Complejidad

Miguel Raggi

Algoritmos
ENES
UNAM

5 de febrero de 2019

Índice:

Índice:

Cómo medimos

- El problema es que casi nunca ocurre que un algoritmo sea mejor en todos los rubros que otro.

Cómo medimos

- El problema es que casi nunca ocurre que un algoritmo sea mejor en todos los rubros que otro.
- Elegir el algoritmo adecuado no siempre es fácil.

Cómo medimos

- El problema es que casi nunca ocurre que un algoritmo sea mejor en todos los rubros que otro.
- Elegir el algoritmo adecuado no siempre es fácil.
- La medida final será qué tan bien corra, haciendo mediciones de tiempo (qué algoritmo se tarda más, cuál menos).

Cómo medimos

- El problema es que casi nunca ocurre que un algoritmo sea mejor en todos los rubros que otro.
- Elegir el algoritmo adecuado no siempre es fácil.
- La medida final será qué tan bien corra, haciendo mediciones de tiempo (qué algoritmo se tarda más, cuál menos).
- Pero hay varios problemas con esto:

Cómo medimos

- El problema es que casi nunca ocurre que un algoritmo sea mejor en todos los rubros que otro.
- Elegir el algoritmo adecuado no siempre es fácil.
- La medida final será qué tan bien corra, haciendo mediciones de tiempo (qué algoritmo se tarda más, cuál menos).
- Pero hay varios problemas con esto:
  - El primero y más importante es que programar los algoritmos es difícil y toma tiempo. No puedes programar 14 algoritmos diferentes sólo para ver cuál será mejor al final.

Cómo medimos

- El problema es que casi nunca ocurre que un algoritmo sea mejor en todos los rubros que otro.
- Elegir el algoritmo adecuado no siempre es fácil.
- La medida final será qué tan bien corra, haciendo mediciones de tiempo (qué algoritmo se tarda más, cuál menos).
- Pero hay varios problemas con esto:
  - El primero y más importante es que programar los algoritmos es difícil y toma tiempo. No puedes programar 14 algoritmos diferentes sólo para ver cuál será mejor al final.
  - Además, varía mucho de implementación a implementación.

Cómo medimos

- El problema es que casi nunca ocurre que un algoritmo sea mejor en todos los rubros que otro.
- Elegir el algoritmo adecuado no siempre es fácil.
- La medida final será qué tan bien corra, haciendo mediciones de tiempo (qué algoritmo se tarda más, cuál menos).
- Pero hay varios problemas con esto:
  - El primero y más importante es que programar los algoritmos es difícil y toma tiempo. No puedes programar 14 algoritmos diferentes sólo para ver cuál será mejor al final.
  - Además, varía mucho de implementación a implementación.
  - Hablando más académicamente, si queremos comparar un algoritmo inventado en china en los 90's con uno inventado en rusia el año pasado, ¿qué computadora usamos para medir?

Medición: Órden

- Lo que usamos es órdenes teóricos. Se dicen cosas como "este algoritmo es $O(n^2)$ o $O(n\log(n))$".

Medición: Órden

- Lo que usamos es órdenes teóricos. Se dicen cosas como "este algoritmo es $O(n^2)$ o $O(n\log(n))$".
- Esto mide, de cierta forma, el crecimiento de la velocidad conforme le pedimos al algoritmo que opere en cosas más y más grandes.

Medición: Órden

- Lo que usamos es órdenes teóricos. Se dicen cosas como "este algoritmo es $O(n^2)$ o $O(n\log(n))$".
- Esto mide, de cierta forma, el crecimiento de la velocidad conforme le pedimos al algoritmo que opere en cosas más y más grandes.
- Entre más grande sea el orden, más tardará el algoritmo en operar en entradas muy grandes.

