Documento de Diseño – ProyeccionesEDD
Estructuras de Datos

Presentado por:

Juan Esteban Urquijo

Ángel David Talero

Iván Alejandro Martínez

Presentado a:

Andrea Rueda Olarte

Pontificia Universidad Javeriana

Estructura de datos

Bogotá D.C, junio 3 de 2022

Correcciones primera entrega:

Problema	**Corrección**
Para el criterio mediana, no se reconoce el comando, es decir, no se pueden generar proyecciones usando la mediana.	El criterio ‘mediana’ fue añadido, se calcula un vector con todos los valores del píxel en las coordenadas I, J de cada una de las capas, luego se ordena este vector y se halla el dato de en medio, o el promedio de los datos de en medio en el caso de un vector de elementos pares
Las proyecciones en Y y Z no están quedando bien, parecen como cortadas por la mitad	El algoritmo fue modificado de tal manera que las proyecciones se realicen capa por capa (como en la dirección de las x) sin importar la dirección, para esto, las proyecciones en ‘y’ y ‘z’ realizan una rotación de 90° sobre volumen previa a la rotación, esto facilita los cálculos, permite un código más limpio y soluciona el problema en la proyección
En la implementación, la definición de las librerías no es correcta. La cabecera de cada TAD debe estar en un archivo .h	Se cambió la extensión de los archivos por *.h* y se actualizaron los ‘*header guards’* e *#includes*
El promedio en X está muy bajito, como si se estuviera dividiendo sobre más de lo esperado.	Como se mencionó anteriormente, el algoritmo de proyección fue reescrito desde 0, el promedio se calcula dividiendo por el tamaño de la cola de capas a promediar
En las direcciones ‘X’ y ‘Y’, al utilizar el criterio ‘mínimo’, los valores no son los esperados.	Todas aquellas imágenes que tengan un borde negro tendrán proyecciones vacías en ‘Y’ y ‘Z’, porque el borde tiene el mínimo valor posible (0), es por este motivo que el valor 0 ya no es tomado en cuenta para hallar el valor mínimo y al final del cálculo, si ningún valor era diferente de 0, el píxel a colocar en esa posición es 0 para así evitar una imagen negativa (fondo blanco)
El TAD Proyección es innecesario	El TAD fue removido y el procedimiento encargado de realizar la proyección ahora es una función miembro del TAD Volumen, porque es el volumen quien se tiene que proyectar y tiene más sentido que esté ahí
La ayuda general sólo requiere el nombre (y si acaso los parámetros) de cada comando.	Se modificó el comando para que muestre el nombre y una breve descripción porque lo consideramos necesario, para ver el uso de un comando en específico es necesario usar el comando: ayuda <comando>

Correcciones segunda entrega:

Problema	**Corrección**
Para el criterio de mínimos la imagen muestra toda la imagen en blanco	El valor mínimo encontrado en Z para todas las capas es de ‘1’ el sistema ajusta automáticamente el valor máximo encontrado en la matriz, en esta entrega se obliga al programa a que inserte un valor máximo de 255
Faltan las relaciones entre TADs	Se creó un diagrama de TADs con todas las asociaciones correspondientes
Los TADs de la primera, segunda y tercera entrega no deberían estar desconectados, deben relacionarse de alguna manera	Los TADs de la entrega 2 y 3 se relacionan con el TAD Imagen de la entrega 1, por lo que hemos dibujado esa relación en el diagrama, además de agregar unas lineas punteadas que representan la dependencia del Controlador con los TADs de la segunda y tercera entrega

(Descripción del proyecto en la siguiente página)

Descripción de entradas

Dentro de las entradas al sistema se encuentra por una parte una imagen o una serie de imágenes (un volumen) en formato PGM, un archivo de texto que representa el valor en escala de grises de cada uno de los píxeles que conforman la conforman, donde los valores pequeños iguales o mayores a 0 representan colores más cercanos al negro y los valores más grandes representan colores más cercanos al blanco.

Descripción de Salidas

Las salidas del sistema son imágenes en formato PGM producto de la proyección 2D sobre un volumen de imágenes en la dirección indicada ya sea X, Y o Z y que contienen una serie de radiografías de un cerebro humano, un archivo binario (.huffman) con la codificación de Huffman de una imagen o una imagen resultado de la segmentación de otra a partir de una serie de semillas.

