UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

Proyecto Final: Encriptación Huffman

Wilson Manuel Santos Ajcot
201907179

Catedrático:
Ing. Carlos Garrido

Curso:
Matemática para Computación 2

Tutor Académico:
Brayan Prado

22 de abril de 2025

Índice

1. Introducción

El algoritmo de Huffman es un método fundamental en la teoría de la información para la compresión de datos sin pérdida. Este proyecto implementa un sistema web completo que permite encriptar y desencriptar mensajes utilizando este algoritmo, junto con la visualización del árbol binario generado.

La solución desarrollada combina tecnologías modernas:

- Backend en Python con Flask para el procesamiento del algoritmo
- Frontend en React para una interfaz interactiva
- Graphviz para la generación visual del árbol
- Exportación a PDF para compartir resultados

2. Objetivos

2.1. General

Implementar un sistema web basado en el algoritmo de Huffman para encriptar y desencriptar mensajes, con visualización del árbol binario y exportación de resultados.

2.2. Específicos

- Desarrollar un backend en Python que implemente eficientemente el algoritmo de Huffman
- Crear una interfaz amigable en React para la interacción con usuarios
- Generar visualizaciones del árbol de Huffman usando Graphviz
- Implementar funcionalidad de exportación a PDF
- Validar el correcto funcionamiento del proceso completo de encriptación/desencriptación

3. Marco Teórico

3.1. Algoritmo de Huffman

El algoritmo de Huffman, desarrollado por David A. Huffman en 1952 durante sus estudios en el MIT, es un esquema de codificación óptimo para compresión de datos sin pérdida que revolucionó el campo de la teoría de la información. Este método utiliza codificación de longitud variable (VLC), asignando códigos más cortos a los caracteres más frecuentes y códigos más largos a los menos frecuentes, optimizando así el espacio de almacenamiento o transmisión.

3.1.1. Árbol de Huffman

Estructura de datos binaria fundamental para el algoritmo que organiza los caracteres según su frecuencia. Cada nodo hoja contiene un símbolo y su frecuencia, mientras que los nodos internos contienen la suma de las frecuencias de sus hijos. La construcción del árbol sigue estos principios:

- Los caracteres menos frecuentes se ubican en niveles más profundos
- No existe un código que sea prefijo de otro (propiedad de prefijo)
- Permite una decodificación unívoca sin ambigüedades

3.1.2. Codificación de Huffman

Proceso que transforma los caracteres en secuencias binarias mediante:

- Asignación de ‘0‘ a las ramas izquierdas y ‘1‘ a las derechas
- Construcción de códigos mediante el recorrido desde la raíz hasta cada hoja
- Generación de una tabla de codificación óptima

Matemáticamente, la longitud esperada L del código viene dada por:

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

donde p_i es la probabilidad del símbolo i y l_i su longitud de código.

3.2. Herramientas Complementarias

3.2.1. Graphviz

Software de visualización gráfica de código abierto que implementa el lenguaje DOT para representar estructuras de datos. En este proyecto se utiliza para:

- Visualizar el árbol de Huffman generado
- Mostrar las relaciones padre-hijo entre nodos
- Exportar los diagramas en formatos vectoriales (PDF, SVG)

3.2.2. Arquitectura de 2 Capas

Sistema cliente-servidor que divide la aplicación en:

Frontend (React) : Capa de presentación que incluye:

- Interfaz gráfica con formularios de entrada
- Visualización del árbol y códigos generados
- Gestión de eventos de usuario

Backend (Python/Flask) : Capa lógica que realiza:

- Cálculo de frecuencias y construcción del árbol
- Generación de códigos binarios
- Servicio de API REST para comunicación con el frontend

3.3. Procesos

3.3.1. Codificación

1. Cálculo de frecuencias mediante histograma de caracteres
2. Construcción del min-heap y posterior árbol binario
3. Generación de códigos mediante recorrido en profundidad (DFS)
4. Sustitución de caracteres por secuencias binarias

3.3.2. Decodificación

1. Recorrido bit a bit del mensaje codificado
2. Navegación por el árbol según los valores binarios
3. Reconstrucción del mensaje original al llegar a nodos hoja

4. Algoritmo de la Solución

4.1. Proceso de Encriptación

Algorithm 1 Encriptación Huffman

```
Calcular frecuencias de caracteres
Construir min-heap basado en frecuencias
while hay más de un nodo en el heap do
    Extraer los dos nodos con menor frecuencia
    Crear nuevo nodo interno con suma de frecuencias
    Insertar nuevo nodo en el heap
end while
El nodo restante es la raíz del árbol
Generar códigos recorriendo el árbol (0=izquierda, 1=derecha)
Codificar mensaje sustituyendo caracteres por códigos
```

4.2. Ejemplo Práctico

Entrada: Hola Mundo

Paso 1: Se ordenan los elementos por frecuencia (de menor a mayor), manteniendo el orden original en caso de empates. Se cuentan todas las letras, caracteres y espacios, almacenándolos en sus respectivos nodos.

