Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

IIC2133 — Estructuras de Datos y Algoritmos
2021 - 1

Tarea 1

Fecha de entrega código e informe: Martes 27 de Abril

Objetivos

- Investigar sobre los árboles *Max Tree* y *Component Tree*.
- Representar una imagen como un árbol y aplicar filtros sobre él.

Introducción

Después de superar el desastre bioquímico causado en la empresa de papel Dunder Mifflin, **Kevin Malone** fue despedido de su puesto dejándolo a la deriva. Pero manteniendo la cabeza en alto, Kevin decide que esta es una nueva oportunidad que le ofrece la vida para poder dedicarse a su antiguo sueño, crear **arte gráfico**.

Kevin contactó a su primo Post Malone, quién le dijo que el artista chileno `Gepe` anda en busca de una portada para su nuevo álbum. A pesar de haber cobrado bastante dinero en sus últimos conciertos, `Gepe` aún no logra encontrar una portada que le guste.

La compañía musical que representa a `Gepe`, *My Guitar*, ha rechazado tajantemente varias propuestas de artistas, y en un acto de desesperación, **le ha pedido a Kevin que diseñe la portada del álbum de `Gepe` usando una técnica muy moderna que consiste en aplicar filtros a imágenes reales**.[1]

El problema es que Kevin no tiene ningún talento artístico ni computacional y él lo sabe, por lo que acude a ti, maestro de `C`, para crear un código que transforme imágenes reales en escala de grises a imágenes que simulen una obra de arte moderna, aplicando **dos filtros especiales.**

[1] Con este enunciado no buscamos ofender a nadie, sino que sacarles una sonrisa para que empiecen con entusiasmo a programar :)

Problema

En esta tarea, deberás construir un ***Max Tree*** para segmentar una **imagen en escala de grises**, y luego utilizar este árbol para generar una nueva imagen, a la cual se le aplicará un **filtro** determinado.

Figura 1: Foto original y los resultados tras aplicar dos filtros distintos.

Imagen de Input

El programa recibe como input una imagen en **escala de grises**, de dimensiones variables. Cada píxel de la imagen tiene un valor de **grisáceo**, el que varía entre 0 y 127, siendo **0 negro y 127 blanco**. Para cada valor grisáceo entre 0 y 127, se cumple que a medida que aumenta su valor representa un gris más claro. Por ejemplo, el grisáceo 40 es más oscuro que el grisáceo 80.

Además, cada píxel tiene **vecinos**. Por facilidad, asumiremos que un píxel puede tener a lo más 4 vecinos, siendo ellos los ubicados inmediatamente hacia la izquierda, la derecha, arriba y abajo. A continuación, se visualiza un ejemplo:

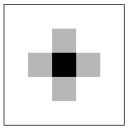

Figura 2: Los vecinos del píxel negro son los 4 coloreados con gris.

Otro concepto importante es **vecindario**. Un vecindario es un conjunto de píxeles que se unen entre sí dado que cada uno de ellos es vecino de al menos otro píxel del vecindario.

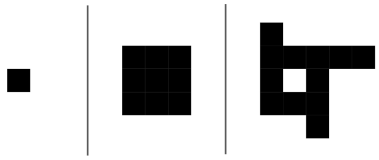

Figura 3: Tres configuraciones que cumplen con la definición de vecindario.

Por último, utilizaremos el concepto de Umbral `U` para referirnos a un valor de grisáceo `mínimo`.

Creación del *Max Tree*

La primera parte de esta tarea consiste en construir un *Max Tree*. Si bien debes investigar por tu propia cuenta cómo funciona este árbol, a continuación aclaramos temas relevantes:

Sea `P` un nodo y `U` un umbral del *Max Tree*. Entonces:

- El nodo `P` se asocia a un vecindario conformado por píxeles cuyo grisáceo es `mayor o igual` al umbral `U`.
- **El nodo `P` sólo guarda los píxeles de su vecindario con un grisáceo igual a U**. Los píxeles mayores a `U` se distribuyen entre los hijos del nodo. El nodo P puede tener una cantidad variable de nodos hijos (≥ 0).
- Los vecindarios de los hijos del nodo `P` son independientes entre sí.
- Se omite un nodo si no guarda píxeles aún teniendo un vecindario. Esto ocurre por ejemplo, si un vecindario descendiente del nodo P estuviese conformado sólo por píxeles con un grisáceo igual a `U + 2`. En ese caso, se omitiría el nodo con umbral `U + 1` y el nodo `P` se asociaría directamente al nodo con un umbral igual a `U + 2`.

