BCC202 - Estruturas de Dados I

Aula 06: Análise de Algoritmos (Parte II)

Pedro Silva

Universidade Federal de Ouro Preto, UFOP
Departamento de Computação, DECOM
Email: **silvap@ufop.edu.br**

Conteúdo

Conteúdo

Função de Complexidade

- Na aula passada aprendemos a calcular a **função de complexidade** $f(n)$.
- Observações importantes:
  - Para valores pequenos de n, praticamente qualquer algoritmo custa pouco para ser executado.
  - **Logo**: a escolha do algoritmo tem pouquíssima influência em problemas de tamanho pequeno.

Comportamento Assintótico

- A análise de algoritmos deve ser realizada para valores grandes de n.
- Para isso, estuda-se o **comportamento assintótico** das funções de custo.
  - Comportamento das funções para valores grandes de n.
- O **comportamento assintótico** de $f(n)$ representa o limite do comportamento do custo quando n cresce.
- É importante lembrar que as definições de notações são independentes da análise de algoritmos, podendo ser utilizados para outros fins.

Comportamento Assintótico

- A análise de algoritmos deve ser realizada para valores grandes de n.
- Para isso, estuda-se o **comportamento assintótico** das funções de custo.
  - Comportamento das funções para valores grandes de n.
- O **comportamento assintótico** de $f(n)$ representa o limite do comportamento do custo quando n cresce.
- É importante lembrar que as definições de notações são independentes da análise de algoritmos, podendo ser utilizados para outros fins.

Comportamento Assintótico

- A análise de algoritmos deve ser realizada para valores grandes de n.
- Para isso, estuda-se o **comportamento assintótico** das funções de custo.
  - Comportamento das funções para valores grandes de n.
- O **comportamento assintótico** de $f(n)$ representa o limite do comportamento do custo quando n cresce.
- É importante lembrar que as definições de notações são independentes da análise de algoritmos, podendo ser utilizados para outros fins.

Comportamento Assintótico

- A análise de algoritmos deve ser realizada para valores grandes de n.
- Para isso, estuda-se o **comportamento assintótico** das funções de custo.
  - Comportamento das funções para valores grandes de n.
- O **comportamento assintótico** de $f(n)$ representa o limite do comportamento do custo quando n cresce.
- É importante lembrar que as definições de notações são independentes da análise de algoritmos, podendo ser utilizados para outros fins.

Crescimento e Domínio Assintótico

- A análise de um algoritmo geralmente conta com apenas algumas operações elementares.
- A **medida de custo**, ou **medida de complexidade**, relata o **crescimento assintótico** da operação considerada.
- **Definição**: Uma função $f(n)$ **domina assintoticamente** outra função $g(n)$ se:
  - Existem duas constantes positivas c e m tais que, para $n >= m$, temos $|g(n)| <= c|f(n)|$.

Crescimento e Domínio Assintótico

- A análise de um algoritmo geralmente conta com apenas algumas operações elementares.
- A **medida de custo**, ou **medida de complexidade**, relata o **crescimento assintótico** da operação considerada.
- **Definição**: Uma função $f(n)$ **domina assintoticamente** outra função $g(n)$ se:
  - Existem duas constantes positivas c e m tais que, para $n >= m$, temos $|g(n)| <= c|f(n)|$.

Crescimento e Domínio Assintótico

- A análise de um algoritmo geralmente conta com apenas algumas operações elementares.
- A **medida de custo**, ou **medida de complexidade**, relata o **crescimento assintótico** da operação considerada.
- **Definição**: Uma função $f(n)$ **domina assintoticamente** outra função $g(n)$ se:
  - Existem duas constantes positivas c e m tais que, para $n >= m$, temos $|g(n)| <= c|f(n)|$.

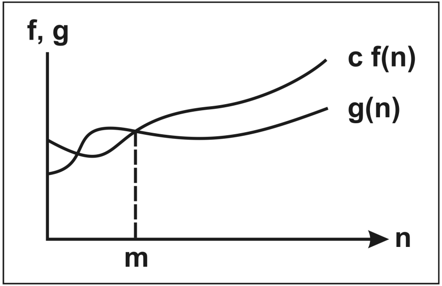
Crescimento e Domínio Assintótico

- A análise de um algoritmo geralmente conta com apenas algumas operações elementares.
- A **medida de custo**, ou **medida de complexidade**, relata o **crescimento assintótico** da operação considerada.
- **Definição**: Uma função $f(n)$ **domina assintoticamente** outra função $g(n)$ se:
  - Existem duas constantes positivas c e m tais que, para $n >= m$, temos $|g(n)| <= c|f(n)|$.

