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

IIC2133 — Estructuras de Datos y Algoritmos
2022 - 2

Tarea 1

Fecha de entrega código: 30 de Septiembre

Objetivos

- Investigar y comprender aplicaciones de árboles en operaciones de búsqueda, inserción y ordenación
- Comprender técnicas de compresión *lossy* sobre información
- Aplicar técnicas algorítmicas sobre árboles

Esta es la Tarea 1 de *Estructura de Datos y Algoritmos* y el objetivo es que aprendan a usar árboles. Consta de 2 partes de código (cuya calificación es 55 % y 40 %, respectivamente) y un breve cuestionario en Canvas (cuya calificación es del 5 %).

Parte 1: QuadTree

Luego de tu gran trabajo organizando las *batiplaylists*, tu héroe **Batman** finalmente pudo organizar correctamente su librería de canciones. Sin embargo, no hubo tiempo para celebrar, ya que uno de los peores criminales hizo lo impensable... limitaron el almacenamiento cloud de BatiDrive. Lugar donde se encuentran imágenes con evidencia de los crímenes cometidos por todos los enemigos de la **DCCiudad Gótica**.

Para prevenir la eliminación de esta importante evidencia, recuerdas una palabra que alguna vez escuchaste: *compresión*. Técnica que permite reducir el tamaño de archivos al sacrificar "un poco" de calidad.

Imágenes

Para un computador, una imagen no es más que una matriz de colores y existen diversos modelos matemáticos para representar colores en este formato, tales como RGB, HSV y CIE-Lab. Para esta tarea, utilizaremos el espacio de color CIE-Lab, el cual permite mejores resultados al hacer operaciones de comparaciones entre colores, ya que se ajusta más a la percepción humana. Esto significa que cada color está representado por una 3-tupla con los siguientes valores:

- L: canal de luminosidad, de 0 a 100
- a: canal cromaticidad verde-magenta, de -128 a 128
- b: canal cromaticidad azul-amarillo, de -128 a 128

Problema

El problema consiste en armar un algoritmo de compresión para **Batman** que permita comprimir imágenes según ciertos criterios dados. Este algoritmo será llamado `DCCompress`. Dicho algoritmo deberá utilizar como base una estructura de datos llamada **QuadTree** que es una estructura de datos que divide los datos en 2 dimensiones en 4 cuadrantes iguales.

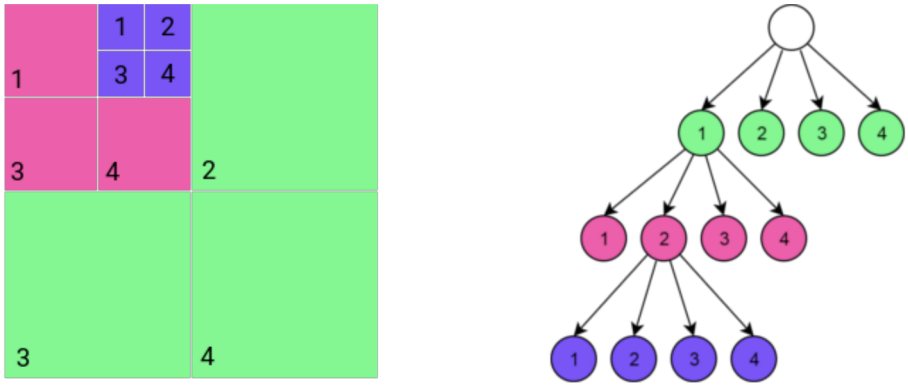

Figura 1: Un QuadTree de profundidad 3.

En el caso de una imagen, cada píxel de esta correspondería a una hoja del árbol, mientras que los demás nodos representan grupos de píxeles dentro de un cuadrante dado. La lógica detrás del algoritmo de compresión es agrupar los píxeles de colores similares dentro de un mismo cuadrante, para luego reemplazarlos por un solo color, que en este caso será el promedio entre los elementos del cuadrante.

Similitud

Para determinar qué tan similares son los colores de un cuadrante, utilizaremos como métrica la desviación estándar. Para un set de n datos $\{x_1, \ldots, x_n\}$, esta se define de la siguiente manera:

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2}$$

donde μ se define como el promedio dado por:

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i$$

Este cálculo debe repetirse para cada uno de los *canales* de la imagen a tratar, es decir, debes calcular σ_L, σ_a y σ_b[1]. Además, definiremos la desviación estándar de un cuadrante como el promedio de las desviaciones:

$$\gamma = \frac{\sigma_L + \sigma_a + \sigma_b}{3}$$

Para calcular el valor de los diferentes σ de forma eficiente, se recomienda aplicar técnicas de `cálculo incremental`[2]

