Análisis de Algoritmos I

Alan Reyes-Figueroa

Teoría de la Computación (Aula 18) 03.octubre.2022

Definición

Inputs

Ejemplos

Notación Asintótica

Análisis de Algoritmos

- Estimar los recursos (tiempo y memoria) que un algoritmos requiere para funcionar.
  - Estructura
  - Operaciones
- Algoritmos: argumentos de entrada
- El consumo de recursos del algoritmo se escribe en función del "tamaño" de estos *inputs*.

Inputs: Ejemplos

- Input: Arreglo a.
- Tamaño: número de elementos del arreglo a.
- Input: Un número entero n.
- Tamaño: Número de bits que requiere la representación binaria de n.
- Input: Grafo G.
- Tamaño: Número de nodos de G.
  Número de nodos + aristas.

Tiempo de Ejecución

Buscamos determinar el tiempo de ejecución (*running time*) de un algoritmos, esto es, el número de pasos u operaciones primitivas realizadas.

Ejemplo: (Algoritmo para contar coincidencias en un arreglo):

Input: Array a; int b.

`n = len(a)`	Asignación $t = c_1$
`count = 0`	Asignación $t = c_1$
`For i in range(0, n):`	Ciclo $t = n$ *
`        if (a[i] == b):`	Comparación $t = c_2$
`                count = count + 1`	Asignación $t = c_1$
	Suma $t = c_3$

Tiempo de Ejecución

- No calculamos directamente el tiempo de ejecución (en ns, μs) por varias razones:
  - no se comporta igual en cada máquina
  - variabilidad
  - dificultad en los cálculos.

- Es mucho más simple calcular el número de operaciones ejecutadas dentro del algoritmos en función de tamaño del input.

Escenarios

- Para un mismo algoritmos (y mismos *inputs*) podemos tener variaciones en el tiempo de ejecución de un algoritmo.

- Consideramos tres escenarios:
  - worst-case (peor caso),
  - average-case (caso promedio),
  - best-case (mejor caso).

Ejemplo

<u>Ejemplo</u>: (Algoritmo para contar coincidencias en un arreglo):

Input: Array a; int b.

Operación	Tiempo
`n = len(a)`	Asignación $t = c_1$
`count = 0`	Asignación $t = c_1$
`For i in range(0,n):`	Ciclo $t = n$ *
`    if (a[i] == b):`	Comparación $t = c_2$
`        count = count + 1`	Asignación $t = c_1$ Suma $t = c_3$

¿Cuántas operaciones hace el algoritmo?

$$T = c_1 + c_1 + n(c_2 + k(c_1 + c_3))$$

Ejemplo

- <u>Peor caso</u>: el condicional If es True las n veces

  $$\mathbf{T} = c_1 + c_1 + n(c_2 + n(c_1 + c_3))$$
  $$= 2c_1 + nc_2 + n^2(c_1 + c_3)$$

- <u>Mejor caso</u>: el condicional If nunca es True

  $$\mathbf{T} = c_1 + c_1 + n(c_2 + 0(c_1 + c_3))$$
  $$= 2c_1 + nc_2$$

- <u>Caso promedio</u>: el condicional If es True 0,1,2,…n veces

  $$T = \frac{1}{n+1}\left(\sum_{k=0}^{n} 2c_1 + \sum_{k=0}^{n} nc_2 + \sum_{k=0}^{n} k(c_1 + c_3)\right)$$
  $$= 2c_1 + nc_2 + \frac{n^2}{2}$$

Ejemplo

- Si construimos una fórmula para contar las operaciones del algoritmo, a los coeficientes en el mejor caso los podemos resumir en constantes a y b, así como en el peor caso en a, b y c.
- Para el mejor caso tendremos una función lineal como tiempo de ejecución, mientras que para el peor caso tendremos una cuadrática.
- Nos interesa: comparar dos algoritmos en cuanto a su tiempo de ejecución (**tasa de crecimiento**).

