数据结构与算法 (Python)

课程概论

谢正茂 webg@PKU-Mail

北京大学计算机系

March 14, 2021

目录

关于计算

- 问题，以及如何解决问题
- 计算模型：图灵机
- 可以通过“计算”解决的问题
- 计算复杂性
- 不可计算问题

问题的种类和解决方案

- 问题：人们在生产、生活中碰到的各种未知的东西
  - 云是什么？什么是无理数？什么是万物的起源？
  - 为什么会下雨？为什么 $\sqrt{2}$ 是无理数？生命的意义是什么？
  - 怎么让粮食长得更多？怎么求最大公约数？怎么维护公平与正义？
- 问题解决的目标“从未知到已知”，有不同的领域：
  - 科学/技术/工程：用逻辑、数学、实验、仿真等方法解决
  - 哲学：思维与存在，意识和物质的关系，通过思考来解决
  - 宗教：灵魂的归宿在哪儿？存在不同的答案
- 有些问题已经解决，很多问题尚未解决，有些问题似乎无法解决
  - 尚未解决和无法解决问题的共性：表述含混、标准不一、涉及主观、结果不确定
  - 有些问题天然无法明确表述（主观、价值观、意识形态、哲学问题等）
  - 有些可明确表述的问题仍然无法解决（留后待述）

解决问题的工具

- 数学：解决问题的终极工具
  - 在长期的发展过程中，人们把已经解决的问题逐渐表述为数学命题与模型；
  - 尚未解决的问题，人们试图通过数学建模，采用数学工具来解决；
  - 无法解决的问题，人们试图转换表述、明晰问题来部分解决。
- 为什么是数学?
  - 数学具有清晰明确的符号表述体系；
  - 严密确定的推理系统；
  - 但正如科学不是万能的，数学也不是万能的

问题的分类和明确表述

- 问题的分类
  - What：是什么？面向判断与分类的问题；
  - Why：为什么？面向求因与证明的问题；
  - How：怎么做？面向过程与构建的问题。
- 问题表述的核心
  - 给定一个特定的输入
  - 通过某些中间过程
  - 得到一些特定的输出

整系数代数多项式方程是否有整数解的问题

- 整系数代数多项式方程也称“丢番图”方程
- 希尔伯特提出的 23 个重要数学问题的第十个，希尔伯特第十问题
- 明确表述的数学问题
- 1900 年希尔伯特提出，1970 年马蒂亚赛维奇证明不可解
- 《可计算性与不可解性》- 戴维斯

“计算”概念的提出

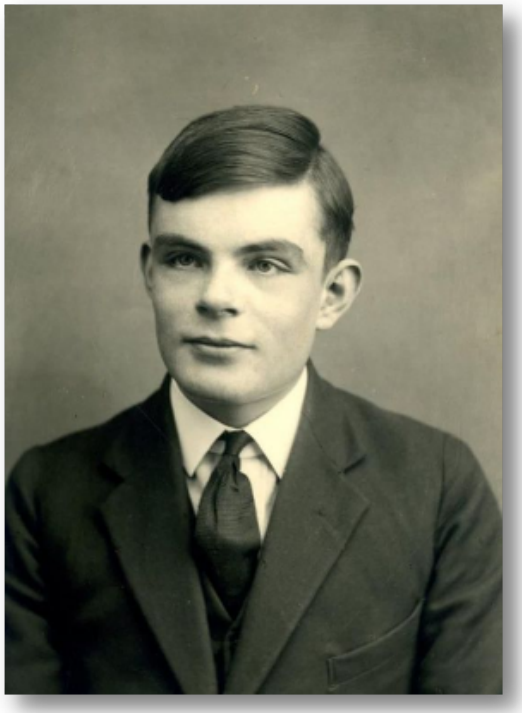
- 能否找到“一种有穷、确定、可行的方法”，来判定任何一个数学命题的真假。——抽象的“计算”概念提出
  - 由有限数量的明确指令构成；
  - 指令执行在有限步骤后终止；
  - 指令每次执行都总能得到正确解；
  - 原则上可以由人单独采用纸笔完成，而不依靠其它辅助；
  - 每条指令可以机械地被精确执行，而不需要智慧和灵感。

“计算”的数学模型

- 20 世纪 30 年代，几位逻辑学家几乎同时各自独立提出了几个关于“计算”的数学模型
  - 奥地利逻辑学家、数学家哥德尔 (K.F. Godel,1906-1978) 和美国逻辑学家、数学家克莱尼 (S.C. Kleene,1909-1994) 的递归函数模型
  - 美国逻辑学家、数学家丘奇 (A. Church,1903-1995) 的 Lambda 演算模型
  - 波兰裔美国逻辑学家、数学家波斯特 (E.L. Post,1897-1954) 的 Post 机模型
  - 英国逻辑学家、数学家图灵 (A.M. Turing,1912-1954) 的图灵机模型
- 后续研究证明，这几个“基于有穷观点的能行方法”的计算模型，全都是等价的，在某个模型下“可计算”的问题，在另外的模型下也是“可计算”的
- 虽然希尔伯特的计划最终被证明无法实现，即不存在“能行方法”来判定任何一个数学命题的真假，总有数学命题，其真假是无法证明的
- 但“能行可计算”的概念，成为了计算理论的基础，其中的一些数学模型（如图灵机）也成为现代计算机的理论基础

图灵机 Turing Machine

- 1936 年，Alan Turing 提出的一种抽象计算模型，基本思想是用机器模拟人们用纸笔进行数学运算的过程，但比数值计算更为简单
- 有限数量的明确指令
  - 在纸上写上或擦除某个符号；
  - 把注意力从纸的一个位置转向另一个位置；
  - 每个阶段要决定下一步的动作，取决于：
    - (a) 当前所关注的纸上某个位置的符号
    - (b) 当前思维的状态。

