ITC: Introdução à Teoria da Computação

Marcos Castilho

DInf/UFPR

28 de julho de 2021

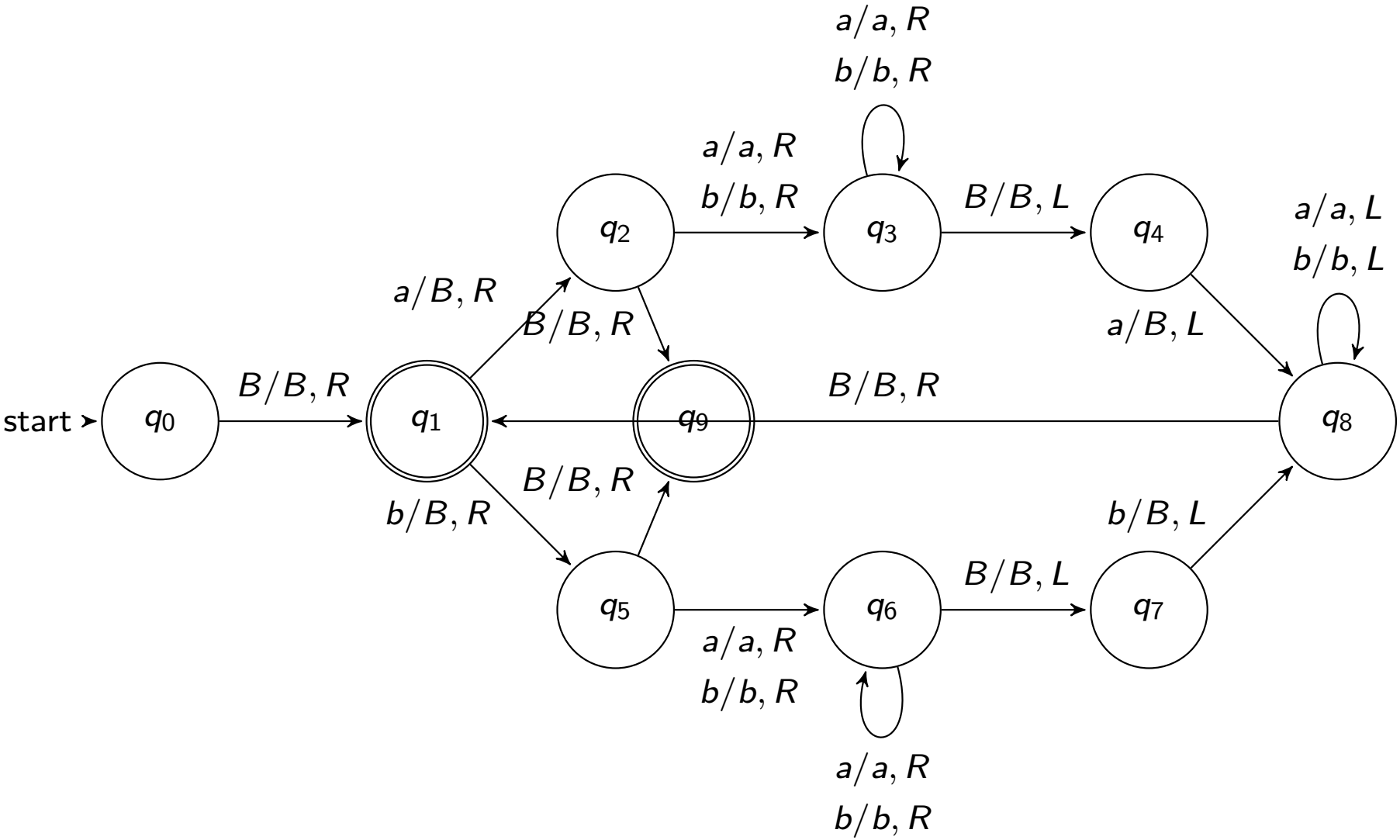
Complexidade Computacional

- A complexidade de tempo de uma máquina de Turing é medida pelo número de transições em uma computação;
- Normalmente se emprega a noção de taxas de crescimento para se descrever a complexidade de uma máquinade Turing;
- A complexidade de uma linguagem é determinada pela complexidade das máquinas que aceitam essa linguagem.

