国家教材委员会专家委员会审核通过 2019

普通高中教科书

XINXI JISHU

信息技术

选择性必修 1

数据与数据结构

ESPH 教育科学出版社

普通高中教科书

XINXI JISHU

信息技术

选择性必修 1

数据与数据结构

教育科学出版社
·北 京·

总 主 编 李 艺　董玉琦
本册主编 李冬梅
本册副主编 陈 斌
主要编者 李冬梅　陈 斌　钟建业　毛华均　张 宁
陈 阳　张春鸿　林志奕　汪 知

出 版 人 李 东
责任编辑 贾立杰
版式设计 国美嘉誉文化 王 辉
责任校对 贾静芳
责任印制 叶小峰

普通高中教科书

信息技术 选择性必修 1 数据与数据结构

教育科学出版社出版发行
(北京·朝阳区安慧北里安园甲 9 号)
邮编：100101
总编室电话：010-64981290 编辑部电话：010-64989637
出版部电话：010-64989487 市场部电话：010-64989009
传真：010-64891796
网址：http://www.esph.com.cn
各地新华书店经销
北京市大天乐投资管理有限公司印装
开本：890 毫米 ×1240 毫米 1/16 印张：11
2020 年 1 月第 1 版 2021 年 12 月第 5 次印刷

ISBN 978-7-5191-1095-6
定价：17.50 元 (含光盘)
批准文号：京发改规〔2016〕13 号 价格举报电话：12315

图片来源：高品（北京）图像有限公司

前　言

在当今大数据和人工智能时代，数据对人类社会的发展越来越重要，尤其在科学发现、技术进步、经济发展和政府决策等方面表现突出。谁拥有大量优质的数据，能对数据进行有效的分析处理，谁就能做出更大贡献。数据结构对同学们深入理解和掌握信息技术学科知识与实践方法、培养信息意识与计算思维、形成学科核心素养，具有非常重要的作用。

本书根据《普通高中信息技术课程标准（2017年版）》编写，分为数据及其结构、线性表及其应用、数据的排序与查找、队列及其应用、栈及其应用、树及其应用六个单元。本书内容主要以项目学习的方式组织，在给出明确的“学习目标”的基础上，设计生动有趣的“任务”引领“活动”，使同学们在体验和游戏的过程中快乐地学习，通过解决真实问题完成相应的任务。本书还设置了必要的“拓展练习”供同学们自我检测，以具体的测评要求和思维导图引导同学们进行单元学习评价与总结。

本书基于Python程序设计语言引导同学们学习数据结构，迭代与递归思想贯穿始终。同学们通过学习与实践，可以掌握数据抽象的方法，掌握常用基本数据结构的实现方法，以及利用数据结构解决实际问题的基本思想和方法，深入理解数据结构与算法的关系。

本书广泛吸收一线教师教学实践中的优秀成果，注重激发同学们学习的内在动力，适合同学们自主学习。另外，本书还配备了相关的配套资源，以支持同学们更有效地进行学习。

希望同学们立足于本书，又不局限于本书，做到理论联系实际，在实践中练就技术本领，树立创新意识。期待大家积极分享学习中的各种收获！

目　录

第 1 单元　数据及其结构　1

1.1　数据及其价值　2
1.2　数据的组织结构　14
1.3　认识数据抽象　24
单元学习评价　32
单元学习总结　33

第 2 单元　线性表及其应用　34

2.1　线性表结构及其实现　35
2.2　随机抽取问题　45
2.3　字符串应用　50
单元学习评价　59
单元学习总结　60

第 3 单元　数据的排序与查找　61

3.1　迭代与递归　62
3.2　数据的排序　69
3.3　数据的查找　77
单元学习评价　85
单元学习总结　86

第 4 单元　队列及其应用　87

4.1　队列结构及其实现　88
4.2　基数排序　96
4.3　排队取号模拟系统　103
单元学习评价　110
单元学习总结　111

第 5 单元　栈及其应用　112

5.1　栈结构及其实现　113
5.2　符号匹配问题　122
5.3　算术表达式求值　129
单元学习评价　135
单元学习总结　136

第 6 单元　树及其应用　137

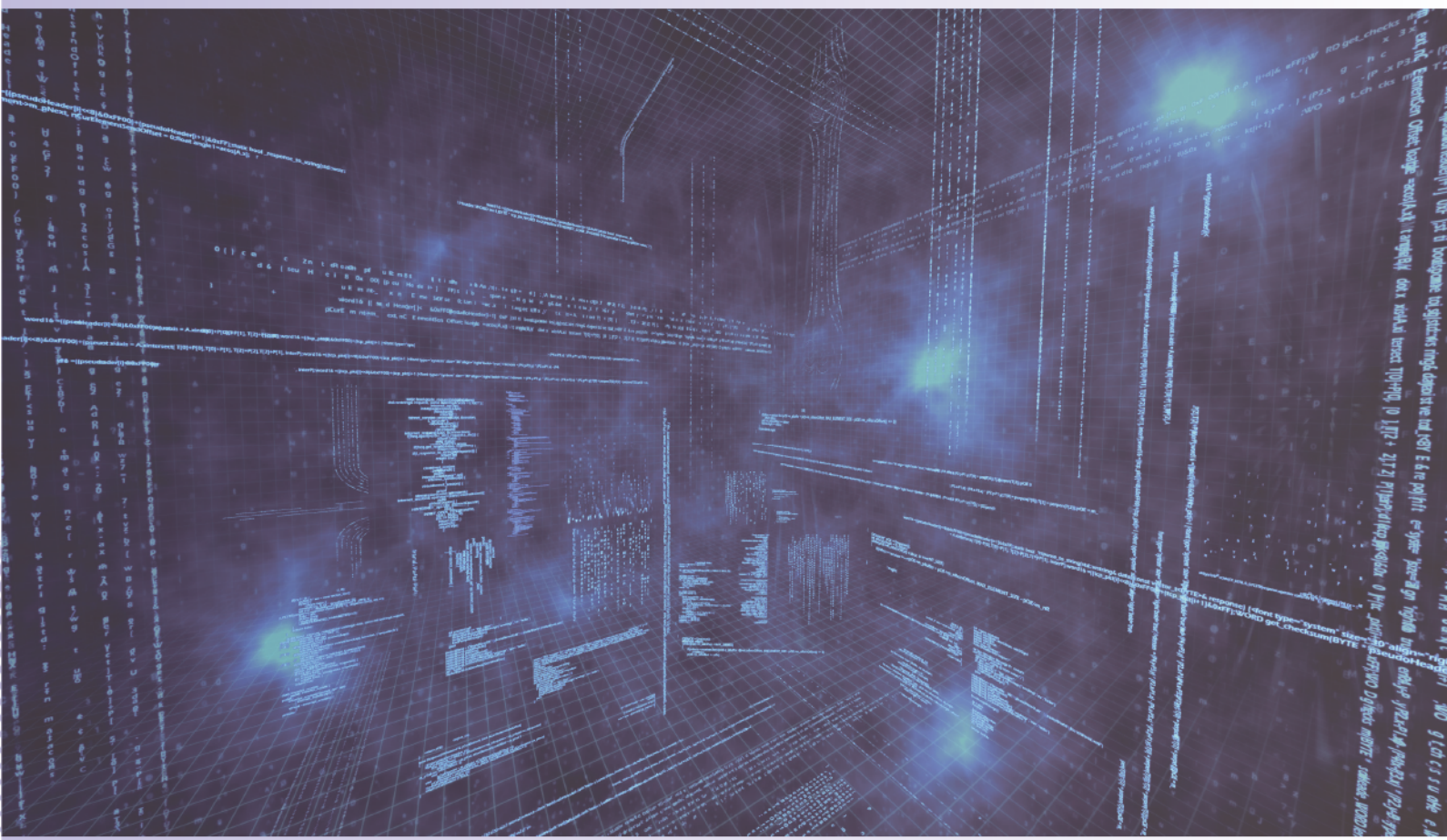
6.1　树结构及其实现　138
6.2　用二叉树排序　150
6.3　画出二叉树　158
单元学习评价　165
单元学习总结　166

后　记　167

第 1 单元　数据及其结构

数据是现实世界中各种事物和现象的抽象化与符号化表示。随着信息技术的快速发展和普及，对数据的获取和处理产生了越来越高的价值。为了更加高效地处理大量数据，需要把数据组织为合理的结构，并针对需要解决的问题进行不同层次的抽象，降低问题解决方案的复杂度。

本单元通过“数据背后的秘密”项目来理解数字、数值和数据的含义，认识数据的价值，通过“校园常见数据”项目来探究数据的组织方式，通过“电视机抽象建模”项目来理解数据抽象概念。



1.1　数据及其价值

数据是现实世界中各种事物和现象的抽象化与符号化表示。现代计算机处理的不再是单纯的数值型数据，更多的是文本或图像、音频、视频等非数值型多媒体数据。随着信息技术的快速发展和普及，对数据的获取和处理超越了科学计算的范围，扩展到社会生活的各个领域，产生了越来越高的价值。

本节主要通过对与人类活动息息相关的数据案例的介绍与分析，学习数据的概念，体会从数据中获取信息及归纳知识、辅助决策的过程，了解大数据的应用，并由此认识数据的价值。

学习目标

- ★ 理解数字、数值和数据的基本含义。
- ★ 体会从数据中获取信息及归纳知识的过程。
- ★ 了解大数据的应用及价值。

地球观测卫星是指专门对地球的环境与资源进行遥感观测的非军事用途卫星，其用途包括环境监测、气象监测和地图制作等。

卫星影像，是各种人造地球卫星在运行过程中，通过照相机、电视摄像机、多光谱扫描仪等设备，对地面地物进行摄影或扫描所获得的图像资料，有时也称卫星相片。

卫星影像的信息，不仅可以用于城乡规划、农作物估产，还可以用于抗灾救灾、军事指挥。如果结合北斗卫星导航系统，你还能知晓现在身处何方，如何到达目的地。

本节围绕“数据背后的秘密”项目展开学习，通过项目活动理解数据的概念和特征，体验从数据中获取信息及归纳知识的过程，认识数据的价值。本节主要包含“机场卫星影像分析”和“大数据案例分析”两个任务。

任务一 机场卫星影像分析

※ 活动1 在卫星影像上找到各种地物

机场是常见的大型人工建筑，在从太空中拍摄到的卫星影像里能很容易地识别出机场。图1.1.1与图1.1.2分别是两个机场周边的卫星影像图，仔细观察卫星影像图，可以辨别出各种天然物体和人工物体，请直接在图中标示出如下地物：

- 机场范围，包括跑道、航站楼等附属设施；
- 城镇、农田区域；
- 道路、桥梁；
- 树林、河流、山峰和湖泊。

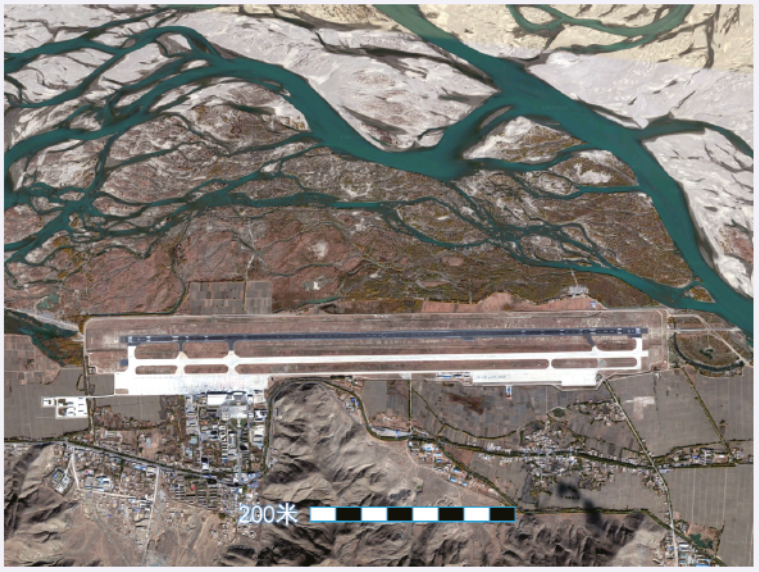


图1.1.1 某高原机场

图1.1.2 某城市机场

地物：位于地球表面的物体，包括天然物体和人工物体。

卫星获取的地球影像中，可见光只是一小部分，更多的是肉眼不可见的红外光、紫外线、微波等形成的影像。它们均具有广泛的用途。

卫星影像图中还附带了比例尺，请在图中找到飞机跑道，并用刻度尺在图中测量、推算跑道长度，填写表 1.1.1。

表 1.1.1　机场跑道长度测量

机场	跑道长度 / 米	实际长度 / 米	误差原因分析
高原机场		4000	
城市机场		3900	

● 数字、数值和数据

一般而言，数字（number）是由代表数的文字，如0~9、零至九等组成的字词；而数值（value）则是可以表征大小、次序等概念的量。数字是形式，数值是内容。

数据（data）是对客观事物的符号表示。数据的类别多种多样，可以是整数、浮点数这样的数值，也可以是英文、中文这样的文本字符；可以是线画元素组成的图形，也可以是像素整齐排成的图像；可以是记录声音的音频，还可以是表示动态影像的视频。

随着信息技术的快速发展和普及，计算机获取的数据范围越来越广、类型越来越丰富、数量越来越多、速度越来越快。

※ 活动 2　测量跑道宽度和飞机翼展

如果把卫星影像放大，就可以看到关于机场的更多细节。从图 1.1.3 和图 1.1.4 中我们可以观察到跑道、航站楼、飞机和登机口。

图 1.1.3　高原机场跑道

图1.1.4 城市机场跑道

请按照图中附带的比例尺，推测机场跑道的宽度和停机坪上最大飞机的翼展，填入表1.1.2，并根据表1.1.3所示的常见客机的数据推测停机坪上的飞机型号。

翼展：指固定翼飞行器的机翼左右翼尖之间的距离。

表1.1.2 机场跑道宽度和飞机翼展

机场	跑道宽度/米	跑道条数	最大飞机翼展/米	推测飞机型号
高原机场				
城市机场				

表1.1.3 常见客机数据

制造公司	机型	长度/米	翼展/米
空中客车	A319	33.84	34.09
空中客车	A330	58.82	60.3
空中客车	A380	72.72	79.75
波音	B737	33.6	34.31
波音	B747	70.6	64.4

人的肉眼观察会比较粗略，要得到确切的结果，往往需要更多其他数据的佐证和支持。

思考：通过对机场卫星影像的观察，我们还能得到关于两个机场的什么信息？

观察上述机场卫星影像的数据，我们可以获取有关两个机场的跑道、停靠机型、规模及繁忙度的信息。由此可知，对数据进行加工后，可以从中提取信息。

除了机场外，我们在观察卫星影像的其他地方时，还会看到各种颜色的“色块”。一片一片的绿色主要是植被，包括草场、森林和耕地等；密布的土黄色和灰色主要是城市和村庄；深浅不同的蓝色主要是湖泊和其他水体。关于地势和海拔，可以根据植被（绿色部分）的形状推断山的形状；一般来说，城市和村庄都处在平原和山谷中的平缓地带。从这些“色块”中统计出不同颜色的数量、大小，就形成了各种有关地貌面积、分布的信息。

※ 活动 3　加工数据进一步获取信息

并不是所有的信息都能够直接从原始卫星影像中获得，为了获取更多的信息，往往需要对数据进行进一步的加工处理。

实际上，城市机场还有一条容易被忽略的小型跑道，只是观察原始卫星影像很难看出来。我们可以对卫星影像进行加工处理，将这条小型跑道显现出来。

人们研发了图像识别系统，可以让计算机自动识别出人工建筑，实现对大量数据的自动处理。

由于人工建筑往往具有规则的外形，对原始卫星影像进行“边缘检测”的加工处理，就能将干扰数据排除，留下规则的人工建筑轮廓线条。对城市机场跑道图像进行边缘检测处理后的效果如图 1.1.5 所示，可以看到跑道上标志线和箭头变得更加醒目。

请在图 1.1.5 中找到小型跑道并标示出来，然后按照图中附带的比例尺，测量小型跑道的长度和宽度。

小型跑道的长度约为_____米，宽度约为_____米。

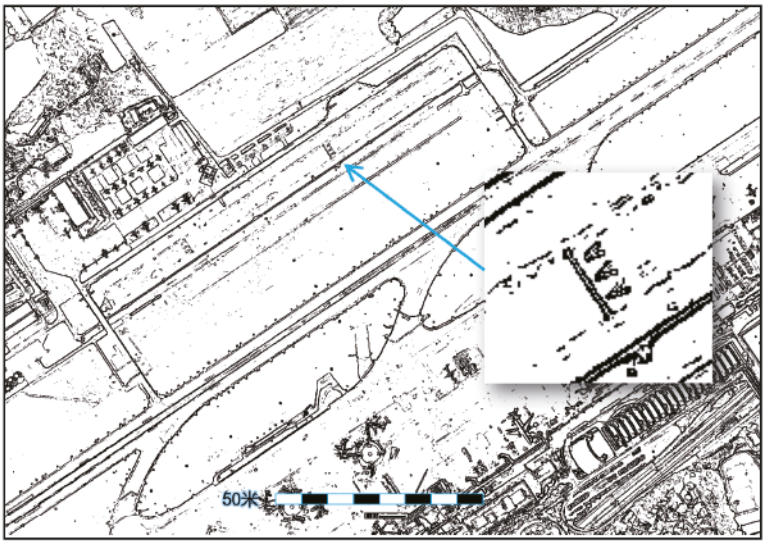


图 1.1.5　进行边缘检测处理后的机场跑道

数据的加工处理并不是一成不变的，随着数据处理技术的进步，人们可以从数据中获取的信息越来越多。对于卫星影像的深度处理，可以让数据产生更大的价值。例如，结合利用低空倾斜摄影技术获取的地面影像，能够快速构建城市的大规模三维模型，应用在测绘、城市规划和虚拟现实等方面。图1.1.6所示为采用基于影像的三维重构技术自动生成的城市机场三维模型。

图1.1.6 城市机场三维模型

任务二 大数据案例分析

※ 活动1 数据揭示全球变化

地球是人类的唯一家园，人类的所有活动都在这个辽阔而又脆弱的生态圈展开。科学技术的飞速发展，使人类获得了前所未有的能力，将累积了亿万年所形成的化石能源——煤炭、石油和天然气，从地下开采出来，用以改造家园。

在经历多次气候和生态危机之后，人类意识到了无节制地索取地球资源的危害，可持续发展的理念被广泛接受。经过研究，科学家们将日益严重的气候和生态危机归结为全球变暖，即地球的年平均温度在持续上升。

人们从几个世纪积累的观测数据中找到了全球变暖的最可能原因：温室气体二氧化碳在大气中急剧增加，而二氧化碳正是化石能源大量燃烧的产物。观察图1.1.7展示的数据变化，回答后面的问题。

在图 1.1.7 中，纵坐标表示的是二氧化碳的浓度（单位：PPM）以及大气温度的变化。

虽然绝大多数科学家都认为目前的全球变暖与人类活动密切相关，但也有一部分科学家认为全球变暖更大程度上是自然现象。

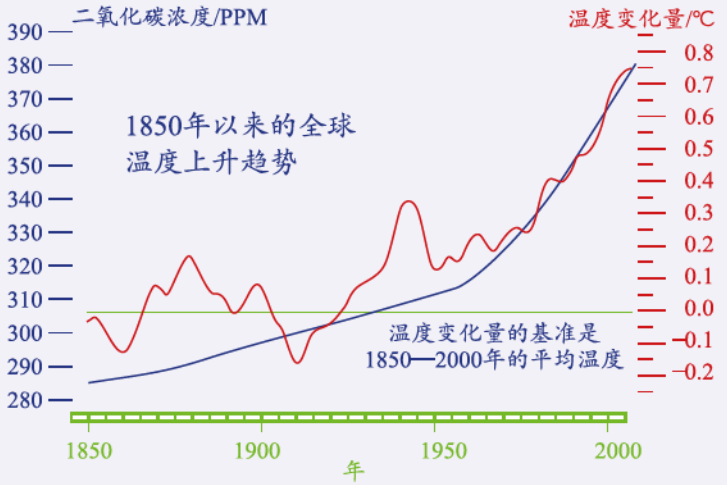


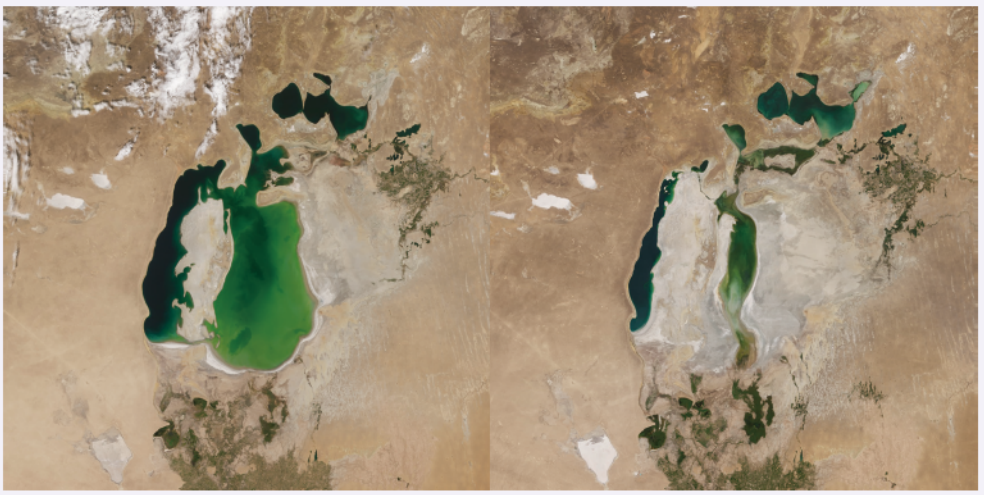
图 1.1.7　二氧化碳排放与全球变暖

1. 从 1850 年到 2000 年，大气中二氧化碳的浓度从_____上升到_____，全球平均温度上升了_____，两者是否有关联？如果有关联，是正相关还是负相关？

2. 人类应该采取什么措施来阻止气候和生态危机的进一步发生？

20 世纪科学进步的一个突出标志是人类开始脱离地球，从太空中观测地球。古诗云：“欲穷千里目，更上一层楼。”在高高的太空中漫游的人造地球卫星一刻不休地观测地球所发生的一切，几十年间积累了大量的遥感数据。人们通过对比历史积累的遥感影像数据，能够更加准确、深刻地把握全球变化所带来的影响。

图 1.1.8 中的咸海，是中亚的一个内陆咸水湖，位于哈萨克斯坦共和国和乌兹别克斯坦共和国的交界处。对照 2000 年和 2016 年的卫星影像，解答以下问题。



（a）2000年的卫星影像　　（b）2016年的卫星影像

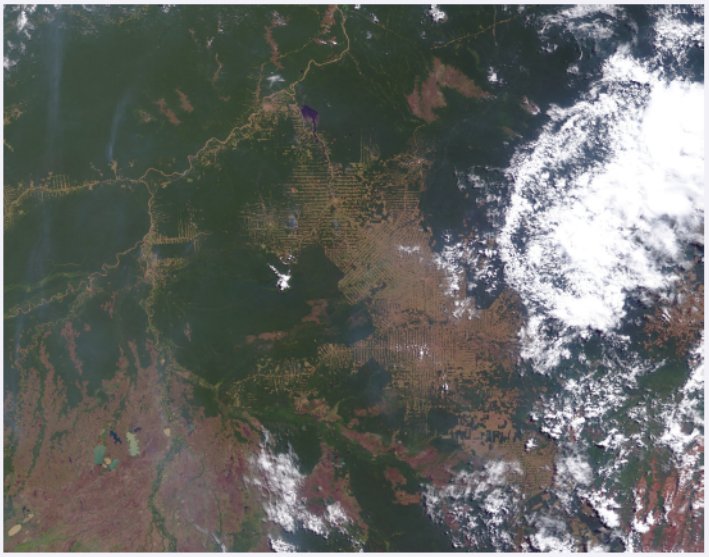
图 1.1.8　咸海变迁

1. 在图中标出消失的水面部分，估计16年间水面面积减少的百分比。

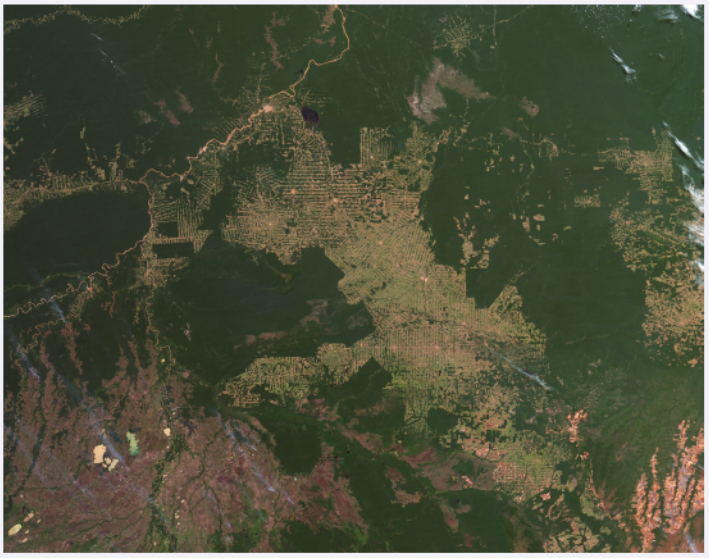
2. 结合全球变暖现象，探讨咸海水面面积大幅度减少的可能原因。

在巴西西部的朗多尼亚州，曾经拥有208 000平方千米的森林，现在那里已经成为亚马孙地区森林砍伐最严重的地方之一。美国国家航空航天局（NASA）的Terra卫星上的中等分辨率成像光谱辐射计（MODIS）从2000年开始观测地球，保留了大量有关亚马孙地区的人类活动对生态环境造成的影响的数据。

仔细观察图1.1.9，分辨卫星影像中的森林、伐木区和道路等，回答后面的问题。



（a）2000年



（b）2012年

图1.1.9　亚马孙地区森林的退化情况

1. 就图中所观测到的范围，估计 2000 年到 2012 年 12 年间森林面积减少的百分比。

2. 探讨森林退化对全球造成的影响。

※ 活动2　“黄金周”北京旅游的体验

微博消息指通过微博系统发布的文本、图像、音频和视频等消息，很多微博消息还附带了发布消息时的位置信息。

以智能手机为核心的移动互联网应用在短短几年内就已普及，配置了移动网络连接、GPS、摄像头的智能手机通过微信、微博等社交网络应用，产生了海量的数据。通过这些涉及每个人的大数据“矿山”，我们可以挖掘出传统“问卷调查”方法无法得到的宝贵知识和决策依据。

从设置国庆“黄金周”假期以来，旅游产业得到迅速发展。“黄金周”期间到北京旅游的游客显著增加，那么这些游客都去了北京的哪些地方？他们的体验如何呢？从微博签到数据是否可以得到一些答案？

热力图：用表示温度高低的颜色系列来描绘数据的聚集程度，聚集程度越高的地方呈现出来的颜色看起来越热。

我们把 2015 年国庆“黄金周”期间所有在北京中心城区签到的微博消息收集起来，按照签到的位置标注到地图上，形成了如图 1.1.10 所示的北京中心城区微博用户签到热力图。

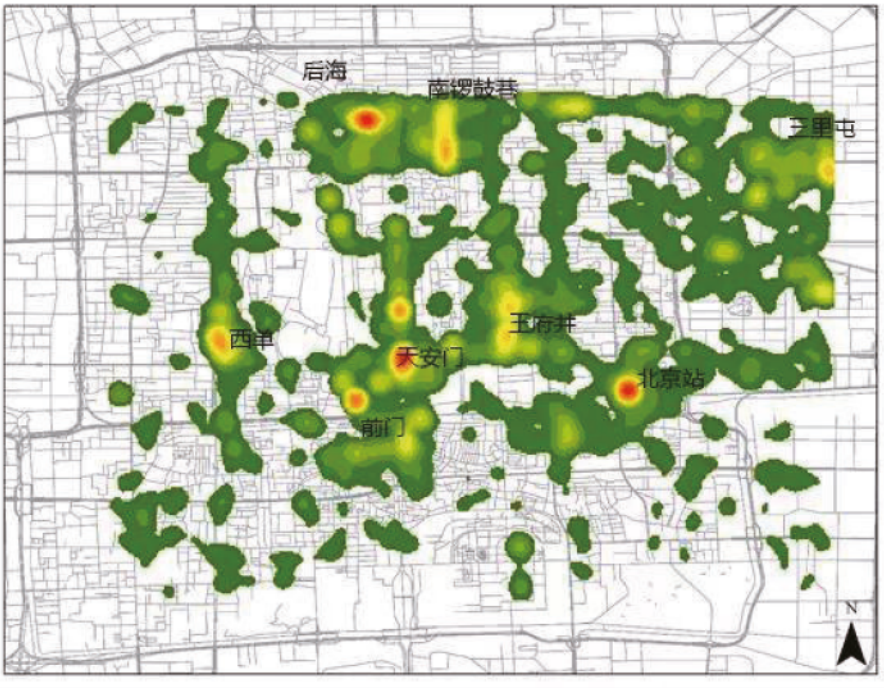


图 1.1.10　2015年国庆假期北京中心城区微博用户签到热力图

仔细观察图 1.1.10，解答以下问题。

1. 对图中标注了地名的热点区域分类，探讨游客在这些地区集中的原因。

2. 图中还有一些没有标注地名的热点区域，选择其中 1~2 个热点，对照地图，探讨游客集中的原因。

微博消息中除了关键的位置信息外，消息内容本身也是很有价值的数据，我们可以通过消息内容中的一些关键词来做进一步的分析，比如可以衡量旅游满意度的游客情绪空间分布。

我们可以收集几类与情绪相关的关键词，统计微博消息中它们出现的频率，再标注到地图上。

涉及的微博消息关键词分为以下三类。

①景点名称：天安门、南锣鼓巷、西单、王府井等。

②形容心情的表情符号：喜欢、开心、害羞、害怕、震惊、赞扬等。

③形容景点情况的关键词：人山人海、问题、挤、美食等。

根据正负情绪关键词的出现频率进行叠加，可以得到如图 1.1.11 所示的微博正负情绪空间分布图。

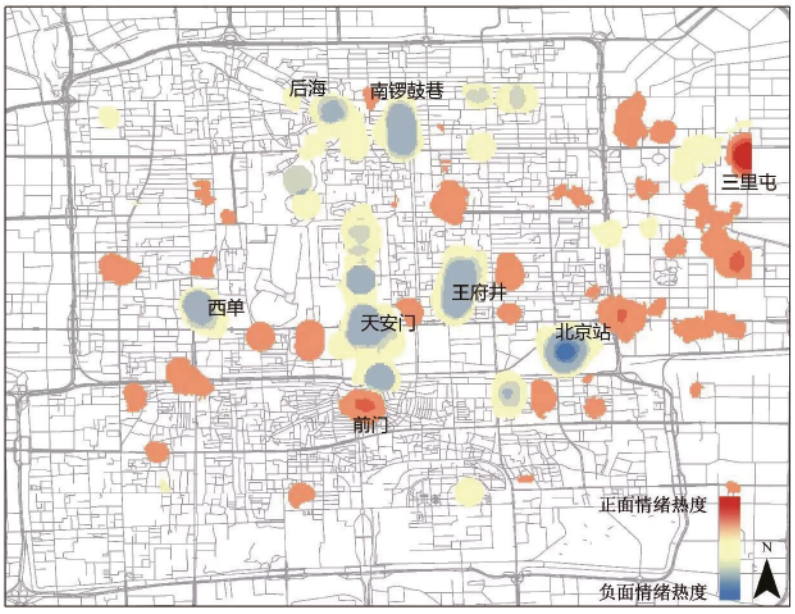


图 1.1.11　微博正负情绪空间分布图

仔细观察微博正负情绪空间分布图，解答以下问题。

1. 对图 1.1.11 中标注了地名的正负情绪热度区域分类，探讨这些地区情绪分布的成因。

2. 微博消息还包含了发布微博的时间信息，根据微博正负情绪空间分布图，你觉得还需要什么数据、进行什么分析，以便为游客提供出行参考？

正负情绪：喜悦、轻松等积极的情绪称为正情绪，而悲伤、焦躁等消极的情绪称为负情绪。

● 数据的价值

随着数据获取手段的极大丰富，与人类密切相关的事物正在全面数据化，依托以数据为核心的信息基础设施，人们生产生活的方方面面，在计算机的帮助下，实现了前所未有的进步。

数据，不光是信息加工的原料，处理结果可以作为决策的依据，影响现实世界的运作，甚至在像影视、娱乐、游戏这样高度数字化的行业中，数据还是唯一的产品形态。这些藏身在虚拟空间里的无形之物，不仅能提高真实事物的价值，而且往往数据本身，就是无价之宝。

※ 活动3　数据的价值和意义

数据是信息社会的重要原材料，我们从“机场卫星影像分析”和“大数据案例分析”两个任务中看到，数据就像矿山中的矿物一样，蕴藏着丰富的信息和知识，有待我们去开采和提炼。但与一切物质资源有根本性不同的是，数据资源并不会消耗殆尽，而是随着人们对世界认识的深入，可以提炼出越来越多的新知识，作为生产资料制造出越来越多的产品。

基础设施是充分利用资源和生产资料的重要机制，人们像对待水资源、矿产资源一样，开发和建成了对数据进行有效的采集、管理、访问、维护、分发利用所必需的政策、技术、标准、基础数据集和人力资源，并将众多数据库连接起来，形成一种类似公路和铁路的基础设施，使全社会能充分地利用和共享数据。

国家地理信息公共服务平台“天地图”是国家基础地理信息中心建设的网络化地理信息共享与服务门户，集成了来自国家、省、市（县）各级测绘地理信息部门，以及相关政府部门、企事业单位、社会团体、公众的地理信息公共服务资源，向各类用户提供权威、标准、统一的在线地理信息综合服务。

请登录“天地图”网站，浏览“在线地图”“专题图层”和“典型应用”等栏目，查阅空间数据基础设施所包含的服务内容，以及典型应用所依托的空间数据生产资料，并填写表1.1.4。

表1.1.4　空间数据基础设施支撑的典型应用

典型应用名称	应用类型	简要功能介绍	依托的数据资源
全国林地一张图政务服务平台	农林牧渔	地图基础浏览、快速搜索定位、林业专题信息的成果展示、林班数据的查询、林地数据的统计分析、多边形区域查询统计等	全国林地图斑数据、林地专题数据、森林资源调查数据等

续表

典型应用名称	应用类型	简要功能介绍	依托的数据资源

从太空中的人造地球卫星到掌心的智能手机，各式各样的数据被源源不断地生成，并在信息社会的数据基础设施中存储、传输、共享和处理，这些数字空间中的无形生产资料，正在重塑着各级产业的传统面貌。

拓展练习

1. 夜晚地球的卫星影像显示了夜间地球表面的灯光，观察图 1.1.12 所示的地球夜景图，讨论其中可能蕴含的结论，并说明还需要什么数据来进一步验证得出的结论。

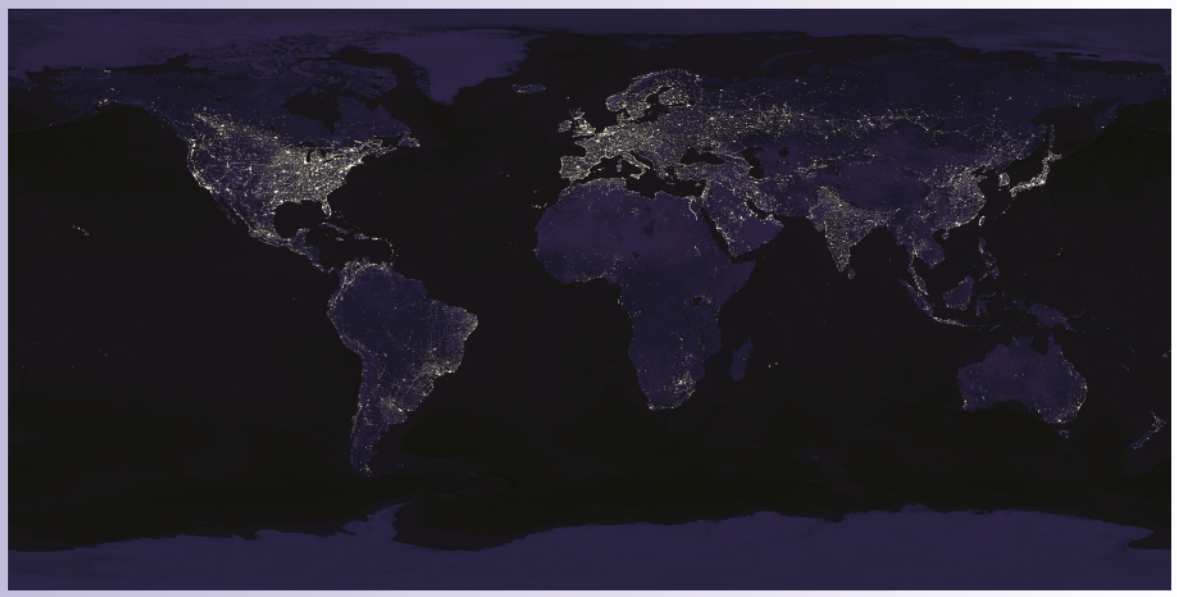


图 1.1.12　地球夜景图

2. 举例说明生活中的哪些消费品离开了数据就几乎没有价值，哪些消费品本身就是纯粹的数据，以及这些消费品的价值可能有多大。

1.2　数据的组织结构

随着信息技术的普及和深入应用，人们需要面对越来越多的数据资源，如图书馆中海量的图书数据、学校里教职员工和学生的数据、铁路运输部门的列车运行线路数据等。在使用和处理这些数据资源的过程中，我们需要研究如何组织数据来提高数据的使用效率。

本节通过具体的生活实例来认识数据组织的重要性，学习数据组织方式、数据结构的概念和实现方法。

学习目标

★ 认识数据组织的重要性。

★ 了解三种数据组织结构，理解数据结构的概念。

★ 掌握数据结构的两种存储方式。

在这个“数据”时代，与问题有关的数据不仅数量庞大，而且存在着错综复杂的相互联系。为了能够在计算机中有效处理这些数据，必须对这些数据进行合理的组织和存储，从而提高处理效率。

本节围绕“探秘校园数据”项目展开学习，通过项目活动，了解数据组织的三种基本结构和两种存储方式。本节主要包含“探究校园数据的组织结构”和“设计校园数据的存储方式”两个任务。

任务一　探究校园数据的组织结构

在我们每天学习、生活的校园中，时时刻刻产生着大量的数据，比如在学校图书馆借书还书、学生社团日常活动、在学校食堂就餐刷卡消费、校园开放日活动等。

※ 活动1 探究图书馆的图书数据组织方式

日常生活中，人们有时候把图书随意摆放，久而久之会形成杂乱无章的书堆，如图1.2.1所示。从没有条理的书堆中找到某本书是非常困难的。

图1.2.1 杂乱的书堆示意图

当你走进学校图书馆的借阅室，看到的是一排排在书架上码放整齐的图书，图书管理员还为每本书编上了索书号，同一层书架上的图书都按照索书号依次排列，如图1.2.2所示（这里为了便于理解，已经将实际的索书号进行了简化）。书架上码放整齐的图书之间就具有了一定关系，在书架上查找图书、取走图书和插入图书就有了一定的“规矩”。

图1.2.2 书架上排列的图书示意图

根据图1.2.2中图书的摆放方式，将索书号和书名填写在下面括号里的横线上。

（160101 诗经）—……—（160116 西游记）—（_____）—（_____）—（_____）—（_____）

如果查找图书时，每次只能看到一本书，那么要找到其中的《三国演义》，需要一个什么过程？把你的方法填写在下面的横线上。

从（160101诗经）开始，_____

_____。

如果从书架上取走《三国演义》后，又放到了《诗经》的右面，

那么书架上的图书摆放会变成什么状态？将索书号和书名填写在下面括号里的横线上。

（160101 诗经）—（_____）—……—（160116 西游记）—（_____）—（_____）

线性结构

在线性结构中，数据元素之间是一对一的线性排列关系。

数据元素之间的排列次序存在一种明确的先后关系，这样的数据组织方式称为线性结构。

在线性结构中，除了最后一个元素，每个元素都有一个唯一的后继元素，所有元素都排成一个线性序列，如图 1.2.3 所示。

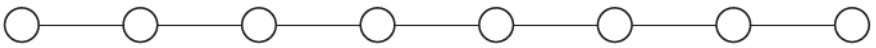


图 1.2.3　线性结构

线性结构是计算机信息系统中最基本和最常见的数据组织结构，很多计算机应用系统都会用到线性结构来组织数据，如超市的商品销售管理、仓库账目管理、学校食堂就餐刷卡消费记录等方面。

