Melhores momentos

AULA PASSADA

MT multifita por MT fita única

Duas máquinas são **equivalentes** se elas reconhecem a mesma linguagem.

Teorema. Dada uma máquina de Turing multifita M, podemos construir uma uma máquina de Turing S com uma única fita e equivalente a M. Além disso, se tendo como entrada uma cadeia de comprimento n a máquina M faz não mais do que $t(n) \geq n$ passos, então S faz $\mathrm{O}(t^2(n))$ passos.

Máquinas de Turing não-determinísticas

Para cada estado q e símbolo a, $\delta(q, a)$ é um **conjunto finito de ternos**:

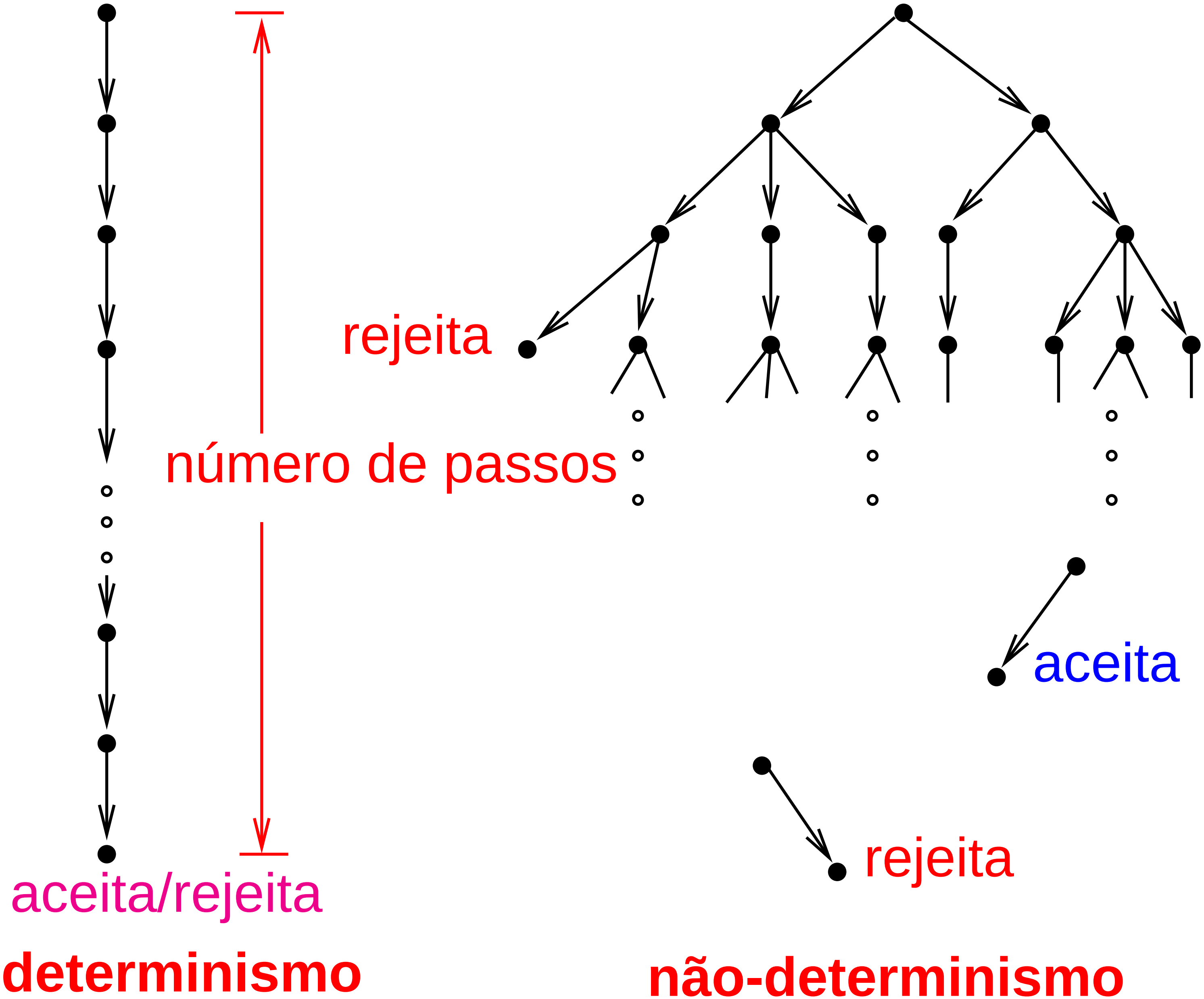
$$(q_1, a_1, L), (q_2, a_2, R), \ldots, (q_b, a_b, R)$$

A computação da MT **se ramifica** dependendo das possibilidades da função δ.

Se algum ramo atinge o estado de aceitação, a máquina aceita a entrada.

Chamamos uma máquina de Turing não-determinística de **decisora** se todos os possíveis ramos de computação param em todas as entradas.

Número de passos



Não-determinístico por determinísmo

Teorema. Dada uma máquina de Turing não-determinística decisora N, podemos construir uma máquina de Turing determinística D equivalente a N. Além disso, se tendo como entrada uma cadeia de comprimento n a máquina N faz não mais do que $t(n) \geq n$ passos, então D faz $2^{\mathrm{O}(t(n))}$ passos.

Conclusões

Corolário. Uma linguagem é Turing-reconhecível se e somente se alguma máquina de Turing não-determinística a reconhece.

