Computando lo áspero

¿Por qué se cuelgan las computadoras?

Santiago Figueira

Departamento de Computación
FCEyN, UBA

Semana de la Computación 2008

Historia de Hilbert y sus amigos

David Hilbert, 1900: Formalicemos la matemática. Si somos suficientemente cuidadosos, en el futuro vamos a poder construir una máquina que demuestre teoremas y así encuentre todas las verdades matemáticas.

Kurt Gödel, 1931: Hay más verdades matemáticas (sobre los números naturales) que teoremas. Hay verdades no demostrables.

David Hilbert: Chapeau!

Historia de Hilbert y sus amigos

Alan Turing: Hola, yo soy Alan. ¿Cómo una máquina que demuestre teoremas? ¿Una máquina de qué tipo?

David Hilbert: No sé... se me ocurrió. Me refiero a un método efectivo. Algo que se pueda hacer mecánicamente.

Alan Turing, 1936: Construí un modelo formal de cómputo, la Máquina de Turing, y demostré que hay problemas que esta máquina no puede resolver.

Historia de Hilbert y sus amigos

David Hilbert: Bueno... yo pensé que...

Kurt Gödel: Ta bien... ¡Igual era una re buena idea!

Alan Turing: Sí, todo bien, David.

Métodos efectivos

- eficaz: "el equipo fue efectivo" quiere decir que el equipo jugó mal pero metió goles
- con billetes: "solo acepto efectivo" quiere decir que no acepto tarjeta
- mecánico, realizable. En esto pensaba Hilbert.

¿Cómo podemos definir los métodos efectivos?

Los métodos resuelven problemas pero 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:

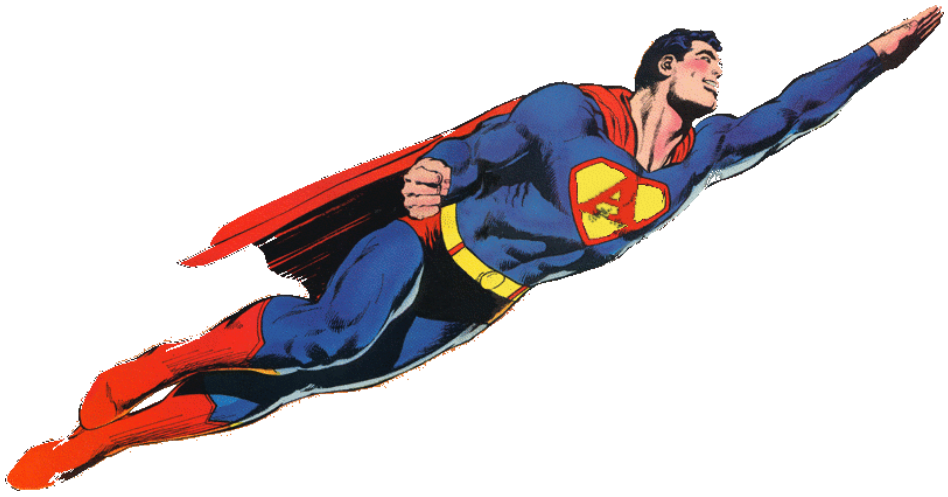
- 0: el cero que todos conocemos
- $+1$: sumarle 1 a un número
- composición de funciones. Por ejemplo $0 + 1 = 1$. También puedo al 1 sumarle 1 y obtener el 2: $(0 + 1) + 1 = 2$
- recursión: una función que se llama a sí misma. Por ejemplo, a la función *suma* se puede definir como:

