

Pontificia Universidad Javeriana

Entrega #2 proyecto de Estructuras de Datos



Angel Eduardo Morales Abril
Tomas Alejandro Silva Correal
Juan Guillermo Pabón Vargas
Gabriel Jaramillo Cuberos
Salomon Avila

Estructuras de datos

Profesor: John Jairo Corredor Franco

Abril 24, 2025

Índice

- Evaluación componente #1
- Introducción
- Diseño y TAD
 - TAD Nodo
 - TAD Árbol
 - TAD Manejador codificación
 - TAD Imagen
 - TAD Volumen de imágenes
- Diagrama de relación
- Pasos a seguir
- Plan de pruebas
 - Primera entrega
 - Segunda entrega
- Conclusiones

Evaluación componente 1

En el componente número uno del desarrollo del proyecto no se identificaron observaciones relacionadas con la implementación en código. Por lo tanto, el único aspecto evaluativo en el que no se obtuvo una calificación perfecta fue el desarrollo del documento descriptivo del proyecto, el cual debía detallar su funcionalidad. Teniendo esto en cuenta, para el componente actual se ha mejorado significativamente la organización del contenido y se ha documentado detalladamente cada cambio y dato relevante involucrado en la realización del proyecto.

Introducción

El presente documento del diseño del componente número dos, “Codificación de imágenes” muestra la especificación y arquitectura creadas, necesarias para implementar el algoritmo de Huffman aplicado a imágenes en escala de grises. Esta entrega tiene como objetivo comprimir y descomprimir una única imagen en formato PGM, construyendo un árbol de Huffman que asigna códigos de longitud variable a cada intensidad de píxel según su frecuencia de ocurrencia en la imagen. En este documento se describirán detalladamente la planeación del código y su implementación. Además, se presentarán los Tipos Abstractos de Datos (TADs) utilizados para modelar el árbol de Huffman, siguiendo la plantilla definida en clase, y se incluirán diagramas y esquemáticos que ilustren la estructura de nodos y el flujo de bits en cada operación. Con ello, este documento servirá de guía de referencia para el desarrollo e implementación del componente, asegurando el cumplimiento de los requisitos de diseño establecidos.

Resumen

El sistema se implementará en C++ como una aplicación de consola interactiva que procesa imágenes en escala de grises en formato PGM. La entrega actual se encuentra dividida en dos componentes, cada uno con sus respectivas características y funcionalidades:

1. Proyección de imágenes

- Carga una serie de imágenes 2D que, en conjunto y en orden, conforman un volumen 3D.
- Emplea estructuras lineales (por ejemplo, `std::vector` o listas enlazadas) para almacenar los píxeles de cada imagen y del volumen completo.
- Permite generar una **proyección 2D** del volumen en la dirección x, y o z, colapsando la información de cada “rayo” de píxeles según un criterio (mínimo, máximo, promedio o mediana).
- Guarda la imagen resultante en formato PGM y muestra mensajes de éxito o error según el estado de la operación .

2. Codificación de imágenes

- Implementa el **algoritmo de Huffman** para comprimir y descomprimir una única imagen en memoria.
- Calcula la frecuencia de cada nivel de gris en la imagen cargada y construye un árbol binario de Huffman, donde las hojas representan intensidades y sus frecuencias.
- Usa esta estructura de árbol para generar códigos binarios de longitud variable: los niveles más frecuentes obtienen códigos más cortos.
- Genera un archivo .huf que contiene el ancho, alto, máximo de gris, frecuencias y la secuencia de bits codificada; y permite decodificarlo para reconstruir la imagen PGM original .

Planteamiento del problema

En las ciencias y los sistemas de computación, gran parte de la información se presenta como imágenes, las cuales, internamente, se representan mediante matrices bidimensionales de píxeles con valores de intensidad (0–255). Dos necesidades críticas al trabajar con este tipo de datos son:

1. Visualización de volúmenes 3D:

- Cómo obtener vistas 2D (proyecciones) de un conjunto de cortes 2D que conforman un objeto tridimensional, simulando procesos como la **radiografía computacional**.

- Esto exige recorrer linealmente los píxeles que forman “rayos” a través del volumen y aplicar un criterio de colapso (mínimo, máximo, promedio o mediana) para generar cada píxel de la imagen resultante.

2. Reducción del tamaño de almacenamiento:

- Las imágenes en formato PGM pueden ocupar mucho espacio. Es necesario comprimirlas sin pérdida para optimizar almacenamiento y transmisión.
- La **codificación de Huffman** aprovecha la distribución no uniforme de frecuencias de los niveles de gris, asignando códigos de longitud variable para representar cada intensidad de manera más eficiente.

