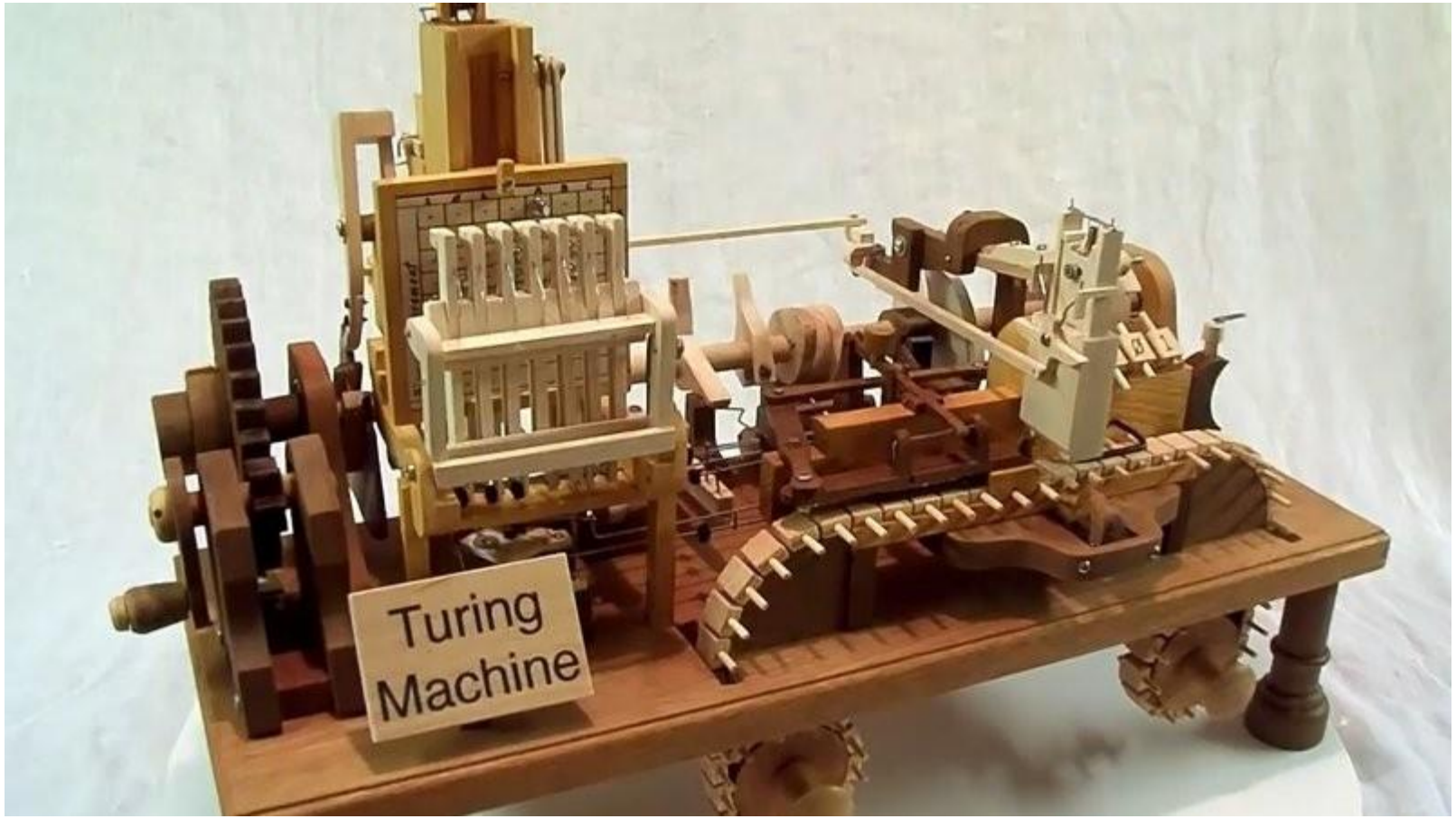
Introdução à Análise de Algoritmos

profº Mauricio Conceição Mario

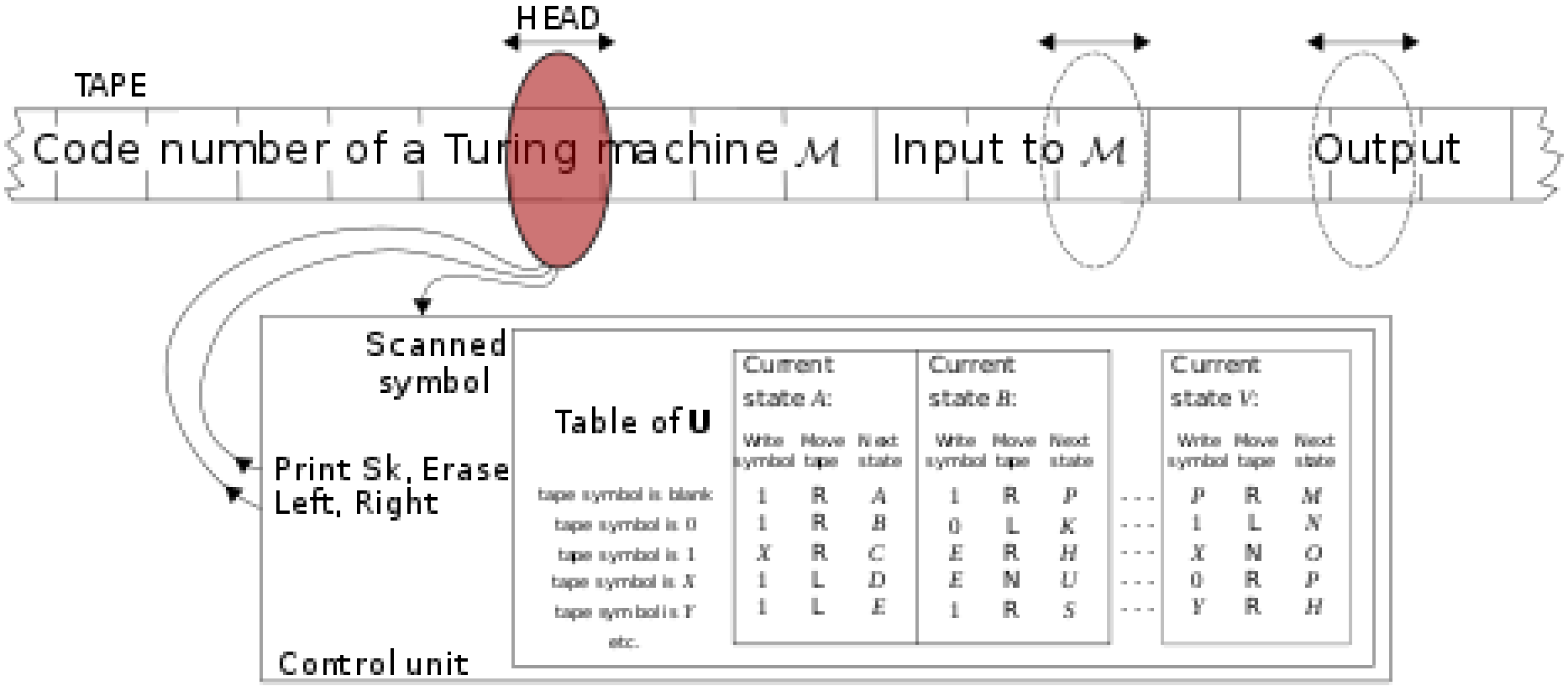
Alan Turing e representação da Máquina Universal de Turing

https://www.puc-campinas.edu.br/museu-anterior/alan-turing/



https://revistagalileu.globo.com/Tecnologia/noticia/2018/03/origem-da-computacao-maquina-de-turing-e-construida-em-madeira.html

Máquina Universal de Turing



https://www.google.com/search?q=m%C3%A1quina+de+turing+universal&sxsrf=ALiCzsZUV4g88

Dado um conjunto de símbolos e estados de uma máquina que será avaliada por uma Máquina de Turing:
→ contém uma fita que codifica as transições de estado da máquina avaliada.

https://dcc.ufrj.br/~collier/CursosGrad/LF/TuringP.pdf

Problemas Computáveis (decidíveis) →
Significa que existe algum algoritmo que calcula uma resposta (ou saída) para qualquer instância do problema (ou para qualquer entrada do problema) em um número finito de etapas simples.

Problemas Não Computáveis (indecidíveis) →
Significa que não existe nenhum algoritmo que possa computar uma resposta ou saída para todas as entradas em um número finito de etapas simples.

Exemplo: Problema de Halting →
Dada a descrição de uma Máquina de Turing e sua entrada inicial, determina se o programa, quando executado nesta entrada, sempre pára (é concluído). A alternativa é que ele funcione para sempre sem parar.

A influência da eficiência de algoritmos no tempo de execução de programas de computadores

- **Diferença entre a tecnologia dos computadores:**
  - velocidade do processador;
  - capacidade de memória;
- **Compilador:**
  - linguagem de máquina;
  - linguagem de alto nível;
- **Paradigma de programação:**
  - programação estruturada;
  - orientação a objetos;
  - programação procedural;
- **Algoritmos:**
  - dimensão dos dados;
  - número de passos;
  - tempo de execução.

