Licenciatura em Ciências da Computação

COMPUTABILIDADE E COMPLEXIDADE

4. Introdução à Teoria da Complexidade

José Carlos Costa

Dep. Matemática
Universidade do Minho
Braga, Portugal

1º semestre 2021/2022

Definição

Sejam $f, g : \mathbb{N}_0 \to \mathbb{R}$ funções. Diz-se que $g(n)$ *é de ordem* $f(n)$, e escreve-se $g(n) \in \mathcal{O}(f(n))$ ou $g(n)$ é $\mathcal{O}(f(n))$, se existem constantes positivas c e n_0 tais que

$$\forall n \geq n_0, \quad 0 \leq g(n) \leq c\, f(n).$$

Ou seja,

$$\mathcal{O}(f(n)) = \{g(n) : \exists c \in \mathbb{R}^+ \; \exists n_0 \in \mathbb{N}_0 \; \forall n \geq n_0, \; 0 \leq g(n) \leq c\, f(n)\}$$

é a classe das funções que são "limitadas superiormente pela função f(n)".

Exemplo

Sejam $f(n) = n^2 + 5n + 2$ e $g(n) = n^2$.

- $g(n)$ é $\mathcal{O}(f(n))$. De facto,

$$\forall n \geq 0, \quad 0 \leq n^2 \leq n^2 + 5n + 2.$$

Portanto,

$$\forall n \geq 0, \quad 0 \leq g(n) \leq 1\, f(n).$$

Ou seja, para deduzir que $g(n) \in \mathcal{O}(f(n))$ basta tomar $c = 1$ e $n_0 = 0$ na definição de $\mathcal{O}(f(n))$.

- Por outro lado, $f(n)$ é $\mathcal{O}(g(n))$. De facto, para cada $n \geq 1$ tem-se

$$\begin{aligned} 0 \leq f(n) &= n^2 + 5n + 2 \\ &\leq n^2 + 5n^2 + 2n^2 \\ &= 8n^2 \\ &= 8g(n). \end{aligned}$$

Tem-se portanto que $\mathcal{O}(f(n)) = \mathcal{O}(g(n))$. Pode-se assim dizer que $f(n)$ e $g(n)$ são funções da mesma ordem de grandeza.

Poder-se-ia ainda verificar que:

- $n^2 + 5n + 2$ não é $\mathcal{O}(n)$;
- $n^2 + 5n + 2$ é $\mathcal{O}(n^3)$;
- n^3 não é $\mathcal{O}(n^2 + 5n + 2)$.

Mais geralmente, para funções polinomiais é válido o seguinte resultado.

Proposição

Seja $f(n)$ um polinómio de grau k. Então,

1. $f(n)$ não é $\mathcal{O}(n^j)$ para todo o $j < k$.
2. $f(n)$ é $\mathcal{O}(n^k)$ e n^k é $\mathcal{O}(f(n))$.
3. $f(n)$ é $\mathcal{O}(n^\ell)$ e n^ℓ não é $\mathcal{O}(f(n))$ para todo o $\ell > k$.

Consideremos agora também funções não polinomiais.

Exemplo

Sejam $f(n) = n^3$ e $g(n) = 2^n$, e note-se que

n	0	1	2	3	$\cdots$	8	9	10	11	12	$\cdots$
$f(n)$	0	1	8	27	$\cdots$	512	729	1000	1331	1728	$\cdots$
$g(n)$	1	2	4	8	$\cdots$	256	512	1024	2048	4096	$\cdots$

- $f(n)$ é $\mathcal{O}(g(n))$. Basta notar que,

$$\forall n \geq 10, \quad 0 \leq f(n) \leq 1\, g(n).$$

- $g(n)$ não é $\mathcal{O}(f(n))$.

n	10	20	30	40	50	100	200
$\lfloor log_2(n) \rfloor$	3	4	4	5	5	6	7
n	10	20	30	40	50	100	200
n^2	100	400	900	1 600	2 500	10 000	40 000
n^3	1 000	8 000	27 000	64 600	125 000	1 000 000	8 000 000
2^n	1 024	1 048 576	1.0×10^9	1.1×10^{12}	1.1×10^{15}	1.2×10^{30}	1.6×10^{60}
$n!$	3 628 800	2.4×10^{18}	2.6×10^{32}	8.1×10^{47}	3.0×10^{64}	$> 10^{157}$	$> 10^{374}$

Proposição

Sejam $k \in \mathbb{N}_0$ e $a \in \mathbb{R}$ tal que $a > 1$. Então,

1. $log_a(n)$ é $\mathcal{O}(n)$ e n não é $\mathcal{O}(log_a(n))$.
2. n^k é $\mathcal{O}(a^n)$ e a^n não é $\mathcal{O}(n^k)$.
3. a^n é $\mathcal{O}(n!)$ e $n!$ não é $\mathcal{O}(a^n)$.

