

# Proyecto

## Presentado por:

Alexander Aponte Largacha  
Correo: `aponte_a@javeriana.edu.co`

Diego Fernando Castrillón Cortés  
Correo: `diego-castrillon@javeriana.edu.co`

Sara Ocampo Feria  
Correo: `sa.ocampo@javeriana.edu.co`

## Asignatura:

Estructura de datos

## Universidad

Pontificia Universidad Javeriana  
June 3, 2025

# Acta de evaluación

## Primera Entrega

A partir de los comentarios recibidos del profesor, se implementaron las siguientes correcciones fundamentales para mejorar la calidad del diseño:

### Independencia del Lenguaje de Programación

Se eliminaron todas las referencias directas a lenguajes específicos de programación, logrando un diseño conceptual puro que se enfoca exclusivamente en la lógica del sistema. Esta modificación garantiza que el diseño sea aplicable independientemente de la tecnología de implementación seleccionada.

### Incorporación de TAD Adicional

Se agregó un nuevo Tipo Abstracto de Dato (TAD) que encapsula los atributos de imagen y volumen. Este TAD representa el sistema general sobre el cual operan las funciones principales del proyecto, proporcionando una representación más completa y estructurada del dominio del problema.

### Desarrollo del Plan de Pruebas

Se elaboró un plan de pruebas integral que abarca los diferentes escenarios de uso del sistema. Este plan permite:

- Validar el correcto funcionamiento de todas las funcionalidades implementadas
- Asegurar la calidad del producto final
- Identificar posibles casos límite y errores potenciales

## Segunda Entrega

Para la segunda entrega, se identificaron únicamente errores en el diagrama de TAD, por lo que se realizaron las correcciones correspondientes al diagrama de clases, tal como se muestra en la imagen adjunta.

### Problema Identificado: Jerarquía Incorrecta entre Clases

El principal error detectado fue una relación incorrecta en la jerarquía entre las clases del sistema:

### **Corrección 1: Relación Sistema - Árbol de Huffman**

- **Error anterior:** El diagrama mostraba incorrectamente que el Árbol de Huffman contenía al Sistema
- **Corrección aplicada:** Se modificó la relación para reflejar que el Sistema contiene un Árbol de Huffman
- **Justificación:** Esta corrección representa adecuadamente la lógica del sistema, donde el Sistema utiliza el Árbol de Huffman como herramienta para realizar las tareas de compresión y descompresión de imágenes

### **Relación Mantenido: Árbol de Huffman - Nodo**

- Esta relación se mantiene sin cambios por ser correcta
- El Árbol de Huffman está compuesto por Nodos
- Cada Nodo representa un carácter y su frecuencia correspondiente
- Los Nodos conforman la estructura del árbol binario utilizado para la codificación Huffman

# Entrega 1

## 1 Análisis Inicial

El proyecto consiste en el desarrollo de un sistema de procesamiento de imágenes en escala de grises, utilizando diversas estructuras de datos y una interfaz basada en línea de comandos. Su propósito es generar representaciones bidimensionales a partir de volúmenes tridimensionales de imágenes.

Para este sistema, se empleará el formato PGM (Portable Gray Map), específicamente en su variante P2, que almacena imágenes en una matriz de valores en escala de grises. Aunque existen otras variantes como P3 y P5 (capaces de manejar color), el proyecto se centrará exclusivamente en PGM P2.

El formato PGM facilita la manipulación y análisis de imágenes, ya que cada imagen se representa mediante una matriz de tamaño  $N \times M$ , donde cada celda corresponde a un píxel con un valor entre 0 y 255. En esta escala, 0 representa el color negro y 255 el blanco.

### 1.1 Construcción de volúmenes

Un volumen se forma cuando se apilan múltiples imágenes en capas sucesivas, permitiendo visualizar la estructura en tres dimensiones. Dependiendo de la perspectiva de observación, se pueden generar vistas desde diferentes ejes:

- Eje X: Permite observar la secuencia de imágenes de lado.
- Eje Y: Muestra la serie de imágenes desde arriba.
- Eje Z: Proporciona una vista frontal de la secuencia.

Cada imagen está definida en el primer cuadrante del plano cartesiano, con coordenadas (x,y) en un rango de valores discretos entre 0 y 255. Para representar volúmenes, se introduce un tercer eje z, que indica la posición de cada imagen dentro del conjunto tridimensional.

## 2 Descripción de Comandos - Entradas, Salidas y Condiciones

### 2.1 Procedimiento Principal

**main()**

- **Entrada:** Comandos ingresados por el usuario a través de la terminal.
- **Salida:** Ejecución de comandos y mensajes correspondientes al usuario.
- **Condiciones:**

- El programa se ejecuta en un bucle continuo hasta que el usuario introduzca el comando **salir**.
- Cada comando debe ser reconocido por la función `obtenerCodigoComando()` para ser procesado correctamente.

## 2.2 Operaciones Auxiliares (Comandos)

### 2.2.1 `cargar_imagen(const string &nombre_imagen)`

- **Entrada:** Nombre del archivo de imagen PGM a cargar.
- **Salida:** Mensaje de éxito o error al cargar la imagen.
- **Condiciones:**
  - El archivo debe existir físicamente en el sistema.
  - El archivo debe tener un formato PGM válido (P2).
- **Ejemplo correcto:** `cargar_imagen img_02.pgm` (carga una imagen existente)
- **Ejemplo incorrecto:** `cargar_imagen` (sin especificar archivo) o `cargar_imagen archivo_inexistente.pgm`

### 2.2.2 `cargar_volumen(const string &nombre_base, int n_im)`

- **Entrada:**
  - `nombre_base`: Base del nombre de los archivos que forman el volumen.
  - `n_im`: Número de imágenes a cargar en el volumen.
- **Salida:** Mensaje de éxito o error al cargar el volumen.
- **Condiciones:**
  - Los archivos deben existir siguiendo la convención `nombre_base.i.pgm` donde `i` va de 0 a `n_im-1`.
  - Todos los archivos deben ser imágenes PGM válidas.
- **Ejemplo correcto:** `cargar_volumen img_ 3` (carga `img_0.pgm`, `img_1.pgm`, `img_2.pgm`)
- **Ejemplo incorrecto:** `cargar_volumen img_ -1` (número negativo) o cuando faltan archivos de la secuencia.

### 2.2.3 info\_imagen()

- **Entrada:** Ninguna (utiliza la imagen global cargada).
- **Salida:** Información detallada sobre la imagen cargada o mensaje de error.
- **Condiciones:**
  - Debe haberse cargado previamente una imagen con `cargar_imagen`.
- **Ejemplo correcto:** `info_imagen` (después de cargar una imagen).
- **Ejemplo incorrecto:** `info_imagen` (sin haber cargado una imagen previamente).

### 2.2.4 info\_volumen()

- **Entrada:** Ninguna (utiliza el volumen global cargado).
- **Salida:** Información detallada sobre el volumen cargado o mensaje de error.
- **Condiciones:**
  - Debe haberse cargado previamente un volumen con `cargar_volumen`.
- **Ejemplo correcto:** `info_volumen` (después de cargar un volumen).
- **Ejemplo incorrecto:** `info_volumen` (sin haber cargado un volumen previamente).

## 3 Tipos Abstractos de Datos (TADs)

### 3.1 TAD Imagen

#### Estado

- `Nodo`: (Tipo Abstracto de Dato Nodo), Nodo raíz del árbol.
- `tablaCodigos`: (Mapa de cadena de caracteres), Camino hacia cada uno de los píxeles **Condición:**  $\text{width} \geq 0$
- `height`: (int), alto de la imagen en píxeles. **Condición:**  $\text{height} \geq 0$
- `maxValue`: (int), valor máximo que puede tener un píxel. **Condición:**  $\text{maxValue} \geq 0$
- `pixeles`: (`vector<vector<int>>`), matriz bidimensional que almacena los valores de los píxeles.
  - `pixeles.size() == height`
  - `pixeles[i].size() == width` para todo `i`

- **type:** (string), tipo de formato de la imagen (P2 para PGM).

