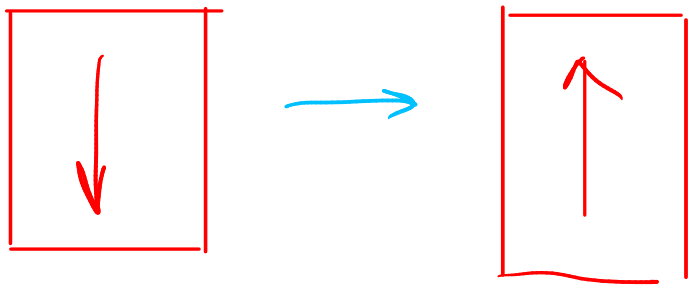
30 Reunião 30: 10/DEZ/2020

Figure 1: http://www.supercoloring.com/

30.1 Reunião passada

Mais funções para nossa **biblioteca para manipulação de matrizes** `matriz.py`:

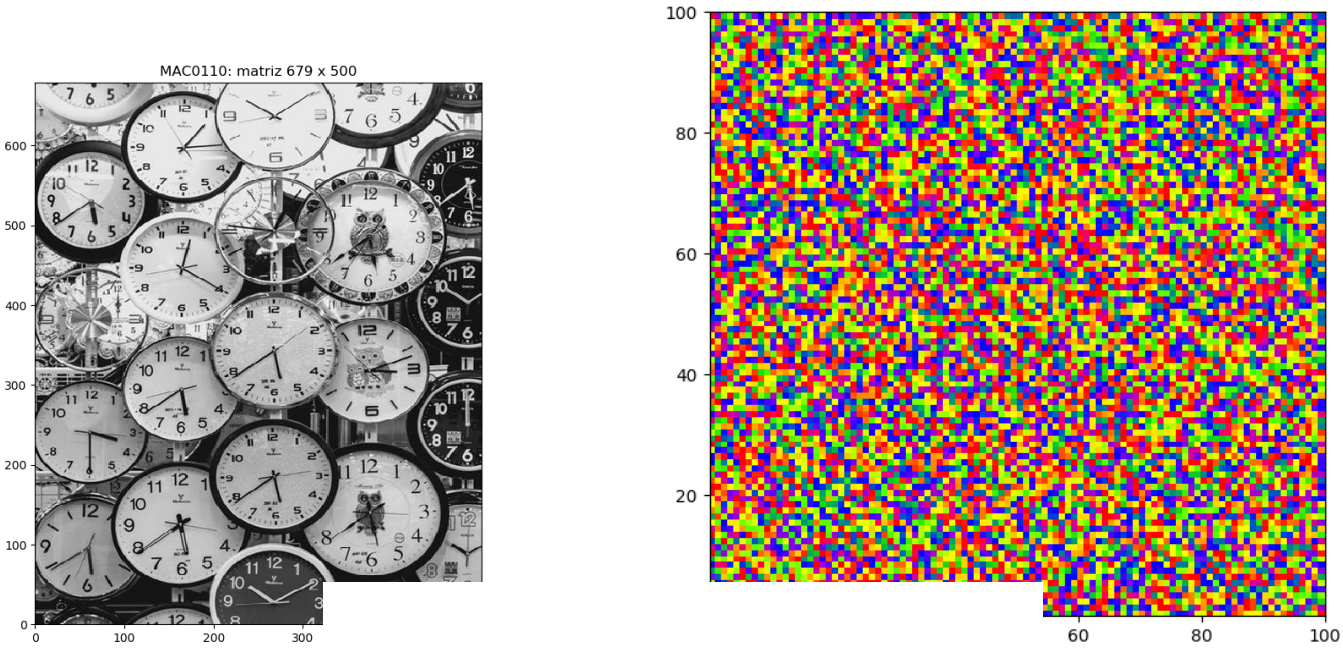
- `leia_matriz()`: invólocro para `leia_matriz_teclado()` e
- `leia_matriz_arquivo()`
- `leia_matriz_teclado()`: para leitura de matrizes através de digitação no teclado
- ~~`leia_matriz_arquivo()`~~: para leitura de matrizes de arquivos
- `gire_horizontal()`: retorna a matriz dada de "ponta-cabeça".
- sobre leitura de arquivos veja os rabiscos da reunião passada e a página https://panda.ime.usp.br/pensepy/static/pensepy/10-Arquivos/files.html



30.2 Hoje

- `rode_direita(mt)`: retorna uma matriz dada rotacionada para a direita
- `grave_matriz(mt, nome_arq)`: grava `matriz` no arquivo `nome_arq`
- `prod(mtA, mtB)`: retorna o produto da `mtA` por `mtB`

