Problemas fáciles, difíciles e imposibles

¿La computadora lo resuelve todo?

Santiago Figueira

Departamento de Computación
FCEyN, UBA

Semana de la Computación 2015

¿Qué son los métodos efectivos?

Intuitivamente son métodos que pueden ser resueltos por medio de un procedimiento mecánico.

¿Qué son los métodos efectivos?

Intuitivamente son métodos que pueden ser resueltos por medio de un procedimiento mecánico.

Los métodos resuelven problemas. No cualquier tipo de problemas: funciones matemáticas que reciben números naturales y devuelven números naturales.

$f : \mathbb{N} \to \mathbb{N}$ o $g : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ o $h : \mathbb{N} \times \mathbb{N} \times \mathbb{N} \to \mathbb{N}$

Intento 1 para *método efectivo*: funciones recursivas

Podemos usar:

- 0: el cero que todos conocemos
- $+1$: sumarle 1 a un número
- composición de funciones. Por ejemplo $0 + 1 = 1$. También puedo al 1 sumarle 1 y obtener el 2: $(0 + 1) + 1 = 2$
- recursión: una función que se llama a sí misma.

Intento 1 para *método efectivo*: funciones recursivas

Podemos usar:

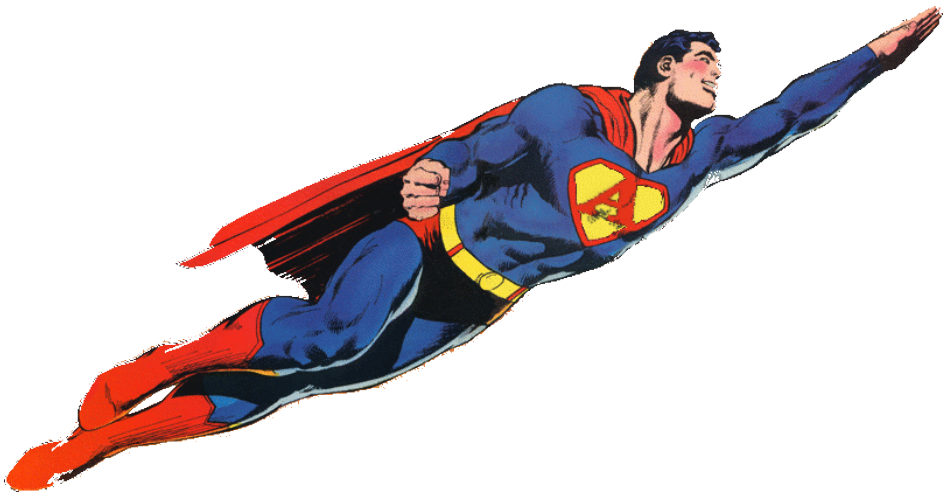
- 0: el cero que todos conocemos
- $+1$: sumarle 1 a un número
- composición de funciones. Por ejemplo $0 + 1 = 1$. También puedo al 1 sumarle 1 y obtener el 2: $(0 + 1) + 1 = 2$
- recursión: una función que se llama a sí misma. Por ejemplo, a la función *suma* se puede definir como:

$$suma(m, n) = \begin{cases} n & \text{si } m = 0 \\ suma(m - 1, n) + 1 & \text{si } m > 0 \end{cases}$$

