普通高中教科书

信息技术

选择性必修1

数据与数据结构

浙江教育出版社
ZHEJIANG EDUCATION PUBLISHING HOUSE

普通高中教科书

信息技术

选择性必修1

数据与数据结构

闫寒冰 主编

浙江教育出版社·杭州

主　　编：闫寒冰
副 主 编：赵　健　魏雄鹰

本册主编：邵红祥
编写人员（按姓氏笔画排列）：
吴建锋　张　木　陈军辉　邵红祥　游光辉

前　言

信息技术作为当今先进生产力的代表，已经成为我国经济发展的重要支柱和建设网络强国的战略支撑。在这样的大背景下，教育部全面修订并颁布了《普通高中信息技术课程标准（2017年版）》，为这门课程设定了与新时代相符的育人目标：帮助学生掌握信息技术基础知识与技能、增强信息意识、发展计算思维、提高数字化学习与创新能力、树立正确的信息社会价值观。

本套教材依据《普通高中信息技术课程标准（2017年版）》编写，包括两本必修教材《数据与计算》《信息系统与社会》，六本选择性必修教材《数据与数据结构》《网络基础》《数据管理与分析》《人工智能初步》《三维设计与创意》《开源硬件项目设计》，两本选修教材《算法初步》《移动应用设计》。

本套教材的编写组汇集了来自信息技术、课程与教学、教育技术等领域的高校学者与教学一线专家。编者们通力合作，从课程内容、教材体例、技术选择、教学方法、学习方法等方面精心打磨，期待以最专业的样态帮助学生达到课程预期的育人目标。

具体而言，本套教材体现了如下特点：

1. 体例上——为核心素养的培养创造空间和条件：将核心学习内容与支持学习的方法有机融合在一起，支持学生在自主、合作、探究的学习情境下发展核心素养。

2. 内容上——体现概念、内容与方法的精准与专业：在增强教材可读性的同时，精炼提升综合素养所必需的核心内容，强调所有概念、内容与方法的精准与专业。

3. 活动上——着力提升学生的高级思维能力：精心设计与布局教材中的练习、思考、讨论、实践与项目学习，追求对高级思维能力的培养。

4. 案例上——体现信息科技的多层需求与多维格局：把案例的呈现作为开阔视野的重要手段，帮助学生理解信息技术对于社会发展所具有的价值与意义。

5. 技术上——引领学生拓宽视野与发展思维：将每种具体应用软件都作为解决某些问题的一条路径来看待，期待学生通过具体的技术操作体验，理解其背后的原理与格局、特点与局限，拓宽视野、发展思维。

本册教材为选择性必修《数据与数据结构》。通过本教材的学习，期待同学们能进一步认识到数据在信息社会中的重要作用，在掌握常用数据结构的概念、特点、操作、编程实现方法等内容的基础上，能对简单的数据问题进行分析，选择恰当的数据结构，并能编程实现，在问题解决过程中对数据抽象、数据结构的思想与方法有初步的认识。

就教材本身讲述的知识内容而言，我们相信，只要同学们潜心自学就可以基本掌握。但“知识内容”只是发展信息技术核心素养的基础部分，所以，我们希望同学们不要仅满足于对具体知识与具体技术的掌握，还要重视教材中的各类学习活动，与老师和学友一起，更多地去创造、研究、解决问题、制作、交流、合作和评价，唯有如此，同学们才能藉由这门课程的学习全面地提升信息素养，增强在信息社会的适应力与创造力，为实现中华民族伟大复兴的宏伟目标做出更大贡献！

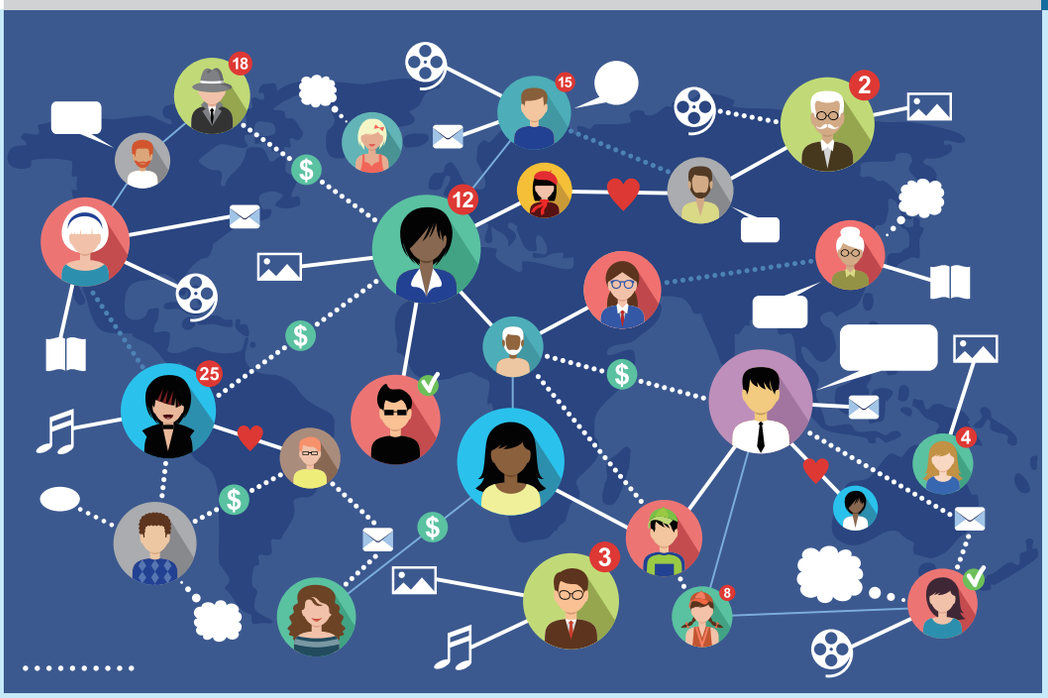
本册教材在编写过程中得到了各方面的大力支持。北京大学计算机系李晓明教授、浙江大学计算机学院卜佳俊教授和翁恺教授、北京航空航天大学欧阳元新副教授在百忙之中对书稿内容进行了审阅。胡金锦、赵军、周海君三位高中教师对书稿内容提出了宝贵的修改意见。

由于水平有限，本书可能还存在不足之处。希望大家在教材使用过程中，能够及时将意见和建议反馈给我们，对此，我们深表谢意。

目 录
MULU

第一章 1

数据与数据的组织



在数字化时代，数据给日常生活、企业经营、社会管理等方面带来了越来越深刻的影响。例如，导航软件可以根据路况数据规划合理的行车线路；高空气象数据通过超级计算机的计算，可以使气象预报更为准确；企业可以根据销售数据来完善生产计划。进入大数据时代，人们更是通过计算从数据内部挖掘出有用的信息，从而加快社会发展速度。

用计算机处理数据时，需要根据事物之间的关系确定合理的数据结构，并借助数据结构来组织数据、设计算法。数据结构的设计在一定程度上决定了问题求解算法的效率。

问题与挑战

- 为了根据销售数据调整生产计划、完善营销策略，企业需要定期收集各地产品的销售数据进行统计和分析。采用怎样的方式进行数据的采集和组织，才能高效地实现对所有数据的汇总和初步的排序？

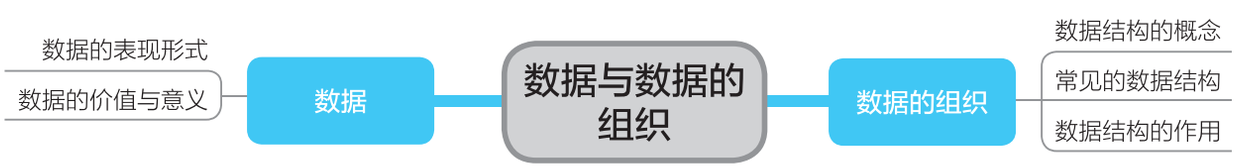
- 一个进行信息分享和交流的网络平台（如BBS论坛等）需要各种角色来协同管理，才能确保平台的正常运行。由于用户角色数量众多且变换频繁（新增用户及角色的改变等），为了实现对这些用户角色的管理，各种用户角色数据该如何表示、组织？用怎样的方式来组织、处理这些数据，效率更高？

- 公共停车场的停车管理系统能自动根据车辆的进入时刻以及离开时刻来计算该车辆的停车费。在整个数据处理过程中，数据是如何采集、存储的？停车管理系统又是如何通过存储的数据来计算停车费的？

学习目标

1. 理解数字、数值和数据的基本含义，了解数据对现代社会发展的作用。
2. 理解数据结构的概念，了解常见数据结构的特点及适用问题。
3. 认识到数据结构在程序设计中的重要作用。

内容总览



1.1 数据

数据是对客观事物的性质、状态以及相互关系等进行描述的物理符号表示或这些符号表示的组合。由于现实世界中事物的多样性和复杂性，在对数据进行加工处理时，就需要对数据进行分类并用合理的方式来组织数据。

1.1.1 数据的表现形式

说到数据，人们通常会想到“数字”“数值”等跟数有关的常识。其实，这些只是数据的几种表现形式。数据的表现形式不仅有数字和数值，还有文字、图形、图像、音频、视频等。

1. 数字

数字指的是由阿拉伯数字“0，1，2，3，4，5，6，7，8，9”或其他含义相同的符号表示。数字本身没有意义，没有量的含义，数字只有在具体的情境中才具有实际的意义。

例如，单独的数字“144”没有任何意义，或者说意义是无法确定的。它可以表示“考试成绩是144分”，也可表示“商品价格是144元”，还可以表示“距离是144千米”。如果将它放置在一块公交车站牌中（如图1.1.1），意义就唯一确定了，那就是“144路公交车”。

图1.1.1 公交车站牌

2. 数值

数值指的是由数字符号组成的、具有量的意义的、可以进行算术运算的数据。显然，图1.1.1所示实例中的“144”只是一个符号，表示从池华街公交站开往翠苑二区方向的公交车编号，不能参与算术运算。而“甲地到乙地距离为144千米”中的144是数值，因为它有量的意义，可以对其进行算术运算。例如，用144除以一辆车的总行驶时间，可以求得这辆车行驶的平均速度；将144加上乙地到丙地的距离，表示从甲地到丙地的距离。

数字化时代中，数据更多地指用计算机进行处理的符号表示的总称。计算机面对孤立的符号144时，无法判别这个符号所处的具体情境，或者说计算机不知道这个144是符号还是数值，这就需要人们事先给予界定并以某种方式告诉计算机。在编程时，给变量和常量设定数据类型就能实现这样的目标。

3. 其他表现形式

早期的计算机由于运算速度、软硬件环境等因素制约，只能处理以数字、文字等形式存在的数据。随着微电子技术及数据处理技术的发展，计算机能处理的数据已经扩展到图形、图像、视频、音频等形式，并能在快速处理这些数据的基础上做出即时的互动反应（如图1.1.2所示的虚拟现实技术能捕捉人的动作数据并做出相应的反馈），甚至能做出一些人类特有的智能化行为。如无人驾驶汽车（如图1.1.3）利用外部摄像机、声音捕捉器、雷达等设备采集到数据，依靠计算机进行快速的数据处理，然后做出实时的驾驶控制反应。

图1.1.2　虚拟现实技术

图1.1.3　无人驾驶汽车

问题与讨论

为什么现实中我们很少会关注“数字”和“数值”的差别，但在计算机科学领域，却必须严格地区分并且在编程时还要定义各种数据类型？

1.1.2　数据的价值与意义

人们可以通过对数据的分析研究来深入认识客观世界，基于数据内容、数据所处情境以及事物发展结果间的规律，对数据进行分析、释义，可以做出事物发展的预测，开展规划、预防等有意义的工作。所以，数据是人类与客观世界进行对话的接口，数据对于人类有着极其重要的价值与意义。

1. 数据促进了人类社会的发展

人类一直以来都在尝试对客观世界进行数据表示，从用肢体计数到用石子和贝壳计数，从结绳计数到用筹码计数，直到现代通过计算机进行编码，数据表示帮助人们实现了信息交流和意义建构。

人类在数据表示的基础上，对数据进行进一步处理和分析，数据才真正成为推动生产

力发展的要素，而数据处理与分析的工具也在此过程中不断发展起来。从绳结、算筹、算盘，直到计算器和今天的计算机，这些计算工具的发明都源于对现实中计算的需要。比如在第二次世界大战中，密码破译工作涉及的数据数量和计算量都非常巨大，为了及时破译敌军密码，英国发明了电子计算机巨人（Colossus）。不久之后，同样出于对现实数据处理的需要，美国军方成功研制了ENIAC电子计算机。

电子计算机的出现引领人类社会进入了信息时代，随着计算机系统软硬件的不断升级改良和人类运用计算机进行数据处理能力的提高，计算机越来越被广泛地应用到数据处理事务中，并从数据中挖掘有用的信息用于社会生产。例如，制药厂根据药品的销售库存数据规划下一年各种药品的生产量，以确保既满足患者的需要又不致产生过多的库存。而政府部门则可以根据数据分析结果进行社会事业的规划决策，如根据当前学校学生数量和未来人口增长趋势，教育管理部门判断是否需要增设新学校等。

可以说，人类社会是在数据的表示和分析中不断发展前进的。

2. 大数据推动人类进入一个崭新的时代

大数据不单指数据的海量，而是针对具有“4V”特征的数据。大数据技术是指提高对这些数据的“加工能力”，通过“加工”实现数据“增值”的一系列硬件和软件技术的综合。

运用大数据技术采集医疗、教育、商贸、社会事业建设以及交通等领域的纵向或横向的海量数据，开展对数据的预处理、分析统计和数据挖掘等处理，就能使数据“增值”，帮助人们进行规划、决策，促进社会各领域的发展。

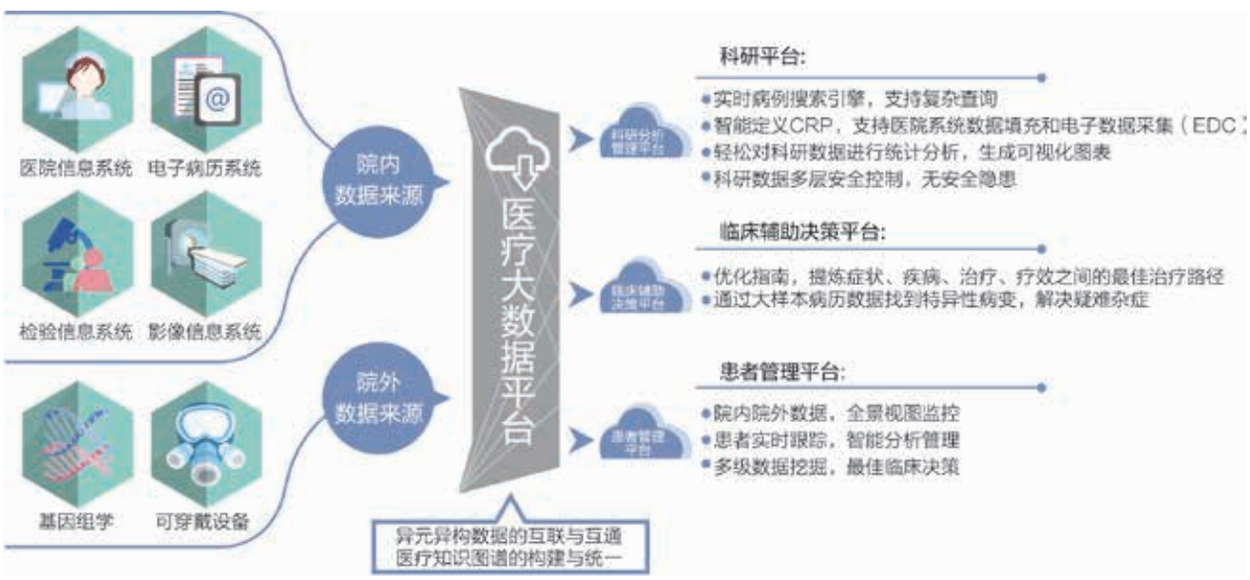


图1.1.4　医疗大数据平台

例如，运用大数据技术，医院可以分析病人特征数据和疗效数据，找到针对特定病人的最佳治疗途径，以减少过度治疗、避免治疗不足（如图1.1.4）。学校采集每位学生学习过程中获取知识的内容偏好、提交作业的错误特征、学习行为的规律等信息，分析出每位

学生知识和技能的缺陷，从而针对性地为每位学生判断缺失的知识和技能环节、推送个性化的学习建议和适合的学习资源，帮助每位学生提升学业水平、优化学习方式。网络运营商可以根据用户浏览网页的痕迹为每个用户记录标签并生成用户画像，在推销时就可根据画像特征进行精准推送。根据手机信号在真实地理空间上的覆盖情况，完整、客观地还原出手机用户的现实活动轨迹，从而挖掘得到人口生活空间分布（如图1.1.5）与工作空间分布（如图1.1.6）等特征信息，为城市规划提供可靠的参考信息。

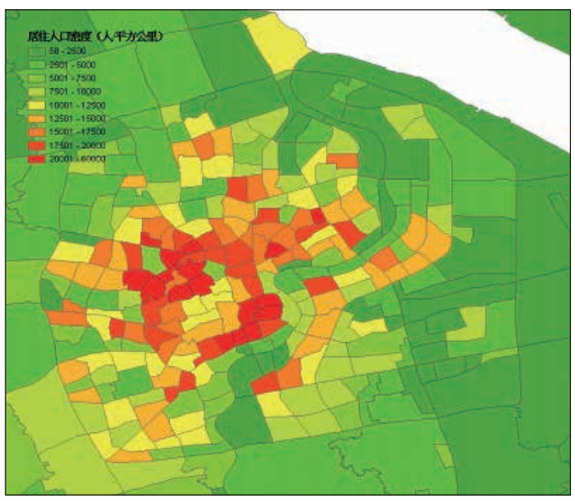


图1.1.5　城市人口生活空间分布图

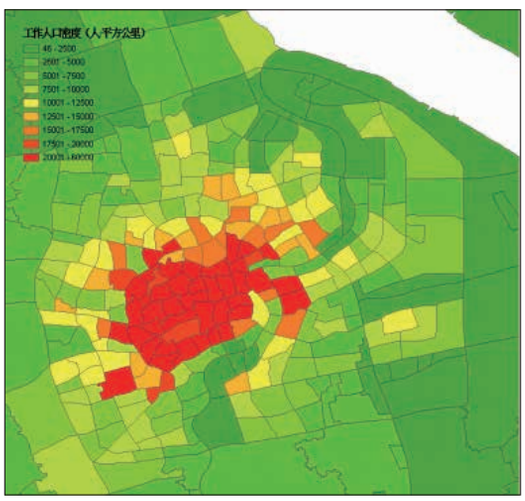


图1.1.6　城市人口工作空间分布图

美国麦肯锡全球研究院在2011年6月发布的研究报告《大数据：下一个创新、竞争和生产力的前沿》中指出“大数据时代已经到来”，数据正成为与物质资料和人力资源相提并论的重要生产要素。

我国政府和各大公司也已开展了大数据技术的布局、研发与应用。2015年9月，国务院印发了《促进大数据发展行动纲要》，从“加快数据共享”“发展大数据应用”“健全大数据安全保障体系”三个方面提出了大数据发展的主要任务，系统部署大数据发展工作。

收集的数据有时并不直接具备价值和意义，要从收集的数据中得到有价值的信息，必须根据需要建立计算模型，然后在设计算法对数据进行加工与分析前，还需要对数据进行有效的组织，使数据变得“干净”而又有“组织”。

实践与体验

剩余行驶时间预估

导航软件已成为现代人们生活中不可或缺的工具，因为导航系统不仅能合理规划路线，还能向用户提供距离目的地多少千米以及大约还剩余多少时间等参考信息，如图1.1.7所示。

实践内容：

早期的导航系统只能根据剩余里程数和汽车当前行驶的速度来估算剩余所需的行驶时间，准确性较差。现在的导航系统能综合路况数据，结合海量的用户数据，在每个时刻为用户提供较为准确的剩余时间预估。

请收集相关的信息，分析当前导航系统预估剩余时间的基本原理和方法。

图1.1.7　导航软件界面

实践步骤：

1. 选择一到两款导航软件，分别进行步行和驾车导航体验，感受导航仪提供剩余所需时间的准确性。

2. 通过各种渠道（上网搜索、查找图书资料、请教相关行业人员等），了解导航系统预估剩余时间的原理和方法。

3. 基于上述体验和学习，分析导航软件的现状并给出完善的建议。

结果呈现：

1. 请简要阐述你所体验和了解的导航仪预估剩余时间的原理和方法。

2. 你觉得导航系统在预估剩余时间时还存在哪些问题？可以如何改进？

思考与练习

从现实中寻找一个计算机处理数据的实例，找出其中的“数字”“数值”“数据”，进一步地分析“数字”“数值”“数据”之间的关系。

1.2 数据的组织

对数据进行加工与分析前，需要对数据进行预处理并将数据组织成一定的形式，方便计算机程序对数据进行加工。在计算机程序设计中，根据问题求解的需要，对数据进行有效的整理和组织，并以一定的形式加以存储和表示的过程称为数据结构的设计。瑞士计算机科学家沃斯（N.Wirth）指出“算法+数据结构=程序”，形象地阐明了计算机程序设计的本质是算法与数据结构的设计。

1.2.1 数据结构的概念

现实问题中的数据往往具有多样性和复杂性的特点，为了有效组织数据，必须对各种数据加以分类，对互相有关联的数据进行合理重组，在此基础上，才能较好地选择、设计数据结构。

1. 数据元素（Data Element）

数据元素是数据的基本单位。有些情况下，数据元素也称为元素、节点、顶点、记录等。有时一个数据元素可以由若干个数据项（也称为字段、域）组成，数据项是具有独立含义的最小数据表示单位。

某天A股物联网部分股价表现一览

代码	名称	最新价格（元/股）	动态市盈	流通股本（万股）
300155	安 居 宝	15.62	48.55	5775.62
000988	华工科技	7.78	35.61	89111.65
002104	恒宝股份	9.92	40.83	34032.05
300168	万达信息	18.4	88.06	17006.35
000997	新 大 陆	9.70	52.81	50870.62
300078	中瑞思创	11.83	26.47	6269.87
002121	科陆电子	7.68	26.96	26762.09
600718	东软集团	9.09	30.90	122227.59
002161	远 望 谷	7.96	41.15	52082.60
002073	软控股份	9.80	38.66	61659.73

图1.2.1　数据元素及其包含的数据项

图1.2.1所示二维表中，每一行实际内容（也称为一条记录）就是数据元素，而每个元素又由5个数据项（“代码”“名称”“最新价格”“动态市盈”“流通股本”）组成。例如，

第一个数据元素的第一个数据项名称为“代码”，该数据项的值为“300155”；第一个数据元素的第二个数据项名称为“名称”，该数据项的值为“安居宝”。

2. 数据类型（Data Type）

数据类型指的是具有相同性质的计算机数据的集合及在这个数据集合上的一组操作。例如，Python中整数数据类型包含了两个方面的关键内容，一是由计算机系统所能表示的整数组成，二是指在这个整数集合上可以进行的加、减、乘、除、取余数等算术运算操作。再如字符串数据类型，既包含了一切能被计算机表示的符号，又包含了在这个字符集合上能进行的查找、切片、联接等操作。

数据类型可以分为基本数据类型（也称为原子数据类型）和结构数据类型。基本数据类型由计算机编程环境提供，编程者可以在编程时直接用系统提供的标识符进行定义，如Python编程语言中的整型、实型、布尔型等。结构数据类型是在程序设计时利用基本数据类型构造出的、复合的新类型，这种新类型由用户根据实际需要定义，能较好地描述数据元素的数据项组成以及数据元素之间的逻辑关系，方便用户根据数据之间逻辑关系的特点进行数据处理，如很多编程语言中提供的记录类型、集合等。

3. 数据结构（Data Structure）

数据结构指的是数据之间的相互关系，即数据的组织形式。它包括了以下三个方面的内容：

①数据元素之间的逻辑关系，也称为数据的逻辑结构。

②数据元素及其关系在计算机存储器内的表示，也称为数据的存储结构或物理结构。

③数据的运算，即对数据施加的操作。

在使用高级程序设计语言进行程序设计时，不需要程序设计者过多关注底层的数据存储，所以大部分情况下数据结构总是指向数据的逻辑结构，这也体现了实际应用中数据结构设计主要考虑数据对象之间逻辑关系的现实。在本书中，除非做特殊说明，数据结构一般指向的就是逻辑结构。

数据结构设计的目的是使数据元素间的相互关系能准确地反映现实问题中的事物逻辑，既确保数据处理的正确性，又提高编程实现和数据处理的效率。例如，对一批身高数据进行升序排序操作，在现实中会将对象按照身高依序排成一列，而在编程时就采用一种既能表示各位置身高数据，又能体现位置之间线性关系的数组来组织数据。数组就是一种数据结构。

1.2.2 常见的数据结构

数据结构设计是为了解决实际问题而出现的科学，因此，科学家针对现实问题中实体间的关系特征，设计了各种数据结构，使得开发人员能根据问题特征，选择合适的数据

结构来组织与存储数据，以达到高效处理数据的目的。常见的数据结构有数组、链表、队列、栈、树和图等。

1. 数组

现实中表示一批数据，有时不仅需要描述数据对象本身，还需要描述数据所处的位置或者数据之间的前后顺序关系，对于这类数据集合的组织，可以用数组这种数据结构来实现。

如图1.2.2所示，当一队人依次排队时，为了描述他们之间的前后关系，可以用“第1个是李彤”“第2个是张强”“第3个是胡洁”“第4个是杜刚”这种方式表达。

如果将上述表达进一步抽象，那么这批数据序列可用“a[1]="李彤"、a[2]="张强"、a[3]="胡洁"、a[4]="杜刚"”来表达。在这种表达方式中，a[1]、a[2]、a[3]、a[4]就组成了数组a。其中，变量名a说明这是一批类型相同的数据，下标1、2、3、4表示了数据元素所处的位置顺序。例如，“a[3]="胡洁"”说明“胡洁”处于第3个位置上，她的前面是第2个位置上的“张强”，后面是第4个位置上的“杜刚”。

图1.2.2　成一列纵队排队的人

用数组来组织数据时，既可以快速地通过下标精确地访问序列中的某个数据元素，如a[99]就表示第100个数据元素的值（第1个元素用a[0]表示），又可以方便地通过变量名后面的下标依次按顺序遍历序列中的每个数据元素。用计算机解决问题时，很多时候都可以用数组来组织、存储数据，如记录并统计全班同学的生日、杭州到全国各省会城市的机票价格等。

拓展链接

遍　历

遍历指的是根据数据之间的逻辑结构，遵循一定的顺序依次对所有数据元素做一次且仅做一次访问。访问某个数据元素时，需要进行的操作依赖于具体实际问题。例如，当需要计算满足条件的元素之和时，访问每个元素时首先判断元素是否符合条件，若满足则将其累加。

2. 链表

同样是对一批人员数据进行组织，有时只需知道相邻人员之间的前后顺序关系，而对每个人员的位置信息并不作要求。例如，学生军训时，因为每次集合和整队地点可能不同，所以不必记住每位同学具体的位置号，而只需知道排队时自己前面的那位同学，那么接下来无论在哪里集合，所有同学都可迅速地按原顺序排成一队。如图1.2.3所示，吴坚知道自己排在首位，王林知道排在自己前面的是吴坚，黄刚知道排在自己前面的是王林，李丰知道排在自己前面的是黄刚。有了这些相邻人员之间的链接关系，即使休息时大家分散在各处，一旦需要集合，大家可以根据链接关系快速地按照原顺序排成队伍（如图1.2.4所示）。虽然整队前后每个人员的站位地点发生改变，但相互之间排队的顺序关系是不变的。

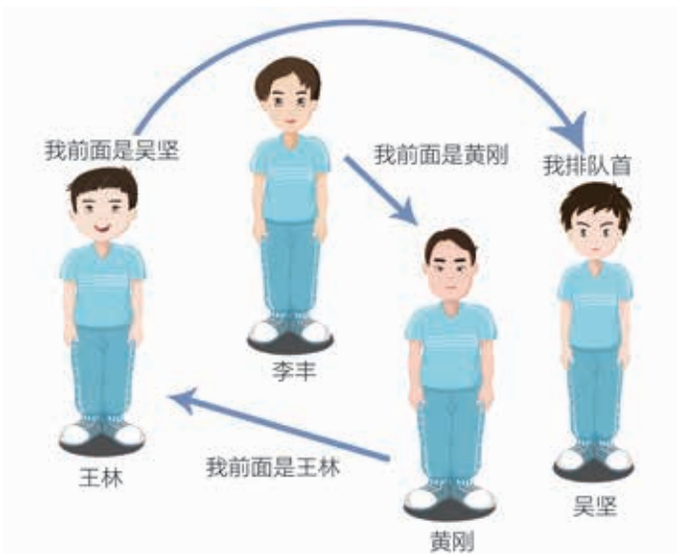


图1.2.3　整队前的位置和链接关系

图1.2.4　整队后的位置和链接关系

若用箭头表示“我前面是谁”，则图1.2.4所示的链接关系可以用图1.2.5所示的抽象化形式表示。

图1.2.5　抽象化后的排队链接关系

同样地，在组织、处理一批数据时，若不关心数据实际所处的具体位置，而只需知道数据之间相互链接的顺序时，可以借鉴上面的方法。在计算机科学中，这种方法的具体实

现形式就是链表。根据解决问题的需要，链表可以有多种存在形式，常用的有单向链表、双向链表、循环链表。具体操作时，为了记录、识别各个链表，需要在链表首部设立一个头节点，同时用一个变量head记住这个头节点，这样就可以从head指向的头节点开始（head在这里也可称为指针）依次逐个遍历链表中的每个节点，链表中的指针可以指向前面，也可以指向后面。若指针定义为指向排在后面的人员，则图1.2.5所示的排队链接关系可用图1.2.6所示的单向链表来表示。该链表中，李丰所在节点中的“^”意为空指针，表示后续没有其他节点。

图1.2.6　单向链表

拓展链接

指　针

在计算机科学中，指针（Pointer）是用来指示一个数据存储地址（内存或者寄存器）的变量。如图1.2.6所示的head，该变量存储的是头节点所在的内存地址，或者说存储的地址指向了头节点，所以形象地称为“指针”。

有时，为了满足链表能在两个方向都能进行遍历的需要，会在单向链表（也称为单链表）基础上给每个节点增加一个指向前趋节点的链接，这样就形成了双向链表（也称为双链表），如图1.2.7所示。当求解问题需要循环遍历链表时，则可在单链表基础上，在链表的首尾之间增加链接，就形成了循环链表（如图1.2.8）。

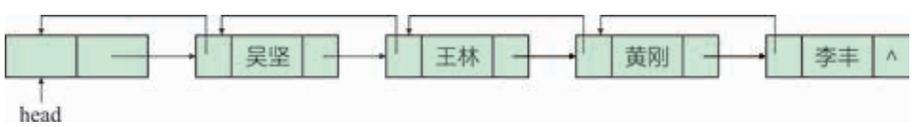


图1.2.7　双向链表

图1.2.8　基于单向链表的循环链表

问题与讨论

火车站或者飞机场为了方便旅客了解最新出乘信息，会在电子屏上滚动显示最近的班次信息（如图1.2.9所示）。该系统内部程序控制滚动显示可以采用怎样的数据结构？主要算法又是怎样的？

图1.2.9　火车站信息显示屏

3. 队列

排队是现实生活中一种常见的现象，在一些人数众多的场合，为了组织人们有序进出，相关部门会竖起栅栏，让人群有序排队形成一列纵队，并按照先进先出的原则逐一安全地进入相应的区域，如景区入口、出租车上车点等。

图1.2.10　有序排队上车的乘客

图1.2.11　有序排队接客的出租车

如图1.2.10所示，乘客排队时先到的总是从队伍的头部出去（出队）上车，而后到的乘客则必须在队伍的尾部加入（入队）。同时，为了确保有序，人们总是规定不能从队伍的中间部位插队。

用计算机程序处理数据时，有时也需要将数据进行“排队”，并遵循现实中排队的规律，对数据进行“先进先出”且中间不能“插队”的组织和操作，计算机科学家由此发明了“队列”这种数据结构。

例如，为了使人们在银行有序地办理各种金融业务，早期的银行会要求人们自觉排队并依次按序办理业务。随着信息技术的发展，银行开发了业务办理叫号排队系统，客户去银行办理业务时，只需先从取号机上取一张排队号，然后就可以一边休息一边等待叫号系统提醒其去哪个窗口办理业务。这个取号、叫号程序内部就可采用“队列”这种数据结构。

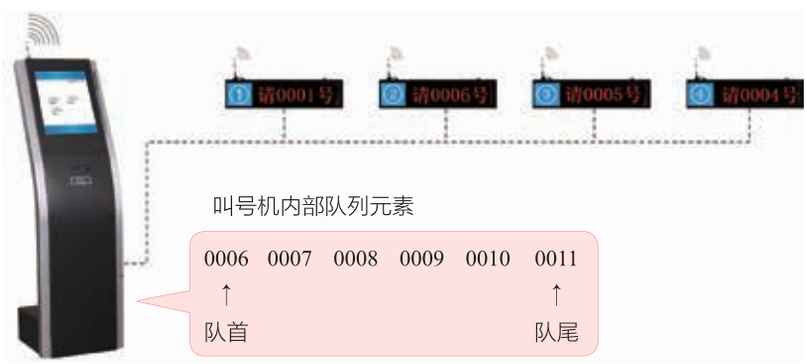


图1.2.12　银行叫号系统

如图1.2.12所示，某时刻银行排队叫号的状态显示：现场有6位客户已取号并处于等待状态，所取号码为0006到0011。如果接下来的时刻窗口2的客户办理结束，接着又有一位新客户在取号机上取了排队号（号码为0012），则系统会让队首元素（0006号）出队并将号码传输到2号窗口的显示屏，同时将新客户刚取走的号码作为数据元素进行入队操作，操作后的结果如图1.2.13所示。

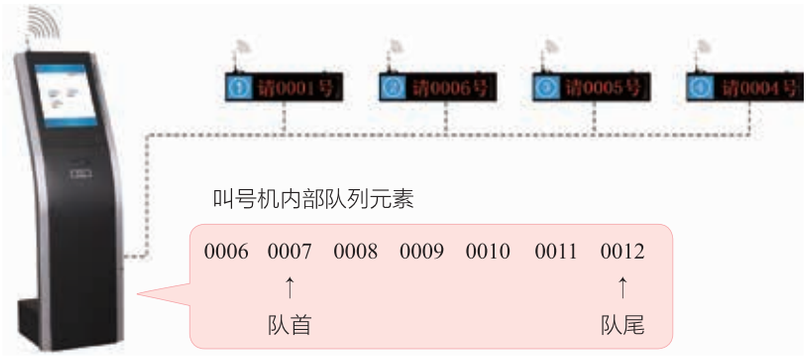


图1.2.13　队列经过一次出队和入队操作后的状态

现实中的很多信息系统内部都采用了队列这种数据结构来组织数据、解决问题。如在计算机硬盘中，为了解决CPU和主存储器（RAM）数据存取之间的速度差问题，生产厂家会在CPU中安装速度远快于RAM的高速缓存（Cache），对高速缓存进行数据存取操作时，就会用到基于队列的FIFO（First In First Out）。再如，有时网络平台瞬间收到很多服务请求，系统会按照服务请求到达的先后顺序，用队列来依次保存各个请求，然后按照入队顺序逐个出队并处理。

4. 栈

“栈”这个字虽然在现实生活中较少见到，但在有些场合的事务处理中还是蕴含了栈的结构和思想，这些事务涉及的装置往往具有一个共同的特点，那就是只有一端是开放的，所有的操作都只能在这开放的一端进行。如消毒桶中餐盘的取放（如图1.2.14）、弹匣中子弹的装弹和出弹等。

图1.2.14　消毒桶中的餐盘

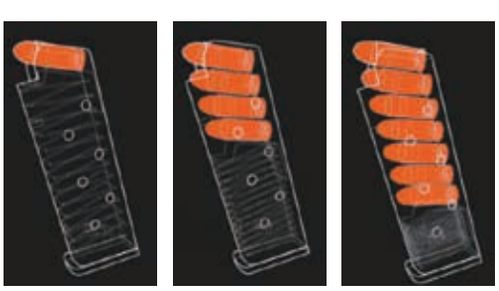


图1.2.15　弹匣的装弹过程（入栈）

弹匣是一种典型的具有栈结构和操作特征的装置（如图1.2.15），即只能在连接枪体的一端进行子弹压装、弹出操作，所以弹匣是一个“栈”。在这个装置中，子弹被装入弹匣称为“入栈”，子弹被弹出弹匣称为“出栈”。

子弹进出弹匣的过程具有下列特点：

①整个装置只有一端开放（最上端），而且进、出只能在这一端进行。

②弹匣中的子弹成一纵队排列。

③任何子弹进出弹匣的规律是“先进后出、后进先出”，即最先装入弹匣的子弹最后才能被弹出，而最后装入弹匣的子弹则最先被弹出。

在数字化任务处理中，很多数据之间的结构以及数据操作规律符合上述特点，计算机科学家就发明了“栈”这种数据结构，用来支持此类数据的处理。如网页浏览器对用户浏览网页的管理，内部就采用了栈进行网页数据的组织。当用户由一个网页跳转到另一个网页浏览时，系统会将原先的网页数据进行入栈操作，而当用户单击浏览器的“后退”按钮时，系统又会将栈中最上方的网页数据出栈，用户即可看到刚才最后浏览过的网页内容。

问题与讨论

在文字处理软件Word中输入若干文字，然后删除其中部分文字，再输入若干文字。然后进行“撤消”操作（按Ctrl+Z键，或者单击撤消操作快捷按钮“ ”）。观察各个操作后的现象，回答下列问题。

（1）根据“撤消”操作所产生的结果，说明Word中符号输入及撤消操作中，内部所依托的数据结构是哪种数据结构？为什么？

（2）结合自己的经历，谈谈哪些信息系统中也采用了栈这种数据结构。

5. 树

无论是队列还是栈，其中的数据元素之间都呈现出一种线性关系，即除首尾两端的元素外，其他数据元素的前面和后面都只有一个相邻的元素，所有元素都成“一条线”排列。但现实世界中有些数据之间的关系并非是线性关系，如生物学中的动物分类图（如图1.2.16）、中国图书馆分类法（如图1.2.17）、行政部门组织结构图等。这些数据元素之间的关系都呈现出一个共同的特点，即一个元素前面（或上面）只有一个元素，而后面（或下面）却有多个（0个或多个）元素相邻，所有数据元素之间的关系特征就像一棵倒放的树，所以称之为树结构。

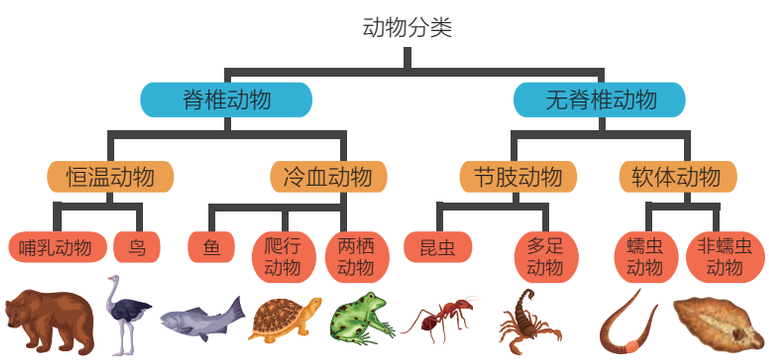


图1.2.16　生物学中的动物分类图

拓展链接

线性表

线性表是由零个或多个数据元素组成的有限序列，数据元素之间的关系是一对一的关系，即除了第一个和最后一个数据元素以外，中间任何一个数据元素的前面和后面都只有一个数据元素与它相邻。线性表是一种基本的、常见的数据结构，可以根据需要向线性表增加元素，或者删除表中的元素。数组、队列、栈、链表都是线性表的特殊形式。

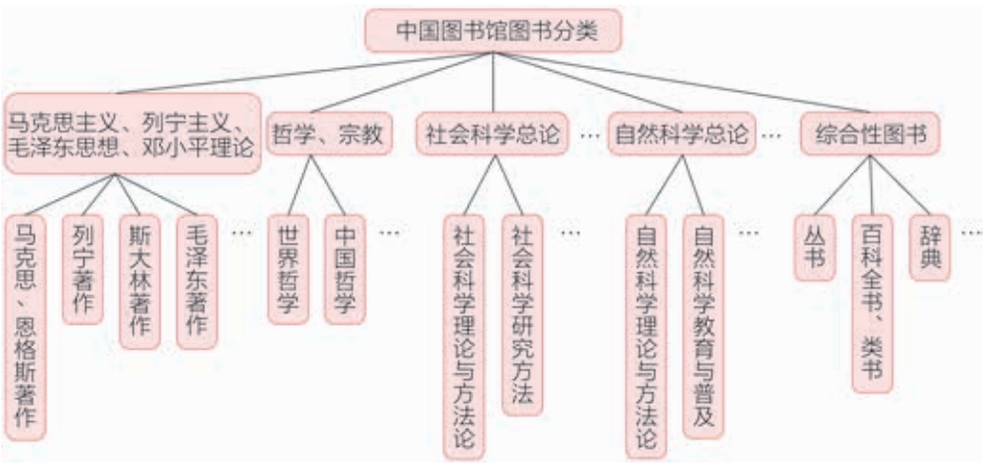


图1.2.17　中国图书馆图书分类法

如图1.2.17所示，中国图书馆图书分类法将图书分为几大基本部类，各个基本部类又可以分出更细化的分类。例如，“马克思主义、列宁主义、毛泽东思想、邓小平理论”基本部类可以细分出“马克思、恩格斯著作”“列宁著作”“毛泽东著作”等。

