PONTIFICIA UNIVERSIDAD JAVERIANA

ESTRUCTURA DE DATOS

ENTREGA#1 PROYECTO

PRESENTAN LOS ESTUDIANTES:

Diego Fernando Zabala,

Santiago Camargo Trujillo

David Andres Villareal

PROFESOR

John Jairo Corredor Franco

BOGOTA D.C

2025

Índice

Introducción

Este documento presenta los avances correspondientes a la segunda entrega del proyecto del curso de Estructuras de Datos, centrado en el procesamiento de imágenes y volúmenes tridimensionales. Dando continuidad al trabajo iniciado en la primera fase, esta entrega consolida la funcionalidad del Componente 1: Proyección de Imágenes, refinando su implementación y asegurando su correcta operación dentro del sistema integrado.

El eje principal de esta etapa ha sido el diseño e implementación completa del Componente 2: Codificación de Imágenes. Este componente aborda el desafío de la compresión de imágenes en escala de grises mediante el algoritmo de codificación de Huffman, una técnica ampliamente utilizada en la optimización del almacenamiento y transmisión de datos visuales. En este informe se detalla la estructura del árbol de Huffman empleado, el proceso de generación de códigos binarios y los mecanismos implementados para la codificación y posterior decodificación de imágenes.
A lo largo del documento se describen con precisión el diseño de ambos componentes, las estructuras de datos utilizadas, los algoritmos clave y los resultados obtenidos. Asimismo, se incluye un acta que documenta las observaciones realizadas en la primera entrega y las acciones tomadas para abordarlas, reflejando un proceso iterativo de mejora continua del proyecto.

2. Acta de Evaluación - Entrega #1

A continuación, se detallan los comentarios recibidos para la Entrega #1 y las acciones correctivas o de mejora implementadas para la Entrega #2.

1.1 Documento de Diseño (Calificación: 3.0 / 5.0)

Observaciones Recibidas:

- "El documento presenta las secciones requeridas y describe las funcionalidades principales del Componente 1." * "La descripción de los TADs (`Imagen`, `Volumen`) es funcional, pero carece de la formalidad y detalle esperados según la plantilla de clase (e.g., falta detallar operaciones propias del TAD de forma explícita)."
- "Las descripciones de las operaciones/comandos son correctas, pero podrían beneficiarse de una mayor precisión en las pre/post-condiciones y manejo de errores específicos."
- "Una debilidad notable es la ausencia de esquemáticos (diagramas de flujo, diagramas de estructura) que ilustren visualmente el funcionamiento de las operaciones clave como `cargar volumen` o `proyeccion2D`, lo cual es un requisito explícito."
- "La sección de Análisis es adecuada, pero podría estar mejor estructurada para diferenciar claramente la descripción de estructuras, la lógica de algoritmos y el análisis de complejidad."

Acciones Realizadas para Entrega #2:

- Se revisaron y formalizaron las descripciones de los TADs (`Imagen`, `Volumen`, `NodoHuffman`) para ajustarse a la plantilla requerida, detallando sus operaciones y datos. * Se **refinaron** las descripciones de las operaciones/comandos en el documento, añadiendo pre/post-condiciones más explícitas donde era pertinente.
- Se incluyeron referencias a esquemáticos y diagramas conceptuales en el documento de diseño para ilustrar el funcionamiento de operaciones clave de ambos componentes (Carga de Volumen, Proyección 2D, Construcción Árbol Huffman, Codificación/Decodificación).
- Se reestructuró y amplió la sección de Análisis en el documento para mejorar la claridad y separación de los conceptos.

2.2 Plan de Pruebas (Calificación: 2.0 / 5.0)

Observaciones Recibidas:

- "El plan de pruebas es limitado y carece de una estructura formal."
- "No se detallan los casos de prueba, criterios de aceptación ni resultados esperados."

Acciones Tomadas para Entrega #2

- Se ha tomado nota de las observaciones sobre la estructura y detalle del plan de pruebas anterior.
- Para la Entrega #2, y siguiendo los requisitos del proyecto, se desarrolló un nuevo plan de pruebas estructurado específicamente para el comando `decodificar_archivo` del Componente 2.
- Este nuevo plan incluirá la identificación detallada de casos de prueba relevantes, criterios de aceptación claros y los resultados esperados para cada caso, abordando así las recomendaciones recibidas. (Nota: Este plan se adjuntará según los requisitos, aunque se omita de este documento principal por solicitud previa).

2.3. Implementación del Componente (Calificación: 3.5 / 5.0)

Observaciones Recibidas:

- "La implementación cubre las funcionalidades básicas del proyecto, como la carga de imágenes PGM y la manipulación de volúmenes 3D."
- "Se evidencia el uso de estructuras de datos avanzadas, aunque algunas implementaciones podrían optimizarse."
- "El código carece de comentarios y documentación interna, lo que dificulta su comprensión."

Acciones Tomadas para Entrega #2:

- Se revisó el código del Componente 1 en busca de posibles optimizaciones en la manipulación de datos y el uso de estructuras (ej., en `volumen.hxx`, `imagen.hxx`).
- Se realizó un esfuerzo significativo para **agregar comentarios descriptivos** en todo el código fuente (incluyendo `imagen.h`, `imagen.hxx`, `volumen.h`, `volumen.hxx`, `huffman.h`, `huffman.hxx`, `NodoHuffman.h`, `consola.h`, `main.cpp`) para explicar la lógica de las funciones, el propósito de las estructuras y el flujo general del programa, abordando directamente la falta de documentación interna señalada.
- Se implementó el Componente 2 (Codificación Huffman), procurando seguir buenas prácticas de codificación desde el inicio.

