Teorema de Cantor y algoritmos

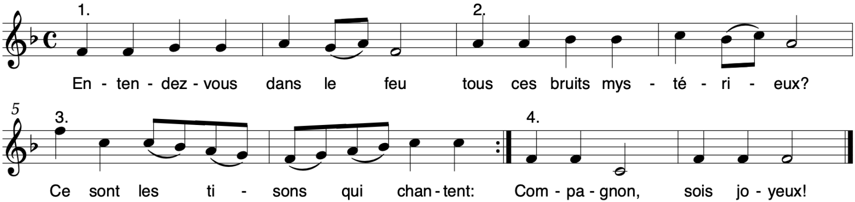
Clase 19

IIC 1253

Prof. Sebastián Bugedo

Outline

Entendez-vous Traditional



Entendez-vous dans le feu
tous ces bruits mystérieux?

Ce sont les tisons qui chantent:
Compagnon, sois joyeux!

Cardinalidad de conjuntos infinitos

¿Existen conjuntos infinitos **no** enumerables?

Teorema

El intervalo real $(0, 1) \subseteq \mathbb{R}$ es infinito pero no enumerable.

Ejercicio

Demuestre el teorema.

Cardinalidad de conjuntos infinitos

¿Qué otros conjuntos tienen la misma cardinalidad que $(0,1)$?

Teorema

$(0,1) \approx \mathbb{R} \approx \mathcal{P}(\mathbb{N})$.

Objetivos de la clase

- Demostrar el teorema de Cantor y comprender su alcance
- Comprender concepto de algoritmo
- Identificar las componentes del análisis de algoritmos
- Demostrar correctitud de algoritmos

Outline

Cardinalidad de conjuntos infinitos

Entonces, ¿dónde hay más elementos, en $\mathbb{N}$ o en $\mathbb{R}$?

Definición

Dados conjuntos A y B, diremos que $A \preceq B$ (A **no es más grande** que B) si existe una función inyectiva $f : A \to B$.

¿Es $\preceq$ una relación de orden?

Si $A \preceq B$, diremos que $|A| \leq |B|$.

Cardinalidad de conjuntos infinitos

Definición

Dados conjuntos A y B, diremos que $A \prec B$ (A es **menos numeroso** que B) si $A \preceq B$ pero $A \not\approx B$.

¿Cómo se define esta noción usando funciones?

- Existe función inyectiva $f : A \to B \ldots$
- $\ldots$ pero no existe función biyectiva $g : A \to B$.

Si $A \prec B$, diremos que $|A| < |B|$.

Cardinalidad de conjuntos infinitos

Ejemplo

$\mathbb{N}$ es menos numeroso que $\mathbb{R}$, y por lo tanto decimos que $|\mathbb{N}| < |\mathbb{R}|$.

¡Hay estrictamente **menos** números naturales que reales!

Corolario

$|\mathbb{N}| < |\mathcal{P}(\mathbb{N})|$.

Demostramos algo parecido para el caso finito...
veremos que aplica para **todo conjunto**

Cardinalidad de A vs $\mathcal{P}(A)$

Dado un conjunto A (no necesariamente finito):

Teorema (Cantor)

A es menos numeroso que su conjunto potencia ($|A| < |\mathcal{P}(A)|$).

¡Podemos repetir este proceso *ad eternum*!

$$|A| < |\mathcal{P}(A)| < |\mathcal{P}(\mathcal{P}(A))| < |\mathcal{P}(\mathcal{P}(\mathcal{P}(A)))| < \cdots$$

Cardinalidad de A vs $\mathcal{P}(A)$

Teorema (Cantor)

A es menos numeroso que su conjunto potencia ($|A| < |\mathcal{P}(A)|$).

Ejercicio

Demuestre el teorema.

Primero, es claro que $|A| \leq |\mathcal{P}(A)|$. Basta tomar

$$f : A \to \mathcal{P}(A) \text{ dada por } f(a) = \{a\}$$

la cual es claramente inyectiva.

Ahora mostraremos que no existe una función biyectiva entre A y $\mathcal{P}(A)$.

Por contradicción, supongamos que sí existe una biyección f entre A y $\mathcal{P}(A)$.

