运筹学通论I

胡晓东

应用数学研究所

中国科学院数学与系统科学研究院

http://www.amt.ac.cn/member/huxiaodong/index.html

4. 算法理论 - 问题

这里我们仅讨论以下两类**抽象问题**（而不是如数理逻辑中那样，考虑字符串或者语言）。

判定问题 (亦称 **识别问题**) 仅有两种可能的答案，“**是**”或者“**否**”。此时，我们可以将一个判定问题视为一个函数，它将问题的输入集合 I 映射到问题解的集合 $\{0, 1\}$。 例如，我们考虑**路问题**： 给定一个图 $G=(V, E)$ 和顶点集 V 中的两个顶点 u, v， 判断 G 中是否存在一条路 u 和 v 之间的路。如果我们用 $i=<G, u, v>$ 表示该问题的一个输入，则函数 $PATH(i)=1$, 当相应的答案为“**是**”，即u 和 v 之间存在一条路；否则函数 $PATH(i)=0$。 注意，这里对于 **是-实例**，我们仅需要给出答案“**是**”，而不必要给出 u 和 v 之间的一条路。

4. 算法理论 - 问题（续一）

优化问题 或者求最小值或者求最大值。一个优化问题通常可以用以下四个部分来描述。

实　　例：若干实例 **I** 组成的集合 D，其中每一个实例 **I** 含有一个问题所有输入的数据信息。

可行解集：每一个实例 **I** 有一个解集合 $S(I)$，其中的每一个解都满足问题的条件，称为**可行解**。

目标函数：映射 $c(\sigma): S(I) \rightarrow \Re$

最 优 化：求**最优解** $\sigma_{opt}(I) \in S(I)$，使得对任意一个可行解 $\sigma \in S(I)$，都有 $c(\sigma_{opt}(I)) \geq c(\sigma)$ (求最大值问题)

或者 $c(\sigma_{opt}(I)) \leq c(\sigma)$ (求最小值问题)

一个优化问题也可以视为一个判定问题。特别地，在问题的实例之外，再给定一个最优值的上（或下）解即可。

4. 算法理论 - 问题（续二）

尽管在本课程的最后一部分我们主要讨论优化问题，但是在本部分（计算复杂性理论），我们还是主要考虑判定问题。这主要基于以下两个原因。

- 表述**形式语言理论**及其相应的基于**判定问题**的计算复杂性理论的结果要更加容易一些。
- 基于判定问题的结果并不会减弱相应理论的影响和应用。

一方面，如果我们可以非常快速地求解一个优化问题，那么我们也可以非常快速地求解相应的判定问题。亦即，如果优化问题容易求解， 那么相应的判定问题也容易求解。

另一方面，如果我们有证据表明，求解一个判定问题是困难的，那么我们也可以给出证据表明，求解相应的优化问题同样是困难的。

4. 算法理论 - 算法

一般地且不是严格地讲，一个**算法**可以理解为一个合理的计算步骤：它以一些变量值为输入，然后生成若干变量值做为输出。换言之，一个算法是一系列计算步骤，它根据输入给出输出。

我们称一个算法是**正确的**，如果对每一个输入的实例，它都可以终止并给出正确的输出（我们需要的解）。我们称一个正确的算法可以求解一个特定的计算问题。一个**不正确的**算法，对某一些输入的实例，可能不能终止，或者可以终止但是输出的解不正确。通常人们可能会觉得奇怪，不正确的算法有时竟然也是有用的， 只要算法出错的可能性控制在一定范围内，亦即，算法对大多数输入的实例，都能输出正确的解。然而，在本课程中我们主要讨论正确的算法。

4. 算法理论 - 图灵机

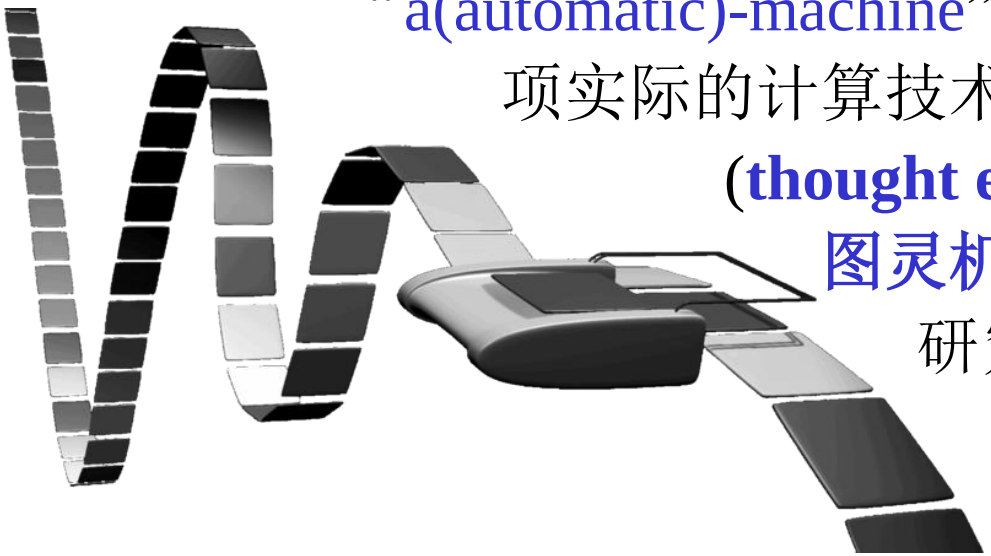
当分析一个算法的性能时，我们需要估算一些算法所需要的资源。有时候，**资源**可能是指存储空间，通讯带宽，或者逻辑门，这些从技术的角度讲非常基本因素。然而，在这里我们主意是指需要考虑的**计算时间**，也就是，算法在生成最终解或者找到问题的答案以前所花费的时间。注意，由于所用计算机的速度、存储空间和其他技术因素的不同，计算时间可能会有巨大的差异。

