



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

IIC2133 — Estructuras de Datos y Algoritmos

2023 - 2

## Tarea 1

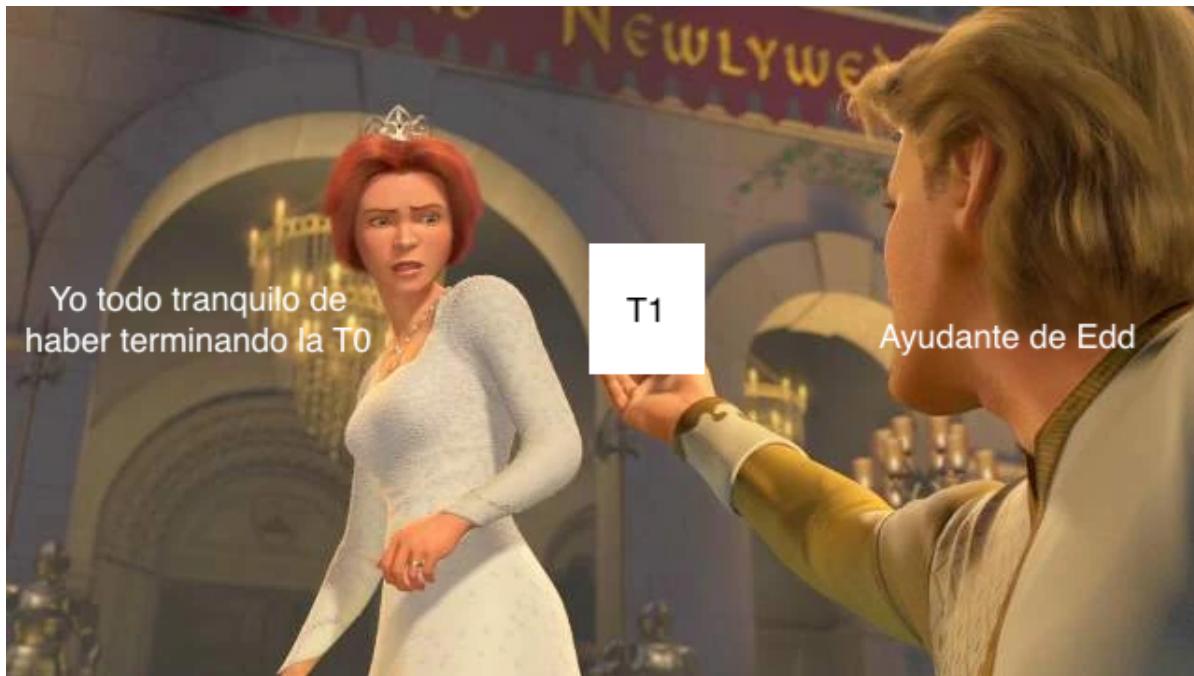
Fecha de entrega código: 26 de septiembre del 2023 a las 23:59

Link a generador de repos: <https://classroom.github.com/a/iUf1ycbw>

---

### Objetivos

- Investigar de manera personal algoritmos y estructuras que permiten resolver problemas.
- Usar técnicas de dividir y conquistar. Además de estrategias recursivas para resolver problemas.
- Familiarizarse con la importancia de la complejidad computacional.



Yo todo tranquilo de  
haber terminando la T0

T1

Ayudante de Edd

Figura 1: Meme hecho con amor

## Parte 1: Una boda muy, muy lejana

Últimamente, el rey de Muy Muy Lejano se ha sentido extremadamente estresado al planear una boda perfecta para su segunda hija favorita. Lamentablemente, hay invitados complicados. Algunos de ellos odian las estructuras de datos, otros no saben programar en C y, lo más insólito, algunos han afirmado que BogoSort es el mejor algoritmo de ordenación, lo cual ha causado un gran conflicto y ha llevado al rey a caer en una depresión mientras intenta organizar a los invitados.



Figura 2: El rey pensando en BogoSort

Por suerte, el rey ha oido hablar de la destreza algorítmica de los estudiantes de Estructura de Datos y Algoritmos, y decide invocar su poder. Basándose en tus profundos conocimientos sobre programación, le recomiendas ordenar los números de los invitados en un Árbol de Búsqueda Binaria (BST), para que luego puedan finalmente sentarlos en una mesa, dar inicio a la boda y disfrutar del pastel.

