Aula 05 – Complexidade de Tempo

Prof. Eduardo Zambon

Departamento de Informática (DI)
Centro Tecnológico (CT)
Universidade Federal do Espírito Santo (Ufes)

Algoritmos e Fundamentos da Teoria de Computação (ToCE)
Engenharia de Computação

Introdução

- Decidibilidade: estudo do limite possível da computação algorítmica, i.e., por Máquinas de Turing (TMs).
- Complexidade: estudo dos recursos (tempo e espaço) necessários para a computação de uma TM.
- **Estes *slides***: Como caracterizar o tempo de execução de uma TM?
- **Objetivos**: Introduzir o conceito de complexidade de tempo para TMs.

Referências

Chapter 14 & Section 15.1
T. Sudkamp
Section 7.1 – Measuring Complexity
M. Sipser
Section 6.1 – The Running Time of Algorithms
A. Maheshwari

Introdução

- Até agora: caracterização do conjunto de problemas resolvíveis e funções computáveis.
- Preocupação era em demonstrar a existência de uma solução algorítmica para um problema.
- A partir de agora: queremos analisar a complexidade de um algoritmo.
- A complexidade é medida pelos recursos necessários para se obter uma solução.

Introdução

- A partir de agora: todos os problemas são decidíveis.
- Não faz sentido analisar a complexidade de um problema que não admite uma solução.
- Existem problemas que teoricamente são resolvíveis mas que não o são na prática por exigirem uma quantidade absurda de recursos.
- Problemas intratáveis: problemas para os quais não existem algoritmos eficientes.
- Teoria de Complexidade serve para distinguir problemas que são tratáveis dos intratáveis.

Introdução

- Queremos determinar a complexidade inerente ao um problema.
- $\Rightarrow$ A análise deve ser independente de uma implementação (arquitetura) em particular.
- Para isolar as características do problema das características da implementação é necessário escolher um método de computação genérico.
- A máquina de Turing padrão é "arquitetura" escolhida.
- Não impõe nenhuma restrição de tempo ou memória disponível.
- A Tese de Church-Turing (CTT) garante que qualquer procedimento efetivo pode ser implementado em uma TM.

Medidas de Complexidade

- Dois tópicos principais em Teoria de Complexidade:
  1. Análise de algoritmos que resolvem um problema em particular.
  2. Comparação da dificuldade inerente de diferentes problemas.
- A análise da complexidade de um algoritmo requer:
  1. A identificação dos recursos a serem considerados.
  2. A caracterização da medida da complexidade da entrada.
  3. A determinação de uma função de complexidade que descreve os recursos necessários para a computação com uma dada complexidade da entrada.
- Exemplos de problemas a seguir.

Medidas de Complexidade

Ordenação de um vetor de inteiros

Input: Vetor $v[1..n]$.

Output: Vetor $v'[1..n]$ com elementos ordenados.

- Complexidade da entrada: tamanho n do vetor.
- Recurso medido: número de movimentos dos elementos.

Medidas de Complexidade

Cálculo do quadrado de uma matriz

Input: Uma matriz B com dimensões $n \times n$.

Output: Matriz $C = B^2$.

- Complexidade da entrada: dimensões n da matriz.
- Recurso medido: número de multiplicações escalares.

Caminho em Grafos Direcionados

Input: Grafo direcionado $G = (\mathrm{N}, \mathrm{A})$, nós $v_i, v_j \in \mathrm{N}$.

Output: sim; se existe um caminho em G de v_i até v_j
não; caso contrário.

- Complexidade da entrada: número de nós do grafo.
- Recurso medido: número de nós visitados na busca pelo caminho.

Medidas de Complexidade

Aceite por uma TM M (que para para todas as entradas)

Input: string w.

Output: sim; se M aceita w
não; caso contrário.

- Complexidade da entrada: tamanho da string de entrada.
- Recurso medido: número de transições da computação.

Medidas de Complexidade

- Comparação de problemas distintos requer que as soluções sejam implementadas em um mesmo sistema algorítmico.
- $\Rightarrow$ TM, pois CTT garante que qualquer algoritmo pode ser implementado como uma TM.
- Medida unificada da complexidade da entrada: tamanho da string de entrada para a TM.
- Complexidade de tempo de uma TM: número de transições executadas pela máquina.
- Complexidade de espaço de uma TM: número de quadrados da fita necessários para a computação.