因而，在我们开始从计算的角度分析一个算法前，我们必须有一个技术上可实现的**计算模型**，它包括计算所需的所有资源和使用它们的相应成本。

4. 算法理论 - 图灵机

TURING CENTENARY CONFERENCE

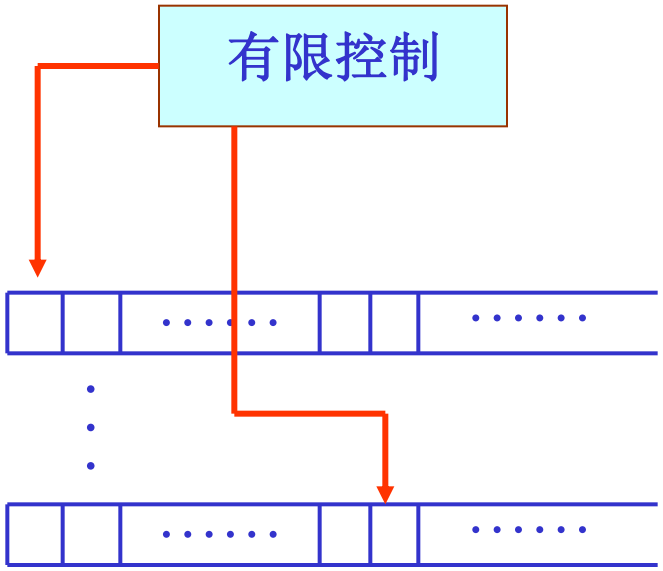
CiE 2012 - How the World Computes



1936年，**图灵**描述了一个计算模型，后称**图灵机**“a(automatic)-machine”。他并不是想将其设计成一项实际的计算技术，而是要用表示思维实验(**thought experiment**)中的计算机器。**图灵机**主要是帮助计算机科学家研究机器(**mechanical computation**)计算的能力。

4. 算法理论 - 图灵机（续一）

一个多带确定型图灵机 (**DTM**) 有 ***k*** 条**带**，每一条都有一个起始**单元**，而它的右侧有无限多个单元。它还有一个有限**字符集合**，每一个单元可储存其中的一个字符。每一条带都有一个**读写头**，它可对带上的单元进行逐一扫描，并读出单元所存储的字符或者将一个字符存储到该单元中。 图灵机是由一个**有限控制**所操作，每一个控制都是处于有限状态中的某个状态。



TM(*S*, *T*, *I*, *δ*, B, s_0, s_f)

S ： 状态集

T ： 带用字符集

I ： 输入字符集

B ： ***T*** \ ***I*** 中的“空”

