期中复习
系统思维概述

徐志伟 zxu@ict.ac.cn

提纲

- 期中复习
  - 对计算思维的十个理解：**Acu-Ex**ams-CP
  - 情况通报：已经学到了哪些知识
  - 计算思维基本解题思路：PEPS
  - 编程入门：基础数据表示，数组与循环，切片与递归
  - 逻辑思维：布尔逻辑，图灵机，邱奇-图灵论题，悖论与不完备性
  - 算法思维：高德纳定义，复杂度渐近记号，分治算法的设计与分析，进阶算法
- 系统思维概述

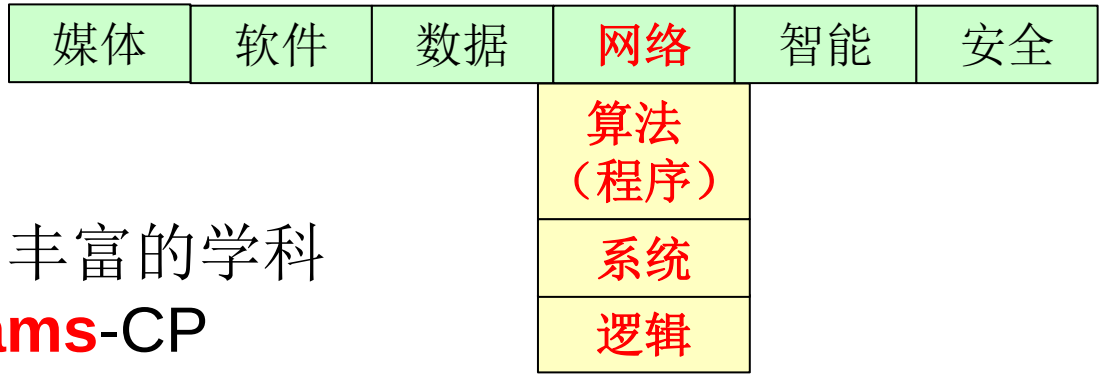
课件中可能包含素材引用，特此致谢！

1. 计算思维概念

对计算思维的十个理解： Acu-Exams-CP

(A): Automatically execution	**自动执行**。计算机能够自动执行由离散步骤组成的计算过程。
(C): Correctness	**正确性**。计算机求解问题的正确性往往可以精确地定义并分析。
(U): Universality	**通用性**。计算机能够求解任意可计算问题。
(E): Effectiveness	**构造性**。人们能够构造出聪明的方法让计算机有效地解决问题。
(X): Complexity	**复杂度**。这些聪明的方法（算法）具备时间/空间复杂度。
(A): Abstraction	**抽象化**。少数精心构造的计算抽象可产生万千应用系统。
(M): Modularity	**模块化**。多个模块有规律地组合成为计算系统。
(S): Seamless Transition	**无缝衔接**。计算过程在计算系统中流畅地执行。
(C): Connectivity	**连接性**。很多问题涉及用户/数据/算法的连接体，而非单体。
(P): Protocol Stack	**协议栈**。连接体的节点之间通过协议栈交互。

计算思维的特征



- 计算机科学已经成为一门丰富的学科
- 最基础的特征是**Acu-Exams**-CP

逻辑思维：保证计算过程是正确的（比特精准的） **Acu-Exams**
Logic thinking: correct computational processes

算法思维：创造巧妙的计算过程 **Acu-Exams**
Algorithmic thinking: smart computational processes

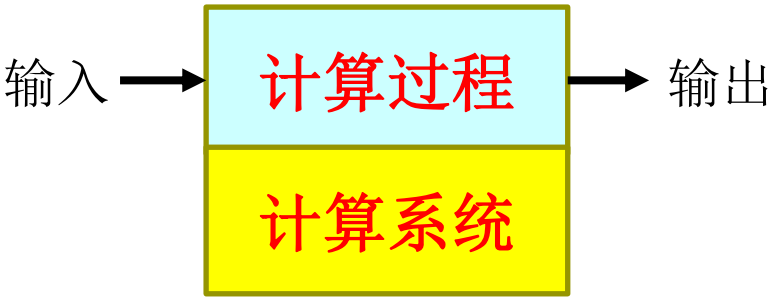
系统思维：使计算过程变得实用 **Acu-Exams**
Systems thinking: practical computational processes

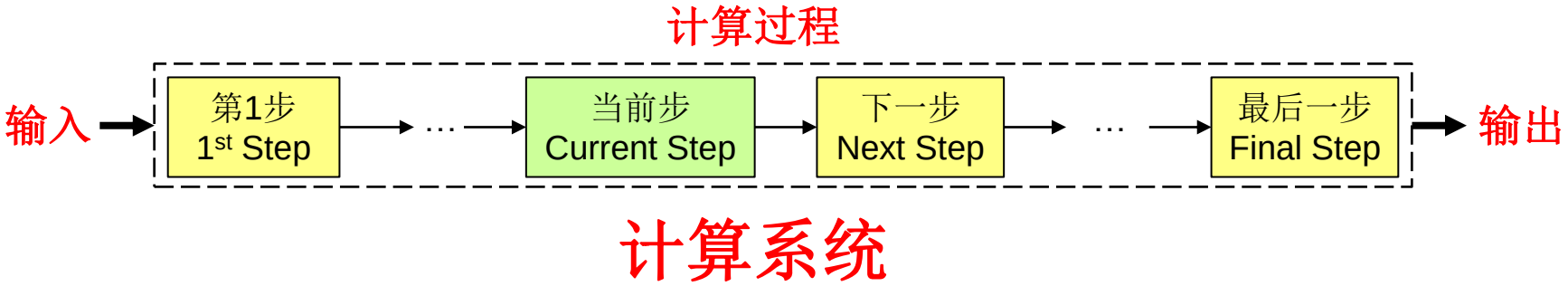
它们是整体：计算思维是一首交响乐

今天复习聚焦五个难题

Acu-Exams-CP

- 什么是计算思维的基本解题思路？
  - 活力法（PEPS）
- 什么是比特精准的正确性？
- 什么是构造性解题方法？
  - 什么是非构造性？
- 什么是可计算问题？
  - 什么是非图灵可计算？
- 算法思维妙在哪里？

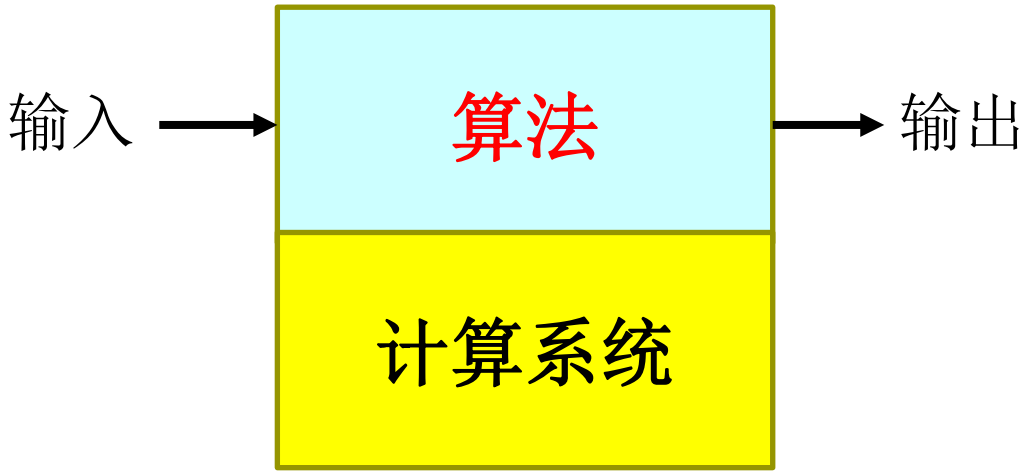




五个难题

Acu-Exams-CP

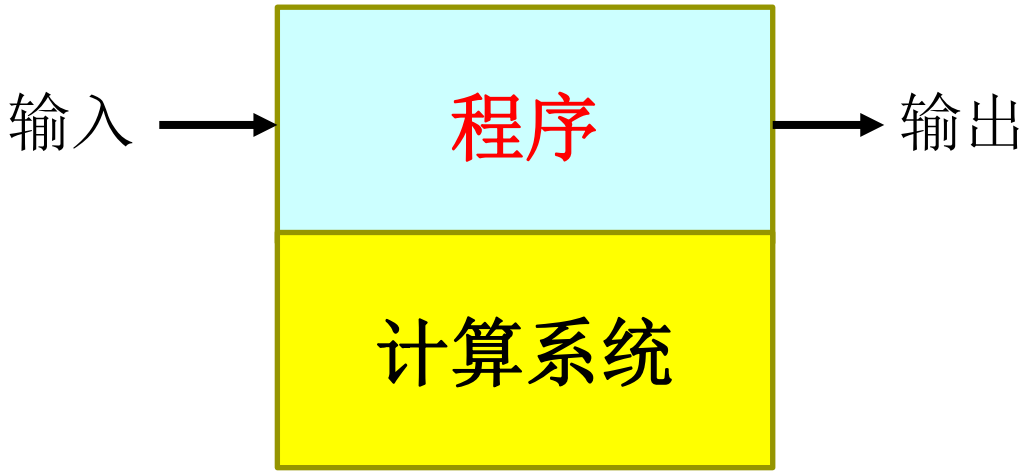
- 什么是计算思维的基本解题思路？
  - 活力法（PEPS）
- 什么是比特精准的正确性？
- 什么是构造性解题方法？
  - 什么是非构造性？
- 什么是可计算问题？
  - 什么是非图灵可计算？
- 算法思维妙在哪里？



五个难题

Acu-Exams-CP

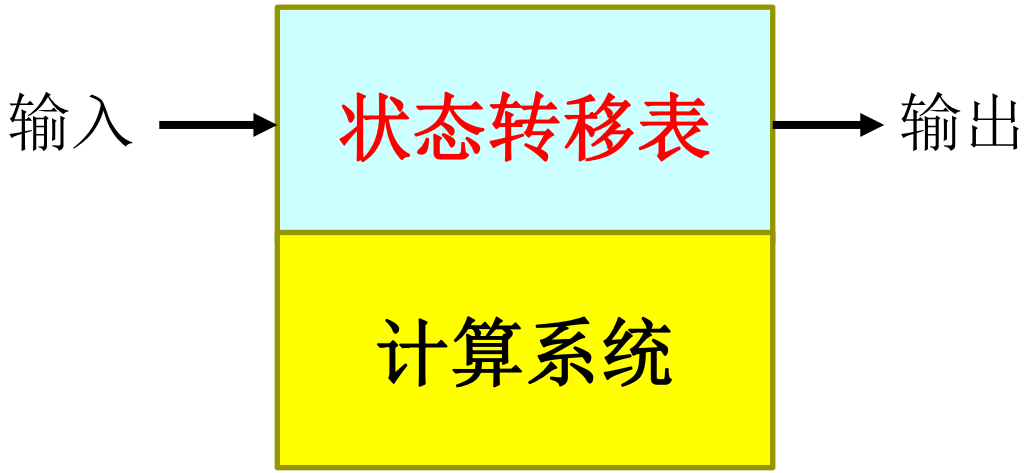
- 什么是计算思维的基本解题思路？
  - 活力法（PEPS）
- 什么是比特精准的正确性？
- 什么是构造性解题方法？
  - 什么是非构造性？
- 什么是可计算问题？
  - 什么是非图灵可计算？
- 算法思维妙在哪里？



五个难题

Acu-Exams-CP

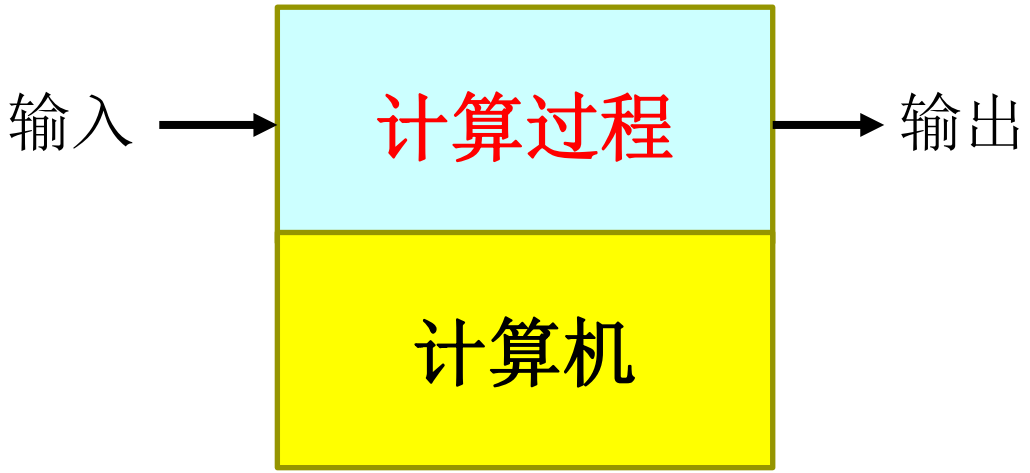
- 什么是计算思维的基本解题思路？
  - 活力法（PEPS）
- 什么是比特精准的正确性？
- 什么是构造性解题方法？
  - 什么是非构造性？
- 什么是可计算问题？
  - 什么是非图灵可计算？
- 算法思维妙在哪里？



