AI2615: 算法设计与分析

上海交通大学

张驰豪

2022年2月18日

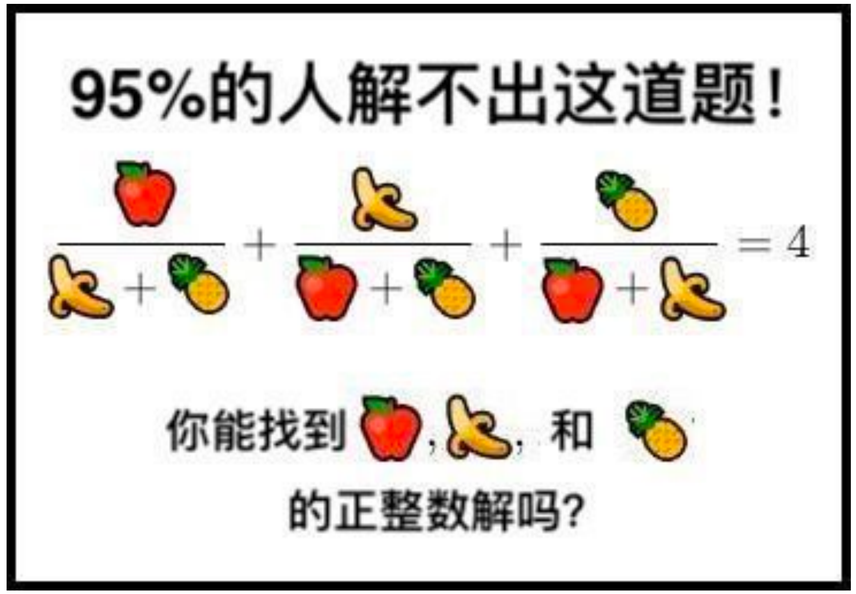
课程信息

- 每周五 12:55 - 15:40 @ 下院 412
- 任课教师：张驰豪（http://chihaozhang.com, chihao@sjtu.edu.cn）
- 助教：王玉林（sky46262@sjtu.edu.cn），负责上机作业
  薛峥嵘（xuezhengrong@sjtu.edu.cn），负责书面作业
  朱梓臣（JamesZhutheThird@sjtu.edu.cn），负责书面作业
- 课程主页：http://chihaozhang.com/teaching/Algo2022spring
- Office hour: 每周一晚上7点-10点@软件学院1402-2

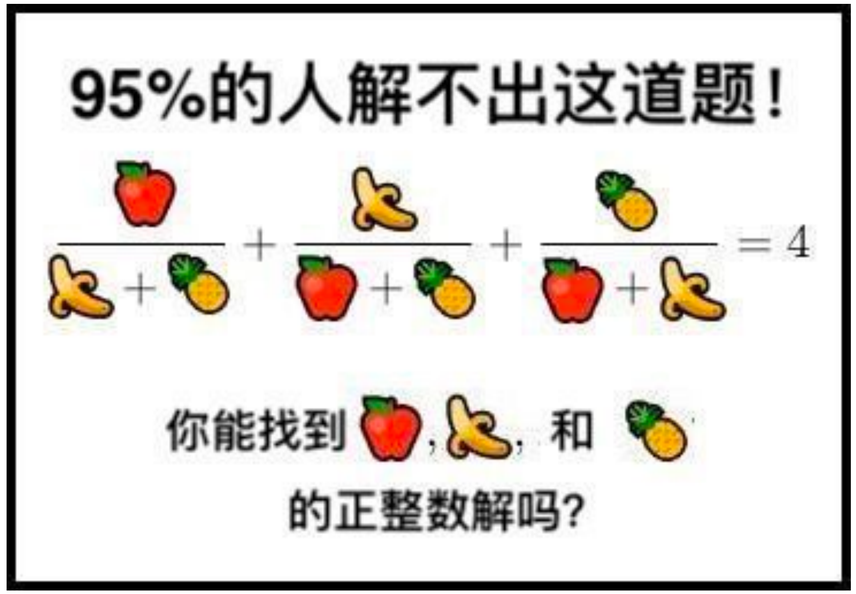
- 总共五次编程作业，六次书面作业
- 编程作业通过 Online Judge 发布
- 最终成绩=55（作业）+15（期中考试）+30（期末考试）
- 教材：*Algorithms*, S. Dasgupta, C. Papadimitriou, U. Vazirani

什么是算法

什么是算法

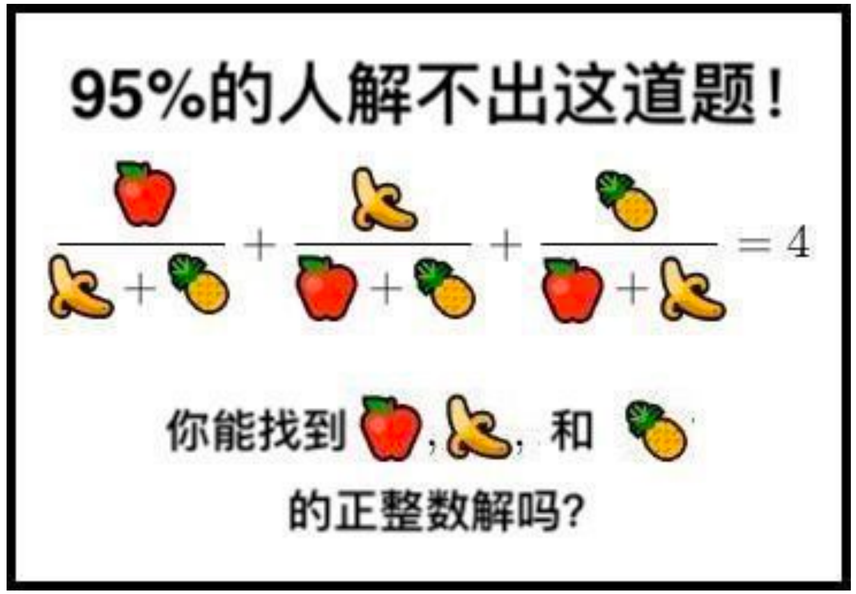


什么是算法



你能写一个程序/算法解这类问题吗?

