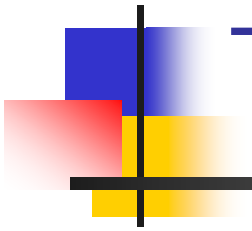
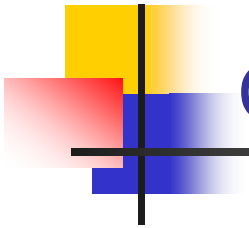


Teoria da Computação

Computabilidade e complexidade computacional



Computabilidade e Complexidade

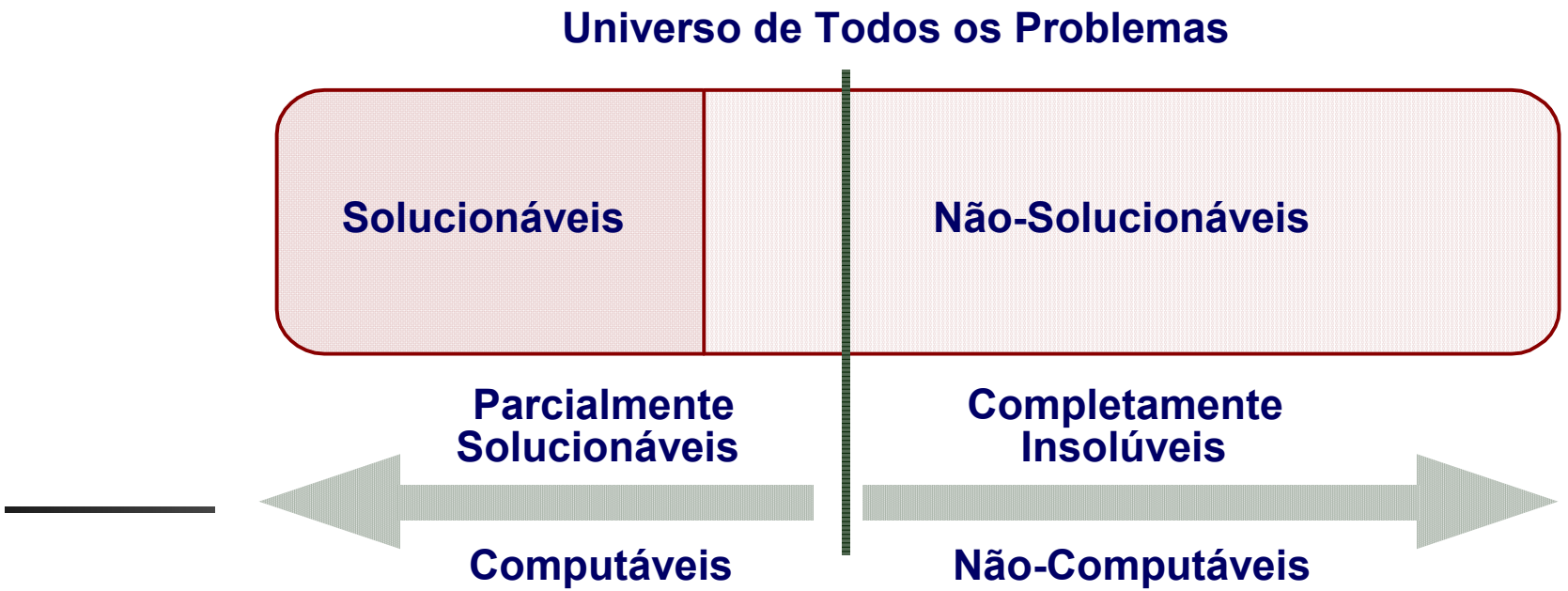
- **Computabilidade**: verifica a existência de algoritmos que resolva uma classe de linguagens – trata a possibilidade da sua construção

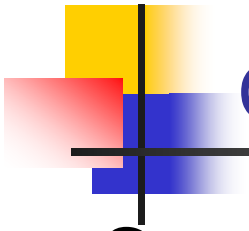
  - Dado um problema
    - se existe uma MT capaz de resolvê-lo logo o computador também resolverá - **computável**
    - caso contrário o problema é dito **incomputável** (insolúvel) – MT não é capaz de resolvê-lo

Computabilidade - Visão geral

Quais problemas os computadores conseguem efetivamente resolver? Como isso pode ser verificado?

Concentra-se nos problemas com respostas binárias (problemas sim/não ou problemas de decisão).





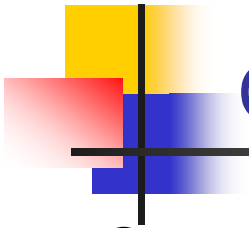
Computabilidade e Complexidade

- **Computabilidade**: verifica a existência de algoritmos que resolva uma classe de linguagens – trata a possibilidade da sua construção

- **Complexidade**: trata da eficiência da computação (dos algoritmos) em computadores existentes