Por ejemplo, el nodo raíz, a profundidad 0, tiene como vecindario a todos los píxeles de la imagen, pues todos cumplen con tener un grisáceo mayor o igual a 0. Sin embargo, el nodo raíz solo almacena a los píxeles negros del vecindario. Los píxeles no-negros se transfieren a sus descendientes.

Es interesante notar que la profundidad máxima que puede alcanzar el *Max Tree* a través de alguna rama es 127, debido a que hay 128 tonos de gris.

A continuación vemos un ejemplo gráfico:

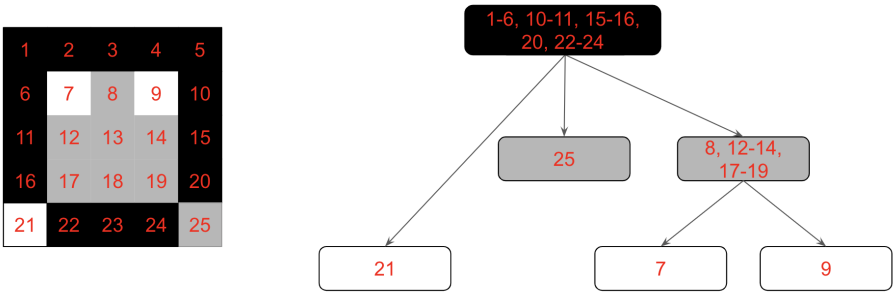

Figura 4: Imagen de 3 colores y su respectivo *Max Tree*, en donde el color del nodo corresponde a su umbral.

Como se puede apreciar, al segmentarse un nodo padre `P` con un umbral `U`, los píxeles con un grisáceo mayor a `U` se reparten entre los nodos hijos de `P` según los vecindarios que se vayan formando.

Analogía del Problema

Si te cuesta entender el problema, imagínalo como si fuera una **inundación de agua** en un mapa geográfico. La imagen corresponde al mapa geográfico, en donde los píxeles negros son el suelo y los píxeles no-negros son colinas, que son más altas a medida que son más blancas. Además, supón que cada hora crece el nivel del agua y tienes que registrar cuáles son los sectores más bajos de tierra. Usemos como ejemplo a la imagen de la `Figura 3`.

Al inicio, ves todo el mapa, pues nada se ha inundado, así que registras todos los píxeles negros que corresponden al nivel del mar. A la hora siguiente (umbral gris), el suelo (píxeles negros) ha desaparecido y se formaron 3 islas (vecindarios), pero solo registras los vecindarios que tienen el nivel más bajo de tierra (gris).

A la segunda hora (umbral blanco), subió nuevamente el nivel del agua y solo quedan 3 islas chicas (7, 9 y 21) . Finalmente, a la tercera hora, ya hay pura agua. A continuación lo visualizamos gráficamente:

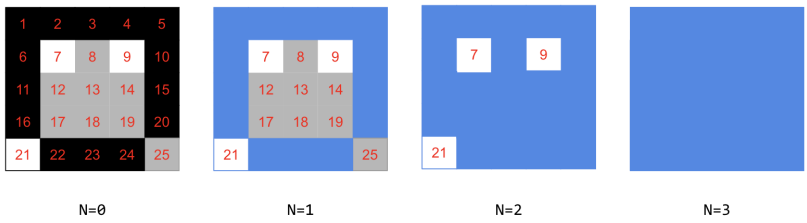

Figura 5: Progresión de cómo el agua inunda el mapa geográfico.

Filtros

Una vez creado el *Max Tree* de una imagen, aplicaremos uno de los siguientes filtros para crear una obra de arte gráfica.

1. **Area Filter**

   Este filtro selecciona los vecindarios asociados a nodos del árbol de una imagen que cumplen con dos condiciones:

   (a) Cada píxel del vecindario es mayor a cierto valor de grisáceo `G`.

   (b) La cantidad del píxeles del vecindario es mayor a una cantidad `A`,

   Si el vecindario cumple con ambas condiciones, sus píxeles mantienen sus colores. En cambio, si un vecindario no cumple con alguna condición, los píxeles pasan a tener el tono de gris que tiene su padre en ese momento. Si dicho píxel es negro, se mantiene en cero su grisáceo.

   Veamos un ejemplo que tiene un `G = 1` y un `A = 2`.

