UNIVERSIDAD DE GUADALAJARA

Centro Universitario de Ciencias Exactas e Ingenierías

División de Tecnologías para la Integración

Ciber-Humana

Análisis de algoritmos

Jennifer Patricia Valencia Ignacio, Código: 223991721

Elizabeth Arroyo Moreno, Código: 221453749

Karla Rebeca Hernández Elizarrarás, Código: 223991977

Ingeniería en computación

Proyecto Final

26 de Noviembre de 2025

Índice

Índice **2**
Introducción **3**
Objetivos **4**
General 4
Específicos 4
Desarrollo **5**
Descripción del problema 5
Implementación de cada técnica 5
Fuerza Bruta 5
Divide y Vencerás 5
Técnica Voraz Huffman 5
Resultados 5
Comparativas o diagramas 6
Conclusión **7**
Conclusión General 7
Jenny 7
Eli 7
Rebe **7**
Referencias **8**

Introducción

La esteganografía oculta información en diferentes tipos de archivos y existen diferentes maneras de ocultar mensajes sin que nadie lo note. Si bien existen múltiples metodologías para lograr este ocultamiento sin alterar la percepción del archivo, hemos desarrollado nuestro proyecto durante todo el semestre implementando nuevas técnicas aplicadas en la LSB Least Significant Bit, integrando progresivamente nuevas técnicas aplicadas que potencian la funcionalidad básica del modelo LSB.

Una de las aplicaciones más destacadas son técnicas como el algoritmo de Huffman, ocultando mensajes todavía más largos por la compresión de nuestro mensaje, comprimiendo nuestro mensaje antes de ocultarlo en nuestra imagen. Con esto nuestro sistema es capaz de ocultar mensajes más largos que a que si se ocultaran con otros métodos sin compresión previa, decidimos enfocar el proyecto en imágenes pero estas técnicas se pueden implementar en MP4, WAV, MP3, hasta en PDFs.

Finalmente, no sólo ocultamos los mensajes si no también podemos detectar si alguna imágen tiene información oculta, creando además un módulo de criptoanálisis diseñado para identificar si una imagen contiene información oculta, ya que el módulo fundamenta el análisis estadístico y su ejecución de algoritmos de búsqueda exhaustiva, encargandose y asegurandose de una validación integral en el ocultamiento y detección de mensajes.

Objetivos

General

El objetivo es combinar varias técnicas de algoritmos para comprimir, ocultar y analizar mensajes dentro de imágenes y demostrar cómo diferentes métodos pueden trabajar juntas y hacen el proceso más rápido.

Específicos

- Implementar la esteganografía LSSB para poder ocultar mensajes dentro de imágenes tomando solo el bit menos significativo de cada pixel. Con esto se busca que el mensaje quede escondido sin cambiar la imagen de una forma visible y que el proceso sea fácil de aplicar.

- Aplicar un método de compresión sirve para reducir el tamaño del mensaje antes de meterlo en la imagen, lo que ayuda a que ocupe menos espacio, sea más fácil de guardar y permita que incluso las imágenes más pequeñas puedan guardar información sin problemas.

- Hacer pruebas y análisis simples sirve para identificar si una imagen tiene datos escondidos, ayuda a comprobar si el algoritmo funciona bien y entender cómo cambian las imágenes cuando tienen alguna información oculta.

Desarrollo

Descripción del problema

El problema principal del proyecto es poder detectar o ocultar información dentro de imágenes usando esteganografía con la técnica LSB (el bit menos significativo). La esteganografía funciona cambiando el último bit de los píxeles para meter un mensaje sin que nadie lo note a simple vista. Lo que buscamos es encontrar patrones que se repitan en los bits haciendo más fácil la detección de anomalías. Con las últimas implementaciones de nuestro proyecto es capaz de comprimir un mensaje para la alteración de imágenes usando el RGB y dándole la opción al usuario de decidir en dónde ocultar el mensaje.

Implementación de cada técnica.

