Clase 2, Módulo 1: Problemas computacionales y razonamiento

1. ¿Qué es un problema computacional?

Cuando hablamos de **problema computacional**, no nos referimos a una dificultad de la vida cotidiana en general, sino a un tipo muy particular de problema: aquel que puede describirse de manera precisa para que una computadora lo resuelva.

Un problema computacional siempre **tiene tres componentes básicos**:

1. **Una entrada**: son los datos iniciales que se le dan al sistema.
   Ejemplo: una lista de números, una frase escrita en un idioma, un tablero de Sudoku incompleto.
2. **Un proceso**: es el conjunto de pasos o instrucciones (un algoritmo) que la máquina debe seguir para transformar la entrada.
   Ejemplo: un algoritmo que ordena la lista de números de menor a mayor.
3. **Una salida**: es el resultado final del proceso aplicado a los datos de entrada.
   Ejemplo: la lista ordenada, la frase traducida a otro idioma, o el Sudoku completado.

Como primer ejemplo sencillo imaginemos el problema de determinar si un número es par o impar. Aquí la **entrada** sería un número, por ejemplo 8. El **proceso** consistiría en aplicar una regla que diga "si el número es divisible por 2, entonces es par; si no, es impar". Y finalmente la **salida** podría ser algo como "el número ingresado es par".

De lo anterior podemos observar que este problema sencillo (corto y claro) el tiempo que requiere la computadora es insignificante incluso con números muy grandes.

Pensemos ahora un ejemplo más complejo, en un **Sudoku**.

- Entrada: un tablero parcialmente completado.
- Proceso: aplicar un conjunto de reglas que indiquen cómo se completan las filas, columnas y subcuadrados con los números del 1 al 9 sin repeticiones.
- Salida: un tablero completo válido.

Aquí el problema sigue siendo computable, pero el proceso puede volverse mucho más complejo. A medida que aumentan los casilleros vacíos en el tablero, la computadora debe considerar muchísimas combinaciones posibles para encontrar una solución válida.

Por otro lado debemos notar que NO todos los problemas de la vida real son **problemas computacionales**. Para que lo sean, deben poder expresarse en términos claros de entradas, procesos y salidas. "Ser feliz" o "tener éxito" no son problemas computacionales, porque no se pueden traducir en un algoritmo preciso. En cambio, "encontrar la ruta más corta entre dos ciudades" sí lo es, porque podemos describirlo en pasos que una máquina puede ejecutar.

En resumen un problema computacional es cualquier tarea que pueda formalizarse como **datos de entrada → algoritmo → resultados de salida**. Esta estructura simple es la base de toda la informática y también el punto de partida para entender los límites de la inteligencia artificial.

2. Problemas tratables vs. Intratables

Una vez que entendemos qué es un problema computacional, surge una pregunta clave: ¿todos los problemas que una computadora puede resolver son igual de "fáciles" de manejar? La respuesta es no. Aquí entra en juego la diferencia entre **problemas tratables** e **intratables**.

Un **problema** se considera **tratable** cuando puede resolverse en un **tiempo razonable**, incluso si la cantidad de datos crece. En ciencias de la computación, lo "razonable" se suele asociar a algoritmos cuya complejidad se expresa como una **función polinomial** del tamaño de la entrada. Para describir esta complejidad, se utiliza la **notación Big-O**, que indica cómo crece el tiempo de ejecución en función de la cantidad de datos n.

Por ejemplo, un algoritmo que ordena una lista con eficiencia tiene una complejidad de $O(n \log n)$. Esto significa que, si duplicamos el tamaño de la lista, el tiempo aumenta, pero de forma controlada. Así, ordenar 10 elementos se hace en milésimas de segundo, y ordenar 10.000 todavía se resuelve en segundos. El tiempo crece, pero no explota de manera incontrolable.

En cambio, los **problemas intratables** son aquellos cuya complejidad crece de manera explosiva. Suelen expresarse con notaciones como $O(2^n)$ ó $O(n!)$. En estos casos, cada pequeño aumento en el tamaño de la entrada multiplica el tiempo de cálculo de forma brutal, volviendo imposible su resolución en la práctica, aunque tengamos las computadoras más potentes.

Un ejemplo clásico es el **problema del viajante**. Con 4 ciudades, hay solo 6 rutas posibles; con 10 ciudades, más de 300.000; con 20 ciudades, el número supera los 60 trillones. En términos de complejidad, el problema crece con factoriales $O(n!)$. y esto lo vuelve impracticable. A este ritmo, aunque usáramos todas las computadoras del mundo, el cálculo podría tardar más que la edad del universo.