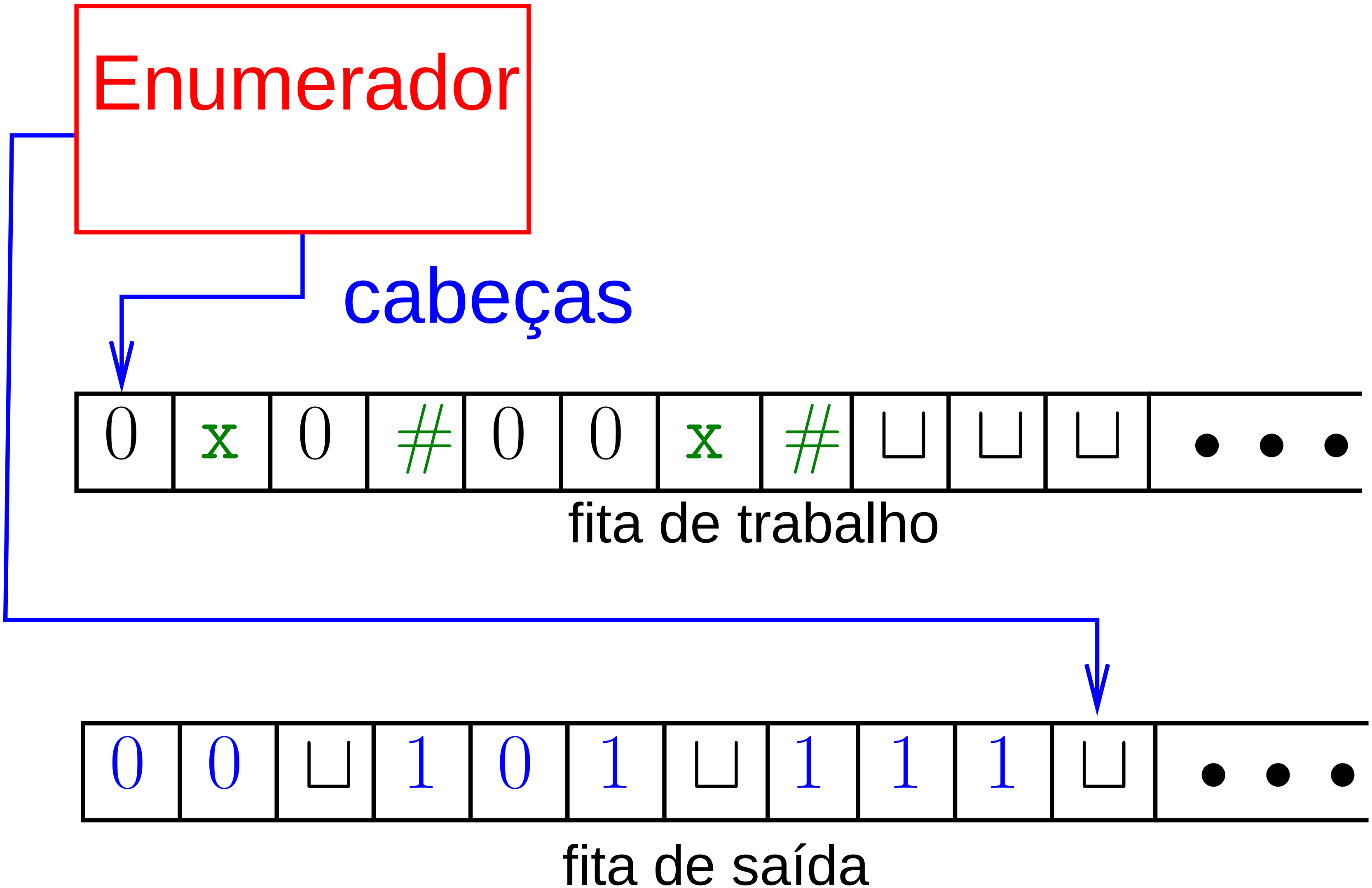
Corolário. Uma linguagem é decidível se e somente se alguma máquina de Turing não-determinística a decide.

AULA 6

Enumeradores

MS 3.2

Enumeradores



Enumeradores

Na fita de saída é somente permitido a escrita e mover para a direita.

A **linguagem enumerada** por um enumerador E é o conjunto das cadeias que E escreve na fita de saída, começando com suas fitas vazias. As cadeias são separadas por $\sqcup$s.

Linguagens enumeráveis

Teorema. Uma linguagem é Turing-reconhecível se e somente se ela é a linguagem enumerada por algum enumerador.

Prova:
($\Leftarrow$) Primeiro mostramos que se a linguagem L é enumerada por um enumerador E, então uma MT M reconhece L.

$M =$ "com entrada w:

1. Simule E. Cada vez que E termina de escrever uma cadeia na fita de saída, a compare com w.
2. Se E escreve alguma vez w na fita de saída, *aceite*."

($\Rightarrow$) Agora suponha que L é reconhecida por M.

Na descrição abaixo $s_1, s_2, s_3, \ldots$ é uma lista das cadeias em Σ^* em ordem lexicográfica (primeiro os mais curtos, etc).

$E =$ "ignore a entrada.

1. Repita os seguintes passos para $i = 1, 2, 3, \ldots$.
2. Simule M por i passos em cada entrada $s_1, s_2, \ldots, s_i$.
3. Se alguma computação aceita, escreva a cadeia correspondente na fita de saída."

■

Tese de Church-Turing

MS 3.2

Definição de Algoritmo

Décimo problema de Hilbert (1900): projetar um algoritmo para decidir se um dado polinômio de coeficientes inteiros tem uma raiz inteira.

Exemplos:

- $10x^2y + 10yz + 5z$ tem raiz inteira $x = 3$, $y = 1$ e $z = -6$
- $x^2 - 5$ não tem raiz inteira.

Como definir *algoritmo*?

Tese de Church-Turing

Noção intuitiva de algoritmo
igual a
máquinas de Turing

Tese de Church-Turing (1936)

A Tese de Church-Turing é uma tese e não um teorema, não é um resultado matemático.

Ela simplesmente afirma que o conceito informal de algoritmo corresponde ao objeto matemático chamado maquina de Turing.

Linguagens indecídiveis

Yuri Matijasevič (1970): **não existe** algoritmo para o décimo problema de Hilbert.

Seja $D = \{\langle p \rangle : p$ é um polinômio com raiz inteira$\}$.

A linguagem D é indecidível. Porém, é Turing-reconhecível.

Seja
$D_1 = \{\langle p \rangle : p$ é um polinômio sobre x com raiz inteira$\}$.

Máquina de Turing que reconhece D_1:

$M_1 =$ "com entrada p:

Calcule o valor de $p(x)$ para $x = 0, 1, -1, 2, -2, \ldots$.
Se em algum ponto o valor der zero, *aceite*."

Para a linguagem D podemos projetar uma máquina de Turing parecida.

D_1 é na verdade decidível: podemos limitar um valor máximo que a raiz pode ter em função dos coeficientes de p.

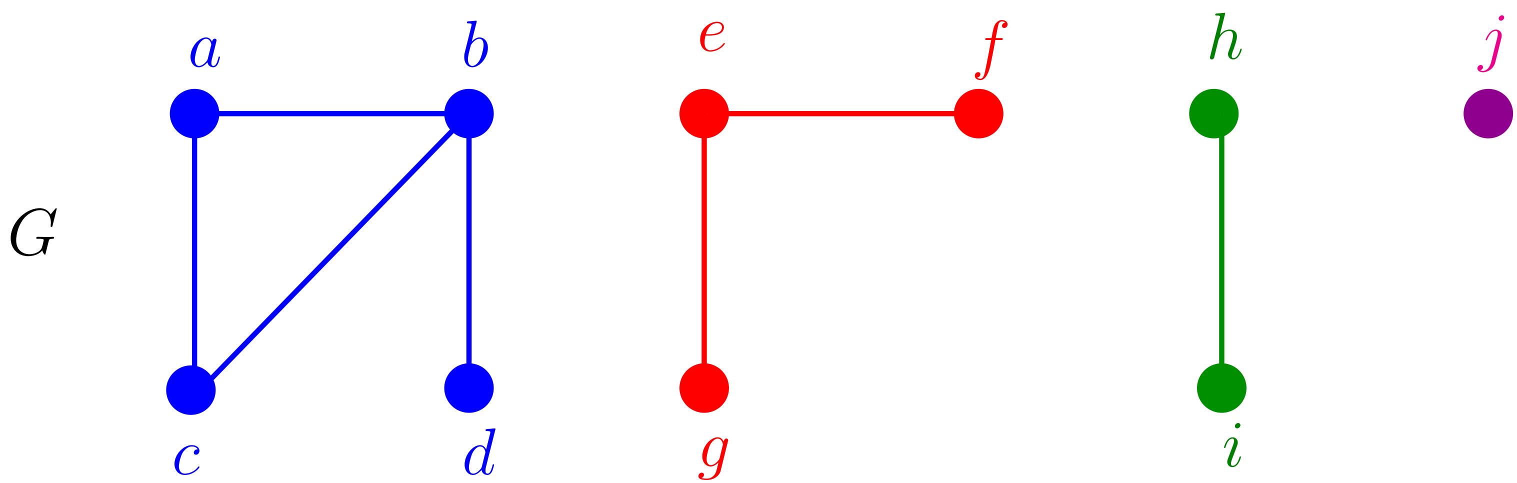
Descrição de máquinas de Turing

Discussão: nível de detalhes para descrever máquinas de Turing; codificação de objetos.