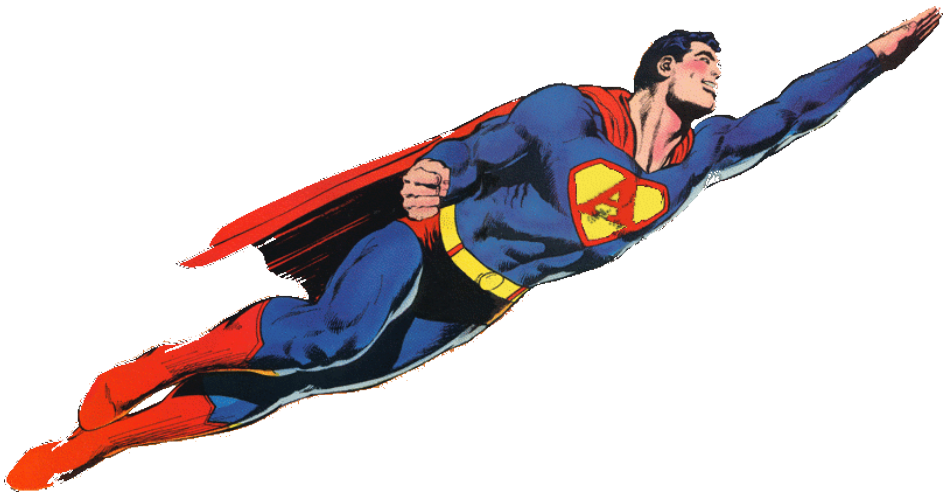
Entonces,

$$suma(2, 2) = suma(1, 2)+1 = suma(0, 2)+1+1 = 2+1+1 = 4$$

La función de Ackermann

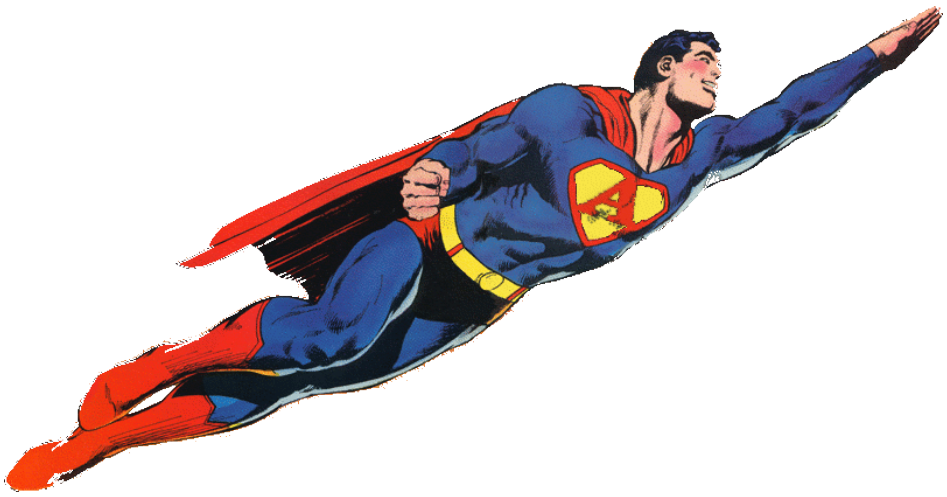


La función de ACKERMANN



$$A(m,n) = \begin{cases} n+1 & \text{si } m = 0 \\ A(m-1,1) & \text{si } m > 0 \text{ y } n = 0 \\ A(m-1,A(m,n-1)) & \text{si } m > 0 \text{ y } n > 0 \end{cases}$$

La función de ACKERMANN



$$A(m,n) = \begin{cases} n+1 & \text{si } m = 0 \\ A(m-1,1) & \text{si } m > 0 \text{ y } n = 0 \\ A(m-1, A(m,n-1)) & \text{si } m > 0 \text{ y } n > 0 \end{cases}$$

- para valores chicos de m, como 1, 2, o 3, A crece relativamente lento con respecto a n

