DCC
DEPARTAMENTO DE
CIÊNCIA DA COMPUTAÇÃO

UFMG
UNIVERSIDADE FEDERAL
DE MINAS GERAIS

Estruturas de Dados

Complexidade Assintótica

Professores: Anisio Lacerda
Wagner Meira Jr.

Quando a instância é grande demais

Considere a função de complexidade abaixo:

$$f(n) = 2n - 1$$

O que acontece quando *n* cresce?

Quando a instância é grande demais

Considere a função de complexidade abaixo:

$$f(n) = 2n \boxed{- 1}$$

O que acontece quando *n* cresce?

- Termos que não dependem de *n* se tornam irrelevantes

Quando a instância é grande demais

Considere a função de complexidade abaixo:

$$f(n) = 2n - 1$$

O que acontece quando *n* cresce?

- Termos que não dependem de *n* se tornam irrelevantes
- Podemos ignorar a constante que multiplica *n*?

Complexidade Assintótica

- Geralmente quando analisamos a eficiência de um algoritmo, estamos interessados em seu comportamento com relação ao **tamanho da entrada** de sua instância
- Observe as funções abaixo. Mesmo que suas angulações sejam diferentes, ambas crescem de forma **linear** com relação à variável *n*.

$$f(n) = 2n - 1$$

$$g(n) = 3n + 4$$

Complexidade Assintótica

- A ideia geral gira em torno de encontrar funções que **limitem** a função de complexidade do nosso algoritmo. Iremos estudar três notações:
  - A notação O (“big oh”)
  - A notação Ω (“big omega”)
  - A notação θ (“theta”)

DCC

Notação *O*

Dada uma função *f(n)* definimos formalmente o conjunto *O(f(n))* como:

$$O(f(n)) := \{g(n) : \exists c, m > 0 \text{ t.q. } g(n) \leq cf(n), \forall n \geq m\}$$

Notação *O*

Dada uma função *f(n)* definimos formalmente o conjunto *O(f(n))* como:

$$O(f(n)) := \{g(n) : \exists c, m > 0 \text{ t.q. } g(n) \leq cf(n), \forall n \geq m\}$$

Mas o que isso significa?

Notação *O*

Dada uma função *f(n)* definimos formalmente o conjunto *O(f(n))* como:

$$O(f(n)) := \{g(n) : \exists c, m > 0 \text{ t.q. } g(n) \leq cf(n), \forall n \geq m\}$$

Esses são os elementos que compõem o conjunto, *g(n)* é uma função, o que indica que *O(f(n))* é um **conjunto de funções**.

Notação *O*

Dada uma função *f(n)* definimos formalmente o conjunto *O(f(n))* como:

$$O(f(n)) := \{\underline{g(n)} : \exists c, m > 0 \text{ t.q. } g(n) \leq cf(n), \forall n \geq m\}$$

O nosso interesse é enquadrar a função de complexidade do nosso algoritmo em algum desses conjuntos!

Notação *O*

Dada uma função *f(n)* definimos formalmente o conjunto *O(f(n))* como:

$$O(f(n)) := \{g(n) : \exists c, m > 0 \text{ t.q. } g(n) \leq cf(n), \forall n \geq m\}$$

Para pertencer ao conjunto *O(f(n))* nossa função *g(n)* deve satisfazer a essa condição.

Vamos entender ela melhor.

Notação *O*

Dada uma função *f(n)* definimos formalmente o conjunto *O(f(n))* como:

$$O(f(n)) := \{g(n) : \exists c, m > 0 \text{ t.q. } g(n) \leq cf(n), \forall n \geq m\}$$

Existem constantes positivas *c* e *m*.

(Vamos ver que boa parte do trabalho está em determinar estas constantes!)

Notação *O*

Dada uma função *f(n)* definimos formalmente o conjunto *O(f(n))* como:

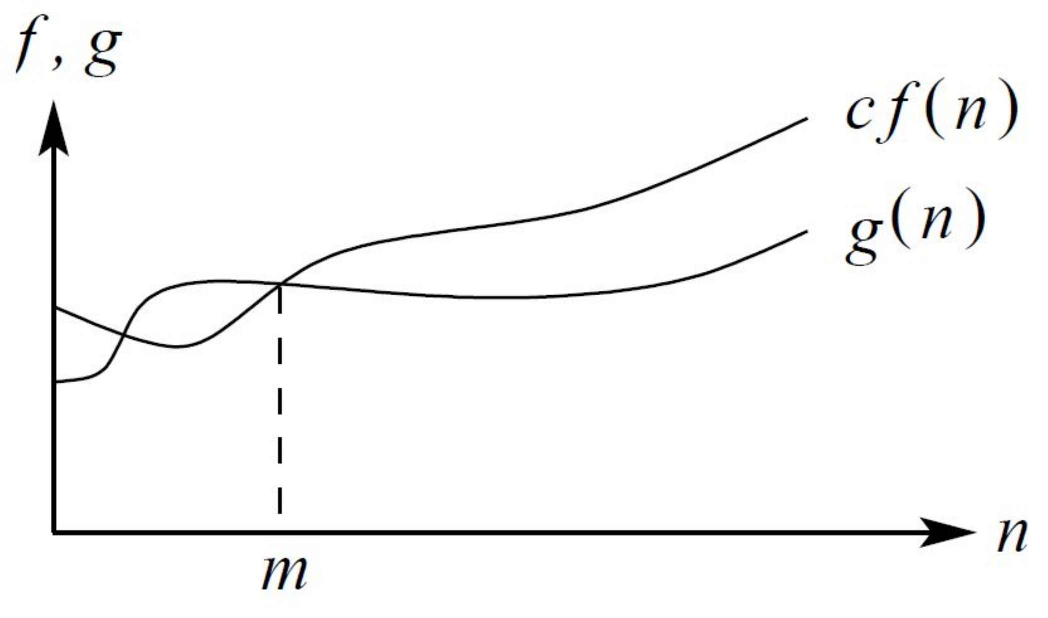
$$O(f(n)) := \{g(n) : \exists c, m > 0 \text{ t.q. } g(n) \leq cf(n), \forall n \geq m\}$$

Se o tamanho da entrada *n* for maior que a constante *m*, então nossa função de complexidade *g(n)* deve ser menor ou igual que *cf(n)*.

Notação O

Dada uma função *f(n)* definimos formalmente o conjunto *O(f(n))* como:

$$O(f(n)) := \{g(n) : \exists c, m > 0 \text{ t.q. } g(n) \leq cf(n), \forall n \geq m\}$$



Notação *O*

$$O(f(n)) := \{g(n) : \exists c, m > 0 \text{ t.q. } g(n) \leq cf(n), \forall n \geq m\}$$

Exemplo 1:

Mostre que *f(n)* = *2n* pertence ao conjunto *O(n)*.

Notação O

$$O(f(n)) := \{g(n) : \exists c, m > 0 \text{ t.q. } g(n) \leq cf(n), \forall n \geq m\}$$

Exemplo 1:

Mostre que $f(n) = 2n$ pertence ao conjunto $O(n)$.

- Tome $c = 2$ e $m = 1$.
- $2n \leq 2n$ para todo $n \geq 1$.

Notação O

$$O(f(n)) := \{g(n) : \exists c, m > 0 \text{ t.q. } g(n) \leq cf(n), \forall n \geq m\}$$

Exemplo 2:

Mostre que $f(n) = n$ pertence ao conjunto $O(n^2)$.

Notação *O*

$$O(f(n)) := \{g(n) : \exists c, m > 0 \text{ t.q. } g(n) \leq cf(n), \forall n \geq m\}$$

Exemplo 2:

Mostre que $f(n) = n$ pertence ao conjunto $O(n^2)$.

- Tome $c = 1$ e $m = 1$.
- $n \leq n^2$ para todo $n \geq 1$.

Notação *O*

Exemplo 3:

Mostre que $f(n) = n^2$ pertence ao conjunto $O(n)$.

Notação *O*

Exemplo 3:

Mostre que $f(n) = n^2$ pertence ao conjunto $O(n)$.

Falso!

- Suponha que existam as constantes.
- Logo $n^2 \leq cn$ para todo $n \geq m$.
- Dividindo ambos lado por n.
- Temos $n \leq c$ para todo $n \geq m$.

Notação O

Exemplo 3:

Mostre que $f(n) = n^2$ pertence ao conjunto $O(n)$.

Falso!