五个难题

Acu-Exams-CP

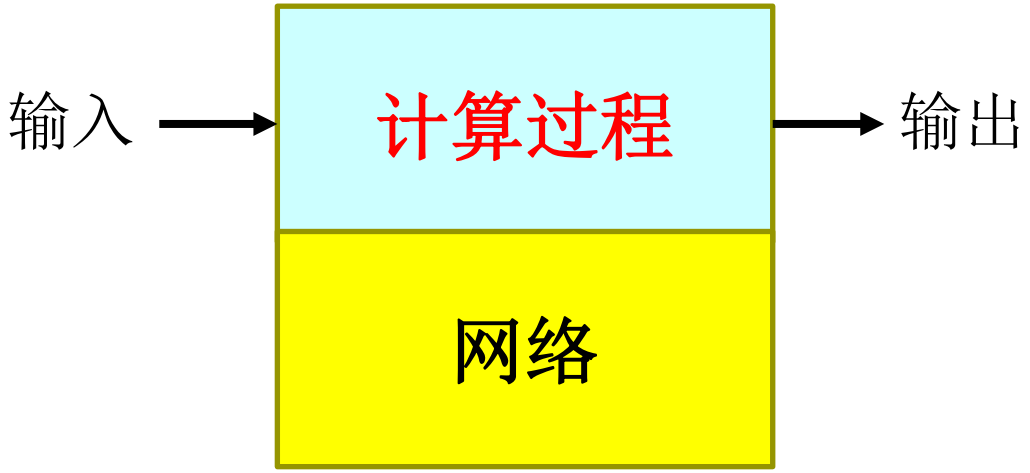
- 什么是计算思维的基本解题思路？
  - 活力法（PEPS）
- 什么是比特精准的正确性？
- 什么是构造性解题方法？
  - 什么是非构造性？
- 什么是可计算问题？
  - 什么是非图灵可计算？
- 算法思维妙在哪里？



五个难题

Acu-Exams-CP

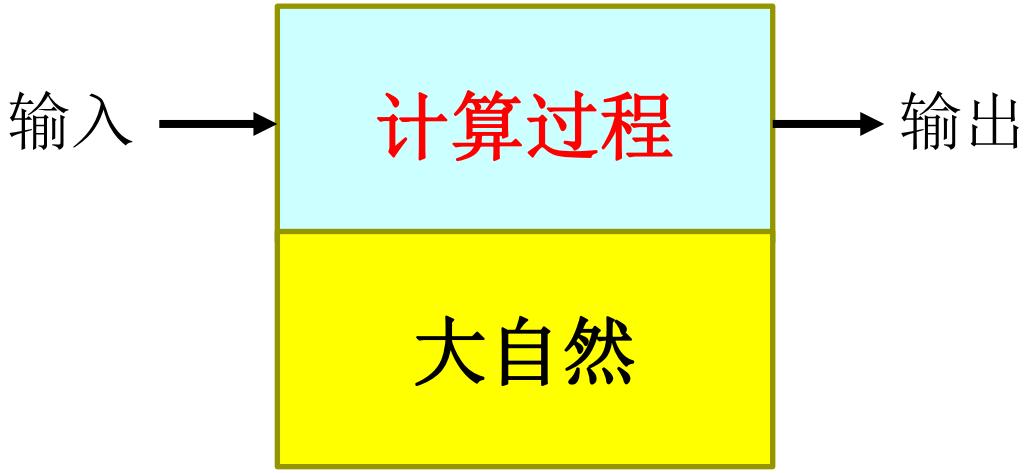
- 什么是计算思维的基本解题思路？
  - 活力法（PEPS）
- 什么是比特精准的正确性？
- 什么是构造性解题方法？
  - 什么是非构造性？
- 什么是可计算问题？
  - 什么是非图灵可计算？
- 算法思维妙在哪里？



五个难题

Acu-Exams-CP

- 什么是计算思维的基本解题思路？
  - 活力法（PEPS）
- 什么是比特精准的正确性？
- 什么是构造性解题方法？
  - 什么是非构造性？
- 什么是可计算问题？
  - 什么是非图灵可计算？
- 算法思维妙在哪里？



五个难题

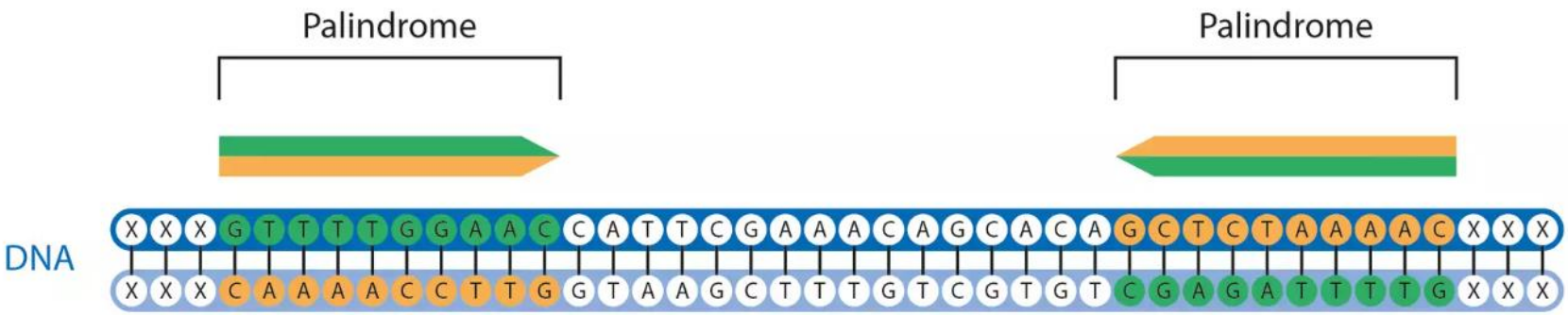
Acu-Exams-CP

- 什么是计算思维的基本解题思路？
  - 活力法（PEPS）
- 什么是比特精准的正确性？
- 什么是构造性解题方法？
  - 什么是非构造性？
- 什么是可计算问题？
  - 什么是非图灵可计算？
- 算法思维妙在哪里？

Human-Cyber-Physical Systems

Karp计算透镜：Nature computes

- “柳庭风静人眠昼，昼眠人静风庭柳”； 数字串1991
  - 完全回文
  - 逻辑思维中已经学到：有限状态机不能识别回文；图灵机可识别回文
- 自然界的回文
  - Y染色体包含大量回文结构
  - 2020年诺贝尔化学奖成果：**CRISPR**-Cas9基因编辑技术
    - CRISPR：“规律间隔成簇短回文重复序列”
    - Clustered Regularly Interspaced Short Palindromic Repeats
  - 并非完全回文



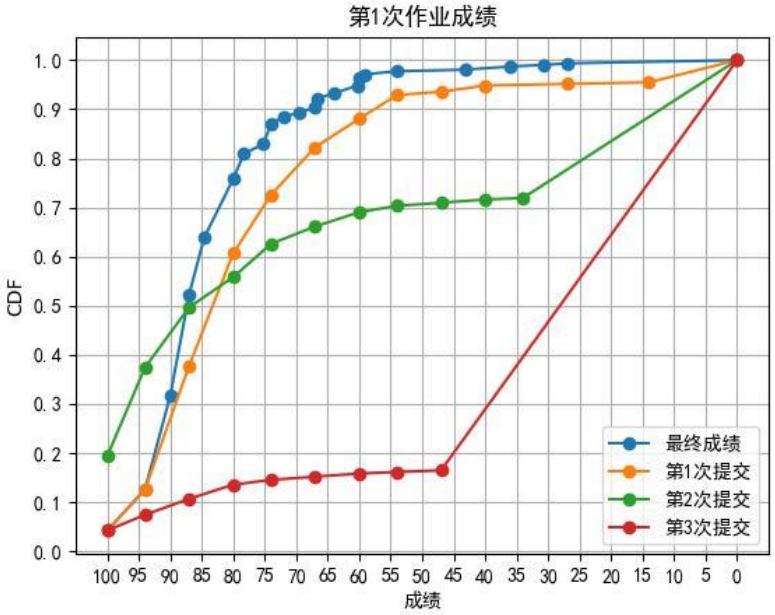
在无乳链球菌中发现的CRISPR的特色回文碱基对序列

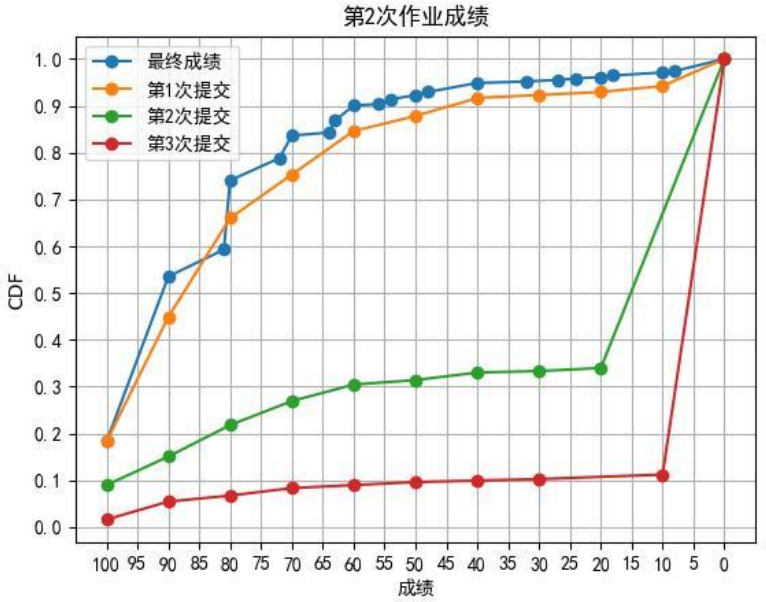
MPG Art for Science https://www.mpg.de/11823627/crispr-cas9-palindromes-structure