Objeto O é representado como uma cadeia $\langle O \rangle$.

Exemplo de codificação

Grafo



Cadeia:

$$(\{a, b, c, d, e, f, g, h, i, j\}, \{\{bd\}, \{eg\}, \{ac\}, \{hi\}, \{ab\}, \{ef\}, \{bc\}\}$$

Comprimento da cadeia: 59

Grafos conexos

$L = \{\langle G\rangle : G$ é um grafo conexo$\}$.

Máquina M que decide L.

$M =$ "com entrada $\langle G\rangle$, a codificação de G:

1. Selecione o primeiro nó de G e o marque.
2. Repita o seguinte até que nenhum novo nó possa ser marcado:
3. Para cada nó em G, marque o nó se ele é ligado por um aresta a um nó já marcado.
4. Verifique se todos os nós de G estão marcados. Se estão, *aceite*; se não, *rejeite*."

Mais detalhes de implementação

- M deve verificar se $\langle G \rangle$ codifica apropriadamente um grafo. Deve haver duas listas, a primeira com números distintos e a segunda com pares de números da primeira lista. Máquina M_4 pode ajudar para verificar se os números são distintos. Depois de completar essas verificações, M vai para o passo 1.
- Para o passo 1, podemos marcar o primeiro nó com um ponto em seu primeiro símbolo.

Ainda mais detalhes de implementação

- Para o passo 2, M pode percorrer a lista de vértices para encontrar um nó não está “pontuado” u_1 e marcá-lo, digamos, sublinhando seu primeiro símbolo. Agora M percorre a lista de novo para encontrar um nó pontuado u_2 e o sublinha também. Agora a lista de arestas é percorrida para ver se existe uma aresta conectando u_1 e u_2. Se existe, M pontua u_1, remove os sublinhados e volta ao início do passo 2. Se não existe tal aresta, então M move o sublinhado de u_2 para o próximo nó pontuado, e percorre a lista de arestas outra vez. Se não existe mais vértices pontuados então M muda o sublinhado de u_1 para o próximo nó não pontuado e repete, voltando ao início do parágrafo. Se não há mais nós não pontuados, M vai ao passo 4.

Ainda mais detalhes de implementação

- Para o passo 4, M percorre a lista de nós para determinar se todos estão pontuados. Se estão, *aceite*; se não, *rejeite*.

Conexo genérico

Recebe um grafo $G = (N, E)$ e decide se G é ou não conexo.

CONEXO-GENÉRICO (N, E)

para cada v em V **faça**
    $T \leftarrow T \cup \{v\}$
$s \leftarrow$ nó qualquer em V
$S \leftarrow \{s\}$ $\quad T \leftarrow T - \{s\}$

enquanto existe $uv \in E$ com $u \in S$ e $v \in T$ **faça**
    $S \leftarrow S \cup \{v\}$
    $T \leftarrow T - \{v\}$

se $T = \emptyset$
    então o grafo é conexo $\quad \triangleright$ aceite G
    senão o grafo não é conexo $\quad \triangleright$ rejeite G

Complexidade de tempo

MS 7.1

Complexidade de tempo

Seja M uma máquina de Turing **determinística** que pára sobre todas as entradas.

O **tempo de execução** ou **complexidade de tempo** ou **consumo de tempo** de M é a função t de $\mathbb{Z}_{\geq}$ em $\mathbb{Z}_{\geq}$ tal que $t(n)$ é o número máximo de passos de M com uma entrada de comprimento n.

Se $t(n)$ é o tempo de execução de M, dizemos que M **roda em tempo** $t(n)$ e que M é uma máquina de Turing **de tempo** $t(n)$.

Exemplo de MT

Descrição alto nível de um máquina M_1 que decide

$$A = \{0^k 1^k : k \geq 0\}.$$

$M_1 =$ "Com entrada w:

1. Faça um varredura na fita e rejeite se for encontrado algum 0 à direita de um 1.
2. Repita se existem 0s e 1s na fita: corte um único 0 e um único 1.
3. Se sobrarem 0s após todos os 1s terem sido cortados ou existerem 1s após todos os 0s serem cortados, *rejeite*. Caso contrário, *aceite*"

Consumo de tempo

O passo 1 consome tempo $\mathrm{O}(n)$.

O passo 2 consome tempo $\mathrm{O}(n^2)$.

O passo 3 consome tempo $\mathrm{O}(n)$.

A complexidade de tempo de M_1 é $\mathrm{O}(n^2)$ ou ainda M_1 consome tempo $\mathrm{O}(n^2)$.

Classes de complexidade

Seja $t(n)$ uma função de $\mathbb{Z}_{\geq}$ em $\mathbb{Z}_{\geq}$.

A **classe de complexidade de tempo** TIME$(t(n))$ é formada por todas as linguagens que são decidíveis por uma máquina de Turing uma fita que consome tempo $\mathrm{O}(t(n))$:

$$\text{TIME}(t(n)) = \{L : L \text{ é decidida por uma MT que roda em tempo } \mathrm{O}(t(n))\}.$$

Exemplos:

- $A = \{0^k 1^k : k \geq 0\}$ está em TIME(n^2)
- $\{0^{2^k} : k \geq 0\}$ está em TIME$(n \lg n)$ (AULA 2)
- $\{z : z \in \{0,1\}^*, z = z^R\}$ está em TIME(n) (AULA 4)

A está em TIME$(n \lg n)$

$M_2 =$ "Com entrada w:

1. Faça um varredura na fita e rejeite se for encontrado algum 0 à direita de um 1.
2. Repita se existem 0s e 1s na fita:
   (a) Faça uma varredura na fita verificando se o número total de 0s e 1s sobrando é par ou ímpar. Se for ímpar, rejeite.
   (b) Faça um varredura na fita, cortando alternadamente um 0 sim e outro não e cortando um 1 sim e outro não.
3. Se sobrarem 0s após todos os 1s terem sido cortados ou existerem 1s após todos os 0s serem cortados, *rejeite*. Caso contrário, *aceite*."

Consumo de tempo

O passo 1 consome tempo $\mathrm{O}(n)$.

Cada execução do passo 2(a) consome tempo $\mathrm{O}(n)$.