Por tanto, el proyecto aborda dos problemas fundamentales de procesamiento y manejo de imágenes en C++:

- **Manipular y combinar** de forma eficaz series de imágenes para generar proyecciones 2D de volúmenes 3D
- **Comprimir y descomprimir** imágenes en escala de grises mediante estructuras de datos avanzadas (árboles de Huffman), garantizando integridad y eficiencia en las operaciones.

Propuesta de solución

Para abordar los problemas de proyección y codificación de imágenes en escala de grises, se propone desarrollar un sistema modular en C++ compuesto por:

1. Módulo de Proyección de Imágenes (Componente 1)

Este módulo permitirá generar imágenes 2D a partir de un conjunto de imágenes 2D ordenadas que representan un volumen 3D. El sistema debe:

- Leer imágenes en formato **PGM** usando estructuras como `vector<vector<int>>` para almacenar intensidades.
- Usar una estructura como `vector<vector<vector<int>>>` para representar el volumen completo.
- Implementar métodos de proyección sobre los ejes x, y y z, con criterios de colapso como mínimo, máximo, promedio o mediana.
- Almacenar la imagen proyectada en un nuevo archivo .pgm.
- Implementar una consola interactiva que reciba comandos del usuario (proyeccion2D, cargar_volumen, info_volumen, etc.).

2. Módulo de Codificación Huffman (Componente 2)

Este módulo se encargará de comprimir imágenes individuales en PGM usando el algoritmo de Huffman:

- Analizar las frecuencias de los valores de gris en la imagen.
- Construir un **árbol de Huffman** usando una cola de prioridad (heap mínimo).
- Codificar la imagen en una secuencia binaria utilizando los caminos del árbol.
- Generar un archivo .huf con cabecera estructurada y la codificación binaria.
- Incluir la opción de decodificar el archivo .huf para restaurar la imagen original en PGM.

Ambos módulos están integrados en una sola aplicación, accesible desde una línea de comandos (\$), cumpliendo con el requisito de interacción en consola.

Método de prueba

El sistema será probado a través de un **plan de pruebas funcionales** para verificar el comportamiento de cada comando, así como la integridad de los archivos generados.

Para el Componente 1:

1. **Pruebas de carga:**
 - Cargar imágenes individuales y volúmenes válidos.
 - Probar con rutas incorrectas, imágenes inexistentes y formatos inválidos.
2. **Pruebas de información:**
 - Comprobar que info_imagen e info_volumen devuelvan la información correcta.
3. **Pruebas de proyección:**
 - Probar proyeccion2D en todas las direcciones (x, y, z) con cada criterio.
 - Verificar visualmente los archivos .pgm generados o compararlos con imágenes esperadas.

Para el Componente 2:

1. **Pruebas de codificación:**
 - Codificar imágenes válidas y revisar que el archivo .huf contenga la cabecera correcta y secuencia binaria.
 - Probar errores al intentar codificar sin imagen cargada.
2. **Pruebas de decodificación:**

- Cargar archivos .huf válidos y verificar que el archivo .pgm generado sea igual al original.
- Probar errores con archivos corruptos o inexistentes.

3. Pruebas de consistencia:

- Codificar → Decodificar → Comparar: verificar que la imagen decodificada sea **idéntica** a la original.

Resultados esperados

- El sistema debe permitir la proyección correcta de volúmenes tridimensionales en 2D desde cualquier dirección y bajo cualquier criterio solicitado, generando archivos .pgm válidos.
- La codificación Huffman debe comprimir efectivamente los datos de una imagen, reduciendo el tamaño del archivo, y permitiendo su recuperación completa mediante decodificación.
- La interfaz debe ser intuitiva y robusta, permitiendo operar completamente desde la consola con mensajes claros de éxito y error.
- Se espera lograr una reducción significativa en el tamaño de los archivos con Huffman (dependiendo de la redundancia de los niveles de gris).
- Las pruebas deben demostrar que el sistema es funcional, confiable y eficiente, cumpliendo con los requisitos del curso.

TADs

Para implementar el apartado de codificación de imágenes, se utilizaron algoritmos de Huffman, los cuales se basan en estructuras de árboles. Con este fin, se diseñaron varios Tipos Abstractos de Datos (TADs). Se ha de resaltar que se han utilizado algunos Tipos Abstractos de Datos (TADs) de la entrega anterior de manera que estos serán agregados a esta entrega también debido a su relevancia para el funcionamiento del código, dentro de estos Tipos Abstractos de Datos (TADs) el único que sufrió cambios desde la entrega anterior fue el de imagen dentro del cual se agregaron dos nuevas funciones cruciales para la codificación de imágenes, “codificacion()” y “frecuenciasDeValores()”. A continuación, se presentan los TADs utilizados:

TAD Nodo

Datos mínimos:

- frecuencia, entero, representa la frecuencia del símbolo o la suma de frecuencias de los nodos hijos.
- valor, entero, representa el símbolo asociado al nodo.
- esHoja, valor booleano, indica si el nodo es una hoja.
- nodoIzquierda, nodo, apuntador al nodo hijo izquierdo.
- nodoDerecha, nodo, apuntador al nodo hijo derecho.

