中国科学院大学
University of Chinese Academy of Sciences

CS101

计算机科学导论

概述-2

计算机与计算过程

Acu-Exams实例

徐志伟
中科院计算所 中国科学院大学
zxu@ict.ac.cn

提纲

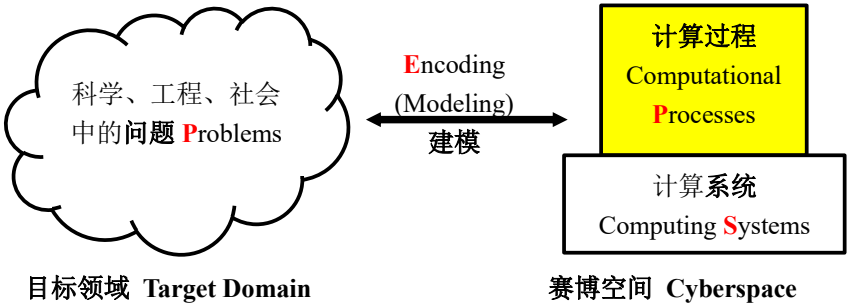
- 课程介绍
  - 教学目的、方法、目标
- 计算思维特征
  - 外部特征：ABC
  - 内部特征：Acu-Exams-CP（对计算思维的十种理解）
- 计算机与计算过程
  - 计算机的冯诺依曼模型
  - 计算过程及其演变
  - 正确、巧妙、实用的计算过程
    - Acu-Exams实例

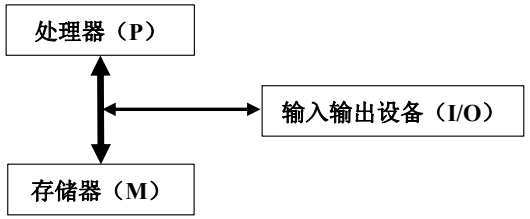
课件中可能包含素材引用，特此致谢！

1. 冯诺依曼模型

计算机用**冯诺依曼模型**刻画，有五个要点：

- **二进制**表示：使用二进制格式表示数据和程序，支持比特精准
- **三类部件**（P-M-I/O）：计算机包含处理器、存储器、输入输出设备
  - 三者连接成为计算机系统



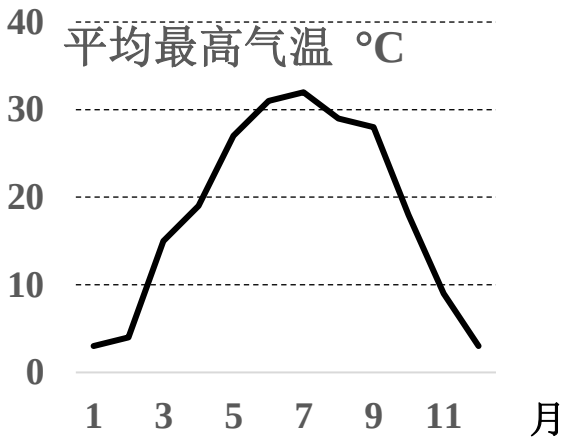


（a）计算机简图：连为一体的三个子系统

离散化支持比特精准

斐波那契数的数学定义
F(0)=0， F(1)=1，
当n>1时F(n)=F(n-1)+F(n-2)

- 有些量天然就是离散量（数字量），如斐波那契数F(n)
- 模拟量则需要离散化后，得到离散量
  - 两个模拟量（实数）中间总有另一个模拟量
    - 3与4中间有3.5，3与3.1之间有3.05
  - 两个离散量中间并不总有另一个离散量



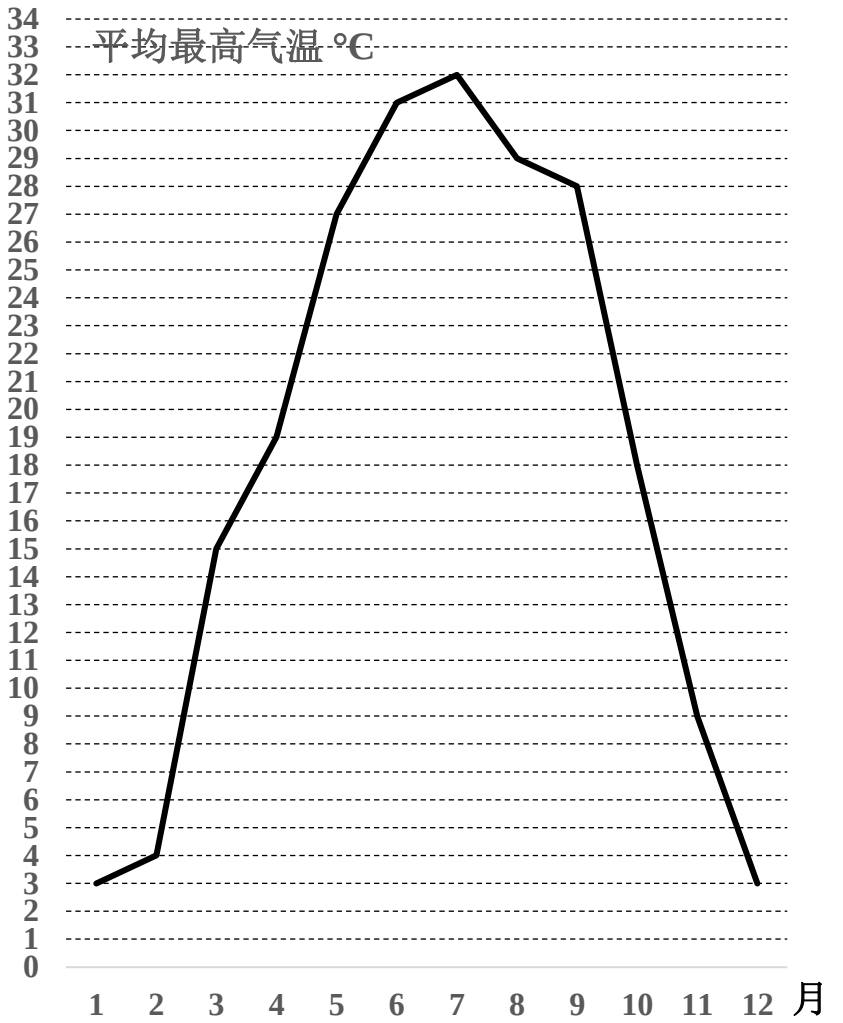
- 计算机使用一个字（word）来表示一个离散量，该字的比特数称为字长（word length）
  - 下表采用2个比特的字，可表示4个整数
    - 将每一个气温模拟值四舍五入，表示为 0°，10°，20°，30°
    - 在计算机内部，表示为二进制值 00， 01，10， 11

月份		一月	二月	三月	四月	五月	六月	七月	八月	九月	十月	十一月	十二月
气温 °C	十进制	0	0	20	20	30	30	30	30	30	20	10	0
	二进制	00	00	10	10	11	11	11	11	11	10	01	00

2019年北京每月平均最高气温

采用更多比特（更大字长）可得到更高精度

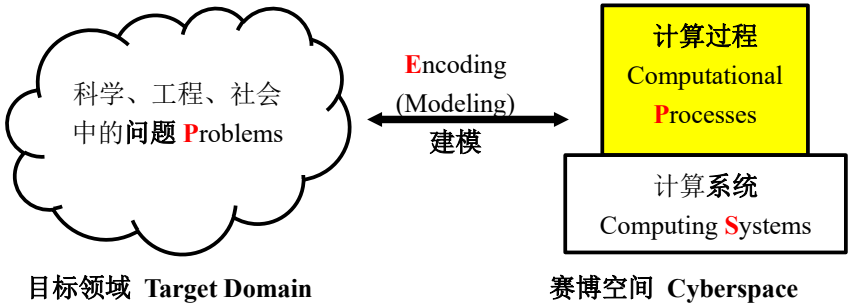
- 采用5比特
  - 可表示0~31 (2^5-1)之间的任意整数
  - 不能表示32
  - 怎么办？
    - 采用6比特
    - 采用5比特与溢出比特（overflow bit）
      - 产生31，并溢出



月份		一月	二月	三月	四月	五月	六月	七月	八月	九月	十月	十一月	十二月
气温 ℃	十进制	3	4	15	19	27	31	32	29	28	18	9	3
	二进制	00011	00100	01111	10011	11011	11111	?	11101	11100	10010	01001	00011

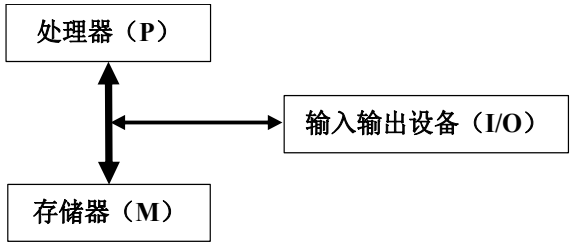
2019年北京每月平均最高气温

计算机的冯诺依曼模型

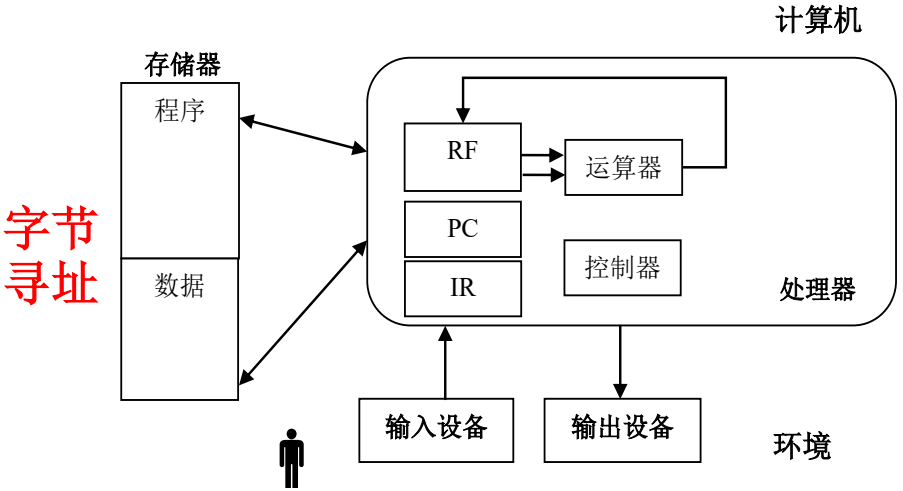


冯诺依曼模型的五个要点：

- **二进制**表示：使用二进制格式表示数据和程序
- **三类部件**（P-M-I/O）：计算机包含处理器、存储器、输入输出设备
  - 处理器包含运算器、控制器、若干寄存器
- **存储程序**计算机：计算机包含一个统一的存储器，可存储数据与程序
  - 一般是**字节寻址**
    - 字节是访问存储器的基本单位
- **指令驱动**的程序执行：计算机有一个指令集，程序是指令序列；计算机只能执行指令告诉它的操作
- **串行执行**：指令一条一条串行执行；执行完当前指令，才能执行下一条指令
- 为什么U盘是输入输出设备？
  - 区分：内存（memory），外存（storage），高速缓存（cache）

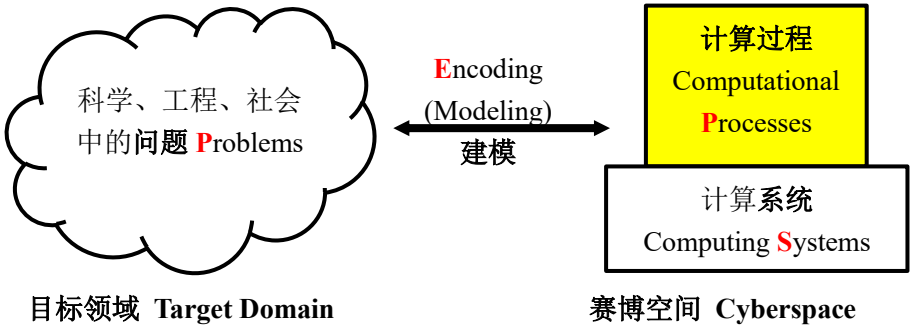


（a）计算机简图：连为一体的三个子系统

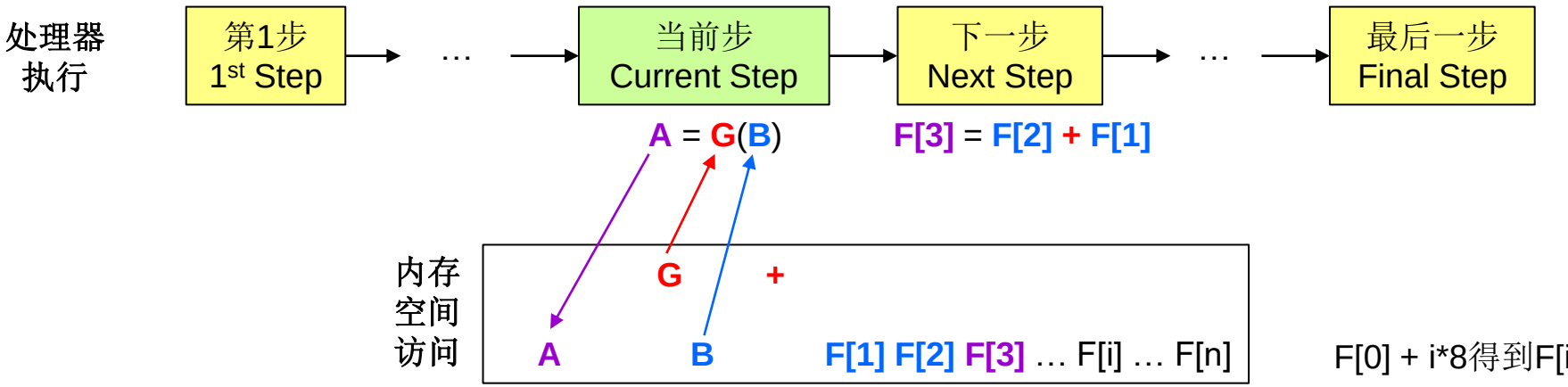


（b）更加详细的刻画

2. 计算过程



- 建模得到计算过程
  - 计算过程由算法、程序或逻辑电路刻画
- 计算过程在计算系统上执行
  - 一步又一步的逐步执行，是逐步执行的计算过程（step-by-step process）
- 计算过程是“操作数字符号的步骤序列”
  - 一个有限系列步骤：有第一步、最后一步、当前步骤、下一步骤
    - 每一步骤都是构造性的　　对比非构造性的步骤：存在性证明、反证法
  - 每一步骤的行为：（1）定位当前操作与操作数；（2）执行操作；（3）定位下一步骤
    - 执行当前操作：输入操作数→操作→结果操作数。如，B→G→A



