Terceiro Exercício-Programa: Vírus Ge0m3tRik4

Entrega: 23/6/2018

Motivação

Um dos professores da disciplina teve seu computador infectado por um vírus de computador após abrir um arquivo executável que estava anexado a um email intitulado "Pedido de revisão da nota do EP2", supostamente enviado por um estudante do curso. O vírus afetou apenas arquivos de imagens, tornando-as irreconhecíveis. As figuras abaixo mostram um exemplo de imagem antes e após ser afetada pelo vírus.

imagem original

imagem deformada pelo vírus

Estudando cuidadosamente as imagens deformadas, o professor percebeu que o vírus aplicou uma transformação geométrica que permutava os pixeis da imagem. Tomando a imagem deformada acima como exemplo, ele conseguiu, após muitas tentativas e erros, descobrir uma sequência de transformações geométricas afins que restaura a imagem original. No entanto, para azar do professor, o vírus utilizou uma transformação diferente em cada imagem. O professor então ponderou que encontrar uma sequência de transformações para cada imagem manualmente levaria muito tempo, e decidiu pedir ajuda aos brilhantes estudantes de MAC2166! Felizmente, as imagens no computador do professor possuíam arquivos contendo um código de verificação da sua integridade, e estes não foram afetados pelo vírus. Dessa forma, é possível verificar se uma sequência de transformações resulta na imagem original de maneira automática.

Objetivo

A sua tarefa neste EP é desenvolver um programa que restaura automaticamente as imagens deformadas pelo vírus através da aplicação de uma sequência de transformações afins na geometria da imagem.

Conhecimentos envolvidos

Além dos conhecimentos de fundamentos de programação associados, ao solucionar esse EP você vai aprender (ou reforçar seu aprendizado) sobre:

- Funções de dispersão (*hash*) de arquivos, em particular, a função Adler32;
- Transformações geométricas;
- Aritmética modular; e
- Algoritmos de busca sistemática.

Organização do EP

O seu código deve ser escrito de forma estruturada, ou seja, divido em funções que realizam funcionalidades distintas. Para isso, você deve basear sua solução no esqueleto de código fornecido. Você pode criar novas funções, mas não deve modificar o protótipo das funções existentes.

As diferentes funcionalidades estão agrupadas em 5 partes e descritas a seguir. Recomenda-se que você desenvolva as partes na ordem apresentada aqui.

Parte 1: Cálculo de identificador de matrizes através da função `calcula_id`;

Parte 2: Manipulação de arquivos de entrada e saída de dados:

1. Leitura de identificador de arquivo através da função `carrega_identificador`;
2. Leitura de imagem pela função `carrega_imagem`;
3. Leitura de transformações pela função `carrega_transformação`;
4. Escrita de imagem pela função `salva_imagem`;

Parte 3: Transformação afim de matriz pela função `transforma`;

Parte 4: Busca em profundidade pela sequência de transformações que gera a imagem original feita pela função `busca`;

Parte 5: Escrever interface do programa, que lê nomes de arquivos digitados pelo usuário, realiza a busca e salva a imagem correspondente (ou avisa falha).

Para auxiliar no desenvolvimento do seu programa, disponibilizamos junto com o esqueleto do código alguns programas de teste que detectam e apontam eventuais problemas no seu código. Esses programas não servem de critério para a avaliação do seu EP. Se seu programa não passa nos testes, ele certamente está incorreto. No entanto, é possível que seu programa passe em todos os testes e ainda contenha falhas (pois o teste não verifica todos os casos possíveis).

⚠ Cada função no esqueleto está inicialmente implementada com o seguinte comando:

```
raise NotImplementedError
```

Esse comando visa informar ao programa de teste que a função não foi implementada. A primeira coisa ao escrever sua implementação é remover a linha acima.