#### Interfaz

- **Imagen()** - Operación para crear una imagen vacía.
  - **Post:** width = 0, height = 0, maxValue = 0
- **Imagen(int w, int h)** - Operación para crear una imagen con dimensiones específicas.
  - **Pre:**  $w \geq 0$ ,  $h \geq 0$
  - **Post:** width = w, height = h, pixeles inicializados con valor 0
- **cargarImagen(const string &nombreArchivo)** - Operación para cargar una imagen desde un archivo.
  - **Pre:** El archivo debe existir y ser de tipo PGM (P2)
  - **Post:** La imagen se carga en memoria con todos sus atributos
- **guardarImagenPersistencia(string &nombre\_archivo)** - Operación para guardar la imagen en un archivo.
  - **Pre:** nombre\_archivo no debe estar vacío
  - **Post:** La imagen se guarda en un archivo con el formato correspondiente
- **infoImagen()** - Operación que devuelve información sobre la imagen.
  - **Post:** Retorna un string con información de nombreImagen, width, height y maxValue
- **getPixel(int i, int j)** - Operación para obtener el valor de un píxel específico.
  - **Pre:**  $0 \leq i < \text{width}$ ,  $0 \leq j < \text{height}$
  - **Post:** Retorna el valor del píxel en la posición (i,j) o -1 si está fuera de rango
- **setPixel(int i, int j, int valor)** - Operación para establecer el valor de un píxel específico.
  - **Pre:**  $0 \leq i < \text{width}$ ,  $0 \leq j < \text{height}$
  - **Post:** El píxel en la posición (i,j) se establece con el valor especificado

## 3.2 TAD Volumen

### Estado

- **cantImagenes:** (int), cantidad de imágenes que contiene el volumen. **Condición:**  $\text{cantImagenes} \geq 0$
- **imagenes:** (vector<Imagen>), vector que almacena las imágenes que conforman el volumen.
  - $\text{imagenes.size()} == \text{cantImagenes}$

### Interfaz

- **Volumen()** - Operación para crear un volumen vacío.
  - **Post:**  $\text{cantImagenes} = 0$ , imagenes es un vector vacío
- **Volumen(int cantImg)** - Operación para crear un volumen con un número específico de imágenes.
  - **Pre:**  $\text{cantImg} \geq 0$
  - **Post:**  $\text{cantImagenes} = \text{cantImg}$ , imagenes es un vector de tamaño  $\text{cantImg}$
- **getCanImagenes()** - Operación que retorna la cantidad de imágenes en el volumen.
  - **Post:** Retorna el valor de  $\text{cantImagenes}$
- **setCantImagenes(int cantImagenes)** - Operación para establecer la cantidad de imágenes del volumen.
  - **Pre:**  $\text{cantImagenes} \geq 0$
  - **Post:** El atributo  $\text{cantImagenes}$  se actualiza con el valor especificado
- **infoVolumen()** - Operación que muestra información sobre el volumen.
  - **Post:** Muestra información de cada imagen del volumen y la cantidad total
- **agregarImagen(const Imagen& imagen)** - Operación para agregar una imagen al volumen.
  - **Post:** La imagen se agrega al vector  $\text{imagenes}$



### 3.3 Diagrama del TAD

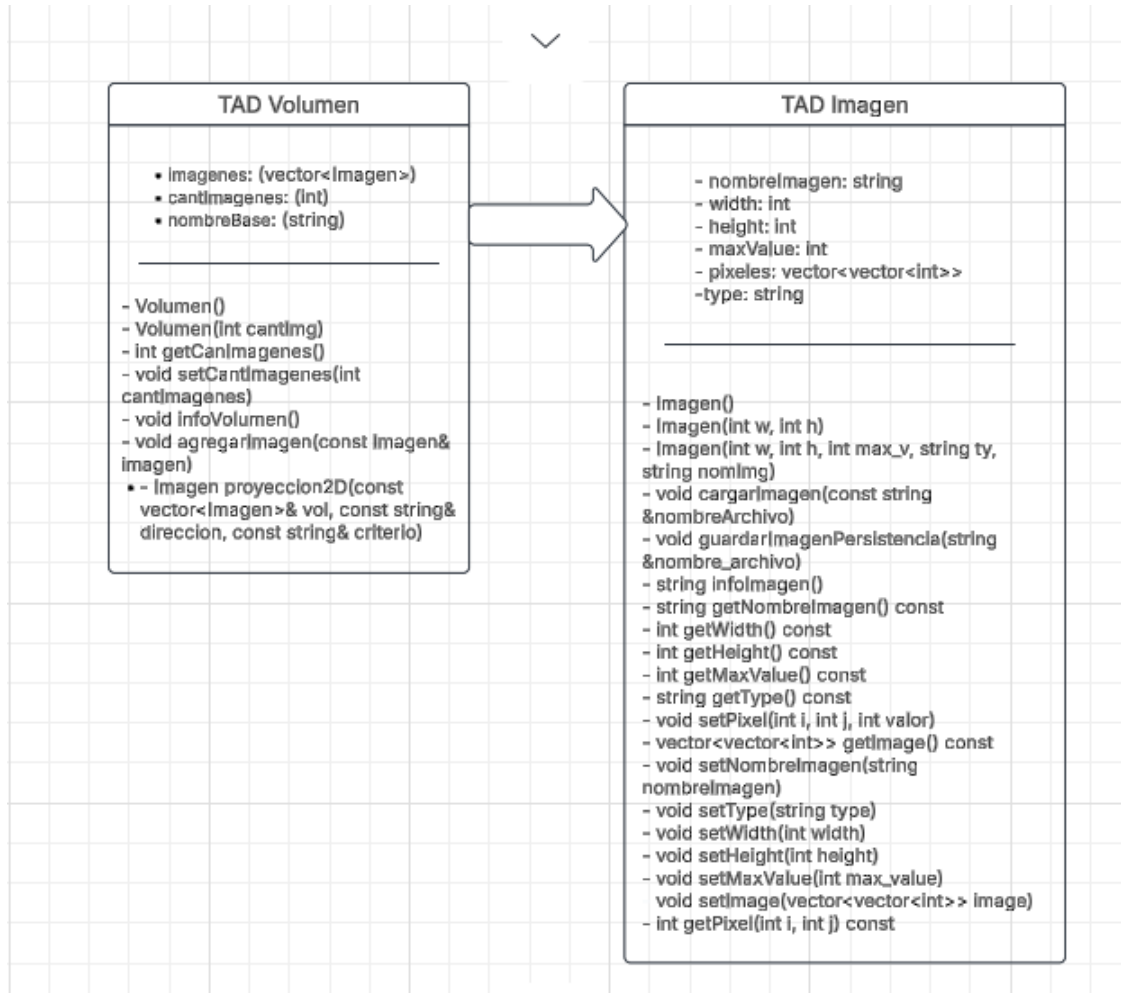
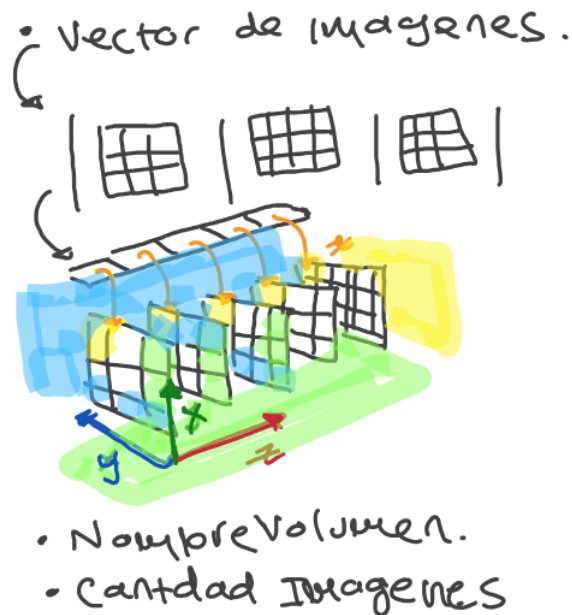


Diagrama del TAD

# Corrección de Entrega 1

A partir de los comentarios recibidos tanto en la entrega 1 como en la 2, se realizaron diversas correcciones e implementaciones clave con el objetivo de mejorar la calidad del diseño y asegurar su independencia del lenguaje de programación. En primer lugar, se eliminó cualquier referencia directa a lenguajes específicos, garantizando así un diseño conceptual limpio y orientado únicamente a la lógica del sistema. Además, se incorporó un Tipo Abstracto de Dato (TAD) adicional que incluye los atributos de imagen y volumen, representando el sistema general sobre el cual se ejecutarán las funciones principales del proyecto. Esta adición permite una representación más completa del dominio del problema. Se mejoró la implementación de la lectura de imágenes y volúmenes enfocándonos en el manejo eficiente de los errores que se pueden presentar en la lectura de estos archivos. Finalmente, se elaboró y se completó el plan de pruebas, haciéndolo más robusto que cubre los distintos escenarios y errores del uso del sistema, permitiendo validar el correcto funcionamiento de las funcionalidades implementadas y asegurando la calidad del producto final.



Analisis del problema de proyeccion 2D

## Tipos Abstractos de Datos (TADs)

### TAD Imagen

#### Estado

- **ancho**: Numero de enteros.
  - Indica el ancho de la imagen.
- **alto**: Numero de enteros.
  - Indica la altura de la imagen.
- **píxeles**: Colección de Imágenes.
  - Representa los píxeles que componen la imagen, organizados en una estructura que permite acceder a cada uno de ellos.
- **nombreArchivo**: Cadena de caracteres.
  - Indica el nombre del archivo asociado a la imagen.
- **valorMaximo**: Numero de enteros.
  - Representa el valor máximo que puede tomar un píxel en la imagen.