Medición: Órden

- Lo que usamos es órdenes teóricos. Se dicen cosas como "este algoritmo es $O(n^2)$ o $O(n\log(n))$".
- Esto mide, de cierta forma, el crecimiento de la velocidad conforme le pedimos al algoritmo que opere en cosas más y más grandes.
- Entre más grande sea el orden, más tardará el algoritmo en operar en entradas muy grandes.
- Luego vemos exactamente qué significa que un algoritmo sea $O(\text{algo})$, pero por ahora, piénsenlo así:

Medición: Órden

- Lo que usamos es órdenes teóricos. Se dicen cosas como "este algoritmo es $O(n^2)$ o $O(n\log(n))$".
- Esto mide, de cierta forma, el crecimiento de la velocidad conforme le pedimos al algoritmo que opere en cosas más y más grandes.
- Entre más grande sea el orden, más tardará el algoritmo en operar en entradas muy grandes.
- Luego vemos exactamente qué significa que un algoritmo sea $O(\text{algo})$, pero por ahora, piénsenlo así:
  - $O(1), O(\log(n))$: Muy rápidos!
  - $O(n\log(n))$, $O(n)$: Rápidos.
  - $O(n^2)$: Medianos.
  - $O(n^3)$, $O(n^4)$ o más: Lentos.
  - $O(2^n)$: Muy lentos.

¿Cómo se mide qué tan bueno es un algoritmo?

- Con el avance en la velocidad de las computadoras, es difícil comparar algoritmos.

¿Cómo se mide qué tan bueno es un algoritmo?

- Con el avance en la velocidad de las computadoras, es difícil comparar algoritmos.
- Utilizamos entonces el análisis asintótico:

¿Cómo se mide qué tan bueno es un algoritmo?

- Con el avance en la velocidad de las computadoras, es difícil comparar algoritmos.
- Utilizamos entonces el análisis asintótico:

Definición

Sea $f : \mathbb{N} \to \mathbb{N}$.

- *Intuitivamente: Decimos que* $g \in O(f)$ *si, hasta constantes,* f *eventualmente le gana a* g*; es decir, si* g *esta casi acotado por arriba por un múltiplo de* f.

¿Cómo se mide qué tan bueno es un algoritmo?

- Con el avance en la velocidad de las computadoras, es difícil comparar algoritmos.
- Utilizamos entonces el análisis asintótico:

Definición

Sea $f : \mathbb{N} \to \mathbb{N}$.

- *Intuitivamente: Decimos que* $g \in O(f)$ *si, hasta constantes,* f *eventualmente le gana a* g*; es decir, si* g *esta casi acotado por arriba por un múltiplo de* f.
- *Formalmente: Definimos el conjunto de funciones* $O(f)$ *por la siguiente propiedad:*

$$g \in O(f) \iff (\exists c \in \mathbb{R})(\exists N \in \mathbb{N})(\forall n \geq N)(g(n) < cf(n))$$

¿Cómo se mide qué tan bueno es un algoritmo?

- Con el avance en la velocidad de las computadoras, es difícil comparar algoritmos.
- Utilizamos entonces el análisis asintótico:

Definición

Sea $f : \mathbb{N} \to \mathbb{N}$.

- *Intuitivamente: Decimos que* $g \in O(f)$ *si, hasta constantes,* f *eventualmente le gana a* g*; es decir, si* g *esta casi acotado por arriba por un múltiplo de* f.
- *Formalmente: Definimos el conjunto de funciones* $O(f)$ *por la siguiente propiedad:*

$$g \in O(f) \iff (\exists c \in \mathbb{R})(\exists N \in \mathbb{N})(\forall n \geq N)(g(n) < cf(n))$$

- Usualmente queremos que f sea una función simple: $O(n^2)$, $O(2^n)$, $O(n \log(n))$, etc. Nada de

$$O\left(\sqrt{e^{\tan(2n)} - 2n \log(\sin(n))} + 1\right)$$

Grande-O

Definición

Sea $f : \mathbb{N} \to \mathbb{N}$.

Grande-O

Definición

Sea $f : \mathbb{N} \to \mathbb{N}$.

- *Intuitivamente: Decimos que* $g \in O(f)$ *si, hasta constantes,* f *eventualmente le gana a* g*; es decir, si* g *esta casi acotado por arriba por un multiplo de* f.

Grande-O

Definición

Sea $f : \mathbb{N} \to \mathbb{N}$.

- *Intuitivamente: Decimos que* $g \in O(f)$ *si, hasta constantes,* f *eventualmente le gana a* g*; es decir, si* g *esta casi acotado por arriba por un multiplo de* f*.*
- *Formalmente: Definimos el conjunto de funciones* $O(f)$ *por la siguiente propiedad:*

$$g \in O(f) \iff (\exists c \in \mathbb{R})(\exists N \in \mathbb{N})(\forall n \geq N)(g(n) < cf(n))$$

Grande-O

Definición

Sea $f : \mathbb{N} \to \mathbb{N}$.

