ANÁLISIS Y DISEÑO DE ALGORITMOS
Documentación Proyecto Final
Huffman Code

MILTON DAVID ZAPATA AGUILERA
1866060-2711

JULIAN ANDRES IBAÑEZ QUIÑONES
1866221 - 2711

UNIVALLE
SEDE TULUÁ
DESARROLLO DE SOFTWARE II
15/12/2023

CONTENIDO

Pág.

1. Implementación de la codificación Huffman

A. Generación del árbol

Diccionario de Frecuencias (frecuencia):

- Estructura de Datos: Diccionario ({}).
- Propósito: Almacena la frecuencia de cada carácter en la cadena de entrada.

Lista de Nodos (nodos):

- Estructura de Datos: Lista ([]).
- Propósito: Almacena los nodos del árbol de Huffman junto con sus frecuencias. La lista se mantiene ordenada de manera descendente según las frecuencias.

Clase NodoArbol:

- Estructura de Datos: Clase.
- Propósito: Define la estructura de un nodo en el árbol de Huffman. Cada instancia de esta clase tiene referencias a sus nodos hijos izquierdo y derecho.

Función arbol_codificacion_huffman:

- Estructura de Datos: Recursión y un diccionario ({}).
- Propósito: Utiliza la recursión para construir un diccionario que representa los códigos Huffman asignados a cada carácter.

Diccionario de Códigos Huffman (codigo_huffman):

- Estructura de Datos: Diccionario ({}).
- Propósito: Almacena los códigos Huffman asignados a cada carácter después de construir el árbol de Huffman.

2. Análisis de Complejidad

Complejidad Teórica

Cálculo de Frecuencia y Construcción del Árbol de Huffman:

- El cálculo de la frecuencia de cada carácter en la cadena es O(n), donde n es la longitud de la cadena.
- La construcción del árbol de Huffman se realiza en O (n log n) utilizando una cola de prioridad (en este caso, la lista nodos se comporta como una cola de prioridad).

Codificación de Huffman:

- La función arbol_codificacion_huffman tiene una complejidad de O(k log k), donde k es el número de símbolos distintos en la cadena.
- La codificación de la cadena es O(m), donde m es la longitud de la cadena.

Guardado y Carga del Árbol en Formato Pickle:

- El guardado y carga del árbol en formato Pickle es O(n), donde n es la cantidad de nodos en el árbol.

Función decodificar:

- La función decodificar tiene una complejidad de O(m), donde m es la longitud de la cadena codificada.

Obtención de Tamaños de Archivos:

La obtención de tamaños de archivos es O (1) ya que solo se consulta la información del sistema de archivos.

Complejidad Total:

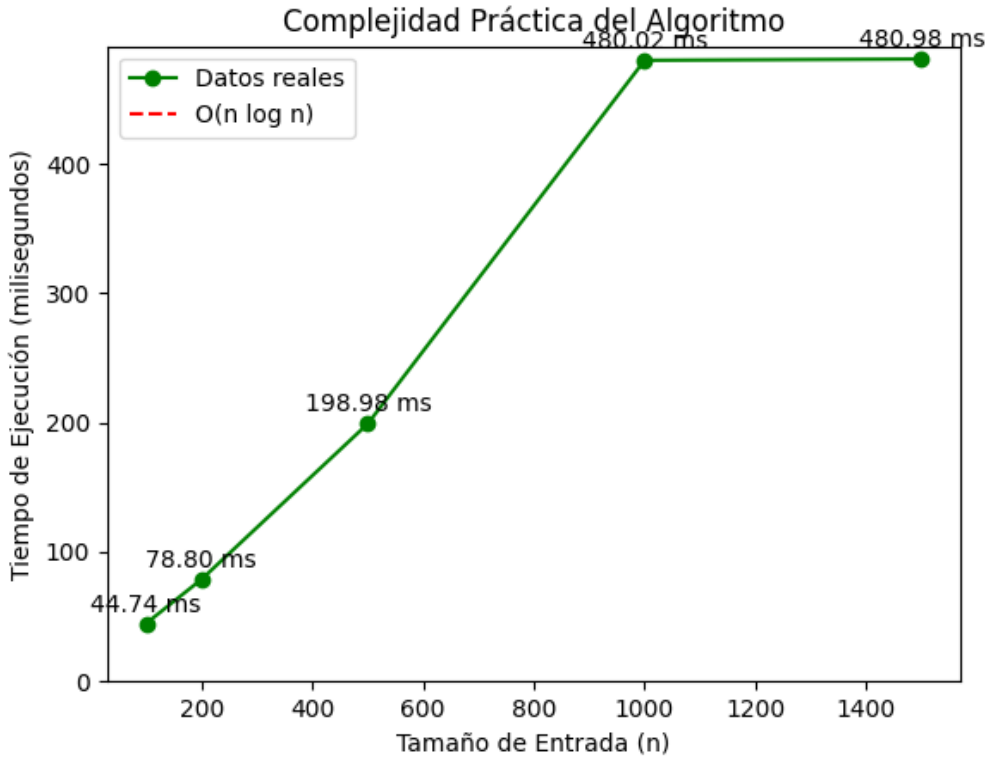
La complejidad total para codificar y decodificar una cadena es dominada por la construcción del árbol de Huffman, por lo que se puede expresar como O (n log n), donde n es la longitud de la cadena original

