Modelos de Computación y Complejidad

Grado en Ingeniería Informática. Tecnologías Informáticas

Presentación de la asignatura

David Orellana Martín
Mario de J. Pérez Jiménez

Dpto. Ciencias de la Computación e Inteligencia Artificial
E.T.S. Ingeniería Informática
Universidad de Sevilla

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

Curso 2021-2022

Objetivos generales del curso

* ¿Qué significa **resolver mecánicamente** un problema abstracto?
  * Modelos de computación.
* Presentación de dos modelos de computación.
  * El **modelo** `GOTO`.
  * El modelo de las **máquinas de Turing**.
* Estudio de la **potencia computacional** de los modelos introducidos.
* Análisis de la complejidad computacional de problemas abstractos.
* Descripción del problema **P** versus **NP**.

Contenidos del curso

PARTE I: Teoría de la Computabilidad.

* **Modelos de computación**: **El modelo** `GOTO`.
* **Funciones computables**.
* **Programas universales**.
* **Recursividad enumerable e indecidibilidad**.

PARTE II: Teoría de la Complejidad Computacional.

* **Nociones básicas de Teoría de la Complejidad Computacional.**
* **El problema P versus NP**.

Evaluación de la asignatura

A. Evaluación alternativa

- ⋆ Cuestiones y ejercicios formulados y propuestos en clase.
- ⋆ **Exámenes online individualizado**.

B. Examen de evaluación final

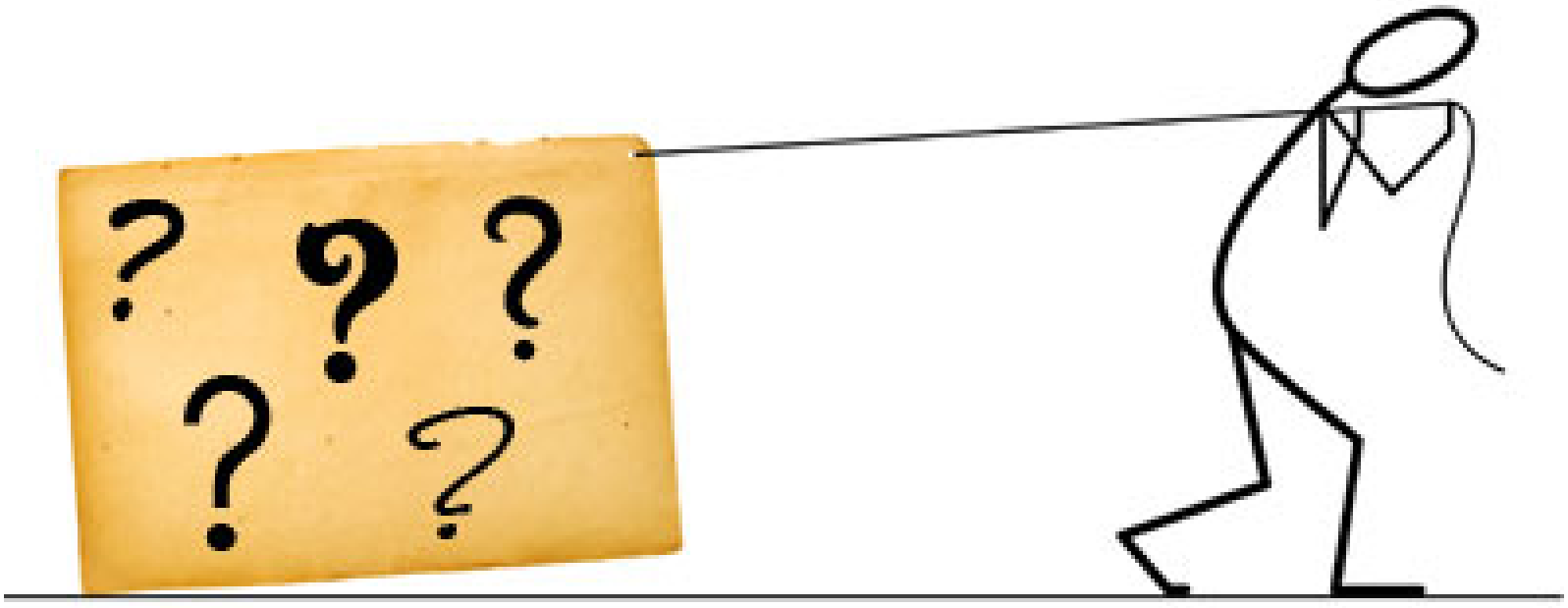
- ⋆ **Examen online individualizado** relativo a los contenidos impartidos a lo largo del curso académico.
- ⋆ Convocatoria oficial.