Diseño de los TADs

TAD Imagen
Conjunto Mínimo de Datos
formato_imagen, cadena de caracteres, representa el formato de la imagen
ancho, entero, representa el ancho de la imagen en píxeles
alto, entero, representa el alto de la imagen en píxeles
max_tam, entero, representa el valor del píxel más grande de la imagen
matriz_pixeles, matriz (vector de vectores) de bytes, representa los valores de gris de los pixeles en las coordenadas bidimensionales
nombre_archivo, cadena de caracteres, nombre del archivo
Comportamiento (Operaciones)
Imagen(), crea una nueva imagen vacía
Imagen(matriz_pixeles), crea una nueva imagen a partir de una matriz de escala de grises
Imagen(nombre_archivo), crea una nueva imagen a partir de un archivo
guardar_archivo(nombre_archivo), guarda la imagen actual en formato PGM
get_formato(), retorna el formato de la imagen cargada
get_ancho(), retorna el ancho de la imagen cargada
get_alto(), retorna el alto de la imagen cargada
get_max_tam(), retorna el valor del pixel mas grande de la imagen cargada
get_pixeles(), retorna la matriz de pixeles de la imagen cargada
get_nombre_archivo(), retorna el nombre del archivo de la imagen
set_formato(formato), establece un nuevo formato para la imagen
set_ancho(ancho), establece un nuevo ancho para la imagen
set_alto(alto), establece un nuevo alto para la imagen
set_max_tam(maxtam), establece un nuevo tamaño máximo de pixel para la imagen
set_pixeles(matriz_pixeles), establece un nueva matriz de pixeles que forma la imagen
set_nombre_archivo(nombre), establece un nuevo nombre para el archivo de la imagen

to_string(), mostrar información básica de la imagen en texto
matriz_vacia(ancho, alto), crea una matriz inicializada en 0
llenar_matrix(mtx, value), llena los contenidos de una matriz con el valor
reflejo_vertical(img), devuelve una copia de la imagen pero reflejada verticalmente

TAD Volumen
Conjunto Mínimo de Datos
volumen, cola de Imagen, representa las capas (Imágenes) que conforman el volumen
nombre_base, cadena de caracteres, representa el nombre base de las imágenes en la serie
tam_volumen, entero, representa a el número de imágenes en la serie de imágenes (capas)
ancho, entero, representa el ancho en pixeles de una capa
alto, entero, representa el alto en pixeles de una capa
Comportamiento (Operaciones)
Volumen(), constructor de un volumen vacío
Volumen(nombre_base, tam), carga todas las imágenes con el nombre base en un volumen
get_nombre_base(), retorna el nombre base de la imagen presente en el volumen
get_tam_volumen(), retorna el tamaño del volumen cargado
get_volumen(), retorna el volumen de imágenes cargado
get_ancho(), retorna el ancho de una imagen dentro del volumen
get_alto(), retorna el alto de una imagen dentro del volumen
set_nombre_base(nombre_base), establece un nuevo nombre base para las imágenes dentro del volumen
set_tam_volumen(tamVolumen), establece un nuevo tamaño para el volumen
set_volumen(volumen), establece un nuevo volumen
to_string(), imprime la información básica del volumen
crear_proyeccion(criterio, dirección, nombre_archivo), crea la proyeccion 2D del volumen con el criterio y dirección especificada y lo guarda en el archivo especificado

TAD Controlador
Conjunto Mínimo de Datos
imagen_cargada, apuntador a Imagen, guarda en memoria la Imagen cargada por el usuario
volumen_cargado, apuntador a Volumen, guarda en memoria el Volumen cargado por el usuario
Comportamiento (Operaciones)
cargar_imagen(argumentos), Carga en memoria la imagen especificada
cargar_volumen(argumentos), Cargar en memoria una serie ordenada de imágenes con un nombre base y un tamaño 'n_im' (Volumen)
info_imagen(argumentos), Muestra información básica de la imagen actualmente carga en memoria.
info_volumen(argumentos), Muestra información básica del volumen actualmente cargado en memoria.
proyeccion2D(argumentos), Generar una proyección 2D a partir de un volumen cargado en memoria y guardarlo en un archivo.
codificar_imagen(argumentos), Codificar la imagen cargada en memoria con la codificación de Huffman

decodificar_imagen(argumentos, Decodificar un archivo de huffman y guardar la imagen decodificada
segmentar(argumentos), segmentar la imagen cargada en memoria a partir una serie de semillas

TAD NodoCodificacion
Conjunto Mínimo de Datos
Frecuencia,entero, almacena la frecuencia con la que se repite el pixel en la imagen
Comportamiento
NodoCodificacion(frecuencia), constructor

TAD NodoElemento
Conjunto Mínimo de Datos
dato, dato tipo plantilla, almacena el dato del código que contiene cada nodo
Comportamiento
NodoElemento(elemento, frecuencia)

TAD NodoFrecuencia
Conjunto Mínimo de Datos
hijoder, apuntador a NodoCodificacion, almacena la referencia del nodo como hijo derecho
hijoIzq, apuntador a NodoCodificacion, almacena la referencia del nodo como hijo izquierdo
Comportamiento
NodoFrecuencia(frecuencia), constructor con parámetros que recibe la frecuencia de un pixel
codigos_elementos(codigo), retornar un vector con el resultado del código teniendo en cuenta si es hijo derecho o izquierdo.