Ejercicio

- Algoritmo (Insertion sort)
  Input: array A

```
For j = 2 to n:
    k = A[j]
    i = j - 1
    While i > 0 and A[i] > k
          A[i+1] = A[i]
          i = i - 1
    A[i+1] = k
```

Notación Asintótica

- Notación **big-Oh**: $\mathcal{O}$(g(x))

Decimos que f es **O-grande** respecto de g, f(**x**) = O(g(**x**)), cuando **x** → **a**, si existe una constante C > 0 tal que

$$|f(\mathbf{x})| \leq C|g(\mathbf{x})|, \text{ para todo } |\mathbf{x}\text{-}\mathbf{a}| \leq r.$$

- Equivalentemente, f(**x**) = O(g(**x**)) cuando **x** → ∞ si existe C > 0 tal que

$$\lim_{\mathbf{x}\to a} |f(x)/g(x)| \leq C.$$

Notación Asintótica

- Notación **big-Oh**: $O\,(g(x))$

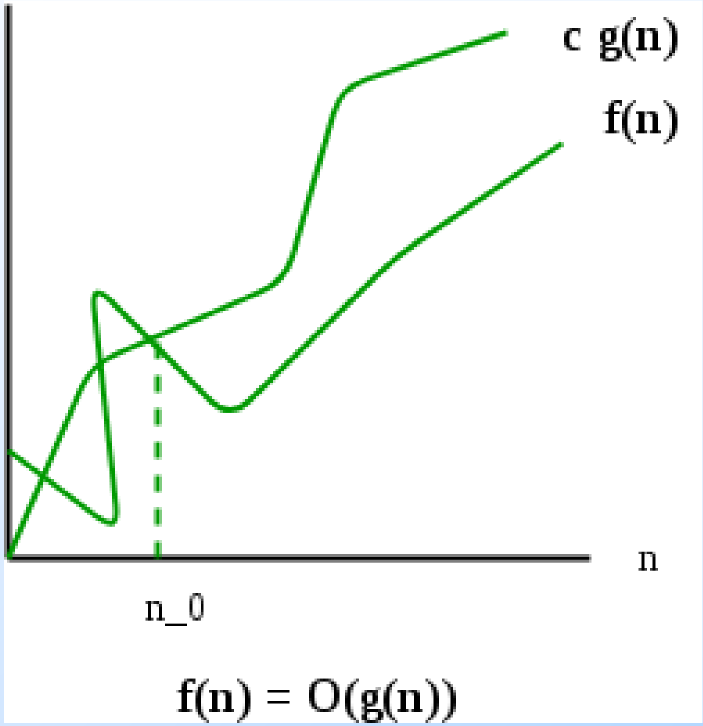
Decimos que f es **O-grande** respecto de g, $f(\mathbf{x}) = O(g(\mathbf{x}))$, cuando $\mathbf{x} \to \infty$ si existen constantes positivas r y C con

$$|f(\mathbf{x})| \le C|g(\mathbf{x})|, \text{ para todo } |\mathbf{x}| \ge r.$$

- Equivalentemente, $f(\mathbf{x}) = O(g(\mathbf{x}))$ cuando $\mathbf{x} \to \infty$ si existe $C > 0$ tal que

$$\lim_{\mathbf{x}\to\infty} |f(x)/g(x)| \le C.$$

Notación Asintótica



- f(n) = O(g(n)) quiere decir: asintóticamente (para valores muy grandes de n), g crece mucho rápido que f.

