Capítulo V : Máquinas de Turing e Complexidade

⇨ Ciências da Computação

*"As **Ciências da Computação** são o estudo dos **Algoritmos**"*
Donald Knuth

⇨ Algoritmo

{ Afinal o que é um Algoritmo? }

Existem registos de Algoritmos desde o tempo da antiga Babilónia.
*O termo Algoritmo tem origem no nome de **Al'Khwarizmi** (séc. IX).*

*Só no início do século XX começou a ser estudada a sua **noção formal** e posteriormente o seu desenvolvimento se tornou vital no **desenvolvimento e utilização** dos computadores.*

⇨ Teoria da Computação

- *Capacidades e Limitações dos Algoritmos / da Computação (Computabilidade e Decidibilidade).*
- *Eficiência dos Algoritmos (Análise de Complexidade).*
- *Correcção dos Algoritmos (Verificação Formal).*
- *Modelo Matemático para o Processamento de Linguagens (Linguagens Formais e Autómatos).*
- *Linguagens de Programação (Sintaxe e Semântica).*
- *Modelos de Dados (Teorias dos Tipos Abstractos de Dados).*
- *Paradigmas de Programação (Programação Imperativa, Funcional, em Lógica, Orientada aos Objectos, ...).*
- *...*

⇨ Modelos Formais para a noção de Algoritmo

{ Formalismos estruturalmente distintos, apesar de "equivalentes" }

- *Máquina de Turing (Alan Mathison Turing)*
- *Funções Recorrentes (Stephen Cole Kleene)*
- *Funções μ-recorrentes (Kurt Gödel)*
- *Cálculo-λ (Alonzo Church)*
- *Sistema de Post (Emil Leon Post)*
- *Lógica Combinatória (Moses Schönfinkel)*

... todos cerca de 1936.

⇨ A Máquina de Turing como Modelo de Computação

*{ Para Alan Turing, "**Computador**" era*
*a **pessoa** que efectuava as "computações". }*

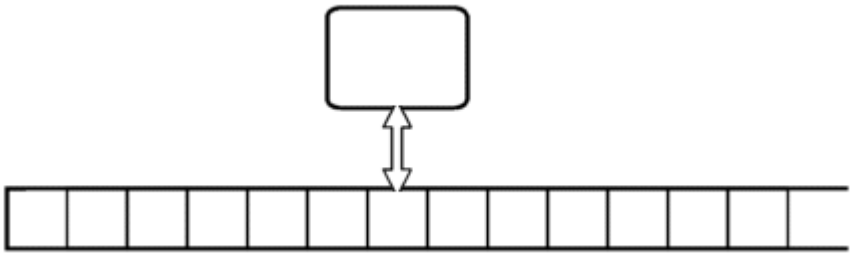
A **Tese de Church – Turing** (versão intuitiva):

Algoritmo = Máquina de Turing

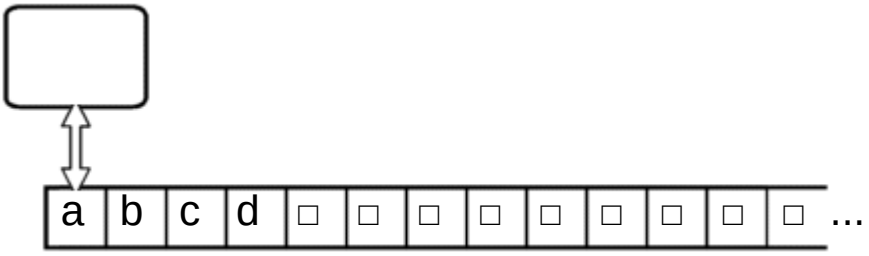
{ Um Problema resolúvel por um Computador
é resolúvel por uma Máquina de Turing }

• A Máquina de Turing

Na sua versão básica, uma Máquina de Turing é semelhante a um Autómato de Pilha. Tem uma Unidade de Controlo dos **Estados,** com uma cabeça de leitura/escrita. Contudo, o armazenamento é feito numa **Fita** "semi-infinita", onde é possível **ler e escrever símbolos**.



A **cadeia de entrada** começa por ser colocada na **Fita**, o espaço restante é preenchido por símbolos **brancos** e a cabeça de leitura/escrita é colocada no início.



Com base no **Estado** actual e no **símbolo** lido, um **passo** da Máquina de Turing consiste em:

1. **Transitar** para outro Estado;
2. **Escrever** um símbolo na Fita;
3. **Mover** a cabeça de leitura (**L** esquerda, **R** direita).

Definição: Uma **Máquina de Turing (MT)** é um séptuplo ordenado,

$$M = (Q, \Sigma, \Gamma, \delta, q_0, \square, F)$$

onde:

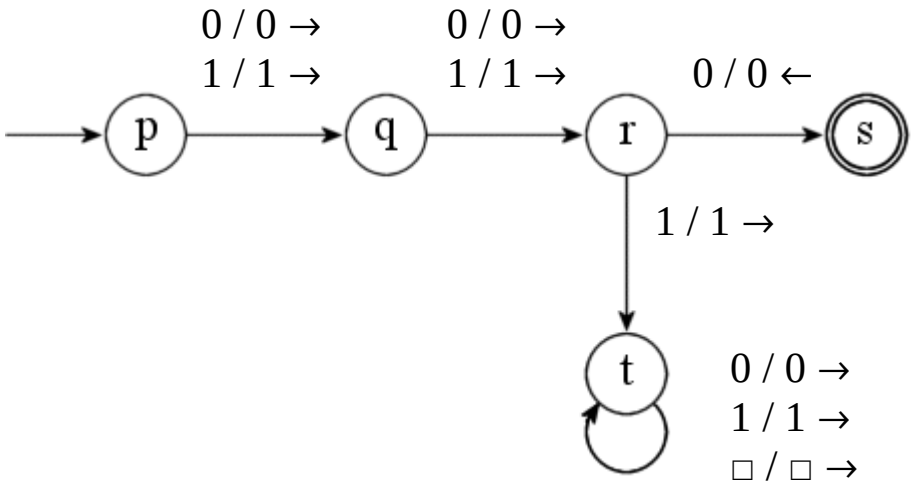
- Q é um conjunto finito de **estados**,
- Σ é o conjunto finito de **símbolos de entrada**,
- Γ é o conjunto finito de **símbolos da Fita**, onde $\Sigma \subseteq \Gamma$,
- $\delta : Q \times \Gamma \longrightarrow Q \times \Gamma \times \{L, R\}$ é a função de **transição**,
- $q_0 \in Q$ é o **estado inicial**,
- $\square \in \Gamma \setminus \Sigma$ é o **símbolo branco**,
- $F \subseteq Q$ é o conjunto dos **estados finais** ou **de aceitação**.