s_0 ： 初始状态

s_f ： 最终/接受状态

δ： 下次一移动函数

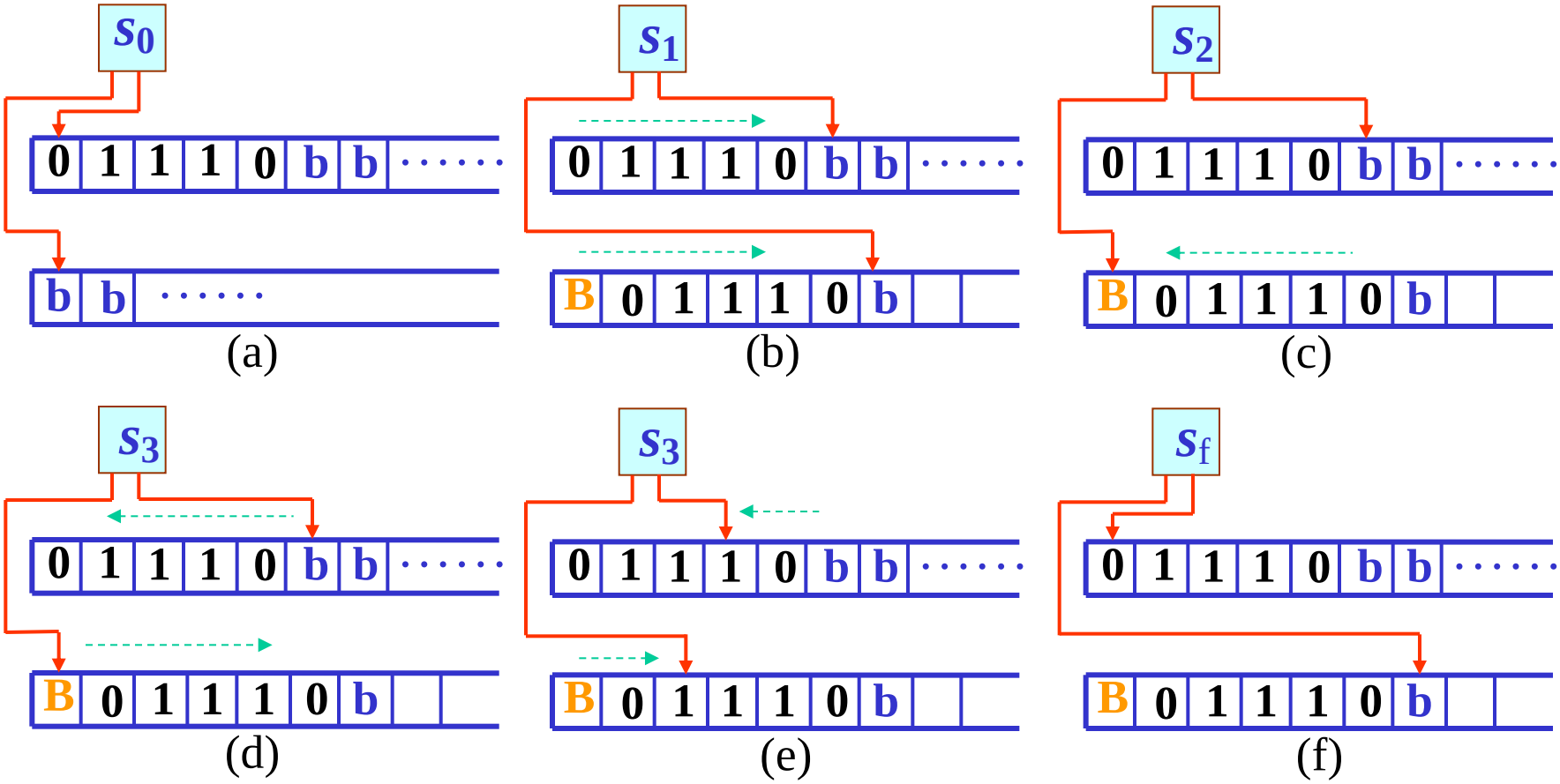
4. 算法理论 - 图灵机（续二）

我们可以用如下的方法设计一个辨识某种特殊字符串的图灵机：带用字符集包括输入的字符和一个特殊字符 **B**，也可以还包含一些其他的字符。初始时，第一条带子上是输入的字符串，其中从最左侧开始，每一个单元存储了一个字符；且字符串的最后一个字符右侧的单元的所含字符皆为**B**。 其他带上的单元所含字符皆为**B**。此时，图灵机处于初始状态 s_0，且所有的读写头都位于最左侧的单位。然后，图灵机根据下次一移动函数 δ 在带子上进行一系列的移动和操作。

图灵机**接受**输入的字符串当且仅当它在系列操作以后达到最终状态 s_f。否则图灵机在某一步不进行任何操作；它会处于某个非接受状态。

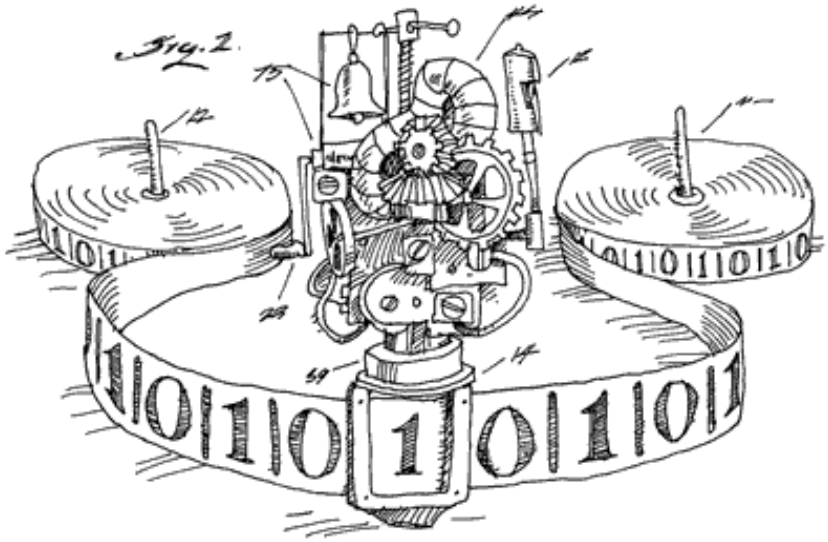
4. 算法理论 - 图灵机（续三）

我们下面考虑一个简单的**2-**带图灵机。它可以**确定**字符集 **{ 0, 1 }** 上的一个字符串是**回文**（字符串从前往后与从后往前读是一样的；例如 **01110**）。