30.3 Matrizes



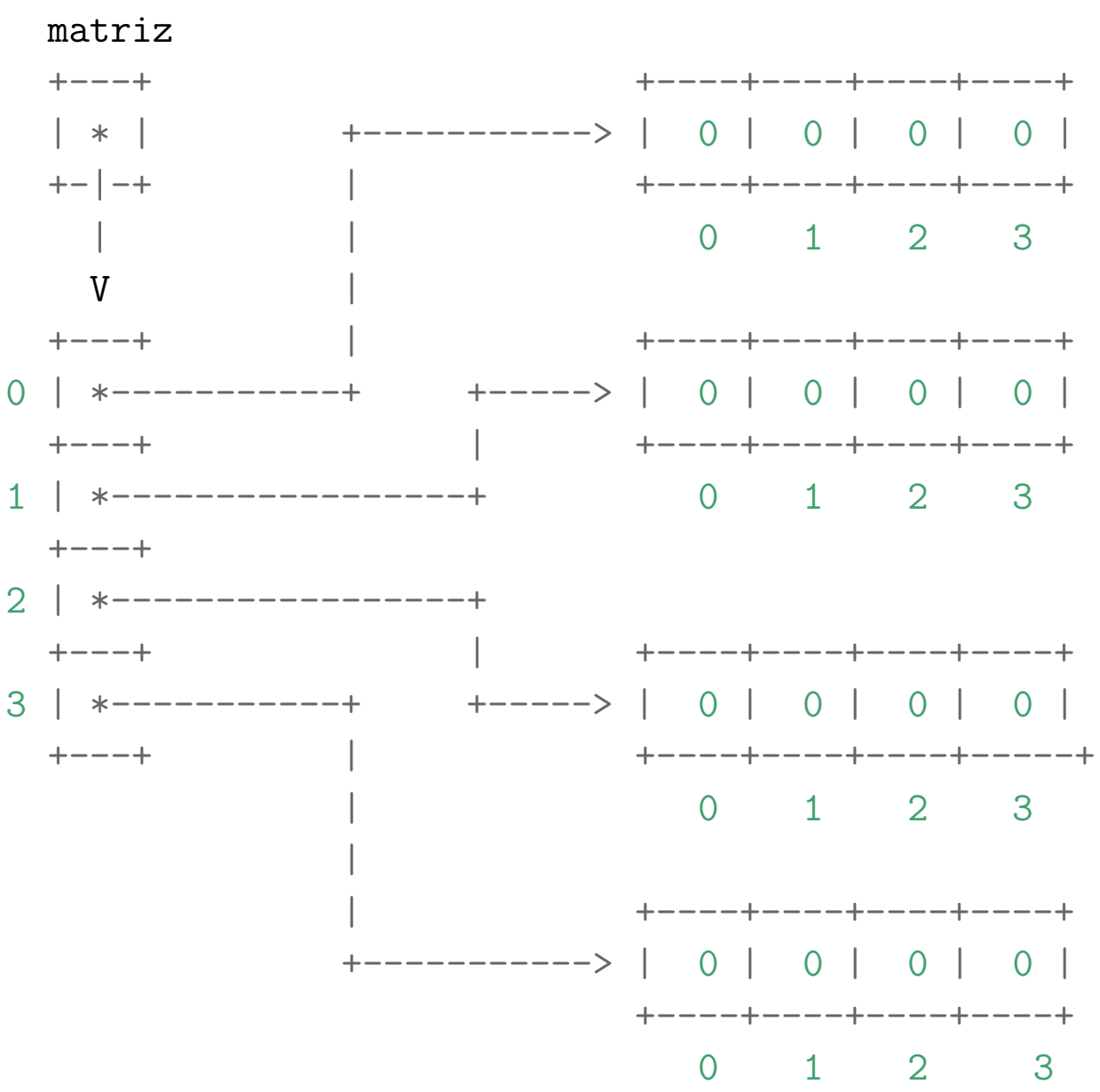
30.4 Manipulação de matrizes

Nossa biblioteca para manipulação de matrizes:

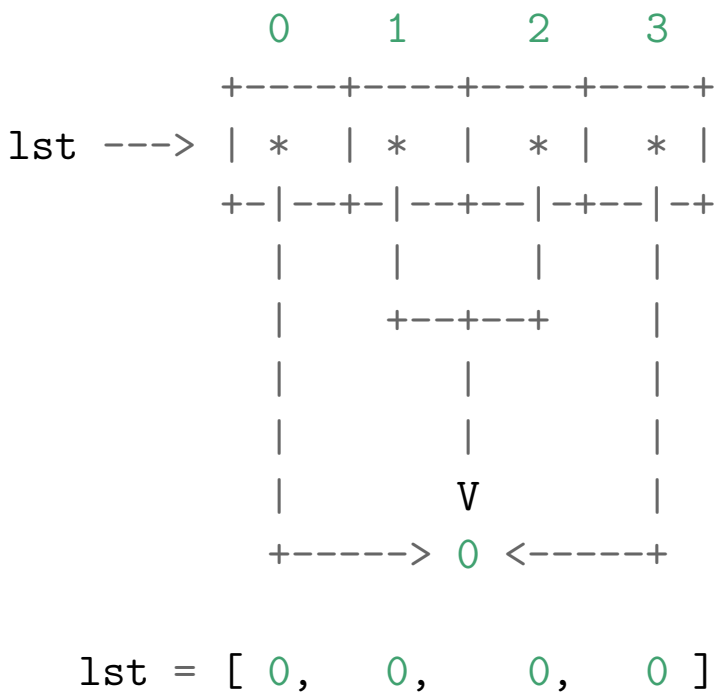
- `init_matriz(nlins, ncols, val=0)`: cria uma matriz de dimensão `nlins` × `ncols` `# REUNIÃO 28`
- `to_str(mt)`: string para ser usada por `print()` para exibir a `mt` `# REUNIÃO 28`
- `exiba_matriz(mt):` exibe `mt` na tela `# FEITA`
- `grave_matriz(mt, nome_arq)`: grama `mt` em um arquivo '# HOJE
- `exiba_imagem(mt)`: mostra `mt` como imagem `# FEITO`
- `simetrica(mt)`: verifica se `mt` é simetrica `# REUNIÃO 27`
- `linhas_val(mt, val)`: linhas com todos valores iguais a `val` `# REUNIÃO 27`
- `colunas_val(mt, val)`: linhas com todos valores iguais a `val` `# REUNIÃO 27`
- `diagonais(mt, val)`: diagonais têm apenas valor `val` `# REUNIÃO 27`
- `prod(mtA, mtB)`: retorna o produto de `mtA` por `mtB` '# HOJE
- `leia_matriz(None)`: leitura de matriz `# REUNIÃO 29`
- `leia_matriz_teclado()`: lê matriz do teclado `# REUNIÃO 29`
- `leia_matriz_arquivo(nome_arq)`: lê matriz de arquivo `# REUNIÃO 29`
- `rode_dir(mt)`: roda matriz para direita `# HOJE`
- `rode_esq(mt)`: roda matriz para esquerda `# HOJE`
- `gire_horizontal(mt)`: reflete `mt` horizontamente `# REUNIÃO 29`
- `give_vertical(mt)`: reflete `mt`verticalmente `# HOJE`

30.5 init_matriz()

