 就是矩阵 M 中从 $M_{3,5}$ 起的一个 3×3 的子区域

Download at http://www.pin5i.com/

（Subblock）。请写一个程序，接收一个方阵（列数与行数相同），并且再接收一个已知的值，令其为 K，请找出在给定方阵中值全部是 K 的最大一个子区域，并且报告出这个子区域在矩阵中的位置与它的大小。

【解答（解法一）】

事实上，这个题目的比较次数可以做到与 n^2 成正比，如图 13-5 所示。

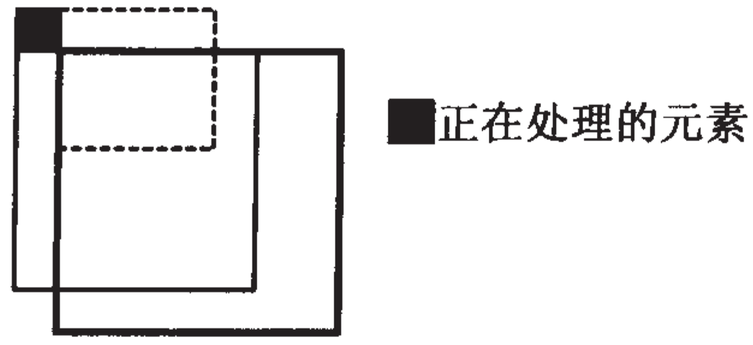


图 13-5

对于任何一个元素 $M_{i,j}$ 而言，如果 $M_{i,j+1}$、$M_{i+1,j}$ 与 $M_{i+1,j+1}$ 开始的最大子区域已经求出，那么求 $M_{i,j}$ 开始的最大子区域就很容易了。解答就是从 $M_{i,j+1}$、$M_{i+1,j}$ 与 $M_{i+1,j+1}$ 起的最大子区域容量中最小的一个再加上 1，如图 13-5 所示，$M_{i+1,j}$ 的子区域是 5×5、$M_{i,j+1}$ 的是 3×3、$M_{i+1,j+1}$ 的是 6×6，因此最小的一个是 3，加上 1 得到 4，所以从 $M_{i,j}$ 起的最大子区域是 4×4。想想为什么，多画几幅图就可以看出来了。

如此看来，因为 $M_{i,j}$ 右下角的结果决定了 $M_{i,j}$ 的结果，所以程序就应该是自下而上、自右而左运行。注意，如果 $M_{i,j}$ 的值与给定的值相同，那么子区域至少会是 1×1，因为这包含了 $M_{i,j}$ 自己，但若 $M_{i,j}$ 的值与给定值不同，子区域就是 0×0 了。于是方阵中最下方的一列各个元素的最大子区域不是 0 就是 1，这只要比一比它们的值是否与给定的值相等即可。因此程序要从倒数第二列起往上做，在每一列上自右往左查。当查到 $M_{i,j}$ 时，要作一次比较，看它的值是否与给定的值一样，如果不一样，$M_{i,j}$ 起的最大子区域就是零；如果一样，就用 3 次比较，由它的右边、右下与下方三角元素的最大子区域容量定出 $M_{i,j}$ 的容量。所以每一个 $M_{i,j}$ 都最多用 4 次比较，又有 n^2 个元素，因此最多 $4n^2$ 次比较就可以得出结果。

注意，因为要用到在 $M_{i,j}$ 之前就算出的最大子区域容量，所以要用另一个矩阵 size[ ][ ] 来保存它；换言之，size[i][j]就是在 $M_{i,j}$ 起的最大子区域容量。了解这些观念后程序就不难写了，请自己研究这个程序。

【程序】

程序 13-11（MAXSQR.C）

```
/* ------------------------------------------------------------------ */
/* FUNCTION maxsquare :                                               */
/*   Given a square matrix and a key, this function scans             */
/* the input matrix to find out the maximum size of a                 */
```

Download at http://www.pin5i.com/


```
/* square sub-block such that all elements of this sub-        */
/* block are all the same.                                      */
/*                                                              */
/* Copyright Ching-Kuang Shene          July/03/1989            */
/* ------------------------------------------------------------ */

#define  MAXSIZE    10
#define  min(a,b,c)  (((a) < (b)) ? ((a) < (c) ? (a) : (c)) \
                                  : ((b) < (c) ? (b) : (c)))

typedef  int MATRIX[MAXSIZE][MAXSIZE];

void maxsquare(MATRIX x,int n,int key,int *row,int*column,int*width)
{
    MATRIX  size;                       /* a working matrix        */
    int     i, j;                       /* working storages        */
    for (j = 0; j < n; j++)             /* copy last row of matrix */
        size[n-1][j] = x[n-1][j];       /* x[][] to size[][]       */
    for (i = n-2; i >= 0; i--) {        /* process bottom-up       */
        size[i][n-1] = x[i][n-1];       /* copy i-th row and       */
        for (j = n-2; j >= 0; j--)      /* update size value       */
            size[i][j] = (x[i][j] == key) ?
                 1 + min(size[i+1][j],size[i][j+1],size[i+1][j+1]) :    0;
    }

    *width = 0;                         /* assume width is zero    */
    for (i = 0; i < n; i++)             /* scan size[] matrix for  */
        for (j = 0; j < n; j++)         /* the maximum and the     */
            if (size[i][j] > *width) {  /* cooresponding           */
                *width  = size[i][j];   /* locations.              */
                *row    = i;
                *column = j;
            }
}
```

【解答（解法二）】

第二个解法也可以做到与 n^2 成正比的比较次数，但比解法一要慢，并且在最坏的情况下会高达 n^3。正因为如此，只大略地讲解一下方法，细节方面请读者自己研读程序。

首先，把方阵查一次，如果 $M_{i,j}$ 与给定的值相同，就把 i 与 j 存起来，可以放到一个连接串行中，串行中每一个节点的内容就是 i 与 j。如果 $M_{i,j}$ 与给定的值不同，就不必管它了。同时，也准备另一个方阵 C，当值相同时，对应元素为 1，不相同时为 0。接下来就反复查这个连接串行，直到没有元素为止。当从头查起，而做到某一个节点时，若它的内容是 p

Download at http://www.pin5i.com/

与 q，那就表示 $M_{p,q}$ 的值与给定的值相同，就去查 $C_{p,q}$，$C_{p+1,q}$，$C_{p,q+1}$ 与 $C_{p+1,q+1}$，如果它们的值相同，比如说是 k，那么自 $M_{p,q}$ 起的最大子区域的容量就至少会多 1（为什么？），因此把 $C_{p,q}$ 加上 1，记下这一事实。但若上述 4 个值不相同，就表示 $M_{p,q}$ 起的最大子区域不能再扩充了，因此把 p 与 q 的节点从连接串行中删除。就这样一直反复，到最后一个节点都处理完了，就从头再查一次。

就以问题说明中的方阵为例，第一次查时，可以找出所有 2×2 的子区域，因此 C 方阵是：

```
1 2 1 0 0 0 0 0
0 1 2 2 1 0 0 0
0 0 1 2 2 2 1 0
0 0 0 2 2 2 2 1
0 2 2 2 1 1 2 1
0 1 2 2 1 0 1 1
0 0 2 2 1 0 0 0
0 0 1 1 1 1 0 0
```

再查第二次就有：

```
1 2 1 0 0 0 0 0
0 1 2 2 1 0 0 0
0 0 1 3 3 2 1 0
0 0 0 2 2 2 2 1
0 2 3 2 1 1 2 1
0 1 3 2 1 0 1 1
0 0 2 2 1 0 0 0
0 0 1 1 1 1 0 0
```

再查一次时，连接串行就不会有节点了。因此就知道最大的子区域是 3×3，有 3 的地方就有一个位置。

比较多少次呢？连接串行至少要查 k 次，如果最大子区域的容量是 k×k 的话，因为每查一次 C 方阵的值只会增加 1。由于矩阵又有 n^2 个元素，连接串行最多有 n^2 个节点（虽然在大多数情况下都不足此数，但是在分析“最坏情况”，所以做此假定），因此要有 kn^2 处理要求，在每一次处理要求中，要比较 4 个数是否相等，要 3 次比较，所以这个方法大约有 $3kn^2$ 次比较。在最坏情况下，如果给定的方阵元素几乎一样或全部一样，则 $k \approx n$，那么比较次数就接近 n^3 了，所以说这个方法比较差。

【习题】

（1）请证明在解法一中决定 $M_{i,j}$ 起最大子区域容量的说法。

（2）在解法一程序中，为了方便起见，在把容量求出之后，再用两个循环找出最大的容量，这是多此一举的，能否把这个部分直接放到求容量的循环？如何实现？

（3）在解法一程序中，为了方便起见，另外用一个 size[ ][ ]方阵记录容量，于是就多用了 n^2 个内存单位。然而，决定 $M_{i,j}$ 的容量时只用了 4 个元素，而且它们都会在两列上，

Download at http://www.pin5i.com/

因此似乎 size[ ][ ]只要两列足够了。请用这个观念改写程序，再加上上一题的改良，就会有一个真正漂亮的程序。

（4）请证明解法二中提到每查一次连接串行 C 中元素的值最多加 1 的说法。

（5）请证明解法二中用 $C_{p,q}$、$C_{p+1,q}$、$C_{p,q+1}$ 与 $C_{p+1,q+1}$ 四者来定出当时自 $M_{p,q}$ 起的最大子区域容量的说法。

【参考文献】

据笔者所知，这个题目是 L.Guibas 与 R.Lipton 在 1980 年提出来的，两年之后，贝尔实验室的 Kiem-Phong Vo 给出第一个解（解法二），至于更好的解法一是 3 年之后由纽约州立大学 Binghampton 校区的 D.F.Stout 做出来的。类似解法可以在其他书或文章中读到。如果依照 David Gries 的说法（1982 年），他是从 Prime（计算机）公司的 Ed Cohen 那里听到这个题目的，不过他发现在 J.Reynolds 的一本有关程序设计的书上就已经有了，并且指出这个问题的起源可以追溯到 W.H.J.Feijen。两个不同来源相差很大，或许是独立发现的也说不定，不过要提醒的是 Gries 在 1982 年（比 D.F.Stout 还早）就得出结果了，推荐也读一读 Gries 的文章。

1 D.Gries. A Note on a Standard Strategy for Developing Loop Invariants and Loops, *Science of Computer Programming*, 1982(2), pp.207～214

2 L.Guibas and R.Lipton. Problem 80-4, *Journal of Algorithms*, 1980(1), pp.108

3 J.Reynolds. *The Craft of Programming*, Prentice-Hall, 1981

4 D.F.Stout. Solution to Problem 80-4, *Journal of Algorithms*, 1983(4), pp.176

5 K.-P.Vo. Solution to Problem 80-4, *Journal of Algorithms*, 1982(3), pp.366

【程序】

程序 13-12（MAXSQR2.C）

```
/* ------------------------------------------------------------------ */
/* FUNCTION maxsquare:                                                */
/*    Given a square matrix and a key ,this function scans            */
/* the input matrix to find out the maximum size of a                 */
/* square sub-block such that all elements of this sub-               */
/* block are all the same.This is the second solution                 */
/* of this same problem.                                              */
/*                                                                    */
/* Copyright Ching-Kuang shene          july/03/1989                  */
/* ------------------------------------------------------------------ */

#include <stdio.h>
#include <stdio.h>

#define MAXSIZE  50          /* matrix size                           */
typedef int MATRIX[MAXSIZE][MAXSIZE];/* matrix type                   */
```

Download at http://www.pin5i.com/


```
typedef struct node * NODE_POINTER;      /* list node ptr        */
struct node {                            /* list node type       */
    int       row;
    int       column;
    NODE_POINTER next;
};
void maxsquare(MATRIX x,int n,int key,int *row,int*column,int *width)
{
    MATRIX        count;                 /* counting matrix      */
    NODE_POINTER list_head, last_node;   /* list ctrl var.       */
    NODE_POINTER p;
    int       i, j;

    list_head = last_node = NULL;        /* initial the list     */
    for (i = 0; i < n; i++)              /* and counting mat.    */
        for (j = 0; j < n; j++)
            if (x[i][j] == key) {        /* if eq to key than    */
                count[i][j] = 1;         /* count it and ...     */
                p = (struct nodr *) malloc(sizeof(struct node));
                p->row  = i;             /* put indices to a     */
                p->column = j;           /* node and put it to   */
                p->next  = NULL;         /* a linked list        */
                if (list_head == NULL)
                    list_head = last_node = p;
                else {
                    last_node->next = p;
                    last_node = p;
                }
            }
            else
                count[i][j] = 0;         /* !eq, set count=0     */

    while(list_head!=NULL){              /* the second part.the  */
        last_node   = NULL;              /* list is scanned      */
        p= list_head;
        while (p !  = NULL) {        /* look at three neighbors */
            if(count[p->row] [p->column] == count[p->row+1]
              [p->column]&& count[p->row] [p->column]==count[p->row]
              [p-> column+1]&& count[p->row][p->column]== count
              [p->row+1] [p->column+1]){
              count[p->row][p->column]++;  /*count it          */
              last_node = p;
```

Download at http://www.pin5i.com/


```
                }
                else
                    if (last_node == NULL)
                        list_head = p->next;
                    else
                        last_node->next = p->next;
                    p = p->next;
                }
            }

            *width = 0;                    /* extract the maximum          */
            for (i = 0; i < n; i++)
               for (j = 0; j < n; j++)
                   if (*width < count[i][j]) {
                       *width  = count[i][j];
                       *row    = i;
                       *column = j;
                   }
        }
```

Download at http://www.pin5i.com/

第14章 排 序

问题5.1 二分插入法（BINSERT.C）

插入式排列法（Insertion Sort）是一个极为简单的方法，但它的效率十分低，因为比较次数与 n^2 成正比，此处 n 是数据个数，而且还有大量移动数据的动作。请写一个程序，在这两点上作一番改良，使得比较次数与 $n\log_2 n$ 成正比，而且移动数据的速度加快。

【解答】

再度提醒，在一组已经排好的数据中找数据，或者是找位置，如果不用很特殊的数据结构，那么二分查找法（Binary Search）就一定可以用得上。所以，当处理 x[i]时，首先看看它是否比 x[0]小，如果是，它就要占据 x[0]所在的位置，而原来 x[0]～x[i-1]这 i 个元素就移到 x[1]～x[i]这一段，空出 x[0]的位置来放 x[i]得来的值。如果 x[i]比 x[i-1]大，x[i]就留在原位而不必有进一步的动作。但若 x[i]比 x[0]大，且比 x[i-1]小，x[i]就应该在 x[0]～x[i-1]之间的某处；可以从 x[0]～x[i-1]中间元素起，也就是 x[(0+(i-1))/2]，看它比 x[i]大还是小，如果比 x[i]大，那么 x[i]在它左边，但若比 x[i]小，x[i]就在它右边。继续这样二分的工作，$\log_2 i$ 次之后就可以找到一个 x[j]，使得 x[j]<x[i]<x[j+1]；于是，把 x[j+1]～x[i-1]的部分移到 x[j+2]～x[i]这一段去，而把 x[i]的值填入 x[j+1]，这就大功告成了。

对 x[i]而言一共作了多少次比较？x[0]～x[i-1]一共有 i 个元素，每次都去掉一半，$\log_2 i$ 次后就会只剩下一个；但每一次分割时都作了一次比较，因此最多 $\log_2 i+1$ 次比较而已。因为 $i=1,2,3,\cdots,n-1$，于是总比较次数就是：

$$
\begin{aligned}
&(\log_2 1+1)+(\log_2 2+1)+\cdots+(\log_2(n-1)+1)\\
&\quad=n-1+\sum_{i=1}^{n-1}\log_2 i\\
&\quad\approx n-1+(n-1)\log_2(n-1)-\frac{n-1}{\ln^2}\\
&\quad\approx n\log_2 n
\end{aligned}
$$

换句话说，总比较次数与 $n\log_2 n$ 成正比；上式中 $\ln x$ 是自然对数。

第一个问题解决了，看第二个问题。上面提过，当找出 x[i]的值要放在 x[j+1]，于是原来的 x[j+1]～x[i-1]这些元素就移到 x[j+2]～x[i]这一段去。当然，可以用一个 for 循环来写，但这正是导致插入法慢的一个原因；前面提示看<string.h>的函数，有没有发现一个叫做 memmove()的函数呢？它是这样写的（见 K&R 第二版附录 B3 节）：

```
void *memmove(void *dest const void *src size_t count);
```

Download at http://www.pin5i.com/

这表示从 src 指出的地方起，移动 count 个 char 到 dest 所指出的地方去，而且不会因为两者重叠而毁掉资料。一般而言，好的编译程序的链接库多半会用汇编语言来写这个函数，而不会用 C 语言写，因此在效率上就会比高级语言好，更重要的是，很多机器硬件都有一个移动一个区域（block move）的指令，有些可以移 100 多个数元组，有些可以移几十到几百字节，因此动作就更快了。所以，可以把 x[j+1]的地址给 src，把 x[j+2]的地址给 dest，把 x[j+1]～x[i-1]的数组个数给 count，于是使用 memmove()一次就可完成移动，效率是十分高的。

【程序说明】

这个程序在 BINSERT.C 中，把它与 SORTTEST.C 联用才能测试它，SORTTEST.C 产生一组均匀随机数，并且用 BINSERT.C 来（从小到大）排列；如果感觉不好，ORTTEST.C 换成自己的程序。程序中，size 是用来记录一个 int 数据有多少个 char 的，low、high 与 mid 分别是二分查找法的左边、右边与中间点；x 用来保存 input[current]（相当于解答中的 x[i]）的值，因为 memmove()会把它毁掉；pos 就是找出来要放 input[current]的地方，else if 的所在就是二分查找法，不过与一般的做法不大一样，不必比较是否相等，而是看 x 是否在 input[mid]右边，如果是，就取右半，否则取左半。其他的内容读者可以参看程序研习。

【习题】

（1）如果元素个数定为 4，请至少找出 3 组数据使传统的插入法进入最坏的情况，能否由第一题的结果导出一个一般性的结论？请试试看。

（2）请修改这个程序，用＜time.h＞中的 clock()函数计时，让二分插入法程序对 n=100,200,…,900,1000,2000,…,9000,10000 个元素操作，并且记录执行时间。令 n 个元素所用的时间为 t_n，请问 n 与 t_n 之间有什么关系，具有 $n\log_2 n$ 的关系吗？

（3）请修改程序，在每一次程序进行数据比较时，就用一个计数变量记录下来，使用上一题的各个 n 值。令 c_n 是 n 个元素所使用的比较次数，请问 n 与 c_n 的关系如何？具有 $n\log_2 n$ 的关系吗？

（4）请用问题说明中的传统写法，并且把此解答中的二分插入改成传统方式，但使用 memmove()。另外，修改本程序，使用二分插入，但不用 memmove()。这样，连同此解答在内，试用第（2）题中的各个 n 值，计算每一个程序所耗费的时间，于是就有了使用二分插入法，或 memmove()，以及两者并用所得来的改良。请问，在此提出的两种改良方式真的那么好吗？哪一个有比较显著的效应呢？

【参考文献】

插入法的起源已不可考证，此处的方法叫做二分插入法（Binary Insertion Sort），当然还有其他的制作方式，但这的办法最简单，有关一些比较结果与支节问题，请参看：

洗镜光.一个差不多被人遗忘的排序技巧－有效率地解决问题之（三），信息传真.Vol.8(1989), No.22, pp.133～147

Download at http://www.pin5i.com/

【程序】

程序 14-1（BINSERT.C）

```
/* ------------------------------------------------------------------ */
/* FUNCTION  sort :                                                   */
/*   An improvement of the insertion sort algorithm by                */
/* using binary search and memmove() function. This is the            */
/* fastest version that we can do without further any data            */
/* structure technique. Note that it attands order  of                */
/* O(n log n) in the comparison based model.  But this                */
/* model is not powerful enough so that this sorting                  */
/* algorithm is still very slow although it is of O(nlogn)            */
/*                                                                    */
/* Copyright Ching-Kuang Shene      June/27/1989                      */
/* ------------------------------------------------------------------ */

#include <string.h>

#define  YES 1
#define  NO  0

#define  SIZE    sizeof(int)/sizeof(char)

void  sort(int input[], int n)
{
     int current,pos;
     int low,high,mid;
     int x;

     for(current=1;current<n;current++={
          X  =input[current];
          pos = -1;      /*pos=-1 means DON'T move                    */
          if(x<input[0] /* insert before head?                        */
               pos = 0;  /* YES,set pos to 0.                          */
          else if(x<=input[current-1]={    /* bin search              */
               low = 0;  /* low index.                                 */
               high = current-1;            /* high index              */
               while(high-low>1){           /* stop?                   */
                    mid = (low+high)/2;     /* the mid pt              */
                    if(x>=input[mid])
                         low = mid;         /* try right               */
                    else
                         high = mid;        /* try left                */
               }
               pos = low+1;
```

Download at http://www.pin5i.com/


```
            }
            if(pos>=0){                          /* move a block        */
                memmove((void *)&input[pos+1],(void *)&input[pos],
                    SIZE*(current-pos));
                Input[pos] = x;
            }
        }
    }
```

问题 5.2　Shell 法（SHELL.C）

在传统的教科书中，Shell 排列法都是直接引用 D.L.Shell 当年原著中的办法，在要排序的数组中，先把间隔为 n/2 的元素排好，接着把间隔为 $n/2^2$ 的元素排好，再排间隔为 $n/2^3, n/2^4, \cdots, 4, 2, 1$ 的元素，最后就是一个依顺序排列完成的结果了。D.Knuth 曾经在他的著作中证明，Shell 方法要用与 $n^{1.5}$ 成正比的比较次数（当然已经比插入法等快了一些），不过最近 Sedgewick 等人却找出了更快的办法，他证明可以把 Shell 方法加快到只用到与 $n^{4/3}$ 成正比的比较次数。请写一个程序，找出一组更快的间隔。

【解答】

由于 $4^{j+1}+3\cdot 2^j+1$ 是需要排序的元素之间的间隔，因此它不能大于所有元素的个数 n；换言之，不管 j 是多少都能满足下式：

$$4^{j+1}+3\cdot 2^j+1\leqslant n$$

如果把 4^{j+1} 改写为 $4\cdot 4^j=4(2^j)^2$，上式会更清楚：

$$4(2^j)^2+3(2^j)+(1-n)\leqslant 0$$

把 $x=2^j$ 代入上式，有：

$$4x^2+3x+(1-n)\leqslant 0$$

上式的根有两个，亦即：

$$x=\frac{-3\pm\sqrt{16n-7}}{8}$$

但若取负号，x 的值就永远小于 0，因此 $x=2^j$ 就求不出 j 来了，所以上式取正号，亦即：

$$2^j=x=\frac{-3+\sqrt{16n-7}}{8}$$

因为 $4x^2+3x+(1-n)=0$ 是一个开口向上的抛物线，因此较大的根就是使式子小于或等于 0 的极大值，所以就可以解出 $4^{j+1}+3\cdot 2^j+1\leqslant n$ 的最大的 j 了：

$$j=\log_2\left[\frac{-3+\sqrt{16n-7}}{8}\right]$$

Download at http://www.pin5i.com/

在程序中，可以用上面求出的 j 算出一个 2^j，代入 $4(2^j)^2+3(2^j)+1$，以求得的值为间隔进行排列的工作，接着把 2^j 除以 2，再代入式子得到一个新间隔，进行排序，…，一直到间隔为 1 元素顺序排好了，整个排序过程就大功告成了。

看看程序 SHELL.C。把求 $4x^2+3x+1$ 的值写成一个集体指令 EQN()。程序中第一道表达式求出 j，暂时存放在变量 power2 中，接着用移位的方式来算 2^{j+1}，再放回 power2。为什么是 2^{j+1} 而不是 2^j 呢？原因就在于用 do…while 来写程序，测试 gap 的值的动作在最后，所以一进入循环就除以 2（用右移），来得到真正合适的 power2 值，再把这个 power2 值代入 EQN()就可以求出 gap 的值。至于其他部分则与传统教科书的写法一样，不会有什么问题。

【习题】

（1）请问程序在 n=1 时能够正常工作吗？为什么？如果不能，请修正它，如果可以，就请证明看法。

（2）间隔值用 $1,3,7,15,\cdots,2^j-1$，或是 $1,3,5,9,\cdots,2^j+1$，甚至于 Knuth 所提出的 $1,4,13,40,\cdots,(3^j-1)/2$，都可以得到比较次数与 $n^{1.5}$ 成正比的结果，请修改程序把这 3 种间隔值作一番比较。在比较时，令 $n=100,200,\cdots,900,1000,2000,\cdots,9000,10000$；它们的时间值、比较次数与其他方法相比，会有什么结果。

（3）Sedgewick 除了提出上面的问题值之外，还提到 $2\cdot4^j-9\cdot2^j+9$ 这个做法也可以得到 $n^{4/3}$ 的结果。请修改程序，把程序、这个新做法，以及第（2）题的结果作个比较。程序与这个习题的做法真的优于第（2）题的方式吗？

（4）从目前的 gap 值算出下一个 gap 值的方式有点琐碎，请化简并且加快它的速度。

（5）其实 Bubble（冒泡）法也可以用 Shell 的观点来改良。冒泡法每次都从头起比较相邻的两个元素，如果顺序不对，就把它们互换，一直到某次完全没有互换的动作发生为止，于是原数组就排列完成。如果用 Shell 的观点，在第一次时用足够大的间隔（比如 n/2），比较相距那么远的数；比较完后，下次用较小的间隔等，一直到最后一次的间隔为 1，这正是标准的冒泡法。请用这个观点来编写一个改良的冒泡法程序，并且将这个新程序与传统的冒泡法程序相互比较，也与 Shell 法作个比较，究竟这样的改良有多大的好处？

【参考文献】

Shell 方法是 D.L.Shell 在 1959 年提出来的，大多数教科书都直接采用他的写法。

1 D.L.Shell. A High-Speed Sorting Procedure. *Communications of ACM*, 1959(2), pp.30～32

在 D.Knuth 的著作（pp.84～95）中把早期的研究与他的发现作了一个总结，他指出使用 2^j-1、2^j+1 的间隔值分别是 Hibbard、Papernovn 与 Stasevich 的想法，而 $(3^j-1)/2$ 则是在他的书中首度出现。

2 D.E.Knuth. The Art of Computer Programming. Vol.3/Sorting and Searching, Addison-Wesley, 1973

Robert Sedgewick（D.Knuth 的学生）近几年的研究，又把 Knuth 的结果向前推进了一步，程序与习题（3）的结果就都取自他与他的学生的研究，请参看：

3 J.Incerpi and R.Sedgewick. Improved Upper Bounds on Shellsort, Journal of *Computer*

Download at http://www.pin5i.com/

and System Sciences. 1985(31), pp.210～224

4 R.Sedgewick. A New Upper Bound for Shellsort, *Journal of Algorithms*. 1986(1), pp.159～173

5 M.A.Weiss, R.Sedgewick, E.Hentschel and A.Pelin. Shellsort and The Frobenius Problem, Congressus Numerantium. 1988(65), pp.253～260

前面习题（5），用 Shell 方法的观点来改良传统的冒泡法，是 W.Dobosiewicz 在 1980 年提出来，他用的间隔是 n/2, 0.75(n/2), 0.75^2(n/2),…；不妨参看 Dobosiewicz 的文章。

6 W.Dobosiewicz. An Efficient Variation of Bubble Sort, *Information Processing Letters*. Vol.11(1980), pp.5～6

【程序】

程序 14-2（SHELL.C）

```
/* ------------------------------------------------------------------ */
/* FUNCTION  shell :                                                  */
/*   This is the Shell Sort program.  But the gap                     */
/* sequence used will affect the performance of the Shell             */
/* sort.  As proved by D.Knuth in his famous book, the                */
/* commonly used sequence, n/2, n/2^2, n/2^3, ..., 4. 2, 1           */
/* can achieve the order of O(n^1.5).  But Sedgewick found            */
/* a new gap sequence recently which can achieve the order            */
/* of O(n^(4/3)).  Thus it is a little bit faster than                */
/* the original version of D.L.Shell.                                 */
/*   The sequence used here is suggested by Sedgewick and             */
/* is defined as 4^(j+1) + 3*2^j + 1 = 1, 8, 23, 77, 281,             */
/* 1073, ...                                                          */
/*                                                                    */
/* Copyright Ching-Kuang Shene      July/30/1989                      */
/* ------------------------------------------------------------------ */

#include <math.h>    /* for sqrt()                                    */

#define  EQN(x)  ((4*x+3)*x+1)

void  sort(int x[], int n)
{
     int  gap;            /* the gap sequence variable                */
     int  power2;         /* represents 2^j                           */
     int  i, j, temp;     /* working variables                        */

     power2 = log((-3.0 + sqrt(16.0*n-7.0))/8.0)/log(2.0);
     power2 = 1 << (power2 + 1);
```

Download at http://www.pin5i.com/


```
    do {
        power2 >>= 1;
        gap    = EQN(power2);
        for (i = gap; i < n; i++)
            for (j=i-gap; j>=0 && x[j] > x[j+gap]; j -= gap) {
                temp = x[j];
                x[j] = x[j+gap];
                x[j+gap] = temp;
            }
        } while (gap > 1);
    }
```

问题 5.3 快速排列法（QSORT.C）

可以找一本书，研究一下快速排列法（Quick Sort）的观点，并且写成程序。

【解答】

在此将介绍 Nico Lomuto 求分界点的方法，这是笔者认为最好的求分界点的方法。在处理过程中随便挑选一个数，令为 a，这个数不一定是 x[ ]中的元素，并且让数组 x[ ]中的元素永远满足下面的关系，如图 14-1 所示。

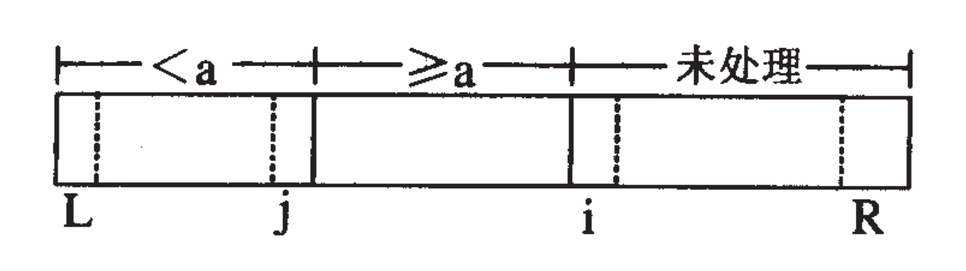


图 14-1

在图 14-1 中，L 与 R 是 x[ ]两个端点的元素的脚码，j 是小于 a 元素中最右边的脚码，第 j+1 个就大于或等于 a 了，i 是下一个待处理的元素的脚码，因此 x[j+1]与 x[i-1]之间的元素都大于或等于 a。

现在要处理 x[i]。x[i]与 a 的关系如何呢？应该只有下面的两种情况：

（1）x[i]≥a：因为 x[i]≥a，所以大于等于 a 的部分就可以往右延伸而涵盖住 x[i]，如图 14-2 所示。

（2）x[i]<a：这种情况下，x[i]这个值的位置就不对了，因为它小于 a，应该在 x[j+1]的左边才是。怎么办呢？有一个解法，x[j+1]≥a，如果把 x[i]与 x[j+1]互换，而把 j 与 i 都各往右移一格，那么关系还是不变。把 x[i]与 x[j+1]互换之后，x[i]的新值来自 x[j+1]旧值，它是大于等于 a 的，所以把 i 往右移一格，在 i 左边的元素还是大于等于 a。在 x[j+1]这一边，因为得到 x[i]的值，现在变成比 a 小了，所以把 j 也往右移一格，于是 j 左方的还是小于 a，但从 j+1 起还是大于等于 a。

因此，当 i 一直到达 R，连 x[R]也处理完之后，工作就结束了，得到的结果如图 14-3

Download at http://www.pin5i.com/

所示。

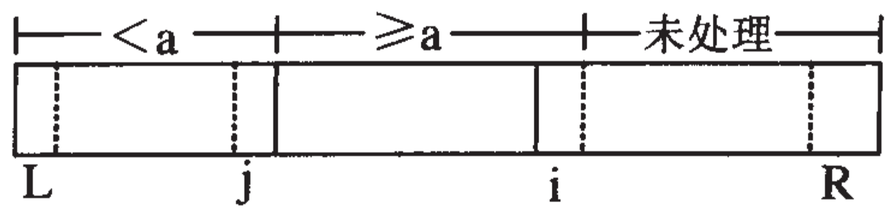


图 14-2

x[j]不就是一个分界点吗？不过，还留下一个问题，a 要如何选择呢？如果把 a 选得比 x[ ]中所有元素都大，就没有≥a 的部分；但若把 a 选得过小，小于 x[ ]中所有元素，就失去了<a 的部分，因此都不会有意义。一般的做法是把 a 选成 x[ ]中的元素，而且最好把 a 选得恰好可以把 x[ ]分成差不多大小的两半，最理想的就是把 a 选成 x[ ]的中位数（Median），因此有一半比 a 小，有一半比 a 大，但求中位数又是另一个难题。通常的做法是在 x[ ]中任选一个元素，比如说第一个元素 x[L]；这样，就不必处理 x[L]，而用上面的方法处理 x[L+1..R]即可，所以做完后的结果如图 14-4 所示。

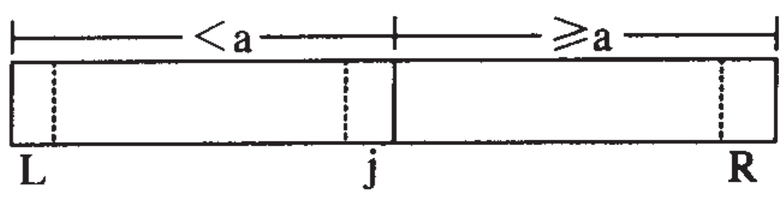


图 14-3

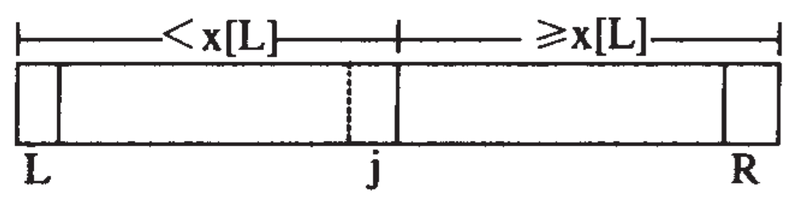


图 14-4

但是因为 x[L]的存在而破坏了在 j 左边（含 j）的元素都小于 x[L]的约定，解决方法是，把 x[L]与 x[j]对调。这样，x[j]还是一个分界点，在它左边的小于 x[j]，在它右边的大于等于 x[j]。

在程序开始时，因为 x[L]是看成 a，不必去处理它，因此 i 的值就应该是 L+1，L+2,…,R。至于 j 就应该是 L。看起来似乎违反了规定，其实并非如此，把 j 定成 L，在第一个被处理的元素（x[L+1]）之前，它的意义是“还没有发现小于 x[L]的元素”，只要一找到，就会往右移了。

下面的程序就是用这个观点编写的，不过使用了不同名称的变量，希望使程序可读性更高而已；程序中用 splitpoint 来表示 j。

【习题】

（1）快速排列法往往不是最好的方法，因为在最坏的情况下，它与最坏的方法相差不远。请问，什么样的输入数据，会使快速排列法有最多的比较次数？

（2）前面提过，如果把 a 选成是 x[L..R]诸元素的中位数（Median），那么就可以把 x[L..R]分成几乎相等的两半。那么为什么不用这个方法呢？请就程序员的观点评估这个做法的效率。

（3）有时候当排大小时，希望相等的元素在排好后还是以原来的顺序输出。请问上面的方法可以做到这一点吗？如果可以，请证明它。如果不可以，请修改程序使得变成可以。

【程序】

程序 14-3（QSORT.C）

```
/* ------------------------------------------------------------------ */
/* FUNCTION  quicksort :                                              */
```

Download at http://www.pin5i.com/


```
/*   An array of int and the number of elements are              */
/* handed to quicksort(), the control routine which              */
/* implements the Quick Sort algorithm.  It in turn              */
/* converts two arguments version to three arguments             */
/* and transfers control to qsort().  Then qsort()               */
   sorts these data and outputs the sorted array.                */
/*                                                               */
/*   qsort  — The real control routine of Quick Sort.            */
/*   split  — This routine splits the input array into           */
/*            two parts and returns the splitting index.         */
/*   swap   — swaps two given elements.                          */
/*                                                               */
/* Copyright Ching-Kuang Shene      June/22/1989                 */
/* ----------------------------------------------                */


/* ----------------------------------------------                */
/*                    function prototypes                        */
/* ------------------------------------------------------        */

void  qsort(int [], int, int);
void  split(int [], int, int, int *);
void  swap(int *, int *);


/* ----------------------------------------                      */
/* FUNCTION sort :                                               */
/*   This routine converts two arguments to three                */
/* arguments and calls the REAL control routine.                 */
/* ------------------------------------------------------        */

void sort(int input[], int n)
{
     int  first = 0;     /* the first element.                   */
     int  last  = n-1;   /* the last element.                    */
     qsort(input, first, last);
}


/* ----------------------------------------                      */
/* FUNCTION  qsort :                                             */
/*   This routine accepts an array and two indices               */
```

Download at http://www.pin5i.com/


```
/* indicating the first and the last element to be sorted.      */
/* Then calls split() to split the array into two parts         */
/* and sorts them recursively.                                  */
/* ------------------------------------------------------------ */

void  qsort(int input[], int first, int last)
{
     int  split_point;

     if (first < last= {
        split(input, first, last, &split_point);
        if (split_point - first < last - split_point= {
           qsort(input, first, split_point-1);
           qsort(input, split_point+1, last);
        }
        else {
           qsort(input, split_point+1, last);
           qsort(input, first, split_point-1);
        }
     }
}


/* ------------------------------------------------------------ */
/* FUNCTION split :                                             */
/*   This routine splits the given array into two parts         */
/* according to Nico Lomuto's aplitting technique and           */
/* returns the splitting index with 'split_point'.  Note        */
/* that all elements in x[first..split_point-1] are less        */
/* than or equal to x[split_point], while all elements in       */
/* x[split_point+1- last] are grater than or equal to           */
/* x[split_point].  Therefore after these two segments          */
/* are sorted, the entire array is sorted also.                 */
/* ------------------------------------------------------------ */

void  split(int input[], int first, int last, int *split_point)
{
     int  current_split,  unknown;
     int x;

     x = input[first];
     current_split = first;
     for (unknown = first+1; unknown <= last; unknown++)
```

Download at http://www.pin5i.com/


```
        if (input[unknown] < x= {
            (current_split)++;
            swap(&input[current_split], &input[unknown]);
        }
    swap(&input[first], &input[current_split]);
    *split_point = current_split;
}


/* ----------------------------------------------------------------- */
/* FUNCTION  swap :                                                  */
/*   This is the simplest routine which swaps two data.              */
/* ----------------------------------------------------------------- */

void  swap(int *p, int *q)
{
     int temp;

     temp = *p;
     *p  = *q;
     *q  = temp;
}
```

问题 5.4 保持等值的原来顺序（QSORT_L.C）

快速排列法（Quick Sort）有一个常被人批评的缺点，这就是相等的数据在排序完成后不会保持原来的顺序。不过，使用连接串行的（Linked List）技巧就可以轻易地克服它，请写一个程序完成这一件工作。

【解答】

为了方便起见，还是把这个函数写成 sort(int x[], int n)的形式，这样才能够与其他排序用的函数相匹配。正因为如此，在 sort()中就把 x[ ]数组中的各个元素转化成一个串行，再交给实际的函数处理。在此所建立出来的串行是由一个指向起头，一个指向结尾的指针，再加上一个连接串行构成的，如图 14-5 所示。

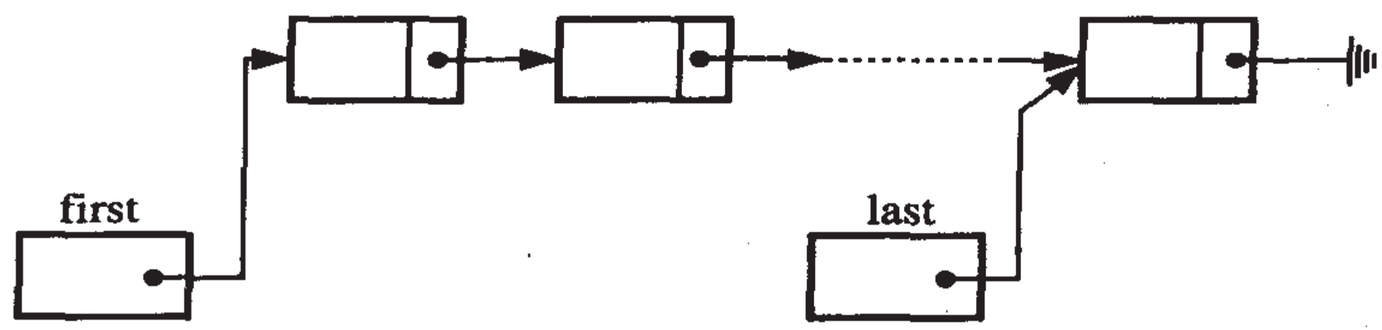


图 14-5

Download at http://www.pin5i.com/

sort()用到 insert()函数把一个节点插到串行的尾部。insert()的动作是这样的：先看看 first（指向开头的）指针是不是 NULL，如果是，这就是第一次的插入动作，于是就令 first 与 last 都指向交给它的节点；但若 first 不是 NULL，于是串行中就有元素，就把新来的元素插在最后面，把 last 所指向的元素的下一个（next 栏）定成新来的元素，而且也把 last 定到那。有一点很特别，insert()会返回新来的元素的 next 栏，也就是它的下一个元素的指针。如果新加入的元素是刚才通过 malloc()取得位置的，那自然不会有下一个，不过也没关系，在建立串行时用不着。

List_quicksort()是整个程序的核心。就像快速排列法中的 split()函数一样，它也用交给它的串行的第一个元素作基准，然后依序去查交给它的串行。如果元素小于基准的值，就串到“小于”的串行中；如果元素大于基准的值，则串到“大于”的串行；如果等于，就加到“等于”的串行中。注意，在开始之前就把基准值串到“等于”串行中。现在，应该可以看到 insert()返回指向交给它的元素的下一个元素的指针是很方便的。

当串行中的元素全都查完了，就有 3 个小串行，因此调用自己去排“小于”与“大于”这两者，最后通过 join()函数把 3 个串行重新接回来，并且通过参数把新串行的开头与结尾指针返回。

在程序中注意指针的用法，其中最重要的一点是：

```
(*first)->data
```

的写法（在 list_quicksort()中），如果省略了括号而变成：

```
*first->data
```

那么这个程序就是：

```
*(first->data)
```

即使用 first 所指向的 data，用它当指针来取出一个值了。另外，因为在程序中常常把一个指针值变成一个函数处理，而希望该函数修改它的值再返回，所以在调用时都加了取地址的运算（&），而且在被调用的函数中则都是指针的指针。以 Node*first 为例，first 是一个指针，它指向另一个指针，而该指针才是指向 struct node 这个结构。

【习题】

（1）为什么程序对于相等的值而言，都不会改变它们在输入中的次序。

（2）有时也会通过多加一个参与比较的字段，来达成不改变值相等的数据的顺序。例如，把 $x_i < x_j$ 定义如下：第一，若 $x_i \neq x_j$，那 $x_i < x_j$ 就是一般的意义；第二，若 $x_i = x_j$，$x_i < x_j$ 就是 $i < j$。请证明这个做法可以做到稳定的要求，不过在最坏的情况（请举一个例子）下，比较次数会加倍。

（3）请举一个例子，说明快速法并不稳定。

（4）请比较一下这个串行写法与原来写法的速度，两者相差多少？令 n=100, 200,…,900,1000,…,9000,10000，各作一次比较。

Download at http://www.pin5i.com/

【参考文献】

本题的观点最早是由 D.Motzkin 在 1981 年提出的，然后由 Lutz M.Wegner 作了进一步的分析，并且证明了这个方法所用的比较次数与 $(n+m)\log_2(n+m)$ 成正比，此处 n 是数据个数，m 是每一个数据的重复次数。上述的两篇文章如下：

1 D.Motzkin, A Stable Quicksort. Software: *Practice and Experience*. 1981(11), pp.607～611

2 L.M.Wegner. Sorting a Linked List with Equal Keys, *Information Processing Letters*, 1982(15), pp.205～208

【程序】

注意，本程序在测试时需要用到 SORTTEST.C。

程序 14-4（QSORT_L.C）

```
/* ------------------------------------------------------------ */
/* FUNCTION  quick sort (list version) :                        */
/*   Given an array of integers, this function sorts this       */
/* array with a variation of Quick Sort.  It transfoms          */
/* the array into a linked list, then sorts this linked         */
/* list with Quick Sort idea by partitioning the list           */
/* into three sublists, namely lists containing all             */
/* elements less than, equal to and greater than the            */
/* pivot.                                                       */
/*   This list version has an advantage in that all equal       */
/* keys are in their original order.  In other words, it        */
/* is stable.                                                   */
/*                                                              */
/* Copyright Ching-Kuang Shene      Jan/11/1990                 */
/* ------------------------------------------------------------ */

#include <stdlib.h>              /* for NULL and malloc()       */

typedef struct node *  Node;     /* pointer to a list node      */

struct node {                    /* list node type              */
     int   data;
     Node  next;
};

/* ------------------------------------------------------------ */
/*                      function prototypes                     */
/* ------------------------------------------------------------ */
```

Download at http://www.pin5i.com/


```
void  list_quicksort(Node *, Node *);
Node  insert(Node *, Node *, Node);
void  join(Node *, Node *, Node, Node);


void  sort(int x[], int n)
{
     Node first = NULL;
     Node last  = NULL;
     Node p, q;
     int  i;
     for (i = 0; i < n; i++) {   /* build up a linked list       */
         p = (Node) malloc(sizeof(struct node));
         p->data = x[i];
         (void) insert(&first, &last, p);
     }

     list_quicksort(&first, &last);          /* sort the list    */

     for (p = first, i = 0; i < n; i++) {    /* put back         */
         x[i] = p->data;
         q = p;
         p = p->next;
         free(q);
     }
}
/* ------------------------------------------------------------ */
/* FUNCTION  list_quicksort :                                   */
/*   This is the heart of the list version of Quick Sort.       */
/* It accepts two pointers pointing to the head and the         */
/* tail of a linked list.  Then sorts this list & returns       */
/* two new pointers pointing to the result list.                */
/* ------------------------------------------------------------ */

void  list_quicksort(Node *first, Node *last)
{
     Node less_first    = NULL;
     Node less_last     = NULL;
     Node equal_first   = NULL;
     Node equal_last    = NULL;
     Node greater_first  = NULL;
```

Download at http://www.pin5i.com/


```
    Node greater_last   = NULL;
    Node p;
    int  pivot;

    pivot = (*first)->data;          /* get the pivot              */
    p = insert(&equal_first, &equal_last, *first);
    while (p != NULL)                /* scan list...               */
        if (p->data < pivot)         /* if < pivot, move to less*/
              p = insert(&less_first, &less_last, p);
        else if (p->data > pivot)    /* if >, move to great        */
              p = insert(&greater_first, &greater_last, p);
        else                         /* if =, move to equal        */
              p = insert(&equal_first, &equal_last, p);

    if (less_first != NULL)          /* sort < and > lists         */
        list_quicksort(&less_first, &less_last);
    if (greater_first != NULL)
        list_quicksort(&greater_first, &greater_last);

    join(&less_first, &less_last, equal_first, equal_last);
    join(&less_first, &less_last, greater_first, greater_last);

    *first  = less_first;
    *last   = less_last;
}

/* ------------------------------------------------------------ */
/* FUNCTION insert :                                            */
/*   This function receives first and last pointers to a        */
/* list, and a pointer to a node, then insert that node         */
/* to the tail of the list.  Finally returns the next node      */
/* of the given node as function value.                         */
/* ------------------------------------------------------------ */

Node  insert(Node *first, Node *last, Node work)
{
    Node  p;

    if (*first == NULL)
        *first = *last  = work;
    else {
        (*last)->next   = work;
        *last   = work;
```

Download at http://www.pin5i.com/


```
        }
        p = work->next;
        work->next = NULL;
        return p;
    }

    /* ----------------------------------------------------------------- */
    /* FUNCTION join :                                                   */
    /*   This function accepts two set of pointers pointing              */
    /* to two linked list, then append the second list to the           */
    /* end of the first list. The two pointers will be                   */
    /* changed to point to the newly formed list.                        */
    /* ----------------------------------------------------------------- */
    void join(Node *first, Node *last, Node head, Node tail)
    {
        if (*first == NULL)
            *first = head, *last = tail;
        else if (head != NULL) {
            (*last)->next   = head;
            *last   = tail;
        }
    }
```

问题 5.5 非递归、无堆栈快速排列法（QSORT_I.C）

许多人都批评快速排列法，但观点却都集中在快速排列法使用了递归的方法。但事实上这并不是快速排列法的最严重缺点。无论如何，写一个不用递归的快速排列法也可以让不少人安心，请问能不能写一个不用递归，但也不用堆栈的快速排列法来排正整数？

【解答】

如果只是排正整数的话，这是一个十分简单的练习。在 split()函数（见快速排列法的说明）完成工作，而把数组的一部分切成两半时，那个分隔点就已经到达了既定的位置，所以排左半（或右半）时根本就可以不理会右半（或左半）的内容如何。因此想法是，一旦在 split()分割完成了，就把分隔点做上记号，亦即变号。所以，下一次再使用 split()时，若从某个位置起向右处理，一旦遇到一个负值就可以停止。原因就是上面说的，遇到了分隔点，而又是从左往右处理，因而穿越分隔点也就没有什么意义。

再用一个指针，指出目前已经排好了哪些值（当然，希望从左到右地排过去），如程序 14-5 所示。

【程序】

程序 14-5（QSORT_I.C）

```
for (sorted = 0; sorted < n; sorted++) {
```

Download at http://www.pin5i.com/


```
        处理 sorted 右边的内容,亦即用 split()求分隔点;
        分割点在 sorted 吗?
        如果不是,继续用 split()作分隔点;
        如果是,sorted 中就有了正确的值
    }
```

这就是本题的解，对于 x[sorted]而言，从 x[sorted]起向右，一直到遇到一个负值之前，用 split()求出分隔点，一旦求出分隔点后，就把它变成负值。接着看看分隔点是否正好就是 sorted，如果是那就太好了，x[sorted]就有了该有的值，因此就可以处理下一个，亦即 x[sorted+1]。但若分隔点不是 sorted，就反复这个过程，找出 sorted 到第一个负值之间的分隔点。这个动作就一直反复，因为每次求分隔点的工作都至少会有一个元素（亦即分隔点本身）被剔除，所以从 sorted 起到第一个负值的距离就会越来越短，最终必定造成 sorted 就是分隔点。当 sorted 就是分隔点时，还得把 x[sorted]的值变号，因为在找分隔点时会把在分隔点的值变号，运行方式如图 14-6 所示。

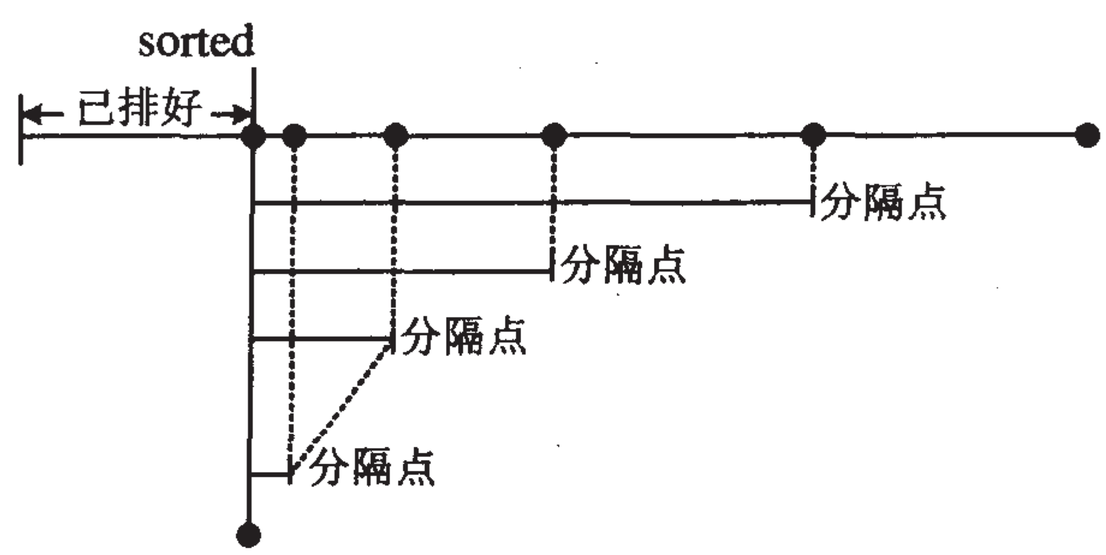


图 14-6

图 14-6 中的分隔点越来越接近 sorted 的位置，最后合二为一。在处理 x[sorted+1]时，如果它的值为负，就表示在此之前它曾经是分隔点，已经有了正确的数，因此把它变号即可。但若 x[sorted+1]为正，就从 sorted+1 起到下一个分隔点之间继续重复求分隔点的工作。

在程序中为了方便起见，在 x[n]，亦即最后一个元素后面放一个负值，使它成为分隔点，所以在一开始时找分隔点的工作就到 x[n]结束。正因为如此，交给这个函数使用的 x[ ]数组就要比原来的数据数目多 1。

观点就是如此，相信并不难了解。排正整数时没有用到堆栈，但排一般的数据呢？请看下面的进一步讨论。

【研讨】

这个程序似乎没有堆栈，但是事实上堆栈就隐含在被变号了的位置中，因为分隔点是被变号了的，所以相邻的两个负值所在的位置就正好是快速排列法的堆栈内容。看个例子就不难明白，为了方便起见，假设分隔点都正好在正中央，计划排 15 个数据，x[0]～x[14]。如果用递归的方式来做，就由 split()所产生的分割方式在左，而各分割的在右，它们都是个完整的树状结构，如图 14-7 所示。

Download at http://www.pin5i.com/

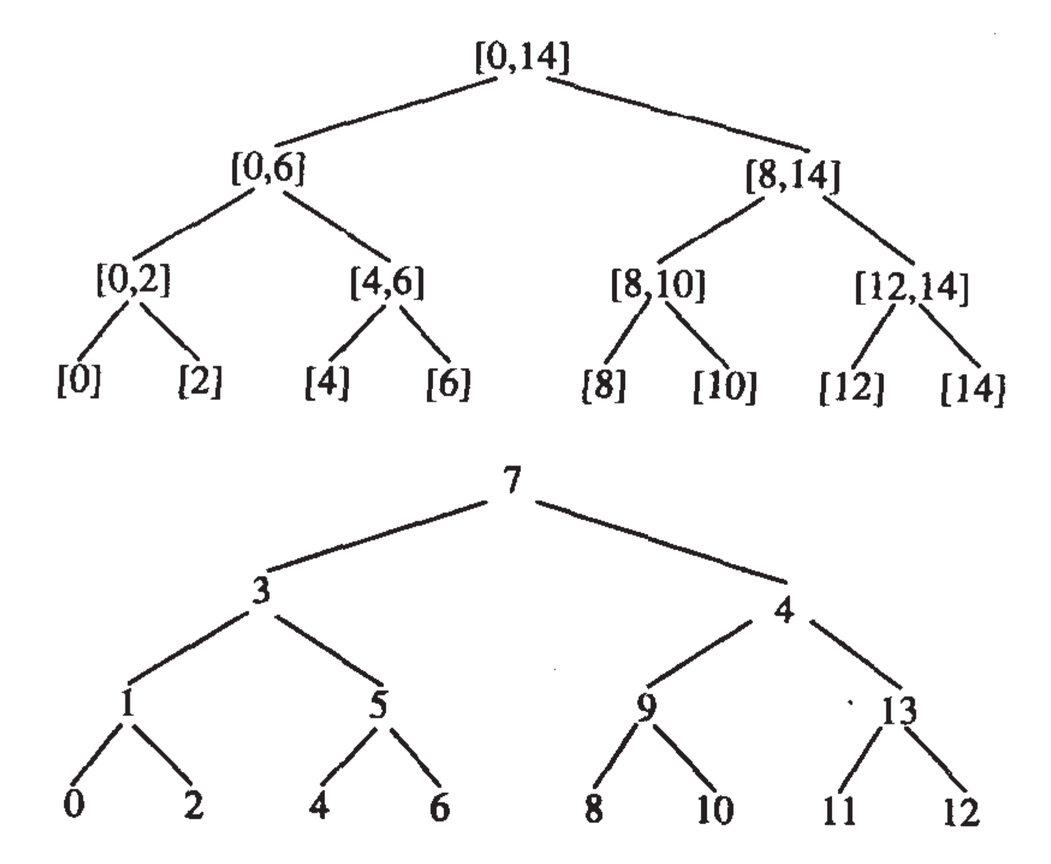


图 14-7

当用[0,14]调用 qsort()时，split()找出 7 是分隔点，qsort()用[0,6]调用自己；qsort()给出分隔点 3，qsort()又用[0,2]调用自己；然后 split()又找出分隔点 1，qsort()用[0]调用自己，x[0]就排好了。此时调用的堆栈中从底向上算是[0,6]、[0,2]、[0]，只要把每一个区间右边的值加 1，就正好是分隔点的值。在上面的程序中，为了要排 x[0]，使用 split()找[0,14]的分隔点，得到 7，再用 split()自 x[0]起找分隔点时到 x[7]就会停止，因为 x[7]已经被变号，就等于是求[0,6]的分隔点，所以分隔点是 3。再来求[0,2]的分隔点，是 1。接着求[0]的分隔点，正好是自己，因此 x[0]就排出来了。看看一连串做上的记号不正是分隔点进入堆栈的顺序吗？

其实，如果仔细些，就会看到这个方法只不过是把上面的递归树状结构用内序走一次而已，这也正好是递归调用时的顺序。所以程序还是有个堆栈，只是不明显地表示出来而已，当要排的不只是正整数时，那就是记下记号，亦即分隔点的位置，那不正好就吻合了前述的堆栈状态吗？

【习题】

（1）请修改程序，维持它的结构，但却可以排任意整数。

（2）以上讲述了如何完善快速排列法，而能够保持值相同的元素在输入中的顺序，除本题方法外还有另外的几个变换形式。最常见到的变换方式，第一就是在 split()之后先排数据较少的那一段，前面的 QSORT.C 就是这样写的，这样可以避免堆栈中有过多的元素。第二，不用最左边元素作基准，而是采用最左、最右与中央 3 个元素的平均值，这就不容易造成基准元素值是那一个段落中的最小或最大值，于是在下次分割时与插入法相同。请用这个方法来写程序，并且与 QSORT.C 的速度进行比较。

（3）另一个常见的变换，就是当分割出来的段落元素太少时，用其他方法来排。原因是元素不多，使用递归与 split()不见得好，不如用其他简单方法，比如插入法更好，但究竟一个段落要少到多少才算少？在此做个实验。先把快速排列法与插入法合并，但把切换用的长度值定成一个变量。当 n=10000 时（假设），把切换的长度用 2,3,4,…,20 这几个值都试一次，看哪一个值的时间最短，把它找出来。

Download at http://www.pin5i.com/

（4）把上面习题找出的值固定，然后用 n=100,200,…,1000,…,9000,10000 来测试程序，请问它比 QSORT.C 以及其他的方法快吗？会快多少？

（5）如果在切换时，不用插入法而改用堆栈法（Heap Sort），那么切换的长度应该是多少才最省时，用这个方法会比第（4）题快吗？快多少？

【参考文献】

程序用的方法是取自 Huang Bing-Chao 与 Donald E.Knuth 在 1986 年合写的文章，不过他们采用的分割的方式不是很理想，因此程序中用的是 Lomuto 的方法（见 QSORT.C 的说明）。

1 B.-C, Huang and D.E.Knuth. A One-Way, Stackless Quicksort Algoithm, *BIT*. 1986(26), pp.127～130

习题（2）、（3）、（4）的做法在 Sedgewick 的博士论文（1975 年）与他的几篇文章中都有极其详细的分析与讨论。

2 R.Sedgewick. The Analysis of Quicksort Programs, *Acta Informatica*. 1977(7), pp.327～355

3 R.Sedgewick. Quicksort with Equal Keys, *SIAM Journal on Computing*. 1977(6), pp.240～267

4 R.Sedgewick. Implementing Quicksort Programs, *Communications of ACM*. 1978(21), pp.847～857

5 R.Sedgewick. *Quicksort*, Garland Publishing Company. New York: 1980

上面第 4 份文献就是 Sedgewick 的博士论文，不过是在 1980 年以书籍形式出版。

习题（5）的想法则是取自：

6 J.Röhrich. A Hybrid Quicksort with O(n log n) Worst Case Complexity, *Information Processing Letters*. 1982(14), pp.119～123

【程序】

程序 14-6（QSORT_I.C）

```
/* ------------------------------------------------------------ */
/* FUNCTION  sort :                                             */
/*   Here is a beautiful little program from Huang              */
/* Bing-Chao and Donald E. Knuth.  Actually this is an          */
/* iterative implementation of Hoare's Quicksort in a           */
/* one-way, stackless fashion.  A restriction is that it        */
/* sorts positive integers only.  Can you find any              */
/* meaningful way to extend this method ?  One more thing,      */
/* this program requires one more element in your array.        */
/* This means that if you are going to sort n integers,         */
/* you have to provide n+1 locations.                           */
/*                                                              */
/* Copyright Ching-Kuang Shene      Jan/11/1990                 */
/* ------------------------------------------------------------ */
```

Download at http://www.pin5i.com/


```
#define  YES 1
#define  NO  0
#define  ALWAYS  1

#define  SWAP(x, y)    { int t;  t=x; x=y; y=t; }

void  sort(int x[], int n)
{
     int  sorted;          /* next will be sorted                    */
     int  split;           /* split element as usual                 */
     int  next;            /* next element pointer                   */
     int  key;

     x[n] = -1;            /* a sentinel at the end                  */
     for (sorted = 0; sorted < n; sorted++) {   /* sort all          */
        while (x[sorted] > 0) { /* if it is !sorted:                 */
           key   = x[sorted];    /* start Lomuto's                    */
           split = sorted;       /* split procedure                   */
           for (next = sorted+1; x[next] > 0; next++)
              if (x[next] < key) {
                 split++;
                 SWAP(x[split], x[next]);
              }
           SWAP(x[sorted], x[split]);
           x[split] = -x[split];
        }            /* set the split sorted                         */
        x[sorted] = -x[sorted]; /* switch back                        */
     }
}
```

问题 5.6　求中位数（MEDIAN1.C）

一组数的中位数（Median），就是把那一组数从小到大排好后位居中间的那一个；如果有奇数个，那么在中间的那一个是存在的，但若有偶数个，就没有中间的那一个数了，因此就取位于中间那两个数的平均数。例如，3,1,7,5,9 经过排列后是 1,3,5,7,9，所以中位数是 5；但 3,1,7,5,9,4，经过排列得到 1,3,4,5,7,9，所以中位数就是(4+5)/2=4，为了方便用整数运算。请写一个程序，接收一个整数数组，不必排大小而找出该数组的中位数。

Download at http://www.pin5i.com/

【解答】

回想一下 split()程序的工作，它接收 x[L..R]，求出一个分界点，在它左边的小于等于它，在它右边的大于等于它，如图 14-8 所示。

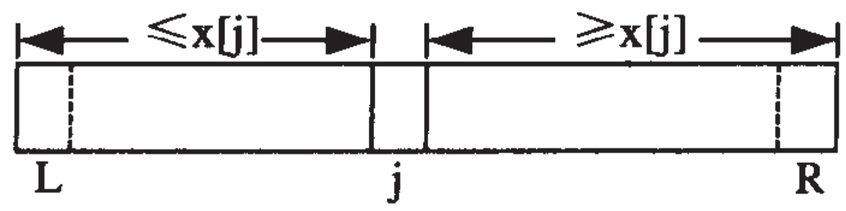


图 14-8

如果在中央的元素是 x[M]，j=M，那么小于等于 x[M]、大于等于 x[M]的不正好是一半吗？但若 j<M，这就表示中位数一定在 x[j+1..R]这一段，因此比 x[j]小或相等的元素不到一半。反过来，若 j>M，则表示比 x[j]大或相等的元素不到一半，因而中位数在 x[L..j-1]这一段。所以，只要不断地做 split()的工作，直到分界点是 L 与 R 中央的元素就行了。注意，还要考虑到有奇数个或是偶数个元素的问题。当反复执行到分界点正好是中点时，若有奇数个元素，这就是答案；但若有偶数个，还得取出相邻的元素求平均数才行。

【习题】

（1）请用手算仿真 3,1,7,4,5,2，以及 1,7,9,2,3 这两组数据的执行情况。前者中位数是 $(3+4)/2=3$；后者为 3。

（2）请问这个程序在最坏的情况下会进行多少次比较，什么样的数据才会出现最坏的情形？

（3）请修改这个程序，使得可以接收一个 k 值，找出在数组中第 k 小的元素；换言之，在 3,1,7,4,5,2 中，最小的是 1，次小是 2，第三小是 3 等，但是不能做排大小的工作。请问，在这个程序中，对于任何 k 而言，效率会比求中位数低吗？一般而言，求 n 个数中最大的那一个，比较 n-1 次就够了。能用这个程序求最大的值吗？求次大（或次小）的又如何呢？这是值得思考的问题。

（4）如果运气很好，在找第 k 小的元素时，每执行一次 split()就去掉一半的元素，请问做一次 split()要多少次比较？在上述情况下，求出第 k 小的元素时有多少次比较？（提示：大约是 2n 次）

【程序】

程序 14-7（MEDIAN1.C）

```
/* ------------------------------------------------------------ */
/* FUNCTION median :                                            */
/*   Given an integer array, this function computes the         */
/* median of this array without sorting.  Note that the         */
/* median of an array of odd number of elements is the          */
/* middle element in sorted order.  But for an array with       */
/* even number of elements, the median is the average of        */
/* the two elements in the middle of the array.  For            */
```

Download at http://www.pin5i.com/


```
/* example, the median of {1,3,5,7,9} is 5, while the          */
/* median of {1, 3, 5, 7, 9, 11} is (5+7)/2=6.                 */
/*                                                             */
/* Copyright Ching-Kuang Shene      July/22/1989               */
/* ---------------------------------------------------         */

#define  ALWAYS   1

void  split(int [], int, int, int *);

int  median(int x[], int n)
{
     int  left  = 0;
     int  right = n - 1;
     int  mid   = (left + right)/2;
     int  split_point;

     while (ALWAYS) {              /* loop until we hit the mid   */
        split(x, left, right, &split_point);   /* split          */
        if (split_point == mid) /* is the spli pt. mid?          */
           break;                 /* YES, found                  */
        else if (split_point > mid)     /* NO, larger than        */
           right = split_point - 1;     /* use left part          */
        else                            /* NO, smaller than       */
           left  = split_point + 1;     /* use right part         */
     }
     return (n & 0x01 != 0) ? x[mid] : (x[mid]+x[mid+1])/2;
}


/* ---------------------------------------------------          */
/* FUNCTION split :                                             */
/*   Given a section of an array, this function splits          */
/* this array in to two parts such that the all elements        */
/* of left part are less than some element while all            */
/* elements of the right part are greater than or equal to      */
/* the same element.  See quick sort for more detailed          */
/* information.                                                 */
/* ---------------------------------------------------          */

#define  SWAP(x, y)  { int  t;  t = x; x = y; y = t; }
```

Download at http://www.pin5i.com/

```
void split(int x[], int left, int right, int *split_point)
{
    int split_data = x[left];
    int i;

    for (*split_point=left, i=left+1; i <= right; i++)
        if (x[i] < split_data) {
            (*split_point)++;
            SWAP(x[*split_point], x[i]);
        }
    SWAP(x[left], x[*split_point]);
}
```

问题 5.7 堆积法（HEAPSORT.C）

请写一个堆积排列法（Heap Sort）的程序，可以参考任何手头上的书籍，但注意一点，即程序必须写得有效率。

【解答】

如果把堆积顶点放到数组的第一个位置——x[1]中（注意，不是 x[0]），而依次地将其左、右两个后代放在 x[2]与 x[3]中，接着从上向下、从左向右把各个节点放到数组中，会得到如图 14-9 所示的结果。

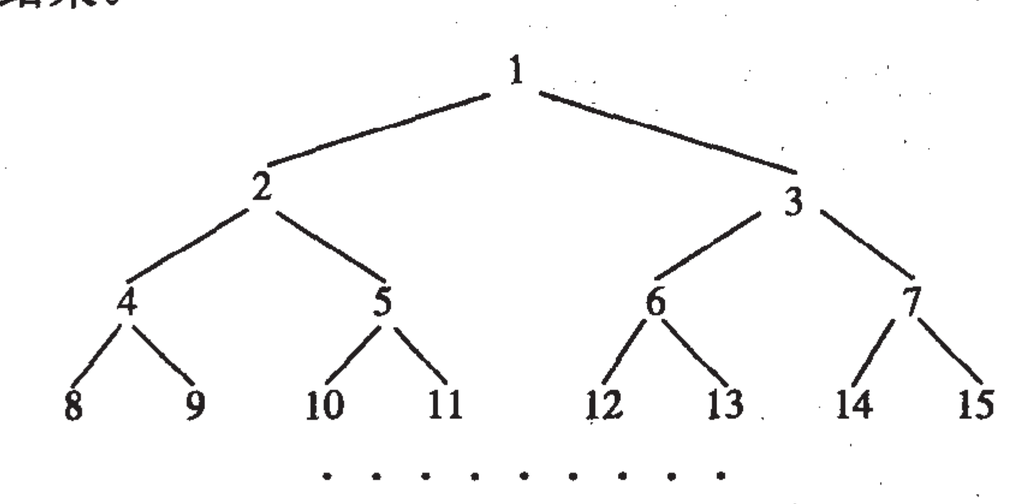


图 14-9

于是很明显地，对于 x[father]这个元素而言，它的两个后代（如果有的话）就是 x[2*father]与 x[2*father+1]。如果 x[father]以下的节点都已经是堆积，只有 x[father]可能不是，依照问题说明，找出 x[2*father]与 x[2*father+1]中较大的那一个，令为 x[son]，再将 x[father]与 x[son]相比，于是会有两种情形：

（1）x[father]≥x[son]，x[father]所在就是一个堆积了。

（2）x[father]<x[son]，x[father]所在就不是堆积。将 x[father]与 x[son]互换，原 x[father]所在就有了堆积的条件，但是因为原来 son 所在得到一个较小的值，所以可能破坏了堆积的条件。没有关系，令 father 等于 son，相当于把 father 移到下一层，反复上述的操作就行

Download at http://www.pin5i.com/

了。最后如果不是第一个情况出现，就会到达叶部，两个都是堆积。

在 HEAPSORT.C 这个程序中，fix_heap()做的就是这件事，它收到储存数据的数组 x[ ]；被破坏了的顶点所在 root，亦即 x[root]以下是堆积，但 x[root]可能不是。key 是应该在 x[root]，但却可能会破坏堆积特性的值；最后，bound 指出目前要处理的元素不能超过 x[bound]。在程序中有个技巧，因为每次都把 x[father]与 x[son]互换实在很浪费时间，所以不这样做。注意，key 是新来的值，如果把它存入 x[root]就可能破坏了堆积，所以在最开始时，father 是 root，而 son 为 father+father。while 循环就看看后代是否超越了范围，如果没有就看看 x[son+1]大还是 x[son]大，把大的那个位置定成 son。如果 key 的值（也就是 x[root]的值）比 x[son]的值小，在 x[father]处就不是堆积，把 x[son]放到 x[father]处，那不就毁了 x[father]的值了吗？没有，因为值在 key 中。接着向下移一层（把 father 定成 son），一直到 key（亦即原 x[root]）的值大于等于 x[son]为止，因此 key 就得放在 x[father]处，这就是个堆积了。

HEAPSORT.C 中主控的部分还是 sort()函数。因为 C 的脚码从 0 起，又要用到 father 的后代是 2*father 与 2*father+1，所以程序要从 1 起才行，这不就与 C 的规定矛盾了吗？程序的参数是 old_x[]，在开始执行之前就把 x[1]定为 old_x[0]，因此当处理 x[i]时，事实上是在处理 old_x[i-1]，这就符合要求了。接着 sort()从第 n/2 个元素起，它正好是最下层最右边的元素 x[n]的 father，依次用 fix_heap()把 x[n/2],x[n/2-1],…,x[2],x[1]起的元素排成堆积，这个顺序正好就是从右向左、自下而上最终到达顶点的顺序。第一个 for 循环的工作就是做这件事。第二个 for 循环的工作就是排大小，它从数组的尾部向前处理，也就相当于把树状结构从下向上，并且从右向左地处理；每一次它都把堆积顶上的元素取出来，当然这就是目前最大的元素，然后把目前在堆积最下方、最右边的元素（x[size]）取出来，要求 fix_heap()把它从顶点往下推到适当的位置去，但是给 fix_heap()的“最后一个位置”却少了 1，因此传来在“最后一个位置”的元素就会给了 x[1]，而被推到另一个所在，而 x[1]则不见了。之后，把第一个元素的值放到 x[size]，也就是目前的最后一个位置中，于是目前最大的元素就到达它应该出现的位置了。一直往前做，当 x[2]也处理完时，数组的顺序也就排好了。

【习题】

（1）算一下堆积排序会用多少次比较。为了方便起，假设一共有 2^k-1 个元素，这样就只有最底层的元素没有后代。如果堆积（有 2^k-1 个元素）中只有第一个，也就是最顶上的元素不满足堆积的条件，其他元素都满足条件，那么把最顶上元素推到适当的位置的动作最多要作多少次比较？

提示：最多的情况是推到最底层，那么一共推了多少次？每推一次有多少次比较？

（2）在建立堆积时，从下向上、从右向左地处理，因此倒数第二层的每一个节点（包括自己在内）以下有 3 个元素，倒数第三层的节点（连自己在内）以下有 7 个元素；一般而言，倒数第 i 层的元素以下（包括自己在内）会有 2^i-1 个元素。请证明这一个命题。

（3）请问，倒数第二层,倒数第三层,…各有多少个元素？因此 fix_heap()在处理倒数第二层,倒数第三层,…的顶点时用了多少次比较呢？

Download at http://www.pin5i.com/

提示：使用第（2）题的结果。

（4）堆积排列法的最后，是把顶点毁掉来排大小的，请问这又进行了多少次比较？合并本题与 5.4 题就可以得到结果，最终的答案应该是与 $n\log_2 n$ 成正比的比较次数。

【程序】

程序 14-8（HEAPSORT.C）

```
/* ------------------------------------------------------------ */
/* FUNCTION heapsort :                                          */
/*   Given an int array and the number of items in this         */
/* array, this routine sorts the array with heap sort.          */
/* A very important note is that due to special subscript       */
/* relationship of a heap, the given array must have one        */
/* more element because heapsort assume the 1st element         */
/* begins with x[1].  In what follows, I have choosen to        */
/* use a trick to fool the C compiler.  heapsort() will         */
/* receive an int pointer old_x pointing to an integer          */
/* array.  But in this function, we define a new point x[]      */
/* which contains the address of old_x[] minus one.  Thus       */
/* x[1] is old_x[0] and therefore the problem is resolved.      */
/*   This is one of the very critical point that I HATE         */
/* programming in C.  All GOOD language, even FORTRAN,          */
/* have some special declaration feature to shift the           */
/* origin of an array to somewhere, for example declaring       */
/* an array in the range of x[5] to x[10] just like in          */
/* the old friend Pascal.  Don't long live C.                   */
/*                                                              */
/* Copyright Ching-Kuang Shene      Mar/22/1990                 */
/* ------------------------------------------------------------ */

#define  YES 1
#define  NO  0

void  fix_heap(int [], int, int, int);

/* ------------------------------------------------------------ */
/* FUNCTION sort :                                              */
/*   Given an array x[] and its number of elements, this        */
/* function uses fix_heap() to build up a heap and then         */
/* sorts the array by swaping the top and the last items.       */
/* ------------------------------------------------------------ */

void  sort(int old_x[], int n)
```

Download at http://www.pin5i.com/


```
{
     int  *x = old_x - 1;            /* fool the C compiler      */
     int  temp;
     int  size, i;

     for (i = n/2; i >= 1; i--)          /* starts off from mid */
        fix_heap(x, i, x[i], n);         /* fix the heap at this*/

     for (size = n; size >=2; size--){  /* sort phase.          */
        temp = x[1];                     /* pick up the largest */
        fix_heap(x, 1, x[size], size-1);/* move to last         */
        x[size] = temp;                  /* fix and restore      */
     }
}


/* ----------------------------------------------------          */
/* FUNCTION fix_heap :                                           */
/*   Given a heap rooted at 'root', a new key 'key' and          */
/* the subscript of the last item 'bound', this function         */
/* inserts the new key and restore the heap properties.          */
/* ------------------------------------------------------------  */

void  fix_heap(int x[], int root, int key, int bound)
{
     int  father, son;
     int  done;

     father = root;              /* start off from the root      */
     son    = father + father;   /* get the subscript of son     */
     done   = NO;
     while (son <= bound && !done) {    /* son available?        */
        if (son < bound && x[son+1] > x[son])   /* latger?       */
           son++;                /* now son is the larger one    */
        if (key < x[son]) {      /* if the son > key             */
           x[father] = x[son];   /* step down until done         */
           father    = son;
        }
        else                     /* son < key, thus the key      */
           done      = YES;      /* get its position.            */
        son = father + father;
     }
     x[father] = key;            /* insert the key               */
}
```

Download at http://www.pin5i.com/

问题 5.8 改良的堆积法（HEAP_NEW.C）

在用堆积法排顺序时会有一个很明显的缺点，许多优秀的程序员可能都会看出来。如果现在要决定 x[i]是不是应该在目前的位置，要找出在堆积中 x[i]的两个后代，x[2•1]与 x[2•i+1]，再用这两者中大的一个（于是用了一次比较）与 x[i]相比（第二个比较），如果 x[i]比较大，问题就解决了；但若 x[i]比较小，就把 x[2•1]与 x[2•i+1]中大的那一个与 x[i]互换，再去处理在新位置的 x[i]值。很明显地，把一个值每往下移一层就要两次比较，是不是太多了一些？请发展一项技巧来克服这一个短处。

【解答】

在堆积中，因为每一个点的值都比它的两个后代大，因此从堆积的顶点起到叶部的路径上，沿途走过的点的值是已经排好的，不但如此按照还是从大到小的顺序排列的。看看问题说明中的图就不难明白，在图 5-14 中，当把 50 放到合适的位置之后，从顶上起到 50 的路径会经过 98、80、70、55、50，正是从大到小地排列的。

因为在堆积排序的方法中，要插入的元素一定与某个点的较大后代相比较，因此要找出从顶上起往下走时，那些个较大的后代所在。用问题说明的图 5-14 来看，顶点（不计在内）的较大元素是 98（指在插入 50 之前），98 的较大后代是 80，80 的较大后代是 70，70 的较大后代是 50，因此从顶点起所经过的各个较大的后代用粗线表示，这正是逐次比较所走的路径，虽然并不见得每一个都用到。

要找出较大后代并不难，就说现在是在 x[son]这个元素，它的两个后代就是 x[son • 2]与 x[2•son+1] 把这两个比较大的那一个的脚码找出来，并且存入 son，因此一直反复这个步骤，son 所经过的就是从顶点到叶部的较大元素；以问题说明的图来看（插入 50 之前），这条路就是 x[2]、x[5]、x[11]、x[22]。也许会想，为什么走到叶部就停下来，而不用一个数组把沿途的脚码记起来呢？想法是对的，但如果了解了堆积的特性，就会打消这个念头。因为 x[i]的后代是 x[2•i]与 x[2•i+1]，所以反过来，如果知道 x[j]而想找它的上一代，不就是 x[j/2]吗？原因是 j 不外是 2•k 或 2•k+1，所以把 j 用 2 去除就得出 k 值了。从顶点起一直沿较大的后代走到叶部，并且记住叶部的脚码 son，于是要反过来找路径时（其实并不需要），把 son 用 2 去除就可以找回上一代。

这个找出顶点沿较大后代走到叶部去的动作一共作了多少次比较？n 个元素所构成的堆积一般而言会有$\lceil \log_2 n \rceil$层，$\lceil x \rceil$表示比 x 大的整数中最小的一个（$\lceil 3.14 \rceil = 4$，$\lceil 5.79 \rceil = 6$，但是并不处理根部，所以只走了$\lceil \log_2 n \rceil - 1$层而已，不妨就用$\log_2 n$（用$\lceil \log_2 n \rceil$才正确）。因此从顶点到叶部，沿较大后代的路径需要$\log_2 n$次比较。

有了这一条（隐含的）路径之后，要如何工作呢？回想一下堆积排序法，必然会发现 x[1]的值（注意，是从 x[1]起，而不是 x[0]起，参考 HEAPSORT.C 的讨论）除了用来存放极大值之外，根本没有用处。因为一旦把一个极大值推到顶上之后，就会把顶点的元素去掉，换到最后面，而把最后面的元素再度推回堆积。

Download at http://www.pin5i.com/

于是当要把一个值放回堆积时，要从顶点起与沿途较大的后代比较，但前面已经提过这条路线，而且路径上的元素又是从大到小排列的，如图 14-10 所示。

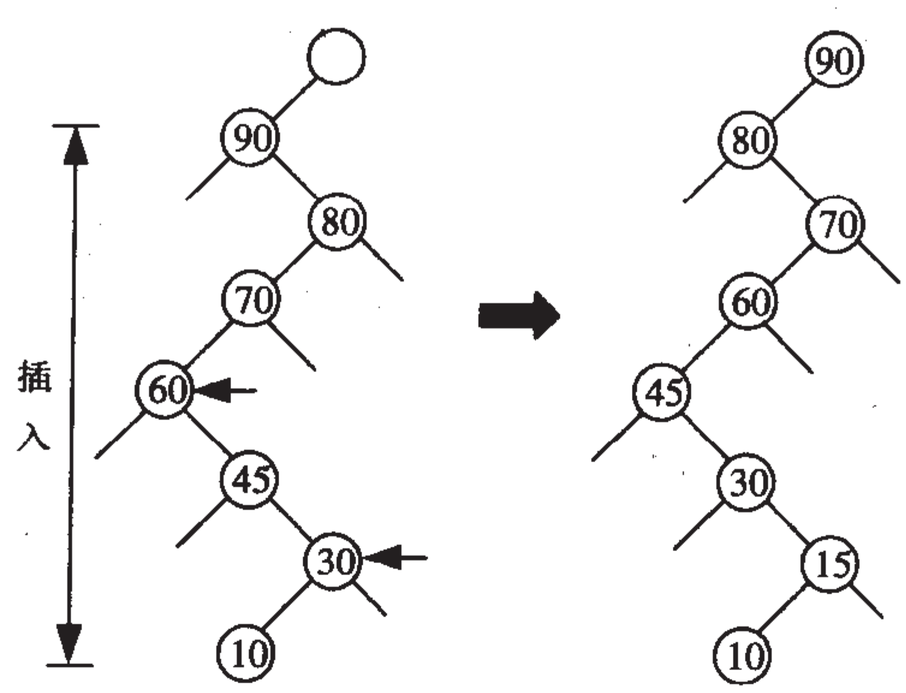


图 14-10

用二分法就可以找出 15 的座落位置。拿 15 与中间元素（60）比较，15 较小，所以范围收缩为 60～10 之间，中间元素是 30；15 又比 30 小，范围就收缩成只有 10，因此 15 就应该在 10 上方、30 下方，然后把 30 以上的元素（含 30）往上挤一层，空出 30 的所在地，把 15 放进去就行了。

那么如何做二分法呢？第一，要找出顶点的下一个元素到对应的叶部之间有多少个元素。前面已经讲过了如何从顶点沿着较大的后代走向叶部的动作，只要每经过一个较大的后代，就往一个计数用的变量（程序中用 level）加上 1 即可，最后调整一下，因为有可能会超越范围。第二，看如何找出这条路径上的元素，回想前几段说过的，如果 son 是该路径上在叶部的节点，它的上一代的脚码为 son/2，再上一代是 son/4，再上一代是 son/8，……；换句话说，如果路径上的某个点与 son 的距离（要经过多少个点才能到 son，但自己不算）是 i，于是它的脚码就是把 son 除以 2 的 i 次方，用 C 的数元运算来写就是 son>>i。

问题解决的方法是：如果用 bottom 表示在二分查找时距 son（也就是最底层的元素的脚码）的距离，那么开始时为 0。再用 top 表示顶点下一个元素到 son 的距离，开始的值是 level。二分法过程中，找出中央元素距 bottom 有多远，这就是 mid，新元素在 x[ ]中的脚码正是 son 除以 2 的 mid 次方，故元素的值就是 x[son>>mid]。因此二分法一定可以找出一个位置来容纳新来的值，其实这个位置正好是二分法结束时 top 的值，因此把在路径上 mid（含）以上的元素全都向上推一格。在 mid 那一层的元素脚码为 son>>mid，那么上一层的脚码为 son>>(mid+1)。最后，把新进的值放到 top 所指的元素，亦即脚码为 son>>top 的元素中。

这就是 fix_heap()的基本概念，通过这个说明，相信不难了解整个程序在做什么事。另外，为了方便起见，sort()中叫做 six_heap()之前的运行方式有少许改变，读者可以看明白。

【习题】

（1）请证明这个做法所用到的比较次数，在最坏情况下，比一般的堆积排序法少，不必实际算出它有多少次比较，这比较难，用一点逻辑推理就可以了。

（2）当然，如果用一个工作用的数组，把沿途经过的脚码记下来，就可以节省不少数

Download at http://www.pin5i.com/

元运算，请用这个想法来修改程序。请问，第一，排列 n 个元素，要用多少额外的内存？第二，两种做法的运行速度有多大差异？

（3）试用 n=100,200,…,900,1000,…,9000,10000 个数据来测试程序，请问它所用时间与比较次数各是多少？与快速排列法相比又如何？

（4）在一般情况下，此方法所需的比较次数可能会比堆积法多，请问这样说的道理何在？

【参考文献】

这个做法是 1987 年由 Svante Carlsson 提出来的，不过笔者相信有不少人知道这个办法，然而 S.Carlsson 是第一个（就笔者所知）把这个方法作了比较完整分析的。他证明了在建立堆积的阶段，此方法使用了不超过 $1.82n-\alpha\log_2 n$ 次比较，α 是个常数，而在堆积建立完成之后的第二阶段（排序），则不会超过 $(n+1)[\log_2(n+1)+\log_2\log_2(n+1)]$ 次比较。于是 Carlsson 证明了这个方法在最坏的情况下所使用的比较次数比快速排列法平均比较次数（大于 $1.38n+(\log_2 n+2)$ 还要少，不过还是比合并法多 $(n\log_2 n-n)$。此外，Carlsson 还建议了其他的方法，但不太适合在本书中讨论。

究竟在使用时，这个变换后的堆积法是否比快速法快，Carlsson 倒是没有提到，建议不妨作个初步的比较。Carlsson 的文章如下：

S.Carlsson. A Variant of Heapsort with Almost Optimal Number of Comparisons. *Information Processing Letters*, 1987(24), pp.247～250

【程序】

程序 14-9（HEAP_NEW.C）

```
/* ------------------------------------------------------------ */
/* FUNCTION  sort :                                             */
/*   This is a modified Heapsort program in which a way         */
/* to minimize number of comparisons has been implemented.      */
/* NOTE that the given array old_x[] is shifted back one        */
/* position and what we are using is the new array x[].         */
/*                                                              */
/* Copyright Ching-Kuang Shene      Mar/22/1990                 */
/* ------------------------------------------------------------ */

#define  YES 1
#define  NO  0

void  fix_heap(int [], int, int, int);


/* ------------------------------------------------------------ */
/* FUNCTION sort :                                              */
/*   Given an array x[] and its number of elements, this        */
```

Download at http://www.pin5i.com/


```
/* function uses fix_heap() to build up a heap and then          */
/* sorts the array by swaping the top and the last items.        */
/* ------------------------------------------------------------- */

void  sort(int old_x[], int n)
{
     int  *x = old_x - 1;
     int  temp;
     int  size, i;

     for (i = n/2; i >= 1; i--)      /* starts off from mid       */
        fix_heap(x, i, x[i], n);     /* fix the heap at this      */

     for (size = n-1; size >= 1; size--) {  /* sort phase.        */
        temp = x[size+1];            /* pick up the last          */
        x[size+1] = x[1];            /* replace with 1st          */
        fix_heap(x, 1, temp, size); /* restore the heap           */
     }
}

/* ------------------------------------------------------------- */
/* FUNCTION  fix_heap :                                          */
/*   A modified fix_heap routine using Svante Carlson's          */
/* suggestion to minimize number of comparisons.                 */
/* ------------------------------------------------------------- */

void  fix_heap(int x[], int root, int key, int bound)
{
     int  son, level;
     int  top, bottom, mid;

     son   = root + root;          /* get the subscript of son   */
     level  = 1;                   /* level counter              */
     for (;son<bound; level++)     /* trace the path of max son  */
        son = (x[son] < x[son+1]) ? (son+1) << 1 : son << 1;
     if (son > bound)
        level--, son >>= 1;        /* son=the leave along max.s  */

     top = level;                  /* start binary search along  */
     bottom  = 0;                  /* the path of max. sons      */
     while (top > bottom) {
        mid = (top + bottom)/2;
        if (key <= x[son >> mid])/* the mid item<=key?           */
           top = mid;              /* squeeze downward           */
```

Download at http://www.pin5i.com/


```
    else
        bottom = mid + 1;    /* squeeze upward                    */
}

for (mid = level-1; mid >= top; mid--) /* copy ...               */
    x[son >> (mid+1)] = x[son >> mid];
x[son >> top] = key;
}
```