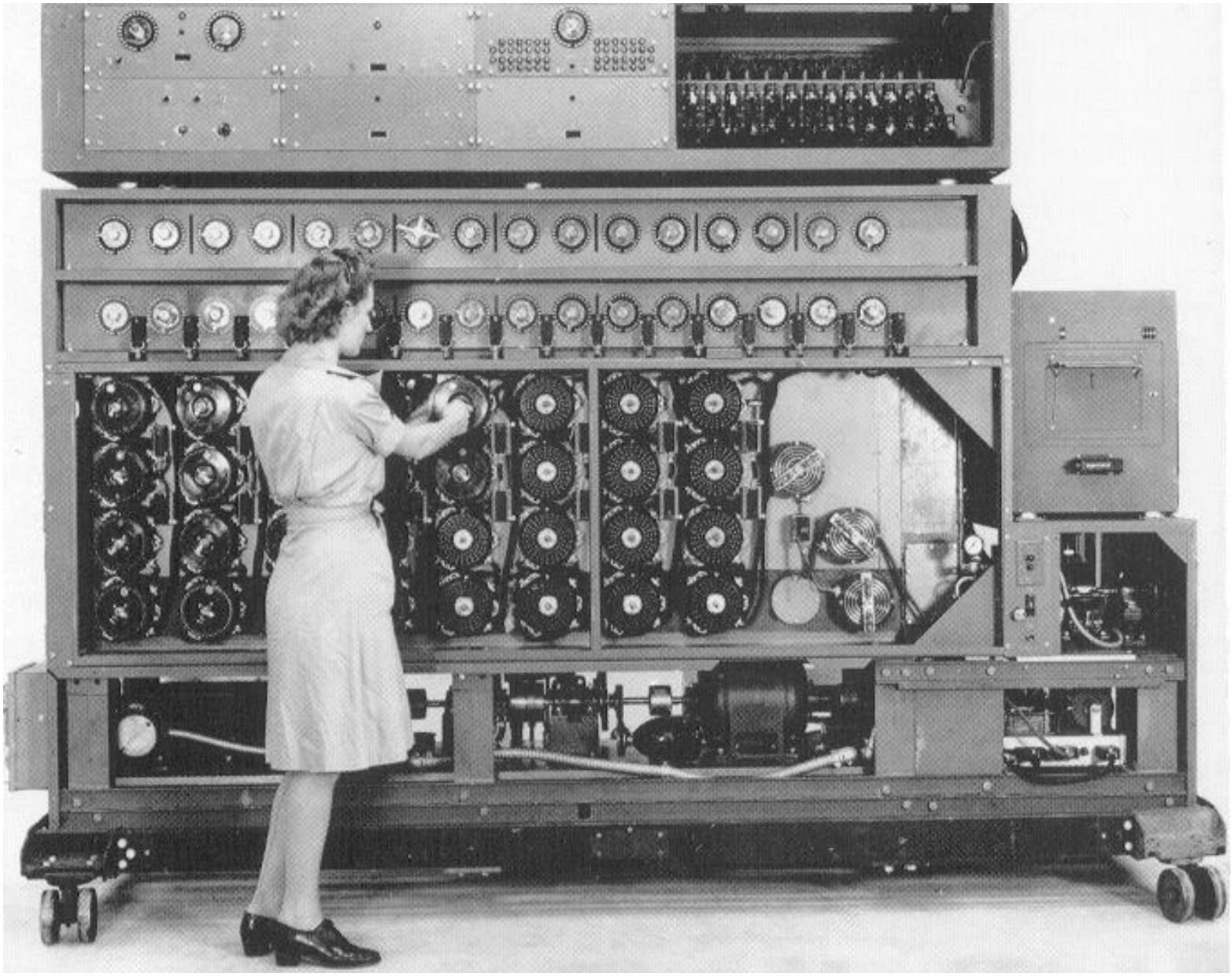
La función de Ackermann

¿Cuánto valdrá A(4,2)? Más o menos

1000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000

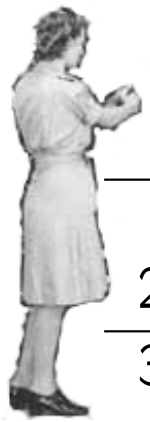
y 18 pantallas más de ceros.

Intento 2 para *método efectivo*: máquinas de Turing



Las máquinas de Turing

Supongamos que la chica del slide anterior quiere calcular:



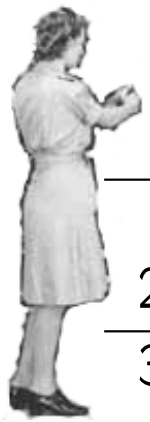
$$\begin{array}{cccc} & 2 & 3 & 1 \\ \times & & 1 & 3 \\ \hline & 6 & 9 & 3 \\ 2 & 3 & 1 & 0 \\ \hline 3 & 0 & 0 & 3 \end{array}$$

No es esencial:

- ¿mientras hace la cuenta está tomando un café?
- ¿escribe con lápiz o lapicera?
- ¿importa el tamaño del papel?

Las máquinas de Turing

Supongamos que la chica del slide anterior quiere calcular:



$$\begin{array}{cccc} & 2 & 3 & 1 \\ \times & & 1 & 3 \\ \hline & 6 & 9 & 3 \\ 2 & 3 & 1 & 0 \\ \hline 3 & 0 & 0 & 3 \end{array}$$

No es esencial:

- ¿mientras hace la cuenta está tomando un café?
- ¿escribe con lápiz o lapicera?
- ¿importa el tamaño del papel?

Lo importante es que:

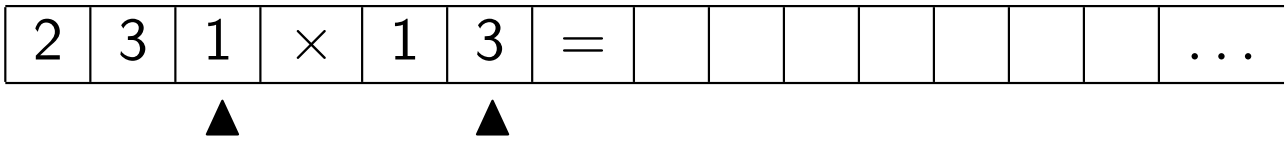
- hace marcas en un papel
- para saber qué escribir en cada paso, le presta atención a lo que escribió antes

Las máquinas de Turing

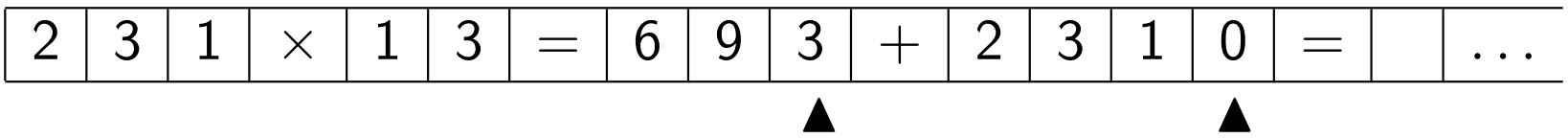
Sin perder nada esencial, podemos suponer:

2	3	1	×	1	3	=	6	9	3	+	2	3	1	0	=	3	0	0	3

Empezamos con la multiplicación



Después de un tiempo, vamos a estar sumando

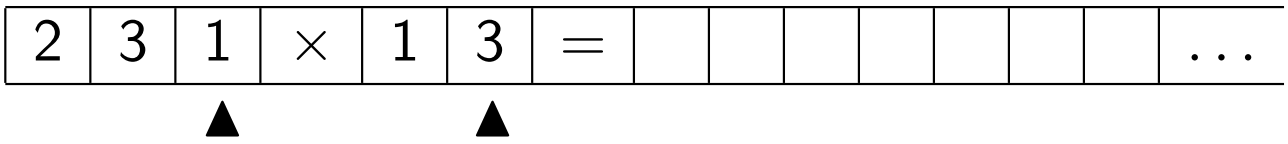


Las máquinas de Turing

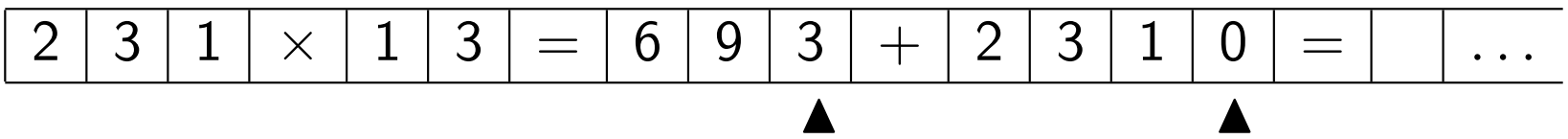
Sin perder nada esencial, podemos suponer:

2	3	1	×	1	3	=	6	9	3	+	2	3	1	0	=	3	0	0	3

Empezamos con la multiplicación



Después de un tiempo, vamos a estar sumando



- en cada paso de la computación sólo un pequeño número de símbolos reciben atención (en el ejemplo, 2). Hay una cantidad finita de símbolos.