TAD ArbolCodificacion
Conjunto Mínimo de Datos
raiz, apuntador a NodoCodificacion, referencia a la raíz del árbol de codificación
Comportamiento
ArbolCodificacion(),constructor de un nodo del árbol
ArbolCodificacion(frecuencias),constructor con parámetros que recibe las frecuencias de los pixeles

TAD CodigoElemento
Conjunto Mínimo de Datos
Elemento, dato tipo plantilla, almacena el pixel de la imagen a codificar
Frecuencia, entero, almacena la frecuencia de repetición de un pixel de la imagen
Codigo,cadena de caracteres, representa el codigo que se le asigna al pixel para codificarlo
Comportamiento
CodigoElemento(elemento,frecuencia,codigo), constructor que recibe todos los parámetros
to_string(), obtener por pantalla una descripción del elemento

TAD Huffman

Conjunto Mínimo de Datos

ancho, entero sin signo de 2 byte, guarda el ancho de la imagen
alto, entero sin signo de 2 byte, guarda el ato de la imagen
maximo, entero, guarda la intensidad máxima de la imagen
arbol, ArbolCodificacion, apunta al árbol de codificación
imagen, Imagen, guarda la imagen desde la que se crea la codificación de huffman
codigos,mapa, mapa de CodigoElemento, guarda los códigos de cada uno de los nodos del árbol

Comportamiento (Operaciones)

Huffman(imagen), construye la codificación de huffman de una imagen
Huffman(nombre_archivo, salida), decodifica un archivo huffman
guardar_archivo(archivo), guarda la imagen formada con la codificación de huffman en un archivo binario

TAD Arista

Conjunto Mínimo de Datos

desde, dato plantilla T, nodo desde el cual proviene la conexión
hasta, dato plantilla T, nodo hasta el cual llega la conexión
peso, entero, peso o costo de la conexión

Comportamiento

Arista(desde, hasta), crea una arista con un costo por defecto de 1
Arista(desde, hasta, peso), crea una arista con el costo determinado
inverso(), obtiene una arista con la dirección inversa a la actual
get_desde(), obtiene el nodo desde donde sale la conexión
get_hasta(), obtiene el nodo hasta donde llega la conexión
get_peso(), obtiene el peso de la conexión

TAD Grafo

Conjunto Mínimo de Datos

vertices, conjunto de datos plantilla T, conjunto de nodos que conforman el grafo de tipo T
lista_adyacencia, multilista de aristas, lista de adyacencia del grafo

Comportamiento

Grafo(aristas), Crea un grafo a partir de un conjunto de aristas
dijkstra_algorithm(vertice_inicial), realiza el recorrido del algoritmo de menor costo de Dijkstra
quitar_vertice(remover), eliminar el vértice especificado del grafo
es_vacio(), verificar si el grafo está vacío
weight_of(list, value), obtiene el costo de un nodo especificado en un camino especificado

TAD Semilla

Conjunto Mínimo de Datos

x, entero, posición en X de la semilla en el plano cartesiano
y, entero, posición en Y de la semilla en el plano cartesiano
intensidad, entero, intensidad del pixel en esa posición (0-255)

etiqueta, entero, intensidad de la etiqueta asignada a ese pixel (0-255)

Comportamiento

Semilla(x, y, intensidad), crea una semilla con los parámetros definidos y una etiqueta en 0
Semilla(x, y, intensidad, etiqueta), crea una semilla con todos los parámetros

TAD GeneradorSemillas

Conjunto Mínimo de Datos

grafo, Grafo no dirigido, grafo en el que se almacenará la imagen
n_aristas, cantidad de aristas que se van a generar dentro del grafo

Comportamiento

GeneradorSemillas(imagen), crea el grafo a partir de una imagen
generar_caminos(semillas), Generar los caminos de Dijkstra a partir de un conjunto de semillas
generar_etiquetas(imagen, caminos), Generar un conjunto de píxeles cuyas etiquetas corresponden a la semilla que lo alcanzan en el menor costo
generar_matrix(semillas, ancho, alto), Generar una matriz de píxeles a partir de un conjunto de semillas etiquetadas

