Complejidad Computacional

.

Tema 1: Introducción

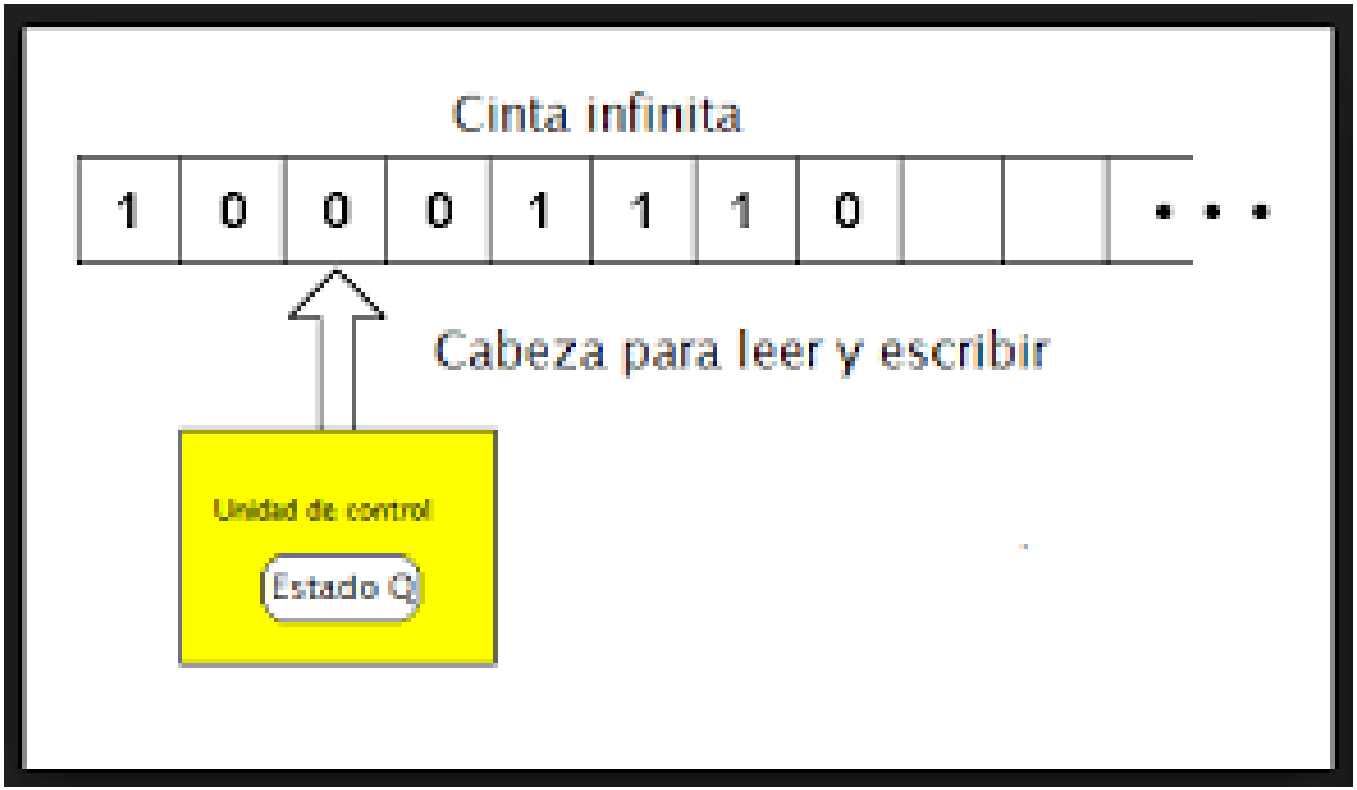
Julián Viera
Alfredo Viola
Universidad de la República

El problema de decisión (Entscheidungsproblem).

- Problema en lógica simbólica propuesto por Hilbert y Ackermann en 1928 de encontrar un algoritmo general que decidiese si una fórmula del cálculo de primer orden es un teorema.
- En 1936, de manera independiente, Alonzo Church y Alan Turing demostraron ambos que es imposible escribir tal algoritmo.
- Problema fundamental: Para probar la inexistencia primero hay que **definir qué es un algoritmo!**
- Alonzo Church presentó el concepto de *calculabilidad efectiva* definiendo el **cálculo lambda**.
- Alan Turing presentó su **máquina que computa** en particular la **máquina universal**, que simula todas las otras máquinas.
- **Los modelos son equivalentes** (tesis de Church-Turing).