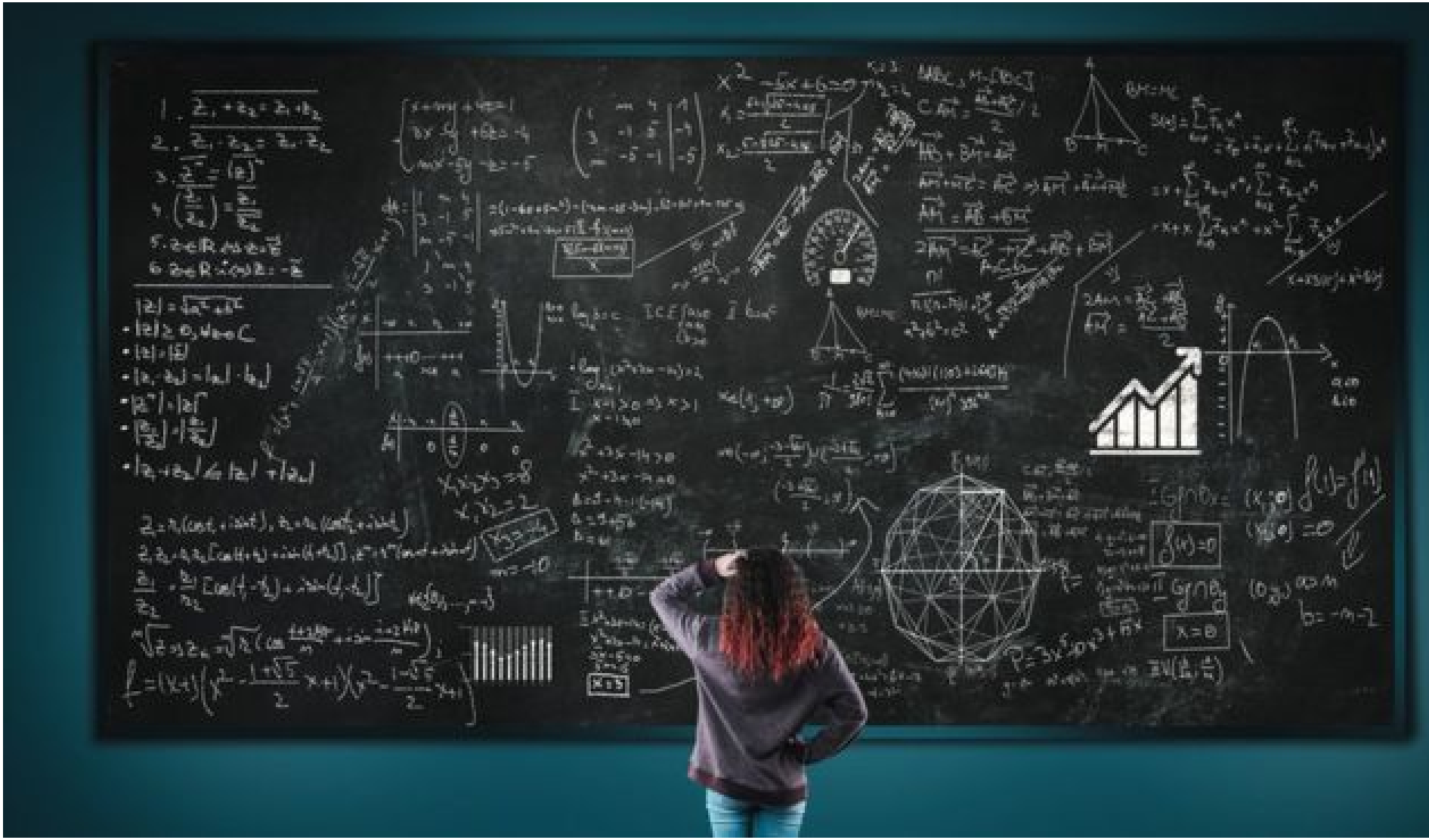
TUTORÍAS ONLINE (previa petición de cita a través del correo electrónico).

