Tecnológico de Monterrey

Complejidad Matemática
de los Problemas Enteros

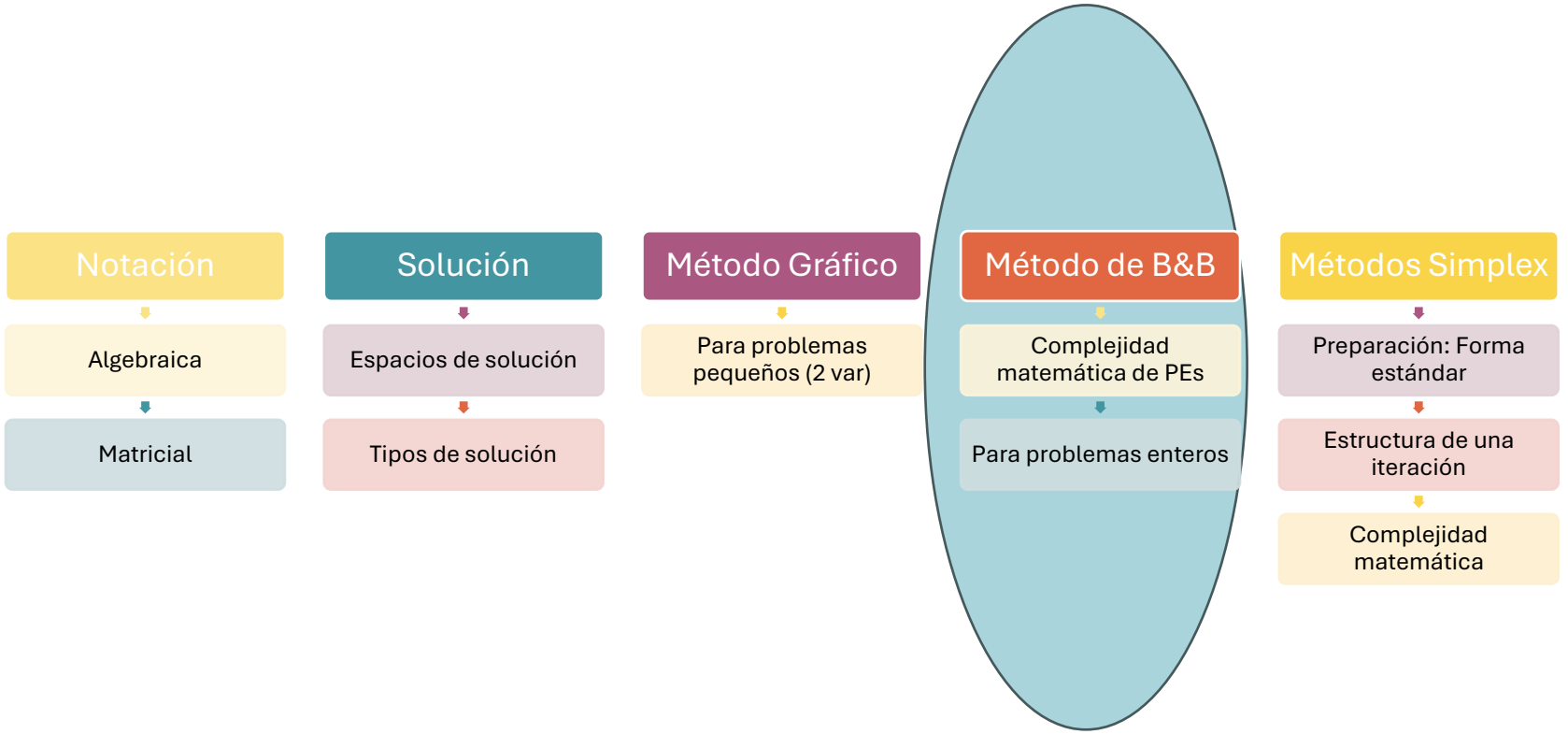
Profesor name

Optimización de Procesos Industriales

Escuela de Ingeniería y Ciencias

Tecnológico de Monterrey

Métodos de Solución - Contenido



Rodolfo Mendoza, Jenny Díaz

Formas de representar un problema

Problema de Optimización

Herramienta de decisión para encontrar la mejor solución factible del problema. Es una representación matemática del problema que cuenta con **variables de decisión**, una **función objetivo** de minimizar o maximizar que cambia en función de los valores de las variables y un **conjunto de restricciones** que se deben cumplir.

Problema Combinatorio

Implican encontrar una agrupación, ordenación o asignación de un conjunto discreto y finito de objetos que satisfagan las condiciones dadas. Las soluciones candidatas son combinaciones de componentes de solución que pueden encontrarse durante un intento de solución pero que no necesitan satisfacer todas las condiciones dadas.

Problema de decision

En la teoría de la computabilidad y la teoría de la complejidad computacional, un problema de decisión es un problema que se puede plantear como una pregunta de sí o no de los valores de entrada.

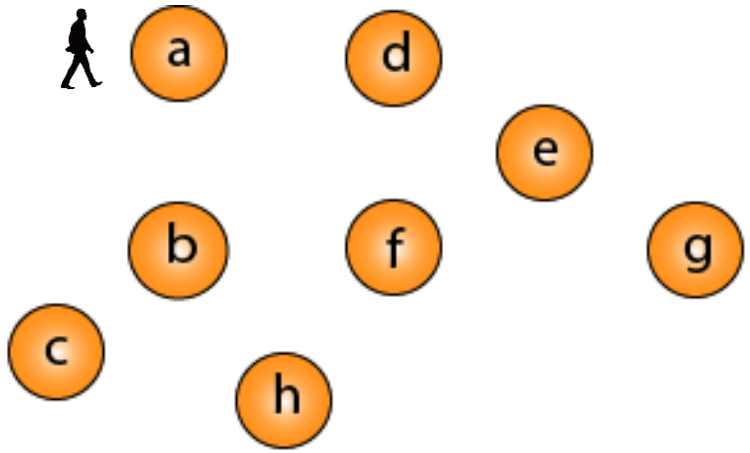
Problema de flujo de red

Clase de problemas computacionales (de optimización combinatoria) en los que la entrada es una red de flujo (un gráfico con capacidades numéricas en sus bordes) y el objetivo es construir un flujo, valores numéricos en cada borde que respeten la capacidad. restricciones y que tienen un flujo de entrada igual al flujo de salida en todos los vértices excepto en ciertas terminales designadas.