La máquina de Turing.

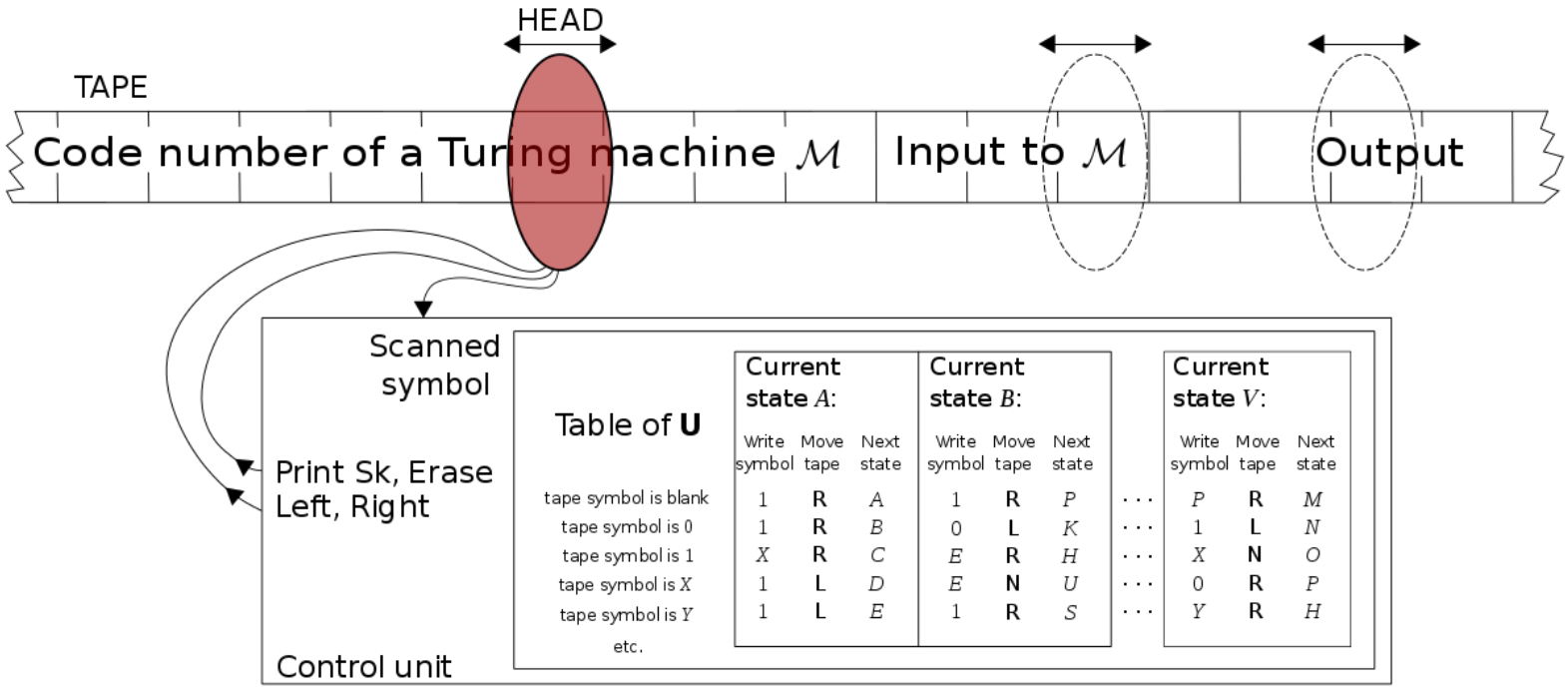
- La gran ventaja del formalismo de Turing es que presenta **de una manea sencilla y elegante** cómo un **humano ideal cumpliría órdenes de una manera precisa y no ambigua**.



- Además el conjunto de todas las Máquinas de Turing **se puede enumerar!**

La máquina de Turing Universal.

- Toma como entrada una pareja de enteros (n, x) donde n codifica una Máquina y x codifica su entrada, y simula la ejecución de la Máquina de Turing n con entrada x.