Gerando os arquivos base

Para esse EP você deve ter recebido um único arquivo compactado denominado `EP3.zip`. Ao descompactar esse arquivo, você criará uma pasta chamada `EP3` contendo dois arquivos de nomes `data.p` e `inicializa.py`. Você **não deve** alterar o conteúdo desses arquivos. O arquivo `inicializa.py` contém um programa Python que gera todos os arquivos de exemplo que usaremos e o esqueleto do seu programa. A primeira coisa a fazer é executar esse programa.

```
######################################################
# Iniciando testes: (Parte00) Teste de integridade #
######################################################
Criando 'testa_parte_01.py'... OK
Criando 'testa_parte_02.py'... OK
Criando 'testa_parte_03.py'... OK
Criando 'testa_parte_04.py'... OK
Criando 'img00.adler32'... OK
Criando 'img00.pgm'... OK
Criando 'img01.adler32'... OK
Criando 'img01.pgm'... OK
Criando 'img02.adler32'... OK
Criando 'img02.pgm'... OK
Criando 'img03.adler32'... OK
Criando 'img03.pgm'... OK
Criando 'img04.adler32'... OK
Criando 'img04.pgm'... OK
Criando 'img05.adler32'... OK
Criando 'img05.pgm'... OK
Criando 'img06.adler32'... OK
Criando 'img06.pgm'... OK
Criando 'img07.adler32'... OK
Criando 'img07.pgm'... OK
Criando 'img08.adler32'... OK
Criando 'img08.pgm'... OK
Criando 'img09.adler32'... OK
Criando 'img09.pgm'... OK
Criando 'img10.adler32'... OK
Criando 'img10.pgm'... OK
Criando 'LEIAME.txt'... OK
Criando 'transformações.txt'... OK
Criando 'duas_transformações.txt'... OK
Criando 'ep3.py'... OK
Criando 'enunciado_ep3_mac2166.pdf'... OK
Criando 'ARQUIVO.txt'... OK
######################################################
Verificando 'enunciado_ep3_mac2166.pdf'... OK
Verificando 'ep3.py'... OK
Verificando 'ARQUIVO.txt'... OK
Verificando 'testa_parte_01.py'... OK
Verificando 'testa_parte_02.py'... OK
Verificando 'testa_parte_03.py'... OK
Verificando 'testa_parte_04.py'... OK
Verificando 'transformações.txt'... OK
Verificando 'duas_transformações.txt'... OK
Verificando 'img00.adler32'... OK
Verificando 'img00.pgm'... OK
Verificando 'img01.adler32'... OK
Verificando 'img01.pgm'... OK
Verificando 'img02.adler32'... OK
Verificando 'img02.pgm'... OK
Verificando 'img03.adler32'... OK
Verificando 'img03.pgm'... OK
Verificando 'img04.adler32'... OK
Verificando 'img04.pgm'... OK
Verificando 'img05.adler32'... OK
Verificando 'img05.pgm'... OK
Verificando 'img06.adler32'... OK
Verificando 'img06.pgm'... OK
Verificando 'img07.adler32'... OK
Verificando 'img07.pgm'... OK
Verificando 'img08.adler32'... OK
Verificando 'img08.pgm'... OK
Verificando 'img09.adler32'... OK
Verificando 'img09.pgm'... OK
Verificando 'img10.adler32'... OK
Verificando 'img10.pgm'... OK
Verificando 'LEIAME.txt'... OK
######################################################
Parabéns! Seu programa passou no teste de integridade.
Em sua pasta, você agora possui todos os arquivos necessários
para execução desse EP, inclusive um chamado LEIAME.txt
com o resumo das atividades previstas para esse EP.
Você já pode começar à trabalhar na Parte 1!
######################################################
Resultados:
No teste de integridade:
Você obteve 64 acertos de um total de 64 possíveis
```

Após executar o programa, a pasta deve conter o arquivo `ep3.py` no qual você deve trabalhar assim como arquivos de exemplo que podem ser usados para testar suas implementações como os arquivos `img00.pgm` e `img00.adler32`. Uma descrição dos arquivos gerados pode ser encontrada no arquivo `ARQUIVOS.txt`. O arquivo `LEIAME.txt` contém uma descrição resumida do EP.

Parte 1: Identificação de matrizes

Quando arquivos são transmitidos, falhas podem ocorrer e causar a corrupção dos dados. Uma forma eficaz de verificar a integridade de um arquivo é gerando um código de identificação que é transmitido separadamente; ao receber o arquivo o recipiente gera novamente o código de identificação e o compara com o código recebido. Se os códigos não coincidirem o arquivo é considerado corrompido.

Uma abordagem comumente utilizada para identificação de arquivos é a utilização de uma função de dispersão ou espalhamento (em inglês, *hash function*), que associa objetos de tamanho variável a identificadores de tamanho fixo, de maneira que dois objetos distintos sejam associados a identificadores distintos *com alta probabilidade* (como o número de objetos é maior que o número de identificadores, existirão objetos com o mesmo número de identificação; funções hash são comumente projetadas para que a modificação de alguns poucos *bytes* leve a um código distinto).

Sua primeira tarefa é implementar a função `calcula_id`, que calcula um código de identificação para uma matriz M de tamanho m-por-n utilizando a seguinte versão simplificada da função de dispersão conhecida como *Adler32*:[1]

$$\mathsf{calcula\_id}(M) = B \cdot 2^{16} + A\,,$$

onde

$$\begin{aligned} A &= (1 + M_{0,0} + M_{0,1} + \cdots + M_{m-1,n-1}) \mod 65521\,, \\ B &= (1 + M_{0,0}) + (1 + M_{0,0} + M_{0,1}) + \cdots + (1 + M_{0,0} + M_{0,1} + \cdots + M_{m-1,n-1}) \mod 65521\,, \end{aligned}$$

$\mod$ representa o resto da divisão (chamado de módulo), e $M_{i,j}$ denota o elemento na i-ésima linha i e j-ésima coluna da matriz M (com linha e coluna começando em 0). Note que a função $\mod$ distribui sobre adição: $(x + y) \mod z = (x \mod z) + (y \mod z) \mod z$. Essa propriedade faz com que consigamos computar os valores A e B sem gerar números maiores que $2^{16} - 15 = 65521$. Por exemplo, para a matriz M definida por:

$$M = \begin{pmatrix} M_{0,0} & M_{0,1} & M_{0,2} \\ M_{1,0} & M_{1,1} & M_{1,2} \end{pmatrix} = \begin{pmatrix} 10 & 20 & 30 \\ 40 & 50 & 60 \end{pmatrix}$$

a função `calcula_id` deve calcular

$$\begin{aligned} A &= (1 + 10 + 20 + 30 + 40 + 50 + 60) \mod 65521 = 211\,, \\ B &= (1 + 10) + (1 + 10 + 20) + (1 + 10 + 20 + 30) + (1 + 10 + 20 + 30 + 40) + \\ &\quad (1 + 10 + 20 + 30 + 40 + 50) + (1 + 10 + 20 + 30 + 40 + 50 + 60) \mod 65521 = 566\,, \end{aligned}$$

e devolver o inteiro

$$\mathsf{calcula\_id}(M) = 566 \cdot 2^{16} + 211 = 37\,093\,587\,.$$

Como no exemplo acima, o código de identificação de uma matriz sempre retorna um inteiro não negativo menor que 2^{32} (por que é importante garantir isso?); nesse caso, dizemos que a função gera um código hash de 32 bits.