※ 活动 2　探究学生社团的数据组织方式

为了丰富学生的学习生活，学校会根据学生的兴趣爱好成立各种各样的学生社团。图 1.2.4 是某中学的学生社团组织机构图。

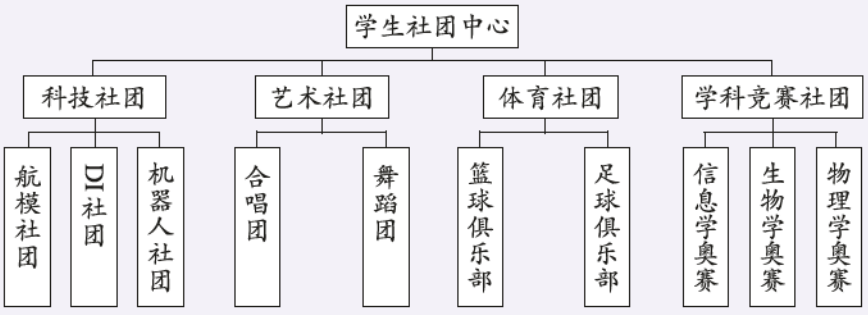


图 1.2.4　学生社团组织机构图

由图 1.2.4 可以看出，学生社团组织机构存在一种层次关系，一个上层机构可以关联多个下层机构。假设把学生社团组织机构抽象成一个数据集合，那么每个机构就是这个集合中的一个数据元素。

请分析下面的问题：

1. 学生社团中心和各分社团之间存在什么样的关系？
2. 科技社团包含哪几个社团？艺术社团包含哪几个社团？

3. 信息学奥赛和生物学奥赛均属于什么社团？

树形结构

数据元素分属于不同的层次，一个上层元素可以关联着一个或多个下层元素，整个结构中只有一个最上层数据元素，这样的数据组织结构像一棵倒放的树，称为树形结构，简称树。如图1.2.5所示，最上层的数据元素称为根。

在树形结构中，数据元素之间是一对多的层次排列关系。

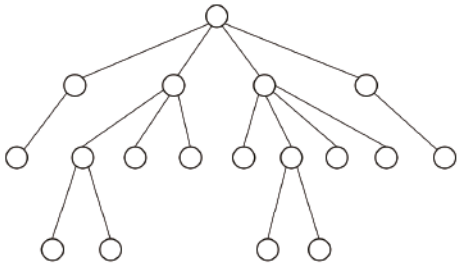


图1.2.5 树形结构

※ 活动3 探究校园参观路线的数据组织方式

每到校园开放日，学校会安排参观校园的活动，以便让更多的人了解校园环境和校园文化。志愿者会分组按照一定的路线带领客人参观校园。图1.2.6是某中学校园示意图，标注了各参观点的位置和通行道路。

图1.2.6 某中学校园示意图

把如图1.2.6所示的校园示意图抽象成一个数据集合，那么图中的每个参观点就是这个集合中的一个数据元素。请根据图1.2.6补全图1.2.7。

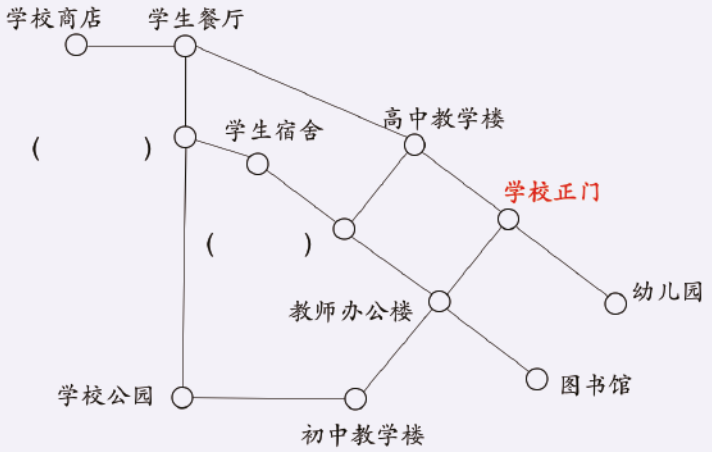


图1.2.7　某中学校园参观路线图

如果从学校正门出发，到达学校公园，请根据图1.2.7，写出至少3条参观路线：_____

_____

● 图状结构

数据元素之间可以有一对一、一对多或多对多的相互关系，这样的数据组织方式称为图状结构或网状结构，如图 1.2.8 所示。

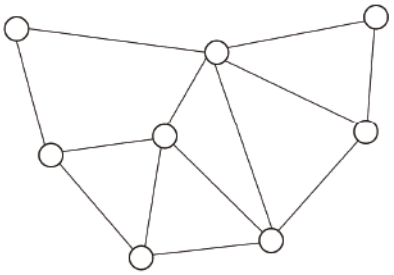


图1.2.8　图状结构

● 数据的逻辑结构

线性结构、树形结构和图状结构分别表示了不同复杂程度的数据元素之间的关联及布局。这些数据元素之间逻辑上的排列和对应关系就是数据的逻辑结构，如图 1.2.9 所示。

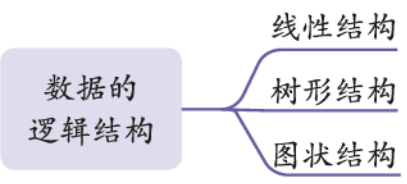


图1.2.9　数据的逻辑结构

任务二 设计校园数据的存储方式

※ 活动1 了解计算机内存中数据对象的存储

程序运行过程中直接使用的数据都保存在计算机的内存储器（简称内存）中。

内存的基本结构是线性排列的一批存储单元。每个单元的大小相同，可以保存1字节大小的数据。内存单元具有唯一的编号，称为地址。内存单元地址为从0开始的连续正整数，如图1.2.10所示。

图1.2.10 计算机内存的基本结构

内存分配是指在程序执行过程中获取或者回收数据存储空间的方法。

对内存单元数据的存取都通过地址进行，数据对象可能占据多个内存单元。

Python语言中内置了返回数据对象的地址及其所占内存单元数量的函数。id()函数返回数据对象的地址，数据对象的__sizeof__方法则返回数据对象的“尺寸”，即占据内存单元的数量。

例如，语句a=2018，将整数对象2018赋值给变量a，可以通过id(a)知道整数对象的地址，a.__sizeof__()则返回整数对象占用的内存单元数量。

以下代码是在Python中对变量的赋值，请根据提示完成表1.2.1。

```
01. counter=100          #整型变量
02. miles=1000.0         #浮点型变量
03. name="alice"         #字符串
```

表1.2.1 内存分配表

变量名	地址	长度
counter		
miles		
name		

● 数据的存储结构

数据结构在计算机内存中的表示方式称为存储结构，存储结构的不同主要体现在数据元素之间相邻关系的表示上，多个数据元素之间的相邻关系则既可以按顺序依次存储来表示，也可以通过记录相邻数据元素的地址来链接引用。

※ 活动2　设计图书数据的存储方式

图书馆中的图书数据组织方式是线性结构，每一本图书的数据之间存在一种一对一的顺序关系，图书管理系统就可以根据计算机内存单元的结构特点来存储书架上的所有图书数据。

在计算机内存中，内存单元按照地址连续排列，可以按照内存单元的顺序来存储数据元素，数据元素之间逻辑上的相邻关系通过内存单元的相邻关系来表示。

假设图书管理系统为每一本图书的数据分配了32个字节来保存索书号和书名，并将书架上的书按照顺序来存储。请仔细观察图1.2.11，填写其中的图书数据和相应的内存单元地址。

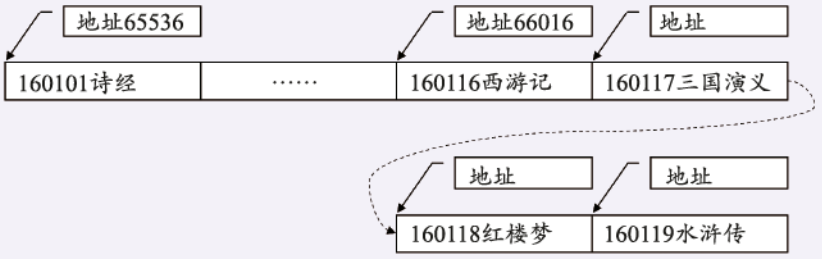


图1.2.11　内存单元中的图书数据

在这种相邻存储的结构中，数据元素占据的内存单元数量是固定的，只要知道了前一个数据元素的地址，就能确定下一个数据元素的地址。如果存储的是一系列大小相同的数据元素，就可以利用公式直接计算出线性结构中任何一个数据元素的地址。

请根据图1.2.11的内容，完成下列图书数据地址的计算公式：

图书数据地址=_____+（索书号-_____）*_____

● 顺序存储结构

把逻辑上相邻的数据元素存储在物理位置上相邻的存储单元中，数据元素之间的逻辑关系由存储单元的邻接关系来体现，这种存储结构称为顺序存储结构。

顺序存储结构的主要优点是节省存储空间，因为存储单元全用于存放数据元素，数据元素之间的逻辑关系没有占用额外的存储空间。采用这种方法时，可实现对数据元素的随机存取，即每一个数据元素对应一个序号，由该序号可以直接计算出数据元素的存储地址。但顺序存储方法的主要缺点是不便于修改，对数据元素的插入、删除运算要移动一系列的数据元素。

Python语言中内置的列表数据类型list就是顺序存储结构，但由于Python语言内部实现的复杂性，无法简单地从list中包含的数据对象的地址观察到典型的连续性。

※ 活动3 设计社团数据的存储方式

社团数据的组织方式是树形结构，数据元素之间存在着层次关系和一对多的关系。对于这样的数据关系，如何在内存中存储和表示呢？

我们知道，在制作幻灯片的时候，可以在幻灯片页面中插入超链接（如图1.2.12所示），用超链接的方式实现页面之间的任意跳转，完成非顺序播放的效果。

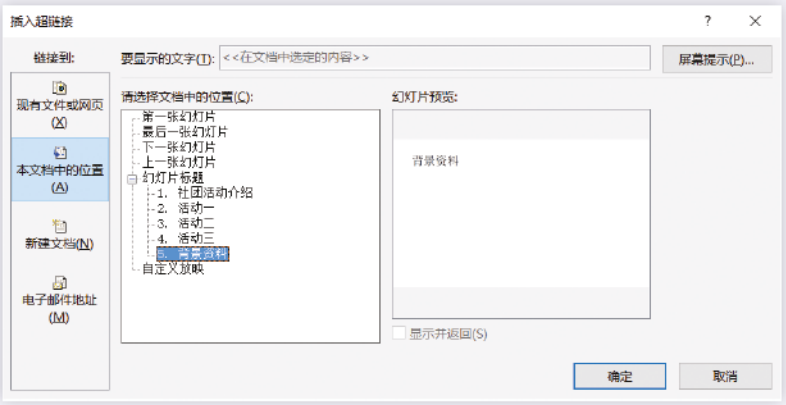


图1.2.12 插入超链接

幻灯片里的超链接，其实就是保存了另一张幻灯片的位置，当我们单击超链接的时候，通过位置定位指向另外一张幻灯片。

根据超链接的思路，在内存里除了存储数据元素外，还可以存储指向另一个数据元素的地址，这样数据元素之间的关系就可以通过链接来表示了，如图1.2.13所示。

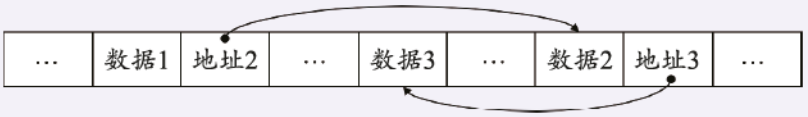


图1.2.13 利用地址引用链接数据元素

采取地址引用来链接数据元素，可以使逻辑上相邻的数据元素在物理上不必相邻，这样就增加了数据操作的灵活性，尤其是插入数据元素时，不需要移动其他数据元素就可以完成此操作，如图1.2.14所示。

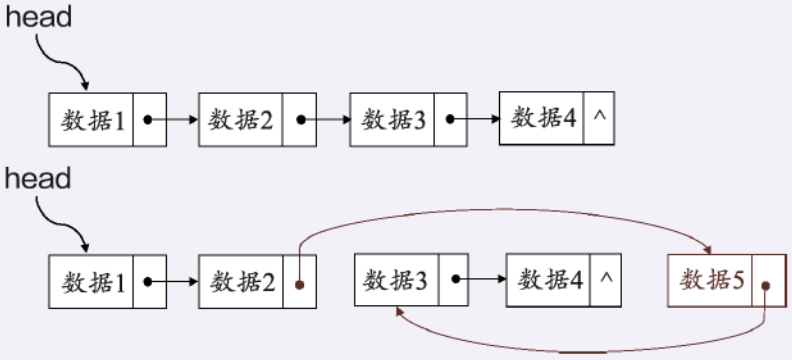


图1.2.14 插入数据元素

图 1.2.15 所示是学校艺术社团的组织结构图，请利用地址引用链接数据元素的方式，将艺术社团的数据存储在图 1.2.16 所示的内存单元中。

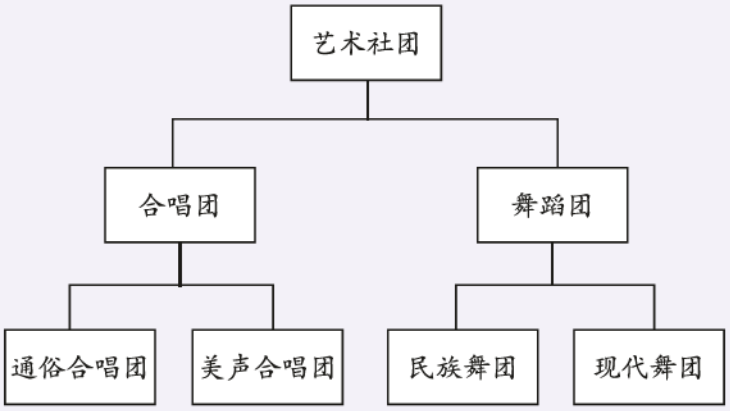


图 1.2.15　艺术社团

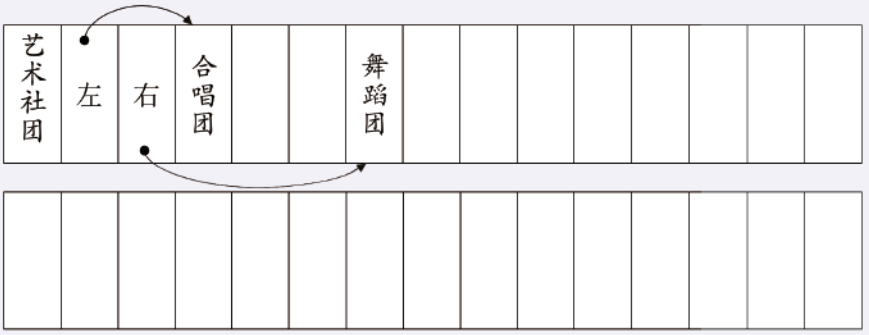


图 1.2.16　艺术社团数据的存储

● 链式存储结构

在数据元素中附加存储一个或多个内存地址，指向逻辑上相邻的数据元素，数据元素之间的逻辑关系不依赖于其所在的内存单元顺序，这种存储结构称为链式存储结构。

链式存储结构的主要优点是灵活，因为可以在数据元素中附加多个内存地址，采用这种方法时，不仅可以实现线性结构的存储，还可以很直观、方便地实现树形结构和图状结构的存储。但链式存储方法的主要缺点是操作比较复杂，而且也需要额外的存储空间来保存链接地址。

● 数据结构

数据结构是由数据的逻辑结构和存储结构构成的，如图 1.2.17 所示。

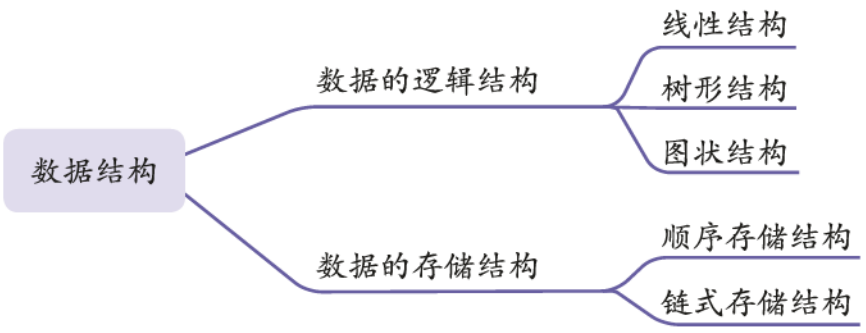


图1.2.17 数据结构

数据的逻辑结构和存储结构并不是一一对应的关系，每一种逻辑结构包括线性结构、树形结构和图状结构，既可以用顺序存储结构来实现，也可以用链式存储结构来实现，至于具体选择哪一种存储方式，则要根据所要解决问题的特点、数据规模和程序运行环境等多种条件来确定。

拓展练习

在学校艺术社团组织结构图上从左到右依次标记数字，如图1.2.18所示。

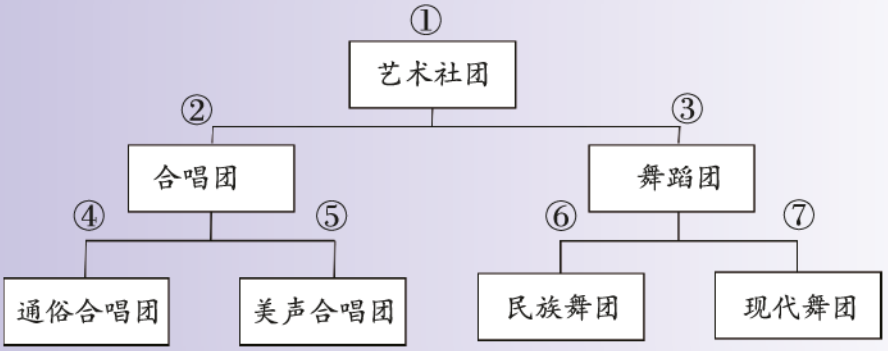


图1.2.18 艺术社团组织结构图 顺序标号

（1）根据顺序标号画出该社团组织的顺序存储结构，完成图1.2.19。

图1.2.19 艺术社团组织顺序存储结构

（2）通过观察上下层顺序标号间的关系，得出已知一个数据元素的地址，求左右分支地址的通用公式。

1.3　认识数据抽象

我们所熟知的俗语“只见树木不见森林”“管中窥豹”和“盲人摸象”都可以用来形容只看到了问题的细节和局部，而没有顾及整体和全局。借助计算机进行问题求解，首先要分析问题的整体特性，为所要处理的数据对象选择合适的数据结构，再根据与数据结构对应的实现方法进行算法设计，最后考虑数据对象的内容细节和计算机软硬件条件，编写高效快速的程序来实现算法。

本节通过完成不同的任务活动来理解什么是抽象，如何用抽象的方式来分析和处理问题，理解抽象的优势。

学习目标

- ★ 理解抽象的概念。
- ★ 了解过程抽象和数据抽象。
- ★ 通过辨析抽象和实现之间的关系，理解抽象的优势，了解抽象数据类型。

我们处理很多事务时，都是分层次进行的。例如，作为某次郊游的组织者，可以先把准备工作分成吃、住、行、玩等几个方面，每个方面分派不同的志愿者负责郊游计划的实施，而组织者则协调时间和费用问题，使其不发生冲突，也不在挑选小吃这样的细节上费心思。只要划分足够的层次，就能控制工作的复杂度，从而成功实施像月球探测这样的巨型工程。

本节围绕“电视机抽象建模”项目开展学习，通过项目活动，了解过程抽象和数据抽象的基本概念，以及抽象应用于建模的优势。本节主要包含“电视机的抽象”和“程序化电视机”两个任务。

在计算机科学中，抽象（abstraction）是一种抛弃局部“物理的（physical）”细节，从整体“逻辑的（logical）”角度来看待事物的方式。所谓“逻辑的”，一般指事物的性质、功能、相互关联等，而“物理的”，则是事物的具体构成、实现细节和运作方式等。

采用抽象方式来处理问题及其解决方案，能够自顶向下层层分解复杂度，更加有条理地解决复杂问题，并通过同一逻辑方案的不同物理实现（implementation），使得问题解决方案具有灵活性和可扩展性。

任务一　电视机的抽象

※ 活动 1　了解用户眼中的电视机

大多数人家里都有电视机，用来收看新闻和影视节目。节目播放时间一到，我们只要找到遥控器，按动上面的按钮，就可以打开电源，切换频道，调整音量，看完电视后，还可以通过遥控器按钮来关闭电视机。

我们注意到，每家每户的电视机在外观、尺寸、内部结构和节目接收方式上会有很多不同，但在收看节目方面，许多人都能够在短时间里熟悉一台电视机的操作方式，熟练地收看节目。

从抽象的角度来看，我们可以说，用户看到的是电视机逻辑的一面。用户使用的是电视机设计者提供的一系列能够收看和切换电视节目的功能，这些功能会通过遥控器上的按钮来提供，电视机身上也有一些按钮，这些按钮也可以被称为操作界面或接口（interface）。

对电视机的操作大部分都通过遥控器来实现，多数电视遥控器都有几个相同的按钮，如电源开关、数字键盘、频道增减、音量增减和切换节目源等，如图 1.3.1 所示。

图1.3.1　电视机的操作界面

从逻辑角度看，电视机具备什么功能？这些功能由哪些接口来提供？将答案写在表 1.3.1 中。

表 1.3.1　电视机基本功能表

序号	电视机的功能	操作按钮
1	开机/关机	电源开关
2	根据号码选择频道	数字键盘
3	切换节目源	
4	切换频道	
5	调节音量	

● 抽象和接口

通过抽象方法，我们可以专注于问题解决方案的顶层逻辑功能设计，从问题的整体和全局出发，定义一系列接口，让用户可以通过调用接口来执行逻辑功能，而无须关心接口的具体物理实现细节。这样，对于用户来说，问题的复杂程度就大大简化了。

※ 活动 2　了解机壳之内的物理实现

同样是一台电视机，家电维修员所看到的可以说是电视机“物理”的一面。家电维修员不仅要知道如何使用电视机，而且需要知道保证电视机功能正常的物理实现细节。

他需要理解节目画面如何被显示、遥控器信号如何被接收、如何连接内部的音箱等。

电视机的物理实现细节都隐藏在机壳之内，如图 1.3.2 所示。

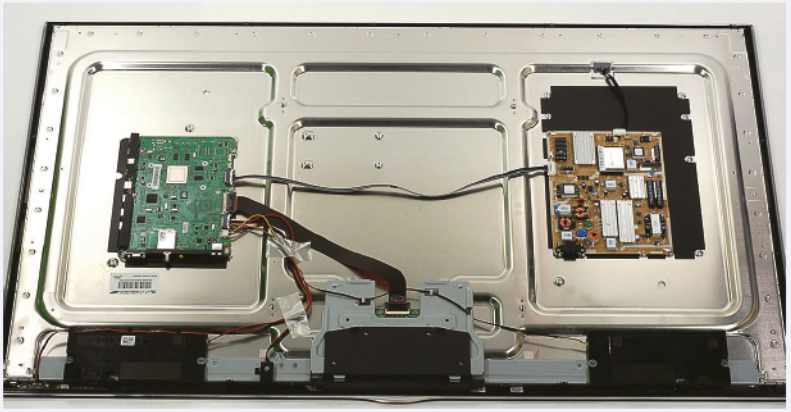


图 1.3.2　机壳里面的秘密

成像模块是电视机画面输出的核心部件，虽然各种电视机的操作界面相似，但形成画面的成像模块却可能差别甚大，阅读表1.3.2中的内容，了解不同成像方式实现的一些细节，试着查找资料，填写表中的空白部分。

表1.3.2 不同成像方式电视机的技术实现对比

类型/特性	成像模块	画面尺寸	显示分辨率
显像管电视机	阴极射线管（CRT）	较小	较低
平板电视机			
投影电视机			

● 程序中的抽象

大多数人使用计算机来上网、听音乐、收发邮件、玩游戏的时候，在乎的是这些程序能否满足需求、是否运行流畅，而对这些程序功能具体是怎么实现的既一无所知，也并不关心。而程序员和系统管理员对这些程序和计算机系统则有更为深入的了解。他们必须知道操作系统如何工作，网络协议在什么配置下运行，用户的鼠标操作如何影响图形界面的更新，以及如何编写程序代码来组合这些功能。

实际上，我们编写程序的时候，也常常需要抽象。Python提供了很多内置函数库，我们可以直接调用这些函数，而无须关心或者了解这些函数功能具体是如何实现的。

例如，我们要计算某个数的平方根，可以在导入Python的math标准模块之后，直接调用math.sqrt()函数来求值，而不用了解这个函数是如何对整数或者浮点数开平方的。

```
>>> import math
>>> math.sqrt(25)
5.0
```

任务二 程序化电视机

※ 活动1 收看虚拟的电视机

如果我们把电视机的逻辑功能接口写成Python函数的形式，那么用户收看一次节目的过程就可以写成一段程序了。

电视机功能接口的定义如表 1.3.3 所示。

表 1.3.3　电视机功能接口定义表

接口	功能定义
TVSet.powerOn()	开启电源
TVSet.shift(source)	切换节目源，可选“TV”“AV”“HDMI”“USB”
TVSet.channel(n)	直接切换频道，n为数字
TVSet.channelUp()	频道号加一切换
TVSet.channelDown()	频道号减一切换
TVSet.volumeUp()	音量增强
TVSet.volumeDown()	音量减弱
TVSet.powerOff()	关闭电源

补充完整下列程序，完成一次节目收看，并在计算机上运行和验证你的程序。

```
#导入电视机模块（见教科书配套资源）
import tv
myTV=tv.TVSet()                    #新建一个电视机对象
myTV.powerOn()                     #开启电源
_____                              #切换到“TV”节目源
_____                              #收看6频道
_____                              #看下一个频道
_____                              #看下一个频道
_____                              #看下一个频道
_____                              #看上一个频道
_____                              #调小音量
_____                              #关闭电源
```

● 过程抽象和数据抽象

如图 1.3.3 所示，调用求平方根函数就是一个过程抽象（procedural abstraction）的例子，我们不需要知道求平方根是怎么运算的，只需要知道这个函数叫什么、如何使用。如果按照说明正确调用函数，那么这个函数就能实现它所声称的功能，而实现细节则被隐藏起来，或者说被抽象了。

同样，如果我们把现实世界中的事物抽象成一类数据对象，就可以只从其逻辑功能来描述这些数据对象的性质、功能和它们之间的关联，

而不涉及这些数据对象的具体实现细节。这就是跟过程抽象相似的数据抽象（data abstraction），如图1.3.4所示。

图1.3.3 过程抽象

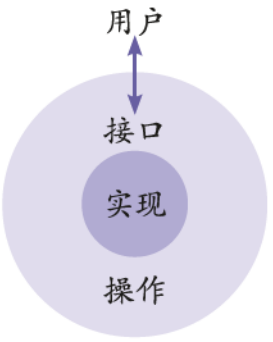


图1.3.4 数据抽象

※ 活动2 了解电视机类TVSet的使用说明

实际上，我们在编写电视收看程序的时候，已经用到了电视机的数据抽象。利用Python语言的面向对象机制，我们在电视机模块tv中定义了一个电视机类TVSet。在教科书配套资源中，对电视机类中的每个方法都做了具体的实现，我们使用电视机类TVSet来编写电视收看程序的时候，并不需要理解这些具体的实现代码，只要按照文档说明来调用它们就可以了。

请从教科书配套资源中找到tv.py文件，查看其中电视机类TVSet的每一个方法说明，了解电视机类TVSet的使用方法，并将每个方法的名称摘抄在表1.3.4中。

表1.3.4 电视机类TVSet的方法

序号	方法名称	功能描述

数据封装

数据抽象实现了数据封装（encapsulation），其优势不仅在于能够把逻辑功能和实现细节分离，分解问题的复杂度，而且还使得数据对象的可扩展性大大增强，在更换逻辑功能的具体实现方案后，只要接口的功能和定义不变，就无须修改上层程序代码。

※ 活动 3　了解虚拟现实头盔

随着计算机交互技术和5G无线通信技术的快速发展，虚拟现实头盔正在为人们带来全新的沉浸式观影体验，它不仅能像传统电视机一样显示一个巨幅画面，还能跟踪检测头部运动，通过实时画面切换，让用户观察到无缝的全景影像节目，具有身临其境的感觉。

与传统电视机相比，虚拟现实头盔在成像方式、交互控制等基础技术实现上都有很大的差别。但虚拟现实头盔在面向用户的功能接口方面，与传统电视机基本相似，我们可以用相同的接口构成程序，让用户完成一次全景影像节目收看。

请从教科书配套资源中找到虚拟现实头盔模块vrhmd，修改前面的电视收看程序，完成全景影像节目收看。

```
#导入虚拟现实头盔模块（见教科书配套资源）
import vrhmd
myTV=vrhmd.TVSet()
myTV.powerOn()
_____________________________    #切换到“VR”节目源
_____________________________    #收看6频道
_____________________________    #看下一个频道
_____________________________    #看下一个频道
_____________________________    #看下一个频道
_____________________________    #看上一个频道
_____________________________    #调小音量
_____________________________    #关闭电源
```

● 抽象的优势

利用抽象方法，我们可以将一个高复杂度的问题分解为若干层次，例如，将整个系统按照不同功能分解为几个子系统，再把每个子系统按照部件结构分解为若干模块，模块内部还可以继续分解下去，直到单个小问题足够简单，再考虑具体的实现方案。

这种层层抽象的细分结构，既有效限制了每个层次上的问题复杂度，又有利于问题解决方案的重复利用，可以做到下层实现细节的变动不影响上层的逻辑功能，具有强大的灵活性和可扩展性。

● 抽象数据类型

抽象数据类型（Abstract Data Type，ADT）是抽象概念在数据结构上的具体应用。将数据对象分析的结果，以逻辑功能接口的形式固定下

来，就成为抽象数据类型。抽象数据类型实现了数据对象的封装，并不涉及数据对象的实现细节，而是通过接口的形式描述了数据的组成和对数据的各种操作。

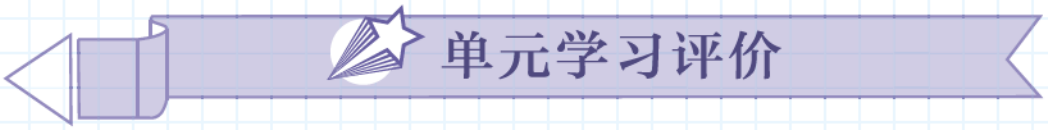
抽象数据类型并不受特定实现和编程语言的约束，可以通过调整实现方法来应对应用需求的变化。抽象数据类型概念的引入，降低了大型软件系统的复杂度，提高了程序的可读性与可维护性，使软件系统的各部分相对隔离，在一定程度上解决了软件可靠性、生产率等方面的问题。

拓展练习

利用所学的物理学知识，编程实现平抛运动（Horizontal Projectile Motion，HPM）类HPMotion，该类提供下列接口：

```
class HPMotion:
  def __init__(self, h, vx):          #初始化接口，参数为高度h和水平速度vx
  #指定时刻t的速度，返回元组（vx, vy）分别是水平、竖直方向的速度值
  def velocity(self, t):
  #指定时刻t的位移，返回元组（dx, dy）分别是水平、竖直方向的位移
  def displacement(self, t):
```

运行程序，通过调用HPMotion类的接口，无须了解内部计算公式，就能够得到指定时刻t的速度和位移值，深入体验数据抽象的概念及其带来的便利。



单元学习评价

本单元学习了数据的含义和价值、数据组织方式和数据抽象的概念，体验了生活中的实际问题的结构特点，以及逻辑功能与物理实现的区别。你理解数据组织与数据抽象的相关概念了吗？你能辨析实际问题中数据的不同组织方式，并会采用自顶向下方法来分析实际问题吗？请参与小组交流并反思，开展自评或小组评价。

一、（多选）以下属于存储结构的是（　　）。

A. 线性结构　　B. 树形结构　　C. 图状结构

D. 顺序结构　　E. 链式结构

二、生活中方方面面的背后都蕴含着各种数据组织方式，请辨析下列事物对应的数据逻辑结构，并画出结构图。

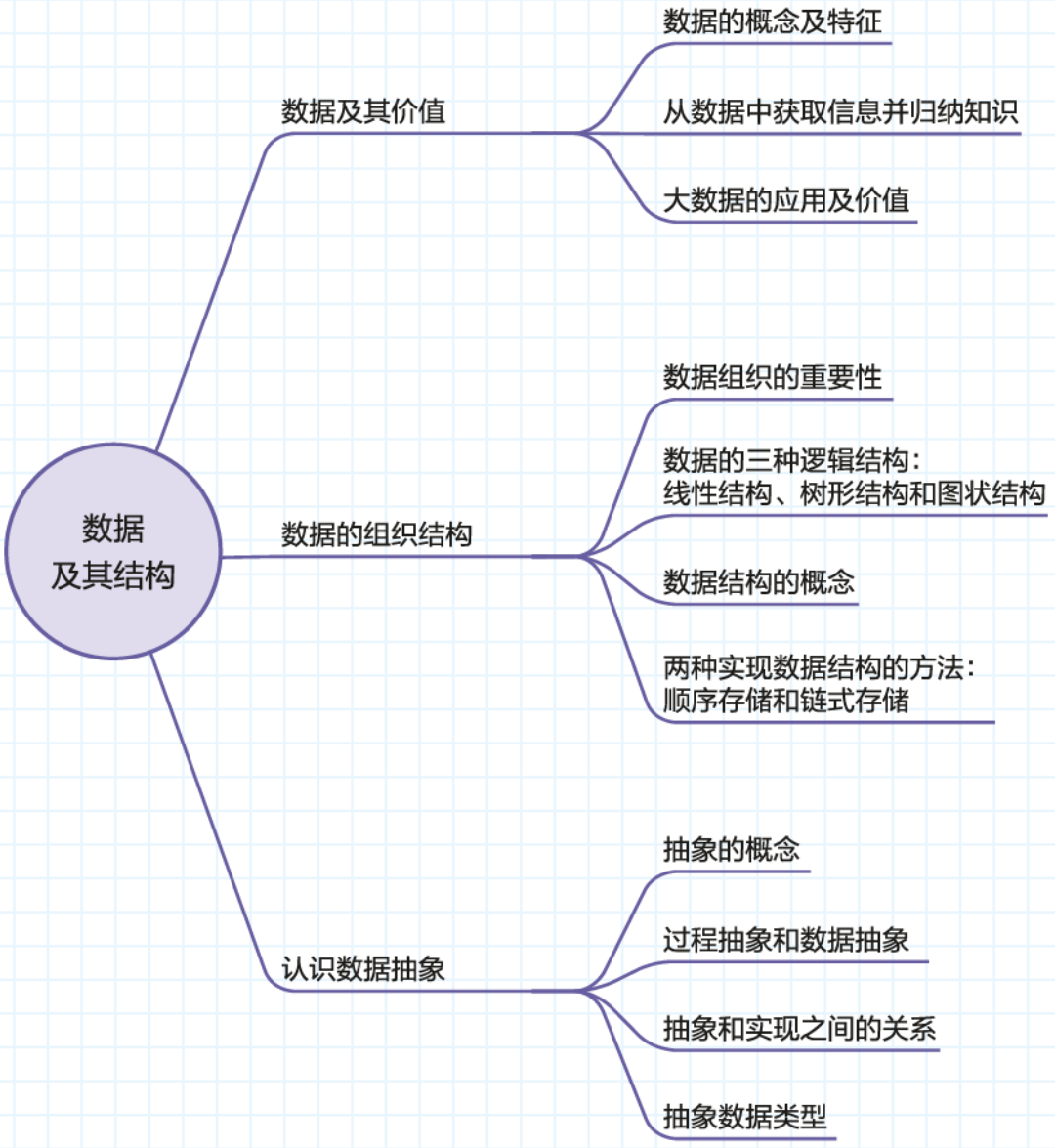
（1）北京、天津、上海、南京、武汉、西安、重庆、广州等8个城市之间的高铁线路：把城市作为数据元素，城市之间的直达高铁线路作为数据元素之间的关系，画出结构图。

（2）太阳、八大行星、各行星的卫星等太阳系天体之间的“围绕运行”关系：把太阳、行星和卫星等天体作为数据元素，天体之间的从属关系作为数据元素之间的关系，画出结构图。

（3）超市收银处排队结账：把顾客作为数据元素，顾客排队的次序作为数据元素之间的关系，画出结构图。

（4）一个家族中男性成员的“父子关系”：把家族男性成员作为数据元素，成员之间的父子关系作为数据元素之间的关系，画出结构图。

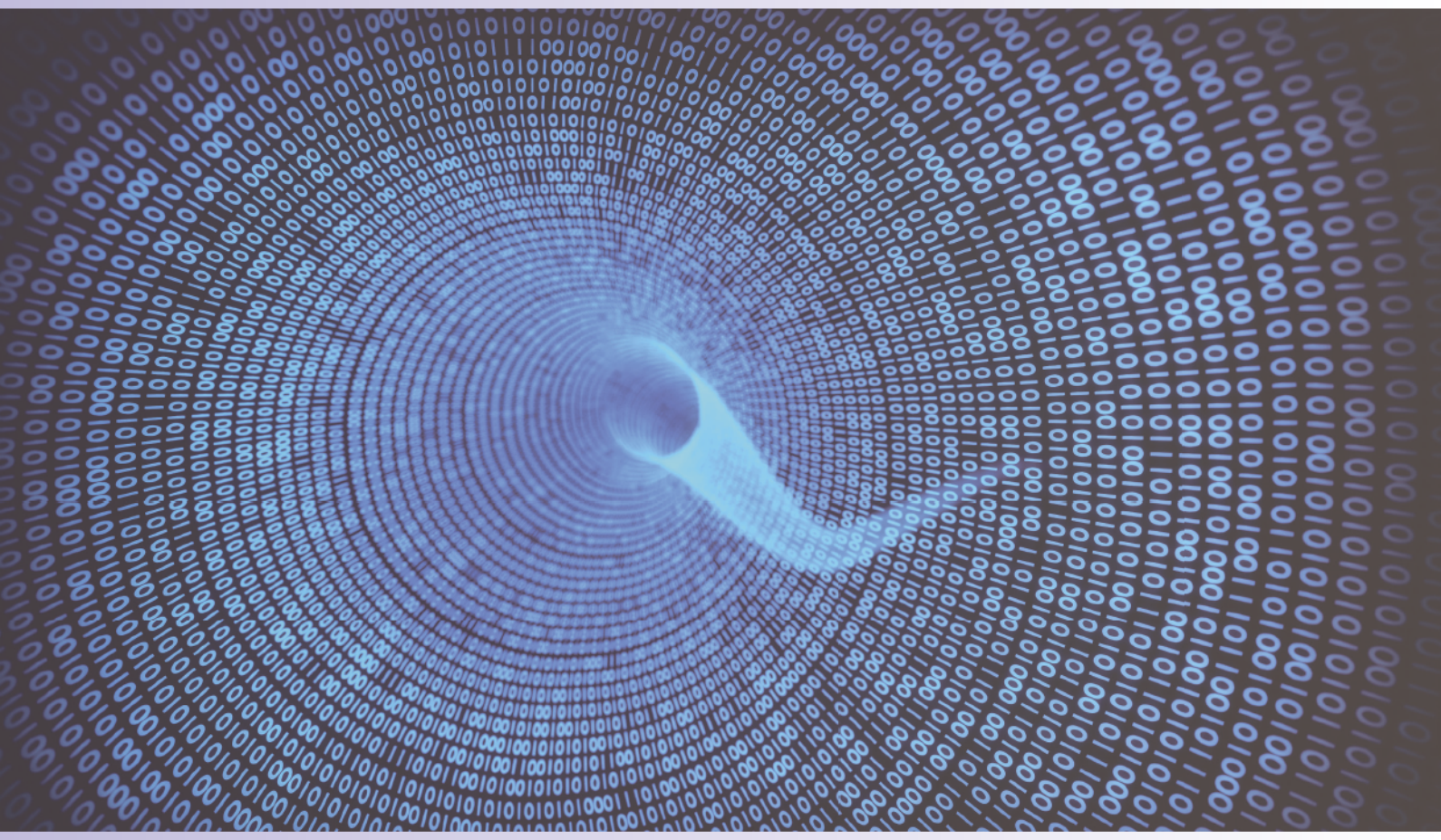
单元学习总结


数据
及其结构
数据及其价值
数据的概念及特征
从数据中获取信息并归纳知识
大数据的应用及价值
数据的组织结构
数据组织的重要性
数据的三种逻辑结构：
线性结构、树形结构和图状结构
数据结构的概念
两种实现数据结构的方法：
顺序存储和链式存储
认识数据抽象
抽象的概念
过程抽象和数据抽象
抽象和实现之间的关系
抽象数据类型