4. 算法理论 - 图灵机（续四）

上述简单例子表明，我们可以将一个图灵机视为一个仪器，用它可以确认**输入的字符串**是否有某种特定的性质。



实际上，我们还可以将一个图灵机视为计算一个函数 ***f*** 的仪器。我们可以将函数的每一个变量值进行编码，再用一个特殊的字符将对应于不同变量值的编码隔开，组成一个字符串**$**。如果图灵机终止操作，并在预先指定的输出带上存储了字符串 ***y***，那么我们就称函数值为 ***y***，即 ***f*(*x*) = *y***。可以看出来，计算函数值的过程与确认字符串具有某种性质的过程有些不一样。

4. 算法理论 - 图灵机（续五）

对于问题的不同输入实例，即使它们输入占用同样多的存储空间（**规模**），一个图灵机所需要进行的操作数目往往还是不相同。 为了处理这种情形：由于不同输入实例所造成的图灵机操作次数的不同，我们通盘考虑输入规模为 ***n*** 的所有实例。

一个图灵机 **M** 的**时间复杂度 *T*(*n*)** 等于它计算输入规模为 ***n*** 的所有实例中，所需要进行的操作的最多次数。如果对于输入规模为 ***n*** 的某些实例，图灵机不能终止操作，那么称时间复杂度 ***T*(*n*)** 没有定义。

一个图灵机 **M** 的**空间复杂度 *S*(*n*)** 等于它计算输入规模为 ***n*** 的所有实例中，从每个带的最左端到最右端移动的最长距离值。 如果图灵机在某一个带上，向右无穷远地移动，那么称空间复杂度 ***S*(*n*)** 没有定义。

4. 算法理论 - 图灵机（续六）

关于函数的几个记法在分析算法的时间复杂度是会经常用到：

$O(g(n))$ ={ $f(n)$: 存在常正数 c 和 n_0 使得，对所有 $n \geq n_0$，都有 $0 \leq f(n) \leq c\, g(n)$ }。

$\Omega(g(n))$ ={ $f(n)$: 存在常正数 c 和 n_0 使得，对所有 $n \geq n_0$，都有 $0 \leq c\, g(n) \leq f(n)$ }。

$\Theta(g(n))$ ={ $f(n)$: 存在常正数 c_1，c_2 和 n_0 使得，对所有 $n \geq n_0$，都有 $0 \leq c_1\, g(n) \leq f(n) \leq c_2\, g(n)$ }。

$o(g(n))$ ={ $f(n)$: 对任意一个常数 $c \geq 0$，存在一个正整数 $n_0 \geq 0$，使得对所有 $n \geq n_0$，都有 $0 \leq f(n) \leq c\, g(n)$ }。

思考题： 试比较 *O*-记法和 *o*-记法之间的关系。

4. 算法理论 - 图灵机（续七）

定理 1 若判定问题 P_0 可被一个时间复杂度为 $T(n)$ 的 k-带图灵机 M 接受，则它可被一个时间复杂度为 $O(T^2(n))$ 的单带图灵机 M_1 接受。

证明思路（练习**）：设计一个时间复杂度为 $O(T^2(n))$ 的单带图灵机 M_1，用它来模拟时间复杂度为 $T(n)$ 的 k-带图灵机 M。主要思想是：将 M_1 的一个带想象成由 $2k$ 个“轨道”构成，换言之，M_1 的一个字符是一个 $2k$ 元组。

定理 2 若判定问题 P_0 可被一个空间复杂度为 $S(n)$ 的 k-带图灵机 M 接受，则它可被一个具有同样空间复杂度的单带图灵机 M_1 接受。

4. 算法理论 - 图灵机（续八）

图灵认识到：不仅问题的输入数据，而且程序本身也可以用一系列的符号来代码化。**图灵**构造了一个程序：只要把程序连同它所要对之操作的输入数据一起，写到通用图灵机的带上。通用图灵机就能模拟程序对输入数据的操作。

