

Entrega Final: Proyecto de Curso 2025-10

Miguel Francisco Vargas Contreras, Juliana Sofia Novoa Solano,
Mateo David Guerra Moreno
Bogotá, Pontificia Universidad Javeriana.

1 Actas de evaluación

1.1 Acta de evaluación - Entrega 1

Para esta segunda entrega se hicieron las correcciones pedidas en base a la entrega del **Componente 1** del proyecto. Esta vez solo tuvimos 2 fallas, las cuales fueron mencionadas de la siguiente forma:

Las conexiones en el diagrama de relación de TADs representan contenencia. Conexiones de CommandManager no son conexiones de contenencia sino de uso. Revisar las demás.

Revisar los símbolos de la descripción de TADs. Están incorrectos o mal formateados algunos de ellos.

Se modificó el diagrama de TADS para incluir las nuevas relaciones creadas por los nuevos TADS y se solucionó el error eliminando la relación de contenencia CommandManager - Memoria y CommandManager - Sistema.

Se corrigieron algunos de los símbolos con el formato erróneo presentes en los TADs, incluido el diagrama de relación. La edición se realizó de modo que ahora las descripciones en el diagrama de TADS mantienen el mismo formato.

1.2 Acta de evaluación - Entrega 2

No fue necesario realizar correcciones de la entrega 2.

2 Introducción

En este documento se muestra el diseño propuesto del proyecto de clase para el año 2025-10. Esta entrega final se enfoca en la segmentación de imágenes haciendo uso de Dijkstra.

El formato utilizado para el proyecto describe un volumen como una serie de imágenes ordenadas en formato PGM.

El formato PGM sigue la siguiente estructura:

Listing 1: Formato PGM

```
P2
W H
M
I (1 ,1) I (2 ,1) ... I (W,1)
I (1 ,2) I (2 ,2) ... I (W,2)
...
I (1 ,H) I (2 ,H) ... I (W,H)
```

- P2 - Indica el formato de la imagen (PGM en ASCII)
- W y H son números enteros positivos, representan ancho y alto de la imagen respectivamente.
- M es un número entero positivo, representa el valor de pixel más grande de la imagen.
- I(i,j) es el valor de gris del pixel de la imagen ubicado en la coordenada bidimensional (i,j) $1 \leq i \leq W$ $1 \leq j \leq H$. Cada valor de pixel debe estar entre 0 y 255.

Construimos nuestro proyecto haciendo uso de un sistema de documentación llamado *doxygen*, el uso de este sistema nos permitió trabajar mucho más fácil en equipo, ya que nos dio herramientas para comunicar el cómo funciona cada parte de nuestro sistema. Pensando en eso decidimos adjuntar la documentación usada en el proyecto ('refman.pdf').

Nuestro diseño se basa en un sistema principal, el cual abstrae la complejidad del sistema a dos simples operaciones: tomar la siguiente línea enviada por el usuario y ejecutar un comando con base en esa línea enviada por el usuario. A continuación, desglosaremos parte por parte este sistema.

3 Proyección 2D

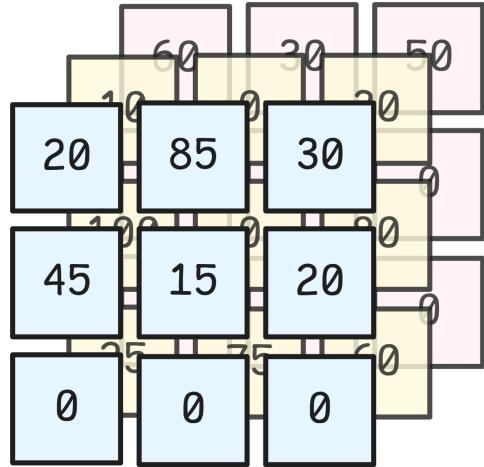


Fig. 1: Volumen de ejemplo

Dado el volumen de ejemplo (Véase Fig. 5.), revisaremos algunas de las operaciones usadas por nuestro sistema para realizar la proyección del volumen sobre un plano 2D. Un volumen esta compuesto por varias capas o "slices", cada una representa una sección transversal del volumen.

Cada imagen se puede ver como una rebanada del volumen en la dirección del eje Z. Todas estas imágenes juntas forman un bloque de datos en 3D. Pero empecemos mostrando como funcionan estos ejes:

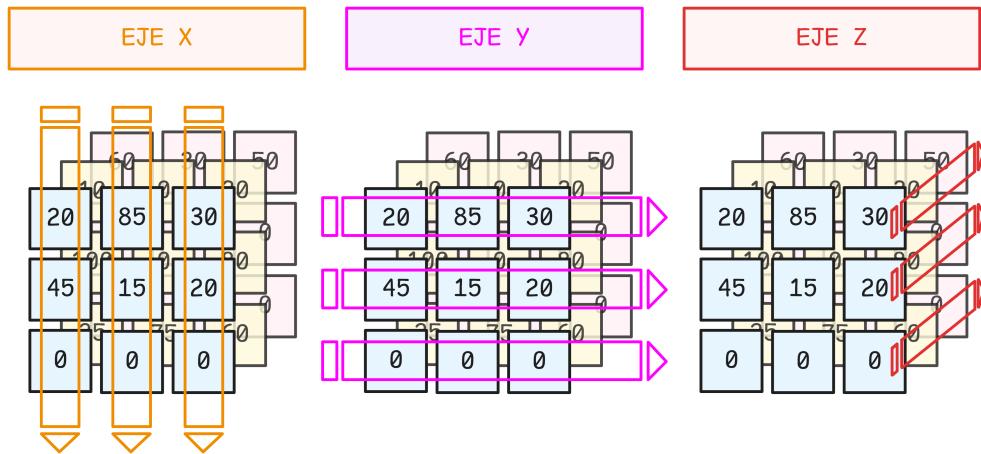


Fig. 2: Ejes de proyección

En este caso tenemos tres imágenes (`diag01.pgm`, `diag02.pgm` y `diag03.pgm`), cada una con una matriz de valores que representan intensidades de píxeles.

<code>diag01.pgm</code>	<code>diag02.pgm</code>	<code>diag03.pgm</code>
20 85 30	10 0 30	60 30 50
45 15 20	100 0 80	40 10 0
0 0 0	25 75 60	25 0 0

Fig. 3: Imagenes del volumen

Realizaremos una proyección 2D en el eje x, esto implica que tomamos los valores de cada columna a lo largo de las diferentes capas del volumen y calculamos un valor representativo para esa columna en la imagen proyectada.

Sobre esto, usaremos (en este ejemplo) el criterio de promedio, esto significa que el valor resultante en la imagen proyectada será el promedio de los valores en la misma posición (x, y) a lo largo del eje Z.

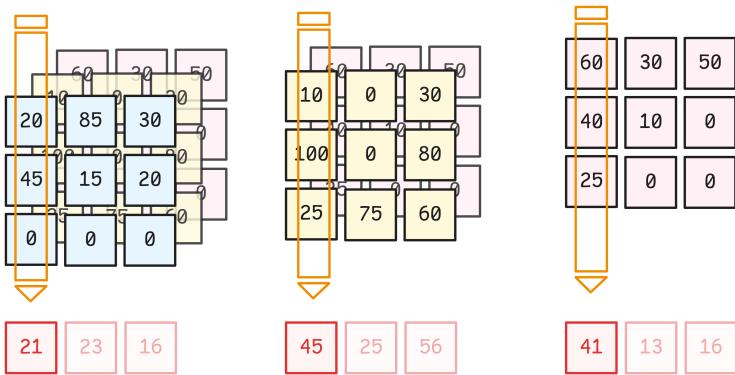


Fig. 4: Proyección 2D del volumen, dirección x, criterio promedio

Al finalizar este proceso, nuestro sistema debe crear una nueva Imagen y guardarla, el contenido de esta imagen generada es el siguiente:

Listing 2: Resultado ejemplo

```
P2
3 3
56
41 13 16
45 25 56
21 33 16
```

Esta imagen final es la proyección 2D del volumen en la dirección X, donde cada valor representa el promedio de las capas en esa columna.

4 Huffman

Abordada en una época donde era necesario simplificar la transmisión de mensajes pesados, en contraste al despilfarre de almacenamiento actual, la búsqueda de métodos que simplifiquen la transmisión de información fue liderada por David A. Huffman, quien comenzaba la publicación de su algoritmo con la siguiente frase: "Si hay más mensajes que podrían enviarse que tipos de símbolos disponibles, entonces algunos de los mensajes deben usar más de un símbolo." (Tomado y traducido de [1]).

Este principio fue la base para el desarrollo de lo que hoy conocemos como código de Huffman, un algoritmo eficiente para la compresión de datos. El algoritmo trabaja mediante la asignación de códigos binarios a los símbolos de un mensaje, de manera que los símbolos más frecuentes se representen con secuencias de menor longitud, mientras que los menos frecuentes se representen con secuencias más largas. Esto es conocido como codificación de longitud variable.

El código de Huffman es un tipo de codificación de longitud variable, lo que significa que la longitud de los códigos asignados a cada símbolo depende de su frecuencia en el mensaje original. Esta característica permite una representación más eficiente de los

datos, reduciendo el tamaño del mensaje al aprovechar las frecuencias de aparición de los símbolos.

El algoritmo comienza construyendo un arbol binario, donde cada hoja representará un simbolo del mensaje, los símbolos más frecuentes se agrupan en nodos cercanos a la raíz del árbol, mientras que los símbolos menos frecuentes se colocan en los nodos más alejados. Al final, cada símbolo es asignado un código binario único, que es la secuencia de ceros y unos que se obtiene al seguir el camino desde la raíz del árbol hasta el nodo que representa ese símbolo.

Por ejemplo, veamos el arbol que se genera a partir de este último parrafo:

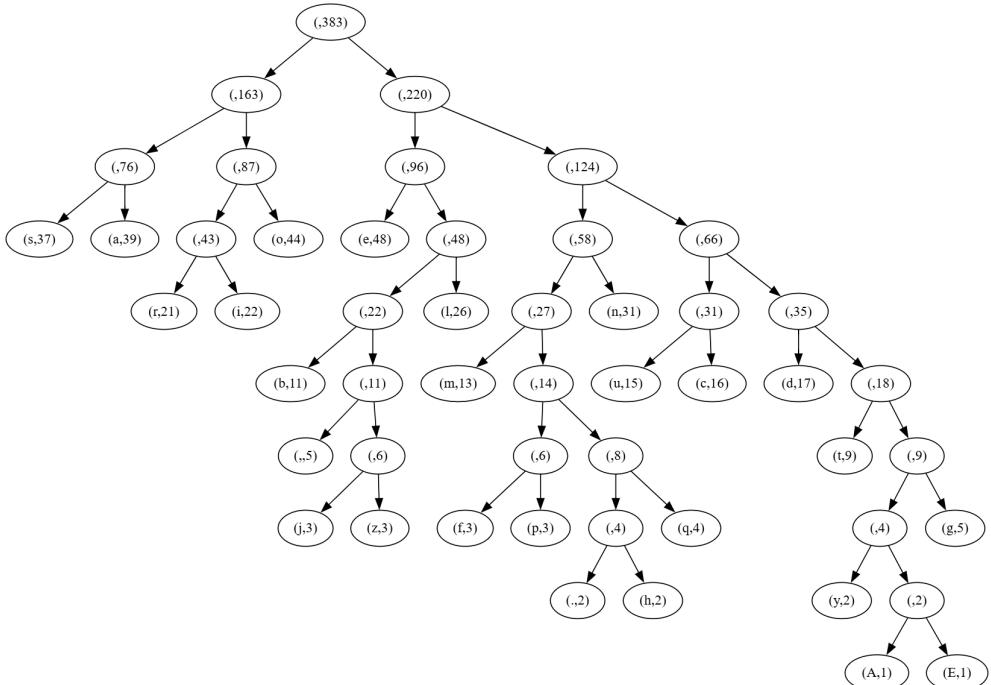


Fig. 5: Arbol Huffman de ejemplo

5 Segmentar con Dijkstra

La segmentación de una imagen puede abordarse utilizando el algoritmo de Dijkstra. Si consideramos cada píxel como un nodo y lo conectamos con sus vecinos adyacentes (aquellos con los que comparte un lado), obtenemos un grafo. En este grafo, las aristas tienen como peso la diferencia de intensidad entre los píxeles que conectan. De este modo, es más fácil “moverse” entre píxeles de intensidades similares.

Si aplicamos el algoritmo de Dijkstra desde un único punto de la imagen, notaremos que el resultado es simplemente una representación plana donde los colores cambian de

forma continua, sin segmentación real. Esta estrategia por sí sola no tiene mucho sentido si buscamos dividir la imagen en regiones significativas. Sin embargo, al ejecutar el algoritmo múltiples veces desde diferentes puntos, podemos identificar segmentos distintos. Por ejemplo, imaginemos un lienzo blanco con un cuadrado negro en el centro: si ejecutamos el algoritmo desde una esquina y luego desde el centro, obtendremos dos regiones claramente diferenciadas: un fondo y una figura central.

Llevando esto a un caso más complejo, podríamos, por ejemplo, distinguir qué segmento corresponde a la boca de un felino, cuál a su cabeza y cuál al fondo.



Fig. 6: Dibujo de un gato

Como humanos, identificar estas partes es intuitivo; para una máquina, en cambio, es una tarea más desafiante. Ahí es donde entra en juego Dijkstra: este algoritmo permite al sistema agrupar píxeles que son similares entre sí, facilitando la segmentación. Así, al intentar dividir una imagen de un gato en fondo, cabeza y boca, podemos obtener una segmentación útil y significativa.

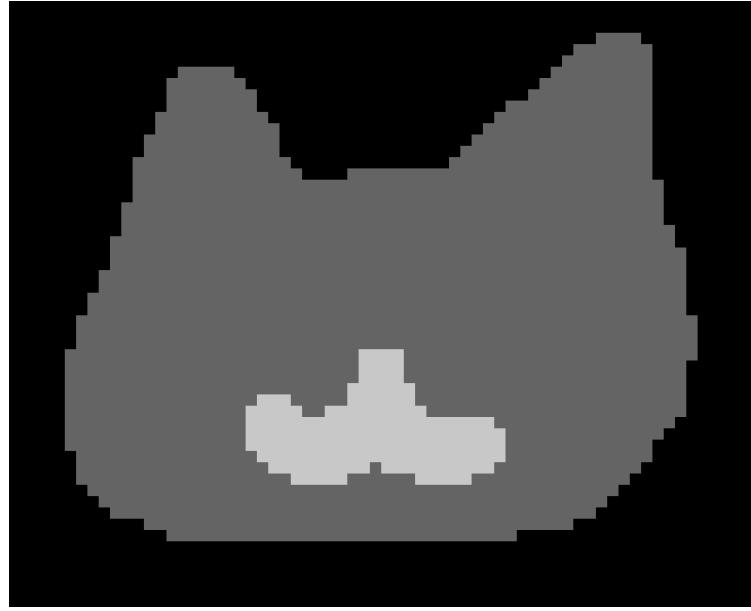


Fig. 7: Resultado de Segmentar el dibujo de un gato

El dibujo original del gato contenía intensidades intermedias que generaban transiciones suaves entre regiones, lo que mezclaba visualmente algunas partes de la imagen. En la versión segmentada, los bordes se han definido con mayor precisión. Aunque esta nueva imagen carece de detalles, resulta útil porque asigna una única intensidad a cada región, permitiéndonos identificar con claridad qué parte de la imagen corresponde a cada elemento específico.

5.1 Dijkstra y como fue implementado

Dijkstra es un algoritmo que encuentra el camino más corto desde un nodo origen a todos los demás en un grafo con pesos no negativos. Utiliza una estructura llamada *min heap* (cola de prioridad mínima) para seleccionar siempre el nodo con la menor distancia acumulada. A medida que recorre el grafo, actualiza las distancias más cortas conocidas hacia los nodos vecinos, asegurando que cada nodo se visite en orden de menor costo posible.

Como Dijkstra recorre todo el grafo, obtenemos la distancia mínima desde el nodo de inicio (nuestra raíz) hasta cada uno de los demás nodos. En el caso de una imagen, cada vez que ejecutamos el algoritmo generamos una matriz del mismo tamaño que la imagen, donde cada celda representa la distancia desde la raíz hasta ese píxel. Al comparar estas matrices entre sí, podemos determinar desde qué punto (o "semilla") llegó primero el recorrido de Dijkstra a cada píxel, lo que nos permite construir la imagen segmentada a partir de estas regiones de influencia.

6 Procedimiento Principal

Nuestro procedimiento general (*main*) está encargado del ciclo infinito que lee comandos enviados por la terminal del programa. En este se inicializa el sistema, el cual se encarga de recibir cada comando de parte del usuario, revisar su formato y luego ejecutarlo si cumple con las condiciones de cada comando.

Debido a la naturaleza modular del sistema, para continuar con el desarrollo del proyecto no fue necesario modificar el procedimiento principal. Las nuevas funcionalidades se integraron fácilmente como módulos independientes, que se acoplaron al flujo ya existente sin afectar su estructura.

Para facilitar al usuario seguir las condiciones de uso de los comandos, se ha implementado un comando *ayuda* que presenta un manual simple de uso para cada uno de ellos.

7 TADs

Los TADs usados en nuestro sistema fueron *Imagen*, *Volumen*, *Memoria*, *Comando*, *Command Manager*, *Sistema*, *DatoHuffman*, *NodoHuffman*, *ArbolHuffman* y *Huffman* a continuación se presenta la relación que se da entre ellos. Vease Fig. 8.

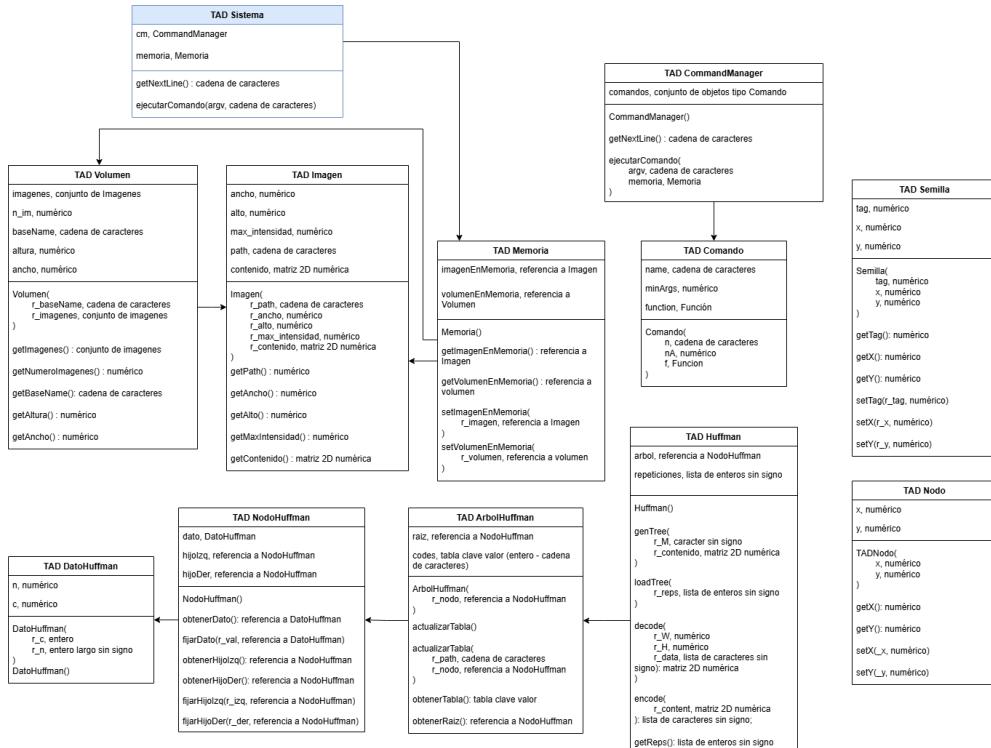


Fig. 8: Diagrama de relación - TADs

Este diseño de TADs fue la base para la implementación de cada una de las funcionalidades propuestas para el proyecto, la especificación y detalles de cada uno de los TADs se encuentra a continuación.