在计算机科学中，当需要处理的数据元素之间呈现出上述特点时，就可以用树这种数据结构来组织数据，并采用针对树的各种算法来解决问题。例如，计算机操作系统中文件夹的组织与管理、下棋算法中对后续局面的状态表示时，都需要借助树结构来组织、保存和处理数据。

问题与讨论

分小组讨论，举出在生活和信息系统中用树组织数据的例子，并用树结构描述数据之间的关系特征。

1.2.3　数据结构的作用

每个实际问题中的数据之间存在着一定的关系，用计算机程序解决问题时，需要将数据之间的这些关系映射到程序的数据表示中，这样才能有效地用计算机程序解决问题。对于同一个问题的解决，当依据不同的数据结构来设计算法时，算法的处理效率、程序的实现效率也是不同的。

1. 设计算法解决问题离不开数据结构

例如，某学校举行趣味运动会，高一开设了“滚铁圈”“打弹子”“拍纸板”“竹蜻蜓”“跳绳”“踢毽子”6个项目的比赛（如图1.2.18）。比赛结束后需要根据每位选手各个项目的得分来统计每位选手的总分以及各班级的总分。


滚铁圈


打弹子


拍纸板

图1.2.18　趣味运动会项目

数据统计前，需要先收集数据并将数据按照既有的逻辑关系进行结构化组织，可以用一张表格来组织数据并表示数据之间的逻辑关系，如表1.2.1所示。

表1.2.1　各位选手在各个项目上的得分

姓名	班级	滚铁圈	打弹子	拍纸板	竹蜻蜓	跳绳	踢毽子
陈　涛	9	3		2			
杨　琼	3					5	1
金　凯	2		4	5			
吴　敏	1	6	1		3		
朱刚强	6	5		1		7	
…	…	…	…	…	…	…	…

根据本问题中数据之间的线性关系特点，可基于数组来设计算法并解决问题。如果不针对数据之间的关系特点设计数据结构，而是直接用一个个简单变量来存储所有学生姓名和各项得分，在此基础上设计算法及编写程序的工作量将变得非常巨大，用计算机处理这些数据就变得毫无意义。

2. 不同数据结构会导致处理效率的不同

（1）用一维数组组织数据

上述运动会数据中，“姓名”列中是一批相同性质和类型的数据，“班级”列中是一批相同性质和类型的数据……可考虑将每一列用一个一维数组存储。同时，可用一维数组sum来表示每位选手的总得分，如表1.2.2所示。为了统计每个班级的总分，可以用一维数组bjdf存储各班总分，其中的bjdf[j]表示第j班的总分。

表1.2.2　用9个一维数组来组织数据

xm[i]	bj[i]	d1[i]	d2[i]	d3[i]	d4[i]	d5[i]	d6[i]	sum[i]
陈　涛	9	3	0	2	0	0	0	
杨　琼	3	0	0	0	0	5	1	

续表

xm[i]	bj[i]	d1[i]	d2[i]	d3[i]	d4[i]	d5[i]	d6[i]	sum[i]
金　凯	2	0	4	5	0	0	0	
吴　敏	1	6	1	0	3	0	0	
朱刚强	6	5	0	1	0	7	0	
…	…	…	…	…	…	…	…	

基于上述这些一维数组组织、存储数据，统计总分时可逐个访问每位选手xm[i]对应的相关数据，并将该选手各个项目的得分累加到sum[i]中，然后将sum[i]的值累加到bjdf[bj[i]]中。当完成对所有选手的遍历时，每位选手的总得分就保存到数组sum中，而每个班级的总分则保存到数组bjdf中。

（2）用二维数组组织数据

进一步分析表1.2.1中的数据集合，可以发现：不但每行中的数据项之间存在着逻辑关系，而且行与行之间也存在着逻辑关系。因为每个数据项可以用一对行号和列号来唯一地定位，且同一行各列中的数据项都属于同一名选手，所以可用包含行号和列号两个位置信息的二维数组来组织、存储数据。用二维数组组织、存储数据的结果如表1.2.3所示。

表1.2.3　用二维数组组织学生成绩数据

xm[i]	a[i,1]	a[i,2]	a[i,3]	a[i,4]	a[i,5]	a[i,6]	a[i,7]	a[i,8]
陈　涛	9	3	0	2	0	0	0	
杨　琼	3	0	0	0	0	5	1	
金　凯	2	0	4	5	0	0	0	
吴　敏	1	6	1	0	3	0	0	
朱刚强	6	5	0	1	0	7	0	
…	…	…	…	…	…	…	…	

表1.2.3中的a[i,1]到a[i,7]依次表示第i位选手的班级以及他（她）在“滚铁圈”“打弹子”“拍纸板”“竹蜻蜓”“跳绳”“踢毽子”6个项目上的得分，而a[i,8]用来保存第i位选手的总得分。

为了统计每位选手和各班的总得分，基于二维数组的总体算法同样可通过遍历每位选手的相关数据进行，但在用计算机程序设计语言描述算法（编写程序）时，相比于一维数组，二维数组在程序实现效率上要高于前者，特别在每位选手总分统计部分。如果数据项数量增加，那么两者相差会更大。

数据合并

生产厂家总会根据各地产品销量的数据分析来预估市场情况，并为后续调整生产规划、完善营销策略提供依据。

由于数据量巨大，为了充分运用分布式处理的优势，总部会要求各下属地区上报数据

时，按各产品销量进行从大到小的排序。

总部收到数据后的第一件事是将所有数据合并并按照销量进行降序排序（从大到小），为了完成数据合并和整理工作，总部数据分析员小刚需要设计合适的数据结构和算法。

分析：小刚可以用一个二维数组存储所有下属地区的产品销量数据，然后直接运用排序算法（排序算法可见本教材第五章数据排序）进行降序排序。如果利用既有数据已是分块有序的特点，设计新的数据结构和算法，则处理效率可以得到相应的提升。

各个地区的数据合并问题可以简化为2个地区的数据合并问题，当2个地区的数据合并完成后，剩余各地区的数据合并就可以按照同样方式完成。因此，接下来着重分析2个地区的数据合并问题。

第一步　抽象与建模

设第1个地区共有n个产品销量数据，第2个地区共有m个产品销量数据。为了简化描述，突出核心部分的分析，考虑将问题抽象为n个有序整数和m个有序整数的合并问题，具体的问题模型如下：

已知一个降序排列的非负整数数列A：a_1，a_2，a_3，…，a_n，以及一个降序排列的非负整数数列B：b_1，b_2，b_3，…，b_m，将两个数列合并形成一个新的有序数列C，使新数列仍按降序排列，即$c_1 \geqslant c_2 \geqslant c_3 \geqslant \cdots \geqslant c_k \geqslant c_{k+1} \geqslant \cdots \geqslant c_{n+m}$（其中$c_k \in A$或者$c_k \in B$）。请完成解决该问题的数据结构和算法的设计。

第二步　设计、描述算法

为了体会不同数据结构对算法效率的影响，下面主要围绕两种数据结构展开算法的设计。

1. 基于数组的算法设计与描述

用一维数组a依次存储第1个数列的所有整数，用一维数组b依次存储第2个数列的所有整数，用一维数组c存储最终合并的有序数列。

首先将a数组的n个元素按原顺序存储到数组c的前n个位置中。为了描述方便，不妨设n=m=5，此时a、b和c数组的状态如下：

a[1]	a[2]	a[3]	a[4]	a[5]			b[1]	b[2]	b[3]	b[4]	b[5]
19	16	12	8	5			20	15	14	10	4
c[1]	c[2]	c[3]	c[4]	c[5]	c[6]	c[7]	c[8]	c[9]	c[10]		
19	16	12	8	5							

接着将b数组中的元素从第一个开始逐个插入到c数组的有序数列中，同时需要将数组c中插入位置后面的元素进行后移。为了充分利用数列既有的有序性特点，每次查找插入位置时，可以从前一个元素的插入位置开始往后查找。所以，当在数组c中插入一个新元素后，需要记录该元素的插入位置p。另外，为了给向右查找插入位置设置一个监视哨，可以预先将数组c中右边没有实际数据元素的位置放置负数−1。

在将b数组中第1个元素20插入数组c时，从p+1（此时p=0）位置开始向右查找插入

位置，查找的条件是“如果c[p]＞b[1]，那么p向右移动一位，直到某次比较发现c[p]=−1则将b[1]存入c[p]；如果c[p]≤b[1]，那么先将c[p]开始右边元素整体向右移动一位，然后将b[1]插入到c[p]位置”。

整体的算法描述如下：

（1）将数组a中所有元素存储到数组c的前n个位置中；

（2）将数组c右边的m个元素赋值为−1（c[n+1]直到c[n+m]）；

（3）变量p赋值为0，将表示数组c中有效元素总个数的变量tot赋值为n；

（4）将数组b中元素b[i]逐个插入到数组c中（1≤i≤m）：

①p值增加1；

②如果c[p]＞b[i]，那么使p值增加1；

③如果c[p]=−1，那么直接将b[i]存储到c[p]中，同时tot值增加1；

④如果c[p]≤b[i]，那么依次将c[tot]，c[tot−1]，…，c[p]向右移动一个位置，然后将b[i]存储到c[p]中，同时tot值增加1。

进一步分析，可以充分利用有序性特点，在c[p+1]到c[tot]中查找b[i]的插入位置时，也可以采用二分查找提升查找效率（二分查找算法可见本教材第五章二分查找）。

2. 基于链表的算法设计与描述

用链表a存储数列A，且从头节点开始链表中每个元素成降序排列。用同样的方法将数列B存储到链表b中。两个链表的初始状态如图1.2.19所示，其中指针p、q分别指向每个链表的当前节点，指针p1指向链表a当前节点的前驱节点，指针q1指向链表b当前节点的前驱节点。

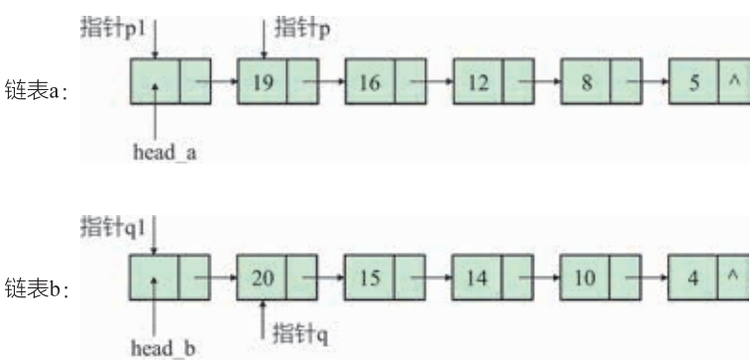


图1.2.19　链表的初始状态

在链表这种数据结构基础上，两个数列的合并处理就变为两个链表的合并问题。当需要将链表b中q指针所指节点插入到链表a时，只需将q所指元素与p所指元素比较，如果p所指元素大于q所指元素，那么使p指针后移到下一个节点（同时指针p1也移到下一个节点），直到p所指元素小于或等于q所指元素。此时，不必像数组中那样移动数列C中的元素，而只需调整指针的指向，即可将q所指元素插入到链表a中p所指元素之前。如图1.2.19所示，当将链表b中第一个元素20插入链表a时，只需将链表a中p1所指节点的后继指针指向q所指元素，链表b中q1所指节点的后继指针指向q所指节点的后继节点（元

素15所在节点），q所指节点的后继指针指向p所指节点，最后将指针q指向q1所指节点的后继节点。

实施上述操作后，链表a和链表b的状态如图1.2.20所示。

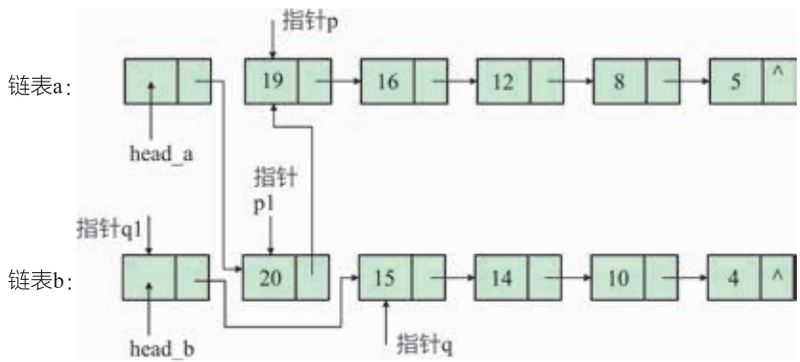


图1.2.20　实施1次插入后的链表状态

从用上述两种数据结构分别来解决同一个问题的过程可知，不同的数据结构会导致算法的不同，而且解决问题的效率也不同。例如，用数组实现时需要将插入位置后面的所有元素逐个右移一位，当数列B中元素全部大于数列A中第一个元素时，将数列B中m个元素全部插入到数列A共需要移动n×m次；而当两个数列的元素大小比较均匀时，如A数列元素为20，18，16，12，6，而B数列元素为21，19，17，15，8时，共需要移动的次数也达到$n+(n-1)+\cdots+2+1=\frac{n(n+1)}{2}$次。而用链表实现时，向链表a中插入一个节点，最多需要5次指针操作，所以最多需要5m次指针操作，要比基于数组的处理低一个数量级。

思考与练习

1. 请结合本课程的学习经历，举例说明其中所涉及的数据元素、数据项、数据类型、数据结构。

2. 银行营业部的叫号系统客户取号界面如图1.2.21所示，该营业部共有6个办理窗口，其中1至4号只办理人民币业务，5号办理外币业务，6号办理理财业务。请运用队列知识，设计该系统取号、叫号处理过程的算法。

图1.2.21　银行叫号系统界面

巩固与提高

1. 请针对前面用链表合并A、B两个数列的案例（图1.2.19和图1.2.20），回答下列问题。

（1）该案例中如果没有头节点及其指针，两个数列的合并操作能完成吗？为什么？

（2）为了完成合并处理，指针p1、q1是必需的吗？如果没有这些指针，分别会出现怎样的结果？

（3）如果将该案例中的单链表改为双链表，为了实现两个数列的合并，你认为数据结构和算法分别又该是怎样的？

（4）你觉得用链表实现数据处理的优点和缺点分别是什么？

2. 如图1.2.22所示，羽毛球装入球筒时，总是球头朝下从“IN”端装入；从球筒中取出羽毛球时，总是从“OUT”端出来。给5个羽毛球分别以1，2，3，4，5编号，这5个羽毛球依次以1，2，3，4，5的顺序进入球筒，那么从“OUT”端取出的羽毛球的顺序是怎样的（用编号表示）？总共有多少种不同的出筒情形？这个操作过程体现了哪种数据结构的操作特征？

图1.2.22　羽毛球的取放

3. 如图1.2.23所示，为了确保盘子叠放得整齐和安全，洗盘子时总是洗完一个就叠放到最上方。而在取用一个盘子时，总是取最上方的盘子。现在有编号为1，2，3，4的4个盘子，按照1，2，3，4的顺序洗干净并依次从下往上叠放在一起，洗盘子期间可能有人（一人或多人）会从叠放的盘子的最上方每次取走一个盘子。问：盘子被取走的编号序列一共有几种结果？请写出所有的编号排列结果。

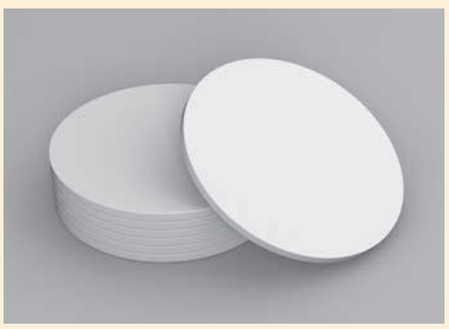


图1.2.23　盘子的取放

4. 画出包含3个节点的所有不同形态的树。

5. 数组、链表、队列、栈都属于线性表的范畴，请从删除既有元素、插入新元素等方面的操作，简要概述基于这些数据结构的操作效率。

项目挑战

网络信息系统的用户角色数据组织

在网络信息系统中，不但用户数量众多，而且每个用户涉及的角色种类也多种多样，这些用户角色有“游客”“注册用户”“栏目1信息发布员”“栏目2信息发布员”“栏目1信息审核员”“栏目2信息审核员”等，有的甚至多达几十个、上百个，组织、管理这些数据的结构和方式，决定了信息系统的高效性和便捷性。

如果用数据库保存这些用户角色数据，可以用表1.2.4的形式，也可用表1.2.5的形式（每一行为一条记录）。

表1.2.4　用户角色数据组织形式1

用户名称	用户角色
Sinor	有限制的管理员
Sinor	栏目1信息发布员
Sinor	栏目1信息审核员
Sinor	栏目1信息管理员
Applesiri	有限制的管理员
Applesiri	栏目2信息发布员
…	…

此种组织方式的问题是当需要删除、修改某个用户的角色数据时，需要涉及多条记录的维护，若涉及的用户和用户角色较多，则数据处理的时间消耗较大。

表1.2.5　用户角色数据组织形式2

用户名称	有限制的管理员	栏目1发布员	栏目1审核员	栏目1管理员	栏目2发布员	栏目2审核员	栏目2管理员	栏目3发布员	…
Sinor	1	1	1	1	0	0	0	0	…
Applesiri	1	0	0	0	1	1	1	0	…
…	…	…	…	…	…	…	…	…	…

此类组织方式维护一个用户信息时只涉及一条记录，系统查找的时效性比前一种方式更优。但一个用户的处理涉及一条记录中的多个字段，当大批量用户数据需要维护时，处理效率就显得低下，而且数据库的结构也显得过于臃肿。

为了降低数据库操作和角色数据处理的复杂度，可以采用“压缩”的思路来改进数据的组织。其实上述第二种方式已经初步显现了这种思想的端倪，即用0，

1表示用户是否具有某个角色的权限，0表示不具备这个角色的权限，1表示具备。如果假定方式2中总共有8个角色（角色名称如表1.2.5所示），那么sinor所具备的角色整体就可用01串11110000来表示，实际保存时将该01串转换成十进制数（转换结果为240），这样，原来数据表字段就缩减为两个，一个保存用户名，一个保存该用户所对应的角色特征值。通过这种“压缩”处理后的数据表就瘦身为如下表1.2.6所示（每条记录包含2个字段）。

表1.2.6 “压缩”后的用户角色特征表

用户名称	用户角色特征值
Sinor	240
Applesiri	142

在这种数据组织方式下，当需要修改用户角色时，首先从数据库中读取该用户的角色特征值，然后将该值（十进制数值）转换为二进制数，通过各个角色对应二进制位的0，1改变实现角色的修改。为了解决上述处理过程中十进制转换为二进制数的问题，可以设计怎样的算法与数据结构？

项目任务

某数据表中采用了前面所述的“压缩”方法存储用户角色特征，现在需要将某个十进制角色特征值转换为一个对应的二进制数。具体要求如下：

1. 抽象与建模，将问题已知条件和求解目标用清晰的语言描述，然后建立一个可行的计算模型。

2. 设计算法与数据结构，并选择合适的方式描述算法，为后阶段用计算机程序求解提供支撑。

过程与建议

1. 抽象与建模

为了确定计算模型，还需要进一步明确条件。不妨假定该系统中一共有20个不同的、相互独立的角色分类，这就确定了最终转换生成的二进制数是20位的。

为了描述精确，不妨设读取的某用户特征值存放于变量x中，同时转换产生的二进制数各数位上的0或者1保存在数组a中。

请根据上述分析，进一步明确相关的数据（如果需要），在下方写出解决该问题的计算模型。

2. 设计算法与数据结构

依托前面建立的计算模型，根据已知数据及数据之间的关系设计数据结构，同时设计解决该问题的算法，并描述你的分析过程。

3. 描述算法

选用恰当的方式（自然语言、流程图或者伪代码等）来描述你的算法，帮助算法执行者顺利地按照你的算法逐步执行（如果要让计算机来实施处理，需要用计算机程序设计语言准确地描述该算法）。

4. 展示交流

为了推广你的项目，你需要将你的项目及解决方案在一定的平台、以一定的形式展示给大家，通过展示实现交流，并基于交流对项目做出进一步的完善。

评价标准

请根据项目实施的过程、效果以及成果展示交流的结果，对自己完成项目的情况进行客观的评价，并思考后续完善的方向。把评价结果和完善方案填写在下面的表格中。

评价条目	说明	评分（1～10分）	评分主要依据阐述	后续完善方向
界定问题	通过抽象与建模确定的问题明确，且已知条件充分，结果能求解			
算法与数据结构设计	设计的数据结构合理、算法正确，且能输出符合实际的正确结论			
算法描述	描述方式的选择或组合有助于正确理解算法			
展示方式	项目成果展示方式的选择利于他人较好地了解项目的意义以及问题求解的特点			
展示效果	在规定时间内有条理地、清晰地介绍研究成果			

拓展项目

1. 在停车场管理系统中，可以用数组来表示车辆和停车时间的对应关系，并基于数组完成各种车辆停车计费的运算。能不能运用树、链表等其他数据结构来优化其中某些环节中的数据运算？具体又是如何进行的？

2. 随着我国高铁事业的发展，高铁已经成为越来越多人外出旅游、求学、公务出差时

选用的交通工具，网上购票也日益成为高铁购票的主要方式。但每当春节、国庆等长假期间，网上购票总会出现系统繁忙、购票失败的现象。请从数据结构的角度分析造成此种现象的原因，并尝试提出完善的方案。

3. 社交平台不仅能让人们点对点地添加新朋友，而且能根据用户的交流记录，运用大数据技术，为用户挖掘潜在可能的新朋友。例如，每当用户通过QQ新加一个朋友时，系统会跳出一个对话框并推荐该用户和新朋友间的共同朋友。通过这种方式，可以帮助用户发现潜在想加的好友，促进人际交流。尝试分析其中数据的组织以及相应的处理机制。

第二章 2

数组与链表

计算机中数据的存储模式分为顺序存储和非顺序存储，所有数据结构的存储也可以分为这两类，而作为常用数据结构中的数组与链表，就是这两种存储模式的典型代表，其他的数据结构都是在这两种数据结构的基础上构建的。

问题与挑战

- 公共资源的分配涉及千家万户，当需求者远超资源量时，采用随机分配的方式既可以体现公平性，又容易被大众所接受。例如，一些大城市为了控制汽车数量、缓解交通压力，对汽车车牌实行随机摇号方式发放。车牌的摇号系统是如何组织、存储数据的？又是如何实现随机摇号功能的呢？

- 多项式合并同类项，当其项数多到一定程度时，用人工方式完成就显得比较烦琐，此时可以利用计算机速度快的特点去编程实现。如果由你来完成该功能，你会使用哪种形式来组织、存储数据？又会如何实现多项式的合并？

- 许多网络平台都有推荐功能，即当用户在平台中进行了某项操作后，平台会自动推荐相关内容给该用户。其中，有些平台的推荐功能是通过设计符合某种关联规则的算法并编程实现的，为了适应不同的算法，其中的数据该如何组织、存储？推荐功能的实现又是如何完成的？

学习目标

1. 理解数组和链表的概念和特性。
2. 掌握数组和链表的基本操作。
3. 能运用数组和链表编程解决实际问题。

内容总览

2.1 数组

数组在内存中的存储方式为顺序存储，适用于数据规模可预估且在处理过程中数据规模保持稳定的问题。

2.1.1 数组的概念与特性

数组作为常用数据结构，有特定的数据组织、存储结构及操作特性。

拓展链接

顺序存储结构和非顺序存储结构

计算机中数据的存储结构主要分为顺序存储结构和非顺序存储结构。其中顺序存储结构是将逻辑上相邻的数据节点存储在物理位置相邻的存储单元中，典型代表为数组；而非顺序存储结构的形式就是链式存储结构，在链式存储结构中可以将逻辑上相邻的数据节点在内存中分开存储，节点之间的前后关系由每个节点中的指针确定，典型代表为链表。

1. 数组的概念

数组是由相同类型的变量构成的一个序列。数组使用一个标识符命名，并用编号区分数组内的各个变量，这个特殊的标识符称为数组名，编号称为下标或索引。由数组名和下标组成数组的各个变量称为数组的分量，也称为数组元素。

数组在内存中存储的结构简单，创建数组时系统会分配一块连续的存储空间，每个数组元素按照下标顺序依次存储。

如图2.1.1所示，数组中的数据在内存中按照下标顺序依次存储，使用数组名指向第一个数组元素存储的位置，由于每个数组元素的类型相同，所需的存储空间一致，因此在明确第一个数组元素的存储位置后，可以利用下标计算出其他数组元素的存储位置，从而达到快速访问的目的。

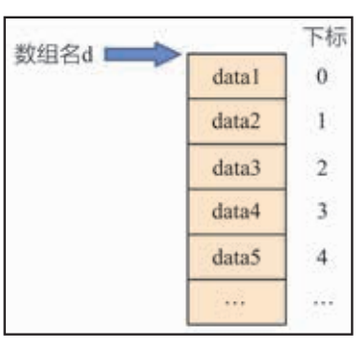


图2.1.1 数组的存储结构

一维数组适合用来表示具有线性特征的数据序列，因此只需要一个下标来表示数据元素在该序列中的位置。如果要表示二维空间内既有纵向联系又有横向联系的一批数据，那么采用二维数组会变得更形象、方便。由于二维数组中的数

据元素有行、列两个维度的位置信息，因此需要两个下标。

拓展链接

数组下标表示方式

不同程序语言中的数组，特别是多维数组的下标表示各有不同。有的语言中使用多个方括号分别表示其中一个下标，如：a[1][3]，典型代表有C、C++、Python等；有的语言中则是使用一个方括号包含所有下标，每个下标之间用逗号隔开，如：a[1,3]，典型代表有Pascal；有的则使用一个圆括号包含所有下标，每个下标之间用逗号隔开，如：a(1,3)，典型代表有VB。

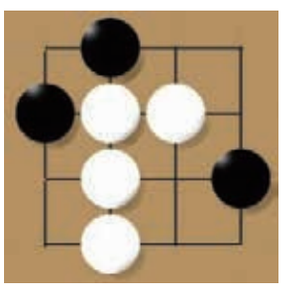


图2.1.2　平面棋盘与棋子布局

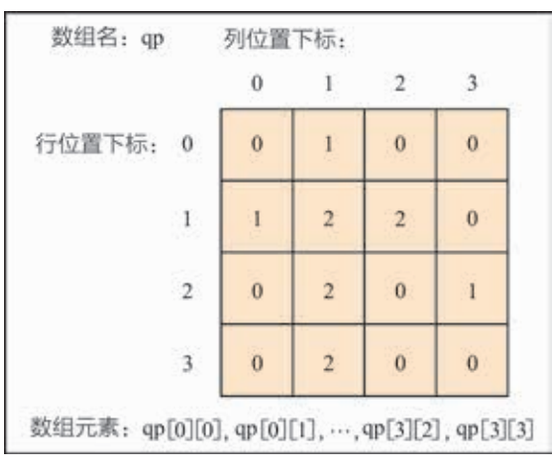


图2.1.3　二维数组描述的棋盘信息

为了能在计算机中有效地存储和处理如图2.1.2所示的棋盘信息，可以用如图2.1.3所示的二维数组来表示，并且约定：用第一个下标表示行位置、第二个下标表示列位置，行列下标均从0开始，并用0表示无子，1表示黑子，2表示白子。这样，第一行的棋盘信息就可以用数组元素表示为：qp[0][0]=0，qp[0][1]=1，qp[0][2]=0，qp[0][3]=0。

用二维数组表示的数据在内存中的存储方式也采用顺序存储，有行优先存储和列优先存储两种方式。以行优先存储方式为例，首先将第一行（行下标为0）数组元素按照列下标顺序依次存储，再将下一行（行下标为1）数组元素按照列下标顺序依次存储，直至所有行存储完毕。

图2.1.4所示为棋盘布局数据的二维数组行优先存储方式，同样，数组名指向数组中第一行第一个数组元素存储的位置，其他数组元素均可以通过其下标快速得到其存储位置。

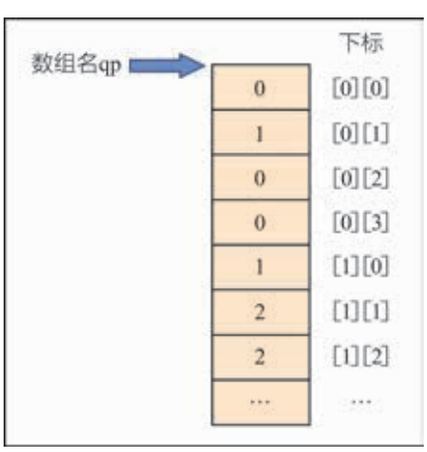


图2.1.4　二维数组行优先存储方式

2. 数组的特性

根据上述数组概念及其组织、存储结构的原理，可总结出数组具有以下特性。

（1）数组元素的数据类型相同

除非特别说明，数组指向的往往是静态数组，即定义一个数组后，系统会根据每个数组元素的数据类型和总元素个数，在内存中开辟一批地址连续且空间固定的存储空间。系统会根据下标及数组中存储的数据类型快速得到其存储位置并进行访问。所以，数组要求所有数组元素的数据类型是相同的。例如，一个数组中每个数组元素都是整型数据，或都是字符数据。

（2）通过数组名和下标对数组元素的值进行访问

因为系统会根据下标和数据类型计算内存地址访问数组中的数据元素，所以在对数组中的数据元素进行访问时，不能只提供数组名称（如a），而必须同时提供数组名称和数据元素对应的下标（如a[1]）。通过这种方式可以对数组中任意位置的数据元素进行高效的访问，如要访问一维数组a中第10个位置的数组元素的值，可以直接使用a[9]来执行。

（3）存储空间固定不变

在对数组进行操作的过程中，即使数组中的某些数据元素已经删除，但其占用的空间会继续保留。例如，在原先包含100个数据元素的数组中，如果删除了其中的50个数据元素，那么这50个位置不会挪作他用，后续的操作如需涉及这50个位置，则不必再次定义，可以直接使用。因此，从存储空间利用率的角度来说，数组适用于数据规模可预估且在处理过程中保持稳定的问题。具有这种特点的数组称为“静态数组”。

静态数组对应的最大元素数量在数组定义时就已经确定，在程序运行期间不会发生改变，但存在以下问题：一是有些问题很难预估需要多少存储空间；二是如果删除了很多数据元素，不能充分利用空余空间，导致系统内存的浪费。为了解决上述问题，有些高级语言采用了动态数组技术，允许数组的元素数量可以根据处理的需要进行增加或减少。如C、C++、Java等编程系统中的动态数组。

拓展链接

动态数组

动态数组就是在声明时没有确定数据规模的数组，可以在任何时候改变数据规模。数组到底应该有多大才合适，有时可能不得而知，所以希望能够在运行时具有改变数组大小的能力，于是引入了动态数组。

2.1.2 数组的基本操作

每一种数据结构都有其基本操作，对于数组来说，常用的操作有数组的创建、数组元素的访问、数组元素的插入与删除等。

不同的程序语言对数组的基本操作的实现各有不同。Python中没有数组这种结构，但提出了一种更灵活的数据结构——列表。当列表中每个数据元素所含数据项的数量和数据类型均相同时，可以将该列表视为数组，列表索引可视为数组的下标。因此，在Python中可以使用列表来实现数组的基本操作。

1. 数组的创建

数组的创建实质是在系统内存中划分一块连续区域，用来保存数组所含的所有数据元素。静态数组创建需指明数组中数据元素的类型和数量，系统根据类型和数量计算所需的容量并进行内存分配，同时会根据其类型给数组中的所有元素赋初值，如给整数类型赋初值为0、给字符串类型赋初值为空字符串等。创建成功后，数组中的数组元素与存储数据元素的内存地址之间就形成了一一对应的关系。在此基础上，可以实现数据元素的存储、访问、插入与删除等基本操作。

拓展链接

列表的存储

虽然可以使用列表来模拟数组，但是列表的存储方式与数组是截然不同的，列表中索引是连续的，但其数据元素在内存中存储时一般不是连续存储的，这是由Python中变量存储模式决定的，如定义1000个初始化为0的列表，这1000个数值为0的列表元素均指向同一个地址，可以使用id()函数进行验证。

●●● 例1 统计分数

学校元旦文艺会演比赛时，现场有9位评委给各班节目打分，统计系统需要根据9位评委的原始分计算平均分，作为各班表演节目的最终得分。

分析：该问题中9位评委给出的分数属于同一类型的数据，可以创建一个包含9个下标变量的数值型数组来存储评委的原始分。

创建保存评委原始分的一维数组s的程序如下：

程序	测试结果
s=[0]*9 print(s)	[0,0,0,0,0,0,0,0,0]

对于图2.1.3中的棋盘信息，可以创建一个4×4的二维数组来保存，并将所有数组元素的值初始化为0，表示所有位置均为无子状态。

在Python中使用列表实现二维数组有两种方式：一是直接定义，适合创建规模较小的二维数组；二是间接定义，适合创建规模较大的二维数组。

拓展链接

Python中二维数组的创建

Python中创建二维数组的方式比较特殊，不能使用如同一维数组的方式创建，若使用语句qp=[[0]*4]*4来创建，则在修改某行元素时，4行中同一列的数据会同时被修改。这是因为其实质是创建了一行数据的4个引用，因此修改某一行中的数据相当于修改了所有行中的数据。这种情况在Python中称为浅拷贝。

下面两个程序分别使用直接定义和间接定义两种方式创建保存棋盘信息的二维数组qp，并将其信息按行输出。

程序（直接定义）	测试结果
qp=[[0,0,0,0],[0,0,0,0],[0,0,0,0],[0,0,0,0]] for i in range(4): 　　print(qp[i])	[0, 0, 0, 0] [0, 0, 0, 0] [0, 0, 0, 0] [0, 0, 0, 0]

程序（间接定义）	测试结果
qp=[[0 for i in range(4)]for j in range(4)] for i in range(4): 　　print(qp[i])	[0, 0, 0, 0] [0, 0, 0, 0] [0, 0, 0, 0] [0, 0, 0, 0]

2. 数组元素的访问

数组元素的访问指的是寻址到特定的数据元素，并根据存储地址对该数据元素进行读取、修改等操作。由于数组元素可以通过数组名和下标快速确定数据元素的内存地址，因此可通过数组名和下标直接进行访问。例如，print(s[5])表示将一维数组s中第6个元素的值进行输出（第1个元素为s[0]）。

例2　基于数组实现数据合并的功能

第一章1.2节将“数据合并”问题简化成合并两个有序数据序列的问题，对如何使用数组解决数据合并问题进行了分析并设计了算法。下面将对算法进行修改和优化，避免插入数据时其他数据元素的移动，并编程实现该功能。

（1）设计算法

①初始化3个数组a，b，c，元素个数分别为n，m和n+m。数组a和数组b用来存储已有的两个有序数据序列（降序），每个有序数据序列中第一个数据使用随机函数randint(start,end)产生，其他数据则由前一个数据减去一个随机值（1～5）产生；数组c用于保存合并后的所有数据（降序），所有数组元素的值初始化为0。

②使用变量i指向数组a当前未处理数据中要处理的数据元素的位置，初始化为0；变量j指向数组b当前未处理数据中要处理的数据元素的位置，初始化为0；变量k指向数组c中可加入元素的位置，初始化为0。

③重复以下操作，直至数组a或数组b中的数据元素全部合并入数组c中：比较a[i]和b[j]的大小，若a[i]≥b[j]，则将a[i]放入数组c中，并将i和k的值增加1；否则，将b[j]放入数组c中，并将j和k的值增加1。

④如果数组a或数组b中尚有未处理的数据元素，那么将剩余数据元素按原有顺序依

次存入数组c中，合并完成，输出数组c。

（2）编写程序

用Python语言实现上述算法的程序如下：

程序	测试结果

from random import randint
 	
a=[0]*20
b=[0]*15
c=[0]*35
a[0]=randint(95,100)
for i in range(1,20): #随机产生降序数据序列一
  a[i]=a[i−1]−randint(1,5)
b[0]=randint(95,100)
for i in range(1,15): #随机产生降序数据序列二
  b[i]=b[i−1]−randint(1,5)
print("原始数据序列一为：")
print(a)
print("原始数据序列二为：")
print(b)
i=0
j=0
k=0
while(i<20 and j<15): #合并数据直至某个数组数据合并完成
  if a[i]>=b[j]:
  c[k]=a[i]
  i=i+1
  k=k+1
  else:
  c[k]=b[j]
  j=j+1
  k=k+1
while i<20: #当数据序列一中还有数据时的处理
  c[k]=a[i]
  i=i+1
  k=k+1
while j<15: #当数据序列二中还有数据时的处理
  c[k]=b[j]
  j=j+1
  k=k+1
print("合并后的数据序列为：")
print(c)
	原始数据序列一为：
	[99, 97, 95, 94, 90, 87, 83, 79, 78, 73, 71, 69, 67, 66, 61, 60, 56, 55, 52, 51]
	
	原始数据序列二为：
	[99, 96, 94, 91, 90, 88, 87, 82, 80, 79, 75, 73, 70, 66, 62]
	
	合并后的数据序列为：
	[99, 99, 97, 96, 95, 94, 94, 91, 90, 90, 88, 87, 87, 83, 82, 80, 79, 79, 78, 75, 73, 73, 71, 70, 69, 67, 66, 66, 62, 61, 60, 56, 55, 52, 51]

同样的，对于二维数组，也可以使用数组的数组名和下标来任意访问数组元素。

下面的程序实现了在空棋盘的第二行、第三列放置一个黑子（用1表示）并更新棋盘信息的功能。

程序	测试结果
qp=[[0 for i in range(4)]for j in range(4)] qp[1][2]=1 for i in range(4): 　　print(qp[i])	[0, 0, 0, 0] [0, 0, 1, 0] [0, 0, 0, 0] [0, 0, 0, 0]

3. 数组元素的插入与删除

当需要在数组中某个位置插入一个新的数据时，必须先将该位置及其后的所有数据向后移动一个位置，在保证顺序不变的前提下保存这些数据，最后再修改该位置上的数据为新数据。

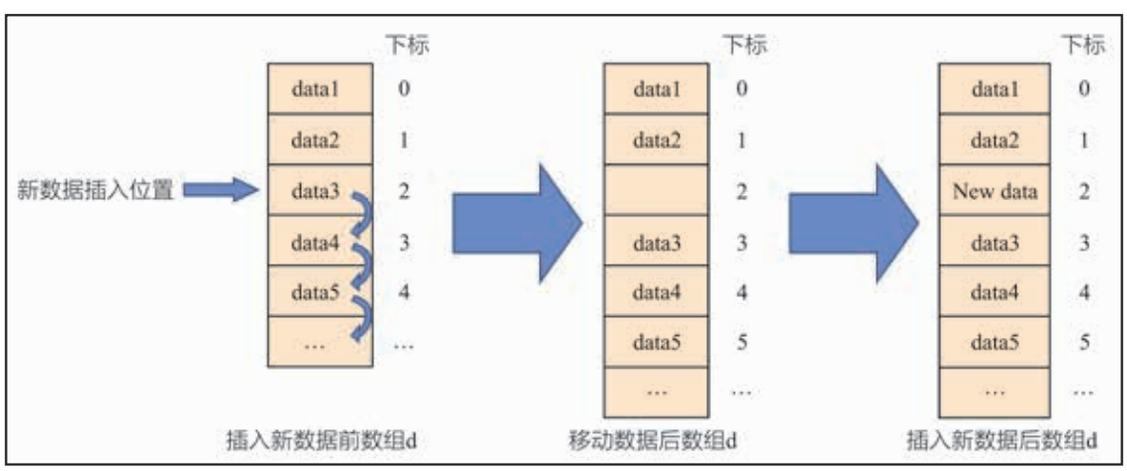


图2.1.5　数组中插入新数据的过程

如图2.1.5所示，当在数组d中第3个位置（d[2]）插入新数据时，需要分两个步骤：第一步，将该位置及其后所有数据依次向后移动一个位置；第二步，修改d[2]的值为新的数据。

问题与讨论

1. 在数组中插入新数据是否可以先插入数据，再移动其他数组元素？移动数组元素时，其顺序应该是怎样的？为什么一定要这样的顺序？

2. 参考数组元素的插入过程，分组讨论并得出在数组中删除一个数组元素的步骤及其注意事项。

对于静态数组，在增加数组元素时，需要将某位置及其后的所有元素后移一个位置后再插入，一方面时间效率较低，另一方面可能会超出限定的数组元素数量而导致数据的丢失；而在删除数组元素时，需要将被删除元素位置后的所有元素前移一个位置，同样有时

间效率低的问题，而且在删除比较多的元素后，数组中存储的有效数据减少，从而造成存储空间的浪费。因此，在使用数组组织、存储数据时应尽量避免数组元素的增删操作。

有了数组概念及数组的基本操作，可以使用数组组织、存储数据，帮助解决问题。

例3　基于数组的车牌摇号系统功能实现

汽车数量的急剧增加，导致城市交通的压力越来越大，许多大城市采取通过摇号方式来发放汽车车牌。在申请人通过资格审核后，车牌摇号系统反馈回一个唯一的编号。每次摇号前，车牌摇号系统需要收集所有本次申请人的编号，再在所有编号中随机抽取不重复的若干个编号来发放车牌。