#### Interfaz

- **Imagen()**: Operación para crear una imagen sin dimensiones y sin píxeles definidos.
  - **Post**: La imagen creada tiene un ancho y alto de 0, y no contiene píxeles.
- **Imagen(ancho, alto)**: Operación para crear una imagen con dimensiones específicas.
  - **Post**: La imagen creada tiene el ancho y alto especificados, y sus píxeles están inicializados a 0.
- **getAncho()**: Operación que retorna el ancho de la imagen.
  - **Pre**: La imagen debe estar correctamente inicializada.
- **getAlto()**: Operación que retorna la altura de la imagen.
  - **Pre**: La imagen debe estar correctamente inicializada.
- **getNombreArchivo()**: Operación que retorna el nombre del archivo asociado a la imagen.
  - **Pre**: La imagen debe estar correctamente inicializada.

- **setAncho(ancho)**: Operación que establece el ancho de la imagen.
  - **Post**: El ancho de la imagen es actualizado al valor especificado.
- **setAlto(alto)**: Operación que establece la altura de la imagen.
  - **Post**: La altura de la imagen es actualizada al valor especificado.
- **setNombreArchivo(nombre)**: Operación que establece el nombre del archivo asociado a la imagen.
  - **Post**: El nombre del archivo de la imagen es actualizado al valor especificado.
- **setValorMaximo(valor)**: Operación que establece el valor máximo que puede tomar un píxel en la imagen.
  - **Post**: El valor máximo de los píxeles es actualizado al valor especificado.
- **setPíxeles(píxeles)**: Operación que establece los píxeles de la imagen.
  - **Post**: Los píxeles de la imagen son actualizados a los valores especificados.
- **getPixel(x, y)**: Operación que retorna el valor del píxel en la posición especificada.
  - **Pre**: Las coordenadas deben estar dentro de los límites de la imagen.
- **setPixel(x, y, valor)**: Operación que establece el valor de un píxel en la posición especificada.
  - **Pre**: Las coordenadas deben estar dentro de los límites de la imagen.
  - **Post**: El píxel en la posición (x, y) tiene el valor especificado.
- **infoImagen()**: Operación que muestra información sobre la imagen.
  - **Post**: Se muestra el nombre del archivo y las dimensiones de la imagen.
- **guardarImagen(nombreArchivo)**: Operación para guardar la imagen en un archivo.
  - **Post**: La imagen es guardada en el archivo especificado.
- **pruebaInfoImagen()**: Operación que muestra los valores de los píxeles de la imagen.
  - **Post**: Se muestran los valores de todos los píxeles de la imagen.

## TAD Volumen

### Estado

- **imagenes**: Colección de Imagen.
  - Representa una colección de imágenes que conforman el volumen.
- **cantImagenes**: Numero de enteros.
  - Indica la cantidad de imágenes en el volumen.
- **nombreBase**: Cadena de caracteres.
  - Representa el nombre base utilizado para cargar las imágenes del volumen.

### Interfaz

- **Volumen()**: Operación para crear un volumen sin imágenes.
  - **Post**: El volumen creado no contiene imágenes y la cantidad de imágenes es 0.
- **Volumen(cantImagenes)**: Operación para crear un volumen con una cantidad específica de imágenes.
  - **Post**: El volumen creado tiene espacio reservado para la cantidad de imágenes especificada.
- **getCantImagenes()**: Operación que retorna la cantidad de imágenes en el volumen.
  - **Post**: Retorna el número de imágenes en el volumen.
- **setCantImagenes(cantImagenes)**: Operación que establece la cantidad de imágenes en el volumen.
  - **Post**: La cantidad de imágenes en el volumen es actualizada al valor especificado.
- **setNombreBase(nom)**: Operación que establece el nombre base del volumen.
  - **Post**: El nombre base del volumen es actualizado al valor especificado.
- **getNombreBase()**: Operación que retorna el nombre base del volumen.
  - **Post**: Retorna el nombre base del volumen.
- **agregarImagen(img)**: Operación que añade una imagen al volumen.

- **Post:** La imagen es añadida al volumen.
- **proyeccion2D(volumen, dirección, criterio, nomArchivo):** Operación de proyeccion 2D según el criterio que el usuario escoga sobre el volumen cargado en memoria en el programa.
  - **Pre:** El volumen de imagenes con extesion pgm debe estar cargado previamente en la memoria del programa.
  - **Post:** La proyeccion 2D se calculara, guardara y se genera un archivo de salida plano.
- **crit(valores, criterio):** Operación que calcula sobre los valores el criterio dado por el usuario.
  - **Pre:** El volumen de imagenes con extesion pgm debe estar cargado previamente en la memoria del programa.
  - **Post:** Retorno de los valores necesarios para la operación de la función Proyeccion2D.
- **getImagen(indice):** Operacion que retorna una imagen específica segun el indice enviado como parametro
  - **Pre:** El volumen de imagenes con extesion pgm debe estar cargado previamente en la memoria del programa.
  - **Post:** Retorna la imagen.
- **infoVolumen():** Operación que muestra información sobre el volumen cargado.
  - **Post:** Se muestra la información del volumen en la consola, incluyendo el nombre del volumen, profundidad, alto y ancho.

## TAD Sistema

### Estado

- **img:** Objeto imagen.
  - Representa la imagen cargada en el sistema.
- **imagenCargada:** Bandera booleana.
  - Representa la bandera del sistema para reconocer si una imagen ha sido cargada en el programa.
- **vol:** Objeto volumen.
  - Representa el volumen de imágenes cargado en el sistema.
- **volumenCargado:** Bandera booleana.

- Representa la bandera del sistema para reconocer si un volumen ha sido cargado en el programa.

## Interfaz

- **getImagen()**: Operación que retorna la imagen cargada en el sistema.
  - **Post:** Retorna la imagen actualmente cargada.
- **getVolumen()**: Operación que retorna el volumen cargado en el sistema.
  - **Post:** Retorna el volumen actualmente cargado.
- **obtenerCodigoComando(comando)**: Operación que asigna un número a cada comando.
  - **Pre:** El comando debe ser una cadena de caracteres válida.
  - **Post:** Retorna el código correspondiente al comando.
- **mostrarAyuda(comando)**: Operación que muestra la lista de comandos disponibles o información detallada sobre un comando específico.
  - **Post:** Se muestra la información de ayuda en la consola.
- **validarArchivoExiste(nombre)**: Operación que verifica si un archivo existe.
  - **Pre:** El nombre del archivo debe ser una cadena de caracteres válida.
  - **Post:** Retorna verdadero si el archivo existe, falso en caso contrario.
- **cargarImagen(nombre\_imagen)**: Operación que carga una imagen desde un archivo.
  - **Pre:** El archivo debe existir y ser un archivo PGM válido.
  - **Post:** La imagen es cargada en el sistema.
- **leerImagen(nombreArchivo)**: Operación que lee una imagen desde un archivo y la retorna.
  - **Pre:** El archivo debe existir y ser un archivo PGM válido.
  - **Post:** Retorna la imagen leída.
- **cargarVolumen(nombreBase, cantImagenes)**: Operación que carga un volumen de imágenes desde una serie de archivos.
  - **Pre:** Los archivos deben existir y ser archivos PGM válidos.
  - **Post:** El volumen es cargado en el sistema.
- **leerVolumen(nombreArchivo)**: Operación que lee un volumen desde varios archivos dado un numero base del archivo y la cantidas de imagenes.

- **Pre:** Los archivos deben existir y ser de extensión PGM válido.
- **Post:** Retorna el volumen.
- **infoImagen():** Operación que muestra información sobre la imagen cargada.
  - **Pre:** La imagen ya debe haber sido cargada en la memoria del programa.
  - **Post:** Se muestra la información de la imagen en la consola.
- **infoVolumen():** Operación que muestra información sobre el volumen cargado.
  - **Pre:** El volumen ya debe haber sido cargado en la memoria del programa.
  - **Post:** Se muestra la información del volumen en la consola.
- **proyeccion2D(direccion, criterio, nombre\_archivo):** Operación para generar una proyección 2D del volumen.
  - **Post:** Se muestra un mensaje de error indicando que la operación no está implementada.
- **codificar\_imagen(nombre\_archivo):** Operación para codificar la imagen en memoria con Huffman.
  - **Post:** Se muestra un mensaje de error indicando que la operación no está implementada.
- **decodificar\_archivo(nombre\_archivo, nombre\_imagen):** Operación para decodificar un archivo .huf a PGM.
  - **Post:** Se muestra un mensaje de error indicando que la operación no está implementada.
- **segmentar(salida\_imagen, semillas):** Operación para segmentar la imagen en memoria.
  - **Post:** Se muestra un mensaje de error indicando que la operación no está implementada.
- **reiniciarSistema():** Operación para reiniciar las banderas booleanas y borrar de la memoria del programa la imagen y volumen cargados en el uso del sistema.
  - **Post:** Se muestra un mensaje de despedida que anuncia el correcto reinicio del sistema.