尽管现在的计算机结构上要复杂得多，计算能力要强大得多，但是，**图灵**认识到：能在任何的计算机上执行的程序，也能用他的机器的一种特定样式，**通用图灵机，**来完成。这意味着：只要给予足够的时间和内存，没有任何一种计算机器能做的计算而用通用图灵机不能够计算的（即图灵机作为一个计算模型是足够的了）。

图灵-丘奇假设 每一个可计算的量都可用某个**图灵机**来计算。

4. 算法理论 - 图灵机（续九）

停机问题：任给图灵机程序***P*** 和一组输入数据 ***I***，是否存在单个程序，接受 ***P*** 和 ***I*** 作为其输入，并在有限步后停机，并在带上的最后做出说明，***P*** 在处理 ***I*** 时是否将在有限步后停止？

停机定理 (1936): 给定一任意的图灵机程序 ***P*** 和一组任意的输入数据 ***I***，不存在单个的图灵机程序，它在有限多步后停机，并告诉我们 ***P*** 是否会结束输入数据 ***I*** 的处理。

证明: 反证假设，存在这样一个图灵机**TM(*P*, *I*)**，它可以判定程序 ***P*** 在输入 ***I*** 的情况下是否可停机：若 ***P*** 在输入 ***I*** 时可停机，**TM(*P*, *I*)** 输出“停机”，否则，**TM(*P*, *I*)** 输出“死循环”。

显然，**TM(*P*, *I*)** 本身可被视程序 P_{TM}，也可作为输入 I_{TM}，故**TM(*P*, *I*)** 可以判定当将 I_{TM} 作为 P_{TM} 的输入时，P_{TM} 是否会停机。

4. 算法理论 - 图灵机（续十）

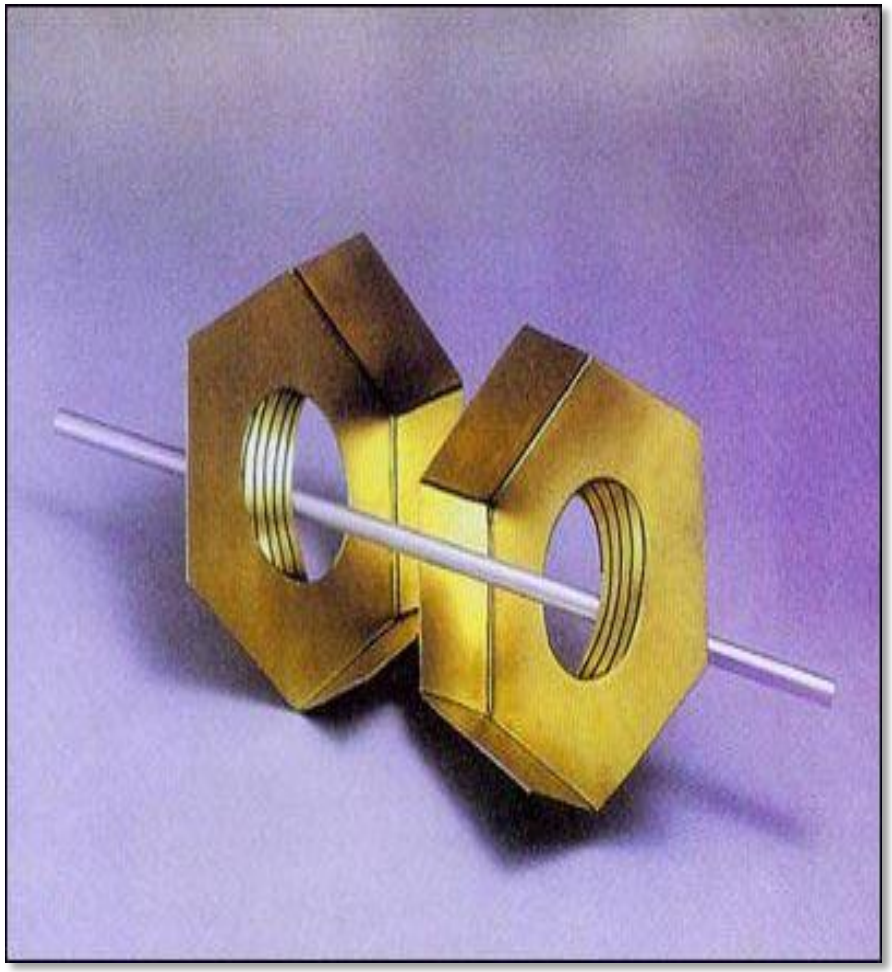
现考虑一个与 **TM**(P, I) 相反的图灵机$\overline{\mathbf{TM}}$(P, I)：首先，它调用**TM**(P_{TM}, I_{TM})。若 **TM**(P_{TM}, I_{TM}) 输出“死循环”，则输出“停机”，否则输出“死循环”。

