Benemérita Universidad Autónoma de Puebla

Facultad de Ciencias Físico Matemáticas

Licenciatura en Matemáticas Aplicadas

ESTEGANOGRAFÍA CON LSB Y COMPRESIÓN MEDIANTE EL ALGORITMO DE HUFFMAN EN IMÁGENES DE MAPAS DE BITS

T E S I S

Para obtener el título de

Licenciado en Matemáticas Aplicadas

Presenta

Dulce Rosario Pérez Hernández

Director de tesis

M.C. Sergio Adán Juárez

Heroica Puebla de Zaragoza, Enero 2026

A todos aquellos que alguna vez intentaron buscar la verdad.

La verdad no se esconde; simplemente está donde nadie se atreve a mirar.

Agradecimientos

Gracias por estar, por sumar y por hacer mi vida más feliz.

A mi familia: A mis queridos padres, por ser mi guía y mi base inquebrantable. Gracias por sumar a mi vida de forma incondicional y por aligerar cada paso en este camino. Este logro les pertenece tanto como a mí. A mis hermanos, por su paciencia, su amor y por ser mis compañeros de vida.

A mi novio: A ti, mi amor, por tu fe inquebrantable en mí. Tu paciencia, tu calma y tu apoyo constante fueron la energía que me impulsó a seguir adelante. Gracias por estar siempre a mi lado, por hacer todo más fácil y por amarme sin condiciones.

A la memoria: A mi niño precioso, mi perrito Bailey. Aunque ya no estés conmigo, tu compañía silenciosa y la alegría pura que me regalaste durante este viaje aligeraron mi carga. Estás presente en cada página de este logro.

A mis compañeros de cuatro patas: A mis seis perritos que siguen a mi lado, por su amor incondicional, su compañía constante y su alegría diaria. En especial, a Kanna, hija de Bailey, por su ternura infinita, sus abrazos silenciosos y ese amor

que siempre sabe llegar en el momento justo. Su presencia ha sido un bálsamo en los días difíciles y una fuente constante de felicidad.

A mis amigos y amigas: A mis incondicionales. Agradezco cada risa compartida, cada palabra de aliento y esa amistad sincera que me demuestran a diario.

A mi asesor: Gracias por tomarse el tiempo de acompañarme y guiarme en este camino hacia la esteganografía. Su orientación, paciencia y compromiso fueron piezas clave para que este trabajo tomara forma y significado.

Con todo mi cariño y eterna gratitud,

Dulce Rosario Pérez Hernández

Resumen

Imagina poder esconder mensajes secretos, *totalmente invisibles*, en las fotografías que vemos todos los días en Internet. Aún más interesante: ¿qué pasaría si también pudiéramos descubrir cómo fueron ocultos?

Esto es precisamente lo que estudia la *esteganografía*, una técnica que consiste en ocultar información dentro de otros archivos de manera que su presencia pase completamente desapercibida. A diferencia de la *criptografía*, que oculta el contenido pero no su existencia, la esteganografía busca que nadie sospeche siquiera que hay un mensaje secreto.

Abstract

Imagine being able to hide secret messages, *completely invisible*, within the photographs we see every day on the Internet. Even more interesting: what would happen if we could also uncover how they were hidden?

This is precisely what *steganography* studies—a technique that consists of hiding information within other files so that its presence goes completely unnoticed. Unlike *cryptography*, which hides the content but not its existence, steganography aims to ensure that no one even suspects that there is a hidden message.

Índice general

Resumen IV

1. Introducción 1

1.1. Contexto y Motivación 1

1.2. Planteamiento del Problema 2

1.2.1. Contexto y Justificación del Desafío 2

1.2.2. Problema de la Capacidad y Optimización del Payload . . . 2

1.2.3. Pregunta de Investigación 3

1.3. Objetivos 3

1.3.1. Objetivo General 3

1.3.2. Objetivos Específicos 3

1.4. Justificación 4

2. Marco Teórico 6

2.1. Conceptos fundamentales 6

2.1.1. Esteganografía 6

2.1.2. Imagen Digital 9

2.1.3. Imágenes de Mapa de Bits (Raster) 10

2.1.4. Formato BMP 13

2.1.5. Píxel 16

2.1.6. Modelo RGB 17

2.1.7. Byte y Bit 19

2.1.8. Codificación ASCII 21

2.2. Codificación de Huffman 23

2.2.1. Historia 23
2.2.2. Definición del problema y principio básico 23
2.2.3. Pasos del proceso de codificación 24
2.2.4. Eficiencia, Entropía y Aplicaciones 24
2.2.5. Ejemplo del Algoritmo de Huffman 25
2.2.6. Construcción del Árbol de Huffman 26
2.2.7. Códigos de Huffman Generados 26
2.2.8. Relación con el tamaño en bits y bytes 27
2.2.9. Cálculo de la Entropía y Verificación de la Optimalidad del Código de Huffman 28
2.2.10. Codificación de la Palabra Completa 30
2.2.11. Cálculo de la Entropía de Shannon 31
2.2.12. Comparación con la Longitud Promedio del Código de Huffman 31
2.2.13. Demostración de Optimalidad 32
2.3. Desplazamientos Lógicos 34
2.3.1. Corrimiento a la Izquierda (SHL o $\ll$) 34
2.3.2. Corrimiento a la Derecha (SHR o $\gg$) 34
2.3.3. Aplicaciones Comunes 35
2.4. Algoritmo para determinar las dimensiones rectangulares óptimas 36
2.4.1. Ejemplos: 37
2.4.2. Aplicación al caso de la imagen portadora 37
2.5. Least Significant Bit (LSB) 38
2.5.1. Definición y Aplicación 38
2.5.2. Técnicas de Incrustación 38
2.5.3. Extracción de Información 39
2.6. Tecnologías relacionadas 39
2.6.1. Lenguaje C 39
2.6.2. Lenguaje Python y Librería Pillow 40
2.6.3. Formato de Imagen BMP 40
2.7. Revisión de literatura 41

2.8. Trabajos relacionados 41
2.8.1. Estudio de Algoritmos de IA Aplicables al Estegoanálisis de Imágenes 41
2.8.2. Estudio y análisis de los procesos de la esteganografía en Colombia 42
2.8.3. Esteganografía: el arte de ocultar información en medios digitales 43
2.8.4. An Extensive Survey of Digital Image Steganography: State of the Art 45
2.8.5. Review and Test of Steganography Techniques 45
2.9. Análisis comparativo de soluciones existentes 45
2.9.1. Comparación de metodologías 45

3. Desarrollo e Implementación 48
3.1. Análisis de Requisitos 48
3.1.1. Requisitos de la Incrustación (Payload) 48
3.1.2. Proceso de incrustación (Portadora) 49
3.1.3. LSB e Incrustación 50
3.1.4. Requisitos de la Extracción (Reconstrucción) 51
3.2. Diseño del Sistema 52
3.2.1. Proceso de Incrustación 52
3.2.2. Proceso de Extracción 53
3.3. Detalles de Implementación 53
3.3.1. Tecnologías Utilizadas 53
3.4. Pruebas de Desempeño y Validación 54
3.4.1. Prueba de Integridad (Reversibilidad) 54
3.4.2. Prueba de Estegoanálisis (Imperceptibilidad) 54

4. Metodología 55
4.1. Enfoque metodológico 55
4.2. Herramientas y tecnologías utilizadas 55
4.3. Diseño de la solución 57

4.4. Fases del desarrollo 57
4.4.1. Preparación del Mensaje Secreto 57
4.4.2. Preparación del Archivo Portador 58
4.4.3. LSB e Incrustación 58
4.4.4. Extracción 59
4.5. Recursos Necesarios 60

5. Resultados y Análisis **62**
5.1. Presentación de Resultados 62
5.2. Fase 1: Reducción de Niveles y Compresión Huffman 62
5.2.1. Ejecución del programa `niveles.c` 63
5.2.2. Ejecución del programa `compresion.c` 64
5.3. Fase 2: Preparación de la Imagen Portadora 65
5.3.1. Ejecución del programa `shlr.c` 65
5.3.2. Ejecución del programa `rectangulo.c` 66
5.4. Fase 3: Incrustación mediante LSB 67
5.4.1. Ejecución del programa `lsbimagen.py` 67
5.4.2. Comparación Visual de Incrustración 68
5.4.3. Análisis de Histogramas de Incrustación 74
5.5. Relación Señal-Ruido Pico (PSNR) 75
5.5.1. Resultados del PSNR 76
5.6. Fase 4: Extracción, Descompresión y Reconstrucción 77
5.6.1. Ejecución del programa `extraerlsb1.py` 77
5.6.2. Ejecución del programa `descompresion.c` 78
5.6.3. Ejecución del programa `niveles8.c` 79
5.6.4. Comparación Visual de Reconstrucción 80
5.6.5. Análisis de Histogramas de Reconstrucción 86
5.7. Difusión de Resultados 87
5.7.1. Participación en Congreso 87

6. Conclusiones y Trabajo Futuro **89**
6.1. Conclusiones 89

6.2. Análisis de Desempeño 89
6.2.1. Cumplimiento de Objetivos 90
6.3. Respuesta a la Pregunta de Investigación 91
6.4. Contribuciones 92
6.5. Limitaciones encontradas 93
6.6. Recomendaciones 93
6.7. Líneas de investigación futuras 94

Referencias Bibliográficas **97**

Apéndice A: Códigos Fuente Utilizados **103**

Índice de figuras

2.1. Ejemplo de imagen de mapa de bits se observan los píxeles individuales. 11
2.2. Ejemplo de gráfico vectorial ampliado: mantiene la nitidez en los bordes. 12
2.3. Comparación entre mapa de bits y gráfico vectorial. 13
2.4. Representación ampliada de una cuadrícula de píxeles en una imagen digital 17
2.5. Separación de los canales de color RGB en una imagen. 18
2.6. Ejemplo de byte. 19
2.7. Ejemplo de tabla ascii. 22
2.8. Árbol de Huffman final construido. 26
2.9. Ejemplo visual de desplazamiento lógico 35
2.10. Ejemplo de ocultación de información en el LSB de una imagen. . . 39

3.1. Fase 1: Incrustación (Payload). 49
3.2. Fase 2: Preparación de la imagen portadora. 50
3.3. Fase 3: LSB e Incrustación 51
3.4. Fase 4: Reconstrucción del mensaje. 52

5.1. Comparación de Mickey: Portadora vs Payload [32] 68
5.2. Mickey: Histograma Original vs. Payload 68
5.3. Comparación de Bosque: Portadora vs Payload [33] 69
5.4. Bosque: Histograma Original vs. Payload 69
5.5. Comparación de Montaña: Portadora vs. Payload [34] 70

5.6. Montaña: Histograma Original vs. Payload 70
5.7. Comparación de Flor: Portadora vs. Payload [35] 71
5.8. Flor: Histograma Original vs. Payload 71
5.9. Comparación de Fruta: Portadora vs. Payload [36] 72
5.10. Frutas: Histograma Original vs. Payload 72
5.11. Comparación de Lago: Portadora vs. Payload [37] 73
5.12. Lago: Histograma Original vs. Payload 73
5.13. Lenna Original vs. Lenna Reconstruida [38] 80
5.14. Lenna Histograma de reconstrucción 80
5.15. Baboon Original vs. Baboon Reconstruida [39] 81
5.16. Baboon Histograma de reconstrucción 81
5.17. Flower Original vs. Flower Reconstruida [40] 82
5.18. Flower Histograma de reconstrucción 82
5.19. BoatsColor Original vs. BoatsColor Reconstruida [41] 83
5.20. BoatsColor Histograma de reconstrucción 83
5.21. Pepper Original vs. Pepper Reconstruida [42] 84
5.22. Pepper Histograma de reconstrucción 84
5.23. Barbara Original vs. Barbara Reconstruida [43] 85
5.24. Barbara Histograma de reconstrucción 85
5.25. Constancia de participación en el 12CIMA, BUAP 2025. 88

Índice de tablas

2.1. Comparación de técnicas esteganográficas y sus ventajas 8
2.2. Estructura detallada de la Cabecera del Archivo BMP 15
2.3. Frecuencia de caracteres en la palabra `matematicas` ordenada de menor a mayor. 25
2.4. Códigos de Huffman ordenados de menor a mayor frecuencia. . . . 26
2.5. Probabilidades de aparición de cada carácter en `matematicas`. . . . 29
2.6. Longitudes de los códigos de Huffman. 32
2.7. Comparación de soluciones existentes en estegoanálisis de imágenes 46

5.1. Resultados de reducción de niveles 63
5.2. Resultados de compresión Huffman 64
5.3. Capacidad de almacenamiento de cada imagen portadora 66
5.4. Resultados de incrustación LSB 67
5.5. Valores de PSNR entre imágenes originales y con payload 76
5.6. Resultados de extracción LSB del archivo .huff 77
5.7. Resultados de descompresión del archivo Huffman 78
5.8. Resultados de la reconstrucción de niveles 79

6.1. Cumplimiento de objetivos específicos del sistema. 90

CAPÍTULO 1

INTRODUCCIÓN

1.1. Contexto y Motivación

En esta sección se explorarán algunas de las técnicas más utilizadas para ocultar información, con especial énfasis en la esteganografía basada en el **Bit Menos Significativo** (LSB, *Least Significant Bit*). Esta técnica aprovecha la forma en que las imágenes digitales almacenan sus datos, permitiendo insertar mensajes en los bits menos relevantes de los píxeles. Gracias a ello, es posible esconder texto o incluso archivos completos sin que se perciban alteraciones visibles en la imagen resultante.

La motivación para esta investigación radica en la **necesidad de fortalecer la seguridad y la discreción en la comunicación digital**, especialmente ante el incremento del intercambio de información sensible en entornos abiertos como Internet. Aunque la técnica LSB es ampliamente utilizada por su simplicidad y eficiencia, presenta vulnerabilidades que pueden comprometer la confidencialidad del mensaje si no se implementa de manera adecuada.

Por ello, esta investigación busca desarrollar una implementación de esteganografía más robusta y eficiente, utilizando la biblioteca **Pillow** (específicamente el módulo `PIL.Image`) para la manipulación de imágenes digitales. Esta herramienta permite cargar, acceder, modificar los valores RGB y guardar las imágenes modificadas, facilitando la experimentación y evaluación del algoritmo LSB propuesto.

Todas las pruebas se realizaron en el entorno **MobaXterm**, empleando el compilador `gcc` para el código en lenguaje C y el intérprete `Python 3` para los scripts de esteganografía.

1.2. Planteamiento del Problema

1.2.1. Contexto y Justificación del Desafío

La esteganografía digital —y en particular el método LSB— presenta un desafío fundamental: mantener el equilibrio entre la *capacidad de ocultamiento* (*payload*) y la *robustez* del esteganograma ante técnicas de estegoanálisis.

En el contexto de esta investigación, el objetivo es ocultar una imagen completa dentro de otra imagen de mayor tamaño (la **imagen portadora**), ambas en formato BMP. Este objetivo resulta inviable si la imagen a ocultar no es preprocesada, ya que la gran cantidad de datos necesaria para la incrustación generaría una distorsión notable y perceptible en la imagen portadora, comprometiendo la invisibilidad del mensaje y facilitando su detección.

1.2.2. Problema de la Capacidad y Optimización del Payload

El problema central que se aborda es la optimización del **payload** para garantizar una alta capacidad de inserción con un bajo impacto visual. La solución propuesta se estructura en dos fases:

1. **Optimización del payload**: Reducir eficientemente el tamaño de la imagen a ocultar mediante un esquema de compresión de doble etapa: primero, por reducción de la profundidad de color y, segundo, mediante la **Codificación de Huffman**. Este proceso minimiza drásticamente la cantidad de datos a incrustar.

2. **Localización óptima**: Realizar la incrustación en la región de la imagen portadora que ofrezca mayor seguridad. Se requiere el diseño e implementación de un algoritmo que identifique el **área rectangular máxima** dentro de la imagen que cumpla con las restricciones de tamaño necesarias para alojar el payload comprimido.

1.2.3. Pregunta de Investigación

A partir del desafío planteado, esta investigación busca responder a la siguiente pregunta:

¿De qué manera la combinación de la reducción de profundidad de color y la Codificación de Huffman, junto con la localización de un área rectangular máxima, permite desarrollar un sistema de esteganografía LSB más robusto y con mayor capacidad de ocultamiento que las implementaciones tradicionales sin preprocesamiento de datos?

1.3. Objetivos

1.3.1. Objetivo General

Desarrollar un sistema de esteganografía digital LSB para imágenes de mapa de bits que optimice la capacidad de ocultamiento mediante la aplicación combinada de la reducción de profundidad de color, la Codificación de Huffman y la localización del área rectangular óptima.

1.3.2. Objetivos Específicos

I. Analizar e implementar la reducción de profundidad de color para la imagen a ocultar, como paso previo a la compresión.

II. Diseñar e implementar el algoritmo de Codificación de Huffman para la compresión sin pérdida de la imagen secreta reducida.

III. Desarrollar un algoritmo de localización que determine el área rectangular máxima disponible dentro de la imagen portadora.

IV. Integrar los datos comprimidos en los Bits Menos Significativos (LSB) de los píxeles del área óptima de la imagen portadora.

V. Evaluar el desempeño del sistema propuesto utilizando métricas objetivas (calidad visual, tasa de compresión y precisión de extracción).

1.4. Justificación

El principal desafío en la **esteganografía basada en Bits Menos Significativos (LSB)** radica en la relación inversa existente entre la **capacidad de ocultamiento** (*payload*) y la **robustez del esteganograma**. Las implementaciones tradicionales del método LSB presentan una limitación fundamental: a medida que se incrementa la cantidad de información incrustada en la imagen portadora, se generan alteraciones tanto **estadísticas** como **visuales** que facilitan su detección mediante técnicas de **estegoanálisis**. Este fenómeno compromete la eficacia de los sistemas de esteganografía digital, reduciendo su aplicabilidad en entornos donde la discreción y la seguridad de la información son esenciales.

En este contexto, la presente investigación se justifica a partir de tres ejes fundamentales que buscan superar las limitaciones mencionadas y aportar una contribución significativa al campo de la esteganografía digital:

1. **Contribución a la eficiencia.** Se propone una mejora sustancial en la eficiencia del método LSB mediante la **integración del algoritmo de Codificación de Huffman**. La aplicación de esta técnica de compresión sin pérdida a la imagen secreta permite **reducir significativamente el volumen de datos a incrustar**, optimizando el uso del espacio disponible en la imagen portadora. Esto no solo libera espacio para la inserción de información adicional, sino que también **disminuye la probabilidad de generar artefactos visuales o distorsiones perceptibles**, aumentando la imperceptibilidad del esteganograma.

2. **Optimización del ocultamiento.** El desarrollo de un **algoritmo de localización del área rectangular óptima** aborda directamente el problema de la distribución espacial de los datos dentro de la imagen portadora. Este enfoque garantiza que la inserción de la información comprimida se realice en la región más adecuada desde el punto de vista estadístico y visual, **maximizando la capacidad de carga sin comprometer la calidad ni la seguridad del contenido visual**. De esta forma, se incrementa la robustez frente a métodos básicos de detección.

3. **Relevancia académica y científica.** El proyecto representa una contribución significativa al estudio y desarrollo de sistemas de **seguridad de la información basados en técnicas esteganográficas**. La propuesta demuestra la viabilidad de un **enfoque híbrido** que integra componentes de **procesamiento digital de imágenes** (reducción de profundidad de color), **compresión de datos** (Codificación de Huffman) y **análisis geométrico** (determinación de un área máxima) para

CAPÍTULO 2

MARCO TEÓRICO

2.1. Conceptos fundamentales

2.1.1. Esteganografía

La esteganografía (del griego *steganós*, que significa "cubierto", y *grafía*, "escritura") es el arte y la ciencia de **ocultar la existencia de un mensaje** dentro de otro mensaje u objeto (llamado **objeto portador** o *cover*), de modo que pase completamente desapercibido ante un observador no autorizado [1].

A diferencia de la **criptografía**, que cifra un mensaje para hacerlo ininteligible, la esteganografía se enfoca en la **invisibilidad**, es decir, en ocultar el hecho mismo de la comunicación. El objetivo principal es garantizar la **confidencialidad** y el **secreto** de la información, evitando que se detecte la presencia de los datos ocultos [2].

Tipos de Esteganografía

Los tipos de esteganografía se clasifican generalmente según el tipo de archivo portador que se utiliza:

a) Esteganografía de Imagen: Consiste en ocultar información dentro de archivos de imagen (JPEG, PNG, BMP). La técnica más popular es la de **Bits Menos Significativos (LSB)**, que reemplaza el último bit de cada píxel de color con un bit del mensaje secreto, siendo el cambio visual casi imperceptible.

b) Esteganografía de Audio: Implica incrustar el mensaje secreto en archivos de audio (MP3, WAV). Es más compleja que la de imagen debido a la sensibilidad del oído humano.

c) Esteganografía de Video: Combina técnicas de imagen y audio. La información puede ocultarse en los fotogramas del video, en la pista de audio, o en las partes menos utilizadas del flujo de datos.

d) Esteganografía de Texto (Lingüística): Oculta información dentro de archivos de texto simple, mediante **acrósticos** o **modificación sutil de formato**.

e) Esteganografía de Protocolo o Red (*Tunneling*): Oculta información aprovechando los campos o bits de los encabezados de los protocolos de red (como TCP, IP, ICMP), creando un **canal encubierto** [3].

Ventajas de la Esteganografía

La **esteganografía digital** representa una técnica estratégica en la seguridad de la información, que complementa métodos tradicionales de cifrado y control de acceso. A diferencia de la criptografía, cuyo objetivo es hacer ilegible la información, la esteganografía se centra en **ocultar la existencia misma del mensaje**, proporcionando un nivel de discreción que resulta crucial en escenarios donde la presencia de datos cifrados puede ser sospechosa [4].

Entre las principales ventajas de la esteganografía destacan:

a) **Ocultación completa de la comunicación.** Permite disimular totalmente la existencia de un mensaje al incrustarlo en archivos aparentemente inocuos, como imágenes, audio o vídeo. Esto hace que la comunicación sea prácticamente indetectable para agentes externos [5].

b) **Evasión de vigilancia y detección automatizada.** Los mensajes esteganográficos circulan sin activar sistemas de filtrado o vigilancia automatizada, lo que facilita la transmisión segura en redes con controles estrictos [6].

c) **Seguridad de doble capa.** Al combinar esteganografía con técnicas criptográficas (por ejemplo, cifrado previo y ocultamiento LSB), se logra una defensa de múltiples niveles: incluso si el mensaje es detectado, su contenido permanece inaccesible sin la clave de cifrado [6].

d) **Protección de propiedad intelectual mediante marcas de agua.** Inserta información de autenticidad de forma imperceptible en archivos multimedia, permitiendo rastrear la distribución no autorizada y prevenir la piratería digital [7].

Tabla comparativa de técnicas esteganográficas

Técnica	Ventaja principal	Aplicación típica
LSB (Least Significant Bit)	Alta capacidad de ocultamiento	Imágenes, audio
Transformada Discreta de Coseno (DCT)	Resistente a compresión JPEG	Imágenes digitales
Transformada de Fourier (FFT)	Robusta frente a filtrado	Audio, vídeo
Marcas de agua invisibles	Protección de propiedad intelectual	Multimedia y DRM
Esteganografía en red (Network Steganography)	Comunicación encubierta en protocolos	Redes y comunicaciones

Tabla 2.1: Comparación de técnicas esteganográficas y sus ventajas

En resumen, la esteganografía no solo permite ocultar información, sino que constituye una **estrategia integral de seguridad**, complementando criptografía, facilitando la comunicación encubierta y protegiendo la propiedad intelectual en entornos digitales complejos [4, 5, 6, 7].

Limitaciones de la Esteganografía

A pesar de sus ventajas, la esteganografía presenta desafíos importantes [3]:

Capacidad de Carga Útil Limitada:
La cantidad de información que se puede ocultar sin que el cambio sea perceptible es finita. Intentar ocultar grandes volúmenes de datos puede degradar el archivo portador. **Detectabilidad (Estegoanálisis):**
El campo del **Estegoanálisis** busca detectar mensajes ocultos mediante el análisis de anomalías estadísticas en el archivo portador. **Fragilidad o Falta de Robustez:**
Si el archivo portador se somete a un procesamiento común (compresión o cambio de formato), el mensaje oculto puede ser destruido. **Preocupaciones Éticas y Legales:**
Su cualidad de invisibilidad la hace atractiva para actividades maliciosas.

