

Complexidade de Algoritmos

Paulino Ng

2020-03-27

Plano da aula

Esta aula apresenta a aproximação das funções que representam o tempo de execução com o crescimento do tamanho da entrada.

1. Comparação de tempos de execução
2. Comportamento assintótico de curvas
3. Notação \mathcal{O} , Ω , Θ , o , ω

Comparação de Tempos de Execução

Para comparar tempos de execução, vamos resolver o problema 1-1 do [CLRS]:

Para cada função $f(n)$ e o tempo t na tabela abaixo, determine o maior tamanho n de um problema que pode ser resolvido em tempo t . Assume-se que o algoritmo para resolver o problema leva $f(n)$ microsegundos.

	1 segundo	1 minuto	1 hora	1 dia	1 mes	1 ano	1 século
$\lg n$							
\sqrt{n}							
n							
$n \lg n$							
n^2							
n^3							
2^n							
$n!$							

Solução para $t = 1s = 10^6 \mu s = 1.000.000 \mu s$

- ▶ $\lg n = 10^6 \rightarrow n = 2^{10^6} \approx 10^{301030}$
- ▶ $\sqrt{n} = 10^6 \rightarrow n = 10^{12}$
- ▶ $n = 10^6$
- ▶ $n^2 = 10^6 \rightarrow n = 10^3$
- ▶ $n^3 = 10^6 \rightarrow n = 10^2$
- ▶ $2^n = 10^6 \rightarrow n = 6 \cdot \log_2 10 \approx 20$
- ▶ $n! = 10^6 \rightarrow n \approx 9$

Calcule o restante da tabela. Observe como o aumento do tempo não resulta num aumento muito significativo do n para as duas últimas funções.

$1s = 10^6 \mu s$	$1\text{min} = 60 \cdot 10^6 \mu s = 6 \cdot 10^7 \mu s;$
$1h = 3,6 \cdot 10^9 \mu s$	$1\text{dia} = 8,64 \cdot 10^{10} \mu s;$
$1\text{mes} = 2,592 \cdot 10^{12} \mu s$	$1\text{ano} = 3,1104 \cdot 10^{13} \mu s$
$1\text{século} = 3,1104 \cdot 10^{16} \mu s$	

Observações sobre este Exercício

- ▶ No lugar de calcular o tempo para cada uma destas funções para um dado tamanho (n), o exercício pede algo mais prático (e difícil de ser calculado) que é o maior problema que podemos resolver com um algoritmo que tenha a função como tempo de execução.
- ▶ É óbvio que não desejamos usar um computador para resolver problemas que precisem de 1 século de cálculo para ser resolvido. Mas o que interessa neste exercício é entender que mesmo se um computador não quebrar durante a execução de um programa durante um século de execução, o problema resolvido não terá sido muito maior do que o que precisou de uma hora para ser resolvido, se a função for exponencial ou fatorial.
- ▶ Conforme veremos, existem problemas que não podem ser resolvidos muito rapidamente (com uma função de complexidade simples). Estes problemas limitam o melhor algoritmo que pode ser usado para resolvê-los.

Assíntotas

- ▶ [ZIVIANI] afirma que a *escolha do algoritmo* não é um problema crítico para problemas de tamanho pequeno. Logo, a análise de algoritmos é realizada para valores grandes de n .
- ▶ Estuda-se o *comportamento assintótico* das **funções de custo**.
Definição: Uma função $f(n)$ domina assintoticamente outra função $g(n)$ se existem duas constantes positivas c e m tais que, para $n \geq m$, temos $|g(n)| \leq c|f(n)|$.

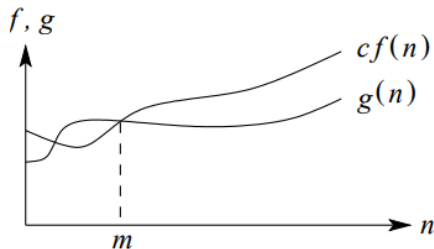


Figure 1: A função $f(n)$ domina assintoticamente a função $g(n)$ [ZIVIANI].

Exemplos de dominação assintótica

- ▶ $g(n) = n$ é assintoticamente dominada por $f(n) = -n^2$ para todo n natural.
- ▶ $g(n) = (n + 1)^2$ e $f(n) = n^2$ dominam assintoticamente uma à outra.
- ▶ As funções dadas na tabela exercício do início da aula foram escolhidas de maneira que as funções das linhas inferiores dominam assintoticamente as funções das linhas superiores.