Operaciones:

- Nodo(), constructor que inicializa un nodo vacío.
- ~Nodo(), destructor que libera los recursos del nodo.
- getFrecuencia(), devuelve la frecuencia del nodo.
- setFrecuencia(frec), establece la frecuencia del nodo.
- getEsHoja(): devuelve el estado de si el nodo es hoja.
- setEsHoja(estado), establece si el nodo es hoja.
- getValor(), devuelve el valor asociado al nodo.
- setValor(valor), establece el valor del nodo.
- getNodeIzquierda(), devuelve el puntero al nodo hijo izquierdo.
- setNodeIzquierda(nodoIzq), establece el nodo hijo izquierdo.
- getNodeDerecha(), devuelve el puntero al nodo hijo derecho.
- setNodeDerecha(nodoDer), establece el nodo hijo derecho.

TAD Arbol:

Datos minimos:

- codificacion, vector de valores booleanos, almacena la codificación actual de un recorrido o código de Huffman.
- raíz, apuntador al nodo raíz del árbol de Huffman.
- valor, mapa, asocia valores enteros (caracteres o símbolos) con sus respectivas codificaciones en forma de vectores de booleanos.

Operaciones:

- Arbol(): Constructor que inicializa un árbol vacío.
- getCodificacion(): Devuelve el vector de codificación actual.
- setCodificacion(bool): Establece el vector de codificación.
- getRaiz(): Devuelve el puntero al nodo raíz.
- setRaiz(Nodo): Establece el nodo raíz del árbol.
- getValores(): Devuelve el mapa de valores con sus codificaciones.
- setValores(valor): Establece el mapa de valores con sus codificaciones.
- eliminar(nodo): Elimina un nodo dado del árbol.
- crearArbol(frecuencias): Construye el árbol de Huffman a partir de un vector de pares (valor, frecuencia).

- completarValores(código, raíz): Genera las codificaciones de los valores recorriendo el árbol desde un nodo dado.

TAD ManejadorCodificacion:

Datos minimos:

- Imagen, objeto de la clase Imagen, contiene la información de la imagen a procesar.
- Codificación, vector de valores booleanos, almacena la codificación resultante del proceso de Huffman.
- frecuencias, vector de pares, representa las frecuencias de los valores en la imagen.
- Árbol, objeto de la clase Árbol, representa el árbol de Huffman construido para la codificación.

Operaciones:

- Manejador(): Constructor que inicializa un manejador vacío.
- getImagen(): Devuelve el objeto imagen.
- setImagen(Imagen): Establece el objeto imagen.
- getCodificacion(): Devuelve el vector de codificación.
- setCodificacion(codificacion): Establece el vector de codificación.
- getFrecuencias(): Devuelve el vector de frecuencias.
- setFrecuencias(frec): Establece el vector de frecuencias.
- getArbol(): Devuelve el objeto árbol de Huffman.
- setArbol(arbol): Establece el objeto árbol de Huffman.
- codificar(): Realiza el proceso de codificación de la imagen utilizando el árbol de Huffman.
- cargarDesdePGM(ruta): Carga una imagen desde un archivo en formato PGM y prepara los datos para la codificación.
- cargarDesdeHUF(rutaHUF, rutaPGM): Carga una codificación desde un archivo en formato HUF para su procesamiento.
- numeroAbinario(): Convierte un número a su representación binaria

TAD Imagen:

Datos minimos:

- vecImagen, vector de vectores de enteros, almacena los valores de los píxeles de la imagen en escala de grises.
- maxClaro, entero, representa el valor máximo de claridad (brillo) permitido en la imagen (generalmente en el rango 0-255).
- dimensionX, entero, indica el ancho de la imagen en píxeles.
- dimensionY, entero, indica el alto de la imagen en píxeles.

- formato, cadena de caracteres, especifica el formato de la imagen (por ejemplo, "P2" para PGM ASCII).
- nombre, cadena de caracteres, almacena el nombre del archivo de la imagen.

