UDESC

Teoria da Computação

Prof. Diego Buchinger
diego.buchinger@outlook.com
diego.buchinger@udesc.br

UDESC

Introdução à Complexidade

UDESC

Introdução

- Temos dois tipos de problemas:
  - Insolúveis
  - Solúveis
- Mas será que os problemas “Solúveis” são todos tratáveis na prática – i.e. é possível resolvê-los de forma eficiente?

UDESC

Introdução

- Temos dois tipos de problemas:
  - Insolúveis
  - Solúveis
- Mas será que os problemas “Solúveis” são todos tratáveis na prática – i.e. é possível resolvê-los de forma eficiente?
  - **No!** (ex: solução demora 1 século para computar)
  - Alguns consomem demais os recursos (tempo e/ou espaço)

- Vamos nos ater ao tempo primeiramente...
- Como calcular o tempo que uma MT, ou um autômato, ou mesmo um computador leva para computar um dado problema?

Introdução

- Considere a linguagem $A = \{0^k 1^k \mid k \geq 0\}$
  - Quanto tempo uma MT de uma única fita precisa para decidir A?
  - Quantos passos uma MT de uma fita precisa para decidir A?

M_1 = "Sobre a cadeia de entrada *w*:

1. Faça uma varredura na fita e *rejeite* se for encontrado algum 0 a direita de algum 1
2. Repita se existem ambos, 0s e 1s, na fita:
3. Faça uma varredura na fita, cortando um único 0 e um único 1
4. Se ainda permanecerem 0s após todos os 1s tiverem sido cortados ou se permanecerem 1s após todos os 0s tiverem sido cortados, *rejeite*. Caso contrário (sem 0s e 1s na fita), *aceite*.