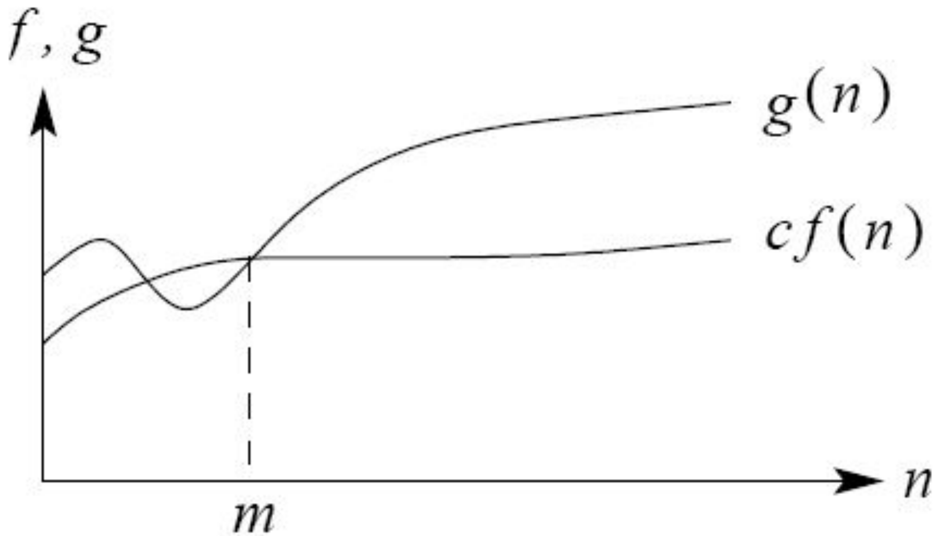
- Suponha que existam as constantes.
- Logo $n^2 \leq cn$ para todo $n \geq m$.
- Dividindo ambos lado por n.
- Temos $n \leq c$ para todo $n \geq m$.

Contradição!

Notação Ω

Dada uma função *f(n)* definimos formalmente o conjunto *Ω(f(n))* como:

$$\Omega(f(n)) := \{g(n) : \exists c, m > 0 \text{ t.q. } g(n) \geq cf(n), \forall n \geq m\}$$



Notação Ω

$$\Omega(f(n)) := \{g(n) : \exists c, m > 0 \text{ t.q. } g(n) \geq cf(n), \forall n \geq m\}$$

Exemplo:

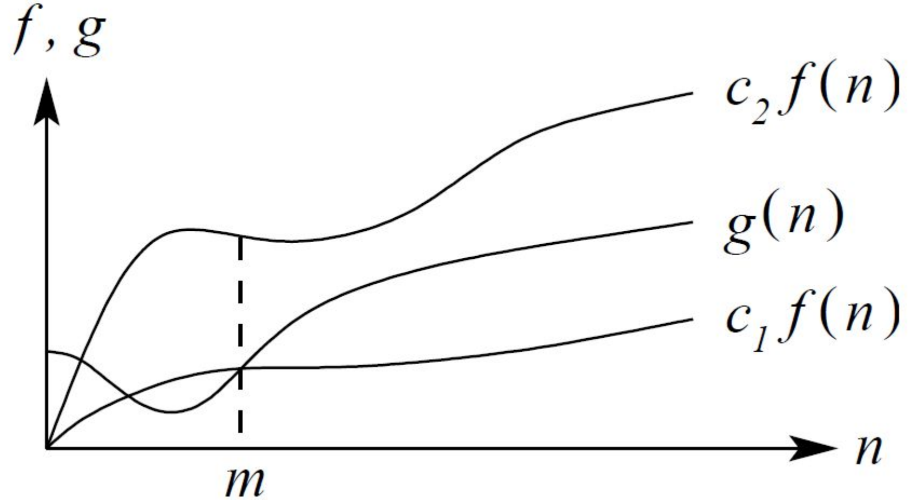
Mostre que $f(n) = 3n^3 + n^2$ pertence ao conjunto $\Omega(n^3)$:

- Tome $c = 1$ e $m = 1$.
- Logo $3n^3 + n^2 \geq n^3$, para todo $n \geq 1$.
- Subtraindo n^3 dos dois lados.
- Temos $2n^3 + n^2 \geq 0$, para todo $n \geq 1$.