Exemplo: A MT seguinte aceita as palavras $w \in \{0, 1\}^*$ cujo terceiro símbolo é um 0. No caso contrário, corre indefinidamente.

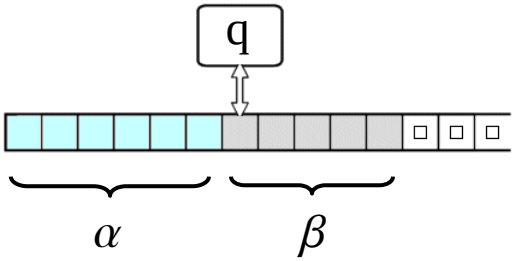
$$M = (\{p, q, r, s, t\}, \{0, 1\}, \{0, 1, \square\}, \delta, p, \square, \{s\})$$

com:

	δ	0	1	□
$\longrightarrow$	p	(q, 0, R)	(q, 1, R)	—
	q	(r, 0, R)	(r, 1, R)	—
	r	(s, 0, L)	(t, 1, R)	—
*	s	—	—	—
	t	(t, 0, R)	(t, 1, R)	(t, □, R)



A **descrição instantânea** de uma Máquina de Turing denota-se por $\alpha \mathbf{q} \beta$ com q ∈ Q e $\alpha, \beta \in \Gamma^*$, onde:



q é o presente Estado, α a sequência de símbolos à esquerda da cabeça de leitura e β a sequência dos restantes símbolos, não brancos.
Um **passo** da Máquina de Turing é representado por **⊢ .**

No caso do exemplo anterior, para w = 0101, a sequência de passos seria:

p0101 ⊢ 0q101 ⊢ 01r01 ⊢ 0s101

Nesse instante a MT **pára**, porque <u>não existem</u> transições a partir de <u>s</u>.

Para w = 0111 a sequência seria:

p0111 ⊢ 0q111 ⊢ 01r11 ⊢ 011t1 ⊢ 0111t ⊢ 0111□t ⊢ 0111□□t ⊢ ...

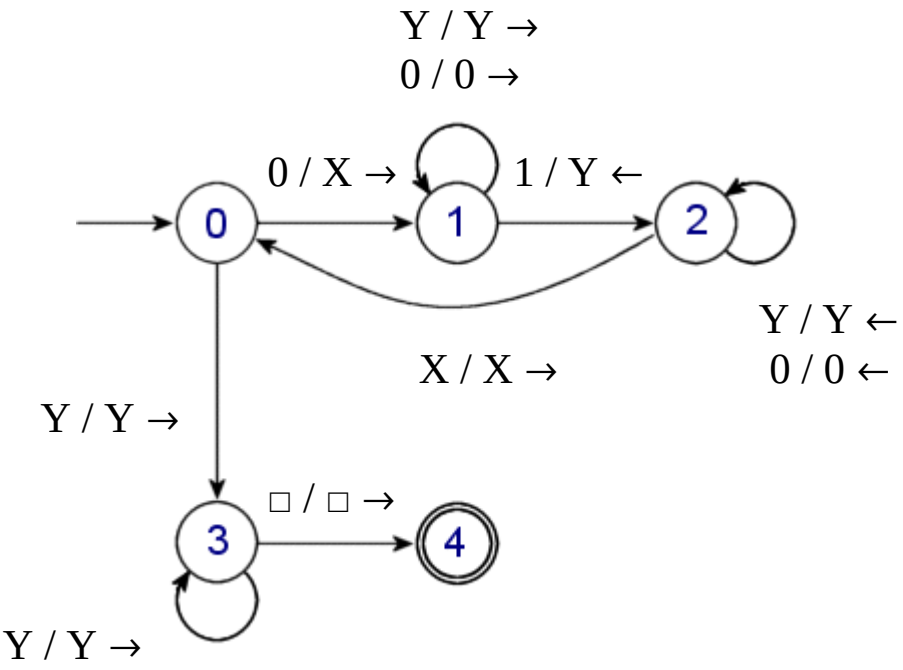
e a MT **continuaria indefinidamente** para a direita ...

Exemplo: Uma MT para reconhecer a Linguagem $\{ 0^n1^n \mid n \geq 1 \}$

$M = (\{q_0, q_1, q_2, q_3, q_4\}, \{0, 1\}, \{0, 1, X, Y, \square\}, \delta, q_0, \square, \{q_4\})$

com:

	δ	0	1	X	Y	□
⟶	q_0	(q_1, X, R)	—	—	(q_3, Y, R)	—
	q_1	$(q_1, 0, R)$	(q_2, Y, L)	—	(q_1, Y, R)	—
	q_2	$(q_2, 0, L)$	—	(q_0, X, R)	(q_2, Y, L)	—
	q_3	—	—	—	(q_3, Y, R)	$(q_4, \square, R)$
*	q_4	—	—	—	—	—



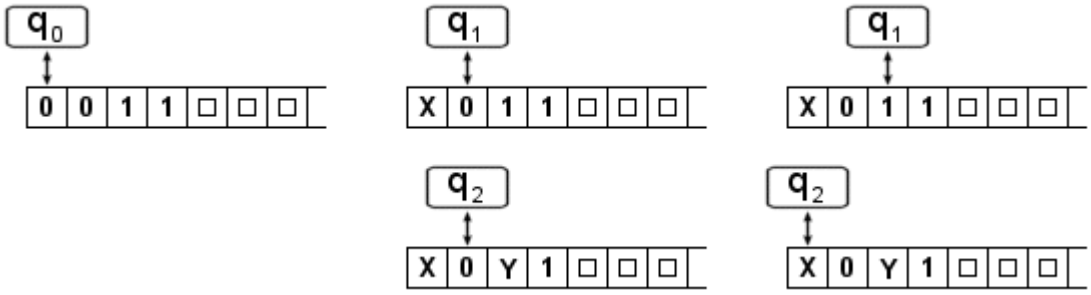
Para w = 0011 seria executada a sequência de passos:

$q_00011 \vdash Xq_1011 \vdash X0q_111 \vdash Xq_20Y1 \vdash q_2X0Y1 \vdash$
$Xq_00Y1 \vdash XXq_1Y1 \vdash XXYq_11 \vdash XXq_2YY \vdash Xq_2XYY \vdash$
$XXq_0YY \vdash XXYq_3Y \vdash XXYYq_3\square \vdash XXYY\square q_4\square$

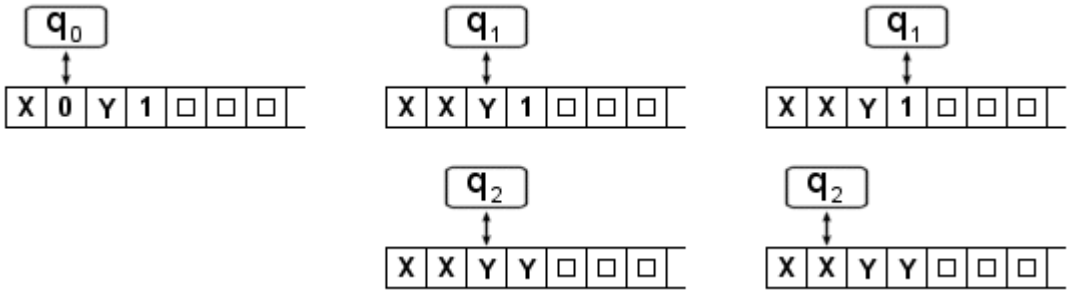
O Algoritmo utilizado vai substituindo, alternadamente, cada 0 por X e cada 1 por Y, verificando a concordância.

Para cada par (0, 1) é executado o ciclo:

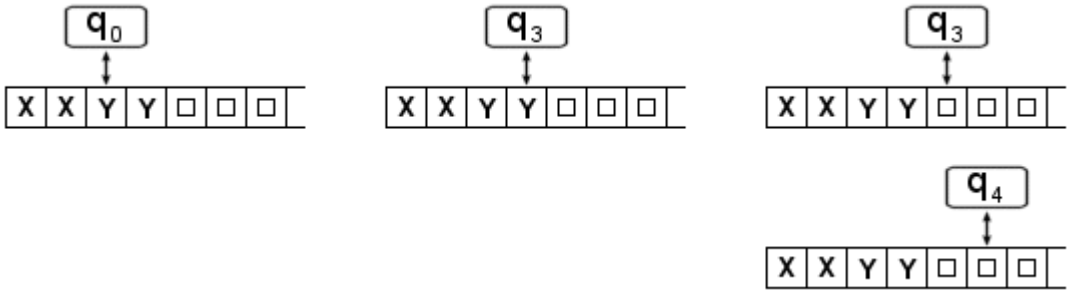
(q_0) substituir o primeiro 0 por X;
(q_1) (saltando 0's e Y's) avançar até ao primeiro 1;
substituir 1 por Y;
(q_2) (saltando 0's e Y's) recuar até encontrar um X;



O ciclo começa no estado q_0 e regressa a q_0 sempre que a cabeça de leitura recua até ao próximo 0 a ser assinalado.
Em cada instante, a zona da Fita já visitada é da forma X*0*Y*1*.



Se (em q_0) em vez de um 0 encontrar um Y,
é porque todos os 0's já foram substituídos por X's.
(Em q_3) saltando todos os Y's, se encontrar um □,
(em q_4) a concordância é verificada e a palavra é reconhecida.



Se (q_3) após o último Y estivesse 0 ou 1 a Máquina de Turing parava (por não estarem definidas transições, a partir de q_3, por 0 ou 1) sem atingir o Estado Final (q_4).

Por outro lado, para w = 0010:

$q_0 0010 \vdash X q_1 010 \vdash X0 q_1 10 \vdash X q_2 0Y0 \vdash q_2 X0Y0 \vdash$
$X q_0 0Y0 \vdash XX q_1 Y0 \vdash XXY q_1 0 \vdash XXY0 q_1 □$

e a MT parava (por não estarem definidas transições, a partir de q_1, por □) sem atingir o Estado Final (q_4).

Identifique os outros casos possíveis de não aceitação e o correspondente comportamento da MT.

Exercício: Construa uma Máquina de Turing para reconhecer

a Linguagem $\{ w \in \{0, 1\}^* \mid n_0(w) = n_1(w) \}$.

*{ Apesar de, no contexto actual, o interesse da MT residir essencialmente no **Reconhecimento de Linguagens**, o conceito original de Alan Turing era o de um dispositivo de **Cálculo** }*

Exemplo: A MT seguinte **calcula a "diferença própria"** entre dois números inteiros e positivos, isto é,

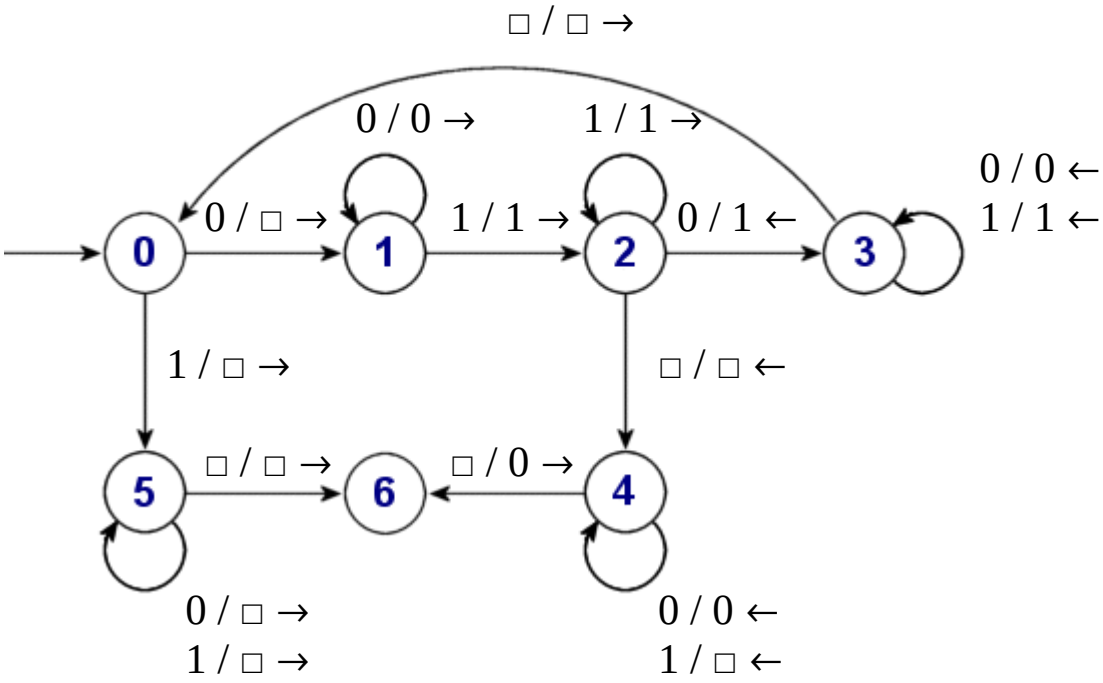
$$\begin{cases} m - n & \text{se } m \geq n \\ 0 & \text{se } m < n \end{cases}$$

