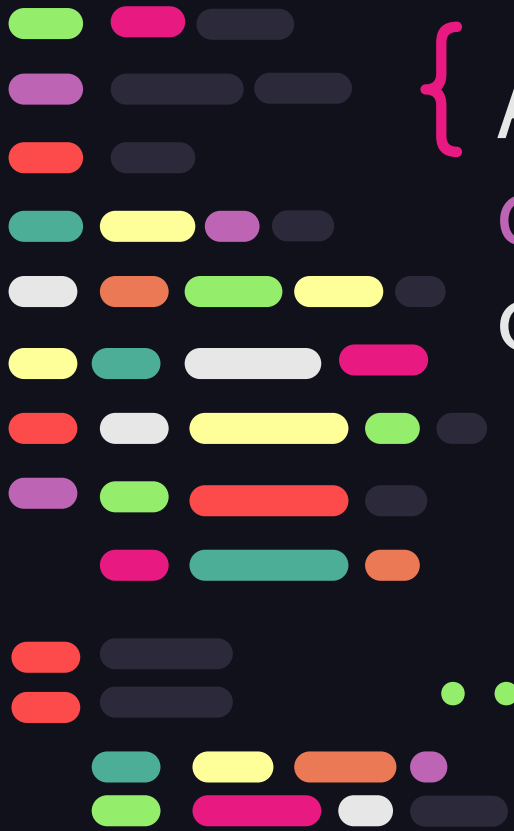


Algoritmo de análisis de archivos cifrados ocultos en imágenes

Análisis de Algoritmos

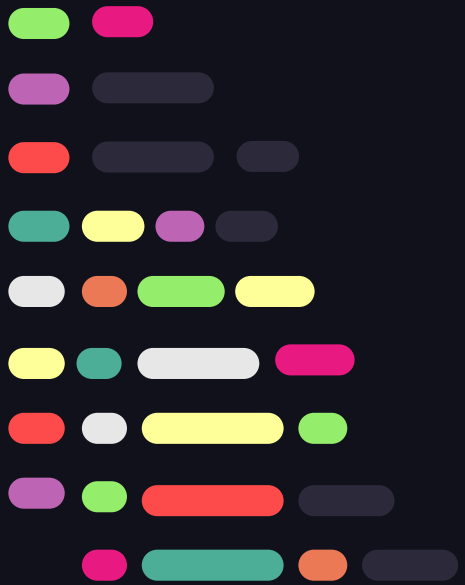
Prof.Jorge Ernesto Lopez Arce Delgado

Arroyo Moreno Elizabeth 221453749

Hernández Elizarrarás Karla Rebeca 223991977

Valencia Ignacio Jennifer Patricia 223991721

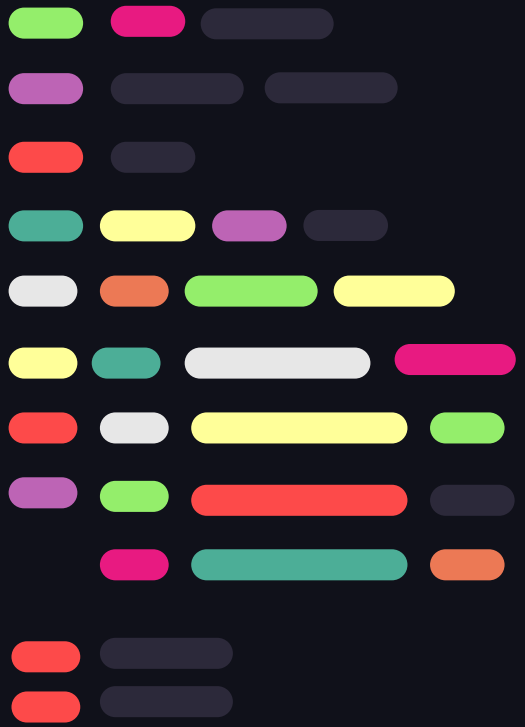
Algoritmo



Este algoritmo mete información en los bits menos importantes de las pixeles de una imagen y luego revisa si hay mensajes escondidos usando pruebas estadísticas en el canal rojo

Permite esconder mensajes en imagenes sin que se note y también detectar cambios o datos ocultos de manera rápida. Se puede usar en seguridad digital y en análisis forense de imágenes.

LSB (Algoritmo Base)



Usamos la técnica LSB (bit menos significativo) que agarra el último bit de cada pixel y lo cambia para esconder información. Funciona en imágenes con compresión sin pérdida, como PNG, JPG, TIFF, BMP.

LSB fue lo primero que usamos y luego lo mejoramos con otras técnicas.