第 2 单元　线性表及其应用

线性表是最基本的线性结构，主要特征是数据元素构成前后相继的列表，它直接来源于计算机中连续编址的存储器构造。线性表结构简单、高效，是许多数据结构的基础。日常生活中的公交线路、排队队列和竞赛积分榜，都是线性表的典型应用。

本单元通过“整理图书”项目理解线性表的基本概念及特征，通过“趣味知识竞赛”项目体会线性表的应用，学习处理线性表的基本算法及利用线性表解决问题的一般方法和过程，通过“破译恺撒密码”项目学习字符串及其应用。



2.1 线性表结构及其实现

生活中许多事物是有顺序的，如公交车沿线设置的车站、图书馆书架上依次摆放的图书。在计算机科学中，人们用线性表来表示依次排列的数据元素以及对这些数据元素的相关操作。

本节主要学习线性表的概念和基本特征、线性表抽象数据类型的定义、线性表的实现方法。

学习目标

- ★ 理解线性表的概念和特征。
- ★ 掌握线性表抽象数据类型的定义。
- ★ 掌握线性表的两种实现方法。
- ★ 理解数组和链表的概念及其特点。

图书是人类文明传承的重要方式。图书馆是知识的宝库，存放了大量的图书。为方便借阅和管理，图书馆把图书有顺序地放置在书架上。借阅和归还图书后，要整理图书，让书架上的图书仍然有顺序。

本节围绕“整理图书”项目展开学习，通过项目活动认识生活中的线性表，学习定义线性表抽象数据类型，并编写代码实现线性表的基本操作。本节主要包含“手工整理图书”和“编程整理图书”两个任务。

任务一 手工整理图书

※ 活动1 认识线性排列

请仔细观察图2.1.1，回答下面的问题。

（1）从左数第1本书的书名是_____，紧挨着它的后一本是_____。

（2）第3本书的书名是_____，紧挨着它的前一本是_____，紧挨着它的后一本是_____。

（3）第8本书的书名是_____，紧挨着它的前一本是_____。

通过观察可以发现，紧挨着每本书的前或后各只有一本书。像这种排列，我们说它是线性的。

思考：如果书架上的图书摆放如图2.1.2所示，它还是线性排列的吗？为什么？

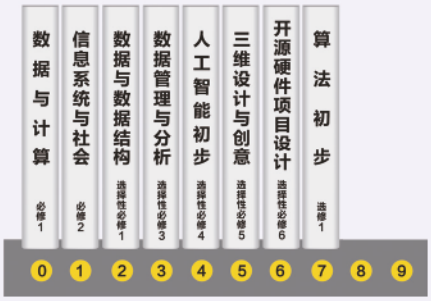


图 2.1.1　书架上的图书示意图1

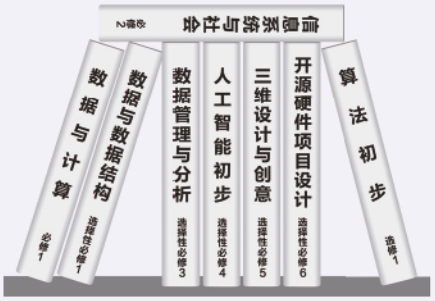


图 2.1.2　书架上的图书示意图2

线性表

线性表中的顺序号从 0 开始，而日常生活中说第几个元素，一般是从 1 开始，比如第 12 个元素的顺序号为 11，请注意区分。

线性表（linear list）按线性结构组织数据元素。在线性表中，数据元素之间存在前后的顺序关系。每个数据元素都有一个顺序号，顺序号是连续的整数。通过顺序号可以访问数据元素。线性表中的数据元素可以是一个数或一个字符，也可以是一个对象。

※ 活动2　整理图书

小明申请了学校图书馆助理的社会实践岗位。他的工作是整理图书馆书架上的图书，使之井然有序。某个书架的图书如图2.1.1所示，小明的任务如下。

（1）添加图书。

图书馆新采购了图书《移动应用设计》，放在书架最右边，请在图2.1.3中相应的书脊上写上书名。

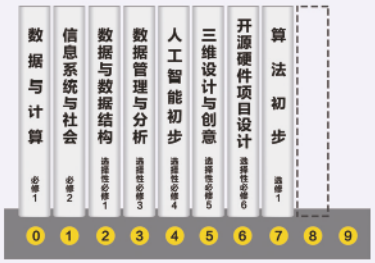


图 2.1.3　添加图书

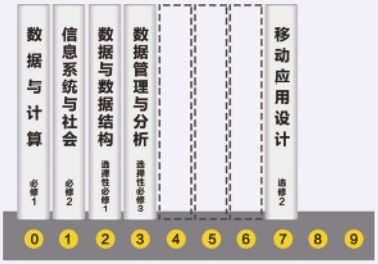


图 2.1.4　借阅图书

（2）借阅图书。

有同学借阅了图书《人工智能初步》，请在图2.1.4中相应的书脊上写上书名。

（3）归还图书。

有同学归还了图书《网络基础》，需将该书放置在图书《数据管理与分析》左面，请在图2.1.5中相应的书脊上写上书名。

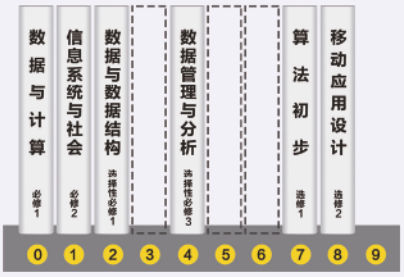


图2.1.5 归还图书

（4）查询图书。

经过以上操作后，图书列表中共有 _____ 本图书，第 8 本书的书名是 _____。

小明在实践中发现，图书的整理工作主要有添加图书、借阅图书、归还图书等。

线性表的特征

在线性表中插入或删除数据元素，该元素之后的数据元素顺序号都将改变。

线性表抽象数据类型

从以上活动可以看出，线性表的基本操作主要包括追加、删除、插入、查询等操作。为了便于在程序中使用线性表解决问题，需要定义线性表抽象数据类型（ADT LinearList），接口如下。

ADT LinearList：

- LinearList ()：创建空线性表。
- appendItem(item)：将数据元素item追加到线性表。
- removeItem(pos)：从线性表中删除pos位置的数据元素。
- getItem(pos)：取得pos位置的数据元素。
- setItem(pos,item)：设置线性表pos位置的数据为item。
- size()：获得线性表中数据元素的个数。
- isEmpty()：判断线性表是否为空。
- insertItem(item,pos)：将item插入表中pos位置。

元素的顺序号也称元素在表中的位置，简称元素的位置。顺序号为pos，也称 pos 位置。

※ 活动3　用线性表实现图书整理

借助抽象数据类型LinearList，可以方便地用线性表实现小明的任务。请补全下面的代码或注释。

```
01. books=LinearList()                          #创建一个空线性表
02. books.appendItem("数据与计算")               #追加“数据与计算”
03. books.appendItem("信息系统与社会")           #追加“信息系统与社会”
04. books.appendItem("数据与数据结构")           #_____
05. books.appendItem("数据管理与分析")           #_____
06. books.appendItem("人工智能初步")             #_____
07. books.appendItem("三维设计与创意")           #_____
08. books.appendItem("开源硬件项目设计")         #_____
09. _____                                       #追加“算法初步”
10. _____                                       #追加“移动应用设计”
11. books.removeItem(4)                         #在位置4删除图书
12. books.insertItem("网络基础",3)               #在位置3插入“网络基础”
13. print(books.getItem(7))                     #显示位置7的图书名称
14. print(books.size())                         #显示图书数量
```

任务二　编程整理图书

※ 活动1　用顺序存储实现线性表

Python的列表采用了顺序存储的方式，可以保存多个数据，并提供了一些好用的方法。利用列表可以方便地实现线性表。列表的第一个存储位置的顺序号为0。

以下用列表items来保存线性表中的数据元素。

（1）创建空线性表。

由构造方法__init__()完成，它的功能是生成一个空的列表items。

```
01. class LinearList:
02.   def __init__(self):
03.     self.items=[ ]              #空列表
```

（2）追加数据元素。

在列表最后添加数据元素，可以使用列表的方法append。

```
04.   def appendItem(self,item):
05.     self.items.append(item)        #在列表最后增加一个元素
```

（3）删除数据元素。

删除线性表指定位置的数据元素，可以使用列表的pop方法。删除数据元素后，它后面的数据元素会向前移，相应的顺序号也会发生变化。在Python中，移动数据的工作由列表自动完成。例如，借阅图书《人工智能初步》的过程如图2.1.6所示。

图2.1.6 借阅图书《人工智能初步》的过程

```
06.   def removeItem(self,pos):        #删除表中pos位置的元素
07.     self.items.pop(pos)
```

（4）插入数据元素。

在线性表中把一个元素插入到指定位置，这个位置及之后的数据元素会向后移动，顺序号发生变化。在Python中，移动数据的工作由列表自动完成。例如，归还图书《网络基础》的过程如图2.1.7所示。

图2.1.7 归还图书《网络基础》的过程

```
08.   def insertItem(self,item,pos):    #把item插入表中pos位置
09.     self.items.insert(pos,item)
```

（5）其他操作。

```
10.   def getItem(self,pos):            #获得位置pos的数据元素
11.     return self.items[pos]
12.   def setItem(self,pos,item):        #设置位置pos的值为item
13.     self.items[pos]=item
14.   def size(self):                    #获取线性表的数据元素个数
15.     return len(self.items)
16.   def isEmpty(self):                 #判断线性表是否为空
17.     return self.size()==0
```

● 顺序表和数组

线性表的顺序存储用一组连续的存储单元依次存储线性表的数据元素。利用这种存储方式实现的线性表叫作顺序表。

如果顺序表中各数据元素占用的存储空间大小相同（比如是同一种类型的数据），这样的顺序表叫数组。各个数据元素叫数组元素，数据元素的序号叫数组下标。如果知道数组的起始存储位置及单个数组元素占用空间大小，各个数组元素的存储位置可以通过计算得到，因而数组具有随机访问的特点，存取数组元素的效率很高。

顺序表的数据元素是连续排列的，插入和删除数据元素都需要移动它后面的元素，这导致操作的代价很高。

将LinearList的顺序存储实现代码输入到文件中，保存为linearList.py。运行任务一活动3中的代码，测试各个接口的效果。

注意：请在活动3的代码第一行添加“from linearList import LinearList”。

※ 活动2　用链式存储实现线性表

用链式存储实现的线性表是一个节点序列。节点中不仅保存数据，还保存下一个节点的存储位置。用链式存储实现的线性表也叫作链表。首先定义链式存储所需要的节点类。

```
01. class Node:
02.   def __init__(self,item):
03.     self.data=item          #数据区
04.     #链接区，指向下一个节点，初始化时为空
05.     self.next=None
```

（1）创建空线性表。

创建空线性表由构造方法__init__()完成。链表需要有一个头节点引用变量head指向第一个节点。新创建的链表还没有节点，所以将head指向空，如图2.1.8所示。

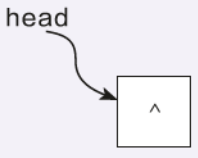


图 2.1.8　创建空链表

```
06. #链式存储实现的线性表
07. class LinearList:
08.   def __init__(self):
09.     self.head=_______       #让头节点引用指向空
```

（2）追加数据元素。

在链表的末端添加节点，首先生成新节点，再找到链表尾节点，最后让尾节点指向新节点。例如，追加图书《移动应用设计》的过程如图

2.1.9所示。请补全下面的代码。

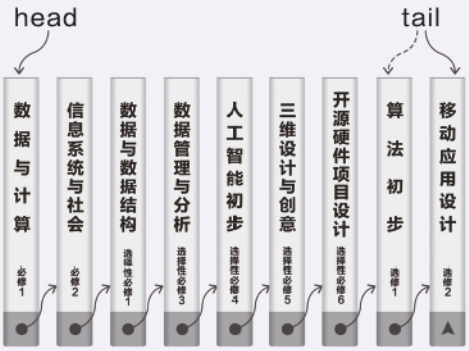


图 2.1.9 追加图书《移动应用设计》

```
10.   #在链表末端添加数据元素
11.   def appendItem(self,item):
12.     temp=Node(item)                    #用item生成节点类对象temp
13.     if self.head==None:                #如果链表为空
14.       self.head=temp
15.       return
16.     tail=__________                    #tail先指向头节点
17.     while tail.next!=None:             #tail还没有指向最后节点
18.       tail=tail.next                   #向后移动
19.     tail.next=__________               #链接新节点
```

（3）删除数据元素。

删除链表中的数据元素，需要先从第一个节点开始向后移，直到指定位置。假设要删除的节点的下一个节点为p。先移动到要删除位置前面的节点previous，再让previous指向节点p。例如，借阅图书《人工智能初步》的过程如图2.1.10所示。请补全下面的代码。

图 2.1.10 借阅图书《人工智能初步》

```
20.   #删除表中pos位置的节点
21.   def removeItem(self, pos):
22.     previous=self.head                 #要删除位置的前一个位置
```

```
23.     if pos==0:                          #如果要删除的是链表头节点
24.       self.head=self.head.next
25.       return
26.     n=0
27.     while n<pos-1:                      #找到要删除的前一个节点
28.       previous=previous.next
29.       n=n+1
30.     previous.next=_________________       #删除节点
```

（4）获取数据元素。

获得链表指定位置的数据元素，需要先从头节点移动到指定位置，再返回该位置的数据元素。请补全下面的代码。

```
31.   #得到pos位置的数据元素
32.   def getItem(self,pos):
33.     current=self.head
34.     count=0
35.     while count<pos:                          #没到指定位置
36.       current=____________                    #指向下一个节点
37.       count=count+1                           #计数加1
38.     return current.data                       #返回数据元素
```

（5）获取数据元素个数。

从头节点遍历整个链表，并计算数据元素节点的个数。

```
39.   #统计链表的节点数
40.   def size(self):
41.     current=self.head                         #指向头节点
42.     count=0
43.     while current!=None:                      #不是链表的结尾
44.       count=count+1                           #节点数增加1
45.       current=____________                    #指向下一个节点
46.     return count                              #返回节点数
```

（6）判断线性表是否为空。

判断线性表是否为空，可以通过判断头节点引用是否为None实现。

```
47.   def isEmpty(self):                          #判断链表是否为空
48.     return self.head==None
```

（7）插入数据元素。

插入节点需要先产生一个新节点，放置要插入的数据元素；然后从头节点移动到插入位置前面的节点previous；最后让新节点指向previous的后一个节点，让previous指向新节点。例如，归还图书《网络基础》的过程如图2.1.11所示。请补全下面的代码。

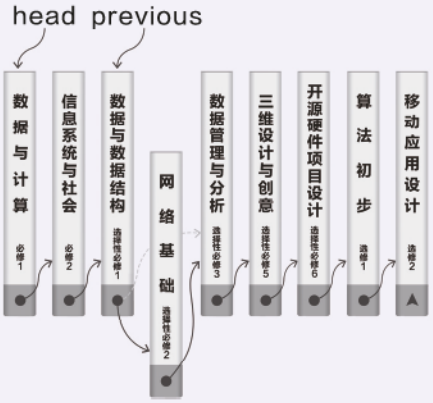


图 2.1.11 归还图书《网络基础》

```
49.                                          #将item插入表中pos位置
50.   def insertItem(self,item,pos):
51.     temp=Node(item)                      #生成新节点
52.     if pos==0:                           #如果在头节点前插入
53.       temp.next=self.head                #新节点指向头节点
54.       self.head=temp                     #头节点指向新节点
55.       return
56.     previous=self.head                   #指向插入位置前一节点
57.     n=0                                  #计数变量置0
58.     while n<pos-1:                       #找到pos位置的前一个位置
59.       n=n+1                              #计数加1
60.       previous=previous.next             #指向下一节点
61.     temp.next=_____                      #指向插入位置处节点
62.     previous.next=temp                   #前一节点指向新节点
```

● 链表

用链式存储实现的线性表叫链表。链表由一系列的节点通过链接串连在一起。

在链表中，相邻节点的存储位置不一定相邻，节点之间的顺序由链接关系决定。在插入和删除数据元素时，并不需要移动节点，所以效率很高。但链表中每个节点都增加了引用信息，需要使用额外的存储空间。另外，在访问数据元素时，要从头节点开始依次向后移动寻找，效率不如顺序表高。

将 LinearList 的链式存储实现代码输入到文件中，保存为 linkList.py。运行任务一活动 3 中的代码，测试各个接口的效果。

注意：请在活动3的代码第一行添加“from linkList import LinearList”。

拓展练习

1. 小明注意到，查询图书的位置也是常见的任务。比如经常有人问：有《雷雨》吗？在哪里能找到？请完善顺序表和链表的代码，实现接口search(bookName)，这个方法返回图书bookName的序号。如果没有这本书，则返回序号-1。

例如，对于图2.1.11所示的数据来说，search("网络基础")的结果为4，search("Java程序设计")的结果为-1。

想一想：这个方法用-1表示没有找到，用其他的正整数可以吗？

2. 为了更好地学习英语，小华希望有一个可以实现以下功能的生词本：

（1）遇到生词可以添加到生词本中；

（2）可以查找单词是否在生词本中；

（3）如果已经学会了，可以把单词从生词本中删除；

（4）可以统计一共有多少生词；

（5）生词本中的单词按词典上的顺序排列。

提示：单词也可以直接比较大小，在词典中位置靠前的小，位置靠后的大。

2.2 随机抽取问题

“随机抽取”在日常生活中的应用比较广泛，如知识竞赛中抽取题目、某项活动中抽取幸运观众等。随机抽取的内容既可以是数字，也可以是名字。随机抽取的应用，既增加了活动的趣味性，又体现了公平性。

本节主要学习什么是随机抽取问题，利用线性表解决随机抽取问题的方法、算法及程序实现。

学习目标

★ 理解随机抽取问题的概念和实现过程。

★ 掌握利用线性表解决随机抽取问题的方法。

★ 体会线性表在问题解决中的作用和意义。

学校组织的每一届趣味知识竞赛活动，都会吸引大批同学参加。在竞赛活动过程中，通过随机抽取幸运观众并随机抽题答题赢礼物的环节更是将整个竞赛活动推向高潮。

本节围绕“趣味知识竞赛”项目展开，通过项目活动了解随机抽取问题，学习利用线性表解决随机抽取问题，体会数据结构在解决问题中的作用和意义。本节主要包含“体验随机抽取”和“编程实现随机抽取”两个任务。

任务一 体验随机抽取

※ 活动1 随机抽取幸运观众

知识竞赛活动中抽取幸运观众的环节，就是在参加活动的现场观众中随机抽取一定数量的人作为幸运观众，可以通过抽取观众的编号或者抽取观众的姓名来实现。本任务中，我们通过抽取观众的编号来

实现。抽取规则如下：①幸运观众数量少于观众总数量；②被抽中的观众编号不再参加此后的抽取活动。

假设现场有200名观众，观众的编号范围是1~200，需要从中随机抽取10名幸运观众，抽取过程如下：

第1次抽取，在200名观众中随机抽取1名幸运观众，如果抽到了180号，那么180号就不能出现在下一次的抽取过程中；

第2次抽取，在199名观众中随机抽取1名幸运观众，如果抽到了88号，那么88号就不能出现在下一次的抽取过程中；

第3次抽取，在_____名观众中随机抽取；

以此类推，最后一次抽取，是在_____名观众中随机抽取。

※ 活动2　随机抽取竞赛题

知识竞赛中随机抽取题目环节的规则如下：①题目数量和参赛选手数量相同；②抽中的题目不再放回，后面的选手在剩下的题目中随机抽取。题目的初始状态如图2.2.1所示。

1	2	3	4	5	6	7	8	9	10
题A	题B	题C	题D	题E	题F	题G	题H	题I	题J

图2.2.1　题目初始状态

第1次抽取：假设第1名选手通过摇号的方式在1~10号码中摇到了3号，那么第3题“题C”被抽中并从预抽取题目中删除。第1次抽取后，剩下9道题可供其他选手抽取，抽取后的状态如图2.2.2所示。

1	2	3	4	5	6	7	8	9
题A	题B	题D	题E	题F	题G	题H	题I	题J

图2.2.2　第1次抽取后的状态

第2次抽取：假设第2名选手通过摇号的方式在1~9号码中摇到了6号，那么第6题“题G”被抽中并从预抽取题目中删除。第2次抽取后，剩下8道题可供其他选手抽取，抽取后的状态如图2.2.3所示。

1	2	3	4	5	6	7	8
题A	题B	题D	题E	题F	题H	题I	题J

图2.2.3　第2次抽取后的状态

第3次抽取：假设第3名选手通过摇号的方式在1~8号码中摇到了8号，那么第8题“题J”被抽中并从预抽取题目中删除。第3次抽取后，

剩下7道题可供其他选手抽取，抽取后的状态如图2.2.4所示。

1	2	3	4	5	6	7
题A	题B	题D	题E	题F	题H	题I

图 2.2.4　第3次抽取后的状态

假设后面的2位选手依次抽取了第1题、第5题，请按照上面的思路，画出每次抽取后的状态。

如果按照上述规则抽题，第6位选手抽题时的摇号范围是什么？

● 随机抽取

随机抽取是指在一个有限数量的数据集范围内，随机选择其中的某一个数据元素的过程。我们重点学习的是无放回的随机抽取，其基本原则是：上一次抽中的数据元素，不能出现在下一次的抽取过程中。

随机抽取分为有放回抽取和无放回抽取两种。有放回抽取允许同一个数据元素被抽取多次，而无放回抽取只允许同一个数据元素被抽取一次。

任务二　编程实现随机抽取

知识竞赛活动中，主要实现随机抽取幸运观众和随机抽取竞赛题目。

※ 活动1　建立数据结构

经过观察分析，在任务一的活动中通过每次抽取位置来获取幸运观众和竞赛题目，因此可以利用线性表来组织数据。

创建线性表对象stulist和contestlist，分别存放观众编号和竞赛题目。请补全下面的代码。

```
01. import random                                    #导入random库
02. from linearList import LinearList                #导入线性表
03. stulist=_____________                  #创建线性表对象，存放观众编号
04. contestlist=_____________              #创建线性表对象，存放竞赛题目
```

※ 活动2　设计算法

假设需要从m位观众中随机抽取n位幸运观众，由他们来抽取n道题目，根据知识竞赛随机抽取幸运观众和题目的规则，实现随机抽取的算法描述如下。

（1）需要进行n次抽取，抽取的步骤如（2）（3）（4）所示，如果未完成则继续抽取。

（2）根据可抽取观众总人数生成一个随机数，根据这个随机数抽取幸运观众编号，删除被抽中的观众编号。

（3）根据可抽取题目总数生成一个随机数，根据这个随机数抽

取题目，删除被抽中的题目。

（4）显示抽取结果。

根据上述算法，定义随机抽取函数randSelect(stulist,contestlist)，参数stulist表示待抽取的观众编号，contestlist表示待抽取的题目。请补全下面的代码。

```
#随机抽取函数
def randSelect(stulist,contestlist):
    n=contestlist.size()                          #确定抽取次数
    for i in range(n):
        #生成随机数确定观众编号位置
        randstu=random.randrange(stulist.size())
        stunum=stulist.getItem(_______)              #获取观众编号
        stulist.removeItem(randstu)               #删除抽中的观众编号
        #生成随机数确定题目位置
        randtest=random.randrange(contestlist.size())
        testnum=contestlist.getItem(randtest)          #获取题目
        contestlist.removeItem(_______)             #删除抽中的题目
        #显示抽取结果
        print("第",stunum,"号观众抽取的题目为:",testnum)
```

※ 活动3　编程实现

假设参加本次知识竞赛活动的观众人数是200人，观众的编号是1~200，随机抽取的题目是以下10个关键词所指定的题目：唐诗、互联网、微信、雾霾、人工智能、二维码、虚拟现实、3D打印、物联网、网络爬虫。请补全下面的代码。

```
for i in range(1, 201):
    stulist.appendItem(_____)              #将观众编号存入线性表中
contestlist.appendItem("唐诗")             #将题目存入线性表中
contestlist.appendItem("互联网")           #将题目存入线性表中
contestlist.appendItem("微信")             #将题目存入线性表中
contestlist.appendItem(_____)              #将题目存入线性表中
contestlist.appendItem(_____)              #将题目存入线性表中
contestlist.appendItem(_____)              #将题目存入线性表中
contestlist.appendItem(_____)              #将题目存入线性表中
```

```
28. contestlist.appendItem(_____)          #将题目存入线性表中
29. contestlist.appendItem(_____)          #将题目存入线性表中
30. contestlist.appendItem(_____)          #将题目存入线性表中
31. randSelect(_______,_______)            #调用随机抽取函数
```

将上述代码输入到一个文件中，运行该程序，显示随机抽取的幸运观众编号和题目。

拓展练习

1. 课堂中，教师经常会通过随机点名的方式指定某位同学来回答问题。请利用线性表编程实现随机抽取同学姓名，解决随机点名的问题。

具体要求如下：

（1）能够实现随机抽取同学姓名；

（2）能够统计并显示被抽中点名的次数；

（3）同一个同学被抽中点名不能超过3次。

2. 随机抽取分为有放回抽取和无放回抽取两种情况，请分析说明在利用线性表解决这两种抽取情况的过程中，对于线性表的操作有哪些不同。第1题中的随机点名问题属于哪种抽取情况？为什么？

2.3　字符串应用

字符串是一种由字符构成的线性结构，被广泛应用在程序的输入输出和语言文字处理中。例如，在文本编辑软件中，利用“查找”命令，可以在指定的文本信息中查找特定形式的字符串；在邮件过滤器中，根据事先定义的字符串属性特征，通过获取电子邮箱地址、标题及正文来识别垃圾邮件。另外，在文本挖掘、机器翻译、信息检索、问答系统和对话系统等自然语言处理技术中，也随处可见字符串的身影。

本节主要学习字符串及其相关的概念、字符串抽象数据类型的定义、字符串处理的基本方法。

学习目标

- ★ 理解字符串及其相关的概念。
- ★ 掌握字符串抽象数据类型的定义。
- ★ 掌握字符串处理的基本方法。

明文：利用通用语言明确表达的文字内容。

密文：由明文经变换而形成的用于密码通信的一串符号。

加密：把明文按约定的变换规则变换为密文的过程。

解密：用约定的变换规则把密文恢复为明文的过程。

破译：根据密文进行分析以找出密码变换规则的过程。

密钥：明文转换为密文或将密文转换为明文的算法中的参数。

密码技术作为信息安全的核心技术，不仅可以保证信息的机密性、完整性和可用性，还可以防止信息被篡改和伪造。它被广泛应用于日常生活和工作学习的方方面面，比如银行卡取款、网上支付系统、无线网络的连接和网站用户登录等都离不开密码技术。

本节围绕“破译恺撒密码”项目展开学习，通过项目活动熟悉字符串的基本操作，理解字符串抽象数据类型的定义，并利用字符串的基本方法编程实现破译恺撒密码的操作。本节主要包含“体验手动破译恺撒密码”和“编程实现破译恺撒密码”两个任务。

任务一　体验手动破译恺撒密码

小明、Stuart和Flora三人组队参加了学校组织的暑期拓展训练。

在规定的时间内完成闯关任务最多的小组得分最高，闯关任务的关键线索隐藏在恺撒密码中。

※ 活动1 体验恺撒加密

恺撒加密作为一种最为古老的加密技术，在古罗马的时候就已经很流行，它通过把字母移动一定的位数来实现加密。明文中的所有字母都在字母表上向左（或向右）按照某个位数进行偏移后被替换成密文，其中的位数就是恺撒密码加密和解密的密钥。

小明和队友遇到的第一个闯关任务是根据明文和密钥，写出恺撒加密后的密文。

明文：Imagination is more important than knowledge.

密钥：3

当密钥为3时，所有字符向左偏移3位，加密时明文里所有的字母A将被替换成X，B变成Y，以此类推，X将变成U，Y变成V，Z变成W，如图2.3.1所示。

明码表	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
密码表	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W

图2.3.1 密钥为3的恺撒加密示意图

请补全加密后的密文：

_____ fp jlob fjmloqxkq _____

※ 活动2 体验恺撒解密

小明和队友遇到的第二个闯关任务是根据密文和密钥，写出恺撒解密后的明文。

密文：Fglzafy yjwsl osk wnwj suzawnwv oalzgml wflzmkaske

密钥：8

当密钥是8的时候，解密时密文里所有的字母A将被替换成I，B变成J，以此类推，X将变成F，Y变成G，Z变成H，如图2.3.2所示。

明码表	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
密码表	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R

图2.3.2 密钥为8的恺撒解密示意图

请补全解密后的明文：

_____ great was ever _____ without _____

● 字符串及其相关概念

字符串：是一种由字符构成的线性结构。上面活动中的明文和密文均是字符串，字符串中的每个字符从左到右依次排列，且有确定的位

置。字符串中第一个字符的位置下标是0，可以根据字符在字符串中的位置下标来访问字符。

字符串长度：字符串所含字符的总数称为字符串的长度，长度为0的字符串称作空字符串或空串。

子串：字符串中某一连续的片段称为字符串的子串。任何字符串都是自己的子串。

如图2.3.3所示，字符串s="Data Structure" 由14个字符构成，因此字符串长度为14，其中位置下标为5的字符是“S”，“ata”是字符串s的一个子串。

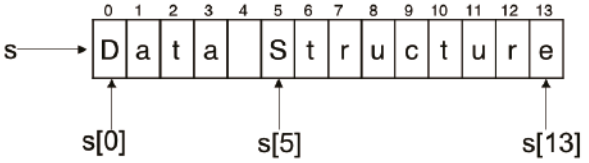


图 2.3.3　字符串示意图

※ 活动3　手动破译恺撒密码

经过齐心协力的合作，小明和队友轻松地完成了前两个闯关任务，第三个闯关任务小明和队友需携带某件神秘物品来换取食物补给。有关这件神秘物品的信息隐藏在一串无空格和标点的恺撒密文中。

密文：pqrabkqzxoapqrxoq

线索：信息中包含了队友Stuart的名字，不区分大小写，共有17个字符。

恺撒加密技术的密钥是有限的，只有1~25，所以可以用穷举的方法来破译密文。小明提议分三步操作来破译密文。

（1）对“stuart”加密，迭代循环，密钥从1到25。

（2）在任务给的密文里面查找“stuart”的密文。

（3）如果存在即可以知道密钥，进而破译密文；否则密钥增加1，继续进行操作（1）。

当密钥为1时，“stuart”的密文为“rstzqs”。手动的字符串查找过程如下：

字符串“pqrabkqzxoapqrxoq”为目标串T，字符串“rstzqs”为模式串P。分别写在等间距格子的纸带上，字符串T的纸带固定不动，字符串P的纸带的首字符与字符串T的首字符对齐，如图2.3.4所示。

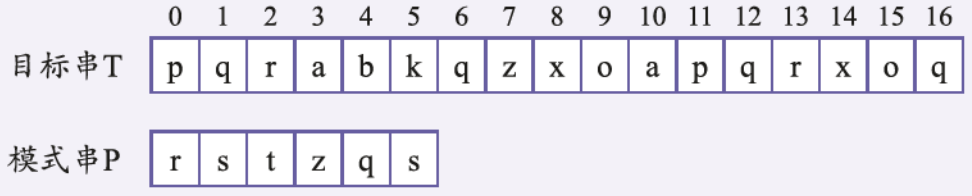


图 2.3.4　字符串查找初始

第1轮匹配：从目标串T的第一个字符开始，将T和P对应的字符逐个依次比对。如图2.3.5所示，第一个字符“p”和“r”不相同，此轮匹配失败。

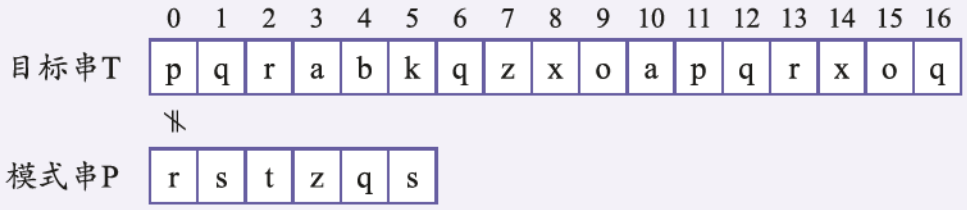


图2.3.5 第1轮匹配

第2轮匹配：将模式串P向右移动一个字符，继续将T和P对应的字符逐个依次比对。如图2.3.6所示，比较的字符对不相同，此轮匹配失败。

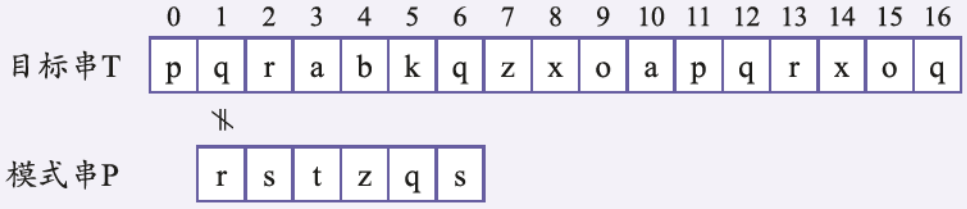


图2.3.6 第2轮匹配

第3轮匹配：将模式串P向右移动一个字符，继续将T和P对应的字符逐个依次比对。如图2.3.7所示，比较的首字符相同，但下一个字符对“a”和“s”不相同，此轮匹配也失败。

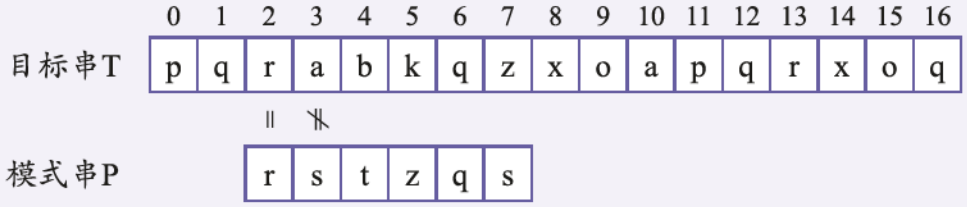


图2.3.7 第3轮匹配

第4轮匹配：将模式串P向右移动一个字符，继续将T和P对应的字符逐个依次比对。如图2.3.8所示，比较的首字符不相同，此轮匹配也失败。

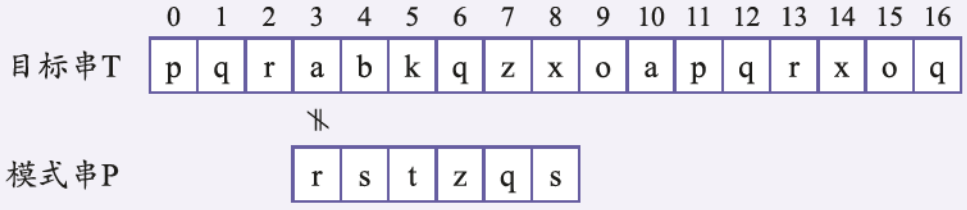


图2.3.8 第4轮匹配

第5~17轮匹配：每轮匹配均失败。

当密钥为2时，“stuart”的密文为“qrsypr”，手动查找字符的结果是字符匹配失败。

当密钥为3时，“stuart”的密文为“pqrxoq”，手动查找字符的结果是在第12轮字符匹配成功。

小明他们找到了密钥，经过解密，密文对应的明文是_____。小明和队友按照提示，顺利通过了第三关任务。

通过体验手动破译恺撒密码，可以总结出字符串匹配的方法。

（1）字符串匹配检测需要进行多轮，首轮匹配检测的起始端从目标串T的第一个字符开始。

（2）将目标串T和模式串P的对应字符逐个依次比对，如果所有字符都相同，匹配成功，操作结束；否则只要有任一个字符不同，本轮匹配失败，执行（3），进行下一轮匹配检测。

（3）若T和P的右边界已对齐，整个匹配检测过程失败；否则，将匹配检测的起始端在T中后移一位，执行（2）。

● 字符串匹配

在字符串中检测查找特定形式的字符串的这类操作都属于字符串匹配（string matching）。即给定目标串T和模式串P之后，判定T中是否存在某一子串与P相同，如果存在则匹配成功。字符串匹配的应用非常广泛，比如垃圾邮件的检测、搜索引擎关键字的查询、网络热搜排行榜的更新等都会用到字符串匹配操作。

● 字符串抽象数据类型的定义

为了方便用字符串来解决问题，需要定义字符串抽象数据类型。我们为字符串抽象数据类型定义了如下接口。

ADT String：

- String(value)：建立一个字符串对象，value是字符序列或者其他类型对象。
- isEmpty()：判断字符串是否是空串。
- size()：获取字符串的长度。
- split(sep)：根据分隔符sep将字符串进行分隔，并返回分隔后字符串的列表。
- strip()：返回去掉前后空格之后的字符串。
- find(sub)：返回sub在字符串中第一次出现的位置，如果没有则返回−1。
- replace(old,new)：返回将所有old子串替换为new后的字符串。
- count(sub)：返回sub在字符串中出现的次数。
- concat(another)：返回与another连接后的字符串。
- substring(start,end)：返回起始位置为start，结束位置为end的子字

符串。

- getItem(index)：返回位置为index的字符。

※ 活动4 体验字符串基本方法的操作

Python内置的字符串类型str是字符串抽象数据类型的一种具体实现，对于抽象数据类型中定义的每个接口，Python的字符串类型都有对应的实现，可以根据需要直接调用。

若字符串s="imagination is more important than knowledge"，如图2.3.9所示，请完成表2.3.1。

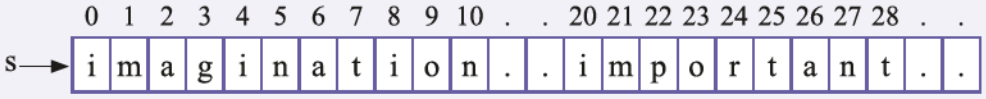


图2.3.9 字符串s

表2.3.1 字符串基本方法的操作实例

ADT String	操作	输出
String(value)	s=str("imagination is more important than knowledge") 或 s="imagination is more important than knowledge"	"imagination is more important than knowledge"
isEmpty()	s==""	False
size()	len(s)	44
split(sep)	s.split()	["imagination","is", "more","important","than", "knowledge"]
strip()	s.strip()	"imagination is more important than knowledge"
find(sub)	s.find("import")	20
replace(old,new)	s.replace("more","less")	
count(sub)	s.count("import")	1
concat(another)	s+" Data Structure"	
substring(start,end)	s[3:10]	
getItem(index)	s[20]	

任务二　编程实现破译恺撒密码

小明和队友们顺利完成了第三关任务，进入到训练的最后一关。这一关的任务要求也隐藏在密文中。考虑到密文较长时手动破译恺撒密码低效且正确率不高，小明和队友们决定尝试通过编程来实现破译恺撒密码，提高准确率和效率。

密文：sohdvhzulwhwkhqdphriwkhiluvwsurjudpphulqwkhzruog。

线索：明文中包含有“programmer”。

※ 活动1　建立数据结构

根据字符串的抽象数据类型的定义，首先创建两个字符串对象，分别来保存已知密文目标串T和线索模式串P。

```
#密文目标串初始化
tStr="sohdvhzulwhwkhqdphriwkhiluvwsurjudpphulqwkhzruog"
pStr="programmer"     #线索模式串初始化
```

※ 活动2　算法设计与实现

如前所述，利用字符串的基本操作实现破译恺撒密码的算法描述如下。

（1）对已知线索字符串加密，密钥初始值为1。

（2）在任务二给出的密文里面查找线索字符串的密文。

（3）如果存在即可以知道密钥，进而破译密文；否则密钥增加1，继续进行操作（1）。

根据以上算法，定义加密函数enCrypt(message,key)，参数key是密钥，message是明文，根据密钥key对明文message进行加密。若左移后的位置大于0则直接左移，否则需要回到字母表末尾继续计算移动位置，这里用对26求余得出移动位置。请补全下面的代码。

```
#加密函数
def enCrypt(message,key):                    #用key对message加密
  alphabet='abcdefghijklmnopqrstuvwxyz' #字母表
  encrypted=''                               #加密后的密文
  for char in message:                       #对明文字符依次加密
    if char not in alphabet:                 #非字符不变
      encrypted=encrypted+char
    else:                                    #字符加密
      rotatedIndex=alphabet.index(char)-key  #左移位数
      #位数对26求余
      encrypted=encrypted+_____________
  return encrypted
```

在任务二给出的密文里面查找线索字符串的密文可以通过字符串匹配算法来实现，具体描述如下。

（1）字符串匹配检测需要进行多轮，首轮匹配检测的起始端从目标串T的第一个字符开始。

（2）将目标串T和模式串P的对应字符逐个依次比对，如果所有字符都相同，匹配成功，算法结束；否则只要有任一个字符不同，本轮匹配失败，执行（3），进行下一轮匹配检测。

（3）若T和P的右边界已对齐，整个匹配检测过程失败；否则，将匹配检测的起始端在T中后移一位，执行（2）。

以上算法可以用嵌套的for循环实现字符串匹配过程，外层的for循环实现每轮匹配检测向右移动，内层的for循环实现依次比对字符。以下代码是通过函数stringMatch(t,p)实现字符串匹配的过程。请补全下面的代码。

```
#字符串匹配函数
def stringMatch(t,p):
  n=len(t)                          #目标串长度
  m=len(p)                          #模式串长度
  for i in range(n-m+1):            #字符串匹配过程
    for j in range(m):              #字符比对过程
      if _________________          #字符不同
        break
    else:                           #匹配成功
      return i
  return -1                         #匹配失败
```

最后还需要定义解密函数deCrypt(message,key)，参数key是密钥，message是密文，依据密钥key对密文message进行解密。若右移后的位置小于26则直接右移，否则需要回到字母表首位继续计算移动位置，这里用对26求余得出移动位置。请补全下面的代码。

```
#解密函数
def deCrypt(message,key):           #用key对message进行解密
  alphabet='abcdefghijklmnopqrstuvwxyz'              #字母表
  decrypted=''                      #解密后的明文
  for char in message:              #对密文字符依次解密
    if char not in  alphabet:       #非字符不变
      decrypted=decrypted+char
    else:                           #字符右移
      rotatedIndex=_________________                 #右移位数
      #位数对26求余
      decrypted=decrypted+alphabet[rotatedIndex % 26]
```

```
38.    return decrypted
```

输出破译后的明文的程序代码如下所示。

```
39. #输出明文
40. for key in range(1,26):              #密钥依次递增
41.   #密文字符串匹配
42.   if stringMatch(tStr.lower(),enCrypt(pStr,key))!=-1:
43.     print(deCrypt(tStr.lower(),key))  #解密密文
44.     break
45. else:
46.   print("can not find it")
```

线性表和字符串都是线性结构，元素之间有先后关系，但字符串中每个元素只能存储字符，而线性表则可以存储任意类型的数据元素。

拓展练习

1. 现有目标串 T ="aaaaaaaaaab" 和模式串 P ="aaaaac"，思考并完成下面的任务。

（1）如果按照学习过的字符串匹配方法，需要做多少次比较?

