Computación Bio–inspirada

Tema 1: Introducción

David Orellana Martín
Mario de J. Pérez Jiménez

Grupo de Investigación en Computación Natural
Dpto. Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla

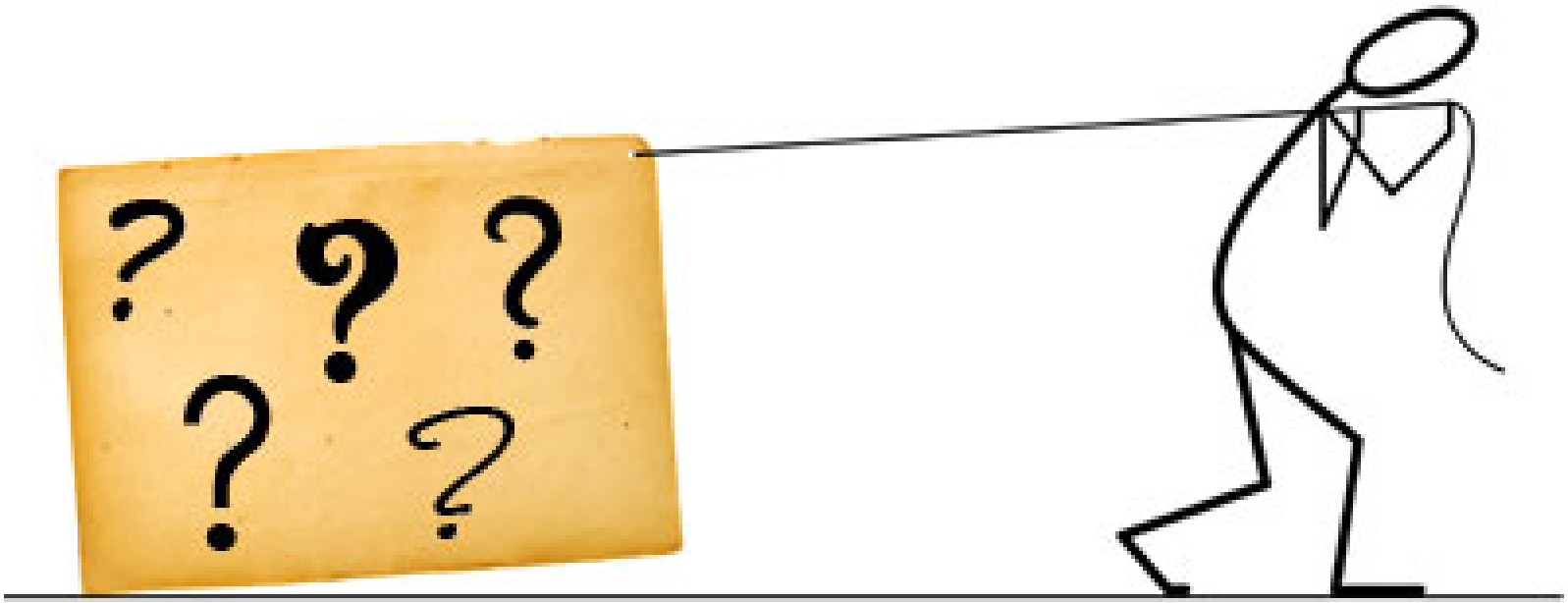
dorellana@us.es (http://www.cs.us.es/∼dorellana/)
marper@us.es (http://www.cs.us.es/∼marper/)

Máster Universitario en Lógica, Computación e Inteligencia Artificial
Curso 2024-2025

Índice

- Problemas abstractos versus problemas concretos.
- Teoría de la Computabilidad.
- Teoría de la Complejidad Computacional.
- El problema **P** versus **NP**.
- Computación Natural.

Problemas



Problemas, problemas, problemas ...

Problemas, problemas, problemas ...

Una eterna aspiración del hombre ...

⋆ **Mejorar la calidad de Vida**.

Para ello, necesita resolver **problemas**.

- A ser posible usando "**procedimientos mecánicos**" ...
  ("**Solución mecánica**" de un problema)

Problemas abstractos vs problemas concretos

(1) Determinar si el número 2234567891 es primo.

(2) Calcular el producto de dos números naturales.

(3) Hallar el máximo común divisor de 194317 y 597684.

(4) Determinar si la suma de los ángulos de un triángulo es $182^{\underline{o}}$.

(5) Para cada número natural n hallar un número primo y mayor que n.

(6) Hallar la suma de los números 3751205 y 64535679.

(7) Determinar si un número natural n es primo.

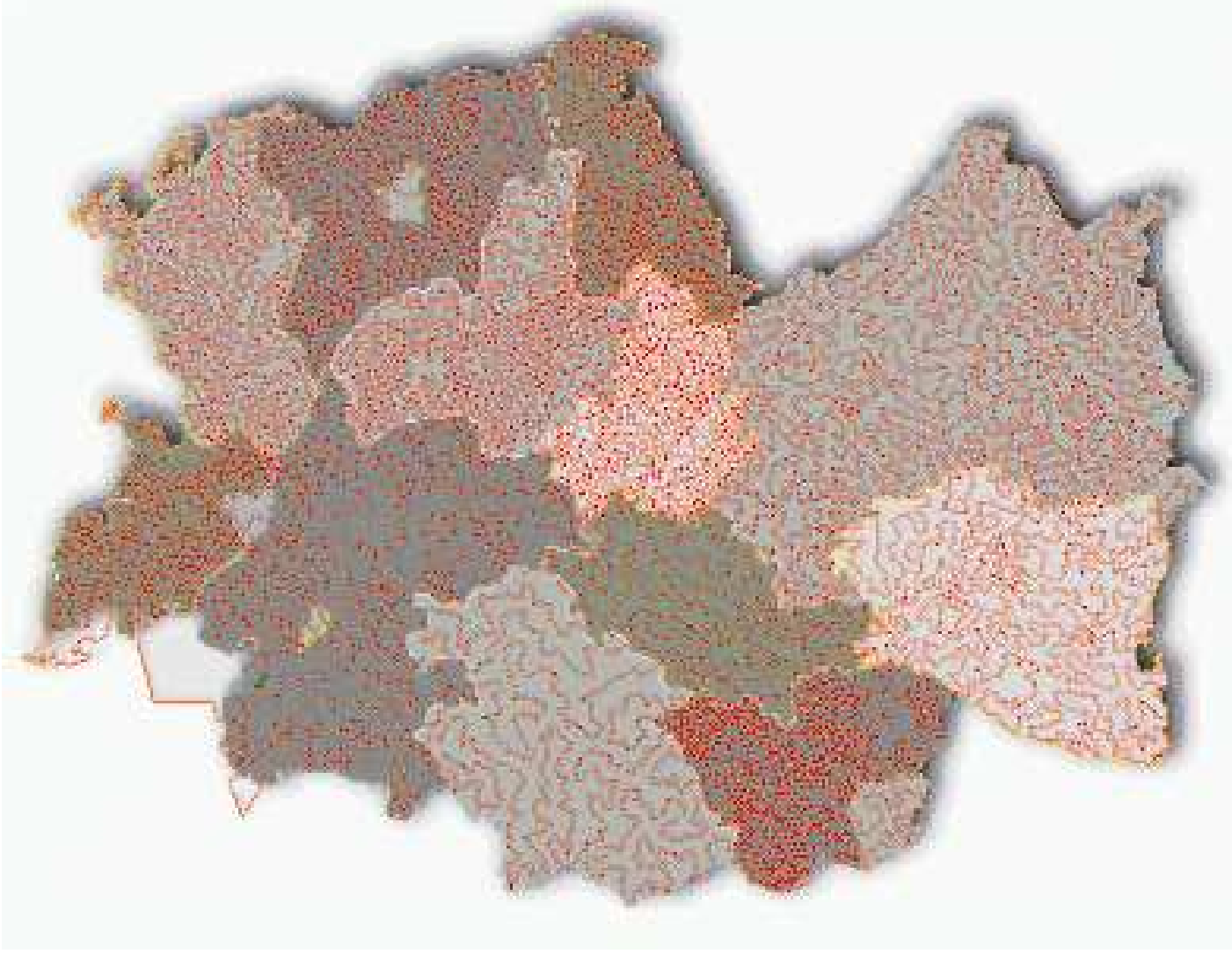
(8) Hace un par de horas, una empresa de reparto ha recibido 37 paquetes de un conocido hipermercado, para su entrega a unos clientes, esta misma tarde. Sabiendo que los paquetes tienen cabida en un camión, ¿qué ruta debe seleccionar el conductor para maximizar las ganancias?

Un problema concreto

- Dadas 42 ciudades de las que se conocen los tiempos t_{ij} para ir de la ciudad i a la ciudad j, hallar un circuito que recorra todas las ciudades en el **menor** tiempo posible.

Otro problema concreto

- Dadas 3150 ciudades de las que se conocen los tiempos t'_{ij} de la ciudad i a la ciudad j, hallar un circuito que recorra todas las ciudades en el menor tiempo posible.



Un problema abstracto

Dadas n ciudades y unos valores t_{ij} que representan los tiempos para ir de la ciudad i a la ciudad j, hallar un circuito que permita recorrer todas las ciudades en el menor tiempo posible.

Problema del **viajante de comercio** (**TSP**: **T**ravelling **S**alesman **P**roblem).

Problemas concretos vs Problemas abstractos

Problema abstracto: conjunto de **problemas concretos** (**instancias**).

- **Tamaño** de un problema concreto.

Cómo **resolver**, en la "práctica", un **problema de la vida real**.

- ⋆ Un problema de la vida real suele ser un problema **concreto**.
- ⋆ Se **modeliza** o representa a través de un problema **abstracto**.
- ⋆ Se diseña una **solución mecánica** del problema abstracto.
- ⋆ Se **implementa** (describe) esa solución mediante un programa (en un lenguaje).
- ⋆ Se **ejecuta** ese programa sobre una "máquina" introduciendo los datos específicos del problema concreto.

Soluciones mecánicas de problemas abstractos

Procedimiento mecánico/Algoritmo:

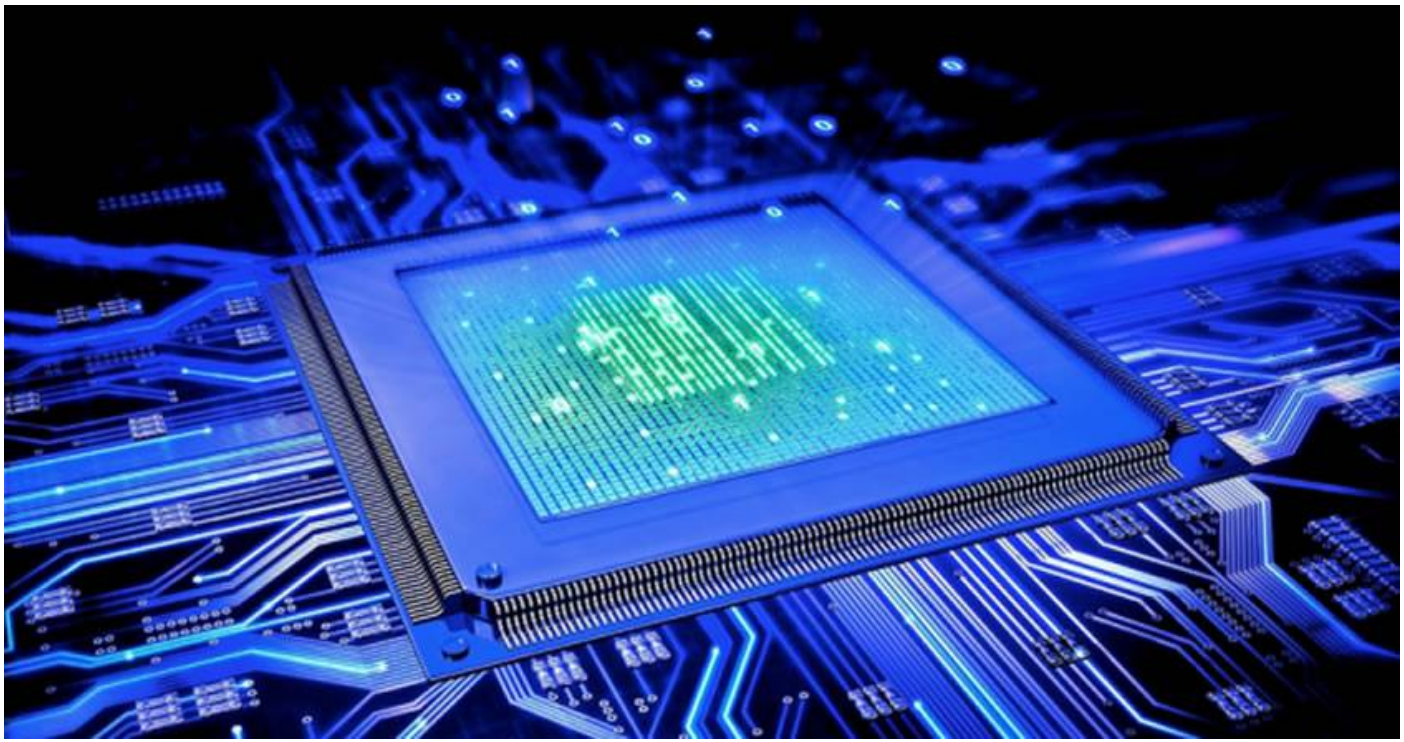
- ⋆ Método **especial** de resolución de problemas.
- ⋆ **Primer algoritmo no trivial**: máximo común divisor de dos números enteros (Euclides entre 400 y 300 a. C.).

Abú Jáfar Mohammed ibn **al–Khowarizmi**:

- ⋆ Procedimientos mecánicos para el Álgebra (año 825 d.C.).

Existencia de problemas **resolubles mecánicamente**.

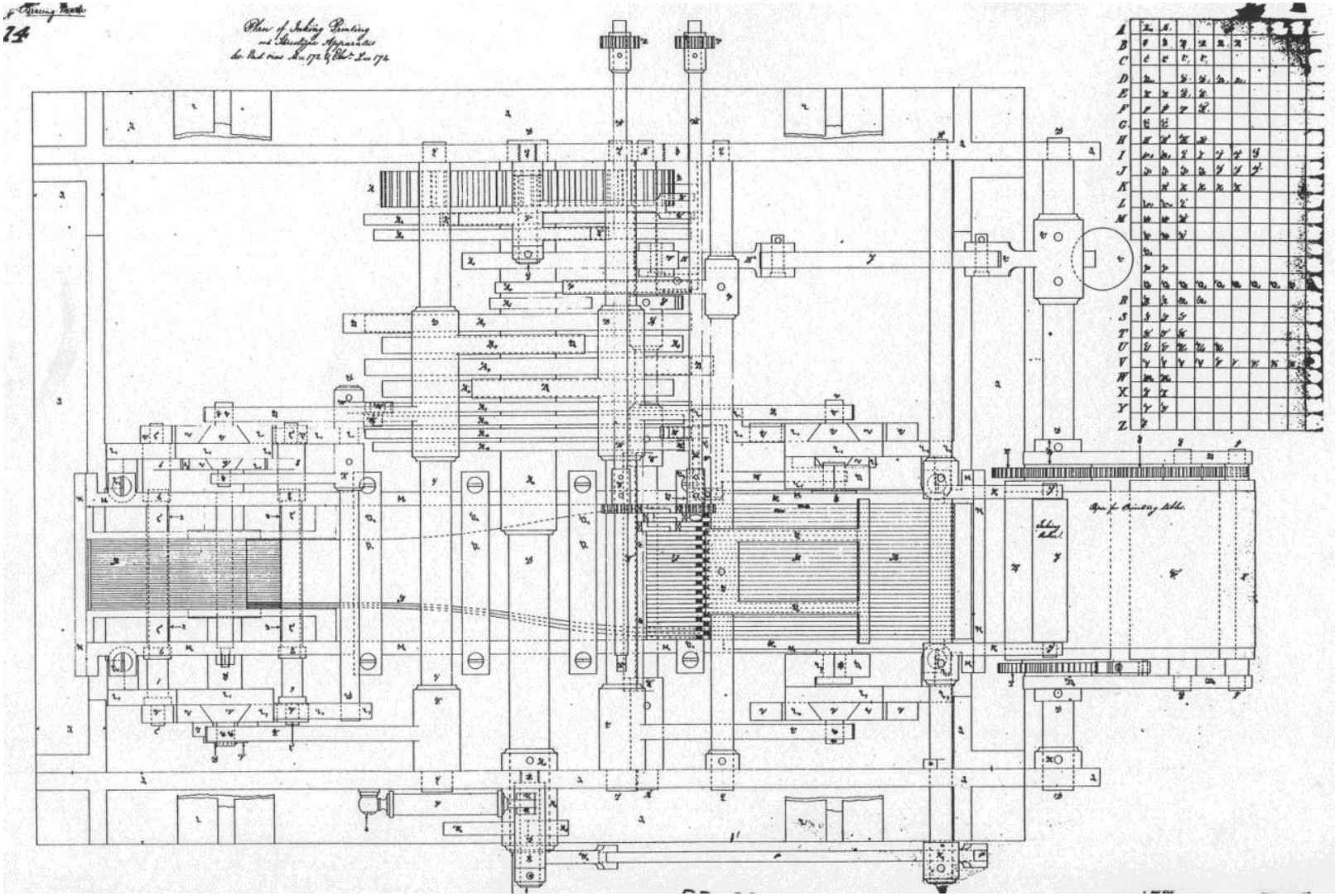
Máquinas, máquinas, máquinas...