(Diagrama de relación de TADs en la siguiente página)

Diagrama de secuencias (Línea de comandos)

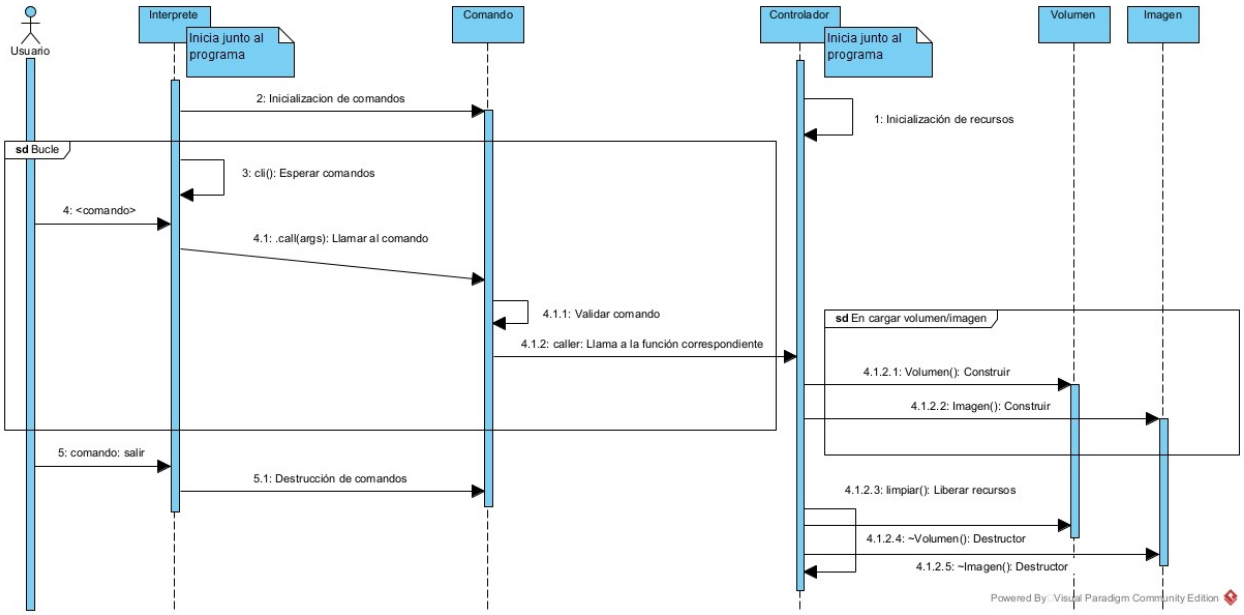
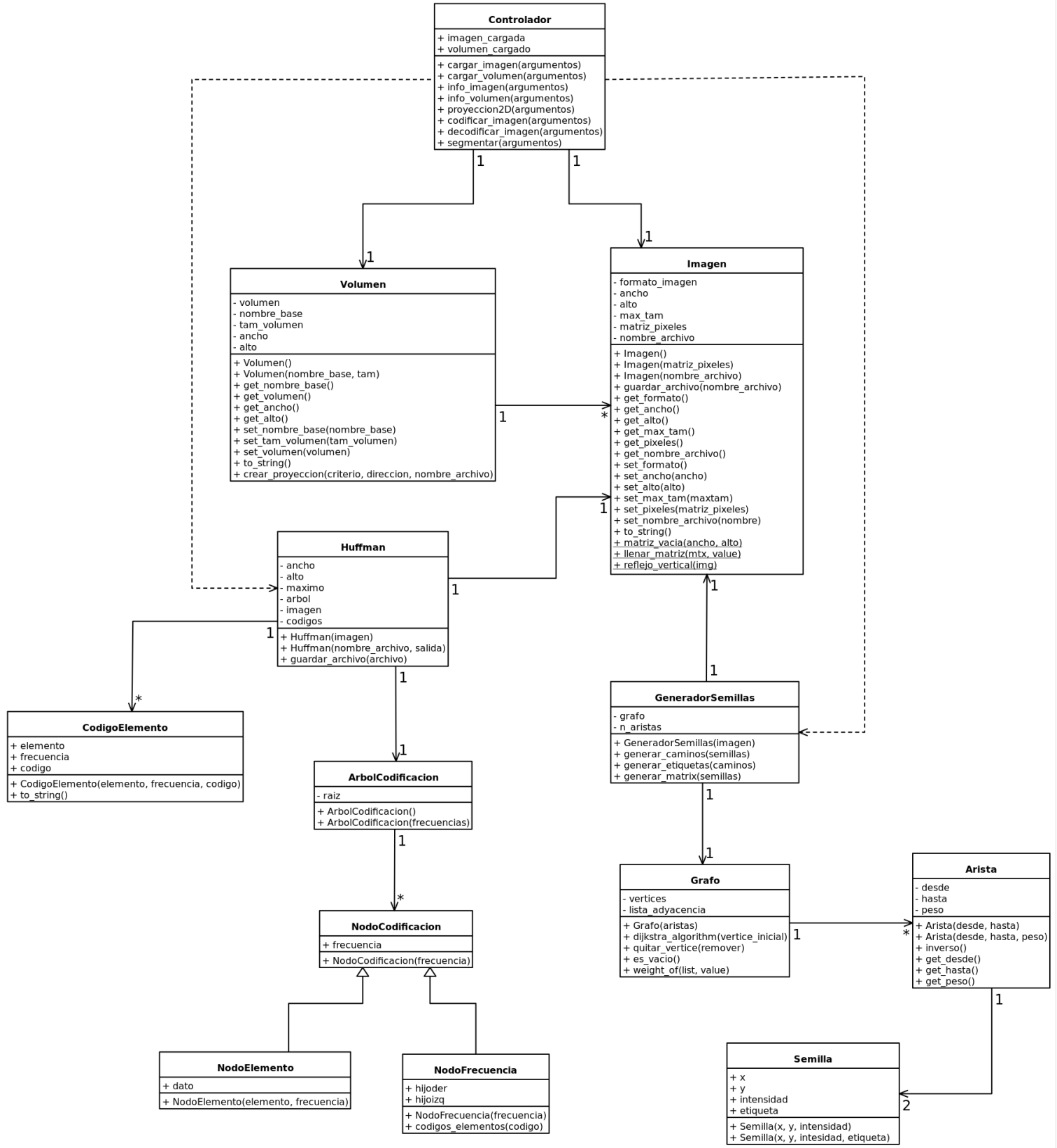
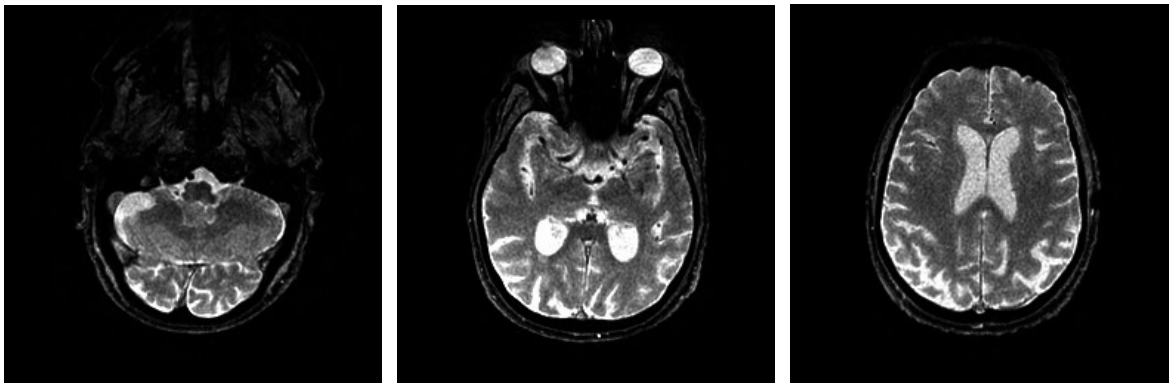