Operaciones:

- Imagen(): Constructor que inicializa una imagen con valores por defecto.
- getVecImagen(): retorna el vector de vectores de vecImagen.
- setVecImagen(vecImg): asigna a vecImagen un vector de vectores de enteros.
- getMaxClaro(): retorna el valor máximo de claridad (brillo) permitido en la imagen.
- setMaxClaro(maximo): asigna al valor de "maxClaro" un valor entero que entra de parámetro.
- getDimensionX(): retorna el entero de dimensionX.
- setDimensionX(dimension): asigna al valor de "dimensionX" un valor entero que entra de parámetro.
- getDimensionY(): retorna el entero de DimensionY.
- setDimensionY(dimension): asigna al valor de "dimensionY" un valor entero que entra de parámetro.
- getFormato(): retorna el string de Formato.
- setFormato(form): asigna al valor de "Formato" un string que entra de parámetro.
- getNombre(): retorna el string de Nombre.
- setNombre(nombre2): asigna al valor de "Nombre" un string que entra de parámetro.
- cargarDesdePGM(ruta): carga imagen .pmg usando una ruta de archivo.
- guardarComoPGM(ruta): guarda imagen como .pmg a una ruta específica.
- mostrarInfo(): muestra la información relacionada a la Imagen.

TAD VolumenImágenes

Datos mínimos:

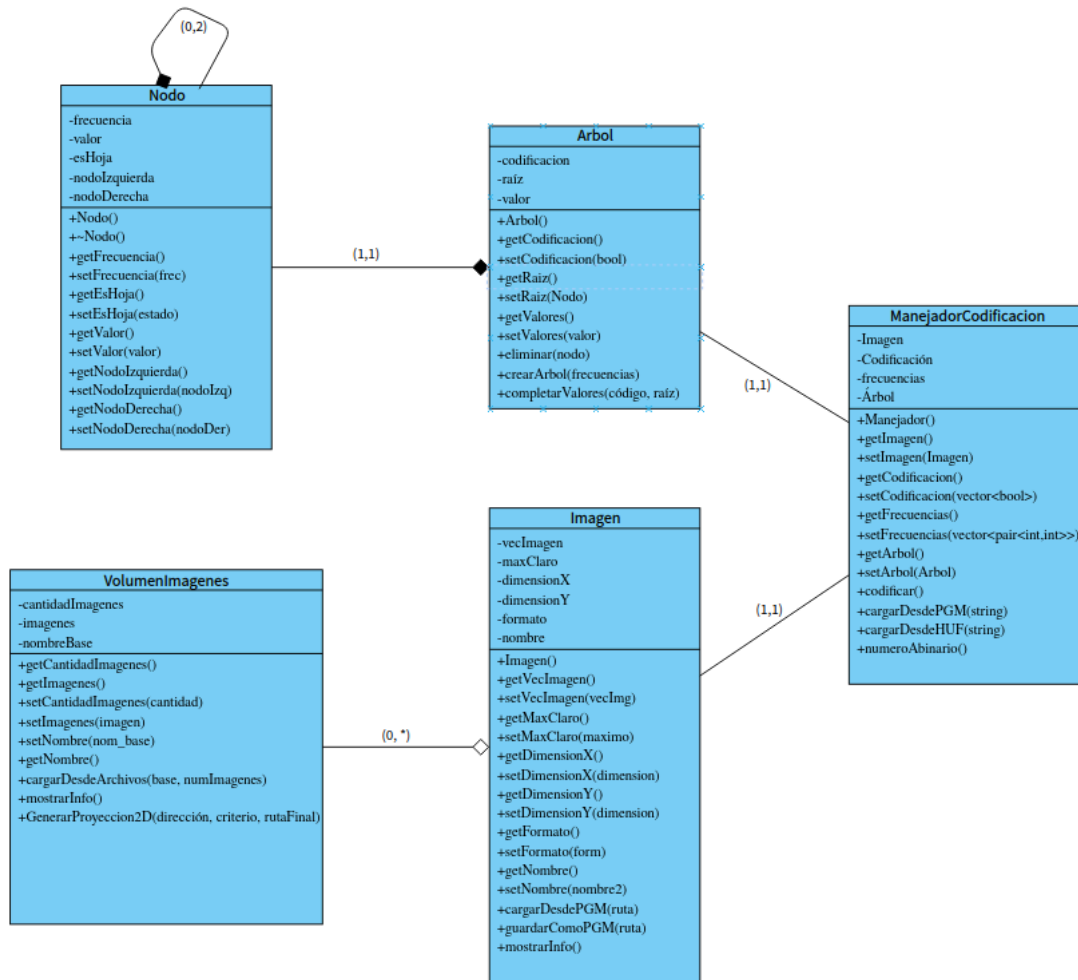
- cantidadImágenes: entero, define la cantidad de imágenes que se van a cargar
- imágenes: vector de imágenes, contiene las imágenes ordenadas según el orden de la imagen en el archivo
- nombreBase: cadena de caracteres, nombre base de la colección de imágenes

Operaciones:

- getCantidadImágenes(): retorna el número de cantidad de imágenes
- getImágenes(): retorna el vector de imágenes
- setCantidadImágenes(cantidad): asigna al valor de "cantidadImágenes" un valor entero que entra de parámetro.
- setImágenes(imagen): asigna al vector de vectores "Imágenes" un vector de mismo tipo que entra como parámetro de entrada.
- setNombre(nom_base): asigna un nombre a la colección de imágenes.
- getNombre(): retorna el nombre de la colección de imágenes.

