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

Tarea 1

IIC2133 - Estructuras de Datos y Algoritmos

Primer semestre, 2018

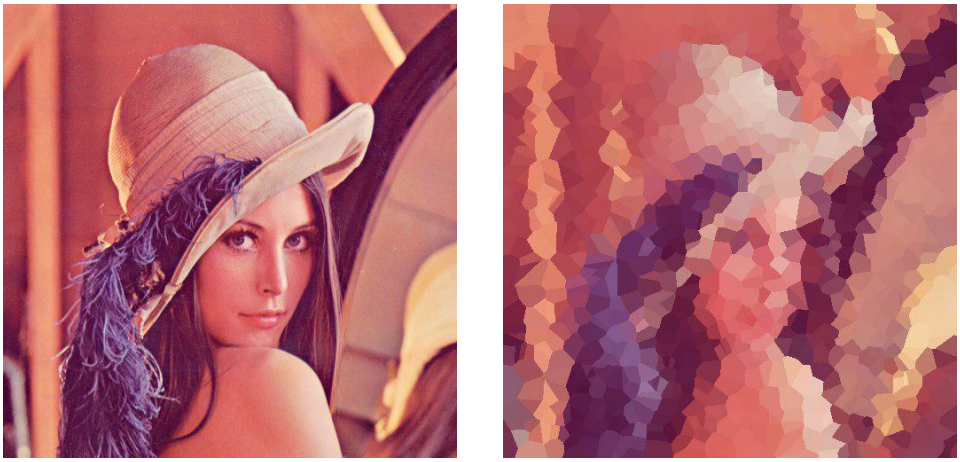
Entrega código: Martes 17 de Abril
Entrega informe: Viernes 20 de Abril

Objetivos

- Jerarquizar un conjunto de datos de manera espacial dentro de una estructura.
- Aprovechar la jerarquía de la estructura para hacer más eficiente operaciones de búsqueda sobre los datos.
- Analizar el impacto de distintas heurísticas de construcción de la estructura sobre la eficiencia de la búsqueda.

Introducción

La empresa *Abobe* te ha contactado para que los ayudes a hacer más eficiente su implementación del filtro "catedral" de su aplicación de edición fotográfica **FotoChop**, el cual consiste en generar un *diagrama de voronoi* sobre la imagen, de manera de lograr el siguiente efecto.



La imagen original, junto con el resultado de aplicar el filtro "catedral"

Voronoi

Un diagrama de Voronoi (también conocido como teselación, descomposición o partición de Voronoi) consiste en una partición de un plano en regiones de acuerdo a la distancia a puntos llamados núcleos previamente determinados. El diagrama divide el plano en regiones dadas por los núcleos, donde todo punto dentro de la región está más cerca de ese núcleo que de cualquier otro.

El filtro

El filtro consiste en lo siguiente:

Para una imagen $\mathcal{I}$, una cantidad de núcleos n y una semilla aleatoria s:

1. Generar aleatoriamente de acuerdo a s un conjunto de n núcleos $\mathcal{N}$.
2. Para cada píxel $p \in \mathcal{I}$, buscar el núcleo $x \in \mathcal{N}$ más cercano a p. Asociar p a x.
3. Para cada núcleo $x \in \mathcal{N}$:
   - Sea c el promedio de los colores de todos los píxeles asociados a x.
   - Pintar cada píxel asociado a x con el color c