（1）抽象与建模

用n表示有资格参加摇号的申请人总数，用数组luck存储编号，luck[0]，luck[1]，…，luck[n−1]依次表示n个编号，其中的下标表示每个编号在数组中的位置，luck[i]的初值为第i+1位申请人的编号。使用变量m表示最终的车牌发放数量，计数器c表示已抽中人数，每当随机抽出一个有效的下标位置（如k）时，计数器c加1，输出申请人的编号后将该下标位置（如luck[k]）的值设为空字符串（表示该编号已被抽取），用来判断其后抽取的编号是否重复。重复该过程直至计数器c的值变为m，摇号结束。

（2）设计算法

由于该问题中数据规模可以预估，在处理过程中其数据规模保持不变，并需要根据随机产生的编号读取和修改对应的值，所以用数组来存储所有申请人的编号，下标表示该编号的位置。算法分为两个步骤：

①根据申请人总数n，初始化数组luck，其数组下标代表位置，数组元素值为申请者编号（类型为字符串），并使用计数器c表示已抽中人数，初始化值为0。

②使用随机整数函数randint(start,end)随机产生一个位置k，若其作为数组下标对应的数组数据元素值为空串，则表示该编号已被抽取，该编号为重复号码，需要重新生成；否则，表示该编号为有效编号，计数器c加1，输出对应的数组元素并修改数组元素值为空串，表示该编号已被抽取。

③重复执行步骤②，直至计数器c变为m，程序结束。

（3）编写程序

以下程序模拟实现从20个申请人编号（存储在文件bh.csv中）中抽取m（待输入）个作为车牌发放编号。

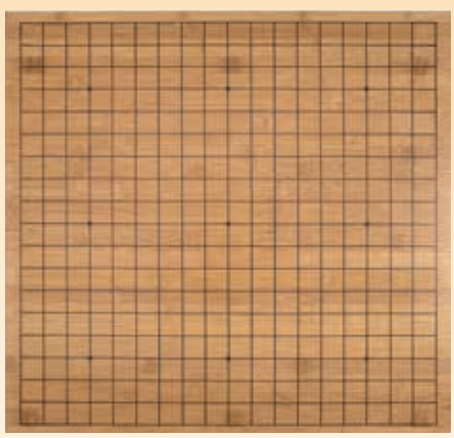
程序		测试结果
from random import randint		测试结果1： 请输入发放数：5
luck=[""] * 20	#初始化一个包含20个数据元素的数组	201811244728
csv_file=open("bh.csv", "r")	#打开存有编号的文件bh.csv	201806217856
flines=csv_file.readlines()	#将文件中所有编号按行读入flines中	201809042816
csv_file.close()	#关闭文件	201806197749
i=0		201809083618
for one_line in flines:		
tmp=one_line.strip("\n")	#将一个编号去除换行后赋给tmp	测试结果2： 请输入发放数：8
luck[i]=tmp	#按读取顺序将编号赋给数组元素luck[i]	201804115630
i=i + 1		201806197749
m=int(input("请输入发放数："))		201802053412
c=0		201809083618
while c < m:		201803095525
k=randint(0,19)		201805156848
if luck[k] != "":	#判断luck[k]是否已被抽取	201810163927
c=c + 1		201806217856
print(luck[k])		
luck[k]=""	#表示该位置编号已被抽取	

上述程序是用Python中的列表来模拟实现数组的操作，还可以充分利用Python系统自带的有关列表的函数与方法来简化代码，列表常见的函数与方法如表2.1.1所示。列表的索引（下标）支持负数，如luck[−1]指向数组luck中最后一个元素。

表2.1.1　Python列表常用函数与方法

函数和方法	功能	实例
len(list)	统计列表list中元素个数	list=[] print(len(list)) 输出为：0
list.append(x)	在列表list末尾添加元素x	list=[22,33,44] list.append(55) print(list) 输出为：[22,33,44,55]
list.insert(i,x)	在列表list中下标为i位置处插入元素x	list=[“A”,”B”,”D”] list.insert(2,”C”) print(list) 输出为：[“A”,”B”,”C”,”D”]
list.pop(i)	将列表list中下标为i的元素删除；若i不指定，默认为−1，即最后一个	list=[“A”,”B”,“C”,”D”] list.pop(2) print(list) 输出为：[“A”,”B”,”D”]

使用以上列表的函数和方法后，可以将由列表模拟实现的数组视为动态数组，其中的增加、删除数组元素的操作由相应的函数实现。



问题与讨论

请列举几个包含相同类型数据的实例或问题，并讨论其中的数据是否适合使用数组来组织、存储。

实践与体验

简易五子棋游戏

五子棋是世界智力运动会竞技项目之一，是一种两人对弈的纯策略型棋类游戏。游戏双方分别使用黑白两色的棋子，下在棋盘直线与横线的交叉点上，先形成5子连线者获胜。棋具与围棋通用，标准棋盘横竖各19条线，互相交叉形成361个点作为棋子位置，如图2.1.6所示。

图2.1.6　五子棋棋盘

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

图2.1.7　使用二维数组描述的空棋盘

实践内容：

使用二维数组表示棋盘上的布局情况，可以将棋盘中每一个位置看成是二维数组中的一个元素，每一枚棋子落到棋盘上便改变二维数组中该位置数组元素的值。图2.1.7所示即为使用二维数组描述的空棋盘信息。

设定棋盘大小为5 × 5，请利用二维数组编程实现“简易五子棋游戏”。

实践步骤：

1. 创建一个5 × 5的二维数组，每个元素值初始化为0表示开始时棋盘为空的状态。
2. 依据黑方先下的规则，黑白交替输入棋子的位置，更改相应位置的二维数组中的元素的值，黑棋为1，白棋为2。
3. 输出每次更改后棋盘（二维数组）的状态。

结果呈现：

程序（代码文件可从教学资源网站下载）运行后部分界面如下面三图所示。

图2.1.8　黑方落子

图2.1.9　白方落子

图2.1.10　游戏结束

思考与练习

1. 参考二维数组的行优先存储方式，画图模拟二维数组的列优先存储方式。观察并比较行优先存储与列优先存储中相同元素的存储位置的改变，如qp[1][2]在行优先存储方式下在序列的第7个位置，则其在列优先存储方式下在序列的第几个位置？

2. 参考摇号系统的功能实现，将自己班级所有同学的名字以一个名字一行的形式存入文件name.csv，并使用文件读入的方式，使用数组实现随机抽奖程序。

2.2 链表

链表是一种存储空间利用率高的数据结构，适用于数据规模不确定或初始时确定但在处理过程中由于频繁增、删数据导致数据规模不稳定的问题。

2.2.1 链表的概念与特性

链表的核心是使用指针链接节点，是一种非常灵活的数据组织方式，与数组相似，链表在组织、存储结构上有其特性，了解并掌握这些有助于使用链表组织、存储数据，并设计算法解决问题。

1. 链表的概念

链表指的是将需要处理的数据对象以节点的形式，通过指针串联在一起的一种数据结构。链表中的每个节点一般由“数据区域”和“指针区域”两部分构成，如图2.2.1所示。数据区域用于保存实际需要处理的数据元素，指针区域用来保存该节点相邻节点的存储地址，通过该地址指向（指针）来实现从当前节点按顺序走到其相邻的节点。某个节点前面的相邻节点称为该节点的前驱节点，后面的相邻节点称为该节点的后继节点。

图2.2.1 链表节点结构

每个链表拥有一个表头——head（也称头指针），头指针的作用之一是链表的入口，只有通过头指针才能进入链表；作用之二是为循环链表设立一个边界，便于数据处理时的边界判断和处理。

链表可以根据每个节点中指针的数量分为两类：当只有一个指针用来指向其后继节点时，称为单向链表；当有两个指针分别用于指向其前驱节点和后继节点时，称为双向链表。有时根据解决问题的需要，会把第一个节点和最后一个节点使用指针链接，这样就形成了循环链表。

如图2.2.2所示，单向链表中各个节点在内存中可以非顺序存储，每个节点使用指针指向其后继节点的存储地址。进入链表只能通过头指针head，其他节点则需要经过所有在它之前的节点才可以访问，尾节点的指针指向为null，表示指向为空。

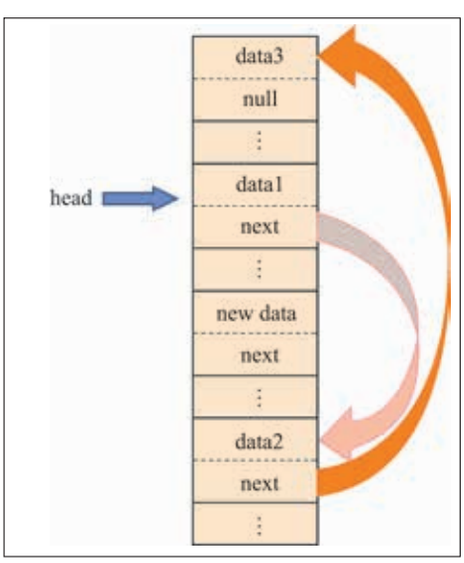


图2.2.2 单向链表在内存中的存储模式

问题与讨论

参考单向链表的节点结构及其指针指向，讨论并描述双向链表和循环链表的节点结构及其指针指向。

2. 链表的特性

根据链表的概念及其结构特点，可总结出链表的一些特性，理解这些特性，有助于合理地运用链表解决问题。

（1）同一链表中每个节点的结构均相同

链表节点中包含数据区域和指针区域。不管是单向链表还是双向链表，每个节点的数据区域中的数据类型是相同的，指针区域中的指针数量和功能是相同的。

（2）每个链表必定有一个头指针，以实现对链表的引用和边界处理

链表的头指针使用变量head表示，当需要引用链表时可以使用该变量进入链表，也就是说，访问链表中的某一节点，只能从头指针开始，通过指针链接依次访问，不能像数组那样通过下标直接引用。对于循环链表，一轮访问的开始和结束都可以借助头指针指向位置来进行判断，即边界处理。

（3）链表占用的空间不固定

链表的节点间通过指针相连，相邻节点存储时不需要连续空间，充分利用了内存的零散空间，提高了存储空间利用率。由于链表的存储空间由节点数决定，增加或减少节点都会改变链表占用的存储空间，因此链表占用的空间不固定。

2.2.2 链表的基本操作

链表的基本操作主要有空链表的创建、节点访问、节点插入和删除等。

Python中没有直接定义链表结构，可以使用其提供的列表来模拟实现。例如，实现单向链表时，列表中每个数据项作为链表中的一个节点，包含两个数据，其中一个作为数据域存储具体数据，另一个作为指针域存储后继节点在列表中的索引，另外需要一个变量head用来存储头节点在列表中的索引，可以视为头指针。

链表在实际应用时可以带头节点或不带头节点，在第一章中介绍链表时采用的都是带有头节点的结构，这样可以方便空链表的判断，而本章中链表使用Python中的列表模拟实现，用列表的索引来代替地址指针，并规定列表索引均为正索引，当某个节点的指针区域值为−1时表示指向为空，该节点为尾节点，此时空链表的判定可以使用更方便的方式进行，因此本章中的链表实现采用不带头节点的结构。

下面结合Python中的列表来模拟实现单链表的常用操作。

1. 链表的创建

创建链表时，首先要根据问题特点规划节点的数据域和指针域，然后根据规划创建一个空链表。接下来就可以根据输入的实际数据形成节点并逐步插入到已有的链表中。如，可以使用下列代码在Python中创建一个新的链表。

```
item=[]
head=-1
```

其中head值为-1，表示头指针指向为空，该链表为空链表。

2. 链表节点的访问

链表只能通过头指针（head）进行访问，其他节点通过节点间的指针依次访问。如图2.2.2，当想访问单向链表中data3所在的节点时，需通过头指针进入链表，再分别按照各个节点的指针指向经过data1和data2所在节点，最后通过data2所在节点的指针才能访问data3所在的节点。

3. 链表节点的插入与删除

链表节点的插入指的是根据新输入的实际数据形成节点，然后修改新节点与其前驱节点的指针，将新节点插入到链表的正确位置。链表节点的删除，则通过将需要删除的节点的前驱节点和其后继节点直接相连的方式实现。

以单向链表插入新节点为例，在空单向链表中插入第一个节点或在非空单向链表中插入第一个节点时，需要先修改新节点的指针指向与头指针一致，再修改头指针指向新节点；在非空单向链表两个节点中间插入新节点时，需要先修改新节点指针指向与前一节点的指针指向一致，再将前一节点的指针指向新节点，如图2.2.3所示。

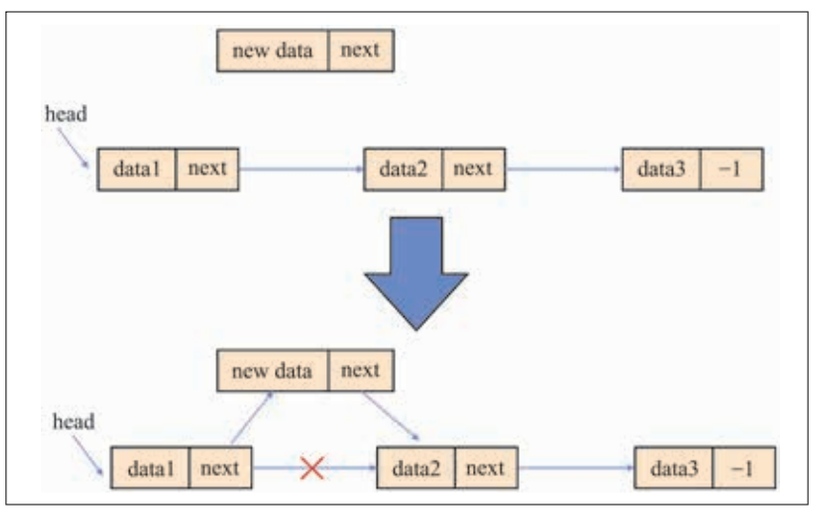


图2.2.3　单向链表中插入新节点过程

问题与讨论

1. 在单向链表中插入新节点时，指针指向的修改是否必须有先后？如果将其顺序逆转，能否完成新节点的插入？为什么？

2. 如图2.2.3，如果在尾节点之后插入新节点，那么节点指针该如何修改？

以上操作都是建立在链表节点不连续存储的基础上，当使用列表模拟实现链表时，其节点中指针区域存储的是后继节点在列表中的索引。

如图2.2.4所示，使用列表data来模拟链表结构，头指针head指向列表索引2，表示该索引位置为链表中的第一个节点，其中data[2][0]为数据区域，表示链表中第一个节点中的数据，data[2][1]为指针区域，表示其后继节点在列表中的索引。其中索引值−1表示指向为空，该节点为链表中的尾节点。

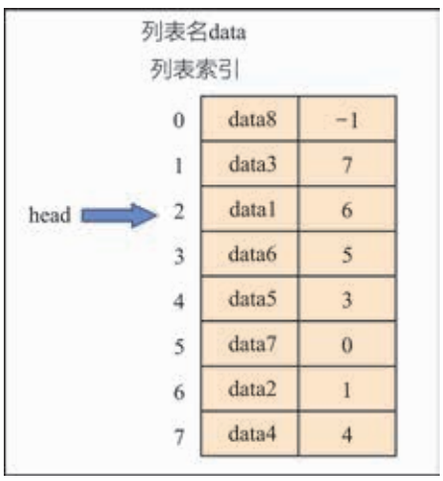


图2.2.4　使用列表模拟链表结构

在列表模拟实现的链表中插入新节点时，一般先在列表尾部插入新的节点，再寻找其前驱节点在列表中的索引，然后修改相关节点的指针区域的值。

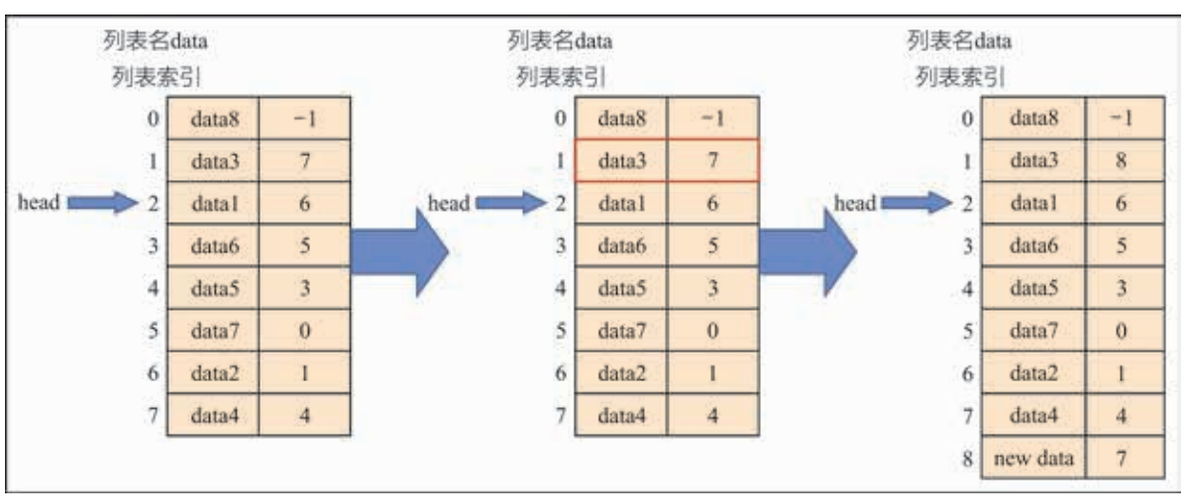


图2.2.5　列表模拟链表中插入新节点过程

如图2.2.5所示，在链表第3个节点（data3所在节点）后插入一个新节点（new data所在节点）时，先通过头指针进入链表并找到其前驱节点（data3所在节点）的列表索引1，然后在列表尾部增加一个新节点（列表中索引为8），并先后修改新节点及其前驱节点的指针区域的值。其中修改指针区域值的语句如下：

```
data[8][1]=data[1][1]
data[1][1]=8
```

问题与讨论

参考图2.2.5，描述在有8个节点的单向链表中删除第一个节点、中间节点及尾节点的过程。

例1　基于链表实现数据合并功能

在第一章中对于该功能如何使用链表实现进行了介绍，在本章中则对算法进行细化并最终编程实现该功能，目标是将两个链表中的数据合并到其中一个链表中。

（1）设计算法

①初始化两个空链表data_a和data_b，并使用head_a和head_b作为两个链表的头指针，其中data_a作为存储结果的链表。

②模拟生成两个降序序列数据，每个序列中第一个数据使用随机函数randint(start,end)产生，其他数据则由前一个数据减去一个随机值（1～5）产生，产生的数据作为新节点的数据区域的值，生成新的节点在尾部插入。

③使用变量k_a与k_b指向两个链表中未合并的数据序列中最前面（值最大）的节点，初始化k_a=head_a，k_b=head_b，由于在链表data_a中需要进行插入节点的操作，必须记录插入位置的前驱节点，使用变量q_a，初始化q_a=head。重复以下操作，直到k_a或k_b指向空（即某个链表元素全部处理完）：比较data_a[k_a][0]和data_b[k_b][0]的大小。若data_a[k_a][0]≥data_b[k_b][0]，则修改q_a与k_a相等，再将k_a指向下一个节点；否则将链表data_b中k_b指向的节点插入到链表data_a中，作为q_a指向节点的后继节点，再将k_b指向链表data_b中的下一个节点。

④若k_b未指向空，则将链表data_b剩余节点按顺序插入链表data_a的尾部。

⑤输出链表data_a中每个节点的数据区域的值。

（2）编写程序

程序	测试结果
from random import randint data_a=[] head_a=-1 data_b=[] head_b=-1 tmp=randint(95,100) data_a.append([tmp,head_a]) head_a=0 for i in range(1,20): 　　tmp=data_a[i-1][0]-randint(1,5) 　　data_a.append([tmp,data_a[i-1][1]])	链表结构的原始数据序列— [[99, 1], [98, 2], [94, 3], [93, 4], [91, 5], [89, 6], [85, 7], [84, 8], [79, 9], [75, 10], [72, 11], [71, 12], [66, 13], [64, 14], [59, 15], [54, 16], [52, 17], [48, 18], [44, 19], [43, -1]]

程序	测试结果

data_a[i−1][1]=i
print("链表结构的原始数据序列一")
print(data_a)
tmp=randint(95,100)
data_b.append([tmp,head_b])
head_b=0
for i in range(1,25):
data_a[i−1][1]=i
tmp=data_b[i−1][0]-randint(1,5)
data_b.append([tmp,data_b[i−1][1]])
data_b[i−1][1]=i
print("链表结构的原始数据序列二")
print(data_b)
k_a=head_a
q_a=head_a
k_b=head_b
while(k_a!=−1 and k_b!=−1):
if data_a[k_a][0]>=data_b[k_b][0]:
q_a=k_a
k_a=data_a[k_a][1]
else:
if k_a==head_a: #在链表data_a的头部插入节点
data_a.append([data_b[k_b][0],head_a])
head_a=len(data_a)−1
q_a=head_a
k_b=data_b[k_b][1]
else:
data_a.append([data_b[k_b][0],k_a])
data_a[q_a][1]=len(data_a)−1
q_a=data_a[q_a][1]
k_b=data_b[k_b][1]
while k_b!=−1:
data_a.append([data_b[k_b][0],−1])
data_a[q_a][1]=len(data_a)−1
q_a=data_a[q_a][1]
k_b=data_b[k_b][1]
print("链表结构的合并后数据序列")
print(data_a)
print("按链表链接顺序输出数据序列")
tmp=head_a
while data_a[tmp][1]!=−1:
print(data_a[tmp][0],end=" ")
tmp=data_a[tmp][1]
print(data_a[tmp][0])
	链表结构的原始数据序列二
	[[98, 1], [95, 2], [93, 3], [91, 4], [90, 5], [89, 6], [84, 7], [80, 8], [79, 9], [75, 10], [71, 11], [69, 12], [68, 13], [63, 14], [58, 15], [56, 16], [52, 17], [47, 18], [42, 19], [41, 20], [38, 21], [34, 22], [32, 23], [29, 24], [24, −1]]
	
	链表结构的合并后数据序列
	[[99, 1], [98, 20], [94, 3], [93, 22], [91, 23], [89, 25], [85, 7], [84, 26], [79, 28], [75, 29], [72, 11], [71, 30], [66, 13], [64, 33], [59, 34], [54, 16], [52, 36], [48, 37], [44, 19], [43, 38], [98, 21], [95, 2], [93, 4], [91, 24], [90, 5], [89, 6], [84, 27], [80, 8], [79, 9], [75, 10], [71, 31], [69, 32], [68, 12], [63, 14], [58, 35], [56, 15], [52, 17], [47, 18], [42, 39], [41, 40], [38, 41], [34, 42], [32, 43], [29, 44], [24, −1]]
	
	按链表链接顺序输出数据序列
	99 98 98 95 94 93 93 91 91 90 89 89 85 84 84 80 79 79 75 75 72 71 71 69 68 66 64 63 59 58 56 54 52 52 48 47 44 43 42 41 38 34 32 29 24

4. 链表节点的访问与遍历

由于链表中的节点通过指针相互链接，当需要访问某个位置的节点元素时，只能通过头指针进入链表并通过节点间的链接关系一个一个往下访问，直到找到指定位置的节点。与数组相比，其节点的访问效率较低。

实现链表及其基本操作后，可以使用其存储和处理数据，帮助人们解决问题。

例2　约瑟夫问题

n个人排成一圈，从某个人开始，按顺时针方向从1开始依次编号。从编号为1的人开始顺时针“1，2，3，…，m，1，2，3，…”报数，报到m（m>1）的人退出圈子。这样不断循环下去，圈子里的人数将不断减少。由于人数是有限的（n个），因此最终会只剩下一个人。试问最后剩下的人的初始编号是多少？

（1）抽象与建模

该问题中的关键数据是n个参与人员的初始编号，1至n的序列。从编号1开始计数，每过一个编号加1，当计数到m时将该编号从数据序列中移除，并从该编号的后一编号从1开始重新计数。而当计数到序列中最后一个编号，又从序列的开始编号继续计数，从而将计数序列构成一个环。重复这个过程，直到序列中只剩一个编号，该编号即为最后剩下人员的初始编号。

（2）设计数据结构与算法

该问题中数据规模在初始时可以确定（最大为n），但是在数据处理过程中需要按照规则不断地移除数据，直至只剩一个为止。也就是说，数据规模在处理过程中逐渐变小，呈现不稳定的特性，符合链表的应用。

因为最终需要输出初始编号信息，所以链表每个节点中数据区域用来保存初始编号，指针区域需要一个用来保存指向后继节点的指针。同时，因为序列中最大编号报数后会从序列中最小编号继续报数，所以可以采用单向循环链表来组织数据。基于链表这种数据结构的设计，可以设计出相应的算法如下：

①创建一个由n个节点组成的单向循环链表，并使报数计数器i的初始化值为1，同时当前报数人的指针k指向链表中第一个节点。

②重复执行下列处理，直到链表中只剩下一个节点：随着报数的进行，指针k不断指向下一个节点，报数计数器i也随之增加，当i增加到淘汰数m时，将对应的链表节点删除，若删除的节点为头指针指向的节点，则需同时修改头指针的指向；在删除节点的同时，需要重置报数计数器i的值为1。

③将链表中唯一节点，也就是头指针指向的节点中的数据（即初始编号）输出。

（3）编写程序

程序	测试结果

llist=[]
n=int(input("请输入参与人数（N）:"))
m=int(input("请输入淘汰数（M）:"))
for i in range(n-1):
　　llist.append([i+1,i+1])
llist.append([n,0])　　#将尾节点的指针指向头节点，构成循环单向链表
head=0
long=n
k=head
i=1
while long>1:
　　i=i+1
　　if i==m:
　　　　t=llist[k][1]
　　　　llist[k][1]=llist[t][1]
　　　　if t==head:　#删除节点为头指针指向的节点
　　　　　　head=llist[k][1]
　　　　i=1
　　　　long=long-1
　　k=llist[k][1]
print(llist[head][0])
	测试结果1：
	请输入参与人数(N)：400
	请输入淘汰数(M)：2
	最后一人的起始编号是：289
	
	测试结果2：
	请输入参与人数(N)：5000
	请输入淘汰数(M)：15
	最后一人的起始编号是：152

拓展链接

链表的类实现

在Python中，链表除了可以使用列表来实现，还可以使用“类”来实现。“类”是一种抽象的数据结构，它将数据及其操作封装在一起。构造单向链表类的具体实现过程如下：

1. 自定义单向链表的节点类，代码如下：

```
class LinkNode:                          #定义节点类LinkNode
def __init__(self,data,next_=None):      #初始化节点包含两个区域self.data、self.next
    self.data=data                       #self.data区域保存数据
    self.next=next_                      #self.next区域保存指针
```

2. 构造单向链表类，代码如下：

```
class LinkList:                          #定义单向链表类LinkList
    def __init__(self):                  #初始化空链表
        self._head=None                  #空链表头指针指向为空
```

有了以上类的构造，就可以创建单向链表并进行空表判断、增删节点等基本操作。使用自定义类的方式建立的单向链表可以充分体现链表的特性。

思考与练习

1. 请在单向链表在内存中的存储模式图（图2.2.2）的基础上，根据双向链表和循环链表的特性，画出它们在内存中的存储模式图。

2. 参考图2.2.3，描述在有3个节点的单向链表中删除第1个、第2个及第3个节点的过程。

3. 数组与链表作为存储相同类型数据的两种数据结构，拥有各自的应用场景、组织结构和操作特性，请根据教学内容进行简要概括并完善以下表格：

	数组	链表
应用场景	适合数据规模确定且在处理过程中保持数据规模稳定的问题	
组织结构		由节点构成，每个节点中包含数据区域和指针区域，相邻节点间通过指针链接
操作特性	优点：数据访问效率较高 缺点：使用前需限定最大空间，容易造成空间资源的浪费	

4. 参考教材中单向链表的列表实现，请思考如何使用列表实现双向链表的结构及其基本操作，思考完善后编程实现双向链表节点的创建、增加、删除及显示等。

巩固与提高

1. 小华规划自驾游路线，出发地为杭州，目的地为北京，在规划过程中经过了多次更改。第一次依次加入的途经地为上海、苏州、南京、济南、石家庄；第二次在南京和济南之间加入了途经地青岛，取消了途径地南京；第三次则在石家庄和北京之间加入了途经地天津。请使用链表编程实现其更改过程，并输出三次更改路线的结果。

2. 多项式的合并同类项对于同学们来说并不陌生，但是当其项数多到一定程度时，用人工方式完成就显得比较烦琐，此时可以通过计算机编程来实现。问：对于两个多项式的合并，可以使用哪种数据结构来组织、存储数据？又该如何实现多项式的合并？

3. 用除2取余法可以将十进制数转为二进制数。分别使用数组和链表编程实现输入一个十进制数后，通过程序输出该数转为二进制的结果。

4. 随着生活水平的不断提高，人们越来越关注自然环境，其中空气质量指数是重要指标之一。根据不同的空气质量，空气质量指数分为优、良、轻度污染、中度污染和严重污染。请收集（或模拟）最近一个月本地的空气质量指数，分别使用数组和链表编程统计其中优、良的天数占总天数的比例。

5. 随机抽奖程序可以使用数组组织、存储规模可预估的参与者数据，随机产生若干个不同的数组下标并输出其对应的中奖者姓名，该过程体现数组访问效率高的操作特性。尝试使用链表设计算法实现随机抽奖，并对比与分析在该程序中使用数组有哪些优势。

项目挑战

学校微课平台推荐功能设计

在购物网站中，当用户购买了一件商品后，系统会依据该商品的类型或用途给用户推荐相关的商品；在视频网站平台中，当用户打开一个视频后，平台会依据该视频的主演或导演给用户推荐相关的视频；在电子图书系统中，当用户借阅某本书籍后，系统会依据该书籍的内容或作者给用户推荐相关的书籍等等，以上这些系统均使用了推荐功能。

推荐功能就是根据用户的信息需求、行为等，依据一定的推荐规则，将用户可能感兴趣的信息、产品等推荐给用户。

项目任务

学校新建了一个微课平台，用来展示校内教师和学生自制的微课。为了更好地服务师生，学校想增加微课平台的推荐功能，即当某个用户点播了某个微课后，平台会依据一定的规则推荐相关的微课。

假如你作为平台的设计者，现在需要你来开发实现这个微课平台的推荐功能。

过程与建议

1. 根据选择或自定的推荐规则来确定并构建相应的问题模型

推荐规则是实现推荐功能的基础，如有以下两种推荐规则：

（1）根据某个用户某次点播微课的操作，在点播列表中统计出与该用户类型相同的用户点播次数（如该用户类型为“高一”，则统计点播列表中高一用户点播过的所有微课的点播次数）最多的若干个微课作为推荐对象。

（2）根据某个用户某次点播微课的操作，在点播列表中，统计出与该微课类型相同的所有微课点播次数，将次数最多的若干个微课作为推荐对象，如微课类型为“语文”，则推荐点播列表中“语文”类型微课点播次数最多的那些微课。

在本项目推荐规则中将不同的点播记录关联起来的字段称为关联字段，如推荐规则1中的“高一”字段，推荐规则2中的“语文”字段。

如果你有自己的想法，也可以自己设计相应的推荐规则并按后续步骤实施。

（1）请描述你选择或设计的推荐规则：_____

_____

（2）你选择或设计的推荐规则的数据特点是什么？是否可以使用数组、链表来实现？

注意：主要考虑相关数据的数据量是否可以事先确定，如果可以，就优先考虑数组；反之，一般会优先考虑链表。

2. 依据某个数据来源，结合上述的推荐规则，分析数据的内容结构

提供教学视频的网站比较多，包括网易公开课，国家开放课程等。请选定一个网站进行浏览，想象用户点播课程时，在平台上会存储怎样的数据，以便后期的推荐。这些数据可能包括（但不限于）如下内容：

☐ 用户 ID　☐ 用户类型　☐ 微课名称　☐ 微课类别　☐ 其他：

3. 确定算法和数据结构

根据前面的推荐规则、数据内容等，考虑完成本项目所采用的算法和数据结构。

考虑算法与数据结构时，要思考：

（1）在这个项目中，选择这个算法及相应的数据结构的理由是什么？

（2）每个字段，在推荐时所起的作用是什么？

（3）算法中的关联字段（如推荐规则 1 中的“高一”，推荐规则 2 中的“语文”）的设置是否合理？

4. 编写程序并测试

根据你所确定的数据内容，模拟填写相关的点播列表信息（wk_click.csv），这个模拟数据表要求：

（1）有一定的数据量（比如说，50 条以上）。

（2）作为关联字段，一定要有同类型的数据存在，以便验证推荐算法。比如说，如果将“微课类型”作为关联字段，那么同一微课类型的数据应不止一条。

5. 展示交流

和同学分别展示自己的成果，评选谁的推荐程序更好。包括以下内容：

（1）选择或设计的推荐规则及理由。

（2）选择的数据结构以及理由（主要依据数据特性）。

（3）展示程序运行的结果，说明当前的效果。

（4）后续改进的想法。

评价标准

请根据项目实施的过程、效果以及成果展示交流的结果，对自己完成项目的情况进行客观的评价，并思考后续完善的方向。把评价结果和完善方案填写在下面的表格中。

评价条目	说明	评分（1～10分）	评分主要依据阐述	后续完善方向
推荐规则的选择或设计	选择或设计的推荐规则容易理解并切实可行（注：自行设计的推荐规则可适当加分）			
数据内容结构设计	设计的数据内容结构明确，可以满足推荐规则的要求			
数据的模拟	模拟数据可以验证推荐规则的正确性，数据量达到基准以上			
数据结构的选择	选择的数据结构与数据特性及推荐规则相适应			
程序编写、调式及运行	程序调式、运行成功，所得数据符合预期			

拓展项目

1. 校园安全越来越受到社会各界的重视，现在几乎每个学校门口都对外来人员进行进出登记。小高是一名刚刚接触编程的高中生，他发现自己所在学校门口进出登记还是手写形式，当某一时刻来访的人数较多时，手写登记就显得效率很低。于是他想帮学校设计一个简易的外来人员进出登记系统，其功能如下：

（1）当有外来人员进入时，在身份证信息采集设备上刷一下身份证，系统从中收集该人员的姓名信息，并将其按进入顺序放入外来人员的序列中。

（2）学校管理人员或门卫可以查看当前时刻在学校内的外来人员信息，查看结果以外来人员进入学校的顺序显示。

（3）当有外来人员离开时，也只需在身份证信息采集设备上刷一下身份证，系统将其从来访人员序列中删除。

请帮助小高确定合适的数据结构来存储数据，并依据选择的数据结构设计算法，编程实现该系统。

2. 约瑟夫问题是经典的链表应用，其特性是数据规模由于在处理过程中人员不断减少而呈现不稳定，但其数据规模的最大值是确定的。是否可以根据这一特性，在维持数据规模稳定的前提下，使用数组设计算法解决该问题？

3. 随着社会经济的发展，我国家用车辆越来越多，这也加速了大型停车场的发展。现在很多停车场都有一个实用的功能，在进入停车场的门口设立电子屏幕，上面显示停

车场的空余车位数量，以便让进入停车场的司机了解停车场中空余车位的多少。该提示功能每隔一定的时间（如15秒）会实时更新，其自动更新的实现过程如下：

（1）停车场中每个车位上都设置有感应装置，可以实时感应所在车位是否有车停驻，更新时会将所有车位的状态传输给管理系统。

（2）管理系统根据接收到的某个时刻所有车位状态的感应信息，统计停车场中空余车位的数量。

（3）将实时统计的结果显示在停车场门口的电子屏幕上。

请结合以上描述，分析其中的关键数据，选择合适的数据结构，编程实现某一时刻停车场空余车位数量的统计，并输出结果。

第三章 3

字符串、队列和栈

在文档中寻找某个关键词时，查找功能可以快速定位到该关键词所在的位置；去银行办理业务时，取号系统可以按照客户到达时间的先后顺序，合理地安排办事次序；单击浏览器的“返回”按钮，可以按访问顺序依次回访网页。这些功能的实现过程中分别用到了字符串、队列、栈等基本数据结构。

问题与挑战

- 在错峰限行的城市中，交通管理部门要快速找出那些不按规则上路行驶的车辆；电子邮箱中经常会接收到一些包含错误或虚假信息的垃圾邮件，邮件系统需要过滤这些垃圾邮件；网络搜索引擎可以根据关键词快速搜索到所要的信息。对于上述问题，涉及的信息量非常大，计算机程序是如何提取数据、组织数据、分析数据的？

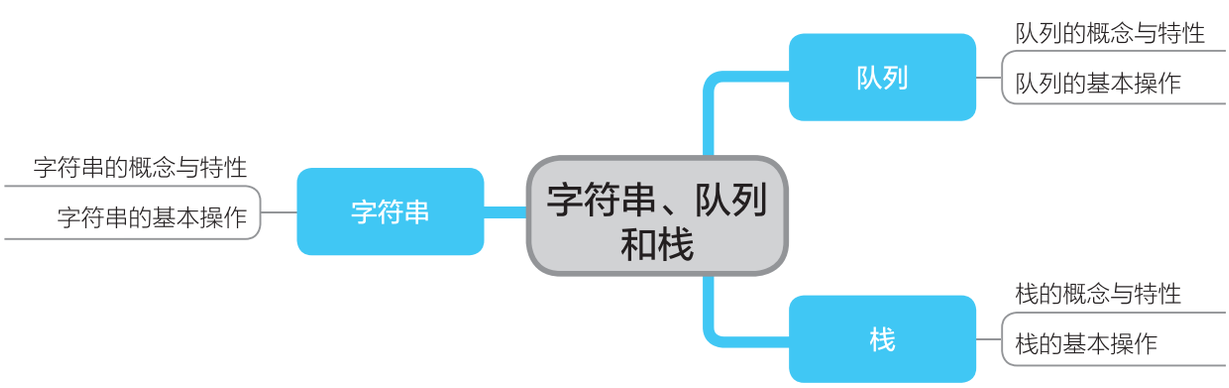
- 去银行、医院、移动服务中心等地方办理业务时，都有一个“排队叫号系统”，方便人们按序办理业务。“排队叫号系统”中的取号功能，可以显示号码、时间及等待的人数；叫号功能可以有序安排客户去相应的窗口办理业务。这种“排队叫号系统”是如何表示、组织数据的？如何实现相应的取号、叫号功能？

- 在方程求解、口算作业批改时，经常面对含有加、减、乘、除、括号等运算符号的表达式求值问题，用计算器、计算机、手机APP等工具可以快速地得出答案。在实现计算功能时，这些工具是如何表示、组织数据的？又是如何实现相应的计算功能的？

学习目标

1. 能理解字符串、队列和栈的概念及特性。
2. 能掌握字符串、队列和栈的基本操作。
3. 能运用字符串、队列和栈等数据结构来编程解决实际问题。

内容总览



3.1 字符串

使用通信软件聊天，当涉及账号、密码、财物等敏感信息时，界面会立即弹出安全提示；在网页信息的检索中，搜索引擎可以通过关键字匹配，在浩瀚的网络中找到所要的信息；在创建标签云时，可以利用单词的个数，以不同的颜色突出显示。上述事件中，可以用字符串来存储、处理相应的数据，实现对应的功能。

3.1.1 字符串的概念与特性

1. 字符串的概念

字符串是由零个或多个字符组成的有限序列。这些字符来自于某些特定的字符集，如ASCII（American Standard Code for Information Interchange，美国信息交换标准码）字符集，它包括大小写英文字母、数字、标点符号、控制字符等共128个字符。一个字符串中可以包含多种字符集字符。字符串中所包含的字符个数称为字符串的长度。为了区分字符串与其他的标识符，字符串一般用双引号或单引号加以界定，如字符串“Hello World”，它包含11个字符（不包括双引号），其中空格也是一个字符。

拓展链接

字符集

字符集是多个字符的集合，种类较多。每个字符集包含的字符个数不同。常见的字符集有ASCII字符集、GB2312字符集、GBK字符集、BIG5字符集、GB18030字符集、Unicode字符集等。

在字符串的应用中，会涉及一些特殊的字符串，如空串、空格串等。

①空串。空串是不包含任何字符的串，其长度为零。

②空格串。空格串是由空格组成的字符串，其长度为空格字符的个数。

一个字符串中任意个连续的字符序列称为该字符串的子串。例如，字符串S=“Hello World”，字符串“H”，“He”，“el”，“ell”，“World”等都是S的子串。串S也是串S的子串。除S本身外，S的其他子串称为S的真子串。

2. 字符串的特性

（1）有限序列性

字符串是一种线性表结构，它的数据元素个数是有限的，由零个或多个字符组成。除

第一个和最后一个元素外，其他元素都有一个唯一的前驱和后继，可以用元素在字符串中的相对位置作为标号来访问每个元素。在用Python实现的字符串中，有正索引标号和负索引标号，如表3.1.1所示。

表3.1.1　字符串“Hello World”的索引标号

字符	H	e	l	l	o		W	o	r	l	d
正索引	0	1	2	3	4	5	6	7	8	9	10
负索引	−11	−10	−9	−8	−7	−6	−5	−4	−3	−2	−1

正索引的第一个位置为0，用索引运算符“[]”来访问该位置上的字符。对于“Hello World”字符串，[6]指向索引位置为6的字符“W”；负索引从字符串的末端开始编号，第一个位置为“−1”，[−10]指向索引位置为−10的字符“e”。

（2）字符串的可比性

在排序、词频统计、查找文本等操作中，需要对字符串进行比较。如果字符串中的字符全来自ASCII字符集时，可以按ASCII码值进行比较，规则为从左至右，依次比较两个字符串中对应位置上字符的ASCII码值。比如字符串“word”与“work”，前3个字符一样，第4个字符“k”的ASCII码值大于字符“d”的ASCII码值，因此字符串“work”大于字符串“word”。当字符串中仅有数字字符时，采用同样的比较方式，如“9”大于“10”。对于含有非ASCII字符集的字符串，需要根据编码的方式进行比较。