  - **Complexidade temporal: tempo de processamento exigido**

  - Complexidade espacial: espaço de armazenamento exigido

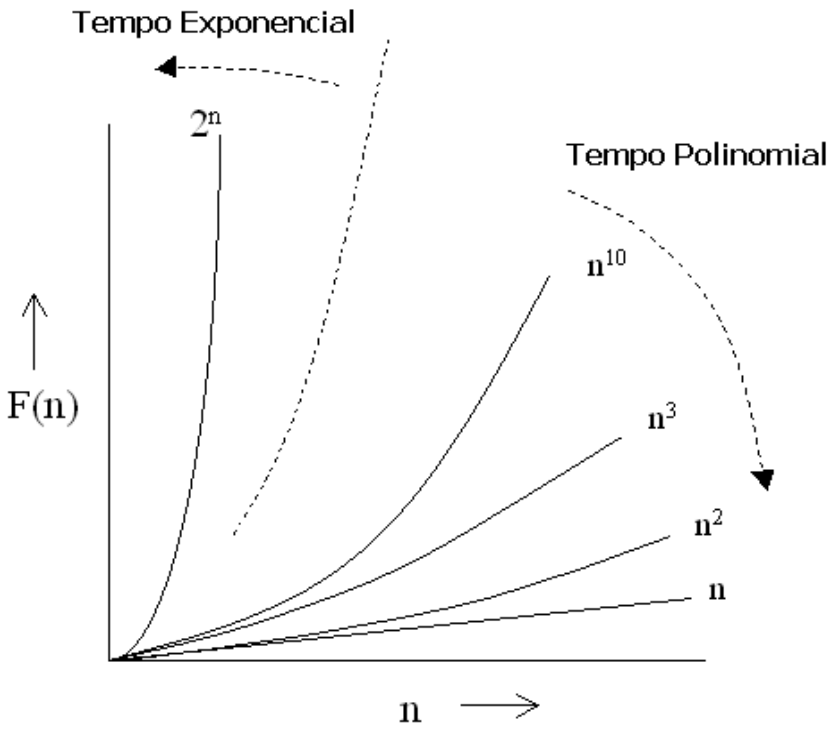


Complexidade computacional

- O conjunto das linguagens decidíveis pode ser dividida em classes de complexidade, que caracterizam os limites dos recursos computacionais usados para as decidir.

- Uma **classe de complexidade** é especificada por um modelo de computação – por exemplo, uma MT com computação determinística ou não determinística. Questões a serem consideradas:
  - recursos: tempo ou espaço

Universo de problemas

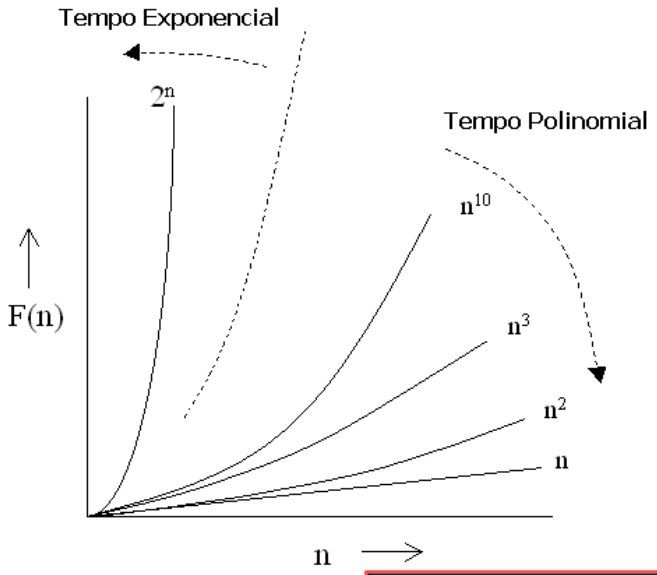


Apresenta uma solução polinomial

Problemas tratáveis

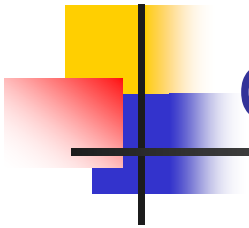
Problemas intratáveis

nenhum algoritmo de tempo polinomial pode resolvê-lo



Crescimento de funções de complexidade

Função de complexidade	Tamanho da Instância do Problema					
	10	20	30	40	50	60
n	0,00001 segundos	0,00002 segundos	0,00003 segundos	0,00004 segundos	0,00005 segundos	0,00006 segundos
n^2	0,0001 segundos	0,0004 segundos	0,0009 segundos	0,0016 segundos	0,0025 segundos	0,0036 segundos
n^3	0,001 segundos	0,008 segundos	0,027 segundos	0,064 segundos	0,125 segundos	0,216 segundos
n^5	0,1 segundos	3,2 segundos	24,3 segundos	1,7 minutos	5,2 minutos	13,0 minutos
2^n	0,001 segundos	1,0 segundos	17,9 segundos	12,7 dias	35,7 anos	366 séculos
3^n	0,059 segundos	58 minutos	6,5 anos	3855 séculos	2×10^8 séculos	$1,3 \times 10^{13}$ séculos



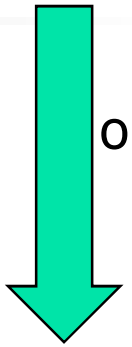
Complexidade computacional

- Medir o tempo gasto por um algoritmo
  - Não é uma boa opção
  - Depende do compilador
    - Pode preferir algumas construções ou otimizar melhor
  - Depende do hardware
    - GPU vs. CPU, desktop vs. smartphone

- **Estudar o número de vezes que operações são executadas**

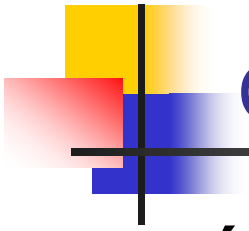
Complexidade computacional – exemplo de tempo de processamento

- Exemplo: achar o maior valor do vetor



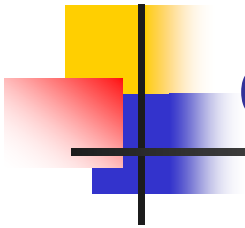

```
int vmax(int *vec, int n) {          -
        int i;                       -
        int max = vec[0];            1
        for(i = 1; i < n; i++) {     n-1
                if(vec[i] > max) {   n-1
                        max = vec[i];  A < n-1
                }                    n-1
        }                            n-1
        return max;                  1
}
```