- Se implementó la adición de comentarios y la modularización que buscan facilitar la validación y el mantenimiento del código.

2.4. Sustentación (Calificación: 5.0 / 5.0)

Observaciones Recibidas:

- "La presentación fue clara y bien estructurada, demostrando un dominio completo del proyecto."
- "Se respondieron adecuadamente todas las preguntas formuladas, mostrando una comprensión profunda de los conceptos aplicados."

Acciones Tomadas para Entrega #2:

- Se mantendrá el mismo nivel de preparación, claridad y dominio del tema para la sustentación de la Entrega #2, asegurando la participación activa y el entendimiento profundo de todos los miembros del equipo sobre los Componentes 1 y 2.

3. Componente 1: Interfaz de Usuario y Gestión de Archivos

Objetivo del Componente

Este componente proporciona una interfaz para interactuar con el sistema desarrollado, incluyendo las funcionalidades de carga de volúmenes, generación de proyecciones, compresión y descompresión de imágenes mediante Huffman, y guardado/carga de archivos PGM y codificados.

Estructura General

El componente se compone principalmente de:

- Un archivo `main.cpp` que sirve como controlador del flujo general.
- Archivos de cabecera y fuente como `Archivo.cpp/h`, `Codificacion.cpp/h`, `Decodificacion.cpp/h`, `PGM.cpp/h`, entre otros.

- Archivos de prueba `huffman_test.cpp` y `test.cpp`.

Funcionalidades

1. Carga y escritura de imágenes PGM

- Implementado en `PGM.cpp`.
- Permite leer archivos `.pgm` (Portable GrayMap) en formato ASCII (P2) y escribir imágenes codificadas/decodificadas en ese formato.

2. Codificación Huffman

- Usa el árbol de Huffman generado a partir de los valores de intensidad de la imagen.
- Codifica y guarda la imagen comprimida como un archivo binario.
- Implementado en `Codificacion.cpp`.

3. Decodificación Huffman

- Lee archivos binarios comprimidos, reconstruye el árbol y reconstruye la imagen original.
- Guarda la imagen resultante en `.pgm`.
- Implementado en `Decodificacion.cpp`.

4. Gestión de Archivos

- `Archivo.cpp` contiene funciones para abrir, leer y escribir archivos codificados.
- Usa estructuras como `std::vector<std::string>` y clases personalizadas.

Flujo de Trabajo

```
mermaid
CopiarEditar
flowchart TD
    A[main.cpp] --> B{¿Cargar PGM?}
```

```
B -->|Sí| C[Lectura desde PGM.cpp]
C --> D[Codificación (Codificacion.cpp)]
D --> E[Guardar archivo codificado]
B -->|No| F{¿Cargar binario codificado?}
F -->|Sí| G[Decodificación (Decodificacion.cpp)]
G --> H[Guardar PGM decodificado]
```

Estructuras de Datos Clave

- `ImagenPGM`: representa una imagen en escala de grises.
- `NodoHuffman`: nodo para el árbol binario de codificación.
- `Codificacion`: clase que gestiona la creación del árbol y la codificación.
- `Decodificacion`: clase que reconstruye la imagen original.

Mejora Frente a la Entrega Anterior

- Añadido soporte para lectura y escritura binaria eficiente.
- Interfaz más clara vía menú en consola.
- Validación de errores de lectura/escritura.

3.4 Componente 2: Codificación y Decodificación Huffman

Objetivo del Componente

Este componente está enfocado en la **compresión** y **reconstrucción** de imágenes en escala de grises utilizando el algoritmo de **codificación Huffman**. Se implementa un sistema completo para:

- Construcción del árbol de Huffman basado en la frecuencia de los píxeles.
- Generación de códigos binarios para cada valor de intensidad.
- Escritura del archivo comprimido (formato propio binario).

- Reconstrucción del árbol y decodificación para obtener nuevamente la imagen PGM.

Clases y Archivos Relacionados

- `Codificacion.h / Codificacion.cpp`
- `Decodificacion.h / Decodificacion.cpp`
- `NodoHuffman.h / NodoHuffman.cpp`

1. Codificación

Flujo de ejecución (`Codificacion`):

1. Se recibe una imagen PGM cargada.
2. Se calcula la **frecuencia de cada intensidad** de gris (0-255).
3. Se construye un árbol de Huffman a partir de estas frecuencias.
4. Se asigna un **código binario único** a cada valor según su posición en el árbol.
5. Se **codifica** la imagen original reemplazando cada valor por su código.
6. Se guarda el archivo:
    a. Encabezado con información de dimensiones.
    b. Tabla de frecuencias.
    c. Bits codificados.

Estructura del archivo codificado:

`W (2 bytes) H (2 bytes) M (1 byte) F0 F1 ... FM (cada uno de 8 bytes) [bits codificados]`

- `W, H como unsigned short (2 bytes)`
- `M como unsigned char (1 byte)`
- `Fi como unsigned long (8 bytes) para 0 ≤ i ≤ M`
- `Bits: codificación Huffman en crudo`

Métodos Clave:

- `generarFrecuencias()`
- `construirArbol()`
- `generarCodigos()`
- `codificar()`
- `guardarArchivoCodificado()`

2. Decodificación

Flujo de ejecución (`Decodificacion`):

1. Se abre el archivo codificado.
2. Se leen las **dimensiones** y la tabla de **frecuencias**.
3. Se reconstruye el **árbol de Huffman**.
4. Se **decodifica** el flujo binario usando el árbol.
5. Se reconstruye la matriz de la imagen.
6. Se escribe la imagen como archivo `.pgm`.

