Análisis de Algoritmos

Alan Reyes-Figueroa

Teoría de la Computación **(Aula 24) 06.noviembre.2023**

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 algoritmo, 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)
count = 0
for i in range(0, n):
        if (a[i] == b):
                count = count + 1
return count;
```

$t = c_0 + c_1$ (lectura + asign.)

$t = c_1$ (asignación)

$t = n * c_2$ (comparación)

$t = c_0 + c_2$ (lect. + comp.)

$t = c_1 + c_3 + c_0$ (lect., +, asign.)

$t = c_0$ (asignación)

Ejemplo

Ejemplo: (Contar ocurrencias de b). Inputs: Array a, int b.

Operación	Tiempo
n = len(a)	t = c0 + c1 (lectura + asignación)
count = 0	t = c1 (asignación)
for i in range(0,n):	ciclo: t = c2 (comparación)
if (a[i] == b):	t = 2c0 + c2 (lectura + comparación)
count = count + 1	t = c0 + c3 + c1 (lectura + suma + asignación)
return count;	t = c2 (asignación)

¿Cuántas operaciones hace el algoritmo?

$$T = 2c_0 + 2c_1 + c_2 + (n-1)[3c_0 + 2c_2 + c_1)]$$

Ejemplo

Ejemplo: (Hallar el máximo). Inputs: Array a.

Operación	Tiempo	
n = len(a)	t = c0 + c1	(lectura + asignación)
max = a[0]	t = c0 + c1	(lectura + asignación)
for i in range(1,n):	ciclo: t = c2	(comparación)
if (a[i] > max):	t = 2c0 + c2	(lectura + comparación)
max = a[i]	t = c0 + c3 + c1	(lectura + suma + asignación)
return max;	t = c2	(asignación)

¿Cuántas operaciones hace el algoritmo?

$$T = c0 + 2c1 + c2 + n[3c0 + c2 + k(c1 + c2 + c3)]$$

$$= c0 + 2c1 + c2 + n(3c0 + c2) + nk(c1 + c2 + c3)$$

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 algoritmo en función de tamaño del input.

Escenarios

- Para un mismo algoritmo (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

Ejemplo: (Hallar el máximo en un arreglo de números):
Inputs: Array a.

Operación	Tiempo	
n = len(a)	t = c0 + c1	(lectura + asignación)
max = a[0]	t = c0 + c1	(lectura + asignación)
for i in range(1,n):	ciclo: t = c2	(comparación)
if (a[i] > max):	t = 2c0 + c2	(lectura + comparación)
max = a[i]	t = c0 + c3 + c1	(lectura + suma + asignación)
return max;	t = c2	(asignación)

¿Cuántas operaciones hace el algoritmo?

$$T = 2c_0 + 2c_1 + c_2 + (n-1)[3c_0 + 2c_2 + c_1)]$$

Ejemplo

