

BCC202 - Estruturas de Dados I

Aula 07: Análise de Algoritmos (Parte III)

Pedro Silva

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



Conteúdo

Revido conceitos

Duas novas notações

Hierarquia de funções

Comparação de Programas

Considerações

Exercícios

Conteúdo

Revido conceitos

Duas novas notações

Hierarquia de funções

Comparação de Programas

Considerações

Exercícios

Função de Complexidade

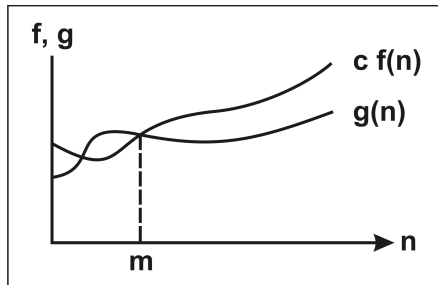
- ▶ Melhor caso x Caso médio x Pior caso.
- ▶ Qual a influência desta função em algoritmos aplicados sobre problemas de tamanho pequeno?
- ▶ E sobre problemas grandes?
- ▶ Estuda-se o comportamento assintótico das funções de custo. O que isso significa?

Comportamento Assintótico de funções

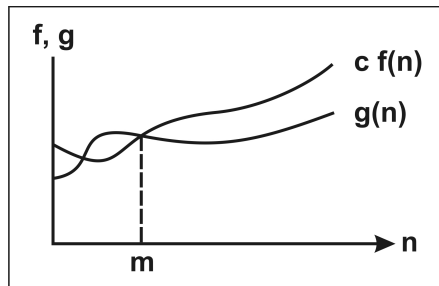
- ▶ Dominação Assintótica.
- ▶ Notação O .
- ▶ Notação Ω .
- ▶ Notação Θ

Dominação Assintótica

- ▶ $f(n)$ **domina assintoticamente** $g(n)$ se:
 - ▶ Existem duas constantes positivas c e m tais que, para $n \geq m$, temos $|g(n)| \leq c|f(n)|$.



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 **$g(n) \leq cf(n)$** , para todo **$n \geq m$** .

Notação O - Operações

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

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

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

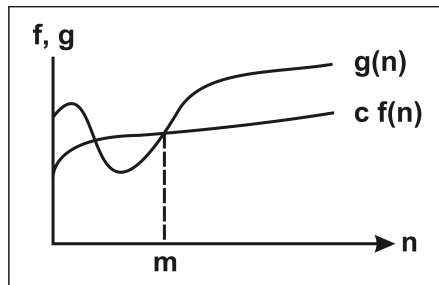
$$O(O(f(n))) = O(f(n))$$

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

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

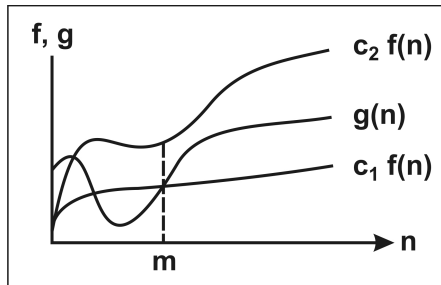
$$f(n) * O(g(n)) = O(f(n) * g(n))$$

Notação Ω



- ▶ 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 $n \geq m$, temos $|g(n)| \geq c |f(n)|$.

Notação Θ

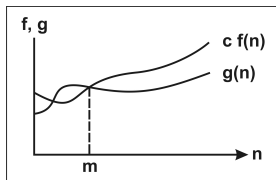
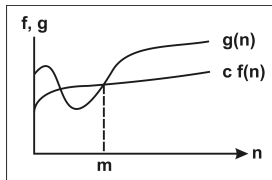
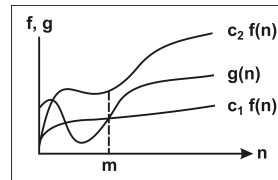


- ▶ Especifica um **limite assintótico firme** para $g(n)$.
- ▶ **Definição:** Uma função $g(n)$ é $\Theta(f(n))$ se:
 - ▶ Existem três constantes positivas c_1 , c_2 e m , tais que, para $n \geq m$, temos:

$$0 \leq c_1 f(n) \leq g(n) \leq c_2 f(n)$$

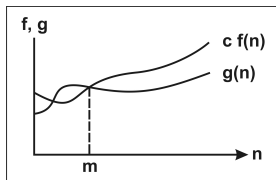
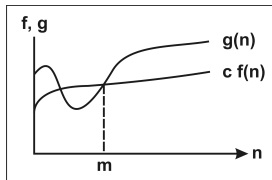
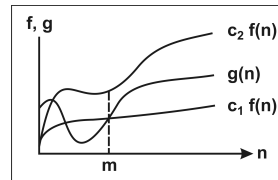
Para uma função ser $\Theta(f(n))$ ela deverá ser, ao mesmo tempo, $O(f(n))$ e $\Omega(f(n))$.