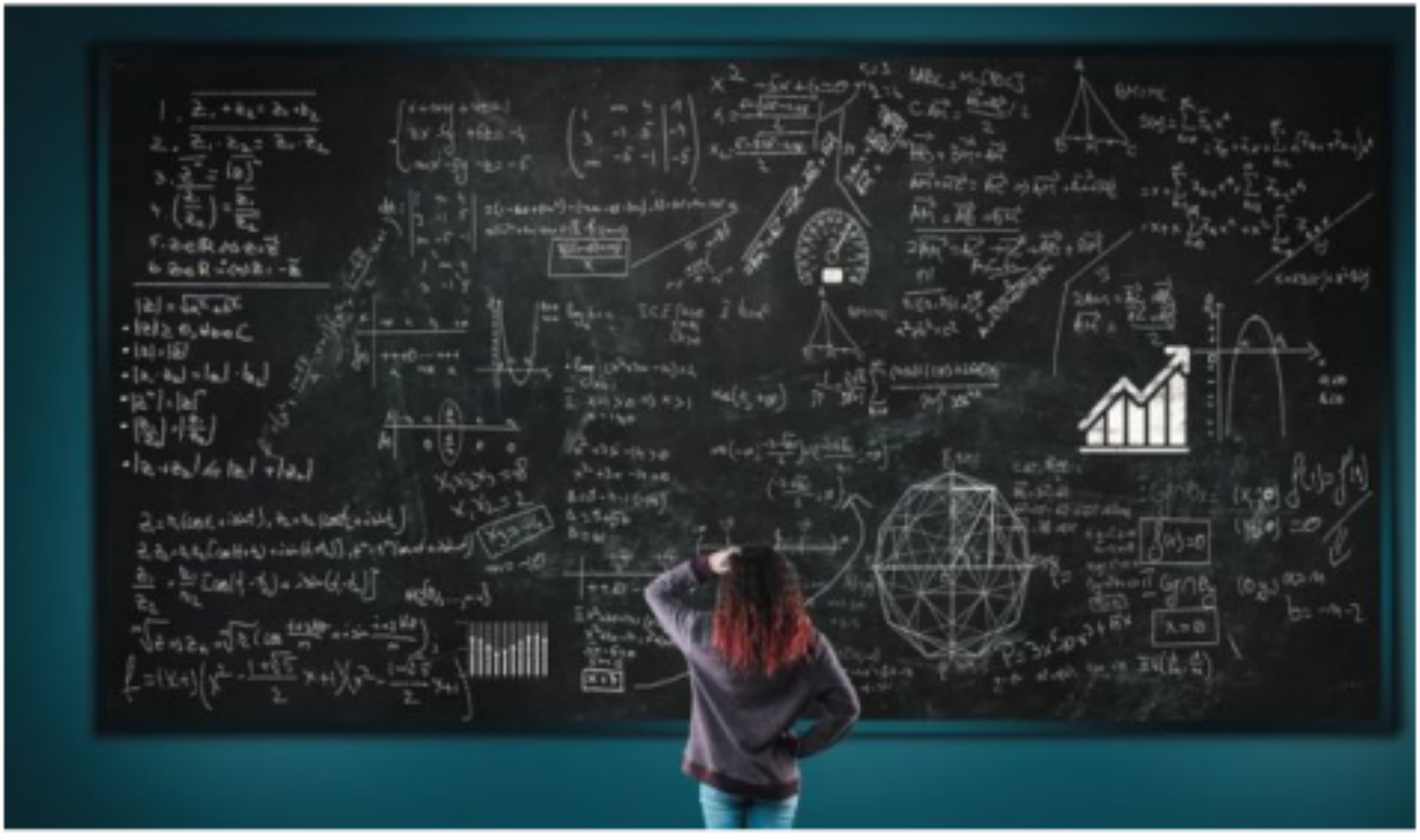
Problemas no computables.

- La respuesta negativa al *Entscheidungsproblem* tuvo al menos dos consecuencias fundamentales:
  1. La definición de varios modelos de computabilidad (algoritmos) todos ellos **equivalentes** desde el punto de vista de su **poder expresivo**.
  2. Que hay funciones que **no son computables** (en particular el *Entscheidungsproblem*) en el sentido de que matemáticamente se prueba que no existe ninguna Máquina de Turing que pueda computarlas.
- De hecho, una inmensa cantidad de **problemas naturales** no son computables.
- Uno de ellos es el **problema de la parada**, que toma como entrada la pareja de enteros (n, x) y devuelve 1 si la Máquina de Turing n para cuando toma entrada x y 0 en caso contrario.

Problemas Computables, Tecnología y Recursos.

