

UNIVERSIDAD NACIONAL DE INGENIERÍA

Teoría de la Computación

Jerarquías, azar y lo que sigue abierto

Teoremas de jerarquía, BPP y el mapa completo del curso

Gerson Garrido

Semana 15

Hoy

Sesión A

1. Lo que la cadena de clases no dice
2. La diagonal, otra vez
3. Teorema de jerarquía de espacio
4. Teorema de jerarquía de tiempo
5. Lo que se sabe distinto
6. Oráculos (opcional)

Sesión B

7. Máquinas que lanzan monedas
8. Freivalds: verificar sin multiplicar
9. Repetir para bajar el error
10. BPP e identidad de polinomios
11. Lo que sigue abierto
12. El mapa completo del curso

Lectura

Apuntes de la semana 15. Apoyo: Sipser, sec. 9.1, 9.2 y 10.2; Barak, cap. 19 y 20. Código en `code/week-15`.

SECCIÓN 01

Lo que la cadena no dice

01

La cadena de la semana 14

$$\mathsf{L} \subseteq \mathsf{NL} \subseteq \mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PSPACE} \subseteq \mathsf{EXPTIME}$$

Inclusión	**Por qué vale**
$\mathsf{L} \subseteq \mathsf{NL}$, $\mathsf{P} \subseteq \mathsf{NP}$	Una máquina determinista es una no determinista que nunca elige
$\mathsf{NL} \subseteq \mathsf{P}$	Hay una cantidad polinomial de configuraciones: alcanzabilidad en ese grafo
$\mathsf{NP} \subseteq \mathsf{PSPACE}$	Se prueban los certificados uno por uno, reutilizando el espacio
$\mathsf{PSPACE} \subseteq \mathsf{EXPTIME}$	Con espacio $f(n)$ hay a lo sumo $2^{O(f(n))}$ configuraciones

- Las cinco se demuestran igual: con una **simulación**.
- Ninguna dice que la clase de la derecha tenga algo que la de la izquierda no tiene.
- Con lo visto hasta hoy, las seis clases podrían ser la misma.

Una inclusión y una separación piden cosas distintas

Inclusión $\mathcal{C}_1 \subseteq \mathcal{C}_2$

- Una máquina imita a otra.
- Es una construcción.

Separación $\mathcal{C}_1 \neq \mathcal{C}_2$

- Un lenguaje concreto de $\mathcal{C}_2$.
- Y un argumento de que **ninguna** máquina de $\mathcal{C}_1$ lo decide.
- Habla de todas las máquinas, también de las que nadie ha escrito.

Idea clave

La única técnica del curso que da un argumento sobre todas las máquinas a la vez es la **diagonalización**. Hoy vuelve, con un límite de recursos.

SECCIÓN 02

La diagonal, otra vez

02

La diagonal sobre una lista concreta

terminal

```
$ build/diagonal_demo
                          ε    0    1   00   01   10
M1: acepta todo          [A]   A    A    A    A    A
M2: no acepta nada        R   [R]   R    R    R    R
M3: longitud par          A    R   [R]   A    A    A
M4: termina en 1          R    R    A   [R]   A    R
M5: contiene un 0         R    A    R    A   [A]   A
M6: empieza con 1         R    R    A    R    R   [A]
D: lo contrario           R    A    A    A    R    R
```

- Entre corchetes, la diagonal: la máquina i con la cadena i.
- D cambia cada casilla de la diagonal.
- D difiere de la fila i en la columna i: no es ninguna de las seis.

El precio

Para llenar su fila, D tiene que averiguar qué responde cada máquina: tiene que **simularla**.

De la semana 9 a hoy

- Semana 9: la máquina simulada podía no detenerse. D no existía, y de ahí salía la contradicción.
- Hoy: las máquinas de la lista usan pocos recursos. Se pueden simular hasta el final, con un poco más.
- Entonces D **sí existe**. Decide un lenguaje con sus recursos, y ninguna máquina más pobre lo decide.

Tres detalles, tres marcas en el teorema

1. D tiene que saber cuánto es $f(n)$ sin gastar más que eso: funciones **construibles**.
2. Simular cuesta un poco más que lo simulado: se separa $O(f(n))$ de $o(f(n))$.
3. "Pocos recursos" es una promesa para entradas grandes: hace falta un **relleno**.

SECCIÓN 03

Teorema de jerarquía de espacio

03

Funciones construibles

Construible en espacio ***(space constructible)***

$f(n) \geq \log_2 n$, y alguna máquina, con la entrada 1^n, escribe $f(n)$ en binario usando espacio $O(f(n))$.

- Lo son todas las usuales: $\log n$, n, $n \log n$, n^2, 2^n.
- Hace falta: D empieza marcando $f(n)$ celdas, y calcular $f(n)$ no puede sacarla de su clase.

La notación o

$g(n) = o(f(n))$: para **toda** $c > 0$ hay un n_0 con $g(n) < c \cdot f(n)$ si $n \geq n_0$.

- $n = o(n \log n)$, $\log^2 n = o(n)$.
- $3n$ no es $o(n)$.

El espacio se mide como en la semana 14: cinta de entrada de solo lectura, y solo cuenta la cinta de trabajo.

El teorema

Teorema de jerarquía de espacio

Para toda función f construible en espacio existe un lenguaje A que se decide en espacio $O(f(n))$ y que **no** se decide en espacio $o(f(n))$.

- Con más espacio se decide estrictamente más.
- Basta con muy poco más: cualquier cantidad que no sea un múltiplo constante.
- A es el lenguaje de una máquina D construida con la diagonal.

El primer intento, y dónde falla

Copiando la semana 9: con la entrada $\langle M \rangle$, simular M con $\langle M \rangle$ dentro de $f(n)$ celdas y responder lo contrario.

1. **La simulación puede no caber.**
   $g(n) = o(f(n))$ solo dice que g es menor para *n grande*. De la longitud de $\langle M \rangle$ no dice nada.
2. M **podría no detenerse.**
   Una máquina puede dar vueltas para siempre dentro de tres celdas. El límite de espacio no la detiene, y D tiene que detenerse siempre.