Conteúdo

Dominação Assintótica

- $f(n)$ **domina assintoticamente** $g(n)$ se:
  - Existem duas constantes positivas c e m tais que, para $n >= m$, temos $|g(n)| <= c|f(n)|$.



Dominação Assintótica: Exemplo

- Sejam $g(n) = (n+1)^2$ e $f(n) = n^2$.
- As funções $g(n)$ e $f(n)$ dominam assintoticamente uma à outra, desde que:
  - $|(n+1)^2| <= 4|n^2|$, para $n >= 1$.

    $|g(n)| <= c|f(n)|$ para $n >= m$; $c = 4$ e $m = 1$
  - $|n^2| <= |(n+1)^2|$, para $n >= 0$.

    $|f(n)| <= c|g(n)|$ para $n >= m$; $c = 1$ e $m = 1$

Intuitivamente

O(f(n))

De maneira informal, quando o tamanho da entrada é "arbitrariamente" grande:

- temos funções que não crescem mais rápido que $f(n)$.
- temos funções menores ou iguais a um múltipo de $f(n)$.

Exemplo: n^2, $3/2n^2$, $100000n^2$, $n^2/400000$ crescem todas com a mesma velocidade, são todas $O(n^2)$.

- *A ordem de crescimento é definida pelo termo de maior ordem, "desconsiderando" as constantes.*
  - $0,00000001n^3 + 4000000000n^2$ não é $O(n^2)$.

Intuitivamente

O(f(n))

De maneira informal, quando o tamanho da entrada é "arbitrariamente" grande:

- temos funções que não crescem mais rápido que $f(n)$.
- temos funções menores ou iguais a um múltipo de $f(n)$.

Exemplo: n^2, $3/2n^2$, $100000n^2$, $n^2/400000$ crescem todas com a mesma velocidade, são todas $O(n^2)$.

- *A ordem de crescimento é definida pelo termo de maior ordem, "desconsiderando" as constantes.*
  - $0,00000001n^3 + 4000000000n^2$ não é $O(n^2)$.

Notação O

- Escrevemos $g(n) = O(f(n))$ para expressar que $f(n)$ domina assintoticamente $g(n)$.
  - Lê-se $g(n)$ é da ordem de no máximo $f(n)$.

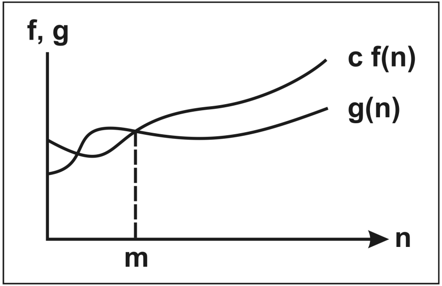
- Exemplo:
  - Quando dizemos que o tempo de execução $T(n)$ de um programa é $O(n^2)$, significa que existem constantes c e m tais que, para valores de $n >= m$, $T(n) <= cn^2$.

Notação O

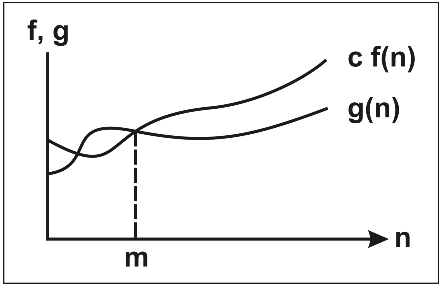
- Escrevemos $g(n) = O(f(n))$ para expressar que $f(n)$ domina assintoticamente $g(n)$.
  - Lê-se $g(n)$ é da ordem de no máximo $f(n)$.
- Exemplo:
  - Quando dizemos que o tempo de execução $T(n)$ de um programa é $O(n^2)$, significa que existem constantes c e m tais que, para valores de $n >= m$, $T(n) <= cn^2$.