2.1.2. Imagen Digital

Esta seccion explicamos los fundamentos de las imágenes digitales, diferenciando entre imágenes de mapa de bits (ráster) y gráficos vectoriales. Se abordan sus estructuras, ventajas, limitaciones, formatos comunes y criterios para su uso adecuado.

Definición de Imagen Digital

Una **imagen digital** es una representación bidimensional de una imagen real o sintética a partir de una **matriz numérica** (generalmente binaria), donde cada elemento de la matriz almacena información de color o luminosidad de un punto de la imagen. La **digitalización** es el proceso mediante el cual una imagen analógica se transforma en una matriz de datos discretos. Esta matriz se compone de elementos básicos llamados **píxeles**, que son las unidades mínimas de información visual[8] .

Tipos de Imágenes Digitales

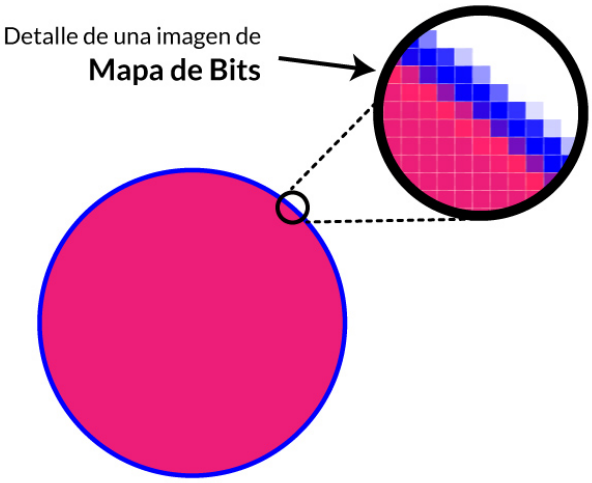
En informática y diseño, las imágenes digitales se clasifican fundamentalmente en dos grandes grupos según su estructura de representación:

(a) Imágenes de **mapa de bits** (raster o bitmap)

(b) Imágenes **vectoriales**

2.1.3. Imágenes de Mapa de Bits (Raster)

Las imágenes de **mapa de bits** (*bitmap* o *raster*) están compuestas por una rejilla rectangular de celdas diminutas conocidas como **píxeles**, cada una con un valor de color asociado [9].



[10].

Figura 2.1: Ejemplo de imagen de mapa de bits se observan los píxeles individuales.

a) **Estructura:** Basada en píxeles. La imagen es una rejilla de puntos de color.

b) **Ventajas:** Ideales para fotografías y arte de tono continuo, ya que capturan matices de color complejos con gran detalle.

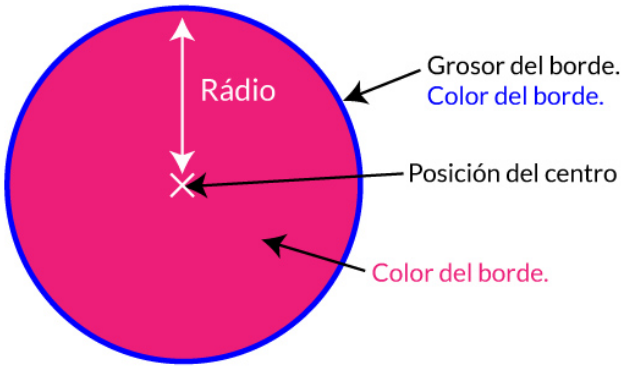
c) **Limitaciones:** Son **dependientes de la resolución**. Al ser ampliadas o escaladas, pierden calidad y presentan el fenómeno de "pixelación".

d) **Formatos Comunes:** JPEG, PNG, GIF, BMP, TIFF, RAW.

Una desventaja clave es que al aumentar su tamaño más allá de la resolución original, se produce una pérdida visible de nitidez. Sin embargo, son preferidas en contextos donde la fidelidad del color y el detalle son prioritarios, como en fotografía digital y vídeo [11, 12].

Gráficos Vectoriales

Los **gráficos vectoriales** se definen mediante objetos geométricos (líneas, curvas, polígonos, círculos) basados en **ecuaciones matemáticas** que describen su forma, color y posición [**wiki_vectorial**].



[10].

Figura 2.2: Ejemplo de gráfico vectorial ampliado: mantiene la nitidez en los bordes.

(a) **Estructura:** Basada en fórmulas matemáticas que describen formas, trayectorias y colores.

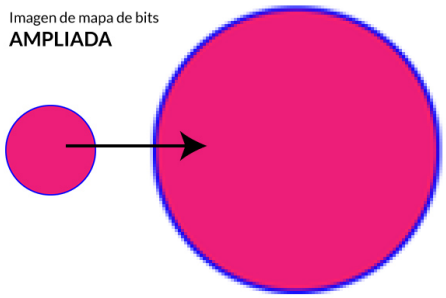
(b) **Ventajas:** Son **independientes de la resolución**. Pueden ser escalados a cualquier tamaño sin perder calidad ni sufrir pixelación. Son ideales para logotipos, tipografía y diseño gráfico.

(c) **Limitaciones:** No son adecuados para representar fotografías o imágenes complejas con variaciones tonales sutiles.

(d) **Formatos Comunes:** SVG, AI, EPS, PDF (que a menudo contienen vectores).

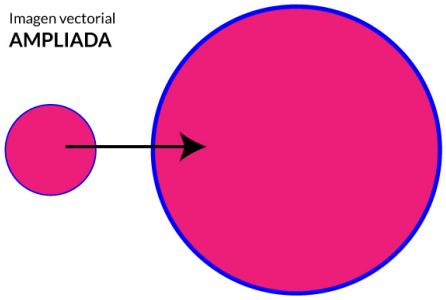
Su principal ventaja radica en la escalabilidad ilimitada y la precisión del trazo. Al no depender de píxeles, mantienen la nitidez independientemente de su tamaño o resolución de salida [13, 11].

Comparación Visual



[10]

(a) Mapa de bits ampliado



[10]

(b) Gráfico vectorial ampliado

Figura 2.3: Comparación entre mapa de bits y gráfico vectorial.

Comparación y Casos de Uso

a) **Escalabilidad:** Las imágenes vectoriales no pierden calidad al ampliarse, mientras que las ráster sí [14].

b) **Tamaño de archivo:** Las imágenes vectoriales suelen ocupar menos espacio si contienen formas simples; las ráster requieren más memoria en alta resolución.

c) **Detalle de color:** Las imágenes ráster permiten gradientes complejos; las vectoriales son más limitadas en ese aspecto.

d) **Usos comunes:**

1) *Ráster:* fotografía, texturas, ilustraciones digitales.
2) *Vectoriales:* logotipos, carteles, tipografía, iconografía.

2.1.4. Formato BMP

El formato BMP (Bitmap) es un formato de archivo de imagen digital de tipo ráster (mapa de bits) desarrollado por Microsoft. Está diseñado para almacenar gráficos con la mayor fidelidad posible, generalmente sin aplicar compresión o utilizando una compresión sin pérdida simple.

Características y Propiedades Clave

El BMP se distingue por sus propiedades de almacenamiento directo:

a) **Sin Pérdida de Calidad:** Almacena cada píxel de la imagen individualmente sin aplicar algoritmos de compresión con pérdida, lo que garantiza una fidelidad total al color original.

b) **Gran Tamaño de Archivo:** La principal limitación del formato es el tamaño excesivo de los archivos, ya que al no usar compresión eficiente, son considerablemente más grandes que otros formatos como JPEG o PNG, haciéndolo impráctico para la web o la transferencia[15] .

c) **Estructura Simple:** Es fácil de leer y procesar programáticamente, lo que contribuye a su compatibilidad universal con casi todos los sistemas operativos y programas de edición.

Estructura Básica

La estructura de un archivo BMP se compone de un flujo de datos organizado en secciones secuenciales:

Encabezado del Archivo (File Header): Contiene la firma ('BM'), el tamaño total del archivo y otros datos preliminares.

Encabezado de Información (Info Header): Contiene detalles clave de la imagen, como la anchura, la altura, la profundidad de color (bits por píxel o BPP) y el tipo de compresión (si aplica).

Paleta de Colores (Opcional): Presente solo en imágenes con baja profundidad de color (menos de 24 bits).

Datos de Píxel (Pixel Data): La matriz de datos de color reales, almacenados píxel por píxel.

Bytes	Información
0, 1	Tipo de fichero “BM”
2, 3, 4, 5	Tamaño del archivo
6, 7	Reservado
8, 9	Reservado
10, 11, 12, 13	Inicio de los datos de la imagen
14, 15, 16, 17	Tamaño de la cabecera del bitmap
18, 19, 20, 21	Anchura (píxels)
22, 23, 24, 25	Altura (píxels)
26, 27	Número de planos
28, 29	Tamaño de cada punto
30, 31, 32, 33	Compresión (0=no comprimido)
34, 35, 36, 37	Tamaño de la imagen
38, 39, 40, 41	Resolución horizontal
42, 43, 44, 45	Resolución vertical
46, 47, 48, 49	Tamaño de la tabla de color
50, 51, 52, 53	Contador de colores importantes

Tabla 2.2: Estructura detallada de la Cabecera del Archivo BMP

Usos Comunes

El formato BMP es ideal para almacenar: **Archivos Maestros y Edición:**

Uso en aplicaciones profesionales de edición gráfica y como formato intermedio, donde la preservación de la máxima calidad es esencial y el tamaño del archivo no es una limitación crítica.

Imágenes de Sistema: Pequeños iconos y gráficos utilizados internamente por el sistema operativo Windows

2.1.5. Píxel

El **Píxel** (acrónimo de *Picture Element*) es la unidad mínima de información que compone una imagen digital de tipo ráster.

Definición

Cada píxel es un punto individual en una cuadrícula (matriz) que contiene la información de color y luminosidad de ese punto específico [1]. Además, ampliando lo suficiente una imagen en pantalla, pueden observarse los píxeles que la componen: cada uno representa un pequeño punto de color que, en conjunto con otros, forma la imagen completa. En las imágenes de mapa de bits, cada píxel se codifica mediante una longitud de bits determinada (la profundidad de color), por ejemplo 8 bits, 24 bits o incluso 48 bits por píxel [16].

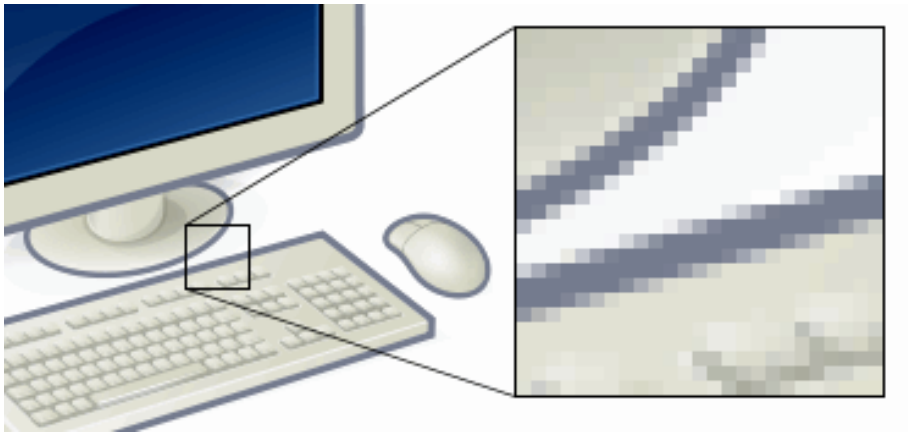
Tipos por Profundidad

(a) **Píxel Binario (1 bit):** Solo blanco o negro (2^1 valores).

(b) **Píxel RGB (24 bits):** Almacena 3 bytes de información (3 canales de 8 bits cada uno), lo que permite aproximadamente $2^{24} = 16\,777\,216$ colores posibles [16].

Resolución, Tamaño y Densidad

El número de píxeles de una imagen se denomina resolución, comúnmente expresada como "ancho × alto" en píxeles (por ejemplo, 640 × 480). A mayor cantidad de píxeles, más detalles puede contener la imagen. La densidad de píxeles, medida en píxeles por pulgada (PPI), también influye en la nitidez: mayor PPI implica mayor claridad de imagen [16].



[16]

Figura 2.4: Representación ampliada de una cuadrícula de píxeles en una imagen digital .

2.1.6. Modelo RGB

El **Modelo RGB** (*Red, Green, Blue*) es un modelo aditivo de color primario utilizado para la representación de colores en dispositivos de visualización digital, como monitores, cámaras o escáneres.

Definición y Estructura

El modelo describe la creación de un amplio rango de colores mediante la combinación de intensidades de las tres luces primarias: roja (R), verde (G) y azul (B). Cada color se obtiene al variar la intensidad de cada componente, generando diferentes tonalidades por adición de luz.

(a) **Codificación 24-bits (True Color):** Es el estándar más utilizado. Cada canal (R, G, B) emplea 8 bits ($2^8 = 256$ niveles de intensidad), permitiendo aproximadamente 16,7 millones de colores posibles (256^3 combinaciones).

(b) **Representación de color:** Un color puede expresarse como un conjunto de tres valores numéricos que indican la intensidad de cada primario:

$$\text{Color} = (R, G, B)$$

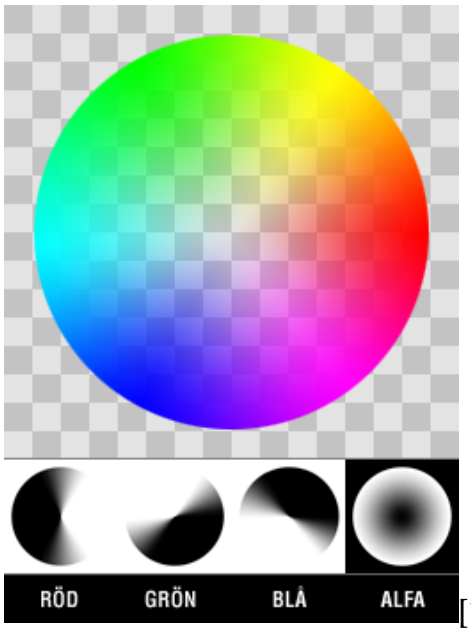
Canales de Imagen

En el procesamiento de imágenes digitales, un **canal** es una matriz de datos que corresponde a un único componente de color o a una propiedad específica del píxel. Cada canal puede entenderse como una "capa" que almacena información particular de la imagen, como los valores de color primario o el nivel de transparencia.

Tipos y Uso

Canales de Color:

Corresponden a las componentes R, G y B en el modelo aditivo. En una imagen de 24 bits, cada canal es una matriz de 8 bits que define la intensidad del color respectivo en cada píxel.



[17]

Figura 2.5: Separación de los canales de color RGB en una imagen.

2.1.7. Byte y Bit

Para la manipulación de información digital y el estudio de la esteganografía a nivel de píxel, es fundamental comprender las unidades básicas de almacenamiento.

Byte

El **Byte** es una unidad de datos que generalmente consta de **8 bits** ($2^8 = 256$ valores posibles) y constituye la unidad básica utilizada para **codificar un carácter**. En el contexto de la **esteganografía LSB (Least Significant Bit)**, cada byte representa la intensidad de un canal de color **Rojo**, **Verde** o **Azul** en una imagen digital.

Además, el byte también se utiliza como unidad fundamental para representar el **payload** (mensaje secreto), el cual suele estar codificado mediante el estándar **ASCII**.

Bit

El **Bit** (*Binary Digit*) es la unidad de información más pequeña posible, representando un valor binario único de **0** o **1**. Constituye la base de toda la **codificación digital**. En el contexto de la **esteganografía** y el procesamiento de imágenes, cada píxel en la imagen portadora utiliza múltiples bits (típicamente 8 para cada canal de color) para definir su intensidad.

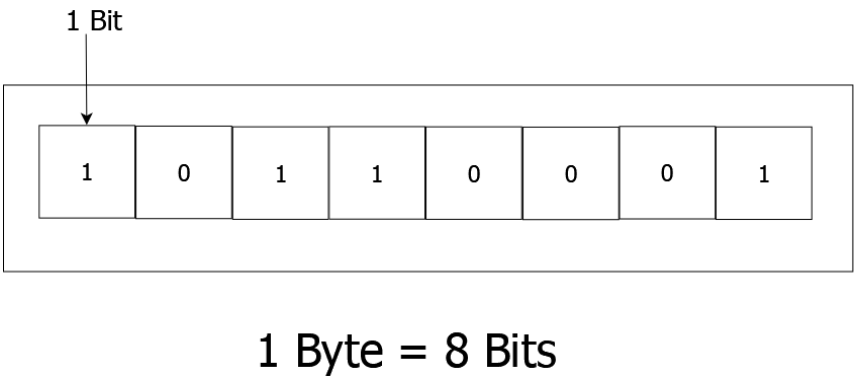


Figura 2.6: Ejemplo de byte.

La Figura 2.6 muestra de manera esquemática la relación jerárquica entre ambas unidades: un **Byte** está conformado por **ocho Bits**, cada uno de los cuales almacena un valor binario (0 o 1). Esta estructura es esencial para la representación digital de la información, ya que permite codificar caracteres, colores e incluso datos multimedia a través de combinaciones binarias.

2.1.8. Codificación ASCII

El estándar ASCII (*Americn Standard Code for Information Interchange*) asigna valores numéricos a caracteres (letras, dígitos, signos de puntuación y caracteres de control), permitiendo la representación de texto en sistemas electrónicos.

Definición y Estructura

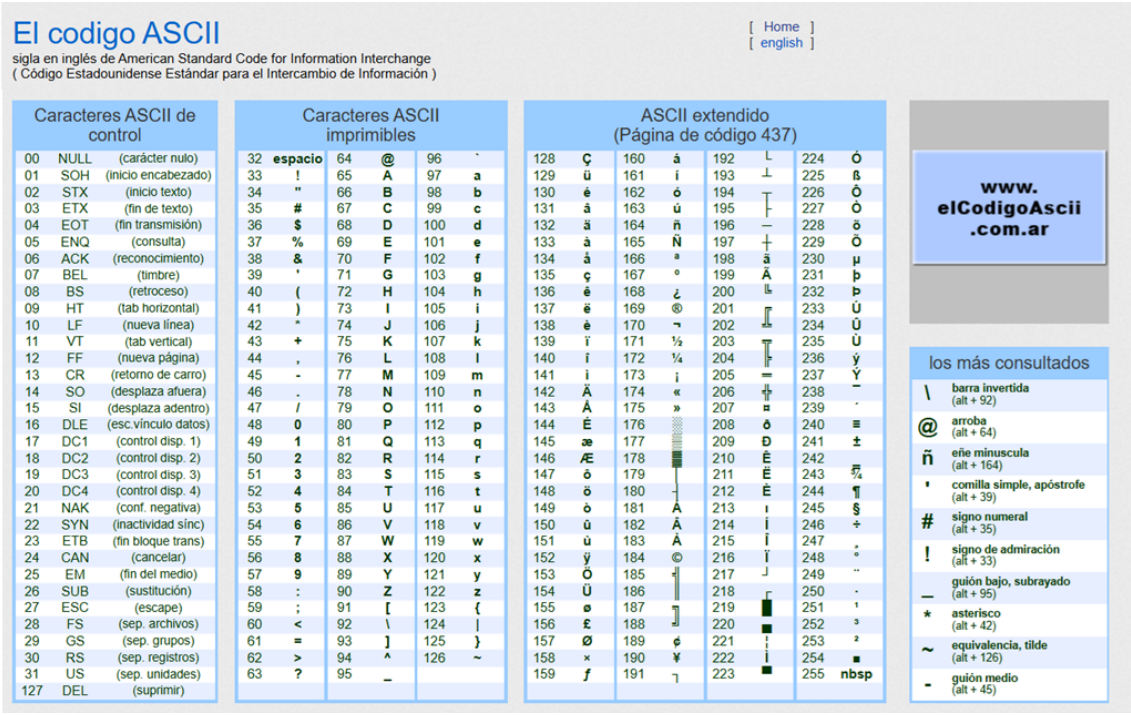
ASCII asigna un valor numérico a las letras, dígitos y símbolos. En su forma extendida, utiliza 8 bits (un Byte) para representar hasta 256 caracteres. Esta codificación es la que permite convertir un mensaje de texto secreto en la secuencia de Bytes que será incrustada en la imagen portadora.

Estructura de la codificación

La versión original de ASCII utiliza 7 bits, permitiendo representar 128 caracteres (de 0 a 127). Posteriormente, se introdujo la extensión a 8 bits para abarcar hasta 256 caracteres, incluyendo acentos y símbolos especiales.

Los primeros 32 códigos (0–31) corresponden a *caracteres de control* (por ejemplo, "salto de línea", "carro de retorno"). Los códigos del 32 al 126 representan caracteres imprimibles (espacio, signos, letras y dígitos). Por ejemplo:

$$65 \to \text{'A'}, \quad 97 \to \text{'a'}, \quad 48 \to \text{'0'}. \tag{2.1}$$



El codigo ASCII

sigla en inglés de American Standard Code for Information Interchange
(Código Estadounidense Estándar para el Intercambio de Información)

[Home]
[english]

Caracteres ASCII de control

00	NULL	(carácter nulo)
01	SOH	(inicio encabezado)
02	STX	(inicio texto)
03	ETX	(fin de texto)
04	EOT	(fin transmisión)
05	ENQ	(consulta)
06	ACK	(reconocimiento)
07	BEL	(timbre)
08	BS	(retroceso)
09	HT	(tab horizontal)
10	LF	(nueva línea)
11	VT	(tab vertical)
12	FF	(nueva página)
13	CR	(retorno de carro)
14	SO	(desplaza afuera)
15	SI	(desplaza adentro)
16	DLE	(esc.vínculo datos)
17	DC1	(control disp. 1)
18	DC2	(control disp. 2)
19	DC3	(control disp. 3)
20	DC4	(control disp. 4)
21	NAK	(conf. negativa)
22	SYN	(inactividad sínc)
23	ETB	(fin bloque trans)
24	CAN	(cancelar)
25	EM	(fin del medio)
26	SUB	(sustitución)
27	ESC	(escape)
28	FS	(sep. archivos)
29	GS	(sep. grupos)
30	RS	(sep. registros)
31	US	(sep. unidades)
127	DEL	(suprimir)

Caracteres ASCII imprimibles

32	espacio	64	@	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[	123	{
60	<	92	\	124	|
61	=	93	]	125	}
62	>	94	^	126	~
63	?	95	_		

ASCII extendido (Página de código 437)

128	Ç	160	á	192	└	224	Ó
129	ü	161	í	193	┴	225	ß
130	é	162	ó	194	┬	226	Ô
131	â	163	ú	195	├	227	Ò
132	ä	164	ñ	196	─	228	õ
133	à	165	Ñ	197	┼	229	Õ
134	å	166	ª	198	ã	230	µ
135	ç	167	º	199	Ã	231	þ
136	ê	168	¿	200	╚	232	Þ
137	ë	169	®	201	╔	233	Ú
138	è	170	¬	202	╩	234	Û
139	ï	171	½	203	╦	235	Ù
140	î	172	¼	204	╠	236	ý
141	ì	173	¡	205	═	237	Ý
142	Ä	174	«	206	╬	238	¯
143	Å	175	»	207	¤	239	´
144	É	176	░	208	ð	240	≡
145	æ	177	▒	209	Ð	241	±
146	Æ	178	▓	210	Ê	242	‗
147	ô	179	│	211	Ë	243	¾
148	ö	180	┤	212	È	244	¶
149	ò	181	Á	213	ı	245	§
150	û	182	Â	214	Í	246	÷
151	ù	183	À	215	Î	247	¸
152	ÿ	184	©	216	Ï	248	°
153	Ö	185	╣	217	┘	249	¨
154	Ü	186	║	218	┌	250	·
155	ø	187	╗	219	█	251	¹
156	£	188	╝	220	▄	252	³
157	Ø	189	¢	221	¦	253	²
158	×	190	¥	222	Ì	254	■
159	ƒ	191	┐	223	▀	255	nbsp

www.
elCodigoAscii
.com.ar

los más consultados

\	barra invertida (alt + 92)
@	arroba (alt + 64)
ñ	eñe minuscula (alt + 164)
'	comilla simple, apóstrofe (alt + 39)
#	signo numeral (alt + 35)
!	signo de admiración (alt + 33)
_	guión bajo, subrayado (alt + 95)
*	asterisco (alt + 42)
~	equivalencia, tilde (alt + 126)
-	guión medio (alt + 45)

[18].