- cargarDesdeArchivos(base, numImágenes): Carga las imágenes desde la ruta e indica la cantidad de imágenes que se cargaron.
- mostrarInfo(): Muestra la información relacionada a la base y el número de imágenes cargadas.
- GenerarProyeccion2D(dirección, criterio, rutaFinal): Se genera una proyección 2D del conjunto de imágenes dado la dirección de vista y el criterio especificado.

Diagrama de relaciones



Diseño de funciones

1) generarProyeccion2D:

- a) Tomar la serie ordenada de imágenes de la memoria, junto a la dirección (x,y o z) y el criterio (mínimo, máximo, promedio y mediana)
- b) Cargar el vector de vectores de cada imagen tras cargar el archivo pgm dentro del vector cargado en memoria.
- c) Dada la dirección y el criterio, calcular el valor del criterio (según los valores de la imagen y el criterio elegido), viajando en torno a la dirección para el cálculo correspondiente (recorriendo por los valores de vector e imágenes necesarias).
- d) Agregar los valores que cumplan con el criterio (nivel de intensidad) al vector de vectores que será la imagen resultado.
- e) Crear una nueva imagen y asignarle el vector de imágenes, junto con las dimensiones y valores correspondientes (el formato siendo el mismo que el de las imágenes cargadas).
- f) Guardar la imagen como archivo .pgm

2) Codificación:

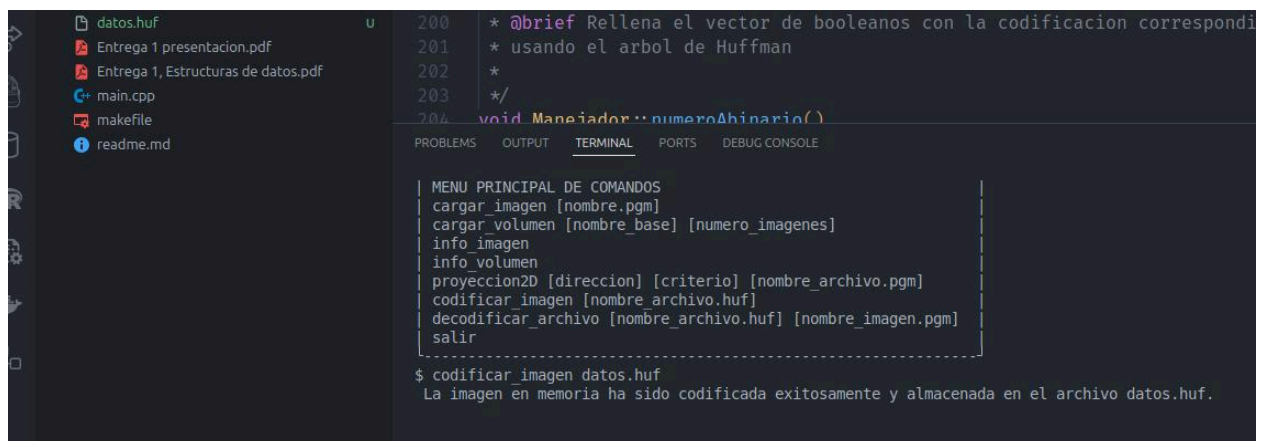
Nota: Ya debe estar cargada una imagen en el programa. En caso de que aún no lo esté, se debe cargar.

- a) Se recorre toda la matriz de la imagen contando la frecuencia de cada uno de sus elementos.
- b) Usando estas frecuencias se puede crear un árbol de la siguiente forma.
- c) Al estar implementando una codificación con árbol de Huffman, esta se debe implementar con un min-heap.
- d) Los elementos que se añaden a la cola son nodos, los cuales, si son hojas, tienen un valor; en caso contrario, solo tienen un valor de frecuencia y el resto de valores por defecto.
- e) Se extraen los dos primeros elementos de la cola y se asignan como hijos a un nuevo nodo. A la derecha irá el de mayor frecuencia y el de menor se incluirá a la izquierda.
- f) Se repite la operación hasta que solo quede un nodo en la cola, el cual representará la raíz del árbol.
- g) Usando otra estructura que nos permita acceder por una clave a un valor específico y único, vamos a recorrer todo el árbol convirtiendo los valores de las hojas en valores binarios. En el recorrido del árbol, si nos desplazamos al lado izquierdo añadimos un 0 al código, y al lado derecho un 1.
- h) Luego vamos a volver a recorrer cada uno de los valores de la imagen, pero ahora consultando el valor en la estructura de clave-valor para poder saber el equivalente de este en binario. Al saberlo, vamos a añadirlo a la estructura que nos permita guardar bits que representan la codificación.