La distinción no es entre problemas "resolubles" y "no resolubles". Ambos tipos son computables: en teoría, una computadora puede hallar la solución siguiendo los pasos del algoritmo. Lo que marca la diferencia es si la solución puede encontrarse en un **tiempo útil en la práctica**.

Por eso, la frontera entre lo tratable y lo intratable es fundamental en Inteligencia Artificial. La IA surge, en parte, para proponer **estrategias inteligentes** (heurísticas, aproximaciones, aprendizaje) que permiten trabajar con problemas intratables sin necesidad de encontrar la solución perfecta, pero sí una solución suficientemente buena en un tiempo razonable.

3. Complejidad computacional: P, NP, NP-Hard y NP-Complete

La **teoría de la complejidad computacional** busca responder una pregunta fundamental**: *¿qué tan difícil es resolver un problema usando una computadora?*** No basta con saber que existe un algoritmo capaz de dar una solución; lo verdaderamente importante es si esa solución puede encontrarse en un **tiempo razonable**.

Para estudiar esto, los científicos de la computación utilizan la **notación Big-O**, que describe cómo crece el tiempo de ejecución de un algoritmo a medida que aumenta el tamaño de la entrada n. Un algoritmo con complejidad $O(n^2)$ implica que el tiempo se multiplica por cuatro si se duplica el tamaño de los datos; en cambio, uno con complejidad $O(2^n)$ crece de forma exponencial, volviéndose inabordable incluso para entradas moderadas.

La clase P: problemas resolubles eficientemente

La clase **P** (Polynomial time) contiene todos aquellos problemas que pueden resolverse mediante algoritmos cuya complejidad es polinomial, es decir, $O(n^k)$ con k una constante. En términos prácticos, son los problemas que las computadoras pueden manejar con eficacia aun cuando los datos crecen mucho.

Un ejemplo claro es el **ordenamiento de una lista de números**. Con algoritmos como *Quicksort* o *Mergesort*, cuya complejidad es $O(n \log n)$, se puede ordenar desde una lista pequeña hasta una con millones de elementos en tiempos perfectamente razonables.

La clase NP: fáciles de verificar, difíciles de resolver

La clase **NP** (Nondeterministic Polynomial time) incluye problemas para los que **verificar una solución es rápido** (polinomial), pero **encontrar la solución puede ser extremadamente costoso**.

Un ejemplo clásico es el **Sudoku**. Si alguien nos entrega un tablero completo, comprobar que cumple las reglas es sencillo y rápido, en tiempo polinomial. Sin embargo, resolverlo desde cero puede implicar probar una cantidad inmensa de combinaciones posibles. Lo mismo ocurre con la **factorización de números grandes**: verificar que 391=17×23391 = 17 es trivial, pero factorizar un número de 300 dígitos puede tardar más que la vida útil del universo con los algoritmos actuales.

NP-Hard y NP-Complete: los problemas más difíciles

Dentro de este marco aparecen dos categorías que marcan la frontera de la dificultad.

- **NP-Hard** son problemas al menos tan difíciles como los más complicados de NP. Resolver uno de ellos en tiempo polinomial equivaldría a poder resolver cualquier problema de NP. No siempre es sencillo verificar sus soluciones, por lo que no necesariamente pertenecen a NP, pero definen el límite superior de la dificultad computacional.
- **NP-Complete** son los problemas más representativos dentro de NP. Constituyen la "prueba de fuego" de la complejidad. Un ejemplo paradigmático es el

problema del viajante: encontrar la ruta más corta que pase por todas las ciudades y vuelva al inicio.

Verificar que una ruta propuesta mide cierta distancia es rápido, pero descubrir la ruta óptima implica analizar combinaciones cuyo número crece factorialmente $O(n!)$.

El dilema P vs. NP

La gran pregunta abierta es si **P es igual a NP**. En otras palabras: ¿existe algún algoritmo eficiente en tiempo polinomial que pueda resolver todos los problemas cuya solución se puede verificar rápidamente?

Si la respuesta fuera afirmativa, problemas como el Sudoku, el viajante o la factorización de números grandes serían tratables. Si la respuesta es negativa, significaría que siempre habrá problemas que solo podamos abordar con aproximaciones. Este dilema no está resuelto aún y es uno de los grandes problemas abiertos de la matemática, con un millón de dólares de recompensa para quien lo logre.