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

Entonces,

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

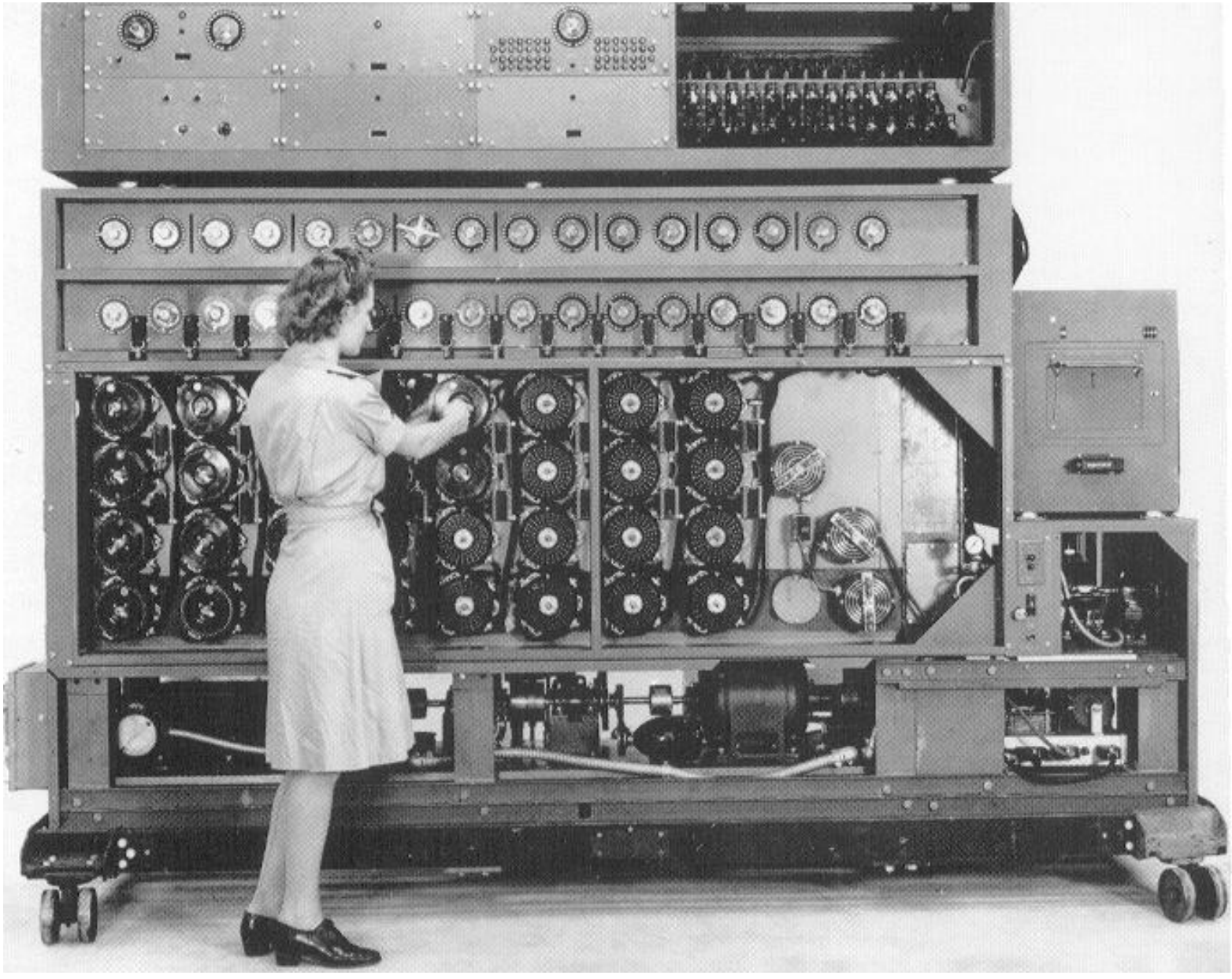
La función de ACKERMANN



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

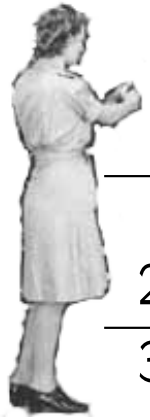
- para valores chicos de m, como 1, 2, o 3, A crece relativamente lento con respecto a n
- para $m \geq 4$, crece más rápido:
  - $A(4,2) \simeq 2 \times 10^{19728}$
  - $A(4,3) >$ cantidad de partículas del universo

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?