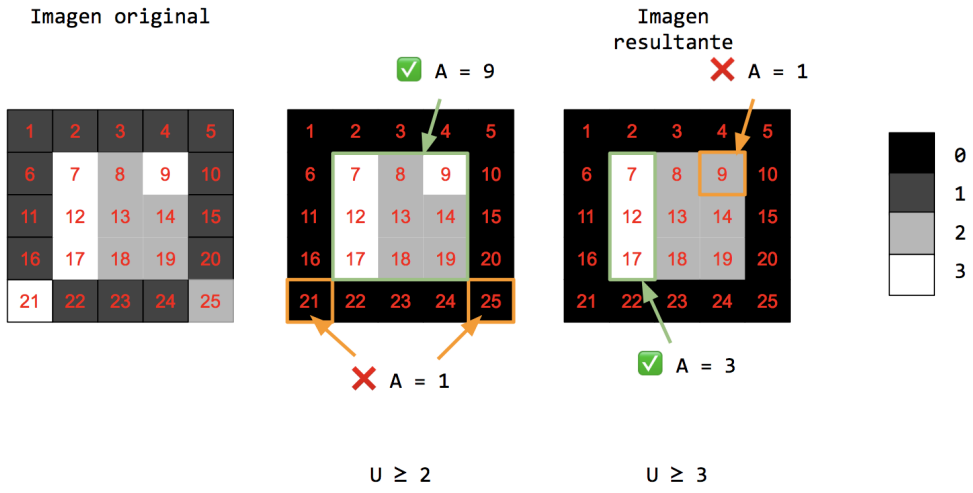

Figura 6: Progresión del filtro a lo largo del árbol.

Analicemos esta imagen por umbral:

- Como no hay ningún píxel con `U = 0`, el nodo raíz se inicia con `U` ≥ 1. Los vecindarios con `U = 1` se colorean de negro puesto que no cumplen con la condición de ser mayor a `1`.
- Con `U` ≥ 2:
  - Los vecindarios {21} y {25} no cumplen con `A > 2`, por lo que se pintan de negro como su padre (El padre antes tenía tonalidad 1, pero pasó a 0 porque no cumple con la condición del filtro).
  - El conjunto {7-9, 12-14, 17-19} cumple con ambas condiciones, por lo que se mantienen los grisáceos de los píxeles.
- Con `U` ≥ 3:
  - El vecindario {7, 12, 17} cumplen con ambos requisitos, por lo que mantienen sus colores.
  - Sin embargo, el vecindario {9} no cumple con `A > 2`, por lo que le debemos asignar el tono de su padre, correspondiente a `2`.

2. **Delta Filter**

   Este filtro selecciona los vecindarios cuya diferencia de cantidad de píxeles con respecto al vecindario de su padre es menor a un `D` calculado como:

$$D = \frac{\#\text{píxeles padre} - \#\text{ píxeles hijo}}{\#\text{ píxeles padre}}$$

   Al igual que en el filtro anterior, los vecindarios que no cumplen con la condición pasan a tener el tono del padre del vecindario, mientras que los que sí cumplen con el `D` mantienen tu grisáceo. El vecindario asociado al nodo raíz no se considera, puesto a que la raíz no tiene un padre. Además, si no hay variación con el nodo padre, no se aplica el filtro.

   Veamos un ejemplo que tiene un `D = 0.7`:

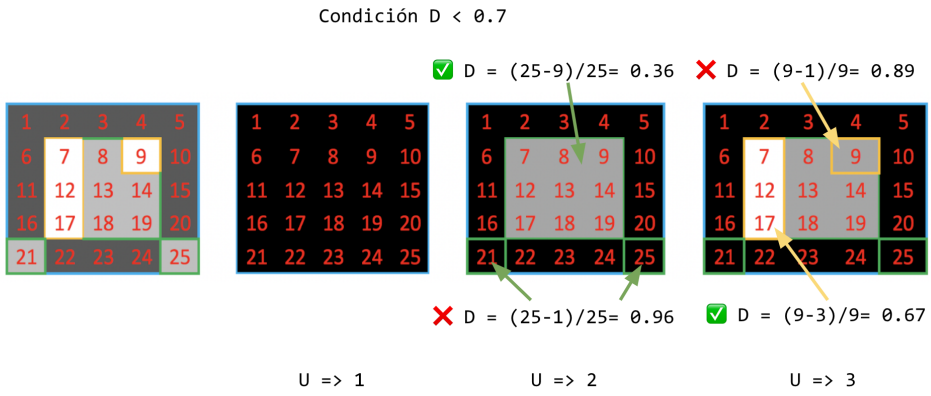

Figura 7: Progresión de este filtro a lo largo del árbol

   Analicemos esta imagen por umbrales:

   - Como no hay ningún píxel con `U = 0`, el nodo raíz se inicia con `U` ≥ 1. Con `U` ≥ 1, el vecindario contiene 25 píxeles. Como no hay variación no el anterior, no se aplica el filtro puesto que el nodo raíz no tiene padre.
   - Con `U` ≥ 2, el vecindario {7-9, 12-14, 17-19} mantiene su color, puesto a que su `D = 0.36` es menor a `0.7`. Los vecindarios hijos {21} y {25} tienen un `D = 0.96 > 0.7`, por lo que toman el color del nodo anterior, es decir, negro.
   - Con `U` ≥ 3, el vecindario {7, 12, 17} se mantiene pues tiene un `D = 0.67 < 0.7`. Sin embargo, el vecindario {9} pasa a tener el color de su antecesor puesto que su `D = 0.98 > 0.7`.

Ejecución

Tu programa se debe compilar con el comando `make` y debe generar un binario de nombre `filter` que se ejecuta con el siguiente comando:

```
./filter <input.png> <output.png> <filtro> <d/A> <G>
```

Donde `input` será una imagen a filtrar **siempre en formato PNG**, `output` será la imagen final y `filter` el filtro que deben aplicar. Este puede ser `área` o `delta`. Además, recibirás uno o dos parámetros extras que representan las variables `d`, `G` y `A` según corresponda.
Tu tarea será ejecutada con *tests* de dificultad creciente, asignando puntaje a cada una de las ejecuciones que tenga un `output` igual al esperado.

Output

El output de tu programa corresponde la imagen obtenida luego de filtrar. Para esto, el código base contiene una imagen en negro donde debes asignar el valor correspondiente a cada pixel en el array `Image->pixels`.

Código Base y Setup

Les entregamos el código con un módulo llamado `image`, que se encarga de preprocesar la imagen en escala de grises antes de crear el ***Max Tree***. Dentro de sus funcionalidades se encuentra el variar los valores de grisáceo entre 128 tonalidades.

Para que puedas utilizar este modulo, debes **instalar la librería libpng**.

El código base de la tarea incluye funciones que procesan la imagen de input. Se incluye un struct `Image` que contiene lo siguiente:

- `height:` altura de la imagen
- `width:` ancho de la imagen
- `pixel_count:` cantidad total de pixeles (height $\cdot$ width)
- `pixels:` array de `int` con el valor de gris cada pixel.

Análisis

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

- Un ***Component Tree*** es similar a un ***Max Tree***, pero se diferencia de este último en que cada nodo guarda todos los píxeles que tienen sus descendientes. En esta tarea, por ejemplo, un nodo guardaría todos los píxeles del vecindario, en vez de solo los píxeles cuyo grisáceo es igual umbral del nodo.

  Compara un ***Component Tree*** con un ***Max Tree***. ¿Cuál es más eficiente? ¿Qué ventajas y desventajas tiene cada uno? ¿En qué situaciones es más conveniente usar uno y no el otro?

- Calcula y justifica la complejidad en notación $\mathcal{O}$ para la implementación en términos de la cantidad de píxeles y la profundidad del árbol.

  Específicamente, analiza la complejidad de la construcción del árbol y la implementación de cada filtro.

Evaluación

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

- 70% a la nota de tu código: que retorne el `output` correcto. Para esto, tu programa será ejecutado con archivos de input de creciente dificultad, los cuales tendrán que ser ejecutados en menos de 10 segundos cada uno.
- 30% a la nota del informe, que debe contener tu análisis.

Entrega

Código: GIT - Repositorio asignado. Se entrega a más tardar el día de entrega a las 23:59 hora de Chile continental.

Informe: SIDING - En el cuestionario correspondiente, en formato PDF. Sigue las instrucciones del cuestionario. Se entrega a más tardar el día de entrega a las 23:59 hora de Chile continental.

Bonus

Los bonus sólo aplican si la nota respectiva es mayor o igual a 5.0.

- **Manejo de memoria perfecto:** (5 décimas a la nota de código) obtendrás este bonus si solicitan y liberan memoria de manera perfecta, es decir, **valgrind** reporta en tu código 0 leaks y 0 errores de memoria en todos los archivos de input. Se aplicará este bonus solo si el output de tu programa es correcto.