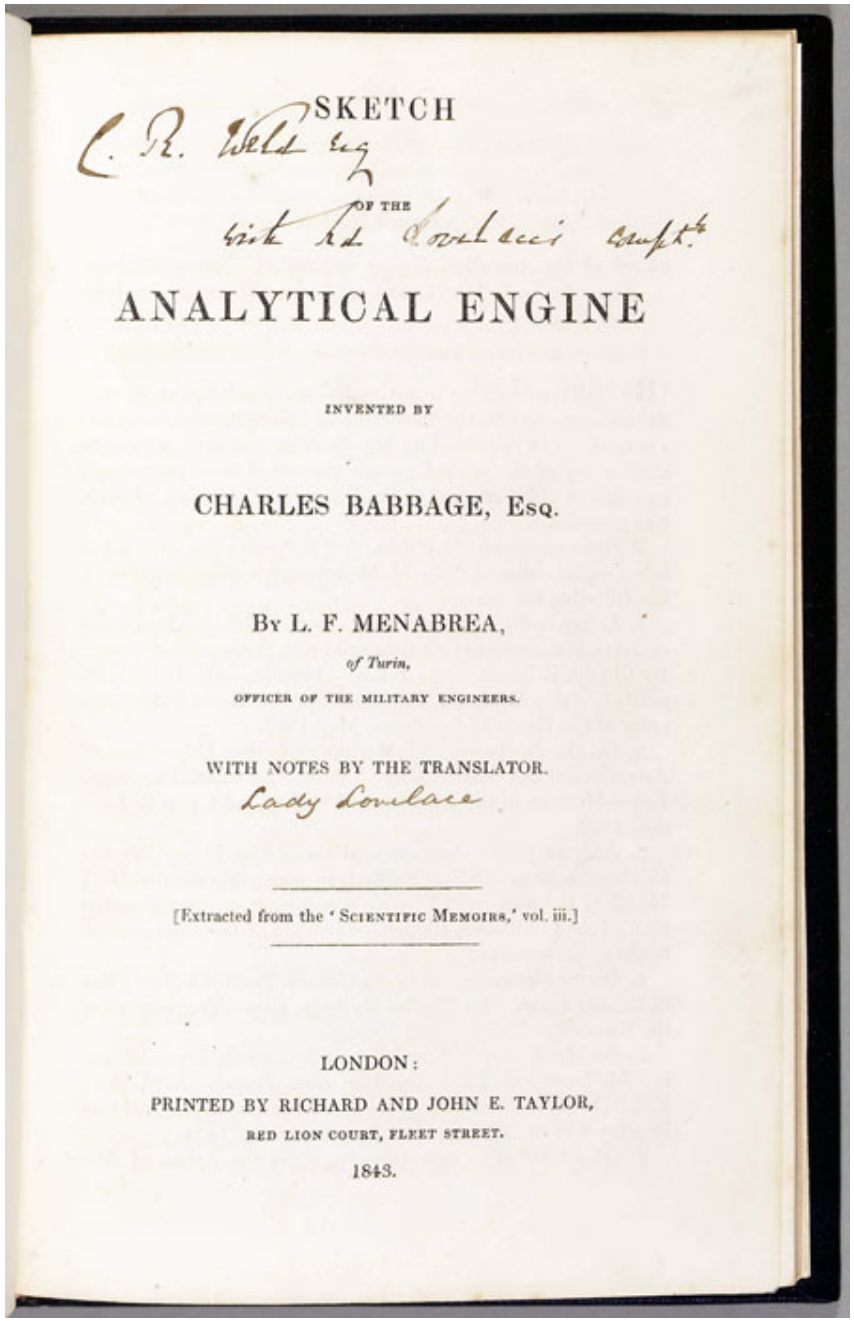
* **Asistentes** para el hombre: tareas tediosas... y otras **tareas inteligentes**.
* Máquinas de **propósito específico** (ábaco, tablas de Néper, máquina de Pascal, máquina de Leibniz, el telar de Jacquard, máquina de diferencias, ...).

Máquinas de propósito general

El sueño de la **máquina analítica**: Charles Babbage (1843-1849).



Máquinas de propósito general



SKETCH

OF THE

ANALYTICAL ENGINE

INVENTED BY

CHARLES BABBAGE, Esq.

By L. F. MENABREA,

of Turin,

OFFICER OF THE MILITARY ENGINEERS.

WITH NOTES BY THE TRANSLATOR.

[Extracted from the 'Scientific Memoirs,' vol. iii.]

LONDON:

PRINTED BY RICHARD AND JOHN E. TAYLOR,

RED LION COURT, FLEET STREET.

1843.

Problemas abstractos

- Problemas de **búsqueda** (**search problems**).
  - Dado un grafo no dirigido y un número natural k, hallar un **clique** de tamaño k (si no existe un tal clique, responder **No**).

    NOTA: Un **clique** de un grafo no dirigido $G = (V, E)$ es un conjunto de vértices $V' \subseteq V$ tal que dos nodos distintos cualesquiera de V' siempre están conectados por una arista de G.

- Problemas de **optimización** (**optimization problems**).
  - Dado un grafo no dirigido, hallar un **clique** que tenga **tamaño máximo**.

    NOTA: El **tamaño** de un clique V' es el número de nodos de V'.

- Problemas de **decisión** (**decision problems**).
  - Dado un grafo no dirigido y un número natural k, determinar si existe un **clique** de tamaño k.

- Problemas de **recuento** (**counting problems**).
  - Dado un grafo no dirigido y un número natural k, determinar el número de **cliques** de tamaño k que existen en dicho grafo.

Lenguajes formales

- **Alfabeto**: conjunto no vacío (**símbolos**).
- **Cadena** o **palabra** sobre un alfabeto Γ: **sucesión finita** de símbolos de Γ.
- **Longitud de una cadena** u: número de símbolos que contiene ($|u|$).
- **Concatenación** de cadenas sobre un alfabeto Γ: "composición" de cadenas.
- La **cadena vacía** consta de 0 símbolos y se suele notar por λ.
- Γ^* : conjunto de todas las cadenas sobre Γ.
- Γ^+ : $\Gamma^* - \{\lambda\}$.
- Γ^n= conjunto de cadenas sobre Γ de longitud n.
- **Lenguaje formal** sobre un alfabeto Γ: un subconjunto de Γ^*.

Problemas de búsqueda

Un **problema de búsqueda**, X, es una terna ordenada (Σ_X, I_X, s_X) donde:

* Σ_X es un alfabeto.
* I_X es un lenguaje sobre Σ_X (cuyos elementos se denominan **instancias**).
* s_X es una función cuyo domino es I_X tal que para cada instancia $a \in I_X$, $s_X(a)$ es un conjunto cuyos elementos se denominan **soluciones del problema** (obviamente, el conjunto $s_X(a)$ puede ser vacío).

RESOLVER un problema de búsqueda: para cada $a \in I_X$, si $s_X(a) \neq \emptyset$ entonces hallar **un** elemento de $s_X(a)$; en caso contrario, responder **No**.

Problemas de optimización

Un **problema de optimización**, X, es una 4-tupla $(\Sigma_X, I_X, s_X, f_X)$ donde:

* $(\Sigma_X, I_X, s_X,)$ es un problema de búsqueda.
* Para cada instancia $a \in I_X$ el conjunto $s_X(a)$ es no vacío (sus elementos se denominan **soluciones candidatas del problema**).
* f_X es una función (**objetivo**) que asigna a cada instancia $a \in I_X$ y cada solución candidata asociada $c_a \in s_X(a)$, un número racional positivo $f_X(a, c_a)$.

f_X proporciona un criterio para determinar qué es una **mejor** solución.

* Una **solución óptima** para $a \in I_X$ es una solución $c \in s_X(a)$ tal que:
  - O bien, $\forall c' \in s_X(a)$ se tiene $f_X(a, c) \leq f_X(a, c')$ (c es una **solución minimal**);
  - O bien $\forall c' \in s_X(a)$ se tiene $f_X(a, c) \geq f_X(a, c')$ (c es una **solución maximal**).

RESOLVER un problema de optimización: para cada $a \in I_X$, hallar una solución óptima para a, en caso de que exista; de lo contrario, responder **No**.

Problemas de decisión

Un **problema de decisión**, X, es un problema de búsqueda (Σ_X, I_X, s_X) tal que:

* Para cada $a \in I_X$ se tiene: o bien $s_X(a) = \{0\}$, o bien $s_X(a) = \{1\}$.

Sea $a \in I_X$ tal que $s_X(a) = \{0\}$: **la respuesta del problema a esa instancia es negativa**

Sea $a \in I_X$ tal que $s_X(a) = \{1\}$: **la respuesta del problema a esa instancia es afirmativa**

Consideraciones interesantes: Todo problema de decisión $X = (\Sigma_X, I_X, s_X)$

* Se puede considerar como un problema de optimización $(\Sigma_X, I_X, s_X, f_X)$ tal que: para cada $a \in I_X$ se tiene que $f_X(a, 0) = 1$, o bien $f_X(a, 1) = 1$.
* Tiene asociado un predicado, θ_X, sobre I_X: $\theta_X(a) = 1 \iff s_X(a) = \{1\}$.
* Tiene asociado un lenguaje $L_X = \{a \in I_X : \ \theta_X(a) = 1\}$.

Además, todo lenguaje L sobre un alfabeto Σ tiene asociado un problema de decisión X_L tal que: $I_{X_L} = \Sigma^*$ y $\forall a \in I_{X_L}$ ($\theta_{X_L}(a) = 1$ si y sólo si $a \in L$).

RESOLVER un problema de decisión: para cada $a \in I_X$, si $s_X(a) = \{1\}$ entonces responder **Sí**; caso contrario, responder **No**.

Problemas de recuento

Un **problema de recuento**, X, es una 4-tupla $(\Sigma_X, I_X, s_X, card_X)$ donde:

* (Σ_X, I_X, s_X) es un problema de búsqueda.
* $card_X$ es una función que asigna a cada instancia $a \in I_X$, el cardinal del conjunto $s_X(a)$ (número de soluciones asociadas a esa instancia).

RESOLVER un problema de recuento: para cada instancia $a \in I_X$, hallar el número de soluciones asociadas a esa instancia.

Problemas de decisión versus problemas de optimización

Todo problema de optimización se puede "**transformar**" en un problema de decisión "**equivalente**".

¿Qué **peaje** se ha de pagar para "transformar" un problema de optimización en un problema de decisión "equivalente"?

- ⋆ Poco "coste computacional".

Ejemplo: El problema **MINIMUM VERTEX COVER**:

- ⋆ Versión de **optimización:** Dado un grafo no dirigido G, hallar el tamaño mínimo de un **recubrimiento de vértices** de G.

  (Un **recubrimiento de vértices** de un grafo no dirigido $G = (V, E)$ es un conjunto de vértices $V' \subseteq V$ tal que cada arista de G contiene, al menos, un vértice de V')

- ⋆ Versión de **decisión:** Dado un grafo no dirigido G y un número natural k, determinar si G posee un r.v. de tamaño *menor o igual* que k.

Supongamos que $\mathcal{A}$ es un algoritmo que resuelve la versión de decisión del problema **MINIMUM VERTEX COVER**.

Entonces se considera el siguiente algoritmo $\mathcal{B}$:

```
Entrada: Un grafo no dirigido G = (V, E); sea V = {x_1, ..., x_n} con n ≥ 1.
  Para k = 1 hasta k = n hacer
    Si la ejecución de A con entrada (G, k) devuelve Sí entonces
      devolver k
```

El algoritmo $\mathcal{B}$ resuelve la versión de optimización del problema **MINIMUM VERTEX COVER**.

El coste en tiempo de $\mathcal{B}$ es menor o igual al coste en tiempo de $\mathcal{A}$ multiplicado por n.

Paradigmas de computación

Hacia finales del siglo XIX:

- Lista de problemas importantes de los que se desconocen **soluciones mecánicas**.

Cuestión 1: Dado un problema abstracto, determinar si existe un procedimiento mecánico que lo resuelve.

- Respuestas positivas *versus* respuestas negativas: **asimetría**.

¿En qué consiste un modelo de computación?

- **Formalizar** el concepto de procedimiento mecánico.

Primeros modelos de computación

- ⋆ **Funciones recursivas**: **K. Gödel**, 1931–1934.
- ⋆ **λ–cálculo**: **A. Church** y **S. Kleene**, 1931-1936.
- ⋆ **Máquinas de Turing**: **A. Turing**, 1936.

Modelo de computación: *sintaxis* y *semántica*.

- ⋆ **Procedimiento mecánico**, **funciones computables**, **máquinas**.

Modelos de computación orientados a:

- **Programas** (Modelo `GOTO`, modelo `WHILE`, etc.)
- **Funciones** (Funciones recursivas, λ-calculables, etc.)
- **Máquinas** (Máquinas de Turing, de Post, URM, etc.)

Problema **decidible** en un modelo: problema resoluble por algún procedimiento mecánico del mismo.

En todos los modelos antes citados, existen problemas **decidibles**

¿Existen problemas indecidibles en un modelo de computación?

Potencia de un modelo de computación.

- Existencia de problemas **no** resolubles mecánicamente (**indecidibles**).
  - ⋆ Indecidibilidad de la lógica de primer orden (Church, abril de 1936).
    - **NO** existe un procedimiento mecánico tal que, dada una fórmula de la lógica de primer orden, decida si es o no un teorema de ese sistema lógico.
    - Respuesta negativa al **Entscheidungsproblem**.
  - ⋆ Indecidibilidad de la lógica de primer orden (Turing, mayo de 1936).
  - ⋆ Problema de la parada (Turing, mayo de 1936).
  - ⋆ Equivalencia de los modelos de computación (Turing, agosto de 1936).

La **tesis de Church–Turing** (Church, 1935 y Turing, 1936).

Modelos universales.

Teoría de la Computabilidad

Objetivo: dado un modelo de computación, clasificar los problemas abstractos según su resolubilidad mecánica en el mismo (**decidibles**/**indecidibles**).

Protocolo de resolución mecánica de un problema abstracto:

- **Diseño**: describir un procedimiento mecánico que resuelve el problema.
- **Verificación formal**: demostrar que el procedimiento mecánico descrito resuelve cada instancia del problema.
- **Análisis**: calcular los recursos computacionales necesarios para obtener la solución de cada instancia (en **función** de su **tamaño**).

Necesidad de verificar procedimientos mecánicos

El apagón de New York (1965).

Necesidad de verificar procedimientos mecánicos

El programa AMADEUS para facturación en aeropuertos (2017).

Verificación formal de procedimientos mecánicos

Sea X un problema abstracto tal que para cada $b \in I_X$ existe una única solución asociada.

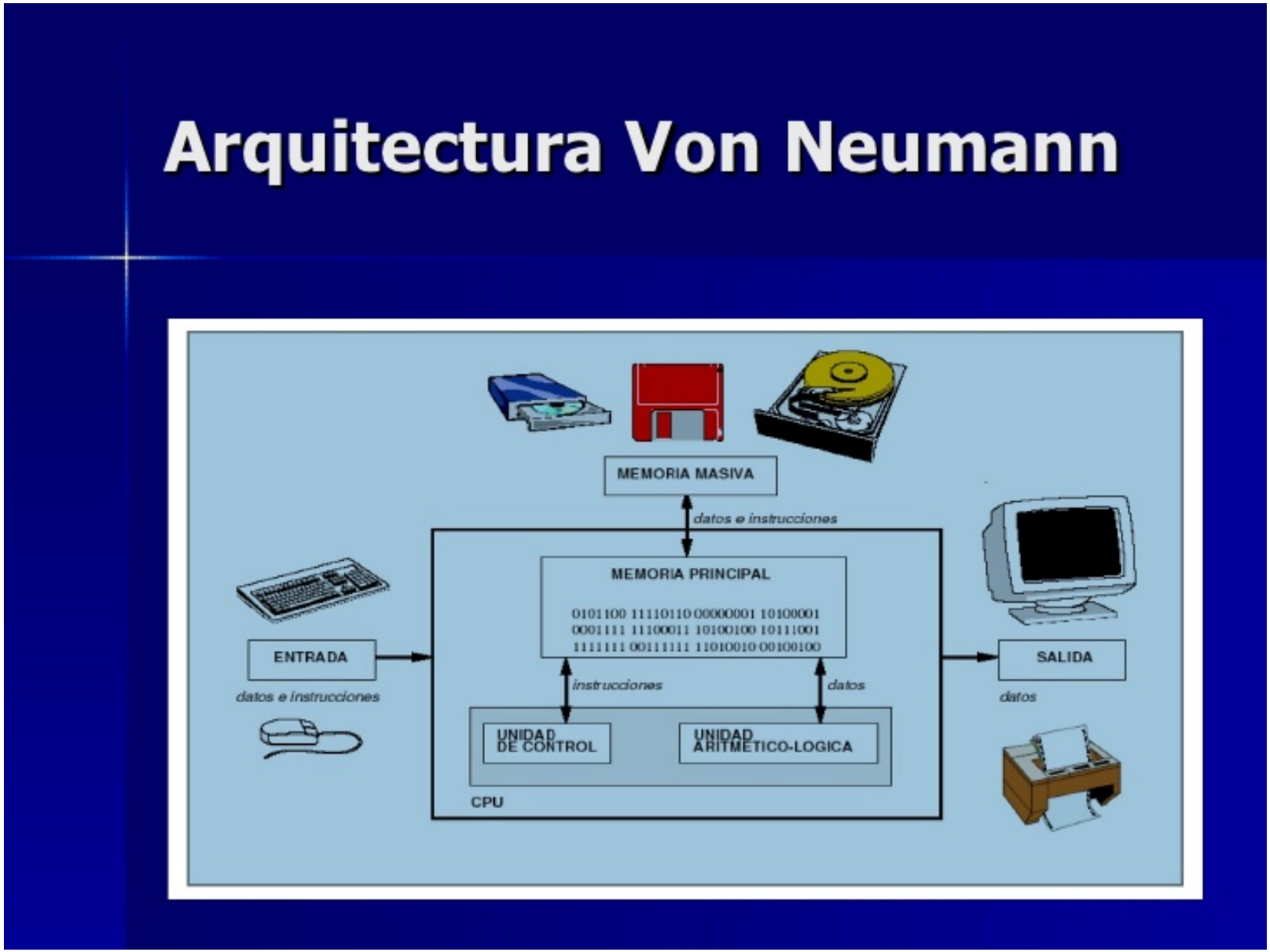
Sea $\mathcal{A}$ un procedimiento mecánico diseñado para resolver el problema X.

$\mathcal{A}$ está caracterizado por una función $F_{\mathcal{A}}$ cuyo dominio (entradas del procedimiento) es el conjunto de instancias, I_X, del problema.

Para verificar formalmente dicho procedimiento mecánico distinguiremos dos casos:

- I_X es un conjunto **finito**: basta comprobar que para cada $b \in I_X$, el valor $F_{\mathcal{A}}(b)$ coincide con $f_X(b)$.
- I_X es un conjunto **infinito**: se realizan dos pasos.
  - **Corrección parcial:** Si $b \in I_X$ y el procedimiento mecánico $\mathcal{A}$ **para** sobre b, entonces $F_{\mathcal{A}}(b) = f_X(b)$.
  - **Test de parada:** El procedimiento mecánico **para** sobre cualquier $b \in I_X$.