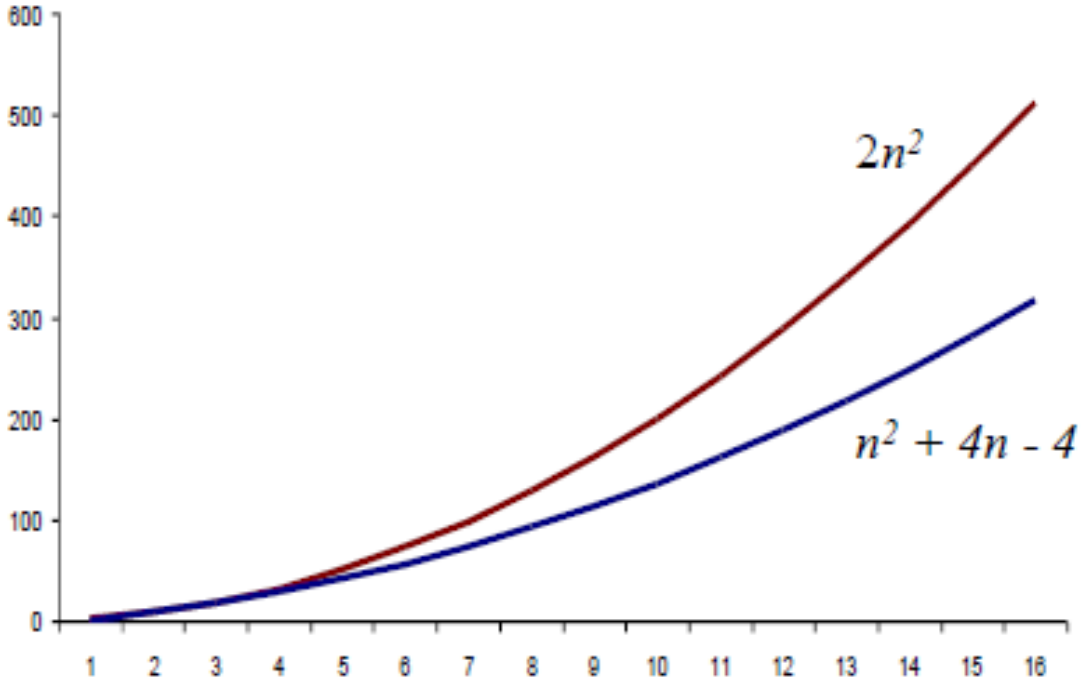
UDESC

Introdução

- O número de passos que um algoritmo usa sobre uma entrada específica pode depender de vários parâmetros:
  - Usualmente tamanho da entrada: vetor, texto, número grande;
  - A configuração inicial da entrada (fita);
  - Em um grafo: vértices, arestas, grau máximo do grafo.
- O número de passos também é influenciado pelo modelo de máquina utilizada!
- É comum analisarmos duas situações específicas
  - Análise pessimista ou do pior caso
  - Análise otimista ou do melhor caso

Notação Assintótica
(Notação O grande – Limite Superior)

Uma função $g(n)$ domina assintoticamente outra função $f(n)$ se existem duas constantes positivas c e n_0 tais que, para $n > n_0$, temos $|f(n)| \leq c.|g(n)|$ → $f(n) = O(g(n))$



$n^2 + 4n - 4 = O(n^2)$

$2n^2 = O(n^2)$

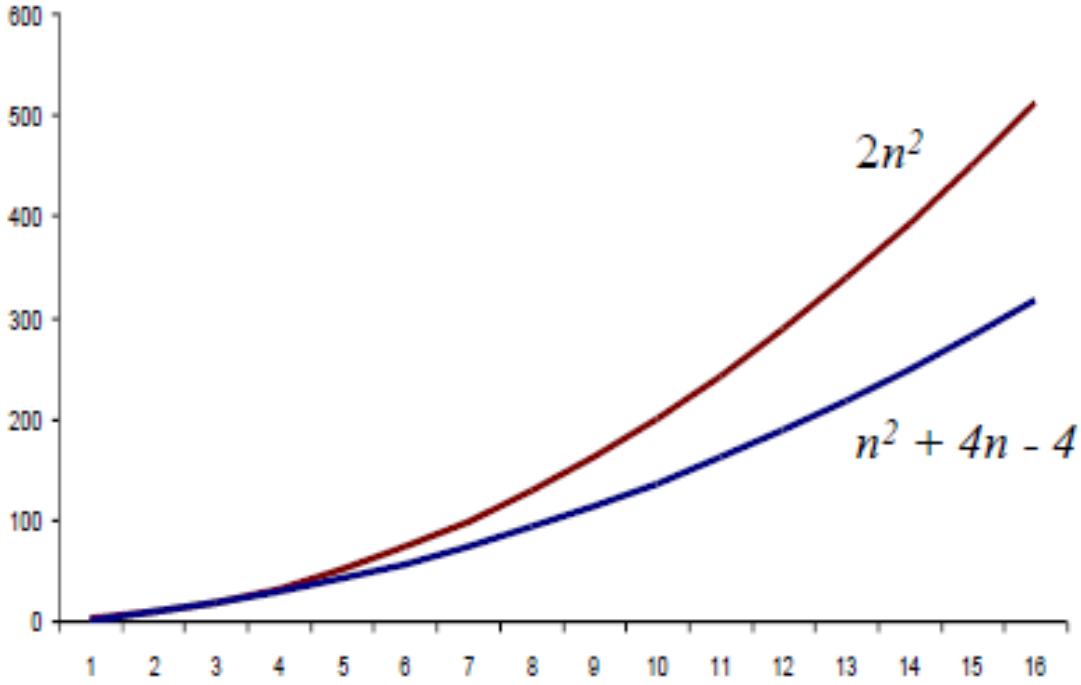
Notação Assintótica
(Notação O grande – Limite Superior)

- $f(n) = n^2 + 4n - 4$ g(n) = $O(n^2)$
- $f(n) = 2n^2$ g(n) = $O(n^2)$ [O(n) ?]