- *Intuitivamente: Decimos que* $g \in O(f)$ *si, hasta constantes,* f *eventualmente le gana a* g*; es decir, si* g *esta casi acotado por arriba por un multiplo de* f*.*
- *Formalmente: Definimos el conjunto de funciones* $O(f)$ *por la siguiente propiedad:*

$$g \in O(f) \iff (\exists c \in \mathbb{R})(\exists N \in \mathbb{N})(\forall n \geq N)(g(n) < cf(n))$$

- $17n \in O(n^2)$

Grande-O

Definición

Sea $f : \mathbb{N} \to \mathbb{N}$.

- *Intuitivamente: Decimos que* $g \in O(f)$ *si, hasta constantes,* f *eventualmente le gana a* g*; es decir, si* g *esta casi acotado por arriba por un multiplo de* f*.*
- *Formalmente: Definimos el conjunto de funciones* $O(f)$ *por la siguiente propiedad:*

$$g \in O(f) \iff (\exists c \in \mathbb{R})(\exists N \in \mathbb{N})(\forall n \geq N)(g(n) < cf(n))$$

- $17n \in O(n^2)$
- $n^2 \in O(n^2)$

Grande-O

Definición

Sea $f : \mathbb{N} \to \mathbb{N}$.

- *Intuitivamente: Decimos que* $g \in O(f)$ *si, hasta constantes,* f *eventualmente le gana a* g*; es decir, si* g *esta casi acotado por arriba por un multiplo de* f*.*
- *Formalmente: Definimos el conjunto de funciones* $O(f)$ *por la siguiente propiedad:*

$$g \in O(f) \iff (\exists c \in \mathbb{R})(\exists N \in \mathbb{N})(\forall n \geq N)(g(n) < cf(n))$$

- $17n \in O(n^2)$
- $n^2 \in O(n^2)$
- $n^3 - 16n^2 \notin O(n^2)$

Grande-O

Definición

Sea $f : \mathbb{N} \to \mathbb{N}$.

- *Intuitivamente: Decimos que* $g \in O(f)$ *si, hasta constantes,* f *eventualmente le gana a* g*; es decir, si* g *esta casi acotado por arriba por un multiplo de* f*.*
- *Formalmente: Definimos el conjunto de funciones* $O(f)$ *por la siguiente propiedad:*

$$g \in O(f) \iff (\exists c \in \mathbb{R})(\exists N \in \mathbb{N})(\forall n \geq N)(g(n) < cf(n))$$

- $17n \in O(n^2)$
- $n^2 \in O(n^2)$
- $n^3 - 16n^2 \notin O(n^2)$
- $n\sqrt{n}\log(n) \in O(n^2)$

Grande-O

Definición

Sea $f : \mathbb{N} \to \mathbb{N}$.

- *Intuitivamente: Decimos que* $g \in O(f)$ *si, hasta constantes,* f *eventualmente le gana a* g*; es decir, si* g *esta casi acotado por arriba por un multiplo de* f*.*
- *Formalmente: Definimos el conjunto de funciones* $O(f)$ *por la siguiente propiedad:*

$$g \in O(f) \iff (\exists c \in \mathbb{R})(\exists N \in \mathbb{N})(\forall n \geq N)(g(n) < cf(n))$$

- $17n \in O(n^2)$
- $n^2 \in O(n^2)$
- $n^3 - 16n^2 \notin O(n^2)$
- $n\sqrt{n}\log(n) \in O(n^2)$
- $\log(n^7 + 800n^2) \in O(\log(n))$

Grande-Ω y Grande-Θ

Sea f una función $f : \mathbb{N} \to \mathbb{N}$.

Grande-Ω y Grande-Θ

Sea f una función $f : \mathbb{N} \to \mathbb{N}$.

Definición

Decimos que $g \in \Omega(f)$ si $f \in O(g)$. Es decir, si eventualmente g "le gana" a f.

Grande-Ω y Grande-Θ

Sea f una función $f : \mathbb{N} \to \mathbb{N}$.