Taxas de Crescimento

- Em geral, não é necessário obter a relação exata entre a complexidade da entrada e a utilização do recurso.
  - Cálculo preciso é bastante complicado.
  - Análise provê mais informação que o necessário.
- Por isso, a complexidade de tempo é indicada pela taxa de crescimento da função de complexidade, ao invés da própria função.
- Taxa de crescimento de uma função: medida assintótica do valor da função quando a entrada cresce arbitrariamente.

Taxas de Crescimento

Intuitivamente, a taxa de crescimento é determinada pelo termo mais significativo.

TABLE 14.3.1 Growth of Functions

n	0	5	10	25	50	100	1,000
$20n + 500$	500	600	700	1,000	1,500	2,500	20,500
n^2	0	25	100	625	2,500	10,000	1,000,000
$n^2 + 2n + 5$	5	40	125	680	2,605	10,205	1,002,005
$n^2/(n^2 + 2n + 5)$	0	0.625	0.800	0.919	0.960	0.980	0.998

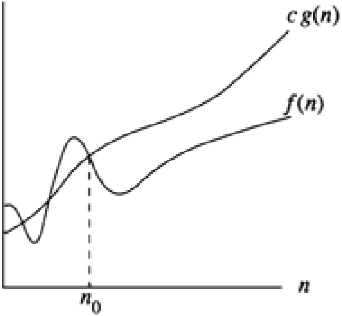
- Os termos linear e constante em $n^2 + 2n + 5$ são ditos termos de baixa ordem.
- Termos de baixa ordem só influenciam no valor da função para valores pequenos de n.

Taxas de Crescimento

Definição 14.2.1 (Sudkamp)

Sejam f e g duas funções numéricas unárias e totais.

1. Função f é de ordem g se existe $c > 0$ e $n_0 \in \mathbf{N}$ tal que $f(n) \leq c \cdot g(n)$ para todo $n \geq n_0$.
2. O conjunto de todas as funções de ordem g é denotado por $O(g)$ (Big Oh de g).



$f \in O(g)$

g é um limite superior assintótico de f.

Ex.: se $f(n) = n^2$ e $g(n) = n^3$ então $f \in O(g)$ e $g \notin O(f)$.

Taxas de Crescimento

A notação de Big Oh pode ser usada para descrever a relação entre o grau de um polinômio e sua taxa de crescimento.

Teorema 14.2.2 (Sudkamp)

Seja f um polinômio de grau r. Então:

1. $f \in O(n^r)$
2. $n^r \in O(f)$
3. $f \in O(n^k)$ para todo $k > r$
4. $f \notin O(n^k)$ para todo $k < r$.

Uma das consequências do teorema acima é que a taxa de crescimento de qualquer polinômio pode ser caracterizado por uma função da forma n^r.

Taxas de Crescimento

Teorema 14.2.3 (Sudkamp)

Seja r um natural e a, b números reais > 1.

1. $\log_a(n) \in O(n)$
2. $n \notin O(\log_a(n))$
3. $n^r \in O(b^n)$
4. $b^n \notin O(n^r)$
5. $b^n \in O(n!)$
6. $n! \notin O(b^n)$.

Hierarquia Big Oh

Big Oh	**Name**
$O(1)$	constant
$O(\log_a(n))$	logarithmic
$O(n)$	linear
$O(n \log_a(n))$	$n \log n$
$O(n^2)$	quadratic
$O(n^3)$	cubic
$O(n^r)$	polynomial $r \geq 0$
$O(b^n)$	exponential $b > 1$
$O(n!)$	factorial

Taxas de Crescimento

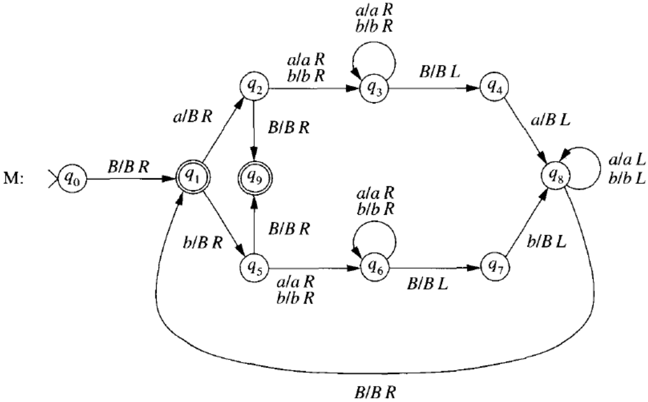
- Uma função f é dita (com limite) polinomial se $f \in O(n^r)$ para algum natural r.
- Funções com limite polinomial formam uma família importante de funções que são associadas com a complexidade de tempo de algoritmos eficientes.
- Uma função f que está em $O(b^n)$ ou $O(n!)$ é dita (com crescimento) exponencial.
- Funções com crescimento exponencial formam uma família de funções associadas com a complexidade de tempo de problemas intratáveis.
- A tabela do próximo slide justifica a distinção entre algoritmos polinomiais e não-polinomiais.