Rodolfo Mendoza, Jenny Díaz

Problema del Agente Viajero (Traveling Salesman Problem – TSP)

Dada una lista de ciudades y las distancias entre cada par de ellas, ¿cuál es la ruta más corta posible que visita cada ciudad exactamente una vez y al finalizar regresa a la ciudad origen?



Rodolfo Mendoza, Jenny Díaz

Modelo de optimización del TSP

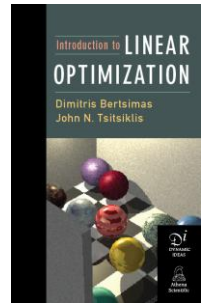
$$x_{ij} = \begin{cases} 1 & \text{the path goes from city } i \text{ to city } j \\ 0 & \text{otherwise} \end{cases}$$

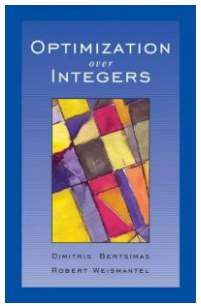
$$\min \sum_{i=1}^{n} \sum_{j \neq i, j=1}^{n} c_{ij} x_{ij} :$$

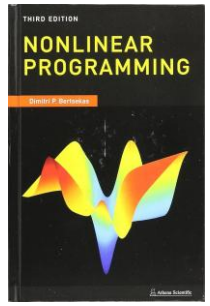
$$\sum_{i=1, i \neq j}^{n} x_{ij} = 1 \qquad j = 1, \ldots, n;$$

$$\sum_{j=1, j \neq i}^{n} x_{ij} = 1 \qquad i = 1, \ldots, n;$$

$$\sum_{i \in Q} \sum_{j \neq i, j \in Q} x_{ij} \leq |Q| - 1 \qquad \forall Q \subsetneq \{1, \ldots, n\}, |Q| \geq 2$$



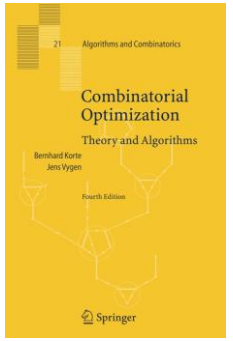


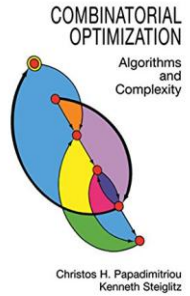


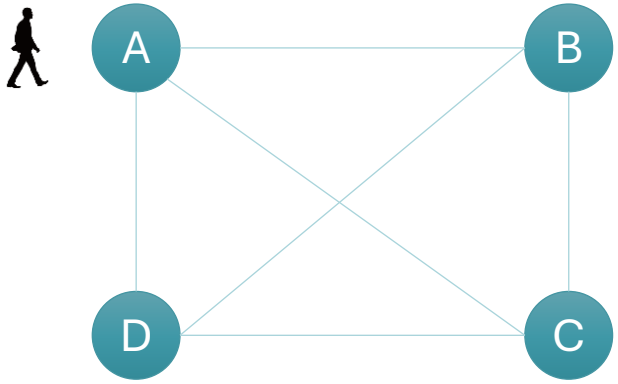
Rodolfo Mendoza, Jenny Díaz

Problema combinatorio

Del conjunto finito de posibles rutas factibles del problema del TSP ¿Cuál es la ruta que minimiza la distancia total recorrida?





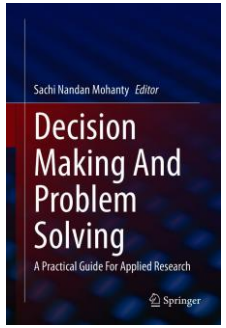


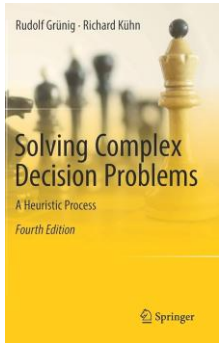
A	-	B	-	C	-	D	-	A
A	-	B	-	D	-	C	-	A
A	-	C	-	B	-	D	-	A
A	-	C	-	D	-	B	-	A
A	-	D	-	B	-	C	-	A
A	-	D	-	C	-	B	-	A

Rodolfo Mendoza, Jenny Díaz

Problema de decisión

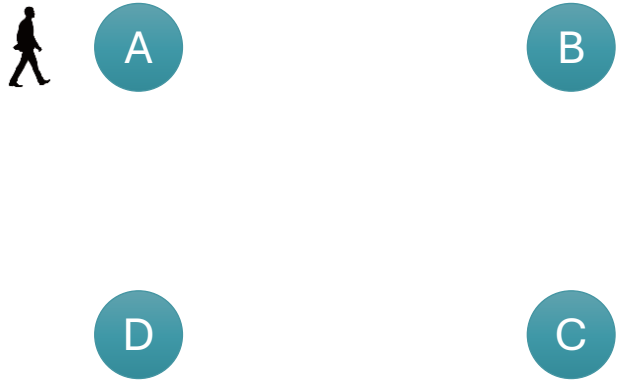
¿Existe una ruta de la que se obtenga la menor distancias posibles?