问题与讨论

数组与字符串有哪些相同点和不同点？

3.1.2　字符串的基本操作

字符串的操作可以实现从字符串中提取子串、将字符串转换成小写或大写、字符串匹配等。实际应用中，基本的操作有子串判断、求子串、字符串的连接等。

1. 子串判断

子串判断用于检查字符串a是否为字符串S的子串。在Python中，可以用in运算实现，格式为“字符串a in 字符串S”。当字符串S包含字符串a时，值为True，否则值为False。Python程序如下：

程序	测试结果

S="Hello"
if("H" in S):
　　print("H 在变量S中")
else:
　　print("H 不在变量 S 中")
	H在变量S中

也可以用in运算，逐个枚举字符串中的字符，程序如下：

程序	测试结果
S="Hello" for a in S: 　　print(a,end="")	Hello

2. 求子串

求子串用于从字符串中取出其子串。在Python中，可以用切片实现，格式为“字符串[位置i:位置j]”。例如，S[i,j]表示取字符串S的索引下标i至j−1范围内的子串。

程序	测试结果
S="Hello World" print(S[6:11])	World

在上面的程序中，S[6:11]为取字符串S中第7个至第11个位置上的字符，得到子串“World”。

3. 字符串的连接

字符串的连接用于把两个字符串连接在一起。在Python中，用“+”号连接符实现，格式为“字符串S1+字符串S2”。 Python程序如下：

程序	测试结果
S1="Wor" S2="ld" S=S1+S2 print("连接后的字符串为：",S)	连接后的字符串为：World

Python中还内置了非常丰富的字符串函数与方法，可以直接调用。表3.1.2列出了一些常用的函数和方法。

表3.1.2　字符串常用函数和方法

函数和方法	功能	实例
len(x)	统计字符串x中字符的个数	x="Hello!" print(len(x)) 输出为：6
x.find(y)	返回字符串x中子串y出现的首字符下标，若找不到，则输出−1	x="Hello!" y="llo" print(x.find(y)) 输出为：2
x.split(str="", num=string.count(str))	以str分隔符切片x，如果num有指定值，那么仅分隔num个子串。切片得到的值存储在列表中	x="I like Python" list=x.split() print(list) 输出： ['I', 'like', 'Python']
x.replace(old, new[, max])	返回字符串中的old（旧字符串）替换成new（新字符串）后生成的新字符串，如果指定第三个参数max，那么替换不超过max次	x="is is is is" print(x.replace("is", "was")) print(x.replace("is", "was", 3)) 输出： was was was was was was was is
x.upper()	把字符串x中的小写字母转换成大写字母	x="I like Python" y=x.upper() print(y) 输出： I LIKE PYTHON
x.lower()	把字符串x中的大写字母转换成小写字母	x="I like Python" y=x.lower() print(y) 输出： i like python

Python中的字符串变量一旦赋值后，就不能修改它的值。

例1　回文串

从左往右读和从右往左读都一样的字符串称为回文串，比如“客上天然居，居然天上客”。输入一个字符串，写一个程序，判断该字符串是否为回文串。

根据回文串的特点，只要字符串左右对称，它就是回文串。判断字符串的左右对称时，可以先找到中间位置，采用切片操作，取出字符串的左半部分，存储到字符串变量left中；再采用切片操作，取出右半部分并反转，存储到字符串变量right中。若left与right相等，则为回文串。用Python实现的程序如下：

程序	测试结果

s=input("请输入一个字符串：")
length=len(s) #串的长度
mid=length // 2
left=s[0:mid] #取左半串
temp=s[-mid:] #取右半串
right=""
#右半串反转
for i in range(len(temp)):
　　right=right+temp[len(temp)−i−1]
if left==right:
　　print("Yes")
else:
　　print("No")
	请输入一个字符串：abcba
	输出：Yes
	请输入一个字符串：abba
	输出：Yes
	请输入一个字符串：心清可品茶,茶品可清心
	输出：Yes
	请输入一个字符串：abcab
	输出：No

也可以直接利用Python语言中的字符串反转命令[::−1]，直接反转原字符串，再与原字符串比较。程序如下：

程序	测试结果

s=input("请输入一个字符串：")
s1=s[::−1]
if s==s1:
　　print("Yes")
else:
　　print("No")
	请输入一个字符串：abcba
	输出：Yes
	请输入一个字符串：abcab
	输出：No

问题与讨论

当一个字符串中既有大小写字母，又有数字等其他ASCII字符集字符时，如何统计大写字母、小写字母以及数字字符的个数？

现实生活中，凡是遇到与字符串相关的问题，都可以用字符串来存储、组织数据，并用与字符串相关的操作来解决。

例2　机动车违章识别系统

随着机动车数量的增长，一些城市为了缓解交通压力，采取了错峰限行的措施。错峰限行，即划定区域在工作日的上下班高峰时段内，禁止挂有外地车牌和车牌尾号是某些数字的本地车牌的车辆行驶。某城市（车牌城市代码为城B）在工作日的7：00—9：00、16：30—18：30内，周一限行数字尾号1和9，周二限行数字尾号2和8，周三限行数字尾号3和7，周四限行数字尾号4和6，周五限行数字尾号5和0，外地车牌全部限行。现有某个路口一周的车辆通行信息，部分信息如表3.1.3所示。请编写程序，输出违章车辆的车牌号码和车主姓名。

表3.1.3　部分车辆通行信息表

车牌号码	车主姓名	日期	星期
城B.2D6AF	何一	3月15日	1
城B.3B09D	王立	3月15日	1
城B.2D3A2	杨子	3月17日	3
城C.36A87	陈顺	3月18日	4
城E.1F0AB	李三	3月19日	5
城B.3E6B8	邵东	3月20日	6

（1）抽象与建模

根据该城市的限行规则及表3.1.3中的数据，只需具备下列条件之一，即可判定某车违章：

①外地车牌且行驶时间在周一至周五。

②本地车牌且车牌数字尾号与限行数字尾号匹配。

（2）设计算法

用字符串存储“车牌号码”，通过索引、字符串的比较等操作，确定车辆的城市代码。如果是本地车牌，从右往左枚举车牌号码，找到第一个数字，即为该车牌的数字尾号。再根据限行规则，与“星期”数据逐一比较。具体算法如下：

①用字符串变量car存储“车牌号码”，字符串变量name存储“车主姓名”，数值变量w存储“星期”。

②在字符串car中，取前2个字符，判断是否为“城B”。若是“城B”，则为本地车牌，执行③；否则，为外地车牌，执行④。

③从右往左扫描字符串car，找到第一个数字，存入变量num中，代表车牌的数字尾号。用week存储限行的星期字符串，值为"12345"，代表周一至周五，t存储该车的数字尾号在星期几限行。根据数字尾号的限行规则，当num=0时，t=week[4]；当1≤num≤5时，t=week[num−1]；当num＞5时，则t=week[9−num]。再将t与w值进行比较，若相等，则车辆违章，输出车牌号码和车主姓名。

④若w的值为6或7，则该车辆不违章；否则，车辆违章，输出车牌号码和车主姓名。

⑤重复①、②、③、④操作，直至所有车辆处理完毕。

（3）编写程序

根据数据结构与算法，编写的程序及测试结果如下：

程序	测试结果

import csv
#数据读入
f=open("车辆号码.csv","r")
f_csv=csv.reader(f)
k=0
for row in f_csv:　　#逐一枚举所有车辆信息	

程序	测试结果

```
    k=k+1
    if k==1:                          #输出文件中的第一行信息
        print(row[0],"  ",row[1])
        continue
    car=row[0]
    name=row[1]
    w=row[3]
    week="12345"
    pos=0
    if car[0:2]=="城B":
        pos=1
    if pos==0 and w!=6 and w!=7:      #外地车牌
        print(car,"  ",name)
    if pos==1:                        #本地车牌
        i=len(car)
        while i>=0:                   #找数字尾号
            i=i-1
            ch=car[i]
            if ch>="0" and ch<="9":
                break
        num=int(ch)
        if num==0:
            t=week[4]
        elif num<=5:
            t=week[num-1]
        else:
            t=week[9-num]
        if t==w:
            print(car,"  ",name)
f.close()
```

车牌号码	车主姓名
城B.3B09D	王立
城C.36A87	陈顺
城E.1F0AB	李三

拓展链接

正则表达式

正则表达式是用一些特定的字符组成的一个“规则字符串”。它可以实现检查一个字符串中是否含有某种子串、将匹配的子串替换或者从字符串中取出符合某个条件的子串等操作。

例如，输入一串字符后，查找是否出现过12位数的电话号码（格式：xxxx-xxxxxxxx，“x”代表一个数字）。不使用正则表达式的Python程序如下：

不使用正则表达式	程序运行

def findphone(text):
  for i in range(0,len(text)):
  flag=True
  if text[i].isdecimal() and i<len(text)-12:
  for j in range(i+1,i+4):
  if not text[j].isdecimal(): #判断是否为数字
  flag=False
  if text[i+4]!="-":
  flag=False
  for j in range(i+5,i+13):
  if not text[j].isdecimal():
  flag=False
  if flag:
  return text[i:i+13]
  else:
  return False
x=input("请输入一串字符：")
result=findphone(x)
if result==False:
  print("没有找到电话号码")
else:
  print("电话号码为：",result)
	请输入一串字符：
	my phone number is:
	0575-85023251
	电话号码为：0575-85023251

使用正则表达式时，可用“\d\d\d\d-\d\d\d\d\d\d\d\d”的规则字符串匹配12位电话号码，其中“\d”表示一个0到9的数字字符 。Python中，所有的正则表达式的函数都在re模块中，使用前，需要导入re模块。

使用正则表达式	程序运行

import re
phone=re.compile(r'\d\d\d\d-\d\d\d\d\d\d\d\d')
x=input("请输入一串字符：")
mo=phone.search(x) #在x中查找第一个符合phone格式的匹配
if mo:
  print("电话号码为：",mo.group())
else:
  print("没有找到电话号码")
	请输入一串字符：
	my phone number is:0575-85023251
	电话号码为：0575-85023251

Python中的正则表达式通过字符与字符类、量词、组与捕获、断言与标记、条件匹配、标志等规则，以及re模块中的各种方法与函数，能实现文本中的模式匹配、搜索功能、替换、分割等强大的功能。

实践与体验

行程编码

行程编码（Run Length Encoding，简称RLE）是一种利用空间冗余度压缩图像的方法，它的主要技术是在数据图像的编码中寻找连续的重复数值，并用出现次数和颜色编号取代这些连续的值。对于计算机生成的图形或二值图像，行程编码能有效减少存储空间。例如，对于某二值图像的颜色编号：00000000011111111000，利用行程编码技术可以压缩为：9 0 8 1 3 0。

实践内容：

利用辅助工具“颜色编号.py”，编写“行程编码”程序，得到计算机生成的图形或二值图像的“行程编码”。

实践步骤：

1. 获取某图形或二值图像的颜色编号。

①利用“颜色编号.py”提取某图形或二值图像的颜色编号，存入“图像编码.txt”文件中。

②利用Python的读取文件功能，从“图像编码.txt”文件中获取颜色编号，用字符串存储。

2. 设计行程编码的算法。

从左往右扫描颜色编号，统计所有相同的颜色段，用颜色的个数和颜色值存储，得到行程编码，并写入“行程编码.txt”文件中。

3. 根据算法分析，编写程序。

结果呈现：

 原图.bmp	 图像编码.txt	 行程编码.txt
463KB	154KB	9.5KB

思考与练习

天干地支，简称为干支，源自中国远古时代对天象的观测。甲、乙、丙、丁、戊、己、庚、辛、壬、癸称为十天干，子、丑、寅、卯、辰、巳、午、未、申、酉、戌、亥称为十二地支。十天干与十二地支按“阳干配阳支，阴干配阴支”两两相配，从甲子到癸亥，形成了六十个基本单位，古人以此作为年的序号，每六十年为一轮，叫“干支纪年法”。从网上查找“干支纪年法”的相关资料，输入公元纪年，输出对应的天干地支。

3.2 队列

在操作计算机时，有时会遇到系统卡住的情况，即无论如何操作，系统都没有响应，而系统一旦恢复正常，则会把刚才的所有操作按顺序执行一遍。这是因为操作系统对程序的执行是按操作的先后顺序依次进行的。去银行、医院办理业务时，取号机能按照到达时间的先后顺序，合理地安排办事次序。这些事件对数据的处理都具有排队的特性，可以使用队列来解决。

3.2.1 队列的概念与特性

1. 队列的概念

队列是一种先进先出的线性表，允许插入的一端称为队尾，允许删除的一端称为队首。队列中的数据元素称为队列元素。在队列中插入一个元素称为入队，从队列中删除一个元素称为出队。如图3.2.1所示，元素a_1最先入队，是队首元素；元素a_n最后入队，是队尾元素。

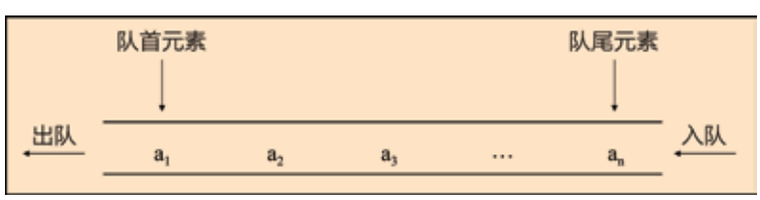


图3.2.1 队列图

2. 队列的特性

（1）先进先出、后进后出

由队列的定义可知，队列具备“先进先出、后进后出”的特点。如图3.2.1所示，出队时，队首元素a_1优先出队，紧接着是a_2，a_3，…，a_{n-1}，队尾元素a_n最后出队。

（2）有限序列性

队列也是一种线性表结构，元素个数是有限的。队列可以是空的，也可以包含多个元素。队列中所有元素呈现线性特征，队首元素只有一个后继点，队尾元素只有一个前驱点，其他元素既有一个前驱点，又有一个后继点。

3.2.2 队列的基本操作

队列一般按顺序结构存储，可以用数组来实现。如图3.2.2所示，数组que中存储了一个队列，共有4个元素，队首元素为a_1，队尾元素为a_4。由于在入队和出队的过程中，队首元素和队尾元素在数组que中的位置在改变，因此需要设置头指针变量head和尾指针变量tail，head记录队首元素所在的位置，tail记录队尾元素的下一个位置。

图3.2.2 队列的存储

对于如图3.2.2所示的队列，初始时，head指针变量与tail指针变量均记录下标为0的位置。元素a_1，a_2，a_3，a_4依次入队后，tail值为4，head值为0，如图3.2.3所示。

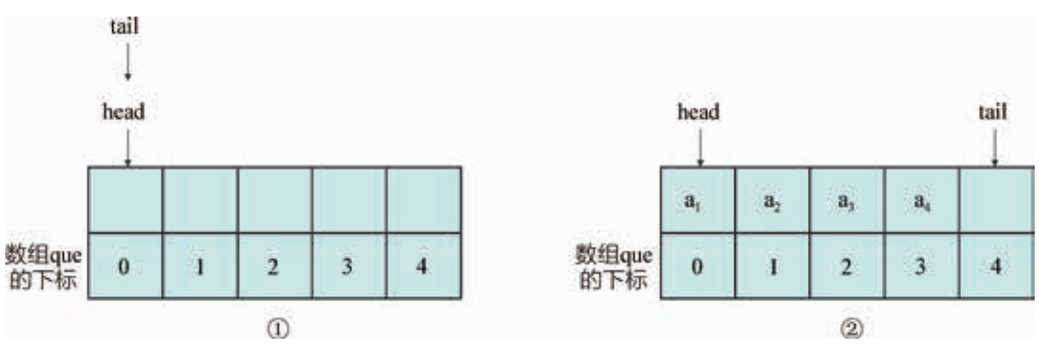


图3.2.3 队列的head、tail指针变化图

当a_1，a_2出队后，head记录下标为2的位置，tail值不变。当a_3，a_4出队后，head与tail的值均为4，队列为空。

拓展链接

队列的链式存储结构

队列的链式存储称为链队列，为了操作方便，可设置队首指针head记录链表的头节点，队尾指针tail记录链表的队尾节点，如图3.2.4所示。

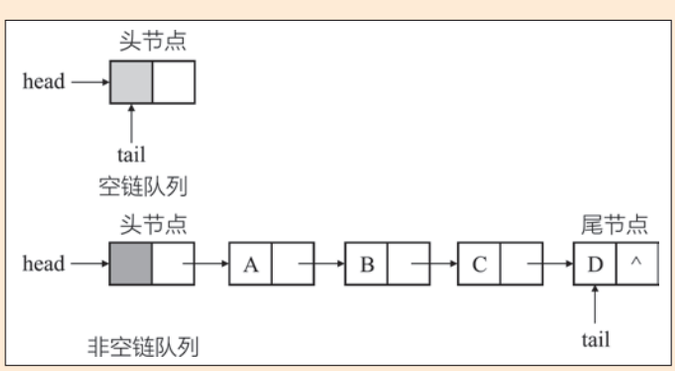


图3.2.4 队列的链式存储结构

队列的常用操作有建队、入队、出队等。

1. 建队

由于队列以数组形式存储，因此Python中用列表创建队列。例如，有4个字母“A”“B”“C”“D”按序入队、出队时，可以创建一个队列que，长度为5，Python代码如下所示：

```
head=0
tail=0
que=[""]*5
```

2. 入队、出队

字母“A”“B”“C”“D”按序入队时，在队列que中，用tail指针变量跟踪各元素的入队。如图3.2.5所示。

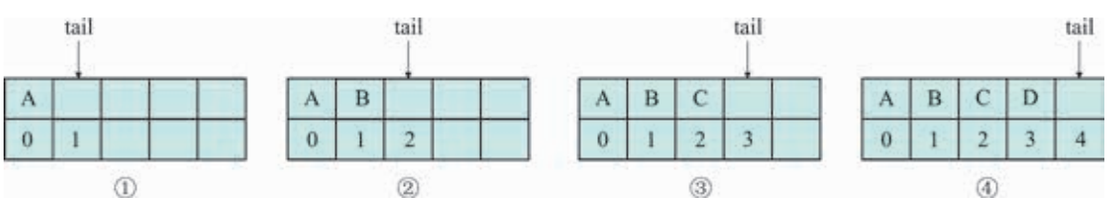


图3.2.5　四个字母“A”“B”“C”“D”入队图

入队的Python代码如下：

```
que[tail]="A"                    #字母A入队
tail=tail+1                      #tail=1
que[tail]="B"                    #字母B入队
tail=tail+1                      #tail=2
que[tail]="C"                    #字母C入队
tail=tail+1                      #tail=3
que[tail]="D"                    #字母D入队
tail=tail+1                      #tail=4
```

出队时，排在队首的元素依次出队，head指针变量依次加1，直至head值等于tail值时，队列为空。

问题与讨论

为了解决计算机与打印机之间速度不匹配的问题，通常设置一个打印数据缓冲区。计算机将所有的打印任务依次写入缓冲区，以便打印机按序打印。数据缓冲区中的数据如何存储及构造？

拓展链接

循环队列

循环队列是将队列的队首和队尾连接起来，形成逻辑上的环状结构。当对循环队列中的元素进行入队、出队操作时，队首指针变量和队尾指针变量可以循环指向所有位置，从而有效地解决队列中“有空闲位置却不能入队”的问题。

图3.2.6　tail超出队列的边界

如图3.2.6所示，某队列分配的最大空间为5，其最后一个位置上的元素为“E”，队首指针变量head的值为4，队尾指针变量tail的值为5（tail超出了队列的边界），此时，数组中存在空闲位置，但新的元素不能入队。将该队列改为循环队列，则在元素“E”入队后，head的值为4，队尾指针重新指向队首（tail的值为0），当新元素“F”入队时，就加入到队首，然后tail的值变为1，如图3.2.7所示。

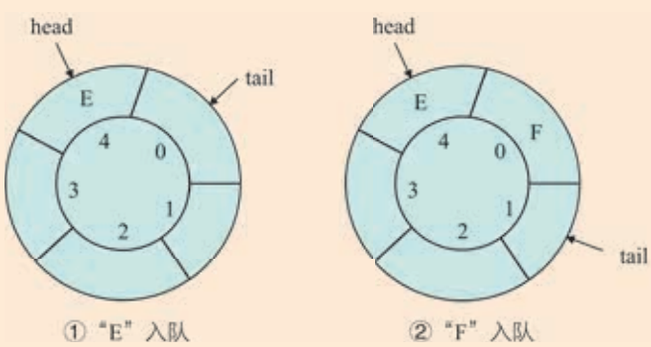


图3.2.7　循环队列

例1　信息的加密

给定一个字符串S_1，S_2，…，S_n，按如下过程加密：取出第一个字符S_1，将第二个字符S_2放到字符串的末尾S_n后面，得到字符串$S_3 \cdots S_n S_2$；接着把S_3取出，S_4放到字符串的末尾S_2后面……直到最后一个字母S_n被取出。这些字母按取出的顺序形成一个新的字符串，称为密串。请编写一个加密程序，输入一个字符串（长度小于等于100），输出该字符串的密串。

加密的过程，类似队列的入队、出队操作。先把原字符串中各字符依次入队，得到一个队列，再执行加密的过程：取出队首元素，存到密串中，队列中第二个元素升级为队首元素；再取出队首元素，并把该元素插入队尾。反复操作，直至队列为空，得到密串。以字符串“STRING”为例，算法步骤如下：

①创建que队列，把字符串“STRING”按序压入队列，tail值为6，head值为0。

②加密过程。先取出队首元素“S”，并输出，同时head值加1，记录新的队首元素“T”所在的位置。再取出队首元素“T”，并把该元素加入队尾，head值、tail值均加1。

③重复操作②，直至队列为空。

前两个字母的出队、入队过程如图3.2.8所示。

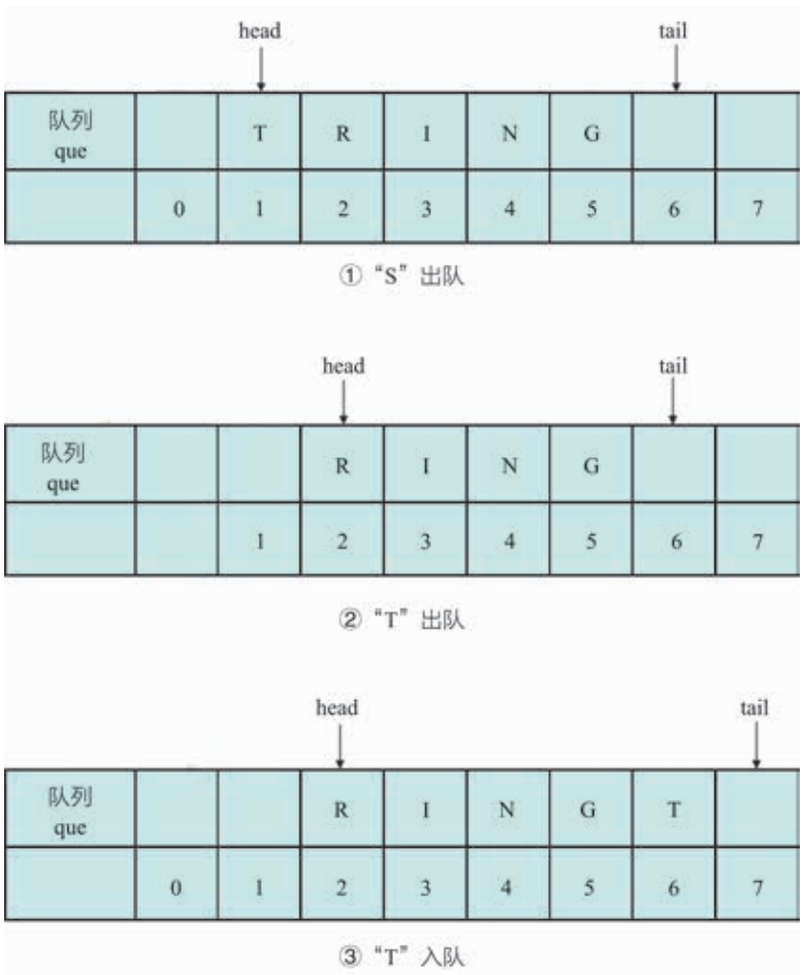


图3.2.8　前两个字母的出队、入队过程

用Python实现的完整程序及测试结果如下所示：

程序	测试结果
s=input("请输入字符串：") print("加密后的串为：") que=[""]*100 head=0 tail=0 for i in range(len(s)):　　#把原串全部压入队列 　que[tail]=s[i] 　tail+=1 while head＜tail:　　#队列非空时 　print(que[head],end="") 　head+=1 　if head＜tail:　　#队列非空时，出队的元素加入队尾 　　que[tail]=que[head] 　　tail+=1 　　head+=1	请输入字符串：Python 加密后的串为：Ptoynh

队列的元素并不局限于单个字符，也可以是其他类型的数据。只要问题中存在着“先进先出、后进后出”的特性，就可以用队列来解决。第一章的“银行叫号排队系统”可以采用队列结构，用入队（取号）、出队（叫号）实现该系统的功能。

为了简要实现“银行叫号排队系统”的功能，把取号、叫号流程集中到一个输入界面上，用队列que存储客户取的号。算法如下：

①建立队列que，队列的初始长度设置为1000，初始值均为−1。设置队首指针变量head、队尾指针变量tail的值均为0。

②设计输入提示界面，实现多次取号和叫号功能。用x存储输入的数字，如果x等于1，实现取号功能；x等于2，实现叫号功能；x等于3时，程序退出。

③当x=1时，分配一个号码，入队指针tail加1，并显示需要等待的人数。

④当x=2时，先判断que队列是否为空。若为空，则显示无等待的人员；否则，que队首元素出队，head指针加1，并显示可以办理业务的客户号码。

根据队列结构及算法设计，用Python编写的完整程序及测试结果如下所示：

程序	测试结果
que=[−1]*1000 head=0 tail=0 print("请输入具体的操作编号：") print("1.新到顾客（取号）") print("2.下一个顾客（叫号）") print("3.程序结束") x=int(input())	请输入具体的操作编号： 1. 新到顾客（取号） 2. 下一个顾客（叫号） 3. 程序结束 请输入操作，输入3结束：1 您当前的号码为：A0，需要等待的人数为0

程序	测试结果

while x!=3:
  if x==1:
  que[tail]=que[tail]+1
  print("您当前的号码为：A%d，需要等待的人数为%d"%(tail, tail–head))
  tail=tail+1
  if x==2:
  if head==tail:
  print("对不起，没有等待的客户")
  else:
  print("请A%d号客户准备，马上将为您办理业务。"%(head))
  head=head+1
  x=int(input("请输入操作\n")) # \n表示换行读入
	请输入操作编号：
	1
	您当前的号码为：A1，需要等待的人数为1
	请输入操作编号：
	2
	请A0号客户准备，马上将为您办理业务。
	请输入操作编号：
	3

拓展链接

Python 自带的队列模块

Python 中自带了队列模块，可以实现队列的建队、入队、出队等操作，代码如下：

```
import queue                #引用队列模块
q=queue.Queue(10)           #建一个长度为 10 的队列 q
q.put("A")                  #字母“A”入队
q.put("B")                  #字母“B”入队
print(q.qsize())            #输出队列中元素的个数，个数为 2
print(q.get())              #队首元素出队，出队元素为“A”
print(q.qsize())            #输出队列中元素的个数，个数为 1
```

思考与练习

1. 当需要统计队列中元素的个数时，如何统计？请编程实现。

2. 有一个最大空间为 10（存储位置标记为 0 到 9）的队列 que，经过一些入队、出队操作后，队尾指针变量 tail 的值为 10，队首指针变量 head 的值为 5。如果还有一个元素“x”要入队，由于位置 10 已超过队列的最大空间，无法入队，但是 head 的值为 5，位置 0 到位置 4 上对应的元素已经出队，这些位置可以重复使用。在不增加队列 que 空间的情况下，解决元素“x”的入队问题，并说明入队和出队时 tail 和 head 的值是如何更新的？

3.3 栈

弹匣中子弹的装弹和出弹、桶型消毒柜中餐盘的取放等都只能在某一端进行操作；网页浏览器上有一个“后退”键，点击后可以按访问顺序的逆序加载浏览过的网页。上述事件都具备栈结构的特点：后进先出。

3.3.1 栈的概念与特性

1. 栈的概念

同队列一样，栈也是一种操作受限的线性表，仅允许在表的一端进行插入或删除。进行插入或删除操作的一端称为栈顶，位于栈顶位置的元素称为栈顶元素；相应地，将表的另一端称为栈底，位于栈底位置的元素为栈底元素。如图3.3.1所示，元素“李览”为栈顶元素，元素“刘力源”为栈底元素。

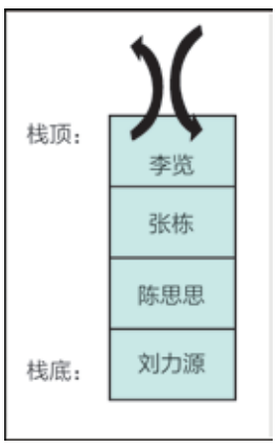


图3.3.1　栈顶、栈底

2. 栈的特性

（1）先进后出、后进先出

由栈的定义可知，栈具备“先进后出、后进先出”的特点。如图3.3.1所示，最后入栈的元素“李览”最先出栈，最先入栈的元素“刘力源”最后出栈。

（2）有限序列性

同队列一样，栈中的元素也是有限的。栈可以是空的，也可以包含多个元素。栈中元素呈现线性关系，栈顶元素有一个前驱点，栈底元素有一个后继点，其他元素既有一个前驱点，又有一个后继点。

问题与讨论

栈与队列有什么相同点和不同点？

3.3.2 栈的基本操作

栈，一般按顺序结构存储，可以用数组实现。由于栈顶元素在数组中的位置会发生改

变，因此使用top变量来记录栈顶元素在数组中的位置。如图3.3.2所示，①图为栈结构，②图为用数组st存储该栈。当top=0时，st[top]存储栈底元素“A”；top=1时，st[top]存储栈中第2个元素“B”；top=2时，st[top]存储栈顶元素“C”。

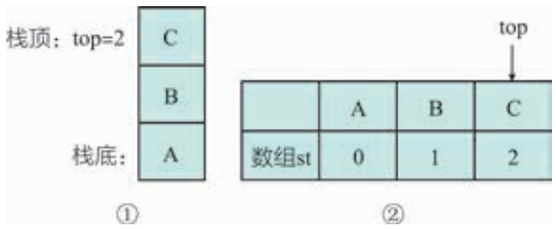


图3.3.2　栈的存储

栈的常用操作有建栈、入栈、出栈等。

1. 建栈

在Python中，当要存储n个元素的栈时，可以用列表创建一个长度为n的栈。例如，要使4个字母“A”“B”“C”“D”按序入栈、出栈，可以建一个长度为4的栈st，元素初始值均为空串。为了操作方便，把指向栈顶元素的指针变量top值设置为–1。Python代码实现如下：

```
top=-1
st=[""]*4
```

拓展链接

栈的链式存储结构

利用链式存储方式实现的栈称为链栈。它可以用单链表的方式实现。如图3.3.3所示，栈顶指针top为链栈的头指针。链栈的优点在于它克服了用数组实现的顺序栈空间利用率不高的缺点，但是要为每个栈元素分配额外的指针空间。

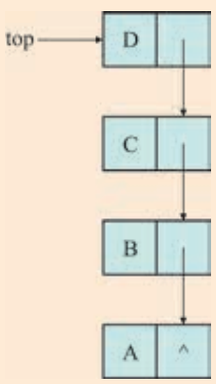


图3.3.3　栈的链式存储结构

2. 入栈、出栈

入栈又叫压栈操作，把数据元素压入栈顶。每次入栈时，栈顶指针变量top值加1，再给st[top]赋值。字母“A”“B”“C”“D”按序入栈的过程如图3.3.4所示。

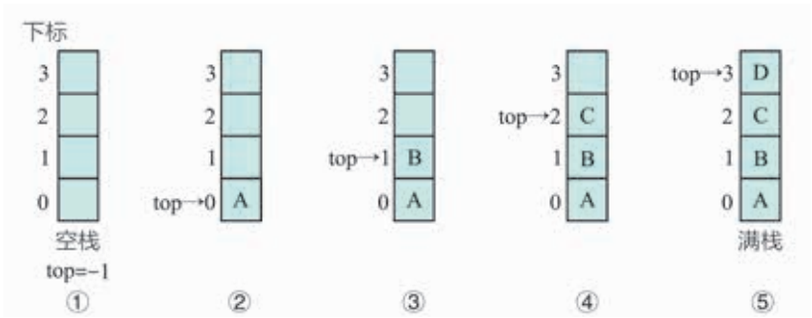


图3.3.4　st栈的入栈过程

Python代码实现如下：

```
top=top+1                    #top=0
st[top]="A"                  #字母A入栈
top=top+1                    #top=1
st[top]="B"                  #字母B入栈
top=top+1                    #top=2
st[top]="C"                  #字母C入栈
top=top+1                    #top=3
st[top]="D"                  #字母D入栈
```

出栈时把栈顶元素取出，同时top值减1。如果栈中没有元素时，即top=−1，不能进行出栈操作。

问题与讨论

编号为1、2、3、4的4列火车，按顺序开进一个栈式结构的站点。问：开出火车站的顺序有多少种？请写出所有可能的出栈序列。

对于第一章项目挑战中的“用户角色特征值”转换成二进制，可以采用“除二取余法”，并利用栈结构存储每次转换过程中得到的余数。以特征值6为例，转成二进制的过程如图3.3.5所示。

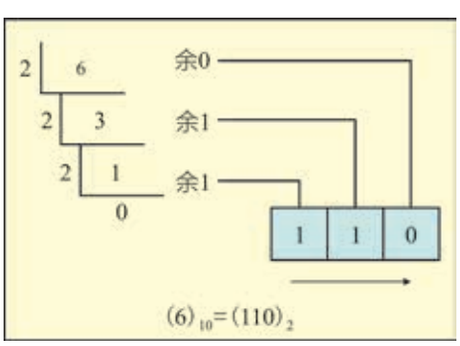


图3.3.5　十进制转二进制的过程

除二取余数的过程中，特征值的变化为6→3→1→0，每一步得到的余数为0→1→1，按序入栈，当特征值变为0时，依次取出栈中元素，得到对应的二进制数“110”。用栈st存储每次得到的余数，number存储特征值，top存储栈顶位置。除二取余数的入栈过程如图3.3.6所示，出栈过程如图3.3.7所示。

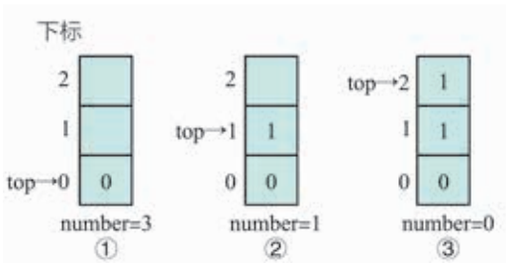


图3.3.6　入栈过程

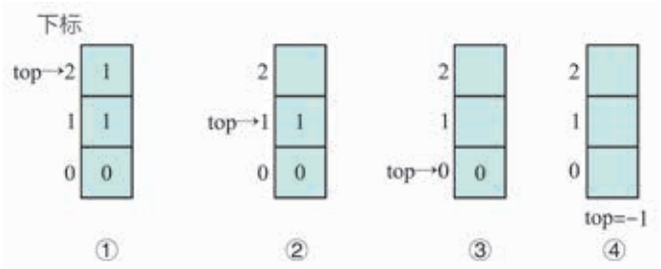


图3.3.7　出栈过程

根据入栈、出栈的步骤，采用Python编写的完整程序及测试结果如下所示：

程序		测试结果

st=[-1]*100
top=-1
number=int(input("请输入十进制整数："))
while number>0:
　　x=number % 2
　　top=top+1
　　st[top]=x
　　number=number // 2
while top>=0:
　　print(st[top],end="")
　　top=top-1
	#列表中元素初始值-1
	
	#入栈
	
	#出栈
		请输入十进制整数：
		13
		输出：
		1101

现实生活中，只要问题中存在着后进先出的特点，就可以用栈来存储数据，以及用栈的相关操作来解决。

例1　括号匹配

在一个数学计算式“(a÷(b-c)+d)×e”中，位置1和位置4有左括号“(”，位置8和位置11有右括号“)”。位置1的左括号与位置11的右括号相匹配，位置4的左括号与位置8的右括号相匹配。而对于数学计算式“a÷(b-c))”，位置8的右括号没有可匹配的左括号。设计一个程序，判断输入的数学计算式中的括号（只有小括号）是否匹配。

（1）抽象与建模

数学计算式中既有数字，又有加减乘除等运算符号，判断括号是否匹配时，可以忽略这些括号以外的数字和运算符号。对于数学计算式：(a÷(b-c)+d)×e，括号的序列如图3.3.8所示：

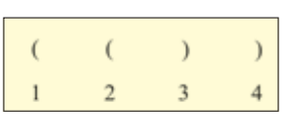


图3.3.8　括号序列

若左右括号的数量相等并且位置匹配，则括号匹配；否则，括号不匹配。判断左右括

号的数量与位置时，可以采用栈结构进行设计：遇到左括号时，入栈；遇到右括号时，则把处于栈顶的左括号出栈。分以下三种情况，判断数学计算式中的括号是否匹配：

①栈空，出现右括号时，不匹配。

②扫描结束，栈中还有左括号时，不匹配。

③扫描结束，栈空，则匹配。

（2）设计算法

设置一个栈st和栈顶指针top，从左往右逐步处理数学计算式。若是左括号，栈顶指针top值加1，并将其压入栈中。若是右括号：如果top大于−1，那么栈中有元素，把栈顶的左括号弹出，top值减1，表示该右括号与弹出的左括号相匹配；如果top等于−1，栈为空，表示没有与该右括号相匹配的左括号，是不匹配的数学计算式。如果数学计算式处理完毕，栈中还有左括号，那么它也是不匹配的数学计算式。数学计算式“(a ÷ (b–c)+d) × e”中括号的入栈及出栈过程如图3.3.9所示。

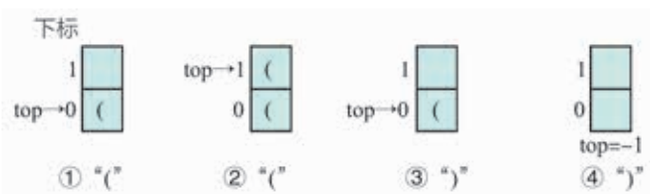


图3.3.9　数学计算式“(a ÷ (b–c)+d) × e”的入栈及出栈过程

（3）编写程序

根据数据结构与算法，用Python编写的完整程序及测试结果如下所示：

程序	测试结果

st=[""]*100
top=−1
flag=True　　#标记是否有不匹配的情况
s=input("请输入数学计算式：")
for i in range(len(s)):
　　if s[i]=="(":
　　　　top=top+1
　　　　st[top]=s[i]
　　elif s[i]== ")":
　　　　if top==−1:
　　　　　　flag=False
　　　　　　break
　　　　else:
　　　　　　top=top−1
if top>=0:　　#栈中还有左括号
　　flag=False
if flag:
　　print("括号匹配")
else:
　　print("括号不匹配")
	请输入数学计算式：
	(((a+b)*(c−d)−e)/f)
	输出：
	括号匹配
	
	请输入数学计算式：
	((a+b)*c)−d)+(e
	输出：
	括号不匹配

拓展链接

用列表自带的函数和方法实现的栈

Python中用列表自带的函数和方法可以实现建栈、入栈、出栈、栈中元素个数的统计等操作。例如：

```
stacklist=[]                #建立一个空栈list
stacklist.append("A")       #字母A入栈
stacklist.append("B")       #字母B入栈
print(stacklist[1])         #输出栈顶元素，为字母B
print(len(stacklist))       #输出栈中元素的个数，为2
stacklist.pop()             #弹出栈顶元素
print(len(stacklist))       #输出栈中元素的个数，为1，是字母A
```

实践与体验

逆波兰表达式

在数学运算表达式中，运算符总是置于与之相关的两个运算对象之间，在计算结果时，要考虑括号、运算符号的优先性。为了程序实现的方便，波兰逻辑学家J.Lukasiewicz提出了另一种表示法，将运算符置于其运算对象之后，没有括号，不用考虑运算符号的优先性。这种表达式称为后缀表达式，又叫逆波兰表达式，如表达式“6 8 2 – 2 * 3 ÷ +”是“6+（8–2）*2 ÷ 3”的逆波兰表达式。

实践内容：

输入逆波兰表达式，写一个程序，输出该逆波兰表达式的值。

实践步骤：

1. 分析逆波兰表达式的计算方法。根据题意中的计算过程，用一个栈存储数字。从左往右扫描该表达式，遇到数字时，入栈；遇到运算符号时，把处于栈最上方的两个元素依次出栈，用运算符计算，并把计算结果压入栈中。如此反复操作，直至表达式扫描结束。

2. 根据算法分析，设计程序。

结果呈现：

输入	输出
6 8 2 – 2 * 3 ÷ +	10

思考与练习

1. 如何区分取栈顶元素与出栈操作？用程序如何实现？

2. 元素 a, b, c, d, e 按序入栈，在所有的出栈序列中，以元素 d 开头的出栈序列是哪些？

巩固与提高

1. 输入一个仅包含ASCII字符集的字符串s，删除其中k（k小于字符串s的长度）个字符，使得剩余的字符在保持相对位置不变的情况下，构成一个最小的字符串。例如字符串“a3b3e4c”在删除3个字符后，得到最小的字符串为“334c”。

2. 在单一的密码表的基础上，维吉尼亚密码引入了一个26×26的英文字母方阵密表（上网查阅相关资料），结合密钥，对信息加密。比如明文为“THEBOOKANDTHEPENCIL”，密钥为“BIG”，加密时，以明文为行，密钥为列，行和列交叉得到密文。当密钥长度小于明文长度时，密钥循环使用。请编写一个程序，输入明文（仅有字母）和密钥，输出“维吉尼亚”密文。

3. 某些生意好的餐馆会设置取号、叫号，并按“二人桌”“四人桌”“六人桌”“十人桌”等分类取号。在不考虑界面设计的情况下，设计一个餐馆取号、叫号的程序。

4. 为四则运算式“6+（8−2）*2÷3”转后缀表达式“6 8 2 −2*3 ÷+”设计算法，并用程序实现。

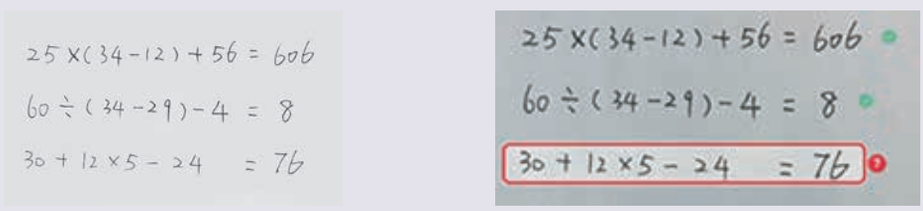
5. 在火车调度入口共有n节硬席或软席车厢（分别以H和S表示）等待调度，现需把所有的软席车厢调度到硬席车厢之前。请编写一个程序，输出调度时的操作序列。

6. 参照十进制转二进制的方法，编写一个将十进制数N转换为r进制数的程序。

项目挑战

口算批改

为了帮助小学生检查口算作业，你所在的软件开发团队计划开发一款“口算批改”APP，以提高口算批改的效率。该APP的基本设想是：将手机对着图3.3.10中的①图拍照，利用OCR（光学字符识别）技术，识别出算式，然后逐一计算，判断答案是否正确，结果如②图所示。



①批改前的算式　　②批改后的算式

图3.3.10　算式示例

项目任务

根据团队的分工，你的任务是：编写程序，根据识别出来的算式（如图3.3.11所示），逐一计算，检验算式的正确性，并添加相应标识，如“√”“?”等。

```
25*(34-12)+56=606
60/(34-29)-4=8
30+12*5-24=76
```

图3.3.11　识别出的算式

过程与建议

1. 准备数据

要完成该项目，需要准备数据。可以准备一个文档，用以保存识别出来的算式。文档格式要求：每个算式占一行，只包括加、减、乘、除、括号等运算符号，且每个算式均有计算结果。

2. 抽象与建模

请思考以下问题，将完成项目任务的问题解决模型通过图示的方式描述出来。

（1）如何提取出计算式中的数字和运算符号?

