Ignacio Méndez Pérez
2023

Resumen Diseño y Análisis de Algoritmos

INDICE

Ignacio Méndez Pérez
2023

Programación dinámica + FFT

Sea $\mathcal{A}$ un algoritmo. Asociamos a $\mathcal{A}$ una **función de tiempo de ejecución**

$$\text{tiempo}_{\mathcal{A}}: \Sigma^* \to \mathbb{N}$$

tal que:

$$\text{tiempo}_{\mathcal{A}}(w) \coloneqq \text{ número de pasos realizados por } \mathcal{A} \text{ con entrada } w \in \Sigma^*$$

Definición:

Sea $f: \mathbb{N} \to \mathbb{R}_0^+$ una función. Se define el **conjunto $\mathcal{O}(f)$** tal que:

$$\boxed{\mathcal{O}(f) = \{\, g: \mathbb{N} \to \mathbb{R}_0^+ \mid \exists c \in \mathbb{R}^+. \exists n_0 \in \mathbb{N}.\ \forall n \geq n_0.\ g(n) \leq c \cdot f(n) \,\}}$$

Decimos entonces que $g \in \mathcal{O}(f)$.

- También usamos la notación g es $\mathcal{O}(f)$, lo cual es formalizado como $g \in \mathcal{O}(f)$.

Definición:

Sea $f: \mathbb{N} \to \mathbb{R}_0^+$ una función. Se definen los **conjuntos $\Omega(f)$ y $\Theta(f)$** tal que:

$$\begin{aligned} \Omega(f) &= \{\, g: \mathbb{N} \to \mathbb{R}_0^+ \mid \exists c \in \mathbb{R}^+. \exists n_0 \in \mathbb{N}.\ \forall n \geq n_0.\ c \cdot f(n) \leq g(n) \,\} \\ \Theta(f) &= \mathcal{O}(f) \cap \Omega(f) \end{aligned}$$

Búsqueda binaria:

BúsquedaBinaria(a, L, i, j)
  if $i > j$ **then return** no
  else if $i = j$ **then**
    if $L[i] = a$ **then return** i
    else return no
  else
    $p := \lfloor \frac{i+j}{2} \rfloor$
    if $L[p] < a$ **then return BúsquedaBinaria**$(a, L, p+1, j)$
    else if $L[p] > a$ **then return BúsquedaBinaria**$(a, L, i, p-1)$
    else return p

Llamada inicial al algoritmo: **BúsquedaBinaria**$(a, L, 1, n)$

Si contamos solo las comparaciones, entonces la complejidad del algoritmo se define como:

$$T(n) = \begin{cases} c & n = 1 \\ T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + d & n > 1 \end{cases}$$

donde $c \in \mathbb{N}$ y $d \in \mathbb{N}$ son constantes tales que $c \geq 1$ y $d \geq 1$.
Esta es una **ejecución de recurrencia**.

Solucionando la ecuación:

Técnica básica sustitución de variables.
Para la ecuación anterior usamos la sustitución $n = 2^k$.

- Suponemos que n es potencia de 2.
- Utilizaremos inducción.

$$T(2^k) = \begin{cases} c & k = 0 \\ T(2^{k-1}) + d & k > 0 \end{cases}$$

Expandimos:

$$\begin{aligned} T(2^k) &= T(2^{k-1}) + d \\ &= (T(2^{k-2}) + d) + d \\ &= T(2^{k-2}) + 2d \\ &= (T(2^{k-3}) + d)2d \\ &= T(2^{k-3}) + 3d \end{aligned}$$

Decimos que la expresión general para $k - i \geq 0$:

$$T(2^k) = T\left(2^{k-i}\right) + i \cdot d$$

Considerando $i = k$ obtenemos:

$$T(2^k) = T(1) + k \cdot d$$

Dado que $k = \log_2(n)$ (por cambio de variable), obtenemos que:

$$T(n) = c + d \cdot \log_2(n)$$

para n potencia de 2.

Queremos demostrar que $T(n) \in \mathcal{O}(\log_2(n))$

- Vale decir, tenemos que demostrar que existen $e \in \mathbb{R}^+$ y $n_0 \in \mathbb{N}$ tales que $T(n) \leq e \cdot \log_2(n)$ para todo $n \geq n_0$.

Inducción constructiva:

Dado que $T(1) = c$ y $\log_2(1) = 0$ no es posible encontrar un valor para e tal que $T(1) \leq e \cdot \log_2(1)$.

Dado que $T(2) = c + d$, si consideramos $e = (c + d)$ tenemos que $T(2) \leq e \cdot \log_2(2)$.

- Definimos entonces $e = (c + d)$ y $n_0 = 2$.

Tenemos que demostrar lo siguiente:

$$\forall n \geq 2.\ T(n) \leq e \cdot \log_2(n)$$

Para esto, utilizaremos inducción fuerte.

- $n = 2$ es el punto de partida y el primer caso base.
- $n = 3$ también es un caso base ya que $T(3) = T(1) + d$ y para $T(1)$ no se cumple la propiedad.
- Para $n \geq 4$ tenemos que $T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + d$ y $\left\lfloor \frac{n}{2} \right\rfloor \geq 2$, por lo que resolvemos este caso de manera inductiva.
    - Suponemos que la propiedad se cumple para todo $k \in \{2, \dots, n-1\}$.

Casos base:

- $T(2) = \quad c + d \quad = \quad e \cdot \log_2(2)$
- $T(3) = \quad c + d \quad < \quad e \cdot \log_2(3)$

Caso inductivo:

Suponemos que $n \geq 4$ y para todo $k \in \{2, \dots, n-1\}$ se tiene que $T(k) \leq e \cdot \log_2(k)$.

Usando la definición de $T(n)$ y la hipótesis de inducción concluimos que:

$$\begin{aligned} T(n) = &\ T\left(\left\lfloor\frac{n}{2}\right\rfloor\right) + d \\ &\le e \cdot \log_2\left(\left\lfloor\frac{n}{2}\right\rfloor\right) + d \\ &\le e \cdot \log_2\left(\frac{n}{2}\right) + d \\ &= e \cdot \log_2(n) - e \cdot \log_2(2) + d \\ &= e \cdot \log_2(n) - (c + d) + d \\ &= e \cdot \log_2(n) - c \\ &< e \cdot \log_2(n) \end{aligned}$$

■

El Teorema Maestro

Muchas de las ecuaciones de recurrencia que vamos a usar en este curso son de la forma:

$$T(n) = \begin{cases} c & n = 0 \\ a \cdot T\left(\left\lfloor\frac{n}{b}\right\rfloor\right) + f(n) & n \ge 1 \end{cases}$$

El **Teorema Maestro** nos dirá cuál es el **orden** de $T(n)$ dependiendo de ciertas condiciones sobre a, b y $f(n)$.

Antes...

Sea $f: \mathbb{N} \to \mathbb{R}_0^+$ una función y $a, b \in \mathbb{R}$ constantes tales que $a \ge 1$ y $b > 1$.

Definición:

La función f es **(a, b)-regular** si existen constantes $c \in \mathbb{R}^+$ y $n_0 \in \mathbb{N}$ tales que $c < 1$ y

$$\forall n \ge n_0. \quad a \cdot f\left(\left\lfloor\frac{n}{b}\right\rfloor\right) \le c \cdot f(n)$$

Ejercicio:

Demuestre que la función $\log_2(n)$ no es $(1,2)$-regular.

Solución:

Por contradicción, supongamos que $\log_2(n)$ es $(1{,}2)$-regular.
Entonces existen constantes $c \in \mathbb{R}^+$ y $n_0 \in \mathbb{N}$ tales que $c < 1$ y

$$\forall n \geq n_0. \log_2 \left\lfloor \frac{n}{2} \right\rfloor \leq c \cdot \log_2(n)$$

En particular, podemos concluir que para todo $k \geq n_0$:

$$\log_2 \left\lfloor \frac{2 \cdot k}{2} \right\rfloor \leq c \cdot \log_2(2 \cdot k)$$

Vale decir:

$$\log_2(k) \leq c \cdot (\log_2(k) + 1)$$

Dado que $0 < c < 1$:

$$\log_2(k) \leq \frac{c}{1 - c}$$

Lo cual nos lleva a una contradicción.

Teorema Maestro:

Sea $f: \mathbb{N} \to \mathbb{R}_0^+$ una función, $a, b, c \in \mathbb{R}_0^+$ constantes tales que $a \geq 1$ y $b > 1$, y $T(n)$ una función definida por la siguiente ecuación de recurrencia:

$$T(n) = \begin{cases} c & n = 0 \\ a \cdot T\left(\left\lfloor \frac{n}{b} \right\rfloor\right) + f(n) & n \geq 1 \end{cases}$$

Se tiene que:

1. Si $f(n) \in \mathcal{O}(n^{\log_b(a) - \varepsilon})$ para $\varepsilon > 0$, entonces $T(n) \in \Theta\left(n^{\log_b(a)}\right)$.

2. Si $f(n) \in \Theta(n^{\log_b(a)})$, entonces $T(n) \in \Theta\left(n^{\log_b(a)} \cdot \log_2(n)\right)$.

3. Si $f(n) \in \Omega(n^{\log_b(a) + \varepsilon})$ para $\varepsilon > 0$ y f es (a, b)-regular, entonces $T(n) \in \Theta\left(f(n)\right)$.

Ejemplo:

Considere la siguiente ecuación de recurrencia:

$$T(n) = \begin{cases} 1 & n = 0 \\ 3 \cdot T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + c \cdot n & n \geq 1 \end{cases}$$

Dado que $\log_2(3) > 1.5$, tenemos que $\log_2(3) - 0.5 > 1$
Deducimos que $c \cdot n \in \mathcal{O}\left(n^{\log_2(3)-0.5}\right)$, por lo que usando el Teorema Maestro concluimos que$T(n) \in \Theta\left(n^{\log_2(3)}\right)$.

El Teorema Maestro sigue siendo válido pero con $T\left(\left\lfloor \frac{n}{b} \right\rfloor\right) + f(n)$ reemplazado por $T\left(\left\lceil \frac{n}{b} \right\rceil\right) + f(n)$

Sea $\mathcal{A}: \Sigma^* \to \Sigma^*$ un algoritmo:

Definición

Decimos que $\mathcal{A}$ en el **peor caso** es $\mathcal{O}(f(n))$ si:

$$t_{\mathcal{A}}(n) \in \mathcal{O}(f(n))$$

Recordar que $t_{\mathcal{A}}(n)$ es el mayor número de pasos realizados por $\mathcal{A}$ sobre las entradas $w \in \Sigma^*$ de largo n.

Dividir para conquistar

Algoritmo genérico:

DividirParaConquistar(w)
    if $|w| \leq k$ **then return InstanciasPequeñas**(w)
    else
        Dividir w en $w_1, \ldots, w_\ell$
        for $i := 1$ **to** ℓ **do**
            $S_i :=$ **DividirParaConquistar**(w_i)
        return Combinar$(S_1, \ldots, S_\ell)$

Algoritmo de multiplicación de Karatsuba

Sean $a, b \in \mathbb{Z}$ con n dígitos cada uno, donde $n = 2^k$ para algún $k \in \mathbb{N}$.
Se puede representar a y b de la siguiente forma:

$$a = a_1 \cdot 10^{\frac{n}{2}} + a_2$$
$$b = b_1 \cdot 10^{\frac{n}{2}} + b_2$$

Tenemos entonces que:

$$\boxed{a \cdot b = a_1 \cdot b_1 \cdot 10^n + (a_1 \cdot b_2 + a_2 \cdot b_1) \cdot 10^{\frac{n}{2}} + a_2 \cdot b_2}$$

Para calcular $a \cdot b$ debemos calcular las siguiente multiplicaciones:

1. $a_1 \cdot b_1$
2. $a_1 \cdot b_2$
3. $a_2 \cdot b_1$
4. $a_2 \cdot b_2$

Podemos calcular $a \cdot b$ realizando las siguiente multiplicaciones:

1. $c_1 = a_1 \cdot b_1$
2. $c_2 = a_2 \cdot b_2$
3. $c_3 = (a_1 + a_2) \cdot (b_1 + b_2)$

Tenemos entonces que:

$$\boxed{a \cdot b = c_1 \cdot 10^n + \big(c_3 - (c_1 + c_2)\big) \cdot 10^{\frac{n}{2}} + c_2}$$

Esta expresión se conoce como el **algoritmo de Karatsuba**.

Tiempo de ejecución del algoritmo de Karatsuba:

$$T(n) = \begin{cases} 1 & n = 1 \\ 3 \cdot T\left(\frac{n}{2}\right) + e \cdot n & n > 1 \end{cases}$$

Supuesto:

- n es una potencia de 2 y $(a_1 + a_2)$ y $(b_1 + b_2)$ tienen $\frac{n}{2}$ dígitos cada uno.

¿Qué representa la constante e?

- Calcular $(a_1 + a_2), (b_1 + b_2), (c_1 + c_2)$ y $\left(c_3 - (c_1 + c_2)\right)$.
- Construir $a \cdot b$ a partir de $c_1,\ c_2$ y $\left(c_3 - (c_1 + c_2)\right)$, lo cual puede tomar tiempo lineal en el pero caso.

Utilizando el Teorema Maestro obtenemos que $T(n)$ es $\Theta\left(n^{\log_2(3)}\right)$, pero este resultado es válido bajo los supestos realizados anteriormente.

Caso general:

Representamos las entradas a y b de la siguiente forma:

$$a = a_1 \cdot 10^{\left\lfloor \frac{n}{2} \right\rfloor} + a_2$$
$$b = b_1 \cdot 10^{\left\lfloor \frac{n}{2} \right\rfloor} + b_2$$

La siguiente ecuación de recurrencia para $T(n)$ captura la cantidad de operaciones relaizadas por el algoritmo (para constantes e_1, e_2):

$$T(n) = \begin{cases} e_1 & n \leq 3 \\ T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + T\left(\left\lceil \frac{n}{2} \right\rceil\right) + T\left(\left\lceil \frac{n}{2} \right\rceil + 1\right) + e_2 \cdot n & n > 3 \end{cases}$$

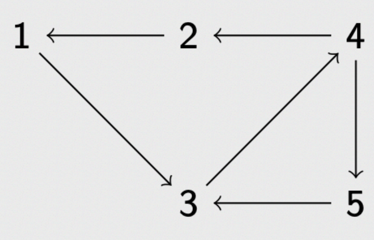
Programación dinámica: Grafos

Sea $G = (V, E)$, un par de nodos $v_i, v_f \in V$, y un número l, queremos desarrollar un algoritmo que cuente el número de caminos desde v_i a v_f en G cuyo largo es igual a l.

Suponemos que $V = \{1, \dots, n\}, 1 \leq l \leq n$ y represenatmos G a través de su **matriz de adyacencia** M tal que:

Si $(i, j) \in E$, entonces $M[i, j] = 1$, en caso contrario $M[i, j] = 0$.

Ejemplo:



$$M = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Queremos calcular el número de caminos de largo l desde un nodo v_i, a un nodo v_f en un grafo G (representado por una matriz de adyacencia M)
Para evitar hacer llamadas recursivas repetidas, y así disminuir el número de llamadas recursivas, definimos una **secuencia de matrices** $H_1, \dots, H_l$ tales que:

$$H_k[v_i, v_f] \quad := \quad \text{número de caminos de } v_i \text{ a } v_j \text{ de } largo\ k$$

La secuencia $H_1, \dots, H_k$ s puede definir recursivamente de la siguiente forma:

1. $H_1 = M$.
2. $H_{k+1} = M \cdot H_k$ para $k \in \{1, \dots, l-1\}$.

Luego, la cantidad de caminos de largo l desde v_i a v_f será $\boxed{H_l[v_i, v_f]}$.

```
ContarTodosCaminos(M[1...n][1...n], ℓ)
    if ℓ = 1 then return M
    else
        H := ContarTodosCaminos(M, ℓ − 1)
        return M · H
```

```
ContarCaminos(M[1...n][1...n], v_i, v_f, ℓ)
    H := ContarTodosCaminos(M, ℓ)
    return H[v_i, v_f]
```

Ignacio Méndez Pérez
2023

Programación dinámica: Palabras

Midiendo la distancia entre dos palabras.
Vamos a utilizar la **distancia de Levenshtein** para medir cuán similares son dos palabras.

Dadas dos palabras $w_1, w_2 \in \Sigma^*$, utilizamos la notación $\mathrm{ed}(w_1, w_2)$ para la edit distance entre w_1 y w_2.

Tres operadores sobre palabras:

Para $i \in \{1, \dots, n\}$ y $b \in \Sigma$ tenemos que:

1. $\mathrm{eliminar}(w, i) \quad = \quad a_1 \cdots a_{i-1} \cdot a_{i+1} \cdots a_n.$
2. $\mathrm{agregar}(w, i, b) \quad = \quad a_1 \cdots a_i \cdot b \cdot a_{i+1} \cdots a_n.$
3. $\mathrm{cambiar}(w, i, b) \quad = \quad a_1 \cdots a_{i-1} \cdot b \cdot a_{i+1} \cdots a_n.$

Dadas palabras $w_1, w_2 \in \Sigma^*$, definimos $\mathrm{ed}(w_1, w_2)$ como el menor número de operaciones $\mathrm{eliminar}$, $\mathrm{agregar}$ y $\mathrm{cambiar}$ que aplicadas desde w_1 generan w_2. Para calcular esta distancia utilizamos **programación dinámica**.

Definimos el **infijo** (*substring*) de w entre las posiciones i y j como:

$$w[i, j] = \begin{cases} w[i] \cdots w[j] & 1 \leq i \leq j \leq n \\ \varepsilon & \text{en caso contrario} \end{cases}$$

Fije dos strings $w_1, w_2 \in \Sigma^*$ tales que $|w_1| = m$ y $|w_2| = n$.

Dados $i \in \{0, \dots, m\}$ y $j \in \{0, \dots, n\}$, definimos:

$$\mathrm{ed}(i, j) = \mathrm{ed}(w_1[1, i], w_2[1, j])$$

Observe que $\mathrm{ed}(w_1, w_2) = \mathrm{ed}(m, n)$

Además, definimos el valor $\mathrm{dif}(i, j)$ como 0 si $w_1[i] = w_2[j]$, y como 1 en caso contrario.

Definición recursiva de la distancia de Levenshtein

Del principio de optimalidad para sub=secuencias obtenemos la siguiente **definición recursiva** para la función ed:

$$\mathrm{ed}(i,j) = \begin{cases} \max\{i,j\} & i = 0 \text{ o } j = 0 \\ \min \begin{array}{l} \{1 + \mathrm{ed}(i-1,j), \\ \ 1 + \mathrm{ed}(i,j-1), \\ \ \mathrm{dif}(i,j) + \mathrm{ed}(i-1,j-1)\} \end{array} & i > 0 \text{ y } j > 0 \end{cases}$$

Tenemos entonces una forma de calcular la función ed que se basa en resolver sub-problemas más pequeños.