Figura 2.7: Ejemplo de tabla ascii.

Importancia en ocultamiento de texto

ASCII es relevante en esteganografía cuando el mensaje oculto es textual, pues cada carácter puede codificarse en binario (7 u 8 bits) para luego distribuir sus bits dentro del medio portador, de forma que la información se oculte de manera imperceptible.

2.2. Codificación de Huffman

La Codificación de Huffman es un algoritmo de codificación de entropía para la **compresión de datos sin pérdida** [**Huffman1952method**]. Asigna códigos binarios de **longitud variable** a los símbolos, basándose en su frecuencia de aparición.

2.2.1. Historia

El algoritmo fue desarrollado por David A. Huffman en 1952, mientras era estudiante de doctorado en el MIT [**Huffman1952method**]. Desde su publicación, se convirtió en la base para casi todos los métodos de compresión de datos sin pérdida en informática.

2.2.2. Definición del problema y principio básico

El objetivo del algoritmo de Huffman es construir un código binario prefijo con la menor longitud media posible, dado un conjunto de símbolos

$$S = s_1, s_2, \ldots, s_n \tag{2.2}$$

y sus respectivas probabilidades o pesos

$$P = p_1, p_2, \ldots, p_n \tag{2.3}$$

donde se cumple que $\sum_i p_i = 1$.

Formalmente, se busca minimizar la longitud promedio del código L:

$$L = \sum_{i=1}^{n} p_i \, l_i, \tag{2.4}$$

donde l_i es la longitud (en bits) del código asignado al símbolo s_i.

Huffman demostró que su algoritmo genera un código C tal que $L(C)$ es mínimo entre todos los códigos posibles, garantizando una compresión óptima para la distribución de probabilidades dada [**wikipedia_Huffman**].

2.2.3. Pasos del proceso de codificación

El proceso de compresión basado en Huffman consta de los siguientes pasos:

a) Calcular la frecuencia de aparición de cada nivel de gris en la imagen.

b) Construir el árbol de Huffman utilizando las frecuencias obtenidas.

c) Asignar códigos binarios a cada valor según su posición en el árbol.

d) Codificar la imagen sustituyendo cada valor por su código correspondiente.

Este método permite reducir el tamaño final del archivo manteniendo la fidelidad de los datos originales. Este enfoque puede extenderse también a imágenes de mayor resolución (como la imagen Portadora) o incluso a otros tipos de datos que presenten redundancias similares.

2.2.4. Eficiencia, Entropía y Aplicaciones

El algoritmo de Huffman está directamente relacionado con la **Entropía de Shannon** (H), el límite teórico inferior de compresión.

a) **Entropía (H):** Mide la información promedio contenida en la fuente:

$$H = -\sum_{i=1}^{n} p_i \log_2(p_i) \tag{2.5}$$

El signo negativo se incluye porque las probabilidades p_i toman valores entre 0 y 1, por lo que $\log_2(p_i)$ es siempre negativo. Sin el signo menos, la suma total sería negativa, lo cual no tendría sentido físico, ya que la entropía representa una cantidad positiva de información o incertidumbre promedio. De este modo, el signo negativo asegura que $H \geq 0$ y que valores mayores de entropía correspondan a fuentes más impredecibles.

b) **Óptimalidad:** La codificación de Huffman es **óptima** porque su longitud de código promedio ($\bar{L}$) se acerca al valor de la entropía:

$$H \leq \bar{L} < H + 1 \tag{2.6}$$

El algoritmo es un pilar en la compresión de datos sin pérdida y se utiliza en formatos comunes como **JPEG**, **MP3**, **ZIP**, y en diversos protocolos de comunicación.

2.2.5. Ejemplo del Algoritmo de Huffman

Ejemplo 2.1. Para ilustrar el funcionamiento del algoritmo de Huffman, consideremos la siguiente palabra:

`Matemáticas`

Dividimos la palabra en caracteres individuales y contabilizamos la frecuencia de aparición de cada uno:

Carácter	Frecuencia
c	1
e	1
i	1
s	1
m	2
t	2
a	3

Tabla 2.3: Frecuencia de caracteres en la palabra `matematicas` ordenada de menor a mayor.

2.2.6. Construcción del Árbol de Huffman

A partir de las frecuencias, cada carácter se convierte en un nodo inicial. Se aplican iterativamente fusiones entre los nodos de menor frecuencia para formar Iniciamos con los primeros nodos:

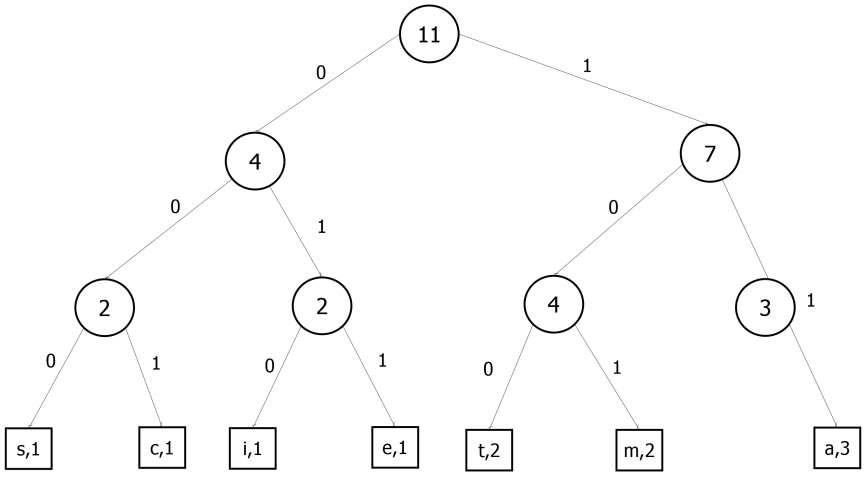


Figura 2.8: Árbol de Huffman final construido.

2.2.7. Códigos de Huffman Generados

Del árbol anterior, se extraen los siguientes códigos para cada carácter:

Carácter	Frecuencia	Código de Huffman
e	1	011
i	1	010
c	1	001
s	1	000
m	2	101
t	2	100
a	3	11

Tabla 2.4: Códigos de Huffman ordenados de menor a mayor frecuencia.

Codificación del Mensaje

Sustituimos cada letra en la palabra original por su código correspondiente:

```
m a t e m a t i c a s
```

```
101 11 100 011 101 11 100 010 001 11 000
```

Unimos los bits:

```
101111000111011110001000111000
```

Este ejemplo muestra cómo se logra una codificación más eficiente utilizando Huffman, asignando códigos más cortos a los caracteres más frecuentes.

2.2.8. Relación con el tamaño en bits y bytes

Cada símbolo de la palabra `matematicas` se ha representado con una secuencia binaria. La cadena completa contiene:

$$30 \text{ bits} \tag{2.7}$$

Dado que 1 byte = 8 bits, el tamaño total ocupado por el mensaje codificado es:

$$\frac{30}{8} = 3{,}75 \text{ bytes} \tag{2.8}$$

Si comparamos este resultado con una codificación estándar (por ejemplo, usando un byte por carácter en ASCII), la palabra original tendría:

$$11 \text{ caracteres} \times 8 \text{ bits} = 88 \text{ bits} = 11 \text{ bytes} \tag{2.9}$$

Por tanto, la codificación Huffman reduce el tamaño de 11 bytes a aproximadamente 3.75 bytes, logrando una compresión del:

$$\frac{11 - 3{,}75}{11} \times 100 \approx 65{,}9\,\% \tag{2.10}$$

Así, de la misma forma que en el ejemplo del texto, donde se redujo el número total de bits al usar códigos más cortos para los símbolos frecuentes, en una imagen o conjunto de datos más grande, el algoritmo de Huffman logra disminuir significativamente el tamaño requerido para almacenar o transmitir la información. Esto se debe a que los símbolos que aparecen con mayor frecuencia reciben códigos más cortos, mientras que los menos frecuentes reciben códigos más largos, optimizando así la codificación global y aumentando la eficiencia de la compresión.

2.2.9. Cálculo de la Entropía y Verificación de la Optimalidad del Código de Huffman

Para ilustrar el funcionamiento del algoritmo de Huffman, se empleó la palabra `matematicas`. A partir de ella se calcularon las frecuencias de cada símbolo, se construyó el árbol binario prefijo correspondiente y finalmente se obtuvieron los códigos asociados a cada carácter. El objetivo de esta sección es complementar dicho ejemplo mediante el análisis teórico que permite evaluar la eficiencia del código obtenido: el cálculo de la Entropía de Shannon y la comparación con la longitud promedio real del código de Huffman.

Frecuencias y Probabilidades

La palabra `matematicas` contiene 11 caracteres. Las frecuencias observadas y sus probabilidades asociadas son:

Carácter	Frecuencia	Probabilidad p_i
c	1	$\frac{1}{11}$
e	1	$\frac{1}{11}$
i	1	$\frac{1}{11}$
s	1	$\frac{1}{11}$
m	2	$\frac{2}{11}$
t	2	$\frac{2}{11}$
a	3	$\frac{3}{11}$

Tabla 2.5: Probabilidades de aparición de cada carácter en `matematicas`.

Estas frecuencias son importantes porque el algoritmo de Huffman asigna códigos cortos a los símbolos más frecuentes y códigos largos a los menos frecuentes, con el fin de minimizar la longitud promedio del código.

Construcción del Árbol de Huffman (Idea General)

El algoritmo sigue los siguientes pasos:

a) Se ordenan los símbolos de menor a mayor frecuencia.

b) Se toman los dos símbolos menos frecuentes y se combinan en un nodo.

c) El nuevo nodo se reinserta en la lista y se repite el proceso, uniendo siempre los nodos de menor peso.

Cada unión asigna:

- bit 0 a la rama izquierda,
- bit 1 a la rama derecha.

Este proceso genera un **árbol binario prefijo**, es decir, ningún código es prefijo de otro, garantizando una lectura sin ambigüedades.

Obtención de los Códigos

Una vez construido el árbol, se recorren las ramas para obtener los códigos. Por ejemplo, moverse a la izquierda añade un "0" y a la derecha un "1". En este caso, la letra `a`, por ser la más frecuente, obtuvo el código más corto (2 bits), mientras que los símbolos menos frecuentes recibieron códigos de 3 bits.

2.2.10. Codificación de la Palabra Completa

Codificando cada carácter se obtiene:

$$\texttt{m a t e m a t i c a s} = 101\,11\,100\,011\,101\,11\,100\,010\,001\,11\,000$$

La secuencia resultante es:

`101111000110111110001000111000`

de longitud total **30 bits**.

En comparación, en ASCII cada carácter ocupa 8 bits; por lo tanto:

$$11 \text{ caracteres} \times 8 \text{ bits} = 88 \text{ bits}$$

La compresión lograda mediante Huffman es:

$$\frac{30}{88} = 0{,}341 \quad \Rightarrow \quad 65{,}9\,\% \text{ de reducción}$$

Esto es relevante para esteganografía:

Menos datos $\Rightarrow$ menos píxeles modificados $\Rightarrow$ mayor discreción.

2.2.11. Cálculo de la Entropía de Shannon

La entropía mide la incertidumbre promedio de la fuente:

$$H = -\sum_{i=1}^{n} p_i \log_2(p_i) \tag{2.11}$$

Sustituyendo los valores:

$$H = -\left(\frac{3}{11}\log_2\frac{3}{11} + 2\cdot\frac{2}{11}\log_2\frac{2}{11} + 4\cdot\frac{1}{11}\log_2\frac{1}{11}\right) \tag{2.12}$$

Evaluando término a término:

$$\begin{aligned}
\frac{3}{11}\log_2\frac{3}{11} &\approx -0{,}473,\\
2\cdot\frac{2}{11}\log_2\frac{2}{11} &\approx -0{,}845,\\
4\cdot\frac{1}{11}\log_2\frac{1}{11} &\approx -1{,}260.
\end{aligned}$$

Sumando los valores:

$$H \approx 2{,}578 \text{ bits/símbolo}$$

$$\boxed{H \approx 2{,}58 \text{ bits/símbolo}}$$

2.2.12. Comparación con la Longitud Promedio del Código de Huffman

Las longitudes asignadas por Huffman son:

Carácter	**Prob.** p_i	**Longitud** l_i **(bits)**
c	$\frac{1}{11}$	3
e	$\frac{1}{11}$	3
i	$\frac{1}{11}$	3
s	$\frac{1}{11}$	3
m	$\frac{2}{11}$	3
t	$\frac{2}{11}$	3
a	$\frac{3}{11}$	2

Tabla 2.6: Longitudes de los códigos de Huffman.

La longitud promedio se calcula como:

$$\bar{L} = \sum_{i=1}^{n} p_i l_i \tag{2.13}$$

Sustituyendo:

$$\bar{L} = \frac{3}{11} \cdot 2 + \frac{2}{11} \cdot 3 + \frac{2}{11} \cdot 3 + 4 \cdot \frac{1}{11} \cdot 3 = \frac{30}{11} \approx 2{,}727 \text{ bits/símbolo}$$

$$\boxed{\bar{L} \approx 2{,}73 \text{ bits/símbolo}}$$

2.2.13. Demostración de Optimalidad

Los resultados obtenidos cumplen la propiedad fundamental de los códigos de Huffman:

$$\boxed{H \leq \bar{L} < H + 1}$$

En este caso:

$$2{,}58 \leq 2{,}73 < 3{,}58$$

Esto confirma la **optimalidad del código de Huffman** para la palabra analizada: la longitud promedio real está extremadamente cerca del límite teórico mínimo. La diferencia entre ambos valores es menor a 0.2 bits por símbolo, lo que evidencia la alta eficiencia del proceso de compresión.

Además, para aplicaciones de esteganografía, esta eficiencia tiene un impacto directo:

$$65{,}9\,\%\ \text{menos datos} \ \Rightarrow\ 65{,}9\,\%\ \text{menos modificaciones en la imagen}$$

lo cual reduce patrones estadísticos indeseados y aumenta la discreción del método.

2.3. Desplazamientos Lógicos

Los **Shift Left** (*Desplazamiento a la Izquierda*, SHL) y **Shift Right** (*Desplazamiento a la Derecha*, SHR) son operaciones lógicas fundamentales en el procesamiento de datos binarios. Estas operaciones se utilizan para manipular bits de manera eficiente, ya sea desplazándolos hacia la izquierda o hacia la derecha dentro de un registro o palabra binaria [19].

2.3.1. Corrimiento a la Izquierda (SHL o $\ll$)

a) **Definición:** Desplaza todos los bits de una secuencia N posiciones hacia la izquierda. El bit más a la izquierda se pierde, y se rellena con ceros por la derecha.

b) **Efecto Aritmético:** Equivale a multiplicar el valor binario por 2^N, donde N es el número de posiciones desplazadas.

Ejemplo 2.2. Si el valor original es **5** (Binario: 00000101), al desplazarlo dos posiciones:

$$5 \ll 2 \tag{2.14}$$

Inicial: 0000 0101 (5 decimal)

Resultado: 0001 0100 (**20** decimal)

Confirmación matemática: $5 \times 2^2 = 20$.

2.3.2. Corrimiento a la Derecha (SHR o $\gg$)

Definición:

Desplaza todos los bits de una secuencia N posiciones hacia la derecha. El bit más a la derecha (LSB) se descarta, y se rellenan ceros por la izquierda. **Efecto Aritmético:** Equivale a la **división entera** del valor binario por 2^N.

Ejemplo 2.3. Si el valor original es **20** (Binario: 00010100), al desplazarlo dos posiciones:

$$20 \gg 2 \tag{2.15}$$

Inicial: 0001 0100 (20 decimal)

Resultado: 0000 0101 (**5** decimal)

Confirmación matemática: $20 \div 2^2 = 5$.

Original	1	1	1	0	1	0	1	0
Left shifted	1	1	0	1	0	1	0	0
Rigth shifted	0	1	1	1	0	1	0	1

Figura 2.9: Ejemplo visual de desplazamiento lógico

2.3.3. Aplicaciones Comunes

Estas operaciones tienen una alta eficiencia computacional y son cruciales en:

(a) **Optimización:** Reemplazo de multiplicaciones y divisiones lentas por operaciones de desplazamiento rápidas (por potencias de 2).

(b) **Manipulación de Bits:** Aislamiento o extracción de bits específicos dentro de un byte o palabra (por ejemplo, leer el valor de un canal de color o acceder a banderas de control).

(c) **Procesamiento de Datos:** Esenciales en algoritmos de codificación, decodificación, compresión de datos y procesamiento de imágenes.

2.4. Algoritmo para determinar las dimensiones rectangulares óptimas

Si queremos formar una imagen rectangular con un total de N píxeles, debemos elegir dos números enteros: el ancho a y la altura b. Estos deben cumplir:

$$a \times b = N,$$

es decir, el área total del rectángulo debe usar exactamente los N píxeles disponibles.

Como el mismo número N se puede factorizar de diferentes maneras, el problema consiste en escoger las dimensiones que generen un rectángulo lo más parecido posible a un cuadrado. Esto es deseable porque, para un área fija, las dimensiones más equilibradas dan como resultado una forma más compacta y con menor desperdicio.

Para medir qué tan cuadrado es un rectángulo, consideramos la diferencia entre sus lados:

$$\Delta(a, b) = |a - b|.$$

La mejor elección es entonces el par (a, b) que minimice esta diferencia. En términos prácticos, esto significa:

(a) Buscar el divisor de N más cercano a $\sqrt{N}$.

(b) Ese divisor será uno de los lados del rectángulo.

(c) El otro lado será N dividido entre ese divisor.

2.4.1. Ejemplos:

a) Para $N = 48$, como $\sqrt{48} \approx 6{,}9$, el divisor más cercano es 6. Las dimensiones óptimas son 6×8.

b) Para $N = 37$, que es un número primo, no hay divisores intermedios. La única forma sin desperdicio es 1×37.

2.4.2. Aplicación al caso de la imagen portadora

Un píxel ocupa 3 bytes, es decir, 24 bits.

Si en la imagen portadora podemos ocultar 3 bits por cada byte, entonces:

Para ocultar 24 bits, se necesitan 8 bytes de capacidad.

Como cada píxel de la imagen portadora ocupa 3 bytes:

a) 8 bytes no alcanzan a formar un número entero de píxeles, porque $8/3 \approx 2{,}66$.

b) Dado que no es posible usar "una fracción de píxel", se debe usar el siguiente número entero de píxeles completos: **3 píxeles**.

Estos 3 píxeles aportan:

$$3 \times 3 = 9 \text{ bytes},$$

los cuales son suficientes para almacenar los 8 bytes que necesitamos.

2.5. Least Significant Bit (LSB)

El **LSB** (*Least Significant Bit* o Bit Menos Significativo) es uno de los métodos más sencillos y populares de esteganografía digital.

2.5.1. Definición y Aplicación

Consiste en reemplazar el bit de menor valor posicional (el bit más a la derecha) del byte que codifica un píxel de la imagen portadora, con un bit del mensaje oculto. Dado que la modificación cambia el valor del byte en solo ± 1 (de 0 a 255), el cambio de color es imperceptible al ojo humano, cumpliendo el principio fundamental de la esteganografía: *no ser detectado*.

Por ejemplo, si el byte original es:

$$\text{Píxel Original} \rightarrow \text{Byte: } 10101010\mathbf{0} \quad (\text{LSB} = 0) \tag{2.16}$$

y el bit del mensaje es 1, el byte modificado será:

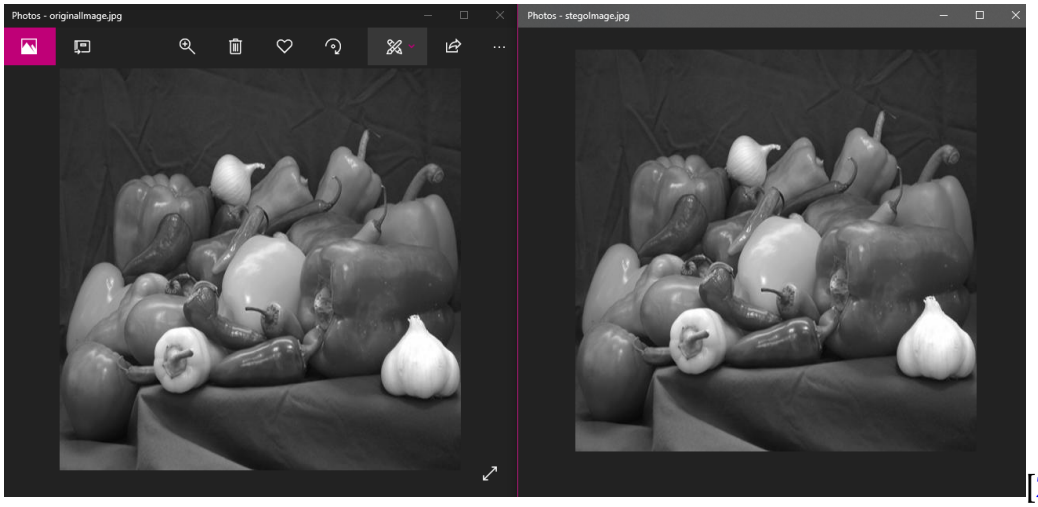
$$\text{Píxel Modificado} \rightarrow \text{Byte: } 10101010\mathbf{1} \quad (\text{LSB} = 1) \tag{2.17}$$

2.5.2. Técnicas de Incrustación

Existen dos técnicas comunes para incrustar información en el LSB:

a) **LSB Replacement:** Consiste en reemplazar directamente el LSB del byte con el bit del mensaje. Esta técnica es sencilla pero vulnerable a ataques de estegoanálisis debido a patrones estadísticos predecibles [20].

b) **LSB Matching:** Modifica el LSB del byte de manera que la diferencia entre el valor original y el modificado sea mínima, reduciendo la detectabilidad del cambio [20].



[21]

Figura 2.10: Ejemplo de ocultación de información en el LSB de una imagen.

2.5.3. Extracción de Información

La extracción del mensaje oculto se realiza leyendo los LSB de los bytes correspondientes en la imagen portadora. Los bits extraídos se agrupan para reconstruir el mensaje original.

2.6. Tecnologías relacionadas

El desarrollo e implementación del sistema esteganográfico propuesto se apoya en tecnologías y herramientas específicas para la manipulación eficiente de imágenes y la gestión de datos a bajo nivel.

2.6.1. Lenguaje C

El lenguaje de programación **C** es la base de los módulos de alta eficiencia crítica en el sistema, específicamente aquellos que requieren una manipulación precisa de bits y bytes. Su uso fue vital para la implementación de: El algoritmo de **Codificación y Decodificación de Huffman**, donde el acceso directo a la memoria y el manejo de estructuras de datos (árbol binario) es fundamental para la velocidad.

Las operaciones de **Desplazamiento Lógico** (*Shift Left* y *Shift Right*) para la reducción de la profundidad de color y la limpieza de los bits menos significativos (LSB) en la imagen portadora.

2.6.2. Lenguaje Python y Librería Pillow

El lenguaje **Python** se utiliza como el orquestador principal del proceso de esteganografía. La librería **Pillow** (*Python Imaging Library Fork*) proporciona las herramientas necesarias para: La lectura y escritura de archivos de imágenes en formato **BMP**.

El acceso directo y la manipulación a nivel de píxel de los canales de color (R, G, B), lo que facilita la implementación de la técnica **LSB** para la incrustación y extracción.

La obtención de metadatos de la imagen (dimensiones, profundidad de color), cruciales para el algoritmo de localización del área rectangular.

2.6.3. Formato de Imagen BMP

El formato **BMP** (Bitmap) fue seleccionado como el formato estándar para la Imagen Portadora y la Imagen a Ocultar. Este formato es ideal para la esteganografía LSB en el dominio espacial debido a dos características:

Es un formato que generalmente no utiliza compresión (o usa compresión sin pérdida simple), lo que garantiza que los cambios realizados en el LSB no sean eliminados o distorsionados por un algoritmo de compresión subsiguiente.