Definição [Complexidade temporal de uma MT]

Seja $\mathcal{T}$ uma máquina de Turing que pára sempre (ou seja, $\mathcal{T}$ é um algoritmo). A *complexidade temporal* de $\mathcal{T}$ é a função $tc_{\mathcal{T}} : \mathbb{N}_0 \to \mathbb{N}_0$ tal que, para cada $n \in \mathbb{N}_0$,

$$tc_{\mathcal{T}}(n) = \max\{m_u : \begin{array}{l} u \text{ é uma palavra de comprimento } n \text{ e} \\ m_u \text{ é o número de passos que } \mathcal{T} \text{ executa} \\ \text{(até parar) quando é iniciada com } u\}. \end{array}$$

Definição [Complexidade temporal de MT não-determinista]

Seja $\mathcal{T}$ uma MT não-determinista que pára sempre. A *complexidade temporal* de $\mathcal{T}$ é a função $tc_{\mathcal{T}} : \mathbb{N}_0 \to \mathbb{N}_0$ definida, para cada $n \in \mathbb{N}_0$, por

$$tc_{\mathcal{T}}(n) = \max\{m_u : \begin{array}{l} m_u \text{ é o maior número de computações que} \\ \text{podem ser efetuadas por } \mathcal{T} \text{ quando iniciada} \\ \text{com uma palavra } u \text{ de comprimento } n\}. \end{array}$$

Definição

Sejam $f : \mathbb{N}_0 \to \mathbb{R}$ uma função (total) e L uma linguagem. Diz-se que L é *aceite em tempo determinista* (resp. *não-determinista*) f(n) se existe um algoritmo determinista (resp. não-determinista) $\mathcal{T}$ tal que:

- $\mathcal{T}$ aceita L;
- $tc_{\mathcal{T}}(n)$ é $\mathcal{O}(f(n))$.

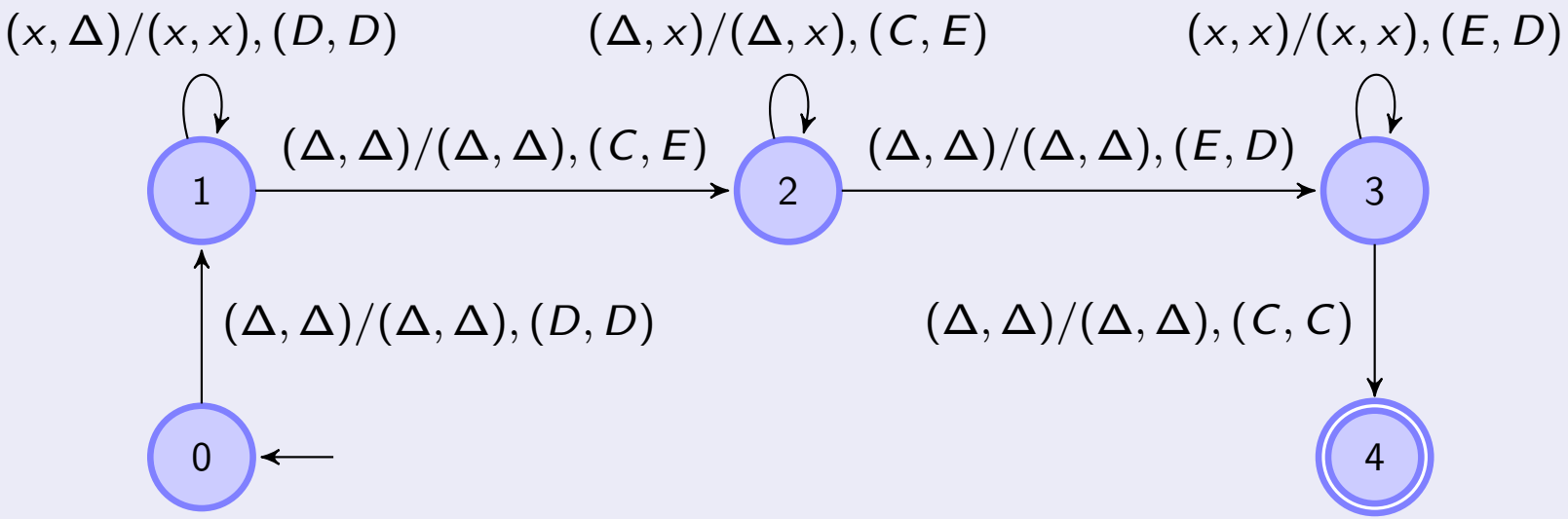
A classe destas linguagens é denotada por $DTIME(f(n))$ (resp. $NTIME(f(n))$). Note-se que $DTIME(f(n)) \subseteq NTIME(f(n))$.

Podemos agora definir duas classes de complexidade importantes:

$$P = \bigcup_{k \geq 0} DTIME(n^k) \quad \text{e} \quad NP = \bigcup_{k \geq 0} NTIME(n^k).$$

Exemplo

A máquina de Turing $\mathcal{T}$ com duas fitas, onde $x \in \{a, b\} = A$,



aceita a linguagem $L = \{u \in A^* : u = u^I\}$ das palavras capicua sobre A. Como se pode verificar, tem-se para todo o $n \in \mathbb{N}_0$

$$tc_{\mathcal{T}}(n) = 3(n+1) + 1 = 3n + 4.$$

Logo $L \in DTIME(3n + 4) = DTIME(n)$. Ou seja, L é aceite por $\mathcal{T}$ em tempo linear.

Observação

- A noção de complexidade espacial será agora indicada para máquinas de Turing com mais do que uma fita. Seja $\mathcal{T}$ uma destas máquinas e suponhamos que a 1^a fita é apenas de leitura. Se em vez de contarmos o número de passos dados por $\mathcal{T}$, contarmos o número de células das fitas de trabalho que a MT utiliza, define-se a função $sc_{\mathcal{T}}$, de *complexidade espacial* de $\mathcal{T}$.

- Pode-se ainda definir as classes de complexidade $DSPACE(f(n))$ e $NSPACE(f(n))$ de forma análoga àquela apresentada para as classes de complexidade $DTIME(f(n))$ e $NTIME(f(n))$.

- Note-se que $DTIME(f(n)) \subseteq DSPACE(f(n))$ e que $NTIME(f(n)) \subseteq NSPACE(f(n))$.