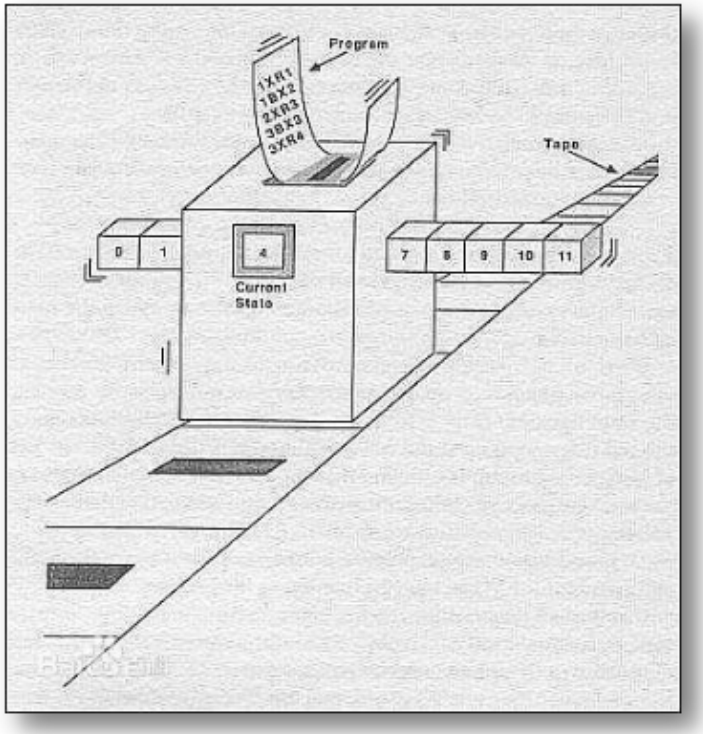


艾伦. 图灵

- 《论可计算数及其在判定问题中的应用》1936
  首先提出了图灵机的概念，并且图灵机是工程可实现的，奠定了所有计算机的基础，图灵被称为“计算机之父”。
- 《Computing Machinery and Intelligence》1950
  首先提出了图灵测试的概念，奠定了所有人工智能的基础，图灵也被称为“人工智能之父”。
- *ACM* 图灵奖有“计算机诺贝尔奖”之称。

图灵机 Turing Machine

- 图灵机由以下几部分构成
  - 一条无限长的纸带，分为一个个相邻的格子，每个格子可以记录一个符号
  - 一个读写头，可以在纸带上左右移动，能读出和擦写格子的字符
  - 一个状态寄存器，记录机器所在的状态，状态的数量是有限的
  - 一系列有限的控制规则
    - 每条规则指明了在当前状态下，根据读写头读入的字符
    - 来确定读写头擦写格子的字符，是否移动，是否改变状态



- 一种语言是否是 “图灵完备”?

图灵机运行的一个例子

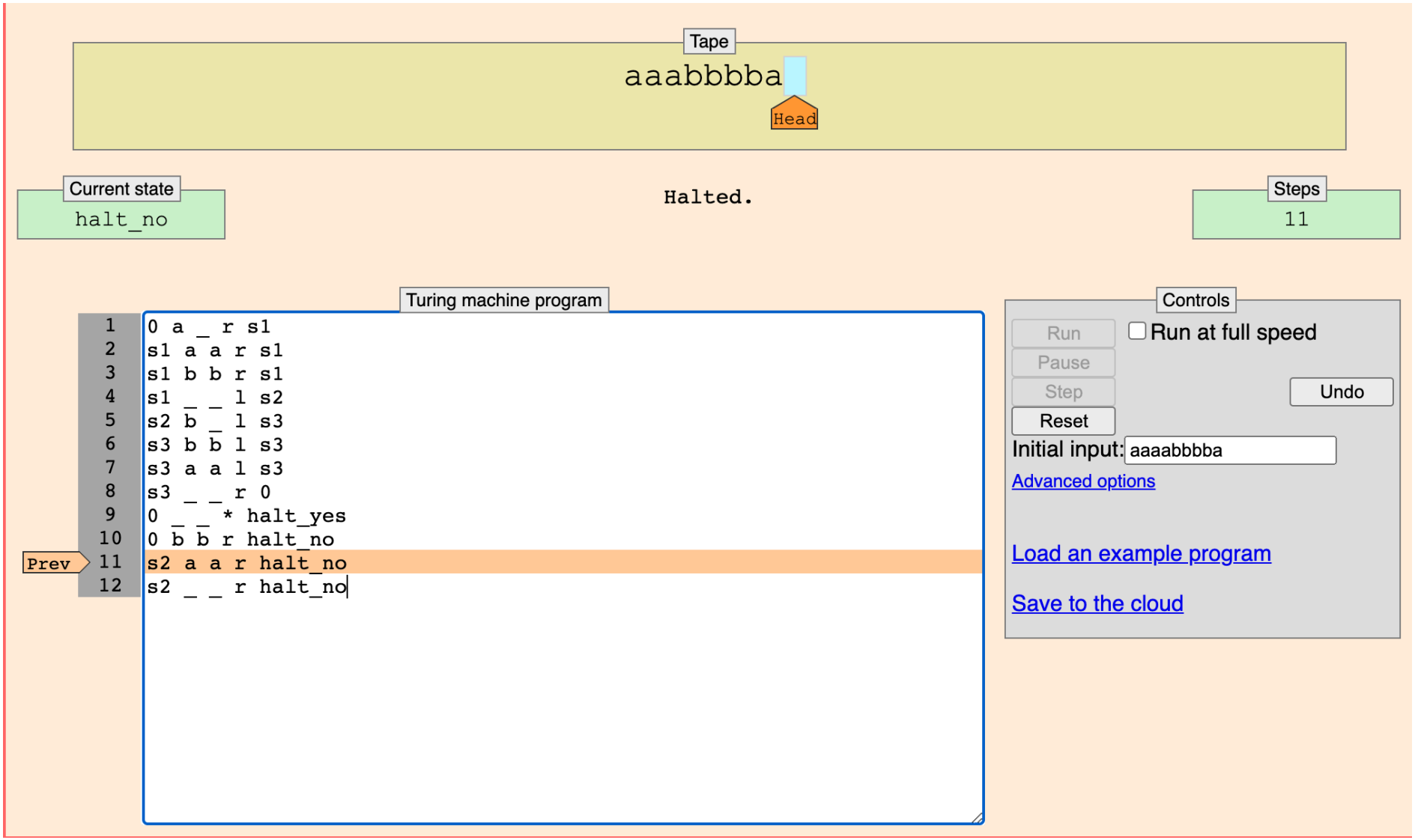
- 判定 $\{a^m b^m | m >= 0\}$：左半部全是 a，右半部全是 b，且 ab 数量相等的字符串
  - 如：ab、aabb、aaaabbbb，进入“接受”状态，如：b、ba、abb，进入“拒绝”状态
- 规则思路：将 a 和 b 一一对消，如果最后剩下空白 B 则接受，否则拒绝
  - $< s0, a, B, s1, R >$：初始碰到 a，消去，s1，右移
  - $< s1, a, a, s1, R >$：消去 1 个 a 的状态，继续右移，找最后一个 b
  - $< s1, b, b, s1, R >$：继续右移
  - $< s1, B, B, s2, L >$：右移到头状态 s2，回移
  - $< s2, b, B, s3, L >$：如果有 b，消去，进入左移状态 s3
  - $< s3, b, b, s3, L >$：左移
  - $< s3, a, a, s3, L >$：左移