Notação θ

Dada uma função *f(n)* definimos formalmente o conjunto *θ(f(n))* como:

$$\Theta(f(n)) := \{g(n) : \exists c_1, c_2, m > 0 \text{ t.q.}$$
$$c_1 f(n) \leq g(n) \leq c_2 f(n), \forall n \geq m\}$$



Notação θ

$$\Theta(f(n)) := \{g(n) : \exists c_1, c_2, m > 0 \text{ t.q. } c_1 f(n) \le g(n) \le c_2 f(n), \forall n \ge m\}$$

Exemplo:

Mostre que $f(n) = n^2 + 400n$, pertence a $\theta(n^2)$

- Tome $c_1 = 1$ e $m_1 = 1$.
- Temos $n^2 \le n^2 + 400n$, para todo $n \ge 1$.
- Subtraia n^2 dos dois lados.
- Temos então $0 \le 400n$, para todo $n \ge 1$.

Notação θ

$$\Theta(f(n)) := \{g(n) : \exists c_1, c_2, m > 0 \text{ t.q.}$$
$$c_1 f(n) \leq g(n) \leq c_2 f(n), \forall n \geq m\}$$

Exemplo:

Mostre que $f(n) = n^2 + 400n$, pertence a $\theta(n^2)$

- Tome $c_2 = 2$ e $m_2 = 400$.
- Temos $n^2 + 400n \leq 2n^2$, para todo $n \geq 400$.
- Subtraia n^2 dos dois lados.
- Temos então $400n \leq n^2$, para todo $n \geq 400$.

Notação θ

$$\Theta(f(n)) := \{g(n) : \exists c_1, c_2, m > 0 \text{ t.q. } c_1 f(n) \leq g(n) \leq c_2 f(n), \forall n \geq m\}$$

Exemplo:

Mostre que $f(n) = n^2 + 400n$, pertence a $\theta(n^2)$

- Queremos que as duas afirmações valham.
- Então tome $m = max(m_1, m_2)$.
- Concluindo: tome $c_1 = 1$, $c_2 = 2$ e $m = 400$.
- Temos $n^2 \leq n^2 + 400n \leq 2n^2$, para todo $n \geq 400$.

Notação θ

- Note que a definição da notação θ soa como se estivéssemos utilizando simultaneamente as notações O e Ω.

Teorema: Sejam $f(n)$ e $g(n)$ funções. Temos que $f(n) = \theta(g(n))$ se e somente se $f(n) = O(g(n))$ e $f(n) = \Omega(g(n))$.

- A ideia por trás deste teorema é intuitiva e é um bom exercício de fixação demonstrá-lo!

Complexidade Assintótica

Observação importante!

- Tanto O, Ω e θ são famílias de conjuntos! Formalmente os operadores apropriados são os de conjuntos (∈, ⊆, etc), no entanto muitas vezes abusamos da notação e utilizamos símbolos de operadores aritméticos (+, =) enquanto operamos com eles.

Algumas propriedades da notação *O*

$$
\begin{aligned}
f(n) &= O(f(n)) \\
c \times O(f(n)) &= O(f(n)) \qquad c = constante \\
O(f(n)) + O(f(n)) &= O(f(n)) \\
O(O(f(n))) &= O(f(n)) \\
O(f(n)) + O(g(n)) &= O(max(f(n), g(n))) \\
O(f(n))O(g(n)) &= O(f(n)g(n)) \\
f(n)O(g(n)) &= O(f(n)g(n))
\end{aligned}
$$

Complexidade Assintótica

- É importante notar que apesar de mais utilizada, a notação O é mais fraca que a θ
  - $f(n) = \theta(g(n))$ implica em $f(n) = O(g(n))$, mas o contrário não é válido.
  - Em termos de conjuntos: $\theta(g(n)) \subseteq O(g(n))$

- Não é surpreendente que a notação O seja a mais utilizada, uma vez que sua intuição nos remete ao pior caso de execução de um algoritmo.

Alguns casos recorrentes

- A notação *O* nos dá uma família de conjuntos. É de se imaginar que ao longo da história da análise de complexidade, alguns desses conjuntos tenham aparecido mais que outros.

- Vamos chamar esses conjuntos de **classes de comportamento assintótico**, ver algumas delas e tentar entender como é a “cara” de um algoritmo que cairia nessa classe.

Classes de comportamento assintótico

Constante ou O(1):

- Um número fixo de instruções é executada.
- Não dependem do tamanho da entrada

Exemplo:

  - Máximo de dois inteiros
  - Tabela Hash

Classes de comportamento assintótico

Logarítmico ou O(log n):

- Tipicamente são algoritmos que quebram o problema em problemas menores
- A função log cresce bem devagar.

