



PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
ESCUELA DE INGENIERÍA
DEPARTAMENTO DE CIENCIA DE LA COMPUTACIÓN

IIC2133 — Estructuras de Datos y Algoritmos
2019 - 2

Tarea 0

Fecha de entrega código: 26 de Agosto

Fecha de entrega informe: 28 de Agosto

Objetivos

- Familiarizar al alumno con el lenguaje de programación C
- Implementar y analizar un algoritmo básico.

Introducción

La sonda espacial Yadrinni de la NASA fue lanzada hacia el centro de nuestra galaxia hace 500 años con el propósito de tomar fotos de los cuerpos celestes que encuentre en su camino y enviarlos de vuelta a la tierra. Desgraciadamente parece ser que un asteroide golpeó la sonda y dañó una de sus placas, ya que la última semana las fotos que se han estado recibiendo de la sonda han venido con píxeles faltantes. Afortunadamente están distribuidos uniformemente a lo largo de cada imagen, por lo que es posible repararlas, aunque sea parcialmente. Tu misión es programar un algoritmo que haga esto.

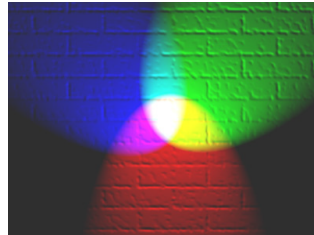


Una de las imágenes enviadas por la sonda. Los píxeles faltantes se representan como 100% blancos

Colores e Imágenes

Las imágenes son básicamente una matriz de colores, y estos a su vez pueden ser representados de múltiples maneras. En el caso de esta tarea se usará la forma estándar de representarlos: RGB.

RGB es un modelo aditivo de color en el cual los tres colores primarios de la luz, Rojo (R), Verde (G) y Azul (B) se combinan en diferentes intensidades para formar los distintos colores que podemos ver.



Las luces roja, verde y azul se combinan para formar blanco.

Esta representación es la estándar ya que es la que deben seguir los monitores para emitir la luz de distintos colores. Siguiendo estas reglas, cada color en la imagen es básicamente un vector de 3 componentes: R, G y B, llamados “canales”. Dado que el valor de cada uno representa la intensidad, y pesan 1 byte, estos van de 0 a 255.

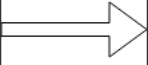
Filtro de mediana

El método para reparar la imagen se conoce como **filtro de mediana**. La manera en que funciona es la siguiente:

- Para cada píxel faltante en la imagen¹ se toman, **de la imagen original**, los valores de los 8 píxeles vecinos y el valor de sí mismo.
- Se calcula la mediana de estos 9 valores para cada canal.
- Cada canal de este píxel toma el valor de la mediana del canal correspondiente.

Tomemos un ejemplo de un canal para ver como funciona. Digamos que tenemos la siguiente imagen de 3x3 píxeles y queremos calcular el nuevo valor del píxel central como se muestra en la figura izquierda. Tomamos los valores de los vecinos en ese canal y sí mismo y calculamos la mediana. Es decir, para este canal, el píxel central toma el valor de $\text{mediana}(255, 240, 200, 233, 255, 206, 220, 231, 232) = 232$.

255	240	200
233	255	206
220	231	232



255	240	200
233	232	206
220	231	232

Ejemplo de la aproximación para el píxel central en una imagen de 3x3

¹100% blanco: R = 255, G = 255, B = 255

Pero, ¿Qué hacer cuando el píxel a calcular queda en el borde de la imagen? Para no perder información, lo que se hace es extender el valor de los bordes de la imagen hacia afuera de esta. Visto de otra forma, si se necesita un píxel que queda fuera de la imagen, se usa en su lugar el píxel más cercano que forma parte de la imagen. De forma visual:



Si realizamos esto para todos los píxeles en la imagen, obtenemos el siguiente resultado.



Imagen con información faltante



Imagen reparada con el algoritmo

Librería

Para esta tarea, el input y output serán archivos PNG con las imágenes recibidas por la sonda y las imágenes procesadas por tu algoritmo respectivamente. Para poder hacer esto, les hemos facilitado una librería escrita en C que lee y escribe archivos PNG.