Algumas perguntas importantes

Notação O Notação Ω (Ômega)Notação Θ (Theta)

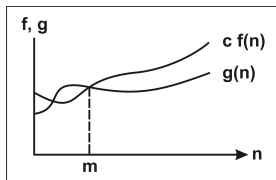
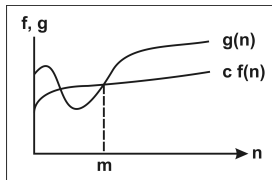
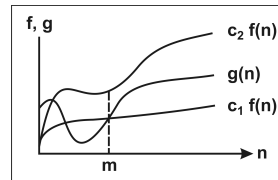
- ▶ n é $O(n \log n)$? Esta afirmação é útil?
- ▶ n^3 é $\Omega(n)$? Esta afirmação é útil?
- ▶ n é $O(n)$?
- ▶ n é $\Omega(n)$?
- ▶ n é $\Theta(n)$?
- ▶ n é $\Theta(n \log n)$?

Algumas perguntas importantes

Notação O Notação Ω (Ômega)Notação Θ (Theta)

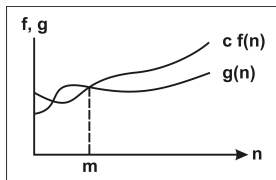
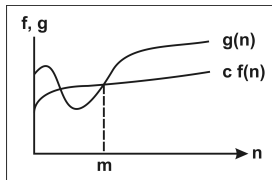
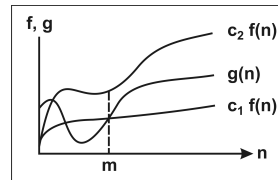
- ▶ n é $O(n \log n)$? Esta afirmação é útil?
- ▶ n^3 é $\Omega(n)$? Esta afirmação é útil?
- ▶ n é $O(n)$?
- ▶ n é $\Omega(n)$?
- ▶ n é $\Theta(n)$?
- ▶ n é $\Theta(n \log n)$?

Algumas perguntas importantes

Notação O Notação Ω (Ômega)Notação Θ (Theta)

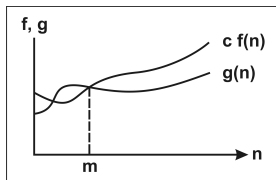
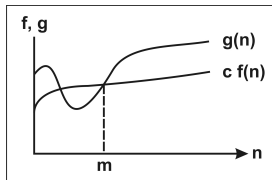
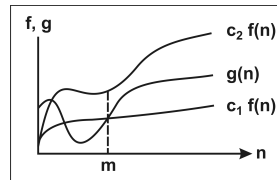
- ▶ n é $O(n \log n)$? Esta afirmação é útil?
- ▶ n^3 é $\Omega(n)$? Esta afirmação é útil?
- ▶ n é $O(n)$?
- ▶ n é $\Omega(n)$?
- ▶ n é $\Theta(n)$?
- ▶ n é $\Theta(n \log n)$?

Algumas perguntas importantes

Notação O Notação Ω (Ômega)Notação Θ (Theta)

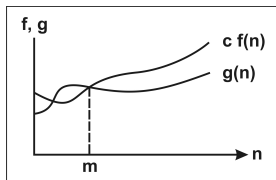
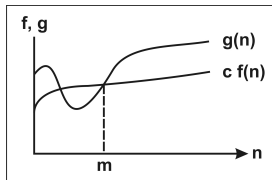
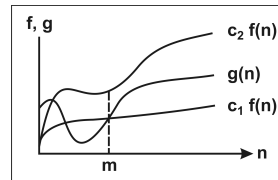
- ▶ n é $O(n \log n)$? Esta afirmação é útil?
- ▶ n^3 é $\Omega(n)$? Esta afirmação é útil?
- ▶ n é $O(n)$?
- ▶ n é $\Omega(n)$?
- ▶ n é $\Theta(n)$?
- ▶ n é $\Theta(n \log n)$?

Algumas perguntas importantes

Notação O Notação Ω (Ômega)Notação Θ (Theta)

- ▶ n é $O(n \log n)$? Esta afirmação é útil?
- ▶ n^3 é $\Omega(n)$? Esta afirmação é útil?
- ▶ n é $O(n)$?
- ▶ n é $\Omega(n)$?
- ▶ n é $\Theta(n)$?
- ▶ n é $\Theta(n \log n)$?