- i) Por último, abrimos un archivo y escribimos datos binarios en el formato especificado. Al finalizar la escritura, cerramos el archivo e imprimimos un mensaje de éxito. Para garantizar una compresión real, vamos a ir empaquetando cada uno de los bits en bytes y escribiendolos una vez llenos o cuando no queden más bits.
- 3) Decodificación:
- a) Vamos a abrir la ruta del archivo binario especificada por el usuario y hacemos la correspondiente lectura, teniendo en cuenta el formato de la codificación. Adicionalmente, vamos a ir asignando los valores de las frecuencias mediante estos datos, al igual que el contenedor de bits para la codificación.
 - b) Una vez leído todo el archivo, usando las frecuencias vamos a crear un árbol tal como en el paso de la codificación.
 - c) Vamos iterando por los valores de la codificación, donde al encontrarnos con una hoja vamos a consultar su valor e incluirlo en un contenedor. Al momento de tener toda una fila de valores completa, la vamos a incluir en la matriz.
 - d) Establecemos valores propios de la imagen, como lo son el formato y la matriz de valores, y los escribimos en un archivo .pgm.

Resultados

```
| decodificar_archivo [nombre_archivo.huf] [nombre_imagen.pgm]
| salir
|-----|
$ cargar_imagen img_02.pgm
La imagen img_02.pgm ha sido cargada
```



```
200 * @brief Rellena el vector de booleanos con la codificacion correspondi
201 * usando el arbol de Huffman
202 *
203 */
204 void Manejador::numeroAbinario()

PROBLEMS OUTPUT TERMINAL PORTS DEBUG CONSOLE

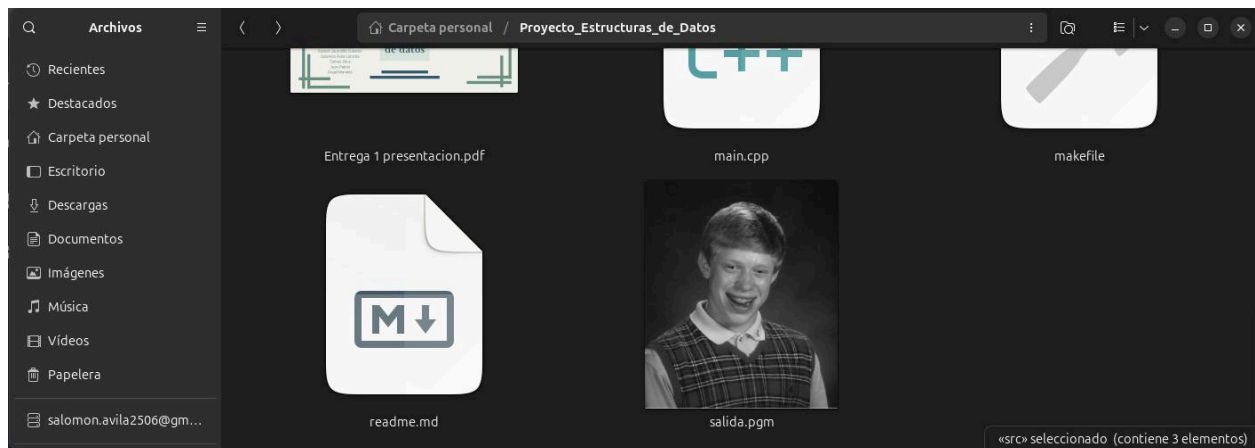
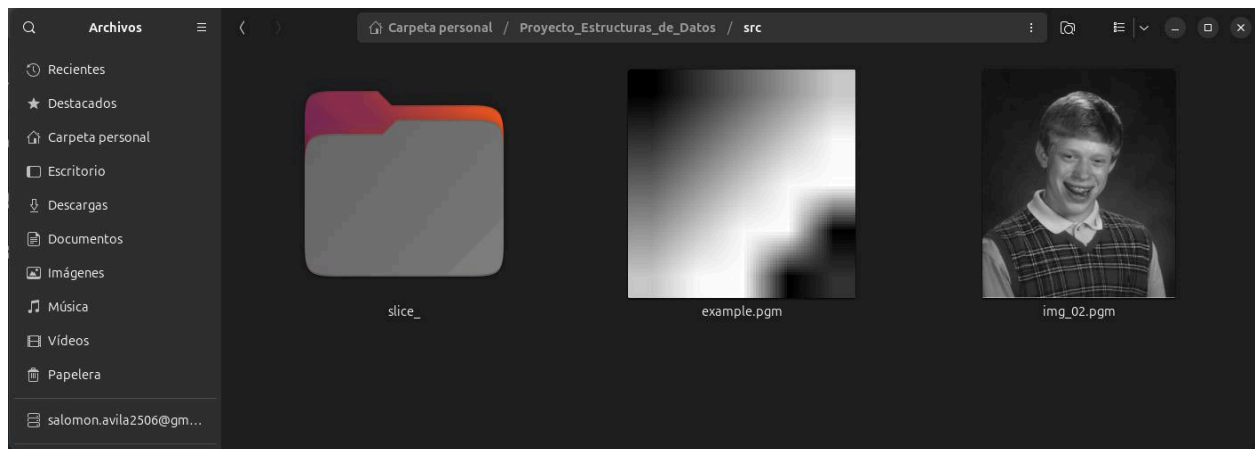
| MENU PRINCIPAL DE COMANDOS
| cargar_imagen [nombre.pgm]
| cargar_volumen [nombre_base] [numero_imagenes]
| info_imagen
| info_volumen
| proyeccion2D [direccion] [criterio] [nombre_archivo.pgm]
| codificar_imagen [nombre_archivo.huf]
| decodificar_archivo [nombre_archivo.huf] [nombre_imagen.pgm]
| salir
|-----|
$ codificar_imagen datos.huf
La imagen en memoria ha sido codificada exitosamente y almacenada en el archivo datos.huf.
```

```
datos.huf 200 * @brief Rellena el vector de booleanos con la codificación correspondiente
Entrega 1 presentacion.pdf 201 * usando el arbol de Huffman