$M = (\{q_0, q_1, q_2, q_3, q_4, q_5, q_6\}, \{0, 1\}, \{0, 1, \square\}, \delta, q_0, \square, \{\})$

com:

	δ	0	1	□
⟶	q_0	$(q_1, \square, R)$	$(q_5, \square, R)$	—
	q_1	$(q_1, 0, R)$	$(q_2, 1, R)$	—
	q_2	$(q_3, 1, L)$	$(q_2, 1, R)$	$(q_4, \square, L)$
	q_3	$(q_3, 0, L)$	$(q_3, 1, L)$	$(q_0, \square, R)$
	q_4	$(q_4, 0, L)$	$(q_4, \square, L)$	$(q_6, 0, R)$
	q_5	$(q_5, \square, R)$	$(q_5, \square, R)$	$(q_6, \square, R)$
	q_6	—	—	—

- Porque não se trata do Reconhecimento de uma Linguagem, não são definidos estados finais (ou de aceitação);
- O processo **pára sempre** no estado q_6, ficando o resultado registado na Fita;
- É utilizada uma forma de representação "unária" dos números;
- Os operandos m e n começam por ser registados na forma 0^m10^n seguidos de brancos. O resultado será 0^{m-n} (entre brancos) ou Fita Vazia (só brancos).



- Essencialmente, cada 0 de m é substituído por □ e cada 0 de n é substituído por 1.
- Se m ≥ n (em q_4) os (n+1) 1's são substituídos por □ e é reposto um 0 que faltava.
- Se m < n (em q_5) todos os 0's e 1's são substituídos por □.

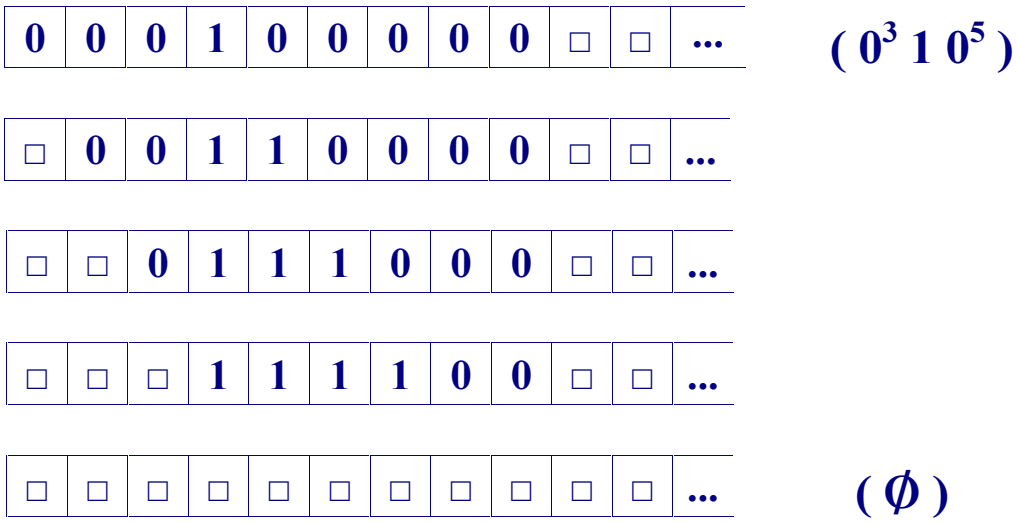
Simulação para **5 – 3** :

0	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	□	□	...
□	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**0**	□	□	...
□	□	**0**	**0**	**0**	**1**	**1**	**1**	**0**	□	□	...
□	□	□	**0**	**0**	**1**	**1**	**1**	**1**	□	□	...
□	□	□	□	**0**	**1**	**1**	**1**	**1**	□	□	...
□	□	□	**0**	**0**	□	□	□	□	□	□	...

(0^5 1 0^3)

(0^2)

Simulação para **3 – 5** :



Exercício: Construa uma Máquina de Turing para **calcular a soma** de dois números inteiros e positivos.

Exemplo: **Calcular o produto** de dois números inteiros e positivos.

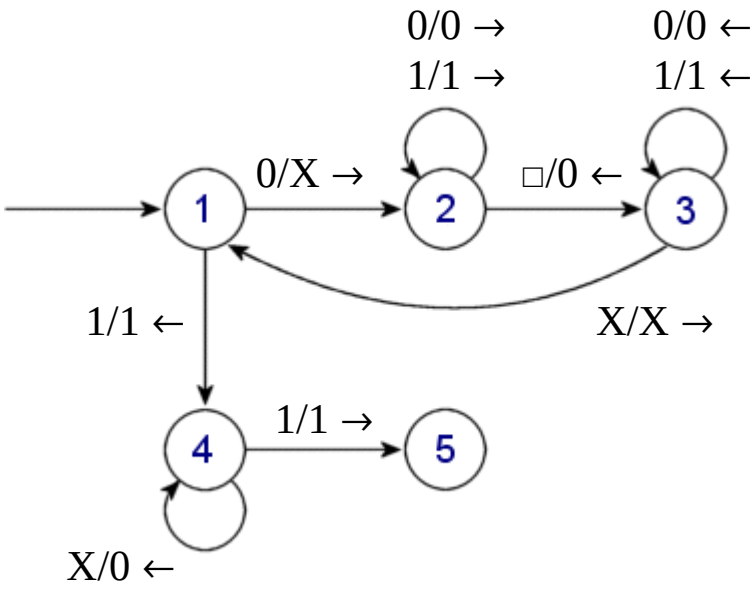
- Os operandos m e n são registados na forma $\mathbf{0^m 1 0^n 1}$ seguidos de brancos e resultado será $\mathbf{0^{mn}}$ (entre brancos), isto é,

