Analisis de Algoritmos

Introduccion

Un algoritmo es un meotodo o conjunto de instruccione para resolver un problema computacional.

- Reciben **INPUTS** y entregan **OUTPUTS**
- Luego, el algoritmo es un metodo para convertir un **INPUT** valido en un **OUTPUT**.
- En este curso nos exigen:
  - Precicion: instrucciones no ambiguas.
  - Determinismo: instrucciones con comportamientos unicos.
  - Finitud: instrucciones finitas
- Estudiaremso cuando y porque los algoritmos son correctos y ademas la cantidad de recursos computacionales usados.
- Entenderemos los algoritmos y como mejorarlos.
- Usaremos Pseudo-Codigo (IF, While, return , notacion de conjuntos y otras)

Correccion de algoritmos

Definición 0.1. Un algoritmo es **correcto** si para todo INPUT valido, el algoritmos se detiene y produce un OUTPUT correcto.
Un algoritmo es **incorrecto** si no se detiene o produce un OUTPUT incorrecto.

Algoritmos Iterativos

- Se deben demostrar dos cosas:

  1. Correccion Parcial: si el algoritmos se detiene, se cumplen las post condiciones.
  2. Terminacion: El algoritmos se detiene.

- El enfoque principal es en los loops.
- Para demostrar correccion parcial, buscamos una invariante $I(k)$ para los loops: encontrándolo demostramos la corrección del loop inductivamente:
  Sea G condicion de el loop ($While(G)$:),

  - **BI**: $I(0)$ es verdadero
  - **HI**: $\forall k > 0, k \in \mathbb{N}$, si G y $I(k)$ son verdaderos antes de la iteracion, $I(k+1)$ es verdadero despues de la iteracion
  - **Correcion** Inmediatamente despues de terminado el loop (i.e. cuando G es falso), si $k = N$ e $I(N)$ es verdadero, entonces la postcondiciones se cumplen.

- **Terminacion** Debemos demostrar que $\exists k > 0, k \in \mathbb{N}$ para el cual G es falso.

Ejemplo:
Multiplicar dos numeros naturales:

- Pre: $n, m \in \mathbb{N}$
- Post: $p = n * m$

Por demostrar que el algoritmos es correcto:

```
def Mult(n,m):
    p=0
    i=m
    while (i >0):
        p +=n
        i-=1
    return p
```

La invariante $I(k)$ en la k iteracion en este caso es:

$$I : p_k = n_k * (m_k - i_k)$$

En la m iteracion, $i = 0$:

$$p_m = n_m * m_m$$

Demostración. Demostracion por induccion:
BI: Tenemos por pre condicion que $n, m \in \mathbb{N}$. En la iteracion 0,

$$p = 0$$

$$n_0 = n$$

$$m_0 = m$$

$$i_0 = m$$

Por lo tanto $I(0)$,

$$p_0 = n_0 * (m_0 - i_0) \tag{1}$$

$$0 = n * (m - m)$$

Lo que claramente se cumple.
HI: Suponemos que $I(k)$ es cierto para iteracion k.
TI: Por demostrar $I(k+1)$ es cierto,

$$p_{k+1} = n_{k+1} * (m_{k+1} - i_{k+1}) \tag{2}$$

Donde estoy en la iteracion $k+1$? Tenemos por **HI**:

$$p_k = n_k(m_k - i_k)$$

Sumando n en cada lado y como n, m son constantes en todas las iteraciones,

$$p_k + n = n * (m - i_k) + n = n * (m - (i_k - 1))$$

Pero,

$$p_{k+1} = p_k + n$$

$$i_{k+1} = i_k - 1$$

$$m_{k+1} = m$$

$$n_{k+1} = n$$

Reemplazando llegamos a (2) que era lo que queriamos demostar:

$$p_{k+1} = n_{k+1} * (m_{k+1} - i_{k+1}) \tag{3}$$

Terminacion: El loop tiene terminacion porque i se hace 0 en alguna iteracion y el valor de G es falso. $(While(G) = While(i > 0))$ □

Algoritmos Recursivos

A diferencia de los algoritmos iterativos, solo es necesario demostrar la correccion deseada.

Ejemplo:
Encontrar el maximo elemento de un arreglo

- Pre: un arreglo $A = [a_1, a_2, ..., a_n]$, y un natural n (largo de el arreglo). $n \in \mathbb{N} - \{0\}$
- Post: $m = max(A)$

```
1 def max(A,n):
2     if n==1:
3         return A[-1]
4     else:
5         k=max(A[:-1],n-1)
6         if A[-1] >k:
7             return A[-1]
8         else:
9             return k
```

Demostración. Por demostrar que el algoritmo es correcto.
BI: El arreglo de largo 1, tiene un unico elemento que trivialmente es el maximo y el algoritmo retorna este unico elemento y por lo tanto es correcto.
HI: Supondremos que el algoritmo es correcto para un arreglo A de largo n
TI: Por demostrar que dado un arreglo de largo $n+1$ el algoritmo retorna el maximo (es correcto).
La primera condicion claramente no se cumple, por lo que debemos enfocarnos desde la linea 4 en adelante.
En la linea 5, dado que tenemos un arreglo de largo $n+1$ entonces:

```
1   k=max(A[:-1],n)
```

Por lo que sabemos que k es el máximo de el arreglo de largo n por **HI**.
Ahora tenemos dos casos:

- Caso 1: Se cumple el primer if,

  $$A[-1] > k$$

  Implica que el ultimo elemento de el arreglo de largo $n+1$ es mayor a el máximo de el arreglo de largo n. Entonces efectivamente este es el nuevo maximo.