Permite un acceso directo a los valores de píxel (RGB), facilitando la manipulación de los bytes menos significativos.

2.7. Revisión de literatura

La esteganografía se enfoca en la ocultación de datos en medios digitales, como imágenes, audio o video, de manera que no genere sospechas. Por su parte, el estegoanálisis busca detectar la presencia de información oculta y prevenir posibles riesgos de seguridad informática [22]. Diversos estudios han explorado métodos tradicionales y avanzados para ocultar información, desde la sustitución de bits menos significativos (LSB) hast a enfoques más sofisticados que emplean inteligencia artificial [23]. La integración de esteganografía y criptografía ha permitido desarrollar sistemas más robustos, capaces de proteger datos sensibles y mantener la privacidad en comunicaciones digitales. Además, investigaciones recientes destacan la importancia de la detección y análisis forense de medios digitales mediante algoritmos de IA, debido al potencial uso malicioso de la esteganografía [24]. La literatura revisada evidencia que la disciplina está en constante evolución y que sus aplicaciones se extienden a la seguridad de la información, protección de derechos de autor y privacidad digital.

2.8. Trabajos relacionados

2.8.1. Estudio de Algoritmos de IA Aplicables al Estegoanálisis de Imágenes

La presente tesis titulada *"Estudio de Algoritmos de IA Aplicables al Estegoanálisis de Imágenes Digitales"*, realizada por el Ing. Sergio Alexis Ramírez Valle, aborda el campo de la esteganografía y el estegoanálisis dentro del ámbito de la seguridad informática y la ocultación de información.

La esteganografía se enfoca en la ocultación de datos en imágenes digitales sin levantar sospechas, mientras que el estegoanálisis busca detectar estos procesos de ocultación.

El estudio propone un método innovador basado en inteligencia artificial para la identificación de imágenes que han sido modificadas mediante técnicas de esteganografía. Para ello, se genera una sub-imagen con el fin de maximizar la detección del mensaje inyectado. Posteriormente, se extraen características de alta dimensión y se entrena un clasificador binario para diferenciar imágenes limpias de aquellas alteradas.

La investigación analiza y prueba los algoritmos HILL y WOW, comparando su desempeño con el enfoque SRM, reportado en la literatura. Los resultados obtenidos evidencian que el método propuesto ofrece un rendimiento competitivo en la detección de esteganografía en imágenes digitales, contribuyendo al desarrollo de nuevas estrategias en la seguridad informática y la inteligencia artificial aplicada [22].

2.8.2. Estudio y análisis de los procesos de la esteganografía en Colombia

Presenta una investigación en curso sobre la esteganografía en Colombia, destacando su importancia como tecnología avanzada en el ocultamiento de información.

La esteganografía permite esconder datos en imágenes, sonidos y otros medios digitales, dificultando su detección mediante técnicas computacionales. En este contexto, el estudio analiza métodos que favorecen el mimetismo de la información, explorando tanto su potencial como sus riesgos en el ámbito de la seguridad informática.

El objetivo principal de la investigación es examinar los procesos de esteganografía para comprender cómo se puede preservar la presencia de mensajes ocultos. Para ello, se plantean tres fases metodológicas:

(a) Revisión de literatura sobre técnicas de ocultamiento de información.

(b) Análisis de las prácticas de seguridad en esteganografía.

(c) Evaluación de las ventajas y desventajas de estas técnicas.

El artículo también revisa la relación entre esteganografía y criptografía, explicando cómo históricamente ambas disciplinas han evolucionado en el ámbito de la comunicación secreta. Se destacan ejemplos desde la antigüedad, como los mensajes ocultos en el cuero cabelludo en la Grecia clásica y el uso de tintas invisibles en el Renacimiento, hasta el desarrollo de herramientas modernas de estegoanálisis que buscan detectar información oculta en medios digitales.

Adicionalmente, el estudio aborda la historia de los criptosistemas, explorando métodos clásicos como el cifrado por transposición y el cifrado César, que fueron precursores de los sistemas actuales de encriptación. Se enfatiza la importancia de seguir investigando estos temas, especialmente ante el avance de la computación cuántica, que podría debilitar los sistemas de cifrado convencionales.

Finalmente, los resultados parciales del estudio evidencian un creciente interés académico en la esteganografía, con investigaciones en distintas áreas como seguridad de la información, ocultamiento de datos en medios digitales y técnicas de detección. Se concluye que la esteganografía sigue siendo un campo en expansión, con aplicaciones tanto positivas como riesgos potenciales en términos de seguridad y privacidad digital [23].

2.8.3. Esteganografía: el arte de ocultar información en medios digitales

El artículo analiza la esteganografía como una técnica fundamental para ocultar información dentro de medios digitales, destacando su papel en la protección de datos y la ciberseguridad.

A lo largo del estudio, se enfatiza el crecimiento constante de la investigación en este campo, evidenciado por el aumento en publicaciones académicas y el desarrollo de nuevas estrategias y algoritmos.

Desde una perspectiva teórica, se examinan las distintas técnicas esteganográficas utilizadas en la actualidad. Entre ellas, se encuentran métodos tradicionales como la sustitución del bit menos significativo (LSB), así como enfoques más sofisticados que emplean imágenes, audio y video como portadores de información oculta.

Además, se analiza la integración de la esteganografía con la criptografía para fortalecer la seguridad de los datos, lo que ha permitido desarrollar sistemas más robustos y resistentes a ataques.

El estudio también destaca la importancia de avanzar en las técnicas de detección de esteganografía digital, ya que su uso indebido podría representar un riesgo para la seguridad informática. En este sentido, se subraya la necesidad de implementar estrategias de análisis forense y herramientas basadas en inteligencia artificial para detectar posibles amenazas.

En términos metodológicos, la investigación sigue un enfoque cualitativo, basado en la revisión exhaustiva de literatura académica obtenida de bases de datos especializadas. Se identifican tendencias clave en el campo de la esteganografía, incluyendo sus aplicaciones prácticas en sectores como la protección de derechos de autor, la privacidad digital y la seguridad en redes sociales.

Los resultados del estudio muestran que la esteganografía es una disciplina en constante evolución, con aplicaciones cada vez más variadas y sofisticadas. La creciente adopción de estas técnicas en ámbitos como la protección de información personal, el resguardo de datos empresariales y la seguridad en comunicaciones digitales demuestra su relevancia en la era actual.

Finalmente, el artículo concluye que, si bien la esteganografía representa una herramienta poderosa para la protección de información, su potencial uso malicioso exige el desarrollo de mecanismos de detección efectivos. En este contexto, se plantea la necesidad de seguir investigando y mejorando los métodos de análisis y prevención para garantizar que esta tecnología sea utilizada de manera ética y segura [25].

2.8.4. An Extensive Survey of Digital Image Steganography: State of the Art

Este trabajo hace un análisis crítico de las técnicas actuales de esteganografía en imágenes digitales, de las tendencias innovadoras y de los retos en el campo. (ARXIV) [26].

2.8.5. Review and Test of Steganography Techniques

Este artículo revisa varias técnicas de esteganografía que se implementan en medios digitales, donde se evalúan estrategias, métodos y principios para la aplicación y detección de estas técnicas [24].

2.9. Análisis comparativo de soluciones existentes

En esta sección se realiza un análisis comparativo de los principales trabajos relacionados con la esteganografía y el estegoanálisis de imágenes digitales. El objetivo es identificar las fortalezas, limitaciones y áreas de oportunidad de cada enfoque, con el fin de justificar el método propuesto en esta investigación.

2.9.1. Comparación de metodologías

A continuación, se resumen los métodos más relevantes evaluados en los trabajos revisados:

Trabajo	**Método**	**Tipo de datos**	**Ventajas**	**Limitaciones**
Ramírez Valle (2020) [22]	IA: sub-imágenes + extracción de características + clasificador binario	Imágenes digitales	Alta precisión en detección de HILL y WOW; competitivo frente a SRM	Requiere gran capacidad computacional; depende de calidad del dataset
Estudio en Colombia [23]	Análisis de prácticas de seguridad y técnicas de esteganografía	Imágenes, audio y otros medios digitales	Proporciona marco teórico y contexto histórico; identifica riesgos	No implementa detección automática; enfoque más teórico
Arte de ocultar información [25]	Técnicas LSB, integración con criptografía	Imágenes, audio, video	Explica métodos tradicionales y avanzados; importancia de IA para detección	No realiza pruebas cuantitativas; falta comparación directa de algoritmos
Survey ARXIV [26]	Revisión crítica de técnicas de esteganografía	Imágenes digitales	Panorama actualizado; identifica tendencias y retos	Limitado a revisión; no propone método propio
Review and Test [24]	Evaluación de diversas técnicas de esteganografía	Imágenes digitales	Comparación de métodos; identificación de fortalezas y debilidades	No implementa IA; resultados dependen del dataset utilizado

Tabla 2.7: Comparación de soluciones existentes en estegoanálisis de imágenes

De la tabla anterior se destacan los siguientes puntos:

a) Los enfoques basados en IA, como el de Ramírez Valle, ofrecen mejor desempeño en la detección de mensajes ocultos, aunque requieren mayor capacidad de procesamiento.

b) La mayoría de los estudios enfatizan la necesidad de integrar esteganografía con criptografía para fortalecer la seguridad, pero pocos implementan métodos de detección automatizados.

c) Existen vacíos en la optimización de algoritmos para datasets grandes y heterogéneos, lo que justifica la propuesta de un enfoque que combine sub-imágenes, extracción de características de alta dimensión y clasificación binaria.

En conclusión, el análisis comparativo evidencia que si bien existen diversas soluciones para estegoanálisis de imágenes digitales, aún hay oportunidades de mejorar la precisión, eficiencia y aplicabilidad de los métodos. Esta investigación busca cubrir estos vacíos proponiendo un enfoque innovador basado en inteligencia artificial.

CAPÍTULO 3

DESARROLLO E IMPLEMENTACIÓN

Este capítulo describe el diseño, la arquitectura y los detalles de implementación del sistema esteganográfico propuesto. El sistema combina técnicas de compresión —reducción de profundidad de color y codificación Huffman— con el método de incrustación **LSB localizada**, garantizando eficiencia, reversibilidad e imperceptibilidad.

3.1. Análisis de Requisitos

El sistema fue diseñado para cumplir con los siguientes **requisitos funcionales**, derivados de los objetivos específicos planteados en esta investigación.

3.1.1. Requisitos de la Incrustación (Payload)

a) **Entrada y salida.** El sistema debe aceptar como entrada la *Imagen a Ocultar* y la *Imagen Portadora*, ambas en formato `.bmp`, y generar como salida el *Payload*.

b) **Compresión de doble etapa.** El mensaje secreto debe someterse a dos fases de compresión:

   (a) una **reducción de profundidad de color**, y

   (b) una **compresión sin pérdida** mediante el **algoritmo de Huffman**.

Como se muestra en la Figura 3.1, esta primera fase corresponde a la generación del payload a partir de la imagen secreta.

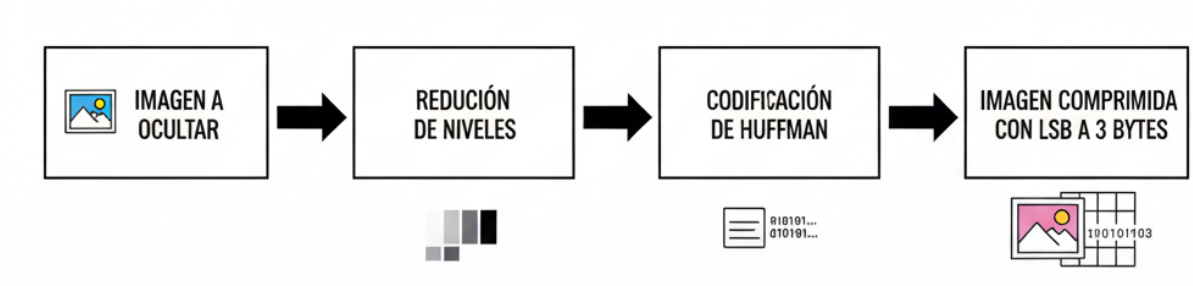


Figura 3.1: Fase 1: Incrustación (Payload).

3.1.2. Proceso de incrustación (Portadora)

El procedimiento de preparación de la imagen portadora es:

(a) **Corrimiento y limpieza LSB (SHR y SHL).** Se aplican operaciones de corrimiento de bits (`SHR` y `SHL`) sobre los componentes de color de la imagen portadora con el fin de limpiar los tres bits menos significativos de cada byte (R, G, B). Este proceso genera un espacio de tres bits por píxel, preparado para la posterior inserción de datos.

(b) **Cálculo del rectángulo máximo.** Mediante una función auxiliar en lenguaje C, se determinan las dimensiones óptimas $(a \times b)$ de la imagen necesarias para representar el total de bytes (N) del archivo comprimido `.huff`, verificando que se cumpla la condición $a \times b \leq N$. Esta etapa asegura la adecuación de la incrustación al espacio bidimensional de la imagen portadora.

(c) **Análisis de la capacidad de la imagen portadora.** Se evalúa la capacidad teórica de la imagen portadora para estimar el volumen máximo de información que puede contener, considerando el uso de un bit por componente de color, es decir, tres bits por píxel.

La preparación completa de la imagen portadora se muestra en la Figura 3.2.

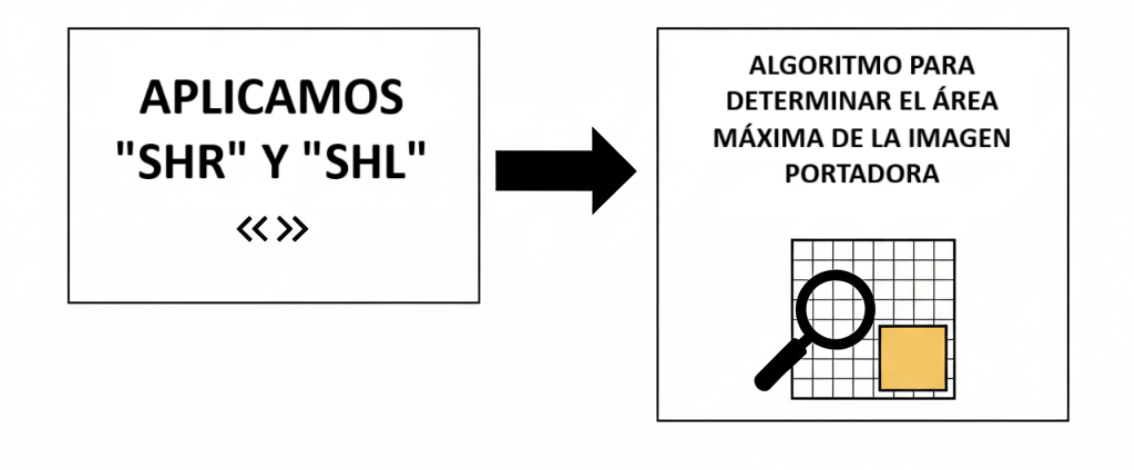


Figura 3.2: Fase 2: Preparación de la imagen portadora.

3.1.3. LSB e Incrustación

Incrustacion del archivo**payload** comprimido debe incrustarse en el bit menos significativo (LSB) de los canales RGB de la imagen portadora, dentro de un área rectangular óptima previamente calculada. Imperceptibilidad, las modificaciones aplicadas sobre la imagen portadora no deben ser perceptibles al ojo humano, lo cual se valida mediante análisis visual y comparación de histogramas. La Figura 3.3 muestra esta fase.

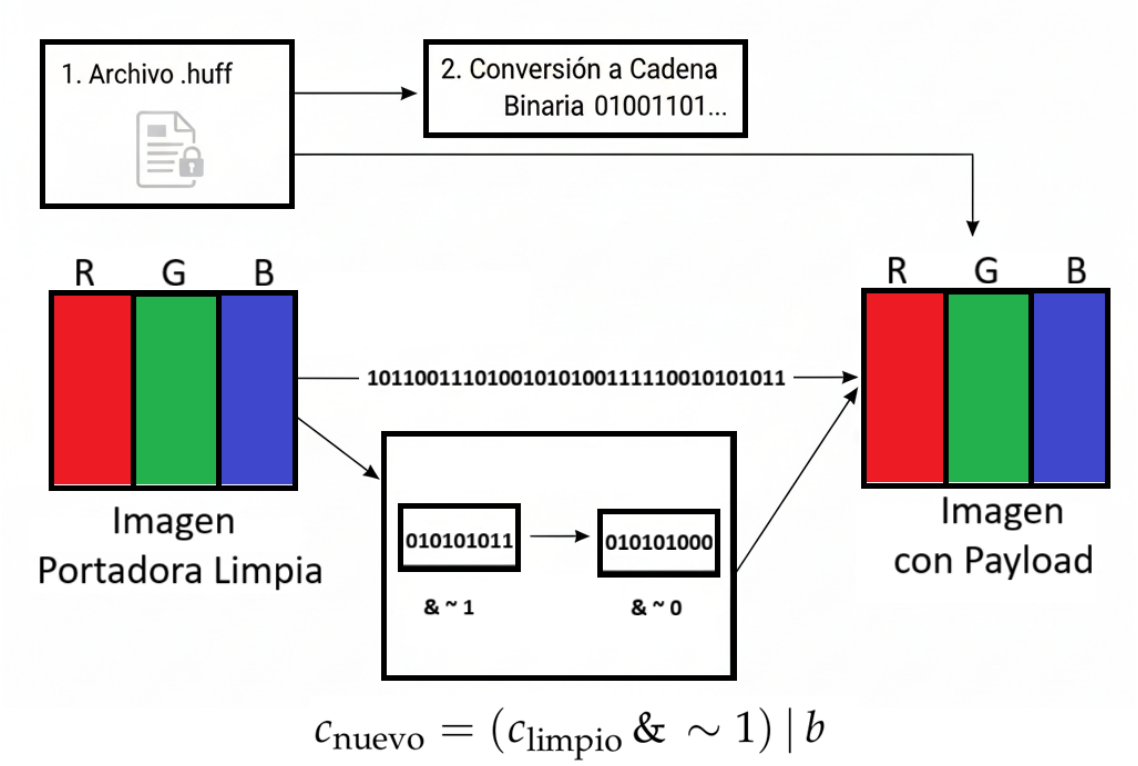


Figura 3.3: Fase 3: LSB e Incrustación

3.1.4. Requisitos de la Extracción (Reconstrucción)

Para poder recuperar la imagen original:

a) **Reversibilidad.** El proceso de extracción debe ser el inverso exacto de la incrustación, permitiendo recuperar el mensaje secreto sin pérdida de información.

b) **Descompresión y reconstrucción.** El sistema debe ejecutar la descompresión Huffman y restaurar la profundidad de color original del mensaje recuperado.

El sistema presenta una arquitectura modular dividida en dos grandes procesos: (1) Proceso de Incrustación y (2) Proceso de Extracción. La secuencia completa se muestra en la Figura 3.4.

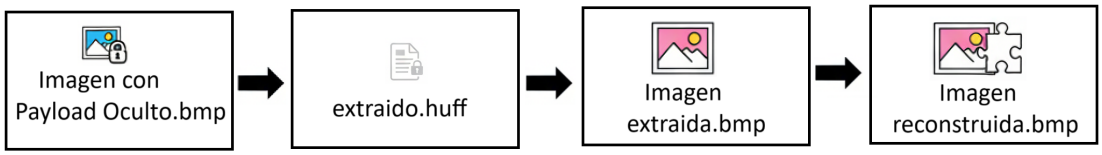


Figura 3.4: Fase 4: Reconstrucción del mensaje.

3.2. Diseño del Sistema

El sistema presenta una **arquitectura modular** dividida en dos grandes procesos: (1) el **Proceso de Incrustación**, y (2) el **Proceso de Extracción**.

3.2.1. Proceso de Incrustación

El flujo de datos sigue una arquitectura secuencial y modular, garantizando que el **payload** esté optimizado antes de su incrustación.

a) **Módulo de Preprocesamiento:**

1) La imagen a ocultar se procesa mediante `niveles.c`, que reduce su profundidad de color.
2) El resultado se comprime con `compresion.c` utilizando el algoritmo de Huffman, generando el **payload**.

b) **Módulo de Localización:**

1) El programa `rectangulo.c` calcula las dimensiones del área rectangular necesaria para alojar la imagen que se desea ocultar según su tamaño.
2) El módulo `shlr.c` limpia los bits menos significativos (LSB) de la imagen portadora dentro del área seleccionada.

c) **Módulo de Esteganografía (Python):**

1) El script `lsbimagen.py` lee la imagen portadora preprocesada y el **payload** comprimido.

2) Sustituye bit a bit el contenido del **payload** dentro de los LSB de los píxeles definidos, generando el Esteganograma (Payload).

3.2.2. Proceso de Extracción

El proceso de extracción revierte las operaciones anteriores en orden inverso.

a) **Módulo de Extracción LSB (Python):**

1) El script `extraerlsb1.py` analiza el Esteganograma y extrae la secuencia de bits del área rectangular, recuperando el **payload** comprimido.

b) **Módulo de Postprocesamiento:**

1) `descompresion.c` aplica la decodificación Huffman inversa, restaurando la versión reducida de la imagen oculta.
2) `niveles8.c` reconstruye la profundidad de color original para obtener la imagen final recuperada.

3.3. Detalles de Implementación

El desarrollo del sistema se realizó empleando dos lenguajes de programación principales, seleccionados por su eficiencia en distintos niveles de procesamiento.

3.3.1. Tecnologías Utilizadas

a) **Lenguaje C:** Utilizado para las operaciones de bajo nivel, donde la eficiencia en el manejo de bits es crítica. Incluye la implementación de los módulos de compresión y descompresión Huffman (`compresion.c`, `descompresion.c`) y la manipulación directa de bytes para la reducción de profundidad de color (`niveles.c`).

b) **Lenguaje Python:** Utilizado para el manejo de imágenes y la incrustación LSB. Se empleó la librería **Pillow** (PIL Fork) para acceder y modificar los valores RGB de los píxeles en archivos BMP.

3.4. Pruebas de Desempeño y Validación

La implementación fue validada mediante pruebas orientadas a evaluar su capacidad de compresión, reversibilidad e imperceptibilidad visual.

3.4.1. Prueba de Integridad (Reversibilidad)

La comparación entre la imagen reducida original y la imagen extraída/-descomprimida evidenció una diferencia de:

a) **Diferencia de bytes:** 0 bytes.

Este resultado confirma la **reversibilidad total** del proceso y la ausencia de pérdida de información.

3.4.2. Prueba de Estegoanálisis (Imperceptibilidad)

Para evaluar la invisibilidad del mensaje incrustado, se compararon los histogramas de la imagen portadora y del Esteganograma:

a) **Análisis visual:** No se observaron diferencias perceptibles entre ambas imágenes.

b) **Análisis estadístico:** Los histogramas presentan el patrón característico de "doble barra" o picos abruptos en los canales modificados, confirmando la efectividad de la incrustación sin comprometer la calidad visual.

CAPÍTULO 4

METODOLOGÍA

4.1. Enfoque metodológico

La metodología propuesta para la incrustación de una imagen secreta en un archivo portador se divide en tres fases principales:

a) Preparación de la imagen Payload.

b) Preparación del archivo Portador.

c) Aplicación de la técnica del Bit Menos Significativo (LSB) para la incrustación final.

Este enfoque utiliza la compresión de Huffman como técnica de preprocesamiento para maximizar la capacidad de almacenamiento y la invisibilidad del secreto.

4.2. Herramientas y tecnologías utilizadas

El desarrollo de los métodos de incrustación y extracción de información en imágenes digitales requirió el uso de diversas herramientas y tecnologías que permiten un manejo eficiente de los datos y la optimización del rendimiento de los algoritmos.

El lenguaje de programación **Python** se empleó como plataforma principal debido a su flexibilidad y a la disponibilidad de bibliotecas especializadas para el procesamiento de imágenes, lo que facilita la implementación de algoritmos bit a bit y la manipulación de matrices de píxeles de manera eficiente [27].

Adicionalmente, se utilizó el lenguaje **C** para el desarrollo de funciones auxiliares que demandan un alto rendimiento, tales como el cálculo de dimensiones óptimas para la inserción de datos y la decodificación mediante el **algoritmo de Huffman**. El uso de C permite un control preciso de la memoria y de operaciones de bajo nivel, optimizando el tiempo de ejecución y la eficiencia del sistema [28].