Algumas perguntas importantes

Notação O Notação Ω (Ômega)Notação Θ (Theta)

- ▶ n é $O(n \log n)$? Esta afirmação é útil?
- ▶ n^3 é $\Omega(n)$? Esta afirmação é útil?
- ▶ n é $O(n)$?
- ▶ n é $\Omega(n)$?
- ▶ n é $\Theta(n)$?
- ▶ n é $\Theta(n \log n)$?

Conteúdo

Revedo conceitos

Duas novas notações

Hierarquia de funções

Comparação de Programas

Considerações

Exercícios

Mais sobre notação assintótica

- ▶ Existem duas outras notações na análise assintótica de funções:
 - ▶ Notação \mathbf{o} ("o" pequeno).
 - ▶ Notação ω (ômega pequeno).
- ▶ Estas duas notações não são usadas normalmente, mas é importante saber seus conceitos e diferenças em relação às notações \mathbf{O} e $\mathbf{\Omega}$, respectivamente.

Notação o ("o" pequeno)

- ▶ O limite assintótico superior definido pela notação O pode ser **assintoticamente firme** ou **não**:
 - ▶ Por exemplo, o limite $2n^2 = O(n^2)$ é assintoticamente firme, mas o limite $2n = O(n^2)$ não é.
- ▶ A notação o é utilizada para definir um limite **superior** que **não é assintoticamente firme**.
- ▶ Matematicamente, $g(n)$ é $o(f(n))$ se:
 - ▶ Existem duas constantes positivas c e m tais que, para $n \geq m$, temos $|g(n)| < c|f(n)|$.

Notação ω (ômega pequeno)

- ▶ De forma análoga, a notação ω está relacionada com a notação Ω .
- ▶ A notação ω é utilizada para definir um limite **inferior** que **não é assintoticamente firme**.
- ▶ Matematicamente, $g(n)$ é $\omega(f(n))$ se:
 - ▶ Existem duas constantes positivas c e m tais que, para $n \geq m$, temos $|g(n)| > c |f(n)|$.

Conteúdo

Revedo conceitos

Duas novas notações

Hierarquia de funções

Comparação de Programas

Considerações

Exercícios

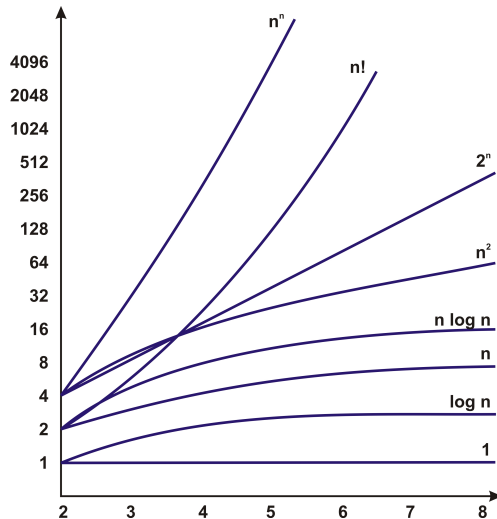
Hierarquia de funções

A seguinte hierarquia de funções pode ser definida do ponto de vista **assintótico**:

$$1 \prec \log \log n \prec \log n \prec n^\epsilon \prec n^c \prec n^{\log n} \prec c^n \prec n^n \prec c^{c^n}$$

onde ϵ e c são constantes em que $0 < \epsilon < 1 < c$.

Hierarquia de funções



Conteúdo

Revedo conceitos

Duas novas notações

Hierarquia de funções

Comparação de Programas

Considerações

Exercícios

Comparação de Programas

- ▶ Um programa com tempo de execução $O(n)$ é melhor que outro com tempo $O(n^2)$.
- ▶ Porém, as constantes de proporcionalidade podem alterar esta consideração.
- ▶ Exemplo: um programa leva $100n$ unidades de tempo para ser executado e outro leva $2n^2$. Qual dos dois programas é melhor?
 - ▶ Depende do tamanho do problema (valor de n).
 - ▶ Para $n = 40$, o problema $O(n^2)$ é melhor do que o $O(n)$.
 - ▶ E para $n = 100$? E para $n = 150$?

Comparação de Programas

- ▶ Um programa com tempo de execução $O(n)$ é melhor que outro com tempo $O(n^2)$.
- ▶ Porém, as constantes de proporcionalidade podem alterar esta consideração.
- ▶ Exemplo: um programa leva $100n$ unidades de tempo para ser executado e outro leva $2n^2$. Qual dos dois programas é melhor?
 - ▶ Depende do tamanho do problema (valor de n).
 - ▶ Para $n = 40$, o problema $O(n^2)$ é melhor do que o $O(n)$.
 - ▶ E para $n = 100$? E para $n = 150$?

