Análise de Algoritmos

Ronaldo Cristiano Prati

Bloco A, sala 513-2

ronaldo.prati@ufabc.edu.br

Análise assintótica

(recapitulando)

- Ao ver expressões como $n + 10$ ou $n^2 + 1$ pensamos geralmente em valores pequenos de n
- A análise algoritmica faz justamente o contrário: ignora valores pequenos e concentra-se nos valores enormes de n
- Esse tipo de análise se chama análise assintótica.

Notação assintótica

(recapitulando)

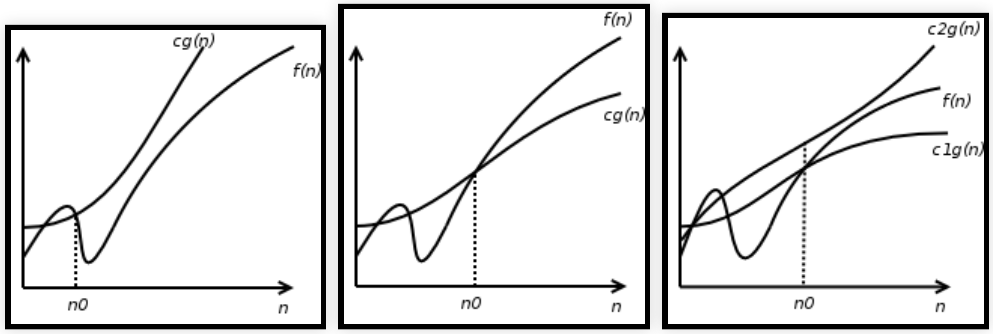
- Ferramenta para analisar algoritmos e descrever a ordem de crescimento dos tempos de execução
- Provê formalismo e vocabulário matemático que nos permitem argumentar sobre a qualidade e eficiência de algoritmos.

Notação assintótica O, Ω, Θ

(recordando)

- $T(n) = O(g(n))$ se existem constantes positivas C e n_0 tais que $T(n) \leq Cg(n)$ para todo $n \geq n_0$;
- $T(n) = \Omega(g(n))$ se existem constantes positivas c e n_0 tais que $cg(n) \leq T(n)$ para todo $n \geq n_0$
- Dizemos que $T(n) = \Theta(g(n))$ se, e somente se:
  - $T(n) = O(g(n))$ e
  - $T(n) = \Omega(g(n))$

Notação assintótica



Notação assintótica

(recapitulando)

- Para valores enormes de n, as funções:
  n^2, $\frac{3}{2}n^2$, $9999n^2$, $\frac{n^2}{1000}$, $n^2 + 100n$,
  crescem todas com velocidades muito parecidas e portanto são todas "equivalentes"
- Esse tipo de análise, interessado somente em valores enormes de n, é chamado assintótica. Nessa análise, as funções são classificadas em "ordens"; todas as funções de uma mesma ordem são "equivalentes"
- As cinco funções acima, por exemplo, pertencem à mesma ordem de crescimento

Notação assintótica o, ω

- $T(n) = o(g(n))$ se existem constantes positivas c e n_0 tais que $0 \leq T(n) < cg(n)$ para todo $n \geq n_0$;
- $T(n) = \omega(g(n))$ se existem constantes positivas C e n_0 tais que $T(n) > Cg(n)$ para todo $n \geq n_0$

Notação assintótica o, ω

- Por exemplo, $2n = o(n^2)$, mas $2n^2$ não é $o(n^2)$
  - Se $T(n) = o(g(n))$, então $T(n)$ é insignificante com relação a $g(n)$, para n grande. Alternativamente, $T(n) = o(g(n))$ quando

$$\lim_{n \to \infty} \frac{T(n)}{g(n)} = 0$$

- Por exemplo, $2n^2 = \omega(n)$, mas $2n^2$ não é $\omega(n^2)$
  - Se $T(n) = \omega(g(n))$, então $g(n)$ é insignificante com relação a $T(n)$, para n grande. Alternativamente, $T(n) = \omega(g(n))$ quando

$$\lim_{n \to \infty} \frac{T(n)}{g(n)} = \infty$$

Notação assintótica

(Exemplo 1)

- Mostre que $T(n) = \sum_{i=0}^{k} a_i n^i$ em que a_i são constante para $i \in 0..k$ é $O(n^k)$
- Escolha $n_0 = 1$ e $c = \sum_{i=0}^{k} |a_i|$. Temos que mostrar que $\forall n \geq 1, T(n) \leq cn^k$

$$\begin{aligned} T(n) &\leq |a_k|n^k + \cdots + |a_1|n + |a_0| \\ &\leq |a_k|n^k + \cdots + |a_1|n^k + |a_0|n^k \\ &= cn^k \end{aligned}$$

Notação assintótica

(Exemplo 2)

- Mostre que para todo $k > 1, T(n) = n^k$ não é $O(n^{k-1})$
- Por contradição, suponha que $n^k = O(n^{k-1})$. Então, $\exists c, n_0 > 0$ tal que
$$n^k \leq cn^{k-1}, \forall n \geq n_0$$
- Cancelando n^{k-1} de ambos os lados:
$$n \leq c, \forall n \geq n_0$$
- O que é claramente falso (contradição).