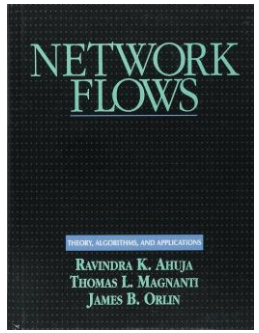


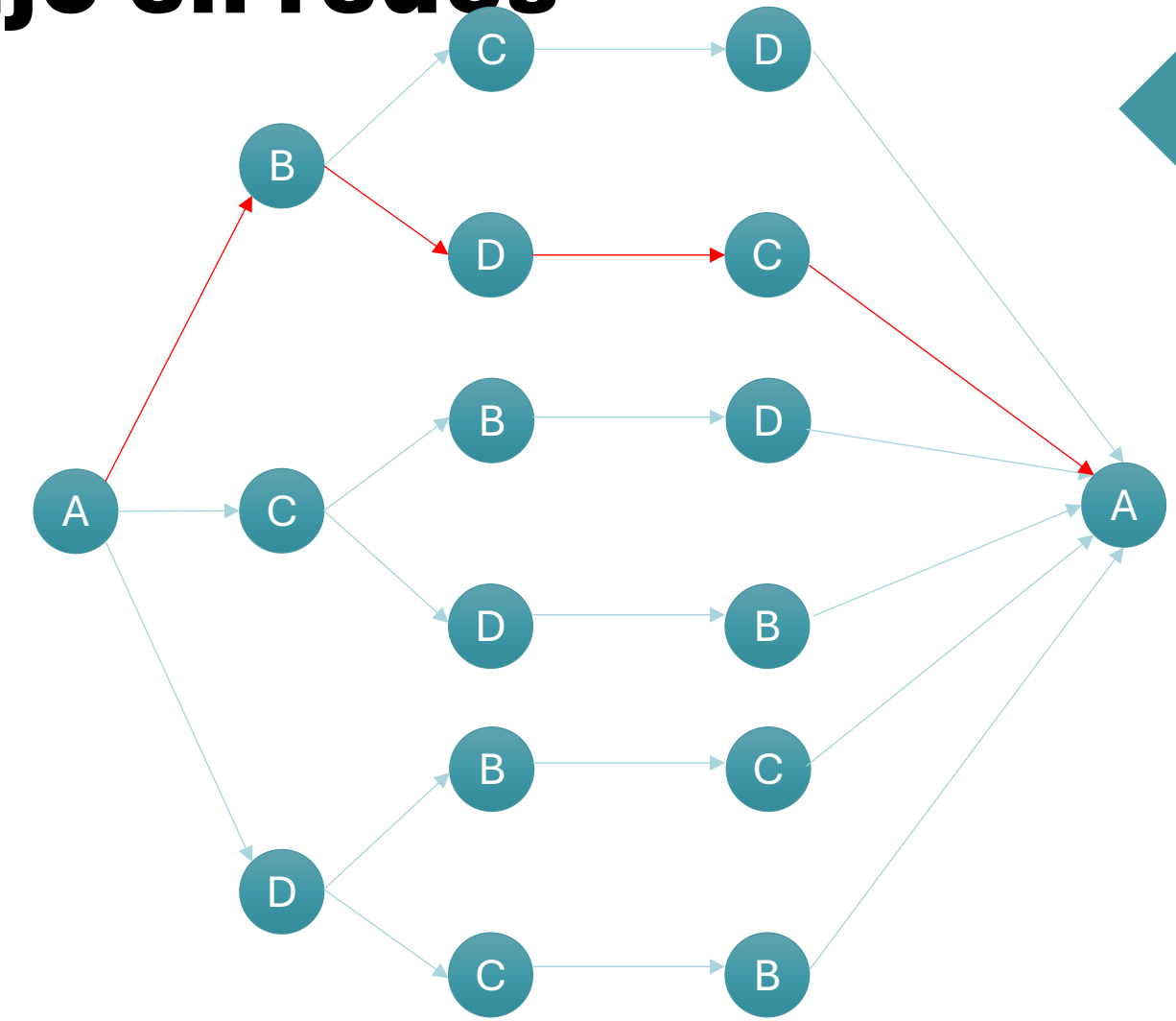


Rodolfo Mendoza, Jenny Díaz

Problema de flujo en redes







Rodolfo Mendoza, Jenny Díaz

Métodos de solución:

Algoritmo exacto:

Los algoritmos exactos son algoritmos que siempre resuelven un problema de optimización de forma óptima.

Algoritmo heurístico:

Un algoritmo heurístico es aquel que está diseñado para resolver un problema de una manera más rápida y eficiente que los métodos tradicionales al sacrificar la optimización, la exactitud, la precisión o la integridad por la velocidad.

Algoritmo metaheurístico:

Una metaheurística es un procedimiento o heurística de alto nivel diseñado para encontrar, generar o seleccionar una heurística que pueda proporcionar una solución suficientemente buena para un problema de optimización.

Algoritmo matemático:

Una característica esencial es la explotación en alguna parte de los algoritmos de características derivadas del modelo matemático.

Rodolfo Mendoza, Jenny Díaz

Métodos de Solución

Algoritmo exacto

- Garantizan resolver un problema de optimización de forma óptima

Algoritmo heurístico

Está diseñado para resolver un problema de una manera más rápida y eficiente que los métodos tradicionales al sacrificar la optimización, la exactitud, la precisión o la integridad por la velocidad.

Algoritmo metaheurístico

Procedimiento o heurística de alto nivel diseñado para encontrar, generar o seleccionar una heurística que pueda proporcionar una solución suficientemente buena para un problema de optimización.

Algoritmo matemático

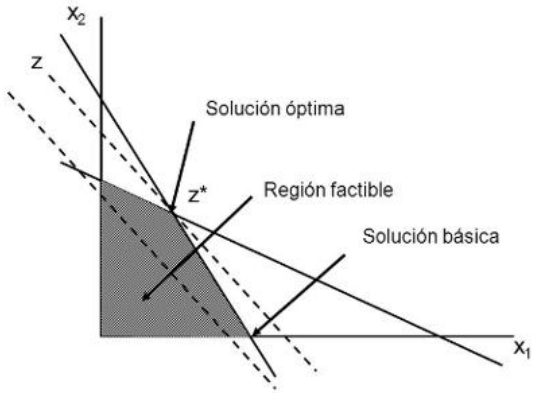
Una característica esencial es la explotación en alguna parte de los algoritmos de características derivadas del modelo matemático.

