8. Decibilidad y Complejidad

Hasta ahora hemos estudiado qué pueden hacer las máquinas. En este tema estudiamos sus **límites** (qué no pueden hacer) y su **eficiencia** (cuánto tardan).

8.1 Computabilidad vs. Decidibilidad

Primero, distinguimos dos tipos de preguntas que le hacemos a una Máquina de Turing (MT):

1. Computabilidad (Calcular valores):
   Una función es computable si existe una MT capaz de calcular el resultado para cualquier entrada del dominio.
   - *Ejemplo:* Calcular $f(x) = x^2$.
2. Decidibilidad (Responder Sí/No):
   Hablamos de decidibilidad cuando el problema tiene una respuesta binaria (aceptar/rechazar).
   - **Problema Decidible:** Existe una MT que **siempre se para** y da la respuesta correcta (Sí o No) para cualquier entrada.
   - **Problema Indecidible:** No existe ninguna MT que pueda resolverlo siempre (se quedaría en bucle infinito para algunos casos).

8.2 El Problema de la Parada (The Halting Problem)

Este es el ejemplo clásico de problema **indecidible**.

> **El Problema:** ¿Existe una máquina universal H que, si le damos el código de otra máquina M y una entrada w, nos diga si M se parará o se quedará en bucle infinito?.

La Respuesta: NO. Es imposible construir tal máquina.

- Se demuestra por reducción al absurdo. Si existiera, podríamos crear una paradoja (una máquina que se para si la predicción dice que no se para, y viceversa).
- **Conclusión:** El problema de la parada es **indecidible**.

Implicación Teórica:

Si pudiéramos resolver el problema de la parada, entonces todos los Lenguajes Recursivamente Enumerables (que aceptan, pero pueden no parar al rechazar) se convertirían automáticamente en Lenguajes Recursivos (que siempre paran).

8.3 Complejidad Computacional

Aquí no nos importa *si* se puede resolver, sino **cuánto cuesta** resolverlo (tiempo y memoria).

- Usamos la notación **O-grande ($O(...)$)** para medir el orden de magnitud. Nos interesa cómo crece el tiempo al aumentar el tamaño de la entrada (n).

El Hardware Importa (en Complejidad)

En el tema anterior dijimos que todas las variaciones de MT son equivalentes. **En complejidad, NO lo son.**

- Un problema puede tardar $O(n^2)$ en una MT estándar (una cinta).
- El mismo problema puede tardar $O(n)$ en una MT con **dos cintas**.
  - *Moraleja:* La estructura de la máquina afecta a la eficiencia, aunque no a la capacidad de resolver el problema.

8.4 Clases de Complejidad: P y NP

Clasificamos los lenguajes según lo difícil que es para una máquina aceptarlos.

Definiciones Formales

Clase	Definición	Tipo de Máquina	Tiempo
P	**Polinómico**	MT **Determinista**	Se resuelve en tiempo n^k (polinómico)8.
NP	**No Polinómico**	MT **No Determinista**	Se resuelve en tiempo n^k (polinómico) usando "adivinación"9.

> **Nota:** NP no significa "No Polinómico" literalmente en inglés, sino "Nondeterministic Polynomial time".

Relación entre P y NP

1. $P \subseteq NP$**:** Todo problema que una máquina determinista resuelve rápido, una no determinista también puede resolverlo rápido (simplemente no usa su capacidad de adivinar)10.
2. **La gran pregunta:** ¿$P = NP$? Nadie lo sabe. Se asume que $P \neq NP$, es decir, que hay problemas en NP que son intrínsecamente difíciles.

Tratabilidad

- **Problemas Tratables (Clase P):** Son viables de resolver en la práctica (ej: ordenar una lista).
- **Problemas Intratables:** Aunque son computables, requieren tantos recursos (tiempo exponencial) que no son viables para entradas grandes11. Según la tesis de Cook-Karp, todo lo que está fuera de P se considera intratable.

8.5 Reducibilidad y NP-Completo

Para estudiar los problemas más difíciles, usamos la técnica de **reducción**.

Reducción Polinomial:
Un problema L_1 se reduce a L_2 si podemos transformar cualquier entrada de L_1 en una entrada de L_2 rápidamente (tiempo polinómico).

- Significa que L_2 es **al menos tan difícil** como L_1.
- Si L_1 se reduce a L_2 y sabemos que L_2 es fácil (P), entonces L_1 también es fácil (P).

NP-Completo (Los "Jefes Finales")

Un lenguaje L es **NP-Completo** si cumple dos condiciones:

1. Pertenece a la clase **NP**.
2. **Cualquier** otro problema de NP se puede reducir a él.

Ejemplo: SAT (Satisfacibilidad)

- Dada una fórmula lógica, ¿existe una combinación de True/False que la haga verdadera?
- En una MT Determinista (ordenador real): Coste $O(2^n)$ (Exponencial - Intratable).
- En una MT No Determinista (teórica): Coste $O(n)$ (Polinómico).

> **Resumen para examen:**
>
> - **Problema de la parada:** Indecidible.
> - **Clase P:** Fácil de resolver (Determinista Polinómico).
> - **Clase NP:** Fácil de verificar, difícil de resolver sin "magia" (No Determinista Polinómico).
> - **NP-Completo:** Los problemas más difíciles de NP. Si resuelves uno rápido, resuelves todos.