Para la manipulación de imágenes en formato BMP se recurrió a la biblioteca **Pillow (Python Imaging Library)**, que proporciona herramientas para la lectura, escritura y modificación de píxeles de forma controlada, facilitando la implementación de técnicas de esteganografía y procesamiento de imágenes [29].

El **algoritmo de Huffman** se implementó como método de compresión sin pérdida, lo que permite reducir el tamaño de la información a ocultar dentro de la imagen portadora sin comprometer su integridad [30]. Esta técnica asegura un almacenamiento eficiente y la preservación completa de los datos originales.

Por último, se aplicaron operaciones de desplazamiento de bits (**SHR** y **SHL**) para la manipulación precisa de los bits menos significativos de los píxeles. Estas operaciones son fundamentales en los procesos de incrustación bit a bit, garantizando una inserción controlada y reversible de la información sin afectar de manera perceptible la calidad visual de la imagen [31].

4.3. Diseño de la solución

El proceso de incrustación se basa en modificar los tres bits menos significativos (LSB) de cada componente de color (R, G, B) de la imagen portadora, de manera que los datos del mensaje secreto se inserten sin alterar perceptiblemente la apariencia visual.

La compresión de Huffman se aplica al mensaje antes de la incrustación, reduciendo su tamaño y optimizando la eficiencia del método.

4.4. Fases del desarrollo

4.4.1. Preparación del Mensaje Secreto

Reducción de Niveles de Color (Preprocesamiento). Se implementa el corrimiento de bits a la derecha (`SHR`) y luego a la izquierda (`SHL`) sobre los tres bits menos significativos de cada componente de color de la imagen a ocultar.

Este proceso reduce los niveles de color de $2^8 = 256$ a $2^5 = 32$, optimizando la posterior compresión.

Compresión de Huffman. Se aplica el algoritmo de compresión sin pérdida de Huffman al archivo de niveles reducidos de la imagen a ocultar. Este proceso genera el flujo de bits secreto, el árbol de decodificación y el encabezado en un solo archivo binario (`.huff`). Como se muestra en la Figura 3.1, esta primera fase corresponde a la generación del payload a partir de la imagen secreta.

4.4.2. Preparación del Archivo Portador

Corrimiento y Limpieza LSB (SHR y SHL). Se aplica la lógica de corrimiento de bits (`SHR` y `SHL`) de tres posiciones a los componentes de color de la imagen portadora. Esto establece un valor de cero en los tres bits menos significativos de cada byte de color (R, G, B), creando el espacio de 3 bits por píxel limpio y listo para la incrustación.

Cálculo del Rectángulo Máximo. Se utiliza una función auxiliar en C para determinar las dimensiones óptimas $(a \times b)$ que puede tener una imagen para representar el total de bytes (N) del archivo `.huff`, cumpliendo que $a \times b \leq N$. Esta verificación valida si la incrustación es adecuada al espacio bidimensional de la portadora.

Análisis de Capacidad de la Imagen Portadora. Se analiza la capacidad teórica de la imagen portadora para determinar el volumen de datos que puede alojar utilizando 1 bit por componente de color (es decir, 3 bits por píxel). La preparación completa de la imagen portadora se muestra en la Figura 3.2.

4.4.3. LSB e Incrustación

En esta fase se realiza la incrustación bit a bit del archivo Huffman en los 3 bits menos significativos de cada píxel de la imagen portadora limpia.

El script en Python `hide_huff_in_image.py` implementa el proceso bit a bit de incrustación, realizando las siguientes acciones:

1. **Lectura del archivo .huff:** Se convierte el archivo Huffman en una cadena binaria de bits.
2. **Iteración sobre la imagen portadora:** Se recorren los componentes de color rojo (R), verde (G) y azul (B) de cada píxel.

3. **Incrustación de bits:** Para cada componente de color c se inserta el bit b correspondiente mediante:

$$c_{\text{nuevo}} = (c_{\text{limpio}} \,\&\, \sim 1) \mid b \tag{4.1}$$

donde:

1) c_{limpio}: valor del componente de color con sus 3 LSB en cero (preparados en la Fase II).
2) $\& \sim 1$: limpia el bit menos significativo (lo establece en 0).
3) $|b$: inserta el bit secreto en el LSB.

Este método permite almacenar un mensaje binario extenso dentro de la imagen portadora de forma segura y discreta, sin alterar significativamente su apariencia visual. El payload comprimido se incrusta en los bits menos significativos (LSB) de los canales RGB dentro del área rectangular óptima. La Figura 3.3 muestra esta fase.

4.4.4. Extracción

Una vez completado el proceso de ocultamiento del archivo comprimido, se procede con las etapas inversas para recuperar la información original. Esta fase comprende tres pasos principales: extracción del archivo comprimido, descompresión de Huffman y reconstrucción de niveles de gris.

Extracción del Archivo Huffman desde la Imagen Esteganográfica. Se recorre la imagen píxel por píxel, extrayendo el bit menos significativo (LSB) de cada componente RGB y reconstruyendo el flujo binario hasta detectar un marcador de fin definido por el patrón:

$$\text{Marcador de fin} = 64 \times' 1' + 8 \times' 0' \tag{4.2}$$

El archivo extraído se guarda como `extraido.huff`, el cual servirá de entrada para la descompresión.

Descompresión Huffman. Se aplica el algoritmo de decodificación Huffman para restaurar la imagen reducida. A partir del archivo `extraido.huff`, el programa `descomprimir.c` reconstruye los valores de intensidad originales en formato BMP, generando `imagen_reducida.bmp`:

$$\text{Huffman: } D(\texttt{extraido.huff}) \rightarrow \texttt{imagen_reducida.bmp} \tag{4.3}$$

Reconstrucción de Niveles de Gris. Dado que la imagen reducida presenta solo 32 niveles por canal (5 bits), se realiza una expansión tonal para mejorar el contraste visual mediante:

$$c_{\text{rec}} = (c \,\&\, 0xF8) + 4 \tag{4.4}$$

donde c representa la intensidad reducida. Esta transformación aproxima el rango perdido durante la cuantización, mejorando la suavidad de los tonos.

El resultado, `imagen_reconstruida.bmp`, presenta una apariencia muy cercana a la original, aunque no idéntica debido a la pérdida de información inicial.

La secuencia completa se muestra en la Figura 3.4.

4.5. Recursos Necesarios

La implementación y evaluación de los métodos de incrustación y extracción de información en imágenes digitales requiere disponer de ciertos recursos de hardware y software que aseguren un desempeño adecuado y permitan la validación de los resultados obtenidos.

En términos de hardware, es necesario contar con un equipo de cómputo que posea capacidad de procesamiento moderada y recursos gráficos suficientes para la manipulación de imágenes de tamaño estándar, así como para la ejecución eficiente de algoritmos de compresión y esteganografía. Esta configuración garantiza que las operaciones de procesamiento y prueba se realicen de manera fluida, minimizando posibles cuellos de botella en el rendimiento.

A nivel de software, se requiere un entorno de desarrollo integrado (IDE) compatible con los lenguajes **Python** y **C**, lo que facilita la implementación, depuración y ejecución de los programas. Asimismo, se emplean librerías especializadas, tales como `Pillow` para la manipulación de imágenes, `NumPy` para operaciones numéricas y manejo de matrices, y un compilador `GCC` para la ejecución eficiente del código en C. Estas herramientas permiten una integración efectiva entre los diferentes módulos del sistema y aseguran la correcta manipulación de los datos.

Para la validación experimental de los métodos desarrollados, se requiere un conjunto de imágenes de prueba en formatos BMP o PNG. Dichas imágenes permiten evaluar tanto la capacidad de incrustación y extracción de información como la preservación de la calidad visual de la imagen portadora.

Finalmente, es indispensable contar con espacio de almacenamiento suficiente para alojar los archivos generados, incluidos los datos comprimidos, los resultados visuales y las imágenes de prueba. Un manejo organizado de estos recursos asegura la trazabilidad y reproducibilidad de los experimentos realizados.

CAPÍTULO 5

RESULTADOS Y ANÁLISIS

5.1. Presentación de Resultados

En este capítulo se presentan las evidencias obtenidas durante la ejecución de cada una de las fases del proceso de incrustación, compresión, extracción y reconstrucción de la imagen payload. El procedimiento fue implementado mediante una combinación de técnicas de compresión Huffman y esteganografía LSB (*Least Significant Bit*).

Todas las pruebas se llevaron a cabo en el entorno `MobaXterm`, utilizando los compiladores `gcc` para el código en C y el intérprete de Python 3 para los scripts de esteganografía. Esta configuración asegura la reproducibilidad de los resultados y permite una evaluación consistente de la eficacia de los métodos implementados.

5.2. Fase 1: Reducción de Niveles y Compresión Huffman

El objetivo de esta fase es preparar y optimizar el mensaje secreto para su posterior incrustación en la imagen portadora. Para ello, se realizan procesos de reducción de niveles de color y compresión mediante el algoritmo de Huffman, lo que permite minimizar el tamaño del *payload* sin pérdida de información, asegurando así un almacenamiento esteganográfico más eficiente y seguro.

5.2.1. Ejecución del programa `niveles.c`

El programa `niveles.c` se emplea para reducir la cantidad de niveles de color de la imagen *Payload*. Para ello, realiza desplazamientos de bits hacia la derecha (SHR) o hacia la izquierda (SHL) en cada componente de color, con el objetivo de simplificar la representación de los valores de intensidad. Esta reducción incrementa la redundancia en los datos de la imagen, lo que a su vez permite mejorar el ratio de compresión en etapas posteriores del proceso. Como resultado, se obtiene una versión de la imagen optimizada para técnicas de compresión sin comprometer significativamente la percepción visual.

```
gcc niveles.c
./a <imagen_entrada> <archivo_reducida> <archivo_verificacion>
```

Nº	Imagen Payload	Valor Máximo	Bits útiles
1	Lenna.bmp	240	9
2	Baboon.bmp	248	9
3	Flower.bmp	248	9
4	BoatsColor.bmp	248	9
5	Pepper.bmp	232	9
6	Barbara.bmp	248	9

Tabla 5.1: Resultados de reducción de niveles

Resultados principales:

Se verificó que los tres bits menos significativos de cada componente RGB de la imagen payload, presentan un valor de cero, lo que garantiza que es posible almacenar hasta 9 bits por píxel para el *payload*. Para validar este procedimiento, se generó una imagen de verificación denominada `imagen_verificacion.` en la cual se comprobó que los valores de color cumplen con los criterios establecidos para la incrustación segura de información.

5.2.2. Ejecución del programa `compresion.c`

En esta etapa se aplica el programa `compresion.c` sobre la imagen previamente reducida en sus niveles de color. El programa implementa el algoritmo de compresión sin pérdida de Huffman, el cual codifica los datos de la imagen optimizando la representación de los símbolos según su frecuencia de aparición. Como resultado de este proceso, se genera un archivo binario con extensión `.huff`, que sirve como *payload* para la posterior incrustación esteganográfica. Este procedimiento asegura que la información se comprima de manera eficiente sin pérdida de datos, facilitando su transmisión o almacenamiento dentro de la imagen portadora.

```
gcc compresion.c
./a <archivo_reducida> <archivo_salida.huff>
```

Imagen	Original (bytes)	Compresión (bytes)	Ratio (%)
Lenna.bmp	786432	468157	40.47
Baboon.bmp	720000	409420	43.14
Flower.bmp	737280	437818	40.62
BoatsColor.bmp	1359936	748941	44.93
Pepper.bmp	786432	470216	40.21
Barbara.bmp	1244160	742444	40.33

Tabla 5.2: Resultados de compresión Huffman

Nota: El ratio de compresión obtenido confirma la eficacia de la reducción de niveles de color, ya que al incrementar la redundancia en los datos, se logra disminuir el tamaño del *payload* sin comprometer significativamente la calidad visual de la imagen. Este resultado valida la estrategia utilizada para optimizar la compresión antes de la inserción esteganográfia.

5.3. Fase 2: Preparación de la Imagen Portadora

El objetivo de esta fase es preparar la imagen portadora para la incrustación de información, asegurando que el espacio esteganográfico disponible sea adecuado y que se minimice cualquier interferencia con los datos originales.

5.3.1. Ejecución del programa `shlr.c`

El programa `shlr.c` se utiliza para limpiar los tres bits menos significativos (LSB) de cada componente de color (RGB) en la imagen portadora. Esta operación garantiza que el espacio destinado a la incrustación del *payload* esté disponible y no se mezclen datos residuales de la imagen original. Al manipular únicamente los bits menos significativos, se preserva la fidelidad visual de la imagen mientras se maximiza la capacidad de almacenamiento esteganográfico

```
gcc shlr.c
./a <imagen_entrada> <archivo_limpio>
Corrimiento (SHR/SHL) aplicado.
Archivo limpio guardado en: <archivo_limpio>
```

Nota: La limpieza de los tres bits menos significativos de cada componente RGB asegura que la imagen portadora cuente con espacio suficiente para incrustar el *payload* sin riesgo de sobrescribir información residual. Además, al modificar únicamente los LSB, se mantiene prácticamente inalterada la percepción visual de la imagen, garantizando la efectividad del método esteganográfico.

5.3.2. Ejecución del programa `rectangulo.c`

El programa `rectangulo.c` se utiliza para calcular el área máxima disponible en la imagen portadora para el almacenamiento del *payload*. Este cálculo se realiza considerando las dimensiones de la imagen y permite determinar la capacidad máxima de incrustación sin comprometer la integridad visual de la misma. De esta manera, se asegura que el *payload* pueda ser incorporado de forma eficiente y segura dentro de los límites esteganográficos de la imagen.

```
gcc rectangulo.c
./a <archivo_limpio>
```

Se ocupa el archivo anterior obtenido, para poder encontrar el area maxima del la imagen *Portadora*

Nº	Portadora	Dimensiones	Capacidad total (bytes)	Tamaño Payload (bytes)
1	Mickey.bmp	1600x1200	720,000	468,157
2	Bosque.bmp	3846x2400	3,456,000	409,420
3	Montaña.bmp	2560x1440	1,382,400	437,818
4	Flor.bmp	1920x1200	864,000	748,941
5	Frutas.bmp	1600x1067	640,200	470,216
6	lago.bmp	4488x1777	2,990,691	742,444

Tabla 5.3: Capacidad de almacenamiento de cada imagen portadora

Nota: La capacidad teórica, expresada en bytes, de la imagen portadora es suficiente para albergar el *payload* comprimido, lo que garantiza que el proceso de incrustación puede realizarse sin pérdida de información ni saturación del espacio esteganográfico disponible.

5.4. Fase 3: Incrustación mediante LSB

En esta fase se demuestra la reversibilidad y la integridad del proceso completo de esteganografía. La incrustación se realiza utilizando el método LSB (*Least Significant Bit*) para incorporar el archivo Huffman dentro de la imagen portadora previamente preparada y limpia. Posteriormente, se procede a su verificación inmediata para asegurar que los datos se han incrustado correctamente sin afectar la fidelidad visual de la imagen.

5.4.1. Ejecución del programa `lsbimagen.py`

El programa `lsbimagen.py` permite incrustar el archivo Huffman dentro de la imagen portadora limpia. Durante este proceso, los bits menos significativos de cada componente RGB se sustituyen por los bits correspondientes del *payload*, asegurando que la información se almacene de manera imperceptible y reversible.

```
Python lsbimagen.py
<archivo_huff> <imagen_portadora> <imagen_salida>
```

Nº	Portadora	Payload	Bytes Ocultados	Resultado
1	Mickey_limpia.bmp	lenna.huff	468,273	Éxito
2	Bosque_limpia.bmp	baboon.huff	409,536	Éxito
3	Montaña_limpia.bmp	flower.huff	437,937	Éxito
4	Flor_limpia.bmp	BoatsColor.huff	749,060	Éxito
5	frutas_limpia.bmp	pepper.huff	470,329	Éxito
6	lago_limpia.bmp	barbara.huff	742,563	Éxito

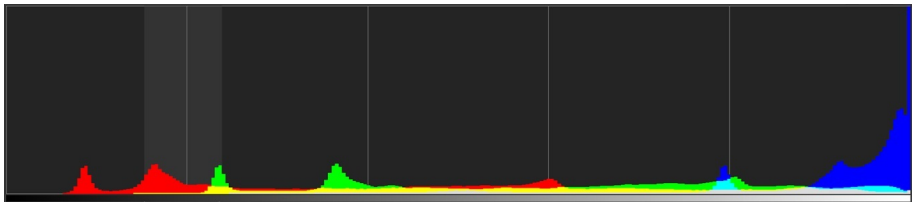
Tabla 5.4: Resultados de incrustación LSB

5.4.2. Comparación Visual de Incrustración

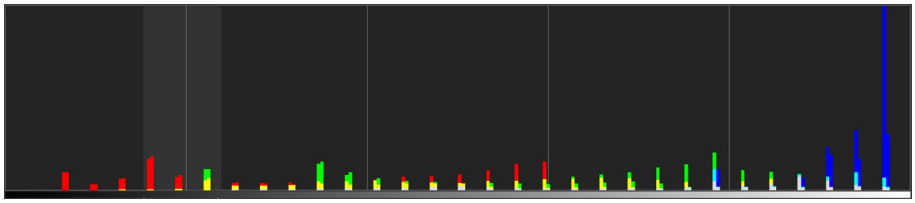
A continuación, se presentan las imágenes originales junto con las imágenes procesadas que contienen el payload incrustado. Estas imágenes permiten observar visualmente los efectos de la técnica de esteganografía digital LSB aplicada, desarrollada para optimizar la capacidad de ocultamiento mediante la combinación de reducción de profundidad de color, codificación de Huffman y selección del área rectangular óptima dentro de imágenes de mapa de bits.

(a) Mickey Limpia 1600x1200 (b) Mickey con Payload

Figura 5.1: Comparación de Mickey: Portadora vs Payload [32]



(a) Histograma de Imagen Original

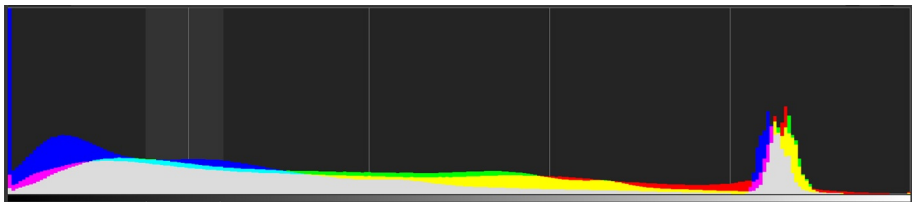


(b) Histograma de Imagen con Payload

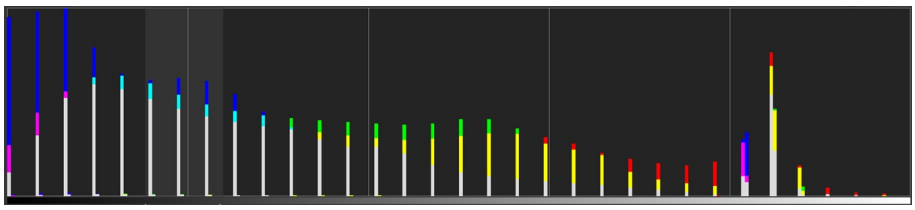
Figura 5.2: Mickey: Histograma Original vs. Payload

(a) Bosque Limpia 3840x2400 (b) Bosque con Payload

Figura 5.3: Comparación de Bosque: Portadora vs Payload [33]



(a) Histograma de Imagen Original

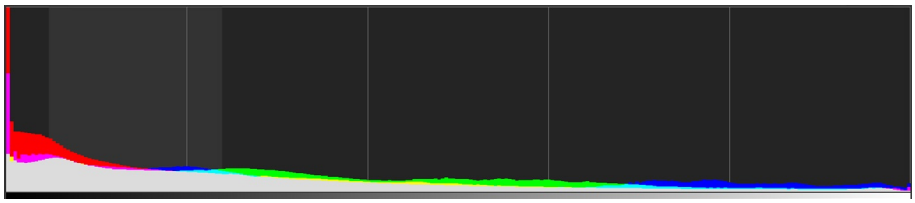


(b) Histograma de Imagen con Payload

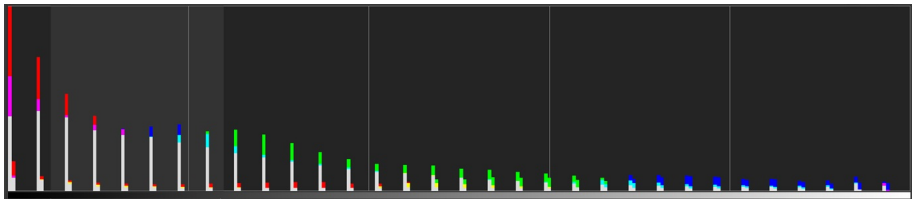
Figura 5.4: Bosque: Histograma Original vs. Payload

(a) Montaña Limpia 2560x1440 (b) Montaña con Payload

Figura 5.5: Comparación de Montaña: Portadora vs. Payload [34]



(a) Histograma de Imagen Original

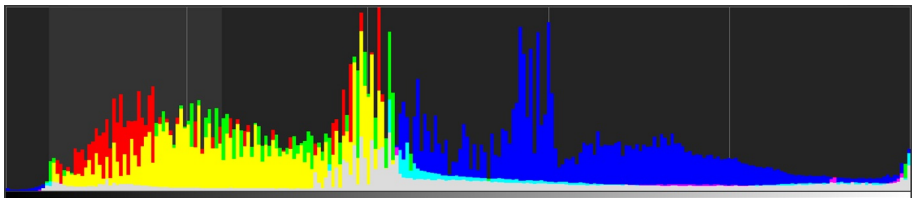


(b) Histograma de Imagen con Payload

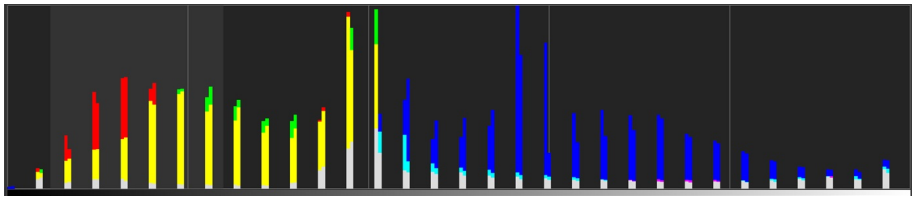
Figura 5.6: Montaña: Histograma Original vs. Payload

(a) Flor Limpia 1920x1200 (b) Flor con Payload

Figura 5.7: Comparación de Flor: Portadora vs. Payload [35]



(a) Histograma de Imagen Original

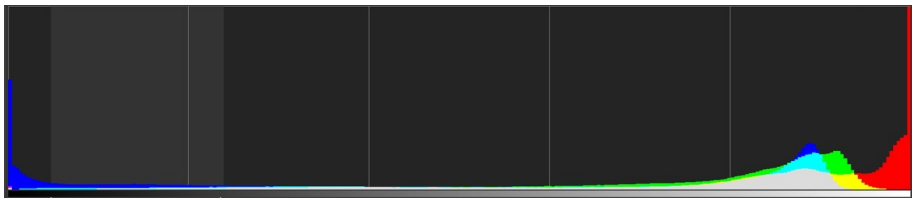


(b) Histograma de Imagen con Payload

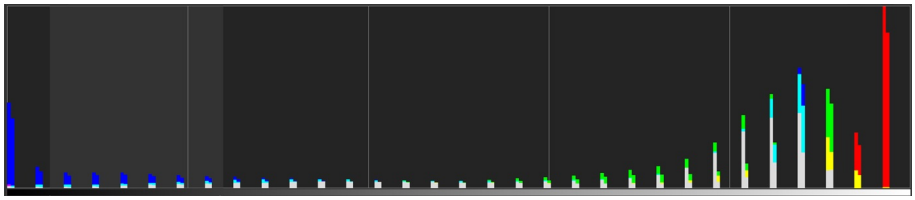
Figura 5.8: Flor: Histograma Original vs. Payload

(a) Fruta Limpia 1600x1067 (b) Fruta con Payload

Figura 5.9: Comparación de Fruta: Portadora vs. Payload [36]



(a) Histograma de Imagen Original



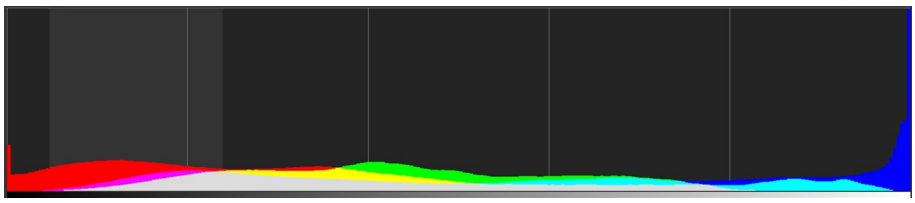
(b) Histograma de Imagen con Payload

Figura 5.10: Frutas: Histograma Original vs. Payload

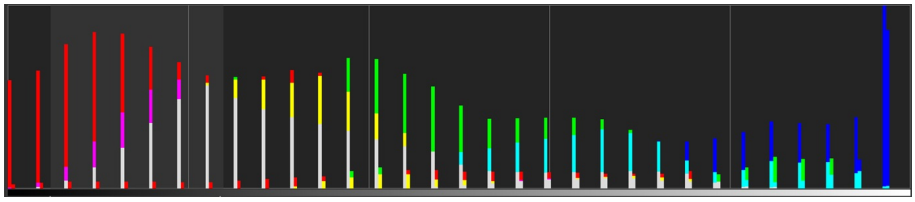
(a) Lago Limpia 4488x1777

(b) Lago con Payload

Figura 5.11: Comparación de Lago: Portadora vs. Payload [37]



(a) Histograma de Imagen Original



(b) Histograma de Imagen con Payload

Figura 5.12: Lago: Histograma Original vs. Payload

5.4.3. Análisis de Histogramas de Incrustación

El análisis de la distribución de los valores de los píxeles es una herramienta fundamental para detectar la presencia de información oculta mediante la técnica LSB. Las Figuras 5.2, 5.4, 5.6, 5.8, 5.10 y 5.12 muestran una comparación entre los histogramas de los canales de color originales y los obtenidos después de aplicar la esteganografía LSB[1]. Las variaciones observadas confirman la correcta incrustación del mensaje mediante los siguientes patrones característicos:

Patrones de redistribución identificados:

a) **Suavización espectral**: Los picos pronunciados en el histograma original se atenúan en el histograma modificado, reflejando la alteración controlada de los bits menos significativos.

b) **Redistribución entre valores adyacentes**: Se observa una transferencia de frecuencias entre valores de píxeles consecutivos (entre V y $V \pm 1$), generando un perfil característico en forma de "dientes de sierra", típico de la esteganografía LSB.

c) **Conservación de la estructura global**: A pesar de las alteraciones locales, la forma general del histograma se preserva, lo que garantiza la imperceptibilidad visual del proceso de ocultamiento.