什么是算法



你能写一个程序/算法解这类问题吗?

= 154476802108746166441951315019919837485664325669565431700026634898253202035277999

= 368751317941299998271978115652254748254929799689719709962831374716372246340555 79

= 437361267792869725786125260237139015281653755816161361862143799337842346777 2036

希尔伯特第十问题

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

David Hilbert

希尔伯特第十问题

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

David Hilbert

希尔伯特第十问题 [编辑]

维基百科，自由的百科全书

希尔伯特的**第十个问题**，就是不定方程（又称为丢番图方程）的可解答性。这是希尔伯特于1900年在巴黎的国际数学家大会演说中，所提出的23个重要数学问题的第十题。

这个问题是问，对于任意多个未知数的整系数不定方程，要求给出一个可行的方法（verfahren），使得借助于它，通过有限次运算，可以判定该方程有无整数解。

这里德文的方法（verfahren），就是英文所谓的算法（algorithm）。对于算法的概念我们是不陌生的，例如远在古希腊时代，人们就知道可以使用辗转相除法，求两个自然数的最大公约数。还有，任给一个自然数，也存在着一个方法，在有限步骤内，可以判定这个数是不是质数。

虽然人们很早就有了算法的朴素概念，但对于到底什么是可行的计算，仍没有精确的概念。一个问题的可解与不可解究竟是什么含意，当时的人们还不得而知。然而为了研究第十问题，必须给予算法精确化的观念。这点还有赖于数理逻辑学对可计算性理论的发展，才得以实现。

希尔伯特第十问题

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

David Hilbert

希尔伯特第十问题 [编辑]

维基百科，自由的百科全书

希尔伯特的**第十个问题**，就是不定方程（又称为丢番图方程）的可解答性。这是希尔伯特于1900年在巴黎的国际数学家大会演说中，所提出的23个重要数学问题的第十题。

这个问题是问，对于任意多个未知数的整系数不定方程，要求给出一个可行的方法（verfahren），使得借助于它，通过有限次运算，可以判定该方程有无整数解。

这里德文的方法（verfahren），就是英文所谓的算法（algorithm）。对于算法的概念我们是不陌生的，例如远在古希腊时代，人们就知道可以使用辗转相除法，求两个自然数的最大公约数。还有，任给一个自然数，也存在着一个方法，在有限步骤内，可以判定这个数是不是质数。

虽然人们很早就有了算法的朴素概念，但对于到底什么是可行的计算，仍没有精确的概念。一个问题的可解与不可解究竟是什么含意，当时的人们还不得而知。然而为了研究第十问题，必须给予算法精确化的观念。这点还有赖于数理逻辑学对可计算性理论的发展，才得以实现。

“精确的定义算法”

希尔伯特的“判定性问题”

希尔伯特的“判定性问题”

Hilbert’s Entscheidungsproblem:

希尔伯特的“判定性问题”

Hilbert’s Entscheidungsproblem:

“是否存在一个算法/程序，能够自动地给出一个定理的证明？”

希尔伯特的“判定性问题”

Hilbert’s Entscheidungsproblem:

“是否存在一个算法/程序，能够自动地给出一个定理的证明？”

人工智能能够代替数学家吗？

希尔伯特的“判定性问题”

Hilbert's Entscheidungsproblem:

“是否存在一个算法/程序，能够自动地给出一个定理的证明？”

人工智能能够代替数学家吗？

In 1936, Alonzo Church and Alan Turing published independent papers[2] showing that a general solution to the *Entscheidungsproblem* is impossible, assuming that the intuitive notion of "effectively calculable" is captured by the functions computable by a Turing machine (or equivalently, by those expressible in the lambda calculus). This assumption is now known as the Church–Turing thesis.

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

希尔伯特的“判定性问题”

Hilbert's Entscheidungsproblem:

“是否存在一个算法/程序，能够自动地给出一个定理的证明？”

人工智能能够代替数学家吗？

In 1936, Alonzo Church and Alan Turing published independent papers[2] showing that a general solution to the *Entscheidungsproblem* is impossible, assuming that the intuitive notion of "effectively calculable" is captured by the functions computable by a Turing machine (or equivalently, by those expressible in the lambda calculus). This assumption is now known as the Church–Turing thesis.

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

Alonzo Church：
Lambda演算

希尔伯特的“判定性问题”

Hilbert's Entscheidungsproblem:

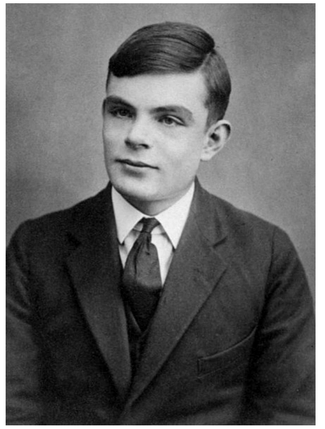
“是否存在一个算法/程序，能够自动地给出一个定理的证明？”

人工智能能够代替数学家吗？

In 1936, Alonzo Church and Alan Turing published independent papers[2] showing that a general solution to the *Entscheidungsproblem* is impossible, assuming that the intuitive notion of "effectively calculable" is captured by the functions computable by a Turing machine (or equivalently, by those expressible in the lambda calculus). This assumption is now known as the Church–Turing thesis.