Diagrama TADs

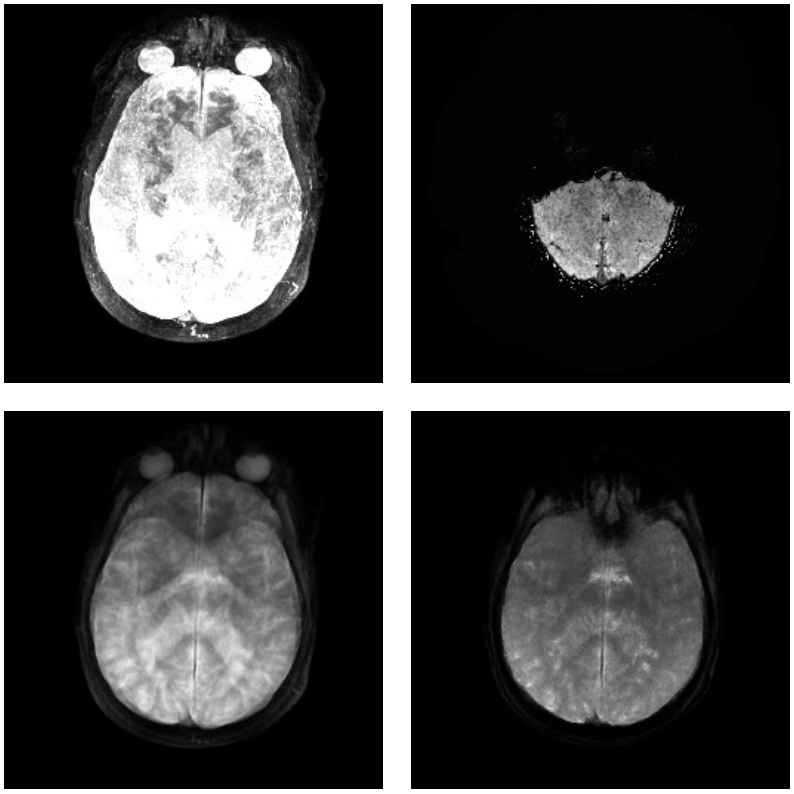


Plan de pruebas Proyecciones 2D: *(Primera entrega)*
Para corroborar el funcionamiento se carga un volumen de las imágenes diagnosticas de una cabeza utilizando las siguientes comandos:
1. > cargar_imagen IM-126-0002-epiT2 36
2.> proyeccion2d <dirección> <criterio> <nombre_salida>

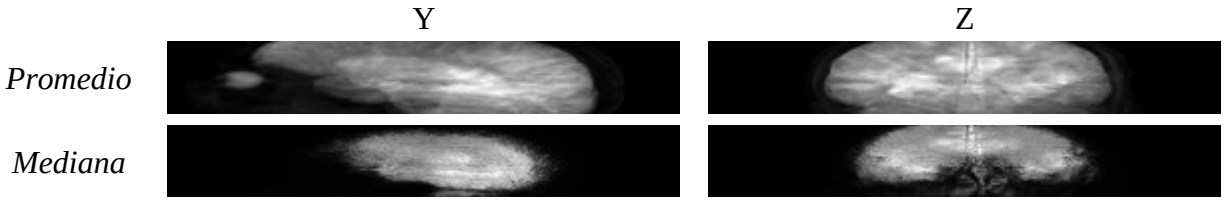
Algunas de las imágenes de *IM-126-0002-epiT2:*



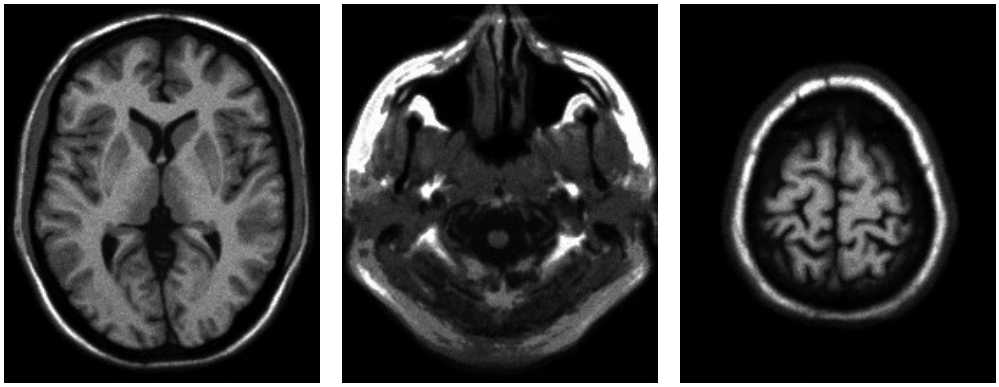
Proyecciones en ‘x’ (Máximo, Mínimo, Promedio, Mediana)



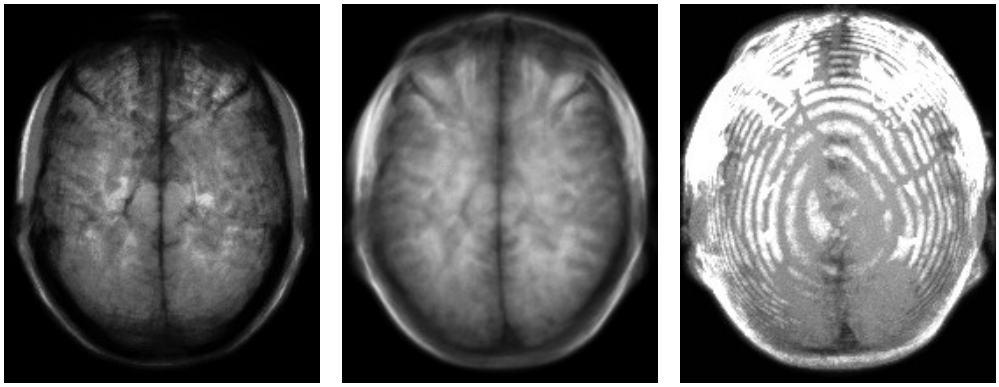
Algunas proyecciones ‘Y’ y ‘Z’:



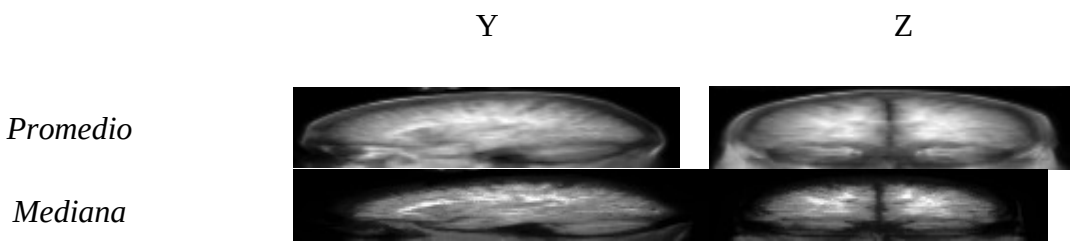
Algunas de las imágenes de *t1_icbm_5mm_:*



Algunas proyecciones en X (mediana, promedio, máximo):

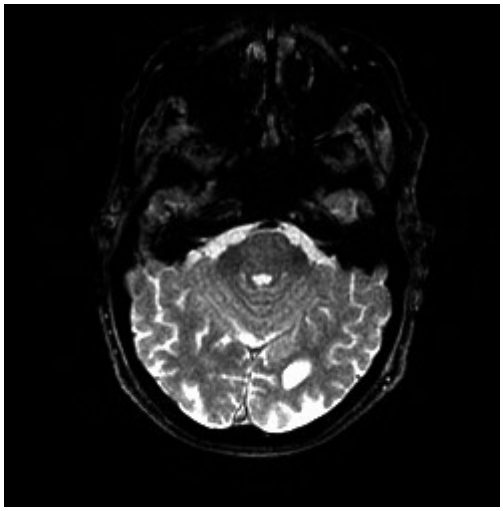


Algunas proyecciones ‘Y’ y ‘Z’:



Plan de pruebas Codificación y Decodificación: *(Segunda entrega)*

Tenemos la siguiente imagen:



Esta imagen tiene una altura y anchura de 256px y un valor de píxel máximo de 255.

1. Se carga la imagen con el comando *cargar_imagen*

```
$ cargar_imagen imagen.pgm
(proceso satisfactorio) La imagen 'imagen.pgm' ha sido cargada
$
```

2. Se utiliza el comando *codificar_imagen* para generar la codificación de huffman.

```
$ codificar_imagen cod.huffman
(proceso satisfactorio) La imagen en memoria ha sido codificada exitosamente
$
```

Ahora se tiene el siguiente archivo binario:

```
build > cod.huffman
          00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000  00 01 00 01 FF AE 7D 00 00 00 00 00 00 0A 2A 00
00000010  00 00 00 00 00 E2 09 00 00 00 00 00 00 9A 04 00
```

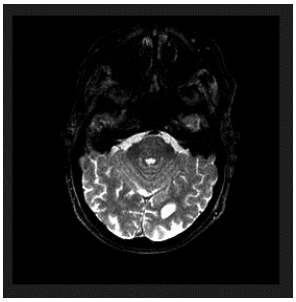
```
FF 8E 7F F9 F8 FF 87 E7 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 01 FF E3 F1 DF F3 FF FF
7E 7F DF FE E7 DE FE 74 67 9E 2A 37 F3 FF 16 7F
7F 69 93 37 47 DF F8 BF 7C F3 FF 7D FB F6 F7 DF
FF CF CE 3F E1 BE EF C7 38 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 7F FF D8 E1 DD C7 FC FF
8F C0 7F FF FE 8C 53 2B CC A3 79 E7 F7 BE FD 6A
```

Como se puede ver en el editor hexadecimal, los primeros dos bytes corresponden a la anchura (0x0100 HEX ó 256 en decimal, los bytes están invertidos por la arquitectura BigEndian del computador) y los siguientes 2 bytes a la altura, el siguiente byte al valor máximo (0xFF hexadecimal, 255 decimal), luego están las 255 frecuencias correspondientes a cada intensidad posible codificadas en valores de 8 bytes, y al final del archivo se puede ver la codificación binaria.