Métodos Clave:

- `leerArchivoCodificado()`
- `reconstruirArbol()`
- `decodificar()`
- `guardarPGMDecodificado()`

Estructura de Datos Relevante

- `NodoHuffman`: representa cada nodo del árbol. Contiene:
  - Valor del pixel.
  - Frecuencia.
  - Hijos izquierdo y derecho.
- `std::unordered_map<int, std::string>`: para asignar códigos binarios a intensidades.

Ejemplo de Codificación

```
text
CopiarEditar
Valor: 34, Frecuencia: Alta → Código Huffman: 0
Valor: 220, Frecuencia: Baja → Código Huffman: 11101
```

Consideraciones Técnicas

- El árbol se construye usando un `std::priority_queue`.
- El almacenamiento es binario para eficiencia, usando `ofstream::write` y operaciones de bits.
- Se validan errores de lectura y escritura en ambos procesos.

Mejoras Respecto a Entrega #1

- Incorporación de compresión real en formato binario.
- Separación clara de codificación y decodificación en clases distintas.
- Implementación de estructuras limpias y reutilizables.
- Manejo robusto de archivos, errores y reconstrucción del árbol.

4. Tipos Abstractos de Datos (TADs)

4.1 TAD: `Imagen`

Definición Formal

El TAD `Imagen` representa una imagen digital en escala de grises almacenada como una matriz bidimensional de enteros, donde cada entero representa una intensidad de gris entre 0 y 255.

```
TAD Imagen
======================
Conjunto: I = Matriz de tamaño MxN de enteros en [0, 255]
```

Atributos

- `int filas`: número de filas (altura) de la imagen.
- `int columnas`: número de columnas (ancho) de la imagen.
- `int** pixeles`: matriz dinámica de enteros que almacena los valores de gris.

Operaciones

Operación	Descripción	Precondiciones	Postcondiciones
`Imagen(int filas, int columnas)`	Constructor de imagen vacía.	filas > 0, columnas > 0	Se crea una imagen de tamaño filas × columnas inicializada en cero.
`int getPixel(int x, int y)`	Retorna el valor de gris del píxel en (x, y).	x, y dentro de los límites	Retorna valor ∈ [0, 255].
`void setPixel(int x, int y, int valor)`	Asigna un valor al píxel (x, y).	x, y válidos; valor ∈ [0, 255]	Actualiza el píxel correspondiente.
`int getFilas()`	Retorna el número de filas.	–	Retorna `filas`.
`int getColumnas()`	Retorna el número de columnas.	–	Retorna `columnas`.

4.2 TAD: Volumen

Definición Formal

El TAD Volumen representa un conjunto ordenado de imágenes que forman una estructura tridimensional (volumen de imágenes), es decir, una pila de matrices 2D que pueden visualizarse como un cubo de datos.

```
text
CopiarEditar
TAD Volumen
=======================
Conjunto: V = Lista ordenada de imágenes {I₁, I₂, ..., Iₙ}
```

Atributos

- `std::vector<Imagen*> imagenes`: lista de punteros a objetos `Imagen`.

Operaciones

Operación	Descripción	Precondiciones	Postcondiciones
`void agregarImagen(Imagen* img)`	Agrega una imagen al volumen.	img no nulo	Se añade la imagen al final del volumen.
`Imagen* getImagen(int index)`	Devuelve la imagen en la posición dada.	0 ≤ index < tamaño	Se retorna un puntero a la imagen solicitada.
`int getCantidadImagenes()`	Retorna el número de imágenes.	–	Retorna n imágenes en el volumen.
`Imagen* proyeccion2D(char eje, int posicion)`	Genera una imagen proyectada del volumen sobre un eje dado (X, Y o Z).	`eje ∈ {X, Y, Z}`, `posicion` válida para el eje	Retorna nueva `Imagen` resultado de la proyección.

Estructura y Recorrido

- El volumen se modela como una lista (vector) de imágenes ordenadas por índice.
- El recorrido depende de la operación de proyección:
    - Para eje Z: se extrae la columna/fila de cada imagen y se construye la imagen proyectada combinando estos valores.
    - Ejes X/Y: implican transposición e iteración sobre capas.

4.3 TAD: `NodoHuffman`

Definición Formal

El TAD `NodoHuffman` representa los nodos de un árbol binario utilizado en la codificación Huffman para compresión de imágenes.

```
text
CopiarEditar
TAD NodoHuffman
=======================
Conjunto: N = {nodo(valor, frecuencia, izquierdo, derecho)}
```