- Complexidade (total de operações): f(n)=n-1



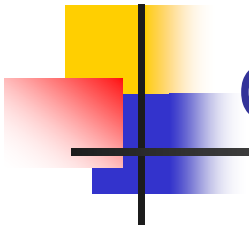
Complexidade computacional

- Análise de complexidade feita em função de n
  - n indica o tamanho da entrada
    - Número de elementos no vetor
    - Número de vértices num grafo
    - Número de linhas de uma matriz



Complexidade computacional

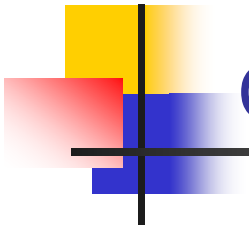
- O parâmetro *n*
  - provê uma medida do tamanho do problema no sentido de que o tempo requerido para solucioná-lo, ou o espaço de armazenamento necessário, ou ambos, serão incrementados conforme n aumenta.
  - A ordem de magnitude dará exatamente a proporção em que ocorre esse incremento. Na verdade, O( ) nos dá o limite superior do recurso (tempo ou espaço) necessário para o algoritmo.



Complexidade computacional

- Uma **classe de complexidade** é um conjunto C(*f(n)*) definido por:

C(f(n)) = {L | L é decidida por uma MT M de modo adequado, tal que para qualquer x, e |x| = n, M gasta no máximo f(|x|) unidades do respectivo recurso} ➔ ***tempo ou espaço***

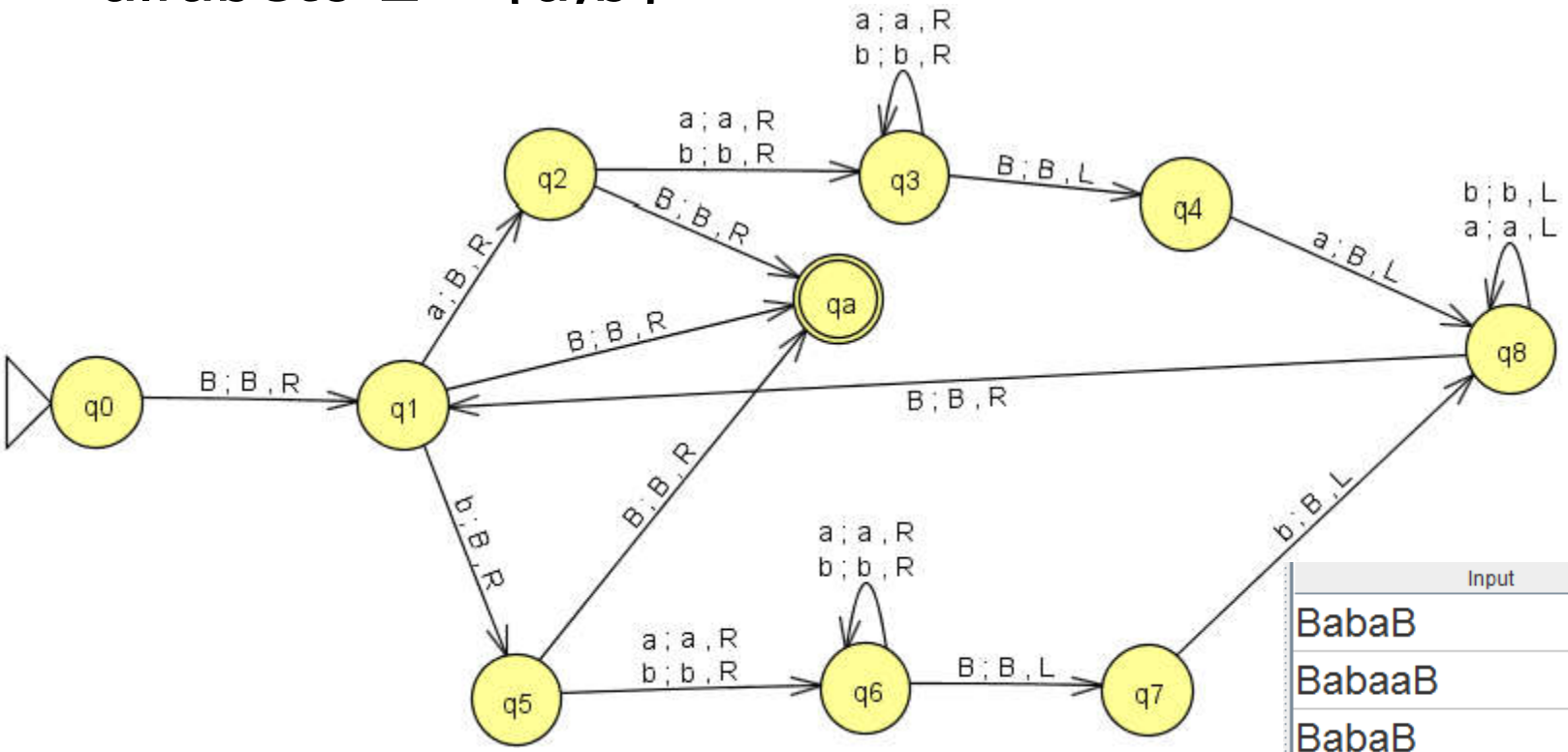


Complexidade computacional

- No caso de funções de complexidade temporal deve ainda ser tal que exista uma MT M de k-fitas que, para quaisquer dados **x**, pare em exatamente *f(|x|)* passos. Esta característica permite a simulação de "relógios" em MT's – quantidade de transições percorridas até a MT parar
  - Restrições semelhantes se impõem para funções de complexidade de espaço.