Interpretación cuantitativa: La superposición y el entrecruzamiento de las curvas (canal original vs. canal con carga útil) permiten concluir que:

a) Las modificaciones se limitan al bit menos significativo, con cambios máximos de ± 1 en los valores de los píxeles.

b) La técnica incorpora la carga útil manteniendo la integridad estadística global de la imagen.

[1]Para la generación y comparación de todos los histogramas de esta sección se utilizó la herramienta de análisis integrada en el software GIMP (GNU Image Manipulation Program, versión 2.10).

c) La redistribución observada es consistente con el modelo teórico de esteganografía LSB, lo que valida la implementación correcta del método.

5.5. Relación Señal-Ruido Pico (PSNR)

La Relación Señal-Ruido Pico (PSNR, por sus siglas en inglés *Peak Signal-to-Noise Ratio*) es una métrica ampliamente utilizada para evaluar la calidad de reconstrucción de imágenes digitales, especialmente en aplicaciones de procesamiento digital de imágenes, compresión y esteganografía. Esta medida permite cuantificar el nivel de distorsión introducido en una imagen procesada en comparación con su imagen original.

El PSNR se expresa en decibelios (dB) y se calcula a partir del error cuadrático medio (MSE, *Mean Squared Error*), el cual mide la diferencia promedio al cuadrado entre los píxeles de la imagen original y la imagen modificada. Para dos imágenes monocromáticas I y K de tamaño $M \times N$, el MSE se define como:

$$MSE = \frac{1}{MN} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} [I(i,j) - K(i,j)]^2 \tag{5.1}$$

A partir del MSE, el PSNR se calcula mediante la siguiente expresión:

$$PSNR = 10 \cdot \log_{10} \left(\frac{MAX_I^2}{MSE} \right) \tag{5.2}$$

donde MAX_I representa el valor máximo posible de intensidad de un píxel en la imagen. En el caso de imágenes representadas con B bits por píxel, este valor se define como:

$$MAX_I = 2^B - 1 \tag{5.3}$$

Para imágenes en formato RGB, el valor del MSE se obtiene como el promedio aritmético de los errores cuadráticos medios calculados para cada uno de los canales de color (Rojo, Verde y Azul).

5.5.1. Resultados del PSNR

Con el objetivo de evaluar el impacto del *payload* incrustado en las imágenes, se calculó el PSNR entre cada imagen original y su correspondiente imagen con información oculta. Los resultados obtenidos se presentan en la Tabla 5.5.

Tabla 5.5: Valores de PSNR entre imágenes originales y con payload

No.	Imagen original	Imagen con payload	PSNR (dB)
1	Mickey.bmp	Mickey_con_Payload.bmp	35.91
2	Bosque.bmp	Bosque_con_Payload.bmp	35.85
3	Montaña.bmp	Montaña_con_Payload.bmp	36.16
4	Flor.bmp	Flor_con_Payload.bmp	36.47
5	Frutas.bmp	Frutas_con_Payload.bmp	36.14
6	Lago.bmp	Lago_con_Payload.bmp	35.78

Los valores de PSNR obtenidos se encuentran en un rango aproximado entre 35 y 36 dB, lo cual indica que la distorsión introducida por el proceso de inserción del payload es poco perceptible para el ojo humano. De acuerdo con la literatura, valores de PSNR superiores a 30 dB se consideran aceptables, mientras que valores por encima de 35 dB reflejan una alta calidad visual en la imagen resultante.

Estos resultados demuestran que el método propuesto preserva adecuadamente la calidad visual de las imágenes, validando su efectividad para la inserción de información sin degradación perceptible significativa.

5.6. Fase 4: Extracción, Descompresión y Reconstrucción

En esta sección vamos a reconstruir la imagen Payload, para ello se usan los script extraerlsb1.py, descompresion.c y niveles8.c, para extraer el payload incrustado y reconstruirlo.

5.6.1. Ejecución del programa `extraerlsb1.py`

En esta etapa se procede a la recuperación del archivo `.huff` previamente incrustado mediante el método LSB. Para ello, se ejecuta el script `extraerlsb1.py`, el cual analiza la imagen portadora y extrae los bits ocultos hasta reconstruir el archivo original. Una vez finalizado el proceso,se obtiene el fichero `.huff` resultante, que posteriormente será utilizado para verificar la integridad de la información y continuar con el proceso de descompresión.

```
Python extraerlsb1.py
 EXTRAYENDO ARCHIVO .huff
 Marcador de fin encontrado. Bits extraídos correctamente.
 <archivo.huff> extraído exitosamente y guardado como: <archivo_extraido.huf
```

Nº	Imagen Portadora	Archivo Extraído	Bytes Extraídos	Estado
1	Mickey	lenna.huff	468157	Éxito
2	Bosque	baboon.huff	409420	Éxito
3	Montaña	flower.huff	437818	Éxito
4	Flor	BoatsColor.huff	748941	Éxito
5	Frutas	pepper.huff	470216	Éxito
6	Lago	barbara.huff	742444	Éxito

Tabla 5.6: Resultados de extracción LSB del archivo .huff

Validación: Una vez extraído el archivo `.huff`, es necesario descomprimirlo para poder reconstruir posteriormente los niveles de gris de la imagen original. Este paso permite verificar la integridad de los datos extraídos y asegurar que el proceso de compresión y esteganografía se haya realizado de manera reversible y sin pérdida de información.

5.6.2. Ejecución del programa `descompresion.c`

En esta fase se procede a la descompresión del archivo obtenido en el paso anterior. Para ello, se ejecuta el programa `descompresion.c`, el cual aplica el algoritmo de decodificación correspondiente al formato `.huff`. Durante este proceso, el programa reconstruye el archivo original a partir de la secuencia de datos comprimidos, verificando la coherencia de la información y restaurando su estructura inicial. Como resultado, se obtiene el archivo final completamente descomprimido y listo para su análisis o utilización posterior.

```
gcc descompresion.c
./a <archivo_extraido.huff> <archivo_extraido.bmp>
```

Nº	Archivo .huff	Bytes Comprimidos	Bytes Descomprimidos	Validación
1	lenna.huff	468157	786432	Correcto
2	baboon.huff	409420	720000	Correcto
3	flower.huff	437818	737280	Correcto
4	BoatsColor.huff	748941	1359936	Correcto
5	pepper.huff	470216	786432	Correcto
6	barbara.huff	742444	1244160	Correcto

Tabla 5.7: Resultados de descompresión del archivo Huffman

Validación: La descompresión del archivo `.huff` resultó en un número de bytes idéntico al de la imagen reducida original, lo que confirma que el *payload* extraído no sufrió ninguna corrupción durante los procesos de compresión e incrustación.

5.6.3. Ejecución del programa `niveles8.c`

El programa `niveles8.c` se emplea para restaurar la profundidad de color de la imagen que había sido previamente truncada. Su función principal consiste en expandir el rango de valores de cada componente de color, llevándolos nuevamente a un formato de 8 bits por canal. De esta manera, se reconstruye la imagen con su fidelidad visual original, asegurando que los niveles de intensidad de cada componente (rojo, verde y azul) se correspondan con los valores completos de 8 bits, lo que permite una correcta visualización y análisis posterior.

```
 gcc niveles8.c
./a <imagen_entrada> <archivo_reconstruido> <archivo_verificacion>
```

Nº	Imagen Descomprimida	Imagen Reconstruida	Estado
1	lenna_extraida.bmp	lenna_reconstruida.bmp	Éxito
2	baboon_extraido.bmp	baboon_reconstruido.bmp	Éxito
3	flower_extraido.bmp	flower_reconstruida.bmp	Éxito
4	BoatsColor_extraido.bmp	BoatsColor_reconstruido.bmp	Éxito
5	pepper_extraido.bmp	pepper_reconstruido.bmp	Éxito
6	barbara_extraida.bmp	barbara_reconstruida.bmp	Éxito

Tabla 5.8: Resultados de la reconstrucción de niveles

Validación: La reconstrucción de la profundidad de color fue verificada comparando cada imagen reconstruida con su correspondiente imagen descomprimida original. Como se observa en la Tabla 5.8, todas las imágenes reconstruidas muestran un estado de *Éxito*, confirmando que los niveles de intensidad de los componentes RGB fueron restaurados correctamente y que la fidelidad visual de las imágenes se mantiene intacta.Esta verificación garantiza la efectividad del programa `niveles8.c` en la recuperación completa de los valores de 8 bits por componente.

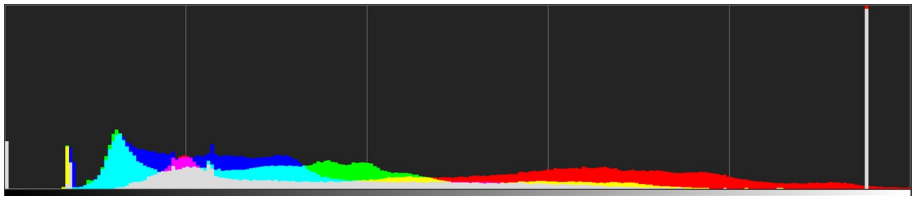
5.6.4. Comparación Visual de Reconstrucción

A continuación se presentan dos imágenes extraídas durante el proceso, acompañadas de los datos obtenidos en cada caso. Esta comparación visual permite evaluar la fidelidad de la reconstrucción y determinar el grado en que la información oculta fue recuperada de manera correcta.

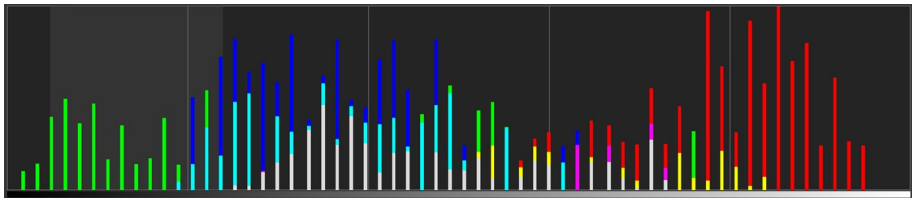
(a) Lenna Imagen Original

(b) Lenna Imagen reconstruida

Figura 5.13: Lenna Original vs. Lenna Reconstruida [38]



(a) Histograma de Imagen Original



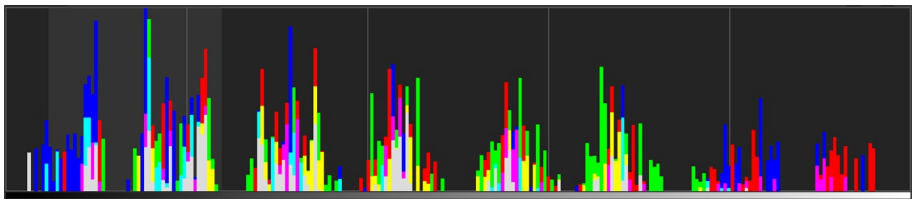
(b) Histograma de Imagen Reconstruida

Figura 5.14: Lenna Histograma de reconstrucción

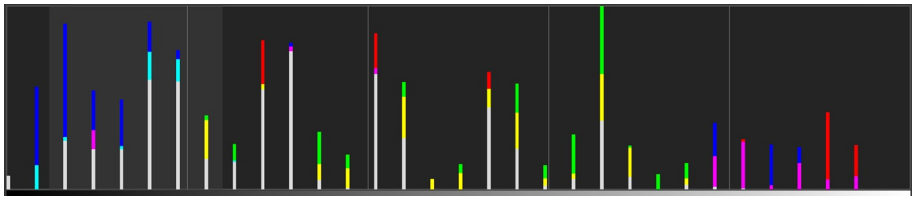
(a) Baboon Imagen Original

(b) Baboon Imagen reconstruida

Figura 5.15: Baboon Original vs. Baboon Reconstruida [39]



(a) Histograma de Imagen Original



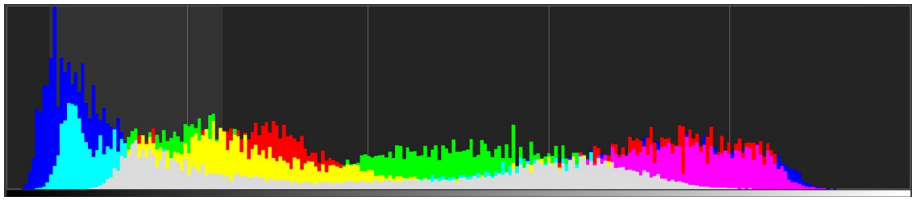
(b) Histograma de Imagen Reconstruida

Figura 5.16: Baboon Histograma de reconstrucción

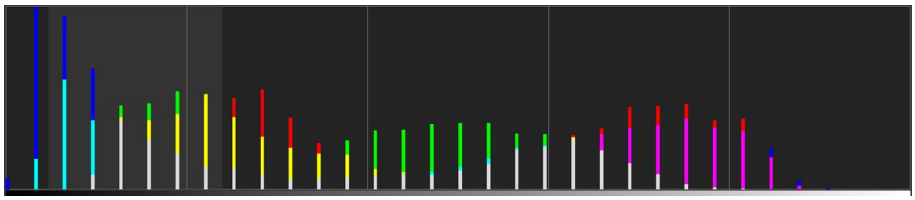
(a) Flower Imagen Original

(b) Flower Imagen reconstruida

Figura 5.17: Flower Original vs. Flower Reconstruida [40]



(a) Histograma de Imagen Original



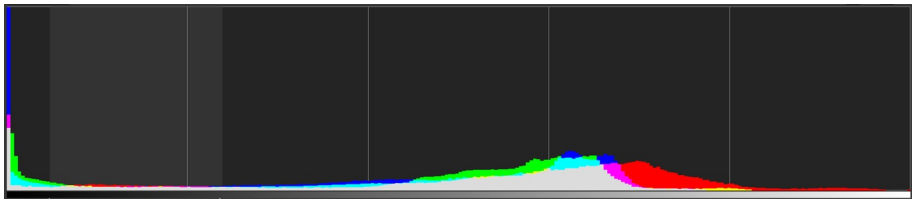
(b) Histograma de Imagen Reconstruida

Figura 5.18: Flower Histograma de reconstrucción

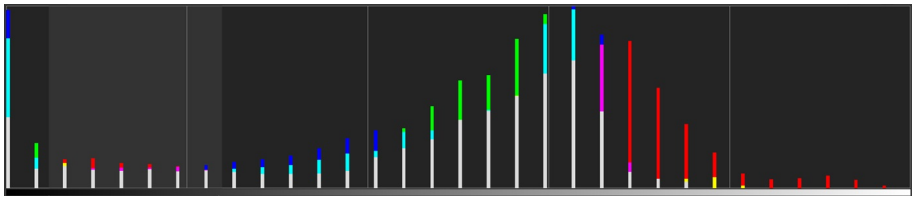
(a) BoatsColor Imagen Original

(b) BoatsColor Imagen reconstruida

Figura 5.19: BoatsColor Original vs. BoatsColor Reconstruida [41]



(a) Histograma de Imagen Original



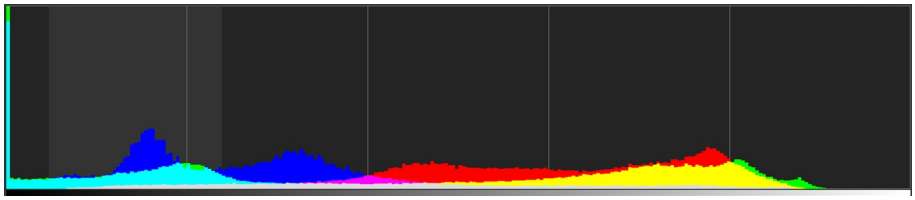
(b) Histograma de Imagen Reconstruida

Figura 5.20: BoatsColor Histograma de reconstrucción

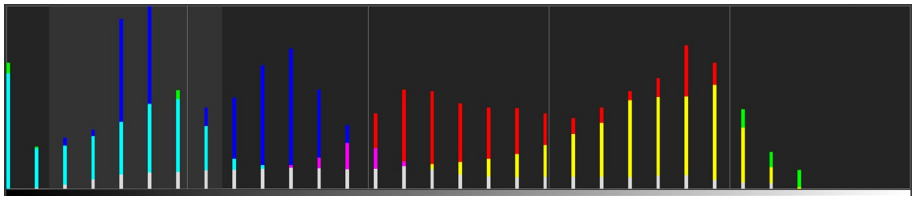
(a) Pepper Imagen Original

(b) Pepper Imagen reconstruida

Figura 5.21: Pepper Original vs. Pepper Reconstruida [42]



(a) Histograma de Imagen Original



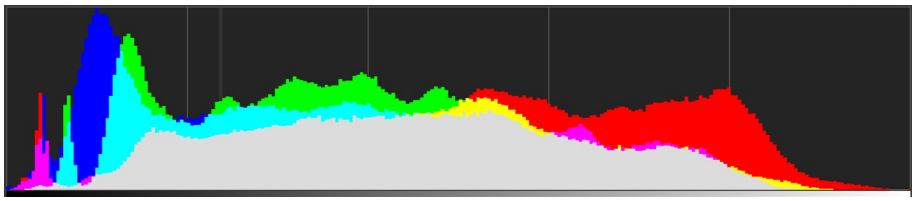
(b) Histograma de Imagen Reconstruida

Figura 5.22: Pepper Histograma de reconstrucción

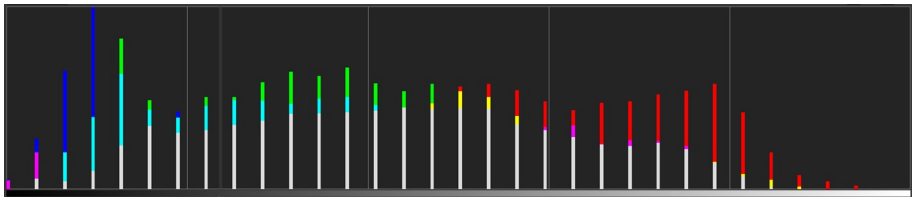
(a) Barbara Imagen Original

(b) Barbara Imagen reconstruida

Figura 5.23: Barbara Original vs. Barbara Reconstruida [43]



(a) Histograma de Imagen Original



(b) Histograma de Imagen Reconstruida

Figura 5.24: Barbara Histograma de reconstrucción

5.6.5. Análisis de Histogramas de Reconstrucción

El análisis comparativo de histogramas es esencial para evaluar la fidelidad en la reconstrucción de imágenes mediante técnicas esteganográficas LSB. Las Figuras 5.14, 5.16, 5.18, 5.20, 5.22 y 5.24 presentan una comparación directa entre los histogramas de las imágenes originales y sus respectivas reconstrucciones[2].

Evaluación de la fidelidad en la reconstrucción:

a) **Coincidencia espectral**: Se observa una correspondencia notable entre las distribuciones de frecuencia de las imágenes originales y las reconstruidas, lo que evidencia la efectividad del proceso de recuperación.

b) **Preservación de la estructura global**: Los patrones característicos de cada imagen—picos, valles y distribuciones particulares—se mantienen, demostrando la integridad de la información recuperada.

c) **Minimización de artefactos**: La ausencia de discontinuidades abruptas o patrones anómalos en los histogramas reconstruidos confirma la calidad del proceso de extracción y descompresión.

Interpretación de los resultados:

a) La superposición casi exacta de las curvas histogramáticas indica que el proceso de reconstrucción recupera fielmente la distribución original de los valores de píxel.

b) Las ligeras variaciones detectadas son consistentes con el margen esperado para técnicas de compresión sin pérdida combinadas con métodos esteganográficos LSB.

c) La correspondencia entre los histogramas valida la reversibilidad completa del proceso, desde la incrustación hasta la recuperación final de la imagen original.

[2]Los histogramas comparativos se obtuvieron utilizando la misma herramienta de análisis del software GIMP, asegurando consistencia metodológica con la sección anterior.

Este análisis cuantitativo, complementado con la evaluación visual directa, confirma que el método implementado logra un equilibrio adecuado entre **capacidad de ocultamiento** y **calidad de reconstrucción**, garantizando la recuperación íntegra de la información embebida sin comprometer la fidelidad de la imagen original.

5.7. Difusión de Resultados

Como parte de la validación y difusión académica del trabajo desarrollado, los resultados obtenidos fueron presentados en un evento de carácter internacional, en modalidad de póster.

5.7.1. Participación en Congreso

El trabajo titulado *"Hiding Information in RGB Images with LSB"* fue presentado como póster en la **Twelfth International Conference on Mathematics and its Applications (12CIMA)**, organizada por la **Facultad de Ciencias Físico Matemáticas (FCFM)** de la **Benemérita Universidad Autónoma de Puebla (BUAP)**.