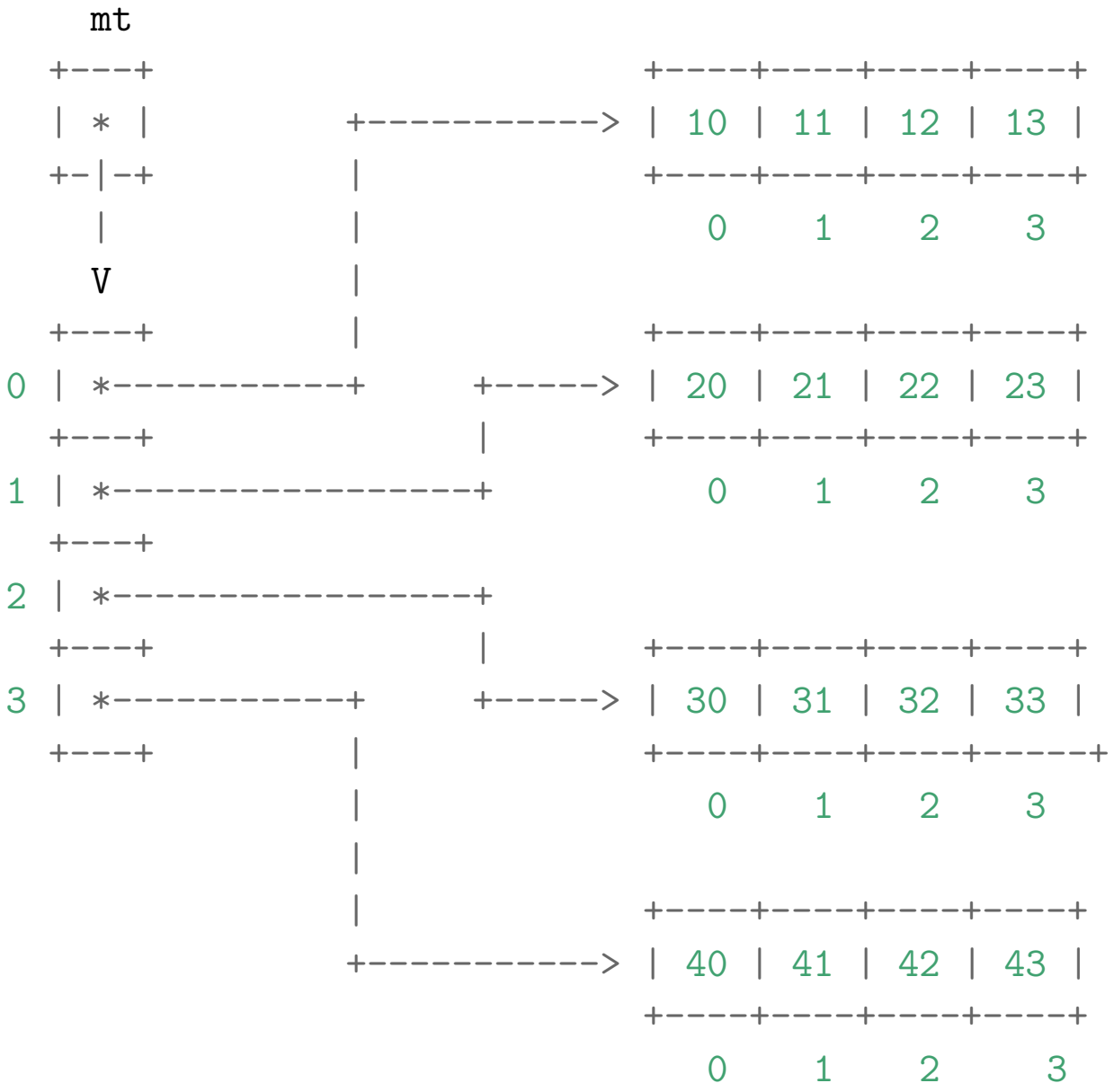
```
init_matriz(4, 4, 0)
```



Rascunhos



Rascunhos

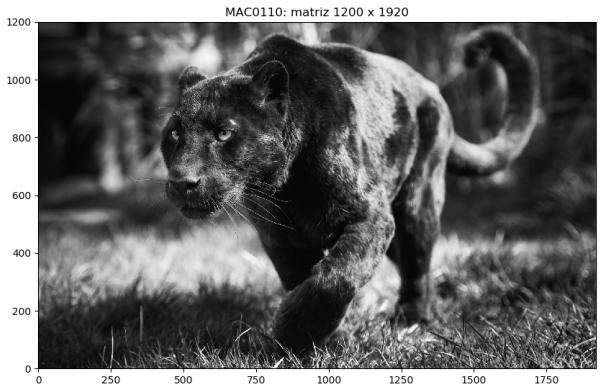

mt
0
1
2
3
10 11 12 13
0 1 2 3
20 21 22 23
0 1 2 3
30 31 32 33
0 1 2 3
40 41 42 43
0 1 2 3

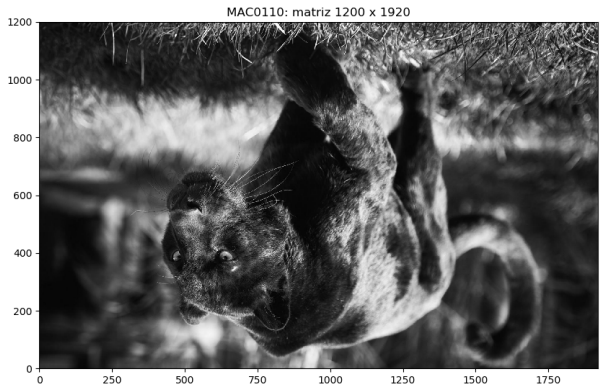
30.6 gire_horizontal()

	0	1	2	3
0	a	b	c	d
1	e	f	g	h
2	i	j	k	l
3	m	n	o	p
4	q	r	s	t

flipH() ----->

	0	1	2	3
0	q	r	s	t
1	m	n	o	p
2	i	j	k	l
3	e	f	g	h
4	a	b	c	d



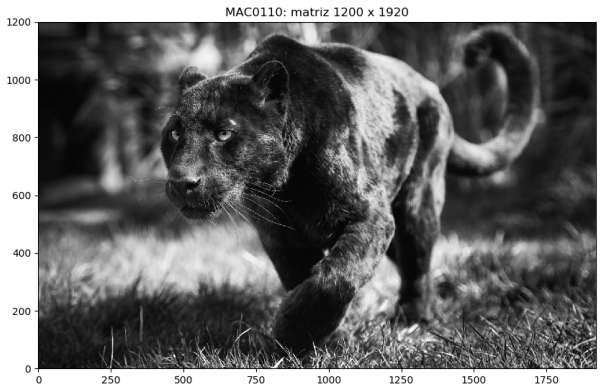


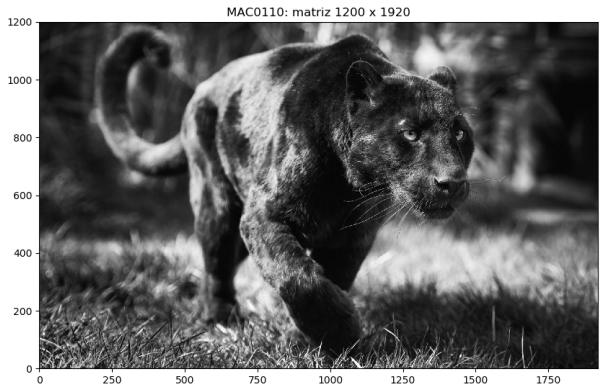
Solução

```
def gire_horizontal(mt):
    '''(matriz) -> matriz

    RECEBE uma matriz `mt`.
    RETORNA a `mt` refletida horizontalmente (cima-baixo).
    Essa função NÃO é mutadora.
    '''
    nlins = len(mt)
    ncols = len(mt[0])
    horizontal = init_matriz(nlins, ncols)
    for i in range(nlins):
        for j in range(ncols):
            horizontal[i][j] = mt[nlins-i-1][j]
    return horizontal
```

30.7 gire_vertical()

```
    0   1   2   3                       0   1   2   3
  +---+---+---+---+                   +---+---+---+---+
0 | a | b | c | d |                 0 | d | c | b | a |
  +---+---+---+---+                   +---+---+---+---+
1 | e | f | g | h |                 1 | h | g | f | e |
  +---+---+---+---+     flipV()      +---+---+---+---+
2 | i | j | k | l |     ----->      2 | l | k | j | i |
  +---+---+---+---+                   +---+---+---+---+
3 | m | n | o | p |                 3 | p | o | n | m |
  +---+---+---+---+                   +---+---+---+---+
4 | q | r | s | t |                 4 | t | s | r | q |
  +---+---+---+---+                   +---+---+---+---+
```





Solução

```
def gire_vertical(mt):
    '''(matriz) -> matriz

    RECEBE uma matriz `mt`.
    RETORNA a `mt` refletida verticalmente (esquerda-direita).
    Essa função NÃO é mutadora.
    '''
    nlins = len(mt)
    ncols = len(mt[0])
    vertical = init_matriz(nlins, ncols)
    for i in range(nlins):
        for j in range(ncols):
            vertical[i][j] = mt[i][ncols-j-1]
    return vertical
```

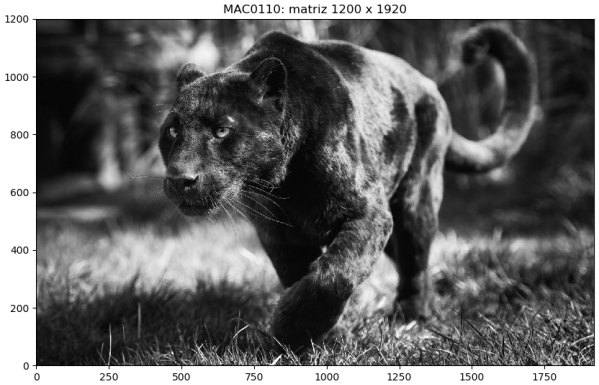
30.8 rode_direita()