Técnica Fuerza Bruta

```python
def extraer_mensaje_lsb(self, image_path):
    """Extraer mensaje oculto usando LSB del canal rojo"""
    try:
        if not self.load_image(image_path):
            return None

        canal_rojo = self.image[:, :, 0]  # Selecciona canal rojo
        lsb_bits = []
        height, width = canal_rojo.shape

        # Extrae el bit menos significativo de cada píxel
        for i in range(height):
            for j in range(width):
                lsb_bits.append(str(canal_rojo[i, j] & 1))

        mensaje_texto = ""
        # Convierte los bits extraídos en caracteres
        for i in range(0, len(lsb_bits), 8):
            byte = lsb_bits[i:i+8]
            if len(byte) < 8:
                continue
            char_code = int(''.join(byte), 2)
            if 32 <= char_code <= 126:
                mensaje_texto += chr(char_code)
```

Complejidad: O(n) donde n = m × m
Iteración secuencial pixel por pixel.

Ventajas:

- Simple.
- Directo.
- Fácil de implementar.

Desventajas:

- Se vuelve lento para imágenes grandes.
- Desperdicia memoria rápida del CPU.
- Procesa píxel por píxel (no paralelo).
- Muchos accesos lentos a memoria principal.
- No escalable: Empeora con imágenes de alta resolución.

Técnica Divide y vencerás.

Ventajas:

- Es más rápido.
- Mejor uso de caché.
- Puede ejecutarse en paralelo.
- Escalable para imágenes grandes.

Desventajas:

- Innecesario para imágenes pequeñas.
- Código más complejo.
- Usa más memoria temporal.

```python
def extraer_bits_divide_y_venceras(self, canal_2d, umbral_base=1024):
    """
    Extrae LSB dividiendo recursivamente la matriz en 4 bloques
    hasta llegar a bloques pequeños (caso base).
    """
    h, w = canal_2d.shape
    n = h * w
    if n == 0:
        return []
    if n <= umbral_base:
        return [str(px & 1) for fila in canal_2d for px in fila]

    mitad_h, mitad_w = h // 2, w // 2
    bloques = [
        canal_2d[:mitad_h, :mitad_w],
        canal_2d[:mitad_h, mitad_w:],
        canal_2d[mitad_h:, :mitad_w],
        canal_2d[mitad_h:, mitad_w:]
    ]
    salida = []
    for bloque in bloques:
        salida.extend(self.extraer_bits_divide_y_venceras(bloque, umbral_base))
    return salida
```

Técnica Huffman

Cuenta que caracteres se repiten mas y les da codigos mas cortos, asi el mensaje queda comprimido y cabe mejor dentro de una imagen

Ventajas:

- Reduce el tamaño del mensaje
- No pierde información
- Ahorra espacio
- Descompresion sencilla

Desventajas:

- Necesita construir un arbol
- No funciona con mensajes cortos
- Mas complejo de implementar

```
def construir_arbol(self, frecuencias):
    if not frecuencias:
        return None
    heap = [NodoHuffman(caracter=car, frecuencia=freq) for car, freq in frecuencias.items()]
    heapq.heapify(heap)
    while len(heap) > 1:
        izq = heapq.heappop(heap)
        der = heapq.heappop(heap)
        padre = NodoHuffman(frecuencia=izq.frecuencia + der.frecuencia, izquierda=izq, derecha=der
        heapq.heappush(heap, padre)
    return heap[0] if heap else None
```

Resultados

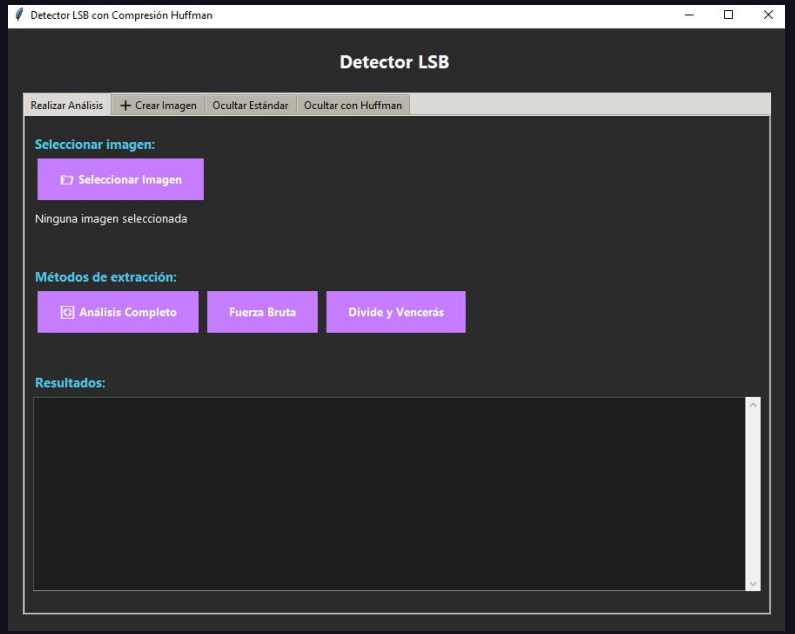
- **Fuerza Bruta:** detecta todo pero se tarda mucho
- **Divide y venceras:** hace lo mismo que la fuerza bruta pero más rápido
- **Huffman:** reduce el tamaño del texto sin perder nada de datos.

```
============================================================
Imagen cargada: (672, 1005, 3)

--------- EXTRACCIÓN ESTÁNDAR -----------

Método Fuerza Bruta:
Tiempo Fuerza Bruta: 0.16328 s

Método Divide y Vencerás:
Tiempo Divide y Vencerás: 0.19025 s

MENSAJE ESTÁNDAR ENCONTRADO: 'a dormir'

------------ EXTRACCIÓN HUFFMAN ----------
Ocultando mensaje con Huffman...
Mensaje original: 'a dormir'
Longitud: 8 caracteres

Compresión Huffman:
  - Cantidad de bits originales: 64
  - Bits comprimidos: 22
  - Ahorro: 65.6%

Bits totales: 840 bits
Imagen guardada: img_huffman.png
Canal usado: Rojo
```