Los dos arreglos

Un relleno, para que la entrada sea tan larga como haga falta. Un contador de pasos, para cortar los ciclos.

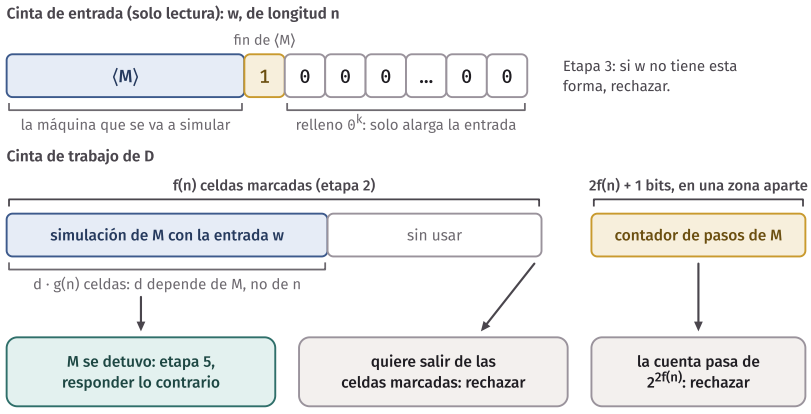
La máquina D

descripción de alto nivel

```
D = "Con la entrada w:
  1. Sea n la longitud de w.
  2. Calcular f(n) y marcar f(n) celdas de la cinta de trabajo.
     Si alguna etapa posterior intenta salir de las celdas marcadas, rechazar.
  3. Si w no tiene la forma ⟨M⟩10* para alguna máquina de Turing M, rechazar.
  4. Simular M con la entrada w, contando los pasos de M.
     Si la cuenta pasa de 2^(2·f(n)), rechazar.
  5. Si M aceptó, rechazar. Si M rechazó, aceptar."
```

- Etapa 2: se puede, porque f es construible.
- Etapa 4: si M usa $g(n)$ celdas, D usa $d \cdot g(n)$. La constante d depende de M, no de n.
- El contador ocupa $2f(n) + 1$ bits: también cabe en espacio $O(f(n))$.

D, en un dibujo: dos cintas y dos guardas



Si M decide en espacio o(f(n)) y n es grande, d · g(n) < f(n) y M se detiene antes del tope:
la simulación termina, y D responde con w lo contrario que M.

El relleno: entradas tan largas como haga falta

La entrada es $\langle M \rangle 10^k$: el 1 marca dónde termina la descripción, y los ceros solo la alargan.

- D simula a M con la entrada w **completa**, con relleno y todo.
- Ejemplo: $f(n) = n$, $g(n) = 100\sqrt{n}$, $d = 3$. La simulación cabe si $300\sqrt{n} < n$, es decir, si $n > 90\,000$.
- Con $|\langle M \rangle| = 500$ no cabe: harían falta ≈ 6708 celdas y hay 500. D rechaza sin saber qué respondería M.
- Con $\langle M \rangle 10^{90\,000}$ sí cabe. Una sola entrada en la que difieran alcanza.

El contador: por qué $2^{2f(n)}$

Configuraciones de M con $g(n)$ celdas de trabajo:

$$\underbrace{|Q|}_{\text{estado}} \cdot \underbrace{n}_{\text{cabezal de entrada}} \cdot \underbrace{g(n)}_{\text{cabezal de trabajo}} \cdot \underbrace{|\Gamma|^{g(n)}}_{\text{contenido}}$$

- Una máquina determinista que se detiene no repite ninguna configuración.
- $n \leq 2^{f(n)}$, porque $f(n) \geq \log_2 n$. El resto es $\leq 2^{c \cdot g(n)} \leq 2^{f(n)}$ para n grande.
- Total: a lo sumo $2^{2f(n)}$ pasos, para n grande.

Corta a las que no se detienen

La etapa 4 termina siempre: D es un decididor.

No corta a las que importan

Un decididor de espacio $o(f(n))$ termina antes del tope.

La demostración

Sea $A = L(D)$.

1. D **decide** A **en espacio** $O(f(n))$**.** Trabaja en las celdas marcadas más el contador, y se detiene siempre.
2. **Ninguna** M **decide** A **en espacio** $g(n) = o(f(n))$**.** Si la hubiera, existe n_0 tal que para $n \geq n_0$:

$$d \cdot g(n) < f(n) \qquad \text{y} \qquad M \text{ se detiene en a lo sumo } 2^{2f(n)} \text{ pasos.}$$

3. Con $w = \langle M \rangle 10^{n_0}$**,** la simulación cabe y termina. D responde lo contrario:

$$D \text{ acepta } w \iff M \text{ rechaza } w.$$

4. Pero M decide $L(D)$: M acepta $w \iff D$ acepta w. **Contradicción.**

La semana 9 y hoy, pieza por pieza

	Semana 9: A_{TM} es indecidible	**Hoy: jerarquía de espacio**
La lista	Todas las máquinas de Turing	Las máquinas de espacio $o(f(n))$
La diagonal	M con $\langle M \rangle$	M con $\langle M \rangle 10^k$, con k grande
Cómo averigua D	Con un decididor H supuesto	Simulando, con límite de celdas y contador
¿D existe?	No: de ahí la contradicción	Sí: es una máquina concreta
Conclusión	Ninguna máquina decide A_{TM}	$L(D)$ está en una clase y no en la otra

Lo que se agrega

La función construible fija el límite, el relleno hace la entrada lo bastante grande, el contador hace que D se detenga.