Notação O

- Exemplo gráfico de dominação assintótica que ilustra a notação O.
  - Abaixo, a função $f(n)$ domina assintoticamente a função $g(n)$.



Notação O



- O valor da constante m mostrado é o menor valor possível, mas qualquer valor maior também é válido.
- **Definição**: uma função $g(n)$ é $O(f(n))$ se existem duas constantes positivas c e m tais que $g(n) <= cf(n)$, para todo $n >= m$.

Operações

$$f(n) = O(f(n))$$

$$c * O(f(n)) = O(f(n)), \text{ c = constante}$$

$$O(f(n)) + O(f(n)) = O(f(n))$$

$$O(O(f(n))) = O(f(n))$$

$$O(f(n)) + O(g(n)) = O(max(f(n), g(n)))$$

$$O(f(n)) * O(g(n)) = O(f(n) * g(n))$$

$$f(n) * O(g(n)) = O(f(n) * g(n))$$

Exemplo 01: $g(n) = (n+1)^2$ e $f(n) = n^2$

- $\boldsymbol{g(n)}$ é $\boldsymbol{O(n^2)}$ quando $\boldsymbol{m = 1}$ e $\boldsymbol{c = 4}$.
  - Isto porque sabe-se que $\boldsymbol{(n+1)^2 <= 4n^2}$.

Ou seja, existem as constantes positivas $\boldsymbol{c}$ e $\boldsymbol{m}$ tal que:

$$\boldsymbol{g(n) <= cf(n)}, \text{ para } \boldsymbol{n >= m}.$$

Exemplo 01: $g(n) = (n+1)^2$ e $f(n) = n^2$

- $g(n)$ é $O(n^2)$ quando $m = 1$ e $c = 4$.
  - Isto porque sabe-se que $(n+1)^2 <= 4n^2$.

Ou seja, existem as constantes positivas c e m tal que:

$$g(n) <= cf(n), \text{ para } n >= m.$$

Exemplo 02: $g(n) = n$ e $f(n) = n^2$

- Sabemos que $g(n)$ é $O(n^2)$, pois para $n >= 1$, $n <= n^2$.
- Entretanto $f(n)$ não é $O(n)$.
- Suponha que existam constantes c e m tais que para todo $n >= m$, $n^2 <= cn$.
  - Se $c >= n$ para qualquer $n >= m$, então deveria existir um valor para c que pudesse ser maior ou igual a n para todo n.

Portanto, não existe a constante positiva c tal que $g(n) <= cf(n)$, para $n >= m$.

Exemplo 02: $g(n) = n$ **e** $f(n) = n^2$

- Sabemos que $g(n)$ é $O(n^2)$, pois para $n >= 1$, $n <= n^2$.
- Entretanto $f(n)$ não é $O(n)$.
- Suponha que existam constantes c e m tais que para todo $n >= m$, $n^2 <= cn$.
  - Se $c >= n$ para qualquer $n >= m$, então deveria existir um valor para c que pudesse ser maior ou igual a n para todo n.

Portanto, não existe a constante positiva c tal que $g(n) <= cf(n)$, para $n >= m$.

Exemplo 02: $g(n) = n$ e $f(n) = n^2$

- Sabemos que $\boldsymbol{g(n)}$ é $\boldsymbol{O(n^2)}$, pois para $\boldsymbol{n >= 1}$, $\boldsymbol{n <= n^2}$.
- Entretanto $\boldsymbol{f(n)}$ não é $\boldsymbol{O(n)}$.
- Suponha que existam constantes $\boldsymbol{c}$ e $\boldsymbol{m}$ tais que para todo $\boldsymbol{n >= m}$, $\boldsymbol{n^2 <= cn}$.
  - Se $\boldsymbol{c >= n}$ para qualquer $\boldsymbol{n >= m}$, então deveria existir um valor para $\boldsymbol{c}$ que pudesse ser maior ou igual a $\boldsymbol{n}$ para todo $\boldsymbol{n}$.

Portanto, não existe a constante positiva $\boldsymbol{c}$ tal que $\boldsymbol{g(n) <= cf(n)}$, para $\boldsymbol{n >= m}$.

