27 Reunião 27: 22/JUL/2021

Figure 1: Fonte: Calvi e Hobbes por Bill Watterson

27.1 Reuniões passadas

Temos contruído uma **biblioteca para manipulação de matrizes** de nome `matriz.py`:

- `init_matriz(nlins, ncols, val=0)`: cria uma matriz de dimensão `nlins` $\times$ `ncols`
- `leia_matriz()`: invólocro para `leia_matriz_teclado()` e `leia_matriz_arquivo()`
- `leia_matriz_teclado()`: para leitura de matrizes através de digitação no teclado
- `leia_matriz_arquivo(nome_arq)`: para leitura de matrizes de arquivos
- `exiba_matriz(mt)`: exibe a matriz `mt`
- `str_matriz(mt)`: retorna uma string que representa `mt`
- `grave_matriz(mt, nome_arq)`: cria um arquivo `nome_arq` e grava `mt` nesse arquivo
- `simetrica(mt)`: verifica se `mt` é simetrica
- `linhas_val(mt, val)`: conta linhas com todos valores iguais a `val`
- `colunas_val(mt, val)`: conta colunas com todos valores iguais a `val`
- `diagonais(mt, val)`: verifica se diagonais têm apenas valor `val`

- `gire_horizontal(no)`: retorna uma matriz que é `mt` de "ponta-cabeça"
- `gire_vertical()`: retorna uma matriz que é `mt` "refletida" no espelho
- `rode_dir(mt)`: retorna uma matriz que é `mt` rotacionada para a direita
- `rode_esq(mt)`: retorna uma matriz que é `mt` rotacionada para a direita
- `prod(mtA, mtB)`: retorna o produto da `mtA` por `mtB` (**exercício**)

27.2 Hoje

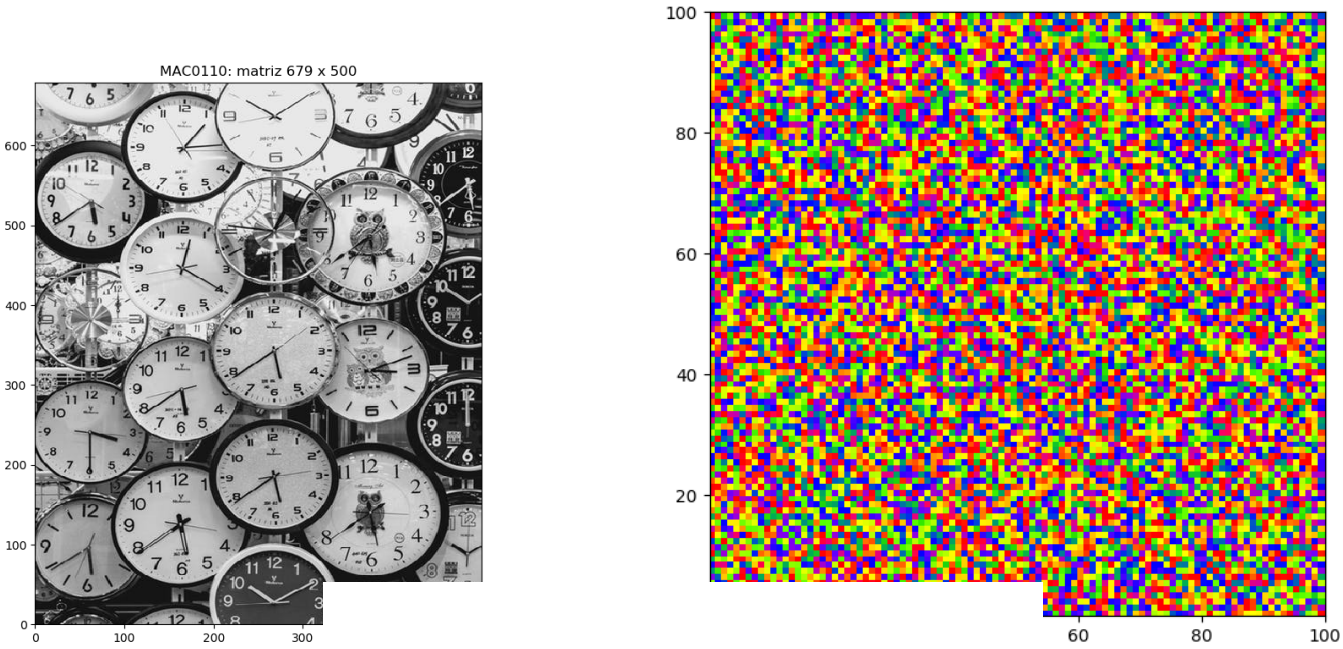

Hoje
• percorrer matrizes
• criar matrizes
• preencher matrizes
list[list]
Ter em mente:
• Atribuição não cria coisa alguma!
• variáveis, posições de listas/matrizes são apelidos!
• cuida para entra no elevad

Sempre ter em mente que

- atribuções não criam nada, apenas criam ou alteram um apelido;
- posições de listas e de matrizes são apelidos para coisas/objetos;
- "**antes** de entrar no elevador, verifique se ele está parado **neste andar**"

Figure 2: Placa na entrada de elevadores

27.3 Matrizes



27.4 Exercício: conte minas

Considere os seguintes apelidos

```
# CONSTANTES
MINA  = -1
LIVRE =  0
```

Escreva uma função `cont_minas()` que **recebe** uma matriz `campo` com os valores `LIVRE` e `MINA` e uma posição inteiros `lin` e `col`. A função **retorna** o número de posições ao redor da posição `[lin][col]` da matriz `campo` que contêm (**são apelidos para**) o valor `MINA`. Há no máximo oito posições ao redor de uma posição qualquer.