$f(n) / c$ & n_0	$c=1$ $n_0=3$	$c=1$ $n_0=50$	$c=2$ $n_0=1$	$c=2$ $n_0=10$	$c=3$ $n_0=2$	$c=3$ $n_0=10$
$2n^2$					-	-
$c(n^2)$						
$n^2 + 4n - 4$						
$c(n)$						

Notação Assintótica
(Notação O pequeno – *Little-O*)

Uma função $g(n)$ domina assintoticamente outra função $f(n)$ se existem duas constantes positivas c e n_0 tais que, para $n > n_0$, temos $|f(n)| < c.|g(n)|$ → $f(n) = o(g(n))$



$n^2 + 4n - 4 \neq o(n^2)$

$n^2 \neq o(n^2)$

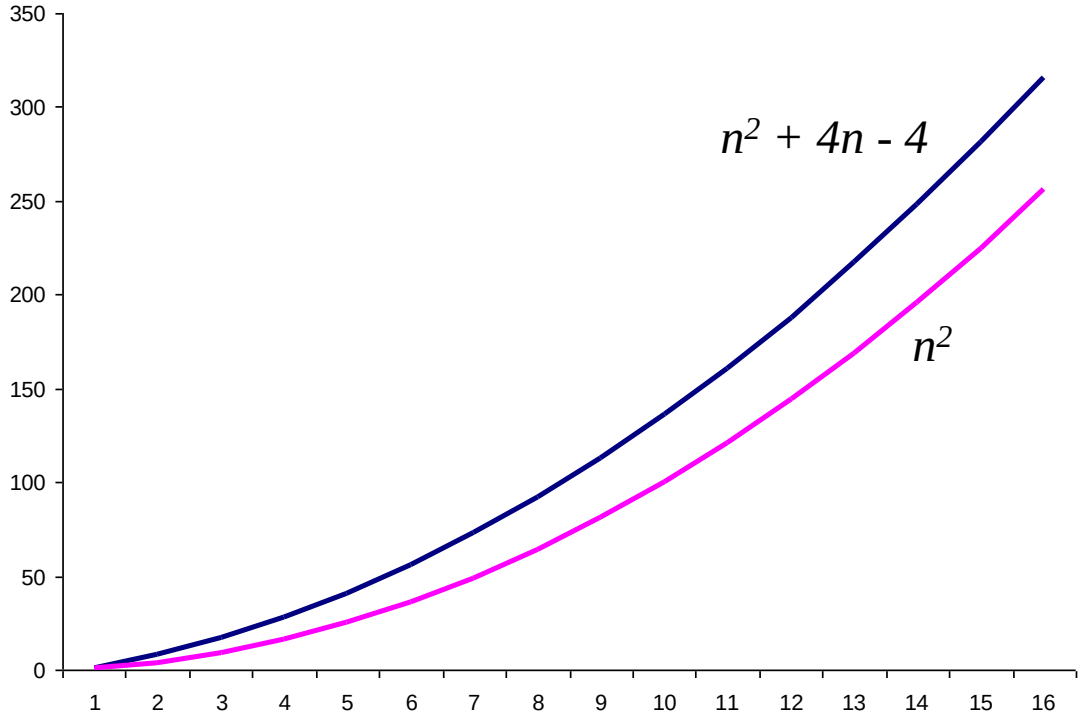
$4n - 8 = o(n^2)$

$n = o(n^2)$

Notação Assintótica

Notação Omega Grande – Limite Inferior

Uma função *f(n)* é o limite inferior de outra função *g(n)* se existem duas constantes positivas *c* e n_0 tais que, para $n > n_0$, temos $|g(n)| \geq c.|f(n)|$, $g(n) = \Omega(f(n))$.