Fuerza Bruta

Este es el algoritmo base. Revisa cada píxel uno por uno, saca el último bit del canal rojo para ver si se puede recuperar un mensaje o si hay algo raro que indique que la imagen fue modificada.

```
def _lsb_bits_brute(self, channel_2d):
    """Extrae bits LSB usando fuerza bruta"""
    h, w = channel_2d.shape
    bits = []
    for i in range(h):
        for j in range(w):
            bits.append(str(channel_2d[i, j] & 1))
    return bits
```

Divide y Vencerás

Esta técnica consiste en partir la imagen en partes más pequeñas, analizarlas y decidir si continuamos revisando esa zona o no. Primero se divide la imagen en dos partes, luego a cada una le hacemos un análisis básico y si la zona se ve normal, la descartamos pero si la zona se ve sospechosa la volvemos a dividir y la revisamos con más detalle.

```
def _lsb_bits_divide_and_conquer(self, channel_2d, base_threshold=1024):
    """Extrae bits LSB usando divide y vencerás"""
    h, w = channel_2d.shape
    n = h * w
    if n == 0:
        return []
    if n <= base_threshold:
        return [str(px & 1) for fila in channel_2d for px in fila]
    mh, mw = h // 2, w // 2
    bloques = [
        channel_2d[:mh, :mw],
```

```
            channel_2d[:mh, mw:],
            channel_2d[mh:, :mw],
            channel_2d[mh:, mw:]
        ]
        out = []
        for b in bloques:
            out.extend(self._lsb_bits_divide_and_conquer(b, base_threshold))
        return out
```

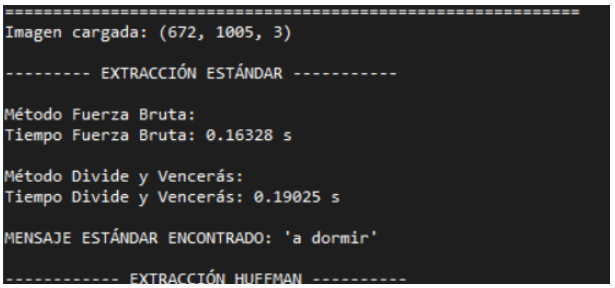
Técnica Voraz Huffman

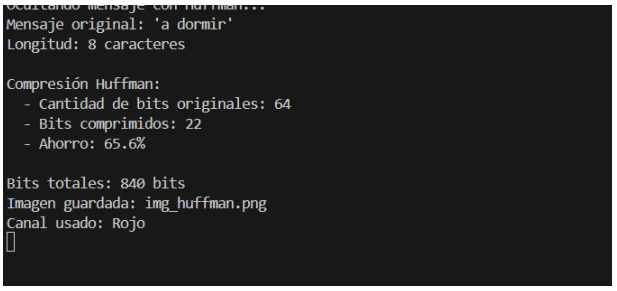
También usamos la técnica de huffman, sirve para comprimir texto sin perder información. Antes de esconder el mensaje en la imagen podemos comprimirlo para que ocupe menos espacio. Huffman cuenta cuantas veces aparece cada caracter, luego crea una árbol juntando los dos que aparecen menos.

```
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
    def decodificar_texto(self, texto_binario, codigos):
        if not texto_binario or not codigos:
            return ""
        codigos_inv = {v: k for k, v in codigos.items()}
        texto_decodificado = []
        codigo_actual = ""
        for bit in texto_binario:
            codigo_actual += bit
            if codigo_actual in codigos_inv:
                texto_decodificado.append(codigos_inv[codigo_actual])
                codigo_actual = ""
        return ''.join(texto_decodificado)
```

