uc3m | Universidad Carlos III de Madrid

Grado en Ingeniería Informática

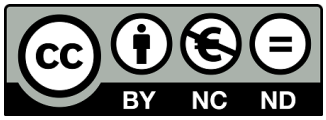
2021-2022

*Apuntes*

# Teoría avanzada de la computación

---

Jorge Rodríguez Fraile[1]





[1]Universidad: 100405951@alumnos.uc3m.es | Personal: jrf1616@gmail.com

# ÍNDICE GENERAL


1. INFORMACIÓN . . . . . 3
1.1. Profesores . . . . . 3
1.2. Sistema de evaluación . . . . . 3
1.3. Examen Practicas . . . . . 3
2. ANÁLISIS DE LA COMPLEJIDAD DE LOS ALGORITMOS . . . . . 5
2.1. Introducción . . . . . 5
2.2. Paso Base . . . . . 6
2.3. Complejidad de los Algoritmos . . . . . 6
2.3.1. Ejemplo . . . . . 8
2.4. Algoritmos Recursivos . . . . . 9
2.4.1. Despliegue de Recurrencias . . . . . 9
2.4.2. Resolución General de la Ecuación de Recurrencia . . . . . 10
2.4.3. Fractales . . . . . 12
2.5. Cotas asintóticas . . . . . 12
2.5.1. Cota Superior Asintótica . . . . . 13
2.5.2. o (o pequeña / ómicron minúscula) . . . . . 13
2.5.3. Cota Inferior Asintótica . . . . . 14
2.5.4. Cota Ajustada Asintótica . . . . . 14
2.6. Tratabilidad . . . . . 14
3. TEORÍA DE LA COMPUTABILIDAD . . . . . 15
3.1. Problema de Satisfabilidad Booleana: SAT . . . . . 15
3.1.1. Algoritmo sencillo . . . . . 16
3.1.2. Aplicaciones . . . . . 16
3.2. Clique . . . . . 16
3.2.1. Aplicaciones . . . . . 16
3.3. Decidibilidad . . . . . 17
3.4. Problemas, lenguajes y máquinas de cómputo . . . . . 18
3.4.1. Notación . . . . . 18

3.4.2. Lenguajes como modelos de problemas . . . . . 18
3.4.3. Función recursiva . . . . . 19
3.4.4. Equivalencia de lenguaje y tipos de problemas . . . . . 19
3.5. Máquina de Turing como modelo computacional . . . . . 19
3.5.1. Máquina de Turing binaria . . . . . 20
3.5.2. Generalizaciones de las Máquinas de Turing . . . . . 20
3.5.3. Máquina de Turing No Determinista (MTND) . . . . . 20
3.5.4. Salidas de la Máquina de Turing . . . . . 21
3.6. Decidibilidad . . . . . 22
3.6.1. Nociones formales . . . . . 22
3.6.2. Máquina de Turing Universal . . . . . 23
3.6.3. Algunos teoremas importantes . . . . . 23
3.6.4. Lenguaje Diagonal . . . . . 24
3.6.5. Resumen . . . . . 25
3.6.6. Reducción de un problema A a otro problema B . . . . . 25
4. TEORÍA DE LA COMPLEJIDAD COMPUTACIONAL . . . . . 27
4.1. La Máquina de Turing y medida de la Complejidad . . . . . 27
4.2. Reducción entre problemas . . . . . 27
4.3. Clases de Problemas . . . . . 28
4.3.1. Complejidad Temporal . . . . . 29
4.3.2. Complejidad Espacial . . . . . 33
5. MODELOS DE COMPUTACIÓN. . . . . 35
5.1. Automatas Finitos (AF) . . . . . 35
5.2. Automata de Pila (AP). . . . . 36
5.3. Generalizaciones de las Máquinas de Turing . . . . . 36
5.4. Random Access Machine Model (RAM) . . . . . 37
5.4.1. Random Access Stored Program (RASP) . . . . . 37
5.5. Automatas Celulares . . . . . 38
5.5.1. Variedades . . . . . 38
5.5.2. Clasificación . . . . . 39
5.5.3. AC Unidimensional . . . . . 39

5.6. L-Systems . . . . . 40
5.6.1. L-Systems vs. Gramáticas de Chomsky . . . . . 41
5.7. Máquinas de Turing . . . . . 41
5.7.1. Origen . . . . . 41
5.7.2. Utilidad . . . . . 41
5.7.3. Modelo de Computación Universal . . . . . 42
5.7.4. Eficiencia . . . . . 42
5.7.5. Coste Computacional con MT . . . . . 43
5.7.6. Variantes del Modelo de Máquina de Turing . . . . . 44

# ÍNDICE DE FIGURAS

3.1 Diagrama Decidibilidad . . . . . 17
3.2 Codificación de las transiciones . . . . . 21

# ÍNDICE DE TABLAS

2.1 Nombres de los órdenes de complejidad . . . . . 12

3.1 Equivalencia de lenguajes y problemas . . . . . 19

5.1 Diferencias finitas . . . . . 43

# 1. INFORMACIÓN

## 1.1. Profesores

Magistral: Jose Luis Mira Peidro, jmira@inf.uc3m.es

Prácticas: Juan Manuel Alonso Weber, jmaw@ia.uc3m.es

## 1.2. Sistema de evaluación

- 34 % Examen Final (nota mínima 4/10)
- 66 % Evaluación continua (EC)
  - 11 % Práctica 1
  - 11 % Práctica 2
  - 11 % Práctica 3
  - 33 % Examen sobre las prácticas

## 1.3. Examen Practicas

Para el examen final no entra la recursividad no homogénea.

Se pregunta sobre las 2 primeras prácticas

De AKS se pregunta del material que se entregó, para verificar que se sabe de todo.

Mirar además de la práctica, el material y las cuestiones.

- Ejem. Describir brevemente el algoritmo.
- Ejem. Saber de la condición suficiente.

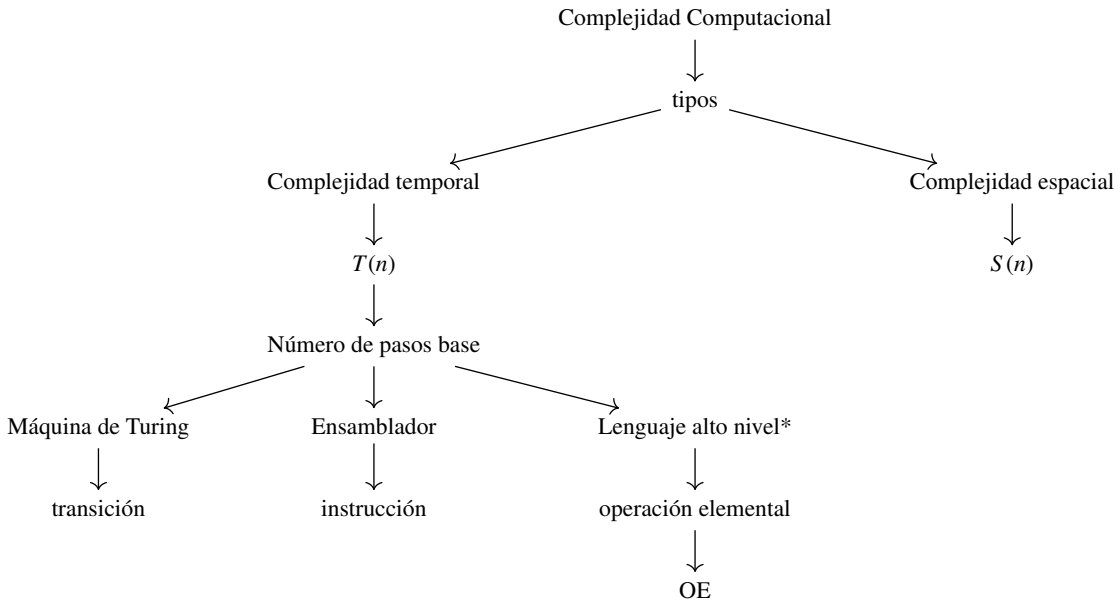
Se puede preguntar en AKS sobre alguna variación sobre el problema, para saber cómo se comportaría, de manera que se sabe cómo se procedió en el original.

Habrá preguntas más de tipo test y preguntas de desarrollar.

Ventajas de AKS sobre otros algoritmos.

No entra la tercera práctica.

# 2. ANÁLISIS DE LA COMPLEJIDAD DE LOS ALGORITMOS




* Estructuras de los algoritmos siempre se pueden considerar como estructuras de tipo Secuencial, Iterativo y Condicional, esto determina la operación elemental.

## 2.1. Introducción

**Complejidad Computacional:** Recursos requeridos durante la ejecución de un algoritmo que da respuesta a un problema. Hay 2 tipos:

- Temporal: Número de pasos base de ejecución de un algoritmo para resolver un problema.
- Espacial: Memoria utilizada para resolver un problema. Registros, Memoria caché y Memoria RAM, etc.

**Problema:** Una función entre el espacio de instancias y el espacio de respuestas.

- Ejem. La ecuación de grado 2 tiene raíces cuadradas? $P(x^2 + 1 = 0)$ = No

$$I \rightarrow P(I)$$

**Problema de decisión:** Tiene 2 respuestas, si o no.

$$I \rightarrow \{Si,\ No\}$$

**Instancia de un problema:** Es la especificación exacta de los datos de un problema para un caso particular.

- **Tamaño de instancia:** Número de datos de la instancia.
- Ejem. Independiente, $x^2$ y $x$ son los 3 parámetros de una instancia de un problema de ecuación de segundo grado.

**Algoritmo:** Conjunto de instrucciones que garantiza encontrar una solución correcta para cualquier instancia en un número finito de pasos.

La complejidad puede hacer referencia a un algoritmo o un problema.

- **Algoritmo:** Resolver las instancias de un problema.
  Medida de número de pasos base requeridos para la PEOR instancia de tamaño n.
  Relación entre n y el número de pasos.
- **Problema:** Es el objeto de estudio de la Teoría de la Complejidad Computacional.

