Matemática Discreta Tema 1. Algoritmos y números

Martín González Dios

14 de diciembre de 2024

1. Algoritmos

Un algoritmo es una **sucesión finita de instrucciones precisas** (pasos sucesivos definidos) para realizar una tarea que **tiene un final**. Sus propiedades principales son:

- Tiene una **entrada** (input) y produce una **salida** (output y solución) correcta en un tiempo.
- Tiene un **propósito** concreto.
- Los pasos están definidos con **precisión**.
- Se aplica a todos los problemas (**generalidad**).
- Es **efectivo**, realizando cada paso con exactitud y en un tiempo finito.
- Es **eficiente**, se ejecuta en un tiempo polinómico.

1.1. Algoritmos de búsqueda

Estos algoritmos **localizan un elemento** (x) en una lista ordenada de elementos. La solución es la ubicación del elemento (x) en la lista o la declaración de que no está en ella.

- **Búsqueda lineal o secuencial** (lista ordenada): compara x con a_0, si x = a_0 la salida es 0, si no, se compara con el siguiente (a_1, a_2, a_3, ...) hasta encontrar una coincidencia. Si no se encuentra la solución es false.
- **Búsqueda binaria** (lista creciente): compara x con el elemento central de la lista (si hay un número par de elementos elegimos m = $\lfloor n/2 \rfloor$), dividiéndola por ese término, que se incluye en la primera lista y restringiendo la búsqueda a la primera o segunda mitad de la lista (depende de si x es mayor o menor que el elemento central). Se repite el proceso en la sublista correspondiente (que tiene como mucho $\lceil (n+1)/2 \rceil$ elementos) hasta obtener un único elemento (que puede ser x o no encontrarse en la lista). (n es la posición del elemento, que empieza en 0)

1.2. Funciones enteras

- Suelo, parte entera o **floor**: $\lfloor x \rfloor$ es el mayor entero tal que $z \leq x$.
  $\lfloor 3/2 \rfloor = \lfloor 1{,}5 \rfloor = 1$
- Techo o **ceiling**: $\lceil x \rceil$ es el menor entero tal que $z \geq x$.
  $\lceil 3/2 \rceil = \lceil 1{,}5 \rceil = 2$

2. Notación Big-$\mathcal{O}$

Sea f y g funciones del conjunto de los enteros o del conjunto de los números reales al conjunto de los números reales. Decimos que $f(x)$ es $\mathcal{O}(g(x))$ si existen constantes C y k tales que $|f(x)| \leq C|g(x)|$ siempre que x > k. [Esto se lee como "$f(x)$ es big-o de $g(x)$"].

Observación: intuitivamente, la definición de que $f(x)$ es $\mathcal{O}(g(x))$ dice que $f(x)$ **crece más lentamente que algún múltiplo fijo** de $g(x)$ a medida que x crece sin límite.

Las constantes **C y k** de la definición se llaman **testigos de la relación** $f(x)$ es $\mathcal{O}(g(x))$. Para establecer que $f(x)$ es $\mathcal{O}(g(x))$ necesitamos solo un par de testigos para esta relación. Es decir, para demostrar que $f(x)$ es $\mathcal{O}(g(x))$ solo necesitamos encontrar un par de constantes C y k, los testigos, tales que $|f(x)| \leq C|g(x)|$ siempre que x > k.

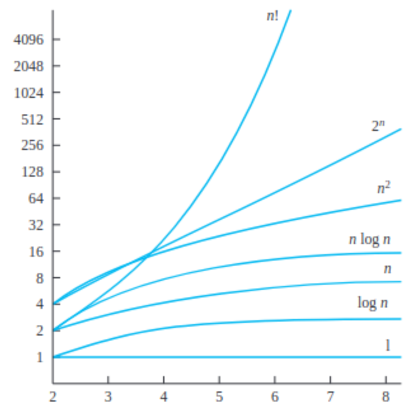

Figura 1: Gráfica de funciones de crecimiento más usadas en Big-$\mathcal{O}$

Cabe destacar que cuando hay un par de testigos para la relación $f(x)$ es $\mathcal{O}(g(x))$, hay infinitos pares de testigos.