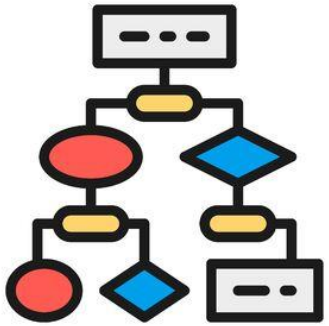
Rodolfo Mendoza, Jenny Díaz

Formas para evaluar la complejidad computacional

- Se puede evaluar la complejidad computacional de un problema.
  - Se considera el mejor algoritmo para resolver el problema.

- Se puede evaluar la complejidad computacional de un algoritmo de solución.
  - Un algoritmo es capaza de resolver uno o varios tipos de problemas.





Rodolfo Mendoza, Jenny Díaz

¿Cómo hacer una evaluación computacional?

Evaluación empírica

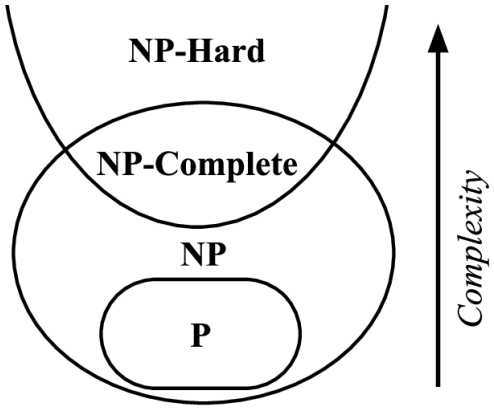
- Se evalúa un problema para diferentes tamaños del problema.
- Se evalúa un problema con diferentes algoritmos.

Notación Big-O

- Se determina la complejidad computación con el número de pasos que se requieren en el algoritmo.
  - ✓ Constante $O(1)$
  - ✓ Lineal $O(n)$
  - ✓ Polinomial $O(n^k)$
  - ✓ Exponencial $O(c^n)$

Clasificación del problema

- Se determina el tipo de problema de acuerdo con su complejidad.



Medición empírica de la complejidad

- Se dice que el número de pasos realizados por el algoritmo es la suma total de todos los pasos que realiza.
- El número de pasos que toma un algoritmo, que en gran medida determina el tiempo que requiere, diferirá de una instancia del problema a otra.
- Aunque un algoritmo puede resolver algunos casos "**buenos**" del problema rápidamente, puede llevar mucho tiempo resolver algunos casos "**malos**".

Análisis empírico	• Para estimar cómo se comportan los algoritmos en la práctica
Análisis de casos promedio	• Para estimar el número esperado de pasos que toma un algoritmo
Análisis del peor de los casos	• Proporciona límites superiores en la cantidad de pasos que un algoritmo dado puede tomar en cualquier instancia de problema

Rodolfo Mendoza, Jenny Díaz

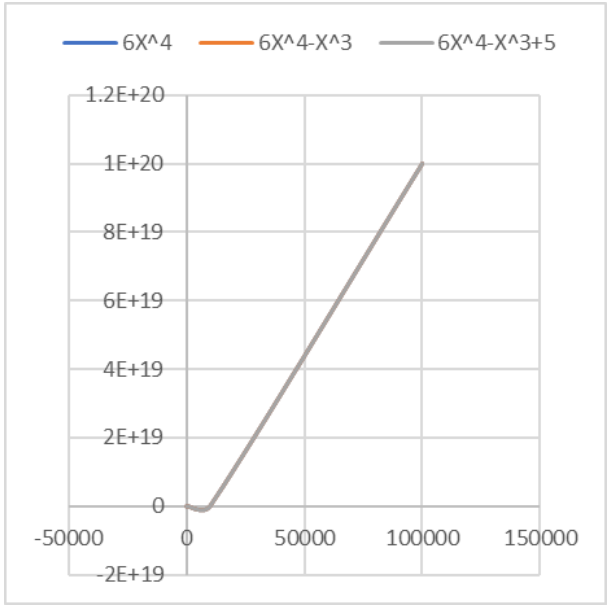
Notación Big-O (Wood, 1987)

- La complejidad de los problemas computacionales se puede discutir analizando un algoritmo y luego considerando cuántos recursos de las máquinas se requieren para las soluciones.
- Para comparar las complejidades de tiempo o espacio de los algoritmos, generalmente nos interesa solo su **orden**, es decir, se ignoran las constantes multiplicativas y los términos de orden inferior. La notación **Big-O** se usa para este propósito.
- Ejemplo:

$$f(x) = 6x^4 - 2x^3 + 5,$$

$$\rightarrow \quad g(x) = x^4 \quad \rightarrow \quad O(x^4).$$

$$f(x) \text{ is } O(g(x)) \text{ as } x \to \infty$$



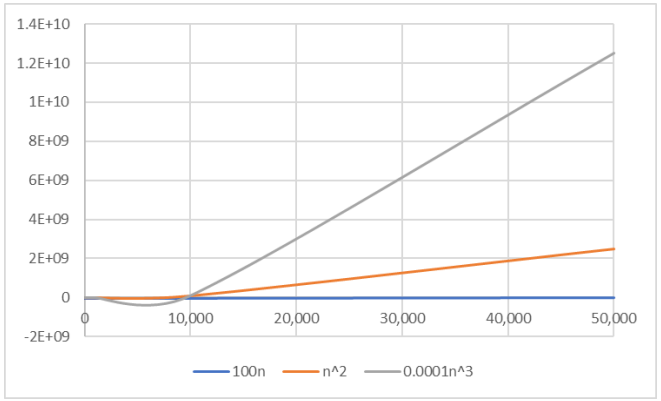
Rodolfo Mendoza, Jenny Díaz

Ejemplo

- Si el tiempo de ejecución de un algoritmo es:

$$100n + n^2 + 0.0001n^3$$

- Para n<100, el primer termino domina al segundo



- Para n >= 10,000, el segundo

n	100n	n^2
1	100	1
50	5,000	2,500
100	10,000	10,000
101	10,100	10,201

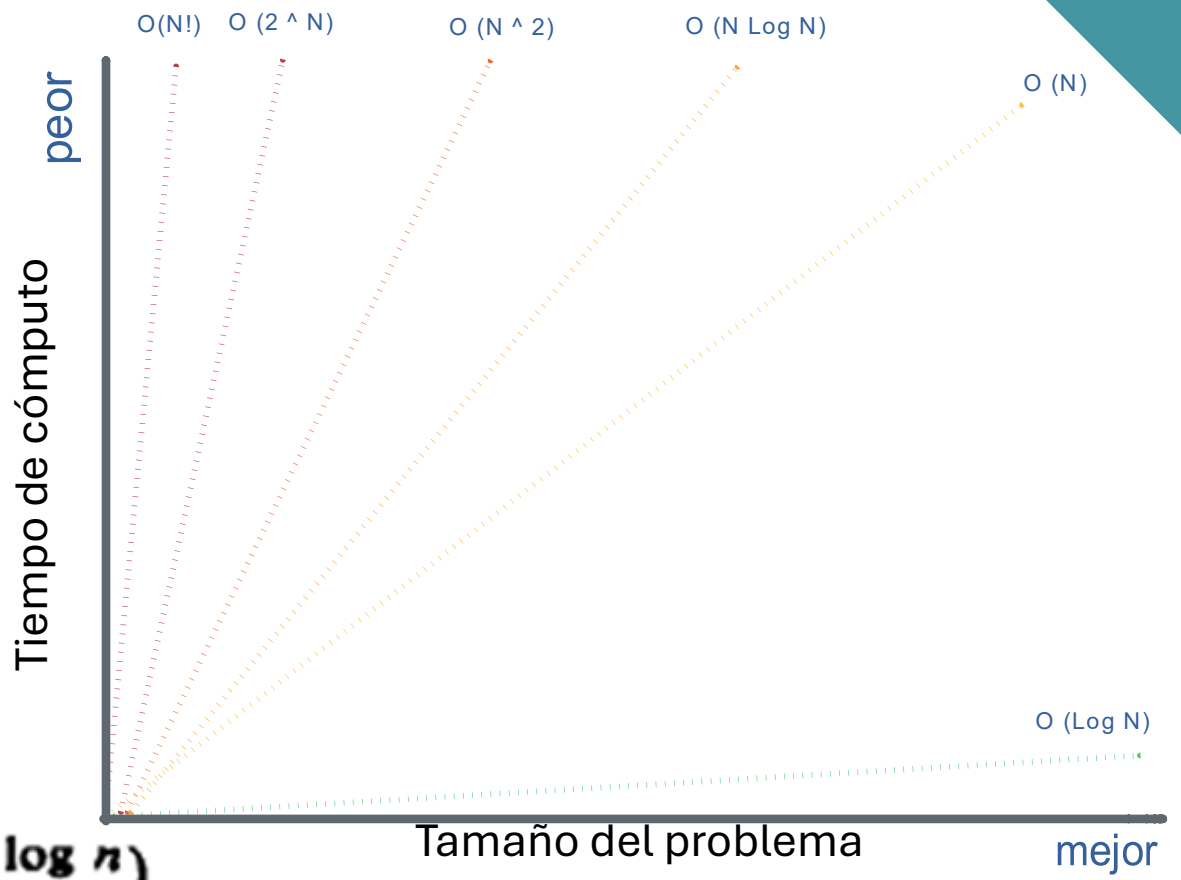
n	100n	n^2	0.0001n^3
1	100	1	0
100	10,000	10,000	100
1,000	100,000	1,000,000	100,000
10,001	1,000,100	100,020,001	100,030,003

- Por lo tanto, la complejidad del algoritmo es: $O(n^3)$

Rodolfo Mendoza, Jenny Díaz

Complejidad del tiempo

- Complejidad del tiempo
  - Tiempo constante $O(1)$
  - Tiempo lineal $O(n)$
  - Tiempo polinomial $O(n^k)$
  - Tiempo exponencial $O(c^n)$ $O(n!), O(n^{\log n})$



Algoritmos de tiempo polinomial y exponencial

- Se dice que cualquier algoritmo de este tipo es un algoritmo de **tiempo polinomial**. Algunos ejemplos de límites de tiempo polinómicos son:

$$O(n^2),\ O(nm),\ O(m + n \log C),\ O(nm) \quad \log(n^2/m)), \text{and } O(nm + n^2 \log U)$$

- Se dice que un algoritmo es un algoritmo de **tiempo exponencial** si su tiempo de ejecución en el peor de los casos crece como una función que no puede ser acotada polinomialmente por la longitud de entrada.

$$O(2^n),\ O(n!), O(n^{\log n}).$$

n	$\log n$	$n^{0.5}$	n^2	n^3	2^n	$n!$
10	3.32	3.16	10^2	10^3	10^3	3.6×10^6
100	6.64	10.00	10^4	10^6	1.27×10^{30}	9.33×10^{157}
1000	9.97	31.62	10^6	10^9	1.07×10^{301}	$4.02 \times 10^{2,567}$
10,000	13.29	100.00	10^8	10^{12}	$0.99 \times 10^{3,010}$	$2.85 \times 10^{35,659}$

Ejemplo

```
int[] intArray = {1,5,10,20,2,6,12};
int min = Integer.MIN_VALUE;
int max = Integer.MAX_VALUE;

for (int x : intArray) {
    if (x < min ) min = x;
    if (x > max) max = x;
}
```

```
int[] arrayA = {1,5,10,20,2,6,12};
int[] arrayB = {100,200,300,400,500,600};

for (int x : arrayA) {
    for (int y : arrayB) {
        System.out.println(x + y);
    }
}
```