Exemplo 02: $g(n) = n$ e $f(n) = n^2$

- Sabemos que $\boldsymbol{g(n)}$ é $\boldsymbol{O(n^2)}$, pois para $\boldsymbol{n >= 1}$, $\boldsymbol{n <= n^2}$.
- Entretanto $\boldsymbol{f(n)}$ não é $\boldsymbol{O(n)}$.
- Suponha que existam constantes $\boldsymbol{c}$ e $\boldsymbol{m}$ tais que para todo $\boldsymbol{n >= m}$, $\boldsymbol{n^2 <= cn}$.
  - Se $\boldsymbol{c >= n}$ para qualquer $\boldsymbol{n >= m}$, então deveria existir um valor para $\boldsymbol{c}$ que pudesse ser maior ou igual a $\boldsymbol{n}$ para todo $\boldsymbol{n}$.

Portanto, não existe a constante positiva $\boldsymbol{c}$ tal que $\boldsymbol{g(n) <= cf(n)}$, para $\boldsymbol{n >= m}$.

Exemplo 02: $g(n) = n$ e $f(n) = n^2$

- Sabemos que $\boldsymbol{g(n)}$ é $\boldsymbol{O(n^2)}$, pois para $\boldsymbol{n >= 1}$, $\boldsymbol{n <= n^2}$.
- Entretanto $\boldsymbol{f(n)}$ não é $\boldsymbol{O(n)}$.
- Suponha que existam constantes $\boldsymbol{c}$ e $\boldsymbol{m}$ tais que para todo $\boldsymbol{n >= m}$, $\boldsymbol{n^2 <= cn}$.
  - Se $\boldsymbol{c >= n}$ para qualquer $\boldsymbol{n >= m}$, então deveria existir um valor para $\boldsymbol{c}$ que pudesse ser maior ou igual a $\boldsymbol{n}$ para todo $\boldsymbol{n}$.

Portanto, não existe a constante positiva $\boldsymbol{c}$ tal que $\boldsymbol{g(n) <= cf(n)}$, para $\boldsymbol{n >= m}$.

Exemplo 03: $g(n) = 3n^3 + 2n^2 + n$

- Sabemos que
  - $\boldsymbol{g(n)}$ é $\boldsymbol{O(n^3)}$.
  - $\boldsymbol{g(n)}$ também é $\boldsymbol{O(n^4)}$.
    - Entretanto, esta afirmação é **mais fraca** do que dizer que g(n) é $\boldsymbol{O(n^3)}$.
- $\boldsymbol{g(n)}$ também é $\boldsymbol{O(n^{40})}$?

Exemplo 03: $g(n) = 3n^3 + 2n^2 + n$

- Sabemos que
  - $\boldsymbol{g(n)}$ é $\boldsymbol{O(n^3)}$.
  - $\boldsymbol{g(n)}$ também é $\boldsymbol{O(n^4)}$.
    - Entretanto, esta afirmação é **mais fraca** do que dizer que g(n) é $\boldsymbol{O(n^3)}$.
- $\boldsymbol{g(n)}$ também é $\boldsymbol{O(n^{40})}$?

Exemplo 03: $g(n) = 3n^3 + 2n^2 + n$

- Sabemos que
  - $\boldsymbol{g(n)}$ é $\boldsymbol{O(n^3)}$.
  - $\boldsymbol{g(n)}$ também é $\boldsymbol{O(n^4)}$.
    - Entretanto, esta afirmação é **mais fraca** do que dizer que g(n) é $\boldsymbol{O(n^3)}$.
- $\boldsymbol{g(n)}$ também é $\boldsymbol{O(n^{40})}$?

Exemplo 03: $g(n) = 3n^3 + 2n^2 + n$

- Sabemos que
  - $\boldsymbol{g(n)}$ é $\boldsymbol{O(n^3)}$.
  - $\boldsymbol{g(n)}$ também é $\boldsymbol{O(n^4)}$.
    - Entretanto, esta afirmação é **mais fraca** do que dizer que g(n) é $\boldsymbol{O(n^3)}$.
- $\boldsymbol{g(n)}$ também é $\boldsymbol{O(n^{40})}$?