Las máquinas de Turing

Sin perder nada esencial, podemos suponer:

2	3	1	×	1	3	=	6	9	3	+	2	3	1	0	=	3	0	0	3

Empezamos con la multiplicación

2	3	1	×	1	3	=								...
		▲			▲									

Después de un tiempo, vamos a estar sumando

2	3	1	×	1	3	=	6	9	3	+	2	3	1	0	=		...
									▲					▲			

- en cada paso de la computación sólo un pequeño número de símbolos reciben atención (en el ejemplo, 2). Hay una cantidad finita de símbolos.
- la acción que se toma en cada paso depende sólo de los símbolos que están recibiendo atención y del estado actual de quien está haciendo el cálculo (en el ejemplo, la señorita está en estado *multiplicando* o *sumando*). Hay una cantidad finita de estados.

Las máquinas de Turing

Sin perder nada esencial, podemos suponer:

2	3	1	×	1	3	=	6	9	3	+	2	3	1	0	=	3	0	0	3

Empezamos con la multiplicación

2	3	1	×	1	3	=								...
		▲			▲									

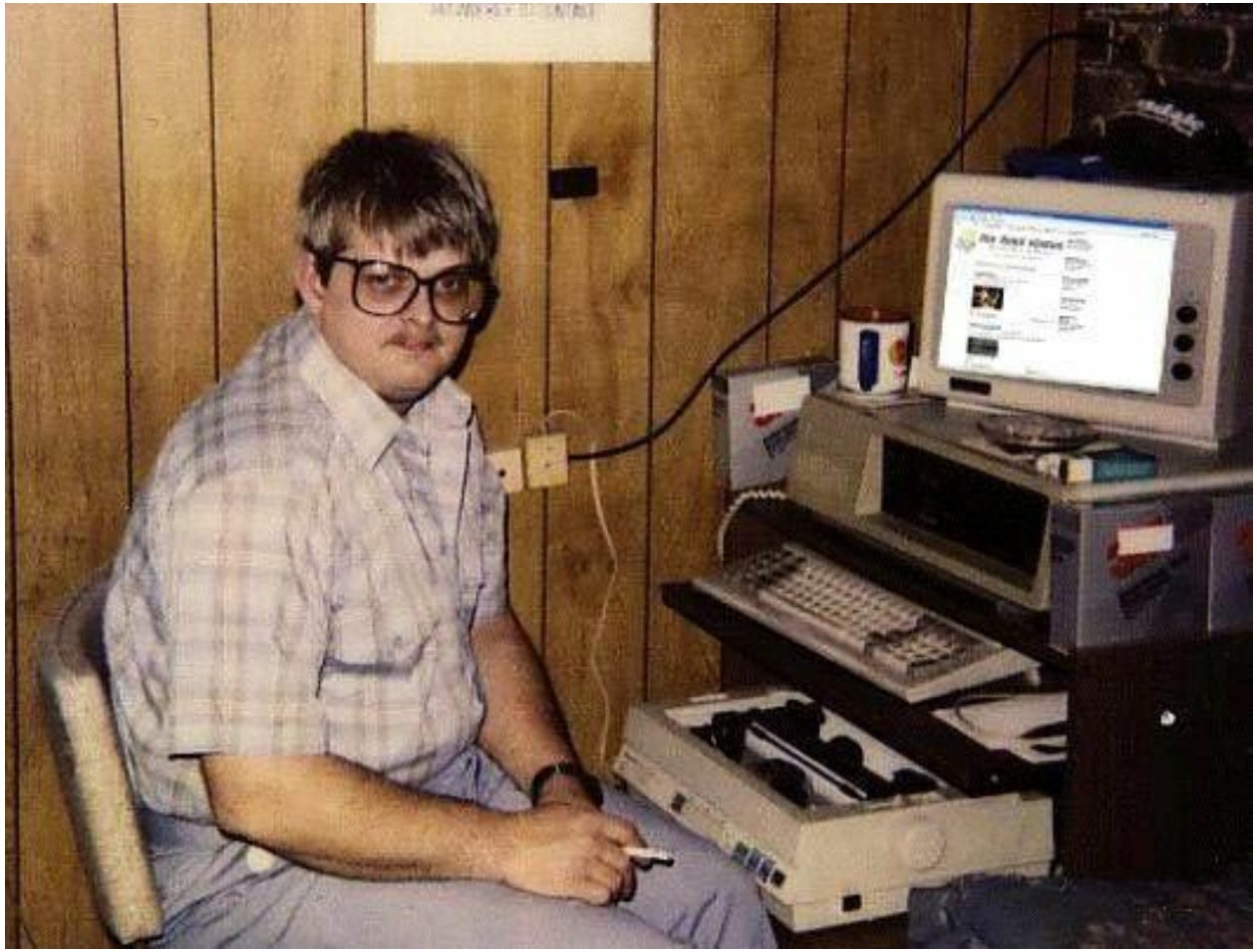
Después de un tiempo, vamos a estar sumando