## Diagrama de relación de TADs

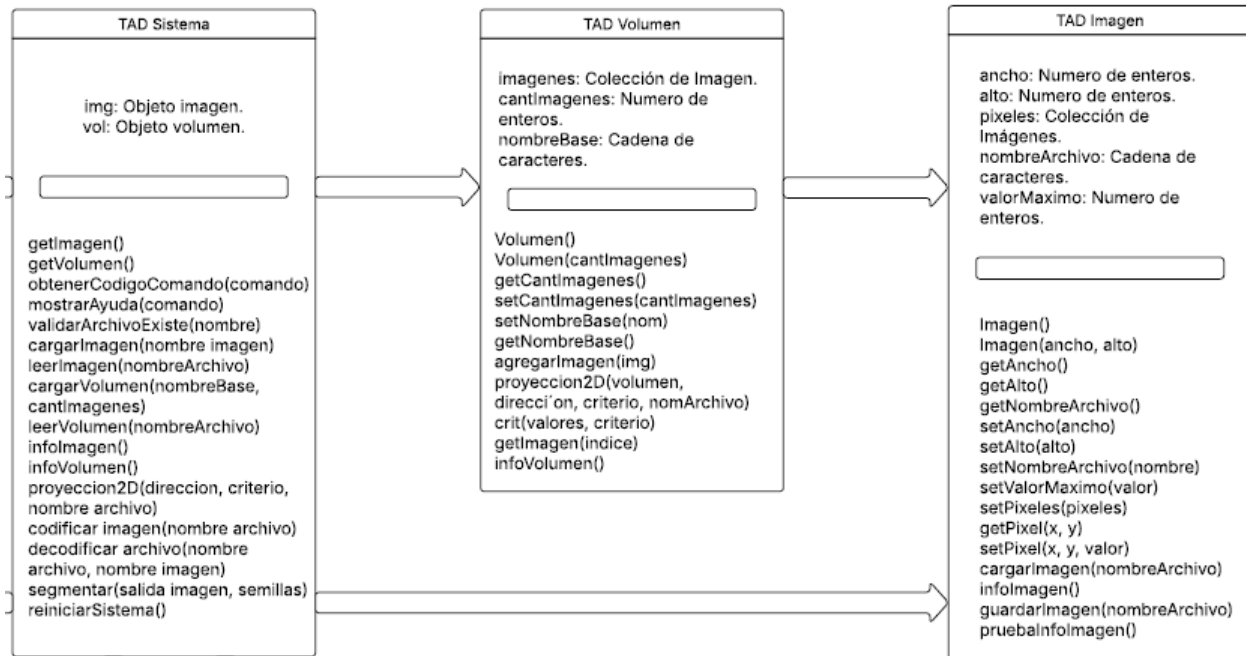


Diagrama del TAD

## Casos de prueba

### Casos de Prueba Entrega 1

Casos de prueba comando Cargar_imagen				
Descripción	Datos de Entrada	Resultado Esperado	Resultado Obtenido	Evidencia
La imagen no existe en la carpeta del proyecto	cargar_imagen imagen.pgm	"Error al cargar la imagen: ... El archivo no existe"	(Error) Error al cargar la imagen: Archivo no encontrado: imagen.pgm	<pre>PS C:\Users\sarao\Downloads\Proyecto2REAL\Proyecto2REAL&gt; ./nombre Sistema de procesamiento de imagenes en escala de grises. Escriba 'ayuda' para ver los comandos disponibles. \$ cargar_imagen imagen.pgm (Error) Error al cargar la imagen: Archivo no encontrado: imagen.pgm \$</pre>
Se da la imagen sin la extensión del archivo pgm	cargar_imagen img	"Error al cargar la imagen: ... No se pudo abrir el archivo"	(Error) Error al cargar la imagen: Archivo no encontrado: img	<pre>\$ cargar_imagen img (Error) Error al cargar la imagen: Archivo no encontrado: img</pre>
Se da la imagen con un encabezado invalido, no correspondiente a P2	P3 64 255 0 0 0 255 70 0 0 0 255 50 0 0 0 255 0 0 0 50	(Error)El archivo no es un archivo PGM válido.	(Error) Error al cargar la imagen: Formato de imagen no válido: (Error) El archivo img.pgm no es un archivo PGM válido.	<pre>\$ cargar_imagen img.pgm (Error) Error al cargar la imagen: Formato de imagen no válido: : (Error) El archivo img.pgm no es un archivo PGM válido. \$</pre>

Casos de prueba cargarImagen

Archivo con pixeles faltantes	P2 6 4 255 0 0 0 0 255 70 0 0 0 0 255 50 0 0 0 0 255 0 0 0 0 50	(Error) Error al cargar la imagen: (Error) Problema al leer los datos de la imagen. Faltan datos o formato incorrecto.	Error al cargar la imagen: (Error) Problema al leer los datos de la imagen. Faltan datos o formato incorrecto	Sistema de procesamiento de imagenes en escala de grises. Escriba 'ayuda' para ver los comandos disponibles.  \$ cargar_imagen img.pgm (Error) Error al cargar la imagen: (Error) Problema al leer los datos de la imagen. Faltan datos o formato incorrecto. \$ █
Archivo con comentarios	P2 # Hola mundo # lalala 6 4 255 0 0 14 0 255 70 0 18 0 0 255 50 0 20 0 0 255 21 0 0 25 0 50 39	Imagen cargada con exito: img.pgm	Imagen cargada con exito: img.pgm 0 0 14 0 255 70 0 18 0 0 255 50 0 20 0 0 255 21 0 0 25 0 50 39	\$ cargar_imagen img.pgm Imagen cargada con exito: img.pgm 0 0 14 0 255 70 0 18 0 0 255 50 0 20 0 0 255 21 0 0 25 0 50 39 \$ █
Archivo con valores de alto y ancho: en 0 y negativos	P2 # Hola mundo -6 0 255 0 0 14 0 255 70 0 18 0 0 255 50 0 20 0 0 255 21 0 0 25 0 50 39	(Error) Error al cargar la imagen: Valores inválidos en la cabecera del archivo.	Valores inválidos en la cabecera del archivo. (Error) Error al cargar la imagen: Formato de imagen no válido: (Error) La cabecera del archivo img.pgm es inválida.	PS C:\Users\saraol\Downloads\Proyecto2REAL\Proyecto2REAL> ./nombre Sistema de procesamiento de imagenes en escala de grises. Escriba 'ayuda' para ver los comandos disponibles.  \$ cargar_imagen img.pgm valores inválidos en la cabecera del archivo. (Error) Error al cargar la imagen: Formato de imagen no válido: (Error) La cabecera del archivo img.pgm es inválida. \$ █

Casos de prueba cargarImagen

Archivo con el valor maximo de pixeles fuera del rango 0 - 255	P2 6 4 345 0 0 14 0 255 70 0 18 0 0 255 50 0 20 0 0 255 21 0 0 25 0 50 39	(Error) Error al cargar la imagen: Valores inválidos en la cabecera del archivo.	Valores inválidos en la cabecera del archivo. (Error) Error al cargar la imagen: Formato de imagen no válido: (Error) La cabecera del archivo img.pgm es inválida.	<pre>\$ cargar_imagen img.pgm Valores inválidos en la cabecera del archivo. (Error) Error al cargar la imagen: Formato de imagen no válido: (Error) La cabecera del archivo img.pgm es inválida.</pre>
Pixeles mayores al valor maximo de la cabecera del enunciado	P2 6 4 240 0 0 14 0 255 70 0 18 0 0 255 50 0 20 0 0 255 21 0 0 25 0 50 39	(Error) Error al cargar la imagen: (Error) Valor de píxel fuera del rango permitido.	(Error) Error al cargar la imagen: (Error) Valor de píxel fuera del rango permitido.	<pre>\$ cargar_imagen img.pgm (Error) Error al cargar la imagen: (Error) Valor de píxel fuera del rango permitido.</pre>
Imagen con mas pixeles que los mencionados en el encabezado	P2 6 4 240 0 0 14 0 255 70 0 18 0 0 255 50 0 20 0 0 255 21 0 0 25 0 50 39 156 34 56 98	(Error) Error al cargar la imagen: Error el archivo contiene datos adicionales despues de los pixeles	(Error) Error al cargar la imagen: (Error) El archivo img.pgm contiene datos adicionales despu -s de los píxeles.	<pre>PS C:\Users\varao\Downloads\Proyecto2024\Proyecto2024&gt; ./cargar_imagen Sistema de procesamiento de imagenes en escala de grises. Escriba 'ayuda' para ver los comandos disponibles.  \$ cargar_imagen img.pgm (Error) Error al cargar la imagen: (Error) El archivo img.pgm contiene datos adicionales despu -s de los píxeles.</pre>
Cargar otra imagen en memoria, es decir mas de una.	Enviar el comando cargar_imagen img.pgm dos veces	(Error): Ya hay una imagen cargada en memoria. Por favor, usa 'info_imagen' para ver la informacion de la imagen actual.	Ya hay una imagen cargada en memoria. Por favor, usa 'info_imagen' para ver la informacion de la imagen actual.	<pre>\$ cargar_imagen img.pgm Imagen cargada con exito: img.pgm 0 0 14 0 255 70 0 18 0 0 255 50 0 20 0 0 255 21 0 0 25 0 50 39  \$ cargar_imagen img.pgm Ya hay una imagen cargada en memoria. Por favor, usa 'info_imagen' para ver la informacion de la imagen actual.</pre>