问题 5.9 合并法（M_SORT.C）

有一个叫做合并排列法（Merge Sort）的技巧是这样的：把要排序的资料分成大约相等的两组，把这两组的顺序排好之后，通过合并的技巧而合成一个全部都排好的结果。请写一个程序实现这个想法。

【解答】

这并不是一个难题，在程序设计的技巧上要比快速排列法简单得多，如果对这个函数的调用与前面的几个方法相同，也就是：

```
void sort(int x[], int n);
```

的话，就要注意到一件事了。在合并的时候，是没有办法在原地（in-place）工作的，也就是说不能把 A 与 B 部分合并之后结果还在原处，于是要有额外的一个数组，用它来存放合并之后的结果，最后把排序完成的结果从那返回 x[ ]。如何取得内存呢？通过 malloc()函数取得一个容量有 n 个 int（假设要排的是 int）的数组 work，然后把 x[ ]、work[ ]与 x[ ]的起点（0）与终点（n-1）交给真正工作的函数去处理。在应用上，就像前面的快速排列法程序一样，为了要有一个兼容的调用界面，不得不弄出另一个做界面转换工作的程序。在下面的程序（M_SORT.C）中，这个程序也叫做 sort()，当它取得工作用的内存之后，就用下面的方式调用真正的合并排列法的程序：

```
mergesort(x, work, 0, n-1);
```

当返回控制权之后，就把要来的工作用内存 work[]还给系统。

在 mergesort()中的写法与问题说明完全相同，因为交给它的数组并不是全部，而是其中的一段，亦即 x[F]～x[L]，如图 14-1 所示。

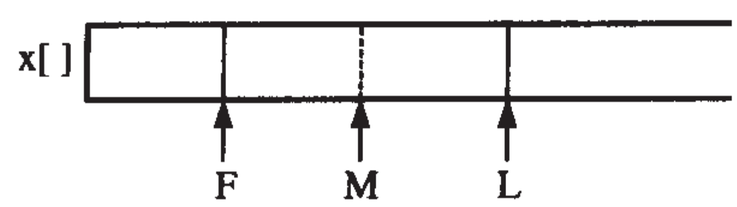


图 14-11

F,M 与 L 对应着程序中的 first、mid 与 last。于是，当 F 比 L 小的时候，x[F]与 x[L]之

Download at http://www.pin5i.com/

间就至少有两个元素，因此把中间元素 M 求出来，也就是 x[(F+L)/2]。接着递归调用自己去处理 x[F]～x[M]，以及 x[M+1]～x[L]，等到从两次递归调用回头之后，那两段内容就都排好了，于是就让合并函数 merge()把它们合并起来存放到 work[F]～work[L]中，最后用 memmove()函数把 work[ ]中的这一段返回 x[ ]中对应的部分，于是 x[F]～x[L]的部分就都排好了。

merge()函数是很简单的，让 merge()处理的还是上一段所提的，即 x[L]～x[M]，以及 x[M+1]～x[L]这两部分。用 3 个脚码（如果喜欢，用指针也无妨），即 in1,in2 与 out；in1 与 in2 的起点是 L 与 M+1，也就是那两段数据的第一个元素，而 out 则是 L，表示在 work[ ]中从什么地方起存放结果。程序中的第一个 while 循环，每绕一次，就把 x[in1]与 x[in2]中比较小的那一个挤出来，放到 work[out]中，并且更新脚码值。第 2 个 while 在当 in1 小于 M 时会有动作，它把在左边那一段中并没有被挤出来的元素全都传到 work[ ]中，这表示这个元素是最大的部分；第 3 个 while 则是对右边一段操作的，道理同前。注意第 2 或第 3 个 while 只可能有一个会执行。

在测试这个程序时，不要忘了用 SORTTEST.C 作主程序。

【习题】

（1）在 merge()函数中，有：

```
work[out++] = (x[in1] < x[in2]) ? x[in1++] :x[in2++];
```

如果改为：

```
work[out++] = (x[in1++] < x[in2++]) ? x[in1] :x[in2];
```

请问会有什么影响？

（2）在合并 x[ ]与 y[ ]两个各有 p 个与 q 个元素的数组时，最多会有 p+q 次比较。请证明这一点，接着用这一事实导出合并排列法在最坏情况下的比较次数。

【程序】

程序 14-10（M_SORT.C）

```
/* ------------------------------------------------------------ */
/* FUNCTION merge sort :                                        */
/*   Given an input int array and number of elements,           */
/* this function sorts the array with merge sort method.        */
/*                                                              */
/* Copyright Ching-Kuang Shene              July/21/1989        */
/* ------------------------------------------------------------ */
#include <stdlib.h>          /* for malloc() and free()         */
#include <string.h>          /* for memmove()                   */

#define  SIZE    sizeof(int)/sizeof(char)  /* int size         */
```

Download at http://www.pin5i.com/


```
void  sort(int [], int);
void  mergesort(int [], int[], int, int);
void  merge(int [], int [], int, int, int);

/* ----------------------------------------------------------- */
/* FUNCTION sort :                                             */
/*   Get working space and start merge sort.                   */
/* ----------------------------------------------------------- */

void  sort(int x[], int n)
{
     int  *work;

     work = (int *) malloc(n*SIZE); /* get work space          */
     mergesort(x, work, 0, n-1);    /* call merge sort         */
     free(work);                    /* free work space         */
}


/* ----------------------------------------------------------- */
/* FUNCTION mergesort :                                        */
/*   The merge sort driver routine.                            */
/* ----------------------------------------------------------- */

void  mergesort(int x[], int work[], int first, int last)
{
     int  mid;

     if (first < last) {
        mid = (first + last)/2;          /* cut from the middle */
        mergesort(x, work, first, mid);  /* sort left           */
        mergesort(x, work, mid+1, last); /* sort right          */
        merge(x, work, first, mid, last);/* merge them          */
        memmove(&x[first], &work[first], (last-first+1)*SIZE);
     }
}

/* ----------------------------------------------------------- */
/* FUNCTION merge :                                            */
/*   Merge routine for merge sort.                             */
/* ----------------------------------------------------------- */
```

Download at http://www.pin5i.com/


```
void  merge(int x[], int work[], int first, int mid, int last)
{
     int  in1 = first;
     int  in2 = mid+1;
     int  out = first;

     while (in1 <= mid && in2 <= last)       /* merge part       */
        work[out++] = (x[in1] <= x[in2]) ? x[in1++] : x[in2++];
     while (in1 <= mid)                  /* output remainder in 1at */
        work[out++] = x[in1++];
     while (in2 <= last)                 /* output remainder in 2nd */
        work[out++] = x[in2++];
}
```

问题 5.10　桶子法（BUCKET.C）

快速排列法（Quick Sort）、二分插入法（Binary Insertion Sort）、堆积法（Heap Sort）、合并法（Merge Sort）等，都至少要用与 $n\log_2 n$ 成正比的比较次数来排 n 个数据，当然不一定是数值。但如果要排的不过是一些正整数（unsigned int），有没有更快的办法？请利用所学知识写出这个特殊的排序的程序，理想是打破 $n\log_2 n$ 的限制。

【解答】

在平常的课程中，一个叫做 Bucket 或 Radix Sort 的方法常常被人忽略，大家都爱谈快速法、合并法等，而会忘了 Bucket Sort（桶子法）的存在。事实上 Bucket Sort 对某些特殊资料而言，具有极端快速的处理能力，因此在此的解答就是这个桶子法。首先说明这个桶子法的概要，再来讲程序的编写方式。假设要排下面的 10 个值（注意：正整数）：234,134,142,114,334,242,232,321,121 与 212。先看十进制的情形，从个位数起向左一位一位地排过去，最终把所有数都排列完成，排的是位数，而不是整个数，因此它就有了 Radix Sort 这个名称。

第一步，把个位数依 0～9 分成 10 组，接着把每一个输入数据依顺序查一次，按照它的个位数的值而放到对应的分组中。注意，放入分组中的数据要放在先放入的数据的后面。这个操作就叫做分组，分组完后有

0	1	2	3	4	5	6	7	8	9
无	121	142	无	234	无	无	无	无	无
		242		134					
		232		114					
		322		334					
		212							

Download at http://www.pin5i.com/

在此只有1、2与4这3组有数据，其他分组因为个位数无该分组的对应值，所以就空无一物。

下一步叫做收集，从第0组到第9组，把数据再串回来，在同一组中的数据的先后顺序不变，因此在串回来的数据中最后一位是排好顺序的：

121 142 242 232 322 212 234 134 114 334

接着再做一次分组的操作，不过这次对象是十位数了，分组的结果如下：

0	1	2	3	4	5	6	7	8	9
无	212	121	232	142	无	无	无	无	无
	114	322	234	242					
			234						
			134						
			334						

注意，在同一分组中的各个数据的十进制是相同的。另外，因为分组中的顺序与处理顺序相同，所以分组中的个位与十位数都已经排好了。接着做一个收集的操作：

212 114 121 322 232 234 134 334 142 242

收集后的数据最后两位是排好了的，原因是在第二次收集时等于在排十进制，但以十进制分组时的各个分组中又已经依个位数排好了，于是第二次收集就等于排好了最后两位。

其次，排百位数，先做分组的操作。

0	1	2	3	4	5	6	7	8	9
无	114	212	322	无	无	无	无	无	无
	121	232	334						
	134	234							
	142	242							

于是再次的收集操作就把百位数排好了，但因十位与个位都已排列完成，所以收集完后就是依顺序排列的结果：

114 121 134 142 212 232 234 242 322 334

综合起来，从右往左一位接一位地处理；对每一个位数，先作一次分组，把它分到各个以位数值为名称的组别中，当每一个数据都分组完成之后，再依各组从小到大地把组中元素收集回来，该位数就排列完成。一旦所有位数都排好了，收集起来的就是依顺序排列的结果。

正因为数据要在分组中来回移动，把数据放在连接串行中是很理想的，因此先把输入数据放到一个连接串行中，并且用一个变量head指向连接串行开头，如图14-12所示。

Download at http://www.pin5i.com/

图 14-12

如何表示分组呢？很简单，用一个表格就可以。不过，在分组时新数据要加在后头，而收集时又是从头开始，所以表格中每一栏都有两个变量 first 与 last，前者指向这个分组中的第一个元素，后者则指向最后一个；整个表格中，每一栏就对应着一个分组，把这个表格叫做 buckets，如图 14-13 所示。

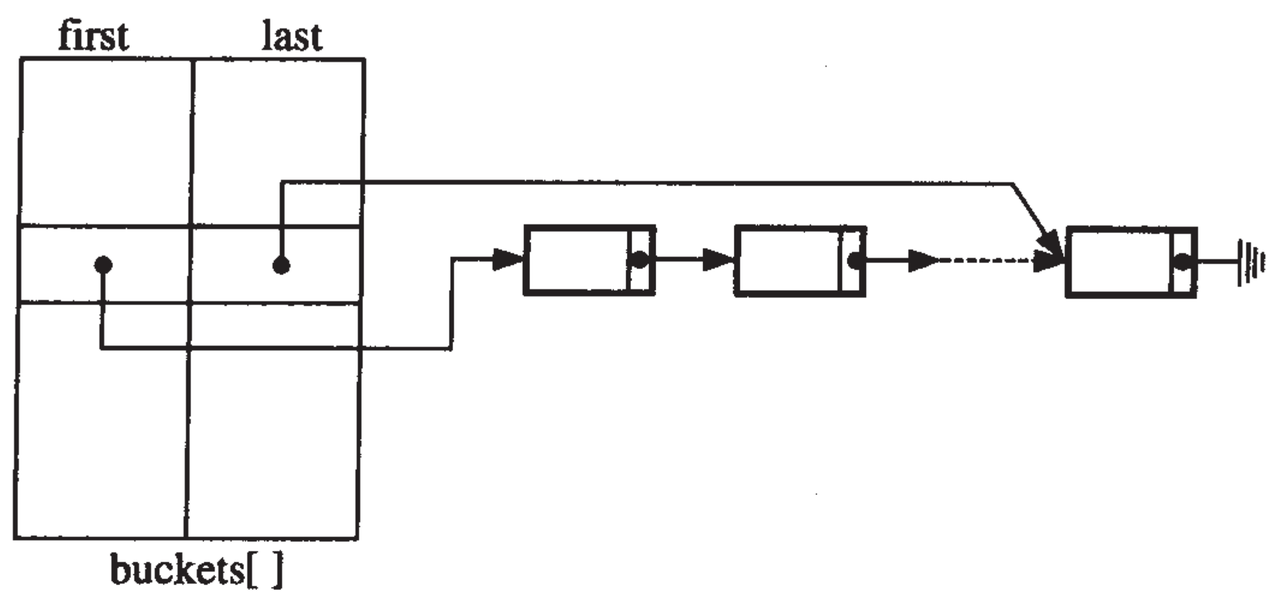


图 14-13

所以分组的操作，就只不过是每次从 head 取出一个元素，依对应的位数值放到 buckets[ ]中对应字段的连接串行中，并且更新 last 栏与 first 栏（如果这是第一次放进去的话）。在收集时，就依 bucket[0],bucket[1],…这样一栏一栏地处理，如果那一栏中有连接串行，就把串行中的元素串回 head 所指的串行的尾部就行了。

当每一个位数都处理完了，head 所指的连接串行就是依顺序排好的，从头到尾走一次，把数据放回数组中就大功告成了。

【程序说明】

在程序 BUCKET.C 中，连接串行的元素有 3 个字段，data 是原输入数据；x 是工作用的，后面再谈；next 则是指向下一个元素的指针。buckets[ ]数组的结构同前，但是 buckets[ ]中该有多少栏呢？如果把输入数据看成是十进制数的 unsigned int，则有 10 组（0～9），如果它们最多五位数，则要分组 / 收集 5 次，一次 1 位；但有更好的做法，把那个数看成十六进位，因此一个位数的值就是 0～15，所以两个数组的正整数就有 4 位，4 个数组的正整数有 8 位。虽然 buckets[ ]此时要有 16 栏，但因为只有 4 个（如果 unsigned int 是 2 个数元组）位数，于是分组 / 收集的操作就只做 4 次。

第一次做分组 / 收集操作时，是用低次位的 4 个数元，准备一个数元值 0x0FU，在要找出 data 中的值是在哪一个分组时，拿 data 与 0x0FU 做数元的&就可以取出它的值，那正是在 buckets[ ]中的脚码。不过这样做有一点不方便，因为在处理高次位数元的值时，就要把 data 向右移 4 个数元再与 0x0FU 做&运算，所以为了方便起见，在一开始时把 data 的值放到 x 中。0x0FU 是与 x 运算，一旦用过了，就把 x 向右移 4 位，把较高次位的数元移到低次位去，以便下次再用。

Download at http://www.pin5i.com/

在程序中 listgen()函数接收一个数组，为数组中的每一个值通过 malloc()函数取得内存而建立一个连接串行，并且返回指向这个连接串行开头的指针。注意，并没有检查 malloc()是否成功地取得了内存，因此要注意这件事，尤其是大量数据时更是如此。

distribute()函数把 listgen()建立的串行从头到尾查一次，对每一个元素而言，取出 x 值的后 4 个数元，作为 buckets[ ]数组的脚码，而把该元素串到 buckets[ ]中对应的串行去。假设已经了解指针的运行，因此就不作说明了。

recollect()则是做收集的操作，它把 buckets[ ]中有元素的字段中的串行取出，重新把数据串回 head 所指的连接串行去。

最后，putback()函数则是把连接串行中已经排好的结果放回数组去，并且归还所有先前由 malloc()所取得的内存。

sort()是整个事件的主控者。一开始时，它用 listgen()建立输入数据的连接串行，然后 4 个数元为一组地处理，在每一次分组之前，把 buckets[ ]的两个字段清成 NULL，接着是分组／收集的操作，这样一直反复，到所有位数都处理完后，通过 putback()把结果存回数组，并且结束工作。

注意，应使用 SORTTEST.C 做主程序来做测试工作。

【效率问题】

它的效率如何？如果输入数据被分解成 α 位，于是在 sort()中的外循环就执行 α 次，每一次都做一个分组／收集的操作，在这两个程序中，每执行一次，就会把所有数据查一次，因此一次分组／收集就要查每一个输入元素两次，总共就有 2n 个查元素的操作。但是这要反复 α 次，共有 2αn 次操作。不过在 listgen()与 putback()中也都各查了一次，所以这个程序总共查了 2(α+1)n 次。一般而言，α 总是个常数（这是对实际机器而言，在理论上不能做这样的假设），与 n 无关，所以查 n 个元素的次数与 n 成正比，是个线性 O(n)的程序。由此看来，Bucket Sort 是要比其他常见的方法快，n 越大时越是如此。不过它有一个缺点，要有额外的内存，不能在原地把工作完成。当内存不成问题，且又要排正整数时，不妨考虑这个方法。另外，这个程序比其他排顺序的程序都要长，如果依下面习题的建议来做几个实验的话，就会得到它“快”的感觉。程序效率与它所使用的方法有关，而与它的长短并没有太大的关系。

【习题】

（1）请修改程序，用 8 个数元（也就是 255 进位）、2 个数元（4 进位）来执行程序，请问执行时间会有什么差异？用 65536 进位有意义吗？

（2）这个程序多了 x 这一个工作用的字段，请修改程序把它去掉。请问，程序在效率上有差异吗？

（3）这个程序有没有所谓的最坏情况与平均情况？请说明理由。

（4）将这个程序在 n=100,200,…,900,1000,2000,…,10000 个数据之下它的执行时间记录下来，看看它是否真的比快速排列法快？如何达到快速的？时间值与 n 成正比吗？注意：使用 PC 的读者在 n 很大的时候，留心记忆模式（Memory Model）的问题。

Download at http://www.pin5i.com/

【参考文献】

Bucket 或 Radix Sort 也是课本中常见的教材，因此不必多说。关于它与其他排序方法的一些简单比较与讲解，请参看下面书籍。

洗镜光．一个差不多被人遗忘的排序技巧－有效率地解决问题之（三），信息传真．1989(8), No. 22, pp.133～147

【程序】

程序 14-11（BUCKET.C）

```
/* ------------------------------------------------------------------ */
/* FUNCTION  bucketsort :                                             */
/*   A list implementation of Bucket sort algorithm for               */
/* int's.  This package contains several functions as                 */
/* follows :                                                          */
/*                                                                    */
/*   sort    :   the control program for bucket sort.                 */
/*   listgen :   this function accepts an input array                 */
/*               and number of elements in that array                 */
/*               than generates a list containing 3                   */
/*               items, namely data (input), x (the                   */
/*               shifted data item) and next (a list                  */
/*               pointer.  The head of this list is                   */
/*               stored in an external variable 'head'.               */
/*   distribute :   this function dispatches data from the            */
/*                  list to a set of buckets.                         */
/*   recollect  :   this function collects data items from            */
/*                  those buckets and rebuilds the list.              */
/*                  Also all data items will be right                 */
/*                  shfited several bits for next run.                */
/*   putback :   this function retrieves data item in                 */
/*               the sorted array and put them back to                */
/*               the input array.                                     */
/*                                                                    */
/* Copyright Ching-Kuang Shene      Feb/26/1989                       */
/* ------------------------------------------------------------------ */

#include <stdio.h>
#include <stdlib.h>

#define  LENGTH  4        /* mask size                                */
#define  BUCKET_NO   16   /* number of buckets=2*LENGTH               */
#define  LOOP_COUNT  sizeof(int)*8/LENGTH
```

Download at http://www.pin5i.com/

```
/* ------------------------------------------------------------ */
/*                  external data items                         */
/*------------------------------------------------------------- */

struct node {                   /* data structure for list      */
     int  data;                 /*   original data              */
     int  x;                    /*   working data               */
     struct node   *next;       /*   list pointer               */
};

struct node *head;              /* list head                    */

struct cell {                   /* bucket cell data structure   */
     struct node *first;        /*   point to 1st list ele.     */
     struct node *last;         /*   point to last list ele.    */
} buckets[BUCKET_NO];

int  mask = 0x0FU;              /* the mask 000...1111          */


/* ------------------------------------------------             */
/*                  function prototypes                         */
/* ------------------------------------------------             */

struct node *listgen(int [], int);
void  distribute(void);
void  recollect(void);
void  putback(int []);


/* ------------------------------------------------             */
/* FUNCTION sort :                                              */
/*   control routine of Bucket sort.                            */
/* ------------------------------------------------             */

void sort(int input[], int n)
{
     int  i, count;
```

Download at http://www.pin5i.com/


```
    head = listgen(input, n);        /* get the data item list   */

    /* iterates enough times to sort the list                    */

    for (count = 1; count <= LOOP_COUNT; count++) {
        for (i = 0; i < BUCKET_NO; i++) /* initial buckets       */
            buckets[i].first = buckets[i].last = NULL;
            distribute();
            recollect();
        }
        putback(input);
    }

    /* ------------------------------------------------------    */
    /* FUNCTION listgen :                                        */
    /*   This function takes an input array input[] and put      */
    /* all elements therein to a list.                           */
    /* ------------------------------------------------------    */

    struct node *listgen(int input[], int n)
    {
        struct node *first;
        struct node *last;
        struct node *temp;
        int   i;
        first = last = (struct node *) malloc(sizeof(struct node));
        first->data = first->x = input[0];
        first->next = NULL;
        for (i = 1; i < n; i++) {
            temp = (struct node *) malloc(sizeof(struct node));
            temp->data  = temp->x = input[i];
            temp->next  = NULL;
            last->next  = temp;
            last = temp;
        }
        return first;
    }

    /* ------------------------------------------------------    */
    /* FUNCTION distribute :                                     */
    /*   This function distributs data items stored in the list  */
    /* to backets according to the value of last 'LENGTH' bits.  */
    /* ------------------------------------------------------    */
```

Download at http://www.pin5i.com/


```
void distribute(void)
{
     struct node *temp;
     int index;

     while (head != NULL) {
         index = (head->x) & mask;
         (head->x) >>= LENGTH;
         if (buckets[index].first == NULL)
            buckets[index].first = buckets[index].last = head;
         else {
            (buckets[index].last)- >next = head;
            buckets[index].last        = head;
         }
         temp      = head->next;
         if (head->next != NULL)
            head->next = NULL;
         head      = temp;
     }
}

/* ------------------------------------------------------------ */
/* FUNCTION recollect :                                         */
/*   This function collects data items from buckets and         */
/* buckets information for next run.                            */
/* ------------------------------------------------------------ */

void recollect(void)
{
     int  i, j;

     for (i = 0; i < BUCKET_NO && buckets[i].first == NULL; i++=
         ;
     head = buckets[i].first;    /* i indicates 1st non-NULL     */
     for (j = i+1; j < BUCKET_NO; j++=
         if (buckets[j].first != NULL) {
            (buckets[i].last)->next = buckets[j].first;
            i = j;
         }
}
```

Download at http://www.pin5i.com/


```
/*-----------------------------------------------------------*/
/* FUNCTION putback :                                        */
/*   This function puts all sorted elements in the list      */
/* back to the given array and frees storage occupied by     */
/* the list.                                                 */
/* ---------------------------------------------------------*/

void  putback(int input[])
{
     struct node *temp;
     int    i;

     for (i = 0; head != NULL; i++) {
        input[i]     = head->data;
        temp = head->next;
        free(head);
        head     = temp;
     }
}
```

问题 5.11　单一重复元素排序（LOT_DUP.C）

如果一个数组中有 n 个元素，但并非每一个元素都不相同。事实上，只有大约 $\alpha\log_2 n$ 个值是不同的，其余 $n-\alpha\log_2 n$ 个值全部一样，请发展一个特别快的程序把数组的元素排好顺序。α 是个常数，为了方便起见，可以当成 1；另外，$n-\alpha\log_2 n>0$，而且 n 比 $\alpha\log_2 n$ 大很多。请问程序用了多少次比较？

【解答】

要让程序快，最重要的就是降低比较的次数，n 个元素中只有大约 $\log_2 n$ 个不同，只要排 $\log_2 n$ 个元素即可。正因为如此，第一件事就是要找出哪一个元素是重复出现的。

因为在 n 个元素中只有 $\log_2 n$ 个不同，其余的 $n-\log_2 n$ 个都一样，于是就至少有一对相邻的元素是一样的。如果相邻的元素都不同，则最多有 $2\log_2 n$ 个元素而已，还不足 n 个元素，因此可以比相邻的元素（x[0]与 x[1]，x[1]与 x[2]，…），一旦找到了有相同的一对，就把那个值存入 dup，而这正是重复了 $n-\log_2 n$ 次的值，一共用了多少次比较呢？$2\log_2 n$ 次，只有在不同的元素之间都有一个 dup 才会发生，比完这 $2\log_2 n$ 个元素，下一对就一定是一样的了。

举例说明，在问题说明中 3 与 7,7 与 3 比都不同，但下一组（x[2]与 x[3]）就是一样的了，比较次数用得最多的，就是 3,7,3,4,3,9,3,…，比了 6 次，第 7 次才发现 dup 是 3。

知道了 dup 之后，可以把不等于 dup 的元素都传到 x[ ]数组开头，因此做完这件事之

Download at http://www.pin5i.com/

后，x[ ]的开头有 $\log_2 n$ 个不同的值。注意，dup 也得算是一个，因为它与其他的 $\log_2 n-1$ 个元素不同。用问题说明中的数据来看，就是 7,4,9,3。这个操作用了多少次比较呢？最多 n 个，为什么？把这 $\log_2 n$ 个元素用任何方法去排列，如果采用的是堆积法，则需要与 $(\log_2 n)(\log_2(\log_2 n))$ 成正比的比较次数。用上面的例子，排列完成后就是 3,4,7,9。

最后一步就是把重复的数据填回去。这很简单，从最后一个元素起，如果它不是 dup，就把它传到 x[ ]的第 n-1 个位置。再看倒数第 2 个，如果它不是 dup，则移到 x[ ]的第 n-2 个位置等，一直到 dup 出现为止，从 dup 所在的位置起，到 x[ ]后面还没有填满的这一段位置中，都得填入 dup 这个值，然后就大功告成了。请问这要多少次比较？

在程序 LOT_DUP.C 中，第 1 个 for 找出重复的元素放到 dup 中；第 2 个 for 把值不是 dup 的元素都移到 x[ ]的开头，count 计算有多少个元素。不过要注意，dup 也是这 $\log_2 n$ 个不同元素其中之一，所以把它放在最后面。接着，用本书中的任何排序程序（比如快速法、堆积法、合并法其中之一）把这一个段落顺序排好；最后一个 for 则是从后往前查这 $\log_2 n$ 个元素，如果它不是 dup，就往后移，直到出现一个等于 dup 的元素为止，这是一定可以找出来的，因为把 dup 放到前 $\log_2 n$ 个元素中参与排列。index 是 x[ ]中从右往左填数据用的脚码，而 i 则是前 $\log_2 n$ 个元素从右往左查时所用的脚码。但 x[i]与 dup 不同，于是 x[i]～x[index]之间用 dup 填满就行了，至于 x[0]～x[i-1]的内容是不必更动的。

【习题】

（1）为什么程序中的第一个 for 循环是对的？请用严格的数学方法证明它。

（2）请证明这个程序所用的比较次数与 n 成正比。

（3）请问，当 n 的范围是多少时，这个程序才能正常运行。

（4）如果 α 不是 1，而是某个常数，能改变这个程序而使它依旧可以正常运行吗？n 至少要是多少？如果 α 未知，能否从输入数据中估计出来呢？要用多少次比较？值得这样做吗？

【程序】

程序 14-12（LOT_DUP.C）

```
/* ------------------------------------------------------------ */
/* FUNCTION dup_sort :                                          */
/*   Given an array of n elements with about log(n)             */
/* duplicated items, this function sorts the array with         */
/* some tricks.                                                 */
/*                                                              */
/* Copyright Ching-Kuang Shene      July/16/1989                */
/* ------------------------------------------------------------ */

void  sort(int [], int);          /* use any sort function      */
void  dup_sort(int  x[], int n)
{
```

Download at http://www.pin5i.com/


```
int  dup;
int  count, index, i, j;

for (i = 1; i < n; i++)     /* detect the dup. item          */
   if (x[i] == x[i-1]) {
      dup = x[i];           /* store it in variable dup      */
      break;
   }

for (count = i = 0; i < n; i++)/* move other items           */
   if (x[i] != dup)         /* to the beginning of the       */
      x[count++] = x[i];    /* array and add one dup         */
x[count++] = dup;           /* at the end                    */

sort(x, count);             /* then sort these log(n)        */

for (index=n-1, i=count-1; i >= 0 && x[i] != dup; i--)
   x[index--] = x[i];       /* put the dup. items back       */
if (i >= 0 && x[i] == dup)
   for (j = i + 1; j <= index; j++)
      x[j] = dup;
}
```

问题 5.12　均匀重复元素排序（LOG_DUP.C）

如果一个数组有 n 个元素，但却有 $\alpha\log_2 n$ 不同的值，换句话说，数组中有 $\alpha\log_2 n$ 不同的值，每一个都重复出现了 $n/\alpha\log_2 n$ 次。请写一个特别快的程序把数组的元素排好顺序。假设 n 很大，比 $\alpha\log_2 n$ 大很多，α 是个常数。请问，程序用了多少次比较。为了方便起见，可以把 α 当成 1。

【解答】

因为 n 个数据中只有 $\log_2 n$ 个不同的值，当 n 比较大时，剩下来的 $n-\log_2 n$ 个数据重复出现的机会居多，所以做法是把 n 个数据分成 $n/\log_2 n$ 组，每一组 $\log_2 n$ 个值。使用比较快速的排序技巧，如快速排列法、合并法、堆积法等，把这 $n/\log_2 n$ 组中的数据的顺序排好，因为每一组都只有 $\log_2 n$ 个值，而合并法等对 m 个数据只要 $m\log_2 m$ 次比较，所以 $\log_2 n$ 个值就只要 $(\log_2 n)(\log(\log_2 n))$ 次比较而已，但因为有 $n/\log_2 n$ 组，因此总比较次数就是：

$$\frac{n}{\log_2 n}(\log_2 n)(\log_2 \log_2 n) = n\log_2 \log_2 n$$

Download at http://www.pin5i.com/

因为 $\log_2\log_2 n < \log_2 n$，所以在这一步已经降低了比较次数。但问题是，整个数组并没有排列完成，只不过是分段地排好而已，离全部排好还有一段距离。

现在 n 个数据中只有 $\log_2 n$ 个不同的值的条件有用了。把 n 个数据已经分成 $n/\log_2 n$ 段排好（每一段有 $\log_2 n$ 个元素），如果把两段合并起来（正因为要合并，所以才先进行排列，合并后的结果有多少个元素呢？猛一看来，若不细心些，答案可能是 $\log_2 n+\log_2 n$ 个，因为每一段都有 $\log_2 n$ 元素。对，但也不太对。说“对”，是因为用一般情况来看问题；说“不对”，则是因为没有考虑到特殊情况。注意，因为只有 $\log_2 n$ 个不同的值，所以把两组各有 $\log_2 n$ 数据的分段合并起来，得到的结果也不外乎那 $\log_2 n$ 个不同的值而已。因此，可以准备两个数组，一个存储合并后不同的值，另一个存储合并后每一个不同的值各出现多少次。

例如，如果第一组数据是 1,4,4,5,7,8,9,9；第二组为 2,4,5,5,7,7,8, 8；每一组各有 8 个数据，而也只有 8 个不同的值。于是合并起来后的数据是 1,2,4,5,7,8,9，而另一个数组则是 1,1,3,3,3,3,2，分别指出 1,2,4,5,7,8,9 各有 1,1,3,3,3,3,2 个。

就这样，把已分段的数据逐段地合并起来，就会得到最终结果。最终结果是什么？很简单，只有 $\log_2 n$ 个数，以及记录每一个数各重复出现了多少次，因此把它展开就行了。那么在合并时作了多少次比较呢？合并两个有 $\log_2 n$ 元素的数组需要 $2\log_2 n$ 次比较，然后再将剩下来的 $n/\log_2 n-2$ 个段落与最先合并完成的段落再合并就行了。有一点是很重要的，即合并的操作永远只会得到 $\log_2 n$ 个值，所变动的是各个值的出现次数，因此就一共有 $(n/\log_2 n)(2\log_2 n)=2n$ 次比较而已。综合起来，一共有最多与 $n\log_2\log_2 n$ 成正比的比较次数，这要比一般的排大小方法要用到与 $n\log_2 n$ 成正比的比较次数快很多。

写成程序也不难，参看 LOG_DUP.C。LOG_sort()是主宰全局的部分。首先它用 sort()函数把 n 个数据中的每 $\log_2 n$ 个数据排列完成，然后用 malloc()调用 6 个数组，这 6 个数组是 in1_data[ ]，in1_count[ ]，in2_data[ ]，in2_count[ ]，out_data[ ]与 out_count[ ]。它们两个一组，前两组是用来合并的输入，后一组则存放合并后的输出。接着调用 LOG_merge()把各个分组（一共 $n/\log_2 n$）合并起来，结果在 in1_data[ ]与 in1_count[ ]中，用 number 记录下 in1_data[ ]，亦即 in1_count[ ]中有多少个数据，最后用 LOG_expand()把合并后的数据展开，把内存还给系统，工作就完成了。

为了清晰起见，现举例说明。如果有 16 个数据，其中有 4 组不同的值（注意，并没有用 n 与 $\log_2 n$ 的关系，原因在于说明整个操作方式，而不在于 n 与 $\log_2 n$ 的限制），如果这 16 个资料如下：

3 4 3 2 1 3 1 1 4 3 2 2 4 2 1 4

把它们分成 4 组，再排好顺序：

<u>2 3 3 4</u> <u>1 1 1 3</u> <u>2 2 3 4</u> <u>1 2 4 4</u>

LOG_merge()的工作是把它合并起来：合并第 1 组与第 2 组，得到：

1(3) 2(1) 3(3) 4(1)

在数字后面的是该值重复出现的次数；把这个结果与第 3 组合并：

1(3) 2(3) 3(4) 4(2)

Download at http://www.pin5i.com/

再与第 4 组合并：

1(4) 2(4) 3(4) 4(4)

这就是 LOG_merge()的工作。至于 LOG_expand()就简单了，它把值依重复次数展开就会得到：

1 1 1 1 2 2 2 2 3 3 3 3 4 4 4 4

了解了 LOG_merge()的运行方式之后，来看实际的程序。在 LOG_ merge()中为了运行方便，引进了一个叫做 compress()的函数（其实可以不必如此，因为这会多做一些比较工作，但如何省下这一步就留作习题吧），它的动作很简单，就是把每一个分段中的 $\log_2 n$ 个数据中不同的列出来，并且加上各个数据重复若干次的信息而已。例如，上面数据的第一组经处理后就是 2(1) 3(2) 4(1)；它的运行方式很简单，就留给读者自己研究吧。

在一开始，LOG_merge()把 x[ ]数组第一段取 seg_size 个元素给 compress()处理，结果放在 in1_data[ ]与 in1_count[ ]中。接下来的 for 循环就从第二段起一段接一段处理。首先它要定出目前的分段有多少个元素，不外乎有 seg_size 个，或者是在最后不满 seg_size 时取其极小值，结果存放在 len 中。接下来，用 compress()处理这一段，结果存入 in2_data[]与 in2_count[ ]中。这样在合并动作之前，有两组输入 in1_data[ ],in1_count[ ]与 in2_data[ ], in2_count[ ]。合并操作与一般做法无异，不过因为不打算使合并后的长度增加，所以有如下变换：

（1）in1_data[i]<in2_data[j]：当然就应该输出 in1_data[i]与 in1_count[i]，输出放到 out_data[ ]与 out_count[ ]的下一个位置。

（2）in1_data[i]>in2_data[j]：运行相反，输出 in2_data[j]与 in2_count[j]。

（3）in1_data[i]==in2_data[j]：在平常，只输出其中一个就行了，但这样做就会使 out _data[ ]中有重复的数据，所以做法是：输出其中的一个，但在输出中的重复次数是 in1_count[i]与 in2_count[j]的和，接着在输入中把两个“一起”移到上一个去；于是输出还是收到一个值，不过它的重复次数倒是增加了。

当有一个输入没有数据时，与往常一样，把另一个输入抄到输出去。当两个输入都用完了，就表示合并完成。当然，此时结果在 out_data[]与 out_count[]中；下一个步骤，把 out_data[ ]、out_count[ ]、in1_data[ ]与 in1_count[ ]互换，因而前者变成输入，后者变成输出。因为在 C 语言中数组不过是个指向某个区域开头的指针，而且这个区域又是用 malloc()取得，所以数组的指针就不是常数，于是用 SWAP()集体指令（个人设计的）把指针对调就行了，对调完成之后，就继续处理下一段数据。

函数 main()的运行，首先问要处理多少个数据，读入之后存到 n 中，然后在第一个 for 中产生 $\log_2 n$ 个随机数（有可能彼此相同），并且把它们重复 $n/\log_2 n$ 次；接着下一组 for 产生 0～n 的随机数，把上面提到的数据的顺序弄乱，再调用 LOG_sort()处理，并且显示结果。

注意：这个程序要用到本书中的其他排序程序，可以用 QSORT.C、HEAPSORT.C 或 M_SORT.C 其中之一，或其他合适的函数。

许多程序员在看到这个解法时常会讽刺说“小题大做”，明明用 QSORT.C 处理一次就好了，为什么大动手脚、劳师动众呢？笔者不愿意对他们有任何批评，但是却愿意提醒一

Download at http://www.pin5i.com/

点，在处理的数据有特殊性质时，如果用一般性的工具就会比较慢，当数据量大到这一个“慢”的因素变得很明显时，那就是程序员的责任了。此题目说明，处理一些特殊类型数据的技巧，并不是一件坏事，很可能会对编程者的思考、技术有所启发，固步自封是一个不好的现象。

【习题】

（1）前面提到过，compress()是为了使 LOG_merge()清楚才加进去的，有没有办法改写 LOG_merge()，而不用 compress()呢？

（2）请问，在 compress()中一共作了多少次比较，因此使整个程序的总比较次数增加为多少？还是与 $n \log_2 \log_2 n$ 成正比吗？

（3）如果学过数据结构，要做一做本题。在数据结构的知识中有哪一个结构可以很漂亮地做出这个问题，而且比较的次数还是与 $n \log_2 \log_2 n$ 成正比呢？（提示：哪一种树状结构有这个特色？）

（4）请用 n=100,200,…,900,1000,2000,…,100000 来测试这个程序，并且计算时间；再用其他方法做一次，请问哪一个比较快？当 n＞10000 时又如何呢？请作出结论。

【程序】

程序 14-13（LOG_DUP.C）

```
/* ----------------------------------------------------------- */
/* FUNCTION  LOG_sort :                                        */
/*   Given an array of n elements with only log(n)             */
/* distinct elements and each element maybe repeated           */
/* n/log(n) times, this function sorts this highly             */
/* duplicated array with merge technique.                      */
/*   This function requires a sorting routine such as          */
/* quick sort, merge sort or heapsort, but not bucket          */
/* sort discussed in this book.                                */
/*                                                             */
/* Copyright Ching-Kuang Shene      July/27/1989               */
/* ----------------------------------------------------------- */

#include <stdlib.h>                          /* for malloc()    */
#include <math.h>                            /* for log()       */

#define  MIN(x, y)   ((x) <= (y) ? (x) : (y))
#define  SWAP(x, y)  { int *t; t=x; x=y; y=t; }

/* ----------------------------------------------------------- */
                 external data                                 */
/* ----------------------------------------------------------- */
```

Download at http://www.pin5i.com/


```
int  *in1_data,  *in2_data,  *out_data;      /* for data        */
int  *in1_count, *in2_count, *out_count;     /* for counting    */
int  number;                                 /* result count    */

/* ----------------------------------------------------------- */
/*                  function prototype                         */
/* ----------------------------------------------------------- */

void LOG_sort(int [], int);                  /* main funct.     */
int  LOG_merge(int [], int, int);            /* for merge       */
int  compress(int [], int, int *, int *);    /* compress        */
void LOG_expand(int []);                     /* expand          */
void sort(int [], int);                      /* sorting         */

/* ----------------------------------------------------------- */
/* FUNCTION  LOG_sort :                                        */
/*   Control driver.  It cuts the given array into small       */
/* segments with about log(n) elements each and then sorts     */
/* these segments.  Finally by using a modified merging        */
/* technique we can merge these segments in a faster way.      */
/* ----------------------------------------------------------- */

void  LOG_sort(int x[], int n)
{
     int  log_n  = (int) (log((double)n) + 0.5);
     int  i;
     for (i = 0; i < n; i += log_n=     /* sort segments        */
         sort(x+i, MIN(log_n, n-i));

/* ----------- allocate working spece ---------------           */

     in1_data  = (int *) malloc(sizeof(int)*log_n);
     in2_data  = (int *) malloc(sizeof(int)*log_n);
     out_data  = (int *) malloc(sizeof(int)*log_n);
     in1_count = (int *) malloc(sizeof(int)*log_n);
     in2_count = (int *) malloc(sizeof(int)*log_n);

     out_count = (int *) malloc(sizeof(int)*log_n);

     number = LOG_merge(x, n, log_n);    /* merge segments       */
     LOG_expand(x);                      /* expand the final     */

     free(in1_data);                     /* free working space   */
```

Download at http://www.pin5i.com/


```
    free(in2_data);
    free(out_data);
    free(in1_count);
    free(in2_count);
    free(out_count);
}

/* ----------------------------------------------------------- */
/* FUNCTION  LOG_merge :                                       */
/*   A special merge routine.  It calls compress() to          */
/* compress the segments into a data/counting form so that     */
/* there is no duplications.  Then merges these short          */
/* forms.  Because there are only log(n) distinct items,       */
/* the merged segment of any two segments contains no more     */
/* than log(n) elements.                                       */
/* ----------------------------------------------------------- */

int  LOG_merge(int x[], int n, int seg_size)
{
    int  no1, no2;                  /* # of elements in two in */
    int  start, len;                /* start is the cursor     */
    int  i, j, k;

    no1 = compress(x, seg_size, in1_data, in1_count);

    for (start = seg_size; start < n; start += len= {
        len = MIN(seg_size, n-start);  /* compute length       */
        no2 = compress(x + start, len, in2_data, in2_count);
        for (i = j = k = 0; i < no1 && j < no2; =     /*merge */
            if (in1_data[i] < in2_data[j]= {
                out_data[k]  = in1_data[i];
                out_count[k] = in1_count[i];
                i++, k++;
            }
            else if (in1_data[i] > in2_data[j]) {
                out_data[k]  = in2_data[j];
                out_count[k] = in2_count[j];
                j++, k++;
            }
            else {      /* if equal, update count              */
                out_data[k]  = in1_data[i];
                out_count[k] = in1_count[i] + in2_count[j];
```

Download at http://www.pin5i.com/


```
                i++, j++, k++;
            }
            for (; i < no1; i++, k++= {    /* the copy part          */
                out_data[k]  = in1_data[i];
                out_count[k] = in1_count[i];
            }
            for (; j < no2; j++, k++= {
                out_data[k]  = in2_data[j];
                out_count[k] = in2_count[j];
            }
            no1 = k;                      /* k contains # of output  */
            SWAP(in1_data,  out_data);  /* let out be in1            */
            SWAP(in1_count, out_count); /* for next run              */
        }
        return no1;
    }

    /* ----------------------------------------------                */
    /* FUNCTION :                                                    */
    /*   Given a segment of an sorted array, this function           */
    /* extracts those distinct elements with its count.              */
    /* ------------------------------------------------------------  */

    int  compress(int x[], int n, int *data, int *count)
    {
        int  i;
        int  no = 0;

        if (n == 0)
            return 0;
        else {
            data[0]  = x[0];
            count[0] = 1;
            for (i = 1; i < n; i++)
                if (x[i] == x[i-1])
                    count[no]++;
                else {
                    data[++no] = x[i];
                    count[no]  = 1;
                }
            no++;
            return no;
        }
```

Download at http://www.pin5i.com/


```
}
/* ----------------------------------------------------------------- */
/* FUNCTION  LOG_expand :                                            */
/*   Given the data/count form (pointed by in1_data[] &              */
/* in1_count[], this function expands it into the original           */
/* form based on counting value.                                     */
/* ----------------------------------------------------------------- */

void  LOG_expand(int x[])
{
     int  total;
     int  i, j;

     for (total = i = 0; i < number; i++=
         for (j = 0; j < in1_count[i]; j++=
             x[total++] = in1_data[i];
}
```

问题 5.13　堆积式合并（HEAPMERG.C）

已知 m 个数组，每一个数组都有 n 个元素，为了方便起见，这 m 个数组正好是 x[ ][ ] 矩阵中的 m 列。如果这 m 列中的元素都各自按照从小到大的顺序排好了，请写一个程序把这 m·n 个元素合并到一个大矩阵去。不过在做这个题目时有进一步的限制，因为内存有限，大概最多只有与 m 成正比的内存可以使用。请问要如何克服这一层限制，当然程序的速度就会慢了。

【解答】

合并的操作，特别是从 m 个文件的 m 个数据中取出最小一个输出，这件事当然不容易操作，不但如此，若最小的值是从第 i 个文件来的，那么就要从第 i 个文件取出下一个数据来填充原来的位置。那么如何维护这 m 个数据，取出最小的值，再从该值所在的文件取出下一个来填充，这 3 件事有没有什么快速的做法呢？

前面讲堆积排序法的时候提过堆积（Heap）的观点，它最好的特性之一，就是取出极大值（或极小值，看如何安排数据）的成本是 0，因为它就在堆积的第一个位置。所以用堆积的话第一个问题就轻易地解决了。因此，一开始时从每一个文件各读一个数，连同文件的号码一并放到堆积中，然后让处理堆积的程序把堆积准备好，并且做到最小的元素在顶上。接着，从堆积顶上取出最小的元素把它输出，再从该元素所附着的文件号码从对应的文件取下一个元素放在顶上，但这样就不成为堆积了，所以再调用处理堆积的程序把这个元素压到下方合适的位置，同时在顶上又会出现新的一个最小的元素。于是就反复地进行这件工作，直到所有文件都没有元素为止。

Download at http://www.pin5i.com/

用这个概念，空间的要求就与 m（文件数目）成正比，因为要有一个数组做堆积，还要有另一个辅助数组记住每一个在堆积中的元素来自何处。用 m 个元素建立一个堆积最多只用与 m 成正比的比较次数，取出最小值不必比较，然而放入一个新元素而把它推到合用的位置却要与 $n/\log_2 m$ 成正比的比较次数。如果每个文件有 n 个元素，而一开始已经用了一个，因此就剩下 n-1 个元素，每一个元素都经过被取出再推到堆积中合适的位置的工作，所以就进行了 $m(n-1)\log_2 m$次，也就是大约与 $mn\log_2 m$ 成正比的比较次数，这可要比简单的合并方法多了 $\log_2 m$ 倍，但却省下了大量的内存。

在程序中并不用文件，而把文件看成是第一个 m 列矩阵中的一列。为了方便，假设每一列，也就是每一个文件的元素个数都相同，各有 n 个数据。正因为数据都在内存中，所以就要记录每一列的下一个元素何在。在程序中 index[ ]数组就是用来做这件事的，也就是说，index[i]就表示第 i 列中下一个要取出的元素在那一行。因为每一列的第一个元素会先被取出来，用来构成堆积，所以在开始时每一个 index[ ]的值都是 1。另外，heap[ ]与 location[ ]分别用来表示堆积，与堆积的元素来自的那一列；heap[ ]、location[ ]与 index[ ]都是外在变量，而且内存都通过 malloc()取得。file_no 与 file_len 也都是外在变量，记录着有多少个文件（列数）与每一个文件有多少元素（行数）。

heap_merge()是主控程序，它的动作很简单，先通过 prepare()取得堆积所需的内存，并且把堆积准备好，接着的 while 循环就每次取出堆积的顶端元素（用 delete_top()函数），delete_top()函数会返回顶端元素是在哪一列。当取出的元素的值是 INT_MAX 时就结束工作，如果不是 INT_MAX，就用 ask_data()从对应的列取出下一个元素，再用 insert()函数把它插入到堆积中。工作全部做完后，调用 release()函数把由 malloc()取得的内存交还给系统。

prepare()函数很简单，它用 malloc()函数为 heap[ ]、location[ ]、index[ ]取得内存，把第一行拿出来放到堆积中，然后通过 makeheap()把堆积建立起来。

ask_data()接收一个值表示列的号码，它先看看该列是否还有数据，如果有，就取出下一个作函数值，如果没有数据了，就返回 INT_MAX 这个值。

delet_top()其实并没有真的把堆积顶上的元素去掉，而不过是取出其值，以及指出哪个值来自哪一列的信息而已。为什么要这样做呢？原因是，delete_top()做完之后，一定会接着有一个 insert()的动作，把下一个元素安排到堆积中，如果在 insert()中把新数据放在顶端，不是正好把被剔除的元素盖过去了吗？所以做一次 siftdown()把元素推到适当的位置的操作就够了，而不必在 delete_top()时做一次，又在 insert()中做一次。注意，insert()中，把 row 的值存入 location[1]是不必要的，事实上 insert()根本连 row 这个参数都没必要，为什么？读者不妨自己想想。

makeheap()与 HEAPSORT.C 中的观点一样，建立一个堆积，而 siftdown ()则永远把第一个元素向下推到合适的位置，这两个都用到了 fix_heap()这个程序。fix_heap()是从 HEAPSORT.C 直接传过来的，把数组的名称与参数作了稍许变动，不过重要的是，在 HEAPSORT.C 中是把最大元素放到顶上，然后与最后一个元素交换，排出序列，但只需要最小的元素，所以比大小的方向全都相反。此外，程序完全没有变动，相信应该看得出来。

Download at http://www.pin5i.com/

【习题】

（1）为什么说 insert()函数的第一道叙述，把 row 的值存入 location [1]是多余的？

（2）请修改程序，把已排好顺序的一列改成一个已排好顺序的文件，并且把输出送到另一个文件去。

（3）假设内存中只能容纳 500 个数，而有一个内含 10 000 个数的文件，它并没有依顺序排列；请修改第（2）题的程序，在上述的限制下把文件中的顺序排好，放到一个输出文件中。可以考虑用随机数的方式来产生有 10 000 个数的文件。

【程序】

程序 14-14（HEAPMERG.C）

```
/* ------------------------------------------------------------ */
/* FUNCTION  heap_merge :                                       */
/*   Given file_no sorted sequences each with file_len          */
/* elements, this function merges these sequence into a         */
/* sorted sequence with a heap.                                 */
/*                                                              */
/* Copyright Ching-Kuang Shene      July/29/1989                */
/* ------------------------------------------------------------ */

#include <stdlib.h>      /* for malloc()                        */
#include <limits.h>      /* for INT_MAX                         */

#define  MAXSIZE   20    /* max of file_len, file_no            */
#define  YES 1
#define  NO  0

/* ------------------------------------------------------------ */
/*                      function prototypes                     */
/*
------------------------------------------------------------ */

void prepare(int [][MAXSIZE], int, int);    /* build heap       */
int  ask_data(int [][MAXSIZE], int);        /* ask for next     */
int  delete_top(int *);      /* delete the top-most item        */
void insert(int, int);       /* insert new item to top          */
void release(void);          /* release allocated memory        */
void makeheap(void);         /* build the heap                  */
void siftdown(void);         /* sift the top-most item          */
void fix_heap(int);          /* actual working heap RTN         */

/* ------------------------------------------------------------ */
```

Download at http://www.pin5i.com/


```
/*                          heap data structure                          */
/* ------------------------------------------------------------------    */

int  *heap;                        /* pointer to heap                   */
int  *location;                    /* pointer to correspond loc         */
int  *index;                       /* next item in sequence             */
int  file_no;                      /* no of sequences                   */
int  file_len;                     /* no of elements in seq.            */

/* ------------------------------------------------------------------    */
/* FUNCTION heap_merge :                                                 */
/*   The control routine.  It accepts matrix[] and its                   */
/* actual row and column numbers, merges them and then                   */
/* send the merged sequence to out[] array.                              */
/* ------------------------------------------------------------------    */

void  heap_merge(int matrix[][MAXSIZE], int m, int n, int out[])
{
     int  out_no;
     int  row;
     int  temp;

     prepare(matrix, m, n);       /* prepare the initial heap           */
     out_no = 0;                  /* # of output = 0                    */
     while ((temp = delete_top(&row)) != INT_MAX) {
         out[out_no++] = temp;    /* always output the top              */
         insert(ask_data(matrix, row), row);    /* ins. new            */
     }
     release();                   /* release all memeory                */
}

/* ------------------------------------------------------------------    */
/* FUNCTION  prepare :                                                   */
/*   This function allocates memory for the heap[],                      */
/* location[] and index[] arrays and then build up the                   */
/* initial heap with the first elements from each seq.                   */
/* ------------------------------------------------------------------    */

void  prepare(int matrix[][MAXSIZE], int m, int n)
{
     int  i;
```

Download at http://www.pin5i.com/

```
heap     = (int *) malloc(sizeof(int)*(m+1));
location = (int *) malloc(sizeof(int)*(m+1));
index    = (int *) malloc(sizeof(int)*m);
file_no  = m;
file_len = n;
for (i = 1; i <= file_no; i++) {    /* use 1st item          */
    heap[i]     = matrix[i-1][0];   /* to build up the       */
    location[i] = i - 1;            /* working heap.         */
    index[i-1]  = 1;        /* next item pointer to seq.     */
  }
  makeheap();                       /* build up the heap.    */
}

/* ----------------------------------------------------------- */
/* FUNCTION ask_data :                                         */
/*   This routine returns the next element in the seq.         */
/* indicated by parameter 'row'.  In case of the seq being     */
/* empty, a maximum value is returned.                         */
/* ----------------------------------------------------------- */

int  ask_data(int matrix[][MAXSIZE], int row)
{
    if (index[row] >= file_len)
        return  INT_MAX;
    else
        return  matrix[row][index[row]++];
}

/* ----------------------------------------------------------- */
/* FUNCTION delete_top :                                       */
/*   This routine deletes the top-most element of the          */
/* heap, but does not make any re-heap operation because       */
/* the next operation must be an insert.                       */
/* ----------------------------------------------------------- */

int  delete_top(int *row)
{
    *row = location[1];
    return heap[1];
}

/* ----------------------------------------------------------- */
```

Download at http://www.pin5i.com/


```
/* FUNCTION  insert :                                                  */
/*   This routine inserts a data 'data' located in the.                */
/* 'row'-th sequence into the top-most element of the heap.            */
/* ------------------------------------------------------------------  */

void  insert(int data, int row)
{
     location[1] = row;
     heap[1]     = data;
     siftdown();
}

/* ------------------------------------------------------------------  */
/* FUNCTION  release :                                                 */
/This routine release all allocated memeory.                           */
/* ------------------------------------------------------------------  */

void  release(void)
{
     free(heap);
     free(location);
     free(index);
}

/* ------------------------------------------------------------------  */
/* FUNCTION  makeheap :                                                */
/*   This routine builds up a heap with the smallest item              */
/* being on the top-most position.                                     */
/* ------------------------------------------------------------------  */

void  makeheap(void)
{
     int  temp;
     int  size, i;

     for (i = file_no/2; i >= 1; i--) /* starts from mid               */
         fix_heap(i);                 /* fix the heap at this          */
}

/* ------------------------------------------------------------------  */
/* FUNCTION  siftdown :                                                */
/*   This routine sift the top-most item, which is put                 */
```

Download at http://www.pin5i.com/


```
/* there by insert(), down to an appropriate position.          */
/* ------------------------------------------------------------ */

void  siftdown(void)
{
     fix_heap(1);
}

/* ------------------------------------------------------------ */
/* FUNCTION fix_heap :                                          */
/*   Given a heap rooted at 'root', this routine sifts          */
/* that element down to an appropriate position so that         */
/* the result is still a heap.  NOTE that the smallest          */
/* element is on the top of each sub-heap.                      */
/*                                                              */
/* ------------------------------------------------------------ */

void  fix_heap(int root)
{
     int  father, son;
     int  key_heap = heap[root];
     int  key_loc  = location[root];
     int  done;

     father = root;              /* start off from the root      */
     son    = father + father;   /* get the subscript of son     */
     done   = NO;
     while (son <= file_no && !done) {  /* son available?        */
         if (son < file_no && heap[son+1] < heap[son])
             son++;              /* now son is the larger one    */
         if (key_heap > heap[son]) {/* if the son > key          */
             heap[father] = heap[son]; /* step down              */
             location[father] = location[son];
             father    = son;
         }
         else                    /* son < key, thus the key      */
             done      = YES;    /* get its position.            */
         son = father + father;
     }
     heap[father] = key_heap;    /* insert the key               */
     location[father] = key_loc;
}
```

Download at http://www.pin5i.com/

问题 5.14 检查数组元素是否相异（UNIQUE.C）

已知一个数组，请写一个程序把相同的元素去掉而只要下一个。

【解答】

理论证明，最快的方法是把给定的数组从大到小排好，然后从第一个元素开始作比较，把相同的元素取一个出来。例如，假设元素已经从大到小排序的数组是 x[ ]，自然 x[0]是一定存在的，就令 i 为 1～n-1，每次比较 x[i]与 x[i-1]，如果相同就略过而继续比较下一组，但若不同，就把 x[i]收集起来，当 n 个元素比完之后，所收集起来的就全是不同的元素了。因为要排大小，所以要用到排大小的程序，可以用本书中任何一种方法的程序来做这件事。unique()才是关键部分，它把给定的数组依小到大排好，再用上述方法把重复的元素去掉。

这个方法会快多少？如此复杂的程序（与本题说明中的方法相比）真的会快吗？写如此复杂的程序值得吗？理论上，快速排列法平均要作 $\alpha n\log_2 n$ 次比较（α 是一个与 n 无关的常数），排完之后还要有 n-1 次比较才能过滤掉重复的部分，所以 $\alpha n\log_2 n+n-1$ 次比较就足够了，当 n 比较大的时候，比较次数就会小于 n(n-1)/2，用随机数的方式产生 1000,2000,3000,4000,5000 笔数据，用这两个方法各做一次，时间值如表 14-1 所示。

表 14-1

n	简单做法	本题程序
1000	9.92	0.95
2000	36.02	1.01
3000	80.97	1.97
4000	139.01	2.94
5000	213.97	3.02

从表 14-1 就可以看到时间的差距有多大了，所以复杂的程序不一定慢，简单的程序不一定快，快慢之差在于使用的方法。很多题目经过排序后都会得到比较快的程序。

【习题】

（1）快速排列法并不是最快的方法，可以改用堆积法（Heap Sort）或更快（通常只能排整数）的“桶子法”（Bucket Sort），然后比较一下速度的差异。

（2）这个程序的输出会依顺序排好，而破坏了原来输入的顺序，请修改这个程序，使其可以依输入的顺序出现，但却不降低程序的效率。

（3）有一个类似的题目是：给出一个数组，请问其中的元素是否全不相同。请用本题程序作蓝本，修改成解决此问题的程序，再用本题说明中的方法来处理，请问处理速度上有什么差异？本题还是比较快吗？注意，理论上已经证明，本题的做法还是快，虽然题目已经被化简了，最好读者自己验证一番。

Download at http://www.pin5i.com/

（4）如果允许使用额外的内存（比如变量、数组），有没有办法把第（3）题做得更快？

【程序】

程序 14-15（UNIQUE.C）

```
/* ------------------------------------------------------------------ */
/* FUNCTION  unique :                                                 */
/*   Given an array of integers, this functions packs                 */
/* this array with all distinct elements.  NOTE that it               */
/* uses sorting functions and thus you can use any one                */
/* sort program in my book.  Don't misunderstand with the             */
/* conventional technique that this program can be done               */
/* efficiently with simple-minded method.  Actually, the              */
/* most efficient technique (of course, it can be proven              */
/* formally) is by using sorting.                                     */
/*                                                                    */
/* Copyright Ching-Kuang Shene      July/22/1989                      */
/* ------------------------------------------------------------------ */

void  sort(int [], int); /* for sort()                                */
void  unique(int x[], int n, int *number)
{
     int  i;

     sort(x, n);                /* sort the data first               */
     *number = 1;               /* the 1st must be presented         */
     for (i = 1; i < n; i++) /* process the 2nd to last              */
         if (x[i] != x[i-1])     /* if it is NOT a duplicate:         */
             x[(*number)++] = x[i]; /* save it.                       */
}
```

问题 5.15　数组中和为零的段落（ZEROSUM.C）

已知一个整数数组，其中可能有正整数、负整数与 0，而且元素也不会重复出现，请写一个程序找出这个数组中是否有一个子数组元素的和为 0。所谓的子数组，就是原来数组中连续的若干（至少 1）个元素。例如，如果数组为 1,2,3,-5,4，那么 2,3,-5 这个子数组中元素的和就是 0。

【解答】

如果数组的 n 个元素写成 $x_1, x_2, \cdots, x_n$。若 $x_i + x_{i+1} + \cdots + x_j$ 为 0，考虑 $x_1 + x_2 + \cdots + x_j$ 与 $x_1 + x_2 + \cdots + x_i$。

Download at http://www.pin5i.com/

$$\sum_{k=1}^{j} x_k = \sum_{k=1}^{i-1} x_k + (x_i + x_{i+1} + \cdots + x_j)$$
$$= \sum_{k=1}^{i-1} x_k + 0$$
$$= \sum_{k=0}^{i-1} x_k$$

换句话说，如果 $x_i + x_{i+1} + \cdots + x_j = 0$，那么 $\sum_{k=1}^{j} x_k$（前 j 项的和）与 $\sum_{k=1}^{i-1} x_k$（前 i－1 项的和）就会相等。把前 k 项的和加出来，存放到一个新数组 y_k 中（用原数组也行，不过原始数据就被毁了），如果在 $y_1, y_2, \cdots, y_n$ 中有两个元素的值相同，假设是 y_i 与 y_j，若 $i < j$，那么原来数组中 $x_{i+1}, x_{i+2}, \cdots, x_j$ 的和就是 0。

如何找出 $y_1, y_2, \cdots, y_n$ 中有没有相同的元素呢？在前面的问题中就有这样的说明；最有效的方法就是把 $y_1, y_2, \cdots, y_n$ 排好顺序，然后再检查相邻的元素是否相同就可以了。假设用这个做法，如果排顺序用比较快速的方法，就不会超过 $\alpha n \log_2 n$ 次比较（α 是一个与元素个数 n 无关的常数），最后看相邻的元素是否相同，n－1 次比较就够了，因而总共用了 $\alpha n \log_2 n + n - 1$ 次比较而已，要比 n(n－1)/2 少，特别是 n 比较大的时候（参看上一个问题的解答中的比较）。

例如，输入的数组为 4,－1,2,1,－2,－1,5；把各个前项加起来就有 4,3,5,6,4,3,8；从小到大排列则得到 3,3,4,4,5,6,8，有相同的元素，有子数组的和为 0。事实上，第一个 3 是前两项的和，第二个 3 是前 6 项的和，亦即 $x_1 + x_2 = 3$，$x_1 + x_2 + x_3 + x_4 + x_5 + x_6 = 3$，因此 $x_3 + x_4 + x_5 + x_6 = 0$，这就是答案之一；同理可以从两个 4 推出另一个答案。

注意，此处没有 0 元素，有 0 元素时找出和为 0 的子数组的工作就会变得复杂些。

【习题】

（1）这个程序只找出有没有一个子数组的和为 0 而已，而没有把这个子数组的脚码找出来，请修改程序，显示出和为 0 的子数组的两个端点的脚码。

（2）请再修改程序，把所有和为 0 的子数组全找出来。分析一下程序一共作了多少次比较，会不会比原来程序的效率低？

【程序】

程序 14-16（ZEROSUM.C）

```
/* ------------------------------------------------------------ */
/* FUNCTION  zero_sum :                                         */
/*   Given an integer array, this function checks to see        */
/* if there is a subarray summing up to zero.  NOTE that        */
/* sorting is still the most efficient way to solve this        */
/* problem.                                                     */
/* ------------------------------------------------------------ */
```

Download at http://www.pin5i.com/


```
#define  YES 1
#define  NO  0

void  sort(int [], int);

int  zero_sum(int x[], int n)
{
    int  i;

    for (i = 1; i < n; i++)          /* compute prefix sum       */
        x[i] += x[i-1];
    sort(x, n);                      /* then sort these sums     */
    for (i = 1; i < n && x[i] != x[i-1]; i++)  /* unique?        */
        ;
    return (i == n) ? NO : YES;      /* if not unique, YES.      */
}
```

问题 5.16 平面上的极大点（MAXSET.C）

在平面上如果有两个点(x,y)与(a,b)，说(x,y)支配（Dominate）了(a,b)，这就是指 x≥a 而且 y≥b；用图来看就是(a,b)座落在以(x,y)为右上角的一个无限的区域中。对于平面上的任意一个有限点集合而言，一定存在若干个点，它们不会被集合中任何一个点所支配，这些个点就构成一个所谓的极大集合。请写一个程序，读入一个新的集合（以(x,y)的坐标形式），找出这个集合中的极大集合。

【解答】

如图 14-14 所示为一个点集合。

图 14-14 中一共有 11 个点，粗点就是极大集合中的元素，看看这些个极大集合中的元素究竟有什么特性。第一，就是最右边与最高的点一定在极大集合中，为什么？道理很简单，因为在它右边与上方不再有元素，自然就不会有点支配它，在最高点的点也一定在极大集合中，道理同前。

第二点就不那么容易看出来了。先不要往下看，先看 4 个极大集合中的元素，如果 4 个点不够，可以多用几个点看看，是否发现它们彼此之间有什么关系。它们的 y 坐标从右往左会愈来愈大，一直到最高点为止，在最高点的地方（也就是左边）就不会再有极大集合中的元素了。为什么？这个现象的证明是很简单的，例如，(x,y)与(a,b)是两个极大集合中相邻的元素（注意，在原来集合中它们不一定相邻，图 14-14 中的中间两个极大集合元素就是一个好例子）；如果(x,y)在右边，(a,b)在左边，亦即 a≤x；因为(a,b)在极大集合中，亦即没有点可以支配它，所以非得 b≥y 不可，若 b≤y 的话，(x,y)就支配了(a,b)，从而(a,b)就不可能是极大集合中的元素了。因此得到一个结论，从右往左看，极大集合中

Download at http://www.pin5i.com/

的元素的 y 坐标是越来越大的。

不但如此，如果(x,y)与(a,b)是极大集合中的元素，并且(x,y)在右，(a,b)在左，于是在原来集合中的任何一点(r,t)，若 a≤r≤t（亦即在(a,b)与(x,y)之间），那么会得到 t≤y，如图 14-15 所示。

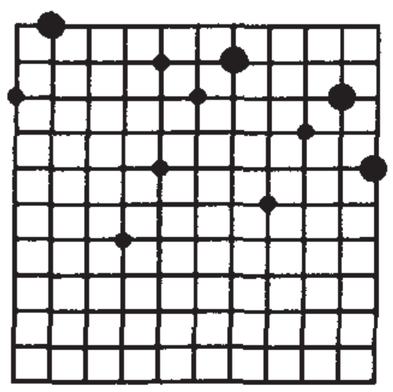


图 14-14

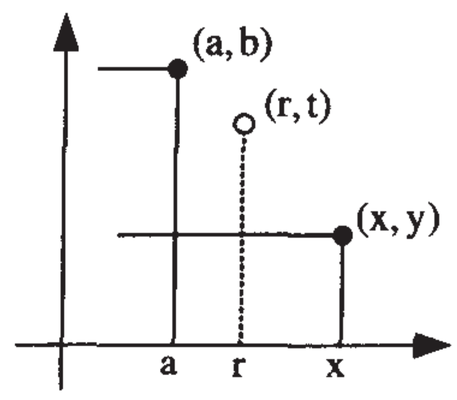


图 14-15

如果 t 比 y 大，如图 14-15 所示，于是在 a 与 x 之间，(r,t)就是另一个支配点，这就与上面说的(a,b)与(x,y)在极大集合中相邻的假设矛盾了。

综合起来，程序并不难写；从右边向左边处理，最右边的点一定在极大集合中，而且记住它的 y 坐标。当从右向左处理的过程中，哪些点的 y 坐标比记住的小，那当然就不会是个支配其他点的点。然而一旦发现某个点的 y 坐标比记住的 y 坐标大了，那个点就一定是一个支配点，而且与上个找出来的支配点相邻。在极大集合中的点的 y 坐标越来越大，因此下一个（如果有的话）在极大集合中的元素的 y 坐标会更大，于是就记住目前最大的 y 坐标值。等到所有点都处理完了，极大集合也就找出来了。

不过，要如何从右向左处理呢？在计算机中就要把输入的点的 x 坐标从小到大排好（当然同时也调动对应的 y 坐标），然后从 x[n-1],x[n-2],…,x[0]这样处理即可。

来看看程序 MAXSET.C。max_set()函数就是主控程序，不过 8 列而已。首先，用 sort()函数把 x[ ]从小到大排好，然后令 i 自 n-1 起往 0 的方向处理，max 存放到目前为止的 y 坐标极大值；在 for 循环中每次都看看 y[i]是否比记住的 max 值大。如果是，依上所述，x[i]与 y[i]就在极大集合中，所以把这个点传到用来储存极大集合的 xx[ ]与 yy[ ]数组去，并且把计数用的变量（no）加 1，最后把 max 换成到目前为止所看到过的极大值，所以，当 for 循环结束了，xx[ ]与 yy[ ]就是极大集合中的点的两个坐标，这些点的个数则在 no 中。

此 sort()函数采用快速排列法。注意，现在多了一个参数 y[ ]，因为在排列时要移动 x[ ]的位置，也要互换 y[ ]，除此之外，程序十分简单。

这个程序一共作了多少次比较呢？在排序上用了与 $n\log_2 n$ 成正比的比较次数，而在 for 循环中每一个点都恰好比较一次，一共 n 次，n 是点的数目，因此就总共有与 $n\log_2 n + n$ 成正比的次数，不过因为 $n\log_2 n$ 比 n 大，去掉 n 并没有影响，所以就是与 $n\log_2 n$ 成正比的比较次数，这可是比问题说明中的 n^2 次快很多的。

【习题】

（1）为什么在最高点的左边就不会再有极大集合中的元素？

（2）如果有两个点(x,y)与(a,b)，若(x,y)支配(a,b)，即(a,b)支撑（Support）着(x,y)。在一个平面的点集合中，总是会有一些点会支撑其他点，但却不被其他点所支撑的，这些

Download at http://www.pin5i.com/

点就构成一个极小集合，请写一个程序找出平面上一组点的极小集合。这个程序与例子有什么差异？

（3）如果已知的点是在三维空间中，请问能否修改例题程序来找出极大集合？例题中究竟有什么地方是不适用于三维空间的？

（4）如果给定的点的坐标都是整数值，有什么办法加快程序的执行速度？（提示：不要用快速排列法，而使用其他方式，因为令程序有 $n\log_2 n$ 比较次数的正是排序）

【参考文献】

此讨论节自孔祥重（H.T.Kung）教授在 1975 年的一篇文章，那篇文章还有其他情况的讨论。另外，M.J.Atallah 与 S.R.Kosaraju 在 1989 年提出了另一个方法，不但可以找出极大集合，还可以找出在输入的集合中的每一个点支配了哪些点，重要的是比较的次数仍然与 $n\log_2 n$ 成正比。

1 M.J.Atallah and S.R.Kosaraju. An Efficient Algorithm for Maxdominance, with Applications, *Algorithmica*. Vol.4(1989), pp.221～236

2 H.T.Kung, F.Luccio and F.P.Preparata. On Finding the Maxima of a Set of Vectors, *Journal of ACM*. Vol.22(1975), pp.469～476

【程序】

程序 14-17（MAXSET.C）

```
/* ------------------------------------------------------------ */
/* FUNCTION maxset :                                            */
/*   Given a set of points on the plane, this function          */
/* computes all those maximal points.  A point (x,y) is         */
/* said to be maximal, if and only if for all other point       */
/* (a,b)we have a<=x, b<=y, or a<=x, b>y, or a>x, b<=y.         */
/*                                                              */
/* Copyright Ching-Kuang Shene      July/28/1989                */
/* ------------------------------------------------------------ */

#include <limits.h>

/* ------------------------------------------------------------ */
/*                        function prototypes                   */
/* ------------------------------------------------------------ */

void sort(int [], int [], int);
void qsort(int [], int [], int, int);
void split(int [], int [], int, int, int *);
void swap(int *, int *);
```

Download at http://www.pin5i.com/


```
void  max_set(int x[], int y[], int n, int xx[], int yy[], int *no)
{
     int  max, i;

     sort(x, y, n);                /* sort the x-coordinate        */
     for (i = n-1, *no = 0, max=INT_MIN; i >= 0; i--)
         if (y[i] > max) {         /* process downward.            */
             xx[*no] = x[i];       /* if y[i]>current max, then    */
             yy[*no] = y[i];       /* y[i] is a newly found max    */
             (*no)++;              /* point.  Thus save it and     */
             max = y[i];           /* update current max.          */
     }
}

/* ----------------------------------------------------------------- */
/* FUNCTION sort :                                                   */
/*   This routine converts two arguments to three                    */
/* arguments and calls the REAL control routine.                     */
/* ----------------------------------------------------------------- */

void sort(int x[], int y[], int n)
{
     int  first = 0;               /* the fisrt element.           */
     int  last  = n-1;             /* the last element.            */

     qsort(x, y, first, last);
}
/* ----------------------------------------------------------------- */
/* FUNCTION  qsort :                                                 */
/*   This routine accepts two arrays and two indices                 */
/* indicating the first and the last element to be sorted.           */
/* Then calls split() to split the first array into two              */
/* parts and sorts them recursively.                                 */
/* ----------------------------------------------------------------- */

void  qsort(int x[], int y[], int first, int last)
{
     int  split_pt;

     if (first < last= {
         split(x, y, first, last, &split_pt);
         if (split_pt - first < last - split_pt= {
```

Download at http://www.pin5i.com/


```
        qsort(x, y, first, split_pt-1);
        qsort(x, y, split_pt+1, last);
    }
    else {
        qsort(x, y, split_pt+1, last);
        qsort(x, y, first, split_pt-1);
    }
    }
}


/* ----------------------------------------------------------- */
/* FUNCTION  split :                                           */
/*   This routine splits the given array into two parts.       */
/* ----------------------------------------------------------- */

void  split(int x[], int y[], int first, int last, int *split_pt)
{
    int  current_split,  unknown;
    int  xx;

    xx = x[first];
    current_split = first;
    for (unknown = first+1; unknown <= last; unknown++=
        if (x[unknown] < xx= {
            (current_split)++;
            swap(&x[current_split], &x[unknown]);
            swap(&y[current_split], &y[unknown]);
        }
    swap(&x[first], &x[current_split]);
    swap(&y[first], &y[current_split]);
    *split_pt = current_split;
}

/* ----------------------------------------------------------- */
/* FUNCTION  swap :                                            */
/*   This is the simplest routine which swaps two data.        */
/* ----------------------------------------------------------- */
void  swap(int *p, int *q)
{
    int temp;
```

Download at http://www.pin5i.com/


```
        temp    = *p;
        *p  = *q;
        *q  = temp;
    }
```

问题 5.17　宴会中访问数目的极大值（MAXVISIT.C）

在一个宴会中一共有 n 位来宾，依照来宾的到达与离去的登记时间，知道第 i 位来宾在 x_i 时到达，在 y_i 时离开，因此第 i 位来宾在宴会场中的时间是 $[x_i, y_i)$，亦即 $x_i \leqslant t \leqslant y_i$ 中所有可能的 t，请写一个程序，读入 x_i 与 y_i，$1 \leqslant i \leqslant n$，找出同一时刻之内最多会有多少人同时在宴会场中。

【解答】

题目极其简单。怎么知道在时间 t 时有多少人留在会场中？大多数人都知道一个简单方法，如果某人在 t 之前或在 t 时离开，他一定不在会场，而某人到达时间在 t 之后，他也不在会场。所以只有到达时间在 t 之前（含 t）、离开时间在 t 之后（不含 t）的人才算是在会场中。但是，为了要做到这一点，人们的到达与离开的时间要以时钟的顺序来处理才行，也就是说时间 t_1 某甲来了，会场多一人，时间 t_2 某乙走了，会场少一人，……，所以首要的事就是排列所有的（到达与离开）时间值。

在后面的程序 MAXVISIT.C 中，max_visitors()是主控程序，交给它的 x[ ]与 y[ ]就是时间值，x[i]与 y[i]表示 i 在 x[i]时到达，在 y[i]离开。为了要排列所有的时间，要一个时间表，它有 2·n 个元素（x[ ]与 y[ ]各有 n 个），此外还要多一栏，指出这是到达还是离开的时间。time_table 就是指向这个时间表的指针，时间表的内存是通过 malloc()函数取得的；time_table 有两个字段，clock 是时间（到达或离开），而 status 则是 IN（表示对应的是到达时间）或 OUT（表示对应的是离开时间）。程序中前两个循环就把 x[ ]与 y[ ]的值填入时间表中。

不过，这样填完的时间表并没有依时间先后顺序排好，接下来进行排序。sort()就是做这件事的，采用快速排列法，并且作适当的修改。注意，排顺序时不能单是把时间值小的排在前面，大的排在后面，因为这可能会引起困扰。例如，甲在 t 时间到达，乙在 t 时间离开，如果经过排列之后甲排在乙之前，对于时间 t 就会先碰到甲，他到达了，会场人数多了 1，因此很可能就会得到错误的结果（为什么？）。所以说，在排序时，如果时间表中有两笔数据时间值相等时，要让 status 是 OUT 亦即离开的数据排在前面。于是上面的例子中乙会排在甲之前，在处理到该时间时，先去掉乙，再加上甲，这就是正确的结果了（为什么？）。在排序完成之后，就把时间表从头查起，并且准备一个计数用的变量 count。当第 i 时间有人离开时，count-1，因为会场中少了一个人；当此时有人到达，count+1，再修正极大值。到所有时间都查完了，最大的计数值不正好就是会场中人数最多时的值吗？

排序的部分可参考快速排列法的说明，惟一值得提醒的地方是在 split()函数中，比大小的操作不单是看 clock 值的大小，而且当两者的 clock 的值相同时，具有 OUT（离开）的时

Download at http://www.pin5i.com/

间要排在前面（也就是比较小），其他的部分可参看程序与快速排列法。

【习题】

（1）为什么时间相等时，离开的时间要排在前面？请举出一个不这样做时就会发生错误的例子。

（2）请问在 max_visitors()中为什么用 if…else 来测试 status 是否为 OUT？用 switch 语句加上 case IN 与 case OUT 会不会有相同的结果？为什么？（提示：不要执行，试用 ANSI C 的标准来探讨结果）

（3）程序只找出最多有多少人。请修改程序，找出所有有那么多人在会场中的时间。

（4）再修改习题（3）的程序，找出在那个时间中有哪些人在会场内。

（5）如果已知数据中，到达时间是从小到大排好的，有没有比较简单而不必排序的方法呢？提示：因为 y[ ]没有顺序，可能会后来的先走，请试试看把离开时间放到堆积（Heap）中，程序会不会快一些？作了多少次比较？

【程序】

程序 14-18（MAXVISIT.C）

```
/* ------------------------------------------------------------ */
/* FUNCTION  max_visitors :                                     */
/*   Given a time table of arrival/departure data, this         */
/* function computes the maximum number of visitors in a        */
/* party at the same time.                                      */
/*                                                              */
/* Copyright Ching-Kuang Shene      Aug/11/1989                 */
/* ------------------------------------------------------------ */

#include <stdlib.h>     /* for malloc()                         */

#define  IN  1        /* this is an ARRIVAL time                */
#define  OUT 0        /* this is an DEPART. time                */
struct   table {      /* a table contains mixed                 */
    int  clock;       /* timing data with IN/OUT                */
    int  tatus;       /* marking by 'status' field              */
};

typedef  struct table  TABLE;    /* table type                   */

/* ------------------------------------------------------------ */
/*                          function prototypes                 */
/* ------------------------------------------------------------ */

void  sort(TABLE [], int);
```

Download at http://www.pin5i.com/


```
void  qsort(TABLE [], int, int);
void  split(TABLE [], int, int, int *);
void  swap(TABLE *, TABLE *);

int  max_visitors(int x[], int y[], int n)
{
     TABLE *time_table;  /* working time table addr                  */
     int  max_count;     /* max. count                               */
     int  count;
     int  i;

     time_table = (TABLE *) malloc(sizeof(TABLE)*(2*n));

     for (i = 0; i < n; i++= {      /* move time to work table */
         (time_table+i)->clock  = x[i];
         (time_table+i)->status = IN;   /* ARRAIVAL time        */
     }
     for (i = 0; i < n; i++= {      /* move departure time      */
         (time_table+i+n)->clock  = y[i];
         (time_table+i+n)->status = OUT;
     }
     sort(time_table, 2*n);          /* sort this time table    */

     for (max_count = count = i = 0; i < 2*n; i++=
         if ((time_table + i)->status == OUT) /* OUT?            */
             count--;                /* visitors decrease by 1   */
         else {                      /* IN !                     */
             count++;                /* visitors increase by 1   */
             max_count = (max_count < count= ? count : max_count;
         }
     free(time_table);
     return max_count;
}

/* ----------------------------------------------------------------- */
/* FUNCTION sort :                                                   */
/*   This routine converts two arguments to three                    */
/* arguments and calls the REAL control routine.                     */
/* ----------------------------------------------------------------- */
void sort(TABLE x[], int n)
{
     int  first = 0;                 /* the fisrt element.       */
     int  last  = n-1;               /* the last element.        */
```

Download at http://www.pin5i.com/


```
    qsort(x, first, last);
}

/* ----------------------------------------------------------------- */
/* FUNCTION qsort :                                                  */
/*   This routine accepts an array and two indices                   */
/* indicating the first and the last element to be sorted.           */
/* Then calls split() to split the array into two parts              */
/* and sorts them recursively.                                       */
/* ----------------------------------------------------------------- */

void qsort(TABLE x[], int first, int last)
{
    int split_point;

    if (first < last= {
        split(x, first, last, &split_point);
        if (split_point - first < last - split_point= {
            qsort(x, first, split_point-1);
            qsort(x, split_point+1, last);
        }
        else {
            qsort(x, split_point+1, last);
            qsort(x, first, split_point-1);
        }
    }
}

/* ----------------------------------------------------------------- */
/* FUNCTION split :                                                  */
/*   This routine splits the given array into two parts              */
/* according to Nico Lomuto's aplitting technique and                */
/* returns the splitting index with 'split_point'.                   */
/* ----------------------------------------------------------------- */

void split(TABLE x[], int first, int last, int *split_point)
{
    int current_split, next;
    int temp;

    temp = x[first].clock;
    current_split = first;
    for (next = first+1; next <= last; next++)
```

Download at http://www.pin5i.com/


```
        if (x[next].clock < temp ||
            (x[next].clock == temp && x[next].status == OUT)) {
                (current_split)++;
                swap(&x[current_split], &x[next]);
        }
    swap(&x[first], &x[current_split]);
    *split_point = current_split;
}


/* ----------------------------------------------------------        */
/* FUNCTION  swap :                                                  */
/*   This is the simplest routine which swaps two data.              */
/* ----------------------------------------------------------        */

void  swap(TABLE *p, TABLE *q)
{
    int  temp;

    temp     = p->clock;
    p->clock     = q->clock;
    q->clock     = temp;

    temp     = p->status;
    p->status    = q->status;
    q->status    = temp;
}
```

问题 5.18　包含在其他区间中的区间（CONTAIN.C）

已知一组区间$[a_i, b_i]$，i=1,2,…,n；请写一个程序，找出这些区间中有哪一些会包含在其他的某个区间中。

【解答】

此题目难以下手的原因，其实就是每一个区间都有两个端点，处理这个同时就不好处理另一个了，不过当做过了问题 5.17 之后，处理这一题就应该会有一些心得才对。

首先，假设所有区间的左边与右边的端点都不相同，这样会容易些，然后再回头取消这一层假设。若$[a_i, b_i]$与$[a_j, b_j]$是任意两个区间，而且$a_i < a_j$，那么b_i与b_j之间不外乎下面两种情况：

Download at http://www.pin5i.com/

（1）$b_i < b_j$：这种情况下，$[a_i,b_i]$既不包含$[a_j,b_j]$，又不会被$[a_j,b_j]$所包含。为什么呢？因为a_i在a_j的左边，b_i也在b_j的左边，那么$[a_i,b_i]$如果不是与$[a_j,b_j]$相互分离，（如图 14-16 所示），就是两者有共同的部分但并没有包含的关系，如图 14-17 所示。

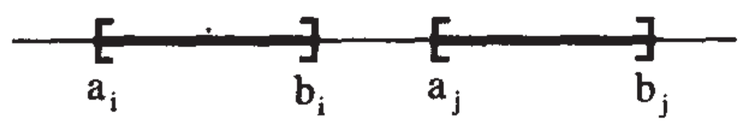


图 14-16

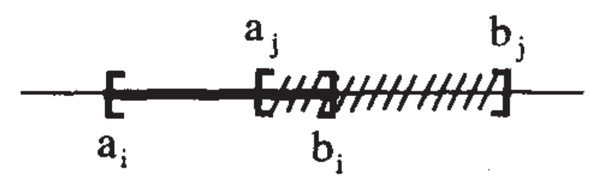


图 14-17

（2）$b_i > b_j$：因为$a_i < a_j$且$b_i > b_j$，于是$[a_j,b_j]$就一定包含在$[a_i,b_i]$中。

正因为如此，可以用区间左边端点的元素排出顺序，也就是说，若$[a_i,b_i]$与$[a_j,b_j]$中有$a_i < a_j$，那就把$[a_i,b_i]$排在$[a_j,b_j]$前面。排好了之后，就看b_1，如果有一个b_i是落在b_1左边（或等于b_1），那么由于$a_1 < a_i$，若依上面的说法，$[a_i,b_i]$就一定会包含在$[a_1,b_1]$中。从b_2起，$b_3,b_4,\cdots$一个接一个地依排列好的顺序查过去，如果$b_i \leqslant b_1$，就报告$[a_i,b_i]$包含在某个区间（$[a_1,b_1]$）中，一直到某个b_k满足$b_k > b_1$，那怎么办呢？记住一点，到目前为止，$b_1,b_2,\cdots b_{k-1}$都小于或等于b_1，而b_k是第一个比b_1大的右边端点；但是排列大小的顺序中又有$a_1 < a_k$，所以$[a_k,b_k]$就不会包含在$[a_1,b_1]$中，当然也就不会包含在$[a_2,b_2],\cdots,[a_{k-1},b_{k-1}]$这个区间中了。但会不会包含在$[a_{k+1},b_{k+1}],\cdots,[a_2,b_2]$其中之一呢？不会，理由就是$a_k < a_{k+1} < \cdots < a_n$，这是排大小的顺序，所以$[a_{k+1},b_{k+1}] \sim [a_2,b_2]$不可能包含$[a_k,b_k]$，于是就只要查一查$[a_k,b_k]$会包含$[a_{k+1},b_{k+1}],\cdots,[a_2,b_2]$就行了。因此工作又回到开始时$[a_1,b_1]$的状态：从$b_{k-1}$查起，依$b_{k+2},\cdots,b_n$顺序处理，一直到$b_p > b_k$为止，$[a_{k+1},b_{k+1}],\cdots,[a_{p-1},b_{p-1}]$都包含在$[a_k,b_k]$中，而$[a_p,b_p]$则不然，因此就一直重复这个操作，直到所有区间都处理完为止。

上面的讨论是以$a_1 \neq a_j$为前提而进行的，若有两个区间的左边终点相等时该怎么办呢？那就是$[a_i,b_i]$与$[a_j,b_j]$中$a_i = a_j$的情况，如图 14-18 所示。

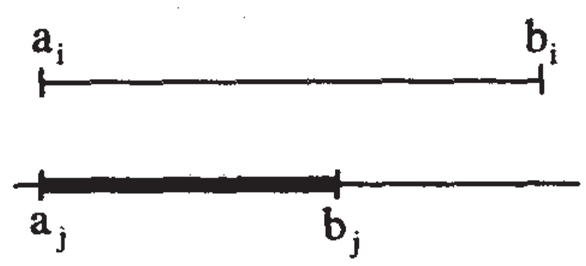


图 14-18

若$b_j < b_i$，则$[a_j,b_j]$包含在$[a_i,b_i]$中。但在排大小时$[a_i,b_i]$与$[a_j,b_j]$的顺序如何呢？回想一下前面的做法，不难看出在$a_i = a_j$时，若$b_i > b_j$则$[a_i,b_i]$就应该排在$[a_j,b_j]$前面。道理很简单，当$[a_i,b_i]$排在前面时，查到b_i而继续往后查时，才有机会遇到b_j而发现$b_j < b_i$，因此作出$[a_j,b_j]$包含在$[a_i,b_i]$中的决定。总结如下，先以$[a_i,b_i]$，$i=1,2,\cdots,n$中的a_i为主把这些区间的顺序排出来，$[a_i,b_i]$排在$[a_j,b_j]$前面的条件是$a_i < a_j$；但若$a_i = a_j$，则$b_i > b_j$。排好顺序之后，就依顺序去查每一个b_i，从b_1起，把它定成目前的最右边边界，再依$b_2,b_3,\cdots$顺序查找，一直到某个$b_k > b_1$为止，在$[a_2,b_2] \sim [a_{k-1},b_{k-1}]$之间

Download at http://www.pin5i.com/

的区间都包含在$[a_1,b_1]$中，但$[a_k,b_k]$却不是。不但如此，$[a_k,b_k]$还不能包含在任何一个区间中。因此把目前的最右边边界放到b_k去，再用相同的手法来处理$b_{k+1},b_{k+2},\cdots$，直到所有区间都处理完为止。

【程序说明】

在 CONTAIN.C 程序中，left[ ]与 right[ ]就是此处的a_i与b_i，即区间的左边、右边端点，mark[ ]是一个输出数组，如果 left[i]与 right[i]包含在某个区间中，mark[i]就是 YES，否则为 NO。