图灵机运行的一个例子

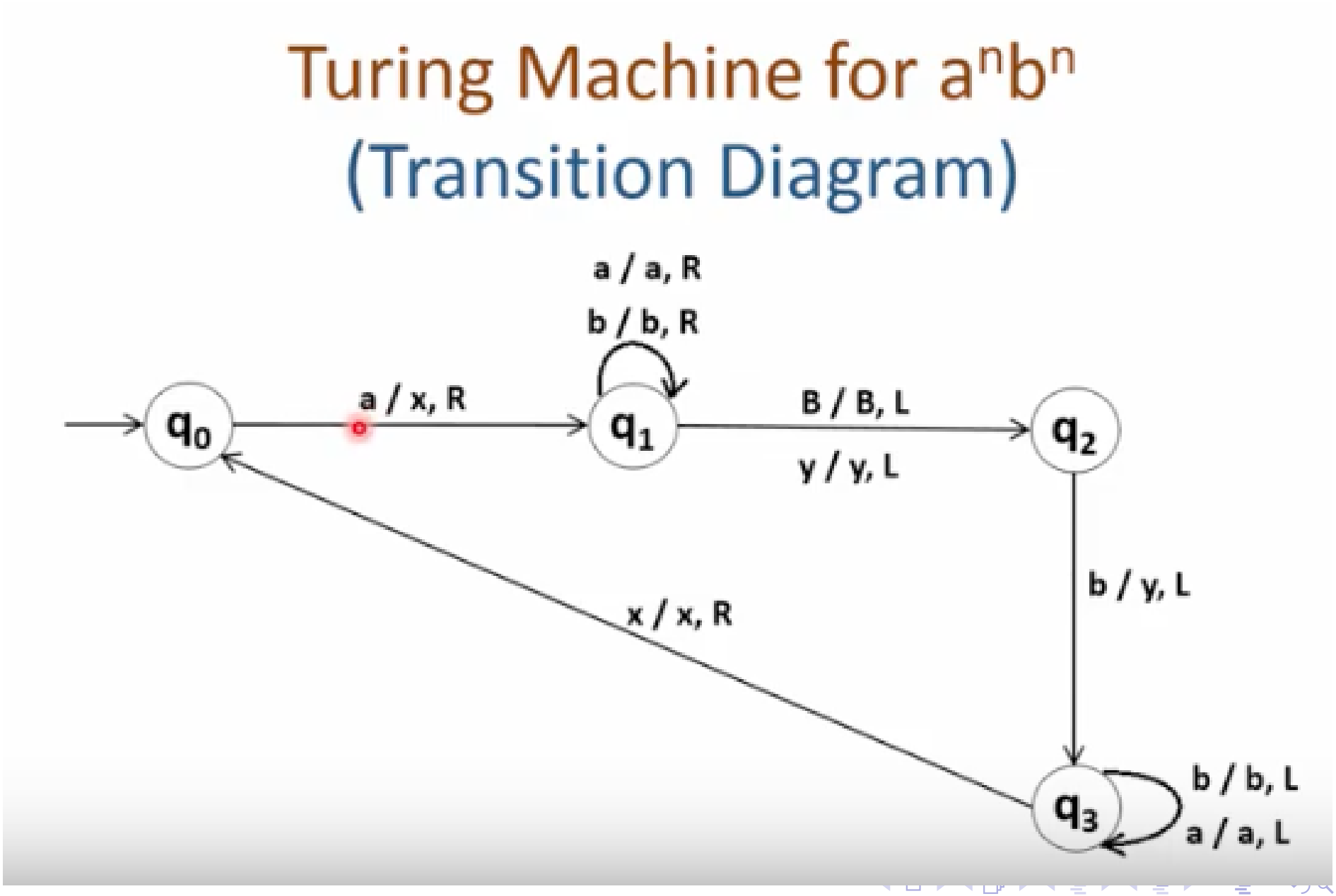
- 继续... ...
  - $<$ s3, B, B, s0, R $>$: 左移到头变初始状态 s0，右移看下个字符
  - $<$ s0, B, B, sY, N $>$: a,b 都能一一消完，则进入“接受”状态，停机
  - $<$ s0, b, b, sN, R $>$: b 多了，或者在 a 前，进入“拒绝”状态，停机
  - $<$ s2, a, a, sN, R $>$: a 多了，或者在 b 后，进入“拒绝”状态，停机
  - $<$ s2, B, B, sN, R $>$: a 多了，进入“拒绝”状态，停机

虚拟图灵机

http://morphett.info/turing/



图灵机 Turing Machine：状态转换图

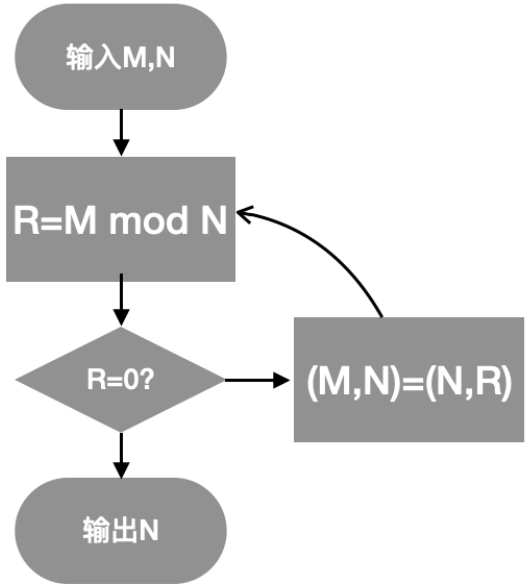


可以通过“计算”解决的问题 1/3

- 用任何一个“有限能行”的计算模型可以解决的问题，都是“可计算”的
- What：分类问题，可以通过树状的判定分支解决
- Why：证明问题，可以通过有限的公式序列来解决
  - 从不证自明的公理出发，根据推理规则，一步步推理得出最后待证明的定理
- How：过程问题，可以通过流程来解决

世界上最早的算法：欧几里德算法（最大公约数）

- 公元前 3 世纪，记载于《几何原本》
  - 辗转相除法求最大公约数
- 辗转相除法处理大数时非常高效
  - 它需要的步骤不会超过较小数的位数（十进制下）的五倍
  - 加百利・拉梅 (Gabriel Lame) 于 1844 年证明了这点，
  - 并开创了计算复杂性理论。