Cada execução do passo 2(b) consome tempo $\mathrm{O}(n)$.

O passo 3 consome tempo $\mathrm{O}(n)$.

Número de repetições de passo 2 é $\mathrm{O}(\lg n)$.

A complexidade de tempo de M_2 é $\mathrm{O}(n \lg n)$ ou ainda M_2 consome tempo $\mathrm{O}(n \lg n)$.

A está em TIME(n)

Descrição alto nível de um máquina M_3 com duas fitas que decide

$$A = \{0^k 1^k : k \geq 0\}.$$

$M_2 =$ "Com entrada w:

1. Faça um varredura na fita 1 e rejeite se for encontrado algum 0 à direita de um 1.
2. Faça um varredura na fita 1 copiando os 0s para a fita 2
3. Faça uma varredura dos 1s na fita 1 e para cada 1 corte um 0 da fita 2. Se todos os 0s forem cortandos antes de todos os 1s serem lidos, *rejeite*.
4. Se todos os 0s da fita 2 foram cortados *aceite*. Caso contrário, *rejeite*"

Consumo de tempo

O passo 1 consome tempo $\mathrm{O}(n)$.

O passo 2 consome tempo $\mathrm{O}(n)$.

O passo 3 consome tempo $\mathrm{O}(n)$.

O passo 4 consome tempo $\mathrm{O}(n)$.

A complexidade de tempo de M_3 é $\mathrm{O}(n)$ ou ainda M_3 consome tempo $\mathrm{O}(n)$.

A classe P

MS 7.2

Classe P

P é classe de linguagens decidíveis em tempo polinomial por uma máquina de Turing determinística de uma fita:

$$\mathsf{P} = \bigcup_k \mathsf{TIME}(n^k).$$

A classe P é importante pois

1. P é invariante para todos os modelos de computação polinomialmente equivalentes a uma MT com uma fita.
2. P contém a classe dos problemas que são resolvidos eficientemente.

Irmãos de O

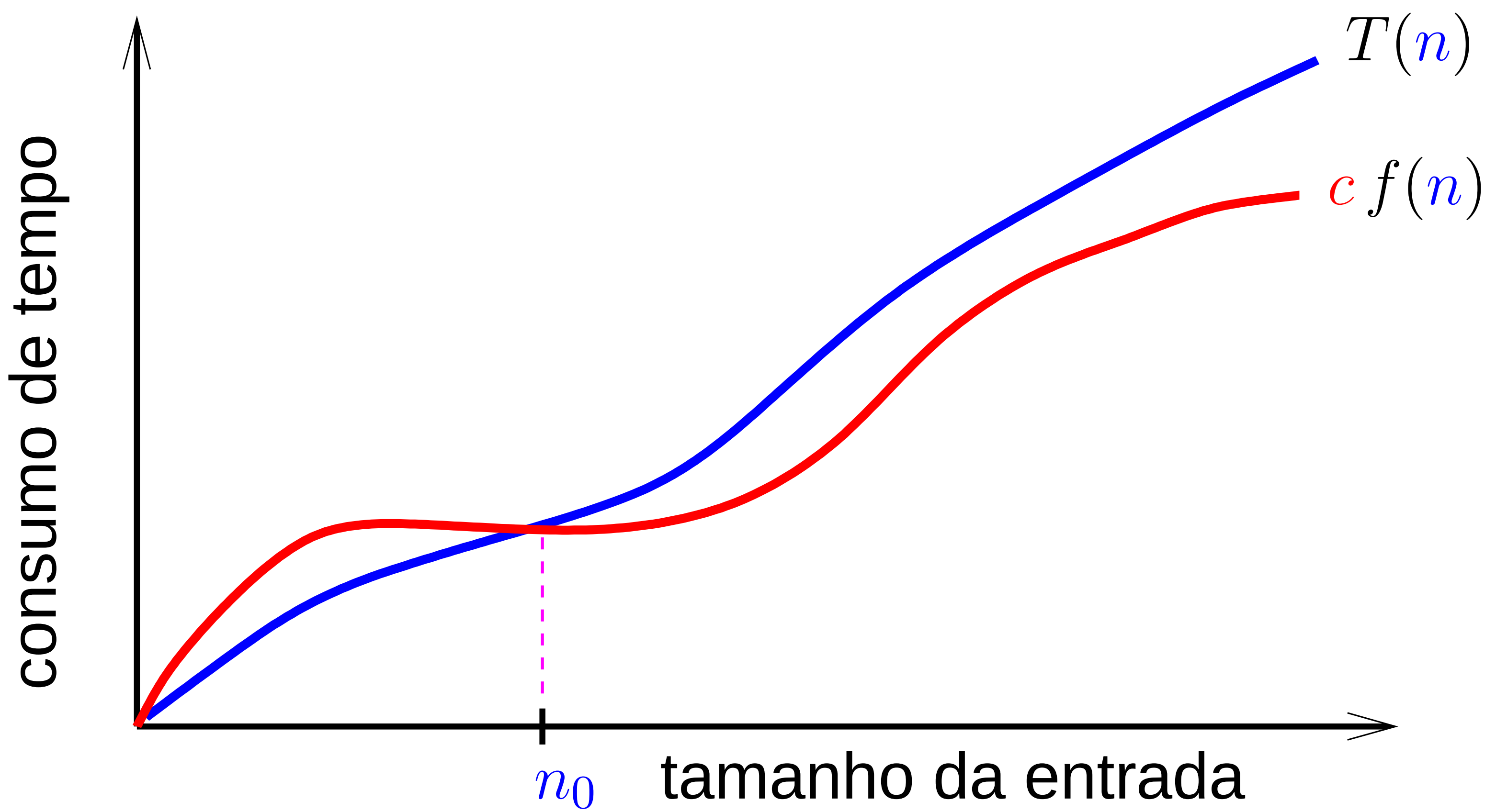
CLRS 3.1

Definição

Dizemos que $T(n)$ **é** $\Omega(f(n))$ se existem constantes positivas c e n_0 tais que

$$c\,f(n) \;\leq\; T(n)$$

para todo $n \geq n_0$.