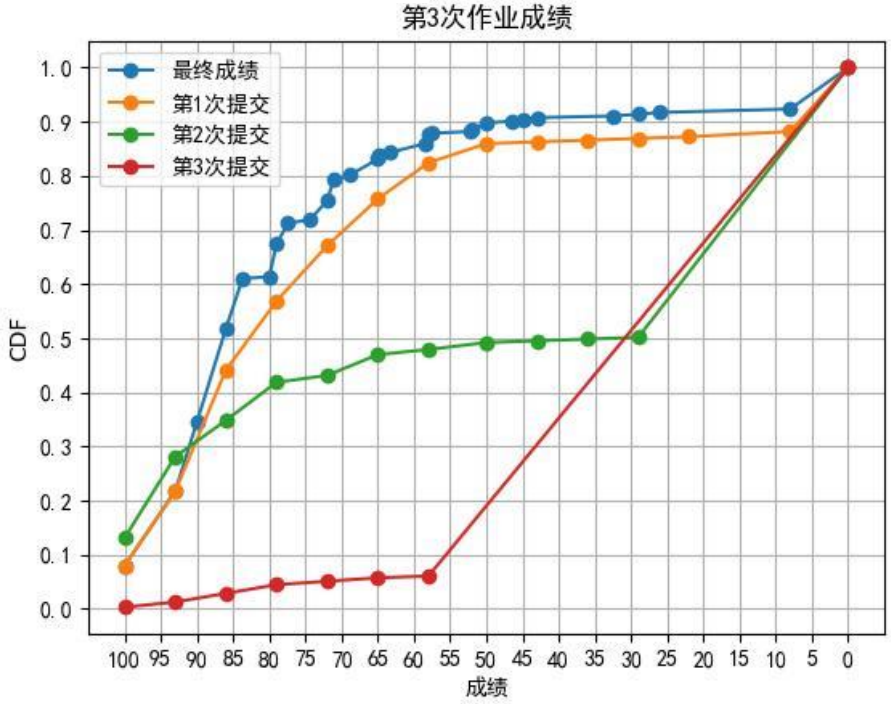
2. 通报：同学们已经学到了不少知识

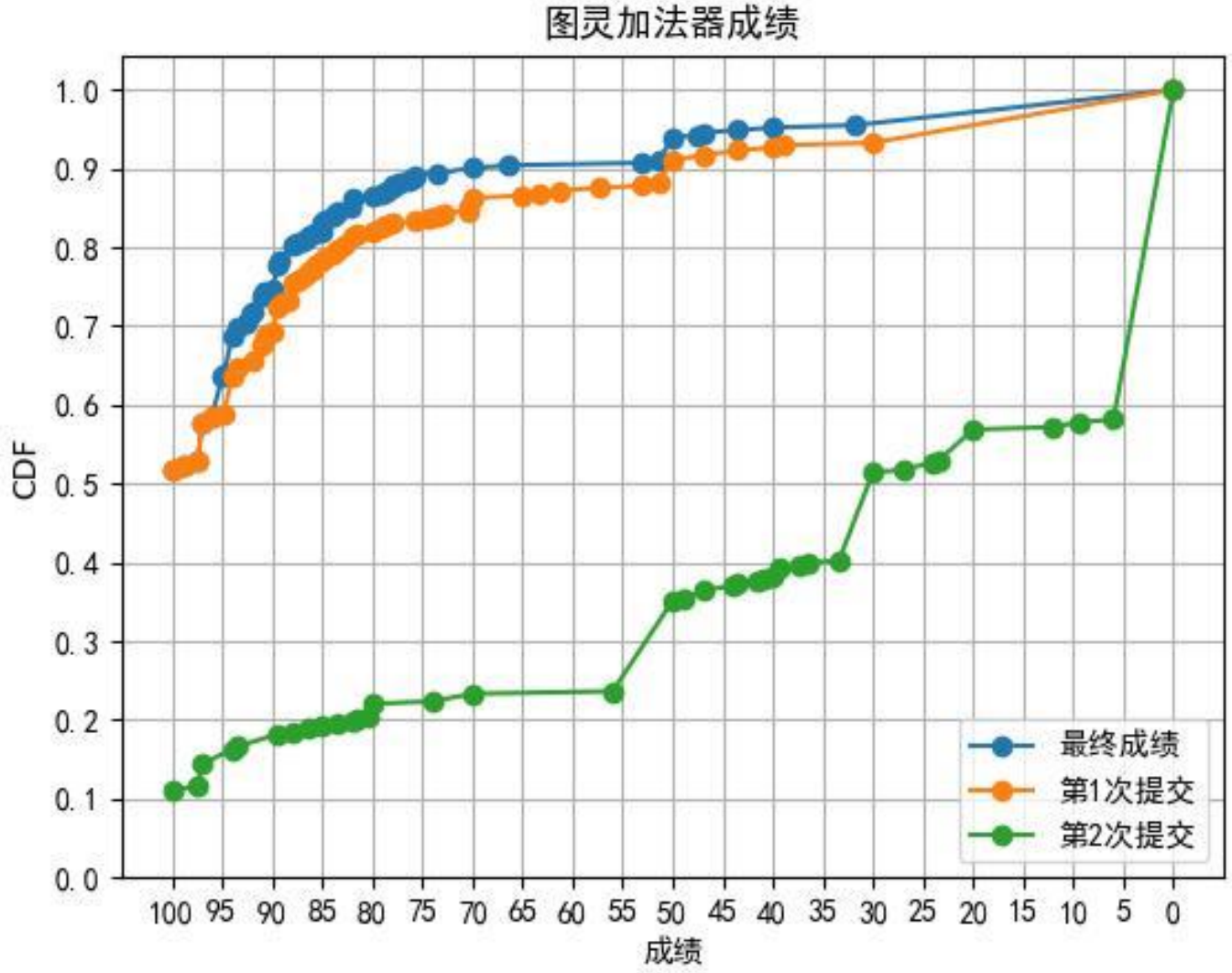
作业和实验的中位数成绩

作业	第一次	最终
计算思维概述	80	87
初识程序设计与执行	80	90
逻辑思维	79	86
算法思维		
实验		
编程基础	100	100
加法图灵机	100	100


第1次作业成绩
CDF
成绩
最终成绩
第1次提交
第2次提交
第3次提交


第2次作业成绩
CDF
成绩
最终成绩
第1次提交
第2次提交
第3次提交


第3次作业成绩
CDF
成绩
最终成绩
第1次提交
第2次提交
第3次提交


图灵加法器成绩
CDF
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
100 95 90 85 80 75 70 65 60 55 50 45 40 35 30 25 20 15 10 5 0
成绩
最终成绩
第1次提交
第2次提交

2. 同学们已经学到了不少知识

比照“计算机科学基础”报告

- “计算机科学是研究计算机以及它们能干什么的一门学科。它研究**抽象计算机**的能力与局限，**真实计算机**的构造与特征，以及用于求解问题的无数**计算机应用**。”
  - 计算机科学涉及**符号**及其操作；
  - 关注多种**抽象**概念的创造和操作；
  - 创造并研究**算法**；
  - 创造各种**人工结构**，尤其是不受物理定律限制的结构；
  - 利用并应对**指数增长**；
  - 探索计算能力的**基本极限**；
  - 关注与人类**智能**相关的复杂的、分析的、理性的活动。

知道概念，能够举例说明其特点

笔记本电脑

有限状态机、图灵机 、
冯诺依曼模型

- “计算机科学是研究计算机以及它们能干什么的一门学科。它研究**抽象计算机**的能力与局限，**真实计算机**的构造与特征，以及用于求解问题的无数**计算机应用**。”
  - 计算机科学涉及**符号**及其操作；
  - 关注多种**抽象**概念的创造和操作；
  - 创造并研究**算法**；
  - 造各种**人工结构**，尤其是不受物理定律限制的结构；
  - 用并应对**指数增长**；
  - 探索计算能力的**基本极限**；
  - 关注与人类**智能**相关的复杂的、分析的、理性的活动。

快速排序程序

各种编程抽象

逻辑推理
命题逻辑
谓词逻辑

斐波那契递归程序
P与NP

图灵机不可计算问题
哥德尔不完备定理

实验的分工配合与关键难点

- 编程入门。在真实计算机上理解应用程序设计与执行
  - 程序设计（编程）的两个特点
    - 将逻辑思维、算法思维、系统思维串成一体（计算思维是交响乐）
    - 满足高德纳测试
      - 我是否理解某个知识点的终极测试，是看我能否向计算机讲清楚

实验的分工配合与关键难点

- 编程入门。在真实计算机上理解应用程序设计与执行
  - 程序设计（编程）的特点
    - 将逻辑思维、算法思维、系统思维串成一体（计算思维是交响乐）
    - 满足高德纳测试
      - 我是否理解某个知识点的终极测试，是看我能否向计算机讲清楚
  - 基本数据类型、数组与循环、切片与递归

实验的分工配合与关键难点

- 编程入门。在真实计算机上理解应用程序设计与执行
- 加法图灵机。用抽象计算机的“汇编语言”实现循环
- 信息隐藏。将文本文件隐藏到图像文件中
  - 单机应用程序如何定位到某个数据的位置，并操作该数据
    - 难点：理解base+index*8+offset 寻址模式如何支持循环，推广到本实验
- 个人作品，创作+实现
  - 网络应用程序如何定位到某个网页元素，并操作该元素
    - 难点：自学网页文档结构（DOM），超链接，getElementById

实验的分工配合与关键难点

- 三个进阶实验（有助教帮助）
  - 斐波那契大数。开发Go程序，10小时内算出F(十亿)
    - 难点：新的系统思维，编程抽象
  - 班级排序计算机。设计真实计算机（班级计算机），实现快速排序
    - 难点：设计班级计算机的组成和指令集，实现递归
  - 哈希查找。开发Go程序，实现在100万个记录中的O(1)查找
    - 暴力查找：O(N)；二分查找：O(logN)；哈希查找O(1)

编程实验涉及高级语言与汇编语言

- 多条指令如何支持循环与数组
  - 在科学计算中，循环和数组往往配套出现
- 斐波那契计算机例子
  - 汇编代码如何忠实反映了循环之变与不变
  - 数组与循环如何配合

```
for i := 2; i < 51; i++ {
   fib[i] = fib[i-1] + fib[i-2]

}
```

```
        MOV 2, R2              // i:=2
Loop:   MOV 0, R1              // label Loop
        ADD M[R0+R2*8-16], R1
        ADD M[R0+R2*8-8], R1
        MOV R1, M[R0+R2*8-0]
        INC R2                 // i++
        CMP 51, R2             // i < 51?
        JL Loop                // if Yes, goto Loop
```

基址索引偏移量寻址模式　天然适配循环

- **address = base + index*8 + offset**
  实际地址 = 基址 + 索引*比例因子 + 偏移量
- 进入for循环，i:=2
  - 基址寄存器R0=24
  - 索引寄存器R2=2
  - 比例因子=8，因为fib[i]是64位整数
  - 赋值语句fib[i] = fib[i-1] + fib[i-2]编译成

    MOV 0, R1

    ADD M[R0+R2*8-16], R1

    ADD M[R0+R2*8-8], R1

    MOV R1, M[R0+R2*8-0]
  - 第一条加法指令实现0+fib[0]

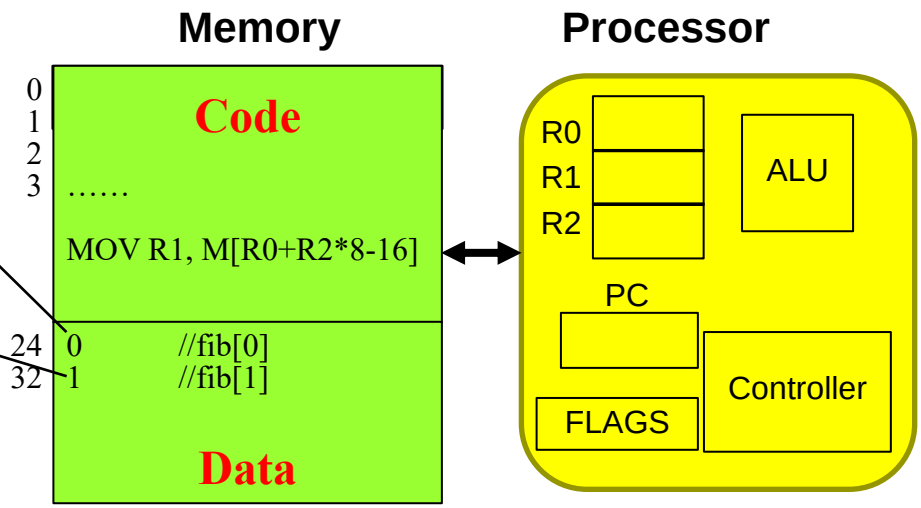
    R1+ M[R0+R2*8-16] → R1，即

    0 + M[24+2*8-16] → R1，即0+fib[0] → R1
  - 第二条加法指令实现0+fib[0]+fib[1]

    R1+ M[R0+R2*8-8] → R1，即

    0 + M[24+2*8-8] → R1，即0+fib[1] → R1

fib[0] = 0	MOV 0, R1
	MOV R1, M[R0] //R0=12 initially
fib[1] = 1	MOV 1, R1
	MOV R1, M[R0+8]
for i := 2; i < 51; i++ {	MOV 2, R2　// i:=2
fib[i] = fib[i-1] + fib[i-2]	MOV 0, R1　// label Loop
	ADD M[R0+R2*8-16], R1
	ADD M[R0+R2*8-8], R1
	MOV R1, M[R0+R2*8-0]
	INC R2　// i++
	CMP 51, R2　// i < 51?
}	JL Loop　// if Yes, goto Loop



Step 1

处理器内容		存储器内容	
寄存器	值	地址	指令
FLAGS		0	MOV 0, R1
PC	2	2	MOV R1, M[R0]
R0	24	4	MOV 1, R1
R1	0	6	MOV R1, M[R0+8]
R2		8	MOV 2, R2
		10 Loop	MOV 0, R1
		12	ADD M[R0+R2*8-16], R1
		14	ADD M[R0+R2*8-8], R1
		16	MOV R1, M[R0+R2*8-0]
		18	INC R2
		20	CMP 51, R2
		22	JL Loop
		24	//fib[0]
		32	//fib[1]
		40	//fib[2]

Step 2

处理器内容		存储器内容	
寄存器	值	地址	指令
FLAGS		0	MOV 0, R1
PC	4	2	MOV R1, M[R0]
R0	24	4	MOV 1, R1
R1	0	6	MOV R1, M[R0+8]
R2		8	MOV 2, R2
		10 Loop	MOV 0, R1
		12	ADD M[R0+R2*8-16], R1
		14	ADD M[R0+R2*8-8], R1
		16	MOV R1, M[R0+R2*8-0]
		18	INC R2
		20	CMP 51, R2
		22	JL Loop
		24	0
		32	
		40	

Step 3

处理器内容		存储器内容	
寄存器	值	地址	指令
FLAGS		0	MOV 0, R1
PC	6	2	MOV R1, M[R0]
R0	24	4	MOV 1, R1
R1	1	6	MOV R1, M[R0+8]
R2		8	MOV 2, R2
		10 Loop	MOV 0, R1
		12	ADD M[R0+R2*8-16], R1
		14	ADD M[R0+R2*8-8], R1
		16	MOV R1, M[R0+R2*8-0]
		18	INC R2
		20	CMP 51, R2
		22	JL Loop
		24	0 //fib[0]
		32	//fib[1]
		40	//fib[2]