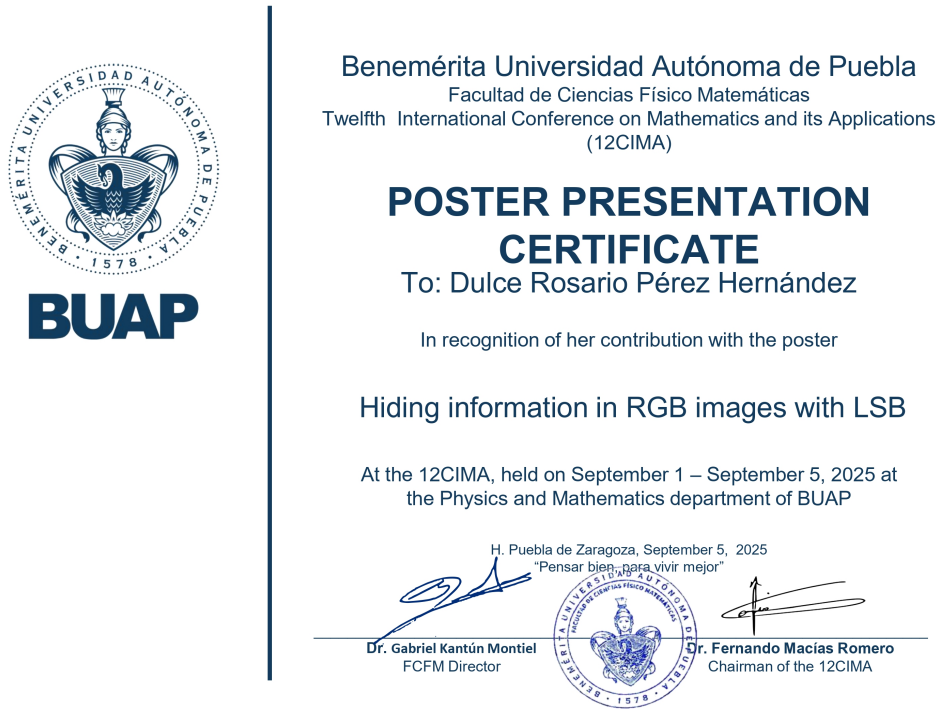
a) **Nombre del congreso:** Twelfth International Conference on Mathematics and its Applications (12CIMA)

b) **Tipo de participación:** Presentación de póster

c) **Fechas:** Del 1 al 5 de septiembre de 2025

d) **Lugar:** Departamento de Física y Matemáticas de la BUAP, H. Puebla de Zaragoza

e) **Autora:** Dulce Rosario Pérez Hernández



BUAP

Benemérita Universidad Autónoma de Puebla
Facultad de Ciencias Físico Matemáticas
Twelfth International Conference on Mathematics and its Applications
(12CIMA)

POSTER PRESENTATION CERTIFICATE

To: Dulce Rosario Pérez Hernández

In recognition of her contribution with the poster

Hiding information in RGB images with LSB

At the 12CIMA, held on September 1 – September 5, 2025 at
the Physics and Mathematics department of BUAP

H. Puebla de Zaragoza, September 5, 2025
"Pensar bien, para vivir mejor"

Dr. Gabriel Kantún Montiel
FCFM Director

Dr. Fernando Macías Romero
Chairman of the 12CIMA

Figura 5.25: Constancia de participación en el 12CIMA, BUAP 2025.

La aceptación y presentación del trabajo en un congreso internacional avala la relevancia académica y la pertinencia del estudio en el ámbito de la esteganografía digital y la compresión de imágenes, contribuyendo a su visibilidad científica y a la validación por parte de la comunidad especializada.

CAPÍTULO 6

CONCLUSIONES Y TRABAJO FUTURO

6.1. Conclusiones

El objetivo central de esta investigación fue desarrollar un sistema de ocultamiento de información en imágenes digitales que optimizara la capacidad de almacenamiento y la discreción (imperceptibilidad y robustez) del mensaje secreto dentro de la técnica de Esteganografía LSB [44, 45]. Esto se logró mediante el preprocesamiento del mensaje a ocultar (la Imagen Payload P) y una inserción estratégica en (la Imagen Portadora G). Los resultados obtenidos validan la hipótesis de trabajo y permiten establecer la conclusión general del proyecto.

6.2. Análisis de Desempeño

Indicadores de Desempeño, los principales indicadores medidos fueron:

a) **Eficiencia de compresión:** Huffman redujo la imagen oculta en **30 % - 40 %**.

b) **Capacidad de incrustación:** LSB utilizó **3 bits por componente RGB** sin pérdida visual apreciable.

c) **Reversibilidad:** Proceso completo (incrustación, extracción, reconstrucción) **sin pérdida de datos**.

d) **Tiempo de ejecución:** Todas las operaciones en **menos de 3 segundos** en equipo estándar.

6.2.1. Cumplimiento de Objetivos

Cumplimiento: Se combinaron exitosamente las tres técnicas:

Reducción de niveles de color.

Compresión Huffman sin pérdida.

Localización del área óptima para incrustación.

Objetivos Específicos

Objetivo Específico	**Cumplimiento**	**Evidencia Experimental**
Analizar e implementar la reducción de profundidad de color como paso previo a la compresión.	Cumplido	Programa `niveles.c`; `<archivo_reducido.bmp>` con 3 bits menos significativos en cero.
Diseñar e implementar Huffman para la compresión sin pérdida de la imagen payload reducida.	Cumplido	Archivo `archivo.huff` con ratio de compresión 40.00 %.
Desarrollar un algoritmo de localización del área rectangular máxima disponible en la imagen portadora.	Cumplido	Programa `rectangulo.c`; área determinada a × b (n píxeles).
Integrar datos comprimidos en LSB del área óptima de la portadora.	Cumplido	Script `lsbimagen.py`; archivo oculto en `imagen_con_payload.bmp`.
Evaluar desempeño con métricas objetivas (calidad visual, tasa de compresión y precisión de extracción).	Cumplido	Comparación visual sin distorsión; histograma modificado; extracción y reconstrucción exitosas.

Tabla 6.1: Cumplimiento de objetivos específicos del sistema.

Nota: El sistema demostró **alta capacidad de ocultamiento**, **eficiencia de compresión sin pérdida** y **precisión en la extracción**, confirmando la efectividad integral del modelo LSB combinado con compresión Huffman.

6.3. Respuesta a la Pregunta de Investigación

La pregunta que guio la presente investigación fue:

> *¿De qué manera la combinación de la reducción de profundidad de color y la codificación de Huffman, junto con la localización de un área rectangular máxima, permite desarrollar un sistema de esteganografía LSB más robusto y con mayor capacidad de ocultamiento que las implementaciones tradicionales sin preprocesamiento de datos?*

Respuesta:
Se concluye que la combinación de las técnicas propuestas sí permite desarrollar un sistema de esteganografía LSB más robusto, discreto y con una capacidad de ocultamiento superior [46, 47]. Este logro se fundamenta en los siguientes puntos:

a) **Optimización del Payload y Capacidad:** La doble fase de compresión (reducción de 28 a 25 niveles de color, seguida de la codificación de Huffman [48]) minimizó drásticamente el tamaño del mensaje a incrustar. Esta reducción optimizó el uso del espacio disponible en la Imagen Portadora G, demostrando un aumento en la capacidad efectiva de ocultamiento.

b) **Aumento de la Robustez e Imperceptibilidad:** Al reducirse la cantidad de datos a incrustar, el sistema minimiza la cantidad de bits LSB modificados [45]. Una menor modificación se traduce directamente en una mayor robustez y menor vulnerabilidad a ataques de estegoanálisis.

c) **Inserción Estratégica:** El algoritmo de localización del área rectangular de mayor tamaño permitió asegurar que la inserción se realizara en la región espacial más óptima [46].

d) **Integridad Garantizada:** Se validó que el sistema permite la recuperación exacta de la Imagen Secreta P original mediante la descompresión y reconstrucción de los niveles de color completos.

Se comprueba que el método de ocultamiento desarrollado, al integrar la optimización del payload mediante la doble fase de compresión y la localización estratégica del área de incrustación, es más eficaz y robusto para la esteganografía LSB en imágenes de mapas de bits, superando las implementaciones tradicionales en eficiencia espacial y discreción.

6.4. Contribuciones

El presente trabajo aporta diversos avances al campo de la esteganografía digital, particularmente en técnicas basadas en el dominio espacial. Las principales contribuciones pueden resumirse de la siguiente manera:

a) Se demostró la eficacia de un modelo de compresión híbrido que integra la reducción de redundancia psicovisual con la codificación sin pérdida mediante el algoritmo de Huffman [48]. Esta combinación permitió disminuir el volumen de información a insertar sin comprometer la integridad del mensaje, optimizando así la capacidad de ocultamiento y manteniendo la calidad perceptual de la imagen portadora.

b) Se implementó una técnica de inserción espacialmente inteligente que articula criterios esteganográficos y geométricos [47]. La estrategia propuesta aprovecha características estructurales de la imagen para ubicar regiones óptimas de inserción, reduciendo la probabilidad de detección y mejorando la estabilidad del mensaje oculto frente a variaciones menores en la imagen portadora.

c) El sistema desarrollado constituye un marco modular y extensible que puede servir como base para investigaciones posteriores orientadas a incrementar tanto la seguridad como la capacidad de los métodos de ocultamiento basados en *Least Significant Bit* (LSB). Su diseño flexible permite la sustitución o mejora de componentes individuales sin alterar la arquitectura general del método.

6.5. Limitaciones encontradas

A lo largo del desarrollo del proyecto se identificaron diversas limitaciones metodológicas y experimentales que deben considerarse al interpretar los resultados obtenidos:

a) El método fue validado exclusivamente sobre imágenes en formato BMP de 24 bits. Esta restricción limita la generalización de los resultados a otros formatos más utilizados en entornos reales, como PNG o JPEG, que incorporan distintos mecanismos de compresión.

b) La técnica depende de manera significativa de la correcta identificación del área rectangular máxima destinada a la inserción. Errores en esta etapa podrían afectar tanto la capacidad de ocultamiento como la imperceptibilidad del mensaje.

c) No se incorporaron técnicas adicionales de cifrado sobre el mensaje antes de su inserción. En consecuencia, aunque el sistema ofrece un nivel de ocultamiento adecuado, la información permanece vulnerable en escenarios donde el atacante logre extraer los bits incrustados.

6.6. Recomendaciones

Con base en las conclusiones derivadas del estudio, se sugieren las siguientes recomendaciones para fortalecer la metodología y ampliar sus posibles aplicaciones:

a) Extender la evaluación experimental a una mayor variedad de formatos de imagen y resoluciones, con el fin de determinar la robustez y adaptabilidad del sistema en escenarios más cercanos a aplicaciones reales.

b) Incorporar mecanismos criptográficos previos al proceso de inserción [49]. La adición de una capa de cifrado incrementaría de manera significativa la seguridad, incluso en el caso de una eventual extracción del mensaje.

c) Realizar pruebas sistemáticas utilizando distintos tipos de ataques de estegoanálisis, tanto estadísticos como basados en aprendizaje automático, para caracterizar cuantitativamente la robustez del sistema frente a intentos de detección o extracción.

6.7. Líneas de investigación futuras

A partir de las posibilidades identificadas durante la investigación, se plantean diversas líneas de trabajo futuro orientadas a la ampliación y fortalecimiento del enfoque propuesto. Estas líneas permiten explorar mejoras tanto en la robustez del método como en su aplicabilidad práctica.

a) **Migración hacia dominios transformados (DCT y DWT)** Una posible dirección de avance consiste en extender el modelo desde el dominio espacial hacia técnicas de esteganografía en el dominio transformado, tales como DCT o DWT [46]. Actualmente, la técnica trabaja modificando directamente los bits de los píxeles (dominio espacial), lo cual la hace vulnerable a compresiones o filtros. En cambio:

- **DCT** (Discrete Cosine Transform), utilizada en JPEG, permite incrustar información en coeficientes frecuenciales menos sensibles a la compresión.
- **DWT** (Discrete Wavelet Transform) posibilita distribuir la información entre subbandas de frecuencia, otorgando mayor resiliencia a modificaciones.

El uso de estas transformaciones podría ofrecer una mayor resistencia ante pérdidas, ataques o degradación, aumentando notablemente la robustez del método frente a manipulaciones comunes en entornos multimedia.

b) **Integración de algoritmos de Inteligencia Artificial para la selección automática de áreas de inserción** Una mejora significativa consiste en optimizar dinámicamente la región donde se insertará la información mediante técnicas de aprendizaje automático. En la propuesta actual, la selección depende de un algoritmo geométrico que identifica el rectángulo máximo disponible; sin embargo, emplear modelos de IA permitiría:

- detectar zonas con alto nivel de detalle donde los cambios son menos perceptibles,
- aprender patrones esteganográficos óptimos mediante redes neuronales,
- predecir regiones estadísticamente más seguras para la inserción.

Esto daría lugar a un proceso más inteligente, adaptable y resistente a métodos de detección esteganográfica.

c) **Desarrollo de una interfaz gráfica multiplataforma** El sistema actual funciona mediante ejecución por consola, lo cual limita su uso a usuarios con conocimientos técnicos. Como trabajo futuro se propone desarrollar una interfaz gráfica que permita:

- cargar imágenes mediante arrastrar y soltar,
- visualizar la región donde se incrustará el mensaje,
- ejecutar procesos de ocultamiento, extracción y compresión con un clic.

Una herramienta de este tipo facilitaría la adopción del método en contextos educativos, investigativos o profesionales.

d) **Compatibilidad con formatos de imagen más comunes (PNG y JPEG)** El proyecto actual opera únicamente sobre imágenes BMP de 24 bits debido a su ausencia de compresión, lo que facilita la manipulación directa del píxel. Como evolución natural, se propone extender la compatibilidad del sistema hacia formatos ampliamente utilizados:

- **PNG**: emplea compresión sin pérdidas, por lo que aún es posible utilizar técnicas basadas en LSB con ajustes adecuados.
- **JPEG**: utiliza compresión con pérdidas, lo que destruye información LSB; por ello su incorporación requeriría migrar a técnicas de dominio transformado, como DCT.

Esta ampliación implicaría rediseñar parte del sistema, pero permitiría trabajar con imágenes más populares y reducir el tamaño de los archivos utilizados.

Bibliografía

[1] Eric Cole. *Hiding in Plain Sight: Steganography and the Art of Covert Communication*. Somerset, New Jersey: John Wiley & Sons, Inc., 2003.

[2] Kefa Rabah. «Steganography—The Art of Hiding Data». En: *Information Technology Journal* 3.3 (2004), págs. 245-269.

[3] Ewa Zielinska, Wojciech Mazurczyk y Krzysztof Szczypiorski. «Trends in steganography». En: *Communications of the ACM* 57.3 (2014), págs. 86-95.

[4] G. C. Kessler. *An Overview of Steganography for the Computer Forensics Examiner*. Inf. téc. Updated Version, February 2015. Embry-Riddle Aeronautical University, 2015.

[5] G. Devi, R. Miglani y A. Dahiya. «Steganography and Its Role in Data Security: Techniques and Applications». En: *International Journal of Science, Engineering and Technology* 12.5 (2024), págs. 45-58.

[6] B. T. Ahmed. «A Systematic Overview of Secure Image Steganography». En: *International Journal of Advances in Applied Sciences (IJAAS)* 10 (2021), págs. 178-187.

[7] Z. Wang et al. *Data Hiding with Deep Learning: A Survey Unifying Digital Watermarking and Steganography*. arXiv preprint arXiv:2103.00001. 2021.

[8] National Instruments. *Definition of a Digital Image*. NI Vision Concepts Help. "An image is a 2D array of values representing light intensity ... each pixel a numeric location and a gray level or color value that specifies the brightness or color of the pixel." n.d. URL: https://www.

ni.com/docs/en-US/bundle/ni-vision-concepts-help/page/definition_digital_image.html.

[9] Wikipedia. *Imagen de mapa de bits*. Accedido el 25 de octubre de 2025. 2023. URL: https://es.wikipedia.org/wiki/Imagen_de_mapa_de_bits.

[10] Miguel Palomar de Diego. *Que es una imagen vectorial y que es una imagen de mapa de bits*. Accedido: 2025-10-20. 17 de jul. de 2019. URL: https://miguelpalomar.com/que-es-una-imagen-vectorial-y-que-es-una-imagen-de-mapa-de-bits/.

[11] IONOS Mexico. *Imagen vectorial o mapa de bits: pros y contra*. Accedido el 25 de octubre de 2025. 2023. URL: https://www.ionos.mx/digitalguide/paginas-web/diseno-web/imagen-vectorial-o-mapa-de-bits-pros-y-contra/.

[12] Impresion Total. *Diferencias entre imagen de mapa de bits e imagen vectorial*. Accedido el 25 de octubre de 2025. 2022. URL: https://impresiontotal.com/diferencia-entre-imagen-de-mapa-de-bits-e-imagen-vectorial/.

[13] The Color Blog. *Diferencia entre una imagen vectorial y una imagen de mapa de bits*. Accedido el 25 de octubre de 2025. 2024. URL: https://thecolor.blog/es/diferencia-una-imagen-vectorial-y-mapa-de-bits/.

[14] CCM. *Imagenes vectoriales y de mapa de bits: en que se diferencian*. Accedido el 25 de octubre de 2025. 2023. URL: https://es.ccm.net/aplicaciones-e-internet/museo-de-internet/enciclopedia/11368-imagenes-vectoriales-y-de-mapa-de-bits-en-que-se-diferencian/.

[15] Ana M. Gonzalez. «Analisis de los formatos de imagen digital y su impacto en aplicaciones web». En: *Revista de Ingenieria de Sistemas y Tecnologia* 4.1 (2016), págs. 15-28.

[16] Wikipedia. *Pixel*. Jun. de 2025.

[17] Wikipedia contributors. *"Composicion alfa"*. Last revised 24 January 2024. 2024.

[18] El Codigo ASCII. *El codigo ASCII completo: tabla con los codigos ASCII completos, caracteres, simbolos, letras, ASCII codigo*. https://elcodigoascii.com.ar/. Consultado el 20 de octubre de 2025. 2025.

[19] CIDECA. *Procesos hacia adelante: Desplazamientos*. Consultado el 26 de octubre de 2025. 2023. URL: http://cidecame.uaeh.edu.mx/lcc/mapa/PROYECTO/libro20/46_procesos_hacia_adelante.html.

[20] Daniel Lerch. *Esteganografia LSB en imagenes y audio*. Accedido el 25 de octubre de 2025. 2025. URL: https://daniellerch.me/stego/intro/lsb-es/.

[21] Mikel Garcia Larragan. *Criptografia (LIX): Esteganografia LSB (I)*. Accedido el 25 de octubre de 2025. 2017. URL: https://mikelgarcialarrPixelWikipedia blogspot.com/2017/10/criptografia-lix-esteganografia-lsb-i.html.

[22] Sergio Alexis Ramirez Valle. «Estudio de algoritmos de IA aplicables al estegoanalisis de imagenes digitales». Tesis en el area de seguridad informatica y esteganografia. Tesis de pregrado. Nombre de la universidad (si la sabes), 2021.

[23] *Estudio y analisis de los procesos de la esteganografia en Colombia*. Investigacion en curso sobre esteganografia en Colombia. 2022.

[24] Joab Kose, Oscar Bautista Chia y Vashish Baboolal. «Review and Test of Steganography Techniques». En: *arXiv preprint* (2020). arXiv: 2012.08460 [cs.CR]. URL: https://arxiv.org/abs/2012.08460.

[25] «Esteganografia: el arte de ocultar informacion en medios digitales». En: (2024). Articulo academico de revision sobre tecnicas esteganograficas.

[26] M. A. Idakwo et al. «An Extensive Survey of Digital Image Steganography: State of the Art». En: *arXiv preprint* (2024). arXiv: 2404.19548 [cs.ET]. URL: https://arxiv.org/abs/2404.19548.

[27] Python Software Foundation. *Python Programming Language*. https://www.python.org/. Accedido: 04 Noviembre 2025. 2020.

[28] Brian W. Kernighan y Dennis M. Ritchie. *The C Programming Language*. 2nd. Prentice Hall, 1988.

[29] Alex Clark y Contributors. *Pillow (Python Imaging Library Fork)*. https://python-pillow.org/. Accedido: 04 Noviembre 2025. 2025.

[30] David A. Huffman. «A Method for the Construction of Minimum-Redundancy Codes». En: *Proceedings of the IRE* 40.9 (1952), págs. 1098-1101.

[31] Wikipedia Contributors. *Bitwise Operations*. https://en.wikipedia.org/wiki/Bitwise_operations. Accedido: 04 Noviembre 2025. 2025.

[32] Usuario Anonimo. *Mickey Mouse*. Accedido: 2025-10-26. 2023. URL: https://galeri.uludagsozluk.com/r/mickey-mouse-723749/.

[33] Xtrafondos. *Montaña en bosque*. Accedido el 11 de noviembre de 2025. s.f. URL: https://www.xtrafondos.com/wallpaper/3840x2400/5769-montana-en-bosque.html.

[34] Xtrafondos. *Montañas en bosque durante el otoño*. Accedido el 11 de noviembre de 2025. s.f. URL: https://www.xtrafondos.com/wallpaper/3840x2400/6501-montanas-en-bosque-durante-el-otono.html.

[35] Wallpapers.com. *Imágenes de Alta Resolución*. Accedido el 11 de noviembre de 2025. s.f. URL: https://es.wallpapers.com/imagenes-de-alta-resolucion#google_vignette.

[36] *Frutas — imagen decorativa*. https://laboutiquetrattoriaviajera.es/wp-content/uploads/2019/07/frutta.jpg. Accedido: 2025-11-28. 2019.

[37] Pinterest user. *Pin 348606827386870705*. Accedido el 11 de noviembre de 2025. s.f. URL: https://mx.pinterest.com/pin/348606827386870705/.

[38] Dwight Hooker y Alexander Sawchuk et al. *Lenna (512x512) - Standard Test Image*. https://en.wikipedia.org/wiki/File:Lenna_(test_image).png. Original from Playboy Magazine, November 1972. Scanned and cropped by Alexander Sawchuk and colleagues at USC-SIPI for image compression research. Use of this scan is generally considered fair use or overlooked for research purposes. 1972.

[39] Vladimir Hlevkin. *Test Image: Baboon*. Accedido el 11 de noviembre de 2025. s.f. URL: https://www.hlevkin.com/hlevkin/TestImages/baboon.bmp.

[40] Vladimir Hlevkin. *Test Image: Flower*. Accedido el 11 de noviembre de 2025. s.f. URL: https://www.hlevkin.com/hlevkin/TestImages/flower.bmp.

[41] Hlevkin TestImages. *BoatsColor.bmp — imagen de prueba en color*. https://www.hlevkin.com/hlevkin/TestImages/BoatsColor.bmp. Accedido: 28 noviembre 2025. 2025.

[42] Vladimir Hlevkin. *Test Image: Pepper*. Accedido el 11 de noviembre de 2025. s.f. URL: https://www.hlevkin.com/hlevkin/TestImages/pepper.bmp.

[43] *barbara.bmp — imagen de prueba en color*. https://www.hlevkin.com/hlevkin/TestImages/barbara.bmp. Accedido: 2025-11-28.

[44] Neil F Johnson y Sushil Jajodia. «Exploring steganography: Seeing the unseen». En: *Computer* 31.2 (1998), págs. 26-34.

[45] Niels Provos y Peter Honeyman. «Hide and seek: An introduction to steganography». En: *IEEE Security & Privacy* 1.3 (2003), págs. 32-44.

[46] Jessica Fridrich. «Steganography in digital media: Principles, algorithms, and applications». En: (2009).

[47] Chi-Kwong Chan y Lee-Ming Cheng. «Hiding data in images by simple LSB substitution». En: *Pattern Recognition* 37.3 (2004), págs. 469-474.

[48] Khalid Sayood. *Introduction to Data Compression*. 5th. Burlington, MA: Morgan Kaufmann, 2017.

[49] Gary C Kessler. «An overview of steganography for the computer forensics examiner». En: *Forensic Science Communications* 6.3 (2014).

Apéndice A: Códigos Fuente Utilizados

A continuación se presentan los archivos fuente implementados en lenguaje C y Python para el desarrollo del sistema de compresión, detección de regiones óptimas e inserción esteganográfica mediante técnica LSB.