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

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

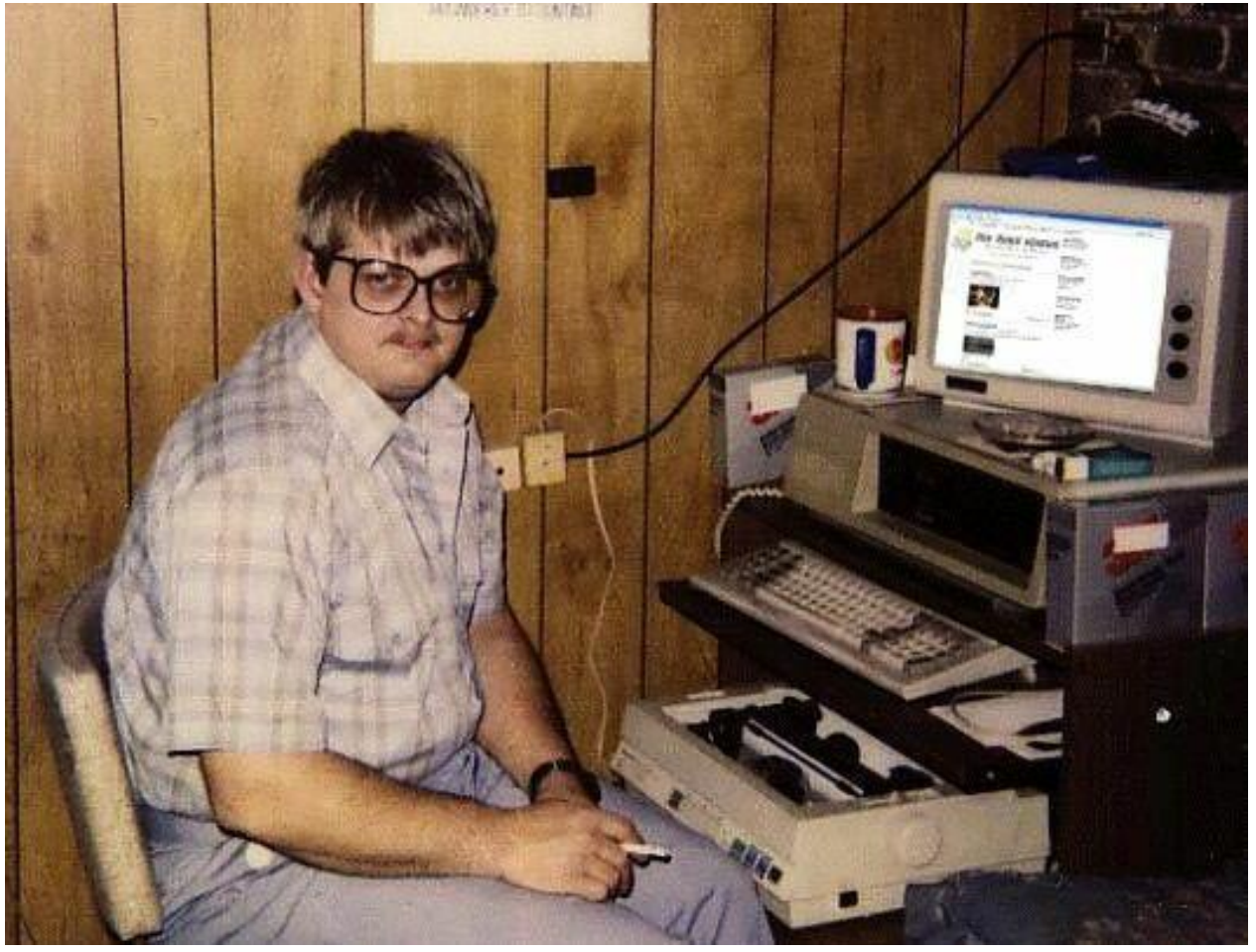
Después de un tiempo, vamos a estar sumando

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

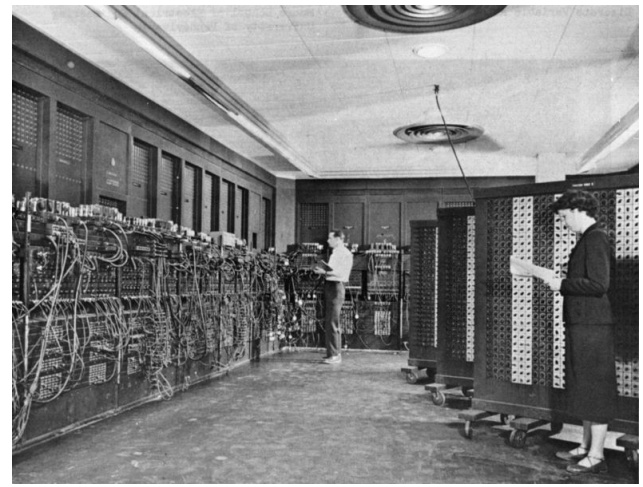


Entonces... ¿qué es un método efectivo?