Notação assintótica

(Exemplo 3)

- Mostre que $T(n) = 2^{n+10}$ é $O(2^n)$
- Temos que escolher constantes n, n_0 tal que

$$2^{n+10} \leq c2^n, \forall n > n_0$$

- Observe que:

$$2^{n+10} = 2^{10} \cdot 2^n = 1024 \cdot 2^n$$

- Basta escolhermos $c = 1024$ e $n_0 = 1$

Notação assintótica

(Exemplo 4)

- Mostre que $T(n) = 2^{10n}$ não é $O(2^n)$
- Por contradição. Se $2^{10n} é O(2^n)$, então $\exists c, n_0 > 0$ tal que

  -Temos que escolher constantes n, n_0 tal que

$$2^{10n} \leq c2^n, \forall n > n_0$$

- Entretanto, cancelando 2^n de ambos os lados

$$2^{9n} \leq c, \forall n > n_0$$

- Que é obviamente falso

Notação assintótica

(Exemplo 5)

- Para cada par de funções positivas $f(n)$ e $g(n)$, mostrar que
$$\max[f(n), g(n)] = \Theta(f(n) + g(n))$$
- Para todo n temos que:
$$\max[f(n), g(n)] \leq f(n) + g(n)$$
- e
$$2 \max[f(n), g(n)] \geq f(n) + g(n)$$
$$\max[f(n), g(n)] \geq \frac{1}{2}(f(n) + g(n))$$
- Então
$$\frac{1}{2}(f(n) + g(n)) \leq \max[f(n), g(n)] \leq f(n) + g(n), \forall n \geq 1$$
- Portanto $\max[f(n), g(n)] = \Theta(f(n) + g(n))$, bastando escolher $c = \frac{1}{2}$ e $C = 1$, para $n_0 = 1$

Notação assintótica

(Exemplo 6)

- Seja $T(n) = 10n + 3\log n$, mostre que $T(n) = o(n^2)$
- Precisamos mostrar que $10n + 3\log n < cn^2, \forall n > n_0$.
- Observe que $10n + 3\log n < 13n$

$$10n + 3\log n < cn^2$$

$$13n < cn^2$$

- Quando $n > \frac{13}{c}, \forall c, 13n < cn^2$, portanto $T(n) = o(n^2)$ com $n_0 = 13/c + 1$

Notação assintótica

Algumas classes de ordem mais comuns são:

Função $g(n)$	**Classe**
1	Constante
$\log n$	Logarítmica
n	Linear
$n \log n$	Loglinear
n^2	Quadrática
n^3	Cúbica
2^n	Exponencial

Função $g(n)$	**Classe**
$n!$	Fatorial

Classes de problemas mais comuns

- **Constante** $T(n) = O(1)$ - número fixo de operações
- **Logarítmica** $T(n) = O(\log n)$ - ocorre tipicamente em algoritmos que dividem um problema grande em dois menores, e assim sucessivamente. Se n é mil, $\log_2 n \approx 10$, se n é um milhão, $\log_2 n \approx 20$ (a base do algoritmo importa pouco).
- **linear** $T(n) = O(n)$ - algumas operações (fixas) são executadas para cada elemento. Cada vez que n dobra de tamanho, o tempo de execução também dobra

Classes de problemas mais comuns

- **log-linear** $T(n) = O(n \log n)$ - ocorre tipicamente quando um problema é quebrado em problemas menores, sendo que cada um deles é resolvido independentemente e depois juntando soluções. Quando n é um milhão, $n \log_2 n$ é 20 milhões. Para n 2 milhões, $n \log_2 n$ é 42 milhões
- **quadrático** $T(n) = O(n^2)$ - caso típico é um laço dentro do outro. Se n é mil, o número de operações é 1 milhão. Quando n dobra, o número de operações é multiplicado por 4
- **cúbico** $T(n) = O(n^3)$ - caso típico são três laços aninhados. Se n é cem, o número de operações é 1 milhão. Quando n dobra, o número de operações é multiplicado por 8