Exemplo 03: $g(n) = 3n^3 + 2n^2 + n$

- Sabemos que
  - $\boldsymbol{g(n)}$ é $\boldsymbol{O(n^3)}$.
  - $\boldsymbol{g(n)}$ também é $\boldsymbol{O(n^4)}$.
    - Entretanto, esta afirmação é **mais fraca** do que dizer que g(n) é $\boldsymbol{O(n^3)}$.
- $\boldsymbol{g(n)}$ também é $\boldsymbol{O(n^{40})}$?

Sim! É fácil mostrar que existem as constantes positivas $\boldsymbol{c}$ e $\boldsymbol{m}$ tal que:

- $\boldsymbol{g(n) <= cf(n)}$, para $\boldsymbol{n >= m}$.

Exemplo 04: $g(n) = log_5 n$ é $O(log\ n)$

- Recorrendo às propriedades logaritmicas, a mudança de base é definida por: $\boldsymbol{log_a x = \frac{log_b x}{log_b a}}$.

- Assim, observa-se que:

$$\boldsymbol{log_5 n = log_5 2 * log\ n}.$$

Logo, $\boldsymbol{log_5 2}$ é a constante $\boldsymbol{c}$, e será fácil encontrar um $\boldsymbol{m}$ que comprove que $\boldsymbol{g(n)}$ é $\boldsymbol{O(log\ n)}$.

Exemplo 04: $g(n) = log_5 n$ é $O(log\ n)$

- Recorrendo às propriedades logaritmicas, a mudança de base é definida por: $\boldsymbol{log_a x = \frac{log_b x}{log_b a}}$.

- Assim, observa-se que:

$$\boldsymbol{log_5 n = log_5 2 * log\ n}.$$

Logo, $\boldsymbol{log_5 2}$ é a constante $\boldsymbol{c}$, e será fácil encontrar um $\boldsymbol{m}$ que comprove que

$$\boldsymbol{g(n)} \text{ é } \boldsymbol{O(log\ n)}.$$

Exemplo 04: $g(n) = log_5 n$ é $O(log\ \ n)$

- Recorrendo às propriedades logaritmicas, a mudança de base é definida por: $\boldsymbol{log_a x = \frac{log_b x}{log_b a}}$.

- Assim, observa-se que:

$$\boldsymbol{log_5 n = log_5 2 * log\ \ n}.$$

Logo, $\boldsymbol{log_5 2}$ é a constante $\boldsymbol{c}$, e será fácil encontrar um $\boldsymbol{m}$ que comprove que $\boldsymbol{g(n)}$ é $\boldsymbol{O(log\ \ n)}$.

Generalizando

$\boldsymbol{log_b n = log_b c * log_c n}$. Logo, a constante $\boldsymbol{c}$ será $\boldsymbol{log_b c}$ e deverá ser definida a constante $\boldsymbol{m}$ que comprove que $\boldsymbol{log_b n}$ é $\boldsymbol{O(log_c n)}$.

Exemplo 05: Ordem de complexidade do MaxMin1

```
int MaxMin1(int* A, int n, int* pMax, int* pMin) {
int i;
  *pMax = A[0];
  *pMin = A[0];
  for(i = 1; i < n; i++) {
    if(*pMax < A[i]) // Comparação envolvendo os elementos
      *pMax = A[i];
    if(*pMin > A[i]) // Comparação envolvendo os elementos
      *pMin = A[i];
  }
}
```

- Como vimos anteriormente, $f(n) = 2(n-1)$ para $n > 0$, para o melhor caso, pior caso e caso médio.

Exemplo 05: Ordem de complexidade do MaxMin1

```
int MaxMin1(int* A, int n, int* pMax, int* pMin) {
int i;
  *pMax = A[0];
  *pMin = A[0];
  for(i = 1; i < n; i++) {
    if(*pMax < A[i]) // Comparação envolvendo os elementos
      *pMax = A[i];
    if(*pMin > A[i]) // Comparação envolvendo os elementos
      *pMin = A[i];
  }
}
```

- Como vimos anteriormente, $f(n) = 2(n-1)$ para $n > 0$, para o melhor caso, pior caso e caso médio.
- Então, **MaxMin1** é $O(n)$.