Complexidade computacional em uma MT

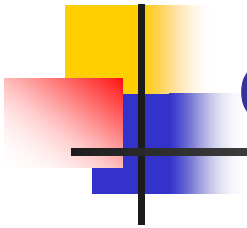
- Exemplo de MT M1= aceita palíndromos sobre o alfabeto $\Sigma = \{a,b\}$



Input	
BabaB	Accept
BabaaB	Reject
BabaB	Accept
BaB	Accept
BaB	Accept

Complexidade computacional em uma MT

$\lvert w\rvert = 0$	$\lvert w\rvert = 1$	$\lvert w\rvert = 2$		$\lvert w\rvert = 3$	
q_0BB	q_0BaB	q_0BaaB	q_0BabB	q_0BabaB	q_0BaabB
Bq_1B	Bq_1aB	Bq_1aaB	Bq_1abB	Bq_1abaB	Bq_1aabB
BBq_a	BBq_2B	BBq_2aB	BBq_2bB	BBq_2baB	BBq_2abB
	$BBBq_a$	$BBaq_3B$	$BBbq_3B$	$BBbq_3aB$	$BBaq_3bB$
		BBq_4aB	BBq_4bB	$BBbaq_3B$	$BBabq_3B$
		Bq_8BBB		$BBbq_4aB$	$BBaq_4bB$
		BBq_1BB		BBq_8bBB	
		$BBBq_aB$		Bq_8BbBB	
				BBq_1bBB	
				$BBBq_5BB$	
				$BBBBq_aB$	



Complexidade computacional em uma MT

- O número de transições em uma computação depende da cadeia de entrada → quantidade de trabalho difere para cadeias de mesmo comprimento.

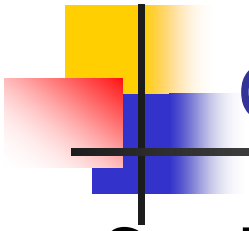
- Ao invés de tentar determinar o número exato de transições para cada cadeia de entrada, a complexidade de tempo de uma MT é medida pelo trabalho necessário para cadeias de um comprimento fixo.

Complexidade computacional em uma MT

- **Definição**: Seja M uma MT. A complexidade de tempo (tempo gasto para execução) de M é dada pela função

$$ct_M: \mathbb{N} \rightarrow \mathbb{N}$$

tal que $ct_M(n)$ é o número máximo de transições processadas por uma computação de M quando iniciada com uma cadeia de entrada de comprimento n, independente de M aceitar ou não.



Complexidade computacional em uma MT

- Considerações:
  - Quando se avalia a complexidade de tempo de uma máquina de Turing, assume-se que a computação termina para toda cadeia de entrada – *Linguagens Decidíveis*.
  - Não faz sentido discutir a eficiência de uma computação que permanece em loop (problema de decisão sem resposta).
  - A complexidade de tempo dá a performance de pior caso da MT.

Complexidade computacional em uma MT

- A máquina M1 que aceita os palíndromos sobre o alfabeto Σ={a,b} é usada para demonstrar o processo de determinar a complexidade de tempo de uma MT.

- Uma computação de M1 termina quando toda a cadeia de entrada foi substituída por brancos ou o primeiro par de símbolos não previsto é descoberto.

- Como a complexidade de tempo mede a performance do pior caso, há necessidade de se preocupar apenas com as cadeias cujas computações fazem com que a máquina faça o maior número possível de ciclos **acha-e-apaga** (substitui por Branco)

- Esta condição é satisfeita quando a entrada é aceita por M1.

Complexidade computacional em uma MT

- Usando estas observações, pode-se obter os valores iniciais da função ct_{M1} das computações da tabela.