Taxas de Crescimento de algumas funções:

	$\log_2(n)$	n	n^2	n^3	2^n	$n!$
5	2	5	25	125	32	120
10	3	10	100	1,000	1,024	3,628,800
20	4	20	400	8,000	1,048,576	$2.4 \cdot 10^{18}$
30	4	30	900	27,000	$1.0 \cdot 10^{9}$	$2.6 \cdot 10^{32}$
40	5	40	1,600	64,000	$1.1 \cdot 10^{12}$	$8.1 \cdot 10^{47}$
50	5	50	2,500	125,000	$1.1 \cdot 10^{15}$	$3.0 \cdot 10^{64}$
100	6	100	10,000	1,000,000	$1.2 \cdot 10^{30}$	$> 10^{157}$
200	7	200	40,000	8,000,000	$1.6 \cdot 10^{60}$	$> 10^{374}$

Complexidade de Tempo de uma DTM

Seja a máquina M abaixo que aceita palíndromos sobre $\{a, b\}$.



Uma computação de M consiste de um *loop* que compara o primeiro símbolo à esquerda com o último à direita.

Complexidade de Tempo de uma DTM

- Computações de M são simétricas com relação aos símbolos *a* e *b*.
- A tabela abaixo lista todas as computações significativas para strings de tamanho 0, 1, 2 e 3.

Length 0	Length 1	Length 2		Length 3	
q_0BB	q_0BaB	q_0BaaB	q_0BabB	q_0BabaB	q_0BaabB
$\vdash Bq_1B$	$\vdash Bq_1aB$	$\vdash Bq_1aaB$	$\vdash Bq_1abB$	$\vdash Bq_1abaB$	$\vdash Bq_1aabB$
	$\vdash BBq_2B$	$\vdash BBq_2aB$	$\vdash BBq_2bB$	$\vdash BBq_2baB$	$\vdash BBq_2abB$
	$\vdash BBBq_9B$	$\vdash BBaq_3B$	$\vdash BBbq_3B$	$\vdash BBbq_3aB$	$\vdash BBaq_3bB$
		$\vdash BBq_4aB$	$\vdash BBq_4bB$	$\vdash BBbaq_3B$	$\vdash BBabq_3B$
		$\vdash Bq_8BBB$		$\vdash BBbq_4aB$	$\vdash BBaq_4bB$
		$\vdash BBq_1BB$		$\vdash BBq_8bBB$	
				$\vdash Bq_8BbBB$	
				$\vdash BBq_1bBB$	
				$\vdash BBBq_5BB$	
				$\vdash BBBBq_9B$	

Complexidade de Tempo de uma DTM

- Como esperado, o número de transições em uma computação depende da string de entrada.
- É impraticável tentar determinar o número exato de transições para cada string de entrada.
- Assim, a complexidade de tempo de uma TM mede a quantidade máxima de trabalho exigida pelas strings de um dado tamanho.

Definição 14.3.1 (Sudkamp)

Seja M uma TM padrão. A complexidade de tempo de M é a função $tc_{\mathrm{M}} : \mathbf{N} \to \mathbf{N}$ tal que $tc_{\mathrm{M}}(n)$ é o número máximo de transições processadas por uma computação de M quando iniciada com uma string de entrada de tamanho n.

Complexidade de Tempo de uma DTM

- Definição de complexidade de tempo mede a performance de pior caso da TM.
- Definição de tc_{M} serve tanto para TMs que aceitam linguagens quanto para as que computam funções.
- Assumimos que as TMs sob análise param para todas as entradas.
- Não faz sentido falar de complexidade de tempo de uma computação que não termina.