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

Alonzo Church:
Lambda演算



Alan Turing:
图灵机

计算问题

计算问题

计算问题：$f: \{0,1\}^* \to \{0,1\}^*$

计算问题

计算问题：$f: \{0,1\}^* \to \{0,1\}^*$

- 表示一个数是不是素数：

计算问题

计算问题：$f:\{0,1\}^* \to \{0,1\}^*$

- 表示一个数是不是素数：

$$\forall x \in \mathbb{N},\quad f((x)_2) = \begin{cases} 1, & \text{if } x \text{ is a prime;} \\ 0, & \text{otherwise .} \end{cases}$$

计算问题

计算问题：$f:\{0,1\}^* \to \{0,1\}^*$

- 表示一个数是不是素数：

$$\forall x \in \mathbb{N},\quad f((x)_2) = \begin{cases} 1, & \text{if } x \text{ is a prime;} \\ 0, & \text{otherwise .} \end{cases}$$

- 表示一个丢番图方程的解：

计算问题

计算问题：$f:\{0,1\}^* \to \{0,1\}^*$

- 表示一个数是不是素数：

$$\forall x \in \mathbb{N},\quad f((x)_2) = \begin{cases} 1, & \text{if } x \text{ is a prime;} \\ 0, & \text{otherwise .} \end{cases}$$

- 表示一个丢番图方程的解：

$$\forall \text{ 丢番图方程的编码 } x,\quad f(x) = \text{ 方程解的编码}$$

图灵机

图灵机

For thousands of years, the term "computation" was understood to mean application of mechanical rules to manipulate numbers, where the person/machine doing the manipulation is allowed a *scratch pad* on which to write the intermediate results. The Turing machine is a concrete embodiment of this intuitive notion. Section 1.2.1

Computational Complexity, S. Arora, B. Barak

图灵机

For thousands of years, the term "computation" was understood to mean application of mechanical rules to manipulate numbers, where the person/machine doing the manipulation is allowed a *scratch pad* on which to write the intermediate results. The Turing machine is a concrete embodiment of this intuitive notion. Section 1.2.1

Computational Complexity, S. Arora, B. Barak

程序＝解问题的“套路”＋草稿纸

图灵机

For thousands of years, the term "computation" was understood to mean application of mechanical rules to manipulate numbers, where the person/machine doing the manipulation is allowed a *scratch pad* on which to write the intermediate results. The Turing machine is a concrete embodiment of this intuitive notion. Section 1.2.1

Computational Complexity, S. Arora, B. Barak

程序＝解问题的“套路”＋草稿纸

一个k−带图灵机是一个三元组 (Γ, Q, δ):

图灵机

For thousands of years, the term "computation" was understood to mean application of mechanical rules to manipulate numbers, where the person/machine doing the manipulation is allowed a *scratch pad* on which to write the intermediate results. The Turing machine is a concrete embodiment of this intuitive notion. Section 1.2.1

Computational Complexity, S. Arora, B. Barak

程序＝解问题的“套路”＋草稿纸

一个$k-$带图灵机是一个三元组 (Γ, Q, δ):

- $k-$带： k 张草稿纸
- Γ: 字符集
- Q : 状态集（算到哪一步了）
- $\delta : Q \times \Gamma^k \to Q \times \Gamma^{k-1} \times \{\mathtt{L}, \mathtt{S}, \mathtt{R}\}^k$

图灵机

For thousands of years, the term "computation" was understood to mean application of mechanical rules to manipulate numbers, where the person/machine doing the manipulation is allowed a *scratch pad* on which to write the intermediate results. The Turing machine is a concrete embodiment of this intuitive notion. Section 1.2.1

Computational Complexity, S. Arora, B. Barak

程序＝解问题的"套路"＋草稿纸

一个k−带图灵机是一个三元组 (Γ, Q, δ):

$\delta : Q \times \Gamma^k \to Q \times \Gamma^{k-1} \times \{\mathtt{L}, \mathtt{S}, \mathtt{R}\}^k$

- k−带： k 张草稿纸
- Γ: 字符集
- Q : 状态集（算到哪一步了）
- $\delta : Q \times \Gamma^k \to Q \times \Gamma^{k-1} \times \{\mathtt{L}, \mathtt{S}, \mathtt{R}\}^k$

图灵机

For thousands of years, the term "computation" was understood to mean application of mechanical rules to manipulate numbers, where the person/machine doing the manipulation is allowed a *scratch pad* on which to write the intermediate results. The Turing machine is a concrete embodiment of this intuitive notion. Section 1.2.1

Computational Complexity, S. Arora, B. Barak

程序＝解问题的"套路"＋草稿纸

一个$k-$带图灵机是一个三元组 (Γ, Q, δ):

当前输入

$$\delta : Q \times \Gamma^k \to Q \times \Gamma^{k-1} \times \{\mathtt{L}, \mathtt{S}, \mathtt{R}\}^k$$

- $k-$带： k 张草稿纸
- Γ: 字符集
- Q : 状态集（算到哪一步了）
- $\delta : Q \times \Gamma^k \to Q \times \Gamma^{k-1} \times \{\mathtt{L}, \mathtt{S}, \mathtt{R}\}^k$

图灵机

For thousands of years, the term "computation" was understood to mean application of mechanical rules to manipulate numbers, where the person/machine doing the manipulation is allowed a *scratch pad* on which to write the intermediate results. The Turing machine is a concrete embodiment of this intuitive notion. Section 1.2.1

Computational Complexity, S. Arora, B. Barak

程序＝解问题的"套路"＋草稿纸

一个$k-$带图灵机是一个三元组 (Γ, Q, δ):

- $k-$带： k 张草稿纸
- Γ: 字符集
- Q : 状态集（算到哪一步了）
- $\delta : Q \times \Gamma^k \to Q \times \Gamma^{k-1} \times \{\mathrm{L}, \mathrm{S}, \mathrm{R}\}^k$

当前输入

$$\delta : Q \times \Gamma^k \to Q \times \Gamma^{k-1} \times \{\mathrm{L}, \mathrm{S}, \mathrm{R}\}^k$$