再考虑运行 **TM**($P_{\overline{TM}}$, $I_{\overline{TM}}$) 时的情形。若 **TM**($P_{\overline{TM}}$, $I_{\overline{TM}}$) 输出“停机”，则根据**TM**的定义，$P_{\overline{TM}}$在输入$T_{\overline{TM}}$时，输出“停机”；但是，根据$\overline{\mathbf{TM}}$的定义，**TM**应该输出“死循环”。类似地，若 **TM**($P_{\overline{TM}}$, $I_{\overline{TM}}$) 输出“死循环”，则根据**TM**的定义，$P_{\overline{TM}}$在输入$T_{\overline{TM}}$时，输出“死循环”；但是，根据$\overline{\mathbf{TM}}$的定义，**TM** 应该输出“停机”。总产生矛盾。

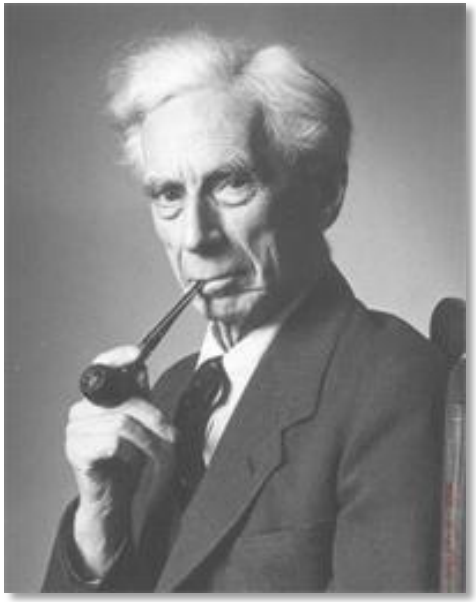
思考题： 哥德尔定理 和 停机定理 之间有什么关系？

4. 算法理论 - 图灵机（续十一）

停机测试悖论：计算机里有个测试程序，这个测试程序的原则是，对于计算机里所有程序，当且仅当这个程序不递归调用自己(输出停机)，测试程序就调用它(对应不停机)。如果这个程序递归调用自己（对应不停机），测试程序就不调用它（对应停机）。无法回答的问题是，测试程序递归调用自己么？



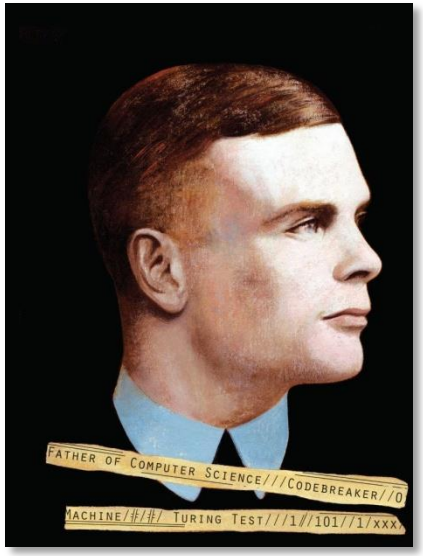
4. 算法理论 - 图灵机（续十二）



理发师悖论（**罗素**，1872－1970）城市里唯一的理发师只为那些不为自己理发的人理发。理发师应该为自己理发吗?

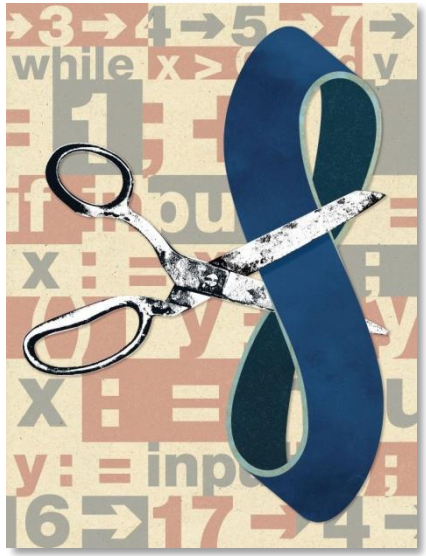
罗素悖论的出现引发了第三次数学危机。**策梅洛**（**Zermelo**，1871-1953）应用自己的公理系统，使得集合在公理的限制下不会太大，从而避免了**罗素悖论**。经过改进，这一系统形成了现在被称为**ZF系统**的公理集合论体系。由**策梅洛**（**Zermelo**）和**弗伦克尔**（**Fraenkel**）提出的这个体系至今还没有发现悖论。

4. 算法理论 - 图灵机（续十三）



Embodied and disembodied computing
at the Turing Centenary
By S. Barry Cooper
Turing's Titanic Machine?
Communications of the ACM, 2012 (55) (3), 74 - 83

In contrast to popular belief,
proving termination is not always impossible
By B. Cook, A. Podelski, A. Rybalchenko
Proving Program Termination
Communications of the ACM, 2011 (54) (5), 88 - 98