Diagonalización entre A y $\mathcal{P}(A)$

Considere el siguiente conjunto:

Definición (complemento de la diagonal)

$$\bar{D} = \{a \in A \mid a \notin f(a)\}$$

Notemos que $\bar{D} \subseteq A$, y por lo tanto $\bar{D} \in \mathcal{P}(A)$. Luego, como f es biyectiva, debe existir $x \in A$ tal que $f(x) = \bar{D}$. Considere ahora los siguientes casos:

- Si $x \in f(x)$, entonces $x \in \bar{D}$ (porque $f(x) = \bar{D}$), pero por definición de $\bar{D}, x \notin f(x)$.
- Si $x \notin f(x)$, entonces $x \notin \bar{D}$ (porque $f(x) = \bar{D}$), pero por definición de $\bar{D}, x \in f(x)$.

Luego, $x \in f(x)$ si y sólo si $x \notin f(x)$, lo cual es una contradicción. Por lo tanto, no existe una biyección entre A y $\mathcal{P}(A)$. $\square$

Reflexiones finales

Dos preguntas

- ¿Cuántos "infinitos" existen? Infinitos!
- ¿Hay algún infinito entre $|\mathbb{N}|$ y $|\mathbb{R}|$? ???

Hipótesis del continuo

No existe conjunto A tal que $|\mathbb{N}| < |A| < |\mathbb{R}|$

¿Por qué se llama hipótesis?

Reflexiones finales

- ¿Qué implica todo lo anterior para la computación?
  - ¿Qué cosas es capaz de hacer un computador?
  - ¿Qué cosas **no** es capaz de hacer?
  - ¿Existen problemas computacionales para los cuales no existan algoritmos que los resuelvan?
  - Respuesta: $\mathrm{IIC2213}$:)

Outline

Tercer Acto: Aplicaciones

Algoritmos, grafos y números

Playlist Tercer Acto

DiscretiWawos #3

Además sigan en instagram:
@orquesta_tamen

Hacia la noción de algoritmo

Necesitamos formalizar la noción de **algoritmo**

Nos interesa la idea de **computación efectiva**

- En el sentido de que *efectivamente puede realizarse*

¿Podemos definir formalmente esta noción?

Intentos de formalización: S. XX

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

λ-calculus
por Alonzo Church.

Sistemas de Post
por Emil Post.

Máquinas de Turing
por Alan Turing.

...

Todos estos métodos son equivalentes

¿Qué es un algoritmo?

Estos métodos capturan la noción intuitiva:

- Algoritmos como secuencias de pasos
- Con precondiciones
- Condiciones de término

Esencialmente, un **algoritmo** es un conjunto de pasos que resuelven un **problema**

Para este curso nos basta con esta intuición

Algoritmos

Diremos entonces que un **algoritmo** es un método para convertir un **INPUT** válido en un **OUTPUT**. A estos métodos les exigiremos ciertas propiedades:

- Precisión: cada instrucción debe ser planteada de forma precisa y no ambigua.
- Determinismo: cada instrucción tiene un único comportamiento que depende sólo del input.
- Finitud: el algoritmo está compuesto por un conjunto finito de instrucciones.

Algoritmos

El análisis de algoritmos es una disciplina de la Ciencia de la Computación que tiene dos objetivos:

- Estudiar cuándo y por qué los algoritmos son **correctos** (es decir, hacen lo que dicen que hacen).
- Estimar la cantidad de **recursos** computacionales que un algoritmo necesita para su ejecución.

De esta manera, podemos, por ejemplo:

- Entender bien los algoritmos, para luego reutilizarlos total o parcialmente.
- Determinar qué mejorar de un algoritmo para que sea más eficiente.

Algoritmos

Usaremos pseudo-código para escribir algoritmos.

- Instrucciones usuales como **if**, **while**, **return**. . .
- Notaciones cómodas para arreglos, conjuntos, propiedades lógicas, etc.

Consideraremos que los algoritmos tienen:

- **Precondiciones:** representan el input del programa.
- **Postcondiciones:** representan el output del programa, lo que hace el algoritmo con el input.

Outline

Corrección de algoritmos

Queremos determinar cuándo un algoritmo es correcto; es decir, hace lo que dice que hace.

Definición

Un algoritmo es **correcto** si para todo INPUT válido, el algoritmo se detiene y produce un OUTPUT correcto.

Entonces, ¿cuándo es incorrecto?

Definición

Un algoritmo es **incorrecto** si existe un INPUT válido para el cual el algoritmo no se detiene o produce un OUTPUT incorrecto.

Algoritmos iterativos

Debemos demostrar dos cosas:

- **Corrección parcial:** si el algoritmo se detiene, se cumplen las postcondiciones.
- **Terminación:** el algoritmo se detiene.

Nos preocupamos sólo de los *loops* de los algoritmos (¿por qué?).