- Estos sub-problemas están traslapados y se tiene un número polinomial de ellos, podemos aplicar entonces programación dinámica.

```
EditDistance(w1, i, w2, j)
    if i = 0 then return j
    else if j = 0 then return i
    else
        r := EditDistance(w1, i − 1, w2, j)
        s := EditDistance(w1, i, w2, j − 1)
        t := EditDistance(w1, i − 1, w2, j − 1)
        if w1[i] = w2[j] then d := 0
        else d := 1
        return min{1 + r, 1 + s, d + t}
```

¿Es esta una buena implementación de **EditDistance**?

R: No porque tenemos muchas llamadas recursivas repetidas, es mejor evaluar esta función utilizando un **enfoque bottom-up**.

Evaluación bottom-up:

Para determinar los valores de la función ed construimos una tabla siguiendo un orden lexicográfico para los pares (i,j):

$$(i_1, j_1) < (i_2, j_2) \quad \text{si y solo si} \quad i_1 < i_2 \quad \text{o} \quad (i_1 = i_2 \text{ y } j_1 < j_2)$$

Por ejemplo, para determinar el valor de $\mathrm{ed}(\mathrm{casa}, \mathrm{asado})$ construimos la siguiente tabla:

		a	s	a	d	o
	0	1	2	3	4	5
c	1	1	2	3	4	5
a	2	1	2	2	3	4
s	3	2	1	2	3	4
a	4	3	2	1	2	3

		a	s	a	d	o
	0	1	2	3	4	5
c	1	1	2	3	4	5
a	2	1	2	2	3	4
s	3	2	1	2	3	4
a	4	3	2	1	2	3

Estas operaciones son la siguientes:

1. $\mathrm{eliminar}(\mathrm{casa}, 1) \quad = \quad \mathrm{asa}$
2. $\mathrm{agregar}(\mathrm{asa}, 3, d) \quad = \quad \mathrm{asad}$
3. $\mathrm{agregar}(\mathrm{asad}, 4{,}0) \quad = \quad \mathrm{asado}$

Corolario:

Utilizando programación dinámica es posible construir un algoritmo para calcular $\mathrm{ed}(w_1, w_2)$ que en el peor caso es $\Theta(|w_1| \cdot |w_2|)$.

Algoritmos codiciosos

Almacenamiento de Datos:

Sea Σ un alfabeto. Dada una palabra $w \in \Sigma^*$ suponga que queremos **almacenar** w **utilizando los símbolos 0 y 1**.

Definimos entonces una función $\tau: \Sigma \to \{0,1\}^*$ que asigna a cada símbolo en $a \in \Sigma$ una palabra en $\tau(a) \in \{0,1\}^*$ con $\tau(a) \neq \varepsilon$.

- Vamos a almacenar w reemplazando cada símbolo $a \in \Sigma$ que aparece en w por $\tau(a)$.
- Llamamos a τ una **Σ-codificación**.

La $\boldsymbol{extensión\ \hat{\tau}}$ de una Σ-codificación τ a todas las palabras $w \in \Sigma^*$ se define como:

$$\hat{\tau}(w) = \begin{cases} \varepsilon & w = \varepsilon \\ \tau(a_1)\cdots\tau(a_n) & w = a_1 \cdots a_n \ \text{ con } n \geq 1 \end{cases}$$

Vamos a almacenar w como $\hat{\tau}(w)$.

- Si la Σ-codificación τ esta fija (como en ASCII) entonces no es necesario almacenarla.
- Si τ no está fija, entocnes debemos almacenarla junto con $\hat{\tau}(w)$
  - En general $|w|$ es mucho más grande que $|\Sigma|$, por lo que el costo de almacenar τ es despreciable.

La función $\hat{\tau}$ debe especificar una traducción no ambigua:

$$\forall w_1, w_2 \in \Sigma^*. w_1 \neq w_2 \Rightarrow \hat{\tau}(w_1) \neq \hat{\tau}(w_2)$$

De esta forma podemos reconstruir el texto original dada su traducción
Vale decir, $\hat{\tau}$ debe ser una $\boldsymbol{función\ inyectiva}$.

Codificaciones de largo fijo

Es claro que para lograr una traducción no ambigua necesitamos que $\tau(a) \neq \tau(b)$ para cada $a, b \in \Sigma$ tal que $a \neq b$.

Para obtener la misma propiedad para $\hat{\tau}$ podemos imponer la siguiente condición:

$$\forall a, b \in \Sigma. |\tau(a)| = |\tau(b)|$$

Si se cumple esta condición entonces diremos que τ es una $\boldsymbol{\Sigma - codificación\ de\ largo\ fijo}$.

Por otro lado, decimos que τ es una **Σ-codificación de largo variable** si

$$a, b \in \Sigma. \quad |\tau(a)| \neq |\tau(b)|$$

¿Por qué nos conviene utilizar codificaciones de **largo variable**?

R: Podemos obtener representaciones más cortas para la palabra que queremos almacenar.

Ejemplo:

Suponga que $w = aabaacaab$

- Para $\tau_1(a) = 00$, $\tau_1(b) = 01$ y $\tau_1(c) = 10$ tenemos que:

$$\hat{\tau}(w) = 000001000010000001$$

- Para $\tau_2(a) = 0$, $\tau_2(b) = 10$ y $\tau_2(c) = 11$ tenemos que:

$$\hat{\tau}(w) = 001000110010$$

Por lo tanto $|\hat{\tau}_2(w)| = 12 < 18 = |\hat{\tau}_1(w)|$.

Lema:

Si existen $w_1, w_2 \in \Sigma^*$ tales que $w_1 \neq w_2$ y $\hat{\tau}(w_1) = \hat{\tau}(w_2)$ entonces existen $a, b \in \Sigma$ tales que $a \neq b$ y $\tau(a)$ es un prefijo de $\tau(b)$.

<u>Demostración</u>

Suponga que $w_1 \neq w_2$, $\hat{\tau}(w_1) = \hat{\tau}(w_2)$ y

$$\begin{array}{rclr} w_1 & = & a_1 \dots a_m & m \geq 1 \\ w_2 & = & a_1 \dots a_n & n \geq 1 \end{array}$$

Además, sin pérdida de generalidad suponga que $m \leq n$.

Si w_1 es **prefijo propio** de w_2 entonces $\hat{\tau}(w_1)$ es prefijo propio de $\hat{\tau}(w_2)$

- Puesto que $\tau(a) \neq \varepsilon$ para cada $a \in \Sigma$.

Dado que $\hat{\tau}(w_1) = \hat{\tau}(w_2)$, tenemos entonces que w_1 no es prefijo propio de w_2.

Ahora, sea $k = \min_{1 \leq i \leq m} a_i \neq b_i$

k está bien definido puesto que $w_1 \neq w_2$ y w_1 no es prefijo propio de w_2.

Dado que $\hat{\tau}(a_1 \dots a_{k-1}) = \hat{\tau}(b_1 \dots b_{k-1})$ y $\hat{\tau}(w_1) = \hat{\tau}(w_2)$, concluimos que $\hat{\tau}(a_k \dots a_m) = \hat{\tau}(b_k \dots b_n)$.

Tenemos entonces que $\tau(a_k)\cdots\tau(a_m) = \tau(b_k)\cdots\tau(b_n)$, de lo cual se deduce que $\tau(a_k)$ es un prefijo de $\tau(b_k)$, o $\tau(b_k)$ es un prefijo de $\tau(a_k)$.

Lo cual concluye la demostración puesto que $a_k \neq b_k$. ■

Decimos que una Σ-codificación τ es **libre de prefijos** si para cada $a, b \in \Sigma$ tales que $a \neq b$, se tiene que $\tau(a)$ no es prefijo de $\tau(b)$.

<u>Ejemplo</u>:
Para $\Sigma = \{a, b, c\}$ y $\tau(a) = 0,\ \tau(b) = 10,\ \tau(c) = 11$, se tiene que τ es libre de prefijos.

Corolario:

Si τ es una codificación libre de prefijos, entonces $\hat{\tau}$ es una **función inyectiva**.

Frecuencias relativas de los símbolos

Palabra a almacenar: $w \in \Sigma^*$

Para $a \in \Sigma$ definimos $\mathrm{fr}_w(a)$ como la frecuencia relativa de a en w, vale decir, el número de apariciones de a en w dividido por el largo de w.

- Por ejemplo, si $w = aabaabaac$, entonces $\mathrm{fr}_w(a) = \frac{2}{3}$ y $\mathrm{fr}_w(c) = \frac{1}{9}$.

Para una Σ-codificación τ definimos el largo promedio para w como:

$$\mathrm{lp}_w(\tau) = \sum_{a \in \Sigma} \mathrm{fr}_w(a) \cdot |\tau(a)|$$

Tenemos que $|\hat{\tau}(w)| = \mathrm{lp}_w(\tau) \cdot |w|$

- Por lo tanto queremos una Σ-codificación τ que minimice $\mathrm{lp}_w(\tau)$.

Problema de optimización a resolver

Dado $w \in \Sigma^*$, encontrar una Σ-codificación τ libre de prefijos que minimice el valor $\mathrm{lp}_w(\tau)$.

La función objetivo del algoritmo codicioso es entonces $\mathrm{lp}_w(x)$.

- Queremos minimizar el valor de esta función.

La función $\mathrm{lp}_w(x)$ se define a partir de la función $\mathrm{fr}_w(y)$.

- No se necesita más información sobre w. En particular, no se necesita saber cuál es el símbolo de w en una posición específica.

Podemos entonces trabajar con funciones de frecuencias relativas en lugar de palabras.

- La entrada del problema no va a ser w sino que fr_w.

Decimos que $f: \Sigma \to (0,1)$ es una **función de frecuencias relativas** para Σ si se cumple que:

$$\sum_{a \in \Sigma} f(a) = 1$$

Dada una función f de frecuencias relaticas para Σ y una Σ-codificación τ, el **largo promedio de τ para f** se define como:

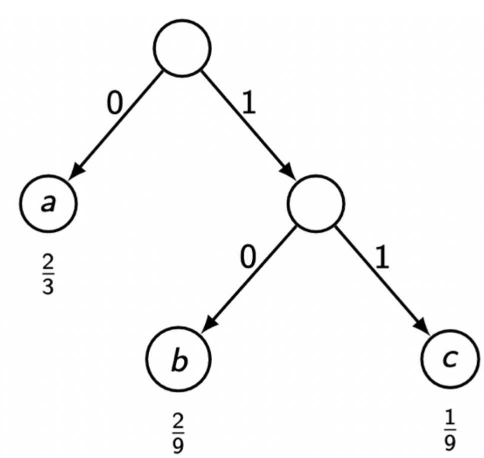
$$\mathrm{lp}_f(\tau) = \sum_{a \in \Sigma} f(a) \cdot |\tau(a)|$$

La entrada del problema es entonces una función f de frecuencias relativas para Σ, y la función objetivo a minimizar es $\mathrm{lp}_f(x)$.

Codificaciones como árboles

Si una Σ-codificación τ es libre de prefijos, entonces el árbol que la representa satisface las siguientes propiedades.

- Cada hoja tiene como etiqueta un elemento de Σ, y estos son los únicos nodos con etiquetas.
- Cada símbolo de Σ es usado exactamente una vez como etiqueta.
- Cada arco tiene etiqueta 0 o 1.
- Si una hoja tiene etiqueta e y las etiquetas de los arcos del camino desde la raíz hasta esta hoja forman una palabra $w \in \{0,1\}^*$, entonces $\tau(e) = w$.



Calculando el mínimo de $\mathbf{lp}_f(x)$

- Función objetivo a minimizar: $\mathrm{lp}_f(x)$.
- Función selección: elige los dos símbolos de Σ con menor frecuencia relativa, los coloca como hermanos en el árbol binario que representa la Σ-codificación óptima, y continua la construcción con el resto de los símbolos de Σ.

El algoritmo de Huffman

En el siguiente algoritmo representamos a las funciones como conjuntos de pares ordenados, y suponemos que f es una función de frecuencias relativas que al menos tiene dos elementos en el dominio.

CalcularCodificaciónHuffman(f)
    Sea Σ el dominio de la función f
    if $\Sigma = \{a, b\}$ **then return** $\{(a, 0), (b, 1)\}$
    else
        Sean $a, b \in \Sigma$ tales que $a \neq b$, $f(a) \leq f(b)$ y
                $f(b) \leq f(e)$ para todo $e \in (\Sigma \setminus \{a, b\})$
        Sea c un símbolo que **no** aparece en Σ
        $g := (f \setminus \{(a, f(a)), (b, f(b))\}) \cup \{(c, f(a) + f(b))\}$
        $\tau^\star :=$ **CalcularCodificaciónHuffman**(g)
        $w := \tau^\star(c)$
        $\tau := (\tau^\star \setminus \{(c, w)\}) \cup \{(a, w0), (b, w1)\}$
        return τ

Teorema

Si f es una codificación de frecuencias relativas, Σ es el dominio de f y $\tau =$**CalcularCodificaciónHuffman**(f)**,** entonces τ es una Σ-codificación libre de prefijos que minimiza la función $\mathrm{lp}_f(x)$.

<u>Demostración:</u>

Vamos a realizar la demostración por inducción en $|\Sigma|$.

Si $|\Sigma| = 2$, entonces la propiedad se cumple trivialmente

Suponga que la propiedad se cumple para un valor $n \geq 2$, y suponga que $|\Sigma| = n + 1$.

Sean a, b, c, g, τ^* y τ definidos como en el código de la llamada **CalcularCodificaciónHuffman**(f), y sea Γ el dominio de g.

Como $|\Gamma| = n$ y $\tau^* =$ **CalcularCodificaciónHuffman**(g), por hipótesis de inducción tenemos que τ^* es una Γ-codificación libre de prefijos que minimiza la función $\mathrm{lp}_g(x)$.

Dada la definición de τ es simple verificar las siguiente propiedades:

- τ es una codificaciónn libre de prefijos.
- Para cada $e \in (\Sigma \setminus \{a, b\})$ se tiene que $\tau(e) = \tau^*(e)$.
- $|\tau(a)| = |\tau(b)| = |\tau^*(c)| + 1$.

Por contradicción suponga que τ no minimiza el valor de la función $\mathrm{lp}_f(x)$,

- Vale decir, existe una Σ=codificación τ' libre de prefijos tal que τ' minimiza la función $\mathrm{lp}_f(x)$ y $\mathrm{lp}_f(\tau') < \mathrm{lp}_f(\tau)$.

Por la definición de a, b y los ejercicios anteriores podemos suponer que existe $w \in \{0,1\}^*$ tal que $\tau'(a) = w0$ y $\tau'(b) = w1$.

- Nótese que $w \neq \varepsilon$ puesto que $|\Sigma| \geq 3$.

A partir de τ' defina la siguiente Γ-codificación τ'':

$$\tau'' = \big(\tau' \setminus \{(a, \tau'(a)), (b, \tau'(b))\}\big) \cup \{(c, w)\}$$

Tenemos que τ'' es una Γ-codificación libre de prefijos.

La relación entre $\mathbf{lp}_{\boldsymbol{g}}(\boldsymbol{\tau}^*)$ y $\mathbf{lp}_{\boldsymbol{f}}(\boldsymbol{\tau})$

$$\mathrm{lp}_g(\tau^*) = \sum_{e \in \Gamma} g(e) \cdot |\tau^*(e)|$$

$$= \left(\sum_{e \,\in\, (\Gamma \setminus \{\mathrm{c}\})} g(e) \cdot |\tau^*(e)| \right) + g(c) \cdot |\tau^*(c)|$$

$$= \left(\sum_{e \,\in\, (\Sigma \setminus \{\mathrm{a,b}\})} f(e) \cdot |\tau(e)| \right) + \big(f(a) + f(b)\big) \cdot |\tau^*(c)|$$

$$= \left(\sum_{e \,\in\, (\Sigma \setminus \{\mathrm{a,b}\})} f(e) \cdot |\tau(e)| \right) + f(a) \cdot |\tau^*(c)| + f(b) \cdot |\tau^*(c)|$$

$$= \left(\sum_{e \,\in\, (\Sigma \setminus \{\mathrm{a,b}\})} f(e) \cdot |\tau(e)| \right) + f(a) \cdot (|\tau(a)| - 1) + f(b) \cdot (|\tau(b)| - 1)$$

$$= \left(\sum_{e \,\in\, (\Sigma \setminus \{\mathrm{a,b}\})} f(e) \cdot |\tau(e)| \right) + (f(a) \cdot |\tau(a)|) + (f(b) \cdot |\tau(b)|) - f(a) - f(b)$$

$$= \left(\sum_{e \,\in\, \Sigma} f(e) \cdot |\tau(e)| \right) - \big(f(a) + f(b)\big)$$

$$= \mathrm{lp}_f(\tau) - \big(f(a) + f(b)\big)$$

La relación entre $\mathbf{lp}_{\boldsymbol{g}}(\boldsymbol{\tau}'')$ y $\mathbf{lp}_{\boldsymbol{f}}(\boldsymbol{\tau}')$

$$\mathrm{lp}_g(\tau'') = \sum_{e \in \Gamma} g(e) \cdot |\tau''(e)|$$

$$= \left(\sum_{e \,\in\, (\Gamma \setminus \{\mathrm{c}\})} g(e) \cdot |\tau''(e)| \right) + g(c) \cdot |\tau''(c)|$$

$$= \left(\sum_{e \,\in\, (\Sigma \setminus \{\mathrm{a,b}\})} f(e) \cdot |\tau'(e)| \right) + \big(f(a) + f(b)\big) \cdot |\tau''(c)|$$

$$= \left(\sum_{e \,\in\, (\Sigma \setminus \{\mathrm{a,b}\})} f(e) \cdot |\tau'(e)| \right) + f(a) \cdot |\tau''(c)| + f(b) \cdot |\tau''(c)|$$

$$= \left(\sum_{e \,\in\, (\Sigma \setminus \{\mathrm{a,b}\})} f(e) \cdot |\tau'(e)| \right) + f(a) \cdot (|\tau'(a)| - 1) + f(b) \cdot (|\tau'(b)| - 1)$$

$$= \left(\sum_{e \,\in\, (\Sigma \setminus \{\mathrm{a,b}\})} f(e) \cdot |\tau'(e)| \right) + (f(a) \cdot |\tau'(a)|) + (f(b) \cdot |\tau'(b)|) - f(a) - f(b)$$

$$= \left(\sum_{e \,\in\, \Sigma} f(e) \cdot |\tau'(e)| \right) - \big(f(a) + f(b)\big)$$

$$= \mathrm{lp}_f(\tau') - \big(f(a) + f(b)\big)$$

Tenemos entonces que

$$\begin{aligned} \mathrm{lp}_g(\tau'') &= \mathrm{lp}_f(\tau') - \big(f(a) + f(b)\big) \\ &< \mathrm{lp}_f(\tau) - \big(f(a) + f(b)\big) \\ &= \mathrm{lp}_g(\tau^*) \end{aligned}$$

Concluimos entonces que τ^* no minimiza la función $\mathrm{lp}_g(x)$, lo cual contradice la hipótsis de inducción.