Corolario: una escalera infinita

Corolario

Si $f_1(n) = o(f_2(n))$ y f_2 es construible en espacio, entonces $\mathrm{SPACE}(f_1(n)) \subsetneq \mathrm{SPACE}(f_2(n))$.

$$\mathrm{SPACE}(\log n) \subsetneq \mathrm{SPACE}(\log^2 n) \subsetneq \mathrm{SPACE}(n) \subsetneq \mathrm{SPACE}(n^2) \subsetneq \mathrm{SPACE}(n^3) \subsetneq \cdots$$

- Dentro de PSPACE hay infinitos escalones, y cada uno tiene lenguajes que el anterior no tiene.
- El eslabón $\mathrm{SPACE}(\log^2 n) \subsetneq \mathrm{SPACE}(n)$ vuelve en diez minutos.

SECCIÓN 04

Teorema de jerarquía de tiempo

04

Tiempo: el mismo teorema, con un logaritmo

Teorema de jerarquía de tiempo

Para toda función t construible en tiempo existe un lenguaje A que se decide en tiempo $O(t(n))$ y que **no** se decide en tiempo $o(t(n)/\log t(n))$.

- Casi el de espacio, con un factor $\log t(n)$ de diferencia.
- t es construible en tiempo si $t(n) \geq n \log n$ y se calcula en tiempo $O(t(n))$.
- La máquina D es la misma, con una sola guarda: el contador de pasos **es** el límite. Empieza en $t(n)/\log t(n)$ y baja en cada paso de la simulación.
- Solo vemos la idea: de dónde sale el logaritmo.

De dónde sale el factor $\log t(n)$

- En espacio, el contador ocupa celdas, y las celdas se reutilizan: llevarlo es gratis.
- En tiempo, cada actualización del contador cuesta $O(\log t(n))$ pasos.
- Y en una cinta hay que llevarlo junto al cabezal, en una pista paralela: ir a buscarlo costaría mucho más.

$$\frac{t(n)}{\log t(n)} \text{ pasos de simulación } \times\ O(\log t(n)) \text{ por paso } =\ O(t(n))$$

- D corre en tiempo $O(t(n))$ y solo alcanza a contradecir a las máquinas de tiempo $o(t(n)/\log t(n))$.

Idea clave

Es el costo del reloj en una máquina de una cinta. No dice que haga falta tanta ventaja para decidir más.

Lo que el teorema dice, y lo que no

$$\mathrm{TIME}(n^2) \subsetneq \mathrm{TIME}(n^3) \subsetneq \mathrm{TIME}(n^4) \subsetneq \cdots \qquad \text{porque } n^k = o\left(n^{k+1} / \log n^{k+1}\right)$$

- También dentro de P hay una escalera infinita.

Error común

Leerlo como "con más tiempo todo problema se resuelve mejor". Dice que **existe algún** lenguaje al que más tiempo le sirve: el de la diagonal.

- $\{0^k 1^k\}$ se decide en $O(n \log n)$. Darle n^5 no cambia nada.
- Que SAT se decida en tiempo 2^n no dice que necesite tanto.
- Los teoremas separan **clases**. Cuánto cuesta un problema dado es otra pregunta.

SECCIÓN 05

Lo que se sabe distinto

05

$\mathsf{P} \neq \mathsf{EXPTIME}$

1. 2^n es construible en tiempo. Por el teorema hay un lenguaje A:

$$A \text{ se decide en tiempo } O(2^n), \qquad A \text{ no se decide en tiempo } o(2^n/n).$$

2. $A \in \mathsf{EXPTIME}$, porque $\mathsf{TIME}(2^n) \subseteq \mathsf{EXPTIME}$.
3. $A \notin \mathsf{P}$: si se decidiera en tiempo $O(n^k)$, como $n^k = o(2^n/n)$ para todo k, se decidiría en tiempo $o(2^n/n)$.

Consecuencia

En $\mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PSPACE} \subseteq \mathsf{EXPTIME}$ los extremos son distintos: **al menos una** de las tres inclusiones es estricta. No se sabe cuál.

NL $\neq$ PSPACE, **y entonces** L $\neq$ PSPACE

$$\mathsf{NL} = \mathsf{NSPACE}(\log n) \underbrace{\subseteq}_{\text{Savitch}} \mathsf{SPACE}(\log^2 n) \underbrace{\subsetneq}_{\text{jerarquía}} \mathsf{SPACE}(n) \subseteq \mathsf{PSPACE}$$

- Savitch: $\mathsf{NSPACE}(f(n)) \subseteq \mathsf{SPACE}(f(n)^2)$, con $f(n) = \log n$.
- Jerarquía: $\log^2 n = o(n)$, y n es construible.
- Como L $\subseteq$ NL, tampoco L es todo PSPACE.

Por qué hace falta Savitch

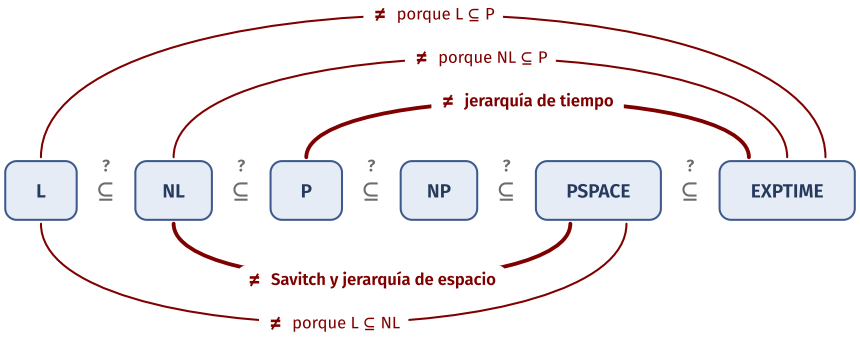
El teorema de jerarquía compara clases deterministas. Para decir algo de NL, primero hay que meterla en una clase determinista.

La cadena, par por par

	NL	P	NP	PSPACE	EXPTIME
L	?	?	?	$\neq$	$\neq$
NL		?	?	$\neq$	$\neq$
P			?	?	$\neq$
NP				?	?
PSPACE					?