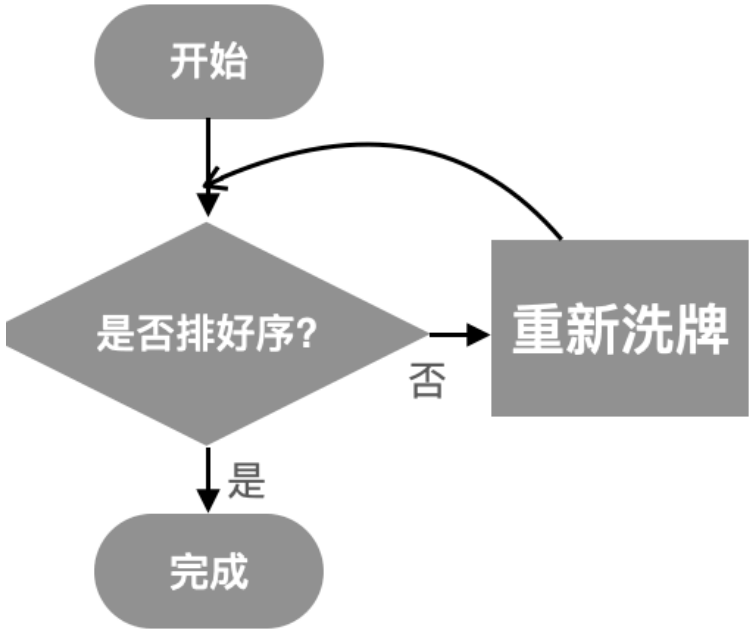


问题本身的计算复杂性

- “基于有穷观点的能行方法”的“可计算”概念仅仅涉及到问题的解决是否能在有限资源内（时间/空间）完成，并不关心具体要花费多少计算步骤或多少存储空间。数学意义上的“有限”，实际上并不“可行”。
- 由于人们对资源（时间/空间）的拥有相当有限，对于问题的解决需要考虑其可行性如何，人们发现各种不同的问题，其难易程度是不一样的
  - 有些问题非常容易解决，如基本数值计算；
  - 有些问题的解决程度尚能令人满意，如表达式求值、排序等；
  - 有些问题的解决会爆炸性地吞噬资源，虽有解法，但没什么可行性，如哈密顿回路、货郎担问题等
- 定义一些衡量指标，对问题的难易程度（所需的执行步骤数/存储空间大小）进行分类，是计算复杂性理论的研究范围

不同算法的复杂性

- 但对于同一个问题，也会有不同的解决方案，其解决效率上也是千差万别，例如排序问题，以 n 张扑克牌作为排序对象
  - “冒泡”排序：即每次从牌堆里选出一张最小的牌，这样全部排完大概会需要 n^2 量级的比较次数
  - “Bogo”排序方法，也称“猴子排序”：如果没有排好序，重新洗牌，直到排序成功！这样全部排完，平均需要 n*n! 量级的比较次数（最坏的情况是永远都无法完成排序）



猴子排序

计算复杂性理论 vs. 算法研究

- 计算复杂性理论研究问题的本质，将各种问题按照其难易程度分类，研究各类问题之间的难度级别，并不关心解决问题的具体方案
- 而数据结构与算法，则研究问题在不同现实资源约束情况下的不同解决方案，致力于找到具体的计算资源条件下，效率最高的那个算法方案
  - 不同的硬件配置（手持设备、平板电脑、PC 设备、超级计算机）
  - 不同的运行环境（单机环境、多机环境、网络环境、小内存）
  - 不同的应用领域（消费级别、工业控制、生命维持系统、航天领域）
  - 甚至不同的使用状况（正常状况、省电状况）
- 如何对具体的算法进行分析，并用衡量指标评价其复杂度，我们在后面的课程里还会详细介绍

不可计算问题

- 有不少定义清晰，但无法解决的问题
  - 并不是目前尚未找到，而是在“基于有穷观点的能行方法”的条件下，已经被证明并不存在解决方案
- “停机问题”：判定任何一个程序在任何一个输入情况下是否能够停机
- 不可计算数：几乎所有的无理数，都无法通过算法来确定其任意一位是什么数字
  - 可计算数很少：如圆周率 Pi，自然对数的底 e

计算机科学的研究对象

- 对计算机的研究（计算机体系结构）只是计算机科学的一个领域。
- 计算机科学主要研究的是问题、解决问题的过程，以及问题的解决方案
  - 包括了前述的计算复杂性理论，以及对算法的研究
- 为了更好地处理机器相关性或独立性，引入了“抽象 abstraction”的概念，用以从“逻辑 logical”或者“物理 physical”的不同层次上看待问题及解决方案
- 一个关于“抽象”的例子：汽车
  - 从司机观点看来，汽车是一台可以带人去往目的地的代步工具，司机上车、插钥匙、点火、换档、踩油门加速、刹车。从抽象的角度说，司机看到的是汽车的“逻辑”层次，司机可以通过操作各个机构来达到运输的目的，这些操纵机构（方向盘、油门、档位）就称为“接口 interface”

对事物的抽象：汽车

司机眼中的汽车

- 钥匙、方向盘、离合器、油门、刹车、挡把、侧后视镜、后视镜
- 中控大屏、时速表、转速表、油量表、水温表、温度调节按钮
- 司机座椅、乘客座椅、车窗、车门、空调出风口
- 部件功能
  - 方向盘：转动（角度）
  - 油门：踩下（深度）
  - ……
- 如果是维修工的话，就需要看到与挡把相连的“变速箱”

