Documentación Segunda Entrega Proy

Pontificia Universidad Javeriana

Estructuras De Datos

Autores:

Santiago Hernandez

Juan Esteban Bello

Esteban Navas

John Corredor

27 de marzo 2025

Acta de evaluación de la entrega anterior

Comentarios Textuales del Profesor:

1. *"El plan de pruebas no es suficientemente claro ni específico. No se evidencia un análisis completo de funcionalidades ni la inclusión de pruebas de escritorio que muestren cómo se comporta el sistema."*
2. *"Aunque el programa funciona correctamente, sería importante mejorar la interfaz de usuario. Desde el inicio del programa debería mostrarse un mensaje de bienvenida más claro. Asimismo, el comando de ayuda debería ser más intuitivo y explícito sobre cómo se deben utilizar los comandos."*
3. *"Eviten el uso de términos en inglés como 'exit', ya que deben mantener coherencia idiomática en todo el sistema."*

Acciones Realizadas para la Corrección:

1. **Mejora del Plan de Pruebas:**
    a. Se reestructuró completamente el plan de pruebas del Componente 1.
    b. Se incluyeron pruebas de escritorio detalladas, con entradas y salidas esperadas, cubriendo casos normales, casos límite y errores.
    c. Se agregaron comentarios explicativos y capturas del funcionamiento real del sistema.
2. **Interfaz de Usuario:**
    a. Se incluyó un mensaje de bienvenida amigable y claro al iniciar el sistema.
    b. Se rediseñó el comando ayuda para que sea más intuitivo: muestra una lista clara de comandos disponibles y la forma correcta de utilizarlos.
    c. Todos los mensajes en pantalla ahora están redactados completamente en español, eliminando cualquier palabra en inglés innecesaria (por ejemplo, se reemplazó exit por salir).

1. OBJETIVOS

1.1. Objetivo General

Diseñar e implementar un sistema de procesamiento de imágenes en escala de grises que permita codificar y decodificar imágenes utilizando el algoritmo de Huffman, asegurando una compresión eficiente y una reconstrucción fiel de la información original, mediante estructuras de datos vistas durante el curso de Estructuras de Datos.

1.2. Objetivos Específicos

- Aplicar el algoritmo de codificación de Huffman para reducir el tamaño de las imágenes PGM, utilizando árboles binarios como estructura principal.
- Implementar un proceso de decodificación que permita reconstruir imágenes a partir de archivos comprimidos con Huffman, garantizando la integridad de la imagen original.
- Estructurar el sistema a través de una interfaz de consola que permita al usuario ejecutar comandos de forma clara, guiada y en español.
- Mejorar la experiencia de usuario incluyendo mensajes de bienvenida, ayudas detalladas y retroalimentación intuitiva ante errores o procesos exitosos.
- Validar el funcionamiento del sistema mediante un plan de pruebas que incluya casos correctos, casos límite y pruebas de errores, enfocándose especialmente en el comando decodificar_archivo.

2. Introducción

Se ha desarrollado un sistema modular para el procesamiento de imágenes en escala de grises, basado en comandos ejecutados desde una consola interactiva. En la primera entrega del proyecto se implementó exitosamente el componente 1, enfocado en la generación de proyecciones 2D a partir de un volumen de imágenes. Esta funcionalidad permitió simular el proceso de captura de radiografías mediante análisis lineal sobre estructuras lineales.

La presente entrega da continuidad al sistema con el desarrollo del componente 2, cuya finalidad es implementar un mecanismo de compresión y descompresión de imágenes utilizando el algoritmo de huffman, un método eficiente y óptimo basado en árboles binarios. Esta técnica permite representar cada símbolo (en este caso, las intensidades de los pixeles) con una cantidad variable de bits, dependiendo de su frecuencia de aparición, logrando así una representación comprimida más eficiente en términos de almacenamiento.

