COMPUTER SCIENCE

Unplugged

Understanding computing through games and puzzles

Tim Bell
Andrea Arpaci-Dusseau 著
Ian Witten
Mike Fellows

Isaac Freeman
Matt Powell
插画

孙俊峰 杨帆
译

不插电的

计算机科学

玩智力游戏 学信息知识

華中科技大學出版社
http://www.hustp.com

- android与iphone及ipad开发书籍 ----------持续不断更新中.....
- c、c++、c#语言pdf书籍及vip视频教程 c、c++、c#、vc等-------持续不断更新中.....
- delphi《书籍》及《视频》教程 ----------持续不断更新中.....
- E网情深VIP系列视频教程 黑客破解菜鸟修练班，VB编程学习班，仿站学习培训，免杀培训，个人系统攻防系列教程，服务器搭建学习班，PHOTOSHOP平面设计班，基础制作论坛（论坛网站搭建），网赚系列教程，网站建设教程，网站漏洞基础，远程控制教程，软件破解班，脚本漏洞提权班
- IT9网络学院VIP系列视频教程 免杀培训班，VMware虚拟机，零基础学习C语言，网游外挂开发精品系列语音教程（外挂教程学习必备研修31课全），VB语言教程30课全，Delphi编程到精通，远程控制软件，加密解密班，网络安全与黑客攻防培训，从入门到精通完整系统化学习C++编程，从入门到精通零基础学习汇编，wordpress教程(个人博客系统49课全)，外行人做易语言盗号和钓鱼程序语音教程 网址：WLSAM168.400GB.COM
- Java书籍 ----------持续不断更新中.....
- photoshop、CorelDRAW、AutocAD等图像处理书籍及vip视频教程 -------持续不断更新中.....
- powerbuilder书籍大全
- Visual Basic语言vip视频教程及pdf书籍 ----------持续不断更新中.....
- windows、linux系统开发、系统封装等pdf书籍及VIP视频教程 ----------持续不断更新中.....
- 《3DS Max》pdf书籍
- 《汇编语言》、《反汇编》及《调试》pdf书籍及vip视频教程 ----------持续不断更新中.....
- 《电子书、电子书、还是电子书》pdf专题库 编程开发，家居美食，儿童益智，人物传记，增强记忆，快速阅读
- 信息系统项目管理师、网络工程师、系统分析师等软考类书籍
- 华中红客系列vip视频教程 脚本攻防培训班，源码免杀培训班，Css语言培训班，C语言，Dreamweaver网页设计，html网页设计培训班，PC安全班， php脚本语言培训班，VMWare虚拟机专题，webshell提权培训班，防站教程，零基础免杀培训班，刷钻速成班，脱壳破解班，外挂编写班，网络赚钱培训班，网站入侵培训班
- 外挂、驱动、逆向及封包视频教程 郁金香、独立团、夜猫论坛、天都吧、看流星论坛、一切从零开始等等
- 安全中国系列vip视频教程 易语言软件编程培训班，ASP.net网站开发项目实战培训班
- 我的收藏
- 按键精灵及TC脚本开发软件视频教程 ----------持续不断更新中.....

当前位置：/《电子书、电子书、还是电子书》pdf专题库

文件名 ⇕

PDF电子书专题库，内容详尽，每天不断更新！！

- 办公类软件使用指南
- 医学
- 历史人物传记
- 哲学宗教
- 外语资料（除英语外） （除英语外）
- 官场类小说
- 建筑工程类
- 情感生活类小说
- 政治军事
- 教育学习科普大全
- 文学理论
- 智力开发、增强记忆、快速阅读技巧大全
- 社会生活
- 科学技术
- 程序编程类
- 经济管理
- 网络安全及管理
- 网赚系列
- 美食小吃烹饪煲汤大全
- 课外读物

本网盘内容太多，持续不断更新，发布各类视频教程、pdf书籍，包括破解、加解密、外挂辅助制作，易语言培训教程、编程语言、网页制作等等，教程及书籍仅用于学习,如用于商业或非法用途的后果自负！

网址：WLSAM168.400GB.COM

- VIP培训教程，易语言黑月VIP视频教程，天都网易语言系列培训教程(100集全)，集中营易语言学习视频(80集)
- 棉猴系列vip视频教程 gh0st远控控制源码讲解教程，套接字编程，DLL程序编写，键盘监听驱动程序编写，驱动基础教程，AsyncSelect模型QQ程序教程，C++语言入门基础，NB5.5源码分析教程
- 游戏开发pdf书籍 ----------持续不断更新中.....
- 炒股投资pdf书籍及视频教程 短线高手系列，短线天王系列，操盘论道系列，翻倍黑马，看盘快速入门，庄家手法大曝光等等。 网址：WLSAM168.400GB.COM
- 热门小说集中营 傲世九重天，网游之三国时代，武动乾坤
- 甲壳虫VIP教程全集 asp教程，Delphi培训班，FLASH培训班，Java培训班，linux培训班，PHP培训班，源码免杀班，甲壳虫C++，脚本功防班，免杀班初、中、高级班，破解班，源码免杀班，脱壳班，易语言培训班，无特征码免杀，网站架构培训班，外挂高级班，外挂初级班第1、2部
- 破解、免杀、入侵、脱壳、攻防及漏洞分析系列VIP视频教程（80多部） 天草、黑客动画吧等等--------持续不断更新中....
- 网站建设相关的pdf书籍及各种vip视频教程 ----------持续不断更新中.....
- 网赚、淘宝系列vip视频教程 网赚30天新人魔鬼训练，屠龙网赚团队vip课程，站长大学网赚视频（50课全），图腾团队日赚1000元竞价行销教程，屠龙团队淘宝宝贝卖疯系列，站群网赚系列，淘宝开店视频，红星挂机日赚10元，百万流量系列，漂流瓶圣手全自动挂机引，贴吧邮件定向营销疯狂成交量月入万元
- 英语学习资料百科大全 不断更新。。。
- 饭客论坛系列VIP视频教程 脚本入侵班，黑客之免杀教程，易语言教程，无线网络攻防教程，入侵教程，delphi系列教程，黑客基础入门
- 黑客书籍 有关黑客、安全、加解密技术等等----------持续不断更新中.....
- 黑手安全网VIP系列视频教程 DIV+CSS网页布局，Dreamweaver教程，flsah动画教程，photoshop教程，跟我一起学C++课程，抓鸡
- 黑鹰、黑基、黑防、黑盾vip系列视频教程 破解提高班66课全，SQL注入，ASP注入教程，完完全全学会抓肉鸡，脱壳破解教程50课全，提权班，C语言特训班26讲全，黑客脚本特训班，黑客工具特训班，dedecms仿站教程，VC编写远控30课全，网页美工特训班，木马免杀特训班，驱动开发技术VIP培训班，外挂破解等等。

- [电脑世界的通关密语：电脑编程基础].(杉浦贤).滕永红.扫描版.pdf
- [程序语言的奥妙：算法解读（四色全彩）].(杉浦贤).李克秋.扫描版.pdf
- [差错：软件错误的致命影响].(帕伯斯).邝宇恒等.扫描版.pdf
- [算法之道（第2版）].邹恒明.扫描版.pdf
- [O'Reilly：深入学习MongoDB].(霍多罗夫).巨成等.扫描版.pdf
- [深入浅出WPF].刘铁猛.扫描版.pdf
- [Go语言·云动力（云计算时代的新型编程语言）].樊虹剑.扫描版.pdf
- [精通.NET互操作：P/ Invoke、C++ Interop和COM Interop].黄际洲等.扫描版.pdf
- [编程的奥秘：.NET软件技术学习与实践].金旭亮.扫描版.pdf
- [O'Reilly：学习OpenCV（中文版）].(布拉德斯基等).于仕琪等.扫描版.pdf
- [Go语言编程].许式伟等.扫描版.pdf 网址：WLSAM168.400GB.COM
- [MySQL技术内幕：SQL编程].姜承尧.扫描版.pdf
- [Tomcat权威指南（第2版）].(布里泰恩等).吴豪等.扫描版.pdf
- [Ext江湖].大漠穷秋.扫描版.pdf
- [IT名人堂·Oracle DBA突击：帮你赢得一份DBA职位].张晓明.扫描版.pdf

Total: 77 1 2 3 4 5 6 ›

COMPUTER SCIENCE

Unplugged

世界范围的信息科学普及项目登陆中国，专为中国青少年量身打造的开放式学习案例。

32个不插电的信息科学活动，点燃“计算思维”灵感，优化信息技术课堂。

今天你UNPLUG了吗？

内容提要

“Computer Science Unplugged”（不插电的计算机科学）是面向世界范围的信息科学普及项目，它透过一些既有趣又容易的活动来达到学习“计算机科学”的目的。这些活动是专门为青少年学习者所设计的。在这些活动中，我们可以学习到计算机运作的一些基本原理，有趣的是，你根本不必用到任何实体的计算机。

这些活动的原创者是新西兰Canterbury大学的教授和两位中小学教师，他们依据自己的实务教学经验设计了大多数活动。随着该项目在世界范围内的普及，具有各国特色的新颖案例不断补充进来。这种**“玩中学，做中学”**（learning by playing，learning by doing）的信息技术学习方法目前在美国、新西兰、意大利、日本、韩国已产生广泛影响。

本书中的活动可以丰富和扩充教师的教学，**在计算机、信息技术、数学或英语课堂（以及奥赛培训）**中可以作为补充知识使用。由于教学者不必是一位计算机专家，所以也适合家长和孩子一起在家中探索计算机的迷人与奇妙！**书中的32个活动妙趣横生，**透过这些有意义的活动，学习者们不仅可以学到信息学科中一些极有意义的科学知识，还能发挥想象、激发创意，有效提升自身的逻辑思维能力以及和同伴沟通的能力。

更多信息请登录 http://csunplugged.org了解。

◎ 策划编辑：徐晓琦
E-mail: xxqbust@163.com
◎ 责任编辑：徐晓琦
◎ 封面设计：刘　卉

ISBN 978-7-5609-6062-3

9 787560 960623 >

定价：29.80元（含1CD）

COMPUTER SCIENCE

Unplugged

Understanding computing through games and puzzles

Tim Bell
Andrea Arpaci-Dusseau 著
Ian Witten
Mike Fellows

Isaac Freeman
Matt Powell
插画

孙俊峰 杨帆
译

不插电的计算机科学

玩智力游戏 学信息知识

NLIC 2970653479

华中科技大学出版社
http://www.hustp.com
中国·武汉

图书在版编目(CIP)数据

不插电的计算机科学/Tim Bell　等著;孙俊峰　杨　帆　译.—武汉:华中科技大学出版社,2010.11

ISBN 978-7-5609-6062-3

Ⅰ.不…　Ⅱ.①T…　②孙…　③杨…　Ⅲ.信息技术-青少年读物　Ⅳ.G202-49

中国版本图书馆 CIP 数据核字(2010)第 046090 号

不插电的计算机科学　　Tim Bell　等著　孙俊峰　杨　帆　译

策划编辑：徐晓琦
责任编辑：徐晓琦
装帧设计：刘　卉
责任校对：周　娟
责任监印：熊庆玉
出版发行：华中科技大学出版社(中国·武汉)
武昌喻家山　　邮编：430074　　电话：(027)87557437
录　　排：武汉正风图文照排中心
印　　刷：湖北新华印务有限公司
开　　本：880mm×1230mm　1/16
印　　张：14
字　　数：393 千字
版　　次：2010 年 11 月第 1 版第 1 次印刷
定　　价：29.80 元(含 1CD)

序

by Tim Bell

计算机深入到我们生活的各方各面。人们在工作中使用计算机，在家中使用计算机，甚至在上下班的路上也在使用计算机(不要忘记手机和 MP3 播放器也是特殊形式的计算机哦)！但是计算机究竟是如何工作的？它们怎样进行"思考"？人们又如何让计算机运行得更快更好呢？

不同于传统意义上教授"如何使用计算机"的教程，本书着眼于各项计算机技术的原理，它将为你展示真实发生在计算机世界中的点滴，让你理解新技术是如何被设计出来的，并且开发你的"计算思维"来提高解决问题的能力。最重要的是，这本书的目的是让你在学习的过程中也能玩得很开心！

本书作为"不插电的计算机科学"项目[1]的一部分，旨在让学生们在学习程序设计之前，先来探寻、领悟计算机运作及其解决问题的精妙思想——事实上，在学习上述知识的时候你甚至都不用打开计算机。书中的每项活动都配有详细的解释说明，指出它和相关技术之间的联系以及它如何进一步影响我们的日常生活。这种计算机教学法目前已在世界范围内广泛开展，并被证明相当有效且深受好评。

不用计算机便能学会计算机，是不是让你非常惊讶呢？你能在官方网站 www.csunplugged.org 中找到更多有关"不插电的计算机科学"项目的介绍。

如何使用这本书

你既可以拿此书自学，也可以将它作为信息技术课或数学课的补充读物，在班上和伙伴们一起学习。老师们在指导学生玩游戏时，并不需要深入讲解专业的计算机知识，只要学生稍稍掌握基本的数学技能和拥有一颗学习新知识的心即可完成书中所有活动。我们也特别编写了教师手册部分，里面提供了每道问题的答案和课堂教学的小窍门。

书中的一些小游戏，如果和伙伴们一起来玩会更加有趣！与此同时，我们也提供了自己一个人进行游戏的方案。本书同样也适用于课外兴趣小组或者父母和孩子们进行的亲子活动。

这里采用的教学法能激发学生们依靠自己的力量发现新事物。这是本书一个非常重要的教学技巧，因为本书的主要目的是帮助学生们开发出更好解决问题的方法和自我获取新知识的能力。它将重点强调"计算思维"的培养，而不是单调地学习已有的计算机技术。

随书的光盘中有我们精心准备的视频录像，将帮助你更好地来理解如何玩书中的小游戏。尽管不观看录像也绝对不会影响到我们的学习，不过我们大力推荐在进行游戏之前观看这些录像。录像并不会完整解释书本中的观点，但它们会给你一个直观的印象怎样去做。请记住，只有你亲身参与游戏中时，你才能真正学到东西。

本书含有大量的问答环节，它们是学习过程中的关键部分。这些问题并不是用来测试你对文字本身的理解；相反，它们是你学习的一部分。所以请不要跳过这些问题，而且请确保你能靠自己的力量回答每道问题。如果你被某些问题卡住，你将会在教师手册中找到每道问题的详细解答。

---

[1] 译者注："不插电的计算机科学"项目(Computer Science Unplugged)是一个世界范围的信息科学普及项目，它透过一些既有趣又容易的活动，来达到学习"计算机科学"的目的。

在一些小游戏中，你需要准备一些供书写的纸张；如果你不想在书上涂涂画画，可以从随书的光盘或从网站上打印出相关内容。请访问本书的网页 http://www.hustp.com/forward/toBookPublic.do 以获取更多丰富的内容和信息。

著名的计算机科学家迪杰斯特拉曾经说过“计算机科学不只是关于计算机，正如天文学不只是关于望远镜”。我们深深寄望，通过不依靠计算机来学习计算机能帮助你更好地理解计算机这门科学，拓展你的创造力，帮助你运用计算思维。更重要的是，我们衷心希望你能在游戏的过程中得到乐趣，并能成功完成本书中每一个挑战！

感谢

这个项目得益于许多人的辛劳付出和宝贵建议。

非常感谢华中科技大学出版社对这本书注入的心血和支持，特别要感谢王连弟总编和徐晓琦编辑。来自华中科技大学的刘宏教授和谢夏老师给予我们大量珍贵的建议和支持，对本书的编撰起了关键的作用。和宋恩民教授一起合作，让我们得以反复讨论和试讲本教程。同样，在和来自桂林电子科技大学的董荣胜教授和刘亮龙的讨论中，我们也大有收获。

课堂环节的内容得益于 Robyn Adams 和 Jane McKenzie 发挥的关键作用，他们带来的许多点子都极大地充实了本书内容。

本书中的文字多亏了 Andrew Bell、Anita Dusseau、Stephen Fitchett、Erin Gonzalez、John O'Malley、Pa Kou Vang、Janina Voigt、Phoua Xiong 和 Yuan Wang 的锐眼来校阅。同样要感谢昆明冶金高等专科学校的韩迎春老师对全书细致地审校。

近年来，在许多孩子和教师们的帮助下，我们不断优化项目思路。在中国，我们曾在华师一附中（武汉）、华中科技大学附属中学（武汉）和雷店高中（湖北省英山县）进行了成功的试讲。基督城中学、雪梨小学、尼泊尔小学、西域小学和插云小学（均位于新西兰基督城），舒尔物德山小学（美国威斯康星州麦迪逊市）和南方公园学校（加拿大卑诗省维多利亚市）的小朋友和老师们是我们许多小游戏的试验先驱。在此，我们要特别感谢 Liz Eddington、Linda Picciotto、Karen Able、Bryon Porteous、Paul Cathro、Tracy Harrold、Simone Tanoa、Lorraine Woodfield 和 Lynn Atkinson 向我们敞开热情的课堂之门，并对我们的小游戏环节提出珍贵的建设性意见。其中，为我们提供小游戏 9.1 中的照片和天平创意的是嘉田胜先生。

课堂试讲离不开 Gwenda Bensemann、Richard Lynders 和 Sumant Murugesh 的大力支持。有一些小游戏得到了维多利亚市“Mathmania” 团队和 Kathy Beveridge 的支持。早期版本的插画由 Malcolm Robinson 和 Gail Williams 绘制提供，其中 Hans Knutson 也贡献了不少宝贵提议。在“不插电的计算机科学”项目的发展中，Matt Powell 为我们提供了无价的支持。特别感谢帮我们测试许多小游戏并提供许多建议的 Paul 和 Ruth Ellen Howard。同样地，Peter Henderson、Ray Hunt、Bruce McKenzie、Joan Mitchell、Nancy Walker-Mitchell、Gwen Stark、Tony Smith、Tim A. H. Bell[2]、Mike Hallett 和 Harold Thimbleby 提供的大量建议也让我们深受裨益。

感谢我们的家人 Fran、Judith、Pam 和 Remzi 无私的支持，Andrew、Anna A、Anna W、Hannah、Madeline、Max、Michael 和 Nikki 为这个项目[3]带来许多启发，他们往往是测试各项小游戏的先锋。

我们同样也深深感谢 Google 公司和 Brian Mason 科学技术研究院对该项目的信任和赞助，他们对许多小游戏的开展提供了支持。

非常欢迎你对本书中的游戏提供想法和建议。你能通过 www.csunplugged.org 站点联系到本书的作者们。

[2] 和本书第一作者非同一人。

[3] 事实上，文字压缩的小游戏是由 Michael 发明的。

你将从本书中读到

许多人都知道如何使用信息技术，但是只有少数人理解技术的原理。掌握这些知识，不仅能让你更高效地使用计算机，甚至能让你研发出促进世界发展的新技术。

你或许已经目睹过近几年产生的各种流行电子技术，像搜索引擎、视频网站、网络社区和 MP3 播放器。但是，是谁设计出这些系统并解决了诸多技术难题呢？是计算机科学家、计算机工程师和软件工程师们设计研发出了大多数新技术，你或许能在高等教育阶段学到这些内容，但是我们认为你应该更早接触这些话题。这也是为什么我们出版了这本书。本书中许多章节的内容来自于大学课程，但是我们把它们特别改编成游戏和谜题的形式，从而在你并不是计算机程序高手的时候也能游刃有余地享受学习过程。事实上，你完全可以将计算机放在一边！

这本书还有一个好处：它将开发你的创新思维，使你自己想出有关信息技术的新点子。为计算机界作出巨大贡献的人们都是那些善于解决问题、富有集体合作精神、能不断想出创意的人们。当你还在学生的时候就应该开始这方面的积累，今后或许你就是那个为大家创造新科技从而改变人们生活质量的那个人！

这本书并没有涉及计算机是如何工作的全部细节，但是书中内容能培养你的洞察力。我们主要的目的是揭示那些被计算机科学家和软件工程师运用于设计计算机系统的多元化的思路和技术。当你发现那些看起来难以解决的问题，原来能如此"简单"地被研究人员找到解决途径时，你一定会感到惊讶并产生浓厚兴趣的。

为了满足人们的需要，计算机会在内部处理两类事情。首先，计算机会储存**数据**。数据就是计算机工作时处理的原材料对象（如数字）。计算机会将内部数据转换为人们可以理解的信息（如词语、数字和图像），这是我们将在第 1 章到第 7 章重点讲述的内容。

然后，计算机通过执行一系列指令来对数据进行处理。这些指令使得计算机能解决许多问题，例如排序、查找和发送信息。**算法**是一系列解决问题的清晰指令。算法描述了我们是如何让计算机解决问题的，这些我们将在第 8 章到第 15 章为大家具体讲解。

所以接下来……拔掉你的计算机插头吧，让我们一起来享受学习计算机的乐趣！

...Contents 目录

Contents

目录

第1章

二进制数和“比特”

Binary Numbers and “Bits”

计算机中的数据无一例外都是以一系列的0和1的形式储存的，但是计算机却能展现形式多样的信息，如文档、网页、照片、音乐、视频等。这是为什么呢？在第1章的小游戏中，我们将来学习如何仅用数字0和1来展现形式各异的信息。

All data in computers is stored and transmitted as a series of zeros and ones. Yet computers can represent a rich range of information: documents, web pages, photographs, music, videos, and so on. How can this be? In these first activities we will begin to learn how different information can be represented using just two different digits: zero and one.

介绍
Introduction

计算机功能强大而且十分灵活，人们可以通过计算机浏览网络、发送邮件、观看视频、编辑图片、欣赏音乐、创作小说或者控制机器。但是计算机也有做不到的事情。有时候，人们会抱怨自己的计算机运行得太慢——尽管事实情况是一台价格低廉的计算机也能进行高达每秒数百万次的运算；有时候，想要下载的文件会因为体积过大而难以下载；有时候，计算机还会搞出一堆费钱费时的麻烦事。然而正是因为我们的生活和世界如此依赖计算机，才更应该一同理解它们的优点和不足。

Computers are incredibly powerful and flexible. They can be used to browse the internet, send emails, watch videos, edit photos, listen to music, write books and control machines. But they also have limitations. Sometimes people get frustrated with how slow they are—even though a low-cost computer can do billions of operations per second. Sometimes files are too big to download. Sometimes computers cause costly and time-consuming problems. Because so much of our lives and our world depends on these machines, it's important to understand both their power and their limitations.

对信息做任何有意义的操作前，你需要对其进行保存，以便于第二天想继续工作时还能再找到它。我们将从"数"在计算机中的储存说起，最终我们将学到如何储存文字及图片，如何向小容量的内存空间中储入体积较大的数据，如何防止存储错误的发生，以及如何计算储存了多少信息。

Before you can do anything useful with information you have to be able to store it, so that you can read it again tomorrow. We will begin by learning how numbers are stored on computers. Eventually, we will learn about how to store words and pictures, how to store lots of data in a small amount of space, how to prevent errors from happening, and how to measure the amount of information we are storing.

首先，让我们来认识一下二进制数(binary number)。所有电子计算机中的数据都是以二进制的形式来储存并传输的，它们直接影响到计算机工作的各项指标：硬盘有多满？下载速度有多快？网上支付有多安全？屏幕可显示的最高分辨率是多少？计算机的最快运算速度是多少？理解了二进制数的工作机制也就意味着你理解了计算机是怎样处理

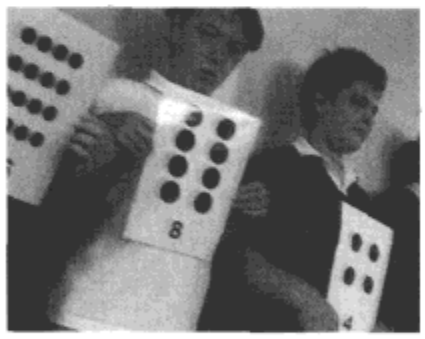
But first we are going to learn about **binary numbers**. Because everything on a digital computer is stored and transmitted using binary numbers, they affect many of the things we worry about on computers: how full the disk is, how fast we can download, how secure an online payment is, how high the quality of graphics is on a screen, and how fast the computer can

信息的,许多计算机技术的神秘面纱也会随之被揭开。

很神奇吧,计算机中全部的信息都只用两个数值来储存:0 和 1。这些信息在计算机的内存中以晶体管的开、关状态来表示(开代表 1、关代表 0)。信息还能被储存在磁盘(比如硬盘)、闪卡(比如数码相机的储存闪卡)或者磁带(比如录影带)上,我们将在第 2 章的一组游戏中进一步讲解这些内容,现在主要的任务是来理解计算机如何只用两个数字来表示信息的,即为什么我们称计算机系统为二进制系统(binary 的前缀"bi"代表"两个"的意思,比如自行车 bicycle,双翼机 biplane 或者双筒望远镜 bifocal)。

在我们开始二进制的探秘之旅前,不妨先思考一下,你怎样仅用 0 和 1 来数数呢? 请继续阅读……

视频:请观看二进制数字游戏的录像。



小游戏 1.1 二进制卡片
Activity 1.1 Binary cards

在进行这项小游戏前,你需要准备 5 张卡片,每张卡片画上如下的圆点图案。你也可以用数字替换下面的圆点。

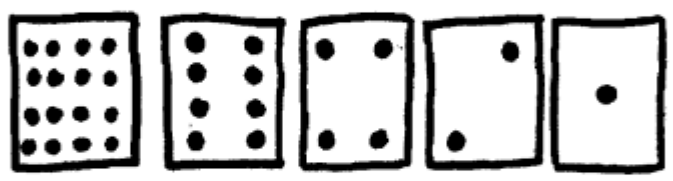
do calculations. Understanding how binary numbers work means that you understand what is happening with the information on your computer in great detail, and it will remove a lot of the mystery of computers.

Amazingly, computers store all information and numbers using only two values: zero and one. This information can be stored in a computer's memory with a transistor that is switched off or on (for zero and one). It can also be stored on magnetic disks (such as a hard disk), flash cards (such as the one in a digital camera), and tapes (such as a digital video tape). We will learn more about this in the next set of activities, but the main thing to understand now is that computers only represent things using two different values, which is why the system is called binary (the prefix "bi-" means two, as in bicycle, biplane, or bifocals).

But first, how can you count using only zero and one? Read on . . .

Video: Watch the Binary Numbers video.

To do this activity, you need to make 5 cards or pieces of paper with dots on them as in the picture below. You can put numbers on the cards instead of dots if you prefer.



按照如上顺序，在桌子上摊开卡片。

Put the cards on the table in front of you, in the order shown.

a 你留意到卡片上的圆点图案有怎样的规律了吗？

a What do you notice about the number of dots on the cards?

b 如果我们在这些卡片的左侧增加一张卡片，那么这张卡片上需要画几个圆点呢？

b How many dots would the next card have if we added a 6th card on to the left?

c 如果向左侧继续再增加一张卡片，那么这张新的卡片上需要画几个圆点呢？

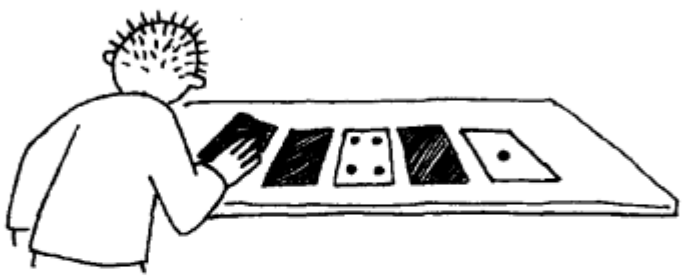
c How many dots would there be if we added yet another card on the left?

d 你是依照怎样的规律来决定刚才新增卡片上的图案的？

d What rule are you using to create the next card?

我们可以将一些卡片翻转使其背面朝上，然后计算正面朝上卡片上点数之和来代表不同的数字。比如，下图中仅将 1 个圆点和 4 个圆点的卡片正面朝上代表了数字 5。这个游戏的规则只有一个：保证卡片要么正面朝上，要么翻过来背面朝上。

We can use these cards to make numbers by turning some of them face down and adding up the number of dots that are showing. For example, we can make the number 5 by turning over the 1-dot and the 4-dot cards. The rule is that each card is either fully visible, or fully hidden.



e 你需要令哪几张卡片正面朝上来表示数字 6 呢？

e Which cards should you turn face up so that exactly 6 dots are showing?

f 需要利用哪几张卡片表示数字 20？

f How can you make the number 20?

g 需要利用哪几张卡片表示数字 15？

g How can you make the number 15?

h 需要利用哪几张卡片表示数字 21？

h How can you make 21?

i 需要利用哪几张卡片表示数字 30?

j 任何数字都能用不止一种方法来表示吗?(比如,你能用两种不同的方法来表示数字 5 吗?)

k 5 张卡片能表示的最大数字是多少?

l 它们能表示的最小数字又是多少呢?

让我们来试试从 0 数到 31。把全部卡片翻转过来背面朝上代表 0,然后一张一张翻开符合要求的卡片,从 1、2、3、4 一直数到 31。

m 在 0 至 31 之中,有你无法表示出来的数字吗?

n 令数字递增 1 最简单的方法是什么?(提示:每当数字增加 1,画着 1 个圆点的卡片发生了变化吗?如果画着 1 个圆点的卡片需要被调整成正面朝下,那么画着 2 个圆点的卡片需要调整朝向吗?借由这点,相信你能找出将数字递增 1 的简单方法。)

刚才的卡片游戏利用了二进制数的原理。我们日常生活中用到的都是由 0 到 9 组成的十进制数(decimal number),所有的十进制数都是用 10 个不同的数字组成(1 到 9,再加上一个 0),一旦某一位大于 9 就需要再增加新的位数。比如十进制数 34 就由两个十进制位组成,"3"在这里代表 3 个十。而计算机仅用两个数字 0 和 1 来表示信息,例如刚才使用的卡片,卡片背面朝上代表数字 0,正面朝上代表数字 1。因为只用到两个数值,我们将它们称之为二进制位。5 张卡片可以表示一个 5 位的二进制数。

由于每位只有两种数值可供选择,所以二进制有时候也被称为基数为 2 的数制(base-two)。

i How can you make 30?

j Is there more than one way to form any number? (For example, are there two ways to make the number 5?)

k What is the biggest number you can make with these five cards?

l What is the smallest number?

Now we are going to try counting from 0 up to 31. Flip all the cards face down to represent 0, then flip cards over to get 1, then 2, 3, 4, and so on up to 31.

m Is there any number you can't make between the 0 and 31?

n Is there an easy way to count up by 1? (Hint: what happens to the 1-card each time you add one? If the 1-card has to change to facedown, what happens to the 2-card? Based on that, can you find a simple description of how to add 1?)

The cards that you have used are using binary numbers. Normally we work with **decimal numbers**, which are made up of digits from 0 to 9. There are only ten different digits (1 to 9, plus the 0 digit), but to get numbers higher than 9, we use more digits. For example, the number 34 has two decimal digits, and the 3 actually means 3 lots of 10. On computers the only digits are 0 and 1—there are just two different digits. With the cards, a 0 is when the card is face down, and a 1 is when you can see the dots. Because there are just two values, we call them binary digits. The 5 cards represented a 5-digit binary number.

Sometimes binary is called "base-two" because there are only two possible values for each digit.

比如，二进制数 01001 一共包含 5 位，如果我们用翻转的卡片来表示各位，再对应卡片上的点数，就会得到十进制数 9。下面每张卡片代表一个二进制位。

For example, the binary number 01001 has five binary digits, and if we turn over the cards to match the digits, we get the decimal number 9. Each of the cards we have used represents a single binary digit.

术语一点通

The phrase "binary digit" is so common in computer science that it is given an abbreviation: "bit". So a bit is just a digit that can be a zero or one.

在计算机科学中被广泛使用的"二进制位"有一个昵称"比特"(bit)。一个比特即是一个数位，其值可以为 0 或 1。

o 这些卡片一共组成多少个比特呢?

o How many bits do we have with these cards?

p 用卡片摆出二进制数 01101，它对应的十进制数为多少?

p Use the cards to make the binary number 01101. Which decimal number does it represent?

十进制数是人们常用并熟悉的数字形式，十进制位上的数值范围从 0 到 9，每位上都有 10 个不同数值，因此十进制有时候也被称为基数为 10 的数制(base-ten)。

Decimal numbers are the types of numbers that people usually use; decimal digits can have the values 0 to 9. Since each digit can have 10 different values, decimal is sometimes called base-ten.

Binary	0, 1	Base-Two
Decimal	0, 1, 2, 3, 4, 5, 6, 7, 8, 9	Base-Ten

q 二进制数 00110 对应的十进制数是多少?

q What is the binary number 00110 in decimal?

r 二进制数 01110 对应的十进制数是多少?

r What is the binary number 01110 in decimal?

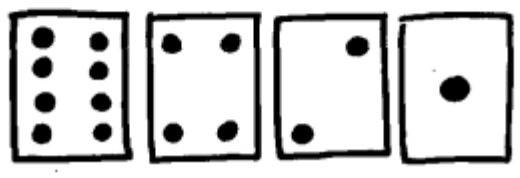
s 二进制数 10001 对应的十进制数是多少?

s What is the binary number 10001 in decimal?

小游戏 1.2 短短的二进制数和长长的二进制数
Activity 1.2 Shorter and longer binary numbers

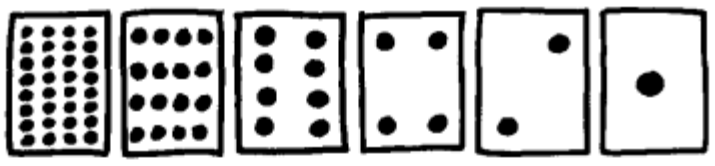
刚才我们一直用 5 个比特来表示二进制数，但是在现实中往往需要用到各种不同位数的二进制数。例如，我们可以用 5 张卡片将十进制数 9 表示为 01001，而用 4 张卡片将其表示为 1001 也是行得通的。

So far we've always used 5 bits to represent numbers, but sometimes we will need more, and other times we use fewer. For example, we wrote the number 9 as 01001 using 5 cards, but with 4 cards we only need to write down 1001.



写下如何用 6 张卡片来表示数字 9。

Write down the number 9 using 6 cards.



用以上 6 张卡片最大能表示多少呢？

What is the largest number that can be represented using 6 cards?

我们可以用任意的符号来代表二进制中的 0 和 1。接下来不妨挑战一下用不同的符号来表示数字。

We can use any symbols to represent our 0 and 1 in the binary numbers. The following challenge uses all sorts of symbols to represent numbers.

算出下面各组符号表示的数值。

Work out the values represented by the symbols in the following chart.

☒☑☒☒☑ =
(☑=1, ☒=0)

+ + × + =
(+=1, ×=0)

⇧⇩⇧ =
(⇧=1, ⇩=0)

↻↻↻↺↻ =
(↻=1, ↺=0)

○○○○○ =
(⊙=1, ○=0)

▲▼▲▼▼ =
(▲=1, ▼=0)

 =
(=1, =0)

♠♠♠♠♠ =
(♠=1, ♣=0)

☹ =
(☺=1, ⊗=0)

✪☆☆☆✪☆ =
(✪=1, ☆=0)

👍👎👍👎 =
(👍=1, 👎=0)

✿✿❀✿❀✿ =
(✿=1, ❀=0)

进阶篇

Details for experts

(跳过每章的这个小节不会影响你的阅读,不过这里将谈到一些你或许会感兴趣的内容哦!)

二进制数中,任何数位都是其右侧数位的 2 倍,这就好比在十进制数中,任何数位都是其右侧数位的 10 倍。比如,十进制数 423 中,最左侧的 4 代表 4 的 100 倍,即 400,而 2 仅代表 2 的 10 倍,即 20。比起你熟悉的十进制,其实二进制一点都不难对不对?它们仅仅使用了不同的进位基数而已。

如果二进制数是以 0 开始的,这些 0 就可以被忽略。例如十进制数 7 可以用 5 位的二进制数 00111 来表示,它也可以写成 0111 或者 111,这就像十进制的三位数最大可以到 999,但是写成 007 的十进制数代表的依旧是数字 7。同理,00111 只意味着数据以 5 个数位的形式储存,而拥有 5 个数位的二进制数最大为 11111,即十进制的 31。

(You don't have to read this section for each topic, but it might contain some information that you were wondering about.)

Each digit in a binary (base-two) number is worth twice as much as the one to the right of it, in the same way that each digit in a decimal (base-ten) number is worth ten times as much as the digit to the right (for example, in the decimal number 423, the left-hand digit is worth 100 times as much as it looks (i.e. 400), while the 2 is only worth 10 times as much as it looks (20). So the binary number system isn't that different to the numbers that you're familiar with – it just uses a different multiplier for the value of the digits!

If a binary number begins with zeros, they can be left out. For example, the decimal number 7 can be written as the 5-bit number 00111. It can also be written as 0111 or just 111, in the same way that the original number 7 can be written as 007. It's still just the number 7, although having 3 digits implies that the 7 is part of a series that might go up to 999. In a similar way, 00111 means that the number is being stored in 5 bits, and the largest possible number is 11111 (31 in decimal).

有趣的事
Curiosity

生日蛋糕或许会很危险哦，特别是当你越来越年长的时候！在你过 10 岁生日时，生日蛋糕上会被插上 10 根蜡烛。可当你到 30 岁的时候，就需要插上更多的蜡烛啦。如果把它们插得过密，火焰弄不好会烧掉整块生日蛋糕……

Birthday cakes can be dangerous, especially when you get older! If you are turning 10, you will have 10 candles on your cake. But when you turn 30 you need lots more candles. And if they are too close together the flames can get too strong, and even burn the cake.

当然你可以买一支十进制数造型的蜡烛插上去，可惜这种蜡烛要贵一些。

You could buy a candle with a decimal digit on it, but they are expensive.

那么能用二进制数来表示年龄吗？答案是肯定的！在二进制的世界里，任何数字都能用拥有两个不同数值的东西来表示。前面的小游戏中我们就用到卡片的正面和反面来表示数值 0 和 1。

Can you use binary numbers instead? Yes! In binary, any number can be represented using anything that can have two different values. In the previous activity we used 0 and 1, and we also used cards that could be face down or face up.

在生日蛋糕上，每支蜡烛可以有点燃或熄灭两种状态。右面这个蛋糕上的蜡烛就代表了二进制数 01001，即 9。而当点燃全部蜡烛的时候，就能代表 31 岁啦——这实在比同时点燃 31 根蜡烛安全多了！

On a birthday cake, each candle could be lit or not lit. So the following cake represents the binary number 01001, which is 9. If you light all 5 candles, the number would be 31 – much safer than using 31 candles!

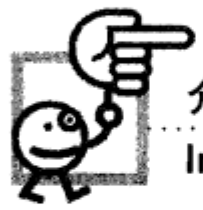
第2章

从小比特到大数字

From Small Bits to Big Numbers

前面一章我们了解到如何只用0和1来数数。本章我们将要探索一下计算机中二进制数是以怎样不同的形式来储存的，以及计算机是如何高效地储存那些庞大数值。

We have discovered how to count with only two digits: zero and one. Now we will explore different ways that binary numbers can be stored inside a computer, and how computers can easily store numbers that are very large indeed.



介绍
Introduction

尽管比特是一个非常简单的小东西，非 0 即 1，可是在计算机世界中它却是被广泛应用的重要概念。你一定遇到过类似 "24 位色彩""100 兆连接速度""32 位计算机"或"128 位 SSL 加密"这样的术语吧。正因为比特在计算机技术中是如此常用的度量单位，所以接下来我们将好好一睹它的庐山真面目。

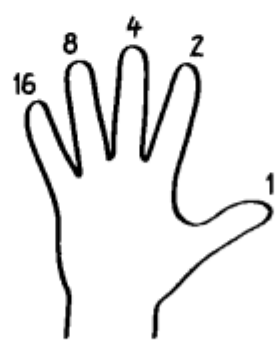
Although a bit is a very simple thing—a zero or one—it is a very widely used concept in computing. You may have come across phrases like "24-bit color", "100 megabit connection", "32-bit computer", or "128 bit SSL encryption". The bit is a very widely used measurement in computing, so we're going to look at it carefully.

首先请记住，一个比特可以也只可以有两个数值：0 或 1，因此任何拥有两种不同状态的东西都能被用来代表比特。比如，电灯的开关可以用来表示比特——点亮的灯代表数值 1；熄灭的灯代表数值 0（如果你非要反过来表示也是行得通的）。而在前一章中，我们使用了许多不同的符号来表示一个比特：勾和叉、向上的箭头和向下的箭头，以及插在生日蛋糕上的蜡烛。

Remember that a single bit can have one of just two values: zero or one. So a bit can be represented by anything that has two different states. For example, you could use a light switch to represent a bit – if the light is on, it represents the value 1; if the light is off, it represents the value 0. (Or the other way round. It doesn't matter so long as you are consistent.) In the previous topic we used all sorts of symbols: ticks and crosses, up and down arrows, and even candles on a birthday cake.

我们也能用手指来表示比特。通常情况下，用双手来数数可以数到 10，但如果让每个指头代表一个比特，你就能数到更大的数字。假设伸直的手指代表 1，弯曲的手指代表 0，那么在二进制系统中，一只手就好比之前小游戏中 5 张画有圆点的卡片，这样单靠一只手便能从 0 数到 31，总共就是 32 个数，不要忘记还有 0 也是一个哦！

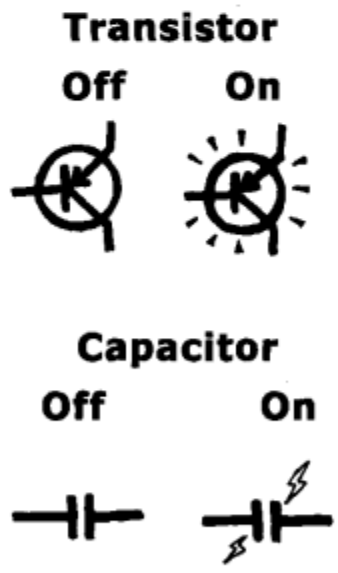
We can also use our fingers to represent bits. Normally people count to ten on the fingers of both hands. But if each finger represents one bit, you can go much higher. If a finger is up it is a one, and if it is down it is a zero. If you use the binary system and let each finger on one hand represent one of the cards with dots, then you can count from 0 to 31. That's 32 numbers. Don't forget that zero is a number too!



下面这张图片代表了二进制数 10011，也就是十进制数 19。

试着用 4 只手指从 0 数到 15，多练习一下便能熟能生巧。

因为设计出仅表示两种数值的硬件设备比较容易，所以计算机采用二进制系统来表达信息。例如，在计算机内存中比特能以晶体管的开或关(分别代表 1 和 0)来表示。有时，它们被储存在一种名为电容器的电子元件中，电容器有充电或放电两种状态，至于用充电代表 0、放电代表 1，还是充电代表 1、放电代表 0 倒无所谓，只要能表达出二进制位就可以了。



For example, the following picture shows the binary number 10011, which is 19 in decimal.

Try counting in order from 0 to 15 using four fingers. It may take some practice before you become good at this.

Computers use the binary system to represent information because it is easy to design electronic devices that can represent two different values. For example, bits can be stored in a computer's memory with a transistor that is switched on or off (for 1 and 0). Sometimes they are stored using an electronic component called a capacitor that is either charged or discharged. The way the bits are stored doesn't matter: it's the bits themselves that count.

术语一点通

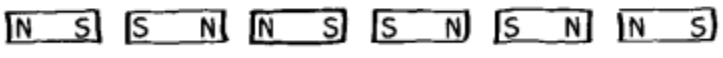
The computer's main memory is often referred to as **RAM**, which stands for "Random Access Memory". This kind of memory uses electronic devices like the transistor and capacitor just described to store data. It is very fast compared with most other ways of storing data, but it is expensive, and when the power is switched off, the data is lost. That's why computer programs usually have a "save" function: usually this is saving the document or data you're working with from the computer's RAM to a hard disk, so that it is stored permanently. The name "Random Access Memory" isn't a very useful description, because almost all kinds of storage can be accessed "randomly" – that is, the computer can start reading from anywhere within the storage device. However, the name made more sense when storing data on tape was common – to read some data on a tape the computer had to read all of the data up that point first!

计算机的内存一般被称为 RAM(随机存取存储器),它采用晶体管和电容器等元件来储存数据。比起其他储存数据的方式,RAM 的优点是数据存取速度快,缺点是价格比较贵,而且一旦断电,数据就会丢失,这也是为什么计算机软件通常都有"保存"的功能,该功能可将用户处理的文档或数据从计算机的内存永久保存到硬盘上。其实随机存储器并不算是个好的名称,因为现在绝大多数储存器都能被"随机"地读取,即计算机能从储存器的任意位置开始读取想要的数据。把"随机存取"这一概念放在过去使用磁带来储存数据的年代会更有意义,那时候,每次需要读取某段数据时,计算机都必须从磁带的开头开始读起,直到获得想要的信息。

比特也能储存在磁盘(如硬盘)和磁带(如数字录影带)中。此时,比特是用磁盘表面涂覆的磁介质的"北-南"或"南-北"状态来表示的。

Bits are also stored on magnetic disks (such as a hard disk) and tapes (such as a digital video tape). Here, bits are represented by the direction of a magnetic field on a coated surface, either North-South or South-North.

a 如果北-南状态代表 1,南-北状态代表 0,那么下面这个磁盘中储存的二进制数是什么呢?



a If North-South represents 1 and South-North represents 0, then what binary number is this disk recording?

也可利用光学原理将数据储存在各类光盘上,此时可用碟片表面反光或不反光这两种状态来表示比特。

Bits are also stored optically on **Audio CDs**, **CD-ROMs** and **DVDs**. The part of the surface corresponding to a bit either does or does not reflect light.

b 下图的 CD 碟片上存储了一个 6 位二进制数。如果吸收光代表 0、反射光代表 1 的话，下图代表的二进制数是多少呢？

b The diagram below shows 6 bits on a CD-ROM. If absorbing light corresponds to a 0, and reflecting light corresponds to a 1, what binary number is this optical disk recording?

小游戏 2.1 二进制数的性质
Activity 2.1 Properties of binary numbers

接下来我们将学习两个十分有用的二进制数性质。如果你仔细观察在第 1 章中我们使用的卡片，你会发现数字之间存在某种有趣的关系。让我们按照顺序依次写下这些数，其间的规律是不是变得明了多了？

1, 2, 4, 8, 16, …

We are now going to learn about two useful properties of binary numbers. If you look carefully at the sequence of cards we used in Topic 1, you can find a very interesting relationship. We've written them here in reverse order to make it clearer:

c 把前两张卡片的数字相加(1+2)，其计算结果是多少？

c Try adding the first two cards, 1+2 . What does it come to?

d 再把前三张卡片的数字相加(1+2+4)，其计算结果是多少？

d Now try adding the first three cards, 1+2+4?

e 那么 1+2+4+8 的计算结果呢？

e What about 1+2+4+8?

f 最后把五张卡片的数字全部加在一起(1+2+4+8+16)，其计算结果等于多少？

f And finally, what if you add all of the cards together 1+2+4+8+16?

g 现在你发现前面卡片的数字之和与后面一张卡片的关系了吗？

g What relationship do you notice between the sum of all of the previous cards and the next larger card?

此外，将 0 插入二进制数的右侧，会发生一些有趣的变化。在十进制系统中，当往数字右侧插入一个 0 时就相当于将数字扩大 10 倍，例如，9 变成 90，30 变成 300。

9 → 90

30 → 300

Another interesting property of binary numbers is what happens when a zero is inserted on the right hand side of the number. If we are working in base-10 (decimal), when you insert a zero on the right hand side of the number, it is multiplied by 10. For example, 9 becomes 90, 30 becomes 300.

可当你向二进制数的右侧添加一个 0 呢？观察下面几组数的变化。

010 → 0100

101 → 1010

110 → 1100

But what happens when you put a 0 on the right of a binary number? Try this:

h 二进制数 010 右侧插入 0 后会变成什么数字？

What number does 010 change to when you put a zero on the right hand side?

i 二进制数 101 右侧插入 0 后会变成什么数字？

What number does 101 change to when you put a zero on the right hand side?

j 二进制数 110 右侧插入 0 后会变成什么数字？

What number does 110 change to when you put a zero on the right hand side?

k 二进制数右侧插入 0 后变成了一个新数字，计算该新数的规律是什么？

What is the rule when you put a 0 after a binary number?

小游戏 2.2　大一点的二进制数
Activity 2.2　Large binary numbers

到目前为止，我们使用的例子均只含有 5 或 6 个数位，即代表了十进制数的 0 到 31 或 0 到 63。在接下来的小游戏中，我们会用到更多的比特数，但在此之前我们先来巩固一下从前一个游戏中得到的技巧，它可以让我们简单地利用比特数计算出能够表示出的最大的二进制数。

So far we've been using groups of up to 5 or 6 bits, which can represent numbers up to 31 or 63 respectively. In this activity we will use even larger groups of bits, but first we will use the trick from the previous activity to make it easy to work out the largest number we can represent with a group of bits.

使用 5 比特最大可以表示 31(即 16+8+4+2+1=31),同样地,你也能用 32 减去 1 的方式来得到这个数字。

The largest value we could represent using 5 bits was 16 + 8 + 4 + 2 + 1 = 31. You can also work out this value by working out what the next card would be (32) and subtracting 1 from it.

l 使用 6 比特最大可以表示多少?

l What is the largest value you could represent with 6 bits?

m 使用 7 比特最大可以表示多少?

m What is the largest value you could represent with 7 bits?

n 那么 8 比特呢?

n What is the largest value you could represent with 8 bits?

术语一点通

It turns out that storing groups of 8 bits is very convenient on modern computers, and it is so common that a group of 8 bits has a name: a byte. The word "byte" came about because that number of bits could be handled by a computer in one "bite". Originally a byte could contain anything from around 5 to 12 bits, but 8 bits became common in the 1980s, and it is the usual size these days.

在实际应用中,计算机系统处理连续存储的 8 位数据是非常便利的,一般将上述连续存储的 8 比特称为一个字节(byte),而在计算机中每次都将这一组比特位一起处理,好像将它们一起咬上一口"bite",并以此为单位"一口一口"处理其余的数据,这就是字节使用"byte"来表示的原因。最开始一个字节包含 5～12 比特,可 80 年代后,8 比特一字节的使用变得越来越普遍,后一直沿用至今。

希望你已经找出上个问题的正确答案:一个字节能表示出数字 0 至 255。计算机当然需要处理大于 255 的数字,这时就会用到更多的字节,如 16 位、24 位、32 位和 64 位。

Hopefully you just worked out in the last question that a byte can represent numbers from 0 to 255. Of course, computers need to work with numbers bigger than 255, so they often work with groups of bytes – 16 bits, 24 bits, 32 bits, and 64 bits are common sizes.

O 使用 16 比特(2 个字节)能表示的最大数值是多少呢?(提示:不要傻傻地将全部数字加起来啦——记住诀窍在于将第 17 张卡片上的圆点数减去 1 便能得到你想要的数字。)

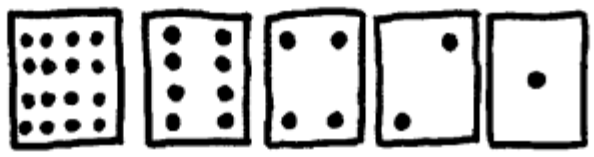
O What is the largest value you could represent with 16 bits (2 bytes)? (Hint: don't try to add up all the values – remember the trick of subtracting one from the number of dots that would be on the next largest card.)

P 用 24 比特(3 个字节)能表示的最大数值是多少呢?

P What is the largest value you could represent with 24 bits (3 bytes)?

对于 5 位二进制数,你靠猜测或逐一累加或许能得到其代表的十进制数。不过在处理更大的二进制数时,我们需要更加有效的计算方法。幸运的是,这里就有一个简单的方法,我们不妨用 5 张卡片来验证一下。

When we had just 5 bits it was easy to work out the binary representation for a decimal number by trial and error, but for larger numbers we need to be a bit more systematic. Fortunately there is a simple way to do it, which we'll demonstrate using just 5 cards.



比如,你正在计算如何得出数字 14 的卡片组合,那么从左侧开始,依次选择每一张卡片。首先问问自己上面画有 16 个点的卡片是不是过大呢?当然太大了,你拿这张画有 16 个圆点的卡片无论如何也表达不出 14!那就把 16 点的卡片放在一边吧。

As an example, suppose you are working out the pattern for the number 14. Start at the left hand side, and ask if the card (16) is too big? In this example it is obviously too big – if you have 16 dots showing then you won't be able to represent 14! So hide the 16 card.

再来看看下一张 8 点的卡片,显然它没有超过目标结果,因此留下这张卡片,并从我们想要得出的结果中减去 8,14 – 8 = 6。然后来看看下一张 4 点的卡片,它比余下的 6 点小,所以保留这张卡片,令其正面朝上。由于 6 – 4 = 2,我们还需要 2 点,所以下面这张 2 点的卡片也留下,令其正面朝上,这时 2 – 2 = 0,我们已经不需要更多点了!而最后一张是 1 点的卡片,不需要,只能让它背面朝上。

Then look at the next card (8). It isn't too big, so leave it showing, and subtract 8 from the total we're aiming for i. e. 14 – 8 = 6. Now consider the 4-card. It is less than the number of remaining dots needed (6), so we can leave it facing up, leaving just 2 more dots to find. The 2-card, is next, and since we need 2 more dots, we leave it face up, giving us 0 more dots to find (!) The 1-card is next, and obviously won't be needed, so turn it face down.

你当然不需要真的准备几张卡片来解决这个问题，而只用提笔写下数字，然后把那些你不需要的数字划掉即可。假设在这里我们使用的是 8 比特来表示数值。

128 64 32 16 8 4 2 1

You don't have to use cards to work this out: you could just write down the number of dots, and cross out the ones you don't want. Suppose we have 8 bits, which represent the values.

这次来试试看用二进制数怎样得到 77。让我们从 128 开始(这个数字太大了，划掉)，接下来是 64(比 77 小，保留，余下的数为 77 − 64 = 13)，接下来是 32(划掉，显然比 13 大)，依此类推。把划掉的数字用 0 来替代，保留的数字用 1 替代，得到下面的二进制数。

~~128~~ 64 ~~32~~ ~~16~~ 8 4 ~~2~~ 1

0 1 0 0 1 1 0 1

To convert the number 77, start with the 128 (which we cross out because it is way too big), then 64 (which we'll accept because it is smaller than 77, leaving 77 − 64 = 13 to find), then 32 (which we cross out because it is larger than the 13 needed), and so on. Then write down a 0 for each number that is crossed out, and a 1 for each one that isn't.