对问题的抽象：信号灯问题 1/2

- 问题：为一个 5 条道路相交的路口设计安全有效的交通信号灯管理系统。

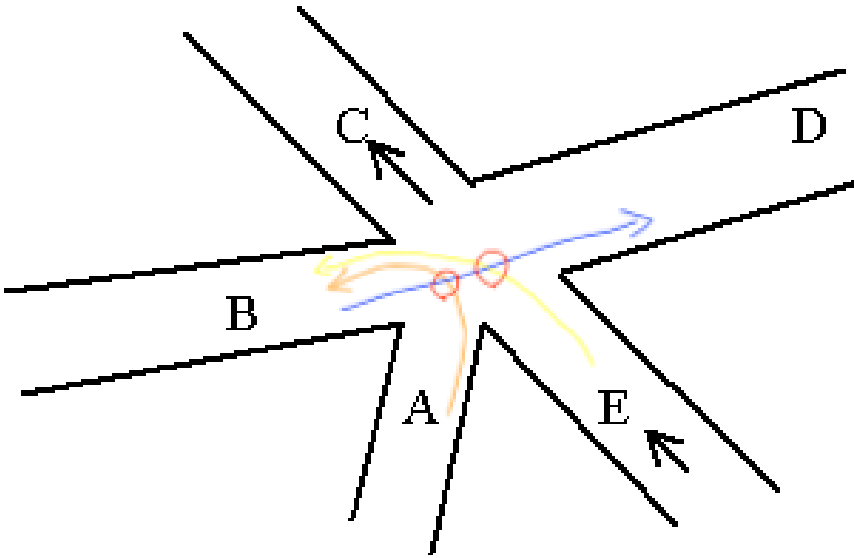


图 1.1　一个交叉路口的模型

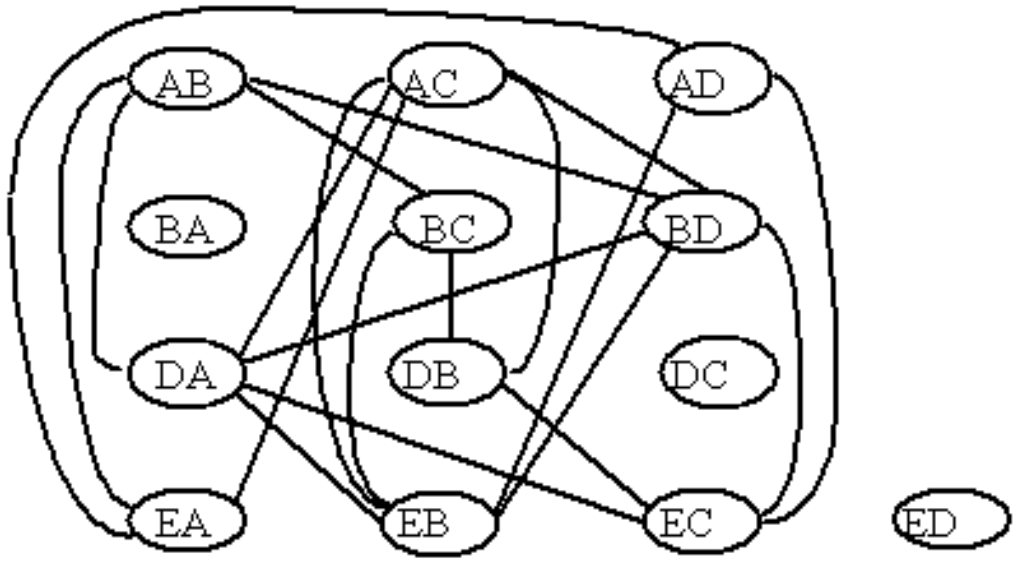
根据路口的实际情况，E、C 单向通行。可以确定 13 个可能的通行路线：

- A→B，A→C，A→D，
- B→A，B→C，B→D，
- D→A，D→B，D→C，
- E→A，E→B，E→C，E→D。

为每个路线设置独立的信号灯，信号灯亮起时，该路线放行。

对问题的抽象：信号灯问题 2/2

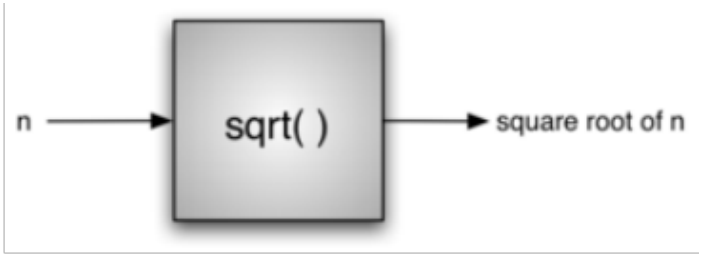
- 假设“靠右通行”，把 A→B 简写成 AB，每个路线看成一个节点，在不能同时行驶的路线间画一条连线 (表示它们互相冲突)，便可以得到图 1.2 所示的网状图。
- 把图中的节点（路线）分组，同一个组里的路线同时放行，“安全性”要保证有连线的节点不在同一个组里。
  - 分成 13 个组，每个组只有一个节点，显然满足要求，但路口通行的效率最低。
  - 如何分最少的组，获得最高的效率?
- 问题变为经典的图着色问题，可采用已知的解决方案。



过程抽象

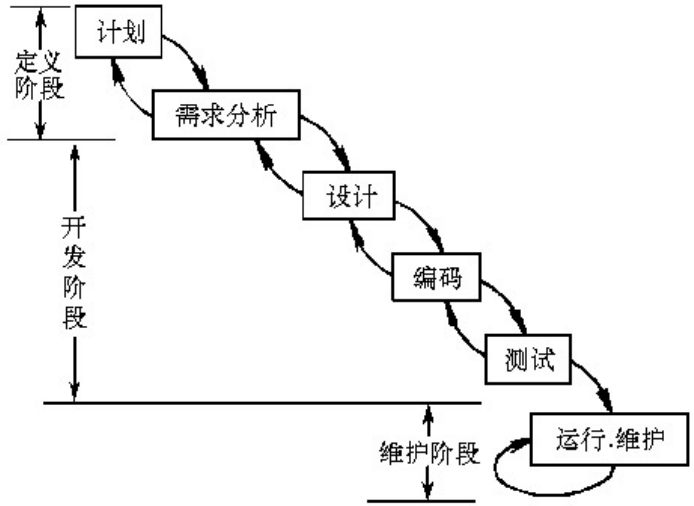
- 使用需要的功能，而不管功能的实现细节
  - 计算机可以用来编辑文档、收发邮件、上网聊天、处理照片等等
  - 普通用户都不需要具备对计算机内部如何处理的知识，实现这些功能是计算机的“逻辑”层次。
  - 相关的计算机科学家、程序员、技术支持、系统管理员分别去了解硬件/软件/网络等各方面的底层细节。
- “抽象”发生在各个不同层次上，程序员进行编程，也会涉及到“抽象”
  - 如计算一个数的平方根，程序员可以调用编程语言的库函数 math.sqrt()，直接得到结果
  - math.sqrt() 的具体实现，由另外的程序员完成。
  - 这种功能上的“黑盒子”称作“过程抽象 procedural abstraction”

```
>>> import math
>>> math.sqrt(16)
4.0
>>>
```



计算机解决问题的过程

- 需求分析：弄清所要解决的问题是什么，并用一种语言清楚地描述出来。
- 概要设计：根据问题，确定程序模块、模块的输入、输出
- 详细设计：建立程序系统的结构，重点是算法的设计和数据结构的设计。
- 编码阶段：采用适当的程序设计语言，编写出可执行的程序。
- 测试和维护：发现和排除在前几个阶段中产生的错误，经测试通过的程序便可投入运行，在运行过程中还可能发现隐含的错误和问题。



数据有关的几个概念

- 数据：计算机所能处理的所有符号的集合
  - 数字、字符、图像、声音、文本
- 数据元素：数据这个集合中的一个个体
- 数据对象：一类数据元素，组成一个数据对象
  - 所有的桌子、所有的同学
- 数据项：一个数据元素中有若干个数据项，是有意义的最小单元
  - 员工号、手机号、身份证号

数据的几个特点

- 数据混杂之后是没有意义的，不能传达信息和交流
  - 举例：160687700118616021531110108197705316333
  - 需要把不同的部分区分、组织起来：
    - 员工号：1606877001
    - 手机号：18616021531
    - 身份证号：11010819770531633 3
- 数据之间是有联系的，合理的利用联系，常常影响算法的效率和选择。
- 数据之间是有结构的，比如分层结构（树状结构）
- 在某种数据结构上，可以定义一组运算
  - 学籍管理，每个同学信息包括：姓名、性别、出生日期、学号、院系、学分等
  - 运算：加入、查找、删除、修改

数据结构的定义和核心问题

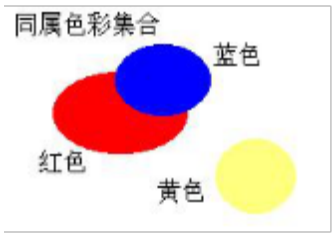
- 数据结构：按照逻辑关系组织起来的一批数据，按一定的存储结构把它存储在计算机中，并在这些数据上定义了相关运算的集合。
- 学习数据机构主要考虑下面四个核心问题：
  - 数据的各种逻辑结构（关系）和存储（物理）结构，以及对应关系
  - 每种结构所适应的各种运算
  - 针对问题，设计相应的算法
  - 能分析算法的效率，包括时间和空间复杂度