Comparação de Programas

- ▶ Um programa com tempo de execução $O(n)$ é melhor que outro com tempo $O(n^2)$.
- ▶ Porém, as constantes de proporcionalidade podem alterar esta consideração.
- ▶ Exemplo: um programa leva $100n$ unidades de tempo para ser executado e outro leva $2n^2$. Qual dos dois programas é melhor?
 - ▶ Depende do tamanho do problema (valor de n).
 - ▶ Para $n = 40$, o problema $O(n^2)$ é melhor do que o $O(n)$.
 - ▶ E para $n = 100$? E para $n = 150$?

Comparação de Programas

- ▶ Um programa com tempo de execução $O(n)$ é melhor que outro com tempo $O(n^2)$.
- ▶ Porém, as constantes de proporcionalidade podem alterar esta consideração.
- ▶ Exemplo: um programa leva $100n$ unidades de tempo para ser executado e outro leva $2n^2$. Qual dos dois programas é melhor?
 - ▶ Depende do tamanho do problema (valor de n).
 - ▶ Para $n = 40$, o problema $O(n^2)$ é melhor do que o $O(n)$.
 - ▶ E para $n = 100$? E para $n = 150$?

Principais Classes de Problemas – Resumo

Ordem de Complexidade	Nome
$O(1)$	Constante
$O(\log n)$	Logarítmica
$O(n)$	Linear
$O(n \log n)$	Linearitmica
$O(n^2)$	Quadrática
$O(n^3)$	Cúbica
$O(2^n)$	Exponencial
$O(n!)$	Exponencial

Principais Classes de Problemas – $f(n) = O(1)$

- ▶ Algoritmos de complexidade **$O(1)$** são ditos de **complexidade constante**.
- ▶ Uso do algoritmo independe de **n** .
- ▶ As instruções do algoritmo são executadas um número fixo de vezes.

Principais Classes de Problemas – $f(n) = O(\log n)$

- ▶ Um algoritmo de complexidade $O(\log n)$ é dito ter **complexidade logarítmica**.
- ▶ Típico em algoritmos que transformam um problema em outros menores.
- ▶ Pode-se considerar o tempo de execução como menor que uma constante grande.
- ▶ Quando n é 1.000, $\log_2 n = 10$, quando n é 1.000.000, $\log_2 n = 20$.
- ▶ Para dobrar $\log_2 n$ temos que elevar n ao quadrado!
- ▶ A base do logaritmo muda pouco estes valores: quando n é 1.000.000, o $\log_2 n = 20$ e $\log_{10} n = 6$.

Principais Classes de Problemas – $f(n) = O(n)$

- ▶ Um algoritmo de complexidade **$O(n)$** é dito ter **complexidade linear**.
- ▶ Em geral, um pequeno trabalho é realizado sobre cada elemento de entrada.
- ▶ É a melhor situação possível para um algoritmo que tem de processar/produzir **n** elementos de entrada/saída.
- ▶ Cada vez que **n** dobra de tamanho, o tempo de execução dobra.

Principais Classes de Problemas – $f(n) = O(n \log n)$

- ▶ Um algoritmo de complexidade $O(n \log n)$ é dito ter **complexidade linearitmica**.
- ▶ Típico em algoritmos que quebram um problema em outros menores, resolvem cada um deles independentemente e juntam as soluções depois.
- ▶ Quando n é 1.000.000, $O(n \log_2 n) \approx 20.000.000$.
- ▶ Quando n é 2.000.000, $O(n \log_2 n) \approx 42.000.000$.
- ▶ Ou seja, quando n dobra, $O(n \log n)$ é pouco mais do que o dobro.

Principais Classes de Problemas – $f(n) = O(n^2)$

- ▶ Um algoritmo de complexidade $O(n^2)$ é dito ter **complexidade quadrática**.
- ▶ Ocorrem quando os itens de dados são processados aos pares, muitas vezes em um laço dentro de outro.
- ▶ Quando n é 1.000, o número de operações é da ordem de 1.000.000.
- ▶ Quando n dobra, o tempo de execução é multiplicado por 4.
- ▶ Úteis para resolver problemas de tamanhos relativamente pequenos.