### 2.2. Paso Base

Se pueden modelar los algoritmos mediante:

1. Construcción matemática denominada Máquina de Turing → Transición / Tabla de transiciones.
2. Código Ensamblador → Instrucción / Conjunto estructurado de instrucciones.
3. Lenguaje de alto nivel → Operación Elemental / Conjunto estructurado de instrucciones.
   - Suma, Restar División, Multiplicación, etc.

### 2.3. Complejidad de los Algoritmos

Se expresa como: $T(n)$, número de pasos peor caso con entrada n.

La complejidad computacional puede referirse también a los recursos de espacio necesarios según el tamaño de la entrada. Así tenemos que:

- Tiempo de ejecución (complejidad temporal) $T(n)$
- Espacio necesario para los datos (complejidad espacial) $S(n)$

Todo algoritmo puede ser implementado con 3 estructuras: Secuencial, Condicional e Iterativa.

**Operación Elemental (OE)**

- Comparación
- Asignación variable
- Acceso a estructura básica
- Llamada función
- Retorno función

**Estructura Secuencial:** Cada I es una instancia o bloque, se calcula:

- El de todos: $T(I_1; I_2; \ldots; I_n) = T(I_1) + T(I_2) + \cdots + T(I_n)$
- General: $T = \text{máx}\{T(I_1), T(I_2), \ldots, T(I_n)\}$ El T es el más alto, el del peor caso/bloque.
- Para los programas es la suma de todas las estructuras: $T = T_1 + T_2 + \cdots + T_k = \text{máx}\{T_1, T_2, \ldots, T_k\}$

**Estructura Condicional:**

- if (Condición) then BloqueThen else BloqueElse
- $T = T(Condicion) + \text{máx}\{T(BloqueThen), T(BloqueElse)\}$

**Estructura Iterativa:**

- while(C) S - $T = C + n_{iter}(C + S)$
- for(A C B) S - $T = A + C + n_{iter}(C + B + S)$
  - A: Inicialización
  - C: Condición
  - B: Incremento
  - S: Bloque

**Llamada a Procedimiento, Función o Método:**

- $funcion(p_1, p_2, \ldots, p_k)f$
- 1 (por la llamada) + tiempo de evaluación de los parámetros + tiempo que tarde en ejecutarse el «cuerpo» de la función.
- $T = 1 + T(p_1) + T(p_2) + \cdots + T(p_n) + T(f)$

No se contabiliza la copia de los argumentos a la pila de ejecución, salvo estructuras complejas (vector, registros) que se pasen por valor.

El paso por referencia y paso de punteros, no se contabiliza.

### 2.3.1. Ejemplo

```
for (int i = 0, acum = 0; i < n; i++)
  acum += i;
```

$T(n) = A + C + n(C + B + S) = 2 + 1 + n(1 + 2 + 2) = 3 + 5n$

```
if (x < 5)
  for (int i = 0; i < n; i++)
    acum += i;
else
  acum = 3;
```

$T(n) = 1 + \text{máx}\{1 + 1 + n(1 + 2 + 2); 1\} = 1 + \text{máx}\{2 + 5n; 1\} = 3 + 5n$

```
for (int i = 0, acum = 0; i < n; i++)
  for (int j = 0; j < i/2; j++)
    acum += i;
```

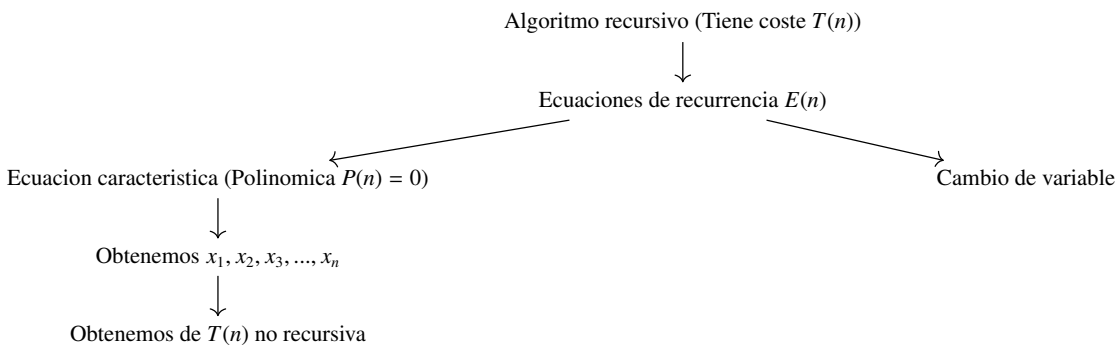
$T(n) = 2 + 1 + n(1 + 2 + [1 + 2 + \frac{n}{2}(2 + 2 + 2)]) = 3 + n(3 + 3 + 3n) = 3 + 6n + 3n^2$

```
x = 5;
while (x < N)
  x = 3 * x;
```

$T(n) = 1 + 1 + p(1 + 2) = 2 + 3p$

$5 \cdot 3^p < n; p \log 3 < \log \frac{n}{7}; p = \frac{\log \frac{n}{5}}{\log 3}$

## 2.4. Algoritmos Recursivos




El coste ($T(n)$) de un algoritmo recursivo será también recursivo.

$T(n) = E(n)$ En la expresión

- $E$ aparece la propia función $T$.
- El objetivo es encontrar $T(n) = f(n)$, sin funciones $T$.

Resolver la ecuación de recurrencia. Dos métodos:

1. Despliegue de la recurrencia
2. Resolución general de recurrencias
   *a*) Recurrencia homogénea (ecuación característica)
   *b*) Recurrencia útiles (fórmulas maestras)
   *c*) Recurrencias no homogéneas y no lineales (no se tratan en este curso)

### 2.4.1. Despliegue de Recurrencias

Aplicar varias veces la fórmula recurrente hasta obtener una fórmula general que relaciona la función para el tamaño original con otros tamaños menores. A partir de esta fórmula se obtiene otra que la relaciona con el caso base.

Pasos:

1. Ecuación ($T(n) = ...$)
2. Despliegue, ($T(n-1), T(n-2), etc$)
3. Aplicaciones del caso base para $n = 0, ..., k \;\; T(0) = a, T(1) = b, ...$
4. Resolución

No debe aplicarse este método si aparecen varios términos recurrentes, como es el caso de la serie de Fibonacci.

Se parte de un caso base y se va sustituyendo dentro de la ecuación el propio término, de manera que cuando vamos sustituyendo y resolviendo la ecuación hasta llegar a un caso en que podamos aplicar el caso base. En ese punto podremos sustituir nuestro caso base en la ecuación y quitarnos completamente la recurrencia, T().

Lo que buscamos es que al desdoblar los términos constantemente encontremos una regularidad. En ese momento hacemos que una variable valga un valor que podamos utilizar para usar el caso base, normalmente T(1).

Ejem. Factorial

- $T(1) = 1$ *y* $T(n) = T(n-1) + 1$
- $T(n) = T(n-2)+1+1 = T(n-2)+2 = T(n-3)+1+2 = T(n-3)+3 = T(n-k)+k$
- Sea $k = n-1$ $T(n) = T(n-(n-1)) + n - 1 = T(1) + n - 1 = n$

### 2.4.2. Resolución General de la Ecuación de Recurrencia

$c_n T(n) + c_{n-1} T(n-1) + ... + c_{n-k} T(n-k) = b$, expresión en n.

Si $b = 0$, se dice que es Ecuación de Recurrencia Lineal (de orden k) HOMOGÉNEA

Los términos de las ecuaciones de recurrencia están afectados por coeficientes. Debemos encontrar una expresión no recursiva de $t$.

No nos ocuparemos de ecuaciones en las que el coeficiente de $T(n-k)$, i.e. $c_{n-k}$ es variable (depende de n), trataremos solo con: ECUACIONES DE RECURRENCIA LINEALES CON COEFICIENTES CONSTANTES.

#### Ecuación de Recurrencia Lineal Homogenea

Método de la Ecuación Característica.

- A una ecuación de recurrencia de orden k, mediante un cambio de variable, se le asocia una ecuación polinómica de grado k (la ecuación característica).
- Las soluciones de la ecuación polinómica proporcionan las soluciones de la ecuación de recurrencia.
- Solución general de la ecuación de recurrencia:
  - Es combinación lineal de las soluciones básicas.
  - Las constantes de la combinación lineal se calculan imponiendo los casos base.

- Con ello obtenemos la solución (no recursiva) de la recurrencia $T(n)$

Cambio de variable: $T(n) = x^n$, se obtiene la ecuación característica $c_n x^n + c_{n-1} x^{n-1} + ... + c_{n-k} x^{n-k} = 0$ siendo $r_1, r_2, ..., r_k$ las raíces reales o complejas.

A una ecuación de recurrencia de orden k, mediante un cambio de variable, se le asocia una ecuación polinómica de grado k (la ecuación característica).

Lo que hacemos tras la sustitución es obtener las raíces, que pueden ser distintas $x = 2$ y $x = 3$ o iguales $x = 1, (x-1) \cdot (x-1)$.

- Para las raíces distintas se aplica:

$$T(n) = \sum_{1}^{k} b_i r_i^n,\ b_i, r_i \in \mathbb{R}$$

  $r_1$ y $r_2$ son raíces diferentes, $T(n) = b_1 r_1^n + b_2 r_2^n$ es solución.

- Una raíz con multiplicidad mayor que 1:

$$T(n) = \sum_{i=1}^{m} b_i n^{i-1} r_1^n + \sum_{i=m+1}^{k} b_i r_{i-m+1}^n,\ b_i, r_i \in \mathbb{R}$$

  $r_1$ son las raíces de la ecuación característica y $m$ la multiplicidad cuando es mayor que 1.
  $r$ raíz doble, $T(n) = b_1 n r^n + b_2 r^n$ es solución.

