Terceiro Exercício-Programa: Vírus Ge0m3tRik4

Entrega: **23/6/2018**

Versão do enunciado: 18/06/2018 (2.1 17/06/2018) (2 de 10/06/2018)

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, como ilustrado abaixo.

imagem original

imagem deformada pelo vírus

Estudando cuidadosamente as imagens deformadas, o professor percebeu que o vírus aplicou uma transformação geométrica sobre os pixeis da imagem. Tomando a imagem deformada acima como exemplo, ele conseguiu, após muitas tentativas e erros, descobrir uma sequência de transformações geométricas (afim lineares) que restaurou 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, o professor possuía um arquivo com as *assinaturas*[1] de cada uma das imagens que tinha em seu computador e este arquivo não foram afetados pelo vírus. Dessa forma, é possível verificar se uma sequência de transformações resulta na imagem original de maneira automática.

[1] Esse é um código para verificação de integridade do arquivo. Vide explicaćão na *Parte 1* na página 3.

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 geométricas sobre a imagem até que seja gerada uma imagem que possa ser a imagem original (que apresente um memo "identificador").

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 (técnica de *tentativa e erro*).

A *função de dispersão* será usada para gerar um *identificador* a ser usado como assinatura para os arquivos. O princípio é o mesmo dos dígitos de controle, como os dois dígitos usado para confirmar os nove (9) primeiros dígitos do *CPF*.

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.

Apoio: Funções de apoio, como funções para impressão de saídas ou para depuração do código;

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

Parte 2: Eliminada!

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 trecho do programa que lê opção de execução (1, 2, 3, 4 ou 5) e invoca função correspondente.

1. `ler_matriz_imagem(...)`: Ler dados via teclado para compor matriz da imagem (opcoes 1, 3, 4, 5)
2. `ler_transformacao(...)`: Ler dados para compor uma matriz de transformacao (opcoes 2, 3, 4, 5)

3. `ler_sequencia_matrizes_transformacoes(...)`: Ler dados lista de matrizes de transformacao (opcoes 3 e 8)
4. `testar_espalhamento(...)`: opcao == 1 => ler via teclado matriz de imagem M e imprimir seu identificador 'ident(M)'
5. `testar_lista_transformacoes(...)`: opcao == 2 => ler via teclado n e n matrizes de transformacoes, imprimir soma elementos matrizes e identif.
6. `testar_transformacoes_imagens(...)`: opcao == 3 => ler imagem M, n e n transformacoes, imprimir soma elementos das transf. e identif. transf. T_i sobre M
7. `testar_sequencia_transformacoes_imagem(...)`: opcao == 4 => ler ident. M, matriz M, n e n transf., indices transf., aplicar transf. T_i_1,... T_i_k e ver se recuperou imagem M
8. `buscar_sequencia_transformacoes_para_imagem(...)`: opcao == 5 => ler ident. M, matriz M, n e n transformacoes T1,... ,Tn, limite maximo recorrencia k, depois tentar recursivamente (ate' k) encontrar a seq. de transf. que recupere a imagem M

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 passar nos testes, certamente ele está com problemas. 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 informação:

```
// Implemente este trecho!
```

ou

```
# Implemente este trecho!
```

Esse marcador serve para informar os trecho que você precisa implementar.

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 gerar um **código de identificação** que servirá como sua **assinatura** e que é transmitido separadamente. Ao receber o arquivo o receptor aplica sobre ele o mesmo algoritmo para gerar *assinaturas* de arquivos e compara o resultado com a *assinatura* recebida. Se os códigos não coincidirem o arquivo é considerado *corrompido*.

Um tipo de algoritmo comumente utilizada para *assinaturas* de arquivos é a utilização de uma função de dispersão ou espalhamento (em inglês, *hash function*), que associa a cada arquivo (de qualquer tamanho) um número natural (com tamanho máximo definido). Em geral essa técnica é utilizada para associar palavras a uma posićão de vetor, mas isso implica que mais de uma palavra poderá gerar a mesma posićão (*colisão*). Assim, as funćões de espalhamento são projetas para gerar baixa *colisão*.