7.1 TAD Imagen

• Estado

- **ancho**, (ancho), numérico, indica el ancho de la imagen, $\text{ancho} \geq 0$.
- **alto**, (alto), numérico, indica el alto de la imagen, $\text{alto} \geq 0$.
- **maxima_intensidad**, (max_intensidad), numérico, $0 \leq \text{maxima_intensidad} \leq 255$.
- **contenido**, (contenido), matriz 2D numérica, es el valor de gris del píxel de la imagen ubicado en la coordenada bidimensional (i, j) , $1 \leq i \leq \text{ancho}$, $1 \leq j \leq \text{alto}$, $0 \leq \text{contenido}(i, j) \leq \text{maxima_intensidad}$.
- **Ubicación** (path), cadena de caracteres, ubicación de la imagen en el sistema (generalmente ubicación relativa).

• Interfaz

- **Imagen(**
 r_ancho, numérico
 r_alto, numérico
 r_max_intensidad, numérico
 r_contenido, matriz 2D numérica
 $)$: Constructor de la imagen.
 - * $\text{r_alto} > 0$
 - * $\text{r_ancho} > 0$
 - * $0 \leq \text{r_max_intensidad} \leq 255$
 - * $\text{r_contenido}(i, j) \quad 1 \leq i \leq \text{ancho}; 1 \leq j \leq \text{alto}; 0 \leq \text{r_contenido}(i, j) \leq \text{r_maxima_intensidad}$
- **getAncho()**: numérico; Retorna el ancho de la imagen.
- **getAlto()**: numérico; Retorna el alto de la imagen.
- **getMaxIntensidad()**: numérico; Retorna la máxima intensidad de la imagen.
- **getContenido()**: matriz 2D; Retorna el contenido de la imagen.
- **getPath()**: cadena de caracteres; Retorna la ubicación de la imagen.

7.2 TAD Volumen

• Estado

- **imagenes**, (imagenes), Conjunto de imágenes, Máximo tamaño de 99, mínimo tamaño de 1.
- **numero_imagenes**, (n_im), numérico, $0 < n_im \leq 99$.
- **Nombre_base**, (baseName), cadena de caracteres, indica el nombre base del volumen, el nombre de cada una de las imágenes contenidas en el volumen debe comenzar con este baseName.

- **altura** (altura), numérico, indica la altura de las imágenes contenidas en el volumen.
- **ancho** (ancho), numérico, indica el ancho de las imágenes contenidas en el volumen.

- **Interfaz**

- **Volumen(**
r_baseName, cadena de caracteres
r_imagenes, conjunto de imagenes
 $)$: Constructor del volumen, recibe el baseName del volumen y cada una de las imágenes, procesa el conjunto de imágenes para encontrar la altura, el ancho y el numero de imágenes.
- **getImagenes()**: conjunto de imágenes; Retorna el conjunto de imágenes del volumen.
- **getNumeroImagenes()**: numérico; Retorna el número de imágenes en el volumen.
- **getBaseName()**: cadena de caracteres; Retorna el nombre base del volumen.
- **getAltura()**: numérico; Retorna la altura de las imágenes en el volumen.
- **getAncho()**: numérico; Retorna el ancho de las imágenes en el volumen.

7.3 TAD Memoria

El uso de un TAD Memoria nos facilita el acceso a los objetos Imagen y Volumen que hayan sido cargados en memoria, el sistema crea un objeto de memoria y lo envia a cada comando.

- **Estado**

- **imagenEnMemoria**, (*ImagenEnMemoria*), Imagen, Contiene la referencia a la imagen utilizada por los comandos.
- **volumenEnMemoria**, (*VolumenEnMemoria*), Volumen, Contiene la referencia al volumen utilizado por los comandos.

- **Interfaz**

- **Memoria()**: Constructor de la memoria.
- **getImagenEnMemoria()**: Imagen; Retorna la referencia de la imagen en memoria.
- **setImagenEnMemoria(*r_imagen, referencia a imagen*)**: Establece la referencia de la imagen en memoria.
- **getVolumenEnMemoria()**: Volumen; Retorna la referencia del volumen en memoria.
- **setVolumenEnMemoria(*r_volumen, referencia a volumen*)**: Establece la referencia del volumen en memoria.

7.4 TAD Comando

- **Estado**

- **nombre**, (nombre), cadena de caracteres, indica el nombre del comando.
- **minArgs**, (minArgs), numérico, indica la cantidad mínima de argumentos requeridos.
- **function**, (function), Función, función que ejecuta el comando.

- **Interfaz**

- **Comando(**
n, cadena de caracteres
nA, numérico
f, Función
 $)$: Constructor del comando.
- **ejecutar(**
argv, cadena de caracteres
memoria, Memoria
 $)$: numérico; Ejecuta el comando con los argumentos y memoria proporcionados.

7.5 TAD Command Manager

- **Estado**

- **comandos**, (comandos), Estructura lineal de comandos, Lista de comandos disponibles.

- **Interfaz**

- **CommandManager()**: Constructor de CommandManager. Crea un Command Manager que contiene en la estructura lineal de comandos todos los comandos a utilizar.
- **encontrarComando(nombre, cadena de caracteres)**: C, Comando; Encuentra un comando, buscando por el nombre recibido y retorna el comando encontrado, o un valor nulo si no existe.
- **getNextLine()**: Estructura lineal de cadena de caracteres; Obtiene la siguiente línea de entrada del usuario y retorna un vector con los argumentos de la línea ingresada.
- **ejecutarComando(**
argv, estructura lineal de cadena de caracteres
memoria, Memoria
 $)$: numérico; Ejecuta un comando a partir de una lista de argumentos. Recibe la lista de argumentos que representan el comando a ejecutar y retorna el código del comando ejecutado.

7.6 TAD Sistema

- **Estado**

- **Administrador de Comandos** (cm), Administrador de comandos encargado principalmente de recibir comandos de parte del usuario y realizar sus operaciones.

- **Memoria** (memoria), Memoria del sistema, esta se manipula para leer o almacenar la imagen y el volumen en la memoria del programa.

- **Interfaz**

- **getNextLine()**: cadena de caracteres; lee un string enviado por el usuario hasta un salto de linea.
- **ejecutarComando(argv, conjunto de cadenas de caracteres)**: numérico; ejecuta el comando con los argumentos especificados en el conjunto de cadenas de caracteres argv y devuelve un valor numérico indicando si su ejecución fue exitosa (0) o hubo error (1).

7.7 TAD DatoHuffman

- **Estado**

- **caracter**, (c), entero, representa el carácter o valor a codificar en el árbol de Huffman.
- **cantidad**, (n), entero largo sin signo, representa la frecuencia del carácter.

- **Interfaz**

- **DatoHuffman(r_c, entero r_n, entero largo sin signo)**: Constructor que inicializa un DatoHuffman con un carácter y su frecuencia.
- **DatoHuffman()**: Constructor por defecto del DatoHuffman.

7.8 TAD NodoHuffman

- **Estado**

- **dato**, (dato), DatoHuffman, almacena la información asociada al nodo.
- **hijoIzq**, (hijoIzq), referencia a NodoHuffman, referencia al hijo izquierdo del nodo.
- **hijoDer**, (hijoDer), referencia a NodoHuffman, referencia al hijo derecho del nodo.

- **Interfaz**

- **NodoHuffman()**: Constructor por defecto del nodo de Huffman.
- **obtenerDato()**: referencia a DatoHuffman; Retorna una referencia al dato almacenado en el nodo.
- **fijarDato(r_val, referencia a DatoHuffman)**; Establece el dato almacenado en el nodo.
- **obtenerHijoIzq()**: referencia a NodoHuffman; Retorna el hijo izquierdo del nodo.
- **obtenerHijoDer()**: referencia a NodoHuffman; Retorna el hijo derecho del nodo.
- **fijarHijoIzq(r_izq, referencia a NodoHuffman)**; Establece el hijo izquierdo del nodo.

- **fijarHijoDer(r_der, referencia a NodoHuffman);** Establece el hijo derecho del nodo.

7.9 TAD ArbolHuffman

- **Estado**

- **raiz,** (raiz), referencia a NodoHuffman, nodo raíz del árbol de Huffman.
- **codes,** (codes), estructura que asocia enteros con cadenas de caracteres, almacena los códigos de Huffman para cada valor.

- **Interfaz**

- **ArbolHuffman(r_nodo, referencia a NodoHuffman);** Constructor del árbol de Huffman, recibe un nodo que será la raíz del árbol.
- **actualizarTabla();** Actualiza la tabla de códigos de Huffman para todos los nodos del árbol.
- **actualizarTabla(
r_path, cadena de caracteres
r_nodo, referencia a NodoHuffman)**
); Método auxiliar para actualizar la tabla de códigos, recibe la ruta actual y el nodo que se está procesando.
- **obtenerTabla();** referencia a codes; Retorna una referencia a la tabla de códigos de Huffman.
- **obtenerRaiz();** referencia a NodoHuffman; Retorna el nodo raíz del árbol de Huffman.

7.10 TAD Huffman

- **Estado**

- **arbol,** (arbol), ArbolHuffman, referencia al árbol de Huffman generado para la codificación y decodificación.
- **repeticiones,** (reps), lista de enteros sin signo, donde el índice o llave representa la intensidad del pixel y el valor representa la frecuencia de aparición de esa intensidad.

- **Interfaz**

- **Huffman();** Constructor del objeto Huffman.
- **genTree(
r_M, carácter sin signo
r_contenido, matriz 2D numérica)**
); Genera el árbol de Huffman basado en la intensidad máxima y el contenido de la imagen.

* r_M representa el valor máximo de intensidad ($0 \leq r_M \leq 255$)

* r_contenido es la matriz 2D donde cada entero representa un píxel, $0 \leq r_contenido(i, j) \leq r_M$

- **loadTree(r_reps, lista de enteros sin signo)**: Carga un árbol Huffman a partir de una lista de frecuencias.
 - * r_reps[i] representa la cantidad de apariciones del valor de intensidad i
- **decode(**
r_W, numérico
r_H, numérico
r_data, lista de caracteres sin signo
 $)$: matriz 2D numérica; Decodifica datos comprimidos y retorna una matriz 2D de píxeles.
 - * r_W es el ancho de la imagen a decodificar ($r_W \geq 0$)
 - * r_H es el alto de la imagen a decodificar ($r_H \geq 0$)
 - * r_data son los datos codificados en Huffman
- **encode(r_content, matriz 2D numérica)**: lista de caracteres sin signo; Codifica la matriz de píxeles y retorna un lista de bytes comprimidos.
 - * r_content es la matriz de píxeles a codificar
- **getReps()**: lista de enteros sin signo; Retorna el lista de frecuencias de cada valor de intensidad.

7.11 TAD Semilla

- **Estado**
 - **tag**, (tag), numérico, representa la etiqueta o tipo de la semilla.
 - **x**, (x), numérico, representa la coordenada horizontal de la semilla.
 - **y**, (y), numérico, representa la coordenada vertical de la semilla.
- **Interfaz**
 - **Semilla(**
r_tag, numérico
r_x, numérico
r_y, numérico
 $)$: Constructor del objeto Semilla, inicializa la etiqueta y las coordenadas.
 - **getTag()**: numérico; Retorna la etiqueta de la semilla.
 - **getX()**: numérico; Retorna la coordenada horizontal de la semilla.
 - **getY()**: numérico; Retorna la coordenada vertical de la semilla.
 - **setTag(r_tag, numérico)**: Establece una nueva etiqueta para la semilla.
 - **setX(r_x, numérico)**: Establece una nueva coordenada horizontal para la semilla.
 - **setY(r_y, numérico)**: Establece una nueva coordenada vertical para la semilla.

7.12 TAD Nodo

- **Estado**

- **x**, (x), numérico, representa la coordenada horizontal del nodo.
 - **y**, (y), numérico, representa la coordenada vertical del nodo.

- Interfaz

- **TADNodo(**
r_x, numérico
r_y, numérico
): Constructor del nodo, inicializa las coordenadas del nodo.
 - **getX()**: numérico; Retorna la coordenada horizontal del nodo.
 - **getY()**: numérico; Retorna la coordenada vertical del nodo.
 - **setX(**r_x, numérico**)**: Establece una nueva coordenada horizontal para el nodo.
 - **setY(**r_y, numérico**)**: Establece una nueva coordenada vertical para el nodo.

8 Comandos

A continuación veremos cada uno de los comandos implementados en la entrega del **Componente 1** del proyecto de curso.

8.1 ayuda [comando]

- Entradas

- Si no se envia comando muestra la lista de comandos disponibles
 - Si se envia un comando muestra la descripción y la forma de usar el comando

- Salidas

- Lista de comandos si no se proporciona un parámetro comando
 - Descripción detallada del comando solicitado si se envía uno
 - Mensaje de error si el comando no existe

- **Condiciones**

- Si el usuario envia un comando válido se muestra su descripción
 - Si el usuario ingresa un comando inexistente, se muestra un mensaje de error
 - Si el usuario no envia nada, muestra la lista de comandos

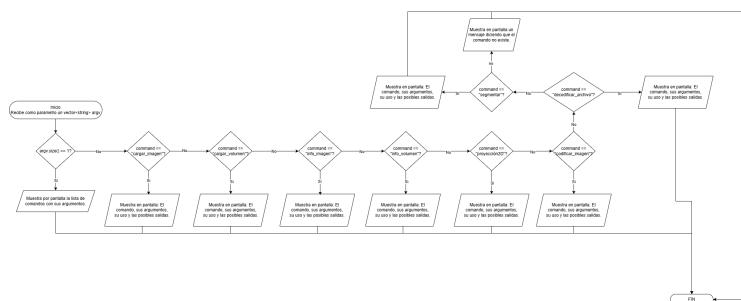


Fig. 9: Diagrama de flujo comando ayuda

8.2 cargar_imagen [nombre_imagen.pgm]

Este comando esta encargado de dado el path de un archivo, intente leerlo, se espera que este archivo contenga una imagen en formato PGM, de lo contrario no podrá ser procesado, luego lo guardará en la memoria del sistema. Las funciones usadas en este comando estan explicadas y se muestra su diseño en la subsección especifica con su nombre.

• Entradas

- Nombre del archivo de imagen en formato PGM.

• Salidas

- Mensaje de confirmación si se leyó la imagen correctamente, de lo contrario un mensaje de error.

• Condiciones

- La imagen existe en el computador
- Se carga únicamente una imagen en memoria
- La imagen sigue el formato PGM
- Las dimensiones son validas
- Cada pixel está en los rangos (0 - 255)

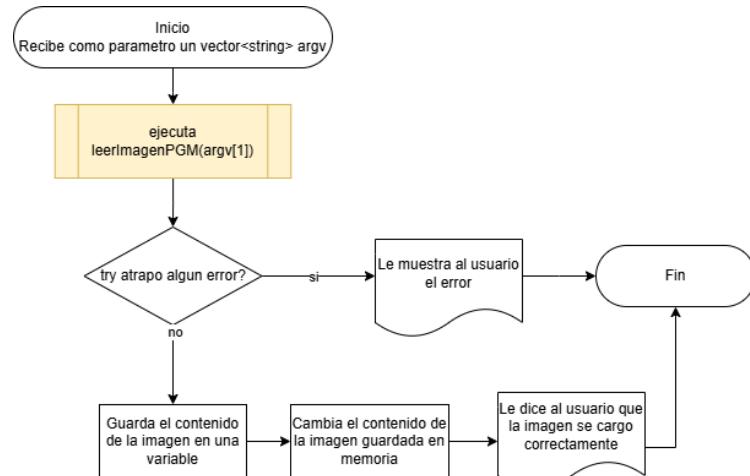


Fig. 10: Diagrama de flujo comando cargar_imagen

8.2.1 leerImagenPGM

Esta función es la encargada de leer el archivo y construir un objeto Imagen en base a la información leída. También, es la encargada de decidir si un archivo tiene o no el formato deseado, si no es el caso, enviará el error que será procesado por *cargar_imagen*.

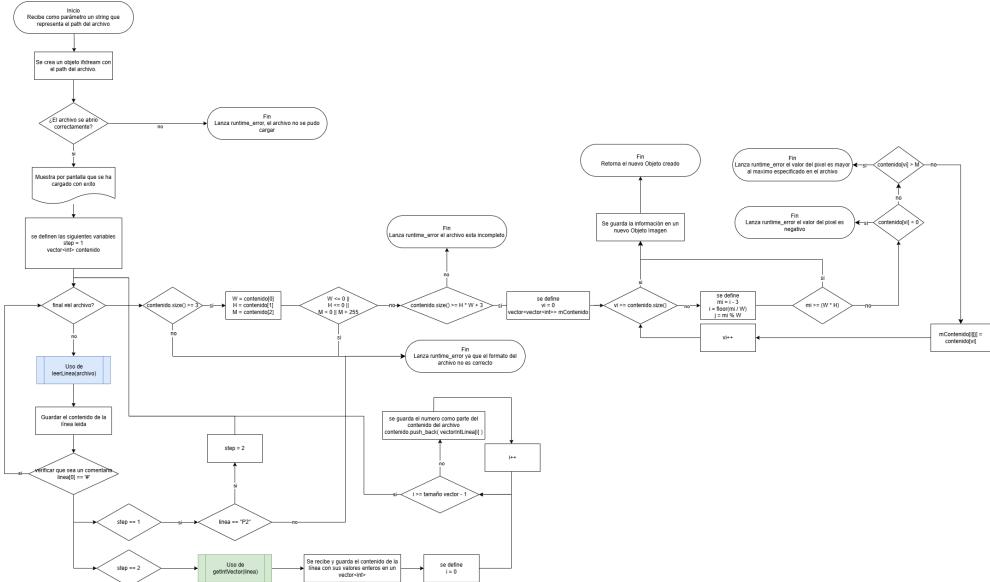


Fig. 11: Diagrama de flujo función leerImagenPGM

8.2.2 leerLineaArchivo

Generalizamos la lectura del archivo a leerlo linea por linea, por lo que esta función solo esta encargada de leer una linea y eliminar los posibles espacios que existan al inicio o al final de la linea, es decir, si tenemos " hola mundo ", esta función nos devolverá "hola mundo".