Se cambia el $T(n)$, $T(n-1)$, ... por variables $x$ que dependan del $n$.

El grado de la variable siempre es un grado menos que el número de términos recurrentes y cada uno de valor menor reduce el grado.

Ejem. $T(n) = T(n-1) + T(n-2) \rightarrow x^2 = x + 1$

- Sus raíces son $r = \frac{1 \pm \sqrt{5}}{2}$, estos términos se sustituirían en las ecuaciones de arriba quedando: $T(n) = b_1 \left(\frac{1+\sqrt{5}}{2}\right)^n + b_2 \left(\frac{1-\sqrt{5}}{2}\right)^n$
- Sabiendo $T(0) = 0$ y $T(1) = 1$
  - $T(0) = b_1 \left(\frac{1+\sqrt{5}}{2}\right)^0 + b_2 \left(\frac{1-\sqrt{5}}{2}\right)^0 = b_1 + b_2 = 0$
  - $T(1) = b_1 \left(\frac{1+\sqrt{5}}{2}\right)^1 + b_2 \left(\frac{1-\sqrt{5}}{2}\right)^1 = 1$
- Despejamos con ambas ecuaciones: $b_1 = -b_2 = \frac{1}{5}$.

**Recurrencias útiles**

Master Theorem:

$T(n) = a \cdot T\left(\frac{n}{b}\right) + c \cdot n^k; \; a \geq 1, \; b > 1, \; k \geq 0$

$T(1) = c$

Solución

$a = b^k \Rightarrow T(n) = O\left(n^k \cdot \log n\right)$

$b^k < a \Rightarrow T(n) = O\left(n^{\log_b a}\right)$

$b^k > a \Rightarrow T(n) = O\left(n^k\right)$

### 2.4.3. Fractales

Fractal, división de figuras geométricas.

Entrada: Figura geométrica y como parámetro un número natural.

Se trata de obtener la figura geométrica resultante de dividir la anterior en 4 partes, hasta el número dado por la resolución.

Ecuaciones temporales, expresadas en función de la resolución.

- Ejem. Triángulo $T(0) = c$ y $T(n) = 3 \cdot T(n-1) + d$
- Sol. $T(n) = c3^n + \frac{3^n - 1}{2} d = O(3^n)$

## 2.5. Cotas asintóticas

| Notación | Orden - nombre |
|---|---|
| $O(1)$ | Constante |
| $O(\log n)$ | Logarítmico |
| $O([\log n]^c)$ | Polilogarítmico |
| $O(n)$ | Lineal |
| $O(n \cdot \log n)$ | Lineal logarítmico |
| $O(n^2)$ | Cuadrático |
| $O(n^c)$ | Polinómico |
| $O(c^n)$ | Exponencial |
| $O(n!)$ | Factorial |
| $O(n^n)$ | Potencial-exponencial |

Tabla 2.1: Nombres de los órdenes de complejidad

Peores instancias de tamaño de entrada n: $O(g(n))$ Cota superior.

Mejores instancias de tamaño de entrada n: $\Omega(g(n))$ Cota inferior.

Instancias medias: $\Theta(g(n))$ Cota ajustada.

### 2.5.1. Cota Superior Asintótica

Es una función, g(n), que sirve de cota superior de otra función, f(n), cuando $n \to \infty$. f(n) no puede pasar de g(n).

Se utilizan la notación de Landau Big O $O(g(n))$ para referirse al Conjunto de las funciones $f(n)$ acotadas superiormente por la función $g(n)$.

- $f(n) \in O(g(n))$
- $f(n)$ es $O(g(n))$
- $f(n)$ es $o$ de $g(n)$

$$O(g(n)) = \left\{ \begin{array}{c} f(n) \ / \ \exists c, n_0 \ constantes \ positivas \ tales \ que \\ \forall n \geq n_0, f(n) \leq c \cdot g(n) \end{array} \right\}$$

$$f(n) = O(g(n)) \Leftrightarrow \lim_{n \to \infty} \frac{f(n)}{g(n)} = k \in \mathbb{R}, k \geq 0$$

Se hacen intervalos de signos de la función $h(n) = f(n) - g(n)$ para ver en que momentos nuestra $g(n)$ sobrepasa $f(n)$ y no vuelve a superarla.

Hay que fijarse a partir del último corte que no ocurra que $f(n) > g(n)$. Puede ocurrir a la izquierda, pero nunca más a la derecha.

Algunas propiedades:

- $O(a \cdot f(n)) = O(f(n))$ con $a \in \mathbb{R}$
- $O(\log_a n) = O(\log_b n)$ con $a, b > 1$. $\log_b n = \frac{\log_a n}{\log_a b}$
- Si $f \in O(g)$ y $g \in O(h)$ entonces $f \in O(h)$
- Regla de la suma: $O(f + g) = O(\text{máx}(f, g))$
- Regla del producto: Si $g_1 \in O(f_1)$ y $g_2 \in O(f_2)$ entonces $g_1 \cdot g_2 \in O(f_1 \cdot f_2)$

### 2.5.2. o (o pequeña / ómicron minúscula)

$f(n) = o(g(n))$ $\exists c, n_0 \ / \ f(n) < c \cdot g(n), \ \forall n \geq n_0$ con $n_0$ positivo. $\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$

Nunca la va a poder alcanzar.

### 2.5.3. Cota Inferior Asintótica

Función que sirve de cota inferior de otra función cuando el argumento tiende a infinito.

Se usa para calcular la complejidad del mejor caso para los algoritmos.

$$\Omega(g(n)) = \left\{ \begin{array}{c} f(n) \ / \ \exists c, n_0 \ constantes \ positivas \ tales \ que \\ \forall n \geq n_0, f(n) \geq c \cdot g(n) \end{array} \right\}$$

$$f(n) = \Omega(g(n)) \Leftrightarrow \lim_{n \to \infty} \frac{f(n)}{g(n)} = k > 0 \ (puede \ ser \ \infty)$$

### 2.5.4. Cota Ajustada Asintótica

$$\Theta(g(n)) = \left\{ \begin{array}{c} f(n) \ / \ \exists C_1 \ y \ C_2 > 0, \exists n_0 > 0, \ tal \ que \ \forall n \geq n_0 \\ C_1 \cdot g(n) \leq f(n) \leq C_2 \cdot g(n) \end{array} \right\}$$

$$f(n) = \Theta(g(n)) \Leftrightarrow \lim_{n \to \infty} \frac{f(n)}{g(n)} = k \in \mathbb{R}, k > 0$$

$$f = \Theta(g) \Leftrightarrow f = O(g) \wedge f = \Omega(g)$$

## 2.6. Tratabilidad

**Decidible:** que se puede resolver.

**Tratable:** Son aquellos problemas decidibles para los que se ha encontrado un algoritmo eficiente que lo resuelva.

Si el problema es decidible, entonces nos preocuparemos de su tratabilidad.

Problemas tratables: $T(n) \leq Polinomio \ en \ n^3$

Problemas Intratables:

- $n^4, n^5, ...$
- Exponencial: $2^{n^{O(1)}}$
- Factorial: $n!$

Existen problemas que son intratables en el peor caso, pero que se usan frecuentemente en otros entornos.

# 3. TEORÍA DE LA COMPUTABILIDAD

Clasificar los distintos problemas y formalizar el concepto de computar.

$$\begin{cases} Complejidad & \begin{cases} Algoritmo & \rightarrow & Coste\ temporal\ T(n) \\ Problemas & \rightarrow & Tipo\ de\ problema\ según\ complejidad \begin{cases} P \\ NP \\ NPC \end{cases} \end{cases} \\ Computabilidad & \rightarrow Problema \begin{cases} Computable \\ No\ computable \end{cases} \end{cases}$$

**Problema:** Descripción de una pregunta que requiere una respuesta.

- p: instancia → respuesta
- $x \rightarrow p(x)$

**Instancia:** Especificación exacta de los datos de un problema para un caso particular.

**Algoritmo:** Conjunto de instrucciones que garantiza encontrar una solución correcta para cualquier instancia en un **número finito de pasos**.

**Analogía geométrica:** Hallar el lado de un cuadrado inscrito en un círculo, se puede dibujar fácilmente, pero en cuanto a matemáticas nunca se halla exactamente al tener raíces cuadradas. $r^2 = \frac{l^2}{2} + \frac{l^2}{2};\ l = \sqrt{2}r$. Un caso en el que no es posible de ambas manera es dibujar un cuadrado con la misma área que un círculo, dado que entra en juego $\sqrt{\pi}$. $l^2 = \pi r^2;\ l = \sqrt{\pi}r$

### 3.1. Problema de Satisfabilidad Booleana: SAT

Un problema SAT 2 puede describirse usando una expresión booleana con una forma restringida especial. Es una conjunción de cláusulas, donde cada **cláusula** es una disyunción de **dos variables**. Las variables o las negaciones de variables que aparecen en la fórmula se denominan literales.

- $(x \vee z) \wedge (y \vee w) \wedge (\overline{y} \wedge z)$

if x then y else z

$(x \wedge y) \vee (\overline{x} \wedge z)$

### 3.1.1. Algoritmo sencillo

Generar un conjunto de posibles asignaciones (permutaciones) para todas las entradas.

- Evaluar el resultado de la expresión entera usando esta asignación
- Repetir hasta encontrar una solución

Es eficaz y eficiente para instancias pequeñas.

En cuanto a coste, no hay un algoritmo conocido que garantice la solución de este problema en un tiempo polinómico en el número de variables.

### 3.1.2. Aplicaciones

Muchos Problemas de decisión se pueden reformular en términos del problema de Satisfacibilidad.

- Verificación de HW / SW
- Planning
- Constraint Programming
- Linear Programming

## 3.2. Clique

Un clique es un conjunto $C$ de vértices donde todo par de vértices de $C$ esta conectado con una arista en $G$, es decir, un Clique $C$ es un subgrafo completo.

Subgrafo completo, en el que todos los vértices están conectados con todos.