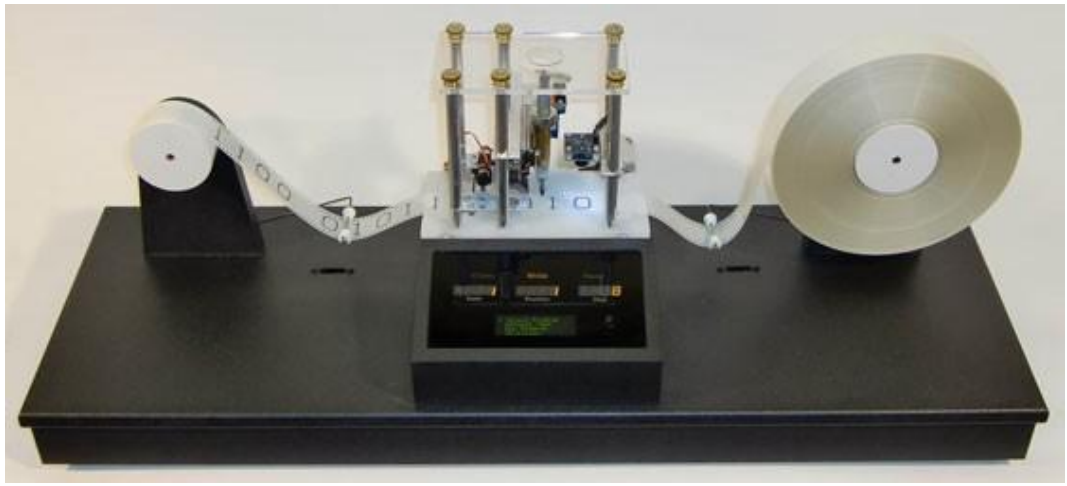
Teoría de la complejidad

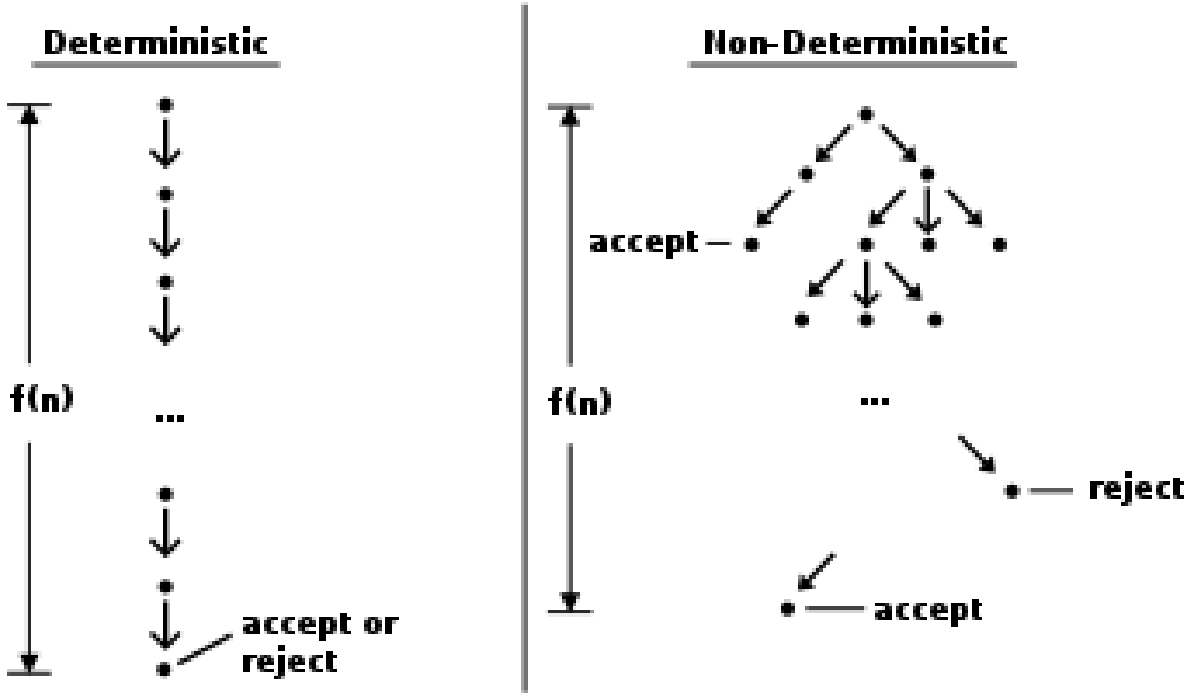
Usamos modelos para medir la complejidad temporal de los algoritmos y la dureza de los problemas independientemente de una máquina específica que ejecuta los algoritmos.

- **Máquina de Turing Determinista (DTM)**
- **Máquina de Turing no determinista (NTM)**

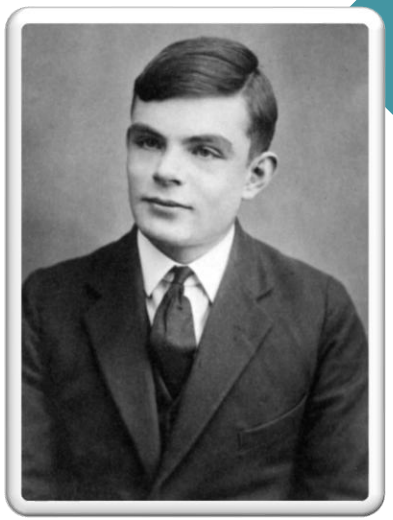
Estos modelos intentan definir el tiempo de ejecución en función del tamaño del problema.

Los problemas se miden como el **número de pasos que se deben tomar para resolver una instancia del problema** en cuestión que aparentemente es independiente de cualquier computadora o máquina específica.



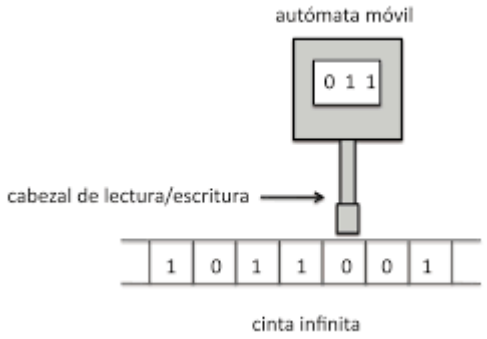


Máquina de Turing



Es un dispositivo que manipula símbolos sobre una tira de cinta de acuerdo con una tabla de reglas. A pesar de su simplicidad, una máquina de Turing puede ser adaptada para simular la lógica de cualquier algoritmo de computador y es particularmente útil en la explicación de las funciones de una CPU dentro de un computador.

