***Fundamentos de Análisis y Diseño de Algoritmos***

Trabajo final:

***“Algoritmo de Huffman”***

*Msc. Carlos Andrés Delgado Saavedra*

*Leider Santiago Cortés Hernández* – 202159879

*Miguel Ángel Rueda Colonia* - 202159896

5 (quinto) semestre

Universidad del Valle sede Tuluá

02 de diciembre de 2023

*Tuluá, Valle del cauca*

***Implementación del programa.***

El proyecto se hizo orientado a objetos, en python, utilizamos como estructuras de datos arreglos y arboles binarios, con tres clases: “arbolbinariohuffman”, “CodificacionHuffman” y “DecodificacionHuffman” las clases contienen lo siguiente:

* ***arbolbinariohuffman*:**

Dentro de esta clase, lo que encontraremos será la forma en la que se crea y se imprime el árbol binario.

* ***DecodificacionHuffman*:**

Aquí, lo que se hace es recorrer el árbol y decodificarlo, las palabras, frases o texto que se escribieron al inicio deberían estar reconstruidas al final gracia a esta clase.

* ***CodificacionHuffman*:**

Esta clase tiene los métodos para inicializar las otras dos clases, además de que aquí se importan estas también, dado que esta es la clase principal, después, codifica el texto a binario (en el método encode), agrupándolos según las hojas del árbol, también tiene otro método en donde tiene una tabla que muestra que valor tiene en bits cada carácter, están las pruebas para ejecutar el código, retorna el árbol generado en la otra clase, retorna la tabla, también obtiene el total de nodos, total de caracteres y calcula la profundidad del árbol, tiene otro método para obtener un resumen de los respectivos datos y también están todas las impresiones aquí, cosa que también retorna.

La complejidad teórica de este algoritmo, dado que recorre el arreglo y el texto en promedio una vez completamente y otra a medias, la cota teórica seria O(n log n).

Datos vs tiempo

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| datos | n=100 | N=200 | 500 | 1000 | 1500 |
| Tiempo  (milisegundos) | 3.9887428283691406 | 7.985115051269531 | 6.969213485717773 | 5.515098571777344 | 8.359193801879883 |

100\*log(100)=200

200\*log(200)=460

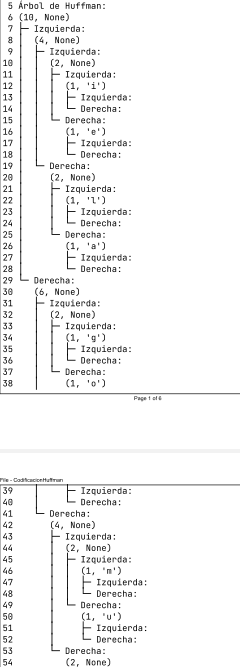
500\*log(500)=1349

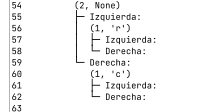
1000\*log(1000)=3000

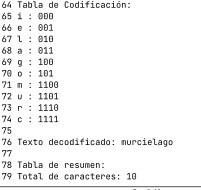
1500\*log(1500)=4764

**Ejemplos y conclusiones del ejercicio:**

**Ejemplo1:** 



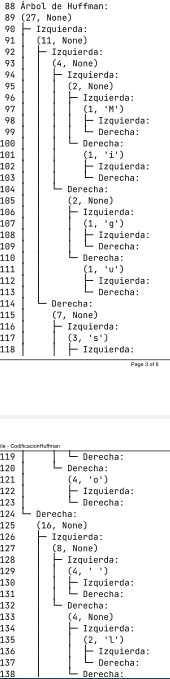


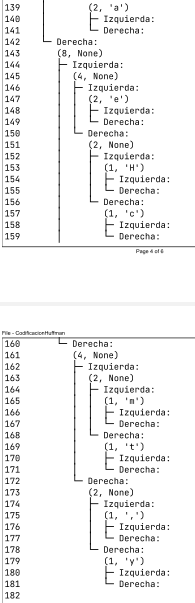


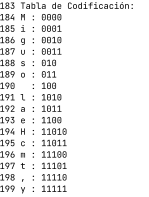


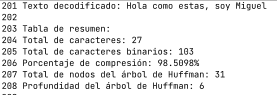
Ejemplo 2:











En cuanto a la comparación de bytes del original entre el generado, podemos apreciar que la compresión se realiza entre un 95 por ciento y un 100 por ciento, obteniendo así una buena comprensión en cuanto a tamaño se refiere.