Definición

Decimos que $g \in \Omega(f)$ si $f \in O(g)$. Es decir, si eventualmente g "le gana" a f.

Definición

Definimos $\Theta(f) = O(f) \cap \Omega(f)$. Es decir, $g \in \Theta(f)$ si f y g "crecen igual" (y entonces $f \in \Theta(g)$.

Grande-Ω y Grande-Θ

Sea f una función $f : \mathbb{N} \to \mathbb{N}$.

Definición

Decimos que $g \in \Omega(f)$ si $f \in O(g)$. Es decir, si eventualmente g "le gana" a f.

Definición

Definimos $\Theta(f) = O(f) \cap \Omega(f)$. Es decir, $g \in \Theta(f)$ si f y g "crecen igual" (y entonces $f \in \Theta(g)$.

Ejemplos: $17n^2 - n \in \Theta(n^2)$, $n \log(n^{500} + 1) \in \Theta(n \log(n))$

Tamaño de una instancia

- Medimos la rapidez de los algoritmos usando la notación de O, Ω y Θ.

Tamaño de una instancia

- Medimos la rapidez de los algoritmos usando la notación de O, Ω y Θ.
- Pensemos en $f(n)$ como el tiempo que tarda el algoritmo en correr cuando le metemos una instancia de tamaño n.

Tamaño de una instancia

- Medimos la rapidez de los algoritmos usando la notación de O, Ω y Θ.
- Pensemos en $f(n)$ como el tiempo que tarda el algoritmo en correr cuando le metemos una instancia de tamaño n.
- Tiempo: ¿Tiempo en dónde? Mejor medir número de pasos. explica paso

Tamaño de una instancia

- Medimos la rapidez de los algoritmos usando la notación de O, Ω y Θ.
- Pensemos en $f(n)$ como el tiempo que tarda el algoritmo en correr cuando le metemos una instancia de tamaño n.
- Tiempo: ¿Tiempo en dónde? Mejor medir número de pasos. explica paso
- Tamaño de una instancia: Un algoritmo, más que resolver un sólo problema, resuelve una clase de problemas que crecen con tamaño n.

Tamaño de una instancia

- Medimos la rapidez de los algoritmos usando la notación de O, Ω y Θ.
- Pensemos en $f(n)$ como el tiempo que tarda el algoritmo en correr cuando le metemos una instancia de tamaño n.
- Tiempo: ¿Tiempo en dónde? Mejor medir número de pasos. explica paso
- Tamaño de una instancia: Un algoritmo, más que resolver un sólo problema, resuelve una clase de problemas que crecen con tamaño n.
- Usualmente n es el número de bits necesarios para describir el problema. mucho cuidado

Tamaño de una instancia

- Medimos la rapidez de los algoritmos usando la notación de O, Ω y Θ.
- Pensemos en $f(n)$ como el tiempo que tarda el algoritmo en correr cuando le metemos una instancia de tamaño n.
- Tiempo: ¿Tiempo en dónde? Mejor medir número de pasos. explica paso
- Tamaño de una instancia: Un algoritmo, más que resolver un sólo problema, resuelve una clase de problemas que crecen con tamaño n.
- Usualmente n es el número de bits necesarios para describir el problema. mucho cuidado
- Como a O, Ω y Θ les dan igual las constantes, bits $\approx$ bytes $\approx$ enteros menores que un entero fijo $=$ etc.

Tamaño de una instancia

- Medimos la rapidez de los algoritmos usando la notación de O, Ω y Θ.
- Pensemos en $f(n)$ como el tiempo que tarda el algoritmo en correr cuando le metemos una instancia de tamaño n.
- Tiempo: ¿Tiempo en dónde? Mejor medir número de pasos. explica paso
- Tamaño de una instancia: Un algoritmo, más que resolver un sólo problema, resuelve una clase de problemas que crecen con tamaño n.
- Usualmente n es el número de bits necesarios para describir el problema. mucho cuidado
- Como a O, Ω y Θ les dan igual las constantes, bits $\approx$ bytes $\approx$ enteros menores que un entero fijo $=$ etc.
- Ejemplo: ¿Cuántos bits se necesitan para describir a todos los enteros hasta 7625?