Complexidade de tempo de uma máquina de Turing

- A complexidade de tempo de uma computação mede a quantidade de trabalho dispendida por uma computação;
- A complexidade de tempo de uma máquina de Turing é quantificada pelo número de transições processadas;
- Vejamos um exemplo.

MT para os palíndromes sobre $\{a, b\}$



Análise do exemplo

- Uma computação de M consiste de um laço que compara o primeiro símbolo não branco na fita com o último.
- O primeiro símbolo é anotado e substituído por um branco a partir do estado q_1.
- O último não branco é comparado com esse e se torna branco também.
- A máquina então se move até o início, passando os não brancos e o ciclo é repetido.
- A palavra é aceita nos estados q_1 (tamanho par) ou q_9 (tamanho ímpar).

Análise do exemplo

- As partes superior e inferior do diagrama de estados são simétricas com relação à a e b.
- Logo, o número de transições em uma computação depende do tamanho da palavra de entrada.
- A complexidade de tempo é medida pelo trabalho requerido para uma palavra de tamanho fixo.

Definição: complexidade de tempo

Seja M uma MT padrão. A complexidade de tempo de M é a função $tc_M : N \rightarrow N$ tal que tc_M é o máximo número de transições processadas por uma computação de M quando iniciada com uma palavra de tamanho n.

Observação

- Assumimos que a MT em questão para para toda entrada;
- Não faz sentido fazer esta análise para máquinas que não param;
- A definição serve tanto para máquinas que reconhecem linguagens quanto aquelas que computam funções.

Análise do pior caso

- A complexidade de tempo nos dá o pior caso do desempenho de uma MT;
- Primeiramente porque consideramos os limites da computação algoritmica, queremos o mínimo de recursos necessários para garantir que uma computação termine;
- Em segundo lugar porque é mais fácil calcular isso do que a média.

Voltando ao exemplo dos palíndromes

- A computação de M termina quando toda a palavra de entrada é processada e substituída por brancos, ou então quando uma falha é encontrada;
- Logo, vamos nos concentrar na aceitação, não na rejeição (pior caso);
- Assim podemos obter valores iniciais da função tc_M:
- $(tc_M(0), tc_M(1), tc_M(2), \ldots)$.

Continuação do exemplo

- Mas queremos uma função geral, por isso é necessário uma análise cuidadosa;
- A máquina alterna sequências de movimentos para direita e esquerda e inicialmente está posicionada no primeiro branco da fita. Consideremos uma palavra de tamanho par.

Continuação do exemplo

- Movimentos para direita:
  - A máquina se move para direita, apagando o não branco mais a esquerda. O restante da palavra é lido e a máquina se encontra em q_4 ou q_7. Isto requer $k+1$ transições, onde k é a parte que não é branca na fita.
- Movimentos para esquerda:
  - M se move para esquerda, apagando o símbolo pareado e continua pulando os não brancos. Isto requer k movimentos.

Continuação do exemplo

- A computação entra em um ciclo apagando sempre dois símbolos até que tudo seja branco;
- O pior caso ocorre quando M aceita a palavra;
- Logo, a computação que aceita a palavra de tamanho n requer $n/2$ iterações.

Continuação do exemplo

Iteração	Direção	Transições
1	direita	n + 1
	esquerda	n
2	direita	n - 1
	esquerda	n - 2
3	direita	n - 3
	esquerda	n - 4
$n/2$	direita	1

Continuação do exemplo

- O número total de transições pela soma total em cada iteração.

$$tc_M(n) = \sum_{i=1}^{n+1} i = \frac{(n+2)(n+1)}{2}.$$

Exemplo 2: MT com duas fitas para o mesmo problema

- Uma MT com duas fitas pode ser construída assim:
  - Copia-se a entrada da fita 1 na fita 2;
  - Volta-se o cabeçote da fita 2 ao início;
  - Enquanto o cabeçote da fita 1 anda para esquerda, o da fita 2 anda para direita, comparando-se os símbolos e eventualmente, aceita.