2	3	1	×	1	3	=	6	9	3	+	2	3	1	0	=		...
									▲					▲			

- en cada paso de la computación sólo un pequeño número de símbolos reciben atención (en el ejemplo, 2). Hay una cantidad finita de símbolos.
- la acción que se toma en cada paso depende sólo de los símbolos que están recibiendo atención y del estado actual de quien está haciendo el cálculo (en el ejemplo, la señorita está en estado *multiplicando* o *sumando*). Hay una cantidad finita de estados.
- el procedimiento lleva una cantidad finita de pasos.

Intento 3 (más moderno) para *método efectivo*: lenguajes de programación

- Java
- C
- C++
- C#
- PHP
- (Visual) Basic
- Python
- Perl
- JavaScript
- ...

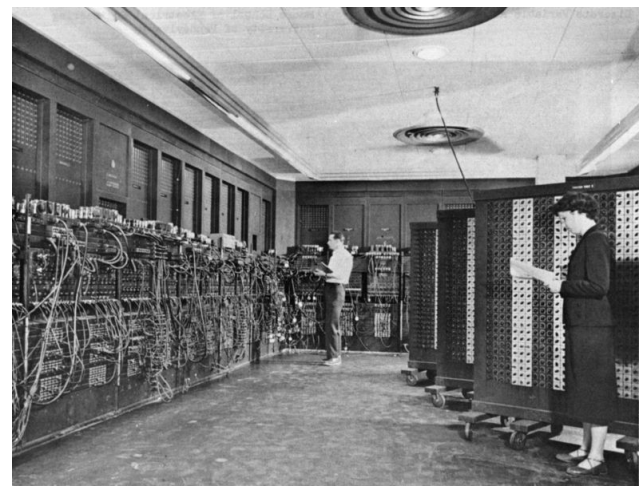


Entonces... ¿qué es un método efectivo?

Tesis de Church-Turing

- todos los lenguajes de programación y todas las computadoras son equivalentes a una máquina de Turing
- no es posible construir un dispositivo de cómputo que sea más poderoso que una computadora

=

=


Entonces, lo efectivo es lo que se puede computar en la computadora que más les guste y con el lenguaje de programación que más les guste.

Problemas fáciles y difíciles

Ahora sabemos qué es un método efectivo para resolver un problema.

Extendamos la noción de problema:

~~$f : \mathbb{N} \to \mathbb{N}$~~ $\qquad f : \mathit{texto} \to \mathit{texto}$

Podemos calcular cuántas operaciones nos lleva calcular el resultado.

- los problemas fáciles se resuelven con programas que ejecutan en pocas operaciones (rápidos)
- los problemas difíciles se resuelven con programas que necesariamente requieren muchas operaciones (lentos)

Ejemplos de problemas fáciles

- buscar un elemento en una lista de números

entrada	salida
"[1, 4, 5, 2, 7, 7, 5, 1, 1, 3] 4"	"sí"
"[1, 4, 5, 2, 7, 7, 5, 1, 1, 3] 3"	"sí"
"[1, 4, 5, 2, 7, 7, 5, 1, 1, 3] 6"	"no"

tiempo $\simeq$ longitud de la entrada

Ejemplos de problemas fáciles

- buscar un elemento en una lista de números

entrada	salida
"[1, 4, 5, 2, 7, 7, 5, 1, 1, 3] 4"	"sí"
"[1, 4, 5, 2, 7, 7, 5, 1, 1, 3] 3"	"sí"
"[1, 4, 5, 2, 7, 7, 5, 1, 1, 3] 6"	"no"

tiempo $\simeq$ longitud de la entrada

- ordenar una lista

entrada	salida
"[1, 4, 5, 2, 7, 7, 5, 1, 1, 3]"	"[1, 1, 1, 2, 3, 4, 5, 5, 7, 7]"

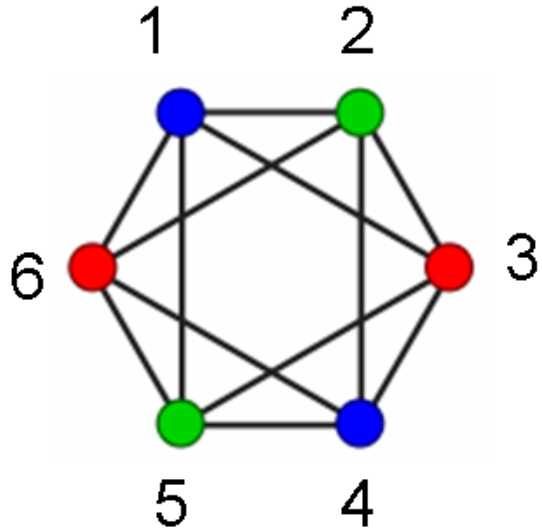
tiempo $\simeq$ longitud de la entrada^2