- 15 pares. Se sabe que 5 son distintos.
- NL y L frente a PSPACE: Savitch y jerarquía de espacio.
- P frente a EXPTIME: jerarquía de tiempo.
- L y NL frente a EXPTIME: porque $\mathrm{NL} \subseteq \mathrm{P}$.
- De los otros diez no se sabe nada. Tampoco de NP frente a EXPTIME.

Los cinco pares, sobre la cadena



De los otros diez pares no se sabe nada. Entre P y EXPTIME, al menos una de las tres inclusiones es estricta.

Un problema que no está en P, por teorema

$$A_{\mathrm{TM}}^{\mathrm{reloj}} = \{\, \langle M, w, k\rangle : M \text{ acepta } w \text{ en a lo sumo } k \text{ pasos} \,\}, \qquad k \text{ en } \textbf{binario}$$

- Decidible: se simula k pasos. Pero k puede valer 2^n: tiempo exponencial.
- Todo $B \in$ EXPTIME se reduce a él: si M_B decide B en 2^{n^c+c} pasos,

$$f(w) = \langle M_B,\ w,\ 2^{|w|^c+c}\rangle, \qquad \text{y el número se escribe con } |w|^c + c + 1 \text{ bits.}$$

- Si estuviera en P, tendríamos EXPTIME $\subseteq$ P. Contradicción.

Sin “se cree”

Es EXPTIME-completo, como el ajedrez y las damas generalizados a un tablero de $n \times n$. Ninguno tiene algoritmo polinomial, y eso es un teorema.

SECCIÓN 06

Oráculos (opcional)

06



Máquinas con oráculo

Oráculo *(oracle)*

Responde en un paso si una cadena está en A. La máquina M^A escribe la consulta y recibe sí o no.

- La respuesta es gratis, sea A fácil o indecidible.

Clases relativas a A

- P^A: tiempo polinomial determinista, con oráculo A.
- NP^A: lo mismo, no determinista.
- $\mathrm{P}^A \subseteq \mathrm{NP}^A$ siempre.

La pregunta

La diagonal separó P de EXPTIME. ¿Por qué no separa P de NP?

Dos oráculos que responden distinto

Con $A = \mathrm{TQBF}$: $\mathsf{P}^A = \mathsf{NP}^A$

$$\mathsf{NP}^{\mathrm{TQBF}} \subseteq \mathsf{NPSPACE} = \mathsf{PSPACE} \subseteq \mathsf{P}^{\mathrm{TQBF}}$$

- Las consultas se responden en espacio polinomial; Savitch; y TQBF es PSPACE-completo: reducir y hacer una consulta.

Existe B con $\mathsf{P}^B \neq \mathsf{NP}^B$

$$L_B = \{\, 1^n : B \text{ tiene alguna cadena de longitud } n \,\}$$

- $L_B \in \mathsf{NP}^B$ para todo B: adivinar la cadena y consultar.
- B se arma por etapas, con una diagonal. La máquina i hace menos de 2^n consultas, y B esconde una cadena donde no miró, o no esconde ninguna, según lo que ella responda.

Qué significa: la relativización

- En el teorema de jerarquía, D trata a M como una caja: la simula y mira la respuesta.
- Con el mismo oráculo para todas, la demostración no cambia: D^A reenvía las consultas de M^A. La demostración **relativiza** *(relativization)*.
- Si una demostración así diera $\mathrm{P} \neq \mathrm{NP}$, daría $\mathrm{P}^A \neq \mathrm{NP}^A$ para todo A. Falso con TQBF.
- Si diera $\mathrm{P} = \mathrm{NP}$, daría $\mathrm{P}^B = \mathrm{NP}^B$ para todo B. Falso con el B construido.

Conclusión, leída con cuidado

Ninguna demostración que relativice resuelve P frente a NP. No dice que no se pueda resolver: dice que simular y diagonalizar no alcanza, y que hace falta mirar **dentro** del cómputo.

SECCIÓN 07

Máquinas que lanzan monedas

07

Una tercera manera de avanzar

Máquina de Turing probabilística

Una máquina no determinista con una o dos opciones por paso. Cuando hay dos, se elige con una moneda justa. Una rama con m monedas tiene probabilidad 2^{-m}.

Máquina	**Acepta** w **si…**
No determinista	**alguna** rama acepta
Probabilística	**la mayor parte** de las ramas acepta, medida por su probabilidad

- Como programa: un algoritmo determinista $M(w, r)$, con r una cadena de bits al azar.
- En NP se pregunta si *existe* un certificado que haga aceptar. Aquí, qué *fracción* de las r hace aceptar.

La probabilidad que hace falta

Solo lo elemental

- Si hay N resultados igualmente probables y un evento ocurre en K de ellos, su probabilidad es K/N.
- Si dos eventos son **independientes**, la probabilidad de que ocurran los dos es el producto.
- Dos ejecuciones de un algoritmo con monedas distintas son independientes.

Error común

Confundir el azar del algoritmo con el azar de la entrada. La entrada no es al azar: es la peor. “Se equivoca con probabilidad a lo sumo $1/2$” vale para **toda** entrada, y la probabilidad es sobre las monedas.

SECCIÓN 08

Freivalds: verificar sin multiplicar

08

¿$A \cdot B = C$?

- Tres matrices de $n \times n$. La manera obvia: multiplicar y comparar. Son n^3 multiplicaciones.
- Los mejores algoritmos conocidos bajan el exponente a poco menos de 2,38, y no se usan. Nadie sabe multiplicar en $O(n^2)$.
- ¿Se puede **comprobar** un producto más rápido de lo que se sabe **calcularlo**?

La idea de Freivalds

No comparar las matrices: comparar lo que le hacen a un vector. Si $A \cdot B = C$, entonces $A \cdot (B \cdot r) = C \cdot r$ para todo r. Y matriz por vector cuesta n^2.