2.1. Reglas de notación Big-$\mathcal{O}$

- Si $f_1(x)$ es $\mathcal{O}(g_1(x))$ y $f_2(x)$ es $\mathcal{O}(g_2(x)) \rightarrow (f_1 + f_2)(x)$ es $\mathcal{O}(max\{|g_1(x)|, |g_2(x)|\})$
  Si $f_1(x)$ y $f_2(x)$ es $\mathcal{O}(g(x)) \rightarrow (f_1 + f_2)(x)$ es $\mathcal{O}(g(x))$
- Si $f_1(x)$ es $\mathcal{O}(g_1(x))$ y $f_2(x)$ es $\mathcal{O}(g_2(x)) \rightarrow (f_1 * f_2)(x)$ es $\mathcal{O}(g_1(x) * g_2(x))$
- Si $f(x)$ es $\mathcal{O}(g(x))$ y $g(x)$ es $\mathcal{O}(h(x)) \rightarrow f(x)$ es $\mathcal{O}(h(x))$
- Si $f(x)$ es $\mathcal{O}(g(x)) \rightarrow a * f(x)$ es $\mathcal{O}(g(x))$ para cualquier a

2.2. Ejemplo 1

$f(x) = x^2 + 2x + 1$ es $\mathcal{O}(x^2)$. **Sea $f(x)$ un polinomio de grado n $\rightarrow$ $f(x)$ es $\mathcal{O}(x^n)$**

2.3. Ejemplo 2

Explicar que $7x^2$ es $\mathcal{O}(x^3)$. Cuando x > 7 tenemos que $7x^2 < x^3$ (obteniendo esta inecuación multiplicando los dos lados de x > 7 por x^2). Consecuentemente, podemos tomar C = 1 y k = 7 como testigos de la relación $7x^2$ es $\mathcal{O}(x^3)$.

2.4. Ejemplo 3

n $<2^n$ con cualquier n entero positivo. Explicar que esta inecuación implica que n es $\mathcal{O}(2^n)$ y usar esta inecuación para demostrar que log(n) es $\mathcal{O}(n)$.

Usando la inecuación n $<2^n$ concluimos que n es $\mathcal{O}(2^n)$ tomando k = C = 1. Nótese que, dado que la función del logaritmo es creciente, al tomar logaritmos (base 2) a ambos lados de esta igualdad se muestra que: log(n) < n (Tomando nuevamente k = C = 1 como testigos). $log(n)$ es $\mathcal{O}(n)$

Si tuviésemos logaritmos en base b, donde b es diferente de 2, seguiríamos teniendo que $log_b(n)$ es $\mathcal{O}(n)$ con cualquier n positivo entero. (si está elevado a un c positivo daría igual, sigue siendo $\mathcal{O}(n)$).
($log_b(n)^c$ es $\mathcal{O}(n^d)$ siendo d el grado del polinomio de la función $f(x)$)

2.5. Ejemplo 4

Estimar el Big-$\mathcal{O}$ para $f(n) = 3nlog(n!) + (n^2+3)log(n)$, donde n es un positivo entero.

En primer lugar se estima $3nlog(n!)$. Sabemos que n! $\leq n^n$: log(n!) $\leq$ log(n^n) = n*log(n), por lo tanto log(n!) es $\mathcal{O}(nlog(n))$. 3n es $\mathcal{O}(n)$. Entonces $3n * log(n!)$ es $\mathcal{O}(nlog(n))$.

El siguiente producto a estimar es $(n^2+3)log(n)$. n^2+3 es $\mathcal{O}(n^2)$. Por lo tanto, $(n^2+3)log(n)$ es $\mathcal{O}(n^2log(n))$.

Combinando las soluciones $f(n) = 3nlog(n!) + (n^2+3)log(n)$ es $\mathcal{O}(n^2log(n))$.

