Algoritmos y computabilidad

Exploratorio Computación 2'2019

Luis Ramirez / Vicente Dominguez

Invitado: Cristian Riveros

¿qué es un algoritmo?

Algorithm

From Wikipedia, the free encyclopedia

> *"In mathematics and computer science, an algorithm is a sequence of instructions, typically to solve a class of problems or perform a computation. [. . .]*
>
> *As an effective method, an algorithm can be expressed within a finite amount of space and time and in a well-defined formal language for calculating a function. Starting from an initial state and initial input (perhaps empty), the instructions describe a computation that, when executed, proceeds through a finite number of well-defined successive states, eventually producing output and terminating at a final ending state. [. . .]"*

¿qué es un algoritmo?

Algorithm

From Wikipedia, the free encyclopedia

> *"In mathematics and computer science, an algorithm is a sequence of instructions, typically to solve a class of problems or perform a computation. [. . .]*
>
> *As an effective method, an algorithm can be expressed within a finite amount of space and time and in a well-defined formal language for calculating a function. Starting from an initial state and initial input (perhaps empty), the instructions describe a computation that, when executed, proceeds through a finite number of well-defined successive states, eventually producing output and terminating at a final ending state. [. . .]"*

¿qué es un algoritmo?

Ejemplo

```
10   INPUT INT A > 0, INT B > 0
20   IF B=0 THEN GOTO 80
30   IF A > B THEN GOTO 60
40   LET B=B-A
50   GOTO 20
60   LET A=A-B
70   GOTO 20
80   PRINT A
90   END
```

- Algoritmo de Euclides

"...one of the oldest algorithms still in common use."

Wikipedia.

¿qué es un algoritmo?

Ejemplo

$t := 3$

while *TRUE* **do**

  for $n = 3$ **to** t **do**

    for $x = 1$ **to** t **do**

      for $y = 1$ **to** t **do**

        for $z = 1$ **to** t **do**

          if $x^n + y^n = z^n$ **then**

            return `TRUE`

  $t := t + 1$

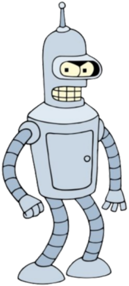
¿qué es un algoritmo?

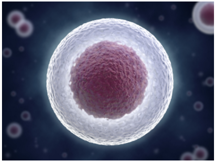
La definición deja varias preguntas sin resolver:

- ¿cuáles son los posibles "estados"?
- ¿cuáles son los posibles "inputs"?
- ¿cuáles son las posibles "instrucciones"?

¿cómo puedo saber si mi algoritmo "termina"?

¿qué significa computar?





Interludio

Definición

Un **quine** es un programa que se imprime a si mismo.

Ejemplo

```
Imprime lo siguiente dos veces, la segunda vez con comillas
‘‘Imprime lo siguiente dos veces, la segunda vez con comillas’’
```

```
s = 's = %r \n print(s%%s)'
print(s%s)
```

¿conocen algún “**quine**”?

¿qué significa computar?

Veamos dos historias paralelas:

- Historia 1: los inicios del computador.
- Historia 2: los inicios de la ciencia de la computación.

Historia 1: los inicios del computador

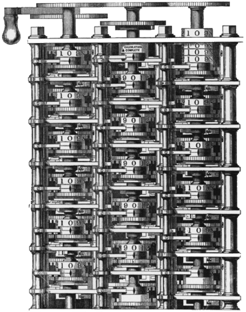
– 1599	Sistemas de conteo, ábacos, ...
1600 – 1799	Primera aparición de la palabra “computador”.

Historia 1: los inicios del computador

– 1599	Sistemas de conteo, ábacos, ...
1600 – 1799	Primera aparición de la palabra "computador".
	Primeros sistemas mecánicos de conteo (Pascal, Leibniz, ...).

Historia 1: los inicios del computador

– 1599	Sistemas de conteo, ábacos, ...
1600 – 1799	Primera aparición de la palabra "computador".
	Primeros sistemas mecánicos de conteo (Pascal, Leibniz, ...).
1800 – 1850	La máquina diferencial de C. Babbage.