Exemplo 06: Operações com a notação O

- Regra da soma $O(f(n)) + O(g(n))$.
- Suponha três trechos cujos tempos de execução são $O(n)$, $O(n^2)$ e $O(n \ log \ n)$.
- O tempo de execução dos dois primeiros trechos é $O(max(n, n^2))$, que é $O(n^2)$.
- O tempo de execução de todos os três trechos é então $O(max(n, n^2, n \ log \ n))$, que é $O(n^2)$.

Exemplo 06: Operações com a notação O

- Regra da soma $O(f(n)) + O(g(n))$.
- Suponha três trechos cujos tempos de execução são $O(n)$, $O(n^2)$ e $O(n \; log \; n)$.
- O tempo de execução dos dois primeiros trechos é $O(max(n, n^2))$, que é $O(n^2)$.
- O tempo de execução de todos os três trechos é então $O(max(n, n^2, n \; log \; n))$, que é $O(n^2)$.

Exemplo 06: Operações com a notação O

- Regra da soma $O(f(n)) + O(g(n))$.
- Suponha três trechos cujos tempos de execução são $O(n)$, $O(n^2)$ e $O(n \; log \; n)$.
- O tempo de execução dos dois primeiros trechos é $O(max(n, n^2))$, que é $O(n^2)$.
- O tempo de execução de todos os três trechos é então $O(max(n, n^2, n \; log \; n))$, que é $O(n^2)$.

Exemplo 06: Operações com a notação O

- Regra da soma $O(f(n)) + O(g(n))$.
- Suponha três trechos cujos tempos de execução são $O(n)$, $O(n^2)$ e $O(n \log n)$.
- O tempo de execução dos dois primeiros trechos é $O(max(n, n^2))$, que é $O(n^2)$.
- O tempo de execução de todos os três trechos é então $O(max(n, n^2, n \log n))$, que é $O(n^2)$.

Notação Ω (Ômega)

- Especifica um **limite inferior** para $g(n)$.
- **Definição**: Uma função $g(n)$ é $\Omega(f(n))$ se:
  - Existem duas constantes positivas c e m tais que, para $n >= m$, temos $|g(n)| >= c\ |f(n)|$.
- Exemplo:
  - Quando dizemos que o tempo de execução $T(n)$ de um programa é $\Omega(n^2)$, significa que existem constantes c e m tais que, para valores de $n >= m$, $T(n) >= c\ n^2$.

Notação Ω (Ômega)

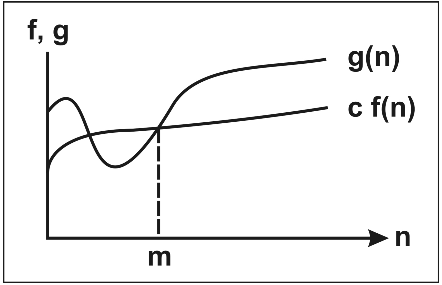
- Especifica um **limite inferior** para $g(n)$.
- **Definição**: Uma função $g(n)$ é $\Omega(f(n))$ se:
  - Existem duas constantes positivas c e m tais que, para $n >= m$, temos $|g(n)| >= c\ |f(n)|$.
- Exemplo:
  - Quando dizemos que o tempo de execução $T(n)$ de um programa é $\Omega(n^2)$, significa que existem constantes c e m tais que, para valores de $n >= m$, $T(n) >= c\ n^2$.

Notação Ω (Ômega)

- Especifica um **limite inferior** para $g(n)$.
- **Definição**: Uma função $g(n)$ é $\Omega(f(n))$ se:
  - Existem duas constantes positivas c e m tais que, para $n >= m$, temos $|g(n)| >= c\ |f(n)|$.
- Exemplo:
  - Quando dizemos que o tempo de execução $T(n)$ de um programa é $\Omega(n^2)$, significa que existem constantes c e m tais que, para valores de $n >= m$, $T(n) >= c\ n^2$.

Exemplo gráfico

- Na figura abaixo, a função $f(n)$ é **dominada assintoticamente** pela função $g(n)$.



Exemplos

- Para mostrar que $g(n) = 3n^3 + 2n^2$ é $\Omega(n^3)$ basta fazer $c = 1$, e então $3n^3 + 2n^2 >= n^3$ para $n >= 0$.
- Seja $g(n) = n$, para n ímpar $(n >= 1)$ e $g(n) = n^2$ para n par $(n >= 0)$. Neste caso $g(n)$ é $\Omega(n^2)$, bastando considerar $c = 1$ e $m = 2, 4, 6, \ldots$.