Máquinas reales de propósito general



J. von Neumann, J. P. Eckert y J. W. Mauchly (1945)

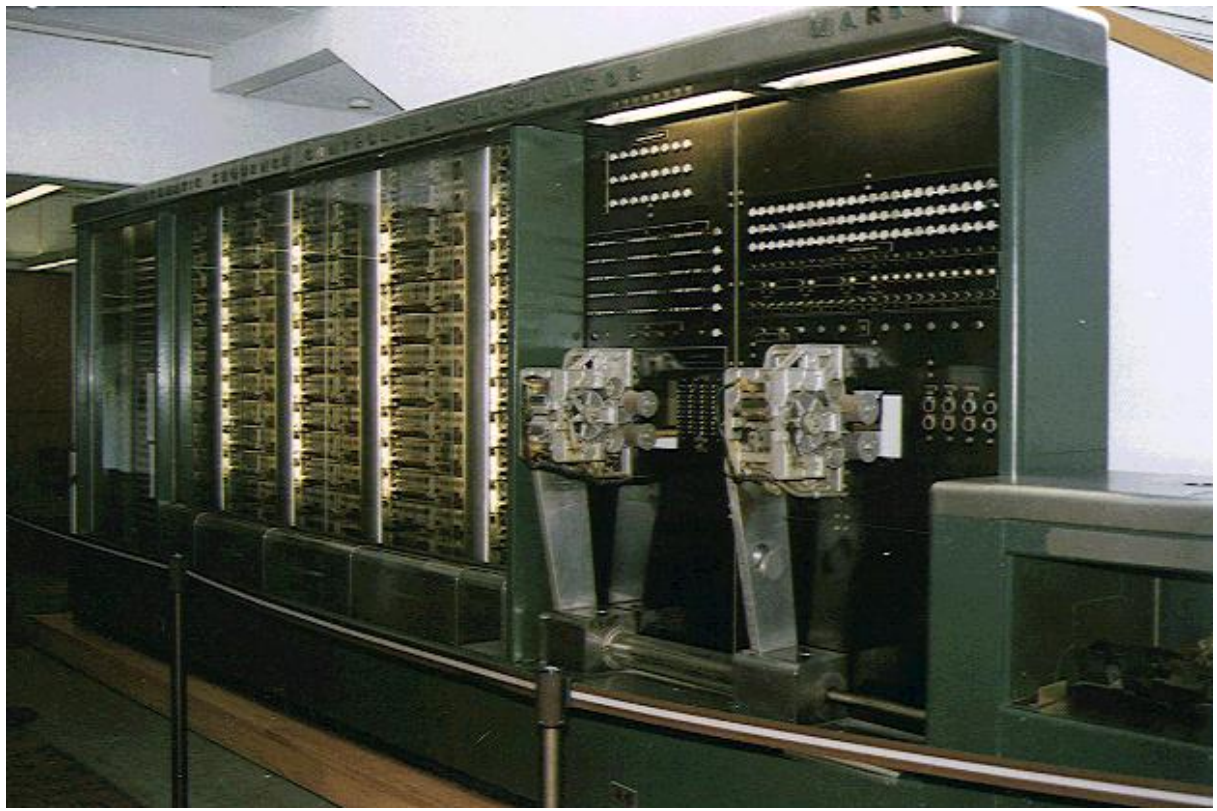
Máquinas reales de propósito general

1940's: Aparición de los primeros ordenadores (máquinas de propósito general).

* Implementación práctica de la arquitectura de J. von Neumann.

Primer **ordenador electromecánico**: **Mark I**, H. Aiken, 1944.

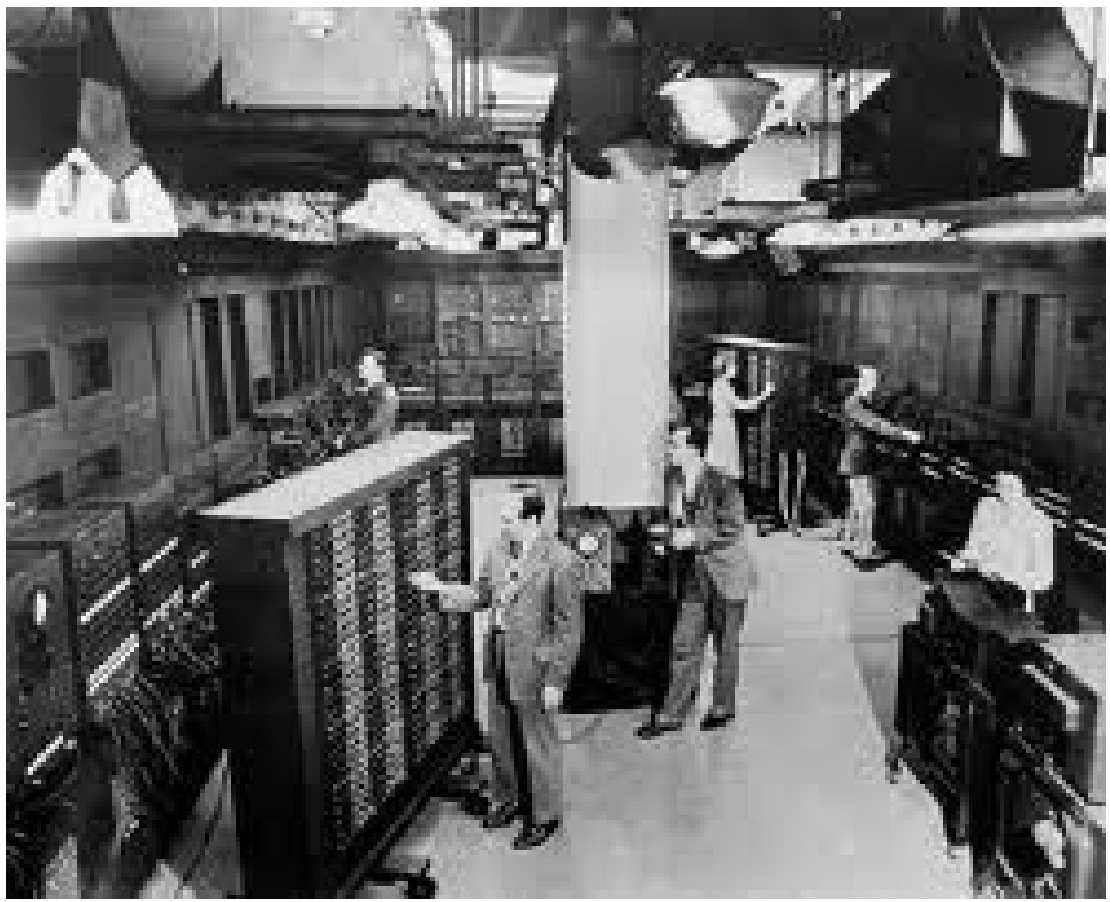
(760.000 ruedas + 800 kms. de cable + 5.000 Kgs. de peso + Superficie de más de 17 m^2)



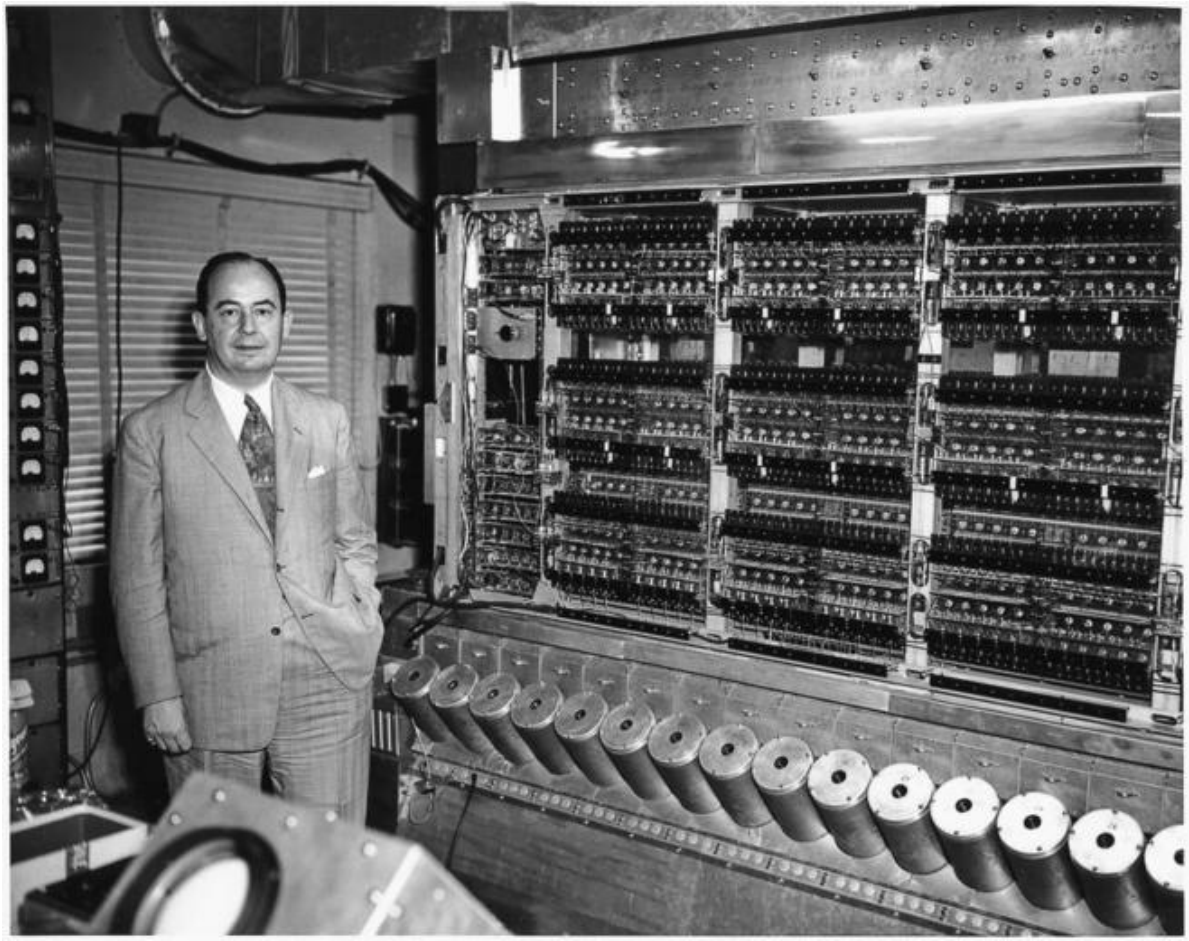
Máquinas reales de propósito general

Primer **ordenador electrónico**: **Eniac**, J.W. Mauchcly y J.P. Eckert, 1945.

(17.468 válvulas + 6.000 interruptores manuales + 27.000 Kgs. de peso + Superficie de más de 160 m^2)



Máquinas reales de propósito general



John von Neumann y el ordenador MANIAC (1952)

(Basada en su arquitectura: simular las condiciones para detonar una bomba de hidrógeno)

Teoría de la Complejidad Computacional

Complejidad computacional "inherente" a un problema abstracto resoluble mecánicamente en un modelo universal.

- ⋆ Menor cantidad de **recursos computacionales** necesarios para resolver mecánicamente del problema.

Búsqueda de **procedimientos mecánicos óptimos** (**algoritmos óptimos**) que resuelven problemas abstractos.

¿Qué problemas resuelven las **máquinas reales** de manera **eficiente**?

Resolubilidad mecánica práctica de problemas abstractos.

Cuestión 2: Dado un problema abstracto, hallar un procedimiento mecánico que lo resuelva con la menor cantidad posible de recursos computacionales.

¿Cómo buscar procedimientos mecánicos óptimos?

Protocolo de búsqueda de un procedimiento mecánico óptimo que resuelve un cierto problema abstracto.

- Hallar una **cota inferior** de los recursos necesarios para ejecutar **cualquier** procedimiento mecánico que resuelva el problema.
- Hallar un procedimiento mecánico que resuelva el problema y que use una cantidad de recursos del orden exacto de esa cota.

Complejidad computacional inherente a un problema.

⋆ A veces, es imposible encontrar procedimientos mecánicos óptimos de ciertos problemas abstractos (**teorema de aceleración de Blum**).

Clases de complejidad

Necesidad de analizar la complejidad de problemas de manera "global":

- **Clases de complejidad**.

Ingredientes necesarios para definir una clase de complejidad:

- ⋆ Un **modelo** de computación (con un **modo** de calcular).
- ⋆ Una **medida** de **complejidad**.
- ⋆ Una **función** (o familia de funciones) total computable que proporcione una **cota superior** de los **recursos computacionales**.

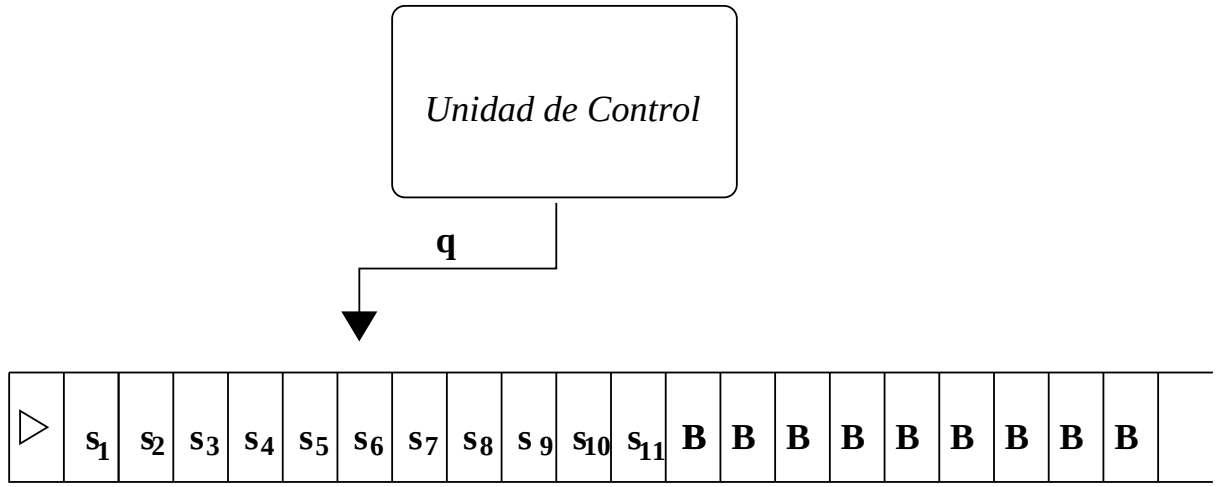
Máquina de Turing determinista: Sintaxis

Es una tupla, $M = (Q, \Sigma, q_0, F, \mathbf{B}, \triangleright, \delta)$:

- Q es un alfabeto finito (**estados**).
- Σ es un alfabeto finito (**de trabajo de la máquina**) tal que $\Sigma \cap Q = \emptyset$.
- $q_0 \in Q$ (**estado inicial**).
- $F = \{q_h, q_y, q_n\} \subseteq Q - \{q_0\}$ (**estados finales**, todos distintos entre sí).
- $\mathbf{B} \in \Sigma$ (**símbolo blanco**).
- $\triangleright \in \Sigma$ (**primer símbolo**) verificando que $\triangleright \neq \mathbf{B}$.
- δ es una aplicación (**función de transición**) de $(Q - F) \times \Sigma$ en $Q \times \Sigma \times \{\mathbf{L}, \mathbf{R}, \mathbf{S}\}$ tal que:
  - Si $\delta(q, \triangleright) = (q', s', x)$ entonces $s' = \triangleright \wedge x = \mathbf{R}$.

Informalmente, una máquina de Turing determinista M consta de:

* Una **cinta infinita**: celdas/casillas enumeradas, con primera casilla.
* Una **unidad de control** con un **cabezal lector/escritor**.



* En un instante determinado:
  - La cinta contiene sólo un número finito de símbolos distintos de **B**.
  - Cada celda contiene un símbolo del alfabeto de trabajo (el símbolo de la primera celda es $\triangleright$).
  - El cabezal inspecciona el contenido de una celda (**lee**), lo sustituye por un símbolo (**escribe**), y se desplaza a lo largo de la cinta (una celda a la izquierda, si $x = \mathbf{L}$, una a la derecha, si $x = \mathbf{R}$, o se queda en el mismo sitio, si $x = \mathbf{S}$)
  - La máquina se encuentra en un **estado** (elemento de Q).

Función de transición

¿Qué significa que la máquina M **ejecute** $\delta(q, s)$ en un cierto instante?

* Si $\delta(q, s) = (q', s', x)$ entonces la máquina M:
  * Pasa del estado **q** al estado **q'**.
  * Sustituye el símbolo **s** por el símbolo **s'**.
  * El cabezal se *desplaza* según el valor de $x \in \{$ **L**, **R**, **S** $\}$.
* La condición $\delta(q, \triangleright) = (q', s', x) \Rightarrow s' = \triangleright \wedge$ **x=R** significa lo siguiente:
  * si el cabezal está inspeccionando la primera celda, entonces el símbolo $\triangleright$ **no se modifica** y el cabezal se desplazará una celda a la derecha.

Máquina de Turing determinista: Semántica

Una **configuración** (**descripción instantánea**) de una MTD, M, en un instante t es una terna ordenada (**q**, **w**, **u**) tal que $q \in Q, w \in \Sigma^*$ y $u \in \Sigma^*$.