Token	Frecuencia
H	1
o	2
l	1
a	1
ESPACIO	1
M	1
u	1
n	1
d	1

Paso 2: Reorganiza los elementos según su frecuencia de aparición (de menor a mayor), manteniendo el orden original de entrada cuando la frecuencia es igual. Cada letra se almacena en su nodo correspondiente.

1	1	1	1	1	1	1	1	2
H	l	a	espacio	M	u	n	d	o

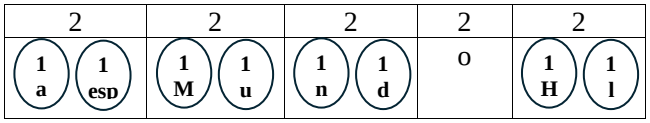
Paso 3: Se combinan los nodos para formar uno solo, actualizando el valor del nodo resultante con la suma de los valores del primer y segundo nodo.

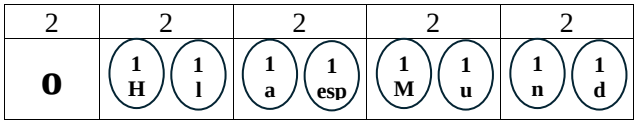
2	1	1	1	1	1	1	2
1 H **1 l**	**a**	**esp**	**M**	**u**	**n**	**d**	**o**

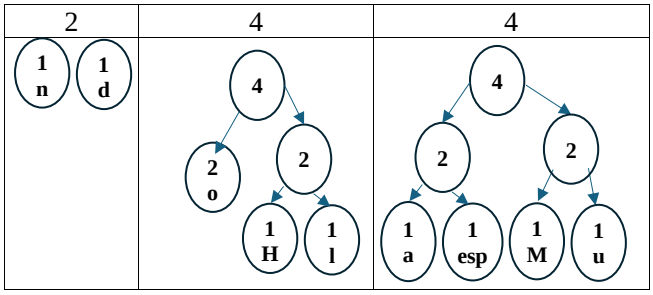
Paso 4: El nuevo nodo se coloca en su posición correspondiente, manteniendo el orden de menor a mayor.

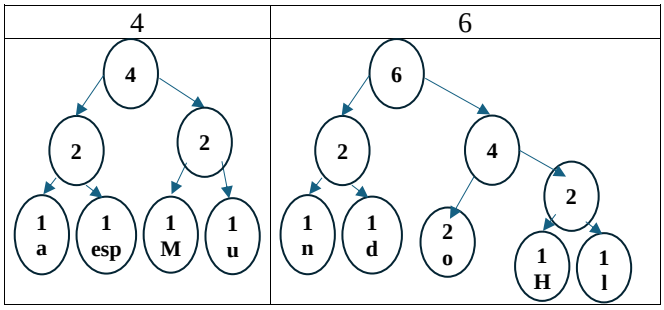
1	1	1	1	1	1	2	2
a	**esp**	**M**	**u**	**n**	**d**	**o**	**1 H** **1 l**

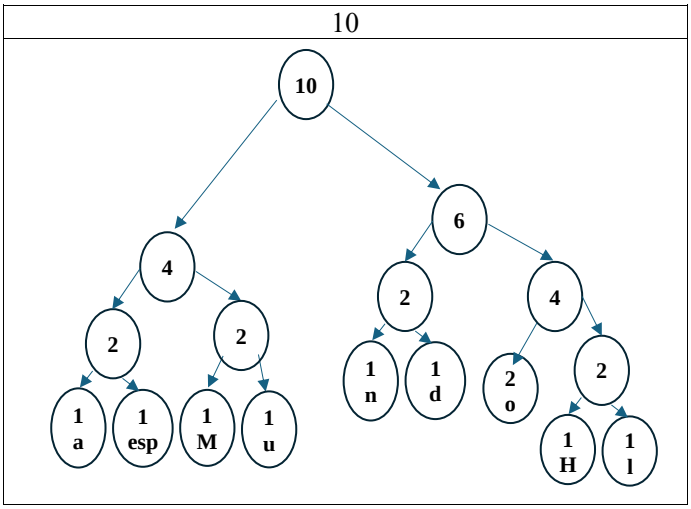
Paso 5: Se repiten los Pasos 3 y 4 de manera continua hasta que quede un único nodo que contenga todos los caracteres.