计算过程

- 计算过程是逐步执行的“操作数字符号的步骤序列”
  - 一个有限系列步骤：有第一步、最后一步、当前步骤、下一步骤。
  - 每一步骤的行为：（1）定位当前操作与操作数；（2）执行操作；（3）定位下一步骤
    - 执行当前操作：输入操作数→操作→结果操作数。如，B→G→A

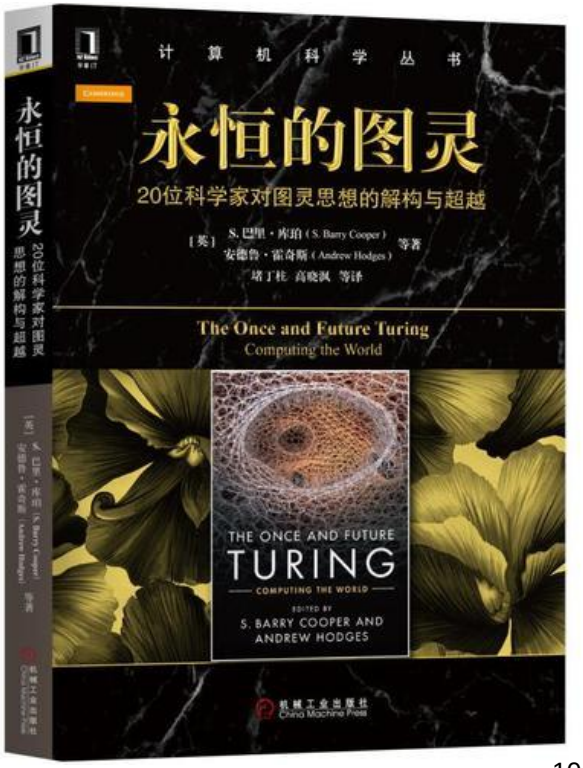
- **进阶理解（研究中）：希望计算过程的步骤具备代数完备性，尽量能够从头到尾执行完毕，不要在计算过程中途中断**
  - 数学的加减乘除算术运算具备代数完备性，除了x/0（包括0/0）等异常之外
    x, y是数，那么x+y，x-y，x*y，x/y也是数
  - 计算机算术不一定具备代数完备性，需要精心设计和使用
  - 结合律不一定成立。X+(Y+Z)不一定等于(X+Y)+Z；可能溢出
  - 例子：8比特无符号数的区间是[0, 255]，0是最小数，255 = 2^8-1 是最大数
    3+(255-254) = 4，不等于 (3+255)-254 = (258)溢出
    3-(255-254) = 2，不等于(3-255)-254 = (-252)溢出， (-506)溢出

当代逐步计算过程已变得十分强大

- 例子：斐波那契兔子问题
  - 有人在2021年1月送你一对刚诞生的兔子；一对兔子出生后，从第三个月开始就每月生一对小兔子。到了n月后， 你家里有多少对兔子（记为F(n)）？
  - 数学公式： F(0)=0， F(1)=1，当n>1时F(n)=F(n-1)+F(n-2)
- 习题1（小学）：到了2021年12月，你家里有多少对兔子？ **求F(12)**
- 习题2（中学）：到了从2021年1月开始的第50个月呢？ **求F(50)**
- 习题3（大学）：到了从2021年1月开始的第10亿个月呢？ **求F(1,000,000,000)**
- 习题4（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）

理论研究：数学与计算机科学融合

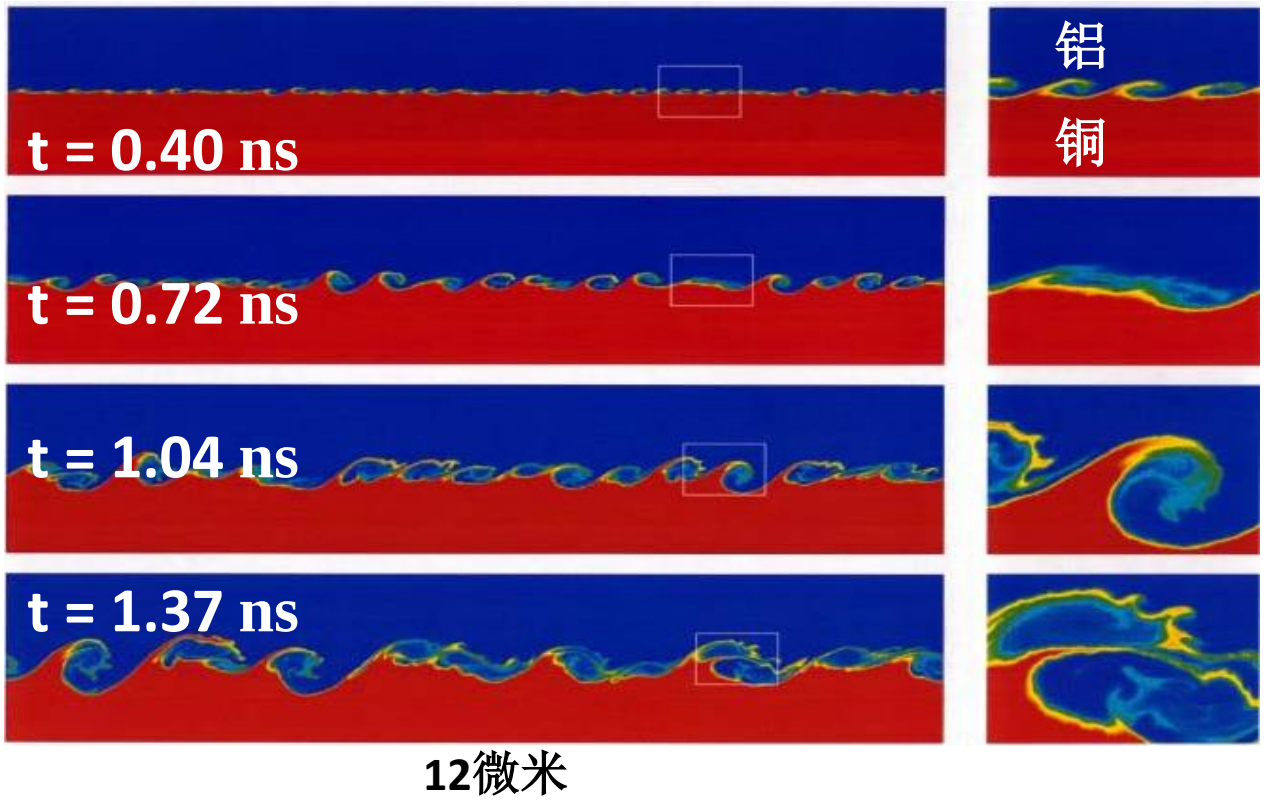
- 希尔伯特第十问题
  - 1900年：希尔伯特在世界数学家大会上提出23个重大数学问题，其中第十问题（也称为丢番图方程问题）是
    - **找到一个算法，能判定：整系数多项式方程是否存在整数解**
  - 不可解性定理
    - **存在不可判定的可列集**
  - 1950年，戴维斯提出猜想
    - **猜想：每个可列集都是丢番图集**
  - 1970年，马季亚谢维奇定理（**DPRM定理**）
    - **Martin Davis，纽约大学计算机科学系教授**
    - Hilary Putnam，美国哲学学会主席
    - Julia Robinson，美国数学学会主席
    - 尤里·弗拉基米罗维奇·马季亚谢维奇，俄罗斯数学家 (Юрий Владимирович Матиясевич)
    - **定理：每个可列集都是丢番图集**
- 重要诀窍：**斐波那契数列**



科学与工程： See the invisible: Atoms in the Surf

使用计算机模拟90亿个原子的行为，重现开尔文-亥姆霍兹不稳定性 (演示)
高温: 2000K; 高速: 2000 m/s；模拟数纳秒行为耗费3600万CPU小时

$A_i(t)$是原子i在t时刻的构象
时间步增量i++为皮秒级或飞秒级




```
for t=0; t<T; t++ {
  for i=0; t<9B; i++ {
    A_i(t+1) = Σ_{j=1}^{9B} f(A_i(t), A_j(t))
  }
}     // 9B=9,000,000,000
```

Richards D F, Krauss L D, Cabot W H, et al. (2008). Atoms in the Surf: Molecular Dynamics Simulation of the Kelvin-Helmholtz Instability Using 9 Billion Atoms. https://arxiv.org/abs/0810.3037 and www.youtube.com/watch?v=Wr7WbKODM2Q.

3. 计算过程演变：从一位诗人说起

- 拜伦（George Byron，1788-1824）
  - 英国浪漫派诗人
    - 品行（dispositions）看重想象力、热情
  - 革命者，36岁逝于希腊民族解放运动中
- 他的从未见面过的女儿
  埃达·拜伦（**Ada** Byron，1815－1852）
    - 36岁逝世

致谢
希腊
邮局

- **She Walks in Beauty**

She walks in beauty, like the night
Of cloudless climes and starry skies;

……

A mind at peace with all below,
A heart whose love is innocent!

吕志鲁译

她在美中徜徉，
她在美中穿行；
象深邃的苍穹缀满繁星，
象皎洁的夜空万里无云。

……

人间万事平心以待，
恰似美的天神；
一颗心装着至爱，
一颗心永远真纯。

“诗一般的科学”

通用的操纵符号的机器

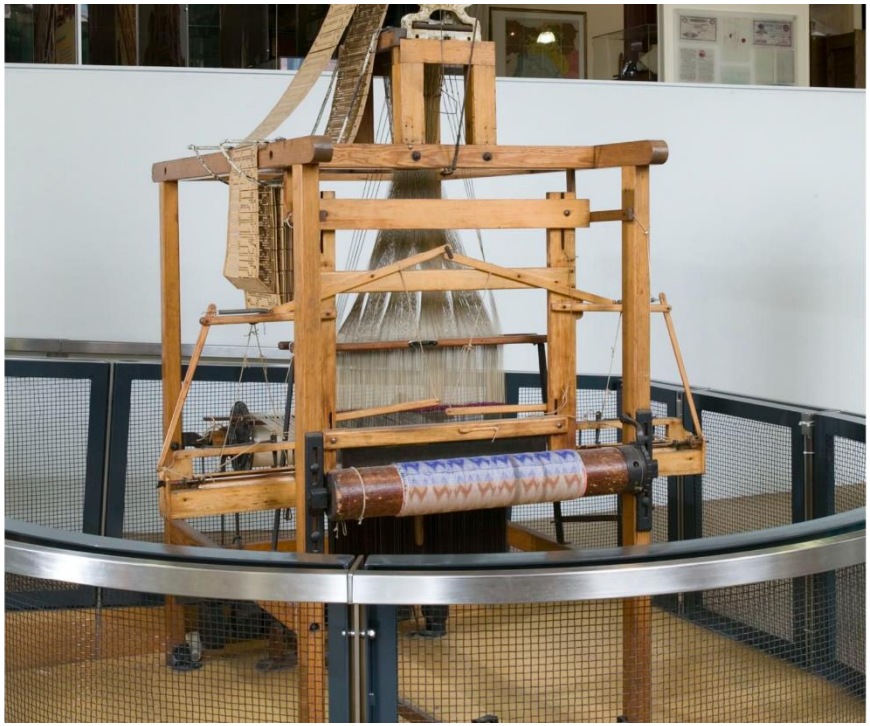
第一位程序员

编程：编织各种美妙事物

第一个计算机程序

编程：编织各种美妙事物

- Ada：就像提花机编织各种花样和树叶一样，分析机编织各种代数模式。
  - … the Analytical Engine weaves algebraical patterns just as the Jacquard-loom weaves flowers and leaves.



https://collection.sciencemuseum.org.uk/

https://www.ribbonworksdesignstudio.com/

Ada的“程序”

有错误，并未真正运行起来

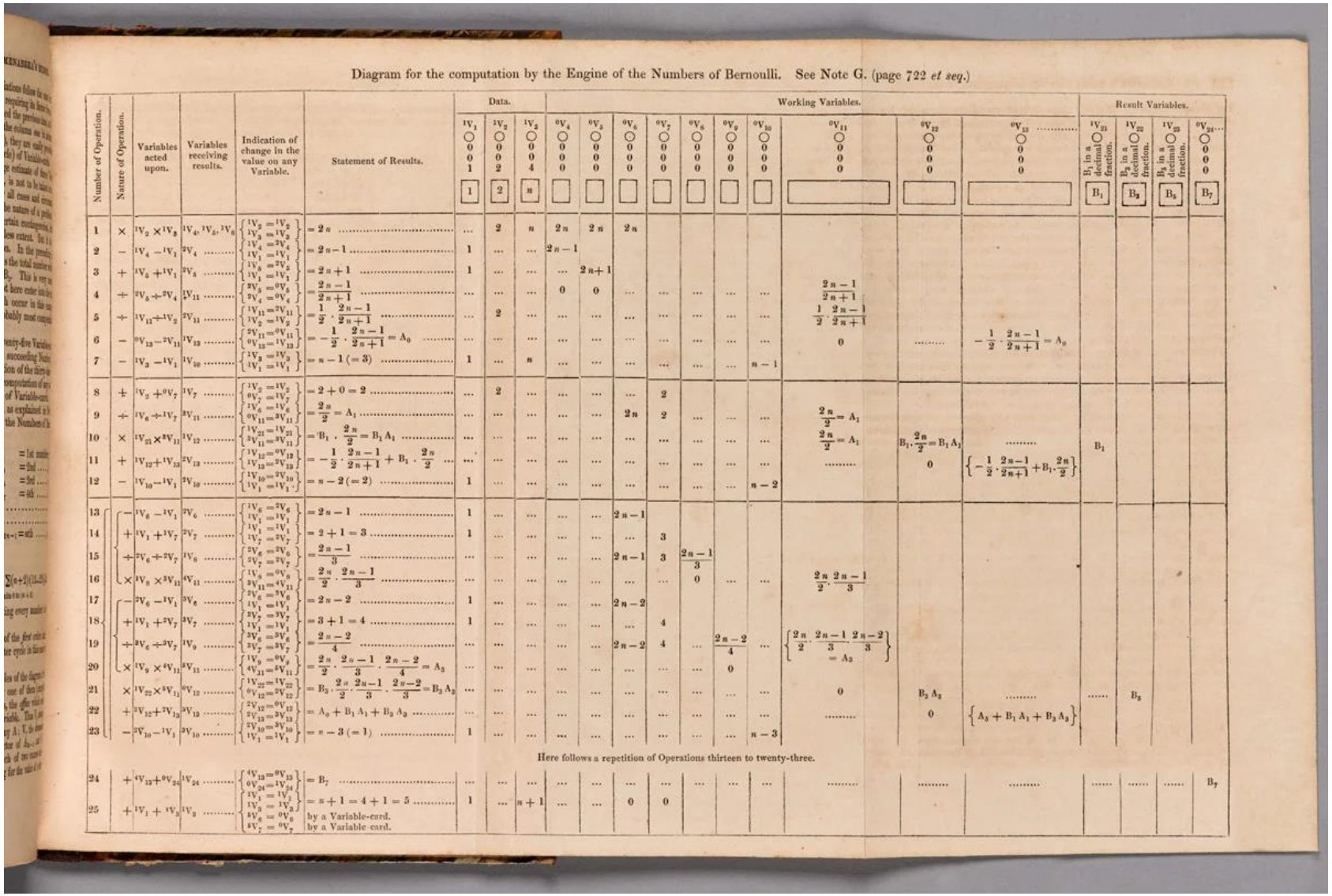


Diagram for the computation by the Engine of the Numbers of Bernoulli. See Note G. (page 722 *et seq.*)