Además de la incorporación de esta nueva funcionalidad, se han realizado mejoras significativas con base en la retroalimentación recibida en la entrega anterior. Se rediseñó el plan de pruebas, haciéndolo más claro y completo, y se ajustó la interfaz del sistema para ofrecer una mejor experiencia de usuario: mensajes de inicio más amigables, ayudas de comandos más intuitivas y coherencia en el idioma de los textos.

Esta entrega incluye el desarrollo completo del componente 2, junto con el mejoramiento del componente 1, un análisis técnico del código implementado, los tads utilizados, esquemáticos explicativos, un plan de pruebas específico para el comando decodificar_archivo y las conclusiones generales del proceso.

3. DESARROLLO

3.1. Retroalimentación de la Entrega Anterior

Durante la primera entrega del proyecto, el sistema presentado cumplía con los requerimientos funcionales establecidos para el Componente 1. Sin embargo, el profesor nos hizo varias observaciones importantes que permitieron mejorar tanto la calidad del código como la documentación y la experiencia de usuario.

Uno de los aspectos señalados fue la elaboración del plan de pruebas. Aunque se incluían ejemplos, estos no evidenciaban completamente el comportamiento del sistema. Faltaba claridad en la estructura de las pruebas, y no se incluyeron pruebas de escritorio que permitieran seguir paso a paso el funcionamiento del programa. Esta retroalimentación nos llevó a replantear el plan de pruebas, incluyendo casos específicos, detallando

entradas y salidas esperadas, y cubriendo tanto escenarios válidos como situaciones de error.

En cuanto al funcionamiento del sistema, el profesor indicó que no era necesario hacer ajustes significativos al código, ya que los comandos implementados en el Componente 1 trabajaban correctamente. Sin embargo, sí se nos sugirió mejorar la interfaz de usuario. Se recomendó iniciar el programa con un mensaje de bienvenida más claro y explicativo, así como ofrecer una ayuda más intuitiva al momento de consultar los comandos disponibles. También se hizo énfasis en evitar el uso de palabras en inglés dentro de la interfaz, como exit, para mantener la coherencia del idioma a lo largo del sistema.

Con base en estos comentarios, realizamos los siguientes ajustes:

- Se agregó un mensaje de bienvenida al iniciar el sistema, que orienta al usuario sobre su propósito.
- El comando de ayuda fue rediseñado para mostrar de forma clara y ordenada todos los comandos disponibles, acompañados de ejemplos de uso.
- Se reemplazaron todos los términos en inglés presentes en la interfaz por su equivalente en español, asegurando uniformidad en los mensajes.

Estas mejoras han sido fundamentales para fortalecer el proyecto en esta segunda entrega, tanto desde el punto de vista técnico como desde la interacción con el usuario.

3.2. Análisis del Código

Para esta segunda entrega, el sistema fue ampliado con la funcionalidad de compresión y descompresión de imágenes usando el algoritmo de Huffman. Esta técnica permite representar los valores de intensidad de una imagen utilizando códigos binarios de longitud variable, en función de la frecuencia con la que aparecen. Esto contribuye a reducir el tamaño del archivo sin perder calidad en la imagen reconstruida.

El código está organizado en módulos que separan claramente las responsabilidades, lo cual facilita su mantenimiento y comprensión. Cada archivo fuente cumple un rol específico dentro del sistema:

main.cpp

Controla el flujo principal del programa. Aquí se procesan los comandos ingresados por el usuario, se validan sus parámetros y se llama a las funciones correspondientes. También se gestionan los mensajes que se muestran en pantalla, incluyendo los de bienvenida, ayuda y resultado de cada operación.

imagen.h y imagen.hxx

Definen el TAD asociado a la imagen. Contienen las estructuras y funciones para almacenar y manipular los datos de una imagen PGM, incluyendo su nombre, dimensiones, valor máximo de intensidad y la matriz de pixeles. También permiten leer una imagen desde disco y escribirla nuevamente en formato PGM.

huffman.h y huffman.hxx

Implementan la lógica del algoritmo de Huffman. Aquí se define la estructura del árbol de codificación, se calculan las frecuencias de los pixeles, se construye el árbol, y se generan los códigos binarios para cada intensidad. Además, se incluye el proceso inverso para decodificar un archivo comprimido y recuperar la imagen original.

proyeccion.h y proyeccion.hxx

Estos archivos, que ya formaban parte del sistema desde la entrega anterior, permiten generar proyecciones 2D a partir de volúmenes de imágenes. Aunque no son parte del Componente 2, siguen estando disponibles dentro del sistema y no interfieren con la nueva funcionalidad.