La implementación actual

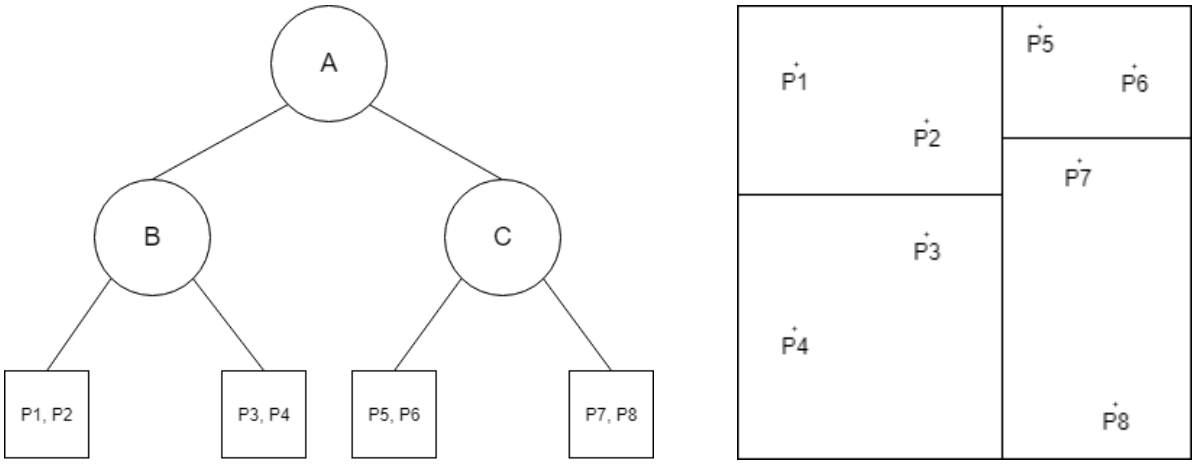
Actualmente el código de *Abobe* implementa este algoritmo correctamente, generando el conjunto de núcleos $\mathcal{N}$ al comenzar el programa. El problema es que el paso 1 se computa con fuerza bruta, es decir, calcula la distancia de cada píxel a cada uno de los núcleos, y así obtiene el núcleo más cercano. En otras palabras, para una imagen de $H \times W$ y N núcleos, se necesita $H \times W \times N$ cálculos de distancia. Esto resulta muy ineficiente en la práctica, especialmente para imágenes muy grandes. Es por esto que es necesario implementar una solución de menor complejidad computacional.

KDTree

Para hacer una solución eficiente es necesario considerar que la complejidad está mayoritariamente dominada por la cantidad de píxeles de la imagen. Esto significa que podemos invertir tiempo preprocesando los datos para poder hacer más eficiente las operaciones de búsqueda de cual núcleo es el más cercano a cada píxel.

Una estructura de datos que permite hacer eficientemente esto es un KDTree o K-Dimentional Tree. Este es un árbol binario que agrupa los datos según posiciones en un espacio de K dimensiones. En este caso nos interesa dividir los núcleos en un espacio de 2 dimensiones.

Cada nodo del KDTree representa una caja k-dimensional que contiene distintos núcleos. Los hijos de un nodo también son cajas que están separadas por un plano divisor, están contenidas dentro del nodo padre y contienen a todos los núcleos del padre:



KDTree de 2 dimensiones y su representación en el plano.

El KDTree es útil para este problema ya que al buscar el núcleo más cercano a un píxel no es necesario comparar con todos los núcleos, sino que es posible hacer una búsqueda inteligente sobre el árbol y calcular la distancia de solo los núcleos necesarios.

Tu deber es investigar esta estructura e implementarla para hacer el proceso más eficiente. Se recomienda buscar sobre su uso en búsqueda de vecinos más cercanos.

Informe y análisis

Debes hacer un informe que contenga un análisis teórico y empírico de tu solución. En particular, se espera lo siguiente:

Análisis empírico

Debes hacer un análisis de tiempo de tu programa para ver cómo se comporta al variar los siguientes parámetros:

- Cantidad de núcleos.
- Tamaño de la imagen (número de píxeles).

También debes analizar los siguientes criterios de construcción del árbol:

- Probar al menos 2 criterios para determinar cuándo una caja del KDTree deja de dividirse.
- Probar dividir las cajas según la mediana de los datos, además de otra métrica que tenga sentido.

Todo análisis y comparación debe estar respaldado con los gráficos correspondientes, además de la teoría que justifica los resultados obtenidos. ¿Se cumplieron las expectativas teóricas?

Input

Tu programa recibe los siguientes parámetros:

```
./filter [imagen] [N] [seed]
```

Donde `imagen` es la imagen a aplicar el filtro (en formato PNG), `N` es el número de núcleos, y `seed` es una semilla aleatoria para la generación de los puntos.