Por sorte, o professor havia recentemente gerado o código de identificação de todas as imagens em seu computador, de forma que podemos verificar se uma imagem restaurada corresponde à imagem original (antes de ser corrompida pelo vírus) com alta probabilidade.

[1] Você pode saber mais sobre essa função consultando a respectiva página da Wikipédia em https://en.wikipedia.org/wiki/Adler-32.

Exemplos de uso da função calcula_id

```
>>> matriz_A = [[0,1,2], [3,4,5]]
>>> matriz_B = [[3,4,5], [0,1,2]]
>>> matriz_C = [[0,1], [1,0]]
>>> matriz_D = [[1,0,2], [3,4,5]]
>>> id = calcula_id(matriz_A)
>>> print(id)
2686992
>>> id = calcula_id(matriz_B)
>>> print(id)
4456464
>>> id = calcula_id(matriz_C)
>>> print(id)
589827
>>> id = calcula_id(matriz_D)
>>> print(id)
2752528
>>> matriz = [[221, 6, 52, 56, 104, 235, 210, 214, 184, 183, 184,
   183, 184, 187,
188, 206], [29, 31, 43, 235, 210, 219, 195, 189, 201, 201, 192,
183, 187, 187, 215, 92], [16, 205, 211, 222, 206, 199, 213, 216,
214, 200, 184, 186, 187, 225, 14, 52], [216, 215, 212, 209, 214,
206, 206, 214, 203, 182, 187, 187, 216, 23, 45, 180], [207, 209,
209, 220, 232, 227, 213, 210, 191, 186, 188, 128, 28, 200, 223,
207], [209, 233, 189, 156, 222, 232, 199, 212, 183, 187, 0, 120,
239, 205, 208, 208], [103, 49, 151, 221, 117, 114, 224, 205, 191,
0, 0, 0, 0, 208, 209, 228], [173, 240, 78, 3, 71, 134, 202, 0, 0,
0, 0, 0, 209, 231, 104, 117], [75, 18, 13, 0, 139, 182, 0, 0, 0,
0, 209, 212, 123, 155, 240, 240], [59, 11, 82, 198, 0, 0, 0, 0,
207, 206, 215, 119, 227, 227, 85, 12], [77, 210, 186, 0, 0, 0, 0,
207, 223, 106, 240, 204, 150, 3, 50, 31], [185, 0, 0, 0, 0, 213,
206, 206, 131, 215, 230, 10, 63, 8, 67, 224], [0, 0, 0, 0, 209,
214, 132, 238, 207, 234, 37, 47, 79, 228, 186, 187], [0, 0, 207,
209, 230, 227, 153, 116, 130, 10, 77, 221, 212, 183, 0, 0], [0,
207, 208, 229, 144, 3, 56, 0, 101, 224, 218, 188, 0, 0, 0, 0],
[209, 211, 168, 10, 34, 6, 136, 236, 211, 206, 185, 0, 185, 188,
186, 187]]
>>> print( calcula_id(matriz) )
4271016262
>>> M0 = [[0, 1, 2, 3, 4]]
>>> print( calcula_id(M0) )
1638411
```

Você pode executar o programa `testa_parte_01.py` para verificar a correção da sua implementação da função `calcula_id` em alguns exemplos simples.

Parte 2: Escrita e leitura de arquivos de texto

Nessa parte você deve implementar as funções `carrega_identificador`, `carrega_imagem`, `salva_imagem` e `carrega_transformações`, que lidam com a manipulação de dados em arquivos de texto.

Leitura do identificador de imagem

A função `carrega_identificador` recebe o nome de um arquivo contendo um inteiro (e presente na mesma pasta que o seu programa `ep3.py`) e devolve esse inteiro.

Na pasta do EP, os arquivos de texto com extensão `.adler32` possuem identificadores (isto é, números inteiros) computados com a função `calcula_id` usando a versão original das imagens correspondentes.

Exemplo de uso da função carrega_identificador

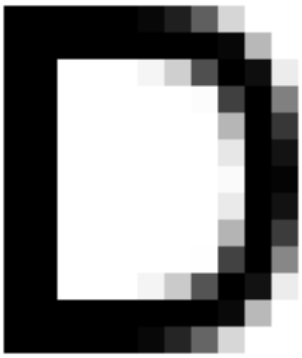
```
>>> identificador = carrega_identificador("img01.adler32")
>>> print(identificador)
297286
```

Leitura e escrita de imagens

Neste EP, trabalharemos com o formato de representação de imagens em arquivo Portable GrayMap (PGM). Em especial, adotaremos a variante P2 do formato que representa imagens monocromáticas (em tons de cinza) e em modo texto.