Notação Θ (Theta)

- Especifica um **limite assintótico firme** para $g(n)$.
- **Definição**: Uma função $g(n)$ é $\Theta(f(n))$ se:
  - Existem três constantes positivas c_1, c_2 e m, tais que, para $n >= m$, temos:
$$0 <= c_1 f(n) <= g(n) <= c_2 f(n).$$
- Isto é, para todo $n >= m$, a função $g(n)$ é igual a $f(n)$ a menos de uma constante.

Notação Θ (Theta)

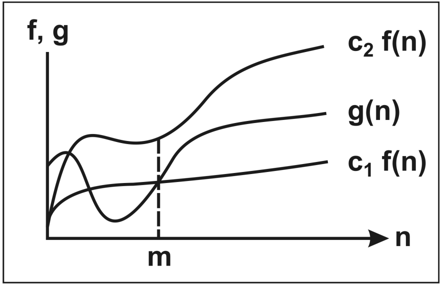
- Especifica um **limite assintótico firme** para $g(n)$.
- **Definição**: Uma função $g(n)$ é $\Theta(f(n))$ se:
  - Existem três constantes positivas c_1, c_2 e m, tais que, para $n >= m$, temos:
  $$0 <= c_1 f(n) <= g(n) <= c_2 f(n).$$
- Isto é, para todo $n >= m$, a função $g(n)$ é igual a $f(n)$ a menos de uma constante.

Notação Θ (Theta)

- Especifica um **limite assintótico firme** para $g(n)$.
- **Definição**: Uma função $g(n)$ é $\Theta(f(n))$ se:
  - Existem três constantes positivas c_1, c_2 e m, tais que, para $n >= m$, temos: $$0 <= c_1 f(n) <= g(n) <= c_2 f(n).$$
- Isto é, para todo $n >= m$, a função $g(n)$ é igual a $f(n)$ a menos de uma constante.

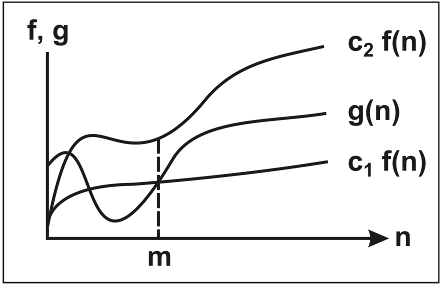
Exemplo gráfico

- Na figura abaixo, a função $f(n)$ é um **limite assintótico firme** para a função $g(n)$.



Relação com O e Ω

- Para uma função ser $\boldsymbol{\Theta(f(n))}$ ela deverá ser, ao mesmo tempo, $\boldsymbol{O(f(n))}$ e $\boldsymbol{\Omega(f(n))}$.



Exemplo: Algoritmos MinMax

- Relembre as funções de complexidade:

Algoritmo	Melhor caso	Pior caso	Caso médio
MaxMin1	2 (n-1)	2 (n-1)	2 (n-1)
MaxMin2	n - 1	2 (n-1)	3 n/2 - 3/2
MaxMin3	3 n/2 - 2	3 n/2 - 1	3 n/2 - 2

- Observe que todos os algoritmos tem a mesma complexidade assintótica.

Exemplo: Algoritmos MinMax

- Relembre as funções de complexidade:

Algoritmo	Melhor caso	Pior caso	Caso médio
MaxMin1	2 (n-1)	2 (n-1)	2 (n-1)
MaxMin2	n - 1	2 (n-1)	3 n/2 - 3/2
MaxMin3	3 n/2 - 2	3 n/2 - 1	3 n/2 - 2

- Observe que todos os algoritmos tem a mesma complexidade assintótica.

Exemplo: Algoritmos MinMax

- Relembre as funções de complexidade:

Algoritmo	Melhor caso	Pior caso	Caso médio
MaxMin1	2 (n-1)	2 (n-1)	2 (n-1)
MaxMin2	n - 1	2 (n-1)	3 n/2 - 3/2
MaxMin3	3 n/2 - 2	3 n/2 - 1	3 n/2 - 2

- Observe que todos os algoritmos tem a mesma complexidade assintótica.
- Todos são $\boldsymbol{O(n)}$ e $\boldsymbol{\Omega(n)}$.