Casos de prueba cargarImagen

Casos de prueba comando Cargar_imagen				
Descripción	Datos de Entrada	Resultado Esperado	Resultado Obtenido	Evidencia
Se dara una serie de volumen con imagenes erroneas	Todos los archivos de volumen tendran los mismos datos, uno contendra un error en la cabecera. Cargar_volumen vol 3	Existio un error al cargar el volumen. ->Cuando se escriba en comando info_volumen no debera haber ninguno.	Formato de imagen no valido: (Error) El archivo vol02.pgm no es un PGM valido. Volumen vacio / (Error) No hay un volumen cargado en memoria.	<pre>\$ cargar_volumen vol 3 Formato de imagen no valido: (Error) El archivo vol02.pgm no es un archivo PGM valido. Volumen vacio \$ info_volumen (Error) No hay un volumen cargado en memoria. \$</pre>
Se dara la serie de volumen con datos correctos	Cargar_volumen vol 3 P2 # Hola mundo # lalala 6 4 255 0 0 0 0 255 70 0 0 0 0 255 50 0 0 0 0 255 200 0 0 0 0 50 10	Volumen cargado con exito. Nombre base: ----	Volumen cargado con exito. Nombre base: vol, Cantidad de imágenes: 3 Nombre del volumen: vol Cantidad de imágenes: 3 Dimensiones (ancho x alto): 6 x 4	<pre>\$ cargar_volumen vol 3 Volumen cargado con exito. Nombre base: vol, Cantidad de imágenes: 3 Nombre del volumen: vol Cantidad de imágenes: 3 Dimensiones (ancho x alto): 6 x 4 \$</pre>
Cargar más imagenes en el volumen que las existentes	Se repiten las mismas condiciones anteriores, cargar_volumen 5	Existio un error al cargar el volumen.	Archivo: vol01.pgm Archivo: vol02.pgm Archivo: vol03.pgm Archivo: vol04.pgm Error: El archivo vol04.pgm no existe. Volumen vacio	<pre>Sistema de procesamiento de imagenes en escala de grises. Escriba 'ayuda' para ver los comandos disponibles.  \$ cargar_volumen vol 5 archivo: vol01.pgm archivo: vol02.pgm archivo: vol03.pgm archivo: vol04.pgm Error: El archivo vol04.pgm no existe. volumen vacio \$</pre>

Casos de prueba cargarVolumen

Cargar un volumen con imagenes de distintas dimensiones	<p>Un volumen de tres imagenes, nombre base vol. Contienen la misma informacion (igual que en el caso 2), la imagen vol2.pgm tiene una fila adicional: P2</p> <pre># Hola mundo # lalala 6 5 255 0 0 0 0 255 70 0 0 0 0 255 50 0 0 0 0 255 200 0 0 0 0 50 10 0 0 0 0 50 10</pre>	Existio un error al cargar el volumen.	<p>Archivo: vol01.pgm  Archivo: vol02.pgm  Error: La imagen vol02.pgm tiene dimensiones diferentes a las de la primera imagen.  Volumen vacio</p>	<pre> Escriba 'ayuda' para ver los comandos disponibles.  \$ cargar_volumen vol 3 Archivo: vol01.pgm Archivo: vol02.pgm Error: La imagen vol02.pgm tiene dimensiones diferentes a las de la primera imagen. Volumen vacio \$  </pre>
Intentar cargar nuevamente un volumen en el programa	<p>Repetir el comando cargar_volumen vol 3 dos veces con la serie de imagenes: P2</p> <pre>6 4 255 0 0 0 0 255 70 0 0 0 0 255 50 0 0 0 0 255 200 0 0 0 0 50 23</pre>	<p>Error: ya existe un volumen cargado en memoria, haga uso del comando info_volumen para conocerla</p>	<p>\$ cargar_volumen vol 3  Archivo: vol01.pgm  Archivo: vol02.pgm  Archivo: vol03.pgm  Volumen cargado con exito. Nombre base: vol, Cantidad de imagenes: 3  Nombre del volumen: vol  Cantidad de imagenes: 3  Dimensiones (ancho x alto): 6 x 4  \$ cargar_volumen vol 3  Ya hay un volumen cargado en memoria. Por favor, usa 'info_volumen' para ver la informacion del volumen actual.  \$</p>	<pre> \$ cargar_volumen vol 3 Archivo: vol01.pgm Archivo: vol02.pgm Archivo: vol03.pgm Volumen cargado con exito. Nombre base: vol, Cantidad de imagenes: 3 Nombre del volumen: vol Cantidad de imagenes: 3 Dimensiones (ancho x alto): 6 x 4 \$ cargar_volumen vol 3 Ya hay un volumen cargado en memoria. Por favor, usa 'info_volumen' para ver la informacion del volumen actual. \$  </pre>

Casos de prueba cargarVolumen

Descripción del caso	Resultado esperado	Resultado obtenido	Evidencia
vol00.pgm: 0 0 0 0 255 70 0 0 0 0 255 50 0 0 0 0 255 200 0 0 0 0 100 10 vol01.pgm: 0 0 0 0 255 70 0 0 0 0 255 50 0 0 0 0 255 200 0 0 0 0 100 10 vol02.pgm: 0 0 0 0 255 70 0 0 0 0 255 50 0 0 0 0 255 200 0 0 0 0 100 10 vol03.pgm: 1	Prueba minima hacia X:		P2 6 4 255 0 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0 1 1
	0 0 0 0 1 1	0 0 0 0 1 1	
	0 0 0 0 1 1	0 0 0 0 1 1	
	0 0 0 0 1 1	0 0 0 0 1 1	
	0 0 0 0 1 1	0 0 0 0 1 1	
	Prueba maxima hacia Y:		P2 6 4 255 1 1 1 1 255 70 1 1 1 1 255 50 1 1 1 1 255 200 1 1 1 1 100 10
	1 1 1 1 255 70	1 1 1 1 255 70	
	1 1 1 1 255 50	1 1 1 1 255 50	
	1 1 1 1 255 200	1 1 1 1 255 200	
	1 1 1 1 100 10	1 1 1 1 100 10	
	Prueba promedio hacia Y:		P2 6 4 255 0 0 0 0 191 52 0 0 0 0 191 37 0 0 0 0 191 150 0 0 0 0 75 7
	0 0 0 0 191 52	0 0 0 0 191 52	
	0 0 0 0 191 37	0 0 0 0 191 37	
	0 0 0 0 191 150	0 0 0 0 191 150	
	0 0 0 0 75 7	0 0 0 0 75 7	
	Prueba minima hacia Z:		P2 6 4 255 0 0 0 0 255 70 0 0 0 0 255 50 0 0 0 0 255 200 0 0 0 0 100 10
	0 0 0 0 1 1	0 0 0 0 1 1	
	0 0 0 0 1 1	0 0 0 0 1 1	
	0 0 0 0 1 1	0 0 0 0 1 1	
	0 0 0 0 1 1	0 0 0 0 1 1	
	Pueba mediana hacia Z:		P2 6 4 255 0 0 0 0 255 70 0 0 0 0 255 50 0 0 0 0 255 200 0 0 0 0 100 10
	0 0 0 0 255 70	0 0 0 0 255 70	
	0 0 0 0 255 50	0 0 0 0 255 50	
	0 0 0 0 255 200	0 0 0 0 255 200	
	0 0 0 0 100 10	0 0 0 0 100 10	

Casos de prueba proyeccion2D

# Entrega 2

## 4 Analisis inicial

En esta etapa del proyecto, se desarrollará un sistema para la codificación y decodificación de imágenes en formato PGM (Portable GrayMap), con el objetivo de optimizar el uso del espacio de almacenamiento. Para ello, se implementará la estructura de datos conocida como Árbol de Huffman, la cual permite realizar una compresión sin pérdida de información mediante la asignación de códigos binarios de longitud variable según la frecuencia de aparición de los valores de intensidad de la imagen. Esta técnica busca reducir significativamente el tamaño de los archivos, manteniendo su integridad y facilitando su manejo en memoria y transmisión.

### 4.1 ¿Que es el arbol de Huffman?

Para este proyecto se utilizará la codificación de Huffman, una técnica de compresión sin pérdida que asigna códigos binarios de longitud variable a los símbolos, de manera que aquellos que aparecen con mayor frecuencia son representados con menos bits. Esta estrategia permite reducir el tamaño total de los datos comprimidos de forma eficiente.

La representación de esta codificación se realizará mediante un Árbol de Huffman. En esta estructura, cada hoja contiene un símbolo del conjunto a codificar, junto con su frecuencia de aparición en los datos originales. Los nodos internos del árbol almacenan la suma de las frecuencias de sus nodos hijos, lo que permite construir la estructura de forma jerárquica y garantizar que los caminos más cortos (y por tanto, los códigos más breves) correspondan a los símbolos más frecuentes.

## 5 Descripción de Comandos - Entradas, Salidas y Condiciones

### 5.1 Operaciones (Comandos)

#### 5.1.1 `codificar_imagen(const string &nombre_archivo)`

- **Entrada:** Nombre del archivo `.huf` donde se guardará la codificación.
- **Salida:** Mensaje de éxito o error al codificar la imagen.
- **Condiciones:**
  - Debe existir una imagen cargada previamente en memoria.
  - El archivo `.huf` se creará o sobrescribirá con la codificación.