Principais Classes de Problemas – $f(n) = O(n^3)$

- ▶ Um algoritmo de complexidade $O(n^3)$ é dito ter **complexidade cúbica**.
- ▶ Úteis apenas para resolver pequenos problemas.
- ▶ Quando n é 100, o número de operações é da ordem de 1.000.000.
- ▶ Sempre que n dobra, o tempo de execução fica multiplicado por 8.

Principais Classes de Problemas – $f(n) = O(2^n)$

- ▶ Um algoritmo de complexidade $O(2^n)$ é dito ter **complexidade exponencial**.
- ▶ Geralmente não são úteis sob o ponto de vista prático.
- ▶ Ocorrem na solução de problemas quando se usa **força bruta**.
- ▶ Quando n é 20, o tempo de execução é cerca de 1.000.000.
- ▶ Quando n dobra, o tempo fica elevado ao quadrado.

Principais Classes de Problemas – $f(n) = O(n!)$

- ▶ Um algoritmo de complexidade $O(n!)$ é dito ter **complexidade exponencial**, apesar de apresentar comportamento muito pior do que $O(2^n)$.
- ▶ Geralmente ocorrem quando se usa força bruta para resolver o problema.
- ▶ Para $n = 20$, temos $20! = 2.432.902.008.176.640.000$, um número de 19 dígitos.
- ▶ Para $n = 40$, temos um número de 48 dígitos.

Principais Classes de Problemas – Resumo

Ordem de Complexidade	Nome
$O(1)$	Constante
$O(\log n)$	Logarítmica
$O(n)$	Linear
$O(n \log n)$	Linearitmica
$O(n^2)$	Quadrática
$O(n^3)$	Cúbica
$O(2^n)$	Exponencial
$O(n!)$	Exponencial

Principais Classes de Problemas – Comparativo de tempo

Qual é o efeito no tempo de processamento ao aumentar n ?

Função de Custo	Tamanho n					
	10	20	30	40	50	60
n	0,00001s	0,00002s	0,00003s	0,00004s	0,00005s	0,00006s
n^2	0,0001s	0,0004s	0,0009s	0,0016s	0,0035s	0,0036s
n^3	0,001s	0,008s	0,027s	0,64s	0,125s	0,316s
n^5	0,1s	3,2s	24,3s	1,7min	5,2min	13min
2^n	0,001s	1s	17,9min	12,7dias	35,7anos	366séc.
3^n	0,059s	58min	6,5anos	3855séc.	10^8 séc.	10^{13} séc.

Principais Classes de Problemas – Comparativo de tamanho

Qual é o efeito no tamanho t do problema com o aumento do poder de processamento?

Função de custo de tempo	Computador atual	Computador 100 vezes mais rápido	Computador 1.000 vezes mais rápido
n	t_1	$100t_1$	$1.000t_1$
n^2	t_2	$10t_2$	$31,6t_2$
n^3	t_3	$4,6t_3$	$10t_3$
2^n	t_4	$t_4 + 6,6$	$t_4 + 10$

O tamanho do problema corresponde ao valor máximo de n suportado pela máquina.

Algoritmo Polinomial

- ▶ **Algoritmo exponencial** no tempo de execução tem função de complexidade $O(c^n)$, onde $c > 1$.
- ▶ **Algoritmo polinomial** no tempo de execução tem função de complexidade $O(p(n))$, onde $p(n)$ é um **polinômio**.
- ▶ A distinção entre estes dois tipos de algoritmos torna-se significativa quando o tamanho do problema a ser resolvido cresce.
- ▶ Por isso, os algoritmos polinomiais são muito mais úteis na prática do que os exponenciais.

Conteúdo

Revedo conceitos

Duas novas notações

Hierarquia de funções

Comparação de Programas

Considerações

Exercícios

Conclusão

- ▶ Nesta aula aprofundamos ainda mais os estudos de análise de complexidade de algoritmos apresentando as **Classes de Problemas**.
- ▶ Limite assintótico **firme** e **não firme**.

Recursividade.

Conteúdo

Revedo conceitos

Duas novas notações

Hierarquia de funções

Comparação de Programas

Considerações

Exercícios

Exercício 01

- ▶ Indique se as afirmativas a seguir são verdadeiras ou falsas e justifique a sua resposta:
 - ▶ É melhor um algoritmo que requer 2^n passos do que um que requer $10n^{10}$ passos.
 - ▶ $2^{n+1} = O(2^n)$.
 - ▶ $f(n) = O(u(n))$ e $g(n) = O(v(n)) \Rightarrow f(n) + g(n) = O(u(n) + v(n))$
 - ▶ $f(n) = O(u(n))$ e $g(n) = O(v(n)) \Rightarrow f(n) - g(n) = O(u(n) - v(n))$