- En 1936 y años siguientes Turing y Church estaban interesados en problemas matemáticos.
- En particular, una vez probado que el *Entscheidungsproblem* no era computable, el objetivo era entender los alcances y los límites de la computabilidad.
- De hecho la Tesis de doctorado de Turing (dirigida por Church) fue en lógica.
- Unos años después (y motivados por la experiencia en la Segunda Guerra Mundial) se entendió que la **Máquina Universal** de Turing podía ser fuente de un producto tecnológico, que es la computadora moderna.
- El curso se centrará en formalizar y entender los **recursos necesarios** para resolver diversas **clases de problemas computables**.
- Algunos ejemplos de recursos son **espacio, tiempo, bits aleatorios**.

Problemas



Tamaño de entrada de un problema.

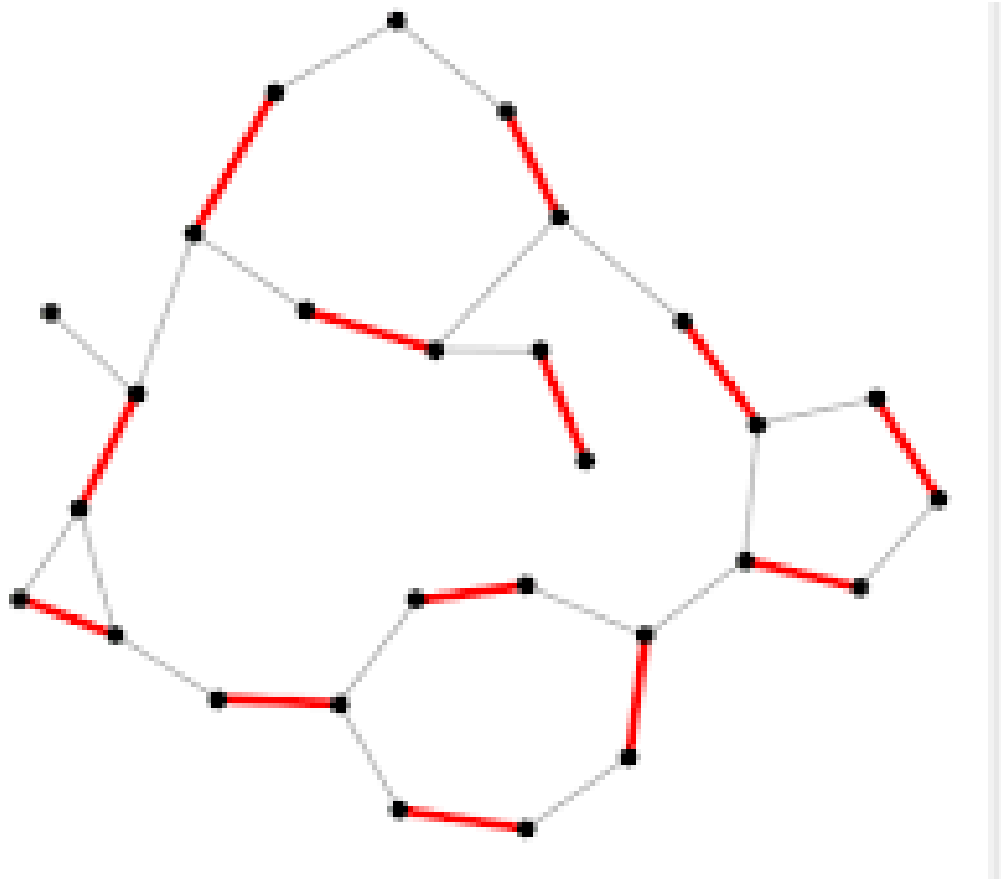
- Informalmente, **"lleva más tiempo y espacio"** multiplicar 12 por 18 que 14731237 por 9732125.
- Para saber cuántos recursos utiliza un algorimo general para resolver un problema, hay que definir su **tamaño de entrada**.
- En este caso parece **"natural"** definirlo como la cantidad de bits necesarios para describir los multiplicadores **en binario**.
- Entonces, se puede analizar **"la cantidad de recursos" T(n)** (p.ej. número de multiplicaciones elementales) que utiliza el algoritmo para resolver entradas de tamaño n en el **peor caso**.
- Lo relevante es saber cómo aumenta **T(n)**, al aumentar el tamaño de entrada n.
- Se quiere entender si el algoritmo es apropiadamente **escalable** para **"valores grandes"** de n.