  $0^m\ 1\ 0^n\ 1 \vdash^* 0^{mn}$

- Utilizamos um módulo auxiliar para **copiar** $\mathbf{0^n}$ no fim da informação registada (antes dos brancos);

  $\alpha\ 1\ 0^n\ 1\ \beta \vdash^* \alpha\ 1\ 0^n\ 1\ \beta\ 0^n \quad \alpha, \beta \in \{0, 1\}^*$

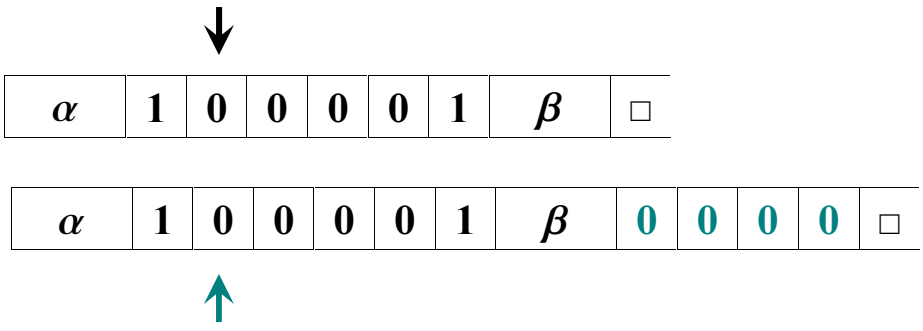
O módulo para **copiar** $\mathbf{0^n}$:



Especificação:

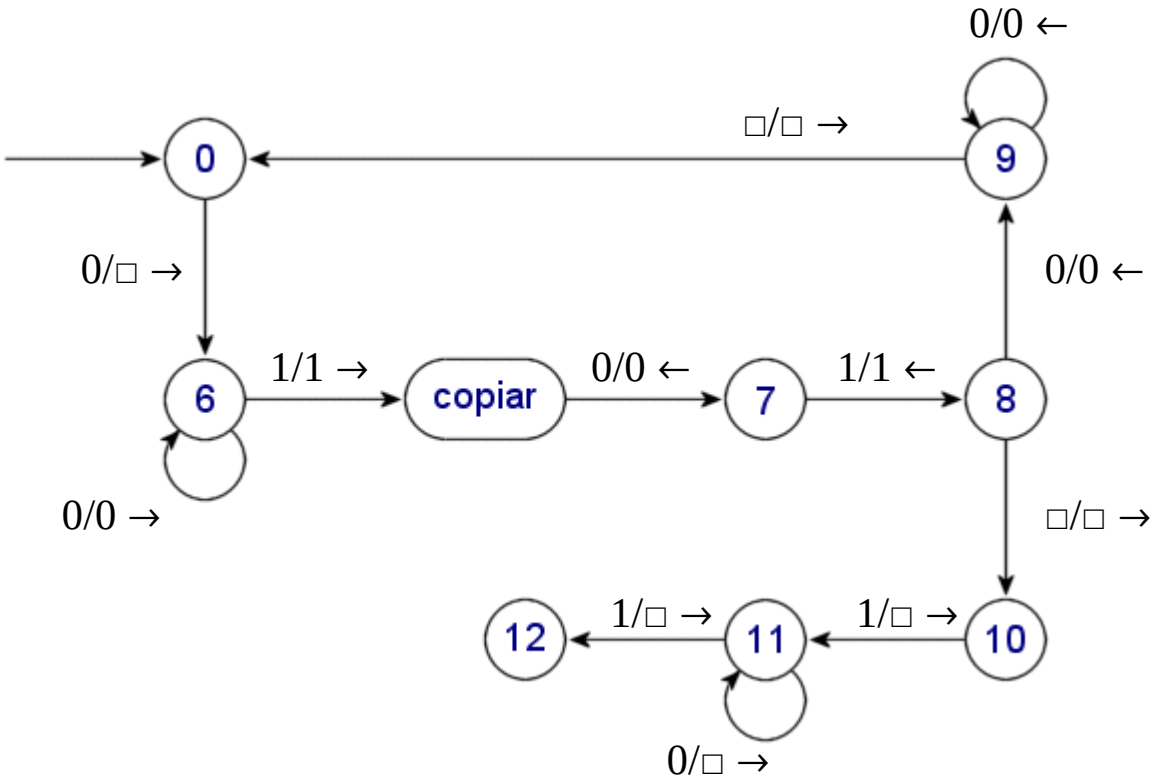
$$\alpha\ 1\ 0^n\ 1\ \beta \ \vdash^*\ \alpha\ 1\ 0^n\ 1\ \beta\ 0^n \qquad \alpha, \beta \in \{0, 1\}^*$$

Simulação para **n = 4** :



- O processo começa com a cabeça de leitura no primeiro 0 (q_1) e acaba na mesma posição (q_5), para que o módulo possa ser repetidamente utilizado durante a multiplicação.
- Os X's auxiliares voltam a ser substituídos por 0's (q_4).

Uma Máquina de Turing para o **produto**:



Especificação:

$$\mathbf{0^m\ 1\ 0^n\ 1 \vdash^*\ 0^{mn}}$$

- No primeiro passo, $\mathbf{0^m\ 1\ 0^n\ 1 \vdash\ 0^{m-1}\ 1\ 0^n\ 1\ 0^n}$
- No passo de ordem k, $\mathbf{0^{m-(k-1)}\ 1\ 0^n\ 1\ 0^{(k-1)n} \vdash\ 0^{m-k}\ 1\ 0^n\ 1\ 0^{kn}}$
- No passo de ordem m, $\mathbf{0\ 1\ 0^n\ 1\ 0^{(m-1)n} \vdash 1\ 0^n\ 1\ 0^{mn}}$
- Quando não existem mais 0's do m inicial (q_8), é apagada a sequência $\mathbf{1\ 0^n\ 1}$ (q_{10}) (q_{11}) (q_{12}).
- No fim do processo, a cabeça de leitura fica no primeiro 0 de $\mathbf{0^{mn}}$.