把快速排列法（QSORT.C）稍作修改来完成的区间排顺序的工作，关键是在 split()函数里 for 循环中的 if 叙述。其中，[x,y]是用来作分隔用的区间，如果目前的区间[left[i],right[i]]满足 left[i]<x，或者是 left[i]==x，如果 right[i]>y，那就表示[left[i],right[i]]要排在[x,y]前面，也就是前者比较小，就要移动分隔指针 current_ split。

至于 containment()中的 for 循环作用就是用来一个接一个地查右边端点，并且用前述的方法来判定[left[i],right[i]]是否包含在某个区间中。current_ right_end 就是目前最右边的边界。

注意：当 containment()传回控制权时，left[ ]与 right[ ]中的顺序已非交给它时的顺序了，而是依在前面所规定的顺序来排列的，所以 mark[ ]所记录下来的，就是以这个新的顺序为准。

在主程序中一共列出 10 个区间，不妨先把它画一个图，然后再依程序的输出顺序来检查一下，看看是否能够了解此区间所选择的顺序，再试试能否用手算来得到相同的结果。在输出中，有了记号的区间是包含在另一个区间中的，但究竟是哪一个却没有指出来，但由画的图则不难看出。

【习题】

（1）分析一下程序的效率，请问它一共进行了多少次比较？

（2）程序并没有报告出某个区间是包含在哪一个区间中，请修改程序，对于任一区间而言，它包含在哪一个区间中（如果有的话），报告一个区间即可。程序效率会不会受到影响？

（3）再写一个程序，对于任一区间而言，报告出包含了它的“所有”区间，请问程序效率会受到影响吗？就以本题的区间为例，[0,4]、[0,3]包含了[1,2]；[0,4]包含了[0,3]；[4,7]与[4,6]包含了[4,5]；[4,7]包含了[4,6]；[6,10]包含了[7,10]。

【参考文献】

这个题目取自 U.Manber 的一篇非常值得阅读的好文章。

U.Manber. Using Induction to Design Algorithms, *Communications of ACM*. Vol.31(1988), pp.1300～1313

【程序】

程序 14-19（CONTAIN.C）

```
/* ------------------------------------------------------------------ */
/* FUNCTION  containment :                                            */
/*   Given a set of intervals, this function determines               */
```

Download at http://www.pin5i.com/


```
/* any one interval is contained in some other intervals.        */
/* It returns an array mark[] in which YES means the             */
/* corresponding interval is contained in some other             */
/* intervals.  NOTE that on return the end points array          */
/* of the intervals, namely left[] and right[] are in            */
/* sorted order.                                                 */
/*                                                               */
/* Copyright Ching-Kuang Shene      Sep/01/1989                  */
/* ------------------------------------------------------------- */

#define  YES 1
#define  NO  0

/* ------------------------------------------------------------- */
/*                  function prototypes of sorting               */
/* ------------------------------------------------------------- */

void  sort(int [], int [], int);
void  qsort(int [], int [], int, int);
void  split(int [], int [], int, int, int *);
void  swap(int *, int *);

void  containment(int left[], int right[], int mark[], int n)
{
     int  current_right_end;     /* indicate current right       */
     int  i;

     ort(left, right, n);        /* sort end points              */

     current_right_end = right[0];
     mark[0] = NO;
     for (i = 1; i < n; i++)     /* scan all other right ends    */
         if (right[i] <= current_right_end)
             mark[i] = YES;      /* if <= current --> YES        */
         else {
             mark[i] = NO;       /* if > current, update it      */
             current_right_end = right[i];
         }
}


/* ------------------------------------------------------------- */
/* FUNCTION sort :                                               */
```

Download at http://www.pin5i.com/


```
/*   This routine converts two arguments to three                    */
/* arguments and calls the REAL control routine.                     */
/* ------------------------------------------------------------      */

void sort(int left[], int right[], int n)
{
     int  first = 0;     /* the fisrt element.                       */
     int  last  = n-1;   /* the last element.                        */

     qsort(left, right, first, last);
}


/* ------------------------------------------------                  */
/* FUNCTION  qsort :                                                 */
/*   This routine accepts an array and two indices                   */
/* indicating the first and the last element to be sorted.           */
/* Then calls split() to split the array into two parts              */
/* and sorts them recursively.                                       */
/* ------------------------------------------------------            */

void  qsort(int left[], int right[], int first, int last)
{
     int  split_point;

     if (first < last) {
         split(left, right, first, last, &split_point);
         if (split_point - first < last - split_point) {
             qsort(left, right, first, split_point-1);
             qsort(left, right, split_point+1, last);
         }
         else {
             qsort(left, right, split_point+1, last);
             qsort(left, right, first, split_point-1);
         }
     }
}


/* ------------------------------------------------                  */
/* FUNCTION  split :                                                 */
/*   This routine splits the given array into two parts              */
/* according to Nico Lomuto's aplitting technique and                */
```

Download at http://www.pin5i.com/


```
/* returns the splitting index with 'split_point'.  Note          */
/* that the sorting order is defined as follows: <left[i],        */
/* right[i]> is placed before <left[j],right[j]> if and           */
/* only if left[i]<left[j] or (left[i]=left[j] & right[i]          */
/* > right[j]).                                                    */
/* ----------------------------------------------------------------- */

void  split(int left[], int right[], int first, int last, int *sp)
{
     int  current_split,  i;
     int  x, y;

     x = left[first];
     y = right[first];
     current_split = first;
     for (i = first+1; i <= last; i++)
         if (left[i] < x || (left[i] == x && right[i] > y)) {
             (current_split)++;
             swap(&left[current_split], &left[i]);
             swap(&right[current_split], &right[i]);
         }
     swap(&left[first], &left[current_split]);
     swap(&right[first], &right[current_split]);
     *sp = current_split;
}


/* ----------------------------------------------------------------- */
/* FUNCTION  swap :                                                  */
/*   This is the simplest routine which swaps two data.              */
/* ----------------------------------------------------------------- */

void  swap(int *p, int *q)
{
     int temp;

     temp = *p;
     *p  = *q;
     *q  = temp;
}
```

Download at http://www.pin5i.com/

第15章 字 符 串

问题 6.1 括号匹配问题（PARCOUNT.C）

在某个字符串中包含有左括号、右括号与其他符号；规定（与常见的算术式子一样）任何一个左括号都从内到外地与在它右边、距离最近的右括号相匹配。请写一个程序，找出无法匹配的左括号与右括号，并且在输入列下方把它们标出来。

【解答】

按照问题说明中做法，要找出左、右括号是否相互匹配是很简单的，用一个计数变量就行了，但要做到本题的要求，还得加上一些额外的东西。为什么？不妨想一下处理下面的输入：

```
((ABCD(X)
```

在处理到 X 时，一共发现 3 个左括号，到达右括号时，因为要用掉一个匹配的左括号，所以计数变量的值减 1，变成 2。如果现在就是数据终了，那当然因为计数变量值不是 0，可以报告说是有多余的左括号，但在哪？这可是无能为力，此时操作已经处理过头了。因此为了克服这一点，要用一个数组 location[ ]来记住每一个左括号在输入中的位置，当见到了一个左括号，就把它的位置放到 location[ ]的下一个元素中，当碰到了右括号，就把 location[ ]中最近放进去的那一个（左括号的）位置去掉，表示它已经有了匹配的对象。于是，location[ ]中有多少个元素的变量，不就是前述的计数用的变量吗？但是，当见到右括号时，location[ ]中却没有元素了，这表示目前所看到的右括号根本没有匹配的对象，所以是一个错误。

接着，要如何在多余的左括号下方输出 1 个“$”，而在多余的右括号下方输出 1 个“?”呢？这倒简单，使用另一个 char 数组 error[ ]。注意，一般而言，不到整列看完，不知道左括号是否多余，不过它们的位置都在 location[ ]中，但是一看到右括号而 location[ ]没有元素了，就表示该右括号是多余的，在 error[ ]中对应这个右括号的位置存一个“?”。当整列输入都处理完了，如果 location[ ]中还有元素，这就是那些多余的左括号的位置，把它们逐一地取出，在 error[ ]中对应的位置存入“$”就行了。

在 par_count()函数中，line[ ]是输入行；error[ ]是用来存放“?”与“$”这些记号用的；SW 是一个整数，用它来指出有没有错误发生。

在程序中有一个内部数组 location[ ]，它的功能就是上面所提到的，loc_ptr 用来记录 location[ ]中有多少个元素。有一点不同的地方就是分别用 left 与 right 这两个变量来储存到目前为止一共有多少个左括号与右括号。如果现在看到一个右括号，而在把 right 加 1 之前，

Download at http://www.pin5i.com/

发现 left<=right，一旦加上 1 就造成 left<right，亦即右括号个数多过左括号个数，因此目前这一个右括号就是多余的。程序其他部分与上述完全相同，请自行研读。

【习题】

（1）这个程序只计算了左、右括号，请修改它，让它可以处理{与}，[与]，以及（与）3 种不同的括号，程序中是否要用 3 个 location[ ]数组，分别存放{、[与（的个数？还是只用一个 location[ ]数组就够了。程序能够处理{[}]与{[(]})这样的错误吗？

（2）请修改程序，使它可以处理有许多列的输入。而且，不妨把（与）改成 begin 与 end，或/*与*/，就有了一个可以检查 begin-end 或/*与*/是否相匹配的类型程序了。

【程序】

程序 15-1（PARCOUNT.C）

```
/* ------------------------------------------------------------ */
/* FUNCTION par_count :                                         */
/*   Given an input string containing ( and ), this            */
/* function counts the # of ('s and )'s and indicates the       */
/* positions of those incorrect placed ('s and )'s.  The        */
/* input string is stored in line[].  error[], which            */
/* stores a set of error position indicators, is returned       */
/* and thus can be printed below the input line so that         */
/* all error positions are macthed with the input.  'sw'        */
/* is an indicator variable, =0 means no error.                 */
/* Copyright Ching-Kuang Shene          July/08/1989            */
/* ------------------------------------------------------------ */

/* ------------------------------------------------------------ */

#define  MAXLENGTH   100
#define  YES 1
#define  NO  0

void par_count(char line[], char error[], int *sw)
{
    int  location[MAXLENGTH];
    int left    =   0;
    int right   =   0;
    int loc_ptr =   -1;
    int i;

    *sw = NO;                   /* assume there is no error        */
    for (i = 0; line[i] != '\0'; i++) {     /* scan the input      */
        error[i] = ' ';         /* initial to no error mark        */
        if (line[i] == '(') {   /* is it a '(' ?                   */
```

Download at http://www.pin5i.com/


```
            location[++loc_ptr] = i;/* YES, save pos            */
            left++;                 /* and increase '(' count   */
        }
        else if (line[i] == ')')    /* or if it is a ')'        */
            if (left <= right) {    /* too many }'s ?           */
                error[i] = '?';     /* YES, record a mark       */
                *sw     = YES;      /* and set error switch     */
            }
            else {
                right++;            /* no error, inc. # of )'s  */
                loc_ptr--;          /* remove the matched (     */
            }
    }
    error[i] = '\0';                /* append a '\0'            */
    if (loc_ptr >= 0) {             /* are there more ('s ?     */
        *sw = YES;                  /* YES, all of them are err.*/
        for (i = 0; i <= loc_ptr; i++) /* therefore set         */
            error[location[i]] = '$';  /* error marks.          */
    }
}
```

问题 6.2　转换成后继式写法（POLISH.C）

请写一个程序，读入一道正确的算术式，把它转译成反向波兰形式（Reversed Polish Form）。为了方便起见，假设整道算术式在同一列上，并且变量只有一个英文字母，不含常数（换言之，所有运算符都是一个字母的变量）。目前，只需要处理+、-、*、/、(、)即可，没有正负号。

【解答】

在程序中用一个堆栈记忆还不能计算的各个运算符，不过事实上这个堆栈是分成两部分来看，第一部分（叫做 stack[ ]）储存运算符符号本身，另一部分（叫做 code[ ]）则储存该运算符的优先级（Precedence）。优先级的定义如下：

BOTTOM（堆栈底部）0
EOL（输入终了）1
LEFT_PAR（左括号）2
RIGHT_PAR（右括号）3
PLUS_MINUS（加或减）4
MUL_DIV（乘或除）5

注意，因为先乘除后加减的缘故，PLUS_MINUS 的值就比 MUL_DIV 的值小，至于其

Download at http://www.pin5i.com/

他符号的优先级稍后再作解释。

在程序中读到了输入值之后，如果对应的是个英文字母，那就是个操作数，把它输出。如果是个左括号，这就表示一个新的操作数开始了，为了要记住这个新操作数的边界，把左括号与它的优先级推入堆栈中。如果得到的不是个空白符号（White Space），就依加与减、乘与除、右括号、输入终了来分类，不是这几类，也不是操作数、右括号的话，那就是个不能使用的符号了，因此报告错误，停止处理。如果是+、-、*、/、)或输入终了其中之一，变量 opr 就会得到对应的优先级值。这样，opr 中就是下一个运算符，而堆栈顶端的就是在此之前被放进去的那一个运算符，如果那一个运算符的优先级比下一个高（比如堆栈顶端的是*，而下一个是+，就好像 a*b+c 中的一样），于是堆栈顶端的运算就可以进行了，它的两个操作数可以输出。所以，就把堆栈顶端的运算符剔除并且输出，直到堆栈顶端元素的优先级比下一个运算符小为止。

例如处理 a*b+c，当读到 a 时，把它输出，读到*，放到堆栈中；读到 b 输出，接下来读到+，在此时有：

输出：a b　　输入：c

堆栈：*　　下一个运算符：+

因为*的优先级比+高，因此把*输出：

输出：a b *　　输入：c

堆栈：　　下一个运算符：+

就这样，可以把堆栈中优先级大过下一个运算符的运算符全部“逼”出来，直到堆栈中空无一物或顶端元素的优先级小于下一个运算符为止。在前者，下一个运算符就是剩下不曾计算的第一个运算符；在后者，因为在下一个运算符之前的（在堆栈中）运算符优先级比下一个运算符小，在看到进一步的运算符之前并不能决定出它能否计算，所以就把下一个运算符推入堆栈，等候进一步的情况。在这种情况下，事实上可以分成 3 个个案来处理。

（1）堆栈顶端是（下一个运算符是）：这表示这一对括号中的运算符全部计算完成，并且也已经输出，但反向波兰形式中是不需要括号的，因此把它们都去掉就行了。在此应该了解为什么左括号的优先级比四则运算符的小，因为当堆栈顶端是左括号，在下一个运算符是+、-、*、/四者之一时，这些运算符才会进入堆栈。换言之，在堆栈中的左括号是用来等待对应的右括号出现的，所以右括号的优先级也比左括号的小。

（2）如果下一个符号是文件终了：因为它的优先级很小，会把所有的运算符都“逼”出来，直到堆栈空了为止，于是翻译的工作就结束了。

（3）在堆栈底部应该先放好一个 BOTTOM 操作数，它具有最小的优先级值，因此在一开始时任何运算符都会被推入堆栈；而且，如果到了输入终了，就会把所有在 BOTTOM 上方的元素全都“逼”出来。

了解这个操作方式之后，就了解为什么有前面的优先级值了。例如，处理 a*(b+c)/d+k，输入、输出、堆栈（从左向右看，右边是顶端）是：

输出：a b c　　输入：　/d+k

堆栈：* (+　　下一个运算符：　)

因为)的优先级比+低，因此+就被“逼”出来了：

Download at http://www.pin5i.com/

输出：a b c +　　输入：/d+k

堆栈：*(　　下一个运算符：　)

照上面的说法，这一对括号都要去掉（也不输出）：

输出 a b c +　　输入：d+k

堆栈：*　　下一个运算符：/

接着再处理/，因为它与*相等，所以输出*，当然同时还要把/放入堆栈；再下一个是 d，其次是+：

输出：a b c + * d　　输入：　k

堆栈：/　　下一个运算符：+

因为+的优先级比/小，所以/就被输出，但是这就使堆栈空了，也就是只剩下 BOTTOM（没写出来），于是就不知道+可不可以计算，只好把+推入堆栈。接着读到 k，把它输出，接下来就是输入终了：

输出：a b c + * d / k　　输入：

堆栈：+　　下一个运算符：EOL

因为除了 BOTTOM 之外，EOL 的优先级是最低的，这就造成把+剔除并且输出；之后，堆栈空了，下一个运算符是 EOL，因此工作结束，输出是：

a b c + * d / k +

在程序中，initial()函数把 BOTTOM 存入 code[ ]（注意，不是 stack[ ]，事实上 stack[ ]只存放运算符符号，真正使用的是存入优先级的 code[ ]），并且把堆栈指针指向那；在堆栈底部存放一个特殊值，而在运用堆栈时依靠这个特殊值来判断堆栈是否还有元素，是一种很常见、常用的方法。stack_top()函数返回 code[ ]堆栈顶端的值，也就是上一个运算符的优先级。push()把运算符符号 opertor 与它的优先级（opr_code）分别存入 stack[ ]与 code[ ]；而 pop()则取出 stack[ ]的第一元素，而非 code[ ]。这几个程序与它使用的数据 top、stack[ ]与 code[ ]等，在编写程序时是希望自成一个文件，而让主程序看不到的，在后面的程序报表中合并在一起是为阅读方便而已。

主程序的写作完全照前面的讨论写成，因此就不重复讲解了。

【习题】

（1）为什么检查现在输入的符号是不是“(”的操作不一起写在 switch 叙述中呢？请说明看法。如果真的写到 switch 叙述中，程序要有什么改变？如果把检查 EOL 的工作放到 switch 叙述之外，又会有什么影响？

（2）为什么程序中的 while 叙述在比较时用的是“>=”，而不是“>”呢？如果改用“>”会有什么现象？

（3）请找出至少一个程序查不出来，但却是错误的输入。能否不执行程序而找出来吗？为什么会有这个现象？请改正它。

（4）如果只用英文字母作变量名称，这个程序还正确吗？为什么？如果变量名称中又有数字符号，那又有什么状况发生？

（5）请扩充这个程序，让它可以接收 C 的变量名称。

Download at http://www.pin5i.com/

（6）请扩充这个程序，让它可以接收正整数作操作数。

（7）请扩充这个程序，让它可以处理指数运算符(^)；也就是说 a^b 表示 a 的 b 次方。注意，^与+、-、*、/不一样。第一，它有最高的优先级。第二，它是从左往右算。以加法而言，a+b+c 是 a b + c +，这是从左往右算的规则；但数学中 a^{b^c} 是 $a^{(b^c)}$ 而不是 $(a^b)^c$，所以 a^b^c 是 a^(b^c)，翻译出来之后是 a b c ^ ^，而不是(a^b)^c（结果为 a b ^ c ^）。

提示：注意^在堆栈中的优先级，若堆栈顶端是^，下一个运算符是^时，会把下一个运算符推入堆栈，而不是把堆栈顶端的去掉。

（8）请修改程序，让它可以处理负号(-)，但为了方便起见，负号不可以与其他 4 个运算符相邻，所以 a+-b 是错的，但 a+(-b)或-b+a 则正确，在输出时，请用"表示负号，所以 a+(-b)与-b+a 就分别转换成 a b "+与 b " a +。

提示：因为负号不能与其他运算符相邻，那它只能在哪些地方呢？

（9）加上设定运算符(=)，于是 a=b+c 就转换成 a b c + =，这表示把 b 与 c 的和存入 a。=的优先级应该是多少？也试试处理像 C 语言中的连续设定的操作，比如把 a=b=x+y 译成 a b x y + = =。

（10）修改程序，让它可以找出表达式中的所有错误，并且报告出来。

【参考文献】

有关使用运算符的优先级来处理算术式的技巧，可参看下面这本书的第 4.6 节：

1 A.V.Aho, R.Sethi and J.D.Ullman. *Compilers: Principles, Techniques, and Tools*, Addison-Wesley, 1986

或者是看它的中译本：

2 编译程序：原理、技巧与工具（上册）. 冼镜光译. 儒林图书公司：1987

【程序】

程序 15-2（POLISH.C）

```
/* ------------------------------------------------------------ */
/* PROGRAM reversed polish form translator :                    */
/*   Given a string containing an arithmetic exoression,        */
/* this program translates it to reversed polish form by        */
/* using a stack.                                               */
/*                                                              */
/* Copyright Ching-Kuang Shene      July/12/1989                */
/* ------------------------------------------------------------ */

#include <stdio.h>       /* for I/O operation                   */
#include <stddef.h>      /* for EXIT_FAULAIR & ...              */
#include <ctype.h>       /* for isspace() & ...                 */
```

Download at http://www.pin5i.com/


```
#define  BOTTOM       '\0'        /* precedence of stack bot.     */
#define  EOL          '\1'        /* precedence of end-of-line    */
#define  LEFT_PAR     '\2'        /* precedence of (              */
#define  RIGHT_PAR    '\3'        /* precedence of )              */
#define  PLUS_MINUS   '\4'    /* precedence of + and -            */
#define  MUL_DIV      '\5'    /* precedence of * and /            */

#define  EXIT_SUCCESS    1        /* remove it if ANSI C          */
#define  EXIT_FAILURE    0        /* remove it if ANSI C          */

void initial(void);               /* initial stack                */
char stack_top(void);             /* return top of prec. Stack    */
void push(char, char);            /* push opr and its preced.     */
char pop(void);                   /* pop operator only            */

int   main(void)                  /* main routine starts here     */
{
     char  line[100];             /* input line                   */
     char  *p;                    /* line cursor                  */
     char  opr, t;

     printf("\nCompile to Reversed Polish Program");
     printf("\n==================================");
     printf("\n\nYour Input --> ");
     p = gets(line);
     printf("\n\nAnswer is ---> ");

     initial();                   /* initialize the two stacks    */
     for (; ; p++) {              /* loop for each char.          */
         if (isalpha(*p))         /* is it an operator            */
             printf("%c ", *p);   /* YES, output it               */
         else if (*p == '(')      /* or, is it a '('              */
             push(*p, LEFT_PAR);  /* YES! push it.                */
         else if (!isspace(*p)){  /* or, not a space?             */
             switch(*p) {         /* it must be an operator       */
                 case '+' :       /* handle + and -               */
                 case '-' : opr = PLUS_MINUS;
                        break;
                 case '*' :       /* handle * and /               */
                 case '/' : opr = MUL_DIV;
                        break;
                 case ')' : opr = RIGHT_PAR; /* ')'               */
```

Download at http://www.pin5i.com/


```
                break;
        case '\0': opr = EOL;        /* EOL                    */
                break;
        default : printf("*** Unrecognizable char ***");
                exit(EXIT_FAILURE);
    }
    while ((t=stack_top()) >= opr) /* pop low                  */
        printf("%c ", pop());       /* & print                  */
    if (t == LEFT_PAR && opr == RIGHT_PAR)
        (void) pop();               /* remove ( and )           */
    else if (opr == EOL)            /* end of parsing           */
        exit(EXIT_SUCCESS);
    else
        push(*p, opr);              /* otherwise, push          */
    }
  }
}


#define  MAX_DEPTH   100
#define  BOTTOM      '\0'

Static char stack[MAX_DEPTH];    /* operator stack              */
static char code[MAX_DEPTH];     /* operator precedence stack   */
static   int top;                /* stack pointer               */

void initial(void)               /* initialize operator stack   */
{
     top = 0;
     code[top] = BOTTOM;
}

char stack_top(void)             /* ret. top element of prec    */
{
     return code[top];
}

void push(char operator, char opr_code)     /* push opr & pre   */
{
     if (++top == MAX_DEPTH) {
         printf("\n*** ERROR *** Stack Overflow.");
         exit(1);
```

Download at http://www.pin5i.com/


```
            }
            else {
                stack[top] = operator;
                code[top]  = opr_code;
            }
        }

    char pop(void)          /* pop up operator only                    */
    {
        if (top == 0) {
            printf("\n*** ERROR *** Stack Underflow.");
            exit(1);
        }
        else
            return stack[top--];
    }
```

问题 6.3 计算前置式写法（PREFIX.C）

课本中都会提到如何算一道用反向波兰形式（Reversed Polish Form）的表达式的计算方式，但如何计算前置式波兰形式（Prefix Polish Form）的表达式呢？为了简单起见，表达式在同一列上，只有加、减、乘、除 4 个运算符，操作数只有一个数字符号。请写一个程序，接收一道前置式波兰形式的表达式，把结果求出来。

【解答】

在做反向波兰形式时，把还不能计算的操作数放到一个堆栈中，看到了一个运算符（来自输入），就从堆栈中取两个操作数出来，与该运算符运算之后，把结果再推回堆栈。但是前置式的写法却多了一重障碍，从问题说明中也可以发现，这就是也有还不能计算的操作数，但是使用堆栈来存放这些还不能计算的操作数与运算符还是一条正确的路。

先不管类型的问题，如果有一个堆栈，看到运算符就把它放进去，因为在前置式写法中操作数都出现在运算符的后面，所以出现运算符就表示还不曾看到它的操作数，自然是不能计算的。当看到了一个操作数，那么它可不可以运算呢？这就得问一问堆栈，看看在最顶上的元素是什么？如果是运算符，则见到的操作数就是该运算符的第一个操作数，第二个操作数还没有出现，所以现在看到的操作数就推入堆栈。如果现在堆栈顶端的是一个操作数，就等于有了两个操作数了，那么运算符在何处？回想一下前置式的写法，运算符写在它的操作数前面，现在在输入中有一个操作数，在堆栈顶端有一个，自然地对应的运算符就是在堆栈中顶端算起的第二个，因此从堆栈中取出那个操作数与运算符，与输入中找出的操作数计算过后，得到一个值，这个值自然是其他运算符的某个操作数。

接下来怎么办？把结果推入堆栈吗？不，因为若把它推入堆栈，原来堆栈中又有另一

Download at http://www.pin5i.com/

个操作数与一个运算符，那个运算符就算不到了（为什么？）。所以，还要再查顶端的是不是操作数，如果是，就一直反复上面的计算步骤——取出顶端的操作数与下一个运算符，算出结果。一直做到堆栈顶端不是操作数为止，算出来的结果就是该运算符的第一个操作数。

举例说明，算出下式的结果：

-*2+54+12/8+13

将要处理的内容分成堆栈、内存、输入 3 个部分，分别表示在堆栈中、在内存中，以及输入中剩余的内容，如表 15-1 所列。

表 15-1

例　数	堆　栈	内　存	输　入
0.			-/*2+54+12/8+13
1.	-	-	/*2+54+12/8+13
2.	-/	/	*2+54+12/8+13
3.	-/*	*	2+54+12/8+13
4.	-/*2	2	+54+12/8+13
5.	-/*2+	+	54+12/8+13
6.	-/*2+5	5	4+12/8+13
7.	-/*2+5	4	+12/8+13
8.	-/*2	9	+12/8+13
9.	-/	18	+12/8+13
10.	-/18	18	+12/8+13
11.	-/18+	+	12/8+13
12.	-/18+1	1	2/8+13
13.	-/18+1	2	/8+13
14.	-/18	3	/8+13
15.	-	6	/8+13
16.	-6	6	/8+13
17.	-6/	/	8+13
18.	-6/8	8	+13
19.	-6/8+	+	13
20.	-6/8+1	1	3
21.	-6/8+1	3	
22.	-6/8	4	
23.	-6	2	
24.		4	
25.	4	4	

表 15-1 中画了底线的部分就是要取出的操作数或运算符，它与内存中的操作数算出结果后留在内存内，再去看堆栈。第 7 与第 8、第 13 与第 14，以及第 20～24 列都有这个

Download at http://www.pin5i.com/

例子。

第二个要解决的问题，就是堆栈既要存放运算符，又要存放操作数，当然在输入时都是 char 类型，但是在中间计算出来的结果却未必，看第 8 列会得到 18 就是一个例子，所以堆栈可以存放 char 与 int 才行。不过为了方便，把运算结果定成 double，堆栈中的元素可能是 char，也可能是 double，不但如此，还要分出该元素是 char 还是 double；这可就要用等位（union）的观点了。定义一个结构如下：

```
struct item {
    union {
        double value;
        char operation;
    } store;
    int type;
};
```

新堆栈中的元素每一个都有这样的结构，它的第一个位置是个等位，既是 double（存放操作数的值），又可以是 char（存放运算符本身），以及一个表示该 union 中的值是哪一类的表示用变量 type。此外为了方便起见，在开始时，就在堆栈中放入一个表示堆栈底部的元素 STACK_BOTTOM，它既不是运算符（OPERATOR），又不是操作数（OPERAND），所以在操作时当堆栈的顶端元素的类型是 STACK_BOTTOM 时，就表示堆栈中已经空无一物了。

在程序 PREFIX.C 中，主程序用 initial()来为堆栈赋初值，再用一个 for 来处理每一个读入的符号，当 is_opr()函数返回真，那么该符号就是运算符，用 push_opr()把运算符推入堆栈。但若读入符号是数字符号，那它就是个操作数，由此把它的值存放到 opn2 中，接下来的 while 循环就不断地测试堆栈顶端元素是不是个操作数，如果是，就把它取出存放到 opn1 中，再取出对应的运算符 opr，交给 compute()函数运算符结果存回 opn2；就这样一直反复，直到顶端元素不是操作数为止，上一个操作数就没有了匹配的另一个操作数与运算符了，所以用 push_opn()把它推入堆栈。在所有工作都做完之后，结果在堆栈顶端（自然也在 opn2 中）把它取出，再显示出来。

维护堆栈的函数有 initial()、push_opn()、push_opr()、pop_opn()、pop_opr()与 stack_top()。堆栈的结构应该放在另一个文件中，避免与其他程序相互干扰，但为了方便，却把它连在一起，在实用上最好不要这样做。注意，stack_top()的函数值是堆栈顶端元素的类型，可不是它的内容。

【习题】

（1）为什么把算出来的结果放到 opn2 中，而把从堆栈取出的操作数放在 opn1？如果倒反过来，结果还正确吗？

（2）如果一旦算出结果之后就把结果推入堆栈，接着处理其他输入；这是不对的；为什么？请找一个例子。

（3）请在程序中加入^（乘幂），-（负号）这两个运算符；如果说输入中的数值不止

Download at http://www.pin5i.com/

一位，但一定是整数，程序要不要改动？如果输入中的数可能是有小数点呢？

（4）请加上一个@运算符，当看到这个运算符时，就把堆栈顶端元素的值显示出来（但不剔除）。

（5）请加上一个 D 运算符，当看到这个运算符时，就把堆栈顶端元素复制一份再放回堆栈，因此执行过 D 之后，堆栈顶端就会有两个完全相同的元素。

【程序】

程序 15-3（PREFIX.C）

```
/* ------------------------------------------------------------ */
/* PROGRAM  prefix polish form evaluator :                      */
/*   Given an input string of Prefix Polish Form with           */
/* integer constants and +, -, * and operator, this             */
/* program will calculate the result of the Prefix form.        */
/*                                                              */
/* Copyright Ching-Kuang Shene      July/29/1989                */
/* ------------------------------------------------------------ */

#include <stdio.h>
#include <ctype.h>              /* for isdigit()                */

#define  LINE_SIZE       100    /* line buffer length           */

#define  STACK_BOTTOM      0    /* stack bottom indicator       */
#define  OPERAND           1    /* operand symbol indicator     */
#define  OPERATOR          2    /* operator symbol indicator    */

/* ------------------------------------------------------------ */
/*                function prototypes                           */
/* ------------------------------------------------------------ */

int  is_opr(char);              /* test for operator            */
double   compute(char, double, double); /* evaluator            */
void     initial(void);         /* stack initialization         */
void push_opn(double);          /* push an operand              */
void     push_opr(char);        /* push an operator             */
double   pop_opn(void);         /* pop an operand               */
char pop_opr(void);             /* pop an operator              */
int  stack_top(void);           /* return type of top item      */

void  main(void)
{
```

Download at http://www.pin5i.com/


```
void main(void)
    double  opn1, opn2;             /* working operands        */
    char    s[LINE_SIZE];           /* input line buffer       */
    char    opr;                    /* working operator        */
    char    *p;                     /* buffer cursor           */

    printf("\nPrefix Form Evaluator");
    printf("\n====================");
    printf("\n\nInput --> ");
    gets(s);
    initial();                      /* initial stack           */
    for (p = s; *p != '\0'; p++)    /* for each character      */
        if (is_opr(*p))             /* is it an operator ?     */
            push_opr(*p);           /* YES, push.              */
        else if (isdigit(*p)) {     /* is it an operand ?      */
            opn2 = *p - '0';        /* YES, convert to double  */
            while (stack_top() == OPERAND) {
                opn1    = pop_opn();/* for all of its          */
                opr = pop_opr();    /* buddy, compute          */
                opn2    = compute(opr, opn1, opn2);
            }
            push_opn(opn2);         /* then push the result    */
        }
    printf("\nResult = %lf", pop_opn());
}

/* ----------------------------------------------------------- */
/* FUNCTION is_opr :                                           */
/*   Tests to see if 'opr' contains an operator.               */
/* ----------------------------------------------------------- */

int is_opr(char opr)
{
    return opr == '+' || opr == '-' || opr == '*' || opr == '/';
}

/* ----------------------------------------------------------- */
/* FUNCTION compute :                                          */
/*   Given two operands and an operator, this function         */
/* computes the result.                                        */
/* ----------------------------------------------------------- */
```

Download at http://www.pin5i.com/


```
double  compute(char opr, double opn1, double opn2)
{
     double  result;

     switch (opr) {
         case '+'    : result = opn1 + opn2;
             break;
         case '-'    : result = opn1 - opn2;
             break;
         case '*'    : result = opn1 * opn2;
             break;
         case '/'    : result = opn1 / opn2;
             break;
     }
     return result;
}

/* ----------------------------------------------------------------- */
/*              stack management package                             */
/* ----------------------------------------------------------------- */

#define  STACK_SIZE   100          /* stack size                     */

struct item {                      /* stack record  :                */
     union {                       /* shared memory for              */
         double  value;            /* operand value, or              */
         char    operator;         /* operator symbol                */
     } store;
     int   type;                   /*   type of stored item          */
};

static struct item  stack[STACK_SIZE];   /* actual stack             */
static int   top;                        /* stack pointer            */

/* ----------------------------------------------------------------- */
/* FUNCTION  initial :                                               */
/*   Initializes the stack with STACK_BOTTOM symbol.                 */
/* ----------------------------------------------------------------- */

void  initial(void)
{
```

Download at http://www.pin5i.com/


```
    top = 0;
    stack[top].type = STACK_BOTTOM;
}

/* ------------------------------------------------------------ */
/* FUNCTION  push_opn :                                         */
/*   Pushes an operand (a value) into the stack.                */
/* ------------------------------------------------------------ */

void  push_opn(double data)
{
    stack[++top].type   = OPERAND;
    stack[top].store.value = data;
}

/* ------------------------------------------------------------ */
/* FUNCTION  push_opr :                                         */
/*   Pushes an operator into the stack.                         */
/* ------------------------------------------------------------ */

void  push_opr(char opr)
{
    stack[++top].type   = OPERATOR;
    stack[top].store.operator   = opr;
}

/* ------------------------------------------------------------ */
/* FUNCTION  pop_opn :                                          */
/*   Pops up and returns the top-most operand.                  */
/* ------------------------------------------------------------ */

double  pop_opn(void)
{
    return  stack[top--].store.value;
}
/* ------------------------------------------------------------ */
/* FUNCTION  pop_opr :                                          */
/*   Pops up and returns the top-most operator.                 */
/* ------------------------------------------------------------ */

char  pop_opr(void)
{
```

Download at http://www.pin5i.com/


```
    return  stack[top--].store.operator;
}

/* ----------------------------------------------------------------- */
/* FUNCTION  stack_top :                                             */
/*   Returns the type of the top-most item in stack.                 */
/* ----------------------------------------------------------------- */

int  stack_top(void)
{
    return  stack[top].type;
}
```

问题 6.4　Knuth-Morris-Pratt 法寻找字符串（KMP.C）

请开发一个效率较高的，寻找部分字符串的程序。

【解答】

先讲解如何求出 p[0]～p[i]的 A 与 B 的方法，A 是 p[0]～p[i]的开头部分，叫做字符串的前段（Prefix），B 在字符串的尾部，叫做后段（Suffix）。找出一个字符串中有没有相等的前段与后段；不但如此，为了节省比较的次数，还要找出最长的相等前段与后段。

如果在 p[0]～p[i]之间的 A 与 B 分别在 p[0]～p[s]、p[i-s]～p[i]，就把 s 这个脚码存放到一个叫做 fail[ ]的数组的第 i 个元素中，亦即 fail[i]=s。因此，当 p[i+1]与对应的 t[ ]元素不相等时，把 A 移到 B 的下方，于是下一个与 t[ ]中对应元素相比较的就是 p[s+1]，亦即 p[fail[i]+1]。很明显地，p[0]不可能有不同的前段与后段，所以不能找出不同的 A 与 B，令 fail[0]为-1，表示这种情况；于是当 p[0]与 t[ ]中元素相比而发现不相等时，下一个要与该 t[ ]中元素比较的就是 p[fail[0]+1]=p[-1+1]=p[0]，仍然还是 p[0]，这相当于把 p[ ]向右移 1 位。

如何算出 fail[1]以后的值呢？只要回头仔细看看问题说明就行了。假设 fail[0]～fail[i-1]都算出来了，现在要填 fail[i]。看 fail[i-1]，这是 p[0]～p[i-1]之间有最长的相等前段与后段的脚码，也就是 p[0]～p[fail[i-1]]为 A，p[i-fail[i-1]]～p[i-1]为 B。因此 p[fail[i-1]+1]（A 的下一个元素）与 p[i]（B 的下一个元素）比较，如果相等，p[0]～p[i]的 A 部分就延伸了一个元素，变成 p[0]～p[fail[i-1]+1]，fail[i]就应该存入 fail[i-1]+1。

如果 p[fail[i-1]+1]与 p[i]不相等，这就表示 p[0]～p[fail[i-1]+1]太长了，要缩短它。从 fail[i-1]可以查出 A（p[0]～p[fail[i-1]]）中最长的相等前段与后段，也就是 p[0]～p[fail[fail[i-1]]]。

接着看 p[fail[fail[i-1]]+1]是否与 p[i]相等，相等的话，fail[i]的值就是 fail[fail[i-1]]+1。如果不相等，再次反复，也就是找出 p[0]～p[fail[fail[fail[i-1]]]]的最长相等前段与后段。一

Download at http://www.pin5i.com/

直反复这个手法，如果不是找到了，就是会碰到 fail[0]，因此这就指出要从头开始了。

上面的说明用不着写成递归程序，简单的两个循环就办得到。对于任一个 i 而言用：

```
for (j = fail[i-1]; j >= 0 && p[j+1] != p[i]; j = fail[j])
;
```

因此 p[0]～p[j]就是 A 这一部分。如果 j 不小于 0，表示 A 还不至于空无一物，就拿 A 的下一个元素 p[j+1]与 p[i]比较，如果不相等，j=fail[j]就是上面把 fail[i-1]变成 fail[fail[i-1]]的形式了。这个循环在 j<0 或 p[j+1]与 p[i]相等时停止。在后者 A 的部分（p[0]～p[j]）就可以延伸 1 位。因此 fail[i]=j+1，在前者，j<0 已经逸出 p[ ]的范围但还没有找到 A 部分，就令 fail[i]为-1 来记录这件事实。下一个要比较的符号是 p[fail[i]+1]=p[-1+1]=p[0]，正是 p[ ]的第一个符号。在后面的程序中，setup()这个函数就是用这个观点写成的，不过用的是 pat[ ]而不是 p[ ]。

有了 fail[ ]之后，就可以很容易写出查找部分字符串的函数了。从头开始把 t[ ]与 p[ ]一个接一个符号地比较，如果发现 t[i]与 p[j]不相等，就去找 p[0]～p[j-1]这一部分 A 区域，把它移到 B 区域去，而将 A 区域的下一个元素与 t[i]继续比较。对 p[0]～p[j-1]而言，A 区域在哪里？现在有 fail[ ]，马上就可以查出是在 p[0]～p[fail[j-1]]之间，而下一个元素正是 p[fail[j-1]+1]，所以 fail[j-1]+1 正是 j 的新值。不过，当 j 是 0 时，j-1 就超过了 fail[ ]的开头，不过这相当于从头开始比较，但是 t[i]与 p[0]已经不相等了，所以用 p[0]与 t[i+1]比较即可，这相当于把 t[ ]的脚码加 1。后面的程序中，KMP()就是做这件事的函数，不过 t[ ]与 p[ ]是用 text[ ]与 pat[ ]，i 与 j 则用 t 与 p 来表示而已。

【习题】

（1）如果 t[ ]是 xyxxyxyxyyxyxyxx，p[ ]是 xyxyyxyxyxx，求出 fail[ ]，并且用这个 fail[ ]来找出 t[ ]中是否有 p[ ]。答案：fail[ ]= {-1,-1,0,1,-1,0,1,2,3,2,0}。

（2）如果 t[ ]是 bcbcbabcabcaabcabcabcacab，p[ ]为 abcabcacab，求出 fail[ ]，并且用它来找出 t[ ]中是否有 p[ ]。答案：fail[ ]={-1,-1,-1,0,1,2,3,-1,0,1}。

（3）请修改程序，让它可以找出 t[ ]中最后一次出现 p[ ]的地方。

提示：比较方向相反。

（4）请写一个程序来测试一个字符串 t[ ]是否是回文（Palindrome）。所谓回文，就是正读与反读完全相同。

提示：p[ ]是什么？

（5）请修改程序，让它可以找出 t[ ]中所有出现 p[ ]的地方；例如，若 t[ ]是 aaaa，p[ ]是 aa，那么在 t[0]、t[1]、t[2]开头的位置都出现了 p[ ]。

提示：找到了 p[ ]之后，不要停下来。

【参考文献】

联系是第一个突破寻常部分字符串不需要 O(m×n)次比较的方法，是由 D.E.Knuth、

Download at http://www.pin5i.com/

J.H.Moris,Jr.与 V.R.Pratt 3 个人在 1977 年提出的，请参看下面的文章（对学习计算机科学的同学来说，值得一读）。

D.E.Knuth, J.H.Morris, Jr. and V.R.Pratt. Fast Pattern Matching in Strings, *SIAM Journal on Computing*. Vol.6(1977), pp.323～350

此外，很多讲数据结构与算法的书也都会讲到它，时至今日，这个方法已经是标准教材了，所以应该了解它。在很多文献中，都会把 3 个人名的第一个字母取出，而合称为 KMP 方法。

依据实验结果，KMP 方法是会快一些，但没有证明一点：KMP 方法只会用与 m+n 成正比的比较次数，而不是与 mn 成正比，请参看上列的文章或相关书籍。但数据显示，当 t[ ]并不长，或 t[ ]与 p[ ]中都是均匀的随机数据时，KMP 方法快得很有限，因为它要花时间建立 fail[ ]。不过，下一题的 Boyer-Moore 方法更快。因此看完、做完本题，请继续下一题。

【程序】

程序 15-4（KMP.C）

```
/* ------------------------------------------------------------ */
/* FUNCTION Knuth-Morris-Pratt string search program :          */
/*   Given text string and pattern string in text[] and         */
/* pat[] respectively, these two routines will find if          */
/* the pat[] string is somewhere in the text[] by using         */
/* the linear Knuth-Morris-Pratt algorithm.                     */
/*                                                              */
/* Copyright Ching-Kunag Shene      July/12/1989                */
/* ------------------------------------------------------------ */
#include <string.h>          /* for strlen()                    */

/* ------------------------------------------------------------ */
/* FUNCTION setup :                                             */
/*   Given the pattern pat[], setup builds up the failure       */
/* function table fail[].                                       */
/* ------------------------------------------------------------ */

void setup(char pat[], int fail[])
{
     int  length = strlen(pat);
     int  i, j;

     fail[0] = -1;
     for (i = 1; i < length; i++) {
         for (j=fail[i-1]; j>=0 && pat[j+1]!=pat[i]; j=fail[j])
             ;
         fail[i] = (j < 0 && pat[j+1] != pat[i]) ? -1 : j+1;
```

Download at http://www.pin5i.com/


```
        }
    }

    /* ------------------------------------------------------------ */
    /* FUNCTION  KMP :                                              */
    /*   The KMP pattern search routine.                            */
    /* ------------------------------------------------------------ */

    int KMP(char text[], char pat[], int fail[])
    {
        int  t_length = strlen(text);
        int  p_length = strlen(pat);
        int  t, p;

        setup(pat, fail);                /* build up the fail func. */
        for (t = p = 0; t < t_length && p < p_length; )
            if (text[t] != pat[p])       /* if not match ...        */
                if (p > 0)               /* is it the 1st location? */
                    p = fail[p-1] + 1;   /* no, shift right         */
                else
                    t++;                 /* or match next in text[] */
            else
                t++, p++;                /* matched, advance both   */
        return (p >= p_length) ? t - p_length : -1;
    }
```

问题 6.5　Boyer-Moore 法寻找字符串（BM.C）

根据实验与理论的分析，KMP 方法在一般情况下，并不见得会比传统的写法快多少，不过还有更好写而且平均会比 KMP 快的方法存在。这个方法由 Boyer 与 Moore 两人差不多与 KMP 方法同时发现，这个题目主要就是探讨 Boyer-Moore 方法。

【解答】

假设要在 text[ ]字符串寻找有没有 pat[ ]这个部分字符串，这两个字符串的长度各为 t_len 与 p_len。依照问题说明中的说法，如果对于 text[ ]中的某个元素 text[i]而言，它与 pat[p_len-1]，也就是 pat[ ]的最后一个元素不相等，并且 text[i]也不出现在 pat[ ]中，就可以一下子跳过 p_len 个符号。当然，不见得每一个 text[ ]中的符号都那么好，所以为了方便起见，可以用一个数组记录下各个符号可以跳多远。这个数组该有多大？这点很容易决定，在常见的计算机系统中，字符串的内容不外是 ASCII 码或 EBCDIC 码，最多 256 个符号而已，因此可以用一个有 256 个元素的整数数组 jump_tab[ ]来做这件事；于是，若 text[i]与

Download at http://www.pin5i.com/

pat[ ]中对应的符号不相同，要决定可以向右移多少格时，就去查 jump_tap[text[i]]，这就是要右移的格子数，所以下一个要比较的符号就是 text[i+jump_tab[text[i]]]。

但是，如何求出 jump_tab[ ]中各个符号该跳多少格呢？方法很简单。它最多是跳 pat[ ]的长度，所以在开始时可以把 jump_tab[ ]的每一个元素都定为 p_len，然后再逐步地修正它。

回想一下前述的比较方式，在固定了某个位置之后，是从右往左比较的，一直到发现 text[i]与 pat[j]不相等为止，接着把 pat[ ]最右边的那个与 text[i]相等的符号移到与 text[i]对齐的位置。在 jump_tab[ ]中对应着 text[i]的移动长度，就应该是 text[i]在 pat[ ]最右边的位置距最后一个符号的长度。方法就是将 pat[ ]中的每一符号，即 pat[i]，作 jump_tab[ ]的脚码，在 jump_tab [pat[i]]中存入 length-i-1，因此虽然在 pat[ ]中可能会出现符号 x 若干次，但若从左向右处理，在 pat[ ]的每一个符号都处理完后，jump_tab[ ]中对应着 x 这个位置的值，正好是 x 在 pat[ ]中距最后一个位置的距离。注意，pat[ ]的最后一个符号是不必算的，如果算了就一定是 0。

当这一份移动长度的表格求出来之后，就可以用它来做寻找字符串的工作了；不过方式却与问题说明中的办法有少许差异，如图 15-1 所示。

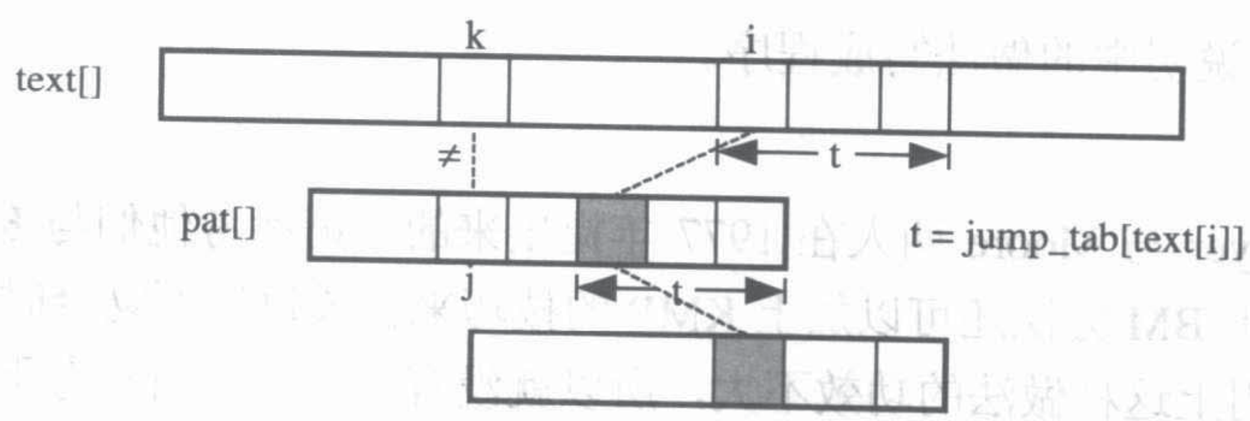


图 15-1

当把pat[ ]的最后一个符号与text[i]对齐而从右向左比较时，如果比到某个text[k]与pat[j]不相等时，就去查 jump_tab[text[i]]的值，这个值就是 pat[ ]中最后一次出现 text[i]的位置距 pat[ ]最后一个符号的距离，令为 t。注意，在 pat[ ]中的这一段位置中是不会出现 text[i]的，当把 pat[ ]向右移 t 格，而把它与 text[i]对齐时，新的 i 就是 i+t 了。因此又可以从 text[i]与 pat[p_len-1]起向右比较。

注意一点，把 pat[ ]向右移少于 t 格是没有意义的，从 jump_tab [text[i]]得知，t 格是第一次可以让 text[i]与 pat[ ]中最后一个等于 text[i]的符号吻合的地方。因此，若移少于 t 格，在比到 text[i]时就一定不会相等，就等于做了多余的事。

下面就是处理 text[ ]为 bcbcbabcabcaabcabcabcacab 而 pat[ ]为 abcabcacab 的例子，一个点表示没有参与比较的符号，加上了底线的符号是用来指出 pat[ ]中与 text[i]对齐的符号。

```
text[]   bcbcbabcabcaabcabcabcacab
pat[]          ......acab
               ....b....b
                   ......acab
                        ....b....b
                          ........ab
                            abcabcacab
```

Download at http://www.pin5i.com/

一共有多少次比较？22 次，比前一题 KMP 方法的 29 次少了 7 次。事实上，对于更长的字符串而言，Boyer-Moore 方法还会节省更多的时间。

【习题】

（1）有什么理由支持在求 jump_tab[ ]时可以不必理会 pat[ ]的最后一个元素？

（2）请证明，当从 text[i]起从右到左比较到 text[k]时，至少可以把 pat[ ]向右移 jump_tab[text[i]]格。虽然解答中已说明了理由，但希望能给出一个严格的论证。

（3）请用不同长度，并且用随机产生的字符串作 text[ ]，然后再在 text[ ]中用随机方法任意选出一个部分字符串作 pat[ ]，用这一组 text[ ]与 pat[ ]来测试一般方法、KMP 方法与 BM 方法的速度差异。自然地 text[ ]长度要长一些。

（4）第（3）题的做法是永远找得出 pat[ ]的例子，请在把 pat[ ]选定之后，把 pat[ ]中若干个符号换掉，再作一次比较。哪一个方法比较快？

（5）能不能指出 KMP 与 BM 两者在假定上的最大差异？

提示：考虑符号集。这一项差异在理论上与实际应用上会有什么影响？

（6）请把问题说明中的做法写成程序。

【参考文献】

这个方法是 Boyer 与 Moore 两人在 1977 年提出来的，请参考他们与 Knuth、Morris 与 Pratt 的文章。事实上 BM 方法还可以加上 KMP 的技巧来拉长每次不吻合时所应该右移的距离，但是在实际应用上这种做法的功效不大，所以就没有写出来。请参考下列的两篇文章。

1 R.S.Boyer and J.S.Moo. A Fast String Searching Algorithm, *Communications of ACM*. Vol.20(1977), pp.762～772

2 D.E.Knuth, J.H.Morris, Jr. and V.R.Pratt. Fast Pattern Matching in Strings, *SIAM Journal on Computing*. Vol.6(1977), pp.323～350

所讲的方法其实并不是个最好的做法，就用 text[ ]为 aaaa…aa，而 pat[ ]为 baaa…aa 为例，每次都要比完 pat[ ]的所有符号，发现第一个不相等，然后右移一格，然后重复这件工作。如果 text[ ]与 pat[ ]的长度各为 n 与 m，那么比较的次数就与 m・n 成正比了；但是，BM 方法是可以稍作修改而使得比较次数与 n(n＞m)成正比的。请参看下面的文章，以及列出来的参考文献：

3 Z.Galil. On Improving the Worst Case Running Time of the Boyer-Moore String Matching Algorithm, *Communications of ACM*. Vol.22(1979), pp.505～508

4 L.Guibas and A.M.Odlyzko. A New Proof of the Linearity of the Boyer- Moore String Searching Algorithm, *SIAM Journal on Computing*. Vol.9(1980), pp.672～682

G.Barth 在 1984 年作过一项分析，指出在平均情况下，KMP 方法不会比一般做法快多少，因为在这种场合，比较时不相等的可能性比相等的可能性大，因此使 KMP 的优势降低。但对 BM 法而言，如果真的是常常不相等的话，大约比较 n/m 次就知道 pat[ ]不在 text[ ]中了。Barth 的文章是：

Download at http://www.pin5i.com/

5 G.Barth. An Analytical Comparison of Two String Searching Algorithms, *Information Processing Letters*. Vol.18(1984), pp.249～256

【程序】

程序 15-5（BM.C）

```
/* ------------------------------------------------------------ */
/* FUNCTION BM :                                                */
/*   The Boyer-Moore String Searching Program.  Given a         */
/* text string text[] and a pattern string pat[], this          */
/* function will find the first occurrence of pat[] in          */
/* test[] by using the naive Boyer-Moore algorithm.             */
/*                                                              */
/* Copyright Ching-Kuang Shene          July/18/1989            */
/* ------------------------------------------------------------ */

#include <string.h>          /* for strlen()                    */

#define  NOT_FOUND     -1

void get_jump(unsigned char [], int []);
int  BM(unsigned char [], unsigned char []);

int  BM(unsigned char text[], unsigned char pat[])
{
     int  jump_tab[256];
     int  t_len = strlen(text);
     int  p_len = strlen(pat);
     int  i, j, k;

     get_jump(pat, jump_tab);    /* compute the jump table       */
     for (i = p_len - 1; i < t_len; = { /* comp. downward         */
         for (j=p_len-1, k=i; j >= 0 && text[k] == pat[j]; k--,j--)
             ;
         if (j < 0=              /* if pat[] exhausted ?         */
             return k + 1;       /* YES, pattern found.          */
         else                    /* NO, update using jump_tab    */
             i += jump_tab[text[i]];
     }
         return NOT_FOUND;       /* text[] exhausted.            */
}
```

Download at http://www.pin5i.com/


```
/* ----------------------------------------------------------------- */
/* FUNCTION get_jump :                                               */
/*   Given the pattern string pat[], this function                   */
/* computes a jump table, jump_tab[], by look at each char           */
/* in the pattern string.                                            */
/* ----------------------------------------------------------------- */

void  get_jump(unsigned char pat[], int jump_tab[])
{
     int  length = strlen(pat);
     int  i;

     for (i = 1; i < 256; i++=      /* assume length is the gap*/
         jump_tab[i] = length;
     for (i = 0; i < length-1; i++=/* now adjust them.          */
         jump_tab[(int)pat[i]] = length - i - 1;
}
```

问题 6.6 所谓的 h-序列（RH_SEQ.C,H_SEQ.C）

题目是要写一个程序，接收一个字符串，辨认它是不是一个 h-序列。所谓的 h-序列是这样定义的：第一，0 这个数字符号是一个 h-序列；第二，任何的 h-序列如果不是一个 0 的话，就是从 1 开始，后面跟着两个 h-序列。

【解答】

学过编译程序的朋友一定会觉得这个问题太简单了，因为 h-序列的定义就是一个 BNF。只要写一个叫做 h_sequence()的函数，它先看看目前的符号是不是‘0’，如果是，就返回 YES，表示找到了一个 h-序列；但若目前的符号不是‘0’，而是一个‘1’，就调用自己从下一个符号起找出一个 h-序列，如果成功地找到了，再从下一个符号起找另一个 h-序列，如果两者都成功了，就返回 YES，或返回 NO。主控的程序是 h_sequence()，它用一个外在变量 cursor 来指出目前正处理到的符号的脚码。自然地它的值从 0 开始，每找到一个合适的符号就向右移，指向下一个待处理的符号。注意，当 h_seq()返回 YES 之后，h_sequence()还要查一查 cursor 是否指向最后一个符号，如果不是，那就表示字符串中只在前方的某一段是个 h-序列而已，后方还有剩下来的未处理的符号，这就不是个 h-序列。例如 10000，100 可以化简成一个 h-序列，但后面的两个 0 就造成了困难。RH_SEQ.C 就是这样的程序。

不过，事实上我们所期望完成的，却不是这样的程序，而是希望能够从 h-序列的定义写出一个简短的程序，因为用上面的递归方式来处理这一问题是有一点小题大做了。看看 h-序列的定义，一旦看到了一个‘0’，就表示找到了一个 h-序列；但若看到了一个‘1’，那么在这个‘1’的后面就期望会出现两个 h-序列。因此，准备一个计数用的变量 count，

Download at http://www.pin5i.com/

用它来表示在目前这个位置的后面，期望会找到多少个 h-序列。当然，在一开始时，count 的值为 1，因为期望输入的字符串从头到尾包含了一个完整的 h-序列。于是，当在处理的过程中看到了一个‘0’，就找到了一个 h-序列，所以把 count 减去来反应这一事件。但若见到‘1’，要把 count 加上 1。为什么呢？原来期望有 count 个，而在‘1’后面又应该有两个 h-序列，因此当‘1’出现时 count 应该减去 1（因为‘1’起头的 h-序列已经找到了），再加上跟在‘1’后面的那两个还没出现的 h-序列，就等于把 count 加上 1。

反复这个过程，到所有符号都处理完的时候，如果 count 是 0，那就表示期望出现的 h-序列都出现过了，所以输入字符串就是正确的 h-序列。count 大于 0 或小于 0 的意义是什么呢？留做习题。在后面的 H_SEQ.C 就是用这个技巧写成的程序。

【习题】

（1）请问，在 H_SEQ.C 中的 h_sequence()函数中为什么要返回 count 为 0，而且 i 大于等于 length 的值呢？少其中一个条件可以吗？为什么？请说明理由。

（2）如果把 h-序列的定义扩充成 H-序列如下：h-序列可以是一个‘0’，或是一个‘1’后面跟着两个 h-序列；或者是一个‘2’后面跟着 3 个 h-序列。请各写一个递归与非递归的程序来辨认 h-序列。

（3）如果 Q-序列是这样定义的：一个 Q-序列可以是一个‘0’，或者是两个 Q-序列后面再跟着一个‘1’，请写一个程序（递归或非递归）来辨认 Q-序列；001、0010011 都是 Q-序列，但 0101 却不是。

提示：有很简单的做法，几乎可以不必写程序。如果读者愿意写，不妨试一试使用堆栈的技巧。

【程序】

程序 15-6（1）（RH_SEQ.C）

```
/* ------------------------------------------------------------------- */
/* FUNCTION  h-sequence :                                              */
/*   The recursive h-sequence recognition function. It                 */
/* employs a recursive decent technique.                               */
/*                                                                     */
/* Copyright Ching-Kuang Shene          Aug/12/1989                    */
/* ------------------------------------------------------------------- */

#include <tring.h>      /* for strlen()                                */

#define  YES 1
#define  NO  0
int  h_seq(char []);    /* function prototype                          */
int  cursor;            /* the working cursor                          */

/* ------------------------------------------------------------------- */
```

Download at http://www.pin5i.com/


```
/* FUNCTION  h_sequence :                                              */
/*   The control routine.                                              */
/* ------------------------------------------------------------        */

int  h_sequence(char x[])
{
     int  length = strlen(x);
     cursor = 0;                        /* set working cursor to 0  */
     if (h_seq(x) == YES)               /* does it recognize prefix?*/
         if (cursor == length - 1)      /* YES, length exhaust      */
             return YES;                /* YES, a h-sequence.....   */
     return NO;                         /* NO.                      */
}

/* ------------------------------------------------------------        */
/* FUNCTION  h_seq :                                                   */
/*   The working routine by using recursive decent                     */
/* parsing technique.                                                  */
/* ------------------------------------------------------------        */

int  h_seq(char x[])
{
     switch (x[cursor]) {               /* for the current char.    */
         case '0' : return YES;         /* a '0' is a h-seq.        */
         case '1' : cursor++;           /* a '1' has to be          */
             if (h_seq(x) == YES) {     /* followed by              */
                 cursor++;              /* two h-seq's              */
                 if (h_seq(x) == YES)
                     return YES;
             }
             return NO;
         default  : return NO;
     }
}
```

程序 15-6（2）（H_SEQ.C）

```
/* ------------------------------------------------------------        */
/* FUNCTION  h_sequence :                                              */
/*   A h-sequence is defined as follows :                              */
/*   <h-seq> -> 0                                                      */
/*   <h-seq> -> 1 <h-seq> <h-seq>                                      */
/* This function accepts a sequence consisting of 0's and              */
/* 1's, then check to see if it is a h-sequence of not                 */
/* by using iterative technique.                                       */
/*                                                                     */
/* Copyright Ching-Kuang Shene          Aug/12/1989                    */
```

Download at http://www.pin5i.com/


```
/* ----------------------------------------------------------------- */

#include <string.h>      /* for strlen()                             */

#define  YES 1
#define  NO  0

int  h_sequence(char x[])
{
     int  length = strlen(x);
     int  count;                    /* counting for # of h-seq      */
     int  i;

     for (count = 1, i = 0; count != 0 && i < length; i++=
         switch (x[i]) {
             case '0' : count--; /* a '0' means one less            */
                 break;
             case '1' : count++; /* a '1' starts a new              */
                 break;
             default  : return NO;
         }
     return count == 0 && i >= length;
}
```


问题 6.7 寻找部分序列（SUBSEQ.C）

如果 s 是一个字符串，把其中（在任何位置）的符号去掉，留下来的内容就是 s 的一个子序列。比如说，如果 s 的内容是“abcdefg”，去掉 b、d、f，留下“aceg”；去掉 e、f、g，留下“abcde”；去掉 a、b、c、e、g 得到“df”。于是“aceg”、“abcde”与“df”都是原来字符串“abcdefg”的子序列（Subsequence），或说是部分序列。请写一个函数，接收字符串 text[ ]与 pat[ ]，看看 pat[ ]是否为 text[ ]的子序列，并且把 pat[ ]在 text[ ]中各符号的位置记录下来。

【解答】

因为要在 text[ ]中找出一个与 pat[ ]吻合的子序列，千万不要想到把 text[ ]中元素删除来迁就定义。只要想到，把它们跳过去，就相当于删除操作就行了。正因为如此，当处理到 pat[j]时，在 text[ ]中已经做到 text[i]，于是就固定 pat[j]，从 text[i+1]起，去比较 text[i+2], text[i+3],…，如果自此之后找不出与 pat[j]相同的符号，那当然 pat[ ]就不会是 text[ ]的子序列，如果在处理到某个 k 时，text[k]与 pat[j]相同，因此对 pat[j+1]而言，就从 text[k+1]找起了。

Download at http://www.pin5i.com/

如图 15-2 所示，对 pat[0]而言，在 text[ ]中到了 i 才找到与 pat[0]相同的，因此在 text[ ]中 i 之前就没有 pat[0]这个符号，接着 pat[1]就从 text[i+1]比起，假设到了 text[j]发现与 pat[1]相同，…。到了 pat[ ]的最后一个元素都能够在 text[ ]中找出来，自然地 pat[ ]就是 text[ ]的子序列，只要把图中不与 pat[ ]中元素对应的部分删除，就可以由 text[ ]造出 pat[ ]。注意，这个方法所找出的，是在 text[ ]中最左边的子序列，因为在 text[ ]中与 pat[ ]最后一个符号相同的那个符号后面，或许还可以造出 pat[ ]也说不定。

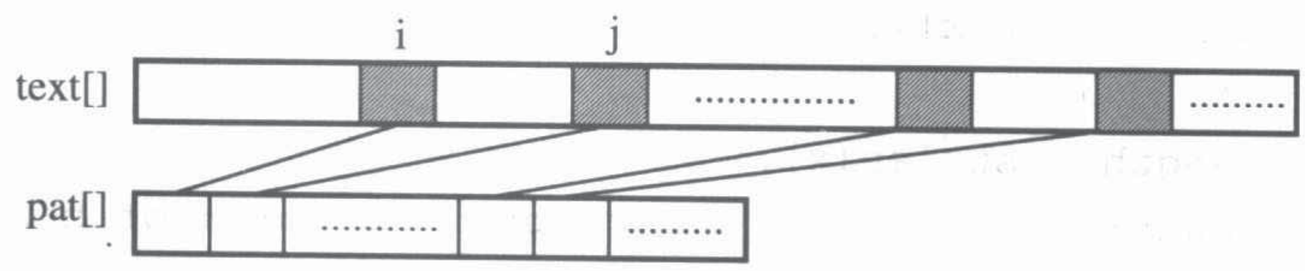


图 15-2

这个方法一共用了多少次比较呢？最多 m 次，m 是 text[ ]的长度，而且不比 pat[ ]的长度小，为什么？想想 text[ ]中的符号每一个都恰好与 pat[ ]中的某一个比较一次就不难明白，所以这是一个线性的程序。

【习题】

（1）在理论上，虚字符串——就是空无一物的字符串，是任何字符串的子序列。请问：程序可以处理 text[ ]或 pat[ ]，或同时是虚字符串的情况吗？如果不能，请修改它。

（2）请修改这个程序，使它可以找出在 text[ ]中最右边的子序列。

（3）请修改这个程序，使它可以找出跨距最大的子序列。所谓的跨距是假设 pat[ ]是 text[ ]中的一个子序列，pat[0]与 text[i]对应，pat[ ]的最后一个元素与 text[j]对应，于是 i 与 j 之间的距离，就是跨距。程序要找出跨距最大的那一个子序列，不过程序还得是线性的。（提示：可以试一试在两端同时进行）

（4）这个程序能否轻易地改成找出 text[ ]中所有对应着 pat[ ]的子序列吗？为什么？如果可以，请试试看，程序还是线性的吗？如果不能，那么问题何在？

【程序】

程序 15-7（SUBSEQ.C）

```
/* ------------------------------------------------------------------ */
/* FUNCTION subsequence :                                             */
/*   Given a text string text[] and a pattern string                  */
/* pat[], this function picks up a subsequence of text[]              */
/* which matches pat[] and stores the corresponding                   */
/* locations into array loc[].                                        */
/*                                                                    */
/* Copyright Ching-Kuang Shene      July/26/1989                      */
/* ------------------------------------------------------------------ */

#define  FOUND   1
```

Download at http://www.pin5i.com/


```
#define  NOT_FOUND   0

int  subsequence(char text[], char pat[], int loc[])
{
     int  t_len = strlen(text);
     int  p_len = strlen(pat);
     int  i, j;

     if (p_len > t_len)      /* is pattern too long ?            */
         return NOT_FOUND;       /* YES, return.                 */

     for (i = j = 0; i < t_len && j < p_len; j++= {
         for ( ; i < t_len && text[i] != pat[j]; i++=
             ;                   /* can we find pat[j] ?         */
         if (i >= t_len)         /* text[] exhausted ?           */
             return NOT_FOUND;   /* YES, not found.              */
         else                    /* NO, pat[j] is located at     */
             loc[j] = i;         /* i-th pos. of text[]          */
     }
     return FOUND;
}
```

问题 6.8　最长重复部分序列（MAX_REPS.C）

如果 t 与 p 是两个字符串，把 p 中的每一个符号重复写 i 次，就得到一个新字符串 p^i，p^i 是 t 的子序列吗？请写一个程序，找出最大的、使 p^i 还是 t 的子序列的 i，如果 p 根本就不是 t 的子序列，则程序返回 0。

【解答】

这个问题有两点需要克服，第一是使比较次数减少，第二是能不能不把 p^i 展开而只用 p 来操作。

先说如何降低比较的次数，关键所在就是二分法。首先，查一查 p 是不是 t 的子序列。如果不是，那就什么事都不要做了，因为连本身都不是，所有 p^i 也都不会是 t 的子序列；如果 p 是 t 的子序列，那么至少有了保证，在 $i \geq 1$ 时会有一个解答。但是在问题说明中又讲到过，i 的解答必定小于或等于 $|t|/|p|$，于是二分法的面目就呼之欲出了。

在 1 与 $|t|/|p|$ 之间的值之中，知道 1 是一个满足要求的值，虽然并不是最大值；不过依问题说明中的方法，去试 2,3,4…到了某个最大值，再往后 p^i 就不会是 t 的子序列了（为什么？）。在 1 与 $|t|/|p|$ 之间，1 是个可能解。于是，就可以利用 L 与 R，保证在 L 与 R 之间有一个令 p^i 是 t 的最长子序列的 i；在开始时，因为 1 正是一个可能的值，所以 L 是 1，

Download at http://www.pin5i.com/

R 是|t|/|p|。

在以后的工作过程中，L 与 R 之间一定有解，因此二分法就有用了。首先，令 M 是 L 与 R 的中点，亦即 (L+R)/2，看看 p^M 是不是 t 的子序列，因此就会有两种情形：

（1）p^M 是 t 的子序列：p^L 是 t 的子序列（在开始时 L 是 1，在往后的过程中也一直维持着这一层关系），M 大于或等于 L，因此在 L 与 M 之间不可能有极大值，p^M 就比 p^2 更靠近极大值。

因此，L 与 R 就更正为 M 与 R。

（2）p^M 不是 t 的子序列：这表示离开了极大值的可能范围，要把范围缩小；因为 L 到 R 之间会有极大值，而且 M 不是一个可能的解，因此可能的解只能在 L 的右边，R 与 L 就转换为 L 与 M。

这步程序就一直反复进行，每次都去掉一半左右的范围，到最后如果不是 L=R，就一定是 L+1=R。在前者 L 或 R 都是解答；而在后者，知道 L 是个可能解，但若 R 也是个解（亦即 p^R 是 t 的子序列），那么极大值就是 R 了，不然就是 L。

在程序中修改了 subsequence()这个函数，它多接收一个参数 number，通过这个来判定 p^{number} 是不是 t 的子序列，如果是子序列，loc[ ]中就是 p^{number} 在 t 中的对应位置。不过要注意，不能只返回 L 或 R，要一并返回 loc[ ]，但是在工作过程中，loc[ ]只有一个，因此无法随 L、M、R 的转换而更动。做法很简单，L 一定是个可能解，所以用 subsequence()去查 R 是不是解，如果是，不但 R 是极大值，loc[ ]的内容也是正确的。如果 R 不是极大值，就用 L 去做一次，因为 L 一定是个可能解，而且 L≤R，现在 R 不是极大值，自然地极大值就一定是 L 了。这一次对 subsequence()的调用，只不过是把正确的数据填回 loc[ ]而已。事实上是可以把对 subsequence()的调用次数降低的，就留作习题吧。

在 max_repetition()中，L、M 与 R 是用 low、mid 与 high 来表示的。

现在看看前述的第二个问题，如何不展开 p^i。其实这倒不难，从第一个问题就容易想出办法，让上一题的 subsequence()函数的第二个循环做若干次就行了。

如图 15-3 所示，当要处理 pat[j]时，当然要拿 text[ ]中下一个符号与 pat 相比较，在比较过程中就计算有多少次相等，因此若要查 p^4 是否为 t 的子序列时，相等 4 次就算是 pat[j]找到了相吻合的内容，而把指针分别移到 pat[ ]与 text[ ]的下一个元素去。不过要注意，loc[ ]得把所有相等的位置都记录下来才行。在上个问题中，loc[ ]长度与 pat[ ]相同，但是在这个问题中，它的长度就有可能到达 text[ ]的长度了。

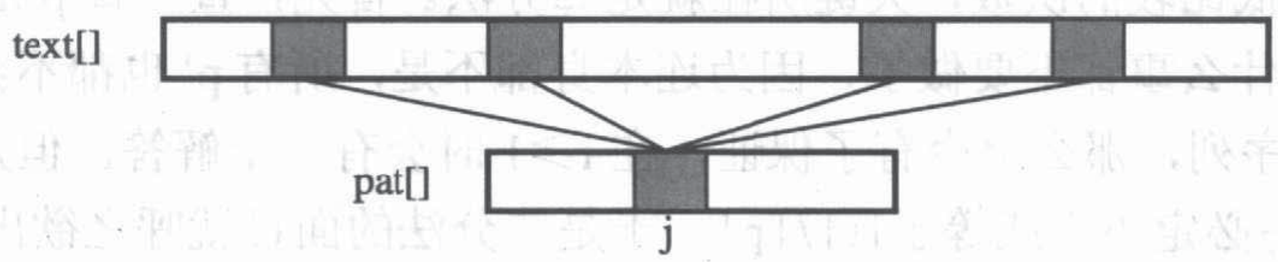


图 15-3

程序中其他部分变动不大，请读者自己研究。

Download at http://www.pin5i.com/

【习题】

（1）请证明这个程序只用了与$|t|\log_2(|t|/|p|)$成正比的比较次数，这不就比问题说明中的方法少了吗？而且程序也未必比较复杂、难写。

（2）如果p^i是t的一个子序列，p^{i+1}不是，那么p^{i+2}、p^{i+3}、…也都不是t的子序列。为什么？请证明它，这是本题解答的核心所在。

（3）请修改max_repetition()函数，降低调用subsequence()函数的次数。

（4）如果p^i是采用连接（Concatenation）的意义，也就是传统地把i个p连接成更长字符串。请用这个定义来重写这个程序。若t=“abcabcabc”，p=“abc”，那么p^3就是结果，因为$p^3=t$；又若 p=“ac”，那么p^3 = “acacac” 正是我们所要的解答，因此p^4就不是 t的子序列了。

【参考文献】

如果t与p的长度分别是n与m，解决此题一共处理了$n\log_2(n/m)$次比较，但事实上这个题目可以做得更好。在1987年Andranik Mirzaian的一篇文章中，他证明了与n（假设n>m）成正比的比较次数就足够了。换言之，这是可以做到线性的境界的；Mirzaian的解法不难，不妨参阅他的文章。

A.Mirzaian. A Halving Technique for the Longest Stuttering Subsequence Problem, *Information Processing Letters*. Vol.26(1987/1988), pp.71～75

【程序】

程序15-8（MAX_REPS.C）

```
/* ------------------------------------------------------------ */
/* FUNCTION max_repetition :                                    */
/*   Given a text string text[] and a pattern pat[], this       */
/* fucntion computes the maximum repetition number of the       */
/* pattern such that the expanded pattern is the longest        */
/* subsequence of the text.  NOTE the 'repetition' means        */
/* repeating individual characters, but the usual meaning       */
/* of concatenation.                                            */
/* Copyright Ching-Kuang Shene          July/27/1989            */
/* ------------------------------------------------------------ */

#define  FOUND   1
#define  NOT_FOUND   0

int  subsequence(char [], char [], int, int []);

int  max_repetition(char text[], char pat[], int loc[])
{
     int  t_len  = strlen(text);
```

Download at http://www.pin5i.com/


```
    int  p_len  = strlen(pat);
    int  low= 1;
    int  high   = t_len/p_len;
    int  mid;

    if (subsequence(text, pat, low, loc) == NOT_FOUND)
        return NOT_FOUND;

    while (low + 1 < high) {    /* there must a solution in     */
        mid = (low + high)/2;   /* the [low,high]               */
        if (subsequence(text, pat, mid, loc) == FOUND)
            low  = mid;
        else
            high = mid;
    }

    /* ------- do it again to recover loc[] array ------        */

    if (subsequence(text, pat, high, loc) == FOUND)
        return high;
    (void) subsequence(text, pat, low, loc);
    return low;
}


/* ------------------------------------------------------------ */
/* FUNCTION  subsequence :                                      */
/*   This is a modified version of the function appeared        */
/* earlier.  subsequence() accepts a text string text[],        */
/* a pattern string pat[] and repeated number 'number',         */
/* than determines if there exists a subsequence in text        */
/* of form pat[0]pat[0]...pat[0]pat[1]pat[1]...pat[1]...        */
/* That is each character in pat[] repeats 'number' times.      */
/* ------------------------------------------------------------ */
int  subsequence(char text[], char pat[], int number, int loc[])
{
    int  t_len = strlen(text);
    int  p_len = strlen(pat);
    int  count, index = 0;
    int  i, j;

    if (p_len > t_len)          /* is pattern too long ?        */
```

Download at http://www.pin5i.com/


```
        return NOT_FOUND;           /* YES, return.              */

    for (i = j = 0; i < t_len && j < p_len; j++) {
        count = 0;
        do {
                for (; i < t_len && text[i] != pat[j]; i++)
                ;                   /* can we find pat[j] ?      */
                if (i >= t_len)     /* text[] exhausted ?        */
                    return NOT_FOUND; /* YES, not found.         */
                else {
                    loc[index++] = i++;
                    count++;
                }
            } while (count < number);
        }
        return FOUND;
    }
```


问题 6.9 最长共同部分序列（LCS.C）

如果 $A=a_1a_2\cdots a_m$ 是一个长度为 m 的字符串，把其中的若干（可能是 0 个，也可能是 n）个符号去掉，而得到一个新字符串，这个新字符串就叫做 A 的部分序列（Subsequence）。例如，若 A=abc0123，那么 b02,abc123,b3,c,abc0123,ab12,…都是 A 的部分序列。

假设给了两个字符串 A 与 B，长度分别是 m 与 n，那么 A 与 B 就含有若干共同的部分序列，至少虚字符串（或说是空字符串）就是一个共同部分序列。所谓 C 是 A 与 B 的共同部分序列，指的是 C 是 A 的部分序列，C 也是 B 的部分序列。请写一个程序，把 A 与 B 的共同部分序列中最长的一个找出来。

这个问题一般都叫做最长共同部分序列（Longest Common Subsequence）问题，简称为 LCS。

【解答】

用一个矩阵 d[ ][ ]来保存程序处理过程的中间结果，换句话说，用 d[i][j]表示 $a_1a_2\cdots a_i$ 与 $b_1b_2\cdots b_j$ 这两个部分字符串中的最长共同部分序列长度，所以 d[m][n]中就是 $a_1a_2\cdots a_m$ 与 $b_1b_2\cdots b_n$ 的最长共同部分序列长度，这也就是所要求的答案。

不过，要如何才能算出 d[i][j]的值呢？在算 d[i][j]时，假设 d[i-1][j-1]、d[i-1][j]与 d[i][j-1]已经算出来了，于是：

（1）若 $a_i=b_j$，可以把 a_i（或 b_j）接到 $a_1a_2\cdots a_{i-1}$ 与 $b_1b_2\cdots b_{j-1}$ 的最长部分序列的后面，所以 $a_1a_2\cdots a_i$ 与 $b_1b_2\cdots b_j$ 的最长共同部分序列的长度，就是 $a_1a_2\cdots a_{i-1}$ 与 $b_1b_2\cdots b_{j-1}$

Download at http://www.pin5i.com/

的最长共同部分序列的长度加上 1。用 d[ ][ ]矩阵的术语来说，就是 d[i][j]=d[i-1][j-1]+1。

（2）若 $a_i \neq b_j$，就没有办法延长既有的最长共同部分序列了。换句话说，$a_1a_2\cdots a_i$ 与 $b_1b_2\cdots b_j$ 的最长共同部分序列长度与在此之前的最长共同部分序列长度相同。即 d[i][j]是 d[i-1][j-1]、d[i-1][j]与 d[i][j-1]中最大的一个；道理是这样的，把 A 与 B 从左到右地处理，在处理到 a_i 与 b_j 时，在此之前最靠近 a_i 与 b_j 的不是 $a_1a_2\cdots a_{i-1}$ 与 $b_1b_2\cdots b_{j-1}$，或 $a_1a_2\cdots a_{i-1}$ 与 $b_1b_2\cdots b_j$，或者是 $a_1a_2\cdots a_i$ 与 $b_1b_2\cdots b_{j-1}$ 的最长共同部分序列的长度。但求 d[i-1][j]与 d[i][j-1]的极大值给 d[i][j]就够了，不必管 d[i-1][j-1]，为什么呢？

因为在算 d[i][j]时，要先算出 d[i-1][j-1]（第一种情况），以及 d[i-1][j]与 d[i][j-1]（第二种情况），因此计算 d[ ][ ]矩阵中元素的顺序就要是一列一列，或一行一行地计算了。

d[ ][ ]矩阵要多大？因为 d[i][j]是 $a_1a_2\cdots a_i$ 与 $b_1b_2\cdots b_j$ 的最长共同部分序列的长度，所以答案在 d[m][n]，m 列 n 行少不了。不过，在算第一列与第一行的元素（d[i][1]与 d[1][j]）时，要用到 d[i][0]与 d[0][j]。d[i][0]是 $a_1a_2\cdots a_i$ 与虚字符串（长度为 0）的最长共同部分序列的长度，那自然是 0，把 a_1 与 a_i 都删除之后才能得到虚字符串，因此 d[i][0]是 0；同理 d[0][j]也是 0。d[ ][ ]矩阵要 m+1 列 n+1 行，而第 0 列与第 0 行是 0。

有了这些初值之后，使用上面的观点，一列一列或一行一行地算，到最后 d[m][n]中就是最长共同部分序列的长度了。例如，如果 A=“abcd”，B=“xbadz”，一眼看去就可以看出最长共同部分序列是 bd。在开始时，d[ ][ ]如图 15-4 所示。

采用一列一列的算法，a 与 x 相比不同，因此把 d[1][0]与 d[0][1]的极大值（=0）存入 d[1][1]，一直反复，得到结果如图 15-5 所示。

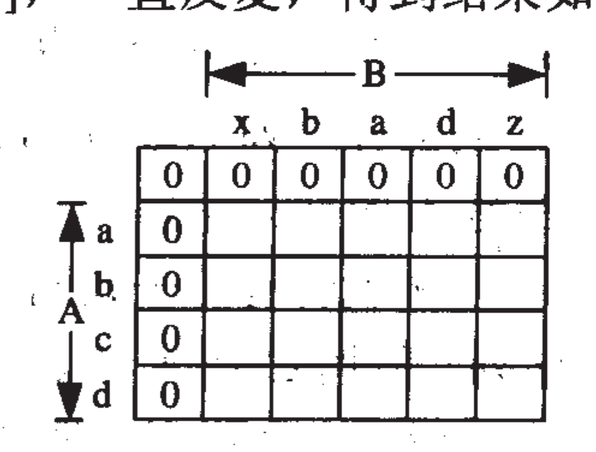


图 15-4

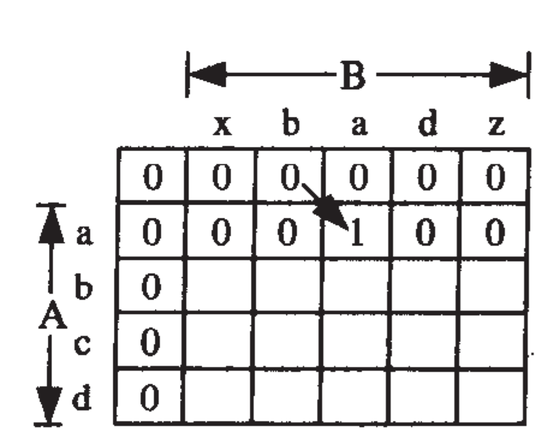


图 15-5

现在算第二列，b 与 x 比较时，d[2][1]是 0，但 b 与 b 相等，所以 d[2][2]为 1。但接下来的 b 与 a 不相等，故 d[2][3]的值是 d[1][3]与 d[2][2]的极大值，两者都是 1，所以结果为 1，如图 15-6 所示。

再算 c 那一列，如图 15-7 所示。

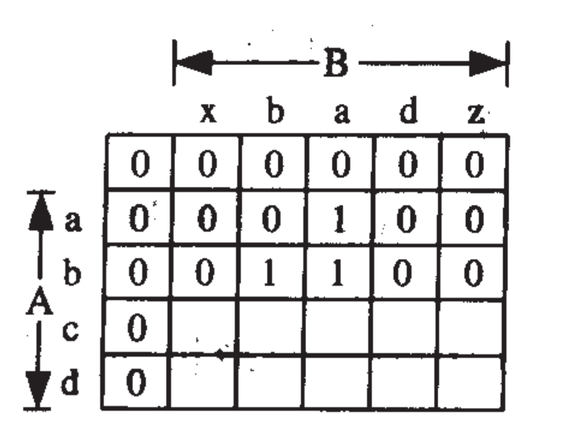


图 15-6

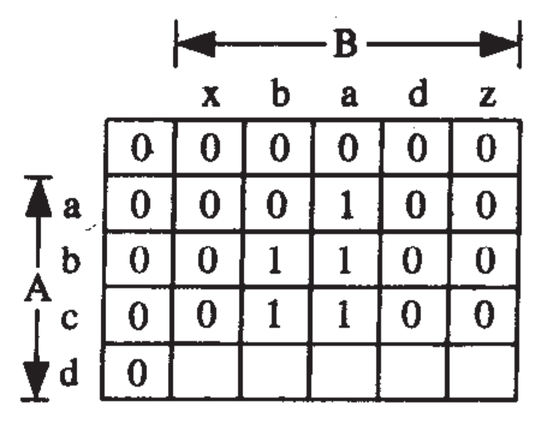


图 15-7

Download at http://www.pin5i.com/

最后算 d 那一列，d[4][4]是由 d 与 d 比较而决定的，所以得到 2，如图 15-8 所示。

		x	b	a	d	z
	0	0	0	0	0	0
a	0	0	0	1	0	0
b	0	0	1	1	0	0
c	0	0	1	1	0	0
d	0	0	0	0	2	2

图 15-8

所以看 d[4][5]，就得到 A 与 B 的最长共同部分序列的长度（2）。

那么，要如何才能找出最长共同部分序列呢？只有在某个 a_i 与某个 b_j 相等时，才会得到延长的机会。所以，只要从 d[m][n]起倒着查回去，找出在哪里把 d[i-1][j-1]加 1 变为 d[i][j]就行了。回想一下 d[i][j]的值的来源有 3 种，如图 15-9 所示。

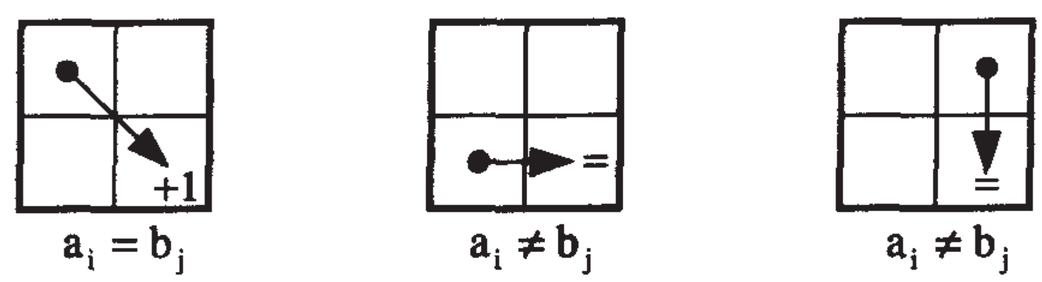


图 15-9

若 d[i][j]与 d[i-1][j]相等，这是上面第三种情况，亦即 d[i][j]的值来自 d[i-1][j]，所以应该把 i 减去 1，向上移一列继续查。若 d[i][j]与 d[i][j-1]相等，按照相同的理由，就是 j-1，向左移一行继续查。但若 d[i][j]与 d[i-1][j]、d[i][j-1]都不相等，那就表示 d[i][j]是从 d[i-1][j-1]加上 1 而来的，因此有 $a_i = b_j$，所以把 i 与 j 都减去 1，这是图 15-9 中的第一种情况。不但如此，还要记住 a_i（或 b_j）。于是一直反复这个步骤，直到有一个 i 或 j 变成 0，这就表示有一个字符串全部都查完了，工作就可以停下来。于是最长共同部分序列就找出来了。但是不要忘了一点，刚才找出来的最长共同部分序列是方向相反的，还要把它倒过来才行。

看看前面的最终结果，下面就是一个可能的检查路径：

d[4][5]→<u>d[4][4]</u>→d[3][3]→d[2][3]→<u>d[1][2]</u>→d[0][2]

画了底线的地方分别是由 d 与 a 产生的，所以 ad 就是结果。

下面的程序分成 3 部分，前两个 for 是为 d[ ][ ]矩阵第 0 列第 0 行给定初值 0，接下来两个套在一起的 for 就一列一列地检查，并且算出每一个 d[i][j]的值。第三部分则是从 d[ ][ ]右下角起反过来追踪出加 1 的所在，找到了一个就把它用反过来的次序存到数组中，当 i 与 j 有一个是 0 时，就停止工作而返回结果了。

【习题】

（1）在第二部分计算 d[i][j]时，为什么当 $a_i \neq b_j$ 时可以不必管 d[i-1][j-1]的值是什么呢？

（2）请计算，找出 abcabc 与 abc，aabbcc 与 abc 的最长共同部分序列。

（3）在程序的第三部分中，是先查 d[i][j]是否与 d[i-1][j]相等的，换句话说永远先向

Download at http://www.pin5i.com/

上走；不过程序自然也可以改成先检查 d[i][j]是否与 d[i][j-1]相等，因此就先向左走了。请问，这有关系吗？

（4）续题（3），如果改成先查 d[i][j]是否是由 d[i-1][j-1]加上 1 而来，程序能够正常运行吗？为什么？请说明理由。

（5）这个程序是把最长共同部分序列返回来的，但有时候，这个序列在给定的那两个字符串中的对应位置要重要些；请修改程序，返回这些位置（一定要做这题，用不了 5 分钟的）。