Exemplos

```
In [13]: import matriz as mt

In [14]: campo = mt.init_matriz(3,4)

In [15]: campo[0][1]=campo[0][3] = -1

In [16]: campo[1][0]=campo[1][2] = -1

In [17]: campo[2][1]=campo[2][3] = -1

In [18]: mt.exiba_matriz(campo)
Matriz: 3 x 4
     0    -1     0    -1
    -1     0    -1     0
     0    -1     0    -1


In [19]: cont_minas(campo,0,0)
Out[19]: 2

In [20]: cont_minas(campo,1,1)
Out[20]: 4

In [21]: cont_minas(campo,1,3)
Out[21]: 3
```

Solução

```
def cont_minas (campo, lin, col):
    '''(matriz, int, int) -> int

    RECEBE uma matriz `campo` com os valores LIVRE e MINA e
        dois inteiros `lin` e `col`.
    RETORNA o número de posições ao redor da posição
        [lin][col] da matriz campo que contêm o valor MINA.

    PRÉ-CONDIÇÃO: supõe que a posição [lin][col] tem um valor
        diferente de MINA
    '''
    # contador de minas
    no_minas = 0

    # dimensão do campo
    nlin = len(campo)
    ncol = len(campo[0])

    # percorra o campo
    for i in range(lin-1, lin+2):
        for j in range(col-1, col+2):
            # verifique se a posição é valida
            if 0 <= i < nlin and 0 <= j < ncol:
                # se tem mina, conte a mina
                if campo[i][j] == MINA:
                    no_minas += 1
    return no_minas
```

27.5 Exercício: campo minado

Considere os seguintes apelidos

```
# CONSTANTES
MINA  = -1
LIVRE =  0
```

Escreva uma função `init_mapa()` que **recebe** uma matriz `campo` com os valores `LIVRE` e `MINA` em cada posição. e **retorna** uma matriz `mapa`, da mesma dimensão que `campo`, tal que se o valor `campo[i][j]` é:

- `LIVRE`, então `mapa[i][j]` o número de posições com `MINA` em volta de `campo[i][j]`

- `MINA`, então `mapa[i][j]` é `MINA`.

Exemplo

```
In [22]: campo = mt.init_matriz(3,4)

In [23]: campo[0][1]=campo[0][3] = -1

In [24]: campo[1][0]=campo[1][2] = -1

In [25]: campo[2][1]=campo[2][3] = -1

In [26]: mt.exiba_matriz(campo)
Matriz: 3 x 4
     0    -1     0    -1
    -1     0    -1     0
     0    -1     0    -1


In [27]: mapa = init_mapa(campo)

In [28]: mt.exiba_matriz(mapa)
Matriz: 3 x 4
     2    -1     3    -1
    -1     4    -1     3
     2    -1     3    -1
```

Rascunhos

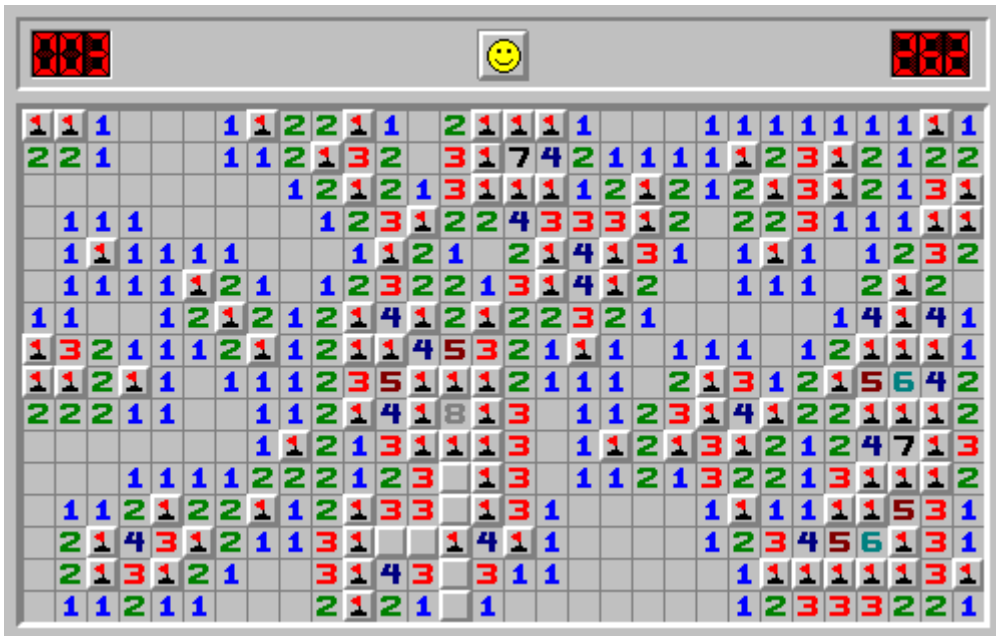


Figure 4: Fonte: Wikipedia

Rascunhos

m \ n	0	1	2	3	4	5
0	0	-1	0	-1	-1	0
1	-1	0	0	0	0	-1
2	0	-1	-1	0	-1	-1
3	-1	0	0	-1	0	-1
4	0	0	0	-1	0	0