- Caso 2: Si la condicion no se cumple, es claro que el maximo de el arreglo de largo $n+1$ sigue siendo k. Por lo que es correcto retornar este mismo valor.

□

Complejidad de Algoritmos

- Comparar tiempo de ejecucion de algoritmos
- Comportamiento conforme crece al tamaño del input
- Independiente del lenguaje, hardware, etc ...

Definición 0.2.

$$f : \mathbb{N} \to \mathbb{R}^+$$

$$O(f) = \{g : \mathbb{N} \to \mathbb{R}^+ | (\exists c \in \mathbb{R}^+)(\exists n_0 \in \mathbb{N})(\forall n \geq n_0)(g(n) \leq c * f(n))\}$$

Desde n_0, la funcion $g(n)$ es por debajo de la funcion $f(n) * c$.
Se dice que $g(n) \in O(f)$; $g(n)$ es O de f ; $g(n)$ es a lo misma de orden que $f(n)$.

Definición 0.3.

$$\Omega(f) = \{g : \mathbb{N} \to \mathbb{R}^+ | (\exists c \in \mathbb{R}^+)(\exists n_0 \in \mathbb{N})(\forall n \geq n_0)(g(n) \geq c * f(n))\}$$

Una funcion $g(n)$ esta acotada por abajo por $f(n) * c$.
Se dice que $g(n) \in \Omega(f)$.

Definición 0.4.

$$\Theta(f) = O(f) \cap \Omega(f)$$

$g \in \Theta$ es exactamente de orden f

Ejemplo:
Demuestre que $g \in \Theta(f)$ si y solo si existen $c, d \in \mathbb{R}^+$ y $n_0 \in \mathbb{N}$ tales que,

$$\forall n \geq n_0 . \big[c * f(n) \leq g(n) \leq d * f(n)\big]$$

Demostración. Sabemos que $g(n) \in \Theta(f)$,

$$g(n) \in \Theta(f) \to g(n) \in \Omega(f) \wedge g(n) \in O(f)$$

Por definicion de los conjuntos, tenemoes que:
Para $O(f)$:

$$(\exists c \in \mathbb{R}^+)(\exists n_1 \in \mathbb{N})(\forall n \geq n_1)\big[g(n) \leq c * f(n)\big]$$

Para $\Omega(f)$:

$$(\exists d \in \mathbb{R}^+)(\exists n_2 \in \mathbb{N})(\forall n \geq n_2)\big[d * f(n) \leq g(n)\big]$$

Si tomamos un $n_0 = Max\{n_1, n_2\}$, entonces se debe cumplir que:

$$\forall n \geq n_0 . \big[c * f(n) \leq g(n) \leq d * f(n)\big]$$

Nota: Esto se puede afirmar ya que el maximo entre n_1 y n_2 seria el primer $n \in \mathbb{N}$ que cumple estar acotado por los dos conjuntos. Antes de n_0, puede pasar cualquier cosa con $g(n)$ por lo que queremos tomar en cuenta solo a partir de n_0. □

Ejemplo:
$60n^2 \in \Theta(n^2)$

$$60n^2 \in \Theta(n^2) \leftrightarrow 60n^2 \in O(n^2) \wedge 60n^2 \in \Omega(n^2)$$

$$n_0 = 0, c = 60 \rightarrow \forall n \geq 0. \left[60n^2 \leq 60n^2\right] \rightarrow 60n^2 \in O(n^2)$$
$$n_0 = 0, c = 1 \rightarrow \forall n \geq 0. \left[60n^2 \geq n^2\right] \rightarrow 60n^2 \in \Omega(n^2)$$

Por lo tanto,

$$60n^2 \in \Theta(n^2)$$

Nota: Esto implica que las constantes no influyen en la complejidad de una funcion
Ejemplo:

$$60n^2 + 5n + 1 \in \Theta(n^2)$$
$$n_0 = 1, c = 66 \rightarrow \forall n \geq 1. \left[60n^2 + 5n + 1 \leq 60n^2 + 5n + n^2 \leq 66n^2\right]$$

Por lo tanto,

$$60n^2 + 5n + 1 \in O(n^2)$$
$$n_0 = 0, c = 60 \rightarrow \forall n \geq 0. \left[60n^2 + 5n1 \geq 60n^2\right]$$

Por lo tanto,

$$60n^2 + 5n + 1 \in \Omega(n^2)$$

Podemos concluir que:

$$60n^2 + 5n + 1 \in \Theta(n^2)$$

Nota: Esto implica que el mayor exponente de un polinomio es el que determina la complejidad
Ejemplo:

$$log_2(n) \in \Theta(log_3(n))$$
$$x = log_2(n) \rightarrow 2^x = n$$
$$y = log_3(n) \rightarrow 3^y = n$$
$$3^y = 2^x$$

Aplicando log_2

$$x = log_2(3^y)$$
$$x = y * log_2(3)$$

Reemplazando x e y originales,

$$log_2(n) = log_2(3) * log_3(n)$$

Entonces,

$$c = log_2(3), n_0 = 1 \rightarrow \forall n \geq 1. \left[log_2(n) \leq log_2(3) * log_3(n)\right] \rightarrow log_2(n) \in O(log_3(n))$$
$$c = log_2(3), n_0 = 1 \rightarrow \forall n \geq 1. \left[log_2(n) \geq log_2(3) * log_3(n)\right] \rightarrow log_2(n) \in \Omega(log_3(n))$$