$ct_{M1}(0) = 2$

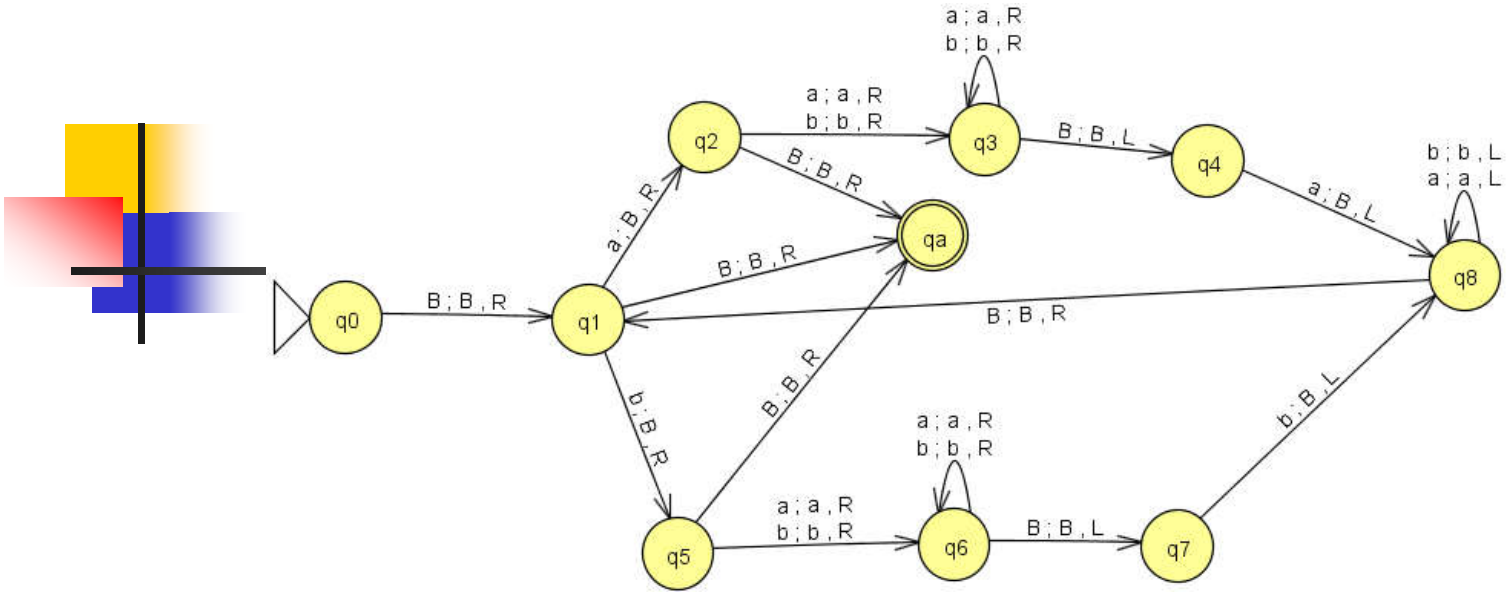
$ct_{M1}(1) = 3$

$ct_{M1}(2) = 7$

$ct_{M1}(3) = 10$

- Quando M processa uma cadeia de comprimento par, a computação alterna entre sequências de movimentos a direita e a esquerda da máquina.

$\|w\| = 0$	$\|w\| = 1$	$\|w\| = 2$		$\|w\| = 3$	
q_0BB	q_0BaB	q_0BaaB	q_0BabB	q_0BabaB	q_0BaabB
Bq_1B	Bq_1aB	Bq_1aaB	Bq_1abB	Bq_1abaB	Bq_1aabB
BBq_a	BBq_2B	BBq_2aB	BBq_2bB	BBq_2baB	BBq_2abB
	$BBBq_a$	$BBaq_3B$	$BBbq_3B$	$BBbq_3aB$	$BBaq_3bB$
		BBq_4aB	BBq_4bB	$BBbaq_3B$	$BBabq_3B$
		Bq_8BBB		$BBbq_4aB$	$BBaq_4bB$
		BBq_1BB		BBq_8bBB	
		$BBBq_aB$		Bq_8BbBB	
				BBq_1bBB	
				$BBBq_5BB$	
				$BBBBq_aB$	



- Inicialmente a cabeça é posicionada na extremidade esquerda da fita:
  - **Movimentos à direita**: a máquina se move à direita, apagando (substitui por B) o símbolo não-branco mais à esquerda. O resto da cadeia é lida e a máquina entra no estado q4 ou q7. Isto requer k +1 transições, onde k é o comprimento da porção não-branca da cadeia.
  - **Movimentos à esquerda**: M se move para a esquerda, apagando o símbolo correspondente, e continua através da porção não-branca da fita. Isto requer k transições.
- As ações acima reduzem o comprimento da porção não-branca da fita por dois.
- O ciclo de comparações e substituições (por B) é repetido até que a fita esteja completamente em branco.

Complexidade computacional em uma MT

- Como observado, a performance do pior caso para uma cadeia de comprimento par ocorre quando M1 aceita a entrada.
- A computação que aceita uma cadeia de comprimento n requer n/2 iterações do loop anterior.

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

Complexidade computacional em uma MT

- O número máximo de transições em uma computação de uma cadeia de comprimento par n é a soma dos primeiros **n+1** números naturais.

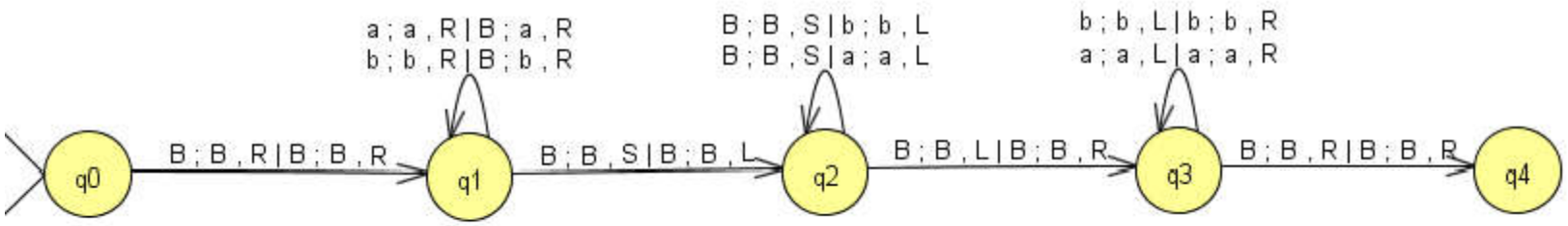
- Uma análise de cadeias de comprimento ímpar produz o mesmo resultado.

- Consequentemente, a complexidade de tempo de M1 é dada pela função:

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

Complexidade computacional em uma MT

- Exemplo 2: MT2 com 2 fitas que aceita a linguagem dos palíndromos sobre o alfabeto Σ ={a,b}



- Uma computação de M2 percorre a entrada fazendo uma cópia na fita 2.
- A cabeça da fita 2 é depois movida à posição inicial.
- Então, as cabeças se movem ao longo da entrada, fita 1 para a esquerda e fita 2 para a direita, comparando os símbolos das fitas 1 e 2.

Complexidade computacional em uma MT

- Se as cabeças encontrarem símbolos diferentes, a entrada **não** é um palíndromo e a computação para em um estado de não-aceitação.

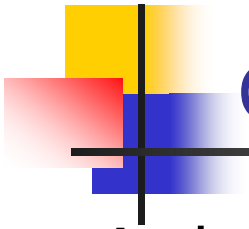
- Quando a entrada é um palíndromo, a computação para e aceita quando brancos são simultaneamente lidos nas fitas 1 e 2.

- Para uma entrada de comprimento n, o número máximo de transições ocorre quando a cadeia é um palíndromo (**pior caso**).