2
1 a
1 esp
2
1 M
1 u
2
1 n
1 d
2
o
2
1 H
1 l

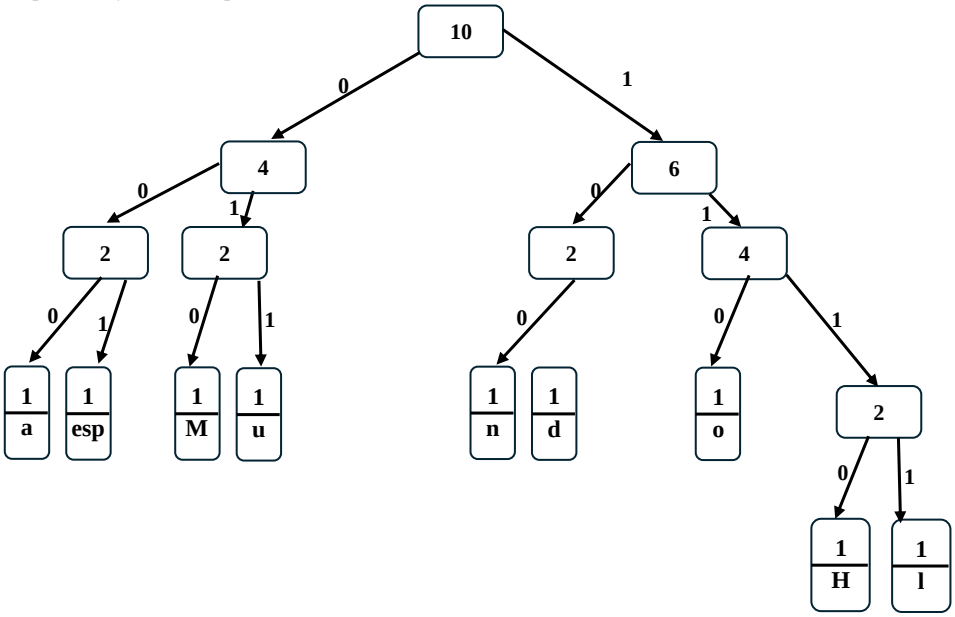

2
o
2
1 H
1 l
2
1 a
1 esp
2
1 M
1 u
2
1 n
1 d


2
1 n
1 d
4
4
2 o
2
1 H
1 l
4
2
2
1 a
1 esp
1 M
1 u


4
4
2
2
1 a
1 esp
1 M
1 u
6
6
2
4
1 n
1 d
2 o
2
1 H
1 l


10
10
6
4
2
4
2
2
2
1 a
1 esp
1 M
1 u
1 n
1 d
2 o
2
1 H
1 l

Paso 6: Se construye el árbol de Huffman asignando 0 a las ramas que van a la izquierda y 1 a las que van a la derecha.



Paso 7: Se obtiene la cadena de codificación huffman siguiendo las ramas del árbol (0 a la izquierda y 1 a la derecha) hasta llegar al carácter deseado.