La siguiente función calcula la codificación de Huffman teniendo como entrada una palabra w:

```
CalcularCodificaciónHuffman(w)
    if w = ε then return ∅
    else
        Σ := conjunto de símbolos mencionados en w
        if Σ = {a} then return {(a, 0)}
        else return CalcularCodificaciónHuffman(fr_w)
```

¿Cuál es la complejidad de **CalcularCodificaciónHuffman**$(\boldsymbol{f})$?

R: $\mathcal{O}(n \cdot \log n)$ (considerando que $|f| \in \Theta(|\Sigma|)$).

Transformada rápida de Fourier

Representación de un polinomio

Sea $p(x)$ un **polinomio no nulo de coeficientes racionales**.

La representación canónica de $p(x)$ es:

$$p(x) = \sum_{i=0}^{n-1} a_i x^i$$

donde $n \geq 1$, $a_{n-1} \neq 0$ y el grado de $p(x)$ es $n-1$.

- Utilizamos el grado $n-1$ para dar énfasis a que estos polinomios poseen n coeficientes.
- Si bien trabajaremos con polinomios de coeficientes racionales, vamos a evaluarlos usando números reales y complejos.

Representamos $p(x)$ a través de una **tupla** $(a_0, \dots, a_{n-1})$ de largo n.

- También podemos representar $p(x)$ como una tupla $(a_0, \dots, a_{n-1}, 0, \dots, 0)$ de largo $m > n$ donde cada término x^i tiene coeficiente 0 si $i \geq n$.

Suma de polinomios:

La **suma** de dos polinomios $(a_0, \dots, a_{n-1})$ y $(b_0, \dots, b_{n-1})$ es un polinomio $(c_0, \dots, c_{n-1})$ tal que:

$$c_i = a_i + b_i \qquad \forall i \in \{0, \dots, n_1\}$$

Consideramos a la **suma y multiplicación de números en** $\mathbb{C}$ como las operaciones básicas a contar.

¿Cuál es la **complejidad** de este algoritmo? **R**: $\mathcal{O}(n)$

Multiplicación de polinomios:

La multiplicación de dos polinomios $(a_0, \dots, a_{n-1})$ y $(b_0, \dots, b_{n-1})$ es un polinomio $(c_0, \dots, c_{n-1})$ tal que:

$$c_i = \sum_{k,l \in \{0,\dots,n-1\} \,:\, k+l=i} a_k \cdot b_l \qquad \forall i \in \{0, \dots, 2n-2\}$$

¿Cuál es la **complejidad** de realizar esta operación? **R**: $\mathcal{O}(n^2)$

Una representación alternativa de un polinomio

Un polinomio $p(x)$ de grado $n-1$ se puede representar de manera única a través de **un conjunto de n pares de puntos-valores** (así como una parábola puede representarse con 3 puntos en $\mathbb{R}^2$):

$$p(x) \mapsto \{(v_0, p(v_0)), (v_1, p(v_1)), \dots, (v_{n-1}, p(v_{n-1}))\},$$

suponiendo que $v_i \neq v_j$ para todo $i \neq j$.

Un polinomio $p(x)$ de grado $n-1$ también se puede representar de manera única a través de un conjunto de n pares de puntos-valores con $m > n$ elementos:

$$p(x) \mapsto \{(v_0, p(v_0)), \dots, (v_{n-1}, p(v_{n-1})), (v_n, p(v_n)), \dots, (v_{m-1}, p(v_{m-1}))\},$$

suponiendo que $v_i \neq v_j$ para todo $i \neq j$.

¿Por qué es útil la representación basada en puntos-valores?

Sean $p(x)$ y $q(x)$ dos polinomios de grado $n-1$ representados por:

$$p(x) \mapsto \{(v_0, p(v_0)), \dots, (v_{n-1}, p(v_{n-1}))\}$$
$$q(x) \mapsto \{(v_0, q(v_0)), \dots, (v_{n-1}, q(v_{n-1}))\}$$

¿Cuál es la representación de $r(x) = p(x) + q(x)$?

R: $r(x) \mapsto \{(v_0, p(v_0) + q(v_0)), \dots, (v_{n-1}, p(v_{n-1}) + q(v_{n-1}))\}$
¿Cómo lo hacemos para $s(x) = p(x) \cdot q(x)$?

Suponga que se agrega n puntos a las representaciones de $p(x)$ y $q(x)$:

$$\{(v_0, p(v_0)), \dots, (v_{n-1}, p(v_{n-1})), (v_n, p(v_n)), \dots, (v_{2n-1}, p(v_{2n-1}))\}$$
$$\{(v_0, q(v_0)), \dots, (v_{n-1}, q(v_{n-1})), (v_n, q(v_n)), \dots, (v_{2n-1}, q(v_{2n-1}))\}$$

El polinomio $s(x) = p(x) \cdot q(x)$ es representado por:

$$\{(v_0, p(v_0) \cdot q(v_0)), \dots, (v_{2n-1}, p(v_{2n-1}) \cdot q(v_{2n-1}))\}$$

Podemos sumar y multiplicar polinomios en tiempo $\mathcal{O}(n)$ si están representados por partes de puntos-valores (y por los mismos puntos).

De la representación puntos-valores a la canónica

Sea $p(x)$ un polinomio de grado $n - 1$ dado por una representación punto-valores:

$$\{(v_0, p(v_0)), \dots, (v_{n-1}, p(v_{n-1})), (v_n, p(v_n)), \dots, (v_{m-1}, p(v_{m-1}))\}$$

donde $m \geq n$.

Podemos pasar a la representación canónica de $p(x)$ utilizando la **fórmula de Lagrange**:

$$p(x) = \sum_{i=0}^{m-1} p(v_i) \cdot \left(\prod_{j \in \{0,\dots,m-1\} \,:\, j \neq i} \frac{x - v_j}{v_i - v_j} \right)$$

La solución: la transformada rápida de Fourier

La transformada rápida de Fourier nos va a permitir entonces calcular la multiplicación de dos polinomios de grado $n-1$ en tiempo $\mathcal{O}(n \cdot \log_2(n))$.

- La idea clave es cómo elegir los puntos $v_0, \ldots, v_{2n-1}$ cuando se calcula la representación como punto-valores de un polinomio de grado $n-1$.

Los **números complejos** y las **raíces de la unidad** juegan un papel fundamental en la definición de la transformada rápida de Fourier.

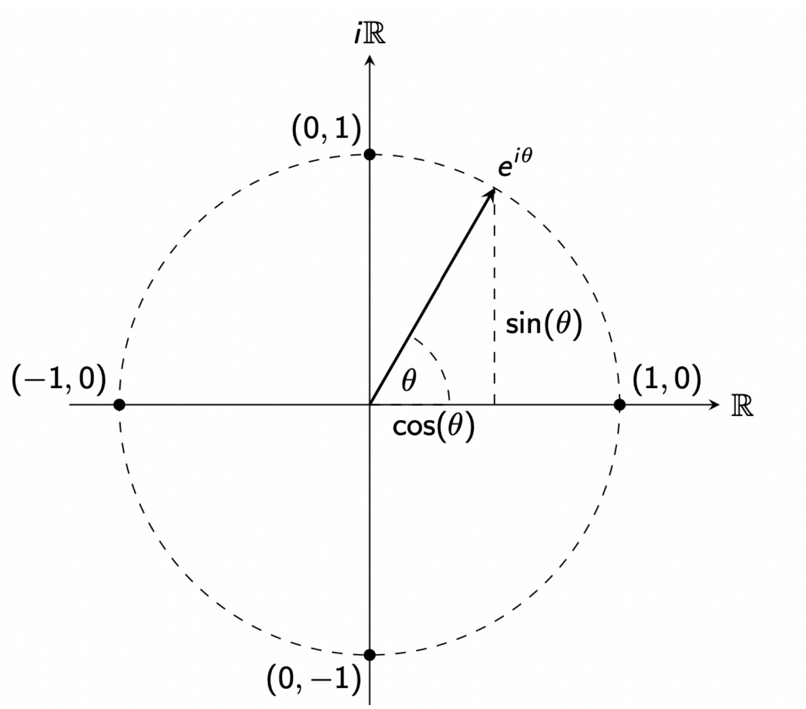
Teorema

Para todo número real x:

$$e^{ix} = \cos(x) + i \cdot \sin(x)$$

Podemos representar entonces a $e^{i\theta}$ como un vectos $(\cos(\theta), \sin(\theta))$ en el plano complejo.

- $e^{i\theta}$ en un vector unitario: $\|e^{i\theta}\| = \cos^2(\theta) + \sin^2(\theta) = 1$.



Dado $n \geq 1$, queremos encontrar las n raíces del polinomio $p(x) = x^n - 1$.

- Sabemos que este polinomio tiene n raíces en los números complejos.
- Llamamos a estos elementos las **n-raíces de la unidad**.

El componente básico para definir las n-raíces de la unidad:

$$\omega_n = e^{\frac{2\pi i}{n}}$$

Las n-raíces de la unidad son $\omega_n^0,\ \omega_n^1,\ \omega_n^2, \dots,\ \omega_n^{n-1}$.

Si $k \in \{0, \dots, n-1\}$ tenemos que:

$$\begin{aligned}
(\omega_n^k)^n &= \left(\left(e^{\frac{2\pi i}{n}}\right)^k\right)^n \\
&= \left(\left(e^{\frac{2\pi i}{n}}\right)^n\right)^k \\
&= \left(e^{2\pi i}\right)^k \\
&= (\cos(2\pi) + i \cdot \sin(2\pi))^k \\
&= 1^k \\
&= 1
\end{aligned}$$

Además, si $0 \leq k \leq l \leq n-1$, entonces:

$$\begin{aligned}
\omega_n^k = \omega_n^l \quad &\Rightarrow \quad \left(e^{\frac{2\pi i}{n}}\right)^k = \left(e^{\frac{2\pi i}{n}}\right)^l \\
&\Rightarrow \quad \left(e^{\frac{2\pi i}{n}}\right)^{l-k} = 1 \\
&\Rightarrow \quad \left(e^{\frac{2\pi (l-k) i}{n}}\right) = 1 \\
&\Rightarrow \quad \cos\left(\frac{2\pi(l-k)}{n}\right) + i \cdot \sin\left(\frac{2\pi(l-k)}{n}\right) = 1 \\
&\Rightarrow \quad \cos\left(\frac{2\pi(l-k)}{n}\right) = 1 \\
&\Rightarrow \quad \frac{l-k}{n} = 0 \\
&\Rightarrow \quad l = k
\end{aligned}$$

Puesto que:
$0 \leq \frac{l-k}{n} \leq \frac{n-1}{n}$

Por lo tanto, $\omega_n^0, \dots, \omega_n^{n-1}$ son elementos distintos.

Ejemplo:

¿Cuáles son las raíces del polinomio $x^4 - 1$?

- Considerando $\omega_4 = e^{\frac{2\pi i}{4}} = e^{\frac{\pi}{2}i}$, tenemos que las 4-raíces de la unidad son:

$$\omega_4^0 = 1$$
$$\omega_4^1 = e^{\frac{\pi}{2}i} = \cos\left(\frac{\pi}{2}\right) + i \cdot \sin\left(\frac{\pi}{2}\right) = i$$
$$\omega_4^2 = \left(e^{\frac{\pi}{2}i}\right)^2 = e^{\pi i} = \cos(\pi) + i \cdot \sin(\pi) = -1$$
$$\omega_4^3 = \left(e^{\frac{\pi}{2}i}\right)^3 = e^{\frac{3\pi}{2}i} = \cos\left(\frac{3\pi}{2}\right) + i \cdot \sin\left(\frac{3\pi}{2}\right) = -i$$

La transformada discreta de Fourier

Definición

Sea $n \geq 2$ y un polinomio $p(x) = \sum_{k=0}^{n-1} a_k x^k$.

La **transformada discreta de Fourier (DFT)** de $p(x)$ se define como:

$$[p(\omega_n^0), p(\omega_n^1), \dots, p(\omega_n^{n-1})]$$

¿Cómo podemos calcular **DFT** de manera eficiente?

Calcular DFT

Recordemos que vamos a representar $p(x)$ a través del vector $\bar{a} = (a_0, \dots, a_{n-1})$.

El problema a resolver es calcular de manera eficiente la DFT de $p(x)$, la cual denotamos como **DFT**$(\bar{a})$.

- Definimos $y_k = p(\omega_n^k)$ para cada $k \in \{0, \dots, n-1\}$, de manera tal que queremos calcular **DFT**$(\bar{a})$.

Calcular DFT de manera eficiente

Suponemos que $n = 2^t$para $t \in \mathbb{N}$, calculamos **DFT**$(\bar{a})$ realizando los siguientes pasos:

1. Calcular **DFT**$(\bar{a}_0)$ y **DFT**$(\bar{a}_1)$ para dos vectores $\bar{a}_0$ y $\bar{a}_1$ de largo $\frac{n}{2}$.
2. Combinar **DFT**$(\bar{a}_0)$ y **DFT**$(\bar{a}_1)$ para obtener **DFT**$(\bar{a})$.

Es fundamental que el paso 2 sea realizado en tiempo $\mathcal{O}(n)$.

Tenemos que:

$$\begin{aligned} p(x) &= \sum_{k=0}^{n-1} a_k x^k \\ &= \sum_{k=0}^{\frac{n}{2}-1} a_{2k} x^{2k} + \sum_{k=0}^{\frac{n}{2}-1} a_{2k+1} x^{2k+1} \\ &= \sum_{k=0}^{\frac{n}{2}-1} a_{2k} x^{2k} + x \cdot \sum_{k=0}^{\frac{n}{2}-1} a_{2k+1} x^{2k} \end{aligned}$$

Definimos los polinomios:

$$\begin{aligned} q(z) &= \sum_{k=0}^{\frac{n}{2}-1} a_{2k} z^k \\ r(z) &= \sum_{k=0}^{\frac{n}{2}-1} a_{2k+1} z^k \end{aligned}$$

Tenemos que:

$$p(x) = q(x^2) + x \cdot r(x^2)$$

Para calcular $[p(\omega_n^0), p(\omega_n^1), \dots, p(\omega_n^{n-1})]$, tenemos entonces que calcular:

$$[q((\omega_n^0)^2), q((\omega_n^1)^2), \dots, q((\omega_n^{n-1})^2)]$$
$$[r((\omega_n^0)^2), r((\omega_n^1)^2), \dots, r((\omega_n^{n-1})^2)]$$

Pero si $k \in \left\{0, \dots, \frac{n}{2} - 1\right\}$, entonces tenemos que:

$$\begin{aligned}\left(\omega_n^{\frac{n}{2}+k}\right)^2 &= \omega_n^{n+2k} \\ &= \omega_n^n \cdot \omega_n^{2k} \\ &= \left(e^{\frac{2\pi i}{n}}\right)^n \cdot (\omega_n^k)^2 \\ &= e^{2\pi i} \cdot (\omega_n^k)^2 \\ &= 1 \cdot (\omega_n^k)^2 \\ &= (\omega_n^k)^2\end{aligned}$$

¿ Si $\omega_n^0, \omega_n^1, \dots, \omega_n^{n-1}$ son las raíces de la unidad, quiénes son $(\omega_n^0)^2, (\omega_n^1)^2, \dots, \left(\omega_n^{\frac{n}{2}-1}\right)^2$?

Lema

Si $n \geq 2$ es par, entonces $(\omega_n^0)^2, (\omega_n^1)^2, \dots, \left(\omega_n^{\frac{n}{2}-1}\right)^2$ son las $\frac{n}{2}$ -raíces de la unidad (vale decir, son las raíces del polinomio $x^{\frac{n}{2}} - 1$).

Demostración

Primero tenemos que demostrar la regla de simplificación $\omega_{m \cdot l}^{k \cdot l} = \omega_m^k$ para $l > 0$:

$$\omega_{m \cdot l}^{k \cdot l} = \left(e^{\frac{2\pi i}{m \cdot l}}\right)^{k \cdot l} = \left(e^{\frac{2\pi i \cdot l}{m \cdot l}}\right)^k = \left(e^{\frac{2\pi i}{m}}\right)^k = \omega_m^k$$

Dado que $k \in \left\{0, \dots, \frac{n}{2} - 1\right\}$, se tiene que:

$$(\omega_n^k) = \omega_n^{k \cdot 2} = \omega_{\frac{n}{2} \cdot 2}^{k \cdot 2} = \omega_{\frac{n}{2}}^k \qquad \text{(por la regla de simplificación)}$$

Por lo tanto, $(\omega_n^0)^2, (\omega_n^1)^2, \dots, \left(\omega_n^{\frac{n}{2}-1}\right)^2$ son las $\frac{n}{2}$ – raíces de la unidad.

- Puesto que $(\omega_n^0)^2, (\omega_n^1)^2, \dots, \left(\omega_n^{\frac{n}{2}-1}\right)^2 = \omega_{\frac{n}{2}}^0, \omega_{\frac{n}{2}}^1, \dots, \omega_{\frac{n}{2}}^{\frac{n}{2}-1}$.

Recuerde que $\bar{a} = (a_0, \dots, a_{n-1})$, y defina:

$$\begin{aligned}\bar{a}_0 &= (a_0, a_2, \dots, a_{n-2})\\ \bar{a}_1 &= (a_1, a_3, \dots, a_{n-1})\end{aligned}$$

De los resultados anteriores concluimos que para calcular **DFT**$(\bar{a})$, primero tenemos que calcular **DFT**$(\bar{a})_0$ y **DFT**$(\bar{a}_1)$.

¿Cómo se construye **DFT**$(\bar{a})$ a partir de **DFT**$(\bar{a}_0)$ y **DFT**$(\bar{a}_1)$?