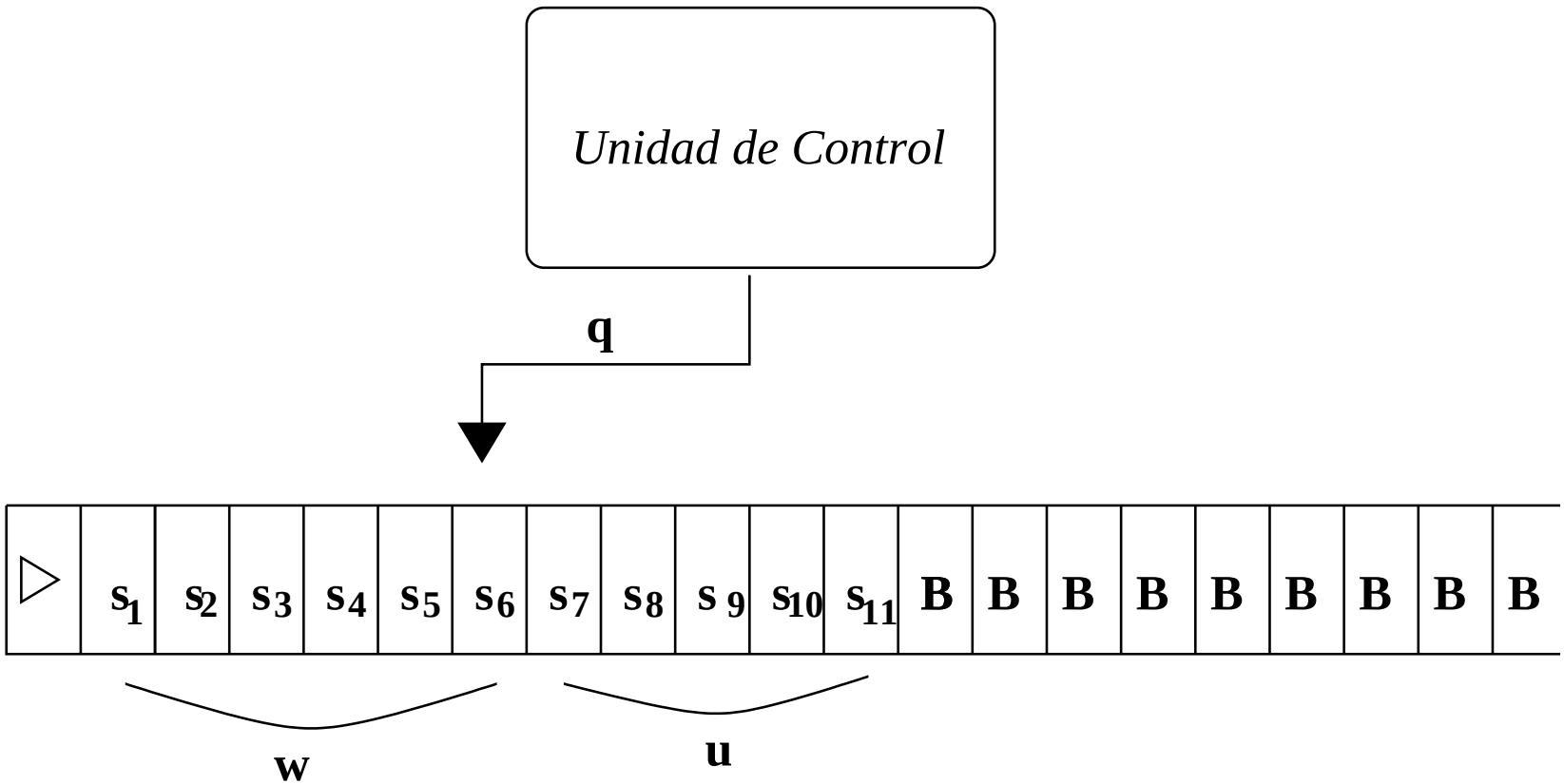
Una configuración (**q**, **w**, **u**) de una máquina MTD, M, en un instante t, **puede ser interpretada** como sigue:

- ⋆ **q** es el estado de la máquina M en el instante t.
- ⋆ $\triangleright$ **w u** es la palabra *escrita* en la cinta de la máquina M, en ese instante (el resto de símbolos de la cinta serán todos iguales a **B**).
- ⋆ Si **w** no es la cadena vacía, el cabezal de la máquina M estará **leyendo** el último símbolo de w, en ese instante. Caso contrario, estará leyendo el símbolo $\triangleright$ de la primera celda.

Genéricamente escribiremos: $\mathbf{C_t} = (\ \mathbf{q},\ \underbrace{w_1 \ldots w_{r-1}\ \overset{\downarrow}{w_r}}_{\mathbf{w}},\ \overbrace{u_1 u_2 \ldots u_s}^{\mathbf{u}})$.

Máquina de Turing determinista: Semántica

Configuración C_t = (**q**, **w**, **u**) de una MTD en un instante t.



Máquina de Turing determinista: Semántica

Tipos interesantes de configuraciones.

Sea $\mathbf{C_t} = (\mathbf{q}, \mathbf{w}, \mathbf{u})$ una configuración de una MTD en un instante t:

* $\mathbf{C_t}$ es la **configuración inicial** asociada al dato de entrada u, si $q = q_0$, w es la cadena vacía λ y $u \in (\Sigma - \{B\})^*$. Notaremos $\mathbf{C_t} = \mathbf{I_u}$.

* $\mathbf{C_t}$ es una **configuración de parada**, si $\mathbf{q} \in F$.

* $\mathbf{C_t}$ es una **configuración de aceptación**, si $\mathbf{q} = q_y$.

* $\mathbf{C_t}$ es una **configuración de rechazo**, si $\mathbf{q} = q_n$.

Máquina de Turing determinista: Semántica

Sea $\mathbf{C_t} = (\ \mathbf{q},\ \underbrace{w_1 \ldots w_{r-1}\ \overset{\downarrow}{w_r}}_{\mathbf{w}},\ \overbrace{u_1 u_2 \ldots u_s}^{\mathbf{u}})$ una configuración de una MTD, en un instante t, tal que **no sea de parada**.

La configuración siguiente, $\mathbf{C_{t+1}}$, de $\mathbf{C_t}$ se obtiene aplicando la función de transición δ al par ordenado (q, w_r).

* **Caso 1:** $\boxed{\delta(q, w_r) = (q', w_r', L)}$

  En este caso, $\mathbf{C_{t+1}} = (\mathbf{q'}, \underbrace{w_1 \ldots \overset{\downarrow}{w_{r-1}}}, \overbrace{\mathbf{w_r'} u_1 u_2 \ldots u_s})$.

* **Caso 2:** $\boxed{\delta(q, w_r) = (q', w_r', R)}$

  En este caso, $\mathbf{C_{t+1}} = (\mathbf{q'}, \underbrace{w_1 \ldots w_{r-1} \mathbf{w_r'}\ \overset{\downarrow}{u_1}}, \overbrace{u_2 \ldots u_s})$.

* **Caso 3:** $\boxed{\delta(q, w_r) = (q', w_r', S)}$

  En este caso, $\mathbf{C_{t+1}} = (\mathbf{q'}, \underbrace{w_1 \ldots w_{r-1}\ \overset{\downarrow}{\mathbf{w_r'}}}, \overbrace{u_1 u_2 \ldots u_s})$.

Máquina de Turing determinista: Semántica

Computación $\mathcal{C}$ de una MTD, M: sucesión finita o infinita ($\mathbf{C_0}$, $\mathbf{C_1}, \ldots, \mathbf{C_r}$) de configuraciones tal que

* $\mathbf{C_0}$ es una configuración inicial (asociada a un dato de entrada $u \in (\Sigma - \{B\})^*$).
* Para cada $t < r$, la configuración $\mathbf{C_{t+1}}$ es la configuración siguiente de $\mathbf{C_t}$.

La **computación** $\mathcal{C}$ es **de parada** si

- $r \in \mathbf{N}$ y la última configuración $\mathbf{C_r}$, es (q, λ, u'), siendo $q \in \{q_y, q_n, q_h)$ y $u' \in (\Sigma - \{B\})^*$.
  - Si $q = q_y$, diremos que M **acepta** u y escribiremos **M(u)**=`yes`.
  - Si $q = q_n$, diremos que M **rechaza** u y escribiremos **M(u)**=`no`.
  - Si $q = q_h$, diremos que el **resultado** de la computación es la cadena $u' \in (\Sigma - \{B\})^*$ y escribiremos **M(u)= u'**.

Si la computación $\mathcal{C}$ es de parada, diremos que M **para sobre u** ($M(u)\downarrow$).

Si la computación $\mathcal{C}$ no es de parada, diremos que M **no para sobre u** ($M(u)\uparrow$).

MTD de decisión

Una **MTD** es de **decisión** si sus estados finales son $\mathbf{q_y}$ y $\mathbf{q_n}$.

Sea L un lenguaje sobre el alfabeto $\Sigma - \{B\}$.

Si M es una MTD de decisión, diremos que:

- $\star$ M **decide el lenguaje** L **sii** para cada $u \in (\Sigma - \{B\})^*$ se verifica:
  - Si $u \in L$, entonces **M(u)**=`yes`; es decir, si el estado de la configuracion de parada es $\mathbf{q_y}$.
  - Si $u \notin L$, entonces **M(u)**=`no`; es decir, si el estado de la configuracion de parada es $\mathbf{q_n}$.

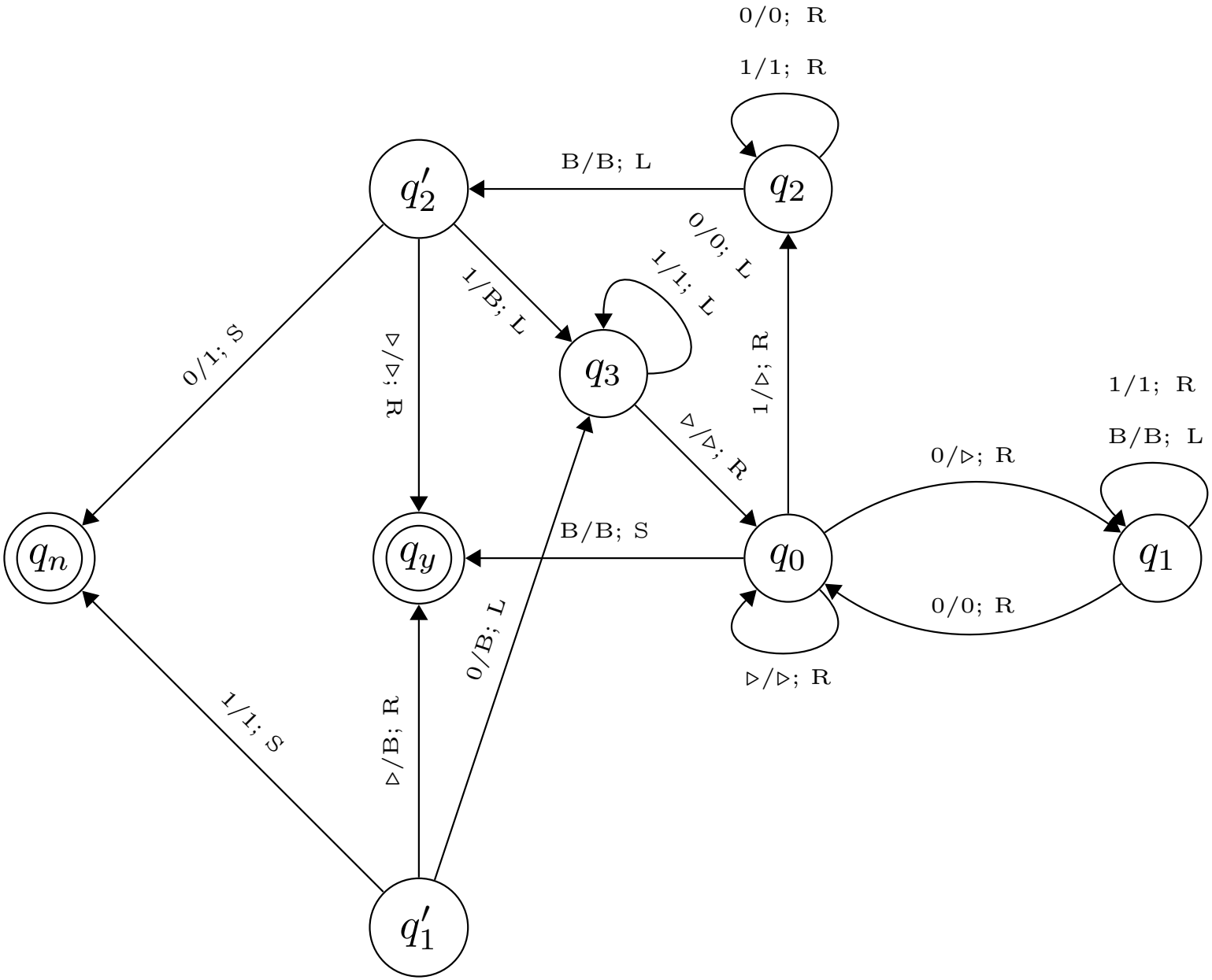
Ejemplo de una MTD de decisión

$Q = \{q_0, q_1, q_2, q_1', q_2', q_3, \mathbf{q_y}, \mathbf{q_n}\}$; $\Sigma = \{0, 1, B, \triangleright\}$

Función de transición:

$(Q - F) \times \Sigma$	$Q \times \Sigma \times \{L, R, S\}$
$(q_0, 0)$	$(q_1, \triangleright, R)$
$(q_0, 1)$	$(q_2, \triangleright, R)$
(q_0, B)	(q_y, B, S)
$(q_0, \triangleright)$	$(q_0, \triangleright, R)$
$(q_1, 0)$	$(q_1, 0, R)$
$(q_1, 1)$	$(q_1, 1, R)$
(q_1, B)	(q_1', B, L)
$(q_2, 0)$	$(q_2, 0, R)$
$(q_2, 1)$	$(q_2, 1, R)$
(q_2, B)	(q_2', B, L)
$(q_1', 0)$	(q_3, B, L)
$(q_1', 1)$	$(q_n, 1, S)$
$(q_1', \triangleright)$	(q_y, B, R)
$(q_2', 0)$	$(q_n, 1, S)$
$(q_2', 1)$	(q_3, B, R)
$(q_2', \triangleright)$	$(q_y, \triangleright, R)$
$(q_3, 0)$	$(q_3, 0, L)$
$(q_3, 1)$	$(q_3, 1, L)$
$(q_3, \triangleright)$	$(q_0, \triangleright, R)$

Diagrama de una MTD de decisión



MTD que calcula una función

Sea f una **función** parcial f de $(\Sigma - \{B\})^*$ en Σ^*

Diremos que una **MTD**, M, **calcula** la **función** f si se verifican las condiciones siguientes:

- El único estado final de M es $\mathbf{q_h}$.
- Para cada $u \in (\Sigma - \{B\})^*$ se tiene que:
  - La máquina M con dato de entrada u para **sii** $u \in \text{dom}(f)$.
  - Si la máquina M con dato de entrada u para entonces $M(u) = f(u)$.

Ejemplo de una MTD que calcula una función

Un ejemplo: $Q = \{q_0, q_1, \mathbf{q_h}\}$; $\Sigma = \{0, 1, B, \triangleright\}$

Función de transición:

$(Q - F) \times \Sigma$	$Q \times \Sigma \times \{L, R, S\}$
$(q_0, 0)$	$(q_0, 0, R)$
$(q_0, 1)$	$(q_0, 1, R)$
(q_0, B)	(q_1, B, L)
$(q_0, \triangleright)$	$(q_0, \triangleright, R)$
$(q_1, 0)$	$(q_h, 1, S)$
$(q_1, 1)$	$(q_1, 0, L)$
(q_1, B)	(q_1, B, S)
$(q_1, \triangleright)$	$(q_h, \triangleright, R)$

Hallemos $M(01)$

1. $\overset{q_0}{\triangleright}\ 0\ 1\ B.$
2. $\triangleright\ \overset{q_0}{0}\ 1\ B.$
3. $\triangleright\ 0\ \overset{q_0}{1}\ B.$
4. $\triangleright\ 0\ 1\ \overset{q_0}{B}.$
5. $\triangleright\ 0\ \overset{q_1}{1}\ B.$
6. $\triangleright\ \overset{q_1}{0}\ 0\ B.$
7. $\triangleright\ \overset{q_h}{1}\ 0\ B.$

MTD's que resuelven problemas

¿Cómo **resolver un problema** X?

* Sea X un problema de decisión.
  * Una **MTD resuelve** X **sii decide** el lenguaje asociado a X.
* Sea X un problema caracterizado por una función total $f_X : L_X \rightarrow \Sigma^*$.
  * Una **MTD resuelve** X **sii calcula** la función f_X.

Máquina de Turing no determinista

Es una tupla, $M = (Q, \Sigma, q_0, F, \mathbf{B}, \triangleright, \delta)$, en donde:

- Q es un alfabeto finito (**estados**).
- Σ es un alfabeto finito (**de trabajo de la máquina**) tal que $Q \cap \Sigma = \emptyset$.
- $q_0 \in Q$ (**estado inicial**).
- $F = \{q_h, q_y, q_n\} \subseteq Q - \{q_0\}$ (**estados finales**, todos distintos entre sí).
- $B \in \Sigma$ (**símbolo blanco**).
- $\triangleright \in \Sigma$ (**primer símbolo**), $\triangleright \neq \mathbf{B}$.
- δ es una aplicación (**función de transición**) de $(Q - F) \times \Sigma$ en $\mathbf{P}(Q \times \Sigma \times \{\mathbf{L}, \mathbf{R}, \mathbf{S}\})$ tal que:
  - Si $(q', s', x) \in \delta(q, \triangleright)$ entonces $s' = \triangleright \wedge x = \mathbf{R}$.