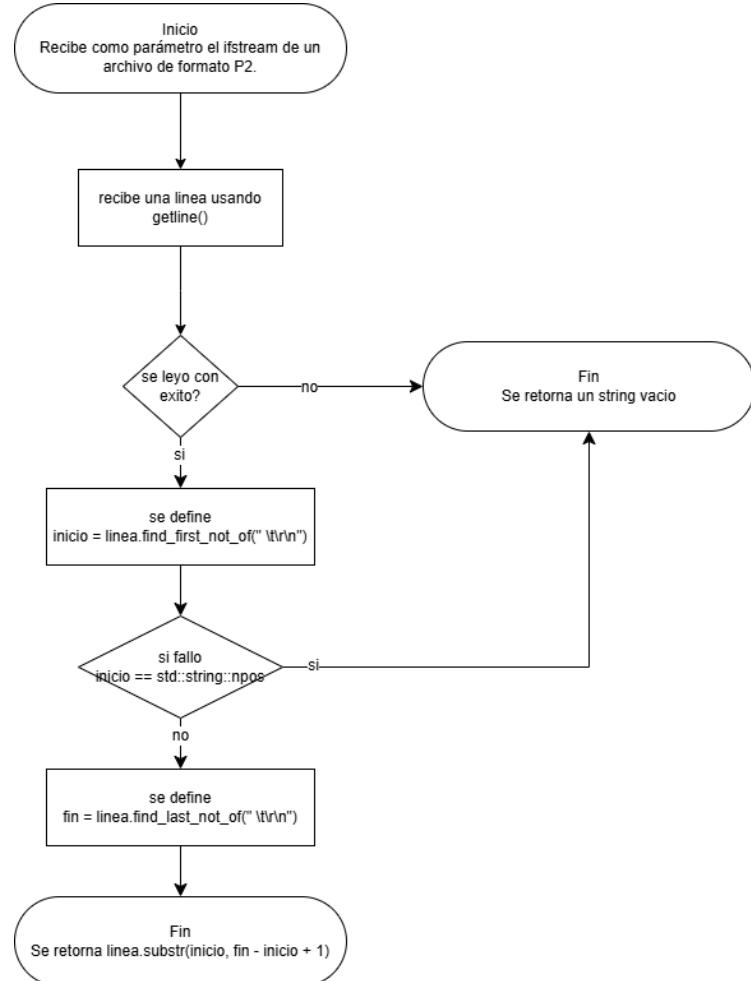


Fig. 12: Diagrama de flujo función leerLineaArchivo

8.2.3 getIntVectorFromStr

A partir de la segunda linea leída por *leerLineaArchivo*, todo el archivo (además de los comentarios) solo debe contener valores numéricicos, por lo que se usa esta función que dada una cadena de caracteres (línea leída desde el archivo) lo convertirá a una estructura lineal de enteros.

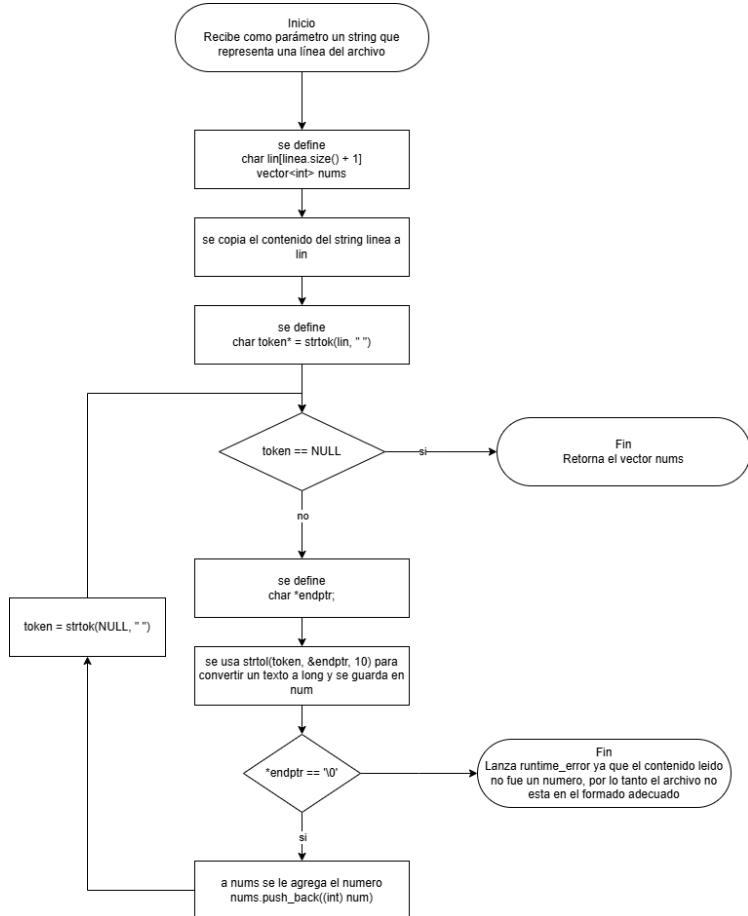


Fig. 13: Diagrama de flujo función getIntVectorFromStr

8.3 cargar_volumen [nombre_base] [n_im]

- Entradas

- Prefijo o nombre base de las imágenes que componen el volumen
- Cantidad de imágenes que se deben cargar en el volumen (n_im)

- Salidas

- Mensaje de confirmación si se cargo el volumen correctamente, de lo contrario un mensaje de error.

- Condiciones

- Las imágenes desde 01 hasta n_im existen en el computador
- Se carga un unico volumen en la memoria

- El volumen tiene minimo 1 imagen y maximo 99
- Cada imagen que compone el volumen es valida
- Todas las imagenes tienen el mismo tamaño

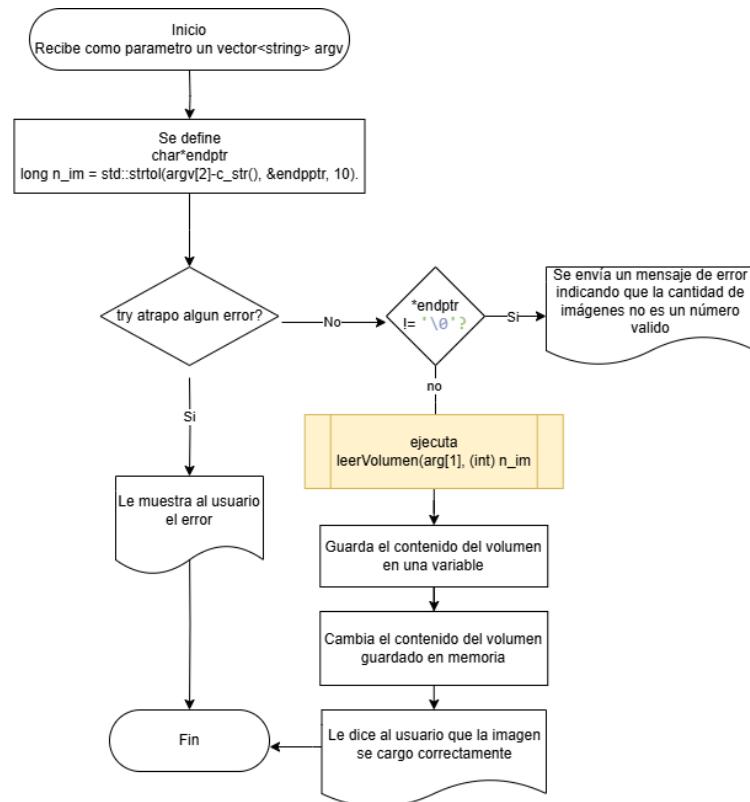


Fig. 14: Diagrama de flujo comando cargar_volumen

8.3.1 leerVolumen

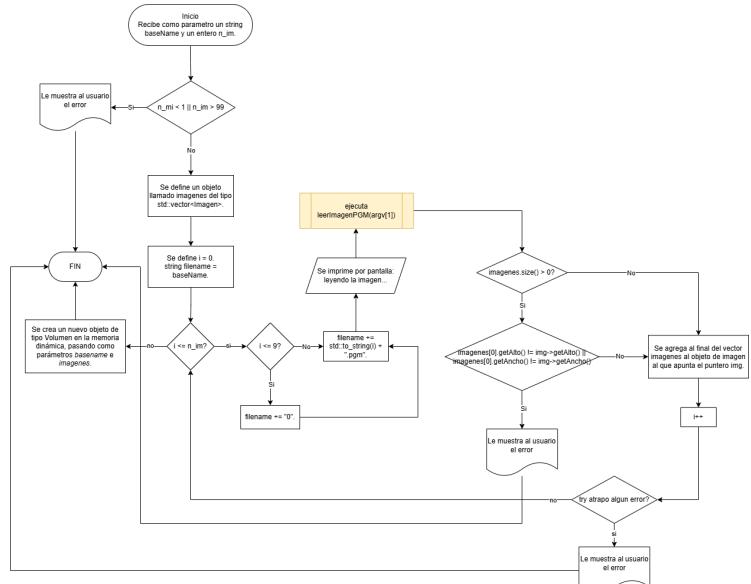


Fig. 15: Diagrama de flujo funcion leerVolumen

8.4 info_imagen

- **Salidas**

- Si la imagen esta cargada muestra el Nombre, ancho y alto de la imagen cargada, de lo contrario mensaje de error.

- **Condiciones**

- Debe haber una imagen cargada en la memoria del sistema

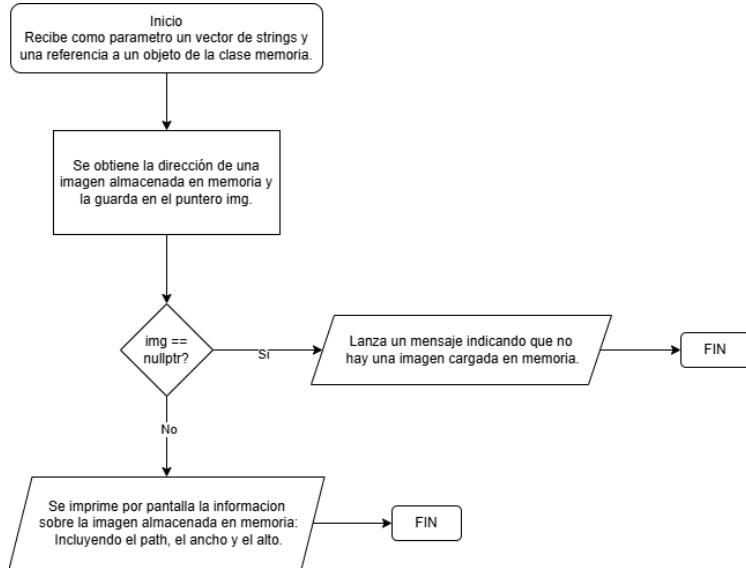


Fig. 16: Diagrama de flujo comando info_imagen

8.5 info_volumen

- **Salidas**

- SI el volumen esta cargado muestra el Nombre base, cantidad de imagenes, ancho y alto de cada imagen del volumen.

- **Condiciones**

- Debe haber un volumen cargado en la memoria del sistema

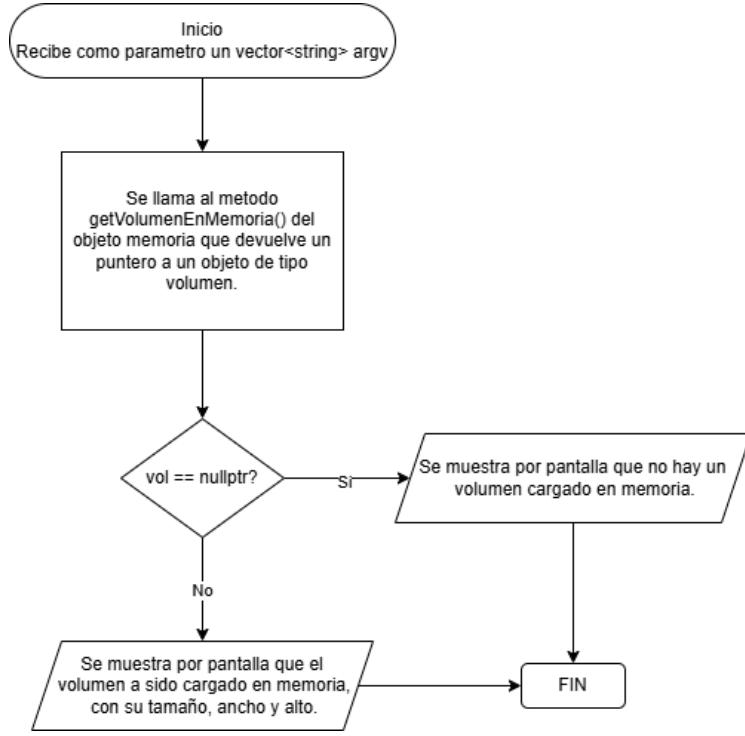


Fig. 17: Diagrama de flujo comando info_volumen

8.6 proyeccion2D [direccion] [criterio] [nombre_archivo.pgm]

- Entradas

- Dirección (x, y, z)
- Criterio (mínimo, maximo, promedio, mediana)
- Nombre o path donde se guardará la imagen resultante de la proyección

- Salidas

- Si fue posible realizar la proyección y guardar el archivo, Archivo PGM con la proyección generada y un mensaje de confirmación de lo contrario un mensaje de error.

- Condiciones

- Debe haber un volumen cargado en memoria
- La dirección debe ser x, y o z
- El criterio debe ser minimo, maximo, promedio o mediana
- Debe ser posible escribir el archivo en el lugar especificado

