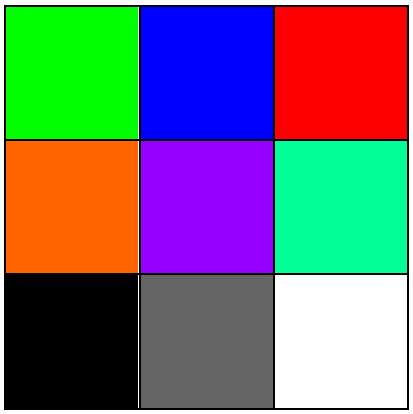
Nivel 4 Laboratorio – Visor de Imágenes

Objetivos

1. Familiarizarse con las tuplas como un tipo que agrupa datos en un único valor compuesto
2. Familiarizarse con las matrices como estructuras de datos de dos dimensiones
3. Ejercitar la implementación de algoritmos de recorrido de matrices
4. Fomentar la habilidad de descomponer un problema en subproblemas y de implementar funciones que los resuelven, lo que se conoce comúnmente como la técnica de “Dividir y Conquistar”

Contexto: Visor de imágenes

Se quiere construir una aplicación que permita visualizar imágenes en formato PNG (Portable Network Graphics) y realizar operaciones de transformación sobre estas. Para poder realizar las transformaciones, la información de cada pixel de la imagen se guarda en una matriz de tuplas, donde el color de cada píxel está representado en el sistema RGB (Red-Green-Blue), es decir un componente rojo, uno verde y uno azul. Estos componentes son representados por un número que va de 0 a 1. Por esta razón, para cada pixel, se deben guardar los 3 componentes del color correspondiente. A continuación, se muestra una imagen de 3x3 pixeles, y su respectiva representación en una matriz de tuplas:



En este ejemplo, en la posición 0, 0 de la imagen hay un pixel verde, cuyos componentes son R:
0, G: 1, B: 0

[[(0, 1, 0), (0, 0, 1), (1, 0, 0)],
[(1, 0.39, 0), (0.59, 0, 1), (0, 1, 0.78)],
[(0, 0, 0), (0.39, 0.39, 0.39), (1, 1, 1)]]

La aplicación debe permitir:

1. Visualizar una imagen PNG
2. Cargar una imagen PNG
3. Transformar una imagen a negativo
4. Transformar una imagen a escala de grises

5. Reflejar una imagen
6. Binarizar una imagen
7. Filtrar una imagen a través de la operación de convolución

Preparación

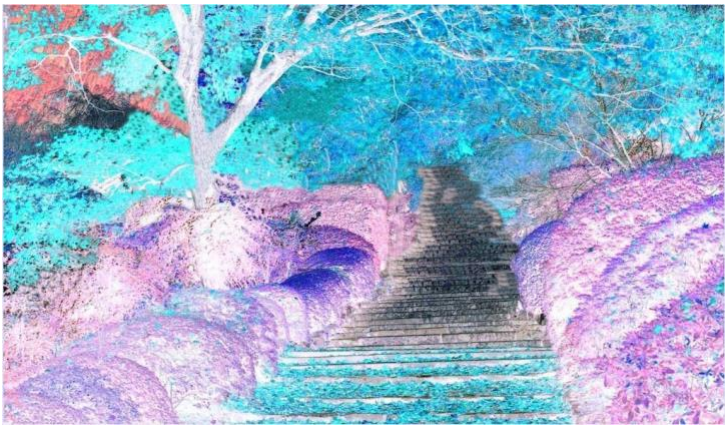
Cree una carpeta de trabajo y descargue allí el archivo `n4-l1-esqueleto.zip` que se encuentra adjunto a este enunciado en Brightspace. Descomprima este archivo y abra desde Spyder los archivos `visor_imagenes.py` y `consola_visor_imagenes.py`

Transformaciones

A continuación, se explican cada una de las transformaciones que la aplicación puede realizar. Para cada transformación se mostrará un resultado, aplicadas a la siguiente imagen:

Transformación 1: Negativo

Para calcular el negativo de una imagen se debe calcular el color negativo de cada píxel. Para ello, se debe restar 1 a cada componente (máximo valor para un componente) y tomar el valor de absoluto de dicha resta. Con los nuevos valores de los componentes se forma cada nuevo color. Recuerde que cuenta con la funicón abs( ) de Phyton.