Archivos C

niveles.c

Listing 1: Archivo `niveles.c` - Reducción de profundidad de color

```c
#include "stdio.h"
#include "stdlib.h"
#include "time.h"
int main(int argc, char **argv)
{
 char *fileIN, *fileOUT, *fileVerif, *head;
 FILE *imgIN, *imgOUT, *imgVerif;
 srand(time(NULL));

 fileIN = argv[1];
 fileOUT = argv[2];
 fileVerif = argv[3];
 printf("%s %s", fileIN, fileOUT, fileVerif);

 imgIN = fopen(fileIN,"rb");
 imgOUT = fopen(fileOUT,"wb");
```

```
17  imgVerif = fopen(fileVerif,"wb");
18
19  head = (char *)malloc(sizeof(char)*54);
20
21  fread(head, 1, 54, imgIN);
22  fwrite(head,1, 54, imgOUT);
23  fwrite(head,1, 54, imgVerif);
24
25  int c, max=0, random;
26
27  while((c=fgetc(imgIN))!=EOF)
28  {
29   c = c>>3;
30   c = c<<3;
31   if (c > max)
32     max = c;
33   fputc(c, imgOUT);
34   random = rand()%8;
35
36   fputc(c + random , imgVerif);
37  }
38
39  printf("\nEl valor maximo es %d\n", max);
40  printf("Ahora los 3 bits menos significativos tienen valor 0,
       ahora se pueden almacenar 9 bits por pixel, es decir, 1
       byte mas 1/8");
41  printf("Se ha escrito una comprobacion de color en la imagen
       de verificacion %s",fileVerif);
42  fclose(imgIN);
43  fclose(imgOUT);
44  fclose(imgVerif);
45 }
```

compresion.c

Listing 2: Archivo `compresion.c` - Algoritmo de compresión de imagen usando Huffman

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdint.h>

#define MAX_TREE_NODES 512
#define MAX_CODE_LEN 256

typedef struct Node {
    uint8_t value;
    int freq;
    struct Node *left, *right;
} Node;

typedef struct {
    Node **nodes;
    int size;
    int capacity;
} MinHeap;

typedef struct {
    uint8_t *bits;
    int length;
} HuffCode;

MinHeap* createMinHeap(int capacity);
void insertMinHeap(MinHeap *heap, Node *node);
Node* extractMin(MinHeap *heap);
Node* createNode(uint8_t value, int freq);
Node* buildHuffmanTree(int *freq);
void generateCodes(Node *root, uint8_t *code, int depth,
    HuffCode *codes);
```

```
32 void writeCompressedFile(const char *filename, uint8_t *data,
       int size,
33                          HuffCode *codes, Node *tree, int width
                                , int height);
34 uint8_t* readPPM(const char *filename, int *width, int *height)
       ;
35 uint8_t* readBMP(const char *filename, int *width, int *height)
       ;
36 void freeCodes(HuffCode *codes);
37 void freeTree(Node *root);
38
39 MinHeap* createMinHeap(int capacity) {
40     MinHeap *heap = (MinHeap*)malloc(sizeof(MinHeap));
41     heap->size = 0;
42     heap->capacity = capacity;
43     heap->nodes = (Node**)malloc(capacity * sizeof(Node*));
44     return heap;
45 }
46
47 void insertMinHeap(MinHeap *heap, Node *node) {
48     heap->nodes[heap->size] = node;
49     int i = heap->size++;
50     while (i > 0 && heap->nodes[(i-1)/2]->freq > heap->nodes[i
           ]->freq) {
51         Node *temp = heap->nodes[i];
52         heap->nodes[i] = heap->nodes[(i-1)/2];
53         heap->nodes[(i-1)/2] = temp;
54         i = (i-1)/2;
55     }
56 }
57
58 Node* extractMin(MinHeap *heap) {
59     if (heap->size == 0) return NULL;
60     Node *min = heap->nodes[0];
61     heap->nodes[0] = heap->nodes[--heap->size];
62     int i = 0;
63     while (1) {
```

```
        int smallest = i;
        int left = 2*i + 1;
        int right = 2*i + 2;
        if (left < heap->size && heap->nodes[left]->freq < heap
            ->nodes[smallest]->freq)
            smallest = left;
        if (right < heap->size && heap->nodes[right]->freq <
            heap->nodes[smallest]->freq)
            smallest = right;
        if (smallest == i) break;
        Node *temp = heap->nodes[i];
        heap->nodes[i] = heap->nodes[smallest];
        heap->nodes[smallest] = temp;
        i = smallest;
    }
    return min;
}

Node* createNode(uint8_t value, int freq) {
    Node *node = (Node*)malloc(sizeof(Node));
    node->value = value;
    node->freq = freq;
    node->left = node->right = NULL;
    return node;
}

Node* buildHuffmanTree(int *freq) {
    MinHeap *heap = createMinHeap(256);
    for (int i = 0; i < 256; i++) {
        if (freq[i] > 0) {
            insertMinHeap(heap, createNode(i, freq[i]));
        }
    }
    while (heap->size > 1) {
        Node *left = extractMin(heap);
        Node *right = extractMin(heap);
        Node *parent = createNode(0, left->freq + right->freq);
```

```
99          parent->left = left;
100         parent->right = right;
101         insertMinHeap(heap, parent);
102     }
103     Node *root = extractMin(heap);
104     free(heap->nodes);
105     free(heap);
106     return root;
107 }
108
109 void generateCodes(Node *root, uint8_t *code, int depth,
        HuffCode *codes) {
110     if (!root) return;
111     if (!root->left && !root->right) {
112         codes[root->value].bits = (uint8_t*)malloc(depth);
113         memcpy(codes[root->value].bits, code, depth);
114         codes[root->value].length = depth;
115         return;
116     }
117     if (root->left) {
118         code[depth] = 0;
119         generateCodes(root->left, code, depth + 1, codes);
120     }
121     if (root->right) {
122         code[depth] = 1;
123         generateCodes(root->right, code, depth + 1, codes);
124     }
125 }
126
127 void writeTree(FILE *fp, Node *root) {
128     if (!root) return;
129     if (!root->left && !root->right) {
130         fputc(1, fp);
131         fputc(root->value, fp);
132     } else {
133         fputc(0, fp);
134         writeTree(fp, root->left);
```

```
135         writeTree(fp, root->right);
136     }
137 }
138
139 void writeCompressedFile(const char *filename, uint8_t *data,
        int size,
140                         HuffCode *codes, Node *tree, int width
                                , int height) {
141     FILE *fp = fopen(filename, "wb");
142     if (!fp) return;
143     fprintf(fp, "HUFF");
144     fwrite(&width, sizeof(int), 1, fp);
145     fwrite(&height, sizeof(int), 1, fp);
146     writeTree(fp, tree);
147     long totalBits = 0;
148     for (int i = 0; i < size; i++) totalBits += codes[data[i]].
            length;
149     int compressedBytes = (totalBits + 7) / 8;
150     fwrite(&compressedBytes, sizeof(int), 1, fp);
151     fwrite(&totalBits, sizeof(long), 1, fp);
152     uint8_t buffer = 0;
153     int bitPos = 0;
154     for (int i = 0; i < size; i++) {
155         HuffCode code = codes[data[i]];
156         for (int j = 0; j < code.length; j++) {
157             buffer = (buffer << 1) | code.bits[j];
158             bitPos++;
159             if (bitPos == 8) {
160                 fputc(buffer, fp);
161                 buffer = 0;
162                 bitPos = 0;
163             }
164         }
165     }
166     if (bitPos > 0) {
167         buffer <<= (8 - bitPos);
168         fputc(buffer, fp);
```

```
169     }
170     fclose(fp);
171 }
172
173 uint8_t* readPPM(const char *filename, int *width, int *height)
        {
174     FILE *fp = fopen(filename, "rb");
175     if (!fp) return NULL;
176     char format[3];
177     int maxval;
178     fscanf(fp, "%s", format);
179     char c = fgetc(fp);
180     while (c == '#') {
181         while (fgetc(fp) != '\n');
182         c = fgetc(fp);
183     }
184     ungetc(c, fp);
185     fscanf(fp, "%d %d %d", width, height, &maxval);
186     fgetc(fp);
187     int size = (*width) * (*height) * 3;
188     uint8_t *data = (uint8_t*)malloc(size);
189     if (strcmp(format, "P6") == 0) {
190         fread(data, 1, size, fp);
191     } else {
192         for (int i = 0; i < size; i++) {
193             int val;
194             fscanf(fp, "%d", &val);
195             data[i] = (uint8_t)val;
196         }
197     }
198     fclose(fp);
199     return data;
200 }
201
202 uint8_t* readBMP(const char *filename, int *width, int *height)
        {
203     FILE *fp = fopen(filename, "rb");
```

```
    if (!fp) return NULL;
    uint8_t fileHeader[14];
    fread(fileHeader, 1, 14, fp);
    if (fileHeader[0] != 'B' || fileHeader[1] != 'M') {
        fclose(fp);
        return NULL;
    }
    uint32_t dataOffset = *(uint32_t*)&fileHeader[10];
    uint8_t infoHeader[40];
    fread(infoHeader, 1, 40, fp);
    *width = *(int32_t*)&infoHeader[4];
    *height = abs(*(int32_t*)&infoHeader[8]);
    uint16_t bitsPerPixel = *(uint16_t*)&infoHeader[14];
    uint32_t compression = *(uint32_t*)&infoHeader[16];
    if (bitsPerPixel != 24 || compression != 0) {
        fclose(fp);
        return NULL;
    }
    fseek(fp, dataOffset, SEEK_SET);
    int rowSize = (*width) * 3;
    int padding = (4 - (rowSize % 4)) % 4;
    int size = (*width) * (*height) * 3;
    uint8_t *data = (uint8_t*)malloc(size);
    uint8_t *tempRow = (uint8_t*)malloc(rowSize + padding);
    for (int y = *height - 1; y >= 0; y--) {
        fread(tempRow, 1, rowSize + padding, fp);
        for (int x = 0; x < *width; x++) {
            int srcIdx = x * 3;
            int dstIdx = (y * (*width) + x) * 3;
            data[dstIdx + 0] = tempRow[srcIdx + 2];
            data[dstIdx + 1] = tempRow[srcIdx + 1];
            data[dstIdx + 2] = tempRow[srcIdx + 0];
        }
    }
    free(tempRow);
    fclose(fp);
    return data;
```

```
241 }
242
243 void freeCodes(HuffCode *codes) {
244     for (int i = 0; i < 256; i++) {
245         if (codes[i].bits) free(codes[i].bits);
246     }
247 }
248
249 void freeTree(Node *root) {
250     if (!root) return;
251     freeTree(root->left);
252     freeTree(root->right);
253     free(root);
254 }
255
256 int main(int argc, char *argv[]) {
257     if (argc < 3) return 1;
258     const char *ext = strrchr(argv[1], '.');
259     int width, height;
260     uint8_t *data = NULL;
261     if (ext && (strcmp(ext, ".bmp") == 0 || strcmp(ext, ".BMP")
            == 0)) {
262         data = readBMP(argv[1], &width, &height);
263     } else if (ext && (strcmp(ext, ".ppm") == 0 || strcmp(ext,
           ".PPM") == 0)) {
264         data = readPPM(argv[1], &width, &height);
265     } else {
266         return 1;
267     }
268     if (!data) return 1;
269     int size = width * height * 3;
270     int freq[256] = {0};
271     for (int i = 0; i < size; i++) freq[data[i]]++;
272     Node *tree = buildHuffmanTree(freq);
273     HuffCode codes[256] = {0};
274     uint8_t code[MAX_CODE_LEN];
275     generateCodes(tree, code, 0, codes);
```

```
    writeCompressedFile(argv[2], data, size, codes, tree, width
        , height);
    freeCodes(codes);
    freeTree(tree);
    free(data);
    return 0;
}
```

shlr.c

Listing 3: Archivo `shlr.c` - Aplicación de operaciones de desplazamiento (Shift) y rotación (Rotate)

```
#include "stdio.h"
#include "stdlib.h"

int main(int argc, char **argv)
{
    if (argc < 3) {
        printf("Uso: %s <imagen_entrada.bmp> <imagen_salida_limpia.bmp>\n", argv[0]);
        return 1;
    }

    char *fileIN = argv[1];
    char *fileOUT = argv[2];

    FILE *imgIN, *imgOUT;
    imgIN = fopen(fileIN, "rb");
    imgOUT = fopen(fileOUT, "wb");

    if (imgIN == NULL || imgOUT == NULL) {
        printf("Error: No se pudo abrir uno de los archivos.\n");
        return 1;
    }

    char head[54];
    fread(head, 1, 54, imgIN);
    fwrite(head, 1, 54, imgOUT);

    int c;

    while((c = fgetc(imgIN)) != EOF)
    {
```

```
31          c = c >> 3;
32          c = c << 3;
33          fputc(c, imgOUT);
34      }
35
36      fclose(imgIN);
37      fclose(imgOUT);
38
39      printf("Archivo limpio guardado en: %s\n", fileOUT);
40
41      return 0;
42  }
```

rectangulo.c

Listing 4: Archivo `rectangulo.c` - Detección del rectángulo máximo en imagen contenedora

```c
#include <stdio.h>
#include <math.h>

void encontrarMaximos(int N, int *a, int *b) {
    int raiz = (int)sqrt(N);

    for (int i = raiz; i >= 1; i--) {
        if (N % i == 0) {
            *a = i;
            *b = N / i;
            return;
        }
    }
}

int main() {
    int N, a, b;

    printf("Ingrese N: ");
    scanf("%d", &N);

    encontrarMaximos(N, &a, &b);

    printf("\nPara N = %d:\n", N);
    printf("Valores maximos: a = %d, b = %d\n", a, b);
    printf("Verificacion: %d × %d = %d\n", a, b, a*b);

    return 0;
}
```

descompresion.c

Listing 5: Script `descompresion.c` - Descompresión de imagen oculta usando decodificación de Huffman

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdint.h>

typedef struct Node {
    uint8_t value;
    int freq;
    struct Node *left, *right;
} Node;

Node* createNode(uint8_t value, int freq) {
    Node *node = (Node*)malloc(sizeof(Node));
    node->value = value;
    node->freq = freq;
    node->left = node->right = NULL;
    return node;
}

Node* readTree(FILE *fp) {
    int marker = fgetc(fp);
    if (marker == EOF) return NULL;
    if (marker == 1) {
        uint8_t value = fgetc(fp);
        return createNode(value, 0);
    } else {
        Node *node = createNode(0, 0);
        node->left = readTree(fp);
        node->right = readTree(fp);
        return node;
    }
}
```

```
33
34 uint8_t* decompressData(FILE *fp, Node *tree, int
      compressedBytes, long totalBits, int expectedSize) {
35     uint8_t *compressed = (uint8_t*)malloc(compressedBytes);
36     if (!compressed) return NULL;
37     if (fread(compressed, 1, compressedBytes, fp) !=
          compressedBytes) {
38         free(compressed);
39         return NULL;
40     }
41     uint8_t *decompressed = (uint8_t*)malloc(expectedSize);
42     if (!decompressed) {
43         free(compressed);
44         return NULL;
45     }
46     int decompIdx = 0;
47     Node *current = tree;
48     long bitCount = 0;
49
50     for (int i = 0; i < compressedBytes && bitCount < totalBits
          ; i++) {
51         uint8_t byte = compressed[i];
52         for (int bit = 7; bit >= 0 && bitCount < totalBits; bit
              --) {
53             bitCount++;
54             int currentBit = (byte >> bit) & 1;
55             current = currentBit ? current->right : current->
                  left;
56             if (!current) {
57                 free(compressed);
58                 free(decompressed);
59                 return NULL;
60             }
61             if (!current->left && !current->right) {
62                 if (decompIdx >= expectedSize) {
63                     free(compressed);
64                     free(decompressed);
```

```
                    return NULL;
                }
                decompressed[decompIdx++] = current->value;
                current = tree;
            }
        }
    }
    free(compressed);
    return decompressed;
}

void writeBMP(const char *filename, uint8_t *data, int width,
    int height) {
    FILE *fp = fopen(filename, "wb");
    if (!fp) return;

    int rowSize = width * 3;
    int padding = (4 - (rowSize % 4)) % 4;
    int imageSize = (rowSize + padding) * height;
    int fileSize = 54 + imageSize;

    uint8_t fileHeader[14] = {
        'B','M',0,0,0,0,0,0,0,0,54,0,0,0
    };
    *(uint32_t*)&fileHeader[2] = fileSize;
    fwrite(fileHeader, 1, 14, fp);

    uint8_t infoHeader[40] = {
        40,0,0,0,0,0,0,0,0,0,0,0,1,0,24,0,
        0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
        0,0,0,0
    };
    *(int32_t*)&infoHeader[4] = width;
    *(int32_t*)&infoHeader[8] = height;
    *(uint32_t*)&infoHeader[20] = imageSize;
    fwrite(infoHeader, 1, 40, fp);

```

```
101     uint8_t paddingBytes[3] = {0,0,0};
102     for (int y = height - 1; y >= 0; y--) {
103         for (int x = 0; x < width; x++) {
104             int idx = (y * width + x) * 3;
105             fputc(data[idx+2], fp);
106             fputc(data[idx+1], fp);
107             fputc(data[idx], fp);
108         }
109         fwrite(paddingBytes, 1, padding, fp);
110     }
111     fclose(fp);
112 }
113
114 void writePPM(const char *filename, uint8_t *data, int width,
        int height) {
115     FILE *fp = fopen(filename, "wb");
116     if (!fp) return;
117     fprintf(fp, "P6\n%d %d\n255\n", width, height);
118     fwrite(data, 1, width * height * 3, fp);
119     fclose(fp);
120 }
121
122 void freeTree(Node *root) {
123     if (!root) return;
124     freeTree(root->left);
125     freeTree(root->right);
126     free(root);
127 }
128
129 int main(int argc, char *argv[]) {
130     if (argc < 3) return 1;
131
132     FILE *fp = fopen(argv[1], "rb");
133     if (!fp) return 1;
134
135     char magic[5];
136     fread(magic, 1, 4, fp);
```

```
    magic[4] = '\0';
    if (strcmp(magic, "HUFF") != 0) {
        fclose(fp);
        return 1;
    }

    int width, height;
    fread(&width, sizeof(int), 1, fp);
    fread(&height, sizeof(int), 1, fp);

    Node *tree = readTree(fp);
    if (!tree) {
        fclose(fp);
        return 1;
    }

    int compressedBytes;
    long totalBits;
    fread(&compressedBytes, sizeof(int), 1, fp);
    fread(&totalBits, sizeof(long), 1, fp);

    int expectedSize = width * height * 3;
    uint8_t *data = decompressData(fp, tree, compressedBytes,
        totalBits, expectedSize);
    fclose(fp);

    if (!data) {
        freeTree(tree);
        return 1;
    }

    const char *ext = strrchr(argv[2], '.');

    if (ext && (!strcmp(ext,".bmp") || !strcmp(ext,".BMP")))
        writeBMP(argv[2], data, width, height);
    else if (ext && (!strcmp(ext,".ppm") || !strcmp(ext,".PPM")
        ))
```

```
172         writePPM(argv[2], data, width, height);
173     else {
174         char bmpName[256];
175         snprintf(bmpName, sizeof(bmpName), "%s.bmp", argv[2]);
176         writeBMP(bmpName, data, width, height);
177     }
178 
179     freeTree(tree);
180     free(data);
181 
182     return 0;
183 }
```

niveles8.c

Listing 6: Archivo `niveles8.c` - Variante para reducción de profundidad de color a 8 bits

```c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
    if(argc != 3) {
        printf("Uso: %s archivo_entrada archivo_salida\n", argv[0]);
        return 1;
    }

    char *input_file = argv[1];
    char *output_file = argv[2];

    FILE *in = fopen(input_file, "rb");
    if(!in) {
        perror("Error al abrir archivo de entrada");
        return 1;
    }

    FILE *out = fopen(output_file, "wb");
    if(!out) {
        perror("Error al abrir archivo de salida");
        fclose(in);
        return 1;
    }

    unsigned char buffer;
    while(fread(&buffer, 1, 1, in) == 1) {
        fwrite(&buffer, 1, 1, out);
    }

    fclose(in);
```

```
32      fclose(out);
33
34      printf("Reconstruccion completada: %s\n", output_file);
35      return 0;
36  }
```

Python

lsbimagen.py

Listing 7: Script `lsbimagen.py` - Inserción de imagen comprimida en LSB de imagen

```
from PIL import Image
import matplotlib.pyplot as plt
import sys

def bytes_to_binary(data_bytes):
    return ''.join(format(byte, '08b') for byte in data_bytes)

def hide_huff_in_image(cover_image_path, huff_payload_path,
    output_path):
    try:
        with open(huff_payload_path, 'rb') as f:
            huff_bytes = f.read()
    except FileNotFoundError:
        print("Error: archivo no encontrado:",
            huff_payload_path)
        return

    binary_payload = bytes_to_binary(huff_bytes)
    end_marker = '1' * 64 + '0' * 8
    binary_data_to_hide = binary_payload + end_marker

    cover_image = Image.open(cover_image_path)
    if cover_image.mode != 'RGB':
        cover_image = cover_image.convert('RGB')

    max_bits = cover_image.width * cover_image.height * 3
    if len(binary_data_to_hide) > max_bits:
        raise ValueError("El archivo es demasiado grande para
            ocultarse en la imagen.")

```

```
    pixels = list(cover_image.getdata())
    new_pixels = []
    data_index = 0

    for pixel in pixels:
        r, g, b = pixel
        new_pixel = []
        for component in (r, g, b):
            if data_index < len(binary_data_to_hide):
                bit = int(binary_data_to_hide[data_index])
                new_component = (component & ~1) | bit
                data_index += 1
            else:
                new_component = component
            new_pixel.append(new_component)
        new_pixels.append(tuple(new_pixel))

    new_image = Image.new(cover_image.mode, cover_image.size)
    new_image.putdata(new_pixels)
    new_image.save(output_path)

    print("Archivo ocultado en:", output_path, "(" + str(len(
        huff_bytes)) + " bytes)")

def mostrar_sin_margenes(img, titulo):
    fig = plt.figure(titulo, frameon=False)
    fig.set_size_inches(img.width / 100, img.height / 100)
    ax = plt.Axes(fig, [0, 0, 1, 1])
    ax.set_axis_off()
    fig.add_axes(ax)
    ax.imshow(img)

if __name__ == "__main__":
    if len(sys.argv) != 4:
        print("Uso: python lsbimagen.py <archivo_huff> <
            imagen_portadora> <imagen_salida>")
        sys.exit(1)
```

```
63
64      huff_payload = sys.argv[1]
65      imagen_portadora = sys.argv[2]
66      imagen_salida = sys.argv[3]
67
68      print("Ocultando archivo...")
69      hide_huff_in_image(imagen_portadora, huff_payload,
           imagen_salida)
70
71      try:
72          original = Image.open(imagen_portadora)
73          oculta = Image.open(imagen_salida)
74
75          mostrar_sin_margenes(original, "Imagen Original")
76          mostrar_sin_margenes(oculta, "Imagen Modificada")
77
78          plt.show()
79      except Exception as e:
80          print("Error al mostrar imagenes:", e)
```

extraerlsb1.py

Listing 8: Script `extraerlsb1.py` - Extracción de datos de imagen desde los LSB de la imagen estego

```
from PIL import Image
import sys
import os

sys.stdout.reconfigure(encoding='utf-8')

def binary_to_bytes(binary_data):
    if len(binary_data) % 8 != 0:
        binary_data = binary_data[:-(len(binary_data) % 8)]
    byte_list = []
    for i in range(0, len(binary_data), 8):
        byte_list.append(int(binary_data[i:i + 8], 2))
    return bytes(byte_list)

def extract_huff_bytes(image_path):
    if not os.path.exists(image_path):
        print(f"Error: no se encontro la imagen {image_path}")
        return False

    output_path = os.path.splitext(image_path)[0] + ".huff"

    try:
        image = Image.open(image_path).convert('RGB')
    except Exception as e:
        print(f"Error al abrir la imagen: {e}")
        return False

    binary_data = ''
    end_marker = '1' * 64 + '0' * 8
    marker_len = len(end_marker)

    for pixel in image.getdata():
```

```
33          for color_component in pixel[:3]:
34              binary_data += str(color_component & 1)
35
36              if len(binary_data) >= marker_len and binary_data[-
                    marker_len:] == end_marker:
37                  binary_payload = binary_data[:-marker_len]
38                  huff_bytes = binary_to_bytes(binary_payload)
39                  try:
40                      with open(output_path, 'wb') as f:
41                          f.write(huff_bytes)
42                      print(f"Archivo Huffman extraido
                            correctamente: {output_path}")
43                      return True
44                  except Exception as e:
45                      print(f"Error al guardar el archivo
                            extraido: {e}")
46                      return False
47
48      print("Marcador de fin no encontrado.")
49      return False
50
51  if __name__ == "__main__":
52      if len(sys.argv) == 2:
53          image_path = sys.argv[1].strip().strip('"').strip("'")
54      else:
55          image_path = input("Ruta de la imagen con payload: ").
                strip().strip('"').strip("'")
56
57      extract_huff_bytes(image_path)
```