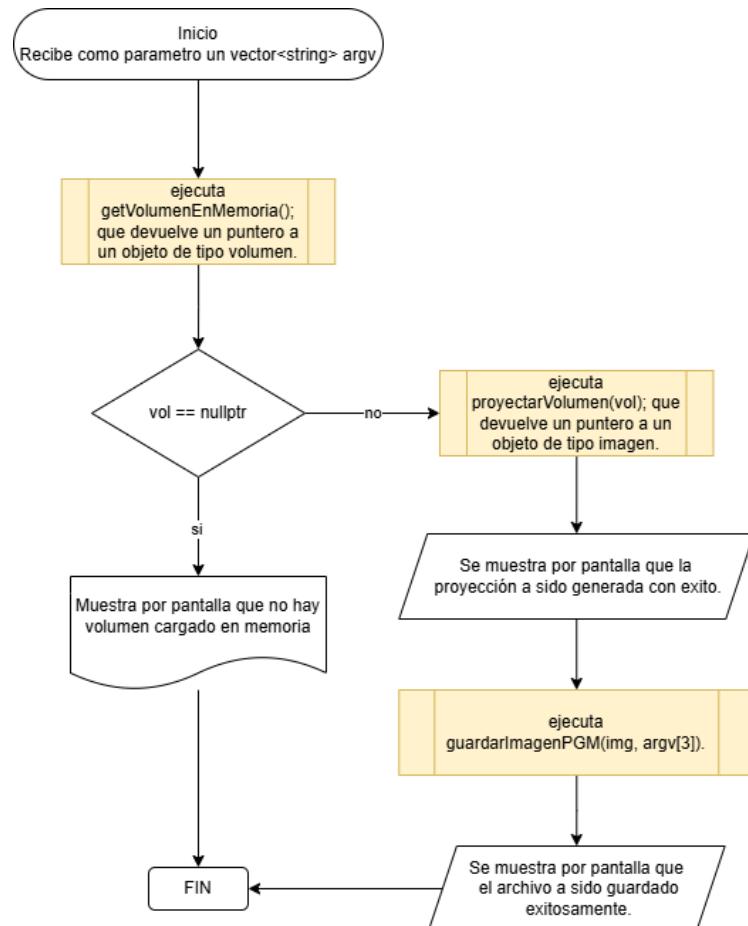


Fig. 18: Diagrama de flujo comando proyeccion2D

8.6.1 lanzarRayo

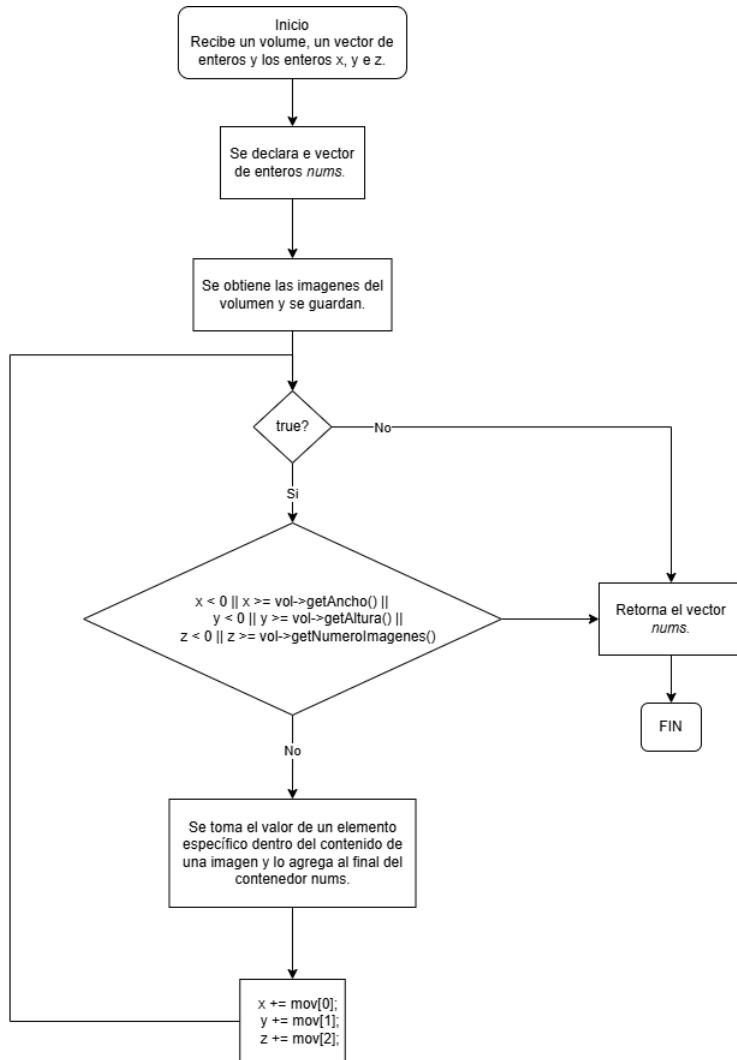


Fig. 19: Diagrama de flujo función lanzarRayo

8.6.2 calcularPixel

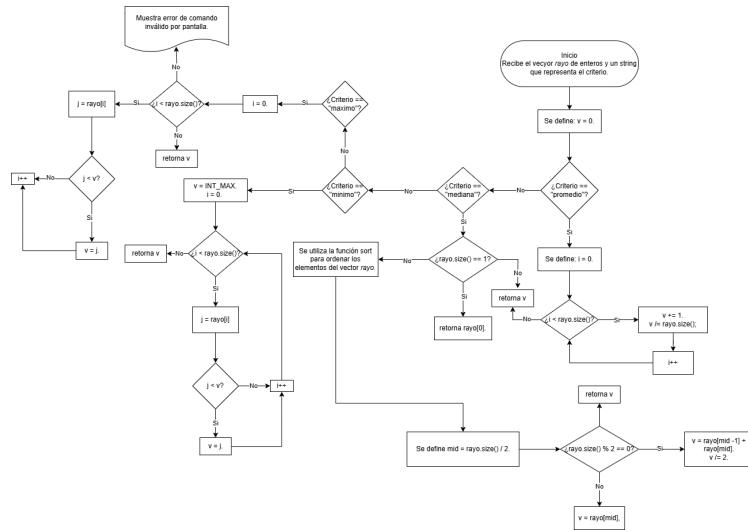


Fig. 20: Diagrama de flujo función calcularPixel

8.6.3 proyectarVolumen

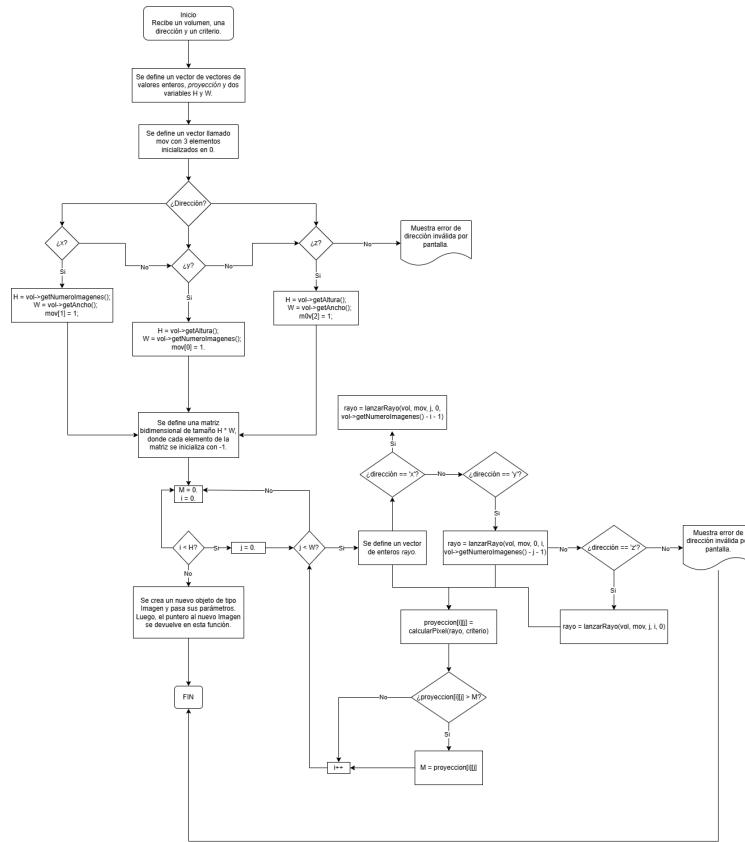


Fig. 21: Diagrama de flujo función proyectarVolumen

8.6.4 guardarImagenPGM

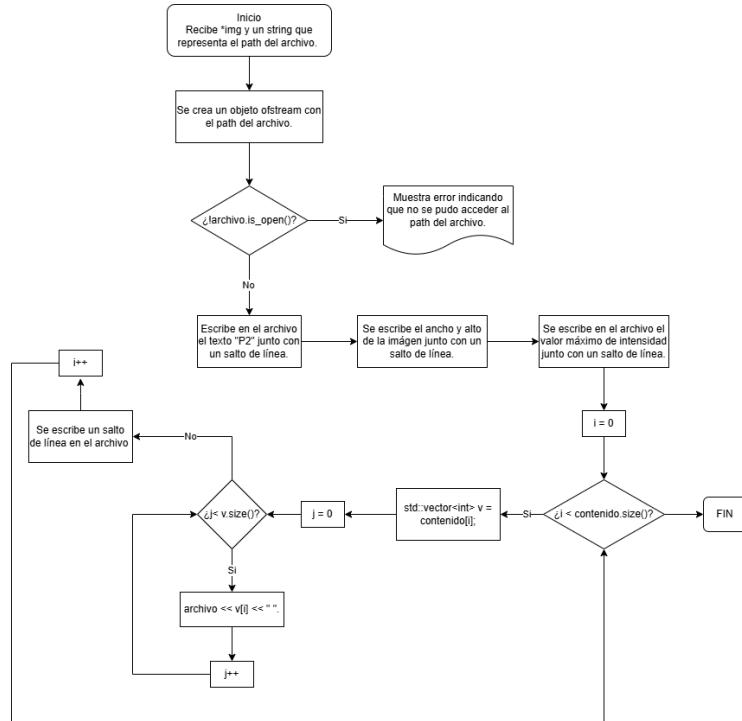


Fig. 22: Diagrama de flujo función guardarImagenPGM

8.7 salir

- Salidas

- Sale del programa sin errores (código 0)

A continuación veremos cada uno de los comandos implementados en la entrega del **Componente 2** del proyecto de curso.

8.8 codificar_imagen

Se implementa la compresión de imágenes mediante el algoritmo de Huffman. La función verifica si existe una imagen cargada en memoria Crea un codificador Huffman y genera su árbol de codificación basado en los datos de la imagen, luego guarda la imagen codificada en un archivo binario especificado por el usuario.

- Entradas

- Argumentos del comando.

- Referencia a un objeto de tipo **Memoria**.
- Imagen actualmente cargada en memoria (puntero a imagen).

• Salidas

- Mensaje en pantalla: “No hay una imagen cargada en memoria.” si el puntero es **nullptr**.
- Mensaje: “Codificando imagen...” si hay una imagen cargada.
- Llamada al método **guardarImagenHUFF()**.
- Mensaje: “El archivo se ha guardado con éxito.” o “Error.” dependiendo del resultado.

• Condiciones

- Si **img == nullptr**, se imprime mensaje de error y se retorna 0.
- Si hay imagen, se crea objeto Huffman y se procede al procesamiento.
- Si ocurre una excepción al guardar, se muestra “Error.” y se retorna 1.
- Si no ocurre excepción, se muestra mensaje de éxito y se retorna 0.

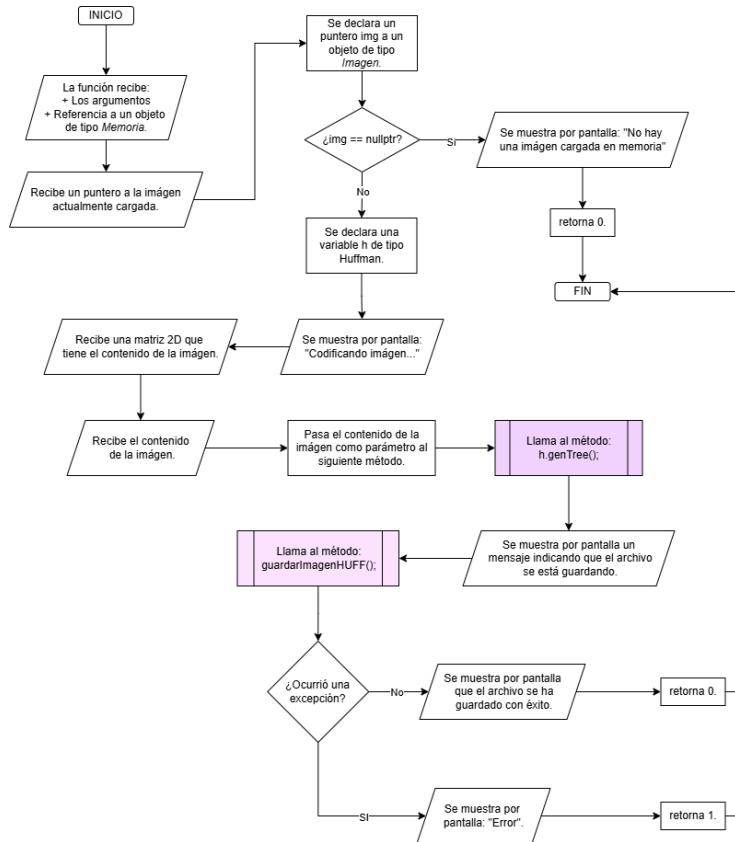


Fig. 23: Diagrama de flujo comando codificar_imagen

8.8.1 actualizar_tabla

- Entradas

- `path`: El camino actual en la codificación (por ejemplo, una cadena de 0s y 1s).
- Un puntero a un nodo del árbol de Huffman.

- Salidas

- Actualización del mapa de códigos (`codes`), donde la clave es el carácter y el valor es el camino binario correspondiente.

- Condiciones

- Si el nodo es una hoja:

- * Se actualiza el mapa `codes`.

- Si el nodo tiene hijo izquierdo:

- * Se añade un 0 al camino.

- * Se llama recursivamente a `actualizarTabla(path, nodoIzquierdo)`.

- Si el nodo tiene hijo derecho:

- * Se añade un 1 al camino.

- * Se llama recursivamente a `actualizarTabla(path, nodoDerecho)`.

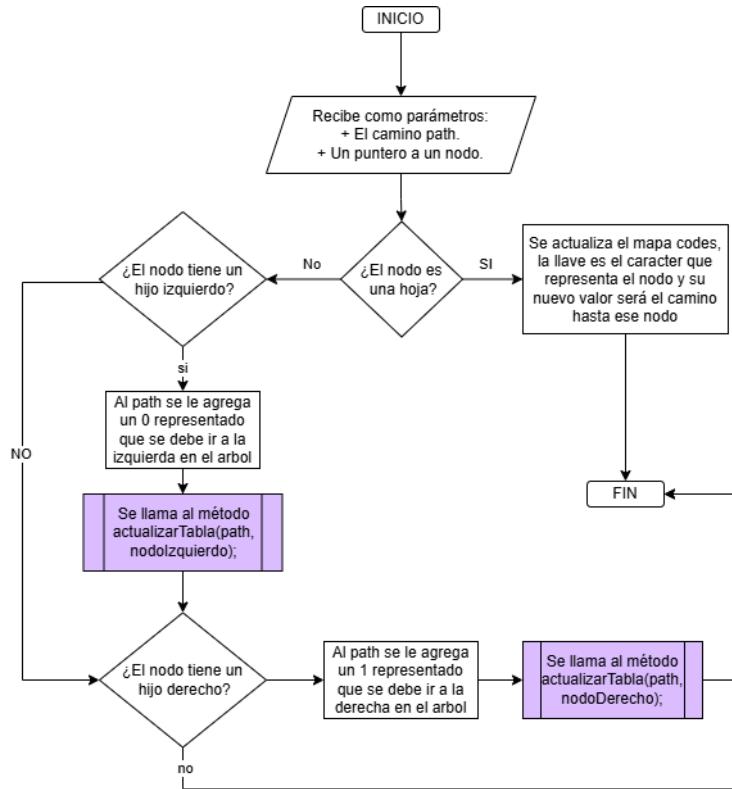


Fig. 24: Diagrama de flujo método actualizar_tabla

8.8.2 load_tree

- Entradas

- Un vector que representa las repeticiones de los valores (frecuencias).

- Salidas

- Árbol de Huffman generado con base en las frecuencias de los datos.
- Llamada al método `arbol->actualizarTabla()`.

- Condiciones

- Se declara una cola de prioridad mínima q .
- Para cada índice i en el vector de frecuencias:
 - * Si `reps[i] != 0`, se crea un `NodoHuffman` y un `DatoHuffman`.
 - * Se llama al método `n->fijarDatos(val)`.
 - * El nodo se inserta en la cola de prioridad.
- Mientras la cola tenga al menos dos nodos:

- * Se extraen los dos nodos con menor frecuencia.
- * Se crean nuevos objetos **DatoHuffman** y **NodoHuffman**.
- * Se asignan los nodos como hijos izquierdo y derecho del nuevo nodo.
- * El nuevo nodo se inserta de nuevo en la cola de prioridad.

- Al finalizar el proceso:

- * Se elimina el nodo de mayor prioridad.
- * Se crea un árbol de tipo **ArbolHuffman** con ese nodo como raíz.
- * Se llama al método **arbol->actualizarTabla()**.

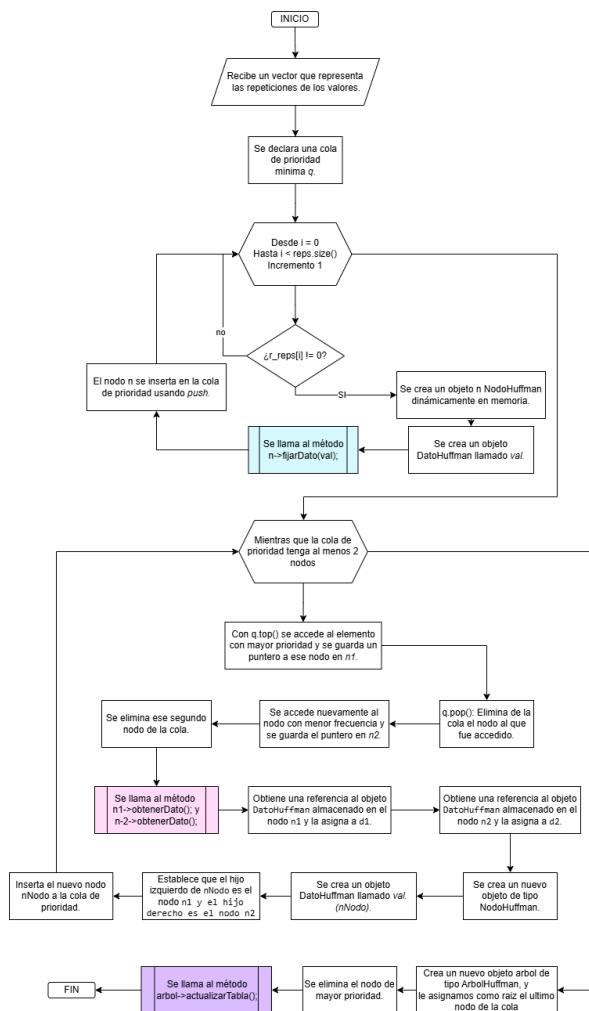


Fig. 25: Diagrama de flujo método load_tree

8.8.3 gen_tree

- **Entradas**

- Una matriz de enteros que representa el contenido de la imagen (valores entre 0 y M).

- **Salidas**

- Llamada al método `loadTree()` después de procesar la matriz.

- **Condiciones**

- Se define un vector `reps` de tipo `unsigned long` con longitud $M + 1$.
 - Para cada fila de la imagen:

- * Para cada píxel de la fila:

- Se incrementa en 1 el valor correspondiente en el vector `reps`.

- Al finalizar el recorrido, se invoca el método `loadTree()`.

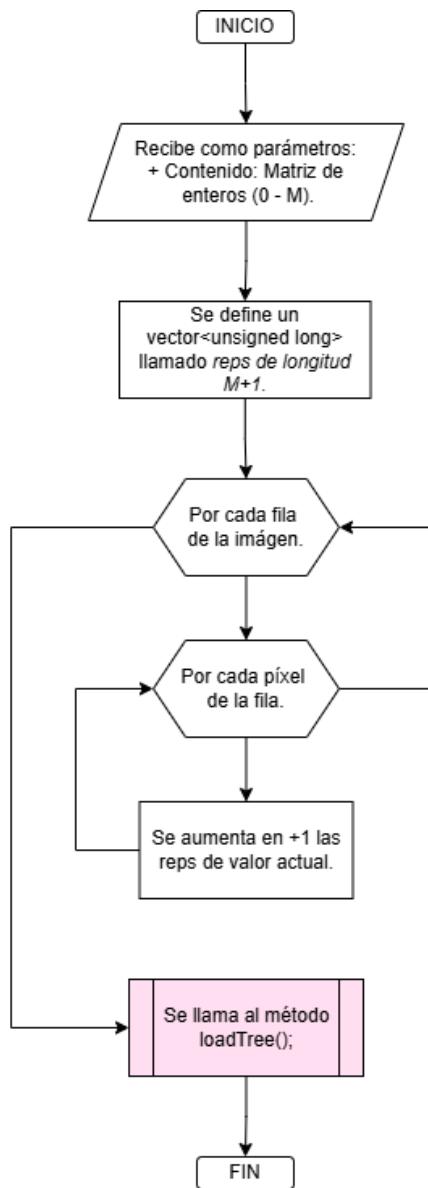


Fig. 26: Diagrama de flujo método gen_tree

8.8.4 encode

La codificación se hace de tal forma que el archivo binario escrito represente bit a bit cada uno de los caminos que se deben tomar desde el arbol para llegar a cada nodo raiz, por esto se debe usar un buffer, los caminos generalmente no ocupan 8 bits, por

lo que se debe tener en cuenta que no sabemos donde empieza o donde terminan cada uno de los caminos, para esto debemos escribirlo bit a bit, veamos un ejemplo:

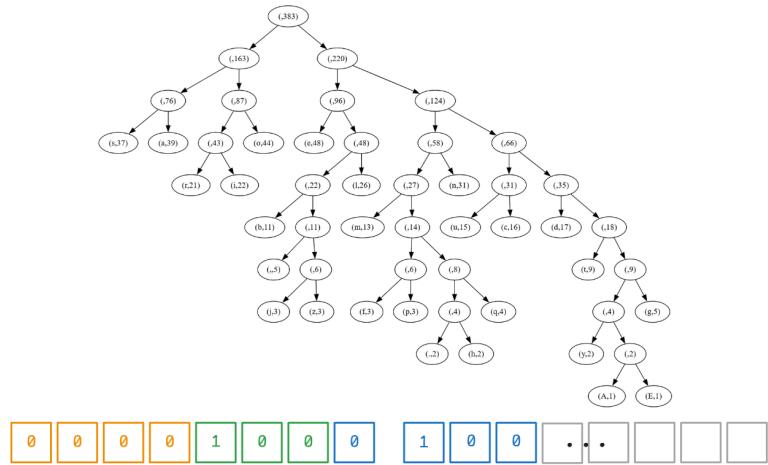


Fig. 27: Ejemplo huffman

En este ejemplo escribimos la palabra "ser", donde la s esta de color naranja, la e de color verde y la r de color azul.

- **Entradas**

- Recibe como parámetro un vector de `unsigned char` con el contenido de la imagen.

- **Salidas**

- Devuelve un vector de bytes como parte de la codificación Huffman.

- **Condiciones**

- Si `bit_pos == 8`, se agrega el byte al vector de salida y se reinicia el buffer.
- Si `bit_pos != 0` al final, se completa el byte restante desplazando y agregando ceros.

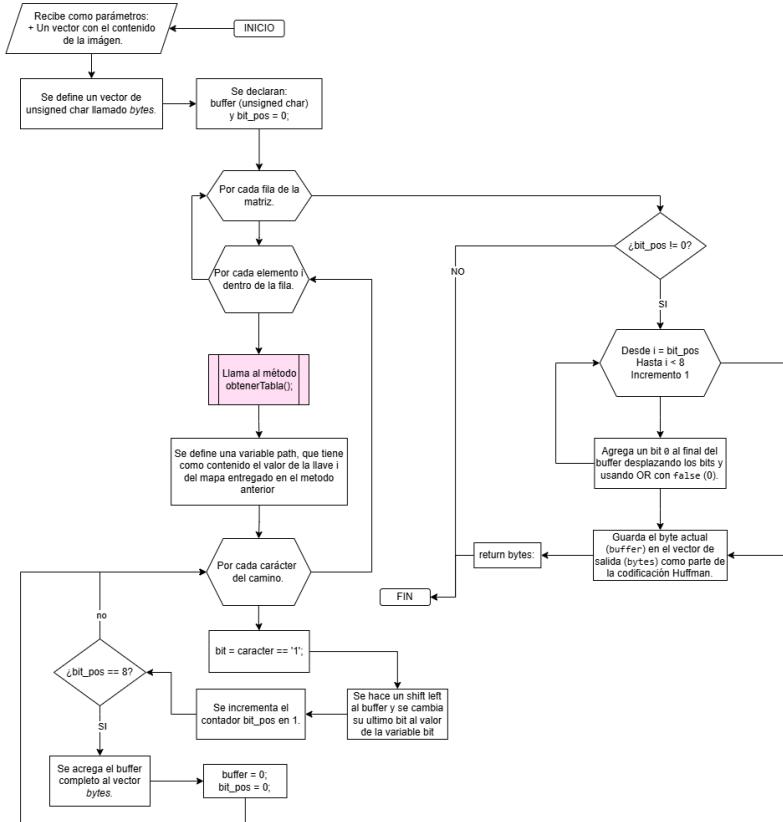


Fig. 28: Diagrama de flujo método encode

8.8.5 guardar_imagen_huff

- Entradas

- Recibe como parámetros:

- * Un puntero a un objeto de tipo `Imagen`.
- * Un vector de tipo `std::vector<unsigned long>` que representa las repeticiones.
- * Un vector de tipo `std::vector<unsigned char>` que contiene los datos comprimidos.
- * Una cadena de texto (`std::string`) con la ruta del archivo a generar.