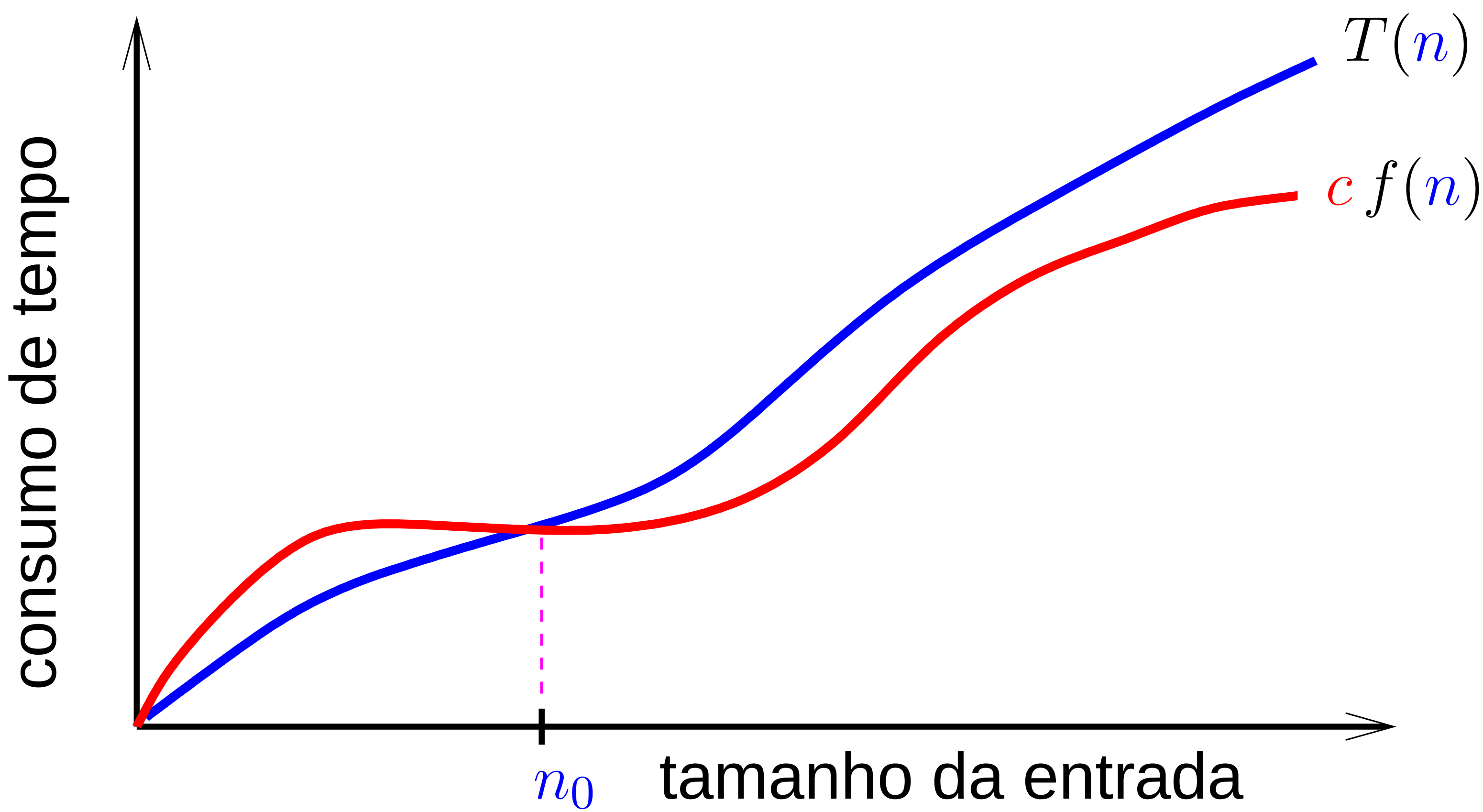


Mais informal

$T(n) = \Omega(f(n))$ se existe $c > 0$ tal que

$$c\, f(n) \leq T(n)$$

para todo n suficientemente GRANDE.



Exemplos

Exemplo 1

Se $T(n) \geq 0.001n^2$ para todo $n \geq 8$, então $T(n)$ é $\Omega(n^2)$.

Prova: Aplique a definição com $c = 0.001$ e $n_0 = 8$.

Exemplo 2

Se $T(n)$ é $\Omega(f(n))$, então $f(n)$ é $\mathrm{O}(T(n))$.

Prova: Se $T(n)$ é $\Omega(f(n))$, então existem constantes positivas c e n_0 tais que

$$c\,f(n) \;\leq\; T(n)$$

para todo $n \geq n_0$. Logo,

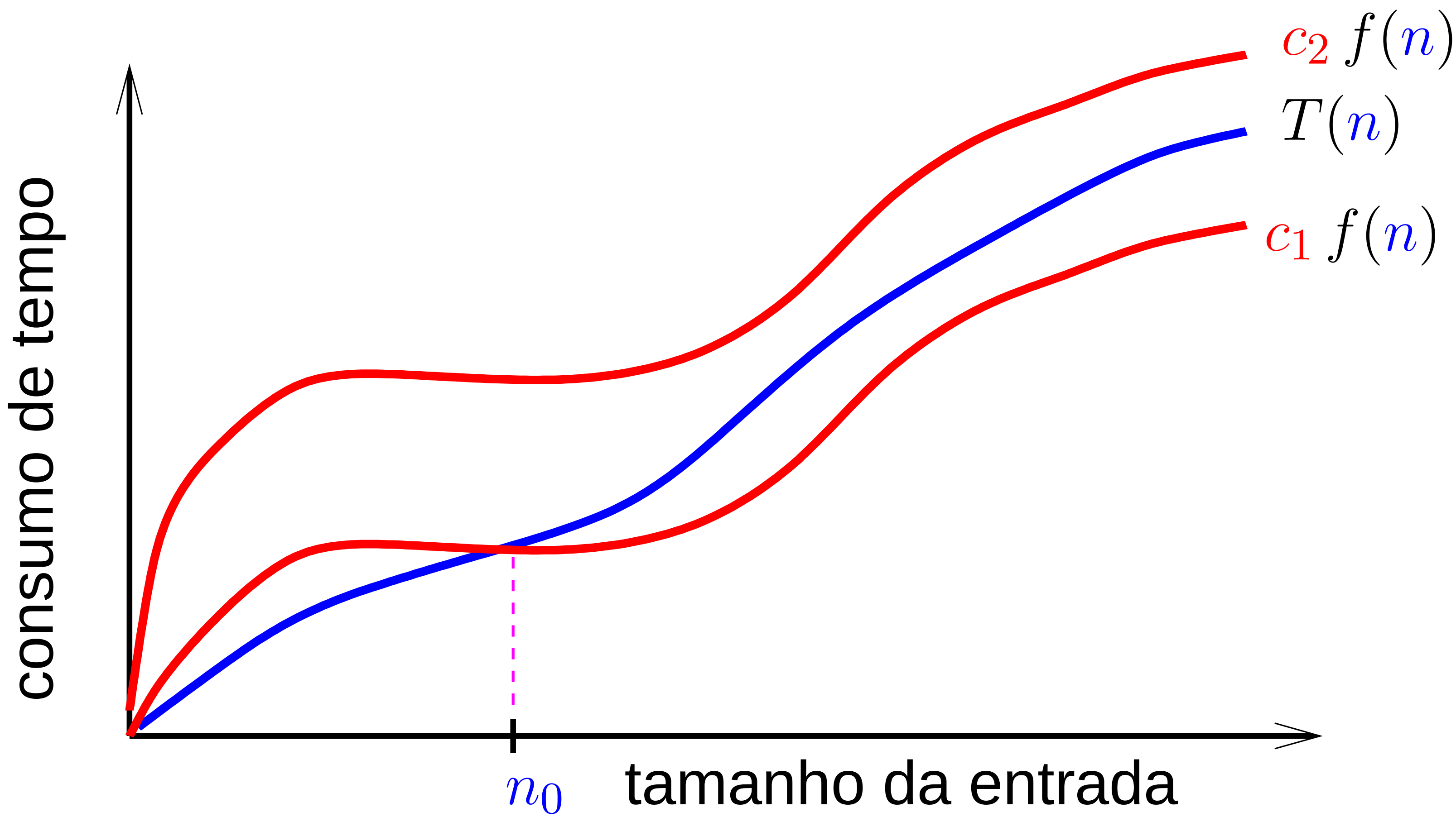
$$f(n) \;\leq\; 1/c\,T(n)$$

para todo $n \geq n_0$. Portanto, $f(n)$ é $\mathrm{O}(T(n))$.