采用这种方法算出以下数值。

Use this method to work out the following values:

q 如何用二进制数来表示 165?

q What is the binary representation of the number 165?

r 如何用二进制数来表示 99?

r What is the binary representation of 99?

s 如何用二进制数来表示 127?

s What is the binary representation of 127?

t 如何用二进制数来表示 1000?（你需要更多的比特位来表示这个数!）

t What is the binary representation of 1000? (You will need some more binary digits for this one!)

千字节(Kilobyte)是另一个计算机中的常用术语，简写成 KB。一般来说"kilo"意味着 1000(比如 1 km 等于 1000 m)，但是在计算机中，一个"kilo"的基数是 1024，这个数字看起来有点奇怪，但是它能帮助你找到下面问题的答案，而且能让你了解到为什么这个数字在计算机中是那么重要。

A term that is common on computers is **kilobyte**, or KB for short. Normally kilo means 1000 (for example, a kilogram is 1000 grams), but on computers a kilo is usually taken to be 1024. This might seem like a strange number, but the answer to the following question might help you to understand why it is a sensible number on computers:

u 如何用二进制数来表示 1024?

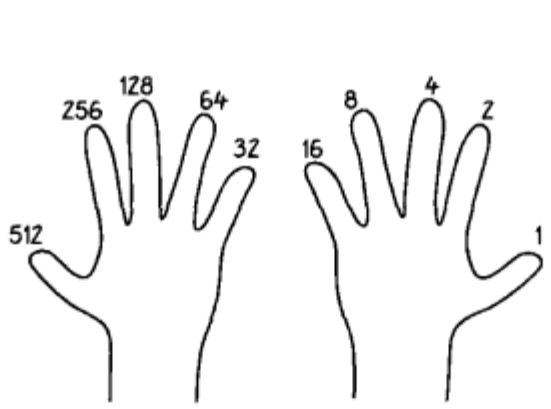
u What is the binary representation of 1024?

1024 对于以二进制储存数据的计算机来说是个非常好用的数字，相比之下你会发现，1000 对应的二进制表示不如 1024 对应的二进制表示一目了然。

The number 1024 is a very comfortable number for computers to work with, since they are storing it in binary. In comparison, the decimal number 1000 isn't so tidy – compare the answers for the previous two questions!

在上一章中，我们用手指代表二进制数数，用一只手可从 0 数到 31，如果用两只手，则可用 10 个比特位继续数到 1023。这样就有 1024 个数字！如果你能弯弯每个脚趾，那么就能一直数到一百万！

In the previous topic we used binary on our fingers to count up to 31 on one hand. If you use two hands, you have 10 bits to count on. That's 1024 numbers! If you had really bendy toes, then you could count even higher, up past 1 million!



（请注意：不是所有人都能像灵活脚趾先生那样能自由摆动自己的脚趾哦）

进阶篇
Details for experts

尽管计算机中所有的数据都以二进制的形式进行储存，但是不同的储存设备有着不同的特性。比如，计算机的内存（RAM）读取数据的速度很快，可是断电时会丢失数据，相较之下，硬盘的数据读取速度要慢得多（尽管硬盘读取数据块的数据速度已经相当快了，但当它读取某一段数据时，其需要花费的时间几乎是内存的一百万倍）。

Although all data is stored using binary numbers, the different kinds of storage devices have quite different characteristics. For example, the computer's main memory (called Random Access Memory, or RAM) is very fast to access, but loses its contents when the power is switched off, whereas a hard disk is much slower at reading its binary data (it could take a million times as long as RAM to find a particular piece of data, although it can read a block of data quite quickly).

刚才谈到千字节(Kilobyte)这个概念,它相当于1024 字节。比它更大的数量级是兆字节(Megabyte),相当于一百万个字节,或者说相当于 1024×1024 字节。这种说法很让人奇怪是不是,因为这两种换算法都广为使用,很难判断人们指的到底是哪一个。而在实际应用中,这两种换算结果相差无几,大多数情况下都不影响计算。

We introduced the idea of a kilobyte, which is usually 1024 bytes. The next larger unit is a **megabyte**, which means a million bytes, or else 1024x1024 bytes. It might seem strange that such a common term isn't well defined, but both definitions are widely used, and it's hard to get people to agree on which one they mean. In practice, the difference isn't too big, so it doesn't matter in most situations.

对储存设备来说,特别是计算机的内存和硬盘,即便兆字节都只是一个小的存储单位。吉字节(Gigabyte)相当于 1000 倍(或 1024 倍)的兆字节,太字节(Terabyte)则相当于 1000 倍的吉字节。有些储存庞大数据的公司或许会使用拍字节(Petabyte)作为计量单位,它相当于 1000 倍的太字节。2T(太字节)的硬盘可以装下相当于两万亿(2000000000000)字节的数据,这才是一组庞大的数据!

Even a megabyte is a relatively small amount of storage, especially for computer memory and disk drives. A **gigabyte** is 1000 (or 1024) megabytes, and a **terabyte** is 1000 gigabytes. Companies that store very large amounts of data might measure it in petabytes, which are 1000 terabytes. So, a 2 terabyte disk can store about 2000000000000 bytes. That's a lot of data!

术语一点通

The terms **megabyte**, **gigabyte**, **terabyte** and so on are not well defined, because sometimes people say that each is 1024 times larger than the previous one, and other times they are 1000 times larger. There is a different set of official terms when data is measured in multiples of 1024: for example, the kibibyte, which is 1024 bytes. Kibibyte is short for "kilo binary byte". The larger units are mebibyte (1024 kibibytes), gibibyte (1024 mebibytes), tebibyte (1024 gibibytes), and so on. However, you will hardly ever see these terms used, and in practice people get used to the idea that units like "gigabyte" aren't precise.

人们并未对兆字节、吉字节、太字节等术语作严格的定义,相邻两量级之间的倍数很多时候是混用的,有时候用 1000,有时候用 1024。而当专门指 1024 的倍数的字节数时,有另一组官方的术语来表示,例如 kibibyte 指 1024 字节,更大的量级还有 mebibyte(1024 kibibytes),gibibyte(1024 mebibytes),tebibyte(1024 gibibytes)等。然而,在实际应用中,上述术语很少使用,人们已经习惯使用"gigabyte"这组以千字节(1000 Byte)估算的单位。

一个字节相当于 8 比特，它可以表示 256 个不同的数值(从 0 到 255)。一般来说，如果你用 k 比特可表示的最大数是 $2^k - 1$，可表示的最小数是 0，也就是可以表达一共 2^k 个不同的数值。例如，10 比特的二进制数能表示 1024 个不同数值，而 16 比特的二进制数能表示 65536 个不同数值。

A byte is 8 bits, and we observed that each byte can represent one of 256 different values (0 to 255). In general, if you are storing a number in k bits then the maximum number that can be represented is $2^k - 1$. Since the minimum number is 0, this means there are 2^k different values. For example, 10-bit numbers have 1024 different values, whereas 16-bit numbers have 65,536 different values.

现在你也许能更好地理解本章开头部分提到的一些术语了。

This should help you to understand some of the terms mentioned at the beginning of this topic:

"24 位色彩"是指一张图片在计算机中可用高达 $2^{24} = 16777216$ 个不同色彩元来显示或存储。这是相当高清晰的画面，其实人眼是无法识别这么多的色彩元的。

"24-bit color" means that a picture on the computer is being displayed or stored using up to $2^{24} = 16777216$ different colors. That means it is very high quality since the human eye can't distinguish this many colours!

"100 兆连接"是指计算机和网络之间的连接速度能达到每秒 100 兆比特，这意味着一眨眼工夫便能完全下载一个超大的文件了。

"100 megabit connection" means that the computer's connection to the network can send up to 100 million bits in each *second*. That means that large files can be transferred in a fraction of a second.

"32 位计算机"是指能一次性储存或运算 32 比特的计算机。处理相同的数据，一台 8 位计算机(这种计算机往往被用来实现一些简单操作，例如报警器)至少需要耗费相当于 32 位计算机 4 倍的时间才能完成。所以计算机处理器的位数在很大程度上影响了计算机的性能。对于个人计算机而言，32 位的处理器意味着它能同时引用内存中 32 位的数据，即处理器最多可以直接处理内存中 $2^{32} = 4294967296$ 字节的内容。看起来这些数据很庞大，其实仅仅只有 4G 字节而已，而现在个人计算机中配备 4G 以上的内存已经不稀奇了。

"32-bit computer" means that the computer stores and processes 32 bits of data at a time. In contrast, an 8-bit computer (used for simple controllers in devices like house alarms) would need to do at least four times as much work to process 32 bits of data, so the number of bits has a significant implication for the speed of the computer. On a personal computer, being a 32-bit processor can also imply that it uses up to 32 bits to refer to locations in the memory, which means that the largest amount of memory it can deal with directly will have $2^{32} = 4294967296$ bytes. That seems like a lot, but it's only 4 gigabytes, and it's not unusual to have more than 4 gigabytes of memory in a personal computer.

"128 位 SSL 加密"是指用 128 位密匙为数据加密。当用户上网浏览一些加密链接时该密匙会自动加载,如使用网上银行或信用卡付款时。128 位密码的运行机理类似暗码锁上的数字,如果你只设定了 8 位密码,意味着别人仅尝试 256 次不同组合便能入侵到你的系统,这太不安全了。相反地,如果加密系统是基于 128 位的,那么攻击者大概要尝试 2^{128} 即 340282366920938463463374607431768211456 次才能试出密码。一旦走上这条不归之路,就算是用世界上最高端的计算机,大概算几个世纪都得不出密码组合吧。

"128 bit SSL encryption" means that the data is being protected by a secret 128-bit number which is like a password. For example, this happens automatically when a web browser goes into a secure connection, perhaps for on-line banking or a credit card payment. The 128-bit number acts like the numbers on a combination lock. If you only had an 8-bit number, an attacker would only need to try out 256 different combinations to break into the system. This is not at all secure, and in contrast, if the security is based on a 128-bit number then there are 2^{128}, or 340282366920938463463374607431768211456 different values to try out. That would take way too long—even a very fast computer could try these out for many centuries, and still hardly have got started.

除了提到的十进制和二进制之外,人们有时候也会用到其他进制。例如,在八进制(octal)系统中,每个数位相当于前一个数位的 8 倍,能获取的数值范围从 0 到 7。八进制中一个数位相当于二进制中的 3 个数位,基于这种规律,我们能很方便地对二进制和八进制进行换算。由于二进制的数位长度相当于在八进制中的 3 倍,有时候人们会用八进制作为二进制的缩写。例如,八进制数 741 就相当于二进制数 111100001。

People use other bases as well as base-10 and base-2 that we've discussed so far. For example, in base-8 (octal), each digit is worth 8 times as much as the previous one, and the digits used range from 0 to 7. Each digit in octal corresponds exactly to 3 bits in binary, so it is very easy to convert between binary and octal. Because numbers in binary are three times longer to write down than numbers in octal, sometimes people use octal as a shorthand for binary. For example, the octal number 741 is the same as the binary number 111100001.

你或许还在计算机中见过十六进制(hexadecimal,其中 hex 代表六,decimal 代表十)。因为十六进制数用到了 16 个不同的基数,所以在 9 之后又添加了 A、B、C、D、E 和 F 几个符号来表示。十六进制中的一个数位相当于二进制中的 4 个数位,所以有时候也会把十六进制作为二进制的缩写。如果对十六进制感兴趣的话,不妨找一本百科全书读读相关内容吧。

You may even come across base-16 on computers. It is called hexadecimal (hex is 6, and decimal is 10, so hexadecimal is 16). There are 16 different digits, so after 9 people usually use A, B, C, D, E and F. Because each hexadecimal digit corresponds to exactly 4 binary digits, it is also often used as shorthand for binary numbers. You can find out more about hexadecimal numbers by looking them up in an encyclopedia.

有趣的事
Curiosity

教你一个读心术的小魔术，你可以用二进制数来读出别人在想什么。让你的朋友在 0 到 63 之间选一个数字（比如他们出生的那一天），然后给他们看看下面的 6 张卡片，让他们选择出有那个数字的所有卡片，然后把这张（些）卡片交给你。

There is a card trick that uses binary numbers to make it appear as though you can read someone's mind. Simply get a friend to think of a number from 0 to 63 (e.g. the day of the month they were born on), show them the following 6 cards, and ask them to give you the cards that contain their number.

32	33	34	35
36	37	38	39
40	41	42	43
44	45	46	47
48	49	50	51
52	53	54	55
56	57	58	59
60	61	62	63

16	17	18	19
20	21	22	23
24	25	26	27
28	29	30	31
48	49	50	51
52	53	54	55
56	57	58	59
60	61	62	63

8	9	10	11
12	13	14	15
24	25	26	27
28	29	30	31
40	41	42	43
44	45	46	47
56	57	58	59
60	61	62	63

4	5	6	7
12	13	14	15
20	21	22	23
28	29	30	31
36	37	38	39
44	45	46	47
52	53	54	55
60	61	62	63

2	3	6	7
10	11	14	15
18	19	22	23
26	27	30	31
34	35	38	39
42	43	46	47
50	51	54	55
58	59	62	63

1	3	5	7
9	11	13	15
17	19	21	23
25	27	29	31
33	35	37	39
41	43	45	47
49	51	53	55
57	59	61	63

只要简单地把他们选择的各张卡片中第一个数相加，你便能"猜"出他选择的秘密数字是多少了。比如，如果他们选择的秘密数字为 17，他们会交给你以 16 和 1 开头的这两张卡片。因为只有这两张卡片上写着 17，而你只用把这两张卡片上的第一个数字相加，即 16 + 1，便能简单得出他们选中的秘密数字了。

You can then tell them what their secret number is by adding up the first number from each card they select. For example, if they think of the number 17, they will give you just the two cards starting with 16 and 1, since these are the only cards containing a 17. You just need to add the first number from those two cards, 16 + 1, and you have their secret number!

你能解开这个魔术背后的秘密吗？（提示：你的伙伴挑选的卡片透露了秘密数字的二进制读法——试着写下每张卡片开头几个数字的 6 位二进制表示形式）

See if you can figure out how this trick works. (Hint: the person is actually telling you the binary representation of their secret number – try writing the 6-bit binary representation for the first few numbers on each card.)

第3章

从比特到字母

0

From Bits to Letters

1

前面提到过，计算机中的所有数据都以二进制的形式储存，这当然包括文本编辑器和网页中显示的所有字母、文字和段落。它们的外观及属性与数字完全不同，但是它们的本质还是数字。上一章向大家介绍了如何使用比特来表示数字，现在让我们看看如何用比特表示文档中的字母和文字。

As we have learned, everything in a computer is stored as binary numbers. That includes all the letters and words and paragraphs in word processor files and web pages, and everything else too. They don't look like numbers, and they don't behave like numbers, but deep inside that's what they are. The previous topics explained the binary system and demonstrated how we can store numbers using bits. Now we look at how to represent letters and words in a document using these bits.

介绍 Introduction

在前面几章中，我们学习了二进制系统，以及计算机如何储存比特(二进制单位)并用它们来组成大数字。下面我们来看看计算机如何使用比特来实现字母和文字的表达。

In our previous lessons, we learned about the binary system and how computers store bits (binary digits) and build them up into large numbers. Here we will learn how computers can use bits to form letters and words.

其实，你在计算机中看到或输入的任何文本都是用0和1来表达的。毕竟，任何与计算机连接的储存设备，硬盘也好，光驱也好，移动设备也好，使用的都是二进制系统。而在计算机间互传的文档、电子邮件以及浏览的网页甚至网络上所有的一切：包括图片、音乐和视频，同样也是采用二进制数表达的。那么到底计算机是如何用比特来表达字母和文字的呢?

Every piece of text that you see or type on a computer is represented using zeros and ones. After all, everything attached to a computer that stores data, such as hard disks, CD-ROMs and flash drives, uses the binary system. And the same is true when text is sent from one computer to another as email or when you're browsing a web page. Everything that goes over the Internet—including pictures, and music, and videos—is also represented using just binary digits. So how can we represent letters and words using only bits?

在前面两章中我们已经学会了用一组比特来表示任意的数。现在为了表示汉语的拼音系统，就需要用到4个声调和26个字母，一共30个元素，我们可以像下面列出的这样用一个数字代表一个声调或字母。

From the previous topics you already know how a computer can represent any number—you just use groups of bits (zeros and ones). To represent words in Pinyin, we need to have codes for the four tones and the 26 letters. That's 30 codes in all. We can just use a number for each one, like this:

1	2	3	4	5	6	7	8	9	10
—	／	∨	＼	a	b	c	d	e	f
11	12	13	14	15	16	17	18	19	20
g	h	i	j	k	l	m	n	o	p
21	22	23	24	25	26	27	28	29	30
q	r	s	t	u	ü	w	x	y	z

例如，用下面的代码能表达出"你好"(nǐ hǎo)。

For example, using this code we can represent nǐ hǎo (ni3 hao3) as follows:

18	13	3	0	12	5	19	3
n	i	3		h	a	o	3

等等,那文字之间的空格怎么办? 实际上我们可以用 0 来表示空格,所以最终一共是 31 个元素。

用预定的方法将供人们阅读的信息转换成计算机可以读取的信息称为"编码"(encode),而将二进制信息转换成可供人们阅读的信息的过程称为"解码"(decode)。

a 使用上表中的十进制编码计算机如何表示字母 d?

b 如何表示字母 y 呢?

c 上表中数字 30 代表的字母是什么?

d 使用上表中的十进制编码如何表示单词"hello"?

e 对数字序列"17 5 3"进行解码能得到什么?

f 采用上述编码方式,你的名字如何表示呢?

在上述的字母编码表中,每个数字都能仅用 5 个比特来表示(事实上,我们在第 1 章的学习中已经了解到,用 5 比特能一直数到 31)。数字 0 可以代表字母之间的空格,例如,上面问题中提到的代码"17 5 3"能表示成如下二进制数:

10001 00101 00011

Wait a minute! What about the space between the words? We have used a zero for this—in fact, there are 31 codes in total, not 30. Don't forget the space!

We use the term "**encode**" when the kind of message that people read is changed into something the computer can use, like binary digits. We use the term "**decode**" when the binary digits are changed back into a human readable message.

a Using this simple decimal code, how would the computer encode the letter d?

b The letter y?

c What letter would the number 30 represent?

d How would the computer encode the word "hello"?

e If you decode "17 5 3", what would you get?

f How would the computer encode your name in decimal numbers?

Each number in this simple alphabet code can be represented using just five bits (in fact, we saw in Topic 1 that with 5 bits you can count up to the number 31). We can use the number 0 to represent a space between words. For example, the code "17 5 3" in the question above could be represented by three binary numbers:

10001 00101 00011

这就是怎样只用 0 和 1 来表示一个单词的好例子。但是上述编码比计算机中实际使用的要简单一些，因为实际应用中需要区分字母的大小写，还需表示数字、标点和一些特殊符号，比如“$”、“～”。现在在计算机中广泛使用的最简单的信息交换码称为 ASCII[4]，ASCII 中每个字符用 7 比特来表示，一共可以表示 128 个不同的字符，这对于英文来说已经足够了。对于拥有更大字符容量的字库（比如中文），计算机使用 Unicode 系统[5]，其中每个字符用 16 比特来表示。

That's a whole word represented using just zeros and ones. This code is simpler than the one that is used by regular computers, because they must store whether letters are capitals or not, and also recognize digits, punctuation and special symbols such as $ or ～. The simplest code used today to represent letters on computers is called **ASCII**, which is based on using seven bits per character. This allows for 128 different characters, which is enough for English text. For larger character sets (such as the Chinese characters), computers use the **Unicode** system, which is based on 16 bits per character.

用 16 比特来编码一共能表示多少个不同的字符？

How many different characters could you represent with 16 bit numbers?

术语一点通

ASCII stands for American Standard Code for Information Interchange. Although it is “American”, it is used all around the world, and still appears as a subset of the codes used for languages like Chinese. The ASCII codes are used in a variety of places on a computer. In a **word processor** file the characters are stored using this code (along with extra codes to specify the formatting), and when you download a web page, each character in the HTML file is sent as one of these codes. Although ASCII codes only require 7 bits per character, they are usually stored with one character per byte, that is, each character takes up 8 bits, with one being unused.

ASCII 的全称是美国信息交换标准编码(American Standard Code for Information Interchange)，它适用于世界范围的编码系统，在中文这样的编码集合中它则以编码子集的形式出现。ASCII 码在计算机信息编码中被广泛使用，在文字处理软件中，每个字符以 ASCII 码的形式储存在文件中(同时也有专门用于定义格式的编码)，所以当你下载网页时，HTML 源代码中的每个字符就是以这些编码传送到你的计算机。尽管 ASCII 码中每个字符的表示仅使用了 7 比特，但通常一个字符被保存为一个字节，也就是占用了 8 比特，其中一位是未被使用的。

[4] 译者注：ASCII 码的全称为美国信息交换标准代码。

[5] 译者注：Unicode 码是基于通用字符集(Universal Character Set)的标准发展而来，又称为统一码、万国码。

小游戏 3.1 储藏室谜题

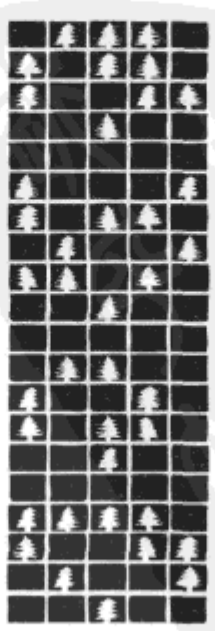
Activity 3.1 The department store puzzle

汤姆被困在一个储藏室的顶楼。快到圣诞节了，他很想快点回到家和父母团聚。他需要找出逃出这里的办法。他试过打电话或大声呼救，但是没有人在旁边，谁也帮不了他。这时他注意到街的对面有一个人深夜依旧在计算机前工作着，汤姆要怎样才能发出求救信号给这个人呢？他环顾四周想找到自己能利用的道具，脑海中突然闪现出一个很棒的点子——用圣诞树的装饰灯来给这位女士发信号！他把所有能找到的彩灯连在一起插上电源以便他控制开关。然后他用到了一个简单的二进制代码，对街的女士一定能明白的代码。

Tom is trapped on the top floor of a department store. It's just before Christmas and he wants to get home to see his presents. He needs to work out what to do! He has tried calling, even yelling, but there is no one around. Across the street he can see a computer person still working away late into the night. How can he get a message to her? Tom looks around to see what he could use. Then he has a brilliant idea—he can use the Christmas tree lights to send her a message! He finds all the lights and plugs them in so he can turn them on and off. He uses a simple binary code, which he knows the woman across the street is sure to understand.

h 在下图圣诞彩灯的闪烁规律中，你能想出汤姆发出的信号是什么吗？

h Can you work out what message Tom is sending from the patterns of Christmas trees on the right?



小游戏 3.2　制作属于你自己的信息
ActiActivity 3.2　Make your own message

现在轮到你来编出一个只属于你自己的信息密码了。你可以使用下表中的空格或画一个类似的表格来完成这件事，首先写下需要编码的文字，然后将这些文字转换成十进制数字，接着转成二进制数字，最后把二进制数字写在一张空白的纸上交给你的朋友，看看他们能否解开你的密文！下表中第一行给出了一个编码的例子。

Now it is your turn to construct, or encode, a secret message. Use the space below or make a similar table to write a few words. Then transform the letters into decimal digits and then into binary digits. Finally, copy the binary digits to a blank sheet of paper and give it to a friend to see if they can decode your secret message! The first line shows an example.

Letter	Decimal Number	Binary Number
o	7	00111

小游戏 3.3 传音游戏

Activity 3.3 Sending messages using sound

二进制系统可以用来表示计算机中所有的数据，所以当计算机通过调制解调器(modem)连接到网络时也是用二进制数来发送信息的。电话系统是用来承载音频信号的，它使用蜂鸣声来表达传送的声音信息。我们可以用高音调的蜂鸣声表示 1，低音调的蜂鸣声表示 0。

The binary number system is used to represent all data on a computer—even computers connected to the Internet through a **modem** use the binary number system to send information. Because the telephone system is designed to carry sound, the system uses beeps to represent the information being sent. We can use a high-pitched beep for a one and a low-pitched beep for a zero.

那么怎样用蜂鸣声来传送二进制数 0111?

How would you send the binary number 0111 using beeps?

十进制数 9 又该怎样传送呢？(你需要将它先转成二进制数，然后再转换成蜂鸣声!)

How would you send the decimal number 9? (You'll need to convert it to binary, and then convert it to beeps!)

在调制解调器中，表示信息的音频以非常快的速度传送，如果你能听到的话，那将是持续的啸叫声，如果你使用过传真机，应该听过这种恐怖的声音——大概每秒蜂鸣数千下!

On a computer's modem these tones go very fast—so fast, in fact, that if you can hear them you get a horrible continuous screeching sound. You can hear this sound if you accidentally dial up a **fax machine**—it is the sound of thousands of beeps every second!

术语一点通

Turning bits into sound is called **modulation**, and converting the sound back into bits is called **demodulation**. So the device that connects a computer to the phone line is called a **modulator-demodulator**, or modem for short.

将比特转换成声音的过程称为调制(modulation)，而将声音转换回比特的过程称为解调(demodulation)。所以连接计算机和电话线的设备又被称为调制解调器，或简称猫(modem)。

现在，我们能用类似 modem 的声音编码来发送信息啦。你已经学过如何用 0 和 1 来表示文字，现在你需要做的就是以高音调的蜂鸣声表示 1，以低音调的蜂鸣声表示 0，然后就可以向你的朋友“唱”出你想要传送的信息了。当然，你不用像计算机那样高速地“唱”出这些蜂鸣声，你可以慢一点表达。

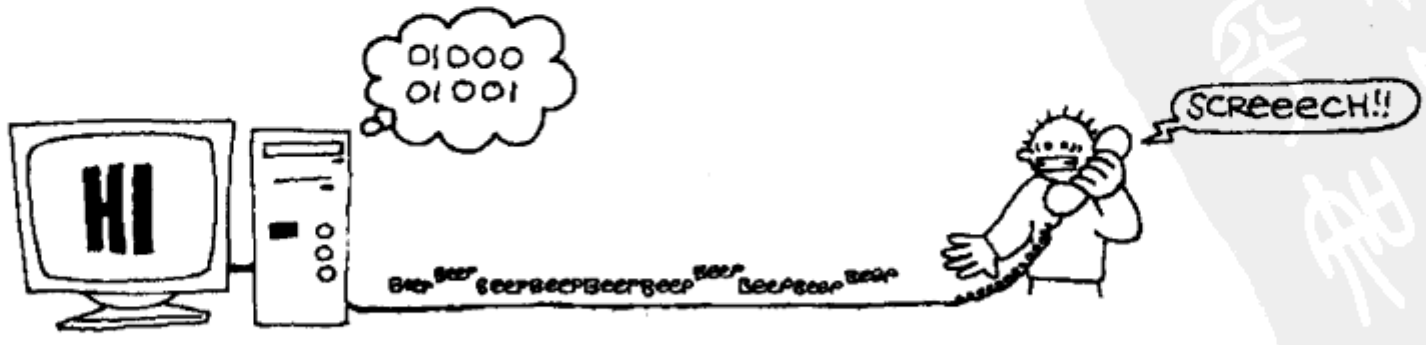
We can use a modem sound code to send messages. You already know how to represent words using just zeros and ones, so all you need to do is send the message to a friend by singing a high beep sound for a one, and a low sound for a zero. You'll need to make the beeps a lot slower than a computer does!

怎样用蜂鸣声来传送字母“q”？

How would you send the letter “q” using beeps?

用这种方法试着发一份电子邮件给你的朋友。

Using this method, try sending an email message to your friend.



进阶篇
Details for experts

前面提到计算机使用 ASCII 码(一个字符 7 或 8 比特)或 Unicode 码(一个字符 16 比特)来存储字符,实际上这些字符将最终组织成文件的形式保存在计算机中。文件是存储在一起的一组字符的集合,例如 word 文档、网页上的文本文件或电子邮件中的正文。由于一个 ASCII 字符占用一个字节(8 比特),所以一份包含 1000 个字符的电子邮件会用到 1000 字节的储存空间(也就是 1KB)。你当然可以说它占用了 8000 个比特,不过通常我们用字节而不是比特来形容磁盘或内存上的储存空间。你会发现计算机中显示文件的长度信息时往往都是用字节(byte)、千字节(kilobyte)、兆字节(megabyte)或吉字节(gigabyte)来表示的。

We mentioned that characters on computers are usually stored using ASCII (7 or 8 bits) or Unicode (16 bits per character). The characters are stored in a **file** on the computer—a file is just a whole lot of characters stored together, and might be a word processor document, the text on a web page, or the content of an email. Since an ASCII character can be stored in **1 byte** (8 bits), a simple email with 1000 characters in it will take about 1000 bytes of storage space (about 1 kilobyte). You could also say that it takes 8000 bits of storage, but normally the space taken on disks and in memory is measured in bytes, not bits. When you list the files on a computer, its length is normally shown in bytes, kilobytes, megabytes or gigabytes.

刚才我们试着用两种声调的音频来表示二进制数,事实上现代调制解调器往往使用多于两种声调的音频来表示信息。例如,使用四种声调可以给 00、01、10 和 11 四个二进制数编码,即每个声调发送了 2 比特,这让调制解调器的速度提高了两倍。为了让调制解调器更快速,现在多数调制解调器(如 DSL)使用了人耳根本听不到的高频,这意味着当你把电话当作 modem 时,还可以同时用它来通话聊天。

Although we used just high and low sounds to represent binary numbers, modern modems usually use more than two different tones. For example, if you have four different tones then you could use the tones to code 00, 01, 10 and 11 respectively, which means that you are sending two bits for every tone, making the modem twice as fast. To make modems go as fast as possible, most modern modems (e.g. DSL) use sounds that are so high that you can't hear them, which means that you can have a voice conversation on the telephone even though it is also being used as a modem at the same time.

有趣的事
Curiosity

二进制位只有两个不同的数值，它们表达起来相当方便。下图中的女孩戴着一条由不同颜色珠子串起来的手链，两种颜色的珠子各代表 0 和 1。基于二进制系统编码这条珠链透露了她的名字，你能猜出她的名字吗？不过这次我们不会告诉你哪个颜色对应 1，哪个颜色对应 0，你必须依靠自己来解决这个难题。（提示：把两种方式都试一遍，看哪一种结果更加合理！）

Because binary numbers only have two different digits, they are very easy to represent. The girl in the picture below is wearing a bracelet made of different colored beads. There are only two different colors, representing 0 and 1. And guess what—the beads spell out her name using the binary system. Can you work out what her name is? But we're not telling you which color is which: you have to figure that out for yourself. (Hint: try it both ways and see which message makes more sense!)

你也可以制作代表自己名字的手链——但是如果你的名字太长，那就做成项链吧！

You could make your own bracelet—or if you have a long name you might want to make a necklace!

下图中有另外一个女孩的一条手链，这次代表她的英文名字，这条手链编码的规律为 A＝1，B＝2，……依此类推。你能算出这个代码表示的名字吗？

Here's another girl wearing a bracelet, but this time she has an English name, and the alphabet code for her bracelet uses A = 1, B = 2 and so on. Can you work out the code?

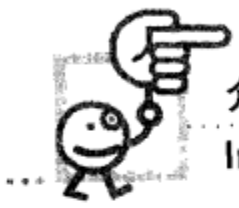
第4章

从比特到图像

0 From Bits to Pictures 1

我们已经了解到计算机如何用二进制位来表示数字、字母和词语。事实上，计算机在储存图画、照片和其他类型的图像时也仅用到0和1。这一章我们要来仔细研究一下计算机是如何做到这一点的。

We have seen how computers can store numbers, letters, and words using just binary digits-bits. In fact, they store drawings, photographs and all other pictures using only the two bits 0, and 1. These activities introduce how they can do this.



介绍
Introduction

通过前三章的学习，我们知道计算机是如何储存数字、字母和文本信息的。然而在多媒体系统中，计算机需要具备储存并显示图像的功能。

In the previous topics we learned how a computer can store numbers, letters and messages. On multimedia systems we also need to store and display pictures.

a 计算机需要储存或发送各种不同类型的图像，你能列举出多少种日常应用中的例子？

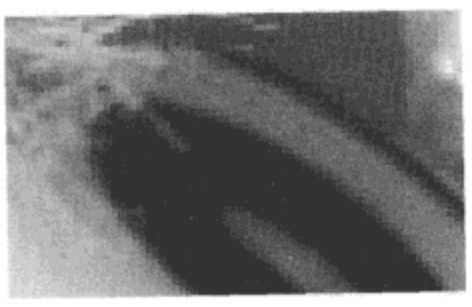
a How many examples can you list of different kinds of pictures that a computer needs to store or send?

我们先来看看计算机是如何在屏幕上显示图像的。下图是鹦鹉木偶阿诺德的照片，照片看起来细腻平滑，可是……

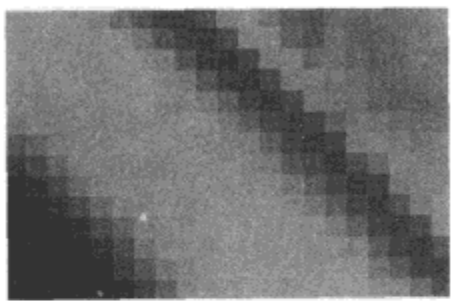
Let's begin by looking at how a computer displays pictures on its screen. The picture on the left shows Arnold the Wonder Parrot. This photo looks like it contains smooth variations in shading, but . . .

……当我们放大它眼睛的部分，却发现细节过渡其实并没有那么平滑……

. . . if we zoom in on his eye, we can see that it isn't completely smooth . . .



……当我们将图片放得更大时，你会注意到，图片其实是由一大堆小方块拼成的，它们被称之为像素(pixel)。在计算机显示器或打印机中，人的肉眼并不能看见如此微小的像素。正因为它们是那么的小，一张图片往往需要数以千计的像素构成，而一张数码照片则往往需要用到高达数百万个像素来表示(兆像素也就是一百万个像素)。



... and if we zoom in even more, we can see that the image is made up of lots of small squares. These are called **pixels**. On a computer screen or printer these pixels are usually so small that you can hardly see them. And because they are so small, we need many thousands of pixels to represent an image. In fact, digital photos use millions of pixels (a megapixel is a million pixels).

术语一点通

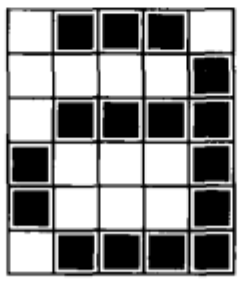
The tiny squares in an image were originally referred to as picture elements. To make it easier to say, people abbreviated it by using "pix" instead of "picture." Combined with the first two letters of "elements" that makes pixel.

最初，图像中的小方块被称为图像元素(picture elements)。为了让这个名称更便于发音，人们用"pix"表示"picture"的缩写，并结合了元素"elements"开头的两个字母，便组成了新名词"pixel"。

在最简单的黑白图像中，每个像素只有两种值：黑或白，比如我们看到的本页上的文字。下图是从页面的局部挑出来的字母"a"的放大图，从这张放大图中，你可以很清晰地看到它是如何用像素来表示的。当计算机储存这样的图片时，它只需记录图像中哪些是黑点哪些是白点，也就是比特的组合。

In the simplest kind of picture, each pixel is either black or white. For example, suppose we look at a page of text. The image below shows an enlargement of a single letter "a" from just a very small part of the page. The letter "a" has been magnified to show the pixels that would be used to display it on a screen. When a computer stores this kind of picture, all it needs to store is which dots are black and which are white. It's just a bunch of bits.

下图中给出了一种存储字母"a"的简单方式,即每个像素用一比特来代替的话:如果是白色像素,计算机将其存为 0;如果是黑色像素,计算机将其存为 1。图中的第一行依次包含 1 个白色像素、3 个黑色像素和 1 个白色像素,因此这一行编码为 0,1,1,1,0。第二行依次包含了4 个白色像素和 1 个黑色像素,因此编码为 0,0,0,0,1。



0,1,1,1,0
0,0,0,0,1
?,?,?,?,?
1,0,0,0,1
?,?,?,?,?
?,?,?,?,?

b 计算机将如何表示第三行呢?

c 第五行呢?

d 最后一行呢?

第 3 章我们学习了如何对字母进行编码,发现仅用一个二进制数字 5 便能表达字母"a"。但这样的编码方式并没有记录字母的形状(shape),解码器接收到编码 5 的时候,还需要知道该如何画出"a"。如果采用图像的形式来表示字母的形状,表达的方法就截然不同了,需要用到的比特数也将大大增加。这样一来,计算机才能在接收到"a"的编码(好比数字 5)后在屏幕上显示或打印出对应的字母来。

The diagram below shows how a picture of the letter "a" can be represented in a very simple way. With this approach, every pixel is represented with a single bit: if the pixel is white, the computer stores a 0 for that bit; if the pixel is black, the computer stores a 1 for that bit. The first line consists of one white pixel, then three black pixels, then one white pixel. Thus the first line is encoded as 0, 1, 1, 1, 0. The second line has 4 white pixels and then 1 black pixel, so this is encoded as 0,0,0,0,1.

b How would a computer represent the third line using this method?

c How would a computer represent the fifth line using this method?

d How would a computer represent the last line using this method?

We already learned how to encode letters in the previous Topic. There the code for the letter "a" was a single binary number, 5 in fact. But that does not encode the shape of the letter: the decoder has to know how to draw an "a" when it receives the code 5. Here we are representing the shape of the letter, as though it was a picture. That's very different. And it needs a lot more bits. The computer would use this information if it received the code for an "a" (like the number 5), and had to display it on a screen or print it.

同样的方法可以用来储存更复杂的黑白图像。例如,可以将写满文字的页面视为一张大图片,问题是存储这张大图片时需要用到许多比特,因为每个像素必须用一个比特来表示。占用太多比特可不是件好事,如果我们想在网络中传送图片,这样便会浪费太多时间。不仅如此,我们还需要购买更大的内存和硬盘来存放这些图片。好在人们已经发明了许多不同的方法来减少储存或发送图片需要占用的体积,减小文件体积的过程称为压缩(compression)。

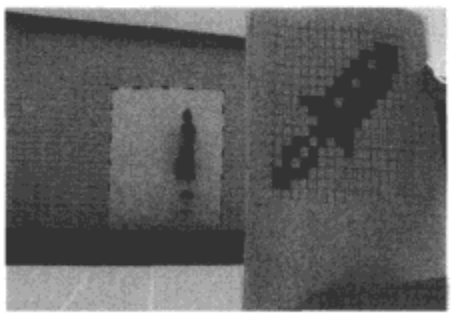
The same approach can be used to store complete black and white pictures, no matter how large. For example, a whole page of text is just a single very big image. But the problem is that a lot of bits are used to store a large picture: we need one bit for every pixel in the picture. Using lots of bits is bad because if we send the picture over the web, it can take a long time. Also we may need to buy more memory or disk space for the computer to store the pictures. Fortunately, people have invented different ways to reduce the number of bits needed to store or send a picture. Reducing the size of a file is called compression.

接下来介绍的方法很适合用来发送文本页面(事实上,传真机采用的就是同样的原理)。这种方法利用了图像中有大块连续的白色像素以及连续的黑色像素的特点,所以只须记录下每个白色或黑色像素连续区块的长度,这种方式被称为游程压缩(run-length compression),或游程编码(run-length encoding, RLE)。

The method that we will use in this activity is very good for sending pages of text (in fact, it's used on fax machines). It uses the fact that often many white pixels are next to each other, and many black pixels are next to each other. It is called **run-length compression**, or run-length encoding, and the idea is to write down how long each run of white or black pixels is.

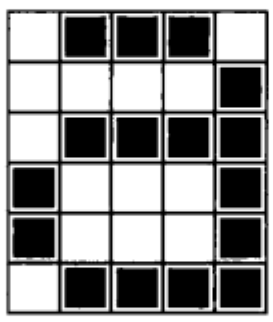
视频:请观看图像压缩游戏的录像。

Video: Watch the video "Image Compression".



让我们回头看看字母"a"的图像,第一行包含 1 个白色像素,接着 3 个黑色像素,接着 1 个白色像素,因此计算机使用 RLE 方法压缩时,第一行会被编码为 1,3,1。同理第二行可简单表示为 4,1。

Going back to our picture of the letter "a", with run-length compression the computer encodes the first line as simply 1, 3, 1, since it has 1 white pixel, then 3 black, then 1 white. The second line is represented simply as 4,1.



1,3,1
4,1
?,?
0,1,3,1
?,?,?,?
?,?

e 采用这种方式，计算机将如何编码第三行呢？

e How would a computer encode the third line using this method?

第四行的编码是0,1,3,1。为什么这里出现一个0呢？因为在这一行并不是以白色像素打头的——如果编码成1,3,1,结果就变成了1个白色像素紧接着3个黑色像素了。

The fourth line is encoded with 0, 1, 3, 1. It begins with a zero because there are zero white pixels at the start of the line—if it was just 1, 3, 1 then it would mean 1 white pixel followed by three black.

f 第五行的编码是什么？

f How would a computer encode the fifth line using this method?

g 计算机将如何编码最后一行呢？

g Finally, how would a computer encode the last line using this method?

游程压缩法中存在的一个问题是：由于游程的长度在计算机中也是采用二进制数表示的，因此能够表达出的游程长度值存在一个上限。例如，如果计算机中设定用5比特表示游程长度，那么这个长度值就无法超过31。

One complication with run-length compression is that there is usually a limit to the length of a run of pixels because the length is being represented as a binary number. For example, if the computer is using five bits to represent each run, then the computer can only record runs up to length 31.

h 如果可用的最大值为31，那么怎样来表达一段长为36的连续黑色像素呢？（提示：如果我们把这个黑色像素段拆成两段的话……）

h If you could only use numbers up to 31, how could you represent a run of 36 black pixels? (Hint: How can it be broken into two smaller runs of black pixels?)

到目前为止，我们见到的图像体积都很小，然而计算机中实际储存的图像往往包含了数百万像素，因此对这些图像进行压缩是非常必要的。

传真机可以通过电话线从一个地方向另一个地方传送纸质文件。它相当于一台简单的计算机，先将一张白底黑字的文稿扫描成 1000×2000 像素的图像，总共包含大约两百万像素。接着，传真机将这些像素通过调制解调器发送给另一台传真机，而接收端的传真机将这些像素再打印到纸张上。传真的图像一般会被压缩为原体积的 1/7 大小。也就是说，如果传送图像时传真机不进行压缩的话，将耗费 7 倍的时间来完成这项工作。

你认为传真机能表示的白色像素段最长能有多长？

我们在这里使用的方法仅表示出黑白图像，那么如何表示彩色图像呢？

The pictures we've looked at so far have been very small, but pictures on a computer can contain many millions of pixels. So, compressing pictures on a computer is very important.

A fax machine is a device that you can use to send a paper document from one place to another over phone lines. It is really just a simple computer that scans a black and white page into a grid of about 1000 × 2000 pixels, for a total of 2 000 000 pixels. One fax machine then sends these pixels to another fax machine using a modem, and the receiving fax machine prints the pixels out on a page. Fax images are generally compressed to about a seventh of their original size. If fax machines didn't use compression it would take seven times longer to send pages!

How long do you think the longest runs of white pixels would normally be on a fax machine?

How could you change the the way you represent pictures if you wanted to have colored pictures instead of just black and white?

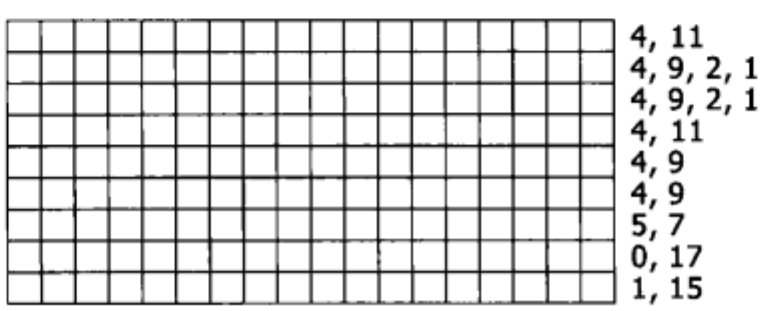
小游戏 4.1 图像解码
Activity 4.1 Converting numbers to pictures

下面你需要完成的任务是对以下图像进行解码。第一张图像最简单，最后的一张最难。很容易就出错哦，所以请先准备好一只铅笔和一块橡皮擦！请记住，每一行开头默认的像素为白色。

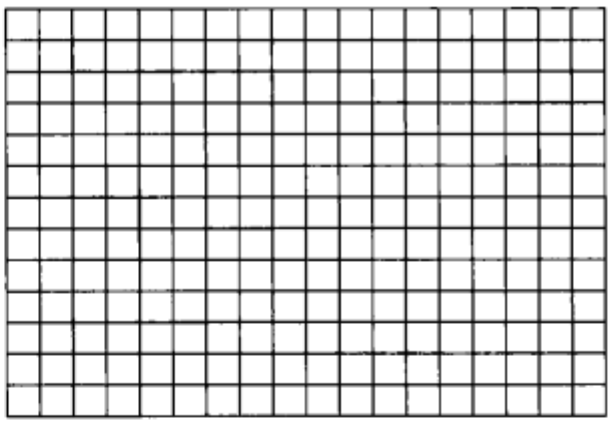
Your task is to decode the following pictures. The first picture is the easiest and the last one is the most complex. It is easy to make mistakes and therefore a good idea to use a pencil and have an eraser handy! Remember that the first number on each line is for white pixels.

完成全部图像的解码工作后，你可能需要站远点来看看都得到了怎样的图像。这些像素的长和宽相当于一台低分辨率打印机打出的 20 倍，也就是其面积的 400 倍！

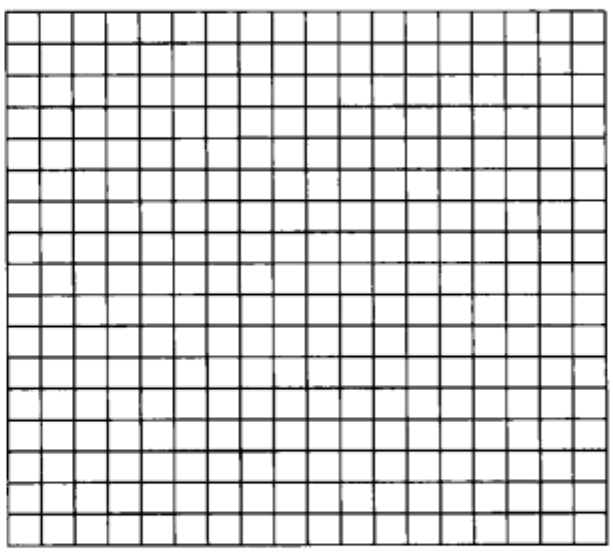
When you've finished each picture you'll probably need to step back quite a way from the paper so you can see what the picture is. These pixels are about 20 times wider and higher than the ones on a low quality printer, so they are about 400 times as big!



4, 11
4, 9, 2, 1
4, 9, 2, 1
4, 11
4, 9
4, 9
5, 7
0, 17
1, 15



6, 5, 2, 3
4, 2, 5, 2, 3, 1
3, 1, 9, 1, 2, 1
3, 1, 9, 1, 1, 1
2, 1, 11, 1
2, 1, 10, 2
2, 1, 9, 1, 1, 1
2, 1, 8, 1, 2, 1
2, 1, 7, 1, 3, 1
1, 1, 1, 1, 4, 2, 3, 1
0, 1, 2, 1, 2, 2, 5, 1
0, 1, 3, 2, 5, 2
1, 3, 2, 5



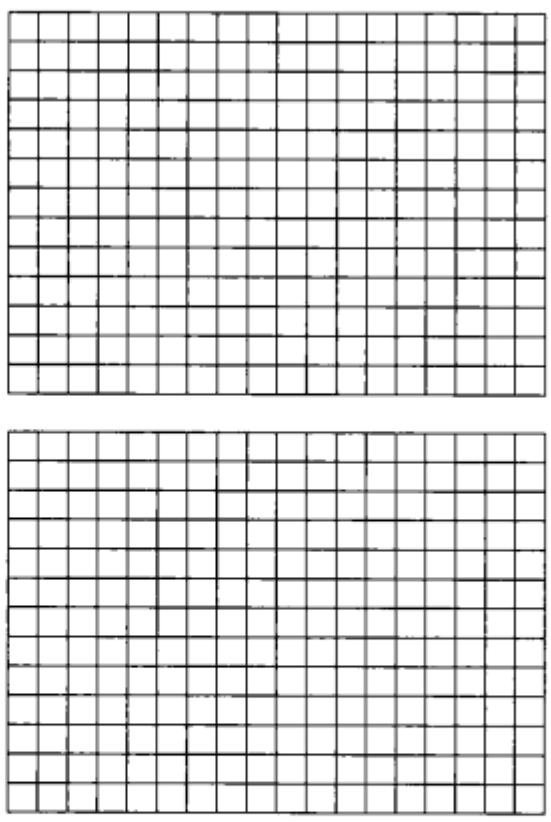
6, 2, 2, 2
5, 1, 2, 2, 2, 1
6, 6
4, 2, 6, 2
3, 1, 10, 1
2, 1, 12, 1
2, 1, 3, 1, 4, 1, 3, 1
1, 2, 12, 2
0, 1, 16, 1
0, 1, 6, 1, 2, 1, 6, 1
0, 1, 7, 2, 7, 1
1, 1, 14, 1
2, 1, 12, 1
2, 1, 5, 2, 5, 1
3, 1, 10, 1
4, 2, 6, 2
6, 6

小游戏 4.2 图像编码

Activity 4.2 Converting pictures to numbers

既然你已经知道如何用数字来表示图像，为什么不自己动手向好朋友展示一下自己编码的图片呢？先在第一张格子图中画出属于你的图像，完成后，把编码的数字写在另一张纸上，交给你的好朋友。你可以用第二张格子图来解码好朋友的图像(注：不必将图中的格子全部填满，如果你的图像不能占满整个画格，可以在底部留出空行)。

Now that you know how numbers can represent pictures, why not try making your own coded picture for a friend? Draw your picture on the first grid, and when you've finished, write the code numbers on a piece of paper and give it to your friend. You can use the second grid to decode your friend's picture. (Note: you don't have to use the whole grid if you don't want to—just leave some blank lines at the bottom if your picture doesn't take up the whole grid.)



进阶篇
Details for experts

本章提到的游程编码一般用于传真机(结合霍夫曼编码法)以及一些 TIFF 图像和 PDF 文件中。采用其他算法的图像压缩格式还有 JPEG(主要用于照片)和 GIF(主要用于简单图片)。JPEG 利用余弦波(将不同波长叠加在一起,用以表示各种不同的色彩和亮度)来重现图像中色彩的改变。GIF 图像采用的原理我们将在第 5 章阐述。

The run length coding method used in this topic is used by fax machines (in combination with a method called Huffman coding), and also in some TIFF and PDF files. Other methods that use different features of the patterns in images are the **JPEG** method (mainly used for photographs) and **GIF** (mainly used for simple pictures). JPEG represents the changing colors in a photograph using cosine waves (different waves are added together to get the variations in color and brightness), while GIF images use a method similar to the one that we will look at in Topic 5.

有趣的事
Curiosity

喷墨打印机通过向纸张上喷射细小的油墨来打印出图像中的每个像素。有些人做出了类似喷墨打印机但体积却大出很多倍的仪器,它利用机械臂控制彩弹枪在墙上喷出壁画!一支彩弹枪可以在一分钟内"打"出几十个像素,而一台性能不错的喷墨打印机则能在一秒之内打印出数百万个像素。

An inkjet print squirts tiny jets of ink onto the paper to print each pixel. Some people have done something similar but a little bigger than that: they used a paintball gun controlled by a robot arm to blast pixels onto a wall! A paintball printer can print a few dozen pixels in a minute, whereas a good inkjet printer prints millions of pixels onto a page in a matter of seconds.

2008 年,Jamie Hyneman 和 Adam Savage 改造的彩弹枪,为视觉计算领域带来了最激动人心的大事件。他们将 1100 支彩弹枪拼装起来,做成了能一枪"打"出一张完整图像的机器。这台机器在不到十分之一秒时间内,画出了达芬奇笔下

In a 2008 mega-event about visual computing, Jamie Hyneman and Adam Savage did a little better with a paintball printer. They made a device with 1100 paintball guns that could fire all the pixels in one shot. Using it, they created an

著名的蒙娜丽莎。尽管如此，这台机器喷出像素的速度还比不上一台喷墨打印机。不过，观看这场精彩的表演要比看喷墨打印机傻傻吐出图像要有趣多了。比起达芬奇为了画出这幅著作耗费的数年时间来说，他们的这部作品可谓是转瞬即成。

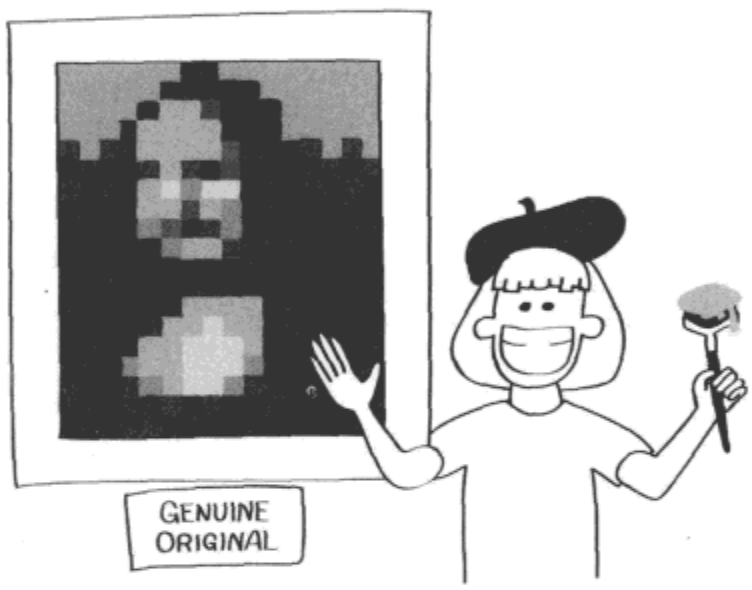


image of Leonardo da Vinci's famous Mona Lisa painting in less than a tenth of a second! That's still fewer pixels per second than an inkjet printer, but it was definitely more fun to watch. And it was a lot faster than Leonardo da Vinci when he painted the original.