- Salidas

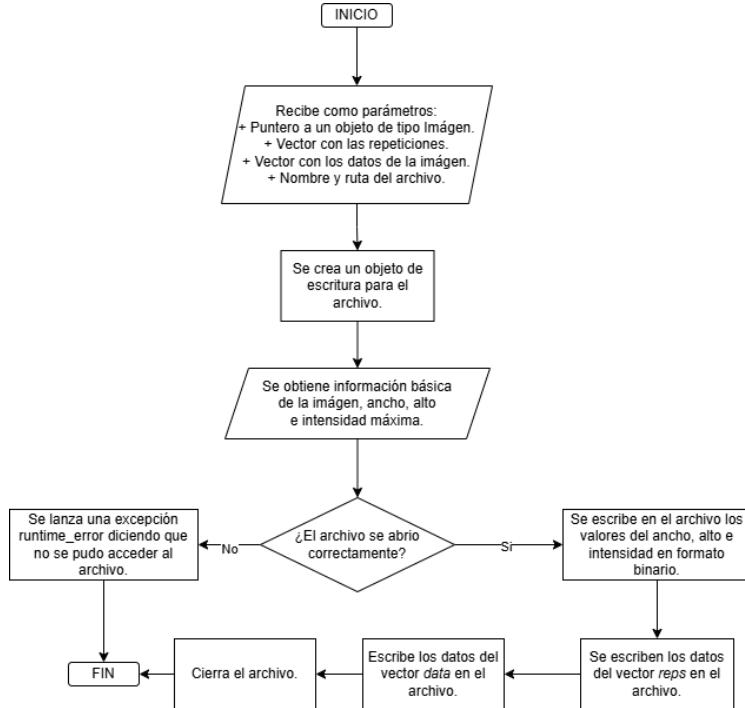
- Se genera un archivo binario con formato HUFF que contiene:

- * Ancho y alto de la imagen (2 bytes cada uno).
- * Valor máximo de intensidad (1 byte).

- * Vector de repeticiones.
- * Datos comprimidos de la imagen.

- **Condiciones**

- Si el archivo no se puede abrir correctamente, se lanza una excepción de tipo `runtime_error`.



8.9 decodificar_archivo

Este comando se encarga de leer una imagen binaria codificada en formato Huffman, decodificarla y almacenarla en memoria para luego guardarla nuevamente como un archivo en formato PGM. El proceso incluye mensajes informativos sobre el progreso y manejo de errores mediante excepciones para asegurar una ejecución controlada de la operación.

- **Entradas**

- Vector de strings (`argv`).
- Referencia a un objeto de tipo `Memoria`.

- **Salidas**

- Imagen decodificada almacenada en memoria.
- Archivo de imagen guardado en formato PGM.
- Mensajes por pantalla indicando el estado del proceso.

- **Condiciones**

- Si ocurre una excepción, se muestra "Error" y se retorna 1.
- Si todo es correcto, se imprime un mensaje de éxito y se retorna 0.

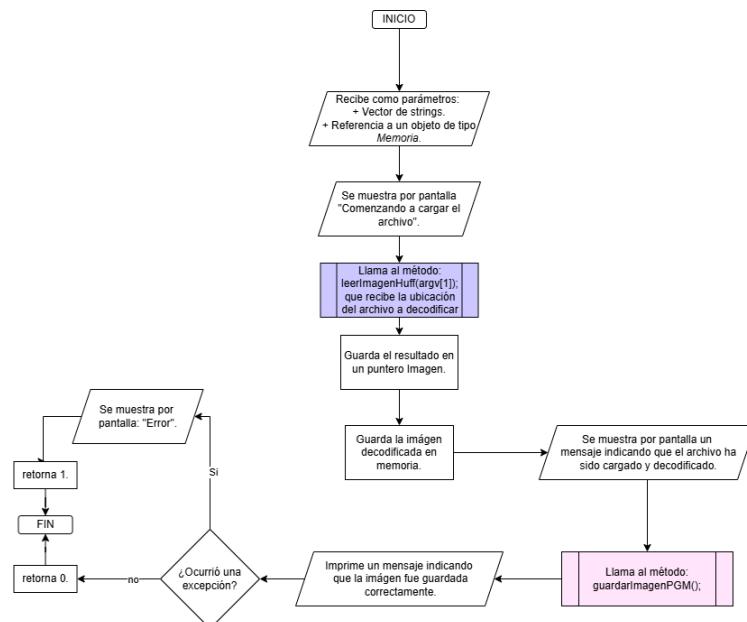


Fig. 30: Diagrama de flujo comando decodificar_archivo

8.9.1 leer_imagen_huff

- **Entradas**

- Ruta al archivo que se va a leer.

- **Salidas**

- Imagen decodificada (retornada como resultado).
- Mensajes de error si el archivo no puede abrirse.

- **Condiciones**

- Si el archivo no se puede abrir, se lanza una excepción y se muestra un mensaje de error.
- El proceso se repite mientras no se llegue al final del archivo.

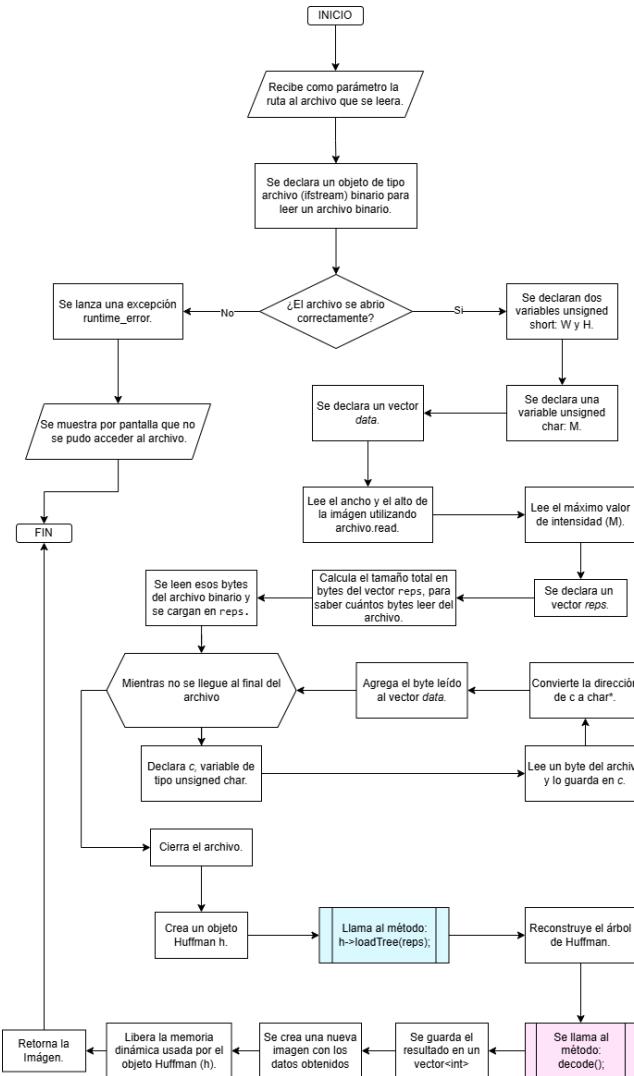


Fig. 31: Diagrama de flujo método leer_imagen_huff

8.9.2 get_raw

- Entradas

- Recibe como parámetro un vector **data** que contiene los datos comprimidos en binario.

- Salidas

- Devuelve una cola de booleanos con los bits extraídos de los bytes del vector.

- **Condiciones**

- Cada byte del vector representa 8 bits que deben ser descompuestos y almacenados en orden.

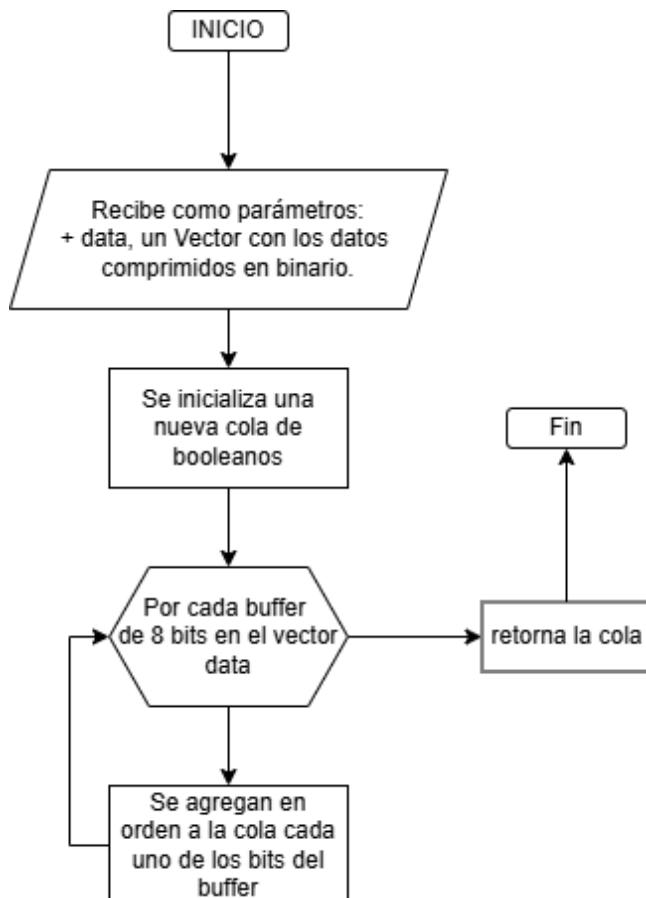


Fig. 32: Diagrama de flujo método `get_raw`

8.9.3 leer_pixel_data

- **Entradas**

- Un puntero `n` a un nodo de tipo `NodoHuffman`.
- Un puntero a una cola de bits `q`.

- **Salidas**

- Retorna el valor `c` almacenado en el nodo hoja al que se llegó.

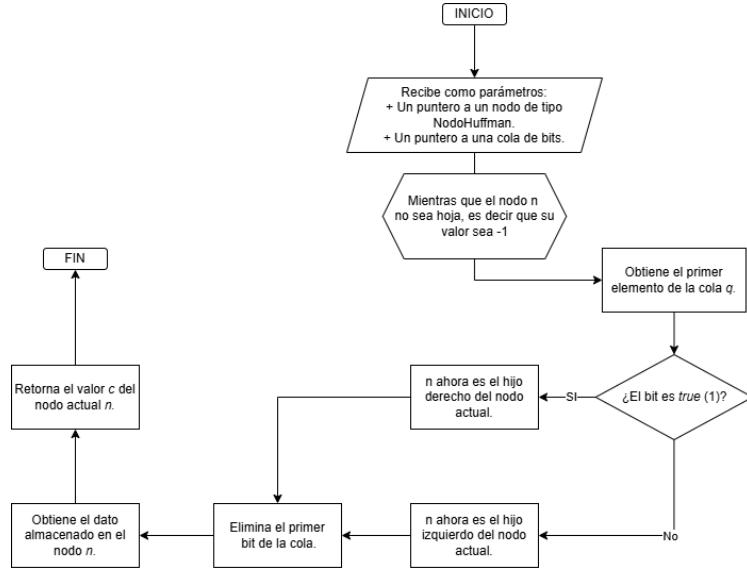


Fig. 33: Diagrama de flujo método leer_pixel_data

8.9.4 decode

- Entradas

- Ancho y alto de la imagen.
- data: Vector con los datos comprimidos en binario.

- Salidas

- contenido: Vector bidimensional con los valores de los píxeles decodificados.

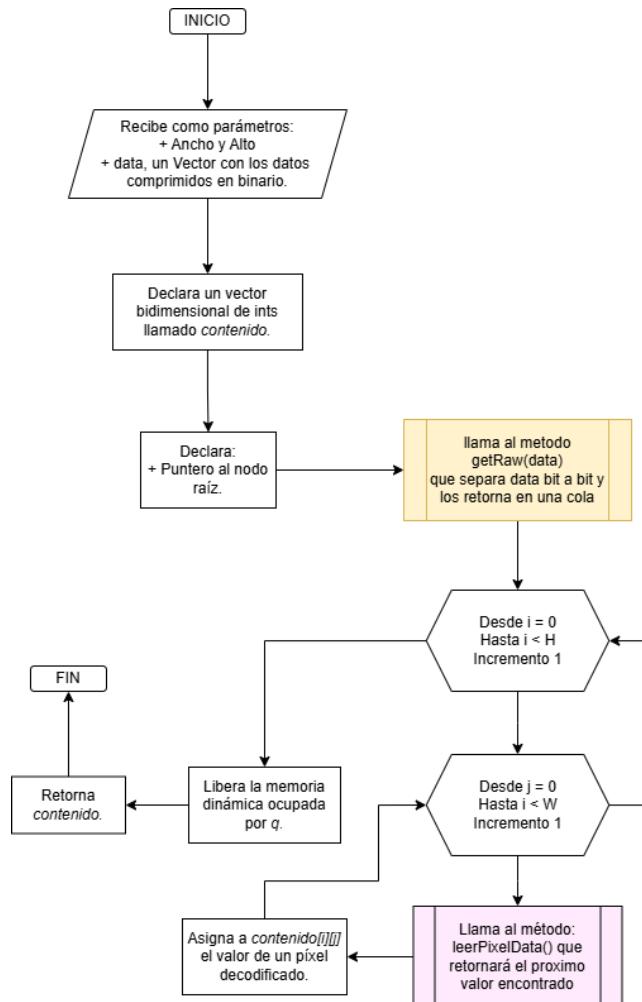


Fig. 34: Diagrama de flujo método decode

8.9.5 guardar_imagen_pgm

- Entradas

- Recibe como parámetros:
 - * Un puntero a un objeto de tipo Imagen.
 - * La ruta del archivo a generar.

- Salidas

- Se genera un archivo en formato P2 (.pgm) con el contenido de la imagen.

- Condiciones

- Si el archivo no se puede abrir correctamente, se lanza una excepción de tipo `runtime_error`.

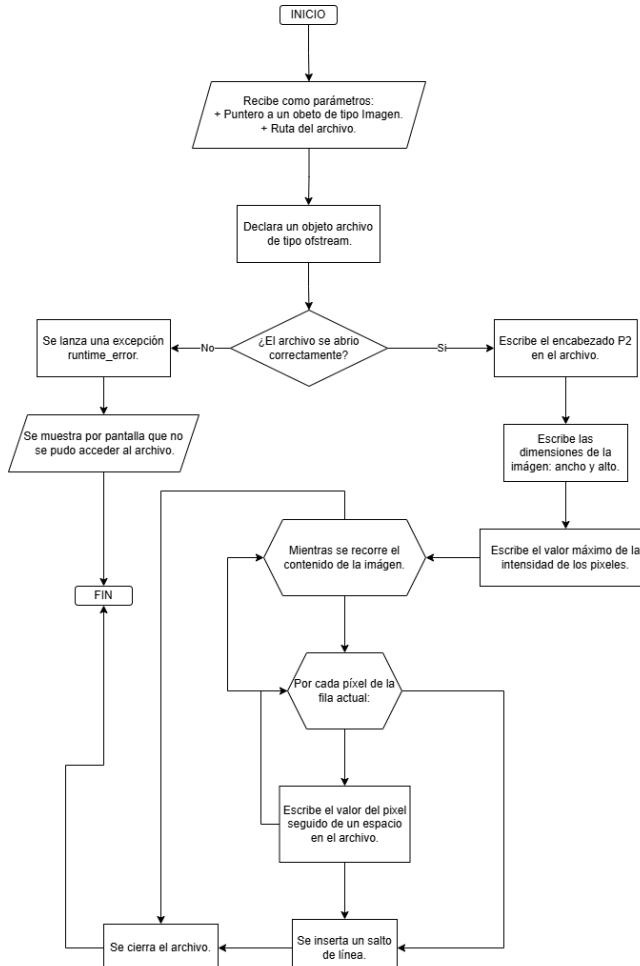


Fig. 35: Diagrama de flujo método guardar_imagen_pgm

8.10 segmentar

Este comando segmenta una imagen previamente cargada en memoria utilizando hasta cinco semillas. Cada semilla define una región en la imagen, la cual se forma agrupando píxeles con intensidades de color similares. Para ello, se emplea el algoritmo de Dijkstra, que se ejecuta de manera independiente para cada semilla. El algoritmo compara la intensidad de cada píxel con la de sus vecinos, expandiendo las regiones en función de la similitud. Al finalizar, cada píxel se asigna a la región cuya semilla resulte

más cercana, en caso de empate, se elige la primer semilla enviada por el usuario. El resultado se guarda en una nueva imagen que representa la segmentación en regiones de la imagen original.

- **Entradas**

- Vector de strings (**argv**), incluyendo ruta de salida y coordenadas de semillas.
- Referencia a un objeto de tipo **Memoria**.
- Imagen actualmente cargada en memoria (puntero a imagen).

- **Salidas**

- Mensaje en pantalla: “No hay una imagen cargada en memoria.” si el puntero es **nullptr**.
- Mensajes de validación por cada semilla leída correctamente.
- Mensaje: “Semillas leídas correctamente.” si todas las semillas se procesan sin error.
- Imagen segmentada guardada en formato PGM usando la ruta dada.
- Mensaje: “Se ha terminado de segmentar la imagen y se ha guardado en [ruta].”
- En caso de error, mensaje con la descripción del problema.

- **Condiciones**

- Si **img == nullptr**, se imprime mensaje de error y se retorna 0.
- Se procesan hasta 5 semillas a partir de los argumentos (cada una con tag, x, y).
- Si alguna coordenada o tag no es un número válido, se lanza excepción.
- Si las coordenadas están fuera de los límites de la imagen, o si el tag no está entre 1 y 255, se lanza excepción.
- Si ocurre una excepción durante el proceso, se muestra “Error: [mensaje]” y se retorna 1.
- Si todo el proceso se realiza correctamente, se retorna 0.