当前算到哪了

图灵机

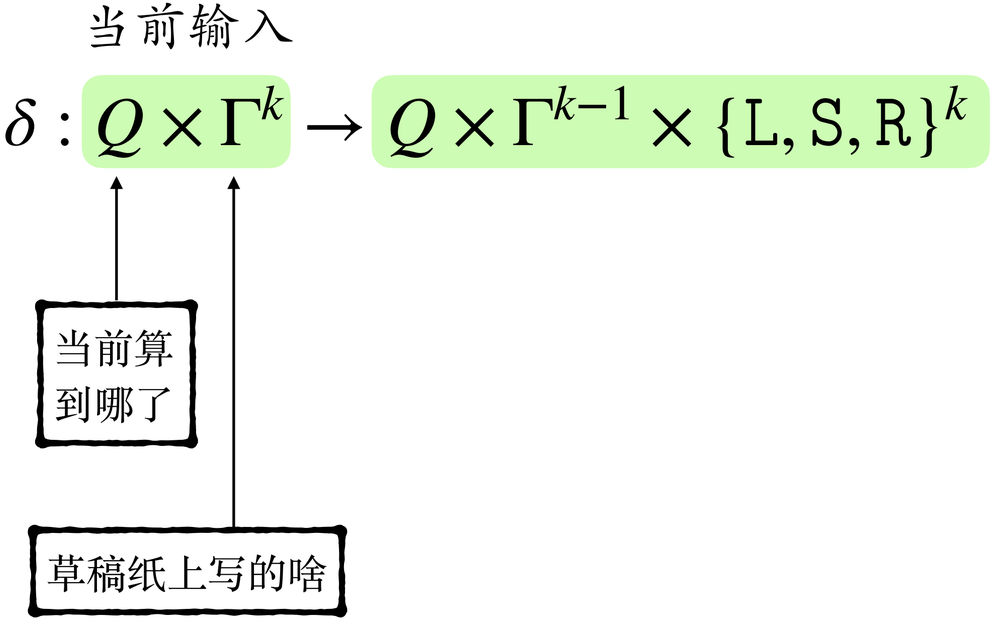
> For thousands of years, the term "computation" was understood to mean application of mechanical rules to manipulate numbers, where the person/machine doing the manipulation is allowed a *scratch pad* on which to write the intermediate results. The Turing machine is a concrete embodiment of this intuitive notion. Section 1.2.1

Computational Complexity, S. Arora, B. Barak

程序＝解问题的"套路"＋草稿纸

一个$k-$带图灵机是一个三元组 (Γ, Q, δ):

- $k-$带： k 张草稿纸
- Γ: 字符集
- Q : 状态集（算到哪一步了）
- $\delta : Q \times \Gamma^k \to Q \times \Gamma^{k-1} \times \{\mathrm{L}, \mathrm{S}, \mathrm{R}\}^k$



图灵机

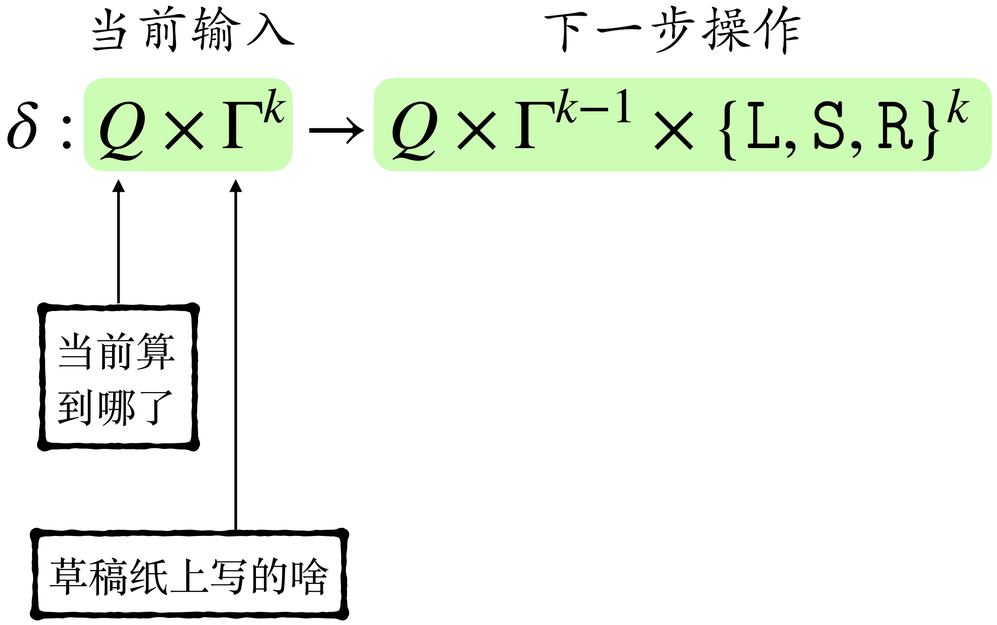
For thousands of years, the term "computation" was understood to mean application of mechanical rules to manipulate numbers, where the person/machine doing the manipulation is allowed a *scratch pad* on which to write the intermediate results. The Turing machine is a concrete embodiment of this intuitive notion. Section 1.2.1

Computational Complexity, S. Arora, B. Barak

程序＝解问题的"套路"＋草稿纸

一个$k-$带图灵机是一个三元组 (Γ, Q, δ):

- $k-$带： k 张草稿纸
- Γ: 字符集
- Q : 状态集（算到哪一步了）
- $\delta : Q \times \Gamma^k \to Q \times \Gamma^{k-1} \times \{\mathrm{L}, \mathrm{S}, \mathrm{R}\}^k$