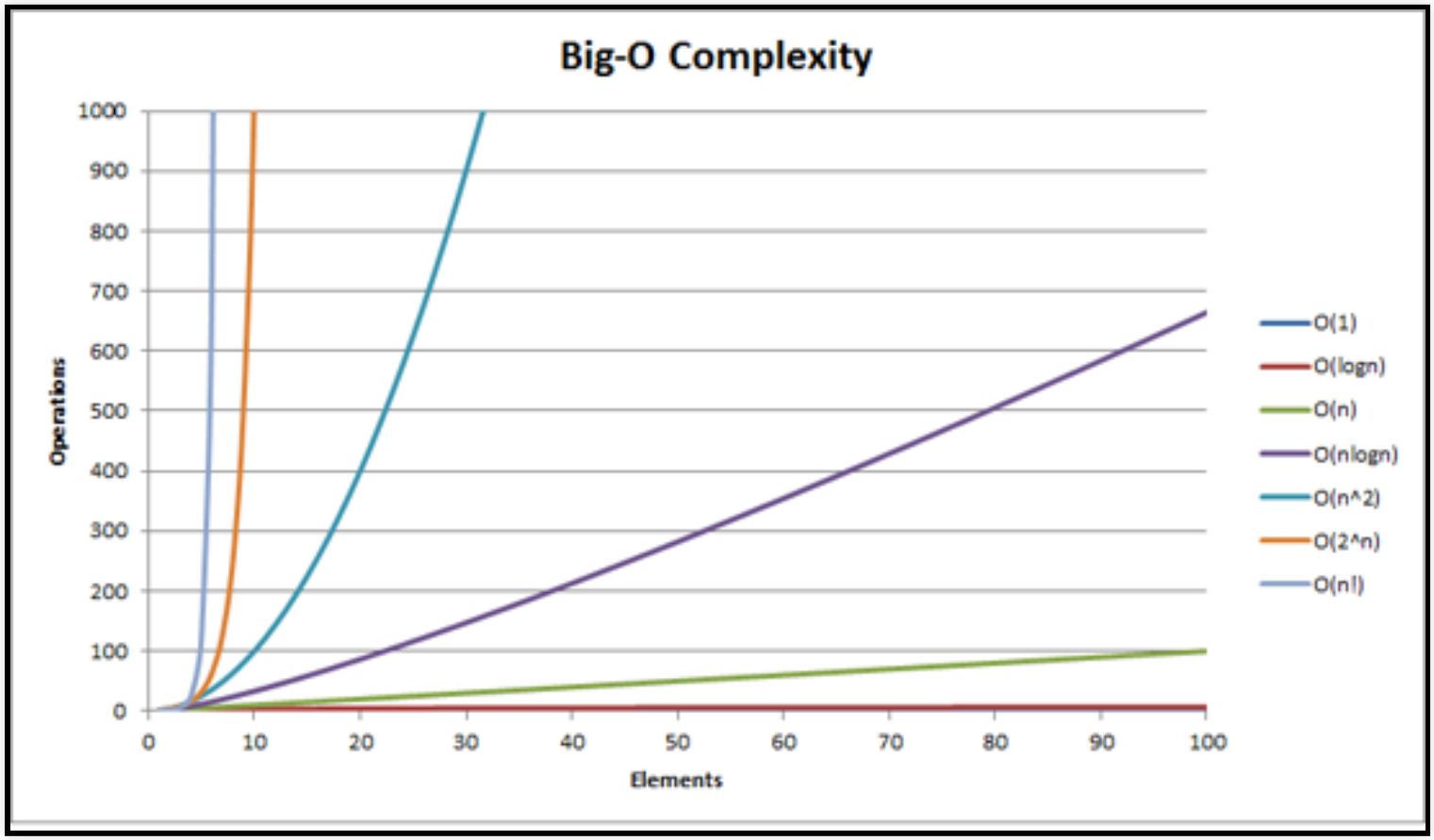
Classes de problemas mais comuns

- **exponencial** $T(n) = O(2^n)$ - usa *força bruta* para resolver um problema, e não tem aplicações práticas para n grande. Se $n = 20$, são necessárias na ordem de 1 milhão de operações. Quando n dobra, o tempo de execução é elevado ao quadrado
- **fatorial** $T(n) = O(n!)$ força bruta, mas muito pior que $O(2^n)$. Se n = 20, 20! = 2432902008176640000, se n = 40, 40! = 815915283247897734345611269596115894272000000000, se n = 60, 60! = 8320987112741390144276341183223364380754172606361245952449277696409600000000000000

Classes de complexidade

Função	Computador Atual	Computador 100x mais rápido	Computador 1000x mais rápido
$\log_{10} n$	N_1	$10^3 N_1$	$10^4 N_1$
n	N_2	$100 N_2$	$1000 N_2$
$n \log_{10} n$	N_3	$50 N_3$	$333 N_3$
n^2	N_4	$10 N_4$	$31.6 N_4$
n^3	N_5	$4.6 N_5$	$10 N_5$
2^n	N_6	$N_6 + 6$	$N_6 + 10$