Sea:

$$\begin{aligned}\mathbf{DFT}(\bar{a}_0) &= \left[u_0, u_1, \dots, u_{\frac{n}{2}-1}\right]\\ \mathbf{DFT}(\bar{a}_1) &= \left[v_0, v_1, \dots, v_{\frac{n}{2}-1}\right]\end{aligned}$$

Para $k \in \left\{0, \dots, \frac{n}{2} - 1\right\}$ tenemos que:

$$\begin{aligned}y_k &= p(\omega_n^k)\\ &= q((\omega_n^k)^2) + \omega_n^k \cdot r((\omega_n^k)^2)\\ &= q\left(\omega_{\frac{n}{2}}^k\right) + \omega_n^k \cdot r\left(\omega_{\frac{n}{2}}^k\right)\\ &= u_k + \omega_n^k \cdot v_k\end{aligned}$$

Además, para $k \in \{0, \dots, \frac{n}{2} - 1\}$ tenemos que:

$$
\begin{aligned}
y_{\frac{n}{2}+k} &= p\left(\omega_n^{\frac{n}{2}+k}\right) \\
&= q\left(\left(\omega_n^{\frac{n}{2}+k}\right)^2\right) + \omega_n^{\frac{n}{2}+k} \cdot r\left(\left(\omega_n^{\frac{n}{2}+k}\right)^2\right) \\
&= q(\omega_n^{n+2k}) + \omega_n^{\frac{n}{2}+k} \cdot r(\omega_n^{n+2k}) \\
&= q(\omega_n^n \cdot \omega_n^{2k}) + \left(e^{\frac{2\pi i}{n}}\right)^{\frac{n}{2}} \cdot \omega_n^k \cdot r(\omega_n^n \cdot \omega_n^{2k}) \\
&= q(1 \cdot \omega_n^{2k}) + e^{\pi i} \cdot \omega_n^k \cdot r(1 \cdot \omega_n^{2k}) \\
&= q(\omega_n^{2k}) - \omega_n^k \cdot r(\omega_n^{2k}) \\
&= q\left(\omega_{\frac{n}{2}}^k\right) - \omega_n^k \cdot r\left(\omega_{\frac{n}{2}}^k\right) \\
&= u_k - \omega_n^k \cdot v_k
\end{aligned}
$$

Resumiendo, para $k \in \left\{0, \dots, \frac{n}{2} - 1\right\}$ tenemos que:

$$
\begin{aligned}
y_k &= u_k + \omega_n^k \cdot v_k \\
y_{\frac{n}{2}+k} &= u_k - \omega_n^k \cdot v_k
\end{aligned}
$$

Para tener un algoritmo recursivo para calcular **DFT** sólo nos falta el **caso base**.

- Consideramos $n = 2$ y un polinomio $p(x) = a_0 + a_1 x$.

Tenemos que:

$$
\begin{aligned}
p(\omega_2^0) = a_0 + a_1 \cdot \omega_2^0 = a_0 + a_1 \\
p(\omega_2^1) = a_0 + a_1 \cdot \omega_2^1 = a_0 - a_1
\end{aligned}
$$

Un algoritmo recursivo eficiente para **DFT**

- La entrada del algoritmo es un polinomio $p(x) = \sum_{k=0}^{n-1} a_k x^k$.
  - Este polinomio es representado por el vector $\bar{a} = (a_0, \dots, a_{n-1})$.
- Supongamos además que $n \geq 2$ y n es una potencia de 2.
- El algoritmo se llama la **transformada rápida de Fourier (FFT)**.
  - Fue propuesto por *Cooley & Tukey* (1965).

```
FFT(a_0, ..., a_{n-1})
    if n = 2 then
        y_0 = a_0 + a_1
        y_1 = a_0 - a_1
        return [y_0, y_1]
    else
        [u_0, ..., u_{n/2-1}] := FFT(a_0, ..., a_{n-2})
        [v_0, ..., v_{n/2-1}] := FFT(a_1, ..., a_{n-1})
        ω_n := e^{2πi/n}
        α := 1
        for k := 0 to n/2 - 1 do
            y_k := u_k + α · v_k
            y_{n/2+k} := u_k - α · v_k
            α := α · ω_n
        return [y_0, ..., y_{n-1}]
```

Representación en coeficientes. Ejemplo:
$1 + 2x = [1, 2]$

Las 2 raíces de la unidad son 1 y -1.
Ejemplo: $p(x) = 1 + 2x$

$$p(1) = 1 + 2$$
$$p(-1) = 1 - 2$$

Recordar que si:
$$\omega_n = e^{\frac{2\pi i}{n}}$$
Las n-raíces de la unidad serán:
$\omega_n^0, \omega_n^1, \omega_n^2, \dots, \omega_n^{n-1}$

Si quisiéramos realizar el caso base para un polinomio de grado 2, sería:

$$p(x) = 1 + 2x + 3x^2$$
$$\omega_3 = e^{\frac{2\pi i}{3}} \Longrightarrow \omega_3^k = e^{\frac{2\pi i \cdot k}{3}}$$

$$p(\omega_3^0) = 1 + 2(\omega_3^0) + 3(\omega_3^0)^2 = 1 + 2(1) + 3(1)$$
$$p(\omega_3^1) = 1 + 2(\omega_3^1) + 3(\omega_3^1)^2 = 1 + 2\left(e^{\frac{2\pi i}{3}}\right) + 3\left(e^{\frac{4\pi i}{3}}\right)$$
$$p(\omega_3^2) = 1 + 2(\omega_3^2) + 3(\omega_3^2)^2 = 1 + 2\left(e^{\frac{4\pi i}{3}}\right) + 3\left(e^{\frac{8\pi i}{3}}\right)$$

Algoritmos Aleatorizados

Algoritmos aleatorizados:

- **Monte Carlo**: El algoritmo siempre entrega un resultado, pero hay una probabilidad de que sea **incorrecto**.
- **Las Vegas**: Si el algoritmo entrega un resultado es correcto, pero hay una probabilidad de **no entregue resultado**.

Equivalencia de Polinomios

Suponga que:

$$p(x) = \sum_{i=1}^{k} \prod_{j=1}^{r_i} (a_{i,j}x + b_{i,j})$$
$$q(x) = \sum_{i=1}^{l} \prod_{j=1}^{s_i} (c_{i,j}x + d_{i,j})$$

```
EquivPolAleatorizado(p(x), q(x))
    K := 1 + max{r_1, ..., r_k, s_1, ..., s_ℓ}
    escoja al azar y con distribución uniforme un elemento a
                del conjunto de números naturales {1, ..., 100 · K}
    if p(a) = q(a) then return sí
    else return no
```

K = Máximo exponente entre ambos polinomios $+1$.

Solo hacemos $\mathcal{O}(n)$ operaciones, donde $n = |p(x)| + |q(x)|$, ya que necesita calcular $p(a)$ y $q(a)$. **Pero me puedo equivocar**.

¿Probabilidad de error?

- Si los polinomios $p(x)$ y $q(x)$ son equivalentes, entonces el algoritmo responde **SI** sin cometer errores.

- Si los polinomios $p(x)$ y $q(x)$ no son equivalentes, el algoritmo puede responder SI al sacar al azar un elemento $a \in \{1, \dots, 100 \cdot K\}$ tal que $p(a) = q(a)$.

  Esto significa que a es una **raíz** del polinomio

$$r(x) = p(x) - q(x)$$

Sabemos que $r(x)$ NO es el polinomio nulo y que es de grado a lo más K.

- Por lo tanto $r(x)$ tiene a lo más K raíces en $\mathbb{Q}$.

Concluimos que:

$$\Pr(a \text{ sea raíz de } r(x)) \leq \frac{K}{100 \cdot K} = \frac{1}{100}$$

¿Cómo mejoramos esta probabilidad? → Repetimos el algoritmo:

EquivPolAleatorizado$(p(x),\ q(x))$
    $K := 1 + \max\{r_1, \ldots, r_k, s_1, \ldots, s_\ell\}$
    $A := \{1, \ldots, 100 \cdot K\}$
    $total := 0$
    for $i := 1$ **to** 10 **do**
        escoja al azar y con distribución uniforme un elemento a en A
        if $p(a) = q(a)$ **then** $total = total + 1$
    if $total = 10$ **then return** sí
    return no

Ahora la probabilidad será de: $\boxed{\frac{1}{100^{10}}}$

Definición:

Un **polinomio en varias variables** es una expresión de la forma:

$$p(x_1, \ldots, x_n) = \sum_{i=1}^{l} \prod_{j=1}^{m_i} \left(\sum_{k=1}^{n} a_{i,j,k} x_k + b_{i,j} \right)$$

donde cada $a_{i,j,k} \in \mathbb{Q}$ y cada $b_{i,j} \in \mathbb{Q}$.

Lema de Schwartz-Zippel

Sea $p(x_1, \ldots, x_n)$ un polinomio no nulo de grado k, y sea A un subconjunto finito y no vacío de $\mathbb{Q}$. Si $a_1, \ldots, a_n$ son elegidos de manera uniforme e independientemente desde A, entonces:

$$\Pr(p(a_1, \ldots, a_n) = 0) \leq \frac{k}{|A|}$$

Suponga que la entrada del algoritmo aleatorizado está dada por los siguientes polinomios:

$$p(x_1, \dots, x_n) = \sum_{i=1}^{l} \prod_{j=1}^{r_i} \left(\sum_{k=1}^{n} a_{i,j,k} x_k + b_{i,j} \right)$$
$$q(x_1, \dots, x_n) = \sum_{i=1}^{m} \prod_{j=1}^{s_i} \left(\sum_{k=1}^{n} c_{i,j,k} x_k + d_{i,j} \right)$$

EquivPolAleatorizado$(p(x_1, \dots, x_n),\ q(x_1, \dots, x_n))$
    $K := 1 + \max\{r_1, \dots, r_\ell, s_1, \dots, s_m\}$
    $A := \{1, \dots, 100 \cdot K\}$
    sea $a_1, \dots, a_n$ una secuencia de números elegidos de
        manera uniforme e independiente desde A
    if $p(a_1, \dots, a_n) = q(a_1, \dots, a_n)$ **then return** sí
    else return no

Utilizando el lema de Schwartz-Zippel

Vamos a calcular la **probabilidad de error** del algoritmo:

- Si los polinomios $p(x_1, \dots, x_n)$ y $q(x_1, \dots, x_n)$ son equivalentes, entonces el algoritmo responde **SI** sin cometer error.

- Si los polinomios $p(x_1, \dots, x_n)$ y $q(x_1, \dots, x_n)$ NO son equivalentes, el algoritmo puede responder SI al escoger una secuencia de números $a_1, \dots, a_n$ desde A tales que $p(a_1, \dots, a_n) = q(a_1, \dots, a_n)$.
    - Donde $A = \{1, \dots, 100 \cdot K\}$.

  Esto significa que $(a_1, \dots, a_n)$ es una raíz del polinomio
  $$r(x_1, \dots, x_n) = p(x_1, \dots, x_n) - q(x_1, \dots, x_n)$$

  $r(x_1, \dots, x_n)$ NO es el polinomio nulo y es de grado t con $t < K$.
    - Dado que $K = 1 + \max\{r_1, \dots, r_l, s_1, \dots, s_m\}$.

Utilizando el lema de Schwartz-Zippel obtenemos:

$$\Pr(r(a_1, \dots, a_n) = 0) \leq \frac{t}{|A|} < \frac{K}{|A|} = \frac{K}{100 \cdot K} = \frac{1}{100}$$

Si es igual a $c \in (\mathbb{Q} \backslash \{0\})$, entonces es una constante, y el grado del polinomio debe ser 0.

Demostración del lema de Schwartz-Zippel

Si $p(x_1, \ldots x_{n+1})$ en su forma canónica es igual a $c \in (\mathbb{Q} \setminus \{0\})$, entonces el lema se cumple trivialmente ya que:

$$\Pr(p(a_1, \ldots a_{n+1}) = 0) = 0$$

Suponemos entonces que $p(x_1, \ldots x_{n+1})$ en su forma canónica no es igual a $c \in \mathbb{Q}$.

- Puesto que además sabemos que $p(x_1, \ldots, x_{n+1})$ NO es nulo.

Por el lema de Schwartz

Tenemos que $p(x_1, \ldots, x_{n+1})$ en su forma canónica contiene un monomio de la forma:

$$c x_1^{l_1} x_2^{l_2} \cdots x_{n+1}^{l_{n+1}}$$

Porque es polinomio de varias variables.

donde $c \neq 0$ y $l_i > 0$ para algún $i \in \{1, \ldots, n+1\}$.

Al menos un l_i debe ser $>$ 0.

Sin pérdida de generalidad suponemos que en el monomio anterior $l_1 > 0$
Tenemos que:

$$p(x_1, x_2, \ldots, x_{n+1}) = \sum_{i=0}^{k} x_1^i p_i(x_2, \ldots, x_{n+1})$$

Sacamos la variable factorizando.

donde cada $p_i(x_2, \ldots, x_{n+1})$ es un polinomio y al menos uno de ellos NO es nulo.

De lo contrario tendríamos el polinomio nulo.

Sea $l = \max\{\, i \in \{0, \ldots, k\} \mid p_i(x_2, \ldots, x_{n+1}) \text{ no es nulo } \}$

- Tenemos que $l > 0$ ya que supusimos que $l_1 > 0$.

l será el máximo exponente de la variable x_1.

Lema de Schwartz

Dado que el grado de $p(x_1, x_2, \ldots, x_{n+1})$ es k, tenemos que el grado de $p_l(x_2, \ldots, x_{n+1})$ es m con $m \leq k - l$.

p_l esta siendo multiplicado por x_1^l

Sea A un subconjunto finito y NO vacío de $\mathbb{Q}$, y sea $a_1, \ldots, a_{n+1}$ una secuencia de números elegidos de manera uniforme e independiente desde A.

Por hipótesis de inducción tenemos que:

$$\Pr(p_l(a_2, \ldots, a_{n+1}) = 0) \leq \frac{m}{|A|} \leq \frac{k-l}{|A|}$$

Lema de Schwartz

Si $p_l(a_2, \ldots, a_{n+1}) \neq 0$, entonces por definición de l tenemos que $q(x_1) = p(x_1, a_2, \ldots, a_{n+1})$ es un polinomio de grado l.
Por lo tanto:

x_1 es de grado l, y el resto son grado 0 ($a_i \in \mathbb{Q}$).

$$\Pr\left(\overbrace{p(a_1, a_2, \ldots, a_{n+1})}^{\text{grado } l} = 0 \mid p_l(a_2, \ldots, a_{n+1}) \neq 0 \right) \leq \frac{l}{|A|}$$

Recordar: $P(A) = P(A|B_1) \cdot P(B_1) + P(A|B_2) \cdot P(B_2)$

Concluimos que:

$$
\begin{aligned}
&\Pr(p(a_1, a_2, \dots, a_{n+1}) = 0) = \\
&\qquad \Pr(p(a_1, a_2, \dots, a_{n+1}) = 0 \mid p_l(a_2, \dots, a_{n+1}) = 0) \cdot \Pr(p_l(a_2, \dots, a_{n+1}) = 0) + \\
&\qquad \Pr(p(a_1, a_2 \dots, a_{n+1}) = 0 \mid p_l(a_2, \dots, a_{n+1}) \neq 0) \cdot \Pr(p_l(a_2, \dots, a_{n+1}) \neq 0) \leq \\
&\qquad \Pr(p_l(a_2, \dots, a_{n+1}) = 0) + \\
&\qquad \Pr(p(a_1, a_2 \dots, a_{n+1}) = 0 \mid p_l(a_2, \dots, a_{n+1}) \neq 0) \cdot \Pr(p_l(a_2, \dots, a_{n+1}) \neq 0) \leq
\end{aligned}
$$

$$\frac{k-l}{|A|} + \frac{l}{|A|} = \frac{k}{|A|}$$

Cálculo de la mediana

Suponga que el procedimiento **MergeSort**(L) ordena una lista L utilizando el algoritmo MergeSort

El siguiente procedimiento calcula la mediana de una lista de enteros $L[1 \dots n]$ (suponiendo que n es impar y L NO tiene elementos repetidos).

```
CalcularMediana(L[1...n])
    if n < 2001 then
        Mergesort(L)
        return L[⌈n/2⌉]
    else
        sea R una lista de ⌈n^(3/4)⌉ números enteros escogido con
            distribución uniforme y de manera independiente desde L
        Mergesort(R)
        d := R[⌊(1/2)·n^(3/4) − n^(1/2)⌋]
        u := R[⌈(1/2)·n^(3/4) + n^(1/2)⌉]
        S := ∅
        m_d := 0
        m_u := 0
        for i := 1 to n do
            if d ≤ L[i] and L[i] ≤ u then Append(S, [L[i]])
            else if L[i] < d then m_d := m_d + 1
            else m_u := m_u + 1
        if m_d ≥ ⌈n/2⌉ or m_u ≥ ⌈n/2⌉ or
                Length(S) > 4·⌊n^(3/4)⌋ then return sin_resultado
        else
            Mergesort(S)
            return S[⌈n/2⌉ − m_d]
```

$$\mathcal{O}\left(n^{\frac{3}{4}} \cdot \log\left(n^{\frac{3}{4}}\right)\right) = \mathcal{O}\left(n^{\frac{3}{4}} \cdot \frac{3}{4} \cdot \log(n)\right) = \mathcal{O}\left(n^{\frac{3}{4}} \cdot \log(n)\right) \in \mathcal{O}(n)$$

Con “no muy grande” no referimos a que el tamaño sea $\mathcal{O}(n/\log n)$, de esta manera, podemos ordenar este conjunto en $\mathcal{O}(n)$.

d y u son mis elementos que escojo de tal manera que la mediana m este entre estos 2 elementos, y el tamaño de los elementos que hay entre d y u no sea muy grande.

Me tomará $\mathcal{O}(n)$ por el tamaño de R.

Agregamos a S todos los elementos que están entre d y u.

$\frac{1}{2} \cdot n^{\frac{3}{4}}$ = La mitad de R. Entonces mi lista S tendrá tamaño $2\sqrt{n}$.

Si esto ocurre, entonces la mediana estará afuera de S, y debemos retornar $sin\_resultado$

Me tomará $\leq \mathcal{O}(n)$ por el tamaño de S.

Retornamos la mediana de S.

La llamada **CalcularMediana**(L) puede NO retornar un resultado.

- El procedimiento en este caso retorna $sin\_resultado$.

Para que **CalcularMediana** pueda ser utilizado en la práctica la probabilidad que no entregue un resultado debe ser baja.

Sea $L[1 \dots n]$ una lista de números enteros tal que $n \geq 2001$, n es impar y la mediana de L es m.

Defina las siguientes variables aleatorias:

$$Y_1 = \left|\left\{i \in \left\{1, \dots, \left\lceil n^{\frac{3}{4}} \right\rceil\right\} \mid R[i] \leq m\right\}\right|$$

Cantidad de números de R que son **menores** o iguales a la mediana.

$$Y_2 = \left|\left\{i \in \left\{1, \dots, \left\lceil n^{\frac{3}{4}} \right\rceil\right\} \mid R[i] \geq m\right\}\right|$$

Cantidad de números de R que son **mayores** o iguales a la mediana.

Estas son variables aleatorias dado que R es construido escogiendo elementos de L con distribución uniforme (y de manera independiente).