(aunque se puede hacer un poquito más rápido)

Ejemplos de problemas difíciles

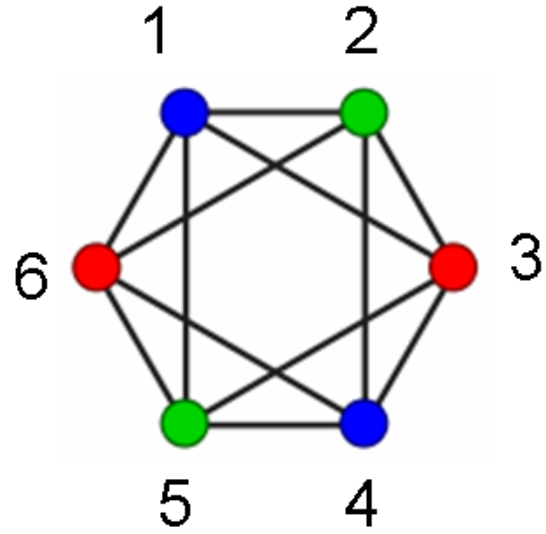
- coloreo de grafos
  ¿hay un coloreo que usa a lo sumo x colores?

entrada	salida
"1-2 1-3 1-5 1-6 2-3 ... 3"	"sí"
"1-2 1-3 1-5 1-6 2-3 ... 2"	"no"



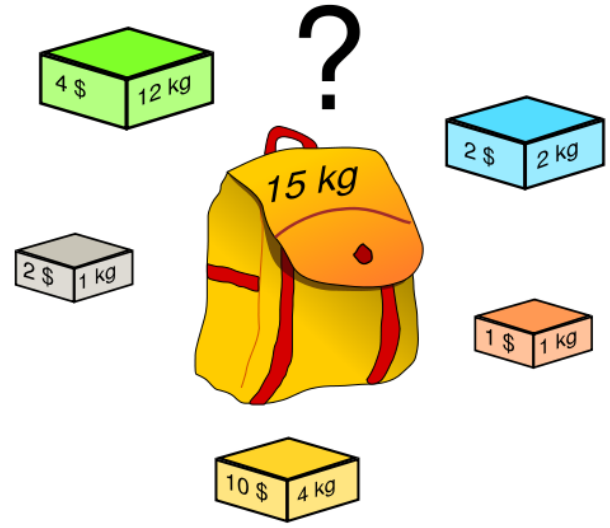
Ejemplos de problemas difíciles

- coloreo de grafos
  ¿hay un coloreo que usa a lo sumo x colores?



entrada	salida
"1-2 1-3 1-5 1-6 2-3 ... 3"	"sí"
"1-2 1-3 1-5 1-6 2-3 ... 2"	"no"

- el problema de la mochila
  ¿podemos juntar al menos $\$x$ en una mochila que carga hasta y Kg.?



entrada	salida
"[4,12] [2,2] [2,1] [1,1] [10,4] 15 15"	"sí"
"[4,12] [2,2] [2,1] [1,1] [10,4] 100 15"	"no"

Problemas fáciles y difíciles

entrada	salida
$\underbrace{\text{“}zzzzzzzzzzzzzzzzzzzzzzzzzz\text{”}}_{\text{longitud}=x}$	“sí” / ”no”

Problemas fáciles y difíciles

entrada	salida
“$\underbrace{zzzzzzzzzzzzzzzzzzzzzzzzzz}_{\text{longitud}=x}$”	“sí” / ”no”

- **P** = conjunto de problemas que se resuelven en tiempo polinomial
  - si la entrada tiene longitud x, entonces computar la salida lleva más o menos x^n pasos

Problemas fáciles y difíciles

entrada	salida
$\underbrace{\text{“zzzzzzzzzzzzzzzzzzzzzzzzzz”}}_{\text{longitud}=x}$	“sí” / ”no”

