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

IIC2133 — Estructuras de Datos y Algoritmos
2021 - 2

Tarea 1

Fecha de entrega código e informe: 7 de octubre 23:59hrs

Objetivos

- Investigar sobre el Component Tree.
- Representar una imagen como árbol y aplicar filtros sobre él.

Introducción

Debido al arduo trabajo que puso el presidente del centro de alumnos de hechicería (CAH), Peter Potter, en organizar y simular los viajes de los alumnos de DCCWarts en tren mágico, Peter no invirtió mucho tiempo en sus cursos, por lo que estuvo a punto de echarse la tripleta en DCCwarts (Desafíos de la Hechicería, Conjuros I y Química para hechiceros). Es por esto que decidió tomarse las cosas con más calma, eligiendo tomar un OFM pasta (optativo de formación muggle) de fotografía que parecía muy interesante. Sin embargo, al partir el curso se da cuenta de que el ramo no es para nada fácil, le piden editar fotos con filtros extraños ocupando técnicas basadas en árboles y Peter es nulo para la tecnología. Es por esto que te pide ayuda a ti, experto en tecnologías muggle y explorador de una de las lenguas antiguas requeridas (C), para poder aplicar filtros a imágenes reales ocupando la menor memoria posible, en el menor tiempo posible.

Figura 1: El laptop de Peter luego de intentar usar C para modelar los traslados a Hogwarts, y posteriormente intentar aplicar filtros a imágenes en su OFM

Problema

El problema consiste en modelar un **Component Tree**, segmentando una **imagen en escalas de grises**, y luego utilizar este árbol para generar una nueva imágen, a la cual se le aplicará un filtro determinado.

Figura 2: Imagen original, zona marcada e imagen editada con filtro de un sauce boxeador.

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:

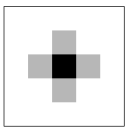

Figura 3: Los vecinos del píxel negro son los 4 coloreados con gris.

Otro concepto importante es el **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.

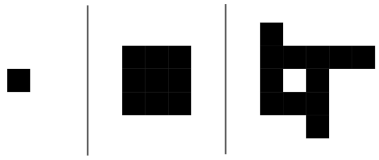

Figura 4: Tres configuraciones que cumplen con la definición de vecindario.

Por último, utilizaremos el concepto de Umbral `U` para referirnos a un valor de grisáceo `mínimo`. A continuación, en la `Figura 5` se visualiza un ejemplo gráfico de los vecindarios pertenecientes a los distintos umbrales existentes en una imagen. En esta figura se muestra, de izquierda a derecha, el vecindario con umbral 0 (color negro), un vecindario con umbral 30 (un tipo de gris oscuro), los vecindarios resultantes con umbral 90 (un tipo de gris más claro), y los vecindarios con umbral 127 (color blanco). Los vecindarios se encuentran remarcados con color verde y las etiquetas rojas representan los grisáceos de los trozos de la imagen que se muestra.

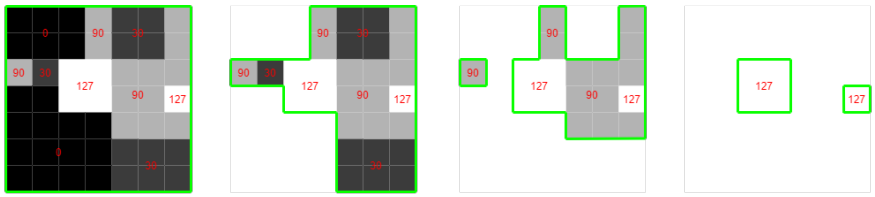

Figura 5: Ejemplos de vecindarios con distintos umbrales para una misma imagen

Como puedes observar, hay 2 vecindarios con umbral 90 y 2 vecindarios con umbral 127.

Creación del *Component Tree*

La primera parte de esta tarea consiste en construir un *Component 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 *Component Tree*. Entonces:

- El nodo `P` se asocia a un vecindario conformado por píxeles cuyo grisáceo es `mayor o igual` al umbral `U`.
- El vecindario del nodo raíz posee todos los píxeles de la imagen.
- **El nodo `P` guarda todos los píxeles de su vecindario**. El nodo P puede tener una cantidad variable de nodos hijos (≥ 0).
- Un nodo Q es hijo de P si se cumple que:

  vecindario del nodo `Q` $\subset$ vecindario del nodo `P`.
- Se omite un nodo si este guarda los mismos píxeles que su padre. Esto ocurre por ejemplo, si un vecindario descendiente del nodo P estuviese conformado solo por píxeles con un grisáceo mayor o 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. Luego, si los pixeles con umbral `U`=1 se encuentran separados, es decir, que no pertenecen al mismo vecindario, entonces existen tantos nodos como vecindarios de umbral 1 haya presentes, y los nodos de grisáceo 1 guardan a todos los pixeles pertenecientes a su respectivo vecindario.