Relevancia en Inteligencia Artificial

La conexión con la Inteligencia Artificial es directa. Muchos de los problemas de la IA, como la planificación de rutas, el reconocimiento de patrones o los juegos estratégicos, pertenecen a NP o son NP-Hard. Resolverlos exactamente sería impracticable, de ahí la necesidad de heurísticas, metaheurísticas (algoritmos genéticos, enjambre de partículas) y métodos de aprendizaje automático que **nos permiten encontrar soluciones aproximadas pero útiles.**

En definitiva, la IA se vuelve imprescindible porque la realidad está llena de problemas que no están en P. **Allí donde los métodos exactos fallan por el costo computacional, las técnicas inteligentes abren el camino a soluciones viables.**

4. Ejemplos para aterrizar los conceptos

Hasta aquí hemos hablado de clases de complejidad y categorías teóricas. Sin embargo, la mejor forma de fijar estos conceptos es mirar ejemplos concretos que nos permitan distinguir entre lo tratable y lo intratable, entre lo que pertenece a P y lo que está en NP.

Un buen punto de partida es la multiplicación de números grandes. Supongamos que tenemos que multiplicar dos números de diez cifras. Una computadora puede hacerlo en fracciones de segundo. Incluso si aumentamos la cantidad de dígitos a cien o mil, los algoritmos que conocemos permiten realizar la operación de manera eficiente. Este es

un ejemplo clásico de problema en P: el tiempo de resolución crece, pero de forma controlada.

Ahora comparemos esa situación con la factorización de un número grande. Si alguien nos da el número 91 y nos pide que lo descompongamos en factores primos, con un poco de prueba y error podemos llegar a que $91 = 7 \times 13$. El proceso no es inmediato, pero tampoco imposible. Sin embargo, si el número tiene 300 dígitos, la situación cambia radicalmente. La computadora tendría que probar una cantidad inmensa de combinaciones hasta encontrar la factorización. Verificar la solución sigue siendo fácil: basta con multiplicar los factores propuestos y comprobar que el resultado coincide. Pero encontrar esa solución es lo que se vuelve intratable.

Otro caso interesante es el Sudoku. Cuando recibimos un tablero ya resuelto, podemos comprobar en segundos si cumple las reglas: cada fila, columna y subcuadro contiene los números del 1 al 9 sin repeticiones. Esa verificación rápida lo coloca en la clase NP. El verdadero desafío está en resolver un tablero vacío o con pocas pistas. A medida que aumentan las posibilidades, la búsqueda de la solución requiere probar millones de configuraciones. Resolverlo es costoso, pero revisar la solución es sencillo.

El ajedrez nos ofrece otro ejemplo ilustrativo. En teoría, si quisiéramos calcular la "jugada perfecta" desde la posición inicial, tendríamos que explorar un número de posibilidades que crece de forma astronómica con cada movimiento. Esa explosión combinatoria convierte el problema en intratable. No obstante, verificar una secuencia de jugadas propuestas es inmediato. Por eso, en la práctica se utilizan heurísticas y algoritmos aproximados que buscan buenas jugadas, aunque no sean perfectas.

Finalmente, volvamos al célebre problema del viajante. Imaginemos un repartidor que debe visitar veinte direcciones distintas y volver al punto de partida. La cantidad de rutas posibles supera los sesenta trillones. Evaluar todas para encontrar la mejor ruta es imposible en la práctica. Sin embargo, si alguien nos muestra una ruta y nos dice que su longitud es, por ejemplo, 120 kilómetros, comprobar esa distancia es una tarea rápida. Nuevamente, lo difícil no es verificar, sino encontrar.

En todos estos ejemplos se refleja una idea central: **la frontera entre lo que podemos resolver de forma eficiente y lo que no depende de cómo crece el tiempo de cómputo con el tamaño del problema.** La teoría de complejidad traduce esa intuición en clases formales como P, NP, NP-Hard y NP-Complete. Pero lo importante es que, al analizar casos concretos, comprendemos que la Inteligencia Artificial surge justamente porque la mayoría de los problemas que enfrentamos en el mundo real no se encuentran en la categoría de los fáciles.

5. ¿Qué significa "razonamiento" en Inteligencia Artificial?

Hasta ahora hemos visto que los problemas computacionales se dividen entre tratables e intratables, y que muchos de los más interesantes pertenecen a la categoría de los difíciles, como los que están en NP. Surge entonces la pregunta: **¿cómo hace la Inteligencia Artificial para enfrentarse a estos problemas que, en principio, parecen imposibles de resolver en un tiempo razonable?**

Aquí entra en juego el concepto de **razonamiento**. En el ámbito de la IA, hablar de razonamiento no significa pensar como un ser humano en el sentido biológico, sino **aplicar métodos sistemáticos para elegir qué pasos dar en la resolución de un problema**. Es, en definitiva, la capacidad de decidir de manera inteligente cómo proceder frente a una tarea compleja.