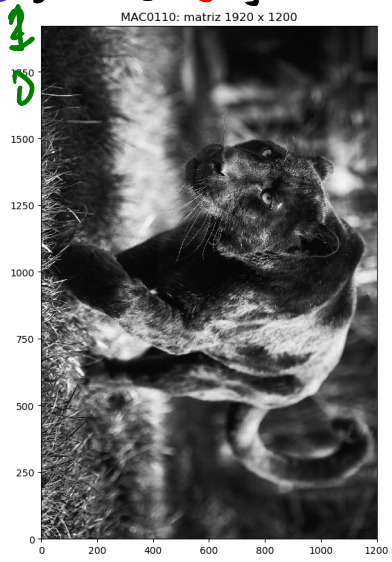
```
   0   1   2   3                    0   1   2   3
 +---+---+---+---+                +---+---+---+---+---+
0| a | b | c | d |               0| q | m | i | e | a |
 +---+---+---+---+                +---+---+---+---+---+
1| e | f | g | h |               1| r | n | j | f | b |
 +---+---+---+---+   rotateR()    +---+---+---+---+---+
2| i | j | k | l |    ----->     2| s | o | k | g | c |
 +---+---+---+---+                +---+---+---+---+---+
3| m | n | o | p |               3| t | p | l | h | d |
 +---+---+---+---+                +---+---+---+---+---+
4| q | r | s | t |
 +---+---+---+---+
```

```
dr = init_matriz(ncols, nlins)
```

```
for i in range(4):
    for j in range(5):
        dr[][j] = mt[???][???]
```

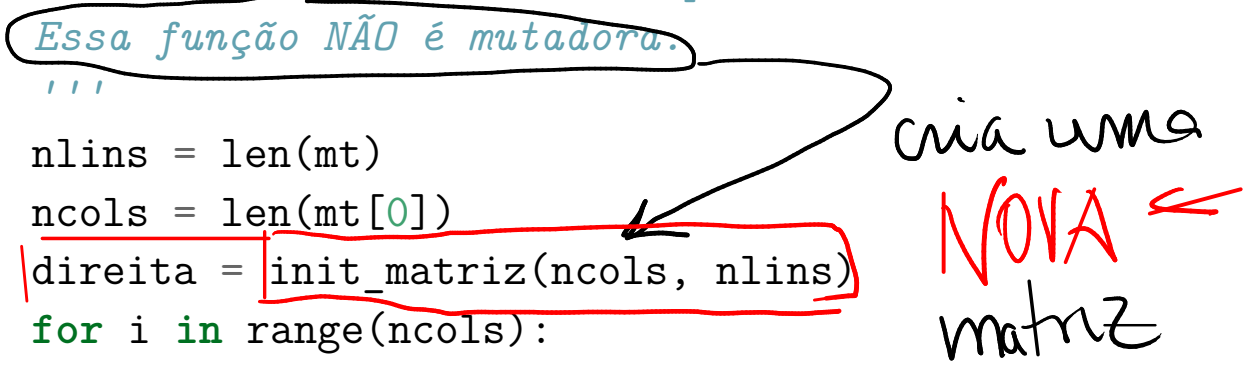
```
dr[0][0] = mt[4][0]
dr[0][1] = mt[3][0]
dr[0][2] = mt[2][0]
dr[0][3] = mt[1][0]
dr[0][4] = mt[0][0]
```





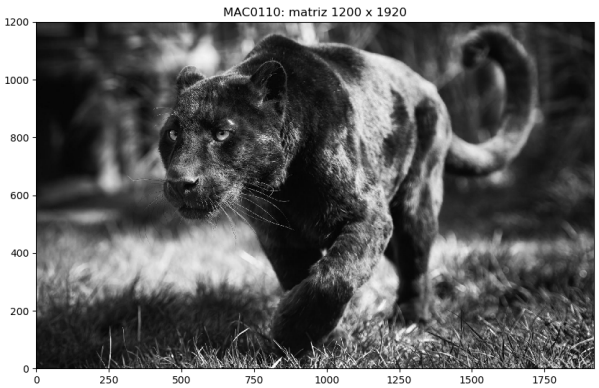
Solução

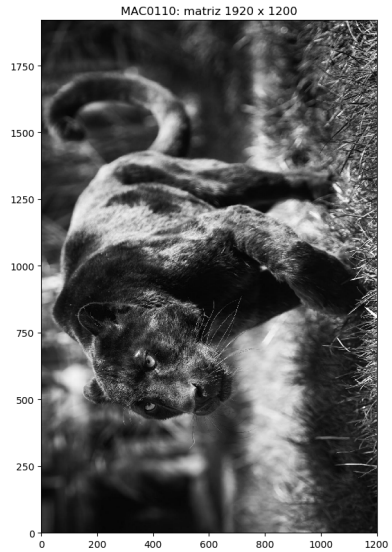
```
def rode_dir(mt):
    '''(matriz) -> matriz

    RECEBE uma matriz `mt`.
    RETORNA a `mt` rotacionada para a direita (horário)
    Essa função NÃO é mutadora.
    '''
    nlins = len(mt)
    ncols = len(mt[0])
    direita = init_matriz(ncols, nlins)
    for i in range(ncols):
        for j in range(nlins):
            direita[i][j] = mt[nlins-j-1][i]
    return direita
```



30.9 rode_esquerda()

```
    0   1   2   3                          0   1   2   3
  +---+---+---+---+                      +---+---+---+---+---+
0 | a | b | c | d |                    0 | d | h | l | p | t |
  +---+---+---+---+                      +---+---+---+---+---+
1 | e | f | g | h |                    1 | c | g | k | o | s |
  +---+---+---+---+    rotateL()         +---+---+---+---+---+
2 | i | j | k | l |     ----->         2 | b | f | j | n | r |
  +---+---+---+---+                      +---+---+---+---+---+
3 | m | n | o | p |                    3 | a | e | i | m | q |
  +---+---+---+---+                      +---+---+---+---+---+
4 | q | r | s | t |
  +---+---+---+---+
```





30.9.1 Solução

```
def rode_esq(mt):
    '''(matriz) -> matriz

    RECEBE uma matriz `mt`.
    RETORNA a `mt` rotacionada para a esquerda (anti-horário).
    Essa função NÃO é mutadora.
    '''
    nlins = len(mt)
    ncols = len(mt[0])
    esquerda = init_matriz(ncols, nlins)
    for i in range(ncols):
        for j in range(nlins):
            esquerda[i][j] = mt[j][ncols-i-1]
    return esquerda
```