Lema

CalcularMediana(L) retorna $sin\_resultado$ si y sólo si alguna de las siguientes condiciones se cumple:

1. $Y_1 < \left\lfloor \frac{1}{2} \cdot n^{\frac{3}{4}} - n^{\frac{1}{2}} \right\rfloor$ — La mediana está antes de d, y por lo tanto NO estará en S
2. $Y_2 \leq \left\lceil n^{\frac{3}{4}} \right\rceil - \left\lceil \frac{1}{2} \cdot n^{\frac{3}{4}} + n^{\frac{1}{2}} \right\rceil$ — La mediana está después de u, y por lo tanto NO estará en S
3. **Length**$(S) > 4 \cdot \left\lfloor n^{\frac{3}{4}} \right\rfloor$ — De lo contrario nos podría llegar a tomar más de $O(n)$.

La desigualdad de Markov

Teorema

Sea X una variable aleatoria NO negativa. Para cada $a \in \mathbb{R}^+$ se tiene que:

$$\Pr(X \geq a) \leq \frac{E(X)}{a}$$

Este resultado se conoce como la **desigualdad de Markov**.

Demostración

Suponemos que el recorrido de X es un conjunto finito $\Omega \subseteq \mathbb{R}_0^+$:

$$\begin{aligned}
E(X) &= \sum_{r \in \Omega} r \cdot \Pr(X = r) \\
&= \left(\sum_{r \in \Omega : r < a} r \cdot \Pr(X = r) \right) + \left(\sum_{s \in \Omega : s \geq a} s \cdot \Pr(X = s) \right) \\
&\geq \sum_{s \in \Omega : s \geq a} s \cdot \Pr(X = s) \\
&\geq \sum_{s \in \Omega : s \geq a} a \cdot \Pr(X = s) \\
&= a \cdot \left(\sum_{s \in \Omega : s \geq a} \Pr(X = s) \right) \\
&= a \cdot \Pr(X \geq a)
\end{aligned}$$

Concluimos que $\Pr(X \geq a) \leq \frac{E(X)}{a}$.

Teorema

El siguiente resultado se conoce como la **desigualdad de Chebyshev**:

$$\Pr(|X - E(X)| \geq a) \ \leq \ \frac{Var(X)}{a^2}$$

Demostración

$$\boxed{Var(X) = E\left[\left(X - E(X)\right)^2\right]}$$

Utilizando la desigualdad de Markov obtenemos:

$$\begin{aligned}
\Pr(|X - E(X)| \geq a) \ &= \ \Pr\left(\left(X - E(X)\right)^2 \geq a^2\right) \\
&\leq \frac{E\left(\left(X - E(X)\right)^2\right)}{a^2} \\
&= \frac{Var(X)}{a^2}
\end{aligned}$$

Lema

$$\Pr\left(Y_1 < \left\lfloor \frac{1}{2} \cdot n^{\frac{3}{4}} - n^{\frac{1}{2}} \right\rfloor\right) \leq n^{-\frac{1}{4}}$$

Demostración

Para cada $i \in \left\{1, \dots, \left\lceil n^{\frac{3}{4}} \right\rceil\right\}$, definimos una variable aleatoria X_i de la siguiente forma:

$$X_i = \begin{cases} 1 & R[i] \leq m \\ 0 & R[i] > m \end{cases} \longrightarrow \boxed{\sim \text{Bernoulli}\left(\frac{1}{2} + \frac{1}{2 \cdot n}\right)}$$

Tenemos que:

$$Y_1 = \sum_{i=1}^{\lceil n^{3/4} \rceil} X_i \longrightarrow \boxed{\sim \text{Binomial}\left(\lceil n^{3/4} \rceil, \frac{1}{2} + \frac{1}{2 \cdot n}\right)}$$

Dado que la lista L no contiene elementos repetidos tenemos que:

$$\Pr(X_i = 1) = \frac{\left\lceil \frac{n}{2} \right\rceil}{n} = \frac{\frac{n-1}{2} + 1}{n} = \boxed{\frac{1}{2} + \frac{1}{2 \cdot n}}$$

De esto de deduce que:

$$E(X_i) = \frac{1}{2} + \frac{1}{2 \cdot n}$$

$$\boxed{Varianza_{binomial(n,p)} = np(1-p)}$$

$$\begin{aligned} Var(X_i) &= \left(\frac{1}{2} + \frac{1}{2 \cdot n}\right) \cdot \left(1 - \frac{1}{2} - \frac{1}{2 \cdot n}\right) \\ &= \left(\frac{1}{2} + \frac{1}{2 \cdot n}\right) \cdot \left(\frac{1}{2} - \frac{1}{2 \cdot n}\right) \\ &= \frac{1}{4} - \frac{1}{4 \cdot n^2} \end{aligned}$$

Aquí hablamos de un único X_i, por eso $n = 1$.

Por lo tanto, tenemos que:

$$\begin{aligned} E(Y_1) &= E\left(\sum_{i=1}^{\lceil n^{3/4} \rceil} X_i\right) \\ &= \sum_{i=1}^{\lceil n^{3/4} \rceil} E(X_i) \\ &= \sum_{i=1}^{\lceil n^{3/4} \rceil} \left(\frac{1}{2} + \frac{1}{2 \cdot n}\right) \longrightarrow \boxed{\text{NO depende de } i} \\ &= \left\lceil n^{\frac{3}{4}} \right\rceil \cdot \left(\frac{1}{2} + \frac{1}{2 \cdot n}\right) \end{aligned}$$

Para $i.j \in \left\{1, \dots, \left\lceil n^{\frac{3}{4}} \right\rceil\right\}$ tal que $i \neq j$ se tiene que X_i es independiente de X_j

Concluimos entonces que:

$$\begin{aligned} Var(Y_1) &= Var\left(\sum_{i=1}^{\lceil n^{3/4} \rceil} X_i\right) \\ &= \sum_{i=1}^{\lceil n^{3/4} \rceil} Var(X_i) \\ &= \sum_{i=1}^{\lceil n^{3/4} \rceil} \left(\frac{1}{4} - \frac{1}{4 \cdot n^2}\right) \longrightarrow \boxed{\text{NO depende de } i} \\ &= \left\lceil n^{\frac{3}{4}} \right\rceil \cdot \left(\frac{1}{4} - \frac{1}{4 \cdot n^2}\right) \\ &\leq \frac{1}{4} \cdot \left\lceil n^{\frac{3}{4}} \right\rceil \end{aligned}$$

Tenemos que:

$$\begin{aligned} \Pr\left(Y_1 < \left\lfloor \frac{1}{2} \cdot n^{\frac{3}{4}} - n^{\frac{1}{2}} \right\rfloor\right) &\leq \Pr\left(Y_1 < \frac{1}{2} \cdot n^{\frac{3}{4}} - n^{\frac{1}{2}}\right) \\ &\leq \Pr\left(Y_1 < \frac{1}{2} \cdot \left\lceil n^{\frac{3}{4}} \right\rceil - n^{\frac{1}{2}}\right) \\ &\leq \Pr\left(Y_1 < \left\lceil n^{\frac{3}{4}} \right\rceil \cdot \left(\frac{1}{2} + \frac{1}{2 \cdot n}\right) - n^{\frac{1}{2}}\right) \\ &= \Pr\left(Y_1 < E(Y_1) - n^{\frac{1}{2}}\right) \\ &= \Pr\left(n^{\frac{1}{2}} < E(Y_1) - Y_1\right) \\ &\leq \Pr\left(|Y_1 - E(Y_1)| > n^{\frac{1}{2}}\right) \\ &\leq \Pr\left(|Y_1 - E(Y_1)| \geq n^{\frac{1}{2}}\right) \end{aligned}$$

Por lo tanto, utilizando la desigualdad de Chebyshev concluimos que:

$$
\begin{aligned}
\Pr\left(Y_1 < \left\lfloor \frac{1}{2} \cdot n^{\frac{3}{4}} - n^{\frac{1}{2}} \right\rfloor\right) &\leq \Pr\left(|Y_1 - E(Y_1)| \geq n^{\frac{1}{2}}\right) \\
&\leq \frac{Var(Y_1)}{n} \\
&\leq \frac{1}{4} \cdot \frac{\left\lceil n^{\frac{3}{4}} \right\rceil}{n} \\
&\leq \frac{1}{4} \cdot \frac{n^{\frac{3}{4}} + 1}{n} \\
&\leq \frac{1}{4} \cdot \frac{2 \cdot n^{\frac{3}{4}}}{n} \\
&= \frac{1}{2} \cdot n^{-\frac{1}{4}} \\
&\leq \boldsymbol{n^{-\frac{1}{4}}}
\end{aligned}
$$

Lema

$$\Pr\left(Y_2 \leq \left\lceil n^{\frac{3}{4}} \right\rceil - \left\lceil \frac{1}{2} \cdot n^{\frac{3}{4}} + n^{\frac{1}{2}} \right\rceil\right) \leq \quad n^{-\frac{1}{4}}$$

Lema

$$\Pr\left(\mathbf{Length}(S) > 4 \cdot \left\lfloor n^{\frac{3}{4}} \right\rfloor\right) \leq \quad 4 \cdot n^{-\frac{1}{4}}$$

Demostración

Si $\mathbf{Length}(S) > 4 \cdot \left\lfloor n^{\frac{3}{4}} \right\rfloor$, entonces al menos una de las siguientes condiciones debe ser cierta:

(a) $|\{i \in \{1, \dots, \mathbf{Length}(S)\} \mid S[i] > m\}| \geq 2 \cdot \left\lfloor n^{\frac{3}{4}} \right\rfloor$ — Mediana estará a la izquierda de S.

(b) $|\{i \in \{1, \dots, \mathbf{Length}(S)\} \mid S[i] < m\}| \geq 2 \cdot \left\lfloor n^{\frac{3}{4}} \right\rfloor$ — Mediana estará a la derecha de S.

Vamos a demostrar que la probabilidad de que (a) ocurra es menor o igual a $2 \cdot n^{-\frac{1}{4}}$.

De la misma forma se demuestra que la probabilidad que (b) ocurra es menor o igual a $2 \cdot n^{-\frac{1}{4}}$.

- De esto se concluye que $\Pr\left(\mathbf{Length}(S) > 4 \cdot \left\lfloor n^{\frac{3}{4}} \right\rfloor\right) \leq 4 \cdot n^{-\frac{1}{4}}$.

Suponga que (a) es cierto, y sea l la posición de u en la lista L ordenada.

- Tenemos que $l \geq \left\lceil \frac{n}{2} \right\rceil + 2 \cdot \left\lfloor n^{\frac{3}{4}} \right\rfloor$

Dado que $u = R\left[\left\lceil \frac{1}{2} \cdot n^{\frac{3}{4}} + n^{\frac{1}{2}} \right\rceil\right]$ al menos $\left\lceil n^{\frac{3}{4}} \right\rceil - \left\lceil \frac{1}{2} \cdot n^{\frac{3}{4}} + n^{\frac{1}{2}} \right\rceil$ elementos de R deben estar en posiciones mayores o iguales a l en la lista L ordenada.

- No podemos asegurar que estos elementos están en posiciones mayores a l puesto que R puede tener elementos repetidos.

- Vamos a acotar superiormente la probabilidad de que esto ocurra para obtener una cota superior para la probabilidad de que (a) ocurra.

Parar cada $i \in \left\{1, \dots, \left\lceil n^{\frac{3}{4}} \right\rceil\right\}$, definimos una variable aleatoria W_i de la siguiente forma:

$$W_i = \begin{cases} 1 & \text{si la posición de } R[i] \text{ en la lista } L \text{ ordenada} \\ & \text{es mayor o igual a } \left\lceil \frac{n}{2} \right\rceil + 2 \cdot \left\lfloor n^{\frac{3}{4}} \right\rfloor. \\ 0 & \text{en otro caso.} \end{cases}$$

Además, definimos la variable aleatoria W como $\sum_{i=1}^{\lceil n^{3/4} \rceil} W_i$

Dado que $l \geq \left\lceil \frac{n}{2} \right\rceil + 2 \cdot \left\lfloor n^{\frac{3}{4}} \right\rfloor$, tenemos que la probabilidad de que (a) ocurra es menor o igual a:

$$\Pr\left(W \geq \left\lceil n^{\frac{3}{4}} \right\rceil - \left\lceil \frac{1}{2} \cdot n^{\frac{3}{4}} + n^{\frac{1}{2}} \right\rceil\right)$$

Como L no contiene elementos repetidos obtenemos:

$$\begin{aligned} \Pr(W_i = 1) &= \frac{n - \left(\left\lceil \frac{n}{2} \right\rceil + 2 \cdot \left\lfloor n^{\frac{3}{4}} \right\rfloor\right) + 1}{n} \\ &= \frac{\left\lfloor \frac{n}{2} \right\rfloor - 2 \cdot \left\lfloor n^{\frac{3}{4}} \right\rfloor + 1}{n} \\ &= \frac{\left\lceil \frac{n}{2} \right\rceil - 2 \cdot \left\lfloor n^{\frac{3}{4}} \right\rfloor}{n} \end{aligned}$$

Tenemos entonces que:

$$E(W_i) \ = \ \frac{\left\lceil \frac{n}{2} \right\rceil - 2 \cdot \left\lfloor n^{\frac{3}{4}} \right\rfloor}{n}$$

$$Var(W_i) \ = \ \frac{\left\lceil \frac{n}{2} \right\rceil - 2 \cdot \left\lfloor n^{\frac{3}{4}} \right\rfloor}{n} \cdot \left(1 - \frac{\left\lceil \frac{n}{2} \right\rceil - 2 \cdot \left\lfloor n^{\frac{3}{4}} \right\rfloor}{n} \right)$$

Por lo tanto:

$$\begin{aligned} E(W) &= E\left(\sum_{i=1}^{\lceil n^{3/4} \rceil} W_i \right) \\ &= \sum_{i=1}^{\lceil n^{3/4} \rceil} E(W_i) \\ &= \sum_{i=1}^{\lceil n^{3/4} \rceil} \frac{\left\lceil \frac{n}{2} \right\rceil - 2 \cdot \left\lfloor n^{\frac{3}{4}} \right\rfloor}{n} \\ &= \left\lceil n^{\frac{3}{4}} \right\rceil \cdot \frac{\left\lceil \frac{n}{2} \right\rceil - 2 \cdot \left\lfloor n^{\frac{3}{4}} \right\rfloor}{n} \end{aligned}$$

Pero tenemos que:

$$\begin{aligned} \left\lceil n^{\frac{3}{4}} \right\rceil \cdot \frac{\left\lceil \frac{n}{2} \right\rceil - 2 \cdot \left\lfloor n^{\frac{3}{4}} \right\rfloor}{n} \ &\leq \ \left(n^{\frac{3}{4}} + 1 \right) \cdot \frac{\frac{n}{2} - 2 \cdot n^{\frac{3}{4}} + 3}{n} \\ &= n^{\frac{3}{4}} \cdot \frac{\frac{n}{2} - 2 \cdot n^{\frac{3}{4}} + 3}{n} + \frac{\frac{n}{2} - 2 \cdot n^{\frac{3}{4}} + 3}{n} \\ &= n^{\frac{3}{4}} \cdot \frac{\frac{n}{2} - 2 \cdot n^{\frac{3}{4}}}{n} + 3 \cdot n^{-\frac{1}{4}} + \frac{1}{2} - 2 \cdot n^{-\frac{1}{4}} + \frac{3}{n} \\ &= n^{\frac{3}{4}} \cdot \frac{\frac{n}{2} - 2 \cdot n^{\frac{3}{4}}}{n} + n^{-\frac{1}{4}} + \frac{1}{2} + \frac{3}{n} \\ &= \frac{1}{2} \cdot n^{\frac{3}{4}} - 2 \cdot n^{\frac{1}{2}} + n^{-\frac{1}{4}} + \frac{1}{2} + \frac{3}{n} \\ &\leq \frac{1}{2} \cdot n^{\frac{3}{4}} - 2 \cdot n^{\frac{1}{2}} + 1 \end{aligned}$$

Concluimos que:

$$E(W) \leq \frac{1}{2} \cdot n^{\frac{3}{4}} - 2 \cdot n^{\frac{1}{2}} + 1$$

Para $i, j \in \left\{1, \dots, \left\lceil n^{\frac{3}{4}} \right\rceil\right\}$ tal que $i \neq j$ se tiene que W_i es independiente de W_j.

Concluimos entonces que:

$$\begin{aligned}
Var(W) &= Var\left(\sum_{i=1}^{\lceil n^{3/4} \rceil} W_i\right) \\
&= \sum_{i=1}^{\lceil n^{3/4} \rceil} Var(W_i) \\
&= \sum_{i=1}^{\lceil n^{3/4} \rceil} \frac{\left\lceil \frac{n}{2} \right\rceil - 2 \cdot \left\lfloor n^{\frac{3}{4}} \right\rfloor}{n} \cdot \left(1 - \frac{\left\lceil \frac{n}{2} \right\rceil - 2 \cdot \left\lfloor n^{\frac{3}{4}} \right\rfloor}{n}\right) \\
&= \left\lceil n^{\frac{3}{4}} \right\rceil \frac{\left\lceil \frac{n}{2} \right\rceil - 2 \cdot \left\lfloor n^{\frac{3}{4}} \right\rfloor}{n} \cdot \left(1 - \frac{\left\lceil \frac{n}{2} \right\rceil - 2 \cdot \left\lfloor n^{\frac{3}{4}} \right\rfloor}{n}\right) \\
&\leq \left\lceil n^{\frac{3}{4}} \right\rceil \cdot \frac{\left\lceil \frac{n}{2} \right\rceil - 2 \cdot \left\lfloor n^{\frac{3}{4}} \right\rfloor}{n}
\end{aligned}$$

Además, tenemos que:

$$\begin{aligned}
\left\lceil n^{\frac{3}{4}} \right\rceil \cdot \frac{\left\lceil \frac{n}{2} \right\rceil - 2 \cdot \left\lfloor n^{\frac{3}{4}} \right\rfloor}{n} &\leq \frac{1}{2} \cdot n^{\frac{3}{4}} - 2 \cdot n^{\frac{1}{2}} + 1 \\
&\leq \frac{1}{2} \cdot n^{\frac{3}{4}} + 1 \\
&\leq n^{\frac{3}{4}}
\end{aligned}$$

Concluimos que:

$$Var(W) \leq n^{\frac{3}{4}}$$

Finalmente tenemos que:

$$\begin{aligned}
\Pr\left(W \geq \left\lceil n^{\frac{3}{4}} \right\rceil - \left\lceil \frac{1}{2} \cdot n^{\frac{3}{4}} + n^{\frac{1}{2}} \right\rceil\right) \; &\leq \; \Pr\left(W \geq n^{\frac{3}{4}} - \frac{1}{2} \cdot n^{\frac{3}{4}} - n^{\frac{1}{2}} - 1\right) \\
&= \; \Pr\left(W \geq \frac{1}{2} \cdot n^{\frac{3}{4}} - n^{\frac{1}{2}} - 1\right) \\
&= \; \Pr\left(W \geq \frac{1}{2} \cdot n^{\frac{3}{4}} - 2 \cdot n^{\frac{1}{2}} + 1 + n^{\frac{1}{2}} - 2\right) \\
&\leq \; \Pr\left(W \geq E(W) + n^{\frac{1}{2}} - 2\right) \\
&\leq \; \Pr\left(W \geq E(W) + n^{\frac{1}{2}} - \left(1 - \frac{1}{\sqrt{2}}\right) \cdot n^{\frac{1}{2}}\right) \\
&\leq \; \Pr\left(W \geq E(W) + \sqrt{\frac{n}{2}}\right) \\
&= \; \Pr\left(W - E(W) \geq \sqrt{\frac{n}{2}}\right) \\
&= \; \Pr\left(|W - E(W)| \geq \sqrt{\frac{n}{2}}\right)
\end{aligned}$$

Por lo tanto, utilizando la desigualdad de Chebyshev concluimos que:

$$\begin{aligned}
\Pr\left(W \geq \left\lceil n^{\frac{3}{4}} \right\rceil - \left\lceil \frac{1}{2} \cdot n^{\frac{3}{4}} + n^{\frac{1}{2}} \right\rceil\right) &\leq \Pr\left(|W - E(W)| \geq \sqrt{\frac{n}{2}}\right) \\
&\leq \frac{Var(W)}{\frac{n}{2}} \\
&\leq \frac{n^{\frac{3}{4}}}{\frac{n}{2}} \\
&= 2 \cdot n^{-\frac{1}{4}}
\end{aligned}$$

Concluimos que la probabilidad de que (a) ocurra es menor o igual a $2 \cdot n^{-\frac{1}{4}}$.