（2）根据模式串的特征，改进字符串匹配方法，使得下一轮字符匹配向右移动更多字符，减少匹配轮数和比较次数，并编程实现。

2. 破译恺撒密码最多需要尝试25次，密码很容易被破译。除了恺撒密码外，还有替代密码，是古典密码中用到的最基本的处理技巧之一 。替代密码是指先建立一个替换表，加密时将需要加密的明文依次通过查表，替换为相应的字符，明文字符被逐个替换后，生成无任何意义的字符串，即密文，替代密码的密钥就是其替换表 ，举例如表2.3.2所示。

表 2.3.2　替代密码实例

字母表	"abcdefghijklmnopqrstuvwxyz"
密钥	"badcfehgjilknmporqtsvuxwzy"
明文	"hello world"
密文	"gfkkp xpqkc"

破译替代密码最多需要尝试26!次，大大提升了破译难度。请利用上面的密钥，编程实现替代密码的加密过程。

单元学习评价

本单元学习了线性表和字符串的概念、特征及两种实现方法，体验了利用线性表和字符串解决实际问题的基本方法和过程。你了解线性表和字符串的相关概念了吗？你是否可以利用线性表和字符串来解决相应的问题？请参与小组交流并反思，开展自评或小组评价。

一、（多选）以下关于线性表的描述正确的是（　　）。

A. 字符串是单个文本字符的线性表

B. 线性表有顺序存储和链式存储两种实现方法

C. 线性表的主要特征是数据元素构成前后相继的列表

D. 线性表直接来源于计算机中连续编址的存储器构造

二、字符频率可以提高破译替代密码的效率。经统计，英文字符频率从高到低依次为：E，T，A，O，I，N，S，H，R，D，L，C，U，M，W，F，G，Y，P，B，V，K，J，X，Q，Z。

下面一段字符串是经过替代密码加密的密文，其中最高频率字符对应字符E，最低频率字符对应字符Z，不区分大小写。请编程统计密文中每个字符出现的频率，结合以上英文字符频率规律，获取密钥，破译密文。

Uenule ltwfdec ta Hepmnws Utuesa oi ade 1960h, mnse adti dtlg t reiawsp tfn, ydei de vovoclp atlkec tbnwa rdolcsei whoif rnmuwaesh th oihaswmeiah gns letsioif tic gns eidtiroif rsetaovoap, oiinvtaoni, tic "rnirseaoxoif" rnmuwataonitl adoikoif.

1. 问题分析。

通过统计分析密文中各字符出现的频率，并与标准字符频率比对，是否可以初步获取密钥？

_____

2. 设计数据结构。

（1）采用什么数据结构保存数据？

_____

（2）为什么采用这种数据结构？

_____

3. 设计算法。

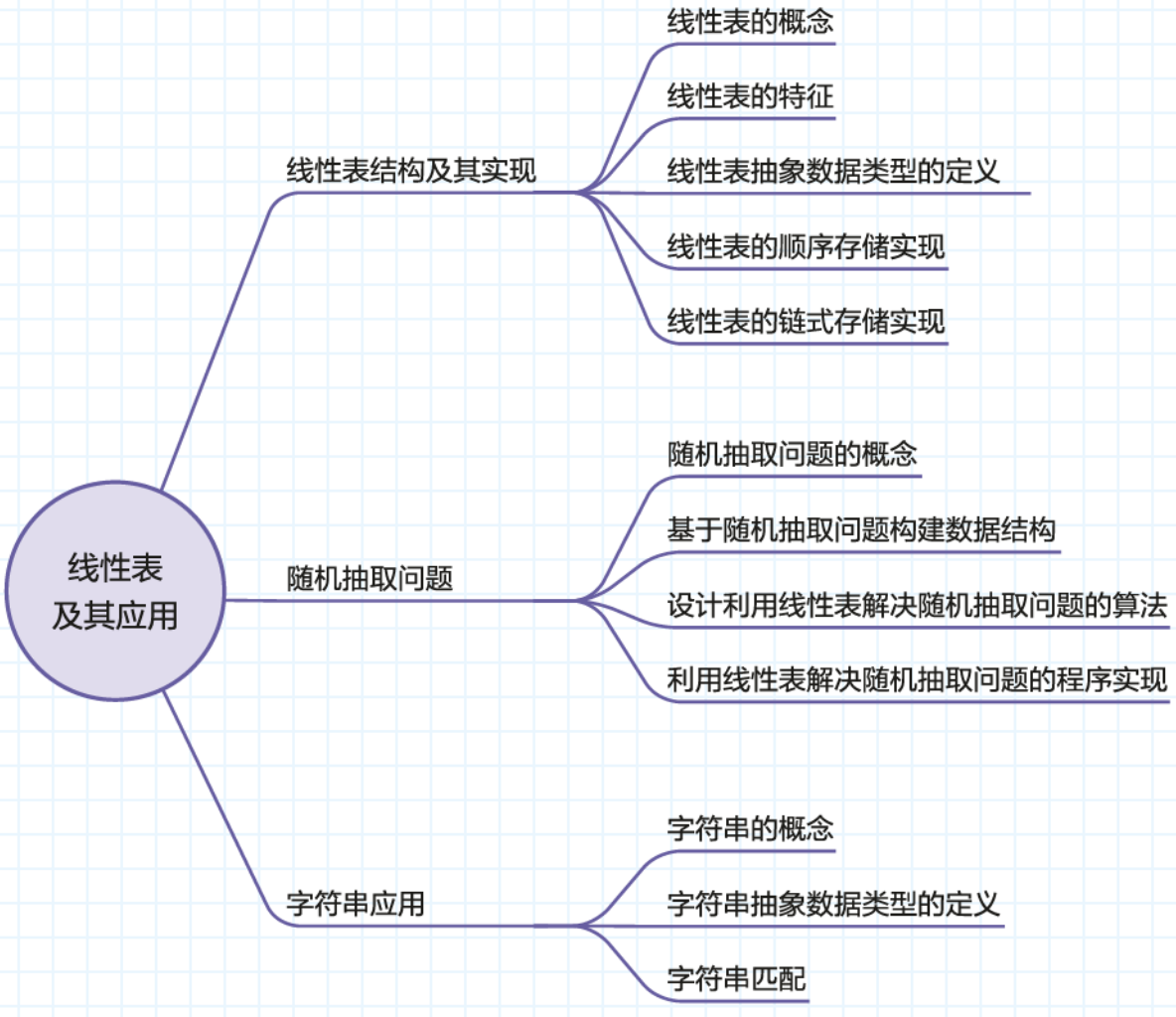
（1）怎样输入、输出数据？

_____

（2）算法流程图。

4. 编程实现。

单元学习总结



第 3 单元　数据的排序与查找

为了高效地处理数据，需要将线性表中的数据整理为有序的排列，建立了有序列表之后，诸如图书馆书目查询、搜索引擎关键字查找等许多实际问题就可以依赖高效查找算法来解决。排序和查找有多种实现方法和策略，往往需要根据不同的应用环境和数据特性来进行合理的取舍。

本单元首先通过“探寻斐波那契数列”项目，理解迭代和递归方法的特点，学习应用迭代和递归方法来解决问题的基本要点；通过“按序查看商品”和“网上商城查找商品”项目进一步学习常用的查找和排序算法，以及迭代和递归方法在解决此类问题时的一般思路，理解算法与数据结构的关系。

3.1　迭代与递归

荀子的《劝学篇》中说：“不积跬步，无以至千里；不积小流，无以成江海。”一次走一小步，不停地走，就能完成行走千里的大任务。支流在形式上与大河相似，只是规模小一些，但许多支流能汇成大江大河。对于重复或自相似的问题，在问题求解中通常会使用迭代和递归。

本节主要学习问题求解中迭代和递归两种常用的方法，体验应用迭代和递归解决问题的过程。

学习目标

- ★ 理解迭代和递归的概念。
- ★ 体验迭代和递归的方法。
- ★ 初步掌握迭代和递归方法的基本要点。

在日常生活中，经常要重复做一些动作，比如饭要一口一口吃、事情要一件一件办。有些结构是自相似的，比如计算机中的文件夹，一层套着一层，外面的文件夹和里面的文件夹是相似的。

本节围绕“探寻斐波那契数列”项目展开学习，通过项目活动学习迭代和递归方法的特点及一般模式。本节主要包含“用迭代法解决斐波那契问题”和“用递归法解决斐波那契问题”两个任务。

任务一　用迭代法解决斐波那契问题

※ 活动1　手工计算斐波那契数列

斐波那契数列是意大利数学家斐波那契在他13世纪初的著作中最早提出的。

小明最近迷上了贝壳螺旋轮廓线，感觉它非常优美。这个螺旋曲线可以由一系列正方形内90° 的圆弧连接而成，如图3.1.1所示。假设最小的两个正方形的边长为1，则这一系列正方形的边长依次为1，

1，2，3，5，8，…。通过观察和思考，小明发现了数列的规律：第1项是1，第2项是1，从第3项开始每项等于前两项的和，这个数列就是斐波那契数列。小明了解到，自然界中的许多现象都与这个数列相关，甚至前后两项的比值也逐渐接近0.618的黄金比例，真是太神奇了！小明很想知道第8个斐波那契数是什么。

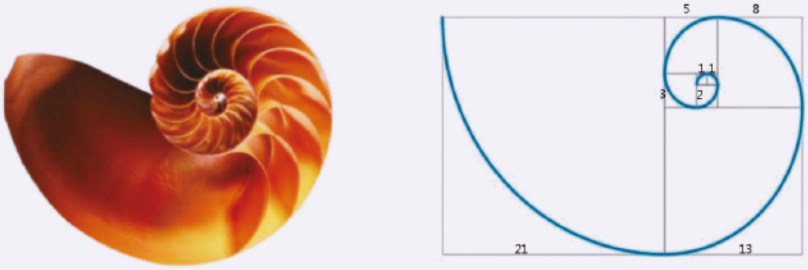


图 3.1.1　贝壳螺旋轮廓线与斐波那契数列

他还发明了计算斐波那契数的小工具——两张透明的卡片，用卡片依次盖住相邻的两项，并把盖住的两项求和可以得出下一项。

在表3.1.1中，深色单元格表示盖住的第1项，浅色单元格表示盖住的第2项，红色数字表示求出的项。

表 3.1.1　求斐波那契数第8项的过程

		第1项	第2项	第3项	第4项	第5项	第6项	第7项	第8项
	初始	1	1						
第1轮	求第3项	1	1	2					
第2轮	求第4项	1	1	2	3				
第3轮	求第5项	1	1	2	3	5			
第4轮	求第6项	1	1	2	3	5	8		
第5轮	求第7项	1	1	2	3	5	8	13	
第6轮	求第8项	1	1	2	3	5	8	13	21

用上面的方法来求斐波那契数列的第8项，共需要6轮。分析上面的操作，可以看出以下几点：

（1）最开始只有两个初始的数；

（2）每一轮都利用已经求出的最后两项计算出下一项斐波那契数；

（3）为了求出第8项，需要依次求出第3项、第4项，直到第7项。每操作一轮，就离目标更近；

（4）经过6轮求出第8项斐波那契数。

请用上面的方法计算出更多斐波那契数，并写在横线上。

1，1，2，3，___，___，___，___，___，___，___，___，___

斐波那契数列还有其他许多有趣的特点，在自然和社会生活中都可以看到它的身影，感兴趣的同学可以查阅相关资料。

● 迭代

迭代是从初始值出发，通过一系列步骤来逐步逼近问题最终解的过程。

例如，在上面求斐波那契数的活动中，初始值为1、1两项，从这两项出发，依次求出第3项、第4项……，每轮通过最后两项相加得到新的一项，每一轮都更接近第8项并最终达到第8项。

※ 活动2　利用迭代法解决斐波那契问题

分析上面的手工操作，使用了两张透明卡片。我们把前一张透明卡片称为变量a，把后一张透明卡片称为变量b，把待计算的斐波那契数称为变量c。

表3.1.2显示了求斐波那契数第8项过程中变量值的变化规律。

表 3.1.2　求斐波那契数第8项过程中变量值的变化规律

	第1项	第2项	第3项	第4项	第5项	第6项	第7项	第8项
初始	1 a	1 b						
第1轮	1 a	1 + b	2 → c					
第2轮	1	1 a	2 + b	3 → c				
第3轮	1	1	2 a	3 + b	5 → c			
第4轮	1	1	2	3 a	5 + b	8 → c		
第5轮	1	1	2	3	5 a	8 + b	13 → c	
第6轮	1	1	2	3	5	8 a	13 + b	21 → c

通过上面的算法分析可知，计算斐波那契数列的过程就是不断做以下操作的过程。

用代表后两项的变量a和b计算出新的项c：

a+b → c

然后让a和b指向新的最后两项，以便下一轮还可以使用a和b计算出新的c。使用以下操作更新变量a和b的值（想一想，下面两步的先后顺序可以交换吗？）：

b → a

c → b

整个程序就是控制以上过程从第3项依次计算到第8项。

通过以上分析，用fib函数求第n项斐波那契数，代码如下。

```
01. #求第n项斐波那契数
```

```
02. def fib(n):
03.   #n小于3时
04.   if n<3:
05.     return 1
06.   #n大于等于3时
07.   a=b=1
08.   for i in range(3,n+1):
09.     c=a+b
10.     a=b
11.     b=c
12.   return c
13. print(fib(8))
```

● 利用迭代法解决问题的要点

（1）确定迭代的起点。

在计算斐波那契数的程序中，由变量a和b计算出下一项，开始时a和b需要有一个初始值。通过代码a=b=1将a和b的初始值赋值为1。

（2）确定逼近最终解的操作。

这些操作的结果能更接近最终目标。例如，在计算斐波那契数的程序中，利用前两项的值计算出后一项的值，然后更新变量的值，让a和b指向已经计算出的数列的最后两项，从而可以用同样的规则继续计算，并且更接近最终目标。

（3）控制迭代过程。

迭代的过程控制一般由循环来完成。要设置好迭代的方向，从而逼近并达到最终目标。要设置好结束条件，以便目标达成时可以停止迭代。例如，在求斐波那契数的程序中，循环的结束条件是计算出第n项斐波那契数。

※ 活动3　用迭代法解决问题

我国古代的《九章算术》中记载了“更相减损术”，即求两个整数最大公约数的方法：

（1）如果两个数相等，则找到了最大公约数；

（2）否则，从大的数中减去小的数；

（3）重复操作（2），直到条件（1）满足，得到最大公约数。

请补全下面的代码。

```
#求两个整数a，b的最大公约数
def gxjs(a,b):
  while _______:                    #a和b不等
    if a>b:                         #大数减小数
      a=a-b
    else:
      _______
  return  a                         #a与b相等时，找到最大公约数
```

任务二　用递归法解决斐波那契问题

※ 活动1　再探斐波那契数列

小梅研究了小明的算法及程序，感到非常有趣。她提出了这样的猜想：既然一个大的斐波那契数由两个更小的数构成，那么计算斐波那契数的任务也可以通过任务的分解来完成。

比如求fib(6)可以分解为求fib(5)和fib(4)，它们又可以继续分解，最终都可以分解为求fib(1)或fib(2)。

如图3.1.2所示，小梅用这种方法来求第6个斐波那契数，箭头表示任务分解的方向。当分解到fib(1)或fib(2)时，得到答案1，然后再依次向回计算大一些任务的答案。比如fib(3)=fib(1)+fib(2)，得到fib(3)=2。fib(4)=fib(2)+fib(3)，得到fib(4)=3。请在图中横线上填写各个任务的答案。

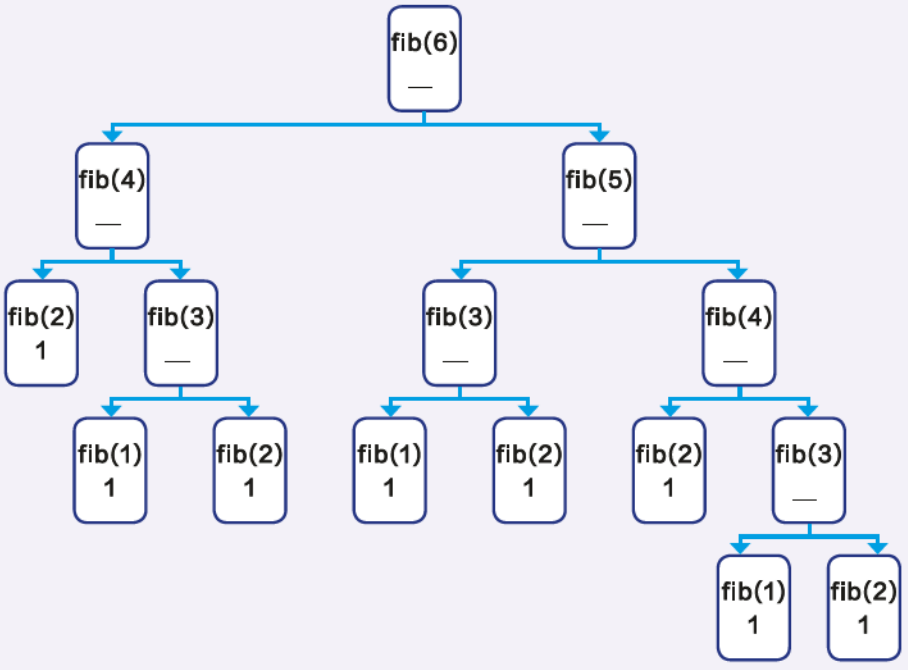


图 3.1.2　求第6个斐波那契数任务的分解

可以看到，小梅提出的这种方法确实能计算出第n项斐波那契数。小梅把求第n项斐波那契数的任务表示如下：

$$fib(n)=\begin{cases} fib(n-1)+fib(n-2) & n>2 \\ 1 & n=1或n=2 \end{cases}$$

● 递归

函数直接或间接调用自身称为递归。递归把一个大问题分解为规模更小的同类问题来求解。因为是同一类问题，所以还是调用原来的函数，只是函数的调用参数有一些变化。通过递归，并在问题足够小的时候直接给出简单解，只需少量的代码就可以解决复杂的问题。递归是一个常用的编程技巧，在许多算法的实现中都会用到。

小梅提出的求斐波那契数的方法就是递归。当n>2时，把任务分解为更小的任务fib(n−1)和fib(n−2)，其结果是两个小任务的和fib(n−1)+fib(n−2)。当问题是n=1或n=2的小问题时，直接得到答案。

在编写递归程序时，直接着眼于最终问题和任务的分解关系，至于执行过程中的细节，计算机会自动完成，不用过多考虑。

※ 活动2　用递归法解决斐波那契问题

在小梅的算法中，求解第n个斐波那契数的任务可以分解为两个更小的任务。在函数执行时，通过函数的递归调用实现任务的分解与结果的合成。请补全下面的代码。

```
01. def fib(n):
02.     if n==1 or _______:                    #n=1或n=2时
03.         return 1
04.     else:
05.         return fib(n-1)+fib(_________)     #否则，分解任务
06. print(fib(6))
```

● 使用递归法解决问题的要点

（1）确定问题最小规模的简单解。

例如，斐波那契数列问题中求第1项和第2项的值是问题的最小规模，可以直接得到结果1。

（2）把问题表示为更小规模的相同问题的组合。

例如，求斐波那契数列第n项可以表示为第n−1项和第n−2项的和，求斐波那契数列第n−1项和第n−2项都是任务相同但规模更小的问题。

（3）通过调用自身来解决问题。

例如，在上面的程序中，求第6项斐波那契数，就是直接调用fib(6)函数来解决问题。

※ 活动3　用递归法解决问题

想知道一个正整数n由几个数字组成吗？

这个问题有一个最简单的情况，当正整数n小于10时，答案为1。

否则，答案为整数n去掉个位数后的位数加1。

如果用lenOfNum(n)表示求正整数n的位数，以上的递归法可以表示为：

$$lenOfNum(n)=\begin{cases} lenOfNum(n//10)+1 & n\geqslant 10 \\ 1 & n<10 \end{cases}$$

补全下面的代码。

```
01. #求正整数n的位数
02. def lenOfNum(n):
03.   if n<10:              #小于10时
04.     _____________
05.   else:                 #大于等于10时
06.     return_____________
```

由于递归会引起一系列的函数调用，并且可能会有一系列的重复计算，所以递归算法的执行效率相对迭代来说更低。但是这种方法非常直观简洁，容易理解。

拓展练习

1. 分解质因数是把一个合数写成几个质数相乘的形式，比如18=2*3*3。一个数的质因数并不容易直接看出，手工分解质因数又比较麻烦，请用迭代和递归方法，分别编写程序，分解质因数。

2. 一个字符串如果正向看和反过来看是一样的，则这样的字符串叫回文字符串。比如，字符串“level”就是一个回文字符串，因为反过来也一样；而“bed”就不是回文字符串，因为反过来是“deb”，与原文不一样。注意：空格和数字也是字符，所以“ ^ ^”，“121”都是回文字符串。显然，任意单个字符都是回文字符。请编写程序判断一个字符串是不是回文字符串。

3.2 数据的排序

排序在日常生活中很常见，如军训时同学们按身高从低到高列队、整理数码照片时按照拍摄时间的先后次序编号等。其实，利用计算机排序也是极为重要、经常要做的一项工作，在许多算法和应用系统中都离不开排序。

本节主要学习排序的基本概念、两种常见的排序算法，以及迭代方法在排序中的应用。

学习目标

- ★ 理解排序的基本概念。
- ★ 掌握常见排序算法的思路和实现方法。
- ★ 体验迭代方法在排序中的应用。
- ★ 理解算法与数据结构的关系。

在网上商城购物的时候，用户通过单击页面中的“价格”按钮，可以快速将商品按照价格进行降序或者升序排列，如图3.2.1所示，以便于选择合适的商品。网上商城平台是如何实现按价格、销量等进行升序或降序排序，并显示商品信息的呢？

图 3.2.1 将商品按价格排序显示

本节围绕“按序查看商品”项目展开学习，通过项目活动学习常见的排序算法，体会迭代方法的具体应用，理解数据结构与算法的关系。本节主要包含“按价格升序显示商品”和“按销量降序显示商品”两个任务。

任务一　按价格升序显示商品

※ 活动1　初步尝试排序

表3.2.1所示是某网上商城的签字笔销售数据，请将表中数据按价格从低到高进行排序，将排序后的结果填入表3.2.2中。

表 3.2.1　签字笔销售数据

品牌	销量/盒	价格/元	评论数/条
博士	80	66	108
英雄	188	78	86
永辉	236	58	186
晨辉	200	46	190
得利	56	68	50
梅花	185	26	92

表 3.2.2　按价格从低到高排序后的签字笔销售数据

品牌	销量/盒	价格/元	评论数/条

将你的排序过程描述出来。

对签字笔销售数据按照价格从低到高进行排序的过程就是整理数据顺序的过程。通过比较价格的大小，调整数据的顺序。在整理的过程中，只调整数据的排列顺序，不改变数据内容。

排序

排序（sorting）就是整理数据的顺序，使其从无序变为有序。

※ 活动2　体验冒泡排序过程

如何将表3.2.1中的所有数据按照价格从低到高进行排序呢？小明提出了一个解决思路：在未排序的签字笔销售数据中找到价格最高者放在序列的末尾，这样未排序的签字笔销售数据就会减少1个。若干次迭代后，当未排序的签字笔销售数据只剩下一个时，排序就完成了。

根据这个思路，他尝试用“比较—交换”的方法将表3.2.1中的签字笔销售数据按照价格从低到高进行排序。

第1次迭代：如图3.2.2所示，比较66与78，66<78，不交换；比较78与58，78>58，交换；比较78与46，78>46，交换；比较78与68，78>68，交换；比较78与26，78>26，交换。

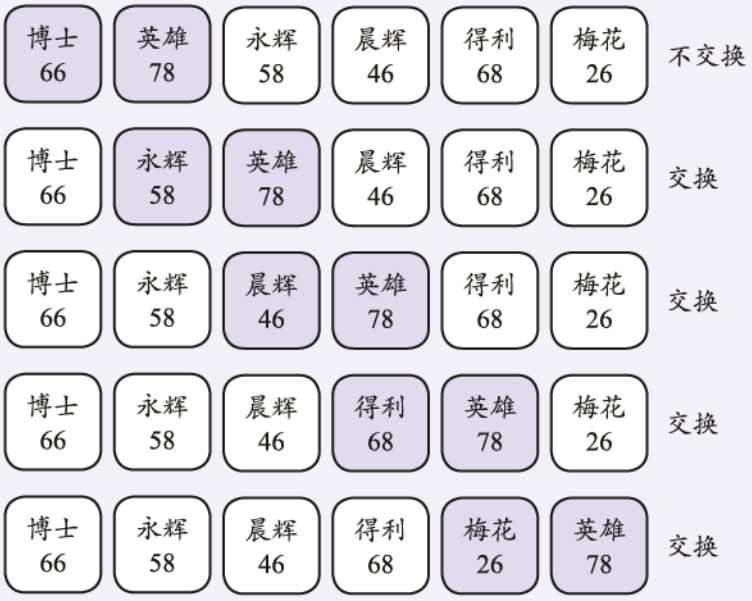


图3.2.2　第1次迭代过程

每一次迭代过程完成一个数据的排序。

经过5次“比较—交换”，价格最高的英雄牌签字笔排到了末尾，这样未排序的数据减少了1个。第1次迭代后价格的序列是：66，58，46，68，26，78。

根据这个思路，补全图3.2.3，完成第2次迭代过程。

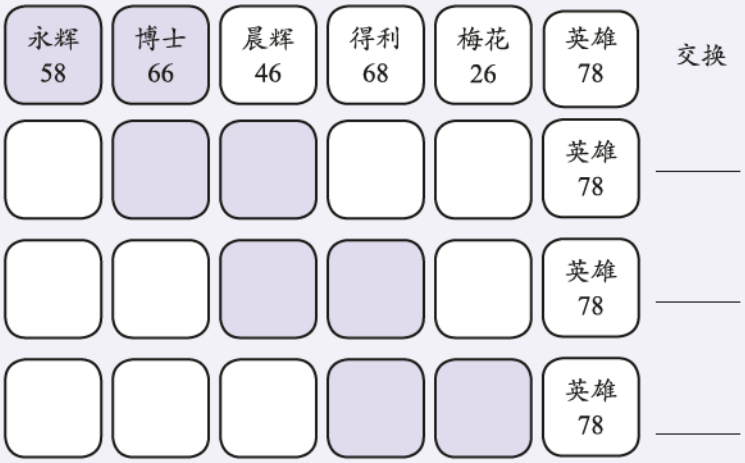


图 3.2.3　第2次迭代过程

第 2 次迭代后价格的序列是:_____,未排序的数据剩下 ___ 个。
第 3 次迭代后价格的序列是:_____,未排序的数据剩下 ___ 个。
第 4 次迭代后价格的序列是:_____,未排序的数据剩下 ___ 个。
第 5 次迭代后价格的序列是:_____,未排序的数据剩下 ___ 个。
经过5次迭代，最后完成了6个签字笔销售数据按照价格升序排列。

● 冒泡排序

冒泡排序（bubble sort）是一种通过“比较—交换”进行排序的方法，首先将第1个数据和第2个数据进行比较，若为逆序，则将两个数据交换位置；然后比较第2个数据和第3个数据，以此类推，直至最后两个数据进行过“比较—交换”为止。上述过程称作第一趟“冒泡”，其结果使得最大的数据被放到最后的位置上。之后对除了最大数据之外的剩余部分进行下一趟“冒泡”，经过若干趟“冒泡”后，如果剩余部分只包含一个数据，则冒泡排序完成。

※ 活动3　建立数据结构

为了便于对签字笔销售数据进行处理，定义签字笔商品类，其Python代码如下。

```
01. #定义签字笔商品类
02. class  pen:
03.   def __init__(self, brand, sales, price, comments)
04.     self.brand=brand                    #品牌
05.     self.sales=sales                    #销量
06.     self.price=price                    #价格
07.     self.comments=comments              #评论数
```

创建一个线性表对象alist，存放表3.2.1所示的签字笔销售数据对象。请补全下面的代码。

```
08. from linearList import LinearList          #导入线性表
09. alist=LinearList()                         #创建线性表对象
10. alist.appendItem(pen("博士",80,66,108))    #添加签字笔数据元素
11. alist.appendItem(pen("英雄",188,78,86))    #添加签字笔数据元素
12. alist.appendItem(pen("永辉",236,58,186))   #添加签字笔数据元素
13. alist.appendItem(pen(_____))               #添加签字笔数据元素
14. alist.appendItem(pen(_____))               #添加签字笔数据元素
15. alist.appendItem(pen(_____))               #添加签字笔数据元素
```

※ 活动4　算法设计与实现

分析活动2中小明的排序过程和冒泡排序的基本思想，假设有n个签字笔销售数据，冒泡排序的算法描述如下。

（1）进行n−1趟“冒泡”过程，每趟操作如步骤（2）和（3），如果未完成则继续进行操作。第1趟的冒泡范围是n个数据。

（2）从第1个数据开始，依次比较相邻两个数据，如果逆序则交换位置，直到比较完最后两个数。冒泡范围内的最大数据会被交换到最后的位置上。

（3）将本趟的最大数据排除出“冒泡”操作范围。

根据上述算法，定义冒泡排序函数bubbleSort(alist,key)，参数alist表示需要排序的线性表，key表示排序的关键词。请补全下面的代码。

```
16. #冒泡排序算法
17. def bubbleSort(alist, key):
18.   for i in range(alist.size()-1):          #控制“冒泡”趟数
19.     #控制“冒泡”范围
20.     for j in range(1, alist.size()-i):
21.       #相邻两数比较
22.       if getattr(_____,key) > getattr(alist.getItem(j),key):
23.         temp=alist.getItem(j-1)
24.         alist.setItem(j-1, alist.getItem(j))
25.         alist.setItem(j, _____)          #交换位置
```

以表3.2.1中的数据为例，利用冒泡排序法实现签字笔销售数据按价格升序排序，并显示排序后的结果。请补全下面的代码。

```
26. bubbleSort(alist, ____)              #调用冒泡排序法函数
27. print("按价格升序排序后的结果是：")
28. for i in range(alist.size()):
29.    #显示排序后的结果
30.    print(getattr(alist.getItem(i),'brand'), ____,____,____)
```

在冒泡排序过程中，充分利用迭代的方法，通过对数据进行“比较—交换”，实现数据的有序排列。

任务二　按销量降序显示商品

※ 活动1　体验直接插入排序过程

如何将表3.2.1中的所有数据按照销量从高到低进行排序呢？小梅提出了一个解决思路：第1个签字笔销售数据已经有序，把待排序的签字笔销售数据按照销量插入到有序序列的合适位置，有序数据就增加1个。这样迭代若干次后，直到所有的待排序签字笔销售数据插入完为止，排序就完成了。

根据这个迭代思路，她尝试用“比较—插入”的方法将表3.2.1中的签字笔销售数据按照销量进行排序。

第1次迭代：销量是80的签字笔销售数据是有序子序列，把销量为188的签字笔销售数据插入。比较80和188的大小，80<188，将销量为80的签字笔销售数据后移一个位置，将销量为188的签字笔销售数据插入到它的前面。第1次迭代后销量的序列是：188，80，236，200，56，185。

第2次迭代：插入销量为236的签字笔销售数据，从有序子序列（188, 80）的最后1个数据开始，将销量为236的签字笔销售数据与序列中的数据逐一比较大小，确定插入位置。比较80和236，80<236，将销量为80的签字笔销售数据后移一个位置；比较188和236，188<236，将销量为188的签字笔销售数据后移一个位置，将销量为236的签字笔销售数据插入到它的前面。经过两次比较后确定位置并插入，第2次迭代后销量的序列是：236，188，80，200，56，185。

请在下面的横线上依次写出其他各次迭代的结果。

第3次迭代：插入____，迭代后销量的序列是：_____；

第4次迭代：插入____，迭代后销量的序列是：_____；

第5次迭代：插入____，迭代后销量的序列是：_____。

经过5次迭代，最后完成了6个签字笔销售数据按照销量降序排列。

直接插入排序

直接插入排序（straight insertion sort）是一种通过“比较—插入”进行排序的方法。基本操作是从有序子序列的最后一个数据开始，待排序的数据与有序子序列中的所有数据逐一进行比较，确定插入的位置并插入数据。上述过程称作一次插入操作，其结果是将一个数据插入到已经排好序的数据序列中，从而得到一个新的、数据数量增加一个的有序子序列。对未排序的数据进行相同的操作，重复若干次插入操作，直到所有的未排序数据插入完为止，则插入排序完成。

直接插入排序是一种简单的排序方法。默认第一个数据是有序子序列。

※ 活动2　直接插入排序算法的设计与实现

建立数据结构的方法与任务一中的活动3相同，创建线性表对象alist并添加数据元素。

分析活动1中小梅的排序过程和直接插入排序的基本思想，假设有n个签字笔销售数据，实现直接插入排序的算法描述如下。

（1）进行n-1次插入操作，每次操作如步骤（2）和（3），如果未完成则继续进行。默认第1个数据是有序子序列。

（2）取得当前待排序数据，从有序数据序列的最后一个数据开始，待排序数据与其逐一进行比较，如果大于对方则后移该数据，确定插入位置。

（3）将待排序数据插入到有序数据序列中。

根据上述算法，定义insertionSort(alist,key)函数，参数alist表示需要排序的线性表，key表示排序的关键词。请补全下面的代码。

```
16. #直接插入排序算法
17. def insertionSort(alist, key):
18.   #默认第1个数据已有序，控制插入次数
19.   for i in range(1,alist.size()):
20.     currentdata =alist.getItem(i)    #取当前待排序数据
21.     j=i
22.     #与待排序数据进行比较
23.     while j>0 and getattr(alist.getItem(j-1),key)<
24.     getattr(_____):
25.       alist.setItem(j,alist.getItem(j-1))      #后移数据
26.       j=j-1                                    #确定插入位置
27.     alist.setItem(j, _________)                #插入数据
```

以表3.2.1中的数据为例，利用直接插入排序法实现签字笔销售数据按销量降序排序，并显示排序后的结果。请补全下面的代码。

```
28. insertionSort(alist, ____)          #调用直接插入排序法函数
29. print("按销量降序排序后的结果是：")
30. for i in range(alist.size()):
31.    #显示排序后的结果
32.    print(getattr(alist.getItem(i),'brand'), ____,____,____)
```

在直接插入排序过程中，同样利用了迭代的方法，通过对数据进行比较和插入的操作，实现数据的有序排列。

对数据进行排序不只有冒泡排序法和直接插入排序法这两种排序算法，每种排序算法有不同的适用范围和针对的应用情境。在冒泡排序的过程中，数据的处理是基于数据的交换，直接插入法排序的过程则是数据的后移。相比之下，直接插入排序法的效率高于冒泡排序法。

拓展练习

1. 给定待排序列表 [29, 1, 9, 7, 3, 10, 13, 25, 8, 12]，下面哪个列表是冒泡排序第三趟冒泡之后的列表？

A. [1, 9, 29, 7, 3, 10, 13, 25, 8, 12]

B. [1, 3, 7, 9, 10, 8, 12, 13, 25, 29]

C. [1, 7, 3, 9, 10, 13, 8, 12, 25, 29]

D. [1, 9, 29, 7, 3, 10, 13, 25, 8, 12]

2. 调查本班同学的生日和身高，制作成统计表。基于本班同学的信息统计表，编写程序解决以下问题。

（1）利用冒泡排序法实现同学信息按身高降序排列显示。

（2）利用直接插入排序法实现同学信息按生日升序排列显示。

3.3 数据的查找

日常生活中，人们几乎每天都要进行“查找”工作。例如，在手机通讯录里查找某人的电话号码，在微信通讯录里查找某位好友的微信号，在图书馆里查找喜欢的图书等。在各种系统软件和应用软件中，查找也是最常用的功能之一。

本节主要学习查找的基本概念、顺序查找和二分查找算法，以及递归方法在查找中的应用。

学习目标

- ★ 理解查找的基本概念。
- ★ 掌握查找算法的思路和实现过程。
- ★ 体验递归方法在查找中的应用。
- ★ 理解算法与数据结构的关系。

用户在网上商城购物的时候，可以在商品搜索输入框中通过输入商品类别名称或者商品品牌名称来快速查找需要的商品信息。在网上商城众多的商品数据中，如何快速地查找到所需商品的相关信息呢？

本节围绕“网上商城查找商品”项目展开学习，通过项目活动体验计算机查找数据的过程，理解算法与数据结构的关系。本节主要包含“根据品牌查找商品”和“根据价格查找商品”两个任务。

任务一　根据品牌查找商品

※ 活动1　体验顺序查找过程

表3.3.1所示是某网上商城的签字笔销售数据，如何在表中查询到品牌为“得利”的签字笔销售数据呢？

表 3.3.1　签字笔销售数据

品牌	销量/盒	价格/元	评论数/条
博士	80	66	108
英雄	188	78	86
永辉	236	58	186
晨辉	200	46	190
得利	56	68	50
梅花	185	26	92
凌梅	68	32	65
巧虎	221	72	198