(Cormen et al., 2012)

Complexidade de Algoritmos – conceitos:

→ Característica de um algoritmo: tempo de execução.

→ *É possível obter uma ordem de grandeza do tempo de execução através de* ***métodos analíticos****:*

→ *determinam uma expressão matemática que traduza o comportamento do tempo de um algoritmo. Um método analítico visa aferir o tempo de execução de forma independente do computador utilizado, da linguagem de programação e dos compiladores empregados, além das condições de processamento.*

Exemplo: função que corresponde ao tempo de execução de uma instrução i, onde c_i é uma constante proporcional à complexidade desta i-ésima instrução e n é a dimensão dos dados que esta instrução manipula.

$$f_{(n)} = c_i.n^2$$

(Szwarcfiter & Markenzon, 2020)

Complexidade de Algoritmos – conceitos:

→ O processo de execução de um algoritmo pode ser dividido em etapas elementares, denominadas **passos**.

→ *Cada passo consiste na execução de um número fixo de operações básicas, cujos tempos de execução são considerados constantes.*

para i := 1, ..., ⌊n/2⌋ fazer

```
temporário := S[i]
S[i] := S[n  - i + 1]
S[n  - i + 1] := temporário
```

⇦ **Exemplo:**
n/2 passos = *nº* operações de atribuição no trecho de código!

(Szwarcfiter & Markenzon, 2020)

Complexidade de Algoritmos – conceitos:

→ O processo de execução de um algoritmo pode ser dividido em etapas elementares, denominadas **passos**.

→ *A operação básica de maior frequência de execução é denominada* ***operação dominante****.*

→ *O* ***número de passos*** *de um algoritmo pode ser interpretado como sendo o* ***número de execuções da operação dominante****.*

```
para i := 1, ..., ⌊n/2⌋ fazer
    temporário := S[i]
    S[i] := S[n  - i + 1]
    S[n  - i + 1] := temporário
```

⇦ **Exemplo:**
operação dominante = atribuição

→ *É então natural definir uma* ***expressão matemática*** *de avaliação do tempo de execução de um algoritmo como sendo uma* ***função que fornece o número de passos efetuados pelo algoritmo****, a partir de uma certa entrada.*

(Szwarcfiter & Markenzon, 2020)

Símbolos usados em algoritmos :

- *Declaração de atribuição* → $f := h$ (valor de h atribuído à variável f);
- *Elementos de vetores e matrizes:* → $A[n]$: enésimo elemento do vetor A;
  → $B[i, j]$: elemento identificado pelos índices (i, j) da matriz B;
- *Notação **T.chave**, para registros* → indica o campo ***chave*** do registro ***T***;
- *Referência de registros* → pode ser feita através de ponteiros, que armazenam endereços utilizando o símbolo ↑; ***pt*↑.*informação*** representa o campo ***informação*** de um registro alocado no endereço contido em ***pt***.
- *Uso de comentários* → após **%**;
- *Notação* $\lfloor n \rfloor$ → piso de *n*, ou seja, o maior inteiro menor ou igual a *n:*
- $\lfloor 9/2 \rfloor = 4$.
- *Notação* $\lceil n \rceil$ → teto de *n*, ou seja, o menor inteiro maior ou igual a *n:*
- $\lceil 9/2 \rceil = 5$.

(Szwarcfiter & Markenzon, 2020)

Exemplo 1

Complexidade de Algoritmos – conceitos:

→ Exemplo de algoritmo: *Inversão de uma sequência*.

para $i := 1, ..., \lfloor n/2 \rfloor$ *fazer*
    temporário $:= S[i]$
    $S[i] := S[n - i + 1]$
    $S[n - i + 1] :=$ *temporário*

n = 5
S = [1, 2, 3, 4, 5]
temp = S[1]
S[1] = S[(5 - (i = 1)) + 1] = S[5]
S[(5- (i = 1)) + 1] = temp = S[1]

Algoritmo: *dada uma sequência de elementos armazenada no vetor* $S[i]$, $1 \leq i \leq n$, *deseja-se inverter os elementos da sequência no vetor, ou seja, considerando a sequência de trás para frente. O algoritmo troca a posição do primeiro com o último elemento e em seguida o segundo com o antepenúltimo, e assim por diante.*

(Szwarcfiter & Markenzon, 2020)

Exemplo 1

Complexidade de Algoritmos – conceitos → número de passos:

→ algoritmo: *Inversão de uma sequência*:

Cada entrada é uma sequência que se deseja inverter. O algoritmo efetua as mesmas operações para sequência de mesmo tamanho *n*. Cada passo corresponde à troca de posição entre dois elementos da sequência. Ou seja, à execução das três instruções de atribuição dentro do bloco ***para*** do algoritmo. O número de passos é, então, igual ao número de execuções do bloco ***para***. Ou seja, **nº passos** = $\lfloor n/2 \rfloor$, $n > 1$.

para $i := 1, ..., \lfloor n/2 \rfloor$ ***fazer***

um passo →

```
temporário := S[i]
S[i] := S[n - i + 1]
S[n - i + 1] := temporário
```

(Szwarcfiter & Markenzon, 2020)

C:\Users\cmari\.spyder-py3

C:\Users\cmari\.spyder-py3\inverte_sequência.py

Exemplo 1

grafico_2.py | grafico_3.py | hipotenusa.py | grafico_4.py | history_internal.py | função_enumerate.py | iteradores.py | inverte_sequência.py

```
1  # -*- coding: utf-8 -*-
2  """
3  Created on Tue Jul  5 12:06:51 2022
4
5  @author: cmari
6  """
7  #https://acervolima.com/como-verificar-o-tempo-de-execucao-do-script-python/
8
9  import time
10 #from datetime import datetime
11
12 n = 11
13 S = [x for x in range (n)]
14 print ("vetor S =           ", S)
15 #inicio = datetime.now()
16 tempo_inicial = time.time()
17 for i in range(0, int((n/2))):
18     temporario = S[i]
19     S[i] = S[(n-1) - i]        #(n-1): para i = zero extrapola tamanho do vetor
20     S[(n-1) - i] = temporario  #(n-1): para i = zero extrapola tamanho do vetor
21 tempo_final = time.time()
22 #fim = datetime.now()
23 print ("vetor S invertido = ", S)
24 print("tempo de execução = ", tempo_final - tempo_inicial )
25 #print("tempo de execução = ", str(fim - inicio))
26
27
```

i = 0

temporário = S[0] = 0

S[0] = S[(11 – 1) – 0] = S[10] = 10

S[(11 – 1) – 0] = S[10] = temporário = 0

i = 1

temporário = S[1] = 1

S[1] = S[(11 – 1) – 1] = S[9] = 9

S[(11 – 1) – 1] = S[9] = temporário = 1

...

Console 1/A

```
In [23]: runfile('C:/Users/cmari/.spyder-py3/inverte_sequência.py', wdir='C:/Users/
cmari/.spyder-py3')
vetor S =           [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
vetor S invertido =  [10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
tempo de execução =  0.0
```

C:\Users\cmari\.spyder-py3

C:\Users\cmari\.spyder-py3\inverte_sequência.py

Exemplo 1

grafico_2.py | grafico_3.py | hipotenusa.py | grafico_4.py | history_internal.py | função_enumerate.py | iteradores.py | inverte_sequência.py

```
1   # -*- coding: utf-8 -*-
2   """
3   Created on Tue Jul  5 12:06:51 2022
4
5   @author: cmari
6   """
7   #https://acervolima.com/como-verificar-o-tempo-de-execucao-do-script-python/
8
9   import time
10  #from datetime import datetime
11
12  n = 100000
13  S = [x for x in range (n)]
14  #print ("vetor S =           ", S)
15  #inicio = datetime.now()
16  tempo_inicial = time.time()
17  for i in range(0, int((n/2))):
18      temporario = S[i]
19      S[i] = S[(n-1) - i]        #(n-1): para i = zero extrapola tamanho do vetor
20      S[(n-1) - i] = temporario  #(n-1): para i = zero extrapola tamanho do vetor
21  tempo_final = time.time()
22  #fim = datetime.now()
23  #print ("vetor S invertido = ", S)
24  print("tempo de execução = ", tempo_final - tempo_inicial )
25  #print("tempo de execução = ", str(fim - inicio))
26
```

Console 1/A

```
In [27]: runfile('C:/Users/cmari/.spyder-py3/inverte_sequência.py', wdir='C:/Users/
cmari/.spyder-py3')
tempo de execução =  0.06185150146484375
```

Referências Bibliográficas

- Estruturas de Dados e Seus Algoritmos
  Jayme L. Szwarcfiter & Lilian Markenzon
  3ª edição – editora *gen* LTC – 2010 – 2020

- Algoritmos – Teoria e Prática
  Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein.
  3ª edição – editora Elsevier - *gen* LTC – 2012