Por lo tanto,

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

Nota: Esto significa que nos podemos independizar de las bases.
Teorema 0.1. *Si* $f : \mathbb{N} \rightarrow \mathbb{R}^+$ *es tal que,*

$$f(n) = a_k \cdot n^k + a_{k-1} \cdot n^{k-1} + ... + a_2 \cdot n^2 + a_1 \cdot n + a_0,\ (a_i \in \mathbb{R}^+) \wedge (a_k > 0)$$

Entonces,

$$f(n) \in \Theta(n^k)$$

Demostración. Por demostrar que $f(n) \in \Theta(n^k)$,
Primero que $f(n) \in O(n^k)$,

$$\exists c \in \mathbb{R}^+ . \forall n \geq n_0 . \big[f(n) \leq c \cdot n^k \big]$$

Si tomamos $c = \sum_{i=0}^{k} a_i$ y $n_0 = 1$,

$$f(n) \leq c \cdot n^k$$

$$a_k \cdot n^k + a_{k-1} \cdot n^{k-1} + ... + a_2 \cdot n^2 + a_1 \cdot n + a_0 \leq \sum_{i=0}^{k} a_i \cdot n^k$$

$$\sum_{i=0}^{k} a_i \cdot n^i \leq \sum_{i=0}^{k} a_i \cdot n^k$$

Expandiendo la sumatoria,

$$a_i \cdot n^i \leq a_i \cdot n^k, \ \forall i \in \{0, 1, ..., k\}$$

Lo que efectivamente es verdadero ya que,

$$a_0 \leq a_0 \cdot n^k$$

$$a_1 \cdot n \leq a_1 \cdot n^k$$

$$\vdots$$

$$a_{k-1} \cdot n^{k-1} \leq a_{k-1} \cdot n^k$$

$$a_k \cdot n^k \leq a_k \cdot n^k$$

Ahora que $f(n) \in \Omega(n^k)$,

$$\exists c \in \mathbb{R}^+ . \forall n \geq n_0 . \big[f(n) \geq c \cdot n^k \big]$$

Si elegimos $c = a_k$ y $n_0 = 0$,

$$f(n) \geq c \cdot n^k$$

$$a_k \cdot n^k + a_{k-1} \cdot n^{k-1} + ... + a_2 \cdot n^2 + a_1 \cdot n + a_0 \geq a_k \cdot n^k$$

Claramente la desigualdad se cumple porque dada la definición de $f(n)$ $(f : \mathbb{N} \to \mathbb{R}^+)$,

$$a_{k-1} \cdot n^{k-1} + ... + a_2 \cdot n^2 + a_1 \cdot n + a_0 \geq 0$$

Entonces,

$$f(n) \in \Theta(n^k)$$

□

Teorema 0.2. *Si $f : \mathbb{N} \to \mathbb{R}^+$ es tal que,*

$$f(n) = log_a(n), a > 1$$

Entonces,

$$\forall b > 1 . \big[f(n) \in \Theta(log_b(n)) \big]$$

Demostración.

$$x = log_a(n) \rightarrow a^x = n$$
$$y = log_b(n) \rightarrow b^y = n$$
$$b^y = a^x$$

Aplicando $log_2()$

$$x = log_a(b^y)$$
$$x = y * log_a(b)$$

Reemplazando los valores originales de x e y,

$$log_a(n) = log_a(b) * log_b(n)$$

Entonces,

$$\forall n \geq 1. \big[log_a(n) \leq log_a(b) * log_b(n)\big] \rightarrow log_a(n) \in O(log_b(n))$$

De la misma manera,

$$\forall n \geq 1. \big[log_a(n) \geq log_a(b) * log_b(n)\big] \rightarrow log_a(n) \in \Omega(log_b(n))$$

Entonces podemos conlcuir,

$$log_a(n) \in \Theta(log_b(n))$$

□

Tabla de notaciones (en orden)

Notación	Nombre
$\Theta(1)$	Constante
$\Theta(\log n)$	Logarítmico
$\Theta(n)$	Lineal
$\Theta(n \log n)$	$n \log n$
$\Theta(n^2)$	Cuadrático
$\Theta(n^3)$	Cúbico
$\Theta(n^k)$	Polinomial
$\Theta(m^n)$	Exponencial
$\Theta(n!)$	Factorial

con $k \geq 0, m \geq 2$.

Nota: Hasta el orden polinomial, se dicen que son eficientes

Definición 0.5. Funcion $T(n)$:

- Queremos encontrar una función $T(n)$ que modele el tiempo de ejecuccion de un algoritmo.
- Donde n es el tamaño del input.Queremos determinar el orden de T(n).
- Si no podemos encontrar el Θ estamos satisfechos con el O porque nos acota por arriba.

Ejemplo:

```
n=int(input())
x=0
for i in range(1,n):
    for j in range(1,i):
        x +=1
```

Fijarse en el paso 4.

$$n = 1 \rightarrow 1$$
$$n = 2 \rightarrow 1 + 2$$
$$\vdots$$
$$\rightarrow 1 + 2 + ... + n = \frac{n(n+1)}{2}$$

Entonces,

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

Como en los ejemplos pasados, el orden mayor es el que acota la funcion.