MTD versus MTND

Es una tupla $M = (Q, \Sigma, q_0, F, \mathbf{B}, \triangleright, \delta)$:

Máquina de Turing determinista

- $\star$ Q es un conjunto finito no vacío (*estados*).
- $\star$ Σ es un alfabeto (*de la máquina*), $\Sigma \cap Q = \emptyset$.
- $\star$ $q_0 \in Q$ (*estado inicial*).
- $\star$ $F = \{q_h, q_y, q_n\} \subseteq Q - \{q_0\}$ (*estados finales*).
- $\star$ $\mathbf{B} \in \Sigma$ (*símbolo blanco*).
- $\star$ $\triangleright \in \Sigma$ (*primer símbolo*), $\triangleright \neq \mathbf{B}$.
- $\delta : (Q - F) \times \Sigma \to \mathbf{Q} \times \mathbf{\Sigma} \times \{\mathbf{L}, \mathbf{R}, \mathbf{S}\}$

Máquina de Turing no determinista

- $\star$ Q es un conjunto finito no vacío (*estados*).
- $\star$ Σ es un alfabeto (*de la máquina*), $Q \cap \Sigma = \emptyset$.
- $\star$ $q_0 \in Q$ (*estado inicial*).
- $\star$ $F = \{q_h, q_y, q_n\} \subseteq Q - \{q_0\}$ (*estados finales*).
- $\star$ $\mathbf{B} \in \Sigma$ (*símbolo blanco*).
- $\star$ $\triangleright \in \Sigma$ (*primer símbolo*), $\triangleright \neq \mathbf{B}$.
- $\delta : (Q - F) \times \Sigma \to \mathbf{P}(\mathbf{Q} \times \mathbf{\Sigma} \times \{\mathbf{L}, \mathbf{R}, \mathbf{S}\})$

Por ejemplo: en una MTD podría ser $\delta(q, s) = (q_1, s_1, \mathbf{L})$, mientras que en una MTND podría ser $\delta(q, s) = \{(q_1, s_1, \mathbf{S}), (q_2, s_2, \mathbf{R}), (q_3, s_3, \mathbf{L}), (q_4, s_4, \mathbf{R})\}$

Máquina de Turing no determinista

Para MTNDs, los conceptos de **configuración** en un instante t y **computacion** se definen de manera similar a como se hizo en las MTDs.

Observaciones importantes:

- Una configuración de una MTND que no sea de parada, **puede tener** más de una configuración siguiente.
- Dada una MTND, M, y un dato de entrada u pueden existir varias computaciones de M sobre u.
- Toda MTD es, en particular, una MTND.
- Una MTND de decisión es aquella cuyos estados finales son $\mathbf{q_y}$ y $\mathbf{q_n}$.

Resolubilidad de problemas de decisión por MTDs

La clave: concepto de **ACEPTACIÓN** de una instancia del problema por una MTD.

Una **MTD**, M, **acepta** un dato de entrada **a** si **la computación** de M con entrada **a** es de parada y devuelve **Sí**.

Una **MTD**, M, **rechaza** un dato de entrada **a** si **la computación** de M con entrada **a** es de parada y devuelve **No**.

Una **MTD**, M, **resuelve un problema de decisión** $X = (\Sigma_X, I_X, s_X)$ si para cada instancia $a \in I_X$ se tiene:

- $s_X(a) = \{1\}$ **sii** M **acepta** la instancia **a**.
- $s_X(a) = \{0\}$ **sii** M **rechaza** la instancia **a**.

Resolubilidad de problemas de decisión por MTNDs

La clave: concepto de **ACEPTACIÓN** de una instancia del problema por una MTND.

Una **MTND**, M, **acepta** un dato de entrada **a** **sii** **EXISTE, AL MENOS, una computación** de M con entrada **a** que es de parada y devuelve **Sí**.

Un par de reflexiones:

- ⋆ ¿Qué significaría que una MTND **no acepta** un dato de entrada **a**?
- ⋆ ¿Se podría establecer "mecánicamente" que una MTND **no acepta** un dato **a**?

Una **MTND**, M, **resuelve un problema de decisión** $X = (\Sigma_X, I_X, s_X)$ si para cada instancia $a \in I_X$ se tiene:

- $s_X(a) = \{1\}$ **sii** M **acepta** **a**.

Tratabilidad de problemas abstractos

Procedimiento mecánico eficiente: la cantidad de recursos necesarios para su ejecución está acotada por un **polinomio** (en el tamaño del dato de entrada).

- ¿Por qué se consideran los polinomios para establecer la frontera?
  - ⋆ Es un conjunto de funciones estables por suma y producto.
  - ⋆ Tienen un crecimiento "**moderado**".

Problema **tratable**: resoluble por una **máquina de Turing determinista** que trabaja en tiempo polinomial.

Teoría de la Complejidad Computacional: clasificar los problemas abstractos según sean o no resolubles eficientemente (problemas **tratables** e **intratables**).

Problemas abstractos ...

- Problema **tratable**:
  - ⋆ Existe **ALGÚN** procedimiento mecánico que resuelve el problema y proporciona (**en tiempo razonable**) soluciones para entradas de **tamaño grande**.
- Problema **intratable**:
  - ⋆ **NINGÚN** procedimiento mecánico que resuelve el problema proporciona (**en tiempo razonable**) soluciones para entradas de **tamaño grande**.
- Problema **presuntamente intratable**:
  - ⋆ **NINGÚN** procedimiento mecánico **CONOCIDO** que resuelve el problema proporciona (**en tiempo razonable**) soluciones para entradas de **tamaño grande**.

Problemas tratables

Son los **más simples**, desde el punto de vista de la complejidad computacional.

* **Resolubilidad mecánica en términos prácticos**: **tratabilidad**.

Polinomial = **Razonable** **Exponencial** = **No razonable**

	10	50	100	300	1000
5n	50	250	500	1.500	5.000
$\mathbf{n^2}$	100	2.500	10.000	90.000	10^6
$\mathbf{n^3}$	1.000	125.000	10^6	$27 \cdot 10^6$	10^9
$\mathbf{2^n}$	1.024	16 dígitos	31 dígitos	91 dígitos	302 dígitos
n!	7 dígitos	65 dígitos	161 dígitos	623 dígitos	inimaginable
$\mathbf{n^n}$	10 dígitos	85 dígitos	201 dígitos	744 dígitos	inimaginable

Nota 1: El número de microsegundos desde el BIG BANG tiene 24 dígitos.

Nota 2: El número de protones en el universo consta de 71 dígitos.

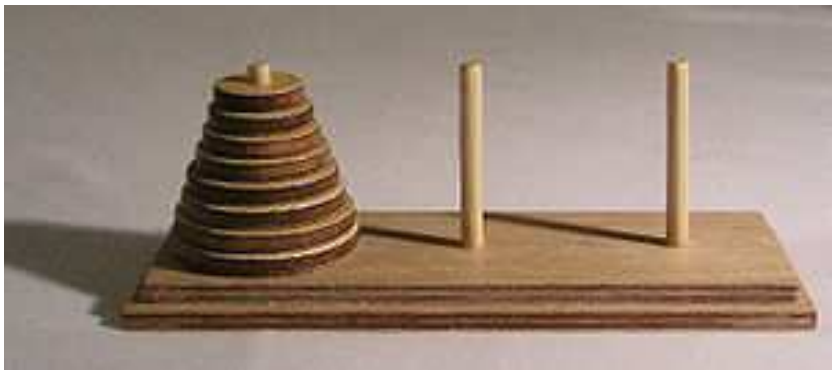
Un problema tratable: la multiplicación rusa

52	**27**	
26	54	
$\star$ 13	108	**108**
6	216	
$\star$ 3	432	**432**
$\star$ 1	864	**864**
52 $\times$ **27**	$=$	**1404**

¿Cuál es el coste en tiempo del algoritmo de la multiplicación rusa?

Un problema intratable: las Torres de Hanoi

La leyenda ... (Edouard Lucas d'Amiens, 1883)

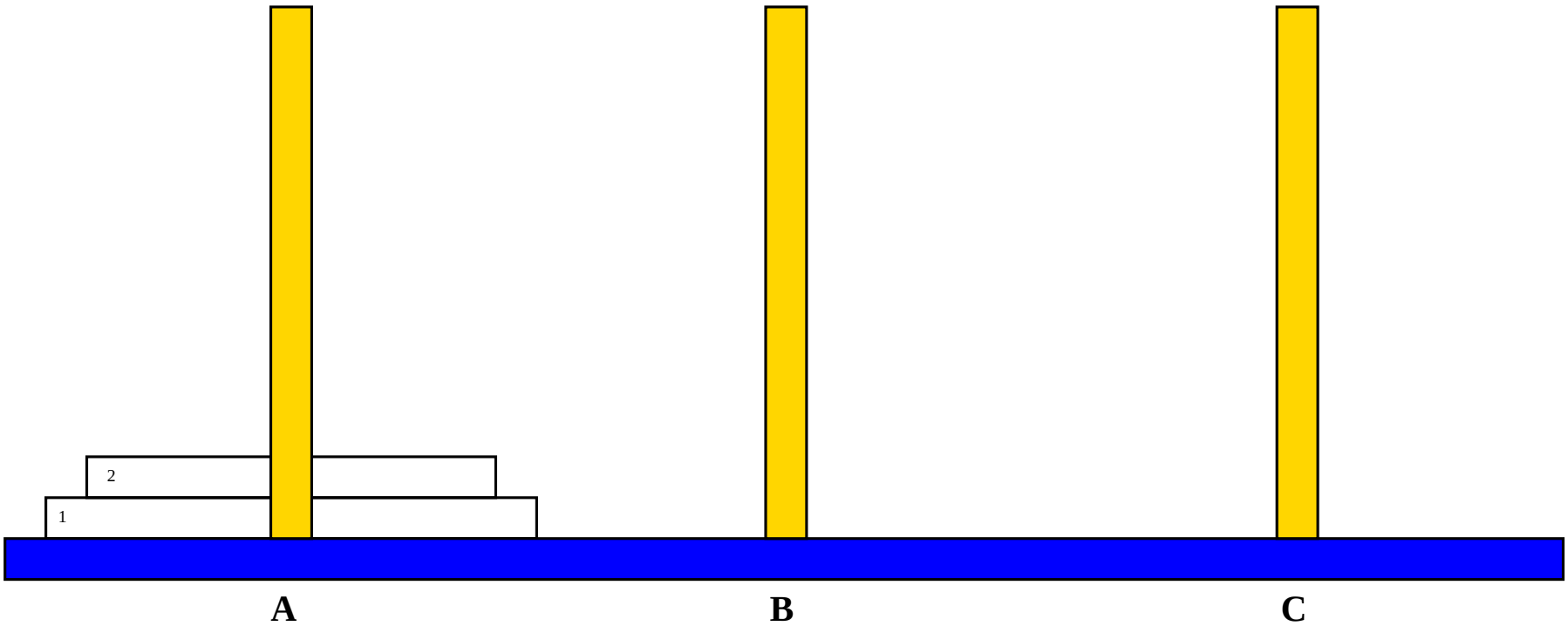


Un **movimiento**: trasladar un disco de una varilla a otra.

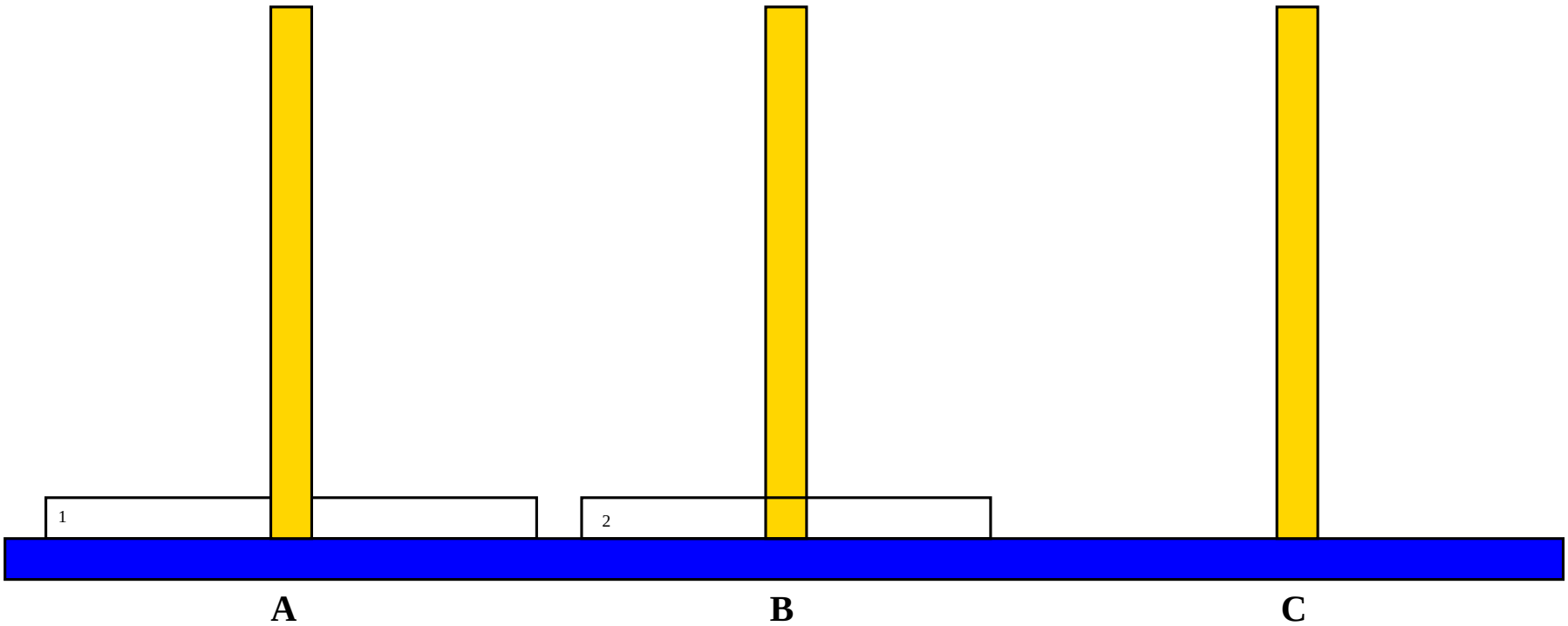
Movimiento válido: un disco **no se puede colocar** encima de otro de menor radio.

* Problema **concreto**: **64** discos.
* Problema **abstracto**: **n** discos.

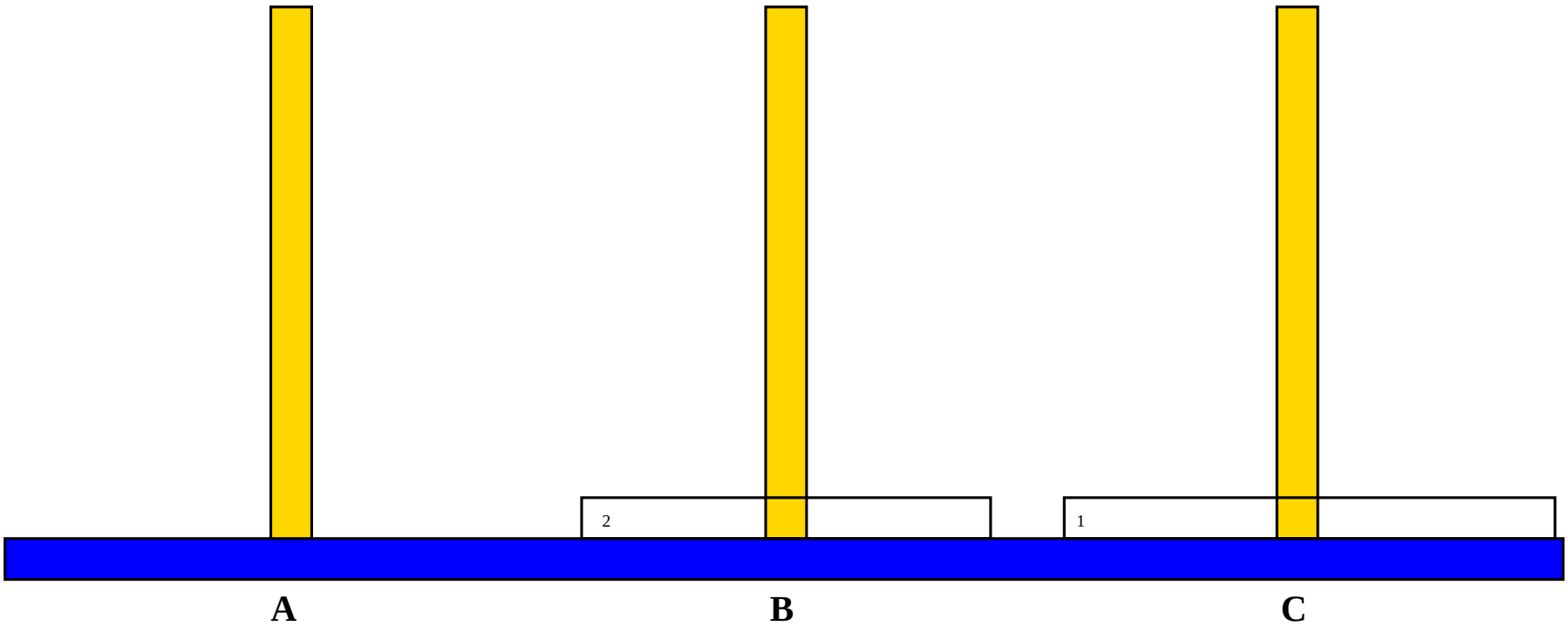
El problema de las Torres de Hanoi: 2 discos



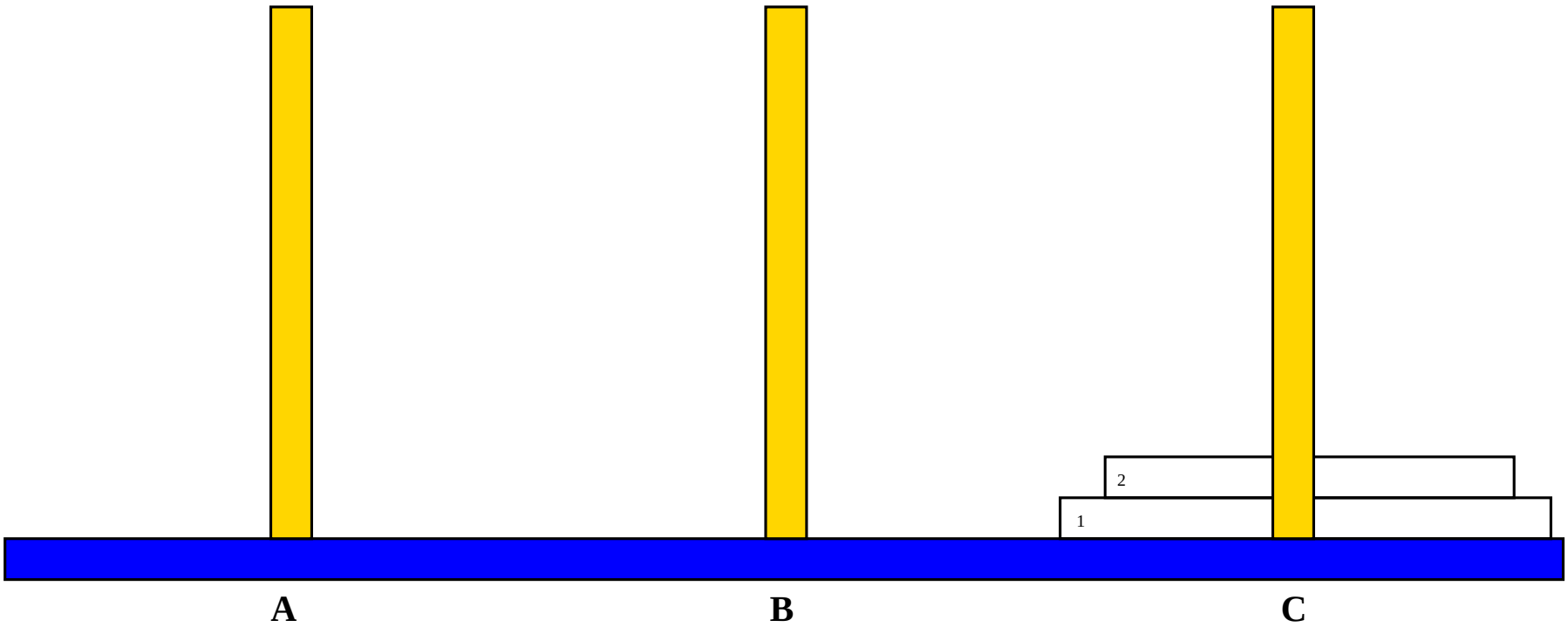
El problema de las Torres de Hanoi: 2 discos