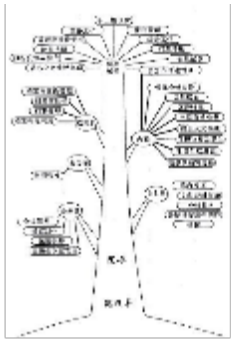
数据结构三要素

- 逻辑结构：定义了结构中的基本元素之间的相互关系。
  - 数据元素之间的逻辑关系；二元组 (D, R)，其中 D 是数据元素的有限集合，R 是 D 上的关系的有限集。
- 存储结构：给出了结构中的基本元素之间的存储方式
  - 包括元素的表示和关系的表示。
- 数据的运算：这个结构具有的行为特征
  - 体现为在存储结构上的具体实现算法。

数据的逻辑结构

- 集合：数据元素仅仅“同属于一个集合”，而没有其他关系
- 线性结构：数据元素之间仅存在一对一的关系（唯一后继、唯一前驱）
- 树形结构：数据元素之间存在一对多的关系（层状结构，唯一前驱，多个后继）
- 图状或网状结构：数据元素之间存在多对多的关系，元素之间的关系是任意的（多个前驱，多个后继）
- 它们之间存在的包含关系：集合 $\subset$ 线性结构 $\subset$ 树形结构 $\subset$ 图状或网状结构





数据的存储结构

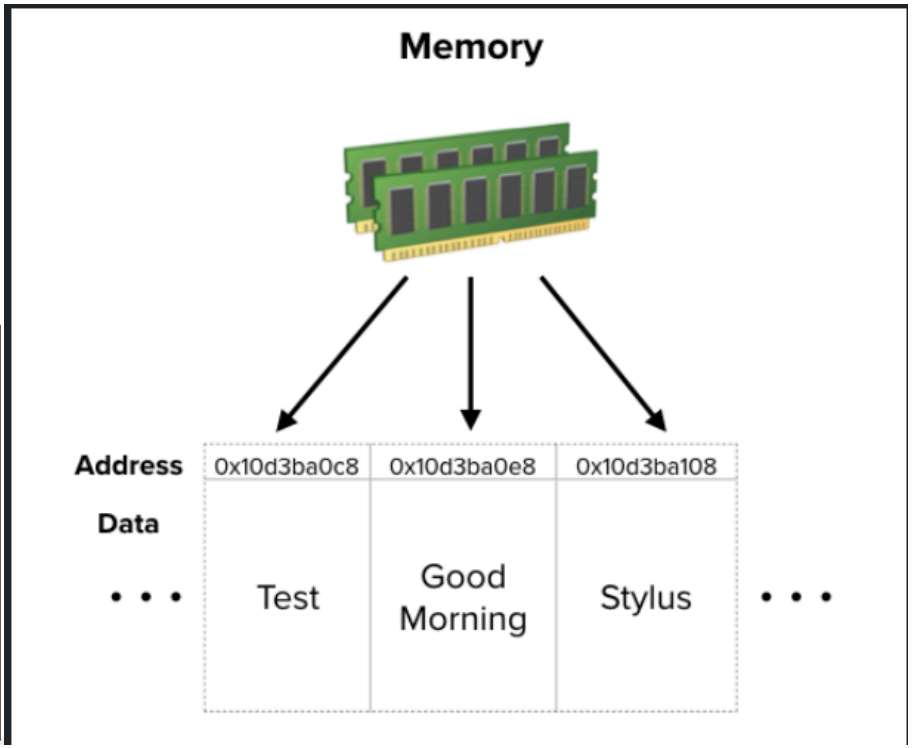
- 顺序存储结构：它是把逻辑上相邻的结点存储在物理位置相邻的存储单元里，结点间的逻辑关系由存储单元的邻接关系来体现。
- 链式存储结构：它不要求逻辑上相邻的结点在物理位置上亦相邻，结点间的逻辑关系是由附加的指针字段表示的。
- 索引存储结构：除建立存储结点信息外，还建立附加的索引表来标识结点的地址。
- 散列存储结构：就是根据结点的关键字直接计算出该结点的存储地址。

四种存储结构既可单独使用，又可组合使用。

顺序存储结构

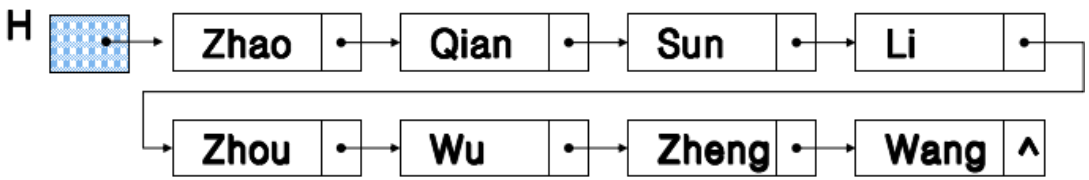
- 顺序存储结构：它是把逻辑上相邻的结点存储在物理位置相邻的存储单元里，结点间的逻辑关系由存储单元的邻接关系来体现。

逻辑地址	数据元素	存储地址	数据元素
0	k_0	$Loc(k_0)$	k_0
1	k_1	$Loc(k_0)+c$	k_1
…	…	…	…
i	k_i	$Loc(k_0)+i*c$	k_i
…		…	
n-1	k_{n-1}	$Loc(k_0)+(n-1)*c$	k_{n-1}