Ejemplo:

Input n

```
j=n
x=0
while j >= 1:
  for i in range(1,j):
      x = x +1
  j=⌊j/2⌋
```

Suponiendo que el loop se ejecuta k veces, $k \geq 1$

$$n + \frac{n}{2} + \frac{n}{4} + ... + \frac{n}{2^{k-1}} = n \cdot \sum_{i=0}^{k-1} \left(\frac{1}{2}\right)^i = n\frac{1 - \frac{1}{2}^k}{1 - \frac{1}{2}} = 2n(1 - \frac{1}{2}^k)$$

Lo ultimo es ≥ 0, y

$$\in \Theta(n)$$

Deducción serie geométrica:

Sea S el valor de la suma,

$$S = n + \frac{n}{2} + \frac{n}{2^2} \cdot n + ... + + \frac{n}{2^{k-1}}$$

Multiplicando por $\frac{1}{2}$,

$$\frac{S}{2} = \frac{n}{2} + \frac{n}{2^2} \cdot n + ... + \frac{n}{2^k}$$

Entonces,

$$S - \frac{S}{2} = n - \frac{n}{2^k}$$

Ordenando,

$$S = n\frac{1 - \frac{1}{2^k}}{1 - \frac{1}{2}}$$

Ejemplo Busqueda:

A: Arreglo $[a_0, ..., a_{n-1}]$
n: Natural ≥ 1
k: Entero

```
1 def Busqueda(A,n,k):
2   for i in range(0,n-1):
3     if A[i]=k:
4         return i
5   return -1
```

- En este ejemplo se cuenta el **IF**, ya que es el unico que ser repite.

- n es el largo de el arreglo , queremos $T(n)$
- Buscamos el tiempo de ejecutarse el algoritmo en un largo n
- En el **mejor caso**, $k = a_0$ entonces $T(n) = 1 \in \Theta(1)$.

- Ahora, en el **peor caso** $k = a_{n-1}$ entonces $T(n) = n \in \Theta(n)$.

En resumen:

$$T(n) = \begin{cases} \Theta(1) & k = a_0 \\ \Theta(n) & k = a_{n-1} \vee k \notin A \end{cases}$$

Ejemplo Insert-Sort:

```
1   for i=2 to n:
2     t=a_i
3     j=i-1
4     while a_j > t and j
```

- Mejor caso: arreglo ya esta ordenado, entonces el loop en linea 5 no se ejecuta nunca.

$$T(n) = n - 1 \in \Theta(n)$$

- Peor Caso: arreglo esta ordenado al revez. El loop en la instruccion 4, se ejcuta hasta que $j = 0$ .Entonces $i - 1$ veces entre 2 y n

$$T(n) = 1 + 2 + ... + n - 1 = \frac{(n-1)n}{2} \in \Theta(n^2)$$

$$T(n) = \begin{cases} \Theta(n) \\ \Theta(n^2) \end{cases}$$

- Nota: El enfoque esta en la comparacion porque en general lo demas es constante y no cuenta en la complejidad

Algoritmos Recursivos

EjemplO Busqueda Binaria:

```
  BinarySearch(a, A, i, j)
if i > j then
    return -1
else if i = j then
    if A[i] = a then
        return i
    else
        return -1
    end if
else
    p = ⌊(i+j)/2⌋
    if A[p] < a then
        return BinarySearch(a, A, p + 1, j)
    else if A[p] > a then
        return BinarySearch(a, A, i, p -1)
    else
        return p
    end if
end if
```

$$T(n) = \begin{cases} 3 & n = 1 \\ 4 + T(\lfloor \frac{n}{2} \rfloor) & n > 1 \end{cases}$$

Como se resuelven estas equaciones? Con sustituciones
Sea $n = 2^k$,

$$T(2^k) = 3, k = 0$$
$$T(2^k) = 4 + T(2^{k-1}), k > 0$$
$$T(2^k) = T(2^{k-1}) + 4$$
$$T(2^k) = (T(2^{k-2}) + 4) + 4$$
$$\vdots$$

Tomo un i tal que $k - 1 \geq 0$:

$$T(2^k) = T(2^{k-i}) + 4 * i$$

Cuando $k = i$:

$$T(2^k) = T(1) + 4k = 3 + 4k$$

Sustituendo de vuelta:

$$T(n) = 3 + 4 * log_2(n)$$

Nota: Con n potencia de 2.

Definición 0.6. Notacion Asintotica Condicional:
Sea $P \subseteq \mathbb{N}$,

$$O(f|P) = \{g : \mathbb{N} \to \mathbb{R}^+ | (\exists c \in \mathbb{R}^+)(\exists n_0 \in \mathbb{N})(\forall n \geq n_0)(n \in P \to g(n) \leq c * f(n))\}$$

$$\Omega(f|P) = \{g : \mathbb{N} \to \mathbb{R}^+ | (\exists d \in \mathbb{R}^+)(\exists n_0 \in \mathbb{N})(\forall n \geq n_0)(n \in P \to g(n) \geq d * f(n))\}$$