m \ n	0	1	2	3	4	5
0	0	-1	0	-1	-1	0
1	-1	0	0	0	0	-1
2	0	-1	-1	0	-1	-1
3	-1	0	0	-1	0	-1
4	0	0	0	-1	0	0

Solução

```
def init_mapa (campo):
    '''(matriz) -> campo

    RECEBE uma matriz `campo` com os valores LIVRE e MINA
        em cada posição.
    RETORNA uma matriz `mapa`, da mesma dimensão que `campo`,
        tal que se o valor campo[i][j] é

       * LIVRE, então mapa[i][j] é o número de posições
                com MINA em volta de campo[i][j]
       * MINA, então mapa[i][j] é  MINA.

    NOTA: A função não é mutadora.
    '''
    nlins = len(campo)
    ncols = len(campo[0])
    mapa = mt.init_matriz(nlins, ncols)
    for i in range(nlins):
        for j in range(ncols):
            if campo[i][j] == LIVRE:
                mapa[i][j] = cont_minas(campo, i, j)
            else:
                mapa[i][j] = MINA
    return mapa
```

27.6 Exercício: produto de matrizes

Neste exercícios escreveremos mais funções que manipulam matrizes para a nossa bibliteca `matriz.py`:

- `prod()`: recebe duas matrizes e retorna o seu produto
- `grave_matriz()`: recebe uma matriz e o nome de um arquivo e grava a matriz no arquivo

Escreva um função que recebe como parâmetros duas matrizes $\texttt{A}_{\texttt{m}\times\texttt{n}}$ e $\texttt{B}_{\texttt{n}\times\texttt{p}}$ e calcula e retorna a matriz $\texttt{C}_{\texttt{m}\times\texttt{p}}$ que é o produto de `A` por `B`.

Exemplos

$A_{2\times3} \times B_{3\times2} = C_{2\times2}$

```
   m = 2   n = 3   p = 2

A    0   1   2   n       B    0   1   p         C   0   1
   +---+---+---+           +---+---+           +---+---+
0  | 1 | 2 |-1 |        0  | 1 |-1 |        0  | 2 |-3 |
   +---+---+---+           +---+---+           +---+---+
1  | 0 | 3 | 2 |    X   1  | 2 | 0 |     =  1  |12 | 4 |
   +---+---+---+           +---+---+           +---+---+
m                       2  | 3 | 2 |
                           +---+---+
                        n
```

```
In [1]: import matriz as mt
In [2]: A = [ [ 1, 2, -1], [ 0, 3, 2] ]
In [3]: B = [ [ 1, -1], [ 2, 0], [ 3, 2] ]
In [4]: mt.exiba_matriz(A)
Matriz: 2 x 3
  1  2 -1
  0  3  2

In [5]: mt.exiba_matriz(B)
Matriz: 3 x 2
  1 -1
  2  0
  3  2
```

```
In [6]: C = mt.prod(A, B)
In [7]: mt.exiba_matriz(C)
Matriz: 2 x 2
  2 -3
 12  4

In [8]: A = [ [ 1, 2, -1], [ 0, 3, 2] ]
In [9]: mt.grave_matriz(A, "arqA.txt")
In [10]: more arqA.txt
2 3
1 2 -1
0 3 2
```

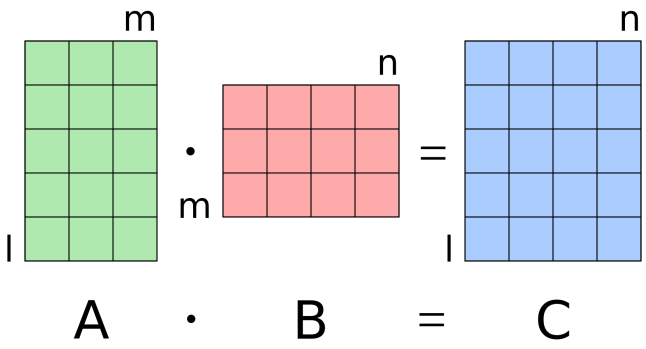


Figure 5: Fonte: Dimensões do produto de matrizes (Wikipedia)