El algoritmo

pseudocódigo

```
algoritmo freivalds(A, B, C)
    r ← vector de n bits al azar
    x ← B · r
    y ← A · x
    z ← C · r
    si y = z entonces
        devolver "iguales"
    sino
        devolver "distintos"
    fin si
fin algoritmo
```

- Tres productos de matriz por vector: $3n^2$.
- Nunca se calcula $A \cdot B$.
- Si $A \cdot B = C$: responde “iguales” **siempre**.
- Si $A \cdot B \neq C$: puede fallar, si r no nota la diferencia.

Error de un solo lado

“Distintos” viene con una prueba: el vector r. Esa respuesta es segura.

Dos rondas, paso a paso

Ronda 2

$$B = \begin{bmatrix} 2 & 1 & 0 \\ 1 & 0 & 2 \\ 0 & 3 & 1 \end{bmatrix} \quad r = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} \quad x = B \cdot r = \begin{bmatrix} 1 \\ 2 \\ 4 \end{bmatrix}$$

$$A = \begin{bmatrix} 1 & 2 & 0 \\ 0 & 1 & 3 \\ 2 & 0 & 1 \end{bmatrix} \quad x = \begin{bmatrix} 1 \\ 2 \\ 4 \end{bmatrix} \quad y = A \cdot x = \begin{bmatrix} 5 \\ 14 \\ 6 \end{bmatrix}$$

C_1 es A·B salvo la entrada (2, 3): vale 6 en vez de 5

$$C_1 = \begin{bmatrix} 4 & 1 & 4 \\ 1 & 9 & 6 \\ 4 & 5 & 1 \end{bmatrix} \quad r = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} \quad z = C_1 \cdot r = \begin{bmatrix} 5 \\ 15 \\ 6 \end{bmatrix}$$

$$y = \begin{bmatrix} 5 \\ 14 \\ 6 \end{bmatrix} \quad z = \begin{bmatrix} 5 \\ 15 \\ 6 \end{bmatrix} \quad y \neq z$$

«distintos» seguro

y ≠ z en la segunda coordenada: la ronda responde «distintos», y el algoritmo termina.

Esa respuesta es segura: r es una prueba de que $A \cdot B \neq C_1$.

Con $n = 3$ se pueden probar los 8 vectores

build/freivalds_demo, primera parte

```
A·B =   4   1   4
        1   9   5
        4   5   1

C1: igual a A·B salvo la entrada (2, 3), que vale 6 en vez de 5.
  r           B·r           A·(B·r)         C·r             ¿detecta?
  (0, 0, 0)   (0, 0, 0)     (0, 0, 0)       (0, 0, 0)       no
  (0, 0, 1)   (0, 2, 1)     (4, 5, 1)       (4, 6, 1)       sí
  (0, 1, 0)   (1, 0, 3)     (1, 9, 5)       (1, 9, 5)       no
  (0, 1, 1)   (1, 2, 4)     (5, 14, 6)      (5, 15, 6)      sí
  (1, 0, 0)   (2, 1, 0)     (4, 1, 4)       (4, 1, 4)       no
  (1, 0, 1)   (2, 3, 1)     (8, 6, 5)       (8, 7, 5)       sí
  (1, 1, 0)   (3, 1, 3)     (5, 10, 9)      (5, 10, 9)      no
  (1, 1, 1)   (3, 3, 4)     (9, 15, 10)     (9, 16, 10)     sí
  Detectan la diferencia 4 de 8 vectores.
```

- Matriz por vector de ceros y unos: **sumar las columnas elegidas**.
- El error está en la columna 3. Se nota si y solo si $r_3 = 1$.
- Otros dos casos del programa: detectan 4 de 8 y 7 de 8.

Por qué al menos la mitad (1)

Teorema

Si $A \cdot B \neq C$, al menos la mitad de los 2^n vectores r de ceros y unos cumple $A \cdot (B \cdot r) \neq C \cdot r$. Freivalds se equivoca con probabilidad a lo sumo $1/2$.

- Sea $D = A \cdot B - C$. Entonces $A \cdot (B \cdot r) - C \cdot r = D \cdot r$.
- r detecta la diferencia exactamente cuando $D \cdot r \neq 0$.
- D no es nula: tiene alguna entrada $d_{ij} \neq 0$. Miramos solo la coordenada i:

$$(D \cdot r)_i \;=\; d_{ij}\, r_j + s, \qquad s = \sum_{l \neq j} d_{il}\, r_l \quad \text{(no depende de } r_j\text{)}$$

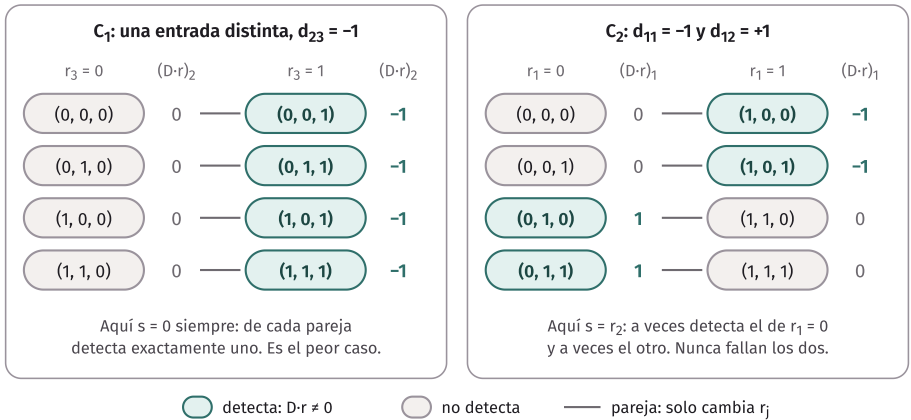
Por qué al menos la mitad (2): las parejas

- Se empareja cada vector con el que solo difiere en la coordenada j: 2^{n-1} parejas.
- Los dos de una pareja tienen el mismo s. Sus coordenadas i valen s y $s + d_{ij}$.
- Como $d_{ij} \neq 0$, **no pueden ser cero los dos**.
- Cada pareja aporta al menos un vector que detecta: 2^{n-1} de 2^n.