（2）如何体现加、减、乘、除、括号等运算符号的优先级?

（3）结合运算符号的优先级，程序中如何设计数字与运算符号的计算？

3. 设计算法

根据“抽象与建模”中形成的问题解决模型，设计算法。

4. 编写程序与测试

根据上面设计的数据结构与算法，用Python实现相应的程序，并利用不同的数据反复测试验证。

5. 展示交流

将编程的思路、程序代码及验证效果向其他同学展示。相互交流，取长补短。

评价标准

请根据项目实施的过程、效果以及成果展示交流的结果，对自己完成项目的情况进行客观的评价，并思考后续完善的方向。把评价结果和完善方案填写在下面的表格中。

评价条目	说明	评分（1～10分）	评分主要依据阐述	后续完善方向
问题解决模型	图示清晰，基于该模型，能够正确地提取数字，呈现运算符号的优先性，并能实现正确计算			
算法设计	根据问题解决模型，能够设计正确的算法			
程序实现	程序语言运用熟练，能够高效率地实现算法			
展示交流与编程效果	编程思路表述清晰、代码简洁规范，程序运行成功			

拓展项目

1. 网络上有各种即时通信软件，方便人们进行便捷的交流。然而，聊天信息中的诈骗信息也让人们深受其害。为了提醒用户免受欺骗，很多软件都设置了安全提示功能，当聊天中出现账号、密码、财物等敏感信息时，会立即弹出安全提示，提醒大家注意。请编写程序，模拟简单的安全提示功能。

2. 某些APP软件提供订票服务，其功能为：按序登记客户的相关信息（客户姓名和所订的票数等），若有票，则按客户登记的顺序出票，输出相应的座位号；若无票，则输出“等待”。根据订票流程，组织客户相关的信息，为某趟列车设计订票系统。

3. 电路板布线问题。电路板的水平直线上，从左向右分布着n个针脚（1，2，3，…，n），用于连接导线。连线（p,q）表示针脚p和q之间通过一根导线相连，导线只允许从水平直线的下方相连，如图3.3.12所示。对于给定的一组连线（p_1,q_1），（p_2,q_2），…，（p_n,q_n），如果能适当安排这组连线，使它们互不相交，那么称这组连线是可布线的。当出现如图3.3.12所示的针脚相连时，则称它们是不可布线的。对于给定的n个针脚和k条连线，试设计一个算法判定这组连线是否可行。

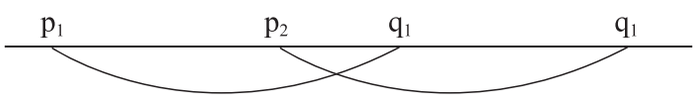
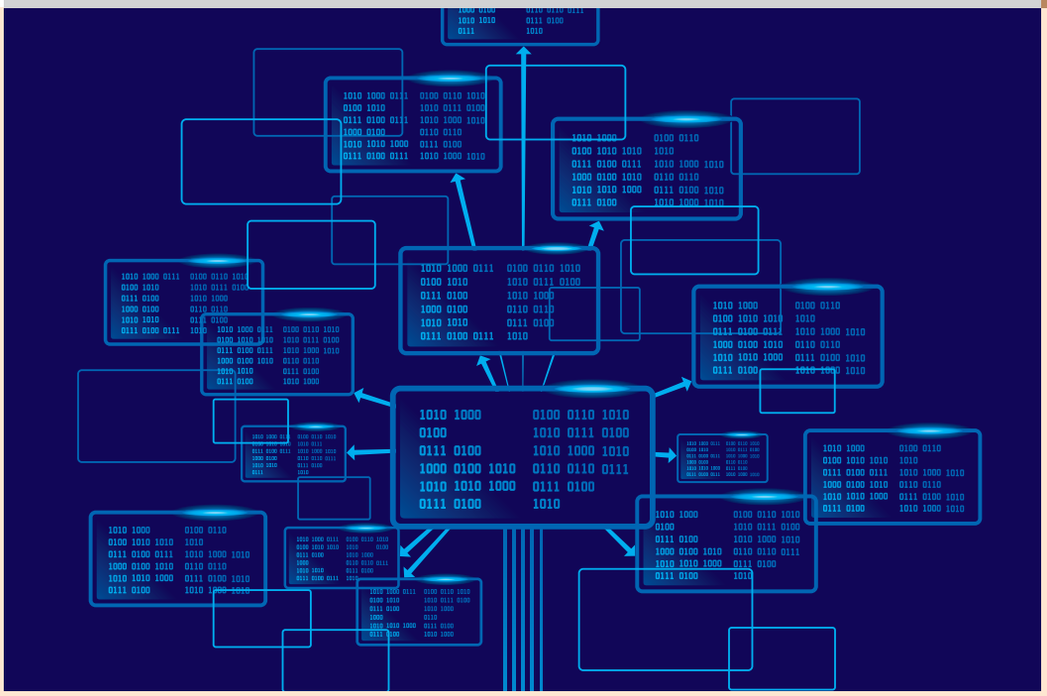


图3.3.12　针脚（p_1,q_1）和针脚（p_2,q_2）

第四章 4

树



在计算机世界中，围绕某个任务可能存在多个数据元素，这些数据元素之间会形成一定的数据结构。如果这些数据元素构成了一个有序数据元素的集合，那么称为线性结构；如果这些数据元素中的一个节点元素有多个直接前趋或多个直接后继，那么称为非线性结构。常见的非线性数据结构有树、图等。

非线性数据结构在实际数据组织管理中有着广泛的应用，比如在制作网页时，为了实现页面间的任意跳转，设计者可将素材按内部联系划分成不同关系的单元，再用制作工具将其组合成一个网状结构，使得用户在浏览时不必按照线性的方式从上往下查看，而是有选择地阅读自己感兴趣的内容。人工智能中的人机对弈可以引入树这种数据结构来建立模型，以便对后续局面的状态数据进行组织、保存和处理。多岔路口交通信号灯的管理问题则可引入图这种数据结构来解决。

问题与挑战

- 计算机系统中的文件管理、社会各机构部门的组织管理、图书馆藏书管理等领域对数据都有明确的分类要求，分析这些数据元素之间的逻辑关系，采用怎样的数据结构来组织数据，才能更好地确保数据处理的正确性，从而提高数据处理、工作管理的效率？

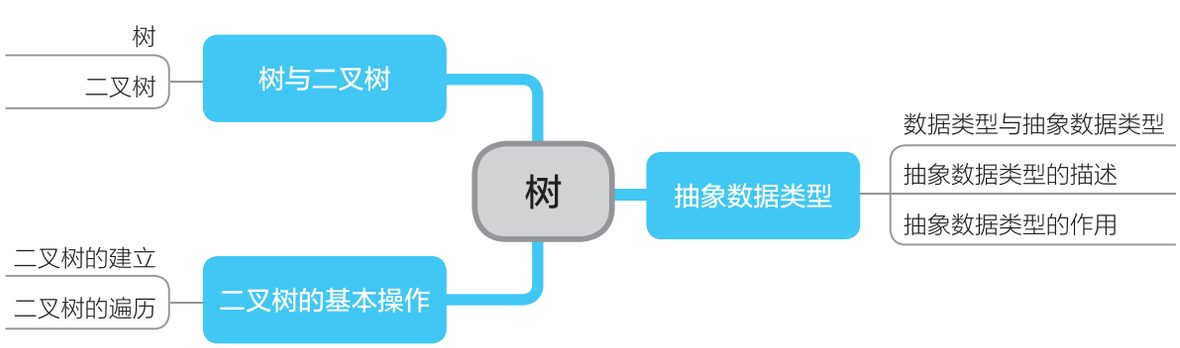
- 现代物流仓储配送建立了层级分明的物流网络，每个中转物流点都有其下属的物流点，每个物流点都有一定数量的快递员或无人机。如何根据某区域范围内的社区及人口数量，合理设置物流点及快递员数量，使得服务运行成本更低、物流配送效率更高？

- 导航软件给人们的日常生活带来了极大的便利，事先在软件中设置了起点和终点，导航软件就会给出合适的行程路径。然而，城市中的公路、地铁等交通路线组成了非常复杂的网状结构，这种结构不是线性结构所能高效描述的，那么如何描述这些具有复杂结构的数据呢？

学习目标

1. 了解二叉树的概念及其基本操作方法。
2. 理解抽象数据类型的概念。
3. 认识抽象数据类型对数据处理的重要性。

内容总览



4.1 树与二叉树

线性数据结构主要用于描述有序数据元素的问题，而像家族谱系等数据需要一种称为树的数据结构进行描述。树是一种非线性的数据结构，用它能很好地描述有分支和层次特性的数据集合。

4.1.1 树

树形结构在现实世界中广泛存在，如图4.1.1所示的公司内部管理的组织关系图就可以用树形结构来表示。树在计算机领域中也有广泛应用，如图4.1.2所示，在编译系统中，用树表示源程序的语法结构。如图4.1.3所示，在数据库系统中，树形结构是数据库层次模型的基础，也是各种索引和目录的主要组织形式。

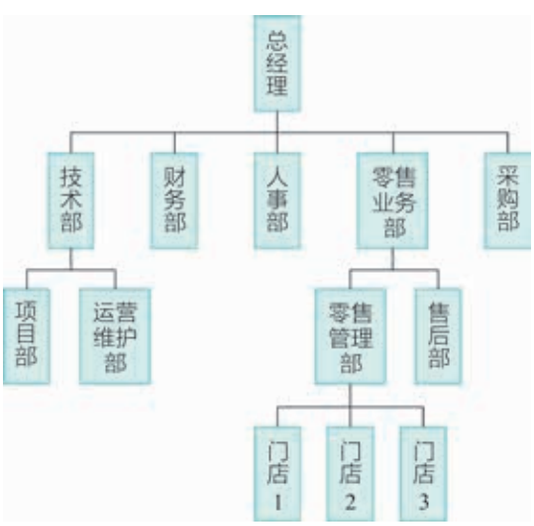


图4.1.1　公司内部管理的组织关系图

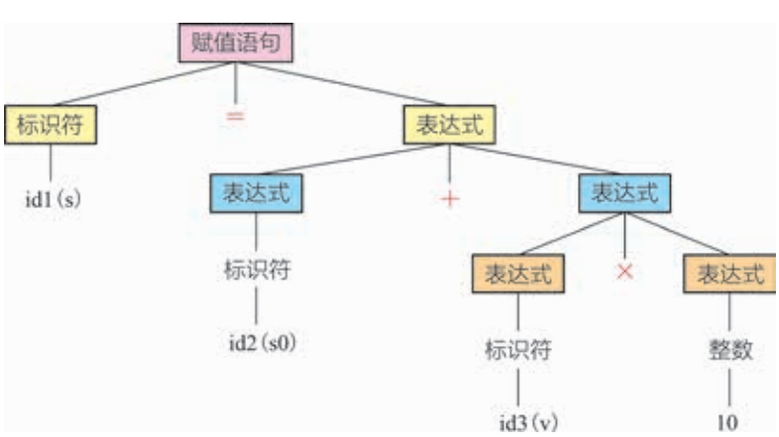


图4.1.2　赋值语句的语法树

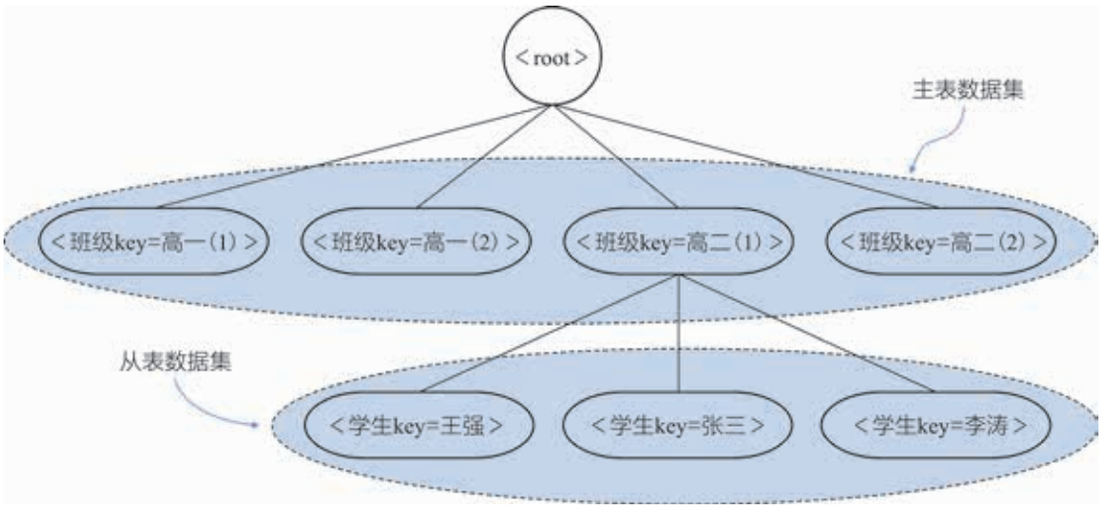


图4.1.3　数据库层次结构图

树（Tree）可以描述为由n（n≥0）个节点（Node）构成的一个有限集合以及在该集合上定义的一种节点关系。集合中的元素称为树的节点，n=0的树称为空树；树中某个节点下面的所有节点所构成的树称为该节点的子树。树的两个节点之间如果有一条边连接，那么称这两个节点之间存在一条边；对于一棵具有n个节点的树，它有n–1条边。如图4.1.4所示的树是根据图4.1.1所示的公司内部管理的组织关系图抽象得到的，树中共有13个节点、12条边，节点B、G、H构成了节点A的一棵子树。

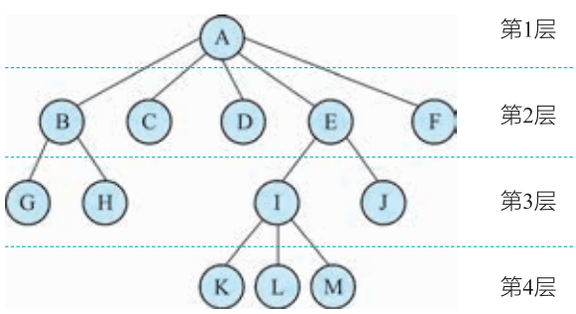


图4.1.4　树的示例

树的一个节点所拥有的子树个数称为该节点的度（Degree），最大的节点的度称为树的度。如图4.1.4所示，节点A的度为5，节点B的度为2，节点I的度为3，因此树的度为5。节点的度和树的度共同体现了树的宽度，也就是体现了节点的分支数和树的发散程度。从度的定义很容易得到，前三章中学习的线性表其实是一种特殊的树状结构，它的度为1。

在树形结构中，没有前驱的节点称为根节点（Root），又称为开始节点。度为0的节点称为叶子节点（Leaf），它又称为终端节点。树中度不为0的节点称为分支节点或者称为非终端节点，除根节点外的分支节点统称为内部节点。如图4.1.4所示，节点A是根节点，节点G、H、C、D、K、L、M、J、F都是叶子节点。在树形结构中，对于两个以边直接连接的节点，上端节点称为下端节点的父节点或双亲节点（Parent）。相应地，下端节点称为上端节点的孩子节点（Child）。如图4.1.4所示，节点K、L、M都是节点I的孩子节点。反过来，

节点B是节点G、H的父节点，节点G、H是兄弟节点。

树中节点的层数（Level）从根开始计算，根的层数为1，其余节点的层数等于其父节点的层数加1。树中节点的最大层数称为树的高度或深度（Depth）。如图4.1.4所示的树的高度为4，节点G、H、I、J都在第3层。

结合线性结构的特点，树形结构与线性结构在结构的连接关系上有如下差异：线性结构中必存在着唯一的一个“第一个元素”和唯一的一个“最后的元素”；除第一个元素以外，其他数据元素均有唯一的“前驱”，除最后元素以外，其他数据元素均有唯一的“后继”。而树形结构是比较复杂的非线性结构，在有多个节点（节点个数＞1）的树形结构中，它只有一个没有前驱、只有后继的根节点，可以有多个没有后继、只有前驱的叶子节点，其余的节点都只有一个直接前驱和多个直接后继。

拓展链接

图（Graph）

图状结构是一种比线性结构和树形结构更为复杂的非线性结构，广泛应用于计算机网络、通信工程等诸多领域。在线性表中，一个数据元素只和它的前驱和后继元素有关系。在树中，一个节点只和它的父节点和孩子节点有关系。而在图中，每个顶点都有可能和其他任意顶点有关系，这就使得图的存储和运算比前两种数据结构更加复杂。

图中节点的关系既可以是单向的，也可以是双向的，有无向图和有向图之分，又有连通图和非连通图之别，如图4.1.5所示。

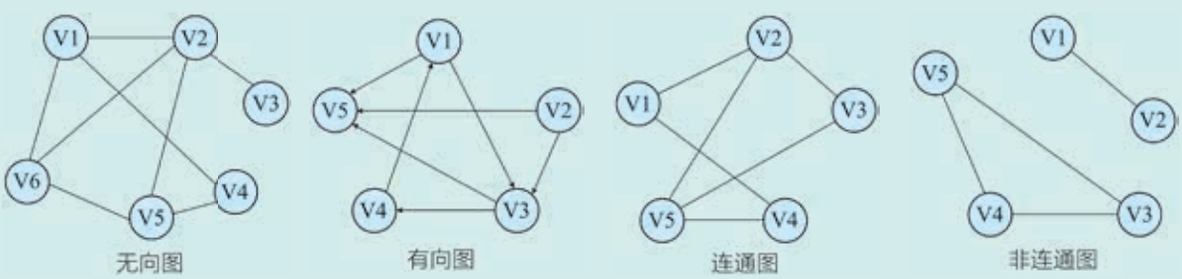


图4.1.5　节点关系图

问题与讨论

1. 如何管理计算机中的照片，使得浏览起来更加方便？

2. 若干个家庭一起组织自助游，准备阶段需要考虑旅游路线的规划、食宿安排以及旅途中各项娱乐活动的人员组队等问题。如何用学过的数据结构知识来更好地帮助制订相关计划？

4.1.2 二叉树

树的形态有很多，在实际的使用过程中，需要对树的形态进一步地进行约束和简化，以便于设计和操作。二叉树是树形结构的一个重要类型，在实际应用中，许多问题抽象出来的数据结构往往就是二叉树的形式。

例如，在猜数字游戏中，甲方事先在纸上写出一个100以内的正整数，让乙方猜，乙方每猜一次，甲方都会告诉乙方“猜大了”或是“猜小了”，直至猜出正确结果。乙方如果采取“折半”的策略进行猜数字，就一定能够在7次以内猜对结果，具体过程可抽象成如图4.1.6所示的二叉树结构（每个节点为乙方所猜的数字，每条边为实际数字与所猜数字之间的大小关系，图中仅呈现前4次的猜数字情况）。

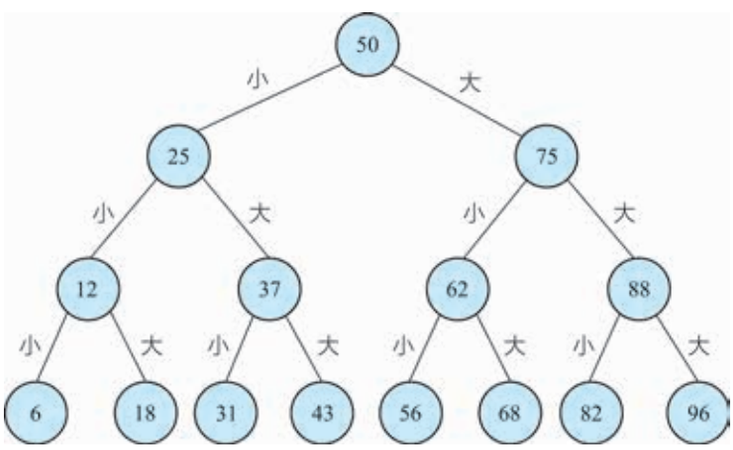


图4.1.6　猜数字过程的抽象形态

1. 二叉树的概念

二叉树（Binary Tree）是一个具有n（n≥0）个节点的有限集合。当n=0时，二叉树是一棵空树；当n≠0时，则是一棵由根节点和两棵互不相交的、分别称作这个根节点的左子树和右子树组成的二叉树，由于左、右子树也是二叉树，因此子树也可以是空树。如图4.1.7所示是一棵二叉树，二叉树具有一个重要的特征，就是它的所有节点的度都小于或者等于2，这给二叉树的操作带来了很大的方便。

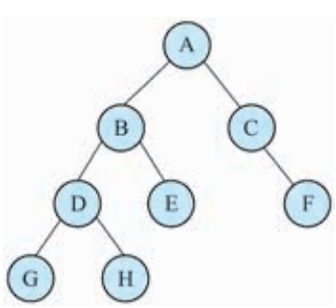


图4.1.7　二叉树的示例

一棵二叉树有5种可能的形态，如图4.1.8所示，从左到右分别是：①空二叉树；②只有根节点的单点树；③只有根节点和左子树；④只有根节点和右子树；⑤左右子树均非空。

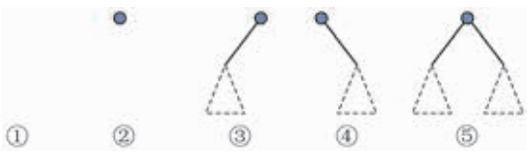


图4.1.8　五种不同形态的二叉树

在实际应用中，有一类特殊的二叉树：完全二叉树。这一类二叉树至多只有最下面两层中的节点度数小于2，且最下面一层的叶子节点都依次排列在该层的最左边位置。如图4.1.9所示为一棵完全二叉树。

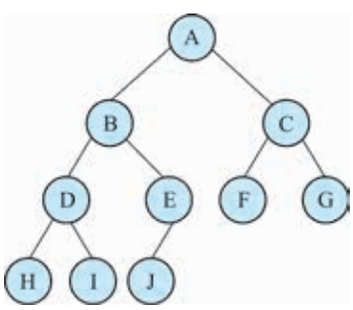


图4.1.9　完全二叉树

2. 二叉树的性质

二叉树有很多性质，作为重要的数据结构，二叉树最重要的性质就是树的高度和树中可以容纳的最大节点个数之间的关系，主要有：

①二叉树的第k层上最多有2^{k-1}（k≥1）个节点。

当k=1时，只有1（2^0=1）个根节点；当k=2时，由于节点的度最大为2，因此，第2层的节点数最多有2（2^1=2）个节点。依次推出，第k层上最多有2^{k-1}个节点。

②深度为k的二叉树最多有2^k–1（k≥1）个节点。

第1层至第k层上的最大节点数相加的结果是：$2^0+2^1+\cdots+2^{k-1}=2^k-1$。因此$2^k$–1是深度为k的二叉树的最多节点总数。

③在任意一棵二叉树中，若度为2的节点数量为n_2，叶子节点（度为0的节点）数为n_0，则$n_0=n_2+1$。

假设度为0，1和2的节点数分别是n_0，n_1和n_2，则二叉树中总的节点数$n=n_0+n_1+n_2$。在二叉树的所有节点中，除了根节点没有前驱外，每个节点均有且只有一个前驱节点，因此有n个节点的二叉树的总边数为n–1条。根据度的定义，二叉树的总边数与度之间的关系：$n-1=0\times n_0+1\times n_1+2\times n_2$，结合上述两个等式，可以得出$n_0=n_2+1$。

例如，观察如图4.1.10所示的甲、乙两棵二叉树：在甲树上，度为2的节点数是1，度为1的节点数是1，叶子节点数为2；在乙树上，度为2的节点数是2，度为1的节点数是1，叶子节点数为3。

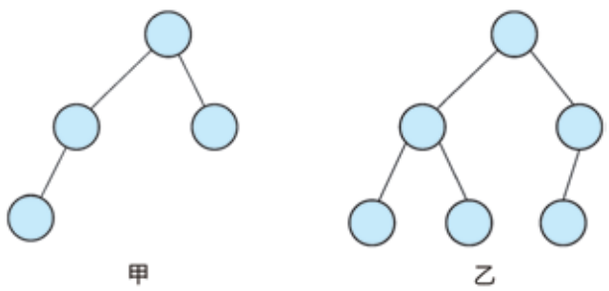


图4.1.10　二叉树

二叉树在实际应用中非常广泛，如基于二叉树的查找方法（二叉排序树查找），具有较高的查找效率，还能够在查找表中进行数据插入和删除等动态查找操作。最优二叉树（又称哈夫曼树，Huffman Tree）广泛应用于编码和决策等方面。

拓展链接

哈夫曼树

哈夫曼树又称最优二叉树，它的相关概念如下。

路径：树中两个节点之间所经过的分支，称为它们之间的路径。

路径长度：一条路径上的分支数，称为该路径的长度。

节点的权：给二叉树中的节点赋一个数，该数称为该节点的权。

节点带权路径长度：从根节点到一个节点的路径长度与该节点的权值的乘积，称为该节点的带权路径长度。

树的带权路径长度：一棵树中所有叶子节点的带权路径长度之和，称为该树的带权路径长度WPL（Weighted Path Length of Tree），WPL的公式如下：

$$\sum_{i=1}^{n} W_i \times P_i$$

其中，n为树中叶子节点的个数，W_i和P_i分别是第i个叶子节点的权值和从根节点到该路径的长度。

最优二叉树：在具有n个带权叶子节点的所有二叉树中，称带权路径长度WPL最小的二叉树为最优二叉树。在最优二叉树中，权值较大的节点离根较近。

如图4.1.11所示的三棵二叉树，叶子节点的权值都分别为2，4，5，8。

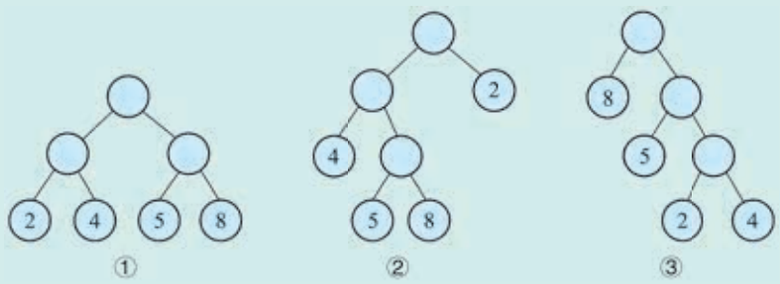


图4.1.11　二叉树

三棵二叉树的WPL分别为：

①WPL=2×2+4×2+5×2+8×2=38

②WPL=5×3+8×3+4×2+2×1=49

③WPL=2×3+4×3+5×2+8×1=36

由此可见，第三棵二叉树是最优二叉树。

问题与讨论

从数据的组织处理效率来看，二叉树本质上是对数组和链表的一种折中处理，如何看待这种说法？

思考与练习

1. 试着用树结构来描述自己家族成员的组成情况。
2. 请画出包含4个节点的所有形态的二叉树。
3. 已知某完全二叉树有200个节点，求出该二叉树的高度。
4. 假设某二叉树包含的节点数据分别为：1，5，8，3，10。请完成下列任务：

（1）画出两棵高度最大的二叉树。

（2）画出两棵完全二叉树，要求每个双亲节点的值大于其孩子节点的值。

4.2 二叉树的基本操作

无论是线性结构还是非线性结构数据，都需要对数据元素逐个进行组织存储和处理。二叉树的基本操作，主要包括二叉树的建立和遍历等。

4.2.1 二叉树的建立

建立二叉树的操作，可以按照层的顺序进行，先由第1层开始，依次到下一层，在每一层中按照从左到右的顺序创建节点。二叉树的建立可以用数组或者链表数据结构来实现。

1. 数组实现

用数组来表示二叉树时，分为以下两种情况。

（1）完全二叉树

从二叉树的根节点开始，按从上而下、自左往右的顺序对n个节点进行编号，根节点的编号为0，最后一个节点的编号为n–1。然后依次将二叉树的节点用一组连续的数组元素来表示，节点编号与数组的下标一一对应。如图4.2.1甲所示的完全二叉树所对应的一维数组表示如图4.2.1乙所示。

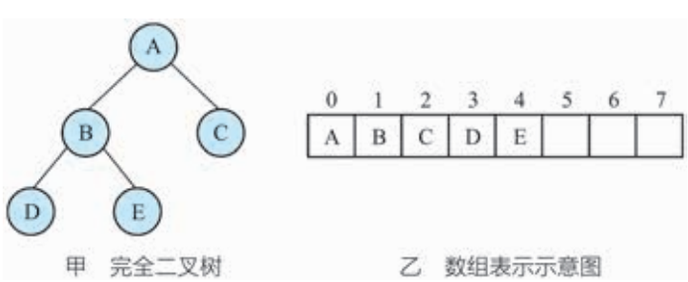


图4.2.1 完全二叉树的数组表示

（2）非完全二叉树

对于非完全二叉树，先将它补全为一棵完全二叉树，补上的节点及分支用虚线表示，如图4.2.2甲所示的一棵非完全二叉树补全为图4.2.2乙所示的完全二叉树。然后将补全后的完全二叉树，从它的根节点开始，按从上而下、自左往右的顺序对n个节点进行编号，根节点的编号为0，最后一个节点的编号为n–1。如图4.2.2丙所示，依次把完全二叉树中原二叉树的节点用一维数组的各个元素来表示，节点编号与数组的下标一一对应。

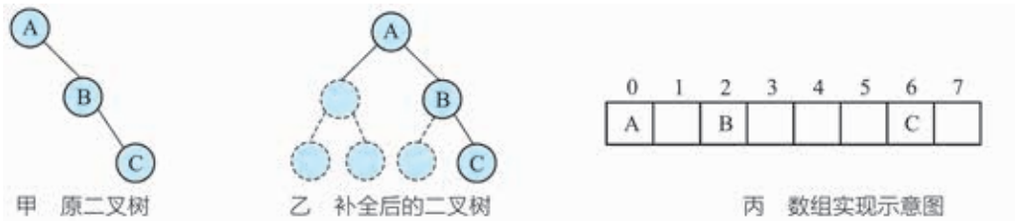


图4.2.2 非完全二叉树的数组表示

由上述可知，对于完全二叉树而言，一维数组的表示方式既简单又节省存储空间。但对于一般的二叉树来说，采用一维数组表示时，结构虽然简单，却容易造成存储空间的浪费。如图4.2.2甲所示，一个深度为k且只有k个节点的树需要如乙图所示2^k-1个节点的存储空间才能表示出来。

2. 链表实现

二叉树也可以采用链表来实现，用任意一组存储单元来存储二叉树的节点，用指向节点的指针来表示节点之间的关系。

由于二叉树的节点可能有两个孩子，即左孩子和右孩子，因此二叉树的节点至少需要3个域：一个数据域和两个指针域。两个指针域分别指向节点的左孩子和右孩子，这两个指针分别称为左指针和右指针，这样得到的链表也称为二叉链表。如图4.2.3所示的是二叉树及其对应的二叉链表实现示意图。

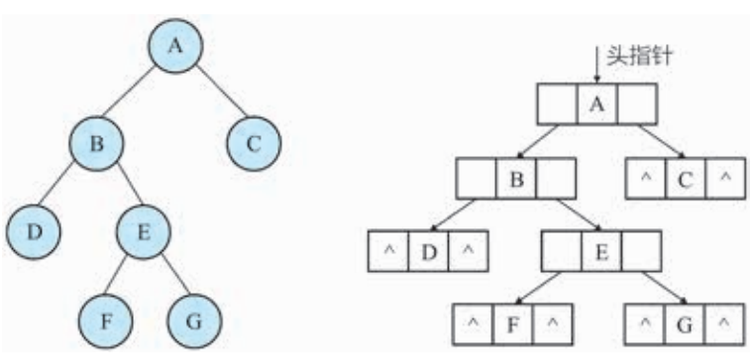


图4.2.3 二叉树的链表实现

拓展链接

二叉树的list实现

二叉树节点可以看成一个三元组，元素是左、右子树和本节点数据。Python的list可以用于组合这样的三个元素。

下面介绍用list构造二叉树的方法。

（1）空树用None表示。

（2）非空二叉树用包含三个元素的列表[d,l,r]表示，其中：d表示根节点的元素，l和r是两棵子树，采用与整个二叉树同样结构的list表示。

这样就可以把二叉树映射到一种分层的list结构，每棵二叉树都有与之对应的（递归结构的）list。例如，如图4.2.4所示的二叉树实例可以用下面的list来表示。

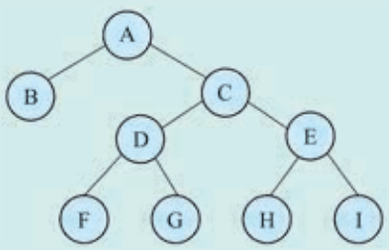


图4.2.4　二叉树

```
['A',['B',None,None],
        ['C',['D',['F',None,None],
                  ['G',None,None]],
        ['E',['H',None,None],
             ['I',None,None]]]]
```

在Python中实现二叉树的其他相关操作，可以引入各种模块，如BinaryTree（非平衡二叉树）、AVLTree（平衡的AVL树）、RBTree（平衡的红黑树）等。

4.2.2　二叉树的遍历

以图形方式表现的树形结构非常直观，但使用计算机处理时，人们需要把非线性结构的树中各个节点变成某种有意义的线性序列，才能方便程序的实现。

在完成二叉树的建立操作后，就可以对二叉树的各个节点进行访问，即遍历操作。二叉树的遍历，是指按照一定的规则和次序访问二叉树中的所有节点，使得每个节点都被访问一次且仅被访问一次。按照不同的遍历方式对节点进行访问，其处理效率不完全相同。

二叉树的遍历方式有很多，主要有前序遍历、中序遍历和后序遍历等。

1. 前序遍历

前序遍历的规则是：若二叉树为空，则空操作返回；否则，先访问根节点，再访问左子树，最后访问右子树。如图4.2.5所示，得到的前序遍历顺序为：A-B-D-H-E-C-F-I-G-J-K。

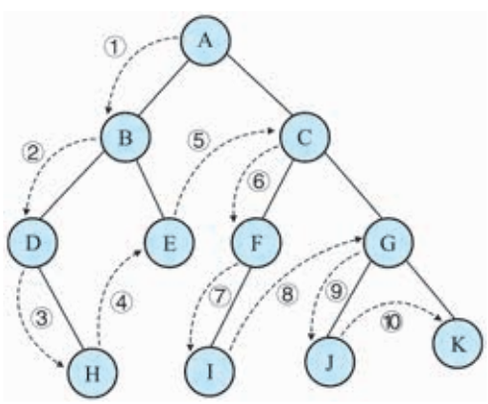


图4.2.5　前序遍历的过程

2. 中序遍历

中序遍历的规则是：若二叉树为空，则空操作返回；否则，先访问左子树，再访问根节点，

最后访问右子树。如图4.2.6所示，得到的中序遍历顺序为：D-H-B-E-A-I-F-C-J-G-K。

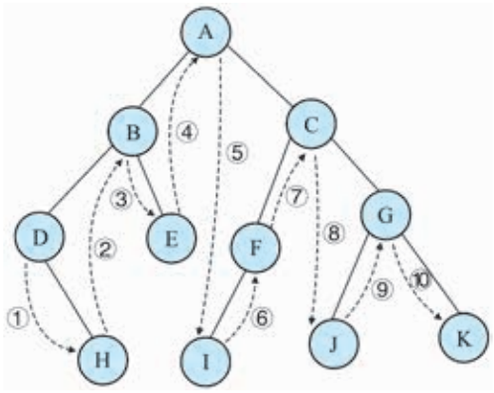


图4.2.6　中序遍历的过程

3. 后序遍历

后序遍历的规则是：若二叉树为空，则空操作返回；否则，先访问左子树，再访问右子树，最后访问根节点。如图4.2.7所示，得到的后序遍历顺序为：H-D-E-B-I-F-J-K-G-C-A。

从上述二叉树的遍历序列中可知，如果选取其中一种遍历策略对二叉树进行遍历，对于二叉树的左右子树，也需遵守该遍历原则，即二叉树的任意一个局部都必须遵守该遍历原则。对一棵二叉树进行前序遍历时，根节点总是处于遍历序列之首；中序遍历时根节点位置居中，左子树的所有节点都在其左边，右子树所有节点都在其右边；后序遍历时根节点位置在最后，其所有节点都在其左边。

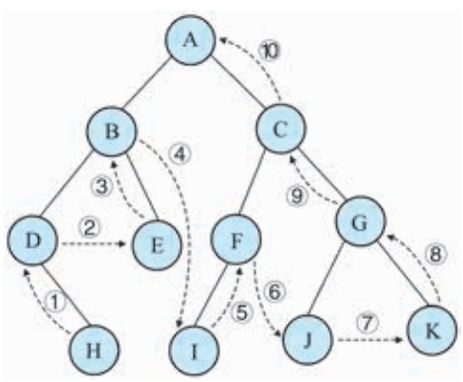


图4.2.7　后序遍历的过程

遍历二叉树的操作应用非常广泛，如果将数学表达式中的运算数和运算符视为二叉树的每个节点，那么可以构造出各种表达式树，如图4.2.8所示的是一棵表达式树。

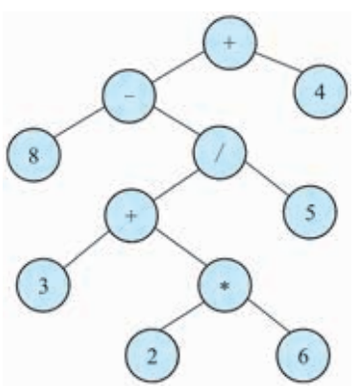


图4.2.8　表达式树

如果对图4.2.8所示的二叉树实行中序遍历，将得到遍历序列：8−(3+2*6)/5+4，即人们平时习惯书写的数学表达式，也称为中缀表达式。如果对图中二叉树实行后序遍历，那么得到遍历序列：8 3 2 6 * + 5 / − 4 +，称为后缀表达式（也称逆波兰表达式）。由于中缀表达式要将运算符放在两个操作数中间，并且在许多情况下为了确定运算顺序，括号是必不可少的。而书写后缀表达式时，因为采用了运算符紧跟在两个操作数之后的方法，从而实现了无括号处理和优先级处理，使计算机的处理规则简化为：从左到右依序完成计算，并方便求得结果。