- **Ejemplo correcto:** `codificar_imagen salida.huf` (codifica la imagen en memoria)
- **Ejemplo incorrecto:** `codificar_imagen` (sin especificar archivo) o `codificar_imagen` cuando no hay imagen cargada

### 5.1.2 `decodificar_archivo(const string &nombre_archivo, const string &nombre_imagen)`

- **Entrada:**
  - `nombre_archivo`: Nombre del archivo `.huf` a decodificar.
  - `nombre_imagen`: Nombre del archivo PGM donde se guardará la imagen decodificada.
- **Salida:** Mensaje de éxito o error al decodificar el archivo.
- **Condiciones:**
  - El archivo `.huf` debe existir físicamente en el sistema.
  - El archivo `.huf` debe tener un formato válido de codificación Huffman.
- **Ejemplo correcto:** `decodificar_archivo comprimida.huf resultado.pgm`
- **Ejemplo incorrecto:** `decodificar_archivo` (sin especificar archivos) o `decodificar_archivo archivo_inexistente.huf salida.pgm`

## 6 Tipos Abstractos de Datos (TADs)

### 6.1 TADs Utilizados

Para esta segunda entrega, se agregaron dos nuevos tipos abstractos de datos, que son los siguientes:

#### 6.1.1 TAD Nodo

##### Estado

- **valor:** Número entero.
  - Representa el valor del símbolo o píxel asociado al nodo. En los nodos internos, este valor se fija como -1.
- **frecuencia:** Número entero sin signo.
  - Indica cuántas veces aparece el símbolo representado por este nodo.
- **hijoIzq:** Referencia a otro nodo.

- Apunta al hijo izquierdo del nodo.
- **hijoDer**: Referencia a otro nodo.
  - Apunta al hijo derecho del nodo.

### Interfaz

- **Nodo()**: Constructor por defecto.
  - **Post**: Crea un nodo sin valor ni frecuencia definidos.
- **Nodo(valor, frecuencia)**: Constructor con parámetros.
  - **Post**: Crea un nodo hoja con un valor específico y su frecuencia.
- **Nodo(izq, der)**: Constructor para nodos internos.
  - **Post**: Crea un nodo combinando dos nodos hijos. El valor se fija en -1 y la frecuencia es la suma de las frecuencias de los hijos.
- **~Nodo()**: Destructor del nodo.
  - **Post**: Libera la memoria de los hijos de forma recursiva.
- **obtenerValor()**: Operación que retorna el valor del nodo.
  - **Post**: Devuelve el valor del símbolo o -1 si es un nodo interno.
- **obtenerFrecuencia()**: Operación que retorna la frecuencia del nodo.
  - **Post**: Devuelve la frecuencia asociada al símbolo o la suma de frecuencias si es interno.
- **obtenerHijoIzq()**: Operación que retorna el hijo izquierdo.
  - **Post**: Devuelve el nodo hijo izquierdo.
- **obtenerHijoDer()**: Operación que retorna el hijo derecho.
  - **Post**: Devuelve el nodo hijo derecho.
- **fijarValor(valor)**: Asigna un valor al nodo.
  - **Post**: El valor del nodo es actualizado.
- **fijarFrecuencia(frecuencia)**: Asigna una frecuencia al nodo.
  - **Post**: La frecuencia del nodo es actualizada.
- **fijarHijoIzq(hijo)**: Asigna el hijo izquierdo del nodo.
  - **Post**: Se actualiza la referencia del hijo izquierdo.
- **fijarHijoDer(hijo)**: Asigna el hijo derecho del nodo.

- **Post:** Se actualiza la referencia del hijo derecho.
- **esHoja():** Verifica si el nodo es una hoja.
  - **Post:** Retorna **true** si no tiene hijos; **false** en caso contrario.

## TAD ArbolHuffman

### Estado

- **raiz:** Puntero a Nodo.
  - Representa la raíz del árbol de Huffman.
- **tablaCodigos:** Mapa de enteros a cadenas.
  - Almacena el código binario asignado a cada valor tras generar los códigos.

### Interfaz

- **ArbolHuffman():** Constructor por defecto.
  - **Post:** Crea un árbol vacío (raíz nula, tabla vacía).
- **~ArbolHuffman():** Destructor.
  - **Post:** Libera toda la memoria asociada al árbol.
- **esVacio():** Consulta si el árbol está vacío.
  - **Post:** Retorna **true** si la raíz es nula.
- **getRaiz():** Obtiene la raíz del árbol.
  - **Post:** Devuelve un puntero al nodo raíz del árbol.
- **getTablaCodigos():** Devuelve la tabla de códigos.
  - **Post:** Retorna el mapa que relaciona valores con sus códigos binarios.
- **construirArbol(frecuencias):** Construye el árbol de Huffman.
  - **Pre:** El mapa de frecuencias no debe estar vacío.
  - **Post:** El árbol de Huffman se construye con base en las frecuencias.
- **generarCodigos():** Genera los códigos Huffman desde la raíz.
  - **Post:** Llena la tabla de códigos con los caminos binarios desde la raíz a cada hoja.
- **codificarDato(dato):** Codifica un valor.
  - **Pre:** El valor debe estar presente en la tabla de códigos.

- **Post:** Retorna el código binario del valor especificado.
- **comprimirImagen(nomArch, ancho, alto, vMaximo, pixeles):** Comprime una imagen.
  - **Post:** Codifica la imagen en un archivo comprimido usando Huffman.
- **decodificarSecuencia(bits, pos):** Decodifica una secuencia de bits.
  - **Pre:** El árbol debe estar construido.
  - **Post:** Retorna el valor decodificado a partir de la posición actual y la avanza.
- **descomprimirImagen(nom, pixeles, ancho, alto, maxVal):** Descomprime una imagen.
  - **Post:** Reconstruye los píxeles a partir del archivo comprimido.

### Interfaz Privada

- **generarCodigosRec(nodo, camino):** Genera los códigos recursivamente.
  - **Post:** Recorre el árbol y asigna códigos a cada hoja.
- **liberarArbol(nodo):** Libera la memoria del árbol recursivamente.
  - **Post:** Libera todos los nodos del árbol.

## 6.2 Diagrama de relación de TADs

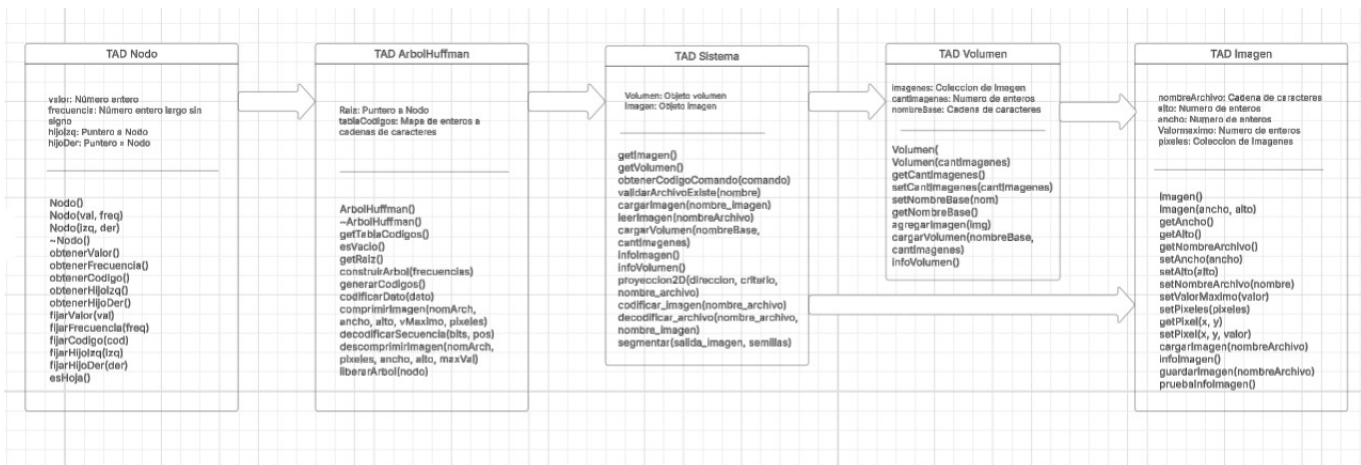
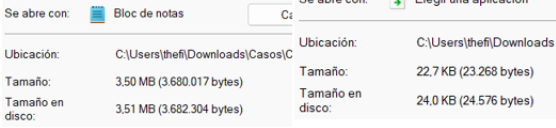
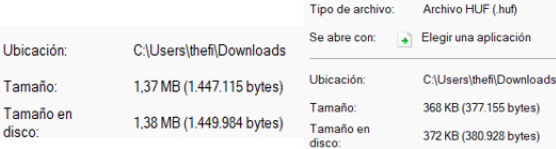
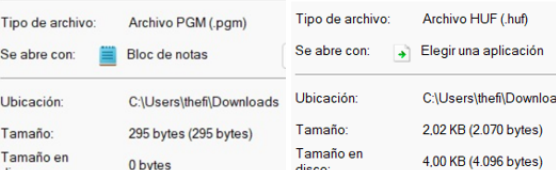