En general, el sistema conserva su estructura modular y funcional. La interacción con el usuario se realiza completamente desde consola, sin dependencias externas, lo que asegura la compatibilidad con distintos entornos y facilita las pruebas del sistema.

Flujo general de ejecución

El flujo básico del programa puede resumirse en los siguientes pasos:

1. **Inicio del sistema**

a. Se muestra el mensaje de bienvenida.
b. El sistema queda a la espera de comandos.

2. **Ingreso de comandos por el usuario**
    a. El usuario puede ingresar un comando relacionado con carga de imagen, codificación, decodificación, consulta de ayuda o salida.
3. **Procesamiento de cada comando**
    a. El sistema valida el comando y sus parámetros.
    b. Dependiendo del comando, se ejecuta la acción correspondiente:
        i. Cargar una imagen (se lee y almacena en memoria).
        ii. Codificar una imagen (se genera el archivo .huf).
        iii. Decodificar un archivo (se reconstruye una imagen .pgm).
        iv. Mostrar información de ayuda.
        v. Finalizar la ejecución.
4. **Resultados en pantalla**
    a. Se muestra un mensaje indicando si el proceso fue exitoso o si ocurrió algún error.
5. **Ciclo de espera**
    a. El programa regresa al estado de espera para recibir un nuevo comando, hasta que el usuario decida salir.

3.3. TADs del Programa

TAD Imagen

Datos mínimos:

- nombre: cadena de caracteres
  Representa el nombre o identificador de la imagen, utilizado para referenciarla en el sistema.
- ancho: número entero
  Indica la cantidad de columnas de la imagen, es decir, su dimensión horizontal.
- alto: número entero
  Indica la cantidad de filas de la imagen, es decir, su dimensión vertical.

- pixeles: matriz de números enteros
  Almacena los valores de intensidad de gris para cada píxel de la imagen, en una estructura bidimensional.

Operaciones:

- obtenerNombre(): devuelve el nombre actual de la imagen.
- fijarNombre(): asigna un nuevo nombre a la imagen.
- obtenerAncho(): devuelve la cantidad de columnas.
- obtenerAlto(): devuelve la cantidad de filas.
- obtenerPixel(): devuelve el valor del píxel en una posición específica.
- fijarPixel(): modifica el valor del píxel en una posición específica.
- cargarDesdeArchivo(): carga los datos de la imagen desde un archivo en formato adecuado.
- guardarEnArchivo(): guarda la imagen actual en un archivo.

TAD Volumen

Datos mínimos:

- nombre: cadena de caracteres
  Representa el nombre o identificador del volumen, útil para diferenciarlo de otros volúmenes cargados o creados.
- imagenes: lista de referencias a TAD Imagen
  Contiene la secuencia ordenada de imágenes que componen el volumen tridimensional. Cada imagen representa una capa en el eje de profundidad.
- proyecciones: lista de referencias a TAD Proyección
  Almacena las proyecciones generadas a partir del volumen, cada una según un criterio y dirección determinados.

Operaciones:

- obtenerNombre(): devuelve el nombre del volumen.
- fijarNombre(): cambia el nombre del volumen.

- agregarImagen(): añade una nueva imagen al volumen.
- obtenerImagen(): devuelve una imagen específica del volumen.
- cantidadImagenes(): devuelve cuántas imágenes contiene el volumen.
- agregarProyeccion(): añade una nueva proyección al volumen.
- obtenerProyeccion(): devuelve una proyección específica almacenada en el volumen.
- cantidadProyecciones(): devuelve cuántas proyecciones han sido generadas y almacenadas.