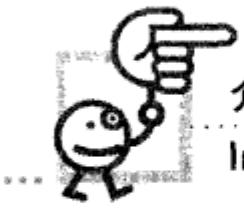
第5章

压缩信息

01 Compressing Information

对我们来说，计算机能越高效地表达信息越好。没人希望通过网络传送数据时总是耗费过久的时间，也没人希望计算机里总没有足够的空间来储存数据。减小数据占用空间的方法称为压缩(compression)。在储存信息之前压缩数据，需要使用的时候再将其解压缩，信息便能在网络中更快地传送了，而硬盘中也能储存更多数据。从网络上下载音乐、图片或视频的时候压缩尤其显得重要，不过，先让我们从如何压缩文档说起吧，因为这会让你对压缩有个大体的概念，而且会更加有趣、简单。

Computers need to represent information as efficiently as possible. This is partly because you don't want to wait too long when they send it over the Internet, and partly because computers only have a limited amount of space to hold information. Reducing the amount of space needed to store data is called **compression**. By compressing data before it is stored and decompressing it when it is retrieved, the computer can send data faster through the Internet, or store more of it. Compression is especially important when you download music, photos or videos, but here we are going to look at how text can be compressed to give you the general idea, and because it's more fun and easier to understand.



介绍
Introduction

第4章中，我们谈到了计算机是如何储存图片和压缩图片的，而我们也掌握了计算机储存文字的方法，下面将学习计算机是如何压缩文档中的文字。本章介绍的压缩方法常用于生成ZIP文件和RAR文件（同样也适用于生成GIF和PNG格式的图片）。

In the previous topic, we learned how computers can store pictures and compress them. We already know how to store text. Now we are going to learn how to compress the text in documents. The method we will look at is used in practice for Zip and RAR file archives. (It's also used for images such as GIF and PNG.)

计算机中能存放大量书籍，就算你想塞进几个图书馆也是没问题的，存放音乐和电影对计算机来说自然也不在话下。但它们往往会占用大量储存空间，而且需要大量时间从网络上下载。通过压缩，文档会变得更"苗条"，从而使我们收集信息的速度变快，使CD、DVD等存储媒介上存放的信息更多。人们一般喜欢使用体积小的随身设备，像是手机或MP3播放器，而它们其实就是计算机，我们希望它们也能塞进更多的内容。对于这种小型的设备来说，压缩尤其重要。

Computers can store whole books or even libraries, and music and movies too, but these all use a lot of storage space and they take a long time to download over the Internet. By compressing the files into a smaller size, we can make things go faster and fit larger collections of information on CDs, DVDs, and other storage. Sometimes we want our gadgets to be very small. Cellphones and mp3 players are computers, and we expect them to store lots of information too. Compression becomes particularly important for small devices.

为了满足上述要求，我们需要做两件事情：压缩数据以便缩小它的体积；然后解压缩以便人们正常阅读信息。通常这两道工序由计算机在后台自动完成。有时候，需要用户明确指示计算机进行压缩，比如压缩成一个ZIP压缩包或GIF图像。更多时候，计算机可自行判断什么时候该自行执行压缩，比如当你准备向网络传送数据前。

To make this work there are two things to do: **compress** the data so that it takes up less space, and then **decompress** it before presenting the information to a person. These two processes are normally done automatically by the computer, behind the scenes. Sometimes users explicitly instruct computers to compress a file, perhaps into a zip archive or a GIF image format. More often computers determine when to do it themselves, such as before transmitting a file over the Internet.

人们发明了多种压缩方法，好比上一章提到的用来压缩图像的游程编码。本章我们将会介绍另一种方法，滑动窗口压缩(LZ compression)。这种方法可以轻易地将文本文件的大小减半，对某些类型图片文件的处理有时甚至效果更加突出。

Many methods of compression have been invented, like the *run—length encoding* we used for pictures in our last topic. In this topic we will look at a different method called **LZ compression**. This can easily halve the size of a file of text, and does even better on some kinds of image files.

术语一点通

Zip and **RAR** files are used to store lots of files efficiently into one file, usually so a collection of files can be emailed or downloaded conveniently in one go. The name "zip" was to indicate that the program producing the files is fast; the name "RAR" stands for "Roshal Archive", named after the person who designed it. **GIF** (Graphics Interchange Format) and PNG (Portable Network Graphics) files are used to store simple images. See the "Curiosity" section at the end of this Topic for an explanation of the name of LZ compression.

ZIP 和 RAR 文件可以将大量的文档打包到一个文件中，因此这个压缩包便能一次性地用 email 寄出或一次性被下载了。"ZIP"这个名字表明了程序生成压缩包的速度相当迅速；"RAR"是"Roshal Archive"的缩写，取自发明这种格式的 Roshal 先生的姓。图形交换格式 GIF(Graphics Interchange Format)和可移植性网络图片 PNG(Portable Network Graphics)一般被用来保存一些简单的图像。关于滑动窗口压缩"LZ"的解释请参见本章最后"有趣的事"小节。

小游戏 5.1 文字的解压缩

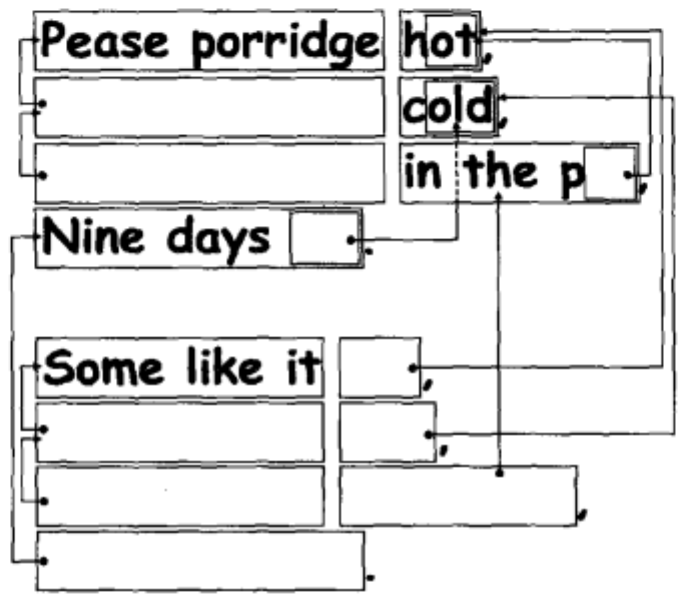
Activity 5.1 Decompressing text

下面是一首缺词少字的诗歌，你能完整地恢复它的原貌吗？不妨依照空格旁箭头的指示，复制箭头指示的内容来寻找缺失的字母和词语。比如，第一个空格引出的箭头指向短语"Pease porridge"(豌豆粥)。

Many of the words and letters are missing in the poem below. Can you fill in the missing letters and words to complete it correctly? To find the missing letters and words, follow the arrow from the empty box, and copy what is in the box that the arrow points to. For example, the first empty box points back to a box containing the phrase "Pease porridge".

你也许注意到有些箭头指向的是空的格子！不用担心，如果你是从头做起的，这些格子总会在你需要它们的时候被填起来。

Notice that some of the arrows point to empty boxes! However, if you start at the beginning, those boxes will be filled by the time you need them.



小游戏 5.2　文字的压缩
Activity 5.2　Compressing text

让我们通过一个有许多叠字的诗歌，来学习一下如何压缩文本。

The Rain[6]

Pitter patter
Pitter patter
Listen to the rain
Pitter patter
Pitter patter
On the window pane

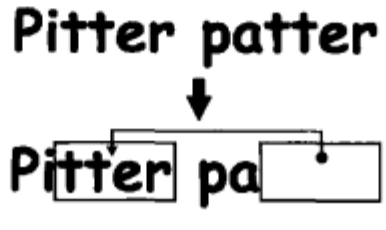
Let us look at a poem that has a lot of repeated words to see how we can compress it.

[6] 译者注：这首诗的大意为“小雨，噼噼啪啪，噼噼啪啪，听这小雨啊，噼噼啪啪，噼噼啪啪，打在窗格上”。

a 在这首诗中，你能找出两个或更多重复的字母、单词或词组吗？不要忘记标题哦！（这个游戏中可以忽略字母的大小写，但现实中计算机系统进行压缩时是不能忽略大小写的。）

a In the poem, can you find groups of two or more letters that are repeated, or even whole words or phrases? Don't forget the title! (You can ignore differences in capitalization for this exercise, although computer systems can't.)

b 你能用下面所示格子来代替这首诗中重复的单词吗？画格子框住重复的字母；然后就像上例一样，画一个箭头从新的格子回指到之前的格子。最后你会得出与小游戏 5.1 一样的结果。



b Can you replace those repeated words with boxes as shown below? You should draw a box around the letters that are repeated; you should draw an arrow pointing backwards from the box that is later in the poem to the earlier box, like the example below. You will end up working out where the empty boxes would be in a puzzle like the one in Activity 5.1.

刚才你所做的正是计算机压缩数据的过程：它在文中查找出现过的重复字母组合，如果发现这种字母组合在前文也出现过，这个字母组合会被移除并用指针（就好比你画的箭头和字格）代替，且指针将指向该字母组合之前出现的位置。而解压缩文档时，计算机处理的过程就好比你在小游戏 5.1 中做的那样，依据指针读取指向前文的字母组合。

You have just done what a computer does when it compresses data: it searches for groups of letters earlier in the text that are the same as the ones to be removed. If it finds that a group of letters has occurred earlier, they can be removed and replaced with a reference (which you drew as arrows and boxes) to where the earlier occurrence was. When a file is decompressed, the computer does what you did in activity 5.1: just follow the reference and read the group of letters from where it refers to.

现在让我们看看替换这些字母组合将这首诗压缩了多少。

Now let's try to figure out how much these replacements compress the poem.

c 原诗中一共包含多少个字母以及空格？

c How many letters and spaces were in the original poem?

d 压缩后的诗中包含多少个字母以及空格？

d How many letters and spaces are in the compressed poem?

e 这首诗压缩后比压缩前缩小了多少？

e How much smaller is the compressed poem than the original?

其实上面的比较并不公平，因为压缩后的诗需要额外的信息来指示被移除的字母。你所每画出一个箭头，在计算机中都将相应地记录下这个位置返回到原始单词的距离、包含字母的个数、字格中的符号等。例如，计算机会用到以下类似的代码：

This isn't quite a fair comparison, because the compressed poem requires extra information to point to the removed letters. For each arrow you drew, the computer must record how far back to go to the original word, and also the number of letters, or characters, in the box. For example, the computer could use a code like this:

Pitter patter → Pitter pa(7,4)

其中，7 代表回退 7 个字符（即返回到第一个“t”）后开始复制（回退的时候不要忘记算上空格）；4 代表需要复制 4 个字母（“tter”）。

The 7 means to go back 7 characters (to the first “t”) and start copying (don't forget to count the space when going back!); the 4 means that 4 letters should be copied (“tter”).

f 压缩后的诗中有多少个箭头？

f How many arrows did you draw for the compressed poem?

g 计算机中表示箭头往往需要用到两个字符的长度。在这里表示全部的箭头一共需额外占用多大空间呢？

g On a computer, the numbers to represent each arrow requires about the same space as two characters. So how much extra space will all of the arrows need?

h 如果把格子占用的空间也算进去，压缩后的诗一共需要多少个字符？

h Counting the space for these boxes, how many total letters are needed to represent the compressed poem?

i 这样一来，压缩后的诗比原诗少了多少字符呢？

i How much smaller is this version of the compressed poem than the original poem?

j 如果用像素代表文字，你觉得还能用这种方式进行压缩吗？

j Do you think this method would work if it were given pixels instead of text?

进阶篇
Details for experts

计算机中实际使用的并不是我们在纸上画出的格子和箭头，而是由两个数字来分别表示它们，例如(6,4)。第一个数字表示箭头指向的位置(比如6表示前文第6个字符)，第二个数字表示字格框的长度(比如4表示复制4个字符)。输入以下代码：hokey p(6,4)，表示我们将回数包括空格在内的6字符(也就是指向这段信息的第二个字符"o")，从这里开始复制4个字符("okey")，输出结果为"hokey pokey "。

A computer can't use arrows and boxes like the ones we drew on paper; instead, it can use two numbers such as (6,4) to represent an arrow and box. The first number indicates where the arrow is pointing to (e.g. 6 means 6 characters earlier in the text), and the second number indicates the size of the box (e.g. a 4 means to copy 4 characters). So in the following input: hokey p(6,4), we count back 6 characters including any spaces (which takes us back to the second character – "o" – of the message), and copy 4 characters from that point ("okey"), giving the output "hokey pokey".

下面的代码表示了一种十分有趣的情况。

The following code presents an interesting situation.

```
miss(3,4)ppi
```

其中3指示我们回退到第一个"i"，并复制4个字符。可这时我们只有3个字符"iss"能用。然而，一旦我们开始复制，第4个字符便能被使用了，这段代码也能被解压缩(输出结果为"mississippi")。事实上，这种自指向代码非常有用，并能实现类似我们第4章提到的游程编码行为。比如，代码 b(1,19)代表将字母"b"重复20次。指针(1,*x*)代表将前面的字符重复 *x* 次。

The 3 takes us back to the first "i" in the text, but we need to copy 4 characters, even though we only have "iss" available. However, as soon as you start copying, the 4th characer becomes available, and the phrase can be decoded (it is"mississippi"). In fact, this kind of self-reference code is very useful, and can behave like the run-length coding we used in Topic 4. For example, the code b(1,19) represents the letter "b" repeated 20 times. A reference (1,*x*) effectively means repeat the previous character *x* more times.

此外，指针的前面还需要一个标识来说明它到底被作为指针使用还是仅仅被作为字符使用。因此，指针一般需要占用两个字符，而指针处通常对应的原字符长度平均为 4 个，这样一来，压缩后的数据体积大约会缩小到原始体积的一半。

Furthermore, the references need an indicator in front of them to say if they should be used as a reference, or if they are just characters. Allowing for this, typically references take the same space as about two characters. Often they will match an average of four previous characters, so the data is reduced to about half the original size.

寻找前方出现的重复词语或字符组合需要占用系统大量的进程，有时候计算机需要在前文数以千计个字符中翻找。然而，人们已经开发了许多在大量数据中快速查找目标数据的方法，这一点将是我们第 9 章的课题。

Finding the earlier occurrences of a word or group of characters takes a lot of processing—the computer may have to search through many thousands of previous characters. However, there are some very fast methods for searching large amounts of data quickly, which we will look at in Topic 9 (Searching).

有趣的事
Curiosity

本章我们谈到的压缩方式称为“Ziv-Lempel”压缩，由 Jacob Ziv 和 Abraham Lempel 这两名计算机科学家共同于 20 世纪 70 年代发明。不幸的是，有人将他们论文上的署名顺序弄错，本该称为“ZL”压缩，却被错误地称作“LZ”压缩。由于这个错误流传太广以至于直到今天我们还依旧沿用着这个错误的名字“LZ 压缩”。

The method we have described here is named “Ziv-Lempel” compression after Jacob Ziv and Abraham Lempel, the two computer scientists who invented it in the 1970s. Unfortunately someone mixed up the order of their names when they wrote an article about it, and called it “LZ” compression instead of “ZL” compression. So many people copied the mistake that Ziv and Lempel’s method is still commonly called “LZ compression”!

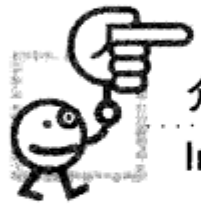
第6章

检测错误

0 Finding Errors 1

当数据被储存在硬盘或传送到网络上时，它们一般是不会发生改变的。不过，有时候一些故障也会导致数据被突然改变，比如电子干扰。这可不是我们想要的，而避免这类事件的发生至关重要。幸运的是，如果我们设置得当，计算机便能自动检测出数据的改变，有时甚至能自动修正错误!本章的小游戏将用一个神奇的魔法来展示计算机是如何做到这些的。

When data is stored on a disk or transmitted over the Internet, it doesn't usually get altered in the process. But sometimes things go wrong and the data is changed accidentally by things like electrical interference. This is not what we want! — and it's usually very important to avoid it. Fortunately, if we set things up right, computers can automatically detect when data has been changed. Sometimes they can even correct it! This activity uses a magic trick to show how this can be done.



介绍
Introduction

本章我们将来学习一种确保数据不会意外发生改变的方法。

In this topic we are going to learn about one way in which computers make sure that data isn't changed accidentally.

假如你向网上卖家发了一封 email，确认你将付 20 元来购买他的一件商品。但是一阵电子干扰过后，email 中的 20 元变成了 80 元。卖家一定会欣然接受你的订单，有人居然愿意出 4 倍的价格来购买一件商品！

Imagine that you send an email to an online trader saying that you will pay $20 for their product. But suppose some interference occurs along the way, and the $20 is changed to $80. The trader will probably be very happy to accept your offer, and charge you 4 times as much for the product as you wanted to pay!

从前面几章我们了解到，任何储存在计算机或传送在计算机之间的数据都是采用比特(二进制数字)的形式予以表达的。存储或传输设备上发生的错误，很容易导致数据的突然变化。CD 上的划痕或表面的小灰尘会把 0 变成 1，或将 1 变成 0；硬盘存放数据的区域可能被意外地磁化；网络中的干扰和连接不畅也会导致比特被改变。那么我们要怎样才不用担心发生这些意外呢？

We know from previous topics that everything stored by computers and sent between them is represented as bits (binary digits). It is very easy for these to be changed accidentally because of errors in the devices that are storing or transmitting them. A CD might get a scratch or piece of dust that changes a zero into a one or vice versa. A hard disk might have the magnetism accidentally fade where a bit is stored. On the Internet, interference and bad connections can cause bits to be altered. So how come we don't have to worry about this?

在计算机发明的早期，这个问题相当严重。很快地，科学家们便发明了让计算机自动检测数据中的错误并自动修复的方法。我们将学习到其中的一个方法：奇偶校验(parity)。依照计算机中使用 0 和 1 的原则，我们使用双面的卡片来模拟一下这个神奇的魔法。

This was a serious problem on early computers, so scientists soon invented methods to allow computers to detect errors in data and correct those errors. We will learn about one way to do this using a method that is called **parity**. But instead of using zero and one bits inside a computer system, we'll use cards with two sides, and do it as a magic trick.

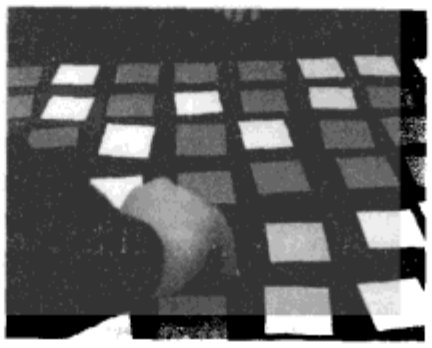
小游戏 6.1 翻卡魔术
Activity 6.1 Card flip magic

请准备好 36 张卡片，保证每张卡片正反两面的图案和颜色不同。你可以用一套扑克，或自己用纸片剪出卡片，一面画上黑色一面保持白色。

You will need 36 cards that have a different color or pattern on each side. A pack of playing cards is suitable, or you can just cut out some cards that are (say) black on one side and white on the other.

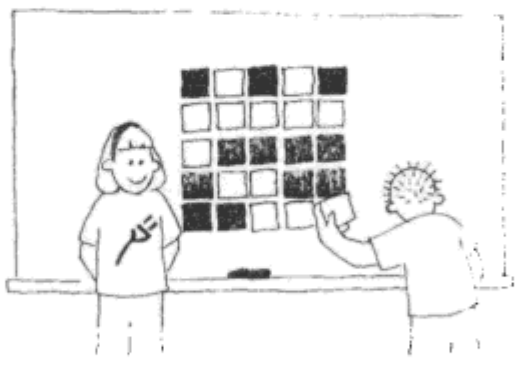
视频：请观看翻卡魔术的视频。

Video: Watch the card flip magic video.

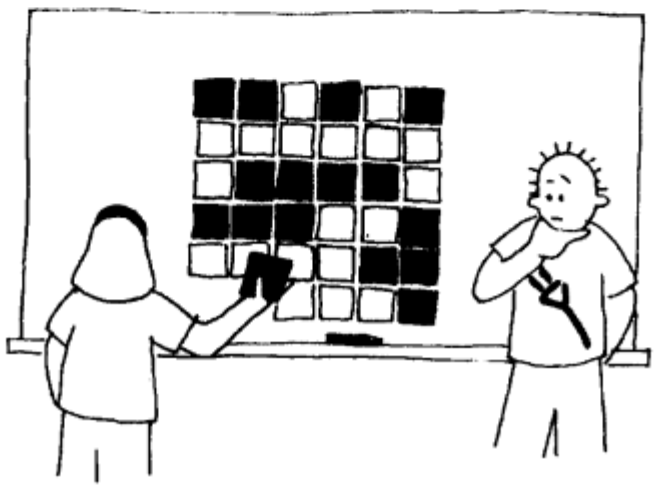


这个魔术需要一个同伴来配合，让他将卡片放在桌子上(如下图中的男孩)，并由他来决定每张卡片放置的正反。接着，你可以增加几张卡片(你可以向他解释这样做是为了增加这个魔术的难度)。最有"魔法"的部分在于，你背过身去，在看不到同伴操作的情况下，让他翻转一张卡片，你转过身来时总能告诉他哪一张卡片是被翻过面的！

The way the trick will go is that you will get a friend to put some cards on the table (for example, they might put them down as the boy has in the picture). Your friend gets to choose which way up each card is. You will then add some extra cards ("to make it harder"). The "magic" part is that you will turn away so you can't see the cards, get your friend to turn one over, and then you turn around and show them which card they turned over.



当然，诀窍就在于你增加的那几张卡片。实际上你是在原有卡片的最右侧增加一列、在最底部增加一行来放新增加的卡片（当然你口头上说的是这样做是为了增加魔术的难度）。下图中的女孩正在增加新的卡片，而她在每个位置上放置卡片的颜色，就是诀窍所在了。



a 上图中，每行有多少张白色的卡片（左下方最后两张从右至左将依次放上黑色和白色卡片）？每列呢？在你实际开始魔术前，试着自己找出这些数字的特殊规律。（提示：这些数字的共同点是什么？）

从刚才的问题中，或许你已经意识到每行及每列白色卡片的数量都是偶数。每当男孩在一行中摆放了奇数张的白色卡片，女孩便会在行末增加一张白色卡片来保证该行白色卡片的总数为偶数。如果男孩摆放时白色卡片的总数已经为偶数了，那么女孩则添加一张黑色的卡片来保持白色卡片总数仍为偶数。

让我们来看看如何利用上述规律算出是哪一张卡片被改变了朝向。

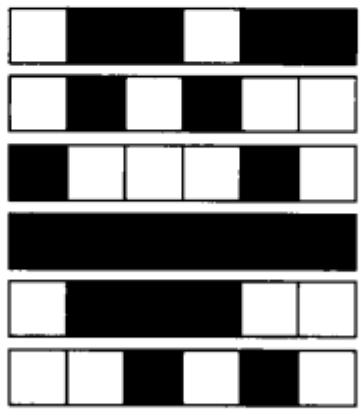
Of course, the secret is in the extra cards that you add yourself. You will add another column of cards going down the right side, and a row across the bottom (you should say that you are doing it just to make it a bit harder). The girl in the picture below is adding the extra cards, and there is a trick to which color she is choosing for each card.

a In the picture above, how many white cards are there in each row across (the last two cards in the bottom left corner will be black, then white)? How many are there in each column down? See if you can work out what is special about those numbers before you carry on (Hint: what do all the numbers have in common?)

For question a, perhaps you realized that the number of white cards in each row and column is an even number. If the boy had put an odd number of white cards in a row, the girl would add another white card on the end to make it an even number. If he had already put an even number there, she should add a black card to keep the number of white cards even.

So let's look at how you can use this to work out which card has been changed.

b 下面的卡片组合中，每行和每列的白色卡片数一开始均为偶数。记下现在各行上白色卡片的总数。



c 哪一行白色卡片的总数不再为偶数了呢？

d 记下每列白色卡片的总数。

e 哪一列白色卡片的总数不再为偶数了呢？

f 现在你能判断出是哪张卡片被翻动过了吗？

通过以上几个问题，想必你已经意识到，被翻动过的卡片所在的行和列中白色卡片的总数都变成奇数了吧。要找到那张被翻动过的卡片，只需看看哪一行哪一列拥有奇数的白色卡片，它们相交的地方就是你寻找的目标了。

这就是魔法的真相。熟能生巧，表演的时候试着谈论一些其他的话题，这样观众就不会发现其中的奥秘啦。

b The following grid of cards started with an even number of white cards in each row and column. Write down the number of white cards in each row.

c Which row no longer has an even number of white cards in it?

d Write down the number of white cards in each column.

e Which column no longer has an even number of white cards?

f Can you deduce which card must have been changed?

From the above questions, you have hopefully realized that the row and column of cards containing the flipped one will now have an odd number of white cards in it. So to find the flipped card, just look for the row and column that have an odd number of cards, and choose the card that is in both of them.

That's all there is to the trick. If you practice it so you can do it quickly, and while talking at the same time, your audience probably won't work out how you're doing it.

可以和你的朋友一起练习一下这个魔术(轮流扮演放置原始25张卡片和增加卡片的角色),当你们两人都能自信地完成这个魔术后,找一个不知道其中奥秘的朋友来试验下吧(可以在家中进行)。练习的时候,让你的朋友将25张卡片放成5×5的正方形,正反朝向随意。然后你在卡片阵列的右边增加一列,让每一行的白色卡片总数保持偶数(如果此行白色卡片总数为奇数,加一张白色卡片,否则加一张黑色卡片),在卡片阵列的底部也增加一行以保证每列白色卡片的总数为偶数。闭上眼睛,让你的朋友任意翻过一张卡片(只能一张哦)。睁开眼睛,告诉他哪张卡片被翻过来了,让你的朋友感受一下魔法的魅力吧。

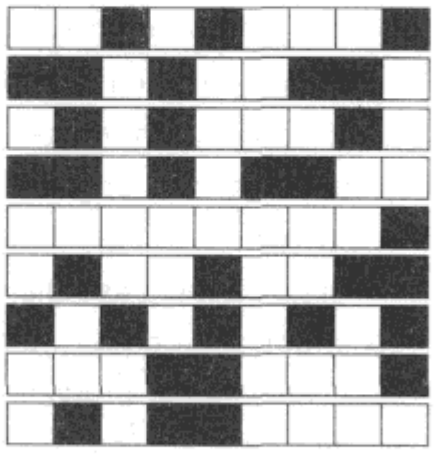
If you can, practice this magic trick with a friend (taking turns at who puts down the first 25 cards, and who adds the extra row and column). Once you can both do it confidently, you can try it out on someone else who doesn't know how it works (perhaps at home). When practicing, ask your friend to lay out 25 cards in a 5 × 5 square, with a random mixture of sides showing. Then add an extra column on the right side, to make the number of white cards in each row an even number (add a white card if it's currently an odd number, or a black card if it's already even). Also add a row along the bottom to do the same with the number of white cards in the columns. Then close your eyes and ask your friend to flip over one (and only one) of the cards. Open you eyes again, and impress your friend by revealing which card was flipped.

如果你能找一个助手帮你放置增加的卡片,这个魔术便能升级成"读心术"。你先离开房间,让一个志愿者随意放置25张卡片,接着你的助手添加余下的卡片让总数变成36张。等志愿者随意翻动其中一张卡片后,完全没有目睹整个过程的你再走入房间,一下便能指出哪一张卡片被翻动过。

A variation of this trick is to appear to be "mind reading", which can be done if you have an assistant who knows how to put down the extra row and column of cards. You can leave the room while a volunteer puts down the 25 random cards, and then your assistant adds the extra cards to make it up to 36. The volunteer can then flip a card, and you return to the room and point out which card they flipped, despite never having seen the cards at all.

这个方法适用于任何呈矩形排列的卡片组合。下图中有81张卡片,已经被放置为每行每列的白色卡片总数均为偶数的形式。那么哪张牌被翻动过呢?

The trick will work for any rectangle of cards. Here's a set of 81 cards that started out with an even number of white cards in every row and column. Which card has been flipped over?



这些被放置的卡片就好比计算机中的比特(0 或 1),而 0 和 1 的组合代表了数字、字母或图像。那些新增加的卡片我们称之为奇偶校验位,计算机就是通过在数据中添加奇偶校验位来保证数据不被随意修改。

本书的光盘提供了翻卡魔术街头艺人表演版的视频,你将看到这个魔法更玄幻的演绎方式。

The cards that you put down represent the bits on a computer (zero or one). The groups of zeros and ones might represent numbers, letters or pictures. The extra cards that you added are called **parity cards**; the parity cards are added by the computer and protect the data cards to make sure that the data isn't changed.

There is an unusual approach to the magic trick in this topic in the video "Computer Science Buskers".

术语一点通

"Parity" comes from the same root word as "pair"—**even parity** means that there is an even number of objects (they can be put in pairs), and **odd parity** means they can't be put into pairs. The system described here uses even parity, since we are making sure that the white cards can be put into pairs (for example, 6 white cards in a row are 3 pairs, but 5 white cards would make two pairs and one left over).

奇偶校验(parity)一词的英文来源于词根"pair"(对),偶校验(even parity)表示物件的总数量为偶数(它们能被成对地放置在一起),奇校验(odd parity)表示它们不能被组成对。在这里我们用偶校验来保证白色卡片的数量总能组成对(比如,一行中的 6 张白色卡片组成了 3 对,但是 5 张白色卡片只能组成 2 对和剩下的一单张)。

利用类似奇偶校验的方法，可以保护计算机中几乎所有的数据。数据硬盘、CD、DVD、闪存、网络下载、电子邮件和网页都在数据中添加了你看不到的校验码。一旦系统中个别比特发生错误，计算机就会在你不知情的情况下自动恢复原始数据。下一个小游戏中，我们将来看看如果不止一个比特发生改变该怎么办。

Methods similar to the parity method are used to protect almost all data on computers—every disk drive, CD, DVD, flash memory, internet download, email and web page adds extra bits to the data that you never see. If a small fault in the system changes a few bits, the computer can usually reconstruct your data without you even knowing that there was a problem. In the next activity, we will look at what happens if there are more errors than just the single bit being changed that we had in the magic trick.

小游戏 6.2　发现更多的错误

Activity 6.2　Detecting more errors

通过前面的小游戏，我们了解到奇偶校验是如何来检测并修正一个错误的，即当一张卡片被翻转的情况。然而，计算机中也会有不止一个比特发生错误的时候。在下面的游戏中，我们将来看看在不只一个错误发生的情况下，奇偶校验卡是如何工作的。

In our previous activity, we saw how parity bits can be used to detect and correct one error, that is, when one of the cards was flipped. However, sometimes in a computer, more than one error can occur at the same time. In this activity we will see how parity cards work when there is more than one error.

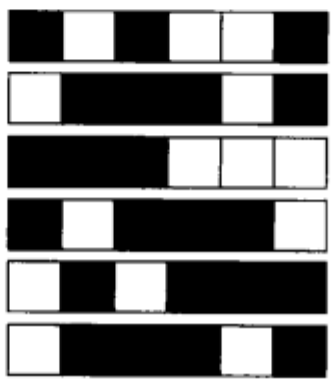
有时候，仅须检测到错误的发生就足够了。比如两台计算机正通过网络收发数据，如果接收方察觉数据在传输过程中被改变了，它只用让发送方再传送一次即可。然而，有时候数据是无法再一次被发送的，例如用磁盘或闪存保存的数据。一旦因为磁化或过热导致磁盘上的数据被改变，除非计算机能够修正错误的部分，否则这个数据就永远地遗失了。因此检错和纠错都是相当重要的事情。

Sometimes, simply **detecting** that an error has occurred is enough. For example, imagine that two computers are sending data and parity bits over a network. If the receiving computer detects that data has been altered during transmission, it can simply ask the sending computer to send the data again. However, in some cases it isn't possible to resend the data. For example, imagine that a computer is storing data on disk or flash card. If the data on a disk is altered by exposure to magnetism or heat, then the data is lost unless the computer can **correct** the errors. So both **error detection** and **error correction** are important.

g 当发生一系列错误时，什么情况下计算机能利用奇偶校验位来检测并修正错误？

g Let's see when a computer can use parity bits for error detection and correction, when different numbers of errors happen.

h 下图所示的卡片阵列中每行和每列的白色卡片数均为偶数，但其中有两张卡片已经被翻过了。哪几行哪几列现在处于错误状态？你能推断出是哪两张卡片被翻动过了吗？



h The following grid of cards used to have an even number of white squares in each row and column, but two cards have been flipped over. Which rows and columns are now wrong? Can you figure out which two cards were flipped?

g 在这个例子中，你能检测到错误发生了（因为有几行和几列都出现了奇数张白色卡片），但你却无法修正它们（因为有两种卡片被翻动的可能性）。

g In this case, you would have been able to detect that some errors occurred (because there were rows and columns with an odd number of white cards), but you couldn't correct them (because there are two ways to flip the cards back).

i 如果计算机接收到的一条信息中，提示它可能存在两个比特的错误。那么计算机此时该如何处理？

i If a computer receives a message that seems to have two bit errors in it, what do you think it should do?

j 试着用几张卡片排列成满足奇偶校验原理的阵列（保证每行和每列的白色卡片均为偶数）。看看你能在翻动两张卡片后，保持每行和每列白色卡片总数仍为偶数吗（因此无法检测错误）？

j Lay out some cards with the correct parity (even number of white cards in every row and column). Now try to find a way to do two card flips in a way that every row and column still has an even number of white cards. Is it possible to do this (and therefore not have the error detected)?

k 你能做到翻动 3 张卡片，但不被检测出来吗？

k Can you find a way to flip 3 cards without it being detected?

l 最后，让我们来看看翻动 4 张卡片的情况。如果我们在同一行中翻动 4 张卡片，能被检测出来吗？

l Finally, let us see what happens with four flips. If we flip four cards in the same row, will it be detected?

有办法翻动4张卡片但并不被检测出来吗?

所以说,如果发生了4个错误,计算机或许连一个错误都检测不出。

下面的表格总结了我们的发现:如果发生了一个错误,它总能被检测出来并能被修正;如果发生2个或3个错误,计算机能够检测出来,但或许无法修复错误;如果发生4个错误,计算机可能连一个错误都检测不出!

Is there a way to flip 4 cards without it being detected?

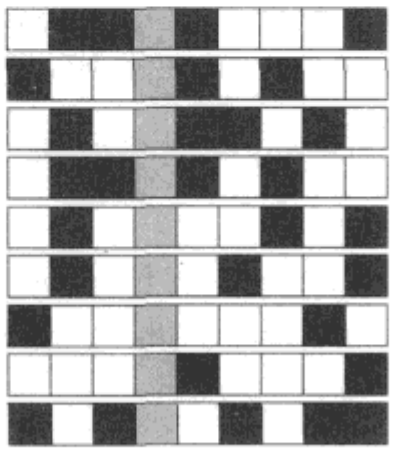
So, if four errors occur, a computer may not be able to even detect that an error occurred at all, but sometimes it will.

The following table summarizes what we have found. If one error occurs, it can always be detected and corrected. If two or three errors occur, the computer can always detect that an error occurred, but it might not be able to fix the errors. And, if four errors occur, the computer might not even be able to detect that an error happened!

Number of Errors	Always Detect?	Always Correct?
1	Yes	Yes
2 or 3	Yes	No
4	No	No

当发生多个错误的时候,有一种特殊情况下错误能被纠正。这一点非常有用。下图显示了一个奇偶校验阵列(每行每列的白色卡片数均为偶数),但是它的第四列全部丢失(灰色区域)。

你能描述一下这个状况代表的意思吗?

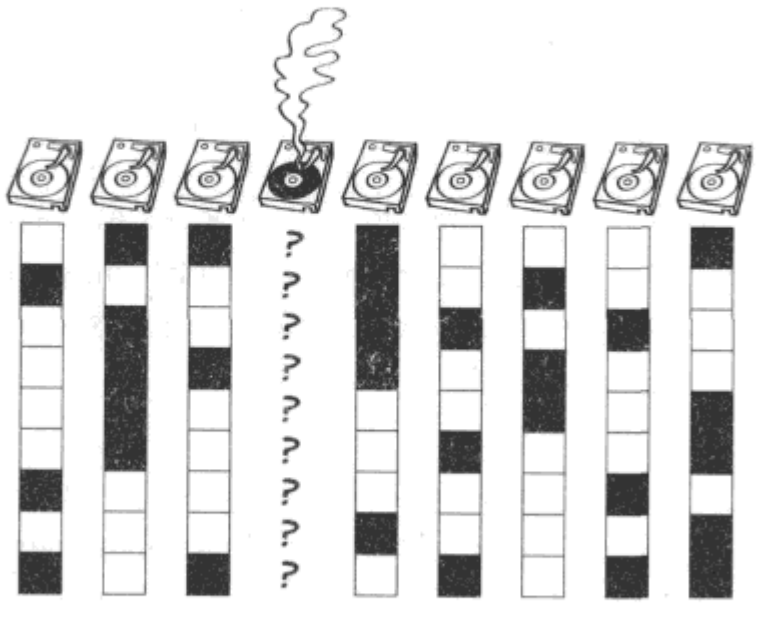


There is one special situation where many errors can be corrected, and it happens to be particularly useful: the grid below was set up for even parity (every row and column had an even number of white squares), but the whole 4th column has been lost (the gray squares).

Can you work out what it was?

RAID 硬盘系统采用的就是这种纠错方式，通过将数据分散储存在多块而不是一块硬盘中，来保证运行的高速性和稳定性。RAID 是“Redundant Array of Independent Disks”(独立冗余磁盘阵列)的简称，它利用额外附加的硬盘来提高硬盘速度和纠错性能。

例如，奇偶校验系统的一个优化方案称为 RAID 5。假设你需要使用 8 个硬盘来储存大量的数据，这些数据包含大量字节，可能超过数亿字节(吉字节)甚至千亿字节(太字节)。这时你可以将每个字节打散成 8 比特分别储存在多个硬盘上，而不是将数据陆续填满每个磁盘。这样的存储方式会让系统运行得更快，因为当计算机需要读取文件时，它只用分别同时向每块硬盘读取片段即可。该方法也可用于提高纠错性能：如果再增加存有奇偶校验位的第 9 块硬盘，我们可以用上面的思路让每一列数据分别放在不同硬盘上(如下图)，这样一来，如果其中一块硬盘被损坏，即使损失全部数据，我们仍然能依靠奇偶校验的思路来修复原始数据——只用算出遗失的比特使得 9 个硬盘上值为 1 的比特数总保持为偶数即可。



This kind of error correction is used in RAID hard disk systems, which are designed to be very fast and very reliable, by spreading the data for one file across several hard disk drives instead of putting it all on one. RAID stands for “Redundant Array of Independent Disks”, and uses extra hard disks to improve the speed and error correction.

For example, the parity system is used in a variant called RAID 5. With this variant, suppose you are storing so much data that you need 8 disks. The data will contain many bytes—probably thousands of millions of bytes (gigabytes), or even thousands of gigabytes (terabytes). Instead of filling up each disk one after the other, you could spread the 8 bits of each byte over the disks—one bit on each disk for each byte in the file being stored! This makes the system faster, because when your computer needs to read a file it is reading parts of the file from each disk at the same time. It can also be used to get excellent error correction: if we add a ninth disk that contains a parity bit, we can use the same idea as the grid above, where each column is on a separate disk (as in the diagram below). If one of the nine disks crashes and we lose all of its data, we can still work out what all the original data was using the approach in the last question—just work out what the missing bit should be to make sure there is an even number of 1 bits in the 9 spread across the disks.

正因如此,RAID 系统的存取速度飞快,就算任何一块硬盘被损坏也可以保证原始数据不被丢失。对于大型数据中心和重要的网站来说,RAID 已成为提高运行速度和保证稳定性的廉价方案。因为就算坏掉了一块硬盘,只要买一块便宜的替代就可以了,比起购买 8 块性能稍稳定但价格不菲的硬盘来说,买 9 块便宜的硬盘和一些备用盘更加经济实用。

So the RAID system is much faster, and any one of the nine disks can fail completely without losing any data. For large data centers and important web sites, this is considered a very cheap way to make them much faster and more reliable—if a disk fails, you just swap it for a cheap new one. It can be more economical to buy 9 cheap disks and a few spares, than 8 very expensive and more reliable disks that might last a lot longer.

小游戏 6.3　ISBN 检测

Activity 6.3　ISBN numbers

书籍编码中也会用到一种检测技术,每一本公开发行的书都会在封底编上一个 10 位或 13 位的编号,称为国际标准书号(International Standard Book Number,ISBN)。ISBN 的最后一位数字称为计算机校验码(check digit),它就好比前面游戏中使用到的奇偶校验位。这样一来,如果你用 ISBN 订购一本书,书店可以用其中的计算机校验码来检查你是否订错。一些经常会发生的错误有:某一位的数值发生改变;两个相邻的数字弄反了;多加了一位数字或少输入了一位数字。

A checking technique is also used with book codes. Published books have a ten-or thirteen-digit code, called an ISBN for International Standard Book Number, which is usually found on the back cover. The last digit is a **check digit**, just like the parity bits in the exercise. This means that if you order a book using its ISBN, the shop can check that you haven't made a mistake. Some common errors are: a digit has its value changed; two adjacent digits are swapped with each other; a digit is inserted in the number; and a digit is removed from the number.

书店的计算机仅通过查看 ISBN 的校验码便能判断你是不是犯了以上错误,避免你买错书。

The shop's computer simply looks at the check digit to make sure that you didn't make any of these errors. That way you don't end up buying the wrong book!

从 2007 年 1 月开始,图书统一开始使用 13 位的 ISBN,而在此之前,ISBN 通常只有 10 位。下面我们来看看如何算出 10 位 ISBN 的校验码。

From 1 January 2007, books started using 13-digit numbers. Before that they were usually 10 digits. Here's how to work out the check digit for a 10-digit number:

将第一位数字乘以 10，第二位乘以 9，第三位乘以 8，依此类推，一直到第九位乘以 2。将它们相加的总和除以 11，记下余数；再将这个余数减掉 11 之后就是 ISBN 的最后一位数字。有时候校验码的值为 10，这种情况下我们用 X 代替（X 在罗马数字中代表 10）。

Multiply the first digit by ten, the second by nine, the third by eight, and so on, down to the ninth digit multiplied by two. Now add all of these values together. You next divide your answer by eleven and remember the remainder; subtract the remainder from 11 and this should be the last digit of the ISBN. It is possible to come up with a check digit of the value of 10; when this happens, the character X is used (X is the Roman numeral for 10).

让我们来看一个例子。

Let us look at an example.

ISBN 0-13-911991-4 :

(0 × 10) + (1 × 9) + (3 × 8) + (9 × 7) + (1 × 6)
+ (1 × 5) + (9 × 4) + (9 × 3) + (1 × 2) = 172

172 ÷ 11 = 15 with a remainder 7

11 − 7 = 4

o 计算结果符合 ISBN 的最后一位吗？

o Does the result of the calculation match the last digit of the ISBN?

如果最后一位不是 4 的话，我们便知道输入 ISBN 的时候一定发生了错误。

If the last digit wasn't a four, then we would know that a mistake had been made typing in the book's number.

p 如果将上面 ISBN 码中的 3 换成 4（即 0-14-911991-4），最后一位的校验码应该是多少呢？

p What will the check digit be if a 4 was typed instead of a 3 in the number above (i.e. 0-14-911991-4)?

q 如果两个数字被颠倒了，即输入成 0-13-191991-4，结果如何？

q What if two digits were typed in the wrong way round, for example, 0-13-191991-4?

r 你能找出只改变 0-13-911991-4 中的一个数字，并保证最后的校验码不变的方法吗？

r Is there any way you could change just one of the digits in 0-13-911991-4, and still have the check digit come out correctly?

s ISBN1-00-235045-X 的校验码正确吗？

s Does the following ISBN have a correct check digit: 1 − 00 − 235045 − X?

当使用 13 位 ISBN 后(从 2007 年 1 月开始),生成校验码的公式变简单了:只需将第 1 位乘以 1,第 2 位乘以 3,第 3 位乘以 1,第 4 位乘以 3,依此类推,直到第 12 位乘以 3;然后将各位结果相加之后,取总和的末位数字(即除以 10 之后的余数)后再减去 10(如果结果为 10,取 0)即可。

例如:

If you have a 13-digit ISBN (used from 1 January 2007), the formula is a little simpler: you multiply the first digit by 1, the second by 3, the third by 1, and fourth by 3, and so on, up to the 12th digit, which is multiplied by 3. Add these together, take the right-hand digit of the sum (that is, the remainder divided by 10), and subtract that from 10 (if this comes out as the number 10, use 0 instead).

For example:

ISBN 978-897283571-4:

$$(9 \times 1) + (7 \times 3) + (8 \times 1) + (8 \times 3) + (9 \times 1) + (7 \times 3) + (2 \times 1) + (8 \times 3) + (3 \times 1) + (5 \times 3) + (7 \times 1) + (1 \times 3) = 146$$

$$146 \div 10 = 14 \text{ with a remainder } 6$$

$$10 - 6 = 4$$

t 如果交换这个 ISBN 其中的两位数字(变成 798-897283571-4),校验码可以发现这个错误吗?

U 如果交换数字 7 和 2(变成 978-892783571-4),校验码可以发现这个错误吗?

V 如果颠倒一个 13 位 ISBN 中连续三个数字,校验码可以发现这个错误吗(比如颠倒 978-897283571-4 的前三位数字得到 879-897283571-4)?

这样的校验码还被用于日用商品的条形码中,而且生成校验码的基本公式也与前述相差无几。商品通过结账处的扫描枪时,扫描枪将检验读入的条形码校验值是否符合计算结果,如果不符合,扫描枪将鸣声报错,收银员便知道他们需要再扫一次条形码。

t If you swap the first two digits in this ISBN (giving 798-897283571-4), will the error be detected?

U If you swap the digits 7 and 2 in the ISBN (giving 978-892783571-4), will the error be detected?

V If you reverse three consecutive digits in a 13-digit ISBN, will it be detected (for example, reversing the first 3 digits in 978-897283571-4 gives 879-897283571-4)?

Another example of the use of a check digit is the bar codes on grocery items, which are usually based on the same kind of formula. When a product is scanned at a checkout, the scanner checks that the bar code check digit that is read is the same as its calculated value. If it isn't, the scanner beeps and the checkout operator knows that they need to re-scan the code.

下图中是一包 Weet-Bix 早餐燕麦包装盒上的条形码(Weet-Bix 是一个在新西兰和澳大利亚非常受欢迎的品牌)。

A barcode from a box of Weet－Bix™ breakfast cereal is shown as following.

进阶篇
Details for experts

相对现在使用的很多高级校验法,奇偶校验的确非常简单。但是这些校验法都是基于类似的原理,即通过在数据中增加额外的校验位以保证原数据的正确性,或保证至少能检测出原数据产生的错误。一个广泛应用的纠错码是里德所罗门码(Reed-Solomon code),它基于用多项式计算来表达数据,被用于硬盘、CD、DVD上的数据存储,以及调制解调器和网络上的数据传输。在本章小游戏中使用的奇偶校验法则对 RAID 相当有用,可以拿一块磁盘专门用来储存奇偶校验位。

The parity method is relatively simple, and there are better methods that are now used, but they work on the same principle of adding extra bits to the data to allow it to be corrected, or at least to detect errors. A widely used method is the "**Reed-Solomon code**", which is used on hard disk drives, CD and DVD discs, as well as for transmitting data on modems and networks. The Reed-Solomon code is based on representing the data in a polynomial function. The simple parity method in this activity is still useful for RAID disks, where a whole disk can be devoted to parity bits!

你或许觉得疑惑,为什么 10 位的 ISBN 要除以 11 来计算校验码,除以 10 不是更方便吗。这主要是因为 11 是质数,无论得到怎样的总和它都能影响到余数值。但改成除以 10 之后,就只能影响

You may be wondering why the 10-digit ISBN divides by 11 to calculate the check digit, when dividing by 10 would be easier. The main reason is that 11 is a prime number, and the remainder

结果的最后一位了。或许在本章的例子中这种影响尚不明显，但是某些时候，除以一个质数可以避免意外得到大量同一的计算结果，否则会大大降低错误被检测出来的可能性。

is influenced by all the digits in the sum, whereas dividing by 10 just uses the right-hand digit of the sum. It isn't so important in this case, but in some situations dividing by a prime number can avoid accidentally having lots of input values giving the same check digit, in which case it would be less likely that an error is detected.

条形码中的校验位有时候又被称作校验和(checksum)或哈希总和(Hash total)，即一个基于全部数据之和，却和具体各位上的数字没有明显关系的数值(即混序组合全部数据)。相关技术同样也被应用于搜索(请参见第 9 章的哈希法)。

The check digit used for the bar codes is sometimes referred to as a "**checksum**" or "**hash total**"—a number based on all the data, with no obvious relationship to the digits (i. e. all the digits are hashed together). A closely related technique is used for searching (see the hash method in Topic 9).

有趣的事
Curiosity

在条形码中，前几位数字表示货品原产地。例如前面提到的早餐燕麦的条形码，开头是 94，代表来自新西兰。然而，不论哪个国家的书刊编码均以 978 或 979 打头。如果你查询下这个号码对应的国家，你将会发现它代表"书国"，一个人们想像中的国度！

The first few digits of a product's barcode tell what country the product originates from. For example, the barcode above begins with 94, which is the code for New Zealand products (in this case, a breakfast cereal). However, all book numbers begin with the code 978 or 979, no matter which country they come from. If you look up which country corresponds to these codes, you'll find that it's called "bookland", an imaginary place!

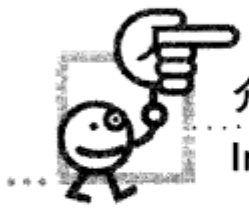
第7章

信息

0 Information 1

与计算机和计算机系统相关的工作往往称为 IT（信息技术）、ICT（信息通信技术）或 IS（信息系统），它们共同的关键词是信息(information)。信息是如此重要，我们必须好好度量它，最直接的方法就是去计算用于储存信息的比特数或字节数，但这样计算有时候挺误导人的。这一章我们将介绍一种度量信息量的更好方法。

Working with computers and computer systems is often called IT (Information Technology), ICT (Information and Communications Technology) or IS (Information Systems). What these all have in common is information. And if information is so important, we ought to be able to measure it. The obvious way is to count how many bits or bytes are used to store the information, but this can be very misleading. This topic introduces a better way of measuring information content.



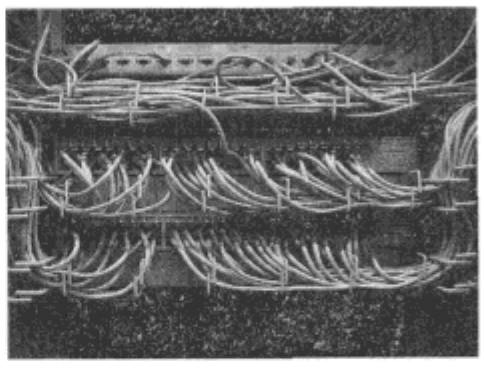
介绍
Introduction

本章我们学习的是有关信息的知识。了解一条消息中包含了多少信息是很重要的事情，因为它决定了将使用多大空间来储存这条消息，并且也有助于我们开发出更好的查错和纠错系统以及用于隐藏信息的加密码。

In this topic we will learn about information. Knowing how much information is in a message is important because that determines how much space is needed to store the message. It also helps us to develop good error detection and correction systems for it. It can also be important for developing secret codes for hiding the information.

事实上，度量信息的困难程度会超出你的想象。看下面两张图片，一张是一个大型网络系统的布线图，另一张是空白图。虽然两图的像素数相等，但对你来说，第一张图或许看起来信息更为丰富！

It's surprisingly hard to measure information. Look at the two photographs below. One shows the details of the wiring for a large network system, while the other is completely white. The white one has the same number of pixels, yet you would probably agree that there is more information in the one on the left!



a 对你而言信息是什么呢？能举出几个例子吗？

a What do you think information is? Can you think of any examples?

b 我们能用什么方法来度量一本书中包含的信息呢？

b How do you think we could measure how much information there is in a book?

下面来讲一个非常关键的概念。计算机科学家们通过估测一段消息中的"惊奇值"来衡量该消息中所含有的信息量。说说你熟悉的事情——比如，一个总是步行上学的朋友告诉你"我今天走路上学了"，对你而言，这句话没有任何信息含量，因为它并不会让你觉得意外。相反，如果你的朋友说"我今天搭了一架直升飞机来上学"，这个就颇为让人意外了，因此其中蕴含了不少信息。

Here's the big idea. Computer scientists measure the amount of information in a message by gauging how surprising it is. Telling you something you know already—for example, when a friend who always walks to school says "I walked to school today"—doesn't give you any information, because it isn't surprising. If your friend said instead, "I got a ride to school in a helicopter today," that would be surprising, and would therefore convey a lot of information.

C 你会如何排列下列书籍，从信息量最高的到信息量最低的？

- 1000 页写满"废话 废话 废话"的纸
- 1000 页的电话簿
- 500 页的电话簿
- 1000 页空白纸

C How would you order these books from the one with the most information to the one with the least?

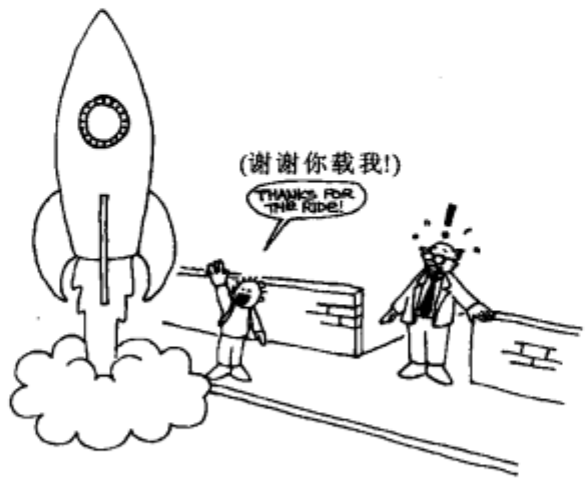
- 1000 pages of the words "blah, blah, blah" repeated over and over
- 1000 pages of a telephone book
- 500 pages of a telephone book
- 1000 pages of blank paper

d 你认为应该怎样去度量一条信息的惊奇值呢？

d How do you think we could measure how surprising a message is?

一个用来衡量信息本身惊奇度的方法，是通过旁人猜出信息的难易程度。如果你的朋友说"猜猜我今天怎么来学校的"，而他正好是走来上学的，你很可能第一次就猜中了。如果要猜中搭乘直升飞机来的话，大概就需要多猜几次了，如果是乘坐宇宙飞船的话，大概会更难猜。

One way to measure surprise is to see how hard it is for someone to guess the information. If your friend says, "Guess how I got to school today," and they had walked, you would probably guess right first time. It might take a few more guesses before you got to a helicopter, and even more if they had travelled by spaceship.



对你而言,消息或许是一张小便条或一条手机短信,可是作为计算机科学家衡量信息量的对象,消息(message)可能是一个单独数字(或许是今天的气温)也可以是一份 10 000 页的说明书;它可能指图片中某个像素的颜色,或是一张拥有全部细节的高精度照片。总之,所有的事物都可以成为消息!