Tamaño de una instancia

- Medimos la rapidez de los algoritmos usando la notación de O, Ω y Θ.
- Pensemos en $f(n)$ como el tiempo que tarda el algoritmo en correr cuando le metemos una instancia de tamaño n.
- Tiempo: ¿Tiempo en dónde? Mejor medir número de pasos. explica paso
- Tamaño de una instancia: Un algoritmo, más que resolver un sólo problema, resuelve una clase de problemas que crecen con tamaño n.
- Usualmente n es el número de bits necesarios para describir el problema. mucho cuidado
- Como a O, Ω y Θ les dan igual las constantes, bits $\approx$ bytes $\approx$ enteros menores que un entero fijo $=$ etc.
- Ejemplo: ¿Cuántos bits se necesitan para describir a todos los enteros hasta 7625?
- Respuesta: $\lceil \log_2(7625) \rceil = 13$ Habla de factorizar números y ajedrez

¿Mejor, peor o promedio?

- Hablé de $f(n) =$ número de pasos, como si eso estuviera bien definido. Pero quizás para instancias del "mismo tamaño", el número de pasos varía muchísimo.

¿Mejor, peor o promedio?

- Hablé de $f(n) =$ número de pasos, como si eso estuviera bien definido. Pero quizás para instancias del "mismo tamaño", el número de pasos varía muchísimo.
- Usualmente nos interesa el peor caso, o el promedio, casi nunca el "mejor caso" (usualmente en el mejor de los casos, no hay nada que hacer y el tiempo es casi 0).

¿Mejor, peor o promedio?

- Hablé de $f(n) =$ número de pasos, como si eso estuviera bien definido. Pero quizás para instancias del "mismo tamaño", el número de pasos varía muchísimo.
- Usualmente nos interesa el peor caso, o el promedio, casi nunca el "mejor caso" (usualmente en el mejor de los casos, no hay nada que hacer y el tiempo es casi 0).
- Usualmente debes considerar ambos, el peor caso y el caso promedio.

¿Mejor, peor o promedio?

- Hablé de $f(n) =$ número de pasos, como si eso estuviera bien definido. Pero quizás para instancias del "mismo tamaño", el número de pasos varía muchísimo.
- Usualmente nos interesa el peor caso, o el promedio, casi nunca el "mejor caso" (usualmente en el mejor de los casos, no hay nada que hacer y el tiempo es casi 0).
- Usualmente debes considerar ambos, el peor caso y el caso promedio.
- A veces en la práctica es falso que si un algoritmo A corre en tiempo $\Theta(n)$ es mejor que uno B que corra en tiempo $\Theta(n^3)$. Quizás es $10000000000n$ vs $0.0001n^3$.

¿Mejor, peor o promedio?

- Hablé de $f(n) =$ número de pasos, como si eso estuviera bien definido. Pero quizás para instancias del "mismo tamaño", el número de pasos varía muchísimo.
- Usualmente nos interesa el peor caso, o el promedio, casi nunca el "mejor caso" (usualmente en el mejor de los casos, no hay nada que hacer y el tiempo es casi 0).
- Usualmente debes considerar ambos, el peor caso y el caso promedio.
- A veces en la práctica es falso que si un algoritmo A corre en tiempo $\Theta(n)$ es mejor que uno B que corra en tiempo $\Theta(n^3)$. Quizás es $10000000000n$ vs $0.0001n^3$.
- Efectivamente, para n suficientemente grande, A le ganará a B, pero quizás sólo nos interesan casos pequeños.

¿Cómo medimos el tiempo?

Si algo cuesta $0.01n^2 + 10n$, ¿por qué decimos que es $\Theta(n^2)$?

n	$a = 0.01n^2$	$b = 10n$	$0.01n^2 + 10n$	$\frac{a+b}{n^2}$
10	1	100	101	1.01
50	25	500	525	0.21
100	100	1000	1,100	0.11
500	2,500	5,000	7,500	0.03
1,000	10,000	10,000	20,000	0.02
5,000	25,000	50,000	300,000	0.01
10,000	1,000,000	100,000	1,100,000	0.01
50,000	25,000,000	500,000	25,500,000	0.01
100,000	100,000,000	1,000,000	101,000,000	0.01
500,000	2,500,000,000	5,000,000	2,505,000,000	0.01

Fin.

¡Gracias!