⇨ Linguagens de uma Máquina de Turing:

Para uma Máquina de Turing $M = (Q, \Sigma, \Gamma, \delta, q_0, \square, F)$ definimos:

- **Linguagens aceites por Estado Final:**

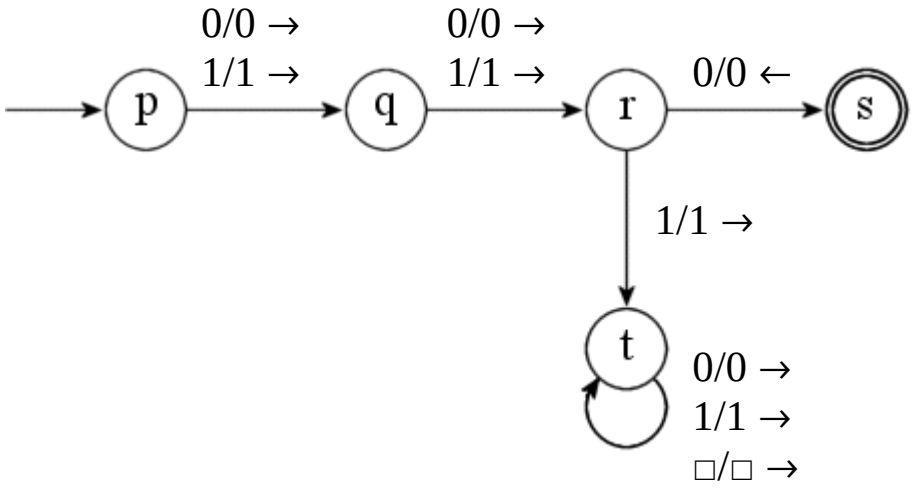
  $\mathbf{L(M)} = \{ w \in \Sigma^* \mid \exists\, p \in F, \exists\, \alpha, \beta \in \Gamma^* : \; q_0 w \vdash^* \alpha p \beta \}$

- **Linguagens aceites por Paragem:**

  $\mathbf{H(M)} = \{ w \in \Sigma^* \mid \exists\, p \in Q, \exists\, \alpha, \beta \in \Gamma^*, \exists\, X \in \Gamma \; :$

  $q_0 w \vdash^* \alpha p X \beta$ e $\delta(p, X)$ não está definido$\}$

Exemplo: Recordemos a Máquina de Turing que reconhece as palavras de $\{0, 1\}^*$ cujo terceiro símbolo é um 0,



$\mathbf{L(M)} = (0 + 1)\,(0 + 1)\,0\,(0 + 1)^*$

$\mathbf{H(M)} = \varepsilon + 0 + 1 + (0 + 1)\,(0 + 1) + (0 + 1)\,(0 + 1)\,0\,(0 + 1)^*$

⇨ Classes de Linguagens:

- **Linguagens Recorrentemente Enumeráveis:**

  As que são aceites por Estado Final, por alguma Máquina de Turing,

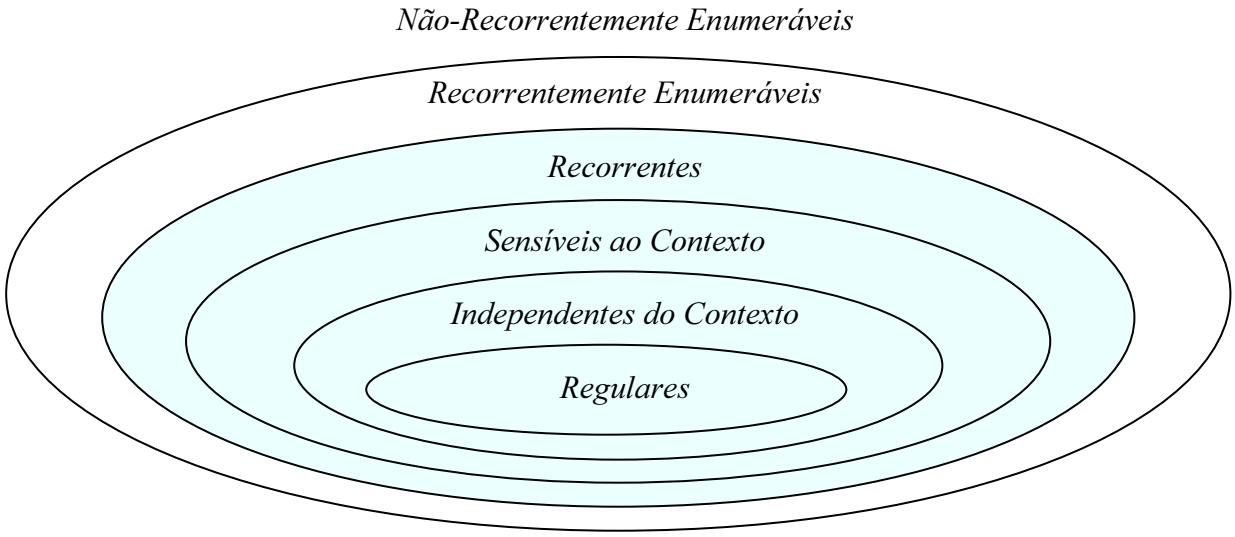
  *{ Podemos estabelecer uma regra (recorrente) para listar ordenadamente todas as palavras da Linguagem. Contudo, para uma palavra que não pertença à Linguagem, a MT pode **não parar**, não sendo portanto possível verificar se a palavra pertence ou não à Linguagem. }*

- **Linguagens Recorrentes:**

  As que são aceites por Paragem, por alguma Máquina de Turing,

  *{ A Máquina de Turing **pára sempre**, verificando se a palavra pertence ou não à Linguagem. }*

A Hierarquia de Chomsky:



{ Interessam-nos as Linguagens Recorrentes. }

{ Uma Máquina de Turing que pára sempre, é um Modelo adequado de Algoritmo. }

⇨ Algoritmo:

Um **Algoritmo** para calcular o valor de uma função f : C ⟶ D é uma Máquina de Turing que, partindo de um d ∈ C registado na Fita, **pára** com o valor exacto de f(d) ∈ D registado na Fita.