链接（链状）存储结构

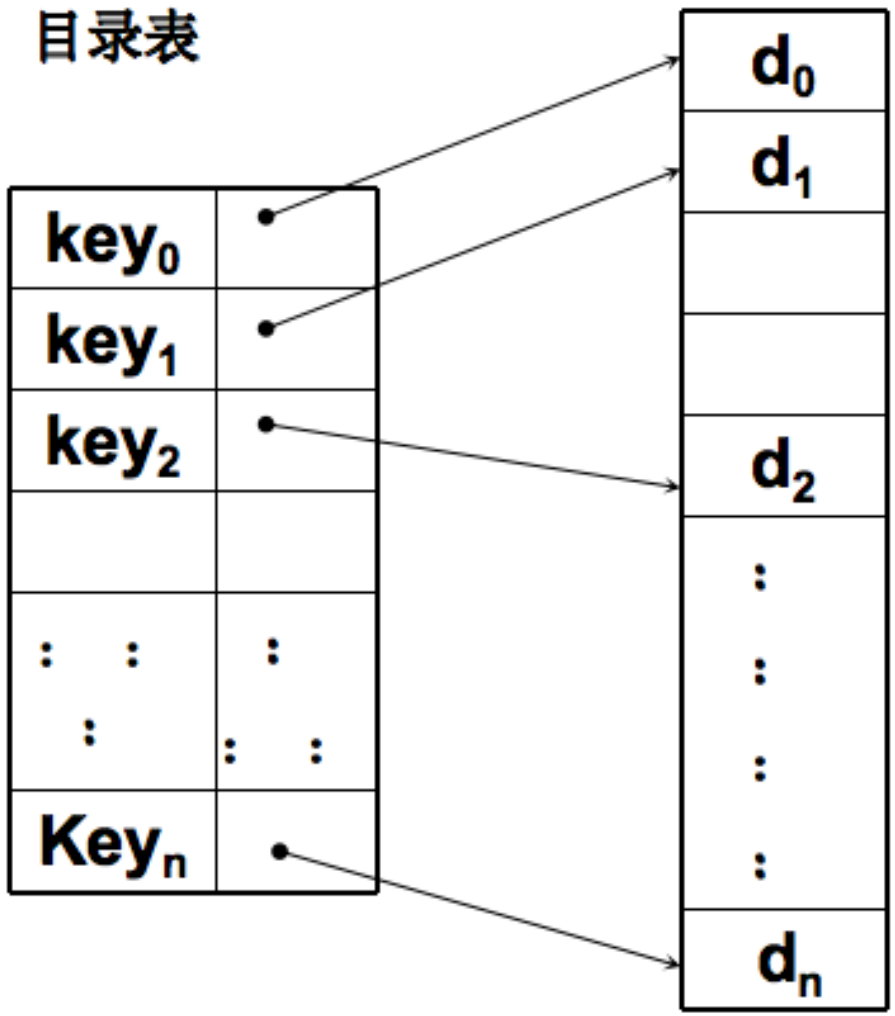
- 链接（链状）存储结构：它不要求逻辑上相邻的结点在物理位置上亦相邻，结点间的逻辑关系是由附加的指针字段表示的



存储地址	数据域	指针域
1	Li	43
7	Qian	13
13	Sun	1
19	Wang	NULL
25	Wu	37
31	Zhao	7
37	Zheng	19
43	Zhou	25

