

# Demostraciones

Santiago Cifuentes y Julián Braier

Departamento de Computación  
Facultad de Ciencias Exactas y Naturales  
Universidad de Buenos Aires

2<sup>do</sup> Cuatrimestre de 2024

# Programa de hoy

- 1 Qué es una demostración?
- 2 Técnicas
- 3 Ejemplos polémicos
- 4 Ejemplos con backtracking
- 5 Ejemplos con algoritmos golosos

# Demostración

## Definición

Una demostración es un argumento deductivo. Una demostración de una fórmula  $\phi$  es una derivación de  $\phi$  que se obtiene partiendo de axiomas y aplicando reglas de derivación.

# Demostración

Esta idea de construir un argumento paso a paso partiendo de algunas verdades “evidentes” es vieja (ver *Los Elementos* de Euclides), aunque se formalizó fuertemente en el 1920, obteniendo los **Sistemas formales**.

Name	Axiom Schema	Description
THEN-1	$\phi \rightarrow (\chi \rightarrow \phi)$	Add hypothesis $\chi$ , implication introduction
THEN-2	$(\phi \rightarrow (\chi \rightarrow \psi)) \rightarrow ((\phi \rightarrow \chi) \rightarrow (\phi \rightarrow \psi))$	Distribute hypothesis $\phi$ over implication
AND-1	$\phi \wedge \chi \rightarrow \phi$	Eliminate conjunction
AND-2	$\phi \wedge \chi \rightarrow \chi$	
AND-3	$\phi \rightarrow (\chi \rightarrow (\phi \wedge \chi))$	Introduce conjunction
OR-1	$\phi \rightarrow \phi \vee \chi$	Introduce disjunction
OR-2	$\chi \rightarrow \phi \vee \chi$	
OR-3	$(\phi \rightarrow \psi) \rightarrow ((\chi \rightarrow \psi) \rightarrow (\phi \vee \chi \rightarrow \psi))$	Eliminate disjunction
NOT-1	$(\phi \rightarrow \chi) \rightarrow ((\phi \rightarrow \neg \chi) \rightarrow \neg \phi)$	Introduce negation
NOT-2	$\phi \rightarrow (\neg \phi \rightarrow \chi)$	Eliminate negation
NOT-3	$\phi \vee \neg \phi$	Excluded middle, classical logic
IFF-1	$(\phi \leftrightarrow \chi) \rightarrow (\phi \rightarrow \chi)$	Eliminate equivalence
IFF-2	$(\phi \leftrightarrow \chi) \rightarrow (\chi \rightarrow \phi)$	
IFF-3	$(\phi \rightarrow \chi) \rightarrow ((\chi \rightarrow \phi) \rightarrow (\phi \leftrightarrow \chi))$	Introduce equivalence

# Demostración

Estos sistemas permiten describir demostración de una forma **precisa**, tanto que se pueden revisar **mecánicamente**. No obstante, son muy engorrosos tanto de escribir como de leer para los seres humanos (igual que lo que pasa con el código), por lo que es común emplear argumentos *informales* (como un buen pseudocódigo, o una descripción alto nivel de los algoritmos).

# Demostración

Estos sistemas permiten describir demostración de una forma **precisa**, tanto que se pueden revisar **mecánicamente**. No obstante, son muy engorrosos tanto de escribir como de leer para los seres humanos (igual que lo que pasa con el código), por lo que es común emplear argumentos *informales* (como un buen pseudocódigo, o una descripción alto nivel de los algoritmos).

En Algo3 nos manejamos con los argumentos de este segundo tipo (y también lo hace el resto del planeta) pero aún así el estudiante dedicado debería ser capaz de traducir cualquier argumento informal en uno formal, dada una cantidad de tiempo razonable.

# Demostración

El poder que nos da el lenguaje informal (escribir menos) viene con una desventaja: es factible equivocarnos.

- El algoritmo para *Matrix Chain Multiplication* en  $O(n \log n)$  publicado en 1981 era correcto, pero tenía un error en la demo (que fue arreglado al poco tiempo).
- Otro algoritmo para el problema *MCM* con la misma complejidad fue publicado en 2013, pero este directamente era incorrecto (aunque la publicación incluía una demo)<sup>1</sup>.

Notemos que una demostración incorrecta no implica que algo sea falso (como pasó con el primer algoritmo de *MCM*).

---

<sup>1</sup>Leer *Revisiting Computation of Matrix Chain Products*" <https://epubs.siam.org/doi/10.1137/18M1195401>

# Demostración (Nota al pie)

Hay muchos ejemplos de sistemas axiomáticos con sus reglas de derivación.

- Lógica proposicional.
- Lógica de primer orden.
- Lógica intuicionista.
- *Description Logics*.



# Demostración (Nota al pie)

Hay muchos ejemplos de sistemas axiomáticos con sus reglas de derivación.

- Lógica proposicional.
- Lógica de primer orden.
- Lógica intuicionista.
- *Description Logics*.