Step 4

处理器内容		存储器内容	
寄存器	值	地址	指令
FLAGS		0	MOV 0, R1
PC	8	2	MOV R1, M[R0]
R0	24	4	MOV 1, R1
R1	1	6	MOV R1, M[R0+8]
R2		8	MOV 2, R2
		10 Loop	MOV 0, R1
		12	ADD M[R0+R2*8-16], R1
		14	ADD M[R0+R2*8-8], R1
		16	MOV R1, M[R0+R2*8-0]
		18	INC R2
		20	CMP 51, R2
		22	JL Loop
		24	0 //fib[0]
		32	1 //fib[1]
		40	//fib[2]

物理专业XXX同学的主动学习例子

- 不该人做指令执行
  - 开发了一个模拟器，能够自动执行斐波那契计算机指令
- 干得棒，值得鼓励！建议改进成为个人作品

物理专业XXX同学的主动学习例子

- 不该人做指令执行
  - 开发了一个模拟器，能够自动执行斐波那契计算机指令
- 干得棒，值得鼓励！
- 建议改进成为个人作品
  - 不要用框架自动生成代码，自己写全部每一行代码

- 为什么课程安排同学做基本操作？
  - 斐波那契计算机，图灵机加法器，
  - 后面还有时序电路，指令流水线
  - 掌握入门知识：计算机执行程序的基本原理
    - 计算机如何支持数组与循环

幸好Go语言提供了for loop循环抽象

- 初始化: 数组索引从零开始 i = 0
- 重复执行循环体，每次迭代索引加1，直到i≥11

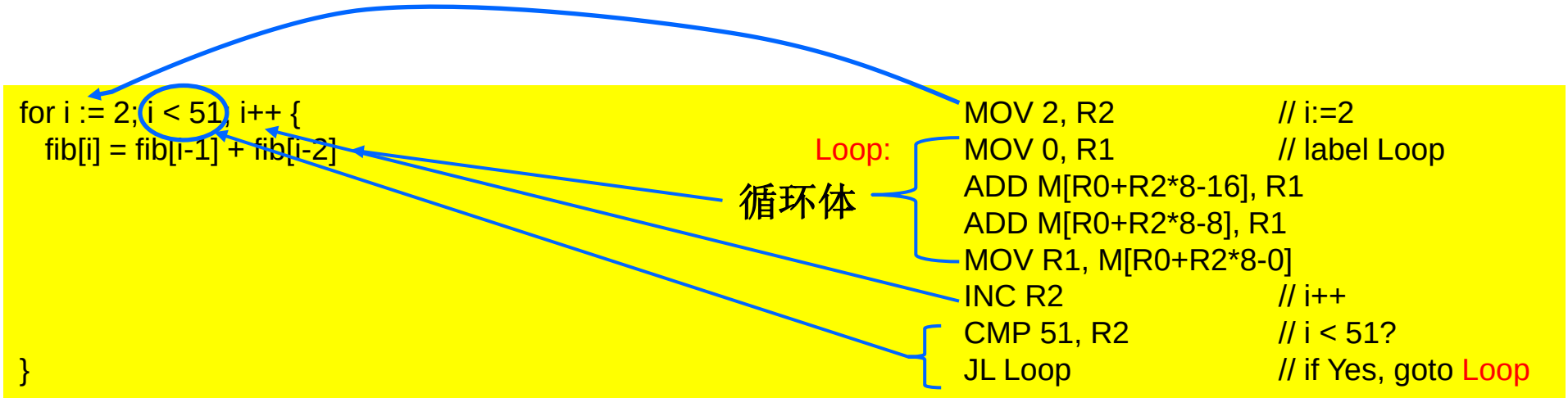
```
package main
import "fmt"
func main() {
   var name string = "Alan Turing"
   sum := 0
   sum = sum + int(name[0])
   sum = sum + int(name[1])
   sum = sum + int(name[2])
   sum = sum + int(name[3])
   sum = sum + int(name[4])
   sum = sum + int(name[5])
   sum = sum + int(name[6])
   sum = sum + int(name[7])
   sum = sum + int(name[8])
   sum = sum + int(name[9])
   sum = sum + int(name[10])
   fmt.Printf("%d\n", sum)
}
```

```
package main
import "fmt"
func main() {
   var name string = "Alan Turing"
   sum := 0
   for i := 0; i < 11; i++ {
      sum = sum + int(name[i])
   }
   fmt.Printf("%d\n", sum)
}
```

循环抽象的诀窍：综合变与不变
不变：for循环结构不变
变：索引值变化，累加值sum变化

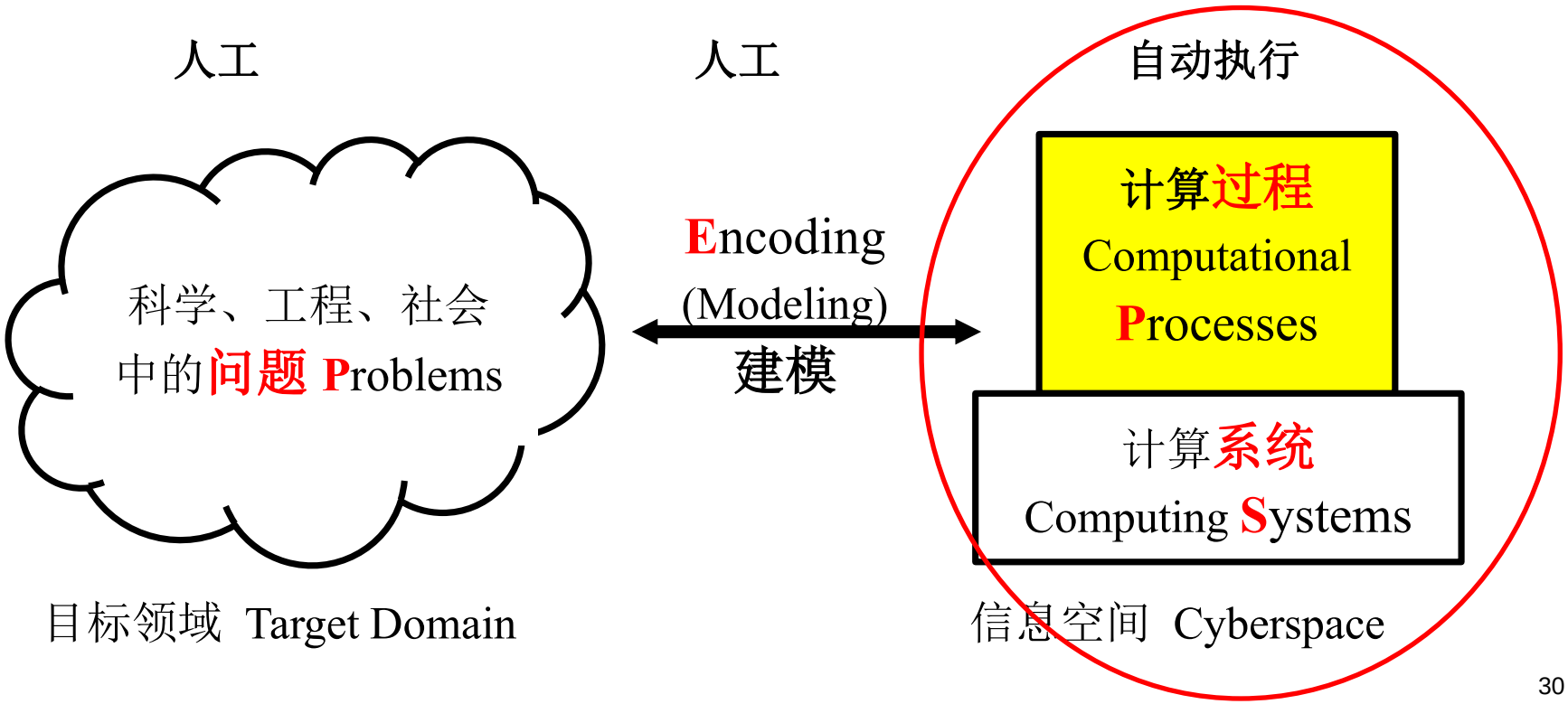
理解重点：多条指令如何支持循环

- 以及数组
  - 在科学计算中，循环和数组往往配套出现
- 注意
  - 汇编代码如何忠实反映了循环之变与不变
  - 数组与循环如何配合



3. 计算思维的基本思路：活力解题法（PEPS）

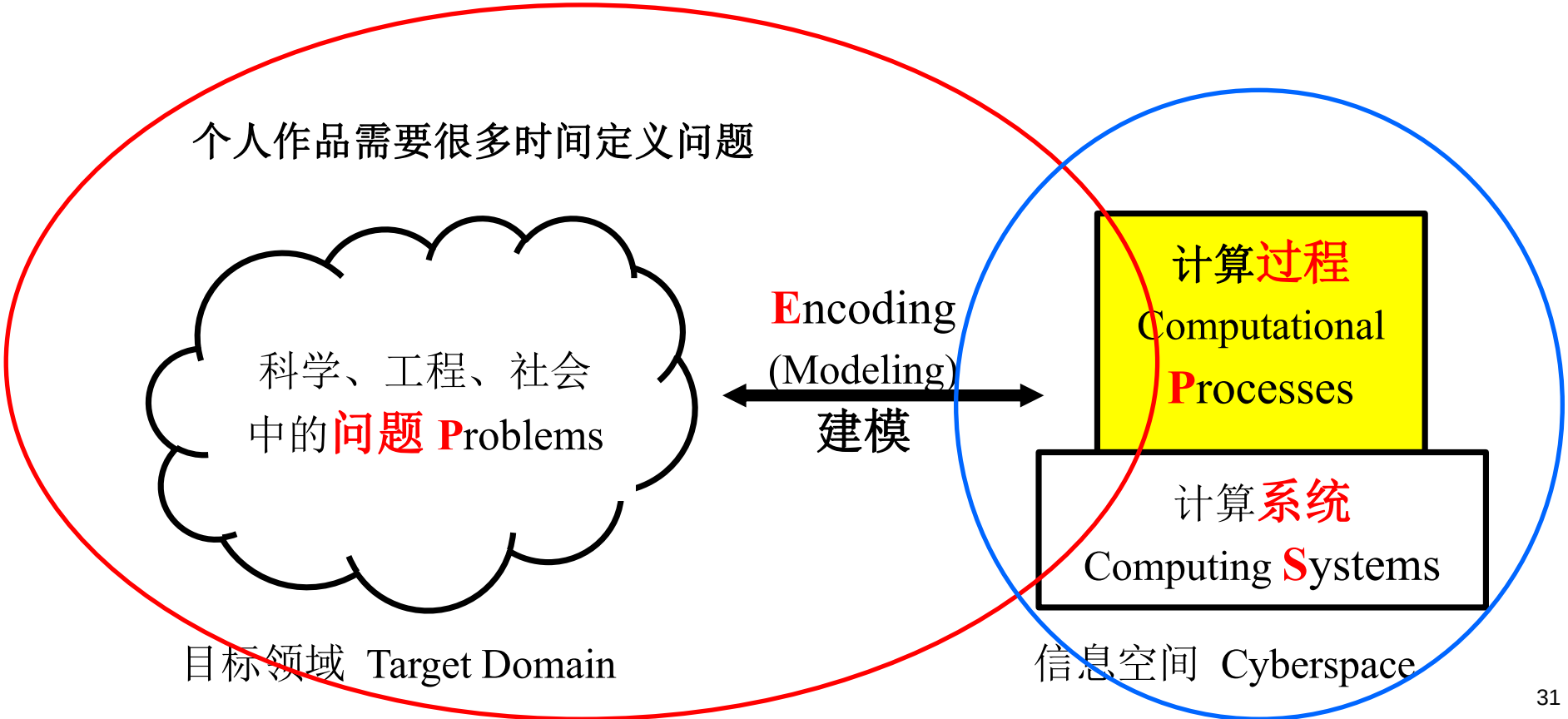
- 定义领域**问题**（**P**roblem in target domain）
- **建模**（**E**ncoding）到信息空间（赛博空间，cyberspace）的计算问题
- 自动执行计算**系统**（**S**ystem）上的计算**过程**（**P**rocess），解决问题
  - 计算系统 = 计算机；算法、程序、状态转移表等刻画计算过程
- 映射回到目标领域



实践中的PEPS（问题-建模-计算过程-系统）

- **P**roblem, 目标领域中的待解问题
- **E**ncoding, 建模
- Computational **P**rocess, 计算过程
- Computing **S**ystem, 计算系统