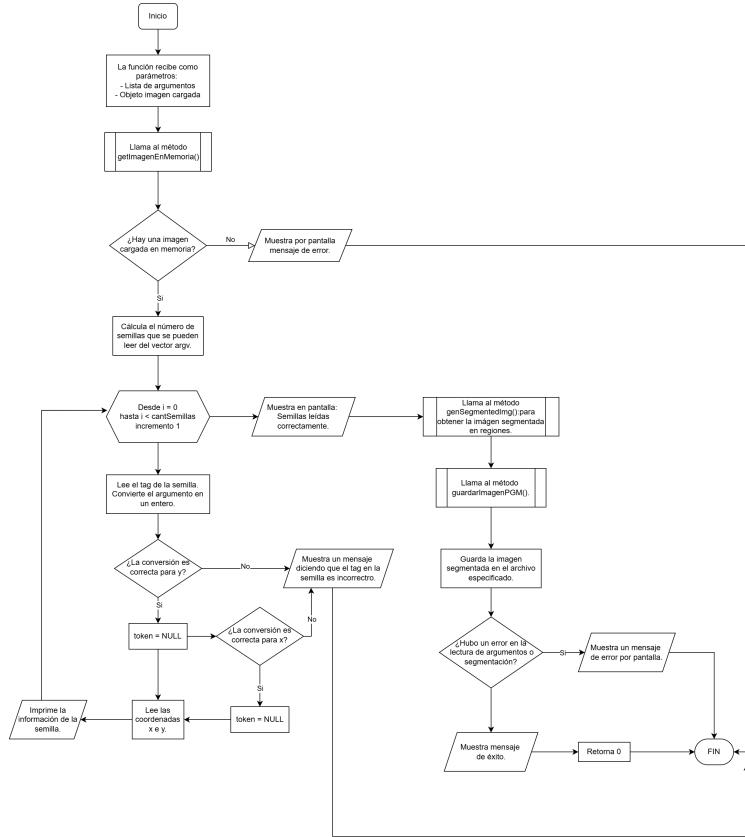


Fig. 36: Diagrama de flujo comando segmentar

8.10.1 processNeighborPixel

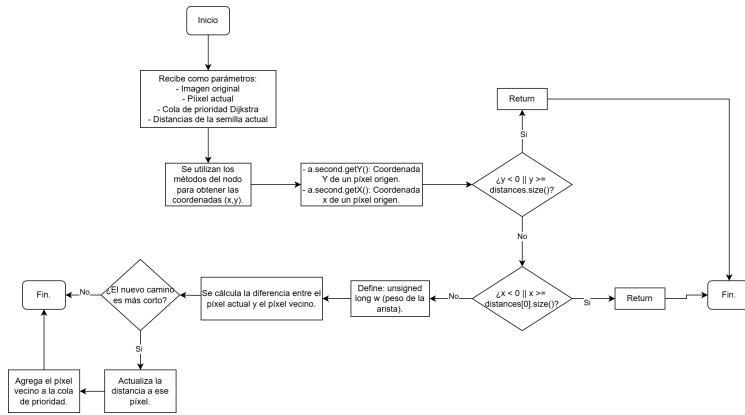


Fig. 37: Diagrama de flujo método processNeighborPixel

8.10.2 genSegmentedImg

Este código implementa el algoritmo de segmentación de imágenes usando Dijkstra, donde a partir de múltiples semillas se propagan etiquetas por los píxeles vecinos según su diferencia de intensidad, asignando a cada píxel la etiqueta de la semilla más cercana (en términos de menor costo de intensidad acumulada); para ello se usa una cola de prioridad que siempre tiene de primeras el siguiente píxel de menor distancia, actualizando una matriz de distancias finales y otra con las etiquetas asignadas, hasta construir y retornar una nueva imagen segmentada.

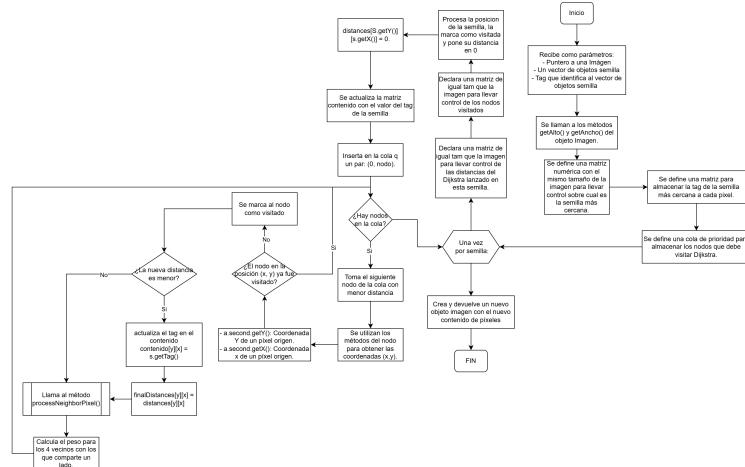


Fig. 38: Diagrama de flujo método genSegmentedImg

9 Plan de pruebas

9.1 Plan de pruebas general

Mirar las tablas 1, 2, 3 y 4.

9.2 Plan de pruebas Proyección 2D

Las siguientes pruebas se realizarán sobre el siguiente volumen:

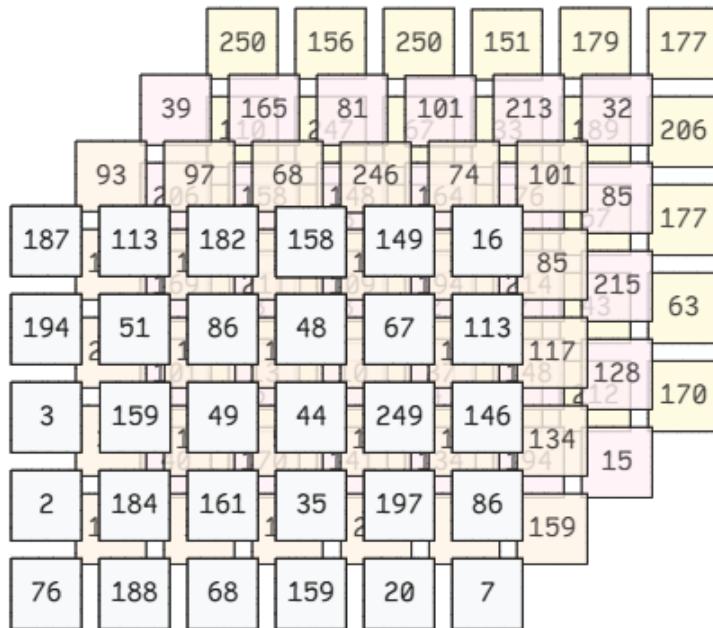


Fig. 39: Volumen plan de pruebas

Listing 3: generated01.pgm

```
P2
6 5
250
187 113 182 158 149 16
194 51 86 48 67 113
3 159 49 44 249 146
2 184 161 35 197 86
76 188 68 159 20 7
```

Listing 4: generated02.pgm

```
P2  
6 5  
250  
93 97 68 246 74 101  
107 164 36 126 16 85  
220 122 180 17 121 117  
37 192 57 100 166 134  
179 16 144 200 31 159
```

Listing 5: generated03.pgm

```
P2  
6 5  
250  
39 165 81 101 213 32  
206 158 148 164 76 85  
169 211 109 194 214 215  
101 13 10 37 148 128  
40 170 141 134 194 15
```

Listing 6: generated04.pgm

```
P2  
6 5  
250  
250 156 250 151 179 177  
110 247 67 33 189 206 26 178 5 63 57 177  
118 138 172 247 43 63  
136 9 224 76 212 170
```

Mirar las tablas [5](#), [6](#), [7](#) y [8](#).

9.3 Plan de pruebas codificación y decodificación

Para verificar el correcto funcionamiento de los comandos codificar_imagen y decodificar_imagen, hemos definido un set de pruebas con 7 imágenes, estas 7 imágenes se usarán para evaluar la codificación. Aún así, estas mismas 7 imágenes (ya codificadas y revisadas) se usarán para revisar la decodificación. Además de este pequeño set de pruebas, hemos creado un programa encargado de generar imágenes aleatorias y comprobar el funcionamiento del proyecto, la explicación y uso de este programa se mostrará al finalizar el set de pruebas.

En el caso de la codificación, revisaremos el archivo binario generado, el cual debe tener la siguiente estructura (según el documento):

Listing 7: Formato Archivo binario

W H M F0 F1 ... FM bits ...

Haciendo uso de la herramienta hex editor de visual studio code, revisaremos la estructura y el contenido de este archivo binario, para esto debemos tener en cuenta

el tamaño de los tipos de dato que estamos almacenando. Ya que los únicos datos con longitud constante que estamos almacenando son W, H, M y cada una de las frecuencias, y sus tamaños son respectivamente *unsigned short*, *unsigned short*, *unsigned char* y *unsigned long* (para cada una de las frecuencias).

9.3.1 Test 01

Este test busca codificar y decodificar una imagen 5x5 rellena de solo el número 0:

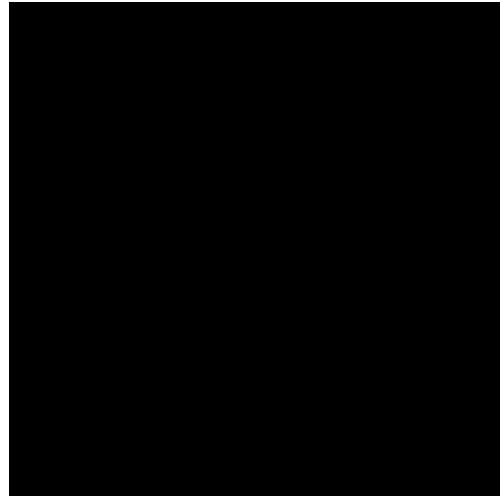


Fig. 40: Test 1

Ya que todos los valores están en 0, el contenido del archivo binario generado debería ser el siguiente:

Listing 8: Binario Test 01

```
5 5 0 25 [contenido]
```

Luego de la ejecución del proyecto cargando, codificando y decodificando esta imagen, revisamos manualmente los archivos generados, incluido el binario, haciendo uso de la herramienta hex editor.

9.3.2 Test 02

Este test busca codificar y decodificar una imagen 5x5 rellena de solo el número 255:

Fig. 41: Test 2

Ya que todos los valores estan en 255, el contenido del archivo binario generado debería ser el siguiente:

Listing 9: Binario Test 02

```
5 5 255 [255 ceros] 25 [contenido]
```

Luego de la ejecución del proyecto cargando, codificando y decodificando esta imagen, revisamos manualmente los archivos generados, incluido el binario, haciendo uso de la herramienta hex editor.

9.3.3 Test 03

Este test busca códificar y decodificar la siguiente imagen 5x5:



Fig. 42: Test 3

Ya que el valor maximo de la imagen es 255, el contenido del archivo binario generado debería ser el siguiente:

Listing 10: Binario Test 03

```
5 5 255 10 [254 ceros] 15 [contenido]
```

Luego de la ejecución del proyecto cargando, codificando y decodificando esta imagen, revisamos manualmente los archivos generados, incluido el binario, haciendo uso de la herramienta hex editor.

9.3.4 Test 04

Este test busca codificar y decodificar la siguiente imagen 5x5:

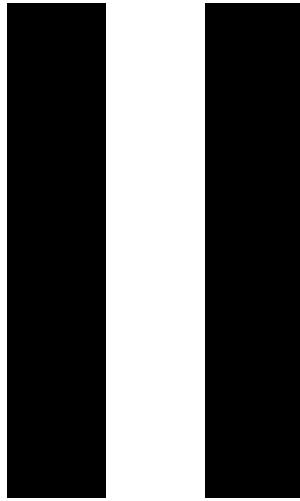


Fig. 43: Test 4

Ya que el valor maximo de la imagen es 255, el contenido del archivo binario generado debería ser el siguiente:

Listing 11: Binario Test 04

```
5 5 255 15 [254 ceros] 10 [contenido]
```

Luego de la ejecución del proyecto cargando, codificando y decodificando esta imagen, revisamos manualmente los archivos generados, incluido el binario, haciendo uso de la herramienta hex editor.

9.3.5 Test 05

Este test busca codificar y decodificar la siguiente imagen 5x5:



Fig. 44: Test 5

Ya que el valor maximo de la imagen es 255, el contenido del archivo binario generado debería ser el siguiente:

Listing 12: Binario Test 05

```
5 5 255 24 [254 ceros] 1 [contenido]
```

Luego de la ejecución del proyecto cargando, codificando y decodificando esta imagen, revisamos manualmente los archivos generados, incluido el binario, haciendo uso de la herramienta hex editor.

9.3.6 Test 06

Este test busca codificar y decodificar la siguiente imagen 5x5:

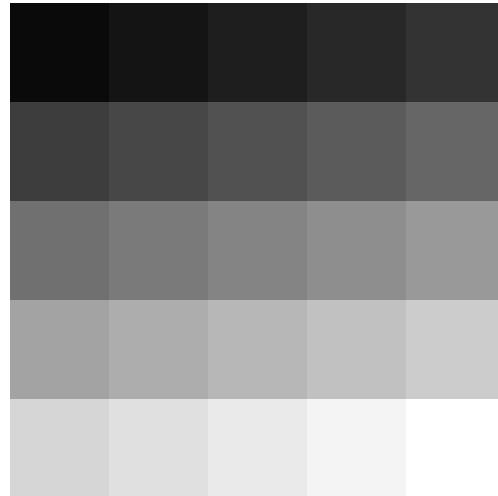


Fig. 45: Test 6

Ya que el valor maximo de la imagen es 25, el contenido del archivo binario generado debería ser el siguiente:

Listing 13: Binario Test 06

```
5 5 25 [25 unos] [contenido]
```

Luego de la ejecución del proyecto cargando, codificando y decodificando esta imagen, revisamos manualmente los archivos generados, incluido el binario, haciendo uso de la herramienta hex editor.

9.3.7 Test 07

Este test busca codificar y decodificar la siguiente imagen 5712x4284, la cual fue tomada por un telefono apple, posteriormente fue convertida a formato PGM (versión 5) y luego se uso un programa para convertir a PGM (versión 2, la usada en el proyecto):



Fig. 46: Test 7

Ya que el valor maximo de la imagen es 255, el contenido del archivo binario generado debería ser el siguiente:

Listing 14: Binario Test 07

```
5 5 255 [repeticiones] [contenido]
```

Este archivo es demasiado grande para ser comparado a mano, por lo que se desarrollo un programa que valida la información decodificada y la compara con la imagen original. El programa paso satisfactoriamente esta prueba, solo luego de solucionar el siguiente problema:

Al escribir cada uno de los buffers en algunos casos el ultimo buffer no se escribia correctamente, ya que los ceros que deberían estar al final, ya que no representan nada, estaban al inicio. Este problema se resolvio haciendo los shifts necesarios para dejar el contenido al inicio y el extra al final.

La imagen al revisarse nuevamente, muestra nuevamente la imagen completa y es totalmente igual a la imagen inicial.

9.3.8 Programa para revisión automatizada

Se creo un programa en c++ que genera imagenes aleatorias en base a los parametros enviados por el usuario, este programa aleatoriza las dimensiones, el pixel maximo y cada uno de los valores de la imagen, guarda la imagen generada como *in.pgm*, posteriormente se ejecuta el proyecto generando *coded.huf* y *out.pgm* usando el siguiente script:

Listing 15: Script de prueba

```
const string sScript = executablePath + "->-/dev/null-2>&1-<<- 'EOF'\n" +
    "cargar_imagen -in.pgm\n"
    "codificar_imagen -coded.huf\n"
    "decodificar_archivo -coded.huf -out.pgm\n"
    "salir\n" +
    "EOF";

const char* script = sScript.c_str();
```

Luego esta imagen es revisada valor por valor y en caso de un error se mostrará al usuario indicando en donde se encontro el error.

Con este programa se realizaron distintas pruebas, para este documento solo se considera necesaria una prueba de 50 imagenes donde la dimensión maxima puede ser 2000:

```
acha@acha:/mnt/c/Users/vmigu/dev/estructuras-20253/scripts$ g++ testHuffman.cpp && ./a.out ../proyecto.out 50 2000
Imagen 1 1393x1109 (226) pixeles = 1544837 Correcta
Imagen 2 1239x201 (41) pixeles = 249639 Correcta
Imagen 3 171x22 (4) pixeles = 3762 Correcta
Imagen 4 1210x1193 (32) pixeles = 1443530 Correcta
Imagen 5 314x483 (99) pixeles = 151662 Correcta
Imagen 6 445x228 (151) pixeles = 101460 Correcta
Imagen 7 1873x391 (4) pixeles = 732343 Correcta
Imagen 8 1832x1965 (112) pixeles = 3599880 Correcta
Imagen 9 1492x716 (140) pixeles = 1068272 Correcta
Imagen 10 497x93 (138) pixeles = 46221 Correcta
Imagen 11 921x889 (199) pixeles = 818769 Correcta
Imagen 12 631x665 (83) pixeles = 419615 Correcta
Imagen 13 216x1086 (92) pixeles = 234576 Correcta
Imagen 14 1482x917 (144) pixeles = 1285634 Correcta
Imagen 15 1449x1588 (170) pixeles = 2301012 Correcta
Imagen 16 1499x436 (40) pixeles = 658061 Correcta
Imagen 17 578x1355 (184) pixeles = 783190 Correcta
Imagen 18 164x247 (77) pixeles = 40588 Correcta
Imagen 19 1447x1567 (242) pixeles = 2267449 Correcta
Imagen 20 665x651 (238) pixeles = 432915 Correcta
Imagen 21 1617x1972 (127) pixeles = 3188724 Correcta
Imagen 22 1768x219 (39) pixeles = 387192 Correcta
Imagen 23 1740x1970 (216) pixeles = 3427800 Correcta
Imagen 24 1183x828 (46) pixeles = 970060 Correcta
Imagen 25 1946x1176 (18) pixeles = 2276820 Correcta
Imagen 26 406x1121 (136) pixeles = 455126 Correcta
Imagen 27 410x1173 (236) pixeles = 480930 Correcta
Imagen 28 1265x695 (6) pixeles = 879175 Correcta
Imagen 29 53x1135 (18) pixeles = 60155 Correcta
Imagen 30 446x1910 (224) pixeles = 851860 Correcta
Imagen 31 1473x1934 (195) pixeles = 2848782 Correcta
Imagen 32 544x272 (6) pixeles = 147968 Correcta
Imagen 33 872x110 (247) pixeles = 95920 Correcta
Imagen 34 79x525 (245) pixeles = 41475 Correcta
Imagen 35 1819x1215 (147) pixeles = 2210085 Correcta
Imagen 36 321x876 (31) pixeles = 281196 Correcta
Imagen 37 144x1463 (176) pixeles = 210672 Correcta
Imagen 38 901x300 (281) pixeles = 278300 Correcta
Imagen 39 919x497 (2) pixeles = 456743 Correcta
Imagen 40 1677x1490 (129) pixeles = 2498730 Correcta
Imagen 41 1723x171 (139) pixeles = 294633 Correcta
Imagen 42 1291x990 (192) pixeles = 1278090 Correcta
Imagen 43 52x123 (183) pixeles = 6396 Correcta
Imagen 44 1756x1483 (192) pixeles = 2604148 Correcta
Imagen 45 1408x503 (61) pixeles = 708224 Correcta
Imagen 46 1668x1563 (198) pixeles = 2607084 Correcta
Imagen 47 1762x113 (238) pixeles = 199166 Correcta
Imagen 48 1289x1083 (61) pixeles = 1395987 Correcta
Imagen 49 1395x1600 (36) pixeles = 2232000 Correcta
Imagen 50 29x1511 (224) pixeles = 43819 Correcta
CORRECTO!
```

Fig. 47: Prueba de ejecución automatizada

9.4 Plan de pruebas segmentar

Para verificar el correcto funcionamiento del comando segmentar, hemos definido un set de pruebas con 8 imágenes, estas 8 imágenes se usarán para evaluar la segmentación. Cada una de las siguientes imágenes fue probadas manualmente y revisadas posteriormente usando herramientas de debug que se implementaron en el desarrollo.

Comenzaremos con las pruebas de entrada de datos, es necesario validar que los datos que proporciona el usuario son válidos, para estas pruebas usaremos el test-1, el cual es una imagen en blanco de 5 pixeles de ancho por 5 pixeles de alto con una intensidad máxima de 200, (Vease [53](#)).