3. Notación Big-Ω y Big-Θ

3.1. Big-Ω

Sea f y g funciones del conjunto de enteros o del conjunto de los números reales al conjunto de los números reales. Decimos que $f(x)$ es $\Omega(g(x))$ si hay constantes positivas C y k tales que $|f(x)| \geq C|g(x)|$ para cualquier x > k. [Se lee como "f(x) es big-omega de g(x).º "f(x) es del orden al menos de g(x)"].

Hay una fuerte relación entre big-$\mathcal{O}$ y big-Ω, **de hecho $f(x)$ es Ω(g(x)) si y solo si $g(x)$ es $\mathcal{O}$($f(x)$).**

3.2. Big-Θ

Sea f y g funciones del conjunto de enteros o del conjunto de los números reales al conjunto de los números reales. Decimos que $f(x)$ es $\Theta(g(x))$ si f(x) es $\mathcal{O}(g(x))$ y f(x) es $\Omega(g(x))$. [Se lee como "f(x) es big-theta de g(x).º "f(x) es del orden de g(x)"]

Cuando $f(x)$ es $\Theta(g(x))$ también es el caso de que $g(x)$ es $\Theta(f(x))$.

3.3. Ejemplo 1

Explica que $3x^2 + 8xlog(x)$ es $\Theta(x^2)$.

Porque $0 \leq 8xlog(x) \leq 8x^2$, lo que sigue que $3x^2+8xlog(x) \leq 11x^2$ para x > 1. Consecuentemente, $3x^2 + 8xlog(x)$ es $\mathcal{O}(x^2)$, y claramente x^2 es $\mathcal{O}(3x^2 + 8xlog(x))$.

El término mayor de un polinomio determina su orden. Por ejemplo, si $f(x) = 3x^5 + x^4 + 17x^3 + 2$, por lo tanto $f(x)$ es del orden de x^5.

4. Complejidad de un algoritmo

Un algoritmo debe producir siempre la respuesta correcta y ser eficiente en términos de tiempo y espacio. La eficiencia se mide a través de la **complejidad temporal**, que se refiere al número de operaciones necesarias para resolver un problema de un tamaño específico, y la complejidad espacial, que se relaciona con la memoria requerida.

La complejidad temporal se expresa en función del número de operaciones básicas (como suma, multiplicación y comparación) en lugar de en tiempo real, debido a las variaciones en la velocidad de diferentes computadoras.

TABLE 1 Commonly Used Terminology for the Complexity of Algorithms.

Complexity	*Terminology*
$\Theta(1)$	Constant complexity
$\Theta(\log n)$	Logarithmic complexity
$\Theta(n)$	Linear complexity
$\Theta(n \log n)$	Linearithmic complexity
$\Theta(n^b)$	Polynomial complexity
$\Theta(b^n)$, where $b > 1$	Exponential complexity
$\Theta(n!)$	Factorial complexity

Figura 2: Terminología para la complejidad algorítmica

4.1. Ejemplo 1

Describe la **complejidad temporal del algoritmo de búsqueda lineal**.

ALGORITHM 2 The Linear Search Algorithm.

procedure *linear search*(x: integer, $a_1, a_2, \ldots, a_n$: distinct integers)
$i := 1$
while ($i \leq n$ and $x \neq a_i$)
    $i := i + 1$
if $i \leq n$ **then** *location* $:= i$
else *location* $:= 0$
return *location*{*location* is the subscript of the term that equals x, or is 0 if x is not found}

En cada paso del bucle del algoritmo, se realizan 2 comparaciones: una i $\leq$ n, para ver si se ha alcanzado el final de la lista, y una x $\leq$ a_i, para comparar el elemento x con un término de la lista. Finalmente, se realiza una comparación más i $\leq$ n fuera del bucle. En consecuencia, si x = a_i, se utilizan 2i + 1 comparaciones. Se requieren más comparaciones, **2n + 2, cuando el elemento no está en la lista**. En este caso, se utilizan 2n comparaciones para determinar que x no es a_i, para i = 1, 2, ..., n, se utiliza una comparación adicional para salir del bucle, y se realiza una comparación fuera del bucle. Por lo tanto, cuando x no está en la lista, se utilizan un total de 2n + 2 comparaciones. Así, una búsqueda lineal requiere $\Theta(n)$ comparaciones en el peor de los casos, porque 2n + 2 es $\Theta(n)$.