TAD Proyección

Datos mínimos:

- nombre: cadena de caracteres
  Identifica la proyección generada, permitiendo diferenciarla de otras proyecciones dentro del sistema.
- direccion: cadena de caracteres
  Indica el plano sobre el cual se realiza la proyección, como XY, YZ o XZ.
- criterio: cadena de caracteres
  Especifica la forma en que se calcula la proyección, por ejemplo: mínimo, máximo, promedio o mediana.
- resultado: referencia a un TAD Imagen
  Representa la imagen generada como resultado de aplicar la proyección al volumen, utilizando el criterio y dirección especificados.
- volumen: referencia a un TAD Volumen
  Indica el volumen desde el cual se ha generado la proyección.

Operaciones:

- obtenerNombre(): devuelve el nombre de la proyección.
- fijarNombre(): cambia el nombre de la proyección.
- obtenerDireccion(): devuelve la dirección en que se proyectó el volumen.
- fijarDireccion(): cambia la dirección de proyección.
- obtenerCriterio(): devuelve el criterio usado para generar la proyección.
- fijarCriterio(): cambia el criterio de proyección.

- obtenerResultado(): devuelve la imagen resultante de la proyección.
- fijarResultado(): asigna la imagen generada como resultado de la proyección.
- obtenerVolumen(): devuelve el volumen asociado a la proyección.
- fijarVolumen(): asigna el volumen del cual se genera la proyección.

TAD NodoHuffman

Datos mínimos:

- valor: número entero
  Representa el nivel de gris correspondiente a un símbolo en la imagen. En nodos internos puede estar ausente o no tener significado.
- frecuencia: número entero
  Indica cuántas veces aparece el valor asociado en la imagen original. En nodos internos representa la suma de las frecuencias de sus hijos.
- izquierdo: referencia a otro TAD NodoHuffman
  Apunta al subárbol izquierdo del nodo actual en el árbol de Huffman.
- derecho: referencia a otro TAD NodoHuffman
  Apunta al subárbol derecho del nodo actual en el árbol de Huffman.

Operaciones:

- obtenerValor(): devuelve el valor de gris del nodo.
- fijarValor(): asigna el valor de gris al nodo.
- obtenerFrecuencia(): devuelve la frecuencia del nodo.
- fijarFrecuencia(): asigna la frecuencia del nodo.
- obtenerIzquierdo(): devuelve la referencia al hijo izquierdo.
- fijarIzquierdo(): asigna la referencia al hijo izquierdo.
- obtenerDerecho(): devuelve la referencia al hijo derecho.
- fijarDerecho(): asigna la referencia al hijo derecho.
- esHoja(): indica si el nodo no tiene hijos, es decir, si es una hoja del árbol.

TAD ÁrbolHuffman

Datos mínimos:

- raiz: referencia a un TAD NodoHuffman
  Es el nodo principal del árbol, desde el cual se pueden recorrer todos los caminos que conducen a los códigos binarios de cada símbolo.
- frecuencias: lista de números enteros
  Almacena la cantidad de apariciones de cada nivel de gris en la imagen. Sirve como base para construir el árbol.
- codigos: lista de cadenas de caracteres
  Guarda los códigos binarios generados para cada nivel de gris, donde cada posición representa un símbolo.

Operaciones:

- construirDesdeFrecuencias(): genera el árbol de Huffman a partir de una lista de frecuencias.
- obtenerCodigo(): devuelve el código binario asociado a un valor específico de gris.
- obtenerRaiz(): devuelve la referencia al nodo raíz del árbol.
- codificarImagen(): recibe una imagen y devuelve la secuencia de bits correspondiente.
- decodificarBits(): reconstruye una imagen original a partir de una secuencia de bits comprimidos.
- limpiar(): libera la memoria ocupada por los nodos del árbol.