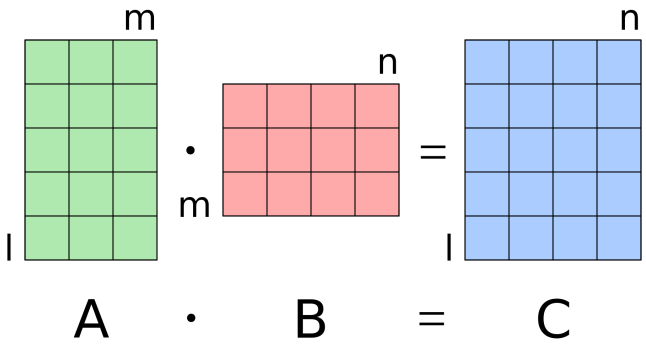
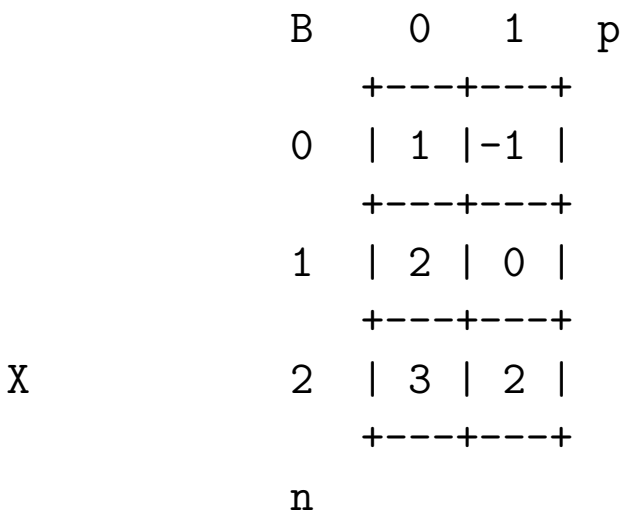
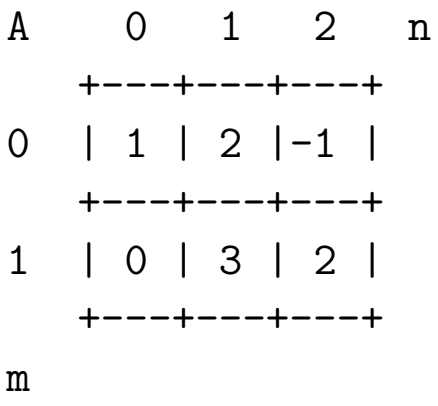
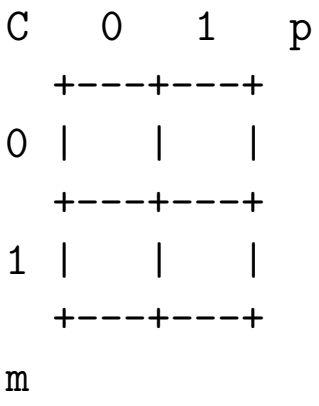


Figure 6: Fonte: Diagrama da multiplicação de matrizes (Wikipedia)