Computer scientists talk about measuring the information content of a **message**. You probably think of a message as a small note to someone, or an SMS. But when scientists use the word it could be anything from a single number (perhaps today's temperature) to a 10 000 page instruction manual. It might be the color of a single pixel in a picture, or it could be all of the pixels in a complicated photo. All these things are messages!

小游戏 7.1 Yes/No 问题

Activity 7.1 Yes and no questions

我们将用猜中消息的次数来衡量一条消息中蕴含了多少信息,先从猜数字的游戏开始吧。做这个游戏你需要一位搭档,让他/她从 1～100 中任选一个数字,而你来猜——这个数字就是我们将要测量拥有多少信息量的"消息"。你可以向你的搭档提出任何问题,但你的搭档只能回答"Yes"或"No"。试着用最少的问题来得出正确的答案。下表中的数据会对你有所帮助,不妨拿两个标记(比如两枚硬币)放在 1 和 100 上,代表目标数字落在 1～100 区间之内,然后通过提问

To see how much information is in a message, we are going to measure the number of guesses it takes to find a secret—for a start, the secret will be a number. You need to get a partner to think of a number between 1 and 100 that you are going to guess—the number is the message and we will measure how much information there is in this message. You can ask your partner any question to try to guess what the number is, but your partner can only answer 'yes' or 'no'. Try to guess the number

来缩小数值的范围。比如，提问“这个数字小于50吗”得到答案“No”后，你就能把标记从1移动到50上，意味着目标数字落在50～100区间内。

with as few questions as possible. You may find the following table of numbers helpful—you can put two markers (such as two coins) on the 1 and 100 squares to show that you only know that the number is between 1 and 100, and then ask questions to narrow down the range. For example, if you ask "is the number less than 50", and the answer is no, then the marker on square 1 can move to square 50, so show that the number is between 50 and 100.

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100

e 你需要提几个问题才能猜对答案呢？（请对比同班其他人的成绩）

f 如果你提问的策略是“是1吗？是2吗？是3吗？是4吗？……”，则一般需要问多少次才能得到正确答案？

g 如何提问能够最快猜中答案？使用这种方案你最多需要问多少个问题？

e How many questions do you need to ask to guess the number? (You can compare your result with other people in your class.)

f If your strategy was to ask "Is it 1? Is it 2? Is it 3? Is it 4?..." how many questions would you typically have to ask?

g What is the best strategy for finding the number, and what is the maximum number of questions you need to ask for that strategy?

现在让我们变个花样再玩一次这个游戏。让你的搭档选择他/她出生年份的最后两位数字作为目标数字(例如,如果他出生于 1998 年便选 98),而你依旧要从 100 个数字中来猜出答案。

这次你一共提了多少个问题呢?

请记住信息量和消息的"惊奇值"或可能性是如何关联的,刚才就算你从来没有见过你的搭档,你也能(通过他的面相)猜出他出生的年份。

这个游戏中另一个有趣的现象是,如果数字的取值范围被扩大到 100～1000,你花费的猜测次数并不会因此增多 10 倍,而是只要多问 3 个问题就可以了! 事实上,每当数字的取值范围扩大一倍,只用多问一个问题即可,因为每一个问题(猜一次)都能将目标数字所在的取值范围缩小一半。

让你的搭档从 1～10000 中任选一个数字。这个范围看起来很广吧,看看你能多快猜出答案。

在这个游戏中,我们用猜中目标数字所需要的猜测次数来衡量一条消息的信息量,而每次猜测都只能用 Yes 或 No 来进行提示,因此也可以用二进制位 0 和 1 来取代这两种提示答案,从而将比特作为计量信息量的单位。比如,14 个 Yes/No 问题便可以锁定一个 1～10000 之间的数字,那么这条消息的信息量便为 14 比特(这个数字是任意选取的)。

Now we will repeat the exercise, but with a twist... have your partner choose a number that is the last two digits of the year they were born in (for example, someone born in 1998 would choose the number 98). There are still 100 possible numbers that you have to guess from!

How many guesses does it take now?

Remember how the information content is related to the "surprise" or probability of a message. Even if you haven't met the person before, you can probably have a good guess at when they were born.

Another interesting thing about information is that if the range is increased from 100 to 1000 it doesn't take 10 times the effort to guess a number—just three more questions are needed. Every time the range doubles you just need one more question to find the answer, because you can halve the range of possible answers with each question.

Have your partner think of a number between 1 and 10 000. It seems like a very big range, but see how quickly you can guess it.

We are measuring the amount of information in a message using the number of guesses here. The guesses have only two answers, yes and no, which could be represented by the bits 0 and 1. For this reason, the unit of measurement for information is usually bits. For example, you can guess a number between 1 and 10 000 with 14 Yes/No questions, so we can say that the information content of that kind of message is about 14 bits (if the number is chosen randomly).

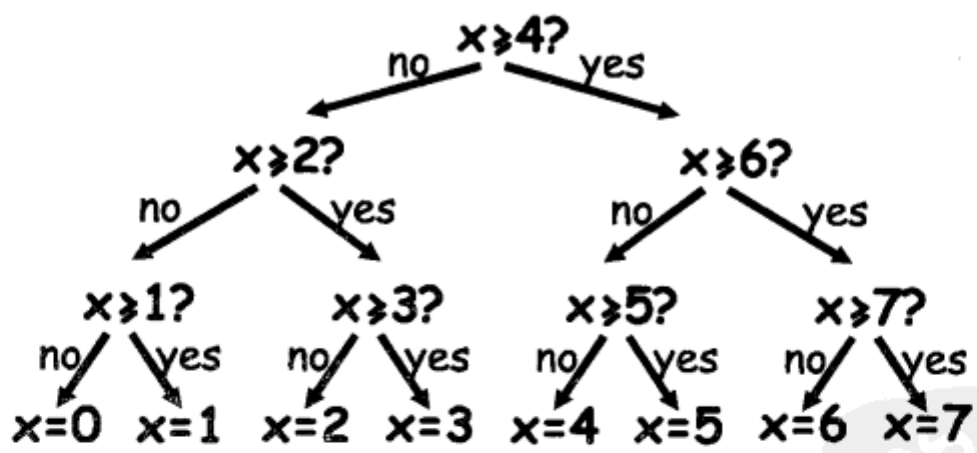
小游戏 7.2 决策树
Activity 7.2 Decision Trees

如果你和你的搭档在提问方式上已经达成共识，那么你甚至可以不问任何问题便得到一条消息。

If you and your partner already agree on a strategy for asking the questions, you can receive a message without having to ask anything.

比如，这里有一张被称为决策树(decision tree)的图，可用来猜测 0～7 之间的数字，方法是从树的顶端开始提问(x≥4 吗?)，然后根据得到的答案沿着树枝的走向提出下一个问题。

For example, here is a chart called a **decision tree** that can be used for guessing a number between 0 and 7. You start by asking the question at the top of the tree (is x≥4?), and from each question follow the branch corresponding to their answer.



例如，第一个问题是“x≥4 吗?”如果答案为“No”，则沿着树枝走向左边；因此第二个问题便为“x≥2 吗?”如果答案是“Yes”，接下来的问题即为“x≥3 吗?”，如果得到“No”的答案，你便能猜出对方选择的数字为 2。

For example, you would first ask “is x≥4?”, and if the answer is “no”, follow the branch to the left. The next question is therefore “x≥2?”, and if the answer is “yes”, you would ask “x≥3?”. If the answer to that is no, then you know the person was thinking of the number 2.

如果目标数字为 5，对方给出的 Yes/No 回答应该是什么？

What are the Yes/No decisions, or answers, needed to guess the number 5?

K 如果目标数字在0到7之间，你需要多少次Yes/No提示？

I 让我们来看一些有趣的东西，在上图决策树的最后一行，将数字0，1，2，3……均用3个比特的二进制数来表示，如果令决策树中的no＝0且yes＝1的话，仔细看看你会发现什么？

让我们这样来理解，这些二进制数字就是问题的一系列Yes/No答案，如果每个备选数字出现的概率相同的话，那么得到目标数字所需猜测的次数或目标数字中包含的信息量就等于用二进制表示该数字所使用的比特数。

m 设计一棵决策树，用来猜中值域范围为0～15的目标数字。（提示：从底部开始向上设计会更容易哦。）

K How many Yes/No decisions do you need to guess any number between 0 and 7?

I We will now see something fascinating. Underneath the numbers 0, 1, 2, 3 ... in the final row of the tree, use three bits to write each number in binary. Look closely at the tree. If no = 0 and yes = 1, what do you see?

Looking at it this way, we can think of the binary numbers that we did in Topic 1 as just a series of Yes/No answers to some questions. The number of guesses, or the amount of information in a number, is exactly equal to the number of bits needed to represent a number if every number is equally likely.

m Design your own decision tree for guessing numbers between 0 and 15. (Hint: it may be easier to write the numbers at the bottom of the tree and work up.)

小游戏7.3 丢失的文字
Activity 7.3 Missing text

刚才的游戏中用Yes/No决策的次数来测量信息量的方法适用于任何数据，包括文本。

n 假设让你猜测在单词“comp▮ter”中缺少的字母是什么，你第一个会猜什么呢？还有其他选择吗？

Our Yes/No measure of information can be applied to any data, including text.

n Suppose you had to guess the missing character in the word “comp▮ter”. Which letter would you guess first? Are there any other options?

o 在句子“I like to t▮…”(我想要……)中,“t”之后的字母丢失了,我们不知道再下一个字母会是什么。通过提出 Yes/No 问题来寻找丢失的字母,你会猜的第一个字母是什么呢?下一个可能猜的呢?按照答案可能性从大到小将它们列出来。

o Now consider the phrase “I like to t▮...”. The letter after the “t” is missing, and we don't know what will come after that either. If you had to ask Yes/No questions to find out what the missing letter is, which character would you suggest first? Which one next? Make a list of all the possible characters, in the order of how likely you think they are.

p 上题中缺少的字母会是“k”吗?

p Could the missing character in the previous example be a “k”?

q 拿一张报纸,用黑笔涂掉上面的一些字母,然后让你的搭档用 Yes/No 问题来猜出丢失的字母。哪些字母比较容易猜出来?哪些比较困难呢?

Th▮re is no reverse on a mot▮rcycle.
A friend ▮ mine foun▮ this out rather
▮▮matically the other day.

q Take a page of a newspaper and block out some of the characters by drawing over them with a black pen. Now get a partner to work out what the missing characters are using only Yes/No questions. Which characters are easy to guess? Which are difficult?

r 再来找一个保密的句子,让你的搭档用 Yes/No 问题依次来猜里面的每个字母(如果是中文,请使用拼音),记下每猜中一个字母需要的次数。

r Choose a secret sentence, and get a partner to guess each character one at a time using only Yes/No questions (if the sentence is in Chinese, using Pinyin is easiest). Keep a record of how many guesses they had to make for each character.

猜中每个字母所需要的次数反映了这个字母所蕴含的信息量(information content)。如果这个字母容易被猜出,说明它并不重要,且不含重要的信息;如果非常难猜出,则说明它的信息量相当高。

The number of guesses that you need to make for each character tells you the **information content** of that character. If it is obvious what the character will be, it isn't very important, and doesn't contain much information. If it is very difficult to guess, then it has high information content.

利用这个原理,计算机系统可将文本压缩成非常小的体积。你或许已经察觉了,刚才的游戏中有一些字母只须猜一两次便能得到答案。如果每一次猜测用一个比特来表示的话(Yes 相当于 1,No 相当于 0),而你也事先准备好了问题(比如,使用小游戏 7.2 中的决策树),那么你根本无须提出问题,只用发送对应的比特作为答案即可。利用这个系统,大多数字母都能只用 1 或 2 比特来传送,而在第 3 章中我们至少要用 5 比特表示一个字母,计算机中常常还会使用 8 比特甚至 16 比特来表示一个字母,显然决策树的方法比普通传送字母的方法要好得多。

Computer systems can use this idea to compress text to a smaller size. You probably found that a lot of characters required only one or two guesses to work out what they were. Each guess could be represented using one bit (1 for yes, 0 for no), and if you knew in advance what the questions would be (for example, using the decision tree in activity 7.2) then you wouldn't need to ask the questions, you could just send some bits to give the answers. Using this system, most characters can be sent using just 1 or 2 bits. This is much better than the normal system for sending characters—in Topic 3 we used at least 5 bits for each character, and saw that computers often use 8 or even 16 bits per character.

进阶篇
Details for experts

尽管在这里我们用 Yes/No 问题的数量来度量信息量,但实际上,当你知道一条消息出现的概率时,可以用更准确的公式来计算出其包含的信息量。假设一条消息出现的概率为 p,那么其信息量就是 $-\log_2 p$ 比特。比如,消息"The coin toss showed tails"(掷出的硬币为反面)出现的概率为 $\frac{1}{2}$,那么其信息量便为 $-\log_2 \frac{1}{2}$ 即 1 比特,这一结果显示的惊奇值就很小了,因为你只须用 0 比特或 1 比特就能表示掷出正面还是反面。如果一条消息是确定的(概率为 1),那么它的信息量为 0 比特,这意味着你不用发送任何数据出去,因为消息的接收方已经确认你会发送这条消息给他。

Although we've measured information here by the number of Yes/No guesses, there's a more exact formula for working out the information content of a message if you know its probability. If a message has probability p then its information content is $-\log_2 p$ bits. For example, the message "The coin toss showed tails" has probability 1/2, and $-\log_2 1/2$ is 1 bit, which is hardly surprising, since you can use a 0 or 1 bit to represent heads or tails. If a message is certain (the probability is 1) then its information content is 0 bits—you don't need to send any bits because the receiver was already certain that you were going to send that message!

这里介绍的信息量度量法来源于“信息理论”(information theory)领域,这些知识对于理解和发展数据传输,甚至生物学(如基因信息)和化学(构造分子架构)的相关技术大有裨益。有时候信息理论又被称为“香农理论”(Shannon theory),因为这个观点是克劳德香农(Claude Shannon)第一次在他的论文中提到的。

The idea of measuring information content as we've discussed here comes from the field of "Information Theory", which is useful for understanding and designing many things relating to data transmission, and even biology (such as genetic information) and chemistry (the makeup of molecules). Sometimes Information Theory is referred to as "Shannon Theory", after Claude Shannon, who first wrote about it.

有趣的事
Curiosity

著名的数学家克劳德香农于1950年首先发明了猜测文字中下一个会出现什么字母的实验。那个时候,他设计的句子中有一句是这样写的“There is no reverse on a motorcycle. A friend of mine found this out rather dramatically the other day.”(我的一个朋友有一天戏剧性地发现了摩托车是没有倒挡装置的。),他发现这句话中的许多字母仅用一次或两次便能猜出来,但是有一些则非常困难。最难猜的是句子中 reverse(倒挡装置)里面的 r 和 v,以及 dramatically(戏剧性地)中的 d。

The experiment with guessing the next character of text was first done by a famous mathematician called Claude Shannon, and published in 1950. One of the sentences that he got people to guess a character at a time was "There is no reverse on a motorcycle. A friend of mine found this out rather dramatically the other day." He found that people were able to guess many of the characters correctly the first or second time. But some were much more difficult. The most difficult ones in this sentence are the *r* and *v* in *reverse*, and the *d* in *dramatically*.

香农同时也是一名魔术师和独轮自行车杂技演员，他还发明了许多和计算机有关的点子，比如他曾做过一台机械驱动的杂耍器、一个自动弹簧高跷和其他小道具。最有趣的要数他发明的“终极机”了，这是一个外面嵌了开关的纸盒，当你碰碰这只开关，盒子会被打开，然后冒出一只机械手来再把这个开关拨回去。点子很简单，但每个碰过开关的人们都被这个装置深深吸引，因为大家都不知道将会发生什么。这也说明了这里蕴含的“信息量”极高！

在这一章的开头，我们说到科学家用来度量信息量的方式是通过衡量它的惊奇值的时候，你有没有觉得很神奇呢？如果是的话，这句话便传达出不少含义……

Shannon was also a keen juggler and unicyclist, and as well as inventing a lot of ideas relating to computing, he built mechanical juggling machines, a motorized pogo stick and other gadgets. One of the most intriguing was his “Ultimate Machine”—it was a plain box with just a switch on the outside. When you flick the switch, the box opens, a hand comes out, and flips the switch back. It is a very simple idea, but it caused quite a reaction from people who flicked the switch, not knowing what to expect. The information content of the event was very high!

Were you surprised when we told you in the introduction to this topic that computer scientists measure the amount of information in a message by determining how surprising it is? If so, that sentence conveyed a lot of information . . .

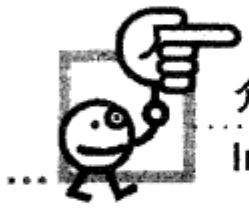
第8章

程序设计

0 Programming 1

计算机的简单指令集由有限个词汇构成，计算机执行各种操作都依照指令集中相应的指令来完成，这些指令依照一定的规则组合使用，从而形成了程序语言。就算最终会得出错误结果，计算机也只会按部就班地按照用户编写的程序执行各项指令，这未免让人有点失望。这一章将从指令的层面来讲讲程序设计的相关知识。

Computers operate according to simple instructions from a limited vocabulary. These instructions, along with the way they can be put together, form a programming "language." One of the most frustrating things about writing programs is that computers always obey the instructions to the letter, even if they produce a crazy result. This topic gives you some experience with this aspect of programming.



介绍
Introduction

在许多情况下，人们都要给出明确的指令步骤，比如编写一个菜谱、给出驾驶指示，或是介绍一个折纸步骤。在这些例子中，用只言片语描述的指令言简意赅，例如，“加一勺盐”“红绿灯的地方左转”“沿着对角线对折”。若干个这种小指令的组合，往往就能描述出完成某项任务的复杂的过程。

There are many situations where people need to be able to give clear instructions, for example, in a recipe, giving driving directions, or providing the steps needed to fold origami! In all these cases a simple language of common instructions emerges, with phrases like “add a teaspoon of salt”, “turn left at the traffic lights”, or “fold along the diagonal”. Using a small number of these simple instructions, a very complex procedure can be specified that produces the desired result.

计算机同样也能执行特定的指令集合，尽管其中的每个小指令都非常简单，可是一个接一个地输入这些指令，就能让计算机帮你解决各种各样复杂的事情。为了执行某个特定任务的指令序列称为程序(program)。这些指令和它们组合在一起的方式称为程序语言(programming language)。程序语言和我们日常说话的语言完全不一样哦，它们是专门被设计成为能告诉计算机如何做事情的有限指令集合。

Computers are also built so that they can carry out a certain set of very simple instructions. By putting these instructions together, one after another, you can make them do a tremendous variety of things—even though the instructions are individually very simple. A list of instructions that has been written to carry out a particular task is called a **program**. The set of possible instructions, along with the way they can be put together, is called a **programming language**. Programming languages are not at all like the kind of language that we speak. They have been specially designed with a limited set of possible instructions to tell computers what to do.

a 如果人们严格按照指令行动会带来什么糟糕的后果，你能想出一些例子吗？

a Can you think of any examples where it would be bad if people followed instructions exactly?

小游戏 8.1 依照指令行事
Activity 8.1 Following instructions

让我们来看看，下列精确的指令能否让你画出一幅图像。

1. 在一张白纸的中间画一个点。

2. 从白纸的左上角开始画一条直线，笔直地穿过这个点到达右下角。

3. 从白纸的左下角开始再画一条直线，笔直地穿过这个点直到右上角。

4. 在页面左边的三角形正中写下你的名字。

b 现在这幅图像看起来像什么呢？

c 如果让其他人按照这段指令，他能画出和你一样的图案来吗？当然他写下的名字会和你的不同。

在接下来的小游戏中，你需要找来一个合作的搭档。先选一张简单的图片（可以参考本章后面的图片），形容给你的搭档听，并让他按照你的描述也来试着画一画。在对方画的过程中，你可以看看他做的是否正确，并加以提醒。看看你们能做得多快多正确。（完成后，交换你们的角色，这次换你的搭档把某幅图案描述给你听。）

Let us see if we can give you precise enough directions that you can draw a particular picture:

1. Draw a dot in the center of your page.

2. Starting at the top left-hand corner of the page draw a straight line through the dot finishing at the bottom right hand corner.

3. Starting at the bottom left-hand corner of the page draw a line through the dot, finishing at the top right hand corner.

4. Write your name in the center of the triangle on the left-hand side of the page.

b What does your picture look like?

c If someone else was following those instructions, in what ways might their picture be different to yours, apart from having a different name?

For the next part of the activity you need a partner to work with. Choose a simple picture (like the ones at the end of this topic), and describe it to your partner, who should then draw it from your description. In this version of the activity, you can watch what your partner is doing and correct them if they are doing anything wrong. See how quickly and accurately you can do this. (After that, swap around and let your partner describe a picture to you.)

d 你的搭档画出的图案和你自己描述的完全一样吗？

e 如果你的搭档画错了，那么你觉得是他并没有按照你说的做，还是你并没有把指示准确地传达给他呢？

上面的游戏中你只使用了若干极为简洁的指令来进行指示。比起直接使用“画一个笑脸”“画一张哭丧的脸”这样步骤模糊笼统的指令，将复杂的图案分解成简单的小元素后再用“画一条线”“画一个圈”这样简单明确的指令来形容要更有效得多。同样地，计算机程序语言也拥有相应的简单指令，它们能用来定义各种不同的应用程序以供人们使用，不论是简单的小游戏还是复杂的国际在线购物系统。

让我们来试着画另一幅图案。这次你不能看着搭档的行动也无法纠正他的错误，但是他可以向你发问，你最好是在描述图案的时候背过身去。完成后，和你的搭档交换角色再玩一次。

f 这次你的搭档画出的图案效果如何呢？为什么当你不能看着并指导他画的时候，他就很难画出图案了呢？

d Do the pictures your partner drew look exactly the same as the one you were describing?

e If you had to correct them, was it because they didn't do what you said, or because you didn't say exactly what you wanted them to do?

You probably used a fairly small set of instructions; it's easier to specify precisely what you want by breaking complex pictures up into simple components such as "draw a line" and "draw a circle", rather than having lots of instructions like "draw a smiley face", "draw a sad face" and so on. The programming languages on computers also have relatively few instructions available, yet they are able to specify the large variety of computer programs that people use, from simple games to international online shopping systems.

Let us try another picture, but this time you aren't allowed to look at what your partner is doing, so you won't be able to correct him. However, he is allowed to ask questions. It's best if you turn your back while you describe the picture this time. Once you have finished, swap and let your partner describe a picture to you.

f How does the picture they drew compare to the one you were describing this time? Why is it harder to get your partner to draw the correct picture when you can't see what they are drawing?

让我们试着画最后一幅图案。这次你既不可以看搭档画图，你的搭档也不能向你提出任何问题。这正如为一台计算机编写程序的过程，在得到最终结果之前，程序员是无法看到编写出的各种指令的效果的。

Let us try one last picture. This time you aren't allowed to look at what your partner is doing and they are not allowed to ask questions. This form of communication is most like the one that computer programmers experience when writing programs. They give a set of instructions to the computer, and don't find out the effect of the instructions until afterwards.

g 最后这次你的搭档画出的图案效果如何？为何当你不能看着指导他们，他们也不能提问时会让这个小游戏的难度增加？

g How does the last picture your partner drew compare to the one you were describing this time? Why is it harder to get them to draw the correct picture when you can't see what they are drawing and they can't ask questions?

程序的设计至关重要，一个小小的错误往往会导致一系列的麻烦。

It is important that programs are well written. A small error can cause a lot of problems.

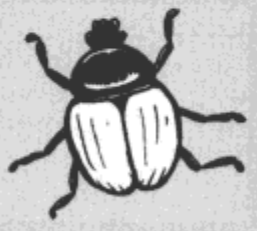
h 你能想象出在空间站发射、核电站运行或铁路轨道信号控制系统中，如果发生了一个程序错误，会导致怎样的后果吗？

h What do you think would happen if there was an error in the program of a computer in a space shuttle launch, a nuclear power plant, or the signals on a train track?

术语一点通

Errors in computer programs are commonly called **bugs** in honor (so it is said) of a moth that was once removed ("debugged") from an early electronic calculating machine in the 1940s.

我们一般将计算机程序中的错误称为 bug(臭虫)，这种说法据说是为了纪念 20 世纪 40 年代从早期的电子计算器中找到的一只飞蛾。同理，在计算机程序中纠错也可称为 debug(抓虫)。

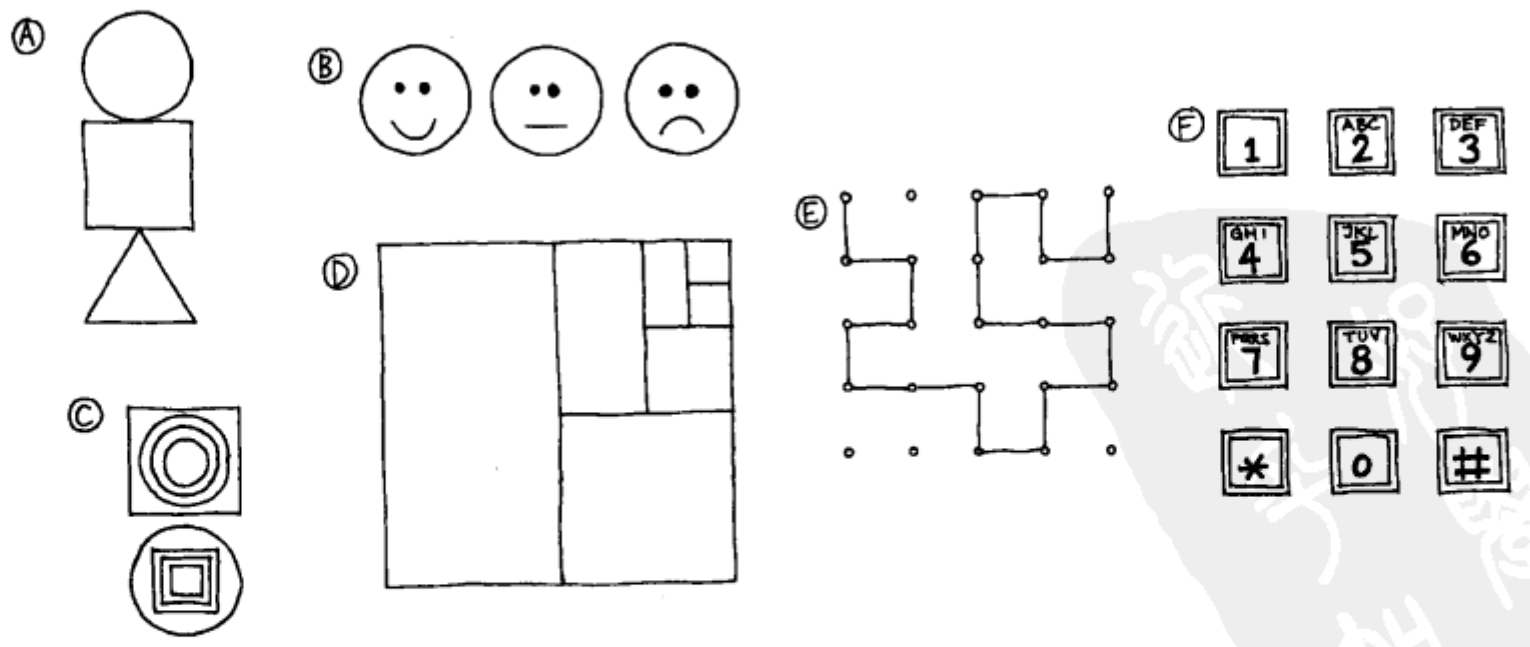


程序越复杂，它们就越容易出错。所以软件使用前必须经过细致的测试，尽量找到存在的 bug，特别应测试那些实现人机交互的重要模块。当然，如果一开始就能非常细致地来编写这个程序，那么日后测试中就不会发现太多的程序错误。因此，计算机科学家们一直试图在寻找各种各样的方式来避免程序错误，从而让它能按照预期状态运行。

The more complex the program, the more errors there are likely to be. Software needs to be tested carefully to find as many bugs as possible before it is used, especially in important systems that interact with people. It's even better if the program can be written very carefully in the first place, so that there won't be any bugs in it to be discovered later. For this reason, computer scientists look for ways to prove that a computer program does not have any bugs and will behave as expected.

游戏中供参考的小图案 我们为你准备了一些图形，如果你的搭档已经事先看过这一页，那你还是最好自己想出其他的图案吧（在你开始将图案描述给搭档听之前，先找个地方偷偷画出来吧）。

Ideas for pictures to describe Here are some possible pictures you could describe, although if you partner has seen this page then it's best if you make up your own picture (draw it somewhere secret before you describe it).



- android与iphone及ipad开发书籍 ----------持续不断更新中.....
- c、c++、c#语言pdf书籍及vip视频教程 c、c++、c#、vc等-------持续不断更新中.....
- delphi《书籍》及《视频》教程 ----------持续不断更新中.....
- E网情深VIP系列视频教程 黑客破解菜鸟修练班，VB编程学习班，仿站学习培训，免杀培训，个人系统攻防系列教程，服务器搭建学习班，PHOTOSHOP平面设计班，基础制作论坛（论坛网站搭建），网赚系列教程，网站建设教程，网站漏洞基础，远程控制教程，软件破解班，脚本漏洞提权班
- IT9网络学院VIP系列视频教程 免杀培训班，VMware虚拟机，零基础学习C语言，网游外挂开发精品系列语音教程（外挂教程学习必备研修31课全），VB语言教程30课全，Delphi编程到精通，远程控制软件，加密解密班，网络安全与黑客攻防培训，从入门到精通完整系统化学习C++编程，从入门到精通零基础学习汇编，wordpress教程(个人博客系统49课全)，外行人做易语言盗号和钓鱼程序语音教程 网址：WLSAM168.400GB.COM
- Java书籍 ----------持续不断更新中.....
- photoshop、CorelDRAW、AutocAD等图像处理书籍及vip视频教程 -------持续不断更新中.....
- powerbuilder书籍大全
- Visual Basic语言vip视频教程及pdf书籍 ----------持续不断更新中.....
- windows、linux系统开发、系统封装等pdf书籍及VIP视频教程 ----------持续不断更新中.....
- 《3DS Max》pdf书籍
- 《汇编语言》、《反汇编》及《调试》pdf书籍及vip视频教程 ----------持续不断更新中.....
- 《电子书、电子书、还是电子书》pdf专题库 编程开发，家居美食，儿童益智，人物传记，增强记忆，快速阅读
- 信息系统项目管理师、网络工程师、系统分析师等软考类书籍
- 华中红客系列vip视频教程 脚本攻防培训班，源码免杀培训班，Css语言培训班，C语言，Dreamweaver网页设计，html网页设计培训班，PC安全班， php脚本语言培训班，VMWare虚拟机专题，webshell提权培训班，防站教程，零基础免杀培训班，刷钻速成班，脱壳破解班，外挂编写班，网络赚钱培训班，网站入侵培训班
- 外挂、驱动、逆向及封包视频教程 郁金香、独立团、夜猫论坛、天都吧、看流星论坛、一切从零开始等等
- 安全中国系列vip视频教程 易语言软件编程培训班，ASP.net网站开发项目实战培训班
- 我的收藏
- 按键精灵及TC脚本开发软件视频教程 ----------持续不断更新中.....

当前位置：/《电子书、电子书、还是电子书》pdf专题库

文件名 ⇕

PDF电子书专题库，内容详尽，每天不断更新！！

- 办公类软件使用指南
- 医学
- 历史人物传记
- 哲学宗教
- 外语资料（除英语外） （除英语外）
- 官场类小说
- 建筑工程类
- 情感生活类小说
- 政治军事
- 教育学习科普大全
- 文学理论
- 智力开发、增强记忆、快速阅读技巧大全
- 社会生活
- 科学技术
- 程序编程类
- 经济管理
- 网络安全及管理
- 网赚系列
- 美食小吃烹饪煲汤大全
- 课外读物

本网盘内容太多，持续不断更新，发布各类视频教程、pdf书籍，包括破解、加解密、外挂辅助制作，易语言培训教程、编程语言、网页制作等等，教程及书籍仅用于学习,如用于商业或非法用途的后果自负！

网址：WLSAM168.400GB.COM

- VIP培训教程，易语言黑月VIP视频教程，天都网易语言系列培训教程(100集全)，集中营易语言学习视频(80集)
- 棉猴系列vip视频教程 gh0st远控控制源码讲解教程，套接字编程，DLL程序编写，键盘监听驱动程序编写，驱动基础教程，AsyncSelect模型QQ程序教程，C++语言入门基础，NB5.5源码分析教程
- 游戏开发pdf书籍 ----------持续不断更新中.....
- 炒股投资pdf书籍及视频教程 短线高手系列，短线天王系列，操盘论道系列，翻倍黑马，看盘快速入门，庄家手法大曝光等等。 网址：WLSAM168.400GB.COM
- 热门小说集中营 傲世九重天，网游之三国时代，武动乾坤
- 甲壳虫VIP教程全集 asp教程，Delphi培训班，FLASH培训班，Java培训班，linux培训班，PHP培训班，源码免杀班，甲壳虫C++，脚本功防班，免杀班初、中、高级班，破解班，源码免杀班，脱壳班，易语言培训班，无特征码免杀，网站架构培训班，外挂高级班，外挂初级班第1、2部
- 破解、免杀、入侵、脱壳、攻防及漏洞分析系列VIP视频教程（80多部） 天草、黑客动画吧等等--------持续不断更新中....
- 网站建设相关的pdf书籍及各种vip视频教程 ----------持续不断更新中.....
- 网赚、淘宝系列vip视频教程 网赚30天新人魔鬼训练，屠龙网赚团队vip课程，站长大学网赚视频（50课全），图腾团队日赚1000元竞价行销教程，屠龙团队淘宝宝贝卖疯系列，站群网赚系列，淘宝开店视频，红星挂机日赚10元，百万流量系列，漂流瓶圣手全自动挂机引，贴吧邮件定向营销疯狂成交量月入万元
- 英语学习资料百科大全 不断更新。。。
- 饭客论坛系列VIP视频教程 脚本入侵班，黑客之免杀教程，易语言教程，无线网络攻防教程，入侵教程，delphi系列教程，黑客基础入门
- 黑客书籍 有关黑客、安全、加解密技术等等----------持续不断更新中.....
- 黑手安全网VIP系列视频教程 DIV+CSS网页布局，Dreamweaver教程，flsah动画教程，photoshop教程，跟我一起学C++课程，抓鸡
- 黑鹰、黑基、黑防、黑盾vip系列视频教程 破解提高班66课全，SQL注入，ASP注入教程，完完全全学会抓肉鸡，脱壳破解教程50课全，提权班，C语言特训班26讲全，黑客脚本特训班，黑客工具特训班，dedecms仿站教程，VC编写远控30课全，网页美工特训班，木马免杀特训班，驱动开发技术VIP培训班，外挂破解等等。

- [电脑世界的通关密语：电脑编程基础].(杉浦贤).滕永红.扫描版.pdf
- [程序语言的奥妙：算法解读（四色全彩）].(杉浦贤).李克秋.扫描版.pdf
- [差错：软件错误的致命影响].(帕伯斯).邝宇恒等.扫描版.pdf
- [算法之道（第2版）].邹恒明.扫描版.pdf
- [O'Reilly：深入学习MongoDB].(霍多罗夫).巨成等.扫描版.pdf
- [深入浅出WPF].刘铁猛.扫描版.pdf
- [Go语言·云动力（云计算时代的新型编程语言）].樊虹剑.扫描版.pdf
- [精通.NET互操作：P/ Invoke、C++ Interop和COM Interop].黄际洲等.扫描版.pdf
- [编程的奥秘：.NET软件技术学习与实践].金旭亮.扫描版.pdf
- [O'Reilly：学习OpenCV（中文版）].(布拉德斯基等).于仕琪等.扫描版.pdf
- [Go语言编程].许式伟等.扫描版.pdf 网址：WLSAM168.400GB.COM
- [MySQL技术内幕：SQL编程].姜承尧.扫描版.pdf
- [Tomcat权威指南（第2版）].(布里泰恩等).吴豪等.扫描版.pdf
- [Ext江湖].大漠穷秋.扫描版.pdf
- [IT名人堂·Oracle DBA突击：帮你赢得一份DBA职位].张晓明.扫描版.pdf

Total: 77 1 2 3 4 5 6 ›

进阶篇
Details for experts

程序员可以选择各种各样的程序设计语言为计算机编程，其中常见的有 C、C++、C#、Perl、Python、PHP、Javascript、BASIC 和 SQL。不同的程序语言拥有不同的命令集和规则，它们各有优劣，因此通常情况下被使用在不同的环境下。例如，在网页浏览器中 Javascript 是个相当适用的语言；Java 和 C++ 则更适合实现一些大型项目，这些项目往往被划分成多个功能模块，并由不同的程序员编程实现；另外，Alice[7] 和 Squeak[8] 则非常适合应用于程序设计的教学中。

There are many different programming languages that programmers use to give instructions to computers. Common ones include Java, C, C++, C#, Perl, Python, PHP, Javascript, BASIC, and SQL. Each language has different commands and rules for specifying what the computer should do, and there are advantages and disadvantages with each language. Often they get used in different circumstances. For example, Javascript is very useful for running in web browsers, while Java and C++ are good for large projects where many different people have to write different parts of the system. There are languages like Alice and Squeak which are designed to be good for teaching programming to students.

有趣的事
Curiosity

有些程序十分庞大，它们由数百万条指令组成，研发人员往往需要用多年的时间来编写程序。这些程序往往太过巨大，以至于没有一个人能了解程序的全貌。

Some programs are extremely large, with millions of instructions. Some take teams of people years to write—and are so big that no single person can possibly understand the whole thing.

[7] Alice，一种面向儿童的教学程序语言。

[8] Squeak，思快客，历史上公认为第二个面向对象的程序设计语言。

任何一个小错误在程序中都有可能造成严重的后果。比如，电话系统中一个小的程序错误会导致大范围的通信信号中断；有些程序错误则导致停电；有一些程序错误还曾扰乱机场的行李分配系统，以至于引起众多旅客的不便并为之浪费了巨额开销。1996 年，欧洲航空局斥资 1 亿美元打造的阿波罗 5 号航空飞行器在发射后一分钟炸毁，就是飞行器导航计算机中的一个程序错误带来的灾难。

And the consequences of an error in a program can be very serious. For example, an error once disabled a large part of the telephone system—for telephone systems are run by computers. Other errors have caused power blackouts. Others have played havoc with the luggage handling system in major airports, causing enormous frustration for passengers and expensive delays for the airplanes. In 1996, the European Space Agency's US $1 billion prototype Ariane 5 rocket was destroyed one minute after launch due to a program error in the on-board guidance computer.

如果说一个庞大的程序没有人可以从头到尾完全理解，那么怎样能保证每一个小指令都正确无误呢？对于计算机科学来说，这是一个相当困难的问题，也是一个非常关键的问题。

How can you have a program so large that no-one can possibly understand it all, and yet be sure that every single thing is correct? This is a hard problem in computer science—and an important one.

第9章

搜索

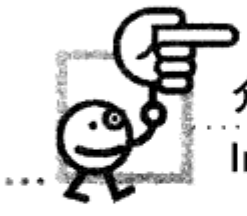
0

Searching

1

计算机最重要的功能之一就是在浩瀚的数据中找到用户所要的信息。逐一查看全部的数据很容易找到你想要的东西，但是计算机的速度还并没有快到能瞬间完成这一过程的程度——而且等待计算机查找的数据集往往是异常庞大的，因此我们需要更快捷更有效的搜索方式。这一章中的小游戏将向大家展示三种不同的搜索方式：线性搜索，二分搜索和哈希搜索。

One important use of computers is to find information in large collections of data. It's easy to find things by painstakingly looking through everything, one piece of data at a time. But although computers are fast, they're not *that* fast—and data collections can be very big indeed. We need quicker and more efficient ways of searching. This activity demonstrates three different search methods: linear searching, binary searching and hashing.



介绍
Introduction

通过前几章的小游戏，我们已经学习了许多关于数据的知识。现在让我们来专注于算法(algorithm)。算法是完成特定任务的方法，它通常由一个指令序列来描述，即计算机程序。

In our previous activities, we've been learning mostly about data. We are now going to focus on **algorithms**. An algorithm is just a method for completing a task, and is usually written as a list of instructions: that is, a computer program.

有些算法的效率明显高于其他算法。当然计算机本身的计算速度已经不可小觑了，但是有时候它们需要完成的任务非常巨大，以至于处理起来需要很长时间才能完成。人们往往无法容忍在计算机前漫长的等待时间，你一定试过坐在计算机前面苦等一个半天都打不开的网页那滋味。用户永远希望系统能反应神速，于是发明快速的算法就变得尤为重要。

Some algorithms are faster than others. Computers work really quickly, but the kind of tasks they need to complete are often so big that they take a long time to finish. People get impatient with computers—you probably know what it's like to have to wait a long time for a web site. Customers prefer systems that respond quickly. That makes it important to discover fast algorithms.

举几个需要提高算法效率的例子。

Here are some examples of algorithms that need to be fast.

1. 在万维网中寻找有你名字或其他指定内容的网页。

1. Finding all pages that contain your name—or anything else—on the World-Wide Web.

2. 在最节省燃料的情况下，寻找货车投递包裹的最佳路径。

2. Finding out the best route for a delivery van to take to drop off some parcels, to economize on fuel.

3. 帮学校设计出一张最合理的时间表，保证在一天之内所有的班级都有课程安排。

3. Working out the most efficient timetable for a school, so that all the classes can happen within the school day.

上述的每个问题都很复杂。

下面我们来探讨一下在一长串数列中查找特定数字的不同算法。虽然我们这里举例的对象是数字，不过通过前几章的学习，你已经了解到计算机中的所有东西都能用数字表达，因此同样的算法也能用于搜索任何类型的数据。我们将计算机需要搜索到的数据，比如文字、条码号或作者名字，称为搜索关键词(search key)。

对于存储了大量数据的计算机而言，由于需要在这些数据中快速读出信息，搜索功能是相当重要的。世界上最庞大的搜索问题是在网络世界中搜索信息，网络搜索引擎能实现同时为大量在线用户在数以百万计的网页中进行高速查询，这无疑是一个令人惊讶的事实，而它们能做到这点无非是因为使用了正确的算法。

我们将通过"战舰游戏"来学习如何进行搜索。首先你需要一个搭档，你们每个人都拥有一艘秘密战舰，其中一个人需要在所有的战舰中找到属于对方的那艘秘密战舰。而你们的搜索关键词(需要查找的数据)就是这艘秘密战舰的号码。

Each of these is quite a complicated problem.

We are going to explore different algorithms for **searching** for a particular number in a long list of numbers. Although we're talking about numbers, you have already learned that everything that computers work with can be represented as numbers, and the same algorithms can be used for searching for almost any kind of data. The data that the computer is asked to look up, such as a word, a bar code number or an author's name, is called the **search key**.

Searching is important because computers store a lot of information, and they need to be able to sift through it quickly. One of the biggest search problems in the world is finding things on the Internet. Yet Internet search engines succeed in searching billions of web pages in a fraction of a second, and they do it for lots of people simultaneously. It's amazing that they can do this, and it only works because they use the right searching algorithm.

We will learn about searching by playing a game that's sometimes called Battleships. You need to work in pairs. Each person has a secret battleship, and the other person has to find it amongst all the other battleships. Their search key—the data that they have to look up—is the number of the secret battleship.

小游戏 9.1　线性搜索战舰
Activity 9.1　Linear search battleships

让我们从最简单的版本来玩起，简单但或许会让你最有挫败感！

你和你的对手每人拿 25 张卡片，每张卡片上均写了随机的 4 位数的数字（如果每人手拿 100 张卡片这个游戏会更好玩，如果你有时间准备那么多张卡片的话，不妨试一下。没时间的话，保证人手不少于 15 张卡片）。你可以使用这一章后面帮你准备好的卡片，直接复印下来或从网站上打印。要保证卡片的纸张不透光，以免你的对手从背面读到你卡片上的数字了。最好保证每张卡片上的数字都不一样。

每个人从手上的卡片中挑选一张作为自己的“战舰”，并告诉对手这艘战舰的数字（最好写下来），然后洗牌（包括战舰那张）并将卡片的背面朝上、乱序摊开在你面前。你的对手也要这样做。

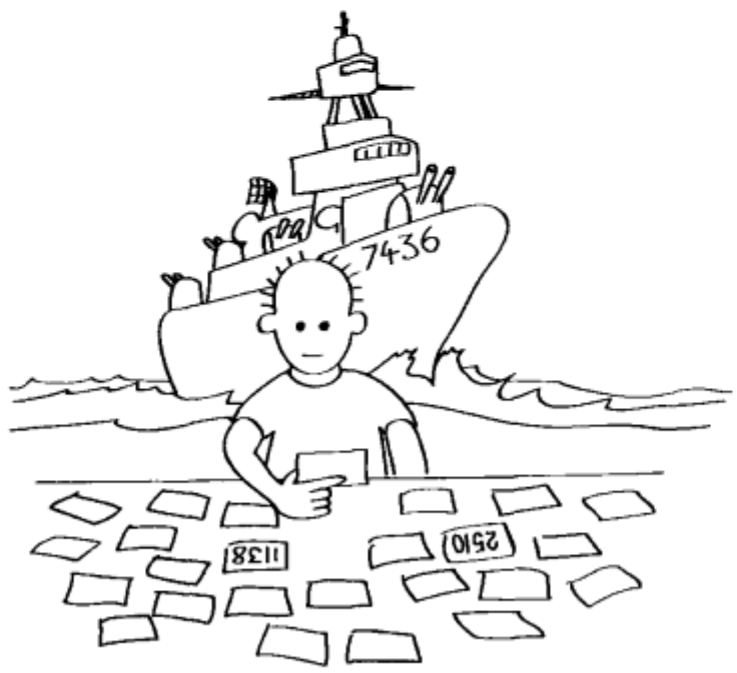
每个人依次从对方那里选一张卡片翻开，第一个找到对方战舰的人算作胜利。

This is the easiest version of the game to set up, but probably the most frustrating one to play!

Both you and your opponent have a set of 25 cards that have random 4-digit numbers written on them (the game is even better with around 100 cards, if you have time to prepare this many, or you could use as few as 15 to keep it simple). You can copy the cards at the end of this topic, or print them from website, or just write your own numbers on some cards. Each card is a battleship that your opponent must find. Make sure that the card is thick enough that your opponent can't see the number on the card when it is face down! Also it's best if you don't repeat any numbers.

Each player chooses one of their cards as the "battleship" that their opponent needs to find, and tells that number to their opponent (it's best to write it down!) Now shuffle your 25 cards (including the chosen one) and place them face down in front of you, in no particular order. Your opponent does the same with their cards.

Each player now takes turns at choosing one of their opponent's cards, and that card is turned over. The winner is the first person to find their opponent's chosen battleship.



a 将每个人找到战舰花费的次数记录下来。如果有时间的话，将这个游戏多玩几次，记录每次你需要猜几次才能成功，其中最多花费的次数是多少？

b 如果一共 25 张卡片，你最多花费的猜测次数是多少？

c 最少猜了几次就成功了？

d 第一次就能猜对的几率有多大呢？

e 如果秘密战舰并不在这些卡片当中怎么办？要猜多少次才能发现这个状况？

f 这个游戏中，你认为一般来说要猜多少次才能猜中秘密战舰？

a Make a note of how many guesses each person had to make before they found the right battleship. If you have time, play the game several times, taking note of how many guesses you made each time. What was the highest number of guesses that you had to make?

b If there are 25 cards, what is the largest number of guesses you would ever have to make?

c What is the lowest number of guesses that you would ever have to make?

d What is the chance that you would get the right card on the very first guess?

e What would happen if the secret ship wasn't there? How many guesses would it take to find that out?

f About how many guesses do you think it will usually take to find the secret ship in this game?

如果一共有 100 艘战舰，你认为一般需要猜多少次才能赢得游戏？

计算机的运行速度很快，你或许认为当计算机进行搜索时，应该从储存数据的开头开始找，直到找到指定数据时结束查找，这种方式被称为线性搜索，也是我们在战舰游戏中用到的方法。但这种算法就算对计算机来说都是非常慢的。

举例来说，假如一家超市货架上放有 10 000 种不同货品，当收银员扫描一件货物的条形码时，计算机需要在 10 000 种不同记录中去寻找这件商品的名称和价格，就算它能在一秒内扫描 1000 次，查完全部的货物也要耗费 10 秒钟。这时客人或许已经等得不耐烦了，也许下次就去别家超市购物了(如果你觉得 10 秒钟很短的话，试试看说任何话的时候停 10 秒的感觉)。幸运的是，下一个游戏中我们将有更好的方法来进行搜索。

If there were 100 battleships, how many guesses do you think it would usually take?

Computers can process information very quickly, and you might think that to find something they should just start at the beginning of their storage and keep looking until the desired information is found. This is called **Linear Searching**, and is what we did in this game of Battleship. But this algorithm is very slow—even for computers.

For example, suppose a supermarket has 10 000 different products on its shelves. When a bar code is scanned at a checkout, the computer must look through up to 10 000 numbers to find the product name and price. Even if it takes only one thousandth of a second to check each code, ten seconds would be needed to go through the whole list. The customers would become very impatient, and probably go to a different supermarket next time (if you don't think 10 seconds is very long, try waiting for 10 seconds before each thing you say in a conversation)! Luckily, there is a better way to search, which we will use in the next activity.

小游戏 9.2　二分法搜索战舰
Activity 9.2　Binary search battleships

下面我们将来使用一种更高效的搜索算法，名叫二分搜索(binary search)。这个算法和之前在第 7 章(信息)中用来猜测 1 到 100 中一个数字的方法类似。

We will now try a faster searching algorithm called **Binary Search**. This is a similar algorithm to the one you may have used in Topic 7 (Information) to guess a number from 1 to 100.

在我们开始游戏前，请先把手上的战舰从小到大排列后放在一行中（最小的数字放在最左边，最大的数字放在最右边）。当你的对手也排列好他手上的战舰后，你可以从正中间的位置选一艘战舰，并进一步判断对方的秘密战舰位于它的左边还是右边。如果对方秘密战舰的数字比你选出的中间战舰大的话，那就说明秘密战舰在右边数字较大的队列这一半；如果对方秘密战舰的数字比你选出的中间战舰小的话，那么说明秘密战舰在左边数字较小的队列那一半。重复这个过程，你便能逐渐缩小秘密战舰所在的范围。

Before we play this time you will sort the battleships from lowest to highest, and put them in a line (lowest numbers on the left, largest on the right). When your opponent's battleships are sorted and you pick a battleship in the middle, you can always figure out on which half the secret ship must be in. If the secret ship has a higher number than the middle ship, then you know the secret ship is somewhere in the higher half; if the secret ship has a lower number, then the secret ship is somewhere in the lower half. You then repeat this process to narrow down the ship.

玩法是这样的，准备 25 张卡片（不要让你的对手看到），按照最小的在顶部，最大的在底部的顺序排列好。与此同时，选一张作为你的秘密战舰。在你面前将卡片背朝上排列成一行。为了避免你的对手选错端，你可以告诉他哪一端放有数字最小的卡片。

To play the game, take your 25 cards, and (without showing your opponent) sort them into order with the smallest one on top and the largest on the bottom. At the same time, choose one as your secret ship. Now put the cards out upside down in front of you in a line. You should remind your opponent which end of the line is the smallest number, since they are probably looking at it from the other side.

让我们开始轮流猜一下对方秘密战舰的位置吧。

Now take turns guessing where the secret ship is located.

h 这次你一共花了几次猜中对手的秘密战舰的呢？将这个次数作为你在这个游戏中的得分。

h How many shots did it take to locate your partner's ship this time? This is your score for the game.

i 如果你是和班上的同学们在课堂上玩这个游戏，互相交流一下每个人都用了多少次猜中对方的秘密战舰。

i If you are in a class, get everyone to share the number of shots that they needed to take.

j 同伴们之中得到最好的分数是多少？能得到的最好(最小)的分数又会是多少呢？

j What is the best score that anyone got for this method? What is the best (minimum) possible score?

k 同伴们之中得到最差(最大)的分数是多少？在正确执行二分搜索的前提下，你可能得到的最差分数是多少？

k What is the worst score that anyone received? What is the worst (maximum) possible score when you use the Binary Search method correctly?

l 如果秘密战舰并不在队列中会发生什么情况？要猜多少次才能发现这一点？

l What would happen if the secret ship wasn't there? How many guesses would it take to find that out?

m 如果一共有 100 艘战舰的话，需要猜多少次才能赢得游戏呢？

m How many guesses would it take if there were 100 ships?

使用二分搜索法花费的猜测次数比使用线性搜索法少——只用检查队列的中间项就可锁定搜索关键词位于哪一半队列，这样一来，每猜一次相当于将待查找的目标数量减少一半。再回头看看超市的例子，如果采用二分搜索方式在 10 000件货品中查找，现在仅须用到 14 次搜索，也许就是两百分之一秒的事——快到让人无法察觉。

The binary search game should take fewer guesses than the linear search game—checking the middle item of the list will identify which half the search key is in, so each "shot" or probe will halve the number of candidate targets. Returning to the supermarket example, the 10 000 items can now be searched with only fourteen probes, which might take two hundredths of a second—hardly noticeable.

n 采用二分搜索法，在 1 000 000 个货品中需要搜索多少次才能找到目标？(提示：结果会比你想象中的少得多；如果将 1 000 000 除以 2，再除以 2，依此类推，一共需要除多少次能得到 1 呢？)

n How many probes would be needed if you had to search 1 000 000 items using the binary search method? (Hint: it isn't as many as you might think; if you divide 1 000 000 by 2, and then by 2 again, and so on, how many times can you do that before you are down to just 1 item?)

小游戏 9.3 哈希法搜索战舰
Activity 9.3 Hash search battleships

有时还会用到一个更高效的查找算法。这次，我们不是排列手中的战舰，而是将战舰代码的各数位哈希化(我们会简短地介绍一下如何操作)，从而为每艘战舰按照它的搜索关键词指定一个类。这样一来，当你要查找该关键词的时候，你便能精确定位需要查找的类(pile)。哈希算法是计算机中搜索数据的最快的方法。

There is an even faster searching algorithm that can sometimes be used. This time, instead of sorting the battleships, we will assign each one to a pile based on its search key. We do this by **hashing** the digits of the battleship's number (we'll explain how to do that shortly). Then, when you want to find those search keys, you know exactly where to look. Hashing is usually the fastest way to search for something on a computer.