Entrega 1, Estructuras de datos.pdf 202 *
main.cpp 203 */
makefile 204 void Manejador::numeroBinario()
readme.md PROBLEMS OUTPUT TERMINAL PORTS DEBUG CONSOLE
salida.pgm

MENU PRINCIPAL DE COMANDOS
cargar_imagen [nombre.pgm]
cargar_volumen [nombre_base] [numero_imagenes]
info_imagen
info_volumen
proyeccion2D [direccion] [criterio] [nombre_archivo.pgm]
codificar_imagen [nombre_archivo.huf]
decodificar_archivo [nombre_archivo.huf] [nombre_imagen.pgm]
salir

$ decodificar_archivo datos.huf salida.pgm
El archivo datos.huf ha sido decodificado exitosamente, y la imagen correspondiente se ha almacenado en el archivo salida.pgm.

MENU PRINCIPAL DE COMANDOS
cargar_imagen [nombre.pgm]
cargar_volumen [nombre_base] [numero_imagenes]
info_imagen
info_volumen
proyeccion2D [direccion] [criterio] [nombre_archivo.pgm]
codificar_imagen [nombre_archivo.huf]
decodificar_archivo [nombre_archivo.huf] [nombre_imagen.pgm]
```



Planes de pruebas

Primera entrega

Esta entrega del proyecto propone una solución al problema de la proyección de imágenes 3D en planos 2D, enfocándose en la simulación del proceso de captura de radiografías. Para ello, se utilizará el sistema desarrollado en la primera entrega del proyecto, ampliándolo con la definición de funcionalidades, tales como:

- Carga de imágenes desde archivos PMG.
- Carga de volúmenes de imágenes desde PMG.
- Almacenamiento y visualización de información de imágenes.
- Almacenamiento y visualización de información de volúmenes de imágenes.
- Proyección de imágenes 3D en un plano 2D.

Con el fin de garantizar el correcto funcionamiento de estas funcionalidades, se diseñará un plan de pruebas específico para cada una de ellas:

- Carga de imágenes desde archivos PMG.

Prueba	Resultado esperado	Resultado real
cargar_imagen nombre_imagen.pgm	La imagen imagen.pgm ha sido cargada.	
cargar_imagen nombre_imagen.png	La imagen imagen.pgm no ha podido ser cargada.	
cargar_imagen	Falta de argumentos.	
cargar_imagen imagen_inexistente.pgm	La imagen imagen.pgm no ha podido ser cargada.	
cargar_imagen nombre_imagen.pgm (después de haber cargado otra imagen)	La imagen imagen.pgm ha sido cargada.	