$$n^2 = \Omega(\log n)$$

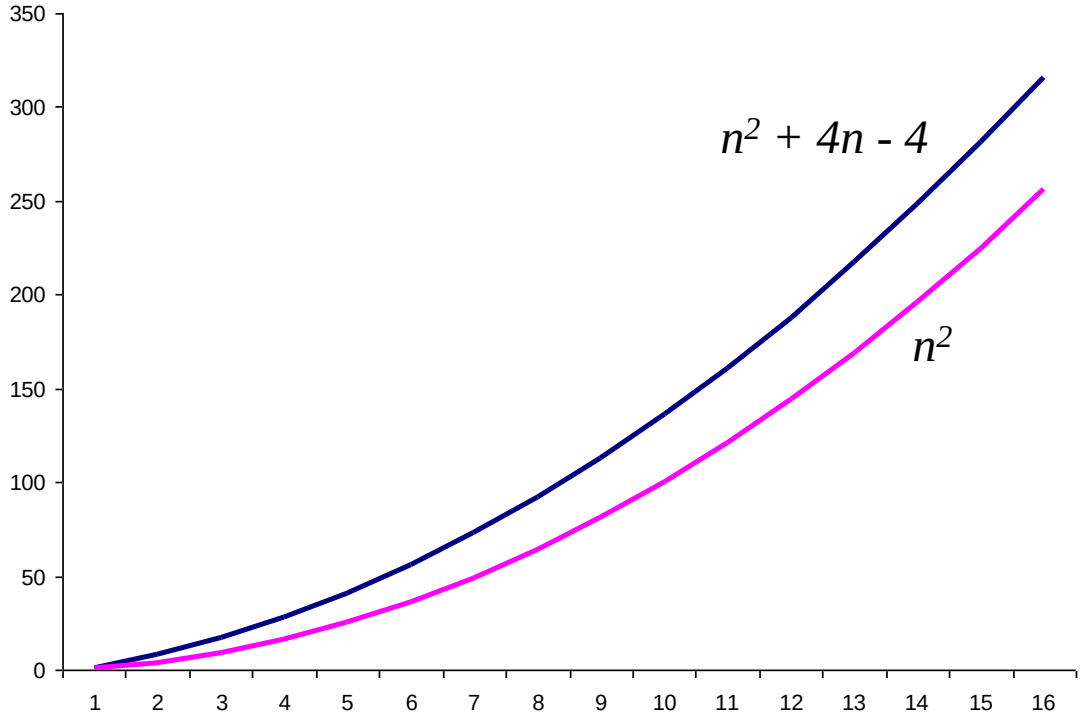
$$n^2 + 4n - 4 = \Omega(n^2)$$

$$n^2 = \Omega(n^2)$$

Notação Assintótica

Notação Omega Pequeno – Limite-Omega

Uma função *f(n)* é o limite inferior de outra função *g(n)* se existem duas constantes positivas *c* e n_0 tais que, para $n > n_0$, temos $|g(n)| > c.|f(n)|$, $g(n) = \omega(f(n))$.