Jugando con los números.

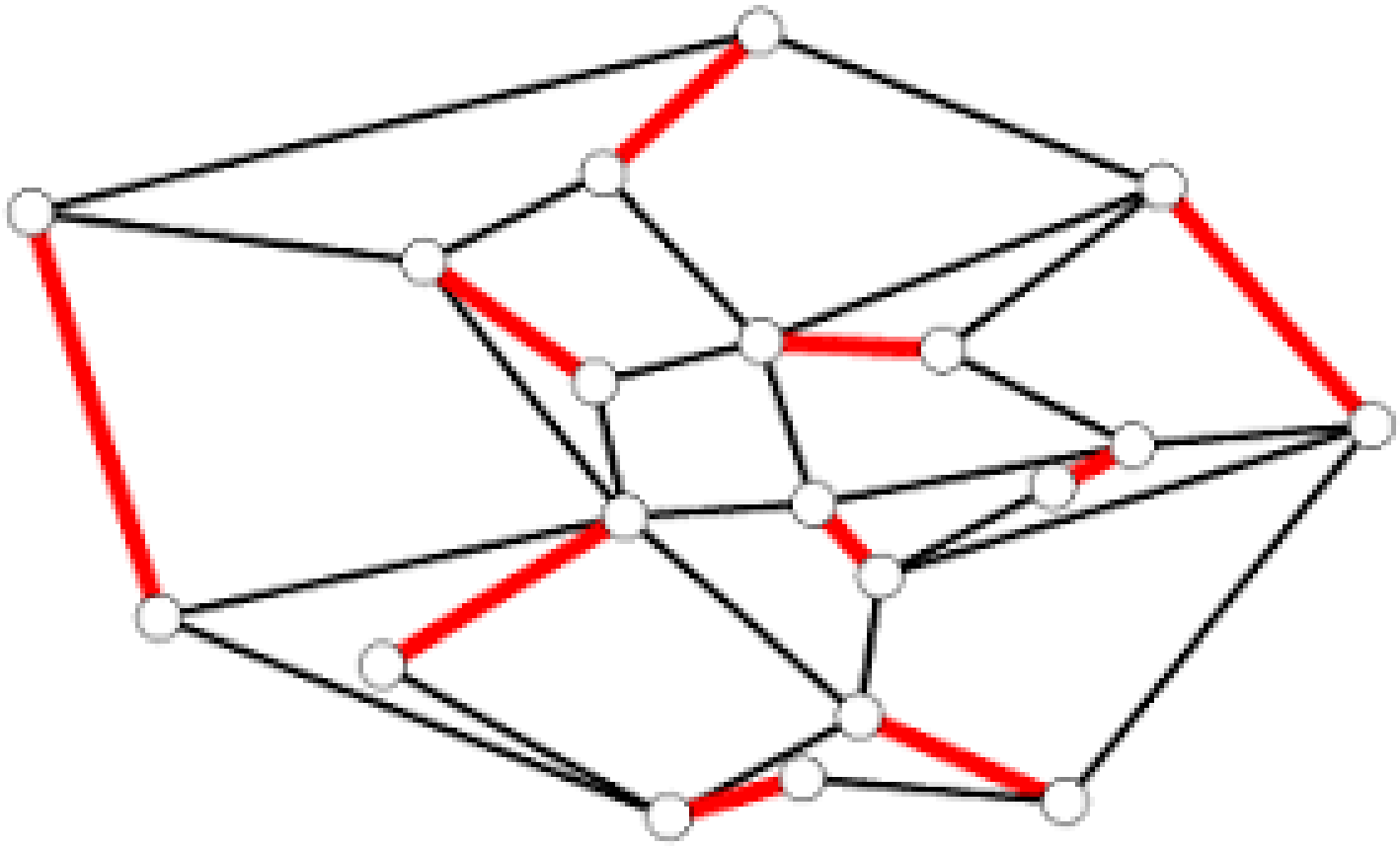
n	$\lceil \lg n \rceil$	n^2	n^3	2^n
10	4	100	1000	1024
20	5	400	8000	$\approx 1.04 \times 10^6$
40	6	1600	64000	$\approx 1.10 \times 10^{12}$
160	8	25600	$\approx 4.09 \times 10^6$	$\approx 1.46 \times 10^{48}$
300	9	90000	$\approx 2.70 \times 10^7$	$\approx 2.03 \times 10^{90}$
512	9	262144	$\approx 1.34 \times 10^8$	$\approx 1.34 \times 10^{154}$
1024	10	$\approx 1.04 \times 10^6$	$\approx 1.07 \times 10^9$	$\approx 1.79 \times 10^{308}$

