Classes de Comportamento Assintótico

Projeto e Análise de Algoritmos

Daniel Capanema

Pontifícia Universidade Católica de Minas Gerais

Classes de Comportamento Assintótico

- Se f é uma função de complexidade para um algoritmo F, então O(f) é considerada a complexidade assintótica ou o comportamento assintótico do algoritmo F.
- A relação de dominação assintótica permite comparar funções de complexidade.
- Entretanto, se as funções f e g dominam assintoticamente uma a outra, então os algoritmos associados são equivalentes.
- Nestes casos, o comportamento assintótico não serve para comparar os algoritmos.
- Por exemplo, considere dois algoritmos F e G aplicados à mesma classe de problemas, sendo que F leva três vezes o tempo de G ao serem executados, isto é, $f(n) = 3g(n)$, sendo que $O(f(n)) = O(g(n))$.
- Logo, o comportamento assintótico não serve para comparar os algoritmos F e G, porque eles diferem apenas por uma constante.

Comparação de Programas

- Podemos avaliar programas comparando as funções de complexidade, negligenciando as constantes de proporcionalidade
- Um programa com tempo de execução O(n) é melhor que outro com tempo O(n2)
  - Porém, as constantes de proporcionalidade podem alterar esta consideração
- Exemplo: um programa leva 100n unidades de tempo para ser executado e outro leva 2n2. Qual dos dois programas é melhor?
  - Depende do tamanho do problema
  - Para n < 50, o programa com tempo 2n2 é melhor do que o que possui tempo 100n
  - Para problemas com entrada de dados pequena é preferível usar o programa cujo tempo de execução é O(n2)
  - Entretanto, quando n cresce, o programa com tempo de execução O(n2) leva muito mais tempo que o programa O(n)

Classes de Comportamento Assintótico

- Complexidade constante $\leftarrow f(n) = \mathrm{O}(1)$
  - O uso do algoritmo independe do tamanho de n
  - As instruções do algoritmo são executadas um número fixo de vezes
  - O que significa um algoritmo ser O(2) ou O(5)?

Classes de Comportamento Assintótico

- Complexidade Logarítmica $\leftarrow f(n) = \mathrm{O}(\log n)$
  - Ocorre tipicamente em algoritmos que resolvem um problema transformando-o em problemas menores
  - Nestes casos, o tempo de execução pode ser considerado como sendo menor do que uma constante grande

- Supondo que a base do logaritmo seja 2:
  - Para n = 1 000, $\log_2 \approx 10$
  - Para n = 1 000 000, $\log_2 \approx 20$

- Exemplo:
  - Algoritmo de pesquisa binária

Classes de Comportamento Assintótico

- Complexidade Linear $\leftarrow f(n) = \mathrm{O}(n)$
  - Em geral, um pequeno trabalho é realizado sobre cada elemento de entrada
  - Esta é a melhor situação possível para um algoritmo que tem que processar/produzir n elementos de entrada/saída
  - Cada vez que n dobra de tamanho, o tempo de execução também dobra

- Exemplo:
  - Algoritmo de pesquisa sequencial

Classes de Comportamento Assintótico

- Complexidade Linear Logarítmica $\leftarrow f(n) = \mathrm{O}(n \log n)$
  - Este tempo de execução ocorre tipicamente em algoritmos que resolvem um problema quebrando-o em problemas menores, resolvendo cada um deles independentemente e depois agrupando as soluções
  - Caso típico dos algoritmos baseados no paradigma divisão-e-conquista

- Supondo que a base do logaritmo seja 2:
  - Para n = 1 000 000, $\log_2 \approx 20\,000\,000$
  - Para n = 2 000 000, $\log_2 \approx 42\,000\,000$

- Exemplo:
  - Algoritmo de ordenação MergeSort

Classes de Comportamento Assintótico

- Complexidade Quadrática $\leftarrow f(n) = \mathrm{O}(n^2)$
  - Algoritmos desta ordem de complexidade ocorrem quando os itens de dados são processados aos pares, muitas vezes em um anel dentro do outro
  - Para n = 1000, o número de operações é da ordem de 1000000
  - Sempre que n dobra o tempo de execução é multiplicado por 4
  - Algoritmos deste tipo são úteis para resolver problemas de tamanhos relativamente pequenos

- Exemplos:
  - Algoritmos de ordenação simples como seleção e inserção

Classes de Comportamento Assintótico

- Complexidade Cúbica $\leftarrow f(n) = \mathrm{O}(n^3)$
  - Algoritmos desta ordem de complexidade geralmente são úteis apenas para resolver problemas relativamente pequenos
  - Para n = 100, o número de operações é da ordem de 1000000
  - Sempre que n dobra o tempo de execução é multiplicado por 8

- Exemplo:
  - Algoritmo para multiplicação de matrizes

Classes de Comportamento Assintótico

- Complexidade Exponencial $\leftarrow f(n) = \mathrm{O}(2^n)$
  - Algoritmos desta ordem de complexidade não são úteis sob o ponto de vista prático
  - Eles ocorrem na solução de problemas quando se usa a força bruta para resolvê-los
  - Para n = 20, o tempo de execução é cerca de 1000000
  - Sempre que n dobra o tempo de execução fica elevado ao quadrado

- Exemplo:
  - Algoritmo do Caixeiro Viajante

Classes de Comportamento Assintótico

- Complexidade Exponencial $\leftarrow f(n) = O(n!)$
  - Um algoritmo de complexidade O(n!) é dito ter complexidade exponencial, apesar de O(n!) ter comportamento muito pior do que O(2n)
  - Geralmente ocorrem quando se usa força bruta na solução do problema

- Considerando:
  - n = 20, temos que 20! = 2432902008176640000, um número com 19 dígitos
  - n = 40 temos um número com 48 dígitos

Comparação de funções de complexidade

Função de custo	Tamanho n					
	10	20	30	40	50	60
n	0,00001 s	0,00002 s	0,00003 s	0,00004 s	0,00005 s	0,00006 s
n^2	0,0001 s	0,0004 s	0,0009 s	0,0016 s	0,0.35 s	0,0036 s
n^3	0,001 s	0,008 s	0,027 s	0,64 s	0,125 s	0.316 s
n^5	0,1 s	3,2 s	24,3 s	1,7 min	5,2 min	13 min
2^n	0,001 s	1 s	17,9 min	12,7 dias	35,7 anos	366 séc.
3^n	0,059 s	58 min	6,5 anos	3855 séc.	10^8 séc.	10^{13} séc.

Função de custo de tempo	Computador atual	Computador 100 vezes mais rápido	Computador 1000 vezes mais rápido
n	t_1	$100\ t_1$	$1000\ t_1$
n^2	t_2	$10\ t_2$	$31,6\ t_2$
n^3	t_3	$4,6\ t_3$	$10\ t_3$
2^n	t_4	$t_4 + 6,6$	$t_4 + 10$

Comparação de funções de complexidade