Nesse formato, uma imagem é representada por uma matriz contendo valores inteiros de intensidade entre 0 e um dado valor máximo. Para simplificar, vamos assumir que o valor máximo é sempre 255. Um arquivo nesse formato é composto de uma descrição do tipo de arquivo (P2), das dimensões da imagem (largura e altura), o valor máximo de intensidade (255) e a matriz de intensidades.

A figura abaixo à esquerda exibe uma imagem de 13 pixels de largura e 15 pixels de altura e sua representação no formato P2. Note que a matriz correspondente possui 15 linhas e 13 colunas, e que a primeira linha representa a parte superior da imagem.



```
P2
13 15
255
255 255 255 255 255 255 255 255 255 255 255 255 255
255   0   0   0   0   0   8  31  95 214 255 255 255
255   0   0   0   0   0   0   0   7 184 255 255 255
255   0   0 255 255 255 245 207  79   0  14 236 255
255   0   0 255 255 255 255 255 252  64   0 128 255
255   0   0 255 255 255 255 255 255 182   0  55 255
255   0   0 255 255 255 255 255 255 231   0  19 255
255   0   0 255 255 255 255 255 255 250   0   4 255
255   0   0 255 255 255 255 255 255 234   0  18 255
255   0   0 255 255 255 255 255 255 180   0  60 255
255   0   0 255 255 255 255 255 254  66   0 135 255
255   0   0 255 255 255 244 202  84   0  18 236 255
255   0   0   0   0   0   0   0   0   9 185 255 255
255   0   0   0   0   0   8  37 101 215 255 255 255
255 255 255 255 255 255 255 255 255 255 255 255 255
```

Mais informações sobre o formato e sobre representações de imagens podem ser encontradas em http://www.vision.ime.usp.br/~pmiranda/mac2166_1s17/aulas/P3/pgm_ppm.pdf.

A função `carrega_imagem` recebe uma string contendo o nome do arquivo de uma imagem em formato P2 (presente na mesma pasta onde está o arquivo `ep3.py`) e devolve a matriz de inteiros com os valores de intensidade de cada pixel.

Exemplo de uso da função carrega_imagem

```
>>> minha_imagem = carrega_imagem('img01.pgm')
>>> print(minha_imagem)
[[224, 205, 191, 0, 0, 0, 0, 208, 209, 228, 103, 49, 151, 221, 117,
  114], [202, 0, 0, 0, 0, 0, 209, 231, 104, 117, 173, 240, 78, 3,
  71, 134], [0, 0, 0, 0, 209, 212, 123, 155, 240, 240, 75, 18, 13,
  0, 139, 182], [0, 0, 207, 206, 215, 119, 227, 227, 85, 12, 59, 11,
  82, 198, 0, 0], [0, 207, 223, 106, 240, 204, 150, 3, 50, 31, 77,
  210, 186, 0, 0, 0], [206, 206, 131, 215, 230, 10, 63, 8, 67, 224,
  185, 0, 0, 0, 0, 213], [132, 238, 207, 234, 37, 47, 79, 228, 186,
  187, 0, 0, 0, 0, 209, 214], [153, 116, 130, 10, 77, 221, 212, 183,
  0, 0, 0, 0, 207, 209, 230, 227], [56, 0, 101, 224, 218, 188, 0, 0,
  0, 0, 0, 207, 208, 229, 144, 3], [136, 236, 211, 206, 185, 0, 185,
  188, 186, 187, 209, 211, 168, 10, 34, 6], [210, 214, 184, 183,
  184, 183, 184, 187, 188, 206, 221, 6, 52, 56, 104, 235], [195,
  189, 201, 201, 192, 183, 187, 187, 215, 92, 29, 31, 43, 235, 210,
  219], [213, 216, 214, 200, 184, 186, 187, 225, 14, 52, 16, 205,
  211, 222, 206, 199], [206, 214, 203, 182, 187, 187, 216, 23, 45,
  180, 216, 215, 212, 209, 214, 206], [213, 210, 191, 186, 188, 128,
  28, 200, 223, 207, 207, 209, 209, 220, 232, 227], [199, 212, 183,
  187, 0, 120, 239, 205, 208, 208, 209, 233, 189, 156, 222, 232]]
```

A função `salva_imagem` recebe uma string contendo o nome de um arquivo e uma matriz representando os valores de intensidades dos pixels de uma imagem e salva essa imagem (na mesma pasta onde está o programa `ep3.py`) em formato P2.

Exemplo de uso da função salva_imagem

```
>>> matrizA = [[1,2,3,4], [5,6,7,8], [9,10,11,12]]
>>> salva_imagem('minhaimg.pgm', matrizA)
```

minhaimg.pgm

```
P2
4 3
255
1 2 3 4
5 6 7 8
9 10 11 12
```

Para visualizar arquivos no formato PGM P2, você pode instalar editores de imagens como o GIMP (https://www.gimp.org), conversores de imagens como o ImageMagick (http://www.imagemagick.org/script/index.php) ou ainda utilizar ferramentas online como o FreeFileConvert (https://www.freefileconvert.com).