Recuerde que estamos considerando una lista $L[1 \ldots n]$ de números enteros donde n es impar y mayor o igual a 2001.

Para la lista L demostramos lo siguiente:

$$\Pr\left(Y_1 < \left\lfloor \frac{1}{2} \cdot n^{\frac{3}{4}} - n^{\frac{1}{2}} \right\rfloor\right) \leq \quad n^{-\frac{1}{4}}$$
$$\Pr\left(Y_2 \leq \left\lceil n^{\frac{3}{4}} \right\rceil - \left\lceil \frac{1}{2} \cdot n^{\frac{3}{4}} + n^{\frac{1}{2}} \right\rceil\right) \leq \quad n^{-\frac{1}{4}}$$
$$\Pr\left(\mathbf{Length}(S) > 4 \cdot \left\lceil n^{\frac{3}{4}} \right\rceil\right) \leq \quad 4 \cdot n^{-\frac{1}{4}}$$

Tenemos entonces que:

$$\Pr(\mathbf{CalcularMediana}(L) \text{ retorne } sin\_resultado) \leq \quad 6 \cdot n^{-\frac{1}{4}}$$

Dado que $n \geq 2001$ concluimos que:

$$\begin{aligned} \Pr(\mathbf{CalcularMediana}(L) \text{ retorne } sin\_resultado) \leq & \quad 6 \cdot 2001^{-\frac{1}{4}} \\ & < \frac{9}{16} \end{aligned}$$

Sea p la probabilidad de que $\mathbf{CalcularMediana}(L)$ retorne resultado.

- Tenemos que $p > \frac{1}{10}$

Sea T una variable aleatoria tal que para cada $i \geq 1$:

$$\Pr(T = i) \quad = \quad (1 - p)^{i-1} \cdot p$$

Vale decir, T representa el número de llamadas a $\mathbf{CalcularMediana}(L)$ hasta obtener un resultado.

Dado que T tiene una distribución geométrica de parámetro p, concluimos que

$$E(T) \quad = \quad \frac{1}{p} \quad \leq \quad \frac{1}{10}$$

Concluimos que en promedio se debe llamar 10 veces a $\mathbf{CalcularMediana}(L)$ para obtener la mediana de la lista L.

Aritmética Modular y más

Aritmética Modular

Dados dos números $a, b \in \mathbb{Z}$, si $b > 0$ entonces $\alpha, \beta \in \mathbb{Z}$ tales que $0 \leq \beta \leq b$ y

$$a = \alpha \cdot b + \beta$$

Además, estos números α, β **son únicos**.

β es llamado **el resto de la división entera entre a y b**, y es denotado como

$$a \bmod b := \beta$$

Definición

$$a \equiv_n b \quad \text{si, y solo si, } n \text{ divide a } (b - a)$$

Usamos la notación $n \mid m$ para indicar que n divide a m.

- $a \equiv_n b$ si $n \mid (b - a)$.

Proposición

1. Si $a \equiv_n b$ y $c \equiv_n d$, entonces:
$$(a + c) \equiv_n (b + d)$$
$$(a \cdot c) \equiv_n (b \cdot d)$$

Exponenciación rápida: Calculando $a^b \bmod b$

Utilizamos el siguiente algoritmo:

```
EXP(a, b, n)
    if b = 1 then return a mod n
    else if b es par then
        val := EXP(a, b/2, n)
        return (val · val) mod n
    else
        val := EXP(a, (b−1)/2, n)
        return (val · val · a) mod n
```

Máximo común Divisor

Definición

Sea $a, b \in \mathbb{Z} - \{0\}$. Se define el máximo común divisor $\text{gcd}(a, b)$ de a y b como el mayor número d tal que $d \mid a$ y $d \mid b$.

En otras palabras, $\text{gcd}(a, b)$ es el máximo del conjunto

$$D_{a,b} = \{\, c \in \mathbb{Z} \mid c \mid a \wedge c \mid b \,\}$$

Proposición

Para todo $a, b \in \mathbb{Z} - \{0\}$, $\text{gcd}(a, b) = \text{gcd}(b, a \bmod b)$

Demostración

Vamos a demostrar que para $c \in \mathbb{Z} - \{0\}$

$$c \mid a \;\text{ y } c \mid b \qquad \Leftrightarrow \qquad c \mid b \;\text{ y }\; c \mid (a \bmod b)$$

De esto se concluye que $\text{gcd}(a, b) = \text{gcd}(b, a \bmod b)$.

Sabemos que $a = \alpha \cdot b + (a \bmod b)$

($\Rightarrow$) Suponga que $c \mid a$ y $c \mid b$.
Dado que $(a \bmod b) = a - \alpha \cdot b$, concluimos que $c \mid (a \bmod b)$

($\Leftarrow$) Suponga que $c \mid b$ y $c \mid (a \bmod b)$.
Dado que $a = \alpha \cdot b + (a \bmod b)$, tenemos que $c \mid a$.

De lo anterior, concluimos la siguiente identidad para $a > 0$:

$$\text{gcd}(a, b) = \begin{cases} a & b = 0 \\ \text{gcd}(b, a \bmod b) & b > 0 \end{cases}$$

Usamos esta identidad para generar un algoritmo para calcular el máximo común divisor, el cuales conocido como **Algoritmo de Euclides**.

MCD$(a,\ b)$
  if $a = 0$ **and** $b = 0$ **then return** error
  else if $a = 0$ **then return** b
  else if $b = 0$ **then return** a
  else if $a \geq b$ **then return MCD**$(b,\ a \bmod b)$
  else return MCD$(a,\ b \bmod a)$

Lema

Si $a \geq b$ y $b > 0$, entonces $(a \bmod b) < \frac{a}{2}$

Demostración

Si $b > \frac{a}{2}$:

$$a \bmod b = a - b < a - \frac{a}{2} = \frac{a}{2}$$

Si $b < \frac{a}{2}$:

$$a \bmod b < b < \frac{a}{2}$$

Si $b = \frac{a}{2}$ (a debe ser par):

$$a \bmod b = 0 < b = \frac{a}{2}$$

Definición

b es **inverso de a en módulo n** si

$$a \cdot b \equiv_n 1$$

Identidad de Bézout

Para cada $a, b \in \mathbb{N}$ tales que $a \neq 0$ o $b \neq 0$, existen $s, t \in \mathbb{Z}$ tales que:

$$\gcd(a, b) \quad = \quad s \cdot a + t \cdot b$$

Demostración:

Sean $a, b \in \mathbb{Z} - \{0\}$. Tomamos:

$$S = \{\, a \cdot x + b \cdot y \mid x, y \in \mathbb{Z} \quad \wedge \quad a \cdot x + b \cdot y > 0 \,\}$$

Sabemos que $S \neq \emptyset$ dado que contiene a a o $-a$ (con $x = \pm 1$ y $y = 0$). Como S es un conjunto no vacío de números positivos, tiene un mínimo elemento $d = a \cdot s + b \cdot t$. Para mostrar que d es el máximo común divisor de a y b, primero debemos mostrar que sí es un común divisor, y que para cualquier otro común divisor c, tenemos que $c \leq d$.

Sabemos que:

$$a = d \cdot q + r \qquad (a \bmod q \ = \ r)$$

Y sabemos que $r \in S \cup \{0\}$ dado que:

$$\begin{aligned} r &= a - q \cdot d \\ &= a - q(a \cdot s + b \cdot t) \\ &= a \cdot (1 - q \cdot s) - b \cdot qt \end{aligned}$$

De este modo, r es de la forma $a \cdot x + b \cdot y$, y por lo tanto $r \in S \cup \{0\}$. Sin embargo, $0 \leq r < d$, y d es el entero positivo más pequeño en S: El resto r puede por lo tanto NO estar en S, haciendo r necesariamente igual a 0. Esto implica que d es divisor de a. Similarmente, d también es divisor de b, y, por lo tanto, $d \mid a$ y $d \mid b$.

Ahora, sea c cualquier divisor común de a y b, esto es, que existen u y v tal que $a = c \cdot u$, y $b = c \cdot v$:

$$\begin{aligned} d &= a \cdot s + b \cdot t \\ &= c \cdot u \cdot s + c \cdot v \cdot t \\ &= c(u \cdot s + v \cdot t) \end{aligned}$$

Luego, $c \mid d$. Y como $d > 0$, esto implica que $c \leq d$.

Teorema

a tiene inverso en módulo n si y sólo si $\gcd(a, n) = 1$.

Demostración

$(\Rightarrow)$ Suponga que b es inverso de a en módulo n:

$$a \cdot b \equiv_n 1$$

Se deduce que $a \cdot b = \alpha \cdot n + 1$, por lo que $1 = a \cdot b - \alpha \cdot n$.
Concluimos que si $c \mid a$ y $c \mid n$, entonces $c \mid 1$. Por lo tanto c debe ser igual a 1, de lo contrario concluimos que $\gcd(a, n) = 1$.

$(\Leftarrow)$ Suponga que $\gcd(a, n) = 1$. Por la identidad de Bézout existen $s, t \in \mathbb{Z}$ tales que:

$$1 \quad = \quad s \cdot n + t \cdot a$$

Entonces $a \cdot t \equiv_n 1$. Así a tiene inverso en módulo n.

Sabemos que **MCD** es un algoritmo eficiente para calcular el máximo común divisor entre dos números.

¡Pero este algoritmo puede hacer más! Puede ser extendido para calcular s y t tales que:

$$\gcd(a, b) = s \cdot a + t \cdot b$$

Vamos a usar este algoritmo para calcular inversos modulares.

Algoritmo Extendido de Euclides

Suponga que $a \geq b$, y defina la siguiente **sucesión**:

$$
\begin{aligned}
r_0 &= a \\
r_1 &= b \\
r_i &= r_{i-2} \bmod r_{i-1} \quad (i \geq 2)
\end{aligned}
$$

y calculamos esta sucesión hasta un número k tal que $r_k = 0$.
Luego, $\boldsymbol{r_{k-1} = \gcd(a, b)}$.

Al mismo tiempo, podemos ir calculando dos sucesiones s_i, t_i tales que:

$$r_i = s_i \cdot a + t_i \cdot b$$

Tenemos que:

$$\gcd(a, b) = r_{k-1} = s_{k-1} \cdot a + t_{k-1} \cdot b$$

Sean:

$$
\begin{aligned}
s_0 &= 1 \qquad & t_0 &= 0 \\
s_1 &= 0 \qquad & t_1 &= 1
\end{aligned}
$$

Se tiene que:

$$
\begin{aligned}
r_0 &= s_0 \cdot a + \cancelto{0}{t_0 \cdot b} = a \\
r_1 &= \cancelto{0}{s_1 \cdot a} + t_1 \cdot b = b
\end{aligned}
$$

Dado que $r_{i-1} = \overbrace{\left\lfloor \frac{r_{i-1}}{r_i} \right\rfloor}^{\text{División parte entera}} \cdot r_i + \overbrace{r_{i-1} \bmod r_i}^{\text{Resto}}$, tenemos que:

$$r_{i-1} = \left\lfloor \frac{r_{i-1}}{r_i} \right\rfloor \cdot r_i + r_{i+1}$$

Por lo tanto:

$$s_{i-1} \cdot a + t_{i-1} \cdot b = \left\lfloor \frac{r_{i-1}}{r_i} \right\rfloor \cdot (s_i \cdot a + t_i \cdot b) + r_{i+1}$$

Concluimos que:

$$r_{i+1} = \left(s_{i-1} - \left\lfloor \frac{r_{i-1}}{r_i} \right\rfloor \cdot s_i\right) \cdot a + \left(t_{i-1} - \left\lfloor \frac{r_{i-1}}{r_i} \right\rfloor \cdot t_i\right) \cdot b$$

Definimos entonces:

$$
\begin{aligned}
s_{i+1} &= s_{i-1} - \left\lfloor \frac{r_{i-1}}{r_i} \right\rfloor \cdot s_i \\
t_{i+1} &= t_{i-1} - \left\lfloor \frac{r_{i-1}}{r_i} \right\rfloor \cdot t_i
\end{aligned}
$$

Ejemplo

Vamos a usar el algoritmo para $a = 60$ y $b = 13$

Inicialmente:

$$\begin{array}{lll} r_0 = 60 & s_0 = 1 & t_0 = 0 \\ r_1 = 13 & s_0 = 0 & t_0 = 1 \end{array}$$

Entonces tenemos que:

$$r_2 = r_0 \bmod r_1$$
$$s_2 = s_0 - \left\lfloor \frac{r_0}{r_1} \right\rfloor \cdot s_1$$
$$t_2 = t_0 - \left\lfloor \frac{r_0}{r_1} \right\rfloor \cdot t_1$$

Por lo tanto:

$$r_2 = 8 \qquad s_2 = 1 \qquad t_2 = -4$$

Y el proceso continúa:

$$\begin{array}{lll} r_3 = 5 & s_3 = -1 & t_3 = 5 \\ r_4 = 3 & s_4 = 2 & t_4 = -9 \\ r_5 = 2 & s_5 = -3 & t_5 = 14 \\ r_6 = 1 & s_6 = 5 & t_6 = -23 \\ r_7 = 0 & s_7 = -13 & t_7 = 60 \end{array}$$

Tenemos que: $1 = 5 \cdot 60 + (-23) \cdot 13$

Dados dos números naturales a y n, con $n \geq 2$, si el inverso de a en módulo n existe el siguiente algoritmo lo retorna, y en caso contrario indica que no existe.

Inverso$(a,\ n)$
  if MCD$(a, n) > 1$ **then** *no_existe_inverso*
  else
    $s_0 := 1$
    $t_0 := 0$
    $s_1 := 0$
    $t_1 := 1$
    $r_0 := n$
    $r_1 := a$
while $r_1 > 1$ **do**
  $aux\_s := s_0 - \left\lfloor \frac{r_0}{r_1} \right\rfloor \cdot s_1$
  $s_0 := s_1$
  $s_1 := aux\_s$
  $aux\_t := t_0 - \left\lfloor \frac{r_0}{r_1} \right\rfloor \cdot t_1$
  $t_0 := t_1$
  $t_1 := aux\_t$
  $r_0 := r_1$
  $r_1 := s_1 \cdot n + t_1 \cdot a$
return t_1

Ejemplo para mortales:

¿Cuántas veces cabe el 54 en 888?

$$\gcd(888{,}54) = \ 6:$$

$$888 = 54(16) + 24$$

$$54 = 24(2) + \boxed{6}$$

Nuestro último resto $\neq 0$ será el gcd

$$24 = 6(4) + 0$$

Pequeño Teorema de Fermat

Sea p un número primo. Si $a \in \{0, \dots, p-1\}$, entonces:

$$a^p \equiv a \pmod p$$

Corolario

Sea p un número primo. Si $a \in \{1, \dots, p-1\}$, entonces:

$$a^{p-1} \equiv 1 \pmod p$$

Demostración

Por teorema anterior sabemos que

$$a^p \equiv a \pmod p$$

Por lo tanto, existe $\alpha \in \mathbb{Z}$ tal que:

$$a^p - a = \alpha \cdot p$$

Dado que $a \mid (a^p - a)$, se tiene que $a \mid (\alpha \cdot p)$

Por lo tanto, dado que $a \in \{1, \dots, p-1\}$ y p es un número primo, se concluye que $a \mid \alpha$. Entonces $(a^p - 1) = \frac{\alpha}{a} \cdot p$, donde $\frac{\alpha}{a}$ es un número entero.

- Concluimos que $a^{p-1} \equiv 1 \quad mod\ p$.

Test de primalidad: Primera versión

El test de primalidad que vamos a estudiar está basado en estas propiedades ($n \geq 2$):

1. Si n es primo y $a \in \{1, \ldots, n-1\}$, entonces:

$$a^{n-1} \equiv 1 \pmod n$$

2. Si n es compuesto, entonces existe $a \in \{1, .., n-1\}$ tal que

$$a^{n-1} \not\equiv 1 \pmod n$$

Demostración

Suponga que n es compuesto. Sea $a \in \{1, \ldots, n-1\}$ tal que $\gcd(a, n) > 1$.
Luego, a no tiene inverso en módulo n.
Concluimos que

$$a^{n-1} \not\equiv 1 \pmod n$$

Dado que a^{n-2} no puede ser inverso de a en módulo n.

Para $n \geq 2$, defina el conjunto $\mathbb{Z}_n^*$ como:

$$\mathbb{Z}_n^* \quad = \quad \{a \in \{1, \ldots, n-1\} \mid \gcd(a, n) = 1\}$$

Es decir, $\mathbb{Z}_n^*$ es el conjunto de todos los primos relativos de n que son menores a él.

Suponga que n es compuesto. Luego, si $a \in \{1, \ldots, n-1\} - \mathbb{Z}_n^*$, entonces $a^{n-1} \not\equiv 1 \pmod n$.