Página web de la asignatura

http://www.cs.us.es/cursos/mcc-2021/

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** ...

Problemas abstractos vs problemas concretos

(1) Determinar si el número 504291871 es primo.

(2) Hallar la suma de dos números naturales.

(3) Hallar el mínimo común múltiplo de 75348 y 1234.

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

(5) Hallar el producto de los divisores del número 52405.

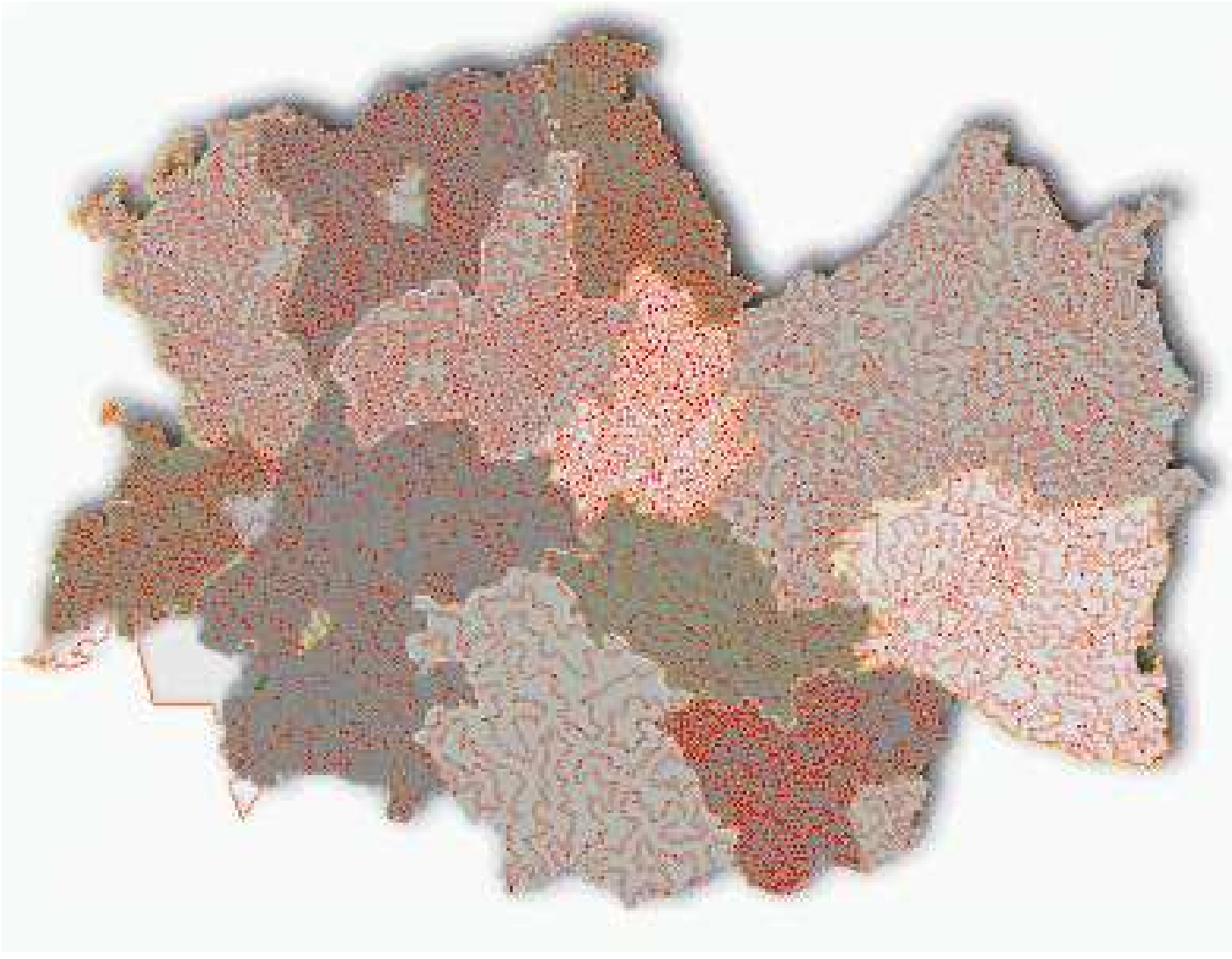
(6) Un repartidor dispone de un camión y trabaja con un hipermercado distribuyendo los productos que adquieren los clientes. Una mañana, recibe 59 electrodomésticos de la empresa para su entrega (todos caben en un camión). ¿Qué ruta debe seleccionar para minimizar los costes?