Tiene mucha potencia social, todos conocen o comparten gustos.

### 3.2.1. Aplicaciones

- Link farms
- Segmentación de imágenes y reconocimiento de patrones, reconocimiento de regiones
- Redes Bayesianas, cálculo de probabilidades condicionales.
- Biología, analizar proteínas similares

### 3.3. Decidibilidad

Un problema de computación puede considerarse como un lenguaje.

Las posibles soluciones del problema pueden considerarse como las palabras que pertenecen al lenguaje.

- Ejem. $ax^2 + bx + c = 0$
- Problema: Tiene solución la ecuación en $\mathbb{R}$
- Respuestas $\in \{Si, No\}$
- Soluciones $\in \mathbb{R}$
- Lenguaje: $L = \{a, b, c\}$

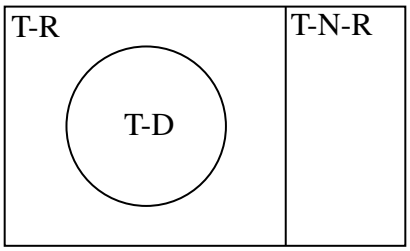



Fig. 3.1: Diagrama Decidibilidad

- **T-D:** Turing-Decidible. Es un problema para el cual una MT acepta todas las soluciones y rechaza las que no son.
  Se para siempre.
- **T-R:** Turing-Reconocible, pero no decidible. Es un problema para el cual la MT acepta todas las que son soluciones.
  Puede entrar en bucle para las no soluciones.
- **T-N-R:** Turing-No-Reconocible. Ni acepta las que son soluciones, ni rechaza las que no son.
  Entra en bucle.

Un T-D es un T-R, siguiendo el diagrama

**Reconocer** un problema (lenguaje) es aceptar todas las soluciones del problema.

**Decidir** un problema es:

- Reconocer todas las soluciones.
- Rechazar todas las no-soluciones.

Solo los T-D se llaman Computables o Decidibles.

- Ejem. El teorema de Fermat ($x^n + y^n = z^n$ con $n > 2/n \in \mathbb{R}$) es indecible porque ni computacional ni matemáticamente se puede demostrar, aunque, se sabe que no hay otro real que cumpla la condición aparte de 2. La máquina estaría infinitamente probando valores.

CUIDADO. Indecidible no especifica si es T-R o T-N-R, a veces al T-R se le llama parcialmente decidible.

**MT Reconocedora:** Que no genera salida, reconocer o no reconoce, pero no lo modifica/manipula.

**MT Transductora:** Que genera salida. Modifica la cinta, además de reconocer o no reconocer la cinta.

## 3.4. Problemas, lenguajes y máquinas de cómputo

### 3.4.1. Notación

$< A, w >$ El automata A con cadena w.

$< M, w >$ La máquina de Turing M con la cadena w.

$L(A)$ El lenguaje que acepta/reconoce el automata A (que puede ser una MT).

$< A, w >$ **acepta** w, si $w \in L(A)$

$< A, w >$ **decide** w, si acepta w si $w \in L(A)$ y rechaza w si $w \notin L(A)$

### 3.4.2. Lenguajes como modelos de problemas

$L(A)$ es el conjunto de soluciones del problema que A se encarga de solucionar.

$A$ es una máquina o algoritmo que resuelve el problema.

- Ejem. Está ordenado un vector de 4 dígitos distintos mayores que 0?
- Instancia $(2, 3, 1, 5)$
- Formulación como lenguaje: $\Sigma = \{1, 2, 3, ..., 8, 9\}$ y $L = \{1234, 1245, ...\}$
- $3145 \in L$? No

### 3.4.3. Función recursiva

No confundir con la recursividad de programación.

f: $X \to \{1, 0\}$ de manera que $f(x) = 1$ si $x \in S$ y $f(x) = 0$ si $x \notin S$.

Es un decisor, de todo el espacio X sabe decir cuáles pertenecen a S.

S es un subconjunto de X, que contiene todas las soluciones.

Se llama **función recursiva** porque es capaz de diferenciar S del conjunto total X.

S es un **conjunto recursivo** si existe una función recursiva aplicable a él. Puede decir siempre si es de S.

S es un **conjunto recursivo-enumerable** si hay una función que puede enumerar sus elementos. Si solo podemos decir si está. f: $\{1, 0\} \to X$

### 3.4.4. Equivalencia de lenguaje y tipos de problemas

| Problema | Lenguaje |
|---|---|
| Decidible | Recursivo |
| T-R | Recursivo-enumerable |
| T-NR | No Recursivo-enumerable |

Tabla 3.1: Equivalencia de lenguajes y problemas

Ejem.

- El problema de la parada es Reconocible y No-Decidible, no es excluyente.
- Problema de determinar el estado de otro problema es Turing Reconocible.
- Matrices mortales es Turing Reconocible, tratará de probar todas la matrices, por lo que las que lo son las reconocerá, pero las que no se pondrá en bucle a probar valores hasta el infinito.
- AKS es Decidible, por mucho que tarde siempre acaba y determina su primalidad.

## 3.5. Máquina de Turing como modelo computacional

Formaliza el concepto de algoritmo.

Es un modelo formal de computador. O lo que es lo mismo: «El reconocimiento de cadenas que forman parte de un lenguaje es un modo formal de expresar cualquier problema».

Se puede probar matemáticamente que para cualquier programa de computadora es posible crear una máquina de Turing equivalente.

### 3.5.1. Máquina de Turing binaria

Máquina de Turing, $M_{2)}$, en la que el alfabeto de cinta es $\{0, 1\} = \Gamma$.

Para toda MT existe una equivalente binaria.

n. símbolos $\leq 2^k \rightarrow$ k número de dígitos binarios para codificar $\Gamma$.

- Ejem. $\Gamma = \{a, b, c, blank\} \quad a = 00, b = 01, c = 10, blank = 11$

Para simular cada paso de MT, mediante $M_{2)}$

- Leer de la cinta el símbolo de M, i.e. varias (k) casillas de $M_{2)}$
- Escribir el símbolo de M que lo reemplaza, varias (k) casillas de $M_{2)}$
- Moverse (k casillas) y cambiar de estado

### 3.5.2. Generalizaciones de las Máquinas de Turing

Incrementar el alfabeto de la cinta $\Gamma$

Añadir más cintas (MT con 2, 3, etc. cintas)

Añadir más cabeceras lectoras y escritoras (o puntos de acceso) en la cinta

- Entramados de dos o más dimensiones en sustitución de la cinta. La cabeza lectora se puede mover a cualquier celda contigua (en cualquier dimensión)
- NO DETERMINISMO

Toda posible generalización de la Máquina de Turing se puede simular con una máquina de Turing: Determinista, con cinta infinita en ambos sentidos, y donde el alfabeto de la cinta es 1,0, b (b es celda vacía, en blanco)

### 3.5.3. Máquina de Turing No Determinista (MTND)

$\delta : Q \times \Gamma \rightarrow P(Q \times \Gamma \times \{L, R, H\})$

$P(A)$ es el conjunto definido como partes de A.

La computación en una MTND:

- Es un árbol donde sus ramas se corresponden con las diferentes posibilidades de la máquina.
- Si alguna de las ramas del árbol alcanza un estado de aceptación, la MTND acepta la entrada

Una MT M es No Determinista si existen al menos dos transiciones que tienen la misma parte izquierda (estado y símbolo leído), pero distinta parte derecha.

- $\delta(estado, simbolo\ leido) \rightarrow (nuevo\ estado, simbolo\ escrito, accion)$

Una MTD es por definición No Determinista.

Toda MTND tiene un MTD equivalente

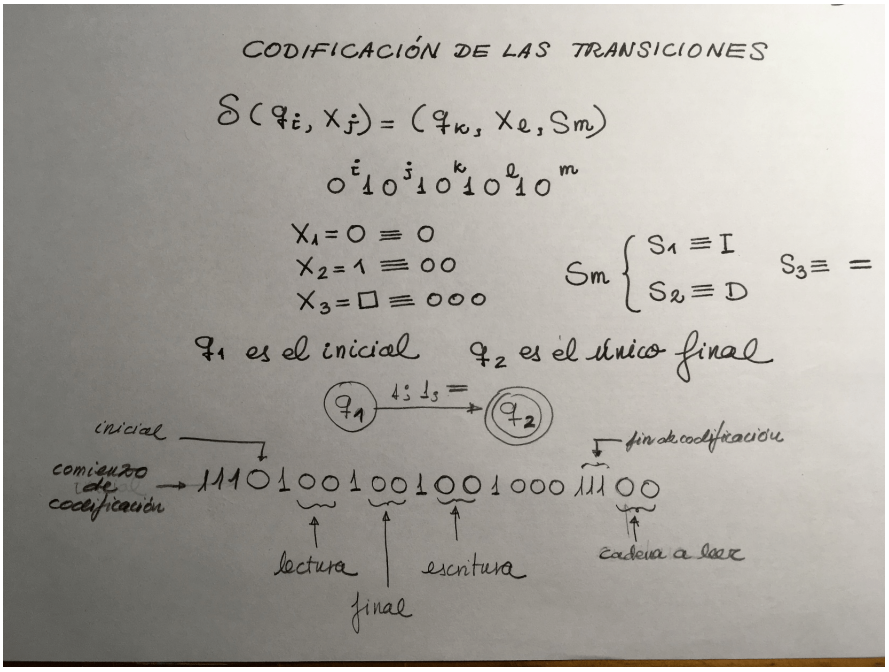



Fig. 3.2: Codificación de las transiciones

### 3.5.4. Salidas de la Máquina de Turing

Dada una MT y una entrada, se pueden dar tres tipos de salidas:

- La MT se detiene, y acepta la entrada (la palabra)
- La MT se detiene, y rechaza la entrada (la palabra)
- No se detiene

Dependiendo cuáles se den:

- T-D: Se detiene con $a \in L$ en un estado final (acepta) y con $a \notin L$ en un estado no final (rechaza).
- T-R: Se detiene para $a \in L$ en un estado final, acepta todas las que pertenecen. Es menos restrictivo.