- Número estimado de átomos en el Universo: $\approx 10^{82}$.
- Operaciones por segundo del chip más rápido: $\approx 1.78 \times 10^{12}$.
- Segundos estimados hasta que explote el Sol: $\approx 1.58 \times 10^{17}$.
- Operaciones totales de un chip más rápido funcionando hasta que el Sol explote: $\approx 2.81 \times 10^{29}$.
- Operaciones totales de conjunto de chips más rápidos igual al número estimado de átomos en el Universo funcionando hasta que el Sol explote: $\approx 2.81 \times 10^{111}$.

Edmonds (1965): apareo máximo en grafos.

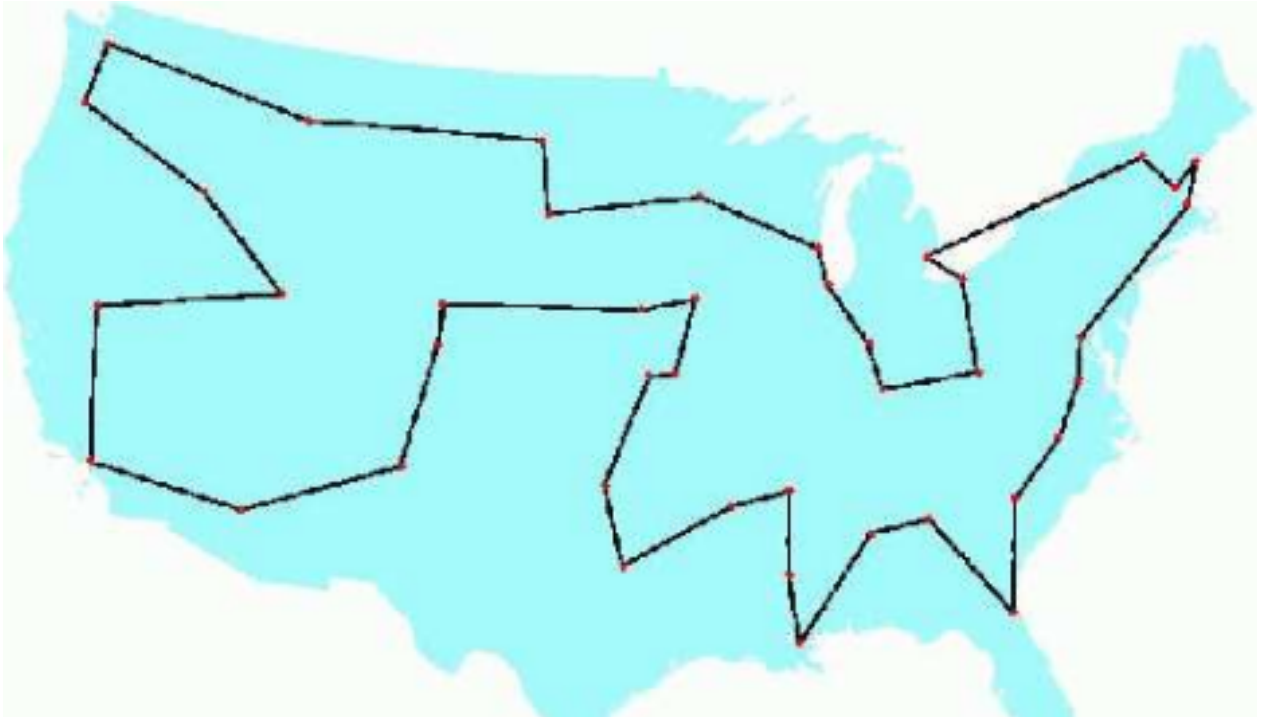


Edmonds (1965): apareo máximo en grafos.



Un problema de decisión.