$$\Theta(f|P) = O(f|P) \cap \Omega(f|P)$$

Definición 0.7. Conjunto $POTENCIA_2$

$$POTENCIA_2 = \{2^i | i \in \mathbb{N}\}$$

Ejemplo Busqueda Binaria (cont.):
Concluimos que,

$$T(n) = 3 + 4 * log_2(n)$$

Pero esto solo para n que pertenecen a el conjunto potencia de 2. Por lo tanto,

$$T(n) \in \Theta(log_2(n)|POTENCIA_2)$$

Demostración. Queremos demostrar que,

$$T(n) \in \Theta(log_2(n)|POTENCIA_2) \to T(n) \in \Theta(log_2(n))$$

Por lo menos, que:

$$T(n) \in O(log_2(n))$$

□

Por definicion de $O(log_2(n)|POTENCIA_2)$:

$$(\exists c \in \mathbb{R}^+)(\exists n_0 \in \mathbb{N})(\forall n \geq n_0)(T(n) \leq c * log_2(n))$$

$$T(1) = 3$$

$$T(2) = 4 + T(1) = 7$$

$$T(3) = 4 + T(2) = 11$$

Por lo tanto, nos srive $n_0 = 2$ y $c = 7$.

Demostración. Por induccion fuerte

$$\forall n \geq 2.\big[T(n) \leq 7 * log_2(n)\big]$$

BI:

$$T(2) = 7 = 7 * log_2(2)$$
$$T(3) = 7 \leq 7 * log_2(3)$$

HI:
Sup que con $n \geq 4$, $\forall k \in \{2, ..., n-1\}$ se cumple que $T(k) \leq 7 * log_2(k)$
TI:
Por Demostrar que:

$$T(n) \leq 7 * log_2(n)$$

Como $n > 1$:

$$T(n) = T(\left\lfloor \frac{n}{2} \right\rfloor) + 4$$

Por HI,

$$T(n) \leq 7 * log_2(\left\lfloor \frac{n}{2} \right\rfloor) + 4$$

Como el log es creciente y monotono,

$$\begin{aligned} T(n) &\leq 7 * log_2(\frac{n}{2}) + 4 \\ &= 7 * log_2(n) - 7 + 4 \\ &= 7 * log_2(n) - 3 \\ &\leq 7 * log_2(n) \end{aligned}$$

Por lo tanto,

$$T(n) = O(log_2(n))$$

□

Definición 0.8. Una funcion $f : \mathbb{N} \to \mathbb{R}^+$ es **asintoticamente no decreciente** o **creciente** si:

$$(\exists n_0 \in \mathbb{N})(\forall n \geq n_0)(f(n) \leq f(n+1))$$

Ejemplos :
Las siguientes son asintoticamente no decrecientes:

$$log_2(n), n, n^k, 2^n$$

Definición 0.9. Dado un natural $b > 0$, una funcion $f : \mathbb{N} \to \mathbb{R}^+$, **b-armonica** si $f(b \cdot n) \in O(f)$. Formalmente:

$$(\exists c \in \mathbb{R}^+)(\exists n_0 \in \mathbb{N})(\forall n \geq n_0)\big[f(b \cdot n) \leq c * f(n)\big]$$

Ejemplos :
Las siguientes son b-armonicas para cualquier b.

$$log_2(n), n, n^k$$

Notar que 2^n no es b-armonica ya que $2^{2n} \notin O(2^n)$
Teorema 0.3. *Si f, g son asintoticamente no decrecientes, g es b-armonica y $f \in O(g|POTENCIA_b)$ entonces $f \in O(g)$.*

Demostración. Como f es asintoticamente no decreciente:

$$(\exists n \in \mathbb{N})(\exists n_0 \in \mathbb{N})(\forall n \geq n_0)(f(n) \leq f(n+1)) \tag{4}$$

Como g es asintoticamente no decreciente:

$$(\exists n \in \mathbb{N})(\exists n_0 \in \mathbb{N})(\forall n \geq n_1)(g(n) \leq g(n+1)) \tag{5}$$

Como $f \in O(g|Potenica_b)$:

$$(\exists c \in \mathbb{R}^+)(\exists n_2 \in \mathbb{N})(\forall n \geq n_2)(n \in POTENCIA_b \rightarrow f(n) \leq c \cdot g(n)) \tag{6}$$

Como g es b-armonica: $g(b * n) \in O(g)$:

$$(\exists d \in \mathbb{R}^+)(\exists n_3 \in \mathbb{N})(\forall n \geq n_3)(g(b * n) \leq d \cdot g(n)) \tag{7}$$

Sea:

$$n_4 = Max\{n_0, n_1, n_2, n_3, 1\}$$

Nota: El 1 es para poder hacer el siguiente paso:

Suponemos $n \geq n_4$:

Como $n \geq 1$, existe $k \geq 0$ tal que

$$b^k \leq n < b^{k+1} \tag{8}$$

De (4) y (8):

$$f(n) \leq f(b^{k+1}) \tag{9}$$

Como $n \geq n_2$ se cumple (6) y entonces:

$$f(b^{k+1}) \leq c \cdot g(b^{k+1}) \tag{10}$$

De (8) multiplicando por b:

$$b^{k+1} \leq b \cdot n \tag{11}$$

Ahora como $n \geq n_1$, se cumple (5):

$$g(b^{k+1}) \leq g(b \cdot n) \tag{12}$$

Como $n \geq n_3$ y q es b-armonica (7):

$$g(b * n) \leq d \cdot g(n) \tag{13}$$

Juntando todo:

$$f(n) \leq f(b^{k1}) \leq c \cdot g(b^{k+1}) \leq c \cdot g(b \cdot n) \leq c\dot{d} \cdot g(n)$$

Entonces,

$$\forall n \geq n_4, f(n) \leq e \cdot g(n),\ e = c \cdot d \in \mathbb{R}^+$$

Podemos concluir que $f \in O(g)$ □

Teorema Maestro

Grafos

Definiciones basicas

Definición 0.10. Grafo Un **grafo** se denota como $G = (V, E)$, V es el conjunto de **nodos** y E es la relación entre ellos ($E \subseteq V \times V$) denominado **aristas**.