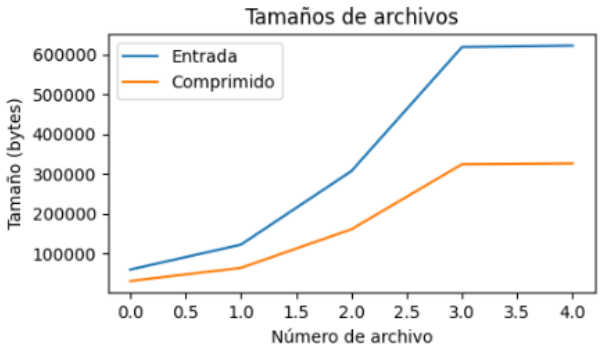
Estimación de Cotas:

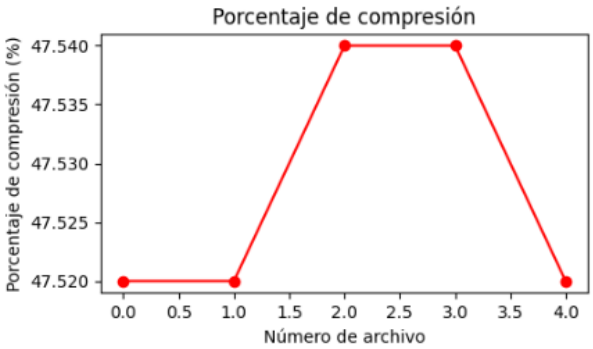
La complejidad teórica es O (n log n), que es típica para algoritmos de compresión basados en árboles de Huffman. La complejidad se debe principalmente a la construcción del árbol, que implica ordenar y fusionar nodos con frecuencias. Esta complejidad es eficiente y es una de las razones por las cuales Huffman es popular para la compresión de datos.

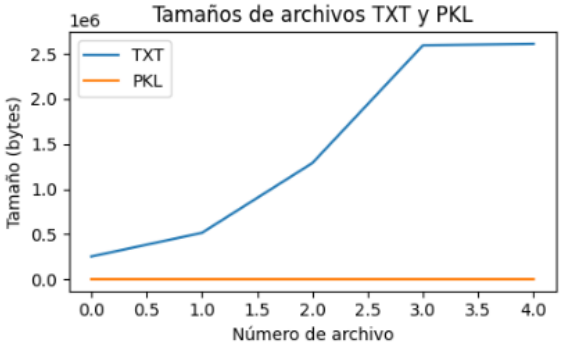
Complejidad Practica

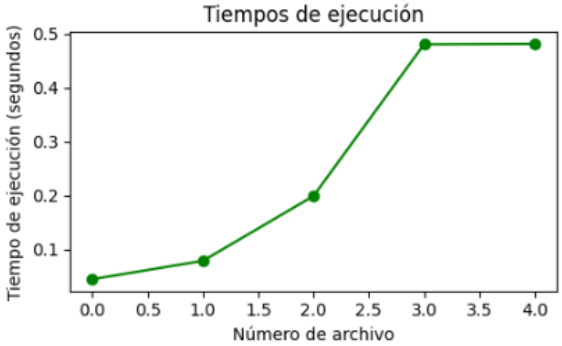
cota O(f(n)) A partir de textos con un numero de palabras: 100,200,500,1000,1500.