4. 算法理论 - 图灵机（续十四）

现在我们就可以将一个**算法**定义为一个确定型图灵机，其复杂度也是通过相应的图灵机的复杂度定义的。注意，这样的定义与通常人们理解的算法的概念有些不同：一个算法是一系列明确的计算步骤，它们与我们要用什么样的机器和硬件（带、读写头等）运行这些算法是无关的。

在实际运行算法时，求解不同问题的不同算法通常是可以在同一个计算机上运行的。同时，一个算法也可以在不同的计算机上运行，只要这些计算机的操作系统都是一样的。

不过，在不同计算机上运行同一个算法，所需要的时间可能会因为不同型号的中央处理器和其他硬件因素而有所不同。实际上，用图灵机来描述算法的一个主要目的就是，分析算法的性能时，我们可以不考虑这些技术上的因素。

4. 算法理论 - 图灵机（续十五）

因而，为了忽略算法的严格定义与非严格定义之间的这种差别，可将一个算法视为一个具有有限控制的图灵机的 **δ下次-移动函数**（而非图灵机本身）。一般来讲，我们并不将一个算法描述成一个δ下次-移动函数，这么做也没必要。在分析图灵机模型下一个算法的性能时，通常是用算法所执行的基本运算的次数，比如算术运算、比较运算等等，来计算它的**时间复杂度**。此时，我们假定在**单位时间**内可以完成这些基本运算。

实际上，人们发现所有可接受的计算模型都有一个显著的性质：如果一个问题在某一个计算模型下是多项式时间可解的，那么它在其他计算模型下同样也是多项式时间可解的。

为了将算法描述的清楚和简洁，我们采用**伪码**。它与常用的 **Fortran** 等不同，它允许使用任意形式的数学表述，只要它的含义是清楚的，而且其指令可以很容易地转换为图灵机所用的（真正的）码（字符）。

4. 算法理论 - 非确定图灵机

计算复杂性理论的一个核心概念是**非确定型图灵机**，**NonDeterministic Turing Machine (NDTM)**，的模型。

确定型图灵机与**非确定型图灵机**之间最重要的区别在于：在要执行每一次（运算）操作时，**后者**有有限多个（移动）方式可以选择，究竟选择哪一个是不确定的；而**前者**仅有一个可以选择，亦即（移动）方式是完全确定的。

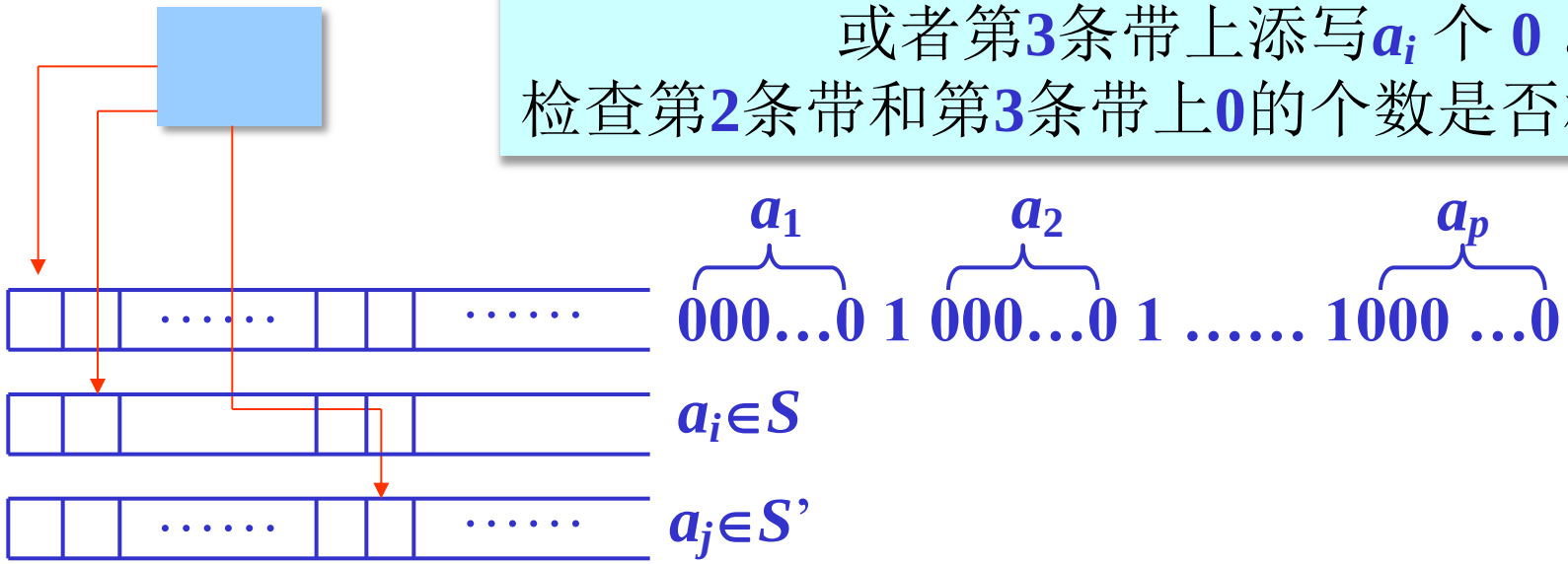
给定一个实例输入 **I**，我们可以将非确定型图灵机 **M** 的运算方式看作如下过程：**并行地**执行所有可能的（运算）操作，直到达到接受状态或者无法再进行操作。