Definição

Sejam $T(n)$ e $f(n)$ funções dos inteiros no reais.
Dizemos que $T(n)$ **é** $\Theta(f(n))$ se

$$T(n) \text{ é } \mathrm{O}(f(n)) \text{ e } T(n) \text{ é } \Omega(f(n)).$$

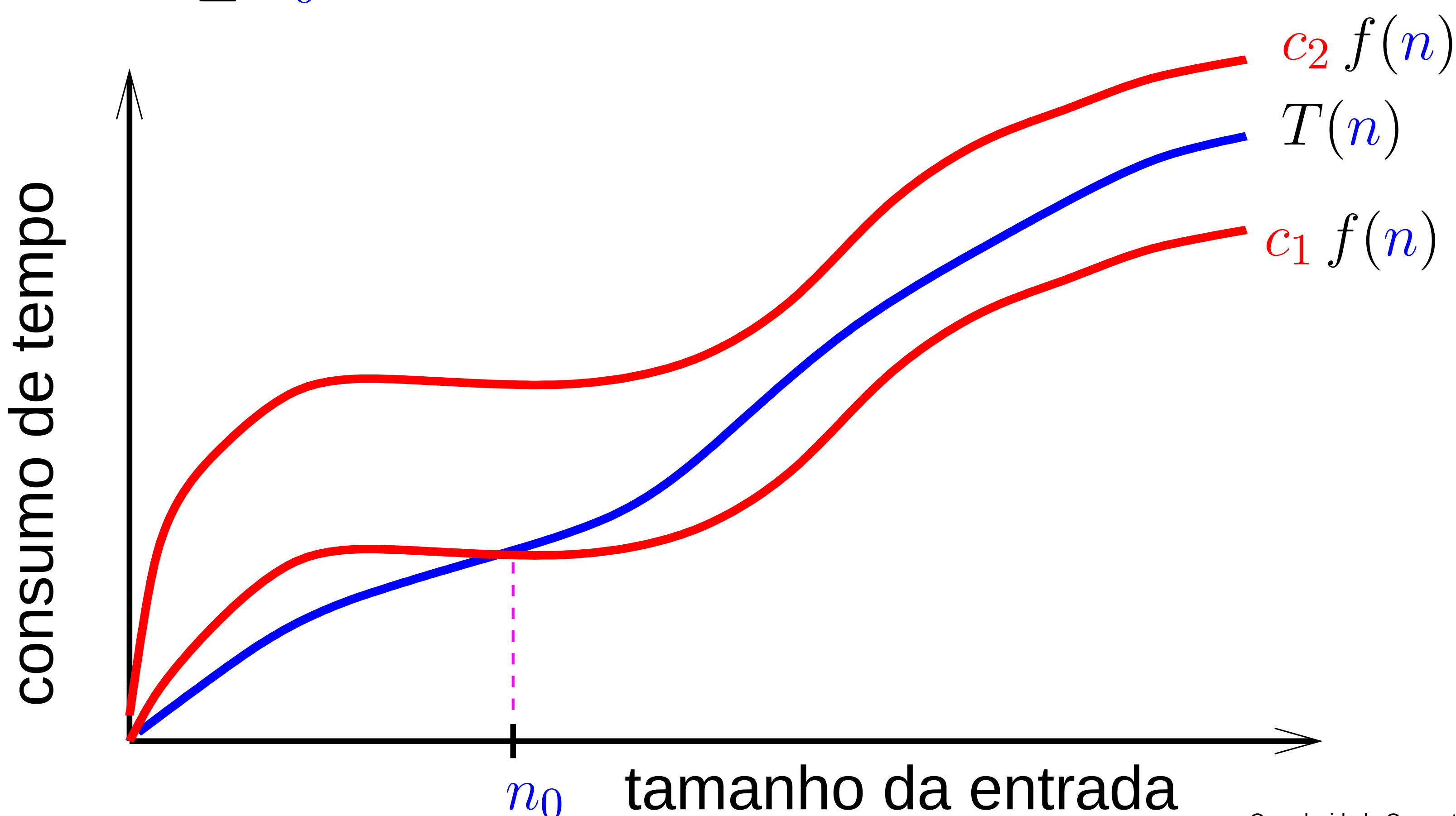


Definição

Dizemos que $T(n)$ **é** $\Theta(f(n))$ se se existem constantes positivas c_1, c_2 e n_0 tais que

$$c_1\, f(n) \;\leq\; T(n) \;\leq\; c_2\, f(n)$$

para todo $n \geq n_0$.



Intuitivamente

Comparação **assintótica**, ou seja, para n ENORME.

comparação	comparação assintótica
$T(n) \leq f(n)$	$T(n)$ é $\mathrm{O}(f(n))$
$T(n) \geq f(n)$	$T(n)$ é $\Omega(f(n))$
$T(n) = f(n)$	$T(n)$ é $\Theta(f(n))$

Tamanho máximo de problemas

Suponha que cada operação consome 1 microsegundo ($1\mu s$).

consumo de tempo(μs)	Tamanho máximo de problemas (n)		
	1 segundo	1 minuto	1 hora
$400n$	2500	150000	9000000
$20\,n\,\lceil\lg n\rceil$	4096	166666	7826087
$2n^2$	707	5477	42426
n^4	31	88	244
2^n	19	25	31

Michael T. Goodrich e Roberto Tamassia, *Projeto de Algoritmos*, Bookman.

Crescimento de algumas funções

n	$\lg n$	$\sqrt{n}$	$n \lg n$	n^2	n^3	2^n
2	1	1,4	2	4	8	4
4	2	2	8	16	64	16
8	3	2,8	24	64	512	256
16	4	4	64	256	4096	65536
32	5	5,7	160	1024	32768	4294967296
64	6	8	384	4096	262144	1,8 10^{19}
128	7	11	896	16384	2097152	3,4 10^{38}
256	8	16	1048	65536	16777216	1,1 10^{77}
512	9	23	4608	262144	134217728	1,3 10^{154}
1024	10	32	10240	1048576	1,1 10^{9}	1,7 10^{308}

Nomes de classes Θ

classe	nome
$\Theta(1)$	constante
$\Theta(\log n)$	logarítmica
$\Theta(n)$	linear
$\Theta(n \log n)$	$n \log n$
$\Theta(n^2)$	quadrática
$\Theta(n^3)$	cúbica
$\Theta(n^k)$ com $k \geq 1$	polinomial
$\Theta(2^n)$	exponencial
$\Theta(a^n)$ com $a > 1$	exponencial

Exercícios

Exercício 6.A
Mostre que o consumo de tempo do algoritmo ORDENA-POR-INSERÇÃO é $\Omega(n)$.