TAD ImagenCodificada

Datos mínimos:

- ancho: número entero
  Indica cuántas columnas tenía la imagen original antes de ser comprimida.
- alto: número entero
  Indica cuántas filas tenía la imagen original antes de la compresión.
- frecuencias: lista de números enteros
  Almacena cuántas veces aparece cada nivel de gris en la imagen original. Es necesaria para reconstruir el árbol de Huffman durante la decodificación.

- bitsCodificados: cadena de caracteres
  Contiene la secuencia de bits resultante de aplicar la codificación de Huffman a la imagen original. Esta cadena representa la versión comprimida de la imagen.

Operaciones:

- obtenerAncho(): devuelve el ancho original de la imagen.
- fijarAncho(): asigna el valor del ancho original.
- obtenerAlto(): devuelve el alto original de la imagen.
- fijarAlto(): asigna el valor del alto original.
- obtenerFrecuencias(): devuelve la lista de frecuencias asociadas a la imagen.
- fijarFrecuencias(): asigna la lista de frecuencias.
- obtenerBitsCodificados(): devuelve la secuencia de bits comprimida.
- fijarBitsCodificados(): asigna la secuencia de bits comprimida.

3.4. Descripción del Programa

- Entradas esperadas
- Salidas generadas
- Comandos principales
- Condiciones previas y posteriores de cada operación

El sistema implementado en esta entrega permite trabajar con imágenes en escala de grises bajo el formato PGM y ofrece una consola interactiva desde la cual el usuario puede ejecutar una serie de comandos para manipular dichas imágenes. Las funcionalidades disponibles están divididas por componentes, cada uno enfocado en una tarea específica del procesamiento de imágenes.

En esta segunda entrega, se conservan todos los comandos desarrollados para el Componente 1 (carga de imágenes, carga de volúmenes, generación de proyecciones) y se integran los correspondientes al Componente 2, los cuales permiten codificar y decodificar imágenes mediante el algoritmo de Huffman. El sistema completo se ejecuta en consola bajo un flujo sencillo e intuitivo.

Interfaz del sistema

Al iniciar el programa, el usuario se encuentra con un mensaje de bienvenida que introduce brevemente el propósito del sistema y ofrece orientación inicial. Desde este punto, se puede ingresar cualquier comando, y el sistema responde con un mensaje de éxito o error dependiendo del resultado de la operación.

Existe un comando llamado ayuda que permite consultar la lista completa de comandos disponibles, así como una descripción detallada de la sintaxis de cada uno. Esto ayuda al usuario a entender rápidamente cómo interactuar con el sistema.

Comandos disponibles (funcionalidades activas)

1. Cargar imagen

- **Comando:** cargar_imagen nombre_archivo.pgm
- **Descripción:** Carga en memoria una imagen en formato PGM. La imagen debe tener valores de intensidad entre 0 y 255.
- **Ejemplo de uso:** $ cargar_imagen img_04.pgm

2. Codificar imagen con Huffman

- **Comando:** codificar_imagen nombre_archivo.huf
- **Descripción:** Codifica la imagen cargada en memoria utilizando el algoritmo de Huffman. El resultado es un archivo binario .huf que contiene:
    - El ancho y alto de la imagen (2 bytes cada uno).
    - El valor máximo de intensidad (1 byte).
    - Las frecuencias de cada intensidad (de 0 a M), con 8 bytes por valor.
    - La secuencia comprimida de bits correspondiente a todos los píxeles leídos por filas.
- **Condiciones de uso:** Solo puede ejecutarse si ya se ha cargado una imagen.
- **Ejemplo de uso:** $ codificar_imagen comprimida.huf

3. Decodificar archivo .huf

- **Comando:** decodificar_archivo archivo.huf imagen_salida.pgm

- **Descripción:** Toma un archivo binario .huf y lo decodifica para generar una nueva imagen en formato PGM. Se reconstruye el árbol de Huffman a partir de las frecuencias almacenadas en el archivo y se recorre la secuencia de bits para recuperar la matriz de pixeles.
- **Condiciones de uso:** No requiere una imagen cargada previamente. El archivo .huf debe seguir la estructura exacta generada por el comando de codificación.
- **Ejemplo de uso:** $ decodificar_archivo comprimida.huf salida.pgm

4. Comando de salida

- **Comando:** salir
- **Descripción:** Finaliza la ejecución del sistema.