Complexidade de Tempo de uma DTM

- Retornando à TM M que aceita palíndromos sobre $\{a, b\}$.
- Computação de M termina quando:
  - M aceita: string de entrada é totalmente reescrita por brancos.
  - M rejeita: quando o primeiro par de símbolos não correspondentes é encontrado.
- Complexidade de tempo mede a performance de pior caso.
- $\Rightarrow$ Só interessam as strings para as quais as computações levam M a fazer o maior número possível de ciclos.
- Para a máquina M, isso acontece quando a entrada é aceita.
- Valores iniciais de tc_{M} a partir da tabela:

$$tc_{\mathrm{M}}(0) = 1 \qquad tc_{\mathrm{M}}(1) = 3 \qquad tc_{\mathrm{M}}(2) = 6 \qquad tc_{\mathrm{M}}(3) = 10.$$

Complexidade de Tempo de uma DTM

- Restante dos valores de n. Assuma k o número atual de símbolos não-brancos da fita. O *loop* da máquina é:
  - Movimento à direita: apaga o símbolo mais à esquerda e anda até o final da string. Requer $k + 1$ transições.
  - Movimento à esquerda: apaga o símbolo mais à direita e anda até o início da string. Requer k transições.

Iteration	Direction	Transitions
1	right	$n + 1$
	left	n
2	right	$n - 1$
	left	$n - 2$
3	right	$n - 3$
	left	$n - 4$
$\vdots$		$\vdots$
$n/2$	right	1

- Computação que aceita uma strings de tamanho n leva $n/2$ iterações.
- Número total de transições: soma da coluna da tabela ao lado.
- Corresponde à soma dos primeiros $n + 1$ naturais.

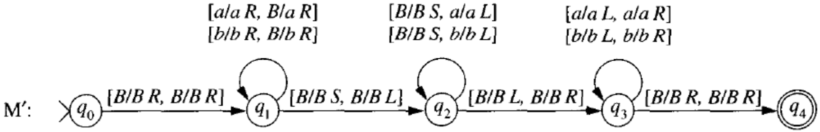
Complexidade de Tempo de uma DTM

- Assim, a complexidade de tempo de M é dada pela função:

$$tc_{\mathrm{M}}(n) = \sum_{i=1}^{n+1} i = \frac{(n+2)(n+1)}{2} \in O(n^2).$$

Exemplo 14.3.1 (Sudkamp)

A máquina de duas fitas M′ abaixo também aceita os palíndromos sobre $\{a, b\}$.



- Para uma entrada de tamanho n, o número máximo de transições de M′ ocorre quando a string é um palíndromo.
- Uma computação de aceite requer três etapas:
  - Copiar a entrada da fita 1 para fita 2: $n + 1$ transições.
  - Rebobinar a cabeça fita 1: $n + 1$ transições.
  - Comparar as duas fitas: $n + 1$ transições.
- Considerando a transição inicial, vemos que a complexidade de tempo de M′ é

$$tc_{M'}(n) = 3(n + 1) + 1 \in O(n).$$

Complexidade e Variações de DTMs

- Para o estudo de Decidibilidade, o modelo da TM utilizado era irrelevante.
- Qualquer problema decidível em um tipo de TM também é decidível usando-se qualquer outro modelo.
- Para o estudo de Complexidade, o modelo de TM tem influência total na análise.
- Exemplo anterior: M e M' resolvem o mesmo problema mas com complexidade de tempo distintas.
- Teoremas a seguir generalizam a relação da complexidade de tempo de variantes de TMs.

Complexidade e Variações de DTMs

Teorema 14.4.1 (Sudkamp)

- Seja L uma linguagem aceita por uma DTM M de *k*-faixas, com complexidade de tempo $tc_{\mathrm{M}}(n)$.
- Então L é aceita por uma TM padrão M′ com complexidade de tempo $tc_{\mathrm{M}'}(n) = tc_{\mathrm{M}}(n)$.

Teorema 14.4.2 (Sudkamp)

- Seja L uma linguagem aceita por uma DTM M de *k*-fitas, com complexidade de tempo $tc_{\mathrm{M}}(n) = f(n)$.
- Então L é aceita por uma TM padrão M′ com complexidade de tempo $tc_{\mathrm{M}'}(n) \in O(f(n)^2)$.

Complexidade de Tempo de uma NTM

- Computações determinísticas e não-determinísticas são fundamentalmente distintas.
- TM determinística (DTM): gera e examina sequencialmente diferentes possibilidades em busca por uma solução.
- TM não-determinística (NTM): "advinha" uma possibilidade e verifica se é uma solução. (*Guess-and-check*)
- NTM só precisa determinar se uma possibilidade é uma solução.
- Computação inerente à NTM cuida das escolhas não-determinísticas.