30.10 Exercício: produto de matrizes

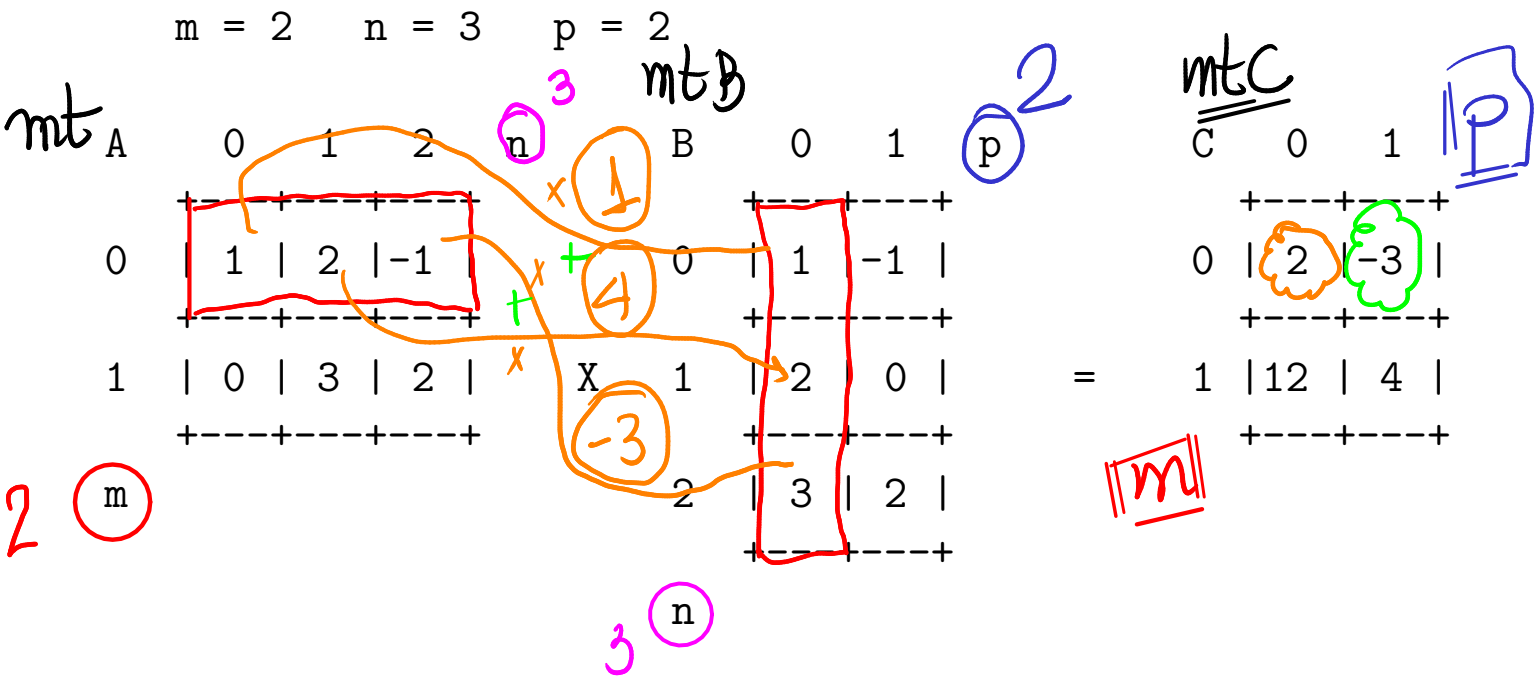
Neste exercícios escreveremos mais funções que manipulam matrizes para a nossa bibliteca `matriz.py`:

- `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 $\mathtt{A}_{\mathtt{m}\times\mathtt{n}}$ e $\mathtt{B}_{\mathtt{n}\times\mathtt{p}}$ e calcula e retorna a matriz $\mathtt{C}_{\mathtt{m}\times\mathtt{p}}$ que é o produto de `A` por `B`.

Exemplos

$\mathtt{A}_{2\times3} \times \mathtt{B}_{3\times2} = \mathtt{C}_{2\times2}$




```
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
```

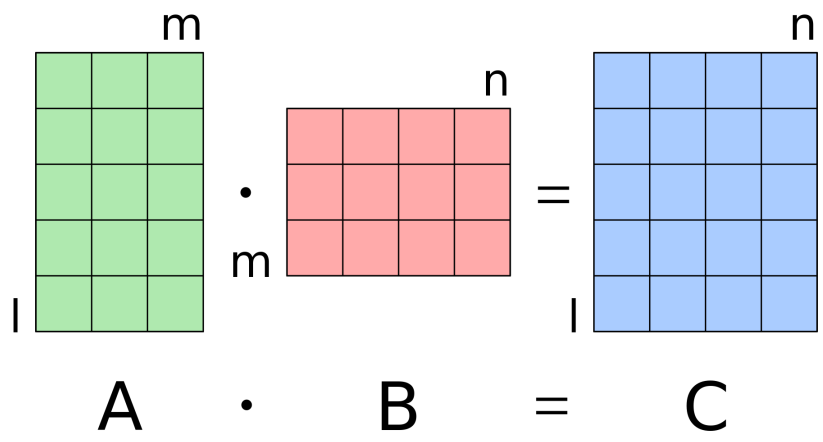


Figure 3: Dimensões do produto de matrizes (Fonte: Wikipedia)