（6）这个程序在最坏的情况下一共处理了 A 与 B 的符号多少次？

（7）很明显地，在算 d[i][j]时，只要 d[i-1][ ]这一列，与 d[i][0]、d[i][1]、…、d[i][j-1]这些元素就足够了，换句话说，d[i][0]矩阵其实只需要用到两列而已；而 d[ ][ ]中的其他信息是留下来为了要反过来查出最长共同部分序列用的。如果程序只用两列，再加上一些有限的位置，而不必用 m+1 列 n+1 行的矩阵，有没有办法找出最长共同部分序列呢？

【参考文献】

这个问题最早出现在 1974 年 R.A.Wagner 与 M.J.Fischer 的一篇文章的最后，此处就采用他们两人的方法。在 1975 年 D.S.Hirschberg 的文章中作了进一步的探讨，自此之后，这就成为一个很引人注目的问题了，相关的研究可以在本题与下一题的参考文献中查到。另外，大多数讲算法（Algorithm）的书中也都会提到这个方法，但很多都没有提到如何把最长共同部分序列找出来；因此，看看算法的书，特别是有关动态规划（Dynamic Programming）的部分，对写作这一类型的程序是很有帮助的。

D.S.Hirschberg. A Linear Space Algorithm for Computing Maximal Common Subsequence, *Communications of ACM*. Vol.18(1975), pp.341～343

【程序】

程序 15-9（LCS.C）

```
/* ------------------------------------------------------------------ */
/* FUNCTION  longest_common_subsequence :                             */
/*   Given two strings, this function computes one of                 */
/* the longest common subsequence of the given strings.               */
/* Here we use a standard dynamic programming technique,              */
/* but there exist faster methods.                                    */
/*                                                                    */
/* Copyright Ching-Kuang Shene          Aug/01/1989                   */
/* ------------------------------------------------------------------ */

#include  <stdlib.h>
#include  <string.h>

void longest_common_subsequence(char *a, char *b, char *result)
{
```

Download at http://www.pin5i.com/


```
void longest_common_subsequence(char *a, char *b, char *result)

    int  **d;                        /* working matrix              */
    int  m = strlen(a), n = strlen(b); /* string length             */
    int  count, i, j;

    /* ----- get memory and initialize dope vector -----          */

    d    = (int **) malloc(sizeof(int)*(m+1));
    d[0] = (int *) malloc(sizeof(int)*(m+1)*(n+1));
    for (i = 1; i <= m; i++)
        d[i] = d[i-1] + n + 1;
    /* ------ Clear the dynamic programming table ------          */

    d[0][0] = 0;
    for (i = 1; i <= m; d[i][0] = 0, i++)
        ;
    or (j = 1; j <= n; d[0][j] = 0, j++)
        ;

    /* --------- dynamic programming technique ---------          */

    for (i = 1; i <= m; i++)          /* for each pair of          */
        for (j = 1; j <= n; j++)      /* chars in strings:         */
            if (a[i-1] == b[j-1])     /* are they equal ?          */
                d[i][j] = d[i-1][j-1] + 1; /* one more             */
                else if (d[i][j-1] > d[i-1][j])/* or pick          */
                d[i][j] = d[i][j-1];          /* a smaller         */
            else                              /* neighbor          */
                d[i][j] = d[i-1][j];

    /* ------- finally display the subsequence ------             */

    count   = d[m][n];                /* d[m][n] is the length     */
    result[count]   = '\0';           /* put a EOS there           */
    for (i = m,j = n; (i != 0) && (j != 0);)   /* R->L odr         */
        if (d[i][j] == d[i-1][j])     /* are they contribute       */
            i--;                      /* equal counting ? NO,      */
        else if (d[i][j] == d[i][j-1])
            j--;                      /* NO, backup                */
```

Download at http://www.pin5i.com/


```
            else {                    /* YES, save this character    */
                    result[--count] = a[i-1];
                    i--, j--;             /* backup both string          */
                }
            free(d[0]);
            free(d);
        }
}
```

问题 6.10　字符串编修（STREDIT.C）

已知两个字符串 s 与 t，要研究如何把字符串 s 经由一连串修改后变成 t。能够使用的就是插入一个符号，以及删除一个符号。把某个符号换成另一个，就可以通过先把它删除再在原地插入所需的符号来完成。请写一个程序，接收 s 与 t，找出如何才能够在最少步骤之下把 s 改成 t。

【解答】

其实这个题目与最长共同部分序列（Longest Common Subsequence，简称 LCS）没有多大分别，所以在向下看之前应该先了解前面的 LCS.C 这个程序。

还是用一个表 cost[ ][ ]来存放处理过程，如果要把 s 字符串修改成 t，而 s 与 t 的长度分别是 m 与 n，cost[ ][ ]这个矩阵就要有 m+1 列 n+1 行。第 i 列第 j 行的元素，cost[i][j]，表示把 $s_1s_2\cdots s_i$ 改写成 $t_1t_2\cdots t_j$ 所需要的最少修改数目。这是很容易求出来的，因为对 s_i 与 t_j 而言，这不外乎两种情形：

（1）$s_i=t_j$：在这种情况下，就不必有任何操作，把 $s_1s_2\cdots s_i$ 改成 $t_1t_2\cdots t_j$ 的修改数目就与把 $s_1s_2\cdots s_{i-1}$ 改成 $t_1t_2\cdots t_{j-1}$ 的修改数目一样，换言之，cost[i][j]就等于 cost[i-1][j-1]。

（2）$s_i\neq t_j$：这就要分成几个可能性来探讨。第一个可能性是从 $s_1s_2\cdots s_{i-1}s_i$ 中把 s_i 删除，得到 $s_1s_2\cdots s_{i-1}$，而希望在以后过程中会把 $s_{i+1}s_{i+2}\cdots s_m$ 改为 $t_1t_2\cdots t_j$。然而把 $s_1s_2\cdots s_{i-1}$ 改成 $t_1t_2\cdots t_j$ 的修改数目在 cost[i-1][j]中，值为 cost[i-1][j]加上 1。第二个可能性是在 $t_1t_2\cdots t_{j-1}$ 的后方插入一个 s_i，得到 $t_1t_2\cdots t_{j-1}\ s_i$，这样做的修改次数是 $s_1s_2\cdots s_i$ 改成 $t_1t_2\cdots t_{j-1}$ 的修改次数再加上 1，因此修改次数是 cost[i][j-1]+1。第三个可能性，可以把 t_j 更换成 s_i，亦即删除 s_i，再插入 t_j，所以修改次数是把 $s_1s_2\cdots s_{i-1}$ 改成 $t_1t_2\cdots t_{j-1}$ 的数目再加上 2，亦即 cost[i-1][j-1]+2。题目是要找出最低的修改次数，因此 cost[i][j]的值是 cost [i-1][j]+1、cost[i][j-1]+1、cost[i-1][j-1]+2 这 3 个的极小值。

在上面的讨论中，当 $s_i\neq t_i$ 时，已知 cost[i-1][j]、cost[i][j-1]以及 cost[i-1][j-1]，如果要算cost[i][j]，那么就一列一列从左向右算，这样在算cost[i][j]时，第i列以上的元素cost[i-1][ ]就都有了值，而且又是从左向右，cost[i][j-1]也有了值，自然cost[i][j]就算出来了。

但问题是，第一列与第一行从何而来？在算第一列的元素 cost[1][j]时需要 cost[0][j]、

Download at http://www.pin5i.com/

cost[0][j-1]与 cost[1][j-1]，cost[0][j]与 cost[0][j-1]就要先有值才行。cost[0][j]代表什么？如果回想 cost[i][j]的意义，它是表示把 $s_1s_2\cdots s_i$ 改成 $t_1t_2\cdots t_j$ 的修改次数，自然 cost[i][0]就是 s 什么都没有时而改成 $t_1t_2\cdots t_j$ 的修改次数了。因为 s 什么都没有，要造出 $t_1t_2\cdots t_j$ 的惟一办法就是插入一个 t_j，所以 cost[0][j]就是 j。同理，cost[0][0]是表示把 $s_1s_2\cdots s_i$ 变成什么都没有的 t，惟一的办法就是删除 $s_1s_2\cdots s_i$。cost[0][j]是多少？0，为什么？因为从什么都没有改成什么都没有，不需有任何修改，所以是 0。于是 cost[ ][ ]矩阵如何算的方法就有了。

在程序 STREDIT.C 中，malloc()具有足够大的区域，令 cost[i] 指向这个区域中的第 i 列，每一列有 t_len+1 个元素，此处 t_len 是 t 字符串（程序中是 target[ ]）的长度。接下来的两个 for 是把 cost[ ][ ]第 0 列与第 0 行的值预备好。下面就是求 cost[i][j]了。注意，因为 C 的数组从 0 起，所以 source[i]与 target[j]事实上是第 i+1、j+1 个元素，是在算 cost[i+1][j+1]。接下来的 for 循环是用来找出 source[ ]与 target[ ]中相等的位置，在讲 LCS.C 时讨论过了。这些位置分别存放在 s[ ]与 t[ ]中，因为是从右往前查的，所以还得把它们反转过来才行。

edit_op()是用来显示修改操作，它的操作很简单；例如，如果 source[s [0]]与 target[t[0]]是相等的，就是为了要把 source[ ]改成 target[ ]，于是就要删除 source[0]～source[s[0]-1]，插入 target[0]～target[t[0]-1]，保留 source[s[0]]。接着删除 source[s[0]+1]～source[s[1]-1]，插入 target[t[0]+1]～target[t[1]-1]，保留 source[s[1]]，…，反复这个操作就行了。这些删除、插入与保留的动作，合起来就可以把 source[ ]修改成 target[ ]。

举例说明，如果 s=“abcdef”、t=“xbyzek”，于是 cost[ ][ ]矩阵如图 15-10 所示。

		x	b	y	z	e	k
	0	1	2	3	4	5	6
a	1	2	3	4	5	6	7
b	2	3	2	3	4	5	6
c	3	4	3	4	5	6	7
d	4	5	4	5	6	7	8
e	5	6	5	6	7	6	7
f	6	7	6	7	8	7	8

图 15-10

追踪一下矩阵中由 edit()函数最后一个 for 所收集下来的值，s[ ]与 t[ ]的结果为：

```
s[ ] = { 4, 1}
t[ ] = { 4, 1}
```

把它们反过来排得到：

```
s[ ] = { 1, 4}
t[ ] = { 1, 4}
```

因此操作就是删除 s[0]、插入 t[0]、保留 s[1]，得到“xb”。再删除 s[2]、s[3]、加入 t[2]、t[3]、保留 t[4]，得到“xbyze”，最后删除 s[5]，加入 t[5]，结果就是“xbyzek”。

【习题】

（1）请拿这个问题与最长部分序列（LCS.C）相比较，有什么相同的地方，又有哪些

Download at http://www.pin5i.com/

不同？

（2）请自程序中增加一个输出，指明一共要有多少个操作。不要在 edit_op()中一个一个地算，cost[ ][ ]中有这个值，在何处？为什么？

（3）已知字符串 s，又有字符串 p，请修改上面的程序，让它可以找出在 s 中与 p 最多只有 k 个元素不同的部分字符串。例如，若 s=“abcde”，而 p=“bxd”，又 k=1，于是在 s 中就是“bcd”这一段与 p 最多只有 k 个元素不同。真的要这样做吗？也请设计根本不必用 cost[ ][ ]这个表格的程序，效率如何？

（4）这个程序事实上可以不必用那么多内存的。如果 s 与 t 各有 m 个与 n 个元素，使用(m+1)(n+1)个元素的表格实在是多了一些。请修改程序，看能否使用 2(n+1)个元素就可以做出结果？计算的步骤会多出多少？

【参考文献】

把一个字符串修改成另一个字符串，有不少人在差不多同一时刻发表过类似的结果，此方法是 R.A.Wagner 与 M.J.Fischer 在 1974 年提出来的，其他的人的著作笔者尚未读过。这个方法一共要查与 m・n 成正比个元素。字符串的比较、修改等操作，在分子化学上有相当的应用，下面由 David Sankoff 与 Joseph B.Kruskal 所编的论文集中有不少有趣而且有用的参考文献。

1 D.Sankoff and J.B.Kruskal (ed.). Time Warps, String Edits，and Macromolecules: The Theory and Practice of Sequence Comparison, Addison-Wesley, 1983

2 R.A.Wagner and M.J.Fischer. The String-to-String Correction Problem, Journal of ACM. Vol.21(1974), pp.168～173

【程序】

程序 15-10（STREDIT.C）

```
/* ------------------------------------------------------------------ */
/* FUNCTION  edit :                                                   */
/*   Given two strings and costs of inserting, deleting               */
/* and exchanging a character, this function computes a               */
/* longest common subsequence and puts those common                   */
/* positions in array s[] and t[].  Then another function             */
/* can print all editing operations by looking at these               */
/* positions.                                                         */
/*                                                                    */
/* Copyright Ching-Kuang Shene          Aug/15/1989                   */
/* ------------------------------------------------------------------ */

#include <stdio.h>       /* for printf()                              */
#include <stdlib.h>      /* for malloc() and free()                   */
#include <string.h>      /* for strlen(), memncpy()                   */
```

Download at http://www.pin5i.com/


```
void edit(char [], char [], int [], int [], int *);
void edit_op(char [], char [], int [], int [], int);
void reverse(int [], int);

#define  INSERT_COST     1 /*    cost for inserting a char       */
#define  DELETE_COST     1 /*    cost for deleting a char        */
#define  EXCHANGE_COST   2 /*    cost for exchanging chars       */

#define  MIN(x, y, z)     ((x) <= (y) ?                          \
                              ((x) <= (z) ? (x) : (z)) :
                              ((y) <= (z) ? (y) : (z))          \
                          )

void edit(char source[], char target[], int s[], int t[], int *count)
 {
void edit(char source[], char target[], int s[], int t[], int *count)
      int  s_len = strlen(source);
      int  t_len = strlen(target);
      int  insert_t, delete_s, exchange;
      int  i, j, no;
      int  **cost;

      /* -------------- get working memory ---------------       */

      cost    = (int **) malloc(sizeof(int)*(s_len+1));
      cost[0]     = (int *) malloc(sizeof(int)*(s_len+1)*(t_len+1));
      for (i  = 1; i <= s_len; i++)
          cost[i] = cost[i-1] + t_len + 1;

      /* ---------------- initialization ----------------       */

      cost[0][0] = 0;
      for (i = 1; i <= s_len; i++)
          cost[i][0] = cost[i-1][0] + 1;
      for (j = 1; j <= t_len; j++)
          cost[0][j] = cost[0][j-1] + 1;

      /* ----- compute minimum cost for each i and j -----       */

      for (i = 0; i < s_len; i++)
          for (j = 0; j < t_len; j++)
              if (source[i] == target[j])
```

Download at http://www.pin5i.com/


```
            cost[i+1][j+1] = cost[i][j];
        else {          /* computes the min. cost          */
            insert_t = cost[i+1][j] + INSERT_COST;
            delete_s = cost[i][j+1] + DELETE_COST;
            exchange = cost[i][j] + EXCHANGE_COST;
            cost[i+1][j+1] = MIN(insert_t, delete_s, exchange);
        }

    /* ---------- get the locations of L.C.S. ----------      */

    for (i = s_len, j = t_len, no = 0; i != 0 && j != 0; )
        if (cost[i][j] == cost[i-1][j] + INSERT_COST)
            i--;
        else if (cost[i][j] == cost[i][j-1] + DELETE_COST)
            j--;
        else {
            s[no] = i - 1;
            t[no] = j - 1;
            no++, i--, j--;
        }

    reverse(s, no);
    reverse(t, no);
    *count = no;
    free(cost[0]);
    free(cost);
}
/* ----------------------------------------------------------  */
/* FUNCTION  edit_op :                                         */
/*   This routine is used to output editing operations.        */
/* ----------------------------------------------------------  */

#define   MAX(x, y)   ((x) >= (y) ? (x) : (y))

void edit_op(char source[], char target[], int s[], int t[], int n)
{
    int  length = MAX(strlen(source), strlen(target)) + 1;
    int  i, j, k;
    char *work;

    printf("\n\nEditing Operations are listed as Follows\n");
    work = (char *) malloc(sizeof(char)*length);
    for (i = j = k = 0; i < n; j=s[i]+1, k=t[i]+1, i++) {
```

Download at http://www.pin5i.com/


```
        if (s[i] - j > 0) {
            (void) strncpy(work, &source[j], s[i] - j);
            work[s[i]-j] = '\0';
            printf("\nDelete \"%s\" from source string.", work);
        }
        if (t[i] - k > 0) {
            (void) strncpy(work, &target[k], t[i] - k);
            work[t[i]-k] = '\0';
            printf("\nInsert \"%s\" to source string.", work);
        }
        printf("\nKeep \'%c\' from source string.", source[s[i]]);
    }
    if (strlen(source) - j > 0)
        printf("\nDelete \"%s\" from source string.", &source[j]);
    if (strlen(target) - k > 0)
        printf("\nInsert \"%s\" to source string.", &target[k]);

    printf("\n\nDONE! Target string generated.");
    free(work);
}

/* ----------------------------------------------------------------- */
/* FUNCTION reverse :                                                */
/*   This function reverse the given array in place.                 */
/* ----------------------------------------------------------------- */

#define  SWAP(a, b)  { t = a; a = b; b = t; }

void reverse(int x[], int n)
{
    int i, j, t;

    for (i = 0, j = n-1; i <= j; i++, j--)
        SWAP(x[i], x[j]);
}
```

问题 6.11 产生无连续重复部分的字符串（DISTSEQ.C）

请写一个程序，产生由 1,2,3 这 3 个数字符号所构成、长度为 n 的字符串，并且在字符串中对于任何一个部分字符串而言，都不会有相邻的、完全相同的部分字符串。

Download at http://www.pin5i.com/

【解答】

事实上并没有简单而且高效率的方法来做这件事，不过简单的回溯式技巧倒是可以完成题目所需。假设字符串长度为 5,，现在有 12321，最后一位可以换成下一个值 2 来试试，得到 12322，因为 2 与 2 是相邻而且是重复出现的，所以不合要求，把最后一位再换成下一个值，得到 12323，同样，因为 23 相邻而且重复出现，所以不合要求。到此为止，最后一位已经到 3，不能再增加了，回溯到前一位，回溯之后就等于只有四位而已，亦即 1232。此时，是因为第 4 位为 2，才从后面回溯来的，所以试下一个值，即 1233，也不行，再回溯，这一次就倒退了两个 3，回到第 2 位，是 12。把 2 加到下一位再试，是 13，可以。

有了 13 之后，因为还不足 5 位，可以尝试延长它，永远从 1 起，得到 131，可以。不过长度还不足 5 位，再延长，得到 1311，却不可以，因为 1 与 1 相邻而且重复，加到下一个值去，得到 1312，可以。所以再度延长它，得到 13121，可以。

13121 的下一个是什么？13122（不可以）、12123（可以）、1313（不可以）、1321（延长）、13211（不可以）、13212（可以）、13213（可以）、1322（不可以）、1323（可以，延长）、13231（可以）、13232（不可以，32 重复而且相邻），…。就这样，不难体会出回溯的过程。

如何检查一个字符串是否符合题目要求呢？要检查一个字符串中是不是有相邻而且重复的部分字符串并不是一件难事，因为一个部分字符串要相邻而且重复，那么它的长度最小就是 n/2，大过 n/2 的部分字符串就不可能有重复的可能性了。于是对于每一个字符串中的元素 x_i 而言，如果 $1 \leqslant k \leqslant n/2$，$x_{i+k}$ 就是与 x_i 相距为 k 的下一个符号，如果 $x_i x_{i+1} \cdots x_{i+k-1}$ 与 $x_{i+k} x_{i+k+1} \cdots x_{i+2k-1}$ 就是两个长度为 k 而且相邻的部分字符串，如果相等，就有了重复而且相邻的部分字符串了。但若不等，就移到 x_{i+1} 与 x_{i+k+1} 起长度为 k 的部分字符串继续处理，一直到所有情况都试过为止，如果都没有发现相邻而且相同的部分字符串时，就可以肯定原字符串中没有长度为 k、相邻而重复的部分字符串；但是要反复去查有没有长度为 $k+1, k+2, \cdots, n/2$ 的才行，一直到全部 k 都试过了，都找不出相邻而重复的部分字符串时，该字符串才是符合要求的。这样的做法一共要多少次比较呢？见图 15-11。

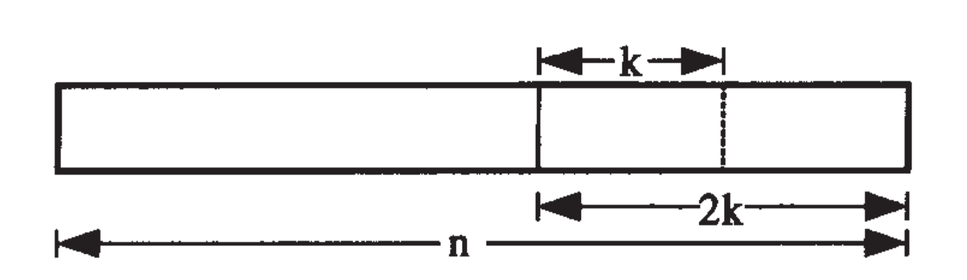


图 15-11

如果现在要处理长度是 k 的部分字符串，那么因为是要查相邻而重复的，所以是 2k 个符号一组，看前半与后半是否相同；在这 2k 个之前有 $n-2k$ 个元素，每一个元素起都可以有 2k 个元素一组，连同最后一组，一共有 $n-2k+1$ 组。因为每一组有 2k 个元素，同组中前后对应位置的元素要相互比较，就用了 k 次比较，总共有 $n-2k+1$ 组，所以比较总数是：

Download at http://www.pin5i.com/

$$\sum_{k=1}^{n/2}(n-2k+1)\cdot k=(n+1)\sum_{k=1}^{n/2}k-2\sum_{k=1}^{n/2}k^2$$
$$=(n+1)\frac{(n/2+1)n/2}{2}-2\cdot\frac{1}{6}n/2\cdot(n/2+1)(n+1)$$
$$=n(n+1)(n+2)\left(\frac{1}{8}-\frac{1}{12}\right)$$
$$=\frac{1}{24}n(n+1)(n+2)$$

因此比较次数与n^3成正比。上式中用到了下面的一个常见的式子：

$$\sum_{i=1}^{n}i^2=\frac{1}{6}n(n+1)(2n+1)$$

这个比较次数能否降低吗？当然是可以的，只要找出有哪些工作是重复做了而可以节省的就行了。如果采用回溯的技巧，在处理到$x_1\ x_2\cdots x_i\cdots x_k$而回溯到$x_i$时，会得到$x_1\ x_2\cdots x_i$，修正$x_i$的值而变成$x_i'$，修正后的字符串为$x_1\ x_2\cdots x_i'$。注意一点，在此之前在得到$x_1\ x_2\cdots x_k$的时候，早就已经查过了它不会有相邻而且重复的部分，自然$x_1\ x_2\cdots x_i$与$x_1\ x_2\cdots x_{i-1}$也不会有，当把x_i改成x_i'而得到$x_1\ x_2\cdots x_{i-1}\ x_i'$时，惟一可能会出问题的所在就是$x_i'$，因为它是新来的。所以检查以$x_i'$为终点，在$x_i'$左方长度为 1,2,3…,i/2 的字符串不会有相邻而且重复的部分字符串。用长度 k 来看，以x_i'为终点、长度为 k 的部分字符串是$x_{i-k+1}x_{i-k+2}\cdots x_{i-1}\ x_i'$，所以比较这两个部分字符串看它们是否相同，如果对于 $1\leqslant k\leqslant i/2$ 而言，这些部分字符串都不同，那么$x_1\ x_2\cdots x_i'$中就没有相邻而重复的部分字符串了。这又有多少次比较呢？最长的部分字符串长度是 i/2，对于 $1\leqslant k\leqslant i/2$ 而言，要比较以x_i'为终点的两个长度为 k 的相邻部分字符串。要有 k 次比较，所以总比较次数是：

$$\sum_{k=1}^{i/2}k=\frac{1}{8}i(i+2)$$

不过是与i^2成正比而已，所以此处程序就用这个方法。

程序的名称是 DISTSEQ.C，主控的部分是 seq_gen()函数，seq[ ]数组就是x_i字符串，为方便计算此处用整数。一开始时先把 1,2 放到数组中，而 no 则是 seq[ ]中最后一个元素的脚码。while 循环指出，当回溯超越数组顶点时就停止；如果还可以继续工作，首先就去查目前的 seq[ ]中的内容是否满足没有相邻而重复的部分字符串的条件，这是 check()函数的工作。如果字符串符合条件，再看看它是否已经装满了 n 个元素，如果是，就把它显示出来，并且修正最后一个元素的值而求得下一个组合方式；但若不满 n 个元素，就调用 extend()函数，尝试在最后面多加一个元素。如果 seq[ ]中的内容不符合条件，那就一定是最后一个元素有问题(见上面的说明)，因此通过修改最后的元素来产生下一个可能的组合。

change()函数非常简单，它把在最后面的 3（也就是最高值）都去掉，直到找出一个不是 3 的位置为止，把它加上 1 变成下一个值。因此如果在 seq[ ]中有 123213333，就会被 change()去掉尾部的 4 个 3，而变成 12321，再把最后一个加 1 变成 12322。至于是否符合条件，就不是 change()的事了，因为 seq_gen()总是会在使用 seq[ ]之前用 check()检查一次的。

Download at http://www.pin5i.com/

extend()也很简单，它永远会把下一个位置放一个 1，因为若 seq[ ]中为 1232，经处理后就会得到 12321。同样地，extend() 也不理会它的正确性。

最后是 check() 函数，它依照上述的讨论来检查 seq[ ]这个数组。length 是要检查的部分字符串的长度，它最长不超过字符串的一半。固定了 length 之后，就将 x[no] 起往左算有 length 个元素的部分字符串与它的相邻的、长度相同的部分字符串比较，一旦发现相同，就马上返回控制权。

【习题】

（1）为什么在 check()中 half 的初值是(no+1)/2 而不是 no/2？两者有何差异？请分析说明。

（2）请扩充这个程序，让它不限定在 3 个符号，而可以是任意多个符号；这个程序的架构还能使用吗？为什么？如果可以，要修改的地方何在？

（3）在 seq_gen()中，不给 seq[1]初值，而只给 seq[0]，可以吗？为什么？

【参考文献】

这个题目选自 Niklaus Wirth 的著作 Systematic Programming-An Introduction，第 142 页～第 149 页，读者可以斟酌参考。

1 N.Wirth. Systematic Programming-An Introduction, Prentice-Hall, 1973

找出字符串中相邻而且重复的部分是个不简单的问题，这方面参考文献很多，但却有相当难度，不容易在书中讨论，可以参看下面的参考文献：

2 A.Apostolico and F.P.Preparata. Optimal Off-Line Detection of Repetitions in a String, *Theoretical Computer Science*. Vol.22(1983), pp.297～315

3 M.Crochemore. An Optimal Algorithm for Computing the Repetitions in a Word, *Information Processing Letters*. Vol.12(1981), pp.244～250

4 M.G.Main and R.J.Lorentz. An O(nlogn)Algorithm for Finding All Repetitions in a String, *Journal of Algorithms*. Vol.5(1984), pp.422～432

这些文章中都提出为了要找出长度为 n 的字符串中最长的、相邻而且重复的部分字符串的方法，这些方法要用与 $n\log_2 n$ 成正比的比较次数。这些方法不但在比较次数上少，而且所做的事还比较多。在一篇长文中，俄国人 A.O.Slisenko 证明了与 n 成正比就可以做到这一件事，实在令人吃惊，不过笔者没时间阅读这篇长达 70 页的文章，如果您有兴趣，可以参看：

5 A.O.Slisenko. Detection of Periodicities and String-Matching in Read Time, *Journal of Soviet Mathematics*. Vol.22(1983), pp.1316～1387

【程序】

程序 15-11（DISTSEQ.C）

```
/* ------------------------------------------------------------------ */
/* FUNCTION  seq_gen :                                                */
/*   Given an input 'n', this function generates all 3               */
```

Download at http://www.pin5i.com/


```
/* letter sequence of length n with no repeating neighbor        */
/* substring.  This function is programed with backtrack         */
/* technique and the check() function is NOT the fastest.        */
/*                                                               */
/* Copyright Ching-Kuang Shene          Aug/12/1989              */
/* ---------------------------------------------------           */

#define  OK  1
#define  NO  0

/* ---------------------------------------------------           */
/*                  function prototypes                          */
/* ---------------------------------------------------           */

void change(int [], int *);
void extend(int [], int *);
int  check(int [], int);
void output(int [], int);

/* ---------------------------------------------------           */
/* FUNCTION  seq_gen :                                           */
/*   The given input is 'n' and seq[] array is the array         */
/* to store generated sequence.                                  */
/* ---------------------------------------------------           */

void  seq_gen(int seq[], int n)
{
     int  no;
     int  i;

     seq[0] = 1;                        /* two initial digits     */
     seq[1] = 2;
     no     = 1;                        /* last index in seq[]    */
     while (no >= 0)                    /* if still have chance....*/
         if (check(seq, no) == OK)      /* is this seq OK ?       */
             if (no == n-1) {           /* Full ?                 */
                 output(seq, n);        /* YES, show it           */
                 change(seq, &no);      /* backtrack...           */
             }
             else
                 extend(seq, &no);      /* OR, extend it          */
         else
```

Download at http://www.pin5i.com/


```
                change(seq, &no);        /* NOT OK, backtrack          */
        }
        /* ------------------------------------------------------------ */
        /* FUNCTION  change :                                           */
        /*   Given a sequence seq[] and the last index of it, no,       */
        /* this function finds the last digit such that has a           */
        /* chance that could be increased.                              */
        /* ------------------------------------------------------------ */

        void  change(int seq[], int *no)
        {
             for ( ; seq[*no] == 3; (*no)--)
                 ;
             seq[*no]++;
        }

        /* ------------------------------------------------------------ */
        /* FUNCTION  extend :                                           */
        /*   Put a '1' into the position next to the last.              */
        /* ------------------------------------------------------------ */

        void  extend(int seq[], int *no)
        {
             seq[++(*no)] = 1;
        }

        /* ------------------------------------------------------------ */
        /* FUNCTION  check :                                            */
        /*   This function checks to see if the sequence has a          */
        /* consecutive duplicated substring.                            */
        /* ------------------------------------------------------------ */

        int  check(int seq[], int no)
        {
             int  length;
             int  half = (no + 1)/2;
             int  i;

             for (length = 1; length <= half; length++) {
                 for (i = 0; i<length && seq[no-i]==seq[no-length-i]; i++)
                     ;
                 if (i >= length)
```

Download at http://www.pin5i.com/


```
            return NO;
        }
    return OK;
}

/* ----------------------------------------------------------------- */
/* FUNCTION  output :                                                */
/*   Function to display the generated sequence.                     */
/* ----------------------------------------------------------------- */
void  output(int seq[], int n)
{
    int  i;

    printf("\n");
    for (i = 0; i < n; i++)
        printf("%d", seq[i]);
}
```

Download at http://www.pin5i.com/

第16章 其他问题

问题7.1 Buffon丢针问题（BUFFON.C）

在地面上画一条距离为L的平行线，有一根长度为D的针。如果把针随意地往地面上一丢，请问压线的概率有多少？这就是有名的Buffon（人名）丢针（Needle）问题。请写一个程序，用仿真的技巧，估计出针压线的概率。

【解答】

这个程序其实很简单，不过这里要强调的是，如何使用C语言的随机数函数。在<stdlib.h>中有srand()与rand()两个函数，srand()函数接收一个unsigned的参数，作为一串拟似随机数（Pseudo-Random Number）的种子（Seed），以后每一次调用rand()时，都会返回一个在0～RAND_MAX（在<stdlib.h>中定义=之间的整数随机数。正因为如此，为了要产生真正的随机数（其实是均匀Uniform随机数），这种种子值也要足够随机，使人猜不到。这有一个好办法，可以用clock()函数做，它会返回当天中以某个单位计算（比如百分之一秒）的时间，类型是clock_t。这个值是时间，要把它转换成unsigned，然后交给srand()使用是很理想的。当然也可以有其他变换方法，不过此处就只用这一点。

因为rand()的函数在0与RAND_MAX之间，所以：

(double) rand()/RAND_MAX

其值就在0与1之间，当然把上式乘上某个正数（如c），于是就得到0～c之间的均匀随机数。

程序BUFFON.C中的变量pi，也就是π，它是用$\cos^{-1}(1)=\pi$的特性算出来的，x与theta（问题说明中的θ）就是针的位置与方向，dist相当于$x+L\sin\theta$。其他部分十分简单，请读者自己研究程序。

【习题】

（1）也可以把针的中点作为参考点，请问在这种情况下要如何改变程序？答案是否一样？

（2）理论上是产生的点越多越好，请列出一张表，当点数是10、50、100、500、1000、5000、10000时，所估计的值（在程序中把表印出来）是否点数越多就越准确呢？

（3）也可以用这个方法来算面积。单位圆在第一象限的部分的面积是$\pi/4$。请在(0,0)、(1,0)、(1,1)、(0,1)这4个点所围成的正方形中产生n个点，看有多少个点会落在单位圆之内，并且估计出单位圆的面积；这个值与$\pi/4$有很大差异，乘上4之后是否很靠近π？

（4）请用这个方法求f(x)=sin(x)函数在x=0～x=π之间的面积，答案是否准确？

（5）$\sqrt{5}$在2与3之间，因此可以产生n个在这之间的均匀随机数x，算出有多少个数满足$x^2<5$的关系，假设有m个，于是$\sqrt{5}$的值就大约是2+m/n。了解为什么这是对的？

Download at http://www.pin5i.com/

如何求 $\sqrt{n}$ 与 $n^{1/3}$？

（6）请写一个函数 xrand(a,b)，它接收两个 double 的值 a 与 b，a>b，并且返回一个在 a 与 b 之间的均匀随机数。

（7）丢一个铜板，正面与反面的概率各是 0.5，请用仿真的方法证实它。

（8）掷两个骰子出现 1～12 点的概率是 0、1/36、2/36、3/36、4/36、5/36、6/36、5/36、4/36、3/36、2/36、1/36。请写一个程序，用仿真方法证实这一个结果。

（9）在单位圆中任意画一条弦，它的长度大约是圆周长 1/3 的概率是多少？

【程序】

程序 16-1（BUFFON.C）

```
/* ------------------------------------------------------------ */
/* PROGRAM Buffon Needle Problem :                              */
/*   This is the simulation program for Buffon Needle           */
/* problem.  It generates pairs of (x,theta), where x is        */
/* is distance from a point to the lower parallel line          */
/* while theta is the angle between the lower parallel          */
/* line and the needle.  From this we can determine if the      */
/* needle crosses any parallel line.  Suppose we repeat         */
/* this procedure enough time, (# of crossing/# of points)       */
/* is an approximate of the true probability, namely 2/pi       */
/*                                                              */
/* Copyright Ching-Kuang Shene          July/10/1989            */
/* ------------------------------------------------------------ */


#include <stdio.h>
#include <stdlib.h>
#include <math.h>                /* for sin()                   */
#include <time.h>                /* for clock()                 */

void main(void)
{
     unsigned seed = (unsigned) clock();    /* the seed         */
     int i, count = 0;
     int n;
     double  pi = acos(-1.0);    /* the number pi = 3.414...    */
     double  x, theta, dist;     /* the needle's length         */
     double  length = 1.0;
     char    line[100];

     printf("\nBuffon Needle Program by Simulation");
     printf("\n=====================================");
     printf("\n\nNumber of Simulation Points --> ");
     gets(line);
     n = atoi(line);
```

Download at http://www.pin5i.com/


```
    srand(seed);
    for (i = 1; i <= n; i++) {
        x     = (double) rand() / RAND_MAX;
        theta = ((double) rand() / RAND_MAX) * 2.0 * pi;
        dist  = x + length * sin(theta);
        if (0. < dist && dist < length)
            count++;
    }
    printf("\n\nCrossing Probability --> %.2f",
                        1.0 - (double) count/n);
    printf(  "\nTrue Probability ------> %.2f", 2.0/pi);
}
```

问题 7.2　三色旗问题（FLAG.C）

假设有一个数组，它有 n 个元素，每一个不外乎是红、白、蓝 3 种颜色之一的代号，就用 R、W、B 代表。这些元素在数组中并没有依同样颜色的元素排在一起的方式来排列，请写一个程序把这个元素排成所有蓝色在前，接着是白色，最后是红色的排列方式，不过在写程序时要满足下面的条件：

（1）不能用额外的内存，换句话说，只能在数组之内用互换的方式完成。

（2）互换两个元素的动作要越少越好。

（3）对于每一个元素而言，测试它是红、是白，还是蓝的工作每种颜色最多只能做一次测试。

在这个条件之下，请写一个最快的程序。

【解答】

做法是把存储颜色元素的数组从头到尾查一次，对于每一个被查到的元素而言，按照它的颜色，把它归到合适的分组中，因为分组完成后与分组之前的元素数量一样多，所以想法是把数组分成 4 个部分，如图 16-1 所示。

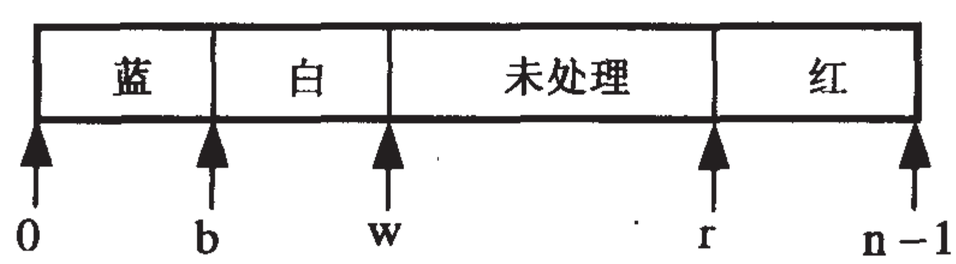


图　16-1

在 0～b－1之间放蓝色、b～w-1 放白色、r+1～n-1 放红色，而 w～r 之间的元素未曾处理。每一次都处理 w 所在的元素，这分为三种情形。

（1）如果 w 所在的元素是白色：那么把 w 加 1，把那个元素包含到白色的范围，整个数组的形态不变。

（2）如果 w 所在的元素是蓝色：于是把 b 与 w 所在的元素对调，因为 b 所在的元素

Download at http://www.pin5i.com/

原来是白色，现在收到 w 所在的蓝色元素，因而变成蓝色；同理 w 的新元素就是白色了。在这种情况下，b 与 w 各加 1，才能保持图 16-1 的形态。

（3）如果 w 所在的元素是红色，就把 w 与 r 所在的元素互换，但是 r 要减 1，涵盖住新换来的元素，但 w 却不能加 1，因为新换来的是什么颜色事先并不知道。

于是这个过程就一直反复，直到 w>r 为止，在这种情况下，就没有未处理的部分，因此分类完成。

开始时的 b、w、r 何在呢？因为在开始时根本就没有任何分组，而且记住 b 与 w 是指向对应分组的下一个位置，而 r 则是指向左边一个位置，于是 b 与 w 在开始时是 0，因为左边没有蓝色与白色分组，r 是 n-1，因为在 r 右边没有红色分组。

有了这个概念之后，编写程序就不难了。看看程序 FLAG.C，当发现 w 所在是个红色元素时，就让红色分组一直向左延伸，直到不能延伸为止，再把新找来的红色换过去，并且再度降低红色的指针 r。它的用意是，当 r 左边的都是红色元素时，增加这道程序，就可以降低把红色元素互换到红色分组中的机会。

在程序中，b、w、r 这 3 个指针用 blue、white、red 取代，存放颜色元素的数组叫做 color[ ]。

【分析】

可以发现，每一个元素最多比较两次，第一次比较是否是白色，第二次（如果有的话）比较是否是蓝色，而且当颜色不对时，只有一个互换。所以比较次数最多是 2n，互换次数则最多是 n，这里 n 是 color[ ]中的元素个数，因此用与 n 成比例的比较与互换的动作就可以完成工作。

程序能够到达这个境界吗？

【习题】

（1）请仔细了解程序中在 else 部分的 while 循环，确定真的能理解它的意义。

（2）在得到上面的解法之前，笔者曾经有过如图 16-2 所示的分组方式。

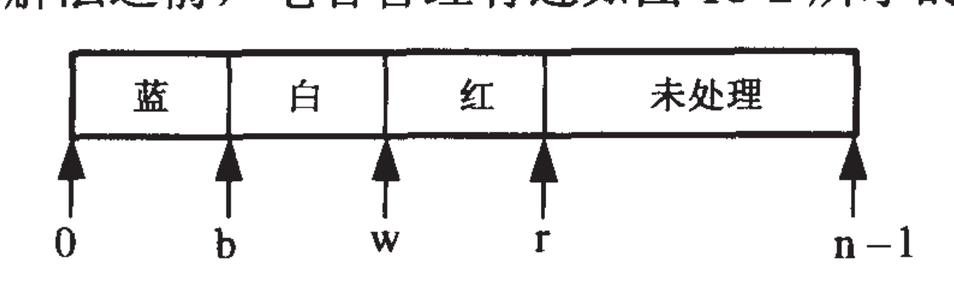


图 16-2

在 0～b-1 之间是蓝色，b～w-1 是白色，w～r-1 是红色，r～n-1 未处理，每次都处理 r 所指的元素，按照颜色将其放到对应的分组中。但笔者很快就放弃了这种方法，能看出这是为什么吗？究竟有哪一点不如上面的程序？

（3）请看下面的程序，它是不正确的，请找出一组使它做出不正确结果的数据。它与上一个习题的想法有何差异？

【程序】

程序 16-2（WRONGC）

```
/* ------------------------------------------------------------------ */
/* FUNCTION  incorrect_dutch_flag :                                   */
/*    This program contains a logical error.  Can you see             */
/* it? Find a configuration of color[] arrae such that                */
/* this program will report an incorrect answer please.               */
```

Download at http://www.pin5i.com/


```
/*                                                                    */
/* Copyright Ching-Kuang Shene                    Aug/30/1989        */
/* ------------------------------------------------------------      */

#include <string.h>          /* for strlen()                         */

#define  BLUE    'b'
#define  WHITE   'w'
#define  RED     'r'

#define  SWAP(x, y)  {char temp;                                      \
                      temp   = color[x];                              \
                      color[x] = color[y];                            \
                      color[y] = temp;                                \
                     }

void  dutch_flag(char color[])
{
     int white   = 0;
     int blue    = 0;
     int red     = 0;

     while (red < strlen(color))
         if (color[red] == RED)
             red++;
         else if (color[red] == WHITE) {
             SWAP(white, red);
             white++, red++;
         }
         else {
             SWAP(blue, red);
                 blue++;
             SWAP(white, red);
                 white++, red++;
         }
}
```

（4）请与快速排列法中 Nico Lomuto 的找分割点的办法作比较，它们有什么共同之处？又有什么不同的地方？

【参考文献】

这个题目是由 E.W.Dijkstra 最先提出的，笔者所能找到的最早文献是 Dijkstra 的一本名著。

E.W.Dijkstra. A *Discipline of Programming*, Prentice-Hall. 1976

参看这本书的第 111～116 页。这个问题通常叫做 Dutch National Flag 问题（荷兰国旗问题——Dijkstra 是荷兰人，这也是他原来的用语），不过也有一些作者称为三色旗（Three-Color Flag）问题。

Download at http://www.pin5i.com/

【程序】

程序 16-3（FLAG.C）

```
/* ------------------------------------------------------------ */
/* FUNCTION dutch national flag :                               */
/*   Given a char array containing 'b', 'r' and 'w', this       */
/* function rearranges these colors into bbb...ww...rrr         */
/* form by testing each element for each color exactly          */
/* once and minimizing number of swaps.                         */
/*                                                              */
/* Copyright Ching-Kuang Shene          July/16/1989            */
/* ------------------------------------------------------------ */

#include <string.h>          /* for strlen()                    */

#define  BLUE    'b'
#define  WHITE   'w'
#define  RED     'r'

#define  SWAP(x, y)  {char temp;                                 \
                      temp       = color[x];                     \
                      color[x]   = color[y];                     \
                      color[y]   = temp;                         \
                     }

void  dutch_flag(char color[])
{
     int white   = 0;
     int blue    = 0;
     int red = strlen(color) - 1;

     while (white <= red)            /* if there are more token */
         if (color[white] == WHITE)  /* is it a white?          */
             white++;                /* YES, just inc. white ptr*/
         else if (color[white] == BLUE) {   /* is it blue?      */
             SWAP(blue, white);      /* YES, swap the last w    */
             blue++, white++;        /* and the last b and inc. */
         }
         else {                      /* or is is a read,skip all*/
             while (white < red && color[red] == RED)
                 red--;              /* red token and           */
             SWAP(red, white);      /* swap with the last w    */
             red--;                  /* dec red ptr             */
         }
}
```

Download at http://www.pin5i.com/

问题 7.3 字符串列整数的转换（X_ATOI.C）

在平时课本中的 atoi()函数范例，通常都不能查出溢位（Overflow）的情况，或者是不理会这个情况而给出错误的结果，请修改这个 atoi()程序，把转换后的结果用第二个参数返回来。例如，x_atoi(s,*t)就用 t 返回结果，而函数则指出 OVERFLOW（大于最大整数值）、UNDERFLOW（小于最小整数值）、NO_ERROR（正常），为了方便起见，假设使用的机器是用 2 的补码存储整数。有没有办法查出机器是用 1 的补码还是 2 的补码？

【解答】

这个题目是本书中少数几个简单题目之一，关键就是在写 x_atoi()时如何发现溢位。注意，不能等到溢位了才做补救，这就来不及了，要在溢位发生之前把它查出来。

一般而言，在 atoi()中有一个变量 t，如果现在读到的数字符号是 d，于是 t 的新值就是：

t = 10*t + (d – '0');

于是就有两个地方可能发生溢位：一是 10*t；二是 10*t 没有溢位，但加上 d-'0'时溢位，但无论如何总是溢位了。

假设 int 的极大值（自<limits.h>中取得）是 INT_MAX，因此上式右边如果不会发生溢位，就会得到：

10 * t + (d – 0)<= INT_MAX

因为 t 与 d 在算这个式子时都是已知的，所以得到：

t<= (INT_MAX – (d – '0'))/10

换句话说，在更新 t 的值之前，t 要满足上式的关系，否则就会发生溢位。

在程序中，*result 是用来存放结果的，如果 x_atoi()发现了溢位，它立刻返回 OVERFLOW 作函数值，因而 result 的值是多少就无法预料了。在使用 x_atoi()函数的过程中，要先看看函数值，检查是否有错误发生，再进行其他工作。

没有写如何处理 UNDERFLOW，就留作习题吧。

【习题】

（1）程序中没有处理负数的 UNDERFLOW 情形，请参照解答中的做法，在程序中填补这一个漏洞。

（2）其实可以用这个观点把所在字符串到整数类型的转换改写，放到自己的链接库以备不时之需，试试看。

【参考文献】

这是一个在设计编译程序时人人都知道的问题，但却未必每一个写链接库或编译程序的人都会注意到这点，笔者不知道是谁最先提出一个可行的做法，但肯定好的程序语言的链接库中都一定会有这样的功能。有一篇短文可以提供参考：

J.R.Parker. On Converting Character Strings to Integers, *Information Processing Letters*. Vol.19(1984), pp.17～19

Download at http://www.pin5i.com/

J.R.Parker 在这篇文章中提到几个编译程序的奇怪现象，不过目前应该不会再出现。Parker 说，在某个 C 语言编译程序中，下面的宣告：

char A[66000];

居然会变成：

char A[464];

原因就是 66000 被 32768 去除的余数（0～32767 之间）正好是 464。笔者不曾碰到过这个问题，但的确从另一个 C 语言编译程序中，通过 atoi("32768")得到-32768。其他细节可参考 Parker 的文章，以及他的参考文献。

【程序】

程序 16-4（X_ATOI.C）

```
/* ------------------------------------------------------------------ */
/* FUNCTION  x_atoi :                                                 */
/*   An extended atoi() function with OVERFLOW detection.             */
/* You are expected to fill the missing UNDERFLOW part.               */
/* Given a string containing a sequence of digits, this               */
/* function converts this string to integer if no OVERFLOW            */
/* condition is detected, or returns OVERFLOW.  The result            */
/* is returned with a parameter.                                      */
/*                                                                    */
/* Copyright Ching-Kuang Shene          July/26/1989                  */
/* ------------------------------------------------------------------ */

#include <limits.h>          /* for INT_MAX and INT_MIN               */
#include <ctype.h>           /* for isdigit()                         */

#define  OVERFLOW    1
#define  UNDERFLOW   -1
#define  NO_ERROR    0

int  x_atoi(char s[], int *result)
{
     int  digit_value;       /* value of the scaned dig               */
     int  i;

     *result = 0;
     for (i = 0; isdigit(s[i]); i++)/* overflow test                 */
        if (*result <= (INT_MAX - (digit_value=s[i]-'0'))/10)
           *result = 10 * *result + digit_value;
        else
           return  OVERFLOW;        /* returns OVERFLOW               */
     return NO_ERROR;               /* normal ended                   */
}
```

Download at http://www.pin5i.com/

问题 7.4 整数类型列的极值（LIMITS.C）

请写一个程序，求出在<limits.h>中所定义的、各个有关整数类型的特征值。

【解答】

因为 4 个类型的处理方式大同小异，所以在下面的说明中就只用 char 类型作为范本，其他部分可参看程序 LIMITS.C。

在程序 LIMITS.C 中一开始，就定义了一个结构 param，其中包含了所有要求值的变量，并且都给出了合适的类型（也要注意，没有引入任何数据）。

第一件事，如何决定 char 类型有多少个数元。要用一个 unsigned char 的变量 uc，最先给它的值是~0，因此不管它有多少个数元，都是 1。对 unsigned 类型执行右移运算时（>>），ANSI C 规定这不是算术移位，换句话说，向右移而造成左边空出来的位置会用 0 填充。现在 uc 中的数元全都是 1，所以把 uc 一次右移一位（于是最左边就定出一个零），右移一位就少了一个 1，所以当右移的结果变成零时，移多少次就有多少个数元。例如，如果 unsigned char 是 4 个数元，因此 uc 为 1111，右移第一次为 0111，再移得到 0011，第三次是 0001，第四次变成 0000，所以一共移了 4 次，这就指出原来有 4 个 1，就是 1 的数元的数目。做这件事的是第一个 for 循环，结果存入 param.CHAR_BIT。

接下来看看如何判别设定的 char 类型是 unsigned char 还是 signed char。这件事并不难做，用一个等位（Union）即可；程序中叫做 u 的等位有 3 种类型的数据，unsigned char 叫做 uc，signed char 叫做 sc，而 char 则是 c，它们都占据同一个位置。当把一个-1 存入 signed char 的 sc 时，因为它带有符号，所以完全正确，但这个负值如果用 char 来看又如何呢？当然，如果 uc 的值小于零，就表示 char 会出现负值，因而就是 signed char 了。结果存储在 param.CHAR _SIGN 中。

对于 unsigned 的类型而言，它的数元全是 1 时（~0）就是最大值，如果它一共有 n 个数元，那就是 2^n-1。但若是 signed，最左边是符号，在正时是 0，负时是 1，因此把 unsigned 值右移一位空出 0，得到 0111…111，这就是 signed 的极大值（$2^{n-1}-1$）。不过 signed 的极小值却不能太过冒失，因为这与机器是用 2 的补码或是 1 的补码有关。

要决定是用 1 的补码还是 2 的补码也很简单，就拿-1 来说，在 1 的补码系统中，会先把正的 1，即 0000…0001 取补码，得到 1111…1110，这就是-1。但若在 2 的补码系统中，取了补码还要加 1，所以-1 的表示方式为 1111…1111，即全部数元都是 1。在程序中，往往将有符号的 char 变量 u.sc 中存入-1，将无符号的 char 变量 uc 中存入~0，使它的数元都是 1，因此，当把 u.sc 的值以无符号的方式取出时（值在 u.uc 中），如果与 uc 相等，那就是 2 的补码，否则就是 1 的补码。

了解补码系统之后，才有办法求出 signed char 的极小值，为什么呢？在 2 的补码系统中，n 个数元所能表示的是$-2^n \sim 2^{n-1}-1$，但是 1 的补码系统之下，却只有$-(2^{n-1}-1) \sim 2^{n-1}-1$而已。现在，signed char 的极大值已经求出，存放在 param.SCHAR_MAX 中，前面已经交代过，它的值是 $2^{n-1}-1$，如果是 2 的补码，极小值就是$-(2^{n-1}-1)-1=2^{n-1}$；但若是 1 的补码，

Download at http://www.pin5i.com/

结果就是$-(2^{n-1}-1)$，这个极小值则存放在 param.SCHAR_MIN 中。

注意上面的每一个算术运算，它们都是在值的范围之下进行的，根本不会发生溢位（Overflow 或 Underflow）的情况。

其他类型的做法大同小异，请读者自己研读程序。

【习题】

（1）下面的 SUBTLERR.C 程序是笔者在某本书中取下来的，它只是其中的一部分，用来决定 long 类型的极大与极小值。请问：第一，这个程序对机器的硬件作了什么假定？第二，在这种假设之下，是不是结果已经呼之欲出？第三，能否解释这个程序用什么技巧来达成原作者的“目的”（提示：把循环的中间运算结果显示出来）？第四，为什么我说这个程序有问题？也就是说，在不同规格的硬件之下都能正常运行吗？在做这个题目时，切记不可以用任何编译程序去编译、执行它，因为若用的机器与原作者的相同，就一定会得到“正确”的结果，应该用 ANSI C 的观点理解，突破这个许多人都不太小心的问题。

【程序】

程序 16-5（SUBTLERR.C）

```
/* ------------------------------------------------------------ */
/* PROGRAM  subtlerr :                                          */
/*   This program contains a subtle error.  Can you see         */
/* it without running this program?  Actually, you CAN NOT      */
/* run it to pinpoint the error and this is what the word       */
/* 'subtle' means.  Suppose you are running this program        */
/* on a machine which fits this program, you won't get any      */
/* error.  So, your job is that pinpoint the error spot by      */
/* examineing the source code only.                             */
/*                                                              */
/* Copyright Ching-Kuang Shene                  Aug/30/1989     */
/* ------------------------------------------------------------ */

void main(void)
{
     long  l;

     for (l = 0x8000000fL; l <= 1; l--)
     ;
     l++;
     printf("\nMinimum long value = %ld", l);

     for (l = 0x7fffffF0L; l > 0; l++)
     ;
     l--;
     printf("\nMaximum long value = %ld", l);
}
```

（2）虽然目前大多数的机器对整数的运算都采用二进制，但也许还有极少数的十进制或十六进制、八进制的机器还在运转中，对这些机器而言，结果还正确吗？要修改什么地方。有没有办法找出机器是用什么进位系统呢？

Download at http://www.pin5i.com/

【程序】

程序 16-6（LIMITS.C）（部分）

```
/* ------------------------------------------------------------ */
/* PROGRAM  integer type characteristics :                      */
/*   This program determines those characteristics, such        */
/* as number of bits, max and min values of integer type.       */
/* There is an include file <limits.h> containing all of        */
/* these data, but a self-contained program that can finds      */
/* all of them should be very interesting.                      */
/*                                                              */
/* Copyright Ching-Kuang Shene          July/26/1989            */
/* ------------------------------------------------------------ */

#include <stdio.h>

#define  SIGNED      0
#define  UNSIGNED    1
/* ------------ structure for int type characteristics ------  */

struct limits {
     int             CHAR_BIT;       /* no. of bits in char      */
     int             CHAR_SIGN;      /* char type signed or not */
     signed char     SCHAR_MAX;      /* max value of signed char*/
     signed char     SCHAR_MIN;      /* min value of signed char*/
     int             SHRT_BIT;       /* bit # of short type      */
     short           SHRT_MAX;       /* max of signed short      */
     short           SHRT_MIN;       /* min of signed short      */
     int             INT_BIT;        /* bit # of int type        */
     int             INT_MAX;        /* max of signed int        */
     int             INT_MIN;        /* min of signed int        */
     int             LONG_BIT;       /* bit # in long type       */
     long            LONG_MAX;       /* max of signed long       */
     long            LONG_MIN;       /* min of signed long       */
     unsigned    char  UCHAR_MAX;    /* max of unsigned char     */
     unsigned    short USHRT_MAX;    /* max of unsigned short    */
     unsigned    int   UINT_MAX;     /* max of unsigned int      */
     unsigned    long  ULONG_MAX;    /* max of unsigned long     */
}  param;

/* ------------------------------------------------------------ */

/* FUNCTION  char_test :                                        */
/*   This function determines various characteristics of        */
/* char type, signed or unsigned.                               */
/* ------------------------------------------------------------ */
void  char_test(void)
{
```

Download at http://www.pin5i.com/


```
    unsigned char  uc;
    union {
        unsigned char   uc;
        signed char     sc;
        char            c;
    } u;
    int  i;
    char *sign[] = { "signed", "unsigned"};

    for (i = 0, uc = ~0; uc != 0; uc >>= 1, i++)
        ;                       /* determine char length        */
    param.CHAR_BIT  = i;    /* and signed or unsigned           */
    u.sc = -1;
    param.CHAR_SIGN = (u.c < 0) ? SIGNED : UNSIGNED;

    param.UCHAR_MAX = ~0;   /* the max of unsigned is ~0        */
    param.SCHAR_MAX = param.UCHAR_MAX >> 1;    /* signed        */
    u.sc = -1;              /* one's or 2's complement?         */
    uc   = ~0;
    if (u.uc == uc)         /* 2's complement                   */
        param.SCHAR_MIN = -param.SCHAR_MAX - 1;
    else                    /* 1's complement                   */
        param.SCHAR_MIN = -param.SCHAR_MAX;
    printf("\nchar Characteristics : \n");
    printf("\nCHAR_BIT (number of bits)    %d", param.CHAR_BIT);
    printf("   (%s)", sign[param.CHAR_SIGN]);
    printf("\nUCHAR_MAX(unsigned char max) %uU", param.UCHAR_MAX);
    printf("\nSCHAR_MAX (signed char max)  %d", param.SCHAR_MAX);
    printf("\nSCHAR_MIN (signed char min)  %d", param.SCHAR_MIN);
}
```

问题 7.5　无限位数算术（ARITH.C）

请写一个使用十进制运算，并且可以计算任意位数精确度的正整数加法与乘法的程序。

【解答】

有一个最笨拙的办法，即用一个 char 数组存储一个很大的十进制数，但是在每一个元素中并不是存储“0”、“1”、…这些数字符号，而是用“\0”、“\1”等；不过这就出现了一个小问题，如何表示字符串的终了呢？做法是使用字符串长度，而不是“\0”这个符号。因此，一个“数”就是一个结构，它包含了一个长度值（length）与一个指向存放了那个数的字符串所在的指针。例如，120 这个数有三位，它的表示方式如图 16-3 所示。

正因为有个长度栏，才不会漏掉 120 中的 0。

在 ARITH.C 中有一个叫做 make_number()的函数，它接收一个 long 类型的正整数，把它的值转化成上面的形式，得到一个无限位数的“数”的第一步。首先，它把 n 这个参数

Download at http://www.pin5i.com/

的绝对值用 sprintf()函数写到一个字符串中，而 len 则表示一共存入了多少个符号。但文字形式并不是需要的，所以下一个 for 循环就把文字形式转换成十进制的值。接下来，配置一个长度为 len+1 的字符串（注意，字符串后面要用“\0”结束），把刚得到的结果抄进去就行了。比如，如果 n 是 120，就会得到图 16-3 所示的样子。注意，不但是那个字符串，甚至于那个数，都是用 malloc()取得内存的。

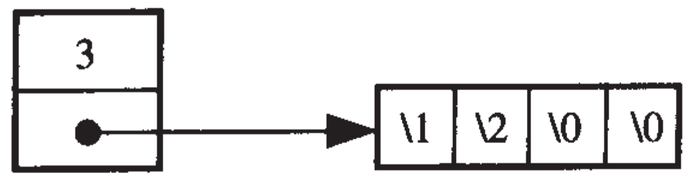


图 16-3

接下来看如何进行加法。使用 add()函数。如果要加两个数，首先要把个位数对齐，得到的结果的长度会比较长的操作数多 1，因为可能会有进位。如果两个操作数各有 m 位与 n 位，结果就有 MAX(m,n)+1 位，因此就用 malloc()先取得内存，并且在最后先放入一个字符串终结的符号。注意，个位数是在最右边，所以从右向左加，假设现在做到 u[i]与 v[i]这两位，它们的值都在 0～9 之间，而 carry 代表从右边一位（u[i+1]与 v[j+1]）相加得来的进位，所以把它相加之后就是目前这一位的结果。例如，若目前是 9 与 6 相加，没有自右而来的进位，因此就是 9+6+0=15，当然 5 才是这一位的结果，而 1 就会进到左边那一位去了。在程序中 BASE 是进位系统，定为 10；因此将加出来的结果（在 temp 中）用 BASE 去除就是进位，而余数正是这一位的值。第一个 for 就是做这件事的。但是，除非两个数长度一样，否则一定会有一个数的位数先被用完，这正是 for 循环中用 i>=0 && j>=0 的意义；如果这个条件不满足了，就把有余位的字符串的地址存入 p，余下来的最后一位放置到 t 中，接着再重复上面的工作，不过此时就是一个数与一个进位相加了。到最后，每一位都处理完了，把进位放到第 0 位中。接下来，把结果字符串中前端的零去掉，结果前面是有可能有零的，因为相加时最左边一位没有进位，最后返回结果，这个结果中就有指向两数和所在的字符串的指针。

相乘当然是复杂些，但并没有复杂到难以掌握的地步。做法是，把乘数的每一位从右向左地与被乘数每一位相乘，再加到结果中。如果被乘数有 n 位，乘数有 m 位，那么结果就会有 m+n 位，当然在左边可能会有零（比如 11 乘 11 是 121，这就不足 4 位了，因此乘出来就会是 0121）。

multiply()函数就是做这件事的，它首先通过 malloc()取得 m+n 个位置的字符串内存，并且把最后面的 n 位清为零，这是为乘数的个位数与被乘数相乘时给定初值用的，不可不做。于是对于乘数中某一位 v[j]而言，它就与被乘数的每一位，u[n-1]、u[n-2]、…、u[1]、u[0]相乘，乘出的值与结果中的对应位置的值相加，再加上进位，接着再像加法一样存入结果。两个 for 循环的脚码在开始时，可能略有了解上的困难。举例说明如下，求 123 乘以 45，结果有 3+2=5 位，一开始时右边三位定成零，因此有 xz000，这里 x 表示值未知。于是，先把 5 与 123 的每一位相乘，加到 xx000 中，因此结果是 x0615，再重复同样的工作，用 4 与 123 相乘，再加到结果中，但相加时要左移一位，得出的结果是 0492，与 x061 相加得 0553，因此最后结果为 05535。程序的最后，还是有一段去掉在前方的零的操作，最后返回指向结果的指针。

Download at http://www.pin5i.com/

display_number()是用来显示结果的。

在主程序中，要求输入一个值（大一点，比如30或100），令其为n，求出1!、2!、…、n!，并且显示出来，最后把1!+2!+…+n!也算出并显示出来，100!是多少？做一次就晓得了，一共有158位。

程序观念本身并不困难，但有一个可能让读者难以掌握的地方，就是一些指针的运用，如果不熟悉这一些，最好先看看C语言的参考书。

【习题】

（1）这个程序有太多free的执行，如果仔细看看主程序与各个函数，就会发现有很多malloc()与free()的执行，这两个执行是很费时间的。有没有办法克服它们？简单的做法是，不要做free()的执行，而把所有没有用的内存串成一个串行，等到malloc()时发现没有内存可用了，再全部free()。另外，不妨考虑一次取得一块较大的位置，然后就从这块区域中自己去切割，因而malloc()与free()的执行也可以降低，还有没有其他好的办法呢？

（2）请扩充这个程序，加上减法与除法，而且让程序可以处理负值。

（3）有一个加快程序速度的办法就是不用十进位，而改成百进位，或更大的进位。在百进位中，值为00～99，一个char中还是容纳得下，但计算的工作却少了一半。请修改程序用百进位来运算。

（4）本书前面提过只用与$\log_2 n$成正比的乘法就可以算出n!，请用这里的乘法函数编写这个程序，并且与这里的主程序作比较，比较哪个快，快多少？这是一个非常值得做的题目，因为只有在位数多了之后，$\log_2 n$的方法才能发挥作用。

提示： 2^n改成10^n，或许需要一个移位的函数。

（5）请用这个程序，再配合$\log_2 n$个运算求Fibonacci数的技巧求出很大的Fibonacci数。

【参考文献】

这里的两个整数相乘一共有m·n个乘法，事实上是一个最简单的方法，在理论计算机与算法设计范畴中，还有一些非常漂亮、快速的方法，其中最有名的就是Schonhage-Strassen方法，只需要与$n\log_2 n\log_2\log_2 n$成正比的数元运算次数就可以求出两个各有n个数元的乘积，当然比上述方法快得多。然而，在几乎所有的计算机代数的系统中都不会采用这些复杂的方法，反而这里介绍的简单方法的各种修改更常见，比如习题（3）。有兴趣的读者不妨参看下面的两份文献：

1 A.Aho. J.E.Hopcroft and J.D.Ullman. *The Design and Analysis of Computer Algorithms*, Addison-Wesley. 1976, pp.270～286

2 D.E.Knuth. *The Art of Computer Programming*, Vol.2/Seminumerical Algorithms, 2nd.ed., Addison-Wesley. 1981, pp.250～265

【程序】

程序16-7（ARITH.C）

```
/* ------------------------------------------------------------------ */
/* FUNCTION  infinite precision addition & multiplication             */
```

Download at http://www.pin5i.com/


```
/*   Given two infinite precision numbers, functions in          */
/* this program can add and multiply them in infinite            */
/* precision.  This package contains four basic functions,       */
/* namely, make_number(), add(), multiply() and                  */
/* display_number().                                             */
/*                                                               */
/* Copyright Ching-Kuang Shene          Aug/15/1989              */
/* ----------------------------------------------------------    */

#include <stdio.h>                /* for display_number()        */
#include <stdlib.h>               /* for malloc() and free()     */
#include <string.h>               /* for memmove()               */

#define  BASE        10           /* radix used in this system   */

#define  MAX(x, y)    ((x) >= (y) ? (x) : (y))
#define  MIN(x, y)    ((x) <= (y) ? (x) : (y))

typedef struct number  NUMBER;    /* type of a number            */

struct  number  {                 /* data struct. of NUMBER      */
     int length;                  /* it consists of a length     */
     char    *data;               /* and a ptr to data           */
};

/* ----------------------------------------------------------    */
/* FUNCTION  make_number :                                       */
/*   This function accepts a positive long integer and           */
/* converts it to the infinite precision form.                   */
/* ----------------------------------------------------------    */

NUMBER  *make_number(long n)
{
     char    buffer[16];          /* assume long has 16 digits   */
     int len, i;
     NUMBER *u;

     len = sprintf(buffer, "%d", (n >= 0) ? n : -n);
     for (i = 0; buffer[i] != '\0'; i++)     /* convert to bin   */
          buffer[i] = buffer[i] - '0';

     u   = (NUMBER *) malloc(sizeof(NUMBER));
     u->length = len;
     u->data   = (char *) malloc(sizeof(char)*(len+1));
     memmove(u->data, buffer, len+1);
     return u;
}
/* ----------------------------------------------------------    */
/* FUNCTION  add :                                               */
```

Download at http://www.pin5i.com/


```
/*   Given two numbers, this function computes their sum        */
/* and returns a new pointer to the sum.                        */
/* ------------------------------------------------------------ */

NUMBER  *add(NUMBER *opn1, NUMBER *opn2)
{
     int m   = opn1->length;
     char *u = opn1->data;
     int n = opn2->length;
     char *v = opn2->data;
     char *w, *p;
     int mn = MAX(m,n)+1;
     int carry = 0, temp;
     int i, j, k, t;
     NUMBER *result;

     w = (char *) malloc(sizeof(char)*(mn+1));
     w[mn] = '\0';

     for (i=m-1, j=n-1, k=mn-1; i>=0 && j>=0; i--,j--,k--) {
         temp    = u[i] + v[j] + carry;
         w[k]    = (char) (temp % BASE);
         carry   = (char) (temp / BASE);
     }

     if (i >= 0)              /* NOTE that last part gets        */
         p = u, t = i;        /* the second half of the          */
     else                     /* result.  Now computes the       */
         p = v, t = j;        /* the first half.                 */

     for ( ; t >= 0; t--, k--) {
         temp    = p[t] + carry;
         w[k]    = (char) (temp % BASE);
         carry   = (char) (temp / BASE);
     }
     w[0] = carry;

     result = (NUMBER *) malloc(sizeof(NUMBER));
     if (w[0] == '\0') {          /* adjust array such that      */
         mn--;                    /* the first digit != 0        */
         result->data = (char *) malloc(sizeof(char)*(mn+1));
         memmove(result->data, &w[1], mn+1);
         free(w);
     }
     else
         result->data = w;
     result->length = mn;
     return result;
}
```

Download at http://www.pin5i.com/


```
/* ------------------------------------------------------------ */
/* FUNCTION multiply:                                           */
/*   Given two infinite precision mumber,this function          */
/* computes their product.                                      */
/* ------------------------------------------------------------ */
NUMBER *multiply(NUMBER *opn1, NUMBER *opn2)
{
     int n   = opn1->length;
     char *u = opn1->data;
     int m   = opn2->length;
     char *v = opn2->sata;
     char *w;
     int mn = m + n;
     int carry, temp;
     int I, j;
     NUMBER *result;

     w = (char* ) malloc(sizeof(char)(mn+1));
     Memset(&W[m], 0, n+1;

     for (j = m - 1; j>= 0; j--)     /* for each sigit...        */
         if (v[j] != '\0') {         /* computes product         */
             for (carry = 0, i = n - 1; i >= 0; i--){
                 temp    = u[i]*v[j] + w[i+j+1] + carry;
                 w[i+j+1]    = (char) (temp % BASE);
                 carry = (char) (temp / BASE);
             }
             w[j] = (char) carry;
         }
         else
             w{j} = '\0';
     for (i = 0; w[i] == '\0'; i++)      /* remove leading 0's   */
         ;
     result = (NUMBER *) malloc(sizeof(NUMBER));
     result->length =mn - i;
     result->data = (char*) malloc(sizelf(char)*(mn-i+1));
     memmove(result->data, &w[i], mn-i+1);
     free(w);
     return result;
}
/* ------------------------------------------------------------ */
/* FUNCTION display number :                                    */
/*   Display an infinite precision number. NOTE that a          */
/* zero is represented with a null string.                      */
/* ------------------------------------------------------------ */
void display_number(NUMBER *no)
{
     int i;
```

Download at http://www.pin5i.com/

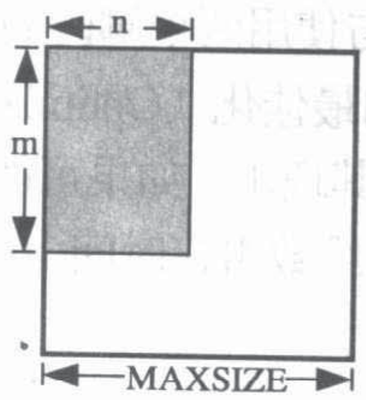

```
        if (no->data == 0)
            printf("0");
        else
            for (i = 0;i < no->length; i++)
                printf("%c", (no->data)[i] + '0');
    }
```

问题 7.6 线性表示的矩阵相乘（MATMUL.C）

在 C 语言中虽然号称功能比 Pascal、Fortran 强，但是在某些方面其实却并不见得有多少改进，函数参数中的高维数组就是一个例子，如果要保持数组的结构，就只能在第一维留下空白而变成不完整数组，其他维度的容量还是要写出来。这一点只比 Pascal 好一点，与 Fortran、PL/1 或 Ada 相比还差得远。在早期 Fortran 程序员为了突破 Fortran 的条件而发展出一组很实用的技巧，本题与下一题的目的就在于此。请写 3 个函数，第一个函数把二维数组转换成一维，第二个函数把一维转换成二维，第三个函数求两个代表二维数组的一维数组运行矩阵相乘。

【解答】

注意，TwoD[ ][ ]中有多少列并不知道，只知道它最多有 MAXSIZE 行，但是在这一块区域中都只用了 m 列 n 行，m・n 个元素而已。



所以要变成一维的形式并不难，把 TwoD[ ][ ]中存了数据的 m 列 n 行从左到右、从上到下地一个接一个元素查过去，依次放到 OneD[ ]中就行了。

反过来，如果 OneD[ ]中前 m・n 个元素是一个 m 列 n 行的矩阵，而且以列的方式排列，但因为 n 小于 MAXSIZE，所以要从右下角开始处理，而且还要从右到左、从下到上才行（为什么）。

如果 a[ ]与 b[ ]分别是 L 列 M 行、M 列 N 行的矩阵，而且是用一维、列为主的方式存放的（也就是 twoD_to_oneD()的结果），那么如何进行矩阵相乘呢？这也不难，如果要求 a[ ]的第 i 列与 b[ ]第 j 行元素两两相乘的结果时，先要知道 a[ ]的第 i 列与 b[ ]的第 j 行何在，就可以相乘。因为是以列为主的顺序存放的，所以 a[ ]的第一列是 a[0]～a[N-1]（有 N 行）、第二列是 a[N]～a[2・N-1]、…、第 i 列就是 a[(i-1)・N]到 a[i・N]，如果用一个变量 p，使它从 0 起，每次增加 N，于是 p 就正好是 a[ ]中每一列开头元素的脚码。再看 b[ ]的第 j 行，这稍微麻烦些，因为每一行上的元素并不连续，第一行是 b[0]、b[N]、b[2・N]、…，第二行为 b[1]、b[N+1]、b[2・N+1]、…，因此第 j 行就是 b[j]、b[N+j]、b[2・N+j]、…。所以，若在求 a[ ]的第 i 列与 b[ ]的第 j 行相乘之前，先找出第 j 列开头位置（在 pp 中）以及第 i

Download at http://www.pin5i.com/

行的开头位置（在 qq 中），那么相乘、相加的程序就是：

```
for (sum = k = 0, pp = p, qq = j; k < M; pp++, qq += N, k++)
     sum += a[pp]*b[qq];
```

注意 pp 与 qq 的增加情形，正好是 p、p+1、p+2、…与 qq、qq+N、qq+2・N、…。程序就是用这个观点写成的，对照一下就不难理解了。

【习题】

（1）为什么 twoD_to_oneD()要从左到右、从上到下，但 oneD_to_twoD()中却要从右到左、从下到上呢？能找出不这样做就可能失败的例子吗？

（2）请为程序加上如下两个子程序：

```
void add(int a[], int b[], int c[], int m, int n);
void transp(int a[], int b[], int m, int n);
```

前者把 a[ ]与 b[ ]的和求出存入 c[ ]中；后者则是把 a[ ]的转置矩阵求出存入 b[ ]，这里 a[ ]与 b[ ]有可能在同一个位置。另外，所有矩阵都是用一维的方式存放。

（3）如果知道如何求矩阵的反矩阵（Inverse），请试写一个函数：

```
int inverse(double a[], double b[], int n);
```

把 a[ ]矩阵的反矩阵求出存入 b[ ]，并且用函数值报告运行是否有问题。

（4）请比较这里的一维数组做法与使用两个脚码的方式，能够看出为什么一维的写法“一般而言”会比较快吗？为什么在很好的最佳化（Optimization）能力的编译程序中，这种做法所得到的效益“可能”并不很大，试说明其理由。如果可能的话，请用不同的编译程序、不同的最佳化效果比较一下这个做法与直接用二维数组在时间上的差异。

【程序】

程序 16-8（MATMUL.C）

```
/* ------------------------------------------------------------------ */
/* PROGRAM  matmul :                                                  */
/*   An old trick for FORTRAN programmers.  For those C               */
/* programmers, they have to remember the maximum size of             */
/* the second dimension of a two dimension array and pass             */
/* this CONSTANT value as part of the array declaration.              */
/* This is a very restrictive feature next to Pascal.  But            */
/* we can recover from this stupid language restriction               */
/* somewaht by old tricks of FORTRAN programmers.                     */
/*                                                                    */
/* Copyright Ching-Kuang Shene          Aug/02/1989                   */
/* ------------------------------------------------------------------ */

#define  MAXSIZE  20

void twoD_to_oneD(int [][MAXSIZE], int, int, int []);
```

Download at http://www.pin5i.com/


```
void  oneD_to_twoD(int [], int [][MAXSIZE], int, int);
void  matmul(int [], int [], int [], int, int, int);

/* ----------------------------------------------------------------- */
/* FUNCTION  twoD_to_oneD :                                          */
/*   This function accepts a two dimensional array in C              */
/* form and transforms it to row major linear form.                  */
/* ----------------------------------------------------------------- */

void  twoD_to_oneD(int TwoD[][MAXSIZE], int m, int n, int OneD[])
{
     int  index, i, j;

     index = 0;
     for (i = 0; i < m; i++)
          for (j = 0; j < n; j++)
               OneD[index++] = TwoD[i][j];
}

/* ----------------------------------------------------------------- */
/* FUNCTION  oneD_to_twoD :                                          */
/*   This function tranforms the row major linear form               */
/* back to C's two dimension array.                                  */
/* ----------------------------------------------------------------- */

void  oneD_to_twoD(int OneD[], int TwoD[][MAXSIZE], int m, int n)
{
     int  index, i, j;

     index = m*n - 1;
     for (i = m - 1; i >= 0; i--)
          for (j = n - 1; j >= 0; j--)
               TwoD[i][j] = OneD[index--];
}
/* ----------------------------------------------------------------- */
/* FUNCTION  matmul :                                                */
/*   Row major linear form matrix multiplication routine.            */
/* ----------------------------------------------------------------- */
void  matmul(int a[], int b[], int c[], int L, int M, int N)
{
     int  index;
     int  i, j, k;
     int  p, pp, qq;
     int  sum;

     index = 0;
     for (i = p = 0; i < L; i++, p += M)
          for (j = 0; j < N; j++) {
```

Download at http://www.pin5i.com/


```
                    for (sum =k=0, pp=p, qq=j; k < M; pp++, qq+=N, k++)
                        sum += a[pp] * b[qq];
                    c[index++] = sum;
            }
        }
```

问题 7.7　对称表示的矩阵相乘（MATMUL_S.C）

问题 7.6 的矩阵是一般形式的，但若是个对称矩阵，转换成线性形式就可以节省差不多一半的内存位置，这就不止是方便的问题了。对一个 n 列 n 行的对称矩阵（它一定是方阵）而言，如果用二维数组方式存储，就需要 n·n 个元素，但是因为对称的缘故，存储上半（或下半）个三角形的部分就足够了，所以就只需要 n(n+1)/2 个位置而已。参照问题 7.6，编写 StwoD_to_oneD()、oneD_to _StwoD()以及 Smatmul()3 个函数，但对象都是对称矩阵。

【解答】

StwoD_to_oneD()与 oneD_to_StwoD()本身并不难，因为基本想法与问题 7.6 一样，所差的是这里的函数只要处理一半左右的元素就够了。而对于后者，转换回二维数组时，只从一维数组中得到上半个三角形部分，而下半部分则是同时抄录过去的。

Smatmul()是进行两个对称矩阵相乘用的，要注意到一点，两个对称矩阵相乘的结果不一定是个对称矩阵，所以这里的结果是一个一维数组，要用问题 7.6 的 oneD_to_twoD()把它展开回二维数组才会得到完整的结果，如图 16-4 所示。

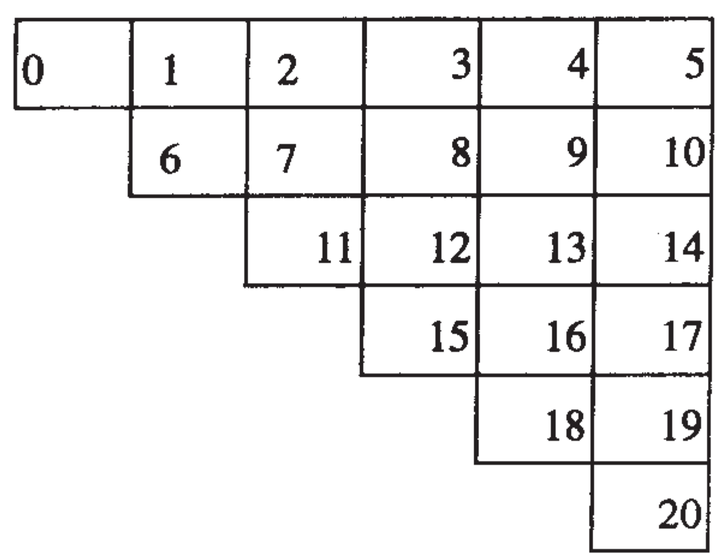


图　16-4

图 16-4 所示是一个 6×6 对称矩阵排成一维时各元素的脚码，因为是对称的，所以当要找第 i 列的数据时，开始的 i-1 个元素并不存在。但是对称的性质说明，它在第 i 行上，所以每一列的数据都是以 L 形出现。同理，对于第 j 行而言，因为对称，它的值与第 j 列的相同，自然也就是一个 L 形。例如，上面的第三列（脚码为 2），就是 2、7（第三列中少了的元素，不过在第三行上）、11、12、13、14（原第三列的元素）；第四行（脚码为 3）则是 3、8、12、15（原第四行的元素），16、17（第四行对角线以下的部分），所以一般而言，要用到第 i 列或第 j 行上的元素时，走一个 L 形是避免不了的。

如果注意各脚码的顺序，会发现第 i 行的开始脚码正是 i，在每一行上从上到下，各个

Download at http://www.pin5i.com/

脚码的递增值是 n-1、n-2、n-3、…、3、2、1，为什么不都是 n-1 呢？原因就在于从第二列起每一列左方都依次少了 1、2、3、…、n-1 个元素，因此第 i 行上元素的脚码就是 i、i+(n-1)、i+(n-1)+(n-1)、…，到了第 i 列上之后，就改成以 1 递增。程序中的 Smatmul() 就是这样写成的，gap 就是递增值，p 与 q 是一维数组中的脚码。当求 a[ ]的第 i 列与 b[ ]的第 j 行的元素两两相乘后的和时，因为第 i 列的元素与第 j 行的完全一样，所以就等于是求第 i 与第 j 行上的对应元素两两相乘后的和，所以 p 与 q 都从第 i 与第 j 行的第一个元素的脚码起，用 GAP()集体指令求递增值，而求出 a[p]与 b[q]的积加到 sum 中。

【习题】

（1）请写一个矩阵相乘的函数，求对称矩阵与非对称矩阵的相乘。注意，这分为两种情形，也就是说，在 $\boldsymbol{A}\cdot\boldsymbol{B}$ 中，$\boldsymbol{A}$ 对称而 $\boldsymbol{B}$ 不对称，$\boldsymbol{B}$ 对称而 $\boldsymbol{A}$ 不对称，需要写成两个函数吗？为什么？

（2）请写一个函数，接收一个矩阵（不对称）$\boldsymbol{X}$，求出 $\boldsymbol{X}'\cdot\boldsymbol{X}$，这里 $\boldsymbol{X}'$ 表示 $\boldsymbol{X}$ 的转置矩阵。$\boldsymbol{X}'\boldsymbol{X}$ 一定是对称矩阵。需要用到转置矩阵的函数吗？为什么？

【程序】

程序 16-9（MATMUL_S.C）

```
/* ------------------------------------------------------------------ */
/* PROGRAM  matmul :                                                  */
/*   Another trick of old FORTRAN programmer to overcome              */
/* the storage, efficiency and syntax program for                     */
/* symmetric matrices.  A symmetric matrix can be                     */
/* transformed to an one dimensional linear array and                 */
/* manipulated by ONE index only.                                     */
/*                                                                    */
/* Copyright Ching-Kuang Shene          Aug/03/1989                   */
/* ------------------------------------------------------------------ */

#define  MAXSIZE  20

void  StwoD_to_oneD(int [][MAXSIZE], int, int []);
void  oneD_to_StwoD(int [], int [][MAXSIZE], int);
void  oneD_to_twoD(int [], int [][MAXSIZE], int, int);
void  Smatmul(int [], int [], int [], int);


/* ------------------------------------------------------------------ */
/* FUNCTION  twoD_to_oneD :                                           */
/*   This function accepts a two dimensional symmetric                */
/* array in C form and transforms it to row major linear              */
/* form.                                                              */
/* ------------------------------------------------------------------ */

void  StwoD_to_oneD(int TwoD[][MAXSIZE], int n, int OneD[])
{
     int  index, i, j;
```

Download at http://www.pin5i.com/


```
        index = 0;
        for (i = 0; i < n; i++)
            for (j = i; j < n; j++)
                OneD[index++] = TwoD[i][j];
    }

    /* ----------------------------------------------------------------- */
    /* FUNCTION  oneD_to_StwoD :                                         */
    /*   This function tranforms the row major linear form               */
    /* back to C's two dimension symmetric array.                        */
    /* ----------------------------------------------------------------- */

    void  oneD_to_StwoD(int OneD[], int TwoD[][MAXSIZE], int n)
    {
        int  index, i, j;

        index = n*(n+1)/2 - 1;
        for (i = n - 1; i >= 0; i--)
            for (j = n - 1; j >= i; j--)
                TwoD[i][j] = TwoD[j][i] = OneD[index--];
    }

    /* ----------------------------------------------------------------- */
    /* FUNCTION  Smatmul :                                               */
    /* . Row major linear form symmetric matrices                        */
    /* multiplication routine.                                           */
    /* ----------------------------------------------------------------- */

    #define  GAP(k, i)   ((k <= i) ? gap : 1)

    void  Smatmul(int a[], int b[], int c[], int n)
    {
        int  index;
        int  i, j, k;
        int  p, q, gap;
        int  sum;

        index = 0;
        for (i = 0; i < n; i++)
            for (j = 0; j < n; j++) {
                sum= 0;  gap = n - 1;
                p = i;  q = j;
                for(k = 0; k < n; k++, p+=GAP(k,i), q+=GAP(k,j), gap--)
                    sum += a[p]*b[q];
                c[index++] = sum;
            }
    }
```

Download at http://www.pin5i.com/

问题 7.8 找零钱问题（CHANGE.C）

某个国家一共发行了$a_1,a_2,a_3,\cdots,a_k$种不同面额的钞票，为了方便起见，假设$a_1<a_2<\cdots<a_k$。现在手上有n，请问要如何把n兑换成$a_1,a_2,a_3,\cdots,a_k$的这些钞票，使得所用钞票的量为最少。

【解答】

这个问题并不难，这里讲解的办法并不十分好，但却很简单，而且比一些好的、复杂的方法容易理解。

假设n是要兑换的钱数，k是能够兑换的面额数目，如果要兑换的是100元，只能兑换成1元、5元、7元的话，于是n=100、k=3。用一个数组，base[ ]把面额先存好，因此就是base[0]=1、base[1]=5、base[2]=7。注意，如果面额没有1元的话，就很有可能某些值兑换不出来（为什么？），再用另一个数组money[ ]存放兑换的钞票数，因此money[i]就是i兑换出来的钞票数目，因为money[0]表示0元的兑换数，自然就是0，同理money[1]一定是1。注意，只保留了兑换出来的钞票数，而没有保留哪一种钞票有多少张，不过要做到这一点并不难，把程序稍作修改就可以了，留作习题。

假设现在要兑换i元；先看看i-base[0]、i-base[1]、…、i-base[k-1]是什么。比如i=10，而钞票面额是{1,3,4}，于是这几个值正是$10-1=9$、$10-3=7$、$10-4=6$，换句话说，如果已经兑好了9元、7元或6元的话，要兑成10元，就只不过多加一张钞票而已。因此，如果已经兑换好了i-base[0]、i-base[1]、…、i-base[k-1]，要兑换成i元，就只不过是多加一张钞票；在i-base[j]的情况下，就是多一张j元。但是，如何得知兑换成i-base[i]元要多少张钞票呢？回想money数组，该钞票张数就应该在money[i-base[j]]中。如果已经兑换出i-base[j]元，用了money[i-base[j]]张钞票，于是多加一张base[j]元就可以兑出i元了，这样就用了money[i-base[j]]+1张钞票。

不过题目所要求的是钞票张数最少的点，因此就要求出各money[i-base [j]]+1的极小值（j=1,2,…,k），再存到money[i]中，于是money[i]就是兑换i元的最少钞票张数了。然而，要注意到一件事，因为i-base[j]一定小于i，所以在计算money[i]时，在i之前的值就应该先算出结果，这样money[i-base[j]]才会有一个有意义的值。

例如，{1,3,4}来兑10元，如表16-1所示，money[0]为0、money[1]是1，所以从i=2起。

表 16-1

i	money[ ]	i-base[j]
0	0	-
1	1	-
2	1	2-1,2-3*,2-4*
3	1	3-1,3-3,3-4*
4	1	4-1,4-3,4-4

Download at http://www.pin5i.com/

（续）

i	money[ ]	i-base[j]
5	2	5-1,5-3,5-4
6	2	6-1,6-3,6-4
7	2	7-1,7-3,7-4
8	2	8-1,8-3,8-4
9	3	9-1,9-3,9-4
10	3	10-1,10-3,10-4

注：*表示小于零，不必理会极小值

兑换 10 元要 3 张钞票，结果是两张 3 元，一张 4 元，用表 16-1 中 10-3 与 10-4 倒溯就会发现两者的结果相同。10-3 表示这是从 7(10-3)元而来，再加一张 3 元；但 7 元用 7-3 倒溯是先一张 3 元，再一张 4 元；不过用 7-4 倒溯，则是一张 4 元，再一张 3 元，结果都相同。