(ver video)

Historia 1: los inicios del computador

– 1599	Sistemas de conteo, ábacos, ...
1600 – 1799	Primera aparición de la palabra "computador".
	Primeros sistemas mecánicos de conteo (Pascal, Leibniz, . . .).
1800 – 1850	La máquina diferencial de C. Babbage.
	La máquina analítica de C. Babbage.

Historia 1: los inicios del computador

– 1599	Sistemas de conteo, ábacos, ...
1600 – 1799	Primera aparición de la palabra "computador". Primeros sistemas mecánicos de conteo (Pascal, Leibniz, . . .).
1800 – 1850	La máquina diferencial de C. Babbage. La máquina analítica de C. Babbage.
1940 –	la **era moderna** de los computadores comienza . . .

Historia 2: los inicios de la ciencia de la computación

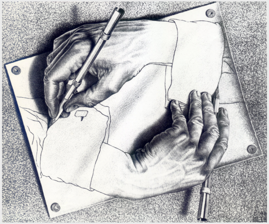
Finales del siglo XIX.

- Grandes contradicciones en los "cimientos" de las matemáticas.

Ejemplo

Paradoja de Russell: ¿es posible definir el siguiente conjunto?

$$R = \{S \text{ es un conjunto } \mid S \notin S\}$$



¿cómo formalizar los "cimientos" de las matemáticas sin contradicciones?

(parentesís diofántico en nuestra historia)

Definición

Sea $p(x_1, \ldots, x_n)$ un **polinomio** con coeficientes en los **números enteros** $\mathbb{Z}$.

Una **ecuación diofántica** es una equación de la forma $p(x_1, \ldots, x_k) = 0$ donde las soluciones estan restringidas a los **números enteros** $\mathbb{Z}$.

Ejemplo

Las siguientes son ecuaciones diofánticas.

- $3x + 7y - 1 = 0$
- $x^2 + x - 2 = 0$
- $x^3 + y^3 - z^3 = 0$

¿cuál de ellas tienen alguna solución entera?

Historia 2: los inicios de la ciencia de la computación

Principios del Siglo XX.

- David Hilbert, en la conferencia internacional de matemáticas (1900):

  "Problema 10: Dada una **ecuación diofántica** *con cualquier número de incógnitas y con coeficientes numéricos racionales enteros:*

  Idear un **proceso efectivo** *de acuerdo con el cual pueda determinarse, en un número finito de operaciones, si la ecuación es resoluble en números racionales enteros."*

¿cómo formalizamos la idea de "**proceso efectivo**"?

Historia 2: los inicios de la ciencia de la computación

Desde 1900 hasta 1930 vinieron muchas propuestas

Funciones parcialmente recursivas
por K. Godel, J. Herbrand, S. Kleene.

λ-*calculus*
por Alonso Church.

Máquinas de Turing
por Alan Turing.

...

¿cuál de todas ellas definen la idea de "**proceso efectivo**"?

Máquinas de Turing

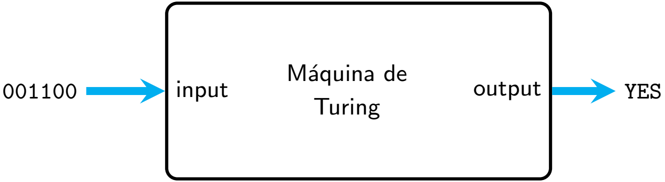
Inventadas en 1936, por **Alan Turing** ("el padre de la computación").

Inventadas mucho antes que existiera los **computadores modernos**.

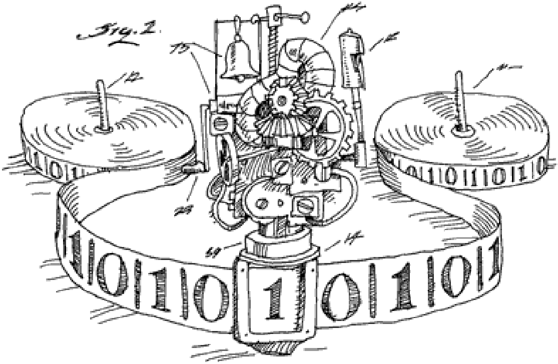
Máquinas de Turing

- Modelo abstracto de una máquina.
- Recibe como **input** una secuencia de símbolos.
  - Ejemplo: `001100`.
- Responde como **output**:

  YES o NO o $\varnothing$ (no detenerse)



Máquinas de Turing



Componentes de una máquina de Turing

1. **Símbolos**: Un conjunto finito de símbolos predefinidos.