Un problema *concreto*

- **Dadas 51 ciudades y 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 y 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.**



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**).

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** dicha solución mediante un programa.
- ⋆ Se **ejecuta** el programa sobre una "máquina" introduciendo los datos específicos del problema concreto.

Soluciones mecánicas de problemas abstractos

Procedimiento mecánico/Algoritmo:

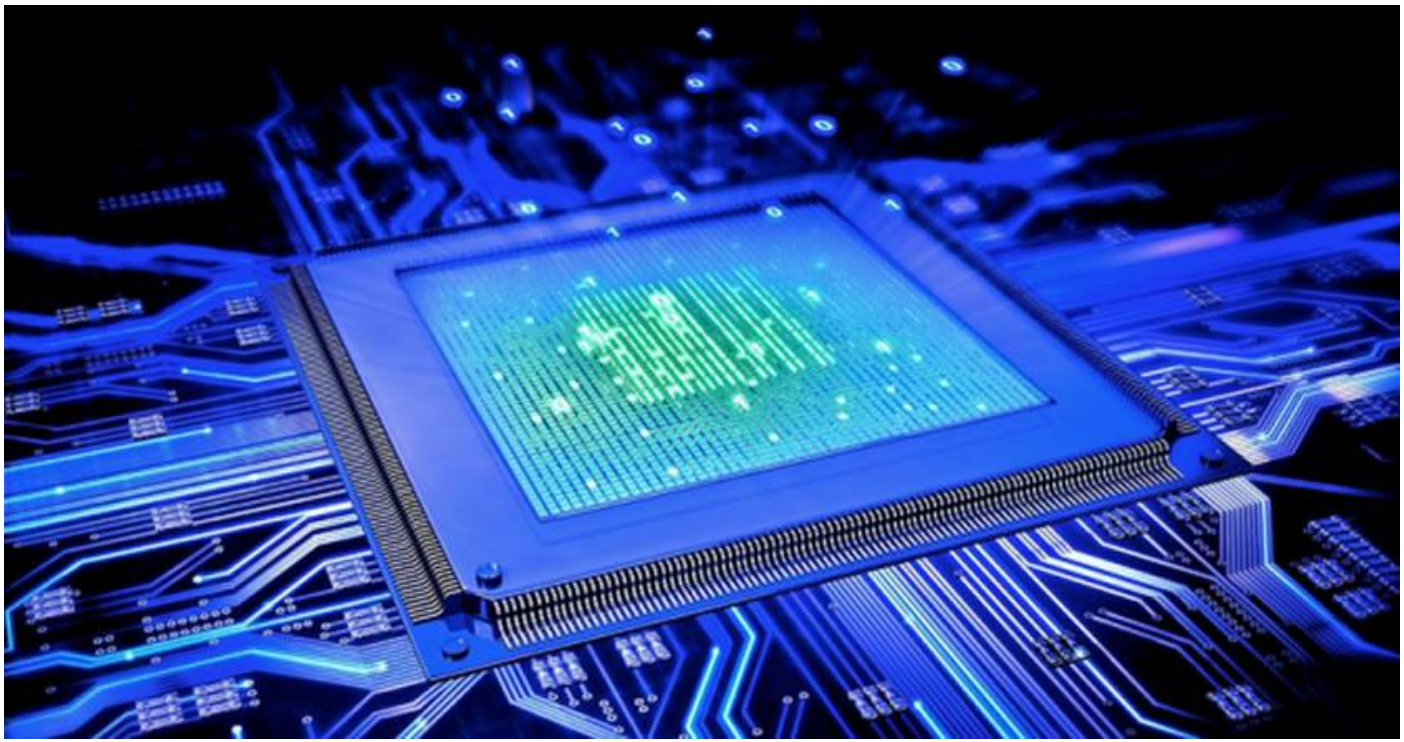
- ⋆ Método *especial* de resolución de problemas.
- ⋆ **Primer algoritmo no trivial**: máximo común divisor de dos números enteros (Euclides entre 400 y 300 a. C.).