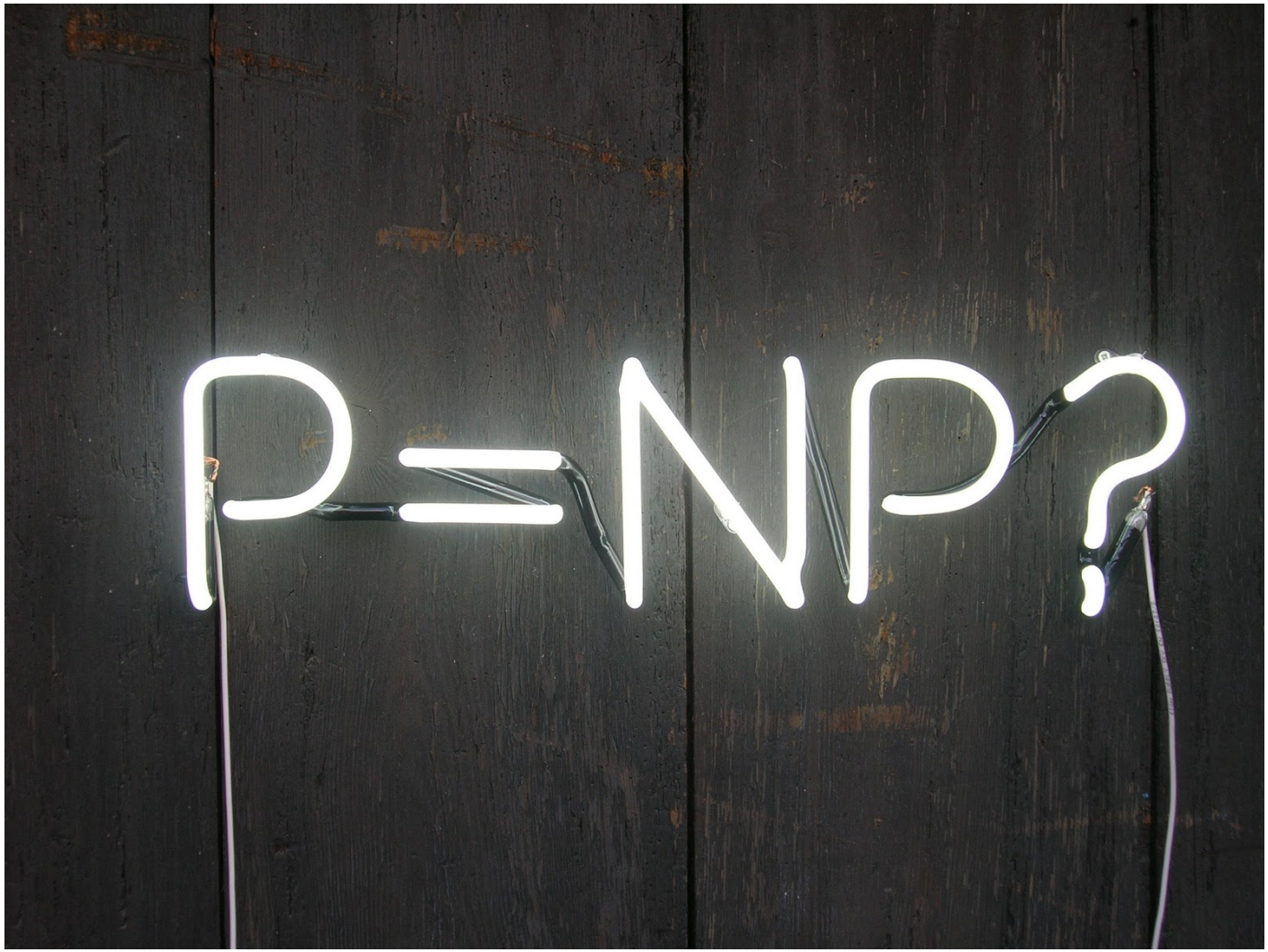
- **P** = conjunto de problemas que se resuelven en tiempo polinomial
  - si la entrada tiene longitud x, entonces computar la salida lleva más o menos x^n pasos

- **NP** = conjunto de problemas que se resuelven en tiempo polinomial pero usando no determinismo
  - si la entrada tiene longitud x, entonces verificar que una solución con salida “sí” es buena lleva más o menos x^n pasos.

Problemas fáciles y difíciles

entrada	salida
$\underbrace{\text{“}zzzzzzzzzzzzzzzzzzzzzzzzzz\text{”}}_{\text{longitud}=x}$	“sí” / ”no”

- **P** = conjunto de problemas que se resuelven en tiempo polinomial
  - si la entrada tiene longitud x, entonces computar la salida lleva más o menos x^n pasos
- **NP** = conjunto de problemas que se resuelven en tiempo polinomial pero usando no determinismo
  - si la entrada tiene longitud x, entonces verificar que una solución con salida “sí” es buena lleva más o menos x^n pasos.
- todo problema de **P** está en **NP**. Pero... ¿realmente hay problemas en **NP** que no estén en **P** o los dos conjuntos son iguales?
  ¡Nadie lo sabe! Es un problema muy famoso en computación.



Tiene muchas consecuencias prácticas. Por ejemplo muchos métodos de encriptar se volverían fáciles de romper.

¿Una computadora puede resolver cualquier problema?

- Hay ∞ problemas, o sea, funciones

$$f : \mathit{texto} \to \mathit{texto}$$

- Hay ∞ programas (no importa el lenguaje de programación).

¿Una computadora puede resolver cualquier problema?

- Hay ∞ problemas, o sea, funciones

$$f : \textit{texto} \to \textit{texto}$$

- Hay ∞ programas (no importa el lenguaje de programación).
- Pero el primer infinito es más grande que el segundo:

¿Una computadora puede resolver cualquier problema?

- Hay ∞ problemas, o sea, funciones

$$f : \textit{texto} \rightarrow \textit{texto}$$

- Hay ∞ programas (no importa el lenguaje de programación).
- Pero el primer infinito es más grande que el segundo:
  - los programas se pueden enumerar, o listar:

    Todos los programas de longitud 1
    Todos los programas de longitud 2
    Todos los programas de longitud 3...