Ejemplos

- Binario: $\{0, 1\}$.
- Númerico: $\{0, 1, 2, \ldots, 9\}$.
- Alfabeto: $\{a, b, \ldots, z, A, B, \ldots, Z\}$.
- ASCII: $\{a, b, \ldots, z, A, B, \ldots, Z, !, \#, \$, \ldots\}$.

Componentes de una máquina de Turing

1. **Símbolos**: Un conjunto finito de símbolos predefinidos.

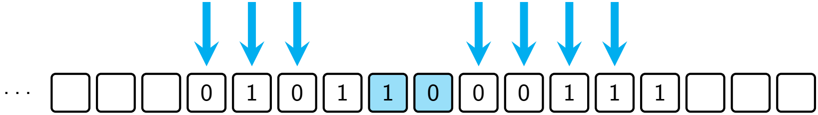
2. **Memoria**: Un arreglo o **cinta infinita** de celdas contiguas que en cada posición almacena un símbolo o está vació.

Componentes de una máquina de Turing

1. **Símbolos**: Un conjunto finito de símbolos predefinidos.

2. **Memoria**: Un arreglo o **cinta infinita** de celdas contiguas que en cada posición almacena un símbolo o está vació.

3. **Cabeza Lectora**: Un **cabezal** sobre la cienta que se mueve en ambas direcciones y lee y modifica las celdas.



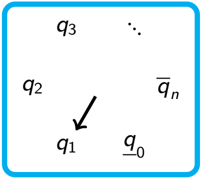
Componentes de una máquina de Turing

1. **Símbolos**: Un conjunto finito de símbolos predefinidos.

2. **Memoria**: Un arreglo o **cinta infinita** de celdas contiguas que en cada posición almacena un símbolo o está vació.

3. **Cabeza Lectora**: Un **cabezal** sobre la cienta que se mueve en ambas direcciones y lee y modifica las celdas.

4. **Control**: Un conjunto finito de **estados** o "memoria finita" junto con un estado **inicial** y un estado **final**.



Componentes de una máquina de Turing

1. **Símbolos**: Un conjunto finito de símbolos predefinidos.

2. **Memoria**: Un arreglo o **cinta infinita** de celdas contiguas que en cada posición almacena un símbolo o está vació.

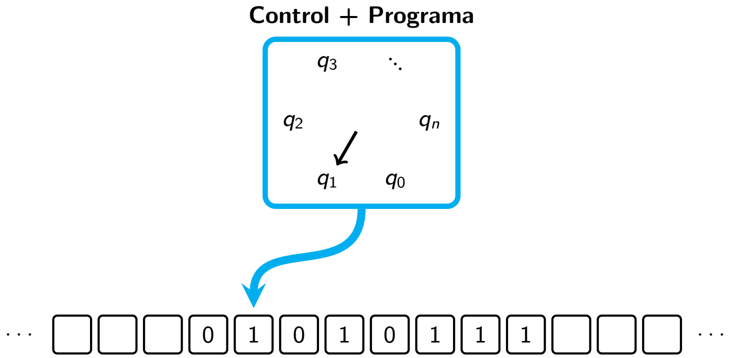
3. **Cabeza Lectora**: Un **cabezal** sobre la cienta que se mueve en ambas direcciones y lee y modifica las celdas.

4. **Control**: Un conjunto finito de **estados** o "memoria finita" junto con un estado **inicial** y un estado **final**.

5. **Programa**: Conjunto de instrucciones.