一个简单的哈希函数是将战舰代码的每个数位相加，取其总和的末位数字。总和的最后一位数字决定了该把这艘战舰放置在哪一类中，因此一共有 10 类(0 到 9)。比如，编号为 2345 的战舰，将代码各位数字相加 2 + 3 + 4 + 5，得到 14。由于总和的末位为 4，所以这艘战舰将被归到数字 4 的类中。

A simple hash function is to add together the digits of the ship's number and then take the last digit of the sum. The last digit of the sum tells you which pile the ship is to be put in, so there will be 10 piles (0 to 9). For example, to locate a ship numbered 2345, add the digits 2 + 3 + 4 + 5, giving 14. The last digit of the sum is 4, so that ship will be stored in pile number 4.

```
Ship  2345 -> 2 + 3 + 4 + 5 = 14
          -> Pile number 4
```

你需要准备 10 个类别的标签放在你面前的桌子上(0 到 9)，或者将类别号直接写在对应战舰卡片的背后。然后算出你手上每张卡片的哈希值，并将它们归放在相应的类中，同时不要忘记选出你的秘密战舰哦。

You will either need to have 10 labels for the piles on the table in front of you (0 to 9), or you can write the hash number for each card on its back. Work out the hash value for all your cards, and put them in their correct piles. Choose your secret ship while you do this.

再和你的搭档玩一次战舰游戏。这次，锁定对手秘密战舰的速度应该很快了吧——比如，如果你的目标是战舰 5678，将各数位相加之后得到 26，说明你只要在对手标为 6 的类中寻找目标即可。

Now play the game again with your opponent. You should be able to locate their ship fairly quickly—for example, if you are looking for ship 5678, add its digits together to get 26, which tells you that it should be in your opponent's pile number 6.

o 这次你一共用了多少次找到对手的战舰？把次数作为这次游戏的得分。

o How many shots did it take to locate your partner's ship this time? This is your score for the game.

如果你是和班上的同学们在课堂上玩这个游戏，互相交流一下每个人都用了多少次猜中对方的秘密战舰。

If you are in a class, get everyone to share the number of shots that they needed to take.

p 用这种方法，大家的最好成绩是多少分？成绩最好（最少）能达到多少分呢？

p What is the best score that anyone got for this method? What is the best (minimum) possible score?

q 大家拿到的最低分是多少？分数最低（最大）能到多少分呢？

q What is the worst score that anyone received? What is the worst (maximum) possible score?

r 如果秘密战舰并不在队列中会发生什么情况？一共需要多少次才能发现这一状况？

r What would happen if the secret ship wasn't there? How many guesses would it take to find that out?

s 你认为一般需要猜多少次才能找到秘密战舰？

s About how many guesses do you think it will usually take to find the secret ship?

t 如果你想要让你的秘密战舰不那么容易被找到，你需要把它藏在哪一类里面？

t If you want to hide your secret ship for as long as possible, which pile should you pick?

u 如果你想要尽快找到对手的秘密战舰，你最希望它被归在哪一类里面呢？

u If you want to find your opponents secret ship as quickly as possible, in which pile do you hope it is in?

Topic 9 Searching
第 9 章 搜索

通常来说，哈希法是最佳的搜索算法，但是它的运行速度取决于类别中对象的数量和类别的数量。你可以设计不同的哈希函数，从而生成更多的类别。在计算机中，类别的数量总是比每类包含的对象数量要多，所以多数情况下，一个类中只包含一个对象，有时候甚至一个类中一个对象都没有，这也就意味着计算机通过简单计算之后，往往能直接锁定关键词储存的位置。

Hashing is often the best searching algorithm, but its speed depends upon how many items are in each pile, which also depends on how many piles there are. You can have more than 10 piles by using different ways to calculate the hash function. On computers the number of piles is usually more than the number of items being searched, so most have only one item in them, and some are even empty, which means that most of the time the computer simply calculates the hash total and goes straight to the place that the key is being stored.

你或许注意到了，哈希法和我们在第 6 章用到的校验位检测法非常类似。它们基于一个同样的原理，即每个关键词生成的“随机”数字都不尽相同，这一点保证了两种算法的成功。

You may have noticed that hashing is very similar to the check digit method that we used in Topic 6. They work on the same principle—the "random" number that is generated from the key is unlikely to be the same as the one generated from another key, which is important for the success of both techniques.

V 本章中介绍的三种搜索方法其各自的优点和缺点是什么？（提示：哪一种速度最快？哪一种需要每个对象采用特定方式排列在一起？）

V What are the advantages and disadvantages of each of the three different ways of searching? (Hint: Which is the fastest? Which require that the items are organized in some way?)

供参考的游戏卡片 你可以把它们剪下来当作你的卡片，也可以设计出属于自己的随机数字。注意不要让纸片背后透出正面的数字。你可以只做 15 到 25 张卡片，不过玩这个游戏用 100 张卡片会更有趣。

Random numbers for cards You can cut these out to use as your cards, or just make up your own random numbers. Make sure you can't see the number through the paper. You may want to use just 15 to 25 of the cards, but the game works well if you have time to work with 100 cards.

9376	7322	8987	5511	4371
3678	8724	4789	1973	2359
4279	7930	3740	2945	1549
5913	8314	0893	8751	4116
4836	5920	4625	8995	3869
3671	2447	5102	1185	0062
6855	0660	9369	9074	0874
5769	7502	7004	4921	0902
1447	2325	8315	3935	4565
2834	3107	4132	1718	0901

5848	4250	8848	0942	5666
3590	1965	4460	8762	3600
2241	9825	6072	1619	3323
0925	5762	8180	1218	9501
3294	9496	2129	5453	6859
5018	3514	6710	6681	0933
4542	3512	4419	4010	3741
4102	7721	7214	4518	4223
0936	9305	0463	8906	1136
3149	4586	4666	6171	2802

进阶篇
Details for experts

采用二分搜索法搜索关键词的速度很快，不过如果你想要增加一个关键词的话，搜索速度就明显变慢了，因为新增对象首先还需要被添加到关键词序列中去——二分搜索只能在排好序的关键词序列中发挥其效率，而在序列尾部新增一个关键词，原有的序列顺序将被打乱。实际上，可以使用一个被称为“二叉树搜索”的类似方法来解决插入新值的问题，它更容易实现数据添加，而且搜索的速度同样很快。

The binary search method is extremely fast at finding keys, but if you want to add a key it can be slow on a computer because it needs to be inserted in the list of keys-binary search only works with sorted lists of keys, and if you just add it to the end then the list won't be in sorted order. In practice, a related approach called a “binary tree” is used because it is easy to add values to it, but it is still usually very fast to search.

在哈希搜索法中我们使用的“类”一般被称为“桶”(bucket)。当你锁定一个桶后，在桶中采用的其实是线性搜索。通常来说，桶中的线性搜索对运行速度不会有太大影响，因为每个桶里面包含的对象数量不多，但一旦桶内包含的对象数较多的话，我们便遇到了另一个新的搜索问题——在搜索中搜索。

The “piles” that we referred to for hash searching are usually called “buckets”. Once we chose a bucket (pile) using the hash number, we then did a linear search in the bucket. Usually that works fine because there are so few keys in each bucket, but if it's possible that the bucket contains a lot of keys, we end up with a whole new searching problem-a search within a search.

二叉树搜索和哈希搜索被广泛应用于数据库中的信息查找。例如，当你用信用卡刷卡(系统需要在百万条信用卡信息中查找你的信用卡号)、当你登录一个网站(系统需要在数据库中找到你的登录名并看看是否匹配密码和其他预留信息)或当你在其他应用软件中使用查找功能的时候。

Methods related to the binary tree search and the hash search are widely used in databases to look up information. This might be when you swipe a credit card at a checkout (the system needs to look up the credit card number amongst the millions of cards on file), when you log into a web site (it needs to look up your login name to find out if your password matches and any information the system has about you), or in many other applications when you look something up on a computer.

有趣的事
Curiosity

让我们来看看,在你的班上是否有两名同学生日相同。让班上每位同学拿一张纸写下自己的生日。集齐全部纸片,在一张366天的表格中划掉出现过的每个日子,就像本章最后部分所示的那样。你会惊讶地发现很快你便能找到拥有同一天生日的两个人。比如,一个40人的班级,学生的生日在一年之中是平均分布的,那么往往总会有两个人的生日相同。事实上,这种概率高达90%。而一个拥有57名学生的班级中,两人拥有一样生日的概率则高达99%。而每367人之中,至少有两人的生日相同!

如果你班上的同学都生于同一月,那么试试将他们7人分一组,虽然他们的生日在一个月中平均分布,但仍有50%的概率其中两人的生日相同。

Let's find out whether two students in your class have the same birthday. Get everyone to write down their date of birth on a piece of paper. Collect all the pieces of paper, and cross off each date on a chart of the 366 days, like the one at the end of this section. You will usually find two people with the same birth date surprisingly quickly. For example, with a class of 40 students whose birthdays are spread evenly throughout the year, it is almost certain that two will share a birthday. In fact, the chances are nearly 90%. With a group of 57 people the chance that two people share a birthday is more than 99%. And if you have 367 people, then at least two of them *must* have the same birthday!

If your class was all born around the same month, try the same thing in groups of 7. With 7 students whose birth dates are spread evenly throughout a month, the chance is more than 50% that two will share a birthday.

一定有人觉得这点很神奇吧，为什么能这么快找到两个生日相同的人。而这只是一个数学游戏而已，现实中出现这种情况的概率会更高，因为一年之中生于某些日子比其他日子更为普遍。

同样的情况也存在于哈希表中。之前的战舰游戏中，即使我们使用多于 10 个的类，不同的战舰被归于同一类的几率仍然很大，这种现象我们称为“冲突”（collision）。30 个类和 7 艘战舰，有 50% 的概率会发生冲突；365 个类和 40 艘战舰，冲突的概率则接近 90%。这也就是为什么对计算机来说有效地处理哈希表冲突是至关重要的。

Most people find it surprising that they can find two people with the same birthday so quickly. But it's a mathematical fact. And in reality the probabilities are even higher because some times of year are more popular than others.

The same effect occurs in hash tables. It is surprisingly likely that different ships end up in the same pile, even if there are many more piles than the ten we used above. When this happens it is called a "collision." With 30 piles and 7 ships, there's a better than 50% chance of a collision. With 365 piles and 40 ships the chance is nearly 90%. That's why it's important for computers to deal with hash table collisions efficiently.

Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
1	1	1	1	1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3	3	3	3	3
4	4	4	4	4	4	4	4	4	4	4	4
5	5	5	5	5	5	5	5	5	5	5	5
6	6	6	6	6	6	6	6	6	6	6	6
7	7	7	7	7	7	7	7	7	7	7	7
8	8	8	8	8	8	8	8	8	8	8	8
9	9	9	9	9	9	9	9	9	9	9	9
10	10	10	10	10	10	10	10	10	10	10	10
11	11	11	11	11	11	11	11	11	11	11	11
12	12	12	12	12	12	12	12	12	12	12	12
13	13	13	13	13	13	13	13	13	13	13	13
14	14	14	14	14	14	14	14	14	14	14	14
15	15	15	15	15	15	15	15	15	15	15	15
16	16	16	16	16	16	16	16	16	16	16	16

续表

Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
17	17	17	17	17	17	17	17	17	17	17	17
18	18	18	18	18	18	18	18	18	18	18	18
19	19	19	19	19	19	19	19	19	19	19	19
20	20	20	20	20	20	20	20	20	20	20	20
21	21	21	21	21	21	21	21	21	21	21	21
22	22	22	22	22	22	22	22	22	22	22	22
23	23	23	23	23	23	23	23	23	23	23	23
24	24	24	24	24	24	24	24	24	24	24	24
25	25	25	25	25	25	25	25	25	25	25	25
26	26	26	26	26	26	26	26	26	26	26	26
27	27	27	27	27	27	27	27	27	27	27	27
28	28	28	28	28	28	28	28	28	28	28	28
29	29	29	29	29	29	29	29	29	29	29	29
30		30	30	30	30	30	30	30	30	30	30
31		31		31		31	31		31		31

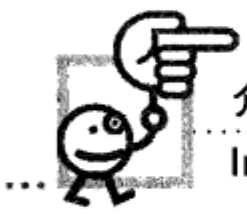
第10章

排序

0 Sorting 1

几乎所有计算机中的序列都是被排过序的。电子邮件列表按照日期排序，最新的邮件被放置在最顶端；播放器中的歌曲按照名字或歌手名排列在一起，以便你快速查找到最喜欢的那首；文件名则往往是按照字母顺序排列的。那么计算机是如何进行排序的呢？这一章将向大家介绍用于排序的三种简单算法：选择排序、插入排序和冒泡排序。了解它们工作的原理之后，大家会发现这三种算法的运算速度并没那么快——更好的方法将在第11章中再介绍。

Nearly every list that you see on a computer is sorted into order. E-mails are sorted by date so that the most recent is at the top. Songs in a music player are sorted by name or artist to make it easy to find the one you want. Filenames are in alphabetical order. How do computers sort things into order? This Topic introduces three simple **algorithms** to do this: selection sort, insertion sort, and bubble sort. As well as finding out how they work, we'll discover that these ones aren't particularly fast — we'll look at better methods in Topic 11.



介绍
Introduction

计算机会将序列自动按照特定种类进行排序，比如，名字往往按照字母顺序进行排列，预约的行程按照时间顺序排列，电子表格则按数字顺序被排列起来。

Computers are often used to put lists into some type of order. For example, names are sorted into alphabetical order, appointments are sorted by date, spreadsheets can be sorted into numerical order.

对序列进行排序有助于更快地找到我们想要的东西。例如，在第 9 章中我们学习到，可以采用快捷的二分搜索法来搜索，但前提是查找的对象是已经被排好序的状态。排序还可以便于找到极值，比如当你把全班考试成绩按照顺序排列出来之后，最高分和最低分就一目了然了。

Sorting lists helps us find things quickly. For example, in the Searching topic (Topic 9), we found that we could use binary search, which is fast—but only if the list is already sorted. Sorting lists also makes extreme values easy to find. For example, if you sort the grades for a class test into order, the lowest and highest marks become obvious.

a 好好想一想，哪些地方必须要将事物排好序。再想象一下，如果字母顺序、数字顺序和日期顺序变成乱序，会发生什么呢？

a Brainstorm all the places where putting things into order is important. Think of alphabetical order, numeric order, and dates. What would happen if these things were not in order?

计算机每次只能比对两个数据，如果你让计算机在序列“10 50 2 30 8”中找到最小数，计算机可做不到只检查数列一次便“看”到“2”是最小的数字，它只能一次对两个数字比较。

Computers can only compare two things at once. For example, if the computer has the task of finding the smallest number in the list “10 50 2 30 8”, it isn't able to examine the whole list at once and just “see” that “2” is the smallest. It has to compare just two numbers at a time.

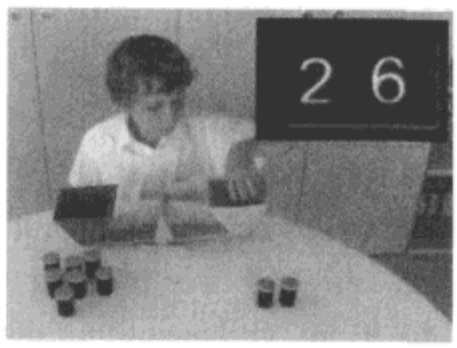
Topic 10 Sorting
第 10 章 排序

为了理解计算机的排序原理，我们要试着模拟出计算机的运行方式。但这样做非常困难！因为就算我们努力按照计算机的运行方式，但人的思维习惯很难避免一次性比较多个数据。在接下来的游戏中，我们将用一个方法使你不能看到全部的数据，这样你就只能一次比较两个数据了。

To understand sorting by computer, we have to work the same way. It's hard! When we try to pretend that we are working the same way as a computer, it's difficult for us to avoid comparing more than two numbers at once. In these activities we're going to solve the problem by preventing you from seeing all the values, so you can only compare two at a time.

视频：请观看排序游戏的录像。

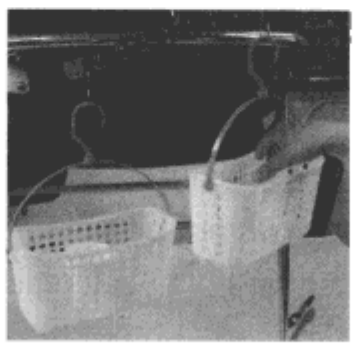
Video: watch the sorting video.



小游戏 10.1 选择排序
Activity 10.1 Selection sort

如果能借到一个天平，你可以在它的帮助下在游戏中一次只比较两个重量；万一找不到天平也没关系，你可以用卡片来取代。只不过用天平来玩这个游戏会好玩得多哦。学校里面一般都能找到天平，你也可以自己做一个——下图就是一副用两个篮子、一个金属条和一些钩子挂在椅子后面做成的天平。

If you have access to a balance scale, you can do this activity by using it to compare two weights at a time; if you don't have a scale then you can use cards instead but it's more fun with a scale. Schools often have a balance scale available, or you can even make one—the picture below shows a balance scale made from two baskets, a piece of metal, and some hooks, hanging from the back of a chair.



拿来作比较的重物外观最好看起来一致,以免仅凭外观就知道孰轻孰重了。在前面第一张照片中,我们使用的是装了硬币的胶卷塑料盒。第二张照片中,我们将弹珠放在一次性的纸杯中,然后用胶带封住杯口,防止从外面看到弹珠。请自己制作 8～9 个用来排序的重物。在这个游戏中,你只能使用天平来比较两个不同物体的重量,且每次只能比较两个被测物。

The weights that you compare need to look identical to prevent you working out which is heaviest just by looking. In the first photo earlier, the weights have been made by putting coins in film canisters. In the second photo, the weights are paper cups containing marbles, and then stuffed on top with tissue paper to hide the marbles. Make up a set of 8 or 9 different weights to sort. In this activity, to compare two different weights, you must always use the scale and you can only compare two weights at the same time.

如果手边没有天平,不妨找一个同伴来帮你做这个游戏。准备标有不同数字的 8 或 9 张卡片(你可以自己做几张卡片或者直接用扑克牌来代替),将卡片正面向下摊开在面前的桌子上,如果要比较其中的两张,让你的同伴查看它们上面的数字,然后告诉你比较的结果。

If you don't have access to a balance scale, you can get another person to do the comparisons for you. To do this, use 8 or 9 cards with different numbers on them (you can either make up some cards, or use some from a deck of playing cards). Keep the cards face down on the table in front of you; if you want to compare two of them, ask your partner to look at them and tell you which is the larger of the two.

排序是将一组无序输入的关键字(key)变成一组有序输出的过程。使用天平的时候,关键字就是每个重物的重量;如果你用卡片来做这个游戏,关键字就是卡片背面书写的数字。

Sorting takes as its **input** a list of unsorted keys and produces as its **output** a list of sorted keys. If you are using the balance scale, then the keys are the weight of each object. If you are using the cards, then the keys are the numbers on the back.

b 要找出重量最轻的物体(或最小数字的卡片),使用怎样的方法最快得出结果呢?记录下你作比较的次数。(最佳方案是让最轻的重物始终都在天平的同一端,然后拿其他重物过来和它进行比较,如果遇到更轻的重物,就用这个更轻的取代之前最轻重物的位置。)

b Find the lightest weight (or the smallest number) among the objects. What is the easiest way of doing this? Count the number of comparisons that you make as you go. (The best way to do this will be to keep the lightest weight so far on the scale, and compare the others with it, replacing the lightest so far if you come across one that is even lighter.)

c 你一共比较了多少次才找到最轻重物?

d 如果需要在 100 个被测物中找到最轻的一个,一共需要比较多少次呢?

e 需要比较多少次才能发现 3 个被测物中最轻的一个?

f 你能找到一个公式,用来计算在 *n* 个重物中找到最轻被测物所需要进行比较的次数 *c* 吗? 用两个重物和一个重物的例子来测试一下你的公式是否正确。

了解排序算法需要执行比较的次数是相当重要的,它关系着算法在计算机中执行的效率,而需要进行比较的次数与算法耗费的时间几乎成正比。所以在本章所有小游戏中,你都需要记录下做出比较的次数。

比起测量算法耗费的时间,记录需要比较的次数更方便,因为耗费的时间可能因排序的操作者或比较工具的不同而有差异,而比较次数在任何人使用任何工具的情况下都是相同的,对在高速计算机上运行的程序也不例外。

通过刚才游戏中找到最小重量的方法,我们可以将一整个序列排序。操作的思路很简单:像刚才那样选一个最轻的重物(数字)放在一边,然后在剩下的重物中挑选最轻的那个,依此类推。重复上述步骤直到所有的重物都被挑了出来,此时

c How many comparisons did you make to find the lightest weight?

d If you had to find the lightest of 100 weights, how many comparisons would you have to make?

e How many comparisons would it take to find the lightest of 3 weights?

f Can you find a formula for the number of comparisons *c* needed to find the smallest of *n* weights? Test your formula to see if it is correct for just 2 weights, and for 1 weight!

Knowing the number of comparisons a sorting algorithm must perform is important for knowing how fast that sort will run on a computer, since the amount of time taken will be roughly proportional to the number of comparisons. For these activities you will always need to count the number of comparisons that you make.

It's better to count the number of comparisons than to measure the time taken, because the time can depend on who is doing the sorting, or even which equipment they are using. Counting the comparisons will give the same results for any person using any equipment — even for a program running on a high-speed computer!

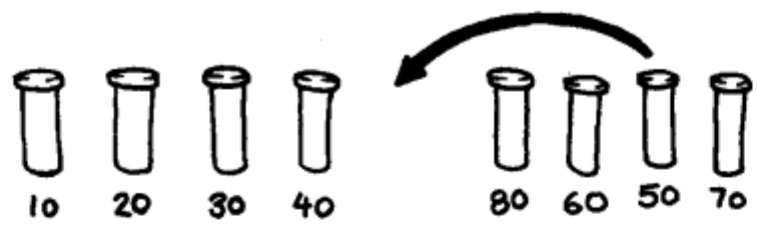
We can sort the whole list using the process that you just used to find the smallest weight. The idea is simple: select the smallest weight (number) like you just did and put it aside, then select the smallest weight from the remaining

它们已从轻到重被排列整齐。这个方法称为选择排序(selection sort),因为你总是持续地选择着最小的数值。

weights, and so on. Repeat this until all the weights have been removed from the list. They will have been put aside in order from lightest to heaviest. This method is called **selection sort** because you keep selecting the lightest value.

比如下图中,重物 10、20、30 和 40 已经被选出并放在一旁,下一步就是在剩下的 4 个重物中找出最轻的那个,即 50。当把 50 选中并移动到左边那组后,就只剩下 3 个重物未被排序了。(尽管为了方便讲解我们把每个重物的重量都标识出来,但玩游戏的时候你并不知道重物的重量,还是只能采用一次比较两个数据的方式来选择。)

For example, in the diagram below, the weights 10, 20, 30 and 40 have already been selected and put aside. The next step is to go through the remaining four weights to find the lightest one, which will be the 50. The 50 is then moved to the group on the left, leaving just 3 weights to be sorted. (Although we have shown what the weights are here, you won't know what they are, and can only use comparisons between weights.)



随机排列 8 个重物,然后使用选择排序法将它们按重量排序。计算一下一共需要比较多少次。

Mix up 8 of the weights so that they are in a random order. Then, use a selection sort to sort the 8 objects. Count how many comparisons you made.

如果你采用的方法正确,那么找到第一个最轻重物需要比较 7 次,找到次轻的重物需要比较 6 次,依此类推。所以得到排序结果总共需要比较的次数为 7+6+5+4+3+2+1。

If you were doing things correctly, finding the first smallest weight would have taken 7 comparisons, the next one 6, and so on. The total number of comparisons should be 7+6+5+4+3+2+1.

h 假设你需用同样的方法来将 21 个重物进行排序。找出最轻的重物需要进行 20 次比较，找到次轻的重物需要进行 19 次比较，依此类推。那么 20 + 19 + 18 + 17 + … + 1 的总和是多少呢？

h Suppose you had to sort 21 weights using this method. Finding the smallest would take 20 comparisons, the next smallest 19, and so on. What is the total of 20 + 19 + 18 + 17 + … + 1?

如果你用错了排序方法，那么在对一个大序列排序时将耗时巨大，就算用一台高速计算机结果也一样。实际上，我们刚才使用的选择排序法并不是一个很好的方法，如用该方法对 1000 个数据排序，我们将需要比较近 50 万次（准确说是 499500 次）。

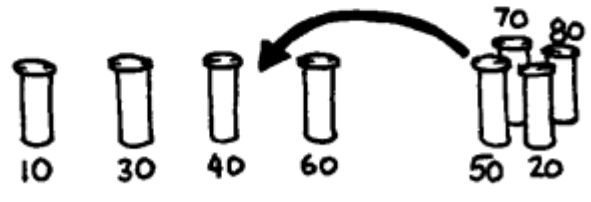
If you use the wrong method for sorting, it can take a long time to sort a large list into order, even on a fast computer. The selection sort method that we just used is not a very good method; for example, to sort 1,000 values, it would make nearly half a million comparisons (499,500 to be precise).

在本章接下来的两个小游戏中，我们将来探索一下用于解决排序问题的另外两种方法。但令人失望的是，除了在某些特定的条件下，这两种方法的效率比选择排序法也好不到哪里去。下一章我们会谈到其他几个高效得多的排序方法。

The remaining two activities in this Topic explore two more methods for sorting which illustrate different approaches to the same problem, although disappointingly neither of them is much better than selection sort except in special cases. In the next Topic we will look at some other approaches that are significantly better.

小游戏 10.2 插入排序
Activity 10.2 Insertion sort

插入排序(insertion sort)的工作原理为：在一个未排序的序列中依次移出每个对象将它们插入到有序序列中正确的位置(见下图)。每成功插入一次，即意味着未排序的对象减少了，排好序的对象增加了，直到整个序列被完全排列好。扑克牌玩家通常使用这种方法来理牌，这种排序法和选择排序不同在于，该算法的重点在于如何将数值放入左侧序列(即已排好序的序列)中的正确位置上，而不是去考虑该在右侧的序列要挑出哪一个数值。



Insertion sort works by removing each object from an unsorted group and inserting it into its correct position in a growing list (see picture below). With each insertion the group of unsorted objects shrinks and the sorted list grows, until eventually the whole list is sorted. Card players often use this method to sort a hand into order. It is different from selection sort because it puts the effort into deciding where to add the value to the list on the left, rather than which one to select from the list on the right.

让我们按照下列步骤来执行一次插入排序吧：第一步，随机挑出一个对象作为左侧有序序列中的第一个物品。第二步，从右侧未排序的序列中选择第二个对象，用天平来决定它与第一个对象的大小。第三步，选取第三个对象，用天平来决定第三个对象同前一次比较中较大者（在右侧的那一个）之间孰大孰小，如果第三个对象较大的话，将它放在左侧序列的最右端，如较小的话，将它与第一个对象比较，根据结果来决定将它放在第一个对象的左侧还是右侧。按上述方法持续从右侧的乱序序列中选择对象，并将其插入到左侧序列中正确的位置上，如果它比前一个被插入对象大的话，则将它直接插入到前一个被插入对象的右侧。

将 8 个重物无序混排，采用插入排序法对它们进行排序。记录下你一共比较了多少次。

现在你应该已经发现，插入排序要比选择排序快一点：平均下来你只用将每个重物与其他半数对象进行比较即可。当然这只是个平均值，实际情况中你的比较次数有时会少一点，有时则会多一点。尽管如此，插入排序仍然是一个很慢的方法，例如采用该方法对 1000 个对象的序列进行排序需要比较 25 万次。

【进阶】如果碰巧你每次从右侧序列选取的重物总是右侧序列中最轻的一个。那么对 8 个对象进行排序需要比较多少次呢？

Specifically, follow these steps to do a insertion sort: First, pick one object at random to form your (very short) list of sorted objects on the left. Next, pick a second object from the unsorted group; use the scale to determine if this second object is less than or greater than the first object. Then, pick a third object; use the scale to determine how much the third object weighs relative to the heavier of the first two objects (the one on the right). If it is heavier than that one then it can just go on the right of the sorted list, otherwise compare it with the first object to see if it should be inserted before or after the first object. Continue to pick an object from the unsorted group and move right to left along the sorted list, inserting it immediately to the right of the first one that is lighter than the one being inserted.

Mix up 8 of the weights so that they are in a random order. Then, use an insertion sort to sort the 8 objects. Count how many comparisons you made.

You should find that insertion sort is a little faster than selection sort: on average you only need to compare each weight with half of the other ones. Because it is an average, you may do better sometimes, and worse other times. However, insertion sort is still regarded as a slow method—for example, it would need about quarter of a million comparisons to sort 1,000 items.

【Expert】 Suppose by chance your choice of weight from the right hand side always happened to be the lightest one on the right. How many comparisons would insertion sort make to sort 8 items?

【进阶】如果碰巧你每次从右侧序列选取的重物总是右侧序列中最重的一个。那么对 8 个对象进行排序需要比较多少次呢?

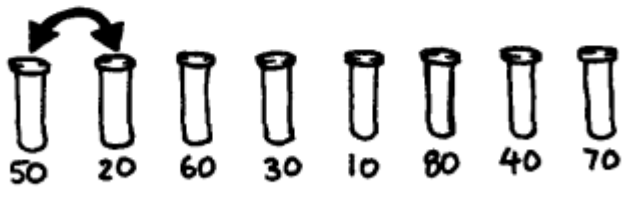
【Expert】 Suppose by chance your choice of weight from the right hand side always happened to be the heaviest one on the right. How many comparisons would insertion sort make to sort 8 items?

小游戏 10.3 冒泡排序
Activity 10.3 Bubble sort

冒泡排序(bubble sort)是一种需要将整个序列反复扫描,并交换所有相对位置错误的相邻数据的方法。当检查整个序列发现不用交换任何数据时便证明序列已被排好序了。这个方法的效率并不高(事实上,它是排序算法中最差的一个),但是它最容易被理解,所以这种方法常常被用于教学举例中。

Bubble sort involves going through the list again and again, swapping any objects side-by-side that are in the wrong order. You know that the list is sorted when no swaps occur during a pass through the list. This method is not very efficient (in fact, it is one of the worst choices of algorithms for sorting), but some people find it easier to understand than the others, so it is sometimes used as an introductory teaching example.

将 8 个重物随机排列,然后用冒泡排序法对它们排序:比较下图所示的第一组重物,如果它们的相对位置错误,则交换它们的位置。接着比较第二组重物(在本例中,经过第一组重物的比较后 50 已处于第 2 位的位置,此时需要将 50 和 60 比较,所以不需要交换它们的顺序)。如果一直检查到序列的最右边都不需要交换任何两相邻重物的时候,说明排序已经完成;否则需要将刚才的过程从头再来一次,直到整个序列中都不需要交换任何数据。记录下你一共比较过多少次。



Mix up 8 of the weights so that they are in a random order. Then, use a bubble sort to sort the 8 objects: compare the first pair of weights as shown in the diagram below, and swap them if they are out of order. Then compare the second pair (in the example, the 50 will now be second and it is compared with the 60, so no swap is needed). If you get right through the list without having to swap any weights then you can stop, otherwise go back to the start and do the whole routine over, until you get through without making any swaps. Count how many comparisons you made.

在冒泡排序中，将整个序列检查过第一遍后，你或许会发现，最右边的对象已经位于正确的位置了，下一次检查只用比较余下的 7 个对象，再下一次只用比较余下的 6 个对象，依此类推。这意味着冒泡排序最多只需要进行 7＋6＋5＋…＋1＝28 次比较。为了检验这个结论，试着在一张纸上列出一个倒序的数字序列，然后用冒泡排序法对其排序，观察每次需要比较多少次。

In bubble sort you may have noticed that after the first pass through, the right－hand object is in its correct place, and only the left－most 7 need checking on the next pass, then 6, and so on. This means that in the worst case bubble sort can be done in 7 + 6 + 5 + … + 1 = 28 comparisons. To test this, try writing down some numbers in reverse order and work out what bubble sort does at each step.

哪种排序算法花费的比较次数最少？

Which of the sorting algorithms required the fewest number of comparisons?

【进阶】如果我们需要对 21 个对象进行排序，在最糟糕的情况下，这些算法各需要进行多少次比较才能完成排序呢？

【Expert】 In the worst case, for all of these sorting algorithms, how many comparisons do you think are needed if you have 21 objects instead of 8?

进阶篇
Details for experts

尽管三种排序算法需要的平均比较次数各不相同，但是它们在最坏情况下需要的比较次数均为 1＋2＋3＋…＋(n－1)。这个总和可以用公式 $n(n-1)/2$计算得出，也就是 $n^2/2-n/2$(在下面有趣的事情一节中将为大家演示一下为什么会得到这个公式)。事实上，公式中包含的 n^2 非常关键：如果 n 变大两倍，意味着需要比较的次数增大到 4 倍；如果 n 变大 10 倍，那么意味着需要比较的次数增大到 100 倍。当你将 $n=1000000$ 代入到 $n^2/2-n/2$ 中，你会发现公式前半部分的值要比后半部分重要得多。正因为公式中前半部分的值如此重要，因此三种算法通常被称为“n^2”排序法，或称为二次排序法。在第 11 章中，我们将学习其他不会被 n^2 影响的排序法，对于大型数据集合来说它们的排序速度将显著提高。

Although the average case number of comparisons for each of the three algorithms mentioned above differs, the worst case number of comparisons needed to sort n items for each of these three algorithms is 1＋2＋3＋…＋(n－1). This sum can be calculated easily using the formula $n(n-1)/2$, which can be expanded to $n^2/2-n/2$. (The curiosity section below demonstrates why this formula works). The fact that the formula contains n^2 is very significant: if n gets twice as big, then the number of comparisons will get almost 4 times as big; and if n becomes 10 times larger, the number of comparisons will become 100 times larger. If you try putting in a number like $n = 1000000$ to the $n^2/2-n/2$ formula, you'll see that the first half of the formula is

much more important than the second half. This part of the formula is so significant that these three algorithms are often referred to as "n^2" sorting methods, or quadratic sorting. In the next Topic we will look at some methods that don't have the n^2 problem, and they really do run massively faster on large sets of data.

有趣的事
Curiosity

有人问一个聪明的孩子如果从 1+2+3+…一直到加到 1000 会得到多少。他用不到一分钟的时间便给出了正确的答案 500500。他是怎样算出来的呢?

There's a story about a clever child who was asked to add 1+2+3+4+…, right up to 1000. The correct answer is 500500, and he got it in less than a minute. How?

这个题目并不难。当你在对 21 个重物进行选择排序时,需要进行比较的次数为 20+19+18+17+…+4+3+2+1。我们可以重组一下数字来得到总和:

It's not so hard. If you sort 21 weights with selection sort, the number of comparisons is 20+19+18+17+…+4+3+2+1. To work out this sum, re-group the numbers:

20+1+19+2+18+3+17+4+16+5+15+6+14+7+13+8+12+9+11+10

这是全部的数字,而每一对数字的和均为 21。所以答案是

That's all the numbers. The sum of each pair is 21. So the answer is

21+21+21+21+21+21+21+21+21+21

结果为 210——因为一共有 10 对 21 嘛。

That's 210—because there are 10 pairs.

如果有 1000 个数字,则可组合为 500 对,每对之和均为 1001。这也就是为什么之前那个孩子算出的总和为 500500。

With 1000 numbers, there are 500 pairs, and each pair adds up to 1001. That's why the total is 500500.

如果一共有 999 个数字 (1+2+3+…+999),那么每对之和为 1000,一共 499 $\frac{1}{2}$ 对,因此总和为 499500。可为什么还有 $\frac{1}{2}$ 对呢? 不要忘记在数列的最中间落单的那"一对"。

With 999 numbers (1+2+3+…+999), each pair adds up to 1000. And there are 499 $\frac{1}{2}$ pairs, so the total is 499500. Why the $\frac{1}{2}$? Think about the "pair" in the very middle.

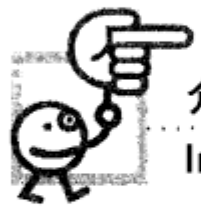
第11章

让排序来得更迅猛吧

01 Sorting Faster

第10章我们研究了三种不同的排序算法，尽管它们都能完成排序的任务，但实际上还有两种更高效的方法分别叫做快速排序法（quicksort）和归并排序法(merge sort)。实际使用中我们常常用到的是这两种算法。

In the last Topic we explored three different algorithms for sorting a list into order. They all get the job done. But there are two clever approaches that will work much faster, called *quicksort* and *merge sort*. These are the **algorithms** that are usually used in practice.



介绍
Introduction

我们已经学习了插入排序、选择排序和冒泡排序的工作原理，并指出各种排序法耗时的差别取决于待排序列表的长度。但是总地来说，三种算法所需要的时间相差无几。

We have seen how three sorting methods work: insertion sort, selection sort and bubble sort. As we explained, there can be differences in how long they take depending on the length of the lists that are being sorted. However, in general they each tend to take about the same amount of time, roughly speaking.

这一章我们将来测试一下两种更快的排序法，分别是快速排序和归并排序。相比之前提到的方法，这两种排序方法在运行速度上优势惊人，对一百万个对象进行排序，使用快速排序法比冒泡排序法要快近 20000 倍，而归并排序法的性能也与此相差无几。所以花一些努力来理解这两种高效的方法是相当值得的。

In this topic we examine two other methods that are almost always *much* faster. They're called quicksort and merge sort. The difference in speed can be huge. Sorting a million items is about 20000 times faster using quicksort instead of bubble sort. Merge sort also gives a similar improvement. So it's worth putting some effort into understanding these clever methods.

小游戏 11.1　快速排序
Activity 11.1　Quicksort

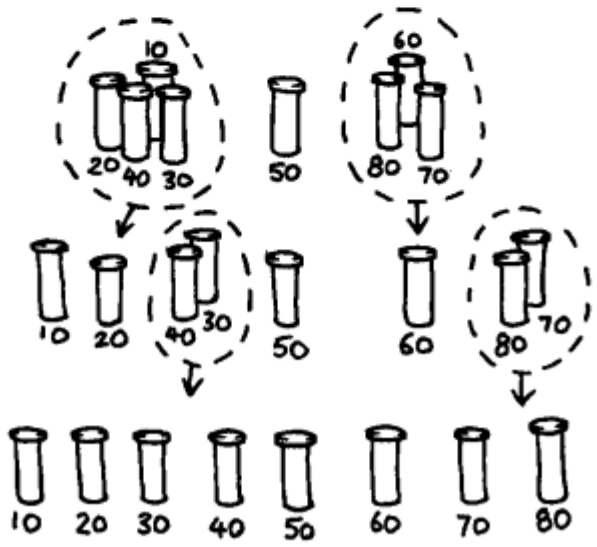
对于处理大量待排序对象的工作来说，快速排序法无疑是最佳选择。我们将用在第 10 章用到的重物和天平来描述一下它的工作原理。

Quicksort is one of the best methods known for sorting large lists of numbers. We will describe how to do it using the balance scale and weights as for Topic 10.

第一步，任意选取一个重物，将其放置在天平的一端。第二步，将剩下的所有重物依次和这个重物进行对比，将较轻的放在你的左边，较重的放在右边，然后（将所有重物分成两组后）将之前选取的重物放在两组之间。第一阶段的比较结果

First, choose one of the objects at random, and place it on one side of the balance scale. Next, compare each of the remaining objects with the chosen object. Put the objects that are lighter on your left, the heavier objects on the right, and

如下图第一行所示，两组对象中分别包括重量小于 50 的一组和重量大于 50 的一组（当然你有可能会得到一组的重物数量比另一组多甚至某一组中重物数量为零的情况）。



(after making the two groups of objects), put the chosen object in the middle of the two groups. The result of this first step is shown in the first line of the diagram below, where the two groups are all the objects lighter than 50, and heavier than 50. (By chance you may end up with many more objects on one side than on the other, or even none on one side.)

一开始被选出的重物我们称它为“基准”(pivot)。请注意，基准（比如上图第一行中的 50）在最后排好序的序列中正好位于正确的位置——它的左侧有 4 个比它小的重物，右侧有 3 个比它大的重物。我们不需要再对基准做任何操作，我们要做的只是对基准两侧的两组对象进行排序。

The chosen object is sometimes called the “pivot”. Notice that the pivot (e.g. the 50 in the first line of the diagram above) is in its correct place for the final sorted list-it has the four smaller objects on its left, and the 3 larger ones on its right. We don't need to do anything more with the pivot; all we need to do is sort the two groups.

快速排序法中最绝妙的一步在于，我们对基准两侧的对象再来排序时，采用的仍是之前的方式——选择其中的一组，然后重复上述“分裂”过程，即在该组中随机选出基准对象再将组中剩余对象“分裂”成两组，比基准对象轻的放左边，比基准对象重的放右边，基准对象放中间。对另一组也进行同样的处理。上图中的第二行展示了第二轮排序后的情况，其中左边一组的基准为 20，右边一组的基准为 60，接下来只剩下两组只含两个对象的序列需再次排序了，而其他的对象则已经处于排好序的状态。

The clever step with quicksort is that we'll sort the two groups using the method we just used – choose one of the two groups and repeat the splitting procedure, choosing one of the objects in the group at random and splitting the group into two piles with the lighter ones in a group on the left, the heavier ones on the right, and the chosen object in the middle. Do the same for the other group. This is shown in the second line above, where the 20 is the pivot for the left-hand group, and 60 is the pivot for the right – hand group. That leaves us with just two groups of two objects that need to be fixed – everything else is already in place!

对剩下的各组进行同样的操作，直到每组中只有一个对象。当所有组都被拆分成单项时，重物序列也完成了从最轻到最重的排序。

Keep repeating this procedure on the remaining groups until no group has more than one object in it. Once all the groups have been divided down to single objects, the objects will be in order from lightest to heaviest.

a 随机排列 8 个重物，然后采用快速排序法对其排序。记录下你一共比较过多少次。

a Mix up 8 of the weights so that they are in a random order. Then, use quicksort to sort the 8 objects. Count how many comparisons you made.

b 快速排序是否“快速”取决于你是否选对了“基准”对象。那么，怎样的“基准”是好的？好的“基准”在这里有什么作用呢？

b Whether quicksort is quick or not depends upon how lucky you are when you pick the “pivot” object that you divide the other objects around. What is a good pivot object? What does a good pivot object do in this case?

在快速排序中我们用到了递归(recursion)原理，即不断用相同的原则将序列划分成越来越小的各部分，并采用相同的步骤对各部分排序。实际上，这种特殊的递归法被称为分而治之(divide and conquer)。序列被重复地分割直到它足够小到能够直接导出结果，对于快速排序来说，序列将被分割到每组只剩下一个对象为止，而对一个对象进行排序是最容易不过的事情了。或许讲解起来非常繁琐，但在实际运用中快速排序法的运行速度明显地快于其他方法。

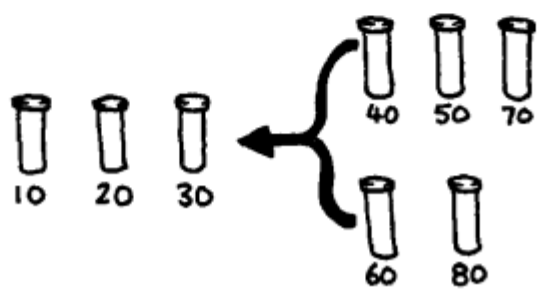
Quicksort uses a concept called **recursion** in which the computer keeps dividing the list into smaller parts and performing the same kind of sort on each of the parts. This particular kind of recursion is called **divide and conquer**. The list is divided repeatedly until it is small enough to conquer. For quicksort, the lists are divided until they contain only one item. It is very easy to sort one item into order! Although this seems very involved, in practice it is dramatically faster than other methods.

小游戏 11.2 归并排序
Activity 11.2 Merge sort

归并排序是另一个用到“分而治之”原理的排序算法。首先，将待排序序列随机分成两组且两组中对象数相同(如果对象总数为奇数的话，两组

Merge sort is another method that uses “divide and conquer” to sort a list of items. First, the list is divided into two randomly chosen lists of

数量则应接近相等)。然后分别对两组对象进行排序,再将两组对象归并起来。归并两个已经排好序的序列很容易,你只要不断地移出两组对象最前端较小的那个即可。在下图中,40克和 60 克重物位于两个子序列的前端,通过比较,选出较轻的 40 克的重物即可。之后,再比较 50 克和 60 克的重物,把 50 克的重物移出放入到排好序的序列中。



那么怎样对刚才的子序列进行排序呢? 其实很简单,还是用归并排序法! 即将子序列再分割成两半,对它们进行排序,再归并起来。最终,所有的子序列都变成了单独的对象,这样便说明此时每个子序列中的对象都已经被排好序了。

让我们用另一种方式来理解归并排序法。开始,每个重物归并到只有 1 个对象(它自己)的小组中。然后,选 2 个重物将它们"归并"成拥有 2 个对象的小组。下一步,再选 2 个排好序的小组(每个小组中均含有 2 个对象)将它们"归并"成含有 4 个对象的小组。最后一步,将 2 个含有 4 个对象的小组"归并"成一个排好序的含有 8 个对象的序列。

C 随机排列 8 个重物,然后用归并排序法对这 8 个重物进行排序。记录下你一共比较了多少次。

equal size (or nearly equal if there are an odd number of items). Each of the two half-size lists is then sorted, and the two lists are merged together. Merging two sorted lists is easy—you repeatedly remove the smaller of the two items at the front of the two lists. In the figure below, the 40 and 60-gram weights are at the front of the lists, so they are compared, and the next item to add is the 40-gram weight. After that, the 50 and 60-gram weights will be compared, and the 50 will go into the sorted list.

How do you sort the smaller lists? Simple—just use merge sort! That is, you split each of the smaller lists in half again, sort them, and merge them. Eventually, all the lists will be split down into individual items, and at that point you know that the individual item is a sorted list.

Here is another way of looking at merge sort. Begin by putting each weight into its own list of size one. Then, pick pairs of weights and "merge" them into a sorted list of size two. For your next step, again pick pairs of sorted lists (this time each sorted list will have two elements) and merge them into lists of four objects. Finally, for your last step, merge the two lists of four objects into a single sorted list of eight objects.

C Mix up 8 of the weights so that they are in a random order. Then, use merge sort to sort the 8 objects. Count how many comparisons you made.

d 你认为选择排序、插入排序、冒泡排序、快速排序和归并排序中哪种方法的速度最快？

e 【进阶】当你将两个各含有 4 个对象的序列合并时，最多需要比较多少次？

f 【进阶】当你将两个各含有 4 个对象的序列合并时，最少需要比较多少次？

d Which algorithm was the fastest of all of those you've seen: selection sort, insertion sort, bubble sort, quicksort, and merge sort?

e [Expert] when you merge two lists of 4 items each, what is the *largest* number of comparisons that you will ever need to make?

f [Expert] when you merge two lists of 4 items each, what is the *smallest* number of comparisons that you will ever need to make?

进阶篇
Details for experts

正因为排序操作在计算机系统中被频频使用，所以系统通常都会内置一个快速的排序算法，从而避免用户再来反复编写程序。尽管如此，算法的选择仍然至关重要，即使你使用系统内置的排序法，还是务必知道不同排序算法的性能差别。例如在实际应用中需特别注意待排数据的存储方式，即数据是存储在计算机内存中还是在硬盘中。因为归并排序法一般会合并大的数据序列，因此它比较适合处理存储在硬盘上的数据，而快速排序法总是要将数据移动到不同的位置，因此它更适合处理储存在计算机内存中的数据。

Sorting is so common that computer systems usually have a fast sorting method built into them, so people don't need to make their own. However, choosing the right sorting algorithm is important, and even if you use a built-in system, it's important to know the difference in performance between the various methods. A key issue in practice is whether the data being sorted is in the computer's memory (RAM) or on disk. Because merge sort is merging long sequential lists, it works well if the data is stored on a disk, whereas quicksort can be moving data all over the place, and is better suited to sorting data stored in the computer's main memory (RAM).

快速排序法对 n 个对象进行排序平均需比较 $1.39n\log_2 n$ 次，其中 $\log_2 n$ 是以 2 为底的 n 的对数，这一结果明显优于需要耗费 n^2 次比较的诸如插入排序这样的算法，因为就算 n 很大的时候，n 的对数值也是一个很小的值。

The number of comparisons that quicksort needs to sort n items is, on average, about $1.39n\log_2 n$, where $\log_2 n$ is the logarithm to base 2 of n. This is a lot better than the n^2 comparisons required by algorithms like insertion sort because the logarithm is a very small value, even if n is very large.

归并排序法比快速排序法需要的比较次数还要少，归并排序法平均只需要 $n\log_2 n$ 次比较，也就是比快速排序法平均需要的比较次数要少 39%。然而在实际应用中它们的差别并没有那么显著，因为归并排序在合并过程中需要占用大量的临时存储空间，这也就是为什么快速排序法一般来说都是最佳选择，除非数据储存在硬盘中（前提是，计算机硬盘中还有大量的剩余空间）。

Merge sort requires even fewer comparisons than quicksort: on average merge sort will make $n\log_2 n$ comparisons, so quicksort requires about 39% more comparisons than merge sort. However, the difference isn't significant, and mergesort requires temporary storage space during the merging process that is quite demanding, which is why quicksort is generally favored unless the data is stored in a disk drive (in which case there is probably a lot of spare space available anyway).

有趣的事
Curiosity

在美国 2008 年大选前，Google 采访了总统候选人贝拉克·奥巴马。当奥巴马被问到“对一百万个 32 位整数进行排序的最佳方法”时，他回答道“冒泡排序法绝对不是正确答案”。这个回答相当巧妙，赢得听众的好评。如果采用冒泡排序法来做这个题目，将需要比较 499 999 500 000 次，而快速排序法需要比较 20 000 000 次，比冒泡排序法整整快了 25 000 倍。这很好的印证了之前的结论：一旦选对了算法，将极大地提高运算效率。

In an interview at Google Inc. prior to the 2008 US presidential elections, candidate Barack Obama was asked to give "the most efficient way to sort a million 32-bit integers." He replied that "bubble sort would be the wrong way to go." This is a very good answer, and it went down well with the audience. Bubble sort needs about 499 999 500 000 comparisons of the keys to do the job. Methods like quicksort need about 20 000 000 comparisons, so they would do the same job about 25 000 times faster. That's a pretty good improvement in speed just by choosing the right algorithm.

对于十亿个对象来说，快速排序法将会快16 700 000倍——就好比一秒钟和193天的区别那样。2000年时，网络上已拥有超过千万计的网页，有人估计说如今这个总数大概翻了1000倍，因此搜索引擎公司真的需要设计出可以处理如此海量数据的算法，而这个时候，选对算法对他们的程序员来说就至关重要了。所以在这些公司招聘时，应聘人常常会被问到奥巴马先生被问到的那个问题(尽管奥巴马先生当时可不是去Google应聘职位)。

With a billion items, quicksort would be about 16 700 000 times faster—that's the difference between taking 1 second and 193 days. The web had a billion pages way back in the year 2000—some estimate that there are more like a thousand times that many today – so web search companies really do have to design algorithms that can process these large amounts of data. It's therefore very important that their programmers choose the right algorithm. For this reason, they often ask job applicants questions like the one that Barak Obama was asked (although he wasn't applying for a job at Google!)

第12章

并行排序

0 1 Sorting in Parallel

尽管计算机的运算速度很快，可它们解决问题的速度还是存在上限。一种进一步加快运行速度的方法是让不同的计算机同时处理问题的不同部分。在上两章中我们学习了排序算法，这一章我们将使用并行网络来实现一次性进行多次比较的高效排序。

Even though computers are fast, there's a limit to how quickly they can solve problems. One way to speed things up is to use several computers to tackle different parts of a problem at the same time. In the last two topics we looked at the problem of sorting; in this one we use parallel networks to sort even faster by doing several comparisons at once.

介绍
Introduction

人们一直渴望让计算机处理信息的速度变得更快。在过去 50 年间,计算机技术飞速发展,然而现在已经接近了一个极限,让计算机变得更快的梦想变得越来越困难了。

We want computers to process information as quickly as possible. Over the last 50 years, vast improvements in technology have increased the speed of computers enormously. However, now a limit is being reached, and it is getting harder and harder to make computers go faster.

尽管如此,计算机的制造成本还是疯狂跳水。因此人们萌发出一个很棒的点子,即采用多个处理器,让每个处理器同时处理问题的不同部分。这一方法被称为并行计算(parallel computation)。众人拾柴火焰高,它让计算机解决问题的速度变得更快。

They're still getting cheaper, though. So an attractive approach is to use more than one processor, each working on different parts of the problem at the same time. This is called **parallel computation**. Many hands make light work! —or in this case faster work.

有许多类型的计算机利用了这个原理,例如,多核处理器(一台计算机中安置多个处理器)现在已经被普遍用于个人计算机;网格计算(多台计算机通过网络连接在一起)可用于解决大型科研课题;另外还有大规模并行超级计算机(数千台计算机紧密连接在一起)。

There are many kinds of computer that use this approach: **multi-core processors** (several processors in one computer), which are now normal in personal computers; **grid computing** (many computers linked together over a network), which is used to solve large scientific problems; and **massively parallel supercomputers** (thousands of small computers coupled together very tightly).

有些问题能容易地被划分成不同部分并分配给不同处理器,比如,处理一张图像中的所有像素时,通常会将图像的不同部分给每个处理器。然而有些问题却很难拆分并实现并行,因为这些问题中的各部分必须严格按照某一先后逻辑来处理。

Some problems are easy to divide between several processors. For example, to process all the pixels in an image, it is often possible to assign parts of the image to each processor. However, some problems are hard to split up because one part must be finished before another part can be started.

Topic 12 Sorting in Parallel
第 12 章 并行排序

在前两章中我们学习了排序的方法，比起其他排序法，快速排序和归并排序已经从根本上提升了排序的效率，但是却无法变得更快了，除非采用完全不同的算法。这时并行计算就是进一步加快运算速度的方案之一，事实上它往往也是唯一选择，因为像归并排序这样的算法，在只能一回比较一次的计算机中已经公认是最快的算法了。在下面的小游戏中，我们将探索一下怎样通过同时进行多次比较来让排序变得更快。

In the last two topics we have looked at sorting. The quicksort and merge sort methods introduced in the previous topic improve very substantially on the earlier methods in terms of the time taken. But you can't get any faster unless a radically different approach is taken. And parallel computation is one possibility—in fact, in most cases it's the only practical possibility for speed-up because algorithms like merge sort are known to be about the best you can do if the computer is only making one comparison at a time. In the following activities we will explore how to sort numbers faster by ensuring that more than one comparison is made at a time.