Estos loops tienen una condición G que determina si se siguen ejecutando:

while(G)

...

end

Algoritmos iterativos

Para demostrar corrección parcial, buscamos un **invariante** $\mathcal{I}(k)$ para los loops:

- Una propiedad $\mathcal{I}$ que sea verdadera en cada paso k de la iteración.
- Debe relacionar a las variables presentes en el algoritmo.
- Al finalizar la ejecución, debe asegurar que las postcondiciones se cumplan.

Algoritmos iterativos

Una vez que encontramos un invariante, demostramos la corrección del loop inductivamente:

- **Base:** las precondiciones deben implicar que $\mathcal{I}(0)$ es verdadero.
- **Inducción:** para todo natural $k > 0$, si G e $\mathcal{I}(k)$ son verdaderos antes de la iteración, entonces $\mathcal{I}(k+1)$ es verdadero después de la iteración.
- **Corrección:** inmediatamente después de terminado el loop (i.e. cuando G es falso), si $k = N$ e $\mathcal{I}(N)$ es verdadero, entonces la postcondiciones se cumplen.

Y para demostrar terminación, debemos mostrar que existe un k para el cual G es falso.

Algoritmos iterativos

Ejercicio

Escriba un algoritmo que multiplique dos números naturales (sin usar la multiplicación):

- **Pre:** $n, m \in \mathbb{N}$.
- **Post:** $p = n \cdot m$.

Demuestre que su algoritmo es correcto.

(Solución: Apuntes Jorge Pérez, Sección 3.1.1, Teorema 3.1.1, páginas 97 a 99.)

Algoritmos iterativos

Demostración

Proponemos el siguiente algoritmo iterativo

input : $n, m \in \mathbb{N}$

output: $p = n \cdot m$

`Multipy`(n, m)**:**

    $z \leftarrow 0$

    $w \leftarrow m$

    while $w \neq 0$ **do**

        $z \leftarrow z + x$

        $w \leftarrow w - 1$

    return z

Ahora debemos determinar un invariante para el bloque **while**.

Algoritmos iterativos

Demostración

input : $n, m \in \mathbb{N}$
output: $p = n \cdot m$

```
Multipy(n, m):
    z ← 0
    w ← m
    while w ≠ 0 do
        z ← z + x
        w ← w − 1
    return z
```

Si $n = a$ y $m = b$, luego de la iteración i se cumple

i	n	m	z	w
0	a	b	0	b
1	a	b	a	$b-1$
2	a	b	$2a$	$b-2$
3	a	b	$3a$	$b-3$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
b	a	b	$b \cdot a$	0

Observemos que

- Existen en total b iteraciones
- Al término de cada una se cumple

$$z_i = n \cdot (m - w_i)$$

Algoritmos iterativos

Demostración

Demostraremos la **corrección parcial del algoritmo** con el siguiente invariante:

$$P(i) \quad := \quad \text{Al término de la iteración } i\text{, se cumple } z_i = n \cdot (m - w_i)$$

Usamos inducción simple sobre el número de iteraciones.

- **CB:** $P(0)$ corresponde al estado previo a la primera iteración. Se tiene
$$z_0 = 0 = n(m - m) = n(m - w_0)$$
- **HI:** Supongamos que $P(i)$ es cierta.
- **TI:** Demostraremos que $P(i + 1)$ es cierta.

Algoritmos iterativos

Demostración

- **TI:** Demostraremos que $P(i+1)$ es cierta.

  Tenemos que

$$\begin{array}{rcll} z_{i+1} & = & z_i + n & \text{(línea 4 de } \texttt{Multiply}\text{)} \\ & = & n \cdot (m - w_i) + n & \text{(hipótesis inductiva)} \\ & = & n \cdot (m - w_i + 1) & \text{(factorización)} \\ & = & n \cdot (m - (w_i - 1)) & \text{(factorización)} \\ & = & n \cdot (m - w_{i+1}) & \text{(línea 5 de } \texttt{Multiply}\text{)} \end{array}$$

Esto demuestra que $P(i)$ es cierta para cada iteración, por lo que `Multiply` cumple corrección parcial.

Ahora debemos probar que `Multiply` termina.

Algoritmos iterativos

Demostración

Observemos que el bloque **while** termina cuando $w = 0$. En la iteración 0, $w = m$ natural y en cada iteración se reduce en 1. Es decir, los valores de w forman una sucesión decreciente de naturales, que en m iteraciones llega a $w = 0$. Por lo tanto, el algoritmo termina.