Cuando hacemos matemática típicamente estamos empleando las dos primeras, aunque hay polémica en su elección, tanto por sus axiomas (axioma de elección) como por sus reglas de derivación (tercero excluido).

# Técnicas

Las demostraciones suelen seguir patrones (al igual que los algoritmos), por lo que es común categorizarlas para facilitar la comunicación y organizar las ideas (al igual que...).

A groso modo, las técnicas troncales son:

- Demostración directa.
- Inducción.
- Demostración por contrarecíproco.
- Demostración por absurdo.
- Por construcción.
- Por contraejemplo.

# Demostración directa

## Idea

Para probar que  $x$  satisface una propiedad  $P(x)$  simplemente tomamos la definición de  $P$  y mostramos que  $x$  la cumple.

## Ejercicio

Probar que  $\log(n!) \in \Theta(n \log n)$ .

# Inducción

## Idea

Para probar que una propiedad  $P$  vale para todos los naturales  $n \in \mathbb{N}$ , vamos a probar que  $P(0)$  es cierto, y que  $P(n) \rightarrow P(n+1)$  para todo  $n$ . Funciona también si queremos probar algo para todos los naturales mayores a un cierto  $k$ .

## Ejercicio

Probar que  $2^n \leq n!$  para todo  $n \geq 4$ .

# Inducción

El esquema anterior es el tradicional, pero también hay otras variantes.

## Inducción completa

En vez de probar  $P(n) \rightarrow P(n+1)$  podemos probar que  $\bigwedge_{k \leq n} P(k) \rightarrow P(n+1)$ . Notar que esto es algo más débil, y por lo tanto debería ser más fácil de probar.

## Inducción estructural

Si tenemos alguna familia de estructuras  $\mathcal{F}$  que se crean cada una con una cierta cantidad de operaciones podemos hacer inducción en esta cantidad:

- 1 Probamos que la propiedad  $P$  vale para las estructuras creadas en 1 paso.
- 2 Probamos que si  $P$  vale para las estructuras creadas en  $n$  pasos, entonces también para las creadas en  $n+1$ .

# Contrarrecíproco

## Idea

Decir que  $\alpha \rightarrow \beta$  es lo mismo que decir  $\neg\beta \rightarrow \neg\alpha$ .

## Ejercicio

Probar que si  $n^2$  es par, entonces  $n$  también lo es.

# Por construcción

## Idea

Si la fórmula habla de la existencia de alguna estructura, podemos simplemente explicar cómo se construye.

## Ejercicio

Probar que para todo conjunto finito  $S \subseteq \mathbb{N}$  vale que existe un polinomio  $p$  no nulo tal que  $p(s) = 0$  para todo  $s \in S$ .

# Por absurdo

## Idea

Para probar que  $\alpha \rightarrow \beta$  vamos a asumir que valen  $\alpha$  y  $\neg\beta$ , para ver luego que esto contradice algún resultado previo.

## Ejercicio

Probar que  $\sqrt{2}$  es irracional.



# Polémicos

Veamos un ejemplo curioso de inducción

**Teorema?**

Todo conjunto finito de caballos es de un mismo color.

# Polémicos

Veamos un ejemplo curioso de inducción

## Teorema?

Todo conjunto finito de caballos es de un mismo color.

Prueba: hagamos inducción en el tamaño del conjunto  $C$  de caballos.

Si  $|C| = 1$  el teorema vale.

Si  $C = \{c_1, \dots, c_{n+1}\}$  hacemos lo siguiente:

- 1 Para  $\{c_1, \dots, c_n\}$  el teorema vale, por lo que los caballos  $c_1, \dots, c_n$  son de un mismo color.
- 2 Para  $\{c_n, c_{n+1}\}$  el teorema también vale.
- 3 Como  $c_n$  está en ambos conjuntos, debe vale que todos los caballos son del mismo color.

# Polémicos

Veamos un ejemplo curioso de inducción

## Teorema?

Todo conjunto finito de caballos es de un mismo color.

Prueba: hagamos inducción en el tamaño del conjunto  $C$  de caballos.

Si  $|C| = 1$  el teorema vale.

Si  $C = \{c_1, \dots, c_{n+1}\}$  hacemos lo siguiente:

- 1 Para  $\{c_1, \dots, c_n\}$  el teorema vale, por lo que los caballos  $c_1, \dots, c_n$  son de un mismo color.
- 2 Para  $\{c_n, c_{n+1}\}$  el teorema también vale.
- 3 Como  $c_n$  está en ambos conjuntos, debe vale que todos los caballos son del mismo color.

¿El teorema es correcto? ¿La demo es correcta?


# Polémicos

La demo tiene un error en el paso inductivo, pero eso no quiere decir que el teorema sea falso. Cómo podríamos probar esto último?

# Polémicos

La demo tiene un error en el paso inductivo, pero eso no quiere decir que el teorema sea falso. Cómo podríamos probar esto último?