## 3.6. Decidibilidad

Un conjunto S se dice que es infinito contable si tiene la misma cardinalidad que $\mathbb{N}$ ($|S| = |\mathbb{N}|$) Si puede establecerse una correspondencia biyectiva entre los elementos de $\mathbb{N}$ y los elementos de S. Se puede decir este elemento es el primero, este el segundo, etc.

Un conjunto se dice que es contable si es finito o infinito contable.

Un conjunto no contable se dice que es incontable.

### 3.6.1. Nociones formales

$\aleph_0 < \aleph_1 \leq 2^{\aleph_0}$

- Cardinalidad de los naturales: $\aleph_0$
- Cardinalidad de los enteros: $\aleph_0$
- Cardinalidad de los racionales: $\aleph_0$
- Cardinalidad de los reales: $2^{\aleph_0}$

Teorema: El conjunto de cadenas binarias infinitas es incontable.

#### Biyección entre $\mathbb{N}$ *y* $\mathbb{Q}^+$

Poniendo todas las fracciones fijando primero denominador 1, 2, 3, ... para cada una de estas se ponen los numeradores 1, 2, 3, ...

Se va recorriendo y contando siguiendo las diagonales, saltándose los que tengan un valor equivalente a uno ya contado. En el borde izquierdo se baja y se va subiendo en diagonal, una vez arriba se pasa a la derecha y se baja en diagonal.

Por lo que tienen la misma cardinalidad, se pueden numerar aunque sea paradójica.

#### Biyección entre $\mathbb{N}$ *y* $\mathbb{Z}$

George Cantor lo demuestra. Se cuentan haciendo pasos de valor 1, el positivo y después el negativo.

1 -1 2 -2 3 -3 4 -4 5 -5 ...

#### Biyección entre $\mathbb{N}$ *y* $\mathbb{R}$

Teorema: No se puede. El conjunto de los números reales en [0,1) es incontable, y por ende, el de los números reales.

Se pueden listar los valores 0,... y cogiendo la diagonal de los decimales, si se le suma 1 con módulo 10 no se encuentra ese valor en los previos, por lo que habrá más.

### 3.6.2. Máquina de Turing Universal

Es aquella Máquina de Turing que recibe como cadena una descripción de lenguaje (M) y la cadena de entrada (w).

Es capaz de simular cualquier otra máquina de Turing.

- Si M acepta w, U se parará en un estado final.
- Si M NO acepta w, U no se parará o bien se parará en un estado no final.
- Si M no se para, U tampoco.

### 3.6.3. Algunos teoremas importantes

Lenguaje Universal: $A_{MT}$ o $L_U$.

El complementario de un lenguaje está formado por aquellos elementos de $\Sigma^*$ que no están en L.

- $L = \{x/|x| = 2$ *y x comienza con 1*$\}$. $L = \{10, 11\}$ y $\overline{L} = \{\lambda, 0, 1, 01, 000, ...\}$

#### Teorema Hay lenguajes que no son reconocibles (NTR) por MTs

$\exists L/L$ no es reconocible por una MT.

Si M no sabe decir (entra en bucle) si $w \in L$ entonces NTR.

El conjunto de todas las MT es contable, dado que $\Sigma^*$ es contable y se puede codificar una MT.

Sea L el conjunto de todos los lenguajes del alfabeto de cadenas infinitas, se ha visto que es incontable.

Por lo tanto, algunos lenguajes no pueden ser reconocidos por MTs. Son NTR.

#### Teorema $A_{MT}$ es Turing-reconocible

Sea $A_{MT} = \{< M, w > |M$ *acepta* $w\}$

La MT U simula M con entrada w. Es decir: U acepta o rechaza si M lo hace, y entra en bucle infinito si este lo hace.

**Teorema $A_{MT}$ es indecidible**

Se puede demostrar por reducción al absurdo.

Se construye H que decide $A_{MT}$, entonces construimos D, como H negada. Por último, pasamos como entrada la propia máquina D ($< D, < D >>$). Entonces llegamos al absurdo.

**Teorema $\overline{A_{MT}}$ es no-Turing-reconocible**

Teorema: Un lenguaje es decidible si y solo si es Turing-Reconocible y su complementario es también Turing-Reconocible.

L es TD $\Rightarrow$ L es TR y $\overline{L}$ es TR.

$\overline{A_{MT}}$ es el conjunto de $< M, w >$ que no reconocen w.

Sea $\overline{A_{MT}}$ Turing-reconocible. Entonces $A_{MT}$ seria Turing-decidible ($A_{MT}$ es TR), pero sabemos que es indecidible y, por tanto, $\overline{A_{MT}}$ es no-Turing-reconocible.

Otro lenguaje no-Turing-Reconocible es $EQ_{MT} = \{< M_1, M_2 > | M_1, M_2 \textit{ son MTs y } L(M_1) = L(M_2)\}$, dadas dos MTs decide si son equivalentes (mismo lenguaje).

**Teorema $HALT_{MT}$ es indecidible**

Dada una MT universal, U, y su entrada $< M, w >$. Decide si M se para o no.

$HALT_{MT} = \{< M, w > / M \textit{ se para con } w\}$. Por absurdo se puede demostrar que no existe.

- Construimos $H_1$, que niega la salida de H.
- Hacemos que $H_1$ sea la entrada de $H_1$.
- Si H decide que $H_1$ se para, $H_1$ dice que no se para.
- Si H decide que $H_1$ no se para, $H_1$ dice que se para.

Alan Turing en 1936 demuestra que NO existe un algoritmo que resuelva este problema.

### 3.6.4. Lenguaje Diagonal

Se hace una matriz de M x w, Máquina de Turing y entrada.

Esta se completa de manera que:

- $(i, j) = 1 \Rightarrow w_i \in L(M_j)$
- $(i, j) = 0 \Rightarrow w_i \notin L(M_j)$

$L_d = \{w_i / M_i \text{ no acepta } w_i\} = \{w_i / w_i \notin L(M_i)\}$

Sea $M_j$ la MT que acepta $L_d$ ($L_d = L(M_j)$):

- $w_j \in L_d \Leftrightarrow (j, j) = 0 \Rightarrow w_j \notin L(M_j)$
- $w_j \notin L_d \Leftrightarrow (j, j) = 1 \Rightarrow w_j \in L(M_j)$

Es contradictorio.

### 3.6.5. Resumen

- $A_{MT}$ o $L_U$ es Turing-Reconocible, pero no Turing-Decidible.
- $\overline{A_{MT}}$ o $\overline{L_U}$ es No-Turing-Reconocible.
- $HALT_{MT}$ es Turing-Reconocible.
- $L_d$ es No-Turing-Reconocible.

### 3.6.6. Reducción de un problema A a otro problema B

Reducir A a B es A en B, de modo que una solución a B pueda usarse para resolver A.

Si A se reduce a B (A es reducible a B), resolver A no será más difícil que resolver B.

$A \leq_m B$ mapping; $A \leq_p B$ polinomial time.

- Que sea polinomial, implica que la reducción se lleva a cabo una función cuyo $T(n) = O(n^k)$

Dados dos lenguajes A y B, y una función $f : \Sigma^* \rightarrow \Sigma^*$, f es una transformación del lenguaje universal del problema A al lenguaje universal del problema B.

$\forall w \in A \Leftrightarrow f(w) \in B; A \leq_f B$

Teoremas:

1. B decidible ⇒ A decidible
2. B Turing-reconocible ⇒ A Turing-reconocible
3. A no-Turing-reconocible ⇒ B no-Turing-reconocible
4. A indecidible ⇒ B indecidible
5. B decidible ⇒ es fácil de resolver A es fácil

Para probar que un problema B es indecidible bastará mostrar que un problema indecidible A se reduce a B.

- Si A es difícil $\Rightarrow$ B es difícil
- Si B es difícil no podemos extraer conclusiones de A
- Si $A \leq_p B$ y $B \leq_p A$ entonces son igual de difíciles.

Ejem.

- El problema M de multiplicar dos números enteros y el problema C de elevar al cuadrado un número entero. El problema C es más fácil que el problema M. De hecho, si podemos hacer la multiplicación, se puede elevar al cuadrado un número multiplicándolo por sí mismo, por lo que $C \leq M$. $x \rightarrow (x, x)$
- *camino Hamiltoniano* $\leq_p$ *ciclo Hamiltoniano*. Dado que para que sea ciclo tiene que ser primero camino. Por lo tanto, las soluciones a ciclo sirven a camino.

### $HALT_{MT}$ es indecidible

$A_{MT} \leq_m HALT_{MT}$

- Construimos S para decidir $A_{MT}$
- Dentro R decide $HALT_{MT}$ y tenemos también M que es la MT de entrada.
  - Si M se para, S simula M con w hasta que pare.
    - Si M acepta, S acepta.
    - Si M rechaza, S rechaza.
  - Si M no se para, R rechaza, por lo que S rechaza.

Si R decide $HALT_{MT}$, S decide $A_{MT}$. Es decir, $A_{MT}$ se reduce a $HALT_{MT}$, pero $A_{MT}$ es indecidible (TR), luego $HALT_{MT}$ es indecidible (es TR, pero no TD).

# 4. TEORÍA DE LA COMPLEJIDAD COMPUTACIONAL

Consideremos 2 tipos de recursos:

- **Tiempo:** tiempo de ejecución o número de pasos base.
- **Espacio:** recursos de memoria utilizados.

## 4.1. La Máquina de Turing y medida de la Complejidad

El tiempo de ejecución de Máquina de Turing es la función $T(n) : N \to N$, donde T(n) es el número máximo de pasos que una MT utiliza para realizar el cómputo para entrada de tamaño n.

Complejidad Computacional: se refiere a la complejidad de los problemas.