Definición 0.11. Rulo o Loop Es una arista $(x, y) \in E$ tal que $x = y$. El conjunto de todos los rulos seria:

$$C_r = \{(x, y) \in E | x = y\}$$

Donde $C_r \subseteq E$.

Definición 0.12. Aristas Paralelas Dos aristas $(x, y) \in E$ y $(z, w) \in E$ son paralelas si $x = w$ e $y = z$. Es decir, si conectan a los mismos vértices. Denominado por "$\|$"

$$(x, y) \parallel (z, w) \leftrightarrow x = w \wedge y = z$$

Definición 0.13. Grafo no dirigido Un grafo $G = (V, E)$ es **no dirigido**, toda arista tiene una arista paralela.

$$\forall x, y \in V. \big[\exists (z, w) \in E \text{ tal que } (x, y) \parallel (z, w)\big]$$

O de una manera alternativa, el conjunto E debe ser <u>simétrico</u> (existen direcciones en ambos lados).

$$\forall x, y \in V. \big[(x, y) \in E \rightarrow (y, x) \in E\big]$$

Definición 0.14. Grafo Simple Un grafo es **simple**, si no tiene rulos.
Alternativamente, se puede decir simple si es E es <u>irrefleja</u>:

$$\forall x \in V. \big[(x, x) \notin E\big]$$

Definición 0.15. Grafo General:
Se usaran grafos $G = (V, E)$:

- Simples (E Irrefleja)
- No dirigidos (E Simetrica)
- No vacios
- Finitos

Es decir:

$$|V| = n,\ n \in \mathbb{N} - \{0\}$$

Definición 0.16. Adyacentes Dado un grafo $G = (V, E)$, dos vertices $x, y \in V$ son **adyacentes** o **vecinos** si $(x, y) \in E$

Definición 0.17. Dos grafos $G_1 = (V_1, E_1)$ y $G_2 = (V_2, E_2)$ son **isomorfos** si existe una funcion biyectiva $f : V_1 \to V_2$ tal que:

$$(x, y) \in E_1 \leftrightarrow \big(f(x), f(y)\big) \in E_2$$

Se dice que f es un isomorfismo entre G_1 y G_2.

Notacion: $G_1 \cong G_2$

Teorema 0.4. $\cong$ *es una relacion de equivalencia*

Demostración. Sean $G_0 = (V_0, E_0)$,$G_1 = (V_1, E_1)$ y $G_2 = (V_2, E_2)$ grafos arbitrarios pertenecientes a un conjunto A.
Refleja:
Sea $G = (V, E)$, Queremos demostrar que:

$$\forall G_0 \big[G_0 \cong G_0\big]$$

Por definición se isomorfismo, debemos demostrar que existe una funcion biyectiva. Es claro que la función identidad nos sirve para esta vaso ya que es $f : V_0 \to V_0$ biyectiva.
Simetrica:
Queremos demostrar que:

$$\forall G_1, G_2. \big[G_1 \cong G_2 \to G_2 \cong G_1\big]$$

Sabemos que $G_1 \cong G_2$, por lo tanto existe una función $f : V_1 \to V_2$ biyectiva. Sabemos que las funciones biyectivas son invertibles y su inversa tambien es biyectiva. Por lo tanto encontramos una funcion $f^-1 : V_2 \to V_1$ biyectiva, por lo tanto $G_2 \cong G_1$.
Transitiva:
Queremos demostrar que:

$$\forall G_0, G_1, G_2. \big[G_0 \cong G_1 \wedge G_1 \cong G_2 \to G_0 \cong G_2\big]$$

Sabemos que existen $f_0 : V_0 \to V_1$ y $f_1 : V_1 \to V_2$ funciones biyectivas.
Definimos $h : V_0 \to V_2$ como,

$$h = (f_0 \circ f_1)$$

Sabemos que composicion de funciones biyectiva es biyectiva, por lo tanto $G_0 \cong G_2$. Podemos concluir que $\cong$ es una relacion de equivalencia. □

Clases de grafos

Ahora sabemos que $\cong$ es una relacion de equivalencia, por lo que podemos construir tomar las clases de equivalencia de la relación $\cong$.
Recordatorio: Clases de equivalencia:
Sea $\sim$ relacion sobre un conjunto A:

$$[x]_\sim = \{y \in A | x \sim y\}$$

Para este caso,

$$[G]_\cong = \{G_1 \in A | G \cong G_1\}$$

Donde G_1 es un grafo cualquiera $\in A$

Definición 0.18. Camino (Path):
Informal: Un camino es un grafo cuyos vértices pueden dibujarse en una línea tal que dos vértices son adyacentes si y sólo si aparecen consecutivos en la línea.