Exemplo: Algoritmos MinMax

- Relembre as funções de complexidade:

Algoritmo	Melhor caso	Pior caso	Caso médio
MaxMin1	2 (n-1)	2 (n-1)	2 (n-1)
MaxMin2	n - 1	2 (n-1)	3 n/2 - 3/2
MaxMin3	3 n/2 - 2	3 n/2 - 1	3 n/2 - 2

- Observe que todos os algoritmos tem a mesma complexidade assintótica.
- Todos são $\boldsymbol{O(n)}$ e $\boldsymbol{\Omega(n)}$. **Portanto, são $\boldsymbol{\Theta(n)}$**.

Transitiva

- $f(n) = O(g(n))$ e $g(n) = O(h(n)) \implies f(n) = O(h(n))$
- $f(n) = \Omega(g(n))$ e $g(n) = \Omega(h(n)) \implies f(n) = \Omega(h(n))$
- $f(n) = \Theta(g(n))$ e $g(n) = \Theta(h(n)) \implies f(n) = \Theta(h(n))$

Reflexiva

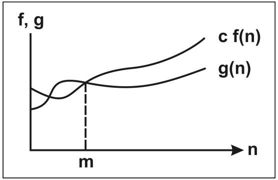
- $f(n) = O(f(n))$
- $f(n) = \Omega(f(n))$
- $f(n) = \Theta(f(n))$

Simetria

- $f(n) = O(g(n)) \Leftrightarrow g(n) = \Omega(f(n))$
- $f(n) = \Theta(g(n)) \Leftrightarrow g(n) = \Theta(f(n))$

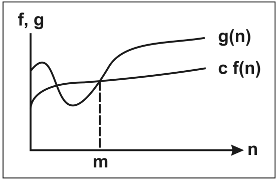
Notações O, Ω (Ômega) e Θ (Theta)

Notação O



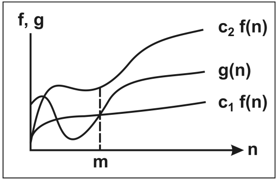
Uma função $g(n)$ é $O(f(n))$ se: Existem duas constantes positivas c e m tais que, para $n >= m$, temos $|g(n)| <= c\ |f(n)|$.

Notação Ω (Ômega)



Uma função $g(n)$ é $\Omega(f(n))$ se: Existem duas constantes positivas c e m tais que, para $n >= m$, temos $|g(n)| >= c\ |f(n)|$.

Notação Θ (Theta)



Uma função $g(n)$ é $\Theta(f(n))$ se: Existem três constantes positivas c_1, c_2 e m, tais que, para $n >= m$, temos:
$0 <= c_1 f(n) <= g(n) <= c_2 f(n)$.

Conteúdo

Conclusão

- Nesta aula aprendemos a estudar o **comportamento assintótico** das funções de custo através da **dominação assintótica**.
- Foco principal para as notações $\boldsymbol{O}$, $\boldsymbol{\Omega}$ e $\boldsymbol{\Theta}$.

Próxima Aula

- *A*nálise de Algoritmos (Parte III) – Classes de Problemas.

Conteúdo

Exercício 01

- Obtenha a função de complexidade $f(n)$ dos algoritmos abaixo.
- Considere apenas as operações envolvendo as variáveis x e y.
- Para cada algoritmo, responda:
  - O algoritmo é $O(n^2)$? É $\Omega(n^3)$? É $\Theta(n^3)$.

```c
void Procedimento1(int n) {
  int i, j, x, y;
  x = y = 0;
  for(i = 1; i <= n; i++) {
    for(j = i; j <= n; j++)
      x = x + 1;
    for(j = 1; j < i; j++)
      y = y + 1;
    }
}
```

```c
void Procedimento2() {
  int i, j, k, x;
  x = 0;
  for(i = 1; i <= n; i++) {
    for(j = 1; j <= n; j++)
      for(k = 1; k <= j; k++)
        x = x + j + k;
    x = i;
}
```