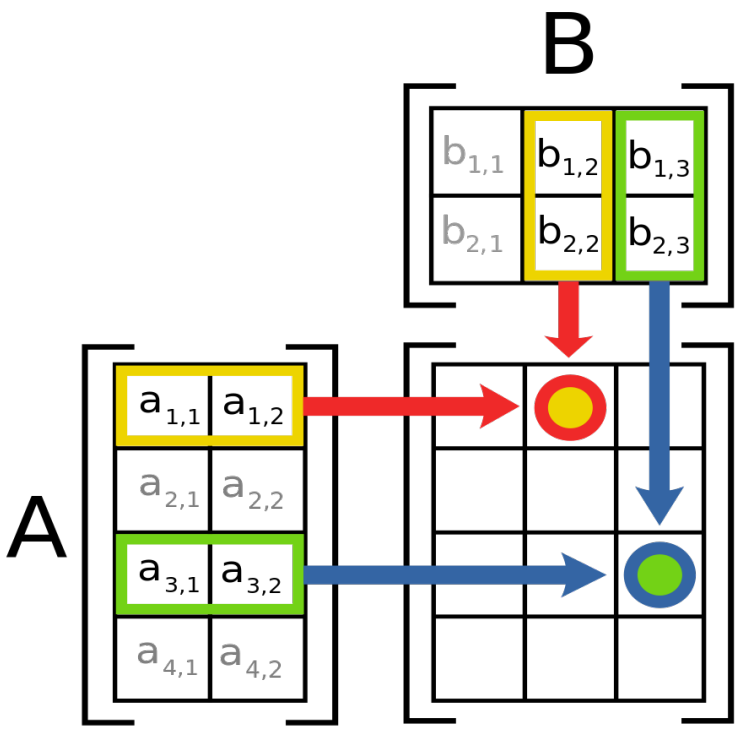
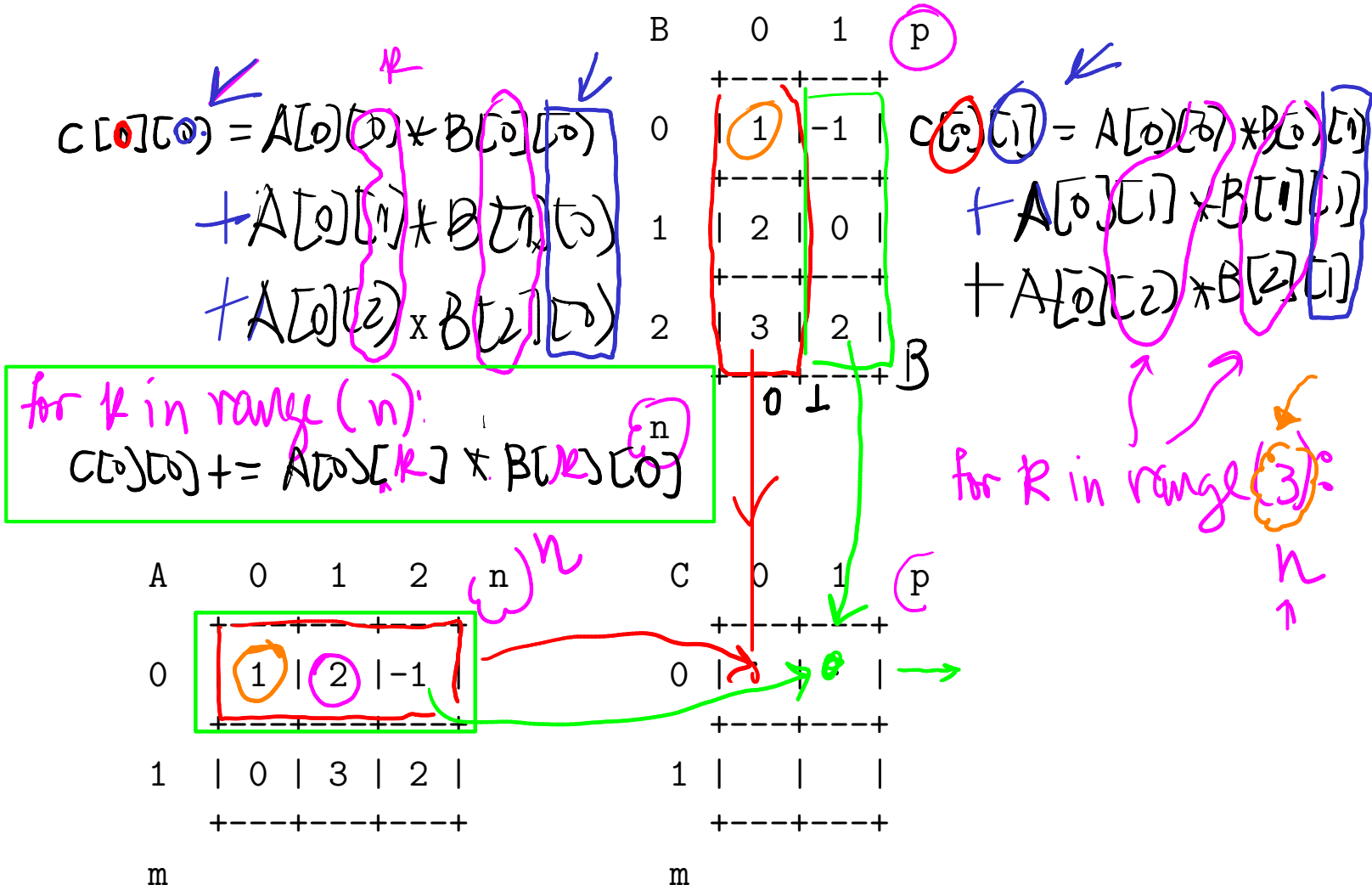
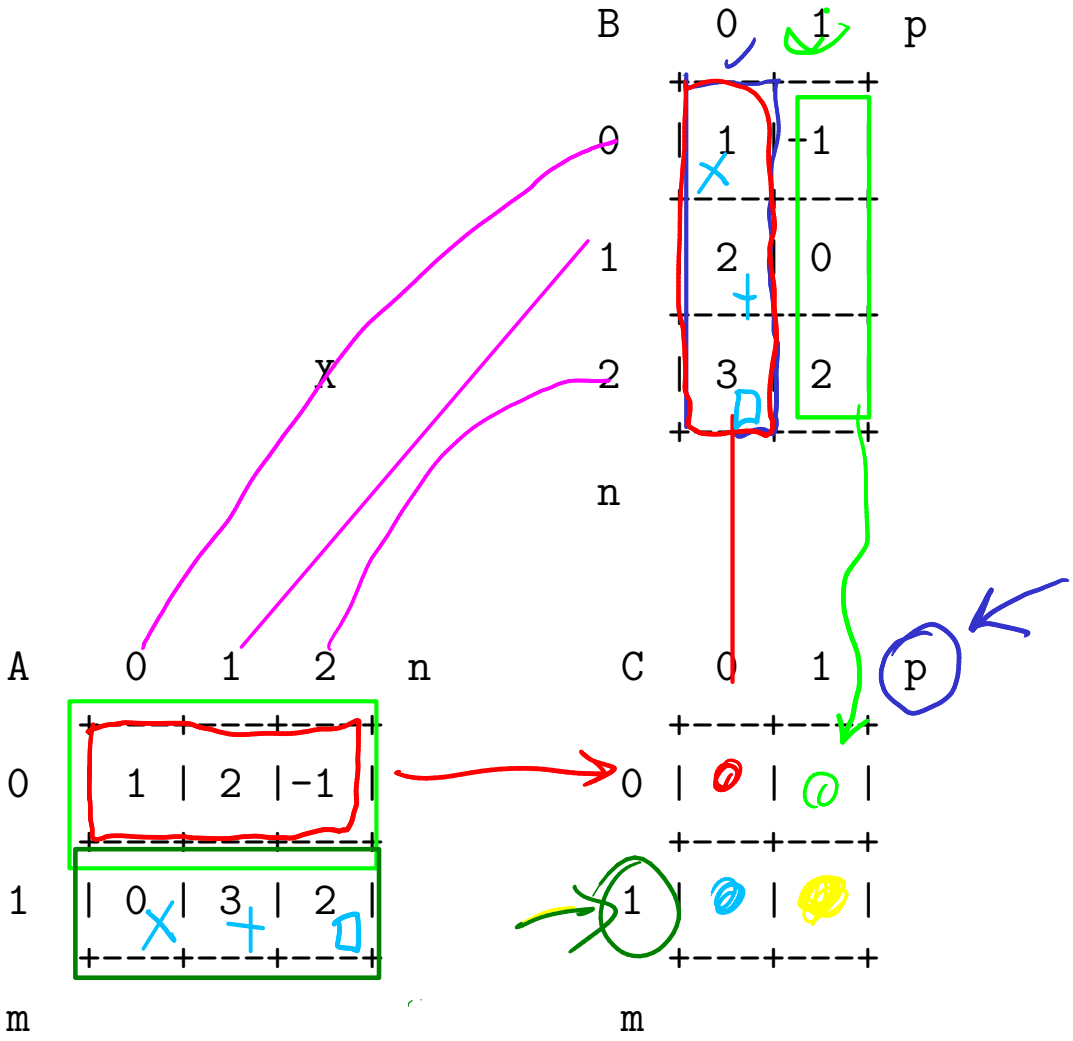


Figure 4: Diagrama da multiplicação de matrizes (Fonte: Wikipedia)