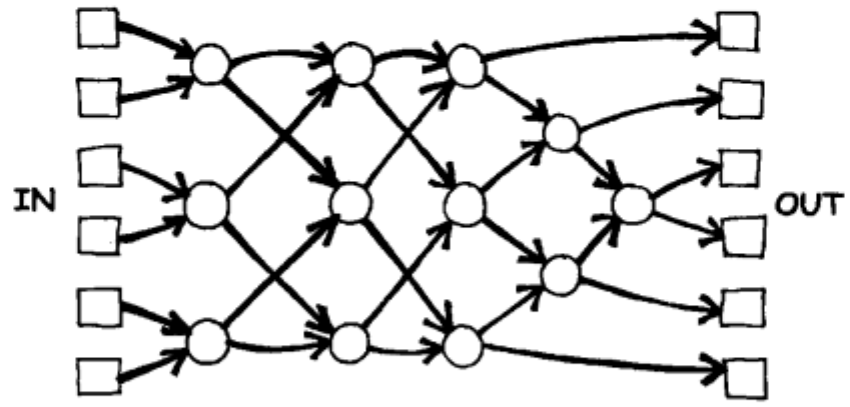
视频：请观看网络排序游戏的录像。

Video: watch the sorting network video.

小游戏 12.1 排序网络
Activity 12.1 Sorting networks

在这一章，我们将来看看怎样利用排序网络来快速地对一组数字序列进行排序。下图所示的排序网络可以一次对 6 个数字进行排序。

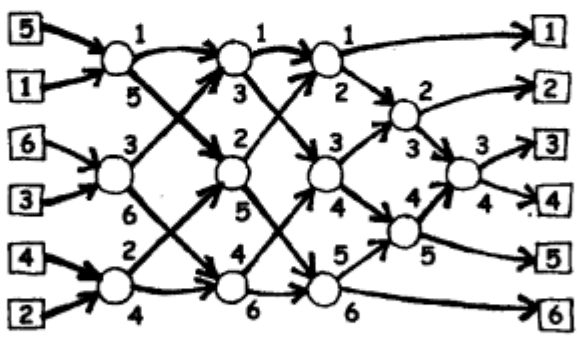
In this topic we will see how a **sorting network** can be used to sort a list of numbers quickly. The sorting network that we will use can sort 6 numbers, and is shown in the following diagram.



你可以在地面上用粉笔或胶带来“画”出这个排序网络(如上图),也可以在棋盘上玩。如果在地面上玩,你需要找 6 个伙伴,每个伙伴需要带着一个数字按着箭头的指示穿过排序网络;如果用棋盘来玩,你只要准备 6 个带有数字的棋子,然后将它们在网格中移动就可以了。

You can do this activity either by marking this diagram on the ground with chalk or tape (as in the photograph above), or as a board game. If it's on the ground, you will get six people carrying numbers to walk through it following the arrows; if it's a board game, you just use 6 counters with numbers on them and move them through the network.

待排序的 6 个数字开始被放在左侧的 6 个框中。每一步,将它们沿着箭头移动到圆圈中(称为一个节点)。在节点处比较此处的两个数字,较小的数字沿着向上的箭头移出节点,较大的数字沿着向下箭头移动(从站在节点的人的角度来看,较小的数字向左移动,较大数字向右移动,就像你从小到大书写数字的顺序一样)。比如,下图中显示了所有数字在该排序网络中移动的方式。



The 6 numbers that need to be sorted start at the six squares on the left. At each time step, they follow an arrow to a circle (called a node), and the two numbers are compared at the node. The lower number is sent along the upper arrow out of the node, and the higher number is sent along the lower arrow (from the point of view of someone standing at the node, the smaller number takes the path to the left, and the larger one takes the path to the right, which is just what you would do if you were writing the numbers in increasing order). For example, the following diagram shows where all the numbers go through the network for that particular input.

以刚才第一轮的操作为例，我们在左上角的节点处(圆圈)比较了 5 和 1，在它下面的节点处比较了 6 和 3，再下面一个节点处比较了 4 和 2。然后输出结果被传送到下级的 3 个节点，在下一轮的操作中将同时比较这 3 处数据。

In the above example, in time step 1, the top left-hand node (circle) compares 5 and 1, while the one below it compares 6 and 3, and the one below it compares 4 and 2. These are passed on to the next three nodes, which do three comparisons at the same time in the next time step.

a 上例中，移动到最右侧的数字已经被排好顺序了。试试将数字的顺序重新打乱后再用这个排序网络来排序看看，是否总能输出排好序的结果呢？

a In the example above, the numbers come out the right-hand end in sorted order. Try the sorting network with the numbers being started in different orders. Do they always come out in sorted order?

b 数字从左至右穿过网络一共需要移动多少步？(提示：注意一些比较是同时发生的，而另一些比较是需要等前一步比较完成后才能开始的。)

b How many time steps are needed for the numbers to go through the network? (Hint: remember that some of the comparisons are happening in the same step, but others have to wait for one step to finish before they can start).

借助排序网络比借助天平排序要快得多，因为网络中能同时进行 3 组比较。一个并行网络比一次只能执行一组比较的系统要快上 2 倍，但排序网络则需要使用到 3 倍数量的设备资源。

A sorting network can be faster than the sorting algorithms we used with a balance scale because the network has up to 3 comparisons being performed simultaneously. A parallel network can sorts the list more than twice as quickly as a system that can only perform one comparison at a time. Unfortunately, it needs three times as many comparison devices.

现在，让我们来看一下排序网络的几种其他形式。

Now, let us consider some variations of this sorting network.

c 如果令较小的数字向下移动而不是向上会发生什么呢？反过来呢？

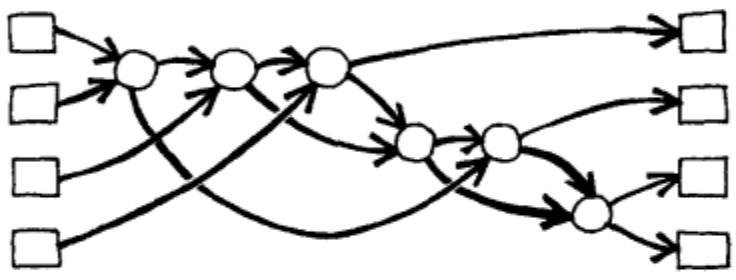
c What happens if the smaller one goes down instead of up, and vice versa?

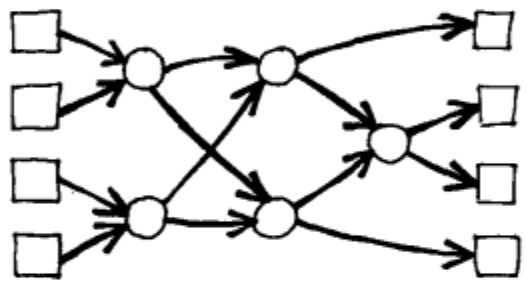
d 【进阶】你能自己设计一个用来将3个数字排序的网络吗？（请记住，每个节点只能比较两个数字。）

d 【Expert】 Can you design your own sorting network that will sort three numbers? (Remember, each compute node can compare only two numbers.)

e 下面有两个不同的网络用于对4个输入数据进行排序。哪一个更快？为什么？

e Below are two different networks that sort four inputs. Which is the faster? Why?

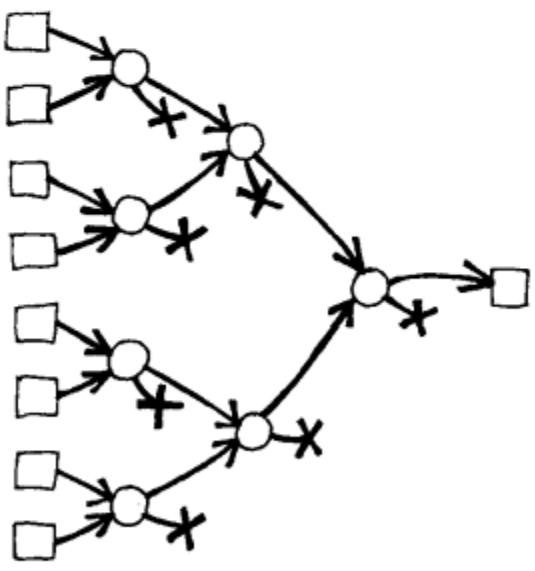




网络结构也被用于计算一连串输入数据的处理结果。

Networks can also be used to perform other computations across a set of inputs.

f 你知道下图所示网络在做什么运算吗？换句话说，最终会输出哪个数字？（×代表此路不通，即数字不会执行下一步操作。）



f Can you figure out what this network computes? In other words, which of the input numbers will this network always give as output? (The ×'s all represent dead ends in which the number isn't sent any further.)

有些日常生活中的事件也能通过并行方法被加速，而有一些则不能。

Some processes from everyday life can be accelerated using parallelism, and some can't.

g 雇更多的人用更多的铁锹能将一个 9 米长的排水渠挖得更快吗?

h 雇更多的人用更多的铁锹能将一个 9 米深的土壕挖得更快吗?

i 做一餐饭的过程中哪些步骤可以并行操作,哪些不能?

j 洗衣服的过程中哪些步骤能并行操作,哪些不能呢?

总的来说,并行操作能让有些事情做得更快,而另一些则未必,因为你需要先完成一部分然后才能开始下一部分。计算机科学家们还在积极寻找分解问题的最佳方案,从而使计算机能够以并行的方式高效处理它们。

g Can digging a 9-metre long ditch be done faster using more people and shovels?

h Can digging a 9-metre deep hole be done faster using more people and shovels?

i What aspects of cooking a meal can and can't be accelerated using parallel processing?

j What aspects of washing clothes can and can't be parallelized?

In summary, some tasks in life and on computers can be parallelized to go faster, and some can't because you need to finish one part before you can start another part. Computer Scientists are still actively trying to find the best ways to break problems up so that they can be solved more efficiently by computers working in parallel.

进阶篇
Details for experts

一般的排序仍多采用第 11 章中的非并行方式来实现,但是思考并着手设计并行方式的排序算法对今后创造出更厉害的新系统是很好的锻炼。实际应用中的排序网络往往太过庞大,不适合手工书写程序,因此计算机科学家们也在着力研究出能自动设计并生成排序网络的算法。

Sorting is still usually done using the non-parallel methods discussed in Topic 11, but thinking about parallel sorting methods is an excellent exercise in designing parallel systems, which is a very important skill for people designing new systems. In practice, sorting networks are too large to design by hand, so computer scientists work on algorithms that will design the networks for them!

让一个计算机程序自动生成出另一个计算机程序是一个非常了不起的想法。事实上，你在计算机中运行的大多数程序都是这样产生的：首先有人用特定的语言编写出人们易于理解的程序（比如 Java、C 或 BASIC），然后计算机将它"编译"（compile）成机器能执行的程序命令。此外，现代计算机芯片的复杂程度已经超出了人为设计的能力，一般来说都是由计算机程序来管理芯片设计的细节。使用计算机来设计新的计算机系统已经非常普通了，但问题是，它们能离开人类的指导自己运作吗?!

The idea of using a computer program to produce another computer program is an important one – in fact, that's exactly what has happened for most programs that you run on a computer: someone writes the program in a language that is easy for a human to understand (such as Java, C, or BASIC), and then it is "compiled" to something that the computer can execute. Furthermore, modern computer chips are too complex for a human to design on their own, so usually a computer program takes care of the details of the design. Using computers to design new computers is therefore very normal. The question is, will they start doing it without human intervention?!

Curiosity
有趣的事

如果你想来测试一下有 6 个输入的排序网络是否能够正常工作，那么输入端的初始状态将有 720 种不同选择，也就是你需要测试 720 次！对象的不同排放顺序被称为排列（permutations）。

Suppose you had to test the 6-input sorting network to make sure that it worked correctly for every possible input. There are 720 different orders that the 6 people could choose from at the start, so you'd have to run 720 tests! The different orders that objects can be lined up in are called **permutations**.

为什么仅 6 个对象就有 720 种排列可能性呢？那么假想这里有 6 个盒子，如果你有 6 个不同的数字准备放在盒子里，对于第一个盒子来说便有 6 种不同的选择，对于第二个盒子来说就只剩下 5 种可能，对于第三个盒子来说就只剩下 4 个可能，依此类推直到你将 5 个数字放入前五个盒子，

Why are there 720 permutations for just 6 objects? Well, there are 6 input boxes. If you had 6 different numbers to place in the boxes there are 6 choices for the first box, which leaves 5 choices for the second, which leaves 4 choices for the third, and so on—until once you

第六个盒子就只有一种选择了。这样总的可能情况为 6×5×4×3×2×1=720 种，也被称为 6 的阶乘。你可以试着用笔写下这 720 种可能排列，不过或许先写下 4 种输入的 4×3×2×1=24 种可能排列要容易得多。你会发现数字越大，情况也会很快变糟。对于 12 位输入网络，将有 12×11×10×…×1 种排列，也就是说你需要做 479001600 次测试；当你将 18 个数字排序，可能有高达 6 402 373 705 728 000种的输入方式！谢天谢地，计算机科学家们已经能用数学方法来证明这种排序网络能正常工作，不至于我们自己来测试几百万次了。

have placed numbers in 5 boxes there is only one choice for the sixth box. This makes the number of possible inputs $6 \times 5 \times 4 \times 3 \times 2 \times 1 = 720$, which is referred to as 6 factorial. You could try it out by writing down the 720 possibilities, but you'd probably find it easier to experiment with just the $4 \times 3 \times 2 \times 1 = 24$ possibilities for a 4-input sorting network. It gets worse very quickly too—if you had a 12-input network, there would be $12 \times 11 \times 10 \times \cdots \times 1$ permutations, so you'd have to do 479001600 tests. And just to sort 18 numbers, there are 6402373705728000 different ways they might start out. Thankfully computer scientists can use mathematical techniques to prove that these sorting networks will work, without having to run billions of tests.

当然，不是所有的问题都能并行处理的。早前我们想到一个点子，如果一个人需要用 9 天时间来挖一个 9 米长的水渠，那么 9 个人同时挖就能在一天内完成了。但是如果是挖 9 米深的土洞，事情就没那么简单了。因为一个女人可以九月怀胎产下一个小孩，但并不意味着 9 个女人一起努力，就能让小孩在一个月之内生出来……

Not all problems are parallelizable. Earlier we brought up the idea that if it takes one person 9 days to dig a ditch 9 meters long, then maybe 9 people could do the job in one day, but if the problem was to dig a hole 9 meters deep, things are not so easy. And just because a woman can bear a child in 9 months doesn't mean that 9 women, working together, could do it in 1 month . . .

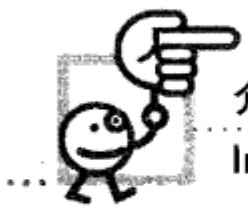
第13章

网络

0 Networks 1

网络在我们的生活中随处可见，电话网络、公共补给网络、计算机网络以及交通网络都将人们的日常生活高效连接起来。不论是网络中的电缆、无线电通讯线路还是道路都能以多种方式来连接事物。这一章中我们将来看看，怎样寻找高效连接网络中各节点的方法。

Our society is linked by many networks including telephone networks, utility supply networks, computer networks, and road networks. For each one there is usually some choice about where the cables, radio links, or roads can be placed to join things up. In this topic we look at how to find efficient ways of linking objects in a network.



介绍 Introduction

在本章中，计算机将为日常生活的实际问题找到好的解决方案，比如用最少条道路将房子连接起来。当然，大多数的道路有一些物理上的限制，比如你不能让路跨过悬崖或横越海洋，而把计算机连接起来的物理上的限制要相对较少。尽管如此，我们还是渴望得到一个高效的网络，不是过多地考虑如何节省网线，而是专注于如何尽可能地让网络传递更加通畅高效。

This topic will show how computers can be used to find good solutions for real-life problems such as connecting houses with the smallest number of roads. Of course, most roads have physical constraints: you can't put a road over a cliff or through the sea. When linking computers together there are fewer physical constraints. But we still want an efficient network, not so much to save wire as to make network traffic flow as efficiently as possible.

在设计计算机网络时总是需要找到高效的连通方案，不妨让计算机代替人们来设计网络，并由计算机帮我们决定如何在这个网络中路由！当然，设计网络时还会有各种各样其他的问题，但这一章的小游戏将让你领略到作为一名网络设计师不得不考虑的问题。

Finding efficient solutions comes up a lot in the design of computer networks – and we can use computers to design our networks for us, and decide how to route traffic through them! Of course, there are many other problems in **network design**, but this activity will give you a taste of one aspect that designers must consider.

小游戏 13.1　泥泞城市
Activity 13.1　The muddy city

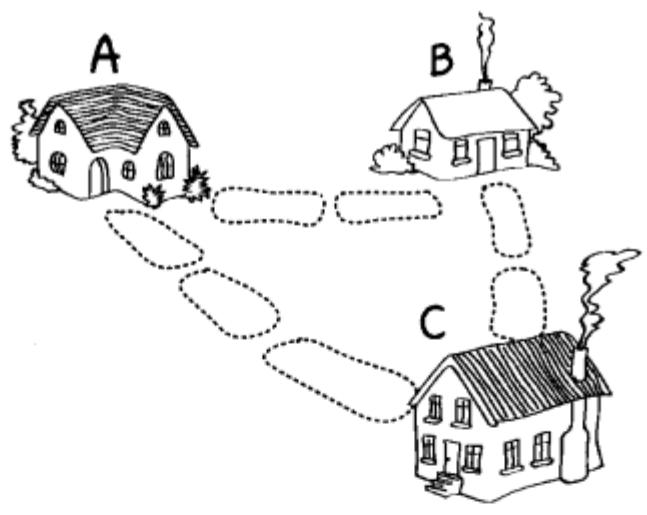
假设有一座城市还未铺上道路，下雨之后要在这座城市中行走是一件非常困难的事情，地面被雨水浸湿，满是泥泞，车辆纷纷陷入泥沼之中，人们的鞋子上沾满了污泥。于是市长痛下决心，一定要为一些道路砌上石砖，但是钱必须用在刀刃上，他不想浪费经费，因为市民也希望能在城市里修一个游泳池。因此他下达了以下两个要求。

Imagine there is a city with no paved roads. Getting around the city is difficult after rainstorms because the ground becomes very muddy – cars get stuck in the mud and people get their boots dirty. The mayor of the city decrees that some of the streets must be paved, but doesn't want to spend more money than necessary because the city also wanted to build a swimming pool. The mayor therefore specifies two conditions:

1. 必须铺设足够的道路，让每个人都能从他的家里沿着铺好的道路到达别人的房子。

2. 所花费的经费越少越好。在两栋房子之间的道路上铺上的石砖数代表了铺路所需要的费用。

a 设想这座小城仅有 3 栋房子和 3 条路。为了连接所有的房子，且要使用最少的石砖，哪些道路是必须铺上石砖的？（房子之间的每个方块代表需在此铺上一块石砖。）



b 如果从房子 A 走到房子 C，你会选择哪一条铺好的路来走？

c 让我们来测试一下大一点的城市，有 5 栋房子的。如果只能用最少的石砖，那么哪些路是必须被铺设的？（你可以试着在每个石砖上放置标记用来测试不同的方案。标记可以是游戏币、硬币、纸片甚至便条。）

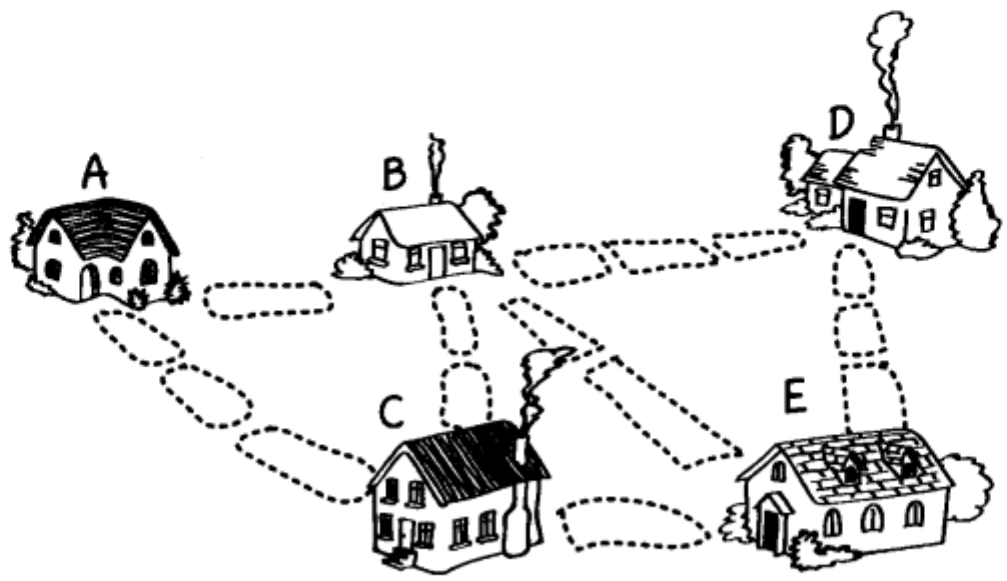
1. Enough streets must be paved so that it is possible for everyone to travel from their house to anyone else's house only along paved roads, and

2. The paving should cost as little as possible. The number of paving stones between each house represents the cost of paving that route.

a Consider this very small town with only three houses and three roads. Which roads should be paved to connect all houses, using the fewest stones? (each square shows where a stone would be needed between the houses.)

b Given these paved roads, what path would you take to walk from house A to house C?

c Let us examine a slightly larger town with five houses. Which roads should you pave in this town to minimize the number of stones needed? (You can experiment with different solutions by placing a marker on each paving stone that you are considering including. Your markers could be game counters, pennies, scraps of paper, or even cut up sticky notes.)



d 从房子 A 到房子 D，你会走哪一条铺好的路？

e 还有其他更好的铺路法吗？为什么？

f 从房子 B 到房子 E，哪条路会比较好？

你或许已经从问题 f 中注意到，使用最少的石砖铺路时的路线不一定是两栋房子间最短的线路。这两个最小值问题都很重要，但是它们的解决方式却不尽相同。

d How would you walk from house A to house D with your paved roads?

e Are there other choices for paving roads that are just as good? Why?

f If you were walking from house B to house E, which version of paved roads would be better for you?

You may have noticed from question f that the solution to the minimum number of paving stones for the whole town isn't necessarily the same as finding the shortest route between any two houses. They are both important problems, but we need different methods for solving each one.

术语一点通

The task that you have been doing, designing a network with a minimal total length, is called **the minimal spanning tree** problem. You have been developing your own **algorithm** to find the minimal spanning tree. The solution is "minimal" because it has the smallest total number of paving stones (i. e. total length of connections). "Spanning" means that every house is connected to every other house somehow. And it is called a "tree" because the shape of a correct solution is a bit like a tree: if you start at a particular house there will be one or more paths "branching" out from it, and each of those branches may have further branches later, but two branches will never come back and join up – if they did join up again, you would have two ways to get to a house, and one of them would be wasting paving stones.

刚才的小游戏要求你设计出一个总长最短的网络,它被称为最小生成树问题(the minimal spanning tree problem)。其实刚才你已经自己设计出寻找最小生成树的算法了,而该问题之所以被称为"最小"是因为它要求拥有最少的石砖总数(也就是连接网络的总长度),"生成"代表了每一栋房子都和另外一栋连接起来,最终得到的正确方案的形状看起来则像一棵"树"——如果你从任一栋房子出发,都会有一条或多条道路从这里"分叉"出去,而这每一条分支稍后又会有其他的分支,但是两条分支永远无法回指或相交,如果你让它们相交了,说明将有两条道路能通往同一栋房子,则其中的一条必是浪费石砖的选择。

最小生成树也能有效地解决其他问题。

Minimal spanning trees are useful for other problems as well.

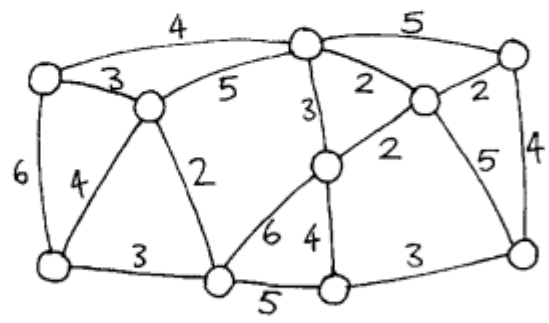
你能想出最小生成树的其他应用吗?(提示:除了道路还有什么需要通往每栋房子的?)

Can you think of any other problems where a minimal spanning tree could be used? (Hint: Besides roads, what else needs to go to every house?)

也可用下图中的圆圈、线条和数字来表示地图中的房子和道路,计算机科学家们称这种结构为图(graph)。在这些图中,圆圈代表的房子称为节点(node),节点之间的线条代表泥泞的道路,每条道路的长度用线条旁的数字标明。下面便是一个

Another way of representing houses and roads is with a diagram like the one below, with circles, lines and numbers to represent the maps we've been using. Computer scientists call this kind of diagram a **graph**. In these graphs, the houses

大一点城市的道路图。



are represented with circles (called **nodes**), the muddy roads with lines between the nodes, and the length of each road with a number beside the line. The following is an example graph of a larger town.

这座城中一共有多少栋房子?

How many houses are in this town?

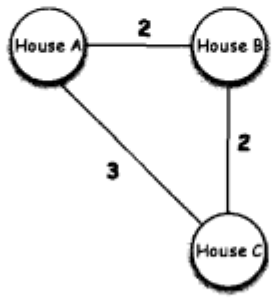
术语一点通

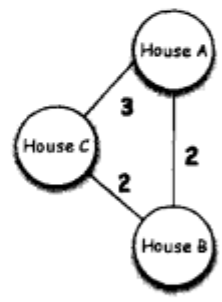
The diagram above is called a graph. Confusingly, the word "graph" is also used in mathematics to refer to statistical charts and plots of functions, but in computer science it almost always refers to the kind of diagram above, with nodes linked together to show some sort of relationship (such as distance) between them. If the graph represents a map, the nodes might be street intersections; if it represents airline flights, the nodes would be airports.

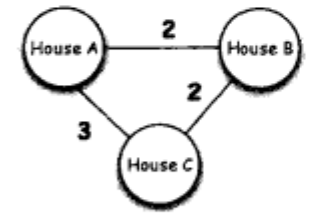
上面的结构被称为图(graph),令人迷惑的是"graph"一词也在数学领域中用来表示统计学中的表格和函数坐标,但在计算机科学中它一般指上面所示的图结构,即用互连的节点构造出它们之间的某种特定关系(比如距离)。如果用图结构表示地图,那么节点可能就是街道的路口;如果表示飞行航线,那么节点可能就代表了机场。

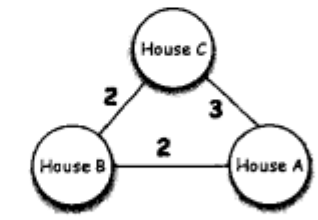
这里有4种版本的图可以用来表示只有3栋房子的小城市,尽管它们看起来不太一样,可是它们都表示同一座城市,也拥有相同的道路解决方案。

Here are four versions of a graph representing the first town with 3 houses. Although the four graphs might look a little different to you, all four graphs are effectively the same, in that they all represent the same town, and all would have the same solution.









这 4 张图是如何来表示同一座城市的？

你能画出与那座拥有 5 栋房子的城市对应的图吗？

How can all four graphs represent the same town?

Can you draw a graph that corresponds to the second town above with 5 houses?

小游戏 13.2 大一点的泥泞城市

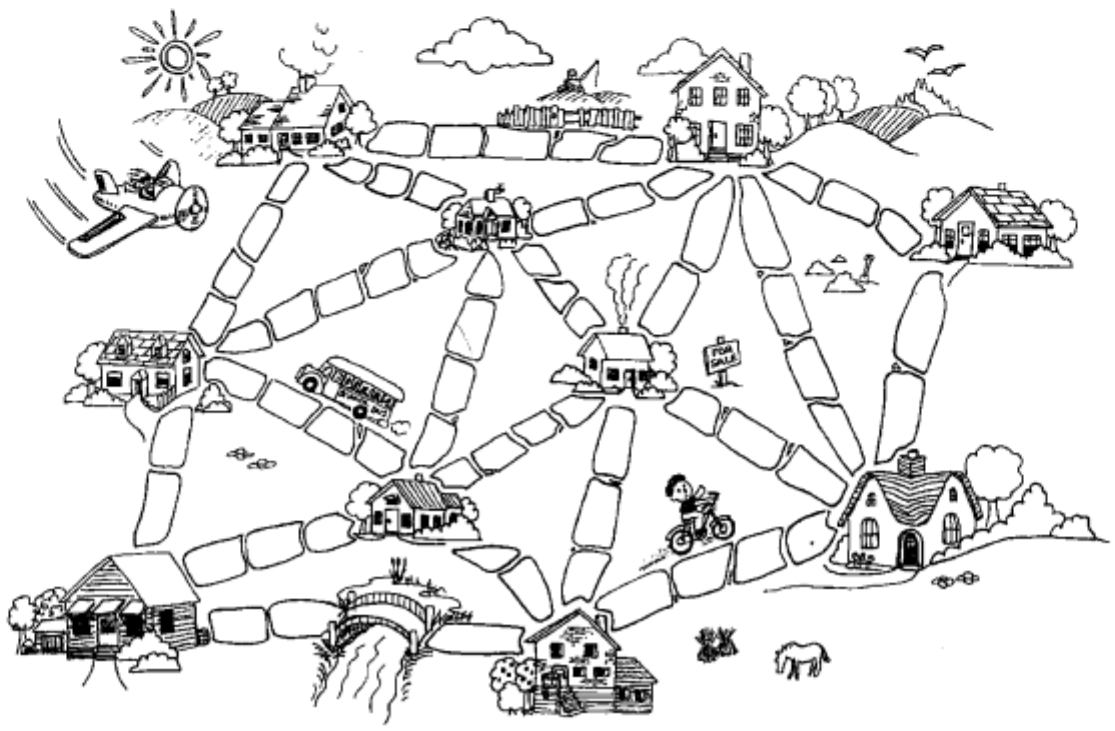
Activity 13.2 A larger muddy city

这里有一张大一点城市的布局图，每栋房子之间的石砖数量反应了修这条道路的费用（桥算作一块石砖）。

在使用最少的标记（铺路的石砖）的情况下找出连接全部房子的最佳路径。（提示：如果你的设计方案中有一个“循环路线”，即从一条路离开一栋房子后还能走另外一条路回到这栋房子，那么说明你用的石砖数量太多了。）

Here is the layout of a much larger city. The number of paving stones between each house represents the cost of paving that route (the bridge counts as one paving stone too).

Find the best route that connects all the houses, but uses as few counters (paving stones) as possible. (Hint: your solution has a "cycle" in it if it is possible to leave a house and return to it by a different path; if you ever have a cycle in your solution then you have used too many stones.)



你能画出与这座城市对应的图结构吗?

Can you draw the graph corresponding to this city?

进阶篇
Details for experts

实际中的网络往往含有冗余,如果一条连接失效了,那么还能从另一条连接连过去。然而,这里我们讨论的模型却是从各种最值问题衍生而来的,特别是有关地图和网络的问题,所以这种表示方法对计算机科学来说很重要。图(graph)、节点(node)以及树(tree)的思想对于表示问题模型来说非常有效。

In practice, networks usually have redundancy in them, so that if one link fails, the network is still connected another way. However, the issues that we looked at in this module come up in all sorts of optimization problems, particularly relating to maps and networks, so this notation is very important in computer science. The idea of a **graph**, **node** and **tree** are very useful for representing problems.

有趣的事
Curiosity

除了最小生成树算法之外，还有许多其他应用于图结构的算法能帮助现实世界的网络解决问题。比如，寻找两点间最短距离的算法决定了如何将你发出的 email 传送给你的朋友。如果一封相同的电子邮件被同时抄送给许多人，你需要一种算法用来寻找访问所有接收方的最短路径，这个问题通常被称为“旅行商问题”（traveling salesperson problem），即一个销售员如何访问居住在不同城市的客户。

There are many other algorithms besides minimal spanning trees that can be applied to graphs, which are used to solve problems for real－world networks. For example, algorithms for finding the shortest distance between two points can be used to decide how to route an email message to your friend. If the same email goes to lots of friends, you need an algorithm to find the shortest route that visits all the points. This problem is often referred to as the “traveling salesperson problem”, because it applies to a salesperson who has to visit many cities using the road network.

有些图问题异常困难，有些则非常简单，像是寻找两点之间的最短距离。计算机科学家们迄今还未在任何网络中找到旅行商问题的最优解，或许对于 GPS 自动定位系统来说很容易就能找出从一个地方到另一个地方的有效路径，但是让一个销售员穿梭于不同地点并拿到全部包裹的最佳路径就很难找出来了！

Some problems on graphs are surprisingly difficult, while others, like finding the shortest distance between two points, are easily solved. Computer scientists have not yet discovered fast enough methods that find the best possible solution to the traveling salesperson problem in any network. It's easy for a GPS automobile navigation system to find efficient routing instructions from one point to another, but it's a very hard problem for it to plan the absolutely best route for a courier van to go around and pick up some parcels!

“世界旅行商问题”（world traveling salesperson problem）用于寻找世界上 1904711 座不同城市的最佳访问路径——其中两个数据库包含世界上全部注册了的城市或小镇，甚至还包括了一些南极的研究基地。下图中的每个城市只能用小点来表示，它们的数量巨大到几乎拼出了一幅世界地图。

The “World Traveling Salesperson Problem” is to find out how best to visit 1, 904, 711 cities throughout the world—all locations from two databases that are registered as populated cities or towns, plus several research bases in Antarctica. Here are the cities you would have to go to shown by a dot for each city—there are so many that it almost makes a map of the world:

没有人能解开这道难题。但是人们离这个目标只差一点点:在这个问题被提出的 2002 年,算出的最短方案长度为 7524170.430 千米;到了 2007 年,一些数学家证明路径最短应为 7512218.268 千米(这也被称为"下界证明",这是另外一个难解的课题);2008 年 11 月,有人写了一个计算机程序,找到了 7515947.511 千米的路径解决方案,比下界仅多出了 0.0497%。最短路径应该介于这两值之间,但是直到现在还没有人知道正确的答案!

No-one knows the best route. But people have come very close. When the problem was first suggested in 2002, the shortest known tour was 7524170.430 km long. In 2007 some mathematicians proved that any tour had to be at least 7512218.268 km (this is known as a "lower bound" proof, and is a difficult thing to do), In November 2008 someone wrote a computer program that found a tour 7515947.511 km long: that's only 0.0497% more than the lower bound. The best tour has to be somewhere in between these two, but no - one on earth knows what it is!

上图引用自 http://www.tsp.gatech.edu,在这个网站中,你能找到旅行商问题的很多有趣应用。如果你能写出一个计算机程序解决这个问题的话,就能赢取一百万美元的奖励哦!

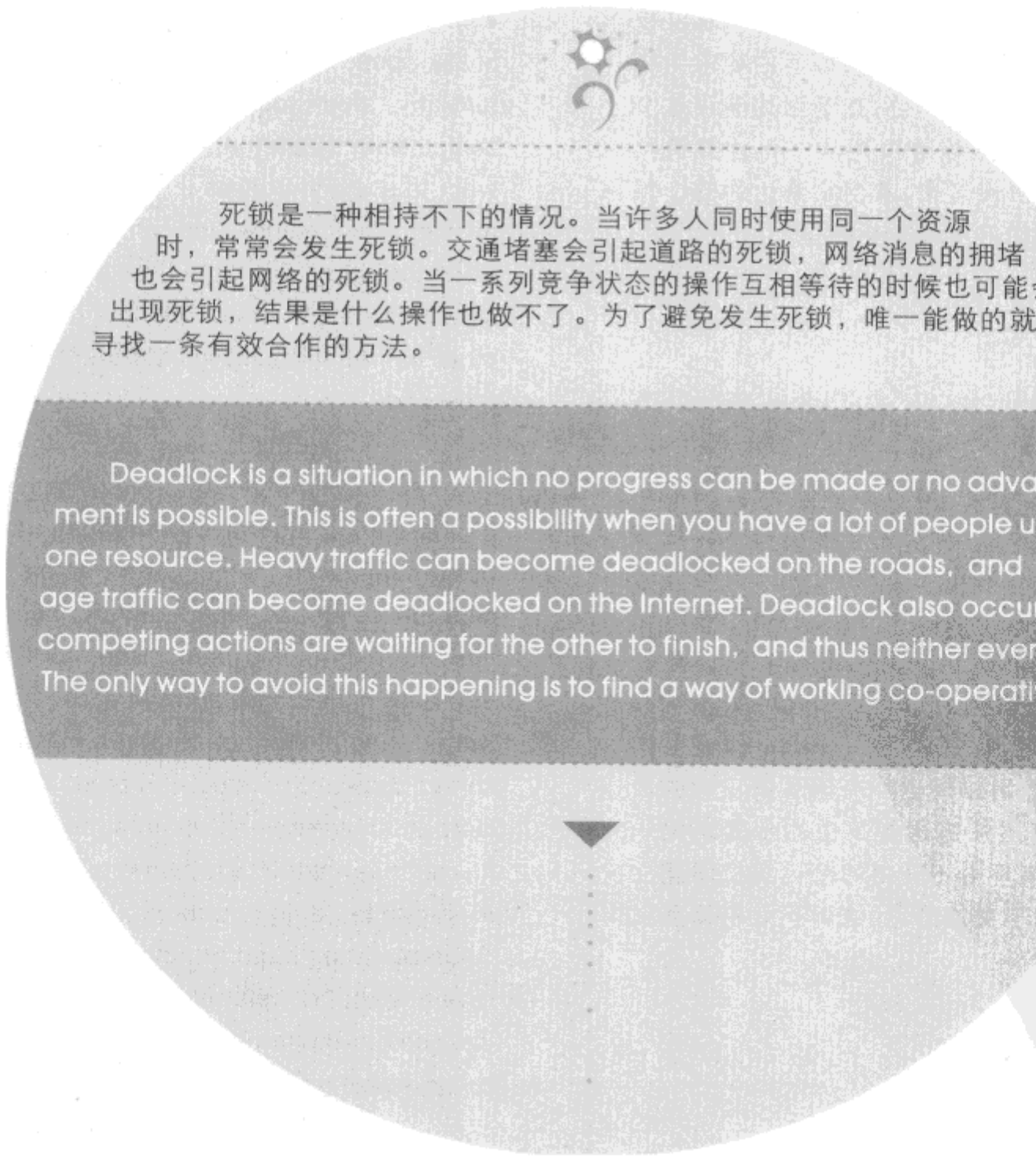
The image above is from http://www.tsp.gatech.edu, where you can find a lot more information about many interesting applications of the Traveling Salesperson Problem, and find out about a $1000000 prize that you would win if you worked out how to write a computer program that could solve this problem efficiently.

第14章

路由和死锁

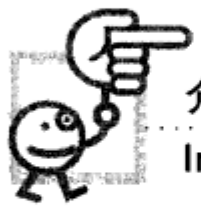
01

Routing and Deadlock



死锁是一种相持不下的情况。当许多人同时使用同一个资源时，常常会发生死锁。交通堵塞会引起道路的死锁，网络消息的拥堵也会引起网络的死锁。当一系列竞争状态的操作互相等待的时候也可能会出现死锁，结果是什么操作也做不了。为了避免发生死锁，唯一能做的就是寻找一条有效合作的方法。

Deadlock is a situation in which no progress can be made or no advancement is possible. This is often a possibility when you have a lot of people using one resource. Heavy traffic can become deadlocked on the roads, and message traffic can become deadlocked on the Internet. Deadlock also occurs when competing actions are waiting for the other to finish, and thus neither ever does. The only way to avoid this happening is to find a way of working co-operatively.



介绍
Introduction

网络负责从一台计算机传送消息到另一台计算机直到它到达目的地。当你给朋友发送电子邮件时，计算机会先将它传给你所在城市的另外一台计算机，然后将电子邮件转发到离接收方较近的另外一座城市，直到它被最终传送到你朋友的计算机上。同样地，当你点击网页上的一条链接后，你的计算机便提交一项请求页面的要求并将它从一台计算机传到另一台，直到到达储存这个网页的计算机。然后该网页再从一台计算机到另一台计算机被回传过来，直到它到达你的计算机。而所有这一切只发生在一瞬间！

The Internet works by passing messages from one computer to another until they reach their destination. If you send email to a friend, your computer might send it to another computer in your local town, which then sends it on to another town nearer the destination, until it is finally passed to your friend's computer. Similarly, when you click a link on the web your computer formulates a request for the page you want to see and passes it from one computer to another until it reaches the computer that actually holds the page. The page itself is then sent back from computer to computer until it arrives back on yours. All this usually happens in a fraction of a second!

你可以把网络想像成一个高速公路系统，信息在繁忙的路上奔驰着，计算机们在岔口处认真地接收着每条信息，然后将它们转发到正确的目的地。和真正的道路不同，这些路上的每个岔口处每秒钟都有数以千计的信息们到达。将信息发送给正确目的地被称为“路由”(routing)。事实上，常常有一个名叫“路由器”(router)的设备作为网络的一部分，用以传送所有信息到正确的计算机。如果我们不按照这些路由的步骤来发送信息，那么我们将不得不在每两个计算机间建立一个直接连接用于交流信息，这弄不好将需用到无穷根网线！

You can visualize this by imagining the Internet as a system of highways. Messages travel along busy routes, and the computers at the junctions must be careful to receive every message and forward it on to the right place. Unlike roads, thousands of messages arrive at each junction every second. The problem of sending a message to the correct destination computer is called "**routing**". In fact, it is common to have a device called a "router" as part of a network to send all these messages to the right computer. If we couldn't send messages along steps like this, we would need to have a direct link between every pair of computers that need to communicate, and that would involve an impossible amount of wiring!

视频:请观看"橘子游戏"的视频。

Video: watch the "Orange Game" video.

小游戏 14.1 橘子游戏
Activity 14.1 The Orange Game

在这个游戏中你将要来切身体验一下如何在网络中传递信息。为了让游戏更容易进行,我们设定在你的网络上只有 5 台计算机,每次只有一条或两条信息传送给每台计算机。我们必须牢牢记住规则,即网络上的每台计算机每次不能同时让两条以上的信息停留。

You are going to get some first-hand experience passing messages around a network. To keep things manageable, we are only going to have about 5 computers on your network, and there are just one or two messages being passed to each computer. However, we will be imposing the restriction that each computer on the network cannot hold more than two messages at once.

这个游戏最理想的玩法是找来 5 个同伴一起玩,否则你需要用下图所示的棋盘来进行游戏。如果你和伙伴们一起玩这个游戏(就像视频所示的那样),你可以将每个人标上号,如字母 A、B、C 等,并为每个人准备两张卡片,上面标上同样的字母;为了更有趣,你也可以让每个人穿一件不同颜色的 T 恤,并分配两个与他们 T 恤颜色相同的水果给每个人,以对应两条信息。我们要求除了一个人只有一条与自身字母编号(或 T 恤颜色)匹配的信息外,其他每个人都有两条信息。由于每只手都只能拿一条信息,因此有一个人将空一只手出来,而其他人每只手上都拿着一个水果(或卡片,如果你使用的不是水果而是卡片的话)。

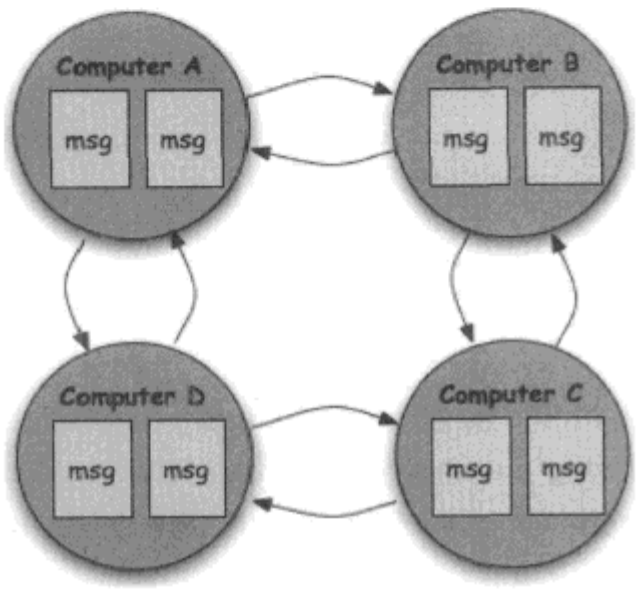
It is ideal if you can find a group of 5 people to do this activity; otherwise you can do it as a board game using the layout below. If you are doing it with a group (as in the video and the photo above), you can either label each person with a letter A, B, C etc., and have two cards for each person with the same label on them, or (for more fun) each person can wear a different colored t-shirt, and the two messages for them are pieces of fruit that are the same color as their t-shirt. There are two messages for each person, except for one person who only has one message with their color or letter on it. Each hand can hold just one message, so just one person will have one empty hand; all the others will be holding one piece of fruit (or a card if you're not using fruit) in each hand.

如果是和同伴们来玩游戏，你的小组成员应该像视频中那样围成圆圈坐在一起。

Your group should sit in a circle as in the photo above.

如果你找不到这么多小朋友来做游戏，准备 7 枚游戏币放在下图中，标上 A、B 和 C 的各两枚，一枚标上 D。这种情况下，游戏币将是你的“信息”。

If you don't have a group of people, make up 7 counters to put on the following diagram, 2 each labeled A, B and C, and one labeled D. In this case the counters are your "messages".



初始情况下，“信息们”（水果、卡片或游戏币）将随机放在这个网络中，每台“计算机”（如果以小组来玩的话，就是每个人）将拥有两条任意的信息，除了一个空位外。

Initially the "messages" (fruit, cards or counters) are dealt randomly around the network. Every "computer" (or person if you're using a group) will have two randomly assigned messages, except there will be one empty space.

游戏的目标是“路由”（route），即将每条信息转发给拥有相同名字或颜色的“计算机”，但你只能将信息传递给你相邻的计算机（人）。如果你玩的是棋盘的版本，你只能沿着箭头移动信息，最终所有“计算机”都需要拿到属于它们的正确的信息。由于每台“计算机”一次只能“拿着”两条信息，因此当你开始传递信息的时候，只有与有空位的“计算机”相邻的“计算机”才能传递信息。应注意每台“计算机”每次只能传递一条信息。

Your goal is to route, or send, each message to the computer with the same name or color, passing messages only to the two computers (people) beside you. In the board game version, you can only move messages along the arrows. At the end, all of the computers should be holding their correct messages. While you are routing the message to the correct destination, each computer has room for only two messages at a time, so only the computers next to the one with the empty space can pass a message to it. Only one message at a time can be passed between computers.

a 将信息随意分配给每台“计算机”。你能在不违反游戏规则的情况下完成信息的传递任务吗(只能传给隔壁的人,只能传给空着的那只手)?

a Randomly assign the messages to the "computers" around the network. Can you route the messages to the correct destination without breaking the rules (only pass to a neighbor, and only pass to an empty hand)?

当你传递信息的时候,你需要控制网络中的全部信息,而且你可以控制每台“计算机”每步应该做什么。这和真实网络中计算机的运行方式不同,因为真实情况下每台计算机都是自主运行的,并没有人告诉它们每一步必须怎么做。

When you were routing messages, you had control over all of the messages in the network and you could control what each computer did at each step. This is different than computers on a real network because there, each computer acts independently with no one telling all of the computers what they should do at each step.

尽管所有计算机都为一个共同目标工作着,但计算机有时候却会使用到贪婪算法(greedy algorithm),即在每一时刻每台计算机都试图按照能让自己获得最大利益的方式来运行。在贪婪路由算法中,一旦某台计算机收到它的目标信息后,它就不会再让这条信息离开自己了。

Sometimes computers use greedy algorithms, even though they are all working toward a common goal. A **greedy algorithm** is an algorithm in which every computer tries to do the best that it can for itself at each moment in time. A greedy routing algorithm could state that once a computer receives one of its destination messages, the computer holds onto it and won't let it go.

b 你认为贪婪算法在我们的路由游戏中行得通吗?

b Do you think a greedy algorithm works for this routing game?

贪婪算法会导致 4 台计算机出现下列后果:计算机 A 和 C 拿到正确的信息,而计算机 B 和 D 却不能。

A greedy algorithm could lead to the following situation with four computers: Computers A and C each have their correct messages, while computers B and D do not.

c 如果计算机 A 和 C 都很贪婪,并拒绝放走它们的信息,正确的信息能被传递到计算机 B 和 D 吗?

c If computers A and C are greedy and refuse to let go of their messages, can the correct messages get to B and D?

在这个路由游戏中,贪婪算法将导致死锁(deadlock)。而当出现死锁,游戏便无法继续下去,因为有人拿着别人需要的资源却不肯放手。

In this routing game, a greedy algorithm can lead to **deadlock**. When there is deadlock, progress cannot be made because somebody is holding on to a resource that someone else needs and they won't let go.

d 如果只有计算机 C 是贪婪的，正确的信息能被传递到计算机 B 和 D 吗？

除了在计算机间传送信息外，网络上还会出现其他情况的死锁。

e 你能举出几个真实世界中会发生死锁的例子吗？（提示：和人或者和车子。）

在许多网络中都存在路由和死锁，好比道路交通系统、电话和计算机系统等。工程师们花费大量时间来试图解决这些问题，并且试着设计出能更容易解决这些问题的网络。

d If only computer C is greedy can the correct messages get to B and D?

Deadlock happens in situations other than computers sending messages over a network.

e Can you think of any other examples where deadlock could happen in the real world? (Hints: With people or with cars.)

Routing and deadlock are problems in many networks, such as road systems, telephone and computer systems. Engineers spend a lot of time figuring out how to solve these problems—and how to design networks that make the problems easier to solve.

进阶篇
Details for experts

在实际应用中，网络上的节点在等待传送信息的过程中可以储存大量信息，这一现象被称为缓冲（buffer）或队列（queue）。然而，缓冲的大小取决于计算机上的可用空间，如果缓冲溢出，这台计算机要么将无法接收任何信息直到缓冲队列重新为空，要么会丢失传送过来的信息。橘子游戏中使用的“缓冲”只能储存一条“信息”（即只有一只手空出来），但是计算机中的缓冲通常能一边等待下一台计算机或网络设备准备好接收下一条信息，一边自己储存大量信息。

In practice, the nodes in a network can store large numbers of messages while waiting to pass them on, in what is called a **buffer** or **queue**. However, the size of buffers always has to be limited by the space available on the computer, and if a buffer overflows then either the computer can't receive data until the queue empties, or incoming data will be lost. The "buffer" used in the above activity could only store one "message" (i.e. there was just one free hand), but buffers on computers can usually store dozens of messages while they wait for the next computer or network device to be ready to receive the next message.

过溢的缓冲可能暴露出计算机被攻击的弱点。比如,在设计糟糕的系统中,一个“拒绝服务”(DOS)攻击会同时发送大量信息给这台计算机,一旦这台计算机的缓冲队列被填满,那么它将丢失接下来进入的信息,包括它自身需要的合法信息。在设计系统时要尽量杜绝这种情况发生,否则一个计算机网络甚至一个政府通信系统将有可能被其他国家的人恶意破坏。

Overfull buffers can be exploited to attack a computer; for example, a “denial of service” (DOS) attack may be possible on a badly designed system where the attacker sends many messages at the same time to a computer. If these fill the computer's buffer then it may end up losing some of the messages coming in, including legitimate ones. Systems need to be designed to prevent this sort of thing being possible, since otherwise a company network, or even a government communication system, could be incapacitated by someone in another country.

有趣的事
Curiosity

一个中世纪的悖论(布里丹之驴)讲述的是,一只驴子在周围有充足草料包围的情况下给活活饿死了。问题的重点在于,驴子的身边有两个距离完全相等的草堆,它站在两个草堆正中间却无法下定决心走向哪一边。但是这个故事有一个圆满的结局:最终,驴子因为太饿而昏倒,它倒下的时候头稍微偏向了其中一个草堆,然后它挣扎地爬了过去……这是一个只含有单一主体的死锁情况——尽管你或许会说这驴子明明有两个心眼(吃到两边的草)……

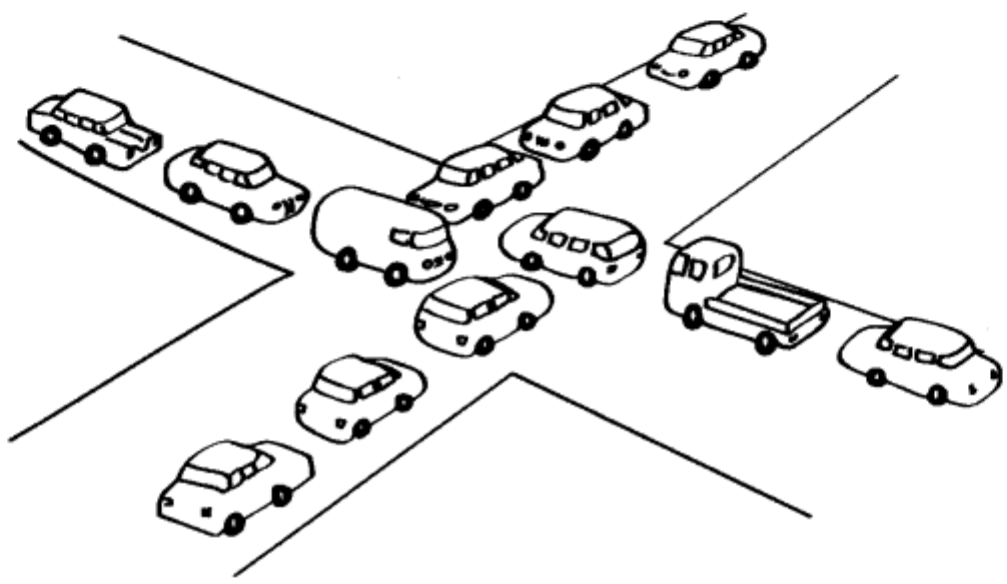
A medieval paradox (called Buridan's Ass) tells of a donkey who starved despite there being plenty of hay nearby. The problem was that the hay was in two equal piles and he was exactly halfway between them: he couldn't make up his mind which to go for. But the story has a happy ending, for finally, faint with hunger, he fell over, putting his head a little nearer one pile, and he managed to stagger there .... This is a kind of deadlock situation involving a single agent – though you could say he was in two minds ...

通常情况下的死锁涉及许多主体。大城市里的交通堵塞导致数以千计的司机被困在路上,无法动弹。这样糟糕的状况一般总由交通事故引发,往往只能靠人为的干预(如交警介入)才能最终得到解决。同样的事情也会发生在网络中。一个意外导致一台计算机出了故障,并改变了正常

Usually deadlock involves lots of agents. Big-city traffic jams bring tens of thousands of drivers to a standstill, trapped on the roads. Usually, these are triggered by accidents. And they are solved by human intervention – the traffic police move in. The same thing sometimes happens with

的信息流量，接下来情况会越来越糟糕，直到人们发现并解决这个问题。

the Internet. An accident – a computer breakdown – alters the normal flow of traffic, and from there things go from bad to worse until someone notices and sorts the problem out.