- Uma aceitação requer três passos completos:
  1. **a cópia,**
  2. **a volta e**
  3. **a comparação.**

- Contando o número de transições em cada passo, vê-se que a complexidade de tempo de M2 é

  $$ct_{M2}(n) = 3(n + 1) + 1.$$



Complexidade computacional em uma MT

- A definição da complexidade de tempo ct_M é baseada nas computações da máquina M e não na linguagem aceita pela máquina.

- Diferentes máquinas podem ser construídas para aceitar a mesma linguagem, cada qual com diferentes complexidades de tempo.

- Diz-se que uma linguagem L é aceita em tempo determinístico *f(n)* se houver uma MT determinística M qualquer com $ct_M(n) \leq f(n)$ para todo $n \in \mathbb{N}$.

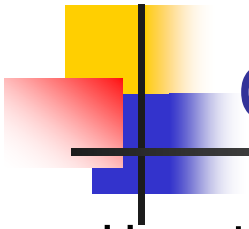
Complexidade computacional em uma MT

- A máquina M1 mostrou que o conjunto de palíndromos sobre Σ ={a,b} é aceito em tempo **$(n^2+3n+2)/2$**

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

- Enquanto que a máquina de duas fitas M2 exibiu aceitação no tempo **3n + 4**.

$$ct_{M2}(n) = 3(n + 1) + 1.$$



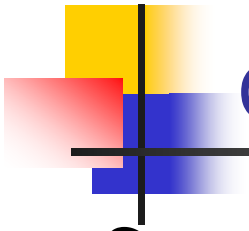
Complexidade computacional em uma MT

- Uma transição de uma máquina de duas fitas utiliza mais informação e realiza uma operação mais complicada do que a máquina de uma fita.

- A estrutura adicional da transição de duas fitas permitiu a redução no número de transições de M2 necessárias para processar uma cadeia em relação à máquina de uma fita M1.

- Portanto, vê-se um equilíbrio entre a complexidade das transições e o número que deve ser processado.

- Há formas de "acelerar" uma máquina que aceita uma linguagem L para produzir uma nova máquina que aceita L em um tempo menor.



Complexidade computacional em uma MT

- Como sempre é possível "melhorar" uma máquina para que aceite a linguagem num tempo menor, é interessante representar a complexidade de tempo por uma **taxa de crescimento** ao invés de uma função.

Taxa de crescimento

- A taxa de crescimento de uma função mede o aumento dos valores da função quando a entrada se torna arbitrariamente grande.

- Intuitivamente, a taxa de crescimento é determinada pelo contribuinte mais significativo ao crescimento da função.

- A contribuição de termos individuais aos valores de uma função pode ser vista examinando o crescimento das funções **n^2** e **$n^2 + 2n + 5$**.

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
$\frac{n^2}{(n^2+2n+5)}$	0	0,625	0,800	0,919	0,960	0,980	0,998

Taxa de crescimento

- A contribuição de $\mathbf{n^2}$ à $\mathbf{n^2 + 2n + 5}$ é medida pela razão dos valores da função na última linha.

- Os termos linear e constante da função $\mathbf{n^2 + 2n + 5}$ são chamados de termos de ordem mais baixa.

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
$\frac{n^2}{(n^2+2n+5)}$	0	0,625	0,800	0,919	0,960	0,980	0,998

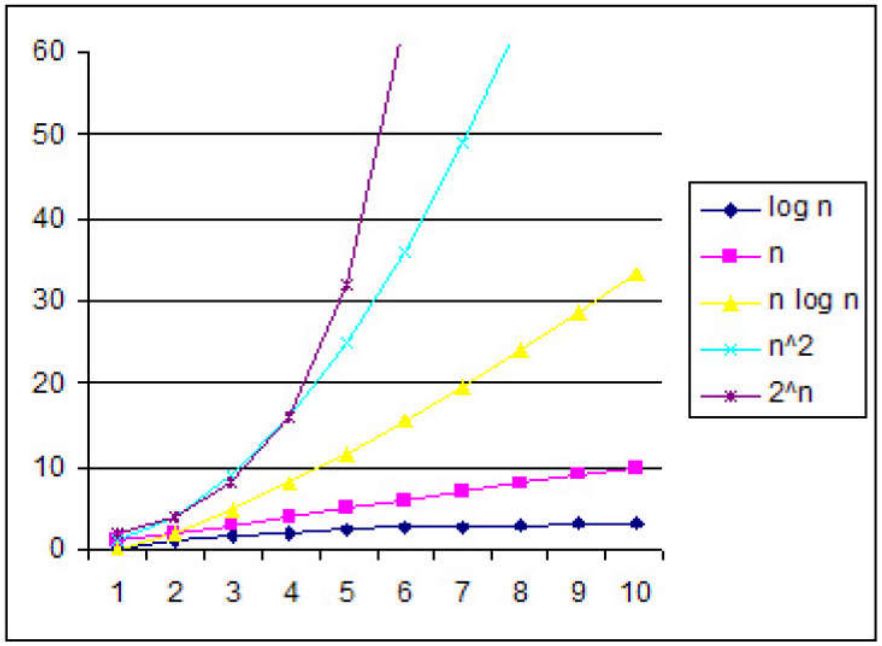
Considerações sobre eficiência

- Estes termos influenciam os valores iniciais da função, mas à medida em que n aumenta, estes termos não contribuem significativamente para o crescimento dos valores da função.