Number of Operation.	Nature of Operation.	Variables acted upon.	Variables receiving results.	Indication of change in the value on any Variable.	Statement of Results.	Data. ${}^1V_1$ 1	Data. ${}^1V_2$ 2	Data. ${}^1V_3$ n	Working Variables. ${}^0V_4$	${}^0V_5$	${}^0V_6$	${}^0V_7$	${}^0V_8$	${}^0V_9$	${}^0V_{10}$	${}^0V_{11}$	${}^0V_{12}$	${}^0V_{13}$	Result Variables. ${}^1V_{21}$ B_1	${}^1V_{22}$ B_3	${}^1V_{23}$ B_5	${}^0V_{24}$... B_7
1	×	${}^1V_2 \times {}^1V_3$	${}^1V_4, {}^1V_5, {}^1V_6$	${}^1V_2 = {}^1V_2$, ${}^1V_3 = {}^1V_3$	$= 2n$	...	2	n	$2n$	$2n$	$2n$											
2	−	${}^1V_4 - {}^1V_1$	${}^2V_4$	${}^1V_4 = {}^2V_4$, ${}^1V_1 = {}^1V_1$	$= 2n-1$	1	...	...	$2n-1$													
3	+	${}^1V_5 + {}^1V_1$	${}^2V_5$	${}^1V_5 = {}^2V_5$, ${}^1V_1 = {}^1V_1$	$= 2n+1$	1	...	...	...	$2n+1$												
4	+	${}^2V_5 + {}^2V_4$	${}^3V_{11}$	${}^2V_5 = {}^0V_5$, ${}^2V_4 = {}^0V_4$	$= \frac{2n-1}{2n+1}$	...	...	...	0	0	...	...	...	...	...	$\frac{2n-1}{2n+1}$						
5	÷	${}^1V_{11} \div {}^2V_2$	${}^2V_{11}$	${}^1V_{11} = {}^2V_{11}$, ${}^1V_2 = {}^1V_2$	$= \frac{1}{2} \cdot \frac{2n-1}{2n+1}$	...	2	...	...	...	...	...	...	...	...	$\frac{1}{2} \cdot \frac{2n-1}{2n+1}$						
6	−	${}^0V_{13} - {}^2V_{11}$	${}^1V_{13}$	${}^2V_{11} = {}^0V_{11}$, ${}^0V_{13} = {}^1V_{13}$	$= -\frac{1}{2} \cdot \frac{2n-1}{2n+1} = A_0$	...	...	...	...	...	...	...	...	...	...	0	.....	$-\frac{1}{2} \cdot \frac{2n-1}{2n+1} = A_0$				
7	−	${}^1V_3 - {}^1V_1$	${}^1V_{10}$	${}^1V_3 = {}^1V_3$, ${}^1V_1 = {}^1V_1$	$= n-1\ (=3)$	1	...	n	...	...	...	...	...	...	$n-1$							
8	+	${}^1V_2 + {}^0V_7$	${}^1V_7$	${}^1V_2 = {}^1V_2$, ${}^0V_7 = {}^1V_7$	$= 2+0 = 2$	...	2	...	...	...	...	2										
9	÷	${}^1V_6 \div {}^1V_7$	${}^2V_{11}$	${}^1V_6 = {}^0V_6$, ${}^0V_{11} = {}^2V_{11}$	$= \frac{2n}{2} = A_1$	...	...	...	...	...	$2n$	2	...	...	...	$\frac{2n}{2} = A_1$						
10	×	${}^1V_{21} \times {}^2V_{11}$	${}^1V_{12}$	${}^1V_{21} = {}^1V_{21}$, ${}^2V_{11} = {}^2V_{11}$	$= B_1 \cdot \frac{2n}{2} = B_1 A_1$	...	...	...	...	...	...	...	...	...	...	$\frac{2n}{2} = A_1$	$B_1 \cdot \frac{2n}{2} = B_1 A_1$	.....				
11	+	${}^1V_{12} + {}^1V_{13}$	${}^2V_{13}$	${}^1V_{12} = {}^0V_{12}$, ${}^1V_{13} = {}^2V_{13}$	$= -\frac{1}{2} \cdot \frac{2n-1}{2n+1} + B_1 \cdot \frac{2n}{2}$ ...	...	...	...	...	...	...	...	...	...	...	.....	0	$\left\{-\frac{1}{2} \cdot \frac{2n-1}{2n+1} + B_1 \cdot \frac{2n}{2}\right\}$	B_1			
12	−	${}^1V_{10} - {}^1V_1$	${}^2V_{10}$	${}^1V_{10} = {}^2V_{10}$, ${}^1V_1 = {}^1V_1$	$= n-2\ (=2)$	1	...	...	...	...	...	...	...	...	$n-2$							
13	−	${}^1V_6 - {}^1V_1$	${}^2V_6$	${}^1V_6 = {}^2V_6$, ${}^1V_1 = {}^1V_1$	$= 2n-1$	1	...	...	...	...	$2n-1$											
14	+	${}^1V_1 + {}^1V_7$	${}^2V_7$	${}^1V_1 = {}^1V_1$, ${}^1V_7 = {}^2V_7$	$= 2+1 = 3$	1	...	...	...	...	...	3										
15	÷	${}^2V_6 \div {}^2V_7$	${}^1V_8$	${}^2V_6 = {}^2V_6$, ${}^2V_7 = {}^2V_7$	$= \frac{2n-1}{3}$	...	...	...	...	...	$2n-1$	3	$\frac{2n-1}{3}$									
16	×	${}^1V_8 \times {}^3V_{11}$	${}^4V_{11}$	${}^1V_8 = {}^0V_8$, ${}^3V_{11} = {}^4V_{11}$	$= \frac{2n}{2} \cdot \frac{2n-1}{3}$	...	...	...	...	...	...	...	0	...	...	$\frac{2n}{2} \cdot \frac{2n-1}{3}$						
17	−	${}^2V_6 - {}^1V_1$	${}^3V_6$	${}^2V_6 = {}^3V_6$, ${}^1V_1 = {}^1V_1$	$= 2n-2$	1	...	...	...	...	$2n-2$											
18	+	${}^1V_1 + {}^2V_7$	${}^3V_7$	${}^2V_7 = {}^3V_7$, ${}^1V_1 = {}^1V_1$	$= 3+1 = 4$	1	...	...	...	...	...	4										
19	÷	${}^3V_6 \div {}^3V_7$	${}^1V_9$	${}^3V_6 = {}^3V_6$, ${}^3V_7 = {}^3V_7$	$= \frac{2n-2}{4}$	...	...	...	...	...	$2n-2$	4	...	$\frac{2n-2}{4}$	...	$\left\{\frac{2n}{2} \cdot \frac{2n-1}{3} \cdot \frac{2n-2}{3} = A_3\right\}$							
20	×	${}^1V_9 \times {}^4V_{11}$	${}^5V_{11}$	${}^1V_9 = {}^0V_9$, ${}^4V_{11} = {}^4V_{11}$	$= \frac{2n}{2} \cdot \frac{2n-1}{3} \cdot \frac{2n-2}{4} = A_3$	...	...	...	...	...	...	...	...	0	...							
21	×	${}^1V_{22} \times {}^5V_{11}$	${}^6V_{12}$	${}^1V_{22} = {}^1V_{22}$, ${}^5V_{12} = {}^2V_{12}$	$= B_3 \cdot \frac{2n}{2} \cdot \frac{2n-1}{3} \cdot \frac{2n-2}{3} = B_3 A_3$	...	...	...	...	...	...	...	...	...	...	0	$B_3 A_3$	.....	.....	B_3		
22	+	${}^2V_{12} + {}^2V_{13}$	${}^3V_{13}$	${}^2V_{12} = {}^0V_{12}$, ${}^2V_{13} = {}^3V_{13}$	$= A_3 + B_1 A_1 + B_3 A_3$	...	...	...	...	...	...	...	...	...	...	.....	0	$\{A_3 + B_1 A_1 + B_3 A_3\}$				
23	−	${}^2V_{10} - {}^1V_1$	${}^3V_{10}$	${}^2V_{10} = {}^3V_{10}$, ${}^1V_1 = {}^1V_1$	$= n-3\ (=1)$	1	...	...	...	...	...	...	...	...	$n-3$							

Here follows a repetition of Operations thirteen to twenty-three.

24	+	${}^4V_{13} + {}^0V_{24}$	${}^1V_{24}$	${}^4V_{13} = {}^0V_{13}$, ${}^0V_{24} = {}^1V_{24}$	$= B_7$	...	...	...	...	...	...	...	...	...	...	.....	.....	.....	.....	.....	.....	.....	B_7
25	+	${}^1V_1 + {}^0V_2$	${}^1V_2$	${}^1V_1 = {}^1V_1$, ${}^1V_2 = {}^1V_2$, ${}^0V_6 = {}^0V_6$ by a Variable-card, ${}^0V_7 = {}^0V_7$ by a Variable-card.	$= n+1 = 4+1 = 5$	1	...	$n+1$	...	...	0	0												

致谢英国科技网站The Register。 https://regmedia.co.uk/2015/10/12/lovelace_algorithm.jpg

3.1 运算流

- 从古老的算盘到帕斯卡加法器、莱布尼茨乘法器、巴贝奇分析机，**数千年来的计算过程都是运算流计算过程**，有两个特点：
  - 算术运算：每个步骤只是一个加减乘除算术运算
  - 直线程序：从第一个步骤开始，执行完当前步骤，顺序执行下一个步骤，直至最后步骤
- Ada的程序计算伯努利数列，即公式$\frac{x}{e^x} = \sum_{n\geq 0} B_n \frac{x^n}{n!}$中的$B_n$的值
  - 实际只计算B_1, B_3, B_5, B_7
- Ada的程序包含控制流思想，但未能精准鲜明地表达出来

运算号	运算	输入变量	结果变量	部分含义注释
1	×	V2,V3	V4,V5,V6	V2×V3 → V4,V5,V6
2	-	V4,V1	V4	V4-V1 → V4
3	+	V5,V1	V5	V5+V1 → V5
4	÷	V5,V4	V11	V5÷V4 → V11
5	÷	V11,V2	V11	V11÷V2 → V11
6	-	V13,V11	V13	V13-V11 → V13
7	-	V3,V1	V10	V3-V1 → V10
……				
23	-	V10,V1	V10	V10-V1 → V10
Here follows a repetition of Operations 13 to 23				
24	+	V13,V24	V24	V13+V24 → V24；计算出了B_7放在V24
25	+	V1,V3	V3	V1+V3 → V3

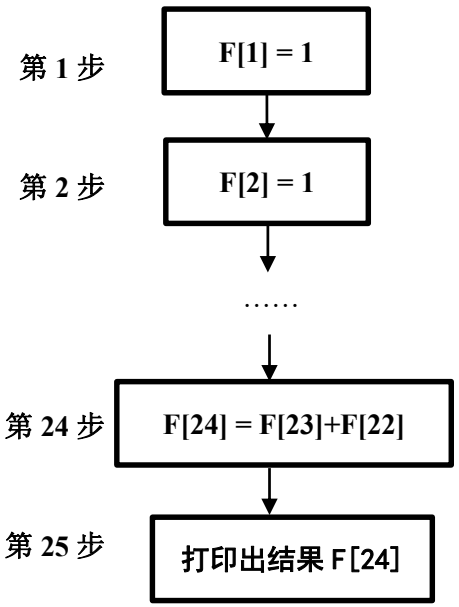
运算流计算过程的表达能力太差

- 从求解斐波那契数列可以看出，运算流存在两个问题
  - 相对于数学的简洁表示，偏离太多
    - **数学公式很简洁： F(0)=0， F(1)=1，当n>1时F(n)=F(n-1)+F(n-2)**
  - “程序”长度与问题规模n相关，需要n+1行；仅适合n很小的情况
    - 求F(1,000,000,000)的程序需要十亿行代码；粗暴枚举步骤的方式不可行
- **程序有限性要求：程序行数与问题规模无关**

运算流计算过程的表达

25个顺序步骤
每一步骤是算术运算

```
1   F[1] = 1
2   F[2] = 1
3   F[3] = F[2]+F[1]
4   F[4] = F[3]+F[2]
5   F[5] = F[4]+F[3]
……
24 F[24] = F[23]+F[22]
25 打印出结果F[24]
```



3.2 控制流

- 1945年，ENIAC设计者提出**让电子能够思考**
  - 计算机可在每一步即时做判断：根据当前操作结果自动决定下一操作步骤，并不一定是事先排好的顺序下一步骤
    - If-then-else, for loop