La estructura **Image** es un objeto que contiene toda la información de la imagen. Es básicamente una matriz de vectores RGB. Para revisar en detalle puedes leer el archivo **imagelib.h**. No modifiques este archivo.

Los métodos disponibles se dan a continuación:

- **img_png_read_from_file(char* filename):** Abre un archivo ubicado en el path **filename** y retorna un puntero a un objeto **Image**
- **img_png_write_to_file(Image* img, char* filename):** Dado un puntero a un objeto **Image** y un path **filename**, guarda un archivo PNG con la imagen.

Input

Una vez compilado su programa debe ejecutarse de la siguiente forma:

```
./filter <input.png> <output.png>
```

Donde `<input.txt>` corresponde al *path* al archivo PNG con la imagen a reparar y `<output.txt>` corresponde al *path* del archivo donde guardaremos la imagen reparada.

Si tu programa no se cae, se generará en el path especificado un archivo PNG correspondiente al output de tu programa.

Output

Tu código será evaluado con diferentes imágenes de distintos tamaños. Para cada imagen entregada a tu programa, se espera que produzcas como output la imagen procesada por tu algoritmo. Tu solución debe ser razonablemente eficiente: para procesar cada imagen tendrás un tiempo límite de 10 segundos, si tu programa no termina en ese espacio de tiempo, tendrás 0 puntos en esa test.

Se otorgará el puntaje completo solo si tu output es **exactamente** igual al output esperado. Para que puedan evaluar eso ustedes mismos durante el desarrollo de su tarea, les hemos entregado los outputs que sus programas deben producir para que puedan comparar su output con el output que esperamos. Para hacer esto deben correr el comando:

```
diff <output.png> <output_pauta.png>
```

donde `<output.png>` es el path al archivo generado como output en su programa y `<output_pauta.png>` es el path al archivo png de la solución pauta, el cual se encontrará junto a los tests. De esta forma podrán saber si su código es correcto o no antes de subirlo (si el comando `diff` no entrega output, significa que son iguales).

Nota, aunque las fotos se vean visualmente parecidas, una diferencia de 1 pixel será considerada como incorrecta y no entregará puntaje, por lo que asegúrense de revisar usando los pasos descritos anteriormente.

Al momento de corregir tu tarea se usarán inputs distintos a los entregados durante el desarrollo de la tarea. Estos inputs de evaluación son secretos pero tienen las mismas características que los entregados para el desarrollo.

Análisis

Deberás escribir un informe de análisis donde menciones los siguientes puntos:

- Calcula y justifica la complejidad esperada de tu algoritmo en función de la cantidad de píxeles de la imagen y la cantidad de píxeles en blanco.
- Realiza un análisis de la complejidad de tu algoritmo. Utiliza gráficos para apoyar tu análisis. Contrasta los tiempos obtenidos con la complejidad teórica de tu algoritmo.

Uso de memoria

Parte de los objetivos de esta tarea es que trabajen pidiendo y liberando memoria. Para evaluar esto usaremos una herramienta llamada *Valgrind* la cual permite revisar si tienen errores de memoria y problemas de *memory leaks*. Leer la parte **6.memoria** del Taller 0 - Intro a C.

Cálculo de nota

La nota de tu tarea se descompone como se detalla a continuación:

- 80% A la nota de tu código, dividido en:
 - 80% a que el output de tu programa sea correcto
 - 10% a que **valgrind** diga que tu programa no tiene errores de memoria
 - 10% a que **valgrind** diga que tu programa no tiene *memory leaks*
- 20% A la nota de tu informe, dividido en:
 - 30% Complejidad teórica y justificación.
 - 70% Análisis y contraste en base a tiempos de ejecución.

Entrega

- **Código:** GIT - Repositorio asignado (asegúrate de seguir la estructura inicial de éste).
 - En la carpeta *Programa* debe encontrarse el código.
 - Se recogerá el estado en la rama *master*.
 - Se espera que el código compile con **make** dentro de la carpeta *Programa* y genere un ejecutable de nombre **filter** en esa misma carpeta.
- **Informe:** SIDING - En la encuesta correspondiente, en formato PDF
- **Hora Límite:** 1 minuto pasadas las 23:59 del día de la entrega.
- No se permitirán entregas atrasadas, **seremos estrictos con esto** por lo que no dejen para último instante el subir su tarea.