小明在尝试查找品牌为“得利”的签字笔销售数据时，采取了一种简单直接的方法进行查找。他从表格的第1行数据开始，先比较第1行数据中的品牌与“得利”是否相等，如果相等则说明查找成功；如果不相等则继续与下一个数据进行比较，直到所有数据都比较完毕。

第 1 次比较：“博士”与“得利”不相等，继续与下一个数据比较。

第 2 次比较：“英雄”与“得利”不相等，继续与下一个数据比较。

请根据这个思路，继续完成后面几个数据的比对过程。

第3次比较：_____

第4次比较：_____

第5次比较：_____

从第1个数据开始，按顺序逐一进行比较，经过5次比较，找到品牌为“得利”的签字笔销售数据。

● 查找

查找（searching）就是在数据表中确定一个与给定值相等的数据，若存在这样的数据则表示查找成功，否则表示查找不成功。

● 顺序查找

顺序查找（sequential search）是一种从头开始逐个数据进行比较的查找方法。查找过程为：从数据表中第1个数据开始，逐个与要查找的数据进行比较，如果与要查找的数据相等，则查找成功；如果直至最后一个数据，与要查找的数据都不相等，则查找不成功。

※ 活动2 建立数据结构

创建一个线性表对象alist，存放表3.3.1所示的签字笔销售数据对象。请补全下面的代码。

```
01. from  linearList  import LinearList        #导入线性表
02. alist=__________                           #创建线性表对象
03. alist.appendItem(pen("博士",80,66,108))    #添加签字笔数据元素
04. alist.appendItem(pen("英雄",188,78,86))    #添加签字笔数据元素
05. alist.appendItem(pen("永辉",236,58,186))   #添加签字笔数据元素
06. alist.appendItem(pen(__________________))  #添加签字笔数据元素
07. alist.appendItem(pen(__________________))  #添加签字笔数据元素
08. alist.appendItem(pen(__________________))  #添加签字笔数据元素
09. alist.appendItem(pen(__________________))  #添加签字笔数据元素
10. alist.appendItem(pen(__________________))  #添加签字笔数据元素
```

※ 活动3 顺序查找算法的设计与实现

分析活动1中小明的查找过程和顺序查找的基本思想，假设有n个签字笔销售数据，实现顺序查找的算法描述如下。

（1）从线性表中的第1个数据开始，依次进行操作（2）。

（2）将当前数据与要查找数据进行比较，如果相等，则查找成功，返回数据在表中的位置，查找结束。如果不相等，则继续查找。

（3）查找失败。

根据上述算法，定义sequentialSearch(alist,key,item)函数，参数alist表示存储数据的线性表，key表示查找的关键词，参数item表示要查找的数据。请补全下面的代码。

```
11. #顺序查找算法
12. def sequentialSearch(alist, key, item):
13.   i=0                              #初始化循环变量
14.   while i<alist.size():
15.     #判断第i个位置上的数据与查找的数据是否相等
16.     if getattr(_____, key)==item:
17.       return _____                 #返回查找到的数据在表中的位置
18.     else:
19.       i=i+1                        #进行下一个数据比较
20.   return -1                        #返回查找失败
```

以表3.3.1中的数据为例，利用顺序查找法查找品牌为“得利”的签字笔，并显示查找到的签字笔销售数据。请补全下面的代码。

```
result=sequentialSearch(alist, ____, ____)    #调用顺序查找函数
print("顺序查找的结果是：")
if result==-1:
    print("查找失败")
else:
    print(getattr(alist.getItem(result),'brand'),____,
    ____,____)                                  #显示查找结果
```

假设有n个数据，利用顺序查找法，整个查找过程需要进行k（1≤k≤n）次数据比较，可以利用循环语句实现。

任务二　根据价格查找商品

※ 活动1　体验二分查找过程

商品价格是用户在选择商品时需要考虑的重要因素。在网上商城选择商品的时候，可以通过查询价格来选择理想的商品。比如，在表3.3.1中如何快速查找到价格是68的签字笔销售数据呢？

小明在解决该问题的时候，想到了这样一种思路：把签字笔销售数据表按照价格进行升序排列，找到中间项（位于表中间位置）数据，如果查找项与中间项相等，则查找结束。如果查找项比中间项大，可以把数据表中较小的那部分（包括中间项）排除了，因为如果查找项在表中，那它一定在较大的那一半中。接下来，可以在较大的一半中重复这个过程。如果查找项比中间项小，可以把数据表中较大的那部分（包括中间项）排除了，因为如果查找项在表中，那它一定在较小的那一半中。接下来，在较小的一半中重复这个过程即可。

将表3.3.1中的签字笔销售数据按照价格进行升序排列，变成一个有序表，如表3.3.2所示。

表 3.3.2　按价格升序排列后的签字笔销售数据

品牌	销量/盒	价格/元	评论数/条
梅花	185	26	92
凌梅	68	32	65
晨辉	200	46	190
永辉	236	58	186
博士	80	66	108
得利	56	68	50

续表

品牌	销量/盒	价格/元	评论数/条
巧虎	221	72	198
英雄	188	78	86

根据小明的思路，查找价格为68的签字笔销售数据的过程如下。

第1次查找：确定数据表的中间项是价格为58的签字笔，如图3.3.1所示。将查找项与中间项进行比较，68>58，排除较小的那部分和中间项。

图3.3.1 第1次查找

第2次查找：在数据表较大的部分中继续查找，确定数据表较大部分的中间项是68，如图3.3.2所示。将查找项与中间项进行比较，68=68，查找成功。

图3.3.2 第2次查找

请根据这个思路，写出查找价格是46的签字笔销售数据的过程。

二分查找

对于有序表的查找，从表的中间项开始比对，如果表的中间项匹配查找项，则查找结束。如果不匹配，就有两种情况：表的中间项比查找项大，那么查找项只可能出现在前半部分；表的中间项比查找项小，那么查找项只可能出现在后半部分。重复上述查找过程，每次都会将比对范围缩小一半，直到查找结束。这种查找的方法，称为二分查找（binary search）。

※ 活动2　二分查找算法的设计与实现

建立数据结构的方法与任务一中的活动2相同，创建线性表对象alist并添加数据元素。

分析活动1中小明的查找过程和二分查找的基本思想，假设有n种商品，实现二分查找的算法描述如下。

（1）确定查找范围，每次查找如步骤（2）和（3），如果没有完成则继续操作。第1次查找的范围是n个数据。

（2）取得查找范围的中间位置，将查找数据与中间位置的数据进行比对，如果二者相等则查找成功，返回数据在表中的位置。

（3）如果查找数据小于中间位置的数据则缩小查找范围至前半部分，如果大于中间位置的数据则缩小查找范围至后半部分。

根据上述算法，定义binarySearch(alist,key,item)函数，参数alist表示按关键字key排序后的线性表，参数key表示查找的关键字，参数item表示要查找的数据。请补全下面的代码。

```
#二分查找算法
def binarySearch(alist, key, item):
    first=0                              #标记查找范围起始位置
    last=alist.size()-1                  #标记查找范围终点位置
    while first<=last:
        mid=(________)//2                #取得查找范围的中间位置
        #判断中间项是否等于查找项
        if getattr(alist.getItem(mid),key)==item:
            #返回查找到的商品在线性表中的位置
            return  mid
        else:
            #判断查找项是否小于中间项
            if item<getattr(_________________):
                last=mid-1                   #缩小查找范围至前半部分
            else:
                first=_______                #缩小查找范围至后半部分
    return -1                            #返回查找失败
```

以表3.3.1中的数据为例，利用二分查找法查找价格为68的签字笔销售数据，并显示查找到的签字笔销售数据。请补全下面的代码。

```
bubbleSort(alist, 'price')                    #调用冒泡排序函数
result=binarySearch(alist, ____, ____)        #调用二分查找函数
```

```
print("二分查找法查找的结果是：")
if result==-1:
  print("查找失败")
else:
  print(getattr(alist.getItem(result),'brand'),____,
  ____,____)                                    #显示查找结果
```

假设线性表中有n个数据，根据二分查找的算法思路，第1次比对后查找范围缩小一半，第2次比对后再缩小一半，每次比对都将查找范围缩小一半，直到找到所需要的数据或者查找范围缩小到0。整个查找过程是一个确定次数的重复操作，因此可以利用循环语句实现。

※ 活动3　利用递归法实现二分查找

利用递归法实现二分查找，定义recursionSearch(alist,key,item,first,last)函数，参数alist表示按关键字key排序后的线性表对象，参数key表示查找的关键字，参数item表示要查找的数据，参数first表示查找范围的起始位置，参数last表示查找范围的结束位置。利用Python编写的代码如下，请补全下面的代码。

```
#利用递归方法实现二分查找算法
def recursionSearch(alist,key,item,first,last):
  if first<=last:
    mid=(first+last)//2                  #取得中间项位置
  else:
    return -1                            #返回查找失败
  #中间项与查找项比较
  if getattr(alist.getItem(mid), key)==item:
    return mid                  #返回查找到的商品在线性表中的位置
  else:
    #判断查找项是否小于中间项
    if item<getattr(__________, key):
      #递归调用
      return recursionSearch(alist,key,item,first, ___)
    else:
      #递归调用
      return recursionSearch(alist,key,item, ___,last)
```

以表3.3.1中的数据为例，利用递归二分查找法查找商品价格为32的商品，并显示该商品的相关信息。请补全下面的代码。

```
bubbleSort(alist, 'price')                    #调用冒泡排序函数
#调用递归二分查找函数
result=recursionSearch(alist, ____, ____, ____, ____)
print("递归二分查找法查找的结果是：")
if result==-1:
    print("查找失败")
else:
    #显示查找结果
    print(getattr(____,'brand'), ____,____,____)
```

当对一个有序表执行二分查找时，首先确定中间项。如果查找项比中间项小，可以在原来表的前半部分执行二分查找；如果查找项比中间项大，可以在后半部分执行二分查找。分析二分查找算法的过程可以发现，它是把一个问题分解成更小规模的问题，先用一些方法解决这些更小规模的问题，然后重组整个问题来得到结果。

拓展练习

1. 假如有以下已排好序的列表[3，5，6，8，11，12，14，15，17，18]，用二分查找算法进行查找。为了找到8，下列哪组数反映了正确的比对顺序？

A. 11，5，6，8　　B. 12，6，11，8　　C. 3，5，6，8　　D. 18，12，6，8

2. 请到学校图书馆调查信息技术类的图书名称和库存量数据，制作成统计表。基于图书数据统计表，编写程序解决以下问题。

（1）通过键盘输入图书名称，利用顺序查找法查找图书信息并显示查找结果。

（2）通过键盘输入库存量，利用二分查找法查找图书信息并显示查找结果。

单元学习评价

本单元学习了迭代和递归的特征和基本思路，体验了常见的排序和查找算法。你理解迭代与递归的相关概念了吗？你能运用排序与查找算法解决实际问题吗？请参与小组交流并反思，开展自评或小组评价。

一、有一种解决问题的方法，通过函数调用自身，把大问题分解为更小规模的、相同的问题的组合，并对最小规模的问题给出简单解，这种解决问题的方法称为（　　）。

A. 迭代　　B. 分解　　C. 递归　　D. 循环

二、高一年级各班献爱心公益活动的数据统计如下：高一（1）班，捐书100册，捐款508元；高一（2）班，捐书124册，捐款618元；高一（3）班，捐书98册，捐款588元；高一（4）班，捐书132册，捐款428元；高一（5）班，捐书118册，捐款822元。请编写程序，使其实现可以按照任意关键字进行升序或降序显示各班的数据。

1. 问题分析。

（1）为便于对数据进行排序处理，可以将统计表数据定义为类，请写出类的定义。

_____

（2）实现按照任意关键字进行排序显示，要解决的关键问题是什么？

_____

2. 设计数据结构。

（1）采用什么数据结构保存数据统计表中的数据？

_____

（2）为什么采用这种数据结构？

_____

3. 设计算法。

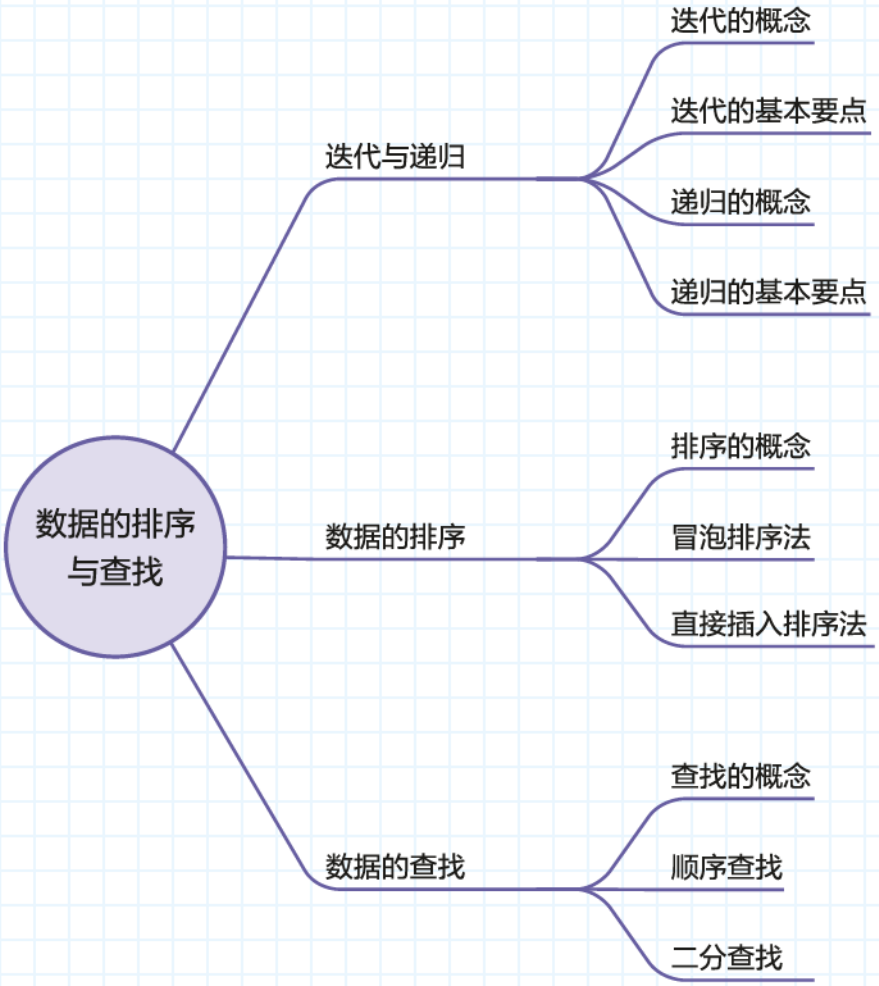
（1）怎样输入、输出数据？

_____

（2）画出算法流程图。

4. 编程实现。

单元学习总结


数据的排序与查找
迭代与递归
迭代的概念
迭代的基本要点
递归的概念
递归的基本要点
数据的排序
排序的概念
冒泡排序法
直接插入排序法
数据的查找
查找的概念
顺序查找
二分查找

第 4 单元　队列及其应用

队列是一种操作受限的线性结构，数据元素只能在一端进，在另一端出，且具有“先进先出”的特征。队列出现在日常生活各类“排队”活动中，如影院排队入场、商店排队付款和餐厅排队就餐时都会形成队列。另外，计算机科学中的很多功能，如打印队列、进程调度和键盘缓冲等也都是利用队列结构来实现的。

本单元通过“排队买票”项目，理解队列结构的基本概念及特征，学习队列的两种实现方法，通过“基数排序”和“餐馆排队取号模拟系统”项目学习并亲历利用队列解决问题的一般方法和过程，体验迭代思想在问题解决中的应用。

4.1　队列结构及其实现

日常生活中，我们经常会遇到排队的情况，如在食堂排队买饭、在电影院排队买票、在超市排队结账等。排队的过程会形成一个队列，排在队列最前面的优先处理，后来的必须排在队尾。在计算机解决问题的过程中也经常会用到队列。

本节主要学习队列的概念及其特征、队列抽象数据类型的定义、队列的实现方法。

学习目标

★ 理解队列的概念及其特征。

★ 掌握队列抽象数据类型的定义。

★ 掌握队列的两种实现方法。

“读万卷书，行万里路。”利用假期时间去游览祖国各地的名胜古迹，既可以放松紧张的心情，又可以学到很多知识。在旅游风景区，排队是不可避免的现象。

本节围绕“排队买票”项目展开学习，通过项目活动认识生活中的队列，学习利用抽象数据类型定义队列，编写代码实现队列的基本操作。本节主要包含“体验排队买票”和“编程实现排队买票”两个任务。

任务一　体验排队买票

※ 活动 1　认识生活中的队列

排队是我们生活中司空见惯的一种现象，图 4.1.1 所示是在旅游风景区大家排队购买门票的场景。前来购票的人自觉排成一个队列，排在队首的人优先购买门票，后来的人只能排在队尾，大家按

顺序依次完成购票。

图4.1.1 排队买票

如果我们将排队买票的人抽象成一个数据元素 a，那么排队买票的队列可以用图4.1.2表示。

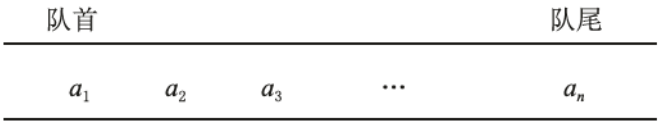


图4.1.2 买票队列

根据图4.1.2所示，分析排队买票的情况，完成以下问题。

（1）购票的行为发生在队列的_____；完成购票的人离开队列后，队列有什么变化？

（2）新来的买票人排在队列的_____；购票的人增加后，队列有什么变化？

队列

队列（queue）是一种操作受限制的线性结构，只允许在表的前端（队首）进行数据元素删除操作，在表的后端（队尾）进行插入操作。

在队列中插入一个数据元素称为入队，从队列中删除一个数据元素称为出队。因为队列只允许在队尾插入、在队首删除，所以先进入队列的元素先从队列中删除，故队列又称为先进先出（First In First Out，FIFO）线性表。

※ 活动2 观察排队买票

暑假期间，小明担任了天文馆售票处的志愿者工作，工作内容是维持游客的买票秩序，在开始售票前后的一段时间内（7：50~8：00），他观察到排队买票的队列发生了以下变化。

（1）7：50，售票窗口前没有人排队。

（2）7：55，售票窗口前有5个人（用p1，p2，p3，p4，p5表示）依次在排队。

（3）8：00，开始售票，有3个人（p1，p2，p3）陆续买票离开，又来了4个人（p6，p7，p8，p9）依次排入队中。

根据上述观察，思考并回答下面的问题。

（1）最先进入队列的是_____。

（2）p3买完票离开后，排在队首的人是_____，队列里有_____个人在排队。

● 队列抽象数据类型

为了在程序中使用队列结构来解决问题，我们需要定义队列抽象数据类型（ADT Queue）。根据问题解决需要，我们为队列抽象数据类型定义了如下接口。

ADT Queue：

- Queue()：创建一个空队列，返回值是一个空的队列。
- enQueue(item)：将数据元素 item 添加到队尾，无返回值。
- deQueue()：从队首删除数据元素，返回队首数据元素。
- size()：返回队列中数据元素的个数。
- isEmpty()：判断是否为空队列，返回布尔值。

※ 活动3　实现排队买票

有了队列抽象数据类型，我们就可以用程序来实现买票人的入队、出队以及统计排队人数的操作了。假设有5个人（用p1，p2，p3，p4，p5表示）依次前来排队买票，完成排队买票的过程如下。请根据操作描述补全下面的代码。

```
01. ticketLine=Queue( )                #创建空队列
02. ticketLine.enQueue(p1)             #p1入队
03. ticketLine.enQueue(p2)             #p2入队
04. _____                              #p3入队
05. _____                              #p4入队
06. _____                              #p5入队
07. ticketLine.deQueue()               #p1出队
08. ticketLine.deQueue()               #p2出队
09. num=ticketLine.size()              #统计当前排队人数
10. _____                              #p3出队
```

```
11. _____                    #p4出队
12. _____                    #p5出队
13. _____                    #查看队列是否为空
```

任务二 编程实现排队买票

※ 活动1 用顺序存储实现队列

队列抽象数据类型主要包括创建队列、入队、出队、统计队列数据元素个数、判断队列是否为空等操作接口。队列的顺序存储实现，可以用Python列表数据类型来实现。

（1）创建队列。

创建空队列，就是建立一个空的列表，用属性items来保存队列中的数据元素。请补全下面的代码。

```
01. class Queue:
02.   def __init__(self):                    #创建队列
03.     self.items=_______                   #空列表
```

（2）入队。

入队操作就是在列表items的最后添加一个新的数据元素item。例如，p6入队的过程如图4.1.3所示。

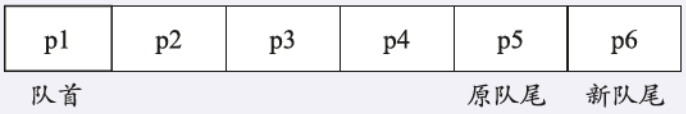


图4.1.3 p6入队

根据图4.1.3所示，补全enQueue方法的实现代码。

```
04.   def enQueue(self,item):                #入队
05.     self.items.append(_________)         #队尾增加一个数据元素
```

（3）出队。

出队操作就是移除并返回列表items中的第一个数据元素。例如，p1出队的过程如图4.1.4所示。

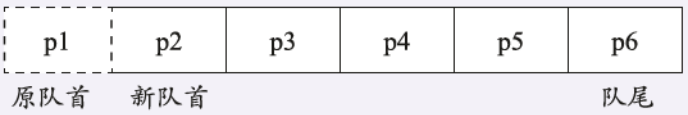


图4.1.4 p1出队

根据图4.1.4所示，补全deQueue方法的实现代码。

```
06.    def deQueue(self):                    #出队
07.      return  self.items.pop(_______)  #移除队首元素并返回
```

（4）统计队列数据元素个数。

```
08.    def size(self):
09.      return len(self.items)              #返回队列的数据元素个数
```

（5）判断队列是否为空。

```
10.    def isEmpty(self):
11.      return self.items==[]               #返回队列是否为空的判断值
```

● 列表实现的不同方案

队列是一种要求先进先出的数据结构，需要在表的一端插入数据元素，在另一端删除数据元素。用列表实现队列存在两种情况：一种情况是队首放在列表头；另一种情况是队首放在列表尾。两种不同的情况，在实现入队和出队的操作中也存在差别，如表 4.1.1 所示。

表 4.1.1　两种用列表实现队列方法的区别

情况	入队	出队
队首在列表头	利用 append()方法实现	利用 pop(0)方法实现
队首在列表尾	利用 insert(0,item)方法实现	利用 pop()方法实现

将队列抽象数据类型的顺序存储实现代码输入到文件中，保存为 queue.py，利用排队买票测试各个接口的效果。例如，小明和小梅来排队买票，小明买到票出队，统计当前队列中的排队总人数。请补全下面的代码。

```
01. from queue import Queue          #导入队列
02. ticketLine=_____________         #创建队列对象
03. ticketLine.enQueue("小明")       #小明入队
04. ___________________________      #小梅入队
05. ___________________________      #小明出队
06. print(ticketLine.size())         #显示当前排队总人数
```

※ 活动 2　用链式存储实现队列

除了利用顺序存储实现队列以外，还可以使用另一种方法——基

于节点引用的链表方式。

（1）创建空队列。

创建空队列，也可以称为队列的初始化，将队首节点引用和队尾节点引用都指向空，如图4.1.5所示。

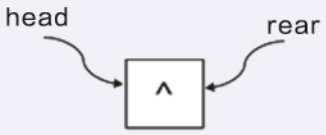


图4.1.5 队列初始化

head：队首节点引用。
rear：队尾节点引用。

根据图4.1.5的提示，补全队列初始化代码。

```
01. class Queue():
02.   def __init__(self):
03.     self.head=None              #队首指向空节点
04.     self.rear=_____             #队尾指向空节点
```

（2）入队。

入队操作就是在链表的队尾插入一个节点。例如，p5入队的过程如图4.1.6和图4.1.7所示。

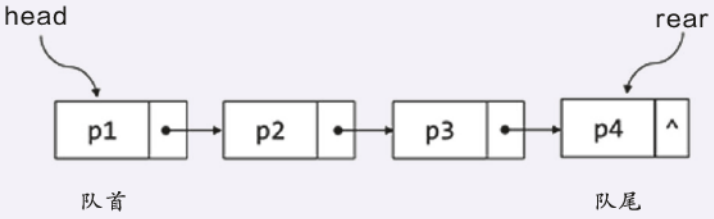


图4.1.6 插入节点前

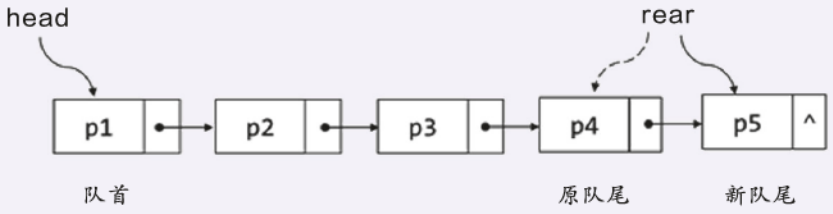


图4.1.7 插入节点后

根据图4.1.6和图4.1.7的提示，补全enQueue方法的实现代码。

```
05.   def enQueue(self, item):     #将元素item加入队列，入队
06.     p=Node(_____)             #生成新节点
07.     if self.head==None:       #判断队首节点是否为空
08.       self.head=p             #队首指向新节点
09.       self.rear=_____         #队尾指向新节点
```

```
10.      else:
11.        tp=self.rear                #获取当前队尾节点
12.        tp.next=p                   #当前队尾指向新节点
13.        self.rear=_____             #队尾指向新节点
```

（3）出队。

出队操作就是在链表的首端删除一个节点，修改头节点引用。例如，p1 出队的过程如图 4.1.8 所示。

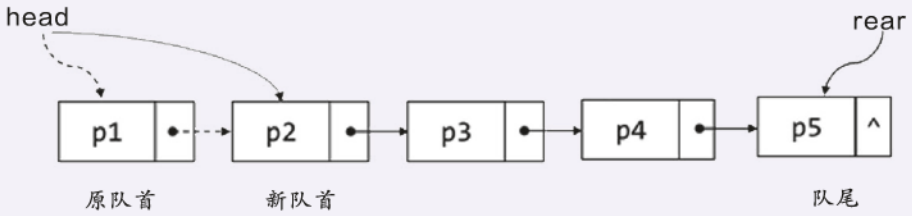


图 4.1.8　删除节点

根据图 4.1.8 的提示，补全 deQueue 方法的实现代码。

```
14.    def deQueue(self):              #删除队首元素，出队
15.      if self.head==None:           #判断队列是否为空
16.        return  None                #返回空
17.      result=_________              #获取队首节点数据元素
18.      self.head=_________           #重新设置队首节点
19.      return result                 #返回节点数据
```

（4）统计队列数据元素个数。

统计队列数据元素个数，就是要遍历链表中的每一个节点，补全 size 方法的实现代码。

```
20.    def size(self):                 #统计队列的数据元素个数
21.      current=self.head             #获取队首节点
22.      count=_____                   #初始化数据元素个数统计变量
23.      while current!=None:          #判断当前节点不为空节点
24.        count=_________             #数据元素个数增加一个
25.        current=current.next        #获取下一个节点
26.      return count                  #返回队列的数据元素个数
```

（5）判断队列是否为空。

```
27.    def isEmpty(self):              #判断队列是否为空
28.      return self.head==None
```

队列的链表实现

相对于列表实现而言，利用链表实现队列时，由于链表方式采用了对象引用来表示数据元素之间的关系，使得入队和出队操作更具灵活性。但对于统计队列元素个数，由于需要遍历链表中的每一个节点，其操作效率较低。

拓展练习

1. 已知队列（13，2，11，34，41，77，5，7，18，26，15），第1个进入队列的元素是13，则第5个出队列的元素是（　　），并说明理由。

A. 5　　B. 41　　C. 77　　D. 34

2. 假设有8个人（用p1，p2，p3，p4，p5，p6，p7，p8表示），依次前来排队买票，前4个人依次买完票离开，最后统计当前排队的人数。请利用队列抽象数据类型的链式存储实现方式，实现上述排队买票的过程和人数统计。

3. 用列表实现队列的方法中，如果队列的队尾放在列表的首端，请编程实现出队和入队的操作。

4.2　基数排序

1890年赫尔曼•霍尔瑞斯（Herman Hollerith）发明的“穿孔卡片制表机”将美国的人口普查工作从8年缩短为6个星期，其核心技术“基数排序”算法就是利用了队列的先进先出特点，实现了无比较排序，大大提高了工作效率。

本节重点学习基数排序的基本过程、算法及其实现，体会队列结构和迭代思想在其中的应用。

学习目标

- ★ 理解基数排序的基本过程。
- ★ 体会队列在基数排序中的应用。
- ★ 初步掌握基数排序算法。
- ★ 体会迭代思想在算法设计中的运用。

本节围绕“基数排序”项目展开学习，通过项目活动认识基数排序的基本过程，学习基数排序的算法和实现，并将迭代思想运用到算法之中。本节主要包含“体验手动基数排序”和“编程实现基数排序”两个任务。

任务一　体验手动基数排序

※ 活动1　热身运动：拆分数位

请拆分3999，28，109的数位，并将结果填写在表 4.2.1 中。

表4.2.1　拆分数位

数	千位数	百位数	十位数	个位数
3999	3	9	9	9

续表

数	千位数	百位数	十位数	个位数
28				
109				

这里所说的个位数、十位数、百位数、千位数是针对十进制记数而言的。个位数上是9，表示数值$9*10^0$；十位数上是9，表示数值$9*10^1$；百位数上是9，表示数值$9*10^2$；千位数上是3，表示数值$3*10^3$，即 $3999=3*10^3+9*10^2+9*10^1+9*10^0$。

基数

十进制记数是逢十进一，从零开始计数，数到十的时候，就要进一位，即变成 10，个位数上是 0，十位数上是 1。一个十进制数的每个数位上可能的数字是 0，1，2，…，9 中的某一个。十进制的基数是10。

二进制记数是逢二进一，从零开始计数，数到二的时候，就要进一位，即变成$10_{(2)}$。一个二进制数的每个数位上可能的数字是0，1。二进制的基数是2。

同理，一个 N 进制数是逢 N 进一，它的基数是N，采用N个不同的数字符号来表示。例如，十六进制的基数是16，采用0～9和A～F这16个符号来记数。

※ 活动2　自制数字纸牌接龙游戏

课间，小明跟同桌玩起了自制数字纸牌接龙游戏。他手中拿着15张纸牌，每张纸牌上有一个数，如图 4.2.1所示。

图 4.2.1　自制数字纸牌

这个游戏的操作主要是纸牌的分发和收集，小明用这副纸牌演示游戏的玩法如下。

第 1 次接龙

第1步：将如图4.2.1所示的纸牌从左到右依次抽出，以纸牌的个位数为准进行分发，个位数相同的纸牌放在同一列上，即个位数为0的放在第1列，个位数为1的放在第2列，以此类推。当纸牌的个位数相同的时候，先抽出的纸牌放在前面，后抽出的纸牌放在后面。结果如图4.2.2所示。

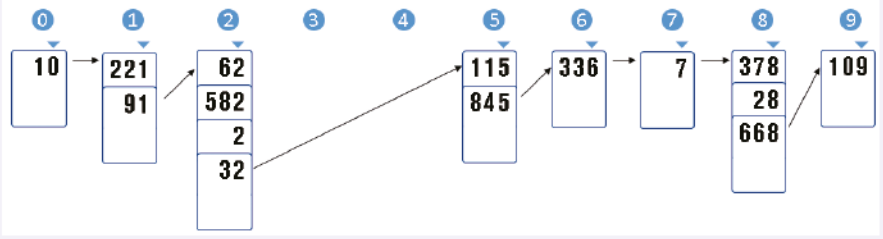


图 4.2.2　依据个位数分发的结果

第 2 步：将纸牌按图 4.2.2 中箭头所示的顺序从左到右从上到下收集，结果如图 4.2.3 所示。

图 4.2.3　第 1 次收集后纸牌的顺序

第 2 次接龙

第 1 步：将如图4.2.3所示的纸牌从左到右依次抽出，以纸牌的十位数为准分发，十位数相同的纸牌排在同一列上，结果如图 4.2.4 所示。

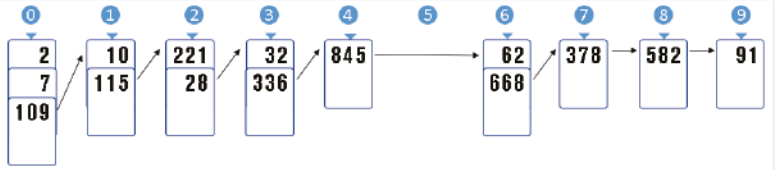


图 4.2.4　依据十位数分发的结果

基数排序不是基于元素的比较和移动来完成的，其原理是将元素按数位切割成不同的数字，然后按数位上的数字把元素分类。这种排序方法不仅适合整数，也适合日期、字符串等数据。

第 2 步：将纸牌按图4.2.4中箭头所示的顺序收集，结果如图4.2.5所示。

图 4.2.5　第 2 次收集后纸牌的顺序

第 3 次接龙

第 1 步：将如图4.2.5所示的纸牌从左到右依次抽出，以纸牌的百位数为准分发，百位数相同的纸牌排在同一列上。

图 4.2.6 中已经将前 5 张纸牌按百位数分发，请将后面的 10 张纸牌按百位数分发，把结果填写在图 4.2.6 中并标出表示收集顺序的箭头。

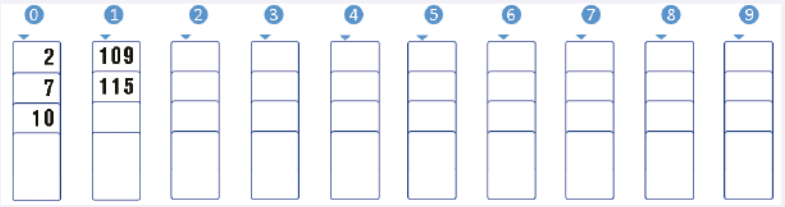


图 4.2.6　依据百位数分发的结果

第 2 步：将纸牌按从左到右、从上到下的顺序收集，把图4.2.7补充完整。

图 4.2.7　第 3 次收集后纸牌的顺序

经过3次接龙，纸牌按照其上面数的大小，从小到大排成了一排。

思考：如果数字的最大位数有5位，会经过多少轮接龙才能实现

排序呢?