精力占比	P	E	P	S
斐波那契	1%	1%	78%	20%
个人作品	40%	10%	30%	20%



个人作品实例——创造性表达

- 每个同学创建自己的动态网页作品（**演示**）
  - 猫猫指挥家
  - 绮
- 提升学习能力

2020级物理系李思悦同学创建“猫猫指挥家”
花了三天，50%构思、设计，50%编码
155行代码

2021级生物系董可昕同学创建“绮”
“绮 2.0”版172行代码
“绮 3.0”版215行代码

4. 逻辑思维

- 关注计算过程的比特精准的正确性与通用性

- 课程讨论了
  - 布尔逻辑
  - 图灵机、图灵可计算性
  - 图灵机的通用性与局限
  - 哥德尔不完备定理

- 能够说明每个概念，并给出具体例子

4.1 布尔逻辑

- 比特精准的最基本体现
- 命题逻辑、布尔函数、系统思维中的组合电路
  - 公理系统（从代数角度理解）
    - 元素：常数（0,1）、变量；布尔表达式
    - 算子：基本操作（与、或、非）
    - 公理（布尔逻辑的基本性质）
    - 推理规则
  - 用真值表辅助
- 谓词逻辑
  - 多了断言和量词（**全称量词∀**和**存在量词∃**）
    - 断言是会传回“真”或“假”的函数
      - 任何自然数，要么它是偶数，要么它加1后为偶数。
      - $\forall n\ [\mathrm{Even}(n) \vee \mathrm{Even}(n+1)]$

布尔函数

- 系统思维：组合电路能实现任意布尔函数吗？
  - 组合电路：由与、或、非门电路组合而成的逻辑电路

- n-输入-1输出的布尔函数是数学函数 $f:\{0,1\}^n \to \{0,1\}$
  - 如，$f(x_1, x_2, \cdots, x_n) = (\vee_{i=1}^{n-1} x_i) \oplus x_n$
- 两个布尔函数（布尔表达式）等价：有相同的真值表
  - 类似：$f(x,y) = x^2 - y^2$和$g(x,y) = (x+y)(x-y)$是相同的多项式
- 一个变量x的布尔函数
  - 一共四个

x	y
0	**0**
1	**0**

$y = 0$

x	y
0	**1**
1	**1**

$y = 1$

x	y
0	**0**
1	**1**

$y = x$

x	y
0	**1**
1	**0**

$y = \overline{x}$

任意布尔函数有范式（唯一性）

- 合取范式(conjunctive normal form, CNF)
  - $f(x_1,\ldots,x_n)= Q_1 \wedge Q_2 \wedge Q_3 \ldots \wedge Q_m$
  - 其中: $Q_i = l_1 \vee l_2 \vee \ldots \vee l_n$，　$l_j = x_j$ 或 $\neg x_j$
- 析取范式(disjunctive normal form, DNF)
  - $f(x_1,\ldots,x_n)= Q_1 \vee Q_2 \vee Q_3 \ldots \vee Q_m$
  - 其中: $Q_i = l_1 \wedge l_2 \wedge \ldots \wedge l_n$，　$l_j = x_j$ 或 $\neg x_j$
- 写出2个变量的全部函数的析取范式（一共有$2^{2^n}=2^4= 16$个）
  - 【注意：另一套等价的与、或、非记号；计算系统思维常用】
  - $y = 0$ 【注】　　$y = \overline{x_1}\cdot\overline{x_2} + \overline{x_1}\cdot x_2 + x_1\cdot\overline{x_2} + x_1\cdot x_2 = 1$
  - $y = x_1\cdot\overline{x_2} + x_1\cdot x_2 = x_1$　　$y = \overline{x_1}\cdot x_2 + x_1\cdot\overline{x_2} + x_1\cdot x_2 = x_1 + x_2$ ……

x_1	x_2	y
0	0	**0**
0	1	**0**
1	0	**0**
1	1	**0**

x_1	x_2	y
0	0	**1**
0	1	**1**
1	0	**1**
1	1	**1**

x_1	x_2	y
0	0	**0**
0	1	**0**
1	0	**1**
1	1	**1**

x_1	x_2	y
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**1**

布尔代数 Boolean Algebra

- 另一种理解布尔函数的方法
  - 从布尔代数(L, +, •, ¬, 0, 1)获得**布尔表达式**
  - 记L为布尔表达式集合
    - $0, 1, x_1, \ldots, x_n \in$ L; 初始假定
    - If $x, y \in$ L, then $\neg x, x \bullet y, x + y \in$ L. 递归直至达到闭包
      - 且满足如下公理（用下列公理消除等价表达式）

1.	Associativity:	$(x \bullet y) \bullet z = x \bullet (y \bullet z)$, $(x + y) + z = x + (y + z)$	=2•3•5 =2+3+5	**结合律**
2.	Commutativity:	$x \bullet y = y \bullet x$, $x + y = y + x$	=2•3 =2+3	**交换律**
3.	Distributivity:	$(x + y) \bullet z = (x \bullet z) + (y \bullet z)$, $(x \bullet y) + z = (x + z) \bullet (y + z)$	(2+3)•5 =2•5+3•5 (2•3)+5 ≠(2+5)•(3+5)	**分配率**
4.	Identity: Annihilator:	$x + 0 = x$, $x \bullet 1 = x$ $x \bullet 0 = 0$, $x + 1 = 1$	2+0=2, 2•1=2 2•0=0, 2+1≠1	**有界律**
5.	Idempotence:	$x \bullet x = x$, $x + x = x$	2•2 ≠ 2, 2+2 ≠ 2	**幂等律**
6.	Absorption:	$(x \bullet y) + x = x$, $(x + y) \bullet x = x$	(2•3)+2≠2, (2+3)•2≠2	**吸收律**
7.	Complementation:	$x + \neg x = 1$, $x \bullet \neg x = 0$		**互补律**

中学代数，设
x, y, z = 2, 3, 5

(排中律)
非真既假

2个变量的布尔表达式

- 第一轮。常数和变量及其非，共6个表达式：

$$0, 1, x_1, x_2, \overline{x_1}, \overline{x_2}$$

- 第二轮。应用与或非到第一轮结果，使用公理整理，共8个新表达式：

$$\overline{x_1} + \overline{x_2}, \quad \overline{x_1} + x_2, \quad x_1 + \overline{x_2}, \quad x_1 + x_2$$
$$\overline{x_1} \cdot \overline{x_2}, \quad \overline{x_1} \cdot x_2, \quad x_1 \cdot \overline{x_2}, \quad x_1 \cdot x_2$$

- 第三轮。应用与或非到第二轮表达式，使用公理整理，共2个新表达式：

$$\overline{x_1} \cdot \overline{x_2} + x_1 \cdot x_2, \quad \overline{x_1} \cdot x_2 + x_1 \cdot \overline{x_2}$$

- 第四轮。应用与或非到第三轮表达式，使用公理整理，共0个新表达式。达到闭包。

计算机科学很幸运地选择了布尔逻辑

- n个变量的布尔函数（布尔表达式）有多少个？
  - n=0：2个（0和1）
  - n=1： 4个（1，0，x，$\neg x$）
  - n=n： 2^{2^n}。
- n个变量的克莱因表达式有多少个？
  - 弱化排中律
  - 不是3^{3^n}；尚未找到闭合公式，但知道 $< 2^{3^n}$

任一布尔函数都有
等价的布尔表达式
任一布尔表达式都有
等价的布尔函数
等价：有相同真值表

n	布尔函数个数 2^{2^n}	布尔表达式个数 2^{2^n}	克莱因表达式个数	克莱因函数个数 3^{3^n}
1	$2^{2^1}=4$	$2^{2^1}=4$	6	$3^{3^1}=27$
2	$2^{2^2}=16$	$2^{2^2}=16$	84	$3^{3^2}=3^9$
3	$2^{2^3}=256$	$2^{2^3}=256$	43918	$3^{3^3}=3^{27}$
4	$2^{2^4}=65536$	$2^{2^4}=65536$	160297985276	$3^{3^4}=3^{81}$

课堂小测验

姓名：　　学号：

我们已经知道N-输入-1-输出的布尔函数一共有2^{2^N}个。

每个布尔函数有唯一的真值表

请问：N-输入-2-输出的布尔函数一共有多少个？（）

A. $2\times(2^{2^N})$个

B. $(2\times 2)^{2^N}$个

C. $2^{(2\times 2)^N}$个

D. $2^{2^{(2\times N)}}$个

4.2 建模需要严谨性

- 给定下列两个论据，能推出结论是成立的吗？
  - 论据1：所有上过计科导课的同学都精通Go语言。
  - 论据2：某些精通Go语言的同学可成为下一年计科导课助教。
  - 结论：有些上过计科导的同学可成为下一年计科导课助教。

建模需要严谨性

- 给定下列两个论据，能推出结论是成立的吗？
  - 论据1：所有上过计科导课的同学都精通Go语言。
  - 论据2：某些精通Go语言的同学可成为下一年计科导课助教。
  - 结论：有些上过计科导的同学可成为下一年计科导课助教。
- 定义下列断言记号
  - $\mathrm{CS101}(x)$：同学x上过计科导
  - $\mathrm{MasterGo}(x)$：同学x精通Go语言
  - $\mathrm{TA}(x)$：同学x可成为下一年计科导课助教

建模需要严谨性

- 给定下列两个论据，能推出结论是成立的吗？
  - 论据1：所有上过计科导课的同学都精通Go语言。
  - 论据2：某些精通Go语言的同学可成为下一年计科导课助教。
  - 结论：有些上过计科导的同学可成为下一年计科导课助教。
- 定义下列断言记号
  - $\mathrm{CS101}(x)$：同学x上过计科导
  - $\mathrm{MasterGo}(x)$：同学x精通Go语言
  - $\mathrm{TA}(x)$：同学x可成为下一年计科导课助教

错误的建模

$\forall x[\mathrm{MasterGo}(x)]$

$\exists x[\mathrm{MasterGo}(x) \rightarrow \mathrm{TA}(x)]$

$\exists x[\mathrm{CS101}(x) \rightarrow \mathrm{TA}(x)]$

论据2的谓词表达式应是$\exists x[\mathrm{MasterGo}(x) \wedge \mathrm{TA}(x)]$

$\mathrm{MasterGo}(x)$	$\mathrm{TA}(x)$	$\mathrm{MasterGo}(x) \rightarrow \mathrm{TA}(x)$	$\mathrm{MasterGo}(x) \wedge \mathrm{TA}(x)$
0	0	1	0
0	1	1	0
1	0	0	0
1	1	1	1

建模需要严谨性

- 给定下列两个论据，能推出结论是成立的吗？
  - 论据1：所有上过计科导课的同学都精通Go语言。
  - 论据2：某些精通Go语言的同学可成为下一年计科导课助教。
  - 结论：有些上过计科导的同学可成为下一年计科导课助教。

错误的建模

$\forall x[\text{MasterGo}(x)]$

$\exists x[\text{MasterGo}(x) \rightarrow \text{TA}(x)]$

$\exists x[\text{CS101}(x) \rightarrow \text{TA}(x)]$

- 严谨地构建谓词表达式、使用推理规则
  - 谓词逻辑往往不关心原初论据本身的正确性。即，假设给定论据是正确的。
  - 定义下列断言记号
    - $\text{CS101}(x)$：同学x上过计科导
    - $\text{MasterGo}(x)$：同学x精通Go语言
    - $\text{TA}(x)$：同学x可成为下一年计科导课助教

论据1：$\forall x[\text{CS101}(x) \rightarrow \text{MasterGo}(x)]$

论据2：$\exists x[\text{MasterGo}(x) \wedge \text{TA}(x)]$

结论：　$\exists x[\text{CS101}(x) \wedge \text{TA}(x)]$

感觉结论不对，试找反例

结论的非：　$\neg \exists x[\text{CS101}(x) \wedge \text{TA}(x)]$

等价于　　$\forall x[\neg\text{CS101}(x) \vee \neg\text{TA}(x)]$

只要找到一个同学集及其映射，满足论据，且使得每个成员x

$\neg\text{CS101}(x)$

或　　$\neg\text{TA}(x)$

建模需要严谨性

- 给定下列两个论据，能推出结论是成立的吗？
  - 论据1：所有上过计科导课的同学都精通Go语言。
  - 论据2：某些精通Go语言的同学可成为下一年计科导课助教。
  - 结论：有些上过计科导的同学可成为下一年计科导课助教。

错误的建模

$\forall x[\text{MasterGo}(x)]$

$\exists x[\text{MasterGo}(x) \rightarrow \text{TA}(x)]$

$\exists x[\text{CS101}(x) \rightarrow \text{TA}(x)]$

- 严谨地构建谓词表达式、使用推理规则
  - 谓词逻辑往往不关心原初论据本身的正确性。即，假设给定论据是正确的。
  - 定义下列断言记号
    - $\text{CS101}(x)$：同学x上过计科导
    - $\text{MasterGo}(x)$：同学x精通Go语言
    - $\text{TA}(x)$：同学x可成为下一年计科导课助教