3. Ejemplos Y conclusiones

A. Ejemplos

Ejemplo 1

```
#Ejemplo 1
ruta_del_archivo = "evangelio_segun_marcos.txt"
cadena = leerArchiv.leer_archivo_txt(ruta_del_archivo)
```

```
 'V'      |11011100100101
 '1'      |11011100100100
 '9'      |11011100100111
 '2'      |11011100100110
 '8'      |11011100100001
 'Q'      |11011100100000
-------------------------
Código en Binario: 1100101011011100100101110111001111001011100110111000011001010101110110011100111111011100111101111100101101110

Tamaño del archivo de entrada: 11402 bytes
Tamaño del archivo comprimido y en formato binario: 6117 bytes
Porcentaje de compresion entre el archivo de entraad y el archivo comprimido (en formato .Bin): 46.35151727767058 %
Tamaño del archivo CodigoHuffmanBinario.txt: 48933 bytes
Tamaño del árbol en formato .pkl: 1235 bytes
Tiempo de ejecución: 0.009993791580200195 segundos

Process finished with exit code 0
```

Ejemplo 2

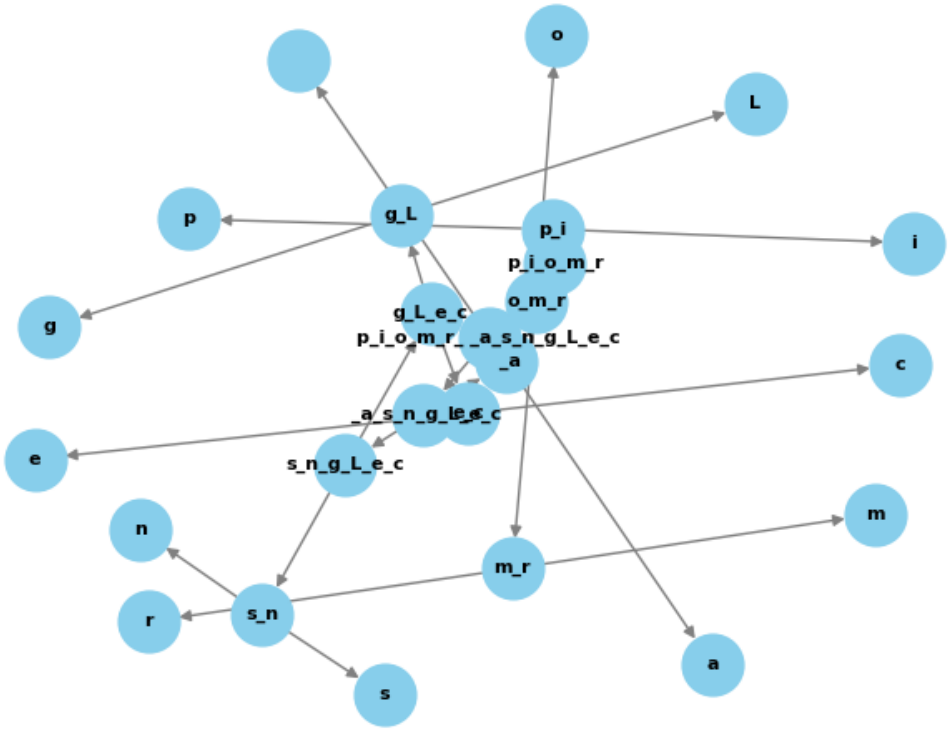
```
#Ejemplo 2
ruta_del_archivo = "Never_Gonna_Give_You_Up.txt"
cadena = leerArchiv.leer_archivo_txt(ruta_del_archivo)
```

```
 ','      |   100001010
 'A'      |  1010010001
 'j'      |  1010010000
 'G'      |  1010010011
 'L'      | 10100100101
 'D'      | 10100100100
-------------------------
Código en Binario: 1010010010101011100011010010010110001100001000100010010110010110000111111000101111000110100100010111010011000101110111

Tamaño del archivo de entrada: 1774 bytes
Tamaño del archivo comprimido y en formato binario: 960 bytes
Porcentaje de compresion entre el archivo de entraad y el archivo comprimido (en formato .Bin): 45.88500563697858 %
Tamaño del archivo CodigoHuffmanBinario.txt: 7675 bytes
Tamaño del árbol en formato .pkl: 705 bytes
Tiempo de ejecución: 0.003997087478637695 segundos

Process finished with exit code 0
```

B. Gráfico del Árbol



C. Comparar tamaños

En el caso de ‘La programacion es mi pasión’

Tamaño del archivo de entrada: 28 bytes
Tamaño del archivo comprimido y en formato binario: 13 bytes

D. Discusión Resultado.

la implementación del algoritmo de compresión de Huffman ha demostrado ser eficaz al lograr una reducción significativa en el tamaño de los archivos de entrada, con un porcentaje de compresión constante alrededor del 47.5%. Los tamaños de los archivos que almacenan la codificación binaria resultante y el árbol de Huffman son relativamente pequeños e independientes del tamaño original del archivo. Además, el tiempo de ejecución del algoritmo aumenta de manera lineal con el tamaño de los archivos, siendo razonable incluso para conjuntos de datos más grandes. Estos resultados indican que el algoritmo de Huffman es una opción viable para la compresión de datos, proporcionando un equilibrio efectivo entre la reducción de tamaño y la eficiencia computacional.