4.2. Ejemplo 2

¿Cuál es el peor caso de **complejidad del bubble sort** en términos de número de comparaciones hechas?

ALGORITHM 4 The Bubble Sort.

```
procedure bubblesort(a_1, ..., a_n : real numbers with n ≥ 2)
for i := 1 to n − 1
    for j := 1 to n − i
        if a_j > a_{j+1} then interchange a_j and a_{j+1}
{a_1, ..., a_n is in increasing order}
```

El bubble sort ordena la lista mediante una secuencia de pasadas a través de la lista. Durante cada iteración el algoritmo compara sucesivamente los elementos adyacentes, intercambiándolos si es necesario. Cuando **la i-ésima iteración** empieza, se garantiza que los i - 1 elementos más grandes están en las posiciones correctas. Durante esta iteración se utilizan **n - 1 comparaciones**. En consecuencia, el número total de comparaciones utilizadas por el ordenamiento burbuja para ordenar una lista de n elementos es **(n-1)+(n-1)+...+2+1**=$\frac{(n-1)n}{2}$. Cabe destacar que el bubble sort siempre hace muchas comparaciones, ya que continúa aunque la lista esté completamente ordenada en un paso intermedio. Por lo tanto, el bubble sort utiliza (n-1)n/2 comparaciones, por lo que es $\Theta(n^2)$ en el peor caso de complejidad en términos de número de comparaciones usado.

4.3. Ejemplo 3

¿Cuántas **sumas y multiplicaciones** de enteros son usadas en el algoritmo de **multiplicación matricial** para multiplicar dos n x n matrices con entradas enteras?

ALGORITHM 1 Matrix Multiplication.

```
procedure matrix multiplication(A, B: matrices)
for i := 1 to m
    for j := 1 to n
        c_ij := 0
        for q := 1 to k
            c_ij := c_ij + a_iq b_qj
return C {C = [c_ij] is the product of A and B}
```

Hay n^2 entradas en el producto de A y B. Para encontrar cada entrada es necesario un total de n multiplicaciones y n - 1 sumas. Así, un **total de n^3 multiplicaciones y $n^2(n-1)$ sumas son usadas**.

5. Aritmética entera

5.1. División

Si a y b son enteros con $a \neq 0$, decimos que a divide a b si hay **un entero c tal que $b = ab$**, o equivalentemente, que $\frac{a}{b}$ es un entero. Cuando a divide a b decimos que a es un factor de a y b es un múltiplo de a. La notación $a \mid b$ denota que a divide a b. Escribimos $a \nmid b$ cuando a no divide a b.

Teniendo n y d positivos enteros, **¿cuántos positivos enteros que no excedan n son divisibles entre d?**
Los positivos enteros divisibles por d son todos los enteros de la forma dk, donde k es un positivo entero. Así, el número de positivos enteros divisibles por d que no exceda n es igual al número de enteros k con $0 < dk \leq n$, o con $0 < k \leq n/d$. Por lo tanto, **hay $\lfloor n/d \rfloor$ positivos que no excedan n y sean divisibles entre d.**

Sea a, b y c enteros, donde $a \neq 0$:

- Si $a \mid b$ y $a \mid c$, entonces $a \mid (b + c)$.
- Si $a \mid b$, entonces $a \mid bc$ para todos los enteros c.
- Si $a \mid b$ y $b \mid c$, entonces $a \mid c$.
- Si $a \mid b$ y $a \mid c$, entonces $a \mid mb + nc$ con cualquier n y m enteros.

El **algoritmo de la división**:
Sea a un entero y d un positivo entero. Entonces hay unos únicos enteros q y r, con $0 \leq r < d$, tal que $a = dq + r$.

5.2. Números primos

Un entero p mayor que 1 es llamado primo si los únicos divisores positivos son 1 y p. Un número positivo que no sea primo se le llama compuesto.

