Curso: CC3501-1 Modelación y Computación Gráfica para Ingenieros
Fecha: 12/6/2019
Profesor: Daniel Calderón
Alumno: Roberto Tapia

**Solución propuesta:**

En primer lugar se abre la imagen mediante la librería pillow para luego pasarla a un arreglo de numpy y poder así manipularla. Luego, se recorre la imagen mediante dos ciclos for anidados y se van guardando las posiciones de los pixeles blancos en una lista. Ahora se crean los arreglos que darán forma a las matrices sparse, las cuales deben ser utilizadas dado que reducen la memoria utilizada y permiten trabajar con trozos faltantes más grandes. Posteriormente se definen la función getK, que será utilizada para dar un orden a los pixeles faltantes y poder referenciarlos más fácilmente.

Terminado todo el setup anterior, se encuentran 3 ciclos for (consecutivos, no anidados), cada uno resuelve la ecuación de laplace para un color distinto para todos los pares guardados en el arreglo que contiene los pixeles faltantes. Primero se obtiene k, y luego se obtienen los k de los píxeles adyacentes. Cada pixel cae en uno (y solo uno) de los 15 casos especificados a continuación, los cuales dependen de si los pixeles que lo rodean pertenecen al trozo a reparar o no. En todos los casos se van agregando los coeficientes correspondientes a los arreglos que construiran las matrices sparse y a al arreglo b, el cual será utilizado posteriormente para resolver el sistema resultante.

Una vez termina cada ciclo, se construye la matriz sparse y se resuelve con su respectivo B. Finalmente, se vuelve a iterar una última vez sobre los pares en la lista de pixeles por arreglar, y se unen todas las soluciones encontradas a una copia de la imagen original para luego ser guardada con el nombre entregado por el usuario en la llamada al programa.

**Dificultades encontradas:**

Al momento de resolver la tarea, uno de los pasos más importantes y que resultaron más difíciles fue entender el apropiado funcionamiento de getK, del cual se habian visto análogos, pero en dichos casos se trabajaba sobre un dominio rectangular, lo que simplificaba su entendimiento, pero en este caso no encontré una forma de implementarla, hasta que sencillamente use los índices del arreglo.

Lo que probablemente tomó más tiempo fue la implementación de matrices sparse, las cuales no fueron usadas desde un principio. Se partió con arreglos de numpy y se completo la funcionalidad con ellas, pero en los test se llegó a casos que no se podían resolver de esa forma lo que llevó a una refactorización del código para implementar una solución con matrices sparse.

**Instrucciones de ejecución:**

El programa debe ser llamado desde una terminal de la siguiente forma:

python image_reconstructor.py entrada salida

Donde entrada y salida deben ser reemplazados por los nombres de los archivos que se desean reconstruir y crear, respectivamente.

**Resultados:**