Resultados

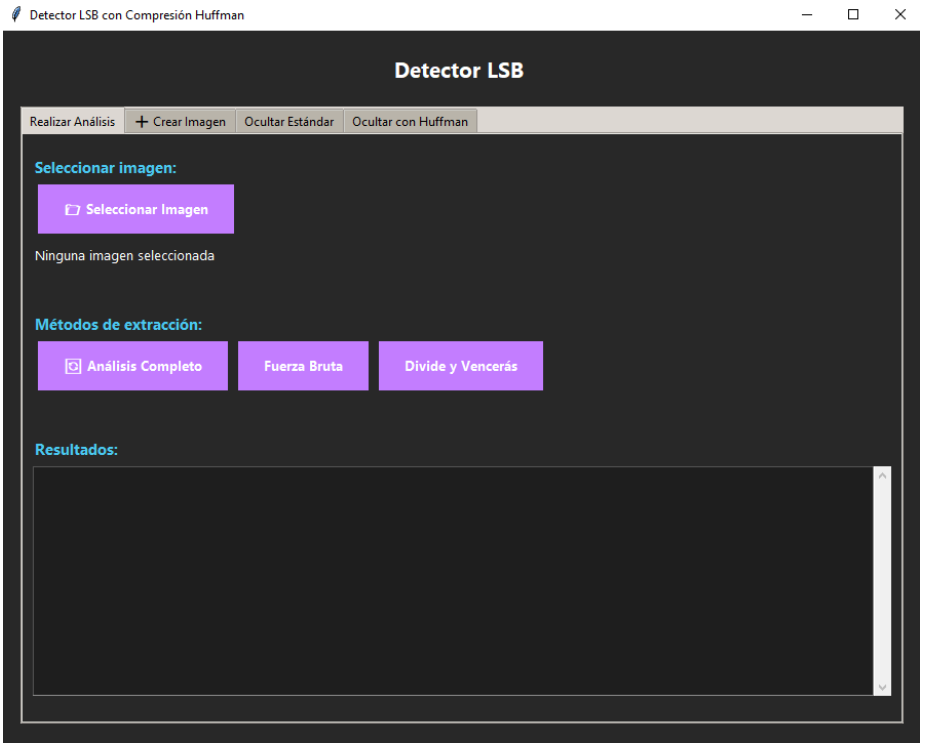
En fuerza bruta, el algoritmo analiza absolutamente todo, lo que lo hace muy preciso, pero también lo hace lento, más en imágenes grandes. Con la técnica de divide y vencerás el tiempo de análisis bajo porque pudimos ignorar zonas completas que están limpias, y aun así mantener la misma precisión, en las pruebas detecta zonas sospechosas igual que la fuerza bruta, pero más rápido. Con huffman, se logra reducir el tamaño del texto antes de ocultarlo, esta técnica puede optimizar el espacio sin perder ningún dato, ya que la compresión y la descompresión funciona.