图灵机

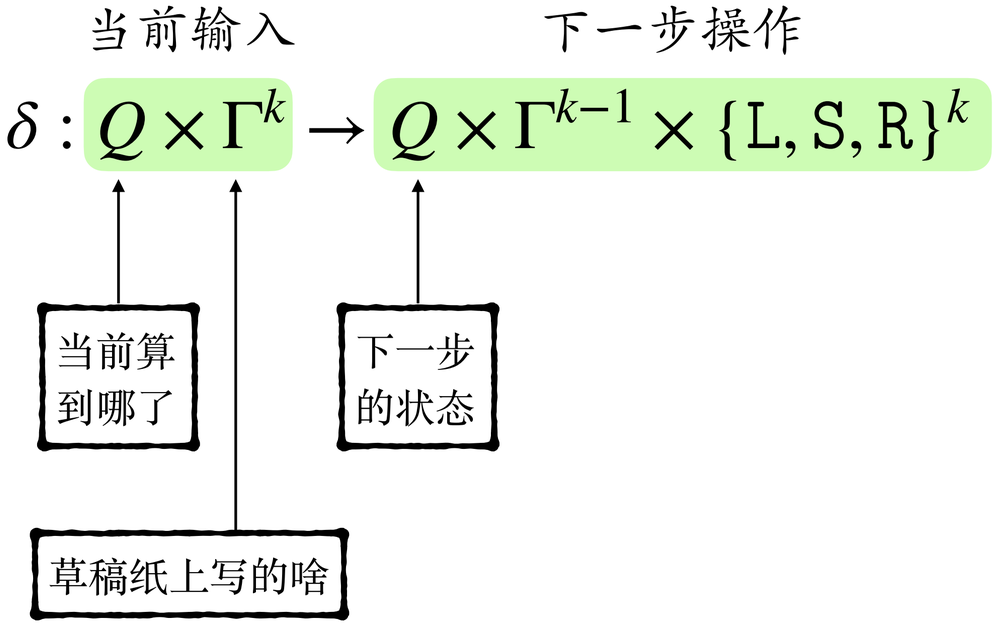
For thousands of years, the term "computation" was understood to mean application of mechanical rules to manipulate numbers, where the person/machine doing the manipulation is allowed a *scratch pad* on which to write the intermediate results. The Turing machine is a concrete embodiment of this intuitive notion. Section 1.2.1

Computational Complexity, S. Arora, B. Barak

程序＝解问题的"套路"＋草稿纸

一个$k-$带图灵机是一个三元组 (Γ, Q, δ):

- $k-$带： k 张草稿纸
- Γ: 字符集
- Q : 状态集（算到哪一步了）
- $\delta : Q \times \Gamma^k \to Q \times \Gamma^{k-1} \times \{\mathrm{L}, \mathrm{S}, \mathrm{R}\}^k$



图灵机

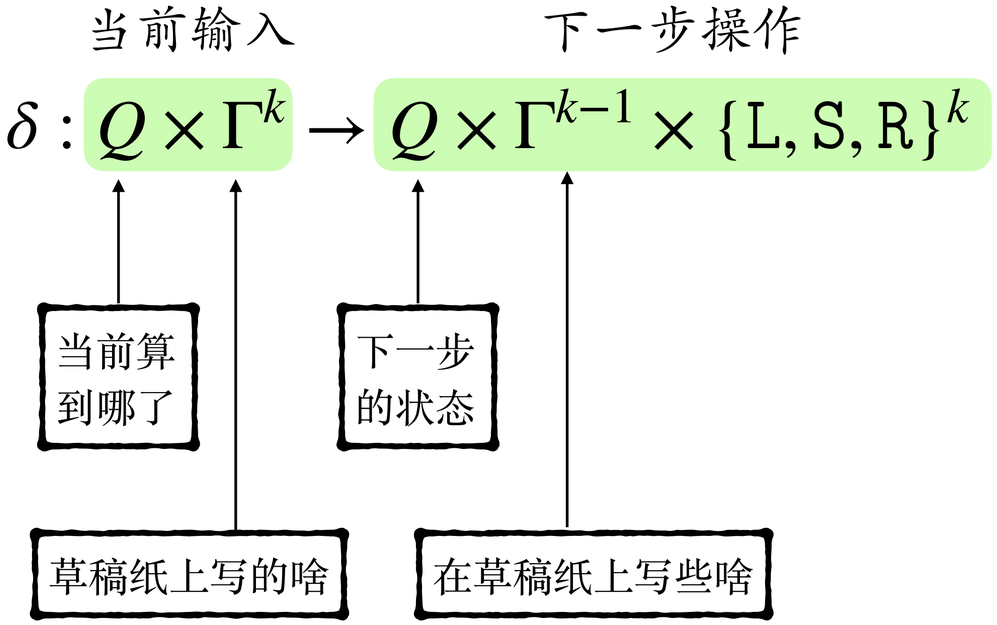
> For thousands of years, the term "computation" was understood to mean application of mechanical rules to manipulate numbers, where the person/machine doing the manipulation is allowed a *scratch pad* on which to write the intermediate results. The Turing machine is a concrete embodiment of this intuitive notion. Section 1.2.1

Computational Complexity, S. Arora, B. Barak

程序＝解问题的"套路"＋草稿纸

一个$k-$带图灵机是一个三元组 (Γ, Q, δ):

- $k-$带： k 张草稿纸
- Γ: 字符集
- Q : 状态集（算到哪一步了）
- $\delta : Q \times \Gamma^k \to Q \times \Gamma^{k-1} \times \{\mathrm{L}, \mathrm{S}, \mathrm{R}\}^k$