### 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. No se entregarán inputs con empate).

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 preparó 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

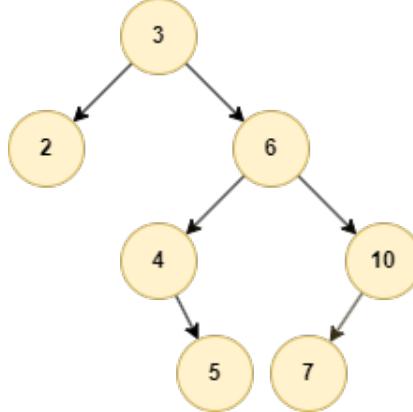


Figura 3: BST de ejemplo

Si la consulta es PATH 4 entonces tu programa debe escribir (al archivo) el resultado **3 6 4** que es el camino hasta dicho nodo. En caso de no existir el nodo, se debe escribir el camino y una X al final. Por ejemplo, si la búsqueda es PATH 8 se debería escribir **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 escrito debería ser 2. En caso de no existir el nodo, se debe escribir **-1**.

### 3. ORDER

Esta operación pide que retorne los **values** ordenados. Ojo, esta consulta no puede tener una complejidad mayor a  $\mathcal{O}(n)$ . Entonces, en caso del ejemplo, tu programa debería escribir **2 3 4 5 6 7 10**

### 4. DEEP-ORDER

Esta operación pide que escribas los **values** ordenados por niveles. Es decir, debes imprimir cada nivel del árbol presentando los nodos de menor a mayor dentro del nivel. Segundo el ejemplo anterior se debería escribir **3 2 6 4 10 5 7**, dado que en el primer nivel está solo el nodo 3, en el segundo nivel se encuentra el nodo 2 y 6, y así iterativamente. Notar que esta función no se puede ver influenciada por el comando INVERT explicado más abajo. Siempre se debe imprimir en orden ascendente el nivel de profundidad.

## 5. INVERT

Esta consulta pide que inviertas completamente el árbol. Deberás cambiar el árbol original por el nuevo árbol invertido, es decir, los hijos derechos pasan a ser hijos izquierdos, y los izquierdos pasan a ser hijos derechos.

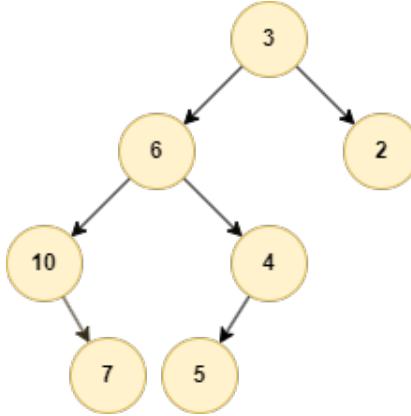


Figura 4: BST INVERT de ejemplo

Para reflejar este cambio, luego de invertir el árbol debes imprimir los **values** por niveles de izquierda a derecha. Según el ejemplo anterior, antes de invertir se debería escribir 3 2 6 4 10 5 7 y luego de invertir sería 3 6 2 10 4 7 5

Hint: Notar que esta impresión es muy parecida a DEEP-ORDER, solo que esta función si cambia la forma de impresión al estar invertido el árbol, es decir, no debe ordenar en orden ascendente los nodos en nivel.

## Código Base y Ejecución

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

```
./bstRey 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
12
PATH 10
DEEP 104
PATH 104
```

```
DEEP-ORDER
ORDER
INVERT
PATH 10
DEEP 104
PATH 104
DEEP-ORDER
ORDER
INVERT
```

Tu output deberán ser  $Q$  líneas con los resultados de cada consulta; debería verse así:

```
51 23 X
2
51 414 104
51 23 414 104
23 51 104 414
51 414 23 104
51 23 X
2
51 414 104
51 23 414 104
23 51 104 414
51 23 414 104
```

## Parte 2: Las EDD de la Hada Madrina

Shrek se dio cuenta del malvado plan que tiene el Rey de Muy Lejano: Ha contratado al Gato con Botas para bajarte puntos en la I1 y obtenido una pócima que podría pasar todo el código de Fiona de C a Python. Enojado, te pidió a ti y al Gato con Botas ayudarle a realizar un asalto a la fábrica de estructuras de datos de la Hada Madrina, para obtener la pócima de EDD que permita a Fiona y el Heap estar libres para siempre.