Complexidade de Tempo de uma NTM

- *Exemplo*: determinar se um natural k é composto (não-primo).
- Solução por DTM: examinar sequencialmente os números no intervalo $2, \ldots, \lfloor\sqrt{k}\rfloor$ para ver se algum é um fator de k.
- Solução por NTM: escolher arbitrariamente um valor no intervalo acima e testar se é um fator de k.
- Se k é composto, ao menos uma das escolhas não-determinísticas encontra um fator de k.
- $\Rightarrow$ Computação da NTM dá uma resposta positiva.

Complexidade de Tempo de uma NTM

- Uma string é aceita por NTM se ao menos uma computação termina em um estado de aceite.
- O aceite da string não é afetado pela existência de outras computações que param em um estado de rejeição.
- Por outro lado, a performance de pior caso de uma NTM mede a eficiência sobre todas as possíveis computações.
- Importante: no estudo de complexidade, assume-se que todas as computações de uma NTM sempre terminam.

Complexidade de Tempo de uma NTM

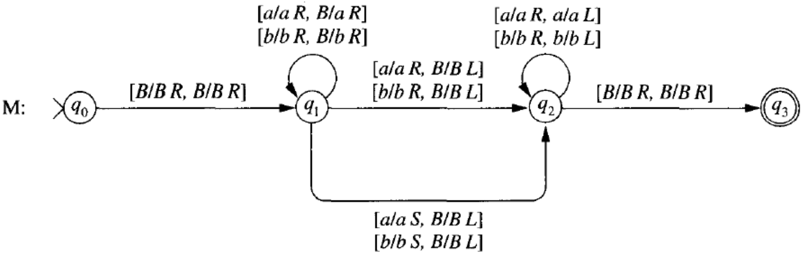
Definição 15.1.1 (Sudkamp)

Seja M uma NTM. A complexidade de tempo de M é a função $tc_{\mathrm{M}} : \mathbf{N} \to \mathbf{N}$ tal que $tc_{\mathrm{M}}(n)$ é o número máximo de transições processadas por uma computação, **através de quaisquer escolhas de transições**, para uma string de entrada de tamanho n.

- Definição acima é praticamente idêntica à da DTM. Difere apenas na parte em **negrito**.
- NTMs que usam uma estratégia de *guess-and-check* em geral são mais simples que as suas DTMs equivalentes.
- Esta simplicidade reduz o número de transições necessárias para uma computação.
- Isso é ilustrado no exemplo a seguir.

Complexidade de Tempo de uma NTM

A NTM de duas-fitas M aceita os palíndromos sobre $\{a, b\}$.



- Ambas cabeças andam para a direita com a entrada sendo copiada para a fita 2.
- As transições saindo de q_1 "advinham" o centro da string.
- A seguir a fita 1 continua para a direita e a fita 2 vai para a esquerda, comparando a segunda parte da string com a primeira.

Complexidade de Tempo de uma NTM

- O número máximo de transições de M ocorre quando uma string de tamanho par é aceita.
- A complexidade de tempo é $tc_{\mathrm{M}}(n) = n + 3$.
- O teorema a seguir provê um limite superior da complexidade de tempo de uma DTM equivalente a uma NTM.

Teorema 15.1.2 (Sudkamp)

- Seja L a linguagem decidida por uma NTM M com complexidade de tempo $tc_{\mathrm{M}}(n) = f(n)$.
- Então L é decidida por uma DTM M′ com complexidade de tempo $tc_{\mathrm{M'}}(n) \in O(f(n)c^{f(n)})$, onde c é número máximo de transições para qualquer par de estado e símbolo em M.

Complexidade de Tempo de uma NTM

- Prova do teorema anterior se baseia na construção de M′, que simula todas as computações de M.
- Serve como um limite superior genérico.
- No entanto, em muitos casos é possível construir uma DTM muito mais eficiente.
- Exemplo: problema da linguagem dos palíndromos.
  - NTM duas-fitas M com $tc_{\mathrm{M}}(n) = n + 3$.
  - DTM duas-fitas M′ com $tc_{\mathrm{M}'}(n) = 3(n + 1) + 1$.
  - DTM padrão M″ com $tc_{\mathrm{M}''}(n) = (n + 2)(n + 1)/2$.
- Na próxima aula vamos caracterizar as classes de problemas resolvíveis em tempo polinomial de forma determinística e não-determinística.

Aula 05 – Complexidade de Tempo

Prof. Eduardo Zambon

Departamento de Informática (DI)
Centro Tecnológico (CT)
Universidade Federal do Espírito Santo (Ufes)

Algoritmos e Fundamentos da Teoria de Computação (ToCE)
Engenharia de Computação