9.4.1 Pruebas Entrada y salida de datos

- **Imagen cargada en memoria** Debemos verificar si hay una imagen cargada o no en memoria:

```
acha@acha:/mnt/c/Users/vmigu/dev/estructuras-20251$ ./proyecto.out
$ segmentar test.pgm 0 0 1
No hay una imagen cargada en memoria
$ cargar_imagen test-1.pgm
Se ha finalizado la carga del archivo test-1.pgm
$ segmentar test.pgm 0 0 1
Semilla 1 - tag: 1 y: 0 x: 0
Semillas leidas correctamente
Se ha terminado de segmentar la imagen y se ha guardado en test.pgm
$
```

Fig. 48: Test de Entrada 1

- **Tag** Debemos verificar que el tag de cada semilla sea válido:

```
acha@acha:/mnt/c/Users/vmigu/dev/estructuras-20251$ ./proyecto.out
$ cargar_imagen test-1.pgm
Se ha finalizado la carga del archivo test-1.pgm
$ segmentar test.pgm 0 0 300
Semilla 1 - tag: 300 y: 0 x: 0
Error: La semilla tiene un tag invalido (0-255)
$ segmentar test.pgm 0 0 -24
Semilla 1 - tag: -24 y: 0 x: 0
Error: La semilla tiene un tag invalido (0-255)
$ segmentar test.pgm 0 0 5
Semilla 1 - tag: 5 y: 0 x: 0
Semillas leidas correctamente
Se ha terminado de segmentar la imagen y se ha guardado en test.pgm
$
```

Fig. 49: Test de Entrada 2

- **Coordenadas** Debemos verificar que cada coordenada de las semillas sea válida

```

acha@acha:/mnt/c/Users/vmigu/dev/estructuras-20251$ ./proyecto.out
$ cargar_imagen test-1.pgm
Se ha finalizado la carga del archivo test-1.pgm
$ segmentar test.pgm -50 0 5
Semilla 1 - tag: 5 y: 0 x: -50
Error: La semilla tiene unas coordenadas invalidas
$ segmentar test.pgm 0 50 5
Semilla 1 - tag: 5 y: 50 x: 0
Error: La semilla tiene unas coordenadas invalidas
$ segmentar test.pgm 4 4 4
Semilla 1 - tag: 4 y: 4 x: 4
Semillas leidas correctamente
Se ha terminado de segmentar la imagen y se ha guardado en test.pgm
$ █

```

Fig. 50: Test de entrada 3

- **Multiples semillas** Debemos verificar que se lean correctamente multiples semillas:

```

acha@acha:/mnt/c/Users/vmigu/dev/estructuras-20251$ ./proyecto.out
$ cargar_imagen test-1.pgm
Se ha finalizado la carga del archivo test-1.pgm
$ segmentar test.pgm 0 0 1 1 1 2 2 2 3 3 3 4 4 4 5
Semilla 1 - tag: 1 y: 0 x: 0
Semilla 2 - tag: 2 y: 1 x: 1
Semilla 3 - tag: 3 y: 2 x: 2
Semilla 4 - tag: 4 y: 3 x: 3
Semilla 5 - tag: 5 y: 4 x: 4
Semillas leidas correctamente
Se ha terminado de segmentar la imagen y se ha guardado en test.pgm
$ █

```

Fig. 51: Test de entrada 4

- **Nueva intensidad máxima correcta** Debemos verificar que la nueva intensidad es correcta:

cargar_imagen test-1.pgm segmentar salida-1.pgm 0 0 255

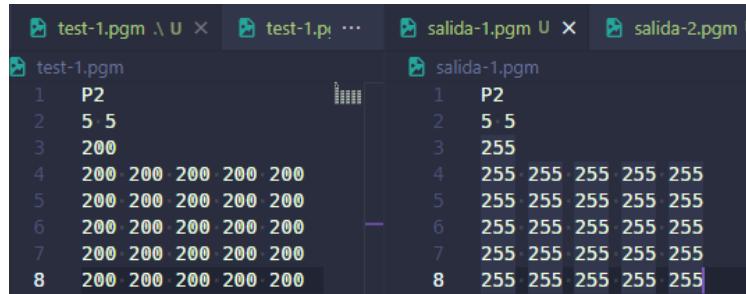


Fig. 52: Test de entrada 5

9.5 Test 01

Este test busca segmentar una imagen totalmente en blanco de tamaño 5x5 usando 1 semilla en la posición (0,0) con el tag 50.

```
cargar_imagen test-1.pgm segmentar salida-1.pgm 0 0 20
```

Fig. 53: Test Segmentar 1

El resultado fue el siguiente, cabe aclarar que el resultado es visualmente el mismo, esto se debe a que gimp toma la intensidad máxima como blanco, ya que la intensidad máxima ahora es 20 y todo está lleno de 20, todo será blanco:



Fig. 54: Test Segmentar 1 - Resultado

Luego de la ejecución del proyecto cargando y segmentando esta imagen, revisamos manualmente los archivos generados, y verificamos cada matriz de distancias de cada semilla.

9.6 Test 02

Este test busca segmentar una imagen con un cuadrado 3x3 dentro, la imagen es de tamaño 5x5 y se usaron 2 semillas, una en una esquina para intentar segmentar el fondo y otra en el centro del cuadrado para intentar segmentar el cuadrado.

```
cargar_imagen test-2.pgm segmentar salida-2.pgm 0 0 50 2 2 200
```

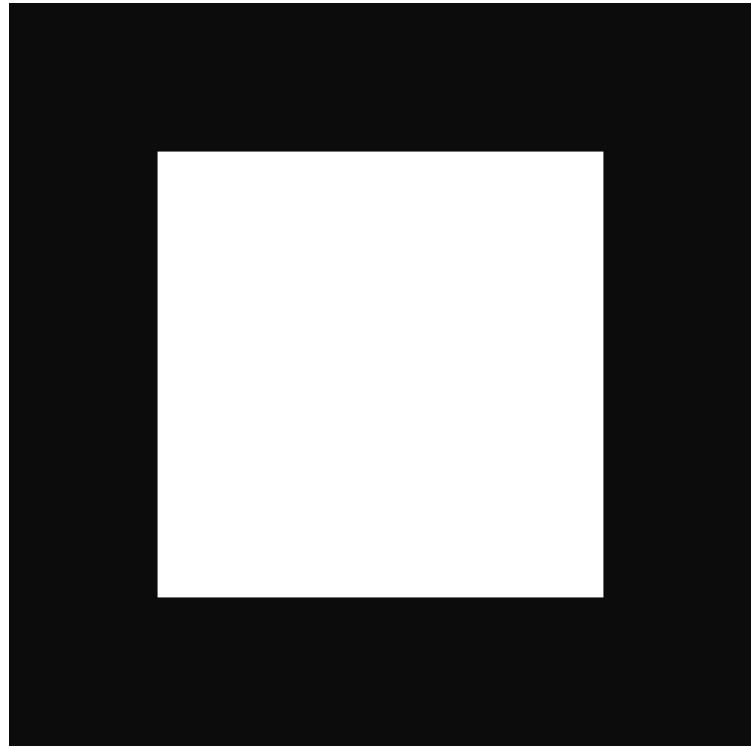


Fig. 55: Test Segmentar 2

El resultado fue el siguiente:

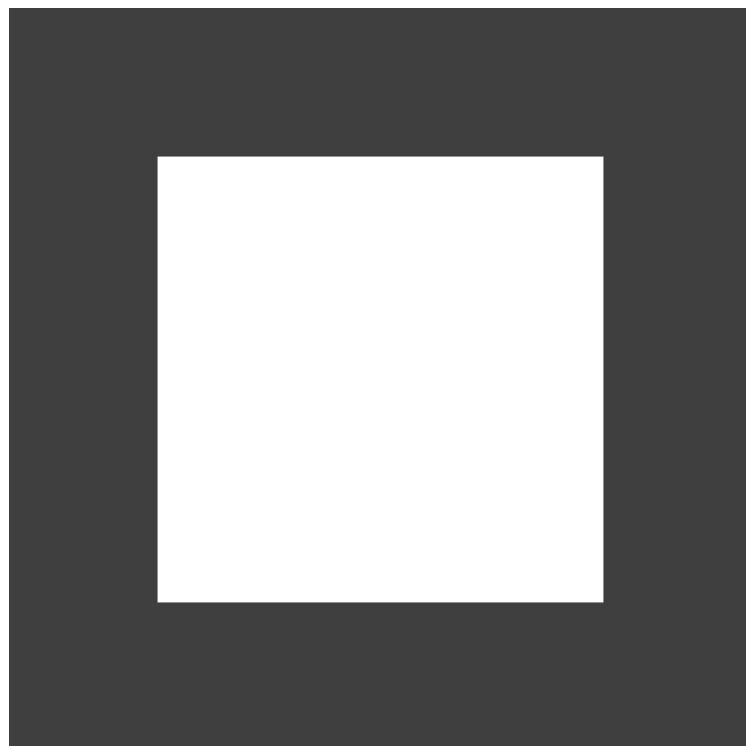


Fig. 56: Test Segmentar 2 - Resultado

Luego de la ejecución del proyecto cargando y segmentando esta imagen, revisamos manualmente los archivos generados, y verificamos cada matriz de distancias de cada semilla.

9.7 Test 03

Este test busca segmentar una imagen que se compone de una serie de 2 cuadrados super puestos, pero con intensidades irregulares de tamaño 10x10 usando 3 semillas, una para segmentar el fondo, y las otras dos para cada cuadrado respectivamente.

```
cargar_imagen test-3.pgm segmentar salida-3.pgm 0 0 10 1 1 20 3 3 60
```

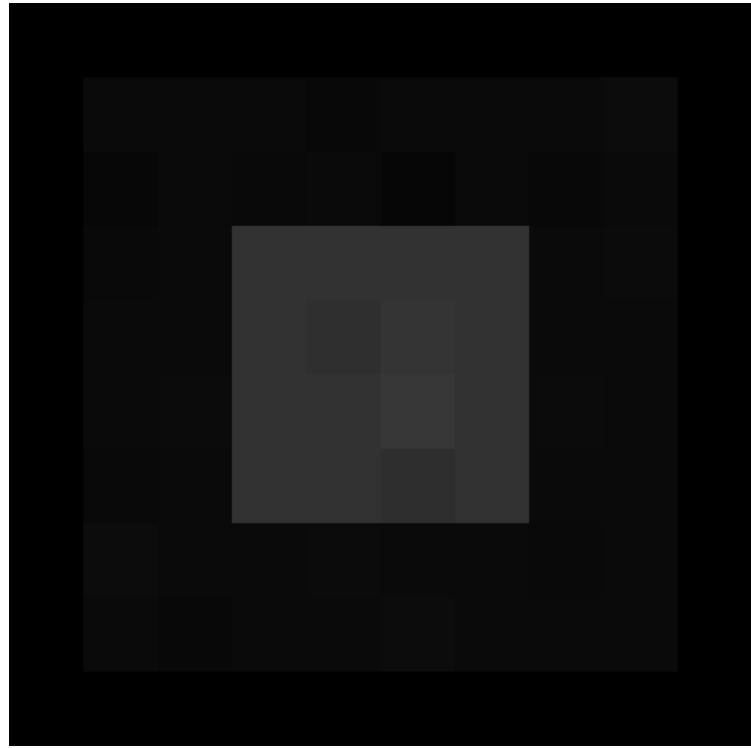


Fig. 57: Test Segmentar 3

El resultado fue el siguiente:

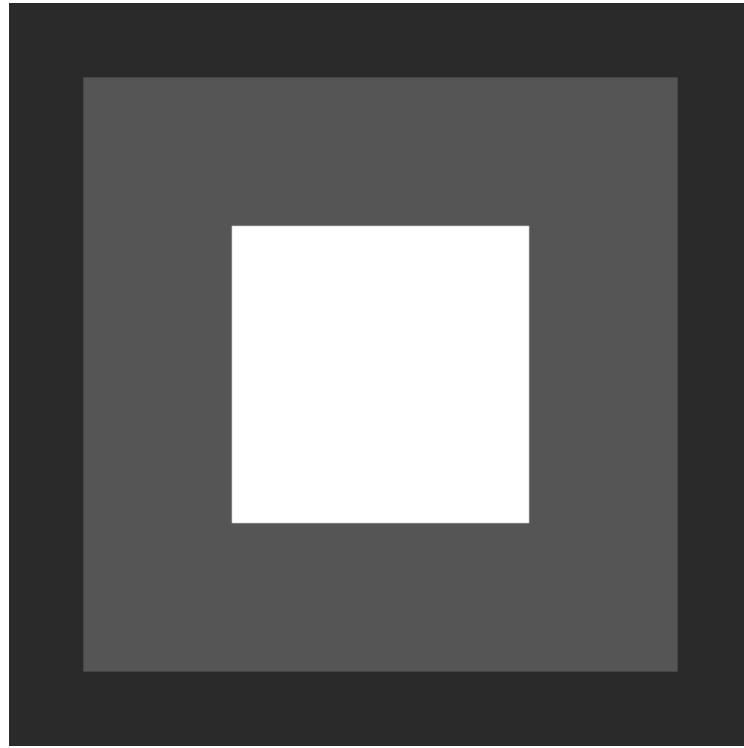


Fig. 58: Test Segmentar 3 - Resultado

Luego de la ejecución del proyecto cargando y segmentando esta imagen, revisamos manualmente los archivos generados, y verificamos cada matriz de distancias de cada semilla.

9.8 Test 04

Este test busca segmentar una imagen de dos rectangulos separados por una linea de intensidad diferente de tamaño 5x5 usando 3 semillas. Lo interesante de este test es ver que ocurre en los empates, se asume que se debe dar prioridad a la primer semilla enviada por el usuario.

```
cargar_imagen test-4.pgm segmentar salida-4.pgm 0 0 10 4 0 20
```



Fig. 59: Test Segmentar 4

El resultado fue el siguiente, podemos ver como el lado izquierdo es más grande que el derecho, con esto confirmamos que estamos dando prioridad a la primer semilla enviada por el usuario (0,0):

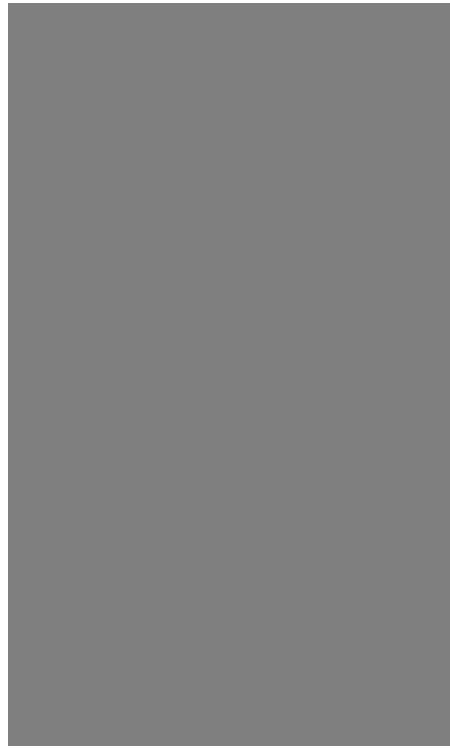


Fig. 60: Test Segmentar 4 - Resultado

Luego de la ejecución del proyecto cargando y segmentando esta imagen, revisamos manualmente los archivos generados, y verificamos cada matriz de distancias de cada semilla.

9.9 Test 05

Este test busca segmentar una imagen de un fondo de 4 cuadrados de igual tamaño con un cuadrado en el centro de intensidad diferente, la imagen tiene tamaño 10x10 y se usaron 5 semillas.

```
cargar_imagen test-5.pgm segmentar salida-5.pgm 0 0 10 9 0 20 0 9 30 9 9 40 4 4 50
```



Fig. 61: Test Segmentar 5

El resultado fue el siguiente:

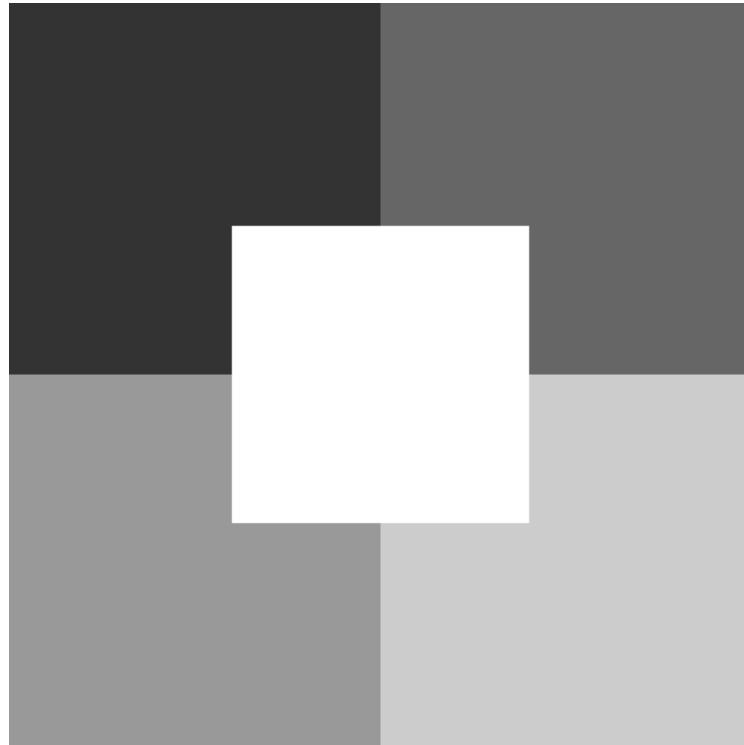


Fig. 62: Test Segmentar 5 - Resultado

Luego de la ejecución del proyecto cargando y segmentando esta imagen, revisamos manualmente los archivos generados, y verificamos cada matriz de distancias de cada semilla.