程序的观念就是如此，因为很简单，所以不作说明。

【习题】

（1）请扩充这个程序，让它至少可以报告出一组张数为最少的钞票面额（注意，答案可能不止一组）。

（2）请举一个例子，说明如果没有 1 元面额的话，就不见得能兑出所有值了。

（3）前面列举的实例说明，用{1,3,4}兑 10 元所得来的结果（钞票张数）比一般用除法、余数的方式少。能够找出除了{1,3,4}之外的其他例子吗？

【参考文献】

以下几篇文章都讨论了兑换的问题，或许可以在这个问题上有进一层的认识，这里所提的解法，其实就是一个动态规划（Dynamic Programming）的技巧。

1 L.Chang and J.F.Korsh. Canonical Coin Changing and Greedy Solutions, *Journal of ACM*. Vol.23(1976), pp.418～422

2 S.K.Chang and A.Gill. Algorithmic Solution of the Change-Making Problem, *Journal of ACM*. Vol.17(1970), pp.113～122

3 M.J.Magazine, G.L.Nemhauser and L.E.Trotter, Jr.. When the Greedy Solution Solves a Class of Knapsack Problems, *Operations Research*. Vol.23(1975), pp.207～217

4 J.W.Wright. The Change-Making Problem, *Journal of ACM*. Vol.22(1975), pp.125～128

【程序】

程序 16-10（CHANGE.C）

```
/* ------------------------------------------------------------------ */
/* PROGRAM change :                                                   */
/*   Given a set of based value base[0],...,base[k] and               */
/* and input number, this program finds a representation              */
/* of the input number with base values such that the                 */
/* number of base values used are at minimum.                         */
```

Download at http://www.pin5i.com/


```
/*    This program employs the dynamic programming tech.     */
/*                                                           */
/* Copyright Ching-Kuang Shene          July/08/1989         */
/* ---------------------------------------------------------- */

#include  <stdio.h>
#include  <stdlib.h>

#define   MAXSIZE   100
#define   min(a,b)  ((a) <= (b) ? (a) : (b))

void  main(void)
{
     int      money[MAXSIZE+1];
     int      base[] = { 1, 3, 4 };
     int      k      = sizeof(base)/sizeof(int);
     int      n;
     int      i, j, MIN;
     char     line[100];

     printf("\nMinimum Money Change Program");
     printf("\n============================");
     printf("\n\nBase Values : ");
     for (i = 0; i < k; i++)
          printf("%d ", base[i]);
     printf("\n\nYour input please --> ");
     gets(line);
     n = atoi(line);

     money[0] = 0;                  /* money[0] can not be used     */
     money[1] = 1;                  /* money[1] is the basic        */
     for (i = 2; i <= n; i++) {     /* for 2 dollars to n...        */
          MIN = n;                  /* find the minimum # by        */
          for (j = 0; j < k; j++)/* looking back.                   */
               if (i >= base[j])
                    MIN = min(money[i-base[j]]+1, MIN);
          money[i] = MIN;                /* min. computed for i     */
     }
     printf("\n\nMinimum = %d", money[n]);
}
```

问题 7.9　背包问题（KNAPSACK.C）

如果有 n 个对象，它们的计量单位各为 $k_1, k_2, \cdots, k_n$，现在有一个可以容纳 K 单位的背包，请写一个程序，找出有没有办法在 1、2、…、n 中选出 p 个元素，使得计量单位的总和为 K，因此可以刚好把背包装满。这就是有名的背包问题（Knapsack Problem）。

Download at http://www.pin5i.com/

【解答】

解决这个问题最好的方法还是动态规划（Dynamic Programming），问题 7.8（找零钱）已经涉及到了。

在动态规划中，如果一个问题有若干个影响的参数，例如 $i_1,i_2,\cdots,i_p$，而问题的解表示为 $P(i_1,i_2,\cdots,i_p)$，于是要找出另一组参数 $i_1,i_2,\cdots,j_p$，并且 $P(i_1,i_2,\cdots,j_p)$已经解出来，再用这个解作为 $P(i_1,i_2,\cdots,i_p)$的解。对于背包问题而言，有两个参数，第一是已知的对象 $k_1,k_2,\cdots,k_n$；第二是背包的总容量 K，如果把问题写成 P(i,j)，表示用 $k_1,k_2,\cdots,k_i$，这 i 个已知的对象可以解出背包容量为 j 的背包问题，换句话说，在 $k_1,k_2,\cdots,k_i$ 这 i 个对象中的某几个的和是 j，因此取出这几个对象就正好可以放满容量为 j 的背包。对于一个新问题 P(p,q)而言，它提出问题：$k_1,k_2,\cdots,k_p$ 之前 p 个对象中，有没有若干个可以组成容量 q 呢？如果比 p 小与 q 小的问题都解决了，P(p,q)是不难解决的，理由如下所述。

P(p,q)是表示用 $k_1,k_2,\cdots,k_p$，之前 p 个对象去解容量为 q 的背包，而且对所有 i≤p、j≤p 的 i,j 而言，P(i,j)都已经解出来了，只是 P(p,q)不曾解出而已。不妨看看目前 p-1 个对象，即 $k_1,k_2,\cdots,k_{p-1}$ 能够解出容量为 q 的背包问题，换句话说，就是看 P(p-1,q)是否有解。如果 P(p-1,q)有解，那么 $k_1,k_2,\cdots,k_{p-1}$ 这 p-1 个对象中就可以凑足容量为 q 的背包，因此根本不用 k_p 的第 p 个对象，所以 P(p,q)的解就是 P(p-1,q)的解。

如果 P(p-1,q)无解，即 $k_1,k_2,\cdots,k_{p-1}$ 之前 p-1 个对象无法凑足容量为 q 的背包，用 $k_1,k_2,\cdots,k_{p-1},k_p$ 之前 p 个对象来凑足容量为 q 的背包时，那么 k_p 就一定能派上用场。不妨把 k_p 装到背包中，因此背包就只剩下 $q-k_p$ 的容量，又因为 k_p 已经用了，所以就要看 $k_1,k_2,\cdots,k_{p-1}$ 中能否凑足剩下来的容量，即 $q-k_p$。要知道这个答案，就要查 $P(p-1,q-k_p)$，如果无解，那么用前 p-1 个对象就凑不出 $q-k_p$，自然用前 p 个就凑不出 q 了；但若 $P(p-1,q-k_p)$有解，用这个解加上 k_p 就凑齐了 q。

综合起来，当要解 P(p,q) 时，就要先看看 P(p-1,q)，如果 P(p-1,q) 有解，就不用 k_p 凑足 q 了，用 $k_1,k_2,\cdots,k_{p-1}$ 就足够，因而 P(p,q) 是有解的。如果 P(p-1,q) 无解，就看 $P(p-1,q-k_p)$，这表示用 $k_1,k_2,\cdots,k_{p-1}$ 解 $q-k_p$，如果 $P(p-1,q-k_p)$指出有解，于是 P(p,q)的解就是 $P(p-1,q-k_p)$的解再加上 k_p；但若 $P(p-1,q-k_p)$表示无解，则 P(p,q)也就是无解的了。

因此从 P(p,q)起，p 与 q 的值会依次降低，直到 p=0 为止，所以要知道 P(0,j)的解是什么。P(0,0)一定有解，因为一个容量为 0 的背包，什么都不放进去，就正好 P(0,0)有解。对于 j≥1 而言，P(0,j)一定是无解，原因是容量大于零的背包一定要放入东西才能有解，而 P(0,j)是不用任何对象解容量为 j 的背包，这是不可能的，所以 P(0,j)无解。

看起来这是个递归的解法，但是却可以用一张表避免递归，把这张表叫做 P，P_{ij} 对应 P(i,j)，因为 P(i,j)会用到比 i 与 j 小的字段，所以在算 P_{ij} 时要确定表格中以 P_{ij} 的左上角的部分中除了 P_{ij} 之外全部都已经算出来了。这一点没有困难，在算这张表时，一列一列或一行一行地算都能符合这个要求。因此到最后，P(n,k)如果有解，题目就是有解了，这正是用 $k_1,k_2,\cdots,k_n$ 可以解出背包容量 K 的问题，表格有 n+1 列、k+1 行。

Download at http://www.pin5i.com/

假设 P(n,k)有解，如何找出凑足 K 的对象呢？用另一张表，Q_{ij} 说明。回想一下前面的两点，在处理 P(p,q)时，如果 P(p-1,q)有解时，用 $k_1,k_2,\cdots,k_{p-1}$ 中的元素就可以凑足 q，不用 k_p，所以 Q_{pq} 中就不必记录任何内容；但是 P(p-1,q)无解，而 P(p-1,q-k_p)有解，就可以先用 $k_1,k_2,\cdots,k_{p-1}$ 的元素凑足 q-k_p（因为有解），再加上 k_p 就可以凑成 q 了，Q_{pq} 中就记下这一点。因此 Q_{pq} 的意义是指出在使用 $k_1,k_2,\cdots,k_p$ 凑成 q 时，是否用到 k_p。

当 P(n,k)指出有解，如何从 Q_{nk} 找出解是什么？只要仿真一下求解的过程就行了。因为 P(n,k)有解，这个解不外乎是由 P(n-1,k)而来（第一种情况），或是由 P(n-1,k-k_n)而来（第二种情况）。如果由第二种情况而来，就要用上 k_n。因此，可以从 P(n,k)起，看看 P(n-1,k)是否有解，如果有解，就停在 P(n-1,k)处，但若无解，那就必定是 P(n-1,k-k_n)有解，所以记录下用到了 k_n，再停在 P(n-1,k-k_n)处。反复地处理，当处理到 P(i,j)时，就查 P(i-1,j)与 P(i-1,j-k_i)，如果前者有解，下一次就从 P(i-1,j)查起，但若无解，那就一定用到了 k_i，因而下一次就从 P(i-1,j-k_i)查起。这项工作一直进行到 i 与 j 有一者为零时为止。

这个解法一共用了多少次处理呢？因为一共有两个(n+1)(k+1)的矩阵，每一个矩阵的元素都要处理一次，所以就一共处理了 2(n+1)(k+1)个元素，换句话说，与 nk 成正比。

在程序中，size[ ]表示对象的容量，n 表示对象个数，而 SIZE 则是背包容量。result[ ]是在背包中各对象的容量，是输出用的，函数值则是背包中有多少个对象。P 与 Q 这两个表格分别为 exist[ ][ ]与 member[ ][ ]。在中央的大 for 循环是用上述的观念为 exist[ ][ ]与 member[ ][ ]填入数据的，即 exist[i][j]为真，就表示用 size[0]、size[1]、…、size[i-1]这 i 个对象可以凑足 j（注意 C 语言中脚码从 0 起的用法）。而 member[i][j]则表示用 size[0]、size[1]、…、size[i-1]凑足 j 时用到了 size[i-1]（注意，不是 size[i]）。在最后的 if…else 中是用来取出解答用的，当 exist[n][SIZE]表示有解时，就把对应的对象容量 size[i]存入 result[ ]，但是因为这个顺序是反的，所以最后用 reverse()函数把 result[ ]反过来排。

【习题】

（1）请用递归的方法来解这个问题。请问，效率会不会高一些？能想出几个不用递归的理由呢？

（2）其实 exist[ ][ ]与 member[ ][ ]根本就可以合二为一，不必用两张表那么浪费，请修改程序完成这项要求。

（3）在题目中，$k_1,k_2,\cdots,k_n$ 这些容量的对象都恰好只有一个，但若是它们不止一个，而是分别有 $m_1,m_2,\cdots,m_n$ 个，现在的背包容量还是 K，为了凑足 K 的容量，对于 k_i 而言，可以不用它，而用 $1,2,\cdots,m_i$ 个。请修改程序，解出这个比较广泛的背包问题。

（4）续题（3），如果使用量并没有条件，要用多少个就用多少个，请在这个前提下解决背包问题。请问，这个问题比上一题难吗？还是根本就是上一题呢？问题 7.8 的找零钱问题与本题有什么关系？

【程序】

程序 16-11（KNAPSACK.C）

```
/* ------------------------------------------------------------------ */
/* FUNCTION  knapsack :                                               */
```

Download at http://www.pin5i.com/


```
/*   Given a list of items with size stored in an array          */
/* size[] and a knapsack of size SIZE, this function             */
/* determines whether there are some items summing up to         */
/* the exact size SIZE.  Those selected items are stored         */
/* in the array result[].  Function value is the number          */
/* of items selected.  NOT_FOUND means the selection is          */
/* impossible.                                                   */
/*                                                               */
/* Copyright Ching-Kuang Shene          Sep/01/1989              */
/* ----------------------------------------------------------    */

#include <stdlib.h>          /* for malloc() and free()          */

#define  YES          1
#define  NO           0
#define  NOT_FOUND   -1

void  reverse(int [], int);

int  knapsack(int size[], int n, int SIZE, int result[])
{
     int  **exist;           /* exist[][] table                  */
     int  **member;          /* member[][] table                 */
     int  count;             /* number of items selected         */
     int  i, j;

     exist   = (int **) malloc(sizeof(int *)*(n+1));
     exist[0]    = (int *) malloc(sizeof(int)*(n+1)*(SIZE+1));
     member  = (int **) malloc(sizeof(int *)*(n+1));
     member[0]   = (int *) malloc(sizeof(int)*(n+1)*(SIZE+1));
     for (i = 1; i <= n; i++) {
         exist[i]    = exist[i-1] + (SIZE+1);
         member[i]   = member[i-1] + (SIZE+1);
     }

     exist[0][0] = YES;          /* for 0, there is solvable     */
     for (j = 1; j <= SIZE; j++) /* but without any item,        */
         exist[0][j] = NO;       /* no solution is possible      */

     for (i = 1; i <= n; i++)                /* for i items..... */
         for (j = 0; j <= SIZE; j++) {       /* solvable for j?  */
             exist[i][j] = member[i][j] = NO;/* default          */
```

Download at http://www.pin5i.com/


```
if (exist[i-1][j]) {          /* i-1 and j solvable? */
    exist[i][j]  = YES;       /* YES, i/j too.       */
    member[i][j] = NO;        /* doesn't inc. i      */
}
else if (j >= size[i-1])      /* over limit?         */
    if (exist[i-1][j-size[i-1]]) { /* back           */
        exist[i][j]  = YES; /* include i             */
        member[i][j] = YES;
    }
}

if (exist[n][SIZE]) {             /* scan back to collect ans*/
    for (count = 0, i = n, j = SIZE; i != 0 && j != 0; )
        if (member[i][j] == YES) {
            result[count++] = --i;
            j -= size[i];
        }
        else
            i--;
    reverse(result, count);
}
    else
        count = NOT_FOUND;

    free(exist[0]);   free(exit);
    free(member[0]);  free(member);
    return count;
}

/* ------------------------------------------------------------ */
/* FUNCTION  reverse :                                          */
/*   Reverse the given array x[].                               */
/* ------------------------------------------------------------ */

#define  SWAP(a, b)  { temp = a; a = b; b = temp; }

void  reverse(int x[], int n)
{
    int  i, j, temp;

    for (i = 0, j = n-1; i < j; i++, j--)
        SWAP(x[i], x[j]);
}
```

Download at http://www.pin5i.com/

问题 7.10 最佳矩阵相乘顺序（PRODSEQ.C）

已知一连串的矩阵，$A_0,A_1,\cdots,A_{n-1}$，要计算它们的连乘积 $A_0\times A_1\times\cdots\times A_{n-1}$；请写一个程序，读入各个矩阵的维度的容量，找出一个使乘法数目最少的计算方式。

【解答】

本书这类的题目已经介绍很多了，这就是所谓的动态规划（Dynamic Programming）的技巧，首先计算 $A_i\times A_{i+1}\times\cdots\times A_j$，它有如下的几个计算方式:

$$
\begin{aligned}
A_i\times A_{i+1}\times\cdots\times A_j &= A_i\times(A_{i+1}\times\cdots\times A_j)\\
&=(A_i\times A_{i+1})\times(A_{i+2}\times\cdots\times A_j)\\
&\ \ \vdots\\
&=(A_i\times\cdots\times A_k)\times(A_{k+1}\times\cdots\times A_j)\\
&\ \ \vdots\\
&=(A_i\times A_{i+1}\times\cdots\times A_{j-2})\times(A_{j-1}\times A_j)\\
&=(A_i\times\cdots\times A_{j-1})\times A_j
\end{aligned}
$$

要如何分解 $A_i\times\cdots\times A_j$，就要看在右边的算法中哪一个使用的乘法数目比较少。如果用 $c_{p,q}$ 表示 $A_p\times\cdots\times A_q$ 所用到的乘法数目，那么 $A_i\times(A_{i+1}\times\cdots\times A_j)$所用的乘法数目就是 $0+c_{i+1,j}$，再加上 A_i 的列数乘以 A_i 的行数再乘上 A_j 的行数，其中 0 是因为 A_i 不必相乘。如果 A_i 的列数是 d_i，行数是 d_{i+1}，就像是问题说明中的输入方式，$A_i\times(A_{i+1}\times\cdots\times A_j)$的乘法数目是 $c_{i+1,j}+d_id_{i+1}d_{j+1}$。同理，$(A_i\times A_{i+1})\times(A_{i-2}\times\cdots\times A_j)$的乘法数目为 $c_{i,i+1}+c_{i+2,j}+d_id_{i+2}d_{j+1}$。一般来说，$(A_i\times A_k)\times(A_{k+1}\times\cdots\times A_j)$的乘法数目是 $c_{i,k}+c_{k+1,j}+d_id_{k+1}d_{j+1}$。

因此 $A_i\times\cdots\times A_j$ 的分解方式也就是它所使用的最少的乘法数目就是:

$$c_{ij}=\min_{1\le k\le j}(c_{i,k}+c_{k+1,j}+d_id_{k+1}d_{j+1})$$

此式中，$i<j$。若能求出 $c_{0,n-1}$，也就是 $A_0\times\cdots\times A_{n-1}$ 的最少乘法数目，问题就解决一半了。下面来分析一下共有多少个 $c_{i,j}$，因为 $0\le i\le j\le n-1$，所以正好可以用一个矩阵的上半部（含对角线）容纳它们，如图 16-5 所示。

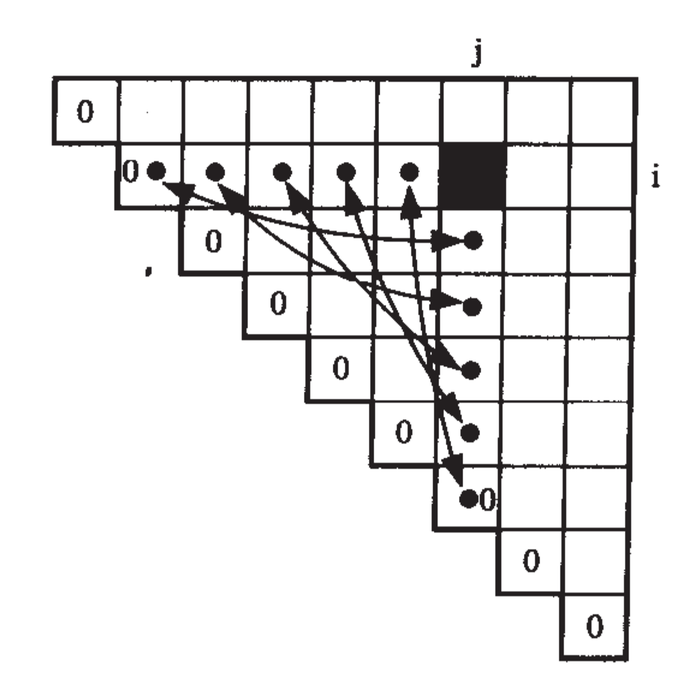


图 16-5

Download at http://www.pin5i.com/

回顾求$c_{i,j}$的式子，要先算出所有$c_{k+1,j}$，$i\leqslant k<j$。由于$i\leqslant k<j$，因此$c_{i,k}$正好是第i列上在$c_{i,j}$左边的元素，而$c_{k+1,j}$则是在第 j 行上$c_{i,j}$下方的元素，$c_{i,k}$与$c_{k+1,j}$匹配的情形如图 16-5 所示；不必顾虑d_i、d_{k+1}、d_{j+1}，因为它们是输入。由此可见，为了要算出$c_{i,j}$，就要算出它左边与下方的元素，所以计算$c_{i,j}$的方法就是从左到右一行一行地算，在每一行中的元素则是从下向上算，最后得到的$c_{0,n-1}$就是所期望的结果。

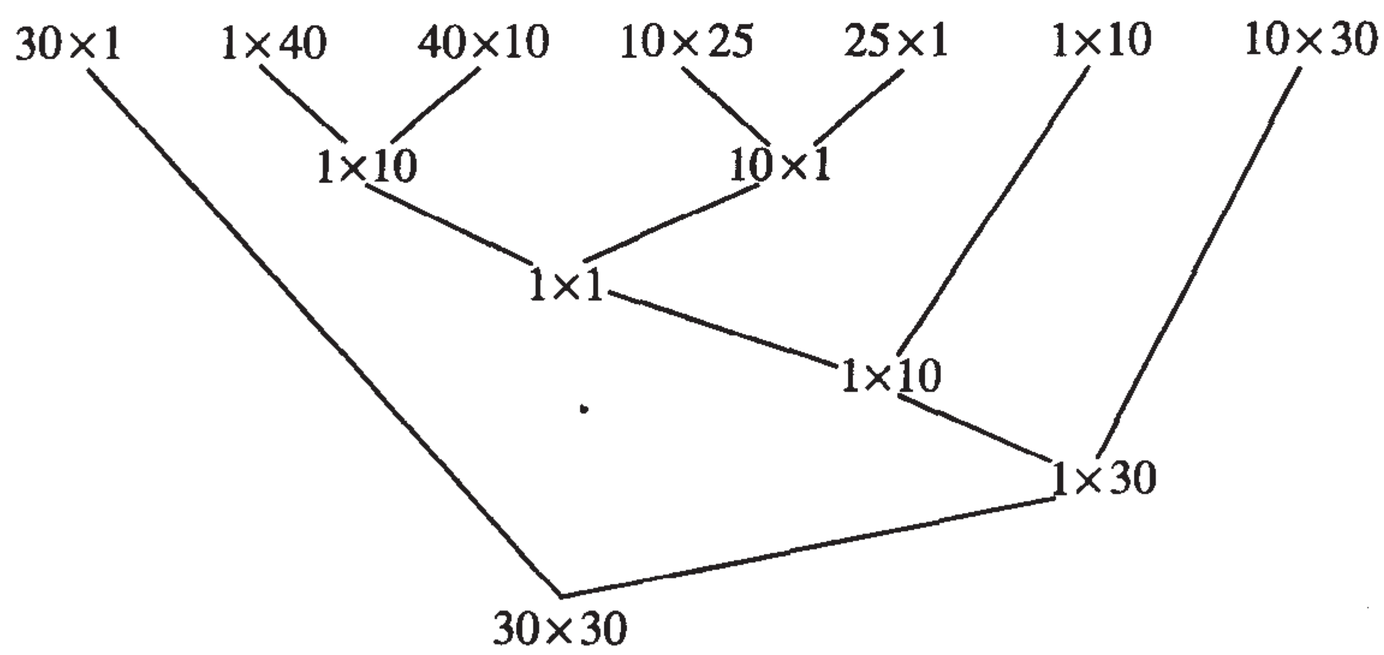
上面的讨论中只告诉$c_{i,j}$是计算$A_i\times\cdots\times A_j$时所需的最少乘法数目，并没有告诉如何分解它，但若仔细些也不难看出一些蛛丝马迹。因为要找出$c_{i,k}+c_{k+1,j}+d_id_{k+1}d_{j+1}$的极小值，这里$i\leqslant k<j$，所以令该式子为最少的 k 就是分解方式。也就是说$(A_i\times\cdots\times A_k)\times(A_{k+1}\times\cdots\times A_j)$的算法是乘法个数最低的。可以再准备一个矩阵$f$，把这个 k 存到$f_{i,j}$中，于是到最后$f_{0,n-1}$中的值，称为 p，就是$A_0\times\cdots\times A_{n-1}$的分解方式，即：

$$A_0\times\cdots\times A_{n-1}=(A_0\times\cdots\times A_p)\times(A_{p+1}\times\cdots A_{n-1})$$

此式算法所需的乘法数目为最少。那么如何算$A_0\times\cdots\times A_p$呢？可以递归地去找$f_{0,p}$就得到了$A_0\times\cdots\times A_p$的分解方式。

在程序 PRODSEQ.C 中，product_sequence()用上面的方式把 cost[ ][ ]与 factor[ ][ ]填满，前者的内容为$c_{i,j}$，后者则是$f_{i,j}$。这两个矩阵的内存都是通过 malloc()得来的；最后它调用 display()把 factor[ ][ ]中的分解方式输出，这一点请读者自己研究。

主程序是在算30×1，1×40、40×10、10×25、25×1、1×10与10×30这 7 个矩阵的连乘积的计算方式，程序的输出相当于：



总共要用 1870 个乘法；如果从左向右算，那么第一与第二个相等要用 1200 个乘法，结果是 30×40 的矩阵；再与 40×10 的矩阵相乘，要用 12000 个乘法，结果是 30×10 的矩阵；再与 10×25 矩阵相乘，要用 7500 个乘法，结果是 30×25 矩阵；再乘以 25×1 的矩阵，要用 750 个乘法，结果是 30×1 矩阵；再与 1×10 矩阵相乘，要用 300 个乘法，结果是 30×10 矩阵；最后与 10×30 矩阵相乘，要用 900 个乘法，因而乘法的总数是：1200+12000+7500+750+300+900=22650 个乘法，要比上面结果慢 12 倍。换句话说，如果用结果来算需要一分钟，那么用从左向右乘就要 12 分钟了。如果是从右向左乘呢？不妨求出来。

【习题】

（1）这个程序一共用了与n^3成正比的比较次数，试证明。

（2）因为矩阵相乘中，要额外的内存存放中间结果，因此虽然计算速度快了，但却需

Download at http://www.pin5i.com/

要很多额外的内存。如果假设额外的内存用完就去掉。以上面的题目为例，在计算 1×40 与 40×10 的结果时，要用 1×10 的内存；在计算 10×25 与 25×1 的乘积时，要用 10×1 的内存，但当把 1×10 与 10×1 的结果求出之后，它们的内存就用不着了。请扩充程序，报告出在最少乘法之下最多会用到多少内存。

（3）请编写一个程序，找出内存使用量最省的计算方法，并且报告出在这个方法下要有多少个乘法运算。

（4）这个程序中，cost[ ][ ]与 factor[ ][ ]两个矩阵都只用了一半多一点，请修改程序，使得不必配置没有用到的部分。

【参考文献】

这里所用的方法并不是最好的，F.Frances Yao 在 1982 年提出了一个加快速度的方法，用与 n^2 成正比的比较次数就够了，用这个方法快了 n 倍。后来，T.C.Hu 与 M.T.Shing 两人在 1982 年与 1984 年连续的两篇文章中指出，n^2 还是太慢，$n\log_2 n$ 就够了，只是不知道是否还有更快的方法。

1 T.C.Hu and M.T.Shing. Computation of Matrix Chain Products. I. *SIAM Journal on Computing*, Vol.11(1982), pp.362～373

2 T.C.Hu and M.T.Shing. Computation of Matrix Chain Products. II. *SIAM Journal on Computing*, Vol.13(1984), pp.228～251

3 F.Frances Yao. Speed-Up in Dynamic Programming. *SIAM Journal on Algebraic and Discrete Methods*, Vol. 3(1982), pp.532～540

当并不要求最佳的解，而只是想要一个近似于最佳解的算法时，Francis Y. Chin 在 1978 年提出一个方法，用与 n 成正比的次数就可以找出一个与最佳解相差不超过 25%的近似解。另外，T.C.Hu 与 M.T.Shing 在 1981 年的一篇文章中也提到一个误差不超过 15.5%的方法，比较次数也与 n 成正比。但是，所有方法中还是此处介绍的程序简单但比较慢，却比较容易理解。

4 Francis Y.Chin. An O(n)Algorithm for Determining a Near-Optimal Computation Order of Matrix Chain Products. *Communications of ACM*, Vol.21(1978), pp.544～549

5 T.C.Hu and M.T.Shing. An O(n) Algorithm to Find a Near_optimum Partition of a Convex Polyson. *Journal of Algorithms*, Vol.2(1981), pp.122～138

【程序】

程序 16-12（PRODSEQ.C）

```
/* ------------------------------------------------------------------ */
/* FUNCTION product_sequence :                                        */
/*   Given a sequence of dimensions of matrices, this                 */
/* function finds out the optimum product sequence such               */
/* that the number of multiplications is minimum.                     */
/*                                                                    */
/* Copyright Ching-Kuang Shene          Aug/17/1989                   */
/* ------------------------------------------------------------------ */
```

Download at http://www.pin5i.com/


```
#include <stdlib.h>                    /* for malloc() and free()   */
#include <limits.h>                    /* for INT_MAX               */

void display(int *[], int, int);
void show_order(int *[], int, int);

void product_sequence(int dim[], int n)
{
    long **cost;                       /* cost matrix               */
    int  **factor;                     /* factorization record      */
    long min_cost;                     /* running minimum cost      */
    long previous, this_one;           /* other two cost value      */
    int  pos, i, j, k;

    /* ----- get memory for dope vector and matrix -----           */

    cost    = (long **) malloc(sizeof(long)*n);
    factor      = (int  **) malloc(sizeof(int)*n);
    cost[0]     = (long *) malloc(sizeof(long)*n*n);
    factor[0]   = (int  *) malloc(sizeof(int)*n*n);
    for (i = 1; i < n; i++) {
        cost[i]     = cost[i-1] + n;
        factor[i]   = factor[i-1] + n;
    }

    for (i = 0; i < n; i++)            /* clear the diagonal        */
        cost[i][i] = 0;

    for (j = 1; j < n; j++)            /* compute each item left to*/
        for (i = j-1; i >= 0; i--){/* right & bot->up           */
            min_cost    = INT_MAX;
            pos = -1;
            for (k = i; k<j; k++){  /* compute min.              */
                previous = cost[i][k] + cost[k+1][j];
                this_one = ((long) dim[i])*dim[k+1]*dim[j+1];
                if (min_cost > previous + this_one)
                    min_cost = previous + this_one, pos = k;
            }
            cost[i][j]  = min_cost; /* save min cost             */
            factor[i][j] = pos;     /* and position              */
        }
    display(factor, n, cost[0][n-1]);  /* display result         */
```

Download at http://www.pin5i.com/


```
        free(cost[0]);                          /* free all memory        */
        free(cost);
        free(factor[0]);
        free(factor);
    }

    /* ------------------------------------------------------------        */
    /* FUNCTION  display :                                                 */
    /*   Given the factorization matrix, this function prints              */
    /* the optimum product sequence by calling show_order().               */
    /* ------------------------------------------------------------        */

    void  display(int *factor[], int n, int total)
    {
        printf("\n\nThe Factorization is ");
        show_order(factor, 0, n-1);
        printf("\n\nNumber of Multiplications to be Perfomed = %d", total);
    }

    /* ------------------------------------------------------------        */
    /* FUNCTION  show_order :                                              */
    /*   The working function to show the optimum product                  */
    /* order.                                                              */
    /* ------------------------------------------------------------        */

    void  show_order(int *factor[], int start, int end)
    {
        int  mid;

        if (start == end)                       /* if there is only 1... */
            printf("M%d", start);               /* show it!              */
        else {
            printf("(");                        /* OR initiate a new seq.*/
            mid = factor[start][end];           /* get the divide pt     */
            show_order(factor, start, mid);/* show 1st half              */
            printf("*");                        /* then a multiplication */
            show_order(factor,mid + 1,end);/* 2nd half                   */
            printf(")");                        /* close this sequence.  */
        }
    }
```

Download at http://www.pin5i.com/

问题 7.11 最短路径问题（SHORTEST.C）

如果有一个矩阵 $D=[d_{ij}]$，其中 $d_{ij}>0$ 表示 i 地到 j 地的距离。若 i 与 j 之间无路可通，d_{ij} 就是无穷大（∞），又 $d_{ij}=0$。编写一个程序，通过这个距离矩阵，把任意两个城市之间的最短距离与其行经路径找出来。

【解答】

先介绍如何找出最短距离，之后再讨论如何找出对应的路径。

为了方便起见，D_{ij} 表示在某个时刻所能发现的从 i 到 j 的最短距离，在开始时，D_{ij} 自然就是输入的距离矩阵，因为除此之外，没有任何进一步的数据。接下来，就可以逐步修正 D_{ij} 的值，使得 D_{ij} 最终可以变成 i 到 j 的最短距离。

注意，对于任何一个城市而言，i 到 j 的最短距离不外乎存在经过 k 与不经过 k 的两种可能，所以可以令 k =1,2,…,n（n 是城市的数目），再检查 D_{ij} 与 $D_{ik}+D_{kj}$ 的值；在此 D_{ik} 与 D_{kj} 分别是目前为止所知道的 i 到 k 与 k 到 j 的最短距离，因此 $D_{ik}+D_{kj}$ 就是 i 到 j 经过 k 的最短距离。所以，若 $D_{ij}>D_{ik}+D_{kj}$，就表示从 i 出发经过 k 再到 j 的距离要比原来的 i 到 j 距离短，自然把新的 i 到 j 的最短距离 D_{ij} 就变成为 $D_{ik}+D_{kj}$ 了，当每一个 k 都查完了，D_{ij} 就是自 i 到 j 的最短距离。编写成程序如下：

【程序】

程序 16-13

```
for (i = 0; i < n; i++)
     for (j = 0; j < n; j++)
          for (k = 0; k < n; k++)
               ...
```

但是这是错的。错在 k 的循环不能在最里面。因为如果 k 循环在最里面，一旦找到了 i→p→j 最短的距离之后，i 到 j 就处理过了，以后不会再改变，因而如果还有机会使 i 到 j 有更短的距离时也不能更正结果了。举个例子，如图 16-6 所示。

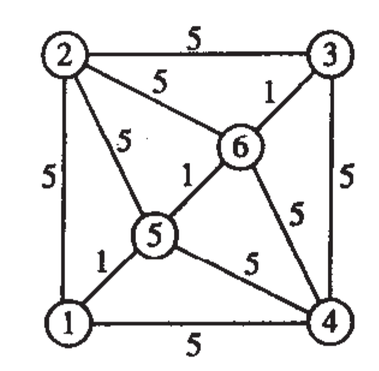


图 16-6

处理 i=1、j=3 的情况。依次查 k=1、2、3、4、5，就会发现 1-1-3、1-2-3、1-3-3、1-4-3、1-5-3 这 5 个 i—k—j 的距离分别是∞、10、∞、10、∞，会得到∞的原因是 1-3、5-3 是不通的，所以距离是∞。于是结果为 10，这当然是错的，因为最短距离是 1-5-6-3，距离值是 3。

从这个例子中可以发现，问题就在于太早把 i—k—j 的距离固定下来了。很多人马上就

Download at http://www.pin5i.com/

想到，把 k 循环放到外面去：

【程序】

程序 16-14

```
for (k = 0; k < n; k++)
    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
            ...
            ...
```

这是正确的想法，但知道这为什么是对的吗？其实不难证明，它的意义是固定了 k，把所有 i 到 j 而经过 k 的距离找出来，用 $D_{ik}+D_{kj}$ 与 D_{ij} 比较，如果前者较小，那么最短路径就是自 i 起经过 k 再到 j。现在因为 k 循环在外，会先把所有 i —j 都处理完后，才会移到下一个 k 去。因此当 k 处理完后，D_{ij} 中的值就是 i 到 j，可能会经过1,2,3,…,k 这几个城市的最短距离（为什么？），当 k=n 也处理完了，D_{ij} 中就是从 i 到 j，会经过1,2,3,…,n 这几个城市的最短距离，正好是答案。

接下来就要看看如何找出最短路径，这得要另一个矩阵 *P*，用它记录最短路径。P_{ij} 的定义是这样的，如果 P_{ij} 中的值是 p，就表示 i 到 j 的最短路径是 i→…→p→j，也就是说 p 是 i 到 j 的最短路径中在 j 之前的最后一个城市。有了这个矩阵之后，要找最短路径就轻而易举了。对于 i 到 j 而言，找出 P_{ij}，令为 p，知道路径是 i→…→p→j；再去找 P_{ip}，如果值为 q，i 到 p 的最短路径为 i→…→q，插回 i 到 j 的最短路径就是 i→…→q→p→j，再去查 P_{iq}，令为 r，就得到 i→…→r→q→p→j，所以一再反复，到了某个 P_{it} 的值是 i 时，就表示 i 到 t 的最短路径为 i→t，就会得出答案。注意，最短路径上的城市是从终点到起点方向找出来的，因此在输出结果时得把它反转过来。

但是，如何填 *P* 这个矩阵呢？回想一下，当 $D_{ij}>D_{ik}+D_{kj}$ 时，就要让 i 到 j 的最短路径改走 i→…→k→…→j 这一条，但是 D_{kj} 的值是已知的，换句话说，k→…→j 这条路也是已知的，所以在 k→…→j 这条路上 j 的上一个城市（亦即 P_{kj}）也是已知的，当然，因为要改走 i→…→k→…→j 这一条路，j 的上一个城市正好就是 P_{kj}。所以一旦发现 $D_{ij}>D_{ik}+D_{kj}$，就把 P_{kj} 存入 P_{ij}。然而，*P* 矩阵的初值是什么？是 P_{ij}=i，在一开始时根本没有最短路径，在 i→j 这一条路上，j 的上一个城市（不管 i 是否有路到 j）都是 i。纵使 i 与 j 不通，也都没有关系（为什么？）。

看看下面 SHORTEST.C 这个程序，函数 floyd()就是找出最短距离与最短路径的操作。程序中 *D* 与 *P* 这两个矩阵就变成 dist[ ][ ]与 path[ ][ ]，也要记住 C 语言数组脚码从 0 开始的事实。dist[ ][ ]中，用 INT_MAX 表示∞。

floyd()中的 MAXSUM()是个集体指令，它相当于求出 $D_{ik}+D_{kj}$，当 D_{ik} 与 D_{kj} 有一个是∞时，就表示有一者无路相通，自然 i 就不能经由 k 到 j 了，所以距离还是∞；若 D_{ik} 与 D_{kj} 都不是∞，才会算出两者真正的和。floyd()非常简单，不用多说。

display_path()是用来显示 *P* 矩阵，path[ ][ ]中所记录下的路径。它首先用 malloc()为 chain[ ]矩阵取得内存，当 i 到 j 相通时，就从 path[i][j]起倒溯，把 j 的上一个城市取出存入 chain[ ]中，最后再通过 reverse()函数把它反过来排，再显示出来。

readin()函数则依照问题说明中的格式读取输入。注意，距离矩阵中的对角线全是 0，而没有读到数据的字段则都是∞（INT_MAX）。

floyd()一共处理了多少个元素呢？这很明显，因为有 3 个相互套着的 for，所以处理的

Download at http://www.pin5i.com/

操作与 n^3 成正比，这里 n 是城市的数目。

【习题】

（1）为什么距离矩阵的对角线是 0，而不用∞？

（2）试证明在某个 k 处理完后，D_{ij} 的值正是 i 到 j 会经过 1、2、3、…、k 这几个城市（但不经过 k+1, …, n）的最短距离。

（3）能否从 floyd()的结果看出两个城市之间有没有路相通？

（4）如果知道 i 与 j 之间是否有路相通，能否通过修改这个程序而得到一个快速的程序？

提示：不要用距离，矩阵中 1 表示通，0 表示不通，因此可以用逻辑运算。此处的程序与著名的 Warshall 方法有多大差异？能导出 Warshall 方法吗？

（5）如果距离 d_{ij} 有可能小于零，这里的方法还能正常工作吗？为什么？

【参考文献】

这个方法是 Robert Floyd 在 1962 年提出的，有关最短距离问题的进一步讨论，可参考 N.Christofides. M. Gondran 与 M.Minoux 合著的书，以及在书中所介绍的参考文献。

1 R.Floyd. Algorithm 97: Shortest Path, *Communications of ACM*, Vol.5(1962), pp.345

2 N.Christofides. *Graph Theory，An Algorithmic Approach*. Academic Press, 1975

3 M.Gondran and M. *Minoux. Graphs and Algorithms*, translated by S. Vajda, John Wiley & Sons, 1984

【程序】

程序 16-15（SHORTEST.C）

```
/* ------------------------------------------------------------------ */
/* FUNCTION  floyd shortest path :                                    */
/*   Given a matrix of path distance of a street map,                 */
/* this function determines the minimum distances between             */
/* all pairs of vertices and its route.  In the distance              */
/* matrix, i->i is filled with zero, while those i-j pairs            */
/* without an edge will be put in an infinity.  On return             */
/* path[][] is the path matrix which could be printed out             */
/* with display_path() function.                                      */
/*                                                                    */
/* Copyright Ching-Kuang Shene          Sep/03/1989                   */
/* ------------------------------------------------------------------ */

#include <stdio.h>          /* for gets() and sscanf()                */
#include <stdlib.h>         /* for malloc() and free()                */
#include  <limits.h>        /* for INT_MAX                            */
```

Download at http://www.pin5i.com/


```
#define      MAXSIZE   20              /* max. size of matrices    */

/* ------------------------------------------------------------     */
/*                  function prototypes                             */
/* ------------------------------------------------------------     */

void  floyd(int [][MAXSIZE], int [][MAXSIZE], int);
void  display_path(int [][MAXSIZE], int [][MAXSIZE], int);
void  reverse(int [], int);
void  readin(int [][MAXSIZE], int *);

/* ------------------------------------------------------------     */
/* FUNCTION  floyd :                                                */
/*   Floyd's All-Shortest Path Algorithm.                           */
/* ------------------------------------------------------------     */

#define  MAXSUM(a, b)    (((a) != INT_MAX && (b) != INT_MAX) ? \
                          ((a) + (b)) : INT_MAX)

void  floyd(int dist[][MAXSIZE], int path[][MAXSIZE], int n)
{
     int  i, j, k;

     for (i = 0; i < n; i++)          /* initialize path matrix   */
         for (j = 0; j < n; j++)
             path[i][j] = i;

         for (k = 0; k < n; k++)      /* for vertex k ...         */
             for (i = 0; i < n; i++) /* update dist. with         */
                 for (j = 0; j < n; j++) /* vertex k.             */
                     if (dist[i][j] > MAXSUM(dist[i][k], dist[k][j])) {
                         path[i][j] = path[k][j];
                         dist[i][j] = MAXSUM(dist[i][k], dist[k][j]);
                     }
}
/* ------------------------------------------------------------     */
/* FUNCTION  display_path :                                         */
/*   This function receives the dist. and path matrices,            */
/* then prints shortest paths of all i-j pairs.                     */
/* ------------------------------------------------------------     */

void display_path(int dist[][MAXSIZE], int path[][MAXSIZE], int n)
```

Download at http://www.pin5i.com/


```
{
    int  *chain;
    int  count;
    int  i, j, k;

    printf("\n\nOrigin->Dest    Dist    Path");
    printf(  "\n------------    ----    ----");

    chain = (int *) malloc(sizeof(int)*n);
    for (i = 0; i < n; i++)          /* scan all i-j pairs       */
        for (j = 0; j < n; j++) {
            if (i != j) {            /* discard diagonal pairs   */
                printf("\n%6d->%4d    ", i+1, j+1);
                if (dist[i][j] == INT_MAX)      /* discard      */
                    printf("  NA    ");         /* un-reachbl   */
                else {
                    printf("%4d    ", dist[i][j]);
                    count = 0;                  /* extract path */
                    k     = j;
                    do {
                        k = chain[count++] = path[i][k];
                    } while (i != k);
                    reverse(chain, count);      /* reverse      */
                    printf("%d", chain[0]+1);   /* prt.         */
                    for (k = 1; k < count; k++)
                        printf("->%d", chain[k]+1);
                    printf("->%d", j+1);
                }
            }
        }
    free(chain);                            /* release memory   */
}

/* ------------------------------------------------------------ */
/* FUNCTION reverse :                                           */
/*   This function reverses the input array.                    */
/* ------------------------------------------------------------ */

#define  SWAP(a, b)    { temp = a; a = b; b = temp; }

void  reverse(int x[], int n)
{
    int  i, j, temp;
```

Download at http://www.pin5i.com/


```
        for (i = 0, j = n-1; i < j; i++, j--)
            SWAP(x[i], x[j]);
    }

    /* ------------------------------------------------------------ */
    /* FUNCTION  readin :                                           */
    /*   Function to read in the distance matrix.                   */
    /* ------------------------------------------------------------ */

    void  readin(int dist[][MAXSIZE], int *number)
    {
        int  origin, dest, length, n;
        int  i, j;
        char line[100];

        gets(line);                  /* read in # of vertices       */
        sscanf(line, "%d", &n);
        *number = n;
        for (i = 0; i < n; i++) {   /* initial dist. matrix        */
            for (j = 0; j < n; j++)
                dist[i][j] = INT_MAX;
            dist[i][i] = 0;          /* disgonal must be zero.      */
        }

        gets(line);                  /* get first data line         */
        sscanf(line, "%d%d%d", &origin, &dest, &length);
        while (origin != 0 && dest != 0 && length != 0) {
            dist[origin-1][dest-1] = length;
            gets(line);              /* get other data lines        */
            sscanf(line,"%d%d%d",&origin,&dest,&length);
        }
    }
```

问题 7.12　产生匹配括号的字符串（PAR_GEN.C）

请编写一个程序，输入一个正整数的值，然后列出所有由 n 个左括号与 n 个右括号正确地组成的字符串。当然，正确的左右括号一定数量相同，所以输入的值要是个偶数。

【解答】

把问题说明中的提示作进一步的探讨，就不难得到想要的结果。如果在产生左、右括号的过程中使用 L 与 R 表示在剩余的过程中还要产生多少个左括号，以及用来匹配已产生

Download at http://www.pin5i.com/

的左括号所需要的右括号数目，那么就得出下面的几种可能。

（1）L 与 R 都是零：这表示已经没有需要产生的左、右括号了，自然一个合用的括号字符串到此产生完成。

（2）R 不是零：正因为 R 不是零，表示还要产生右括号，所以就在字符串下一个位置放入一个“)”，当然 R 就要减 1 反映这个事实，因此下一次就要依 L 与 R-1 进行工作。

（3）L 不是零：这表示还有左括号要产生出来，所以在字符串下一个位置放一个左括号，因此下一个阶段就是对 L-1、R 操作。这样不对！因为产生了一个左括号之后，就要有一个与之相匹配的右括号，所以下一个阶段是 L-1 与 R+1，而不是 L-1 与 R。

这却留下了两个细节。第一，当 L 与 R 都不是零的时候，会有什么情况需要处理。如果 L 与 R 都不是零，这表示左、右括号都还没有产生完成，于是若先产生右括号，那么策略就是“一旦有未匹配的左括号出现了，就先把它们处理掉”；但若先产生左括号，就变成“先尽可能地把左括号产生出来，到都产生完了，再慢慢用右括号来匹配”。这就是两个完全不一样的策略，在程序中采用前者。

第二个细节是，在开始的时候 L 与 R 是多少？如果希望产生长度为 n 的字符串，那自然左、右括号各有 n/2 个，而且左括号一定先出现，因此一开始时 L 是 n/2，而 R 则为 0，因此当第一个左括号被产生出来后，下一个阶段就是 L 为 n/2-1，R 为 1，表示左括号少一个，右括号还有一个未产生。如果是第一个策略，下一个符号就是右括号，因而相继产生的就是()……，但若采用第二个策略，就变成((……。在这里用“……”表示向后的递归程序所补内容。

程序本身并不复杂，parentheses_gen()是准备用的程序，它取得一个足够长的位置用来存放括号，先把它清空，令外在变量 count 为零，表示到目前没有产生任何字符串，接着调用 gen()函数正式工作，工作完成之后归还内存，并且把一共产生了多少个字符串的值返回来。

gen()函数收到字符串的数组 x[ ]字符串长度的一半 n，以及 left_no（表示还要产生多少个左括号）与 right_no（还要产生多少个与已产生的左括号相匹配的右括号），其他的事就是前述的 3 个要点。不过在 else 下求 i 的式子倒是值得一提，因为在 gen()中 n 是字符串长度的一半，因而字符串总长为 2•n，还要产生 left_no 个左括号，当然也就要有同样数目的右括号，所以在未完成的过程中至少要产生 2•left_ no 符号。right_no 是表示为了匹配已产生的左括号而产生出来的右括号，这也是要产生出来的，所以说以后一共要产生 2•left_no+right_no 符号，反过来已经产生的符号总数就是：

$$2\cdot n-(2\cdot left_no+right_no)=2\cdot(n-left_no)-right_no$$

所以下一个产生出来的符号就要放在 x[ ]中的那个位置上。

【习题】

（1）请把这个程序改成不用递归的方式。

（2）请把程序的使用策略改成以先产生足够的左括号，然后与这里的输出结果相比较，会发现什么现象？

（3）如果说左括号大于右括号，这里的程序所产生的正是字典顺序，那么定成右括号大于左括号，不正是反字典顺序吗？如何修改程序？结果与（2）相同吗？

Download at http://www.pin5i.com/

【参考文献】

本题的内容取自 M.C.Er 的一篇很有趣的短文，他把三项看来互不相关的内容以产生相匹配左、右括号的观点来取得联系，这篇文章如下：

M.C.Er. A Note on Generating Well-formed Parenthesis Strings, Lexicograph- ically, *The Computer Journal*, Vol.26(1983), pp.205～207

【程序】

程序 16-16（PAR_GEN.C）

```
/* ------------------------------------------------------------ */
/* FUNCTION parentheses_gen :                                   */
/*   Given an even representing the length of a string,         */
/* this function generates all strings of well-formed           */
/* parentheses in a 'lexicagraphically' order, namely           */
/* '(' > ')'.  This is a recursive program, but it is a         */
/* matter to convert it to the iterative  version.              */
/*                                                              */
/* Copyright Ching-Kuang Shene          July/28/1989            */
/* ------------------------------------------------------------ */

#include <stdlib.h>          /* for malloc()                    */

void gen(char [], int, int, int);

int count;                   /* # of string counter             */

int parentheses_gen(int n)
{
     char *x;
     int i;

     x = (char *) malloc(sizeof(char)*n);   /* get memory       */
     for (i = 0; i < n; i++)     /* initialize it to blank      */
         x[i] = ' ';
     x[i] = '\0';
     count = 0;                  /* set counter to zero.        */
     gen(x, n/2, n/2, 0);        /* and GO! gen n/2 ('s & )'s   */
     free(x);                    /* release memory              */
     return count;               /* return counter              */
}

/* ------------------------------------------------------------ */
```

Download at http://www.pin5i.com/

```
/* FUNCTION  gen :                                                    */
/*   This function receives # of ('s and # of )'s, then               */
/* generates corresponding # of ('s and )'s.                          */
/* ------------------------------------------------------------------ */

void  gen(char x[], int n, int left_no, int right_no)
{
      int  i;

      if (left_no == 0 && right_no == 0) /* balanced ?                */
          printf("\n%5d  %s", ++count, x);   /* YES, show             */
      else {                     /* not yet balanced...               */
          i = 2*(n - left_no) - right_no;    /* get next loc          */
          if (right_no != 0) {/* if too many )'s left                 */
              x[i] = ')';     /* gen. a ) and keep trying             */
              gen(x, n, left_no, right_no - 1);
          }
          if (left_no != 0) { /* now ('s=)'s, still have (            */
              x[i] = '(';     /* gen. a ( and add one )               */
              gen(x, n, left_no - 1, right_no + 1);
          }
      }
}
```

问题 7.13　稳定伴侣问题（STABLE.C）

有 n 个男孩 $m_1, m_2, \cdots, m_n$，还有 n 个女孩 $w_1, w_2, \cdots, w_n$。每一个男孩 m_j 都依照他喜爱这 n 个女孩的程度列成一张表，最喜欢的女孩排在第一位，最不喜爱的女孩排在第 n 位。同样地，每一个女孩 w_i 也依照她喜爱 n 个男孩的程度列成一张表。请写一个程序，把每一个男孩与女孩的喜爱表格读入，并且把男孩与女孩一一配对，使得：如果 m_p 与 w_q 为一对的话，那么第一，对 m_p 的喜爱表格中排在 w_q 之前的女孩而言，她的伴侣在其表格中一定排在 m_p 之前；第二，对 w_q 的喜爱表格中排在 m_p 之前的男孩而言，他的伴侣在其表格中一定排在 w_q 之前。这就是稳定伴侣（Stable Marriage）问题。

【解答】

稳定伴侣有一个非常简单的解法。以男孩对女孩求婚为主，所以这个解就叫做男士为主的稳定伴侣解。当然，换成女士主动，就会得到女士为主的稳定伴侣解。

首先，假设所有男孩与女孩都没有对象，然后就一个个地处理男孩（叫做男士为主的稳定伴侣解）。随便找一个还没有对象的男孩 m，再找出在该男孩的喜爱表格中下一个女孩 w（第一次是最中意的、第二次是次中意、……），看看这个女孩是否有对象，如果没有，

Download at http://www.pin5i.com/

就与她配成一对。如果这个女孩已经有了对象，就看看她目前的对象是不是“稳定”的，也就是说她能否找到更好的对象。这件事不难做，如果 w 目前的对象是m′，而在她的喜爱表格中m′排在 m 前面，那么到目前为止，w 还是在最佳的状况，但若是 m 排在m′之前，那么原来 w 与m′的对象就不是最好的了。在这种情况下，就把 w 与m′的约定解除，还m′一个自由身，而让 w 与 m 配对就行了；反之，若在 w 的表格中m′排在 m 前面，m 自己就高攀不上 w 了，所以就要找他心目中的下一位白雪公主。

就这样，男士不断地“降格以求”，而女士则不断“升高水平”，到最后就会达到一个均衡的状况，这就是“稳定伴侣”的解。

这个解法有一个很重要的事实：不管选择男孩的顺序如何，到最后不但一定可以到达一个均衡的状况，而且答案肯定相同，所以究竟如何去选择一个没有对象的男孩并不是一个大问题，这一点极大地帮助简化了程序编写的工作。

【程序说明】

第一，要想想如何表示没有对象的男孩。在上一段讲过，先选哪个男孩处理并不会改变问题的结果；另外，一旦有某人比m′的条件还好，m′就要让贤，而变回单身。正因为如此，用一个堆栈是相当方便的，因为永远取出顶上的男孩，当m′变回单身时，就放回堆栈顶上就行了。程序中，waiting[ ]就是这个堆栈，top 永远指向堆栈的顶点。

第二，每一个男孩都要知道下一次（如果有的话）要转而找哪一个女孩，因此要用一个数组 next[ ]记录下一个对象。在程序中，man[ ][ ]与 woman[ ][ ]分别是男孩与女孩的喜爱表格，man[i][ ]与 woman[j][ ]则是第 i 号男孩与第 j 号女孩的表格。所以，如果令 next[i]表示第 i 号男孩在下一次要找的对象是其表格中的第几个的话，那么 man[i][next[i]]正是那个女孩的号码。

第三，在上面提到的解法中也指出：当男孩 m 挑中女孩 w，但 w 却已有对象m′时，要知道在 w 的喜爱表格中 m 是否排在m′的前面。这不难做，笨方法是用 m 与 woman[w][ ]的元素一个一个比对，到找出等于 m 的元素为止，假设是在第 j 行；再用同样方法去找出m′在第 j′行，如果 j< j′，那么 w 就比较中意 m 了。不过，这太浪费时间了，可以准备另一个矩阵 rank[ ][ ]，如果 woman[w][j]=m，也即第 w 号女孩的第 j 个选择是第 m 号男孩时，就令 rank[w] [m]=j，rank[p][q]=r 的意义，就是男孩 q 在女孩 p 的喜爱表格中的第 r 位。

第四，程序要返回配对的结果，为了查阅方便，传回两个数组 m_eng[ ]与 w_eng[ ]。m_eng[i]=j 表示男孩 i 与女孩 j 配对，而 w_eng[ ]则是以女孩为主的结果，当 m_eng[i]=j 时，就一定会有 w_eng[j]=i。

有了这些内容，程序就非常简单了。首先，调用 get_mem()这个函数为 waiting[ ]、next[ ]与 rank[ ][ ]取得内存，并且把 rank[ ][ ]的元素用上述的办法填入值，还在 stable_marriage()中把 w_eng[ ]定成 FREE，表示女孩们都没有对象。

程序中的 for 循环每次把堆栈顶端的男孩找出来，存入 boy（注意，并没有从堆栈中剔除），然后找出 boy 的喜爱表格中的下一个女孩，存入 girl。接着，如果 girl 目前没有对象，就把 boy 与 girl 配成一对，再把这个男孩从堆栈中剔除。但若 girl 已有对象，她的对象就存放在 w_eng[girl]中，因此就得在 rank[ ][ ]中查一下究竟 girl 比较中意 boy 还是中意现任的男孩 w_eng[girl]。如果是前者，就与 w_eng[girl]解除约定，还她自由身，当然，w_eng[girl]

Download at http://www.pin5i.com/

就得放到堆栈中了；注意，把 w_eng[girl]放入堆栈是把 boy 所在的位置盖过去而已。最后，把 boy 与 girl 配成一对。如果 girl 还是比较中意现任男孩 w_eng[girl]，于是就又回到 for 的开头，而 boy 仍旧在顶端，因此让他向下一个女孩展开攻势。注意(next[boy])++的写法。做完之后，通过 release_mem()把内存归还系统。

【习题】

（1）请把程序改成以女孩为主动的写法，答案会相同吗？

（2）如果把男孩与女孩随便搭配，但保证一个男孩配一个不同女孩，请编写一个程序，检查这个配对方式是不是“稳定伴侣”的结果？一共作了多少次比较？

（3）续题（2）。如果通过产生所有可能的配对方式，再用（2）的方法来查出“稳定伴侣”的解，这个做法会不会比较快？提示：一共有多少种配对方式？

（4）请证明此处的方法会处理与 n^2 成正比对男女，这里 n 是男或女的总数。

【参考文献】

这个问题最先出现在 1962 年 D.Gale 与 L.S.Shapley 的一篇文章中，他们提到学生选学校（学校也选学生）等的类似问题，并且在这篇文章中提出了上面的解法，通常就叫做基本的 Gale-Shapley 方法。这个方法自然有许多扩充与变形，但不属于本书的范围。1976 年 D.Knuth 的一本小册子（法文），后来引发了许多计算机科学家研究这个问题。就在 1989 年，Gusfield 与 Irving 还就这个问题写成一本厚达 240 页的专著，有兴趣的朋友不妨一读。习题（4）中要求证明 Gale-Shapley 方法一共处理了与 n^2 成正比的男女配对，事实上，稳定伴侣问题在最坏的情况下，不论用的是什么方法，一定是要处理与 n^2 成正比的男女配对的，所以说 Gale-Shapley 方法是最佳方法（Optimal）。

1 D.Gale and L.S.Shapley. College Admissions and the Stability of Marriage, *American Mathematical Monthly*. Vol.69(1962), pp.9～15

2 D.Gusfield and R.W.Irving. *The Stable Marriage Problem: Structure and Algcrithms*, The MIT Press, 1989

3 D.E.Knuth. *Mariages Stables*, Les Presses de 1Universite Montreal. 1976

【程序】

程序 16-17（STABLE.C）

```
/* ------------------------------------------------------------ */
/* FUNCTION  stable_marriage :                                  */
/*   Stable Marriage problem by Gale-Shapley Algorithm.         */
/* Given man's and woman's preference matrices man[][] &        */
/* woman[][], this function determines the stable marriage      */
/* couples in man-oriented order and returns the marriage       */
/* by two arrays, namely m_eng[] and w_eng[].                   */
/*                                                              */
/* Copyright Ching-Kuang Shene          Sep/05/1989             */
/* ------------------------------------------------------------ */
```

Download at http://www.pin5i.com/


```
#include <stdlib.h>          /* for malloc() and free()          */

#define      SIZE       10      /* size of preference matrix    */
#define      FREE       -1      /* means no engagement          */

/* ------------------------------------------------------------ */
/*                function prototypes                           */
/* ------------------------------------------------------------ */

void  get_mem(int [][SIZE], int, int ***, int **, int **);
void  release_mem(int **, int *, int *);

/* ------------------------------------------------------------ */
/* FUNCTION  stable_marriage :                                  */
/*   The main routine.                                          */
/* ------------------------------------------------------------ */

void  stable_marriage(int man[][SIZE], int woman[][SIZE], int n,
                      int m_eng[], int w_eng[])
{
     int  **rank;                    /* pointer to rank[][]      */
     int  *waiting;                  /* pointer to waiting stack*/
     int  *next;                     /* pointer to next to try   */
     int  boy, girl, top;            /* working variables        */

     get_mem(woman, n, &rank, &waiting, &next); /* get mem       */
     for (girl = 0; girl < n; girl++)/* no one engaged.          */
         w_eng[girl] = FREE;

     for (top = n - 1; top >= 0; ) {/* for a FREE man...         */
         boy  = waiting[top];                /* get his name and */
         girl = man[boy][(next[boy])++];     /* girl in list     */
         if (w_eng[girl] == FREE) {  /* is she free?             */
             w_eng[girl] = boy;              /* YES, engage with */
             m_eng[boy]  = girl;             /* this girl.       */
             top--;                          /* pop off this guy */
         }
         else if (rank[girl][boy] < rank[girl][w_eng[girl]]) {
             waiting[top] = w_eng[girl];/* kick off the          */
             w_eng[girl] = boy;      /* current fiance of        */
             m_eng[boy]  = girl;     /* this girl. Engage        */
         }
     }
     release_mem(rank, waiting, next);  /* release all mem.       */
}
```

Download at http://www.pin5i.com/


```
/* ----------------------------------------------------------- */
/* FUNCTION  get_mem :                                         */
/*   Allocate memory and initialize.                           */
/* ----------------------------------------------------------- */

void get_mem(int w[][SIZE], int n, int ***rank, int **wait, int **next)
{
    int  i, j;

    *rank = (int **) malloc(sizeof(int *)*n);
    *wait = (int *) malloc(sizeof(int)*n);
    *next = (int *) malloc(sizeof(int)*n);
    (*rank)[0] = (int *) malloc(sizeof(int)*n*n);
    for (i = 1; i < n; i++)
        (*rank)[i] = (*rank)[i-1] + n;

    for (i = 0; i < n; i++) {           /* make ranking list    */
        (*wait)[i] = n - 1 - i;         /* n-1,...,1,0 in stack */
        (*next)[i] = 0;                 /* next girl to try     */
        for (j = 0; j < n; j++)         /* for the j-th boy....*/
            (*rank)[i][w[i][j]] = j;    /* save his rank        */
    }
}
/* ----------------------------------------------------------- */
/* FUNCTION  release_mem :                                     */
/*   Release all allocated memory.                             */
/* ----------------------------------------------------------- */
void  release_mem(int **rank, int *wait, int *next)
{
    free(rank[0]);
    free(rank);
    free(wait);
    free(next);
}
```

问题 7.14　单调矩阵的极值（MONO_MAX.C）

在这个题目中介绍一个所谓的单调（Monotone）矩阵的观点。如果一个矩阵中的第 i 列上的极大值所在的那一行记为 i_j；若 $i<k$ 时，$i_j \leqslant k_j$，也就是在上方的列的极大值会座落在下方的列的极大值的左边，满足这个条件的矩阵就叫做一个单调矩阵。请编写一个程序，接收一个单调矩阵，找出这个单调矩阵中每一列的极大值。请估计一下程序用了多少次比较。

【解答】

说明做到只查 $n \log_2 m$ 成正比的矩阵元素就可以找出单调矩阵中每一列的极大值，关

Download at http://www.pin5i.com/

键就在于如何发挥“单调”这一性质。使用分而治之的方法。

可以从矩阵 x[ ][ ]中间的一列开始，如果矩阵有 m 行 n 列，那就是第 m/2 列，于是求出第 m/2 列上的极大值，因为第 m/2 列有 n 个元素，求极大值比较 n 次就足够了，如图 16-7 所示。

如果极大值出现在第 m/2 列第 j 行，于是在矩阵的上半部，由于“单调”的限制，它们的极大值不可能在第 j 行右边，同时，下半部的各列的极大值都不会在第 j 行的左边，因此递归地解左上角（第 0～m/2-1 列，第 0～j 行）与右下角（第 m/2+1～m-1 列，第 j～n-1 行）就可以了。

程序中 get_max()就是用这个观点写成的，它收到 T、B、L 与 R 四个值，表示矩阵中要找极大值的上方列、下方列、左方行与右方行的号码，而要在这个区域中找每一列的极大值。首先，它求出这个区域的中间列(T+B)/2，再在这一列上找出极大值，把这个极大值所在行的号码存入 loc[ ]中对应该列的位置，最后再用图 16-7 的观点递归地调用自己。

如图 16-8 所示的粗线条表示为了要找极大值所查过的元素位置。

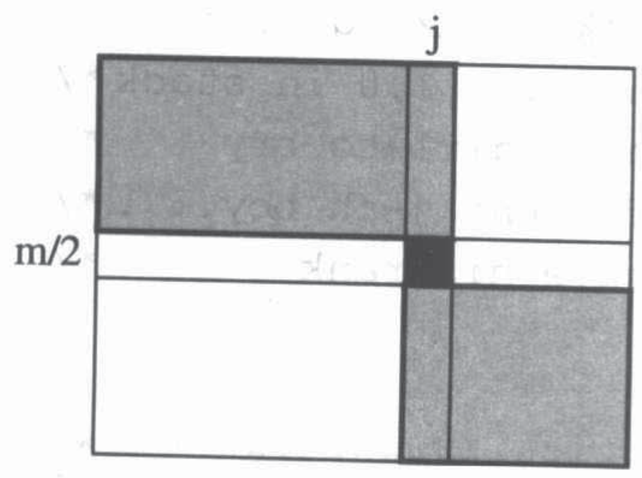


图 16-7

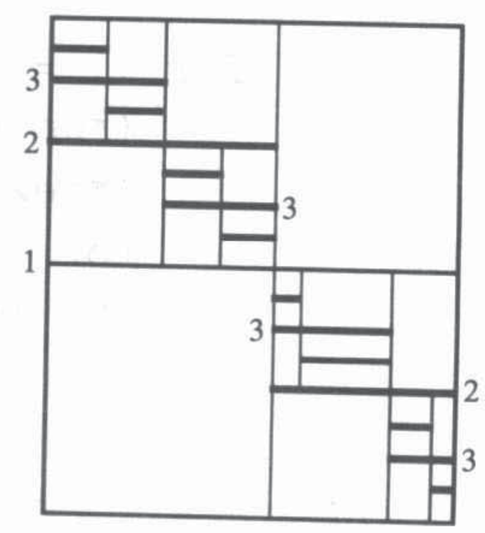


图 16-8

从图 16-8 中不难看出，第一次查了 n 个元素，第二次就有两个段落，每一个段落都要查一次，虽然不知道两个段落各有多少个元素，但是总元素个数是 n+1（因为在垂直分隔线上的元素是两部分都有的），第三次分割时要处理 n+3 个元素，第四次时是 n+7 个，…一般来说，第 i 次分割时要处理 $n+2^i-1$ 个元素。m 列会有多少次分割呢？$\lfloor \log_2 m \rfloor$ 次，原因在于第一次分割有一列、第二次两列、第三次 4 列、第十次 8 列、……，当然“$\log_2 m$”次就可以涵盖 m 列；这里$\lfloor \log_2 m \rfloor$表示比 $\log_2 m$ 大的最小整数，例如“3.14”=3。为了方便起见，就写成 $\log_2 m$，因此处理元素的总数是：

$$
\begin{aligned}
&(n+1)+(n+3)+\cdots+(n+2^i-1)+\cdots+(n+2^{\log_2 m}-1) \\
&=\sum_{i=1}^{\log_2 m}(n+2^i-1) \\
&=n\log_2 m+\sum_{i=1}^{\log_2 m}2^i-\log_2 m \\
&=(n-1)\log_2 m+2m-2
\end{aligned}
$$

当列数（m）比行数（n）小时，上式比 $n\log_2 m$ 小，所以程序大约要比较 $n\log_2 m$ 次，这比 m•n 次少很多了。

Download at http://www.pin5i.com/

【习题】

（1）为什么分解第 i 次求极大值时要处理 $n+2^i-1$ 个元素？请证明。

（2）如果 m 大，n 小，$(n-1)\log_2 m+2m-2$ 会是什么情况？请讨论这一个在解答中没有讲到的情况。

【参考文献】

如果已知一个凸多边形，对于每一个顶点而言找出距它最远的所有顶点。这是一个经历多年都不曾被有效率地解出的问题。直到 1987 年，Alok Aggarwal、M.M.Klawe、S.Moran、P.Shor 与 R.Wilber 才得到一个合理的解答，不过并不完全，完全的解是 A.Aggarwal 与 D.Kravets 在 1989 年做出来的。他们的解答背后所隐藏的就是在单调矩阵与全单调（Totally Monotone）矩阵中找每一列极大值的技巧，对计算几何学有兴趣的朋友不妨看看他们的文章以及文章中列出的参考文献，这里所提的问题不过是一个最简单的情形。

1 A.Aggarwal, M.M.Klawe, S.Moran, P.Shor and R.Wilber, Geometric Applica tions of a Matrix-Searching Algorithm, *Algorithmica*. Vol.2(1987), pp.195～208

2 A.Aggarwal and D.Kravets, A Linear Time Algorithm for Finding All Farthest Neighbors in a Convex Polygon, *Information Processing Letters*. Vol.31(1989), pp.17～20

【程序】

程序 16-18（MONO_MAX.C）

```
/* ------------------------------------------------------------ */
/* FUNCTION monotone_max :                                      */
/*   Given a monotone matrix, this finds the maximum of         */
/* each row in O(n*log(n)) time with divide-and-conquer         */
/* technique.                                                   */
/*                                                              */
/* Copyright Ching-Kuang Shene          Aug/03/1989             */
/* ------------------------------------------------------------ */

#include <limits.h>          /* for INT_MIN                     */

#define  SIZE  10
/* ------------------------------------------------------------ */
/*              function prototypes                             */
/* ------------------------------------------------------------ */
void monotone_max(int [][SIZE], int, int, int []);
void get_max(int [][SIZE], int, int, int, int, int []);

void monotone_max(int data[][SIZE], int m, int n, int loc[])
{
     get_max(data, 0, m-1, 0, n-1, loc);
}
/* ------------------------------------------------------------ */
/* FUNCTION get_max :                                           */
/*   The working routine for monotone matrix max. finder.       */
```

Download at http://www.pin5i.com/


```
/* Given a monotone matrix, its top(T), bottom(B), left(L)      */
/* and right(R)bound,this function finds the maximum of         */
/* each row by divide-and-conquer technique and stores the      */
/* column numbers in which the max.occurres in loc[].           */
/* ------------------------------------------------------------ */
Void get_max(int data[][SIZE].int T,int B, int L,int R,int loc[])
}
    int mid_row=(T+B)/2;
    int pos,max;
    int j;

    if(T<=B){                                /*if we still have more */
        for(max=INT_MIN,j=L;j<=R;j++)                              */
            if(data[mid_row][j]>max)         /*get max.              */
            max=data[mid_row][j];pos=j;
        loc[mid_row]=pos;                    /*save max.location     */
        get_max(data,T,mid_row-1,L,pos,loc);
        get_max(data,mid_row+1,B,pos,R,loc);
    }
}
```

问题 7.15　向量分类（CLASSIFY.C）

已知一组元素 $X_1,X_2\cdots,X_m$，其中每一个元素都是有 n 个量的向量，即 $X_i=[x_{i1},x_{i2},\cdots,x_{in}]$，这里 n 是个固定值。由于这些量 x_{ij} 并不一定是数值，因此比较大小有时不见得有意义，所以假定只能对它们做比较相等或不相等的操作。请写一个程序，定出合适的数据结构，处理这些 $X_1,X_2\cdots,X_m$，把相同的归在一类。X_i 与 X_j 相同，就是指 $x_{i1}=x_{j1}$、$x_{i2}=x_{j2}$、…、$x_{in}=x_{jn}$。在编写程序时，不妨假定 x_{ij} 已经被编写成整数，但不能比较大小。

【解答】

在解法中，用连接串行存储向量，这样可以轻而易举地达成所有要求。在开始做任何工作之前，先建立好一个连接串行，在串行中的每一个节点都有两个数据，第一个是列数，也正是向量的号码，第二个则是指向下一个元素的指针。在建立这个连接串行时，一共有 m 个元素（列的数目），其中的内容就依次是 0、1、2、…、m-1。可参看程序中 Node 的类型定义。

程序中工作量比较大的是 dispatch()函数，它接收一个其中内容为列号码（不一定是包含所有的列，事实上通常只有一小部分），以及一个行的号码 column，表示要求 dispatch()用 column 这一行中的值做分类的工作。dispatch()的工作很简单，它只不过把交给它的串行切成两个而已，在第一个串行中的元素，它的第 column 行的值与串行中的第一个元素的第 column 行的值相同，而另一个串行中的则是不满足这个条件的元素。例如，如果串行中有 2、4、7、9 四列，而它们的第 column 行的值分别是 3、1、2、3，于是 dispatch()做完后的结果就是串行一为 2 与 9，串行二是 4 与 7。因为 dispatch()要有把一个元素从串行中删除

Download at http://www.pin5i.com/

的动作，所以用两个指针 head 与 p，从第二个元素开始工作，head 是 p 的上一个元素。在开始时，把第一个元素那一列、第 column 行的值取出放到 value 中。于是往后在 p 所指到的元素所对应的列中、第 column 行的值如果与 value 不相等，就把 p 所指的元素删除，而移到（由 first 到 last 指出头尾）新串行中。在所有元素都查完之后，把新串行的指针当作函数值返回，不过原来用 head 所指向的串行也可能已经被改动过了。

程序中主控的部分是 group()这个函数，它也是接收一个内含有列号码的串行与 column，表示要用某一行分类，n 是总行数。当 column 比 n 还大时，就等于是没有可以分类的行了，所以交来的串行就是最终结果，因而存到 class_list[ ]的下一个元素中（意义稍后就会明白）。如果还有可以分类的行，那么就调用 dispatch()把该串行中的各列依第 column 行分解成两个，list 中的元素各列中的第 column 行的值完全相同，这些列显然还没有分解完成，所以把这个串行递归地交给自己通过下一行（column+1）做分类的工作。在这一部分，可以看到在递归地调用自己之前，list 中的各列在第 column-1 行（含）以前的各行的值都相同，而递归地交给自己的 list 串行中的各列，在第 column 行（含）以前的都相同，在以后的递归调用中就会一直分解它，直到无行可以分类时，那么 list 中的列就全都是相同的了。

当递归地调用函数 group()把 list 分类完成之后，只做了一半的工作，因为 dispatch()建立了另一个串行——other，它的元素的第 column 行的值都与在 list 中的不一样，但 other 串行中第 column 行自然可能有相同的元素，所以调用自己，仍旧用第 column 来把 other 分类。这就是 group()的工作。

如果调用 group()时 list 中的元素是 1、4、9、7、6、10 列，它们的第 column 行的值分别是 3、1、2、1、3、3，于是 dispatch()就会把 list 变成 1、6、10，第 column 行都是 3，而 other 则是 4、9、7。再把 1、6、10 递归地调用自己，而用 column+1 行分类，再把 4、9、7 交给自己仍然用第 column 行分类。当 group()收到 other 之后，把 4 与 7 归为一类，而新的 other 是 9，因为 9 的第 column 行与 4、7 的不同，因此就分成{1,6,10}、{4,7}和{10}三类，当然递次递归调用中还会再细分，但操作就是这样简单。

classify()函数只不过是做准备与收尾的工作而已。它首先为数组 class_list()取得内存，这个数组的元素是指向串行的指针。简单地说，class_list[i]就是在第 i 类的列的节点所构成的指针，接着把这个数组清成 NULL。第二步再建立一个包含了所有列的串行，于是把这个串行交给 group()，让它自第 0 行起分类，当 group()返回控制权之后，class_list[ ]中就已经装好分类过的串行的指针了，个数是存放在 class_no 中。最后一步，就是把 class_list[i]中的列（表示在第 i 数）都标上分类号码 i。这个工作很简单，因为只要把 class_list[i]中各元素中的列号码取出，并且在 class[ ]数组的对应脚码中存入 i 就行了。换句话说，class[j]=i 表示第 j 列是第 i 类。最后的工作就是把内存还给系统。

在主程序中有一组数据，不妨顺着这组数据追踪程序，这样对程序的运行方式是会有帮助的。

Download at http://www.pin5i.com/

【习题】

（1）把这个递归的程序改写成不用递归的程序。

（2）事实上这个程序的结果满足问题说明的最后一个条件，为什么？能看出来吗？试证明它。

（3）在 group()函数中，如果在 else 部分对自己的递归调用时也是用 column 而不是 column+1，请问会产生什么后果？

【参考文献】

笔者曾从 S.R.Kosaraju 教授那里看到这个问题，这里的做法是 Richard Biegle 想出来的，但同时也有其他人想到这个技巧，不过最干净利落的方法是 Simon Kasif 教授做出来的，他使用树状结构做题目。因为本书在题材上限定了不用数据结构，因而不打算讲解 Simon Kasif 教授的方法，但是可以自己想想，他的着眼点是把每一个向量合并成一个树状结构。S. R. Kosaraju 教授的问题是要找出一个比较次数最少的办法解决问题，可以很容易证明至少要有与 $n^2 \cdot m$ 成正比的比较次数，Richard Biegle 与 Simon Kasif 的方法都可以达到这个要求。

【程序】

程序 16-19（CLASSIFY.C）

```
/* ------------------------------------------------------------------ */
/* PROGRAM  classify :                                                */
/*   Given a set of m n-vectors organized into a m*n                  */
/* matrix, this program will classify these vector into               */
/* several class by assigning class numbers without                   */
/* sorting.  In other words, only comparisons of equal or             */
/* unequal are used in this program.                                  */
/*                                                                    */
/* Copyright Ching-Kuang Shene              Aug/01/1989               */
/* ------------------------------------------------------------------ */

#include  <stdio.h>

#define   MAXSIZE    20

/* ------------------------------------------------------------------ */
/*                  type definitions                                  */
/* ------------------------------------------------------------------ */

typedef   struct node   Node;     /* type of node                     */

struct node {                     /* a node consists of ...           */
    int   row;                /*     row #                            */
```

Download at http://www.pin5i.com/


```
    Node  *next;              /*     and a pointer to next        */
};
/* ----------------------------------------------------------      */
/*              external variables                                  */
/* ----------------------------------------------------------      */

int   class_no;               /* # of classes found                */
Node  **class_list;           /* table for class list              */

/* ----------------------------------------------------------      */
/*              function prototypes                                 */
/* ----------------------------------------------------------      */

void  classify(int [][MAXSIZE], int, int, int []);
void  group(Node *, int, int, int [][MAXSIZE]);
Node  *dispatch(Node *, int, int [][MAXSIZE]);

/* ----------------------------------------------------------      */
/* FUNCTION  classify :                                             */
/*   Given m n-vectors stored in matrix data[][], this              */
/* function returns the class number of each vector with            */
/* array class[].                                                   */
/* ----------------------------------------------------------      */

void  classify(int data[][MAXSIZE], int m, int n, int class[])
{
    Node *node_list, *tail, *p;
    int  i;

    /* -----------------initial class list ----------------      */

    class_list = (Node **) malloc(sizeof(Node *)*m);
    class_no   = 0;
    for (i = 0; i < m; i++)
        class_list[i] = (Node *) NULL;

    /*-----------------build up node list -----------------      */

    node_list = tail = (Node *) malloc(sizeof(Node));
    node_list->row   = 0;
    node_list->next  = NULL;
    for (i = 1; i < m; i++) {
        p = (Node *) malloc(sizeof(Node));
```

Download at http://www.pin5i.com/


```
            p->row  = i;
            p->next = NULL;
            tail = tail->next = p;
        }

        group(node_list, 0, n, data);   /* classify                    */

        for (i = 0; i < class_no; i++)   /* retrieve class #            */
            for (p = class_list[i]; p != NULL; p = p->next)
                class[p->row] = i;

        free(class_list);               /* release storage              */

        while (node_list != NULL) {
            p = node_list;
            node_list = node_list->next;
            free(p);
        }
    }

    /* ------------------------------------------------------------     */
    /* FUNCTION group :                                                 */
    /*   Given a list containing those vector code to be               */
    /* classified and their classification column, this                 */
    /* function classify all nodes in the list recursively.             */
    /* ------------------------------------------------------------     */

    void  group(Node *list, int column, int n, int data[][MAXSIZE])
    {
        Node  *other;

        if (column >= n)                /* any column available ?  */
            class_list[class_no++] = list;      /* NO, store ans    */
        else {                          /* YES, dispatch the list   */
            other = dispatch(list, column, data);
            group(list, column+1, n, data);     /* classify orig    */
            if (other != NULL)          /* if there is 'other' list*/
                group(other,column,n,data);     /* classify         */
        }
    }
```

Download at http://www.pin5i.com/


```
/* ------------------------------------------------------------ */
/* FUNCTION dispatch :                                          */
/*   Give a list of nodes, this function separates the          */
/* list into to lists, one containing all nodes with same       */
/* classification column and the other contains all others.     */
/* ------------------------------------------------------------ */

Node  *dispatch(Node *head, int column, int data[][MAXSIZE])
{
    int   value;
    Node  *first, *last, *p;

    value = data[head->row][column];     /* class. value          */
    first = last = NULL;
    p= head->next;                       /* the followup pointer */

    for(; p != NULL; p = head->next)     /* scan each node        */
        if(data[p->row][column]!= value){   /* same val          */
            head->next = p->next;        /* NO, extract it        */
            p->next    = NULL;           /* link to another       */
            if (first == NULL)
                first = last = p;
            else
                last = last->next = p;
        }
        else                             /* YES, scan next node */
            head = head->next;

    return first;
}
```

Download at http://www.pin5i.com/

第17章 游戏问题

问题 8.1 奇数阶魔方阵（MAGIC_O.C）

这个题目是填魔方阵（Magic Square）。所谓的 n 阶魔方阵，就是把 1～n^2 这 n^2 个连续的正整数填到一个 n×n 的方阵中，使得每一列的和、每一行的和以及两个对角线的和都相等。在这个题目中，只填奇数阶的魔方阵，因为它好填，而且也一定成功。如果不知道如何填法，可参看下面的说明。

【解答】

知道了填表的方法，解答其实就用不着说了，不过为了方便起见还是作个简单的说明。这个程序要写得简洁，恐怕用 C 语言才能办得到，不过也要注意到比较的顺序。

程序中用 count 来保存 1～order 的这些平方数，order 就是问题说明中的 n，即魔方阵的阶数。填完一个数之后，先看看是否要下移，回想一下，每填 order 个数之后就要下移，这很容易检查，看看 count 被 order 除后的余数就行了。如果不必下移，再看看它会不会从上方与右方越出边缘，并且适当地调整行与行的脚码 row 与 column。剩下的部分，就请读者研读程序了。

【习题】

（1）注意，上面说只有两种情况会越出边缘，但有没有可能现在的位置是在表格的右上角，下一格就同时越出上方与右方的边缘呢？试证明这种情况不会发生，至少应该找一些有力的证据说服自己相信这一点。

（2）其实，魔方阵中的数不一定是 1～n^2，任意连续的 n^2 个数都行（比如在填 5 阶魔方阵时，不用 1～25，而是用-10～14），有没有办法证明此方法还是可用的？请在程序中用最少的改变完成这一工作。

（3）请问，魔方阵中每一列、每一行，以及两条对角线的和是多少？请在程序中加上这一道输出，但不可以用求和的方式去加各列、各行或对角线。

（4）笔者在某本书中曾经见到过下面的 STUPID.C 程序，它也用此方法填魔方阵，检查这个程序，请问它的哪些地方写得不 好，或者多余，或者很没效率。现在读者已经知道填表的方法了，能改良它吗？相信一定是可以的。

程序 17-1（STUPID.C）

```
#include <stdio.h>
#include <stdlib.h>
```

Download at http://www.pin5i.com/


```
#define  MAXSIZE  20

void main(void)
{
     int matrix[MAXSIZE][MAXSIZE];       /* the magic square     */
     int count;                          /* 1...n*n counting     */
     int row;                            /* row index            */
     int column;                         /* column index         */
     int order;                          /* input order          */
     char    line[100];

     gets(line);
     order = atoi(line);

     for (row = 0; row < order; row++)
        for (column = 0; column < order; column++)
           matrix[row][column] = 0;

     row = 1;
     column  = order/2 - 1;
     for (count = 1; count <= order*order; count++) {
        row--;
        column++;
        if (row < 0 && column == order)
           row = 1, column = order - 1;
        else if (row < 0)
           row = order - 1;
        else if (column == order)
           column = 0;
        if (matrix[row][column] != 0)
           row += 2, column--;
        matrix[row][column] = count;
     }
     printf("\n\nMagic Square of order %d :\n\n", order);
     for (row = 0; row < order; row++) {
        for (column = 0; column < order; column++)
           printf("%4d", matrix[row][column]);
        printf("\n");
     }
}
```

（5）还有另一个填奇数阶魔方阵的方法是这样的：要填的值是 0～n^2-1，在填到 i 时，它的列与行的脚码 R 与 C 为：

Download at http://www.pin5i.com/

$$R=\left(a+c\bullet i+e\left\lfloor\frac{i}{n}\right\rfloor\right)\bmod n$$

$$C=\left(b+d\bullet i+f\left\lfloor\frac{i}{n}\right\rfloor\right)\bmod n$$

$\left\lfloor\frac{i}{n}\right\rfloor$表示比$\frac{i}{n}$小的最大整数，例如⌊3.14⌋3，⌊-3.14⌋=-4，mod 为余数运算，例如 5 mod 3 为 5 被 3 除的余数，亦即 2。当$a=b=n/2$，$c=d=1$，$e=-1$，$f=-2$时，就可以填出奇数阶魔方阵，但不保证对角线的和相同。这个方法叫做 Lehmer 法，是 1929 年由 D.N.Lehmer 提出来的，现在所讲的其实不过是个特例而已。请写一个用 Lehmer 法填魔方阵的程序，并且改变 a 与 b 的值（0≤a<n，0≤b<n=，看看能否把对角线的和也凑成相等。

（6）这个程序有一个被人批评的地方，就是它用了余数运算，请问有没有办法不用它呢？

提示：用两个 for 循环。

【参考文献】

3 阶的魔方阵在近四千年前中国科学家就知道了，甲骨文可以为证，而且河图正是一个 3 阶魔方阵的例子。据考证，很多中国所知的、填魔方阵的方法是在中世纪时由旅客、商人传入欧洲的。De la Loubere，是法皇路易十四在 1687—1688 年派驻暹逻（现在的泰国）的大使，他在暹逻学会了前面所讲的方法，所以就沿用他的名字，叫做 De la Loubere 法。补充一点，这个方法是 De la Loubere 在 1693 年发表的。

【程序】

程序 17-2（MAGIC_O.C）

```
/* ------------------------------------------------------------------ */
/* PROGRAM magic square of odd order :                                */
/*   This program will fills a matrix of order n*n with               */
/* integers of 1 to n*n such that the sums of rows,                   */
/* columns and two disgonals are all the same.  DON'T                 */
/* READ THIS PROGRAM before read my book.                             */
/*                                                                    */
/* Copyright Ching-Kuang Shene              July/03/1989              */
/* ------------------------------------------------------------------ */

#include <stdio.h>
#include <stdlib.h>

#define  MAXSIZE  20

void main(void)
{
     int matrix[MAXSIZE][MAXSIZE];   /* the magic square              */
```

Download at http://www.pin5i.com/


```
    int count;                          /* 1...n*n counting          */
    int row;                            /* row index                 */
    int column;                         /* column index              */
    int order;                          /* input order               */
    char    line[100];

    printf("\nOdd Order Magic Square Generator");
    printf("\n================================");
    printf("\n\nOrder Please --> ");
    gets(line);
    order = atoi(line);

    if (order > MAXSIZE)
       printf("\n*** ERROR ***  Order should be <= %d", MAXSIZE);
    else if (order % 2 == 0)
       printf("\n*** ERROR ***  Order must be an odd integer");
    else {
       row    = 0;                     /* start of from the middle*/
       column = order/2;               /* of the first row.       */
       for (count = 1; count <= order*order; count++) {
          matrix[row][column] = count; /* put next #              */
          if (count % order == 0)  /* move down ?                 */
             row++;                /* YES, move down one row      */
          else {                   /* compute next indices        */
             row    = (row == 0) ? order - 1 : row - 1;
             column = (column == order-1) ? 0 : column + 1;
          }
       }
       printf("\n\nMagic Square of order %d :\n\n", order);
       for (row = 0; row < order; row++) {
             for (column = 0; column < order; column++)
                printf("%4d", matrix[row][column]);
             printf("\n");
       }
    }
}
```

问题 8.2　单偶数阶魔方阵（MAGIC_SE.C）

会填奇数阶魔方阵之后，填 2(2m+1)阶的魔方阵也就不是难事了。请写一个程序填任何 2(2m+1)阶的魔方阵，如果不知道方法，可参看下面的说明。

Download at http://www.pin5i.com/

【解答】

注意 B、C、D 3 个部分与 A 部分的关系。如果 (i, j) 是 A 中的一个元素，而 A 有 n/2 列、n/2 行，于是 B、C、D 中的对应位置就是 (i + n/2, j + n/2)、(i, j + n/2) 与 (i + n/2, j)，因为 A 部分是以寻常的方式自 1 填到 $(n/2)^2$，而 B、C 与 D 则分别从 $(n/2)^2+1$、$2(n/2)^2+1$ 与 $3(n/2)^2+1$ 开始，所以当 A 中 (i, j) 被填了 c 这个值时，B、C 与 D 的对应位置的值就是 $c+(n/2)^2$、$c+2(n/2)^2$ 与 $c+3(n/2)^2$。为什么可以这样做呢？道理很简单，请证明它。程序中函数 magic_o()就是这样来的，它不过是修改了奇数阶魔方阵程序，可以同时填 4 个部分而已。