- Se toma la peor instancia
- Se utiliza la Cota Superior Asintótica: O(g)

La principal herramienta para comparar problemas y clasificarlos:

- Desde el punto de vista de la computabilidad
- Desde el punto de vista de la complejidad computacional

## 4.2. Reducción entre problemas

Dado dos Lenguajes A y B, y una función $f : \Sigma^* \to \Sigma^*$.

f es una transformación del lenguaje universal del problema A al lenguaje universal del problema B. Se dice que **A es reducible a B**, bajo f si:

$$\forall w \in A \Leftrightarrow f(w) \in B$$
$$A \leq_f B$$

**Reducción:** transformar un problema en otro. Transformar toda instancia de un problema en instancias de otro problema.

Intuitivamente:

- es una manera de convertir un problema a otro problema, de tal manera que la solución al segundo problema se puede utilizar para resolver el primer problema.

- «Si el problema A es reducible al problema B, entonces la solución al problema B se puede utilizar para resolver el problema A»

Formalmente:

- Si el coste de Reducir A en B, es decir, el proceso de transformar una instancia de A en una instancia de B, es de orden polinómico (o menor) se puede afirmar que:
  - Resolver A no puede ser más complejo («harder») que resolver B. Es decir, resolver A a lo sumo es tan complejo de resolver como B.
  - Resolver B al menos es tan complejo como resolver A
  - Estas dos afirmaciones se expresan así: $A \leq B$
- Reducir el problema A al problema B en tiempo polinómico implica que la reducción se lleva a cabo con una función cuyo $T(n) = O(n^k)$, $A \leq_p B$
- Si $A \leq_p B$ y $B \leq_p A$ son igual de difíciles.

### 4.3. Clases de Problemas

Tiempo

- Clase P
- Clase NP
- Clase NP-Completo
- Clase NP-Hard

Espacio

- Clase P-Space
- Clase NP-Space

**TIME(f(n)):** Conjunto de problemas de decisión que se pueden resolver mediante una Máquina de Turing Determinista con coste $O(f(n))$.

**NTIME(f(n)):** Problemas de decisión que se pueden resolver mediante una Máquina de Turing NO Determinista con coste $O(f(n))$.

Lo que una MTD puede resolver en Time($2^n$), una MTND lo puede resolver en menor tiempo NTIME(n). Pero no al revés, porque no sabemos cuánto más tardara.

### 4.3.1. Complejidad Temporal

#### Clase P

Problemas (Lenguajes) decidibles por una MT Determinista (de una sola cinta) en TIEMPO POLINÓMICO. $P = \bigcup_k TIME(n^k)$

- Time($n$), Time($n^2$), ..., Time($n^k$)

La clase P contiene todos los problemas decibles por un algoritmo que cuyo coste computacional (tiempo) es de orden polinómico con respecto al tamaño de la entrada (n).

«P se corresponde con la clase de problemas que se pueden resolver desde un punto de vista realista/práctico en un computador».

La mayoría de los problemas corrientes pertenecen a la clase P: Ordenación, Búsqueda, Operaciones con vectores.

"La clase P es robusta"

- La clase P es invariante (i.e. robusta) para todos los modelos de computación que son polinómicamente equivalentes a la MT Determinista de una sola cinta.
- Los problemas en P permanecen en P aunque se cambie el modelo de computación.

Ejemplos de problemas en P:

- Si existe un camino entre dos puntos en un grafo
- Si x e y son coprimos
- Si x es primo (AKS lo demuestra)

#### Clase NP

Problemas (Lenguajes) decidibles por una MT NO Determinista (de una sola cinta) en TIEMPO POLINÓMICO. $NP = \bigcup_k NTIME(n^k)$

- Time($n$), Time($n^2$), ..., Time($n^k$)
- NTime($n$), NTime($n^2$), ..., NTime($n^k$)

Es decir, problemas para los que existe una MT NO DETERMINISTA que da solución parándose (da una respuesta sea cual sea la entrada) en TIEMPO POLINÓMICO.

NP es la clase de **lenguajes que tienen un verificador en tiempo polinómico**. Que no se conozca, no quiere decir, no exista uno en tiempo polinómico.

Todo P es NP, $P \leq NP$, pero no se ha podido demostrar que sean el mismo conjunto, por ahora $P \subset NP$.

Podemos **evitar la fuerza bruta para conseguir soluciones de muchos problemas** en tiempo polinómico. Pero para **otros problemas no se conocen soluciones en tiempo polinómico**. No se sabe si se podrán conseguir, podría suceder que estos **problemas fueran intrínsecamente difíciles**.

Si un problema **no es NP, no es TD**.

Lo primero y más fácil que podemos utilizar para saber si pertenece a NP es hace un verificador, y si este lo hace en tiempo polinómico.

El segundo paso es ver si este se puede resolver con una MT ND.

### Verificabilidad

Verificador V para un lenguaje L: $L = \{w / V \ acepta \ < w, c > \ para \ alguna \ palabra \ c\}$

$$w \in L \Rightarrow \exists c \in \Sigma^*. V \ acepta \ < w, c >$$

- **Verificador:** Es un algo que permite ver si una instancia/cadena cumple el problema o no, pero no lo resuelve. Dice si la solución es buena o no.

  El verificador asociado a un problema debe tener para cada una de las palabras de dicho problema (w) una cadena c llamada certificado que hace que termine el verificador en un estado de aceptación. Para aquella que no sean del lenguaje las rechazara, no habrá un c.

- **Certificador:** Cadena que se asocia a una salida w que recibe el verificador que ayuda a saber si la solución pertenece al lenguaje.

Si V se ejecuta en tiempo polinómico, V es un verificador en tiempo polinómico, y L es polinómicamente verificable

NP es la clase de lenguajes que tienen verificadores en tiempo polinómico.

Encontrar un verificador es más sencillo que resolver el problema, este solo dice si es una buena solución o no resuelve el problema.

### P vs. NP

**P** es la clase de lenguajes en los que la pertenencia de una palabra puede ser **decidida rápidamente**. (reconocer/aceptar)

**NP** es la clase de lenguajes en los que la pertenencia puede ser **verificada rápidamente**. (verificar)

### Clase EXPTIME

$EXPTIME = \bigcup_k TIME(2^{n^k})$

Todos los conjuntos exponenciales. $NP \leq EXPTIME$

### Clase NP-Completo

Un problema de decisión $p \in NP - Completo$ si y solo si:

- $p \in NP$
- $\forall q \in NP$ existe una reducción de q en p. Es decir, existe una transformación f de coste polinómico de q en p, tal que $q \leq_p p$

Los problemas NPC se pueden reducir entre ellos.

Todos los problemas de NP se pueden reducir en tiempo polinómico en NP-Completo. De esta manera se puede demostrar que un problema pertenece a NP-Completo, si se puede reducir en tiempo polinómico un problema NP-Completo a este.

- Tenemos los problemas $p$ y $q \in NP - Completo$, por lo que si $q \leq_p p$ entonces $p \in NP - Completo$.
- Ejem. Como el Buscaminas (Minesweeper) es más difícil que SAT y sabemos que SAT es NP-Completo, entonces Buscaminas es NP-Completo.

Formas de resolver problemas en tiempo razonable:

- **Aproximaciones:** Un algoritmo que rápidamente encuentra una solución no necesariamente óptima, pero dentro de un cierto rango de error.

  En algunos casos, encontrar una buena aproximación es suficiente para resolver el problema, pero no todos los problemas NP-completos tienen buenos algoritmos de aproximación.

- **Probabilístico:** Un algoritmo probabilístico obtiene en promedio una buena solución al problema planteado, para una distribución de los datos de entrada dada.
- **Casos particulares:** Cuando se reconocen casos particulares del problema para los cuales existen soluciones rápidas.
- **Heurísticas:** Un algoritmo que trabaja razonablemente bien en muchos casos. En general son rápidos, pero no existe medida de la calidad de la respuesta.
- **Meta heurística (Computación evolutiva, Algoritmo Genético):** Algoritmos que mejoran las posibles soluciones hasta encontrar una que posiblemente esté cerca del óptimo. Tampoco existe forma de garantizar la calidad de la respuesta.

3-SAT es NP-Completo, pero 2-SAT no.

### 3SAT ≤ K-Clique

K-Clique, sabemos que es NP, vamos a demostrar si es NP-Completo.

$$3SAT \leq K\text{-}Clique$$

Cada literal es un vértice del clique. Con 3 cláusulas tenemos 9 literales, por tanto, 9 vértices. $\phi = (x \vee y \vee z) \wedge (x \vee x \vee \overline{y}) \wedge \ldots$

Entonces se hacen todas las conexiones posibles siguiendo:

- No se conectan literales en la misma cláusula.
- Un literal no se conecta con su opuesto. $y$ *e* $\overline{y}$

La solución será aquella que haga True todas las cláusulas, que coincide con un clique en el grafo que hemos generado.

1. $w \in 3SAT \rightarrow f(w) \in K\text{-}Clique$ y $f(w)$ se puede hacer en tiempo polinómico.

   Supongamos que $\phi$ tiene una asignación satisfacible de literales.

   - Entonces un literal por lo menos en cada cláusula debe tener valor de verdadero (ya que en cada cláusula hay solo disyunciones)
   - En cada cláusula representada en el grafo seleccionamos un solo vértice con valor de verdadero. Habremos seleccionado k vértices que estarán interconectados en CLIQUE.

2. $f(w) \in K\text{-}Clique \rightarrow w \in 3SAT$

   Supongamos que el grafo tiene un k-clique

   - No habrá vértices en el grafo que correspondan a la misma cláusula de una forma normal conjuntiva de k cláusulas, ya que hemos seleccionado solo un vértice por cláusula: cada cláusula correspondiente contiene un vértice en el grafo.
   - Asignemos valores a los literales para que cada uno de los que etiquetan un nodo en el CLIQUE sea verdadero.