H	o	l	a	esp	M	u	n	d	o
1110	110	1111	000	001	010	011	100	101	110

Cadena Codificada

4.3. Diagrama de Flujo

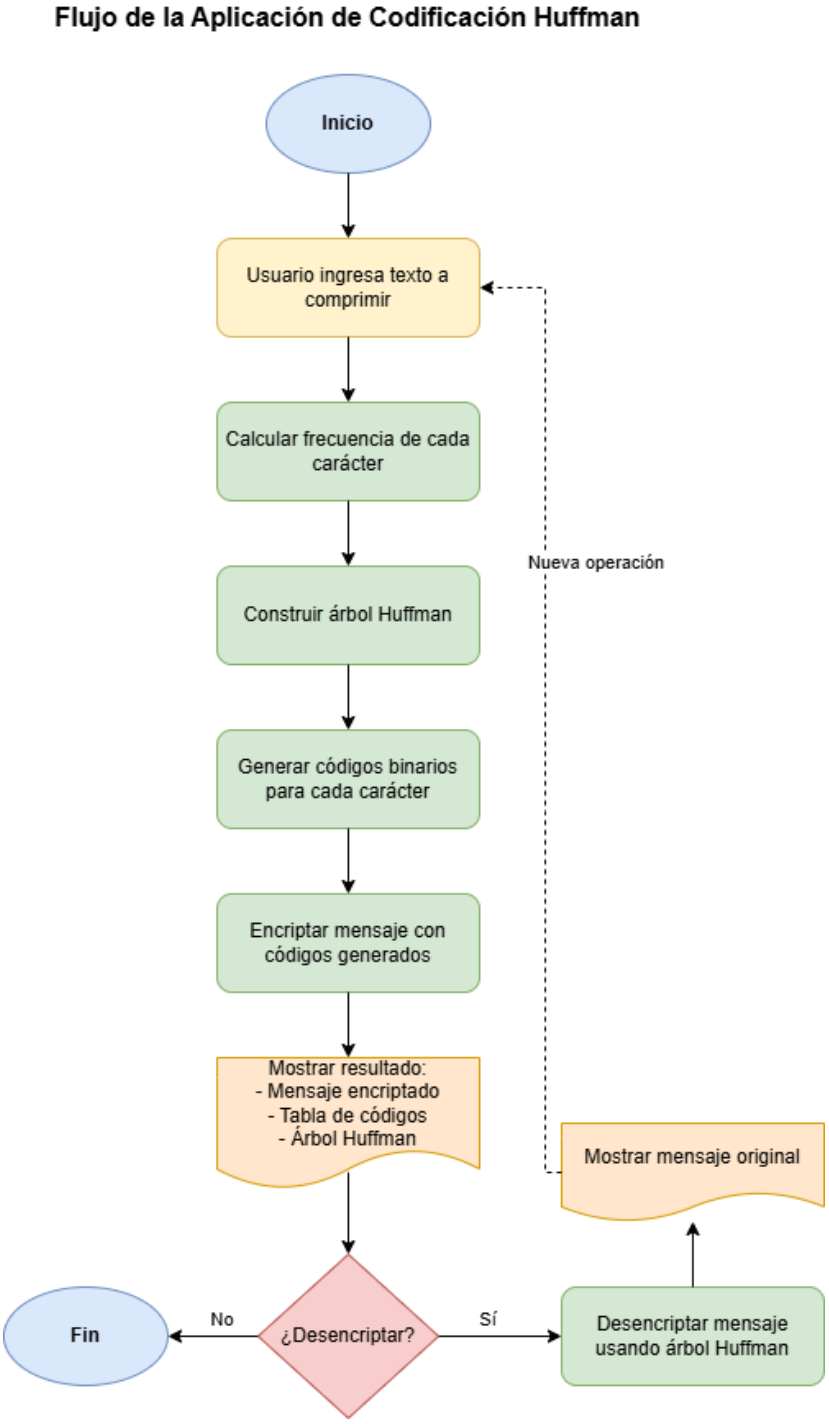


Figura 1: Diagrama del proceso de encriptación

5. Implementación

5.1. Backend (Python)

El backend del sistema fue desarrollado en Python y se encarga de ejecutar el algoritmo de codificación de Huffman. Para ello, se utilizaron estructuras de datos personalizadas, como la clase `Node`, que representa cada nodo del árbol binario:

```
class Node:
    def __init__(self, char=None, freq=None):
        self.char = char  # Caracter (solo en nodos hoja)
        self.freq = freq  # Frecuencia del car cter
        self.left = None  # Hijo izquierdo (representa un 0)
        self.right = None # Hijo derecho (representa un 1)
```

Las funciones principales implementadas en el backend incluyen:

- `calculate_frequencies()`: Calcula la frecuencia de cada carácter en la frase ingresada.
- `build_huffman_tree()`: Construye el árbol binario de Huffman a partir de las frecuencias.
- `generate_codes()`: Genera los códigos binarios correspondientes a cada carácter.
- `huffman_encrypt()`: Coordina todo el proceso de codificación.

Además, se integró la herramienta `Graphviz` para generar una imagen del árbol resultante.