El peor caso

Con una sola entrada distinta, detecta exactamente uno por pareja: falla con probabilidad $1/2$ exacta.

Las parejas, con los ocho vectores del ejemplo



En una pareja, la coordenada i de D·r vale s y s + d_{ij}. Como $d_{ij} \neq 0$, no pueden ser cero los dos.

Módulo un primo, y el overflow

- Con entradas hasta 10^9 y $n = 2000$, una entrada de $A \cdot (B \cdot r)$ llega a $4 \cdot 10^{24}$: no cabe en 64 bits.
- Se calcula todo módulo $p = 10^9 + 7$. La pregunta pasa a ser si $A \cdot B = C$ módulo p.
- El teorema vale igual: si s y $s + d_{ij}$ fueran los dos múltiplos de p, su diferencia d_{ij} lo sería.
- El argumento de las parejas no usa que p sea primo. La identidad de polinomios, más adelante, sí.

En C++

Dos restos menores que 2^{30} se multiplican en 64 bits: `std::uint64_t{a} * b % p`. En 32 bits el producto no cabe, y `a * b` entre dos `std::uint32_t` ya llega dañado.

SECCIÓN 09

Repetir para bajar el error

09

k rondas: el error baja a 2^{-k}

terminal

```
$ build/freivalds_experiment
n = 30, p = 1000000007, semilla = 2015, 1000 ejecuciones por fila
C difiere de A·B solo en la entrada (18, 5).

 k   fallos   fracción   cota 2^-k
 1      508      0.508    0.500000
 2      245      0.245    0.250000
 5       34      0.034    0.031250
10        1      0.001    0.000977

Con C = A·B y k = 10: 0 respuestas "distintos" en 1000 ejecuciones.
```

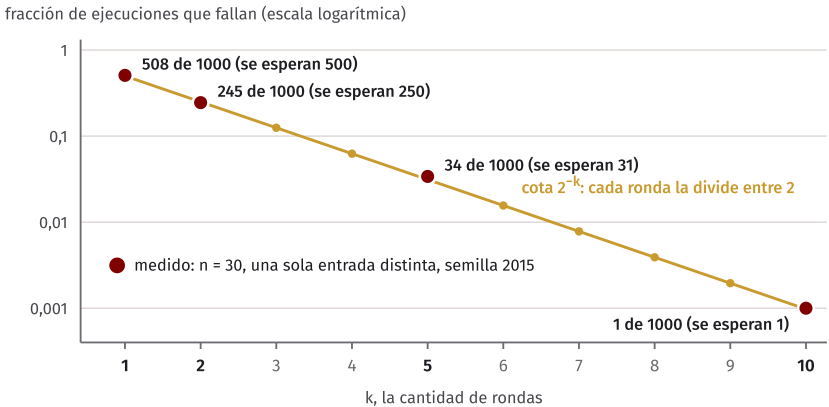
- "Iguales" solo si ninguna ronda ve una diferencia.
- Falla si fallan las k rondas. Son independientes:

$$\frac{1}{2} \cdot \frac{1}{2} \cdots \frac{1}{2} = 2^{-k}$$

- Se esperaban cerca de 500, 250, 31 y 1 fallos.
- Si $A \cdot B = C$, no se equivoca nunca.

Semilla fija: la tabla sale igual cada vez. Con `build/freivalds_experiment 7` salen otros números igual de cercanos.

El experimento, en escala logarítmica



¿Cuántas rondas?

k	**cota del error**
10	$\approx 10^{-3}$
20	$\approx 9{,}5 \cdot 10^{-7}$
30	$\approx 9{,}3 \cdot 10^{-10}$
100	$\approx 7{,}9 \cdot 10^{-31}$

- Para un error menor que 10^{-9}: $k = 30$, porque $2^{30} > 10^9 > 2^{29}$.
- Con 100 rondas, el error es menos probable que un fallo del hardware durante el cálculo.
- El costo: $O(k \cdot n^2)$.

Error común

Creer que un algoritmo probabilístico "a veces se equivoca y no hay nada que hacer". El error es un número que se **elige**: quien corre el algoritmo fija k.

SECCIÓN 10

BPP e identidad de polinomios

10

La clase BPP

BPP ***(bounded-error probabilistic polynomial time)***

$A \in \text{BPP}$ si hay una máquina probabilística M de tiempo polinomial tal que, para toda w:

$$w \in A \implies \Pr[M \text{ acepta } w] \geq 2/3, \qquad w \notin A \implies \Pr[M \text{ rechaza } w] \geq 2/3.$$

- Error de los dos lados, a lo sumo $1/3$ en cada caso.
- Freivalds con dos rondas cabe: acepta con probabilidad 1, o rechaza con probabilidad $\geq 3/4$.
- Pero ese lenguaje ya estaba en P. Ahí el azar **acelera**, de n^3 a n^2.
- ¿Hay algo en BPP que no se sepa poner en P? Sí: dos diapositivas más adelante.

La constante no importa: amplificación

Lema de amplificación

Con error a lo sumo $1/3$ por ejecución, la **mayoría** de k ejecuciones independientes se equivoca con probabilidad a lo sumo $(8/9)^{k/2}$.

- Con dos lados no basta un testigo: se vota.
- Con $k = 12m$ el error baja de 2^{-m}.
- Con error $1/2$ exacto no hay nada que amplificar.

terminal

```
$ build/majority_table
  k   la mayoría se equivoca   cota (8/9)^(k/2)
  1                3.333e-01          9.428e-01
  5                2.099e-01          7.449e-01
 11                1.221e-01          5.232e-01
 25                4.151e-02          2.294e-01
 51                6.888e-03          4.961e-02
101                2.724e-04          2.611e-03
201                5.531e-07          7.231e-06
```

Probabilidad exacta con error $1/3$ por ejecución, y la cota.