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

A continuación vemos un ejemplo gráfico:

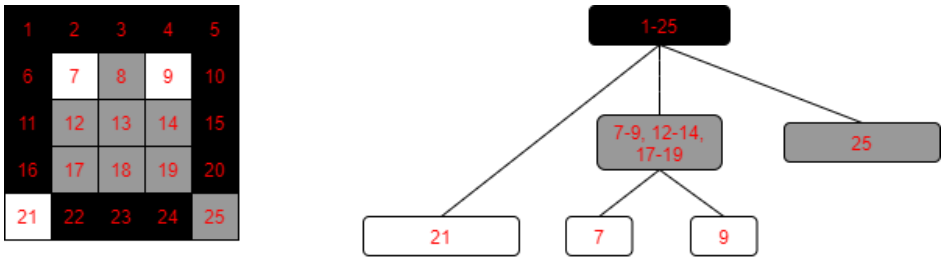

Figura 6: Imagen de 3 colores y su respectivo Component Tree, en donde el color del nodo corresponde a su umbral. Las etiquetas rojas corresponden al identificador de los pixeles y los pixeles van del 1 al 25.

Como se puede apreciar, al formar un nodo con un umbral `U`, este incluye a todos los píxeles que poseen un grisáceo mayor o igual a su umbral. Si se tiene un nodo N_1 y otro nodo N_2 que es descendiente de N_1 entonces se puede afirmar que $N_2 \subset N_1$.

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 que no se han inundado todavía. Usemos como ejemplo a la imagen de la `Figura 6`.

Al inicio, ves todo el mapa, pues nada se ha inundado, así que registras todos los píxeles, que no se han inundado todavía. A la hora siguiente (umbral gris), el suelo (píxeles negros) ha desaparecido y se formaron 3 islas (vecindarios), entonces registras las zonas que no se han inundado (las 3 islas completas). A la segunda hora (umbral blanco), subió nuevamente el nivel del agua y solo quedan 3 islas chicas (7, 9 y 21), las cuales procedes a registrar. Finalmente, a la tercera hora, ya solo hay agua. A continuación lo visualizamos gráficamente:

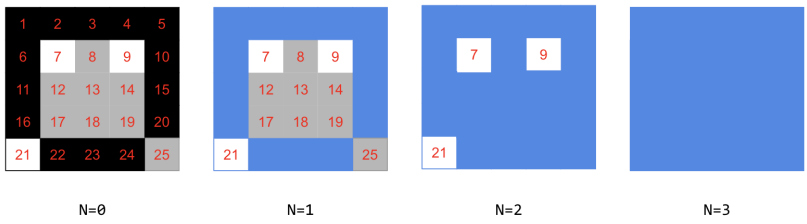

Figura 7: Progresión de cómo el agua inunda el mapa geográfico.

Filtros

Una vez creado el *Component Tree* de una imagen, aplicaremos un filtro para crear una obra de arte gráfica.

1. **Alpha Filter**

   Este filtro lo que busca es capturar un % de la imagen en relación al marker asignado según corresponda. A menor alpha extenderemos el vecindario que queremos recrear, mientras que a un nivel de alpha mayor buscaremos ser más puristas con el output en relación a nuestro marcador. A niveles

cercanos a 1 toma en cuenta el tamaño del vecindario de los marcadores evitando missclicks en los marcadores.
Para que esto funcione bien deberás investigar,o acá. Cuáles son las condiciones que deben tener los nodos de component tree para agregarlos o no a los vecindarios que finalmente serán incluidos en la imagen final, al igual que cuando corresponda pintar los pixeles con grises distintos al original. Respecto a los valores alpha que deberás utilizar para cada imagen no debes preocuparte, en cada test se dirá el alpha a utilizar.

Veamos unos ejemplos de como variar el parámetro `alpha` influye sobre el resultado.