Validaciones y manejo de errores

El sistema incluye múltiples validaciones para asegurar que cada comando se ejecute correctamente:

- Si el usuario intenta codificar una imagen sin haberla cargado previamente, se muestra un mensaje de error: *"No hay una imagen cargada en memoria."*
- Si el archivo.huf a decodificar no existe, está corrupto o no cumple con la estructura esperada, se informa al usuario: *"El archivo no ha podido ser decodificado."*
- Si se ingresan comandos con una sintaxis incorrecta, el sistema guía al usuario con ejemplos apropiados a través del comando ayuda.

Comportamiento del sistema

- El sistema almacena en memoria una única imagen cargada a la vez. Si se carga una nueva, esta reemplaza a la anterior.
- Las operaciones de codificación y decodificación no alteran la imagen original, sino que generan nuevos archivos de salida.
- La estructura modular del código permite mantener separado el manejo de la interfaz, la lógica del algoritmo de Huffman y las operaciones sobre imágenes.

Integración con el Componente 1

El sistema mantiene la funcionalidad original del Componente 1, incluyendo comandos como cargar_volumen, info_imagen, info_volumen y proyeccion2D, que siguen operando de manera independiente. Esto permite al usuario trabajar con volúmenes de imágenes o imágenes individuales, según el componente que desee utilizar.

3.5. Plan de Pruebas

Objetivo:

- Se espera que mediante el plan de pruebas se pueda verificar el correcto funcionamiento del comando decodificar imagen

Compilacion y Ejecucion

Compilacion

Pasos Realizados:

- Compilacion de los archivos: imagen.h, imagen.hxx, proyeccion.h, proyeccion.hxx, huffman.h, huffman.hxx, main.cpp ▪
- Comando utilizado para la compilaci´on con informaci´on de depuracion: G+ -std-c++11 -o prog imagen.h imagen.hxx proyeccion.h proyeccion.hxx, huffman.h huffman.hxx main.cpp

Ejecucion

Pasos realizados:

- Ejecucion del programa mediante ./prog
- Cargar una imagen en memoria mediante la consola con el comando ¸cargarimagen”junto a el archivo que contiene la imagen
- Comando ejemplo: Cargar Imagen x.pgm

- Comprimir la imagen en memoria mediante la consola con el comando “codificar_imagen” junto a el nombre del archivo con la imagen comprimida
- Comando ejemplo: codificar imagen x.huf
- Descomprimir la imagen previamente comprimida mediante el comando ”decodificar_archivo”junto con el nombre del archivo que contiene la imagen comprimida y el nombre del archivo donde se quiere guardar la imagen descomprimida
- Comando ejemplo: decodificiar archivo x.huf x.pgm

Resultado Esperado:

- Se espera que siguiendo esta secuencia de pasos al momento de la ejecución la imagen cargada en memoria mediante el comando “cargar_imagen” que fue comprimida mediante el comando “codificar_imagen” sea la misma que se obtiene cuando se descomprime al usar el comando “decodificar_imagen”.

Caso de Prueba	Comando Ejecutado	Descripcion	Resultado Obtenido	Paso a Paso
CP01	Cargar_imagen Entrada.pgm	Ruta a un archivo de formato .pgm (Ejemplo img_02.pgm)	La imagen es cargada correctamente en memoria y se muestra el mensaje de exito	1. Ejecutar cargar_imagen entrada.pgm 2. Comprobar que se haya cargado en memoria
CP02	Codificar_imagen salida.pgm	Comprime la imagen cargada en memoria y la almacena en un	La imagen en memoria es comprimida y se guarda en el archivo .huf	1. Ejecutar cargar_imagen entrada.pgm 2. Ejecutar codificar_imagensalida.pgm