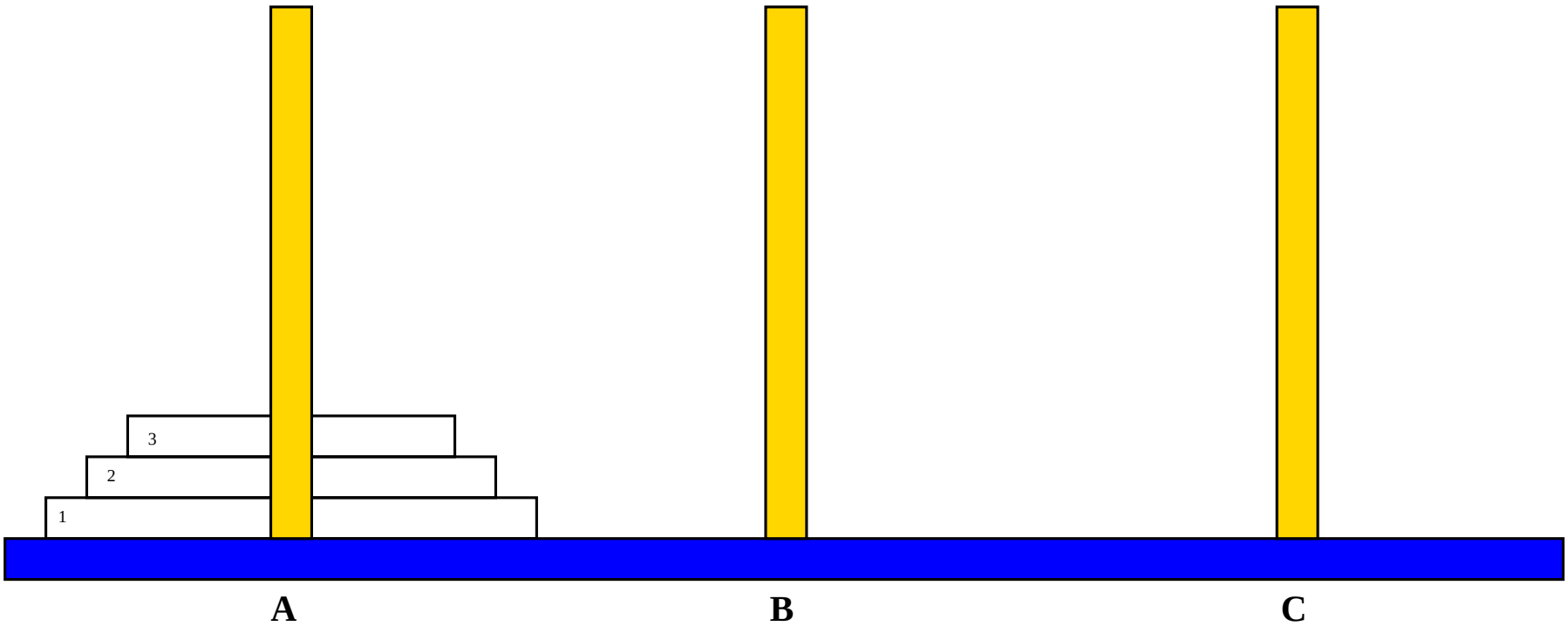
El problema de las Torres de Hanoi: 2 discos



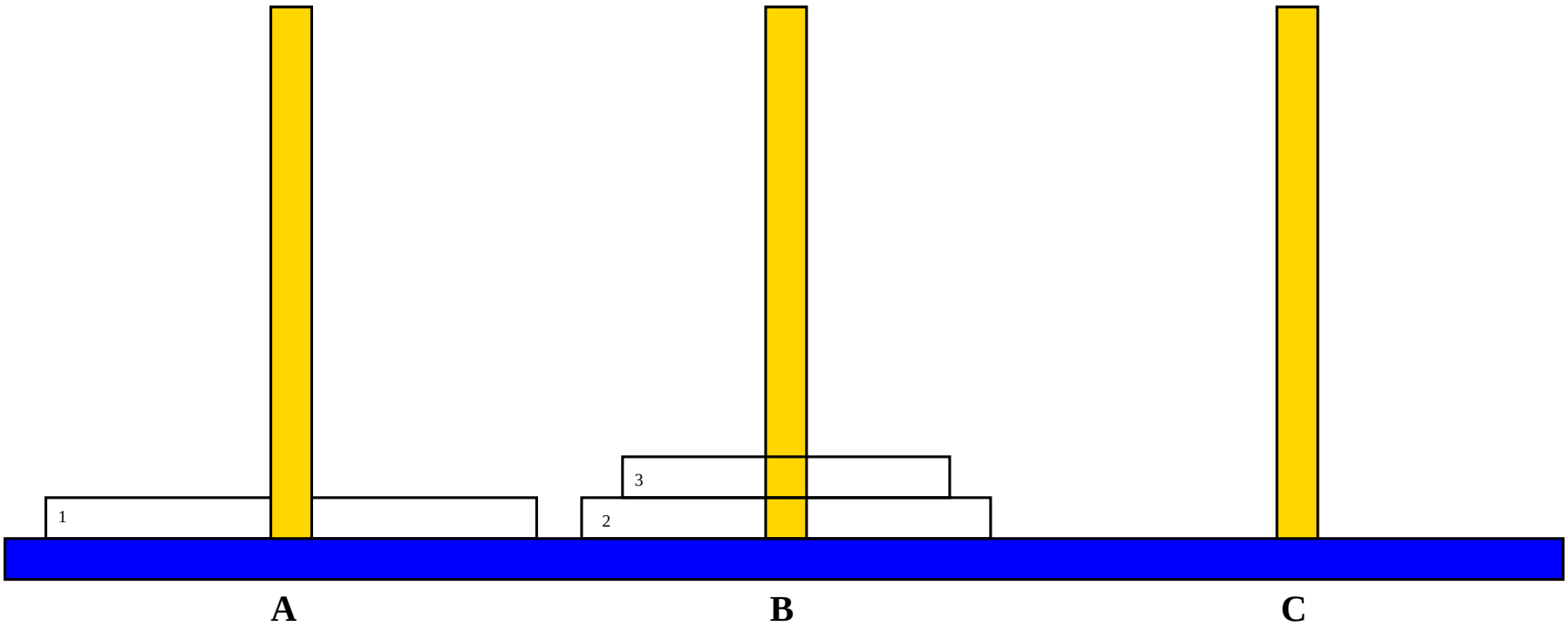
El problema de las Torres de Hanoi: 2 discos



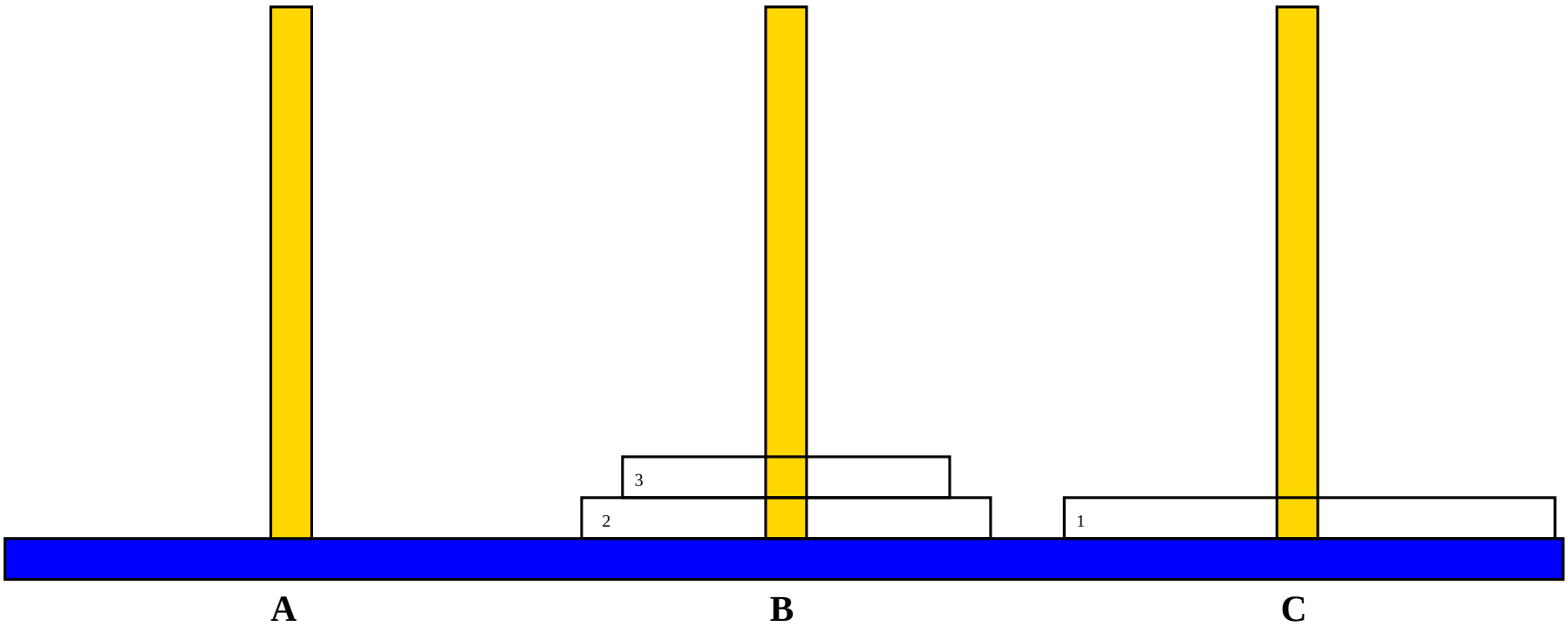
El problema de las Torres de Hanoi: 3 discos



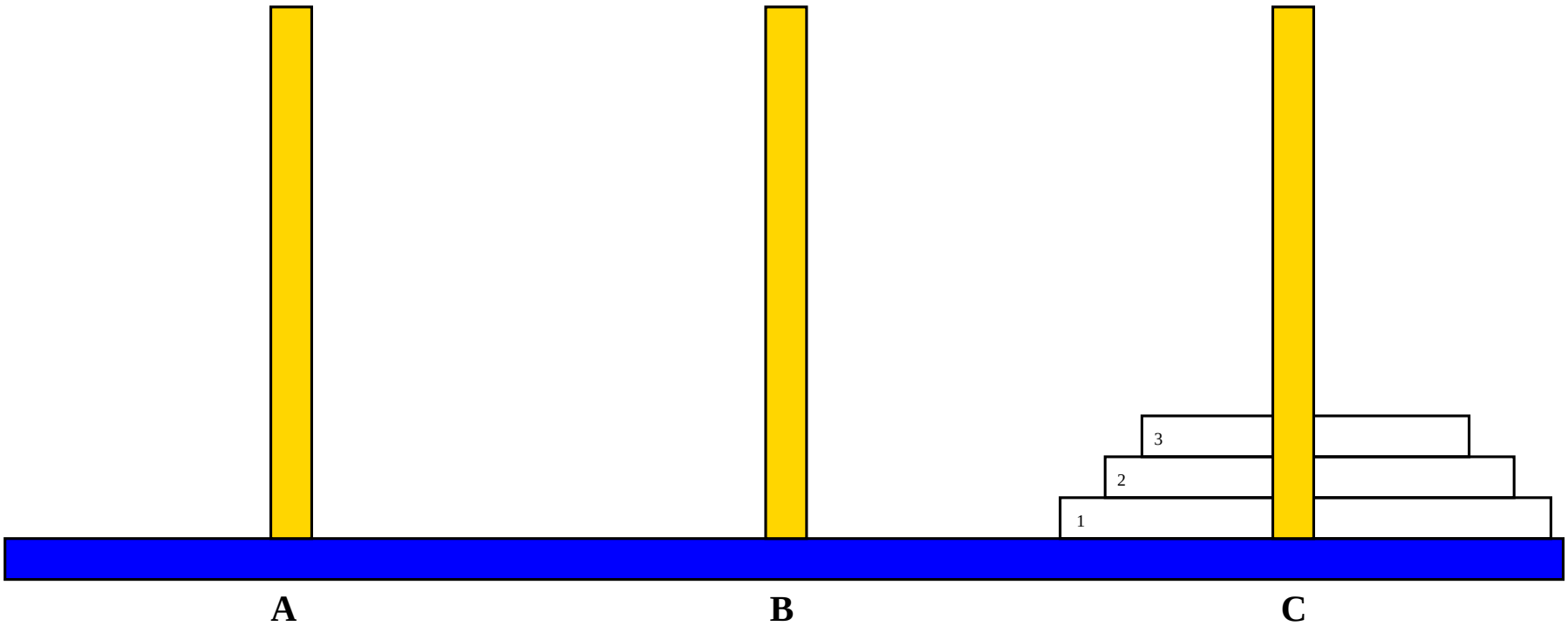
El problema de las Torres de Hanoi: 3 discos



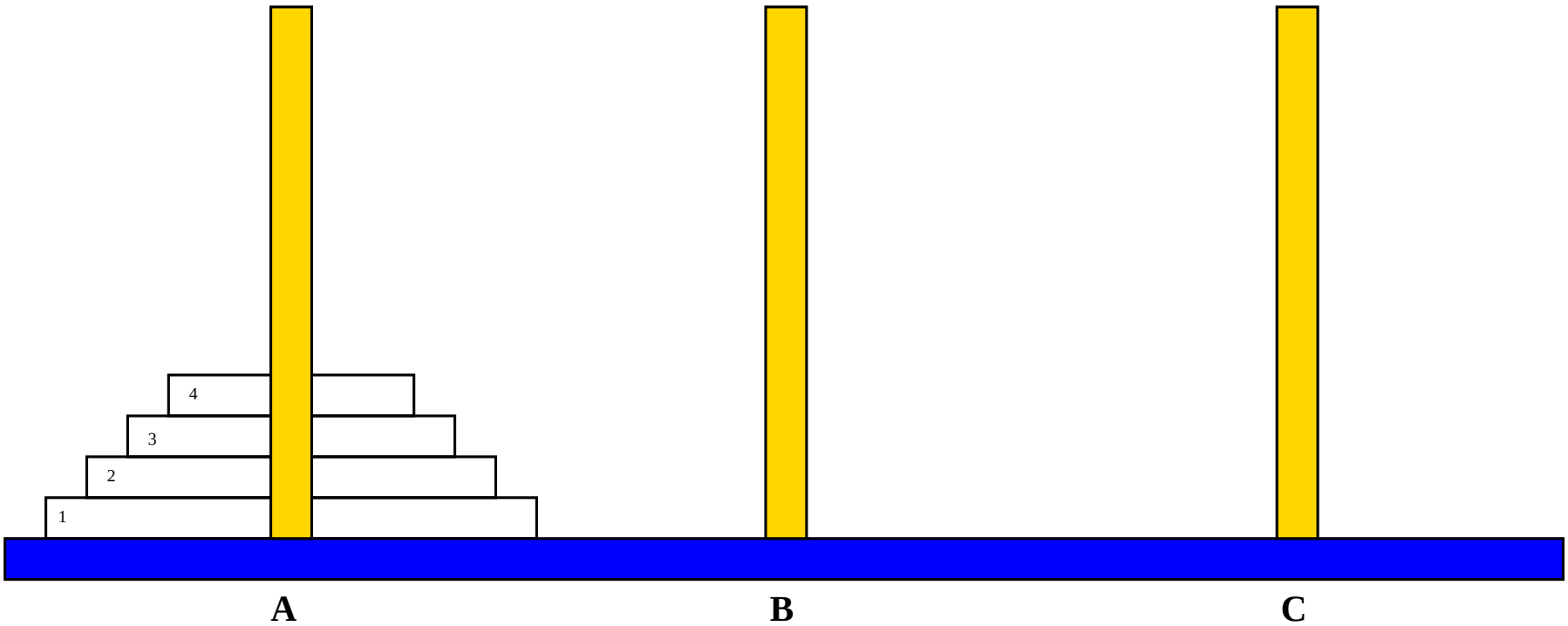
El problema de las Torres de Hanoi: 3 discos