问题与讨论

1. 已知前序遍历序列和后序遍历序列，能否唯一确定一棵二叉树？

2. 在实际生活中，学生的成绩可以划定到各个等级中，具体如表4.2.1所示。

表4.2.1　成绩等级

成绩区间	[90,100]	[80,90)	[70,80)	[60,70)	[0,60)
等级	优秀	良好	中等	及格	不及格

用二叉树来描述上述成绩等级，当成绩的数据量非常庞大时，设置不同顺序的判定条件对于查询学生成绩等级所花费的时间有何影响？

实践与体验

二叉树遍历的Python程序实现

在本章学习中，我们已经掌握了二叉树遍历的各种方法，下面将通过Python语言的程序实现来进一步体验二叉树的遍历机制。

实践内容：

给出事先设计好的二叉树结构图，分别写出前序、中序和后序的遍历结果。通过输入Python程序分别建立、遍历二叉树，运行程序输出各种遍历结果，完成验证和体验。

实践步骤：

1. 给定下面的二叉树，如图4.2.9所示。

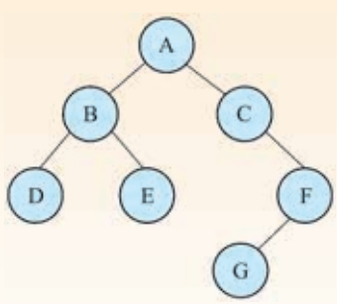


图4.2.9　二叉树

2. 写出该二叉树的前序、中序和后序遍历的序列。

3. 建立二叉树，代码如下。

```
class Node:                              #建立二叉树
    def __init__(self,value=None,left=None,right=None):
        self.value=value
        self.left=left                   #左子树
        self.right=right                 #右子树
```

4. 遍历二叉树，代码如下。

```
def preTraverse(root):                   #前序遍历
    if root==None:
        return
    print(root.value)
    preTraverse(root.left)
    preTraverse(root.right)
def midTraverse(root):                   #中序遍历
    if root==None:
        return
    midTraverse(root.left)
    print(root.value)
    midTraverse(root.right)
def afterTraverse(root):                 #后序遍历
    if root==None:
        return
    afterTraverse(root.left)
    afterTraverse(root.right)
    print(root.value)
```

5. 给定节点信息，输出遍历结果，代码如下。

```
if __name__=='__main__':
    root=Node('A',Node('B',Node('D'),Node('E')), Node('C',right=Node('F',Node('G'))))
    print('前序遍历：')
    preTraverse(root)
    print('中序遍历：')
    midTraverse(root)
    print('后序遍历：')
    afterTraverse(root)
```

结果呈现：

基于上述实践任务，完成程序体验和结果验证。

1. 运行程序，对比自己所写的遍历序列与程序输出的结果是否一致？

2. 修改程序中的节点信息，输出如图4.2.8所示的表达式树的各种遍历序列，完成结果验证。

思考与练习

1. 写出如图4.2.10所示的二叉树的前序遍历、中序遍历以及后序遍历的结果。

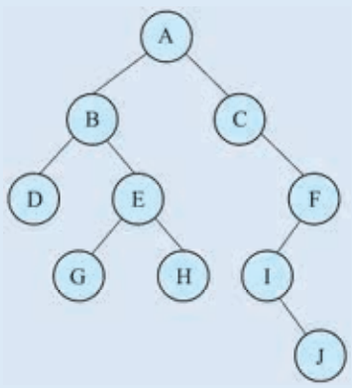


图4.2.10　二叉树

2. 如果某二叉树中有编号为1、2、3、4的四个节点，设计该二叉树，使得其前序遍历结果为1234，后序遍历结果为4321。

4.3 抽象数据类型

在程序开发实践中，仅有计算层面的抽象还不够，还需要考虑数据层面的抽象。抽象数据类型能够将数据定义表示与数据操作实现分离，以更好地支持程序的模块化组织，它也是分解和实现大型复杂系统的最重要基础技术。

4.3.1 数据类型与抽象数据类型

数据类型是程序设计领域最重要的基本概念之一，它是指一组性质相同的值的集合及定义在此集合上的一些操作的总称。使用计算机程序处理的数据，通常属于不同的类型，如整数型、浮点数型或逻辑型等。每种程序设计语言都提供了一些内置数据类型，并为每个内置类型提供了一批操作。例如，有下列Python程序语句：

```
a=13                    #语句1
b=10                    #语句2
c=a+b                   #语句3
print(c)
```

在上述程序中，语句1、语句2在给变量a和b分别赋值的同时也定义了两个数据类型是整型的变量，而语句3中的“+”操作因为已经在整型类中已经定义，对于编程者来说，此时不必关心“+”操作的含义是如何让计算机理解的。同理，该编程语言中的其他数据类型的运算操作，也都可以考虑把它们抽象出来。

抽象是指抽取出事物具有的普遍性本质，是对具体事物的一个概括。抽象是一种思考问题的方式，它隐藏了繁杂的细节，只保留实现目标所必需的信息，实现抽象化后有利于对事物的抽象，便于实现功能、提高模块独立性。人们对已有数据类型进行抽象，就有了抽象数据类型。

抽象数据类型（Abstract Data Type，简称ADT）是指一个数学模型及定义在该模型上的一组操作。ADT的基本思想是抽象，它的定义仅取决于它的一组逻辑特性，把数据结构及其操作作为一个整体来研究，而与其在计算机内部如何表示和实现无关。因此，程序设计语言的一个内置类型就可以看作是一个抽象数据类型。如图4.3.1所示，Python的字符串类型（str）就是一个抽象数据类型，在字符串对象的内部提供了一组操作可供编程者使用，每个操作都有明确的抽象意义，编程者不必受制于操作内部的具体实现技术，在程序设计时可以直接使用。

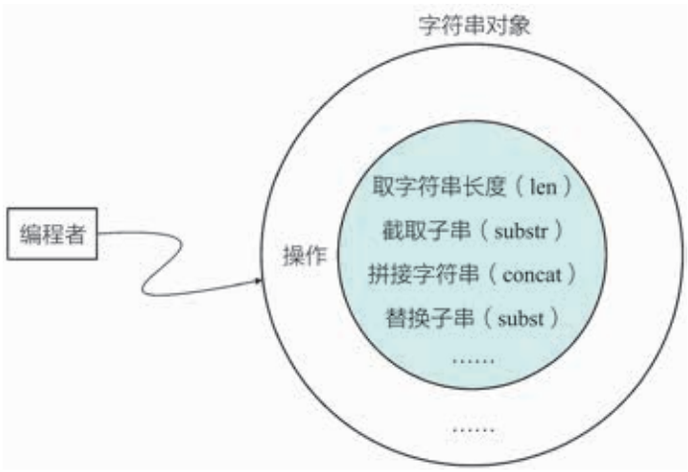


图4.3.1　字符串抽象数据类型的内部表现

抽象数据类型除了那些已经定义并实现的数据类型，还可以是编程者在程序设计时使用的函数或者单独定义的数据类型等。如使用Python的内建函数时，编程者只需考虑函数的功能是否满足实际需要，再确保函数调用时的表达式是否符合函数构造的要求，就可以使用此函数，而不需要知道该函数内部实现的任何具体细节。

根据抽象数据类型的定义，它还包括定义在该模型上的一组操作，即一个数据对象、数据对象中各数据元素之间的关系及对数据元素的操作。例如，在设计某互动媒体中的一个角色对象时，设计者可以对其定义几种基本操作，如跑动、跳跃等；一个角色对象到底需要哪些操作，只能由设计者根据实际需要来确定，如为角色对象增加下蹲等操作。

4.3.2　抽象数据类型的描述

定义一个抽象数据类型，需要清晰地表述出各方面的形式要求（如操作的名字、参数的个数和类型等）和功能要求（希望这个操作完成什么样的计算或产生什么效果等）。这类对象的功能体现为一组可以对它们使用的操作。当然，还需要为这一抽象数据类型确定一个类型名。为了便于对抽象数据类型进行规范的描述，下面给出了描述抽象数据类型的标准格式：

```
ADT 抽象数据类型名:
Data
    数据元素之间逻辑关系的定义
Operation
    操作1
        初始条件
        操作结果描述
    操作2
        ……
    操作n
        ……
endADT
```

在前三章有关线性表数据结构的学习中，已经知道了线性表数据结构会涉及的一些操作。比如：

1. 如何为创建操作提供初始元素序列；
2. 检查线性表中是否存在某个特定数据对象；
3. 改变线性表中的内容，包括加入新元素或删除已有元素等；
4. 实现一个或两个表的操作，包括表的组合操作等；
5. 实现对线性表中的每一个元素的操作，即对表元素的遍历。

上面所列的这些操作，有些是所有数据结构都需要的标准操作，如对结构的创建操作、对组合结构的判空操作等。下面呈现的是一个简单的线性表抽象数据类型，其中定义了一组最基本的操作。虽然还可以为这个类型增加更多操作，但这些基本操作已经能反映表操作的各方面特征，使用者也不必知道这些特征的任何具体实现细节。

线性表抽象数据类型

```
ADT List:
    List(self)              #创建一个新表
    is_empty(self)          #判断self是否为一个空表
    len(self)               #返回表的长度
    prepend(self,elem)      #在表头插入元素elem
    append(self,elem)       #在表尾插入元素elem
    insert(self,elem,i)     #在表的第i个位置插入元素elem
    del_first(self)         #删除第一个元素
    del_last(self)          #删除最后一个元素
    del(self,i)             #删除第i个元素
    search(self,elem)       #查找元素elem在表中第一次出现的位置
    forall(self,op)         #对表中的每个元素执行op操作
```

4.3.3 抽象数据类型的作用

抽象数据类型主要体现了程序设计中问题分解、抽象和信息隐藏的特征。它把实际生活中的问题分解成多个规模较小且容易处理的问题，然后建立一个计算机能处理的数据模型，把每个功能模块作为一个独立单元，隐藏具体的实现过程，通过一次或多次的模块调用来实现整个问题的解决。

基于上述特点，抽象数据类型能给算法和程序设计带来很多好处，主要有：使用抽象数据类型编写出来的程序结构清晰、层次分明，便于程序正确性的证明和复杂性的分析；因为其模块化的特点，在程序设计中容易纠正，具有很好的可维护性；由于抽象数据类型的表示和实现都可以封装起来，便于移植和重用；因为算法设计与数据结构设计的隔开，降低了算法和程序设计的复杂度，有助于在开发过程中少出差错，保证编写的程序有较高的可靠性，同时，允许数据结构的自由选择，给了算法的优化空间，提高了程序运行的效率。

拓展链接

常见数据结构中的抽象数据类型定义

一个简单的字符串ADT，其中定义了一些字符串操作：

```
ADT String:
    String(self,sseq)          #基于字符序列sseq建立一个字符串
    is_empty(self)             #判断本字符串是否空串
    len(self)                  #取得字符串的长度
    char(self,index)           #取得字符串中位置index的字符
    substr(self,a,b)           #取得字符串中[a:b]的子串，左闭右开区间
    match(self,string)         #查找串string在本字符串中第一次出现的位置
    concat(self,string)        #获得本字符串与另一个字符串string的拼接串
    subst(self,str1,str2)      #获得将本字符串里的子串str1替换为str2的结果串
```

最后两个操作可以实现为变动操作，实际修改本字符串；也可以实现为非变动操作，操作中生成满足要求的另一个字符串。

队列的操作通常采用另一套习惯的操作名（enqueue/dequeuer/peek）。下面为一个简单的队列抽象数据类型：

```
ADT Queue:
    Queue(self)                #创建空队列
    is_empty(self)             #判断队列是否为空，空时返回True，否则返回False
    enqueue(self,elem)         #将元素elem加入队列，即入队
    dequeue(self)              #删除队列里最早进的元素并将其返回，即出队
    peek(self)                 #查看队列里最早进入的元素，不删除
```

定义二叉树抽象数据类型时，主要考虑如下几个方面：节点是二叉树的基础，通常主要用节点保存与应用有关的信息；作为二叉树的表示，还需要记录二叉树的结构信息，至少需要保证能够检查节点的父子关系，例如，能从一个节点找到其左/右子节点。下面是一个基本的二叉树抽象数据类型的定义：

```
ADT BinTree:                   #一个二叉树抽象数据类型
    BinTree(self,data,left,right) #构造操作，创建一个新二叉树
    is_empty(self)             #判断self是否为一个空二叉树
    num_nodes(self)            #求二叉树的节点个数
    data(self)                 #获取二叉树根节点存储的数据
    left(self)                 #获得二叉树的左子树
    right(self)                #获得二叉树的右子树
    set_left(self,btree)       #用btree取代原来的左子树
    set_right(self,btree)      #用btree取代原来的右子树
    traversal(self)            #遍历二叉树中各节点数据的迭代器
    forall(self,op)            #对二叉树中的每个节点的数据执行op操作
```

思考与练习

1. 列举数据结构、数据类型和抽象数据类型三者之间的联系和区别。
2. 结合第三章中栈数据结构的相关操作，尝试定义栈的抽象数据类型。

巩固与提高

1. 一个四世同堂的大家庭，曾祖一辈（被视作第1代）有三个孩子，第2代分别有两个、一个和两个孩子，第3代的一些孩子也已经长大成家并有了孩子，共计14人。给这个大家庭的每个人用一个字母进行标记，用集合形式表示如下：

（1）所有人的集合是N={A,B,C,D,E,F,G,H,I,J,K,L,M,N}。

（2）反映其家庭结构的父子关系是R={<A,B>,<A,C>,<A,D>,<B,E>,<B,F>,<C,G>,<D,H>,<D,I>,<G,J>,<G,K>,<H,L>,<I,M>,<I,N>}。

请用树形结构来抽象描述上述家庭成员的组成情况。

2. 已知一棵完全二叉树的第6层有8个叶子节点，求该完全二叉树可能的最大节点数量。

3. 已知一棵二叉树共有25个节点，其中5个是叶子节点，求出度为1的节点数量。

4. 已知一棵二叉树的中序遍历序列为DGBAECHF，后序遍历序列为GDBEHFCA，求此二叉树的前序遍历序列。

5. 某表达式树如图4.3.2所示。求出该表达式树的中缀表达式的值。

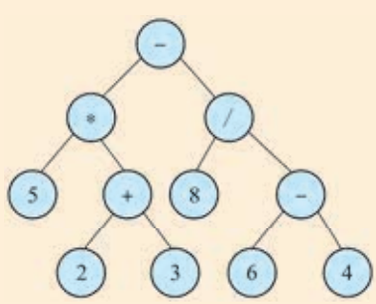


图4.3.2　表达式树

6. 某软件界面上需要添加一些文本框对象并进行相关操作，结合实际需求，尝试定义一个文本框的抽象数据类型。

项目挑战

利用二叉树实现信息加密

信息加密是保证信息安全的重要手段，加密的方法有很多，其中利用二叉树的遍历生成新的信息串序列，是典型的生成法加密，加密性能比较高。例如，在甲、乙双方共同完成的加密、解密过程中，甲方先以二叉树作为载体按照指定的规则对一个二进制数序列进行加密，同时将加密得到的密文（二叉树的节点编号串）和该二叉树的中序遍历序列发送给乙方。乙方确认收到密文后，甲方再发送二叉树的前序序列或后序序列（密钥）给乙方。乙方在综合各种信息的基础上推导出该二叉树的结构，最后完成解密。

例如，甲方设计了如图4.3.3所示的用于加密的二叉树，规则中规定左、右孩子节点的边分别用“0”和“1”来表示，甲方事先将该二叉树的中序遍历序列（274158396）发送给乙方。

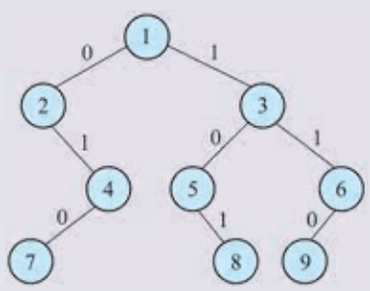


图4.3.3　二叉树

对二进制数序列进行加密的规则如下：

从二叉树的根节点开始，取明文二进制序列的每一位（“0”或“1”），找准对应的左、右孩子节点进行遍历。若二叉树中不存在序列当前位所对应的左、右孩子，则结束这一轮的编码，同时回到二叉树的根节点重新开始新一轮的遍历，直到序列中的每一位都完成编码加密。

下面呈现的是甲方对二进制数序列“1001101010110”的加密过程：

（1）序列第1位是“1”，从根节点开始寻找右孩子节点，来到节点3；第2位是“0”，来到节点5；第3位是“0”，由于节点5没有左孩子，则第一轮编码结束，同时回到根节点，得到序列前两位（10）的编码为“5”。

（2）取序列第3位是“0”，从根节点开始来到节点2；第4位是“1”，来到节点4；第5位是1，节点4没有右孩子节点，则第二轮编码结束，同时回到根节点，得到序列第3至4位（01）的本轮编码为“4”，下一轮将从第5位开始。

（3）按照上述方法重复执行，得到最后的编码为：54879，并将此编码（密文）发送给乙方。

乙方确认收到密文后，甲方再发送二叉树的前序序列（124735869）或后序序列（742859631）给乙方，乙方推导出该二叉树的结构，并根据二叉树重现密文中每个数字（即二叉树的节点编号）的访问路径（用边的0、1值来表示），最后完成二进制数序列的解密。

项目任务

学校创新社团举办信息安全竞赛，你与同学被选派参加“利用二叉树实现信息加密”的竞赛项目。该项目需要你（甲方，也是加密方）与同学（乙方，也是解密方）双方共同完成。两位同学需要事先熟悉加密、解密规则，熟练掌握二叉树遍历的相关技能，以良好的状态快速完成竞赛项目。

过程与建议

1. 竞赛前的准备

（1）熟悉加密规则。进行任务挑战前，甲、乙双方需一同熟悉加密规则，例如规则中左、右孩子节点的边是用“0”还是“1”来表示？考虑选择哪两种遍历序列来确定二叉树的形态？

（2）掌握遍历技能。二叉树的遍历是完成本次竞赛项目的必备技能。为确保竞赛项目的顺利开展，甲方必须具备根据自己创建的二叉树准确无误地写出前序、中序和后序遍历结果的能力，乙方也需要具备根据遍历结果，推导出二叉树形态的能力。

（3）交换角色，实践预演。为了在竞赛中有良好的表现，前期训练必不可少。甲、乙双方可交换加密、解密的角色，进行多次预演，确保竞赛项目准确高效完成。

2. 竞赛中的实施

根据上述信息加密、解密的规则，甲、乙双方需要完成如下任务：

（1）甲方构建一棵二叉树，写出该二叉树的各种遍历序列，并将该二叉树的中序遍历序列发送给乙方。

（2）甲方基于该二叉树，根据规则对明文进行加密并将得到的密文发送给乙方。

（3）乙方根据甲方所提供的两种二叉树遍历结果，推导二叉树的形态，完成密文的解密。

3. 竞赛后的反思

（1）展示竞赛结果。你和同学是否已经挑战成功？如果挑战成功，请谈谈成功的关键所在；如果挑战失败，请反思失败的原因。

（2）交流探讨。用几句话概括说明此加密技术在信息传输中的实际应用价值。

评价标准

请根据项目实施的过程、效果以及成果展示交流的结果，对自己完成项目的情况进行客观的评价，并思考后续完善的方向。把评价结果和完善方案填写在下面的表格中。

评价条目	说明	评分（1～10分）	评分主要依据阐述	后续完善方向
甲方任务	能够生成具有合理复杂度的、用于加密的二叉树			
	根据加密规则，能够准确生成密文			
	能准确描述二叉树的各种遍历序列			
乙方任务	能根据各种二叉树的遍历序列，推导出二叉树的形态			
	根据加密规则，结合二叉树的形态完成二进制序列的解密			
甲方、乙方共同任务	报告内容与项目研究目标一致，对二叉树加密解密的原理和实施有了科学的理解			

拓展项目

1. 决策树是附加概率结果的一个树状决策图，是直观运用统计概率分析的图法。机器学习中决策树是一个预测模型，树中的每一个节点表示对象属性的判断条件，其分支表示符合节点条件的对象，树的叶子节点表示对象所属的预测结果。如图4.3.4所示的是银行系统用于预测贷款用户是否具有偿还贷款的能力的简单决策树。

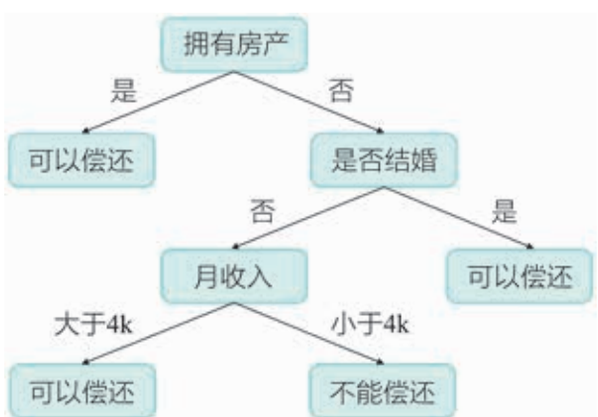


图4.3.4　决策树

请根据个人的兴趣特长、学习情况、现有资源准备等信息，构造一棵能够预测学生应该选报哪所大学、什么专业的决策树。

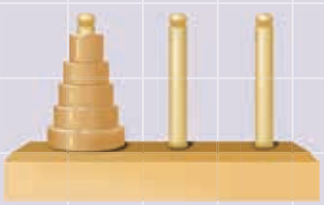
2. 某连锁超市集团打算在某社区新建一个大型超市，这个社区包含N个住宅小区，相邻小区的直线距离相同。现已知每个小区的居住人口数量，请你为该集团设计超市选址的方案。

第五章 5

数据结构与算法

在计算机领域，一个算法实质上是针对所处理问题的需要，在数据的逻辑结构和存储结构的基础上施加的一种运算规则。对问题中数据的不同组织方式，解决问题的不同的策略将导致不同的算法。因此，数据结构是算法实现的基础，算法总是要依赖于某种数据结构来实现。在问题解决时，要根据问题的需要，分析数据的特征及各数据之间的关系，选择合适的逻辑结构和存储结构，进而设计出更为高效的算法。

问题与挑战



- 汉诺塔问题。有个古老的传说：在某个庙里的一块黄铜板上插着三根宝石针，最左边的一根针上从下到上穿插了由大到小的64片金片（右图所示是5片的模型）。有一个僧侣按照下面的法则移动这些金片：一次只移动一片，不管在哪根针上，小金片必须在大金片上面，将最左边一根针上的金片全部移动到最右边一根针上，总共需要移动金片18446744073709551615次。若用计算机程序来模拟实现，该如何设计算法？

- 在现代社会，生活节奏越来越快，人们经常需要在某一类事物中利用数据的有序性快速地进行处理。例如，在网购火车票时，需要根据出发时间、到站时间、时长、票价、余票数量等方式的排序结果选择相应的车票；在计算机上查看某文件夹下的所有图片时，可按名称、修改日期、类型、大小等方式进行排序；利用无人机送快递时，为了提升客户体验，可按目的地的远近进行排序，合理安排配送顺序。那么，数据的排序是如何实现的？

- 若某大型超市货架上放有100000余种不同货品，当收银员扫描一件货物的条形码时，计算机需要在100000余种不同记录中去寻找这件商品的名称和价格。如果用于收款的计算机能在1秒内查找10000次，且采用逐个比较的查找方式，那么完成对所有货物查找一遍就需要耗时约10秒。如果将你所买的若干商品都扫描一遍，花费的时间是难以忍受的。而超市扫描商品几乎是瞬间完成，那背后的数据结构和算法是如何设计的？

学习目标

1. 初步理解数据结构与算法的关系。
2. 理解迭代与递归思想及其在问题解决中的应用。
3. 掌握排序算法，理解排序算法与数据结构的关系。
4. 掌握查找算法，理解查找算法与数据结构的关系。

内容总览

5.1 数据结构与算法效率

在计算机科学中，数据结构与算法有着密不可分的关系。解决实际问题时，数据总是以一定的组织结构关系体现并存储，数据结构的设计和选择关注的是数据的逻辑结构、存储结构以及基本操作；算法的设计和选择更多的是关注如何在数据结构的基础上综合运用各种基本数据操作来解决实际问题。因此，算法的设计和选择总是依赖于数据结构，算法设计的同时也伴随着数据结构的设计，两者都是为最终解决问题服务。同时，数据结构的设计和选择会对算法效率产生一定的影响。

5.1.1 算法效率

同一个问题可能会有不同的解决方法，也就是说可能有不同的算法，就如同“一题多解”一样。有的算法效率高一些，有的算法效率低一些。通常，算法效率的高低可由算法复杂度来度量。算法复杂度又分为算法的时间复杂度和空间复杂度，其中时间复杂度反映了算法执行所需要的时间，而空间复杂度反映了算法执行所需要占用的存储空间。

算法运行需要时间，一般将算法中语句的执行次数作为时间复杂度的度量标准。语句总的执行次数T(n)是关于问题规模n的函数。 所谓问题规模（也称为输入的大小）是指处理问题的大小，即用来衡量输入数据量的整数。例如，在约瑟夫问题中，用数组或链表中元素的个数作为问题规模。时间复杂度常用符号O表示，不包括T(n)函数的低阶项和首项系数，如$\frac{1}{2}n(n-1)$的量级与n^2相同，其时间复杂度可表示成$O(n^2)$。

拓展链接

推导大O阶的方法

用O()来体现算法时间复杂度，称之为大O记法。其推导方法如下：

1. 用常数1取代运行时间中的所有加法常数。
2. 在修改后的运行次数函数中，只保留最高阶项。
3. 如果最高阶项存在且不是1，那么去除与这个项相乘的常数。

得到的结果就是大O阶。

例如，$\frac{1}{2}n(n-1) \xrightarrow{\text{保留最高阶项}} \frac{1}{2}n^2 \xrightarrow{\text{去除常数}} n^2$

通常，随着问题规模n的增大，函数值增长较慢的算法较优。例如，求1+2+…+n的和可以用以下两种算法来实现：

算法一：

```
n=int(input())                #执行1次
s=(1+n)*n/2                   #执行1次
print(s)                      #执行1次
```

算法二：

```
n=int(input())                #执行1次
s=0                           #执行1次
for i in range(1,n+1):        #执行n次
    s=s+i                     #执行n次
print(s)                      #执行1次
```

算法一是顺序结构，语句总的执行次数为3次，算法中语句的执行次数与问题规模n无关，时间复杂度为O(1)，称为常量阶；而算法二中有一重循环，语句总的执行次数为2n+3次，则算法中语句的执行次数与问题规模n呈线性增大关系，时间复杂度是O(n)，称为线性阶。

再来看下面的例子：

```
n=int(input())
s=0
x=0
for i in range(1,n+1):
    for j in range(1,n+1):
        x=x+1                 #执行n*n次
        s=s+x
print(s)
```

这个例子中，程序段中包含二重循环，i从1到n，每次都让j循环n次，而内循环的循环体由语句“x=x+1”和“s=s+x”组成，其执行次数就是n×n次，也就是这个算法中的循环体需要执行n^2次，则算法中语句的执行次数与问题规模n呈平方增大关系，时间复杂度是$O(n^2)$，称为平方阶。此外，算法的时间复杂度还有对数阶$O(\log_2 n)$、指数阶$O(2^n)$等。

算法的时间复杂度反映了程序执行时间随问题规模增长而增长的量级，在很大程度上能很好地反映出算法的优劣。

拓展链接

常见时间复杂度的耗时比较

常见的时间复杂度耗费时间的大小关系为：$O(1)<O(\log_2 n)<O(n)<O(n^2)<O(n^3)<O(2^n)<O(n!)$。一些常见函数的增长率如图5.1.1所示。

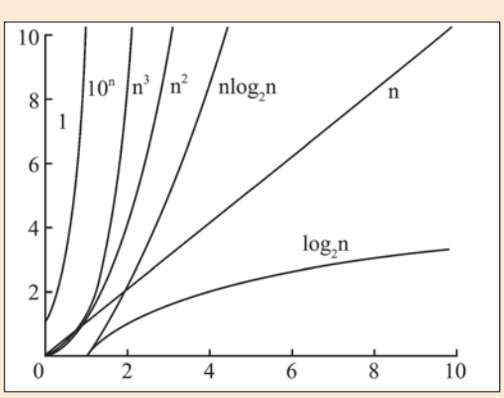


图5.1.1　常见函数的增长率

问题与讨论

查阅相关资料，举例说明空间复杂度是如何度量的。

5.1.2 数据结构对算法效率的影响

数据组织成不同的结构，是为了满足不同问题的需求，便于算法对数据的操作，提高算法处理数据的效率。

例如，要显示10000种商品中某个商品的价格，在设计数据结构时，可以将这10000种商品的价格数据采用数组或链表来组织。但从算法效率的角度考虑，此时数据结构采用数组比链表更方便，这是因为数组中的元素可以按其下标方便地被随机访问，时间复杂度为O(1)；而在链表中访问任何节点都必须从首节点（或尾节点）开始按序访问到指定节点，时间复杂度为O(n)。另一方面，如果要在10000种商品中添加或删除某个商品信息，这时，算法需要对元素做插入、删除操作，那么数据结构采用链表比较合适，这是因为在数组中插入或删除一个元素都可能引起大量元素的移动操作，时间复杂度为O(n)；而在链表中插入或删除一个节点只需要对个别节点的链域进行修改，时间复杂度为O(1)。因此，数据结构的不同选择会影响算法的运行效率。

思考与练习

1. 如何度量某个算法的效率高低？
2. 分析下面程序段的时间复杂度。

（1）

```
a=int(input())
b=int(input())
if a>b:
    print(a)
else:
    print(b)
```

（2）

```
def func(n):
    i=1
    k=100
    while i<n:
        k=k+1
        i=i+2
    return k
```

5.2 迭代与递归

在前面的学习中，“让某个变量的值在原值基础上不断更新，逐步逼近所求解”“二叉树由一个根节点和两棵分别称为左子树和右子树的互不相交的二叉树构成”等内容中蕴含了迭代和递归的思想。

5.2.1 迭代

迭代是重复反馈过程的活动，其目的通常是为了使结果符合目标需求。例如，针对某个产品的开发，可以采用迭代的方式进行，即初期先开发出一个产品，这个产品可能会很不完善，很多功能不稳定或不具备等，但客户看过后会提出详细的修改意见，开发人员就能清楚所开发的产品与客户需求之间的差距，经过改进后，再一次让客户提意见，再次改进……经过若干次的重复反馈，所开发出来的产品在功能和质量等方面都逐渐符合客户的需求。计算机解决问题时，也经常采用这种迭代的方式，即迭代算法。它利用计算机运算速度快、适合做重复性操作的特点，让计算机重复执行一组指令（或一些步骤），这组指令（或这些步骤）每执行一次时，都会将变量从原值递推出一个新值。

利用迭代算法处理问题，需要考虑以下三个方面：

①确定迭代变量。在能够用迭代算法处理的问题中，至少具有一个直接或间接地不断由旧值递推出新值的变量，这个变量就是迭代变量。

②建立迭代关系式。所谓迭代关系式，指如何从变量的前一个值推出其下一个值的公式（或关系）。

③控制迭代过程。迭代过程在经过若干次重复执行以后要能结束，因此，要设定迭代结束的条件。

例如，采用迭代算法求a的平方根，其基本思路是：先估测一个近似值x，然后不断令x等于x和$\frac{a}{x}$的平均数（迭代公式为：$x_{n+1}=\frac{1}{2}(x_n+\frac{a}{x_n})(n\geq 0)$），经过若干次迭代后，x的值将逐渐逼近a的平方根（当x_{n+1}与x_n值无限逼近时，可看作$x_{n+1}=x_n$，则公式$x_{n+1}=\frac{1}{2}(x_n+\frac{a}{x_n})$可化简为$x_{n+1}^2=a$，$x_{n+1}$就是a的平方根）。以求2的平方根为例，可估测一个近似值（如$x_0=1$）作为初值，设定前后两次求出的x的差的绝对值小于10^{-5}。具体步骤如下：

①设定迭代变量x的初始值x_0，如取$x_0=1$。

②确定迭代公式：$x_{n+1}=\frac{1}{2}(x_n+\frac{a}{x_n})$，并求出$x_{n+1}$的值。

③重复步骤2，直到前后两次求出的x值（x_n和x_{n+1}）满足一定的误差，即$|x_{n+1}-x_n|<10^{-5}$，迭代结束。x_{n+1}就是所求的平方根。

迭代变量x的迭代过程如表5.2.1所示。

表5.2.1 迭代变量迭代过程

迭代次数	x_n	x_{n+1}	$\lvert x_{n+1}-x_n \rvert$
1	1	1.5	0.5
2	1.5	1.416667	0.083333
3	1.416667	1.414216	0.002451
4	1.414216	1.414214	0.000002

当前后两次求出的x值（x_n和x_{n+1}）的差的绝对值为0.000002时，停止迭代，得出2的平方根约为1.414214。相应的程序及测试结果如下所示：

程序	测试结果
a=int(input("请输入一个需要求其平方根的数：")) x=a/2 while ((abs((x+a/x)/2-x))>0.00001): 　　x=(x+a/x)/2 print(a,"的平方根约为",round((x+a/x)/2,6))	请输入一个需要求其平方根的数：2 2 的平方根约为 1.414214

问题与讨论

在用迭代法求2的平方根的例子中，若将迭代变量x的初值换为其他数值，对运行结果和迭代次数是否有影响？

拓展链接

欧几里得算法

欧几里得算法又称辗转相除法，用于计算两个整数m, n的最大公约数。基于定理：

gcd(m, n)=gcd(n, m mod n)

即：整数m，n的最大公约数等于n和m除以n的余数的最大公约数。

欧几里得算法在执行时，也是一个反复迭代的过程，直到余数等于0为止。Python代码实现如下：

```
def gcd(m, n):
    while n !=0:
        temp=n
        n=m % n
        m=temp
    return m
```

从上面的程序可以看出，m，n是迭代变量，迭代关系是n→m和m% n→n，由旧值推出新值，然后循环执行，直到余数为0，结束迭代。

5.2.2 递归

生活中有类似情景：一个学生坐在第10排，他的作业本被放到了第1排。要想拿回作业本，他可以将“帮拿作业本”的信息传给第9排的学生，而第9排的学生可以将同样的信息传给第8排的学生……如此下去，直到信息传到第1排的学生，于是第1排的学生把作业本递给第2排，第2排又递给第3排……最后，作业本到了第10排学生的手里。在这个过程中，第10排的学生要想拿到作业本，先要第9排的学生拿到作业本……他们处理问题的方式是一样的，即大问题的解决中嵌套着与原问题相似的规模较小的问题。这种解决问题的方式在计算机科学中称为递归，通过函数自己调用自己来实现，即一个函数在其定义中直接或间接调用自身的一种方法。在数据结构与算法设计中，递归十分有用，它往往能使函数的定义和算法的描述简洁且易于理解，极大地减少程序代码量。如前面学过的二叉树，由于其本身固有的递归特性，特别适合用递归的形式来描述。

能采用递归描述的算法通常有这样的特征：为求解规模为N的问题，设法将它分解成规模较小的问题，然后从这些小问题的解中方便地构造出大问题的解，并且这些规模较小的问题也能采用同样的分解和综合方法。当递归到达某个边界，如问题规模缩减为0或1时，能直接得解。因此，在设计递归算法时，要满足两个条件：确定递归公式和递归结束条件。

例如，利用递归算法求n的阶乘（n!=1×2×…×n）。由数学知识可知，n阶乘的递归定义为：它等于n乘以n−1的阶乘，即n!=n×(n−1)!，并且规定0!=1。设函数fac(n)=n!，则fac(n)可表示为：

$$fac(n)=\begin{cases}1 & (n=0)\\ n\times fac(n-1) & (n>0)\end{cases}$$

按照这个公式，可以将求n!的问题转化成求(n−1)!的问题；而求(n−1)!的问题，又可以转化成求(n−2)!的问题；求(n−2)!的问题，又可以转化成求(n−3)!的问题，如此继续，直到最后转化成求0!的问题。再反过来，依次求出1!，2!，…，直到最后求出n!。因此，在该问题中，递归公式是fac(n)=n×fac(n−1)，当n=0时递归结束。相应的程序及测试结果如下所示：

程序	测试结果

def fac(n):
　　if n == 0:
　　　　s=1
　　else:
　　　　s= n * fac(n−1)
　　return s
print(fac(3))
	6

当主程序执行函数fac(3)时，引起第1次函数调用，进入函数后，参数n=3，应执行计算3*fac(2)。

为了计算 fac(2)，将引起对函数 fac 的第 2 次调用（递归调用），重新进入函数，参数 n=2，应执行计算 2* fac(1)。

为了计算 fac(1)，将引起对函数 fac 的第 3 次调用（递归调用），重新进入函数，参数 n=1，应执行计算 1* fac(0)。

为了计算 fac(0)，将引起对函数 fac 的第 4 次调用（递归调用），重新进入函数，参数 n=0，此时执行 s=1，完成第 4 次调用，回送结果 fac(0)=1，返回到调用处（即回到第 3 次调用层）。

计算 1* fac(0)=1*1=1，完成第 3 次调用，回送结果 fac(1)=1，返回到第 2 次调用层。

计算 2* fac(1)=2*1=2，完成第 2 次调用，回送结果 fac(2)=2，返回到第 1 次调用层。

计算 3* fac(2)=3*2=6，完成第 1 次调用，回送结果 fac(3)=6，返回到主程序。

以上调用的执行和返回情况，如图 5.2.1 所示。

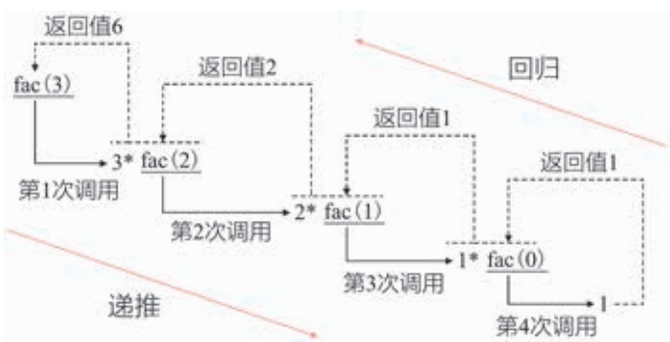


图5.2.1　递归调用过程

递归算法的执行过程分递推和回归两个阶段。在递推阶段，把较复杂的问题（规模为 n）的求解递推到一些简单问题（规模小于 n）的求解。例如，上例中，求解 fac(3) 递推到求解 fac(2)，即为了计算 fac(3)，必须先计算 fac(2)，而计算 fac(2)，又必须先计算 fac(1)。依次类推，直至计算 fac(0)，能立即得到结果 1。在递推阶段，必须要有终止递推的情况，即在函数 fac 中有 n 为 0 的情况。

在回归阶段，当获得最简单情况的解后，逐级返回依次得到稍复杂问题的解。例如，得到 fac(0) 后，返回得到 fac(1) 的结果……在得到 fac(2) 的结果后，返回得到 fac(3) 的结果。

问题与讨论

利用迭代和递归思想解决问题时有何区别？算法实现时两者有哪些优缺点？

拓展链接

递归与栈

计算机在执行递归程序时，是通过栈的调用来实现的。递归程序是将复杂问题分解为一系列简单的问题，从要解的问题起，逐步分解，并将每步分解得到的问题放入“栈”中，运用栈的“后进先出”的性质，这样栈顶是最后分解得到的最简单的问题，解决了这个问题后，次简单的问题可以得到答案，以此类推。所以，分解问题可以看作是进栈的过程，而解决问题则是出栈的过程。

对于求n阶乘的递归函数fac(n)，当调用它时系统自动建立一个栈，该栈中的元素包含每次调用的计算结果和返回地址的情况。例如，求fac(3)，即y=fac(3)，判断可知执行得到f = 3*fac(2)，但是，这并不是确切的值，还有fac的函数调用，所以系统就会将当前计算出的结果存放入栈内，相当于执行完fac(3)后在执行fac(2)之前，系统就会在栈顶插入f=3*fac(2)以及一个返回地址。把这些运算得到的内容压入栈顶后，然后去执行fac(2)，产生f=2*fac(1)，n变为了1。此时还没有结束，在执行fac(1)前，系统又把产生的f=2*fac(1)和一个返回地址又压入到先前开辟的栈中……当执行fac(0)时，符合“n=0”这个条件，所以f=1，此时系统就会把f=1和一个返回地址压入到栈中。此时，不再进行向后的fac函数运算，递归调用结束。下个阶段就是数据出栈，系统会自动地把栈顶的数据读出来，即首先把f=1和一个返回地址读出，然后读出第二个数据f=2*fac(1)以及一个返回地址……直到执行完f=6，按照栈底数据的返回值，就把f赋值给了主程序中的y。如图5.2.2所示是函数参数的入栈和出栈过程。可以清楚地发现，递归函数刚好利用了栈的“后进先出”的特性。

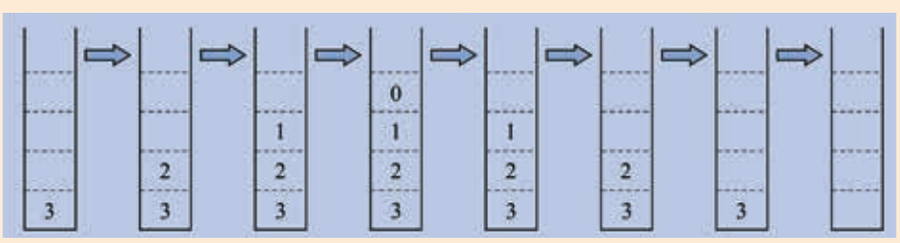


图5.2.2　计算fac(3)过程中参数的入栈和出栈过程

例1　汉诺塔游戏

汉诺塔游戏的装置是一块铜板，上面有三根针，其中最左侧一根针上放着从大到小的n个圆盘。游戏的目标是把所有圆盘从最左侧一根针上移动到最右侧一根针上，中间一个针作为过渡。游戏规定每次只能移动一个圆盘，并且大盘子不能压在小盘子上面。如图5.2.3所示是3个圆盘的情形。