Notación Asintótica

- Notación **big-Omega**: $\Omega(g(x))$

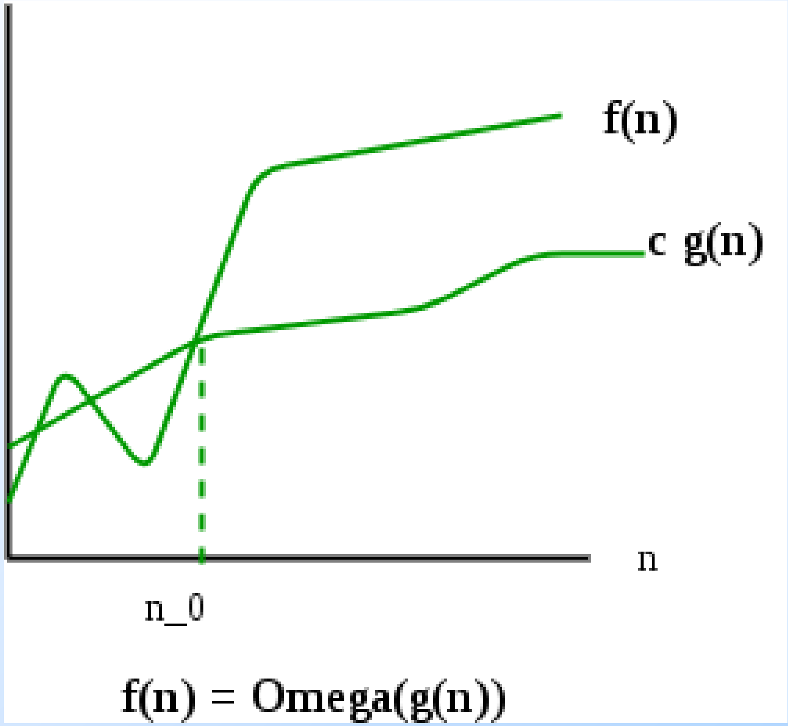
Decimos que f es **Ω-grande** respecto de g, $f(\mathbf{x}) = \Omega(g(\mathbf{x}))$, cuando $\mathbf{x} \to \infty$ si existen constantes positivas r y C con

$$|f(\mathbf{x})| \geq C|g(\mathbf{x})|, \text{ para todo } |\mathbf{x}| \geq r.$$

- Equivalentemente, $f(\mathbf{x}) = \Omega(g(\mathbf{x}))$ cuando $\mathbf{x} \to \infty$ si existe $C > 0$ tal que

$$\lim_{\mathbf{x}\to\infty} |f(x)/g(x)| \geq C.$$

Notación Asintótica



- $f(n) = \Omega(g(n))$ quiere decir: asintóticamente (para valores muy grandes de n), f crece mucho rápido que g.

Notación Asintótica

- Notación **big-Theta**: Θ (g(x))

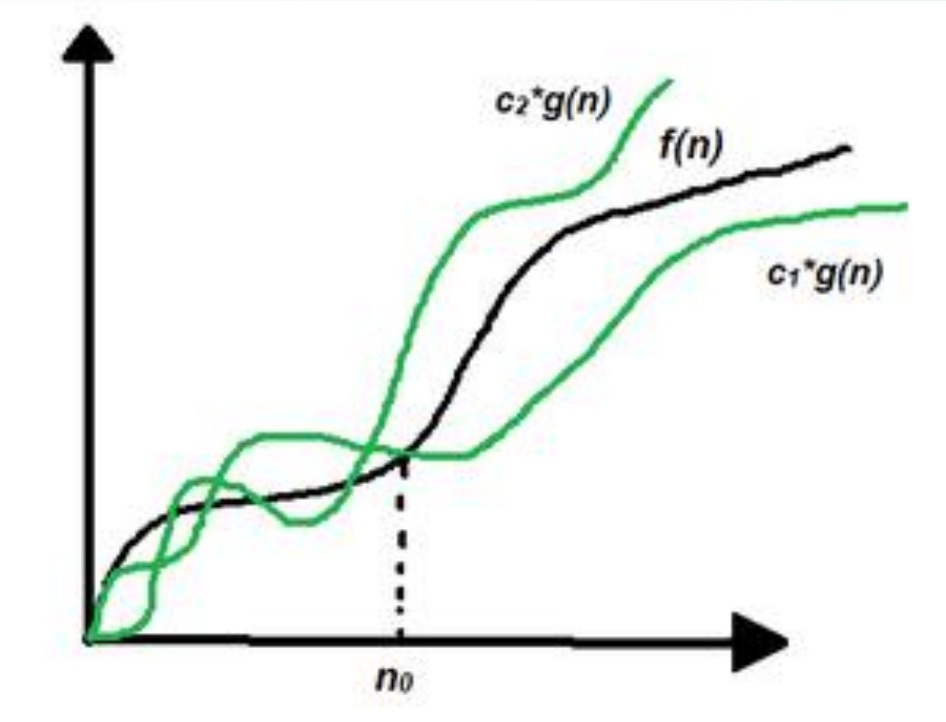
Decimos que f es **Θ-grande** respecto de g, f(**x**) = Θ(g(**x**)), cuando **x** → ∞ si existen constantes positivas r y c_1, c_2 con

$$c_1|g(\mathbf{x})| \le |f(\mathbf{x})| \le c_2|g(\mathbf{x})|, \text{ para } |\mathbf{x}| \ge r.$$