$$n^2 + 4n - 4 \neq \omega(n^2)$$

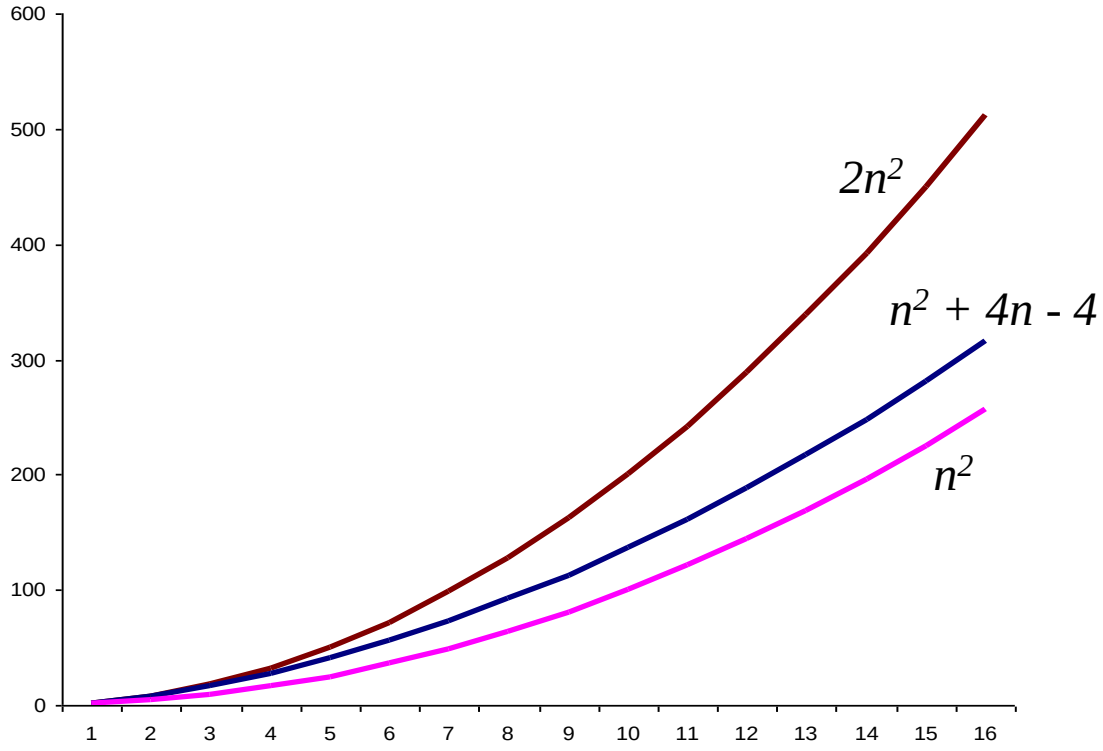
$$n^2 \neq \omega(n^2)$$

$$n^2 + 4n - 4 = \omega(n)$$

$$n^2 = \omega(n)$$

Notação Assintótica
Limite Firme (Notação Θ)

Uma função *f(n)* é o limite restrito (ou exato) de outra função *g(n)* se existem três constantes positivas c_1, c_2, e n_0 tais que, para $n > n_0$, temos $c_1.|f(n)| \geq |g(n)| \geq c_2.|f(n)|$, $g(n) = \Theta(f(n))$



[funções crescem com mesma rapidez]

$$n^2 + 4n - 4 = \Theta(n^2)$$

UDESC

Algumas Operações com Notação *O*

$c.O(f(n)) = O(f(n))$, onde c é uma constante.

$O(f(n)) + O(g(n)) = O(MAX(f(n), g(n))$

$n.O(f(n)) = O(n \cdot f(n))$

$O(f(n)) \cdot O(g(n)) = O(f(n) \cdot g(n))$

UDESC

Funções de Complexidade

- Algumas funções são consideradas equivalentes assintoticamente – i.e. possuem um mesmo “peso”

- Quais são as funções genéricas que não são equivalentes?
  - Vamos testar alguns casos!?

Funções de Complexidade

Determine a quantidade de passos necessários para cada situação hipotética abaixo considerando os vários valores para *n*:

$f(n) / n$	*n=10*	*n=100*	*n=1.000*	*n=10.000*	*n=100.000*	*n=1.000.000*
n						
$log_{10} n$						
1						
n^2						
$3n$						
$\sqrt{n}$						
$n\ log_{10} n$						
2^n						
$n!$						

Funções de Complexidade

Podemos classificar os tempos de execução em:

$$1 < \log\log n < \log n < n^{\varepsilon} < n^{c} < n^{\log n} < c^{n}$$

Definição: definimos uma classe de complexidade de tempo denominada TIME(t(n)) como a coleção de todas as linguagens que são decidíveis por uma MT de tempo O(t(n))

Análise de Complexidade de Tempo

Voltemos ao nosso exemplo: $A = \{0^k 1^k \mid k \geq 0\}$

- Considere que a entrada é composta por n símbolos
- Qual é a complexidade de tempo do algoritmo apresentado?

M_1 = "Sobre a cadeia de entrada *w*:

1. Faça uma varredura na fita e *rejeite* se for encontrado algum 0 a direita de algum 1
2. Repita se existem ambos, 0
3. Faça uma varredura na f
4. Se ainda permanecerem 0s
   se permanecerem 1s após t
   Caso contrário (sem 0s e 1s

Note que não foi mencionado o reposicionamento do cabeçote no início da fita na descrição do passo 1!

A notação assintótica nos permite omitir detalhes da descrição da máquina que afetam o tempo de execução por, **no máximo**, um fator **constante**!