Notação *big-O*

- ▶ Segundo [ZIVIANI], Knuth sugeriu a notação usada atualmente para indicar que uma função $f(n)$ domina assintoticamente a outra, $g(n)$: $\mathcal{O}(f(n)) = g(n)$
- ▶ A wikipedia diz que a notação é parte da notação de Bachmann–Landau.
- ▶ Uma definição mais formal é:
Seja g uma função complexa, ou real, e f uma função real, ambas definidas num subconjunto não limitado dos números reais positivos, tais que $f(x)$ é estritamente positiva para todos os x suficientemente grandes. Escreve-se:
 $g(x) = \mathcal{O}(f(x))$ *com $x \rightarrow \infty$ se, e apenas se, para todo valor suficientemente grande de x , o valor absoluto de $g(x)$ é no máximo igual a uma constante positiva vezes o $f(x)$.*

Propriedades e Exemplos

[ZIVIANI] Uma função $g(n)$ é $\mathcal{O}(f(n))$ se existem duas constantes positivas c e m tais e $g(n) \leq c f(n)$, $\forall n \geq m$.

- ▶ Exemplo: Seja $g(n) = (n + 1)^2$. Logo, $g(n)$ é $\mathcal{O}(n^2)$, quando $m = 1$ e $c = 4$. Isso porque $(n + 1)^2 \leq 4 n^2$ para $n \geq 1$.
- ▶ Exemplo: A função $g(n) = 3 n^3 + 2 n^2 + n$ é $\mathcal{O}(n^3)$.
- ▶ Exemplo: A função $g(n) = \log_5 n$ é $\mathcal{O}(\log n)$.

Propriedades

$$f(n) = \mathcal{O}(f(n))$$

$$c \times \mathcal{O}(f(n)) = \mathcal{O}(f(n)) \quad c = \text{constante}$$

$$\mathcal{O}(f(n)) + \mathcal{O}(f(n)) = \mathcal{O}(f(n))$$

$$\mathcal{O}(\mathcal{O}(f(n))) = \mathcal{O}(f(n))$$

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

$$\mathcal{O}(f(n)) \mathcal{O}(g(n)) = \mathcal{O}(f(n) g(n))$$

$$f(n) \mathcal{O}(g(n)) = \mathcal{O}(f(n) g(n))$$

Exercícios

1. Suponha que um programa tenha 3 trechos com tempos de execução $\mathcal{O}(n)$, $\mathcal{O}(n^2)$ e $\mathcal{O}(n \log n)$. Qual o tempo de execução do programa como um todo?
2. Indique quais afirmações abaixo são verdadeiras ou falsas:
 - a. $2^{n+1} = \mathcal{O}(2^n)$
 - b. $2^{2n} = \mathcal{O}(2^n)$
 - c. $f(n) = \mathcal{O}(u(n))$ e $g(n) = \mathcal{O}(v(n)) \Rightarrow f(n) + g(n) = \mathcal{O}(u(n) + v(n))$
 - d. $f(n) = \mathcal{O}(u(n))$ e $g(n) = \mathcal{O}(v(n)) \Rightarrow f(n) - g(n) = \mathcal{O}(u(n) - v(n))$
3. [desafio] Prove que $f(n) = 1^2 + 2^2 + \dots + n^2$ é igual a $\frac{n^3}{3} + \mathcal{O}(n^2)$

Outras definições

- ▶ A notação big-O diz que $f(n)$ é um limite superior para a taxa de crescimento da função $g(n)$. Outras definições permitem outras aproximações assintóticas.
- ▶ Definição da notação Ω : Uma função $g(n)$ é $\Omega(f(n))$ se existirem duas constantes c e m tais que $g(n) \geq c f(n)$, para todo $n \geq m$.
 - ▶ $g(n)$ é $\Omega(f(n))$ quer dizer que $f(n)$ é um limite inferior para a taxa de crescimento de $g(n)$.
 - ▶ Exemplo: $g(n) = 3n^3 + 2n^2$ é $\Omega(n^3)$
- ▶ Definição notação Θ : Uma função $g(n)$ é $\Theta(f(n))$ se existirem constantes positivas c_1 , c_2 e m tais que $0 \leq c_1 f(n) \leq g(n) \leq c_2 f(n)$, para todo $n \geq m$.
 - ▶ $c_1 f(n)$ está abaixo de $g(n)$, $c_2 f(n)$ está acima de $g(n)$, dizemos que $f(n)$ é um **limite assintótico firme** de $g(n)$.