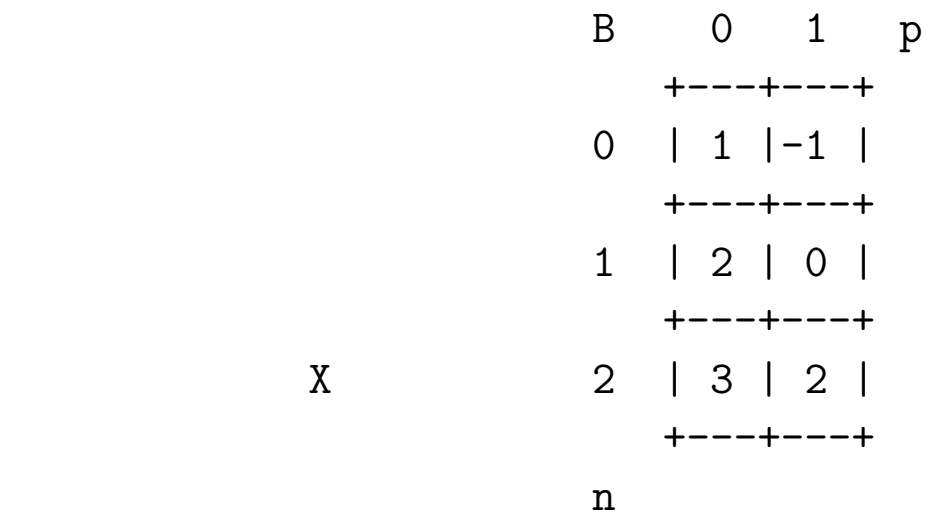
Rascunho

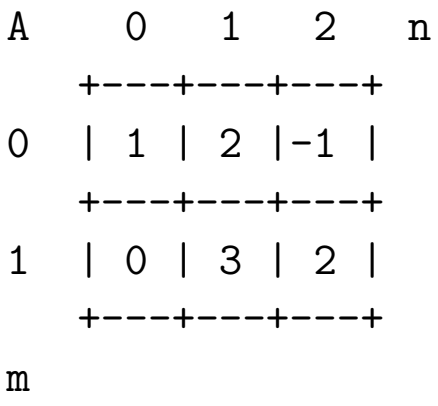


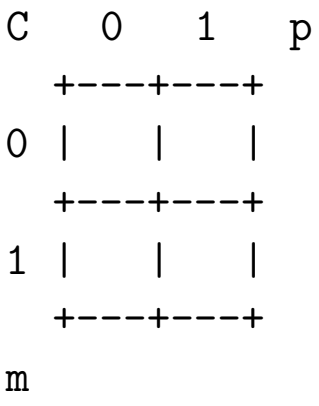




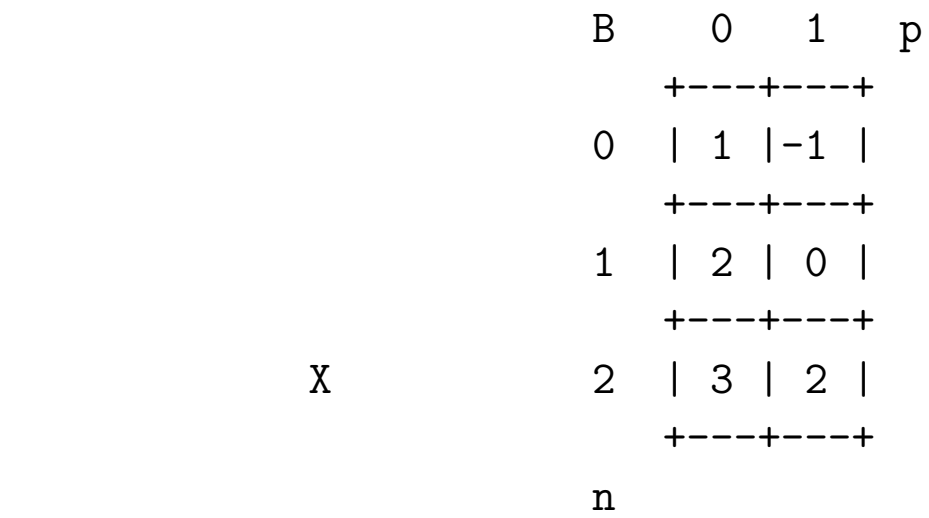
Rascunho