互换各个部分的程序也不难。A 部分的中间那一列在阶数是 2(2m+1)时，正好就是第 m=2(2m+1)/4 列，所以 width 就存了 n/4 这个值。对于不是第 width 列的第 i 列而言，前 width 元素就是 x[i][j]，j 是 0～width-1 之间的值。这 width 个数据要与 D 的对应部分，也就是 x[n/2+i]互换，这是 if…then 中的第一个 for 循环。第二个 for 循环则用来互换在一列最后面的 width-1 个元素。else 的部分是用来处理 A 部分的中间列的，它把中间那一格起向左的 m 格与 D 的对应部分互换。因为 n=2(2m+1)，而 A 有 2m+1 行，中间那一行自然就是 m，亦即 width，所以这连续的 m 个元素就正好是第 n/4 列上的第一、第二到第 n/4=width 个元素。程序中函数 exchange()就是负责完成这项操作的。

主控程序在 magic_se()中，它只不过是让 magic_o()去排 A、B、C 与 D 四个部分，再调用 exchange()做互换工作而已，十分简单。

【习题】

（1）在 exchange()中的 if 写得很琐碎，把它改写，让其更有效率并且更加简洁。

（2）试证明任何一个魔方阵的每一个元素在都加上一个常数值（正负均可）之后，还是一个魔方阵。在解答中填 B、C、D 三个部分的做法也就可以得到 3 个魔方阵的结果了。

（3）经过 magic_o()处理过后，试证明 2(2m+1)方阵中各行的和是相同的。但是列与对角线是否相同？试证明经过 exchange()处理后就是一个真正的魔方阵。

【参考文献】

单偶数阶魔方阵是仅次于奇数阶的简单题目，本来这个题目应该列作奇数阶魔方阵的习题，但是因为在程序设计上如何处理脚码的技巧值得一提，所以还是把它单独列了出来。在此处的填法是 Ralph Strachey 在 1918 年 8 月写给 W.W.Rouse Ball 的一封信中提出来的，笔者从 W.W.Rouse Ball 的著作中学到这个方法。

W.W.Rouse Ball and H.S.M.Coxeter. *Mathematical Recreations and Essays*. 13th ed., Dover, 1987

【程序】

程序 17-3（MAGIC_SE.C）

```
/* ------------------------------------------------------------------ */
/* PROGRAM  singly-even magic square :                                */
/*   Given an integer of form 2*(2k+1), this program                  */
```

Download at http://www.pin5i.com/


```
/* builds up a magic square with numbers from 1 to n*n.          */
/*                                                               */
/* Copyright Ching-Kuang Shene          July/23/1989             */
/* ---------------------------------------------                 */

#define   MAXSIZE   30

void  singly_even(int [][MAXSIZE], int);
void  magic_o(int [][MAXSIZE], int);
void  exchange(int [][MAXSIZE], int);
/* ---------------------------------------------                 */
/* FUNCTION  singly_even :                                       */
/*   This is the driver program.  It fills the upper-left        */
/* lower-right, upper-right and lower-left parts by using        */
/* odd order magic square routine.  Then exchange some           */
/* parts in each routine in order to maintain the magic          */
/* properties.                                                   */
/* ---------------------------------------------                 */

void  singly_even(int matrix[][MAXSIZE], int n)
{
     int  half = n/2;

     magic_o(matrix, half);
     exchange(matrix, n);
}


/* ---------------------------------------------                 */
/* FUNCTION  magic_o :                                           */
/*   Odd order magic square routine.  It fills the block         */
/* bounded by left, right, top and bottom with numbers           */
/* starting from 'start'.  Otherwise all are the same as         */
/* the magic square routine discussed before.                    */
/* ---------------------------------------------                 */

void  magic_o(int matrix[][MAXSIZE], int n)
{
 int count;              /* fill counting                        */
 int row;                /* row index                            */
 int column;             /* column index                         */
```

Download at http://www.pin5i.com/


```
        row = 0;                    /* start of from the middle     */
        column  = n/2;              /* of the first row.            */
        for (count = 1; count <= n*n; count++) {
            matrix[row][column] = count;    /* put # in A           */
        matrix[row+n][column+n] = count + n*n; /* in B              */

            matrix[row][column+n] = count + 2*n*n;  /* in C         */
            matrix[row+n][column] = count + 3*n*n;  /* in D         */
            if (count % n == 0)     /* move downward ?              */
                row++;              /* YES, move down one row       */
            else {                  /* compute next indices         */
                row  = (row == 0) ? n - 1 : row - 1;
                column   = (column == n-1) ? 0 : column + 1;
            }
        }
    }

    /* ----------------------------------------------------------   */
    /* FUNCTION  exchange :                                         */
    /*   This function exchanges some parts in each row in          */
    /* order to maintain the magic square properties.               */
    /* ----------------------------------------------------------   */

    #define  SWAP(x,y)   { int t; t = x; x = y; y = t;}

    void  exchange(int x[][MAXSIZE], int n)
    {
        int width   = n / 4;
        int width1  = width - 1;
        int i, j;

        for (i = 0; i < n/2; i++)
            if (i != width) {       /* if not the width-row         */
                for (j = 0; j < width; j++)
                    SWAP(x[i][j], x[n/2+i][j]);
                for (j = 0; j < width1; j++)
                    SWAP(x[i][n-1-j], x[n/2+i][n-1-j]);
            }
            else {                  /* width-row is special         */
                for (j = 1; j <= width; j++)
                    SWAP(x[width][j], x[n/2+width][j]);
                for (j = 0; j < width1; j++)
```

Download at http://www.pin5i.com/


```
                SWAP(x[width][n-1-j], x[n/2+width][n-1-j]);
            }
        }
```

问题 8.3 双偶数阶魔方阵（MAGIC_DE.C）

偶数阶的魔方阵是个难题，但阶数是 4 的倍数（4m）时却很简单，略比奇数阶的复杂些。请写一个程序来填 4m 阶（m>0）的魔方阵。如果不知道这个方法，可参看下面的说明。

【解答】

这个程序写得是否漂亮，就在于如何做记号。注意看问题说明的做记号方式，同一列或同一行上相邻的元素都不会有记号，因此在处理一列时，从头开始不外乎是有、没有、有、没有…，或者是没有、有、没有、有…这个状态，能有这样正反出现的，最简单的莫过于 1,-1,1,-1,…或-1,1,-1,1,…了。所以如果把记号用 marker 来表示，而且它已经有了 1 或-1 的值，则每处理完一个值，就把 marker 变号，亦即-marker 就行了。不过，另一个问题是，阶数是 4m，A 部分一列上有 2m（偶数）个元素，如果每一列开始时 marker 的值都相同，那么所做的记号每一列上都一样就不会错开了，因此一列处理完后，marker 还要多变号一次才行。

另外，没有必要先把 A 填好，以对称方式抄到 B，再把 A 与 B 以对称方式抄到 C。因为在为 A 做记号时就可以同时把 B 的记号做出来，为什么？当 A 的(i,j)这个元素得到 marker 这个值时，(i,j)以 A 与 C 中央线的对称元素自然是(i,n-1-j)，所以它们可以得到相同的记号。如果仔细研究一下问题说明中填数字的说明，就会发现 D 与 B 的记号根本没有用，因为它们的值是依 A 与 C 部分是否有记号而来的，所以根本就不必理会 D 与 B 这两部分。

程序 MAGIC_DE.C 中最外层的 else 里面的两个 for 就是做记号用的，内层的 for 处理 A 的一列，并且同时也把记号放到 C 中对称的位置。因为 for 循环每绕一次 marker=-marker 就执行一次，所以到结束时 marker 的值与进入 for 时一样（一共变了偶数次）。故在内层 for 循环做完，回到外层 for 循环底部时，外层 for 循环就把 marker 再变号一次，这就是第一段末的技巧。做完这两个 for 循环之后，方阵中的值不外乎是 1 或-1，将-1 标有记号（但是，1 标有记号又有何不可？），接下来的两个嵌套的 for 就是填值用的。

在填值时，一次是填两个格子。如果某格没有记号，则那一格是填下一个递增的值，但以中心点与它对称的则填的是自 n^2 起递降的值。若某格有记号，则正好相反，所以用两个计数用的变量来做这件事。count 从 1 递增，inv_count 自 n^2 递降，程序中包含在两个 for 循环中的 if 就在于检查 square[i][j]有没有记号，再使用合适的填值程序。程序的其他部分很简单，所以就不作进一步的说明了。

【习题】

（1）在程序中一共有两组嵌套式的 for 循环，一组用来做记号，一组用来填值，但如

Download at http://www.pin5i.com/

果用算术式计算，并且仔细些，就会发现用一组就足够了。能够想出什么方法来达成目的吗？程序的效率是否会比较高？

（2）做记号的方式是很规则的，请问有没有其他做记号的方法，试把它写成程序。

（3）因为方阵的阶数是 4m，所以整个方阵是由 m^2 个 4×4 的方阵构成的。在一个 4×4 方阵中，两条对角线如果做了记号，对 4×4 方阵来说是合乎规定的。然后用 m^2 个这样的 4×4 方阵就可以组成一个 4m 阶方阵。试证明：由此而来的做记号方式合乎规定，再用这个观点写成程序。

【参考文献】

这个问题也不是难题，重点是在一个方阵中脚码的运用方式而已。笔者是在 W.W.Rouse Ball 的著作中学到这个方法，不过做记号的方式则是为了程序写作的方便而改成程序中的形式。W.W.Rouse Ball 书中的方法，据他说是由 Kraitchik 提出来的，这就是习题（3）中的方法。

W.W.Rouse Ball and H.S.M.Coxeter. *Mathematical Recreations and Essays*. 13th ed., Dover, 1987

【程序】

程序 17-4（MAGIC_DE.C）

```
/* ------------------------------------------------------------------ */
/* PROGRAM  doubly even magic builder :                               */
/*   Given an integer n = 4*m, m > 0, this program builds             */
/* up a magic square of order n.                                      */
/*                                                                    */
/* Copyright Ching-Kuang Shene          July/23/1989                  */
/* ------------------------------------------------------------------ */

#include <stdio.h>
#include <stdlib.h>
#define  MAXSIZE     20          /* square size                       */
#define  MARK        -1          /* marker for build up               */

void  main(void)
{
     int square[MAXSIZE][MAXSIZE];  /* the square                     */
     int n;                         /* order of the magic square      */
     int count, inv_count;          /* 1 -> n^2 and n^2 -> 1          */
     int marker;                    /* working marker 1,-1,1,-1       */
     int i, j;
     char    line[100];

     printf("\nDoubly-Even Magic Square");
     printf("\n========================");
```

Download at http://www.pin5i.com/


```
    printf("\n\nOrder (4*m, m>0) please --> ");
    gets(line);
    n = atoi(line);
    if (n % 4 != 0)
       printf("\n*** Illegal Order *****");
    else {
       marker = MARK;       /* mark the upper part                    */
       for (i = 0; i < n/2; i++, marker = -marker)
          for (j = 0; j < n/2; j++, marker = -marker)
             square[i][j] = square[i][n-1-j] = marker;

       count     = 1;       /* upward counter                         */
       inv_count = n*n;     /* downward counter                       */
       for (i = 0; i < n/2; i++)
          for (j = 0; j < n; j++)
             if (square[i][j] != MARK) {  /* marked                   */
                square[i][j] = count++;
                square[n-1-i][n-1-j] = inv_count--;
             }
             else {         /* unmarked                               */
                square[i][j] = inv_count--;
                square[n-1-i][n-1-j] = count++;
             }
       printf("\n");
       for (i = 0; i < n; i++) {
          for (j = 0; j < n; j++)
             printf("%4d", square[i][j]);
          printf("\n");
       }
    }
}
```

问题 8.4　N 后问题公式解（N_QUEEN1.C）

8 后问题（Eight Queen Problem）是指在一个 8×8 的西洋棋盘上要如何放置 8 个皇后棋而不会互相吃到对方。皇后棋可以吃掉任何它所在的那一列、那一行，以及那两条对角线（米字型）上的任何棋子。信不信有一道公式对于每一个 n>3 而言，在 n×n 棋盘上都可以找出（一个）N 后问题的解？请依下面的说明写一个程序，读入 n，把一组解显示出来。

【解答】

程序本身并没有什么奇特的地方，惟一要提醒的就是一个存储了指向函数的指针的数

Download at http://www.pin5i.com/

组。一共有 4 个函数，即 class_1()、class_2()、class_3()与 class_4()，分别对应着 4 个类型，所以在主程序中就有一个数组 funct[ ]，它存储的是指向这 4 个函数的指针。

在第一种情况中，n 是 6k 或 6k-2，6k 表示用 6 可以整除，6k-2 相当于用 6 去除余-2，亦即余 4，因此 funct[0]与 funct[4]中的函数就是 class_1()。对第二类而言，6k+1 是用 6 除余 1，6k-1 是用 6 除余-1，这也相当于余 5，所以 funct[1]与 funct[5]的内容就是 class_2()的地址。第三类很简单，6k+3 就是用 6 除后余 3，因此 funct[3]中的地址就是 class_3()。第四与第五类可以合并，6k+8 与 8 都是用 6 除后余 2，前者 n>8，后者则正好是 8，所以 funct[2]中是 class_4()，而在 class_4()中去分辨 n 是否正好是 8。

主程序中在读入了 n 之后，先看 n 是否大于 3（小于等于 3 无解），如果是，就依 n 被 6 除的余数在 funct[ ]中取出适合的函数来执行。其他各个函数都是非常简单的，读者不妨自己研究。

【习题】

（1）证明 1≤n≤3 时，N 后问题无解。

（2）证明：上述 5 种情况所对应的皇后位置，都是正确的解。

提示：算出每一对坐标的和与差，由此而推出彼此互不侵犯。

（3）将和式对 n=4 以上都执行一次，并且输出答案，请问这些解是否有规则性？

【参考文献】

本题的解法源自 Matthias Reichling 在 1987 年的一篇文章，他的观点其实是早一年由 Bernd-Jurgen Falkowski 与 Lother Schmitz 两人提出的方法的化简，其实后面两位才是这个方法的原创人。参看如下书籍：

1 B.-J.Falkowski and L.Schmitz. A Note on The Queens'Problem, *Information Processing Letters*. 1986(23), pp.39～46

2 M.Reichling. A Simplified Solution of the N Queens'Problem, *Information Processing Letters*. 1987(25), pp.253～255

3 B.Abramson and M.Yung. Divide and Conquer under Global Constraints: A Solution to the N-Queens Problem, *Journal of Parallel and Distributed Computing*. 1989(6), pp.649～662

前两篇文章都不难，对于喜欢这一类题目的朋友，建议依上列顺序都读一次。笔者在第一次看到这样的解法时，心中实在是很兴奋的，在图书馆书架前面站着把它读完，因为这是第一次看到一个 N 后问题的公式解，希望读者的感觉与我相同。另外，下面这本书中也有一个很好的说明。

4 A.M.Yaglom and I.M.Yaglom. Challenging Mathematical Problems with Elementary Solutions. Vol.I: Combinatorial Analysis and Probability Theory, Dover. 1987

【程序】

程序 17-5（N_QUEEN1.C）

```
/* ------------------------------------------------------------------ */
/* FUNCTION one solution for n queens' problem :                      */
```

Download at http://www.pin5i.com/


```
/*   Given a board size n > 3, this program computes one        */
/* solution on the N Queens' Problem by using a formula         */
/* of legal positions.  This method is interesting because      */
/* there actuall are some regularities in the N Queens'         */
/* problem which confused C.F.Gauss once upon a time.           */
/*                                                              */
/* Copyright Ching-Kuang Shene          July/17/1989            */
/* ------------------------------------------------------------ */

#include <stdio.h>
#include <stdlib.h>

#define  MAXSIZE     30
#define  MARK        'Q'

void initial(void);
void display(void);
void class_1(void);
void class_2(void);
void class_3(void);
void class_4(void);

int  n;                              /* input board size        */
char board[MAXSIZE+1][MAXSIZE+1];    /* the chess board         */

void main(void)
{
     void (*funct[])() = { class_1, class_2,    /* functions    */
                           class_4, class_3,    /* of four      */
                           class_1, class_2     /* classes      */
                          };
     char line[100];

     printf("\nOne Solution for N Queens' Problem");
     printf("\n==================================");
     printf("\n\nBoard Size Please (N > 3) --> ");
     gets(line);
     n = atoi(line);

     if (n > 3) {
        initial();           /* initial the chess board          */
        (*funct[n % 6])();   /* call appropriate funct.          */
```

Download at http://www.pin5i.com/


```
            display();          /* print result                    */
        }
        else
            printf("\nIllegal Board Size.");
    }


    /* ----------------------------------------------------------- */
    /* FUNCTION initial :                                          */
    /*   Initialize the chess board to blank.                      */
    /* ----------------------------------------------------------- */

    void initial(void)
    {
        int  i, j;

        for (i = 1; i <= n; i++)
            for (j = 1; j <= n; j++)
                board[i][j] = ' ';
    }

    /* ----------------------------------------------------------- */
    /* FUNCTION display :                                          */
    /*   Function to display the chess board.                      */
    /* ----------------------------------------------------------- */

    #define  DRAWGRID(N) {   int i;                                 \
                             printf("\n+");                         \
                             for (i = 1; i <= N; i++)               \
                                 printf("-+");                      \
                         }

    #define  DRAWLINE(N, r)  {   int i;                             \
                                 printf("\n|");                     \
                                 for (i = 1; i <= N; i++)           \
                                     printf("%c|", board[r][i]);    \
                             }

    void display(void)
    {
        int  r;
```

Download at http://www.pin5i.com/


```
        DRAWGRID(n);
        for (r = 1; r <= n; r++) {
            DRAWLINE(n, r);
            DRAWGRID(n);
        }
    }


    /* ----------------------------------------------------------- */
    /* FUNCTION class_1 :                                          */
    /*   Solution for n mod 6 == 0 or 4.                           */
    /* ----------------------------------------------------------- */

    void  class_1(void)
    {
        int  i;

        for (i = 1; i <= n/2; i++)
            board[2*i][i] = board[2*i-1][n/2+i] = MARK;
    }


    /* ----------------------------------------------------------- */
    /* FUNCTION class_2 :                                          */
    /*   Solution for n mod 6 == 1 or 5.                           */
    /* ----------------------------------------------------------- */

    void  class_2(void)
    {
        int  i;

        for (i = 1; i <= (n-1)/2; i++)
            board[2*i][i] = MARK;
        for (i = 1; i <= (n+1)/2; i++)
            board[2*i-1][(n-1)/2+i] = MARK;
    }


    /* ----------------------------------------------------------- */
    /* FUNCTION class_3 :                                          */
```

Download at http://www.pin5i.com/


```
/*   Solution for n mod 6 == 3.                                    */
/* ------------------------------------------------------------    */

void  class_3(void)
{
     int  i;

     for (i = 1; i <= (n-3)/2; i++)
         board[2*i+2][i] = board[2*i+3][(n-1)/2+i] = MARK;
     board[1][n-1] = board[2][(n-2)/2] = board[3][n] = MARK;
}

/* ------------------------------------------------------------    */
/* FUNCTION class_4 :                                              */
/*   Solution for n mod 6 = 2.                                     */
/* ------------------------------------------------------------    */

void  class_4(void)
{
     int  i;

     if (n > 8) {
         for (i = 1; i <= 3; i++)
             board[2*i-1][n/2-2+i] = MARK;
         board[2][n] = board[4][1] = board[6][n-1] = MARK;
         for (i = 1; i <= n/2 - 3; i++)
             board[2*i+5][i+1] = board[2*i+6][n/2+1+i] = MARK;
     }
     else {
         for (i = 1; i <= 4; i++)
             board[2*i][i] = MARK;
         board[1][6] = board[3][5] = MARK;
         board[5][8] = board[7][7] = MARK;
     }
}
```

问题 8.5　N 后问题递归解（N_QUEENR.C）

请写一个程序，读入一个值 n 表示棋盘的大小，然后求出 n×n 格棋盘上放 n 个皇后棋而不会相互吃掉对方的所有解答。

Download at http://www.pin5i.com/

【解答】

这个题目有一个关键，这就是判定某个位置可不可以放一个皇后棋，只有在该位置同一列、同一行以及两个对角线上都没有其他皇后棋的时候，才可以把皇后棋放在那，因为每一行上都只能有一个皇后棋，所以是用不着准备一个n×n的棋盘的，用一个有n个元素的数组就足够了，用pos[ ]来存放各个棋子的所在位置，换言之，pos[j]=i就表示第j行上的棋子是在第i列，这是个很精简的表示法。用pos[ ]数组可以肯定每一行上只会有一个棋子，所以就用不着去检查一行上是否有两个棋子了。如何检查一列上是否有两个棋子呢？用同样的方法，可以准备一个数组in_row[ ]，如果in_row[i]是OCCUPIED，那就表示第i列上已经有了棋子。

两个对角线是麻烦一些，图17-1所示是一个5×5的棋盘，标出了它的各条反对角线，并且注明了号码。

注意，在最长的对角线与它的上方，在n×n的棋盘之下有n条反对角线，但在最长的对角线下方还有n-1条，所以反对角线就总共有2n-1条。如果仔细检查一下，第i条反对角线上的元素(r,c)都满足r+c-1的关系。因此，可以准备一个back_diag[ ]的数组，如果要看看棋子能否放在第r列第c行，第一件事是查in_row[r]，看它是否是OCCUPIED，如果不是，就查back_diag [r+c-1]，如果不是OCCUPIED，则3项检定就通过两项了。

最后是n×n棋盘的主对角线了，图17-2所示是一个5×5棋盘的所有主对角线，参照反对角线的说明，主对角线也应该有2n-1条。

1	2	3	4	5
2	3	4	5	6
3	4	5	6	7
4	5	6	7	8
5	6	7	8	9

图 17-1

5	4	3	2	1
6	5	4	3	2
7	6	5	4	3
8	7	6	5	4
9	8	7	6	5

图 17-2

也可以很容易验证，第r列第c行的元素在第n-(c-r)条主对角线上。因此，用diag[ ]这个有2n-1个元素的数组来记录哪一条主对角线已经被占据了。如果要在第r列第c行上放一个皇后棋，那么

```
in_row[r]
diag[n-(c-r)]
back_diag[r+c-1]
```

三者都要是EMPTY(空的)才行。在下面的程序中，这一项测试写成一个叫做SAVE(r,c)的集体指令。

当知道何处可以放棋子之后，整件事就容易多了。程序N_QUEENR.C的关键所在就是函数try()，这个函数是参照问题说明中的架构写成的。try()函数接收一个值column，指出让它在第column行找出一个位置来放棋子，在函数中它就令row自1变到n，把那一行上各列的格子都试一次。在查到第row列、第column行的时候，就用上述的SAVE(r,c)集

Download at http://www.pin5i.com/

体指令看看那个位置是不是安全，如果是安全的，就用 SET(r,c)集体指令在该位置上放一个棋子，并且在右边如果还有空行的话，调用自己尝试在下一行放个棋子。若是右边已经没有空行了，这就表示所有 n 行都已经有了合适的棋子，则把结果显示出来。但不论如何，当第 row 列、第 column 行试过之后（不管成功与否），由于是要找出所有解答，因此就去试 column 行上的下一列，在做之前，要把方才用 SET(r,c)放下去的棋子取消才行，这就是 RESET(r,c)集体指令的工作。

SET(r,c)是把棋子放在第 r 列第 c 行，要做 4 件事。第一件是把 r 存入 pos[c]，其他 3 件是把第 r 列第 c 行所在的列、正与反对角线都定成 OCCUPIED。反过来，RESET(r,c)就是取消 SET(r,c)所做的动作，但是却不曾对 pos[ ]操作，为什么可以如此？读者不妨自己想一想。

程序的其他部分很容易，读入棋盘大小之后，就用 initial()函数给定各数组初值，用 try()找出解答，而 try()又会用 display()显示结果。

不过有几点要提醒：第一，显示出来的结果是 pos[ ]的内容，而不是棋盘，但要改成棋盘倒也不难，做法可参考上一题；第二，为了与说明的部分相配合，全都不用第 0 个元素，所有脚码都从 1 起；第三，因为参与递归的只有 column 这个参数，所以所有数组都是外在变量。

【习题】

（1）有没有办法把这个程序改成不用递归的程序？（这是很简单的，在 try()中除了参数 column 之外就只有一个内在变量 row，用一个堆栈保存它们就行了，相信不会是难事）

（2）西洋棋盘中武士的走法是 L 形的，请设计一个程序，找出棋盘上要放多少个武士棋而不会造成冲突的所有解答。

（3）请写一个程序，能够涵盖整个棋盘的最少皇后棋的数目（8×8 时，5 个皇后棋就够了）。

（4）8 后问题一共有 92 个解，但是因为棋盘有对称的关系（比如旋转 90°、180°、720°；左、右或上、下；沿正、反对角线的翻转等），最终可以归纳出 12 组彼此不能通过对称关系而得来的解答，这些就叫做基本解（Fundamental Solution）。请在 92 组解中找出那 12 组基本解；如果读者学过离散数学或对称群论，了解 Polya 方法，有没有办法写个程序只产生这 12 组基本解呢？

【参考文献】

这个问题之所以风行，是由于 E.W.Dijkstra 与 N.Wirth 两人最先介绍的，他们分别在 1970 年与 1971 年利用这个问题来讲解程序设计的技巧，之后可以在其他的文章与书籍中找到越来越好的写法。本书中的解法是以 N.Wirth 的方式作蓝本写成的。

1 E.W.Dijkstra. *Notes on Structured Programming*, Technical University Eind-hoven. 1976

2 N.Wirth. *Systematic Programming-An Introduction*, Prentice-Hall. 1973

3 N.Wirth. *Algorithm and Data Structure, Prentice-Hall.* 1986

Dijkstra 的讲义后来被 O.-J.Dahl 等人收集到他们编著的一本书中。

4 O. -J. Dahl, E. W. Dijkstra and C. A. R. Hoare. *Structured Programming*, Academic Press. 1972

Download at http://www.pin5i.com/

8 后问题的对称解在很多数学游戏的书籍中都一定会讨论到，可参看下面两本知名之作。

5 W.W.Rouse Ball and H.S.M.Coxeter. *Mathematical Recreations and Essays*. 13th ed., Dover, 1987

6 M.Kraitchik. *Mathematical Recreations*. 2nd ed., Dover, 1953

虽然很早就有人对 8 后问题的对称性进行讨论，但在计算机文献方面并不多。以下是笔者找到的两篇，Peter Naur 在 1972 年讲解了对称的问题，但并没有完整的程序，不过 Naur（BNF 中，N 代表 Naur）的文章很值得一读。1982 年 Rodney W.Topor 的文章则是有完整 Pascal 程序的文献，不过因为用到了群论的知识，这就不是每个人都能很快读懂的了。

7 P.Naur. An Experiment on Program Development, *BIT*. 1972(12), pp.347～365

8 R.W.Topor. Fundamental Solutions of t he Eight Queens Problem, *BIT*. 1982(22), pp.42～52

【程序】

程序 17-6（N_QUEENR.C）

```
/* ------------------------------------------------------------ */
/* PROGRAM all possible solutions of n queens' problem :        */
/*   Given an input n, this program computes all possible       */
/* solutions of the N Queens' Problem by using recursive        */
/* method.                                                      */
/*                                                              */
/* Copyright Ching-Kuang Shene              July/17/1989        */
/* ------------------------------------------------------------ */

#include <stdio.h>
#include  <stdlib.h>

#define  MAXSIZE     20          /* max. board size            */
#define  DIAG_SIZE   9           /* max. # of diagonals        */
#define  OCCUPIED    0
#define  EMPTY       1


void initial(void);
void display(void);
void try(int);

int  pos[MAXSIZE+1];             /* pos[j]=i means Q is (i,j)  */
int  in_row[MAXSIZE+1];          /* in_row[i] true->row occ.   */
int  diag[DIAG_SIZE+1];          /* diag[i] or back_diag[i]    */
int  back_diag[DIAG_SIZE+1];     /* = true -> occupied.        */
int      n;                      /* input board size           */
```

Download at http://www.pin5i.com/


```
int  count = 0;                    /* # of solutions counter       */

void main(void)
{
     int i, j;
     char    line[100];

     printf("\nAll Possible Solutions of N Queens' Problem");
     printf("\n=========================================");
     printf("\n\nBoard Size (N > 3) --> ");
     gets(line);
     n = atoi(line);

     initial();
     try(1);                       /* starting from column 1       */
     printf("\n\nThere are %d solutions in total.", count);
}


/* ----------------------------------------------------          */
/* FUNCTION initial :                                            */
/*   Function to initial the chess board.                        */
/* ----------------------------------------------------          */

void  initial(void)
{
     int  i;

     for (i = 1; i <= n; i++)
        in_row[i] = EMPTY;
     for (i = 1; i <= 2*n - 1; i++)
        diag[i] = back_diag[i] = EMPTY;

     printf("\nSolutions are represented the row # in a column");
     printf("\n\n  #");
     for (i = 1; i <= n; i++)
        printf("%3d", i);
     printf("\n---");
     for (i = 1; i <= n; i++)
        printf(" --");
}
```

Download at http://www.pin5i.com/


```
/* ----------------------------------------------------------------- */
/* FUNCTION display :                                                */
/*   Function to display a single solution.                          */
/* ----------------------------------------------------------------- */

void  display(void)
{
     int  i;

     printf("\n%3d", count);
     for (i = 1; i <= n; i++)
        printf("%3d", pos[i]);
}


/* ----------------------------------------------------------------- */
/* FUNCTION try :                                                    */
/*   Given a column number, this function tries to put               */
/* a queen on some row of that column until all possible             */
/* rows are attampted.                                               */
/* ----------------------------------------------------------------- */

#define  SAVE(r,c)    (in_row[r] && back_diag[r+c-1] && diag[n-(c-r)])
#define  SET(r,c)     {   pos[c]  = r;                                \
                          in_row[r]    = OCCUPIED;                   \
                          back_diag[r+c-1]    = OCCUPIED;            \
                          diag[n-(c-r)]   = OCCUPIED;                \
                      }
#define  RESET(r,c)   {   in_row[r]       = EMPTY;                   \
                          back_diag[r+c-1]    = EMPTY;               \
                          diag[n-(c-r)]   = EMPTY;                   \
                      }

void  try(int column)
{
     int  row;

     for (row = 1; row <= n; row++) /* for each row                 */
        if (SAVE(row, column)) {    /* is it save ?                 */
           SET(row, column);        /* YES, make a record           */
```

Download at http://www.pin5i.com/

```
            if (column < n)       /* all columns tried?          */
                try(column+1);    /* NO, try next col.           */
            else {
                count++;          /* OR, we found a sol          */
                display();        /* display it.                 */
            }
            RESET(row, column);   /* restore and back.           */
        }
    }
```

问题 8.6　武士巡逻（KNIGHT.C）

在西洋棋中的武士（Knight）与象棋的马相似，走的是 L 形的路，请写一个非递归的程序，输入一个表示棋盘的大小值 n，再输入一个起点的坐标，找出一条从那一个起点，可以让武士棋走完 n^2 格而不重复的路径。请问，找出的路径会回到起点吗？什么时候一定做不到？

【解答】

首先看如何表示下一步可以走到的 8 个可能位置。如果目前位置的坐标定成 0，那么 8 个位置就可以用与目前位置的差距表示。为了方便起见，上方与右方是正，这 8 个点的坐标以逆时针方向表示，就是(2,1)、(1,2)、(-1,2)、(-2,1)、(-2,-1)、(-1,-2)、(1,-2)与(2,-1)，两个数的第一个是横坐标，第二个是纵坐标。若 x 与 y 是目前点的位置，而希望以上面顺序算来的第三个方向为(-2,1)，于是下一个点的坐标就是(x-2,y+1)。在程序中，用 offset_x[ ]与 offset_y[ ]存放这 8 个方向的差距。要用一个棋盘记录下武士棋的行径顺序（0～n^2-1），故用一个二维数组 board[ ][ ]来做棋盘。为了方便起见，n×n 的棋盘不用第 0 列第 0 行，而得用 n+1 列 n+1 行才够。

接下来介绍解题的构想。首先，在棋盘的各个格子都定成 EMPTY。因为一共有 n^2 个格子，所以依着武士棋第一步、第二步、第三步、…、第 n^2 步的顺序，在对应的格子中填入 0,1,2,…, n^2-1，因此在最后输出时，跟着这些数值就可以顺序走一次了。在走棋盘时会一再地尝试，并且做回溯的工作，由此用一个堆栈来保存到目前为止所经过的各个格的坐标，以及从这一格到下一格所使用的方向。这个堆栈实际上是 3 个数组，path_x[ ]与 path_y[ ]存放格子的坐标，direction[ ]存放使用的方向，top 指出堆栈最顶上元素何在，在开始时它的值是-1。

因为有一个起点，所以在进行尝试与回溯的操作之前，把开始的点的坐标与使用方向（有 0～7 表示上述的 8 个方向）放到堆栈中，并且在棋盘中对应位置存入一个 0，表示第一步，接着就开始正式运行了。

回溯的操作很简单，因为堆栈顶端的点与它的方向是最近所到达的位置。于是，就沿

Download at http://www.pin5i.com/

着该方向找出下一个点的位置：

```
new_x = path_x[top] + offset_x[top];
new_y = path_y[top] + offset_y[top];
```

接着看这个新位置是不是在棋盘上：

```
(1 <= new_x) && (new_x <= n) && (1 <= new_y) && (new_y <= n)
```

而且还要看那个位置是不是空的：

```
board[new_x][new_y] == EMPTY
```

依着这两个测试，有下面的两种情况。

（1）如果上述两个条件都是真，就到达了一个新位置，这个新位置在棋盘上，而且也是空的，当然就可以走到那去。接着要做的事就是：① 把这个新位置的坐标与它的第一个方向存到堆栈中，下一个回合，它就变成目前的位置了；② 堆栈顶端的指针 top 的值，正好是走到第几步的数目，目前为止所经过的格子都存放在堆栈中，因此把 top 的值存入棋盘的对应位置中，表示已经到过这个点。接着，再回头处理新进入堆栈的位置。

（2）如果有一个条件为假，就表示这个新位置不能使用，要试下一个方向：

```
direction[top]++ ;
```

但若 8 个方向都已经试过了，所能到达的 8 个位置都不能用，目前的这个位置一定也不对，所以就回溯到上一个位置。回溯操作很简单，把 top 减去 1，top 就指向上一个位置了。于是再回头处理新跳到堆栈顶端的位置。

然而，当 8 个方向都试过了而找不出下一个位置时不能冒失地把 top 减去 1，为什么呢？如果 top 现在是 0，再减 1 就“出界”了；所以要有进一步的操作。当然，如果找到了下一个位置，这个操作就是多余的。因此在找不出下一个可用的位置时，先查一查 top 值是否大于 0，如果是，不但要把 top 减去 1，回到上一个点，还要把上一个点的方向移到下一个去，并且把目前这个位置定成 EMPTY，好让以后操作可以走到这：

```
board[path_x[top]][path_y[top]] = EMPTY;
direction[--top]++;
```

这个操作一再反复，直到堆栈中放满了 n^2 个点为止，即表示所有的位置都走完，可以结束工作，显示结果了。

由于当 n 很大时，一试再试的操作是很多的，因此这个程序非常耗时。一般而言，在普通 AT(Intel 286)型机器之下，n=7 或 n=8 都至少要执行半小时到一小时，因此自己测试时要先有心理准备。如图 17-3 所示，是 n=5,6,7,8 所做出来的结果。

从上面的结果会发现，没有一个能够回到起点的。当然，可以修改程序来完成这件事，也就是说，看第 n^2 步能否与第一步相同，如果不同，那就回溯，而不是显示结果。但是，当 n 是奇数时，可以证明这是件不可能的事。纵使 n 是偶数，程序也会耗费更长的时间，因为单纯的回溯技巧并不足以解决这个问题。

Download at http://www.pin5i.com/

n=5

0	5	14	9	20
13	8	19	4	15
18	1	6	21	10
7	12	23	16	3
24	17	2	11	22

n=6

0	15	6	25	10	13
33	24	11	14	5	26
16	1	32	7	12	9
31	34	23	20	27	4
22	17	2	29	8	19
35	30	21	18	3	28

n=7

0	37	30	7	18	35	14
31	28	19	36	15	6	17
38	1	32	29	8	13	34
27	24	39	20	33	16	5
40	21	2	25	44	9	12
23	26	47	42	11	4	45
48	41	22	3	46	43	10

n=8

0	59	38	33	30	17	8	63
37	34	31	60	9	62	29	16
58	1	36	39	32	27	18	7
35	48	41	26	61	10	15	28
42	57	2	49	40	23	6	19
47	50	45	54	25	20	11	14
56	43	52	3	22	13	24	5
51	46	55	44	53	4	21	12

图 17-3

【习题】

（1）证明当 n 是奇数时，不可能有回到起点的路径。提示：西洋棋盘上的格子都是黑白相间的，如果现在在白格上，下一格一定是黑格；另外，当 n 是奇数时，n^2 也是奇数，于是……

（2）请把这个程序写成递归的形式。注意，递归的层次可能高达 n^2 层，小心系统中的堆栈可能不够用。

（3）De Moivre 曾经提过一个方法来走 8×8 的棋盘。他的方法是：首先把棋盘分成内层与外围，外围就是围在棋盘四周，宽度为两格的一个环形，内层则是一个 4×4 的棋盘。走法如下：假设起点在外围，于是就沿一个固定的方向绕着外围走，只有在无法进展时才进入内层，但立刻回到外层，一直到外围走完之后，才进入内层。内层只有 4×4，而且在此之前就已经用了一些格子，所以不足 16 个，这就很好走了，回溯的地方通常只有进入内层要进的那一格，与从内层回到外围的那一格，以及外层的 4×4 棋盘走法，因此操作是比较简单的，如图 17-4 所示就是一个走法。

如果起点是在内层，那就把操作反过来，先走完内层。试写一个程序，不用递归的方式来实现 De Moivre 方法。

（4）还有一个很有名但不很可靠的方法，它是在 1983 年由 H.C.Warnsdorff 提出的。其原理很简单，如果现在的位置是 X，下一个位置可以到 $Y_1, Y_2, \cdots, Y_k$，如果从 X 走到 Y_i 之后，自 Y_i 起能到的有 p_i 格，于是就走到最小的 p_i 值所对应的 Y_i 去。比如说，如果 X 可以到 Y_1, Y_2, Y_3，而从 Y_1 起有 3 种不同可能的走法，从 Y_2 起有两种，从 Y_3 起有 5 种，当然以 Y_2 的为最小，因此就从 X 走到 Y_2 去。Warnsdorff 的原则有点像是先把难事（能够走的格子数比较少）做完了，其他的就简单了。当走到一个格子无法找出下一步时，就停止，

Download at http://www.pin5i.com/

继而显示结果。

34	49	22	11	36	39	24	1
21	10	35	50	23	12	37	40
48	33	62	57	38	25	2	13
9	20	51	54	63	60	41	26
32	47	58	61	56	53	14	3
19	8	55	52	59	64	27	42
46	31	6	17	44	29	4	15
7	18	45	30	5	16	43	28

图 17-4

请把 Warnsdorff 方法写成程序，对于那些同时为最小的位置而言，可以任择其一，再用不同的起点来测试程序。它会比回溯法快吗？但是 Warnsdorff 法会有失败的时候，也就是说，当它停止时，有可能还有未曾走过的格子，能找得出几个例子来吗？

（5）续题（4）。如果把程序改成在有好几个极小值的地方并不任选一个，而是进行回溯。换言之，一个接一个地试下去，直到走通为止。程序会成功地以 64 个不同格子作起点而走遍 64 格吗？改成 m・n 棋盘还对吗？

【参考文献】

武士巡逻问题从什么时候被人提出，以及由谁提出，似乎已经不可考证，但可以肯定的是，L.Euler 在 1759 年就研究过这个问题，所以说起源是相当早的。习题（3）、（4）与（5），以及进一步的参考资料与方法（比如 Roget 法），可以参看下面的这两本书。

1 W.W.Rouse Ball and H.S.M.Coxeter. *Mathematical Recreations and Essays*. 13th ed., Dover, 1987

2 M.Kraitchik. *Mathematical Recreation*. 2nd ed., Dover, 1953

B.R.Stonebridge 在文章中指出 Warnsdorff 方法只向前一步（见习题（4））是不够的，他建议要向前看两步才行，并作了程序的实验报告，在 8×8 棋盘上不会有失败的例子。有兴趣的朋友请参考他的文章，在这篇文章中还附有 Basic 写成的程序。

3 B.R.Stonebridge. The Knight's Tour of a Chessboard, *Mathematical Spectrum*. Vol.19 (1986/87), No.3, pp.83～89

【程序】

程序 17-7（KNIGHT.C）

```
/* ------------------------------------------------------------------ */
/* PROGRAM knight tour :                                              */
/*   Given a n*n chess board and a starting position,                 */
/* this program will find a knight tour path passing                  */
/* through each square on the chess board exactly once by             */
/* using backtrack technique without recursion.  For                  */
```

Download at http://www.pin5i.com/


```
/* larger n, for example n > 6, it will take very long          */
/* time to find such path.  Therefore be patience please.       */
/*                                                              */
/* Copyright Ching-Kuang Shene          July/20/1989            */
/* ------------------------------------------------------------ */

#include <stdio.h>
#include <stdlib.h>

#define  MAXSIZE     10      /* max. board size                 */
#define  MAX_STACK   100     /* stack size = board-size^2       */
#define  SUCCESS     1       /* return value for a succ.        */
#define  FAILURE     0       /* for a failure.                  */
#define  EMPTY   -1          /* value to indicate empty         */

/* ----------------- external variables ------------------      */

int  board[MAXSIZE+1][MAXSIZE+1];   /* chess board              */
int  n;                             /* working board size       */
int  offset_x[] = { 2,  1, -1, -2, -2, -1,  1,  2};
int  offset_y[] = { 1,  2,  2,  1, -1, -2, -2, -1};

int  path_x[MAX_STACK+1];    /* stack for x coordinate          */
int  path_y[MAX_STACK+1];    /* stack for y coordinate          */
int  direction[MAX_STACK+1]; /* stack for direction             */
int  top;                    /* stack pointer                   */

/* ------------------ function prototype ------------------     */

void initial(void);
void display(void);
int  try(int, int);

/* ----------------- main program --------------------          */
void  main(void)
{
     int   row, column;
     char  line[100];

     printf("\nRecursive Knight Tour Problem");
     printf("\n==============================");
     printf("\n\nBoard Size ----> ");
```

Download at http://www.pin5i.com/


```
    gets(line);
    n = atoi(line);
    printf("Start Row -----> ");
    gets(line);
    row = atoi(line);
    printf("Start Column --> ");
    gets(line);
    column = atoi(line);

    initial();
    if (try(row, column) == FAILURE)
        printf("\nNO SOLUTION AT ALL.");
    else
        display();
}


/* ----------------------------------------------------------- */
/* FUNCTION initial :                                          */
/*   initialize the chess board to EMPTY.                      */
/* ----------------------------------------------------------- */

void  initial(void)
{
    int  i, j;

    for (i = 1; i <= n; i++)
        for (j = 1; j <= n; j++)
            board[i][j] = EMPTY;
}


/* ----------------------------------------------------------- */
/* FUNCTION display :                                          */
/*   display to chess board.                                   */
/* ----------------------------------------------------------- */

#define  DRAWGRID(N)     {   int i;                            \
                             printf("\n+");                    \
                             for (i = 1; i <= N; i++)          \
                                 printf("--+");                \
                         }
```

Download at http://www.pin5i.com/


```
#define  DRAWLINE(N, r)  {   int i;                                  \
                              printf("\n|");                         \
                              for (i = 1; i <= N; i++)               \
                                  printf("%2d|",board[r][i]);        \
                          }


void  display(void)
{
     int  r;

     printf("\n\nHere is One Possible Solution :\n");
     DRAWGRID(n);
     for (r = 1; r <= n; r++) {
        DRAWLINE(n,r);
        DRAWGRID(n);
     }
}


/* ----------------------------------------------------------------- */
/* FUNCTION try :                                                    */
/*   The main non-recursive backtrack working routine.               */
/* ----------------------------------------------------------------- */

#define  YES 1
#define  NO  0
#define  BOARD(x,y)  (1<=x) && (x<=n) && (1<=y) && (y<=n)
#define  CHECK(x,y)  board[x][y] == EMPTY
#define  PUSH(x,y)   {   top++;                                       \
                         path_x[top] = x; path_y[top] = y;            \
                         direction[top] = 0;                          \
                         board[x][y] = top;                           \
                     }

int  try(int x, int y)
{
     int new_x, new_y;
     int found;
```

Download at http://www.pin5i.com/


```
top = -1;                    /* initial to empty              */
PUSH(x, y);                  /* push first pos. and dir.      */
while (top < n*n-1) {        /* loop until the board full     */
    found = NO;
    while (direction[top] < 8) {    /* try all 8 pos.         */
        new_x = path_x[top] + offset_x[direction[top]];
        new_y = path_y[top] + offset_y[direction[top]];
        if (BOARD(new_x,new_y) && CHECK(new_x,new_y)) {
            PUSH(new_x, new_y);  /* a new pos. PUSH           */
            found = YES;         /* set flag                  */
            break;               /* try next pos.             */
        }
        else
            direction[top]++;/* OR try next dir               */
    }
    if (!found)                  /* if no new pos. is found */
        if (top > 0) {           /* do we have prev. item?  */
            board[path_x[top]][path_y[top]] = EMPTY;
            direction[--top]++;  /* YES, backtrack            */
        }
        else
            return FAILURE;      /* otherwise, FAILURE        */
    }
    return SUCCESS;              /* all pos. visited. DONE  */
}
```

问题 8.7 环游世界（HAMILTON.C）

假设有一张地图，城市与城市之间有道路相连，请写一个程序，在这份地图中找出一条把每一个城市都只走一次而不重复，并且还会回到起点的路线。如果找不出这条路线，也请报告这个现象。这个问题就叫做 Hamilton 循环（Hamilton Cycle）问题。

【解答】

首先讲数据结构的问题。第一是如何表示两个城市之间有路相连。这个问题可以用一个矩阵表示。在程序中准备了一个矩阵 connected[ ][ ]，如果 connected[i][j]的值为 YES，i 与 j 就有路相连。但是 i 与 j 有路，反过来看 j 与 i 不也有路相连吗？所以 connected[ ][ ]这个矩阵是对称的。在输入中如果 i 与 j 有路相连，就会在一列上有 i 与 j 这两个值，所以在输入了这一列时，不但要把 connected[i][j]定成 YES，还得把 connected[i][j]也定成 YES 才行。至于其他没有因为输入而定成 YES 的字段，就都是 NO，表示没有路径相连。第二，

Download at http://www.pin5i.com/

要有一个数组来存放行经的路径，在程序中就把它叫做 cycle[ ]。不但如此，还要用一个额外的数组来记录哪一个城市已经走过了，哪一个还没有。程序中是用 visited[ ]表示，但是因为它只是个工作用的数组，所以它所占用的内存是用 malloc()取得的，用完之后就通过 free()还给系统。

有了以上的基础，现在就可以讲程序的核心部分了。方法是这样的，在处理的过程中会把经过的城市依照次序存放在 cycle[ ]数组里，而 cycle[top]则是下一个要处理的城市。如果对城市 i 而言，visited[i]为 YES，这就表示已经到过了 i，因而下一步不能走到 i 去。

如图 17-5 所示，已经走到过的城市用圆圈围住，而下一个可能的城市就在 cycle[ ]中的最后一个（top）位置。可以从 cycle[top-1]中知道已经走到了哪一个城市，那么如何决定下一个城市呢？城市有 n 个，可以一个接一个地查过去，当查到第 i 个时，要如何决定它能够环绕在路径上呢？这有两个条件：

（1）visited[i]为 NO：这表示 i 这个城市还没有走到过。

（2）connected[cycle[top-1]][i]为 YES：因为 cycle[top-1]是最后一个被加到路径中的城市，而 connected[x][y]为真时就表示 x 与 y 有路相通，因此上面的条件就是说 cycle[top-1]与 i 有路相通。

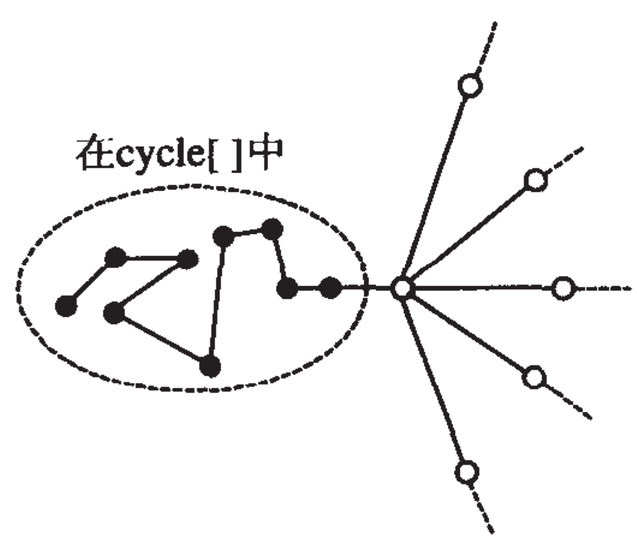


图 17-5

若 i 没有走过而且又与上一个城市 cycle[top-1]有路通，就可以走到 i 去，因此 cycle[top-1]的下一站就是 i，继而把 i 存入 cycle[top]就完成了延长路径的工作。那么下一个城市在何处？不知道，但总是从头开始查。程序中对程序的编号是 0～n-1，而不是 1～n，因而把 0 存入 cycle[top+1]，就表示下一个要查的城市是 0。

万一从 cycle[top-1]起根本到不了任何一个城市，也就是说与 cycle[top-1]相通的城市都在此以前就走过了（亦即 visited[i]为 YES），那就表示 cycle[top-1]是一个不正确的选择，才会造成走投无路的情况，所以回溯的操作就发生了。由于 cycle[top-1]是从 cycle[top-2]走过来的，但 cycle[top-1]是与 cycle[top-2]相连的城市中的一个，因此不妨试一试 cycle[top-1]+1 这个城市。换句话说，如果目前 cycle[top-1]的值是 5（表示城市 5），但却造成无法再延长路径，不妨试试 6。

于是程序就这样地回溯、增长，如果到了某个地步，cycle[ ]中已经放了 n 个元素，这不就是 n 个城市都走到了吗？但工作做完没有，还要查一下最后的一个城市 cycle[n-1]是否与 start（开始城市）相连，如果相连，环绕的路径就找出来，顺着 cycle[0],cycle[1],…,cycle[n-1]再回到 cycle[0]就是这条路径。若 cycle[n-1]与 start 不相连，这表示固然已经有

Download at http://www.pin5i.com/

一条路径经过了所有的城市，但却回不到起点，cycle[n-1]的选择不对，应当试下一个（亦即 cycle[n-1]+1）。

如果在回溯的过程中要把 cycle[0]加 1，回想一下，cycle[0]不正是起点吗？把 cycle[0]加 1，表示 cycle[0]的选择不对，那自然就是从 start 起找不到环绕的路径了，这时停止程序运行，输出“根本找不到这样的路径”。

程序中用 read_in()函数读取数据，用 hamilton()找出环绕的路径，如果找到了，就用 display()显示出来。有了上述的观点，追踪一下程序就可以了解整个程序运行的过程。

另外，不妨用提供的输入数据 HAM.IN1 与 HAM.IN2 来追踪一下程序的工作。会发现，回溯的方法常常是很有效的解题利器。这个程序并没有用递归的方法来写，不过不妨一试。然而当城市的数目很多时，递归的解法在小机器上常会出现问题，原因就在于每碰到一个新的城市，就会调用自己去试一试，因此有多少个城市，递归的调用层次就有多深，很可能常常会见到堆栈溢出（Stack Overflow）的信息。

【习题】

（1）试把这个程序写成递归的程序。

（2）为什么任选一个城市出发，用回溯的方法把所有可能情况都试过之后还找不出答案时，就可以说没有环绕的路径呢？换另一个城市就会找到吗？为什么？

（3）为什么有 n 个城市时递归调用会有 n 层，用递归程序做个实验，是不是常常会有堆栈溢出的情况产生？

（4）如果只要求从某一个城市起找一条路径把所有城市都走一次，但不必回到起点，请修改程序完成这件事。请问路径存在与否与起点有关吗？为什么？

（5）如果城市与城市之间的道路是有方向性的，也就是说，当输入是 i 与 j 时，就表示从 i 可以到 j，但却不一定可以从 j 到 i。请问，在这种情况下，程序的什么地方要修改，还能正常运行吗？

（6）请修改程序，找出从起点起，所有不同的、会回到起点的路径。

【参考文献】

这个问题就是有名的 Hamilton 循环问题，如果不要求回到起点，就叫做 Hamilton 路径问题（这个问题是以数学家 William Rowan Hamilton 爵士命名的）。事实上，最先做系统研究的并不是他，而是与他差不多时代的另一位数学家 Thomas Renyngton Kirkman，但 Hamilton 使这个问题成名则是不争的事实。Hamilton 在 1856 年 10 月的信件中讨论了这个问题，并且还设计了一个游戏，以 25 英磅卖给一位专门出版游戏与谜题的商人，但在市场上没多大成功。

在计算机科学界，找 Hamilton 循环是一个“极难”的问题之一。说“难”，并不是说程序难写，而是到目前为止，还不能找出一个很有效率的方法来解这个题目。简单地说，如果有 n 个城市，程序大约要用与 2^n 成正比的时间才能知道会不会有 Hamilton 循环。但是，计算机科学家根本就不知道有没有更快的方法，这是一个所谓 NP-完全性（Nondeterministic Polynomial—time Complete）的问题。

程序只找出一个环绕的路径，但是，若记忆足够，也不难找出所有的环绕路径（习题

Download at http://www.pin5i.com/

（6））。有关 Hamilton 循环的源起、数学的讨论，以及程序写作的问题，可参看笔者一篇介绍性文章。

冼镜光．Hamilton 循环一环游世界，图论与它的应用之二，信息传真微电脑版．Vol.9(1989), No.6, pp.150～165

【程序】

程序 17-8（HAMILTON.C）

```
/* ------------------------------------------------------------ */
/* PROGRAM  Hamilton cycle finder :                             */
/*   Given an undirected graph with at least 3 vertices,        */
/* this program dins out one of the Hamilton Cycle.  If         */
/* there is no such cycle at all, this result will also         */
/* be reported.  NOTE that Hamilton Cycle problem is a          */
/* NP-Complete problem and so don't expect any efficient        */
/* algorithm.  This program employs the backtrack method        */
/* used in previous problems.                                   */
/*                                                              */
/* Copyright Ching-Kuang Shene          July/23/1989            */
/* ------------------------------------------------------------ */

#include <stdio.h>

#define  MAXSIZE     30      /* no more than 30 vertices        */
#define  ALWAYS       1
#define  SUCCESS      1
#define  FAILURE      0
#define  YES          1
#define  NO           0

/* ------------------------------------------------------------ */
/* FUNCTION  read_in :                                          */
/*   Read in the number of vertices and the edges.  Then        */
/* build up the adjacenct matrix.  NOTE that because the        */
/* graph is undirected, an edga (s,t) will represent two        */
/* 'edges' in the graph, namely (s,t) and (t,s).  This          */
/* function reflects this fact by set two entries.              */
/* ------------------------------------------------------------ */

void  read_in(int connected[][MAXSIZE], int *n)
{
     int  i, j;
```

Download at http://www.pin5i.com/


```
    scanf("%d", n);                 /* read in the number of VTX    */
    for (i = 0; i < *n; i++)        /* clear the adjacent matrix    */
        for (j = 0; j < *n; j++)
            connected[i][j] = NO;

    scanf("%d%d", &i, &j);              /* read in first edge       */
    while (i != 0 && j != 0) {          /* end ?                    */
        connected[i-1][j-1] = YES;  /* NO, setup two                */
        connected[j-1][i-1] = YES;  /* symmetric edges              */
        scanf("%d%d", &i, &j);          /* read next one            */
    }
    for (i = 0; i < *n; i++)            /* clear diagonal           */
        connected[i][i] = NO;
}

/* ------------------------------------------------------------     */
/* FUNCTION hamilton :                                              */
/*   Given the adjacent matrix connected[], the number              */
/* of vertices and the start vertex, this function finds            */
/* a Hamilton Cycle and stores it in cycle[].                       */
/* ------------------------------------------------------------     */

int hamilton(int connected[][MAXSIZE],int cycle[],int n,int start)
{
    int  *visited;                  /* visited marks                */
    int  top, i;

    visited = (int *) malloc(sizeof(int)*n); /* get mem.            */
    for (i = 0; i < n; i++)         /* clear marks to NO            */
        visited[i] = NO;
    visited[start] = YES;           /* but the start has visited    */

    cycle[0] = start;               /* the cycle has 'start'        */
    cycle[1] = 0;                   /* next in cycle is VTX 0       */
    top      = 1;                   /* the top working element      */

    while (ALWAYS) {                /* loop until done              */
        for (i = cycle[top]; i < n; i++)/* find next                */
            if (connected[cycle[top-1]][i] && !visited[i])
                break;
        if (i < n) {                /* all neighbors tried?         */
            cycle[top] = i;         /* NO, a new vertex in cycle    */
            visited[cycle[top]] = YES;
```

Download at http://www.pin5i.com/


```
                if (top == n-1 && connected[cycle[top]][start]) {
                    free(visited);      /* if all visited ...       */
                    return SUCCESS;     /* return SUCCES            */
                }

                else                    /* otherwise, advance.      */
                    cycle[++top] = 0;
            }
            else {                      /* next not found ...       */
                visited[cycle[--top]] = NO; /* backtrack            */
                if (top == 0) {         /* return to the start VTX?*/
                    free(visited);      /* YES, failed.             */
                    return FAILURE;
                }
                    cycle[top]++;       /* NO, move to neighbr      */
            }
        }
    }

    /* ------------------------------------------------------------ */
    /* FUNCTION  display :                                          */
    /*   Display the found cycle.                                   */
    /* ------------------------------------------------------------ */

    void  display(int cycle[], int n)
    {
        int  i;

        printf("\n\nA Hamilton Cycle is Listed as Follows :");
        printf("\n\n%d", cycle[0]+1);
        for (i = 1; i < n; i++)
            printf("->%d", cycle[i]+1);
        printf("->%d", cycle[0]+1);
    }

    /* ------------------------------------------------------------ */

    void  main(void)
    {
        int  connected[MAXSIZE][MAXSIZE];
        int  cycle[MAXSIZE];
        int  n;
```

Download at http://www.pin5i.com/


```
    printf("\nHamilton Cycle Program");
    printf("\n======================");
    read_in(connected, &n);
    if (hamilton(connected, cycle, n, 0) == SUCCESS)
        display(cycle, n);
    else
        printf("\n\nNO Hamilton Cycle at all.");
}
```

问题 8.8 一笔画（EULER.C）

请设计一个可以画一笔画的程序，在程序中应该想到要如何设计合适的数据结构，以及如何表示一笔画的动作。

【解答】

首先还是从数据结构出发，请注意问题说明中的图，两个点之间的连线有时会不止一条，因此记录两个点之间的关系时就不再是“通”还是“不通”的问题，而变成有多少条边线的说法，所以准备一个矩阵 connect[ ][ ]，如果读到 i 与 j 有一条边相通，就往 connect[i][j]和 connect[j][i]中加上 1。又因为 Euler 的定理，则当奇点数目太多时就不会有一笔画，所以也要多一个数组 deg[ ]；当读到 i 与 j 之间有一条边时，deg[i]与 deg[j]就都加上 1；当所有的边都读完了，connect[i][j]（或 connect[j][i]）就记录着 i 与 j 之间有多少条边相通，而 deg[i]则正好是 i 的次数。C 语言中数组一定从 0 点起，当读到 i 与 j 时，要把 connect[i-1][j-1]、connect[j-1][i-1]、deg[i-1]、deg[j-1]加上 1，因此在外界顶点是 1,2,3,…，但在程序内部却用的是 0,1,2,…。函数 read_in()就是作输入函数，EULER.IN1 是七桥问题的图，EULER.IN2 是问题说明中有两个奇点的图，EULER.IN3 则是完全没有奇点的图。

程序中掌握主控权的是函数 euler()，它会返回一个指针，指向一个连接串行，如果把这个连接串行中所记录的顶点号码依次显示出来，就正好是走一笔画时所经过的各个顶点。如何找出这个代表一笔画的连接串行呢？其实很简单，首先它调用 prepare()函数，由该函数先找出可以从什么地方开始走一笔画，并且建立好一个连接串行，开头在 head，结尾在 tail 所指出的地方。但若 prepare()无法做到这一点，换句话说，prepare()发现根本不能做一笔画，就会返回 FALL。假设 prepare()成功地建立了一个初步的、代表一笔画的连接串行，开始在 head，结尾在 tail。euler()中剩下来的 while 循环就只不过是不断地尝试延伸这个串行，以便把所有的边都包含进去而已。

下面介绍它是如何做延伸的工作。它从已有的一笔画的尾部 tail 开始，把这个指针放到 current 中，以后就用 current 来运行。在每一次 while 的循环中，都会调用 find_next()函数，由它从 current 所指出的顶点起所能够到达的下一个顶点，并且存放在 VTX 中，而且让 current 指向那个顶点。如果找不出来，就表示所有的边都已经走过了，一笔画操作完成。

Download at http://www.pin5i.com/

找出这个顶点之后，交给函数 find_trail()，让它从该顶点起去找一笔画，并且把一笔画所经过的路径用连接串行表示，开头在 p1，结尾在 p2。不过应注意一点，在这个串行中，不包含给 find_trail()的 VTX。做到这里，有两个串行，一个是在先前找出来的，head 与 tail 指向头与尾，current 指向现在的起点；另一个是新指出来的串行，p1 与 p2 分别指向头与尾，如图 17-6 所示。

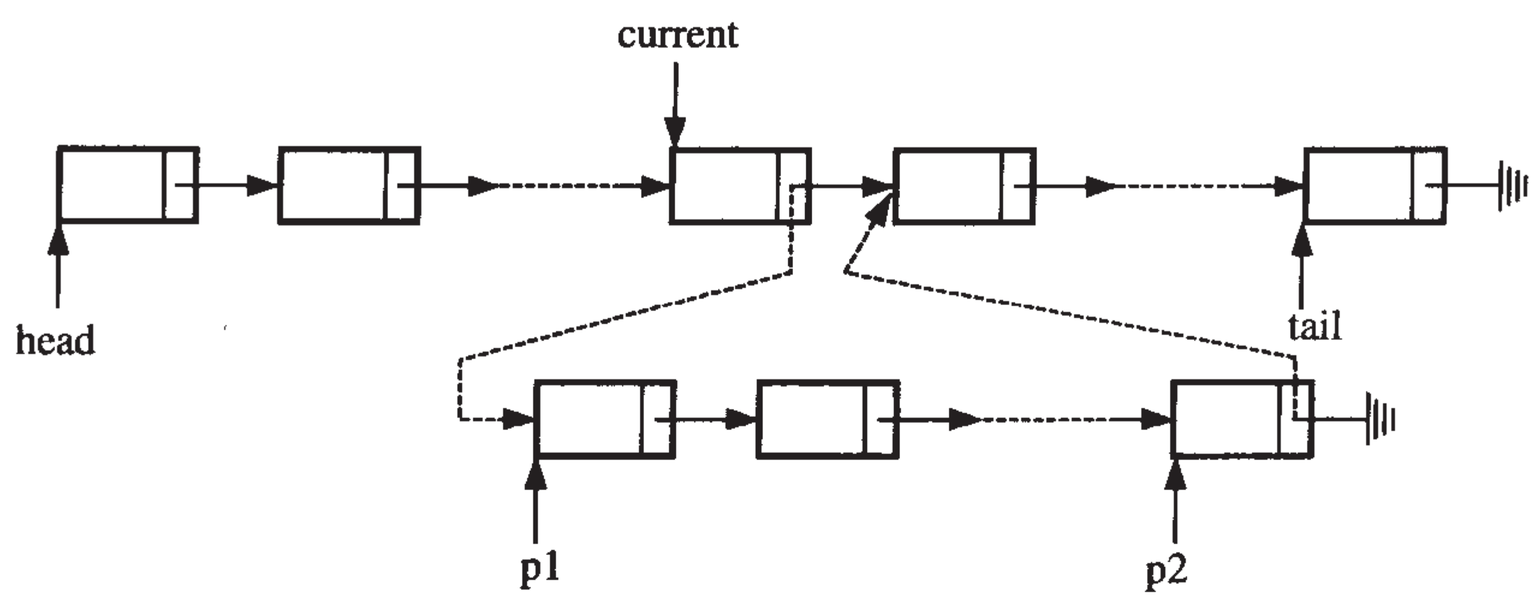


图 17-6

因此，要把新找出来的一笔画插到原来的一笔画中，就像是图 17-6 中的虚线，换句话说，一笔画从 head 所指出的那个顶点起一直进行到 current 所指出的顶点，接着就进入 p1 所指出的顶点，用新增的部分一直走到 p2 所指出的顶点，再回到原一笔画中 current 的下一个再走到结尾。插入完成之后，再回头反复相同的操作，直到找不出来为止。

以 EULER.IN3 完全没有奇点的图为例进行说明，如果从 A 开始，就会有如图 17-7 所示。

一旦回到 A，A 的所有边就都用完了（注意，B 与 C 都用掉了两条边）。自然，下一次 find_next()会找出 B，因为 B 是第一个碰到而且还有边可用的顶点，这一次它能够走的图如图 17-8 所示（注意，用完的边已经拿掉了）。

图 17-7

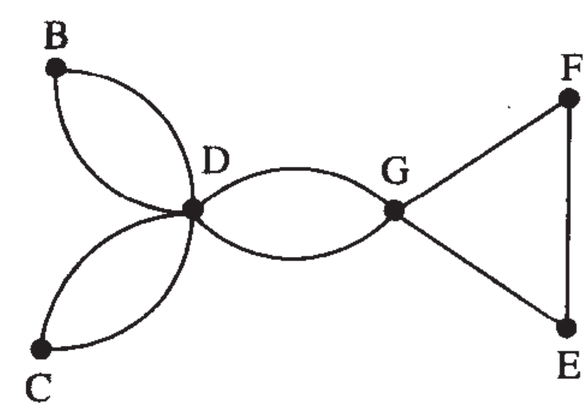


图 17-8

或许 find_trail()会找到下面的结果，如图 17-9 所示。

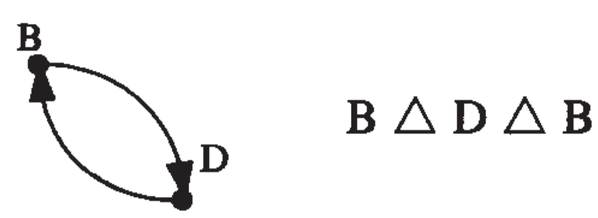
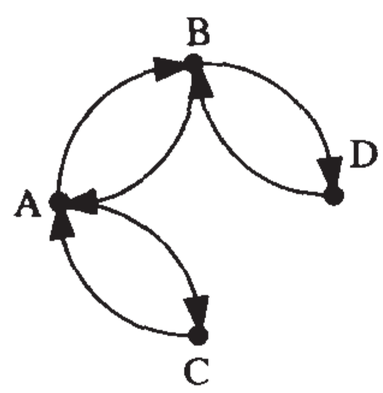


图 17-9

把它插回上一次的串行中，如图 17-10 所示。

Download at http://www.pin5i.com/



A △ B △ D △ B △ A △ C △ A

图 17-10

下一次能够走的如图 17-11 所示。

find_next()可能会找到 D，从 D 起去走一笔画，因此 find_trail()会得到如图 17-12 所示的情况。

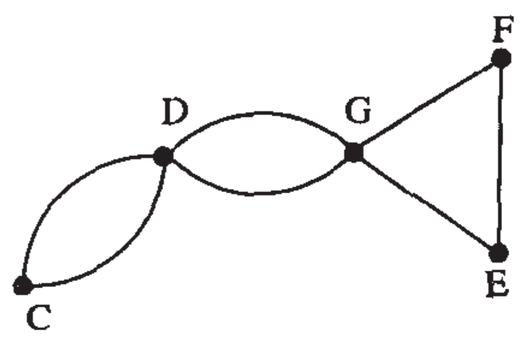
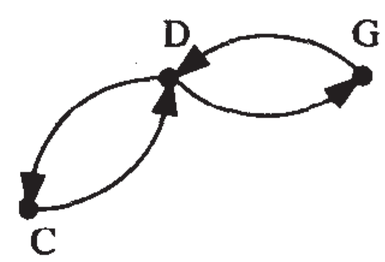


图 17-11

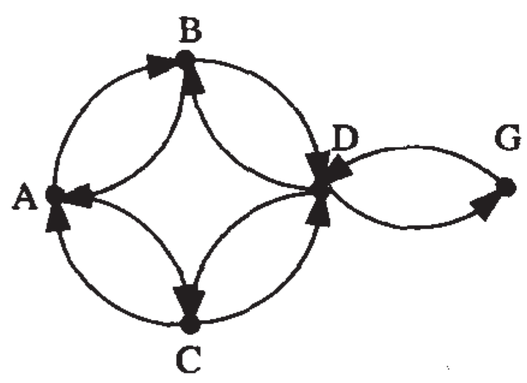


D △ C △ D △ G △ D

图 17-12

把这个新找出来的部分嵌回原来 D 的所在地，如图 17-13 所示。

最后剩下未走的部分如图 17-14 所示。



A △ B △ D △ C △ D △ G △ D △ B △ A △ C △ A

图 17-13

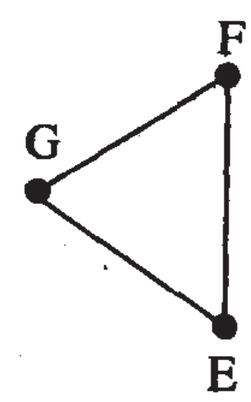


图 17-14

find_next()找到 G，find_trail()从 G 起找出的一笔画是 G→E→F→G，再放回 G 的所在，就会得到最终的结果：

A→B→D→C→D→G→E→F→G→D→B→A→C→A

现在再看 prepare()。第一个 for 循环检查有多少个奇点，并且把遇到的第一与第二个奇点放在数组 odd[ ]中。如果发现奇点大于两个，就返回 FAIL，此时不可能有一笔画。如果恰好有两个奇点，prepare()就用这两个奇点建立一个串行，当成是初步的一笔画（从其中一个奇点到另一个奇点），而且把这两个奇点之间的 connect[ ][ ]减去 1 反应出这一条边已经用过了。另外，这两个奇点的 deg[ ]也得减去 1，最后得到的图就是完全没有奇点的图了，而 euler()在收到这个初步结果之后就会尝试在尾部延伸一笔画的成果。但若所有点都是偶点，初步结果中就只有一个顶点，可以任选一个，因为一笔画会走完每一条并从每一个顶点而回到起点，所以选哪一个作起点都一样，此例中选 0。

Download at http://www.pin5i.com/

find_next()收到一个指向某个已经在一笔画中的顶点的指针（这个顶点已经被串到表示局部结果的连接串行中），于是它就从该顶点起检查连接串行中的各个顶点，看有没有哪一个的次数（也就是剩下来还没有走的边线）不为 0，如果有，这个顶点的号码就是函数值，而参数则是指向这个才被找出来的元素的指针。注意，因为该顶点还有边，因此表示从它开始至少还可以走出一条不完整的一笔画。

find_trail()是程序中工作最繁重的部分。它收到一个叫它找一笔画的起点 start，并且希望它把找出来的一笔画的行经路径放到一个连接串行中，返回指向这个串行的头与尾的指针（head 与 tail）。它的工作是，先把 start 的值存到一个工作用的变量中，然后就进入 while 循环开始自 p 起找一笔画中的下一个点。下一个点如何找？connect[p][i]表示 p 与 i 之间的边线数目，如果有个 i 使 connect[p][i]不为 0，从 p 就可以走到 i 去。正因为从 p 走到了 i，p 与 i 之间的边线就去掉了一条，因此 connect[p][i]与 connect[i][p]减去 1。当然，记录下与 p（与 i）相邻的 deg[p]（与 deg[i]）的次数数组也要减 1 来反映这件事实。接着为 i 建立一个连接串行的节点，并且串到连接串行的尾部，这就大功告成了。因为已经从 p 走到 i，所以把 i 存到 p 去，回到循环开头，于是下一个循环就相当于从 i 出发走一笔画了。但是，若 for 循环找不出 i，亦即 p 的每一个邻居都走过了，于是一笔画就到 p 结束，p 一定与开始时的 start 相等（为什么？）。另外，在连接串行中的顶点不是从 start 开始，而是从 start 的下一个元素开始，却一定在 start 结束。

display()函数很简单，就留给读者研究了。

【习题】

（1）为什么在 find_trail()中所找出来的一笔画连接串行一定在 start 处结束？

提示，每一个点都是偶点。

（2）本问题的解法故意略去这个程序背后的基本技巧，能够从解法的例子、程序的运行方式，一一拼凑出来吗？

（3）如果用 n,m 表示这个图的顶点数与边数，请估计这个程序一定做了多少处理操作？

（4）这个程序中的许多地方对指针的作用，对初学者来说可能觉得很诡异，请仔细研读程序，特别是 find_next()，了解为什么是如此用法。

【参考文献】

有关一笔画的历史源流与初期发展，可参看下面一本很有趣的书的第 1 章。

1 N.L.Biggs, E.K.Lloyd and R.J.Wilson. *Graph Theory* 1736—1936, Oxford University Press: 1976

一般而言，大家对 Euler 的七桥问题的解答都认为是图论（Graph Theory）的开山之作，所以图论到现在已经有了 250 年以上的历史，以下是几篇有关七桥问题的介绍。

2 L.Euler. Leonhard Euler and The Königsberg Bridges, *Scientific American*. Vol.189 (1953), July, pp.66～70

3 H.Sach, M.Stiebitz and R.J.Wilson. An Historical Note: Euler's Konigsberg Letters,

Download at http://www.pin5i.com/

Journal of Graph Theory. Vol.12(1988), pp.133～139

4 R.J.Wilson. An Eulerian Trail Through Königsberg, *Journal of Graph Theory*. Vol.10 (1986), pp.265～275

下面的一篇短文介绍了如何用数学归纳法，在一页之内证明 Euler 对七桥问题的解法（Euler 没有完全解决问题）。

5 D.A.Fowler. The Konigsberg Bridges-250 Years Later, *American Mathematical Monthly*. Vol.95(1988), pp.42～43

如果要走的图的边线是单行道，只能沿某个特定的方向进行，下面的这篇文章提供一个解法。

6 J.Ebert. Computing Eulerian Trails, *Information Processing Letters*. Vol.28(1988), pp.93～97

在中文方面，笔者曾经在《漫谈 Micro》中提过，内容包含了历史变革等趣事，但那些文字中的解法不是很好，但却比较容易懂。

7 冼镜光．一笔画—图论与它的应用之一，信息传真．Vol.8(1988), No.14, pp.168～176

另外，一笔画在大多数文献资料中都叫做 Euler Trail，喜爱读原文著作的朋友不可不知。

【程序】

程序 17-9（EULER.C）

```
/* ----------------------------------------------------------- */
/* PROGRAM  euler :                                            */
/*   This porgram accepts a multi-graph and finds an           */
/* Euler Trail if possible.                                    */
/*                                                             */
/* Copyright Ching-Kuang Shene          July/30/1989           */
/* ----------------------------------------------------------- */

#include <stdio.h>
#include <stdlib.h>

#define  MAXSIZE     20
#define  ALWAYS      1
#define  YES         1
#define  NO          0
#define  SUCCESS     1
#define  FAIL        -1


/* ------------------------------------------------------------ */
/*            types and external variables                      */
/* ------------------------------------------------------------ */
```

Download at http://www.pin5i.com/


```
typedef   struct node  Node;      /* trail node (linked list)     */

struct node {                     /* a node consists of ...       */
     int vertex;                  /*    a vertex number field     */
     Node    *next;               /*    and a next node ptr       */
};

int  connect[MAXSIZE][MAXSIZE]; /* the connection matrix        */
int  deg[MAXSIZE];              /* degree array                 */
int  n;                         /* number of vertices           */
Node *trail;                    /* pointer to Euler trail       */

/* ------------------------------------------------------------ */
/*                function prototypes                           */
/* ------------------------------------------------------------ */

void read_in(void);
Node *euler(void);
int  prepare(Node **, Node **);
int  find_next(Node **);
void find_trail(int, Node **, Node **);
void display(void);

/* ------------------------------------------------------------ */

void  main(void)
{
     printf("\nEuler Trail Program");
     printf("\n===================\n");

     read_in();                      /* get data                 */
     trail = euler();                /* compute Euler Trail      */
     display();                      /* display result           */
}

/* ------------------------------------------------------------ */
/* FUNCTION  read_in :                                          */
/*   This function reads in the connection matrix and           */
/* then compute the degree array.                               */
/* ------------------------------------------------------------ */
```

Download at http://www.pin5i.com/


```
void  read_in(void)
{
     int  i, j;
     char line[100];

     gets(line);                    /* get in number of vertices    */
     n = atoi(line);
     for (i = 0; i < n; i++) {   /* clear the connection mtx     */
         deg[i] = 0;
         for (j = 0; j < n; j++)
             connect[i][j] = 0;
     }
     gets(line);                    /* get 1st line of edge set     */
     sscanf(line, "%d%d", &i, &j);
     while (i != 0 && j != 0) { /* end of file ?                */
         if (i != j) {              /* a loop ?                     */
             connect[i-1][j-1]++;/* increase edge cnt            */
             connect[j-1][i-1]++;
             deg[i-1]++;            /* increase degree count        */
             deg[j-1]++;
         }
         else                       /* ignore all self loops        */
             printf("\n*** ERROR *** A loop found.  Data ignored");
         gets(line);                /* get next edge                */
         sscanf(line, "%d%d", &i, &j);
     }
}

/* ----------------------------------------------------------------- */
/* FUNCTION  euler :                                                 */
/*   This is the main working routine of this program.               */
/* It calls prepare() to initialize various data field and           */
/* some checks.  Then compute the Euler Trail loop by loop.          */
/* ----------------------------------------------------------------- */

Node  *euler(void)
{
     Node    *current;          /* processing cursor            */
     Node    *head, *tail;      /* bound a partial trail        */
     Node    *p1, *p2;          /* working pointer variables    */
     int VTX;
```

Download at http://www.pin5i.com/


```
    if (prepare(&head, &tail) == FAIL) /* prepare data          */
        return NULL;                   /* if fail, return NULL*/

    current = tail;                    /* start from the tail  */
    while (ALWAYS)
        if ((VTX = find_next(&current)) != FAIL) {
            find_trail(VTX, &p1, &p2);
            p2->next = current->next;  /* join the trail       */
            current->next = p1;
            current = p1;              /* step to next node    */
        }
        else
            break;
    return head;                       /* return the trail list   */
}

/* ----------------------------------------------------------------- */
/* FUNCTION prepare :                                                */
/*   This function checks to see if there are more two               */
/* odd degree vertices.  It reject a graph with more than            */
/* two odd degree vertices.  Then it builds a preliminary            */
/* trail list consisting of one node (if all vertices are            */
/* even), or two nodes (for the two odd degree vertices).            */
/* ----------------------------------------------------------------- */

int  prepare(Node **first, Node **last)
{
    Node    *p1, *p2;
    int no, odd_no, i;
    int odd[2];

    for (no = odd_no = i = 0; i < n; i++)  /* test odd deg     */
        if (deg[i] % 2 != 0) {
            odd_no++;
            if (no < 2)
                odd[no++] = i;
        }
    if (odd_no > 2) {                  /* more than two odd deg VTX*/
        printf("\n*** ERROR *** too many odd degree vertices.");
        return FAIL;
    }
    if (odd_no == 2) {                 /* exactly two odd VTX      */
```

Download at http://www.pin5i.com/


```
        p1 = (Node *) malloc(sizeof(Node));     /* get mem.      */
        p2 = (Node *) malloc(sizeof(Node));
        connect[odd[0]][odd[1]]--; /* just remove this          */
        connect[odd[1]][odd[0]]--; /* odd degree edge           */
        deg[odd[0]]--;
        deg[odd[1]]--;
        p1->vertex  = odd[0];        /* these two vertices are  */
        p1->next    = p2;            /* the must for first step */
        p2->vertex  = odd[1];        /* thus put them into the  */
        p2->next    = NULL;          /* trail list              */
        *first  = p1;                /* return this list        */
        *last   = p2;
    }
    else {                           /* all vertices are even   */
        p1 = (Node *) malloc(sizeof(Node));     /* get mem.      */
        p1->vertex = 0;              /* it is the only one node */
        p1->next   = NULL;           /* in the trail list       */
        *first = *last = p1;         /* return the trail list   */
    }
    return SUCCESS;
}

/* ------------------------------------------------------------ */
/* FUNCTION  find_next :                                        */
/*   Given a pointer to some vertex which has already           */
/* been put into trail list, this function scans the trail      */
/* list in order to find a vertex with non-zero degree.         */
/* ------------------------------------------------------------ */

int   find_next(Node **p)
{
    for (;(*p)!=NULL && deg[(*p)->vertex]==0; (*p)=(*p)->next)
        ;
    return ((*p) == NULL) ? FAIL : (*p)->vertex;
}

/* ------------------------------------------------------------ */
/* FUNCTION  find_trail :                                       */
/*   Given a vertex, this function computes a trail             */
/* starting from the given vertex and returns the trail         */
/* list found.                                                  */
/* ------------------------------------------------------------ */
```

Download at http://www.pin5i.com/


```
void  find_trail(int start, Node **head, Node **tail)
{
     Node *first, *last, *ptr;
     int  p, i, done;

     first   = last = NULL;       /* no node in list currently    */
     p   = start;                 /* p is a moving vertex         */
     done    = NO;
     while (ALWAYS) {
         for (i = 0; i < n && !connect[p][i]; i++)
             ;                    /* find a VTX adjacent to p     */
         if (i < n) {             /* p->i is possible             */
             connect[p][i]--,  connect[i][p]--;
             deg[p]--,         deg[i]--;
             ptr = (Node *) malloc(sizeof(Node));
             ptr->vertex = i;     /* make node and put            */
             ptr->next   = NULL; /* it into the trail             */
             if (first == NULL)
                 first = last = ptr;
             else {
                 last->next  = ptr;
                 last    = ptr;
             }
             p = i;               /* step to the next             */
         }
             else                 /* if can not proceed, stop     */
                 break;
     }
     *head = first;               /* return the trail list        */
     *tail = last;
}

/* ------------------------------------------------------------- */
/* FUNCTION  display :                                           */
/*   Simple routine.  It display the Euler Trail.                */
/* ------------------------------------------------------------- */

void  display(void)
{
     Node    *ptr = trail;
     int i = 0;
```

Download at http://www.pin5i.com/


```
    if (trail == NULL)
        return;
    printf("\nAn Euler Trail has been Found :\n");
    for ( ; ptr->next != NULL; ptr = ptr->next, i++) {
        if (i % 15 == 0)  printf("\n");
        printf("%2d->", ptr->vertex+1);
        }
        if (i % 15 == 0) printf("\n");      /* the last item        */
        printf("%2d", ptr->vertex+1);
    }
```

问题 8.9 非递归河内之塔（HANOI_I.C）

河内之塔（Towers of Hanoi）是一个在入门书籍中常见的例题或习题。它是说：有 3 根柱子 1、2 与 3，在柱子 1 上串了从上而下编号是 1,2,…,n 的圆片，号码小的圆片也小。问题是，请写一个程序，把柱子 1 上的圆片搬到柱子 3 去。在搬的时候有 3 个要求：第一，每次只能搬一个圆片；第二，要搬的圆片要从某个柱子取出，并且放到另一根柱子上；第三，任何时刻、任何柱子上的圆片，从上而下都是从小到大排列。其他书上的解法都是用递归的方法写成的程序，请写一个非递归并且不用堆栈来仿真的程序。

【解答】

要解出这个问题，第一件要做的工作就是看看递归法是如何搬圆片的。为了了解递归的工作方式，把历次的调用以及使用的参数通过一个树状结构来表示是很理想的。递归法中有一个 printf 叙述，它是用来显示结果的，如果把它的值放到递归调用的树状结构中，搬圆片的操作就一目了然了。表示方式如图 17-15 所示。

它表示把第 n 号圆片从柱子 i 搬到柱子 j 去。如果圆片总数是 3，在 else 部分如图 17-16 所示。

图 17-15

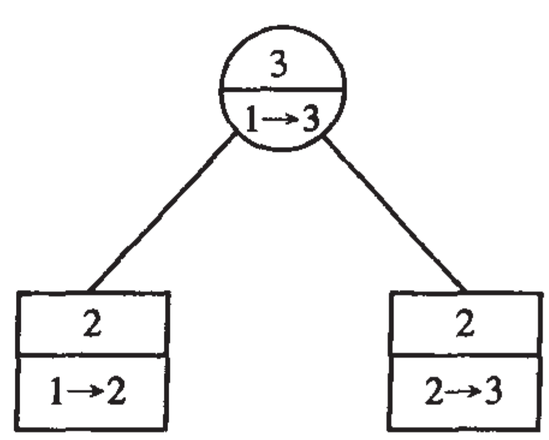


图 17-16

如图 17-17 所示，表示把 k 个圆片从 i 搬到 j 的递归调用。因此进入下一层之后如图 17-18 所示，分解成如图 17-19 所示的情况。

Download at http://www.pin5i.com/

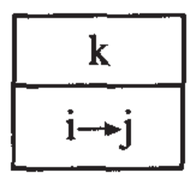


图 17-17

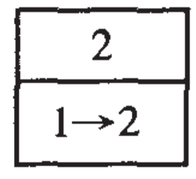


图 17-18

再递归地把只有一个圆片的情况做出来，如图 17-20 所示。

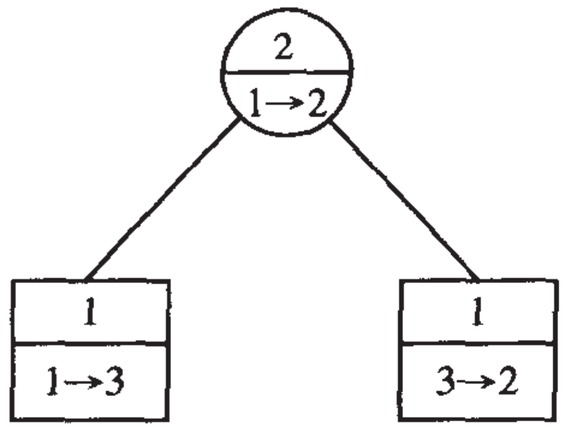


图 17-19

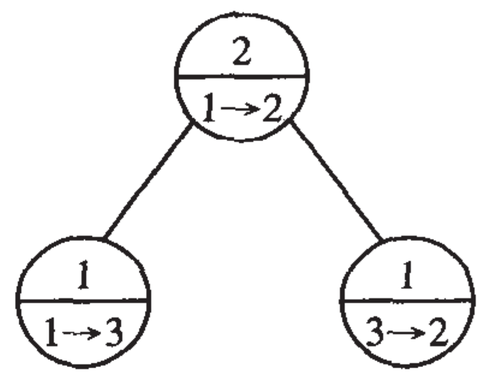


图 17-20

如果把所有的方框都展开，就会有如图 17-21 所示的情况。

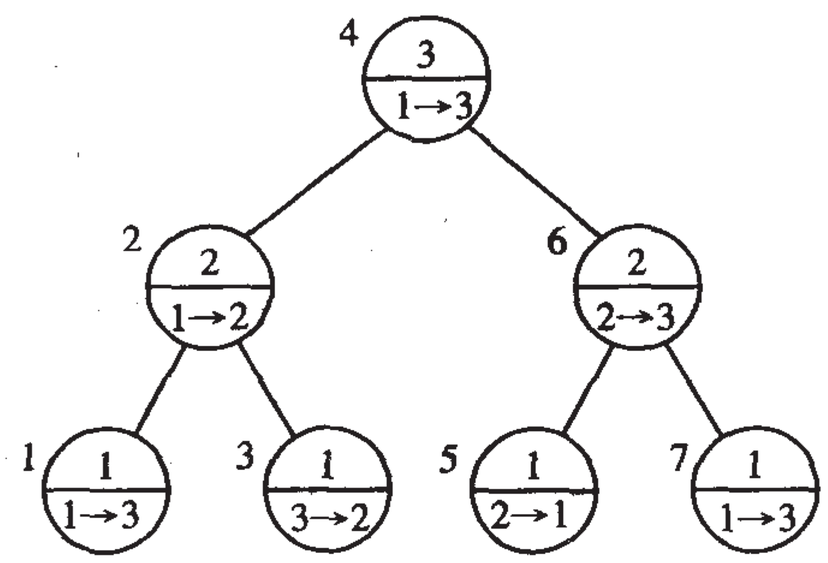


图 17-21

如图 17-22 所示，是搬 4 个圆片的结果，至于更多图片的搬法也是相似的。

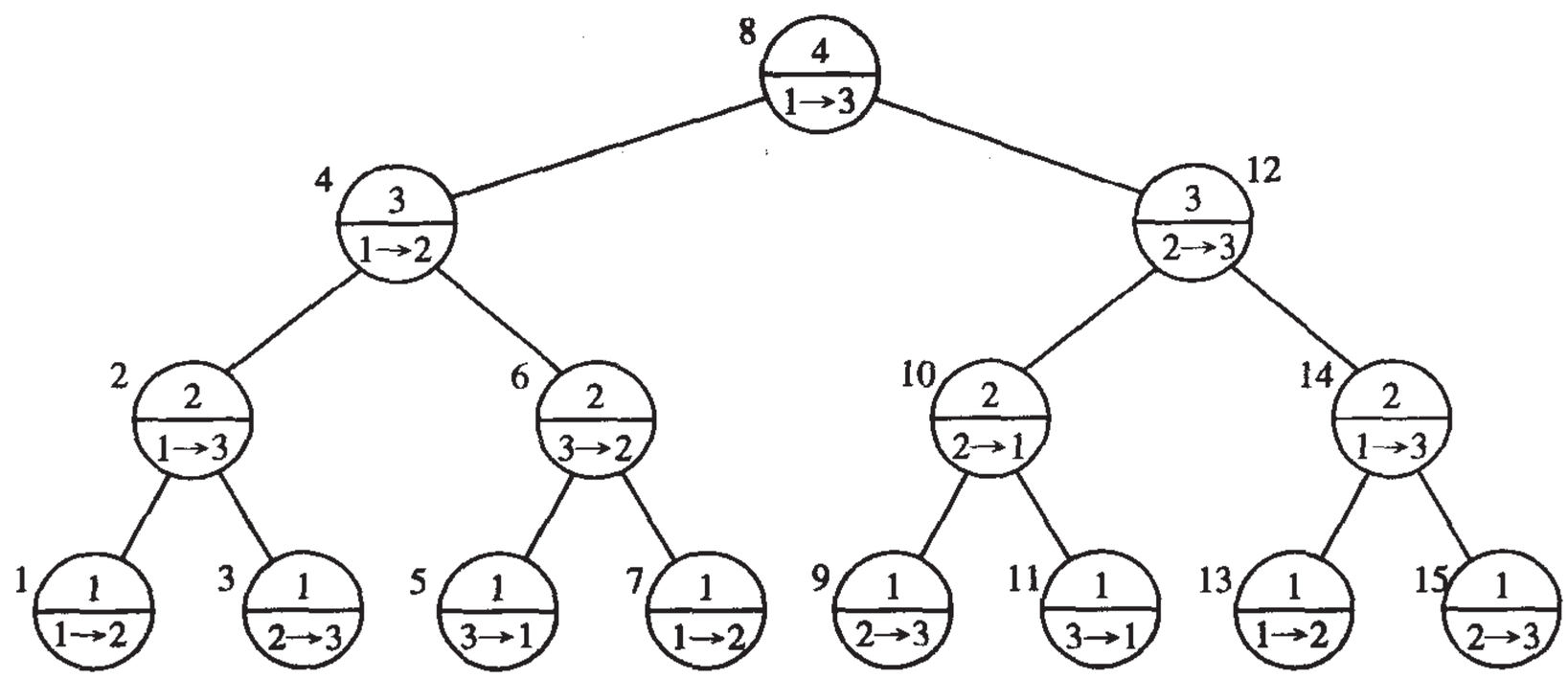


图 17-22

从图 17-21 和图 17-22 可以看到以下几点情况：

（1）最底层，也就是叶部，都是第一号圆片，然后向上数依次为 2,3,4,…。

（2）圆片的搬动方式正好是把那棵树以内序游历一次，把节点上方标着的圆片依下方

Download at http://www.pin5i.com/

的指示搬动。这一点是很明显的，因为这棵树是参照递归程序的运行方式画成，而且在递归程序中的执行方式正是递归地调用自己（走向左边的子树），显示结果（目前的根部），再递归地调用自己（走向右边的子树）。