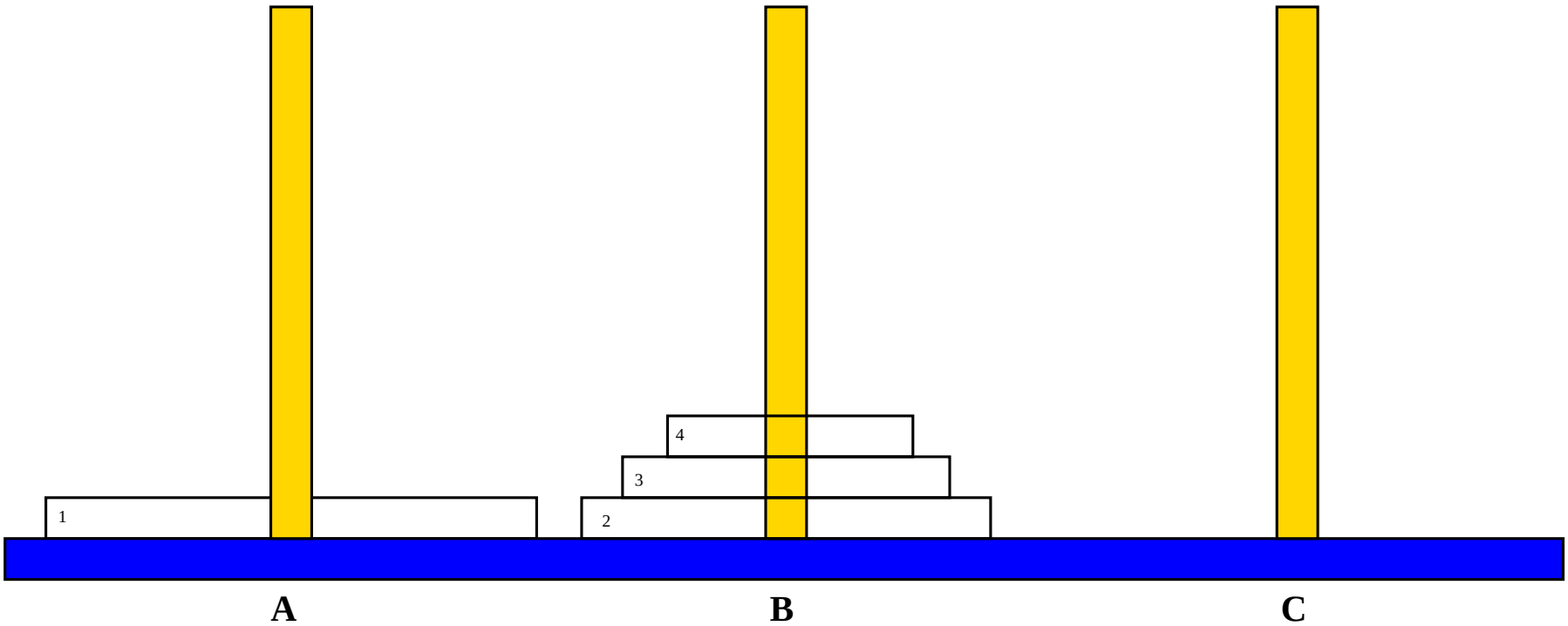
El problema de las Torres de Hanoi: 3 discos



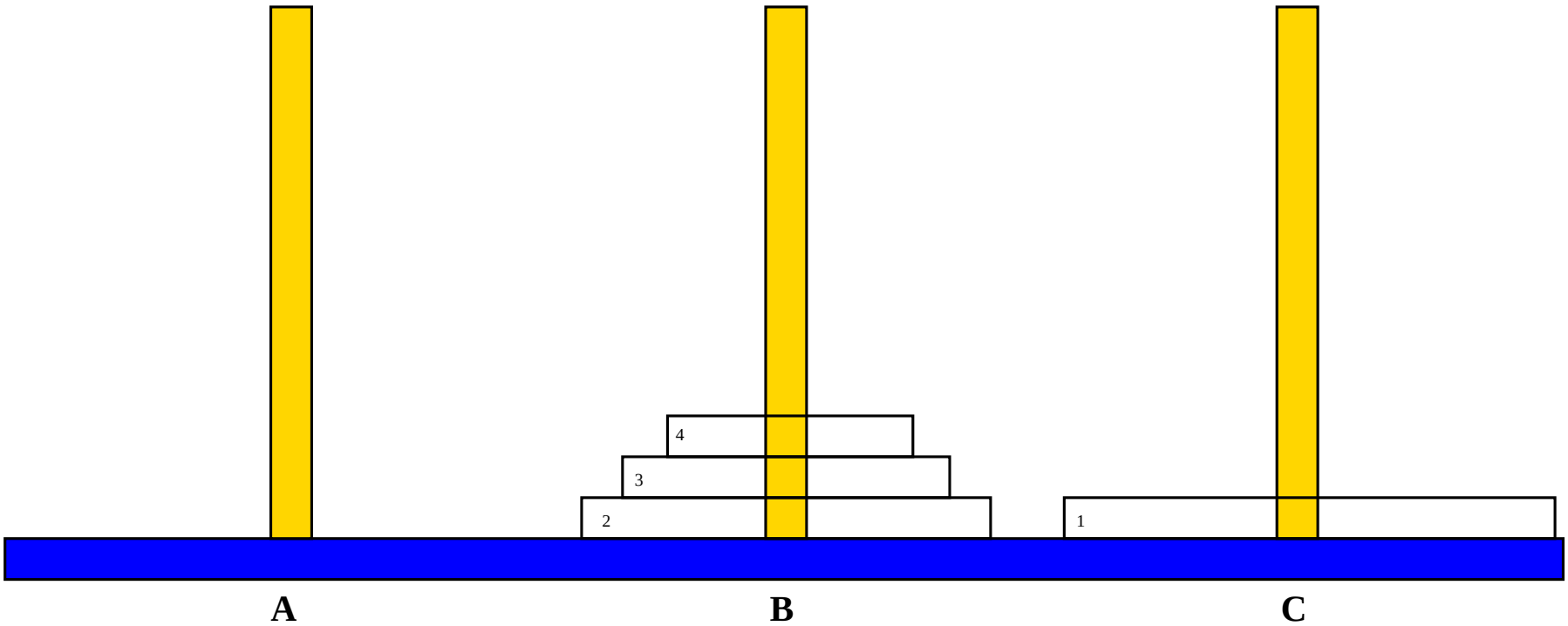
El problema de las Torres de Hanoi: 4 discos



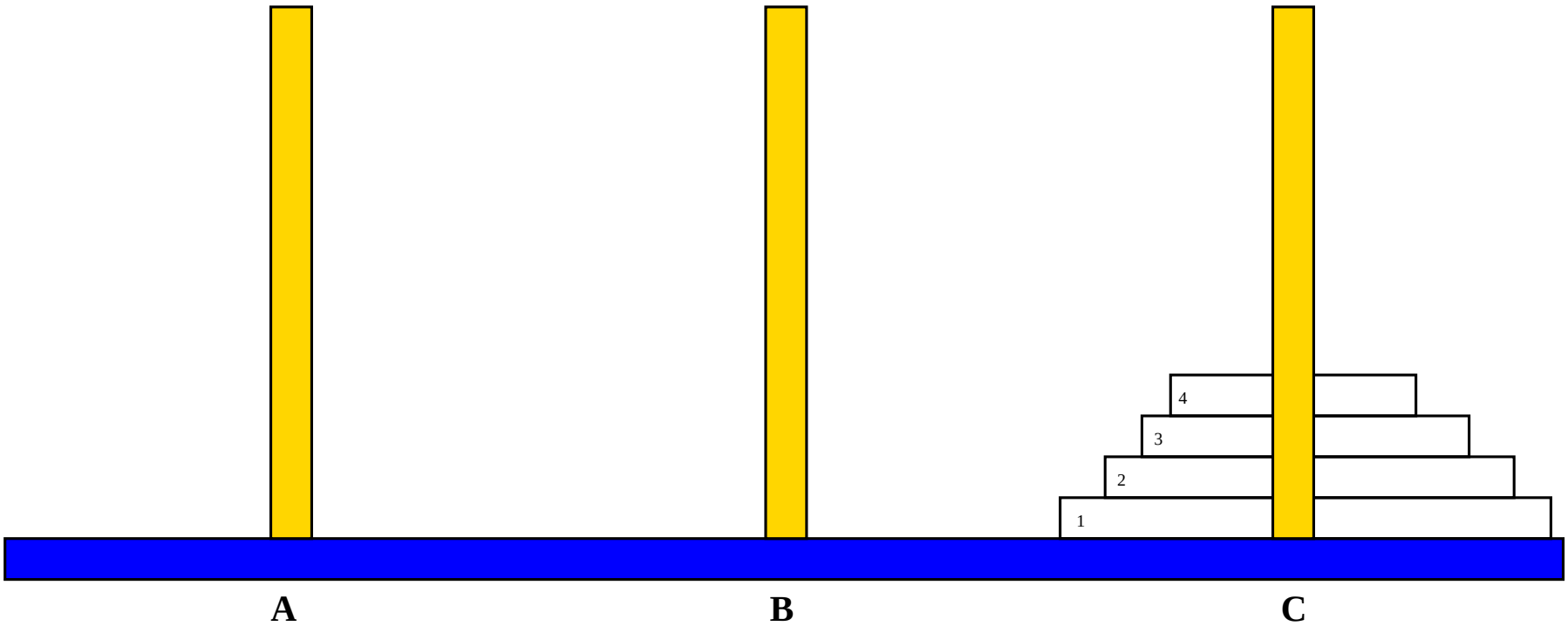
El problema de las Torres de Hanoi: 4 discos



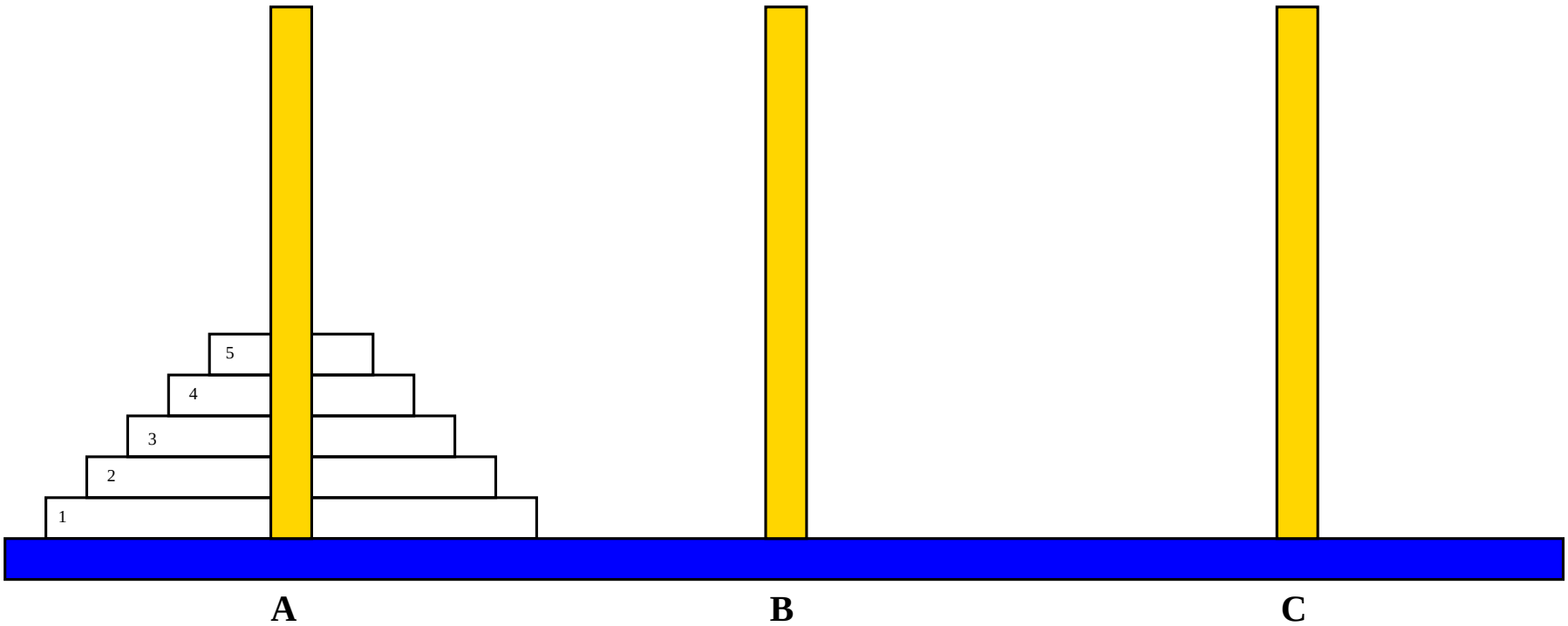
El problema de las Torres de Hanoi: 4 discos



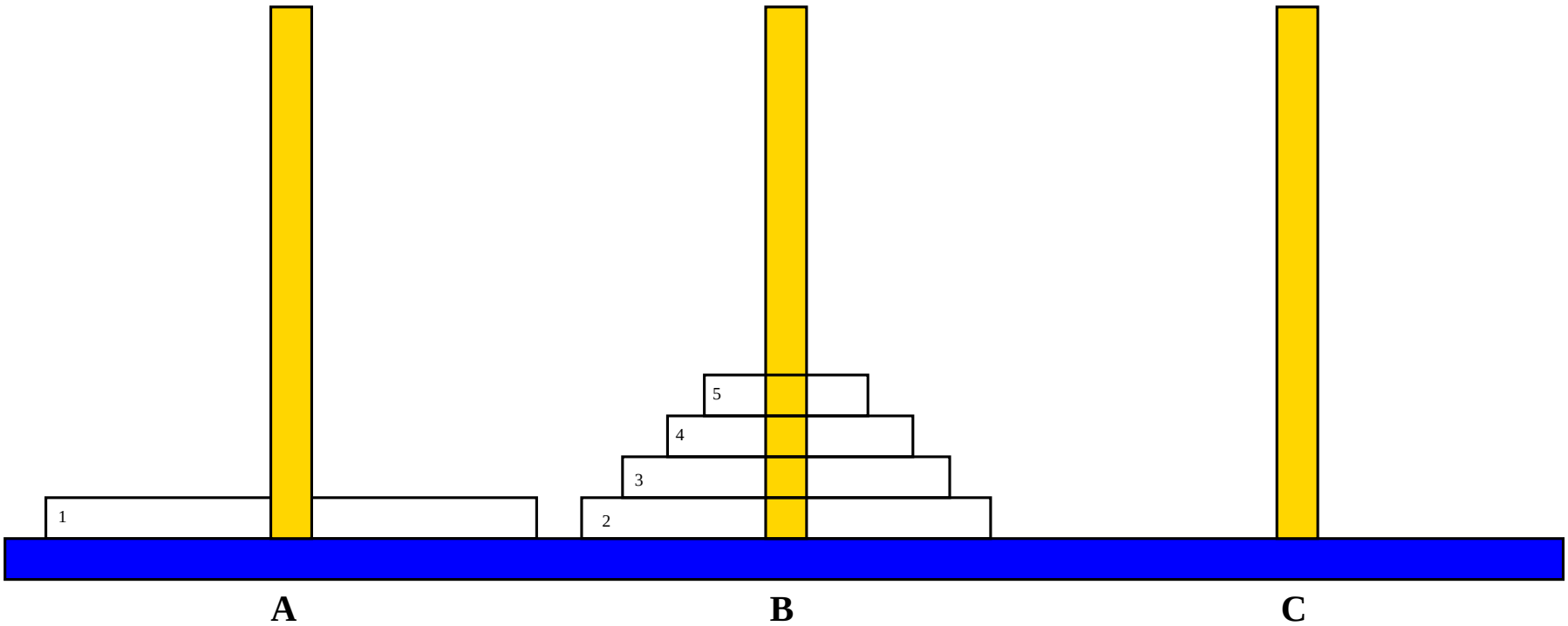
El problema de las Torres de Hanoi: 4 discos



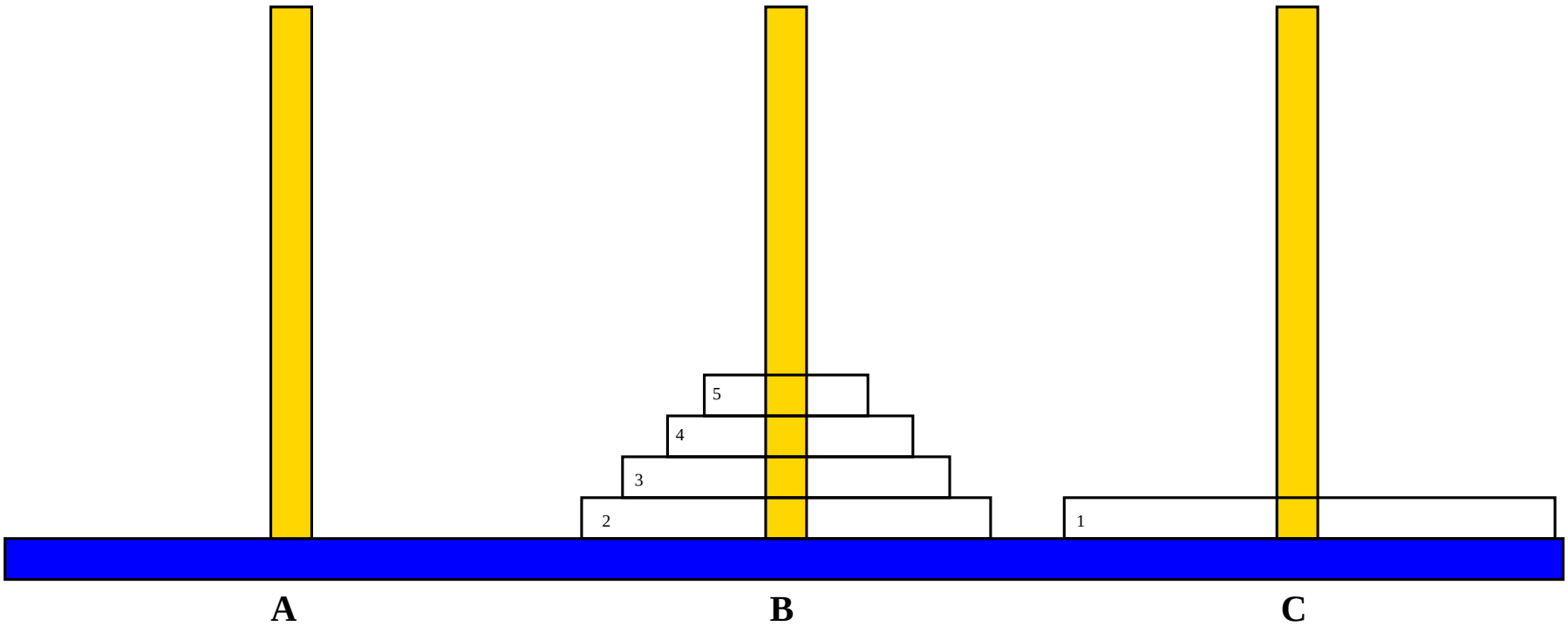
El problema de las Torres de Hanoi: 5 discos