Análise de Complexidade de Tempo

Voltemos ao nosso exemplo: $A = \{0^k 1^k \mid k \geq 0\}$

- Considere que a entrada é composta por n símbolos
- Qual é a complexidade de tempo do algoritmo apresentado?

M_1 = "Sobre a cadeia de entrada *w*:

1. Faça uma varredura na fita e *rejeite* se for encontrado algum 0 a direita de algum 1
2. Repita se existem ambos, 0s e 1s, na fita:
3. Faça uma varredura na fita, cortando um único 0 e um único 1
4. Se ainda permanecerem 0s após todos os 1s tiverem sido cortados ou se permanecerem 1s após todos os 0s tiverem sido cortados, *rejeite*. Caso contrário (sem 0s e 1s na fita), *aceite*.

Análise de Complexidade de Tempo

Voltemos ao nosso exemplo: $A = \{0^k 1^k \mid k \geq 0\}$

- Utilizando M_1 mostramos que $A = O(n^2)$

  logo, $A \in TIME(n^2)$

- Mas será que $A = O(n^2)$ ou $A = o(n^2)$?

- Alguma ideia para construir uma máquina melhor?

  (=mais eficiente)

UDESC

Análise de Complexidade de Tempo

- Alguma ideia para construir uma máquina melhor?
  - Ao invés de cortar apenas um 1 e 0 cortar dois a cada passo
    - Isso ajuda?
    - Qual é a complexidade de tempo?

Análise de Complexidade de Tempo

- Alguma ideia para construir uma máquina melhor?
  - ~~Ao invés de cortar apenas um 1 e 0 cortar dois a cada passo~~
  - Proposta B:

M_2 = "Sobre a cadeia de entrada *w*:

1. Faça uma varredura na fita e *rejeite* se for encontrado algum 0 a direita de algum 1
2. Repita se existem ambos, 0s ou 1s, na fita:
3. Faça uma varredura na fita, verificando se o número total de 0s e 1s remanescentes é par ou ímpar. Se for ímpar, *rejeite*
4. Faça uma varredura novamente na fita, cortando alternadamente um 0 sim e outro não começando com o primeiro 0, e, então, cortando alternadamente um 1 sim e outro não começando com o primeiro 1.
5. Se nenhum 0 e nenhum 1 permanecerem na fita, *aceite*. Caso contrário, *rejeite*.

Análise de Complexidade de Tempo

- Alguma ideia para construir uma máquina melhor?
  - ~~Ao invés de cortar apenas um 1 e 0 cortar dois a cada passo~~
  - Proposta B – M_2 => O(n log n)

    Assim podemos dizer que $A \in TIME(n \log n)$

    Mas será que A = O(n log n) ou A = o(n log n) ?

Análise de Complexidade de Tempo

- Alguma ideia para construir uma máquina melhor?
  - ~~Ao invés de cortar apenas um 1 e 0 cortar dois a cada passo~~
  - Proposta B – M_2 => O(n log n)

    Assim podemos dizer que $A \in TIME(n \log n)$

    Mas será que A = O(n log n) ou A = o(n log n) ?

    No!!

    É possível provar que qualquer linguagem que pode ser decidida em tempo o(n log n) em uma MT de uma fita é regular!

UDESC

Relacionamentos de Complexidade entre Modelos Computacionais

UDESC

Desempenho Computacional

- Já mostramos que certas modificações em uma MT não aumentam a sua capacidade computacional.

- Mas será que essas modificações que definem um **modelo computacional** não melhoram o desempenho da computação (i.e. realizar a mesma coisa de maneira mais rápida assintoticamente)?

Desempenho Computacional

Voltemos ao nosso exemplo: $A = \{0^k 1^k \mid k \geq 0\}$

- Será que é possível construir uma MT de duas fitas que possui complexidade de tempo o(n log n)?