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:

$f : \mathbb{N} \longrightarrow \mathbb{N}$ $\qquad$ $f : \mathit{texto} \longrightarrow \mathit{texto}$

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

- ordenar una lista

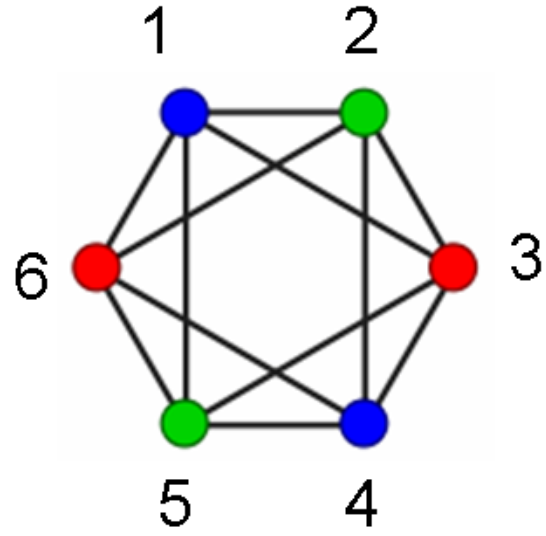
entrada	salida
"[1, 4, 5, 2, 7, 7, 5, 1, 1, 3]"	"[1, 1, 1, 2, 3, 4, 5, 5, 7, 7]"

tiempo $\simeq$ longitud de la entrada^2

(aunque se puede hacer un poquito más rápido)

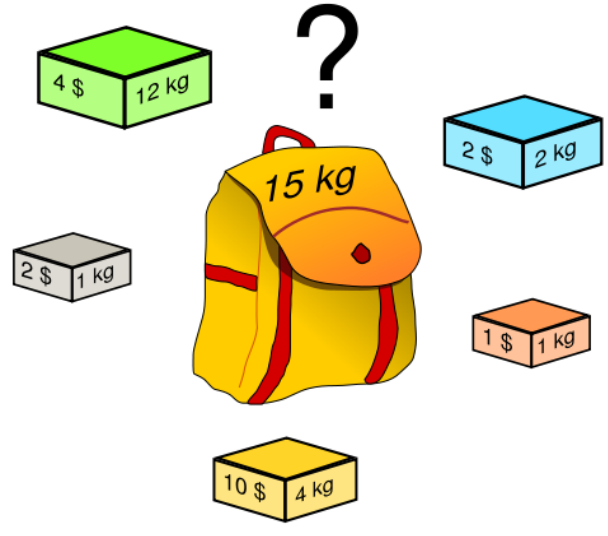
Ejemplos de problemas difíciles

- 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"

- 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"

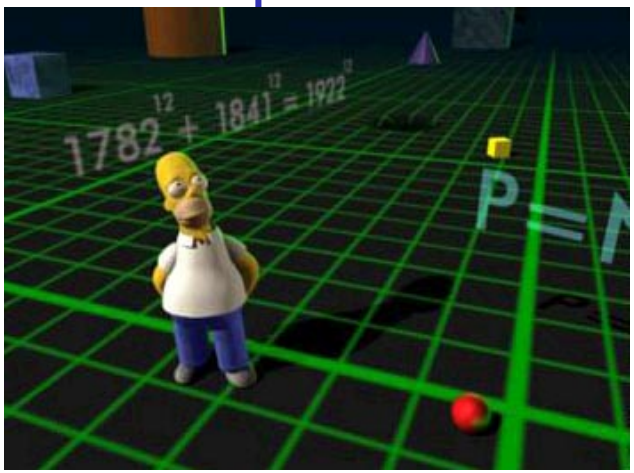
Problemas fáciles y difíciles

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

- **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.

¿Por qué **P** vs. **NP** es tan famoso en computación?