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

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

Existencia de problemas **resolubles mecánicamente**.

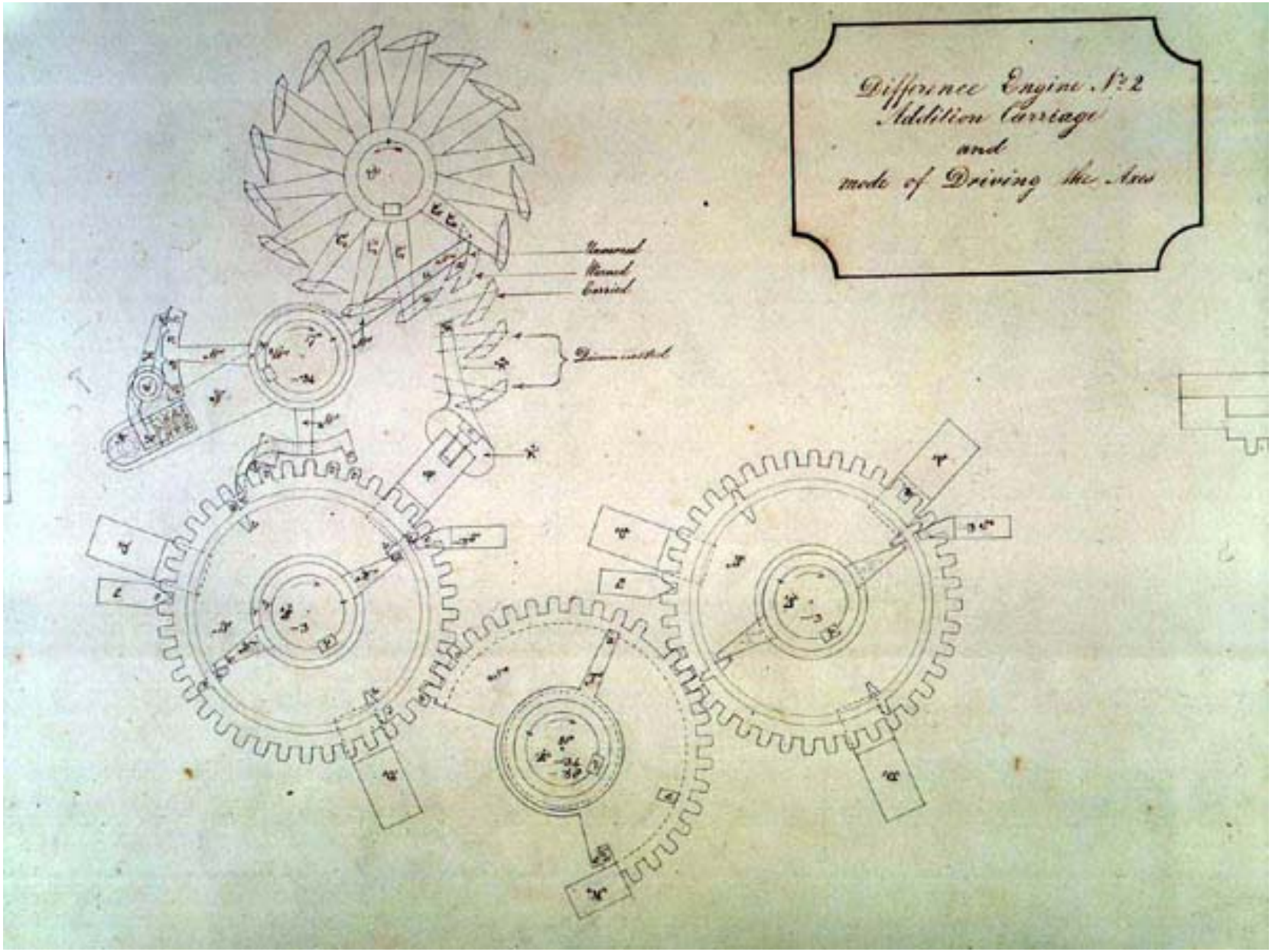
Máquinas, máquinas, máquinas...