Test de primalidad entonces depende de qué tan grande es $\mathbb{Z}_n^*$.
Supongamos que $|\mathbb{Z}_n^*| \leq \left\lfloor \frac{n}{2} \right\rfloor$ para cada número compuesto $n \geq 2$.

```
TestPrimalidad1(n)
    sea a un número elegido de manera uniforme desde {1, . . . , n − 1}
    if EXP(a, n − 1, n) ≠ 1
        then return COMPUESTO
    else
        return PRIMO
```

Ejercicio:

De un algoritmo que reciba como parámetros a dos números enteros $n \geq 2$ y $k \geq 1$, y determina si n es un número primo con probabilidad de error menor o igual a $\left(\frac{1}{2}\right)^k$.

```
TestPrimalidad2(n, k)
    sea a_1, ..., a_k una secuencia de números elegidos de
                 manera uniforme e independiente desde {1, ..., n − 1}
    for i := 1 to k do
        if EXP(a_i, n − 1, n) ≠ 1
            then return COMPUESTO
    return PRIMO
```

¿Pero la probabilidad de error de **TestPrimalidad2** está bien acotada?

Supusimos que $|\mathbb{Z}_n^*| \leq \left\lfloor \frac{n}{2} \right\rfloor$ para cada número compuesto $n \geq 2$.

¿Es correcta esta suposición?

Considere la **función de Euler**: $\phi: \mathbb{N} \to \mathbb{N}$ definida como:

$$\phi(n) \begin{cases} 0 & n = 1 \\ |\mathbb{Z}_n^*| & n > 1 \end{cases}$$

Teorema

$$\phi(n) \in \Omega\left(\frac{n}{\log_2(\log_2 n)}\right)$$

Conclusión

Para cada número n, el conjunto $\mathbb{Z}_n^*$ tiene un número de elementos cercano a n.

- No es cierto que $|\mathbb{Z}_n^*| \leq \left\lfloor \frac{n}{2} \right\rfloor$ para cada número compuesto $n \geq 2$.
- No podemos basar nuestro test en los elementos del conjunto $(\{1, \ldots, n-1\} - \mathbb{Z}_n^*)$.

Test de primalidad: Segunda versión

Una observación importante: si n es compuesto, entonces puede existir $a \in \mathbb{Z}_n^*$ tal que $a^{n-1} \not\equiv 1 \pmod{n}$.

- Por ejemplo: $3^{15} \bmod 16 = 11$.

En lugar de considerar $\mathbb{Z}_n^*$ en el test de primalidad, consideramos:

$$J_n = \{a \in \mathbb{Z}_n^* \mid a^{n-1} \equiv 1 \pmod{n}\}$$

Si demostramos que para cada número compuesto n se tiene que $|J_n| \leq \frac{1}{2} \cdot |\mathbb{Z}_n^*|$, entonces tenemos un test de primalidad.

- Puesto que para p primo: $|J_p| = |\mathbb{Z}_p^*| = p - 1$.

Recuerde que en nuestros algoritmos consideramos $n \geq 2$.

```
TestPrimalidad3(n, k)
    sea a_1, ..., a_k una secuencia de números elegidos de
                manera uniforme e independiente desde {1, ..., n − 1}
    for i := 1 to k do
        if MCD(a_i, n) > 1 then return COMPUESTO
        else
            if EXP(a_i, n − 1, n) ≠ 1
                then return COMPUESTO
    return PRIMO
```

Si después de las k iteraciones nunca devuelve $COMPUESTO$, entonces con $p \geq \left(\frac{1}{2}\right)^k$ el algoritmo entregara correctamente que el número es $PRIMO$.

Teoría de Grupos

Definición

Un conjunto G y una función (total) $\circ : G \times G \rightarrow G$ forman un **grupo** si:

1. (**Asociatividad**) Para cada $a, b, c \in G$:

$$(a \circ b) \circ c = a \circ (b \circ c)$$

2. (**Elemento neutro**) Existe $e \in G$ tal que para cada $a \in G$:

$$a \circ e = e \circ a = a$$

3. (**Inverso**) Para cada $a \in G$, existe $b \in G$:

$$a \circ b = b \circ a = e$$

Propiedades básicas

- Neutro es único.
- Inverso de cada elemento a es único.

Recordatorio…

$\subset \rightarrow$ Subconjunto.
$\subseteq \rightarrow$ Subconjunto o igual.
$\subsetneq \rightarrow$ Subconjunto pero no igual.
$\not\subseteq \rightarrow$ Ni subconjunto ni igual.

Definición

$(H, \circ)$ es un subgrupo de un grupo $(G, \circ)$, para cada $\emptyset \subsetneq H \subseteq G$, si $(H, \circ)$ es un grupo.

Propiedades básicas

- Si e_1 es el neutro en $(G, \circ)$ y e_2 es el neutro en $(H, \circ)$, entonces $e_1 = e_2$.
- Para cada $e \in H$, si b es el inverso de a en $(G, \circ)$ y c es el inverso de a en $(H, \circ)$, entonces $c = b$.

Teorema de Lagrange

Si $(G, \circ)$ es un grupo finito y $(H, \circ)$ es un subgrupo de $(G, \circ)$, entonces $|H|$ divide a $|G|$.

Demostración

Suponga que e es el elemento neutro de $(G, \circ)$ y a^{-1} es el inverso de a en $(G, \circ)$.
Sea $\sim$ una relación binaria sobre G definida como:

$$a \sim b \quad \text{si y sólo si} \quad b \circ a^{-1} \in H$$

Lema

$\sim$ es una relación de equivalencia.

Demostración

(**Refleja**) $a \sim a$ ya que $a \circ a^{-1} = e$ y $e \in H$.

Un elemento se relaciona consigo mismo a través de la relación, y el resultado pertenece al dominio.

(**Simétrica**) Suponga que $a \sim b$. Demostramos que $b \sim a$.

Dado que $a \sim b$: $b \circ a^{-1} \in H$, tenemos que demostrar que $a \circ b^{-1} \in H$.
Tenemos que:

$$\begin{aligned} (b \circ a^{-1}) \circ (a \circ b^{-1}) &= \left(b \circ (a^{-1} \circ a)\right) \circ b^{-1} \\ &= (b \circ e) \circ b^{-1} \\ &= b \circ b^{-1} \\ &= e \end{aligned}$$

Como se cumple que $(a \circ b^{-1}) \circ (b \circ a^{-1}) = e$, entonces:

$$(b \circ a^{-1})^{-1} = a \circ b^{-1}$$

Concluimos que $a \circ b^{-1}$ está en H, ya que $(H, \circ)$ es un subgrupo de $(G, \circ)$.

(**Transitiva**) Suponga que $a \sim b$ y $b \sim c$. Tenemos que demostrar que $a \sim c$.

Por hipótesis: $b \circ a^{-1} \in H$ y $c \circ b^{-1} \in H$. Tenemos que demostrar que $c \circ a^{-1} \in H$.
Pero $(c \circ b^{-1}) \circ (b \circ a^{-1}) = c \circ a^{-1}$ y $\circ$ es cerrada en H.
Por lo tanto: $c \circ a^{-1} \in H$.

Lema

Sea $[a]_\sim$ la clase de equivalencia de $a \in G$ bajo la relación $\sim$. Luego:

1. $[e]_\sim = H$.
2. Para cada $a, b \in G$: $|[a]_\sim| = |[b]_\sim|$.

Demostración

1. Se tiene que:

$$\begin{aligned} a \in [e]_\sim &\Leftrightarrow e \sim a \\ &\Leftrightarrow a \circ e^{-1} \in H \\ &\Leftrightarrow a \circ e \in H \\ &\Leftrightarrow a \in H \end{aligned}$$

2. Sean $a, b \in G$, y defina la función f de la siguiente forma:

$$f(x) = x \circ (a^{-1} \circ b)$$

Se tiene que:

$$\begin{aligned} x \in [a]_\sim &\Rightarrow a \sim x \\ &\Rightarrow x \circ a^{-1} \in H \\ &\Rightarrow (x \circ a^{-1}) \circ e \in H \\ &\Rightarrow (x \circ a^{-1}) \circ (b \circ b^{-1}) \in H \\ &\Rightarrow \big(x \circ (a^{-1} \circ b)\big) \circ b^{-1} \in H \\ &\Rightarrow f(x) \circ b^{-1} \in H \\ &\Rightarrow b \sim f(x) \\ &\Rightarrow f(x) \in [b]_\sim \end{aligned}$$

Por lo tanto: $f: [a]_\sim \to [b]_\sim$

Vamos a demostrar que f es una **biyección**, de lo cual concluimos que $|[a]_\sim| = |[b]_\sim|$.

<u>f es inyectiva (1-1):</u>

$$\begin{aligned}
f(x) = f(y) \quad &\Rightarrow \quad x \circ (a^{-1} \circ b) = y \circ (a^{-1} \circ b) \\
&\Rightarrow \quad \big(x \circ (a^{-1} \circ b)\big) \circ (b^{-1} \circ a) = \big(y \circ (a^{-1} \circ b)\big) \circ (b^{-1} \circ a) \\
&\Rightarrow \quad \big(x \circ (a^{-1} \circ (b \circ b^{-1}) \circ a)\big) = \big(y \circ (a^{-1} \circ (b \circ b^{-1}) \circ a)\big) \\
&\Rightarrow \quad \Big(x \circ \big((a^{-1} \circ e) \circ a\big)\Big) = \Big(y \circ \big((a^{-1} \circ e) \circ a\big)\Big) \\
&\Rightarrow \quad \big(x \circ (a^{-1} \circ e)\big) = \big(y \circ (a^{-1} \circ e)\big) \\
&\Rightarrow \quad x \circ e = y \circ e \\
&\Rightarrow \quad x = y
\end{aligned}$$

<u>f es sobreyectiva:</u>

$$\begin{aligned}
y \in [b]_\sim \quad &\Rightarrow \quad b \sim y \\
&\Rightarrow \quad y \circ b^{-1} \in H \\
&\Rightarrow \quad (y \circ b^{-1}) \circ (a \circ a^{-1}) \in H \\
&\Rightarrow \quad \big((y \circ b^{-1}) \circ a\big) \circ a^{-1} \in H \\
&\Rightarrow \quad a \sim \big((y \circ b^{-1}) \circ a\big) \\
&\Rightarrow \quad \big((y \circ b^{-1}) \circ a\big) \in [a]_\sim
\end{aligned}$$

Sea $x = \big((y \circ b^{-1}) \circ a\big)$. Tenemos que:

$$\begin{aligned}
f(x) &= x \circ (a^{-1} \circ b) \\
&= \big((y \circ b^{-1}) \circ a\big) \circ (a^{-1} \circ b) \\
&= y \circ (b^{-1} \circ (a \circ a^{-1}) \circ b) \\
&= y \circ \big((b^{-1} \circ e) \circ b\big) \\
&= y \circ (b^{-1} \circ b) \\
&= y \circ e \\
&= y
\end{aligned}$$

Dejamos pendiente la siguiente pregunta:

¿Qué enfoque podríamos usar para demostrar que $|J_n| \leq \frac{1}{2} \cdot |\mathbb{Z}_n^*|$?

R: Usamos el Teorema de Lagrange.

Dado que $(J_n, \cdot)$ es un subgrupo de $(\mathbb{Z}_n^*, \cdot)$:

Si existe $a \in (\mathbb{Z}_n^* \setminus J_n)$, entonces $|J_n| \leq \frac{1}{2} \cdot |\mathbb{Z}_n^*|$

Definición

Un **número n es de Carmichael** si $n \geq 2$, n es compuesto y $|J_n| = |\mathbb{Z}_n^*|$.

Ejemplo
561, 1105 y 1792 son números de Carmichael.

Teorema
Existe un número infinito de números de Carmichael.

Conclusión: Este test de primalidad NO va a funcionar, pero no todo está perdido:

En lugar de utilizar J_n, vamos a usar las herramientas que desarrollamos sobre el siguiente conjunto (n impar):

$$S_n \quad = \quad \left\{ a \in \mathbb{Z}_n^* \mid a^{\frac{n-1}{2}} \equiv 1 \pmod{n} \;\; \text{ó} \;\; a^{\frac{n-1}{2}} \equiv -1 \pmod{n} \right\}$$

Vamos a diseñar un test de primalidad considerando los conjuntos:

$$\begin{aligned} S_n^+ &= \left\{ a \in \mathbb{Z}_n^* \mid a^{\frac{n-1}{2}} \equiv 1 \pmod{n} \right\} \\ S_n^- &= \left\{ a \in \mathbb{Z}_n^* \mid a^{\frac{n-1}{2}} \equiv -1 \pmod{n} \right\} \end{aligned}$$

Así, podemos definir S_n a partir de estos conjuntos:

$$S_n = S_n^+ \cup S_n^-$$

Para hacer esto necesitamos estudiar algunas propiedades de los conjuntos S_n^+, S_n^- y S_n.

- Consideramos primero el caso en que n es primo, y luego el caso en que n es compuesto.

Proposición 1

Si $n \geq 3$ es primo, entonces $S_n = \mathbb{Z}_n^*$.

Demostración

Si $a \in \{1, \dots, n-1\}$, tenemos que $a^{n-1} \equiv 1 \pmod{n}$

Por lo tanto $\left(a^{\frac{n-1}{2}}\right)^2 \equiv 1 \pmod{n}$, de lo cual se deduce que:

$$\left(a^{\frac{n-1}{2}} + 1\right) \cdot \left(a^{\frac{n-1}{2}} - 1\right) \equiv 0 \pmod{n}$$

Así, dado que n es primo se concluye que $a^{\frac{n-1}{2}} \equiv 1 \pmod{n}$ ó $a^{\frac{n-1}{2}} \equiv -1 \pmod{n}$.

Proposición 2

Si $n \geq 3$ es primo: $|S_n^+| = |S_n^-| = \frac{n-1}{2}$

Para demostrar la proposición, primero vamos a demostrar un lema.

Sea $p(x)$ un polinomio:

$$p(x) = \sum_{i=0}^{k} a_i x^i,$$

donde $k \geq 1$, cada $a_j \in \{0, \ldots, n-1\}$ para $0 \leq j \leq k-1$, y $a_k \neq 0$.

Decimos que a es una **raíz de $p(x)$ en módulo n** si:

$$p(a) \equiv_n 0$$

Lema

$p(x)$ tiene a lo más k raíces en módulo n.

Demostración

Decimos que dos polinomios $p_1(x)$ y $p_2(x)$ son **congruentes en módulo n** si para todo $a \in \{0, \ldots, n-1\}$:

$$p_1(a) \equiv_n p_2(a)$$

Sea a una raíz de $p(x)$ en módulo n. Vamos a demostrar que existe un polinomio $q(x)$ de grado $k-1$ tal que:

$$p(x) \equiv_n \ (x-a) \cdot q(x)$$

Veamos que al demostrar esta propiedad se concluye que el lema es cierto.

Si c es una raíz de $p(x)$ en módulo n, entonces $p(c) \equiv_n 0$.
- Como $p(x) \equiv_n (x-a) \cdot q(x)$, concluimos que $(c-a) \cdot q(c) \equiv_n 0$.

Dado que n es primo, si $d \cdot e \equiv_n 0$, entonces $d \equiv_n 0$ o $e \equiv_n 0$.
- Tenemos entonces que $c \equiv_n a$ o $q(c) \equiv_n 0$.

Así, tenemos que c es la raíz a que ya habíamos identificado o es una raíz de $q(x)$ en módulo n.

Concluimos que el número de raíces de $p(x)$ en módulo n es menor o igual a uno más el número de raíces de $q(x)$ en módulo n.

- Como $q(x)$ tiene grado $k-1$, si continuamos usando este argumento (o usamos inducción) concluimos que el número de raíces de $p(x)$ es menor o igual a k.

Nótese que el argumento anterior no funciona si n es compuesto.

- Dado que podemos tener d y e tales que $d \cdot e \equiv_n 0$, $d \not\equiv_n 0$ y $e \not\equiv_n 0$.

De hecho, si n es compuesto no es necesariamente cierto que el número de raíces de un polinomio está acotado superiormente por su grado.

Ejemplo

Si $n = 35$, tenemos que $5 \cdot 7 \equiv_{35} 0$, pero $5 \not\equiv_{35} 0$ y $7 \not\equiv_{35} 0$.
En este caso tenemos cuatro raíces para el polinomio $p(x) = x^2 - 1$.

- Ya que $1^2 \equiv_{35} 1$, $6^2 \equiv_{35} 1$, $29^2 \equiv_{35} 1$ y $34^2 \equiv_{35} 1$.

Volvemos entonces a la demostración de que existe un polinomio $q(x)$ de grado $k-1$ tal que:

$$p(x) \equiv_n (x-a) \cdot q(x)$$

Definimos $q(x)$ como:

$$q(x) = \sum_{i=0}^{k-1} b_i x^i$$

donde $b_i = a_{i+1} + a_{i+2} \cdot a + \cdots + a_k \cdot a^{k-1-i}$
Se tiene que:

$$\begin{aligned}
(x-a) \cdot q(x) &= \left(\sum_{i=0}^{k-1} b_i x^{i+1}\right) + \left(\sum_{i=0}^{k-1} (-a \cdot b_i) x^{i+1}\right) \\
&= \left(\sum_{i=1}^{k} b_{i-1} x^{i}\right) + \left(\sum_{i=0}^{k-1} (-a \cdot b_i) x^{i+1}\right) \\
&= b_{k-1} \cdot x^k + \left(\sum_{i=1}^{k-1} (b_{i-1} - a \cdot b_i) x^{i+1}\right) - a \cdot b_0
\end{aligned}$$

Así, dado que:

$$b_{k-1} = a_k$$

Y dado que para $i \in \{1, \dots, k-1\}$:

$$\begin{aligned}(b_{i-1} - a \cdot b_i) &= a_i + a_{i+1} \cdot a + \cdots + a_k \cdot a^{k-i} - a \cdot \left(a_{i+1} + \cdots + a_k \cdot a^{k-1-i}\right) \\ &= a_i + a_{i+1} \cdot a + \cdots + a_k \cdot a^{k-i} - a_{i+1} \cdot a - \cdots - a_k \cdot a^{k-1} \\ &= a_i\end{aligned}$$

Concluimos que:

$$(x-a) \cdot q(x) = \left(\sum_{i=1}^{k} a_i \cdot x^i\right) - a \cdot b_0$$

Pero:

$$\begin{aligned}-a \cdot b_0 &= -a \cdot (a_1 + a_2 \cdot a + \cdots + a_k \cdot a^{k-1}) \\ &= -a_1 \cdot a - a_2 \cdot a^2 - \cdots - a_k \cdot a^k\end{aligned}$$

De lo cual deducimos que:

$$a_0 \equiv_n - a \cdot b_0$$

ya que $a_k \cdot a^k + \cdots + a_1 \cdot a + a_0 \equiv_n 0$.