- aparece la pregunta "**P**=**NP**?" en Homero[3]



- aparece en una pared de ladrillos de la Universidad de Princeton (dice "$P = NP$?" en ASCII)

- hay un premio de un millón de dólares para el que resuelva el problema

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

Consejo práctico 1:
¿qué hacer en caso de resolver el problema de **P** vs. **NP**?

1. escribir la solución en un papel
2. pasarla a computadora
3. revisarla bien
4. mandarla por email a santiago@dc.uba.ar
5. no comentarle el descubrimiento a nadie más
6. esperar mis instrucciones

Problemas imposibles para una computadora

Los problemas de **NP** son difíciles porque toman mucho tiempo, pero una computadora puede resolverlos (quizá en miles de años).

Hay problemas más difíciles que los de **NP**.

¡Incluso hay problemas que ninguna computadora puede resolver!

El más famoso se llama problema de la parada

o problema de la detención o halting problem, en inglés.

A problem has been detected and windows has been shut down to prevent damage
to your computer.

The problem seems to be caused by the following file: SPCMDCON.SYS

PAGE_FAULT_IN_NONPAGED_AREA

If this is the first time you've seen this Stop error screen,
restart your computer. If this screen appears again, follow
these steps:

Check to make sure any new hardware or software is properly installed.
If this is a new installation, ask your hardware or software manufacturer
for any windows updates you might need.

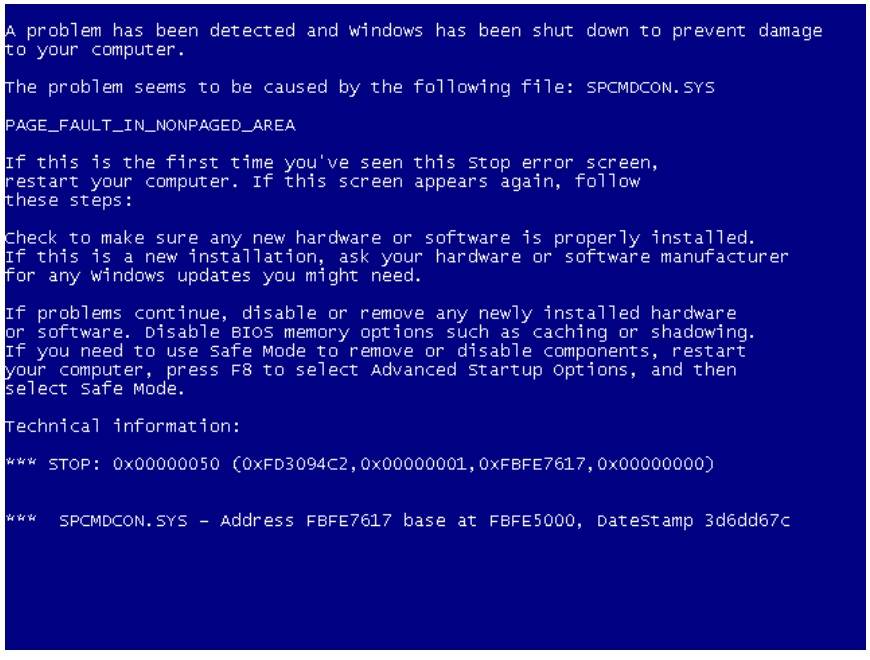
If problems continue, disable or remove any newly installed hardware
or software. Disable BIOS memory options such as caching or shadowing.
If you need to use Safe Mode to remove or disable components, restart
your computer, press F8 to select Advanced Startup Options, and then
select Safe Mode.

Technical information:

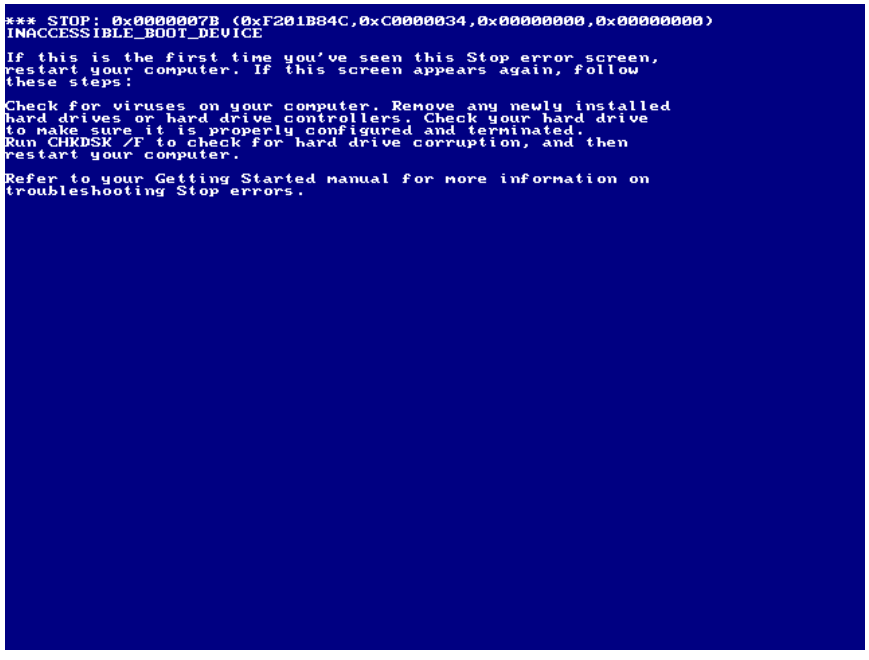
*** STOP: 0x00000050 (0xFD3094C2,0x00000001,0xFBFE7617,0x00000000)

*** SPCMDCON.SYS - Address FBFE7617 base at FBFE5000, DateStamp 3d6dd67c

Algunos programas se cuelgan

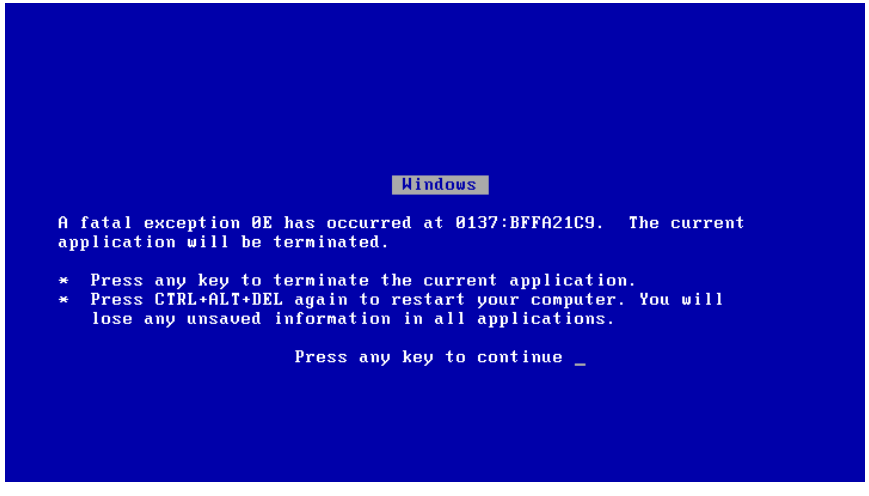


Windows XP



Windows 2000

Windows NT



Windows 9x

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?

No, no existe tal programa. Se dice que 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 con premio

- Eternity II (www.eternityii.com). Demo para 16 piezas:

- reglas:
  - bordes adyacentes tienen el mismo dibujo
  - se pueden rotar
- el verdadero tiene 256 piezas
  - buena noticia: hay 2 millones de dólares para el primero que encuentre una solución
  - mala noticia: la solución no aparece dibujada en la tapa

Consejo práctico 2:
¿qué hacer en caso de resolver el Eternity II?

1. escribir la solución en un papel
2. pasarla a computadora
3. revisarla bien
4. mandarla por email a santiago@dc.uba.ar
5. no comentarle el descubrimiento a nadie más
6. esperar mis instrucciones

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

Conclusiones

- Hilbert usaba sombrero (pero a veces se lo sacaba)
- si quieren un millón de dólares, resuelvan **P** vs. **NP**
  - pidan el premio a Clay Mathematics Institute
- si quieren dos millones de dólares, resuelvan el Eternity II
  - pidan el premio a los que hicieron ese rompecabezas
- si quieren aprender computación, estudien acá, en Exactas
  - pidan el título en el departamento de alumnos (Pab. II)