比起人类来说，计算机更像是布里丹之驴——它们没有大脑。一旦发生死锁，它们会一直等到有人介入，否则就像那只驴子一样，耗尽电力。你的计算机曾经陷入一些奇怪的状态没有？比如对鼠标和键盘毫无反应？那么恭喜你遇到死锁了。其实，仅仅只要关掉它并重启（或给它一点干草），一切便迎刃而解。

Computers are more like Buridan's Ass than like people – they're brainless. Once deadlocked, they will wait forever until someone steps in, or until, like the ass, they run out of power. Has your computer ever got stuck in some strange state, not responding to mouse or keyboard? Deadlock. Just switch it off and reboot (or give it some hay).

第15章

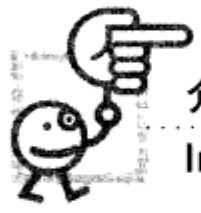
处理输入

01

Processing Input

无论是敲击键盘还是点击屏幕上的按钮，计算机都需要理解它接收到的输入信息。有许多方式可以完成这项工作，其中一个简单而有效的表示方法是有限状态自动机（finite state automaton）。有限状态自动机是一种可以有效分析各种计算机问题的逻辑图，特别适合对计算机的输入进行处理。在这一章里，我们将要试着画出这些图，同时也要理解不同逻辑图的意义。

Whether the input to a computer comes from someone pressing keys on the keyboard or clicking buttons on the screen, somehow the computer has to make sense of the sequence of input that it receives. There are many ways to do this, but one approach that is very effective is a notation that is a simple idea with a complicated name: the **finite state automaton**. A finite-state automaton is a kind of logical map that is a useful way of thinking about many different computer problems, and especially processing the input to a computer. In this topic we experiment with drawing these maps, and also figure out the meaning of maps that other people have drawn.



介绍
Introduction

计算机程序通常需要处理一系列的符号，例如文档、网页中的字母或词语，甚至包括另一个计算机程序中的文本。计算机科学家们常常使用一种称为有限状态自动机(FSA)的流程图来处理这些输入。尽管这个名字听起来挺复杂的，但 FSA 其实就是一张能够指导计算机处理输入的简单流程图。顺带一提，“Automaton”(动态机)的复数形式写作“Automata”[9]，稍后我们会进一步谈到这个不寻常的名字。

Computer programs often need to process sequences of symbols such as letters or words in a document or web page, or even the text of another computer program. Computer scientists often use a scheme called a **finite-state automaton (FSA)** to do this. Although the name sounds complicated, an FSA is really just a simple map that's easy to follow. By the way, the plural of “Automaton” is “Automata”—but we'll talk more about the unusual name later.

尽管这些流程图非常简单，它们却能适用于任何问题，不仅能处理键入计算机的文本信息，还能帮助设计计算机的接口，甚至还能找出图像中的图案规律。但是在一开始，我们还得从类似 FSA 的东西着手讲起——那就是藏宝图！

Even though these maps are quite simple they can be used for all sorts of things. They process the words that people type into a computer. They help in designing computer interfaces. They can even find patterns in images. But to start off with we will be working with something equivalent to an FSA—treasure maps!

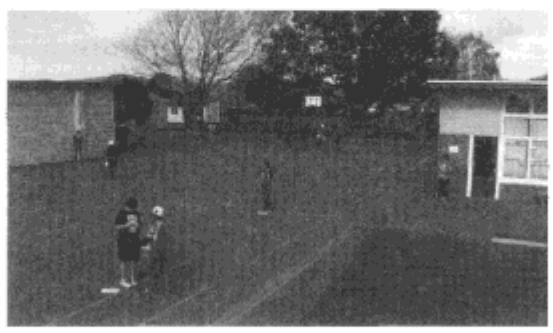
小游戏 15.1　金银岛
Activity 15.1　Treasure Island

你既可以和几个伙伴一起来玩这个游戏(见视频)，也可以自己一个人来玩(用卡片)。两种玩法都将在下文提到。

This activity can either be done with a group of people (as in the video), or on your own (as a kind of card game). Instructions for both approaches are given in the activity.

[9] 译者注：英文中常规复数形式一般直接在单词后面加 s。

视频:请观看"寻宝游戏"视频。



设想一下你现在身处于只有岛屿的世界,海盗船们来往于不同的岛屿之间,幸运的是,海盗们都非常友善,而且乐于让旅行的人"搭便船"。每个岛屿配备了两艘船 A 和 B,你可以任选其一开始你的旅途。每当你到达一个岛屿,你都能再选一艘船 A 或 B(但不是同时选两个)。比如,在后面的小地图中,如果你从海盗岛启程,搭乘船 A,你将会抵达船难湾;如果你再次搭乘船 A,你将回到海盗岛。

a 如果你从海盗岛启程,搭乘 B 船,再乘 B 船,再乘 A 船,你将会抵达哪座岛屿呢?

b (假设你从海盗岛启程)如果你搭船的顺序是船 A、船 A、船 A,你将会抵达哪座岛屿呢?

现在我们换一张有 7 座岛屿的地图,你的目标是找到从海盗岛到金银岛的路线。唯一的问题是地图上没有标上箭头哦!你需要自己去探索旅行的路线。为了达成这个目标,你可以问每座岛上的 A 船或 B 船各驶向哪里。下面我们将来解释一下自己一个人或和伙伴们一起是如何玩这个游戏的。

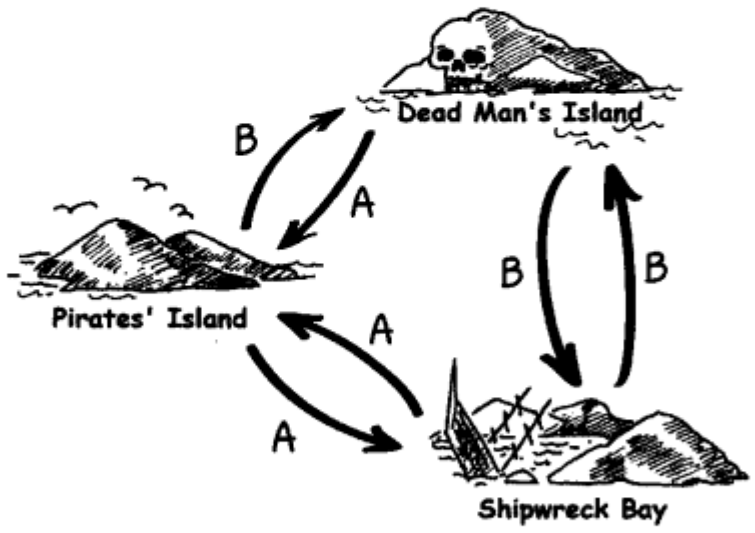
Video: watch the "Treasure Hunt" video.

Imagine that you live in world of islands, with pirate ships sailing between the islands. Luckily they are friendly pirate ships, and they offer rides to travelers. Each island has two departing ships, A and B, which you can choose to travel on. As you arrive at each island you may ask for either ship A or B (not both). For example, in the small map below, if you start at Pirates' Island, and take ship A, you'll end up at Shipwreck Bay. If you then take ship A, you'll get back to Pirates' Island.

a If you start at Pirates' Island, and take ship B, then B, then A, where do you end up?

b What if (starting at Pirates' Island) you take the sequence of ships: A, A, A?

Now we will use a different map with seven islands. Your goal is to find a path from Pirates' Island to Treasure Island. The only problem is, there are no arrows on the map! You need to discover what the routes are for yourself. To do this, you can ask to take ship A or B from the island that you are on, and you'll be sent to another one. We will explain below how to do this game, either on your own, or with a group of friends.



在本章最后附有每个岛屿上 A 船和 B 船的驶向说明卡。如果你是和伙伴们一起玩这个游戏的，选 7 个人来当岛屿，其中 6 个人拿着每个普通岛屿对应的说明卡，第七个人来代表金银岛，其他人需要来回在岛屿之间找到下一个驶向的岛屿。如果你是自己玩这个游戏的，可以将卡片背面朝上放在桌上，每驶向一个岛屿，将代表它的卡片翻开，从而找到你的下一个目的地。

The instructions for where ship A and B go from each island are on the cards at the end of this topic. If you can play this game with a group of people, choose 7 people to be the islands – 6 of the people get the cards for a particular island each, and the 7th person represents Treasure Island. Everyone else will run around from one person to another finding out which island to go to next. If you play this game on your own, you can just place the cards face down on the table and turn over the one corresponding to the island and ship that you are going to take, to find out where to go next.

在游戏准备阶段，你可以将书中提供的卡片剪下来(如果你要一个人玩这个游戏，试着不要去看卡片上的说明文字)，也可以复印一张或者从网站上打印出一页新的。由于文字是印在外侧的，你可以将卡片对折使用。

To prepare the game, cut out the cards at the end of this topic (try not to read what is on the cards if you are going to play on your own)! You can copy the page, or print out a new version from the web site. Fold the cards in half so that the writing is on the outside.

如果你是独自一人玩这个游戏的，把卡片放在桌子上，让有问号的一面朝上。为了“询问”你所在岛屿的每艘船的目的地，只要将对应的 A 或 B 卡片翻过来即可。卡片背面上将会写着下一个驶向的岛屿是哪一个。

If you are playing on your own, put the cards on the table so that the side with the question mark is facing up. To “ask” for a particular ship from your current island, just turn over the card for the island with the corresponding “A” or “B” on it. The back will show you which island to go to next.

如果你是和伙伴们一起玩这个游戏的，那么将 6 套卡片交给对应的“岛屿”，然后让他们随意站在房间里。（最好大家在外面的操场来玩这个游戏！）

If you are doing this game with a group, give out the 6 sets of cards to people who will be the "islands", and get them to spread out around the room. (Or even better, around a playground outside!)

如果你是要寻找金银岛的人，你可以准备一张空白地图（如下图），选择 A 船或者 B 船从海盗岛启程，并在到达下一个岛屿前，在地图上画上箭头和做好标签，因为过一会你可能会兜回来，这样就能记起你上一次选择的路线了。

If you are one of the people solving the puzzle then take the empty map, and starting at Pirates' Island, choose either ship A or B. You will then be sent to the next island, but before going there, make sure you draw an arrow on your map and label it, because you might find yourself back there again, and may want to remember which path you took last time.

c 你能找到通往金银岛的路线吗？请标记好你在地图上发现的路线。

c Can you find a path to Treasure Island? Remember to mark the routes you discover on your map as you go.

d 如果你已经到达了金银岛，但并没有访问到地图上全部地点，请继续旅程直到地图上每个岛屿之间的路径都被走过一遍（这样你便知道每座岛屿上 A 船和 B 船的行驶方向了）。

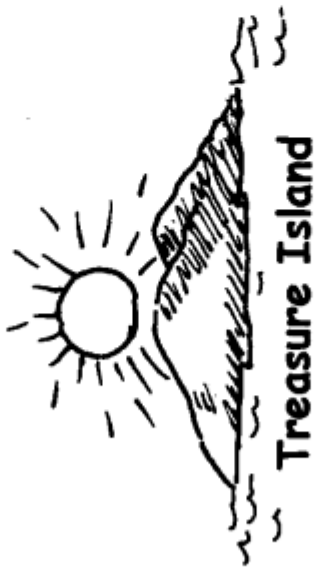
d If you got to Treasure Island but didn't create a complete map, carry on around the islands creating a map of all possible paths between islands (you'll then know where both A and B go from each island).

e 当你完成了整幅地图之后，能指出去金银岛的最短航线么？

e Now that you have the complete map, what is the quickest route to Treasure Island?

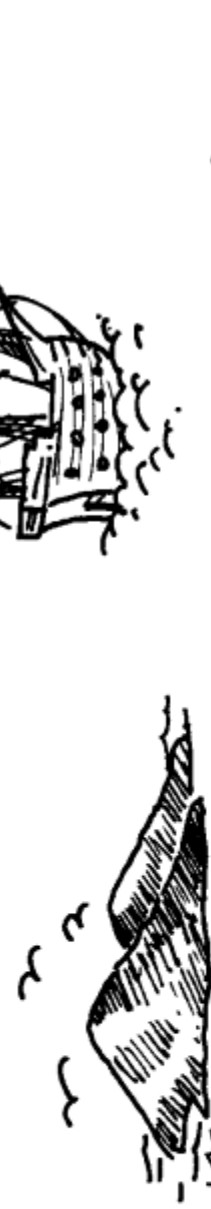
f 如果需要途经所有岛屿后再抵达金银岛，这条稍微慢一点的航线又是哪一条呢？

f Can you find a slower route to Treasure Island, such as one that goes to all of the islands?


Treasure Island


Smugglers' Cove


Shipwreck Bay


Pirates' Island


Dead Man's Island


Mutineers' Island


Musket Hill

g 你能找到包含有回路(即访问某些岛屿一次以上的)并最终抵达金银岛的航线吗?

g Can you create a route to Treasure Island that involves loops (that is, that visits some islands more than one time)?

小游戏 15.2 用 FSA 来寻找规律

Activity 15.2 Using an FSA to find patterns

有限状态自动机可以帮助计算机处理一连串的字符或事件。

Finite-state automata are used in Computer Science to help a computer process a sequence of characters or events.

举一个简单的例子。当你打电话时,往往会听到提示语音"请按 1 键……请按 2 键……请按 3 键转总台",而你按下的这些按钮将输入到电话那端的有限状态自动机中。提示语音的内容或许很简单,或许很复杂,有时候甚至让你转了一大圈又回到开始,因为在有限状态自动机中可能遇到了一个罕见的循环。当发生了这种情况,说明在设计系统时出现了错误,而这对于正在打电话的人来说可真是一件倒霉的事儿!

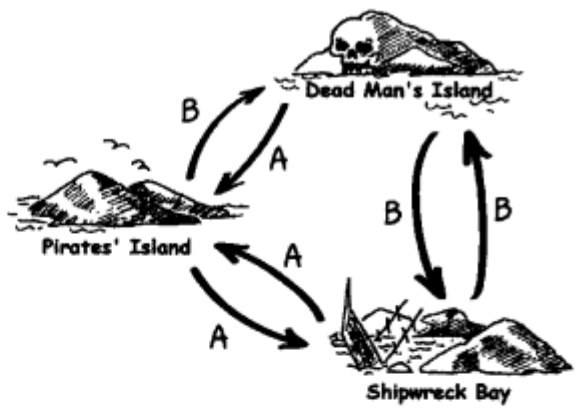
A simple example is when you dial up a telephone number and you get a message that says "Press 1 for this … Press 2 for that … Press 3 to talk to a human operator". Your key presses are inputs for a finite state automaton at the other end of the phone. The dialogue can be quite simple, or very complex. Sometimes you are taken round in circles because there is a peculiar loop in the finite—state automaton. If this occurs, it is an error in the design of the system—and it can be extremely frustrating for the caller!

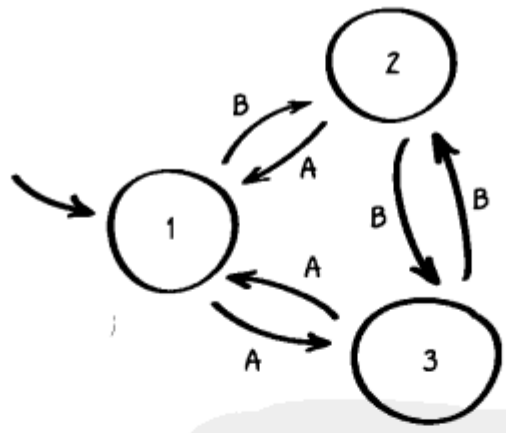
再举一个例子。当你去银行的自动提款机上取现金时，这台机器的计算机程序将指示你的操作。在这个程序中，所有可能的用户操作都被保存为有限状态自动机的形式。你每按一个按钮，有限状态自动机便自动将你导向图中的另一座“岛屿”，这些“岛屿”在计算机中有相应的指示，比如“提取 100 元现金”“打印回条”或者“取出您的磁卡”。

Another example is when you get cash from a bank cash machine. The program in the machine's computer leads you through a sequence of events. Inside the program all the possible sequences are kept as a finite-state automaton. Every key you press takes the automaton to another "island" in the map. Some of the "islands" have instructions for the computer on them, like "dispense $ 100 of cash" or "print a statement" or "eject the cash card".

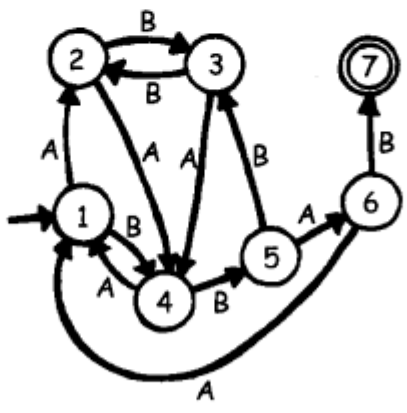
计算机科学家们使用圆圈来代表每座岛屿，用箭头指示移动的方向，从而用有限状态自动机的形式画出寻宝地图。比如，之前第一幅画有死亡岛的海盗地图能表示为如下的 FSA。

Computer scientists draw maps of finite state machines using circles to represent each island and arrows to show how to move from one island to the next. For example, our first Pirate Map with Dead Man's Island would be represented as an FSA like this:





同样地，你在小游戏 15.1 中探索的金银岛可以被表示成如下的 FSA。



Similarly, the map to Treasure Island that you discovered in Activity 15.1 would be represented with an FSA like this:

在上图中，最终藏有宝藏的岛屿用双线圆圈表示。用 FSA 术语来说，一个岛屿被称为一个状态(state)，“金银岛”则被称为终结状态(accept state)。之所以称为状态是因为它说明了在有了之前的输入事件后(输入 A 和输入 B)你来到进程中怎样的结果。初始状态由没有标记的箭头指示(通常是数字 1)，到达“终结状态”说明输入是“能被接受的”(accceptable)。在我们的例子中，最终拿到宝藏说明用海盗岛作为初始状态可以被接受，但是在计算机系统中，判断能否接受初始输入的理由往往更加通俗，例如去看它是否构成一个有效的命令序列。

In this example, the final island with the treasure is shown with a double circle. In FSA terminology, an island is called a **state**, and "Treasure Island" is called the **accept state**. It's called a state because it indicates what "state" you are in as a result of what has happened so far (the sequence of A and B inputs). The starting state is the one with an unlabeled arrow pointing at it (usually number 1). Getting to the "accept state" means that the input was "acceptable" —in our case, it was acceptable because we got to the treasure, but on computer systems there are usually more mundane reasons that the sequence of letters was acceptable, such as making up a valid sequence of commands.

术语一点通

The phrase "**Finite State Automaton**" (FSA) might seem complex, but each word is quite simple. "Finite" just means that there is a limited number of states (islands) in the map; the "state" is just as another name for the islands we were using; and "automaton" is an old word meaning a machine that acts on its own, usually following very simple rules (such as the cuckoo in a cuckoo clock). Sometimes an FSA is called a **Finite State Machine** (FSM), or even just a "state machine".

有限状态自动机(FSA)这个名字听起来似乎很复杂，但是名字中各组成部分的意思却非常简单。“有限”(finite)指在逻辑图中有有限数量的状态(如岛)；“状态”(state)是刚才游戏中岛屿的别称[10]；“自动机”(automaton)是能遵循简单规则自主运行的机器的旧名(就好像布谷鸟时钟中的布谷鸟)。有时候，人们也称 FSA 为有限状态机(finite state machine，FSM)或简称为“状态机”。

[10] 译者注：state 在英文中有“州”之意。

h 利用上面提到的 FSA，哪一条是到达金银岛最有效的路径(用每步选择的字母来表述)?

h Using the FSA above, what is the most efficient way to get to Treasure Island (express this using the letters to choose at each step)?

i 到金银岛的路线中包含循环路线吗？能举出例子吗?

i Are there paths with loops in them that get to Treasure Island? Can you give an example?

j 路径 BBBABBABAB 能通往金银岛吗?

j Does the path BBBABBABAB lead to Treasure Island?

k 路径 BAABAAABABAB 能通往金银岛吗?

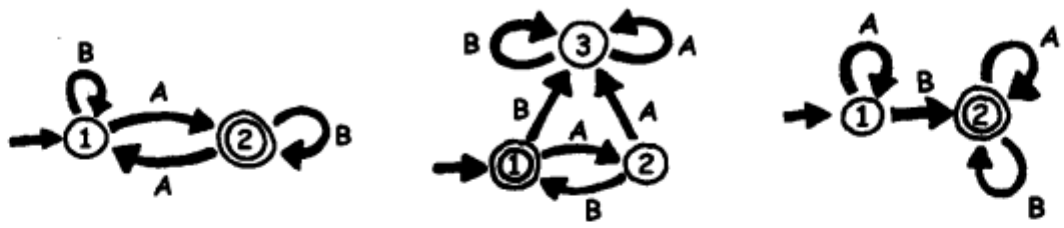
k Does the path BAABAAABABAB?

l 路径 BBBAAAB 能通往金银岛吗?

l Does the path BBBAAAB?

再来举 3 个有关有限状态自动机的例子。

Here are three more examples of Finite State Automata.



(a) (b) (c)

在图(a)中，下面哪条指令能被接受(即能让你达到双线圆圈状态)?

In map (a), which of the following strings will be accepted? (That is, do you end up in the state with the double circle?)

m AB

m AB

n BABAA

n BABAA

o ABBBA

o ABBBA

p AAABABA

q AAABA

r 你能描述出能被图(a)接受的全部指令的共同点吗(提示:不妨数数 A 的数量)? 为什么呢?

在图(b)中,下面哪条指令能被接受?

s AB

t BA

u ABAB

v AABB

w ABABA

x ABBA

y 你能描述出能被图(b)接受的指令的共同点吗?

在图(c)中,下面哪条指令能被接受?

z B

aa BBB

bb BBA

p AAABABA

q AAABA

r Can you describe what is the same about all of the strings that are accepted by map(a)? (Hint: Count the number of As.) Why is this?

In map (b) which of the following strings of letters will be accepted?

s AB

t BA

u ABAB

v AABB

w ABABA

x ABBA

y Can you describe what is the same about all of the strings accepted by map(b)?

In map (c) which of the following strings of letters will be accepted?

z B

aa BBB

bb BBA

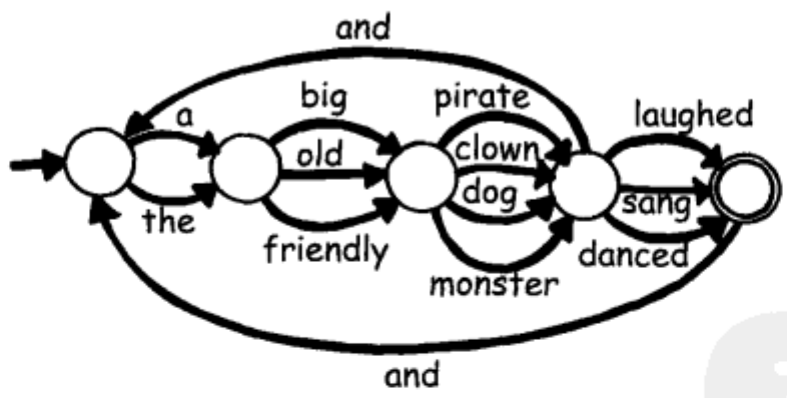
c c ABBA

d d AAA

e e 你能描述出能被图(c)接受的指令的共同点吗?

一些使用 FSA 的计算机程序专门用于处理文本中的句子,它们既可以自己造句,也可以处理用户输入的句子。

这里有一个方法,能借由选择状态图上任意的路径并记录得到的单词来造句。



f f 你能用这张状态图来造句吗?

g g 试着自己设计一个能造句的 FSA,其他人能使用你的 FSA 来造句吗?

c c ABBA

d d AAA

e e Can you describe what is the same about all of the strings that are accepted by map (c)?

Some computer programs deal with sentences of text using FSAs. They can both generate sentences themselves, and process sentences that the user types in.

Here is a way of constructing sentences by choosing random paths through the map and noting the words that are encountered.

f f Can you use this map to create a sentence?

g g Try creating an FSA to produce sentences of your own. Can someone else use your FSA to produce sentences?

小游戏 15.1 的卡片
Cards for Activity 15.1

Pirates' Island
海盗岛
A → ?

Shipwreck Bay
船难湾
A → ?

Shipwreck Bay
船难湾

Musket Hill
步枪山

Pirates' Island
海盗岛
B → ?

Shipwreck Bay
船难湾
B → ?

Musket Hill
步枪山

Dead Man's Island
死人岛

Musket Hill

步枪山

A → ?

Dead Man's Island

死人岛

A → ?

Pirates' Island

海盗岛

Musket Hill

步枪山

Musket Hill

步枪山

B → ?

Dead Man's Island

死人岛

B → ?

Mutineers' Island

暴徒岛

Shipwreck Bay

船难湾

Mutineers' Island
暴徒岛
A → ?

Smugglers' Cove
走私港

Smugglers' Cove
走私港
A → ?

Pirates' Island
海盗岛

Mutineers' Island
暴徒岛
B → ?

Dead Man's Island
死人岛

Smugglers' Cove
走私港
B → ?

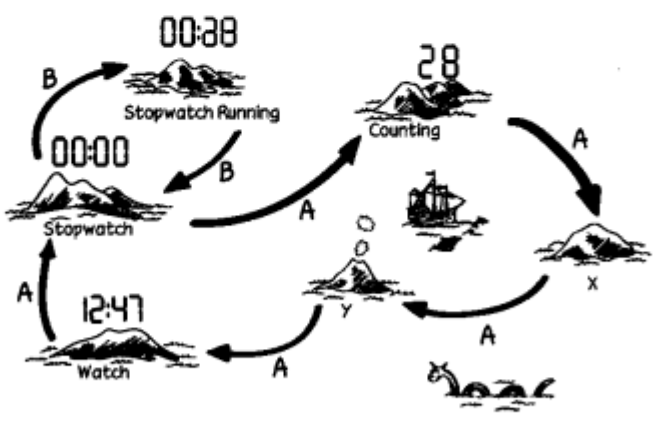
Treasure Island
金银岛

进阶篇
Details for experts

我们在本章中谈到的自动机被广泛用于计算机系统的设计和功能实现，其中最常见的应用是编译器，即一种用来读取其他程序并将之转化为计算机能识别的命令的程序。一个编译器必须能接受程序员输入的全部指令并理解它们的意义，对这样的工作来说，FSA 再适合不过了。FSA 被广泛应用于各种计算领域，并在本章中展示的基础状态图模型上衍生出了各种更复杂的模型（例如，允许从一个状态引申出去的两个箭头拥有同一个标记，或以某种动作关联一个状态）。

The automata we have been looking at here are widely used in the design and implementation of computer systems. The most common application is in compilers, which are programs that read another program and convert it into something that the computer can understand. A compiler must take all the instructions that the programmer has typed in, and work out what their meaning is, so an FSA is very useful for this. There are many other applications of FSAs in computing, and there are many variations on the basic structure that we used here (for example, allowing two arrows out of a state to have the same label, or having an action associated with a state).

举一个简单的例子，你可以用 FAS 来描述一个接口。准备一只电子表，在上面的按钮上标上 A、B、C，等等。假设这只手表正显示时间，画一个名叫“看时间”的状态。假设按钮 A 能设置手表是否显示秒表，那么你可以从 A 画一个箭头指向“秒表”状态来描述这一功能。依此类推，多描述几个按钮的功能，你不仅能更深入地了解这块表的各个接口，也能了解到看似简单的结构背后却拥有复杂的程序！下图是描述一块简单电子表各功能接口的初级 FSA，不难想象，当你遇到更复杂的接口后会多么抓狂了！



For example, you could draw an FSA to describe an interface. As a simple exercise, take a simple digital wrist watch, and label the buttons on it A, B, C etc. Suppose the watch is showing the time. Draw a state that is called “Watch”. Now suppose button A changes the wristwatch to showing a stopwatch. You can represent this by drawing an arrow labeled A from the time state to a “Stopwatch” state. Keep doing this for more sequences of button presses, and not only will you have a better understanding of the interface, but you'll appreciate how complicated it is! The following diagram is just the beginning of the FSA that represents a cheap digital watch. It's no wonder that people get confused by more complicated interfaces!

有趣的事
Curiosity

世界上最庞大的、无数人每天都会用到的 FSA 是什么？那就是万维网(World-Wide Web)。每个网页就好比一座岛屿，页面上的链接就好比行驶在岛屿之间的船只。就像我们在第 11 章结尾提到的那样，在 2000 年世界上一共有数千万个网页，但是 2008 年 Google 却发现了万亿个不同的网页地址。这是个大到恐怖的数字，如果一本书有一千万页，那么它将有 50 km 厚；而一本拥有一万亿页的书，它的厚度将超过地球的周长。

What's the biggest FSA in the world, one that lots of people use every day? It's the World-Wide Web. Each web page is like an island, and the links on that page are the ships that sail between islands. As we said at the end of Topic 11, back in the year 2000 the web had a billion pages. In 2008 Google declared they had found a trillion different web page addresses. That's a lot. A book with a billion pages would be 50 km thick. With a trillion pages, its thickness would be more than the circumference of the earth.

但是网络也仅仅只是一个有限状态机而已。为了给人们提供索引，像是 Google 这样的搜索引擎公司必须测试全部网页以查看它们包含了哪些词语。像你在游戏中做的一样，他们沿着每条链接探索整个网络。正因为网络被称为“网”(web)，因此网页搜索也被称作“爬虫”(crawling)——就像蜘蛛侠做的事儿一样。

But the web is just a finite-state machine. And in order to produce an index for you to use, search engine companies like Google have to examine all the pages to see what words they contain. They explore the web by following all the links, just as you did in this exercise. Only, because it's called the "web", exploring is called "crawling" – like spiders do.

PDG

教师手册

附录

教师手册使用说明

我们在本书的序言部分已经详细阐述了采用“不插电教学法”来进行信息技术教学的好处，该方法利用小游戏而非计算机本身来教授计算机科学的相关知识，而教师手册部分将帮助你更好地将原书中的内容和课堂教学联系起来，并提供每章的答案和提示。

教师手册中将会指出各种小游戏适用的课程类型。作为信息技术课的有益补充，许多游戏对于拓展学生的数学能力和问题解决能力也大有裨益。涉及数学能力的内容包括二进制数、使用编码来表示字母和图像、数据的搜索和排序、构建网络等。而书中其他章节则较好地对应了信息技术课或其他计算机入门课。更多的课程链接将在稍后的“能力链接”部分中谈到。

在所有的小游戏中，学生们的交流能力、问题解决能力、创新能力以及思维能力以有效的途径一一得以训练。这些能力不仅在他们接受普通义务教育阶段十分重要，尤其在今后的职业发展中更为有价值，那些致力于开发计算机前沿新技术的公司非常看重能掌握这些技巧的新人。

资源

除了本书附带的光盘，我们还在网站 http://csunplugged.org 及 http://www.hustp.com/forward/toBookPubic.do 上提供了示范课程录像。许多老师表示，在教授小游戏前观看这些录像对教学活动很有启发。

教学方式

本书的教学方式旨在鼓励学生们开发自身的创新能力和问题解决能力。书中安排的大量问题不是用来测验的——它们强调过程而非单纯的结论。这些问题是学习过程的一部分，而非对学生们的考试。应该鼓励学生们靠自己的力量来回答这些问题，或者通过和朋友的讨论得出答案。有些题目并没有唯一正确的答案，有些答

案只是部分正确。教师手册包括的内容将帮助你指导学生们寻找答案。

a 每道题目的格式类似现在这个段落。无论是进行集体游戏还是面向单独的学生，我们都推荐你在教学过程中检查学生们解题的思路。如果面向的是一组学生，试着鼓励他们互相之间讨论各自的答案。

本书涉及的一些概念看起来颇为复杂，但自己找到理解的方法其实并不难，而且这远比单纯阅读概念性的定义要有意义得多。尽管有些知识并不是使用计算机所必需，但是与此相关的解决问题的能力和创新能力却能使学生们在将来更好地接受新知识和拓展思路。

本书中的小游戏主要为已掌握基础数学技能（加法和乘法）并能进行简单的公式计算（比如 n^2）的高中生设计，如果掌握了指数（2^n）和对数（$\log_2 n$）的知识则能更好理解相关的内容（但这并不是必需条件）。尽管本书的语言是特别按照高中生的习惯来润色的，但是大部分游戏适用于各年龄段的学生[11]。

本书可用于大班或小班授课的课内教学或课外教学，或非学术活动（例如青少年俱乐部）。书中包括了大量问题练习，这让学生们有机会运用他们的算数能力和数学能力，对于数学课堂，本书是一个不错的补充，同时也能作为准备竞赛活动的参考用书，例如奥赛。

每个小游戏只需要事先做少量的准备活动或准备一些小道具，因此只要你有时间，这些游戏都能随时进行。有时你可能需要准备一些可以分发的卡片或其他便宜易得的材料，但从来无须准备昂贵的器材。

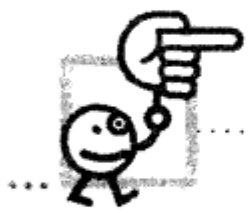
大部分游戏均能在 1 小时之内完成（一堂课的时间），如果时间有限，则在 15 分钟的游戏时间内一般就能涵盖到这个游戏的主要思想。

教师手册中会提供全部题目的解题思路或答案，每道题目的字母编号与其解答的字母编号对应。

每章都有一个“进阶篇”小节，它介绍了一些非理解本书所必须的技术细节，但可能是一些知识储备更丰富的学生想深入学习和感兴趣的内容。每章最后的“有趣的事情”小节旨在用轻松有趣的小故事来拓展本章涉及到的知识点。

正规的术语在本书中均以粗体字表示。几乎每章都带有“术语一点通”知识框，里面用通俗的语言解释了一些计算机中用到的专业技术词汇，比如比特、字节、内存，等等。

[11] 编者注：UNPLUGGED 项目适合的年龄层为 7～17 岁。



能力链接

"不插电的计算机科学"项目中的小游戏能锻炼学生们的各项信息技术技能、数学能力、解决问题的能力以及语言和交流能力。在这里，我们将列举出书中涵盖的重点能力和课程领域，以及涉及这些知识点的章节。大部分小游戏学生们都能独立完成，你也可以通过下面的指南来灵活选择在课堂教授相应知识点时所需的补充材料。

Arithmetic 算数

参见"计数和算数"。在第 6 章"错误检测"中，"ISBN 游戏"提供大量机会来锻炼算数能力（如加法、乘法和减法）。

Binary numbers 二进制数

- 第 1 章：二进制数和"比特"
- 第 2 章：从小比特到大数字
- 第 3 章：从比特到字母

Comparing numbers 数值比较

- 第 7 章：信息
- 第 9 章：搜索
- 第 10 章：排序
- 第 11 章：让排序来得更迅猛吧
- 第 12 章：并行排序（这里有一个超有趣的数字比较游戏，它可以用来比较大数字、小数、分数等多种数据）

Computer storage 计算机储存系统

- 第 1 章：二进制数和"比特"
- 第 2 章：从小比特到大数字
- 第 3 章：从比特到字母
- 第 4 章：从比特到图像
- 第 5 章：压缩信息
- 第 6 章：检测错误

Copying written text 文本复制

- 第 5 章：压缩信息

Counting and arithmetic 计数和算术

- 第 1 章：二进制数和"比特"
- 第 2 章：从小比特到大数字
- 第 4 章：从比特到图像

- 第 5 章：压缩信息
- 第 6 章：检测错误(ISBN 游戏提供了大量算术练习)
- 第 7 章：信息
- 第 9 章：搜索(哈希值的计算中有数位的加法运算)
- 第 10 章：排序
- 第 11 章：让排序来得更迅猛吧
- 第 12 章：并行排序

Developing algorithms 算法开发

很多章节(特别是第 9～15 章)都涉及到算法开发的问题,但以下算法需要特别掌握。

- 第 1 章：二进制数和“比特”(二进制数递增的算法)
- 第 12 章：并行排序 (三路并行排序网络)
- 第 13 章：网络(生成树算法)
- 第 14 章：路由和死锁 (路由算法)

Dictionary order 字典顺序

在有关排序的章节(第 10～12 章)中,排序游戏使用的数字可用单词替代,这样一来能促使学生们用字典顺序进行排序。这种拓展特别适用于第 12 章“并行排序”。

Following instructions 执行指令

许多章节要求学生们按照指令说明来操作,以下章节尤其注重这个要求。

- 第 8 章：程序设计
- 第 15 章：处理输入

Formulating questions and instructions 问题或指令的形式化建模

- 第 7 章：信息
- 第 8 章：程序设计

Logical reasoning and deduction 逻辑推理和演绎

- 第 1 章：二进制数和“比特”
- 第 6 章：检测错误
- 第 7 章：信息
- 第 9 章：搜索
- 第 12 章：并行排序
- 第 14 章：路由和死锁
- 第 15 章：处理输入

Map reading 读图

- 第 15 章：处理输入

Matching 匹配

- 第 1 章：二进制数和“比特”(多种方式来表示比特位)
- 第 5 章：压缩信息(找出重复的文本)

Number bases 数制

- 第 1 章：二进制数和“比特”
- 第 2 章：从小比特到大数字(特别是“进阶篇”单元)

Number series 数字序列

- 第 2 章：从小比特到大数字(序列 1+2+4+8+…)
- 第 10 章：排序(序列 1+2+3+4+…)

Odd and even numbers 奇数和偶数

- 第 6 章：检测错误

Permutations and factorials 排列和阶乘

- 第 12 章：并行排序(“有趣的事”单元)

Probability 概率

- 第 7 章：信息
- 第 8 章：程序设计

Problem solving 问题求解

尽管所有章节都能培养学生们解决问题的能力,但是下列章节重点着墨于解决一个既定的问题。

- 第 13 章：网络
- 第 14 章：路由和死锁

Programming 程序设计

涉及算法的第 8 章到第 15 章都与程序设计相关。第 15 章还介绍了一个非常简单的编程系统:FSA。

- 第 8 章：程序设计(全章介绍的都是关于编程语言的问题)

Question asking 提问技巧

- 第 7 章：信息

Sequencing 序列

- 第 1 章：二进制数和“比特”

Teamwork 小组合作

以下章节的游戏尤其适合于演练团队合作模式。

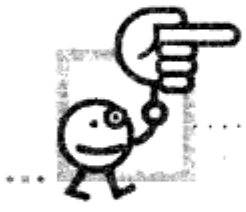
- 第 12 章：并行排序
- 第 14 章：路由和死锁

Specific computer concepts 专业计算机概念

以下列出了书中每章涵盖的专业计算机术语,这些术语在书中都已用黑体字标出。

- Algorithm (算法)：Topic 9，10，11，13，14
- ASCII Code (ASCII 码)：Topic 3
- Binary Numbers (二进制数)：Topic 1，2，3，7
- Binary Search (二分搜索)：Topic 9
- Bit (比特)：Topic 1，2，3
- Bubble Sort (冒泡排序)：Topic 10
- Buffer (缓冲)：Topic 14

- Parity (奇偶校验): Topic 6
- Pixels (像素): Topic 4, 5, 7, 12
- Programming Language (程序设计语言): Topic 8
- Quicksort (快速排序): Topic 11
- RAID(独立冗余磁盘阵列): Topic 6
- RAM (随机存取存储器): Topic 2, 11
- RAR Files (RAR 文件): Topic 5
- Reed－Solomon Code (里德所罗门码): Topic 6
- Routing (路由): Topic 13, 14
- Search Key (搜索关键词): Topic 9
- Searching Algorithms (搜索算法)(linear,binary, hash search 线性,二分法,哈希搜索): Topic 9
- Selection Sort (选择排序): Topic 10
- Sorting Algorithms (排序算法): Topic 10, 11
- Sorting Network (排序网络): Topic 12
- State (状态): Topic 15
- Traveling Salesperson Problem (旅行商问题): Topic 13
- Unicode(统一码): Topic 3
- Word Processors (文字处理器): Topic 3
- Zip Files (Zip 文件): Topic 5



问题答案

第1章　二进制数和“比特”

■ 小游戏 1.1：二进制卡片

a 每张卡片都比左边的一张多 2 个圆点。

b 16 的 2 倍，即 32。

c 32 的 2 倍，即 64。

d 将圆点数量乘以 2。

e 为了得到 6，你需将 4 个圆点和 2 个圆点的卡片翻到正面朝上。

f 为了得到 20，你需将 16 个圆点和 4 个圆点的卡片翻到正面朝上。

g 为了得到 15，你需将 8 个、4 个、2 个和 1 个圆点的卡片翻到正面朝上。

h 为了得到 21，你需将 16 个、4 个和 1 个圆点的卡片翻到正面朝上。

i 为了得到 30，你需将 16 个、8 个、4 个和 2 个圆点的卡片翻到正面朝上。

j 错，每个数字只有一种表示方法。

k 能组成的最大数字是 16+8+4+2+1=31。另一个得到 31 的方法是看下一张卡片是否标着 32 个圆点。

l 最小的数字为 0(不是 1!)。

m 没有，0～31 之中的所有数字都能表示得出来。

n 一个简单的方法是，从右到左翻动每张卡片直到将卡片 1 翻到正面朝上。比如，如果卡片 1 是正面向下的，你需将它放过来正面向上，然后停止动作。如果卡片 2 是正面向下，卡片 1 正面是向上的话，你需要先翻动卡片 1，然后翻动卡片 2，一旦卡片 2 正面向上了便停止动作。试试看按照这种方法将 15 变成 16。

o 5 比特，每张卡片代表一个比特。

p 表示 01101 时，16 个圆点的卡片正面向下，8 个和 4 个圆点的卡片正面向上，2 个圆点的卡片正面向下，1 个圆点的卡片正面向上，这样一来便能代表十进制数 8 + 4 + 1 = 13。

q 00110 = 4 +2 = 6

r 01110 = 8 + 4 + 2 = 14

s 10001 = 16 + 1 = 17

■ 小游戏 1.2:短短的二进制数和长长的二进制数

t 用来表示 9 的 6 位二进制数是 001001。只用在数字前加 0 满足位数要求即可,无论加多少个 0 在前面,表示的都是 9。

u 二进制数 111111,即 32+16+8+4+2+1=63。

v

01001 = 9	1101 = 13
101 = 5	10001 = 17
00000 = 0	10100 = 20
10 = 2	11111 = 31
0 = 0	100010 = 34
1010 = 10	110101 = 53

第2章　从小比特到大数字

a 二进制数 101001，即十进制的 41。

b 二进制数 101101，即十进制的 45。

■ 小游戏 2.1：二进制数的性质

c 1+2 = 3

d 1 + 2 + 4 = 7

e 1 + 2 + 4 + 8 = 15

f 1+2+4+8+16 = 31

g 前面所有卡片的数字之和总是比下一张卡片的数值少 1，比如上一个问题的答案是 31，正好比下一张 32 点的卡片少 1。另外一个方法是，将最大的卡片(16)乘以 2，然后减去 1。

h 010，即十进制的 2，变成 0100 后，即十进制的 4。

i 101，即十进制的 5，变成 1010 后，即十进制的 10。

j 110，即十进制的 6，变成 1100 后，即十进制的 12。

k 当插入一个 0 到最右侧后，二进制数的数值变成原始值乘以 2。

■ 小游戏 2.2：大一点的二进制数

l 最大的卡片是 32，因此应比 64 小 1，即 63。如果要计算的话，为 32+16+8+4+2+1=63。

m 127（比 128 小 1；或 1+2+4+8+16+32+64）

n 255（比 256 小 1；或 1+2+4+8+16+32+64+128）

o 65535。计算第 17 张卡片(比特)的数值并减去 1，便能算出这道题目的答案。第 17 张卡片的数值可以通过将前 16 张卡片的数值反复乘以 2 得到，即从右侧的卡片 1 开始，每移向左侧一张卡片就将数值再乘以 2。用计算器可能会算得快些，不过一个更快的方法是用 2^{16} 表示第 17 张卡片的值，即 65536。而前 16 张卡片的总和应比这个数小 1。

p 16777215。用上题中的快捷方法，结果即为 $2^{24}-1$。请注意，24 位二进制数一般被用来表示计算机中的色彩，它们能表示出超过 16000000 种不同颜色，这么多颜色甚至连人眼都分辨不出来。因此在计算机显示设定中，24 位色彩往往被称为“真彩色”或“兆色彩”。

q 10100101(请注意,无论对这个数还是其他二进制数,在其前面加任意个零的答案都是正确的,所以010100101,或者 00000000101001011100011 都是正确答案,正如 165、0165 和 00000165 代表相同数值。通常我们不用担心开头的零,但有时它们被放置在前端是专门为了表达可用的数位,比如,乐透彩票号码 0165 表明一共可有 9999 张彩票,而二进制数 0000000010100101 表明数字将保存为 16 位。)

r 1100011

s 1111111

t 1111101000

u 10000000000

■ 有趣的事

卡片上的数字能通过 5 位二进制数来算出。第一张卡片上写的是第一位为 1 的所有 5 位二进制数(10000、10001、10010、10011 等),第二张卡片包含了第二位为 1 的所有 5 位二进制数,依此类推。最后一张卡片包含了末位为 1 的全部二进制数,它们均为奇数。

第3章　从比特到字母

a 在拼音表上字母 d 用十进制数 8 来表示。

b 字母 y 的编码是 29。

c 30 代表 z。

d “hello”可表示为 12 9 16 16 19。

e “17 5 3”代表“mǎ”

f 此题答案因人而异。

g 16 位数可表示 $2^{16} = 65536$ 个不同字符。

■ 小游戏 3.1:储藏室谜题

h 消息为“救命被困”(“Help, I'm trapped”)。

■ 小游戏 3.2:制作属于你自己的信息

此题答案不受限定。大家可以通过发送其他符号来代表 0 和 1,甚至可以自己创作只有两种形状的个性化符号。另一个例子在本章的最后,即用手链代表信息。

■ 小游戏 3.3:传音游戏

一开始,不妨将在小游戏 3.2 中写下的信息发给另一个不同的伙伴。

i 0111 表示为一个低音紧跟着三个高音。

j 9 为二进制的 1001,即一个高音、两个低音、再一个高音。如果你使用的是 5 位二进制,你可以用 01001 表示,在最前面加一个低音就可以了。

k 在拼音表中数字 21 代表字母 q,也就是二进制的 10101,表示为 5 个音:高、低、高、低、高。

■ 有趣的事

第一条手链的二进制编码为 10000 01101 10000 01101 00100,主人的名字叫 LILI4(LiLi)。

第二条手链的二进制编码为 00001 01110 01110 00001,主人的名字叫 ANNA。

第4章　从比特到图像

a 答案不固定。可供选择的答案包括：照相机中储存的照片、电影的片段、游戏中的动画、网页上的图像，甚至包括计算机中的图标和字母也算是一种体积较小的图像。而一些由计算机芯片控制的设备，比如手机中的图片信息也是正确的答案。

b 第 3 行：0，1，1，1，1

c 第 5 行：1，0，0，0，1

d 最后一行：0，1，1，1，1

e 第 3 行：1，4

f 第 5 行：0，1，3，1

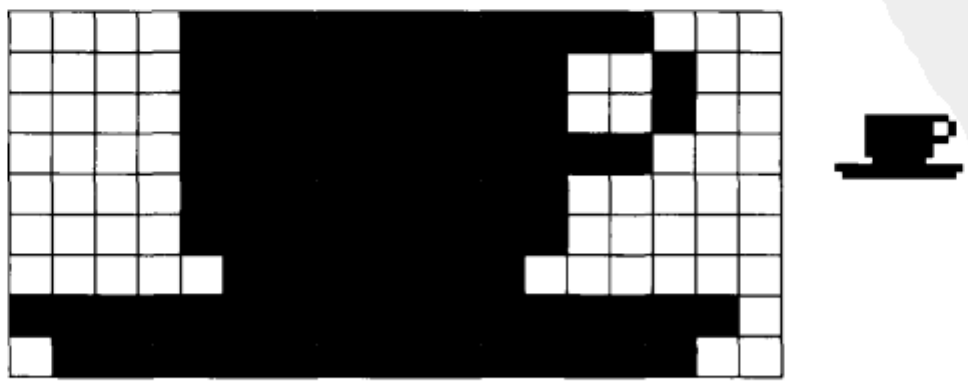
g 最后一行：1，4

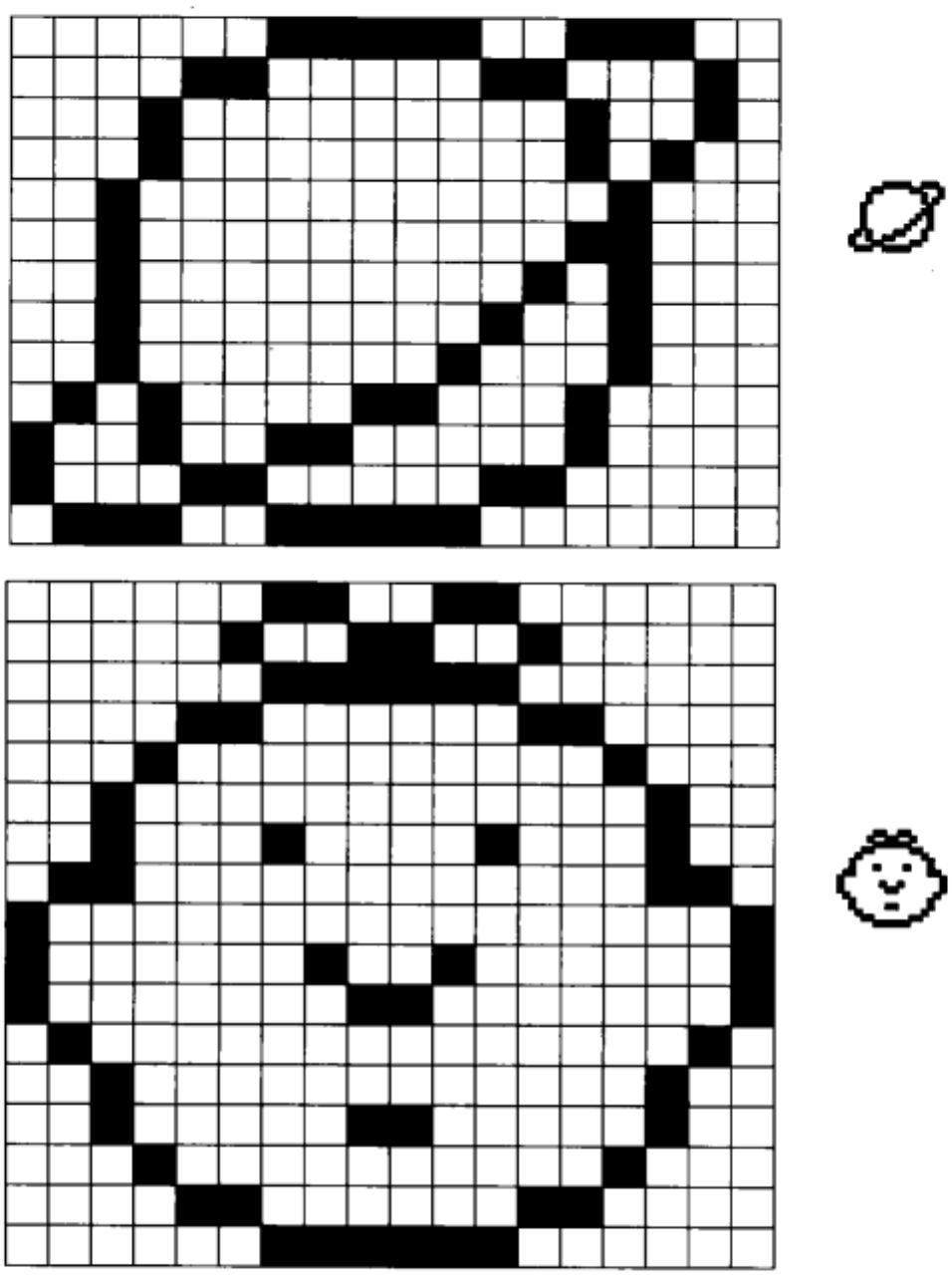
h 你需要用到两个和为 36 的数字，中间夹 0 个白色像素。比如：31,0,5(31 个黑色像素，接着是 0 个白色像素，再接着 5 个黑色像素)或 5,0,31 或 6，0，30……可供选择的组合很多。你或许注意到，对于很短的像素段来说，我们也能用同样的方法来表示。比如，当我们写出(1,2,0,1,1)时，能顺利解码为第一行(1,3,1)，但是这样没什么意义。事实上，如果一个压缩系统拥有不止一种编码方式的话，意味着它并不是高效的，即还有改进的空间。然而，也有为了保证系统的简单而保留使用低效率编码的时候。

i 一张传真页面大概为 1000×2000 pixels，每两行字之间是 1000 pixels 的白色的像素带，顶部和底部的边缘也同样空出这么多。黑色像素带往往就短多了，因为它们一般是页面上的文字，不过页面上的划线或图案也会形成较长的黑色像素带。

j 最简单储存彩色像素的方式是，将每个颜色用相应的数字编码，然后储存每个像素对应的数字(颇类似“用数字画画”!)。由于一般都有数以千计不同的色彩，而你需要储存屏幕上每个像素对应的数字，这样一来便会占用大量空间。对于彩色图像来说，游程编码并不适用，因为相邻的两个像素一般并不是完全相同的颜色。对于照片而言，计算机采用的是另外的解决方案(通常使用的是 JPEG 格式)来减少所需的储存空间。

■ 小游戏 4.1：图像解码





■ 小游戏 4.2：图像编码

如果正确执行了这个游戏，那么第二名学生应该能解码第一名学生画出的图案。当然，任何人都有可能犯错，如果图案的一部分有错误，应该试着自行找出问题。

第5章　压缩信息

■ 小游戏 5.1:文字的解压缩

这首童诗解码后为:

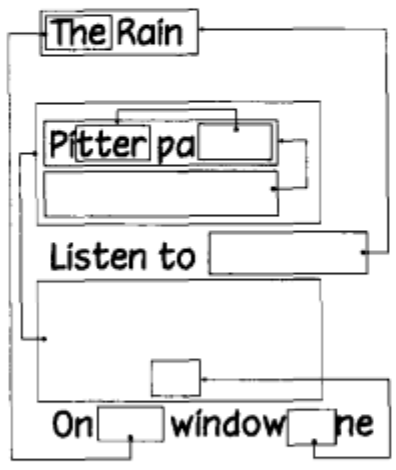
Pease porridge hot,
Pease porridge cold,
Pease porridge in the pot,
Nine days old.

Some like it hot,
Some like it cold,
Some like it in the pot,
Nine days old.