- Carga de volúmenes de imágenes desde PMG

Prueba	Resultado esperado	Resultado real
cargar_volumen volumen 10	El volumen "volumen" ha sido cargado.	
cargar_volumen volumen -5	La cantidad de imágenes no es válida.	
cargar_volumen volumen	Falta de argumentos.	
cargar_volumen volumen 200	La cantidad de imágenes supera el límite permitido.	
cargar_volumen volumen 10 (con una imagen faltante)	El volumen "volumen" no ha podido ser cargado.	
cargar_volumen volumen 10 (después de haber cargado otro volumen)	El volumen "volumen" ha sido cargado. (Sobrecribe el anterior)	

- Almacenamiento y visualización de información de imágenes.

Prueba	Resultado esperado	Resultado real
Info_imagen (con imagen cargada)	Imagen cargada en memoria: imagen.pgm, ancho: W, alto: H.	
Info_imagen (sin imagen cargada)	No hay una imagen cargada en la memoria.	

- Almacenamiento y visualización de información de volúmenes de imágenes.

Prueba	Resultado esperado	Resultado real
Info_volumen (con volumen cargado)	Volumen cargado en memoria: volumen, tamaño: n_im, ancho: W, alto: H.	
Info_volumen (sin volumen cargado)	No hay un volumen cargado en memoria.	

- Proyección de imágenes 3D en un plano 2D

Prueba	Resultado esperado	Resultado real
proyeccion2D x maximo salida.pgm (con volumen cargado)	La proyección 2D del volumen en memoria ha sido generada y almacenada en el archivo salida.pgm.	
proyeccion2D a maximo salida.pgm	El volumen aún no ha sido cargado en memoria.	
proyeccion2D z promedio salida.pgm (sin volumen cargado)	Dirección no válida. Las opciones son: x, y, z.	
proyeccion2D x suma salida.pgm	Criterio no válido. Las opciones son: minimo, maximo, promedio, mediana.	
proyeccion2D x maximo	Falta de argumentos.	

Segunda entrega

Esta entrega del proyecto propone una solución al problema de la función de decodificación de archivos .huf a imágenes .pgm

Para ello, se realizará la función “cargarDesdePGM” en un manejador correspondiente para la codificación, donde se implementan funciones de otras clases como lo son “imagen”, “árbol” entre otros.

Con el fin de garantizar el correcto funcionamiento de estas funcionalidades, se diseñará un plan de pruebas específico para los dos escenarios principales mostrados a continuación:

- Carga exitosa y decodificación de archivo **HUF**, junto a la creación de archivo pgm resultado.

Prueba	Resultado esperado	Resultado real
Cargar una imagen y decodificarla	La imagen fue cargada en memoria y se ha decodificado en un archivo .huf que se almacena en el disco	
Prueba de decodificación (similitud entre imágenes)	La imagen resultado de la decodificación es igual a la imagen original que se codificó	

Prueba de decodificación (tamaño entre imágenes)	El peso de la imagen generada tras la decodificación es igual a la imagen original	
--	--	--

- Carga fallida de archivo **HUF**, por diversos motivos

Prueba	Resultado esperado	Resultado real
Nombre de archivo erróneo o no existencia del archivo	Se envía una notificación sobre la falla en el nombre	
Formato erroneo	Se envía una notificación sobre la falla en el formato para codificación	

Conclusiones

- Integración de conocimientos teóricos y prácticos: el proyecto hizo posible la aplicación de los conceptos de estructuras de datos y algoritmos aprendidos, mostrando cómo la teoría de las estructuras y los tipos abstractos de datos se convierten en soluciones prácticas para el procesamiento de imágenes.
- Modularidad y escalabilidad: la división del sistema en componentes muestra la importancia del diseño de soluciones haciendo uso de módulos, facilitando el mantenimiento y la ampliación del sistema, además de la creación de librerías para el acceso a métodos y funciones de forma eficiente.
- Eficiencia en el manejo de datos: la implementación del sistema para el procesamiento de imágenes necesitó utilizar estructuras de datos, tanto para almacenar grandes volúmenes de información como para realizar operaciones complejas.

- Cambios respecto a la entrega anterior: pese a ideas originales, y por temas de complejidad y tiempo, se optó por eliminar la estructura o modelo MVC (modelo vista-controlador), con tal de permitir correcciones más sencillas y modificaciones ágiles para el desarrollo del programa.
- Importancia de la interacción entre el usuario y el sistema: el desarrollo de una interfaz de línea de comandos muestra cómo una interacción clara y amigable puede mejorar la experiencia del usuario al permitirle ejecutar y comprender los diferentes procesos de transformación y análisis de imágenes.