{ Assumindo a Tese de Church-Turing, em vez de "Máquina de Turing" podemos dizer "programa". }

⇨ Algoritmos e Problemas:

{ Representando toda a informação em binário. }

Uma **Instância** (particularização de um Problema) é um par ordenado:

$$(x, y) \mid x, y \in \{0, 1\}^*$$

Um **Problema** é um conjunto de Instâncias.

$$\{ (x, y) \mid x, y \in \{0, 1\}^* \}$$

Exemplo: Soma de Números Naturais

Instância: $(0^m 1 0^n , 0^{m+n}) \mid m, n \in \mathbb{N}$
Problema: $\{ (0^m 1 0^n , 0^{m+n}) \mid \forall\ m, n \in \mathbb{N} \}$

Algoritmo: $0^m 1 0^n \vdash^* 0^{m+n}$

{ Um Algoritmo é a solução de um Problema. }

⇨ Problemas de Decisão e Linguagens:

Num **Problema de Decisão** os resultados possíveis são **Sim** ou **Não**.

$$\{ (x, y) \mid x \in \{0, 1\}^*, \ y \in \{0, 1\} \}$$

Exemplo: Verificar se um dado número $n \in \mathbb{N}$ é Primo.
$x \in \{0, 1\}^*$ é uma representação de n
$y \in \{0, 1\}$ representa {Não, Sim}

A cada **Problema de Decisão** D corresponde uma **Linguagem**, formada pelas palavras para as quais o resultado é **Sim**.

$$\mathbf{L(D)} = \{ x \in \{0, 1\}^* \mid (x, 1) \in D \}$$

Resolver o Problema de Decisão equivale a Reconhecer a Linguagem.

Exemplo: Reconhecer a Linguagem,
$\{ x \in \{0, 1\}^* \mid x$ representa um número Primo $\}$

Qualquer Problema pode ser formulado como um Problema de Decisão. Se conhecermos o resultado …

$$P = \{ (x, y) \mid x, y \in \{0, 1\}^* \}$$

$$D(P) = \{ (xy, z) \mid z = 1 \Leftrightarrow (x, y) \in P \}$$

Exemplo: Dados m, n e p, verificar se m+n = p.

$$L(D) = \{ (0^m 1 0^n 1 0^{m+n}) \mid \forall \ m, n \in \mathbb{N} \}$$

{ Podemos encarar um Problema como uma Linguagem.}

{ Todo o Problema pode ser "resolvido"? }

⇨ Problemas Decidíveis e Problemas Indecidíveis:

Um Problema diz-se **Decidível** se a linguagem associada for uma Linguagem Recorrente e **Indecidível** se não for Recorrente.

Isto é, um Problema é Decidível se existir uma Máquina de Turing capaz de calcular o resultado ao fim de um número finito de passos, para todas as suas Instâncias.

{ Um Problema é Decidível quando existe um Algoritmo que o resolve e pára! }

Hilbert (1928): *Qualquer problema pode ser "resolvido"?*

Gödel (1931): *Não pode! Demonstrei com um Algoritmo Recorrente.*

Turing (1936): *Um Algoritmo é uma "máquina" ...*

Alguns Problemas **Indecidíveis**:

- O Problema da Paragem.
- O 10º Problema de Hilbert (1900).

  Resolver a equação, $x^n + y^n = z^n$
  com $x, y, z, n \in \mathbb{Z}$, $n > 2$

- Verificar se uma Gramática é Ambígua.
- Verificar se uma Linguagem é Finita.
- ...
- *{Muitos, muitos mais}*

{ Podemos verificar se a Máquina pára? }

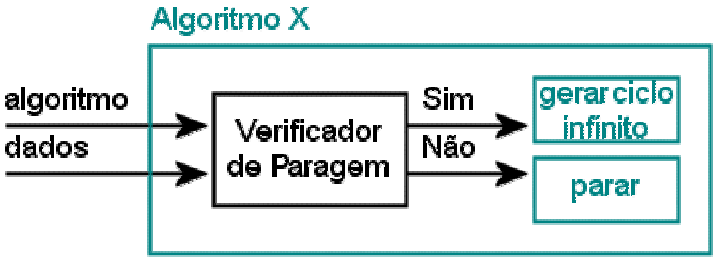
⇨ O Problema da Paragem:

Existirá um algoritmo (MT) para verificar se todo o algoritmo (MT) **pára**, para qualquer instância?

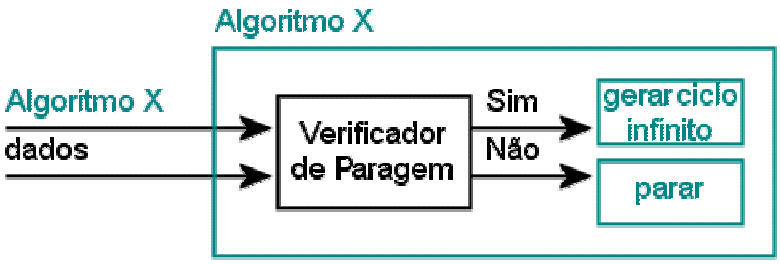
Resposta: **Não!**

Esquema de Demonstração:

Se tal algoritmo existisse, poderíamos construir outro:



… e como o algoritmo de entrada é qualquer:



… o que é absurdo!

{ Portanto não podemos! }

*{ Fiquemo-nos pelos Problemas **Resolúveis**. }*

*{ Mas há muitas maneiras de resolver um Problema Resolúvel e há problemas mais "**facilmente** resolúveis" do que outros ... }*

⇨ Complexidade Temporal:

Seja M uma Máquina de Turing Determinista que resolve o Problema $\{ (x, y) \mid x, y \in \{0, 1\}^* \}$ e pára para todo o x.

A **Complexidade Temporal** de M é uma função $f : \mathbb{N} \longrightarrow \mathbb{N}$, onde f(n) é o **maior número de passos** que M efectua sobre as palavras x de comprimento n.

No conjunto das palavras x de comprimento n, esse número de passos pode variar bastante, consoante a **Instância** considerada. A definição anterior corresponde ao **Pior Caso**.

O mesmo Problema pode ser resolvido por várias Máquinas de Turing com diferentes Complexidades Temporais.