Exercício 6.B
Mostre que o consumo de tempo do algoritmo ORDENA-POR-INSERÇÃO é $\Omega(n^2)$ no pior caso.

Exercício 6.C
Mostre que o consumo de tempo do algoritmo ORDENA-POR-INSERÇÃO é $\mathrm{O}(n)$
no melhor caso.

Mais exercícios

Exercício 6.D

Prove que $n^2 + 10n + 20 = \Omega(n^2)$. Prove que $n^2 - 10n - 20 = \Theta(n^2)$.

Exercício 6.E

Prove que $n = \Omega(\lg n)$.

Exercício 6.F

Prove que $\lg n = \Theta(\log_{10} n)$.

Exercício 6.G

É verdade que $2^n = \Omega(3^n)$?

Exercício 6.H

É verdade que $2n^3 + 5\sqrt{n} = \Theta(n^3)$?

Mais exercícios ainda

Exercício 6.I

Suponha que os algoritmos $\mathcal{A}$ e $\mathcal{B}$ só dependem de um parâmetro n. Suponha ainda que $\mathcal{A}$ consome $S(n)$ unidades de tempo enquanto $\mathcal{B}$ consome $T(n)$ unidades de tempo. Quero provar que algoritmo $\mathcal{A}$ é pelo menos tão eficiente quanto o algoritmo $\mathcal{B}$ (no sentido assintótico). Devo mostrar que existe $f(n)$ tal que

$S(n) = \mathrm{O}(f(n))$ e $T(n) = \mathrm{O}(f(n))$?

$S(n) = \mathrm{O}(f(n))$ e $T(n) = \Omega(f(n))$?

$S(n) = \Omega(f(n))$ e $T(n) = \mathrm{O}(f(n))$?

$S(n) = \Omega(f(n))$ e $T(n)\Omega(f(n))$?

Que devo fazer para mostrar que $\mathcal{A}$ é mais eficiente que $\mathcal{B}$?

Exercício 6.J

Mostre que o consumo de tempo do algoritmo INTERCALA é $\Theta(n)$, sendo n o número de elementos do vetor que o algoritmo recebe.

Primos de O

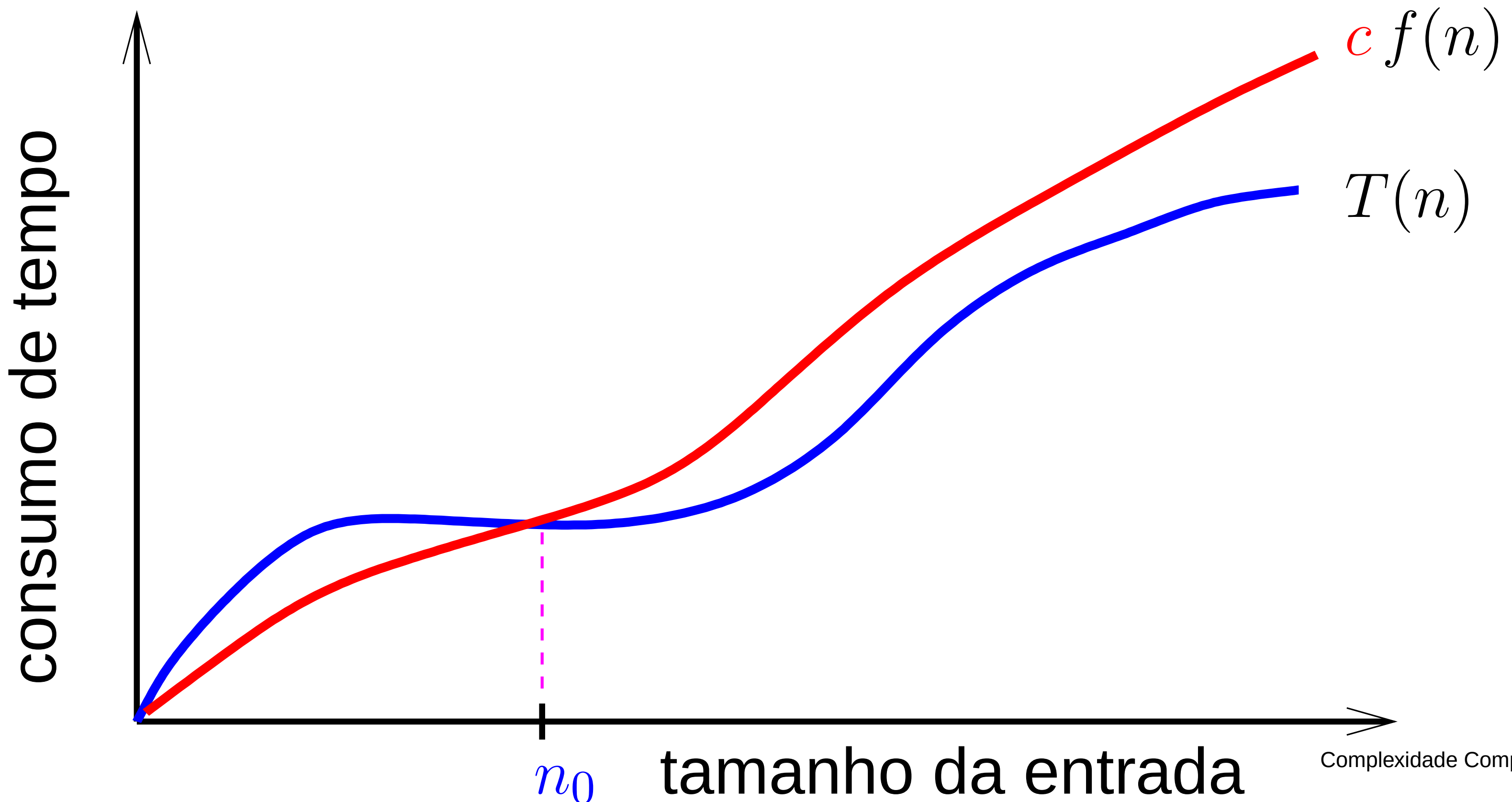
CLRS 3.1

Definição

Sejam $T(n)$ e $f(n)$ funções dos inteiros nos reais. Dizemos que $T(n)$ **é** $\mathrm{o}(f(n))$ se **PARA TODA** constante positiva c **EXISTE** uma constante inteira n_0 tal que

$$T(n) \ < \ c\, f(n)$$

para todo $n \geq n_0$.



Exemplos

$T(n)$ é $\mathrm{o}(f(n))$ lê-se "$T(n)$ é ó pequeno de $f(n)$"

Exemplo 1

$12n^2 + 6n$ é $\mathrm{o}(n^3)$.

Exemplos

$T(n)$ é $\mathrm{o}(f(n))$ lê-se "$T(n)$ é ó pequeno de $f(n)$"

Exemplo 1
$12n^2 + 6n$ é $\mathrm{o}(n^3)$.