图灵机

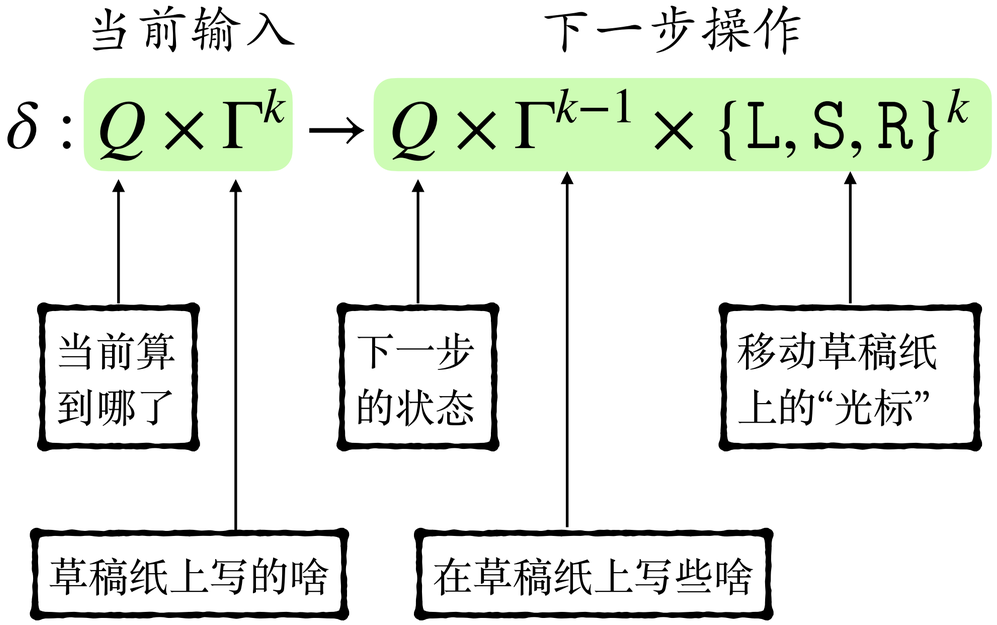
> For thousands of years, the term "computation" was understood to mean application of mechanical rules to manipulate numbers, where the person/machine doing the manipulation is allowed a *scratch pad* on which to write the intermediate results. The Turing machine is a concrete embodiment of this intuitive notion. Section 1.2.1

Computational Complexity, S. Arora, B. Barak

程序＝解问题的"套路"＋草稿纸

一个$k-$带图灵机是一个三元组 (Γ, Q, δ):

- $k-$带： k 张草稿纸
- Γ: 字符集
- Q : 状态集（算到哪一步了）
- $\delta : Q \times \Gamma^k \to Q \times \Gamma^{k-1} \times \{\mathrm{L}, \mathrm{S}, \mathrm{R}\}^k$



丘奇-图灵论题

丘奇-图灵论题

任何“有效的计算法则”都和图灵机等价

丘奇-图灵论题

任何“有效的计算法则”都和图灵机等价

图灵机 = λ-演算 = 递归函数 = C++ = LaTeX = 量子计算机=康威的生命游戏（Game of Life）=...

丘奇-图灵论题

任何“有效的计算法则”都和图灵机等价

图灵机 = λ-演算 = 递归函数 = C++ = LaTeX = 量子计算机=康威的生命游戏（Game of Life）=...

问题 $f: \{0,1\}^* \to \{0,1\}^*$ 可计算

丘奇-图灵论题

任何“有效的计算法则”都和图灵机等价

图灵机 = λ-演算 = 递归函数 = C++ = LaTeX = 量子计算机=康威的生命游戏（Game of Life）=...

问题 $f: \{0,1\}^* \to \{0,1\}^*$ 可计算

$\Leftrightarrow$

丘奇-图灵论题

任何“有效的计算法则”都和图灵机等价

图灵机 = λ-演算 = 递归函数 = C++ = LaTeX = 量子计算机=康威的生命游戏（Game of Life）=...

问题 $f: \{0,1\}^* \to \{0,1\}^*$ 可计算

$\iff$ 存在图灵机 $T: \forall x, T(x) = f(x)$

丘奇-图灵论题

任何“有效的计算法则”都和图灵机等价

图灵机 = λ-演算 = 递归函数 = C++ = LaTeX = 量子计算机=康威的生命游戏（Game of Life）=...

问题 $f: \{0,1\}^* \to \{0,1\}^*$ 可计算

$\iff$ 存在图灵机 $T: \forall x, T(x) = f(x)$

$\iff$

丘奇-图灵论题

任何“有效的计算法则”都和图灵机等价

图灵机 = λ-演算 = 递归函数 = C++ = LaTeX = 量子计算机=康威的生命游戏（Game of Life）=...

问题 $f: \{0,1\}^* \to \{0,1\}^*$ 可计算

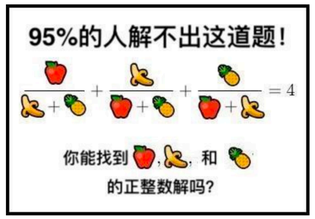
$\iff$ 存在图灵机 $T: \forall x, T(x) = f(x)$

$\iff$ 存在C++程序 $P: \forall x, P(x) = f(x)$

函数的可计算性

函数的可计算性

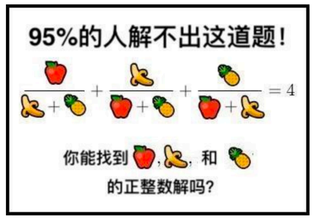
什么是算法



你能写一个程序/算法解这类问题吗?

函数的可计算性

什么是算法



= 154476802108746166441951315019919837485664325669656431700026634898253202035277999

= 368751317941299998271978115652254748254929779968971970996283137471637224634055579

= 437361207792869725786125260237139015281653755816161638186214379933784234677772036

你能写一个程序/算法解这类问题吗?