(**Teorema de Euclides**: existen infinitos primos. **Primos de Mersenne**: 2^p - 1 con p primo, son primos [falso para p = 11 por ejemplo])

Teorema fundamental de la aritmética:
Todo entero positivo > 1 se puede factorizar (descomponer) de manera única como un primo o un producto de números primos donde los factores primos están escritos en orden no decreciente. (Si n es un número compuesto, entonces n tiene un divisor primo $\leq \sqrt{n}$)

5.3. Mínimo Común Múltiplo y Máximo Común Divisor

$MCM(lcm(a, b))$: sean a y b enteros positivos $\neq 0$, el menor entero m tal que $a \mid m$ y $b \mid m$ se llama **MCM** de a y b.

$MCD(gcd(a, b))$: sean a y b enteros positivos $\neq 0$, el mayor entero tal que $d \mid a$ y $d \mid b$ se llama **MCD** de a y b.

Los enteros a y b son **coprimos** (relativamente primos) si su mayor divisor común es 1.

Algoritmo de Euclides: sea $a = bq + r$, donde a, b, q y r son enteros y a $\geq$ b. Entonces **gcd(a, b) = gcd(b, r)**. Sea d un divisor de a y de b, entonces $d \mid r$, por lo tanto a - $bq = r$.
Ejemplo: encontrar el gcd de 442 y 662:
662 = 414·1 + 248
414 = 248·1 + 166
166 = 82·2 + 2
82 = 2·41
Por lo tanto, **gcd(414, 662) = 2, porque 2 es el último residuo no nulo**.

ALGORITHM 1 The Euclidean Algorithm.

procedure *gcd*(a, b: positive integers)
$x := a$
$y := b$
while $y \neq 0$
    $r := x$ **mod** y
    $x := y$
    $y := r$
return x\{gcd(a, b) is x\}

Teorema de Bézout: sea a y b enteros positivos, $\exists$ enteros s y t tal que gcd(a, b) = $s \cdot a + t \cdot b$ (siendo s y t coeficientes de Bézout).
Ejemplo: expresar gcd(252, 198) = 18 como combinación lineal de 252 y 198.
252 = 1·198 + 54
198 = 3·54 + 36
54 = 1·36 + 18
36 = 2·18

18 = 54 - 1·36 (tercera división)
36 = 198 - 3·54 (segunda división)

18 = 54 - 1·36 = 54 - 1·(198 - 3·54) = 3·54 - 1·198
54 = 252 - 1·198 (primera división)

resultado: 18 = 4·(252 - 1·198) - 1·198 = **4**·252 **- 5**·198

Si a, b y c son positivos enteros tal que gcd(a, b) = 1 y $a \mid bc$, entonces $a \mid c$.

6. Arimtética modular

Si a y b son enteros y m es un positivo entero, entonces a es **congruente** con b módulo m si m divide a-b ($m \mid a - b$). Usamos la notación $a \equiv b$ (mod m) para indicar lo anterior. Se dice que $a \equiv b$ (mod m) es una **congruencia** y que m es su módulo.

Sean a y b enteros positivos $a \equiv b$ (mod m) entonces a (mod m) = b (mod m).
Ejemplo: $34 \equiv 46$ (mod 12). 46 mod 12 = 34 mod 12 (10)

Sea m entero positivo, si $a \equiv b$ (mod m) entonces $a \pm c \equiv b \pm d$ (mod m).
Ejemplo: m = 12, a = 13, b = 1, c = 5, d = 17.
$13 \equiv 25$ (mod 12)
$5 \equiv 17$ (mod 12)
13 + 5 = 18 y 25 + 17 = 42
$18 \equiv 42$ (mód 12)

Sea m un entero positivo y a y b enteros. Entonces
$(a + b)$ mod(m) = ((a mod(m)) + (b mod(m))) mod(m) y
ab mod(m) = ((a mod(m))(b mod(m))) mod (m).
Ejemplo: (25 + 17) mod(12) = (25 mod (12) + 17 mod(12)) mod(12) = (1 + 5)mod(12) = 6