Analizamos tres posibles casos:

- Mejor Caso:

  $T = 2c_0 + 2c_1 + c_2 + (n-1)[3c_0 + 2c_2 + c_1)]$

- Peor Caso:

  $T = 2c_0 + 2c_1 + c_2 + (n-1)[3c_0 + 2c_2 + c_1)]$

- Caso Promedio:

  $T = 2c_0 + 2c_1 + c_2 + (n-1)[3c_0 + 2c_2 + c_1)]$

Ejemplo

Ejemplo: (Contar coincidencias en un arreglo):

Inputs: Array a, int b.

Operación	Tiempo
n = len(a)	t = c0 + c1 (lectura + asignación)
count = 0	t = c1 (asignación)
for i in range(0,n):	ciclo: t = c2 (comparación)
if (a[i] == b):	t = 2c0 + c2 (lectura + comparación)
count = count + 1	t = c0 + c3 + c1 (lectura + suma + asignación)
return count;	t = c2 (asignación)

¿Cuántas operaciones hace el algoritmo?

$$T = 2c_0 + 2c_1 + c_2 + (n-1)[3c_0 + 2c_2 + c_1)]$$

Ejemplo

Analizamos tres posibles casos:

- Mejor Caso: el condicional if nunca es True

  T = c0 + 2c1 + c2 + n[3c0 + c2 + 0(c1 + c2 + c3)]

  = c0 + 2c1 + c2 + n[3c0 + c2]

- Peor Caso: el condicional if es True las n veces

  T = c0 + 2c1 + c2 + n[3c0 + c2 + n(c1 + c2 + c3)]

  = c0 + 2c1 + c2 + n(3c0 + c2) + n^2(c1 + c2 + c3)

- Caso Promedio: el promedio de todos escenarios $T = \frac{1}{n+1} \sum_{k=0}^{n} \left(c_0 + 2c_1 + c_2 + n\left(3c_0 + c_2 + k(c_1 + c_0 + c\_3)\right)\right)$

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**).

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 $\mathbf{x} \rightarrow \mathbf{a}$, si existe una constante $C > 0$ tal que

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

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

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

Notación Asintótica

- Notación **big-Oh**: $O(g(x))$

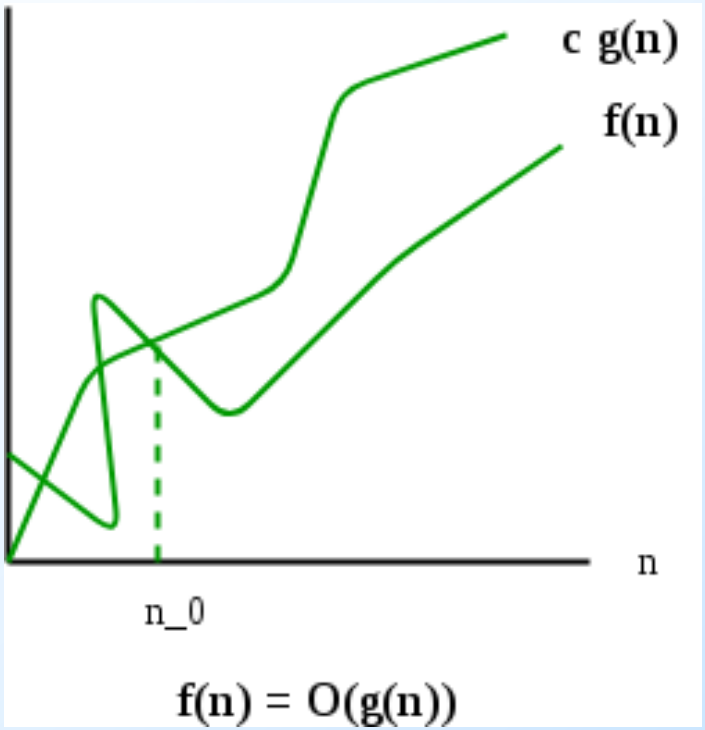
Decimos que f es **O-grande** respecto de g, $f(\mathbf{x}) = O(g(\mathbf{x}))$, cuando $\mathbf{x} \to \infty$ si existen constantes positivas r y C con

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

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

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

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

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

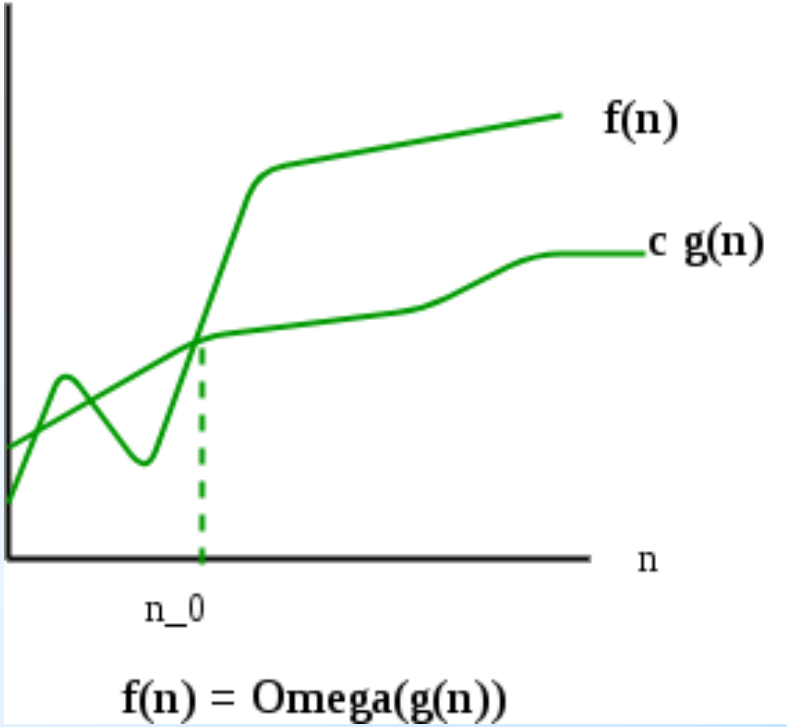