Diagrama del TAD modificado.png

## 7 Casos de Prueba Entrega 2

Descripción del caso	Resultado esperado	Resultado obtenido	Evidencia
Cargar imagen big.png (solo valor 255)	Reduccion del peso de la imagen	El peso de la imagen se redujo.	 <p>Se abre con:  Bloc de notas</p> <p>Ubicación: C:\Users\thefi\Downloads\Casos\C</p> <p>Tamaño: 3.50 MB (3.680.017 bytes)</p> <p>Tamaño en disco: 3.51 MB (3.682.304 bytes)</p> <p>Ubicación: C:\Users\thefi\Downloads</p> <p>Tamaño: 22,7 KB (23.268 bytes)</p> <p>Tamaño en disco: 24,0 KB (24.576 bytes)</p>
Cargar imagen grande con valores variados.	Reduccion del peso de la imagen	El peso de la imagen se redujo.	 <p>Ubicación: C:\Users\thefi\Downloads</p> <p>Tamaño: 1,37 MB (1.447.115 bytes)</p> <p>Tamaño en disco: 1,38 MB (1.449.984 bytes)</p> <p>Tipo de archivo: Archivo HUF (.huf)</p> <p>Se abre con:  Elegir una aplicación</p> <p>Ubicación: C:\Users\thefi\Downloads</p> <p>Tamaño: 368 KB (377.155 bytes)</p> <p>Tamaño en disco: 372 KB (380.928 bytes)</p>
Cargar imagen pequeña.	Reduccion del peso de la imagen	El peso de la imagen aumento.	 <p>Tipo de archivo: Archivo PGM (.pgm)</p> <p>Se abre con:  Bloc de notas</p> <p>Ubicación: C:\Users\thefi\Downloads</p> <p>Tamaño: 295 bytes (295 bytes)</p> <p>Tamaño en disco: 0 bytes</p> <p>Tipo de archivo: Archivo HUF (.huf)</p> <p>Se abre con:  Elegir una aplicación</p> <p>Ubicación: C:\Users\thefi\Downloads</p> <p>Tamaño: 2,02 KB (2.070 bytes)</p> <p>Tamaño en disco: 4,00 KB (4.096 bytes)</p>

Caso de prueba Codificar y Decodificar

# Corrección de Entrega 2

Para la segunda entrega, solo se presento errores con el diagrama de TAD, entonces se realizaron las correcciones correspondientes al diagrama de clases, las cuales están reflejadas en la imagen adjunta. Anteriormente, existía una relación incorrecta en la jerarquía entre las clases del sistema.

- **Relación entre Sistema y Árbol de Huffman:** En la versión anterior, se mostraba incorrectamente que el Árbol de Huffman contenía al Sistema, lo cual no representa adecuadamente la lógica del sistema. Se corrigió esta relación para reflejar que el Sistema contiene un Árbol de Huffman, ya que es el sistema quien lo utiliza para realizar tareas como compresión y descompresión de imágenes.
- **Relación entre Árbol de Huffman y Nodo:** Esta relación es correcta y se mantiene: el Árbol de Huffman está compuesto por Nodos, ya que cada nodo representa un carácter y su frecuencia, y conforman la estructura del árbol binario utilizado para la codificación Huffman.

## Tipos Abstractos de Datos (TADs)

### Diagrama de relación de TADs

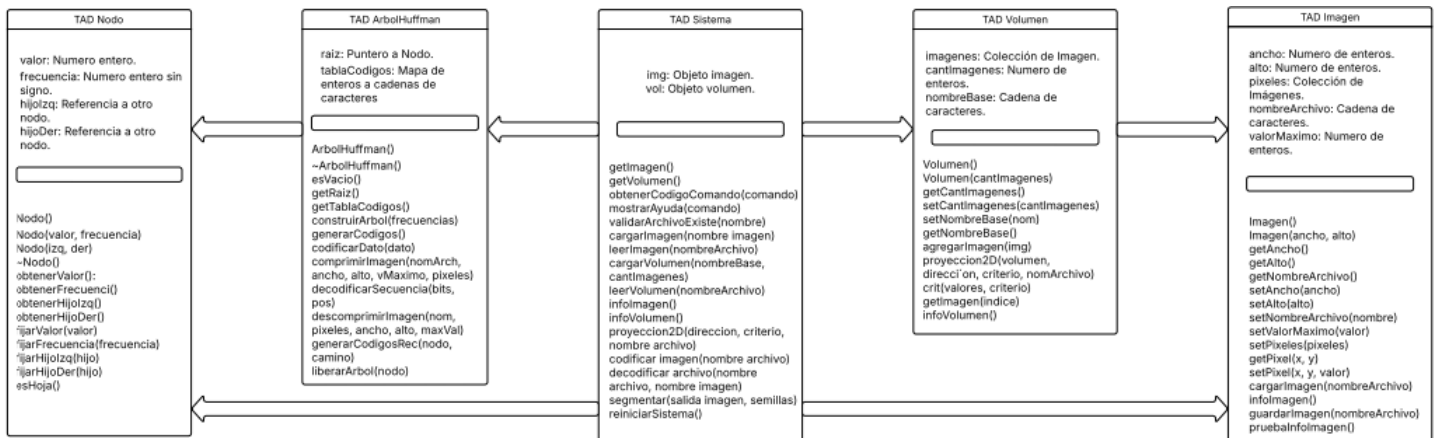


Diagrama del TAD modificado.png

# Entrega 3

## 8 Análisis del Componente 3

### 8.1 Descripción del componente

El Componente 3 del sistema se enfoca en la segmentación de imágenes en escala de grises utilizando un enfoque basado en grafos y el algoritmo de Dijkstra. Este componente permite dividir una imagen en regiones significativas a partir de semillas proporcionadas por el usuario.

### 8.2 Fundamentos teóricos

#### Representación de la imagen como grafo

- Cada píxel de la imagen se representa como un nodo en el grafo.
- Las conexiones (aristas) se establecen entre píxeles vecinos (4-conectividad: arriba, abajo, izquierda, derecha).
- El peso de cada arista se calcula como la diferencia absoluta entre los valores de intensidad de los píxeles conectados.

### 8.3 Algoritmo de segmentación

1. **Entrada de semillas:** El usuario proporciona coordenadas  $(x, y)$  y etiquetas (1-255) para cada región.
2. **Inicialización:** Cada semilla inicia una instancia del algoritmo de Dijkstra.
3. **Expansión:** Las regiones crecen según los caminos de menor costo (diferencias de intensidad).
4. **Asignación:** Cada píxel se asigna a la región cuya instancia de Dijkstra lo alcanza primero.

## 9 Descripción de Comandos - Entradas, Salidas y Condiciones

### 9.1 Comando: segmentar

- **Entrada:**
  - `salida.imagen.pgm`: Nombre del archivo de salida.
  - `sx1 sy1 l1 sx2 sy2 l2 ...`: Coordenadas y etiquetas de las semillas (máximo 5 semillas).

- **Salida:**
  - Mensaje de éxito o error según el resultado de la operación.
- **Condiciones:**
  - Debe existir una imagen cargada en memoria.
  - Las coordenadas de las semillas deben estar dentro de los límites de la imagen.
  - Las etiquetas deben ser valores entre 1 y 255.
  - El número de semillas no puede exceder 5.
- **Ejemplo correcto:** La imagen en memoria fue segmentada correctamente y almacenada en el archivo `salida_imagen.pgm`
- **Ejemplo incorrecto:**
  - No hay una imagen cargada en memoria.
  - La imagen en memoria no pudo ser segmentada.

**Proceso:**

1. Verificar precondiciones (imagen cargada, semillas válidas).
2. Construir el grafo a partir de la imagen.
3. Inicializar las semillas en el grafo.
4. Ejecutar el algoritmo de segmentación (Dijkstra multicomienzo).
5. Generar la imagen segmentada.
6. Guardar el resultado en el archivo especificado.

## 10 Tipos Abstractos de Datos (TADs)

Para la implementación del algoritmo de segmentación de imágenes, se utilizaron dos tipos abstractos de datos principales:

### 10.1 TAD Nodog

**Estado**

- **X:** Número entero.
  - Representa la coordenada horizontal del nodo en la imagen.
- **Y:** Número entero.



- Representa la coordenada horizontal del nodo en la imagen.
- **intensidad:** Número entero.
  - Almacena el valor de intensidad del píxel asociado al nodo.
- **etiqueta:** Número entero.
  - Identifica a qué región pertenece el nodo después de la segmentación.
- **costo:** Número entero largo
  - Representa el costo acumulado mínimo para llegar a este nodo desde una semilla.
- **visitado:** Valor booleano.
  - Indica si el nodo ya ha sido procesado durante el algoritmo de segmentación.