Leitura de transformações

Como veremos mais adiante na Parte 3, uma transformação afim pode ser representada por uma matriz de tamanho 2-por-3. A função `carrega_transformações` recebe uma string contendo o nome de um arquivo e devolve uma lista de transformações (matrizes 2-por-3) contidas no arquivo. O arquivo de transformações possui a seguinte estrutura:

- A primeira linha descreve o número de transformações no arquivo;
- Linhas iniciadas pelo caractere # indicam comentários e devem ser ignoradas;
- As demais linhas representam uma matriz T de tamanho 2-por-3 como $T_{0,0}\ T_{0,1}\ T_{0,2}\ T_{1,0}\ T_{1,1}\ T_{1,2}$.

transf1.txt

```
5
# T1
# 10-10 DB
1 0 10 0 1 10
# T2
1 0 -20 0 1 -20
# T3
-1 0 0 0 -1 0
# T4
1 1 0 0 1 0
# T5
1 0 0 1 1 0
```

Exemplo de uso da função carrega_transformações

```
>>> nome = "transf1.txt"
>>> tsf = carrega_transformações(nome)
>>> print(tsf)
[[[1, 0, 10], [0, 1, 10]],
 [[1, 0, -20], [0, 1, -20]],
 [[-1, 0, 0], [0, -1, 0]],
 [[1, 1, 0], [0, 1, 0]],
 [[1, 0, 0], [1, 1, 0]]]
```

Testando as funções

Você pode executar o programa `testa_parte_02.py` para verificar possíveis erros na sua implementação das funções dessa parte.

Parte 3: Transformações geométricas afins

Uma transformação geométrica é uma bijeção que mapeia cada ponto ou coordenada de uma região em um ponto distinto. Nesse EP, estamos interessados em transformações geométricas que mapeiam cada pixel de uma imagem M de tamanho A-por-L em um pixel da imagem N tal que $N_{y',x'} = M_{y,x}$, onde

$$\begin{aligned} x' &= T_{0,0} \cdot x + T_{0,1} \cdot y + T_{0,2} \mod L\,, \\ y' &= T_{1,0} \cdot x + T_{1,1} \cdot y + T_{1,2} \mod A\,, \end{aligned}$$

e T é uma matriz de inteiros de tamanho 2-por-3. Essa tipo de transformação realiza uma permutação dos pixeis da imagem original. A figura abaixo mostra o resultado da aplicação da transformação

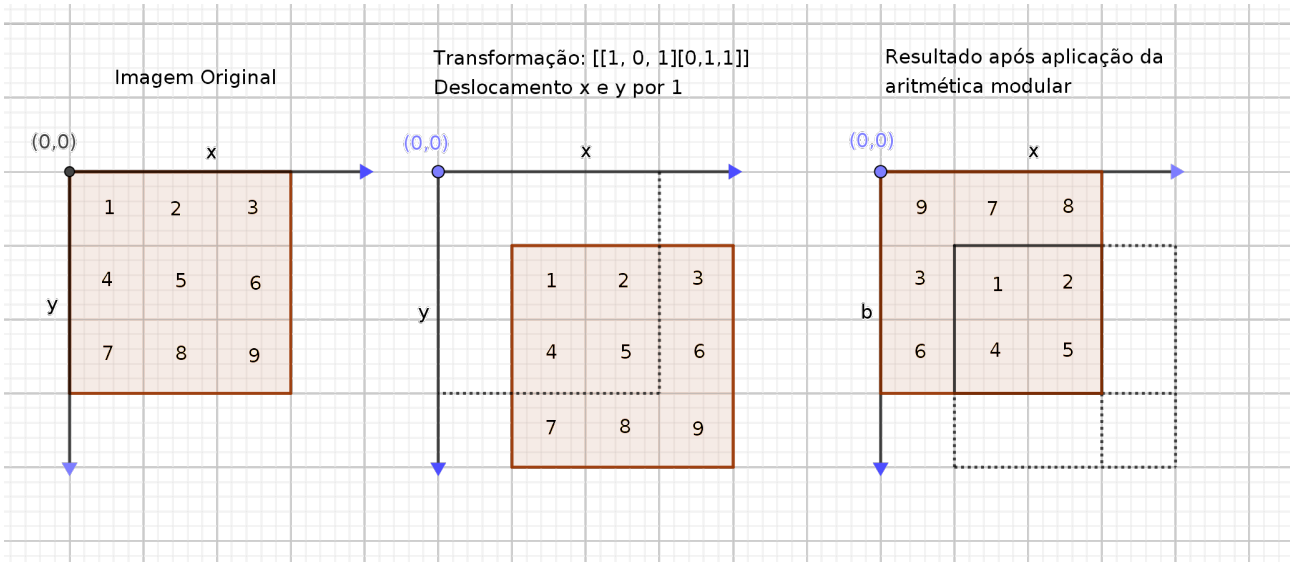
$$T = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}$$

na imagem

$$M = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix},$$

resultando na imagem

$$N = \begin{pmatrix} 9 & 7 & 8 \\ 3 & 1 & 2 \\ 6 & 4 & 5 \end{pmatrix}.$$



A função `transforma` recebe uma matriz de inteiros M representando uma imagem monocromática e uma matriz de transformação T, e devolve uma matriz N produzida pela aplicação de T em M.

⚠ A função `transforma` não deve modificar a matriz M.