Output

El output de tu programa debe ser la imagen filtrada mostrada en la interfaz gráfica que se te provee. Para esto se te darán comandos para interactuar con la interfaz.

Interfaz

La interfaz funciona como una librería externa a tu programa la cual no debes modificar. Tiene distintas funciones que puedes usar para debuguear y para mostrar el resultado de tu programa.
Las funciones de la librería que vas a necesitar son:

- `watcher_open(height, width)`: Abre una ventana para mostrar una imagen de las dimensiones dadas.
- `watcher_set_color(R, G, B)`: Selecciona un color para las siguientes operaciones de dibujo.
- `watcher_draw_segment(xi, yi, xf, yf)`: Dibuja un segmento de recta de un punto a otro del color seleccionado.
- `watcher_paint_pixel(row, col)`: Pinta el pixel del color seleccionado.
- `watcher_snapshot(filename)`: Genera una imagen PNG con el contenido actual de la ventana.
- `watcher_close()`: Cierra y libera los recursos de la ventana.

Evaluación

La nota de tu tarea está descompuesta en dos partes:

- 50 % corresponde a que tu código pase los tests dentro del tiempo máximo.
- 50 % corresponde a la nota de tu informe.

Se probará tu programa con distintos test de dificultad creciente. Tu programa deberá ser capaz de obtener el output los tests dentro de un tiempo acorde a la complejidad esperada. Pasado ese tiempo el programa será terminado y se asignarán 0 puntos en ese test.

Entrega

Deberás entregar tu tarea en el repositorio que se te será asignado; asegúrate de seguir la estructura inicial de éste.

Se espera que tu código compile con `make` dentro de la carpeta **Programa** y genere un ejecutable de nombre `filter` en esa misma carpeta. **No modifiques código fuera de la carpeta *src/filter*.**

Se espera que dentro de la carpeta **Informe** entregues tu informe en formato *PDF*, con el nombre *Informe.pdf*

Estar reglas ayudan a recolectar y corregir las tareas de forma automática, por lo que su incumplimiento implica un descuento en tu nota.

Se recogerá el estado de la rama `master` de tu repositorio, 1 minuto pasadas las 23:59 horas del día de entrega (tanto para el código como para el informe). Recuerda dejar ahí la versión final de tu tarea. No se permitirá entregar tareas atrasadas.

Bonus

Buen uso del espacio y del formato (+5 % a la nota de *Informe*)

La nota de tu informe aumentará en un 5 % si tu informe, a criterio del corrector, está bien presentado y usa el espacio y formato a favor de entregar la información.

Manejo de memoria perfecto (+5 % a la nota de *Código*)

Se aplicará este bonus si **valgrind** reporta en tu código 0 leaks y 0 errores de memoria, considerando que tu programa haga lo que tiene que hacer.

Anexo

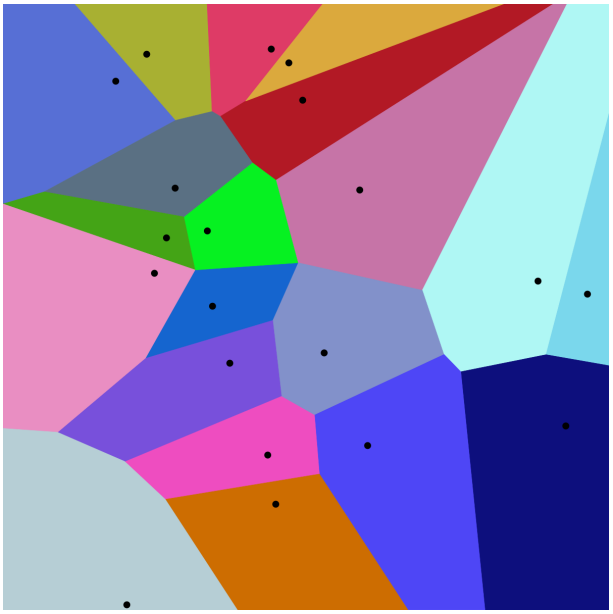


Diagrama de voronoi, con los núcleos en negro.