Rascunho

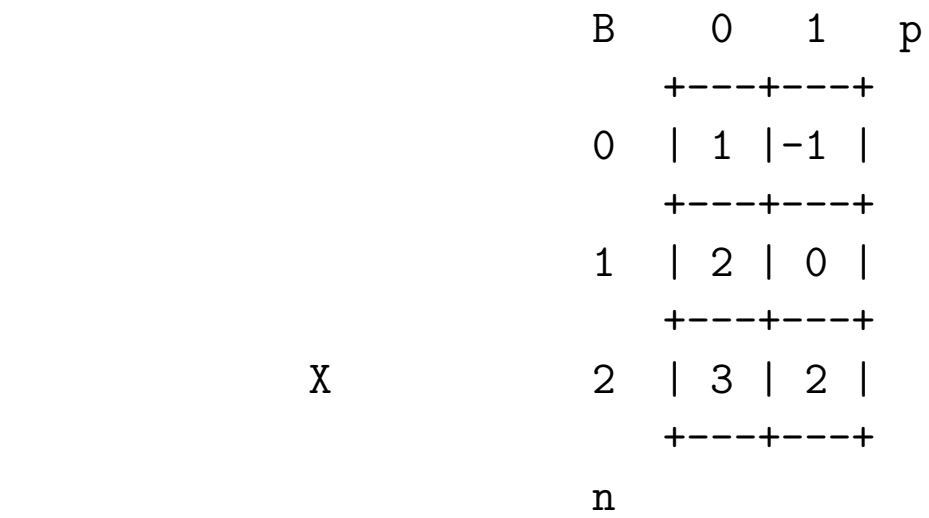

C[0][0] = A[0][0]*B[0][0]
+A[0][1]*B[1][0]
+A[0][2]*B[2][0]
for k in range(n):
C[0][0] += A[0][k] * B[k][0]
C[0][1] = A[0][0]*B[0][1]
+A[0][1]*B[1][1]
+A[0][2]*B[2][1]
for k in range(3):
B
A
C
n
p
m

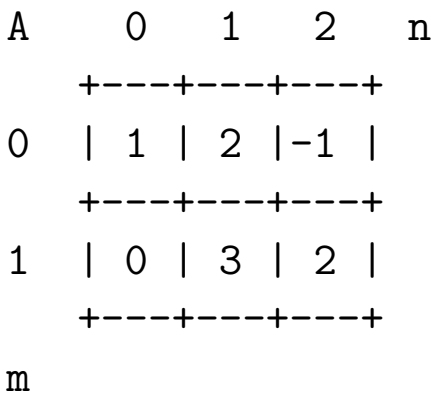
Rascunho

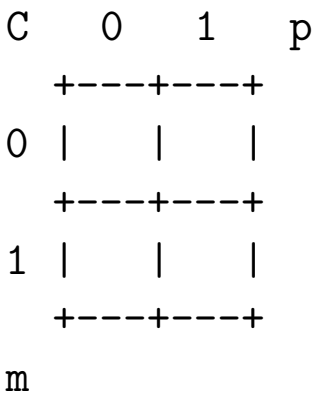



```
for j in range(p):
    for k in range(n):
        C[0][j] += A[0][k] * A[k][j]
```

Rascunho







Solução

```
#------------------------------------------------
def prod(mtA, mtB):
    '''(matriz, matriz) -> matriz

    RECEBE matrizes `mtA` e `mtB`.
    RETORNA a matriz que é o produto de `mtA` e `mtB`.

    Pré-condição: supõe que as dimensões das matrizes são
    compatíveis.
    '''
    nlinsA = len(mtA)
    ncolsA = nlinsB = len(mtA[0])
    ncolsB = len(mtB[0])

    # crie uma matriz para armazenar o produdot
    mtC = crie_matriz(nlinsA, ncolsB)

    # preencha mtC com o produto
    for i in range(nlinsA):
        # determine a linha i do produto
        for j in range(ncolsB):
            # determine o elemento [i][j]
            for k in range(ncolsA):
                mtC[i][j] += mtA[i][k] * mtB[k][j]

    # retorna a matriz com o produto
    return mtC
```

30.10.1 grave_matriz()

```
#-----------------------------------------------------------------
def grave_matriz(mt, nome_arq = "matriz.txt"):
    '''(matriz, str) -> None

    RECEBE uma matriz `mt` e uma string `nome_arq`
    GRAVA `mt` em um arquivo de nome `nome_arq`
    '''
    # 1 abra o arquivo
    arq = open(nome_arq, 'w', encoding='utf-8')
    # pegue nlins e ncols
    nlins = len(mt)
    ncols = len(mt[0])
    arq.write(f"{nlins} {ncols}\n")
    for i in range(nlins):
        for j in range(ncols):
            arq.write(f"{mt[i][j]} ")
        arq.write("\n")
    # feche o arquivo
    arq.close()
```

30.11 Exercício: campo minado

Escreva um programa que leia uma matriz de de 0's (posições *livres*) e -1's (*minas*) e imprime a quantida de minas ao redor de cada posição livre da matriz.

Exemplo

Veja aqui um esqueleto de `minas.py` além do arquivo `minas.txt`.

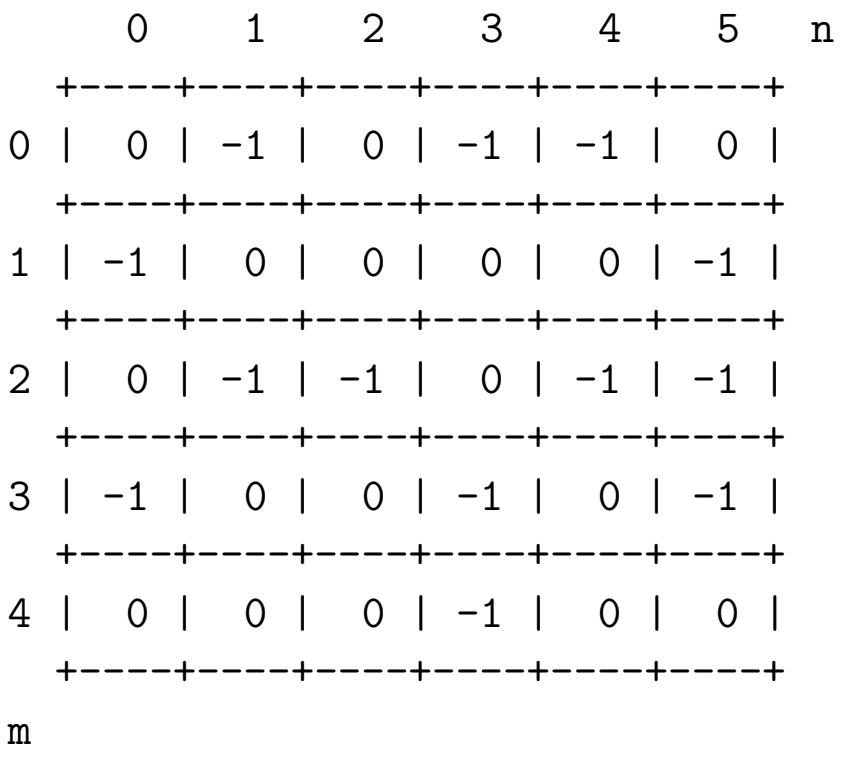
```
Campo minado 5 x 6:
Matriz: 5 x 6
  0 -1  0 -1 -1  0
 -1  0  0  0  0 -1
  0 -1 -1  0 -1 -1
 -1  0  0 -1  0 -1
  0  0  0 -1  0  0

Posicao livre   no. de minas
-------------   -----------
   [0][0]            2
   [0][2]            2
   [0][5]            2
   [1][1]            4
   [1][2]            4
   [1][3]            4
   [1][4]            5
   [2][0]            3
   [2][3]            3
   [3][1]            3
   [3][2]            4
   [3][4]            5
   [4][0]            1
   [4][1]            1
   [4][2]            2
```

```
[4] [4]                 3
[4] [5]                 1
```