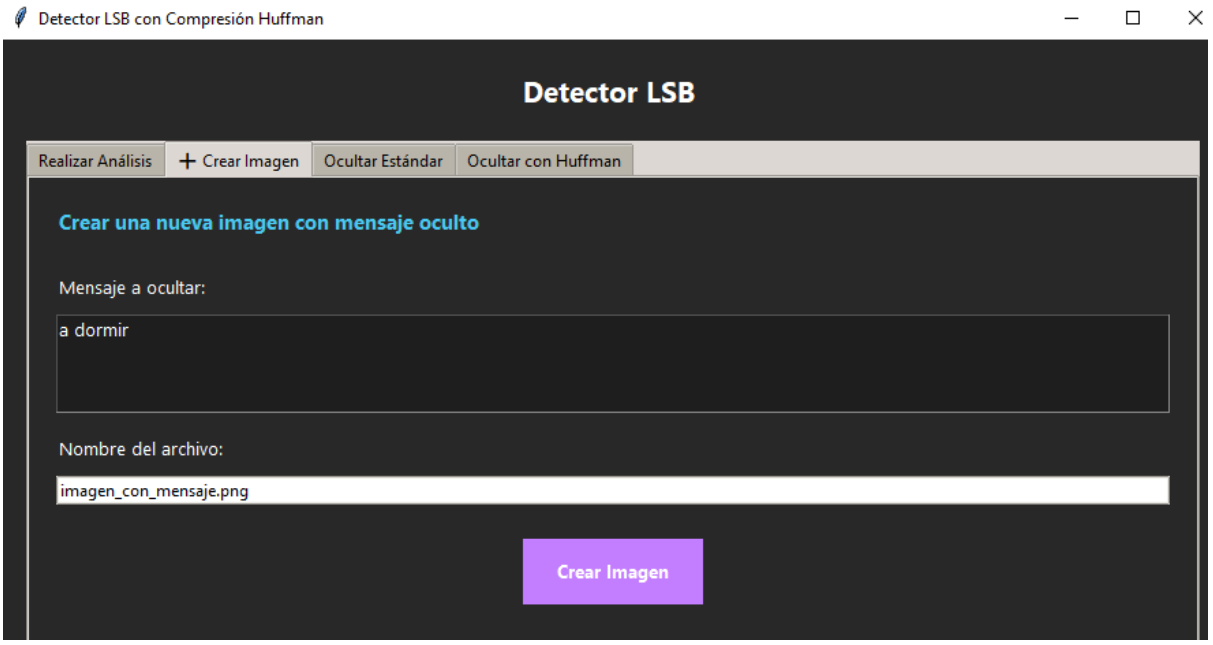


Interfaz

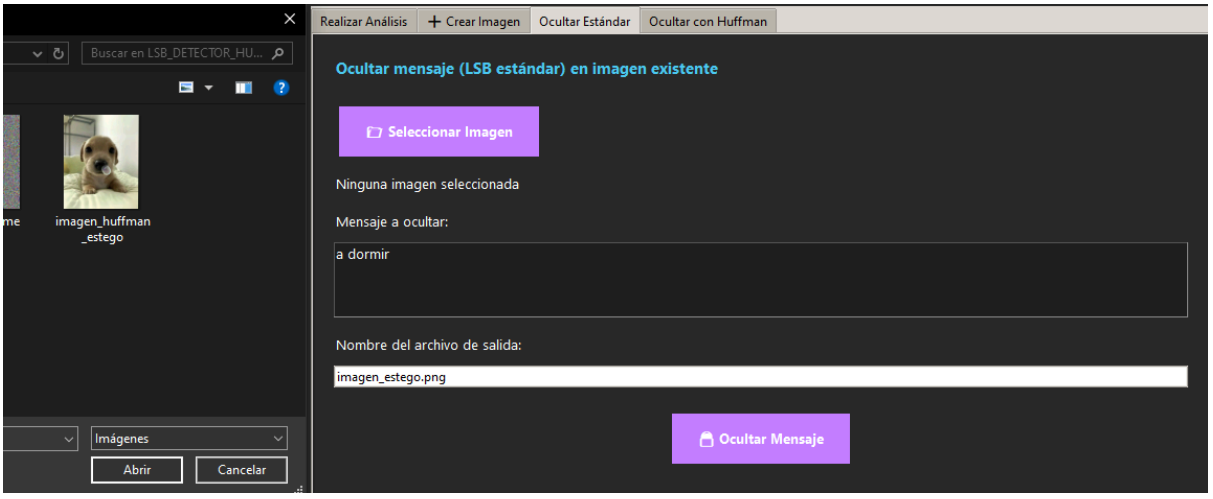
La interfaz gráfica de nuestra aplicación esta diseñada como un panel de control unificado que organiza lógicamente todas las herramientas del proyecto en una sola ventana. Al iniciar el programa, lo primero que encuentra el usuario en la zona superior izquierda es el módulo de carga de evidencia, el cual permite explorar el almacenamiento local para seleccionar la imagen portadora sobre la cual se trabajará, validando automáticamente que el formato sea compatible antes de habilitar las siguientes funciones.

