

{ Algoritmo de análisis de archivos cifrados ocultos en imágenes



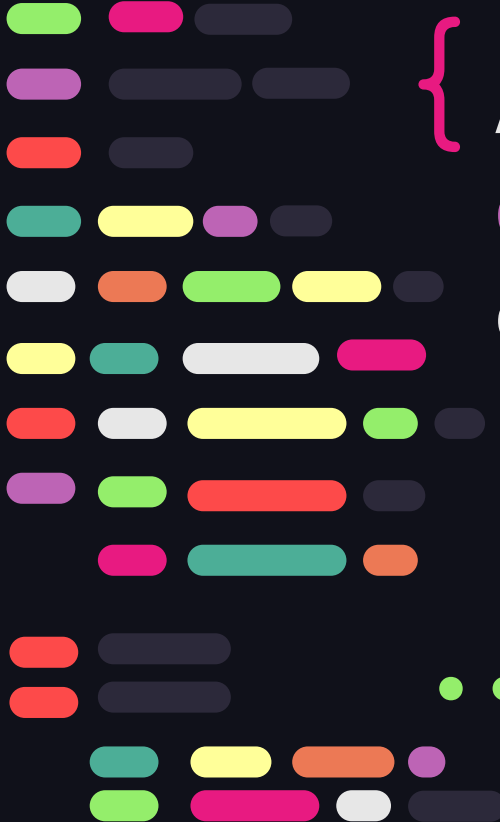
Análisis de Algoritmos

Prof. Jorge Ernesto Lopez Arce Delgado

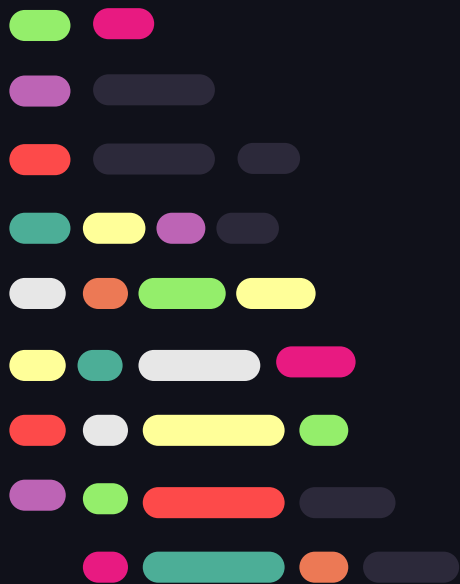
Arroyo Moreno Elizabeth 221453749

Hernández Elizarrarás Karla Rebeca 223991977

Valencia Ignacio Jennifer Patricia 223991721



Algoritmo



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

Permite esconder mensajes en imágenes 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.

Fuerza Bruta

```
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 $n = m \times m$

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.

Adaptación a Divide y vencerás

Divide la imagen recursivamente en 4 cuadrantes (bloques más pequeños) hasta llegar a un tamaño manejable (1024 píxeles), procesa cada bloque por separado y luego combina los resultados.