Two differently colored horses,   
providing a counterexample to the  
general theorem.

# Backtracking

Intentemos usar inducción para probar cosas que nos importen...

## Fibonacci

Cuál es la complejidad exacta del algoritmo que implementa Fibonacci recursivamente, sin memorización?

# Backtracking

Intentemos usar inducción para probar cosas que nos importen...

## Fibonacci

Cuál es la complejidad exacta del algoritmo que implementa Fibonacci recursivamente, sin memorización?

Deberíamos encontrar el tamaño exacto del árbol de recursión, que tiene cierta pinta a Fibonacci...

# Backtracking

Intentemos usar inducción para probar cosas que nos importen...

## Fibonacci

Cuál es la complejidad exacta del algoritmo que implementa Fibonacci recursivamente, sin memorización?

Deberíamos encontrar el tamaño exacto del árbol de recursión, que tiene cierta pinta a Fibonacci...

Definamos  $G(n)$  = “Cantidad de llamados recursivos al llamar  $fibo(n)$ ” y conjeturemos que  $G(n) = aF_n + b$ , a ver hasta donde llegamos.



# Backtracking

## Recorriendo combinaciones

Supongamos que se quieren procesar todos los subconjuntos de tamaño  $k$  de un conjunto  $A$  de  $n$  elementos. Se proponen dos esquemas de backtracking para generarlos.

# Backtracking

## Propuesta 1

$$bt(i, sol) = \begin{cases} process(sol) & i = 0 \\ bt(i - 1, sol \cup A_i) \oplus bt(i - 1, sol) & \end{cases}$$

## Propuesta 2

$$bt(i, j, sol) = \begin{cases} process(sol) & i = 0 || j = 0 \\ process(sol \cup \{A_1, \dots, A_i\}) & i = j \\ bt(i + 1, j - 1, sol \cup A_i) \oplus bt(i + 1, j, sol) & \end{cases}$$

La inclusión de las podas en la segunda propuesta altera el orden de la cantidad de llamados recursivos que se hacen?

# Backtracking

- La primera formulación hace  $\Theta(2^n)$  llamados.

# Backtracking

- La primera formulación hace  $\Theta(2^n)$  llamados.
- La segunda tiene  $\binom{n}{k}$  hojas. Tendrá también  $\Theta(\binom{n}{k})$  nodos?

# Backtracking

- La primera formulación hace  $\Theta(2^n)$  llamados.
- La segunda tiene  $\binom{n}{k}$  hojas. Tendrá también  $\Theta(\binom{n}{k})$  nodos?
- Definimos  $G(n, k)$  como la cantidad de llamados que se hacen al llamar con  $n$  y  $k$ .

Vale que:

- 1  $G(i, 0) = G(i, i) = 1$  para todo  $0 \leq i \leq n$  (debido a las podas).
- 2 Tenemos que  $G(n, k) = G(n-1, k) + G(n-1, k-1) + 1$ .

# Backtracking

- La primera formulación hace  $\Theta(2^n)$  llamados.
- La segunda tiene  $\binom{n}{k}$  hojas. Tendrá también  $\Theta(\binom{n}{k})$  nodos?
- Definimos  $G(n, k)$  como la cantidad de llamados que se hacen al llamar con  $n$  y  $k$ . Vale que:
  - 1  $G(i, 0) = G(i, i) = 1$  para todo  $0 \leq i \leq n$  (debido a las podas).
  - 2 Tenemos que  $G(n, k) = G(n-1, k) + G(n-1, k-1) + 1$ .
- Igual que antes, podemos conjeturar que  $G(n, k) = a\binom{n}{k} + b$ , y es posible encontrar valores de  $a$  y  $b$  para los cuales esta igualdad es cierta (usar inducción para esto último).

# Selección de Actividades

Tenemos un conjunto de  $n$  actividades, cada una con un tiempo de inicio y finalización ( $s_i$  y  $f_i$ ). Encontrar un subconjunto máximo de actividades que no se intersequen.

# Suma de Subconjuntos

En este ejercicio vamos a resolver el problema de suma de subconjuntos usando la técnica de programación dinámica.

Sea  $n = |C|$  la cantidad de elementos de  $C$ . Considerar la siguiente función recursiva  $c: \{0, \dots, n\} \times \{0, \dots, k\} \rightarrow \{V, F\}$  (donde  $V$  indica verdadero y  $F$  falso) tal que:

$$c(i, j) = \begin{cases} j = 0 & \text{si } i = 0 \\ c(i-1, j) & \text{si } i \neq 0 \wedge C[i] > j \\ c(i-1, j) \vee c(i-1, j - C[i]) & \text{si no} \end{cases}$$

Demostrar que la función recursiva del inciso a) es correcta. **Ayuda:** demostrar por inducción en  $i$  que existe algún subconjunto de  $\{c_1, \dots, c_i\}$  que suma  $j$  si y solo si  $c(i, j) = V$ .