4. 算法理论 - 非确定图灵机（续一）

考虑**划分问题**，并采用简单的编码方式：给定一组自然数，$a_1, \ldots, a_p$，判断集合$\{1, 2, \ldots, p\}$是否存在一个划分，S和S'，使得

$$\sum_{i \in S} a_i = \sum_{j \in S'} a_j,$$ 其中 $S \cup S' = \{1, 2, \ldots, p\}$ 且 $S \cap S' = \varnothing$。

读取输入带上的信息
以随机地方式在第2条带
　　或者第3条带上添写a_i个 0 。
检查第2条带和第3条带上0的个数是否相等



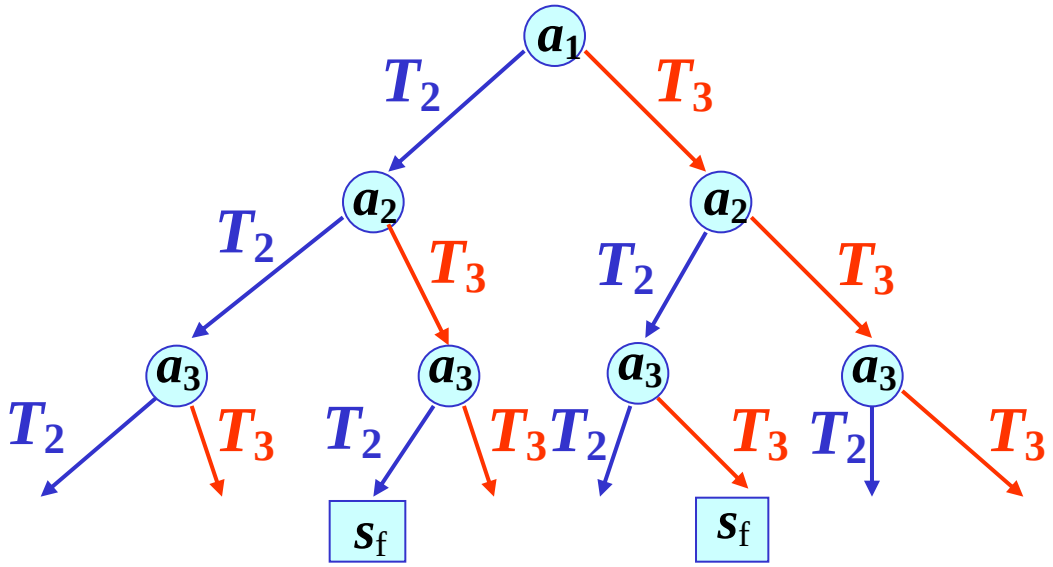
4. 算法理论 - 非确定图灵机（续二）

输入为

$a_1 = 3$: 000

$a_2 = 8$: 00000000

$a_3 = 5$: 00000



我们称非确定型图灵机 **M** 的时间复杂度为$T(n)$ 如果对于任意输入长度为 n 且可达接受状态的实例，都存在不超过 $T(n)$ 次系列操作使得最后达到接受状态 s_f。称 **M** 的空间复杂度为 $S(n)$ 如果对于任意输入长度为 n 且可达接受状态的实例， 都存在在任意一个带上读取不超过 $S(n)$ 个单元的一系列操作，使得最后达到接受状态 s_f。

4. 算法理论 - 非确定图灵机（续三）

至于**确定型图灵机**与**非确定型图灵机**之间的关系，我们不难看出，它们可以模拟彼此的（运算）操作。 然而，当后者模拟前者时，不需要增加运行的时间和空间。与之不同的是，当前者模拟后者时，时间复杂度应会增加很多。 以下定理说明，第二种模拟增加的时间复杂度有一个指数阶的上界。

定理 3 设一个非确定型图灵机 **M** 在时间$T(n)$ 内可以接受判定问题 P_0，且 $T(n)$ 是一个时间可构造的复杂性函数。则存在一个确定型图灵机 **M'**，它可在时间 $O(c^{T(n)})$ 内接受 P_0，其中c 是一个常数。

我们称图灵机 **M** 接受一个判定问题 P_0, 如果 **I** 是问题 P_0 的一个**是-实例**当且仅当 **M** 接受相应 **I** 的字符串。