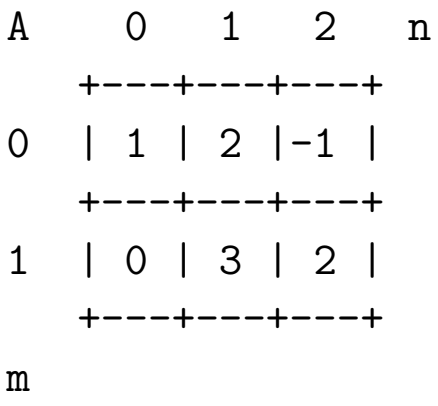


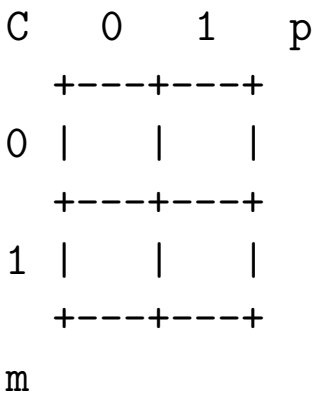




Rascunho

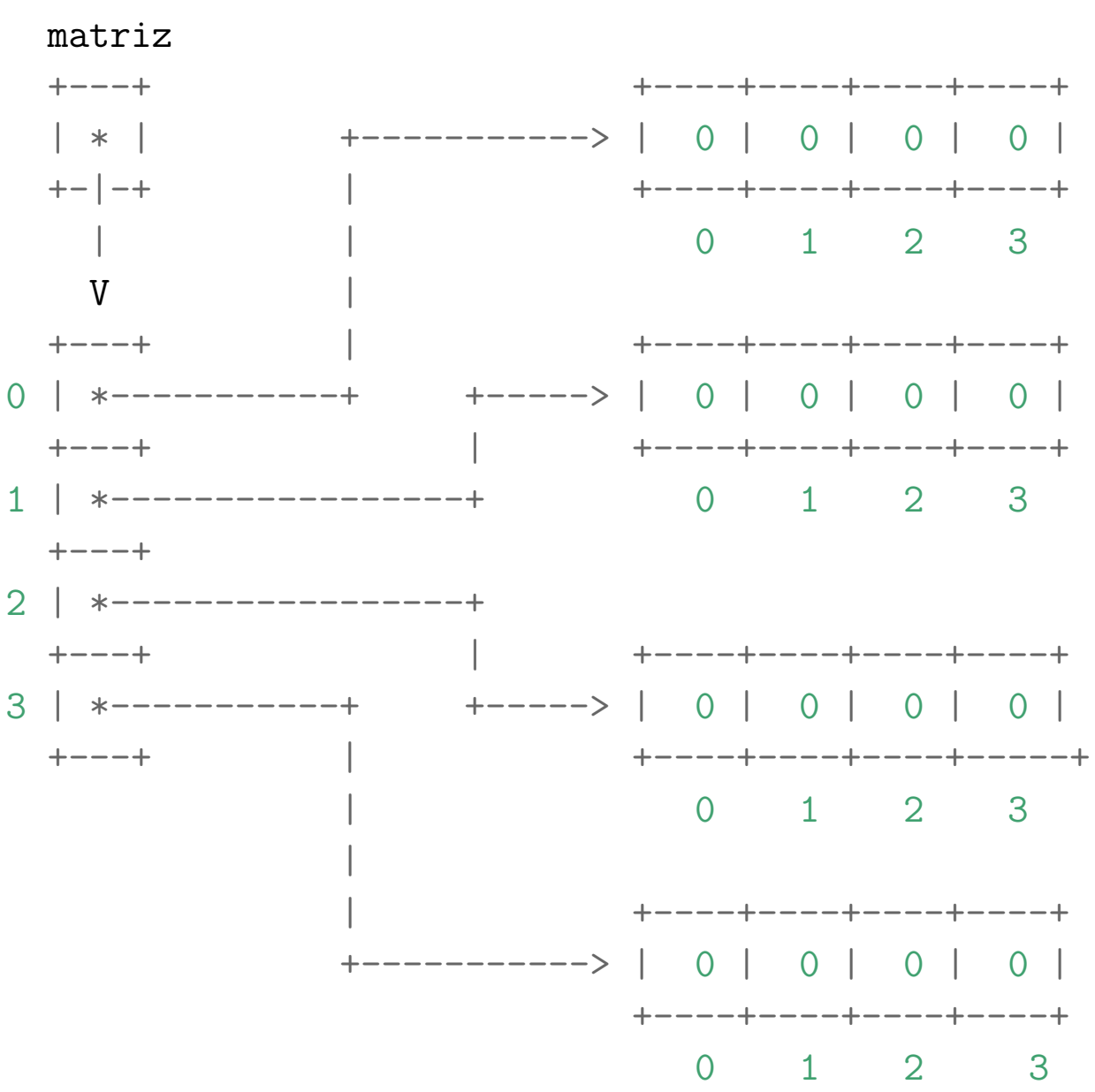




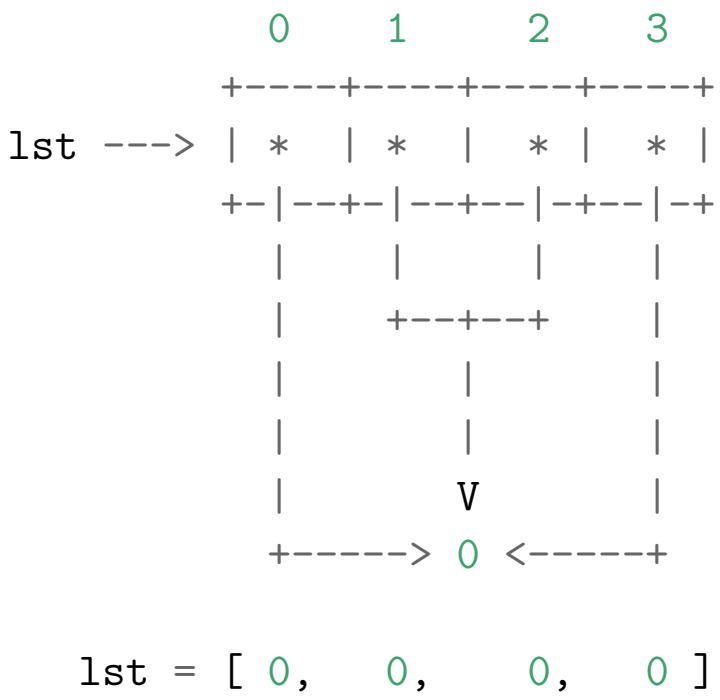