Interfaz

1. Panel de Control y Entradas

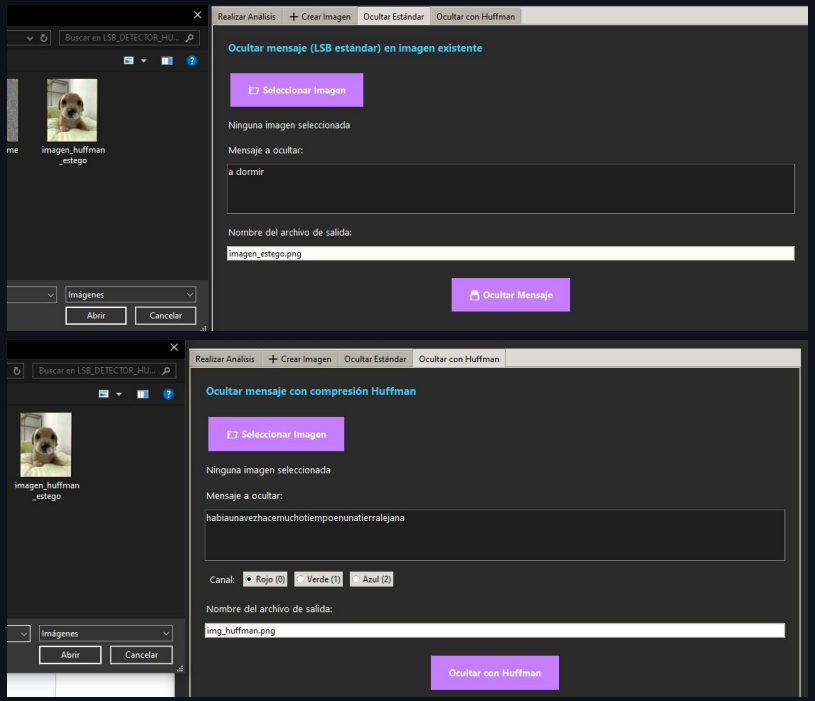
- Gestión de Archivos: Carga nativa de imágenes en formatos PNG y BMP.
- Configuración del Payload:
- Entrada de texto para mensaje secreto.
- Selector de Canal RGB y activación de Compresión Huffman.
- Ejecución: Botones independientes para Ocultar y Analizar.



Interfaz

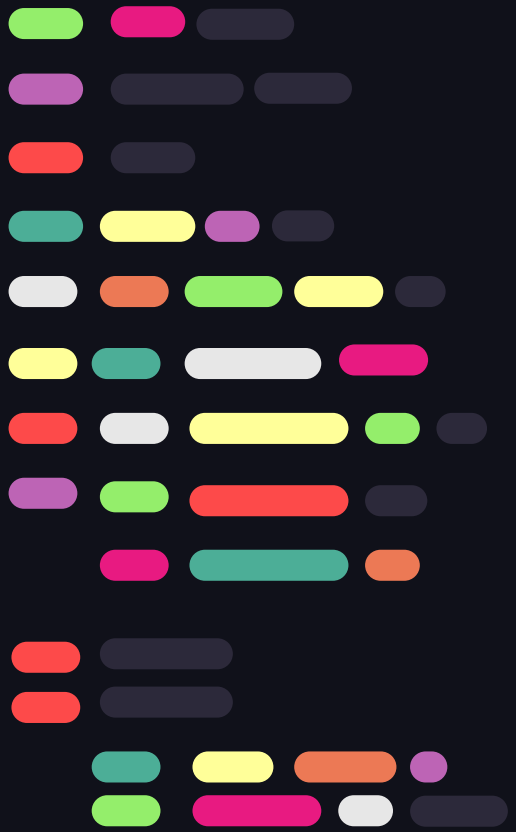
2. Consola de Resultados y Salidas

- Monitoreo: Logs de ejecución en tiempo real de los algoritmos de búsqueda.
- Criptoanálisis: Despliegue de métricas matemáticas como Entropía y Chi cuadrada.
- Veredicto: Indicador del estado final de la imagen como Limpia o Sospechosa.



Conclusión

Al combinar LSB con fuerza bruta, divide y venceras y huffman, pudimos esconder mensajes sin que se vean, analizar imágenes más rápido, detectar si tienen información oculta y gracias a la compresión, podemos agregar mensajes largos.



Gracias por su atención!

< Esperamos que les haya gustado >