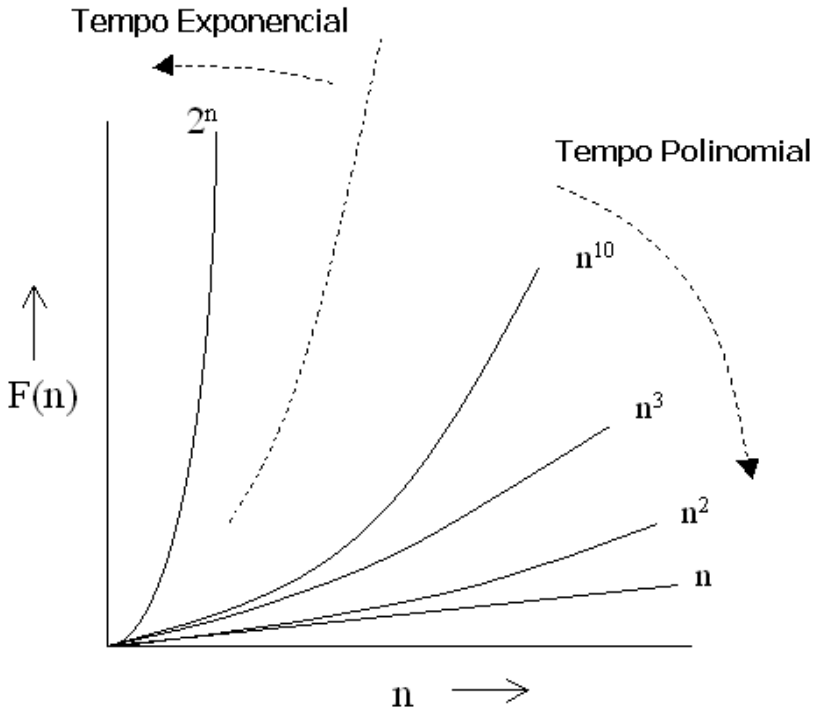
- Para n pequeno (até 25), o termo 20n + 500 produz um resultado maior que n^2, pois a diferença entre n^2 e n é pequena. A medida em que n cresce, o termo n^2 domina o termo 20n. Dessa forma, para n grande, o tempo necessário para a aceitação é quase proporcional à n^2.

- Para essa tendência de uma função tornar-se proporcional a outra à medida em que aumenta, é dito ser "da ordem" da função.

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
$\frac{n^2}{(n^2+2n+5)}$	0	0,625	0,800	0,919	0,960	0,980	0,998



Considerações sobre eficiência



Hierarquia da notação *O*

- $\mathcal{O}(1)$: ordem constante
- $\mathcal{O}(log_a\ n)$: ordem logarítmica
- $\mathcal{O}(n)$: ordem linear
- $\mathcal{O}(n\ log_a\ n)$: *n log n*
- $\mathcal{O}(n^2)$: ordem quadrática
- $\mathcal{O}(n^3)$: ordem cúbica
- $\mathcal{O}(n^r)$: ordem polinomial $r \geq 0$
- $\mathcal{O}(b^n)$: ordem exponencial $b > 1$
- $\mathcal{O}(n!)$: ordem fatorial.

Considerações sobre eficiência: notação *O*

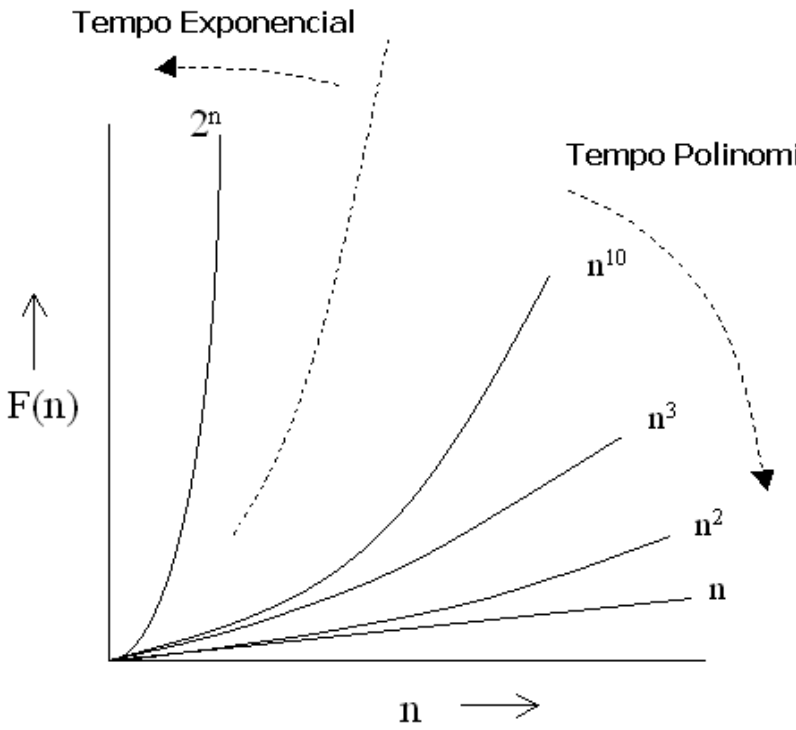
- **Algoritmos exponenciais** são, em geral, simples: variações de pesquisa exaustiva no espaço de soluções (força bruta).

- **Algoritmos polinomiais** são obtidos através de um entendimento mais profundo da estrutura do problema.

- Um problema é considerado **intratável** se não existe um algoritmo polinomial para resolvê-lo.

- Um problema é considerado bem resolvido/**tratável** se existe um algoritmo polinomial para o problema. Tais problemas são considerados eficientes.