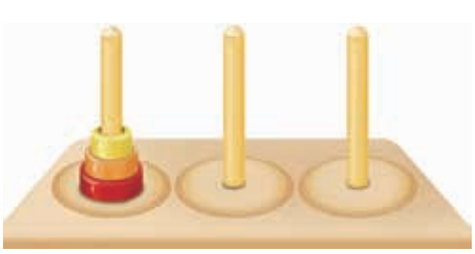


图5.2.3　3个圆盘的汉诺塔模型

（1）抽象与建模

汉诺塔游戏可抽象为：从左到右有A、B、C三根柱子，其中A柱子上面放着从大到小的n个圆盘，现要求按一定规则，将A柱子上的圆盘移到C柱子上去。如图5.2.4所示是3个圆盘抽象后的模型。

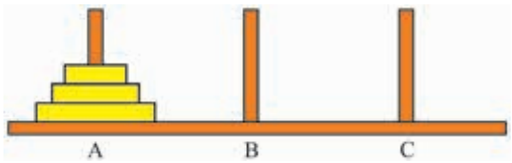


图5.2.4　抽象后的汉诺塔模型

要将3个圆盘从A柱移动到C柱，需要借助B柱进行过渡。一个可行的移动方案是：A→C（表示将A柱上的最上面的一个圆盘移动到C柱上），A→B，C→B，A→C，B→A，B→C，A→C。每一步移动情况如图5.2.5所示。

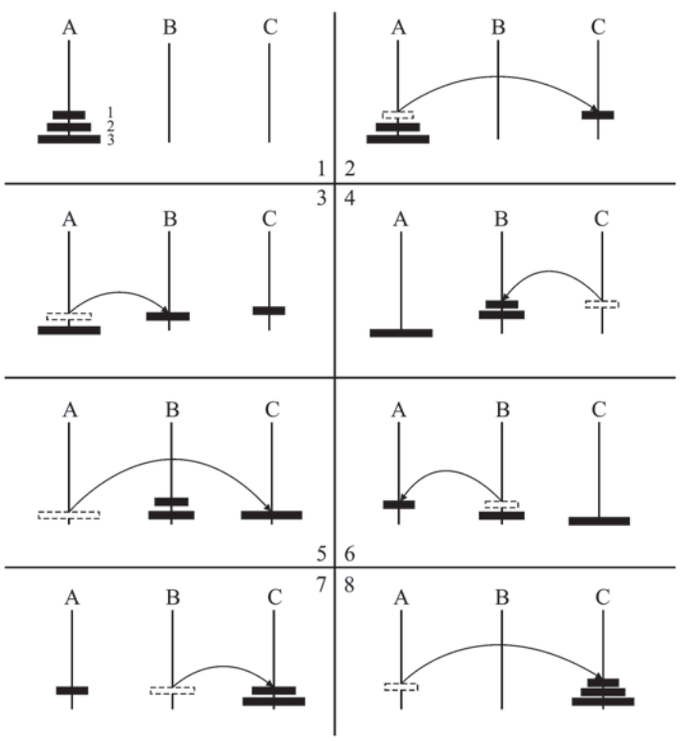
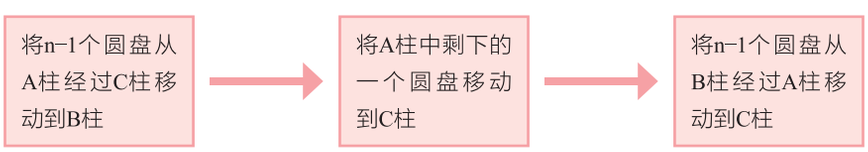


图5.2.5　3个圆盘的移动情况

考虑一般情形，将n个圆盘从A柱经过B柱移动到C柱，可以建立如下模型：



（2）设计算法

从问题的抽象与建模中可以发现，原来与n有关的问题变成了与n−1有关的问题，重复这个过程，每次n减1，最后当n=1时，直接移动该圆盘。因此，该问题可采用递归算法来实现，即：

①定义一个实现圆盘移动的函数move。如将n个圆盘从A柱经过B柱移动到C柱，可调用函数move(n, a, b, c)，其中n表示A柱上的圆盘个数，a、b、c分别表示A柱、B柱、C柱。

②将n个圆盘从A柱经过B柱移动到C柱，可以分解成如下递归调用：

move(n−1, a, c, b)

a→c

move(n−1, b, a, c)

③重复执行步骤②，直到n=1时，直接移动圆盘，递归结束。

（3）编写程序

根据算法，编写的程序及测试结果如下：

程序	测试结果

def move(n, a, b, c):
  if(n == 1):
  print(a,"−>",c)
  return
  move(n–1, a, c, b)
  move(1, a, b, c)
  move(n–1, b, a, c)
move(3, "A", "B", "C")
	A−>C
	A−>B
	C−>B
	A−>C
	B−>A
	B−>C
	A−>C

实践与体验

走迷宫

有一个迷宫，图案如图5.2.6所示，红色区域表示不能通行，蓝色区域表示能通行，在迷宫中通行的方向是上下左右四个方向。

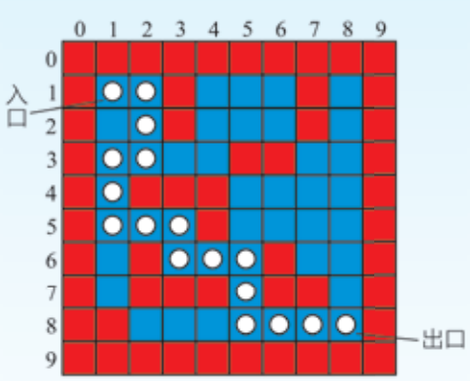


图5.2.6　迷宫图案

实践内容：

从入口(1,1)位置进入迷宫，编程判断能否从出口(8,8)位置成功走出。

实践步骤：

1. 设计数据结构和算法。

迷宫图案中的通行区域用数字表示：1是可以通行的点，0是不能通行的点。这样，迷宫图案就转化成一张由0和1组成的二维表，可存储成二维数组，如图5.2.6所示的10×10迷宫图案可以存储成如下数组：

```
maze=[[0,0,0,0,0,0,0,0,0,0],
      [0,1,1,0,1,1,1,0,1,0],
      [0,1,1,0,1,1,1,0,1,0],
      [0,1,1,1,1,0,0,1,1,0],
      [0,1,0,0,0,1,1,1,1,0],
      [0,1,1,1,0,1,1,1,1,0],
      [0,1,0,1,1,1,0,1,1,0],
      [0,1,0,0,0,1,0,0,1,0],
      [0,0,1,1,1,1,1,1,1,0],
      [0,0,0,0,0,0,0,0,0,0]]
```

走迷宫时，从入口(1,1)位置进入，每一次移动都有上下左右四个方向可选。如果下一个位置可以走，那么把原来的位置标记为已经走过，并走到下一个位置。这时候问题又变成从4个方向里选一个，这是问题的递归部分。当走到(8,8)位置的时候，即找到出口，求解完成，也是递归结束的条件。

2. 编写程序。

（1）判断坐标有效性的函数（若超出数组边界或是不满足值为1的条件，说明该点无效，返回False，否则返回True）。

```
def valid(maze,x,y):
    if (x>=0 and x<len(maze) and y>=0 and y<len(maze[0]) and maze[x][y]==1):
        return True
    else:
        return False
```

（2）移步函数。

```
def walk(maze,x,y):
    # 如果位置是迷宫的出口，说明成功走出迷宫
    if(x==8 and y==8):
        print("successful!")
        return True
```

（3）递归主体。

```
    if valid(maze,x,y):
        maze[x][y]=2                    # 做标记，防止折回
        # 针对四个方向依次试探
```

```
        walk(maze,x-1,y)
        walk(maze,x,y-1)
        walk(maze,x+1,y)
        walk(maze,x,y+1)
```

（4）函数调用。

```
walk(maze,1,1)
```

结果呈现：

基于上述实践步骤，完成程序体验。程序运行结果如下：

```
successful!
successful!
```

思考与练习

1. 用迭代法求$y=\sqrt[3]{x}$的值（误差小于10^{-5}）。要求：编写一个程序，x由键盘输入，迭代公式为：$y_{n+1}=\frac{2}{3}y_n+\frac{x}{3y_n^2}$。

2. 验证角谷猜想。所谓角谷猜想，是指对于任意一个正整数，若是奇数，则乘3加1；若是偶数，则除以2。得到的结果再按照上述规则重复处理，最终总能够得到1。

要求：编写一个程序，输入一个正整数n，把n经过有限次运算后，输出最终变成1的全过程。

3. 斐波那契数列是这样一个数列：1，1，2，3，5，8，13，21，34，…，其定义如下：

f(0)=0

f(1)=1

f(n)=f(n−1)+f(n−2)（n≥2）

编程求f(40)的值，请分别用迭代和递归算法实现，并分析这两种算法的时间复杂度。

5.3 数据排序

在计算机软件系统中，排序是一种常见的操作，如文件夹中图片文件可分别按名称、大小、类型、修改日期等方式进行排序；电子邮件列表一般按照日期排序，最新的邮件被放置在最顶端；购物网站上搜索到的某类商品可按价格、销量、信用等方式进行排序。

5.3.1 排序

排序（sorting）就是整理数据的序列，使其中元素按照某个值的递增（或递减）的次序重新排列的操作。在排序的过程中，数据元素的值保持不变，但其在序列中的顺序可能会改变。

对一次具体排序而言，总是针对某一组数据元素的某种具体的序关系进行操作。待排序数据的存储方式一般有两种：一种是将数据依次存放在一组地址连续的存储单元中，即以数组作为存储结构。在这种情况下，排序过程是对数据本身进行物理重排，即通过关键字之间的比较判断，将数据移到合适的位置。另一种存储方式是以链表作为存储结构，排序过程中无须移动数据，仅需修改指针即可。两种存储方式如图5.3.1所示。

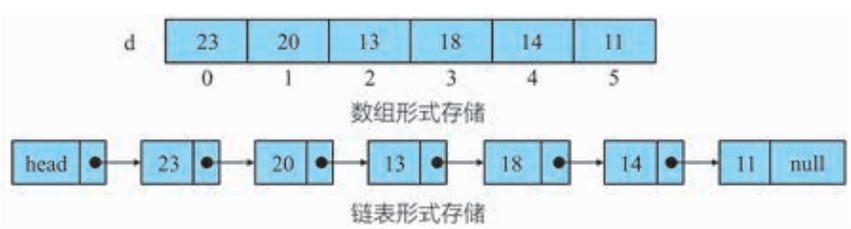


图5.3.1　数据存储情况

以数组为例，每个数组元素都对应存储一个数据。例如，存储在数组元素d[0]中的数据是23，d[1]中存储的是20，等等。

如果对数组d中的6个数据按升序进行排序，即调整数组d中所有数据的存储位置，使最小的数据存储在d[0]中，次小的数据存储在d[1]中……最大的数据存储在d[5]中。数组d中的所有数据满足：d[0]≤d[1]≤d[2]≤d[3]≤d[4]≤d[5]。

这里两个数组元素的比较：d[i]≤d[j] (i=0,1,…,5; j=0,1,…,5)，指的是d[i]中的数据小于或等于d[j]中的数据。

对数组d按升序进行排序后，数据的存储情况如图5.3.2所示。

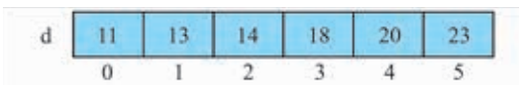


图5.3.2　排序后的数据存储情况

5.3.2 常见的排序算法

实现排序有多种方式，Python中提供的排序函数能满足绝大多数需求，但还是有必要介绍一些基本的排序算法，如冒泡排序。

拓展链接

Python中的排序函数

Python中，对列表进行排序的方法有两种：一种是列表自带的sort方法，只适用于列表，直接对列表进行排序，不会产生新的序列；另外一种是内建函数sorted方法，返回一个新的序列，而原来的序列依然存在。两者的使用方法如下所示：

```
>>> a=[5,7,6,3,4,1,2]
>>> b=sorted(a)
>>> print(a)
[5, 7, 6, 3, 4, 1, 2]
>>> print(b)
[1, 2, 3, 4, 5, 6, 7]
>>>a.sort()
>>> print(a)
[1, 2, 3, 4, 5, 6, 7]
>>>a.sort(reverse=True)                    # reverse=True实现降序排序
>>> print(a)
[7, 6, 5, 4, 3, 2, 1]
```

冒泡排序（Bubble Sort）是在一系列数据中对相邻两个数依次进行比较和调整，让较大的数“下沉（上冒）”，较小的数“上冒（下沉）”的一种排序技术。以升序为例，冒泡排序算法把待排序的n个元素的数组看成是垂直堆放的一列数据，对相邻两个数进行比较，将较小的数据换到上面的一个元素中（或将较大的数据换到下面的一个元素中）。重复这一过程，直到处理完最后两个元素中的数据，称为一遍加工。当第一遍加工完成时，最小的数据已经“上浮”到第一个元素的位置（或最大的数据“下沉”到最后一个元素的位置）。然后对余下的n−1个元素重复上述处理过程，直至最后进行余下两个数据的比较和交换。由于每一遍加工都是将本遍最小的元素像气泡一样“上浮”至本遍的顶端位置（或将本遍最大的元素“下沉”至本遍的底端位置），故称为冒泡排序。

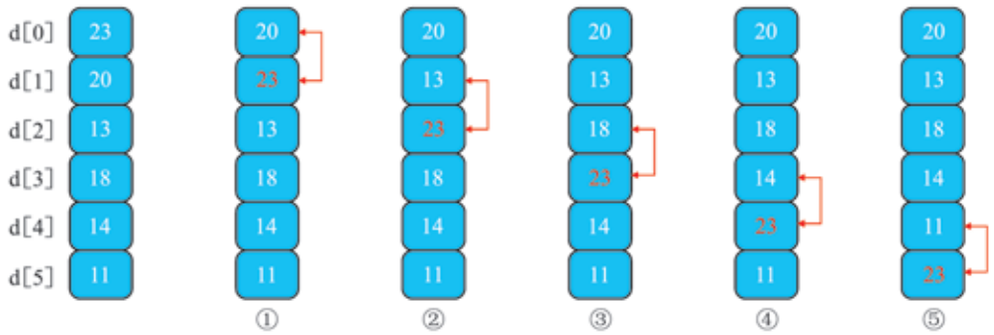


图5.3.3　冒泡排序算法对数组d的第一遍加工过程

如图5.3.3所示，第一遍加工完毕，最大值23“下沉”至数组最后一个元素的位置。即第一遍加工，先比较相邻两元素d[0]与d[1]，不符合所要求的顺序，则交换两者的位置；再比较调整d[1]与d[2]……最后比较调整d[4]与d[5]。

若要排序的数有n个，则需要n−1遍加工。第i遍加工中，从第一个数开始，相邻两数比较，若反序则交换两者的位置，直到第n+1−i个数为止。第一个数与第二个数比较，第二个数与第三个数比较……第n−i个与第n+1−i个比较，共比较n−i次。此时第n+1−i个位置上的数已经按要求排好，所以不参加以后的比较和交换操作。

对n个元素的数组，用冒泡排序算法进行排序时，共需比较：

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

其时间复杂度为$O(n^2)$。

排序过程中，按下面的方式使用变量i和j:

i：记录正进行的处理遍数，第1遍处理时值为1，第2遍处理时值为2，以此类推。

j：记录当前数组元素的下标。每遍处理过程中，j值总是从第一个数组元素的下标值0开始，按每次加1的方式，直至j=n−i−1为止。每当j取定一个值后，当前数组元素d[j]将与它的后一个元素d[j+1]进行比较，若d[j] ＞d[j+1]，则互换这两个数组元素中的数据。

如图5.3.4所示是6个元素进行5遍加工的过程。

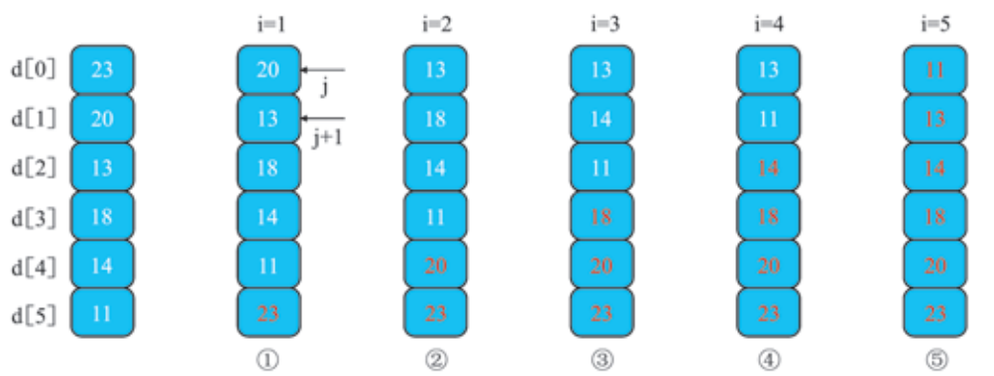


图5.3.4　冒泡排序算法对数组d的5遍加工过程

对规模为n的数组d，冒泡排序的算法流程图如图5.3.5所示。

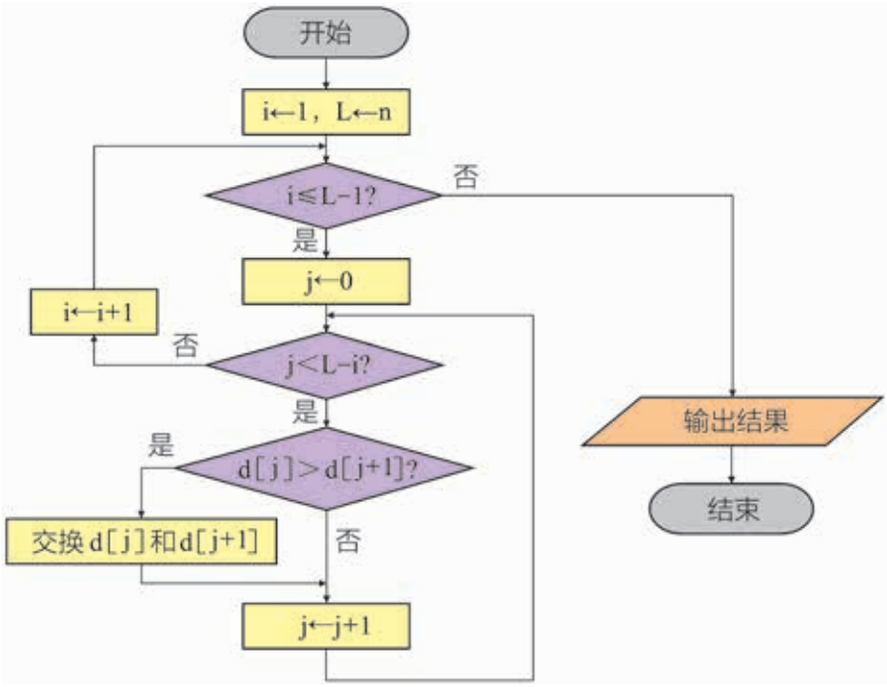


图5.3.5　冒泡排序的算法流程图

代码实现：

```
def bubble_sort(d):
    length=len(d)
    #序列长度为length，需要执行length–1遍加工
    for i in range(1,length):
        for j in range(0,length–i):
            if d[j]>d[j+1]:
                temp=d[j]
                d[j]=d[j+1]
                d[j+1]=temp
```

问题与讨论

冒泡排序算法的每一遍加工中，最小的元素“上浮”和最大的元素“下沉”这两种方式在程序实现时有何区别？

常见的排序算法还有选择排序、插入排序、快速排序、堆排序、归并排序、桶排序等。在选取排序算法时，可以根据待排序数据自身的特点来选择相应的算法。

5.3.3 排序算法的应用

排序能使数据有序，而有序的数据更加便于使用。下面，通过一个实例来体验排序算法在实际生活中的应用。

例1 奥运排行榜

每届奥运会各大媒体都会公布一个排行榜，但每个排行榜可能略有不同，如有的是按金牌总数排列的“金牌榜”，有的是按奖牌总数排列的“奖牌榜”，甚至还有“国民人均奖牌榜”，等等。现在，以2008年北京奥运会的奖牌数为例，部分数据（数据统计时间为2008年8月25日）如表5.3.1所示。请你编写一个程序，编制金牌排行榜。

表5.3.1 2008年北京奥运会奖牌数表

编号	国家/地区	人口数量（万）	金牌	银牌	铜牌	总数
8	中国	136407	51	21	28	100
20	印度	130420	1	0	2	3
53	美国	32262	36	38	36	110
21	印度尼西亚	26110	1	1	3	5
13	巴西	20529	3	4	8	15
23	尼日利亚	18231	0	1	3	4
52	俄罗斯	14253	23	21	28	72
6	日本	12703	9	6	10	25

（1）抽象与建模

从表中的数据可以看出，每个国家的信息是一条记录，包括编号、国家/地区、人口数量、各奖牌数等数据项。编制金牌排行榜，要按排序关键字“金牌”降序排序。排序过程中若要交换，则要将待交换的两条记录整体进行交换。

（2）设计算法与数据结构

对于每个国家，需要记录其一条完整的记录信息，排序之前要将所有国家的信息存储起来，可以选用数组来实现。有两种方案：一是采用7个一维数组按列存储，即每个数组分别存储每个国家的编号、国家/地区、人口数量、各奖牌数等，如定义b数组存储表中8个国家的金牌数量，其对应的值为[51,1,36,1,3,0,23,9]；二是采用1个一维数组按行存储，每个数组元素对应某个国家的一条记录信息，如[8,"中国",136407,51,21,28,100]对应中国的相关信息。采用不同的存储方式，排序时数据的交换方式也有不同：

①若采用7个一维数组按列存储，排序过程中，两条记录的对应数据项都要相应交换，即要考虑7个一维数组的操作。

②若采用1个一维数组按行存储，排序中的数据交换可对整条记录进行交换操作。

（3）编写程序

若存储结构采用1个一维数组，以金牌数为关键字进行降序排序，程序如下：

```
import csv
#数据读入
csvFile=open("jp.csv", "r")                    #打开相应数据文件
reader=csv.reader(csvFile)                     #建立一个读入数据的对象
a=[]
for item in reader:
    a.append(item)
csvFile.close()
#排序
for i in range(1,len(a)–1):                    #排序不包含第一行数据
    for j in range(1,len(a)–i):
        if int(a[j][3])<int(a[j+1][3]):
            temp=a[j]
            a[j]=a[j+1]
            a[j+1]=temp
#数据写入
csvFile2=open('jp2.csv','w', newline='')
writer=csv.writer(csvFile2, dialect='excel')
m=len(a)
for i in range(m):
    writer.writerow(a[i])
csvFile2.close()
```

程序中的“a[j][3]”表示某条记录索引位置为3的信息，即某个国家的金牌数。而“a[j]=a[j+1]”表示整条记录的所有信息的赋值。

问题与讨论

从待排序数据的规模、对稳定性的要求以及数据结构等角度，谈谈排序算法应该如何选取。

实践与体验

K–近邻分类算法初步

怎样让计算机根据照片来辨别鲜花的品种、根据内容来过滤垃圾邮件、根据镜头内容来区分电影的类别？这类操作可利用K–近邻（K-Nearest Neighbors, KNN）分类算法来实现，该算法的核心思想是：一个样本在特征空间中的k个最相邻的样本中的大多数属于某一个类别，则该样本也属于这个类别。如图5.3.6

所示，在二维坐标中有六个点，分别是a1(1,20)，a2(1,16)，a3(1,3)，b1(25,15)，b2(25,10)，b3(25,2)。其中，a1，a2，a3为A类，b1，b2，b3为B类。若要判定c(14,1)点属于哪个类别，可以通过如下步骤来实现：

1. 计算c点和其余所有点的距离（两点间的距离计算公式：$d_{12}=\sqrt{(x_1-x_2)^2+(y_1-y_2)^2}$）。

2. 按距离升序排序。

3. 取前k（样本的k个最近邻居，k小于等于样本个数）个距离。

4. 统计出在前k个距离中，出现频次最多的类别，则c点就属于该类别。

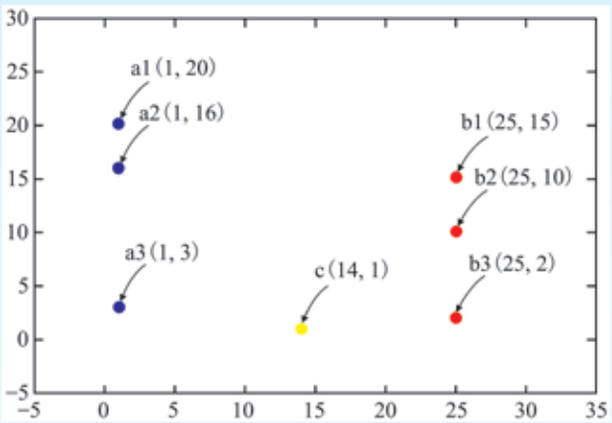


图5.3.6　二维坐标中点的分布情况

实践内容：

编程判定c点的类别。

实践步骤：

1. 读入如图5.3.7所示的样本数据（CSV文件类型），输入c点坐标值(14,1)和k（k≤6）的值。

2. 按照K-近邻分类算法编写程序。

3. 调试程序，输出c点的类别。

	A	B	C	D
1	序号	x值	y值	类别
2	1	1	20	A
3	2	1	16	A
4	3	1	3	A
5	4	25	15	B
6	5	25	10	B
7	6	25	2	B

图5.3.7　样本数据

结果呈现：

当k值为4时，c点属于B类。

思考与练习

1. 结合“奥运排行榜”的例子，分别编写程序满足下列要求：

（1）若数据以7个一维数组的形式存储。

（2）若将排序关键字改为“国民人均奖牌数”，并按升序排序。

2. 选择排序算法是在参加待排序序列的所有元素中找出最小（或最大）数据的元素，使它与第一个元素中的数据相互交换位置。然后再在余下的元素中找出最小（或最大）数据的元素，与第二个元素中的数据相互交换位置。以此类推，直到所有元素成为一个有序的序列。请根据该排序算法的思想，实现对数据序列[23,20,13,18,14,11,98,4]的升序排列。

5.4 数据查找

在日常生活中，人们经常会进行各种查找操作，如在手机上查找公交车的线路、在电子词典中查找某个单词的读音和含义、在音乐网站上查找自己喜欢的音乐以及利用搜索引擎在浩瀚的信息海洋中查找需要的信息等。

5.4.1 查找

查找（Search）又称检索，计算机根据所给条件查找出满足条件的对象，即在存储的一批数据内寻找出一个特定的数据，或者确定在该批数据内是否存在这样的数据。若没有找到满足条件的对象，则返回特定值，表明查找失败；若查找到满足条件的对象，则表明查找成功，一般要求返回该对象的存储位置或对象值本身。通常，程序将按照查找的结果（找到或未找到）来决定接着应执行后面哪一个计算步骤。如超市购物付款时，当收银员扫描一件货物的条形码时，计算机需要在几万条不同的记录中寻找这件商品，然后显示相应的商品名称和价格。

5.4.2 常见的查找算法

查找算法有许多种，本节中介绍两种查找算法：顺序查找和二分查找。

1. 顺序查找

顺序查找（Sequential Search）又称线性查找，从顺序表的一端开始，依次将每个元素的关键字与给定值key（查找键）进行比较。若某个元素的关键字等于key，则表明查找成功；若所有元素都比较完毕仍找不到，则表明查找失败。

以“在成绩查询系统中查找某学号”为例，介绍顺序查找的处理过程。假定某学习小组8个学生的学号数据存储在数组d中，要查询的学生学号（查找键）已经存储在变量key中。

从数组d的第1个元素d[0]开始，依次判断各元素的值是否与查找键key的值相等。若某个数组元素d[i]的值等于key，则结束处理（找到了指定的数据）；若找遍了所有8个元素，无任何元素的值等于key，则结束处理（未找到指定的数据）。

图5.4.1所示是在规模为8的数组d中，分别按顺序查找算法寻找数据18和15的情况，处理过程中找到的第4个数组元素d[3]中的数据与18相等，表示8个数据中存在值为18的元素；而若key为15时，查找完所有数据仍未找到，表示8个数据中不存在值为15的元素。

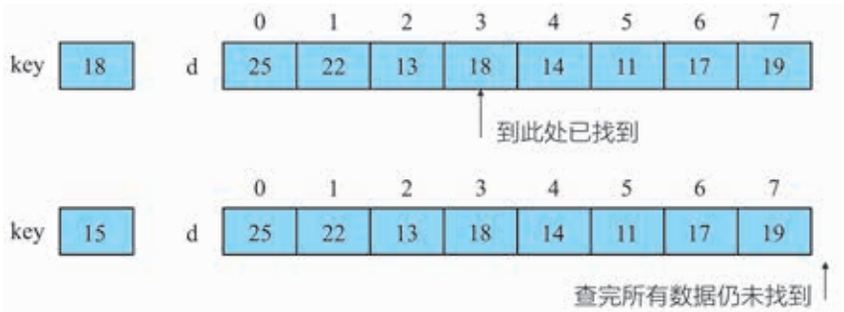


图5.4.1　顺序查找过程实例

顺序查找将待查找的数值与序列中的数逐个进行比较，直到找出与给定数值相同的数为止，其算法简单，时间复杂度为O(n)。

图5.4.2所示是在规模为n的数组d中进行顺序查找的流程图。

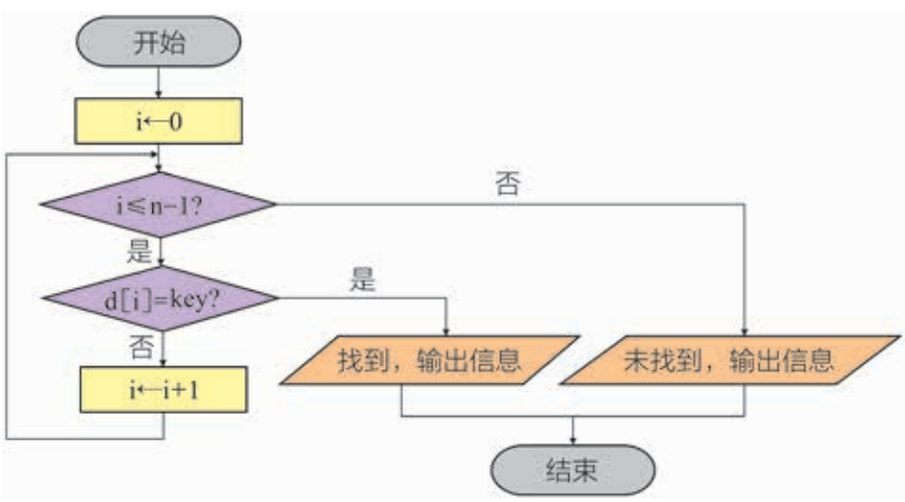


图5.4.2　顺序查找算法流程图

实现此算法的Python程序如下：

```
d=[25, 22, 13, 18, 14, 11, 17, 19]
key=18
flag=False
length=len(d)
for i in range(length):
    if d[i] == key:
        flag=True
        break
if flag==True:
    print("查找成功！")
else:
    print("未找到")
```

顺序查找算法也可以写成函数的形式，如下所示：

```
def seq_search(s, a):
    length=len(s)
    flag=False
    for i in range(length):
        if s[i] == a:
            flag=True
            break
    if flag==True:
        return i
    else:
        return False

d=[25, 22, 13, 18, 14, 11, 17, 19]
key=15
result=seq_search(d, key)
print(result)
```

2. 二分查找

二分查找（Binary Search）又称折半查找，对分查找。它是一种效率很高的查找方法，但被查找的数据序列必须是有序的。

二分查找首先将查找键与有序数组内处于中间位置的元素进行比较，如果中间位置上的元素的数值与查找键不同，根据数组元素的有序性，就可确定在数组的前半部分还是后半部分继续查找；在新确定的范围内，继续按上述方法进行查找，直到获得最终结果。

在数组中的数据是有序的，若是升序的，是指下标越小的数组元素中存储的数据也越小，降序则相反。为简单起见，设数组d中存储了n个互不相同的数据，并且数组d中的数据是升序的，则应有：d[0]＜d[1]＜…＜d[n−2]＜d[n−1]。

使用两个变量i和j分别记录查找范围内的第一个数组元素和最后一个数组元素的下标。开始时，整个数组中的所有n个元素都在查找范围内，即变量i的初值为0，j的初值为n−1，用(i, j)表示查找范围从d[i]起到d[j]为止。

二分查找的基本方法是：在查找范围(i, j)内，可以利用数据的有序性，而不必逐个地进行查找。

①首先确定该区间的中点位置：m=⌊(i+j)/2⌋（m表示小于等于$\frac{i+j}{2}$的最大整数）。

②然后将查找键key与d[m]比较，结果必然是如下三种情况之一：

（a）key＜d[m]，查找键小于中点d[m]处的数据。因为数组d中的数据递增，可以确定在(m, j)内不可能存在值为key的数据，所以只需在新的范围(i, m−1)中继续查找。

（b）key=d[m]，找到了需要的数据。

（c）key＞d[m]，由与（a）相同的理由，只需在新的范围(m+1, j)中继续查找。

在通过一次比较后，新的查找范围不会超过上一次查找范围的一半。

以规模为11的数组d为例，观察二分查找的过程。在数组d的11个元素中，已按增序存储了11个数据，要寻找的数据为12（已存储在变量key中）。

查找过程中，查找范围的变化情况如图5.4.3所示。最初，数组d中的11个元素（从

d[0]到d[10]）都在查找范围内，中点处的数组元素是d[5]。通过第1次比较（d[5]与key的比较）后发现，key的值比中点处的数据小，在数组d的右半部分（从d[6]到d[10]的范围）内不存在要找的数据，只需在数组d的左半部分（从d[0]到d[4]的范围）进行下一次查找。第2次比较时，中点处的数组元素是d[2]。比较后发现，第3次查找的范围为d[0]到d[1]，中点处的数组元素是d[0]。此时，key的值比中点处的数据大，第4次的查找范围只有一个元素，即d[1]，而d[1]的值与key的值相等，找到指定的数据。

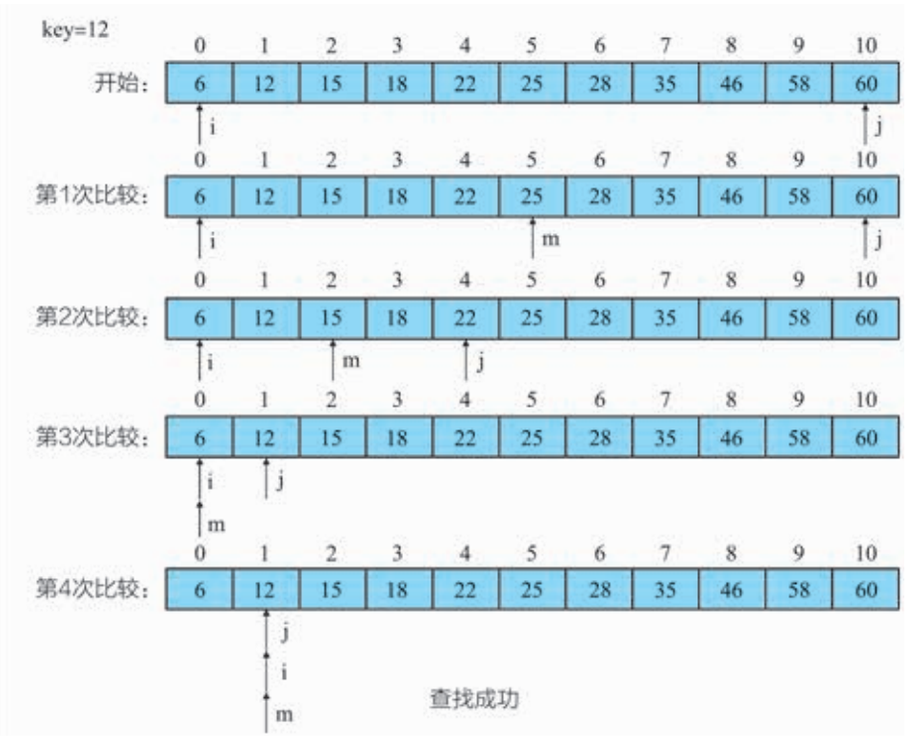


图5.4.3　二分查找过程实例

图5.4.4所示是在规模为n的数组d中进行二分查找的流程图。

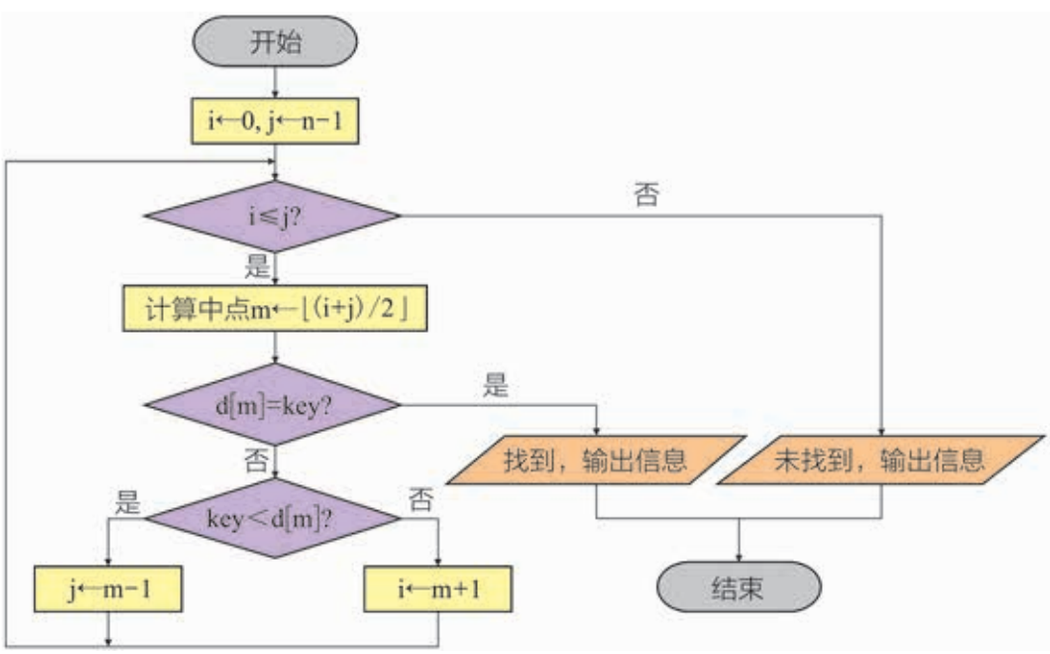


图5.4.4　二分查找算法流程图

实现此算法的Python程序如下：

```
key=12
d=[6,12,15,18,22,25,28,35,46,58,60]
f=False
# i和j定义子数组的边界，一开始搜索的是整个数组
i=0
j=len(d)-1
while i <= j:
    m=(i+j) //2
    if d[m] == key:
        f=True
        b=m
        break
    if key < d[m]:                    # 到左边去找
        j=m-1
    else:                             # 到右边去找
        i=m + 1
if f==True:
    print("查找成功！第"+str(b)+"个")
else:
    print("没有找到！")
```

拓展链接

二分查找算法的递归实现

二分查找过程中的每一次判断都是将需要查找的值和数组的中间值进行不断的比较，直到找到或找遍整个序列。因此，二分查找算法可利用递归来实现，程序如下：

```
def bsearch(s,array):
    if len(array) == 0:
        print("未找到！")
        return False
    mid=(len(array)-1) //2
    if array[mid] == s:
        print("找到了！")
        return True
    elif s < array[mid]:
        return bsearch(s, array[:mid])
    else:
        return bsearch(s, array[mid+1:])
key=12
d=[6,12,15,18,22,25,28,35,46,58,60]
print(bsearch(key,d))
```

二分查找过程中，在每次关键字比较时，如果不匹配，那么根据匹配结果将查找表一分为二，排除没有包含查找键的那一半，然后在可能含有查找键的那一半中继续二分查找，直至找到查找键或找遍整个数组。处理思想是将一个大范围的查找问题转化为一个与原问题相似的查找范围较小的查找问题，并且查找能在一定条件下停止。这个思想符合递归算法的思想。

问题与讨论

若查找对象采用链表结构，能否适用二分查找？

二分查找过程可用一棵二叉树来描述，树中的每个根节点对应当前查找区间的中点元素，它的左子树和右子树分别对应该区间的左子表和右子表，如图5.4.5所示。通常把此树称为二分查找的判定树。

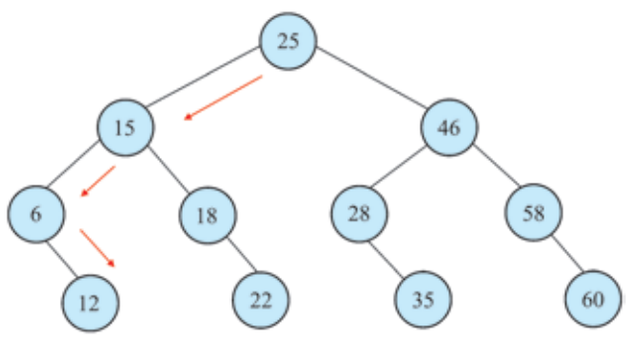


图5.4.5　二分查找的判定树实例

在有序表上二分查找一个关键字等于key的元素时，对应着判定树中从根节点到待查节点的一条路径，与关键字进行比较的次数就等于该路径上的节点数，或者说等于待查节点的层数。如上例中，查找key为12的元素时，从根节点到待查节点的一条路径为25→15→6→12，比较次数为4次。通过观察可知，在n个元素排序的顺序表里，某一次查找过程中，所做比较次数不超过判定树的高度加1，即$\lfloor \log_2 n \rfloor +1$。