5.2. Frontend (React)

La interfaz web fue desarrollada en React y permite al usuario interactuar fácilmente con el sistema. Entre sus componentes principales se encuentran:

- Un campo de texto para ingresar la frase a codificar.
- Visualización gráfica del árbol de Huffman generado.
- Una tabla que muestra la frecuencia y el código binario de cada carácter.
- Botones funcionales para:
  - Encriptar y desencriptar el mensaje.
  - Exportar el resultado a PDF.
  - Visualizar la imagen del árbol.

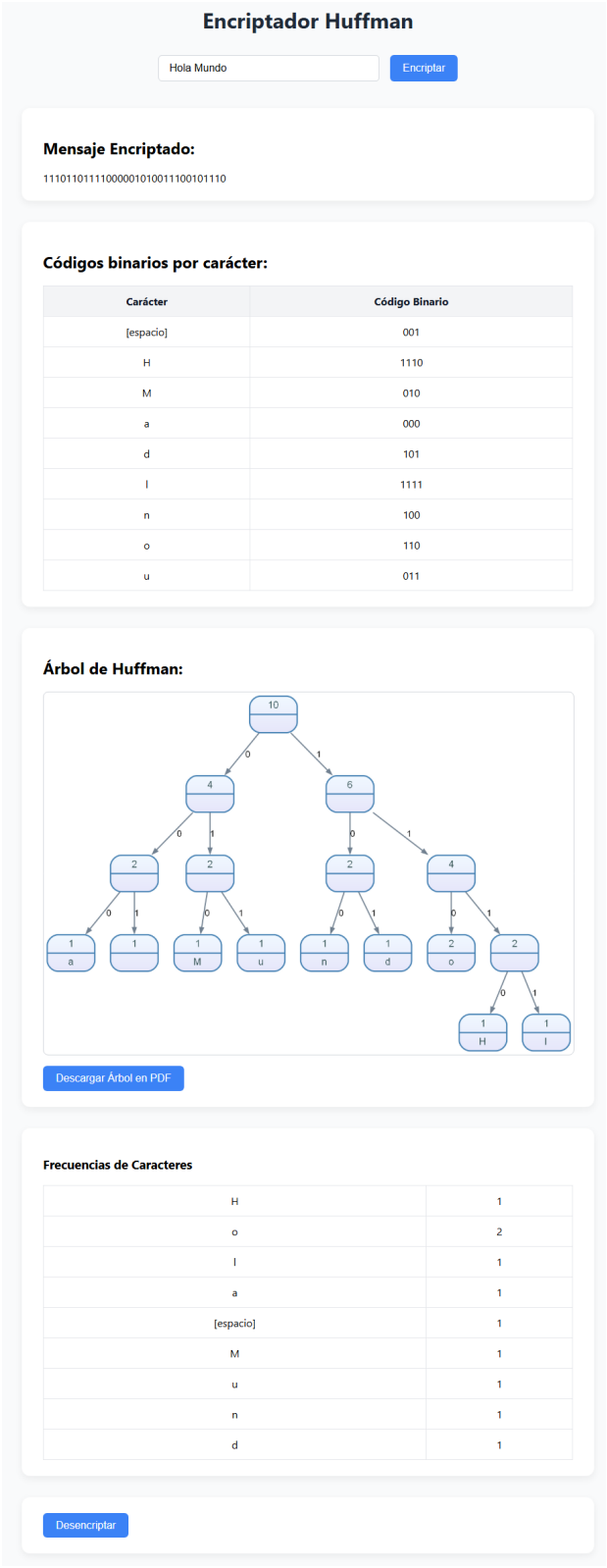


Figura 2: Interfaz principal del sistema

6. Mockups del Software

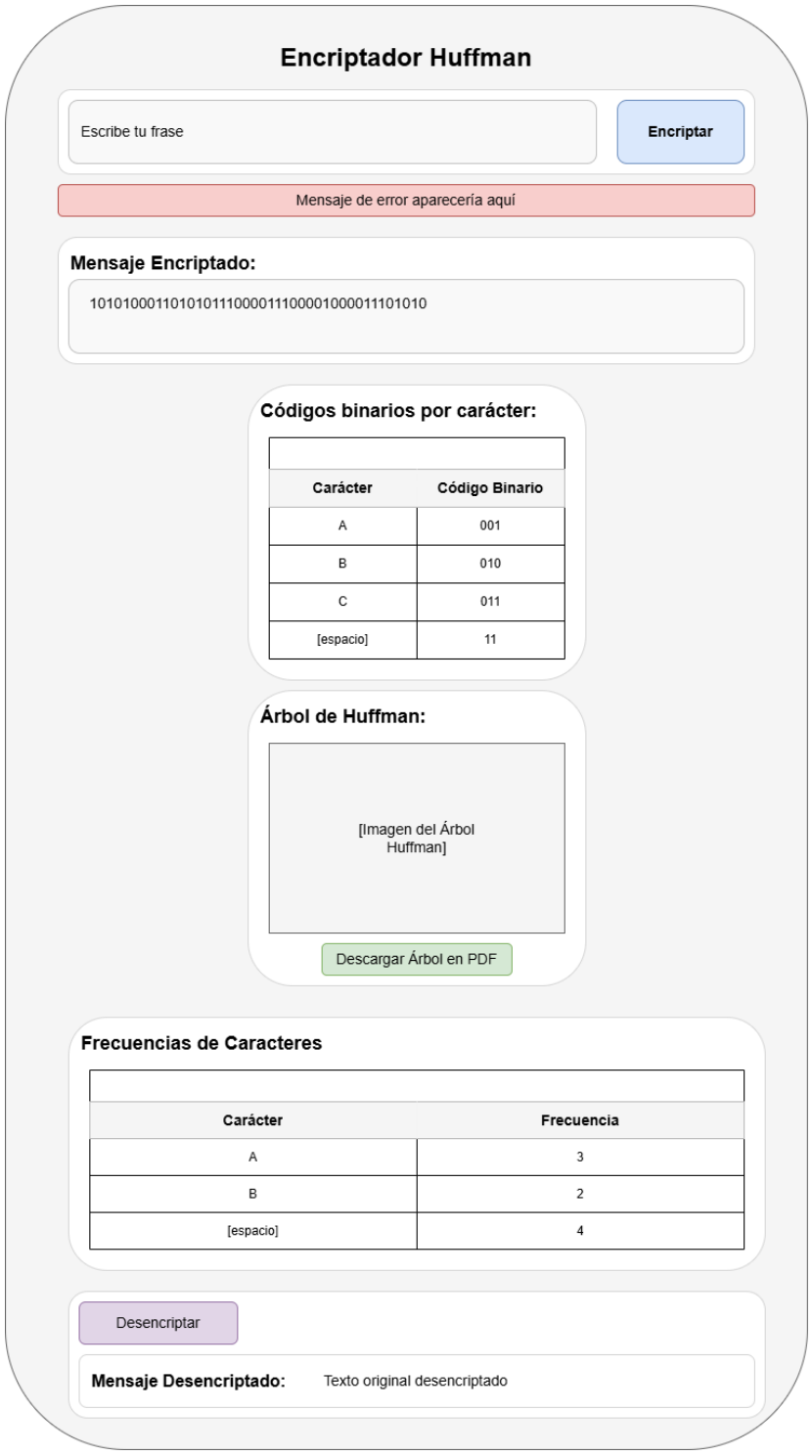


Figura 3: Interfaz principal del sistema

7. Conclusiones

- Se implementó con éxito el algoritmo de Huffman en una arquitectura web moderna (Python + React), demostrando su eficacia para compresión de textos mediante codificación variable.
- La integración de Graphviz permitió visualizar el árbol binario resultante, facilitando la comprensión del proceso de codificación y validando los resultados.
- El sistema desarrollado sirve como base para futuras mejoras, particularmente en optimización para textos largos y extensión a formatos binarios.

8. Referencias

1. Cover, T. M., & Thomas, J. A. (2006). *Elements of information theory* (2nd ed.). Wiley-Interscience.

2. Sayood, K. (2017). *Introduction to data compression* (5th ed.). Morgan Kaufmann.

3. Wikipedia contributors. (s.f.). *Huffman coding*. Wikipedia, The Free Encyclopedia. Recuperado de https://en.wikipedia.org/wiki/Huffman_coding

4. The Python Software Foundation. (s.f.). *The Python Tutorial*. Recuperado de https://docs.python.org/3/tutorial/index.html

5. The Pallets Projects. (s.f.). *Flask Documentation*. Recuperado de https://flask.palletsprojects.com/en/2.3.x/

6. The React Team. (s.f.). *Main Concepts*. React. Recuperado de https://react.dev/learn