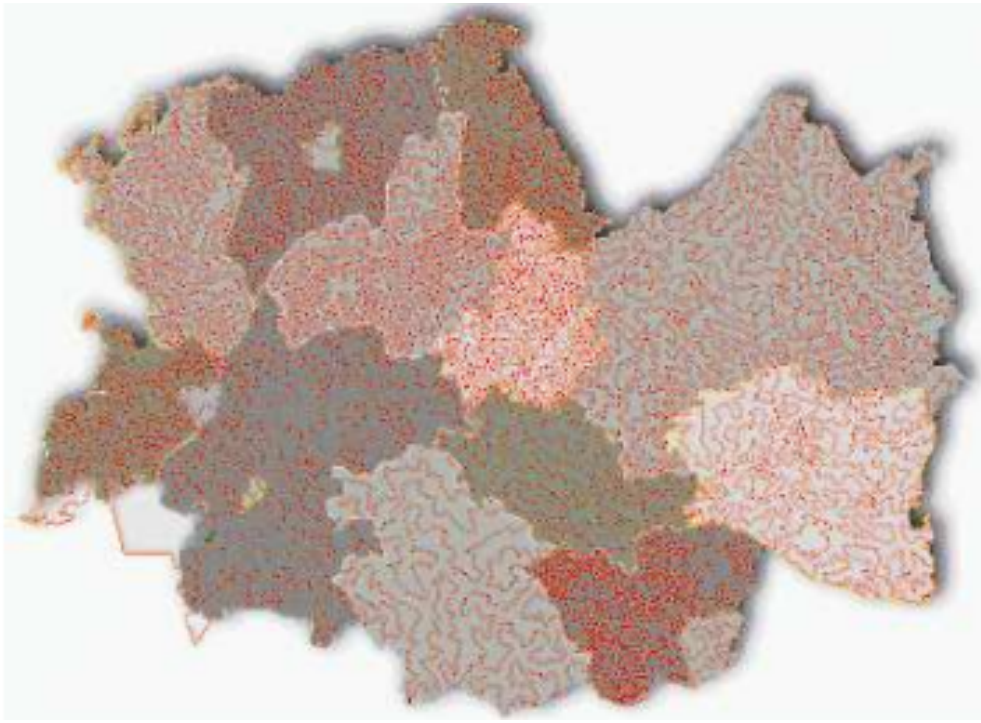
- Dadas 25 ciudades, distancias $d_{i,j}$ entre las ciudades i y j, y una cota D, determinar si existe un circuito que recorra las 25 ciudades con una distancia total menor o igual a D.



- Si nos dan una solución, se puede verificar con 25 sumas.
- Con muchos recursos se puede hallar solución.

Otro problema de decisión.

- Dadas 3200 ciudades, distancias $d_{i,j}$ entre las ciudades i y j, y una cota D, determinar si existe un circuito que recorra las 3200 ciudades con una distancia total menor o igual a D.



- Si nos dan una solución, se puede verificar con 3200 sumas.
- Se especula que no se puede resolver el problema de manera eficiente.

Problema del viajante de comercio (TSP)

Dadas n ciudades, distancias $d_{i,j}$ entre las ciudades i y j, y una cota D_n, determinar si existe un circuito que recorra las n ciudades con una distancia total menor o igual a D_n.

- Si nos dan una solución, se puede verificar con n sumas.
- Se especula que no se puede resolver el problema de manera eficiente.

Poder verificar eficientemente una solución de un problema y sin embargo **no poder probar la inexistencia de algoritmos eficientes para resolverlo** es el desafío fundamental de la **Complejidad Computacional**.

Lo que vamos a estudiar en el curso.

1. **Modelos de computación**. Fundamental para tener bases rigurosas sobre las cuales razonar y probar teoremas.
2. Estudiar **recursos necesarios** para resolver diversos problemas. Estudiamos problemas decidibles, y hay que resolverlos usando **máquinas concretas**.
3. **Definir rigurosamente clases de complejidad** y estudiar **relaciones fundamentales** entre ellas.
4. **Complejidad en el tiempo.**
   1. Clases **P, NP, EXP, NEXP, CO-NP**.
   2. **Problemas NP completos**.
   3. **Conjetura fundamental de la Complejidad Computacional**.
5. **Complejidad en el espacio.**
   1. Clases **PSPACE, NPSPACE, L, NL**.
   2. **Problemas completos**.
6. **Jerarquía polinomial.**
7. **Complejidad aleatoria.**
   1. Clases **RP, CO-RP, ZPP, BPP**.
   2. **Relaciones fundamentales** con otras clases de complejidad.
8. Aplicaciones paradigmáticas.