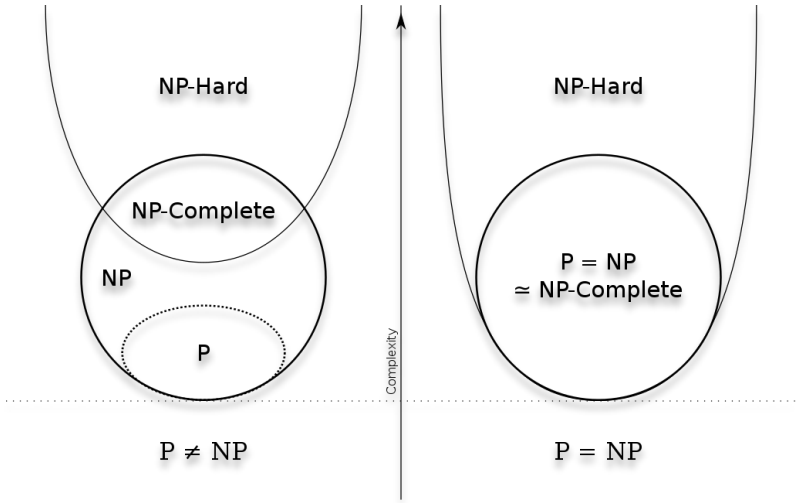
Originalmente fue definida por el matemático inglés Alan Turing como una «**máquina automática**» en 1936 en la revista *Proceedings of the London Mathematical Society*. La máquina de Turing no está diseñada como una tecnología de computación práctica, sino como un dispositivo hipotético que representa una máquina de computación. Las máquinas de Turing ayudan a los científicos a entender los límites del cálculo mecánico.



Clases de complejidad para los problemas

Muchas clases de complejidad importantes se definen delimitando el tiempo o el espacio utilizado por un algoritmo. Varias clases de complejidad importantes definidas de esta manera se explican a continuación.

- **P (deterministic polynomial time)**
- **NP (nondeterministic polynomial time)**



P es la clase de problemas que se pueden resolver con una máquina de Turing determinista en tiempo polinomial y **NP** es la clase de problemas que se pueden resolver con una máquina de Turing no determinista en tiempo polinomial.

A menudo se dice que P es la clase de problemas que una computadora determinista puede resolver "***rápidamente***" o "***eficientemente***", ya que la complejidad temporal de resolver un problema en **P** aumenta relativamente lentamente con el tamaño de entrada.

Una característica importante de la clase **NP** es que puede definirse de manera equivalente como la clase de problemas cuyas soluciones son verificables por una máquina determinista de Turing en tiempo polinomial.

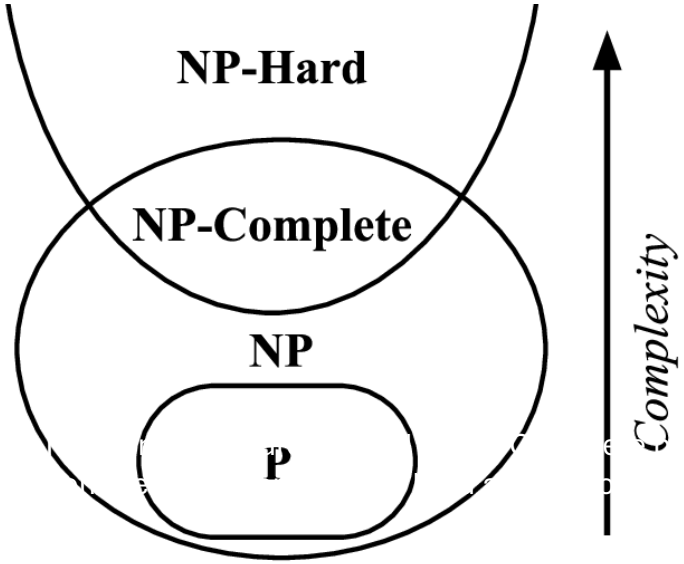
Rodolfo Mendoza, Jenny Díaz

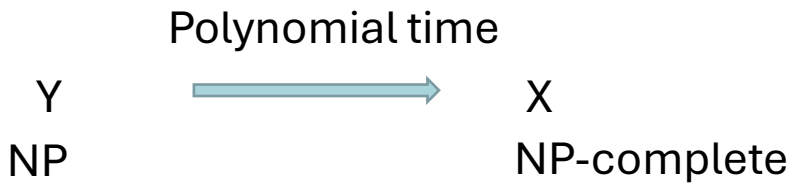
Problemas NP-Completos

En la teoría de la complejidad computacional, un problema es **NP-completo** cuando:

Es un problema para el cual la corrección de cada solución se puede verificar rápidamente y un algoritmo de búsqueda de fuerza bruta puede encontrar una solución probando todas las soluciones posibles.

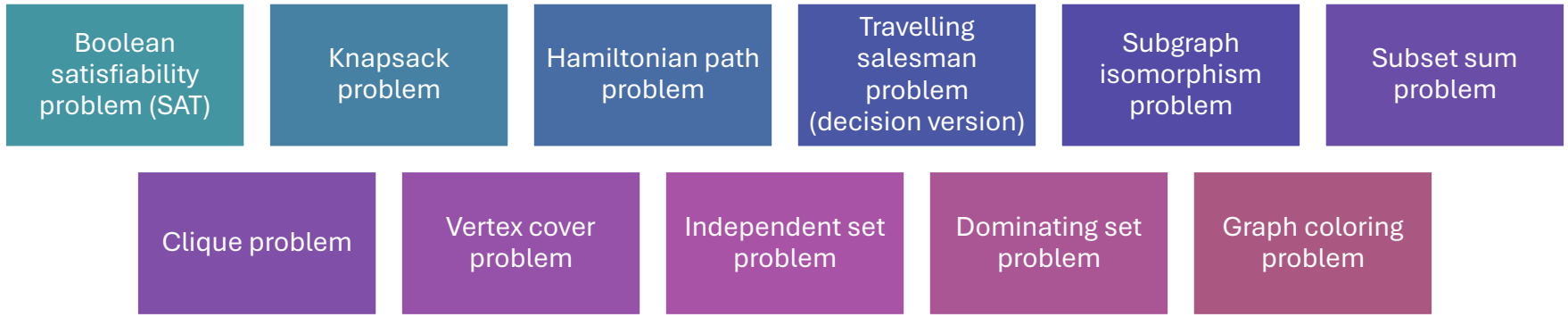
Un problema X es NP-Completo si existe un problema NP Y, tal que Y es reducible a X en tiempo polinomial. Los problemas NP-Completos son tan difíciles como los problemas NP. Un problema es NP-Completo es parte tanto de NP como de NP-Hard. Una máquina de Turing no determinista puede resolver el problema NP-Completo en tiempo polinomial.