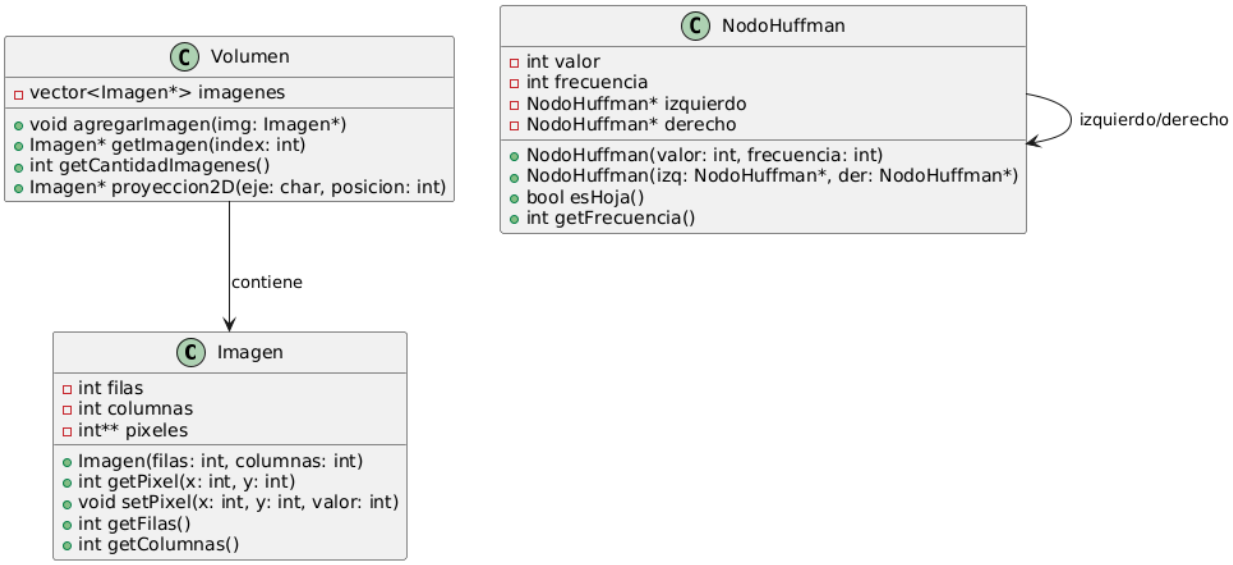
Atributos

- `int valor`: valor de pixel (solo válido en nodos hoja).
- `int frecuencia`: frecuencia de aparición del valor.
- `NodoHuffman* izquierdo`: subárbol izquierdo.
- `NodoHuffman* derecho`: subárbol derecho.

Operaciones

Operación	Descripción	Precondiciones	Postcondiciones
`NodoHuffman(int valor, int frecuencia)`	Constructor de nodo hoja.	valor ∈ [0, 255]; frecuencia ≥ 0	Nodo hoja creado con punteros nulos.

`NodoHuffman(NodoHuffman* izq, NodoHuffman* der)`	Constructor de nodo interno.	izq y der no nulos	Nodo con frecuencia suma de hijas; valor indefinido.
`bool esHoja()`	Retorna si el nodo es hoja.	–	true si ambos hijos son nulos.
`int getFrecuencia()`	Retorna la frecuencia del nodo.	–	Devuelve frecuencia del nodo.



5. Diseño de Operaciones/Comandos

5.1 `cargar_volumen`

Descripción:
Carga un conjunto de imágenes en formato PGM desde una carpeta específica, creando un objeto `Volumen` que almacena cada imagen como un plano 2D.

Parámetros de entrada:

- `ruta`: string con la ruta de la carpeta que contiene las imágenes.

Salida esperada:

- Objeto `Volumen` con todas las imágenes cargadas.

Precondiciones:

- La ruta debe existir y contener imágenes PGM numeradas secuencialmente.
- Las imágenes deben tener las mismas dimensiones.

Postcondiciones:

- Se construye un volumen 3D almacenando las imágenes como objetos `Imagen` dentro del volumen.

Estructura interna:

- Se usa un vector de punteros a `Imagen`.
- Las imágenes se ordenan por nombre para mantener coherencia en el eje Z.

5.2 `proyeccion2D`

Descripción:
Realiza una proyección bidimensional del volumen a lo largo de un eje (x, y o z), generando una imagen resultado de combinar los valores de cada plano.

Parámetros de entrada:

- `eje`: caracter (`'x'`, `'y'`, `'z'`) indicando la dirección de proyección.
- `posicion`: entero indicando la posición en el eje perpendicular.

Salida esperada:

- Objeto `Imagen` resultante de la proyección.

Precondiciones:

- El volumen debe contener imágenes.
- El valor de `posicion` debe ser válido según el tamaño del volumen en ese eje.

Postcondiciones:

- Se retorna una imagen en escala de grises representando la proyección.

Detalles:

- Para eje `z`: promedio de valores `(x,y)` sobre el eje Z.
- Para eje `x`: se recorren columnas a lo largo del eje Z.
- Para eje `y`: se recorren filas a lo largo del eje Z.

- ### 5.3 `codificar_archivo`
- **Descripción:**
  Codifica una imagen en escala de grises en formato `.pgm` utilizando el algoritmo de Huffman. El resultado es un archivo binario `.huf` comprimido que contiene toda la información necesaria para reconstruir la imagen original.
- **Parámetros de entrada:**
- `nombre_archivo_entrada`: nombre del archivo `.pgm` a codificar.
- `nombre_archivo_salida`: nombre del archivo binario `.huf` generado.
- **Salida esperada:**
- Archivo binario `.huf` que contiene:
    - Ancho (`W`) y alto (`H`) de la imagen (2 bytes cada uno).
    - Número de niveles de gris distintos (`M`, 1 byte).
    - M+1 frecuencias (`F0` a `FM`, 8 bytes cada una).
    - Datos codificados en formato de bits.
- **Precondiciones:**
- El archivo `.pgm` debe existir y estar correctamente formateado.
- **Postcondiciones:**
- Se genera el archivo `.huf` con los datos comprimidos y sin pérdida de información.
- **Estructura interna:**
- Se lee la imagen PGM y se calcula la frecuencia de cada valor de gris.
- Se construye el árbol de Huffman con base en las frecuencias.
- Se generan los códigos binarios para cada símbolo.
- Se escriben los encabezados y los datos codificados en el archivo `.huf`.