Formal: Considere un grafo $G_n^P = (V_n^P, E_n^P)$ donde $V_n^P = \{v_1, ..., v_n\}$ y
$E_n^P = \{(v_i, v_j) | i \in \{1, ..., n-1\} \wedge j = i+1\}$
Un **camino** de n vertices es un grafo isomorfo a G_n^P

Definición 0.19. Conjunto de caminos con n vertices

$$P_n : [G_n^P]_{\cong}$$

Definición 0.20. Ciclo:
Informal: Un ciclo es un grafo cuyos vértices pueden dibujarse en un círculo tal que dos vértices son adyacentes si y sólo si aparecen consecutivos en él.

Formal: Considere un grafo $G_n^C = (V_n^C, E_n^C)$ donde $V_n^C = \{v_1, ..., v_n\}$ y
$E_n^C = \{(v_i, v_j) | i \in \{1, ..., n-1\} \wedge j = i+1\} \cup \{(v_n, v_1)\}$
Un **camino** de n vertices es un grafo isomorfo a G_n^C

Definición 0.21. Conjunto de ciclos con n vertices

$$C_n : [G_n^C]_{\cong}$$

Definición 0.22. Grafo Completo:
Un grafo **completo** es un grafo en el que todos los pares de vértices son adyacentes.

Considere un grafo $G_n^K = (V_n^K, E_n^K)$ donde $V_n^K = \{v_1, ...v_n\}$ y
$E_n^K = \{(v_i, v_j) | i \neq j \wedge i \in \{1, ..., n\}\}$.
Un grafo **completo** de n vertices es un grafo isomorfo a G_n^K

Definición 0.23. Conjunto de grafos completos con n vertices

$$K_n : [G_n^K]_{\cong}$$

Definición 0.24. Grafo bipartito: Un grafo $G = (V, E)$ se dice **bipartito** si V se puede particionar en dos conjuntos no vacíos V_1 y V_2 tales que:

$$\forall (x, y) \in E. \big[(x \in V_1 \wedge y \in V_2) \vee (x \in V_2 \wedge y \in V_1)\big]$$

Es decir:

- $V = V_1 \cup V_2$
- $V_1 \cap V_2 = \varnothing$
- Cada arista une a dos vértices en conjuntos distintos de la partición.

Definición 0.25. Grafo bipartito completo:
Un grafo **bipartito completo** es un grafo bipartito en que cada vértice es adyacente a todos los de la otra partición.
A la clase de equivalencia de los grafos bipartitos completos la llamaremos $K_{n,m}$, donde n y m son los tamaños de las particiones.

Mas clases

Dado un grafo $G = (V_G, E_G)$,

Definición 0.26. Subgrafo:
Un grafo $H = (V_H, E_H)$ es un **subgrafo** de G ($H \subseteq G$) si $V_H \subseteq V_G$, $E_H \subseteq E_G$ y E_H solo contiene aristas entre vértices de V_H.

Definición 0.27. Clique:
Un **clique** es un subgrafo completo de G.
Es decir, un clique en G es un conjunto de vertices $K \subseteq V_G$ tal que:

$$\forall u, v \in K. \big[(u, v) \in E_G\big]$$

Definición 0.28. Conjunto Independiente:
Un conjunto independiente en G es un conjunto de vértices $K \subseteq V_G$ tal que:

$$\forall u, v \in K. \big[(u, v) \notin E_g\big]$$

Definición 0.29. Grafo complemento:
El **complemento** de G es el grafo $\overline{G} = (V_G, \overline{E_G})$ si,

$$\forall u, v \in V_G. \big[(u, v) \in E_G \leftrightarrow (u, v) \notin \overline{E_G}\big]$$

Definición 0.30. Grafo autocomplementario:
Un grafo se dice **autocomplementario** si $G \cong \overline{G}$

Teorema 0.5. *Dado un grafo $G = (V, E)$, un conjunto $V' \subseteq V$ es un clique en G si y sólo si es un conjunto independiente en $\overline{G}$.*

Demostración. :

($\Rightarrow$)
Supondremos que V' es un clique en G, por definicion de clique (tomando u, v arbitrarios):

$$\forall u, v \in V. \big[(u, v) \in E]$$

Luego por complemento se debe cumplir:

$$(u, v) \in E \leftrightarrow (u, v) \notin \overline{E}$$

Ya que tomamos u, v arbitrarios, podemos generalizar que:

$$\forall u, v \in V. \big[(u, v) \notin \overline{E}\big]$$

Que es por definicion un conjunto independiente en $\overline{G}$.
($\Leftarrow$) Analoga.

□

Definicion Matriz de Adyacencia

Definicion Matriz de Incidencia

Dado un Grafo G y un vértice v,

Definición 0.31. Grado de un vértice:

El **grado** de un vertice v es la cantidad de aristas que inciden en v.
Denotado como $\delta_G(v)$
Definición 0.32. Vecindad de un vértice:

La **vecindad** de un vertice v es el conjunto de vecinos de v.

$$N_G(v) = \{u | (v, u) \in E\}$$

En un grafo simple, $\delta_G(v) = |N_G(v)|$
Teorema 0.6. *Handshaking lemma:*
Si $G = (V, E)$ es un grafo sin rulos, entonces:

$$\sum_{v \in V} \delta_G(v) = 2|E|$$

Demostración. Por inducción en la cantidad de aristas,
BI:
Si $|E| = 0$, entonces el grado de cada vertice es 0.