- Equivalentemente, f(**x**) = Θ(g(**x**)) cuando **x** → ∞ si existe C > 0 tal que

$$c_1 \le \lim_{\mathbf{x}\to\infty} |f(x)/g(x)| \le c_2.$$

Notación Asintótica



- f(n) = O(g(n)) quiere decir: asintóticamente (para valores muy grandes de n), f y g crecen de forma similar.

Notación Asintótica

- Notación **little-oh**: $o\,(g(x))$

Decimos que f es **o-pequeña** respecto g, $f(\mathbf{x}) = o(g(\mathbf{x}))$, cuando $\mathbf{x} \to \infty$ si

$$\lim_{\mathbf{x}\to\infty} |f(x)/g(x)| = 0.$$

Notación Asintótica

Típicamente vamos a tener:

$f(x) = O(g(x)) \;\Rightarrow\; g(x) = \Omega(f(x))$

$f(x) = \Omega(g(x)) \;\Rightarrow\; g(x) = O(f(x))$

$f(x) = o(g(x)) \;\Rightarrow\; g(x) = \Omega(f(x))$ y $\lim_{x\to\infty} |g(x)/f(x)| = \infty$

$f(x) = \Theta(g(x)) \;\Leftrightarrow\; g(x) = \Theta(f(x))$

Si $f(x) = \Theta(g(x))$ y $\lim_{x\to\infty} |g(x)/f(x)| = 1$, f y g son **asintóticamente equivalentes**.

Ejemplos

- Ejemplo 1: Estudiar la relación asintótica entre las funciones
  $f(n) = n^3 - n + 1$ $g(n) = n^3$

- Ejemplo 2: ¿Qué es $f(n) = O(\log n)$?

- Ejemplo 3: ¿Qué significa $f(n) = O(1)$?

Ejemplos

- Ejemplo 4: ¿Cuál función es mayor?
  $f(n) = \log n$ $g(n) = \mathrm{sqrt}(n)$

- Ejemplo 5: ¿Cuál es mayor?
  $f(n) = 0.5n^{1.5}$ $g(n) = 25n \log_{10} n$

- Ejemplo 6: ¿Cuál es mayor?
  $f(n) = n^3 + 5$ $g(n) = n^3 - 1$

Ejemplos

- Ejemplo 7: ¿Cuál es mayor?
  $f(n) = n^{1000}$ $g(n) = 5^n$

- Ejemplo 8: ¿Cuál es mayor?
  $f(n) = 10^n$ $g(n) = n^n$

- Ejemplo 9: ¿Qué es mayor?
  $f(n) = n^n$ $g(n) = n!$

Ejemplos

- Ejemplo 10: Hay dos algoritmos A y B, con tiempos de ejecución
  $T_A(n) = 5n \log_{10} n$ ms
  $T_B(n) = 25n$ ms

- ¿Cuál es mejor asintóticamente?

- Cuál es mejor para resolver un problema de tamaño n=512?

Growth ratio

- $O(\log(n))$
- $O(\sqrt{n})$

- $O(n)$
- $O(n\log(n))$
- $O(n^2)$
- $O(n^3)$
- …

- $O(2^n)$
- $O(3^n)$
- $O(10^n)$
- …

- $O(n^n)$
- $O(n!)$

Ejemplos: Growth

$O(1)$	hacer una operación arit.
$(\log(n))$	búsqueda binaria
$O(n)$	búsqueda lineal
$O(n\log(n))$	MergeSort
$O(n^2)$	suma de matrices, *shortest path* entre 2 nodos *Knapsack problem*
$O(n^3)$	producto de matrices Dijkstra en grafo completo
$O(k^n)$	optimización finita exhaustiva *n-queens*
$O(n!)$	determinante por cofactores *traveling salesman problem*