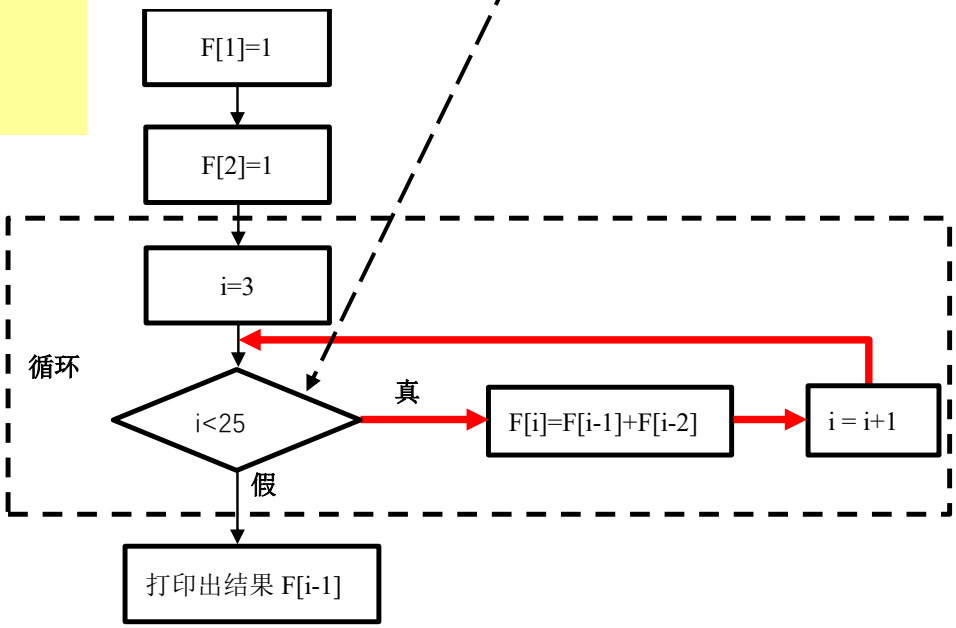
```
1    F[1] = 1
2    F[2] = 1
3    F[3] = F[2]+F[1]
4    F[4] = F[3]+F[2]
5    F[5] = F[4]+F[3]
……
24   F[24] = F[23]+F[22]
25   打印出结果 F[24]
```

- **3~24 行相似，重复了 n-2 遍。任意两行只是索引不同。**
- **将第 3~24 行用 1~3 行新抽象表示，可做到行数与问题规模无关。**
- **索引值从 3 开始，到 24 结束；每次迭代索引值增 1。**

即时判断：根据当前操作结果i<25，判断真、假决定下一步骤

使用for循环，3行替代n-2行

```
1   F[1] = 1
2   F[2] = 1
3   for i := 3; i<25; i=i+1 {
4     F[i] = F[i-1]+F[i-2]
5   }
6   打印出结果F[i-1]
```



常见的五种控制流

- 表达式中的**运算优先级**（先乘除后加减）
- **顺序**，这是缺省控制流，sequence
- **条件判断**，if-then-else
- **循环**，for loop

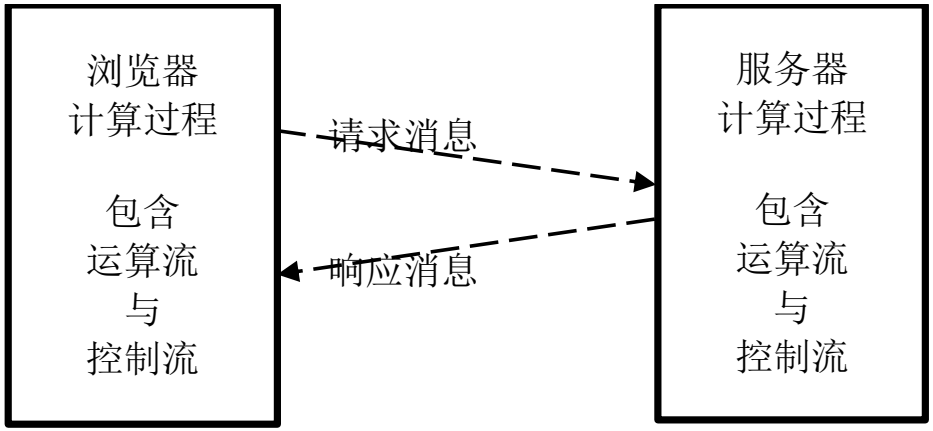
- **函数**调用，包括**递归**调用，recursion （模块化抽象）
  - 循环和递归都能实现程序有限性

- 没有改变逐步计算过程的本质
  - step-by-step computational process

3.3 消息流

- 体现网络思维
- 多台计算机上的程序发送消息，协同工作
  - 笔记本电脑上的浏览器程序
  - 服务器计算机上的万维网服务器程序

例子：个人作品实验
动态网页



网络

笔记本电脑

服务器

4. 正确、巧妙、实用的计算过程　Acu-Exams实例

- 例子：斐波那契兔子问题
  - 有人在2021年1月送你一对刚诞生的兔子；一对兔子出生后，从第三个月开始就每月生一对小兔子。到了n月后， 你家里有多少对兔子（记为F(n)）？
  - 数学公式： F(0)=0， F(1)=1，当n>1时F(n)=F(n-1)+F(n-2)
- 不需要计算思维也能做（这种问题有时被称为玩具问题，toy problem）
  - 习题1（小学）：到了2021年12月，你家里有多少对兔子？　　求F(12)
- **需要计算思维**
  - 习题2（中学）：到了从2021年1月开始的第50个月呢？　　**求F(50)**
  - 习题3（大学）：到了从2021年1月开始的第10亿个月呢？　　**求F(1,000,000,000)**
  - 习题4（研究生）：用斐波那契数列思想求解希尔伯特第十问题　**思维实验**
- **Acu-Exams实例**
  - 自动（**Acu-Exams**）：二进制程序在计算机上自动执行
  - 正确（**Acu-Exams**）：通用的比特精准
    - 自动计算有代价：与数学的手工计算相比，计算机的自动计算需要应对更多种误差
  - 巧妙（**Acu-Exams**）：巧妙的算法和系统可指数倍提升计算速度
  - 实用（**Acu-Exams**）：比特精准地自动执行的系统抽象（霍尔悖论）

4.1 二进制程序在计算机上自动执行（Acu-Exams）

- 用算法和程序刻画计算过程
  - 算法，往往用伪代码表示
    - 写给人看的
  - 程序，计算机语言表达的算法

易理解	高级语言程序	
难理解	汇编语言程序	
很难理解		机器语言程序
	不能直接执行	能直接执行

小学的直觉思维过程表达	数学表达	
	n	F(n)
2021年0月：0对。	0	F(0)=0
2021年1月：1对。	1	F(1)=1
2021年2月：还是1对；因为第一对兔子还未成年。	2	F(2)=1
2021年3月：2对；第一对兔子成年了，生了一对小兔子。	3	F(3)=1+1=2
2021年4月：3对；第一对兔子又生了一对小兔子。	4	F(4)=2+1=3
2021年5月：5对；除了4月的3对兔子之外，第一对兔子又生了一对小兔子，第二对兔子生了一对小兔子。	5	F(5)=F(4)+F(3)=5
……	…	……
2021年12月：144对。	12	F(12)=F(11)+F(10)=144


```
Output F(12)                    // 算法，伪代码
where F(n) is defined as
   if (n=0 or n=1) then F(n)=n
   else F(n)=F(n-1)+F(n-2)
```

```
package main                    // Go程序
import "fmt“                    // fib-12.go
func main() {
   fmt.Println("F(12)=",fibonacci(12))
}
func fibonacci(n int) int {
   if n == 0 || n == 1 { return n }
   return fibonacci(n-1)+fibonacci(n-2)
}
```

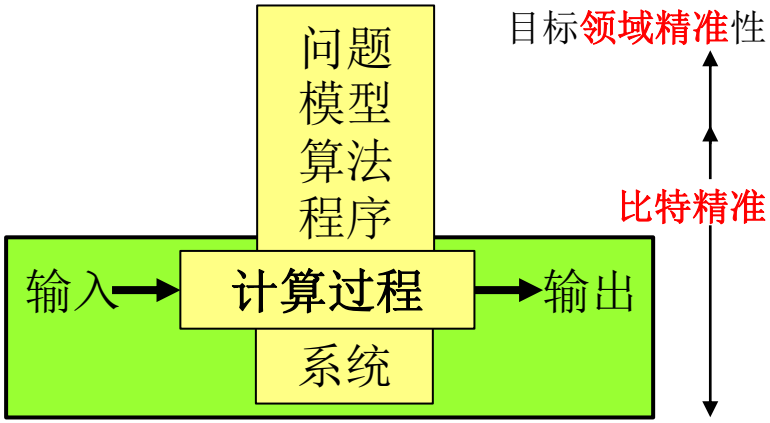
```
…
MOVQ   (TLS), CX          // 汇编程序
CMPQ   SP, 16(CX)         // fib-12.asm
JLS    181
SUBQ   $96, SP
MOVQ   BP, 88(SP)
LEAQ   88(SP), BP
…
```

```
01111111 01000101 01001100 01000110
00000010 00000001 00000001 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000010 00000000 00111110 00000000
00000001 00000000 00000000 00000000
```

二进制程序
fib-12

4.2 什么是通用的比特精准？（Acu-Exams）

- 计算过程的每一个比特是**正确**的
  - **每一个步骤按照程序指令执行，产生指令规定的输出比特**
- 结果精度可为精确或近似
  - 算法尽量产生正确结果；某些算法不强求精确答案；由应用建模确定
- 理论支撑：邱奇-图灵命题



精确答案

F(50)=12586269025

$$F(n)=\begin{cases}0 & n=0\\ 1 & n=1\\ F(n-1)+F(n-2) & n>1\end{cases}$$

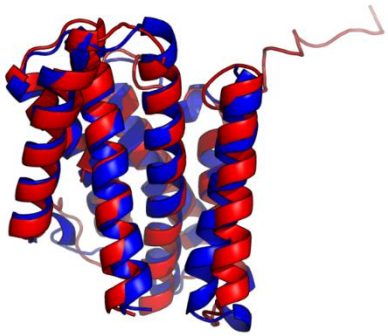
```
var fib [51]int
fib[0] = 0
fib[1] = 1
for i := 2; i < 51; i++ {
   fib[i] = fib[i-1] + fib[i-2]
}
```

非精确答案

蛋白质结构预测

精度>50%

红：预测；蓝：真实



图片来源：许锦波教授

斐波那契数列习题2（中学）：求F(50)

- 编译运行fib-50.go（演示）
  - 计算速度太慢了
- 重新建模，利用比内公式，开发出fib.binet.go程序

$$F(n)=\frac{\varphi^n-(1-\varphi)^n}{\sqrt{5}},$$

其中 $\varphi=\frac{1+\sqrt{5}}{2}=1.618\ldots$

- 编译运行fib.binet.go演示
  - 速度很快，但只得到近似结果

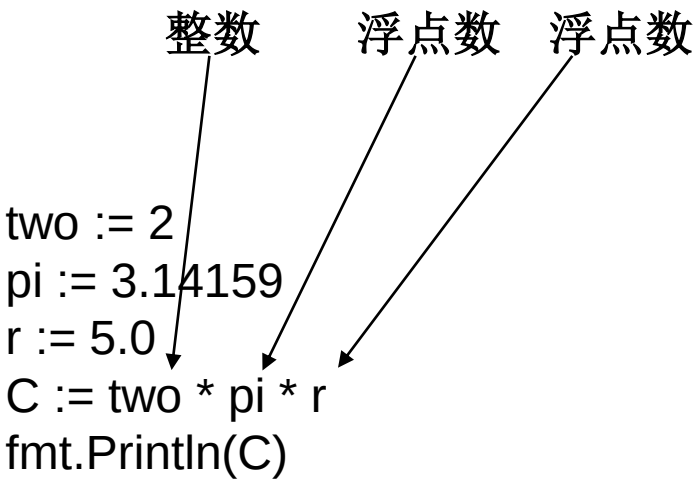
F(50)= 1.2586269024999998e+10
= 12586269024.999998

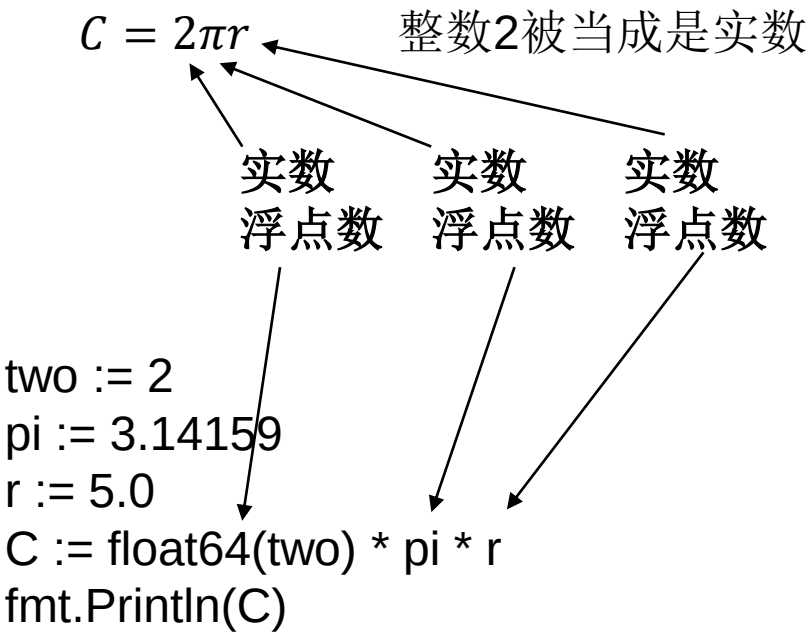
正确结果是F(50)=12586269025

  - 出现舍入误差（roundoff error），10^{-6}量级
  - 可容忍的舍入误差由各个应用问题决定
    - 很多日常问题使用牛顿力学公式且假设10^{-6}米精度就足够了
    - 引力波探测则要求严格得多的精度，达到5×10^{-23}米

数学精准 与 计算机科学的比特精准 对比

- 数学计算强调数学精准，可有明确定义的异常情况
  - $Y = X/0$；实数计算出现 $A = \sqrt{-2.3}$；
  - 数据类型失配（type mismatch）
    - 整数 + 三角形 = ？（无意义）
- 计算机科学继承数学的异常，出现异常可能报错
  - $Y = X/0$（**除零异常**）；实数计算出现 $A = \sqrt{-2.3}$ （**NaN**）
  - **数据类型失配**
    - 周长公式
    - 错误和正确的计算机代码





计算机科学的计算还可能出现下列误差

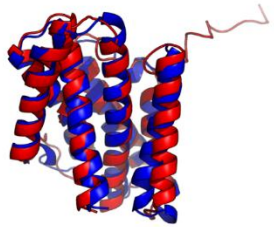
- 建模误差
  - 如，蛋白质结构预测精GDT_TS>50%
  - 应用领域决定何种误差是可以容忍的
- 截止误差（roundoff error）
  - fib.go演示　　F(50)=12586269025
  - fib.binet.go演示　　F(50)=1.258626902499998e+10 = 12586269024.999998
- 溢出（overflow）
  - 编译运行fib.dp.go，我们得到（演示）