在游戏中，对纸牌进行了多次分发和收集。第1轮以纸牌上的个位数为准分发纸牌，将纸牌排成不同的列，然后收集；第2轮以纸牌上的十位数为准分发纸牌，然后收集，以此类推，直到以所有纸牌上的最高位为准进行分发和收集。最后发现，纸牌按从小到大的顺序排好了。

分发时把纸牌分发到 10 个队列中，可以把这些队列分别叫 0 队、1 队、2 队、……、8 队、9 队，统称基数队列。每一轮收集时，从序号小的队列向序号大的队列收集，每一队列按从前向后的顺序收集。

● 基数排序

基数排序（radix sort）通过多轮分发和收集来完成排序。从个位开始，由低位到高位依次分发和收集。

每一次分发，以指定数位为准，把正整数分发到不同的基数队列中。收集时，按基数队列序号从小到大依次把正整数收集到一个队列中。

当对所有数位进行分发和收集后，排序就完成了。

对每个基数队列来说，先放入队列的正整数，最先从该队列中取出。最先放入收集队列中的正整数，最先被取出。因此，其数据处理具有先进先出的特点，可以使用队列作为实现基数排序的数据结构。

任务二　编程实现基数排序

※ 活动1　建立数据结构

编写函数radixSort(alist)实现基数排序。把手中的队列命名为mainQueue；把桌面上的基数队列命名为radixQueues，这是有10个元素的列表，每个元素都是一个队列。

mainQueue最开始保存用户要排序的数据，可以通过循环让列表alist中的待排序数据依次进入队列。

```
#基数排序主程序
def radixSort(alist):
    mainQueue=Queue()                          #建立mainQueue队列
    radixQueues=[Queue() for i in range(10)]          #建立基数队列
    for a in alist:                            #数据放入mainQueue队列
        mainQueue.enQueue(a)
```

※ 活动2　设计算法

基数排序的过程就是把队列mainQueue中的数据分发到列表radixQueues的队列中，然后再把radixQueues中的各个队列收集到队列mainQueue中。如果参与排序的最大正整数是k位，则整个排序中分发和收集将进行k轮。分发与收集从个位开始，直到第k位为止。

思考：用这种方法对一组二进制数据排序，需要几个队列？如果要排序的数据是十六进制的，又需要几个队列？

基数排序的算法描述如下。

（1）循环步骤（2）（3）共 k 轮。

（2）把队列mainQueue中的数据分发到radixQueues的各个队列中。

（3）把radixQueues各个队列中的数据收集到队列mainQueue中。

根据以上算法，补全下面的代码。

```
07.   k=len(str(max(alist)))                    #最大数的位数
08.   for i in range(1, k+1):                   #分发和收集k轮
09.     distribute(mainQueue, radixQueues, i)   #分发
10.     gather(mainQueue, radixQueues)          #收集
11.   alist.clear()                             #清空alist
12.   #把队列mainQueue中的数据放入列表
13.   while not _____:
14.     alist.append(mainQueue.deQueue())
```

在以上代码中，k=len(str(max(alist)))用于求列表中的最大数的位数。

接下来，对算法中调用的两个函数 distribute 和 gather 进一步细化，确定其算法。

函数 distribute 的功能是把 mainQueue 中的数据分发到基数队列 radixQueues 中，分发时以第 i 数位上的数字为准。这个函数依次取出队列 mainQueue 中的每个数据 a，取出第 i 数位上的数字 m，把 a 分发到 radixQueues 的第 m 个队列中，放在队列的最后面。怎样取得一个数 a 的第 i 位（i 从 1 开始）呢？以 a=1234 为例：

（1）取得第 1 位：a%10//1

（2）取得第 2 位：a%100//10

（3）取得第 3 位：a%1000//100

（4）取得第 4 位：a%10000//1000

可以看到，可以通过表达式得到第 i 数位上的基数：

a%(10**i)//(10**(i-1))

mainQueue 中的数据依次出队，然后根据基数入队到相应的队列中。

函数 distribute 的算法描述如下。

（1）从队列 mainQueue 中依次取出数据 a。

（2）如果是第 i 轮分发，则取出数据 a 的第 i 数位上的数字 m。

（3）把数据 a 入队到队列 radixQueues[m] 中。

以上算法可以通过下面的代码来实现。

```
#把mainQueue中的数据按第i位分发到队列radixQueues中
def distribute(mainQueue, radixQueues, i):
  while not mainQueue.isEmpty():
    a=mainQueue.deQueue()
    m=a%(10**i)//(10**(i-1))                #取出第i位上的数字
    radixQueues[m].enQueue(a)
```

函数 gather 的功能是收集 radixQueues 中各队列数据到 mainQueue 中，其算法描述如下。

（1）按序号从小到大收集 radixQueues 中各队列的数据。

（2）各个队列中的全部数据依次出队，然后将它们加入队列 mainQueue 中。

代码如下。

```
#把列表radixQueues中的数据收集到队列mainQueue中
def gather(mainQueue, radixQueues):
  for q in radixQueues:
    while not q.isEmpty():
      a=q.deQueue()
      mainQueue.enQueue(a)
```

※ 活动 3　编程实现

活动 2 已经完成了基数排序的算法设计和代码实现，请将活动2的代码集中输入到一个程序文件中，用以下数据对程序进行测试：

```
alist=[109,378,28,5,10,7,21,845,62,582,91,336,2,68,32]
radixSort(alist)
print(alist)
```

● 基数排序的基本思路

分发过程

分发的过程是按照数据的某个数位上的数字依次把队列 mainQueue 中的各个数据分发到相应的基数队列中，最先入队的数据处于队列的队首位置。

收集过程

收集的过程是按照基数队列下标从小到大的顺序，将所有基数队列中的数据移出队列，放入队列 mainQueue，最先入队的数据处于队列的队首位置。

迭代次数

因为要从低位到高位对数据的每一位进行一轮分发和收集，所以整个排序过程要进行k次分发和收集。k是需要排序的所有数据中最大值的位数，它可以通过以下表达式得到：

k=len(str(max(alist)))

拓展练习

1. 英语课上老师让每位同学各找十个与所学课文相关的单词，并汇总到科代表小明手中，请编写程序将这些单词按词典上的顺序排列。

提示：英文单词可以看作26进制的数。

2. 学校准备为每位同学制作一张生日卡片，为了方便管理，需要把同学们的身份证号码按生日排序，请编程实现。

提示：身份证号码最后一位可能是X，建议把身份证号码保存为字符串。

3. 基数排序为什么要从低位开始，而不是从高位开始？可以从高位开始吗？

4.3 排队取号模拟系统

现实生活中，越来越多的服务行业如餐馆、银行等使用了取号机。用户不必亲自排队，只要在取号机上轻轻一点，就可以拿到一个“号”，上面不仅显示排多少号，还显示前面正在等待服务的人数。用户只要等待叫号即可。这种取号、叫号排队的方式是一个典型的队列应用。

本节主要学习利用队列结构实现餐馆排队取号模拟系统，体会数据结构与算法的关系。

学习目标

- ★ 了解排队取号模拟系统的基本功能。
- ★ 掌握队列在排队取号模拟系统中的应用方法。
- ★ 体会排队取号模拟系统在辅助决策中的应用。

在很多人口密集的区域，用餐高峰时段，在餐馆需排队就餐是普遍现象。由于等待时间过久而流失顾客对餐馆来说是个损失。通过应用队列结构来模拟顾客排队取号用餐的过程，统计顾客平均等待时间，优化餐桌配置，缩短等待时间，可以减少顾客流失。

本节围绕“餐馆排队取号模拟系统”项目展开学习，通过项目活动来了解餐馆排队取号模拟系统的基本思路，体验队列在解决实际问题中的作用。本节主要包含“模拟餐馆排队取号”和“编程实现餐馆排队取号模拟系统”两个任务。

任务一　模拟餐馆排队取号

※ 活动1　体验餐馆排队取号

在餐馆就餐桌位满员的情况下，顾客需要排队等候就餐。餐馆服务人员会根据顾客用餐人数，从餐馆现有的餐桌类型，如“2人

桌”“4人桌”“8人桌”中选择一种，通过取号机给顾客打印一张排号单，如图4.3.1所示。

小明和同学一起到餐馆排队就餐，拿到了如图4.3.1所示的排号单。从排号单中可以看出，小明在“2人桌”中排在了_____号，目前在此队等候的共有____桌顾客。

图4.3.1　取号机和排号单

当有顾客用餐完毕后餐桌出现空余，系统就自动根据餐桌类型进行叫号。排在该类型餐桌队列最前面的顾客就可以前去用餐，该队列就减少一桌顾客。在小明和同学等待就餐的过程中，没有新顾客到来，“2人桌”叫号已经叫到A010号，此时还有_____桌顾客等候就餐。

● 排队取号

餐馆排队取号包括取号和叫号两个部分。取号就是根据所选餐桌类型生成排队号码等信息将顾客入队。叫号就是根据空闲餐桌类型，将排在该餐桌类型队列队首的顾客出队。

※ 活动2　模拟排队取号

假设餐馆共有10张餐桌，“2人桌”5张，桌号1~5；“4人桌”3张，桌号6~8；“8人桌”2张，桌号9~10。为了区分不同的餐桌类型，排队号码的开头用不同的字母表示。“2人桌”用A表示，“4人桌”用B表示，“8人桌”用C表示。

小明等两人12：08来到餐馆等待就餐，取到了A012号，排号单显示前面有4组顾客等候，则表示“2人桌”等待队列中有4组顾客，如图4.3.2所示。

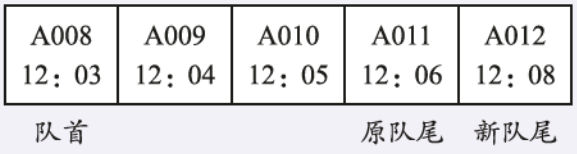


图4.3.2　“A012”入队

12：10时，系统叫号“A008”到5号桌就餐，表示空出一张“2人桌”，排在“2人桌”队首的顾客可以出队就餐，如图4.3.3所示。

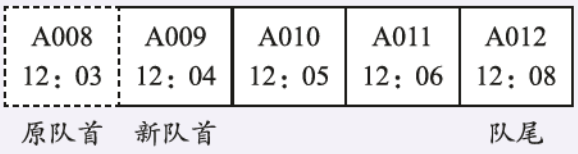


图4.3.3 “A008”出队

排在A008号的顾客排队等候时间为_____分钟。

小梅等四人12：20来到餐馆等待就餐，取到了“B008”号，当前“4人桌”的排队情况如图4.3.4所示，请按照上面的思路将B008排进队列。

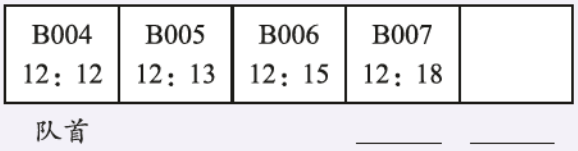


图4.3.4 “4人桌”排队情况

分析以上模拟过程可以发现，餐馆排队取号入队，总是排到不同餐桌类型的队尾。叫号就餐，总是从不同餐桌类型的队首开始。这个过程具有先进先出的特点，可以使用队列结构作为实现排队取号模拟系统的数据结构。

在取号叫号的过程中，排队取号系统还可以记录顾客的取号时间，统计顾客的等待时间和顾客用餐总数，以便作为餐桌优化配置的依据。

任务二 编程实现餐馆排队取号模拟系统

※ 活动1 建立数据结构

在排队取号用餐的过程中，与顾客有关的信息包括顾客所选的餐桌类型、顾客的用餐时间、顾客取号时间等。为了方便处理这些信息和操作，定义顾客类Guest，其Python代码如下。

```
class Guest:                          #定义顾客类
  def __init__(self, tableType):
    self.tableType=tableType          #顾客所选餐桌类型
    #随机生成顾客用餐时间
    self.stayTime=random.randrange(20, 60)
  #生成顾客取号信息
  def takeTicket(self, number,order,timeTick)
    #根据餐桌类型、排队号码、等候人数生成排号单
    self.ticket='%s%03d 前面还有%d位'% (self.tableType,
    number, order)
```

```
11.      #餐桌类型和排队号码
12.      self.tableTypeNum='%s%03d'% (self.tableType,number)
13.      self.arriveTime=timeTick                    #顾客取号时间
```

叫号过程中，需要根据餐桌类型和状态，呼叫相应餐桌队列中的顾客到指定餐桌就餐。为了方便处理餐桌信息，定义餐桌类Table，其Python代码如下。

```
14. class Table:                                    #定义餐桌类
15.   def __init__(self, id, type):
16.     self.id=id                                  #餐桌桌号
17.     self.type=type                              #餐桌类型（'A','B','C'）
18.     self.status='free'                          #餐桌状态（free, busy)
19.     self.remain=0                               #顾客占用时长
20.   def setBusy(self, remain):                    #设置餐桌状态
21.     self.status='busy'                          #设置桌子为“在用”状态
22.     self.remain=remain                          #设置占用时长
23.   def tickRemain(self):                         #餐桌用餐状态检测
24.     if self.remain>0:                           #判断是否还有占用时长
25.       self.remain-=1                            #占用时长减少1
26.     if self.remain==0:                          #判断占用时长是否为零
27.       self.status='free'                        #设置桌子为“空闲”状态
```

排队取号模拟系统的核心是实现取号和叫号功能，为了方便实现排号和取号的操作，定义排号机类QueueSystem。排队取号过程中，需要根据不同的就餐人数选择在相应的餐桌队列中排队，根据任务一活动2的描述，需要建立三个队列，分别用来存放A、B、C三种餐桌类型的排队顾客Guest对象。为了方便处理，创建队列字典guestQueues和排队号码字典guestCount，其Python代码如下。

```
28. class QueueSystem:                  #定义排号机类
29.   def __init__(self):               #排号机构造函数
30.     #顾客队列
31.     self.guestQueues={'A': Queue(), 'B': Queue(), 'C': Queue()}
32.     self.guestCount={'A': 0, 'B': 0, 'C': 0}          #排队号码
33.     self.totalWaitTime=0            #总等待时间
34.     self.totalGuest=0               #总就餐次数
```

※ 活动2　设计算法

排号机的主要功能包括取号、叫号和显示模拟系统统计信息，各功能模块的算法描述如下。

取号过程的算法描述如下。

（1）根据顾客所选餐桌类型，生成该餐桌类型的排队号码。

（2）生成顾客取号信息。

（3）顾客入队。

（4）显示取号信息。

根据上述算法，定义取号函数assignTicket(self, guest, timeTick)，参数self表示取号机，参数guest表示取号的顾客，timeTick表示取号的时间。请补全下面的代码。

```
35. def assignTicket(self, guest, timeTick):      #取号函数
36.   table=guest.tableType                       #顾客的餐桌类型
37.   queue=self.guestQueues[table]               #顾客所排的餐桌队列
38.   self.guestCount[table]+=1                   #生成排队号码
39.   guest.takeTicket(self.guestCount[table], queue.size(),
40.   timeTick)                                   #顾客取号
41.   queue.enQueue(_____)                        #顾客入队
42.   #显示顾客取号信息
43.   print('顾客取号：', guest.ticket,'取号时间：',timeTick)
```

叫号过程的算法描述如下。

（1）可以用餐餐桌类型队列中的顾客出队。

（2）设置就餐餐桌的状态和用餐时间。

（3）累计用餐总人数和总等待时间。

（4）显示叫号信息。

根据上述算法，定义叫号函数arrangeTable(self, table, timeTick)，参数self表示取号机，参数table表示可以用餐的餐桌，timeTick表示叫号的时间。请补全下面的代码。

```
44. def arrangeTable(self, table, timeTick):      #叫号函数
45.   #判断队列是否为空
46.   if not self.guestQueues[table.type].isEmpty():
47.     #顾客出队
48.     guest=self.guestQueues[table.type].____________
49.     table.setBusy(guest.stayTime)             #设置餐桌状态为占用
50.     self.totalGuest+=1                        #累计用餐总次数
51.     #累计总等待时间
52.     self.totalWaitTime+=timeTick-guest.arriveTime
53.     #显示叫号信息
54.     print('叫号：', guest.tableTypeNum, '到桌号：', table.id)
```

显示模拟系统统计信息：

定义函数showResult(self, totalTime)，参数self表示取号机，参数

totalTime表示模拟的总时长，代码如下。

```
def showResult(self, totalTime):        #显示模拟信息函数
    print('*' * 40)                     #输出一行*
    print('本次模拟一共', totalTime, '分钟')  #显示模拟总时长
    #显示模拟期间总用餐桌数
    print('共有', self.totalGuest, '桌顾客用餐')
    #顾客平均等待时间
    print('每桌顾客平均等待时间为', round(self.totalWaitTime
    /self.totalGuest), '分钟')
    print('模拟结束时各队列情况如下：')
    for q in self.guestQueues:          #显示各队列剩余排队情况
        print('桌型：', q, '还有', self.guestQueues[q].size(), '桌
        顾客在排队')
    print('*' * 40)                     #输出一行*
```

※ 活动3　编程实现

活动2已经完成了排队取号模拟系统基本功能的算法设计和代码实现，接下来利用排队取号系统模拟餐馆就餐客流排队情况。首先，我们确定模拟系统的时间流逝，是一个以分钟为单位的整数增长过程。然后，在每一分钟内，模拟下列事件：

（1）用随机数按照一定的概率来确定是否有顾客排队取号就餐；

（2）如果有顾客排队取号，用随机数按照一定概率决定就餐餐桌的类型，执行取号；

（3）检查餐桌状态，如果有餐桌可以就餐，则执行叫号；

（4）显示模拟统计信息。

排队取号模拟系统的主程序代码如下，请补全下面的代码。

```
import random
#变量randomBox用于设定模拟来客的概率，20%概率2人桌，10%概率4人
桌，5%概率8人桌
randomBox=[None]*65+['A']*20+['B']*10+['C']*5
#餐桌设置，5张2人桌，3张4人桌，2张8人桌
tableSetting='AAAAABBBCC'
#生成餐桌对象列表
tableList=[Table(i+1, tableSetting[i]) for i in range
(len(tableSetting))]
qsys=_____          #生成一个排号机对象
```

```
totalTime=4 * 60              #模拟时间长度为4小时，最小单位为分钟
for timeTick in range(__________):     #按分钟进行迭代循环
  dice=random.choice(randomBox)        #以设定概率随机模拟来客
  if dice is not None:                 #判断是否有顾客来
    guest=___________                  #生成顾客对象
    qsys.assignTicket(guest, timeTick)  #顾客取号
  for table in tableList:
    table.tickRemain()                 #餐桌用餐状态检测
    if table.status=='free':
      qsys.arrangeTable(table, timeTick)          #叫号入桌
qsys.showResult(totalTime)             #显示模拟统计信息
```

将上述代码输入一个文件，执行该程序，显示的模拟系统统计信息如图4.3.5所示。

```
****************************************
本次模拟一共 240 分钟
共有 56 桌顾客用餐
每桌顾客在用餐前平均等待时间为 37 分钟
模拟结束时各队列情况如下：
桌型： A 还有 17 桌顾客在排队
桌型： B 还有 12 桌顾客在排队
桌型： C 还有 1 桌顾客在排队
****************************************
```

图 4.3.5 显示模拟系统统计信息

依据排队取号模拟系统计算出顾客平均等待时间，可以通过修改餐桌的种类和数量tableSetting、顾客的构成概率randomBox，来优化餐桌配给，减少顾客等待时间，提高满意度。

拓展练习

1. 在队列中，存取数据的原则是（　　）。

A. 先进先出　　B. 先进后出　　C. 后进后出　　D. 没有限制

2. 为了缩短顾客的等待时间，可以灵活安排顾客的排队方式。例如：8人桌空闲，且没有人等候的情况下，可以把两个4人桌合并到8人桌用餐。请修改排队叫号模拟系统，实现该功能。

单元学习评价

本单元学习了队列结构的基本概念及特征，亲历了利用队列解决问题的一般过程，体验了迭代思想在问题解决中的应用。你了解队列的相关概念了吗？你是否能运用队列和迭代思想解决实际问题？请参与小组交流并反思，开展自评或小组评价。

一、在解决以下问题的过程中，可以利用队列结构解决的是（　　）。

A. 求解斐波那契数列　　B. 打印机任务调度

C. 银行取号叫号机　　D. 数据排序

二、约瑟夫环是一个数学应用问题：已知n个人（以编号1，2，3，…，n分别表示）围坐在一张圆桌周围，从编号为1的人开始报数，数到m的那个人出列；相邻的下一个人又从1开始报数，数到m的那个人又出列；依此规律重复下去，直到圆桌周围的人全部出列。求出最后一个出列的人的编号。

1. 问题分析。

解决此问题，可以用迭代还是递归方法？为什么？

2. 设计数据结构。

（1）采用什么数据结构保存数据？

（2）为什么采用这种数据结构？

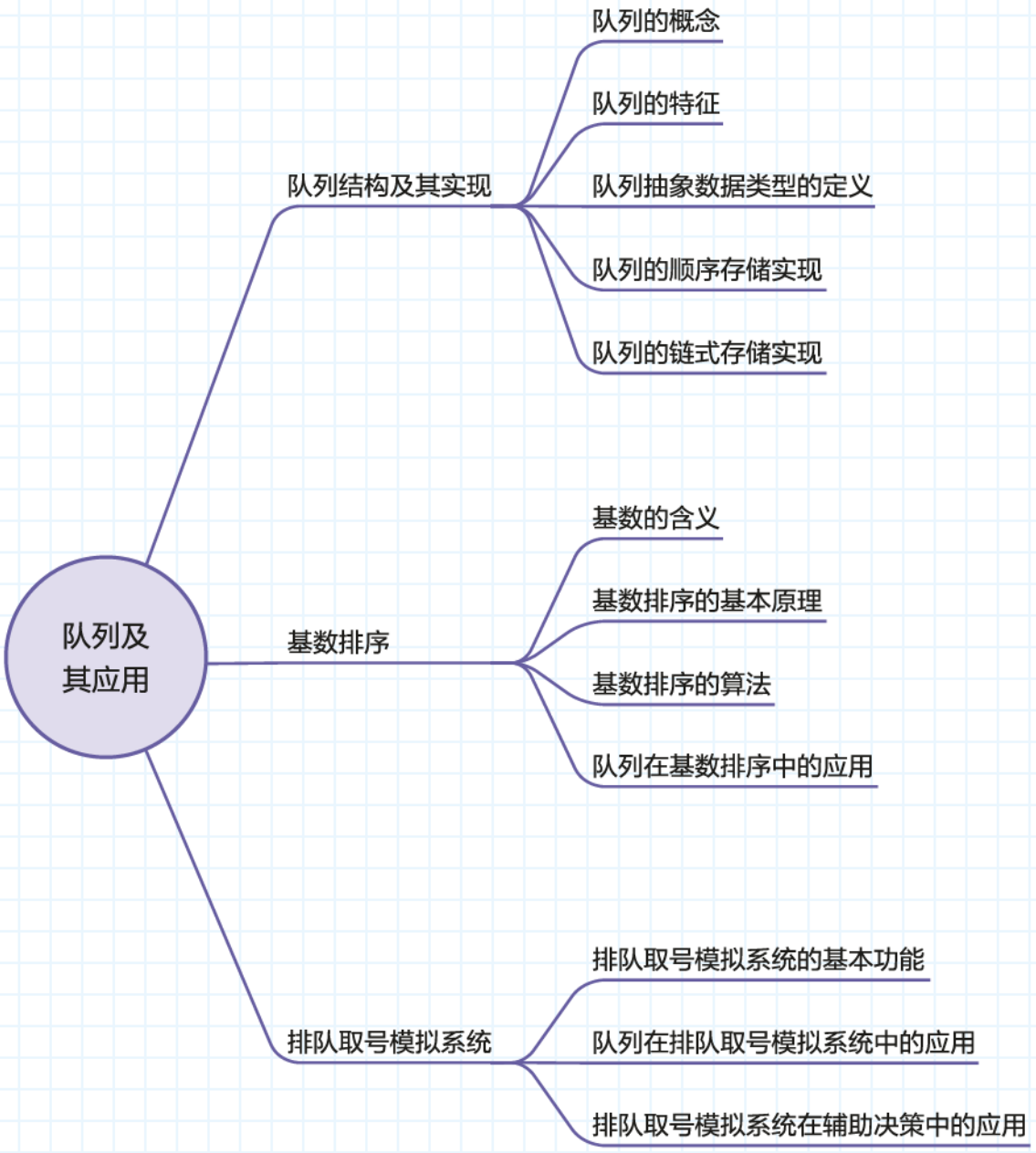
3. 设计算法。

（1）怎样输入、输出数据？

（2）画出算法流程图。

4. 编程实现。

单元学习总结


队列及
其应用
队列结构及其实现
队列的概念
队列的特征
队列抽象数据类型的定义
队列的顺序存储实现
队列的链式存储实现
基数排序
基数的含义
基数排序的基本原理
基数排序的算法
队列在基数排序中的应用
排队取号模拟系统
排队取号模拟系统的基本功能
队列在排队取号模拟系统中的应用
排队取号模拟系统在辅助决策中的应用

第 5 单元　栈及其应用

栈与队列一样，也是一种操作受限的线性结构，但与“先进先出”的队列不同的是，它具有“后进先出”的特征。栈的应用非常广泛，在进制转换、表达式求值和迷宫求解等实际问题中，都会用到栈结构来辅助求解。另外，函数调用及返回也是通过栈结构来实现的。

本单元通过“单行道车辆调度”项目，理解栈结构的基本概念及特征，学习栈的两种实现方法，通过“符号匹配识别”和“算术表达式求值”项目体验利用栈结构解决实际问题的方式和过程，形成学科思维。

5.1 栈结构及其实现

浏览网页时，单击一个个超链接，可以不断打开新的页面，单击浏览器上的“后退”按钮，又可以一步步返回到浏览过的网页。类似地，各类编辑软件也大都提供编辑操作的历史记录功能，保存用户的每次编辑操作，如果出现错误操作，单击“撤销”按钮，就可以一步步返回到此前的编辑状态。在这些习以为常的操作行为中，蕴含着一个基础但影响深远的数据结构——栈。

本节主要学习栈的基本概念及其特征，并掌握栈的抽象数据类型的定义及其实现方法。

学习目标

- ★ 理解栈的基本概念及其特征。
- ★ 掌握栈的抽象数据类型的定义。
- ★ 掌握栈的两种实现方法。

新华路为由北向南的单行道。一场暴风雨过后，一棵大树被连根拔起，粗壮的树干倾压在道路中间，四辆行进中的轿车被堵在单行道上，进退两难，其中黄色轿车离大树倾倒地点最近，如图5.1.1所示。

本节将围绕“单行道车辆调度”项目展开学习，通过项目活动深入理解栈的基本概念及其特征，通过实例活动感知栈的基本操作，掌握栈的抽象数据类型的定义及栈的实现方法。本节主要包含“体验单行道车辆调度”和“编程实现单行道车辆调度”两个任务。

任务一 体验单行道车辆调度

※ 活动1 记录车辆停放信息

道路被大树堵塞后，交通协管员被立即派往新华路负责疏散车

辆和行人。通过查看道路监控录像得知，大树倒下的时间大概为早上 8 点，车牌号为 A101 的黄色轿车 8：05 驶入新华路，2 分钟后车牌号为 B102 的绿色轿车停靠在黄色轿车后面，随后车牌号为 C102 的蓝色轿车 8：08 驶入新华路，2 分钟后红色轿车 D102 停靠在其后。

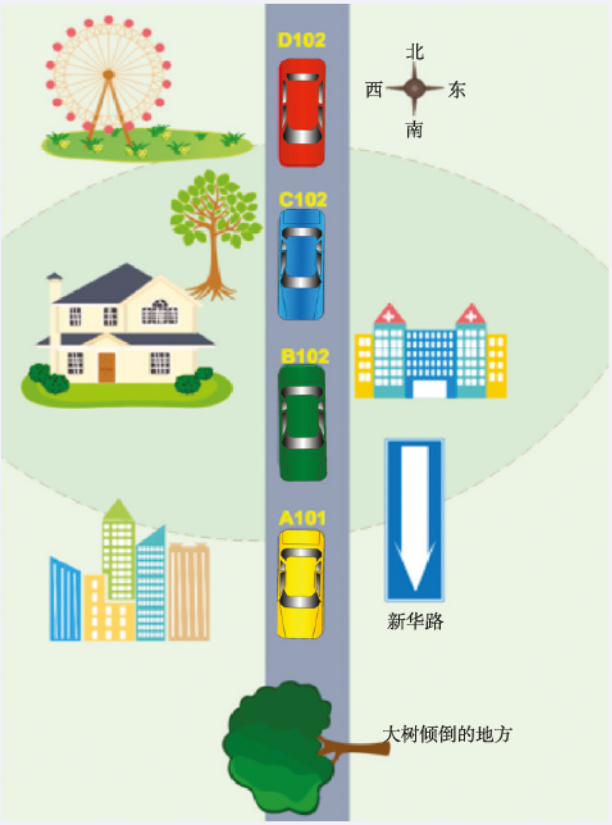


图 5.1.1　新华路交通状态

根据轿车停放时间的先后顺序，完成图 5.1.2。

8：10 驶入	车牌号：
8：08 驶入	车牌号：
8：07 驶入	车牌号：B102
8：05 驶入	车牌号：A101

图 5.1.2　新华路车辆信息 1

※ 活动 2　根据路况更新车辆停放信息

线性结构的两端在不同情况下名称不同，有时称为“左”“右”端，有时称为“前”“后”端。

红色轿车司机等了半个小时后，还是不见前方车辆有启动的迹象，便于 8：40 倒车驶回主路，选择其他路线。请用笔画掉已驶出的车辆，更新图 5.1.2 中的信息，得出图 5.1.3。

8：42分在主路行驶的车牌号为E201的黑色轿车进入新华路，减速慢行后停在蓝色轿车后面。请根据新华路路况更新图5.1.4。

8：40驶出	车牌号：
8：08驶入	车牌号：
8：07驶入	车牌号：B102
8：05驶入	车牌号：A101

图5.1.3 新华路车辆信息2

8：42驶入	车牌号：
8：08驶入	车牌号：
8：07驶入	车牌号：B102
8：05驶入	车牌号：A101

图5.1.4 新华路车辆信息3

栈

栈是一种出入有序的数据集合。作为一种操作受限的线性结构，数据元素的插入和删除都仅发生在同一端，这一端为栈顶（Top），相应地，另一端则为栈底（Bottom）。数据元素的插入操作一般称为入栈（Push），又称进栈；删除操作则称为出栈（Pop），又称退栈。

栈底固定而栈顶浮动，数据元素像积木那样一层层堆起来，后面加入的会被放置在最上面。

※ 活动3 新华路车辆调度操作

交通协管员被告知事故现场短时间内无法清理完毕，建议出行者选择其他的行车路线。与四位司机沟通后，他们选择规划新路线去往目的地。下一步新华路上的车辆该如何进行指挥调度呢？

请填写下面的车辆调度信息。

第一步：车牌号为_____的轿车倒车驶出新华路，这时新华路还停有_____辆轿车，下一辆即将驶出的轿车车牌号为_____。

第二步：车牌号为_____的轿车倒车驶出新华路，这时新华路还停有_____辆轿车，下一辆即将驶出的轿车车牌号为_____。

第三步：车牌号为_____的轿车倒车驶出新华路，这时新华路还停有_____辆轿车，下一辆即将驶出的轿车车牌号为_____。

第四步：车牌号为_____的轿车倒车驶出新华路，这时新华路还停有_____辆轿车，道路已被清空。

分析以上调度过程，思考并回答下面的问题。

（1）最早驶入新华路的车辆为_____，最晚驶出新华路的车辆为_____。

（2）最晚驶入新华路的车辆为_____，最早驶出新华路的车辆为_____。

● 栈的特征

数据元素入栈和出栈的次序正好相反，并且所有操作都是对栈顶数据元素进行，如图 5.1.5 所示。

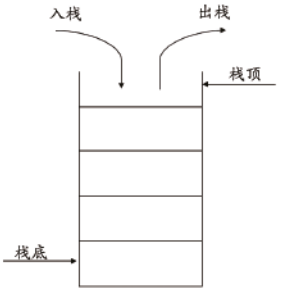


图 5.1.5　栈示意图

距离栈底越近的数据元素，留在栈中的时间就越长，而最新加入栈的数据元素会被最先移除，这种特点通常称为“后进先出”，即 Last In First Out，简称 LIFO。

● 栈抽象数据类型

从以上活动可以看出，栈的基本操作主要包括创建空栈、入栈、出栈、查看栈顶数据元素及判断栈是否为空。为了便于在程序中使用栈解决问题，需要定义栈抽象数据类型（ADT Stack）。

栈抽象数据类型的接口如下。

ADT Stack：

- Stack()：创建并返回一个不包含任何数据元素的空栈。
- push(item)：将数据元素 item 压入栈顶，无返回值。
- pop()：弹出并返回栈顶的数据元素。
- peek()：查看、返回但不移除栈顶的数据元素。
- isEmpty()：判断是否为空栈，返回布尔型的值。
- size()：计算并返回栈中数据元素的个数。

※ 活动 4　实现车辆调度过程

了解了栈抽象数据类型的基本操作，我们可以利用栈来实现新华路车辆调度过程。用车牌号来表示轿车，补全下面的代码。

```
01. s=Stack()                          #创建栈类的实例 s
02. s.push("A101")                     #车牌号为 A101 的黄色轿车入栈 s
03. _____________                      #车牌号为 B102 的绿色轿车入栈 s
04. _____________                      #车牌号为 C102 的蓝色轿车入栈 s
05. _____________                      #车牌号为 D102 的红色轿车入栈 s
06. _____________                      #查看栈中轿车数量
07. _____________                      #车牌号为 D102 的红色轿车出栈 s
08. _____________                      #查看栈顶是什么轿车
09. _____________                      #车牌号为 E201 的黑色轿车入栈 s
```

```
10. _____                    #车牌号为E201的黑色轿车出栈s
11. _____                    #车牌号为C102的蓝色轿车出栈s
12. _____                    #车牌号为B102的绿色轿车出栈s
13. _____                    #车牌号为A101的黄色轿车出栈s
14. _____                    #查看是否为空栈
```

任务二　编程实现单行道车辆调度

栈作为一种受限制，仅在一端进行操作的线性结构，有顺序存储和链式存储两种实现方式。

※ 活动1　用顺序存储实现栈

栈抽象数据类型主要包括创建栈、入栈、出栈、判断栈是否为空、查看栈顶数据元素和返回栈中数据元素个数。用Python内置的数据类型列表来实现栈的操作如下所示。

（1）创建空栈。

创建空栈，就是建立一个空的列表，用items来保存栈中的数据元素。

```
01. #创建Stack类
02. class Stack:
03.   def __init__(self):                #栈初始化为空列表
04.     self.items=[]
```

（2）判断栈是否为空。

```
05.   def isEmpty(self):                #判断是否为空栈
06.     return self.items==[]
```

（3）入栈。

入栈就是在栈顶（列表末尾）加入新的数据元素。例如，车牌号为F201的轿车驶入单行道，如图5.1.6所示。请补全下面的代码。

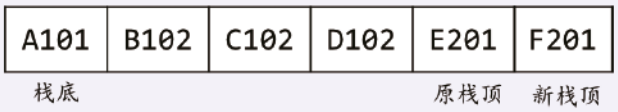


图5.1.6　F201入栈

```
07.   def push(self, item):              #入栈操作
08.     self.items._____
```

（4）出栈。

出栈是将栈顶数据元素弹出并返回。例如，车牌号为F201的轿车

驶出单行道，如图 5.1.7 所示。请补全下面的代码。

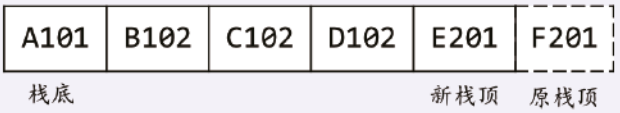


图 5.1.7　F201出栈

```
09.   def pop(self):                          #出栈操作
10.     if self.isEmpty():
11.       return self.items
12.     else:
13.       return self.items________
```

（5）查看栈顶数据元素。

列表的最后一项即为栈顶数据元素。请补全下面的代码。

```
14.   def peek(self):                         #查看栈顶数据元素
15.     if self.isEmpty():
16.       return self.items
17.     else:
18.       return self.items[_______________]
```

（6）计算并返回栈中数据元素的个数。

```
19.   def size(self):                         #返回栈中数据元素的个数
20.     return len(self.items)
```

将实现栈抽象数据类型顺序存储的代码输入到文件中，保存为 stack.py，利用单行道车辆调度测试各个接口的效果。请补全下面的代码。

```
01. from stack import Stack      #导入栈类
02. s=________                   #创建栈对象
03. s.push("A101")               #车牌号为A101的轿车驶入单行道
04. _____________                #车牌号为B102的轿车驶入单行道
05. _____________                #车牌号为B102的轿车驶出单行道
06. print(_________)             #显示当前单行道车辆总数
```

● 栈的顺序存储实现

若选用列表的首端（index=0）为栈顶，则栈的操作需要用列表的insert和pop 方法来实现。

栈的列表实现可以将列表的任意一端（index=0 或者−1）设置为栈顶，上面的活动我们选用列表的末端（index=−1）作为栈顶，栈的操作通过列表的append和pop方法实现。

※ 活动2 用链式存储实现栈

根据链表的特征分析链表实现栈的基本操作并补全下面的代码。

（1）创建空栈。

栈的初始化，栈顶节点引用为空。

```
01. class Stack:                          #创建栈类
02.   def __init__(self):
03.     self.head=None                    #栈初始化为空链表
```

（2）判断栈是否为空。

通过判断栈顶节点引用是否指向空来判定栈是否为空。

```
04.   def isEmpty(self):                  #判断栈是否为空
05.     return self.head==None
```

（3）入栈操作。

入栈的链表实现是通过在栈顶加入新节点完成的。例如，车牌号为E201的轿车驶入单行道，如图5.1.8所示。

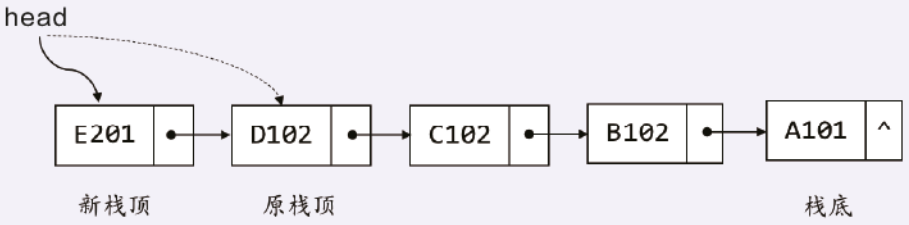


图5.1.8 E201入栈

```
06.   def push(self, item):              #入栈
07.     temp=Node(item)                   #生成新节点
08.     ____________________              #新节点引用指向原栈顶节点
09.     self.head=temp                    #栈顶节点引用指向新节点
```

（4）出栈操作。

出栈的链表实现是通过弹出栈顶节点完成的。例如，车牌号为E201的轿车驶出单行道，如图5.1.9所示。请补全下面的代码。

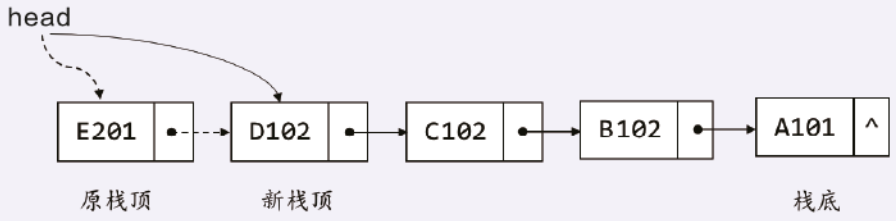


图5.1.9 E201出栈

```
10.   def pop(self):                    #出栈
11.     if self.isEmpty():              #判断栈是否为空
12.       print("Can't pop from empty stack.")
13.     else:
14.       temp=self.head.data           #保存原栈顶节点数据元素
15.       ____________________          #栈顶节点引用指向下一个节点
16.       return temp                   #返回原栈顶节点数据元素
```

（5）查看栈顶数据元素。

```
17.   def peek(self):                   #查看栈顶数据元素
18.     if not self.isEmpty():
19.       return self.head.data
```

（6）查看栈中数据元素的个数。

```
20.   def size(self):                   #查看栈中数据元素的个数
21.     count=0
22.     index=self.head
23.     while index!=None:              #判断节点是否为空
24.       count+=1
25.       index=index.next
26.     return count
```

● 栈的链式存储实现

栈的另一种实现方式是基于节点引用的链式存储方式。数据元素被存放在节点对象中，每个节点对象除了拥有存储数据元素的数据域外，还包括对下个节点的引用。显然，当节点为栈底时，引用为空，如图 5.1.10 所示。

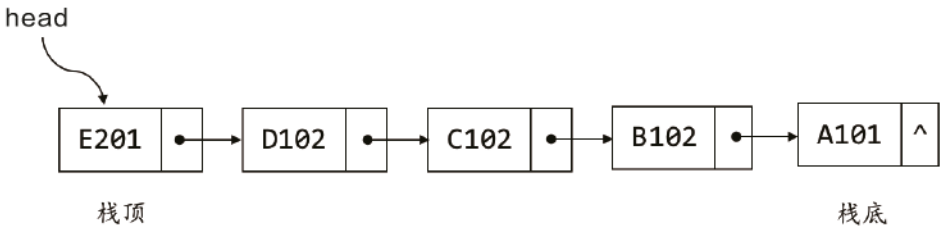


图 5.1.10　栈的链式存储实现

作为只在同一端进行数据元素的插入和删除操作的特殊线性结构，栈的操作并不会发生大量的数据移动。因此，栈的顺序存储和链式存储的性能差异并不大。但链式存储需要额外的空间来存储后继节点的地址，因此栈的实现方式需要根据实际情况灵活选择。

拓展练习

1. 现有单行道路况信息如图5.1.11所示，车牌号为G201的车辆驶入单行道。请根据栈的列表实现方法，把列表的首端（index=0）设置为栈顶，实现栈抽象数据类型接口，并测试车牌号为G201的车辆驶入单行道后车辆的总数量。

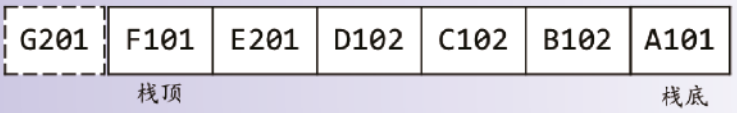


图5.1.11 单行道车辆停放示意图

2. 回文句是一种句型，一个句子如果正着读与倒着读的意思一样，就可以称为回文句。例如，蜜蜂酿蜂蜜；风扇能扇风；奶牛产牛奶；清水池里池水清；静泉山上山泉静等。

为了减少大量机械的重复操作，小明想用编程来实现自动检测回文句。你觉得他的想法可行吗？如果可行，请协助他通过编程判定下面的句子是否为回文句。

（1）上海自来水来自海上；

（2）雾锁山头山锁雾；

（3）天连水尾水连天；

（4）院满春光春满院；

（5）门盈喜气喜盈门。

5.2　符号匹配问题

栈的应用非常广泛，所解决的问题也非常有趣。特别是在符号匹配问题中，栈发挥着重要的作用。例如：编辑文章时需要识别书名号、双引号等符号是否正确匹配；程序编译和运行时需进行语法检查，即检验各类符号是否正确匹配等。

本节主要学习符号匹配的含义及原理，掌握用栈解决符号匹配问题的方法，进而体会模块化思想在问题解决过程中的运用。

学习目标

★ 理解符号匹配的基本原理。

★ 体会栈在解决符号匹配问题中的应用。

★ 初步掌握用栈来识别符号匹配的方法。

小明作为一名志愿者，要协助出版社工作人员进行符号匹配识别的工作。经过认真的学习和练习，他初步掌握了符号匹配识别的人工方法。鉴于人工识别耗时久，正确率低，他利用业余时间刻苦钻研，用计算机程序设计实现了符号匹配识别的自动化。不仅提升了工作效率，而且正确率也有了显著提高，受到出版社叔叔阿姨的一致好评。

本节将围绕“符号匹配识别”项目展开学习，通过项目活动理解符号匹配的含义，认识栈在解决符号匹配问题中的重要作用，并掌握用栈解决符号匹配问题的方法。本节主要包含“体验手动符号匹配”和“编程实现符号匹配”两个任务。

任务一　体验手动符号匹配

※ 活动 1　手动检查符号是否匹配

暑假，小明去出版社进行社会实践，跟着编辑老师学习审稿。他

主要负责检查稿件中的书名号、双引号、单引号等成对出现的符号是否匹配。例如：以下内容摘自小明的稿子。

> 《算法与数据结构》一书中提到："我们定义'类'（class）去描述数据的外观（状态）和功能（行为）。'类'类似于抽象数据类型，'类'的用户只能看到数据项的状态和行为。数据项在面向对象的范式里被称为对象（objects）。"

小明审稿时边读边在稿子上做标注。

（1）读到第一个左书名号，在左书名号下方标注①。

（2）读到右书名号，标注①，表示与左书名号匹配成功。

（3）读到左双引号，标注②。

（4）读到左单引号，标注③。

（5）读到与前一个左单引号对应的右单引号，便在右单引号下标注③。

按照这种方法进行下去，标注结果如下。

> 《算法与数据结构》一书中提到："我们定义'类'（class）
> ① ① ② ③ ③④ ④
> 去描述数据的外观（状态）和功能（行为）。'类'类似于抽象
> ⑤ ⑤ ⑥ ⑥ ⑦ ⑦
> 数据类型，'类'的用户只能看到数据项的状态和行为。数据项
> ⑧ ⑧
> 在面向对象的范式里被称为对象（objects）。"
> ⑨ ⑨ ②

标注完成后，小明进行匹配检查，发现标注的段落文本中的符号是匹配的，于是得出结论：该段落符号匹配正确。

下面的内容是小明审稿过程中遇到的第2段话，完成标注后，结果如下。

> 队列（Queue）是一系列有顺序的元素的集合，新元素的加
> ①
> 入发生在队列的一端，这一端叫作"队尾（"rear），已有元素
> ② ③② ③
> 的移除发生在队列的另一端，叫作"队首"（front）。
> ④ ④⑤ ⑤

