Introducción a la Computabilidad y Complejidad

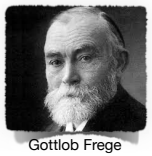
Diseño de Algoritmos

2021

Límites

¿qué problemas NO podemos computar, o NO tenemos tanto tiempo para esperar la solución?

¿ Qué es computar ?

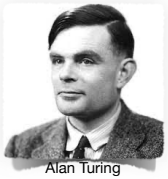


Gottlob Frege

Leyes Básicas de la Aritmética

David Hilbert

Congreso de Paris

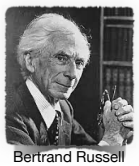


Alan Turing

Halting Problem

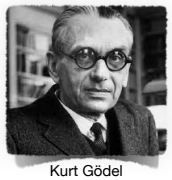
1850 1900 1950

Paradoja de Russell



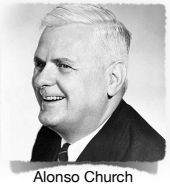
Bertrand Russell

Teorema de Incompletitud



Kurt Gödel

Tesis de Church



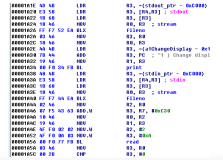
Alonso Church

Formalismos de Cómputo

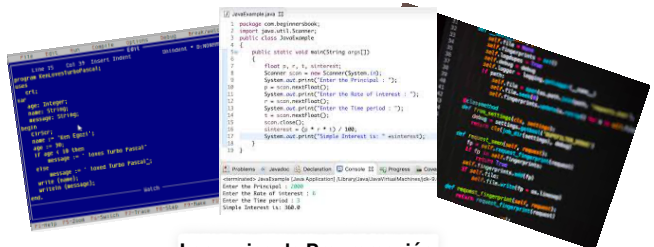
… y la Tesis de Church-Turing

Máquinas de Turing

Lambda Cálculo



Máquinas RAM (assembler)

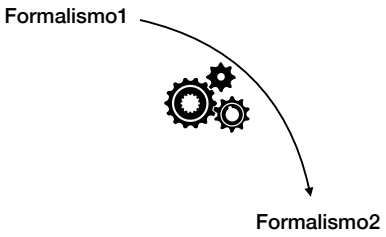


Lenguajes de Programación

Formalismos de Cómputo

… y la Tesis de Church-Turing

Compiladores



Intérpretes

¿ Qué no podemos computar ?

Halting Problem

```
//imagina que existe el código para este programa
//es decir, es computable
program Halt ( program p ) : boolean

//construyamos este programa correcto que usa Halt
program Q (program p) {
   boolean r = Halt(p);
   if r
      while true do skip
   else
      System.out.print("Terminé")
}

//corramos Q: ¿Qué debería suceder?
> run Q(Q)
```

¡¡¡ ***Q*** Termina si y sólo si ***Q*** NO Termnia !!!!!

¿Cómo podemos razonar sobre la

eficiencia (complejidad) para computar los problemas ?

¿Cómo podemos medir la complejidad de computar un problema?

¿Da lo mismo computarlos con cualquier Formalismo?

¿Hay problemas más difíciles de solucionar que otros?

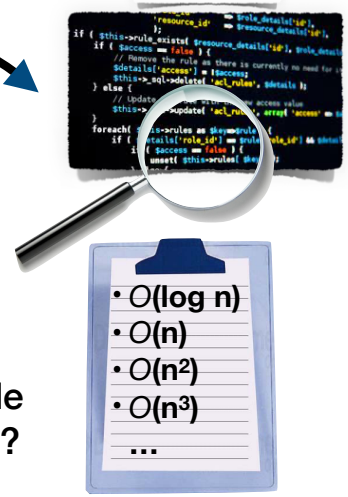
¿Hay problemas que computar su solución puede **demorar** más de lo que podemos esperar?

¿Puede el avance tecnológico colaborar?

¿Cómo podemos medir la complejidad de computar un problema?