   "Si estoy en el estado q y la cabeza lectora lee un 0*, entonces cambio al estado p, modifico el* 0 *por un* 1 *y muevo la cabeza lectora a la derecha."*

Componentes de una máquina de Turing



Programación de una máquina de Turing

Para programar una Máquina de Turing el usuario debe entregar:

- El conjunto de **símbolos**.
  - *Ejemplo*: $\{0, 1, \text{␣}\}$ donde ␣ significa vació.
- El conjunto de **estados**.
  - *Ejemplo*: $\{q_0, q_1, q_2, q_3\}$
- La tabla de **transiciones**:

E_{actual}	S_{actual}	E_{sgte}	S_{nuevo}	D
q_0	0	q_1	0	$\rightarrow$
q_0	1	q_2	1	$\leftarrow$
q_0	␣	q_2	0	$\downarrow$
q_1	0	q_3	1	$\rightarrow$

- Estado **inicial** y estado **final**.
  - *Ejemplo*: estado inicial es q_0 y el estado final es q_3.

Programación de una máquina de Turing

Definición formal

Un **máquina de Turing** es una estructura:

$$\mathcal{M} \quad = \quad (\Sigma, Q, \delta, q_0, q_f)$$

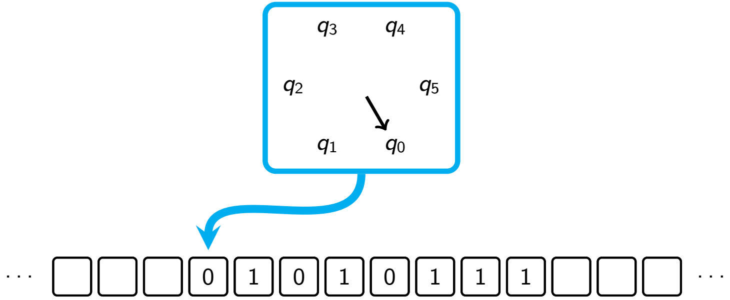
- Σ es el conjunto de **símbolos**.
- Q es un conjunto finito de **estados**.
- $\delta : Q \times \Gamma \rightharpoonup Q \times \Gamma \times \{\leftarrow, \rightarrow, \downarrow\}$ es la **función parcial de transición**.
- q_0 es el estado **inicial**.
- q_f es el estado **final**.

Funcionamiento de una máquina de Turing

Configuración inicial

Para un input **w**, la máquina inicialmente:

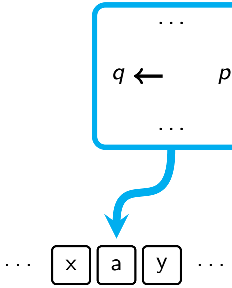
- Coloca **w** en alguna parte de la cinta.
- La cabeza lectora se posiciona en la primera celda del input.
- La máquina queda en el estado inicial $\mathbf{q_0}$.



Funcionamiento de una máquina de Turing

En cada paso

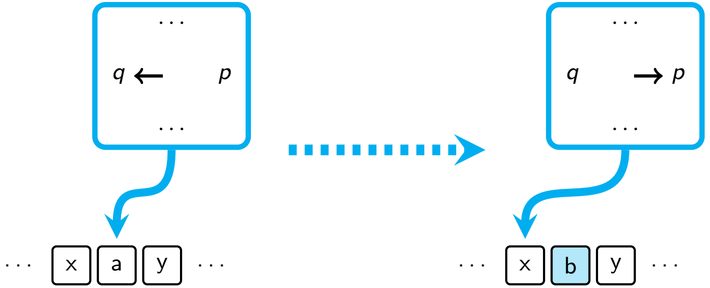
- La máquina lee el símbolo **a** en la celda apuntada por la cabeza lectora y determina en que estado **q** se encuentra.
- Busca en el programa una instrucción para $(\mathbf{q}, \mathbf{a})$. Si esta instrucción no existe, entonces la máquina se detiene.



Funcionamiento de una máquina de Turing

En cada paso

- Si la instrucción $(\mathbf{q}, \mathbf{a}) \rightarrow (\mathbf{p}, \mathbf{b}, \mathbf{d})$ existe, entonces la ejecuta:
  - Pasa al nuevo estado **p**,
  - Escribe el simbolo **b** apuntado por la cabeza lectora,
  - Mueve la cabeza lectora según la dirección **d**.