[1] Recordar que L, a y b fueron definidos en la página anterior
[2] En el siguiente enlace puedes encontrar el desarrollo para dicho cálculo

Filtro alfa

Este sub-algoritmo tiene un parámetro, que llamaremos α, el cual indica la desviación estándar máxima que puede tener un cuadrante. Dado α, para cada cuadrante que tenga $\gamma \leq \alpha$, se deja ese cuadrante como hoja del árbol y el color que le corresponde en la imagen es μ, es decir, el promedio entre todos los colores del cuadrante. En caso contrario, el cuadrante se subdivide en 4, y se aplica este criterio recursivamente para cada uno de estos sub-cuadrantes.[3]
Llamaremos a esta operación un "filtro" del árbol según α.

(a) Imagen original

(b) $\alpha = 5$

(c) $\alpha = 10$

Compresión

El algoritmo de compresión recibe como parámetro la cantidad máxima de hojas h que pueden existir en el árbol comprimido. Lo que hace este algoritmo es buscar, usando `búsqueda binaria`, el $\alpha \in \mathbb{N}$ más pequeño tal que se cumpla, que al filtrar el árbol según este α, la cantidad de hojas del árbol sea menor a h. Los límites posibles para α van de 0 a 128

(a) Imagen original

(b) $h = 50000$

(c) $h = 10000$

Notas

- Se espera que desarrolles los algoritmos de filtro y compresión usando un QuadTree.
- Se espera que al iniciar tu programa construyas el árbol a partir de la imagen, y que luego lo utilices mediante consultas para llevar a cabo los distintos pasos del algoritmo.
- El siguiente enlace explica un poco más el funcionamiento de este árbol

[3] Puedes ver un ejemplo de la división en el siguiente enlace

Librería y Código Base

Para el manejo de imágenes, tus ayudantes han preparado una librería a tu disposición. Esta se encarga de todo lo que es lectura y escritura de imágenes. Recuerda leerla atentamente y familiarizarte con su interfaz, así no perderás tiempo implementando funciones que te han sido entregadas.

Además, en el código base se dejó un ejemplo de como sobreescribir el color de un cuadrado de la imagen.

Input

Tu programa deberá responder llamadas de la forma

```
./dccompress input_image output_image command param
```

En donde `command` indica el modo de funcionamiento y puede tener dos valores:

1. filter: Tu programa deberá filtrar la imagen `input_image` (en formato png) con `param` como α.
2. compress: Tu programa deberá comprimir la imagen `input_image` (en formato png) con `param` como h.

Puedes asumir que la imagen de input siempre será cuadrada y tanto su ancho como su largo serán siempre una misma potencia de 2.

Output

El output de tu programa es la imagen resultante, la cual deberás guardar en la ruta output imagen que se te ha otorgado.

- Si la imagen que entregas no coincide con la imagen esperada, tendrás automáticamente 0 puntos en ese test.
- Si tu algoritmo demora más de 10 segundos en un test (sin considerar el tiempo que toma leer / escribir el archivo de imagen), será cortado y tendrás 0 puntos en ese test.
- En el repositorio base de su tarea está explicado como medir el tiempo de su función sin considerar el leer / escribir el archivo de imagen. Tu programa se probará con diversas imágenes de tamaños crecientes.

Parte 2: BST

Batman ha infiltrado la mafia coreana para detener al **Music Meister** quien se ha robado su disco y pretende obligar al mundo a odiar el K-pop. Por lo tanto, nuestro héroe quiere recuperar el disco de su grupo favorito "BTS". Sin embargo, para molestar a **Batman**, nuestro villano ha decidido usar la dislexia como herramienta. Ahora, el amado disco de **Batman** se llama **BST**. Al investigar descubres que un BST es un **Binary Search Tree** (o Árbol de Búsqueda Binaria). Lamentablemente, **Batman** odia al medio ambiente y no siente un amor fuerte por árboles porque le recuerdan a su árbol familiar. Por esto te encarga a ti solucionarlo y crear el árbol "BST" para vencer al villano. Con este plan despertará el amor de **Music Meister** por el K-pop.

Problema

Para este problema se te entregará un input con N valores desordenados y lo que tendrás que hacer es colocarlos en una estructura de árbol (BST) para realizarle distintas operaciones.
(Hint: Puedes asumir que todos los números son enteros. En caso de empate de valores, hacerlo por orden de input)
Lo que deberás hacer es representar el árbol, con respecto a los valores de los nodos. Luego de armado el árbol, se evaluará la construcción de este con las siguientes operaciones.
Además, el equipo de ayudantes preparo una cápsula para implementar BST en C que puede ser de mucha utilidad. Puedes encontrarla en el siguiente enlace.