小明进行符号匹配检查，发现标注的段落文本中②左双引号与右双引号是成对出现的，但是匹配失败，这是怎么回事？

● 符号匹配

符号的使用需要遵循一定的平衡原则：首先，每个左符号要恰好对应一个右符号；其次，每对左右符号要正确地嵌套，不能互相交错。如图 5.2.1 所示，符号串中每个左符号都有对应的右符号，且左右符号配对次序正确，因此该符号串符号匹配正确。

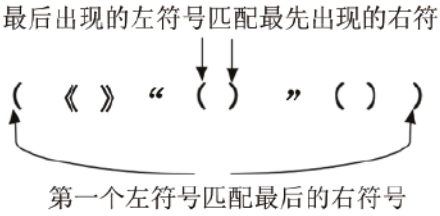


图 5.2.1　符号匹配示意图

通过活动1，我们可以总结出以下符号匹配失败的三种可能：

（1）左右符号配对次序不正确；

（2）右符号多于左符号；

（3）左符号多于右符号。

※ 活动2　单人符号纸牌匹配游戏

午休间隙，为了活跃气氛，小明提议大家一起玩一个自制符号纸牌游戏。他拿起待审的书稿随机选取了一段文字，依次摘取段落中的符号(<({})>)，并将这些符号依次画在了纸牌上，如图5.2.2所示。

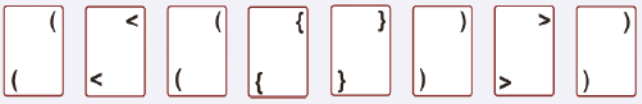


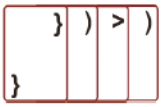
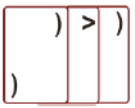
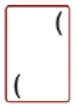
图 5.2.2　符号纸牌的排列

将纸牌按从左到右的顺序叠成扇形拿在手中，第一张在最左边。从手中一张张依次取牌。具体操作如表5.2.1所示。请补全表中的操作步骤。

表 5.2.1　单人符号纸牌匹配游戏

步骤	操作	桌上的牌	手中的牌
1	取出手中第一张“（”牌并放在桌面上	(	< ({ }) >)
2	取出“<”并放在“（”上面	< (	({ }) >)

续表

步骤	操作	桌上的牌	手中的牌
3	取出“（”并放在“<”上面		
4	取出“{”并放在“(”上面		
5	取出“}”与“{”匹配，符号类型一致，匹配成功，取走“{”和“}”		
6	取出“)”与“(”匹配，符号类型一致，匹配成功，取走“(”和“)”		
7	取出“>”与____匹配，符号类型____，匹配____，取走____和____		
8	取出“)”与____匹配，符号类型____，匹配____，取走____和____。此时桌面上已没有左符号牌，并且手中没有剩余的牌，因此符号匹配成功		

取完手中所有符号牌且桌面上左符号牌区域为空，说明符号串匹配成功。

此轮游戏结束后，小明和出版社工作人员意犹未尽，又随机选取了如下三个符号串继续自制符号纸牌游戏，符号匹配结果会是怎样的呢？

◆（‘(《》)’）“{}()””：符号匹配_____。

◆（‘(《》)’）“{}()”{(：符号匹配_____。

◆（‘(《》)’）“{(})”：符号匹配_____。

通过以上游戏，思考并总结出以下单人符号纸牌匹配游戏规则。

（1）从左到右依次取符号牌直到结束。

（2）取出的牌如果是_____，放在桌面上叠成一摞，后取出的牌放在上面。

（3）取出的牌如果是_____，则检查它是否与桌面上最上面的那张牌配对（即左右符号匹配）。若不是，则表明匹配失败，游戏结束。若是，将这对牌取走，继续游戏。如果此时桌面上没牌，表明匹配失败，游戏结束。

（4）当手中的牌已发完，但是桌面上还有牌，则匹配_____，游戏结束。

（5）当手中的牌已发完，并且桌面上也没有牌，则匹配_____，游戏结束。

通过以上游戏发现，先放在桌面上的符号后匹配，后放在桌面上的符号先匹配，其特点就像栈一样，可以利用栈来实现符号匹配。

任务二　编程实现符号匹配

※ 活动 1　建立数据结构

符号匹配过程中，先读取的左符号，匹配后读取的右符号，而后读取的左符号，则匹配先读取的右符号。这种次序反转的匹配过程，恰好符合栈的特性。因此，我们可以把手中的各种符号牌利用字符串存放，桌面上叠放的左符号牌则存放在栈中。

定义 parChecker(symbolString) 函数以进行符号匹配检测，参数 symbolString为需要检验符号匹配的符号串，变量s为存放左符号牌的栈。请补全下面的代码。

```
01. from stack import Stack                    #导入Stack类
02. checkString="(<({})>)"                     #初始化需要检验的符号串
03. #定义符号匹配检验函数parChecker
04. def parChecker(symbolString):
05.     s=Stack()                              #创建一个空栈s
```

※ 活动 2　设计算法

用栈解决符号匹配检测问题的算法描述如下。

（1）从左到右依次读取存放各类符号的字符串。

（2）如果为左符号则进栈。

（3）如果为右符号，则判断是否为空栈。如果是，此次匹配失败；否则，取栈顶符号并判断跟右符号是否同类。如果是，此次匹配成功，否则此次匹配失败。

（4）如果符号字符串结束且栈空了，则匹配成功，否则匹配失败。

根据以上算法，请补全下面的代码。

```
    for index in range(len(symbolString)):  #依次扫描所有字符
      symbol=symbolString[index]
      if symbol in "<[({":   #如果为左符号则进栈
        s.push(_______)
      else:           #如果为右符号则需要判断是否与栈顶左符号匹配
        if s.isEmpty():      #如果栈为空则匹配失败
          break              #跳出循环
        else:
          top=___________
          #取栈顶左符号进行匹配判定
          if not matches(top,symbol):
            break
    else:
      #已读取的全部符号匹配成功且左符号栈为空则符号串匹配成功
      if s.isEmpty():
        return True
    return False
```

定义matches(open,close)函数来判定栈顶的左符号是否跟右符号属于同一类，参数open存放左符号串，参数close存放对应的右符号串。通过比较字符串的index函数返回值来比较左符号和右符号是否属于同一类。请补全下面的代码。

```
#判定栈顶左符号是否与右符号匹配
def matches(open,close):
    opens="<[({"
    closers=">])}"
    #字符串的index相同则左右符号匹配，否则不匹配
    return opens.index(open)==_________________
print(parChecker(checkString))
```

※ 活动3　编程实现

活动2已经完成了符号匹配的算法设计和部分代码实现，请将代码集中输入到一个程序文件中，并检测符号串(()){[]{()}}是否匹配。

人类解决复杂的任务时，通常把任务分解成许多子任务。这样能自顶向下、逐步细化地思考问题，让思路清晰自然，也便于多人合作。在程序设计中也需要自顶向下、逐步细化地把复杂任务分解为子任务，并通常用函数来实现各个任务。

拓展练习

1. 小明暑期在出版社实习时表现突出，不仅提升了符号匹配的效率，而且正确率也有了显著提高。每当出版社工作人员遇到符号匹配问题时，都喜欢请他帮忙。下面几组符号串是出版社叔叔阿姨请小明帮他们解决的符号匹配问题，请协助小明通过编程判断这些符号串是否匹配。

第1组：(((({}}){{}}{(){{{}}}}

第2组：(((({}}))){}(()()()({}){{{}}}}

第3组：(((({}}))){}(()({}){}{{{}}}}))))))))))))

2. 超文本标记语言（HTML）是用来描述网页的一种语言。HTML标签是由尖括号包围的关键词，比如<html>与</html>、<body>与</body>、<h1>与</h1>和 与 等，如图5.2.3所示。

HTML标签通常是成对出现的，比如和，标签对中的第1个标签是开始标签，第2个标签是结束标签。

请编写程序检查图5.2.3中的标签是否能成功匹配。

```
<html>
    <head>
        <title>
            符号匹配问题
        </title>
    </head>

    <body>
        <h1>符号匹配原理</h1>
    </body>
</html>
```

图 5.2.3　HTML源代码示例

5.3 算术表达式求值

表达式求值，就是计算表达式的值。在程序设计中，表达式求值的应用非常普遍，例如，在程序中通过求根公式可以算出一元二次方程的根。怎样编程计算一个字符串表达式的值？计算机利用了栈结构来进行表达式求值。

本节主要学习算术表达式求值的含义及原理，体会栈在算术表达式求值中的应用，掌握算术表达式求值的算法。

学习目标

- ★ 理解算术表达式求值的基本原理。
- ★ 体会栈在算术表达式求值中的应用。
- ★ 初步掌握算术表达式求值的算法。

编写程序时，给出一个复杂的四则运算表达式后，计算机可以轻松计算出结果。这是怎么做到的呢？要知道，+、−、*、/的优先级不同。机器为什么这么聪明呢？

本节围绕“算术表达式求值”项目展开学习，通过项目活动体验计算机算术表达式求值的算法和过程，理解栈与算术表达式求值算法的关系。本节主要包含“体验手动算术表达式求值”和“编程实现算术表达式求值”两个任务。

四则运算由加、减、乘、除四种运算符构成，可以用括号来改变运算顺序。

任务一 体验手动算术表达式求值

※ 活动 四则运算游戏

小明在班级中玩一种四则运算游戏，他让同桌随意写一个四则运算的算式，比如3+2*4−5，然后将算式制成如图5.3.1所示的纸牌。

图5.3.1　算式纸牌

桌面上设置两个区域，一个是数字区，一个是符号区，如表5.3.1所示。游戏的目标是求出算术表达式的值，把手中的牌全部放入数字区或符号区。最初所有牌都拿在手中，从左到右取牌。

放牌：指把牌放入数字区或符号区的最上面。

消牌：指取出数字区最上面的两张牌、运算符区最上面的一张牌，用两个数字做运算符指明的运算，把结果写入空牌并放入数字区。

他用如图5.3.1所示的纸牌演示游戏过程，具体如表5.3.1所示，游戏规则如下。

① 如果取出的牌上是数字，直接把牌放入数字区。

② 如果取出的牌上是乘除号，而符号区也是乘除号，就不断消牌，直到遇到加减号或符号区清空，然后把牌放入符号区。

③ 如果取出的牌上是加减号，就不断消牌，直到符号区清空，然后把牌放入符号区。

④ 当手中没有牌但符号区还有牌时，消牌。

⑤ 当手中没有牌并且符号区也没有牌时，数字区的数值就是表达式的结果。

表 5.3.1　求表达式3+2*4−5的过程

步骤	操作	桌上的牌	手中的牌
1	最开始，牌都在手中，符号区和数字区均没有牌	数字区　符号区	3 + 2 * 4 - 5
2	取到数字“3”，按规则①放入数字区	3　数字区　符号区	+ 2 * 4 - 5
3	取到符号“+”，按规则③放入符号区	3　+　数字区　符号区	2 * 4 - 5
4	取到数字“2”，按规则①放入数字区	2　3　+　数字区　符号区	* 4 - 5

续表

步骤	操作	桌上的牌	手中的牌
5	取到符号“*”，按规则②放入符号区	数字区：2 2 3　符号区：* * +	4 - 5 4
6	取到数字“4”，按规则①放入数字区	数字区：4 4 2 3　符号区：* * +	- 5 -
7	取到符号“–”，按规则③消牌。从数字区取出“4”“2”，从符号区取出“*”，计算2*4，把结果8写入空牌并放入数字区	数字区：8 8 3　符号区：+ +	- 5 -
8	按规则③继续消牌。从数字区取出“8”“3”，从符号区取出“+”，计算3+8，把结果11写入空牌并放入数字区，把符号“–”放入符号区	数字区：11 11　符号区：- -	5 5
9	取到数字“5”，按规则①放入数字区，此时手上没有牌了	数字区：5 5 11　符号区：- -	
10	按规则④消牌。取出“5”“11”，取出“–”，计算11–5，把结果6写入空牌并放入数字区	数字区：6 6　符号区：	
11	按规则⑤得到表达式“3+2*4–5”的结果为6	数字区：6 6　符号区：	

通过以上游戏可以发现，先进入数字区的数后运算，后进入数字区的数先运算；先进入符号区的运算符后计算，后进入符号区的运算符先计算，就像两个栈一样，所以可以利用栈来实现表达式求值。

任务二　编程实现算术表达式求值

※ 活动1　设计数据结构

编写函数expressCalc(s)实现算术表达式求值。任务一的游戏主要涉及符号区、数字区及手上的牌。把符号区和数字区分别用栈opStack和numStack表示，手中的牌用字符串表示。前面已经学习了栈的相关知识，这里直接使用栈对象。

Python 的 eval 函数可以把字符串当作 Python 代码来执行，可以求表达式的值。比如，运行代码 print(eval("3+2"))，可以得到结果 5。

```
01. #计算表达式s的值
02. def expressCalc(s):
03.     opStack=Stack()                    #符号区
04.     numStack=Stack()                   #数字区
```

为了简单起见，这里假设四则表达式的数字都小于10。

※ 活动2　设计算法

参照任务一中的游戏规则，把表达式用字符串s表示，依次取牌就相当于依次取字符串中的字符。放牌时，把牌放到符号区或数字区上面，相当于入栈；而消牌时取出牌做运算，则相当于出栈。算法描述如下。

（1）依次取字符串s中的各个字符，依字符做不同处理。

① 如果字符是数字，把字符压入栈numStack。

② 如果字符是乘除号，当栈opStack的栈顶也是乘除号时，就不断消牌，然后把字符压入栈opStack。

③ 如果字符是加减号，就不断消牌，直到符号区清空，然后把字符压入栈opStack。

（2）当栈opStack不为空时，则消牌，直到栈空。

（3）程序结束时，栈numStack的栈顶数值就是表达式的值。

根据以上算法，请补全下面的代码。

```
05.     for c in s:                                    #取表达式的各个字符
06.         if c=="*" or c=="/":                       #如果是乘号或除号
07.             #如果符号区顶部是乘号或除号
08.             while not opStack.isEmpty() and opStack.peek()=="*" or
09.             opStack.peek()=="/":
10.                 calculate(numStack, opStack)       #消牌
11.             opStack.push(c)                        #入符号区
12.         elif c=="+" or c=="-":                     #如果是加减号
13.             #如果符号区不为空，则不断消牌
```

```
14.       while not opStack.isEmpty():
15.         calculate(numStack, opStack)      #消牌
16.       opStack.push(c)                     #入符号区
17.     else:
18.       ___________                         #入数字区
19.   while not opStack.isEmpty():            #如果符号区不为空
20.     calculate(numStack, opStack)          #消牌
21.   return ____________                     #得到答案
```

在以上代码中，方法calculate(numStack, opStack) 完成消牌操作。消牌的算法为：取出栈numStack的两个数字，取出栈opStack的一个运算符，用两个数字做运算符指明的运算，把结果压入栈numStack。请补全下面的代码。

```
22. #消牌
23. def calculate(nums, ops):
24.   b=float(nums.pop())          #获得第1个操作数
25.   a=float(nums.pop())          #获得第2个操作数
26.   op=ops.pop()                 #获得符号区的操作符
27.   if op=="+":                  #如果操作符是加号，做加法
28.     nums.push(a+b)
29.   if op=="-":                  #如果操作符是减号，做减法
30.     _______________
31.   if op=="*":                  #如果操作符是乘号，做乘法
32.     _______________
33.   if op=="/":                  #如果操作符是除号，做除法
34.     nums.push(a/b)
```

※ 活动3　编程实现

活动2已经完成了对算术表达式求值的算法设计和代码实现，请将代码集中输入到一个程序文件中，并用以下代码进行测试：

```
35. answer=expressCalc("3+2*4-5")
36. print(answer)
```

输入的时候请注意，数字要小于10。

想一想怎样修改程序，让表达式中的数字可以超过10。

拓展练习

1. 进一步完善算术表达式求值的算法，编写程序，计算包含求余运算符的表达式的值，如5%3的结果是2。

提示：%的优先级与*和/一样。

2. 进一步完善算术表达式求值的算法，编写程序，计算包含小括号的四则运算表达式的值，如(3+2)*4-5的结果是15。

注意：只有小括号，没有其他括号，如中括号[]、花括号{}等，括号可以嵌套。

提示：

（1）左括号入栈时可以看作优先级最高，右括号入栈时可以看作优先级最低，不断做栈顶运算直到遇到左括号。

（2）编写代码判断括号是否配对正确，避免出现类似(3+2))*4的表达式。

3. 进一步完善算术表达式求值的算法，编写程序，计算运算数多于一位的四则运算表达式的值，如123+2*4-50的结果是81。

提示：

（1）可以在表达式中添加空格，把运算符和数字分开，用字符串的split方法把表达式的运算符和数字拆分到列表中。

（2）自定义一个read函数，每次从表达式左边读取一个运算符或数字。

单元学习评价

本单元学习了栈结构的基本概念及特征，亲历了利用栈解决问题的一般过程，体验了递归思想在问题解决中的应用。你了解栈的相关概念了吗？你是否能运用栈和递归思想解决实际问题？请参与小组交流并反思，开展自评或小组评价。

一、（多选）以下场景可以应用栈结构的是（　　）。

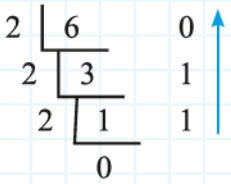
A. 某文件夹中嵌套多个子文件夹，子文件夹中还嵌套多个文件夹，统计该文件夹包含的文件夹数量

B. 食堂服务人员把盘子从底到上摞起来，学生用餐时从上到下依次取出

C. 求迷宫从入口到出口的所有路径是一个经典的程序设计问题。计算机解迷宫时，从入口出发，顺着某一方向向前探索，若能走通，则继续往前走；否则沿原路退回，换一个方向再继续探索，直至探索到所有可能的通路为止

D. 公交车进站时，乘客依次排队上车。排在前面的乘客先上车，后来的乘客排到队尾

二、一个数值可以用不同进制的数字来表示。比如十进制的6，可以用二进制的110来表示，记作$6_{(10)}=110_{(2)}$。对于十进制转二进制，可以用短除法，如右图所示。



编写程序，实现从十进制到二进制的转换，填写下面的设计方案。

1. 问题分析。

在做短除法时，余数出现的次序和在结果中的位置有什么关系？

2. 设计数据结构。

（1）采用什么数据结构保存数据？

（2）为什么采用这种数据结构？

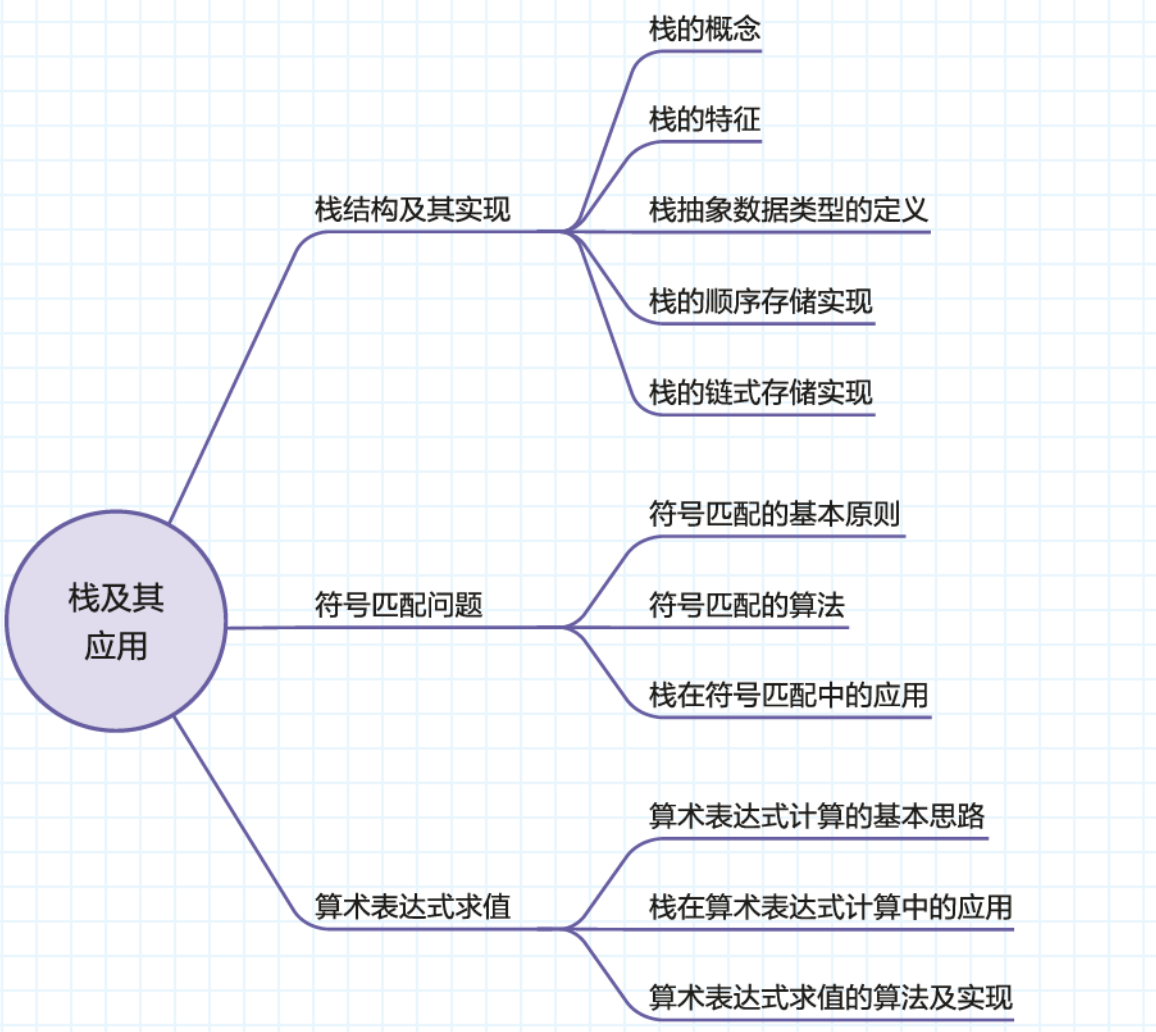
3. 设计算法。

（1）怎样输入、输出数据？

（2）画出算法流程图。

4. 编程实现。

单元学习总结



第 6 单元　树及其应用

树结构是不同于线性表、队列和栈的一种“非线性”结构，它是对自然界中“树”的结构仿生。在生物学中对物种进行分类，国家治理中划分行政管理区域，甚至在读心术游戏中，都会用到树结构。当然，信息技术中的很多问题，也是通过各种各样的树结构来解决的。只要你愿意，还可以尝试用树结构来创作计算机艺术作品。

本单元通过“无处不在的树”项目对现实生活中各种树形层次结构的提炼，理解树结构的抽象定义，通过“树的递归处理”和“树的视觉艺术”项目，体验并亲历利用树结构和递归算法解决问题的一般过程和方法，促进学科思维的形成。

6.1　树结构及其实现

正如其名称“树”所提示的那样，树结构跟自然界中的树有着千丝万缕的联系。人们已经将树结构深深植入到知识体系、思维方式和行为模式中。在我们日常的学习和生活中，不仅是低头不见抬头见的成荫绿树，处处都有分层、分支的树结构在发挥作用。

本节通过对几个现实生活问题的介绍和分析，引入树结构的基本概念和特征，并给出树抽象数据类型的定义，最后通过顺序存储和链式存储两种方式来实现二叉树抽象数据类型。

学习目标

- ★ 理解树结构的概念和特征。
- ★ 认识各种现实生活问题中蕴含的树结构。
- ★ 掌握二叉树抽象数据类型的定义。
- ★ 掌握二叉树的两种实现方法。

本节围绕“无处不在的树”项目展开学习，通过项目活动来学习什么是树结构，如何用顺序存储和链式存储方式实现树结构。本节主要包含“生活中的树”和“算术表达式树的实现”两个任务。

任务一　生活中的树

※ 活动 1　表示生物的分类

博物学家们走遍世界的每个角落，探索各地的新奇动植物，带回成千上万箱标本，在实验室里仔细对比它们的外形和内部结构，根据它们的异同分门别类。

我们在生物课上就是按照这个分类体系来学习的。生物学分类法

将地球上的所有生物都归类到域、界、门、纲、目、科、属、种8个层次中，如图6.1.1所示。

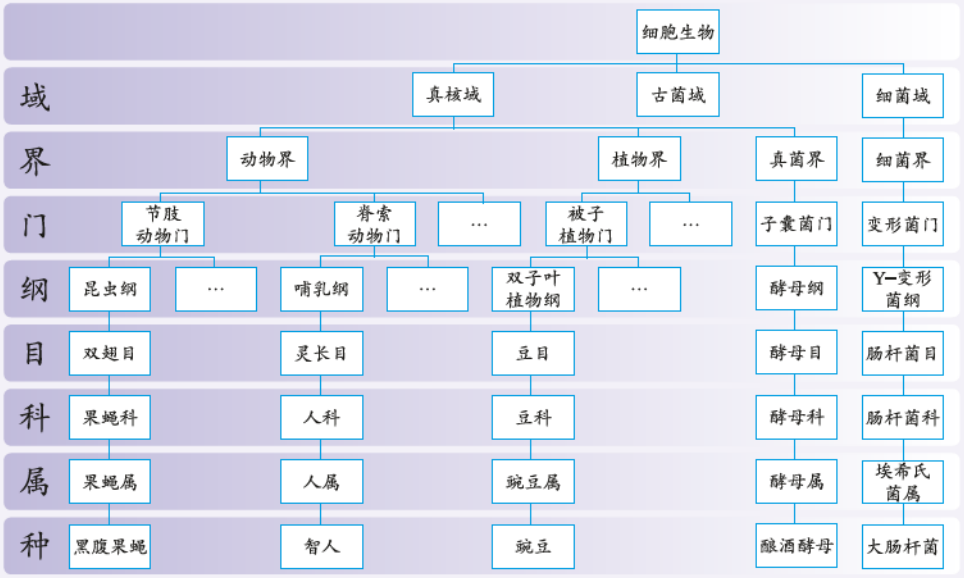


图6.1.1 生物分类体系（部分）

某些少数生物分类具有模糊或者多重性，不符合树的特征。

这样的生物分类体系是不是正像一棵有主干、分枝和树叶的大树？只不过方向是倒立的，根和主干在上，分枝和树叶朝下。

请根据图6.1.1填写表6.1.1，并在图6.1.1中将每种生物所属的各层次类别用线圈起来。

表6.1.1 生物所属类别层次表

物种	域	界	门	纲	目	科	属	种
黑腹果蝇								
智人								
豌豆								
酿酒酵母								
大肠杆菌								

根据你的观察和理解，回答以下问题。

1. 具体的生物会出现在分类体系的哪个位置？中间还是末端？
2. 上层的大类和下层的小类之间是什么关系？
3. 同一大类下面的几个小类之间是什么关系？
4. 同一个生物会属于不同种类吗？

● 树结构的基本概念

树结构是对自然界中树的一种抽象和结构仿生。树结构中的分叉部分称为“节点（Node）”，节点之间的连线称为“边（Edge）”。在对树结构进行图示的时候，其方向正好与自然界中的树相反，对应树根的“根节点（Root Node）”画在上方，对应枝干和树叶的“内部节点（Internal Node）”和“叶节点（Leaf Node）”画在中间和下方，再用代表边的线连接起来，如图 6.1.2 所示。

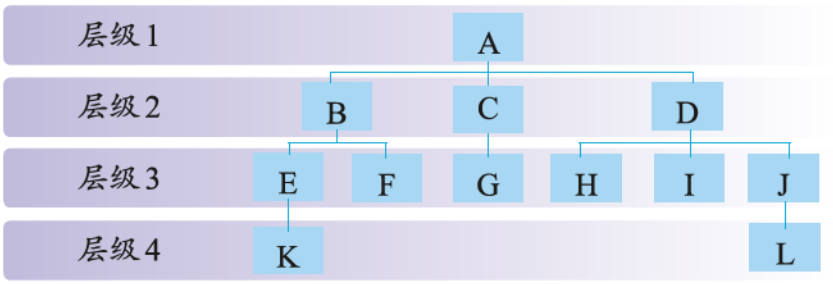


图 6.1.2　树结构示意图

在图 6.1.2 中，（A）节点就是根节点，（B）（C）（D）（E）（J）节点是内部节点，（F）（G）（H）（I）（K）（L）节点是叶节点。

我们将根节点或者内部节点连接的下方节点称为这个节点的“子节点（Child Node）”，相应地，其本身被称作“父节点（Parent Node）”，具有同一个父节点的子节点之间互称“兄弟节点（Sibling Node）”。例如：（A）节点是（B）（C）（D）节点的父节点；（E）（F）节点是（B）的子节点；（H）（I）（J）节点是兄弟节点，有共同的父节点（D）。

子节点通常也是一棵树的树根，我们把以子节点为树根的这棵树称为“子树（Subtree）”。例如：以（B）为根节点的树（B）（E）（F）（K），就是节点（A）的一棵子树。节点（A）一共有 3 个子节点，相应地，它也有 3 棵子树。

在树结构中，每个节点到根节点之间都有唯一的一条路径（Path），如（E）节点到根的路径为（E—B—A）；（L）节点到根的路径为（L—J—D—A）。

按照树结构中每个节点到根节点的路径长度，我们将树结构分为若干层（Level）。例如：（D）节点位于第 2 层，（L）节点位于第 4 层。一棵树的最大层数称为树的深度（Depth）。图 6.1.2 所示的树的深度为 4。整棵树只有一个根节点的话，树的深度为 1。

※ 活动 2　表示行政区划

人类在长期的历史发展中，建立过各种社会结构，从最原始的几

十人的部落，到拥有亿万人民和宽广疆域的国家。目前，联合国会员国有193个，全部都是主权独立的国家。世界上的国家一般都按照树结构来层层划分自己的疆域，组织成分级别的行政管理区域，称为行政区划。

中国的行政区划分为全国、省级、地级、县级、乡级等5级。我们可以将中国的行政区划结构表示成这样一张图，如图6.1.3所示。

人类社会中，许多组织的管理模式都是层级的树状结构，如政府部门、公司、学校等。

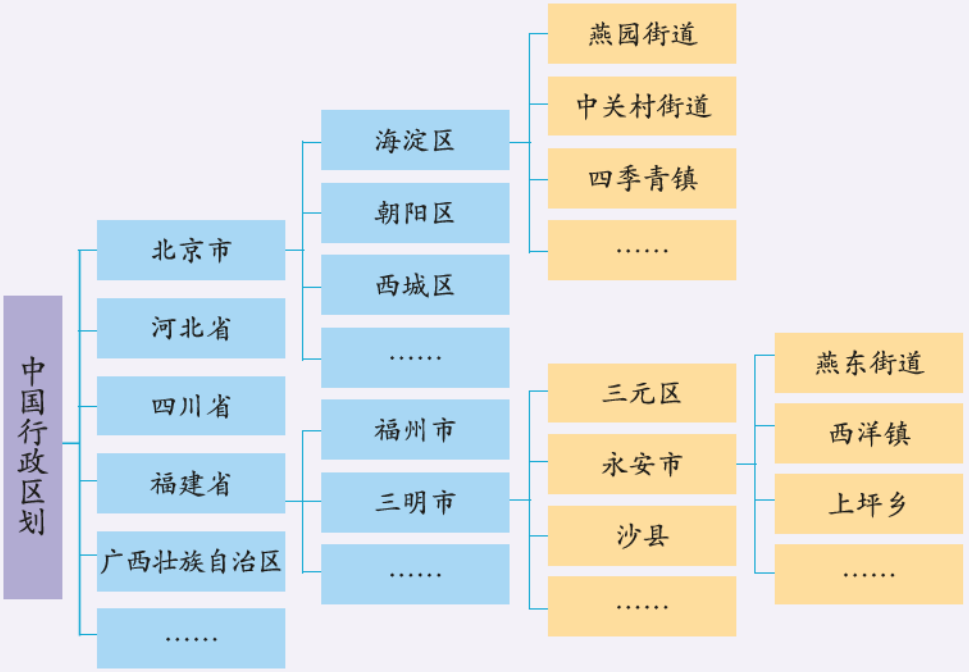


图6.1.3 中国行政区划树（部分）

可以看到，中国的行政区划层级也是一个树结构。为了图形布局方便，我们也可以把根节点画在左边，从左到右画上内部节点和叶节点。

在中国行政区划树上，树根是全国，区域范围最大，作为一级行政区的省级行政区包括省、自治区、直辖市、特别行政区4类，目前全国共划分为34个一级行政区，即23个省、5个自治区、4个直辖市和2个特别行政区。

请根据图6.1.3填写表6.1.2，并在图6.1.3中将每个行政区的直接组成部分用线圈起来。

表6.1.2 行政区直接组成部分

行政区	直接组成部分
北京市	
海淀区	
永安市	

根据你的观察和理解，思考并回答以下问题。

1. 上层行政区和下层行政区之间是什么关系？
2. 同一个行政区下的几个行政区之间是什么关系？
3. 一个行政区会隶属于不同的上层行政区吗？

● 树结构与数据关系

在树结构中，除根节点外，其余节点有且仅有一个父节点，而可以有多个子节点。我们可以用树结构来表示两种常见的数据关系：“一般—特殊”关系和“整体—部分”关系。在“一般—特殊”关系中，父节点表示具有一般性的大类，而大类下的特殊细分小类则是多个子节点，每个小类只属于一个大类。在“整体—部分”关系中，父节点表示事物的整体，而整体的每个部分则是多个子节点，每个部分只归属于一个整体。

当然，我们还会遇到一些更为复杂的数据关系。有些分类体系中可能存在不同的分类标准。例如，对汽车进行分类，电动轿车就会归属于按能源分类的“电动车”类和按运载对象分类的“客车”类。也就是说，一个小类可能会归属于多个大类。这样的数据关系就无法用树结构来表示了，而是需要更为复杂更为灵活的网状结构。

任务二　算术表达式树的实现

※ 活动 1　认识算术表达式树

前面我们已经用栈结构解决了算术表达式的求值问题，下面我们用树结构来解决。我们知道，算术表达式由运算数和运算符相间组成，如 1+2*3。有时候还需要括号来指定运算的优先级，如 4*(2+3)。关于如何将算术表达式表示为树结构，我们先来看一个最简单的算术表达式：3-1。

显然，运算符是“-”，而两个运算数是3和1，其中 3 是左运算数，1 是右运算数。用树结构可以表达为如图 6.1.4 所示的结构。

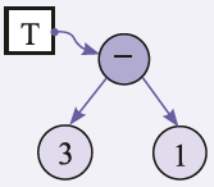


图 6.1.4　“3-1”的树结构

要创建这样一个树结构，步骤如下：

（1）创建一棵树T，树根是“-”；

（2）为T插入左边的子节点，内容是“3”；

（3）为T插入右边的子节点，内容是“1”。

由于运算符要连接两个运算数，而运算数之间不会直接连接，所以我们用内部节点来表示运算符，叶节点表示运算数。显然，我们可以把计算的结果标记在运算符节点中。上述的树中，根节点“-”就可以加一个标记“2”，表示“3-1”的运算结果。

按照这样的思路，我们就可以处理更复杂一些的算术表达式了，如5 +(3-1)。

因为“+”的右运算数是一个减法表达式，当然就是一棵子树

了，下面我们把表达式的值标注在树和子树的根节点上，如图6.1.5所示。

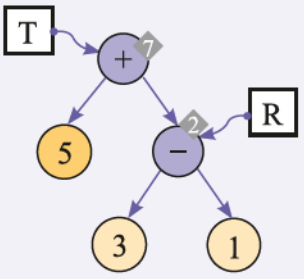


图6.1.5 “5+（3−1）”的树结构

按照图6.1.5所示补全以下创建树的步骤：

（1）创建一棵树T，树根是___；

（2）为T插入左子节点，内容是___；

（3）为T插入右子节点，内容是___，右子节点记为R；

（4）为R插入_____，内容是___；

（5）为R插入_____，内容是___。

用来表示表达式的树结构称为“表达式树”，由于每个节点只有左、右两个子节点，所以也称作“二叉树”。请注意运算的次序，画出下列表达式的树结构，并写出这些表达式树的创建步骤：

（1）(3+4)*(9−6)；

（2）1+3*4−9/3。

树抽象数据类型

为了在程序中使用树结构来解决问题，我们需要定义树抽象数据类型。实际应用中的树结构多种多样，按照子节点最大数目来分，常用的有二叉树、四叉树、八叉树、多叉树等，其中最简单的是仅有左、右子节点的二叉树（Binary Tree）。二叉树被广泛应用在表达式处理、排

序、二进制编码等问题求解上。

我们为二叉树抽象数据类型（ADT BinaryTree）定义了如下接口。

ADT BinaryTree：

- BinaryTree(val)：创建一棵二叉树，树根内容为 val。
- getLeftChild()：返回左子节点为根的左子树。
- getRightChild()：返回右子节点为根的右子树。
- setRootValue(val)：将二叉树树根的内容设置为 val。
- getRootValue()：返回二叉树树根的内容。
- insertLeft(val)：给二叉树插入左子树，其树根内容为 val，并返回左子树。
- insertRight(val)：给二叉树插入右子树，其树根内容为 val，并返回右子树。

※ 活动 2　用抽象数据类型创建表达式树

有了二叉树抽象数据类型，我们就可以把前面的表达式树的创建过程写成程序了。例如，“3-1”对应的程序如下。

```
01. T=BinaryTree("-")
02. T.insertLeft(3)
03. T.insertRight(1)
```

请自行写出创建下列 3 个表达式树的程序：

```
5+(3-1);
(3+4)*(9-6);
1+3*4-(9/3)。
```

利用教科书配套资源，可以用二叉树的 plot 接口把构建好的二叉树画在屏幕上。调用步骤如下：

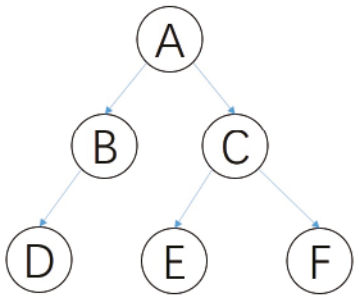
（1）导入二叉树模块：from btree import BinaryTree；

（2）插入上述调用 ADT BinaryTree 接口的程序代码；

（3）添加语句 T.plot()。

● 二叉树的顺序存储实现

可以利用 Python 的内置数据类型列表实现二叉树抽象数据类型。我们约定，表示二叉树的列表恰好包含 3 个元素，第 1 项元素是树根内容，第 2、第 3 项元素分别是左子树和右子树，如果没有左、右子树，则设置为空列表。实际上，左、右子树也是二叉树，所以这个列表将会是一个嵌套列表。图 6.1.6 所示是一棵二叉树及其对应的顺序存储实现。



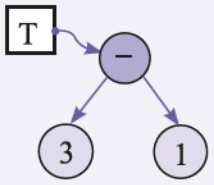

```
T = ['A',  #树根
       ['B',   #左子树
         ['D', [ ] ,[ ]],
         [ ]
       ],
       ['C',   #右子树
         ['E', [ ], [ ]],
         ['F', [ ], [ ]]
       ]
     ]
```

图 6.1.6　一棵二叉树及其对应的顺序存储实现

这样，对于一棵顺序存储实现的二叉树 T 而言，T[0]就是树根，而 T[1]和 T[2]则分别是左、右子树。如果 T 的左、右子树不为空的话，还可以继续引用它们的子树。例如，T[1][0]是B，T[1][1]则是B的左子树。

※ 活动3　表达式树的顺序存储实现

如果二叉树抽象数据类型用顺序存储来实现，表达式树看起来会是什么样的？例如，表达式“3−1”表示为一个3元素嵌套列表后如图6.1.7所示。




```
T=[ '−',
      [ '3', [], [] ],
      [ '1', [], [] ]
  ]
```

图 6.1.7　表达式树及其顺序存储实现

请继续写出下列3个表达式的表达式树及其列表实现：

5+(3−1)；

(3+4)*(9−6)；

1+3*4−(9/3)。

利用教科书配套资源，可以用二叉树的字符串表示验证答案。调用方法如下：

（1）导入二叉树的顺序存储实现：from btreelist import BinaryTree；

（2）插入上述调用ADT BinaryTree接口的程序代码；

（3）添加语句 print（T）。

我们把二叉树的顺序存储表示用Python的类来编程实现。首先是用类的构造函数__init__()来实现ADT BinaryTree中的创建二叉树接口BinaryTree(val)。

```
01. class BinaryTree:
02.   def __init__(self, val_or_list):
03.     if type(val_or_list) is list:
04.       self.items=val_or_list
05.     else:
06.       self.items=[val_or_list, [],[]]
```