Tenemos entonces que:

$$(x-a) \cdot q(x) \equiv_n p(x)$$

Sea $R = \left\{\{b^2 \mid 1 \le b \le \frac{n-1}{2}\right\}$

Por el Teorema de Fermat, tenemos que:

$$R \subseteq \left\{a \in \{1, \dots, n-1\} \mid a^{\frac{n-1}{2}} \equiv_n 1\right\}$$

Además, sabemos que si $1 \le b < c \le \frac{n-1}{2}$ y $b^2 \equiv_n c^2$:

$$(c-b) \cdot (c+b) \equiv_n 0$$

Así, dado que $2 \le b + c \le n - 1$, concluimos que $b \equiv_n c$.

- Dado que n es primo.

Pero $b \equiv_n c$, no puede ser cierto puesto que $1 \le (c-b) \le \frac{n-1}{2}$.

- Por lo tanto: $|R| = \frac{n-1}{2}$

Además, sabemos que $p(x) = x^{\frac{n-1}{2}} - 1$ tiene a lo más $\frac{n-1}{2}$ raíces en módulo n.

- Por lo tanto: $\left|\left\{a \in \{1, \dots, n-1\} \mid a^{\frac{n-1}{2}} \equiv_n 1\right\}\right| \leq \frac{n-1}{2}$

Concluimos que:

$$\frac{n-1}{2} = |R| \leq \left|\left\{a \in \{1, \dots, n-1\} \mid a^{\frac{n-1}{2}} \equiv_n 1\right\}\right| \leq \frac{n-1}{2}$$

Por lo tanto:

$$|S_n^+| = \left|\left\{a \in \{1, \dots, n-1\} \mid a^{\frac{n-1}{2}} \equiv_n 1\right\}\right| = \frac{n-1}{2}$$

Así, dado que $|S_n| = |\mathbb{Z}_n^*| = n - 1$ y $|S_n^+| + |S_n^-| = |S_n|$, concluimos que:

$$|S_n^-| \;\; = \;\; \frac{n-1}{2}$$

■

Teorema

Sea $n = n_1 \cdot n_2$, donde $n_1, n_2 \geq 3$ y $\gcd(n_1, n_2) = 1$. Si existe $a \in \mathbb{Z}_n^*$ tal que $a^{\frac{n-1}{2}} \not\equiv_n -1$, entonces:

$$|S_n| \;\; \leq \;\; \frac{1}{2} \cdot |\mathbb{Z}_n^*|$$

Para demostrar el teorema necesitaremos el **Teorema Chino del resto**.

Teorema Chino del Resto:

Suponga que $\gcd(m, n) = 1$. Para todo a y b, existe c tal que:

$$c \equiv_m a$$
$$c \equiv_n b$$

Demostración:

Dado que $\gcd(m, n) = 1$, existen d y e tales que:

$$n \cdot d \equiv_m 1$$
$$m \cdot e \equiv_n 1$$

Sea $c = a \cdot n \cdot d + b \cdot m \cdot e$. Se tiene que:

$$c \equiv_m a$$
$$c \equiv_n b$$

Ahora, para el demostrar el teorema anterior, suponga que $e \in \mathbb{Z}_n^*$ y $a^{\frac{n-1}{2}} \equiv_n -1$

Por Teorema Chino del Resto, existe b tal que:

$$b \equiv_{n_1} a$$
$$b \equiv_{n_2} 1$$

Entonces: $a = \alpha \cdot n_1 + b$ y $1 = \beta \cdot n_2 + b$

- Por lo tanto $\gcd(b, n) = 1$, ya que $n = n_1 \cdot n_2$ y $a \in \mathbb{Z}_n^*$

Además, tenemos que:

$$\begin{aligned} b^{\frac{n-1}{2}} \equiv_{n_1} a^{\frac{n-1}{2}} &\equiv_{n_1} -1 \\ b^{\frac{n-1}{2}} &\equiv_{n_2} 1 \end{aligned}$$

Dado que $n = n_1 \cdot n_2$ concluimos que:

$$\begin{aligned} b^{\frac{n-1}{2}} &\not\equiv_n 1 \\ b^{\frac{n-1}{2}} &\not\equiv_n -1 \end{aligned}$$

Sea $c = (b \mod n)$. Concluimos que $c \notin S_n$ y $c \in \mathbb{Z}_n^*$. Es decir:

$$S_n \subsetneq \mathbb{Z}_n^*$$

Pero se tiene que $(S_n, \cdot)$ es un subgrupo de $(\mathbb{Z}_n^*, \cdot)$.
Entonces por Teorema de Lagrange:

$$|S_n| \leq \frac{1}{2} \cdot |\mathbb{Z}_n^*|$$

Ya tenemos los ingredientes esenciales para el test de primalidad

- Sólo nos falta implementar algunas funciones auxiliares

Necesitamos desarrollar un algoritmo eficiente para determinar si un número n es la potencia (no trivial) de otro número.

Primero necesitamos una función para calcular n^k

- Usamos el algoritmo de exponenciación rápida sin considerar el módulo

```
EXP(n, k)
    if k = 1 then return n
    else if k es par then
        val := EXP(n, k/2)
        return val · val
    else
        val := EXP(n, (k-1)/2)
        return val · val · n
```

Dado un número natural $n \geq 2$, la siguiente función verifica si existen $m, k \in \mathbb{N}$ tales que $k \geq 2$ y $n = m^k$

```
EsPotencia(n)
    if n ≤ 3 then return no
    else
        for k := 2 to ⌊log₂(n)⌋ do
            if TieneRaízEntera(n, k, 1, n) then return sí
        return no
```

La siguiente función verifica si existe $m \in \{i, \dots, j\}$ tal que $n = m^k$.

- Vale decir, la llamada **TieneRaízEntera**$(n, k, 1, n)$ verifica si n tiene raíz k-ésima entera.

```
TieneRaízEntera(n, k, i, j)
    if i = j then
        if EXP(i, k) = n then return sí
        else return no
    else if i < j then
        p := ⌊(i+j)/2⌋
        val := EXP(p, k)
        if val = n then return sí
        else if val < n then return TieneRaízEntera(n, k, p + 1, j)
        else return TieneRaízEntera(n, k, i, p − 1)
    else return no
```

Consideramos la multiplicación de números enteros como la operación básica a contar.

Tenemos que:

- En el peor caso **EsPotencia**(n) realiza $(\lfloor \log_2 n \rfloor - 1)$ llamadas a la función **TieneRaízEntera**.

- Existe $c \in \mathbb{N}$ tal que la llamada **TieneRaízEntera**$(n, k, 1, n)$ realiza en el peor caso a lo más $c \cdot \log_2 n$ llamadas a la función **EXP**.

- **EXP**(n, k) en el peor caso es $\mathcal{O}(\log_2 k)$.

Concluimos que **EsPotencia**(n) en el peor caso es $\mathcal{O}([\log_2 n]^3)$

Vale decir, **EsPotencia** en el peor caso es de orden polinomial en el tamaño de la entrada.

- Se puede llegar a la misma conclusión si consideramos todas las operaciones realizadas por **EsPotencia**.

El siguiente algoritmo aleatorizado determina si un número entero $n \geq 2$ es primo.

```
TestPrimalidad(n, k)
    if n = 2 then return PRIMO
    else if n es par then return COMPUESTO
    else if EsPotencia(n) then return COMPUESTO
    else
        sea a_1, ..., a_k una secuencia de números elegidos de
                manera uniforme e independiente desde {1, ..., n − 1}
        for i := 1 to k do
            if MCD(a_i, n) > 1 then return COMPUESTO
            else b_i := EXP(a_i, (n−1)/2, n)
        neg := 0
        for i := 1 to k do
            if b_i ≡ −1 mod n then neg := neg + 1
            else if b_i ≢ 1 mod n then return COMPUESTO
        if neg = 0 then return COMPUESTO
        else return PRIMO
```

El algoritmo recibe como entrada un valor entero $k \geq 1$ que es usado para controlar la probabilidad de error.

TestPrimalidad se puede equivocar de dos formas:

1. Suponga que $n \geq 3$ es primo. En este caso **TestPrimalidad** da una respuesta incorrecta si $b_i \equiv_n 1 \ \forall i \in \{1, \ldots, k\}$.

   Dado que $|S_n^+| = |S_n^-| = \frac{n-1}{2}$

   - La probabilidad de que para un número a elegido con distribución uniforme desde $\{1, \ldots, n-1\}$ se tenga que $a^{\frac{n-1}{2}} \equiv_n 1$ es $\frac{1}{2}$.

   Por lo tanto, la probabilidad de que **TestPrimalidad** diga COMPUESTO para $n \geq 3$ primo es $\left(\frac{1}{2}\right)^k$.

2. Suponga que n es compuesto, n es impar y n NO es de la forma m^l con $l \geq 2$.

   a. Si n es par o m es de la forma m^l con $l \geq 2$, entonces **TestPrimalidad** da la respuesta correcta COMPUESTO.

Tenemos entonces que $n = n_1 \cdot n_2$ con $n_1 \geq 3$, $n_2 \geq 3$ y $\gcd(n_1, n_2) = 1$.

Además, debe existir $a \in \{1, \ldots, n-1\}$ tal que $\gcd(a, n) = 1$ y $a^{\frac{n-1}{2}} \equiv_n -1$

   b. Si esto NO es cierto **TestPrimalidad** retorna COMPUESTO, dado que si **TestPrimalidad** logra llegar a la última instrucción **if** entonces neg necesariamente es igual a 0.

Concluimos que $|S_n| \leq \frac{1}{2} \cdot |\mathbb{Z}_n^*|$.

   c. Por la caracterización que dimos de S_n para n compuesto.

Vamos a utilizar este resultado para acotar la probabilidad de error:

$$\Pr\left(\left(\bigwedge_{i=1}^{k} \gcd(a_i, n) = 1 \wedge (b_i \equiv_n 1 \vee b_i \equiv_n -1)\right) \wedge \left(\bigvee_{j=1}^{k} b_j \equiv_n -1\right)\right)$$

Tenemos que:

$$\Pr\left(\left(\bigwedge_{i=1}^{k} \gcd(a_i, n) = 1 \wedge (b_i \equiv_n 1 \vee b_i \equiv_n -1)\right) \wedge \left(\bigvee_{j=1}^{k} b_j \equiv_n -1\right)\right)$$
$$\leq \Pr\left(\left(\bigwedge_{i=1}^{k} \gcd(a_i, n) = 1 \wedge (b_i \equiv_n 1 \vee b_i \equiv_n -1)\right)\right)$$

Por lo tanto, sólo necesitamos una cota superior para la última expresión.

Tenemos que:

$$\Pr\left(\left(\bigwedge_{i=1}^{k} \gcd(a_i, n) = 1 \wedge (b_i \equiv_n 1 \vee b_i \equiv_n -1)\right)\right)$$
$$= \prod_{i=1}^{k} \Pr\big(\gcd(a_i, n) = 1 \wedge (b_i \equiv_n 1 \vee b_i \equiv_n -1)\big)$$

$$= \prod_{i=1}^{k} \Pr\big((b_i \equiv_n 1 \lor b_i \equiv_n -1) \,\big|\, \gcd(a_i, n) = 1\big) \cdot \Pr(\gcd(a_i, n) = 1)$$

$$\leq \prod_{i=1}^{k} \Pr\big((b_i \equiv_n 1 \lor b_i \equiv_n -1) \,\big|\, \gcd(a_i, n) = 1\big)$$

$$= \prod_{i=1}^{k} \Pr(a_i \in S_n \mid a_i \in \mathbb{Z}_n^*) \leq \prod_{i=1}^{k} \frac{1}{2} = \left(\frac{1}{2}\right)^k$$

Concluimos que la probabilidad de que el test diga PRIMO para el valor compuesto n está acotada por $\left(\frac{1}{2}\right)^k$.

En ambos casos (si n es primo o compuesto) la probabilidad de error del algoritmo está acotada por $\left(\frac{1}{2}\right)^k$.

¡Si $k = 100$, está probabilidad está acotada por $\left(\frac{1}{2}\right)^{100} \approx 7.9 \times 10^{-31}$!

Caso promedio de un algoritmo:

Para definir el caso promedio, para cada $n \in \mathbb{N}$ usamos una **variable aleatoria** X_n tal que para cada $w \in \Sigma^n$ se tiene que:

$$X_n(w) = \text{tiempo}_{\mathcal{A}}(w)$$

Para las entradas de largo n, el número de pasos $\mathcal{A}$ en el caso promedio es el **valor esperado** de la variable X_n:

$$E(X_n) = \sum_{w \in \Sigma^n} X_n(w) \cdot \Pr_n(w)$$

Definición: Complejidad en el caso promedio

Decimos que $\mathcal{A}$ en el **caso promedio** es $\mathcal{O}\big(f(n)\big)$ si $E(X_n) \in \mathcal{O}\big(f(n)\big)$

Notación

Las definiciones de peor caso y caso promedio pueden ser modificadas para considerar las notaciones Θ y Ω

3. Simplemente reemplazando $\mathcal{O}(f(n))$ por $\Theta(f(n))$ u $\Omega(f(n))$, respectivamente.

Por ejemplo, decimos que $\mathcal{A}$ en el caso promedio es $\Theta(f(n))$ si $E(X_n) \in \Theta(f(n))$.

Quicksort es un algoritmo de ordenación muy utilizado en la práctica.

La función clave para la definición.
de Quicksort ⟹

```
Partición(L, m, n)
    pivote := L[m]
    i := m
    for j := m + 1 to n do
        if L[j] ≤ pivote then
            i := i + 1
            intercambiar L[i] con L[j]
    intercambiar L[m] con L[i]
    return i
```

La definición de Quicksort ⟹

```
Quicksort(L, m, n)
    if m < n then
        ℓ := Partición(L, m, n)
        Quicksort(L, m, ℓ − 1)
        Quicksort(L, ℓ + 1, n)
```

Nótese que en la definición de **Partición** la lista L es pasado por referencia.

Vamos a contar el **número de comparaciones** al medir la complejidad de Quicksort.

Si **Partición** siempre divide a una lista en dos listas del mismo tamaño, entonces la complejidad de **Quicksort** estaría dada por la siguiente ecuación de recurrencia:

$$T(n)\begin{cases} 1 & n = 0 \\ 2 \cdot T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + c \cdot n & n \geq 1 \end{cases}$$

Esto está en la clase 24, pero no en la 25.

Dado que $c \cdot n \in \Theta(n^{\log_2 2})$, concluimos usando el Teorema Maestro que:

$$T(n) \in \Theta(n \cdot \log_2(n))$$

Teorema

Supongamos que, cada vez que se elige un pivote para Random Quicksort, se elige de manera independiente y uniforme al azar entre todas las posibilidades. Entonces, para cualquier entrada, el número esperado de comparaciones es de:

$$\boxed{2n \cdot \ln n + \ O(n)}$$

Demostración:

Sean $y_1, y_2, \dots, y_n$ los mismos valores que los valores de entrada $x_1, x_2, \dots, x_n$ pero ordenados en orden creciente. Para $i \ < \ j$, sea

$$X_{ij} = \begin{cases} 1 & y_i \text{ e } y_j \text{ se comparan en algún momento del algoritmo} \\ 0 & \text{en otro caso} \end{cases}$$

Entonces, el número total de comparaciones X satisface

$$X = \sum_{i=1}^{n-1} \sum_{j=j+1}^{n} X_{ij}$$

Y

$$E[X] = E\left[\sum_{i=1}^{n-1} \sum_{j=j+1}^{n} X_{ij}\right]$$
$$= \sum_{i=1}^{n-1} \sum_{j=j+1}^{n} E[X_{ij}]$$

Dado que X_{ij} es una variable aleatoria indicadora que solo toma los valores 0 y 1, $E[X_{ij}]$ es igual a la probabilidad de que X_{ij} sea 1. Por lo tanto, todo lo que necesitamos hacer es calcular la probabilidad de que dos elementos y_i e y_j se comparen. Ahora, y_i e y_j se comparan si y solo si y_i o y_j son el primer pivote seleccionado por Random Quicksort del conjunto $Y^{ij} = \{y_i, y_{i+1}, \dots, y_{j-1}, y_j\}$. Esto se debe a que si y_i (o y_j) es el primer pivote seleccionado de este conjunto, entonces y_i e y_j aún deben estar en la misma sublista, y por lo tanto se compararán. De manera similar, si ninguno es el primer pivote de este conjunto, entonces y_i e y_j se separarán en sublistas distintas y no se compararán.

Dado que nuestros pivotes se eligen de manera independiente y uniforme al azar en cada sub-lista, se sigue que la primera vez que se elige un pivote de Y^{ij}, es igualmente probable que sea cualquier elemento de este conjunto. Por lo tanto, la probabilidad de que y_i o y_j sea el primer pivote seleccionado de Y^{ij}, que es la probabilidad de que $X_{ij} = 1$, es $\frac{2}{j-i+1}$. Usando la sustitución $k = j - i + 1$, se obtiene

$$
\begin{aligned}
E[X] &= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1} \\
&= \sum_{i=1}^{n-1} \sum_{k=2}^{n-i+1} \frac{2}{k} \\
&= \sum_{k=2}^{n} \sum_{i=1}^{n+1-k} \frac{2}{k} \\
&= \sum_{k=2}^{n} (n+1-k) \frac{2}{k} \\
&= \left((n+1) \sum_{k=2}^{n} \frac{2}{k} \right) - 2(n-1) \\
&= (2n+2) \sum_{k=1}^{n} \frac{1}{k} - 4n \\
&= \ \ 2 \cdot (n+1) \cdot \left(\sum_{k=1}^{n} \frac{1}{k} \right) - 4 \cdot n
\end{aligned}
$$

Para terminar el cálculo de $E(X_n)$ tenemos que acotar la sumatoria armónica $\sum_{k=1}^{n} \frac{1}{k}$.

Tenemos que:

$$
\sum_{k=2}^{n} \frac{1}{k} \leq \int_{1}^{n} \frac{1}{x}\, dx \leq \sum_{k=1}^{n} \frac{1}{k}
$$

Dado que $\int_{1}^{n} \frac{1}{x}\, dx = \ln(n) - \ln(1) = \ln(n)$, concluimos que:

$$
\ln(x) \leq \sum_{k=1}^{n} \frac{1}{k} \leq \ln(n) = 1
$$

Por lo tanto:

$$
2 \cdot (n+1) \cdot \ln(n) - 4 \cdot n \leq E(X_n) \leq 2 \cdot (n+1) \cdot (\ln(n) + 1) - 4 \cdot n
$$

De lo cual concluimos que $E(X_n) \in \Theta(n \cdot \log_2 n)$

- Vale decir, **Quicksort** en el caso promedio es $\Theta(n \cdot \log_2 n)$.