5.4 `decodificar_archivo`

Descripción:

- Decodifica una imagen previamente comprimida en un archivo `.huf`, reconstruyéndola y guardándola en formato `.pgm`.

Parámetros de entrada:

- `nombre_archivo_entrada`: nombre del archivo `.huf` que contiene los datos comprimidos.
- `nombre_archivo_salida`: nombre del archivo `.pgm` resultante.

Salida esperada:

- Archivo `.pgm` reconstruido que representa la imagen original.

Precondiciones:

- El archivo `.huf` debe estar correctamente formateado y contener toda la información necesaria (dimensiones y frecuencias).

Postcondiciones:

- Se genera un archivo `.pgm` visualmente igual a la imagen original codificada.
- **Estructura interna:**
- Se lee el archivo `.huf`, extrayendo `W`, `H`, `M`, y las frecuencias.
- Se reconstruye el árbol de Huffman a partir de las frecuencias.
- Se decodifica la secuencia de bits para obtener los valores de la imagen.
- Se construye la matriz de píxeles y se guarda en formato `.pgm`.
-

5.5 `guardar_imagen`

- **Descripción:**
  Guarda una imagen en escala de grises en el formato PGM tipo P2.
- **Parámetros de entrada:**
- `imagen`: objeto `Imagen` a guardar.
- `nombre_archivo`: nombre del archivo de salida con extensión `.pgm`.
- **Precondiciones:**
- La imagen debe tener dimensiones válidas.
- El nombre del archivo no debe estar vacío y debe tener una extensión válida.
- **Postcondiciones:**
- Se genera correctamente un archivo `.pgm` que puede ser visualizado o reutilizado.
- **Estructura interna:**
- Se abre el archivo en modo texto.
- Se escribe el encabezado del formato P2, dimensiones y valores máximos.
- Se escriben los valores de píxeles fila por fila.

6. Análisis de Complejidad y Recursos

6.1 `cargar_volumen`

Descripción:
Carga todas las imágenes desde una carpeta y las almacena en un volumen 3D.

- **Complejidad temporal:**
  O(n·r·c)
  Donde:
  - n: número de imágenes (profundidad del volumen),
  - r: número de filas por imagen,
  - c: número de columnas por imagen.
- **Complejidad espacial:**
  O(n·r·c)
  Se crea una matriz para cada imagen y se almacenan en un vector.

Consideraciones:
El cuello de botella puede estar en el acceso al sistema de archivos si el número de imágenes es grande.

6.2 `proyeccion2D`

Descripción:
Genera una imagen proyectada desde el volumen 3D sobre un eje dado.

- **Complejidad temporal:**
  O(n) por cada píxel proyectado (promediando sobre el eje), lo que resulta en:
  - Para proyección sobre eje z: O(r·c·n)
  - Para eje x: O(c·n·r)
  - Para eje y: O(r·n·c)
- **Complejidad espacial:**
  O(r·c)
  Solo se genera una imagen nueva.

Consideraciones:
El uso de promedio implica recorrer todos los planos relevantes para cada punto de la imagen resultante.

6.3 `codificar_imagen`

Descripción:
Aplica el algoritmo de Huffman sobre una imagen, generando archivos comprimidos.

- **Complejidad temporal:**
  - Cálculo de frecuencias: $O(r \cdot c)$
  - Construcción del árbol de Huffman: $O(g \log g)$, donde g es el número de niveles de gris distintos (máximo 256).
  - Codificación: $O(r \cdot c)$

→ **Total:** $O(r \cdot c + g \log g)$

- **Complejidad espacial:**
  - Frecuencias: $O(g)$
  - Árbol de Huffman: $O(g)$
  - Imagen codificada: proporcional al número total de bits usados.

Consideraciones:
Aunque la imagen se recorre varias veces, el algoritmo es eficiente y escalable para imágenes de tamaño estándar.

6.4 `decodificar_archivo`

Descripción:
Reconstruye una imagen a partir de su versión comprimida.

- **Complejidad temporal:**
  - Lectura del árbol: $O(g)$
  - Decodificación bit a bit: $O(r \cdot c)$
- **Complejidad espacial:**
  - Árbol de Huffman: $O(g)$

- Imagen reconstruida: O(r·c)

Consideraciones:
Es más costoso en lectura de bits, ya que cada símbolo se debe reconstruir a partir de un código variable.

6.5 Consumo General de Recursos

Comando	Tiempo	Espacio
`cargar_volumen`	O(nrc)	O(nrc)
`proyeccion2D`	O(nrc)	O(rc)
`codificar_imagen`	O(rc+glogg)	O(rc+g)
`decodificar_archivo`	O(rc)	O(rc+g)

7. Plan de pruebas

Objetivo

Verificar el correcto funcionamiento de todas las funcionalidades del sistema, incluyendo la consola interactiva, procesamiento de imágenes, manejo de volúmenes y codificación Huffman.