Razonamiento exacto y razonamiento aproximado

En algunos casos, es posible usar un razonamiento exacto. Esto implica seguir un algoritmo que garantiza encontrar la solución correcta. Por ejemplo, si queremos resolver un sistema de ecuaciones lineales, podemos aplicar un método algebraico que nos dará la respuesta exacta en tiempo polinomial.

Pero en muchos problemas intratables no tenemos esa posibilidad. Allí aparece el razonamiento aproximado o heurístico. En lugar de buscar la solución perfecta —que tardaría siglos en encontrarse—, **los sistemas de IA se concentran en hallar soluciones buenas y útiles en tiempos razonables**. Estas soluciones no siempre son óptimas, pero permiten avanzar en la práctica.

Heurísticas: pensar con reglas prácticas

Una de las formas más frecuentes de razonamiento en IA es el uso de **heurísticas**. Una heurística es una **regla práctica que guía la búsqueda de soluciones reduciendo el espacio de posibilidades.**

Un ejemplo clásico es el juego del ajedrez. Calcular todas las jugadas posibles es inviable, pero una heurística puede ser: "prefiere las jugadas que centralizan las piezas" o "protege primero al rey". Estas reglas no garantizan la jugada perfecta, pero permiten tomar decisiones inteligentes sin necesidad de analizar todo el universo de posibilidades.

Métodos de razonamiento en IA

El razonamiento en IA se puede manifestar de diferentes maneras:

- **Razonamiento lógico**, basado en reglas formales de la lógica matemática. Es característico de la IA simbólica y de los sistemas expertos.
- **Razonamiento probabilístico**, que trabaja con incertidumbre y calcula probabilidades para orientar decisiones (muy usado en diagnóstico médico y predicción).
- **Razonamiento basado en analogías o casos previos**, donde el sistema compara la situación actual con experiencias pasadas almacenadas en su base de datos.

Cada enfoque busca lo mismo: que la máquina pueda **"pensar" en cómo resolver** un problema sin necesidad de revisar cada posibilidad exhaustivamente.

La importancia del razonamiento en IA

La noción de razonamiento es fundamental porque explica por qué la IA no se limita a aplicar fórmulas rígidas, sino que puede adaptarse, elegir caminos y resolver problemas que, de otro modo, estarían fuera de nuestro alcance computacional.

En resumen, cuando decimos que un sistema de IA "razona", no queremos decir que tiene conciencia, sino que **aplica estrategias inteligentes** para llegar a soluciones útiles en un entorno donde el cálculo puro y exhaustivo sería imposible.

6. Ideas clave de síntesis

Al hablar de problemas computacionales, lo primero que debemos recordar es que todo problema se define por una entrada, un proceso y una salida. Sin embargo, no todos los problemas se comportan de la misma manera frente al cálculo. Algunos son **tratables**, porque se pueden resolver en tiempos razonables, y otros son **intratables**, porque la cantidad de pasos necesarios crece de manera explosiva y los vuelve impracticables en la realidad.

La teoría de la complejidad nos ayuda a clasificar estos problemas. Los de la clase **P** son los que podemos resolver eficientemente con algoritmos polinomiales. Los de la clase **NP** son más complejos: verificar una solución es rápido, pero encontrarla puede ser extremadamente costoso. Dentro de esta categoría, los problemas **NP-Hard** y **NP-Complete** marcan los límites más difíciles de la computación moderna, al punto de que resolver uno de ellos de manera eficiente cambiaría toda la informática.

Ejemplos como la factorización de números grandes, el Sudoku, el ajedrez o el problema del viajante muestran que la diferencia entre lo tratable y lo intratable no es un detalle académico: es una frontera real que determina qué puede hacer una computadora en la práctica.

En este contexto, la Inteligencia Artificial aporta el **razonamiento**: estrategias que permiten encontrar soluciones útiles cuando los métodos exactos fallan por el costo computacional. A través de heurísticas, aproximaciones y aprendizaje automático, la IA nos permite trabajar en escenarios donde el cálculo exhaustivo es imposible, pero aún así necesitamos resultados confiables.

En resumen, la importancia de este tema radica en comprender que la potencia de la IA no reside en resolverlo todo, sino en encontrar formas inteligentes de abordar lo que, de otro modo, sería intratable.