Examen final 2020-09-14

95.14/75.40 - Algoritmos y Programación I - Curso Essaya

Objetivo

Implementar la clase `Imagen`, que representa una imagen digital de tamaño arbitrario.

Se dispone de los siguientes archivos:

- `imagen.py`: implementación (inicialmente vacía) de la clase `Imagen`
- `pruebas.py`: Pruebas automáticas

El archivo `pruebas.py` efectúa una serie de pruebas para verificar el correcto funcionamiento de la clase `Imagen`.

La idea es agregar en `imagen.py` todo el código necesario para que las pruebas automáticas pasen.

Hay 5 ejercicios. Es condición necesaria (pero no suficiente) para aprobar el examen que haya 3 ejercicios OK.

Salida del programa

Al ejecutar el programa (`python3 pruebas.py`), se ejecutan todas las pruebas, y se imprime el resultado de cada ejercicio (OK o FAIL), junto con la cantidad de ejercicios OK. Ejemplo:

```
$ python3 pruebas.py
ejercicio_1: OK
ejercicio_2: OK
ejercicio_3: OK

Traceback (most recent call last):
File "pruebas.py", line 237, in main
    ejercicio()
File "pruebas.py", line 148, in ejercicio_4
    assert ok
AssertionError

ejercicio_4: FAIL
ejercicio_5: OK
Cantidad de ejercicios OK: 4
```

Recordar: es condición necesaria (pero no suficiente) para aprobar el examen que haya al menos 3 ejercicios OK.

La clase Imagen

La clase `Imagen` modela una imagen digital de tamaño arbitrario.

- Una **imagen** está formada por `ancho × alto` **pixels**.

- Los pixels forman una grilla rectangular con coordenadas cartesianas `(x, y)`. Las coordenadas `(0, 0)` corresponden al pixel en la esquina superior izquierda. Las coordenadas `(ancho - 1, alto - 1)` corresponden al pixel en la esquina inferior derecha.
- Cada **pixel** tiene un color determinado. Un color se determina por 3 valores numéricos `(R, G, B)`, que corresponden a la intensidad del color rojo (`R`), verde (`G`) y azul (`B`).
- Cada uno de los valores `R`, `G`, `B` puede ir entre 0 y `valor_max`.

Por ejemplo, si `valor_max = 255`, el color `(0, 0, 0)` representa el color negro, `(255, 0, 0)` es el color rojo, `(255, 255, 255)` es el color blanco, `(128, 128, 128)` es un color gris, etc.

Descripción de las pruebas

`ejercicio_1`: Funcionamiento básico Prueba el funcionamiento básico de la clase `Imagen`. Sin esta prueba funcionando probablemente no se pueda pasar ninguna de las otras pruebas.

Métodos a implementar:

- `__init__`: Recibe `valor_max`, `ancho` y `alto`, y crea una instancia de `Imagen` con estos parámetros. Todos los pixels son inicializados con el color `(0, 0, 0)` (negro).
- `get_valor_max`, `get_ancho`, `get_alto`: Devuelven el valor de cada uno de los parámetros.
- `get`: Recibe un par de coordenadas `(x, y)` y devuelve una tupla `(R, G, B)` con el color del pixel correspondiente. Si las coordenadas recibidas están fuera de rango, devuelve el color `(0, 0, 0)`.
- `set`: Recibe un par de coordenadas `(x, y)` y un color `(R, G, B)` y asigna el color en el pixel correspondiente. Si las coordenadas recibidas están fuera de rango, no hace nada.
- `pintar`: Recibe un color y asigna ese color a todos los pixels de la imagen.

`ejercicio_2`: Escribir PPM Prueba que la imagen pueda ser guardada en un archivo con formato PPM.

PPM es un formato de imagen mucho más simple de implementar que JPG o PNG. En su forma más básica es un archivo de **texto**, que tiene la siguiente disposición:

```
P3
4 5
255
0 0 0   1 0  0   2 0  0   3 0   0
0 1 0   1 1  1   2 1  4   3 1   9
0 2 0   1 2  4   2 2 16   3 2  36
0 3 0   1 3  9   2 3 36   3 3  81
0 4 0   1 4 16   2 4 64   3 4 144
```

La primera línea siempre contiene el texto `P3` (para indicar que el archivo tiene formato PPM).

La segunda línea tiene el `ancho` y el `alto` separados por un espacio.

La tercera línea tiene el valor `valor_max`.

Luego, por cada fila de pixels de la imagen hay una línea de texto, y cada línea contiene los valores `(R, G, B)` de cada pixel, todo separado por espacios (es indiferente la cantidad de espacios utilizados).

Métodos a implementar:

- `escribir_ppm`: Recibe el nombre de un archivo y escribe en ese archivo el contenido de la imagen en formato PPM.

`ejercicio_3`: Leer PPM Prueba que podamos leer un archivo PPM.

Funciones a implementar:

- `leer_ppm`: Recibe el nombre de un archivo PPM y devuelve una `Imagen` a partir del contenido del mismo.

Nota: `leer_ppm` es una función suelta, no es un método de `Imagen`.

`ejercicio_4`: Histograma Prueba que podamos obtener el histograma de colores de una imagen, y otros datos descriptivos.

Un histograma de colores es una descripción resumida de la frecuencia en la que cada color aparece en una imagen.

Métodos a implementar:

- `histograma`: Devuelve un diccionario de la forma `{color: cantidad, ...}`, en el que las claves son cada uno de los colores presentes en la imagen y los valores son la cantidad de pixels con ese color.
- `colores_mas_frecuentes`: Devuelve una lista de tuplas con la forma `[(color, cantidad), ...]`. Cada elemento es un par de valores del histograma, y la lista está ordenada en forma descendente con respecto a la cantidad. Es decir que el primer elemento es el color más frecuente.
- `promedio`: Devuelve el color promedio (formado por el promedio de cada uno de los valores `R`, `G`, `B`, calculado con división entera).

`ejercicio_5`: Balde de pintura Prueba que podamos rellenar una región de la imagen con un color.

Queremos implementar un mecanismo similar al "Paint", cuando seleccionamos la herramienta de "balde", luego seleccionamos un color y hacemos click en un pixel de la imagen. Lo que esperamos es que se pinte la imagen con el color seleccionado solanente dentro del área en el que los pixels tienen un color uniforme.

En nuestra variante vamos a simplificar un poco y vamos a rellenar con el color `c` todos los pixels contiguos al pixel seleccionado, hasta que encontremos un pixel que ya tenía el color `c`.

Ejemplo, si tenemos esta imagen:

```
........#.
...######.
...#......
####..####
......#...
##.X.#....
.#...#....
.#..#.....
.##.#.....
..#.#.....

. = color blanco (valor_max, valor_max, valor_max)
# = color negro (0, 0, 0)
```

y luego rellenamos con color negro a partir del pixel `X`, ubicado en las coordenadas `(3, 5)`, el resultado debería ser:

```
........##
...#######
...#######
##########
#######...
######....
.#####....
.####.....
.####.....
..###.....
```

Métodos a implementar:

- `balde_de_pintura`: Recibe un par de coordenadas `(x, y)` y un color `c` (R, G, B), y pinta el pixel y todos sus vecinos del color `c`, expandiendo hasta que se encuentra con un pixel que ya tenía el color `c` previamente.

Ayuda: este algoritmo suele ser fácil de implementar en forma recursiva.