Rascunhos

Figure 5: Minesweeper (Fonte: Wikipedia

Rascunhos



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

Solução

```
# mt.leia_matriz(), mt.exiba_matriz()
import matriz as mt

# CONSTANTES
MINA  = -1
LIVRE =  0


#-----------------------------------------------------------------
def main():
    '''
    Programa que lê uma matriz de dimensão nxm de 0's (posições
    livres) e -1's (minas) e exibe a quantida de minas ao redor
    de cada posição livre da matriz.
    '''
    # 1. leia o campo minado do arquivo minas.txt
    campo = mt.leia_matriz("minas.txt")

    # 2. pegue a dimensão do campo
    m = len(campo)
    n = len(campo[0])
    print(f"\nCampo minado {m} x {n}:")

    # 3. exiba a matriz do campo
    mt.exiba_matriz(campo)

    # 4. pare para admirarmos o campo
    mt.pause()

    # 4. exiba o número de posições livres ao redor de cada posi
    print("Posicao livre   no. de minas")
```

```
    print("-------------   ------------")
    for i in range(m):
        for j in range(n):
            # a posição [i][j] é livre
            if campo[i][j] == LIVRE:
                # conte o número de posições livres
                num = no_minas(campo,i,j)
                # exiba o número de posições livres
                print(f"   [{i}][{j}]             {num}")
```

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

    RECEBE uma matriz `campo` de 0s e -1s e uma posição [lin][co
    RETORNA o número de posições ao redor de [lin][col] que cont
    o valor MINA.

    Pré-condição: supõe que a posição [LIN][COL] tem o valor LIV
    '''
    # contador de minas
    no_minas = 0

    # dimensão do campo
    nlins = len(campo)
    ncols = 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 and i < nlins and 0 <= j and j < ncols:
            # conte se tem mina
            if campo[i][j] == MINA:
                no_minas += 1

return no_minas
```

Figure 6: Placa elevador

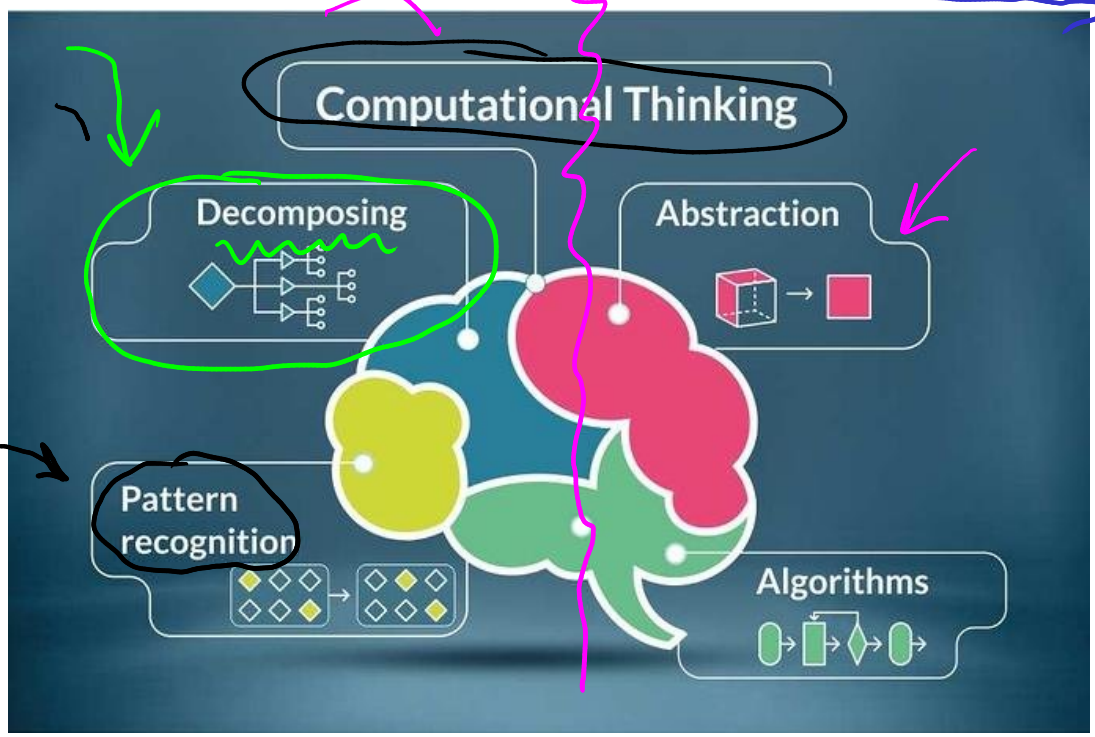


Figure 7: Fonte: https://www.researchgate.net

30.13 MAC0110

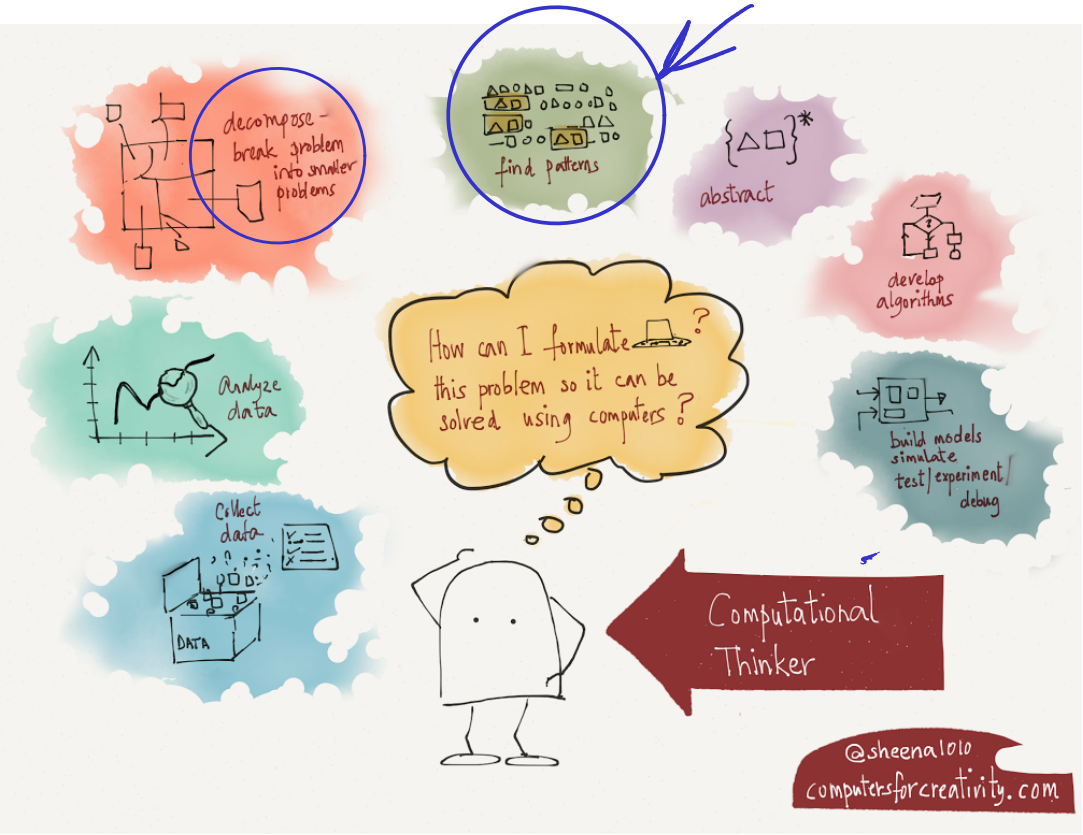


Figure 8: Raciocínio

Reunião

próximo

tenço