（3）如果依上一点，也就是把节点中的圆片号码依内序写出来，当 n=3 时是 1,2,1,3,1,2,1；当 n=4 时是 1,2,1,3,1,2,1,4,1,2,1,3,1,2,1；能否看出这正是 Gray 码（参看前面有关 Gray 码的说明，以及对应的程序 GRAYCODE.C）。要想知道搬哪一个圆片，就依次地找出在 Gray 码中究竟哪一个数元从 0 变到 1 即可。

（4）有了该搬哪一个圆片的信息之后，要知道该往哪一根柱子上搬。现在把 3 根柱子排成环形，如图 17-23 所示。

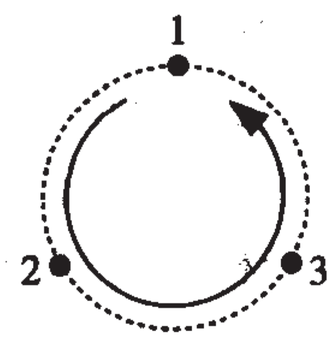


图 17-23

把方向定成反时针，于是 1→2、2→3、3→1 是正向，但 1→3、3→2、2→1 就是反向了，不过如果以反时针来看，把它们说成是正向但搬了两步倒也是事实。在图 17-23 中，根部是 1→3，这是要求永远是反向，从根部往下，则都是正向，再下来是反向，……于是就是正、反向交互出现。

（5）当知道要搬第 k 号圆片时，它目前在柱子 i 上；当 k 在树状结构中的层次决定了之后，就知道是正向还是反向了。如果是正向，就从 i 反时针移一根柱子，如果是反向，则是从 i 反时针移两根柱子。

不过产生 Gray 码却要另一个程序。方法是从 0 起，每次加 1，比如原来是 i，加了 1 之后得到 i+1，于是在 i+1 中从右往左算找出第一个从 0 变成 1 的位数，这一系列的值正好就是 Gray 码。表 17-1 所示就是几个简单的结果。

表 17-1

i	i+1	0→1
0000	0001	1
0001	0010	2
0010	0011	1
0011	0100	3
0100	0101	1
0101	0110	2
0110	0111	1

可以用更多的步骤来检证，至于为什么如此，就请读者自己证明了。

但是一共要搬多少步呢？这倒是容易，去算那一棵树就行了，3 个圆片有 3 层，4 个圆

Download at http://www.pin5i.com/

片有 4 层，依次类推，n 个圆片就有 n 层。在树中每一个节点正好是一个步骤，于是 n 个圆片不就有 2^n-1 步吗？所以，可以用循环让它绕 2^n-1 圈，在绕第 i 次时，看看从上一次的值（i-1）变到这一次的值（i）时，在最右边的从 0 变成 1 的元素是哪一个就行了。

如何定出方向？如果 n 是奇数，如前所述，搬第 n 个一次是反向，于是 n-2、n-4、n-6、…这些奇数值就都是反向；反之，n-1、n-3、…这些偶数值则都是正向，同理可以讨论 n 是偶数的情况。如果是正向，则若目前是在 1,2,3，那么下一次就分别在 2,3,1；因此用 i%3+1 就可以求出下一个柱子何在，i 表示现在的柱子。当是反向时，1,2,3 就分别搬到 3,1,2 去，可以用(i+1)%3 +1 算出来。把它们合起来，如果目前是在第 i 号柱子上，那么下一个位置就是(i+dir)%3+1，正向时 dir 为 0，反向时 dir 为 1。

现在就可以看程序了。在程序中 number_of_disk 相当于此例的 n，亦即圆片数目。number_of_moves 是一共要搬多少次，正是前述的 2^n-1，数组 pin[ ]记录下哪一个圆片在哪一根柱子上，例如 pin[i]=j 就表示第 i 号圆片在第 j 号柱子上。dir[ ]是存放正向与反向各加上多少的数组，与上一段所提的 dir 一样。这是为了方便计算当有奇数个圆片时，dir[ ]的值为 0、1；而当有偶数个圆片时，dir[ ]的值则是 1 与 0。为什么要这样做呢？原因是，当要搬的圆片是 i，而 i 又是偶数，在求下一根柱子时用 dir[0]去加，当 i 是奇数时则加的是 dir[1]，因此就很轻易地躲开了看看 number_of_disk 是偶数还是奇数而决定方向的情况，再由 i 是否为奇数来继承此方向的复杂测试工作。

在一开始时，所有圆片都在第一号柱子上，这是第一个 for 的工作。为了计数方便，用一个变量 counter，让它从 0 起每次加 1。在搬 2^n-1 步的过程中，每一步都把 counter 交给 which_disk()这个函数，由它把 counter 加 1，并且与先前的值相比较，找出在最右边从 0 变成 1 的位置，当作函数值返回来，顺便返回来的 counter 就是加上 1 之后的新值了。函数值指出要搬哪一个圆片，存放在 disk 中。接着看看 disk 是偶数还是奇数，用&运算的理由是，若 disk 是偶数则 index 为 0，否则为 1，那正好是搬动的方向 dir[index]。用前面的方法求出下一个位置 next，显示结果，最后更新 disk 的新位置。

最后看看函数 which_disk()的运行方式，在程序中 a 与 b 分别存放了 counter 与 counter+1 的值。注意，所在意的是 a 到 b 的变化中有哪一个元素从 0 变到 1，而不理会从 1 变到 0 的数元。a 与 b 的关系如下：

```
   xxx.....011 =a
   1.....1
+            1
------------------
   xxx.....100 =b
   0.....0
```

如果 a 的右边有 0 个到若干个 1，再来是一个 0，那么加上 1 之后 1 就会变成 0，只有最靠近 1 的 0 会变成 1，而 a 中在 0 左边的部分就都没有变动。因此用互斥的 OR 把 a 与 b 的结果求出来，于是从 0 变到 1 的数元左边的部分的结果就都是 0（因为在 a 与 b 的对应部分都相同），而从该数元起右边的部分则全是 1（因为在 a 与 b 的对应部分都不一样），因此只要用右移的技巧，一直移到结果为 0 时，就可以算出移了多少位，而这个位数正是

Download at http://www.pin5i.com/

0 变成 1 的那一位的所在，这就是所要求的结果了。

【习题】

（1）请仔细证明 n 个圆片要搬 2^n-1 步。

（2）请证明用内序把本题的树状结构走一次，并且把各节点中的圆片号码显示出来，那正是 Gray 码中惟一被改变的数元的位置。

（3）请证明每次把一个数加 1，而前后两个值中从 0 变到 1 的数元位置正好是 Gray 码中被改变了的数元的位置。

（4）请用本书介绍的方法，产生 n 位的 Gray 码，于是又有了一个产生 Gray 码的程序了。

（5）为什么程序中要把 number_of_moves、counter 与 i 定成没有符号的长整数，有什么理由呢？

（6）请通过产生 Gray 码的程序来取代此处的 which_disk()，速度会加快吗？

【参考文献】

河内之塔（Towers of Hanoi）是法国人 M.Claus（Lucas）在 1883 年从泰国带到法国的（越南那个时候是法国属地），河内曾经是越战时北越的首都，现在的胡志明市。有很多人译成“汉诺威塔”是不对的，音虽相差不远，但却显然地误解了 Hanoi 这个字。河内之塔有一个起源的故事，这是 De Parville 在 1884 年讲的。在 Benares 地方有一座神庙，从天神创世起，就在神庙地底、大地的中心处立了 3 根钻石的针，一根针上串了 64 个用金子铸成的圆片，从上到下圆片愈来愈大，神庙的僧侣按照前面提过的方法把这 64 个金圆片搬到第三根钻石针上，一旦工作完成，这些针、圆片、神庙以及这个世界都会随之消失。这段时间有 $2^{64}-1$ 长，亦即 18446744073709551615 步，纵使僧侣毫不犯错，恐怕也得花几百亿年才能搬得完。以上的史料出自下面两本书：

1 W.W.Rouse Ball and H.S.M.Coxeter. *Mathematical Recreations and Essays*. 13th ed., Dover, 1987

2 M.Kraitchik. *Mathematical Recreations*. 2nd ed., Dover, 1953

不用递归法来解河内之塔，曾经是一个很热门的业余问题，文献相当多，本文不打算一一列举，与 Gray 码的联系也陆续地被人们一再“发现”，本书方法是取自 H.Mayer（加州圣地亚哥州立大学教授）与 Don Perkins（美制四年级，等于中国的高一学生）在 1985 年的一篇文章，Mayer 教授指出这是 Perkins 想出来的方法，但他们两位都不曾指出这个方法与 Gray 码的联系，不过 Perkins 不知道 Gray 码是可以预见的。

3 H.Mayer and D.Perkins. Towers of Hanoi Revisited, A Nonrecursive Surprise, *SIGPLAN Notices*. Vol.19(1984), No.2, pp.80～84

这方面的中文文献并不多，前几年笔者在《Micro 随笔》上一连写过 3 个月的介绍，或许可以提供参考。那 3 篇文章入手比较浅，也不曾提到 Gray 码，所以有些地方说得不很清楚，如果现在有机会重写，一定可以做得更好。在那一系列的第三篇中，还有另一个有趣的公式与进一步的参考文献，有兴趣的朋友可以一看：

4 冼镜光．谈河内之塔（一，二，三）——《Micro 随笔》，微电脑时代．1986 年 9

Download at http://www.pin5i.com/

月，pp.170～174；1980 年 10 月，pp.156～163；1986 年 11 月，pp.159～163

【程序】

程序 17-10（HANOI_I.C）

```
/* ------------------------------------------------------------ */
/* PROGRAM towers of hanoi :                                    */
/*   Given the number of disk to be moved, this function        */
/* moves the number of disks from pin 1 to pin 3 subject        */
/* to the restriction of the rules of the towers of Hanoi       */
/* without recursion.  The basic idea behind this program       */
/* is the Gray Code.  But the formulation of this program       */
/* follows from an article of Don Perkins and H. Mayer.         */
/*                                                              */
/* Copyright Ching-Kuang Shene          July/20/1989            */
/* ------------------------------------------------------------ */

#include <stdio.h>
#include <stdlib.h>

#define  MAX_DISK  31

int  which_disk(unsigned long *);

void  main(void)
{
     int  number_of_disk;              /* the number of disks      */
     int  pin[MAX_DISK+1];             /* locations of disks       */
     int  dir[2];                      /* directions; 0=pos,1=neg  */
     int  disk;                        /* disk to be moved         */
     int  next;                        /* next position of 'disk'  */
     int  index;                       /* direction subscript      */
     unsigned long number_of_moves;    /* number of moves          */
     unsigned long counter;            /* counter for Gray Code    */
     unsigned long i;                  /* working                  */
     char line[100];                   /* input line               */

     printf("\nIterative Towers of Hanoi Program");
     printf("\n================================");
     printf("\n\nHow many disks (<=31) ? ");
     gets(line);

     number_of_disk  = atoi(line);
```

Download at http://www.pin5i.com/


```
number_of_moves = (0x01UL << number_of_disk) - 1;
counter = 0;                        /* counter for Gray Code    */

if (number_ofdisk & 0x01)           /* setup direction          */
    dir[0] = 0, dir[1] = 1;
else
    dir[0] = 1, dir[1] = 0;

for (i = 1; i <= number_of_disk; i++)  /* set up loc.           */
    pin[i] = 1;

printf("\n   Step   Disk #    From      To");
printf("\n   ------------------------------");

for (i = 1; i <= number_of_moves; i++) {
    disk  = which_disk(&counter);   /* get disk #               */
    index = disk & 0x01;            /* compute direction index */
    next = (pin[disk] + dir[index]) % 3 + 1;
    printf("\n%6lu%8d%10d%8d", i, disk, pin[disk], next);
    pin[disk] = next;
    }
}


/* ------------------------------------------------------------ */
/* FUNCTION which_disk :                                        */
/*   Given the previous counter value, this function            */
/* computes the only one bit changed from 0 to 1 by adding      */
/* one.  This value corresponding to Gray Code.                 */
/* ------------------------------------------------------------ */

int  which_disk(unsigned long *counter)
{
    unsigned long a, b, c;
    int  i;

    a = *counter;
    *counter = b = a + 1;
    for (c = a^b, i = 0; c != 0; c >>= 1, i++)
        ;
    return i;
}
```

Download at http://www.pin5i.com/

问题 8.10　生命游戏（LIFE.C）

请写一个仿真生命游戏（Game of Life）程序，如果不知道所谓生命游戏，请看下面的说明。

【解答】

首先研究一下如何输入。在输入中指出一共要产生几代，也就是时间的极大值。没有这个值是很危险的，因为在问题说明中的第三与第四这两种情况就无法掌握了。所以输入中的第一列就是 3 个数，第一个数指出要产生多少代（generations），第二个数指出棋盘有多少列（row），第三个数则指出有多少行（column）。接着，输入一个开始，亦即是时间为 0 时的状态。为了方便起见，并不期望使用者输入他的棋盘中每一格的内容。做法是，使用者在输入列上用空格表示机体，用任何非空白的符号代表有机体，当那一列上最后一个有机体输入完后，就可以结束那一列，而且他也不必指出有多少列，一直读取输入，读到多少列，就算是有那么多列。当然，使用者要留神，列数与行数不能超过他要求的棋盘大小。下面就是一个输入的例子，如图 17-24 所示。

这个输入指出要产生 10 个世代，棋盘有 8 列、10 行，他所输入的第 0 代内容如图 17-25 所示。

```
10  8  10
 * * *
* * **
 * ** *
*** *
```

图　17-24

不过，把使用者的输入永远顶着左上角放可不是什么好主意，所以打算为使用者多做一点事，把他的输入移到他要求的棋盘的中央去，如图 17-26 所示。

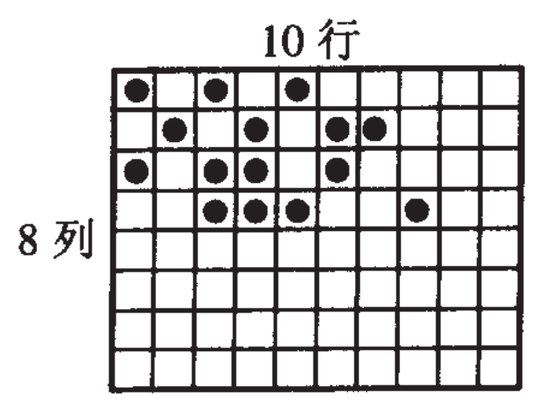


图　17-25

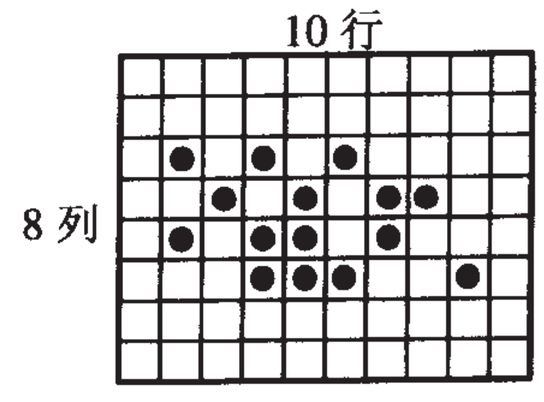


图　17-26

因为在输入中，第 0 世代只用了 4 列 8 行，但棋盘大小有 8 列 10 行，所以往右移一行，往下移两列就是中央位置。

在程序 LIFE.C 中函数 read_in()就是参照这个观点写成的。第一组 for 循环把棋盘 cell[ ][ ] 清成空无一物，第二组 for 循环读入第 0 世代的每一列，max_col 用来找出到目前为止读到

Download at http://www.pin5i.com/

的最长一列的长度，那正好就是使用者输入的最大宽度。max_row 则用来计算一共读了多少列。因此第二组 for 做完之后，存放在 cell[ ][ ]中的就是图 17-25。第三组 for 则是用来把输入移到中央。因为棋盘有 row 列，但输入有 max_row 列，所以棋盘就有 row-max_row 列没有填满，这些列都在下方，所以把输入的内容向下移(row-max_row)/2 列，就可以移到列的中央。同样的理由，向右移(column-max_col)/2 就移到行的中央。注意，移动是从右下角起，从右而左，然后一列接一列地往上处理的，因为只有这样才不会毁掉原有的资料。内循环有两个 for，第一个 for 向右下角移动输入数据，第二个 for 则把前头空出来的地方清除掉。同理，当所有资料都搬完之后，上方也有空列，也一并清除才行，这就是 read_in()的运行，不妨执行程序，将 LIFE.IN1、LIFE.IN2 与 LIFE.IN3 作为输入，看看能否了解它的运行。

display()是用来显示的，很简单，所以就不多作说明。

接下来介绍程序的核心，也就是仿真生命游戏的函数 game_of_life()。注意，没有办法在原来的 cell[ ][ ]矩阵中产生下一个世代，因为这会毁掉这一个世代的状态，所以折衷（偷懒）的做法是，弄一个一样大小的工作用矩阵 workcopy[ ][ ]出来。在一开始时把 cell[ ][ ]的内容抄过去，因此 workcopy[ ][ ]永远是这一个世代，而 cell[ ][ ]则用来存放下一个世代的内容。另外，也想掌握下一个世代有机体是否都死光，或者是进入稳定状态，所以另外准备了两个变量——cell_count 用来记录下一个世代有多少个活着的有机体，而 stable 则记录着下一个世代的内容是否与这个世代的完全一样。

程序中最外面的 for 循环让程序反复做 generations 次，除非因为下一个世代有机体都死光，或者是进入稳定状态（于是 done 就会被定成 YES）。第二与第三层的 for 就检查在这个世代中的每一个格，workcopy[i][j]，算出它有多少个邻居，存入 neighbors。如何算出第 i 列第 j 行有多少个邻居呢？可以参看图 17-27。

第 i 列第 j 行的格子，连自己在内，一共有 9 格，左、右、上、下的坐标分别是 j-1，j+1，i-1 与 i+1，当然这些坐标有可能会越出边界，如果越出上方或左方，就定成 0，但若越出右方或下方，就定成 column-1 与 row-1，它们正是最右边一行与最下方一列。当找出左、右、上、下的坐标之后，就存放在 left、right、top 与 bottom 中，接着最内层的两个 for 循环就把 workcopy[ ][ ]中的值加到变量 neighbors 中。注意，程序一开始的#define，把代表有机体的值 OCCUPIED 定成为 1，而把代表无有机体的值 UNOCCUPIED 定成 0。所以 neighbors 中的值正好是有有机体的格子数的总和，由此可以发现适当地定义代表值是一件很重要的化简程序写作的工具。接下来的连串 if-then 就是很简单的了，它们反映出问题说明中的 4 个出生或死亡的条件。最后，看看如何算 stable，这个代表是否进入稳定状态的值。如果对所有 workcopy[i][j] 与 cell[i][j]而言，这两者都相等，那么它们与 stable 经过&&运算之后，stable 都可以维持在 YES 的状态，一旦有某一对 cell[i][j]与 workcopy[i][j]不相等，就会把 stable 变成假，因此下一个世代如果与这一个世代相同，stable 的值就是 YES。注意，此例的&&不可以合写成&&=，因为 ANSI C 中没有这个运算。

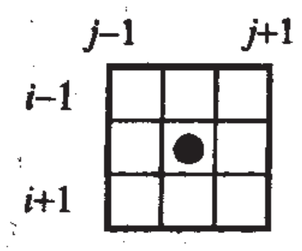


图 17-27

当所有 workcopy[ ][ ]都查完了，就到达最后面的 if…else，看看是否所有有机体都死光，还是进入了稳定状态，如果真是如此，就不必继续做了，令 done 为 YES，就可以脱离最外

Download at http://www.pin5i.com/

面的循环。如果不是这两者，就显示结果，返回准备产生下一个世代。

主程序本身无事可做，只不过是调用 read_in()读取数据，再通过函数 game_of_ life()运行而已。

因此，这个程序其实并不难，只是复杂些而已，对程序的解说也仔细些，希望对读者有所帮助。

【习题】

（1）用两个完全相同大小的矩阵 workcopy[ ][ ]与 cell[ ][ ]，不是太浪费了吗？请问有没有降低内存使用量，但却不使程序复杂的办法？

（2）如果使用 PC，那么就会有比较顺手的对屏幕的控制功能。请改写这个程序，把整个棋盘以文字模式或绘图模式在屏幕显示，这样就会看到更直接的、有动感的结果。另外，把对 generations 的输入去掉，通过显示完一个世代，让使用者按某个键的方式而停止运行，不然就一直反复运行。

（3）如果做了第（2）题，请再度扩充它，这一次不必输入了，用随机数产生一组有机体的位置，它的效果就会更有趣。

（4）续题（3），在程序中加上年龄的效果。不妨把每一个有机体的年龄记下来，于是不同年龄就用不同颜色，当有机体的年龄大过某个值时就一直维持颜色不变。这个程序就会提供很漂亮的随年龄而异的效果。

（5）续题（4），如果一个有机体有老化现象，到了某个年龄就会死亡，因而生命游戏又多了一条规则，试试看多了这个条件时又有什么效果？

【参考文献】

生命游戏是一个流传很广的程序习作，它原来是由 John H.Conway 构想出来的，不过 Conway 的原意却并不是一个游戏，虽然名叫 Game，但它实在有更进一层的含意。现在只知道用计算机来仿真这个游戏，但事实上这个游戏的威力却等于一部计算机，因为 Conway 证明了通过对有机体的组织方式，可以用生命游戏反过来仿真电子线路中的 3 个基本运算 OR、AND 与 NOT。正因为计算机的威力来自这 3 个运算的复杂结合，所以生命游戏也就可以达成计算机所能做的事，因此生命游戏的威力，在理论上与所使用的计算机完全相同。讨论生命游戏的著作不多，但若有兴趣，下面列出来的书不可不读，它不但有趣，而且有意义，值得推荐。

E.R.Berlekamp, J.H.Conway and R.K.Guy. *Winning Ways, Vol.2*: Games in Particular, Academic Press. 1982

【程序】

程序 17-11（LIFE.C）

```
/* ------------------------------------------------------------------ */
/* PROGRAM  game of life :                                            */
/*   This is a finite implementation of John H. Conway's              */
/* Game of Life.  Refere to my book for detail please.                */
/*                                                                    */
```

Download at http://www.pin5i.com/


```
/* Copyright Ching-Kuang Shene          July/25/1989              */
/* ------------------------------------------------               */

#include <stdio.h>
#include <stdlib.h>

#define  MAXSIZE     50              /* board size                */
#define  OCCUPIED     1              /* occupied flag             */
#define  UNOCCUPIED   0
#define  YES          1
#define  NO           0

char cell[MAXSIZE][MAXSIZE];         /* the board                 */
char workcopy[MAXSIZE][MAXSIZE];     /* a working copy            */
int  row;                            /* No. of rows you want      */
int  column;                         /* no.of columns you want    */
int  generations;                    /* maximum no.of generation*/

/* ------------------------------------------------               */
/* FUNCTION  read_in :                                            */
/*   This function reads in the number of generations,            */
/* the number of rows, the number of columns and finally          */
/* the initial configuration (the generation 0).  Then put        */
/* your configuration to the center of the board.                 */
/* ------------------------------------------------               */

void  read_in(void)
{
     int max_row, max_col;           /* max # of row and col.     */
     int col_gap, row_gap;           /* incremnet of row and col*/
     int i, j;
     char line[100];

     gets(line);                     /* read in gens,row and col*/
     sscanf(line, "%d%d%d", &generations, &row, &column);
     for (i = 0; i < row; i++)       /* clear the board           */
         for (j = 0; j < column; j++)
             cell[i][j] = UNOCCUPIED;
     max_col = 0;                    /* read in the config.       */
     for (max_row = 0; gets(line) != NULL; max_row++) {
         for (i = 0; line[i] != '\0'; i++)
             if (line[i] != ' ')
```

Download at http://www.pin5i.com/


```
                    cell[max_row][i] = OCCUPIED;
            max_col = (max_col < i) ? i : max_col;
        }

        row_gap = (row - max_row)/2;    /* the moving gap          */
        col_gap = (column - max_col)/2;
        for (i = max_row + row_gap - 1; i >= row_gap; i--) {
            for (j = max_col + col_gap - 1; j >= col_gap; j--)
                cell[i][j] = cell[i-row_gap][j-col_gap];
            for ( ; j >= 0; j--)
                cell[i][j] = UNOCCUPIED;
        }
        for ( ; i >= 0; i--)
            for (j = 0; j < column; j++)
                cell[i][j] = UNOCCUPIED;
    }

    /* ----------------------------------------------------------- */
    /* FUNCTION  display :                                         */
    /*   Display the board.                                        */
    /* ----------------------------------------------------------- */

    #define   DRAW_BOARDER(n)    {   int i;                        \
                                     printf("\n+");               \
                                     for (i = 0; i < n; i++)      \
                                         printf("-");             \
                                     printf("+");                 \
                                 }
    void  display(int gen_no)
    {
        int  i, j;

        if (gen_no == 0)
            printf("\n\nInitial Generation :\n");
        else
            printf("\n\nGeneration %d :\n", gen_no);

        DRAW_BOARDER(column);
        for (i = 0; i < row; i++) {
            printf("\n|");
            for (j = 0; j < column; j++)
                printf("%c", (cell[i][j] == OCCUPIED) ? '*' : ' ');
```

Download at http://www.pin5i.com/


```
        printf("|");
    }
    DRAW_BOARDER(column);
}

/* ----------------------------------------------------------- */
/* FUNCTION  game_of_life :                                    */
/*   This is the main function of Game of Life.                */
/* ----------------------------------------------------------- */

void  game_of_life(void)
{
    int  stable;                    /* stable flag             */
    int  iter;                      /* iteration count         */
    int  top, bottom, left, right;  /* neighborhood bound      */
    int  neighbors;                 /* # of neighbors          */
    int  cell_count;                /* # of cells count        */
    int  done;
    int  i, j, p, q;
    display(0);                     /* display initial config. */
    done = NO;
    for (iter = 1; iter <= generations && !done; iter++) {
        memmove(workcopy, cell, MAXSIZE*MAXSIZE);  /*copy      */
        stable  = YES;              /* assume it is in stable  */
        cell_count  = 0;            /* # of survived cells = 0 */
        for (i = 0; i < row; i++) {/* scan each cell...        */
            top = (i == 0) ? 0 : i - 1;
            bottom  = (i == row - 1) ? row-1 : i + 1;
            for (j = 0; j < column; j++) {
                left  = (j == 0) ? 0 : j - 1;
                right = (j == column - 1) ? column-1 : j + 1;

                /* compute number of neighbors                 */

                neighbors = 0;
                for (p = top; p <= bottom; p++)
                    for (q = left; q <= right; q++)
                        neighbors += workcopy[p][q];
                neighbors -= workcopy[i][j];

                /* determine life or dead                      */
```

Download at http://www.pin5i.com/


```
                    if (workcopy[i][j] == OCCUPIED)
                        if (neighbors == 2 || neighbors == 3) {
                            cell[i][j] = OCCUPIED;
                            cell_count++;
                        }
                        else
                            cell[i][j] = UNOCCUPIED;
                    else if (neighbors == 3) {
                        cell[i][j] = OCCUPIED;
                        cell_count++;
                    }
                    else
                        cell[i][j] = UNOCCUPIED;
                    stable = stable && (workcopy[i][j] == cell[i][j]);
                }
            }
            if (cell_count == 0) {
                printf("\n\nAll cells die out.");
                done = YES;
            }
            else if (stable) {
                printf("\n\nSystem enters a stable state.");
                done = YES;
            }
            else
                display(iter);
        }
    }

    /* ----------------------------------------------------------------- */

    void  main(void)
    {
        read_in();
        game_of_life();
    }
```

Download at http://www.pin5i.com/

第18章 终 曲

问题 9.1 等量正负号段落（BALANCE.C）

已知一个数组，其中有正数、0、负数，请写一个函数，找出这个数组中最长的，而且正数与负数个数相同的部分数组的长度。

【解答】

按照问题说明中的方式继续发展，不必记录从第 i 到第 j 个有多少个负数（或正数）。为什么呢？用 n_i 表示数组中从头起到第 i 个元素“多”了多少个负数，也就是说，n_i 是数组中从头起到第 i 个元素中负数的个数减去正数的个数。因此，若 $n_i>0$，就表示从头起到第 i 个元素这一段范围中负数较多，反之 $n_i<0$ 则表示负数较少，但 $n_i=0$ 则是正负数一样多。不过，单单 n_i 一个数于事无补，因为这是从头算起，不过这没关系，如果有两个脚码 i 与 j，i<j，而且 $n_i=n_j$，这表示从头起到第 i 个元素正、负数比正数多出来的个数，与从头起到第 j 个元素止的个数一样多，因此从第 i+1 个元素到第 j 个元素之间（含两端）的正数与负数个数相同。这很容易证明，如果从头起到第 i 个元素（含）的负数与正数的个数分别是 m_i 与 p_i，于是 $n_i=m_i-p_i$，同理 $n_j=m_j-p_j$。如果从第 i+1 个元素到第 j 个元素之间的正数与负数各有 t 与 s 个，于是 $m_j=m_i+t$，$p_j=p_i+s$。由于 $n_i=n_j$，得：

$$
\begin{aligned}
m_i-p_i &= n_i \\
&= n_j \\
&= m_j-p_j \\
&= (m_i+t)-(p_i+s) \\
&= m_i+t-p_i-s
\end{aligned}
$$

就得到了 t=s。

这一说明，当从头起一个接一个元素处理时，可以在第 i 个元素时计算 n_i，如果往后到了某个 j 时发现 n_i 与 n_j 相同，则 i+1 到 j 之间正、负数个数就会一样，这一段区间的长度是 j-(i+1)+1=j-i。因此再多一个变量来管理这些 j-i 的极大值就行了。不过问题是，固然可以用一个数组把 $n_0,n_1,\cdots,n_i$ 这些值保存起来，但是在算出 n_j 之后，要在 $n_0\sim n_{j-1}$ 之间找出一个 n_k 使得 $n_j=n_k$ 而且 j-k 为极大，倒不是一件容易的事。所以，需要换个角度看问题。

注意，因为 $-i\leqslant n_i\leqslant i$（如果从头起到第 i 个元素止全是负数，则 $n_i=i$；但如果全是正数，则 $n_i=-i$），所以可以准备一个有 2n+1 个元素的数组 e，脚码是 $-n,-(n-1),\cdots$

Download at http://www.pin5i.com/

,–3,–2,–1,0,1,2,…,n，此处 n 是元素个数。在 e_k 中储存了 $n_i = k$ 的 i。也就是说，如果做到第 i 个元素时，n_i=k，就把 i 存到 e_k 中。因此，下一个做到某个 j 时又发现 n_j=k，存入 e_k 的就变成 j 了。换句话说，e_k 中的值就指出原数组从头起到该值的脚码为止，负数的个数减去正数的个数就是 k。

于是上一段的问题就不难做了，假设现在做到了第 i 个元素，从第 i 个元素可以算出 n_i，因此上一次出现负数个数减正数个数与 n_i 相等的位置就是第 e_{n_i} 个，现在位置是 i，e_{n_i} ～ i 之间的负数与正数的个数就是相等的，所以问题就解决了。

由上所述，多用了一个 2n+1 个元素的数组，把原数组查一次就得出结果，这是个线性的程序，速度要比问题说明中的所有方法快，因为那些方法都要处理与 n^2 成正比的元素个数。

【程序说明】

依照上面的讨论，下面就是对程序的简单说明。首先，上面提到的 e 数组在 C 语言中是做不到的，因为 C 语言中数组的脚码永远大于等于 0。解决之道就是，从数组的中央算起。程序中 loc 这个指针变量的值是通过 malloc()取得 2n+1 个位置之后的第一个元素的指针，mid 正是这 2n+1 个位置在中央的那一个，它就对应着上面的脚码 0。

变量 min_over、max_over 分别是到目前为止所有 n_i 的极小值与极大值，neg_ over 就是 n_i——负数个数减去正数个数，length 则是到目前为止所能找到的负数个数与正数个数相同的最大范围的长度，在开始时这些内容都不存在，因此全都是 0。另外 loc[mid]，也就是 e_0 的值在开始时也是 0。

接着就查每一个元素，对于 x[i]而言，如果它是负的，neg_over 就多 1，负数个数多了 1；反之，若 x[i]为正，负数个数与正数个数的差就少了 1，neg_over 就减去 1。如果 x[i]是 0，由于 0 非正非负，因此 neg_over 的值就不用改变。

有了 neg_over（亦即 n_i）的新值之后，就存放到 e 数组，亦即 loc[ ]中。如果 neg_over 比在此之前的极小值还小，因此现在的 neg_over 就是新的极小值，在对应着 neg_over 的位置中存入 i+1（注意，不是 i，这是为方便而这么做的，原因是 C 语言中数组从 0 算起，故 i+1 是元素个数，而 i 是该元素的脚码）。反之，若 neg_over 比目前所存储的极大值还大，于是新的极大值就变成 neg_over，而且在对应位置存入 i+1。当然，如果 neg_over 落在 min_over 与 max_over 之间，就不会更新 loc[ ]数组了，而这也正是要特别处理的地方，原因是 neg_over 这个值在此之前就出现过了，看看 loc[neg_over]就可以知道上一次它出现时何在，因此 i+1-loc [neg_over]就是这两个之间的元素个数（亦即区间长度），用这个长度来修正最大的长度值 length。当所有元素都查完后，length 中就是最大的长度值。

【习题】

（1）这个程序只做到了找出最大长度，而无法指出从何处到何处。请修正这个程序，让它可以做到这一点。

（2）有时解答可能不止一个，请修改第（1）题的程序，把所有的解答都找出来，程序还是具有线性的特性吗？

（3）如果程序中把 min_over 与 max_over 这两个变量去掉，而直接用 neg_over 去修改

Download at http://www.pin5i.com/

loc[ ]的内容，看起来会简单得多。请问这是正确的想法吗？为什么？请证明它或找出反例。

（4）更正长度值 length 的程序是放到 if…else 程序之外，请问这与放在 if 以及 else 之内有什么差别？有没有做了多余的工作？

【参考文献】

这个题目取自 M.Rem 的文章，他说这个题目事实上是 W.H.J.Feijen 提供的：

M.Rem, Small Programming Excrcise 12, *Science of Computer Programming*, Vol.7 (1986), pp.87～97

【程序】

程序 18-1（BALANCE.C）

```
/* ------------------------------------------------------------ */
/* FUNCTION  pos_neg_balance :                                  */
/*   Given an integer array x[], this function computes         */
/* the largest section of the array with equal number of        */
/* positive and negative numbers.                               */
/*                                                              */
/* Copyright Ching-Kuang Shene          Aug/10/1989             */
/* ------------------------------------------------------------ */

#include <stdlib.h>                /* for malloc()              */

#define  MAX(x, y)   ((x) >= (y) ? (x) : (y))

int  pos_neg_balance(int x[], int n)
{
     int length;                   /* maximum length so far     */
     int neg_over;                 /* #of neg - #of pos         */
     int min_over;                 /* min of neg_over so far    */
     int max_over;                 /* max of neg_over so far    */
     int mid;                      /* the middle item of work   */
     int i;
     int *loc;                     /* addr. of working array    */

     length      = neg_over = min_over = max_over = 0;
     loc         = (int *) malloc(sizeof(int)*(n+n));
     mid         = n;
     loc[mid]    = 0;              /* #of neg=#of pos at start  */
     for (i = 0; i < n; i++) {     /* for each element in x[]..  */
         if (x[i] < 0)             /* if negative               */
             neg_over++;           /* #of neg increased         */
         else if (x[i] > 0)        /* or if positive            */
```

Download at http://www.pin5i.com/


```
            neg_over--;                    /* #of neg. decreased   */
        if (neg_over < min_over) {         /* update min.          */
            min_over = neg_over;
            loc[mid+min_over] = i + 1;     /* and store it         */
        }
        else if (neg_over > max_over) {/* update max.              */
            max_over = neg_over;
            loc[mid+max_over] = i + 1;
        }
        length = MAX(length, i+1-loc[mid+neg_over]);
    }
    free(loc);
    return length;
}
```

问题 9.2 寻找长方形（RECT.C）

在圆周上有若干个点，已经知道这些点与点之间的弧长，并且依圆弧顺序排列，请写一个程序，找出这些点中有没有 4 个可以围成长方形的。为了方便计算，弧长均为正整数。

【解答】

要直接在圆周上找 4 个点出来，使得可以围成一个长方形的工作比较难做，不过这个问题如果换一个角度来看的话，就会简单很多。

如图 18-1 所示，如果已经有了一个圆内接的长方形，这个长方形的对角线就一定是圆的直径，因此也就等分了圆周长，用图 18-1 来看，会有 a+b=c+d，以及 b+c=d+a 的关系。

在直径两端的点，叫做对极点，或者是直径点（Diametrical Point）。图 18-1 中 A 与 C，B 与 D 都是直径点。因此，只要能够找出两对直径点，它们就可以构成一个长方形了。但如何找出直径点呢？如图 18-2 所示。

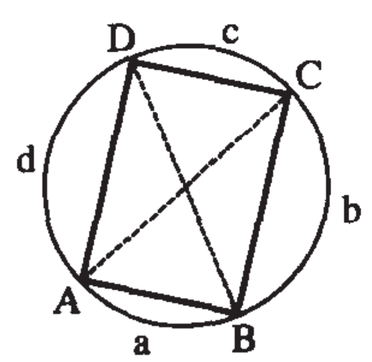


图 18-1

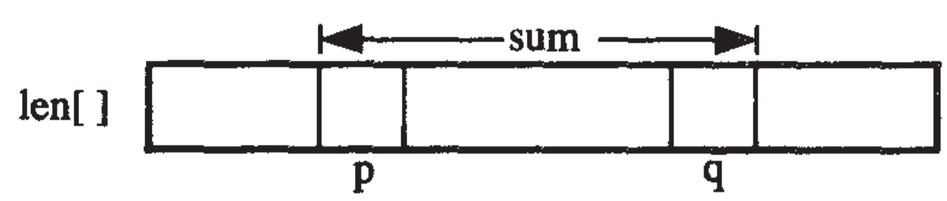


图 18-2

如果弧长是存放在数组 len[ ]中，则 total 表示圆周长（也就是 len[ ]中元素的总和），令 half 是半周长（假设 total 是偶数）。因此，找直径点的工作就等于是要找出一对脚码，p 与 q（p<q），使得 len[p]+len[p+1]+…+len[q]的值正好是 half。因为 half 已知，所以查找 p 与 q 这两个脚码是一件很简单的事，在此之前已经介绍过类似的题目了。

Download at http://www.pin5i.com/

注意一点，len[p]～len[q]的和是半周长，len[q+1]～len[n-1]再加上 len[0]～len[p-1]也正好是半周长，因此可以把 p 与 q 局限在 0≤p≤q<n 这个范围内，n 是圆弧，亦即 len[ ]数组的元素个数。令 p 与 q 从 0 起，而 sum 从 len[0]起，当 sum 的值小于 half 时，就表示 p 与 q 所围住的这一段的值不足半周长，所以在 sum 中加上 len[++q]的值，以求得 sum 更大些，更有机会接近半周长。但若 sum 大于 half，则表示 sum 太大了，很可能是因为上一次把 len[q]加到 sum 中所造成，但不能扣除 len[q]的值，因为这就倒退回头了，只能从 sum 中扣除 len[p++]的值，来符合半周长。于是 p 与 q 向右移动，如果有某个时刻 sum 与 half 相等，count 这个计数变量就加上 1 来记录这件事实。这个操作在什么时候可以停止呢？在 q=n-1 的时候，如图 18-3 所示。

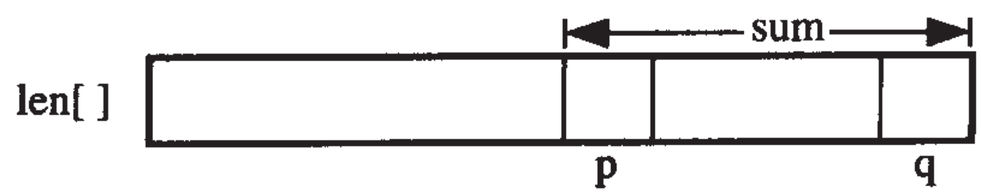


图 18-3

当 q 到达了最后一个元素，p 在它的左方，于是 sum 有 3 种情况：

（1）sum<half：这是可以不必顾虑的，因为 p 与 q 之间不可能再加出个半周长来了。

（2）sum>half：那么是不是还要增加 p，固定 q，然后从 sum 中扣除 len[p]的值呢？也不必，即使在这个过程中会碰到一个 r，使得 len[r]～len[q]是半周长，但这一次却不能算了，因为会在 len[0]～len[r-1]的长度也是半周长时算过一次，换句话说 len[r]～len[q]的长度正好是圆的另一半。

（3）sum==half：道理同上一点。

程序就是用这个观点写成的，不同的地方是把 p 与 q 换成 start 与 end 而已。注意，只计算了有多少组直径点而已，并没有记录下这些点何在。如果函数值大于 1，那就至少有两条对角线，因此这 4 个顶点就是一个矩形，答案就做出来了。

【习题】

（1）请修改程序，报告出所有直径点，有了直径点的名称（或号码），能列出所有可能出现的矩形吗？

（2）如果要问的是，这些点中有没有若干个点可以围成一个内接正 k 边形，k 是输入值。请问这个方法还适用吗？要不要在什么地方修正呢？

（3）请问，这些弧长是否一定要是圆弧？在椭圆上可以吗？能否想出最多的可能性？

【程序】

程序 18-2（RECT.C）

```
/* ------------------------------------------------------------------ */
/* FUNCTION  find_rectangle :                                         */
/*   Given a consecutive length of arcs of a circle, this             */
/* function determines if there exists a rectangle formed             */
/* by these end points of arcs.                                       */
```

Download at http://www.pin5i.com/


```
/*                                                                    */
/* Copyright Ching-Kuang Shene          July/25/1989                  */
/* ----------------------------------------------------------------- */

int  find_rectangle(int len[], int n)
{
     int total;                        /* total arc length            */
     int half;                         /* half = total/2              */
     int sum;                          /* running sum                 */
     int start;                        /* running index #1            */
     int end;                          /* running index #2            */
     int count;                        /* counter of diametrical pt*/
     int i;

     for (total = i = 0; i < n; i++)/* compute total len              */
         total += len[i];
     half    = total/2;                /* and its half                */

     count   = start = end = 0;        /* initialization              */
     sum = len[start];                 /* running sum = 1st arc       */
     while (start < n - 1)
         if (sum < half)               /* if shorter -> increase      */
             sum += len[++start];
         else if (sum > half)          /* if longer -> decrease       */
             sum -= len[end++];
         else                          /* got one and inc.            */
             sum += len[++start], count++;
     return count;
}
```

问题 9.3　多边形的直径（DIAMETER.C）

假设平面上的点 $p_0, p_1, p_2, \cdots, p_{n-1}$ 围成一个多边形，$\overline{p_0p_1}, \overline{p_1p_2}, \cdots, \overline{p_{n-2}p_{n-1}}, \overline{p_{n-1}p_0}$ 为这个 n 边多边形的边。多边形的直径（Diameter）的定义是：找出多边形的两个点 p_i 与 p_j，$0 \leqslant i < j < n$，使得 $\overline{p_ip_{i+1}} + \overline{p_{i+1}p_{i+2}} + \cdots \overline{p_{j-1}p_j}$ 与 $\overline{p_jp_{j+1}} + \cdots + \overline{p_{n-2}p_{n-1}} + \overline{p_{n-1}p_0} + \overline{p_0p_1} + \cdots + \overline{p_{i-1}p_i}$ 的差的绝对值为最小，于是 p_i 与 p_j 之间的连接就是该多边形的直径。当然，直径可能不止一条。把 $\overline{p_0p_1}, \overline{p_1p_2}, \cdots, \overline{p_{n-2}p_{n-1}}, \overline{p_{n-1}p_0}$ 的长度记成 $d_0, d_1, \cdots, d_{n-1}$ 等，找出两个点 i 与 j，$0 \leqslant i < j < n$，使得 p_i 与 p_j 的连接为该多边形的直径。

Download at http://www.pin5i.com/

【解答】

用一个图来说明会简便些。如图 18-4 所示，用 s 与 e 指出到目前为止所找到的两个顶点，在它们之间的各边的和与另一半的差的绝对值是个极小值。

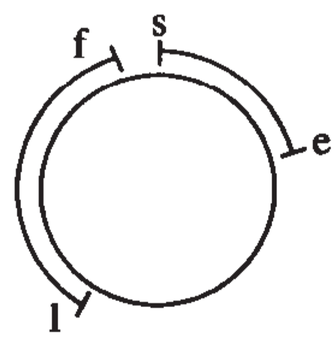


图 18-4

再用 l 与 f 表示目前正在考虑的一段。依照问题说明中的做法，把 $p_i \sim p_{i+1}$ 的边的长度写为 d_i，而 $p_{n-1} \sim p_0$ 的长度为 d_{n-1}，因此顶点 f 到顶点 l 这一段的长度就是 $d_f + d_{f+1} + \cdots + d_{l-1}$，为了方便起见用 $D_{f,l}$ 来表示。周长中剩下来的部分为 $d_l + d_{l+1} + \cdots + d_{n-1} + d_0 + d_1 + \cdots + d_{f-1}$，写成 $D_{l,f}$，于是 $D_{f,l} + D_{l,f} = T$，T 是周长。有了上述表示方法之后，所要做的事就是，找出一对 s 与 e，使得 $|D_{e,s} - D_{s,e}|$ 有极小值。正因为如此，免不了要检查 $D_{f,l} - D_{l,f}$ 值，$f \leqslant l$。

看看下面的演算：

$$\begin{aligned} D_{f,l} - D_{l,f} &= D_{f,l} - D_{l,f} + D_{f,l} - D_{f,l} \\ &= 2D_{f,l} - (D_{l,f} + D_{f,l}) \\ &= 2D_{f,l} - T \end{aligned}$$

求 $|D_{f,l} - D_{l,f}|$ 的极小值，就相当于求 $|2D_{f,l} - T|$ 的极小值。后者是比较容易做的，只有一个会变动的量，亦即 $D_{f,l}$。$|2D_{f,l} - T|$ 最大值会是多少呢？很简单，就是 T，原因是当 $D_{f,l} = 0$ 时，就得到这个极大值。当然 $|2D_{f,l} - T|$ 的极小值“可能”会是 0，但不应该这么假设。在开始时，因为 f 与 l 在同一个顶点上，$D_{f,l} = 0$，因此 $2D_{f,l} - T$ 就是 -T。于是在以后的工作中，就变动 f 与 l 的位置而希望降低 $|2D_{f,l} - T|$ 的值。

在执行过程中，如果发现 $2D_{f,l} - T \geqslant 0$，就表示 $D_{f,l}$（也就是从 f 算到 l）可能太大了，把 f 向前移，因此新的 $D_{f,l}$ 的值就相当于 $D_{f+1,l}$，于是：

$$\begin{aligned} 2D_{f,l} - T &= (2d_f + D_{f+1,l}) - T \\ &= 2d_f + 2D_{f+1,l} - T \end{aligned}$$

新的 $2D_{f+1,l} - T = (2D_{f,l} - T) - 2d_f$，亦即从 $2D_{f,l} - T$ 中扣除两个 $2d_f$ 就可以得到 $2D_{f+1,l} - T$。反过来，如果在执行过程中发现 $2D_{f,l} - T \leqslant 0$，就表示 $D_{f,l}$ 太小，在 l 这一端加上 d_l，希望可以增加 $D_{f,l}$ 的值来使 $2D_{f,l} - T$ 的值更靠近 0。这个新的 $2D_{f,l+1} - T$ 值是：

$$\begin{aligned} 2D_{f,l+1} - T &= 2(D_{f,l} + d_e) - T \\ &= (2D_{f,l} - T) + 2d_l \end{aligned}$$

亦即把 $2D_{f,l} - T$ 加上两倍的 d_l 就行了。

程序中不过是一直令 l 与 f 递增，每次 $2D_{f,l} - T$ 的值改变了，就可以求出 $|2D_{f,l} - T|$，拿它与在此之前记住的极小值进行比较，如果 $|2D_{f,l} - T|$ 比较小，就记住这个新的极小值，

Download at http://www.pin5i.com/

以及 f 与 l 这两个顶点，因此程序结束之后，记下来的两个顶点就是在直径上的两个顶点了。程序该在什么时候停止呢？当 p_{n-1} 处理完后就可以停下来了，当 l 穿越 p_{n-1} 而到达 p_0，由此 $D_{l,f}$ 的值就可以用 $D_{l,f}$ 来替代，但这早就是处理过的了。

【程序说明】

程序中，一开始就先算出 $-T$，存放在变量 diff 中，其实 diff 所存的值就是上面的 $2D_{f,l}-T$。$|2D_{f,l}-T|$ 则存放在 abs_diff 中，在一开始时 abs_diff 与目前所记住的极小值（current_min）都是 T。接着程序中惟一的 for 循环就一个顶点接一个顶点地查过去。在此，f 与 l 用 follow 与 lead 变量取代，而极小值的顶点位置则放在 end 与 start 中，这正是图 18-4 中的 e 与 s。当 $l<n$ 时，用第一个 if 来更新极小值的值，接着如果 $2D_{f,l}-T$ 大于等于 0（diff >=0），就从 diff 中扣除两倍的 d_f（亦即 len[follow]），并且把 f 往前移。同理，当 $2D_{f,l}-T$ 小于 0 时，加上两倍的 d_l（亦即 len[lead]）来使 $2D_{f,l}-T$ 更靠近 0。最后再把新的 diff 的绝对值求出，存放在 abs_diff 中。最后，程序返回记下来的两个会造成 $|2D_{f,l}-T|$ 为极小值的顶点。事实上，这样的相互追逐的技巧在前面的例题中已经出现不少次了，希望再多看一题之后能够有进一步的认识。

【习题】

（1）请问程序（在最坏情况下）一共处理了多少个点？

（2）用 $2D_{f,l}-T$ 来简化计算工作，但是直接使用 $D_{f,l}-D_{l,f}$ 却也不见得十分复杂，请把程序稍作修改，用 $D_{f,l}-D_{l,f}$ 操作。

（3）题目是给了边长，但若给的是各个顶点的坐标，能够把程序稍作修饰来完成工作并且求出直径的长度吗？

（4）程序只要求找出一个直径，但若是改为要找所有直径，亦即所有令 $|2D_{f,l}-T|$ 为极小值的 f 与 l，要如何处理？程序的效率会降低吗？有没有办法在不降低效率的大前提下解决这个问题呢？

【参考文献】

这个问题要解决的技巧取自 M.Rem 的专栏：

M.Rem. Small programming Exercise 14, *Science of Computer Programming*, Vol.7 (1986), pp.313～319

【程序】

程序 18-3（DIAMETER.C）

```
/* ------------------------------------------------------------ */
/* FUNCTION  diameter :                                         */
/*   Given the length of each side of a polygon, this           */
/* function determines its diameter.  The diameter of a         */
/* polygon is the line segment joining from i to j, i < j,      */
/* such that absolute difference of d[i]+d[i+1]+...+d[j]        */
/* and d[j]+...+d[n-1]+d[0]+...+d[i-1] is minimal.              */
```

Download at http://www.pin5i.com/


```
/*                                                                    */
/* Copyright Ching-Kuang Shene          Sept/01/1989                 */
/* ------------------------------------------------------------       */

#include  <stdlib.h>

void  diameter(int len[], int n, int *first, int *last)
{
     int start, end;              /* [end,start] is the min          */
     int lead, follow;            /* [follow,lead]-running min       */
     int diff, abs_diff;          /* diff and absolute diff          */
     int current_min;             /* current min - [end,start]       */
     int i;

     for (i = diff = 0; i < n; diff -= len[i], i++)
         ;                        /* compute max. diff               */

     abs_diff = current_min = -diff;
     for (lead = follow = start = end = 0; lead < n; ) {
         if (current_min > = abs_diff) {/* update min                */
             current_min = abs_diff;
             start = lead;
             end = follow;
         }
         if (diff >= 0)                /* if diff > 0 ...            */
             diff -= 2*len[follow++];/* too big, dec.                */
         else                          /* if diff < 0 ...            */
             diff += 2*len[lead++];  /* too small, inc               */
         abs_diff = abs(diff);         /* update absolute diff       */
     }
     *first = end;                     /* return point number, NOT*/
     *last  = start;                   /* side number.               */
}
```

问题 9.4　机器人旋转角度（TURNS.C）

平面上有 n 个点 $p_1, p_2, \cdots, p_n$，有一个机器人目前放在 p_1 上，而且面向 X 轴的正方向，这个机器人会从 p_1 出发，沿途依次经过 $p_1, p_2, \cdots, p_n$ 这些点，最后回到 p_i 而且面向 X 轴的正方向。当机器人到达某个点 p_i 时，它会以逆时针方向旋转，使自己面对 p_{i+1}，再沿直线方向从 p_i 走到 p_{i+1}。写一个程序，读入 $p_1, p_2, \cdots, p_n$ 的坐标值，求出这个机器人总共做了

Download at http://www.pin5i.com/

多少次 360° 的旋转。

【解答】

这个题目并没有表面上看来那么困难，事实上还相当简单。回头看看问题说明中的图，把 $p_1 \sim p_2$、$p_2 \sim p_3$、$p_3 \sim p_4$、$p_4 \sim p_1$ 这 4 个向量画在原点上就得到图 18-5，在图中这 4 根向量就依次叫做 $\vec{u}_1$，$\vec{u}_2$，$\vec{u}_3$ 与 $\vec{u}_4$。

机器人首先从 X 轴转成 $\vec{u}_1$ 方向，再从 $\vec{u}_1$ 转成 $\vec{u}_2$，第三次从 $\vec{u}_2$ 转到 $\vec{u}_3$，第四次从 $\vec{u}_3$ 转到 $\vec{u}_4$，最后，从 $\vec{u}_4$ 转回 X 轴方向。很明显地，可以看到一共转了两圈。事实上，转了多少圈，就相当于在旋转时跨过多少次正向的 X 轴。由图 18-5 可以看出 $\vec{u}_3$ 到 $\vec{u}_4$ 是第一次，而 $\vec{u}_4$ 回到 X 轴是第二次。所以，只要能够算出一共跨越了多少次 X 轴就行了。

虽然不知道两个向量的夹角，但是这些向量说明不少数据。$\vec{u}_1$ 在第一象限，$\vec{u}_2$ 在第三象限，因此从 $\vec{u}_1$ 转到 $\vec{u}_2$ 最多是 180°，而 $\vec{u}_3$ 在第四象限，$\vec{u}_1$、$\vec{u}_2$、$\vec{u}_3$ 都不会造成跨越正向 X 轴，但 $\vec{u}_4$ 却在第三象限，这表示在第四象限的 $\vec{u}_3$ 要转成在第三象限的 $\vec{u}_4$ 那就一定得跨越正向 X 轴。同理从 $\vec{u}_4$ 回到正向 X 轴，就是完成最后的一个 360°。

如果把 $p_{i+1} \to p_i$ 定成向量 $\vec{u}_i$，而令 $p_0 \to p_n$ 为向量 $\vec{u}_n$，于是从 $p_1, p_2, \cdots\cdots, p_n$ 就得到一组机器人每次面对的方向向量了，它们依次是 X 轴，$\vec{u}_1, \vec{u}_2, \cdots\cdots, \vec{u}_2$，X 轴，从 $\vec{u}_i$ 与 $\vec{u}_{i+1}$ 就可以知道自 $\vec{u}_i$ 逆时针转到 $\vec{u}_{i+1}$ 有没有跨越正向 X 轴。首先可以从 $\vec{u}_i$ 找出它在那一个象限（如果 $\vec{u}_i = <x,y>$，于是从 x 与 y 的正负关系就可以决定 $\vec{u}_i$ 在哪一个象限=；如果 $\vec{u}_i$ 与 $\vec{u}_{i+1}$ 分别在第 q_i 与第 q_{i+1} 象限，于是当 $q_{i+1} > q_i$ 时就一定不会跨越正向 X 轴，只要想想从第一到第三象限、第一到第四象限、第二到第三象限、……就不难明白。但是，当 $q_{i+1} < q_i$ 时，就一定跨越了正向 X 轴。为什么呢？如果 $q_i = 3$，$q_{i+1} = 1$ 或 2，看从第三象限到第一或第二象限能否跨越正向 X 轴吗？剩下来比较麻烦的就是 $q_{i+1} = q_i$，这有两个可能性，如图 18-6 所示。

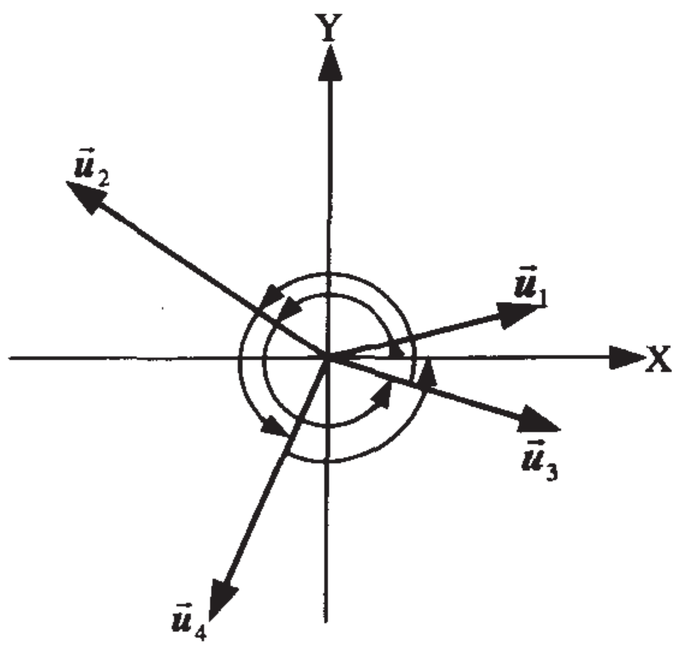


图 18-5

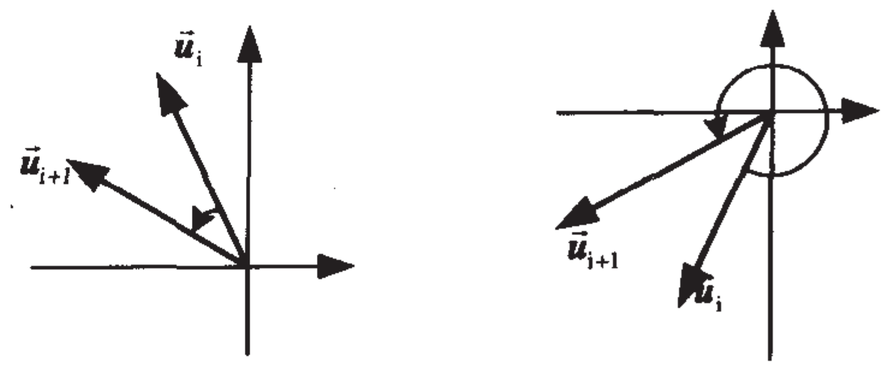


图 18-6

左边的情形是从 $\vec{u}_i$ 转到 $\vec{u}_{i+1}$ 不足 90°，右边的情况是大于 270°；注意，不可能大过 360°（为什么？）。所以，当发现 $\vec{u}_i$ 与 $\vec{u}_{i+1}$ 在同一象限时，惟一要做的事，就是分辨出是第一种情况还是第二种情况，在前者自然不会跨越正向 X 轴，但后者却一定会跨越。

要决定是哪一种情况，看看 $\vec{u}_i$ 与 $\vec{u}_{i+1}$ 的斜率就行了。在第一种情况下，$\vec{u}_i$ 的斜率小于 $\vec{u}_{i+1}$，但是第二种情况则正好相反，$\vec{u}_{i+1}$ 的斜率比 $\vec{u}_i$ 的大（为什么？）。综合起来，从头起处理 $\vec{u}_1, \vec{u}_2, \cdots, \vec{u}_n$ 等，在处理 $\vec{u}_i$ 时，找出它在哪一个象限，令为 q_i，再与 $\vec{u}_{i+1}$ 所在的象限 q_{i+1}

Download at http://www.pin5i.com/

相比较。若 $q_i > q_{i-1}$，表示没有跨越正向 X 轴；若 $q_i < q_{i+1}$，表示跨越正向 X 轴，因此次数就多了一次；若 $q_i = q_{i+1}$，就检查 $\vec{u}_i$ 与 $\vec{u}_{i+1}$ 的斜率，若前者较小，则没有跨越，若前者较大，跨越次数则多了 1，但斜率不会相等（为什么？）。因此当所有向量都处理完后，就有了跨越的总次数了（至少是 1）。

不过在计算上还可以稍许简化。第一是如何决定 $\vec{u}_i$ 在哪一个象限，第二则是如何使用斜率。对于第一个问题而言，固然可以用一连串的比较，用 $\vec{u}_i$ 的两个分量的正、负值来决定它究竟在哪一个象限，但下面的方法却也是很方便的。见图 18-7，有一个向量 $\vec{u}$，要决定它在哪一个象限。

对于 $\vec{u}$ 而言，可以把它顺时针方向旋转 90°，看看多少次能够把它放到第一象限去，亦即第一个坐标为正，第二个坐标不小于 0。如果 $\vec{u} = <x,y>$，那么顺时针方向转 90° 时，结果为 $<y,-x>$，从图 18-7 不难看出来，反复这个过程，看看转多少次能使两个分量都不小于 0 就行了。若是转 3 次，原来就在第四象限，转两次或一次时则原来分别在第三与第二象限，换言之，若转了 t 次，原来就在第 t+1 象限。例如，如果原来有 $\vec{u} = <1,-2>$，转第一次得到 $<-2,-1>$，第二次是 $<-1,2>$，第三次是 $<2,1>$，因此 $\vec{u}$ 在第四象限，如图 18-8 所示。

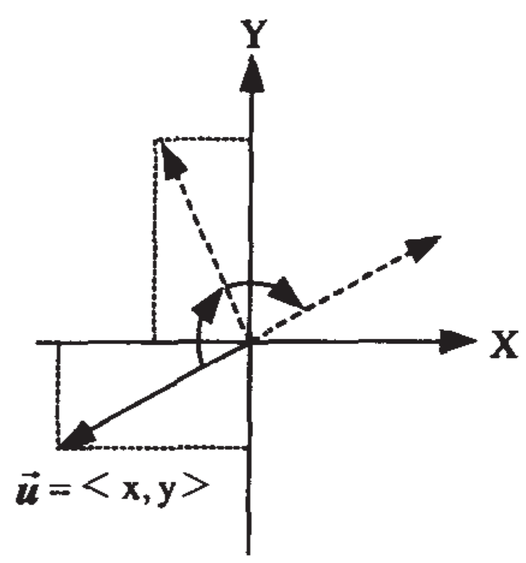


图 18-7

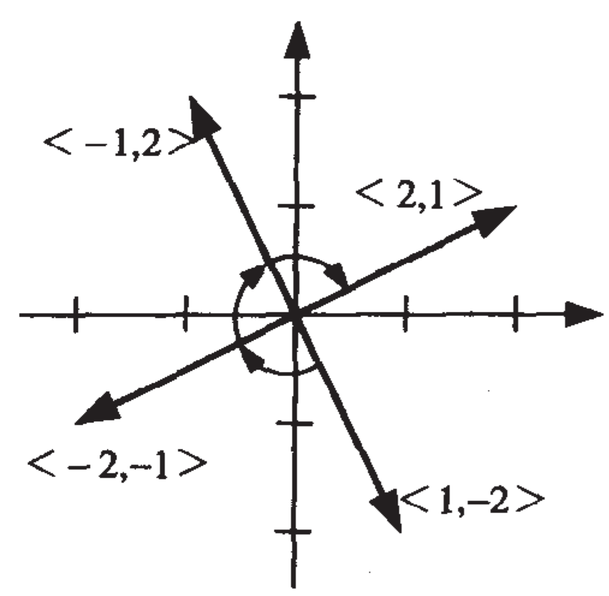


图 18-8

所以，求所在象限就有了一个比较规律的技巧了。

再看第二个问题，如何利用斜率。注意一点，如果 $\vec{u}$ 与 $\vec{v}$ 两个向量在同一个象限，而且 $\vec{u}$ 的斜率比 $\vec{v}$ 的小，那么在做顺时针方向旋转时并不会影响到斜率之间的关系。如果 $\vec{u} = <a,b>$，$\vec{v} = <x,y>$，于是 $\vec{u}$ 与 $\vec{v}$ 的斜率分别是 b/a 与 y/x；$\vec{u}$ 与 $\vec{v}$ 被顺时针地转 90° 之后变成 $<b,-a>$ 与 $<y,-x>$，向量的斜率是 $-a/b$ 与 $-x/y$。若原来 $\vec{u}$ 的斜率比较小，亦即 $b/a < y/x$，于是乘上 -1 得到 $-b/a > -y/x$，再把分式颠倒就得到 $-a/b < -x/y$，所以旋转 90° 之后原来较小的斜率还是比较小。这个结果有什么用？会把 $\vec{u}$ 一直转到第一象限，而通过旋转次数来定出 $\vec{u}$ 原来在哪一个象限。如果 $\vec{u}_i$ 与 $\vec{u}_{i+1}$ 被转到第一个象限之后（$\vec{u}_i$ 与 $\vec{u}_{i+1}$ 在同一个象限），用最后结果计算斜率即可，而不必保留 $\vec{u}_i$ 与 $\vec{u}_{i+1}$ 原来的斜率值。如果转到第一象限后，$\vec{u}_i$ 与 $\vec{u}_{i+1}$ 变成 $<a,b>$ 与 $<x,y>$，于是比较 b/a 与 y/x 就行了，亦即看看 $b/a > y/x$ 是否成立。但 a 与 x 很有可能是 0，所以不能用 $b/a > y/x$，而需乘开再比较 $xb > ay$。

到此为止，已经有了所有应该知道的技巧，能写出程序吗？

Download at http://www.pin5i.com/

【程序说明】

这个程序是 TURNS.C。用 X 与 Y 表示上一个向量经过若干次 90° 的旋转后到达第一象限的横坐标与纵坐标，TURNS 则是一共转了多少次，依照前面所提的，TURNS+1 就是原来所在的象限，new_x、new_y、new_turns 则是正在处理的向量在第一象限中的横坐标、纵坐标与旋转次数。count 是用来记录有多少次跨越正向 X 轴的。

接下来的 for 循环就一个点接一个点地查过去，把 new_x 与 new_y 算出来，而且先把 new_turns 定成 0。在内部的 while 循环则把 new_x、new_y 做顺时针的 90° 旋转，new_turns 顺便记录下转了多少圈，当 new_x 与 new_y 被转到第一象限之后 while 循环就会结束。下方的 if 就开始检查 new_x 与 new_y 有没有跨越正向 X 轴。这有两个可能，第一是 new_x 与 new_y 原来所在的象限数比上一个的小（TURNS＞new_turns）；第二是象限一样，但 new_x 与 new_y 的斜率较小。如果两个条件都成立，于是跨越次数就多了 1。最后现在的 new_x、new_y 与 new_turns 就变成下一个算的“上一个”点了，接着回头处理下一个点。当所有点都处理完后，函数返回 count 的值，这就是跨越的次数。

【习题】

（1）在程序中的 for 循环中为什么是 i<=n，而不是 i<n 呢？

（2）为什么当 $\vec{u}_i$ 与 $\vec{u}_{i+1}$ 在同一象限，若 $\vec{u}_i$ 的斜率大于 $\vec{u}_{i+1}$ 的斜率时，就一定会跨越正向 X 轴？

（3）为什么机器人在任何一点的旋转角度都不可能大于 360° 。

（4）如果一个向量＜x, y＞经过顺时针方向旋转 90° 之后，结果为＜y,−x＞，请证明。

（5）如果机器人是以顺时针方向在 p_i 点旋转面对 p_{i+1} 的话，程序要有什么变动？请完成它。

【参考文献】

常常会以为一个难的程序一定会很长，但是本书中很多例子都给出了反证，此处又是一个例子。程序其实十分短，但若不明了其中的涵义，就很难读懂它，希望这个程序再次提醒，难的程序不必长，长的程序却不一定难。

本题的观点与解法取自 M.Rem 的专栏。

M.Rem, Small Programming Exercise 3, *Science of Computer Programming*, Vol.4(1984), pp.87～94

【程序】

程序 18-4（TURNS.C）

```
/* ------------------------------------------------------------------ */
/* FUNCTION  turns :                                                  */
/*   Given a sequence of points on the plane and a robot              */
/* at the first point facing the X-axi direction, this                */
/* function determines the number of counter-clock turns              */
/* along the tour of all points. The tour starts from the             */
/* first point.  At any point, the robot makes a counter-             */
```

Download at http://www.pin5i.com/


```
/* clock-wise turn in order to face the next point.  After      */
/* a turn, it proceeds to next point.  The final position       */
/* is the first point facing the X-axi.                         */
/*                                                              */
/* Copyright Ching-Kuang Shene          Sept/01/1989            */
/* ----------------------------------------------------         */

int  turns(int x[], int y[], int n)
{
     int X = 1;              /* robot's previous direct X        */
     int Y = 0;              /* robot's previous direct Y        */
     int TURNS = 0;          /* # of 90 turns to 1st Q.          */
     int count;              /* # of complete 360 turns          */
     int new_x;              /* robot's next direct. X           */
     int new_y;              /* robot's next direct. Y           */
     int new_turns;          /* # of 90 turns to 1st Q.          */
     int i, temp;            /* working                          */

     for (count = i = 0; i <= n; i++) {     /* scan all points  */
         new_x   = x[(i+1) % n] - x[i % n]; /* new dir.         */
         new_y   = y[(i+1) % n] - y[i % n];
         new_turns   = 0;            /* assume no 90 turn.      */
         while (new_x <= 0 || new_y < 0) {  /* turn it to       */
             temp    = new_x;        /* first quartant           */
             new_x   = new_y;
             new_y   = -temp;
             new_turns++;
         }
         if(TURNS>new_turns||(TURNS==new_turns && Y*new_x>new_y*X))
             count++;            /* crossing X-axi, inc ocunt    */
         X   = new_x;            /* proceed to next point        */
         Y   = new_y;
         TURNS   = new_turns;
     }
     return count;               /* return # of 360 turns        */
}
```

问题 9.5　最大涵盖距离（MAXCOVER.C）

已知一个数组 x[ ]，假设 x[j]是 i 到 j 之间的极大点，这指的就是对于任一在 i 与 j 之间（含 i 与 j）的 k 而言，x[k]小于等于 x[j]的绝对值，x[j]的绝对值涵盖（Cover）住[i,j]

Download at http://www.pin5i.com/

这个区间。请写一个程序，找出最长的被涵盖的区间。

【解答】

这个题目有一个非常漂亮的解法，不但如此，这个解法还可以达到与 n 成正比的线性程序。在问题说明中的片段之所以会变成与 n^2 成正比，原因就在于它做了许多多余的工作，在下面先指出多余的工作何在，了解了之后就很容易写出高效率的程序了。

假设数组元素用 $x_0, x_1, x_2, \cdots$ 表示，对于 x_i 而言，已知它涵盖住 $x_i, x_{i-1}, x_{i-2}, \cdots, x_j$ 这些元素，但却不包含 x_{j-1}。换言之，对于 k，$i \leqslant k \leqslant j$ 而言，有 $x_k \leqslant |x_i|$，但是 $x_{j-1} \leqslant |x_i|$。因此 x_i 所涵盖住的区间长度就是 $i-j+1$。若要找出更长的区间时，就要往左找，也就是说在 x_i 的左边，在 i 与 j 之间找到了一个 x_j 满足 $x_{j-1} \leqslant |x_p|$ 的条件，而 $x_{p+1}, x_{p+2}, \cdots, x_j$ 都不满足。换言之，x_p 是在 x_i 左边的第一个满足 $x_{j-1} \leqslant |x_p|$ 的元素，于是从 x_{n-1} 起就“有可能”开始一个更长的区间，如图 18-9 所示。

图 18-9

对于 $j \leqslant k \leqslant p$ 的 x_k 而言，满足 $x_k \leqslant |x_p|$ 证明并不难做，因为 x_i 涵盖住 [j,i] 这一段，所以 $x_k \leqslant |x_i|$。但是 x_i 不能涵盖住 x_{j-1}，故 $x_{j-1} \geqslant |x_i|$。不过，x_p 是在 x_i 左边第一个满足 $x_{j-1} \leqslant |x_p|$ 的元素，所以 $x_k \leqslant |x_i| < x_{j-1} \leqslant |x_p|$，因此 x_p 就涵盖住 $j-1 \sim p$ 这一段距离。

注意，因为 x_p 是在 x_i 左边的第一个满足 $x_{j-1} \leqslant |x_p|$ 的元素，所以，$x_{p+1}, x_{p+2}, \cdots, x_j$ 这些元素的绝对值都小于 $|x_p|$。因此，x_p 是在 x_i 左边第一个可以涵盖住 $x_{p-1}, x_{p-2}, \cdots, x_i, x_{j-1}$ 的元素，而在 $x_{p+1} \sim x_i$ 之间的元素都涵盖不到 x_{j-1}。可以尝试把 x_p 的涵盖范围自 x_{j-1} 延伸到 $x_{j-2}, x_{j-3}, \cdots$ 去，到不能再延伸时，假设到了 x_s 这个元素，亦即 x_p 可以涵盖住 $x_s, x_{s+1}, \cdots\cdots, x_p$，但不能涵盖到 x_{s-1}，反复上面的动作就又可以再度突破 x_{s-1} 这一点。

用这个观点写成的程序在 MAXCOVER.C。程序中 length 表示目前找出的边的长度，bound 就是上述的 x_p，它从 $n-1$ 起，在开始时，x_{n-1} 一定可以涵盖住自己。

程序中在外面的 for 循环从 x[bound]的左边元素起递减，它相当于上面的 x_j，因此在内的 for 循环就从 x[bound]起到 x[i]止找出一个 x[bound]使得它的绝对值可以涵盖住 x[i]。找到了新的 bound 之后，就改正 length 的值，接着 i 再继续降低，直到它比原 x[bound]的绝对值大，然后再降低 bound 的值，就这样一直到所有元素都处理完为止。

这个程序一共进行了多少次比较呢？注意一点，内循环的 bound 没有在每一次进入时都复位初值，而是令 bound 一直降低，于是最多就是绕 n 次而已。外循环也是最多绕 n 次，当固定在某个 i，x[i]会与 x[bound]比较，而使 bound 递减，但是下一次 i 的值就从这一次留下的 bound 值比起了，所以比较的次数就与 n 成正比。

Download at http://www.pin5i.com/

【习题】

（1）请很严谨地证明比较次数与 n 成正比。请问在最坏的情况下要做多少次？

（2）在实用上程序还可以写得更快些，不过或许长一点。有没有觉得在外循环中一共要更新 length 的值 n－1 次太多了一点？请修改程序在不增加其他比较次数的前提下降低更新 length 的次数。

（3）这个程序并没有指出具有最大长度的区间在什么地方，请修改程序，使它除了报告最长区间之外，也把所有具有该长度的区间找出来。请问最多会有多少个方案？为什么？

【参考文献】

这个题目取自 M.Rem 的专栏，M.Rem 指出它来自 A.Kaldewaij。M.Rem 的解法是源自很严格的数学论证，所以最后的程序就反映出这一结论，在此的解法则是习题（2）的建议，道理是一样的，不过效率上有些差异而已。

M.Rem, Small Programming Exercise 11, *Science of Computer Programming*, Vol.7 (1986), pp.313～318

【程序】

程序 18-5（MAXCOVER.C）

```
/* ------------------------------------------------------------ */
/* FUNCTION  maxcover :                                          */
/*   Given an integer, this function computes the max.           */
/* cover interval length of the array.  A cover interval         */
/* of an element of the array, say x[i], is an interval          */
/* [j,i], j <= i, such that for all k, j<=k<=i, we have          */
/* x[k] <= abs(x[i]).  A maximal cover interval is an            */
/* interval with maximum length.                                 */
/*                                                               */
/* Copyright Ching-Kuang Shene          Aug/11/1989              */
/* ------------------------------------------------------------ */

#include <math.h>            /* for abs()                        */

#define  MAX(x, y)  ((x) >= (y) ? (x) : (y))

int  maxcover(int x[], int n)
{
     int length;             /* max. interval length             */
     int bound;              /* last element searched            */
     int i;

     length  = 1;            /* min. length is one               */
     bound   = n - 1;        /* start from n-1 the item          */
```

Download at http://www.pin5i.com/


```
for (i = n - 2; i >= 0; i--) { /* for each item...          */
    for ( ; x[i] > abs(x[bound]); bound--) /* bound?         */
        ;                          /* now x[i] <= x[bound] ...*/
    length = MAX(length, bound-i+1);   /* update len.        */
}
return length;
}
```

问题 9.6 最大连续元素和（MAXSUM.C,MAXSUM1.C）

已知数组 x[ ] 储存了一组整数，请写一个程序，找出在数组中连续元素的和中最大的一个。举例而言，如果有数组 1,2,-6,3,-2,4,-1,3,2,-4，那么连续的元素的和有 1+2=3，1+2+(-6)=-3，2+(-6)=-4，…，但值最大的就是 3+(-2)+4+(-1)+3+2 这一段，值为 9，规定，和为负值时就定成 0，所以结果永远不为负。这个题目通常叫做最大连续元素和（Maximum Consecutive Sum）问题。

【解答（解法一）】

先看一个干净利落的解法，要用两个变量，一个叫做 e，另一个叫做 s。当在处理过程中做到 x[i]时，e 就是从第 i 个元素之前的某处起一直加到 x[i-1]的和，而 s 则是在 x[1]～x[i-1]之间，某一段和为最大的值，如图 18-10 所示。

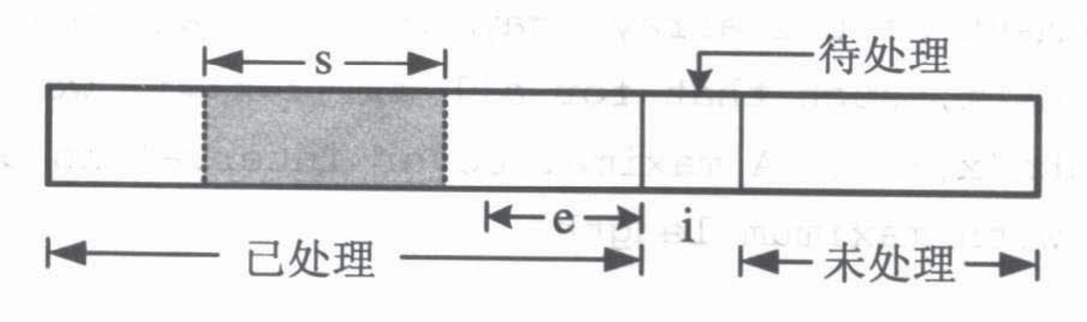


图 18-10

当 x[i]出现时，e 与 s 的值都可能会受到影响，为什么呢？e 是从 x[i-1]之前某处加到 x[i-1]的和，当然 x[i]出现了，就得把 e 加上 x[i]，而延伸到 x[i]。但是这个新的 e 值如果比 s 还大，那么到 x[i]为止的最大的和就是新的 e 而不是 s 了，所以 s 就得换成新的 e 值。

换言之，当处理到 x[i]时，得：

（1）把 x[i]加到 e 中，因此 e 就是从 x[i]之前某处起到 x[i]的和。当然，如果这个和比 0 小，就把 e 定成 0，因为说过和小于 0 时就看成 0。

（2）新的 e 值得到了，如何更新 s 值呢？s 值原来是 x[0]～x[i-1]之间的最大连续元素和，如果在上一步得到的新 e 值比 s 还大，那么最大连续元素和（在 x[0]到 x[i]之间）就变成新 e 值了，因此把 s 换成 e 的值。

（3）上面两步做完之后，s 与 e 分别变成 x[0]～x[i]之间的最大连续元素和，以及从 x[i]之前某处起到 x[i]的和。所以当 x[n-1]也处理完后，s 中就是所求的结果了。

不过或许还会有一个疑点，e 从某处起，那个“某处”究竟何在？事实上可以根本不

Download at http://www.pin5i.com/

必管这个问题，但若要追根究底，就来看看。一开始，也就是连 x[1]都不曾处理时，e 与 s 都是 0，相信是同意的，接下来，e 的值就会上升、下降，只要与 x[i]的和还是大于 0，e 就有可能再增加；但若与 x[i]的和小于 0，e 的值就应该告一段落了，一个全新的 e 就从 x[i]算起。因此 x[0]～x[n-1]是分成好几段的，每一段如果加上下一个元素就会小于或等于 0，如图 18-11 所示。

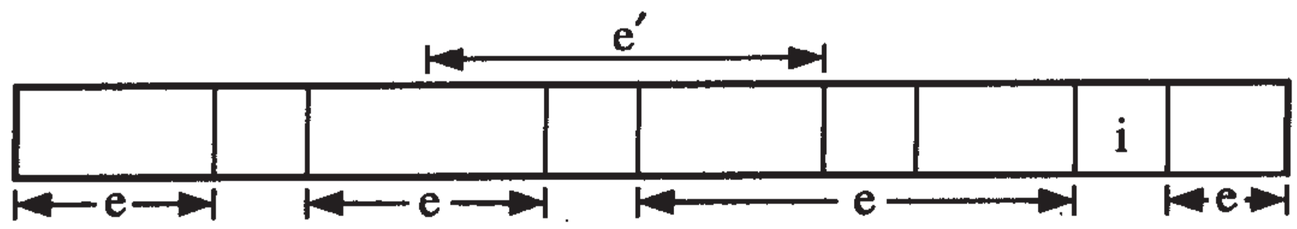


图 18-11

那么为什么到 x[i]之前不会有图 18-11 中 e′的那个情况呢？这一点要读者自己思索一下了。

举例说明这些疑点，下面表格中第一列从左至右是 x[ ]的各个值，第二与第三列是处理完对应的 x[i]后的 e 与 s 的值，如图 18-12 所示。

		e			e				e		
x[ ]		1	2	-3	1	4	-3	-3	2	4	-3
e	0	1	3	0*	1	5	2	0*	2	6	3
s	0	1	3	3	3	5	5	5	5	6	6

图 18-12

注：表示和小于或等于 0。

能否从这个图看出为什么 e 的范围不会跨越和小于等于 0 的那些地方呢？

【程序说明】

程序中 max_so_far 是 s，max_ended_here 则是 e，因为 C 语言没有求若干数据极大值的函数，因此用 if…else 来写，程序是极其简单的寥寥数列而已。注意，这个程序把每一个元素都只处理一次，所以就总共查了 n 个元素，效率上是比问题说明中的方法快了 n 倍。

【习题】

见解法二的部分，两者合并。

【解答（解法二）】

有另一个方法虽然效率不及上一个，但也是值得一提的。这个方法用分而治之（Divide-and-Conquer）的技巧，把 x[0],…,x[n-1]分成两等份，即左边与右边，分别把左边与右边的最大连续元素和找出来，令为 L 与 R，如图 18-13 所示。

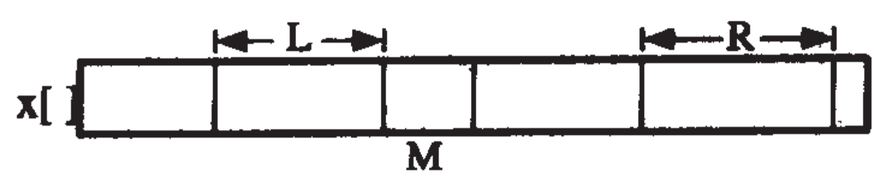


图 18-13

如何从 L 与 R 把整个 x[ ]数组的最大连续和找出来呢？如果中点是 M，L 与 R 的值都没有跨越中线，所以不可能成为解答。因此，要算出包含了中间点 M 的最大连续元素和。

Download at http://www.pin5i.com/

不过这并不难，从 M 起往左算，找出最大的和，再从 M 起往右算，找出最大的和，两者相加就是包含了中间点的最大连续和了。例如，如果有 1,2,-3,1,4,-3,-3,2,4,3，左半是 1,2,-3,1,4，右半是-3,-3,2,4,3。于是左半从中间点往右算依次求出的和是 4,5,2,4,5，极大值是 5；右半从左往右算，依次加出来的和是-3,6,-4,0,3，最大的和是 3，所以含中央的最大和是 5+3=8。单单左边的和为 L，单单右边的和是 R，一定含有中间的和 8，求这 3 者的极大值就是答案了。不过，现在 L 与 R 未知，但也不难，递归地调用自己就可以求出来了。

【程序说明】

程序中用 left 与 right 表示要处理 x[left]到 x[right]之间（含）的内容（注意，x[ ]放到外边成为外在变量了）。

程序中先把简单的情况（left>right，或 left==right）做出来，但若 left<right，求 left 与 right 的中点 middle，调用自己把 left 与 middle（左边）、middle+1 与 right（右边）的结果算出来。接着向左、向右求出包含了中间的最大和，最后求 3 者的极大值就是答案了。

这个程序的效率如何？如果令 T(n) 表示有 n 个元素时要查的元素个数，会把 n 分成两半去处理，换言之，这两半各花 T(n/2) 这么多时间。然后向左、向右都求和，一共又查了 n 个元素，所以总共查了 $T(n)=2T(n/2)+n$ 个。当 $n=1$，亦即只有一个元素时根本不必相加，$T(1)=0$。如果把 $T(n)=2T(n/2)+n$ 展开几项，就会看到：

$$
\begin{aligned}
T(n) &= 2T(n/2)+n \\
&= 2[2T(n/2^2)+n/2]+n \\
&= 2^2T(n/2^2)+2n \\
&\vdots \\
&= 2^iT(n/2^i)+i\cdot n
\end{aligned}
$$

令 $2^i=n$，那么 $i=\log_2 n$，代入上式有：

$$
\begin{aligned}
T(n) &= nT(n/n)+n\log_2 n \\
&= nT(1)+n\log_2 n \\
&= n+n\log_2 n
\end{aligned}
$$