27.7 Vale a pena ver de novo...

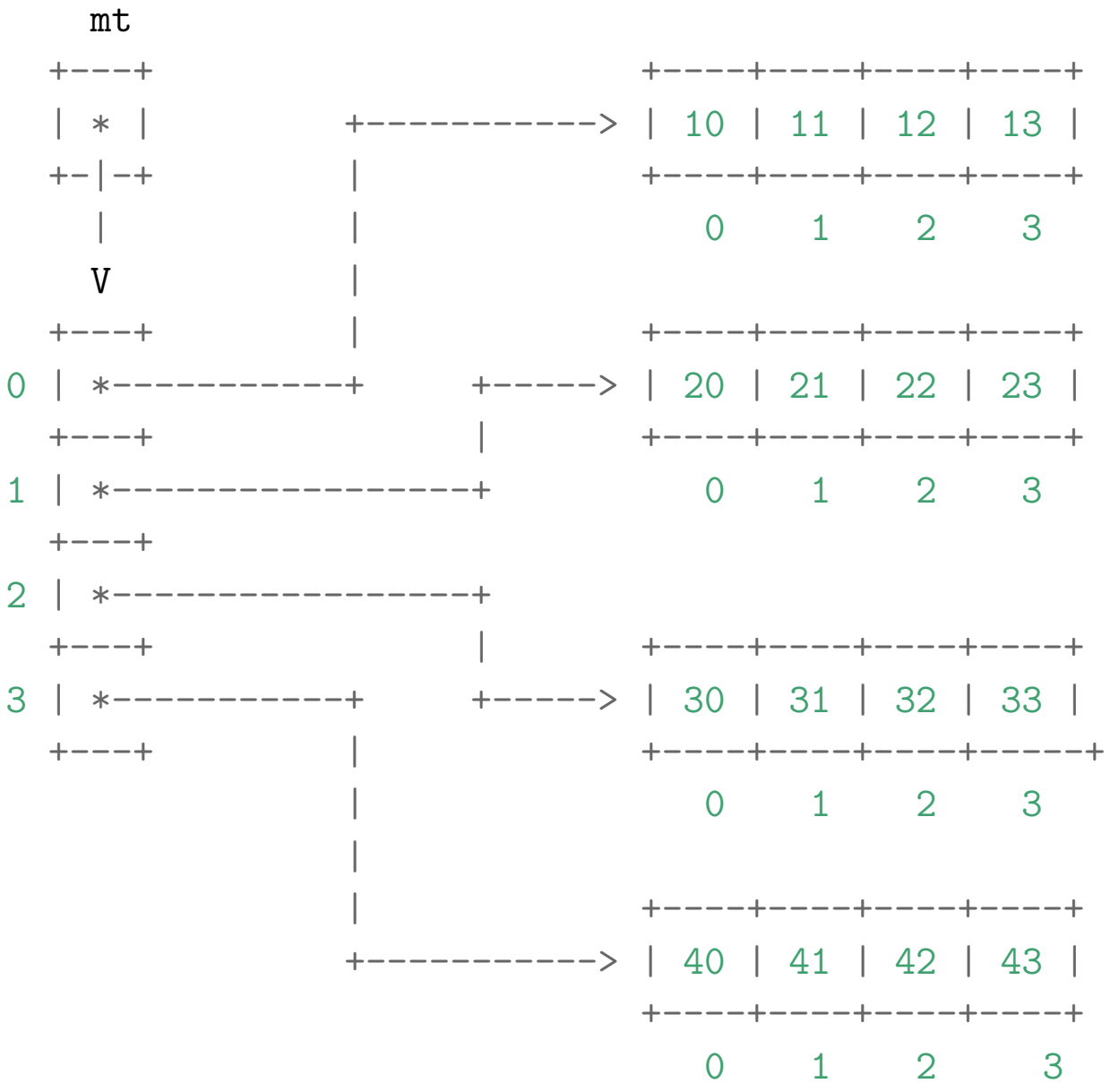
```
init_matriz(4, 4, 0)
```



Vale a pena ver de novo



Vale a pena ver de novo



27.8 MAC0110+MAC0122+...