**Clase NP-Hard**

Un problema de decisión $h \in NP\text{-}Hard$ si y solo si:

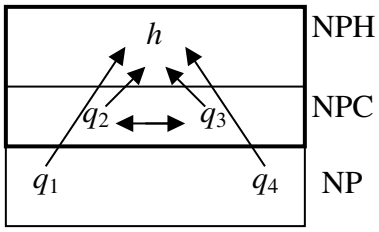
- $\forall q \in NP$, existe una reducción de q en h. Es decir, existe una transformación f de coste polinómico de q en h tal que $q \leq_p h$

Los NP-Hard pueden reducir los NP sin ser NP, siendo NTD.

Todos los NPC son NPH, $NPC = NP \cap NP\text{-}Hard$, pero no todos los NPH son NPC porque no están en NP.

Es el conjunto de los problemas de decisión que contiene los problemas h tales que todo problema q en NP puede ser transformado polinómicamente en h ($q \leq_p h$).

- Se puede saber que un NPH está en NP, si se puede reducir este a un NPC.




### 4.3.2. Complejidad Espacial

Se utiliza como otra forma de clasificar problemas, según su dificultad computacional.

El modelo matemático para esta clasificación es la MT.

La complejidad espacial de una MTD M, que para siempre, es una función $f : N \to N$, tal que $f(n)$ es el máximo número de celdas que M visita para una entrada de longitud n.

Cantidad de espacio (memoria) que se requiere. El espacio a diferencia del tiempo se puede reutilizar.

Se dice que M se ejecutara en un espacio $f(n)$:

- Ejem: $< MTD, w >$ MTD se ejecuta en $f(n) = 4n$ para $n = 2$ $f(2) = 8$. Con entrada 2 usará un total de 8 celdas.

**SPACE(f(n)):** Problemas (Lenguajes) decidibles por una MT Determinista (de una sola cinta) en ESPACIO POLINÓMICO. $PSPACE = \bigcup_k SPACE(n^k)$

**NSPACE(f(n)):** Problemas (Lenguajes) decidibles por una MT NO Determinista (de una sola cinta) en ESPACIO POLINÓMICO. $NPSPACE = \bigcup_k NSPACE(n^k)$

SAT por ejemplo es $O(n)$ en cuanto a espacio y en tiempo es exponencial.

**Teorema de Savitch (1970)**

$\forall f : N \to R^+$, donde $f(n) \geq n$. $NSPACE(f(n)) \subseteq SPACE(f^2(n))$

En otras palabras, cualquier problema decidible en una máquina de Turing no determinista en el espacio $f(n)$ también se puede decidir en una máquina de Turing determinista en el espacio $f^2(n)$.

Corolario: $PSPACE(n^k) = NPSPACE(n^k)$. Como los PSPACE absorben los NPSPACE resultan ser lo mismo.

- $P \subseteq NP$. P incluido en NPC
- $P \subseteq PSPACE$
- $NP \subseteq NPSPACE$
- $PSPACE = NPSPACE$
- $P \subseteq NP \subseteq PSPACE = NPSPACE \subseteq EXPTIME$

# 5. MODELOS DE COMPUTACIÓN

Es un modelo que describe la estructura que hace posible computar una salida a partir de una entrada, conocido como modelo, se puede medir la complejidad de un algoritmo que resuelve un problema.

Un modelo recibe como entrada una instancia, aplica un algoritmo y se obtiene una salida, analizando este proceso se determina la complejidad.

Algunos modelos de computación:

- Automatas Finitos (AFD)
- Automatas de Pila (AP)
- Máquinas de Turing (MT)
- Random Access Machine (RAM)
- Automatas Celulares (AC)

**Turing Completo:** Un modelo es Turing-completo si puede computar (resolver) las mismas funciones (problemas) que una Máquina de Turing.

- AF y AP NO son Turing Completos.
- RAM, $\lambda$-calculus y Automatas Celulares son Turing Completos.

Un problema No-Turing-Decidible aunque se cambie el modelo de computación no lo va a dejar de ser.

La MT es el modelo de modelos. La MT es muy potente en capacidad de resolver problemas, pero muy ineficiente en cómputo, por lo que utilizar otras puede resultar más eficiente, pero no son más capaces.

## 5.1. Automatas Finitos (AF)

Se definen mediante: ($\Sigma$, Q, f, $q_0$, F)

- $\Sigma$: alfabeto de entrada.
- Q: conjunto de estados.
- f: función de transición.

- $q_0$: estado inicial, $q_0 \in Q$.
- F: conjunto de estados finales, F $\subset$ Q.

Pueden ser Deterministas (AFD) y No Deterministas (AFN). Se puede pasar un AFN a un AFD.

Es No Determinista si:

- Para un mismo signo en un estado hay más de una transición.
- Transitar con $\lambda$.
- No hay estado final?

El problema de estas máquinas es que no tiene memoria, por lo que no pueden recordar.

### 5.2. Automata de Pila (AP)

Se definen mediante: ($\Sigma$, $\Gamma$, Q, $A_0$, $q_0$, f, F)

- $\Sigma$: alfabeto de entrada.
- $\Gamma$: alfabeto de pila (normalmente son mayúsculas).
- Q: conjunto de estados.
- $A_0$: conjunto de estados de la pila inicial, $A_0 \subset Q$.
- $q_0$: estado inicial, $q_0 \in Q$.
- f: función de transición.
- F: conjunto de estados finales, F $\subset$ Q.

Un APN NO se puede pasar a APD.

### 5.3. Generalizaciones de las Máquinas de Turing

Toda posible generalización de la Máquina de Turing se puede simular con una máquina de Turing:

- Determinista,
- con cinta infinita en ambos sentidos, y
- donde el alfabeto de la cinta es 1,0, b (b es celda vacía, en blanco)

## 5.4. Random Access Machine Model (RAM)

También conocido como Máquina de Registros.

Modelo abstracto de los computadores digitales. Reflejan más los aspectos de la computación real que las MT.

MT debe recorrer las celdas anteriores a una concreta para leerla, esta, sin embargo, puede acceder directamente a los registros al estar numerados.

Estructura:

- $R_0$, $R_1$, ... Un número ilimitado de registros (celdas). Capaz de almacenar un entero cualquiera.
- $I_0$, $I_1$, ... La entrada es un conjunto finito de datos, no almacenados en los registros.
- **Programa:** Es un conjunto finito de instrucciones, que no están almacenados en los registros. Almacenado en un número finito de registros, y uno adicional, denominado Program Counter (PC).

Conjunto de posibles instrucciones (cada operación aumenta el PC, excepto el salto condicional que llega a un determinado valor):

- $R_i \leftarrow R_j$ Asignación
- $< R_i > \leftarrow R_j$ y $R_i \leftarrow < R_j >$ Direccionamiento indirecto ($< R_i >$ valor almacenado en la celda con dirección almacenada en $R_i$)
- $R_i \leftarrow R_j + R_k$ Suma
- $R_i \leftarrow R_j - R_k$ Resta
- $R_i \leftarrow R_j \; bool \; R_k$ Operación booleana
- If $R_i$ comp $R_j$ label 1 else label 2 Salto condicional, para comp=comparación

Hay diferentes tipos de arquitecturas (como pasa con los tipos de MT) con las que se puede implementar: Counter Machine, Pointer Machine, Random Access Machine, Random Access Stored Program Machine.

### 5.4.1. Random Access Stored Program (RASP)

Es una máquina abstracta empleada para implementar algoritmos.

Se diferencia de la RAM, en que el programa está almacenado en los propios registros de la máquina. Los datos y las instrucciones están en la misma memoria (memoria de acceso aleatorio)

- RAM → Arquitectura Von Harvard → Máquina de Turing.
- RASP → Arquitectura Von Neumann → Máquina de Turing Universal.

## 5.5. Automatas Celulares

Sucesión de sitios que contienen un valor numérico, que está determinado a seguir unas reglas que hacen evolucionar estos valores a lo largo del tiempo.

Es un modelo de un «mundo» con una física muy sencilla. El mundo está dividido en celdas (entramado de celdas).

El Automata Celular está gobernado por reglas que rigen la evolución del sistema en el tiempo.

- El tiempo se considera Discreto (steps).
- Las reglas para el siguiente paso en el tiempo (t+1) se basan en el estado del autómata en el instante actual (t).
- El comportamiento de una celda se puede representar (computar) mediante un Automata Finito.

**AC Deterministas:** Las reglas no tienen componentes de aleatoriedad, dado el mismo estado inicial, siempre se obtendrán el mismo estado final.

**AC No Deterministas:** Sigue una aleatoriedad controlada, pero aleatoria.

Aplicaciones: Generadores de contenido multimedia, Generadores de números aleatorios, Desarrollo de Computadores paralelos y Supercomputación.

### 5.5.1. Variedades

Pueden diferir en:

- Dimensión del entorno d=1,2,3...
- Formación de celdas: triangular, cuadrado, rectangular, pentagonal, hexagonal, etc.
- Forma y tamaño del entorno: triangular, cuadrado, rectangular; finita, infinita.
- Tipo se automata finito asociado a cada celda: determinista, probabilístico.
- Definición de los vecinos de cada automata
    - Puede indicarse con un número r (distancia), en función del cual obtenemos el número de vecinos que rodean a una celda concreta.

- En AC bidimensionales podemos considerar:
  - Vecindad de Von Neumann (arriba, abajo, izquierda y derecha)
  - Vecindad de Moore (todas las direcciones, incluyendo las diagonales).
- Número de celdas que rodean a una celda concreta central: $(2r + 1)^2 - 1$

### 5.5.2. Clasificación

- Clase I: Desde casi cualquier estado inicial convergen siempre a una misma configuración final.
- Clase II: Genera un patrón simple, donde hay una estructura que se repite.
- Clase III: Generan patrones aleatorios.
- Clase IV: Generan estructuras locales que son capaces de sobrevivir durante mucho tiempo (mantenerse iguales).

Un ejemplo de este tipo de autómata, es el Juego de la Vida de John H. Conway, en el que la configuración va evolucionando de manera que parece una población.