6.1. Aritmética módulo m

Por una parte: $a_m + b :=$ (a + b) mod(m), y por otra: $a_m \cdot b :=$ (a · b) mod(m).
Los "números buenos" o unidades de Z_m son primos relativos con m, el resto son "malos". En Z_m hay p - 1 unidades (todos menos el 0) si m es primo y $\Phi(m)$ si no lo es, las unidades son los elementos que tienen inverso multiplicativo. (a es una unidad si $\exists$ un b tal que $a \cdot b = 1$, siendo así a inversible).
$m = p^k$, y $\Phi(p^k) = p^{k-1}(p-1)$

Teorema fundamental de la aritmética: $m = p_1^{\alpha 1} \cdot p_2^{\alpha 2} \cdot ... \cdot p_s^{\alpha s}$ (p_i primos diferentes).
$\Phi(m) = \Phi(p_1^{\alpha 1}) \cdot \Phi(p_2^{\alpha 2}) \cdot ... \cdot \Phi(p_s^{\alpha s})$
$\Phi_{12} = \Phi(2^2) \cdot \Phi(3) = 2^{2-1} \cdot (2-1) \cdot 3^{1-1} \cdot (3-1) = 2 \cdot 2 = 4$. $(12 = 2^2 \cdot 3)$ En Z_{12} hay 4 números buenos.

6.2. Congruencias lineales

Sea $ax \equiv b$ mod(m):
$d = \gcd(a, m) \nmid b$ (**no tiene solución**): 2x $\equiv$ 3 mod(6), gcd(2, 6) = 2 $\nmid$ 3.
$d = \gcd(a, m) \mid b$ (**tiene d soluciones**): 2x $\equiv$ 4 mod(6), gcd(2, 6) = 2 | 4.
$\frac{a}{d}x \equiv \frac{b}{d}mod(\frac{m}{d})$ (**tiene una única solución** x_0): 237x $\equiv$ 4 mod(491), gcd(237, 491) = $1 = s \cdot 237 + t \cdot 491$,$4 = 4s \cdot 237 + 4t \cdot 491$. (gcd($\frac{a}{d}$, $\frac{m}{d}$) = 1).

6.3. Pequeño teorema de Fermat

Si p es primo y $p \nmid a$, entonces $a^{p-1} \equiv 1 mod(p)$.
Ejemplo: 13 $\nmid$ 237 (p = 13, a = 237). $237^{13-1} = 237^1 2 = 1 mod(13)$

6.4. Euler

Sea m primo, si gcd(a, m) = 1, entonces $a^{\Phi(m)} \equiv 1 mod(m)$.
gcd(a, p) = 1 $\Leftrightarrow p \nmid a$
$\Phi(p) \equiv 1 mod(p)$
$\Phi(p) = p - 1$.

6.5. Teorema chino de los restos

Sean m_1, m_2, ..., m_n primos relativos entre ellos. Sean a_1, a_2, ..., a_k enteros. Entonces el sistema (el de abajo) tiene una única solución módulo m = $m_1 \cdot m_2 \cdot m_3 \cdot ... \cdot m_n$ $(0 \leq x < m)$

$$\begin{cases} x \equiv a_1 \mod m_1 \\ x \equiv a_2 \mod m_2 \\ ... \\ x \equiv a_k \mod m_k \end{cases}$$

x = $a_1 \cdot \frac{m}{m_1} \cdot [\frac{m}{m_1}]^{-1}_{m_1} + a_2 \cdot \frac{m}{m_2} \cdot [\frac{m}{m_2}]^{-1}_{m_2} + ... + a_k \cdot \frac{m}{m_k} \cdot [\frac{m}{m_k}]^{-1}_{m_k}$