论据1：$\forall x[\text{CS101}(x) \rightarrow \text{MasterGo}(x)]$

论据2：$\exists x[\text{MasterGo}(x) \wedge \text{TA}(x)]$

结论：　$\exists x[\text{CS101}(x) \wedge \text{TA}(x)]$

感觉结论不对，试找反例

结论的非：　$\neg\, \exists x[\text{CS101}(x) \wedge \text{TA}(x)]$

等价于　　$\forall x[\neg\text{CS101}(x) \vee \neg\text{TA}(x)]$

只要找到一个同学集及其映射，满足论据，且使得每个成员x

$\neg\text{CS101}(x)$

或　　$\neg\text{TA}(x)$

- 结论不对，找简单反例
- 设某年级只有两名同学a, b，其映射如下
  - 同学a：$\text{CS101}(a) = \text{MasterGo}(a) = \mathit{True}, \text{TA}(a) = \mathit{False}$　　$\neg\text{TA}(a)$
  - 同学b：$\text{CS101}(b) = \mathit{False}$，$\text{MasterGo}(b) = \text{TA}(b) = \mathit{True}$　　$\neg\text{CS101}(b)$
  - 容易验证论据1和2都成立，但结论不成立

4.3 图灵机的通用性和局限性

- **图灵机能够求解任意可计算问题**
- **邱奇-图灵论题（Church-Turing Hypothesis）**
  - 任意抽象计算机与图灵机等价
    - （足够大存储器的）冯诺依曼计算机与图灵机等价
- **存在不可计算问题**
  - 例如：停机问题

4.3 图灵机的通用性和局限性

- **邱奇-图灵论题（Church-Turing Hypothesis）**
  - 任意抽象计算机与图灵机等价
    - （足够大存储器的）冯诺依曼计算机与图灵机等价
  - 可计算性意义上等价。
  - 实用意义上不等价。冯诺依曼机是桥接模型，图灵机不是。
    - 图灵机加法非常繁琐

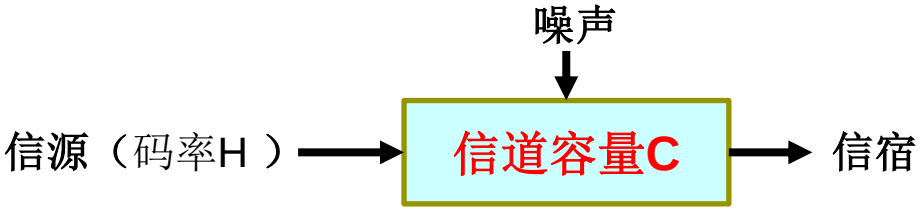
4.4 什么是构造性?

Acu-Exams-CP

- Effectiveness = Constructiveness + Finiteness

  （广义）构造性 = （狭义）构造性 + 有穷性

- 信息论中的香农第二定理是存在性定理，不是构造性定理

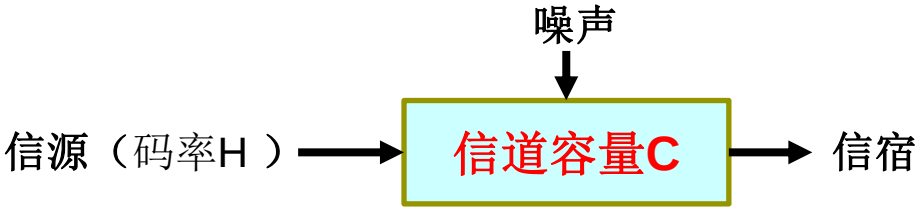


假如H<=C，存在编码，
实现无错消息传递

什么是狭义构造性?

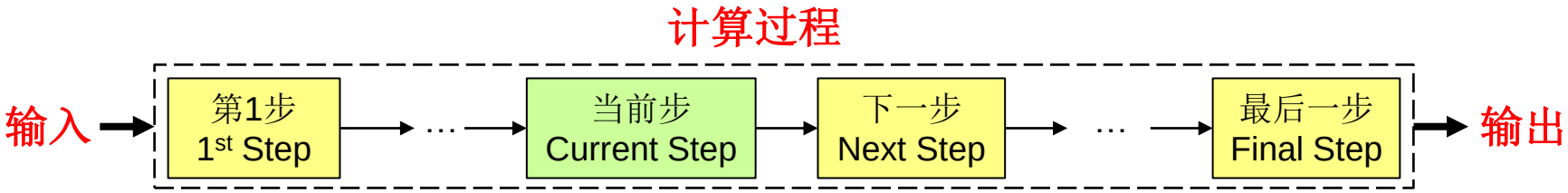
- Effectiveness = Constructiveness + Finiteness
  （广义）构造性 = （狭义）构造性 + 有穷性

- 信息论中的香农第二定理是存在性定理，不是构造性定理



假如H<=C，存在编码，
实现无错消息传递

- 计算过程是构造性过程，能够一步一步算出（构造出）结果



计算系统

什么是有穷性?

- Effectiveness = Constructiveness + Finiteness

（广义）构造性 = （狭义）构造性 + 有穷性

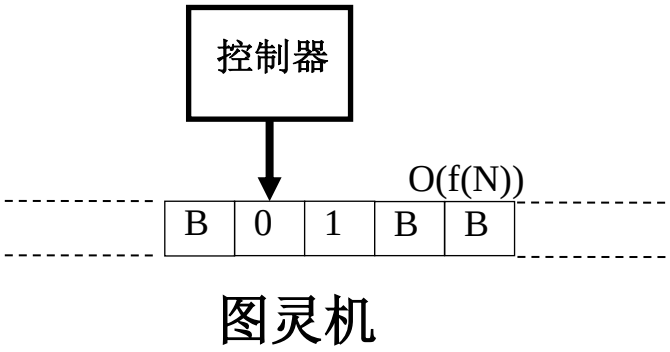
- 两类有穷性
  - 需要的资源量与问题规模N有关，是O(f(N))；
  - 需要的资源量与问题规模N无关，是O(1)
  - 其中，问题规模N可以任意大，但不是无穷大

什么是有穷性?

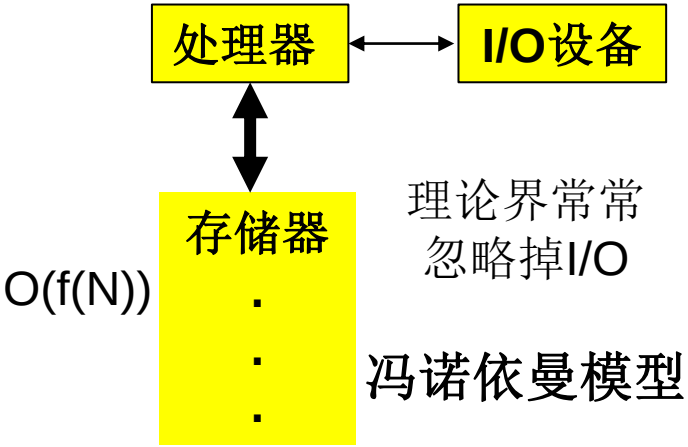
- Effectiveness = Constructiveness + Finiteness

（广义）构造性 =（狭义）构造性 + 有穷性

- 两类有穷性：与问题规模N有关O(f(N))；与问题规模N无关O(1)
  - 存储有穷性：与问题规模有关，空间复杂度　　O(f(N))，N任意大
    - 笔记本电脑，与图灵机**不等价**，因为有限存储容量　　O(1)
    - （足够大存储容量的）笔记本电脑，与图灵机等价　　O(f(N))



图灵机



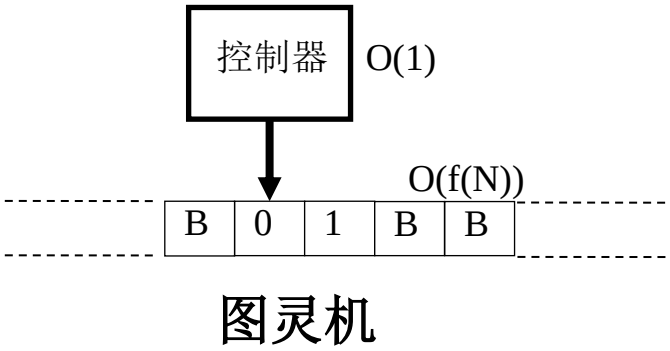
冯诺依曼模型

什么是有穷性?

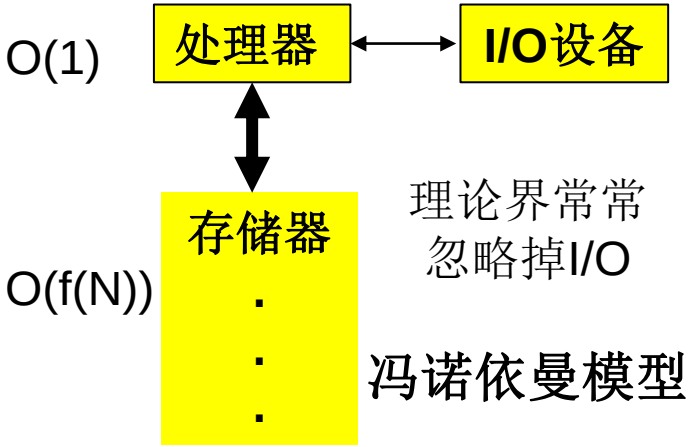
- Effectiveness = Constructiveness + Finiteness

（广义）构造性 =（狭义）构造性 + 有穷性

- 两类有穷性：与问题规模N有关O(f(N))；与问题规模N无关O(1)
  - 存储有穷性：与问题规模有关，空间复杂度 O(f(N))，N任意大
    - 笔记本电脑，与图灵机**不等价**，因为有限存储容量 O(1)
    - （足够大存储容量的）笔记本电脑，与图灵机等价 O(f(N))
  - 程序有穷性：程序行数与问题规模无关 O(1)
    - 用有限行程序表达计算过程
      - Ada运算流程序→控制流程序
    - 任何图灵机的状态转移表只有有限行
      - 加法图灵机：几十行



图灵机



理论界常常忽略掉I/O

冯诺依曼模型

4.5 什么是图灵可计算性?

- 很多理论和现实问题都是图灵可计算问题

- 计算机科学只能用于研究图灵可计算问题吗?
- 不是!

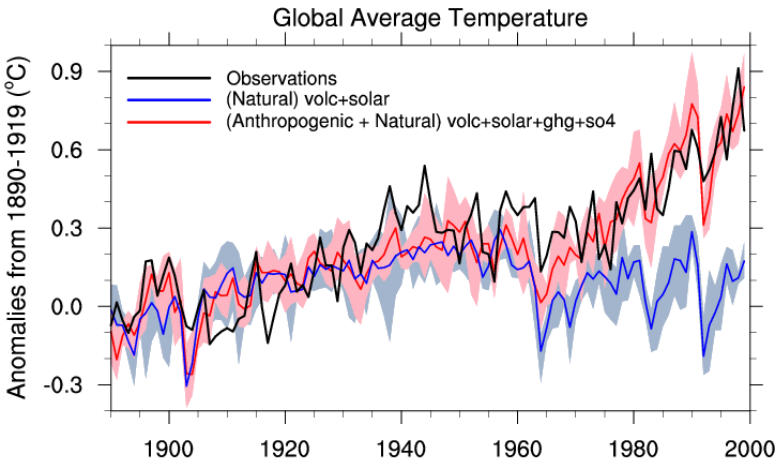
- 遇到非图灵可计算问题怎么办?

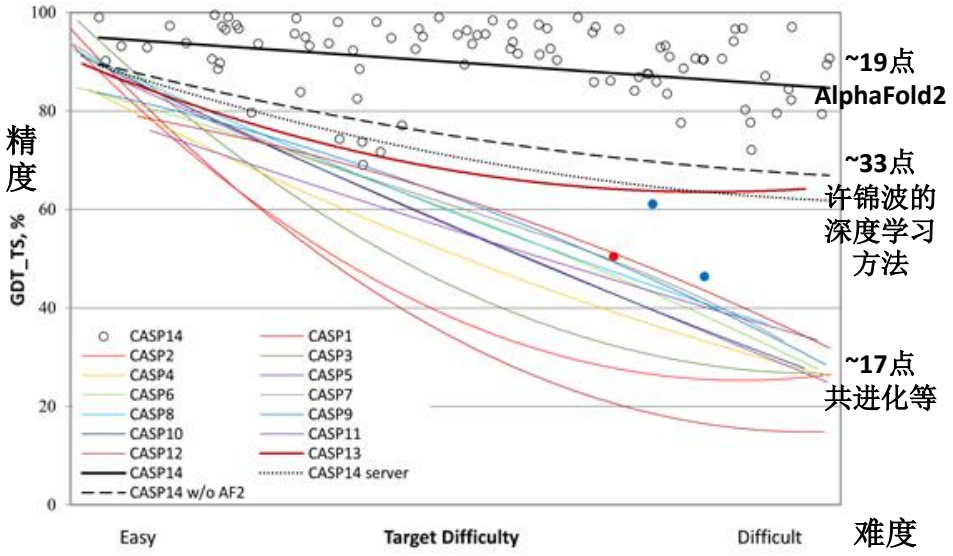
什么是图灵可计算问题？

- 从给定问题，能够设计图灵机，从输入求出期望的输出
  - 针对任意输入，图灵机总会停机，产生正确输出
    - 加法是图灵可计算问题　　3+2 = 5

什么是图灵可计算问题？

- 从给定问题，能够设计图灵机，从输入求出期望的输出
  - 针对任意输入，图灵机总会停机，产生正确输出
    - 加法是图灵可计算问题　　3+2 = 5
- 下述问题是图灵可计算的吗？（不能判断输出是否正确）
  - 气候变化模拟、天气预报
  - 蛋白质结构预测





来自ICPP报告

来自John Moult团队的CASP网站https://predictioncenter.org/

什么是图灵可计算问题？

- 从给定问题，能够设计图灵机，从输入求出期望的输出
  - 针对任意输入，图灵机总会停机，产生正确输出
  - 加法是图灵可计算问题　　3+2 = 5

- 下述问题是图灵可计算的吗？
  - 天气预报
  - 气候变化模拟
  - 蛋白质结构预测

- 什么是万维网可计算问题？
- 什么是人机物系统可计算问题？
- 很多人机物智能计算问题可能是非图灵可计算的

什么是非图灵可计算问题?