希尔伯特的“判定性问题”

Hilbert's Entscheidungsproblem:

“是否存在一个算法/程序，能够自动地给出一个定理的证明？”

人工智能能够代替数学家吗？

In 1936, Alonzo Church and Alan Turing published independent papers[2] showing that a general solution to the *Entscheidungsproblem* is impossible, assuming that the intuitive notion of "effectively calculable" is captured by the functions computable by a Turing machine (or equivalently, by those expressible in the lambda calculus). This assumption is now known as the Church–Turing thesis.

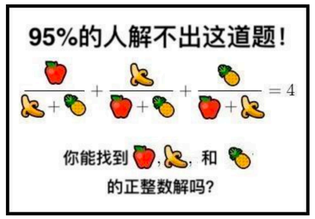
https://en.wikipedia.org/wiki/Entscheidungsproblem

Alonzo Church:
Lambda演算

Alan Turing:
图灵机

函数的可计算性

什么是算法



你能写一个程序/算法解这类问题吗?

希尔伯特的“判定性问题”

Hilbert's Entscheidungsproblem:

“是否存在一个算法/程序，能够自动地给出一个定理的证明？”

人工智能能够代替数学家吗？

In 1936, Alonzo Church and Alan Turing published independent papers[2] showing that a general solution to the *Entscheidungsproblem* is impossible, assuming that the intuitive notion of "effectively calculable" is captured by the functions computable by a Turing machine (or equivalently, by those expressible in the lambda calculus). This assumption is now known as the Church–Turing thesis.

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

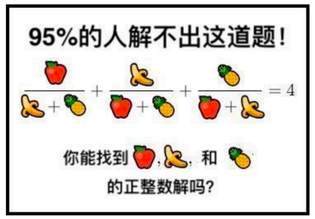
Alonzo Church:
Lambda演算

Alan Turing:
图灵机

问题：每一个函数都是可计算的吗?

函数的可计算性

什么是算法



你能写一个程序/算法解这类问题吗?

希尔伯特的“判定性问题”

Hilbert's Entscheidungsproblem:

“是否存在一个算法/程序，能够自动地给出一个定理的证明？”

人工智能能够代替数学家吗？

In 1936, Alonzo Church and Alan Turing published independent papers[2] showing that a general solution to the *Entscheidungsproblem* is impossible, assuming that the intuitive notion of "effectively calculable" is captured by the functions computable by a Turing machine (or equivalently, by those expressible in the lambda calculus). This assumption is now known as the Church–Turing thesis.

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

Alonzo Church:
Lambda演算

Alan Turing:
图灵机

问题：每一个函数都是可计算的吗?

几乎所有的函数都是不可计算的：函数的个数是不可数的，不同程序的个数是可数的。

停机问题（Halting Problem）

停机问题（Halting Problem）

每一个自然数都可以和C++程序建立一一映射：

停机问题（Halting Problem）

每一个自然数都可以和C++程序建立一一映射：

$$x \in \mathbb{N} \mapsto P_x \in \{\text{C++程序}\}$$

停机问题（Halting Problem）

每一个自然数都可以和C++程序建立一一映射：

$$x \in \mathbb{N} \mapsto P_x \in \{\text{C++程序}\}$$

考虑如下函数：

停机问题（Halting Problem）

每一个自然数都可以和C++程序建立一一映射：

$$x \in \mathbb{N} \mapsto P_x \in \{\text{C++程序}\}$$

考虑如下函数：

$$\texttt{Halt}(x, y) = \begin{cases} 1, & \text{如果 } P_x(y) \text{有死循环;} \\ 0, & \text{如果 } P_x(y) \text{没有死循环.} \end{cases}$$

停机问题（Halting Problem）

每一个自然数都可以和C++程序建立一一映射：

$$x \in \mathbb{N} \mapsto P_x \in \{\text{C++程序}\}$$

考虑如下函数：

$$\texttt{Halt}(x, y) = \begin{cases} 1, & \text{如果 } P_x(y) \text{有死循环;} \\ 0, & \text{如果 } P_x(y) \text{没有死循环.} \end{cases}$$

定理：Halt 函数是不可计算的。

反证。假设存在一个程序 A 可以计算 $\texttt{Halt}$，即 $A(x, y) = \texttt{Halt}(x, y)$

反证。假设存在一个程序 A 可以计算 $\texttt{Halt}$，即 $A(x, y) = \texttt{Halt}(x, y)$

我们可以把 A 当做子程序，写出如下程序 B

反证。假设存在一个程序 A 可以计算 $\texttt{Halt}$，即 $A(x,y) = \texttt{Halt}(x,y)$

我们可以把 A 当做子程序，写出如下程序 B

```
Program B:
int main(int x) {
    If A(x,x)=1 then
        return 0;
    else
        for(;;); // loop forever
}
```

反证。假设存在一个程序 A 可以计算 $\texttt{Halt}$，即 $A(x, y) = \texttt{Halt}(x, y)$

我们可以把 A 当做子程序，写出如下程序 B

```
Program B:
int main(int x) {
    If A(x,x)=1 then
        return 0;
    else
        for(;;); // loop forever
}
```

用 $x(B)$ 表示 B 对应的自然数。

反证。假设存在一个程序 A 可以计算 Halt，即 $A(x, y) = \texttt{Halt}(x, y)$

我们可以把 A 当做子程序，写出如下程序 B

```
Program B:
int main(int x) {
    If A(x,x)=1 then
        return 0;
    else
        for(;;); // loop forever
}
```

用 $x(B)$ 表示 B 对应的自然数。

$B(x(B))=?$

希尔伯特的“判定性问题”

Hilbert's Entscheidungsproblem:

“是否存在一个算法/程序，能够自动地给出一个定理的证明？”

人工智能能够代替数学家吗？

In 1936, Alonzo Church and Alan Turing published independent papers[2] showing that a general solution to the *Entscheidungsproblem* is impossible, assuming that the intuitive notion of "effectively calculable" is captured by the functions computable by a Turing machine (or equivalently, by those expressible in the lambda calculus). This assumption is now known as the Church–Turing thesis.