		archivo .huf		3. Verificar la creacion del archivo .huf
CP03	Codificar_imagen salida.huf	Comprimir imagen cargada en memoria sin haber cargado una imagen en memoria anteriorme nte	Se muestra el mensaje de error de “Error: No hay una imagen en memoria.”	1. Ejecutar codificar_ima gensalida.pgm 2. Verificar mensaje de error
CP04	Decodificar_Arch ivo entrada.huf salida.pgm	Decodificar el archivo de formato .huf (Ejemplo x.huf)	Se reconstruye la imagen original y se guarda en salida.pgm	1. Ejecutar decodificar_ar chivo entrada.huf salida.pgm 2.Verificar que se cree salida.pgm y verificar sus contenidos

Prueba de Escritorio

Objetivo

- Verificar el funcionamiento del programa y de el comando descomprimir archivo siguiendo los pasos establecidos anteriormente en la subseccion de Ejecucion

Casos escogidos

- Para verificar el funcionamiento del proceso ya especificado se utilizará el archivo img _02.pgm que contiene una imagen la cual cargaremos en memoria, sera comprimida en el archivo x.huf y descomprimida en el archivo x.pgm, se espera que no exista diferencia de bits entre img02.pgm y x.pgm , ademas que el tamaño de x.huf sea menor en tamaño a el tamaño de img02.pgm
- Para verificar que en el proceso de comprensión y descomprensión el contenido de la imagen se procesaran las dos imágenes y se compararan entre si en una herramienta de edición de imágenes

Prueba 1

En la prueba 1 se ejecutó un procedimiento estándar del uso del gestor de imagenes con objetivo de realizar el proceso desde que se carga una imagen en el sistema hasta que se descomprime el archivo de una imagen comprimida.

Como se puede ver todo el procedimiento ocurrio sin ningun tipo de error y se obtuvo los resultados esperados

Prueba 2

En la prueba 2 se ejecutó un procedimiento estándar del uso del gestor de imágenes con objetivo de realizar el proceso desde que se carga una imagen en el sistema hasta que se descomprime el archivo de una imagen comprimida, pero sin cargar una imagen en el sistema.

Como se puede ver al momento de intentar codificar la imagen cargada en memoria, el sistema arroja el error de que no hay ninguna imagen cargada en memoria por lo que se detiene el proceso.

Prueba 3

En la prueba 3 se ejecutó solo el comando decodificar archivo para verificar el funcionamiento en solitario del comando.

Como se puede ver el comando se ejecutó sin problemas y descomprimió el archivo que contenía la imagen y la almaceno en nombre indicado por el usuario.

Comparacion entre imagen antes y despues de comprimir

- Imagen img 02 antes de comprimir:

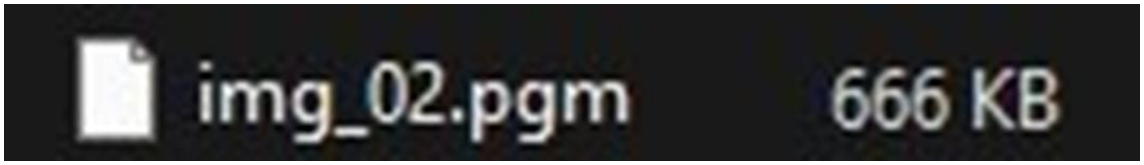
- Imagen img 02 despues de descomprimir:

- Comparacion entre las dos imagenes en una aplicacion de edicion de imagenes

Como se puede ver dentro del editor de imagenes, al momento de sobreponer las imagenes entre si, no se evidencia ninguna diferencia entre la imagen antes de comprimir y la imagen descomprimida.