Identidad de polinomios

$$¿(x-1)(x-2)\cdots(x-10) \;=\; x^{10} - 55x^9 + \cdots?$$

- Desarrollar puede ser carísimo: $(x_1 + y_1)\cdots(x_n + y_n)$ tiene 2^n términos. Evaluar en un punto es barato.
- La misma idea de Freivalds: comparar lo que valen en un punto r al azar, módulo p.

Teorema

Si p es primo y $f \neq g$ tienen grado a lo sumo d, entonces $f(r) = g(r)$ módulo p para a lo sumo d de los p valores de r. El error es a lo sumo d/p.

- $f - g$ es un polinomio no nulo de grado $\leq d$: tiene a lo sumo d raíces.
- Aquí p tiene que ser primo: módulo 6, $2x$ tiene grado 1 y dos raíces, 0 y 3.

$f(x) = (x-1)(x-2)\cdots(x-10)$, **con** $p = 101$ **y con** $p = 10^9 + 7$

build/pit_demo, las dos tablas

```
p = 101, 1000 ejecuciones por fila, cota 10/p = 9.90e-02
  f se compara con                          coinciden en    responde "coinciden"
  su desarrollo                             101 de 101      1000 de 1000
  el desarrollo con un coeficiente cambiado   1 de 101        10 de 1000
  el polinomio cero                          10 de 101       102 de 1000

p = 1000000007, 1000 ejecuciones por fila, cota 10/p = 1.00e-08
  f se compara con                          coinciden en    responde "coinciden"
  su desarrollo                             (no se cuenta)  1000 de 1000
  el desarrollo con un coeficiente cambiado (no se cuenta)     0 de 1000
  el polinomio cero                         (no se cuenta)     0 de 1000
```

- Con el polinomio cero coinciden en las 10 raíces: el máximo del teorema.
- Para bajar el error basta **agrandar** el conjunto del que sale el punto.
- Con varias variables vale lo mismo (Schwartz-Zippel): error $\leq d/p$, con d el grado total.

Dónde queda BPP

$$\mathrm{P} \subseteq \mathrm{BPP} \subseteq \mathrm{PSPACE}$$

- $\mathrm{P} \subseteq \mathrm{BPP}$: una máquina determinista no lanza monedas y no se equivoca.
- $\mathrm{BPP} \subseteq \mathrm{PSPACE}$: probar **todas** las cadenas de monedas, una por una, y contar cuántas aceptan.
- El azar no cambia lo que se puede calcular: solo el costo.
- Identidad de polinomios dados por fórmulas: en BPP, y no se sabe si en P.

Lo que se cree: $\mathrm{P} = \mathrm{BPP}$

Primalidad pasó de BPP a P en 2002. Y si existen problemas suficientemente difíciles, las monedas se pueden reemplazar. El azar ayudaría en la práctica.

SECCIÓN 11

Lo que sigue abierto

11

Cuatro preguntas sin respuesta

	Si son iguales	Si son distintas
P frente a NP	Encontrar es tan fácil como comprobar. La criptografía de clave pública de hoy cae, en principio.	No hay atajo general para los NP-completos. Pide una técnica que hoy no existe.
NP frente a coNP	Toda fórmula insatisfacible tiene una demostración corta de que lo es.	Entonces $P \neq NP$, porque P es cerrada bajo complemento.
P frente a PSPACE	Entonces $P = NP$. Los juegos se deciden en tiempo polinomial.	Afirma menos que $P \neq NP$, y tampoco se ha podido.
L frente a NL	PATH con memoria logarítmica sin adivinar. En grafos no dirigidos ya se sabe.	Adivinar el camino sería indispensable.

Lo común

En las cuatro se pregunta si un recurso agrega poder, en las cuatro se cree que sí, y en ninguna se sabe.

Más allá del curso: demostraciones interactivas

- NP: un certificado escrito, que un verificador determinista revisa.
- ¿Y si el verificador lanza monedas y **hace preguntas**, en varias rondas, y solo se le pide convencerse con alta probabilidad? Esa clase es IP.
- Resultado: IP $=$ PSPACE (Shamir, hacia 1990).
- Un verificador polinomial puede convencerse de que una fórmula **no** es satisfacible, algo sin certificado corto conocido.

Conexión con hoy

La demostración convierte fórmulas en polinomios y usa la identidad de polinomios. Y es un resultado que no relativiza.

Más allá del curso: criptografía

- Todo el curso trató la dificultad como un obstáculo. La criptografía la usa como materia prima.
- Necesita una **función de un solo sentido**: fácil de calcular, difícil de invertir en casi todas las entradas.
- Si existe alguna, entonces $\mathrm{P} \neq \mathrm{NP}$.
- No se sabe si existen. No se sabe si $\mathrm{P} \neq \mathrm{NP}$ alcanzaría: hace falta dificultad en el caso típico, no solo en el peor.
- Los sistemas reales se apoyan en problemas concretos que se creen difíciles, como factorizar.

En una línea

La seguridad de internet descansa, hoy, sobre conjeturas.

Más allá del curso: computación cuántica

- Otro modelo de máquina, con su clase de tiempo polinomial con error acotado: BQP.
- Se sabe: BPP $\subseteq$ BQP $\subseteq$ PSPACE.
- Algoritmo de Shor (1994): factorizar en tiempo polinomial cuántico. Con una computadora clásica no se sabe.

No cambia lo calculable

Una máquina de Turing la simula, con costo exponencial. La tesis de Church-Turing no se toca.