Sua primeira tarefa é implementar a função `calcula_id_dispersao`, 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*:[2]

$$\mathsf{calcula\_id}(M) = B \cdot 65536 + A, \quad (\text{note que } 2^{16} = 65536)$$

onde

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

Note que $\mod$ 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). Observe ainda que a função $f(x) := x \mod z$ apresenta um propriedade distributiva em relaćão a operaćão de adição, que é a seguinte: $f(x + y) = f(f(x) + f(y))$, ou seja, $(x + y) \mod z = ((x \mod z) + (y \mod z)) \mod z$. Por exemplo, para a matriz $M_{2\times3}$ definida por:

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

ao aplicar a função `calcula_id_dispersao` sobre M obtemos

$$A = (1 + 10 + 20 + 30 + 40 + 50 + 60) \mod 65521 = 211\,,$$
$$B = (1 + 10) + (1 + 10 + 20) + (1 + 10 + 20 + 30) + (1 + 10 + 20 + 30 + 40) +$$
$$(1 + 10 + 20 + 30 + 40 + 50) + (1 + 10 + 20 + 30 + 40 + 50 + 60) \mod 65521 = 566.$$

O que resulta em

$$\mathsf{calcula\_id}(M) = 566 \cdot 65536 + 211 = 37\,093\,587.$$

Como no exemplo acima, o código de identificação de uma matriz sempre resulta em um inteiro não negativo menor que 2^{32} (por que é importante garantir isso?). Nesse caso, dizemos que a função gera um código de espalhamento (*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.

Exemplos de uso da função calcula_id

```
matriz_A = [0, 1, 2]
           [3, 4, 5]
calcula_id_dispersao(matriz_A) = 2686992

matriz_B = [3, 4, 5]
           [0, 1, 2]]
calcula_id_dispersao(matriz_B) = 4456464

matriz_C = [0, 1]
           [1, 0]
calcula_id_dispersao(matriz_C) = 589827

matriz_D = [1, 0, 2]
           [3, 4, 5]
calcula_id_dispersao(matriz_D) = 2752528
```

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

Você pode usar o programa `testa_parte_01.*` (com extensão para *C* ou para *Python*) para verificar a correção da sua implementação da função `calcula_id_dispersao` em alguns exemplos simples. Estes arquivos estarão respectivamentes nos pacotes `mac2166_2018_web_modelo_c.tgz` para *C* e `mac2166_2018_web_modelo_py.tgz` para *Python*.

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

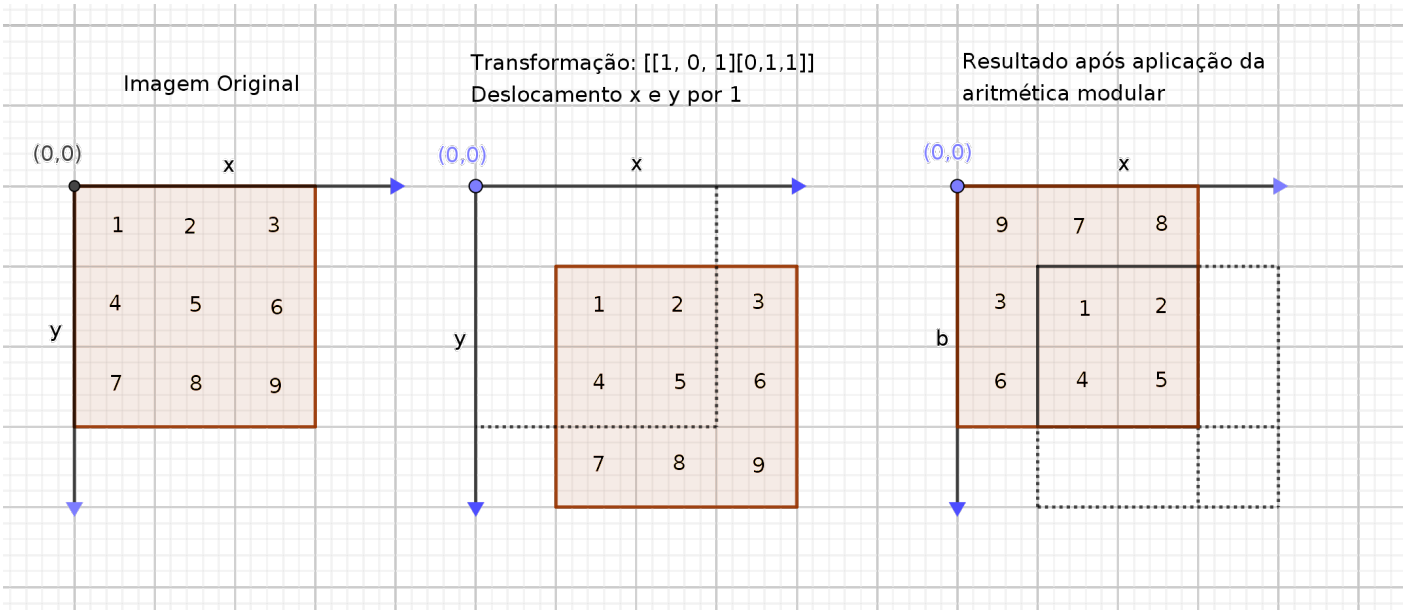
$$T = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}$$