Listing 1: Exemplos de uso da função transforma

```
>>> matriz_A = [[1,2,3],[4,5,6],[7,8,9]]
>>> T = [[1,0,1],[0,1,1]]
>>> matriz_transf_A = transforma(matriz_A, T)
>>> print(matriz_transf_A)
[[9, 7, 8], [3, 1, 2], [6, 4, 5]]
>>> matriz_B = [[1,2],[3,4],[5,6]]
>>> matriz_transf_B = transforma(matriz_B, T)
>>> print(matriz_transf_B)
[[6, 5], [2, 1], [4, 3]]
>>> M0 = [[0, 1, 2, 3, 4]]
>>> TE = [[1,0,-1],[0,1,0]]
>>> M1 = transforma(M0, TE)
>>> print(M1)
[[4, 0, 1, 2, 3]]
>>> M2 = transforma(M1, TE)
>>> print(M2)
[[2, 3, 4, 0, 1]]
>>> M3 = transforma(M2, TE)
>>> print(M3)
[[3, 4, 0, 1, 2]]
>>> M4 = transforma(M0, T)
>>> print(M4)
[[4, 0, 1, 2, 3]]
```

Você pode executar o programa `testa_parte_03.py` para verificar possíveis erros na sua implementação dessa parte.

Parte 4: Busca sistemática em profundidade

Muitos problemas computacionais consistem em encontrar uma sequência de operações que transformam um objeto de uma configuração inicial em uma configuração desejada. Por exemplo, a solução de um quebra-cabeça Sudoku pode ser encontrada através de uma sequência de operações que preenchem uma lacuna vazia restante com um digito válido até que uma compleição válida seja alcançada (uma compleição é válida se cada linha, coluna e região contém todos os inteiros entre 1 e 9). A figura abaixo contém um exemplo de tabela original (à esquerda) com as lacunas disponíveis e uma possível solução (à direita) obtida por uma sequência de operações de compleição (escrita de um número válido em lacuna).

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

O processo de encontrar uma sequência de operações que produza uma configuração desejada (chamada de solução do problema) é chamado de busca sistemática ou busca exaustiva (ou simplesmente busca). Se aplicamos uma operação em um objeto, produzimos um novo problema de busca, um com uma configuração inicial distinta e mesma configuração desejada. Por exemplo, no jogo de Sudoku se completamos uma lacuna qualquer geramos um novo jogo com mesmo objetivo mas configuração inicial distinta. Essa propriedade permite que tais problemas sejam resolvidos por um algoritmo recursivo simples conhecido por busca em profundidade que consiste nos seguintes passos:

1. Selecione e aplique uma operação gerando um novo problema.
2. Se esse problema possui solução, retorne essa solução e encerre a busca.
3. Caso contrário, repita o item 1.

Se todas as operações levarem a subproblemas sem solução então o problema original também não possui solução. Caso contrário o algoritmo garantidamente encontra uma sequência de operações que transforma um objeto de uma configuração inicial a uma configuração final. Na prática, encontrar uma solução pode requerer a aplicação de um número muito grande de operações, tomando muito tempo. Para evitar encontrar soluções muito custosas computacionalmente, podemos definir um limite no número máximo de operações aplicadas, de forma a garantir que o algoritmo sempre termine dentro de um período de tempo razoável (por outro lado, isso faz com que o algoritmo às vezes falhe ao não encontrar uma solução mesmo quando ela existe).

O nosso problema de restauração das imagens pode ser resolvido por um algoritmo de busca em profundidade no qual as operações são transformações geométricas na imagem e o objetivo é encontrar uma imagem cujo identificador é o mesmo da imagem original (note que como os identificadores não são únicos, é possível encontrar uma imagem distinta da original com mesmo identificador – isso também será considerado uma solução do problema). Como testar todas

as transformações geométricas afins seria muito ineficiente, vamos assumir que possuímos uma lista de transformações candidatas.

O pseudocódigo abaixo descreve o algoritmo de busca em profundidade com número máximo de operações que aplica transformações geométricas a fim de encontrar uma imagem cujo identificador coincide com um identificador dado. O algoritmo recebe como argumento uma matriz de inteiros M representando a imagem modificada, uma lista de transformações L, um inteiro C contendo o identificador da imagem original e o número máximo restante de transformações S. Caso o algoritmo não encontre uma sequência de até S transformações T em L que produza uma imagem cujo identificador é S, ele retorna a palavra reservada `None`, caso contrário ele retorna a matriz encontrada.

```
função Busca(M, L, C, S)
  se calcula_id(M) = C então devolva M
  se S = 0 então devolva None
  para toda transformação T ∈ L faça
      N = transforma(M, T)
      R = Busca(N, L, C, S − 1)
      se R ≠ None então devolva R
  devolva None
```

A título de exemplo, considere a seguinte família de imagens/matrizes:

$$M_0 = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 \end{pmatrix} \qquad M_1 = \begin{pmatrix} 1 & 2 & 3 & 4 & 0 \end{pmatrix}$$
$$M_2 = \begin{pmatrix} 2 & 3 & 4 & 0 & 1 \end{pmatrix} \qquad M_3 = \begin{pmatrix} 3 & 4 & 0 & 1 & 2 \end{pmatrix}$$
$$M_4 = \begin{pmatrix} 4 & 0 & 1 & 2 & 3 \end{pmatrix}$$