Análise do exemplo 2

- O pior caso ocorre quando a entrada for um palíndrome, que fará a MT aceitá-la;
- Logo, requer: uma cópia, um retorno ao início e a comparação;
- Em suma, $tc_M = 3(n+1)+1$.

Observações

- A definição de complexidade de tempo é prevista para uma computação na máquina M e não na linguagem aceita pela máquina;
- Como uma mesma linguagem pode ser aceita por várias MTs diferentes, cada uma pode ter uma complexidade de tempo diferente;
- Para os exemplos acima, a primeira é aceita com tempo $(n^2 + 3n + 2)/2$, enquanto que a segunda em $3n + 4$;

Observações

- Dizemos que uma linguagem L é aceita em tempo determinístico $f(n)$, se existe uma máquina de Turing M determinística, de qualquer variedade, com $tc_M(n) \leq f(n), \forall n \in N$;
- Evidentemente, a máquina com duas fitas faz computações mais complexas em cada transição, e a complexidade de tempo é menor do que a outra, cujas transições são mais simples.

Taxas de crescimento

- A taxa de crescimento de uma função mede o aumento dos valores da função com relação ao aumento do tamanho da entrada;
- Ela é determinada pelo termo que mais contribui para o crescimento da função.
- Por exemplo, considere as funções n^2 e $n^2 + 2n + 5$

Tabela comparativa

n	0	5	10	25	50	100	1000
n^2	0	25	100	625	2500	10000	1000000
$n^2 + 2n + 5$	5	40	125	680	2605	10205	1002005

Taxas de crescimento

- Os termos lineares e constantes podem fazer alguma diferença para valores pequenos de n, mas conforme n cresce, eles passam a ser desprezíveis.
- Eles são os termos de mais baixa ordem, enquanto n^2 é o termo de mais alta ordem.

Definição: O grande

Sejam f e g duas funções numéricas. A função f é dita ser da ordem g, escrito $f = O(g)$, lido "f é O grande de g", se existe uma constante positiva e um natural n_0 tal que:

$$f(n) \leq cg(n)$$

Para todo $n \geq n_0$.

O grande

- A ordem de uma função nos dá um limitante superior para os valores da função conforme o tamanho da entrada cresce.
- A função f é da ordem g se seu crescimento é limitado por uma constante múltipla dos valores de g.
- Por causa da influência dos termos de menor ordem, a desigualdade $f(n) \leq cg(n)$ ocorre apenas para valores maiores do que algum número especificado.

O grande

- Duas funções f e g são ditas terem a mesma taxa de crescimento se $f = O(g)$ e $g = O(f)$.
- Assim:
  - $f(n) \leq c_1 g(n)$, para $n \geq n_1$; e
  - $g(n) \leq c_2 f(n)$, para $n \geq n_2$.

O grande

- Logo:
  - $f(n)/c_1 \leq g(n) \leq c_2 f(n)$;
  - $g(n)/c_2 \leq f(n) \leq c_1 g(n)$.
- E isso vale para todo n maior do que o máximo entre n_1 e n_2.

Exemplo

- Seja $f(n) = n^2 + 2n + 5$ e $g(n) = n^2$.
- Então $f = O(g)$ e $g = O(f)$.

Prova

- Obviamente $n^2 \leq n^2 + 2n + 5$, para todos os naturais, com $c = 1$ e $n_0 = 0$.
- Assim, claramente, $g = O(f)$.

Prova, continuação

- Para o outro lado, basta notar que:
- $2n \leq 2n^2$ e $5 \leq 5n^2$, para todo $n \geq 1$.
- Então:
  - $f(n) = n^2 + 2n + 5$
  - $\leq n^2 + 2n^2 + 5n^2$
  - $= 8n^2$
  - $= 8g(n)$.
- quando $n \geq 1$. Assim, $n^2 + 2n + 5 = O(n^2)$.

Exemplo 2

- Seja $f(n) = n^2$ e $g(n) = n^3$.
- Então: $f = O(g)$ e $g \neq O(f)$.
- Pois, obviamente, $n^2 = O(n^3)$ pois $n^2 \leq n^3$, para todo natural.
- Mas, se $n^3 = O(n^2)$, então existem constantes c e n_0 tais que $n^3 \leq cn^2$, para todo $n \geq n_0$. Mas se n_1 for o máximo entre $n_0 + 1$ e $c + 1$, então $n_1^3 = n_1 n_1^2 > cn_1^2$ e $n_1 > n_0$, o que é um absurdo.