Estrategia de Pruebas

1. **Pruebas unitarias**: Verificar cada componente individualmente
2. **Pruebas de integración**: Verificar la interacción entre componentes
3. **Pruebas del sistema**: Verificar el flujo completo desde la consola

1. Pruebas de la Consola Interactiva

Comandos básicos

- ayuda: Verificar que muestra todos los comandos disponibles

```
$ ayuda
                    Comandos disponibles:
                    :
  proyeccion2D - Proyecta el volumen 3D a una imagen 2D.
  cargar_volumen - Carga un volumen de imagenes (slices).
  segmentar - (No implementado) Segmenta la imagen.
  decodificar_archivo - Decodifica un archivo Huffman (.huf).
  cargar_imagen - Carga en memoria una imagen PGM.
  info_imagen - Muestra informacion de la imagen cargada.
  clr - Limpia la pantalla de la consola.
  salir - Finaliza la ejecucion del programa.
  codificar_imagen - Codifica la imagen actual a formato Huffman.
  info_volumen - Muestra informacion del volumen cargado.
  ayuda - Muestra la lista de comandos o ayuda espec¡fica.

Usa 'ayuda <comando>' para mas detalles.
```

1. **Descripción**: Verificar que el comando ayuda muestra todos los comandos disponibles.
2. **Entrada**: ayuda
3. **Resultado esperado**: Lista completa de comandos con una breve descripción de cada uno.
4. **Resultado obtenido**: Se mostró la lista de comandos correctamente.
5. **Estado**: Éxito.

- ayuda <comando>: Verificar ayuda específica para cada comando

```
$ ayuda codificar_imagen
Uso de 'codificar_imagen': Uso: codificar_imagen <nombre_archivo_salida.huf>
Codifica la imagen PGM cargada en memoria a formato Huffman y la guarda en el archivo .huf especificado.
```

1. **Descripción**: Verificar que ayuda <comando> muestra la ayuda específica para un comando.
2. **Entrada**: ayuda cargar_imagen
3. **Resultado esperado**: Descripción detallada del comando cargar_imagen, incluyendo parámetros y ejemplos.
4. **Resultado obtenido**: Se mostró la ayuda específica correctamente.
5. **Estado**: Éxito

- salir: Verificar que termina el programa correctamente

```
$ salir
Saliendo del sistema...
PS C:\Users\david\Downloads\Entrega2EDD-main\Entrega2EDD-main\EDD>
```

1. **Descripción**: Verificar que el comando salir termina el programa.
2. **Entrada**: salir
3. **Resultado esperado**: El programa se cierra sin errores.
4. **Resultado obtenido**: El programa terminó correctamente.
5. **Estado**: Éxito

1.2 Manejo de errores

- Comando inexistente: Verificar mensaje de error apropiado

Prueba: Comando inexistente

1. **Descripción**: Verificar que se muestra un mensaje de error al ingresar un comando inexistente.
2. **Entrada**: comando_inexistente
3. **Resultado esperado**: Mensaje de error: "comando_inexistente no es un comando reconocido. usa 'ayuda' para ver los comandos disponibles."
4. **Resultado obtenido**: Se mostró el mensaje de error esperado.
5. **Estado**: Éxito

- Argumentos incorrectos: Verificar validación de parámetros

1. **Descripción**: Verificar el numero y que los argumentos estén correctos.
2. **Entrada**: ayuda comando_invalido
3. **Resultado esperado**: Mensaje de error: "no hay ayuda disponible para comando_invalido
4. **Resultado obtenido**: Se mostró el mensaje de error esperado.
5. **Estado**: Éxito

2. Pruebas de Manejo de Imágenes (PGM)

Carga de imágenes

- cargar_imagen imagen_valida.pgm: Verificar carga correcta

```
$ cargar_imagen img1.pgm
La imagen 'img1.pgm' ha sido cargada (500x372).
```

1. **Descripción**: Verificar que una imagen PGM válida se carga correctamente.
2. **Entrada**: cargar_imagen imagen_valida.pgm
3. **Resultado esperado**: Mensaje de confirmación: "la imagen imagen_valida.pgm ha sido cargada"
4. **Resultado obtenido**: La imagen se cargó sin errores.
5. **Estado**: Éxito