* **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 (1847-1849).



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.

* 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 y para cada $a \in I_X$, $s_X(a)$ es un conjunto cuyos elementos se denominan **soluciones del problema** (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 $a \in I_X$, $s_X(a) \neq \emptyset$ y sus elementos se denominan **soluciones candidatas del problema**.
* f_X es una función (**objetivo**) que asigna a cada $a \in I_X$ y cada $c_a \in s_X(a)$, un número racional positivo $f_X(a, c_a)$.

La función f_X proporciona el 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 $s_X(a) = \{1\}$.

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

* Tiene asociado un predicado, θ_X, sobre I_X definido así: para cada $a \in I_X$ se tiene que $\theta_X(a) = 1$ si y sólo si $s_X(a) = \{1\}$.
* 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 $f_X(a, \theta_X(a)) = 1$.
* 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 $s_{X_L}(a) = \{1\}$ 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:

* $(\Sigma_X, I_X, s_X,)$ 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}
Para cada 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 que n veces el coste en tiempo de $\mathcal{A}$.

Paradigmas de computación

Hacia finales del siglo XIX:

* Importante lista de problemas de los que se desconocen **soluciones mecánicas**.

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

* **Asimetría** entre la respuesta afirmativa o negativa a esta cuestión.

¿En qué consiste un modelo de computación?

* Formalizar el concepto de procedimiento mecánico.

Primeros modelos de computación

- ⋆ **Funciones recursivas**: **K. Gödel**, 1931–1934.
- ⋆ **Funciones λ–cálculables**: **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**: resoluble mecánicamente en un modelo.

¿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).
  - ⋆ 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

Búsqueda de **algoritmos** (**soluciones mecánicas**) que resuelven problemas abstractos.

Objetivo: clasificar los problemas abstractos según sean o no resolubles mecánicamente (problemas **decidibles** e **indecidibles**).

Protocolo de resolución mecánica de un problema abstracto:

- **Diseño**: descripción de un algoritmo para resolver el problema.
- **Verificación formal**: el algoritmo resuelve cada instancia del problema.
- **Análisis**: cálculo de los recursos que un algoritmo necesita para procesar una instancia arbitraria (en **función** de su **tamaño**).

Teoría de la Complejidad Computacional

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

- ⋆ **Algoritmos óptimos** que resuelven problemas abstractos.
- ⋆ **Complejidad computacional inherente** a un problema abstracto.
- ⋆ Existen problemas que carecen de algoritmos óptimos que lo resuelven.
- ⋆ Problema **tratable**: existe un algorimo que lo resuelve usando una cantidad polinomial de recursos.

Objetivo: clasificar los problemas abstractos según sean **tratables** o no.

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

Resolubilidad mecánica práctica de problemas abstractos.