Mais definições

- ▶ Definição notação o : Uma função $g(n)$ é $o(f(n))$ se, para qualquer constante $c > 0$, então $0 \leq g(n) < c f(n)$, $\forall n \geq m$.
 - ▶ Exemplo: $2n = o(n^2)$, mas $2n^2 \neq o(n^2)$
 - ▶ A diferença entre \mathcal{O} e o é que na big-O existe uma constante c e na o a relação vale para todo c positivo.
- ▶ Definição da notação ω : Uma função $g(n)$ é $\omega(f(n))$ se, para qualquer constante $c > 0$, então $0 \leq c f(n) \leq g(n)$, $\forall n \geq m$.
 - ▶ Exemplo: $\frac{n^2}{2} = \omega(n)$, mas $\frac{n^2}{2} \neq \omega(n^2)$

Classes de Comportamento

- ▶ Se f é uma **função de complexidade** para um algoritmo F , então $\mathcal{O}(f)$ é considerada a **complexidade assintótica** ou o comportamento assintótico do algoritmo F .
- ▶ Um programa com tempo de execução $\mathcal{O}(n)$ é melhor do que um programa com tempo de execução $\mathcal{O}(n^2)$ para n acima de um certo valor.
- ▶ A maioria dos algoritmos possui um parâmetro que afeta mais significativamente o tempo de execução do que os outros. Em geral, este parâmetro é o número de itens a ser processado.
- ▶ O n que representa o tamanho da entrada pode ser: o número de registros num arquivo ou o número de nós de um grafo.

► As principais **classes de problemas** com suas **funções de complexidade** são:

1. $f(n) = \mathcal{O}(1)$: Algoritmos de **complexidade constante**.
2. $f(n) = \mathcal{O}(\log n)$: Algoritmos de **complexidade logarítmica**.
3. $f(n) = \mathcal{O}(n)$: Algoritmos de **complexidade linear**.
4. $f(n) = \mathcal{O}(n \log n)$: Algoritmos que resolvem um problema particionando-o em problemas menores e resolvendo os problemas menores para compor a solução do problema original.
5. $f(n) = \mathcal{O}(n^2)$: Algoritmos de **complexidade quadrática**.
6. $f(n) = \mathcal{O}(n^3)$: Algoritmos de **complexidade cúbica**.
7. $f(n) = \mathcal{O}(2^n)$: Algoritmos de **complexidade exponencial**.
8. $f(n) = \mathcal{O}(n!)$: Algoritmos de **complexidade fatorial**, ou **complexidade combinatório**.

Técnicas para Análise de Algoritmos

- ▶ Aho, Hopcroft e Ullman enumeram alguns princípios a serem seguidos na análise de algoritmos, não existem técnicas gerais:
 1. O tempo de execução de um comando de atribuição, de leitura ou de escrita pode ser considerado $\mathcal{O}(1)$.
 2. O tempo de execução de uma sequência de comandos é determinado pelo maior tempo de execução de qualquer comando da sequência.
 3. O tempo de execução de um comando de decisão é composto pelo tempo de execução dos comandos executados dentro do comando condicional mais o tempo para calcular a condição, $\mathcal{O}(1)$.
 4. O tempo para executar um laço é a soma do tempo de execução do corpo do laço mais o tempo de calcular a condição para terminar multiplicado pelo número de iterações do laço.

5. Quando o programa possui procedimentos não recursivos, o tempo de execução de cada procedimento deve ser computado separadamente, um a um, iniciando com os procedimentos que não chamam outros procedimentos. A seguir devem ser avaliados os procedimentos que chamam os procedimentos cujos tempos já foram computados. Esse processo é repetido até chegar ao programa principal.
6. Quando o programa possui **procedimentos recursivos**, a cada procedimento é associada uma função de complexidade $f(n)$ desconhecida, na qual n mede o tamanho dos argumentos para o procedimento, conforme veremos depois.

Exercício

1. Este exercício procura mostrar que a complexidade depende do que se considera como tamanho da entrada, n .
 - a. Seja N um número inteiro positivo escrito na base 10, analise a complexidade do algoritmo para encontrar o seu valor binário (base 2) pelas divisões sucessivas por 2. O tamanho do problema inicial é o próprio valor de N .
 - b. Seja o mesmo problema do item a., mas agora o tamanho do problema é dado pelo número de dígitos de N .