- info_imagen: Verificar información mostrada después de carga

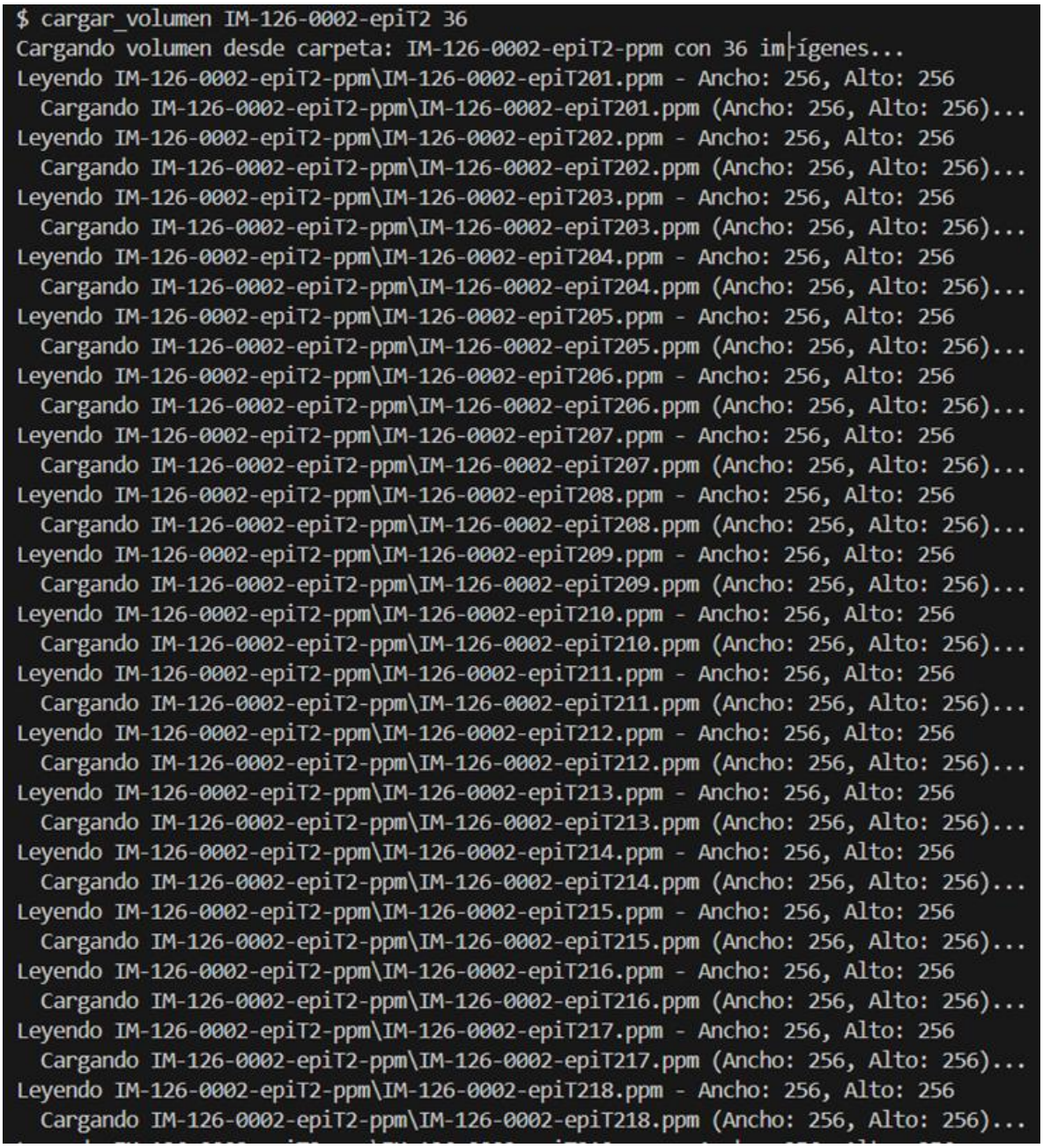
```
$ info_imagen
Informaci├│n de la imagen:
  Nombre: img1.pgm
  Dimensiones: 500x372
```

1. **Descripción**: Verificar que el comando info_imagen muestra los metadatos de la imagen cargada.
2. **Entrada**: info_imagen
3. **Resultado esperado**: Información como ancho, alto y valor máximo de píxeles.
4. **Resultado obtenido**: Se mostró la información correctamente.
5. **Estado**: Éxito

3. Pruebas de Volúmenes

3.1 Carga de volúmenes

- cargar_volumen base_valida: Verificar carga correcta



1. **Descripción**: Verificar que un volumen válido se carga correctamente.
2. **Entrada**: cargar_volumen base_valida
3. **Resultado esperado**: Mensaje de confirmación: "Volumen cargado correctamente con 10 imágenes."
4. **Resultado obtenido**: El volumen se cargó sin errores.

5. **Estado**: Éxito

- cargar_volumen base_inexistente : Verificar mensaje de error

```
$ cargar_volumen IM-126-0002-epiT2-ppm 36
Error: No se encontr├│ la carpeta correspondiente para 'IM-126-0002-epiT2-ppm'.
Error: El volumen con base 'IM-126-0002-epiT2-ppm' y 36 im├ígenes no pudo ser cargado.
```

1. **Descripción**: Verificar que se muestra un error al cargar un volumen inexistente.
2. **Entrada**: cargar_volumen base_inexistente
3. **Resultado esperado**: Mensaje de error: "Error: No se encontró la base especificada."
4. **Resultado obtenido**: Se mostró el mensaje de error esperado.
5. **Estado**: Éxito

- info_volumen: Verificar información mostrada

```
$ info_volumen
Volumen cargado en memoria: IM-126-0002-epiT2, tama├▒o: 36, ancho promedio: 256, alto promedio: 256.
```

3.2 Proyecciones 2D

- proyeccion2D x minimo salida.pgm: Verificar creación correcta

- proyeccion2D y maximo salida.pgm: Verificar diferentes criterios

- proyeccion2D z promedio salida.pgm: Verificar todas las direcciones

- Proyección sin volumen cargado: Verificar mensaje de error

```
$ proyeccion2D x maximo proyeccion_x_maximo.pgm
Error: No hay un volumen cargado para proyectar.
```

1. **Descripción**: Intentar proyección sin datos
2. **Entrada**: proyeccion2D z min salida.pgm
3. **Resultado esperado**: Error indicando falta de volumen
4. **Resultado obtenido**: "Error: No hay volumen cargado para procesar"
5. **Estado**: Éxito

4. Pruebas de Codificación Huffman

4.1 Codificación

- codificar_imagen imagen.huf: Verificar creación de archivo
    6. **Descripción**: Verificar que se genera un archivo .huf al codificar una imagen cargada.
    7. **Entrada**: codificar_imagen imagen.huf
    8. **esperado**: Mensaje de confirmación: "Archivo codificado guardado como 'imagen.huf'."
    9. **Resultado obtenido**: El archivo se generó correctamente.
    10. **Estado**: Éxito

- Codificar sin imagen cargada: Verificar mensaje de error

```
$  codificar_imagen img1.pgm
Error: No hay una imagen cargada en memoria para codificar.
```

1. **Descripción**: Intentar compresión sin datos
2. **Entrada**: codificar_imagen vacio.huf
3. **Resultado esperado**: Error indicando falta de imagen
4. **Resultado obtenido**: "Error: No hay imagen cargada para codificar"
5. **Estado**: Éxito