我们在类中设置一个属性items来保存这个3元素嵌套列表。在创建二叉树的时候，第1个元素保存树根，第2、第3个元素则都是空列表。接下来，我们就可以通过其他的类方法来实现ADT BinaryTree的其他接口了。insertLeft的接口实现如下：

```
07.   def insertLeft(self, val):
08.     oldLeft=self.items.pop(1)              #取到现有的左子树
09.     #如果现有左子树不为空,把它作为新左子节点的左子树
10.     if len(oldLeft)>1:
11.       self.items.insert(1, [val, oldLeft, []])
12.     #如果现有左子树为空,则插入新的左子节点作为左子树
13.     else:
14.       self.items.insert(1, [val, [], []])
15.     return BinaryTree(self.items[1])     #返回刚插入的左子树
```

insertRight的接口实现类似，可以参见教科书配套资源。getLeftChild是获取二叉树的左子树，其代码实现非常简单，只要返回第2个元素即可。

```
16.   def getLeftChild(self):
17.     BinaryTree(return self.items[1])
```

getRightChild的接口实现也只需要返回第3个元素即可，在此不再赘述。getRootValue和setRootValue这一对接口涉及树根的操作，与列表的第1个元素相关，其实现代码如下：

```
18.   def getRootValue(self):
19.     return self.items[0]
20.   def setRootValue(self, val):
21.     self.items[0]=val
```

※ 活动4　讨论二叉树的顺序存储实现

列表可以说是Python中应用最广泛的内置数据类型，列表类型的功能强大，其包含的元素可以是任意类型的数据项，包括列表类型。

这样使得列表不只能用来实现队列和栈这样的线性数据结构，也可以通过嵌套列表的方式来实现树这样的层次结构。

【讨论】你觉得二叉树的顺序存储实现有哪些优势？又有哪些不足之处？除了二叉树之外，列表可以用来实现四叉树、八叉树甚至多叉树吗？

● 二叉树的链式存储实现

实现二叉树的另一种更为通用的方式是基于节点引用的链式存储方式，许多其他编程语言如C语言可以通过指针等方式来实现链式存储，所以可以采用同样的方法来实现二叉树。在链式存储实现中，每个二叉树节点对应一个节点对象，每个节点对象具有两个属性，分别引用本节点的左、右子树，如果没有左、右子树，则引用设置为None。图6.1.8所示是一棵二叉树及其对应的链式存储实现。

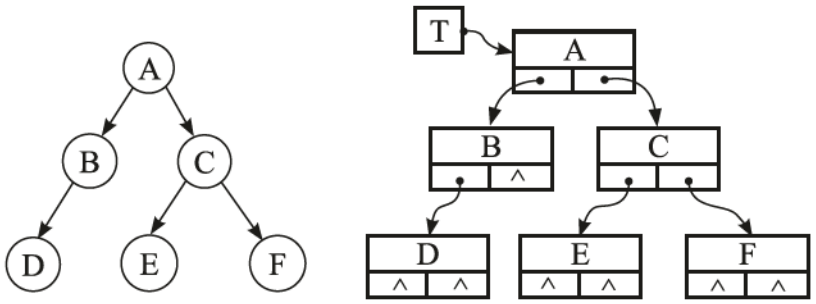


图6.1.8 一棵二叉树及其对应的链式存储实现

这样，对于一棵链式存储实现的二叉树T而言，T指向根节点A，而T.left和T.right则分别指向左、右子树。如果T的左、右子树不是空的话，还可以继续指向它们的子树。例如T.left的树根是‘B’，T.left.left则是B的左子树。

※ 活动5 表达式树的链式存储实现

如果二叉树抽象数据类型用链式存储来实现，表达式树看起来会是什么样的？例如，表达式“3-1”表示为一个链式存储结构后如图6.1.9所示。

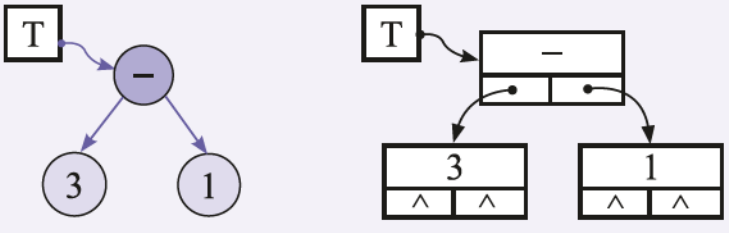


图6.1.9 表达式树及其链式存储实现

请继续画出下列 3 个表达式的表达式树及其链式存储实现：

5+(3-1)；
(3+4)*(9-6)；
1+3*4-(9/3)。

利用教科书配套资源，可以用二叉树的可视化来验证答案。调用方法如下：

（1）导入二叉树的链式存储实现：from pythonds.btreelink import BinaryTree；

（2）插入上述调用 ADT BinaryTree 接口的程序代码；

（3）添加语句 T.plot()。

我们把二叉树的链式存储也用 Python 的类来编程实现。首先是用类的构造函数__init__()来实现 ADT BinaryTree 中的创建二叉树接口 BinaryTree(val)。

```
01. class BinaryTree():
02.   def __init__(self, val):
03.     self.value=val
04.     self.leftChild=self.rightChild=None
```

我们在类中设置属性 value 来保存节点的值，属性 left 和 right 则分别引用左、右子树。在创建二叉树的时候，属性 value 保存树根，属性 left 和 right 则都是 None。接下来，我们就可以通过其他的类方法来实现 ADT BinaryTree 的其他接口了。insertLeft 的接口实现如下。

```
05.   def insertLeft(self, val):
06.     #如果现有左子树不为空,把它作为新左子节点的左子树
07.     if self.leftChild is not None:
08.       newLeft=BinaryTree(val)
09.       newLeft.leftChild=self.leftChild      #注意赋值顺序
10.       self.leftChild=newLeft
11.     #如果现有左子树为空,则插入新的左子节点作为左子树
12.     else:
13.       self.leftChild=BinaryTree(val)
14.     return self.leftChild                   #返回刚插入的左子树
```

insertRight的接口实现与insertLeft的实现过程类似，可以参见教科书配套资源。getLeftChild是获取二叉树的左子树，其代码实现非常简单，只要返回leftChild即可。

```
15.    def getLeftChild(self):
16.      return self.leftChild
```

getRightChild的接口实现也只需要返回rightChild即可，不再赘述。getRootValue和setRootValue这一对接口涉及树根的操作，与属性value相关，其实现代码如下：

```
17.    def getRootValue(self):
18.      return self.value
19.    def setRootValue(self, val):
20.      self.value=val
```

※ 活动6　讨论二叉树的链式存储实现

链式存储可以说是实现各种数据结构的利器，线性结构、树形结构和更复杂的图状结构，用它都能够实现。链式存储实现通用性强，效率较高，但操作过程较为复杂，尤其需要注意链式引用指向操作的次序，如果疏忽大意，很容易导致程序错误。

【讨论】你觉得二叉树的链式存储实现相对于顺序存储实现来说有哪些优势？又有哪些不足之处？除了二叉树之外，链式存储可以用来实现四叉树、八叉树甚至多叉树吗？

拓展练习

在算术表达式中，除了常见的“+”“−”“*”“/”这样的“双目运算符”之外，还有一些“单目运算符”，如求负“−(2+3)*5”和开根号“$\sqrt{2}$+1”等。请思考如何在表达式树中表示这些单目运算符。

6.2　用二叉树排序

二叉树是进一步学习高级数据结构和算法的基础，排序和查找是数据处理中一类最基本的操作。如果我们需要对一组数据排序，以便进行快速查找，可以用二叉树来组织这组数据。

本节重点学习二叉排序树的基本原理、算法及其实现，以及二叉树的结构和递归思想在其中的应用。

学习目标

- ★ 理解二叉排序树的基本原理。
- ★ 体会二叉排序树在数据查找中的应用。
- ★ 初步掌握二叉树排序算法。
- ★ 体会递归思想在算法设计中的运用。

本节围绕“树的递归处理”项目展开学习，通过项目活动来理解什么是二叉排序树，如何用二叉树结构实现排序。本节主要包含“通过实例认识二叉排序树”和“利用二叉树结构实现排序”两个任务。

任务一　通过实例认识二叉排序树

※ 活动 1　用二叉树组织数据

请将如图 6.2.1 所示的数据依照从左到右的次序，按照“左子节点<父节点<=右子节点”的规则组织成一棵二叉树。

70	31	93	94	14	23	73	71	96

图 6.2.1　一个数据集

组织过程如下。

1. 首先是数据70，由于二叉树里没有节点，它自然成为唯一的根节点。

2. 其次是数据31，我们从根节点70开始对比，由于31<70，所以31应该是70的左子节点。

3. 再次是数据93，我们还从根节点70开始对比，由于70<93，所以93应该是70的右子节点。

4. 最后是数据94，我们还从根节点70开始对比，由于70<94，所以94应该是70的右子节点，但这已被93占据，我们继续将94与93对比，由于93<94，所以94应该是93的右子节点。

这样，我们就将4个数据组织成了一棵二叉树，如图6.2.2所示。

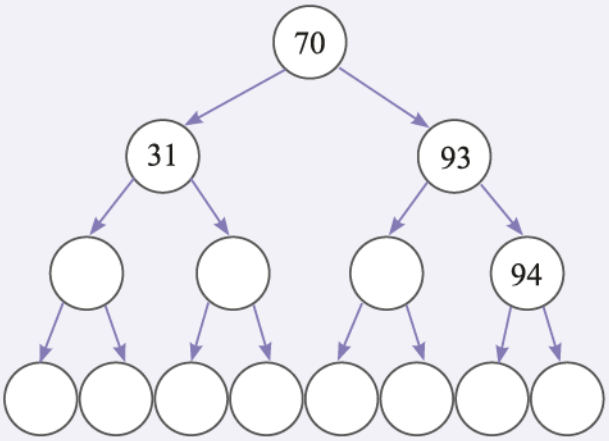


图6.2.2 将数据集组织成二叉树

请按照“左子节点<父节点<=右子节点”的规则，将剩下的5个数据依次填进二叉树模板中。填写完毕后，模板中还会留下6个空格，请忽略这些空格。

● 二叉排序树

我们知道，二叉树中的每个节点最多可能有左、右两个子节点，而二叉树的层次是没有限制的。这样，我们按照“左子节点 < 父节点 <= 右子节点”的顺序，将任意数量的数据组织成一棵二叉树，再有序输出二叉树的每个节点，即可实现数据排序。根据这样的大小次序规则组织成的二叉树，就叫作二叉排序树。

※ 活动2 体验相同的数据，不同的树

我们发现，在组织二叉排序树的过程中，数据集里的第一个数据总是成为根节点，对于相同的一组数据，由于插入的顺序不同，会组织成不同的二叉排序树。

同样是活动 1 中的 9 个数据，我们将 70 和 73 的位置对调一下，如图 6.2.3 所示。

73	31	93	94	14	23	70	71	96

图 6.2.3　对调了部分数据的数据集

将从 73 开始的数据依次插入节点，会重新生成二叉排序树。请将结果填写在如图 6.2.4 所示的模板中。

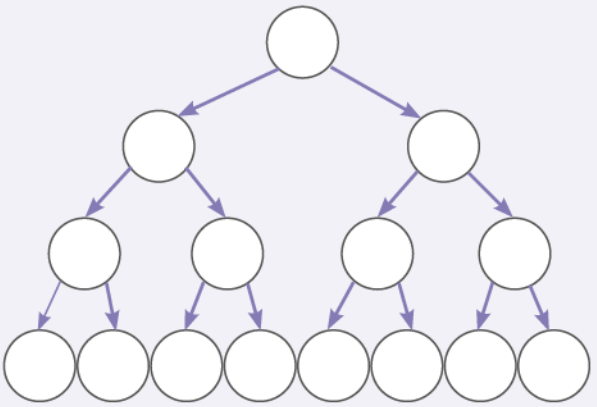


图 6.2.4　重新组织的二叉排序树

请讨论以下问题：

1. 是否只要插入数据的次序有所不同，就会生成不同的二叉排序树？

2. 对于一个包含 n 个数据的数据集，可以生成多少种不同的二叉排序树？

● 二叉排序树的高度

同一个数据集，由于插入数据节点的次序不同，能够生成差异很大的二叉排序树。如所有节点都仅有左子节点，或者仅有右子节点，这样会使二叉排序树的高度等于数据的个数。而所有节点中，如果最多一个节点仅有左子节点，其余的内部节点都拥有左、右子节点，会使二叉排序树的高度最小化。

※ 活动 3　输出排序结果

组织生成二叉排序树是实现排序的第一步，接着还要输出所有二叉树节点，才算最终完成数据排序。由于二叉树不是线性结构，无法按照前驱后继的次序来输出节点，为了保持“左子节点<父节点<=右子节点”的节点顺序，我们输出节点时，也要按照“左子树所有节点→父节点→右子树所有节点”的次序进行。

例如，图 6.2.5 所示的二叉排序树的输出为“13，45，56”；而图 6.2.6 所示的二叉排序树，由于左子树不仅仅是一个节点，我们将左子树也按照“左子树所有节点→父节点→右子树所有节点”的次序来输出。

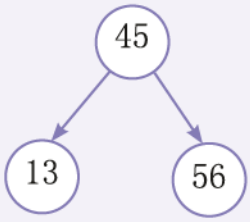


图 6.2.5　二叉排序树示例

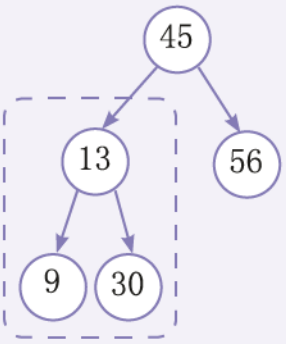


图 6.2.6　子树的输出

这样，左子树输出为“9，13，30”，接着输出父节点 45，再输出右子树所有节点 56。最后，总输出结果为“9，13，30，45，56”。

接下来，观察图 6.2.7 所示的二叉排序树，注意区分左子树、父节点和右子树。

直接在图中用线标示出左、右子树，并按照“左子树所有节点→父节点→右子树所有节点”的次序来输出所有数据，把结果填写在下面。

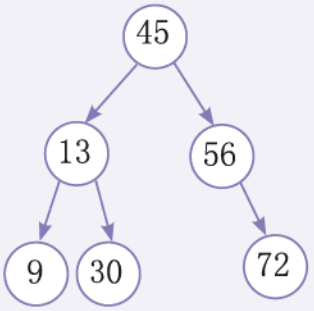


图 6.2.7　二叉排序树示例

将活动 1 中组织好的二叉排序树，按照同样的方式，用线标示出每一个左、右子树，并将所有节点数据输出，把结果填写在下面。

最后，将活动 2 中组织好的二叉排序树，也按照同样的方式，用线标示出每一个左、右子树，并将所有节点数据输出，把结果填写在下面。

对比上面两个输出结果，它们是否相同？为什么不同的二叉排序树会输出相同的结果？

● 有序输出二叉排序树的节点

二叉排序树按照“左子树所有节点→父节点→右子树所有节点”的次序来输出所有节点数据，就能得到排好序的数据集。只要是同一组数据生成的二叉排序树，无论其结构有多大差异，都输出相同的有序数据集。

● 二叉排序树用于数据查找

除了输出有序数据集之外，二叉排序树还广泛应用于数据查找。如果我们需要在一组数据中查找是否存在某个数据，除了从第一个数据开始顺序比对之外，还可以将数据组织为二叉排序树，从树根开始比对。

查找过程如下：

（1）如果当前节点数据等于要查找的数据，则查找成功，结束查找；

（2）如果要查找的数据比当前节点数据小，则进入左子节点比对，如果当前节点没有左子节点，则查找失败，结束查找；

（3）如果要查找的数据比当前节点数据大，则进入右子节点比对，如果当前节点没有右子节点，则查找失败，结束查找；

（4）重复步骤（1）~（3），直到查找成功或者失败，结束查找。

查找示例如图 6.2.8 所示。

可以看出，利用二叉排序树进行数据查找，可以比顺序查找方法大大减少比对的次数。数据量越大，需要查找的数据越多时，利用二叉排序树查找数据的优势就越明显。

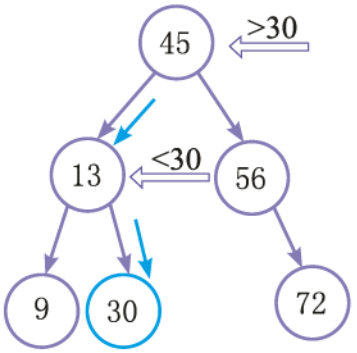


图 6.2.8　二叉排序树用于数据查找

任务二　利用二叉树结构实现排序

※ 活动 1　建立数据结构

我们来实现一个用二叉树对数据集进行排序的函数 BinaryTreeSort(lst)，函数参数 lst 是原始数据列表，函数返回排好序的数据列表。

我们需要构建一棵二叉树，首先将列表中的第一个数据作为二叉树的首个节点——根节点。请补全下面的代码。

```
01. def BinaryTreeSort(lst):
02.    btree=BinaryTree(_______)
```

利用二叉树结构对一组数据进行排序，需要一棵二叉树来容纳数据集里的所有数据，并通过左、右子树来体现数据之间的大小关系，即左子树所有节点的数据都小于父节点的数据，而父节点的数据小于右子树所有节点的数据。将数据集中的数据加入二叉排序树的过程中，二叉排序树应始终保持这个性质。

※ 活动2 设计算法

如前所述，利用二叉树结构实现排序的算法分为组织二叉排序树和输出所有节点数据两个步骤。其算法描述如下。

（1）将列表lst中的每个数据项item插入二叉排序树btree中。

（2）将二叉排序树btree中的节点数据按规则加入列表result。

（3）函数返回result。

根据上述算法，补全下面的代码。

```
03. for item in lst[____]:      #从第2个数据项开始
04.   #调用函数，将item插入btree
05.   BSTInsertItem(btree, item)
06. #调用函数，输出btree所有节点
07. result=BSTOutput(btree)
08. return ________
```

接下来，我们对算法中调用的BSTInsertItem和BSTOutput两个函数进一步细化，确定其算法。

函数BSTInsertItem(btree, item)将数据项item插入二叉排序树btree中，其算法描述如下。

（1）将item与二叉排序树的根节点数据进行比对。

（2）如果item小于根节点数据，则如果左子树存在，将item插入左子树；如果左子树不存在，将item作为新左子树插入。

（3）如果item不小于根节点数据，则如果右子树存在，将item插入右子树；如果右子树不存在，将item作为新右子树插入。

根据上述BSTInsertItem函数的算法，补全下面的代码。

```
09. def BSTInsertItem(btree, item):
10.   val=btree.getRootValue()                #val是根节点
11.   if item<val:                            #item小于根节点数据
12.     lbtree=btree.getLeftChild()           #lbtree是左子树
13.     if lbtree is None:                    #没有左子树
14.       btree.insertLeftChild(item)
```

```
15.     else:
16.       BSTInsertItem(lbtree, item)
17.   else:                                  #item不小于根节点数据
18.     rbtree=btree._____________
19.     _____________________________
20.     _____________________________
21.     _____________________________
22.     _____________________________
```

函数BSTOutput(btree)将二叉排序树btree中所有节点数据输出为列表并作为结果返回，其算法描述如下。

（1）如果btree有左子树，则输出左子树所有节点数据，添加到result列表中。

（2）将btree根节点数据添加到result列表中。

（3）如果btree有右子树，则输出右子树所有节点数据，添加到result列表中。

（4）返回result列表。

根据上述BSTOutput函数的算法，补全下面的代码。

```
23. def BSTOutput(btree):
24.   result=[]
25.   lbtree=btree.getLeftChild()
26.   if lbtree is not None:
27.     result=result+BSTOutput(lbtree)
28.   result.append(btree.getRootValue())
29.   rbtree=btree._____________
30.   ____________________________
31.   ____________________________
32.   return result
```

● 递归算法应用

由于二叉排序树可以分为根节点和左、右子树，而无论是左子树还是右子树，仍然是二叉排序树，所以对二叉排序树进行数据插入和数据输出操作，用递归来实现算法是最为简单和清晰的。

将新的数据插入二叉排序树，首先从根节点开始比对，如果新数据比根节点数据小，则表示它应该被插入到左子树中。这样，如果左子树为空，那么新数据就成为左子树；如果不为空，则将新数据插入左子树，具体算法就可以直接以左子树作为参数，递归调用新数据插

入二叉排序树函数。如果新数据比根节点数据大，则表示它应该被插入到右子树中，此时可以通过同样的思路来设计算法。

要将二叉排序树节点数据有序输出，只需先有序输出左子树所有节点，再输出根节点，最后输出右子树所有节点。这样，如果左、右子树为空，无须做任何处理，否则都可以用左、右子树作为参数，递归调用二叉排序树节点数据有序输出函数。

可以看出，递归调用的使用能够使算法程序简洁易读。当然，有时候递归算法在运行效率上会出现不可接受的劣化，这时，就需要针对具体情况，采用一些改善递归调用性能的特殊方法，或者用非递归算法来解决问题。

※ 活动3　编程实现

活动2已经完成了利用二叉树结构排序的算法设计和代码实现，请将代码集中输入到一个程序文件中，用如下数据对程序进行测试。

```
33.   ret=BinaryTreeSort([45, 12, 33, 87, 99, 23, 69, 7, 1])
34.   print(ret)
```

利用教科书配套资源，可以用二叉树可视化来验证答案。二叉树可视化的调用方法如下：

（1）导入二叉树的链式存储实现：from pythonds.btreelink import BinaryTree；

（2）在BSTInsertItem函数调用完成之后、BSTOutput函数调用之前，添加语句btree.plot()，运行程序即可看到建立好的二叉排序树图片。

拓展练习

1. 探究将数据插入二叉排序树的过程中，什么样的插入次序会导致二叉排序树的高度最大？什么样的插入次序可以使二叉排序树的高度最小？

2. 对于同一个数据集，不同的数据插入次序会导致组织二叉排序树的效率出现差异吗？分析效率最高和效率最低的情况。

3. 设计一种二叉树排序的非递归算法。

6.3　画出二叉树

可视化是认识数据的直观途径，能帮助我们更加深入地理解在计算机中看不见摸不着的数据结构及其操作。同时，数据可视化也是计算机视觉艺术的灵感来源。利用看似枯燥的程序代码，无须画笔就能够绘制出令人惊叹的绚丽图形，给人以精神享受。

本节利用二叉树可视化对自然界的植物进行模拟仿真，通过设置和调整一系列绘图参数，绘制出丰富多变的图案，展现一种数字化艺术的实现方法。

学习目标

- ★ 理解二叉树可视化的基本原理。
- ★ 体会二叉树绘制参数对可视化效果的控制。
- ★ 掌握二叉树绘制算法。
- ★ 体会和欣赏计算机视觉艺术形式。

大自然是人们获取灵感和美的源泉，作为植物整体结构姿态的典型仿生，树结构的可视化能够展现自然与理性结合之美。本节围绕“树的视觉艺术”项目展开学习，通过项目活动来对二叉树进行可视化，并展开想象，通过修改可视化参数，绘制出千变万化的二叉树艺术图形。本节主要包含“绘制简单的二叉树”和“绘制多彩的二叉树”两个任务。

任务一　绘制简单的二叉树

※ 活动 1　海龟作图入门

Python 语言的发行版本内置了一个作图模块 turtle，可以通过模拟海龟在沙滩上爬行的动作，绘制出计算机图形。turtle 模块的使用非常

简洁，可以用很少量的代码画出精美的图形。

运行下面的程序可以画出一个边长为100的等边三角形。

```
01. import turtle                    #导入turtle模块
02. p=turtle.Pen()                   #创建一支画笔（海龟）
03. p.pencolor('blue')               #设置画笔的颜色为蓝色
04. p.pensize(5)                     #设置画笔的粗细为5
05. #最初画笔（海龟）朝向正右方，向前画长度为100的直线
06. p.forward(100)
07. p.left(120)                      #画笔（海龟）向左转120度
08. p.forward(100)                   #向前画长度为100的直线
09. p.left(120)                      #画笔（海龟）向左转120度
10. p.forward(100)                   #向前画长度为100的直线
11. p.left(120)                      #画笔（海龟）向左转120度
```

部分绘图函数可以缩写如下：

fd() : forward()

bk() : backward()

lt() : left()

rt() : right()

运行程序的结果如图 6.3.1 所示。

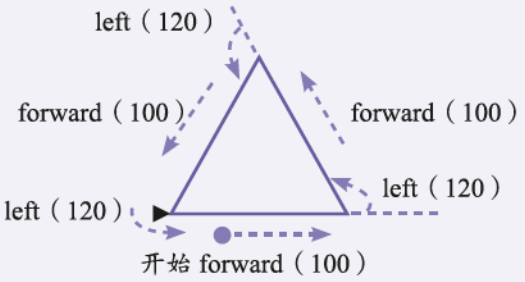


图 6.3.1　海龟作图之画等边三角形

当然，画笔的函数还包括后退画线backward(n)和向右转right(n)，如果只是想移动画笔（海龟）而不画线，可以用 penup()函数将画笔抬起，再用 forward(n)和 backward(n)移动，到需要画线的时候，再用 pendown()函数来恢复画线状态。

如图 6.3.2 所示，请在计算机上试试海龟作图。

（1）输入和运行画等边三角形的程序，并将程序改为使用 for 循环语句。

（2）编写一个画绿色正方形的程序，边长为 100，线宽为 3。

（3）编写一个画红色五角星的程序，边长为 100，线宽为 7。

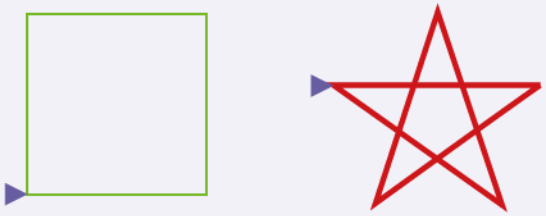


图 6.3.2　海龟作图练习

● 分解二叉树图形

二叉树结构是对自然界植物的一种仿生，我们对二叉树进行可视化，就能够还原出一棵树的大致样子。与数据结构的图示稍有不同的是，我们要把树根画在下方，让树干和树枝向上伸展，如图 6.3.3 所示。

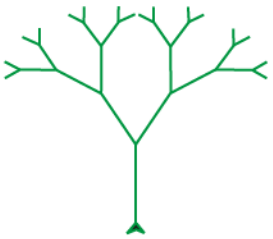


图 6.3.3　二叉树图形

正如二叉树结构分为根、左子树和右子树，可视化二叉树也同样分为三个部分：树干、向左倾斜的小二叉树和向右倾斜的小二叉树。二叉树的定义满足使用递归的条件，所以我们可以用递归的方法来绘制二叉树，如图 6.3.4 所示。

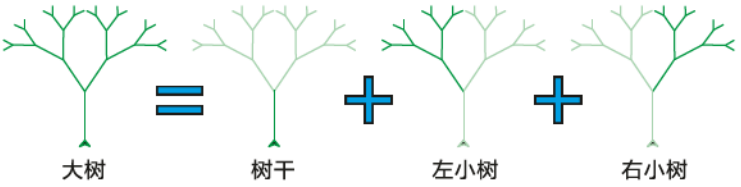


图 6.3.4　分解二叉树图形

※ 活动 2　绘制简单的二叉树

用递归方法，结合海龟作图模块的画图功能，可以很快写出绘制二叉树的程序。绘制二叉树的递归算法可描述如下。

（1）从初始位置出发，绘制指定长度的树干。

（2）向左旋转一个角度，绘制缩小的左子树。

（3）向右旋转一个角度，绘制缩小的右子树。

（4）画笔回到初始方向和位置。

下面，我们来实现一个绘制二叉树的函数 DrawBTree(n)，函数参数 n 表示树干的长度。

根据上述算法描述，补全下面的代码。

```
01. import turtle                #导入turtle模块
02. p=turtle.Pen()               #创建一支画笔（海龟）
03. def DrawBTree(n):
04.     if n<10:                 #如果树干太短，则不绘制直接退出
05.         return
06.     p.forward(n)             #画出长度为n的树干
```

```
07.    p.left(30)              #向左转30度
08.    DrawBTree(n-10)         #画一棵树干更短的二叉树
09.    p.right(__)             #向右转__度
10.    _____________           #画一棵树干更短的二叉树
11.    p.left(__)              #回正画笔的方向
12.    p.backward(__)          #回退到初始位置
13.    return
```

补全上面的程序代码之后，请将程序代码输入到计算机中，并加上下列调用代码，进行调试运行。

```
14. p.pencolor('green')       #设置画笔的颜色为绿色
15. p.left(90)              #将画笔从初始的朝向正右方调整为朝向正上方
16. DrawBTree(50)             #画一棵树干长度为50的二叉树
```

成功绘制出一棵简单的二叉树后，可以尝试修改树干的长度、左右子树倾斜的角度，看看绘制效果如何。

分小组讨论，为什么绘制二叉树的递归算法中，最后一步将画笔回复到初始方向和位置很重要?

● 递归与分形图形

分形（Fractal）通常被定义为“一个粗糙或零碎的几何形状，可以分成数个部分，且每一部分都（至少近似地）是整体缩小后的形状”，即具有自相似的性质。1973年，数学家曼德尔布罗（Benoit B. Mandelbrot）首次提出了分形的概念。

由于分形图形具有自相似的性质，所以特别适合用递归算法来绘制。除了二叉树之外，科赫雪花、谢尔宾斯基三角等也是分形图形，都可以用递归算法来绘制，如图6.3.5和图6.3.6所示。

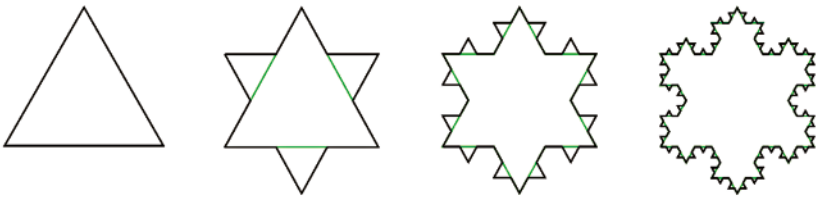


图6.3.5 科赫雪花分形图形

图6.3.6 谢尔宾斯基三角分形图形

分形几何是一门以不规则几何形态为研究对象的几何学。基于点、直线、曲线、平面、曲面这些“规则形状”的传统几何，无法恰当地描述大自然中普遍存在的不规则现象，如海岸线、山川形状、地面起伏特征、云、闪电、植物根系等。因此，分形几何学又被称为描述大自然的几何学，它引起了各个学科的关注。

任务二　绘制多彩的二叉树

※ 活动 1　设置可视化参数

画出简单的二叉树以后，我们还可以发挥想象力，为二叉树加入更多变化，使其更具有艺术化效果。例如：

- 树干的宽度随层级变细；
- 树干的颜色和长度引入随机变化；
- 小二叉树伸展方向引入随机变化。

由此，将艺术二叉树的递归算法修改如下。

（1）从初始位置出发，绘制指定长度、宽度和颜色的树干。

（2）向左旋转一个随机的角度，绘制随机缩小尺寸、随机颜色的左子树。

（3）向右旋转一个随机的角度，绘制随机缩小尺寸、随机颜色的右子树。

（4）画笔回到初始方向和位置。

绘制艺术二叉树的函数 DrawArtBTree(n,w,c)相比绘制简单的二叉树的函数 DrawBTree（n）增加了两个参数 w 和 c，其中 w 表示宽度，c 表示颜色。

根据上述算法描述，补全下面的代码。

```
01. import turtle                    #导入turtle模块
02. import random                    #导入random随机数模块
03. p=turtle.Pen()                   #创建一支画笔（海龟）
04. #颜色集合
05. colorset=('green', 'red', 'blue', 'brown', 'black')
06. def DrawArtBTree(n, w, c):
07.   if n<10:                       #如果树干太短，则不绘制直接退出
08.     return
09.   p.pencolor(__)                 #设定画笔的颜色
10.   p._____                        #设定画笔的粗细
11.   p.forward(n)                   #画出长度为n的树干
12.   tleft=random.randint(30, 60)                #左转随机角度
13.   tright=random.randint(30, 60)               #右转随机角度
14.   nleft=n-_____                  #左子树长度减少5~10随机量
15.   nright=n-_____                 #右子树长度减少5~10随机量
16.   #随机挑选左子树颜色
17.   lcolor=colorset[random.randint(__,__)]
18.   #随机挑选右子树颜色
```

```
    rcolor=colorset[random.randint(__,__)]
    p.left(_____)                                  #向左转
    DrawArtBTree(_____, w*0.5+1, ____)             #画左侧的小二叉树
    p.right(_____)                                 #向右转
    _____________________________________          #画右侧的小二叉树
    p.left(_____)                                  #回正画笔的方向
    p.pencolor(____)                               #设定画笔的颜色
    p._____________                                #设定画笔的粗细
    p.backward(____)                               #回退到初始位置
    return
```

补全上面的程序代码之后，请将程序代码输入到计算机中，并加上下列调用代码，进行调试运行。

```
p.left(90)              #将画笔从初始的朝向正右方调整为朝向正上方
#画一棵树干长度为60的艺术二叉树
DrawArtBTree(60, 8, 'green')
```

成功绘制出一棵艺术二叉树后，可以尝试修改调用参数、程序中的随机量、颜色集合等，看看绘制效果如何，并选出自己满意的绘制结果截屏保存。图 6.3.7 所示为一种艺术二叉树的绘制效果。

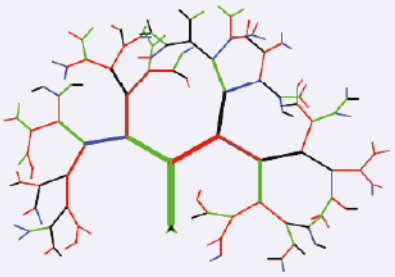


图 6.3.7　艺术二叉树的绘制效果

※ 活动 2　加入更多变化元素

我们还可以往艺术二叉树程序中添加代码，画出更多装饰，得到更加多姿多彩的树图形。例如，把树干太短不绘制改为绘制片片树叶，可以画出带有树叶的树，如图 6.3.8 所示。还可以随机加入花和果实，如图 6.3.9 所示。

李然　作品

图 6.3.8　艺术二叉树示例 1

张沙洲　作品

图 6.3.9　艺术二叉树示例 2

仔细调节树干之间的粗细过渡，增加背景元素，如图 6.3.10 所示。

计算机图形艺术：通过计算机程序绘制出具有欣赏价值的图形统称计算机图形艺术。计算机图形艺术不受绘画材料、绘画技法，甚至时间的限制，既能表现出传统书画艺术作品的形式，也能表现出从未有过的创新形式。除了创意线画图形之外，计算机动画、视频游戏、虚拟现实等计算机图形艺术已经在文化创意产业中广泛应用，深入千家万户。

图 6.3.10　艺术二叉树示例 3

有了这些参考作品的启发，请修改活动 1 中的艺术二叉树绘制函数 DrawArtBTree，绘制属于你自己的计算机艺术图形。

提示：由于缺省情况下海龟作图会显示作图过程的动画效果，所以如果绘制复杂图形的话，等待时间会非常漫长。可以在程序中导入 turtle 模块之后添加一行代码，取消绘制过程的动画效果，直接出现绘制完成的图形。

```
01. import turtle                    #导入turtle模块
02. turtle.tracer(0)                 #取消动画效果，直接显示绘制的图形
```

拓展练习

1. 如何画出三叉、四叉或者更多叉的树？

2. 查阅相关资料，编写绘制科赫雪花、谢尔宾斯基三角和希尔伯特曲线等分形图形的程序。

单元学习评价

本单元学习了树结构的基本概念及特征，亲历了利用二叉树解决问题的一般过程，进一步体验了递归思想在问题解决中的应用。你了解树的相关概念了吗？你是否能运用二叉树和递归思想解决实际问题？请参与小组交流并反思，开展自评或小组评价。

一、（多选）在解决以下问题的过程中，适合用树结构解决的是（　　）。

A. 实现一组数值中的快速查找　　B. 在城市道路网中寻找最短路线

C. 下棋过程的决策评估　　D. 对句子结构进行分析

二、各级行政区划的面积可以从下一级行政区划的面积求和得到，如果有最低乡级行政区划的面积数据，就可以统计出整个国家的行政区划面积。

编写程序来实现行政区划面积统计。

1. 问题分析。

解决此问题，用迭代还是递归的方法实现？为什么？

_____

2. 设计数据结构。

（1）采用什么数据结构来保存行政区划的面积数据？

_____

（2）为什么采用这种数据结构？

_____

3. 设计算法。

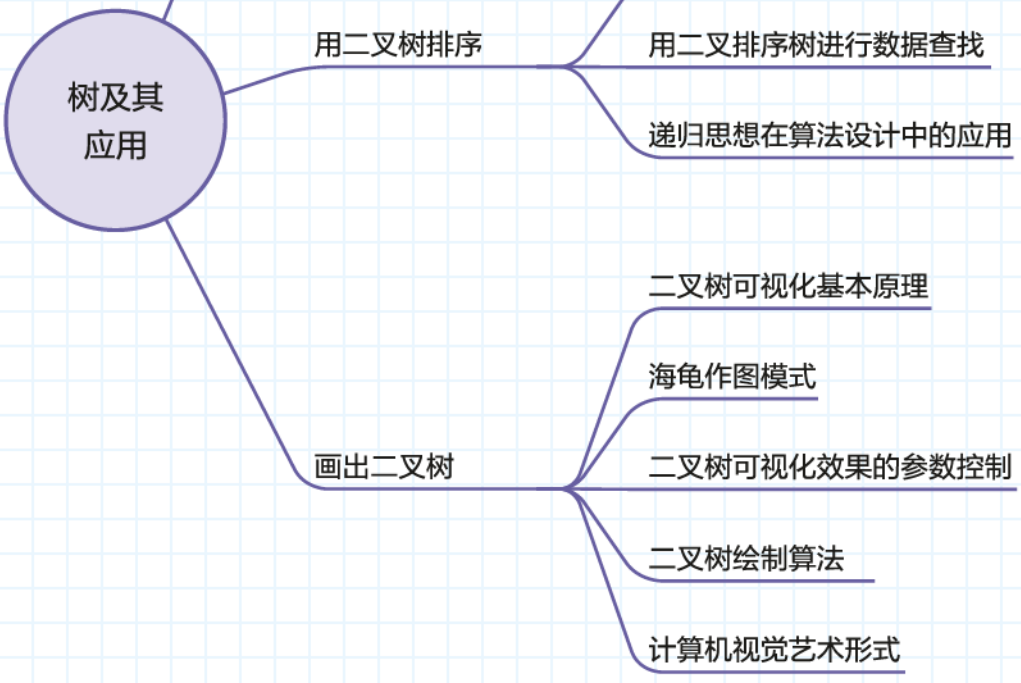
（1）怎样输入、输出数据？

_____

（2）画出算法流程图。

4. 编程实现。

单元学习总结


树及其应用
树结构及其实现
树结构的概念和特征
生活中蕴藏的树结构
二叉树抽象数据类型
二叉树的两种实现方法：顺序存储和链式存储
用二叉树排序
二叉树排序的基本原理
用二叉排序树进行数据查找
递归思想在算法设计中的应用
画出二叉树
二叉树可视化基本原理
海龟作图模式
二叉树可视化效果的参数控制
二叉树绘制算法
计算机视觉艺术形式

后　记

为全面落实立德树人根本任务，着力发展学生的核心素养，根据《普通高中课程方案（2017年版）》的精神，我们按照《普通高中信息技术课程标准（2017年版）》的要求对高中信息技术教科书进行了修订。

本书的修订由李冬梅、陈斌、钟建业、毛华均、张宁、陈阳、张春鸿、林志奕、汪知等直接参与，李艺、董玉琦、解月光、李冬梅、张义兵等在整体设计的过程中给予指导，后期张铭、张路教授和特级教师蒙广平审阅了本书修订稿并提出了宝贵意见，刘宝艳、王祺磊和邢波老师对部分章节进行了试教并提出了宝贵意见。在此，我们对所有关心、支持本书编写与修订的专家、学者和老师们表示衷心的感谢。

本书选用了一些图片和文字资料，对相关作者和出版社，我们一并表示诚挚的谢意。

编者

2019年6月