Prova: Seja $c > 0$ uma constante.
Tome $n_0 := \lceil (12 + 6 + 1)/c \rceil$.
Temos que

$$12n^2 + 6n < 12n^2 + 6n^2 = (12 + 6)n^2 < c\,n^3$$

para todo $n \geq n_0$.

Exemplos

$T(n)$ é $\mathrm{o}(f(n))$ lê-se "$T(n)$ é ó pequeno de $f(n)$"

Exemplo 2

$12n^2 + 6n$ não é $\mathrm{o}(n^2)$.

Exemplos

$T(n)$ é $\mathrm{o}(f(n))$ lê-se "$T(n)$ é ó pequeno de $f(n)$"

Exemplo 2

$12n^2 + 6n$ não é $\mathrm{o}(n^2)$.

Prova: Para $c := 12$ temos que

$$12n^2 + 6n \geq 12n^2 = c\,n^2$$

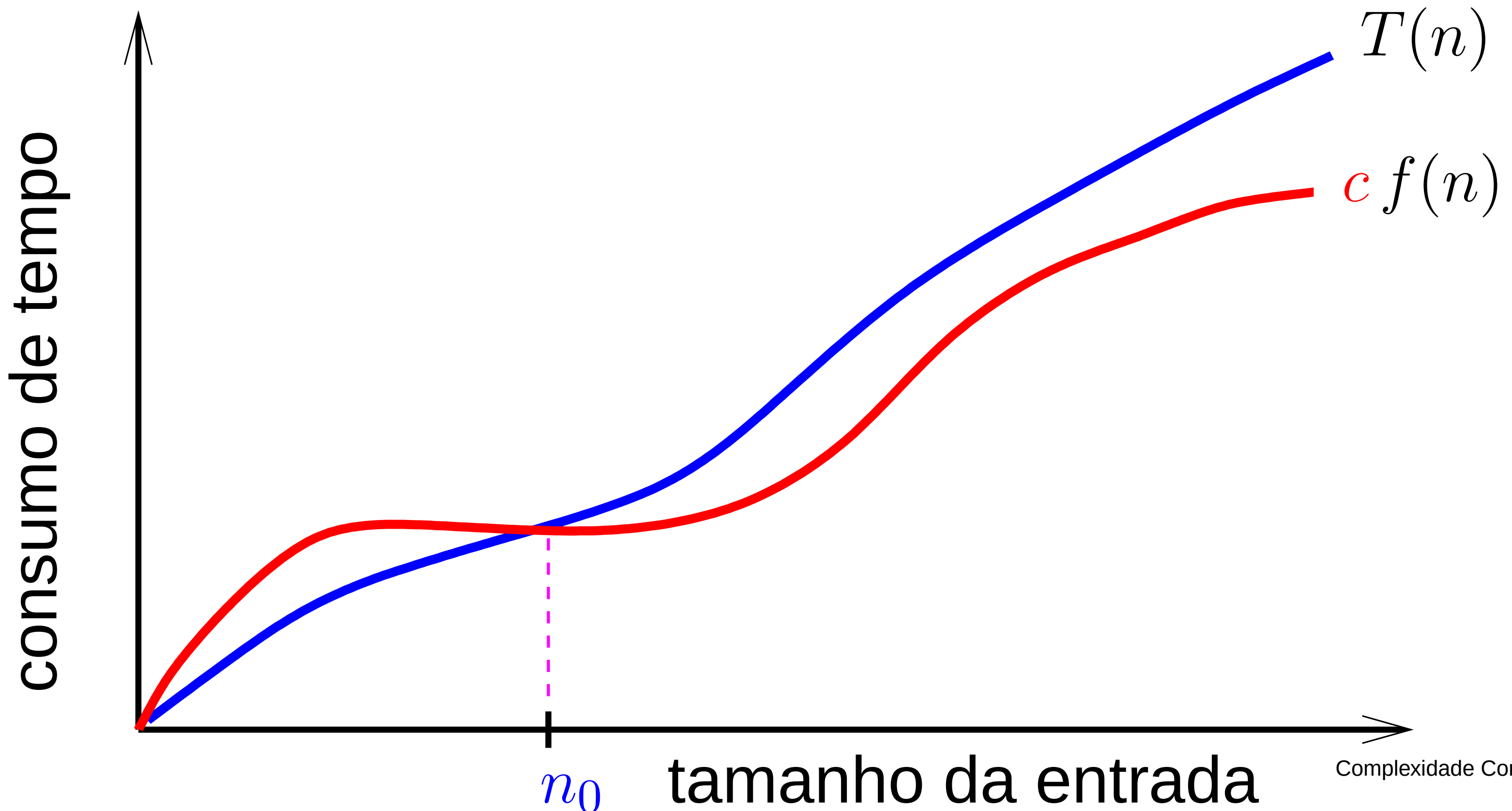
para todo $n \geq 0$.

Definição

Sejam $T(n)$ e $f(n)$ funções dos inteiros no reais.
Dizemos que $T(n)$ **é** $\omega(f(n))$ se para **PARA TODA** constante positiva c **EXISTE** uma constante inteira n_0 tal que

$$T(n) \; > \; c\,f(n)$$

para todo $n \geq n_0$.



Exemplos

$T(n)$ é $\omega(f(n))$ lê-se "$T(n)$ é omega pequeno de $f(n)$"

Exemplo 1

$12n^2 + 6n$ é $\omega(n)$.

Exemplos

$T(n)$ é $\omega(f(n))$ lê-se "$T(n)$ é omega pequeno de $f(n)$"

Exemplo 1

$12n^2 + 6n$ é $\omega(n)$.

Prova: Seja $c > 0$ uma constante.

Tome $n_0 := \lceil c/12 \rceil$.

Temos que

$$12n^2 + 6n > 12n^2 \geq c\, n$$

para todo $n \geq n_0$.

Exemplos

$T(n)$ é $\omega(f(n))$ lê-se "$T(n)$ é omega pequeno de $f(n)$"

Exemplo 2

$12n^2 + 6n$ não é $\omega(n^2)$.

Exemplos

$T(n)$ é $\omega(f(n))$ lê-se "$T(n)$ é omega pequeno de $f(n)$"

Exemplo 2
$12n^2 + 6n$ não é $\omega(n^2)$.

Prova: Para $c := 18$ temos que

$$12n^2 + 6n \leq 18\, n^2 = c\, n^2$$

para todo $n \geq 0$.

Intuitivamente

Comparação **assintótica**, ou seja, para n ENORME.

comparação	comparação assintótica
$T(n) \leq f(n)$	$T(n)$ é $\mathrm{O}(f(n))$
$T(n) \geq f(n)$	$T(n)$ é $\Omega(f(n))$
$T(n) = f(n)$	$T(n)$ é $\Theta(f(n))$
$T(n) < f(n)$	$T(n)$ é $\mathrm{o}(f(n))$
$T(n) > f(n)$	$T(n)$ é $\omega(f(n))$