索引存储结构

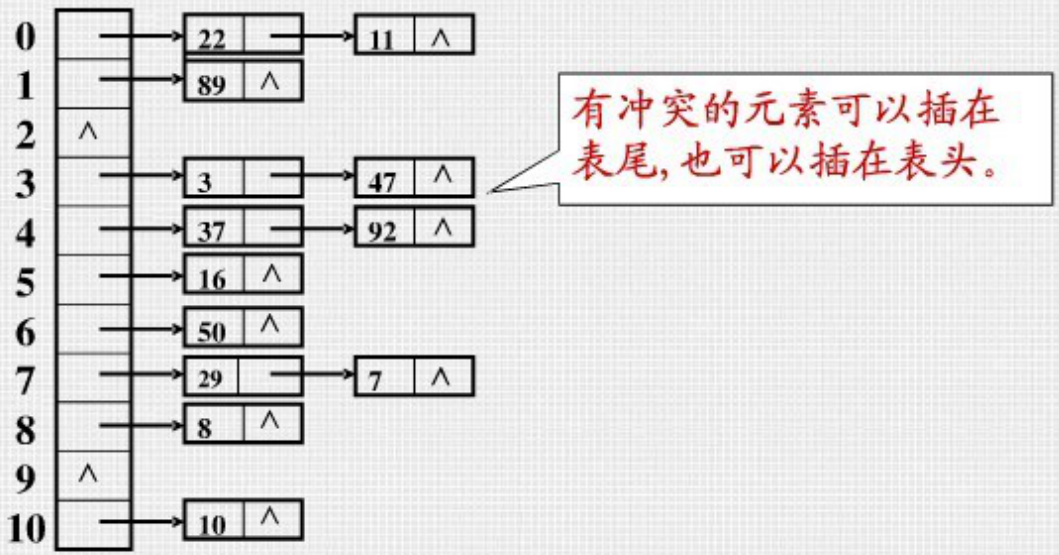
- 索引存储结构：除建立存储结点信息外，还建立附加的索引表来标识结点的地址。



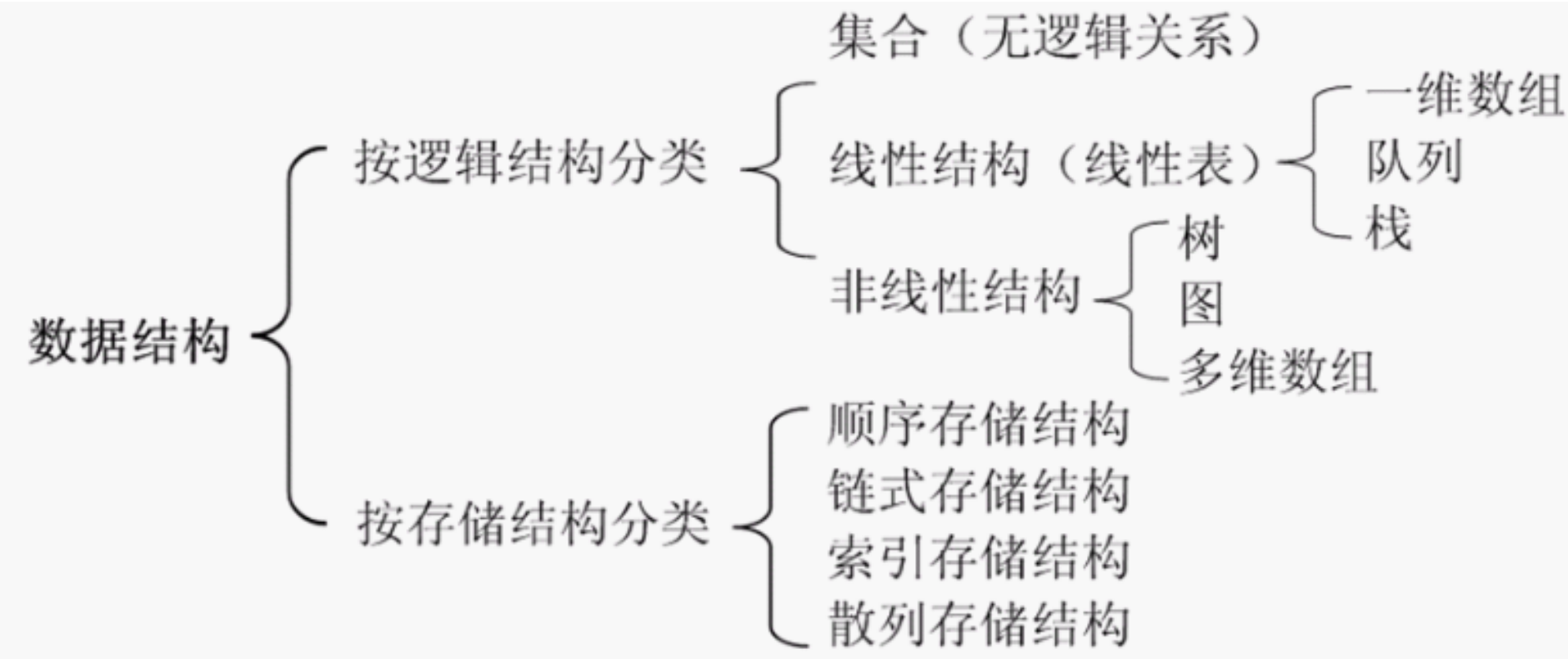
散列（hash，哈希）存储结构

- 散列存储结构：就是根据结点的关键字直接计算出该结点的存储地址。

例：设{ 47，7，29，11，16，92，22，8，3，50，37，89 }的哈希函数为：Hash(key)=key mod 11，用拉链法处理冲突，建表。



总结：数据结构的分类

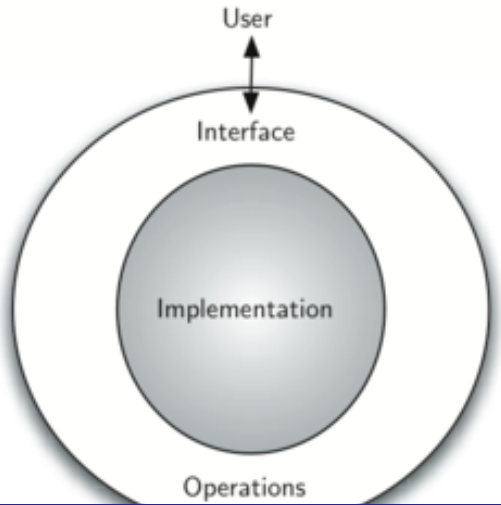

数据结构
按逻辑结构分类
集合（无逻辑关系）
线性结构（线性表）
一维数组
队列
栈
非线性结构
树
图
多维数组
按存储结构分类
顺序存储结构
链式存储结构
索引存储结构
散列存储结构

数据的运算

- 定义在逻辑结构上的一系列操作以及这些操作在存储结构上的实现；
  - 数据的运算是定义在逻辑结构上的，而具体的实现是基于存储结构。
- 常用的运算：
  - 检索/定位
  - 插入
  - 删除
  - 修改
  - 排序

为什么要有“抽象数据类型”

- 为了控制问题和问题解决过程的复杂度，我们需要利用抽象来保持问题的“整体感”而不会陷入到过多的细节中去
- 这要求对现实问题进行建模的时候，对算法所要处理的数据，也要保持与问题本身的一致性，不要有太多与问题无关的细节
- 前面谈到的“过程抽象”启发我们进行“数据抽象”
  - 相对于基本数据类型的“抽象数据类型 ADT:Abstract Data Type”
  - ADT 是对数据进行处理的一种逻辑描述，并不涉及实现细节
  - ADT 建立了一种对数据的“封装 encapsulation”
  - 封装技术将可能的处理实现细节隐蔽起来，能有效控制算法的复杂度
  - 用户通过 ADT 提供的操作来与接口交互



数据结构对 ADT 进行具体实现

- 数据结构 Data Structure，是对 ADT 的具体实现，同一种 ADT 可以采用不同的数据结构来实现
  - 数据结构采用程序设计语言的控制结构和基本数据类型来实现 ADT 所提供的逻辑接口，属于 ADT 的“物理”层次
  - 对数据实现“逻辑”层次和“物理”层次的分离，可以定义复杂的数据模型来解决问题，而不需要考虑此模型如何实现
  - 由于对抽象数据类型可以由多种实现方案，这种独立于实现的数据模型让底层程序员专注于实现和优化数据处理，而无须改变数据的使用接口，让用户专注于问题的解决过程
- 如电动车与汽油车，底层动力实现不同，但开车的操作接口（方向盘、油门、刹车、档位）基本都是相同的

主要介绍的数据结构-线性结构

- 线性表：线性表中各元素之间是一种简单的“线性”关系。
  - 顺序表和链表：是两种常用的实现线性表的数据结构。
  - 字符串：字符串也是一种特殊的线性结构，以字符为元素。
  - 栈：栈元素的存入和取出按照后进先出原则，最先取出的总是在此之前最后放进去的那个元素。
  - 队列：队列实现先进先出的原则，最先到达的元素也最先离开队列。

主要介绍的数据结构-非线性结构

- 树与二叉树：树和二叉树都属“树形结构”，在逻辑上表示了结点的层次关系。
- 字典：字典是一种二元组的集合，每个二元组包含着一个关键码和一个值。
  - 按关键码进行检索是字典中最常用的操作。
- 排序：针对由一组记录组成的文件，每个记录由若干字段组成，以排序码为依据进行排序。
- 图：包括一个结点集合和一个边集合，边集合中每条边联系着两个结点。

什么是算法

- 算法是利用一个有限的指令集，遵循指令流完成特定的功能
  - 算法是计算的狭义解释
  - 完成从输入 $\Rightarrow$ 输出的过程
- 算法的基本特性
  - 有穷性：算法经过有限步以后结束
  - 确定性：算法的下一步必须是明确的
  - 可行性：每一步都可行，且最终是正确的

什么是编程 Programming

- 编程是通过程序设计语言，将算法变为计算机可以执行的代码的过程
  - 没有算法，编程无从谈起
  - 图灵奖获得者 Niklaus Wirth 的名言：算法 + 数据结构 = 程序
- 程序设计语言需要为算法的实现提供实现“过程”和“数据”的机制
  - 具体表现为程序设计语言中的“控制结构”和“数据类型”
- 实现算法所需要的基本控制结构，程序设计语言均有语句相对应
  - 顺序处理、分支选择、循环迭代
- 程序设计语言也提供了最基本的数据类型来表示数据，如整数、字符等
  - 但对于复杂的问题而言，这些基本数据类型不利于算法的表达
- 还需要引入数据结构和抽象数据类型，来控制复杂度，便于清晰高效地表达算法

算法 $\neq$ 程序

- 算法是解决问题的一个方法或者过程，一个问题可以有多个算法
- 程序是算法在某种程序语言下的实现
  - 程序 = 算法 + 数据结构
- 主要区别在于：有穷性、确定性、正确性
  - 程序可以是无穷的，例如 *OS*
  - 程序可以是错误的，算法必须正确
  - 程序一定基于某种语言，在某种机器上可以执行
  - 算法可以用框图、自然语言、伪代码描述

为什么要学习算法

- 首先，通过学习各种不同问题的解决方案，有助于我们在面对未知问题的时候，能够根据类似问题的解决方案来更好解决
- 其次，各种算法通常有较大的差异，我们可以通过算法分析技术来评判算法本身的特性，而不仅仅根据算法在特定机器和特定数据上运行的表现来评判它
  - 即使同一个算法，在不同的运行环境和输入数据的情况下，其表现的差异可能也会很大
- 在某些情况下，我们或许会碰到棘手的难题，得能区分这种问题是根本不存在算法，还是能找到算法，但需要耗费大量的资源
- 某些问题的解决可能需要一些折衷的处理方式，我们需要学会在不同算法之间进行选择，以适合当前条件的要求