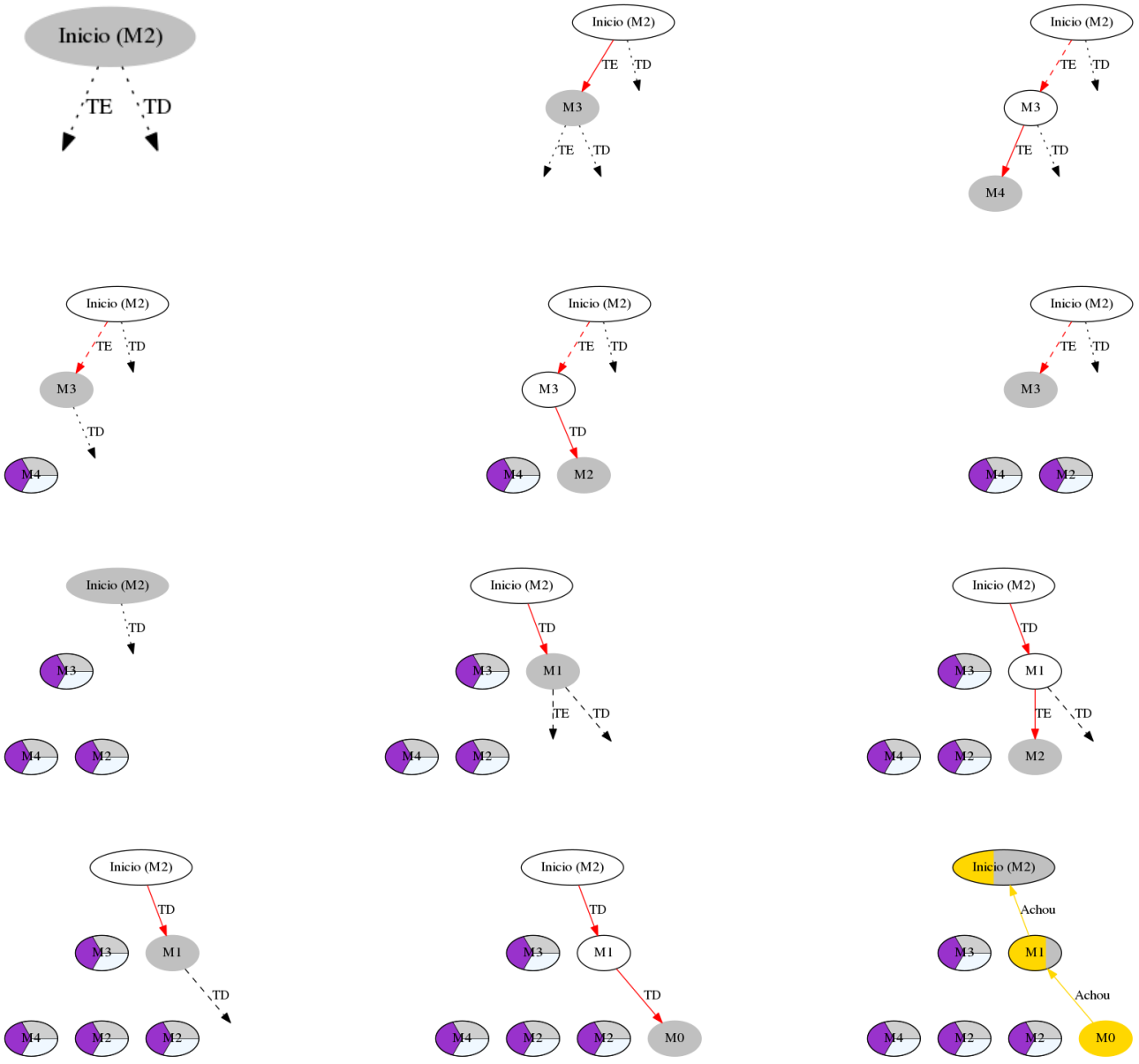
Considere também as seguintes transformações

$$TE = \begin{pmatrix} 1 & 0 & -1 \\ 0 & 1 & 0 \end{pmatrix}, \qquad TD = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}.$$

A transformação TE desloca a imagem 1 pixel à esquerda, enquanto que a transformação TD desloca a imagem 1 pixel à direita. Suponha que queremos encontrar a sequência de no máximo 2 transformações que transforma a matriz M_2 em uma matriz cujo identificador é $C = 1\,638\,411$. Para isso, executamos o algoritmo Busca com argumentos $M = M_2$, $L = [TE, TD]$, $C = 1638411$ e $S = 2$.

De acordo com a descrição do algoritmo, primeiro calculamos o código de identificação de M_2, que é 2621451, e verificamos que ele não coincide com a matriz buscada. Então, aplicamos a transformação TE em M_2, obtendo a matriz $N = M_3$. A busca é então repetida utilizando M_3 como entrada e decrementando o número máximo de transformações restantes (ou seja, chamando $\text{Busca}(M_3, L, C, 1)$). Essa nova chamada nos leva ao cálculo da identificação de M_3, que é 2621451, e a aplicação de TE em M_3, obtendo $N = M_4$ e gerando a chamada recursiva $\text{Busca}(M_4, L, C, 0)$. Essa chamada calcula o código de identificação de M_4 (que é 2293771), verifica que não é igual ao da matriz buscada e então retorna `None` (pois $S = 0$). Tentamos então aplicar a transformação TD em M_3, iniciando uma nova busca a partir de M_2, que retorna `None`, indicando que não é possível obter matriz cuja identificação é S aplicando TE ou TD em M_2. Tentamos então aplicar TD em M_2, gerando a chamada $\text{Busca}(M_2, L, C, 1)$. Essa busca por fim encontra a matriz M_0, cujo código de identificação é S, aplicando duas transformações

TD. Essa matriz é devolvida, encerrando a busca. Os diagramas abaixo ilustram passo a passo a sequência de chamadas recursivas feitas pelo algoritmo como descrito.