A partir de estos resultados, `Multiply` es correcto. $\square$

Algoritmos recursivos

En el caso de los algoritmos recursivos, no necesitamos dividir la demostración en corrección parcial y terminación (¿por qué?).

- Basta demostrar por inducción la propiedad (corrección) deseada.
- En general, la inducción se realiza sobre el tamaño del input.

Ejercicio

Escriba un algoritmo recursivo que encuentre el máximo elemento de un arreglo:

- **Pre:** un arreglo $A = [a_0, a_1, \ldots, a_{n-1}]$, y un natural n (largo del arreglo).
- **Post:** $m = \max(A)$.

Demuestre que el algoritmo es correcto.

Solución: Apuntes Jorge Pérez, Sección 3.1.1, página 101.

Algoritmos recursivos

Demostración

Proponemos el siguiente algoritmo recursivo

input : Arreglo $A = [a_0, \ldots, a_{n-1}]$ y largo $n \geq 1$
output: $m = \max(A)$

`RecMax`(A, n)**:**
    if $n = 1$ **then**
        return a_0
    else
        $k \leftarrow$ `RecMax`$(A, n-1)$
        if $a_{n-1} \geq k$ **then**
            return a_{n-1}
        else
            return k

Observemos que el llamado `RecMax`(A, i) solo toma en cuenta los primeros i elementos de A, es decir, el tramo $[a_0, a_1, \ldots, a_{i-1}]$.

Demostración

Demostraremos la **corrección del algoritmo** con el siguiente invariante sobre número de elementos considerados:

$$P(i) \quad := \quad \text{El valor retornado por } \texttt{RecMax}(A, i) \text{ cumple } \texttt{RecMax}(A, i) \geq a_0, a_1, \ldots, a_{i-1}$$

Usamos inducción simple sobre el número de elementos considerados.

- **CB:** $P(1)$ considera solo el tramo $[a_0]$ y su retorno cumple $a_0 \geq a_0$.
- **HI:** Supongamos que $P(i)$ es cierta, i.e.

$$\texttt{RecMax}(A, i) \geq a_0, a_1, \ldots, a_{i-1}$$

- **TI:** Demostraremos que $P(i+1)$ es cierta, i.e.

$$\texttt{RecMax}(A, i+1) \geq a_0, a_1, \ldots, a_{i-1}, a_i$$

Algoritmos recursivos

Demostración

- **TI:** Demostraremos que $P(i+1)$ es cierta, i.e.

$$\texttt{RecMax}(A, i+1) \geq a_0, a_1, \ldots, a_{i-1}, a_i$$

Supongamos que se ejecutó $\texttt{RecMax}(A, i+1)$. Dado que el número de elementos es estrictamente mayor a 1, no estamos en el caso base y se hace un llamado a $\texttt{RecMax}(A, i)$.

Por **HI** dicho llamado es correcto y queda guardado en k, i.e.

$$k \geq a_0, a_1, \ldots, a_{i-1}$$

Este valor puede cumplir uno de dos casos en el **if** de la línea 5:

  - Cumple $a_i \geq k$. Luego, por transitividad, $a_i \geq a_0, a_1, \ldots, a_{i-1}, a_i$ y $\texttt{RecMax}(A, i+1)$ sería el máximo de A.
  - Cumple $a_i < k$. En tal caso, $k \geq a_0, a_1, \ldots, a_{i-1}, a_i$ y el retorno $\texttt{RecMax}(A, i+1)$ sería el máximo de A.

Con lo anterior, se prueba que $\texttt{RecMax}(A, i)$ es correcto. $\square$

Complejidad de algoritmos

Ya vimos cómo determinar cuando un algoritmo era correcto.

- Esto no nos asegura que el algoritmo sea útil en la práctica.
- Necesitamos estimar su tiempo de ejecución.
  - En función del tamaño del input.
  - Independiente de: lenguaje, compilador, hardware...

Lo que nos interesa entonces no es el tiempo *exacto* de ejecución de un algoritmo, sino que su comportamiento a medida que crece el input.

Introduciremos notación que nos permitirá hablar de esto.

Complejidad de algoritmos

Vamos a ocupar funciones de dominio natural ($\mathbb{N}$) y recorrido real positivo ($\mathbb{R}^+$).

- El dominio será el tamaño del input de un algoritmo.
- El recorrido será el tiempo necesario para ejecutar el algoritmo.

Outline

Objetivos de la clase

- Demostrar el teorema de Cantor y comprender su alcance
- Comprender concepto de algoritmo
- Identificar las componentes del análisis de algoritmos
- Demostrar correctitud de algoritmos