na imagem

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

resultando na imagem

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



Note que nesse sistema de coordenadas, as ordenadas (eixo vertical y) "crescem" para baixo.

A função `transforma` recebe (ao menos) uma matriz de inteiros M, representando uma imagem monocromática, e uma matriz de transformação T, e gera uma matriz N produzida pela aplicação de T em M. No caso de *C*, essa função nada devolve (`void`) e em *Python* devolve N .

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

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);
imprima(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);
imprima(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);
imprima(M1);
>> [[1, 2, 3, 4, 0 ]]
M2 = transforma(M1, TE);
>> imprima(M2);
[[2, 3, 4, 0, 1]]
M3 = transforma(M2, TE);
imprima(M3);
>> [[3, 4, 0, 1, 2]]
M4 = transforma(M3, T);
imprima(M4);
>> [[4, 0, 1, 2, 3]]
```

2018/06/17: **Atenção**, acerto no exemplo acima, particularmente no resultado da primeira impressão de `imprima(M1);` e nas linhas que aplicava a transformação (estava usando erroneamente a matriz `M0`, mas deveria iterar, aplicar sobre `M1`, `M2` e `M3`).

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 devolve nada. caso contrário ele retorna a matriz encontrada.

```
função Busca(M, L, C, S) // em C precisa de mais parâmetros

Se      (calcula_id(M) == C)
        Devolva M
        FimSe
Se      (S == 0)
        Devolva vazio
        FimSe
Para    transformação T ∈ L
        State N = transforma(M, T)
        State R = Busca(N, L, C, S − 1)
        Se (R ≠ vazio)
            Devolva R
            FimSe
        FimPara
Devolva vazio
```