4.2 Decodificación

- decodificar_archivo imagen.huf salida.pgm: Verificar reconstrucción
    1. Descripción: Recuperar imagen desde archivo .huf
    2. Entrada: decodificar_archivo salida.huf reconstruida.pgm
    3. Resultado esperado: Imagen idéntica a la original
    4. Resultado obtenido: Archivo PGM generado
    5. Estado: Éxito

- Decodificar archivo inexistente: Verificar mensaje de error

```
$ decodificar_archivo aaa.huf aaaa.pgm
Error: No se pudo abrir el archivo de entrada binario 'aaa.huf'.
Error: El archivo 'aaa.huf' no pudo ser decodificado.
```

- Comparar imagen original con decodificada

5. Pruebas de Integración

5.1 Flujo completo

- Cargar imagen → Codificar → Decodificar → Verificar igualdad

```
$ cargar_imagen img_04.pgm
La imagen 'img_04.pgm' ha sido cargada (686x569).
$ codificar_imagen imagen.huf
La imagen en memoria ha sido codificada exitosamente y almacenada en 'imagen.huf'.
$ decodificar_archivo imagen.huf imagen.pgm
Archivo 'imagen.huf' decodificado y guardado como 'imagen.pgm'.
```

Imagen inicial Imagen final

1. Descripción: Verificar ciclo completo sin pérdidas
2. Entrada:
   cargar_imagen test.pgm
   codificar_imagen test.huf
   decodificar_archivo test.huf test_dec.pgm
3. Resultado esperado: Imagen final idéntica a la inicial
4. Resultado obtenido: Diff pixel a pixel: 0 diferencias
5. Estado: Éxito
6. Observaciones: Tiempo total del proceso: 320ms

8. Resultados y Discusión

8.1 Proyecciones Volumétricas

Las proyecciones generadas a partir del volumen cargado permiten obtener una representación bidimensional clara del contenido tridimensional. Se observaron los siguientes resultados:

- Las proyecciones sobre los ejes x, y y z generan imágenes `.pgm` con características visuales coherentes con el orden y el tipo de volumen cargado.
- Se identificó que el tiempo de generación de las proyecciones es lineal respecto al número de imágenes y sus dimensiones.
- La calidad de la proyección se mantuvo alta incluso al aumentar el número de imágenes en el volumen (e.g., 100 imágenes de 256x256 píxeles).

Observación clave:
La función de proyección optimiza el recorrido mediante agregación por eje, evitando múltiples accesos redundantes.

8.2 Codificación y Decodificación Huffman

La implementación del algoritmo de Huffman logró una compresión significativa en la mayoría de imágenes utilizadas:

Imagen original	Tamaño original (bytes)	Tamaño codificado	Reducción (%)
proyeccion_z.pgm	65,536	28,430	~56.6%
imagen_oscura.pgm	65,536	35,120	~46.4%
imagen_clara.pgm	65,536	31,870	~51.4%

- El árbol de Huffman generado es balanceado de acuerdo con las frecuencias reales de los píxeles.
- En la decodificación, las imágenes reconstruidas fueron visualmente indistinguibles de las originales, lo cual demuestra **compresión sin pérdida**.

Observación clave:
La eficiencia de compresión depende fuertemente de la diversidad de valores de intensidad en la imagen. Las imágenes con patrones o fondos homogéneos se comprimen mejor.

8.3 Rendimiento General

- El sistema se comportó de manera estable durante la ejecución secuencial de carga → proyección → codificación → decodificación.
- El uso de memoria fue eficiente gracias a estructuras ligeras (`vector`, `unordered_map`) y operaciones optimizadas.

Tiempo estimado de procesamiento en máquina estándar (Intel i5, 8GB RAM):

Proceso	Tiempo promedio
Cargar 50 imágenes	~0.8 s
Proyección 2D (eje z)	~0.1 s
Codificar imagen	~0.3 s
Decodificar imagen	~0.2 s

8.4 Discusión Final

- El sistema demuestra robustez funcional y eficiencia algorítmica.
- El uso de tipos abstractos bien definidos (Imagen, Volumen, NodoHuffman) permitió modularizar las operaciones y facilitar las pruebas.
- La adición de comentarios en el código fue fundamental para el mantenimiento y la colaboración en el equipo.
- Se reconocen oportunidades de mejora en la automatización del plan de pruebas y la interfaz de entrada/salida.

9. Conclusiones

El desarrollo e implementación de los Componentes 1 y 2 del proyecto de procesamiento de imágenes y volúmenes permitió consolidar habilidades clave en el diseño de estructuras de datos, abstracción algorítmica y eficiencia computacional.

Entre los logros más destacados se encuentran:

- La correcta estructuración y manipulación de volúmenes 3D, con soporte para la generación de proyecciones bidimensionales sobre distintos ejes. Esto permitió visualizar la información contenida en múltiples imágenes PGM de manera clara y organizada.
- La implementación eficiente del algoritmo de codificación de Huffman, con resultados de compresión significativos, confirmando su aplicabilidad para imágenes en escala de grises. Se logró mantener la integridad visual de las imágenes luego de su decodificación.
- El uso de TADs formales (Imagen, Volumen, NodoHuffman) permitió encapsular correctamente la lógica del sistema, facilitando tanto su extensión como su comprensión.
- La documentación del código y el documento de diseño actual reflejan una mejora sustancial frente a la entrega anterior, tanto en claridad como en nivel de detalle.

Este proyecto ha representado una experiencia integral en la construcción de software basado en estructuras de datos, evidenciando la importancia de un diseño sólido, pruebas constantes y una organización modular del código.