Funcionamiento de una máquina de Turing

Configuración final

- Si la máquina se **detiene** en un estado **final**, entonces responde `YES`.
- Si la máquina se **detiene** en un estado **NO final**, entonces responde `NO`.

La máquina puede NO detenerse!

Ejemplo de una máquina de Turing

Problema a resolver:

Verificar si el input tiene una cantidad par de 0's.

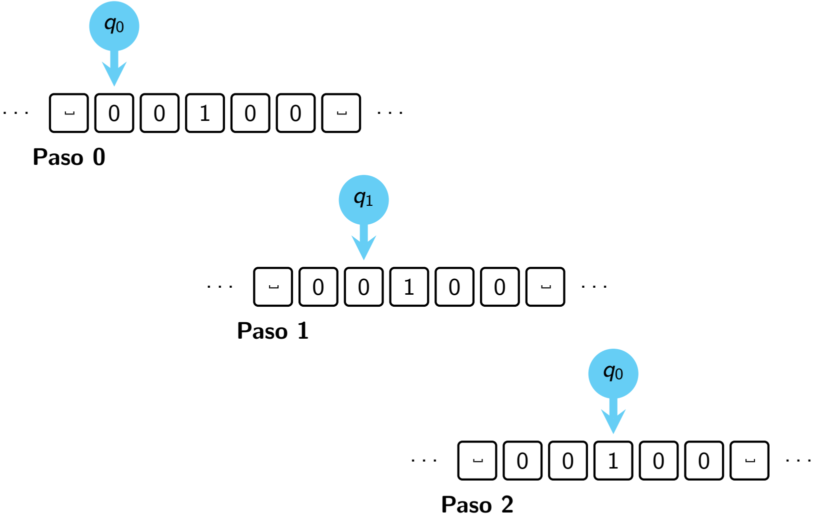
Solución:

- Simbolos: $\Sigma = \{0, 1\}$.
- Estados: $Q = \{q_0, q_1, q_f\}$
- Instrucciones:

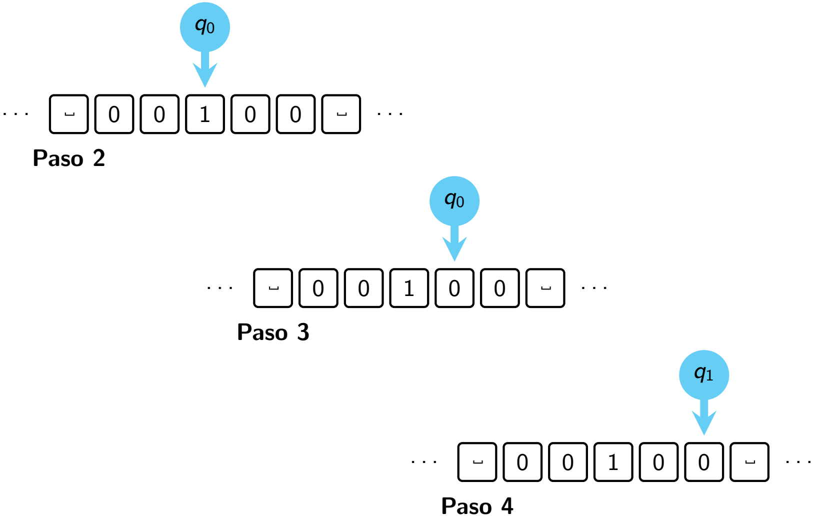
E_{actual}	S_{actual}	E_{sgte}	S_{nuevo}	D
q_0	0	q_1	0	$\rightarrow$
q_0	1	q_0	1	$\rightarrow$
q_1	0	q_0	0	$\rightarrow$
q_1	1	q_1	1	$\rightarrow$
q_0	$\sqcup$	q_f	$\sqcup$	$\downarrow$

- Estado inicial: q_0.
- Estado final: q_f.