Figura 5: Shrek buscando cuál algoritmo sirve

Lamentablemente, no todo ocurrió acuerdo al plan, pero lograron salir sanos y salvos de la ahora destruida fábrica. Tú y Shrek deciden tomarse la pócima, pero por el momento, no ven ningún efecto. El día siguiente, se dan cuenta de que no se leyeron completamente la etiqueta: la pócima que tomaron era de QuadTree, así que tendrán que sufrir unas transformaciones de imágenes por el momento que dure la tarea.

## 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 **imágenes PNG** en 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 QuadTree que permita a **Shrek** controlar diferentes cambios de imágenes. Se usará un algoritmo que permite comprimir imágenes según ciertos criterios dados, llamado **HadaCompress**. 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.

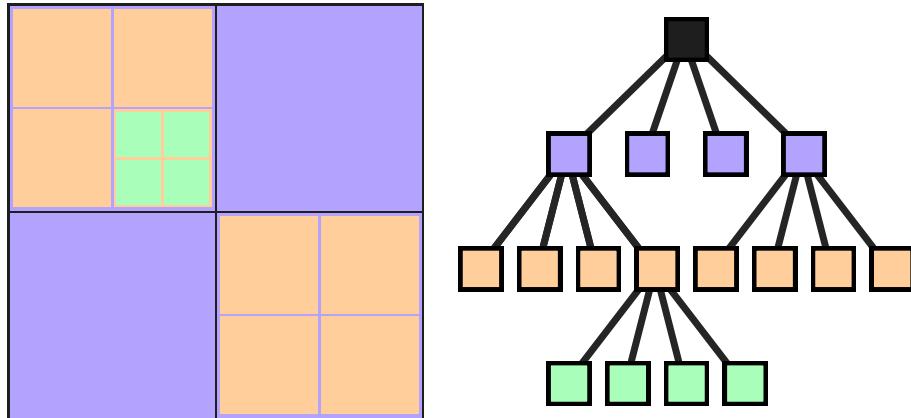


Figura 6: Un QuadTree de profundidad 3.

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  $\{x_1, \dots, x_n\}$ , esta se define de la siguiente manera:

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

donde  $\mu$  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  $\sigma_L$ ,  $\sigma_a$  y  $\sigma_b$ <sup>1</sup>. 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  $\sigma$  de forma eficiente, se recomienda aplicar técnicas de **cálculo incremental**<sup>2</sup>

---

<sup>1</sup>Recordar que  $L$ ,  $a$  y  $b$  fueron definidos en la página anterior

<sup>2</sup>En el siguiente [enlace](#) puedes encontrar el desarrollo para dicho cálculo

## Filtro alfa

Este sub-algoritmo tiene un parámetro, que llamaremos  $\alpha$ , el cual indica la desviación estándar máxima que puede tener un cuadrante. Dado  $\alpha$ , 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  $\mu$ , 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.<sup>3</sup> Llamaremos a esta operación un “filtro” del árbol según  $\alpha$ .



(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  $\alpha$ , la cantidad de hojas del árbol sea menor a  $h$ . Los límites posibles para  $\alpha$  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

---

<sup>3</sup>Puedes ver un ejemplo de la división en el siguiente [enlace](#)

## Código Base y Ejecución

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.

**IMPORTANTE:** Para el manejo de imágenes se requiere libpng, lo que puede ser difícil de instalar, especialmente en MacOS. Te recomendamos usar [EDD Runner](#) o [EDD Dev Container](#) para **compilar**.