  - las funciones $\textit{texto} \rightarrow \textit{texto}$ no se pueden enumerar

El argumento diagonal de Cantor

Veamos que las funciones

$$f : \textit{número} \to \textit{texto}$$

no se pueden enumerar (estas son un subconjunto de las funciones $\textit{texto} \to \textit{texto}$).
Por el absurdo. Supongamos que sí se pueden enumerar:

$$f_0, f_1, f_2, \ldots$$

Me defino una nueva función:

$$g : \textit{número} \to \textit{texto}$$

del siguiente modo:

$$g(n) = \text{“hola” seguido de } f_n(n)$$

Tenemos que g no está en la lista. Absurdo.

Hay problemas que ninguna computadora resuelve

$$\underset{\text{programas}}{\infty} \quad < \quad \underset{\text{problemas}}{\boldsymbol{\infty}}$$

Hay problemas que ninguna computadora resuelve

$$\underset{\text{programas}}{\infty} \quad < \quad \underset{\text{problemas}}{\infty}$$

- Hay más problemas que programas
- Tiene que haber problemas que no tengan una solución efectiva, o sea problemas que ninguna computadora puede resolver.
- Se llaman problemas indecidibles o incomputables.
- ¿Cuáles?

A veces la computadora se cuelga

A veces la computadora se cuelga

A veces la computadora se cuelga

A veces la computadora se cuelga

El problema de la parada

Es el problema indecidible más famoso

También se lo llama problema de la detención o halting problem, en inglés.

Tiene que ver con programas que se cuelgan y programas que no.

El problema de la parada

Dado un programa p, ¿p se cuelga?

Supongamos que $p(x)$ es el programa:

```
int i=1;
while (i>x) {
i=i+1;
}
```

- si $x = 0$ entonces p no para (se cuelga)
- si $x = 1$ entonces p para (no se cuelga)

El problema de la parada es indecdible

entrada	salida
“(`int i=1; while (i>x) {i=i+1;}`)(0)”	“sí”
“(`int i=1; while (i>x) {i=i+1;}`)(1)”	“no”

¿Existe un programa que recibe un texto y devuelve “sí” cuando el texto representa un programa que para y “no” en caso contrario?

El problema de la parada es indecdible

entrada	salida
"(`int i=1; while (i>x) {i=i+1;}`)(0)"	"sí"
"(`int i=1; while (i>x) {i=i+1;}`)(1)"	"no"

¿Existe un programa que recibe un texto y devuelve "sí" cuando el texto representa un programa que para y "no" en caso contrario?

No, no existe tal programa. El problema de la parada es indecidible.

- no importa en qué lenguaje queramos programarlo
- es un límite absoluto al poder de cómputo. Los métodos efectivos no llegan tan lejos.

Un rompecabezas mu difícil

- Eternity II

  - 256 piezas
  - bordes adyacentes tienen el mismo dibujo
  - se pueden rotar

Un rompecabezas mu difícil

- de 2007 a 2010 hubo 2 millones de dólares para el primero que encuentre una solución

Un rompecabezas mu difícil

- de 2007 a 2010 hubo 2 millones de dólares para el primero que encuentre una solución
- nadie encontró la solución
- hay más o menos $1{,}115 \cdot 10^{557}$ posibles configuraciones:
  1115000000000000000000000000000000000000000000
  0000000000000000000000000000000000000000000000
  0000000000000000000000000000000000000000000000
  0000000000000000000000000000000000000000000000
  0000000000000000000000000000000000000000000000
  0000000000000000000000000000000000000000000000
  0000000000000000000000000000000000000000000000
  0000000000000000000000000000000000000000000000
  0000000000000000000000000000000000000000000000
  0000000000000000000000000000000000000000000000
  0000000000000000000000000000000000000000000000
  0000000000000000000000000000000000000000000000
  000000

Otro problema indecidible: el tiling

- dado un conjunto de piezas, por ejemplo

- reglas:
  - bordes adyacentes tienen el mismo dibujo
  - no se pueden rotar
  - hay infinitas copias de cada pieza

Otro problema indecidible: el tiling

- dado un conjunto de piezas, por ejemplo

- reglas:
  - bordes adyacentes tienen el mismo dibujo
  - no se pueden rotar
  - hay infinitas copias de cada pieza
- ¿podemos ubicarlos de modo tal que cubran todo el plano?

Otro problema indecidible: el tiling

- dado un conjunto de piezas, por ejemplo

- reglas:
  - bordes adyacentes tienen el mismo dibujo
  - no se pueden rotar
  - hay infinitas copias de cada pieza
- ¿podemos ubicarlos de modo tal que cubran todo el plano?
- no hay ningún programa que dado un conjunto finito de azulejos, decida si se puede cubrir todo el plano o no (teniendo en cuenta las reglas).

Resumen

- ahora es fácil definir un método efectivo: es un programa.
  - pero se empezó a pensar en esto mucho antes de que existieran las computadoras
- se sabe que hay problemas fáciles (se resuelven rápido)
  - el problema de buscar un elemento en una lista
  - el problema de ordenar una lista
- hay un montón de problemas más difíciles para los que no se conoce una solución rápida (puede que exista pero que nadie la haya encontrado; o puede que no exista)
  - el problema del coloreo de grafos
  - el problema de la mochila
- hay problemas tan difíciles que ninguna computadora (ni ningún método efectivo) puede resolver
  - el problema de la parada
  - el problema del tiling