Classes de comportamento assintótico

Exemplo artificial de algoritmo O(log n):

- Recebe como entrada um vetor com *n* inteiros.
  - Imprima o primeiro elemento do vetor.
  - Se n > 1, divida o vetor em dois vetores de tamanho n/2 e chame a função recursivamente para segunda metade.

Esse algoritmo possui complexidade *O(log n)*.

Classes de comportamento assintótico

Exemplos de algoritmos O(log n):

- Busca binária.
- Exponenciação.
- Encontrar o *n*-ésimo termo da sequencia de fibonacci.

Classes de comportamento assintótico

Linear ou O(n):

- Algoritmos que realizam uma quantidade constante de operações para cada "elemento" da entrada.

  Exemplos:

  - Os algoritmos vistos na aula passada!

Classes de comportamento assintótico

O(n log n):

- Típico de algoritmos que utilizam um paradigma chamado divisão e conquista, quebrando o problema em problemas menores e depois "juntando" as soluções de alguma forma.

Classes de comportamento assintótico

Exemplo artificial de algoritmo O(n log n):

- Recebe como entrada um vetor com *n* inteiros.
  - Imprima o maior elemento do vetor.
  - Se n > 1, divida o vetor em dois vetores de tamanho n/2 e chame a função recursivamente para as duas metades.

Esse algoritmo possui complexidade *O(n log n)*.

Classes de comportamento assintótico

Exemplos de algoritmos O(n log n):

- Merge Sort.
- Heap Sort.
- Quick Sort.
- Dijkstra (caso o grafo de entrada seja esparso)

Classes de comportamento assintótico

Quadráticos ou O(n^2):

- Geralmente ocorre quando os "elementos" são processados aos pares, muitas vezes tendo dois laços de repetição aninhados

  Exemplos:
  - Percorrer uma matriz n x n.
  - Ordenação por seleção.
  - Maior Subsequência comum (LCS).

Classes de comportamento assintótico

Cúbicos ou $O(n^3)$:

- Úteis para resolver problemas com instâncias pequenas

  Exemplos:
  - Multiplicação de matrizes.
  - Decomposição de Cholesky.
  - Algoritmo de Bellman-Ford.
  - Algoritmo de Floyd Warshall.

Classes de comportamento assintótico

O(2^n):

- São algoritmos que envolvem testar todas escolhas binárias em um conjunto de *n* elementos. Tipicamente essa escolha binária se parece com *"este elemento está ou não na minha solução?"*.

Classes de comportamento assintótico

O(2^n):

Exemplos:

- Problema da satisfabilidade.
- Problema da Mochila 0/1.
- *Subset Sum*.
- Problema da partição.
- Conjunto Independente Máximo.

Classes de comportamento assintótico

O(n!):

- Geralmente são algoritmos que envolvem testar todas as permutações de um conjunto com *n* elementos.

  Exemplos:
  - Problema do caixeiro viajante.
  - Caminho Hamiltoniano.
  - Ciclo Hamiltoniano.

Classes de comportamento assintótico

É possível piorar?

- Bogosort $O(n\ n!)$

```
void algoritmo(int *v, int n){
  while not inOrder(v){
      shuffle(v);
    }
}
```

- Classes como $O(n^n)$, $O(n!^n)$, $O(n!^{n!})$, …

Classes de comportamento assintótico

- **Algoritmo exponencial** no tempo de execução tem ordem de complexidade maior que $O(c^n); c > 1$.

- **Algoritmo polinomial** no tempo de execução tem ordem de complexidade $O(p(n))$, onde $p(n)$ é um polinômio em n.

Classes de comportamento assintótico

- Algoritmos exponenciais são geralmente simples variações de pesquisa exaustiva.

- Algoritmos polinomiais são geralmente obtidos mediante entendimento mais profundo da estrutura do problema.

Classes de comportamento assintótico

- A distinção entre estes dois tipos de algoritmos torna-se significativa quando o tamanho do problema a ser resolvido cresce

- Um problema é considerado:
  - **intratável**: se não se conhece um algoritmo polinomial para resolvê-lo.
  - **bem resolvido**: quando existe um algoritmo polinomial para resolvê-lo.

Classes de comportamento assintótico

- Quando dois algoritmos caem no mesma classe de comportamento assintótico, podemos considerar que eles são equivalentes em algum sentido.
  - Conduzir uma análise experimental em sistemas reais.
  - Fazer uma análise teórica mais cuidadosa (se possível).
  - Observar a complexidade assintótica de outro parâmetro de desempenho.