9.10 Test 06

Este test busca segmentar una imagen con distintas figuras geométricas de tamaño 100x100 usando 4 semillas, una para el fondo y las demás para cada figura.

```
cargar_imagen test-6.pgm segmentar salida-6.pgm 0 0 50 20 20 100 20 70 150 80  
60 200
```

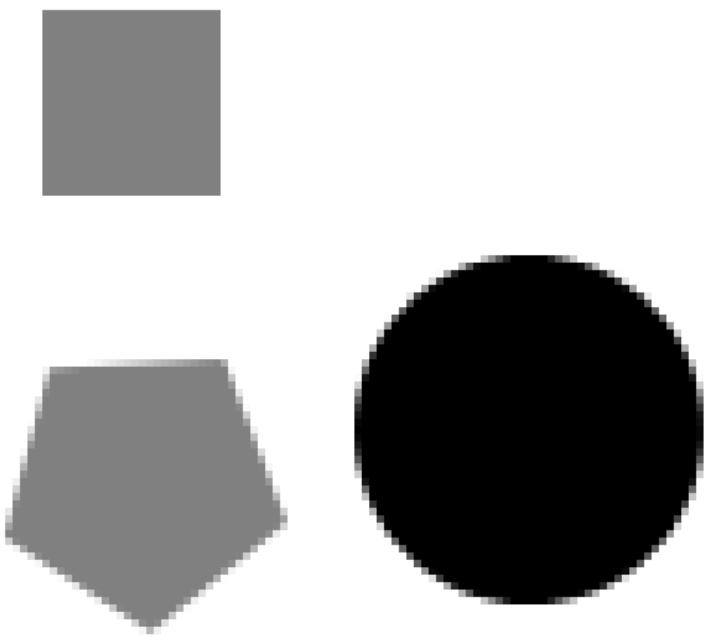


Fig. 63: Test Segmentar 6

El resultado fue el siguiente:

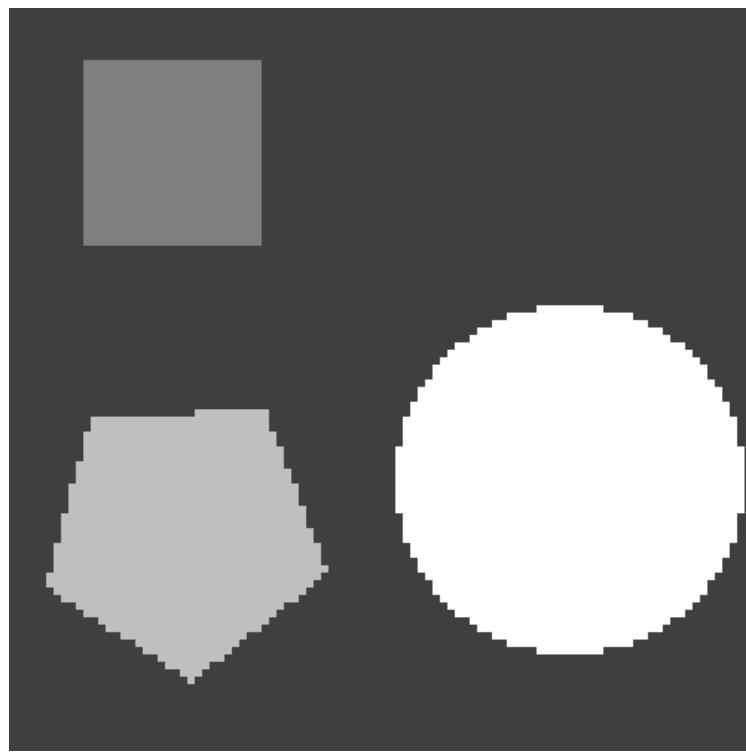


Fig. 64: Test Segmentar 6 - Resultado

Luego de la ejecución del proyecto cargando y segmentando esta imagen, revisamos manualmente los archivos generados, y verificamos cada matriz de distancias de cada semilla.

9.11 Test 07

Este test busca segmentar una imagen de un dibujo de la cara de un gato de tamaño 66x54 usando 3 semillas, una para el fondo, otra para la cabeza del gato y otra para la boca.

```
cargar_imagen test-7.pgm segmentar salida-7.pgm 0 0 50 33 25 100 32 38 150
```



Fig. 65: Test Segmentar 7

El resultado fue el siguiente:



Fig. 66: Test Segmentar 7 - Resultado

Luego de la ejecución del proyecto cargando y segmentando esta imagen, revisamos manualmente los archivos generados, y verificamos cada matriz de distancias de cada semilla.

9.12 Test 08

```
cargar_imagen test-8.pgm segmentar salida-8.pgm 0 0 1 67 82 255
```

Este test busca segmentar una imagen convertida a PGM de un gato sobre un fondo blanco de tamaño 148x149 usando 2 semillas, una para el fondo y otra para el gato.



Fig. 67: Test Segmentar 8

El resultado fue el siguiente:



Fig. 68: Test Segmentar 8 - Resultado

References

- [1] Huffman, D.A.: A method for the construction of minimum-redundancy codes. Proceedings of the IRE **40**(9), 1098–1101 (1952) <https://doi.org/10.1109/JRPROC.1952.273898>

PLAN DE PRUEBAS GENERAL 1			
Descripción de caso	Valores de entrada	Resultado esperado	Resultado obtenido
Carga de imagen válida	<code>cargar_imagen imagen1.pgm</code> (Archivo PGM válido)	Confirmación de carga exitosa	Se ha finalizado la carga del archivo imagen1.pgm
Carga de imagen válida con todo su contenido en una sola linea	<code>cargar_imagen imagen1.pgm</code> (Archivo PGM válido, contenido en una linea)	Confirmación de carga exitosa	Se ha finalizado la carga del archivo imagen1.pgm
Carga de imagen inexistente	<code>cargar_imagen inexistente.pgm</code> (Archivo no existe)	Error: Archivo no encontrado	ERROR inexistente.pgm: El archivo inexistente.pgm no se pudo abrir
Carga de imagen con formato inválido	<code>cargar_imagen archivo.txt</code> (Archivo no es PGM)	Error: Formato inválido	ERROR archivo.txt: La imagen esta en un formato invalido, no comienza con P2
Carga de imagen con tamaño invalido	<code>cargar_imagen imagen1.pgm</code> (Altura es negativo)	Error: Formato inválido	ERROR imagen1.pgm: La imagen esta en un formato invalido, alto es negativo
Carga de imagen con pixeles negativos	<code>cargar_imagen imagen1.pgm</code> (Pixeles negativos)	Error: Formato inválido	ERROR imagen1.pgm: La imagen esta en un formato invalido, ancho es negativo
Carga de imagen con pixel mayor a la intensidad máxima especificada	<code>cargar_imagen imagen1.pgm</code> (pixel mayor a intensidad maxima)	Error: Formato inválido	ERROR imagen1.pgm: Se intento leer un pixel con intensidad mayor a la maxima especificada por el archivo.
Carga de volumen válido	<code>cargar_volumen diag 3</code> (Archivos existen)	Confirmación de carga exitosa	Se ha finalizado la cargar del volumen diag
Carga de volumen con exceso de imágenes	<code>cargar_volumen diag 100</code> (Más de 99 imágenes)	Error: Excede el límite de imágenes	ERROR cargando volumen diag: Cantidad de imagenes fuera de rango (1-99)
Carga de volumen con archivos inexistentes	<code>cargar_volumen diag 5</code> (Los archivos no existen)	Error: No se encuentran archivos	ERROR cargando volumen diag: Error leyendo diag04.pgm: El archivo diag04.pgm no se pudo abrir

Table 1: Plan de pruebas general 1

PLAN DE PRUEBAS GENERAL 2			
Obtener información de imagen cargada	info_imagen (Imagen cargada previamente)	Nombre, ancho y alto de la imagen	Imagen cargada en memoria: diag01.pgm Ancho: 6 Alto: 5
Obtener información sin imagen cargada	info_imagen (No se ha ejecutado cargar_imagen)	Error: No hay imagen en memoria	No hay una imagen cargada en memoria
Obtener información sin volumen cargado	info_volumen (No se ha ejecutado cargar_volumen)	Error: No hay volumen en memoria	No hay un volumen cargado en memoria
Obtener información de volumen cargado	info_volumen (Volumen cargado previamente)	Nombre base, cantidad de imágenes, dimensiones	Volumen cargado en memoria: diag Tamano: 3 Ancho: 6 Alto: 5
Mostrar ayuda del sistema	ayuda (Comando sin parámetros)	Lista de comandos disponibles con su descripción	<pre>U1.0.8 Usa "ayuda" para ver esta lista Usa "ayuda comando" para encontrar mas informacion sobre "comando" cargar_imagen [nombre_imagen.pgm] cargar_volumen [nombre_base] [n_im] decodificar_imagen info_volumen procesar_imagen [direccion] [(tamaño)] [nombre_archivo.pgm] codificar_imagen [nombre_archivo.huf] decodificar_archivo [nombre_archivo.huf] [nombre_imagen.pgm] generar_imagen [imagen.pgm] [salida_imagen.pgm] [salida_imagen.huf] ... salir</pre>

Table 2: Plan de pruebas general 2

EVIDENCIA DE EJECUCIÓN - PRUEBAS GENERAL 1	
Descripción de caso	Evidencia de ejecución
Carga de imagen válida	\$ cargar_imagen imagen1.pgm Se ha finalizado la carga del archivo imagen1.pgm
Carga de imagen válida con todo su contenido en una sola linea	\$ cargar_imagen imagen1.pgm Se ha finalizado la carga del archivo imagen1.pgm
Carga de imagen inexistente	\$ cargar_imagen inexistente.pgm ERROR inexistente.pgm: El archivo inexistente.pgm no se pudo abrir
Carga de imagen con formato inválido	\$ cargar_imagen archivo.txt ERROR archivo.txt: La imagen esta en un formato invalido, no comienza con P2
Carga de imagen con tamaño invalido	\$ cargar_imagen imagen1.pgm ERROR imagen1.pgm: La imagen esta en un formato invalido, alto es negativo \$ █
Carga de imagen con pixeles negativos	\$ cargar_imagen imagen1.pgm ERROR imagen1.pgm: La imagen esta en un formato invalido, ancho es negativo \$ █
Carga de imagen con pixel mayor a la intensidad máxima especificada	\$ cargar_imagen imagen1.pgm ERROR imagen1.pgm: Se intento leer un pixel con intensidad mayor a la maxima especificada por el archivo. \$ █
Carga de volumen válido	\$ cargar_volumen diag 3 Leyendo la imagen diag01.pgm (1/3) Leyendo la imagen diag02.pgm (2/3) Leyendo la imagen diag03.pgm (3/3) Se ha finalizado la cargar del volumen diag \$ █
Carga de volumen con exceso de imágenes	\$ cargar_volumen diag 100 ERROR cargando volumen diag: Cantidad de imagenes fuera de rango (1-99) \$ █
Carga de volumen con archivos inexistentes	\$ cargar_volumen diag 5 Leyendo la imagen diag01.pgm (1/5) Leyendo la imagen diag02.pgm (2/5) Leyendo la imagen diag03.pgm (3/5) Leyendo la imagen diag04.pgm (4/5) ERROR cargando volumen diag: Error leyendo diag04.pgm: El archivo diag04.pgm no se pudo abrir \$ █

Table 3: Evidencia de ejecución - Plan de pruebas general 1

EVIDENCIA DE EJECUCIÓN - PRUEBAS GENERAL 2	
Descripción de caso	Evidencia de ejecución
Obtener información de imagen cargada	\$ cargar_imagen diag01.pgm Se ha finalizado la carga del archivo diag01.pgm \$ info_imagen Imagen cargada en memoria: diag01.pgm Ancho: 6 Alto: 5 \$ █
Obtener información sin imagen cargada	\$ info_imagen No hay una imagen cargada en memoria \$ █
Obtener información de volumen cargado	\$ info_volumen No hay un volumen cargado en memoria \$ █
Obtener información sin volumen cargado	\$ cargar_volumen diag 3 Leyendo la imagen diag01.pgm (1/3) Leyendo la imagen diag02.pgm (2/3) Leyendo la imagen diag03.pgm (3/3) Se ha finalizado la cargar del volumen diag \$ info_volumen Volumen cargado en memoria: diag Tamano: 3 Ancho: 6 Alto: 5 \$ █
Mostrar ayuda del sistema	\$ ayuda v1.0.0 Usa `ayuda` para ver esta lista Usa `ayuda comando` para encontrar mas informacion sobre `comando` cargar_imagen [nombre_imagen.pgm] cargar_volumen [nombre_base] [n_im] info_imagen info_volumen proyeccion2D [direccion] [criterio] [nombre_archivo.pgm] codificar_imagen [nombre_archivo.huf] decodificar_archivo [nombre_archivo.huf] [nombre_imagen.pgm] segmentar [imagen.pgm] [salida_imagen.pgm] [sx1 syl sl1] ... salir \$ █

Table 4: Evidencia de ejecución - Plan de pruebas general 2

PLAN DE PRUEBAS 2 PROYECCION 2D			
Descripción de caso	Valores de entrada	Resultado esperado	Resultado Obtenido
Proyección 2D con dirección inválida	proyeccion2D w minimo salida.pgm (Dirección "w" no válida)	Error: Dirección inválida	Error: Direccion Invalida
Proyección 2D con criterio invalido	proyeccion2D x aguacate salida.pgm (Criterio "aguacate" no válido)	Error: Criterio inválido	Error: Criterio Invalido
Proyección 2D sin volumen cargado	proyeccion2D x maximo salida.pgm (No se ha ejecutado car-gar_volumen)	Error: No hay volumen en memoria	No hay un volumen cargado en memoria
En eje X - Máximo	proyeccion2D x maximo xmax.pgm	250 247 250 247 212 206 206 211 148 194 214 215 220 192 180 246 166 159 194 188 182 159 249 146	250 247 250 247 212 206 206 211 148 194 214 215 220 192 180 246 166 159 194 188 182 159 249 146
En eje Y - Máximo	proyeccion2D y maximo ymax.pgm	250 213 246 187 247 206 164 194 178 215 220 249 247 148 192 197 224 194 200 188	250 213 246 187 247 206 164 194 178 215 220 249 247 148 192 197 224 194 200 188
En eje Z - Máximo	proyeccion2D z maximo zmax.pgm	250 165 250 246 213 177 206 247 148 164 189 206 220 211 180 194 249 215 118 192 172 247 197 134 179 188 224 200 212 170	250 165 250 246 213 177 206 247 148 164 189 206 220 211 180 194 249 215 118 192 172 247 197 134 179 188 224 200 212 170
En eje X - Mínimo	proyeccion2D x minimo xmin.pgm	26 9 5 33 43 63 39 13 10 37 76 15 37 16 36 17 16 85 2 51 49 35 20 7	26 9 5 33 43 63 39 13 10 37 76 15 37 16 36 17 16 85 2 51 49 35 20 7
En eje Y - Mínimo	proyeccion2D y minimo ymin.pgm	151 32 68 16 33 76 16 48 5 109 17 3 43 10 37 2 9 15 16 7	151 32 68 16 33 76 16 48 5 109 17 3 43 10 37 2 9 15 16 7
En eje Z - Mínimo	proyeccion2D z minimo ymin.pgm	39 97 68 101 74 16 107 51 36 33 16 85 3 122 5 17 57 117 2 13 10 35 43 63 40 9 68 76 20 7	39 97 68 101 74 16 107 51 36 33 16 85 3 122 5 17 57 117 2 13 10 35 43 63 40 9 68 76 20 7

Table 5: Plan de pruebas 2 para proyección 2D

PLAN DE PRUEBAS 2 PROYECCION 2D			
Descripción de caso	Valores de entrada	Resultado esperado	Resultado Obtenido
En eje X - Mediana	proyecction2D x mediana xmed.pgm	118 156 172 76 179 177 101 165 109 134 194 85 107 122 68 126 74 117 76 159 86 48 149 86	118 156 172 76 179 177 101 165 109 134 194 85 107 122 68 126 74 117 76 159 86 48 149 86
En eje Y - Mediana	proyecction2D y mediana ymed.pgm	178 91 95 153 149 153 96 76 60 202 121 97 128 69 117 123 153 137 151 72	178 91 95 153 149 153 96 76 60 202 121 97 128 69 117 123 153 137 151 72
En eje Z - Mediana	proyecction2D z mediana zmed.pgm	140 134 131 154 164 66 152 161 76 87 71 99 97 168 79 53 167 161 69 161 109 68 157 107 106 93 142 146 112 87	140 134 131 154 164 66 152 161 76 87 71 99 97 168 79 53 167 161 69 161 109 68 157 107 106 93 142 146 112 87
En eje X - Promedio	proyecction2D x promedio xprom.pgm	128 145 143 114 136 158 111 143 97 126 169 95 127 118 97 137 81 119 92 139 109 88 136 73	128 145 143 114 136 158 111 143 97 126 169 95 127 118 97 137 81 119 92 139 109 88 136 73
En eje Y - Promedio	proyecction2D y promedio yprom.pgm	193 105 113 134 142 139 89 93 84 185 129 108 130 72 114 110 137 115 121 86	193 105 113 134 142 139 89 93 84 185 129 108 130 72 114 110 137 115 121 86
En eje Z - Promedio	proyecction2D z promedio zprom.pgm	142 132 145 164 153 81 154 155 84 92 87 122 104 167 85 79 160 163 64 131 100 104 138 102 107 95 144 142 114 87	142 132 145 164 153 81 154 155 84 92 87 122 104 167 85 79 160 163 64 131 100 104 138 102 107 95 144 142 114 87

Table 6: Plan de pruebas 2 para proyección 2D

EVIDENCIA DE EJECUCIÓN - PRUEBAS 1 PROYECCION 2D	
Descripción de caso	Evidencia de ejecución
Proyección 2D con dirección inválida	\$ proyeccion2D w minimo salida.pgm Error: Direccion Invalida \$ █
Proyección 2D con criterio invalido	\$ proyeccion2D x aguacate salida.pgm Error: Criterio Invalido \$ █
Proyección 2D sin volumen cargado	\$ proyeccion2D x maximo salida.pgm No hay un volumen cargado en memoria \$ █
En eje X - Maximo	\$ proyeccion2D x maximo xmax.pgm Proyeccion generado con exito, guardando archivo... Criterio: maximo Direccion: x Archivo xmax.pgm guardado con exito \$ █
En eje Y - Maximo	\$ proyeccion2D y maximo ymax.pgm Proyeccion generado con exito, guardando archivo... Criterio: maximo Direccion: y Archivo ymax.pgm guardado con exito \$ █
En eje Z - Maximo	\$ proyeccion2D z maximo zmax.pgm Proyeccion generado con exito, guardando archivo... Criterio: maximo Direccion: z Archivo zmax.pgm guardado con exito \$ █
En eje X - Minimo	\$ proyeccion2D x minimo xmin.pgm Proyeccion generado con exito, guardando archivo... Criterio: minimo Direccion: x Archivo xmin.pgm guardado con exito \$ █
En eje Y - Minimo	\$ proyeccion2D y minimo ymin.pgm Proyeccion generado con exito, guardando archivo... Criterio: minimo Direccion: y Archivo ymin.pgm guardado con exito \$ █
En eje Z - Minimo	\$ proyeccion2D z minimo zmin.pgm Proyeccion generado con exito, guardando archivo... Criterio: minimo Direccion: z Archivo zmin.pgm guardado con exito \$ █

Table 7: Evidencia de ejecución - Plan de pruebas 1 proyeccion 2D

EVIDENCIA DE EJECUCIÓN - PRUEBAS 2 PROYECCION 2D	
Descripción de caso	Evidencia de ejecución
En eje X - Mediana	\$ proyeccion2D x mediana xmed.pgm Proyección generado con éxito, guardando archivo... Criterio: mediana Dirección: x Archivo xmed.pgm guardado con éxito \$ █
En eje Y - Mediana	\$ proyeccion2D y mediana ymed.pgm Proyección generado con éxito, guardando archivo... Criterio: mediana Dirección: y Archivo ymed.pgm guardado con éxito \$ █
En eje Z - Mediana	\$ proyeccion2D z mediana zmed.pgm Proyección generado con éxito, guardando archivo... Criterio: mediana Dirección: z Archivo zmed.pgm guardado con éxito \$ █
En eje X - Promedio	\$ proyeccion2D x promedio xprom.pgm Proyección generado con éxito, guardando archivo... Criterio: promedio Dirección: x Archivo xprom.pgm guardado con éxito \$ █
En eje Y - Promedio	\$ proyeccion2D y promedio yprom.pgm Proyección generado con éxito, guardando archivo... Criterio: promedio Dirección: y Archivo yprom.pgm guardado con éxito \$ █
En eje Z - Promedio	\$ proyeccion2D z promedio zprom.pgm Proyección generado con éxito, guardando archivo... Criterio: promedio Dirección: z Archivo zprom.pgm guardado con éxito \$ █

Table 8: Evidencia de ejecución - Plan de pruebas 2 proyección 2D