Arnold's Cat Map

Consider taking a square image, consisting of N-by-N pixels, where the coordinate of each pixel is represented by the ordered pair (X, Y).

Arnold's cat map induces a discrete-time dynamical system in which the evolution is given by iterations of the mapping $\Gamma_{cat} : \mathbb{T}^2 \to \mathbb{T}^2$ given the formula $\Gamma_{cat}(x, y) \to (2x + y,\ x + y) \ mod\ 1$ where

$$\Gamma_{cat}\left(\begin{bmatrix} X_{n+1} \\ Y_{n+1} \end{bmatrix}\right) = \begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix}\begin{bmatrix} X_n \\ Y_n \end{bmatrix}(mod\ 1)$$

Connection between Arnold's cat and Fibonacci's rule

Let the nth number of the Fibonacci sequence be defined by the recurrence relation $F_n = F_{n-1} + F_{n-2}$ with $F_0 = 0,\ F_1 = 1$.

We can get the first Fibonacci numbers: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, …

Hence, the matrix representation of $F = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} F_0 & F_1 \\ F_1 & F_2 \end{bmatrix}$ will generate the sequence

$$F^n = \begin{bmatrix} F_{n-1} & F_n \\ F_n & F_{n+1} \end{bmatrix}$$

Let's say we iterate twice F^2, then $F = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix}$ which is also equal to the largest eigenvalue of F.

In 1774, Lagrange discovered repeating patterns in the Fibonacci sequence $mod\ N$. The N^{th} Pisano period, written $\pi(N)$, is the number of times the Fibonacci sequence, modulo N, repeats.

For example, if we take the first 15 Fibonacci numbers F_{15} with $F_0 = 0,\ F_1 = 1$, we will get

$$0,1,1,2,3,5,8,13,21,34,55,89,144,233,377$$

Then we take the remainder using the modular arithmetic of each number k in the sequence by dividing it to the N.

If $N = 3$, we will get $0,1,1,2,0,2,2,1,0,1,1,2,0,2,2$ with a pattern of

$$0,1,1,2,0,2,2,1, \qquad 0,1,1,2,0,2,2$$

As a result, we can see the 3rd Pisano period is 8.

Python implementation

At first, we want to load the image and get its width and height

```
img = Image.open(input_img)
width, height = img.width, img.height
```

In this program, user may only take a square size of the image. If the condition does not meet this requirement, we can create a function that resizes it to square.

```python
def resize_img(self, img) -> Image:
    min_size = min(img.size)
    imageBoxSize = 200 # maximum width of image placeholder

    # arnold's cat map must be square
    if min_size >= imageBoxSize:
        resized_im = img.resize((imageBoxSize,imageBoxSize))
    else:
        resized_im = img.resize((min_size,min_size))

    return resized_im
```

This simply explains to resize the image to the desired size if the image exceeds the value of the image placeholder's otherwise, resize the image to its minimum size of (width, height) to make a square image.

Now to make the chaotic map, we can create a new canvas and take the image we have as reference.

```python
canvas = Image.new(img.mode, (img.width, img.height))
```

Given by the formula for Arnold's Cat Map, we can draw and apply the rule on each pixel of the image in the x and y coordinates of the canvas and setting the N equal to the width or height of the image. Thus,

```
done = False
while not done:
    for x in range(canvas.width):
        for y in range(canvas.height):
            nx = (2 * x + y) % canvas.width
            ny = (x + y) % canvas.height
            canvas.putpixel((nx,canvas.height-ny-1),
                             img.getpixel((x, canvas.height-y-1)))
    self.iteration += 1
    new_image = self.images_path + f'ACM-{namex}-{self.iteration}.png'
    canvas.save(new_image)
    img = Image.open(new_image)

    if images_the_same(self.images_path + f'ACM-{namex}-0.png', new_image):
        done = True
```

This is done by iterating infinitely until we get the same image. This is done using the OpenCV module in python.

```
import cv2
def images_the_same(image1, image2):
    """
        :param image1: path of image1
        :param image2: path of image2
        :return: True if images are the same, False if images are not the same
    """
    im1 = cv2.imread(image1)
    im2 = cv2.imread(image2)

    if im1.shape != im2.shape:
        return False

    difference = cv2.subtract(im1, im2)
    b, g, r = cv2.split(difference)

    if (cv2.countNonZero(b) == 0 and cv2.countNonZero(g) == 0 and
        cv2.countNonZero(r) == 0):
        return True
    return False
```

As we recall the `imageBoxSize` is sets to 200 because the higher size of the image will result in longer process of doing the map. Overall, the time complexity represented in Big O notation of this program is $O(n^2) + m$ where n represents the width and height of the image and m is the iteration until the original image reappears.