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

Alonzo Church:
Lambda演算

Alan Turing:
图灵机

希尔伯特的“判定性问题”

Hilbert's Entscheidungsproblem:

“是否存在一个算法/程序，能够自动地给出一个定理的证明？”

人工智能能够代替数学家吗？

In 1936, Alonzo Church and Alan Turing published independent papers[2] showing that a general solution to the *Entscheidungsproblem* is impossible, assuming that the intuitive notion of "effectively calculable" is captured by the functions computable by a Turing machine (or equivalently, by those expressible in the lambda calculus). This assumption is now known as the Church–Turing thesis.

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

Alonzo Church:
Lambda演算

Alan Turing:
图灵机

给定一个程序，其是否死循环可以写成一个数学命题

希尔伯特的“判定性问题”

Hilbert's Entscheidungsproblem:

“是否存在一个算法/程序，能够自动地给出一个定理的证明？”

人工智能能够代替数学家吗？

In 1936, Alonzo Church and Alan Turing published independent papers[2] showing that a general solution to the *Entscheidungsproblem* is impossible, assuming that the intuitive notion of "effectively calculable" is captured by the functions computable by a Turing machine (or equivalently, by those expressible in the lambda calculus). This assumption is now known as the Church–Turing thesis.

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

Alonzo Church:
Lambda演算

Alan Turing:
图灵机

给定一个程序，其是否死循环可以写成一个数学命题

不存在通用的算法判定命题是否成立！

希尔伯特的“判定性问题”

Hilbert's Entscheidungsproblem:

“是否存在一个算法/程序，能够自动地给出一个定理的证明？”

人工智能能够代替数学家吗？

In 1936, Alonzo Church and Alan Turing published independent papers[2] showing that a general solution to the *Entscheidungsproblem* is impossible, assuming that the intuitive notion of "effectively calculable" is captured by the functions computable by a Turing machine (or equivalently, by those expressible in the lambda calculus). This assumption is now known as the Church–Turing thesis.

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

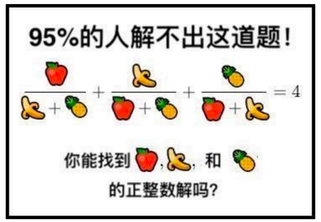
Alonzo Church:
Lambda演算

Alan Turing:
图灵机

给定一个程序，其是否死循环可以写成一个数学命题

不存在通用的算法判定命题是否成立！

什么是算法



你能写一个程序/算法解这类问题吗？

希尔伯特的“判定性问题”

Hilbert's Entscheidungsproblem:

“是否存在一个算法/程序，能够自动地给出一个定理的证明？”

人工智能能够代替数学家吗？

In 1936, Alonzo Church and Alan Turing published independent papers[2] showing that a general solution to the *Entscheidungsproblem* is impossible, assuming that the intuitive notion of "effectively calculable" is captured by the functions computable by a Turing machine (or equivalently, by those expressible in the lambda calculus). This assumption is now known as the Church–Turing thesis.

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

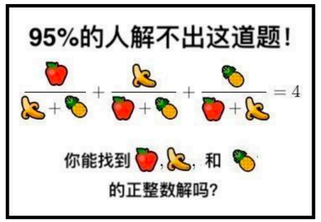
Alonzo Church:
Lambda演算

Alan Turing:
图灵机

给定一个程序，其是否死循环可以写成一个数学命题

不存在通用的算法判定命题是否成立！

什么是算法



你能写一个程序/算法解这类问题吗？

= 154476802108746166441951315019919837485664325669565431700026634898253202035277999

= 36875131794129999827197811565225474825492979968971970996283137471637224634055579

= 4373612677926869725786125260237139015281655755816161361862143799337842346772036

希尔伯特第十问题：是否存在算法判断丢番图方程是否有解？

希尔伯特的“判定性问题”

Hilbert's Entscheidungsproblem:

“是否存在一个算法/程序，能够自动地给出一个定理的证明？”

人工智能能够代替数学家吗？

In 1936, Alonzo Church and Alan Turing published independent papers[2] showing that a general solution to the *Entscheidungsproblem* is impossible, assuming that the intuitive notion of "effectively calculable" is captured by the functions computable by a Turing machine (or equivalently, by those expressible in the lambda calculus). This assumption is now known as the Church–Turing thesis.

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

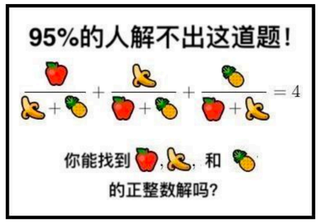
Alonzo Church:
Lambda演算

Alan Turing:
图灵机

给定一个程序，其是否死循环可以写成一个数学命题

不存在通用的算法判定命题是否成立！

什么是算法



你能写一个程序/算法解这类问题吗?

= 154476802108746166441951315019919837485664325669565431700026634898253202035277999

= 3687513179412999982719781156522547482549297996897197099628313747163724634055579

= 4373612677926897257861252602371390152816537558161613618621437993378423467772036

希尔伯特第十问题：是否存在算法判断丢番图方程是否有解?

Matiyasevich 定理（1970）：不存在求解丢番图方程的通用算法。

计算问题分类

计算问题分类

计算问题分类

- 可计算理论/递归论：研究不可计算函数

计算问题分类

- 可计算理论/递归论：研究不可计算函数

这个问题能不能计算？

计算问题分类

- 可计算理论/递归论：研究不可计算函数

  这个问题能不能计算？

- 计算复杂性/算法：研究可计算函数

计算问题分类

- 可计算理论/递归论：研究不可计算函数

这个问题能不能计算？

- 计算复杂性/算法：研究可计算函数

这个问题能多快计算？