Para decodificar el archivo utilizamos el comando *decodificar_archivo*

```
$ decodificar_archivo cod.huffman salida.pgm
(proceso satisfactorio) El archivo cod.huffman ha sido decodificado exitosamente
$
```

Que nos devuelve como resultado la imagen original:



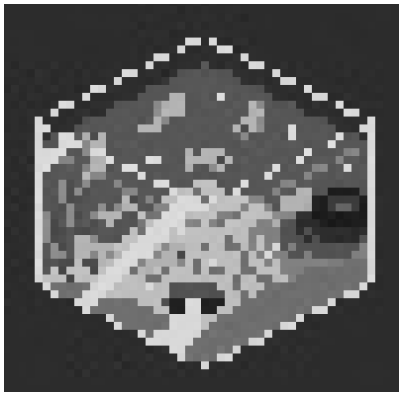
El resultado de la codificación y decodificación reduce significativamente el peso del archivo.

```
$ du -sh imagen.pgm
152K    imagen.pgm
(0) angel at AngelgoLapt
$ du -sh cod.huffman
36K     cod.huffman
(0) angel at AngelgoLapt
$ du -sh salida.pgm
152K    salida.pgm
```

Plan de pruebas Segmentación: *(Tercera Entrega)*

El algoritmo se segmentación requiere crear un grafo donde cada uno de los píxeles de la imagen corresponde a un nodo, y luego evaluar el algoritmo de dijkstra para hallar los caminos más cortos desde cada una de las semillas hacia todos y cada uno de los nodos. Este proceso puede tomar bastante tiempo pues la cantidad de nodos en el grafo está dada por $n*m$, y la cantidad de aristas (arriba, abajo, izquierda, derecha), está dada por $2(n-1)(m-1)+(n-1)+(m-1)$
Es por esta razón que no utilizamos las imágenes diagnósticas para probar este componentes, pues tienen un tamaño de 256 x 256, lo cual implica que tendríamos que calcular un grafo con 65536 nodos y 130560 aristas (261120 si tenemos un par de aristas ida y vuelta) y a su vez re-calcular dijkstra para este algoritmo tantas veces como semillas. Este proceso tiene una complejidad $O(n^4)$ y puede tardar más de una hora

En su lugar utilizamos las siguientes imágenes más pequeñas:

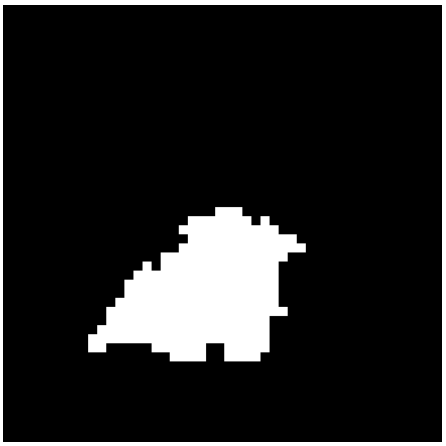


fishtank.pgm (50x50px)

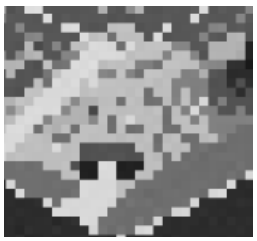
mario.pgm (75x100px)

Pruebas sobre *fishtank.pgm*

> cargar_imagen fishtank.pgm
> segmentar fishtank_out.pgm 0 0 0 0 49 0 49 0 0 49 49 0 25 25 255



Lo que buscamos con esta serie de semillas es colocar una etiqueta de intensidad 0 en cada una de las esquinas y una etiqueta de intensidad 255 en el medio, de esta manera al propagarse las semillas logramos identificar una mancha blanca que corresponde a un componente de la imagen original:



Pruebas sobre *mario.pgm*

> cargar_imagen mario.pgm
> segmentar mario_out.pgm 35 25 255 35 75 125 0 99 50 74 99 50

Con esta segmentación lo que buscamos es crear una semilla para la cabeza (35, 25), otra para el cuerpo (35, 75), y otras dos para las botas (0,99) y (74, 99)

Imagen original

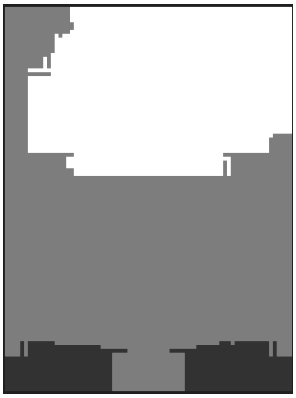


Imagen segmentada

Como podemos evidenciar, las semillas logran propagarse al rededor de las distintas partes que conforman la imagen, se logra distinguir entre las botas, cabeza y torso