Decimos que f es **Θ-grande** respecto de g, $f(\mathbf{x}) = \Theta(g(\mathbf{x}))$, cuando $\mathbf{x} \to \infty$ si existen constantes positivas r y c_1, c_2 con

$c_1|g(\mathbf{x})| \leq |f(\mathbf{x})| \leq c_2|g(\mathbf{x})|$, para $|\mathbf{x}| \geq r$.

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

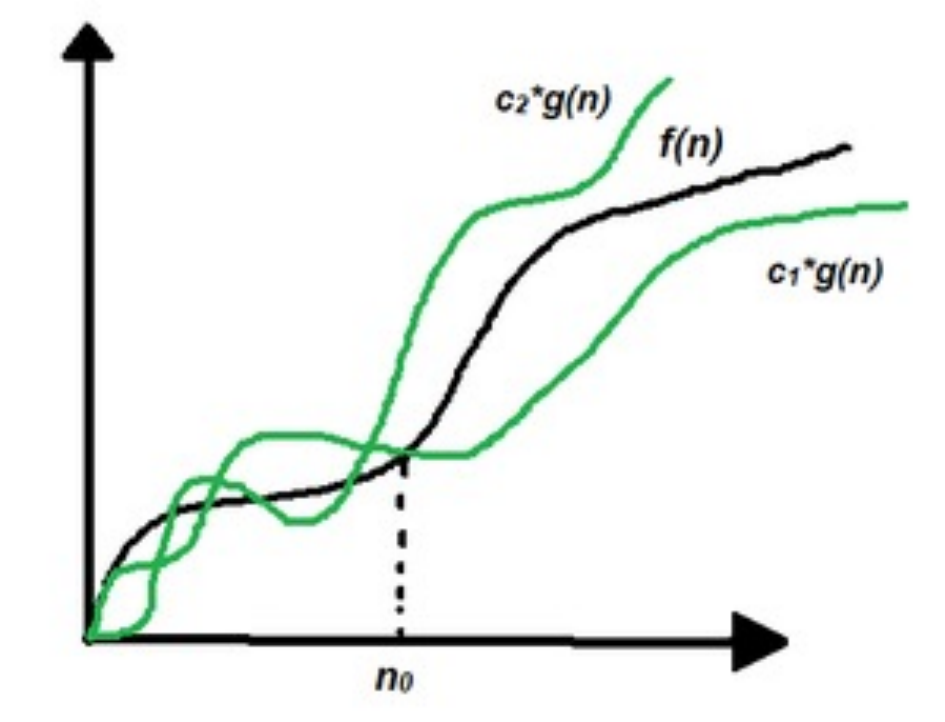
$$c_1 \leq \lim_{\mathbf{x}\to\infty} |f(x)/g(x)| \leq c_2.$$

Notación Asintótica



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

Notación Asintótica



- f(n) = Θ(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 $\mathbf{x} \rightarrow \infty$ si

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

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?
  $f(n) = \log n$ $g(n) = \text{sqrt}(n)$

- 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?
  $f(n) = n^n$ $g(n) = n!$

Ejemplo

Ejemplo: (Contar coincidencias en un arreglo):

Inputs: Array a; int b.

Operación	Tiempo
n = len(a)	t = c0 + c1 (lectura + asignación)
count = 0	t = c1 (asignación)
for i in range(0,n):	ciclo: t = c2 (comparación)
if (a[i] == b):	t = 2c0 + c2 (lectura + comparación)
count = count + 1	t = c0 + c3 + c1 (lectura + suma + asignación)
return count;	t = c2 (asignación)

¿Cuántas operaciones hace el algoritmo?

$$T = c_0 + 2c_1 + c_2 + n[3c_0 + c_2 + k(c_1 + c_2 + c_3)]$$

Ejercicio

- Algoritmo (Insertion sort) Input: array A

```
1.Function InsertionSort(A):
2.For j = 2 to n:
3.    k := A[j]
4.    i := j - 1
5.    While i > 0 and A[i] > k
6.          A[i+1] := A[i]
7.          i := i - 1
8.    A[i+1] := k
```

Ejercicio

- Algoritmo (Binary search) Input: array A

```
1. Function BinarySearch(A, n, T):
2.     L := 0
3.     R := n - 1
4.     while L ≤ R do:
5.         m := floor((L + R) / 2)
6.         if A[m] < T then
7.             L := m + 1
8.         else if A[m] > T then
9.             R := m - 1
10.         else:
11.             return m
12.     return unsuccessful
```

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

- $O(\log(n))$
- $O(\sqrt{n})$
- $O(n)$
- $O(n\log(n))$
- $O(n^2)$
- $O(n^3)$
- …
- $O(2^n)$
- $O(3^n)$
- $O(10^n)$
- …
- $O(n!)$
- $O(n^n)$

Ejemplos: Growth

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