Exemplo: Reconhecer a Linguagem $\{ 0^n 1^n \mid n \geq 0 \}$.
A MT trivial tem Complexidade $O(n^2)$
mas existe outra de Complexidade $O(n \log n)$
e uma MT de duas Fitas pode fazê-lo em $O(n)$.

{ Todas as noções anteriores são para MT's Deterministas. }

Definição: Numa Máquina de Turing **Não Determinista**,

$$\delta : Q \times \Gamma \longrightarrow \mathcal{P}(Q \times \Gamma \times \{L, R\})$$

⇨ A Classe $\mathcal{P}$:

Chama-se $\mathcal{P}$ ao conjunto de Problemas (Linguagens) resolúveis (reconhecíveis) por uma Máquina de Turing **Determinista** que pára em **tempo polinomial**.

{ Claro que a MT pode resolver o problema em $\mathcal{O}(n^6)$ enquanto que um programa o resolve em $\mathcal{O}(n^2)$. }

⇨ A Classe $\mathcal{NP}$:

Chama-se $\mathcal{NP}$ ao conjunto de Problemas (Linguagens) resolúveis (reconhecíveis) por uma Máquina de Turing **Não Determinista** que pára em **tempo polinomial**.

{ portanto $\mathcal{P} \subseteq \mathcal{NP}$ }

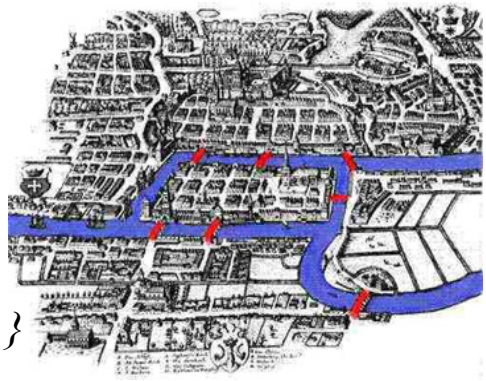
São Problemas da Classe $\mathcal{P}$:

- Máximo Divisor Comum,
- Ordenação,
- Programação Linear,
- Circuito Euleriano,
- Caminho Mais Curto (1959),
- Teste de Primalidade (2002),
- …

São Problemas da Classe $\mathcal{NP}$:

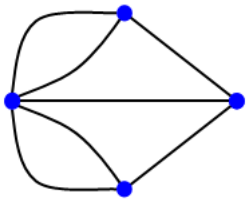
- Torres de Hanoi,
- Circuito Hamiltoniano (Problema do Caixeiro Viajante),
- Coloração de Grafos,
- "Minesweeper"
- ...

⇨ Circuito Euleriano:



{ Euler e as pontes de Königsberg (1736). }

O Problema:



Para um dado grafo, procurar um circuito que passe por cada arco uma só vez.

O Circuito Euleriano é um Problema da Classe $\mathcal{P}$ se os vértices forem de grau ímpar.

⇨ Circuito Hamiltoniano:

Caixeiro Viajante:
Dadas n cidades e as respectivas distâncias, procurar o circuito total mínimo.

O Problema:

Para um dado grafo, procurar um circuito que passe por cada vértice uma só vez.

O Circuito Hamiltoniano é um Problema da Classe $\mathcal{NP}$.

{ *Encontrar a solução é "difícil", mas verificar a solução é "fácil".* }

⇨ Determinismo e Não-Determinismo:

Numa Instância de um Problema $\mathcal{NP}$ a pesquisa (determinista) do resultado pode crescer de forma **exponencial**.

Exemplo: Problema do Caixeiro Viajante (pesquisa exaustiva):

```
Gerar todas as (2^n) Permutações de (1,2,3,…,n);
calculando a soma das distâncias;
Escolher a menor.
```

… mesmo para 1 nanossegundo por permutação:

n	2^n (ns)	segundos	dias	anos
10	1024.	1.024×10^{-6}	1.18519×10^{-11}	3.24708×10^{-14}
20	1.04858×10^{6}	0.00104858	1.21363×10^{-8}	3.32501×10^{-11}
30	1.07374×10^{9}	1.07374	0.0000124276	3.40481×10^{-8}
40	1.09951×10^{12}	1099.51	0.0127258	0.0000348653
50	1.1259×10^{15}	1.1259×10^{6}	13.0312	0.0357021
60	1.15292×10^{18}	1.15292×10^{9}	13344.	36.5589
70	1.18059×10^{21}	1.18059×10^{12}	1.36643×10^{7}	37436.3
80	1.20893×10^{24}	1.20893×10^{15}	1.39922×10^{10}	3.83348×10^{7}
90	1.23794×10^{27}	1.23794×10^{18}	1.4328×10^{13}	3.92548×10^{10}
100	1.26765×10^{30}	1.26765×10^{21}	1.46719×10^{16}	4.01969×10^{13}

Mas se o Problema é $\mathcal{NP}$, uma MT **Não Determinista** encontra o resultado em **tempo polinomial**.

Se à partida conhecermos o resultado (Problema de Decisão associado), uma MT **Determinista** pode **verificar** (Sim ou Não) esse resultado em **tempo polinomial**.

{ *Problemas $\mathcal{NP}$:*
Tenta-se encontrar um algoritmo polinomial e tornam-se $\mathcal{P}$,
Tenta-se provar que são mesmo $\mathcal{NP}$,
Tenta-se construir soluções aproximadas. }

*{ Mas a verdade é que **não sabemos** se existirá, ou não, alguma MT determinista capaz de resolver estes problemas em tempo polinomial.*

Nem sequer sabemos se existe algum problema $\mathcal{NP}$ que não seja $\mathcal{P}$. }

⇨ A Grande Questão:

$$\mathcal{P} \stackrel{?}{=} \mathcal{NP}$$

{ Pode ganhar 1 milhão de dólares! }

⇨ A Classe $\mathcal{NP}$ – completo:

Não fazemos a menor ideia se $\mathcal{P} = \mathcal{NP}$ mas sabemos demonstrar que **alguns** Problemas são tão "difíceis" que, se alguma vez algum deles for classificado como $\mathcal{P}$, então $\mathcal{P} = \mathcal{NP}$.

São Problemas da Classe $\mathcal{NP}$-completo:

- Problema do Caixeiro Viajante,
- Coloração de Grafos,
- ...