```
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
```

Divide y venceras.

Ventajas:

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

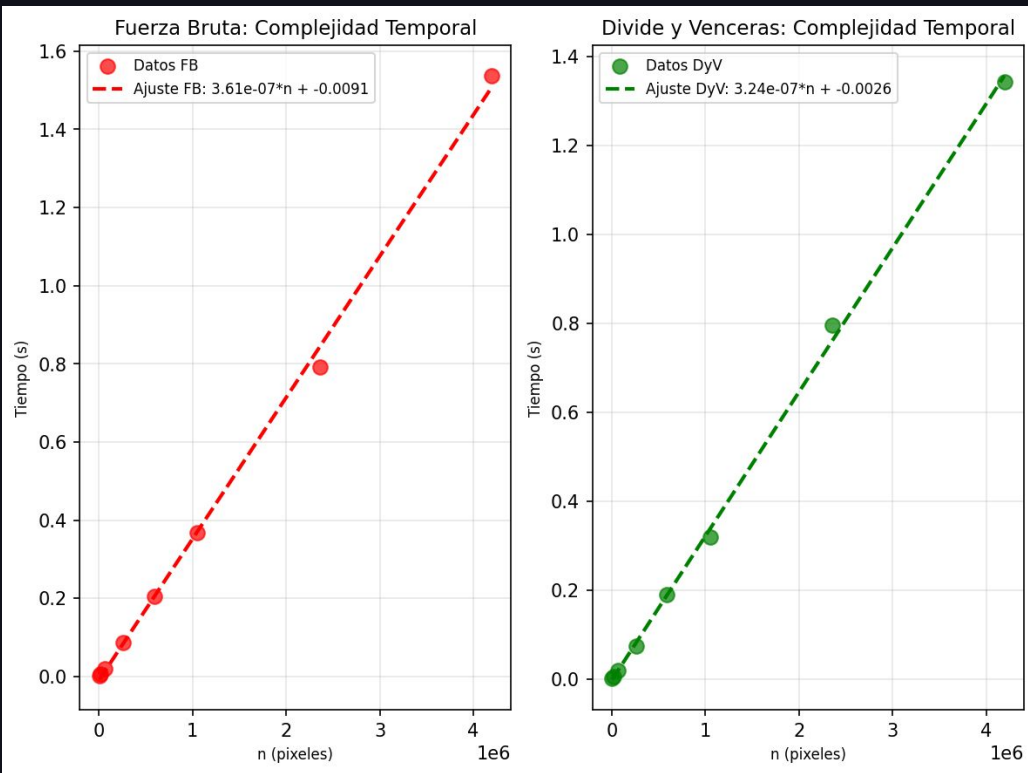
Desventajas:

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

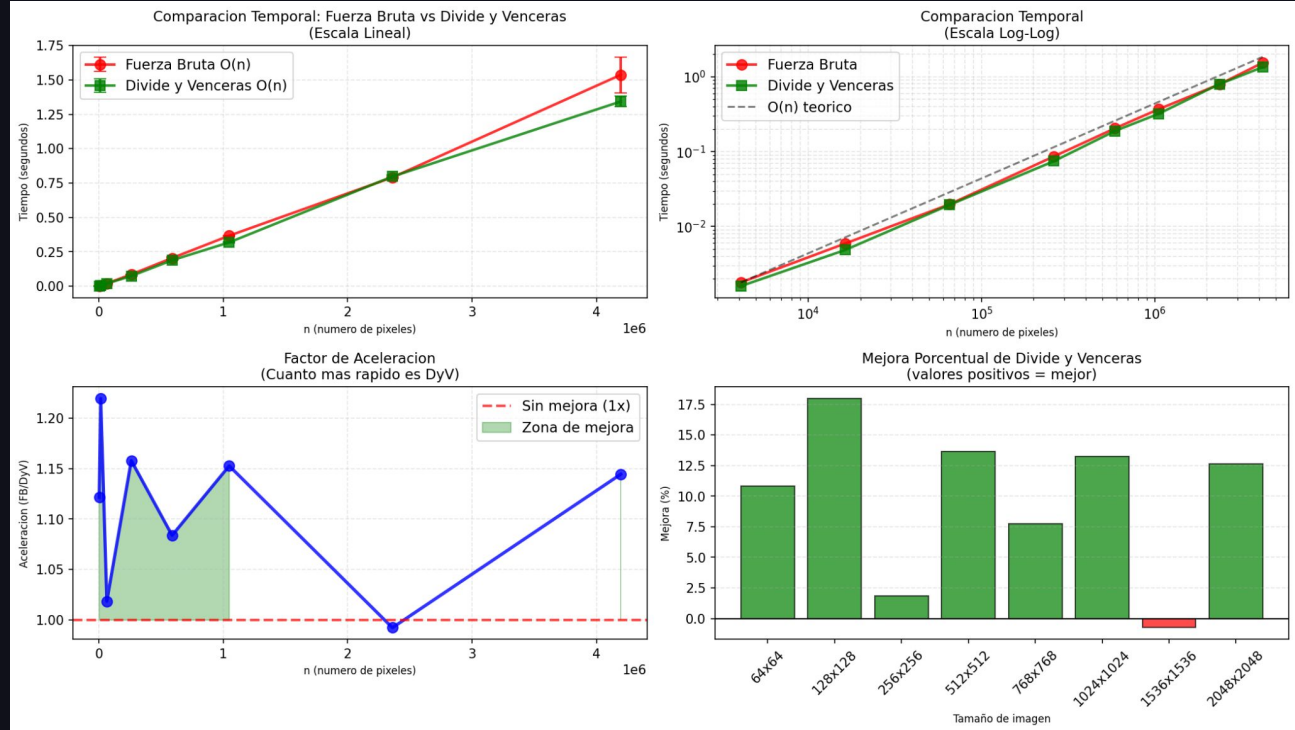
```
def extraer_bits_divide_y_venceras(self, canal_2d, umbral_base=1024):  
    """  
    Extrae LSB dividiendo recursivamente la matriz en 4 bloques  
    hasta llegar a bloques pequeños (caso base).  
    """  
    h, w = canal_2d.shape  
    n = h * w  
    if n == 0:  
        return []  
    if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

Análisis Complejidad temporal

Fuerza bruta vs Divide y vencerás



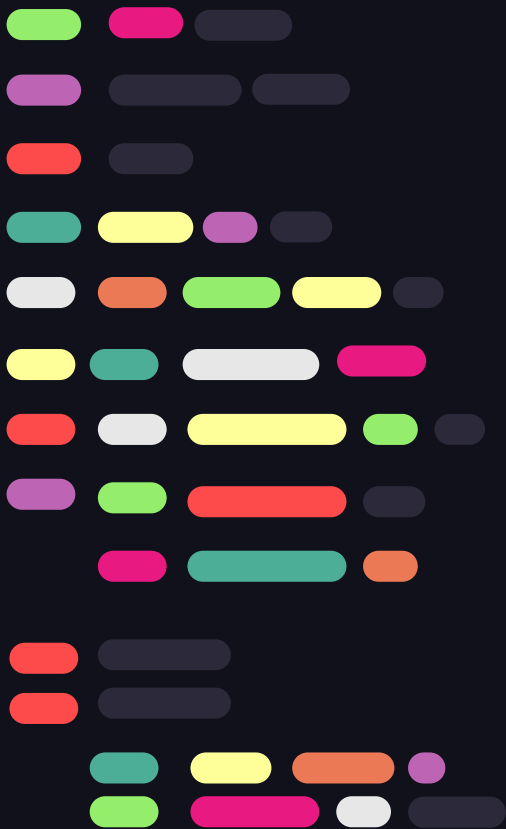
Comparación Completa



Conclusión

La adaptación de Fuerza Bruta a Divide y Vencerás en la extracción LSB demostró mejoras en velocidad para imágenes medianas y grandes, gracias al mejor uso de caché al procesar bloques pequeños. DyV es más eficiente en imágenes de 128×128 a 1536×1536 píxeles. No es recomendable para imágenes muy pequeñas ni extremadamente grandes. Se podría mejorar implementando paralelización real, un sistema híbrido automático según tamaño de imagen.





Gracias por su
atención!

< Esperamos que les haya gustado >