F(91) = 4660046610375530309

F(92) = 7540113804746346429

F(93) = -6246583658587674878（错误）

= 12200160415121876738 - 2^{64}



$$F(n) = \frac{\varphi^n - (1-\varphi)^n}{\sqrt{5}},$$

其中 $\varphi = \frac{1+\sqrt{5}}{2} = 1.618\ldots$

- 64比特整数能够表示的最大值是 2^{63}-1 = 9223372036854775807
- F(93) = 12200160415121876738 太大了， 64比特整数类型装不下 溢出！

Acu-Exams

正确性（**Correctness**）：计算过程中每一个比特都是正确的（比特精准）

通用性（**Universality**）：用比特精准支持各种应用问题的领域精准

4.3 巧妙性（Acu-Exams）

- 假设我们要求精准结果F(50)=12586269025
  - 不准有舍入误差
- 采用一种聪明的动态规划算法开发出新程序fib.dp.go
- 编译运行fib.dp.go（**演示**），得到精准结果F(50)=12586269025
  - 速度很快
  - 时间复杂度降低
    - fib.go的时间复杂度：$O(2^n)$
    - fib.dp.go的时间复杂度：$O(n)$

```
package main            // fib-50.go
import "fmt“
func main() {
   fmt.Println("F(50)=",fibonacci(50))
}
func fibonacci(n int) int {
   if n == 0 || n == 1 { return n }
   return fibonacci(n-1)+fibonacci(n-2)
}
```

```
package main            // fib.dp-50.go
import "fmt"
const N = 50
var mem [N + 1]int
func main() {
   for i := 0; i <= N; i++ { mem[i] = -1 }
   fmt.Printf("F(%d)=%d\n", N, fibonacci(N))
}
func fibonacci(n int) int {
   if mem[n] != -1 { return mem[n] }
   if n == 0 || n == 1 {
      mem[n] = n
      return mem[n]
   }
   mem[n] = fibonacci(n-1) + fibonacci(n-2)
   return mem[n]
}
```

4.4 实用性（Acu-Exams）

- 斐波那契数列习题3（大学）：求F(1,000,000,000)
- 程序fib.dp.go能解决问题吗？
- 编译运行程序fib.dp-93.go（演示），求F(93)
  - 出错：F(93)= -6246583658587674878
  - 正确值F(93)= 12200160415121876738
  - 原因：一个64位整数装不下，溢出
  - 64比特整数能够表示的最大值是
    $2^{63}-1$ = 9223372036854775807
  - F(93) = 12200160415121876738
    太大了
- 需要系统支持任意字长的整数

```
package main              // fib.dp-93.go
import "fmt"
const N = 93
var mem [N + 1]int
func main() {
   for i := 0; i <= N; i++ { mem[i] = -1 }
   fmt.Printf("F(%d)=%d\n", N, fibonacci(N))
}
func fibonacci(n int) int {
   if mem[n] != -1 { return mem[n] }
   if n == 0 || n == 1 {
      mem[n] = n
      return mem[n]
   }
   mem[n] = fibonacci(n-1) + fibonacci(n-2)
   return mem[n]
}
```

正确、巧妙、实用的计算过程 Acu-Exams

- 斐波那契数列习题3（大学）：求F(1,000,000,000)
- 重新建模得到程序fib.matrix.go，利用了斐波那契数列的矩阵性质
  - $\begin{bmatrix}1 & 1\\ 1 & 0\end{bmatrix}^n = \begin{bmatrix}F(n+1) & F(n)\\ F(n) & F(n-1)\end{bmatrix}$ $A^{16} = (((A^2)^2)^2)^2$
  - 并通过矩阵平方求矩阵n次方，将计算复杂度从$O(n)$降为~$O(\log n)$
- 利用系统提供的**任意长整数big.Int**抽象，避免了溢出（ **Acu-Exams** ）
  - **计算F(5,000,000)仅需要4秒；计算F(1,000,000,000)需要十多万秒**
    - F(1,000,000,000)是一个很大的正整数，有2000多万个十进制数字
    - 进阶实验：十小时内计算出F(1,000,000,000)

Acu-Exams
构造性（**Effectiveness**）：人们构造出聪明的方法让计算机有效地解决问题
复杂度（Complexity）：这些聪明的方法（算法）具备时间/空间复杂度

求解F(n)的执行时间（秒）

n	fib.go $O(2^n)$	fib.dp.go ~$O(n)$	fib.matrix.go ~$O(\log n)$
50	725	0.059	0.000012
500	出错、极慢	出错	0.000022
5,000,000	出错、极慢	出错	4.13
1,000,000,000	出错、极慢	出错、很慢	187,160

霍尔悖论

- 1959年在莫斯科国立大学当访问学生时产生快排思想
- 1960年在Elliott Brothers公司为Elliott 803计算机开发排序程序时，发明并实现快排算法。但是，

“**Very difficult to explain**” 很难向他人说明快排算法

- 1961年发表，非常简洁易懂（只有半页，8行伪代码）

Hoare, C. A. R. (1961). "Algorithm 64: Quicksort". Comm. ACM. 4 (7): 321

ALGORITHM 64
QUICKSORT
C. A. R. HOARE
Elliott Brothers Ltd., Borehamwood, Hertfordshire, Eng.

```
procedure  quicksort (A,M,N); value M,N;
             array A; integer M,N;
comment  Quicksort is a very fast and convenient method of
sorting an array in the random-access store of a computer. The
entire contents of the store may be sorted, since no extra space is
required. The average number of comparisons made is 2(M−N) ln
(N−M), and the average number of exchanges is one sixth this
amount. Suitable refinements of this method will be desirable for
its implementation on any actual computer;
begin      integer I,J;
           if M < N then begin partition (A,M,N,I,J);
                               quicksort (A,M,J);
                               quicksort (A, I, N)
                         end
end        quicksort
```

与教科书版本基本一样

ALGORITHM 65
FIND
C. A. R. HOARE
Elliott Brothers Ltd., Borehamwood, Hertfordshire, Eng.

```
procedure  find (A,M,N,K); value M,N,K;
             array A; integer M,N,K;
comment  Find will assign to A [K] the value which it would
have if the array A [M:N] had been sorted. The array A will be
partly sorted, and subsequent entries will be faster than the first;
```

Communications of the ACM 321

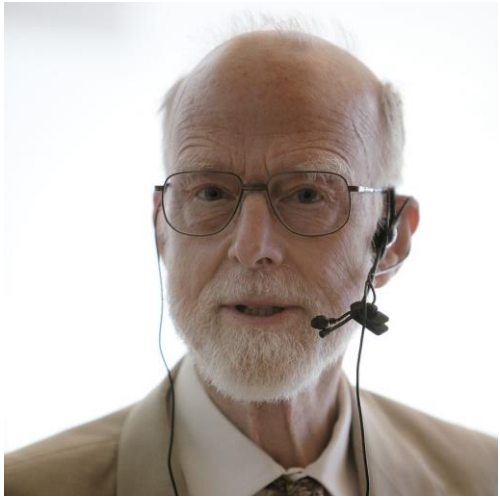
为什么？

因为1961年系统提供了递归抽象！

函数quicksort调用自身；quicksort也是自身的模块

这个递归程序能够无缝衔接自动执行

— Tony Hoare, 1980图灵奖演说：皇帝的旧衣



Acu-Exams

抽象化（**Abstraction**）：少数精心构造的计算抽象可描述万千应用
模块化（**Modularity**）：多个模块有规律地组合成为计算系统
无缝衔接（**Seamless Transition**）：计算过程在系统中流畅地执行

谢谢 Thank You

Q&A

zxu@ict.ac.cn

中科院计算所
INSTITUTE OF COMPUTING TECHNOLOGY

4. 正确、巧妙、实用的计算过程

Acu-Exams

- 自动执行（**A**）的逐步过程非常强大
  - 计算机科学的五个为什么
    - 为什么复杂的计算过程能够正确执行？
    - 为什么计算机科学有诸多奇妙？
    - 为什么计算机科学技术能够实用？
    - 为什么人们能够灵活使用诸多应用？
    - 为什么逐步计算过程渗透如此之广？

五者同时成立

正确性（CU）

巧妙性（EX）

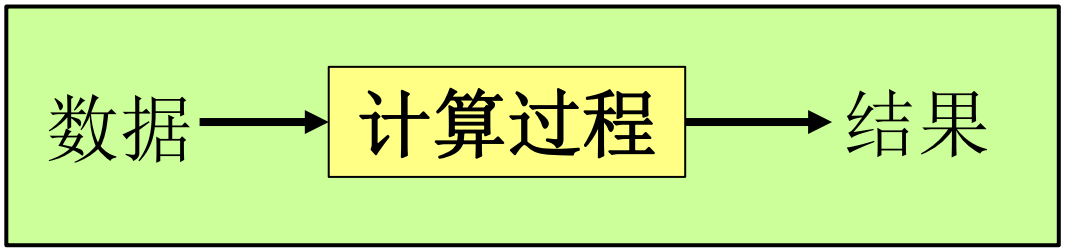
实用性（AMS）

方便性

广泛性

为什么妙？
不断解决**图灵问题**；
出现了巧妙的模型、算法和系统，体现三大妙处

为什么正确？
比特精准的
逻辑思维



为什么使用方便？
不断解决**布什问题**；出现了丰富的使用模式

为什么实用？速度快、成本低、可靠
不断解决**巴贝奇问题**
以自动执行的抽象为特色的系统思维，应对系统复杂性

4.1 自动执行（Acu-Exams） 用算法和程序刻画计算过程

- 如，计算斐波那契数F(10)
- 算法，往往用伪代码表示
  - 写给人看的
- 程序，三类编程语言表示
  - 计算机语言表达的算法
  - 写给计算机看的
  - 代码是一段程序
- 高级语言程序，fib-10.go
  - 与处理器无关
- 汇编语言程序，fib-10.asm
  - 与处理器相关
    - 如x86汇编语言程序
- 二进制程序，fib-10
  - 与处理器相关
    - 如x86二进制程序

100字节

人工编程

391字节

编译

go build fib-10.go

22,653字节

汇编

2,011,793字节

```
Output F(10)                          // 算法，伪代码
where F(n) is defined as
    if (n=0 or n=1) then F(n)=n
    else F(n)=F(n-1)+F(n-2)
```

```
package main                          // Go程序
import "fmt“                          // fib-10.go
func main() {
   fmt.Println("F(10)=",fibonacci(10))
}
func fibonacci(n int) int {
   if n == 0 || n == 1 { return n }
   return fibonacci(n-1)+fibonacci(n-2)
}
```

```
…
MOVQ   (TLS), CX                      // 汇编程序
CMPQ   SP, 16(CX)                     // fib-10.asm
JLS    181
SUBQ   $96, SP
MOVQ   BP, 88(SP)
LEAQ   88(SP), BP
…
```

```
01111111 01000101 01001100 01000110
00000010 00000001 00000001 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000010 00000000 00000010 00000000
00000001 00000000 00000000 00000000
```

二进制程序

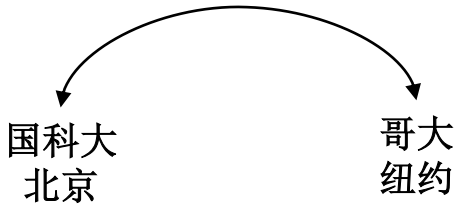
fib-10

Acu-Exams

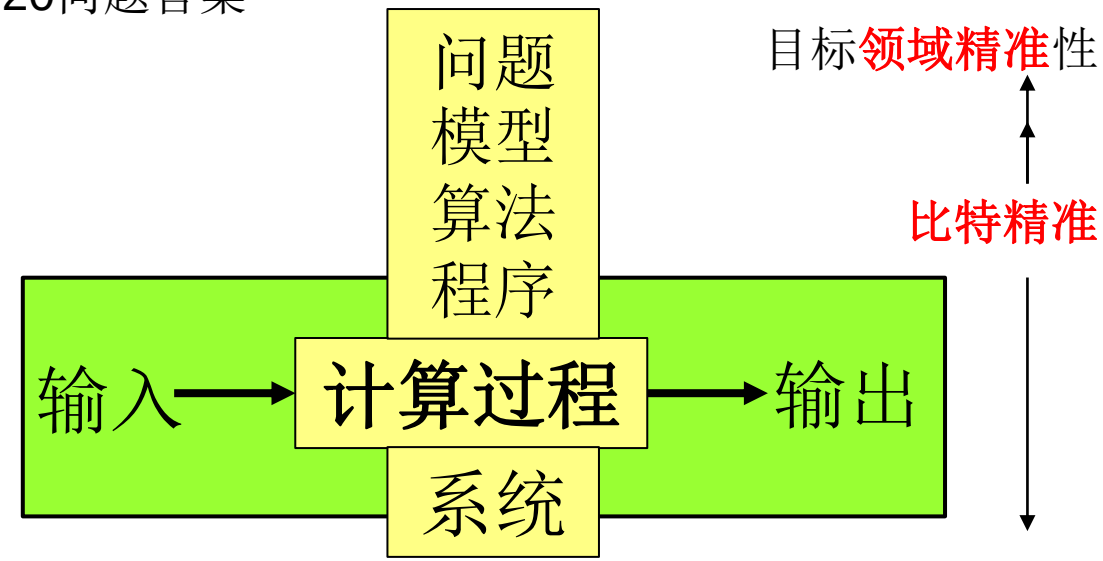
Automatic execution：二进制程序在计算机上自动执行

4.2 正确性（Acu-Exams）

- 实例：与国科大2022届同学商讨推荐信
  - 微信视频：
    跨越太平洋，万亿个操作的计算过程正确执行

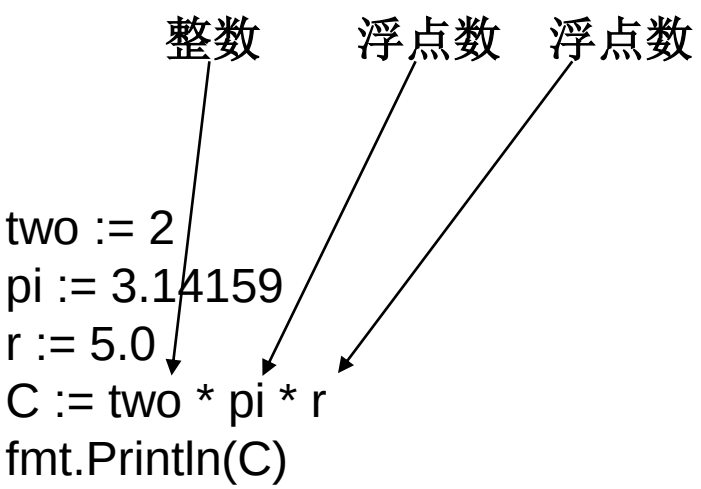


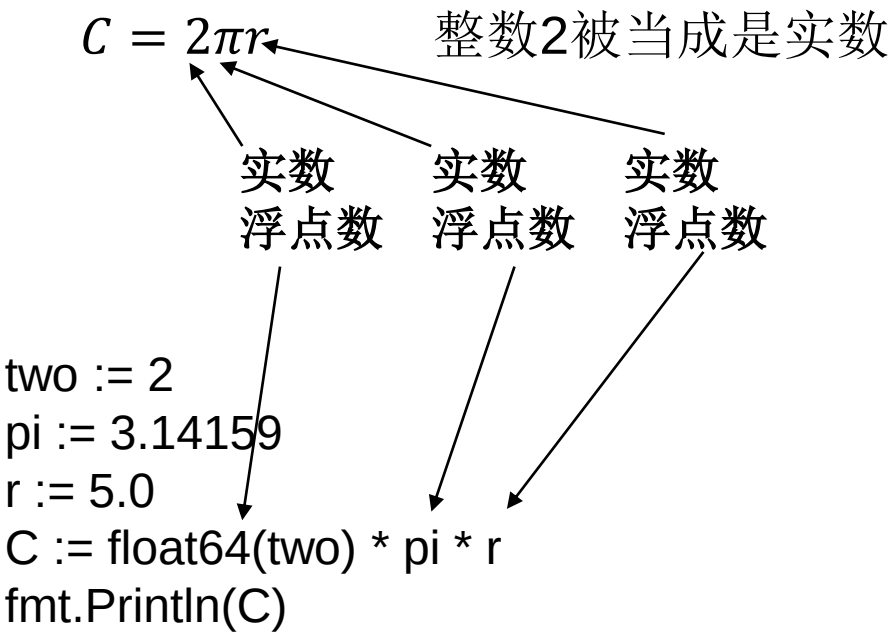
- **比特精准的逻辑思维**，支撑
  - 正确的程序
    - 正确的建模、算法和程序
  - 正确的输入数据
    - 难以保证，但有希尔伯特第20问题答案
      - 很多实际问题是适定问题
        Well-Posed Problems
    - 警惕“垃圾进-垃圾出”
      - Garbage in, garbage out
  - 正确的系统执行
    - 无缝衔接的自动执行