这就是检查元素的次数，比第一种解法多了 $n\log_2 n$，不过还是比用 $n(n+1)$ 个加法快得多。

【习题】

（1）请说明为什么 e（第一种解法）一定会在它小于或等于 0 的地方分界。

（2）解法一的程序只给了结果，请修改程序，把 x[ ]中造成最大值的那一段数据显示出来。

（3）如果 x[ ]中有 0，对解法一有没有影响，解法二呢？

（4）请说明在解法二中，求左边、右边与中间 3 个最大和的极大值就是对 x[ ]的最终结果。

（5）像第（2）题一样，请修改第二个解法，让它也可以返回造成最大和的那一段数据的左、右两边的脚码。

（6）如果程序能够像第（5）题一样返回那一段会造成最大和的数据的开始与最后的

Download at http://www.pin5i.com/

脚码时，程序也就可以使用与 n 成正比的加法个数找出结果，而不必用到 $n+n\log_2 n$ 个加法了。能看出为什么吗？请把程序写出来。

（7）能够证明不论用哪一种方法来解这个题目，在最坏的情况下，一定要至少查过 n 个元素吗？

（8）试写一个程序，求出数组中连续元素的和中最小的一个（数组元素为实数）。

【参考文献】

根据 David Gries 的记载，Carnegie-Mellon 大学的 M.I.Shamos 看到一位统计学家用一个与 n^3 成正比的方法做这个题目（其实是习题（8）），于是 Shamos 与 J.Bently 合力发展了与 n^2 成正比的方法，一周之后，他们做出了解法二的方法（大约是 $n\log_2 n$）。两周之后，他们与另一位统计学家讨论这个问题时，那位统计学家马上就给出了解法一中与 n 成正比的技巧。Gries 没有明确指出日期，但他的文章在 1982 年发表，因此这个问题早在 1982 年就出现了。Gries 的文章是最先出现的，很系统地介绍了这个问题，并且指出其起源的。原来，Bentley 曾经去康乃尔（Cornell）大学讲学，顺便地就把这个问题提出来考考人，从而使 Gries 把这个问题的解法原原本本地用很严密的方式推导出来。至于 Bentley 本人的现身说法，则一直要迟到 1984 年 9 月才在他自己有名的专栏 Programming Pearls 中讨论，所采用的就是 Bentley 的形式，不过 Gries 的文章一样值得研读。

1 J.Bentley. *Programming Perals*, Addison-Wesley, 1986, pp.69～80

2 D.Gries. A Note on a Standard Strategy for Developing Loop Invariants and Loops, *Science of Computer Programming*. Vol.2(1982), pp.207～214

【程序】

解法一是 MAXSUM.C，解法二为 MAXSUM1.C。

程序 18-6（1）（MAXSUM.C）

```
/* ------------------------------------------------------------------ */
/* FUNCTION max_sum :                                                 */
/*   This function accepts an integer array and computes              */
/* the maximum consecutive sum.  If the maximum sum is                */
/* negative, this programs returns zero.                              */
/*                                                                    */
/* Copyright Ching-Kuang Shene          July/01/1989                  */
/* ------------------------------------------------------------------ */

int  max_sum(int  x[], int n)
{
     int max_ending_here = 0;    /* max. computed ended here          */
     int max_so_far      = 0;    /* global maximum so far             */
     int i;

     for (i = 0; i < n; i++) {   /* scan each element in turn         */
         if (max_ending_here + x[i] < 0)/* sum positive?              */
```

Download at http://www.pin5i.com/


```
            max_ending_here = 0;    /* NO, set it to 0          */
        else                        /* else update the max. sum*/
            max_ending_here += x[i];/* ended here and           */
        if (max_ending_here > max_so_far)      /* sum inc?     */
            max_so_far = max_ending_here;      /* YES, update  */
    }
    return max_so_far;
}
```

程序 18-6（2）（MAXSUM1.C）

```
/* ------------------------------------------------------------ */
/* FUNCTION max_sum :                                           */
/*   Another slower solution of the max_sum problem with        */
/* order O(n log(n)). Not only it is slower, but also its       */
/* program is much longer.                                      */
/*   Note the difference in the function head. This one         */
/* requires 'left', the left boundary pointer, and 'right'      */
/* the right boundary pointer.                                  */
/*                                                              */
/* Copyright Ching-Kuang Shene          July/01/1989            */
/* ------------------------------------------------------------ */

#define max(x, y)   ((x) >= (y) ? (x) : (y))

int max_sum(int x[], int left, int right)
{
    int middle;                 /* the middle element          */
    int max_to_left;            /* max found in [left,mid]     */
    int max_to_right;           /* max found in [mid+1,rig]    */
    int max_crossing;           /* max found cross middle      */
    int sum;                    /* working variable            */
    int left_max;               /* left max from recursion     */
    int right_max;              /* right max from recursion    */
    int result, i;

    if (left > right)           /* no element remains?         */
        result = 0;             /* result must be zero         */
    else if (left == right)     /* exactly one element?        */
        result = (x[left] > 0) ? x[left] : 0;  /* ret max      */
    else {                      /* now we have many element    */
        middle = (left + right)/ 2;            /* get the mid. */
        left_max    = max_sum(x, left, middle); /* left        */
        right_max   = max_sum(x, middle+1, right); /* rig.     */

        sum = 0;                /* compute max from mid down   */
        max_to_left = 0;
```

Download at http://www.pin5i.com/


```
        for (i = middle; i >= left; i--) {
            sum += x[i];
            if (max_to_left < sum)
                max_to_left = sum;
        }
        sum = 0;                    /* compute max from mid+1 up    */
        max_to_right = 0;
        for (i = middle+1; i <= right; i++) {
            sum += x[i];
            if (max_to_right < sum)
                max_to_right = sum;
        }

        max_crossing = max_to_left+max_to_right;/*cross          */
        if (max_crossing >= max_to_left)        /* get the max */
            result = max(max_crossing, max_to_right);
        else
            result = max(max_to_left, max_to_right);
    }
    return result;
}
```

问题 9.7　最大连续元素积（MAXPROD.C）

一个整数数组中有正数、负数与 0，请写一个程序，找出这个数组中连续的元素相乘的极大值。

【解答】

在这个题目中，如果像问题 9.6 一样，用一个变量 M 记录下从目前位置之前到目前位置的各个元素的乘积的极大值，这是不够用的，问题所在就是有了负数。举个例子，如果数组中有-2,3,4,-4，而目前正要处理-4，自然，在-4 之前的 M 应该是$12=3\times4$；当-4 出现了，$(-4)\times12=-48$，比 12 小，能够说极大值是 12 吗？当然不行，因为$(-2)\times3\times4\times(-4)=96$才是真正的极大值。

为什么会这样呢？关键就是两个负值相乘会成正数，换句话说，值变大了。要克服这一点并不难，用另一个变量 m 来记录下到目前这个数的负值最小连乘积。用-2,3,4,-4 为例，在处理-4 之前，负值最小的是$(-2)\times3\times4=-24$，所以 m 是-24，一旦乘上了-4，就变成 96，如果它比到目前为止所记住的极大值还大，于是这个新的积就变成极大值了。综合起来，用 M 与 m 记录下到目前这个位置为止的连乘积中正值的极大值与负值的极小值，如图 18-14 所示。

Download at http://www.pin5i.com/

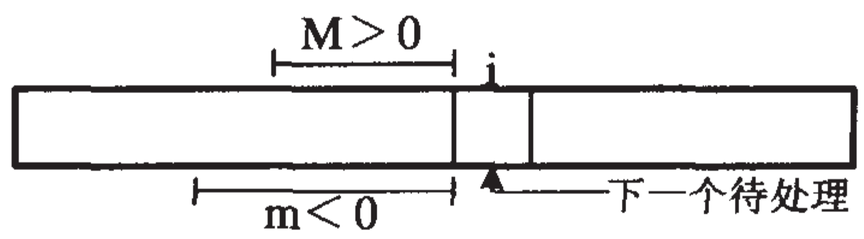


图 18-14

如果现在要处理的元素是 x_i，那么依 x_i 的符号，只有 3 种可能性：

（1）$x_i>0$：由于 x_i 是整数，而且大于 0，因此 x_i 至少会是 1，$M\cdot x_i$ 不可能变小，到 x_i 为止的 M 值，就变成 $M\cdot x_i$。但正的 x_i 与负的 m 相乘，$m\cdot x_i$，当然会变得负的更小。不过因为相乘时 m 的初值为 1，所以到 x_i 为止的新 m 值就是 $m\cdot x_i$ 与 1 的极小值，而 m 不一定是负的，当它不是负值时，就一定是 1。

（2）$x_i<0$：$m\cdot x_i$ 变成正的了，如果它比 M 还大，那当然新的 M（到这个位置的相连乘的最大正值）就得换成为 $m\cdot x_i$。反过来，新的 m 就是原来的 M 值与 x_i 的乘积，原因就是 M 本来是到 x_{i-1} 的最大正的连乘积，现在变成负的，而原来的 m 又消失了，所以 $M\cdot x_{i-1}$ 就成为新的 m。

（3）$x_i=0$：这是一个简单的情况，如果没有正值时，极大值就是 1，现在 x_i 是 0，它不能与 M 与 m 相乘，一乘就会令它们为 0，所以一切就都要从 x_i 开始重新计算了。因此 M 是 1，而 m 则定成 0。

用这个观点编写的程序是 MAXPROD.C，M 与 m 分别用 max_ending _here 与 min_ending_here 取代，而整体的极大值还是用 max_so_far 变量来保存。特别注意一点，在 else 的部分，因为 C 语言中无法同时把 $m\cdot x_i$ 与 $M\cdot x_i$ 分别存入 M 与 m，所以引进了一个工作用的变量 temp 来暂时保存 M 的值。

【习题】

（1）请修改程序，找出造成有极大相乘积的是哪些元素。

（2）取消当没有正值时，极大值为 1 的要求。因此若数组为-1,0 时，极大值为 0；数组为-1 时，极大值为-1；数组为-1,0,-1 时，极大值为 0。请修改程序，完成这一个比较合理的做法。

（3）在本题中的数组只有整数，但若是改变成可以有实数，问题就更复杂了。例如，若数组为 1,2,0.7,3,4，极大值是多少？当然是1×2×0.7×3×4=16.8。换言之，乘上一个比 1 小的正数有时并不会对连乘积有任何影响，但若数组为 1,2,0.1,3,4，那么极大的连乘积就是 3×4=12 了。请在数组元素都是正值的前提下，修改此程序来完成工作。

（4）修改第（3）题的程序，使它可以处理数组中有 0 与负值的情况，而且像第（2）题一样，输出真正的极大值，而不是用 1 来取代。（提示，先做第（3）题，再使用本例题的技巧）

【程序】

程序 18-7（MAXPROD.C）

```
/* ------------------------------------------------------------------ */
/* FUNCTION max_prod :                                                */
```

Download at http://www.pin5i.com/


```
/*   This function accepts an integer array and computes        */
/* the maximum consecutive product.  If the maximum is          */
/* negative, this programs returns one.                         */
/*                                                              */
/* Copyright Ching-Kuang Shene          Sept/01/1989            */
/* ----------------------------------------------               */

#define  MIN(x, y)  ((x) <= (y) ? (x) : (y))
#define  MAX(x, y)  ((x) >= (y) ? (x) : (y))

int  max_prod(int  x[], int n)
{
     int max_ending_here = 1;     /* max. computed ended here     */
     int min_ending_here = 1;     /* min. computed ended here     */
     int max_so_far      = 1;     /* global maximum so far        */
     int temp, i;

     for (i = 0; i < n; i++) {   /* scan each element in turn    */
         if (x[i] > 0) {             /* if x[i] is positive ...  */
             max_ending_here *= x[i];/* max will incr.           */
             min_ending_here = MIN(min_ending_here*x[i],1);
         }
         else if (x[i] == 0) {       /* if zero, reset           */
             max_ending_here = 1;
             min_ending_here = 0;
         }
         else {                      /* x[i] < 0                 */
             temp            = max_ending_here;
             max_ending_here = MAX(min_ending_here*x[i],1);
             min_ending_here = temp * x[i];
         }
         max_so_far = MAX(max_so_far, max_ending_here);
     }
     return max_so_far;
}
```

问题 9.8　寻找名人（SINK.C）

假设有一群人，而您是新闻记者，想要找出这群人中的“名人”。何谓名人？就是人家都认得他，而他却不认识别人的那一个人。您在访问时只能用这样的问题，当您访问张三

Download at http://www.pin5i.com/

时，您只能问他“您认不认得李四？”，“您认不认得王五？”等。换句话说，您在访问 A 时（A 可以任您挑选，要访问多少个 A 与问 A 多少个问题都行）只能问 A：“您认不认得 X？”。当然，被访问者只会回答“认得”与“不认得”，而不会有其他答案。请问，您从访问而得的答案中要如何找出谁是名人，或这一群人中根本就没有名人呢？这个问题通常叫做名人问题（Celebrity），或水槽问题（Sink）。

【解答】

通常在一大堆资料中查找自己需要的那一个是不容易的（比如在乱七八糟的桌上找某本书、某篇文章、某个磁盘），但把不是自己要的部分去掉就简单多了。就以找书为例，把不是自己要的书搬到一边去，再查下一本，……，最终不外是所有书都搬走了（桌上没有您要的书），就是留下您要的书。这个观点是很容易运用的，就用它来解决名人问题。访问 i，问他“你认得 j 吗？”，如果 i 说认得 j，那么 i 就一定不是名人，因为名人是不认得其他人的，于是 n 个人中就被去掉了 i，而只剩下 $n-1$ 个人了。如果 i 说不认得 j，那么 j 就一定不是名人，因为人人都会认得名人，因此去掉 j 之后，n 个人就变成 $n-1$ 个人，问题也同样被化简了。如图 18-15 所示是 5 个人的 k[ ][ ]矩阵，从图看来会发现 4 是名人。

	1	2	3	4	5
1		1		1	1
2	1		1	1	1
3		1		1	
4					
5	1		1	1	1

1：真

图 18-15

先访问 1，问他“1 认得 2 吗？”，k[1][2]是“真”，所以 1 不是名人；因此名人只会在{2,3,4,5}中。现在看 2（刚才问题中没有被去掉的人），问他“2 认得 3 吗？”，是，2 不是名人，于是名人只可能在{3,4,5}中。接着看 3，问他“3 认得 4 吗？”，是，3 不是名人，因此名人可能在{4,5}中。再来问 4，“4 认得 5 吗？”不认得，所以 5 不是名人，这样就剩下{4}了，能不能说 4 就是名人呢？不能。如果 k[4][1],k[4][2],k[4][3]有一个是“真”，4 就不是名人了。换句话说，上面的一连串问题到最后的结论是“不是名人的人都去掉了”，但最后剩下来的 4 却不能保证是名人。要如何才能肯定剩下来的那个人是名人呢？就用 4 为例，再去查第 4 列者是不是全为“假”，第 4 行上除了 k[4][4]之外全为“真”就行了。如果不是，那么连 4 也得去掉，换句话说，没有名人。但上面的 k[ ][ ]中却是成立的，所以 4 是名人。

上面的方式一共问了多少个问题呢？最先要问 $n-1$ 个问题，把 n 个人中去掉 $n-1$ 个而留下一个，再用 $n-1$ 个问题看看剩下来的那个人是否不认得其他人（不必查自己认不认得自己）。接着，另外的 $n-1$ 问题看看其他人是否都认得他，在最坏情况下，就一共问了 $3(n-1)$ 个问题，不是要比 n^2 个问题的方法快很多吗？

在后面的程序中，第一个 for 循环是用来去掉那些不是名人的。用 candidate 来储存“可能”的名人，一开始是 1，然后在逐次比较中更换 candidate 的值，方式已如上述。在进入第二个 while 循环时，就检查 candidate 是否真的是名人，方法就是看看第 candidate 列

Download at http://www.pin5i.com/

是否都是“假”，而第 candidate 行上除了第 candidate 列的元素之外都是“真”，如果都成立，candidate 就是名人，于是 the_sink 就返回 candidate 的值，否则返回 0。

【习题】

（1）依对“名人”的定义，请证明在一群人中如果有名人存在的话，名人的数目就只会有一个，因此函数只返回一个值就是合理的了。

（2）事实上，这个题目用 $3n-\lfloor \log_2 n \rfloor-3$ 个问题就足够了，$\lfloor \log_2 n \rfloor$表示把 $\log_2 n$ 的小数部分去掉，例如，$\lfloor 4.5 \rfloor=4$，$\lfloor 3.1 \rfloor=3$。原因在于若在第一个阶段多找出些信息的话，第二阶段就不必问 2n-2 个问题的。可以把人们两人一组，假设 A 与 B 一组，问 A，“你认得 B 吗？”，如果 A 答是，就把 A 去掉，不然就把 B 去掉。把每一组都问完后就去掉一半的人（用了 n/2 个比数），把剩下来的人再两两分组，一组一组地问，因此重复 $\log_2 n$ 次就可以找出“可能”的名人，不过也是用了 n-1 次比较（为什么？）。

但是，分组时可以更小心些，如果有一个人在这一轮中落了单，那么在下一轮中就一定要与某人配一组；如果在这一轮中有两个人曾经落了单，这两人就分在一组。于是“可能”的名人就一定与$\lfloor \log_2 n \rfloor$个人配对过（为什么？），与他配对过的人的结果自然就可以记录下来，在第二个阶段进行检查时，这$\lfloor \log_2 n \rfloor$个人就不必查了，因此就只用了 $3n-\lfloor \log_2 n \rfloor-3$ 次比较而已。

请回答以下的问题：（1）证明上面的结论。（2）请写一个程序完成上面的做法。

【参考文献】

此解法采用了 Udi Manber 的方法，但是在第一阶段经过修改而变得简单些。对这个问题有兴趣的读者，不妨参看 King 与 Smith-Thomas 的文章，与所附的问题来源。

1 K.N.King and B.Smith-Thomas. An Optimal Algorithm for Sink-Finding, *Information Proces sing Letters*, Vol.14(1982), pp.109～111

2 U.Manber. Using Induction to Design Algorithms, *Communications of ACM*, Vol.31 (1988), pp.1300～1313

【程序】

程序 18-8（SINK.C）

```
/* ------------------------------------------------------------------ */
/* FUNCTION  sink :                                                   */
/*   Given a matrix containing who-knows-who information              */
/* this function check the matrix to see if there exists              */
/* the sink, or the celebrity.                                        */
/*                                                                    */
/* Copyright Ching-Kuang Shene          Aug/27/1989                   */
/* ------------------------------------------------------------------ */

#define  YES        1
#define  NO         0
#define  NO_SINK   -1
```

Download at http://www.pin5i.com/


```
#define  MAXSIZE     100

int sink(char know[MAXSIZE][MAXSIZE], int n)
{
     int candidate;              /* possible candidate          */
     int next;                   /* working; points to next     */
     int wrong;                  /* a loop control variable     */
     int the_sink;
     int i, j, k;

     the_sink = NO_SINK;         /* suppose there is no sink    */
     candidate = 0;              /* ask question from the 0th   */
     for (i = 1; i < n; i++)
         if (know[candidate][i]) candidate = i;

     for (wrong = NO, k = 0; (k < n) && !wrong; k++) {
         if (know[candidate][k]) /* cand. knows nobody          */
             wrong = YES;        /* so the cand. is wrong       */
         if (!know[k][candidate] && candidate != k)
             wrong = YES;        /* also wring if no one know   */
     }
     if (!wrong)
         the_sink = candidate;
     return the_sink;
}
```

问题 9.9　投票问题（VOTING.C）

这个问题一般叫做过半数（Majority）问题，也有人称为投票（Voting）问题，但以前者的说法居多。假设有 1,2,…,n 等 n 个人参加投票，他们只能圈选一个人，但是却可以选任何一人，甚至于可以选不在这 n 个人中的另一个人都行。例如，如果 5 个人投票，分别投给 1,8,1,100,1，1,3,5 都投给 1，2 投给 8，4 投给 100。问题是，写一个程序接收这些投票结果，看看有没有人的得票过半数，就以上例来看，1 的得票数就过半数了。

【解答】

解这个问题不必排大小，似乎有一点不可思议。也许会想，那效率一定会很低。不，错了，事实上非常快，比用排大小的方法还快。关键何在？注意，在做这个题目时面临一个难题，这就是没有候选人的信息，可能这几个人都选不在他们之中的另一些人，可能选第一百万号、选第一兆号，因此使一些简单方法无用武之地，而难也就难在这。

Download at http://www.pin5i.com/

常常可以把不必要的东西去掉，而把问题变得比较单纯，在本书中有不少问题都有这个特征。假设在给定的投票结果中任选两张，一张投给 A，一张投给 B，假设 A 与 B 不相等。如果把 A 与 B 去掉，那会有什么结果呢？可以作如下的分析。

（1）如果 A 与 B 都没有得到过半数的票，那么把 A 与 B 这两张票去掉的话，是不会影响到真正票数过半数的那一个人当选的。

（2）如果 A 与 B 中有一个人的得票过半数，把 A 与 B 这两张票去掉，也不会影响到真正票数过半数的那一人当选。为什么呢？如果得票过半数的人票数是 $n/2+k$，而未过半数的人的票数为 h，n 是投票人数，n/2 正好一半，所以 k 是过半数的部分。如果 A 与 B 中有一个人过半数，另一个不过半数，那么把 A 与 B 去掉，而留下 $n-2$ 张票。现在过半数的人的票数是 $n/2+k-1$，未过半数的人的票数为 $h-1$。于是 $n/2+k-1$ 不还是大于 $(n-2)/2$ 吗（注意，这是 $n-2$ 张选票中的半数）？

（3）不可能 A 与 B 都过半数的（为什么？），因此就不必讨论这个情况。

由上面的讨论如果 A 与 B 不相等，那么把 A 与 B 这两张选票去掉，并不会影响到得票过半数的人，就一直把选票去掉，最后剩下的就一定是过半数的票了。但是，若 A 与 B 相等又怎么办呢？因为 A 的票数多了一张，不过却不能去掉，若 A 是过半数的，一旦去掉了这两张，就很可能变成不是过半数了（想想，如果投票结果是 2,3,2,2,3，现在 $A=2$，$B=2$，把 A 与 B 去掉会有什么后果？）。

为了克服这一点，可以随便拿一张选票（比如第一张），把这个人叫做“可能当选人”，因此在一开始时，他得一票。接着，就查其他的选票，按照上面的分析，有下面的几种情况：

（1）如果“可能当选人”与正在查的选票上的人相同，这就加强了“可能当选人”的当选机会，所以把“可能当选人”的票数加上 1。

（2）如果“可能当选人”与正在查的选票上的人不同，把“可能当选人”的一张票与正在查的选票去掉，是不会影响到实际当选人当选的，因此就把“可能当选人”到目前的票数减去 1，表示去掉一票。至于正在查的选票，不理会它，不就是等于去掉了吗？

（3）但是，当把“可能当选人”的得票数减了之后，而使他的票数变成 0，这表示“可能当选人”的票数都与已经查过的人抵消完全，在以后过程中，“可能当选人”很可能不会当选，当然也有可能当选，但不论如何，依照在上面的讨论，都不会影响到实际当选人当选，所以在这种情况下，可以随便再找另一个“可能当选人”作基准而继续处理。

这个过程可以一直反复，直到所有选票都查完为止，因此在最后留下来的“可能当选人”真正当选的机会就大极了。不过事实上他却不一定当选，为什么呢？如果处理到最后一个元素时，那么减去 1 而使“可能当选人”的票数变成 0，把最后一个元素定成新的“可能当选人”，但若最后一个元素只有一票，那他哪有可能当选呢？所以，要算出“可能当选人”的实得票数，超过半数，他才是“真正当选人”。

以 1,2,2,1,1 为例说明。最先，1 是“可能当选人”，他有一票，接着 1 与 2 相比，不相等，两张票都去掉，留下 2,1,1 共 3 张。把第二个元素 2 当成新的“可能当选人”，只有一票。接着与第三个元素相比，都是 2，“可能当选人（2）”的票数多了 1 而变成 2；与第四个比，不相等票数减 1；再与第五个比，不相等票数再减 1，但票数就变成 0 了。所以把第五张票（1）选作新的“可能当选人”，继续处理，因为没有元素，因此停止工作。现在，

Download at http://www.pin5i.com/

"可能当选人"是 1，查一查选票中有多少张是投给 1 的，发现有 3 张，过半数，所以 1 是所求。

【程序说明】

程序中 vote[ ]数组是投票结果，n 是它的元素个数，candidate 所存的就是目前"可能当选人"的号码，vote_count 是该"可能当选人"到目前为止的统计票数（不含被去掉的部分）。第一个 for 就从第二张票起查过去，进行上述的动作。第二个 for 则用来算出 candidate 的真正得票数。函数值为 0 时，表示没有人过半数。

【习题】

（1）请找出一组投票结果，使得没有人得票过半数。

（2）请证明，如果有人得票过半数，那么就只会有一个人能够如此。把这个问题写成一个函数，它只需要返回一个值。

（3）如果只能投票给 1,2,3,…,k 这 k 个人（k 已知），请写一个程序做相同的工作（提示：很简单，但不要排大小）。

（4）如果只能投票给 $v_1,v_2,\cdots,v_k$ 这 k 个人（$v_1,v_2,\cdots,v_k$ 与 k 都为已知。例如可以投给 234,12,789,1234567890 这 4 个人），不过 n（投票的人数）比候选人的数目（k）大很多，请写一个程序做相同的工作（提示：也不难，但也不要把选票排大小）。

（5）请问，在程序中，在最坏的情况下，每一张选票被处理了多少次？解决第（3）、第(4)题的方法又把选票处理了多少次？在处理特殊情况时，这种方法真的比较有效率吗？

【参考文献】

这个题目起源于容错(Fault Tolerant)计算机系统中的一个问题，是在 1981 年由 J.Moore（发现配合字符串的 Boyer-Moore 方法的那一个 Moore)提出来的。1982 年 J.Misra 与 David Gries 两人合写了一篇深入探讨程序写作的文章，同年 M.J.Fischer 与 S.L.Salzberg 提出了一个比此例复杂的方法，比较 1.5n+1 次就可以找出结果，而且还证明了不管用什么办法，比较次数都不可能少于 1.5n-2 次。推荐读读上述的文章，以及所附的文献，了解问题的来龙去脉。

1 M.J.Fischer and S.L.Salzberg. Solution to Problem 81-5, *Journal of Algorithms*, Vol.3 (1982), pp.375～379

2 Misra and D.Gries. Finding Repeated Elements, *Science of Computer Programming*. Vol.2(1982), pp.143～152

3 J.Moore. Problem 81-5, *Journal of Algorithms*. Vol.2(1981), pp.208～209

【程序】

程序 18-9（VOTING.C）

```
/* ------------------------------------------------------------------ */
/* FUNCTION majority :                                                */
/*   Given vote[] containing a voting record, say x[i]=j              */
/* meaning that person i votes person j, this function               */
```

Download at http://www.pin5i.com/


```
/* determine if there is any one is a majority.  There is        */
/* no limit to the value of j.  In other words, anybody          */
/* can vote someone else and there is no fixed candidate.        */
/* Thus value in x[] could be very large, or even larger         */
/* than the size of x[].  'Majority' means that there is         */
/* a guy such that more than n/2 persons votes for him.          */
/*                                                                */
/* Copyright Ching-Kuang Shene          July/01/1989              */
/* ------------------------------------------------------------   */

int majority(int vote[], int n)
{
     int candidate;                  /* current candidate        */
     int vote_count;                 /* vote count for the cand. */
     int count;                      /* final count              */
     int answer;                     /* final answer             */
     int i;

     candidate   = vote[0];          /* assume the guy voted by */
     vote_count  = 1;                /* #1 is the candidate      */
     for (i = 1; i < n; i++)         /* for all other people ...*/
         if (vote_count == 0) {      /* if current no count      */
             candidate  = vote[i];   /* pick up a new can.       */
             vote_count = 1;         /* give him count 1         */
         }
         else if (candidate == vote[i]) /* is he a candi?        */
             vote_count++;           /* increase his count       */
         else
             vote_count--;           /* NO, decrease his count   */

     if (vote_count == 0)    /* finally check if the cand        */
         answer = 0;             /* is the guy with majority     */
     else {
             for (i = 0, count = 0; i < n; i++)
                 if (vote[i] == candidate)
                     count++;
             answer = (count > (int)(n / 2.0 + 0.5)) ? candidate : 0;
         }
         return answer;
}
```

Download at http://www.pin5i.com/

问题 9.10 寻找 1 对 1 函数（1TO1.C）

如果 S={0,1,2,3, …, n}是一个集合，f:S→S 是个从 S 映射到 S 的函数，请写一个程序，找出 S 的一个部分 S'集合，使得 f:S'→S'是一个 1 对 1 的对应函数，而且 S'的元素个数为最多。

【解答】

写这个程序有两个诀窍。第一，如果在 S 中有某个元素 j，它是孤零零的，没有任何元素会影响到它（请看问题说明中的 2 与 7），那么一定不能要 j。为什么呢？只要想到如果把 j 选到 S′ 的话，没有任何的 i 会对应到 j，j 就孤零零的单独一个，不是多余的吗？

如何找出那些 j 呢？这倒不难，可以准备一个数组 y[ ]，元素与 S 的个数一样多，在开始时定为 0。当 i 从 0 变到 n-1 时，找出 f (i)，把 y[f(i)]加上 1，因此 y[k]就表示会对应到 k 去的元素的个数。用问题说明的例子来看，y[ ]的值就分别依次为 2（1 与 2 对应到 0），0，1（0 对应到 2），1（5 对应到 3），2（3 与 4 对应到 4），1（6 对应到 5）与 0。当把 S 的每一个元素都查完之后，找出 y[ ]中会令 y[j]为 0 的 j 即可。上面提到的例子中，y[1]与 y[6]为 0，所以不能留住 1 与 6。

第二个决窍叙述如下。虽然没有元素会对应到 1 与 6，但 1 与 6 却会对应到其他的元素去。前述例子中，1 与 6 分别对应到 0 与 5。自然，1 与 6 被去掉了，1 与 6 所对应的元素就少了一个来源。例如，上述的例子在 1 与 6 被去掉之后就变成如图 18-16 所示的情况。

换言之，1 与 6 所对应到的元素（0 与 5）的 y[ ]值都少了 1，看图 18-16 就不难明白，把从 1 与 6 出发的前头都去掉了。但是，这又可能会造成一个新元素的 y[ ]值为 0，在此例中为 y[5]。这表示与 5 对应的元素（6）被去掉了，因此使 5 变成孤零零的一个，当然 5 也可以被去掉。注意，y[0]本来是 0，去掉 1→0，y[0]就变成 1，还有 2→0。再把 5 去掉，因此连同 5→3 也去掉了，如图 18-17 所示。

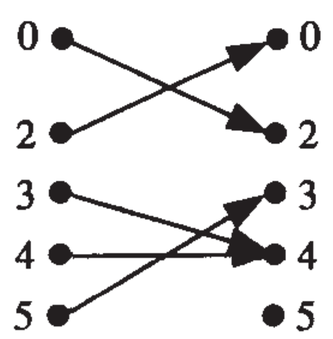


图 18-16

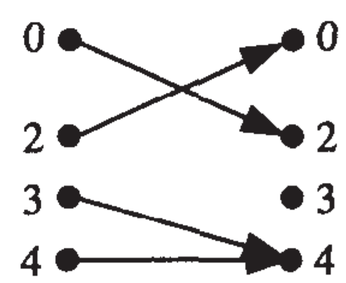


图 18-17

然而一旦把 5→3 拿掉，y[3]就从 1 变成 0，又成为孤零零的一个。照上面的说法，3 可以去掉，如图 18-18 所示。

于是 y[ ]中再也没有 0 的元素 3，这就是得来的结果。换句话说，当算出 y[ ]之后，就去找有没有 y[j]为 0，如果有，就把 j 去掉。至于把 j→f(j)去掉是不必做的，因为把 j 去掉了之后，把 y[f(j)]减去 1 就有相同的效果。就这样反复操作，直到 y[ ]中不再可以选出元素为止。

Download at http://www.pin5i.com/

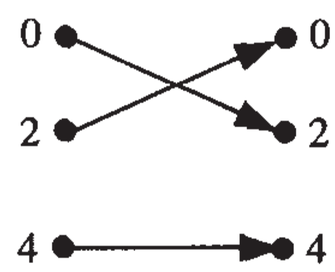


图 18-18

但是，反反复复去查 y[ ]并不是个很有效率的做法，如果有一点数据结构的知识，就知道用等待线（Queue，也叫做队列）来做是最好的办法。可以用一个数组 q[ ]，一开始时把令 y[j]为 0 的 j 存入 q[ ]，依照队列的定义，只能从头取元素，但加元素时要加到尾部。所以在以后的每一次循环中，都从 q[ ]的头部取一个 k 出来（因此 y[k]一定是 0），把 y[f(k)]减去 1（f(k)是 k 对应去的元素），这会使 y[f(k)]为 0，因此 f(k)就是个孤零零的元素了，所以就把 f(k)加到等待线 q[ ]的尾部，再继续下一个循环。这个操作就一直做到 q[ ]中没有元素为止，这就表示剩下来的元素就是所要求的 S'了。

【程序说明】

在程序中用 counter[ ]来代表上述的 y[ ]数组；funct[ ]则是函数 f()，换言之，如果 f 把 i 对应到 j，亦即 f(i) = j，funct[i]中就是 j。另外，status[ ]是用来记录要保留（SAVED）或删除（DELETED）对应的元素。

每一个 for 中把 counter[ ]归 0，status[ ]定成每一个元素都保留下来。第二个 for 求出对每一个 i 而言，有多个元素会对应到 j，结果就储存在 counter[ ]中。第三个 for 把 counter[ ]为 0 的也就是孤零零的元素放到队列 queue[ ]中，tail 记录着最后一个元素何在。最后的 while 循环是重点所在，第一次取出队列的第一个元素（head 指出的位置），令为 j。因为 counter[j]为 0（这样 j 才会被放入队列），所以就可以把 j 去掉，亦即把 DELETED 存入 status[j]。不但如此，还把 counter[funct[j]]减去 1。注意，funct[j]就是 j 所对应的元素。如果减了 1 之后，会使 funct[j]孤零零一个，就把 funct[j]放到队列最后面。就这样一直反复，当队列中不再有元素，就找到答案了。换言之，status[ ]中记录着 SAVED 的元素就是 S'中的内容。

【习题】

（1）为什么当队列没有元素了，就找到了答案呢？请证实这一点。

（2）把队列换成堆栈，效果一样吗？请证明此想法。

（3）请问 counter[ ]中的元素，在最坏的情况下，会被这个程序处理多少次？

（4）既然 S 是一个集合，那么把这个程序中有关 S 的部分不用数组，而改成用数元字段来仿真集合，程序会比较容易写吗？效果会提高吗？

（5）请问，S'是不是一定找得出来呢？这是一个问题的边界条件，好程序员不得不留心。如果一定可以找出 S'，为什么？如果有可能找不出 S'，也就是 status[ ]中每一个元素都是 DELETED，请举一个找不出来的例子。

（6）找出来的 S'中的元素要是最多的，但上面的说明中都没有讲到这一点，能够看出这一点吗？有没有可能有另一个不同的 S'存在呢？

Download at http://www.pin5i.com/

【参考文献】

无法查出这个问题的起源，但相信是有很长一段时间了，因为它流传很广。此处的解法采自 Udi Manber 教授的一篇很有趣的文章：

U.Manber, Using Induction to Design Algorithms, *Communications of ACM*. Vol.31(1988), pp.1300～1313

【程序】

程序 18-10（1TO1.C）

```
/* ------------------------------------------------------------ */
/* FUNCTION find_one_to_one :                                   */
/*   This function accepts accepts a table representing         */
/* a function and then construct a one-to-one function          */
/* from the given function table.  funct[] is the given         */
/* function table representing a function from [0,n-1] to       */
/* [0,n-1].  status[] is an output vector containing SAVED      */
/* or DELETED, where SAVED means the corresponding point        */
/* should be kept while DELETED indicates it has to be          */
/* deleted.  counter[] is a working array.                      */
/*   This is a rather complicated program.  Read my book        */
/* before try to study it please.                               */
/*                                                              */
/*                                                              */
/* Copyright Ching-Kuang Shene          July/01/1989            */
/* ---------------------------------------------------          */

#define  QUEUE_SIZE   100        /* size of a queue              */
#define  SAVED          1
#define  DELETED       -1

void find_one_to_one(int  funct[], int status[], int counter[], int n)
{
     int  queue[QUEUE_SIZE];     /* we need a queue              */
     int  head, tail;            /* queue pointers               */
     int  i, j;

     for (i = 0; i < n; i++) {   /* initialization               */
         counter[i]  = 0;        /* size of inverse-images       */
         status[i]   = SAVED;    /* assume all are SAVED         */
     }

     for (i = 0; i < n; i++)     /* count inverse-image size     */
         counter[funct[i]]++;
```

Download at http://www.pin5i.com/


```
    for (tail = -1, i = 0; i < n; i++) /* put all i such        */
        if (counter[i] == 0)    /* that counter[i]=0 to Q       */
            queue[++tail] = i;

    head = 0;                   /* main loop. start from H      */
    while (head <= tail) {      /* if there have elements       */
        j = queue[head++];      /* get it and put it to j       */
        status[j] = DELETED;    /* delete it. no inv-image      */
        if (--counter[funct[j]] == 0)
            queue[++tail] = funct[j];
    }
}
```

问题 9.11　寻找支配元素（DOMINATR.C）

已知一个整数数组 x[ ]，对于任何一个元素 x[i]而言，在 x[i]右边，而且值不比 x[i]小的元素（亦即 j>i 而且 x[j]>=x[i]的 x[j]）都叫做 x[i]的右支配元素（Right Dominator）。在 x[i]所有的右支配元素中最靠近 x[i]的那一个，就叫做最靠近的右支配元素（Right Nearest Dominator）。当然，不见得每一个元素都有支配元素的，例如最大元素就没有。这个问题是，写一个程序，接收一个数组，把数组中每一个元素的最靠近的右支配元素找出来。

【解答】

身为程序员，常常要记住一点，就是如果在程序中有可用的信息，就要充分利用它，这样程序的效率才会提高。那么这个程序中有什么可用的信息呢？注意，当自 x[0],x[1],…，一个接一个元素地查过去而到达 x[i]时，却发现 x[i]<=x[i+1]，于是 x[i+1]就至少会是 x[i]的最靠近右支配元素，但若 x[i-1]>x[i]，那么 x[i+1]也有可能会是 x[i-1]的最近右支配元素（为什么？）。若 x[i]>x[i+1]，这就表示 x[i]的最靠近右支配元素还没有决定，因此去查 x[i+2]，一旦发现 x[i+1]<=x[i+2]，如上所述，不但 x[i+1]最靠近右支配元素被找了出来，x[i]的，甚至于 x[i-1]的（只要 x[i-1]>x[i]）也一并出现。因此 x[i]>x[i+1]就是一个关键因素，而问题说明中的做法就忽略了这一点。

当 x[i]>x[i+1]时，x[i]的最靠近右支配元素还没有出现，所以不妨把 x[i]先保留起来，继续处理 x[i+2]。若 x[i]<=x[i+1]，x[i+1]就是 x[i]的最靠近右支配元素，但是，如果在处理 x[i]之前因为 x[i-1]>x[i]而把 x[i-1]保留了起来，若 x[i-1]也大于等于 x[i]的话，那么若 x[i-1]<=x[i+1]时，x[i-1]的最靠近右支配元素也正是 x[i+1]。

因此综合起来，可以用一个堆栈来记录那些还不能找出最靠近右支配元素的 x[i]，做法分成下面几个步骤。程序如下所示：

【程序】

程序 18-11

```
把 x[0]放入堆栈;
for (i = 1; i < n; i++) {
```

Download at http://www.pin5i.com/


```
        while（堆栈还有元素，而且顶端 <= x[i]） {
            如果顶端元素是 x[k]，当然 k < I;
            因此 x[k]的最靠近右支配元素为 x[i];
            把堆栈顶端丢掉;
        }
        把 x[i]放入堆栈
    }
```

这个做法与上面的讨论相吻合，把每一个元素一一查过去，n 是元素个数，在开始时堆栈顶端只有 x[0]。堆栈中的元素就是那些未曾决定最靠近右支配元素的 x[k]；当然 k 一定小于 i，因为是从头向尾部查的。

对于任何一个 x[i]而言，拿堆栈顶端的元素与 x[i]相比较，当它比 x[i]大，x[i]就是它的最靠近右支配元素。这样顶端元素的最靠近右支配元素就找出来了，所以把它去掉，继续处理堆栈的下一个元素。这个操作就一直反复，直到堆栈中没有元素，或顶点的元素大于 x[i]为止，因此 x[i]就不可能是顶点元素的最靠近右支配元素了，所以把 x[i]也放入堆栈。由此看来，堆栈中的元素从顶点往底部看，值是愈来愈大的（为什么？）。这个操作一直反复到所有元素都处理完为止，残留在堆栈中的元素就都是没有最靠近右支配元素的（为什么？）。

【习题】

（1）如果现在处理到 x[i]，而且发现 x[i]<=x[i+1]，请问在什么条件下 x[i-1]的最靠近右支配元素也是 x[i+1]？

（2）现在处理到 x[i]，试证明下面两种情况：

① 在堆栈中的元素 x[k]都满足 k<i。

② 在堆栈中自顶端向底部的元素是从小到大排列的。

（3）为什么所有元素都处理完后，在堆栈中的元素都不会有最靠近的右支配元素呢？

（4）这个程序最多会作多少次比较？笔者认为与 n 成正比，但在最坏的情况下究竟是多少呢？

（5）请定义什么是最靠近左支配元素，写个程序把它们找出来。请问，有没有办法同时找出最靠近的右与左支配元素呢？

【参考文献】

这个题目与解法来自 Hubert Wagener 在 1988 年的一篇文章。Wagener 是用这个观点来解决一个计算几何（Computational Geometry）中的重要问题，不过他的重点是平行式的算法，但是他的文章中两个解法都有。有兴趣的朋友可以参看：

H.Wagener. Triangulating a Monotone Polygon in Parallel. *Computational Geometry and Its Applications*, Lecture Notes in Computer Science, No.333, Springer–Verlag. 1988, pp.136～147

【程序】

程序 18-12（DOMINATR.C）

```
/* ------------------------------------------------------------------ */
/* FUNCTION nearest_dominator :                                       */
```

Download at http://www.pin5i.com/


```
/*   Given an array, this function computes the right          */
/* nearest dominator of each element in the array.  A          */
/* right dominator of x[i] is any element x[j] with j > i      */
/* and x[i] <= x[j].  The right nearest dominator is the       */
/* right dominator with smallest index.  You can define        */
/* the left nearest dominator in a similar way.                */
/*                                                             */
/* Copyright Ching-Kuang Shene          July/01/1989           */
/* ----------------------------------------------------------- */

#define  STACK_SIZE  100
#define  YES          1
#define  NO           0


void nearest_dominator(int x[], int dom[], int n)
{
     int stack[STACK_SIZE];      /* the working stack          */
     int top;                    /* the stack pointer          */
     int i;                      /* working variables          */

     for (i = 0; i < n; i++)     /* clear dominator array      */
         dom[i] = -1;            /* -1 means no dominator      */

     top = 0;                    /* stack initialization       */
     stack[top]  = 0;
     for (i = 1; i < n; i++) {   /* for each element :         */
         while (top >= 0 && x[stack[top]] <= x[i])
             dom[stack[top--]] = i;
         stack[++top] = i;       /* push i into stack          */
     }
}
```

问题 9.12　最长递增部分序列（LIS.C）

已知一个整数数组，在这个数组中的一个递增序列，就是把若干元素删除后所留下的元素是按从小到大的顺序排列的。举例而言，如果数组是 1,3,5,7,8,2,9,4,10,6，那么 1,5,8,9,10 是一个递增序列，3,5,7,8,9,10 也是一个递增序列，1,3,5,7,8,9,10 也是递增序列。简单地说，可以依数组元素的次序选出若干元素，使得从小到大排好，那么这就是个递增序列。以下是更多的几个例子，比如 3,8；2,9；7,10；2,4,6；…。但是 9,4,10；5,7,8,2,9 等却不是递增

Download at http://www.pin5i.com/

序列。问题是，写一个程序接收一个整数数组，找出这个数组中最长的递增序列的长度。以上面的例子来看，1,3,5,7,8,9,10 是最长的递增列，所以程序就输出 7 的结果。

【解答】

注意，当查到第 i 个元素时，已经找出来了的任何递增序列的长度绝对不会超过 i，因此最长的那一个也不能超过 i。在程序中，用一个变量 length 记录到目前为止所找出来的最长递增序列的长度。另外，也准备一个数组 last[ ]，用 last[j]表示长度为 j 的递增序列中最后一个元素的值。但遗憾的是，长度为 j 的递增序列不止一个，那么要保存哪一个呢？这就有学问了。保存最小的那一个。例如，如果到目前为止已经依次处理过 1,2,6,4,7 这 5 个元素了，长度为 3 的递增序列有 1-2-6,1-2-4,1-2-7,1-6-7,1-4-7 这 5 个，但是以 1-2-4 的最后一个元素值为最小，所以 last[3]的值就是 4。为什么要这么做呢？理由是很简单的，因为最后一个元素越小，那么这个递增序列在往后被延长的机会就越大。例如，如果下一个元素是 5，那么 1-2-6,1-2-7,1-6-7,1-4-7 就不能再延长，但 1-2-4 却可以把 5 接在后面而成为 1-2-4-5，长度变成 4。

当然，在一开始时处理第一个元素 x[1]，到目前为止，惟一的一个递增序列就是 x[1]，长度为 1，所以 length 的值为 1，而 last[1]也是 1。接下来就是 2,3,4…地一个接一个元素查过去，当查到第 i 个元素 x[i]时可能有以下几种情况。

（1）如果 x[i]比目前最长的递增序列的最后一个元素大或相等，那么 x[i]就可以接在后面，而使原来的最长递增序列的长度加上 1，当然还是最长的。最长的递增序列的最后一个元素在哪呢？不是在 last[leng th]吗？于是长度加上 1 而加长了原来的递增序列之后，x[i]就在那个序列的最后面，所以得把 length 加上 1，再把 x[i]存入 last[leng th]。

（2）x[i]比 last[1]小。last[1]是长度为 1 的递增序列的惟一元素（最小值），现在 x[i]比它还小，如果把 x[i]看成是长度为 1 的递增序列的话，它会被延长的潜力就大多了。

（3）x[i]<last[length]，而且 x[i]>=last[1]。last[ ]数组中的元素是从小到大“自动”地排列好的，从定义与工作过程可以看出来，不过并不十分明显而已，这个证明就留作习题。既然如此，就可以在 last[1],last[2],…,last[length]中找出一个 j，使得 last[j-1]<=x[i]<last[j]，x[i]比某个长度为 j-1 的递增序列的最后一个大，因此 x[i]可以补在该序列后面而得到一个长度为(j-1)+1=j 的递增序列，而 x[i]是它的最后一个元素。但是，因为 x[i]<last[i]表示 x[i]比原来某个长度为 i 的递增序列的最后一个元素小，上述新得出的有 j 个元素的递增序列的最后一个元素（x[i]）就比原来记录下来的长度为 j 的序列的最后一个元素小，依照定义，x[i]就应该存到 last[j]中。这一步的意义，就是指出新产生的长度为 j 的序列的最后一个元素比原来的小，因此被（用 x[i]）更新了。

反复做下去，当所有元素都查完之后，length 中就是最长的序列的长度，而 last[i]就是长度为 i 的递增序列的最后一个元素。另外，上述的第三步中提到要找出一个 j，使得 last[j-1]<=x[i]<last[j]，可以用二分查找法做。因为这是件简单的工作，所以就不多作说明了。

这个方法一共处理了多少个 x[ ]中的元素呢？当处理到 x[i]时，在最坏的情况下，会做到第三种情形，而在 last[1]～last[length]之间用二分查找法找出一个 j，使得 last[j]<=x[i]<last[j]。因为 length<=i，所以不妨看成在 i 个数据中采用二分查找法，于是大约要用 $2\log_2 i$

Download at http://www.pin5i.com/

次比较，但是因为 i 可以是 1,2,…,n，所以最坏情况下的比较次数绝对不会多过下式：

$$
\begin{aligned}
&2\log_2 1 + 2\log_2 2 + \cdots + 2\log_2 i + \cdots + 2\log_2 n \\
&= 2\sum_{i=1}^{n} \log_2 i \\
&\leqslant 2\sum_{i=1}^{n} \log_2 n \quad （因为 \log_2 i \leqslant \log_2 n） \\
&= 2n\log_2 n
\end{aligned}
$$

所以说，比较的次数与 $n\log_2 n$ 成正比，n 是元素个数，此方法用了多少次比较呢？

【习题】

（1）请证明 last[ ]中的元素是从小到大排列的。

（2）如果用 1,3,5,7,8,2,9,10,6 作输入，请用手算方式验证一下 last[ ]的值在最后是 1,2,4,6,8,9,10，而最长的递增序列长度是 7。

（3）请找出一组输入，使得它有两组长度相同的最长递增序列。请说明为什么最长递增序列的长度值只会有一个。

（4）这个程序只求出了长度，而没有找出有这个长度的递增序列，请修改程序把序列找出来。

（5）请修改程序，把最长递降序列的长度找出来，如果同时要找出最长的递增、递降序列的长度，而不是分开来做，请问程序会不会非常复杂？

（6）（十分简单，请马上回答）求最短递增、递降序列有没有困难，为什么？

（7）已知一个整数数组，请把它分解成最少数目的递增序列。

提示：对于数组中的每一个元素，都保存在原数组中到它为止的递降序列的最大长度，再用本题来解。

【参考文献】

依照 David Gries 的记录，这个问题是 1978 年由 Dijkstra 提出来的，可参看 The Science of Programming 一书（Gries）的第 303 页中的简史。此处的解法就是以 Gries 的做法为蓝本写成的。在 1980 年 Dijkstra 以这个题目为例，讲解如何使用数学归纳法来发展出程序，并且证明这个算法是正确无误的。1982 年 Robert B.K.Dewar、Susan M. Merritt 与 Micha Sharir 有一篇后续的文章，用另一个角度来看 Dijkstra 的问题，并且得到另外两个类似解法，但他们的程序却是用 SETL 语言写的。总之，David Gries 的书与 Dijkstra 的文章可读性很高，有兴趣的话，看看 Dewar 等人的文章也十分有益。最后，习题(7)采自 1985 年 Anne Kaldewaij 的文章。

1 R.B.K.Dewar, S.M.Merritt and M. Sharir. Some Modified Algorithms for Dijkstra's Longest Upsequence Problem, *Acta Informatica*. Vol.18(1982), pp.1～15

2 E.W.Dijkstra. Some Beautiful Arguments Using Mathematical Induction, *Acta Mathematica*. Vol.13(1980), pp.1～8

3 D.Gries. *The Science of Programming*, Springer-Verlag. 1981

Download at http://www.pin5i.com/

4 A.Kaldawaij. On the Decomposition of Sequences into Ascending Subsequences, *Information Processing Letters*. Vol.21(1985), pp.69

【程序】

程序 18-13（LIS.C）

```
/* ------------------------------------------------------------ */
/* FUNCTION longest increasing subsequence :                    */
/*   Given an integer sequence of length n, this function       */
/* determines the longest increasing subsequence in nlogn       */
/* stpes with a dynamic programming technique.                  */
/* ------------------------------------------------------------ */

#include <stdlib.h>              /* for malloc() and free()     */

int longest_inc_sequence(int x[], int n)
{
    int *last;                   /* the last of a n-seubseq.    */
    int length;                  /* current max inc. subseq.    */
    int left;                    /* left bound for bin search   */
    int right;                   /* right bound                 */
    int mid;                     /* the middle point.           */
    int i;

    last   = (int *) malloc(sizeof(int)*n);   /* get work mem */

    last[0] = x[0];              /* start from the beginning    */
    length  = 0;                 /* len=1 and x[0] in subseq    */
    for (i = 1; i < n; i++)      /* scan all other items        */
        if (x[i] >= last[length])  /* higher than max?          */
            last[++length] = x[i]; /* YES, record it.           */
        else if (x[i] < last[0])   /* NO, less then 1st         */
            last[0] = x[i];        /* it would be the new head*/
        else {                   /* now insert it with bin.sr   */
            for (left = 0, right = length; left != right-1; ) {
                mid = (left + right) / 2;
                (last[mid] <= x[i]) ? (left = mid) : (right = mid);
            }
            last[right] = x[i];
        }
    free(last);                  /* finally release memory      */
    return length + 1;           /* return the true length      */
}
```

Download at http://www.pin5i.com/

参考文献

书籍部分

1 ACM, Collected Algorithms from ACM, Vol. I and II, ACM, 1980

2 A. V. Aho, J. E. Hopcroft and J. D. Ullman, The Design and Analysis of Computer Algorithms, Addison-Wesley, 1976

3 A. V. Aho, R. Sethi and J. D. Ullman, Compilers: Principles, Technique, and Tools, Addison-Wesley, 1986

4 A. Apostolico and Z. Galil (ed.), Combinatorial Algorithms on Words, Springer-Verlag, 1985

5 S. Baase, Computer Algorithms, Introduction to Design and Analysis, 2nd ed., 1988

6 W. W. Rouse Ball and H. S. M. Coxeter, Mathematical Recreations and Essays, 13th ed., Dover, 1987

7 F. L. Bauer and K. Samelson (eds.), Language Hierarc ies and Interfaces, Lecture Notes in Computer Science, No. 46, Springer-Verlag, 1976

8 J. Bentley, Programming Pearls, Addison-Wesley, 1986

9 E. R. Berlekamp, J. H. Conway and R. K. Guy, Winning Ways, Vol. 2: Games in Particular, Academic Press, 1982

10 P. Berlioux and P. Bizard, Algorithms, The Construction, Proof, and Analysis of Programs, John Wiley and Sons, 1986

11 N. L. Biggs, E. K. Lloyd and R. J. Wilson, Graph Theory 1736-1936, Oxford University Press, 1976

12 G. Brassard and P. Bratley, Algorithmics: Theory and Practice, Prentice-Hall, 1988

13 D. M. Bressoud, Factorization and Primality Testing, Springer-Verlag, 1989

14 N. Christofides, Graph Theory, An Algorithmic Approach, Academic Press, 1975

15 R. Courant and H. Robbins, What is Mathematics, Oxford University Press, 1941

16 O. -J. Dahl, E. W. Dijkstra and C. A. R. Hoare, Structured Programming, Academic Press, 1972

17 E. W. Dijkstra, Notes on Structured Programming, Technical University Eindhoven, 1970

18 E. W. Dijkstra, A Discipline of Programming, Prentice-Hall. 1976

19 E. W. Dijkstra, Selected Writings on Computing: A Personal Perspective, Springer-Verlag, 1982

Download at http://www.pin5i.com/

20 S. Even, Algorithmic Combinatorics, Macmillan, 1973

21 M. Gardner, Knotted Doughnuts and Other Mathematical Entertainments, W. H. Freeman and Company, 1986, pp.11～27

22 M. Gondran and M. Minoux, Graphs and Algorithms, translated by S. Vajda, John Wiley & Sons, 1984

23 F. Gray, Pulse Code Communication, US Patent 2632058 (March 1953)

24 D. Gries, The Science of Programming, Springer-Verlag, 1981

25 D. Gusfield and R. W. Irving, The Stable Marriage Problem: Structure and Algorithms, The MIT Press, 1989

26 E. Horowitz and S. Sahni, Fundamental of Computer Algorithms, Computer Science Press, 1978

27 D. E. Knuth, The Art of Computer Programming, Vol. 2/Seminumerical Algorithms, 2nd. ed., Addison-Wesley, 1981

28 D. E. Knuth, The Art of Computer Programming, Vol. 3/Sorting and Searching, Addison-Wesley, 1973

29 D. E. Knuth, Mariages Stables, Les Presses de 1' Universite Montreal, 1976

30 M. Kraitchik, Mathematical Recreations, 2nd ed., Dover, 1953

31 A. D. McGettrick and P. D. Smith, Graded Problems in Computer Science, Addison-Wesley, 1983

32 U. Manber, Introduction to Algorithms, A Creative Approach, Addison-Wesley, 1989

33 H. A. Maurer and M. R. Williams, A Collection of Programming Problems and Techniques, Prentice-Hall, 1972

34 A. Nijenhuis and H. S. Wilf, Combinatorial Algorithms, 2nd ed., Academic Press, 1978

35 E. M. Reingold, J. Nievergelt and N. Deo, Combinatorial Algorithms: Theory and Practice, Prentice-Hall, 1977

36 J. Reynolds, The Craft of Programming, Prentice-Hall, 1981

37 P. Ribenboim, The Book of Prime Number Records, 2nd. ed., Springer-Verlag, 1989

38 D. Sankoff and J. B. Kruskal (ed.), Time Warps, String Edits, and Macro- molecules: The Theory and Practice of Sequence Comparison, Addison-Wesley, 1983

39 R. Sedgewick, Quicksort, Garland Publishing, New York, 1980

40 R. Sedgewick, Algorithms, 2nd ed., Addison-Wesley, 1988

41 D. Stanton and D. White, Constructive Combinatorics, Springer-Verlag, 1986

42 H. S. Wilf, Algorithms and Complexity, Prentice-Hall, 1986

43 N. Wirth, Systematic Programming-An Introduction, Prentice-Hall, 1973

44 N. Wirth, Algorithm and Data Structure, Prentice-Hall, 1986

45 A. M. Yaglom and I. M. Yaglom, Challenging Mathematical Problems with Elementary Solutions, Vol. I: Combinatorial Analysis and Probability Theory, Dover, 1987

Download at http://www.pin5i.com/

论文部分

1 B. Abramson and M. Yung, Divide and Conquer under Global Constraints: A Solution to the N-Queens Problem, Journal of Parallel and Distributed Computing, Vol. 6 (1989), pp.649～662

2 A. Aggarwal, M. M. Klawe, S. Moran, P. Shor and R. Wilber, Geometric Applications of a Matrix-Searching Algorithm, Algorithmica, Vol. 2 (1987), pp.195～208

3 A. Aggarwal and D. Kravets, A Linear Time Algorithm for Finding All Farthest Neighbors in a Convex Polygon, Information Processing Letters, Vol. 31 (1989), pp.17～20

4 L. Allison and T. I. Dix, A Bit-String Longest-Common-Subsequence Algorithm, Information Processing Letters, Vol. 23 (1986), pp.305～310

5 A. Apostolico and F. P. Preparata, Optimal Off-Line Detection of Repetitions in a String, Theoretical Computer Science, Vol. 22 (1983), pp.297～315

6 M. J. Atallah and S. R. Kosaraju, An Efficient Algorithm for Maxdominance, with Applications, Algorithmica, Vol. 4 (1989), pp.221～236

7 R. Backhouse, Writing a Number as the Sum of Two Squares: A New Solution, Information Processing Letters, Vol. 14 (1982), pp.15～17

8 T. A. Bailey and R. G. Dromey, Fast String Searching by Finding Subkeys in Subtext, Information Processing Letters, Vol. 11 (1980), pp.130～133

9 G. Barth, An Analytical Comparison of Two String Searching Algorithms, Information Processing Letters, Vol. 18 (1984), pp.249～256

10 P. B. Borwein, On the Complexity of Calculating Factorials, Journal of Algorithms, Vol. 6 (1985), pp.376～380

11 R. S. Boyer and J. S. Moore, A Fast String Searching Algorithm, Communica- tions of ACM, Vol. 20 (1977), pp.762～772

12 S. Carlsson, A Variant of Heapsort with Almost Optimal Number of Comparisons, Information Processing Letters, Vol. 24 (1987), pp.247～250

13 L. Chang and J. F. Korsh, Canonical Coin Changing and Greedy Solutions, Journal of ACM, Vol. 23 (1976), pp.418～422

14 S. K. Chang and A. Gill, Algorithmic Solution of the Ching-Making Problem, Journal of ACM, Vol. 17 (1970), pp.113～122

15 B. A. Chartres, Algorithm 310, Prime Number Generator, Communications of ACM, Vol. 10 (1967), pp.569

16 B. A. Chartres, Algorithm 311, Prime Number Generator, Communications of ACM, Vol. 10 (1967), pp.570

Download at http://www.pin5i.com/

17 Francis Y. Chin, An O (n) Algorithm for Determining a Near-Optimal Computation Order of Matrix Chain Products, Communications of ACM, Vol. 21 (1978), pp.544～549

18 C. R. Cook and D. J. Kim, Best Sorting Algorithm for Nearly Sorted Lists, Communications of ACM, Vol. 23 (1980), pp.620～624

19 M. Crochemore, An Optimal Algorithm for Computing the Repetitions in a Word, Information Processing Letters, Vol. 12 (1981), pp.244～250

20 P. Cull and L. Holloway, Computing Fibonacci Numbers Quickly, Information Processing Letters, Vol. 32 (1989), pp.143～149

21 R. B. K. Dewar, S. M. Merritt and M. Sharir, Some Modified Algorithms for Dijkstra's Longest Upsequence Problem, Acta Informatica, Vol. 18 (1982), pp.1～15

22 E. W. Dijkstra, Some Beautiful Arguments Using Mathematical Induction, Acta Informatica, Vol. 13 (1980), pp.1～8

23 E. W. Dijkstra, Smoothsort, An Alternative for Sorting in Situ, Science of Computer Programming , Vol. 1 (1982), pp.223～233

24 B. Djokic, M.. Miyakawa, S. Sekiguchi, I. Semba and I. Stojmenovic, A Fast Iterative Algorithm for Generating Set Partitions, The Computer Journal, Vol. 32 (1989), pp.281～282

25 W. Dobosiewicz, An Efficient Variation of Bubble Sort, Information Processing Letters, Vol. 11 (1980), pp.5～6

26 J. Ebert, Computing Eulerian Trails, Information Processing Letters, Vol. 28 (1988), pp.93～97

27 G. Ehrlich, Loopless Algorithms for Generating Permutations, Combinations, and Other Combinatorial Configurations, Journal of ACM, Vol. 20 (1973), pp.500～513

28 M. C. Er, A Note on Generating Well-formed Parenthesis Strings, Lexioographically, The Computer Journal, Vol. 26 (1983), pp.205～207

29 M. C. Er, A Fast Algorithm for Computing Order-K Fibonacci Numbers, The Computer Journal, Vol. 26 (1983), pp.224～227

30 M. C. Er, Computing Sums of Order-k Fibonacci Numbers in LOG Time, Information Processing Letters, Vol. 17 (1983), pp.1～5

31 M. C. Er, The Towers of Hanoi and Binary Numerals, Journal of Information & Optimization Sciences, Vol. 6 (1985), pp.147～152

32 M. C. Er, A Formal Derivation of an O (logn) Algorithm for Computing Fibonacci Numbers, Journal of Information & Optimization Sciences, Vol. 7 (1986), pp.9～15

33 M. C. Er, A Fast Algorithm for Generating Set Partitions, The Computer Journal, Vol. 31 (1988), pp.283～284

34 L. Euler, Leonhard Euler and The Königsberg Bridges, Scientific American, Vol. 189 (1953), July, pp.66～70

35 B. -J. Falkowski and L. Schmitz, A Note on The Queens'Problem, Information

Download at http://www.pin5i.com/

Processing Letters, Vol. 23 (1986), pp.39～46

36 M. J. Fischer and S. L. Salzberg, Solution to Problem 81-5, Journal of Algorithms, Vol. 3 (1982), pp.375-379

37 R. Floyd, Algorithm 97: Shortest Path, Communications of ACM, Vol5 (1962), pp.345

38 D. A. Fowler, The Königsberg Bridges-250 Years Later, American Mathematical Monthly, Vol. 95 (1988), pp.42～43

39 D. Gale and L. S. Shapley, College Admissions and the Stability of Marriage, American Mathematical Monthly, Vol. 69 (1962), pp.9～15

40 Z. Galil, On Improving the Worst Case Running Time of the Boyer-Moore String Matching Algorithm, Communications of ACM, Vol. 22 (1979), pp.505～508

41 M. Gardner, Mathematical Games, Scientific American, Vol. 277 (1972), August, pp.106～109

42 E. N. Gilbert, Gray Codes and Paths on the n-Cube, The Bell System Technical Journal, Vol. 37 (1958), pp.815～826

43 D. Gries, A Note on a Standard Strategy for Developing Loop Invariants and Loops, Science of Computer Programming, Vol. 2 (1982), pp.207～214

44 D. Gries and G. Levin, Computing Fibonacci Numbers (and Similarly Defined Function) in LOG Time, Information Processing Letters, Vol. 11 (1980), pp.68～69

45 D. Gries and J. Misra, A Linear Sieve Algorithm for Finding Prime Numbers, Communications of ACM, Vol. 21 (1978), pp.999～1003

46 R. Grossi and F. Luccio, Simple and Efficient String Matching with k Mismatches, Information Processing Letters, Vol. 33 (1989/1990), pp.113～120

47 L. Guibas and R. Lipton, Problem 80-4, Journal of Algorithms, Vol. 1 (1980), pp.108

48 L. Guibas and A. M. Odlyzko, A New Proof of the Linearity of the Boyer- Moore String Searching Algorithm, SIAM Journal on Computing, Vol. 9 (1980), pp.672～682

49 C. C. Handley, An in Situ Distribution Sort, Information Processing Letters, Vol. 23 (1986), pp.265～270

50 M. Harrison, Problem 80-3, Journal of Algorithms, Vol. 1 (1980), pp.108

51 S. Hertel, Smoothsort's Behavior on Presorted Sequences, Information Processing Letters, Vol. 16 (1983), pp.165～170

52 D. S. Hirschberg, A Linear Space Algorithm for Computing Maximal Common Subsequence, Communications of ACM, Vol. 18 (1975), pp.341～343

53 D. S. Hirschberg, An Information-Theoretic Lower Bound for the Longest Common Subsequence Problem, Information Processing Letters, Vol. 7 (1978), pp.40～41

54 C. A. R. Hoare, Quicksort, The Computer Journal, Vol. 5 (1962), pp.10～15

55 T. C. Hu and M. T. Shing, An O (n) Algorithm to Find a Near-Optimum Partition

Download at http://www.pin5i.com/

of a Convex Polyson, Journal of Algorithms, Vol. 2 (1981), pp.121～138

56 T. C. Hu and M. T. Shing, Computation of Matrix Chain Products, I, SIAM Journal on Computing, Vol. 11 (1982), pp.362～373

57 T. C. Hu and M. T. Shing, Computation of Matrix Chain Products, II, SIAM Journal on Computing, Vol. 13 (1984), pp.3228～251

58 B. -C. Huang and D. E. Knuth, A One-Way, Stackless Quicksort Algorithm, BIT, Vol. 26 (1986), pp.127～130

59 J. Incerpi, Practical Variations of Shellsort Information Processing Letters, Vol. 26 (1987/88), pp.37～43

60 J. Incerpi and R. Sedgewick, Improved Upper Bounds on Shellsort, Journal of Computer and System Sciences, Vol. 31 (1985), pp.210～224

61 R. W. Irving, Permutation Backtracking in Lexicographic Order, The Computer Journal, Vol. 27 (1984), pp.373～375

62 S. M. Johnson, Generation of Permutations by Adjacent Transposition, Mathematics of Computation, Vol. 17 (1963), pp.282～285

63 A. Kaldewaij, On the Decomposition of Sequences into Ascending Subsequences, Information Processing Letters, Vol. 21 (1985), pp.69

64 R. Kaye, A Gray Code for Set Partitions, Information Processing Letters, Vol. 5 (1976), pp.171～173

65 K. N. King and B. Smith-Thomas, An Optimal Algorithm for Sink-Finding, Information Processing Letters, Vol. 14 (1982), pp.109～111

66 D. E. Knuth, J. H. Morris, Jr. and V. R. Pratt, Fast Pattern Matching in Strings, SIAM Journal on Computing, Vol. 6 (1977), pp.323～350

67 H. T. Kung, F. Luccio and F. P. Preparata, On Finding the Maxima of a Set of Vectors, Journal of ACM, Vol. 22 (1975), pp.469～476

68 R. Lesuisse, Some Lessons Drawn from the History of the Binary Search Algorithm, The Computer Journal, Vol. 26 (1983), pp.154～163

69 R. Lowrance and R. A. Wagner, An Extension of the String-to-String Correction Problem, Journal of ACM, Vol. 22 (1975), pp.177～183

70 M. J. Magazine, G. L. Nemhauser and L. E. Trotter, Jr., When the Greedy Solution Solves a Class of Knapsack Problems, Operations Research, Vol. 23 (1975), pp.207～217

71 M. G. Main and R. J. Lorentz, An O (nlogn) Algorithm for Finding All Repetitions in a String, Journal of Algorithms, Vol. 5 (1984), pp.422～432

72 H. G. Mairson, Some New Upper Bounds on the Generation of Prime Numbers, Communications of ACM, Vol. 20 (1977), pp.664～669

73 U. Manber, Using Induction to Design Algorithms, Communications of ACM, Vol. 31 (1988), pp.1300～1313

74 A. J. Martin and M. Rem, A Presentation of The Fibonacci Algorithm, Information

Download at http://www.pin5i.com/

Processing Letters, Vol. 19 (1984), pp.67～68

75 H. Mayer and D. Perkins, Towers of Hanoi Revisited, A Nonrecursive Surprise, SIGPLAN Notices, Vol. 19 (1984), No. 2, pp.80～84

76 K. Mehlhorn, Sorting Presorted Files, Theoretical Computer Science, 4th GI Conference, Aachen, March 26-28, 1979, ed. by K. Weihranch, in Lecture Notes in Computer Science, No. 67, Springer-Verlag, 1979

77 B. Meyer, Incremental String Matching, Information Processing Letters, Vol. 21 (1985), pp.219～227

78 A. Mirzaian, A Halving Technique for the Longest Stuttering Subsequence Problem, Information Processing Letters, Vol. 26 (1987/1988), pp.71～75

79 J. Misra and D. Gries, Finding Repeated Elements, Science of Computer Programming, Vol. 2 (1982), pp.143～152

80 J. Moore, Problem 81-5, Journal of Algorithms, Vol. 2 (1981), pp.208～209

81 D. Motzkin, A Stable Quicksort, Software: Practice and Experience, Vol. 11 (1981), pp.607～611

82 P. Naur, An Experiment on Program Development, BIT, Vol. 12 (1972), pp.347～365

83 J. R. Parker, On Converting Character Strings to Integers, Information Processing Letters, Vol. 19 (1984), pp.17～19

84 A. Pettorossi, Derivation of an O(k^2logn) Algorithm for Computing Order-k Fibonacci Numbers from the O (k^3logn) Matrix Multiplication Method, Information Processing Letters, Vol. 11 (1980), pp.172～179

85 P. Pritchard, A Sublinear Additive Sieve for Finding Prime Number, Communications of ACM, Vol. 24 (1981), pp.18～23

86 P. Pritchard, Explaining the Wheel sieve, Acta Informatica, Vol. 17 (1982), pp.447～485

87 M. Regnier, Knuth-Morris-Pratt Algorithm: An Analysis, Mathematical Foundations of Computer Science 1989, Poraaka-Kozubnik, Poland; Lecture Notes in Computer Science, No. 379, Springer-Verlag, 1989, pp.431～444

88 M. Reichling, A Simplified Solution of the N Queens' Problem, Information Processing Letters, Vol. 25 (1987), pp.253～255

89 M. Rem, Small Programming Exercise, Science of Computer Programming, Vol. 3 (1983), pp.217～222

90 M. Rem, Small Programming Exercise 2, Science of Computer Programming, Vol. 3 (1983), pp.313～319

91 M. Rem, Small Programming Exercise 3, Science of Computer Programming, Vol. 4 (1984), pp.87～94

92 M. Rem, Small Programming Exercise 11, Science of Computer Programming, Vol. 6 (1986), pp.313～318

Download at http://www.pin5i.com/

93 M. Rem, Small Programming Exercise 12, Science of Computer Programming, Vol. 7 (1986), pp.87～97

94 M. Rem, Small Programming Exercise 14, Science of Computer Programming, Vol. 7 (1986), pp.313～319

95 J. S. Rohl, A Faster Lexicographical N Queens Algorithm, Information Processing Letters, Vol. 17 (1983), pp.231～233

96 E. S. Rosenthal, Solution of Problem 80-3, Journal of Algorithms, Vol. 3 (1982), pp.365

97 J. Röhrich, A Hybrid Quicksort with O(n long n) Worst Case Complexity, Information Processing Letters, Vol. 14 (1982), pp.119～123

98 H. Sach, M. Stiebitz and R. J. Wilson, An Historical Note: Euler's Königsberg Letters, Journal of Graph Theory, Vol. 12 (1988), pp.133～139

99 N. Santoro and J. B. Sidney, Interpolation-Binary Search, Information Processing Letters, Vol. 20 (1985), pp.179～181

100 R. Sedgewick, The Analysis of Quicksort Programs, Acta Informatica, Vol. 7(1977), pp.327～355

101 R. Sedgewick, Quicksort with Equal Keys, SIAM Journal on Computing, Vol. 6 (1977), pp.240～267

102 R. Sedgewick, Permutation Generation Methods, Computing Surveys, Vol. 9(1977), pp.137～164

103 R. Sedgewick, Implementing Quicksort Programs, Communications of ACM, Vol. 21 (1978), pp.847～857

104 R. Sedgewick, A New Upper Bound for Shellsort, Journal of Algorithms, Vol. 7 (1986), pp.159～173

105 A. Shamir, Factoring Numbers in O(log n) Arithmetic Steps, Information Processing Letters, Vol. 8 (1979), pp.28～31

106 M. I. Shamos, Geometry and Statistics; Problems at the Interface, in Algorithms and Complexity, New Directions and Recent Results, edited by J. F. Traub, Academic Press, 1976

107 D. L. Shell, A High-Speed Sorting Procedure, Communications of ACM, Vol. 2 (1959), No. 2, pp.30～32

108 J. Shortt, An Iterative Program to Calculate Fibonacci Numbers in O (log n) Arithmetic Operations, Information Processing Letters, Vol. 7 (1978), pp.299～303

109 R. C. Singleton, Algorithm 356, A Prime Number Generator, Communica tions of ACM, Vol. 12 (1969), pp.563

110 R. C. Singleton, Algorithm 357, A Prime Number Generator, Communica tions of ACM, Vol. 12 (1969), pp.563～564

111 H. W. Six and L. Wegner, EXQUISIT: Applying Quicksort to External Files, Proceedings Nineteenth Annual Allerton Conference on Communication, Control and

Download at http://www.pin5i.com/

Computing, Sept. 30, Oct, 102, 1981, pp.348～354

112 A. O. Slisenko, Detection of Periodicities and String-Matching in Real Time, Journal of Soviet Mathematics, Vol. 22 (1983), pp.1316～1387

113 B. R. Stonebridge, The Knight's Tour of a Chessboard, Mathematical Spectrum, Vol. 19 (1986/87), No. 3, pp.83～89

114 D. F. Stout, Solution to Problem 80-4, Journal of Algorithms, Vol. 4 (1983), pp.176

115 R. W. Topor, Fundamental Solutions of the Eight Queens Problem, BIT, vol. 22 (1982), pp.42～52

116 F. J. Urbanek, An O(log n) Algorithm for Computing the nth Element of the Solution of a Difference Equation, Information Processing Letters, Vol. 11 (1980), pp.66～69

117 K. -P. Vo, Solution to Problem 80-4, Journal of Algorithms, Vol. 3 (1982), pp.366

118 H. Wagener, Triangulating a Monotone Polygon in Parallel, Computational Geoetry and Its Applications. Lecture Notes in Computer Science, No. 333, Springer-Verlag, 1988, pp.136～147

119 R. A. Wagner and M. J. Fischer, The String-to-String Correction Problem, Journal of ACM, Vol. 21 (1974), pp.168～173

120 L. M. Wegner, Sorting a Linked List with Equal Keys, Information Processing Letters, Vol. 15 (1982), pp.205～208

121 M. A. Weiss, Bad Cases for Shaker-Sort, Information Processing Letters, Vol. 28 (1988), pp.133～166

122 M. A. Weiss, R. Sedgewick, E. Hentschel and A. Pelin, Shellsort and The Frobenius Problem, Congressus Numerantium, Vol. 65 (1988), December, pp.253～260

123 M. A. Weiss, A Good Case for Shellsort, Congressus Numerantium, Vol. 73(1990), pp.59～61

124 J. W. J. Williams, Algorithm 232: Heapsort, Communications of ACM, Vol. 7 (1964), pp.347～348

125 R. J. Wilson, An Eulerian Trail Through Königsberg, Journal of Graph Theory, Vol. 10 (1986), pp.265～275

126 T. C. Wilson and J. Shortt, An O (log n) Algorithm for Computing General Order-k Fibonacci Numbers, Information Processing Letters, Vol. 10 (1980), pp.68～75

127 N. Wirth, Program Development by Stepwise Refinement, Communications of ACM Vol. 14 (1971), pp.221～227

128 J. W. Wright, The Change-Making Problem, Journal of ACM, Vol. 22 (19), pp.125～128

129 F. Frances Yao, Speed-Up in Dynamic Programming, SIAM Journal on Algebraic and Discrete Methods, Vol3 (1982), pp.532～540

Download at http://www.pin5i.com/

[General Information]
书名= C 语言名题精选百则技巧篇
作者= 冼镜光编著
页数= 491
SS 号= 11422183
出版日期= 2005 | 出版社= 机械工业出版社

Download at http://www.pin5i.com/

封面
书名
版权
前言
目录
序
前言
第一部分 问题
第1章 序曲
问题1.1 最长平台(PLATEAU.C)
问题1.2 支配值数目(GT_COUNT.C)
问题1.3 等值数目(EQ_COUNT.C)
问题1.4 两数组最短距离(MINDIST.C)
问题1.5 等值首尾和(HEADTAIL.C)
第2章 数字问题
问题2.1 Armstrong数(ARMS1.C,ARMS2.C)
问题2.2 数字谜(TRENTE.C)
问题2.3 求质数(PRIME1.C)
问题2.4 筛法(SIEVE.C)
问题2.5 线性筛法(L_SIEVE.C)
问题2.6 因子分解(FACTOR.C)
问题2.7 数值自乘递归解(R_POWER.C)
问题2.8 数值自乘非递归解(I_POWER.C)
问题2.9 Fibonacci数非递归解(FIB_IT.C)
问题2.10 快速Fibonacci数算法(FIB_MT.C)
问题2.11 扩充Fibonacci数(EX_FIB.C)
问题2.12 二项式系数加法解(CNR_ADD.C)
问题2.13 快速二项式系数算法(CNR_LOG.C)
问题2.14 快速阶乘运算(FACTLOG2.C)
问题2.15 更快的阶乘算法(FACTLOG.C)
问题2.16 连续整数的固定和(GIVENSUM.C)
第二部分 解答
第10章 序曲
问题1.1 最长平台(PLATEAU.C)
问题1.2 支配值数目(GT_COUNT.C)
问题1.3 等值数目(EQ_COUNT.C)
问题1.4 两数组最短距离(MINDIST.C)
问题1.5 等值首尾和(HEADTAIL.C)
第11章 数字问题
问题2.1 Armstrong数(ARMS1.C,ARMS2.C)
问题2.2 数字谜(TRENTE.C)
问题2.3 求质数(PRIME1.C)
问题2.4 筛法(SIEVE.C)
问题2.5 线性筛法(L_SIEVE.C)
问题2.6 因子分解(FACTOR.C)
问题2.7 数值自乘递归解(R_POWER.C)
问题2.8 数值自乘非递归解(I_POWER.C)
问题2.9 Fibonacci数非递归解(FIB_IT.C)
问题2.10 快速Fibonacci数算法(FIB_MT.C)
问题2.11 扩充Fibonacci数(EX_FIB.C)
问题2.12 二项式系数加法解(CNR_ADD.C)
问题2.13 快速二项式系数算法(CNR_LOG.C)
问题2.14 快速阶乘运算(FACTLOG2.C)
问题2.15 更快的阶乘算法(FACTLOG.C)
问题2.16 连续整数的固定和(GIVENSUM.C)
第3章 排列、组合与集合
问题3.1 列出所有子集(DIRECT.C)
问题3.2 列出所有子集——字典顺序(LEXICAL.C)

Download at http://www.pin5i.com/

问题3.3 产生Gray码(GRAYCODE.C)
问题3.4 产生所有排列——旋转法(PERMUT_R.C)
问题3.5 产生所有排列——字典顺序(PERMU_LR.C)
问题3.6 所有K个元素的子集(KSUBSET.C)
问题3.7 集合的所有分割方式(SETPART.C)
问题3.8 整数的所有不同分割数目(INTPART#.C)
问题3.9 整数的分割方式(INTPART.C)
第4章 查找
问题4.1 寻找脚码(ISEARCH.C)
问题4.2 寻找固定的和(FIXSUM.C)
问题4.3 无限式查找(INF_SRCH.C)
问题4.4 寻找极小值(CYCLEMIN.C)
问题4.5 两个数组的中位数(MEDIAN2.C)
问题4.6 寻找中间值(B_SEARCH.C)
问题4.7 3个数组的共同元素(SEARCH3.C)
问题4.8 寻找最小与次小元素(1ST&2ND.C)
问题4.9 查找矩阵(M_SEARCH.C)
问题4.10 表示成两个数平方和(TWOSQUAR.C)
问题4.11 最大方块区域(MAXSQR.C,MAXSQR2.C)
第5章 排序
问题5.1 二分插入法(BINSERT.C)
问题5.2 Shell法(SHELL.C)
问题5.3 快速排列法(QSORT.C)
问题5.4 保持等值的原来顺序(QSORT_L.C)
问题5.5 非递归、无堆栈快速排列法(QSORT_I.C)
问题5.6 求中位数(MEDIAN1.C)
第12章 排列、组合与集合
问题3.1 列出所有子集(DIRECT.C)
问题3.2 列出所有子集——字典顺序(LEXICAL.C)
问题3.3 产生Gray码(GRAYCODE.C)
问题3.4 产生所有排列——旋转法(PERMUT_R.C)
问题3.5 产生所有排列——字典顺序(PERMU_LR.C)
问题3.6 所有K个元素的子集(KSUBSET.C)
问题3.7 集合的所有分割方式(SETPARTC)
问题3.8 整数的所有不同分割数目(INTPART#.C)
问题3.9 整数的分割方式(INTPART.C)
第13章 查找
问题4.1 寻找脚码(ISEARCH.C)
问题4.2 寻找固定的和(FIXSUM.C)
问题4.3 无限式查找(INF_SRCH.C)
问题4.4 寻找极小值(CYCLEMIN.C)
问题4.5 两个数组的中位数(MEDIAN2.C)
问题4.6 寻找中间值(B_SEARCH.C)
问题4.7 3个数组的共同元素(SEARCH3.C)
问题4.8 寻找最小与次小元素(IST&2ND.C)
问题4.9 查找矩阵(M_SEARCH.C)
问题4.10 表示成两个数平方和(TWOSQUAR.C)
问题4.11 最大方块区域(MAXSQR.C,MAXSQR2.C)
第14章 排序
问题5.1 二分插入法(BINSERT.C)
问题5.2 Shell法(SHELL.C)
问题5.3 快速排列法(QSORT.C)
问题5.4 保持等值的原来顺序(QSORT_L.C)
问题5.5 非递归、无堆栈快速排列法(QSORT_I.C)
问题5.6 求中位数(MEDIAN1.C)
问题5.7 堆积法(HEAPSORT.C)
问题5.8 改良的堆积法(HEAP_NEW.C)
问题5.9 合并法(M_SORT.C)

Download at http://www.pin5i.com/

问题5.10 桶子法(BUCKET.C)
问题5.11 单一重复元素排序(LOT_DUP.C)
问题5.12 均匀重复元素排序(LOG_DUP.C)
问题5.13 堆积式合并(HEAPMERG.C)
问题5.14 检查数组元素是否相异(UNIQUE.C)
问题5.15 数组中和为零的段落(ZEROSUM.C)
问题5.16 平面上的极大点(MAXSET.C)
问题5.17 宴会中访问数目的极大值(MAXVISIT.C)
问题5.18 包含在其他区间中的区间(CONTAIN.C)
第6章 字符串
问题6.1 括号匹配问题(PARCOUNT.C)
问题6.2 转换成后继式写法(POLISH.C)
问题6.3 计算前置式写法(PREFIX.C)
问题6.4 Knuth-Morris-Pratt法寻找字符串(KMP.C)
问题6.5 Boyer-Moore法寻找字符串(BM.C)
问题6.6 所谓的h-序列(RH_SEQ.C,H_SEQ.C)
问题6.7 寻找部分序列(SUBSEQ.C)
问题6.8 最长重复部分序列(MAX_REPS.C)
问题6.9 最长共同部分序列(LCS.C)
问题6.10 字符串编修(STREDIT.C)
问题6.11 产生无连续重复部分的字符串(DISTSEQ.C)
第7章 其他问题
问题7.1 Buffon丢针问题(BUFFON.C)
问题7.2 三色旗问题(FLAG.C)
问题7.3 字符串列整数的转换(X_ATOI.C)
问题7.4 整数类型列的极值(LIMITS.C)
问题5.7 堆积法(HEAPSORT.C)
问题5.8 改良的堆积法(HEAP_NEW.C)
问题5.9 合并法(M_SORT.C)
问题5.10 桶子法(BUCKET.C)
问题5.11 单一重复元素排序(LOT_DUP.C)
问题5.12 均匀重复元素排序(LOG_DUP.C)
问题5.13 堆积式合并(HEAPMERG.C)
问题5.14 检查数组元素是否相异(UNIQUE.C)
问题5.15 数组中和为零的段落(ZEROSUM.C)
问题5.16 平面上的极大点(MAXSET.C)
问题5.17 宴会中访问数目的极大值(MAXVISIT.C)
问题5.18 包含在其他区间中的区间(CONTAIN.C)
第15章 字符串
问题6.1 括号匹配问题(PARCOUNT.C)
问题6.2 转换成后继式写法(POLISH.C)
问题6.3 计算前置式写法(PREFIX.C)
问题6.4 Knuth-Morris-Pratt法寻找字符串(KMP.C)
问题6.5 Boyer-Moore法寻找字符串(BM.C)
问题6.6 所谓的h-序列(RH_SEQ.C,H_SEQ.C)
问题6.7 寻找部分序列(SUBSEQ.C)
问题6.8 最长重复部分序列(MAX_REPS.C)
问题6.9 最长共同部分序列(LCS.C)
问题6.10 字符串编修(STREDIT.C)
问题6.11 产生无连续重复部分的字符串(DISTSEQ.C)
第16章 其他问题
问题7.1 Buffon丢针问题(BUFFON.C)
问题7.2 三色旗问题(FLAG.C)
问题7.3 字符串列整数的转换(X_ATOI.C)
问题7.4 整数类型列的极值(LIMITS.C)
问题7.5 无限位数算术(ARITH.C)
问题7.6 线性表示的矩阵相乘(MATMUL.C)
问题7.7 对称表示的矩阵相乘(MATMUL_S.C)

Download at http://www.pin5i.com/

问题7．8　找零钱问题（CHANGE.C）
问题7．9　背包问题（KNAPSACK.C）
问题7．10　最佳矩阵相乘顺序（PRODSEQ.C）
问题7．11　最短路径问题（SHORTEST.C）
问题7．12　产生匹配括号的字符串（PAR_GEN.C）
问题7．13　稳定伴侣问题（STABLE.C）
问题7．14　单调矩阵的极值（MONO_MAX.C）
问题7．15　向量分类（CLASSIFY.C）
第8章　游戏问题
问题8．1　奇数阶魔方阵（MAGIC_O.C）
问题8．2　单偶数阶魔方阵（MAGIC_SE.C）
问题8．3　双偶数阶魔方阵（MAGIC_DE.C）
问题8．4　N后问题公式解（N_QUEEN1.C）
问题8．5　N后问题递归解（N_QUEENR.C）
问题8．6　武士巡逻（KNIGHT.C）
问题8．7　环游世界（HAMILTON.C）
问题8．8　一笔画（EULER.C）
问题8．9　非递归河内之塔（HANOI_I.C）
问题8．10　生命游戏（LIFE.C）
第9章　终曲
问题9．1　等量正负号段落（BALANCE.C）
问题9．2　寻找长方形（RECT.C）
问题9．3　多边形的直径（DIAMETER.C）
问题9．4　机器人旋转角度（TURNS.C）
问题9．5　最大涵盖距离（MAXCOVER.C）
问题9．6　最大连续元素和（MAXSUM.C，MAXSUM1.C）
问题9．7　最大连续元素积（MAXPROD.C）
问题7．5　无限位数算术（ARITH.C）
问题7．6　线性表示的矩阵相乘（MATMUL.C）
问题7．7　对称表示的矩阵相乘（MATMUL_S.C）
问题7．8　找零钱问题（CHANGE.C）
问题7．9　背包问题（KNAPSACK.C）
问题7．10　最佳矩阵相乘顺序（PRODSEQ.C）
问题7．11　最短路径问题（SHORTEST.C）
问题7．12　产生匹配括号的字符串（PAR_GEN.C）
问题7．13　稳定伴侣问题（STABLE.C）
问题7．14　单调矩阵的极值（MONO_MAX.C）
问题7．15　向量分类（CLASSIFY.C）
第17章　游戏问题
问题8．1　奇数阶魔方阵（MAGIC_O.C）
问题8．2　单偶数阶魔方阵（MAGIC.SE.C）
问题8．3　双偶数阶魔方阵（MAGIC　DE.C）
问题8．4　N后问题公式解（N_QUEEN1.C）
问题8．5　N后问题递归解（N_QUEENR.C）
问题8．6　武士巡逻（KNIGHT.C）
问题8．7　环游世界（HAMILTON.C）
问题8．8　一笔画（EULER.C）
问题8．9　非递归河内之塔（HANOI_I.C）
问题8．10　生命游戏（LIFE.C）
第18章　终曲
问题9．1　等量正负号段落（BALANCE.C）
问题9．2　寻找长方形（RECT.C）
问题9．3　多边形的直径（DIAMETER.C）
问题9．4　机器人旋转角度（TURNS.C）
问题9．5　最大涵盖距离（MAXCOVER.C）
问题9．6　最大连续元素和（MAXSUM.C，MAXSUM1.C）
问题9．7　最大连续元素积（MAXPROD.C）
问题9．8　寻找名人（SINK.C）

Download at http://www.pin5i.com/

问题9．9　投票问题（VOTING.C）
问题9．10　寻找1 对1 函数（1TO1.C）
问题9．11　寻找支配元素（DOMINATR.C）
问题9．12　最长递增部分序列（LIS.C）
问题9．8　寻找名人（SINK.C）
问题9．9　投票问题（VOTING.C）
问题9．10　寻找1 对1 函数（1TO1.C）
问题9．11　寻找支配元素（DOMINATR.C）
问题9．12　最长递增部分序列（LIS.C）
参考文献