Desempenho Computacional

Voltemos ao nosso exemplo: $A = \{0^k 1^k \mid k \geq 0\}$

- Será que é possível construir uma MT de duas fitas que possui complexidade de tempo o(n log n)?
- Considere a magnifica MT de duas fitas M_3:

M_3 = "Sobre a cadeia de entrada *w*:

1. Faça uma varredura na fita e *rejeite* se algum 0 for encontrado à direita de algum 1.
2. Faça uma varredura nos 0s sobre a fita 1 até o primeiro 1. Ao mesmo tempo, copie os 0s para a fita 2.
3. Faça uma varredura nos 1s sobre a fita 1 até o final da entrada. Para cada 1 lido sobre a fita 1, corte um 0 sobre a fita 2. Se todos os 0s estiverem cortados antes que todos os 1s sejam lidos, *rejeite*.
4. Se todos os 0s tiverem sido cortados, *aceite*. Se restar algum 0, *rejeite*.

Desempenho Computacional

Voltemos ao nosso exemplo: $A = \{0^k 1^k \mid k \geq 0\}$

- Será que é possível construir uma MT de duas fitas que possui complexidade de tempo o(n log n)?
- Considere a magnifica MT de duas fitas M_3:

M_3 = "So

1. F à direita d
2. Faça uma varredura nos 0s sobre a fita 1 até o primeiro 1. Ao mesmo tempo, copie os 0s para a fita 2.
3. Faça uma varredura nos 1s sobre a fita 1 até o final da entrada. Para cada 1 lido sobre a fita 1, corte um 0 sobre a fita 2. Se todos os 0s estiverem cortados antes que todos os 1s sejam lidos, *rejeite*.
4. Se todos os 0s tiverem sido cortados, *aceite*. Se restar algum 0, *rejeite*.

Qual a **complexidade de tempo** da MT M_3?

Desempenho Computacional

- A MT de duas fitas M_3 possui complexidade de tempo: **O(n)**
- Mas será que $M_3 = O(n)$ ou $M_3 = o(n)$??

Desempenho Computacional

- A MT de duas fitas M_3 possui complexidade de tempo: **O(n)**
- Mas será que $M_3 = O(n)$ ou $M_3 = o(n)$
- É necessário *n* operações no mínimo para ler a entrada, então não tem como ser melhor do que isso!

Resumo da complexidade de tempo sobre A:

MT com uma fita: O(n log n)

MT com duas fitas: O(n)

A complexidade de A depende do modelo computacional utilizado!

Relacionamento de Complexidade

Teorema: seja $t(n)$ uma função onde $t(n) \geq n$. Então toda MT multifita de tempo $t(n)$ tem uma MT de uma fita equivalente de tempo $O(t^2(n))$.

Ideia da prova: lembrando-se que é possível converter qualquer MT multifita em uma MT de uma única fita que a simula, é possível perceber que simular cada passo da máquina multifita usa, no máximo, $O(t(n))$ passos na máquina de uma única fita. Logo, o tempo total usado é $O(t^2(n))$

Relacionamento de Complexidade

Teorema: seja $t(n)$ uma função onde $t(n) \geq n$. Então para toda MT não-determinística de uma fita de tempo $t(n)$, existe uma MT determinística de uma fita equivalente de tempo $2^{O(t(n))}$.

Ideia da prova: lembrando-se que é possível converter qualquer MT não-determinística em uma MT determinística de uma única fita que a simula, é possível perceber que é preciso explorar todo e cada ramificação da árvore de possibilidades gerada por uma MT não-determinística: $O(b^{t(n)})$ o que nos leva a uma complexidade de tempo de simulação de: $2^{O(t(n))}$.

Relacionamento de Complexidade

Teorema: seja $t(n)$ uma função onde $t(n) \geq n$. Então para toda MT não-determinística de uma fita de tempo $t(n)$, existe uma MT determinística de uma fita equivalente de tempo $2^{O(t(n))}$.

Ideia da prova: lembrando-se que é possível converter qualquer M... ca fi... e ca... T nã... le de tempo de simulação de: $2^{O(t(n))}$.

A definição do tempo de execução de uma MT não-determinística não tem o objetivo de corresponder a algum dispositivo de computação do mundo real. Ela é uma definição matemática útil que ajuda na caracterização da complexidade de uma classe importante de problemas computacionais!