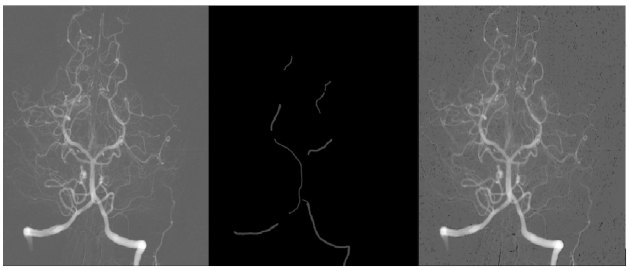

Figura 8: Imagen original - Marcador de la imagen - Imagen out.png alpha = 0

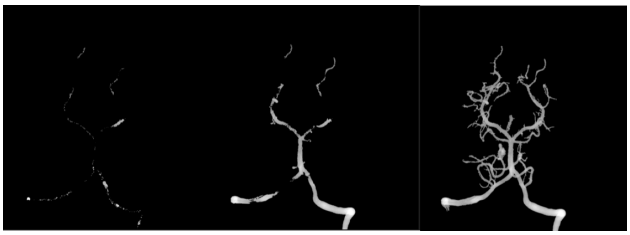

Figura 9: Imágenes out con variados Alpha: 0.5 - 0.1 - 0.01

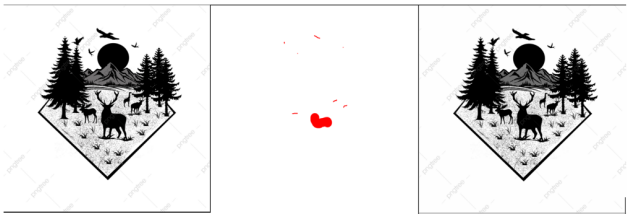

Figura 10: Imagen original - Marcador de la imagen - Imagen out.png alpha = 0

Figura 11: Imágenes out con variados Alpha: 0.1 - 0.01 - 0.5

En la figura 4 podemos notar como con un alpha alto obviamos los posibles missclicks del marcador, mientras que con un alpha muy pequeño incluimos figuras a la imagen que no aparecen en los marcadores cumplen con las condiciones.

Ejecución

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

```
./potterphoto <original_image.png> <output_image.png> <marked_image.png> <alpha> opcional <R>
```

Donde `original_image` será la imagen original a la cual hay que aplicar el filtro **siempre en formato PNG**, `output_image` será la imagen final luego de aplicarle el filtro, `marked_image` será la zona de la imagen marcada, y `alpha` es un margen para el filtro que deben aplicar, este último es un valor entre 0 o 1 que busca evitar falsos positivos / negativos.
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 ***Component Tree***. Dentro de sus funcionalidades se encuentra el variar los valores de grisáceo entre 128 tonalidades.

Para que puedas utilizar este módulo, 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 píxel.

No es necesario que modifiquen este código, simplemente esta para facilitarles la vida en la lectura y reconstrucción de la imagen.

Recomendación de tus ayudantes

Las tareas requieren de mucha dedicación de tiempo generalmente, por lo que desde ya te recomendamos distribuir tu tiempo considerando los plazos definidos. Así mismo, te recomendamos fuertemente que antes de empezar a programar tu tarea, leas el enunciado y te dediques a entender de manera profunda lo que te pedimos. Una vez que hayas comprendido el enunciado, dedica el tiempo que sea necesario para la planificación y modelación de tu solución, para posteriormente poder programar de manera eficiente. Estos son consejos de tus ayudantes que te puede ayudar a pasar el ramo :)

Uso de memoria

Parte de los objetivos de esta tarea es que trabajen solicitando y liberando memoria manualmente. Para evaluar esto, usaremos *valgrind*. Se recomienda fuertemente ver los videos de este repositorio.

Para asegurarte que no tengas errores de memoria debes correr tu programa con:

```
valgrind ./potterphoto <original_image.png> <output_image.png> <marked_image.png> <alpha> opcional <R>
```

y el output debe contener

`"All heap blocks were freed -- no leaks are possible"` y
`"ERROR SUMMARY: 0 errors from 0 contexts"`

Análisis

Deberás escribir un informe de análisis[1] donde menciones los siguientes puntos:

- Explica ¿Por qué es conveniente usar un ***Component Tree***? ¿Qué ventajas y desventajas tiene?
- 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 del 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.
  La nota se dividirá de la siguiente manera :
  - 70% Tests Subdividos tal que :
    - 35% Easy
    - 35% Medium
    - 20% Hard

[1] en un máximo de 5 planas

- 30% a la nota del informe, que debe contener tu análisis.
- Bonus, si tienes manejo perfecto de memoria sin leaks, acorde a valgrind obtendrás un bonus del 5% sobre tu nota de código.
  Para acceder a este bonus debes tener nota 4 mínima en el apartado de código.

Entrega

Código: GIT - Rama master del 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.

Atraso: A lo largo del semestre tendrás 4 días de gracia, los cuales podrás utilizar en caso de que no alcances a entregar alguna tarea en el tiempo indicado. Para esto, puedes usar un máximo de 2 días en una tarea, sin importar si tienes más días disponibles, y tendrás que avisar si quieres que se revise un commit posterior a la fecha de entrega en el formulario que se mandará el día siguiente. Cabe destacar que si entregas a las 00:01 hrs perderás un día en caso de llenar el formulario, y no será revisado ese commit en caso de que decidas no contestarlo. Por otro lado, si se te acaban los 4 días y entregas una tarea atrasada, entonces tendrás la calificación mínima **sin derecho a reclamo.** Cabe recalcar que el informe solo se puede realizar en un **máximo de 5 hojas**, de pasarse no podrán objetar por posibles descuentos.

Integridad académica

Este curso se adscribe al Código de Honor establecido por la Escuela de Ingeniería. Todo trabajo evaluado en este curso debe ser hecho **individualmente** por el alumno y **sin apoyo de terceros**. Se espera que los alumnos mantengan altos estándares de honestidad académica, acorde al Código de Honor de la Universidad. Cualquier acto deshonesto o fraude académico está prohibido; los alumnos que incurran en este tipo de acciones se exponen a un Procedimiento Sumario. Es responsabilidad de cada alumno conocer y respetar el documento sobre Integridad Académica publicado por la Dirección de Pregrado de la Escuela de Ingeniería.