

Fecha: 24/04/2025

Informe Entrega 2 Proyecto

Autores: Santiago Mesa y Jeronimo Chaparro Tenorio.

Materia: Estructura de Datos.

Pontificia Universidad Javeriana

Resumen de lo solicitado:

Para la entrega número dos del proyecto, se nos solicita implementar un sistema de codificación y decodificación de archivos de imagen en escala de grises utilizando el algoritmo de compresión de Huffman. El sistema debe:

- Leer una imagen en formato `.pgm` cargada en memoria.
- Calcular la frecuencia de los valores de intensidad de los píxeles.
- Construir un árbol de Huffman y codificar los píxeles.
- Almacenar la información codificada en un archivo `.huf` binario con la estructura: `W H M F0...FM bits....`
- Decodificar posteriormente el archivo `.huf` y reconstruir la imagen en formato `.pgm`

Propuesta de solución:

Se reutiliza el TAD `Imagen` de la entrega anterior y se implementa un nuevo TAD basado en árboles binarios para representar el árbol de Huffman. La solución se basa en los siguientes pasos clave:

- Calcular las frecuencias de aparición de cada intensidad de píxel.
- Construir el árbol de Huffman a partir de las frecuencias.
- Recorrer el árbol para generar los códigos binarios por intensidad.
- Codificar la imagen en una secuencia de bits y escribirla en archivo binario.
- Leer el archivo, reconstruir el árbol y decodificar la imagen.

1. Diseño de TADs: Huffman Tree e imagen

TAD Imagen:

- **Atributos:**
 - `nombre` (string): Nombre del archivo de la imagen.
 - `codigo` (string): Formato del archivo (por ejemplo, "P2").
 - `xTamano` (int): Ancho de la imagen en píxeles.
 - `yTamano` (int): Alto de la imagen en píxeles.
 - `maxIntensidad` (int): Valor máximo de intensidad de los píxeles.
 - `lista` (list<list<int>>): Lista bidimensional que almacena los valores de los píxeles.
- **Métodos:**

- `setNombre, setCodigo, setXTamano, setYTamano, setMaxIntensidad, setLista`: Métodos "set" para establecer los valores de los atributos.
- `getNombre, getCodigo, getXTamano, getYTamano, getMaxIntensidad, getLista`: Métodos "get" para obtener los valores de los atributos.
- `~Imagen()`: Destructor que limpia la lista de píxeles.

TAD HuffmanTree:

- **Atributos:** nodo raíz, mapa de códigos por intensidad.
- **Métodos:**
 - Constructor que recibe el vector de frecuencias.
 - `getCodes()`: retorna un `map<unsigned char, string>` con los códigos.
 - `getRoot()`: retorna el nodo raíz para la decodificación.

2. Comandos y sus Entradas/Salidas

2.1 codificar_imagen <nombre_archivo.huf>

- **Entrada:** imagen previamente cargada en memoria.
- **Salida:** Archivo binario `.huf` generado, mensaje de éxito o error.

2.2 decodificar_archivo <nombre_archivo.huf> <nombre_imagen.pgm>

- **Entrada:** Archivo `.huf` válido.
- **Salida:** Imagen `.pgm` reconstruida, mensaje de éxito o error.
-

3. Funciones implementadas

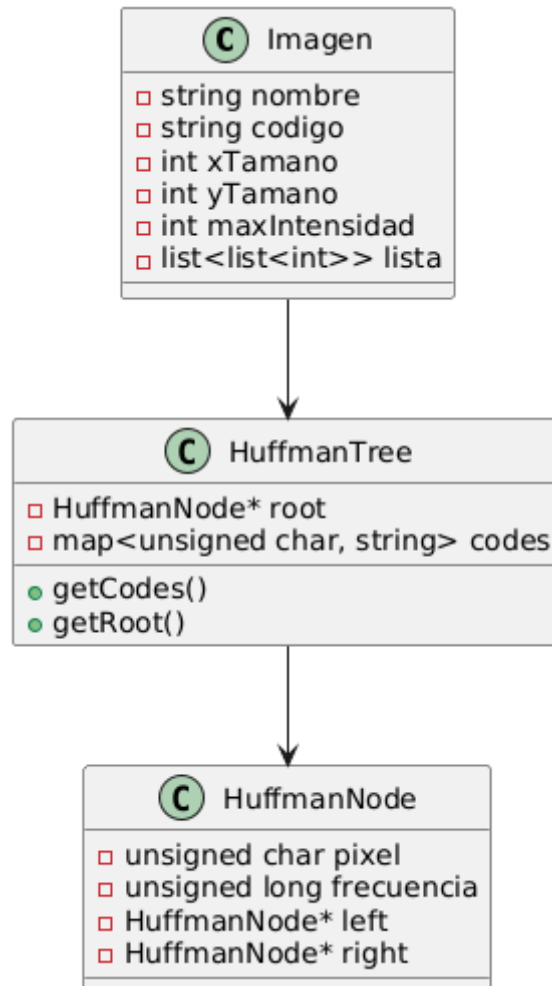
3.1 codificarImagen(argumentos):

- Recorre los píxeles y cuenta frecuencias.
- Construye el árbol de Huffman.
- Escribe cabecera binaria (W, H, M, F0...FM).
- Codifica imagen a bits y guarda `numBitsValidos` al final.

3.2 decodificarArchivo(argumentos):

- Lee cabecera y frecuencias.
- Reconstruye el árbol y decodifica el bitstream.
- Verifica tamaño correcto y guarda la imagen `.pgm`.