Considerações sobre eficiência

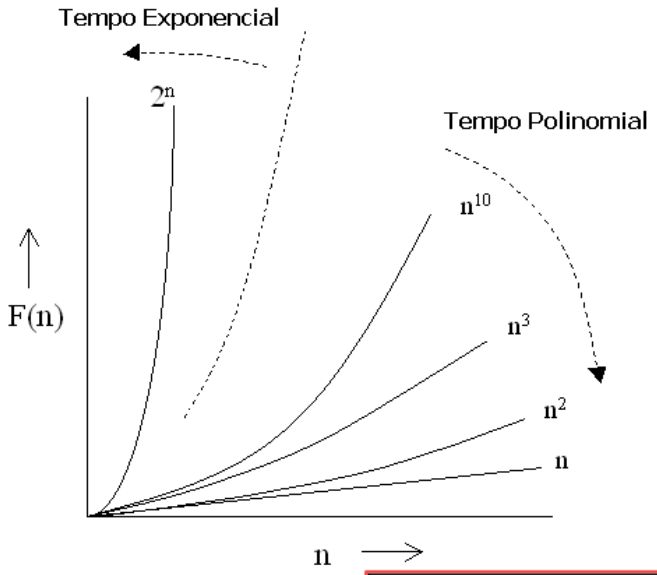


Apresenta uma solução polinomial

Problemas tratáveis

Problemas intratáveis

nenhum algoritmo de tempo polinomial pode resolvê-lo

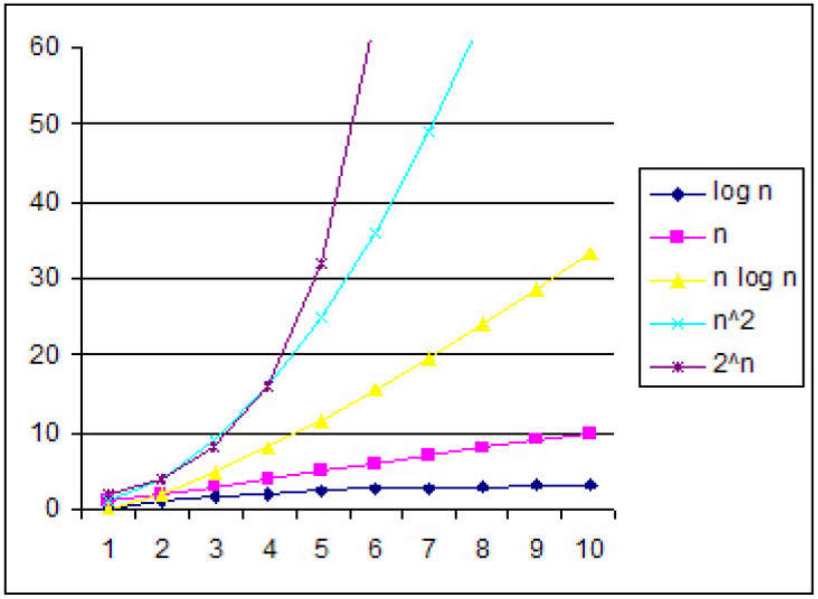


Crescimento de funções de complexidade

Função de complexidade	Tamanho da Instância do Problema					
	10	20	30	40	50	60
n	0,00001 segundos	0,00002 segundos	0,00003 segundos	0,00004 segundos	0,00005 segundos	0,00006 segundos
n^2	0,0001 segundos	0,0004 segundos	0,0009 segundos	0,0016 segundos	0,0025 segundos	0,0036 segundos
n^3	0,001 segundos	0,008 segundos	0,027 segundos	0,064 segundos	0,125 segundos	0,216 segundos
n^5	0,1 segundos	3,2 segundos	24,3 segundos	1,7 minutos	5,2 minutos	13,0 minutos
2^n	0,001 segundos	1,0 segundos	17,9 segundos	12,7 dias	35,7 anos	366 séculos
3^n	0,059 segundos	58 minutos	6,5 anos	3855 séculos	2×10^8 séculos	$1,3 \times 10^{13}$ séculos

Algoritmos polinomais e não-polinomiais

- Número de transições de máquina com complexidade de tempo ct_M com entrada de comprimento n.



n	ct_M					
	$log_2(n)$	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	$1{,}6 \cdot 10^{60}$	$> 10^{374}$

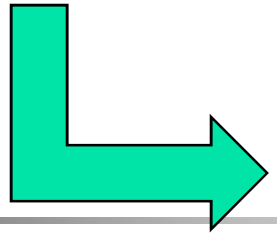
Computabilidade e Complexidade

- **Computabilidade**: verifica a existência de algoritmos que resolva uma classe de linguagens – trata a possibilidade da sua construção

- **Complexidade**: trata da eficiência da computação (dos algoritmos) em computadores existentes

  - Complexidade temporal: tempo de processamento exigido

  - **Complexidade espacial: espaço de armazenamento exigido**

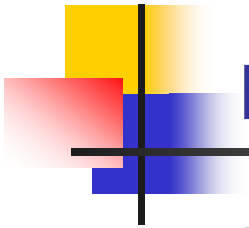


Custo computacional

Referências bibliográficas

- Rosa, João Luis. Teoria da Computação e Linguagens Formais. Notas de Aula. Ciências de Computação. Instituto de Ciências Matemáticas e de Computação. Universidade de São Paulo, 2007.

- Hopcroft, John E.,1939. Introdução à teoria de autômatos, linguagens e computação. Rio de Janeiro : Campus, 2003.



Exercícios

1. Defina e exemplifique a análise de complexidade por tempo.
2. Defina e exemplifique os seguintes termos: a) Ordem de complexidade de melhor caso. b) Ordem de complexidade de caso médio. c) Ordem de complexidade de pior caso.
3. Construa uma MT que decide a linguagem A={w | w contém o mesmo número de 0's e 1's} e dê a complexidade de tempo.