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

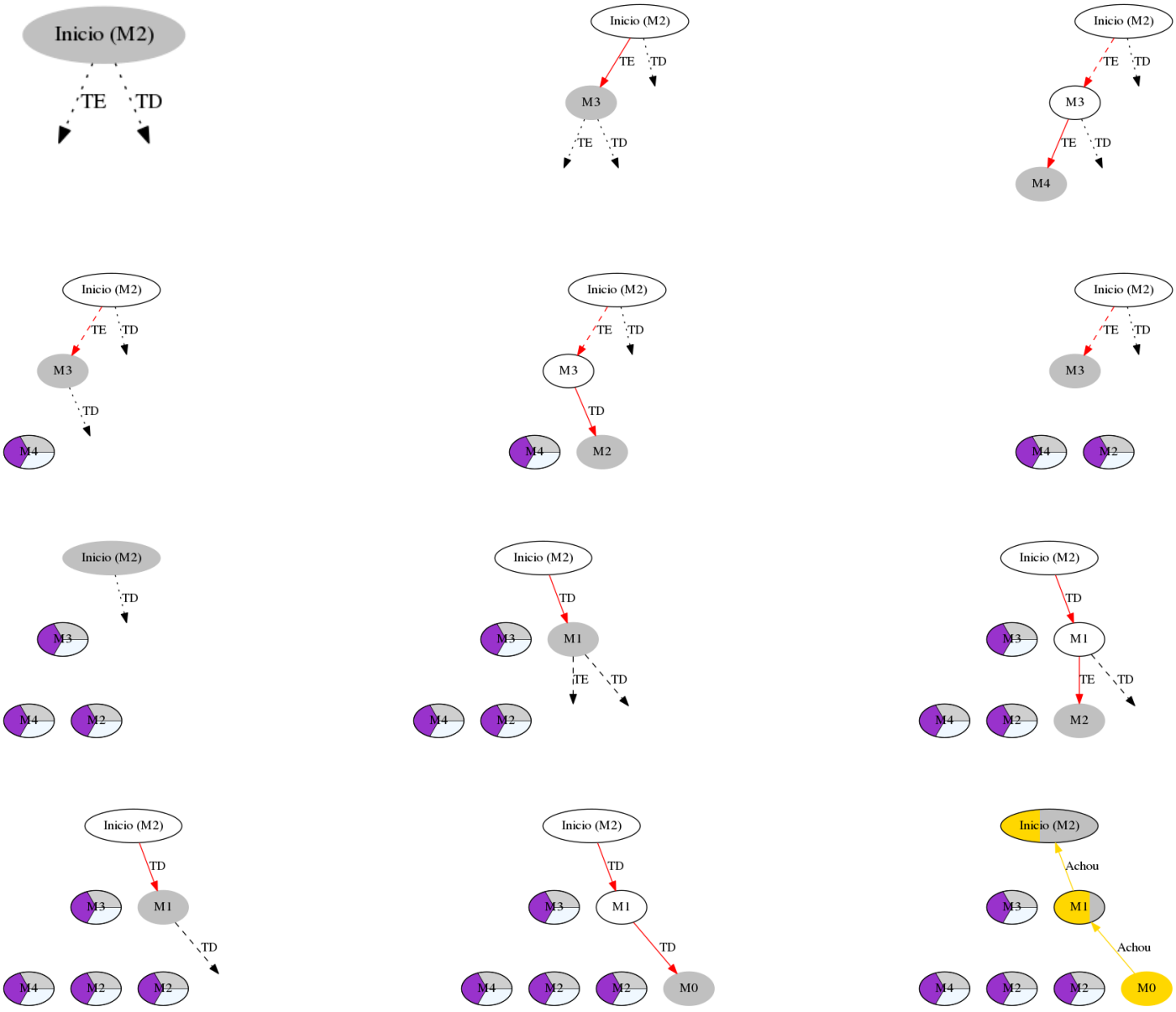
$$M_0 = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 \end{pmatrix} \qquad M_1 = \begin{pmatrix} 1 & 2 & 3 & 4 & 0 \end{pmatrix}$$
$$M_2 = \begin{pmatrix} 2 & 3 & 4 & 0 & 1 \end{pmatrix} \qquad M_3 = \begin{pmatrix} 3 & 4 & 0 & 1 & 2 \end{pmatrix}$$
$$M_4 = \begin{pmatrix} 4 & 0 & 1 & 2 & 3 \end{pmatrix}$$

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 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 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 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 código para o EP3 deve requisitar e receber as informações do usuário nessa exata ordem:

1. um inteiro *opcao* com valor entre 1 e 5;

2. dependendo do valor, muda o que deve ser digitado (consultar o modelo).

⚠ 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 seu arquivo via *SAW* 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 e dispare o avaliador automático para ele (entre novamente e verifique que ele está ali e com a nota do avaliador automático).

Se os casos de testes ainda não estiverem disponíveis, vá trabalhando e a cada nova versão, atualize no *SAW*. Não deixe para os últimos minutos para evitar problemas. A cada nova submissão experimente o avaliador automático (se já disponível).

Avaliação

O seu programa deverá primeiro receber uma avaliação automática do *SAW*. Posteriormente, faremos um exame "manual", avaliando a legibilidade do código e se atendeu às restrições.

Conforme mencionado anteriormente, é **muito importante** que seu código passe em todos os testes disponibilizado. Entretanto, passar neles 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!