- 我们已经遇到几类非图灵可计算问题
  - 正确结果不定的问题（？）
  - 悖论（ill-posed problems）
    - 感兴趣的同学参见数学中的希尔伯特第20问题进展
  - 停机问题（Turing incomputable problems）
  - 不完备问题（Incomplete problems）

- 常用方法
  - 重新定义问题
  - 重新建模
  - 利用人+机+物的分工协作

克雷数学研究所2000年七个问题

https://en.wikipedia.org/wiki/Millennium_Prize_Problems

千禧年大奖难题

- **P与NP问题**
- 霍奇猜想
- 庞加莱猜想（已证明）
- 黎曼猜想
- 杨-米尔斯存在性与质量间隙
- 纳维-斯托克斯存在性与光滑性
- 贝赫和斯维讷通-戴尔猜想

1900年希尔伯特23个问题

https://zh.wikipedia.org/wiki/希尔伯特的23个问题

第1题	**连续统假设**	部分解决	1963年美国数学家保罗·柯恩以力迫法证明连续统假设不能由策梅洛-弗兰克尔集合论（无论是否含选择公理）推导。也就是说，连续统假设成立与否无法由ZF/ZFC确定。
第2题	**算术公理之相容性**	部分解决	库尔特·哥德尔在1931年证明了哥德尔不完备定理，但此定理是否已回答了希尔伯特的原始问题，数学界没有共识。
第3题	两四面体有相同体积之证明法	已解决	**答案：否。**1900年，希尔伯特的学生马克斯·德恩以一反例证明了是不可以的。
第4题	所有度量空间所有线段为测地线	太隐晦	希尔伯特对于这个问题的定义过于含糊。
第5题	所有连续群是否皆为可微群	已解决	1953年日本数学家山边英彦证明在无“小的子群”情况下，答案是肯定的[1]；但此定理是否已回答了希尔伯特的原始问题，数学界仍有争论。
第6题	**公理化物理**	部分解决	希尔伯特后来对这个问题进一步解释，而他自己也进一步研究这个问题。柯尔莫哥洛夫对此也有贡献。然而，尽管公理化已经开始渗透到物理当中，量子力学中仍有至今不能逻辑自洽的部分（如量子场论），故该问题未完全解决。
第7题	若b是无理数、a是除0、1之外的代数数，那么a^b是否超越数	已解决	**答案：是。**分别于1934年、1935年由苏联数学家亚历山大·格尔丰德与德国数学家特奥多尔·施耐德独立地解决。
第8题	黎曼猜想及哥德巴赫猜想和孪生素数猜想	未解决	虽然分别有比较重要的突破和被解决的弱化情况，三个问题均仍未被解决。
第9题	任意代数数域的一般互反律	部分解决	1927年德国的埃米尔·阿廷证明在阿贝尔扩张的情况下答案是肯定的；此外的情况则尚未证明。
第10题	**丢番图方程可解性**	已解决	**答案：否。**1970年由苏联数学家尤里·马季亚谢维奇证明。
第11题	代数系数之二次形式	部分解决	有理数的部分由哈塞于1923年解决。
第12题	一般代数数域的阿贝尔扩张	未解决	埃里希·赫克于1912年用希尔伯特模形式研究了实二次域的情形。虚二次域的情形用复乘复乘理论已基本解决。一般情况下则尚未解决。
第13题	以二元函数解任意七次方程	部分解决	1957年苏联数学家柯尔莫哥洛夫和弗拉基米尔·阿诺尔德证明对于单值解析函数，答案是否定的；然而希尔伯特原本可能希望证明的是代数函数的情形，因此该问题未获得完全解答。
第14题	证明一些函数完全系统之有限性	已解决	**答案：否。**1962年日本人永田雅宜提出反例。
第15题	舒伯特演算之严格基础	部分解决	一部分在1938年由范德瓦登得到严谨的证明。
第16题	代数曲线及表面之拓扑结构	未解决	此问题进展缓慢，即使对于度为8的代数曲线也没有证明。
第17题	把有理函数写成平方和分式	已解决	**答案：是。**1927年埃米尔·阿廷解决此问题，并提出实封闭域。[2][3]
第18题	**球体最紧密的排列**	已解决	1911年比伯巴赫做出“n维欧氏几何空间只允许有限多种两两不等价的空间群”；莱因哈特证明不规则多面体亦可填满空间；托马斯·黑尔斯于1998年提出了初步证明，并于2014年**用计算机完成了开普勒猜想的形式化证明**，证明球体最紧密的排列是面心立方和六方最密两种方式。
第19题	拉格朗日系统之解是否皆可解析	已解决	**答案：是。**1956年至1958年Ennio de Giorgi和约翰·福布斯·纳什分别用不同方法证明。
第20题	**所有边值问题是否都有解**	已解决	实际上**工程和科研中遇到的边值问题**都是适定的，因而都可以确定是否有解。[4]
第21题	证明有线性微分方程有给定的单值群	已解决	此问题的答案取决于问题的表述：部分情况下是肯定的，部分情况下则是否定的。
第22题	将解析关系以自守函数一致化	部分解决	1904年由保罗·克伯和庞加莱取得部分解决。详见单值化定理。
第23题	**变分法的长远发展**	开放性问题	包括希尔伯特本人、昂利·勒贝格、雅克·阿达马等数学家皆投身于此。理查德·贝尔曼提出的**动态规划**可作为变分法的替代。

哥德尔不完备定理的一个实例

- 哥德尔不完备定理：在任何包含初等算术的数学系统中，存在真但不可证的命题。其中，初等算术=皮亚诺算术

皮亚诺算术五公理
(引自大不列颠百科全书)

1. Zero is a natural number.
   零是自然数
2. Every natural number has a successor in the natural numbers.
   任意自然数有一后继自然数
3. Zero is not the successor of any natural number.
   零不是任何自然数的后继
4. If the successor of two natural numbers is the same, then the two original numbers are the same.
   后继相同的两个自然数相等
5. If a set contains zero and the successor of every number is in the set, then the set contains the natural numbers. 归纳公理

哥德尔不完备定理的一个实例

- 哥德尔不完备定理：在任何包含初等算术的数学系统中，存在真但不可证的命题。（1931）
  - 初等算术=皮亚诺算术
- **哪个命题？**

皮亚诺算术五公理
(引自大不列颠百科全书)

1. Zero is a natural number.
   零是自然数
2. Every natural number has a successor in the natural numbers.
   任意自然数有一后继自然数
3. Zero is not the successor of any natural number.
   零不是任何自然数的后继
4. If the successor of two natural numbers is the same, then the two original numbers are the same.
   后继相同的两个自然数相等
5. If a set contains zero and the successor of every number is in the set, then the set contains the natural numbers. 归纳公理

哥德尔不完备定理的一个实例

- 哥德尔不完备定理：在任何包含初等算术的数学系统中，存在真但不可证的命题。
- **哪个命题？**
  - 古德斯坦定理：古德斯坦序列趋近0
    - 回想斐波那契数的递归定义。递推公式也可以涌现复杂现象。

$$(n)_0 = n$$
$$(n)_1 = R_2(n) - 1$$
$$(n)_2 = R_3\big((n)_1\big) - 1$$
$$\cdots$$
$$(n)_{k+1} = \begin{cases} R_{k+2}\big((n)_k\big) - 1 & \text{if } (n)_k > 0 \\ 0 & \text{if } (n)_k = 0 \end{cases}$$

$R_b(n)$ replaces every b with b+1 in the hereditary base-b representation of n.

古德斯坦序列快速增长，但趋近0！

设 $n = 266$

$$(266)_0 = 2^{2^{2+1}} + 2^{2+1} + 2 = 266$$
$$(266)_1 = 3^{3^{3+1}} + 3^{3+1} + 2 \approx 4.4 \times 10^{38}$$
$$(266)_2 = 4^{4^{4+1}} + 4^{4+1} + 1 \approx 3.2 \times 10^{616}$$
$$\cdots$$
$$(266)_{G(266)} = 0$$

Goodstein sequence for n=4
郭泓锐的结果（计算机，2017）

- 提出了新算法，指数性降低计算复杂度
  - 1400行Go程序，在单节点服务器上算了13小时
- 计算结果如下，注意$G(4) = 3\cdot 2^{402653211}-2$，最大数所在下标$k= 3\cdot 2^{402653209}$

$(4)_0 =$ 4
$(4)_1 =$ 26
$(4)_2 =$ 41
$(4)_3 =$ 60
$(4)_4 =$ 83

$$最大数(4)_K = 3.447540401546 31\ldots \times 10^{121210695}$$

.....

$(4)_{k-2} =$ 这五个最大的数有下面两个性质：字长是121210695十进制位，最高位1000个十进制数字是

$(4)_{k-1} =$ 3447540401546310082868194979805754978478874937901486829482582471181371747989862435944126537723138836357706343870670981471373770123119725827117104237084886899557319167763456466003661752256536556670766073663822155027872496625257503330885348667848633411493190314615269655969736866492160948225290436847365886709876147562092042005868664973311829175856332138120219517198418201812335339301056271348711228774295067529019486998025111083607801145279169927216822914248107894561933408544103589430851495052430471521491596915665031176899651610957212217360780656154707158846933785793375188967822229622822797778043761152773386718099235161662127038925419805394792980981939486485522909222878857883887560348367316381270673280675347382769219015375432616565108108081814310923711203313305968399971676957781217795694754003625391588939038853733479876344772723623575017620929937195505529441455741049571737770925493092772866804132388313424521454495162309274452552557713106522747593529930053066060085629731708801300896843377521210695
$(4)_K =$ 2252904368473658867098761475620920420058686649733118291758563321381202195171984182018123353393010562713487112287742950675290194869980251110836078011452791699272168229142481078945619334085441035894308514950524304715214915969156650311768996516109572122173607806561547071588469337857933751889678222296228227977780437611527733867180992351616621270389254198053947929809819394864855229092228788578838875603483673163812706732806753473827692190153754326165651081080818143109237112033133059683999716769577812177956947540036253915889390388537334798763447727236323575017620929937195505529441455741049571737770925493092772866804132388313424521454495162309274452552557713106522747593529930053066060085629731708801300896843377521210695
$(4)_{k+1} =$ 8896782222962282279778043761152773386718099235161662127038925419805394792980981939486485522909222878857883887560348367316381270673280675347382769219015375432616565108108081814310923711203313305968399971676957781217795694754003625391588939038853733479876344772723623575017620929937195505529441455741049571737770925493092772866804132388313424521454495162309274452552557713106522747593529930053066060085629731708801300896843377521210695
$(4)_{k+2} =$ 0552944145574104957173777092549309277286680413238831342452145449516230927445255255771310652274759352993005306606008562973170880130089684337752

.....

$(4)_{G(4)-1} =$ 4
$(4)_{G(4)-1} =$ 3
$(4)_{G(4)-1} =$ 2
$(4)_{G(4)-1} =$ 1
$(4)_{G(4)} =$ 0

哥德尔不完备定理的一个实例

- 哥德尔不完备定理：在任何包含初等算术的数学系统中，存在真但不可证的命题。（1931）
  - 初等算术=皮亚诺算术
- **哪个命题？**
- Goodstein Theorem, 1944
  - Every Goodstein sequence approaches zero.
- Kirby & Paris, 1982
  - 戈德斯坦定理不能在皮亚诺算术中证明.
- Caicedo, 2007
  - Kirby & Paris结论的新证明.

皮亚诺算术五公理
(引自大不列颠百科全书)

1. Zero is a natural number.
   零是自然数
2. Every natural number has a successor in the natural numbers.
   任意自然数有一后继自然数
3. Zero is not the successor of any natural number.
   零不是任何自然数的后继
4. If the successor of two natural numbers is the same, then the two original numbers are the same.
   后继相同的两个自然数相等
5. If a set contains zero and the successor of every number is in the set, then the set contains the natural numbers. 归纳公理

Andrés Eduardo Caicedo, Goodstein's function. Revista Colombiana de Matemáticas 41 (2) (2007), 381–391.