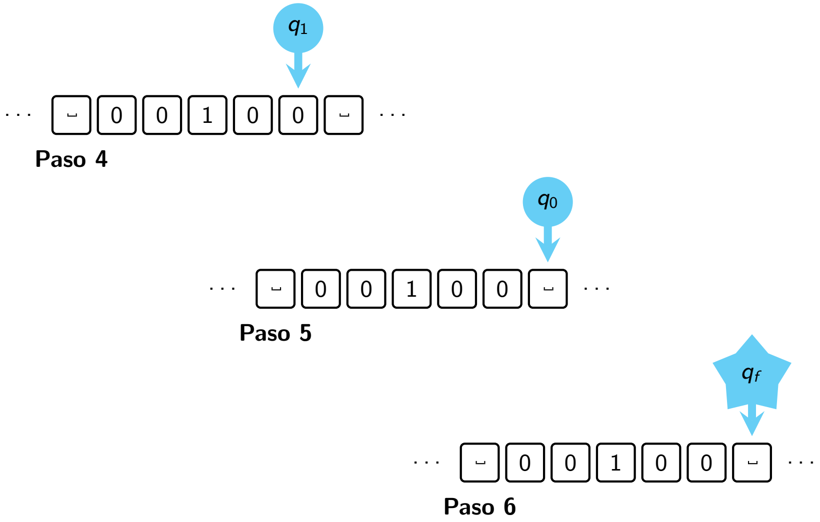
Ejecución con input 00100



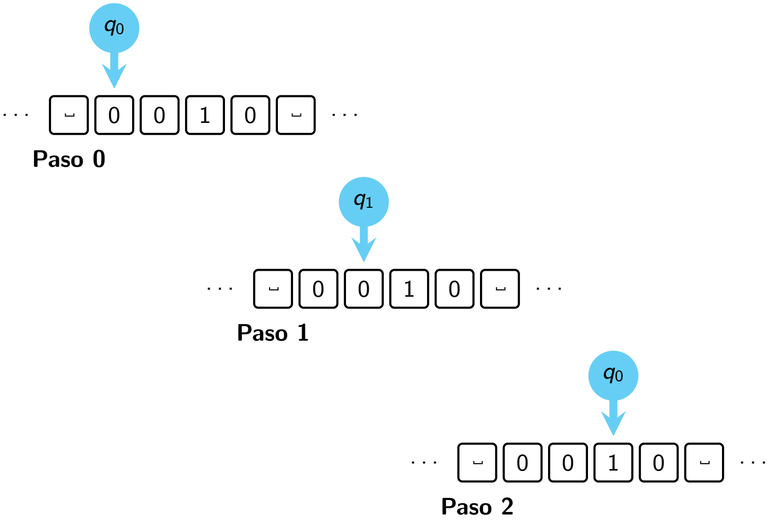
Ejecución con input 00100



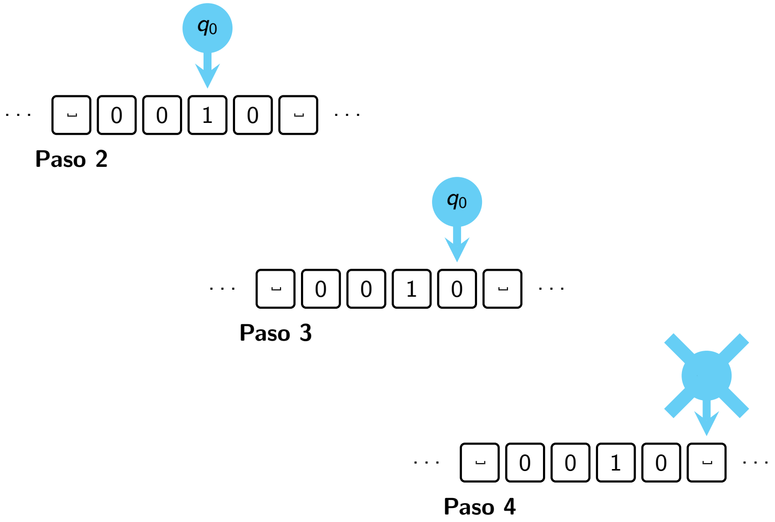
Ejecución con input 00100



Ejecución con input **0010**



Ejecución con input **0010**



Simulador de máquinas de Turing

TURING MACHINE

https://turingmachinesimulator.com/

- Creado y mantenido por **Martin Ugarte** (Ex-alumno doctorado, DCC)
- Simulador online de Máquinas de Turing.