数学精准 与 计算机科学的比特精准 对比

- 数学计算强调数学精准，可有明确定义的异常情况
  - $Y = X/0$；实数计算出现 $A = \sqrt{-2.3}$；
  - 数据类型失配（type mismatch）
    - 整数 + 三角形 = ？（无意义）
- 计算机科学继承数学的异常，出现异常可能报错
  - $Y = X/0$（**除零异常**）；实数计算出现 $A = \sqrt{-2.3}$ （**NaN**）
  - 数据类型失配：周长公式 $C = 2\pi r$ 整数2被当成是实数
    - 错误和正确的计算机代码





计算机科学的计算还可能出现下列误差

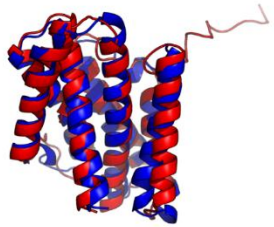
- 建模误差
  - 如，蛋白质结构预测精GDT_TS>50%
  - 应用领域决定何种误差是可以容忍的
- 截止误差（roundoff error）
  - fib.go演示　　F(50)=12586269025
  - fib.binet.go演示　　F(50)=1.258626902499998e+10 = 12586269024.999998
- 溢出（overflow）
  - 编译运行fib.dp.go，我们得到（演示）

F(91) = 4660046610375530309

F(92) = 7540113804746346429

F(93) = -6246583658587674878（错误）

= 12200160415121876738 - 2^{64}



$$F(n) = \frac{\varphi^n - (1-\varphi)^n}{\sqrt{5}},$$

其中 $\varphi = \frac{1+\sqrt{5}}{2} = 1.618\ldots$

- 64比特整数能够表示的最大值是 2^{63}-1 = 9223372036854775807
- F(93) = 12200160415121876738 太大了， 64比特整数类型装不下 溢出！

Acu-Exams

正确性（**Correctness**）：计算过程中每一个比特都是正确的（比特精准）

通用性（**Universality**）：计算机能够比特精准地求解任意可计算问题

4.3 巧妙性（Acu-Exams）

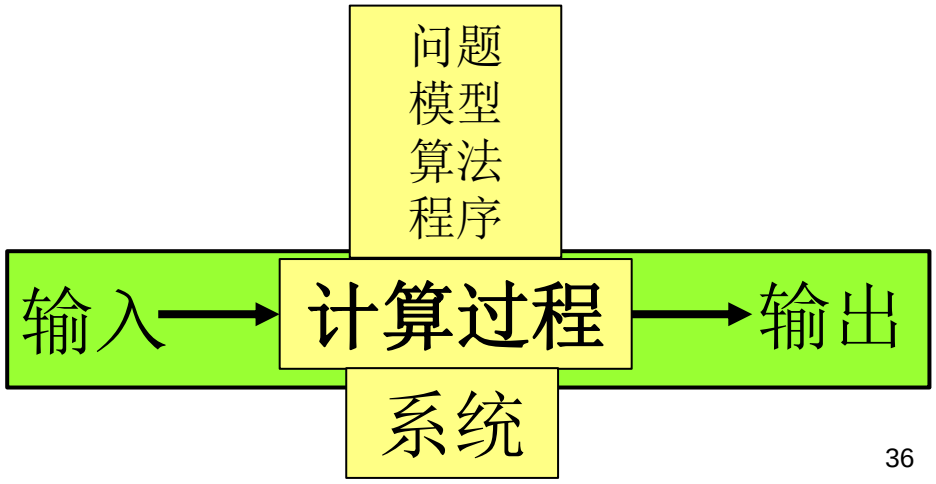
- 不断解决图灵问题
  - 如何用聪明的方法解决应用问题
- 算法思维：巧妙的建模、算法
  - 如，曾庆存老师的半隐式差分法
  - 如，随机排序算法quicksort
    - 简洁：只需几行伪代码
    - 快速：执行时间从$O(n^2)$降到$O(n\log n)$
    - 省空间：只需要最小内存空间
- 系统思维：巧妙的系统抽象
  - 例如，循环、递归、桥接模型
- 三个奇妙（wonders）
  - 指数之妙
  - 模拟之妙
  - 虚拟之妙

显式：$\frac{x^{n+1}-x^n}{\Delta t}=f(x^n)$

隐式：$\frac{x^{n+1}-x^n}{\Delta t}=f(x^{n+1})$

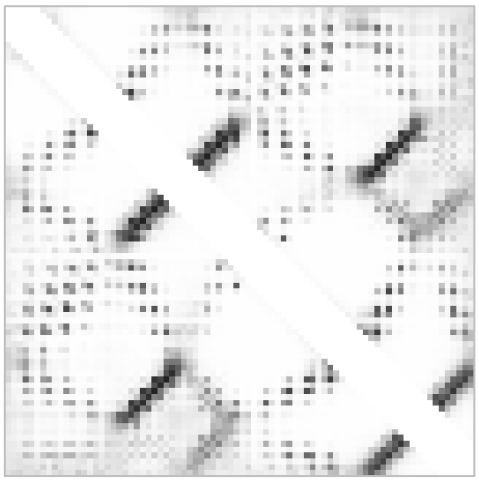
半隐式：$\frac{x^{n+1}-x^n}{\Delta t}=x^{n+1}+g(x^n)$

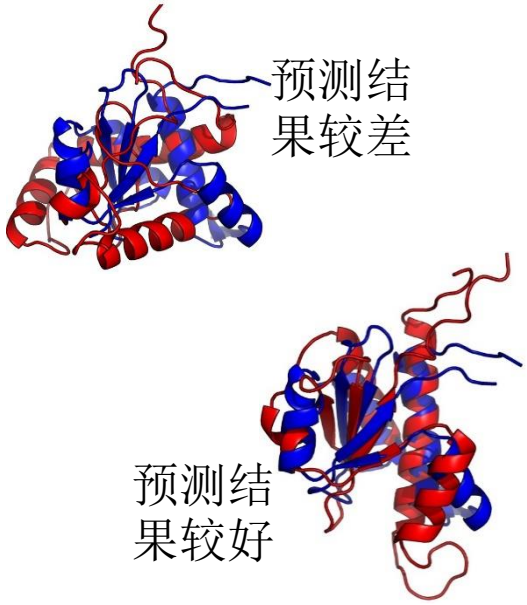
```
procedure  quicksort (A, M, N);
   array A; integer M, N;
begin integer I, J
   if M < N then begin   partition (A, M, N, I, J);
                         quicksort (A, M, J);
                         quicksort (A, I, N)
                 end
end  quicksort
```



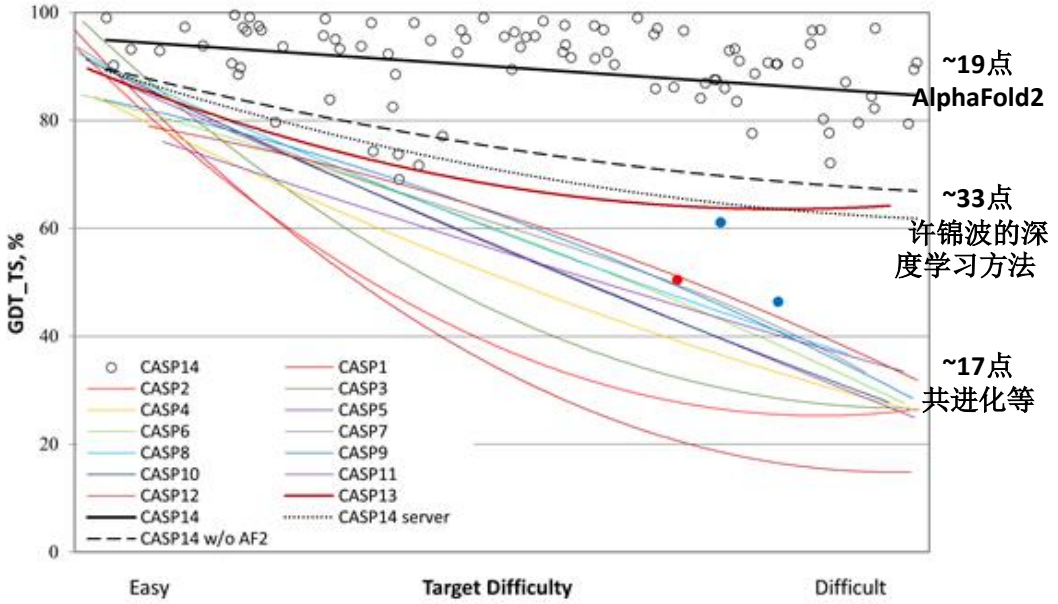
狭义图灵问题

- 如何实现智能应用？
  - 什么是通过图灵测试？
    - 提问者（人）通过5分钟的问答，正确辨别是计算机还是人在回答的概率小于70%，即不能区分人和计算机的概率大于30%
  - 智能应用例子
    - 下棋胜过人类棋手
    - 机器翻译
    - 图像识别
    - 自动驾驶
    - 机器人
    - 蛋白质折叠
      - 许锦波的工作
- 图灵问题的研究，催生了人工智能子领域





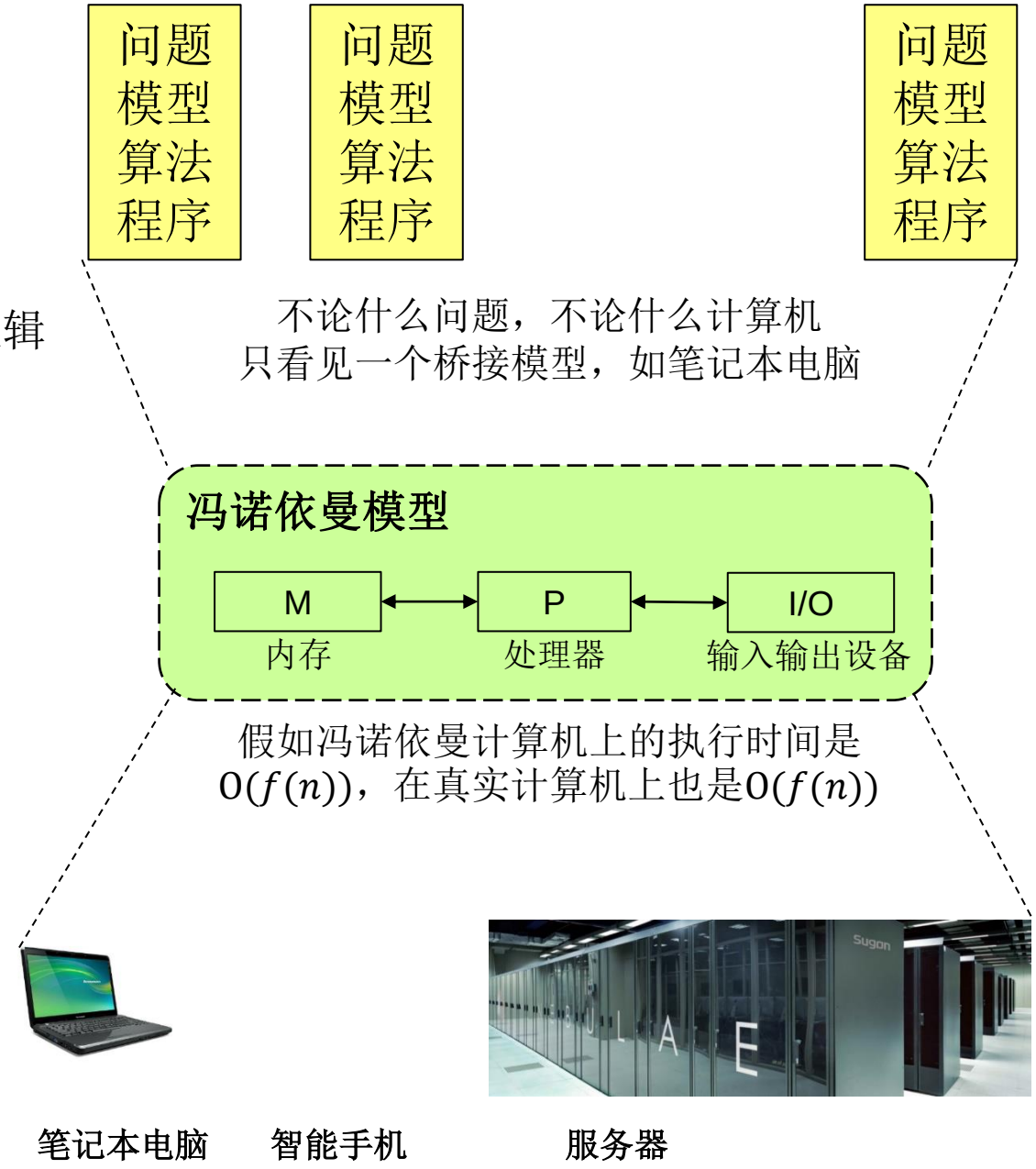
将蛋白质折叠的原子距离计算问题转换为图像语义分割问题



图片来源：许锦波教授

桥接模型

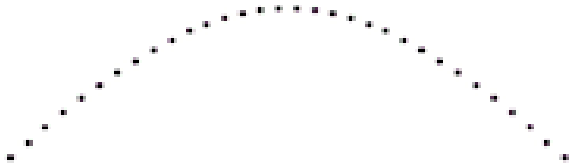
- 本课程学习五种
  理论计算机的入门知识
- 非桥接模型
  - **布尔逻辑**：命题逻辑与谓词逻辑
    - 无状态，无记忆
  - **图灵机**，及自动机
    - 有状态，有记忆
    - 真实计算机模拟开销：$O(n^4)$
  - **组合电路**，等价于命题逻辑
  - **时序电路**
    - 有状态，有记忆
- 桥接模型（bridging model）
  - **冯诺依曼模型**
    - 有状态，有记忆
    - 桥接计算机硬件与软件
      - 桥接开销： $O(1)$
    - 例如，Go语言加法器
      比图灵机加法器容易得多