Outras funções importantes

- Logarítmicas;
- Exponenciais;
- Fatoriais.

Observações

- As funções logaritmicas são limitadas polinomialmente;
- As funções Exponenciais e fatoriais não são. Elas apresentam taxa de crescimento exponenciais.
- Estas classes são importantes e serão úteis na sequência deste estudo.

Hierarquia O grande

- A eficiência de um algoritmo é comumente caracterizada pela sua taxa de crescimento.
- Um algoritmo com tempo polinomial é um cuja complexidade de tempo é limitada polinomialmente.
- A diferença deste para os não polinomiais aparece quando se considera o número de transições de uma computação conforme o tamanho da entrada cresce.

Curiosidade: número de transições com tc_M para um n

n	$log_2(n)$	n	n^2	2^n	$n!$
5	2	5	25	32	120
10	3	10	100	1024	3628800
40	5	40	1600	10^{12}	10^{47}
200	7	200	40000	10^{60}	10^{374}

Nomes para as funções

O grande	Nome
$O(1)$	constante
$O(log_a(n))$	logarítmico
$O(n)$	linear
$O(nlog_a(n))$	$n\ log\ n$
$O(n^2)$	quadrático
$O(n^3)$	cúbico
$O(n^r)$	polinomial, $r \geq 0$
$O(b^n)$	exponencial, $b \geq 1$
$O(n!)$	fatorial

Complexidade das variantes de MT

- Teorema: Seja L uma linguagem aceita por uma MT M com k trilhas, determinística com complexidade de tempo $tc_M(n) = f(n)$. Então L é aceita por uma MT padrão M' com complexidade de tempo $tc'_M(n) = f(n)$.
- Teorema: Seja L uma linguagem aceita por uma MT M com k fitas, determinística com complexidade de tempo $tc_M(n) = f(n)$. Então L é aceita por uma MT padrão N com complexidade de tempo $tc_N(n) = O(f(n)^2)$.

Complexidade não determinística

Definição: seja M uma MT não determinística. A complexidade de tempo de M é uma função $tc_M : N \to N$ tal que $tc_M(n)$ é o máximo número de transições processadas por uma computação, empregando qualquer escolha de transições, de uma entrada de tamanho n.

Exemplo

Seja o problema de reconhecer palíndromes.

- Uma MT não determinística com duas fitas.
- Os dois cabeçotes se movem para a direita enquanto a entrada é copiada para a fita 2.
- Uma transição que adivinha o meio da palavra faz com que o cabeçote da fita 1 se mova para a esquerda enquanto o outro continua se movendo para a direita e ao mesmo tempo comparando os símbolos.
- $tc_M(n) = n + 2$.

Complexidade de espaço

- Usamos uma MT com k fitas, a primeira é read-only e contém a entrada. As outras são fitas de trabalho. Na primeira fita, o cabeçote se move apenas até o final da string de entrada.
- O trabalho é feito nas outras fitas e queremos medir o espaço ocupado nelas, não na fita de entrada.
- Definição: Seja M uma MT com k fitas. conforme acima. A complexidade de espaço de M é a função $sc_M : N \to N$ tal que $sc_m(n)$ é o máximo número de espaços na fita lidos nas fitas de 2 até k por uma computação de M que é iniciada com uma palavra de tamanho n na fita de entrada.

Complexidade de espaço

- Teorema: Seja M uma MT com k fitas com complexidade de tempo $tc_M(n) = f(n)$. Então $sc_M(n) \leq (k-1)f(n)$.
- Teorema: Seja M uma MT com duas fitas com complexidade de espaço $sc_M(n) = f(n)$. Então $tc_m(n) \leq c^{f(n)}$, onde c é dependente de n, de $f(n)$, do número de símbolos da fita de M e do número de estados de M.

Licença

- Slides feitos em LaTeX usando beamer e tikz, editados com vim.
- Licença