### Input

Tu programa deberá responder llamadas de la forma

```
./hadaCompress 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 con **param** como  $\alpha$ .
2. **compress**: Tu programa deberá comprimir la imagen con **param** como  $h$ .

Con **input\_image** una imagen PNG existente y **output\_image** la imagen PNG a crear. 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 o 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 o escribir el archivo de imagen. Tu programa se probará con diversas imágenes de tamaños crecientes.

## Documento de diseño

La tarea contempla un documento de diseño que posee ponderación en la nota. En particular, el documento es evaluado de forma binaria. Asignando puntaje completo en caso de realizar las tres secciones, y ningún puntaje en caso contrario.

La idea del documento son varias. Primero es incentivar el análisis del enunciado y del problema en el inicio. Es recomendable realizar al menos el 50 % del documento antes de empezar a programar.

Por otro lado, la idea es permitir realizar correcciones manuales de código. Donde se asigna puntaje parcial a la implementación realizada.

**IMPORTANTE: Si no se realiza el documento de diseño, NO se podrá optar a corrección manual.**

### Contenidos mínimos

El documento de diseño ha de ser llenado en el template `docs/design.md` ubicado en el repositorio base de la tarea.

No se espera que el documento sea formal, sino que se espera que queden como registros del proceso de diseño de su solución. Es por esto que ha de contener como mínimo las siguientes secciones:

1. Análisis del enunciado: Breve análisis identificando las estructuras y sus relaciones.
2. Planificación de solución: una idea a rasgos generales sobre como será abordado el problema, identificando que Estructura de datos se utilizará.
3. Justificación: Sección donde se justifican las decisiones de diseño consideradas para la implementación de la tarea

### Entrega documento de diseño

El documento de diseño se entrega en conjunto con el repositorio de la tarea, el contenido principal ha de ir en el archivo `docs/design.md`. En caso de querer dejar anexos como imágenes o diagramas, han de ser añadidos en la misma carpeta.

## Evaluación y Cálculo de Nota

La tarea contempla dos secciones principales que contribuyen a la nota. La sección práctica, compuesta por resolver correctamente los tests y la parte teórica.

### Parte Práctica (80 %)

Para cada test de evaluación, tu programa deberá entregar el output correcto en menos de 10 segundos y utilizar menos de 1 GB de ram<sup>4</sup>. De lo contrario, recibirás 0 puntos en ese test.

- Parte 1: 30 %
- Parte 2: 60 % (50 % compress y 50 % filter)
- No Leaks de Memoria: 5 %
- No Errores de Memoria: 5 %

La nota individual de cada parte se evalúa con la cantidad de tests correctos más 1. En el caso de leaks y errores. Puedes chequear que tu programa no los posea utilizando la herramienta `valgrind`.

### Parte Teórica (20 %)

La parte teórica contempla el documento de diseño y el cuestionario.

**Documento de diseño (20 %):** Su evaluación es binaria. Ósea, obtiene puntaje máximo en caso de realizarlo completo y puntaje mínimo en caso contrario. Solo se evalúa la presencia de las secciones, no es importante que sea detallado.

**Cuestionario (80 %):** El cuestionario posee un total de 25 puntos obtenibles, de los cuales se consideran 20 para la nota máxima. Su fórmula es la siguiente:

$$nota = 6 \cdot \frac{\min(puntaje, 20)}{20} + 1$$

Para poder acceder al cuestionario debes ingresar a esta página: [Enlace](#)

## 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, veas este link para obtener el bonus 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 :)

---

<sup>4</sup>Puedes revisarlo con el comando `htopm`, en el servidor, o con `valgrind --tool=massif`.

## 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:** Para esta tarea **aplica** la política de atraso del curso, definida de la siguiente forma:

- Cada día de entrega atrasada, descontará 7 décimas de la **nota máxima obtenible**. Con un máximo de 3 días
- Todos los estudiantes parten con 2 cupones de atraso. El uso de un cupón *elimina* un día de atraso

En resumen, el código para calcular la nota es el siguiente:

```
double nota_con_atraso(double nota, int dias_de_atraso, int cupones_usados) {  
    int atraso_efectivo = dias_de_atraso - cupones_usados;  
  
    if (atraso_efectivo > 3) { return 1.0; }  
  
    return min(nota, 7.0 - 0.7 * atraso_efectivo);  
}
```

## 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.