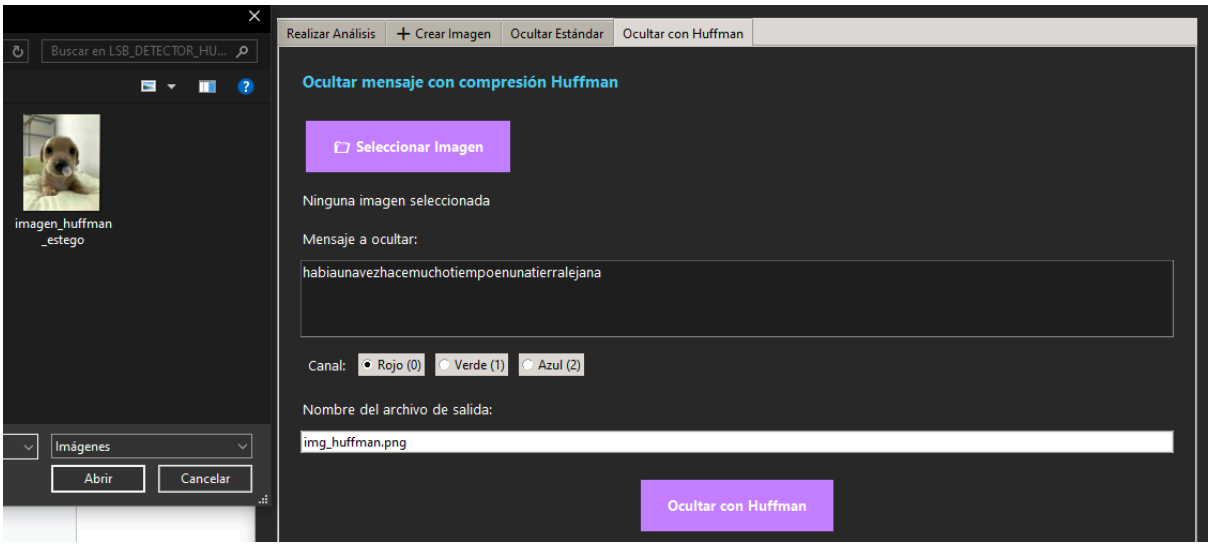


Justo debajo de la selección de archivos se abren los controles de operación, divididos según la intención del usuario. Por un lado, se encuentra el apartado de ocultamiento, el cual presenta cajas de texto para la redacción del mensaje secreto y campos para nombrar el archivo de salida; es aquí donde destaca la configuración específica para la técnica Huffman, la cual incluye un selector exclusivo de canales RGB que otorga al usuario el poder de decidir si desea inyectar los bits comprimidos en la capa roja, verde o azul de la imagen.

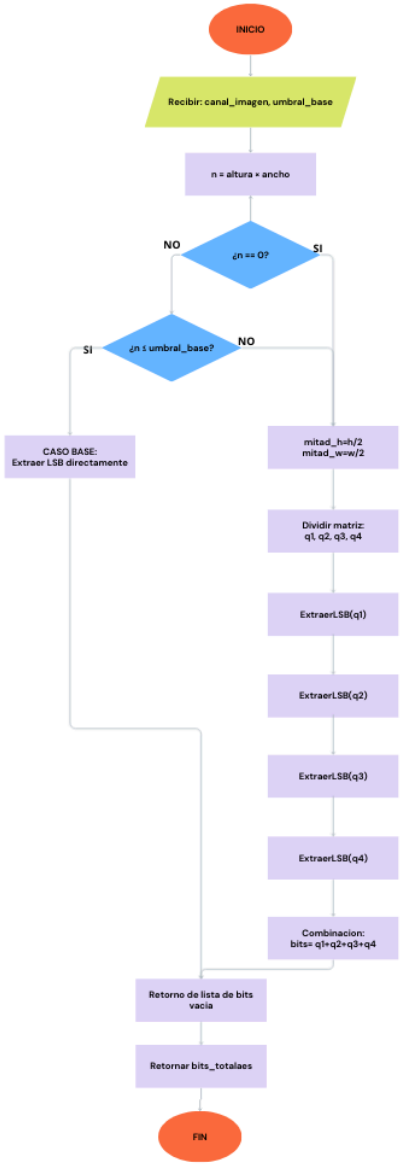


La interfaz integra los disparadores del módulo de criptoanálisis, permitiendo ejecutar con botones dedicados las rutinas de extracción y validación estadística sin necesidad de comandos externos. Finalmente, abarcando toda la mitad derecha de la ventana, se sitúa la consola de monitoreo en tiempo real, un área de texto dinámica con desplazamiento que actúa como la bitácora del sistema, narrando paso a paso lo que ocurre en el backend, desde los tiempos de ejecución de los algoritmos de fuerza bruta hasta el despliegue detallado de los valores matemáticos de entropía y Chi-cuadrada al finalizar un análisis.




Buscar en LSB_DETECTOR_HU...
imagen_huffman
_estego
Imágenes
Abrir
Cancelar
Realizar Análisis
Crear Imagen
Ocultar Estándar
Ocultar con Huffman
Ocultar mensaje con compresión Huffman
Seleccionar Imagen
Ninguna imagen seleccionada
Mensaje a ocultar:
habiaunavezhacemuchotiempoenunatierralejana
Canal:
Rojo (0)
Verde (1)
Azul (2)
Nombre del archivo de salida:
img_huffman.png
Ocultar con Huffman

Comparativas o diagramas.



Algoritmo	Ventajas	Desventajas	Conclusión
Fuerza Bruta	• Simplicidad: Menor uso de memoria (stack). • Velocidad base: Fue el más rápido en la imagen limpia (0.041s). • Implementación: Código directo y fácil de mantener.	• Escalabilidad: En imágenes de muy alta resolución, el tiempo crece linealmente sin posibilidad de optimización por bloques. • Bloqueo: En un solo hilo, bloquea la ejecución secuencialmente.	Ideal para comprobaciones rápidas en imágenes pequeñas o cuando se sospecha que la imagen está limpia.
Divide y Vencerás	• Rendimiento bajo carga: Fue más rápido extrayendo el mensaje oculto (0.047s vs 0.052s). • Paralelismo: Su estructura modular permitiría procesar cuadrantes en hilos separados (futura mejora).	• Overhead: La recursión añade un pequeño tiempo extra de gestión (notable en la imagen limpia) • Memoria: Mayor consumo de pila de llamadas (Stack Overflow risk en imágenes gigantes).	Demostró ser superior cuando existe información compleja que procesar, así justifica su implementación para el análisis forense profundo.