Classes de Complexidade



Complexidade do problema versus complexidade do algoritmos

- Em análise de algoritmos, podemos fazer a análise de duas maneiras distintss:
  - **Análise de um algoritmo em particular:** Quantos recursos (tempo, memória) o algoritmo X precisa para rodar, em função do tamanho dos dados de entrada
  - **Análise de um problema:** Em que procuramos pelo limite inferior de recursos para resolver um determinado problema

Complexidade do problema versus complexidade do algoritmos

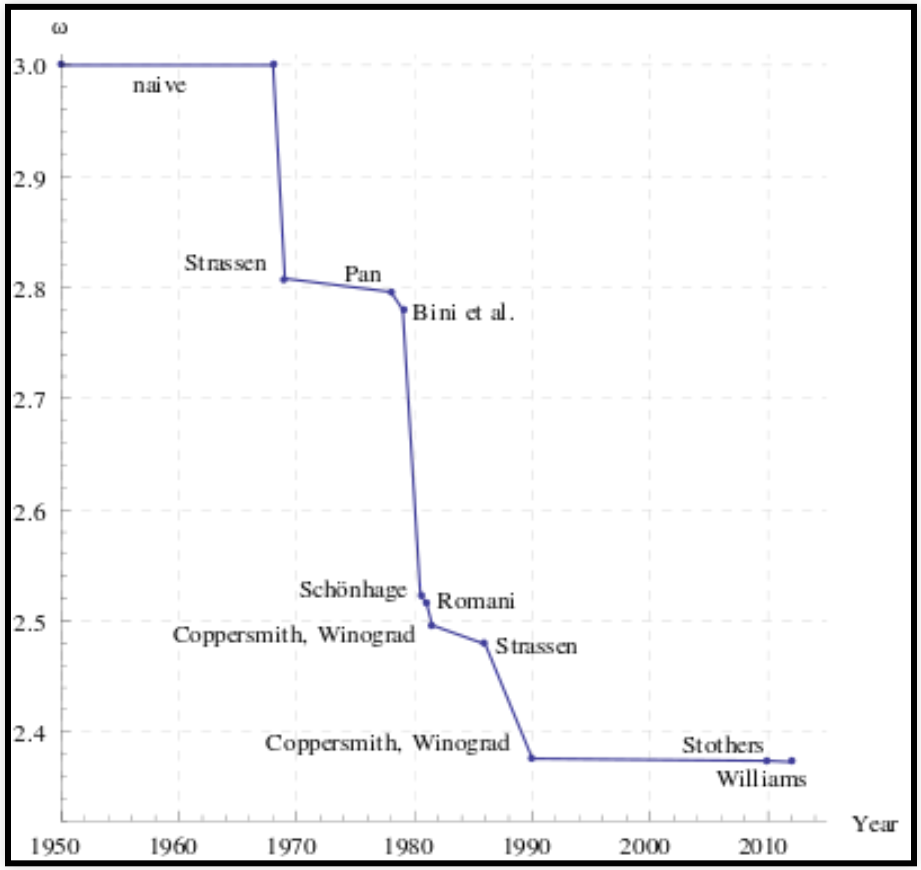
- Em alguns casos, as duas coisas estão ligadas: mesmo sem um resultado teórico, existe o algoritmo A para um problema, que consome x recursos, e A é o melhor algoritmo conhecido para aquele problema. Ou seja, não existe (ou é muito difícil) uma análise teórica, mas conhecemos um algorithmo que resolva com recursos x)
- Também existem casos em que existe um limite inferior, mesmo que não se conheça um algoritmo com o desempenho daquele limite inferior (multiplicação de matrizes é um exemplo). Nesse caso, existe um resultado teórico para o problema, mas um algoritmo que consumo no máximo aquele limite teórico não é conhecido

Limite superior

- A notação O também é utilizada para indicar **limites superiores** para problemas
- Por exemplo, dado o problema de multiplicação de duas matrizes $n \times n$
  - Conhecemos um algoritmo para resolver esse problema (pelo método trivial) de complexidade $O(n^3)$

Cota superior

- A **cota superior** é a complexidade do melhor algoritmo conhecido para aquele problema



Analogia com record mundial

- A cota superior para um problema é análogo ao recorde mundial para uma modalidade do atletismo
- Ele é estabelecido pelo melhor atleta (algoritmo) do momento
- Assim como o record mundial, a cota superior pode ser melhorada por algoritmo (atleta) mais veloz.

Limite inferior

- Às vezes é possível demonstrar que, para um problema, qualquer que seja o algoritmo, o problema requer pelo menos um certo número de operações.
- Para o problema de multiplicação de matrizes, apenas ler os elementos leva tempo $O(n^2)$
- Assim, uma cota inferior trivial é $\Omega(n^2)$

Algoritmo Ótimo

- Se um algoritmo tem uma complexidade igual à cota inferior do problema, ele é **assintoticamente ótimo** (ou simplesmente **ótimo**)