No resuelve NP, que se sepa

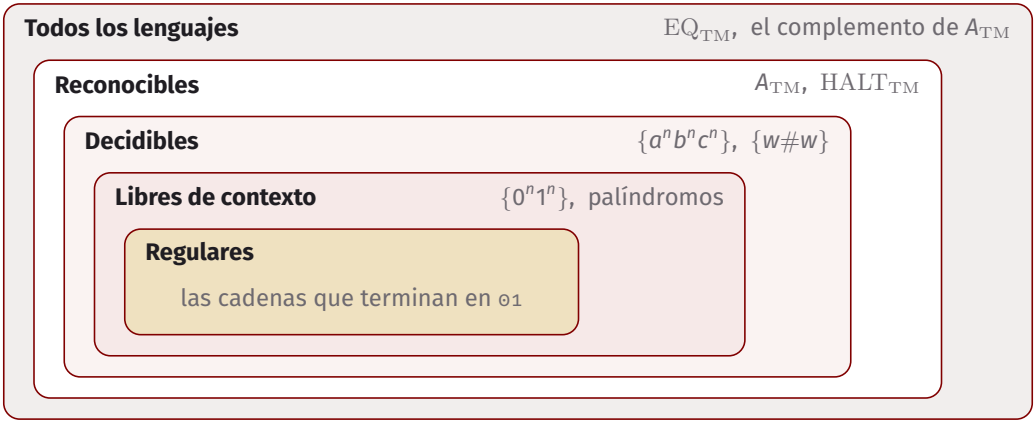
Buscar entre 2^n candidatos baja a unos $2^{n/2}$ pasos: sigue siendo exponencial.

SECCIÓN 12

El mapa completo del curso

12

De lo regular a lo no reconocible



Cada inclusión es estricta: a la derecha, un lenguaje que está en ese nivel y no en el de adentro.

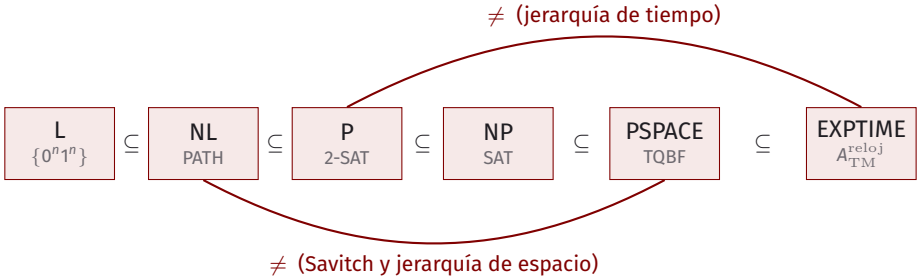
Un lenguaje que separa cada nivel, y la técnica

Nivel	Modelo	No está en el anterior	Técnica
Regulares	DFA, NFA, expresión regular		
Libres de contexto	Gramática, PDA	$\{0^n 1^n\}$	Lema de bombeo; Myhill-Nerode
Decidibles	MT que siempre se detiene	$\{a^n b^n c^n\}$	Lema de bombeo para lenguajes libres de contexto
Reconocibles	Máquina de Turing	A_{TM}	Diagonalización
Todos	Ninguno	$\overline{A_{\mathrm{TM}}}$, $\mathrm{EQ}_{\mathrm{TM}}$	Complemento reconocible; reducción por función

Para ubicar un lenguaje hacen falta dos argumentos

Pertenencia: una máquina o un algoritmo. Exclusión: bombeo, diagonalización o reducción.

Dentro de lo decidible: las clases de complejidad



- **Igualdades conocidas:** PSPACE $=$ NPSPACE, NL $=$ coNL.
- **Además:** P $\subseteq$ BPP $\subseteq$ PSPACE, y EXPTIME $\subsetneq$ decidibles.
- **Los dos mapas se cruzan:** todo lenguaje libre de contexto está en P, y $\{a^n b^n c^n\}$, que no lo es, está en L.
- **Lo que no se sabe se organiza con reducciones:** SAT en P $\Rightarrow$ P $=$ NP; TQBF en P $\Rightarrow$ P $=$ PSPACE; PATH en L $\Rightarrow$ L $=$ NL.

Las cuatro técnicas del curso

Técnica	La idea	Dónde apareció
Simulación	Una máquina imita a otra	Subconjuntos; varias cintas; máquina universal; Savitch; BPP $\subseteq$ PSPACE
Palomar	Con memoria finita algo se repite, y se puede bombear	Los dos lemas de bombeo; Myhill-Nerode; contar configuraciones
Diagonalización	Hacer lo contrario que cada máquina de una lista	Más lenguajes que máquinas; A_{TM}; teoremas de jerarquía
Reducción	Si A se resuelve con B, entonces B es al menos tan difícil	Indecidibilidad; NP, PSPACE y EXPTIME-completitud

La dirección, una vez más

Para demostrar que B es difícil se reduce **desde** un problema que ya se sabe difícil **hacia** B.

Laboratorio: verificar un producto de matrices al azar

1. Leer A, B y C de $n \times n$ y responder `iguales` o `distintos`, con Freivalds repetido $k = 30$ veces.
2. Aritmética módulo $p = 10^9 + 7$: los productos, en 64 bits.
3. Pasar los casos grandes, con $n = 2000$: el producto en $O(n^3)$ da TLE.
4. Con una C que difiere en una sola entrada, medir en 1000 ejecuciones cuántas veces falla $k = 1$. Comparar con $1/2$.
5. Repetir con $k = 2$, 5 y 10, y anotar la tabla.

Dónde: `code/week-15/lab/freivalds`. La lectura ya viene en el esqueleto. Para los pasos 4 y 5, `experiment.cpp`.

casos/01.in

```
2
1 2
3 4
5 6
7 8
19 22
43 50
```

casos/01.out

```
iguales
```

Para la próxima clase

- Esta fue la última clase. La próxima semana es el **examen final**: todo el curso, con énfasis en las semanas 9 a 15.
- Rehacer las preguntas de repaso de las guías 9 a 15, y las prácticas 3 y 4 sin mirar la solución.
- Dibujar de memoria el mapa: un lenguaje que separe cada nivel y la técnica que lo demuestra.
- Para cada reducción de los apuntes: de qué problema a cuál va, y por qué en esa dirección.

Pregunta

De los 15 pares de clases de la cadena $L \subseteq NL \subseteq P \subseteq NP \subseteq PSPACE \subseteq EXPTIME$, ¿cuáles se sabe que son distintos, y con qué teorema?