$$\sum_{v \in V} \delta_G(v) = 2|E| = 2|0| = 0$$

HI:
Suponemos que se cumple para un grafo con n aristas que:

$$\sum_{v \in V} \delta_G(v) = 2 \cdot |E| = 2 \cdot n$$

TI:
Por demostrar que se cumple para $n + 1$ aristas, Sean $u, v \in V$ vertices arbitrarios, si se agrega una arista e entre los 2 (es claro que como no tiene rulos, $u \neq v$). Al agregar una arista, se aumenta el grado de cada vértice en 1.
Por lo tanto, al agregar la arista

$$\sum_{v \in V} \delta_G(v) + 2$$

Sea $E' = E \cup \{e\}$, podemos ver que:

$$|E'| = |E| + 1$$

Por lo tanto tenemos que,

$$\sum_{v \in V} \delta_G(v) + 2 = 2 \cdot |E'| = 2 \cdot (|E| + 1)$$

Por lo que,

$$\sum_{v \in V} \delta_G(v) = 2|E'|$$

Por lo tanto, si retiro la arista que agrege volvería a la HI. □

Corolario: En un grafo sin rulos siempre hay una cantidad par de vértices de grado impar.

Demostración. Sea,

$$P = \sum_{u \in V | \delta(u) \in \mathbb{P}} \delta(u)$$

$$I = \sum_{v \in V | \delta(v) \in \mathbb{I}} \delta(v)$$

Es claro que P es par.
Por "Handshake Lemma", la suma de los grados de los vertices es **2** veces las cantidad de aristas. Por lo tanto $I + P$ es necesariamente par. Con esto, se debe cumplir que I es par (suma de par y impar es impar). □

Definición 0.33. Caminata y Caminata Cerrada:
Una **caminata** C en un grafo G es una secuencia de vértices y aristas:

$$C = (v_0, e_1, v_1, e_2, v_2, ..., e_k v_k)$$

tal que la arista ei conecta a los vértices $v_i - 1$ y v_i, con i entre 1 y k.
Una **caminata cerrada** C' en un grafo es una caminata que empieza y termina en el mismo vértice: $v_0 = v_k = v$

$$C' = (v, e_1, v_1, e_2, v_2, ..., e_k, v)$$

Nota: En una caminata se pueden repetir aristas

Definición 0.34. Camino y Ciclo:
Un **camino** en un grafo es una caminata en la que no se repiten aristas.
Un **ciclo** en un grafo es una caminata cerrada en la que no se repiten aristas.

Definición 0.35. El **largo** de una caminata, camino o ciclo es la cantidad de aristas que lo componen. Si está compuesto por un único vértice (sin aristas), diremos que tiene largo 0

Definición 0.36. Conectados:
Dos vértices x e y se dicen **conectados** si existe un camino en G que empieza en x y termina en y

Ejemplo :
Muestre que "estar conectados" es una relación de equivalencia (")

$$x - y \leftrightarrow x \text{ esta conectado con } y$$

Demostración.

- **Refleja:** Sea $v \in V$, basta consideras $v - v$ de largo 0
- **Simetrica:** Supongamos que existe $x - y$, luego es trivial que existe $y - x$, pues G es no dirigido.
- **Transitiva:** Supongamos $x - y$, $y - z$. Por demostra que $x - z$.
  Sea,

  $$x - y = x, a_1, ..., a_n, y$$

  $$y - z = y, b_1, ..., b_n, z$$

  Luego:

  $$x - z = x, a_1, ..., a_n, y, b_1, ..., b_n, z$$

Pero como es un camino se puede repetir aristas. Pero sin embargo, basta con tomar un camino C intermedio tal que se cruzen los camino $x - y$ e $y - z$ y tomo:

$$x - z = x, a_1, ..., C, ..., b_n, z$$

□

Definición 0.37. Componente Conexa:
Dado un vértice v de un grafo G, su clase de equivalencia bajo la relación "estar conectados" es una **componente conexa** de G.
En general, diremos que la componente conexa también contiene a las aristas entre los vértices de ella.

Definición 0.38. Grafo Conexo:
Un grafo G se dice **conexo** si todo par de vértices $x, y \in V$ está conectado. En otro caso, G es **disconexo**. Es decir, G tiene sólo **una** componente conexa.

Teorema 0.7. *Un grafo G con n vértices y k aristas tiene al menos $n - k$ componentes conexas.*

Demostración. Un grafo G con n vertices puede tener como maximo n componentes conexas, cuando no tiene ninguna arista, cada nueva arista que se le agregue puede reducir la cantidad de componentes a lo mas en 1, por lo que luego de agregar k aristas la cantidad de componentes se ha reducido como minimo a $n-k$, por lo que la cantidad de componentes siempre se mantiene mayor o igual a $n - k$ □

Nota: El anterior teorema implica que si se quiere un grafo conexo de n vertices, entonces al menos $n - 1$ aristas son necesarias.

Definición 0.39. Arista de corte:
Una **arista de corte** en un grafo G es una arista tal que al eliminarla aumenta la cantidad de componentes conexas de G.

Definición 0.40. Vertice de corte:
Un **vértice de corte** en un grafo G es una arista tal que al eliminarlo (junto con todas sus aristas incidentes) aumenta la cantidad de componentes conexas de G.

Teorema 0.8. *Una arista en un grafo G es de corte si y sólo si no pertenece a ningún ciclo en G.*