Problemas NP-completos

- Estos se denominan problemas NP-Intermedios y existen si y solo si P≠NP.
- Es el más difícil de los problemas en los que las soluciones se pueden verificar rápidamente, de modo que si realmente pudiéramos encontrar soluciones de algún problema NP-Completo rápidamente, podríamos encontrar rápidamente las soluciones de cualquier otro problema para el cual una solución una vez dada es fácil de verificar .

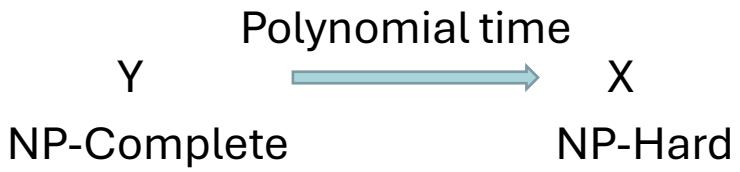


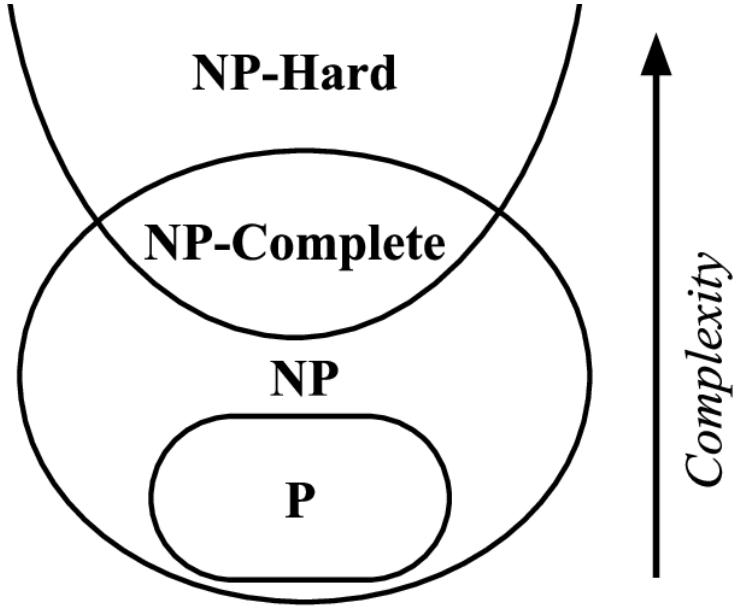
Rodolfo Mendoza, Jenny Díaz

NP-Duros (NP-Hard)

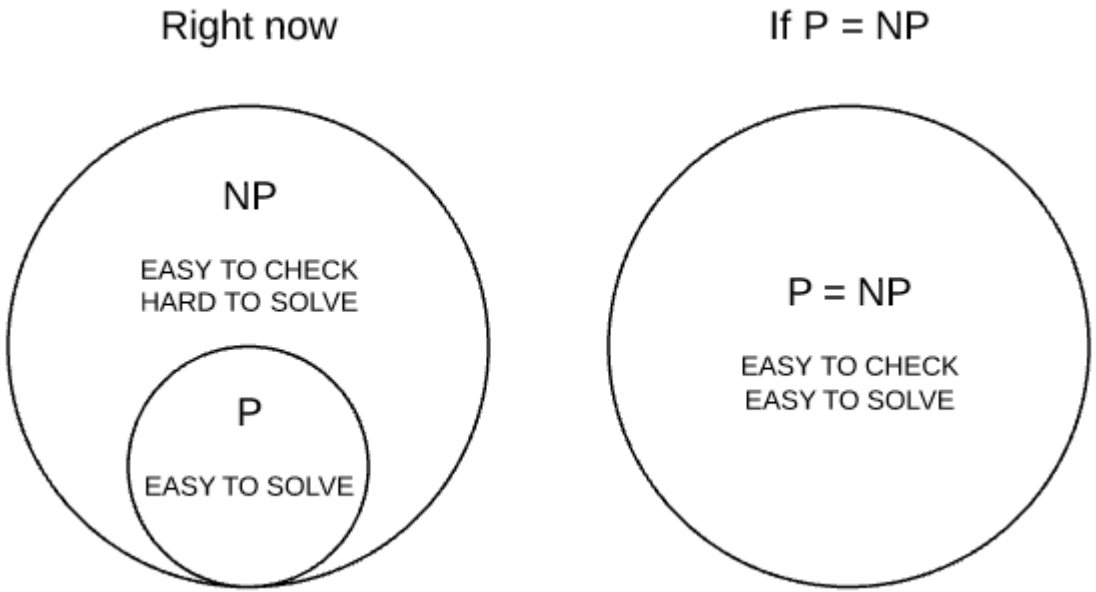
En la teoría de la complejidad computacional, la dureza NP (dureza de tiempo polinomial no determinista) es la propiedad definitoria de una clase de problemas que son informalmente "al menos tan difíciles como los problemas más difíciles en NP".

Los problemas NP-Hard (por ejemplo, X) se pueden resolver si y solo si hay un problema NP-Completo (por ejemplo, Y) que se puede reducir a X en tiempo polinomial.





Rodolfo Mendoza, Jenny Díaz



https://www.youtube.com/watch?v=UR2oDYZ-Sao

Rodolfo Mendoza, Jenny Díaz