Ejemplo:

$$\begin{cases} x \equiv 2 \mod 3 \\ x \equiv 3 \mod 5 \\ x \equiv 2 \mod 7 \end{cases}$$

x = $2 \cdot \frac{105}{3} \cdot [\frac{105}{3}]^{-1}_3 + 3 \cdot \frac{105}{5} \cdot [\frac{105}{5}]^{-1}_5 + 2 \cdot \frac{105}{7} \cdot [\frac{105}{7}]^{-1}_7$

x = $2 \cdot 35 \cdot [35]^{-1}_3 + 3 \cdot 21 \cdot [21]^{-1}_5 + 2 \cdot 15 \cdot [15]^{-1}_7 = 2 \cdot 35 \cdot 2 + 3 \cdot 21 \cdot 1 + 2 \cdot 1 = 233$

$233 \equiv 23 \mod 105$

7. Criptografía

Disciplina que trata el estudio de códigos cifrados. Existen diferentes tipos de sistemas criptográficos:

- **Cifrado de clave simétrica** (una única clave privada): dentro de esta categoría existen: cifrado por sustitución (reemplaza un carácter por otro) o el cifrado por traslación (desplaza el alfabeto un número de posiciones). En este último tipo de cifrado se encuentra el **cifrado César**, donde cada letra del texto se desplaza un número fijo de posiciones en el alfabeto. Por ejemplo, con un desplazamiento de 3, la letra A se convierte en D.
- **Cifrado de clave asimétrica** (dos claves diferentes): se usan funciones unidireccionales, dónde un sentido de la transmisión es fácil, pero el contrario es difícil. Se emplea una clave pública para cifrar el mensaje y se descifra con la clave privada que tiene el receptor. Multiplicamos 2 primos, p y q, dando n, que es muy fácil de obtener, pero es difícil de factorizar para conseguir p y q. Esta es la base del **RSA**.

7.1. RSA

El cifrado RSA se basa en la dificultad de factorizar un número grande. Para ello, se eligen dos números primos grandes, p y q. A partir de estos, se calcula n como:

$$n = p \cdot q$$

Luego, se calcula la función Φ de Euler $\Phi(n)$:

$$\Phi(n) = \Phi(p) \cdot \Phi(q) = (p-1) \cdot (q-1)$$

A continuación, se elige un número e tal que $1 < e < \Phi(n)$ y que sea coprimo con $\Phi(n)$. Esto significa que $\gcd(e, \Phi(n)) = 1$. El número e debe tener un inverso multiplicativo d tal que:

$$d \cdot e \equiv 1 \pmod{\Phi(n)}$$

Esto implica que d es el inverso multiplicativo de e módulo $\Phi(n)$.

Para cifrar un mensaje m, se utiliza la siguiente fórmula:

$$E(m) \equiv m^e \pmod{n}$$

Donde $E(m)$ es el mensaje cifrado. Para descifrar el mensaje, se aplica la siguiente operación:

$$D(c) \equiv c^d \pmod{n}$$

Donde c es el mensaje cifrado. Al aplicar el descifrado al mensaje cifrado, se obtiene:

$$D(E(m)) = E(m)^d = (m^e)^d \equiv m \pmod{n}$$

Esto demuestra que el proceso de cifrado y descifrado es correcto, ya que se recupera el mensaje original m.

Ejemplo:
Supongamos que elegimos los primos $p = 61$ y $q = 53$.
Calculamos n y $\Phi(n)$:

$$n = 61 \cdot 53 = 3233$$

$$\Phi(n) = (61-1)(53-1) = 60 \cdot 52 = 3120$$

Elegimos $e = 17$ (que es coprimo con 3120). Ahora, encontramos d tal que:

$$d \cdot 17 \equiv 1 \pmod{3120}$$

Usando el algoritmo extendido de Euclides, encontramos que $d = 2753$.
Ahora, ciframos un mensaje $m = 123$:

$$E(m) \equiv 123^{17} \pmod{3233}$$

Calculando $E(m)$:

$$E(123) \equiv 855 \pmod{3233}$$

Para descifrar, aplicamos:

$$D(c) \equiv 855^{2753} \pmod{3233}$$

Calculando $D(855)$:

$$D(855) \equiv 123 \pmod{3233}$$

Así, hemos cifrado y descifrado correctamente el mensaje original.