Transformación 2: Reflejar verticalmente

Refleja la imagen verticalmente sobre una línea imaginaria en el centro de la figura, creando una imagen espejo de la figura original. Después de reflejar una figura, la distancia entre la línea de reflexión y cada punto en la figura original es la misma que la distancia entre la línea de reflexión y el punto correspondiente en la imagen de espejo. Para hacer esta transformación, se intercambian las columnas de píxeles de la imagen: La primera con la última, la segunda con penúltima, etc.

Transformación 3: Binarización

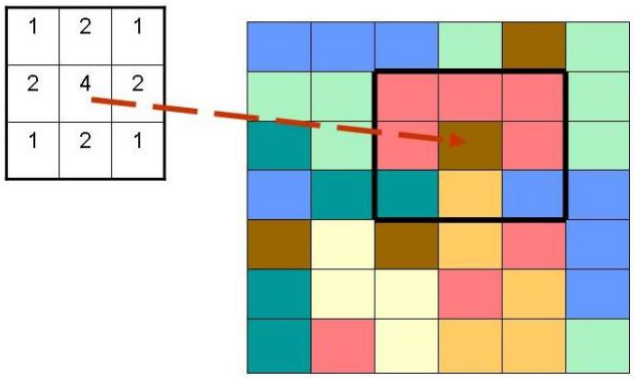
Consiste en llevar los colores de la imagen a dos colores: Negro y Blanco. Para ello, se establece un umbral (valor entre 0 y 1) y los píxeles con colores que están por encima o son iguales al umbral se cambian a blanco y los que están por debajo se cambian a negro. En este caso, se sugiere el umbral como el color promedio de la imagen.

Transformación 4: Escala de Grises

Para convertir la imagen a grises, se promedian los componentes de cada píxel y se crea un nuevo color donde cada componente (RGB) tiene el valor de dicho promedio.

Transformación 5: Convolución

La convolución consiste en hacer una serie de operaciones sobre cada píxel, que utilizan sus píxeles vecinos y unos factores que se ingresan en una matriz cuadrada de dimensión impar (3x3, 5x5, 7x7, etc.).



La idea es que cada píxel de la nueva imagen se obtenga a través de hacer operaciones con la región de píxeles vecinos y con la matriz de la siguiente manera:

- Para cada píxel, se alinea el centro de la matriz con el píxel que se está procesando. En la figura, se está ilustrando el caso en el que se aplique sobre el píxel de la tercera fila y cuarta columna, pero el proceso se tiene que hacer para todos los píxeles de la imagen, incluyendo los bordes.
- Para cada píxel vecino, se multiplica cada componente (rojo, verde y azul) con el factor de la matriz correspondiente. La región de vecinos está definida por la matriz. Por ejemplo, en el caso de la figura nos interesan los 8 vecinos y el píxel del centro porque la matriz es de 3x3. En este caso, cada uno de los componentes del píxel que está arriba a la izquierda (segunda fila, tercera columna) se va a multiplicar por 1 (ese es el valor que está en la esquina superior izquierda de la matriz. Cada uno de los componentes del píxel del centro se va a multiplicar por 4 (ese es el valor que está en el centro de la matriz).
- Para cada componente, se suman los resultados de estas multiplicaciones. Es decir que para el componente rojo se tienen que sumar los resultados de las 9 multiplicaciones de los componentes rojos, para el azul los otros 9 resultados, y para el verde los otros 9 resultados.

- Luego, las sumatorias anteriores se dividen por la suma de los factores que fueron operados. Si el píxel está cerca del borde de la imagen (tomando en cuenta la dimensión de la matriz de convolución), no aplican todos los factores de la matriz (porque no todos tienen un píxel correspondiente), sólo los que efectivamente se operaron. En el caso de nuestra figura, la suma sería 16 para todos los píxeles que no están en el borde (1+1+1+1+2+2+2+2+4).
- El color que se crea con el resultado de las operaciones anteriores para cada componente, reemplazando el color del pixel central de la región. Sin embargo, puede darse el caso de que la suma de los factores sea 0 y en ese caso no se realiza la división. Esto puede dar origen a interesantes efectos como la detección de bordes.

La operación de la imagen con la matriz de convolución debe hacerse sobre el mapa de píxeles original y no acumular las transformaciones a píxeles hechas por operaciones anteriores con la matriz. En este caso, usaremos la siguiente matriz de convolución:

1	2	1
2	3	2
1	2	1