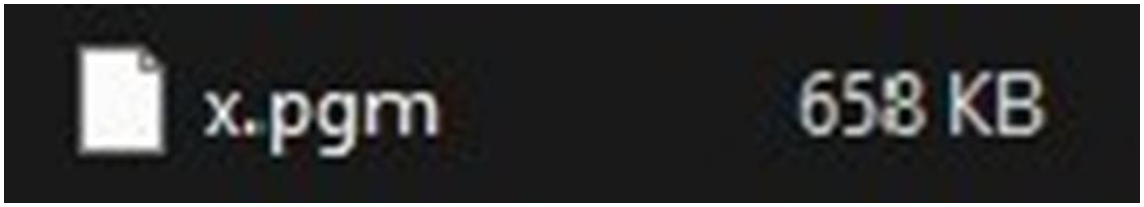
Archivos utilizados

- Img_02.pgm



- x.huf

- x.pgm



- Comparación entre Archivos:

  Comparando el tamaño de la imagen antes de comprimir y después de comprimir vemos que el algoritmo es capaz de comprimir archivos hasta en un ¼ del tamaño original.

Justificacion de Casos de Uso

Se decidio realizar tres pruebas, un proceso de ejecucion completo, un proceso de ejecucion completo sin cargar imagen en memoria y un proceso solo con el comando descomprimir archivo, la prueba 1 con proposito de verificar la compilacion y ejecucion que se realizaria normalmente en un caso de uso, la prueba 2 con propocito de verificar que pasaria si se trata de comprimir una imagen aun que no se carge en memoria y la prueba 3 para comprobar el funcionamiento en solitario de el comando que estamos evaluando

Correcion de Erorres

Se corrigio errores donde siempre se mostraba el mensaje de confirmacion de exito de comprension y descompre- si´on de imagen

4. CONCLUSIONES GENERALES

El desarrollo de esta segunda entrega del proyecto permitió consolidar y ampliar las capacidades del sistema de procesamiento de imágenes en escala de grises, integrando exitosamente el Componente 2, relacionado con la compresión y descompresión de imágenes mediante el algoritmo de Huffman. A partir de la retroalimentación recibida en la entrega anterior, se lograron importantes mejoras en la claridad de la interfaz de usuario, en la calidad de la documentación, y en la estructuración del plan de pruebas.

Uno de los principales aprendizajes alcanzados en esta entrega fue la comprensión profunda del funcionamiento del algoritmo de Huffman y su implementación mediante estructuras de datos como árboles binarios. El trabajo realizado nos permitió aplicar de manera práctica conceptos fundamentales de organización de datos, optimización de almacenamiento y manipulación de archivos binarios, fortaleciendo nuestra capacidad para diseñar sistemas modulares y eficientes.

Entre los desafíos enfrentados se destaca la construcción correcta del árbol de Huffman y la gestión de la codificación y decodificación de las secuencias de bits, garantizando que el proceso fuera completamente reversible. También representó un reto la implementación de mecanismos de validación y manejo de errores en la interfaz de consola, para asegurar que el sistema fuera robusto ante entradas inválidas o archivos mal formateados.

Respecto a la entrega anterior, se evidenció un avance significativo tanto en la calidad técnica como en la madurez del sistema. Se corrigieron las deficiencias señaladas en el plan de pruebas, se mejoró la presentación y funcionalidad de los comandos de ayuda, y se eliminó el uso de términos en inglés dentro de los mensajes al usuario, logrando una mayor coherencia en la interacción. Además, se reforzó la organización modular del código, separando adecuadamente las funciones de interfaz, lógica de negocio y manipulación de datos.

De cara a la siguiente entrega, el proyecto se encuentra en una posición sólida para integrar el Componente 3, orientado a la segmentación de imágenes mediante técnicas basadas en grafos. La experiencia adquirida hasta este punto en el manejo de estructuras complejas, validación de procesos y diseño de pruebas exhaustivas servirá como base para enfrentar los nuevos retos que plantea el desarrollo de algoritmos de segmentación sobre imágenes digitales.

En conclusión, esta entrega representa un avance no solo en el cumplimiento de los requerimientos técnicos del proyecto, sino también en el fortalecimiento de competencias esenciales para el diseño, implementación y documentación de sistemas basados en estructuras de datos.