■ 小游戏 5.2:文字的压缩

a 第一行:"tter"重复
第二行:整行重复
第三行:"the rain"和标题重复
第四行:整行重复
第五行:整行重复
第六行:"the"和"pa"重复

b 诗应该看起来像下面这样:



c 原诗第一行(标题):8

第二行:13
第三行:13
第四行:18
第五行:13
第六行:13
第七行:18
总计:96 个字母和空格
(请注意:计算机科学家们往往视每行末尾的转行符也为一个字符,所以总数应再加 7。这里我们可以先不用考虑。)

d 这道题有点难度(请不要忘记数空格)。答案应如下。
压缩后的诗的第一行:8
第二行:9
第三行:0
第四行:0
第五行:0
第六行:0
第七行:12
总计:39

e 压缩后的诗体积为原诗体积的 29/96(41%),即压缩了体积的一半以上,或者你也可以称它节省了 96－39 ＝ 57 个字母。无论怎么说,都大大缩小了体积。尽管这里得到的 41%的压缩率差不多等于这种压缩法对普通文本的压缩率,不过,如果你真的要测试压缩系统的效能,不妨多拿几个例子进行研究。

f 6 个箭头

g 6 个箭头×2 字符每箭头＝12 字符

h 39＋12 ＝ 51

i 压缩后的诗体积为原诗的 51/96,比一半略多一点。或者你可以说减少了 96－51＝45 个字母和空格。

j 这种方法也适用于图像中不包括大范围色彩的情况,其实是这正是 GIF 和 PNG 格式使用的方法。箭头指向图像之前出现过的相同图案,大部分情况下重复的图案类似“52 个白色像素接 8 个黑色像素接 9 个白色像素”这样简单的模式。但是对于照片来说,这种方法不太管用,因为照片中用到的色彩范围太大,而且许多细节部分有变化,不太可能有大量重复的像素序列。但是对于简单的图像,比如扫描页和图标,这种方法还是相当好用的。

第6章　检测错误

■　小游戏 6.1:翻卡魔术

a 每行和每列的白色卡片数量均为偶数(比如,从上向下每行白色卡片的数量分别为 2、6、2、2、4、4)。额外的那张卡片称为奇偶校验卡,它们用来保证每行和每列都拥有偶数张正面向上的卡片。比如,第一行的奇偶校验卡为背面向上,而该行有 2 张卡片正面向上;第二行的奇偶校验卡正面向上,因此该行有 6 张正面向上的卡片;第一列的奇偶校验卡为背面向上,因此有 2 张卡片正面向上;第二列奇偶校验卡为正面向上,因此有 4 张卡片正面向上。

b 每行包括的卡片数分别为 2、4、4、0、3、4。

c 第五行现在有奇数张卡片(3 张)。

d 每列包括的卡片数分别为 4、2、2、3、2、4。

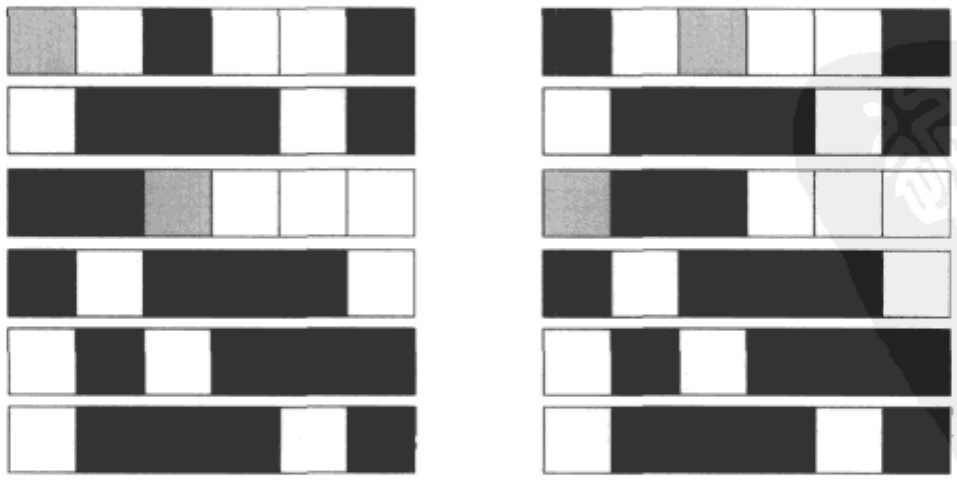
e 第四列现在有奇数张卡片(3 张)。

f 同时位于错误的行和列的只有唯一一张卡片,即位于第 4 列和第 5 行相交处的黑色卡片。

g 行 6 列 3。

■　小游戏 6.2:发现更多的错误

h 第一行和第三行出错了,第一列和第三列同样也出错了(因为它们都拥有奇数张卡片)。我们不能确定到底是哪两张卡片被翻过来,因为可能性有两种,见下图中表示为灰色的卡片。



i 如果同时发生两处错误(如上),你便无法确定哪个是出错的比特。也就是说我们能检查出错误,但无法纠错。你必须让发送人重新再发送一次消息。

j 不能,除非你将同一张卡片翻动两次,但这就根本不会造成问题了! 如果两张被翻动的卡片位于同一行,那么它们所在的列将会显示出错误;如果它们位于同一列,那么它们所在的行将会显示出错误;如果它们位

于不同行和列，我们会得到上面提到的结果，这样一来我们就无法确定出错的准确位置了。

k 不能，因为你总会得到至少一行或一列存在奇数个错误，总是会被检查出来。

l 可以，因为四列全部都会出错(尽管所在行可能有正确的奇偶对)。

m 可以，只需令四张卡片形成一个长方形。即挑选两行和两列，翻转位于行和列的四个交叉处的四张卡片，则每行和每列都能保持正确的奇偶对，因为每行每列均有两张卡片被翻动。

n 第四列如下：

■ 小游戏 6.3：ISBN 检测

o 是的，最后一位为 4。

p 校验号应为 7，和 4 不匹配，因此系统检查出错误。

q 比如，校验号为 1，则检查出错误。事实上，根据校验公式，当相邻的两位数字被颠倒，系统总能检查出错误。

r 改变任何一位数字都会导致校验和出错。如果单个数位发生错误，系统总能检查出来。

s 1-00-235045 的校验和为：1×10+0×9+0×8+2×7+3×6+5×5+0×4+4×3+5×2=89；89/11=8 余数为 1；11−1 = 10，于是校验和为 X，正确。

t 可以。新的校验和为 0，可这里却是 4。

u 不能！这里校验和为 4，校验结论为正确。由于两个被颠倒的数字相差 5(比如 2 和 7，3 和 8)，因此导致看起来没错，而对于总能检查出颠倒数位错误的 ISBN-10 方法来说，这个错误却能被检查出来。ISBN-13 方法也能检查出任何数位变化，和 ISBN-10 一样。

v 不能，因为两个数字虽然现在不在正确位置上，可是它们却都被乘以同一个值。然而，如果你只交换两个数位，则会影响校验和。

第7章　信　息

a 本题答案不定。信息可以是任何有用或让人有兴趣知道的事情。可能的答案包括：百科全书的内容、字典、电话本或地图；也可以是别人昨天晚餐吃了什么、一个东西的价格、学校位于的方位或者玩游戏的规则，等等。

b 本题答案不定。可能的答案包括：数数一本书的页数、一共包括多少个句子或多少个字。其他可能的答案可能是：在一本书里你认为有趣的地方有多少个？你会花多长时间来读这本书？

c 从信息含量最多的到最少的依次为：

- 1000 页的电话簿
- 500 页的电话簿
- 1000 页写满“废话 废话 废话”的纸
- 1000 页空白纸

d 本题答案不定。最佳的答案是猜出这条信息的困难度，从技术的角度而言，即为这条信息出现的概率。

■ 小游戏 7.1：Yes / No 问题

e 如果使用的是“二分搜索”（即每次将搜索范围缩小一半），则最多猜 7 次。否则，如果采用了糟糕的方法，或许猜一百次也猜不出来。

f 平均来说，你需要猜测的次数为搜索对象总数的一半（50 次），尽管有时候你或许很幸运一下子就猜中了。

g 在 e 题中提到的二分搜索法是最佳的方案，即每次将搜索范围缩小一半。比如，如果你第一次提出问题“这个数字是否小于 50”，得到了“不是”的回复，接着你可以将搜索范围缩小至 50～100。然后你可以通过问“这个数字是否小于 75”再将这个范围缩小一半，得到的答案将让你的备选数字缩小到 25 个。接下来，范围会依次被缩小至 13、7、4、2 和 1 个备选数字。所以将 100 个可能值锁定到最终目标只用对半“切”7 次。

h 本题答案不定，但是大家能很快猜出来，因为这些数比随机的数更容易估计。

i 如果采用了二分搜索的猜测法，只需要猜 14 次。

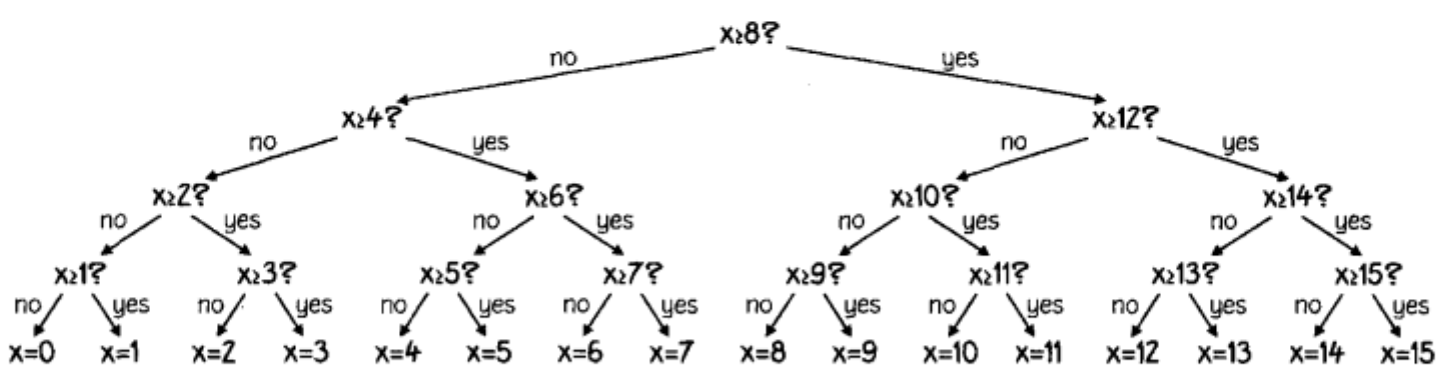
■ 小游戏 7.2：决策树

j Yes，No，Yes

k 三次。决策树的每层进行一次决策。

l 决策树某一分支上 Yes/No 组成的 1/0 序列正好是分支末端数字的二进制表示（参见第 1 章）。比如，对于 x=6 的答案是“yes、yes、no”，翻译成 110，即 6 对应的 3 位二进制数。

m 画出的决策树应该和书中给出的决策树类似，不过顶端问题应为“x≥8?”，左侧分支与得到数字 0～7 的树一样；右侧的分支是将书中决策树的每个数字加 8 即可。



■ 小游戏 7.3：丢失的文字

n 一般会根据“computer”这个单词来猜“u”，但也可能是“competer”，不过这个单词太少见了。当然，你可以说任何字母都能在这里出现，比如“compbter”——然而这个单词根本不存在，没有任何意义，但你不能阻止其他人将它就这样输入计算机！

o 本题有多个可选答案。大家或许会倾向于找那些首字母为“t”的单词（好比 talk（说话）、trick（诡计）、trade（贸易）、tell（告诉）、tickle（逗）或 take（拿））。紧跟“t”之后最常出现的字母为“r” “h” “e” “a” “o” “i” “u”和 “w”。最重要的是，一些字母很少会出现，比如“k”和“q”。

p 有可能，但是不常见。事实上没有很多单词会以“tk”作为开头，但是你至少能在某一本书找到这个词——对，就是你现在读到的这本书的这句话！

q 有些字符比其他的字符容易猜出来。原文如下：

There is no reverse on a motorcycle. A friend of mine found this out rather dramatically the other day.（我的一个朋友有一天戏剧性地发现了摩托车是没有倒挡装置的。）

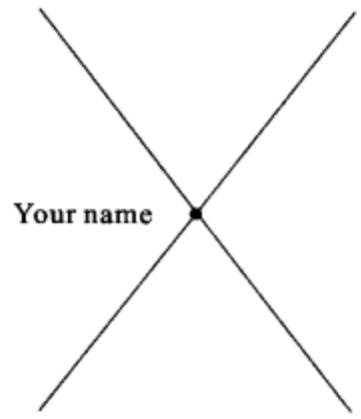
r 这个游戏用英文或拼音（只用到 26 个字母）玩会比较容易，而用中文玩则会比较困难。大家会发现有一些字母很容易被猜出来，甚至一次就中，但一些比较不常用的字母则需要多猜几次。

第8章　程序设计

a 在给别人下指令的时候，通常会忘记注明许多细节，就像“走过那扇门”时忘记说“先开门”……有些时候，指令会欠缺严密，比如“请切开所有的苹果”，可能真正的意思是切开手上拿着的所有苹果，而不是世界上存在的所有苹果。

■　小游戏 8.1：依照指令行事

b 画出的图案应该类似下图。



c 这里并没有规定画出的线的粗细，所以如果线比圆点粗的话，你或许就看不到被盖住的圆点了；写下名字的位置或许也会不同；由于使用的纸张形状也可能会不同，所以线段之间的夹角大小会因为纸张的形状改变而改变，等等。

d 希望大家能画出类似的图案。

e 这两种问题都有可能会发生，在计算机中后者更普遍。

f 当描述者没有在旁边监督画图者的操作，错误是无法被纠正的。对于描述者来说，如果他们看不到画出的图案，是不可能知道已经画错了的。

g 这是最困难的状况，因为描述者和画图者之间只有一种交流方式。不到最后，描述者是无法知道发生的问题的。

h 对于程序来说正确性是相当关键的，一个小小的错误就会导致无法估计的后果。只要一个小错，宇宙飞船或许就会迷失自己的航线，发电厂或许会失控并导致一场灾难，两辆火车可能因此对撞！无论以上哪种情况，都会让人丧失性命，由此可见正确设计的程序是多么重要了！就算是一些无关生死的情况（比如信用卡支付系统），一旦它们失控，也会导致严重的恶果——设想一下如果商店里无法刷卡，购物的人群将会乱成怎样。

第9章 搜 索

■ 小游戏 9.1:线性搜索战舰

a 本题的答案不定。平均来说,大概需要翻动半数的卡片,不过运气好坏会导致少翻几张或多翻几张。

b 猜 25 次。(或者 24 次,因为剩下的那一张一定就是答案了。然而,在搜索中你并不能确定关键词是否存在候选目标中,这样就还是需要检查一下最后那张卡片了。)

c 一张卡片。

d 一次就中,对于 25 张卡片来说,即为 4%的概率。

e 一定需要翻动全部 25 张卡片才可能知道目标不在候选项中。

f 平均来说,你需要查看一半数量的战舰(12.5)。即 1 到 25 的一半,如果猜的话应该为 13 次。

g 大概为战舰总数的一半,或猜 50 次(确切的次数为 50.5 次)。

■ 小游戏 9.2:二分法搜索战舰

h 本题的答案不定,但是对于 25 张卡片来说,猜 5 次就可以了,或许还能更少。

i 本题答案不定,最多需要猜 4～5 次,幸运地话一次就能猜中。

j 最佳情况是一猜就中。

k 如果正确使用了二分搜索的话,最差的情况需要猜 5 次。如果猜测的次数大于 5 次,说明猜的人并没有用最佳的猜测策略:即选择中间的卡片,每次缩小候选数量的一半。

l 需要猜 5 次。

m 需要猜 7 次。

n 最多需要猜 20 次。一般来说,每搜索 k 次可以覆盖的目标数量为 2^k,因此搜索 20 次我们的搜索能覆盖多达 1048576 个对象。

■ 小游戏 9.3:哈希法搜索战舰

o 本题答案不定,可以为 1～25 之间的任何次(如果有 25 张卡片的话)。如果碰巧所有卡片都拥有同一个哈希和,会需要猜 25 次。

p 最佳情况是只用猜一次。

q 最糟糕的情况是,所有战舰被归到同一类中。在这种情况下,游戏变成线性搜索,不过这种可能性很低。

r 你需要在类中寻找全部的战舰。每一类中应该只有 2～3 艘战舰,所以搜索起来应该很快,不过也有可能有一类中包含更多艘战舰的情况。

s 如果是 25 艘战舰分为 10 个类，即为每一类有 2.5 艘战舰。一般来说 1～2 次猜测便能找到目标。

t 放在战舰数量最多的类中。

u 放在战舰数量最少的类中。

v 线性搜索唯一的优点是，它不需要战舰被提前归类。然而这种方法很慢，不实用。一般来说，只要能将关键词尽可能地分散到不同类中去，哈希搜索法的速度最快；倘若担心过多的对象被分配给同一类的话（这种情况下哈希搜索非常低效），二分搜索法会更好。二分搜索的优点也在于容易找到目标数字所在范围。例如，如果你需要找 3400～3409 间的某个数字，你只用搜索 3400，其他数字是紧跟其后的。相反，如果使用哈希法（就像我们在游戏中用到的），那么每个属于3400～3409 区间的数字会被归入不同类，反而需要一个一个搜索。

第10章 排 序

a 在字典和电话簿中对字母排序十分重要，如果都是无序状态，要找到某个词语或人名将异常困难。按照数字大小排序同样也很重要，比如给学生的成绩排序看看谁的成绩最好、谁需要更加努力。就连网页搜索的结果也是排过序的，这样一来你便能在最前端看到最符合你搜索要求的结果了。

■ 小游戏 10.1：选择排序

b 最简单的方式是比较两个被测物，不动最轻的重物，将另一个放在一边，之后持续将其他重物与当前最轻的进行比较，直到完成全部重物的比较。比如，如果你有 4 个不同重量的重物（你不知道重物的重量各为多少，假设它们分别为 3、2、4、1），然后你可以比较 3 和 2，将 2 视为“当前最轻的”。然后比较 2 和 4，保留 2，再对比 2 和 1，并将 1 选为当前最轻的。

c 需要比较的次数为对象的总数减去 1，即对于 8 个重物来说需比较 7 次。

d 99 次。

e 2 次。

f 公式为 $c = n - 1$（c 题已经提到）。比如，如果 $n=1$，那么你一次都不需要比较就能找到最轻的重物。

g 如果操作正确，需要 $7+6+5+4+3+2+1 = 28$ 次比较。

h 总和为 210。一个简便的计算方法是，将数字重组起来变成 $(20+1) + (19+2) + (18+3) + \cdots + (11+10)$，即 10 组 21，即 210。

■ 小游戏 10.2：插入排序

i 对于插入排序，本题答案不定。在最佳的情况下（虽然可能性很低）你只需要比较 7 次（比如，如果你一直都能选中剩下未排序那一堆中最轻的一个，并与已排好序队列中最重的进行比较）；而最糟糕的情况下你需要比较 28 次（比如，你每次都挑中剩下重物中的最重的那个，从而需要将排好序的整个队列过一遍才能将最重的那个放到该队列的最前端）。对于选择排序法来说，答案一样，一般情况下你需要比较 7 到 28 次。

j 这是上一题中描述的最佳情况，即只需比较 7 次。

k 这是刚才描述的最糟情况，需要比较 28 次（$1+2+3+\cdots+7$）。

■ 小游戏 10.3：冒泡排序

l 需要比较的次数不定。在最佳情况下，只需比较 7 次，但这种结果发生的前提是你碰巧遇到一开始就排好序的序列！如果遇到的序列碰巧为倒序，那么第一轮将最重的物体移动到最右侧就需比较 7 次。接着，每移动一个剩下的物体到其正确的位置都需要比较 7 次，所以一共需要比较 $7\times7 = 49$ 次！大家或许在第一轮比较

之后便能发现最右侧的物体已经处于正确的位置了，只有左侧的 7 个物体需要在下一轮移动中被检查位置，接着是左侧的 6 个，依此类推。一般需要 28 次比较（为选择排序和插入排序的最糟情况）：其中需 7 次比较将最重的物体“冒泡”到它该处的位置；需 6 次比较将次重的物体排出来，依此类推。

m 当对象完全乱序时，这三种算法中插入排序耗时最少。当对象一开始是逆序排列时，三种算法将需要比较相同次数来完成排序（你采用的应是之前提到过的改进版冒泡排序法）。当对象序列的初始状态是升序排列时，那么执行插入排序法和冒泡排序法都将非常迅速。

n 对于这些算法而言，给定 21 个对象，在最糟情况下都需要比较 $20+19+18+17+\cdots+1 = 210$ 次——那将是一个很大的工程！我们第一次提到的冒泡排序法比通常的排序法要略微简单一点，利用它需要一共比较 420 次（21 轮，每次 20 次比较）！

第11章　让排序来得更迅猛吧

■ 小游戏 11.1:快速排序

a 答案不确定。幸运的话,采用快速排序法只需要 13～14 次比较。如果运气不佳,那么最多需要比较 28 次。

b 当你选择序列最中间的那个对象为基准时,快速排序法需要的比较次数最少,此时基准将剩余对象分成大小相等的两堆。

■ 小游戏 11.2:归并排序

c 采用归并排序法在这里需要比较 14 次。就算你很不走运,也只用比较 17 次。在最好的情况下,只需要比较 12 次。

d 一般来说,快速排序和归并排序要比选择排序、插入排序或冒泡排序的速度快很多(也就是需比较的次数较少)。

e 7 次。除了最后一个对象不需要和任何对象比较外,其余每个对象都将在整个序列中进行一次比较。

f 4 次。即碰巧其中一个序列中的每个对象都比另一个序列中的任一对象轻,那么只有 4 个较轻的对象需要进行比较,接着剩下的 4 个对象只用直接移过来,不需要再比较了。

第12章　并行排序

请注意，如果玩游戏的人数少于 6 人，这个游戏的形式做成棋盘游戏会比较好。如果参加者很多的话，你可以将他们分成几组，每组 6 人来互相竞赛。这个游戏对任何年龄段的同学来说都很受欢迎，他们会发现，不让任何一名队员落后是非常重要的，因为任何一人落后他们都没有办法再进行比较了。除了采用案例中的数字 1 到 6 外，你也可以用更大的数字，甚至词语都行（最后按字母顺序排列）。

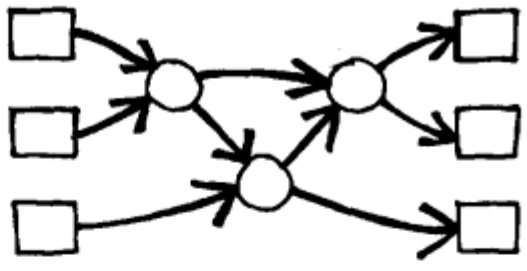
■ 小游戏 12.1：排序网络

a 是的。这个网络是计算机科学家们设计用来将每个输入排好序的。如果输出的结果并不是排好序的，那你在操作中一定出现了错误。

b 一共需要 5 步，图中每列结点处需要一步。（或者换种方式来理解，图中最长的两条路线是数字 3 和数字 4，它们从一头走到另一头一共需要做 5 次比较。）

c 将会把数字从大到小排列起来（而不是从小到大）。

d 下图能对三个数字进行排序。为了保证总能正确的排序，你至少需要 3 个节点。大家不妨用全部 6 种可能的初始输入序列来测试一下设计的网络（1-2-3，1-3-2，2-1-3，2-3-1，3-1-2，3-2-1），最好互相测试一下对方的网络，这样大家会有更高的热情来找出令网络输出错误结果的输入。



e 第二个网络运算速度更快。第一网络要求每次比较都能按顺序来，一个接一个（有一条路径甚至需要比较 6 次）。而第二个网络中有一些比较可以同时进行（通过整个网络的最长路径只需要比较 3 次）。第一个网络是连续处理的例子，而第二个网络使用的是速度更快的并行处理。

f 输出全部输入数字中最小的那个。

g 可以，每新雇用一个人和多用一把铁锹都能在当时多开挖一段排水渠。

h 不能。挖土壕的时候必须从上至下一步一步来挖，如果上端没有挖出来，后端是无法开工的。学生们或许会说增加更多的工人，因为人多工作起来就没那么累，那工人速度便会更快，但是比起能够并行化的任务来说，这样的时间节省几乎可以忽略不计。

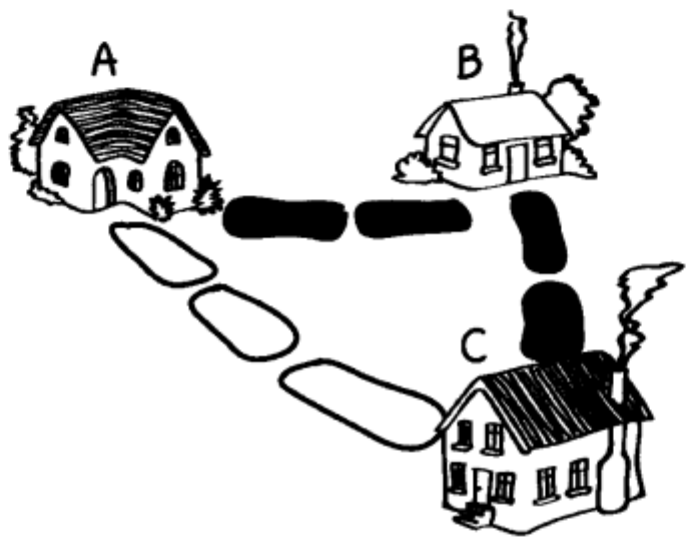
i 增加人手可以分工做不同的菜或同时切不同的原材料以加快做饭的速度，而且你还能同时烹饪不同的食物。但是对于单独的菜来说，你却无法并行处理烘烤或煮的过程（比如用两个炉子），因为弄熟它需要的时间是无法改变的。

j 多件衣服可以同时一起洗涤（并行）。在洗衣服的同时，你还能用烘干机烘干另一堆洗好的衣服（并行）。但是对于同一件衣服来说，你却无法同时洗涤它并烘干它（这两个步骤是连续的）！而且有些衣服也无法同时洗涤，比如你不能将一件新的大红 T 恤和你最爱的白色衬衣洗在一起！

第13章 网 络

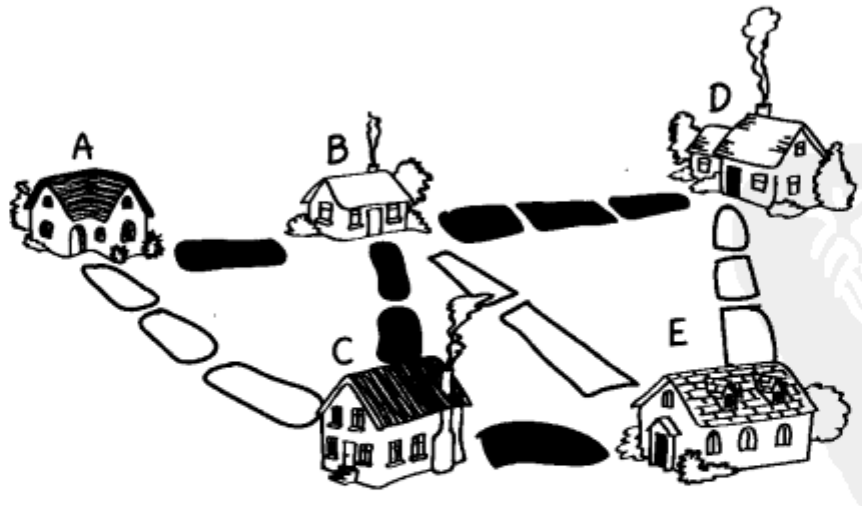
■ 小游戏 13.1:泥泞城市

a 房子 A 和房子 B、房子 B 和房子 C 之间的道路必须被铺设。一共需要 4 块石砖。



b 从房子 A 走到 B 再走到 C,这样便能一路都是走在铺好的路上了。

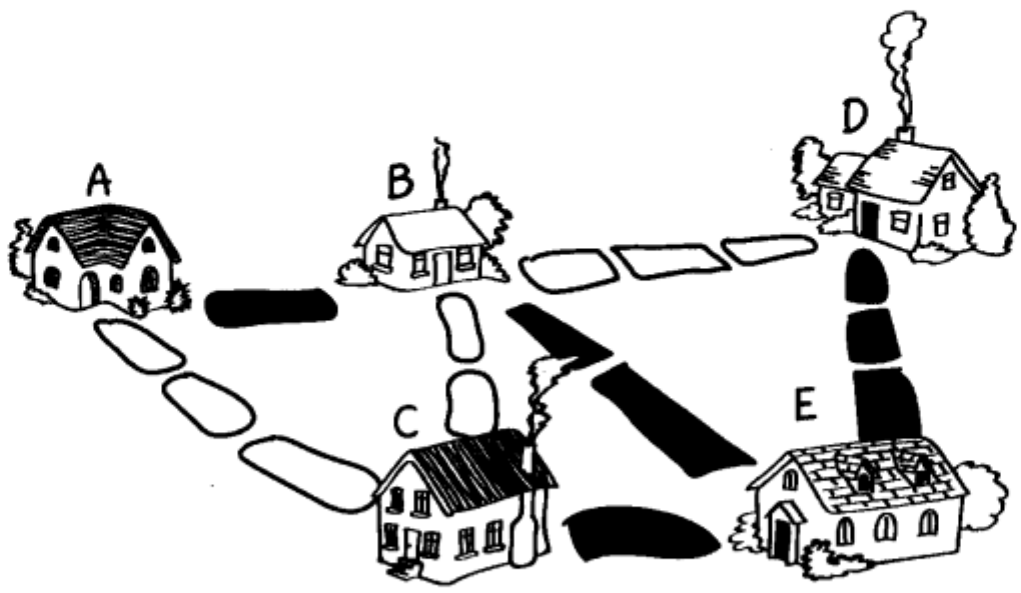
c 下图是正确答案之一,即铺设 7 块石砖。当然其他铺设 7 块石砖的方案也行得通。本题铺设石砖的数量最少为 7 块。



d 要从房子 A 走到 D,必须经过房子 B。

e 只铺设 7 块石砖有很多铺设法,比如下图。任何只用到 7 块石砖并能连结全部房子的方案可视为正确答案。

f 这种情况下,应选择直接从房子 B 到 E 的道路进行铺设,就像 e 题中给出的方案。c 题给出的方案不对,因为从房子 B 到 E 用到了 3 块石砖,而不是 2 块。

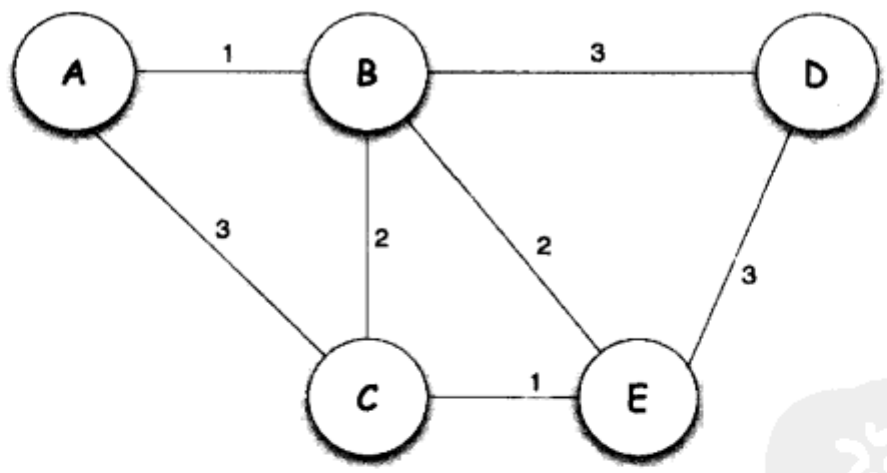


g 规划城市的水管铺设、天然气管道或电线都会涉及最小生成树的问题。最小生成树同样也能有效解决网络中计算机连接的相关问题，比如从一台计算机发送电子邮件到另一台。

h 10 栋(即圆圈所在处)。

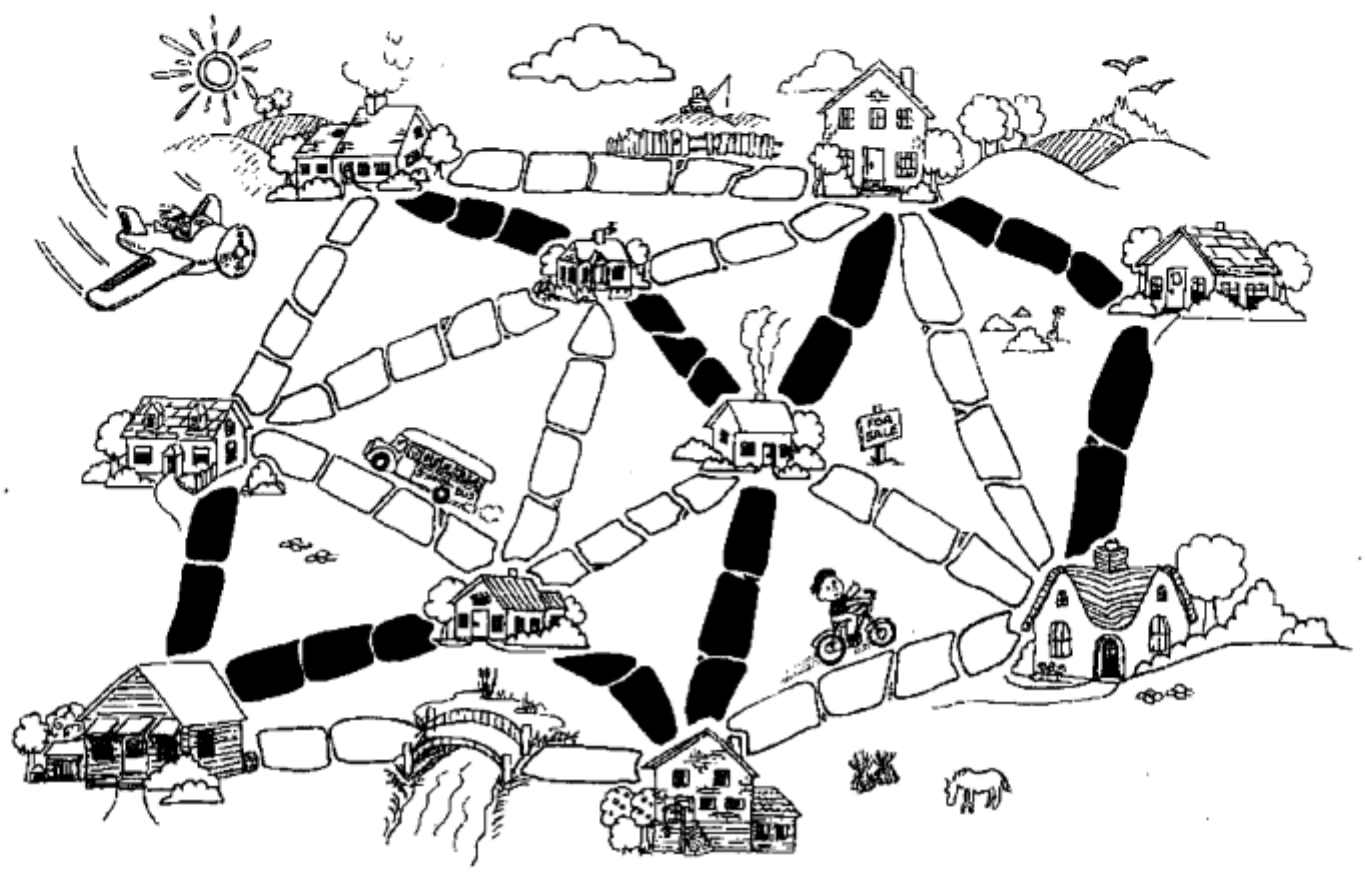
i 图的外观并不重要，只要房子 A 到 B 之间长度为 2、房子 B 和 C 之间长度为 2、房子 C 到 A 长度为 3，除此之外再没其他连接即可。无论哪张图，都能解决"保证房子 A 到 B 是最短的路径"这个问题。

j 答案如下图。

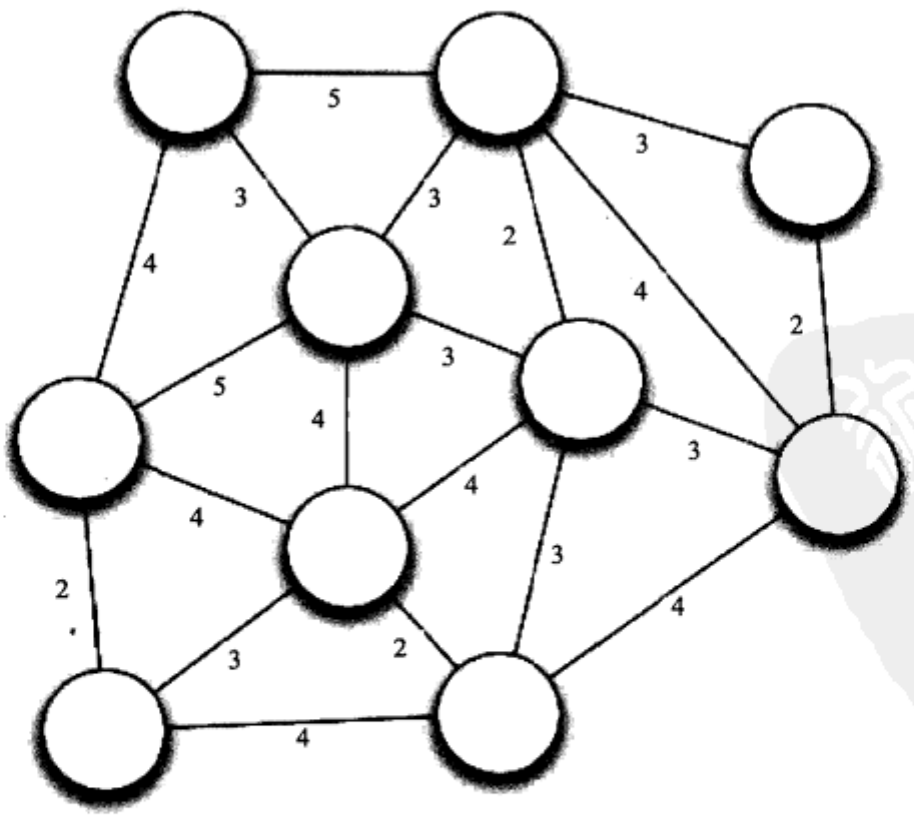


■ 小游戏 13.2：大一点的泥泞城市

k 寻找最佳路径有一个窍门，即从一张空白的地图开始，逐步增加石砖直到全部的房子都被连接起来，务必按照路径长度从小到大来添加道路，记住不要将两个已经连接的房子再用其他道路连起来。如果你改变添加道路的顺序，则会产生不同的最终布局，但是最后的结果还是能保持需要石砖的总数最少。本题的答案之一如下图所示。无论你信不信，你总能用这个简单的方法得到最优化的解答，这个方法被称为 Kruskal 算法。因此最佳方案是铺设 23 块石砖。你会发现孩子们很快就能找到属于他们自己的方案。鼓励他们互相对比一下用到的石砖数量，他们并不知道 23 块是最优的答案，所以就算他们已经得出 23 块石砖的方案，还是想要看看能否做得更好。注意，如果方案中存在"循环"(即从一条路离开一栋房子后，能走另外一条再返回这栋房子)，这就绝对不是最优的方案了。



1 下图是对应这个城市的图。



第14章　路由和死锁

■　小游戏 14.1：橘子游戏

每组安排 6 个人最适合来玩这个游戏，如果人数比较多，不妨将学生们分成几个小组。

a 无论什么情况，都能将信息正确传递到指定目的地。主要的问题是，如果有人拿到属于自己的“消息”后，可能会拒绝放手让这个“消息”离开，这样一来便阻塞了其他人拿到消息的道路。应该鼓励学生们在必要时候放弃属于自己的消息。

b 如果一直固执拿着消息不放手会阻塞其他消息的传递。特别当你拿着的是一条错误消息，而你两旁的人都拿到了属于他们的正确消息时，你就永远不要想能把手中的错误消息传递出去了。

c 这正是在问题 b 中描述过的情况。所以本题的答案为不能，正确的消息无法被传递。

d 在这个特殊的例子中，问题倒是也可以解决，因为这个网络中有另外一条路径可以绕过 C。

e 真实世界中存在着死锁。比如有两人在打架，每个人都扬言“他道歉之后我才会道歉”，那么他们永远都不会道歉言和了。另外一个例子是，设想有 4 辆车同时到达一个交叉路口，如果每辆车都等它右边的车先行，那么没有一辆车能再发动了！更多的例子我们在“有趣的事”小节中会谈到。

第15章　处理输入

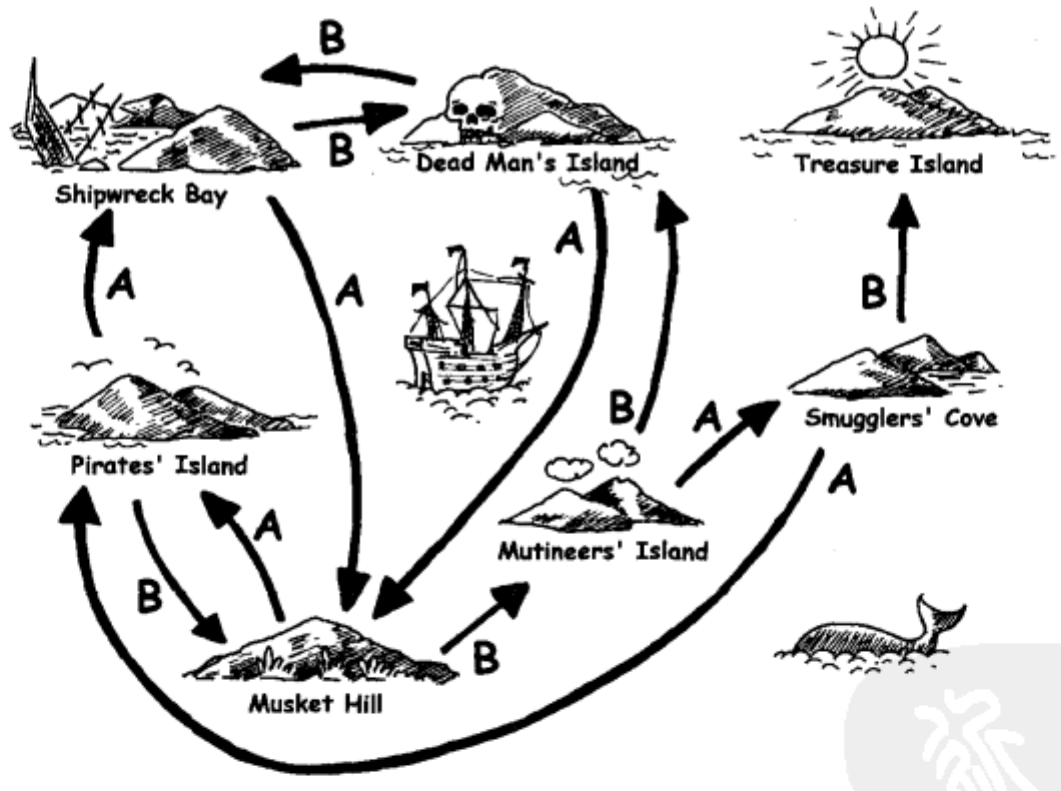
■ 小游戏 15.1：金银岛

a 抵达海盗岛（途经死亡岛和船难湾）。

b 船难湾（搭第二艘 A 船后，又从海盗岛再折回这里）。

c 本题的答案不定。从海盗岛出发最短的路径是，搭船 B 到步枪山，搭船 B 到暴徒岛，再搭船 A 到走私港，最后搭船 B 到金银岛。然而，很有可能学生们选择的路径并不是最短的，但只要他们能抵达终点就可以。

d 下图是完成后的地图。



e 最短航线是：在海盗岛搭乘船 B 到步枪山，再搭乘船 B 到暴徒岛，再搭乘船 A 到走私港，最后搭乘船 B 到金银岛。

f 比较慢的航线就有很多了，其中一条能走遍全部地点的路径是：从海盗岛搭乘船 A 到海难湾，搭乘船 B 到死亡岛，搭乘船 A 到步枪山，搭乘船 B 到暴徒岛，搭乘船 A 到走私港，搭乘船 B 到金银岛。孩子们得到的路径中或许会包括循环（参考 g 题的答案）。

g 可以找到许多种。比如，你可以在抵达暴徒岛之前反复在海盗岛和步枪山之间兜圈子。

■ 小游戏 15.2：用 FSA 来寻找规律

h 最有效的路径为 BBAB。

i 有许多答案都能满足要求。一个简单的方式是重复 BA 任意次，然后走路径 BBAB。比如 BABABABABABABBAB 这样的方案是能满足本题要求的。

j 可以，其中 BBA 循环了两次。

k 可以，这个方案中其实返回了海盗岛两次。

l 不能，最终会停在状态 3。

m AB：可以

n BABAA：可以

o ABBBA：不能

p AAABABA：可以

q AAABA：不能

r 只要指令中包含奇数个 A 就可以。因为 A 是接受状态的开关，而 B 则没有这种功能。

s AB：可以

t BA：不能

u ABAB：可以

v AABB：不能

w ABABA：不能

x ABBA：不能

y 包含且仅包含重复个的“AB”。

z B：可以

aa BBB：可以

bb BBA：可以

cc ABBA：可以

dd AAA：不能

ee 包含一个或几个 B 的指令都可以。

ff 短一点的例子比如“a big pirate laughed”（一个高大的海盗笑了）；长一点的例子比如“a friendly dog and the old pirate laughed and the friendly dog laughed”（一只友善的狗和一个老海盗笑了和这只友善的狗笑了）。这些句子完全不通顺，甚至连语法都是错的，再比如，你可以造一个句子“a old clown sang”（一个年迈的小丑在唱歌）。

gg 本题答案不定。

PDG

英文索引[12]

[12] 该部分为全书英文术语的章节索引。

- android与iphone及ipad开发书籍 ----------持续不断更新中.....
- c、c++、c#语言pdf书籍及vip视频教程 c、c++、c#、vc等-------持续不断更新中.....
- delphi《书籍》及《视频》教程 ----------持续不断更新中.....
- E网情深VIP系列视频教程 黑客破解菜鸟修练班，VB编程学习班，仿站学习培训，免杀培训，个人系统攻防系列教程，服务器搭建学习班，PHOTOSHOP平面设计班，基础制作论坛（论坛网站搭建），网赚系列教程，网站建设教程，网站漏洞基础，远程控制教程，软件破解班，脚本漏洞提权班
- IT9网络学院VIP系列视频教程 免杀培训班，VMware虚拟机，零基础学习C语言，网游外挂开发精品系列语音教程（外挂教程学习必备研修31课全），VB语言教程30课全，Delphi编程到精通，远程控制软件，加密解密班，网络安全与黑客攻防培训，从入门到精通完整系统化学习C++编程，从入门到精通零基础学习汇编，wordpress教程(个人博客系统49课全)，外行人做易语言盗号和钓鱼程序语音教程 网址：WLSAM168.400GB.COM
- Java书籍 ----------持续不断更新中.....
- photoshop、CorelDRAW、AutocAD等图像处理书籍及vip视频教程 -------持续不断更新中.....
- powerbuilder书籍大全
- Visual Basic语言vip视频教程及pdf书籍 ----------持续不断更新中.....
- windows、linux系统开发、系统封装等pdf书籍及VIP视频教程 ----------持续不断更新中.....
- 《3DS Max》pdf书籍
- 《汇编语言》、《反汇编》及《调试》pdf书籍及vip视频教程 ----------持续不断更新中.....
- 《电子书、电子书、还是电子书》pdf专题库 编程开发，家居美食，儿童益智，人物传记，增强记忆，快速阅读
- 信息系统项目管理师、网络工程师、系统分析师等软考类书籍
- 华中红客系列vip视频教程 脚本攻防培训班，源码免杀培训班，Css语言培训班，C语言，Dreamweaver网页设计，html网页设计培训班，PC安全班， php脚本语言培训班，VMWare虚拟机专题，webshell提权培训班，防站教程，零基础免杀培训班，刷钻速成班，脱壳破解班，外挂编写班，网络赚钱培训班，网站入侵培训班
- 外挂、驱动、逆向及封包视频教程 郁金香、独立团、夜猫论坛、天都吧、看流星论坛、一切从零开始等等
- 安全中国系列vip视频教程 易语言软件编程培训班，ASP.net网站开发项目实战培训班
- 我的收藏
- 按键精灵及TC脚本开发软件视频教程 ----------持续不断更新中.....

当前位置：/《电子书、电子书、还是电子书》pdf专题库

文件名 ⇕

PDF电子书专题库，内容详尽，每天不断更新！！

- 办公类软件使用指南
- 医学
- 历史人物传记
- 哲学宗教
- 外语资料（除英语外） （除英语外）
- 官场类小说
- 建筑工程类
- 情感生活类小说
- 政治军事
- 教育学习科普大全
- 文学理论
- 智力开发、增强记忆、快速阅读技巧大全
- 社会生活
- 科学技术
- 程序编程类
- 经济管理
- 网络安全及管理
- 网赚系列
- 美食小吃烹饪煲汤大全
- 课外读物

本网盘内容太多，持续不断更新，发布各类视频教程、pdf书籍，包括破解、加解密、外挂辅助制作，易语言培训教程、编程语言、网页制作等等，教程及书籍仅用于学习,如用于商业或非法用途的后果自负！

网址：WLSAM168.400GB.COM

- VIP培训教程，易语言黑月VIP视频教程，天都网易语言系列培训教程(100集全)，集中营易语言学习视频(80集)
- 棉猴系列vip视频教程 gh0st远控控制源码讲解教程，套接字编程，DLL程序编写，键盘监听驱动程序编写，驱动基础教程，AsyncSelect模型QQ程序教程，C++语言入门基础，NB5.5源码分析教程
- 游戏开发pdf书籍 ----------持续不断更新中.....
- 炒股投资pdf书籍及视频教程 短线高手系列，短线天王系列，操盘论道系列，翻倍黑马，看盘快速入门，庄家手法大曝光等等。 网址：WLSAM168.400GB.COM
- 热门小说集中营 傲世九重天，网游之三国时代，武动乾坤
- 甲壳虫VIP教程全集 asp教程，Delphi培训班，FLASH培训班，Java培训班，linux培训班，PHP培训班，源码免杀班，甲壳虫C++，脚本功防班，免杀班初、中、高级班，破解班，源码免杀班，脱壳班，易语言培训班，无特征码免杀，网站架构培训班，外挂高级班，外挂初级班第1、2部
- 破解、免杀、入侵、脱壳、攻防及漏洞分析系列VIP视频教程（80多部） 天草、黑客动画吧等等--------持续不断更新中....
- 网站建设相关的pdf书籍及各种vip视频教程 ----------持续不断更新中.....
- 网赚、淘宝系列vip视频教程 网赚30天新人魔鬼训练，屠龙网赚团队vip课程，站长大学网赚视频（50课全），图腾团队日赚1000元竞价行销教程，屠龙团队淘宝宝贝卖疯系列，站群网赚系列，淘宝开店视频，红星挂机日赚10元，百万流量系列，漂流瓶圣手全自动挂机引，贴吧邮件定向营销疯狂成交量月入万元
- 英语学习资料百科大全 不断更新。。。
- 饭客论坛系列VIP视频教程 脚本入侵班，黑客之免杀教程，易语言教程，无线网络攻防教程，入侵教程，delphi系列教程，黑客基础入门
- 黑客书籍 有关黑客、安全、加解密技术等等----------持续不断更新中.....
- 黑手安全网VIP系列视频教程 DIV+CSS网页布局，Dreamweaver教程，flsah动画教程，photoshop教程，跟我一起学C++课程，抓鸡
- 黑鹰、黑基、黑防、黑盾vip系列视频教程 破解提高班66课全，SQL注入，ASP注入教程，完完全全学会抓肉鸡，脱壳破解教程50课全，提权班，C语言特训班26讲全，黑客脚本特训班，黑客工具特训班，dedecms仿站教程，VC编写远控30课全，网页美工特训班，木马免杀特训班，驱动开发技术VIP培训班，外挂破解等等。

- [电脑世界的通关密语：电脑编程基础].(杉浦贤).滕永红.扫描版.pdf
- [程序语言的奥妙：算法解读（四色全彩）].(杉浦贤).李克秋.扫描版.pdf
- [差错：软件错误的致命影响].(帕伯斯).邝宇恒等.扫描版.pdf
- [算法之道（第2版）].邹恒明.扫描版.pdf
- [O'Reilly：深入学习MongoDB].(霍多罗夫).巨成等.扫描版.pdf
- [深入浅出WPF].刘铁猛.扫描版.pdf
- [Go语言·云动力（云计算时代的新型编程语言）].樊虹剑.扫描版.pdf
- [精通.NET互操作：P/ Invoke、C++ Interop和COM Interop].黄际洲等.扫描版.pdf
- [编程的奥秘：.NET软件技术学习与实践].金旭亮.扫描版.pdf
- [O'Reilly：学习OpenCV（中文版）].(布拉德斯基等).于仕琪等.扫描版.pdf
- [Go语言编程].许式伟等.扫描版.pdf 网址：WLSAM168.400GB.COM
- [MySQL技术内幕：SQL编程].姜承尧.扫描版.pdf
- [Tomcat权威指南（第2版）].(布里泰恩等).吴豪等.扫描版.pdf
- [Ext江湖].大漠穷秋.扫描版.pdf
- [IT名人堂·Oracle DBA突击：帮你赢得一份DBA职位].张晓明.扫描版.pdf

Total: 77 1 2 3 4 5 6 ›

[General Information]
书名=不插电的计算机科学
作者=（新西兰）贝尔著
页码=211
ISBN=211
SS号=12767226
dxNumber=000008023854
出版时间=2010.11
出版社=该引擎未能查询到
定价：29.80
试读地址=http://book.duxiu.com/bookDetail.jsp?dxNumber=000008023854&d=F1E99861C6E979408800B578C2D6DEE9A&fenlei=07030102 31&sw=%B2%BB%B2%E5%B5%E7%B5%C4%BC%C6%CB%E3%BB%FA%BF%C6%D1%A7
全文地址=http://ndfe.5read.com/image/ss2jpg.dll?did=n27&pid=F0C77943B1F98B4F43FEB79A4E4F5B9C92D87207E780B33D5B68DCAD27C7ACBC0EDA6CE5599363EA0DF64AF3BFFDCCCA96167D33C621BA6A328CAF4BEBB4A67418 20A16207 9B9CA2DB6A530AA12D6DD433990B02F483EE2204CB3304F995A94DF9F0B14DC6B6967A434E8A4A2E19D9698C85&jid=/