Historia 2: los inicios de la ciencia de la computación

Desde 1900 hasta 1930 vinieron muchas propuestas:

Funciones parcialmente recursivas
por K. Godel, J. Herbrand, S. Kleene.

λ-*calculus*
por Alonso Church.

Máquinas de Turing
por Alan Turing.

...

¿cuál de todas ellas definen la idea de "**proceso efectivo**"?

Todas definen el mismo fenómeno

Funciones parcialmente recursivas	$\equiv$	
λ-calculus	$\equiv$	
Máquinas de Turing	$\equiv$	
...	$\equiv$	
Computador (moderno)	$\equiv$	Todos pueden "computar" los mismos problemas

Tesis de Church-Turing

Todo proceso efectivo ("computable") es equivalente a una máquina de Turing.

Es posible ejecutar cualquier algoritmo en una máquina de Turing!

Todas definen el mismo fenómeno

Implicación mas importante de la Tesis de Church-Turing:

> *"Existen problemas para el cual NO existe*
> *proceso efectivo (algoritmo) que entregue su solución."*

¿cuáles son estos **problemas sin solución**?

Detención de una función en phyton

Dada una función en python:

```
def mifuncion( arg1 ):
  ...
  return;
```

¿se detiene `mifunción` en algún momento si la ejecuto con el input 10?

Problema Detención de una función en python

Input: - código de una función en python f
- input w

Output: si al ejecutar f con w la ejecución termina.

¿existe algún algoritmo que resuelva este problema?

Detención de una función en phyton

Supongamos que **existe** un programa en python:

```
def haltingChecker( func, arg ):
```

Responde `true` si, y solo si, al ejecutar `func(arg)` el computador se detendrá.

Construyamos la siguiente función:

```
def invertAnswer( func ):
   if haltingChecker( func, func ) == True:
      while True:
         print 'loop';
   else:
      return;
```

¿qué ocurre si ejecuto `invertAnswer( invertAnswer )`?

Detención de una función en phyton

```
def invertAnswer( func ):
    if haltingChecker( func, func ) == True:
        while True:
            print 'loop';
    else:
        return;
```

¿qué ocurre si ejecuto `invertAnswer( invertAnswer )`?

Supongamos que `invertAnswer` **se detiene** con su propio código:

- `haltingChecker(invertAnswer, invertAnswer)` responde `True`.
- `invertAnswer` se queda en el while (**NO se detiene**).

Supongamos que `invertAnswer` **NO se detiene** con su propio código:

- `haltingChecker(invertAnswer, invertAnswer)` responde `False`.
- `invertAnswer` retorna (**se detiene**).

Detención de una función en phyton

Como llegamos a una **contradicción**, el error fue suponer que existe:

```
def haltingChecker( func, arg ):
```

Por lo tanto, **NO** existe un algoritmo para el problema:

Algoritmo	Detención de una función en python
Input:	código de una función en python f input w
Output:	si al ejecutar f con w la ejecución termina.

Decimos que el problema de la detención de un programa es **indecidible**.

Historia 2: los inicios de la ciencia de la computación

Implicación mas importante de la Tesis de Church-Turing:

> *"Existen problemas para el cual NO existe proceso efectivo (algoritmo) que entregue su solución."*

¿existen más problemas sin solución?

Un problema de nuestra historia

> *"Problema 10: Dada una* **ecuación diofántica** *con cualquier número de incógnitas y con coeficientes numéricos racionales enteros:*
>
> *Idear un proceso efectivo de acuerdo con el cual pueda determinarse, en un número finito de operaciones, si la ecuación es resoluble en números racionales enteros."*

...FIN!

¿qué significa computar?

Algunas conclusiones de estas dos historias . . .

1. Es posible entender la computación sin conocer los computadores.

2. La computabilidad es un fenómeno intrínseco del universo.

3. Entender el "computar" nos entrega resultados inesperados.

"Computer science is not about machines,
in the same way that astronomy is not about telescopes."

Edsger Dijkstra