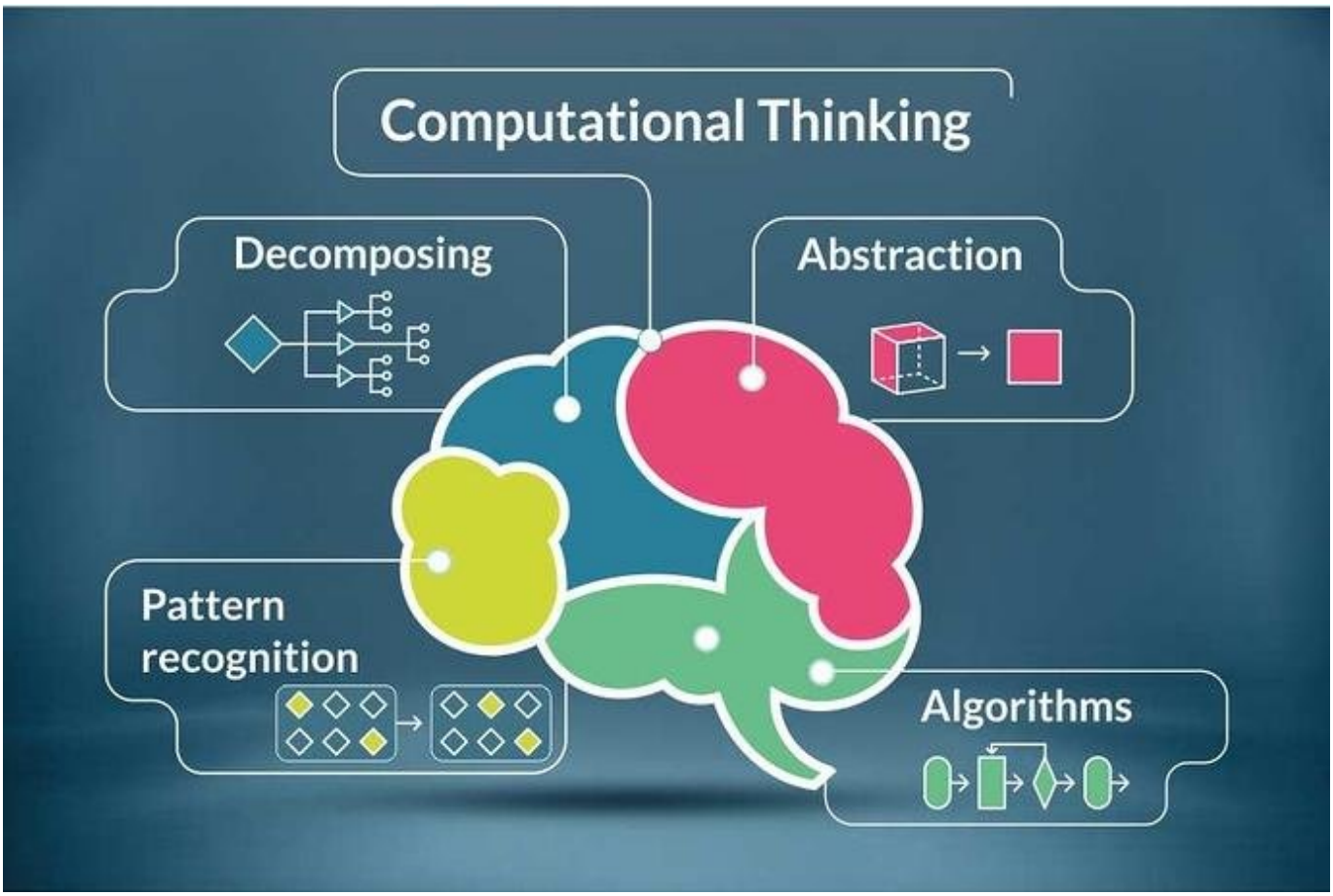


Figure 7: Fonte: https://www.researchgate.net

27.9 MAC0110+MAC0122+...

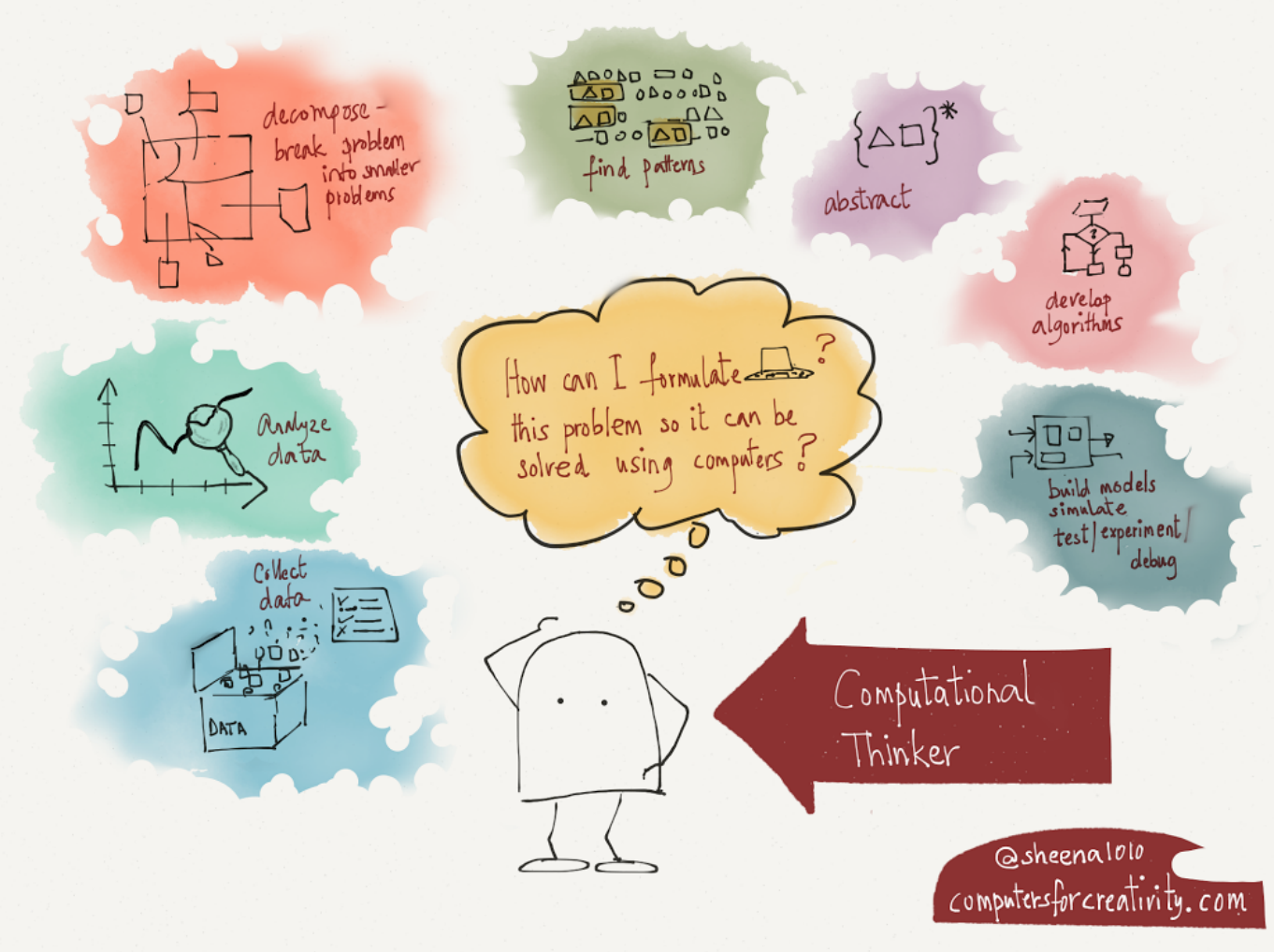


Figure 8: Raciocínio


IT'S OVER
IT'S DONE
quickmeme.com