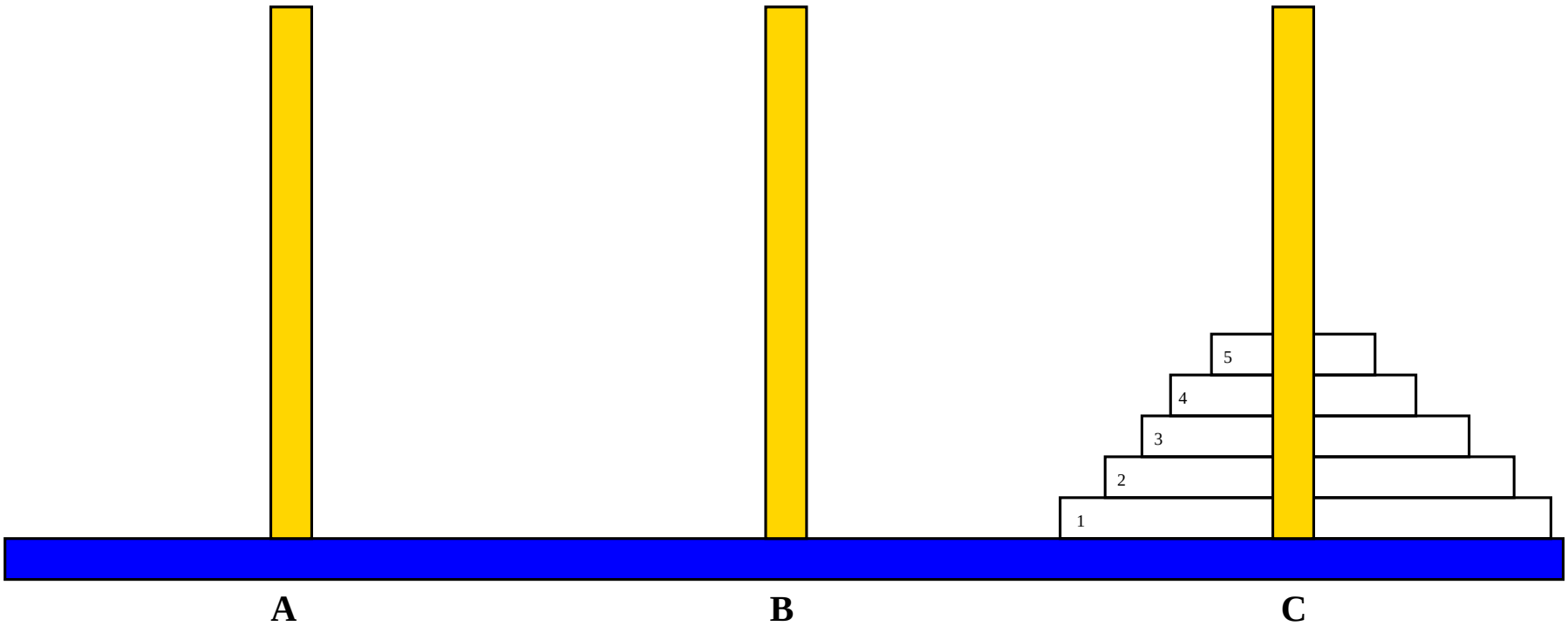
El problema de las Torres de Hanoi: 5 discos



El problema de las Torres de Hanoi: 5 discos



El problema de las Torres de Hanoi: 5 discos



El problema de las Torres de Hanoi

* **Diseño** (recursivo)

```
procedimiento Hanoi (n, 1, 3), con n > 0
        si n > 0 entonces
                Hanoi (n-1, 1, 2)
                mover disco de 1 a 3
                Hanoi (n-1, 2, 3)
```

* **Verificación** (inducción).

* **Análisis** (inducción):

$f(n)$ = número de movimientos para resolver el problema de Hanoi correspondiente a n discos.

$$\begin{cases} f(0) & = & 0 \\ f(n+1) & = & 2 \cdot f(n) + 1 \end{cases}$$

Se prueba que $f(n) = 2^n - 1$.

El problema de las Torres de Hanoi

Para resolver el problema con n discos

* Número de movimientos a realizar : $2^n - 1$.

Supongamos que:

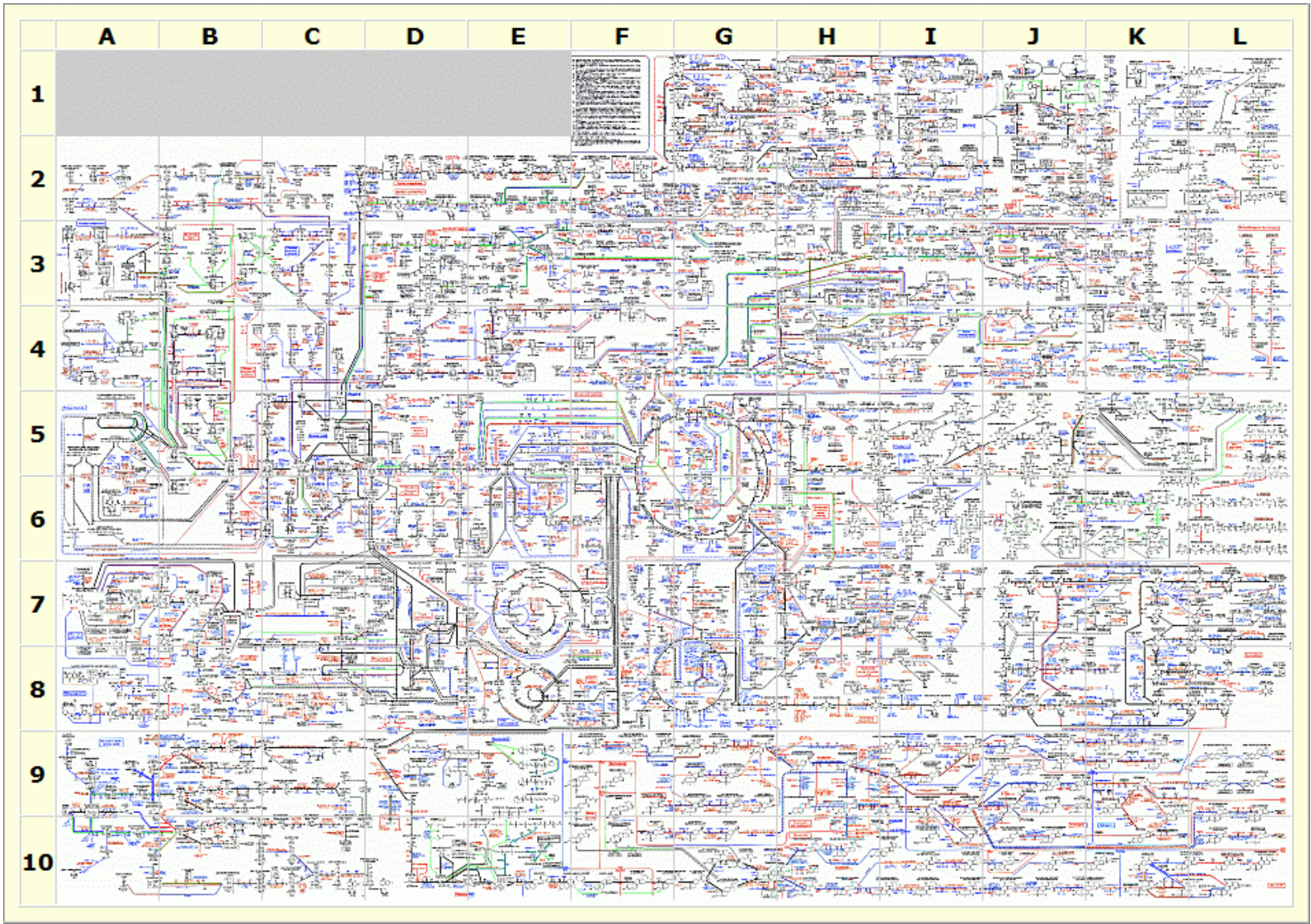
* Tenemos **64 discos**.
* Se tarda **un segundo** en mover un disco de una varilla a otra.
* Los monjes **trabajan las 24 horas** del día.

¿Cuánto tiempo se necesitará para resolver el problema "divino"?

* Aproximadamente ... **¡500 millones de años!**

Problemas presuntamente intratables

Problema del que se desconoce si es tratable o no, si bien la comunidad científica "está convencida" de que es intratable.



El problema del campeonato de liga de fútbol

Tras la jornada 25 del campeonato de liga de fútbol de primera división, un aficionado desea saber si su equipo tiene posibilidades matemáticas de quedar campeón.

* Sistema antiguo de puntuación $(2, 1, 0)$: **tratable**
* Sistema nuevo de puntuación $(3, 1, 0)$: **presuntamente intratable**

Las clases P y NP

P: clase de todos los problemas de decisión que son resolubles por **MTDs** que trabajan en tiempo polinomial (**problemas tratables**).

NP: clase de todos los problemas de decisión que son resolubles por **MTNDs** que trabajan en tiempo polinomial (**tratabilidad en modo no determinista**).

Puesto que toda MTD es una MTND, se tiene que **P** $\subseteq$ **NP**.

¿Es estricta la inclusión **P**$\subseteq$**NP**?

El problema P versus NP

DETERMINAR SI LAS CLASES P y NP SON IGUALES

Informalmente:

* **Encontrar** soluciones versus **comprobar** si una posible solución es correcta.

Se cree que es más difícil **resolver** un problema que **chequear** la corrección de una presunta solución (es decir, se cree que **P** $\neq$ **NP**).

(Clay Mathematics Institute: The Millenium Prize Problems)

La conjetura P $\neq$ NP

Si se verificara que **P** $\subsetneq$ **NP**, los candidatos idóneos para su demostración serían los **problemas más difíciles** de la clase **NP**.

Criptografía moderna (**public–key cryptography**: **DSE** y **RSA**).

* ¿Descifrar textos codificados en el sistema **Data Encryption Standard**?

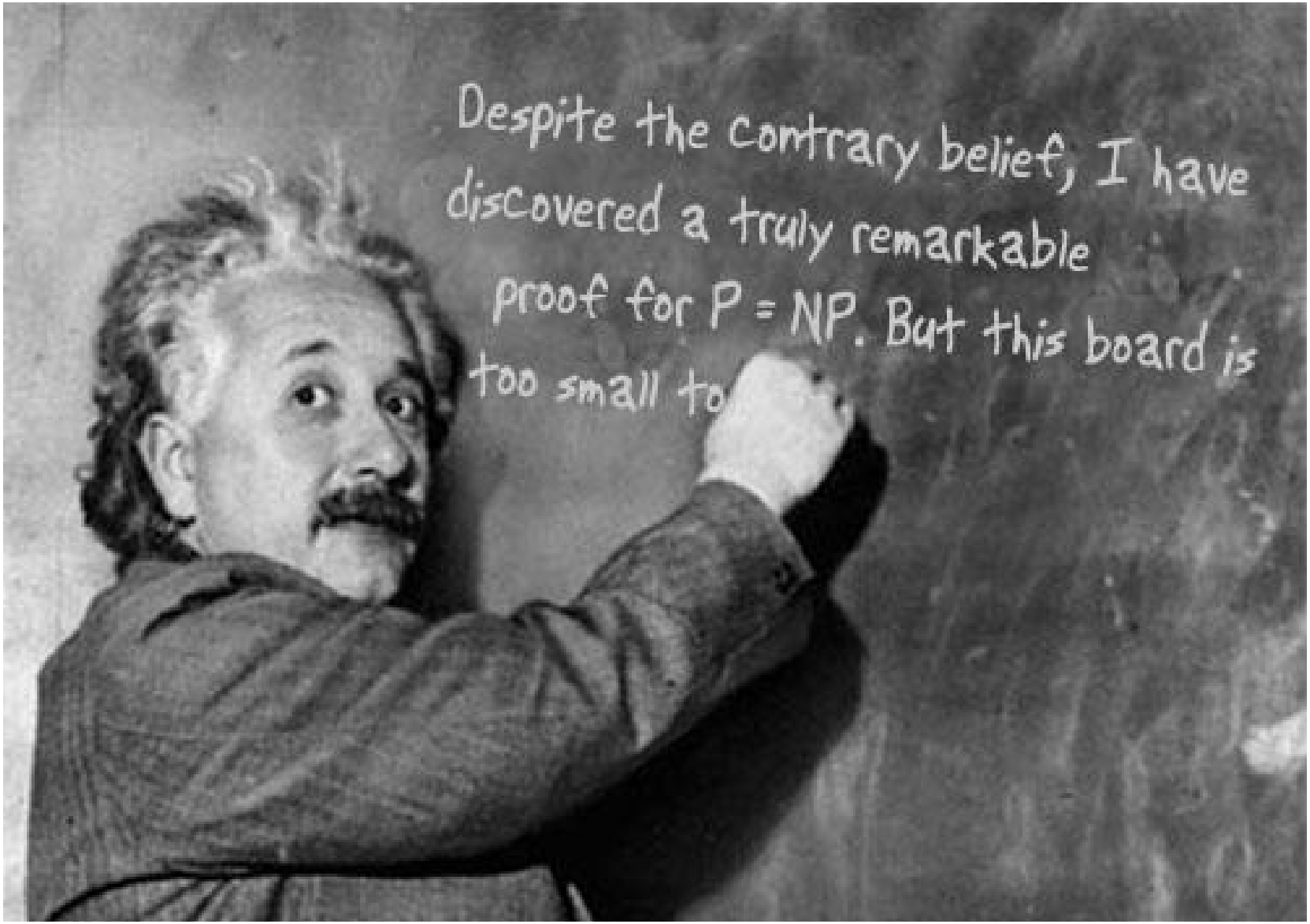
Ataque convencional sobre un texto **DES** mediante búsqueda exhaustiva:

* Un ordenador capaz de realizar un millón de operaciones por segundo, tardaría unos **mil años**.

* Puede ser descifrado por un **ordenador molecular** (de propósito general) en unos cuatro meses (D. Boneh, Ch. Dunworth y R.J. Lipton, 1996).

- ¿Qué sucedería si se probara que **P** $\neq$ **NP**?
  - ⋆ Reforzamiento de la **seguridad** de los actuales **sistemas de encriptación**.
  - ⋆ Existencia de problemas de complejidad **intermedia** (R. Ladner, 1975).

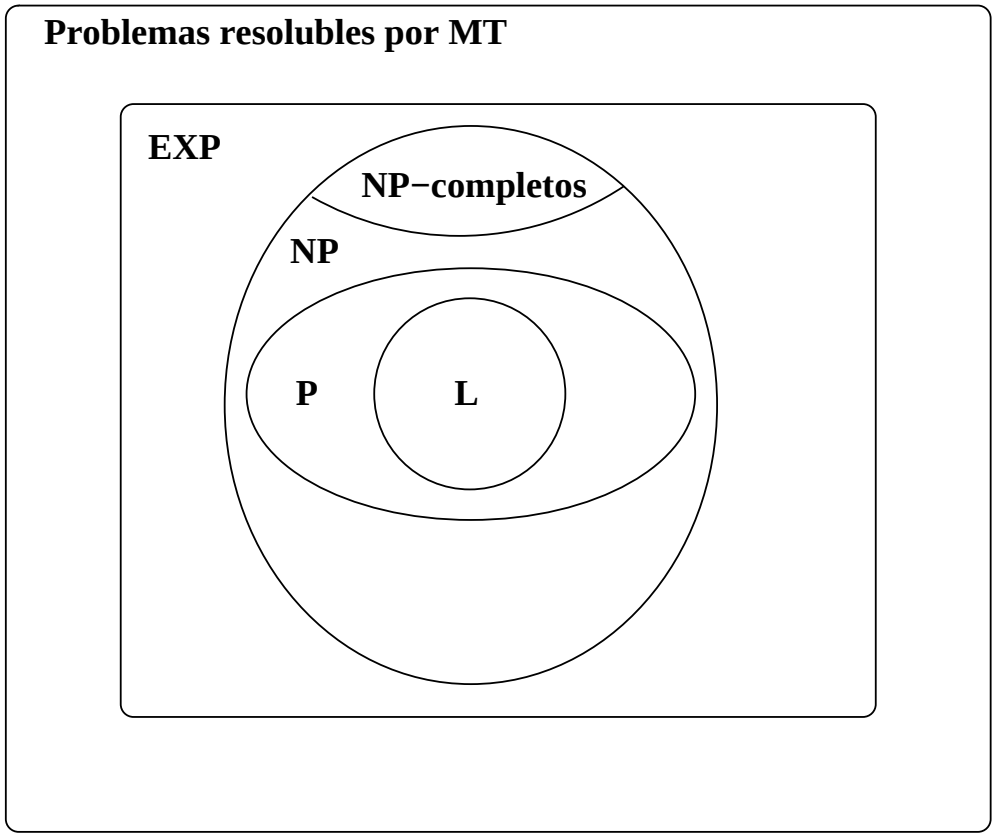
- ¿Qué sucedería si se probara que **P** $=$ **NP**?
  - ⋆ Consecuencias funestas para los sistemas de encriptación actuales.
  - ⋆ Obtención de **pruebas mecánicas de teoremas** matemáticos de interés.
  - ⋆ Posibilidad de construir un **decodificador universal** (**!!!**).


Despite the contrary belief, I have
discovered a truly remarkable
proof for P = NP. But this board is
too small to

Problemas NP–completos

Son los **problemas más difíciles** de la clase **NP**.

Candidatos idóneos para atacar el problema **P versus NP** (son **presuntamente intratables**).



Primer problema **NP**-completo (S.A. Cook, 1971):

* El problema **SAT** de la satisfactibilidad de la lógica proposicional.

Necesidad de mejorar **cuantitativamente** la resolución mecánica de problemas **NP**–completos.

¿Cómo enfrentarnos a un problema que es presuntamente intratable?

* Preguntarnos en qué aspecto del problema radica la razón de la dificultad.
* Intentar buscar una **solución aproximada** más simple.
* Tener presente que para algunos problemas, los algoritmos usan muchos recursos sólo en el caso peor.
* Considerar otros **modelos alternativos (no convencionales)**, una vez conocidas las limitaciones de las máquinas electrónicas.

Por ejemplo, los inspirados... en la ¡Naturaleza!