由于二分查找在有序表上进行，所以其对应的判定树就是一棵二叉排序树。

拓展链接

二叉排序树

二叉排序树也称为二叉查找树，这种结构的二叉树既能实现排序功能，也能实现查找功能。

一、排序

二叉排序树的排序功能主要通过二叉树的建立和遍历过程来实现，其在建立过程中要始终满足如下性质：

1. 若它的左子树不为空，则左子树上所有节点的值均小于它的根节点的值。
2. 若它的右子树不为空，则右子树上所有节点的值均大于它的根节点的值。
3. 它的左右子树也分别为二叉排序树。

一组数据“23，20，13，18，14，11”建立的二叉排序树如图5.4.6所示，对此二叉树进行中序遍历时，结果为从小到大的有序序列：11，13，14，18，20，23。

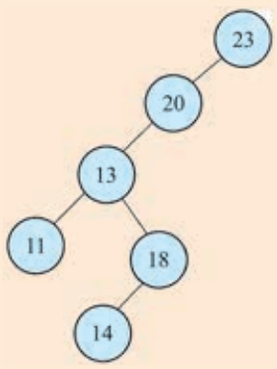


图5.4.6　二叉排序树

二、查找

二叉查找树的查找过程为：首先将给定值和根节点的关键字比较，若相等，则查找成功；若不相等，则根据给定值和根节点关键字之间的大小关系，在左子树或右子树中继续进行查找。若查到为空树时，说明该树中没有待查记录，故查找不成功。在如图5.4.6所示的二叉查找树中查找关键字key为14的节点，则查找过程为：

1. 先将数据14与根节点中的数据23比较，由于14小于23，则在左子树中继续查找。
2. 将数据14与节点数据20比较，由于14小于20，则在左子树中继续查找。
3. 将数据14与节点数据13比较，由于14大于13，则在右子树中继续查找。
4. 同样的方法继续查找，当找到节点数据14时，与要查找的数据相等，则查找成功。

上例中，查找key为14的节点时，从根节点到待查节点的一条路径为23→20→13→18→14，比较次数为5次。因此，在二叉查找树上查找其关键字等于给定值的节点的过程，恰是走了一条从根节点到该节点的路径，和给定值比较的关键字个数等于路径长度加1。因此，和二分查找类似，与给定值比较的关键字个数不超过树的高度加1。

5.4.3　查找算法的应用

在计算机应用中，查找是常用的基本运算，一般先通过查找算法找到某个对象，然后对其进行相应操作。如乘客乘公交车刷卡时，系统需要在几万条甚至几十万条公交卡信息中找到乘客的卡号，然后进行扣款操作。

例2　航空公司VIP会员积分查询

不少航空公司都会提供优惠的会员服务，当某会员飞行里程累积达到一定数量后，可以使用里程积分兑换奖励机票或奖励升舱等服务。现给定某航空公司部分VIP会员的飞行里程、积分等信息，如表5.4.1所示，要求实现根据VIP号码快速查询会员积分的功能。

表5.4.1　某航空公司的部分会员信息表

VIP号	姓名	飞行里程（km）	积分
600214	韩江辉	16801	519
601278	蒋志来	5321	78
600815	李亚东	28745	436

续表

VIP号	姓名	飞行里程（km）	积分
607854	王庆生	1861	39
605719	李燕	7493	138
603532	王晓燕	6875	102
600101	郑煜明	14253	236
600087	蔡佳宁	112703	958

（1）抽象与建模

从表中的数据可以看出，每个会员的信息是一条记录，包括VIP号、姓名、飞行里程、积分等数据项。要显示某个会员的积分信息，先得从多条记录中查找到该会员的记录，如下所示：

××××××	×××	×××××	×××

若用a[i]表示该条记录，则该会员的积分可采用以下形式表示：

a[i][3]（表示该条记录的第4个数据项的值）

（2）设计算法与数据结构

对表格数据可采用4个一维数组按列或1个一维数组按行来存储。要显示某个会员的积分，先要从多条会员信息的数据中找到该会员。查找可采用顺序查找算法或二分查找算法。从算法的时间复杂度方面考虑，二分查找算法的效率高于顺序查找算法，但若采用二分查找算法，被查找的数据序列必须是有序的，即要按VIP号为关键字进行排序。

（3）编写程序

假如数据以1个一维数组按行来存储，利用二分查找算法查找，程序如下：

```
import csv
#数据读入
csvFile=open("vip.csv", "r")
reader=csv.reader(csvFile)
a=[]
for item in reader:
    a.append(item)
csvFile.close()
#排序
def bubble_sort(d):
    for i in range(1,len(d)):
        for j in range(1,len(d)-i):
            if int(d[j][0])>int(d[j+1][0]):
                temp=d[j]
                d[j]=d[j+1]
                d[j+1]=temp
#二分查找
```

```
def bsearch(s,array):
    i=1                          #查找范围不包含第一行数据
    j=len(array)-1
    while i <= j:
        m=(i+j) //2
        if int(array[m][0]) ==s:
            return m
        if s <int(array[m][0]):
            j=m-1
        else:
            i=m+1
    return -1                    #未找到返回-1

bubble_sort(a)
key=int(input('请输入要查询的VIP号: '))
m=bsearch(key,a)
if m !=-1:
    print(a[m][1],"先生/女士，您的积分为: ",a[m][3])
else:
    print('找不到VIP号对应的用户信息！')
```

运行上面的程序，当输入VIP编号“600815”时，程序输出“李亚东 先生/女士，您的积分为：436”的信息。

实践与体验

比较查找程序的效率

数据查找，对顺序查找和二分查找而言，理论上，它们的时间复杂度是不一样的，分别为O(n)和$O(\log_2 n)$，反映在实际的程序中，它们在运行时间上是有差别的。因此，可通过计算程序的运行时间来比较程序的性能。

实践内容：

对同一个数据集分别采用顺序查找和二分查找算法查找同一个查找键，分别记录程序运行的时间，可分以下多种情况进行测试：

1. 数据集和查找键都是随机生成的。

2. 数据集是固定且按某个关键字有序的，查找键分别取关键字的第1个、中间值和最后1个。

实践步骤：

1. Python中测试程序运行时间的方法如下：

```
import time
start=time.clock()
run_fun()
end=time.clock()
print(str(end-start))
```

其中，run_fun() 函数就是要测试的程序代码。

2. 分情况测试，并将测试结果记录在下表中。

数据集	查找键	查找算法	运行时间
随机	随机	顺序	
		二分	
固定且有序	第 1 个	顺序	
		二分	
	中间值	顺序	
		二分	
	最后 1 个	顺序	
		二分	

结果呈现：

根据测试结果，比较顺序查找和二分查找程序的效率。

思考与练习

在“航空公司 VIP 会员积分查询”例子中，将其中的二分查找改成顺序查找，并上机实践。

巩固与提高

1. 假设现在需要统计全世界的人口，希望在1天之内完成工作。如果采用的算法具有线性复杂度，多快的计算机就足以满足工作需求？如果所用算法具有平方复杂度呢？用“神威·太湖之光”超级计算机大约能处理什么复杂度的算法？

2. 有一段楼梯有n级台阶，规定每一步只能跨1级或2级，要登上第n级台阶有几种不同的走法？请分别用迭代和递归的算法求解。

3. 给定一个包含n个元素的列表，要找出最小两个值的索引。请用至少两种算法编程实现。

4. 最优服务次序问题：设有n 个顾客同时等待同一项服务，顾客i 需要的服务时间为t_i（n≥i≥1），应如何安排这n个顾客的服务次序才能使平均等待时间达到最小？平均等待时间是n个顾客等待服务时间的总和除以n。

5. 记录按关键字排列的有序表：6，13，20，25，34，56，64，78。采用二分查找，画出判定树，并给出查找关键字为13和55的记录过程。

6. 注册网站账号时，用户名常常要符合某些要求，比如不能含有英文的“;”“!”“~”这三个字符。写一个函数，读入用户输入的用户名，返回“用户名合法”或者“用户名不合法”。

7. 用二分查找实现开平方根函数squareroot(x,p)。x是被开方的数，假定输入的数都为非负整数，p是误差上限，输出一个浮点数结果。

项目挑战

无人机送快递的配送决策

在人工智能大发展的今天，无人机产业发展迅猛。比如，为了解决偏远地区人力配送较难、较慢的问题，很多快递公司已经开始使用无人机送货。

项目任务

有一家刚成立的快递公司，给你提供了一批快递配送的信息，需要你编程来编排无人机送货的顺序表。要求：每次只能送一个快递，急件需要优先送达，普通件的配送需要考虑客户的感受，即让所有的用户等待的时间和最短。

过程与建议

1. 分析存储数据

在接受该项目时，快递公司给你提供了如下表所示的配送信息：

编号	快递单号	经度	纬度	是否急件
6600410	479093420863	116.515367	39.908484	否
6600507	481613822030	116.495463	39.908789	否
6600389	487821623686	116.475794	39.9086051	是
6600458	488017459876	116.4601	39.908506	否
6600497	532477051004	116.45046	39.9085528	否
6600325	630885192575	116.435242	39.9086068	否
6600364	719547782864	116.418076	39.908283	否
…	…	…	…	…

同时，快递公司也表示，如因客户的临时要求，快递的配送等级可由原先的普通件变更为急件。因此，快递公司还会给你几个需要变更的快递编号，如下图所示。

×××× 快递公司快递配送等级变更单	
编号	变更内容
6600497	普通件→急件
6600364	普通件→急件
…	…

根据已知的数据和要求，分析存储数据至少要考虑以下几个问题：

（1）如何判断无人机和收货人之间的距离？

（2）如何判断配送的级别？

（3）如出现快递级别的更改，哪些操作将会受到影响？

2. 设计并选择数据结构和算法

根据已知数据及数据之间的关系，设计并选择解决该问题的数据结构和算法。

考虑数据结构和算法时要思考：

（1）读取存储文件中的数据采用何种数据结构，选择此数据结构的理由是什么？

（2）查找临时变更配送等级的快递，你采用了哪种算法？理由是什么？

（3）要满足使所有的用户等待的时间和最短这一要求，你采用了哪种算法？理由是什么？

3. 编写程序并测试

根据你所确定的数据结构和算法，编写并测试程序，考虑能否进一步优化程序。

4. 展示交流

为了总结你的项目实施过程以及成果，你需要从以下几个方面进行展示：

（1）项目目标实现情况。

（2）你的解决方案（确定的算法和数据结构），并阐述理由。

（3）目前存在的问题以及对项目进一步完善的思考。

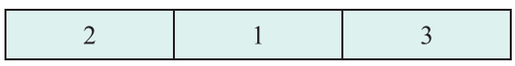
评价标准

请根据项目实施的过程、结果和交流效果，对自己完成项目的情况进行客观的评价，并思考后续完善的方向。将评价结果和完善方案填写在下面的表格中。

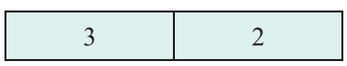
评价条目	说明	评分（1～10分）	评分主要依据阐述	后续完善方向
问题界定	通过抽象与建模确定的问题明确，且已知条件充分，结果能求解			
算法与数据结构	设计的数据结构和算法正确，且便于实现			
程序实现	程序已编写完成，且功能模块清晰，并能输出正确解			
项目拓展	有明确的目标，并在相关方面有了进一步的学习，形成了一定的思路，甚至实现了部分功能			

拓展项目

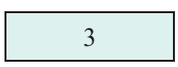
1. 将同花色的13张扑克（1—10，J为11，Q为2，K为13）牌叠在一起，经过下列操作：每次将这叠扑克牌最上面的一张移到最下面后，移出此时扑克牌最上面的一张。重复这个操作，直到这叠扑克牌全部移出时停止操作，使依次移出的扑克牌是由小到大的排列。编程求出原扑克牌的叠放次序。以1，2，3三张扑克牌为例，要使依次移出的牌的顺序为1，2，3，则原叠放次序应为2，1，3。



①将2移到扑克牌的末尾（3的后面），将1移出，剩下：



②将3移到扑克牌的末尾（2的后面），将2移出，剩下：



③ 最后将3移出。

2. 查找相关资料，实现一种简单的搜索引擎的功能。

第六章 6

大数据时代数据的组织

进入大数据时代，数据的量和计算量越来越大，数据的非结构化程度越来越高，基于传统数据结构的数据组织和存储方式已不能满足数据业务的实际需要。因此，需要根据问题特点去改造既有的数据结构，运用新的数据组织方式来设计算法解决问题。如在电商服务平台中，由于商品数量庞大以及各种不同类型数据的出现，仅靠传统的文件组织和数据库等方式已不能满足海量而复杂的数据组织和存储的需求。而在POI（Point of Interest）数据业务中，也同样需要人们采用新的方法和技术去组织数据、存储数据和处理数据。

问题与挑战

- 一个电商平台上的同类商品数量往往多达几十万，每个商品又有名称、价格、销量、地区、评价等各种属性。每时每刻，可能都会有成千上万名买家对商品进行各种查询操作。用怎样的数据结构来组织这些商品信息（这些商品信息可能经常更新），使得平台能为用户提供快捷的查询结果？

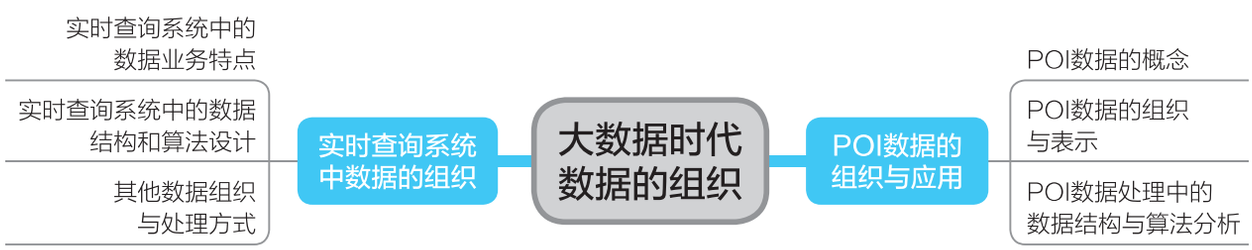
- 手机用户有时希望能实时查询话费的使用情况。这对通信运营商而言，意味着需要在瞬间为提出申请的上千名甚至更多的用户提供实时的查询结果。同时，每个手机用户的通信资费标准是多重的。如何组织手机用户的数据，才能使通信运营商为用户提供实时的费用统计和查询服务？

- 导航技术的应用已经成为现代生活的重要组成部分，它为我们的生活提供了极大的便利。通过导航软件可以搜索自己所在的位置以及附近的银行网点、餐馆等信息。那么，导航软件是如何有效地组织这些信息数据，并快速检索到它们的？

学习目标

1. 感受数据结构设计过程中的迭代思想。
2. 了解大数据处理过程中常见的数据组织与处理方式。
3. 了解数据业务中，数据进行分类、整理等组织工作的必要性。

内容总览



6.1 实时查询系统中数据的组织

人们经常会提出实时查询各种信息的请求，如电信业务中实时话费与剩余网络流量的查询、网购用户对特定商品的各种信息查询等。面对这种实时访问，系统必须在瞬间给出结果并呈现给用户。

大数据背景下，全部数据的组织、存储和处理，仅凭单个服务器和数据库的数据组织与存储方式，无论从存储容量还是处理速度上都不能满足实际应用的需求。此时，采用分布式存储技术，将所有数据分别保存在不同的服务器中，需要时从中提取并进行合并，就可以满足海量数据的存储与处理需求。

拓展链接

分布式存储系统

分布式存储系统利用分布在不同物理位置的服务器来分担系统存储任务，既能提高数据存储的安全性，又能提升系统数据访问的速度，同时也具有较好的可扩展性。当用户提出访问请求时，系统根据元数据服务器(进行数据访问索引的服务器) 将访问定位到目标数据的服务器上。

6.1.1 实时查询系统中的数据业务特点

用户在网购平台购买商品时通常对一类商品进行查询，通过比较做出购买选择。如图6.1.1所示，用户可能会从商品的“人气”“销量”“信用”“价格”等各个方面进行排序后实施查询，网站就必须快速地根据用户选择的查询要求进行商品的排序并呈现排序结果。在上述商品查询过程中，数据业务体现出下列特殊性。

图6.1.1　网络购物中的商品查询

①能实现上千个请求的实时响应。网站应能做到“即点即现”，即当某用户选择一种查询方式后，系统能马上呈现结果，而且同一时刻可能有上千名用户提出了查询请求。

②支持后续商品信息的更改。当用户选择了某种查询后，可能马上又有另一名用户提出其他方式的查询请求，而此时恰好网站中更改了某个商品的属性，也或者是删除、增加了商品，那么这个修改信息应该体现在最新的查询结果中。

面对这种查询业务，如果直接从数据库中提取查询结果，特别是在上千名用户同时

发出请求的情况下，系统需频繁地读取计算机硬盘并访问数据库，一方面将增加系统的负担，甚至在访问量增大的瞬间出现系统崩溃；另一方面，会造成用户较长的等待时间，结果导致用户的流失。

为了减轻磁盘数据库访问的负担，可事先将数据库中的商品信息读取出来并保存在内存中，这样用户的查询就能直接面对内存进行，大大提高数据读取的速度。

6.1.2 实时查询系统中的数据结构和算法设计

当数据从磁盘数据库读取出来并保存到内存中时，需要考虑用某种数据结构来组织并存储。这种数据结构既能体现数据之间的逻辑关系，又能为后续的查询提供算法上的支持。

1. 基于数据间线性关系的数据结构设计

（1）数组

考虑到各种商品在排序时呈现出的线性特征，一种方案是采用数组，能比较直接地表示商品之间按照某种属性呈现的有序线性关系，如按照销量排序的商品序列。

数据从数据库中读取到数组后，可事先按照各个属性（如人气、销量、信用、价格等）进行排序并分类存储，当用户发出相应的请求时，系统就从已排序数组中选择符合用户查询要求的内容呈现给用户。基于这种数据结构的问题主要出现在后续的商品信息维护阶段，首先是当商家增加了新商品时，系统需要在数组中插入一个新数据并维持数组元素继续有序。这里涉及两个主要操作：一是查找，即在一个有序序列中查找新增元素的插入位置，可以采用二分查找算法，时间复杂度为$O(\log_2 n)$（n表示数组元素的总个数），速度比较快；二是插入，即在找到可以插入的位置x后，将新元素插入到找到的位置x中，但必须先将位置x到n之间的所有元素往后移一位，为新元素空出位置，这个时间复杂度就比较大，为O(n)。当瞬间有上千名用户提出请求，同时进行上千个这样的处理时，时效性较差。

（2）链表

针对在数组中插入新元素时引起的数据移动低效的问题，可以考虑用链表代替数组。因为在一个链表中插入一个新元素，时间复杂度为O(1)，大大优于采用数组时O(n)的线性复杂度。

正如前面所述，插入操作主要涉及查找和插入两个关键操作，链表虽然在插入操作时能确保O(1)的时间复杂度，但在进行查找时（查找新元素的插入位置），却需要从链表的一端依次遍历查找，时间复杂度为O(n)。因此，采用链表来存储数据，虽然整体复杂度有所下降，但O(n)的复杂度还是达不到现实的需求。

问题与讨论

除了上面所述的数组和链表，是否还有其他的数据结构来解决上述问题的数据组织与存储？如果有，请简要描述用该数据结构组织数据并处理的算法，并尝试分析用该数据结构解决问题的时间复杂度。

2. 基于链表的数据结构和算法优化设计

各种数据结构是人们在用计算机解决问题的过程中，根据数据特点不断研究、探索所得到的。因此，随着数据业务的发展，既有的数据结构就得以不断改进，甚至演变出新的数据结构。

由于基于链表的处理，只是在查找时效率较低，而插入操作却完全能满足要求，所以可以在链表的基础上继续加以改进，以解决顺序查找导致的低效问题。主要考虑从查找角度来优化数据结构，这个改进可按以下思路进行。

（1）减少查找插入位置过程中的比较次数

基于链表数据结构的处理时间主要消耗在插入位置的查找中，因此，可着眼减少查找过程中的数据两两比较的次数来优化数据结构。

（2）借鉴二分查找算法的思想

二分查找算法之所以效率较高，首先是因为数据是有序的，其次是利用有序性进行跨区间、跳跃性的比较，从而避免低效的逐个依次比较。

实现的思路是首先将数据进行有序化处理，然后像二分查找一样确定比较的关键节点，根据新元素与关键节点的比较结果来高效地取舍剩余的查找区间。如图6.1.2所示，在原链表基础上增设一批关键节点（对每个节点采用抛硬币的方法确定是否放在关键节点中），就可以大大提高查找速度。如果要在原链表中插入元素6，原来的链表数据结构需要比较4次（从第一个节点开始，依次与1，3，4，10比较）。有了关键节点，就让新元素6依次与关键节点比较，只需3次即可找到6插入的位置（从第一个节点开始，依次与关键节点1，4，10比较），如图6.1.3所示。从整体效率分析，增设关键节点后的查找速度是原来的2倍（原链表中每个节点被作为上一层关键节点的概率是二分之一，所以原链表中的一半节点会出现在关键节点中）。

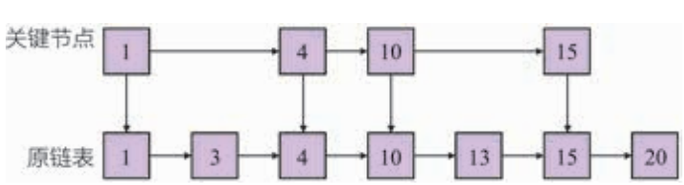


图6.1.2　在原链表基础上增设关键节点

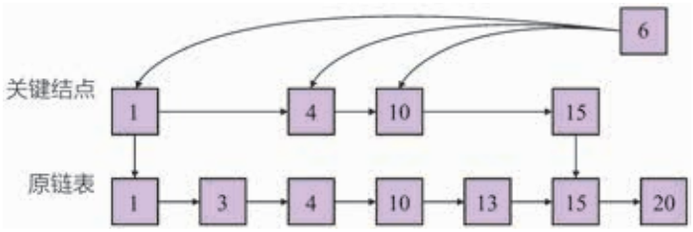


图6.1.3　通过关键节点只需比较3次

由此可见，关键节点起到一个索引表的作用，帮助算法快速定位到一个较小的插入区间，然后只需将索引位置对应到原链表，即可找到最终的插入位置，如图6.1.4所示。

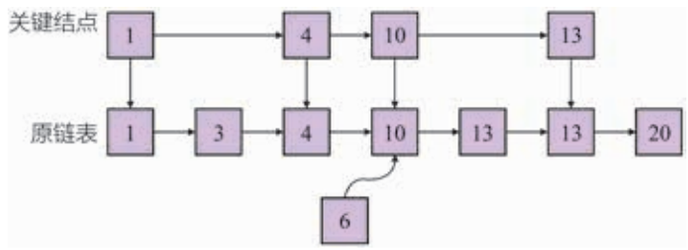


图6.1.4　通过由关键节点组成的索引表找到原链表中的插入位置

按照同样方法再增设一批关键节点，为原来的一级索引建立二级索引，则可以进一步将查找速度提升2倍（如图6.1.5）。

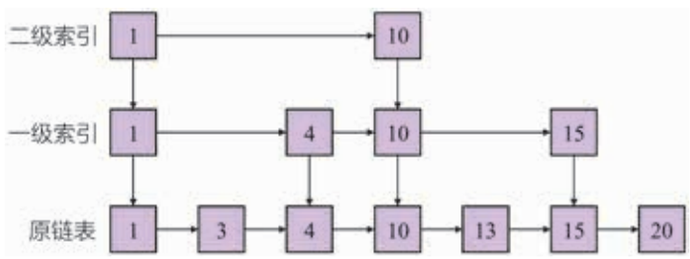


图6.1.5　在原来基础上建立二级索引

在二级索引的基础上，若要插入新元素6，则先从二级索引中经过2次比较确定一个大致区间，然后通过对应关系到达一级索引，最终到达原链表中可插入的位置，这样就可将在原一级索引中的顺序比较次数减少一半。如果数据比较多，还可以在此基础上继续增设三、四级索引，进一步提高查找的速度。总体来说，通过对链表进行改进的数据结构可以用$O(\log_2 n)$的时间复杂度实现查找，然后进行新元素的插入，并且保证新序列仍然有序。

因为数据序列在不断地变化，所以需要对关键节点进行动态的调整，即数据增加时增设关键节点，而数据减少时删除关键节点。对各级索引表中的关键节点进行增加和删除的实现方法如下：

①增设关键节点。随着数据元素的增加，原链表会变得越来越长，结果导致区间内查找效率降低。为了解决这个问题，可以考虑对新增元素基于“抛硬币”原则的选拔，以确定是否把新增元素提升为上一层的关键节点，并且逐层进行。若新增元素为24，通过“抛

硬币”发现需要提升为关键节点，则新的链表组织结构如图6.1.6所示。

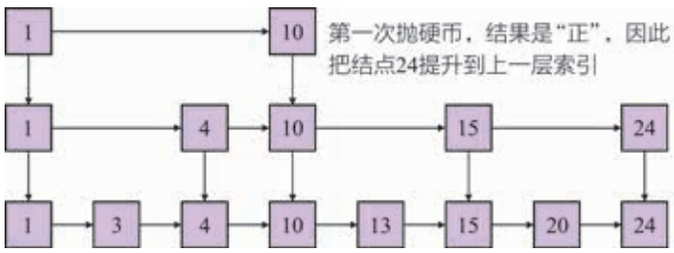


图6.1.6　将新增元素24所在节点提升为关键节点

若通过“抛硬币”发现不需要提升为关键节点，则新的链表组织结构如图6.1.7所示（新增元素只在原链表中出现）。

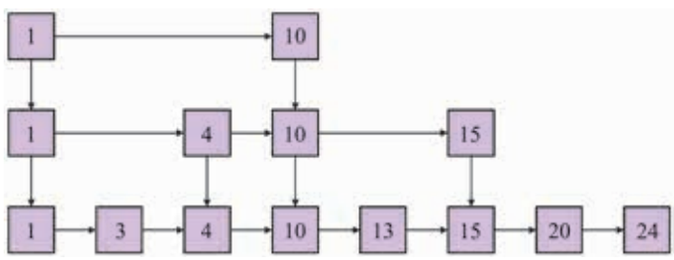


图6.1.7　新增元素24所在节点不提升为关键节点

②删除关键节点。随着原链表中数据元素的不断删除，各级索引中的关键节点也需要随之删除。删除时按照查找时的层次从上往下依次进行，每当找到对应的元素，就删除当前层的关键节点，直到最底层的原链表。若原来链表的状态如图6.1.8所示，现在要删除原链表中的元素10，则从顶层索引开始，依次往下删除各层的元素10。

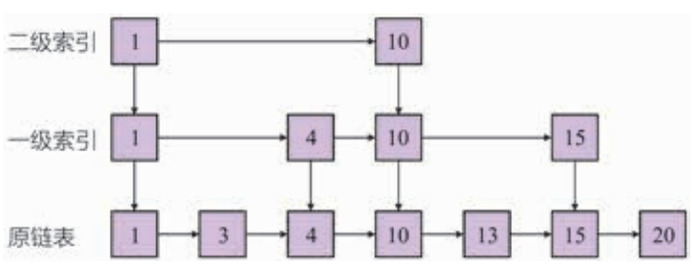


图6.1.8　删除元素10之前的链表状态

二级索引中删除元素10所在的节点后，因为只剩下一个关键节点，对于区间划分和提高查找效率没有任何意义，所以将剩下一个节点也删除，得到的最终状态如图6.1.9所示。

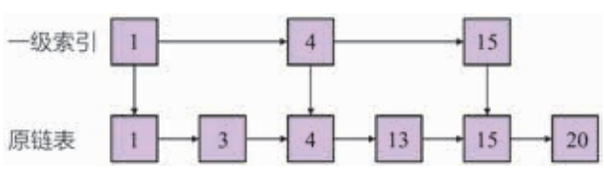


图6.1.9　删除元素10之后的链表状态

因为上述数据结构是在一个有序链表中通过索引表跳跃着进行查找，所以称为“跳跃

表”，跳跃表是William Pugh于1990年发明并提出的一种数据结构。

从数组到链表，再到跳跃表，可以发现，一个切合实际的数据结构和算法不是一蹴而就的，而是根据问题中数据及其关系的特点，通过迭代逐步优化得到的。

问题与讨论

“跳跃表”是立足链表，借鉴二分查找的思想而形成的数据结构。能否立足有序数组，借鉴链表的思想构造一种新的数据结构来解决上述问题？

前面的例子分别采用数组和链表来解决问题，除了这两种方式以外，还可以采用二叉排序树等数据结构。

6.1.3 其他数据组织与处理方式

单纯采用传统的磁盘数据库来组织、处理海量的数据，其固有的数据组织、存取、处理等模式无法适应当今很多数据业务对实时数据管理和查询的需求。于是，人们针对磁盘数据库存在的瓶颈，发明了内存数据库，并于1994年推出了第一个商业化的、实际应用的内存数据库。大部分的内存数据库主要从以下几个方面来提升数据的处理性能。

1. 减少对磁盘的访问

内存数据库顾名思义就是将需要处理的数据保存在内存中并直接操作的数据库。因为内存的数据读写速度比磁盘要高出几个数量级，所以内存数据库首先在数据的输入和输出上极大地提高了系统的性能。相比于传统的磁盘数据库，内存数据库可以将数据处理速度提高至少10倍以上，理想状态下甚至可达到1000倍。

2. 对数据进行分级存储

内存数据库对所有需要处理的数据重新进行组织，并根据应用需要将数据分级，再在处理器缓存中进行分级存储。由于处理器缓存速度高于内存，因此进一步提升了数据的存取效率。

3. 采用改进后的数据结构来组织、存储数据

内存数据库还能将全部数据在内存中进行重新组织、存储，进行新的体系结构设计，用更快速的算法来处理数据，并运用支持快速算法的数据结构来组织数据。例如，AT & T Bell实验室的Silicon数据库核心组织用AVL树（一种平衡树）作为索引结构来提高查找速度；Redis数据库运用了Sorted-set技术，而Sorted-set这种有序集合技术正是对跳跃表的改进和应用。

6.2 POI数据的组织与应用

随着计算机技术、通信技术和3S技术（遥感技术RS、地理信息系统GIS、全球定位系统GPS的统称）的发展融合，互联网环境中的地理信息服务已经成为人们网络生活的重要组成部分。网络地理信息服务的开展离不开大数据的支持，如电子地图在生活中的应用就需要POI数据的支持。本节主要介绍POI数据在电子地图中的组织与应用。

6.2.1 POI数据的概念

POI是“Point of Interest”的缩写，可以翻译成“兴趣点”，有些时候也叫作“Point of Information”，即“信息点”。电子地图上一般用气泡图标来表示POI，如图6.2.1所示的是在导航地图中显示的某区域范围内的兴趣点。比如电子地图上的景点、公司、商场、饭馆等，都是POI。对于一个地理信息系统来说，POI的数量在一定程度上代表着整个系统的价值。

图6.2.1 导航地图中显示的兴趣点

POI作为可以在电子地图中查询到的信息点要素，它描述了空间实体或者区域的空间位置、名称地址等信息，基本信息包括名称、类别、空间坐标信息（经纬度）、地址、电话、邮政编码等，涵盖了政府机构、学校、商业、金融、餐饮、娱乐、景点、加油站等最为基础的信息。衡量POI数据价值的指标有：空间位置的准确性和覆盖率、空间位置的数量。

POI数据在社会各个领域都得到了广泛的应用。例如，目前交通系统中监控设备覆盖面积较广，能产生非常详细的交通和事故数据，如公交动态数据，包括公交行进线路名称、公交当前运行方向、到站时间、站点名称等。这些信息随时会发生变化、实时更新，对这些数据的存储分析是城市公交信息研究的重点。比如目前基于GPS获取的公交数据的一个重要应用是路段行程速度估算，其估算结果作为城市拥堵的重要评价指标。又如，城市中共享单车所产生的POI数据给城市公交线路的设置提供了科学依据；根据某条道路的24小时动态人流量，做出道路的改造方案、商业店铺选址决策等。

6.2.2 POI数据的组织与表示

POI数据一般以表记录或点状数据集的形式存在，如以表结构形式储存于Oracle的大型数据库中，其数据结构主要包含下列数据：点要素属于的矩形分幅的网格号、唯一标识号、点的POI分类编码、POI名称（汉字和拼音）、POI电话、POI地址和POI的经纬度坐标等。

基于网络的电子地图信息服务应用广泛，用户一般通过浏览电子地图上的POI符号及注记信息，或者直接查询要寻找的POI名称来实现空间位置定位。

随着位置服务信息的多样化和精细化，需要大量的POI数据作为支撑，基于POI数据的特点，对POI数据的组织、管理和操作也有别于传统数据结构所体现的数据之间的逻辑关系，并要考虑为后续的查询提供算法设计的支持。

Hadoop作为Google云计算技术的开源实现，提供了一种超大规模、高可靠性、高可扩展性的存储及计算海量数据的框架，可以在海量的POI数据组织管理方面获得更方便的体验和更低廉的成本。采用Hadoop作为地理信息存储与计算的基础框架，基于HDFS文件系统的高容错性和高吞吐量特点存储空间影像数据；基于HBase的存储可靠性强、检索性能高、存储列可按需增加的特点存储地理信息专题数据；基于MapReduce的计算能力对地理信息中的各种数据进行搭建，对地理信息专题数据进行信息提取，提取有效信息。

POI数据的组织主要涉及空间索引问题，空间索引是指依据空间对象的位置和形状或者空间对象之间的某种空间关系，按一定的顺序排列的一种数据结构。它包含空间对象的概要信息，如对象的标识、外接矩形及指向空间对象实体的指针。空间索引可以使空间操作快速访问对象，缩短空间查询时间，从而提高效率。空间索引技术大致分为基于树结构、基于网格划分等。考虑到POI数据本身为点信息，一个数据仅可能出现在一个索引位置中，因此经常使用网格空间索引来对POI建立空间索引。

网格索引的空间索引技术是将一幅地图的地理范围均等划分为M行N列，即规则地划分为二维空间数据，得到M×N个小矩形网格区域。每一个网格区域为一个索引项，为全部或者部分落入该网格内的地图对象建立空间索引，如此，检索空间将大大减小。

如图6.2.2所示，对POI（图中以三角形代表）建立网格索引，要判断哪些POI落在图中矩形选择框中，就不需要把所有的POI与此矩形框进行几何包含运算，只需对A、B、C、D、E这5个点作运算即可。也就是说，如果一个图层有10万个POI，不建立空间索引，任何地图操作都将对整个图层的所有POI遍历一次，也就是要循环10万次；建立索引将使得程序循环的次数下降很多，效率也随之提高。

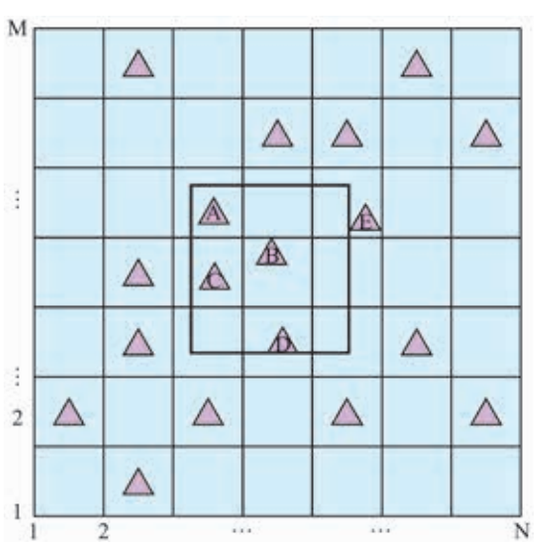


图6.2.2　网格示意图

问题与讨论

以查询自己所处位置50米范围内的POI数据为例，试着讨论引入空间索引技术的必要性。

6.2.3 POI数据处理中的数据结构与算法分析

对于POI数据的操作，可以先确保连接到POI数据库，通过SQL语句读取每一个POI记录，将POI的名称、地址、电话、分类、经纬度坐标写入索引数据。例如，将兴趣点信息“名称：浙江大学(玉泉校区)；地址：浙江省杭州市西湖区浙大路38号；分类：教育；经纬度坐标：120.129649,30.270072”写入索引数据库。同样也可以去索引文件中根据关键字进行查询，存储符合条件的记录，获得结果集。

在获得结果集的前提下，如果要根据某个点的坐标在海量的数据点中找到该坐标所在的区域或者最接近的点，可以使用二分查找或B树查找。但是，问题在于不仅要找到这个点，还要找到这个点附近的点。对于用经纬度来确定的坐标值，不方便直接使用二分法进行查找，如果直接用数据库索引进行查找，由于数据库通常是B树索引和Hash索引，因此查找效率并没有提高。为了解决此问题，现在的主流数据库都增加了对地理坐标优化的空间索引算法。

用R树、K-D树或者是四叉树之类的数据结构来存储这些点可以做到高效地查找临近点。但是这些数据结构存在数据冗余、不稳定的查改效率等缺点。况且抛开时间效率、空间效率以及算法复杂度等因素，用了这些数据结构也就意味着放弃了使用现成强大的数据库而需要自己编写数据查改系统。因此，如果能够把一个二维的信息转化为一维的数据加以存储，就可以直接存储到数据库中以便快速的查找，GeoHash就是一种能够满足上述问题解决、广泛应用于空间检索，尤其是POI数据查询的算法。GeoHash算法因为其稳定性和拓展性，在POI数据的处理中得到了广泛的应用。

拓展链接

GeoHash算法

GeoHash算法由Gustavo Niemeyer（大概于2013年）提出，目的是为地球上的每一个点（根据经纬度）确定一条短的URL作为唯一的标识。

GeoHash算法把一个坐标点映射到一个字符串上，每一个字符串代表的就是一个以经纬度划分的矩形区域。比如图6.2.3展现了北京地区所在的九个区域，分别是WX4ER、WX4G2、WX4G3等。

图6.2.3

与此同时，每一个区域又可以继续划分为许多个子区域，每个子区域的GeoHash值都是在父区域后面拓展一个字符，这样就形成了一个层次分明的结构。越高级的区域GeoHash值越短，表示的区域越大；越低级的区域GeoHash值越长，表示的区域越小。GeoHash算法用类似四叉树的方法来寻找一个点，对经度和纬度不断地进行二分，最终确定到想要的精度，划分的过程如图6.2.4所示。

每一个区域最后都会得到一个二进制的字符串，为了方便，每5位为一组，用0—9，b—z（去掉a, i, l, o）进行Base32编码，即可得到该区域经过编码后的GeoHash值。在坐标写入数据库的同时，程序计算出坐标的GeoHash值并一起入库，就可以实现快速进行周边POI查找之类的功能。

图6.2.4

项目挑战

模拟创业

随着城市边界的扩大，人们的生活成本也随之增加，商业的合理布局对于提高社区居民的生活质量、城市的优化布局起着重要的作用。作为商业投资者，需要最大限度地挖掘社区POI数据的价值，精准获取本区域相关产业的整合度情况，从而为创业决策提供科学的依据。

项目任务

基于现代创新创业的生态环境，结合传统的商业运作模式，本项目模拟创业的具体要求如下：

1. 分析并统计区域内各类POI数据信息。
2. 初步确定创业的行业方向及选址意向。

过程与建议

你可以借助“传统+现代”的信息获取和决策方式来完成模拟创业的项目挑战，建议步骤如下：

1. 市场调查、实地考察

以某个社区为中心点选择模拟创业的大致区域范围，可以使用传统的方式，如问卷调查、居民走访等形式进行前期的市场调查或实地考察。

2. POI数据的获取和分析

借助数据技术和能力，从大数据的视角进行分析。可以考虑从以下几方面获取信息并进行分析：

（1）获取人口信息，包括覆盖的人数、性别、年龄等，即此区域的生活人群。

（2）通过查询相关POI数据提供商的网站，获取相关POI数据。

（3）分析POI数据信息，包括竞争分析、交通网络分析、基础设施分析等，确定创业的初步设想。

3. 行业方向及选址意向的评估

建议对获取的数据做进一步挖掘：

（1）行为信息。在人口信息中，除了人口数量，可以考虑适当购买商业数据，更需要挖掘这些人有没有购买意向、会购买多少，未来的趋势如何。

（2）查看每个时间段的热力图，获取某地域某店铺的热度，评估行业方向及选址意向，如：到人流量大的地方开设餐饮店。

4. 展示交流

明确创业意向并撰写创业企划书，并与同学进行展示交流。创业企划书应包括以下内容：

（1）创业的行业方向。

（2）具体的创业内容描述。

（3）选址意向。

（4）创业内容的可行性分析。

（5）选址意向的可行性分析。

评价标准

请根据项目实施的过程、结果和交流效果，对自己完成项目的情况进行客观的评价，并思考后续完善的方向。将评价结果和完善方案填写在下面的表格中。

评价条目	说明	评分（1～10分）	评分主要依据阐述	后续完善方向
数据采集	能够初步采集到区域内居民日常生活消费情况的数据			
	能够获取区域内商业相关的POI数据			
确定意向	分析数据，能够初步确定创业的行业方向和选址意向			
决策论证	能够对创业的内容进行具体描述			
	有较为明确的选址意向			
	对创业内容和选址做过详细的可行性分析			
展示效果	撰写完整的创业企划书，完成交流			