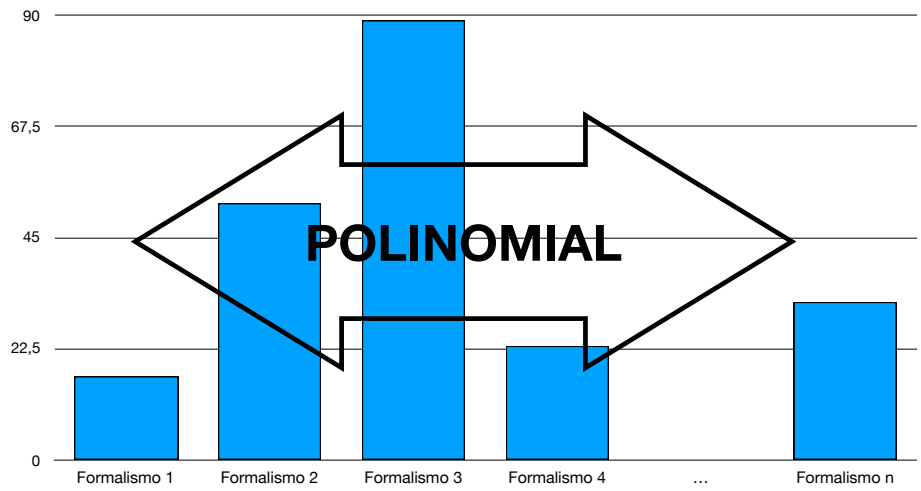
Problema

Programa



pero… ¿ si cambiamos de formalismo de cómputo ?

¿Cuál es la diferencia de eficiencia entre los diferentes formalismos de cómputo?



Denominaremos la clase de problemas **P** a aquellos que pueden computarse en *tiempo* **POLINOMIAL** por alguno de los formalismos mencionados (oficialmente por una Máquina de Turing **Determinística**)

Vamos a considerar a este conjunto de problemas como TRATABLES

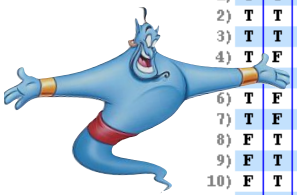
¿Hay problemas más complejos de los de la clase P ? ¿cómo podemos clasificarlos ?

La Clase NP

problema de ejemplo: SAT

¿ No Determinismo como herramienta de cómputo ?

¿ Qué es más fácil, solucionar o comprobar ?



	P	Q	R	S	(P v Q => R) & (R => S) => (~S => ~P)
0)	T	T	T	T	T T T T T F T F
1)	T	T	T	F	T T F F T T F F
2)	T	T	F	T	T F F T T F T F
3)	T	T	F	F	T F F T T T F F
4)	T	F	T	T	T T T T T F T F
			T	F	T T F F T T F F
6)	T	F	F	T	T F F T T F T F
7)	T	F	F	F	T F F T T T F F
8)	F	T	T	T	T T T T T F T T
9)	F	T	T	F	T T F F T T T T
10)	F	T	F	T	T F F T T F T T
11)	F	T	F	F	T F F T T T T T
12)	F	F	T	T	F T T T T F T T
13)	F	F	T	F	F T F F T T T T
14)	F	F	F	T	F T T T T F T T
15)	F	F	F	F	F T T T T T T T
					^ ^ ^ ^ ^ ^ ^ ^
					1 2 4 3 8 5 7 6

vs.

¿Cómo sabemos si estamos ante el dilema

…y no morir en el intento

P = ó ≠ NP ?

Existen Problemas COMPLETOS para las clases:

“El problema de determinar si una fórmula de la lógica de predicados es satisfactible (SAT) es completo para la clase NP” - ***Stephen Cook 1971***

Otros problemas conocidos NPC

Subconjunto de suma k

Partición de conjunto de suma igual

Grafos: Clique (máximo sub-grafo completo), Coloreo, Cubrimiento de vértices

Problema del viajante

…

A no tirar la toalla

…existen técnicas para abordar estos problema , relajando nuestras expectativas

El vaso medio lleno

…aprovechando la complejidad para mejorar la seguridad