A função `busca` implementa o algoritmo de busca em profundidade por uma matriz cujo código de identificação é C utilizando uma sequência de no máximo S transformações da lista L.

Você não precisa implementar o algoritmo exatamente como descrito no pseudocódigo dessa seção, mas sua implementação deve ser recursiva e retornar a mesma solução que o algoritmo descrito retornaria. Como sempre, você não deve modificar o protótipo da função `busca`.

Você pode rodar o programa `testa_parte_04.py` para testar a sua função de busca.

Parte 5: Juntando tudo

A sua última tarefa é implementar a função `main` contendo a interface entre o usuário e as funções previamente implementadas. Quando executado, o seu programa `ep3.py` deve requisitar e receber as informações do usuário nessa exata ordem:

1. Nome do arquivo com a imagem transformada
2. Nome do arquivo contendo o identificador da imagem original
3. Nome do arquivo contendo as transformações a serem usadas na busca
4. Nome do arquivo com a imagem original (caso seja encontrada)
5. Número máximo de transformações

⚠ Caso seu programa não encontre à imagem, ele deve salvar no arquivo de saída e mesma entrada do arquivo de entrada, e sugerir ao usuário que aumente o número máximo de transformações aplicadas ou modifique a lista de transformações.

Alguns exemplos de execução da interface são exibidos abaixo (você está livre para implementar sua interface desde que ela obedeça a ordem de entrada dos dados acima, e que gere os arquivos correspondentes corretamente).

```
*****************************
*** Programa desfaz vírus ***
*****************************

Autor: Denis Mauá
NUSP: 3730790

Entre com o nome do arquivo contendo imagem transformada: img00.pgm
Entre com o nome do arquivo contendo o identificador da imagem original: img00.adler
Entre com o nome do arquivo contendo as transformações disponíveis: transformações00.txt
Entre com o nome do arquivo onde a imagem original deve ser salva: resultado00.pgm
Entre com o número máximo de transformações: 2

Tentando restaurar imagem 'img00.pgm'... Falhou!

Não foi possível encontrar uma imagem com o identificador 870647247 utilizando uma sequência de no máximo 2
    transformações em 'transformações00.txt'!

Você pode tentar aumentar o número máximo de transformações ou mudar o arquivo de transformações.
```

```
*****************************
*** Programa desfaz vírus ***
*****************************

Autor: Denis Mauá
NUSP: 3730790

Entre com o nome do arquivo contendo imagem transformada: img01.pgm
Entre com o nome do arquivo contendo o identificador da imagem original: img01.adler
Entre com o nome do arquivo contendo as transformações disponíveis: transformações.txt
Entre com o nome do arquivo onde a imagem original deve ser salva: resultado01.pgm
Entre com o número máximo de transformações: 6

Tentando restaurar imagem 'img01.pgm'... Pronto!

Imagem com o identificador 870647247 salva em 'resultado01.pgm'!
```

⚠ A busca pode demorar bastante tempo, especialmente se o número de transformações é grande e a imagem não é muito pequena. Assim, recomenda-se que durante a implementação você teste suas funções com poucas transformações e com o menor número máximo necessário de transformações para conseguir encontrar a imagem original. Esse número pode ser encontrado executando seu programa múltiplas vezes gradualmente aumentando o número máximo até que a imagem original seja encontrada.

Instruções para entrega

Você deve submeter via Graúna o arquivo `ep3.py` contendo a sua solução até às 23:55 do dia 23/6/2018. Para evitar que seu EP seja zerado, certifique-se que o arquivo foi submetido sem problemas (baixando e executando o arquivo do site), e que ele consiste em um executável Python (ou seja, que se trata de um arquivo de texto contendo comandos Python e não de um arquivo do Jupyter notebook, ou de outro editor de texto como o Word).

Avaliação

O seu programa será avaliado pela implementação bem sucedida de cada função independentemente, assim como por questões subjetivas como documentação do código e simplicidade (assim, você ainda pode conseguir uma boa nota mesmo que não consiga implementar todas as funções corretamente, e ainda pode obter uma nota baixa mesmo que implemente todas as funções). O critério detalhado de avaliação será divulgado posteriormente.

Conforme mencionado anteriormente, é **muito importante** que seu código passe em todos os testes. Entretanto, passar em todos os testes não é garantia de que seu código receberá nota máxima. É possível que os testes não verifiquem alguns casos peculiares onde seu programa pode vir a falhar.

Exemplos de imagens transformadas e os respectivos códigos de identificação são fornecidos junto com o esqueleto do código, assim como arquivos com transformações. Embora estes exemplos envolvam imagens pequenas e arquivos com poucas transformações, eles são suficientes para testar seu programa. Você no entanto pode gerar novos casos de exemplo usando as funções no seu próprio código. Lembre-se que a transformação precisa ser inversível, e que o objetivo é encontrar uma transformação inversa àquela aplicada para gerar a imagem (portanto é melhor que o arquivo de transformação contenha as inversas das matrizes usadas para gerar a imagem).

Bom trabalho!