- Cada célula está vida o está muerta (estados binarios).
- Tiene una vecindad de r=1, de tipo Moore (todas las adyacentes).
- Se van formando diferentes patrones.
- Regla:
  - Muere si hay menos de 2 o más de 3 células vivas alrededor.
  - Vivos si hay exactamente 2 o 3 células vivas alrededor.
  - Nace si es un 0 y hay exactamente 3 células vivas alrededor.

### 5.5.3. AC Unidimensional

Dimensión del AC: Dimensiones del entramado que forman el conjunto de celdas.

En una dimensión, hay 3 configuraciones:

- **Secuencia finita:** Hay posibles estados finitos.
- **Anillo:** Siempre vuelve al mismo estado.
- **Secuencia infinita:** El cambio siempre es diferente.

Las Reglas dependen del estado que tenga la celda $x_i$ en el instante anterior y del estado del vecindario (izq. y dch.) en el instante anterior.

Rule 50 y 30, ejemplo de reglas que con vecindario de las celdas adyacentes (3 valores) define que valor tomara la siguiente, la distribución será el valor en binario 50 o 30 escritos de menos a más significativo de arriba a abajo.

### 5.6. L-Systems

Propuesto por Aristid Lindemayer en 1968.

Es un tipo de lenguaje formal generado por una gramática. Es Turing-Completo.

Aplicación: Simulación de procesos biológicos (plantas, hojas, etc.), Turtle interpretation, Patrones bidimensionales, Formalización de curvas fractales.

**Definición formal:** (Σ, w, P)

- Σ: Alfabeto de símbolos.
- w: Axioma, $w \in \Sigma^*$.
- P: Conjunto finito de reglas del tipo u::=v donde $u \in \Sigma^*$ y $v \in \Sigma^*$.

**Lenguaje generado por un L-system:** $\{x | u \in \Sigma^+, w \rightarrow^* x\}$. Las reglas se aplican en Paralelo.

**Tipos de Gramáticas:**

- **Sensibles al contexto (1L):** La parte derecha de la producción sea igual o más larga que la izquierda, y esta puede tener más de un símbolo. $xay ::= xvy$
- **Independientes del contexto (0L):** $V \rightarrow w$. Donde V es un símbolo no terminal y w es una cadena de terminales y/o no terminales
- **Deterministas (DL):** solo existe una regla posible que ser aplicada.
- **Propagativos (PL):** la parte derecha no puede ser más corta que la izquierda.
- **Con Tablas (TL):** en cada paso de la derivación se aplican distintos conjuntos de reglas.
- **Con Extensiones (EL):** solo pertenecen al lenguaje las palabras formadas por un subconjunto del alfabeto.

### 5.6.1. L-Systems vs. Gramáticas de Chomsky

**Gramáticas de Chomsky:** Las reglas se aplican de una en una, Derivación Secuencial.

**L-Systems:** Todas las derivaciones se realizan Simultáneamente, Derivación en paralelo. Se aplican las reglas para todos los símbolos (distintos o no) de la palabra.

## 5.7. Máquinas de Turing

Se definen mediante: (Q, Γ, b, Σ, $\delta$, $q_0$, F)

- Q: conjunto de estados.
- Γ: alfabeto de cinta.
- b: símbolo en blanco, b ∈ Γ.
- Σ: alfabeto de entrada.
- $\delta$: función de transición.
- $q_0$: estado inicial, $q_0 \in Q$.
- F: conjunto de estados finales, F ⊂ Q.

### 5.7.1. Origen

Formulada por Turing como un medio para formalizar el proceso sistemático de resolver problemas de un matemático.

En sentido más amplio: Para representar cualquier Algoritmo, por precaución hablamos de Procedimientos Efectivos para resolver un problema.

Actualmente, es una máquina evolucionada con base en contribuciones de Post, Turing y otros.

### 5.7.2. Utilidad

Turing se sirvió de ella para desarrollar diversas cuestiones:

- Máquina de Turing Universal.
- Problema de la parada (demostración computacional) en equivalencia con el Entscheidungsproblem de Gödel.
- Modelo Universal de Computación.

### 5.7.3. Modelo de Computación Universal

Las Máquinas de Turing definen un Modelo de Computación que es la base de los Lenguajes Imperativos y Procedimentales.

Suposiciones previas:

- MT de cinta infinita
- Alfabeto= $\{b, 1\}$ siendo b o # representan celda en blanco
- Alfabeto auxiliar con símbolos de marcado $\{\$, X, etc\}$
- Se representan Números Naturales en base 1

#### Aritmética de Peano

Conjunto axiomático que sirve para definir los Números naturales

- R0 - Objeto inicial. 0 es un objeto especial y es un Número Natural.
- R1 - Sucesor. Para cada n NN existe otro NN $succ(n)$.
- R2 - Predecesor. Si a es $succ(b)$ entonces b es $pred(a)$.

  Ojo con $pred(k = 1)$, no está definido si no incluimos 0
- R3 - Única. No hay dos NN diferentes con el mismo sucesor.
- R4 - Igualdad. Comparar a y b NN en cuanto a igualdad.
  - Reflexiva $Eq(a, a)$
  - Simétrica $Eq(a, b) \Leftrightarrow Eq(b, a)$
  - Transitiva $Eq(a, b), Eq(b, c) \Leftrightarrow Eq(a, c)$
- R5 - Inducción. Predicado $P(n)$ es cierto $\forall n$ si cumple:

  $P(0)$ cierta.

  $P(n)$ cierta para algún n y se puede demostrar que $P(succ(n))$ es cierta.

### 5.7.4. Eficiencia

No, no busca la eficiencia.

La MT es un modelo de computación abstracta.

Igual sabe hacer cosas que podríamos calcular de otra manera. No obstante, vamos a estudiar la eficiencia $\Rightarrow$ Coste Computacional.

### 5.7.5. Coste Computacional con MT

Uso para el Análisis del Coste Computacional de Algoritmos.

#### Ventajas

- Propone un modelo único, al contrario que con el uso de un Lenguaje X y una CPU Y
- No hay dudas sobre el coste del Paso Base

#### Desventaja

- Es muy pesado para algoritmos complejos ⇒ Limitaremos el tamaño de las MT

#### Propósito

- Relacionar conceptos vistos en la asignatura.
- Basándose en un modelo muy sencillo
- Abstraer de construcciones de muy alto nivel

#### Marco necesario para el Análisis

- MT de cinta infinita hacia ambos lados
- Alfabeto determinado, en principio con pocos símbolos, pero pueden ser más
- Una instancia de un Problema se describirá con una cinta de Entrada
- El número de símbolos de la Entrada define el Tamaño de la Instancia
- Coste de Paso Base: Una Transición con su Lectura/Escritura y Desplazamiento

#### Coste computacional empírico: Aplicando diferencias finitas

| Tamaño n | **1** | **3** | **5** | **7** | **9** | **11** |
|---|---|---|---|---|---|---|
| **Pasos** | 2 | 9 | 20 | 35 | 54 | 77 |
| **A: T(n) - T(n-2)** | | 7 | 11 | 15 | 19 | 23 |
| **B: A(n) - A(n-2)** | | | 4 | 4 | 4 | 4 |
| **C: B(n) - B(n-2)** | | | | 0 | 0 | 0 |

Tabla 5.1: Diferencias finitas

Si $T(n)$ **exponencial**, las diferencias finitas no se anulan, pero aparecen comportamientos particulares.

Si $T(n)$ **polinómico** de orden k, las diferencias finitas de orden $k = cte$. En este caso orden 2.

Sabiendo el orden se plantea un polinomio genérico de ese orden: $T(n) = xn^2 + bn + c$.

Se resuelve la ecuación y ya tenemos la complejidad.

$$a + b + c = 2;\ 9a + 3b + c = 9;\ 25a + 5b + c = 20 \Rightarrow a = \frac{1}{2},\ b = \frac{3}{2},\ c = 0$$

$$T(n) = \frac{1}{2}n^2 + \frac{3}{2}n$$

**Coste computacional empírico: Recursivamente**

Se plantean las observaciones y analizan los patrones de los datos.

- $n = 1 \Rightarrow 2$ pasos
- $n = 3 \Rightarrow 4$ pasos a derecha $+3$ a izquierda $+T(1)$
- $n = 5 \Rightarrow 6$ pasos a derecha $+5$ a izquierda $+T(3)$
- $n \Rightarrow n + 1$ pasos a derecha $+n$ a izquierda $+T(n-2) = \Sigma i \Rightarrow T(n) = \frac{(n+1)(n+2)}{2} - 1 = \frac{1}{2}n^2 + \frac{3}{2}n$

### 5.7.6. Variantes del Modelo de Máquina de Turing

Abordamos:

- Máquinas de Turing de dos o más cintas (multicinta)
- Máquinas de Turing No Deterministas
- Automata de Dos Pilas
- Máquinas de Turing 2D
- Aplicaciones
- Costes Computacionales

### Máquinas de Turing de Dos Cintas

La misma estructura que una MT convencional, pero:

- Tiene dos cintas y sus respectivos cabezales
- Pueden operar de forma independiente

Implica cambios en el funcionamiento, las transiciones se definen:

- Cambio de estado
- Para cada cabezal:
    - Una Lectura
    - Una Escritura
    - Un Desplazamiento

Las operaciones de cada cabezal son independientes entre sí, pero están vinculados en cada transición.

### Máquinas de Turing No Deterministas

La misma estructura que una MT convencional, pero

- Admiten transiciones No Deterministas, es decir, para un mismo símbolo leído define al menos dos transiciones que:
    - Escriben símbolos diferentes
    - Desplazan el cabezal de distinta forma
    - Transitan a estados diferentes

Cuando se encuentra una transición ND, entendemos que la MT bifurca en dos o más instancias de la misma que adoptaran estados diferentes.

Si alguna de las instancias alcanza un Estado Final $\Rightarrow$ MT acepta