5. 算法思维妙在何处？

- 算法概念的巧妙定义（高德纳五点定义）
  - 简洁、通用、可检查
  - 加法图灵机设计是算法设计，状态转移表满足高德纳定义

算法思维妙在何处？

- 算法概念的巧妙定义（高德纳五点定义）
  - 简洁、通用、可检查
  - 加法图灵机设计是算法设计
- 巧妙的效率度量（复杂度的o，O，Ω，Θ记号表达）
  - 忽略不重要细节，有利于算法分析
  - 快速排序：最坏复杂度$O(n^2)$，平均复杂度$O(n\log n)$

算法思维妙在何处？

- 算法概念的巧妙定义（高德纳五点定义）
  - 简洁、通用、可检查
  - 加法图灵机设计是算法设计
- 巧妙的效率度量（复杂度的o，O，Ω，Θ记号表达）
  - 忽略不重要细节，有利于算法分析
  - 快速排序：最坏复杂度$O(n^2)$，平均复杂度$O(n\log n)$
- 巧妙的算法设计方法（algorithmic paradigms）
  - 分治、动态规划、贪心、随机

算法思维妙在何处？

- 算法概念的巧妙定义（高德纳五点定义）
  - 简洁、通用、可检查
    - 加法图灵机设计是算法设计
- 巧妙的效率度量（复杂度的o，O，Ω，Θ记号表达）
  - 忽略不重要细节，简化算法分析
    - 快速排序：最坏复杂度$O(n^2)$，平均复杂度$O(n\log n)$
- 巧妙的算法设计方法（algorithmic paradigms）
  - 分治、动态规划、贪心、随机
- 巧妙的变奏，适应问题特色
  - 单因素优选法中的0.618法，而不是二分查找法。
    - 复用查询结果
    - 二分法的复杂度是$\log_{\sqrt{2}} n$，$> \log_{1.618} n$

5.1 高德纳的算法定义

一个算法是一组有限条规则，给出求解特定类型问题的操作序列，并具备下列五个特征：

① 有限性：算法在有限步骤之后必然要终止。

② 确定性：算法的每个步骤都必须精确地（严格地和无歧义地）定义。

③ 输入：算法有零个或多个输入，在算法开始或中途动态地给定。

④ 输出：算法有一个或多个输出。

⑤ 能行性：算法的所有操作必须是充分基本的，原则上一个人能够用笔和纸在有限时间内精确地完成它们。

冒泡排序算法

- 输入：待排序的数组A，数组A的长度n。
- 输出：排好序的数组A。
- 算法步骤描述：

```
for i = 1 to n − 1
  for j = 1 to n − i
    if A[j]>A[j+1]
      exchange A[j] with A[j+1].
```

什么不是算法？

- Ada的运算流程序
  - 程序行数应与问题规模无关
- 无穷循环不停机
- 包含下列操作
  - 两个物理线段求余
  - if哥德巴赫猜想为真{…}

注意两种有穷性

- **高德纳的算法定义**

一个算法是一组有限条规则，给出求解特定类型问题的操作序列，并具备下列五个特征：

① 有限性：算法在有限步骤之后必然要终止。

② 确定性：算法的每个步骤都必须精确地（严格地和无歧义地）定义。

③ 输入：算法有零个或多个输入，在算法开始或中途动态地给定。

④ 输出：算法有一个或多个输出。

⑤ 能行性：算法的所有操作必须是充分基本的，原则上一个人能够用笔和纸在有限时间内精确地完成它们。

冒泡排序算法

- 输入：待排序的数组A，数组A的长度n。
- 输出：排好序的数组A。
- 算法步骤描述：

```
for i = 1 to n − 1
  for j = 1 to n − i
    if A[j]>A[j+1]
      exchange A[j] with A[j+1].
```

什么不是算法？

- Ada的运算流程序
  - 程序行数应与问题规模无关
- 无穷循环不停机
- 包含下列操作
  - 两个物理线段求余
  - if哥德巴赫猜想为真{…}

5.2 算法的复杂度分析

- 求解斐波那契数列的递归算法GO代码耗时多少？

```
func fibonacci(n int) int {
        if n == 0 || n == 1 {
                return n
        }
        return fibonacci(n-1) + fibonacci(n-2)
}
```

- 复杂度分析（常用求解递推式的分析方法）
  - T(n) = T(n-1) + T(n-2) = T(n-2)+T(n-3)+T(n-3)+T(n-4)
  - 2*T(n-2) < T(n) < 2*T(n-1)
  - T(n) < 2*T(n-1) < 4*T(n-2) < 8*T(n-3) … < 2^n
  - T(n) > 2*T(n-2) > 4*T(n-4) > 8*T(n-6) … > $2^{n/2}$
  - $2^{n/2}$ < T(n) < 2^n
  - **T(n) = $\Omega(2^{n/2})$，T(n) = $O(2^n)$，指数复杂度！**

小o，大O，Ω，Θ记号

假设：当n足够大

等号是单向的，不同于数学等号

$n^{1.58} = O(n^2)$，$n^{1.58} = O(n^3)$，但$O(n^2) \neq O(n^3)$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$$

- $f(n) = o(g(n))$
  - $n^{1.58} = \mathbf{o}(n^2)$，$n^{1.58} \neq \mathbf{o}(n^{1.58})$，$n^2 \neq \mathbf{o}(n^{1.58})$
- $f(n) = O(g(n))$

  ∃常数$c > 0, f(n) \leq cg(n)$
  - $n^{1.58} = \mathbf{O}(n^2)$，$n^{1.58} = \mathbf{O}(n^{1.58})$，$n^2 \neq \mathbf{O}(n^{1.58})$
- $f(n) = \Omega(g(n))$

  ∃常数$c > 0, f(n) \geq cg(n)$
  - $n^{1.58} \neq \mathbf{\Omega}(n^2)$，$n^{1.58} = \mathbf{\Omega}(n^{1.58})$，$n^2 = \mathbf{\Omega}(n^{1.58})$
- $f(n) = \Theta(g(n))$

  $f(n) = O(g(n))$ 并且 $f(n) = \Omega(g(n))$
  - $n^{1.58} \neq \mathbf{\Theta}(n^2)$，$n^{1.58} = \mathbf{\Theta}(n^{1.58})$，$n^2 \neq \mathbf{\Theta}(n^{1.58})$

小o，大O， Ω，Θ记号

共同假设：当n足够大。g是f的严格上阶、上阶、下阶、等阶

- $f(n) = o(g(n))$ g是f的严格上界（阶） $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$
  - $n^{1.58} = \mathbf{o}(n^2)$， $n^{1.58}$ ？ $\mathbf{o}(n^{1.58})$ ， $n^2 \neq \mathbf{o}(n^{1.58})$
- $f(n) = O(g(n))$ 上界 $\exists$常数$c > 0, f(n) \leq cg(n)$
  - $n^{1.58}$ ？ $\mathbf{O}(n^2)$， $n^{1.58} = \mathbf{O}(n^{1.58})$， $n^2 \neq \mathbf{O}(n^{1.58})$
- $f(n) = \Omega(g(n))$ 下界 $\exists$常数$c > 0, f(n) \geq cg(n)$
  - $n^{1.58}$ ？ $\mathbf{\Omega}(n^2)$， $n^{1.58} = \mathbf{\Omega}(n^{1.58})$， $n^2 = \mathbf{\Omega}(n^{1.58})$
- $f(n) = \Theta(g(n))$ 等阶 $f(n) = O(g(n))$ 并且 $f(n) = \Omega(g(n))$
  - $n^{1.58}$ ？ $\mathbf{\Theta}(n^2)$， $n^{1.58} = \mathbf{\Theta}(n^{1.58})$， $n^2 \neq \mathbf{\Theta}(n^{1.58})$

5.3 NP vs P

- 输入2*n*个数，判断是否可以把这些数等分成两组（每组n个数），使得两组的和相同。
- 是否是NP的？
  - 是！
  - 为什么？
    - 给定结果，即满足题意的分组方式
    - 验证算法：验证每组都是n个数，且两组数的和相等
    - **验证算法是多项式的**（事实上验证算法的复杂度是O(n)）！
- 是否是P的？
  - 目前不知道。如果找到多项式时间算法，则NP=P。

NP vs P

问题改一点点，复杂度可能有本质变化

- 输入2*n*个数，判断是否可以把这些数等分成两组（每组n个数），使得两组的和不相同。
- 是否是NP的？
  - 是！方法类似之前。
- 是否是P的？
  - 是！
  - 只要这2n个数不全相同，则答案为是。为什么？
  - O(n)

5.4 算法思维方法（Algorithmic Paradigms）

- 分治思想
  - 各种排序算法
  - 单因素优选法，好于二分法
  - 大整数乘法
  - 矩阵乘法

5.4 算法思维方法（Algorithmic Paradigms）

- 分治思想
  - 各种排序算法
  - 单因素优选法，好于二分法
  - 大整数乘法
  - 矩阵乘法
  - 分析复杂度的一般递推公式

将规模为n的待解问题
分为a个子问题，
每个子问题规模为n/b；
合并子问题结果的开销为$O(n^d)$。

时间复杂度的递推公式为

$$T(n)=\begin{cases} aT\left(\frac{n}{b}\right)+O(n^d) & n>1 \\ O(1) & n=1 \end{cases}$$

时间复杂度为

$$T(n)=\begin{cases} O(n^d) & d>\log_b a \\ O(n^d \log n) & d=\log_b a \\ O(n^{\log_b a}) & d<\log_b a \end{cases}$$

归并排序：$a=b=2, d=1$，
$T(n)=O(n\log n)$

算法思维方法

- 分治思想
  - 各种排序算法
  - 单因素优选法，好于二分法
  - 大整数乘法
  - 矩阵乘法
  - 分析复杂度的一般递推公式
- 其他方法：
  - 动态规划、贪心、随机
  - **快速排序**
    - **fastsort.go vs. quicksort.go**（演示）

```
package main  // fastsort.go
import (
   "fmt"
   "math/rand"
   "time"
)
const N = 1 * 1024 * 1024
func main() {
   rand.Seed(time.Now().UnixNano())
   s := make([]int, N)
   for i := 0; i < len(s); i++ {
      s[i] = rand.Int()
   //   s[i] = i
   }
   fastsort(s)
   fmt.Println("s[0]=",s[0],"s[1]=", s[1],"s[",N-1,"]=", s[N-1])
}
func fastsort(A []int) {
   if len(A) < 2 {
      return
   }
   lowerA, upperA := partition(A)
   fastsort(lowerA)
   fastsort(upperA)
}
func partition(A []int) ([]int, []int) {
   // pivotIndex := rand.Intn(len(A))   // randomly select a pivot
   // A[pivotIndex], A[len(A)-1] = A[len(A)-1], A[pivotIndex]
   lower := 0
   for i := 0; i < len(A); i++ {
      //if A[i] < A[pivotIndex] {
      if A[i] < A[len(A)-1] {
         A[lower], A[i] = A[i], A[lower]
         lower++
      }
   }
   A[lower], A[len(A)-1] = A[len(A)-1], A[lower]
   return A[0:lower], A[lower+1 : len(A)]
}
```

新内容：什么是系统思维?

- 系统思维使得计算过程实用
  - 如：求斐波那契数F(十亿)
  - 求斐波那契数的四个层级（斐波那契兔子问题）
    - 小学：1年后你家里有多少对兔子？ **求F(12)**
    - 中学：第50个月呢？ **求F(50)**
    - 大学：第10亿个月呢？ **求F(1,000,000,000)**
    - 研究生：用斐波那契数列思想求解希尔伯特第十问题（英文版第7页）

新内容：什么是系统思维?

- 系统思维使得计算过程实用
- How?
  - 使用**抽象**，组合**模块**成为系统，**无缝执行**计算过程
  - 向编程者提供抽象，比特精准地将抽象映射到最底层硬件
    - 霍尔悖论展示的递归抽象

新内容：什么是系统思维?

- 系统思维使得计算过程实用
- How?
  - 使用**抽象**，组合**模块**成为系统，**无缝执行**计算过程
  - 向编程者提供抽象，比特精准地将抽象映射到最底层硬件
- 系统思维的内涵
  - 两个基本思路：抽象栈与周期
  - 三个追求：周到性、系统性（整体性）、应对系统复杂性（不是算法复杂度）
  - 三个利器：抽象化、模块化、无缝衔接

- 系统思维涉及科学、工程、艺术
  - 系统设计者称为架构师（architect）
- 抽象（abstraction）是核心

计算思维的核心是抽象

At the heart of computational thinking is abstraction.

—Alfred Aho and Jeffrey Ullman, 2022

Alfred Aho

Jeffrey Ullman

Turing Lecture 图灵奖演讲论文

Aho, A. and Ullman, J. Abstractions, their algorithms and their compilers. CACM 65, 2 (Feb, 2022), 76--91.

抽象栈（T型栈）与周期

- 向编程者提供抽象，比特精准地将抽象映射到硬件
  - 硬件部分只需要掌握做加法

比特、字节、整数、数组、切片、文件、结构体、指针