模拟之妙

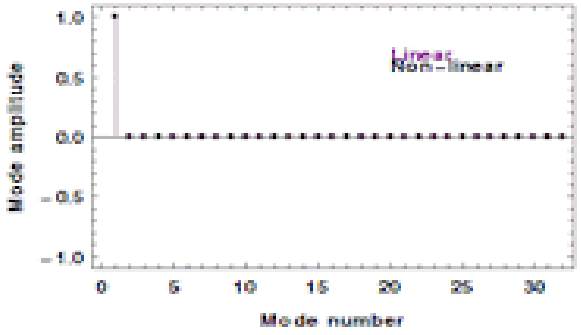
- 1953年，费米等发明计算机模拟
  - 开启了第三种科学研究方法
  - 在Maniac计算机上求解
  - By Mary Tsingou
  - “Let us refer from now on to the Fermi-Pasta-Ulam-Tsingou problem.”

Dauxois, *Physics Today*, 2008



Mary Tsingou in 1955

与她先生一起(2019)
91岁了！



“They thought nuclear energy was going to change the world,” she says, “but it’s the computers that have changed the world.”

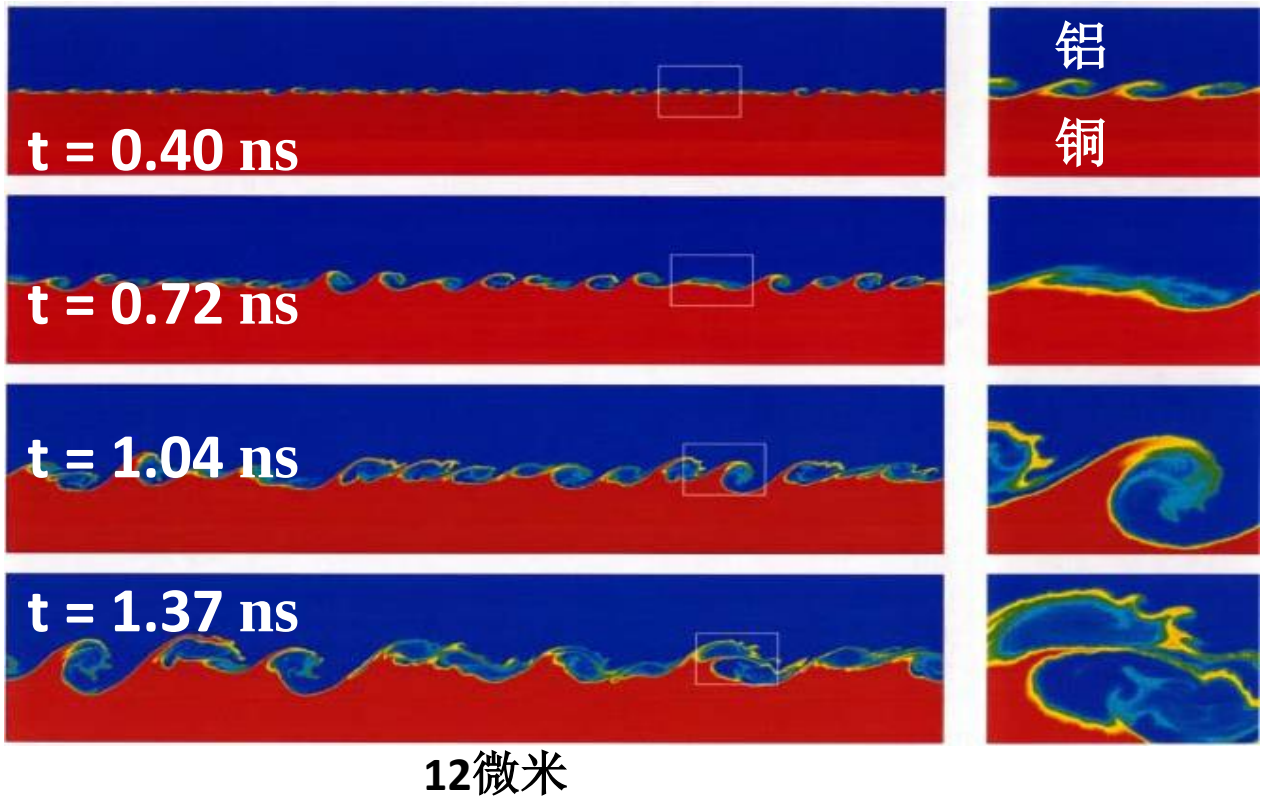
https://www.lanl.gov/discover/publications/national-security-science/2020-winter/mary-tsingou.shtml

en.wikipedia.org/wiki/Fermi–Pasta–Ulam–Tsingou_problem

See the invisible: Atoms in the Surf

使用计算机模拟90亿个原子的行为，重现开尔文-亥姆霍兹不稳定性 (演示)
高温: 2000K; 高速: 2000 m/s；模拟数纳秒行为耗费3600万CPU小时

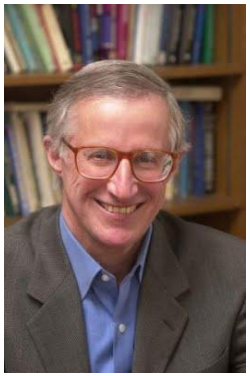
$A_i(t)$是原子i在t时刻的构象
时间步增量i++为皮秒级或飞秒级




```
for t=0; t<T; t++ {
  for i=0; t<9B; i++ {
    A_i(t+1) = Σ_{j=1}^{9B} f(A_i(t), A_j(t))
  }
}     // 9B=9,000,000,000
```


Richards D F, Krauss L D, Cabot W H, et al. (2008). Atoms in the Surf: Molecular Dynamics Simulation of the Kelvin-Helmholtz Instability Using 9 Billion Atoms. https://arxiv.org/abs/0810.3037 and www.youtube.com/watch?v=Wr7WbKODM2Q.

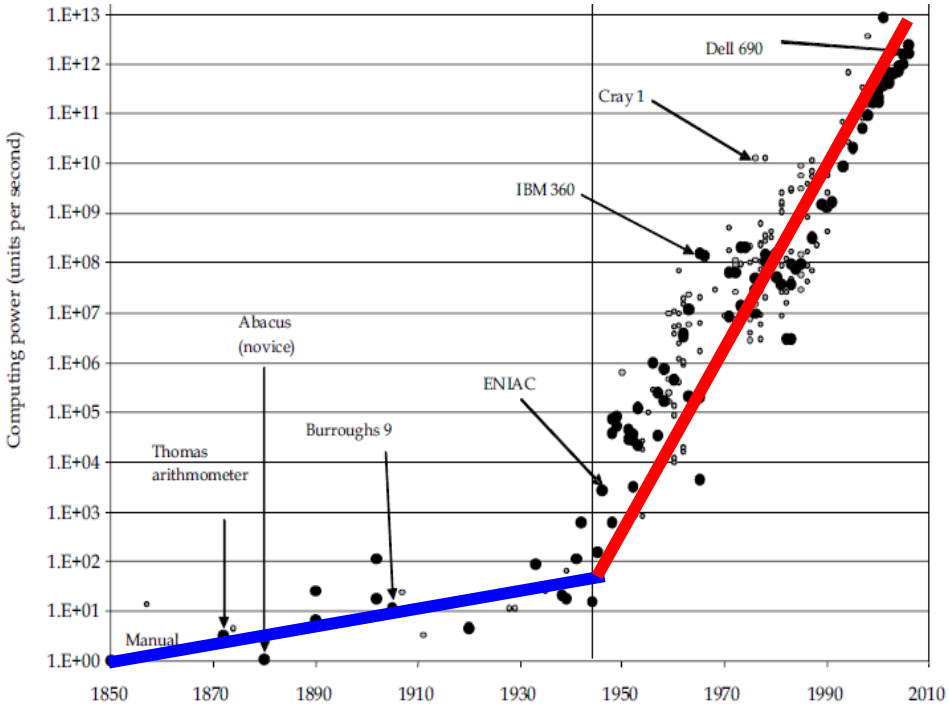
指数之妙



William Nordhaus
威廉•诺德豪斯

耶鲁大学教授
2018年诺贝尔经济学奖获得者

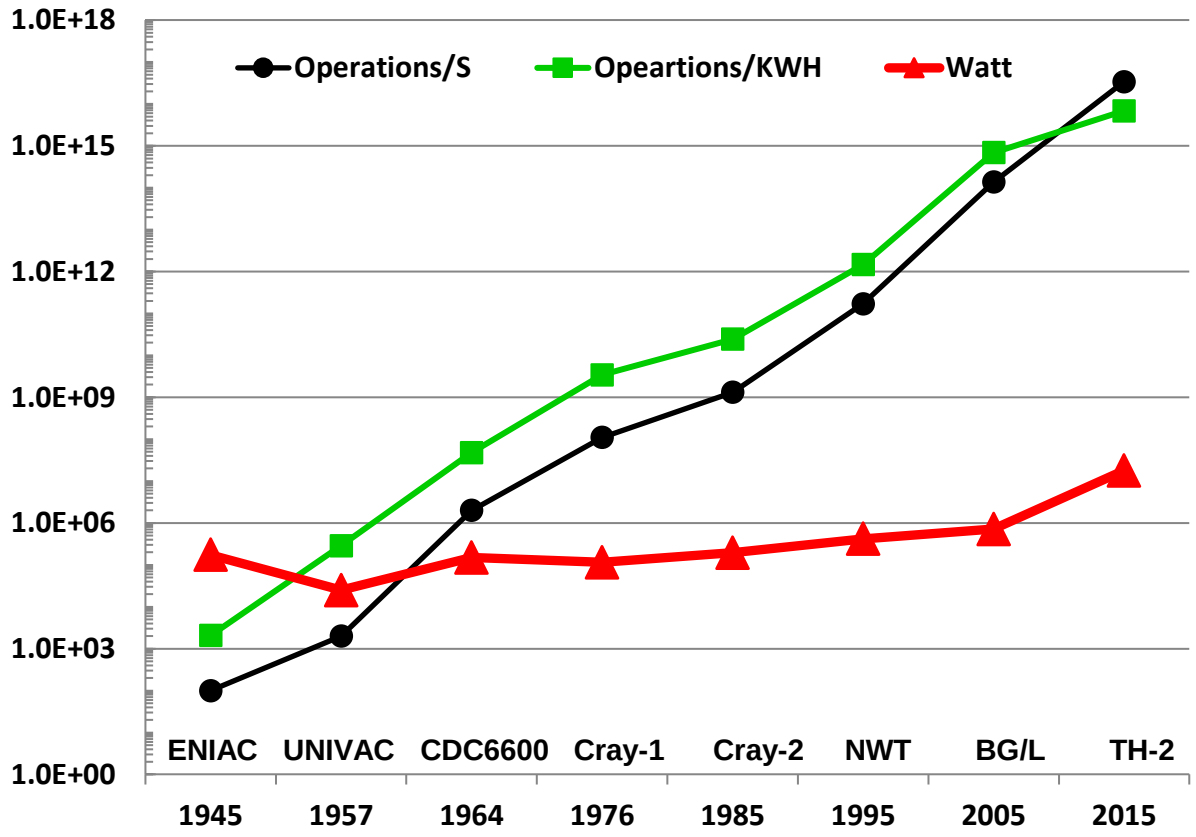
- 诺德豪斯观察
  - 计算速度是生产力
  - 计算速度增长趋势大体上可分为慢速增长（1850-1945）和快速增长（1946-2006）两个阶段
  - 慢速增长阶段：1850年以前主要是手工计算，随后市场上出现机械计算机和机电计算机等半自动计算方式，使得计算速度在第一阶段的95年间增长了数百倍
  - 快速增长阶段：60年间计算机速度增长了上千亿倍，平均每年大约增长50%
- 有了自动执行之后，计算速度随时间指数增长
  - 自动执行是指数之妙的重要原因



Nordhaus W D. Two Centuries of Productivity Growth in Computing. Journal of Economic History, 2007, 67(1):128-159.