Operaciones

1. `PATH value`

Esta consulta entrega un `value` que será el valor que debes buscar en el árbol. El output de esta consulta debe ser el valor de `value` para cada nodo recorrido hasta encontrar el buscado.
Por ejemplo, considerar el siguiente BST

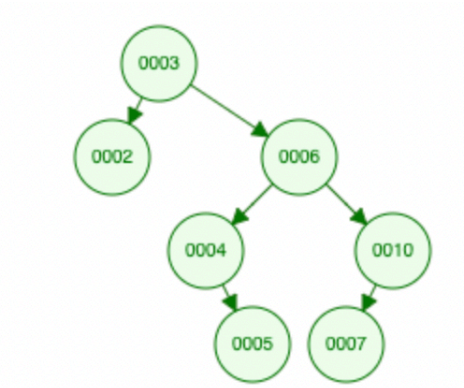

Figura 4: BST de ejemplo

Si la consulta es `PATH 4` entonces tu programa debe imprimir (al archivo) el resultado **3 6 4** que es el camino hasta dicho nodo. En caso de no existir el nodo, se debe imprimir el camino y una X al final. Por ejemplo, si la búsqueda es `PATH 8` se debería imprimir `3 6 10 7 X`

2. `DEEP value`

Esta consulta al igual que la anterior entrega un valor buscado en el árbol. Como output debes entregar la profundidad a la que se encuentra dicho valor. Considera que el root se encuentra en profundidad 0. En el ejemplo anterior, si la consulta es `DEEP 4`, el resultado entregado debería ser 2.

3. `ORDER`

Esta operación pide que retornes los `values` ordenados. Ojo, esta consulta no puede tener una complejidad mayor a $\mathcal{O}(n)$.

4. `SUBTREE S`

Esta consulta pide que busques si un subárbol se encuentra dentro del árbol original o no. Debes retornar un `1` si lo encuentras, o un `0` si no. La consulta entregará un número `S` que indica el número de nodos del subárbol, seguido de una línea con los valores de los S nodos. Por ejemplo:

```
SUBTREE 5
78 34 2 18 46
```

Hint: Piensen en buscar cada nodo recursivamente.

Código Base y Ejecución

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

```
./bstman input.txt output.txt
```

El código base posee solo un archivo `main.c` encargado de manejar la lectura inicial del archivo. El resto de la lógica debe ser creado por ustedes.

Input y Output

Como input se entregará primero una línea con un número N que indica el número de nodos que se entregarán, seguido de una línea con los valores de los N nodos. Pueden asumir que **no se entregarán nodos repetidos**. Finalmente, se entregará una línea Q que indica el número de consultas a realizar, seguido de Q líneas con cada consulta en el formato descrito anteriormente.
Ejemplo:

```
4
51 414 23 104
4
PATH 10
DEEP 104
SUBTREE 2
62 34
ORDER
```

Tu output deberán ser Q líneas con los resultados de cada consulta.

Cuestionario

El cuestionario será una mini evaluación de Canvas donde ser realizaran preguntas de alternativas y deberás responder preguntas sobre el enunciado y las estructuras de datos involucradas. Tendrás intentos ilimitados y se utilizará el mejor puntaje.
Será publicado durante la semana y la fecha de entrega del cuestionario será el día **Martes 27 de septiembre**.

Evaluación

La nota de tu tarea es calculada a partir de tests. Separando entre la Parte 1 y 2 de la tarea

- 55 % Parte 1
  - 75 % Filtro alpha
  - 25 % Operación compresión
- 40 % Parte 2
- 5 % Cuestionario Canvas

Para cada test de evaluación, tu programa deberá entregar el output correcto en menos de 10 segundos y utilizar menos de 1.2 GB de ram[4]. De lo contrario, recibirás 0 puntos en ese test.

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 pueden 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 ./dccompress input_image output_image command param`

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

[4]Puedes revisarlo con el comando htop o con el servidor

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.

Política de atraso: La política de atraso del curso considera dos criterios:

- Utilización de **cupones**[5] de atraso. Donde un cupón te proporciona 24 horas adicionales para entregar tu tarea sin penalización a tu nota
- Entrega atrasada sin cupón donde se aplica un descuento *suave*[6]. Calculada de la forma

$$N_f = min(70 - 0{,}7 \cdot d^{1{,}3}, N_o)$$

  Donde d es la cantidad de días atrasados, N_f la nota final y N_o la nota obtenida en la tarea

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.

[5] Recuerda que solo tienes 2 cupones para todo el semestre
[6] Es un descuento a la nota máxima posible