8. Representación de n en base > 1 (expansión)

Sea b un entero mayor que 1. Si a es un entero positivo, entonces un número n se puede expresar de manera única en la forma:

$$n = (a_k \cdot b^k + a_{k-1} \cdot b^{k-1} + \ldots + a_1 \cdot b^1 + a_0 \cdot b^0)$$

donde k es un entero no negativo, $0 \leq a_0, a_1, \ldots, a_k < b$ y $a_k \neq 0$.

- $b = 2$ (Binario): $15_{10} = 1111_2$
- $b = 8$ (Octal): $14_{10} = 2371_8$

Para convertir un número de decimal a una base b, se sigue el siguiente procedimiento:

1. Dividir el número entre b.
2. Anotar el resto.
3. Repetir el proceso con el cociente hasta que este sea 0.
4. Los restos anotados, leídos en orden inverso, representan el número en la base b.

Ejemplo:
Supongamos que queremos convertir el número decimal $n = 100$ a base 7.

1. $101 \div 7 = 14$ con un resto de 3.
2. $14 \div 7 = 2$ con un resto de 0.
3. $2 \div 7 = 0$ con un resto de 2.

Ahora, leemos los restos en orden inverso: $2, 0, 3$.
Por lo tanto, el número 101 en base 7 se representa como:

$$(203)_7$$

9. Criterios de divisibilidad

Los criterios de divisibilidad son reglas que permiten determinar si un número es divisible por otro sin necesidad de realizar la división completa. A continuación se presentan algunos de los criterios más comunes:

- **Divisibilidad por 2:**

$$n = 2 \quad \Rightarrow \quad \text{El número es par.}$$

Un número es divisible por 2 si su última cifra es par (0, 2, 4, 6, 8).

- **Divisibilidad por 3:**

$$n = 3 \quad \Rightarrow \quad 3 \mid S \quad \text{(donde } S \text{ es la suma de las cifras del número).}$$

Un número es divisible por 3 si la suma de sus cifras es divisible por 3.

- **Divisibilidad por 4:**

$$n = 4 \quad \Rightarrow \quad 4 \mid (2 \cdot a_1 + a_0)$$

Un número es divisible por 4 si los dos últimos dígitos forman un número que es divisible por 4.

- **Divisibilidad por 5:**

$$n = 5 \quad \Rightarrow \quad a_0 = 0 \quad \text{o} \quad 5$$

Un número es divisible por 5 si su última cifra es 0 o 5.

- **Divisibilidad por 6:**

$$n = 6 \quad \Rightarrow \quad \text{Divisible por 2 y 3.}$$

Un número es divisible por 6 si es divisible por 2 y por 3.

- **Divisibilidad por 7:**

$$n = 7 \quad \Rightarrow \quad a_k \cdot a_{k-1} \cdots a_1 - 2 \cdot a_0$$

Se toma el número formado por todas las cifras menos la última, se le resta el doble de la última cifra, y si el resultado es divisible por 7, entonces el número original también lo es.

- **Divisibilidad por 8:**

$$n = 8 \quad \Rightarrow \quad 2 \mid a_2 a_1 a_0$$

Un número es divisible por 8 si los últimos tres dígitos forman un número que es divisible por 8.

- **Divisibilidad por 9:**

$$n = 9 \quad \Rightarrow \quad 9 \mid S \quad \text{(donde } S \text{ es la suma de las cifras del número).}$$

Un número es divisible por 9 si la suma de sus cifras es divisible por 9.

- **Divisibilidad por 10:**

$$n = 10 \quad \Rightarrow \quad a_0 = 0$$

Un número es divisible por 10 si su última cifra es 0.

- **Divisibilidad por 11:**

$$n = 11 \quad \Rightarrow \quad 11 \mid (S_p - S_i)$$

Donde S_p es la suma de las cifras en las posiciones pares y S_i es la suma de las cifras en las posiciones impares. Un número es divisible por 11 si la diferencia entre estas dos sumas es divisible por 11.