### **Interfaz**

- **NodoG():** Constructor por defecto.
  - **Post:** Crea un nodo con coordenadas (0,0), intensidad 0, etiqueta 0, costo máximo y marcado como no visitado.
- **NodoG(x\_, y\_, intensidad\_):** Constructor con parámetros.
  - **Post:** Crea un nodo con las coordenadas e intensidad especificadas, etiqueta 0, costo máximo y marcado como no visitado.
- **getX():** Operación que retorna la coordenada x del nodo.
  - **Post:** Devuelve la coordenada horizontal del nodo.
- **getY():** Operación que retorna la coordenada y del nodo.
  - **Post:** Devuelve la coordenada vertical del nodo.
- **getIntensidad():** Operación que retorna la intensidad del nodo.
  - **Post:** Devuelve el valor de intensidad del píxel asociado.
- **getEtiqueta():** Operación que retorna la etiqueta del nodo.
  - **Post:** Devuelve el identificador de la región a la que pertenece el nodo.
- **getCosto():** Operación que retorna el costo del nodo.

- **Post:** Devuelve el costo acumulado mínimo para llegar al nodo.
- **isVisitado():** Operación que verifica si el nodo ha sido visitado.
  - **Post:** Retorna true si el nodo ha sido procesado; falso en caso contrario.
- **setX(x\_):** Asigna una nueva coordenada x al nodo.
  - **Post:** La coordenada horizontal del nodo es actualizada.
- **setY(y\_):** Asigna una nueva coordenada y al nodo.
  - **Post:** La coordenada vertical del nodo es actualizada.
- **setIntensidad(intensidad\_):** Asigna una nueva intensidad al nodo.
  - **Post:** El valor de intensidad del nodo es actualizado.
- **setEtiqueta(etiqueta\_):** Asigna una nueva etiqueta al nodo.
  - **Post:** El identificador de región del nodo es actualizado.
- **setCosto(costo\_):** Asigna un nuevo costo al nodo.
  - **Post:** El costo acumulado del nodo es actualizado.
- **setVisitado(visitado\_):** Marca o desmarca el nodo como visitado.
  - **Post:** El estado de visitado del nodo es actualizado.

## 10.2 TAD Grafo

### Estado

- **ancho:** Número entero.
  - Representa el ancho de la imagen en píxeles.
- **alto:** Número entero.
  - Representa la altura de la imagen en píxeles.
- **nodos:** Matriz bidimensional de NodoG.
  - Almacena todos los nodos que representan los píxeles de la imagen, organizados según sus coordenadas.

### Interfaz

- **construir(imagen):** Construye el grafo a partir de una imagen.

- **Post:** Inicializa la matriz de nodos con las dimensiones y valores de intensidad de la imagen proporcionada.
- **segmentar(semillas):** Ejecuta el algoritmo de segmentación por crecimiento de regiones.
  - **Post:** Asigna etiquetas a todos los nodos según el algoritmo de Dijkstra modificado, utilizando las semillas proporcionadas como puntos de inicio.
- **getEtiquetas():** Operación que retorna la matriz de etiquetas resultante.
  - **Post:** Devuelve una matriz bidimensional con las etiquetas asignadas a cada píxel después de la segmentación.
- **obtenerVecinos(x, y, ancho, alto):** Operación que retorna los vecinos válidos de un píxel.
  - **Post:** Devuelve una lista de coordenadas de los píxeles vecinos (conectividad 4) que están dentro de los límites de la imagen.

### 10.3 Diagrama de relación de TADs

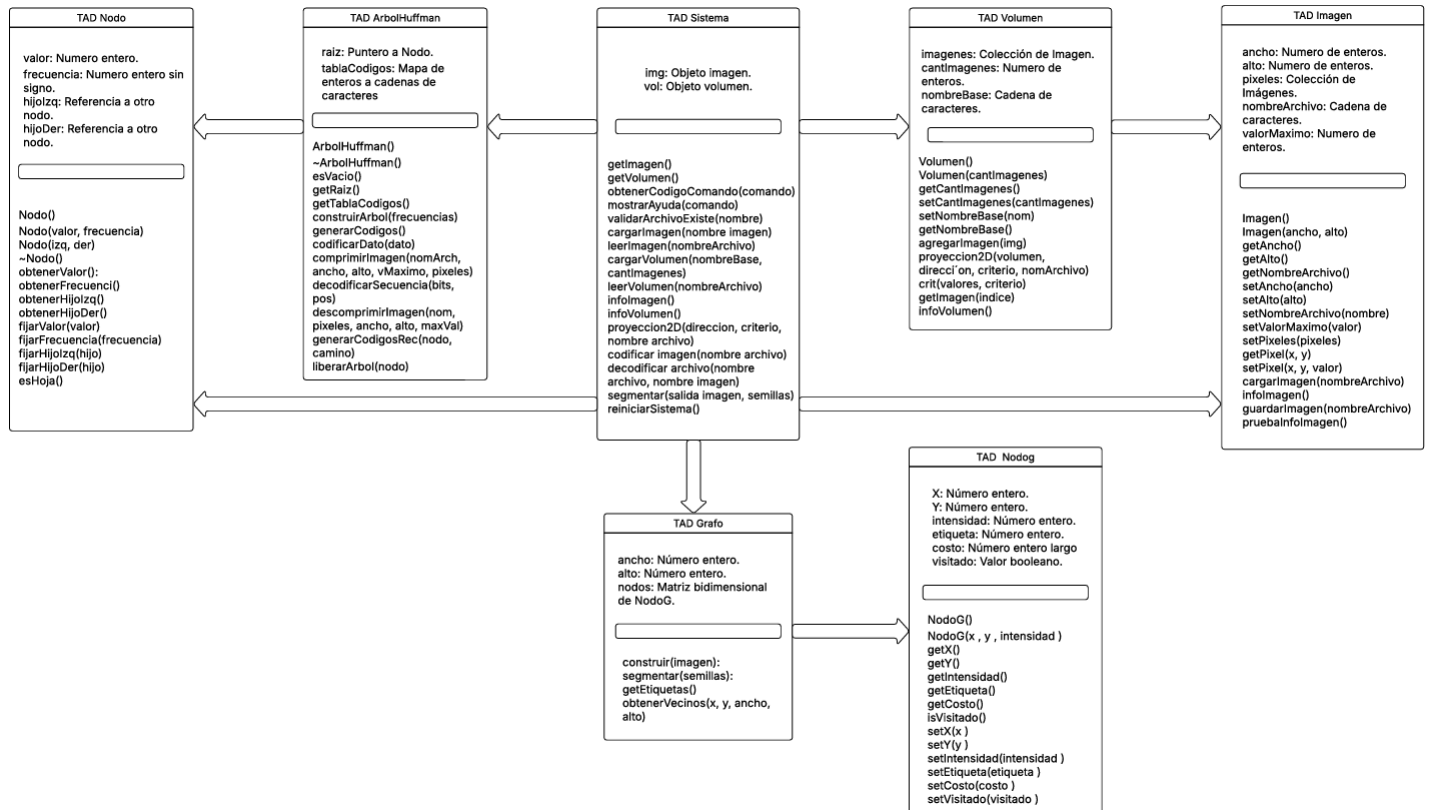
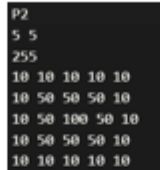
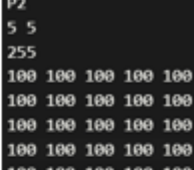
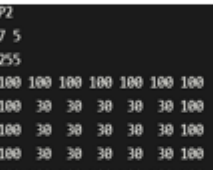
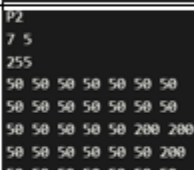
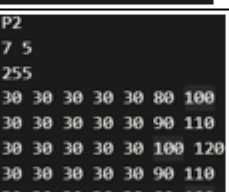
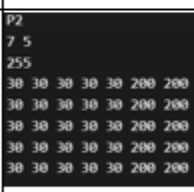
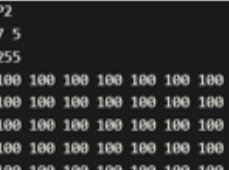
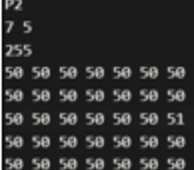


Diagrama del TAD modificado.png

## 11 Casos de Prueba Entrega 3

Plan de Pruebas -- Entrega 3 Componente				
Prueba	Descripción	Entrada	Esperado	Salida
Probar que la etiqueta se esparca por los nodos	Imagen pequeña con segmentar salida.pgm 1100		Toda la imagen debe tener etiqueta 100	
Probar que la etiqueta menor gane mas terreno	Imagen pequeña con segmentar y dos semillas (50 y 200)		El terreno con etiqueta 50 debe ser mayor a la etiqueta 200	
Probar que la etiqueta menor gane mas terreno	Imagen pequeña con segmentar y dos semillas (30 y 200)		El terreno con etiqueta 30 debe ser mayor a la etiqueta 200	
Prueba dispersion de una etiqueta menor a otra	Imagen con la misma intensidad siempre para ver como se esparce la etiqueta		El terreno con etiqueta 50 se deberia esparcir mucho mas que el de 51	

Plan de pruebas