Conclusión

Conclusión General

Este proyecto logró unir varios algoritmos e implementarlos en algo funcional, el ocultar simples letras en una imagen, se transformó en un sistema integrando algoritmos complejos como Huffman de tal manera que sea capaz de ocultar y descifrar mensajes largos. La compresión fue lo que más logramos notar en nuestro proyecto además que nos llamó mucho la atención todo el desarrollo detrás de este.

Ver nuestra interfaz gráfica procesando imágenes, reduciendo el peso de los mensajes, nos hace notar que valió la pena. Este proyecto demostró que una buena idea es lo que realmente permite crear grandes sistemas y la planificación demuestra que una herramienta de seguridad sea robusta y capaz de crecer con nuevas funciones en el futuro siempre será capaz de evolucionar.

Jenny

Durante la implementación, comprendí que la eficiencia no es solo que el programa vaya rápido, sino también saber manejar bien el espacio. Cuando metimos Huffman,nos ayudó a que la imagen no se llenara demasiado y a poder guardar más información sin que explotara todo, al final entendí que saber adaptarse a diferentes cantidades de datos es lo que hace que un algoritmo pase de ser pura teoría a algo que realmente sirve en la vida real.

Eli

La implementación de Divide y Vencerás y las pruebas de Chi-cuadrada funcionan como puntos de control indispensables. De tal manera estas verificaciones aseguran la sincronización de los datos y mitigan riesgos de corrupción. Considero que esta estructura de validación fue la clave para manejar la complejidad de las alteraciones de los bit, garantizando que el flujo de ejecución se mantuviera estable y seguro ante cualquier entrada. Me llamó mucho la atención el algoritmo Huffman y la compresión de los mensajes.

Rebe

El método de Fuerza Bruta, me gustó comprender cómo un recorrido exhaustivo de la matriz de píxeles permite recuperar información sin pérdidas. De tal manera, la importancia del módulo de análisis de anomalías; notar cómo el sistema evolucionó desde una primera versión que arrojaba un simple nivel de probabilidad o puntuación de sospecha, me hizo valorar el impacto de la estadística aplicada en la detección de vulnerabilidades digitales.

Referencias

¿Qué es la esteganografía? ¿Cómo funciona? (2023, febrero 8). /. https://latam.kaspersky.com/resource-center/definitions/what-is-steganography

Sedgewick, R. (2013). An introduction to the analysis of algorithms. Pearson Education India.

¿Qué es el Bit Menos Significativo y Cómo Afecta a la Manipulación de Datos? (s/f). Lenovo.com. Recuperado el 27 de noviembre de 2025, de https://www.lenovo.com/es/es/glossary/least-significant-bit/?srsltid=AfmBOornhZSM_z5ylJ8mK8LAO5V3teYrSbKDP238o_nYxZexhhhMdFwa

Casero, A. (2023, diciembre 13). ¿Qué es el algoritmo de fuerza bruta en programación? *KeepCoding Bootcamps*. https://keepcoding.io/blog/algoritmo-de-fuerza-bruta-en-programacion/

freeCodeCamp. (2020, enero 6). *Brute Force Algorithms explained*. Freecodecamp.org. https://www.freecodecamp.org/news/brute-force-algorithms-explained/

(S/f). Medium.com. Recuperado el 27 de noviembre de 2025, de https://medium.com/@davidcabreraygarcia/la-recursividad-y-el-algoritmo-de-divide-y-vencerás-9418325e55b5

Levitin, A. (2008). Introduction to design and analysis of algorithms, 2/E. Pearson Education India.

Martínez, J. E. (2020, junio 10). *Algoritmia: Divide y vencerás*. Adictos al trabajo. https://adictosaltrabajo.com/2020/06/10/algoritmia-divide-y-venceras/

Nb, T. H. A. (2023, mayo 12). *What is a Greedy Algorithm? Examples of Greedy Algorithms*. Freecodecamp.org. https://www.freecodecamp.org/news/greedy-algorithms/

Huffman coding. (s/f). Programiz.com. Recuperado el 27 de noviembre de 2025, de https://www.programiz.com/dsa/huffman-coding