4. Diagrama de Relaciones



5. Plan de Pruebas

Objetivo del componente

Verificar que el sistema es capaz de codificar correctamente una imagen `.pgm` utilizando Huffman y luego decodificarla desde el archivo `.huf`, generando una imagen `.pgm` idéntica en contenido (visual y estructuralmente) a la original

5.1 `codificar_imagen <nombre_archivo.huf>`



Caso de Prueba	Valores de Entrada	Resultado Esperado	Resultado Obtenido	Estado (Éxito/Falla)
Codificación exitosa imagen pequeña	<code>codificar_imagen img_04.huf</code>	Se genera <code>img_04.huf</code> sin errores	La imagen en memoria ha sido codificada exitosamente y almacenada en el archivo <code>img_04.huf</code> .	✓ Éxito
Codificación imagen mediana	<code>codificar_imagen img_08.huf</code>	Se genera <code>img_08.huf</code> correctamente	La imagen en memoria ha sido codificada exitosamente y almacenada en el archivo <code>img_08.huf</code> .	✓ Éxito
Sin imagen cargada	<code>codificar_imagen test.huf</code>	Error: "No hay una imagen cargada en memoria."	Error: "No hay una imagen cargada en memoria."	✓ Éxito
Nombre inválido	<code>codificar_imagen /ruta/fallida.huf</code>	Error del sistema al guardar archivo	Error: No se pudo crear el archivo <code>/ruta/fallida.huf</code> .	⚠ (depende del SO)

5.2 decodificar_archivo <archivo.huf> <archivo.pgm>

Caso de Prueba	Valores de Entrada	Resultado Esperado	Resultado Obtenido	Estado (Éxito/Falla)
Decodificación exitosa	decodificar_archivo img_04.huf salida.pgm	El archivo img_04.huf ha sido decodificado exitosamente, y la imagen correspondiente se ha almacenado en el archivo salida.pgm.	El archivo img_04.huf ha sido decodificado exitosamente, y la imagen correspondiente se ha almacenado en el archivo salida.pgm.	✓ Éxito
Archivo .huf inexistente	decodificar_archivo no_existe.huf salida.pgm	El archivo no_existe.huf no ha podido ser decodificado.	El archivo no_existe.huf no ha podido ser decodificado.	✓ Éxito
Archivo truncado	decodificar_archivo truncado.huf salida.pgm	Advertencia: se decodificaron 1111084 pixeles, pero se esperaban 34443635. El archivo img_04.huf ha sido decodificado exitosamente, y la imagen correspondiente se ha almacenado en el archivo salida.pgm.	Advertencia: se decodificaron 1111084 pixeles, pero se esperaban 34443635. El archivo img_04.huf ha sido decodificado exitosamente, y la imagen correspondiente se ha almacenado en el archivo salida.pgm.	✓ Éxito

Criterios de aceptación

- ✓ Se puede codificar cualquier imagen .pgm cargada en memoria.
- ✓ Se genera un archivo .huf con la estructura W H M F0...FM bits.
- ✓ Se puede decodificar el archivo .huf y obtener una imagen .pgm válida.

-  La imagen reconstruida tiene las mismas dimensiones y píxeles.
-  Se reportan errores correctamente si:
 - No hay imagen en memoria.
 - El archivo **.huf** no existe o está corrupto.
 - No se puede reconstruir el árbol.

6. Guía de compilación

Esta guía explica cómo compilar y ejecutar el proyecto en Linux y Windows utilizando tanto make como g++

1. Compilar y Ejecutar con make

El proyecto incluye un Makefile que permite compilar y ejecutar el programa de manera sencilla. El Makefile es compatible con **Linux** y **Windows**.

En Linux:

Abre una terminal en la carpeta donde se encuentra el Makefile.

Compila el proyecto:

\$ "make"

Esto generará el archivo ejecutable "**mi_programa.exe**".

Ejecuta el programa:

\$ "make run"

Limpiar archivos generados:

\$ "make clean"

En Windows:

Abre PowerShell en la carpeta donde se encuentra el Makefile (o la terminal de IDE).

Compila el proyecto:

"make"

Esto generará el archivo ejecutable “*mi_programa.exe*”.

Ejecuta el programa:

“make run”

Limpiar archivos generados:

“make clean”

2. Compilar y Ejecutar con g++

En Linux:

Abre una terminal en la carpeta src y compila todos los archivos fuente:

**\$ “g++ -Wall -Wextra -std=c++17 -g main.cpp comandos.cpp interfaz.cpp
utilidades.cpp clases.cpp -o mi_programa.exe”**

Esto generará el archivo ejecutable “*mi_programa.exe*”.

Ejecuta el programa:

“./mi_programa.exe”

En Windows:

Abre PowerShell en la carpeta srcb (o la terminal de IDE).

Compila todos los archivos fuente:

**“g++ -Wall -Wextra -std=c++17 -g main.cpp comandos.cpp interfaz.cpp
utilidades.cpp clases.cpp -o mi_programa.exe”**

Esto generará el archivo ejecutable “*mi_programa.exe*”.

Ejecuta el programa:

“.\mi_programa.exe”

7. Conclusión

El componente 2 del proyecto ha sido implementado exitosamente, cumpliendo con los requisitos del enunciado. Se logró realizar la codificación eficiente de las imágenes en escala de grises y su posterior decodificación sin pérdida de información. Se validó que las imágenes reconstruidas son visualmente y estructuralmente iguales a las originales. El plan de pruebas permite demostrar el correcto funcionamiento bajo diversos escenarios, incluyendo errores y archivos corruptos. Se recomienda en futuras extensiones optimizar la lectura binaria y considerar soporte para otros formatos.