2005年后的指数增长障碍

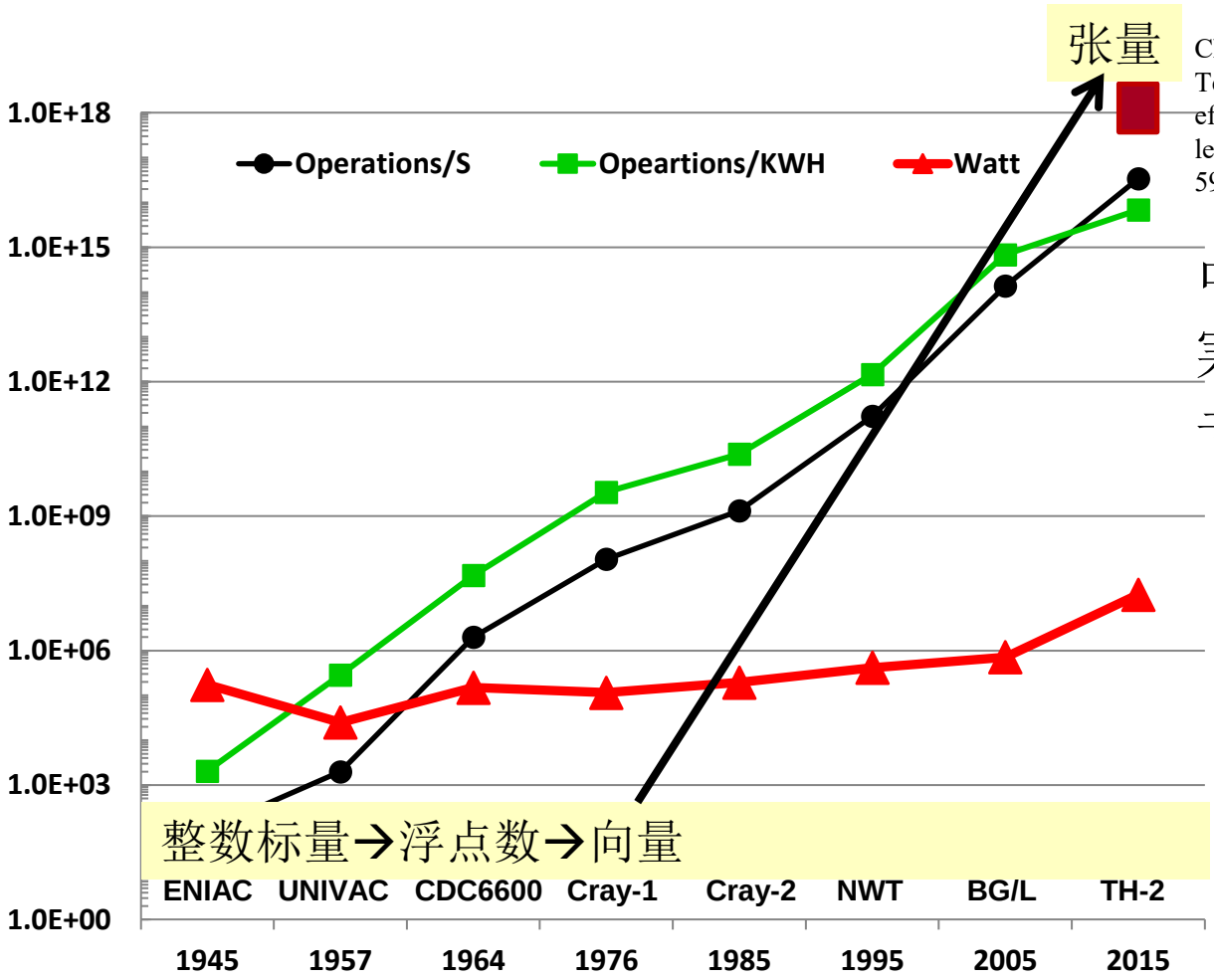
- 能效挑战
  - 70年来第一次出现：能效增长落后与速度增长
    - 将能效曲线扳回上方！可能吗？



Growth trends of computing speed, energy efficiency, and power consumption of the world's fastest computers (supercomputers) from 1945 to 2015. Special thanks to Drs. Gordon Bell, Jonathan Koomey, Dag Spicer and Ed Thelen for providing data for the first three computers.

近年来的指数增长趋势

- 在提供越来越强大的抽象的同时，
  计算机系统速度和能效持续指数增长



Chen, Y., Chen, T., Xu, Z., Sun, N., & Temam, O. (2016). DianNao family: energy-efficient hardware accelerators for machine learning. Communications of the ACM, 59(11), 105-112.

中科院计算所的寒武纪机器学习专用处理器研究

Growth trends of computing speed, energy efficiency, and power consumption of the world's fastest computers (supercomputers) from 1945 to 2015. Special thanks to Drs. Gordon Bell, Jonathan Koomey, Dag Spicer and Ed Thelen for providing data for the first three computers.

4.4 实用性（Acu-Exams）

- CS长期追求：不断解决**巴贝奇问题**
  - 如何构建实用的计算机？
  - 速度快、成本低、可靠、能够应对系统复杂性

- 以**自动执行的抽象**为特色的系统思维
  - 向上提供系统抽象，应对复杂性
  - 将系统抽象描述的程序变换为最底层的硬件操作
  - 比特精准地高效地自动执行

- 例子：霍尔悖论

霍尔悖论

- 1959年在莫斯科国立大学当访问学生时产生快排思想
- 1960年在Elliott Brothers公司为Elliott 803计算机开发排序程序时，发明并实现快排算法。但是，

“**Very difficult to explain**” 很难向他人说明快排算法

- 1961年发表，非常简洁易懂（只有半页，8行伪代码）

Hoare, C. A. R. (1961). "Algorithm 64: Quicksort". Comm. ACM. 4 (7): 321

ALGORITHM 64
QUICKSORT
C. A. R. HOARE
Elliott Brothers Ltd., Borehamwood, Hertfordshire, Eng.

```
procedure  quicksort (A,M,N); value M,N;
             array A; integer M,N;
comment  Quicksort is a very fast and convenient method of
sorting an array in the random-access store of a computer. The
entire contents of the store may be sorted, since no extra space is
required. The average number of comparisons made is 2(M−N) ln
(N−M), and the average number of exchanges is one sixth this
amount. Suitable refinements of this method will be desirable for
its implementation on any actual computer;
begin      integer I,J;
           if M < N then begin partition (A,M,N,I,J);
                               quicksort (A,M,J);
                               quicksort (A, I, N)
                         end
end        quicksort
```

与教科书版本基本一样

ALGORITHM 65
FIND
C. A. R. HOARE
Elliott Brothers Ltd., Borehamwood, Hertfordshire, Eng.

```
procedure  find (A,M,N,K); value M,N,K;
             array A; integer M,N,K;
comment  Find will assign to A [K] the value which it would
have if the array A [M:N] had been sorted. The array A will be
partly sorted, and subsequent entries will be faster than the first;
```

Communications of the ACM 321

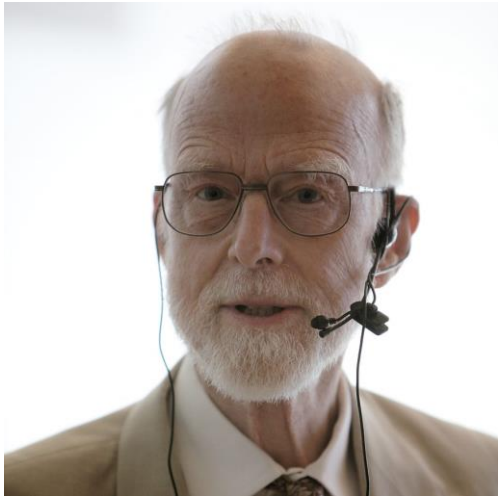
为什么？

因为1961年系统提供了递归抽象！

函数quicksort调用自身；quicksort也是自身的模块

这个递归程序能够无缝衔接自动执行

— Tony Hoare, 1980图灵奖演说：皇帝的旧衣



Acu-Exams

抽象化（**Abstraction**）：少数精心构造的计算抽象可描述万千应用

模块化（**Modularity**）：多个模块有规律地组合成为计算系统

无缝衔接（**Seamless Transition**）：计算过程在系统中流畅地执行

正确、巧妙、实用的计算过程 Acu-Exams

- 斐波那契数列习题3（大学）：求F(1,000,000,000)
- 对比fib.go和fib.dp.go
  - 程序fib.dp.go的速度快得多，但求F(93)出错（一个64位整数装不下，溢出）
- 重新建模得到程序fib.matrix.go，利用了斐波那契数列的一个矩阵性质
  - $\begin{bmatrix}1 & 1\\ 1 & 0\end{bmatrix}^n = \begin{bmatrix}F(n+1) & F(n)\\ F(n) & F(n-1)\end{bmatrix}$ $A^{16} = (((A^2)^2)^2)^2$
  - 并通过矩阵平方求矩阵n次方，将计算复杂度从$O(n)$降为$O(\log n)$
- 程序fib.matrix.go体现了正确、巧妙、实用的斐波那契数列F(n)计算过程
  - 利用系统提供的任意长整数数据类型big.Int抽象，避免了溢出（**Acu-Exams**）
  - **计算F(5,000,000)仅需要4秒；计算F(1,000,000,000)需要十多万秒**
    - F(1,000,000,000)是一个很大的正整数，有2000多万个十进制数字

Acu-Exams
构造性（**Effectiveness**）：人们构造出聪明的方法让计算机有效地解决问题
复杂度（Complexity）：这些聪明的方法（算法）具备时间/空间复杂度

求解F(n)的执行时间（秒）

n	fib.go $O(2^n)$	fib.dp.go $O(n)$	fib.matrix.go $\sim O(\log n)$
50	725	0.059	0.000012
500	出错、极慢	出错	0.000022
5,000,000	出错、极慢	出错	4.13
1,000,000,000	出错、极慢	出错、很慢	187,160

4.5 为什么使用越来越方便？

- CS长期追求：不断解决**布什问题**
  - 如何方便有效地使用计算机？为此，需要研究
    - 思考的人与人类知识的关系
    - 如何连接用户、计算机、信息
- 具象化为**使用模式**
  - 用户社区 + 信息的组织模式 + 人机交互模式
- 人机交互取得的大进展，人们提出了多种使用模式
  - Batch　　批处理
  - Interactive　　交互式处理
  - Personal computing　　个人计算
  - GUI　　图形界面
  - Multimedia　　多媒体计算
  - Portable computing　　便携式计算
  - Network computing　　网络计算、云计算
  - Mobile Internet　　移动互联网
    - 人机界面：键盘鼠标显示器的图形界面 → 触摸屏

4.7 为什么计算机科学渗透广

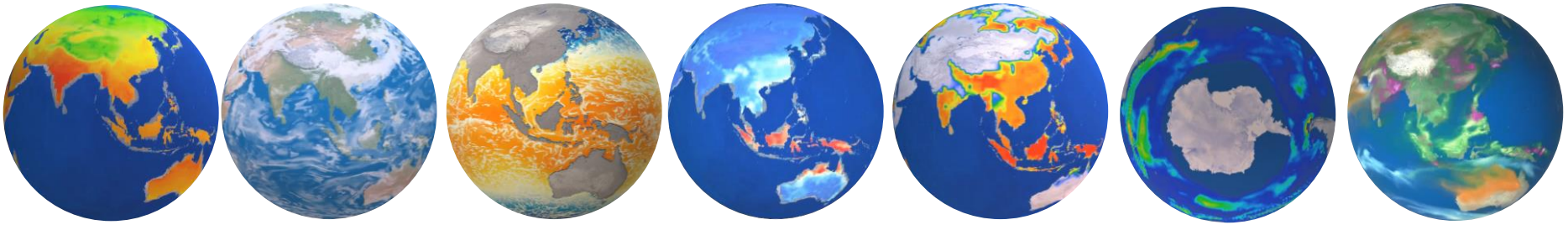
- 理论原因：图灵-邱奇命题
  - 图灵机可求解任何可计算问题；无穷内存的笔记本电脑 ≈ 图灵机
  - 人们也在不断地应用计算机解决图灵不可计算问题
- 应用原因
  - 人类正在进入信息社会：农业社会 → 工业社会 → 信息社会
  - 网络效应（梅特卡夫定律、里德定律、病毒性市场）
    - 2021年1月数据：全球互联网渗透率已经超过53%
    - 平均每个用户每天使用互联网接近7小时；巴西最高，超过10小时

Year	Worldwide Internet Users/Population
1996	1%
2001	8%
2006	18%
2011	31%
2016	44%
2021	53%

Country	Daily Time Spent Using Internet 2021
Brazil	10:08 (10 hours 08 minutes)
Indonesia	08:25
Russia	07:52
USA	07:11
Worldwide	06:54
Germany	05:26
China	05:22
Japan	04:25

Data sources: International Telecommunications Union, Datareportal

各学科过程可被转换成计算过程，即计算机应用



土壤温度　大气云量　海洋温度　蒸散发　植被覆盖度　海洋生产力　气溶胶

“中国科学院地球系统模式”（CAS-ESM）的部分应用
资料来源：朱江教授

科学、技术、经济、社会的各种问题和过程

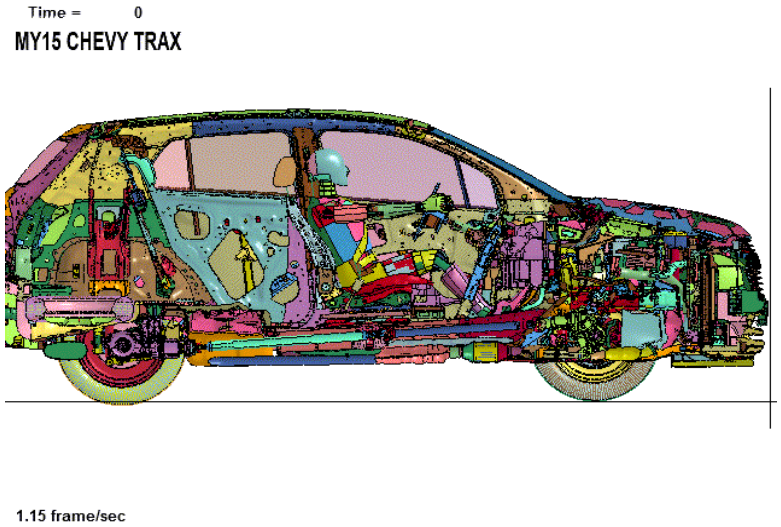
各领域的专业表达

都可以用该领域的语言表示

数字无穷性假说
+
计算透镜假说

有穷数字符号的组合可表达无穷语言
+
Nature computes.
Society computes.

计算问题和计算过程



https://img.auto-testing.net/testingimg/202001/16/191523291.gif

为什么渗透广？四个假说

- 乔姆斯基**数字无穷性**原理
  - 任何人类语言的无穷含义都可以采用有限符号表示。因此，都可以通过有限的数字符号集表示，由计算机科学处理。
- 卡普**计算透镜**命题
  - 很多自然界和人类社会的过程也是计算过程。用计算机科学研究，可以产生新视角、新方法、新价值。
- 巴贝扬**黄金隐喻**
  - 计算速度计算速度是像黄金一样的硬通货，可以换成任何其他东西，包括新的功能、更好的性能、更容易使用、更好的产品、成本更低的服务，等等。
- 布当**蜜蜂隐喻**
  - 信息技术就像蜜蜂一样，产生的两类价值，间接价值（授粉）比直接价值（蜂蜜）大得多
    - 授粉的经济价值是蜂蜜的经济价值的28-373倍
    - 2016年全球数字经济大约为11.5~24万亿美元，远大于该年的信息产业市场（约3.4~4.3万亿美元）

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。