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

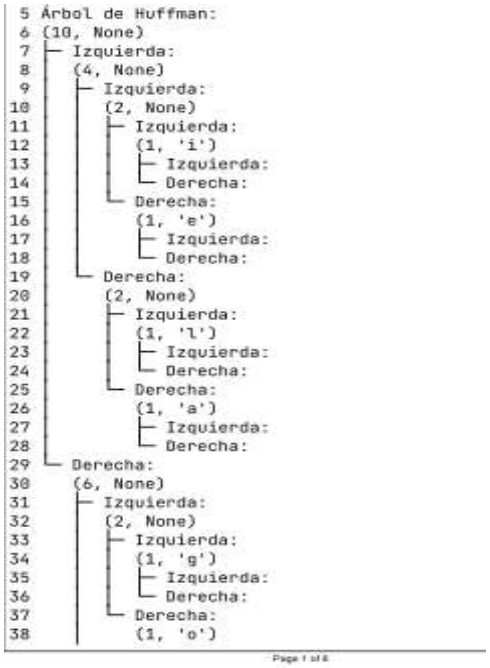
1000*log(1000)=3000

1500*log(1500)=4764

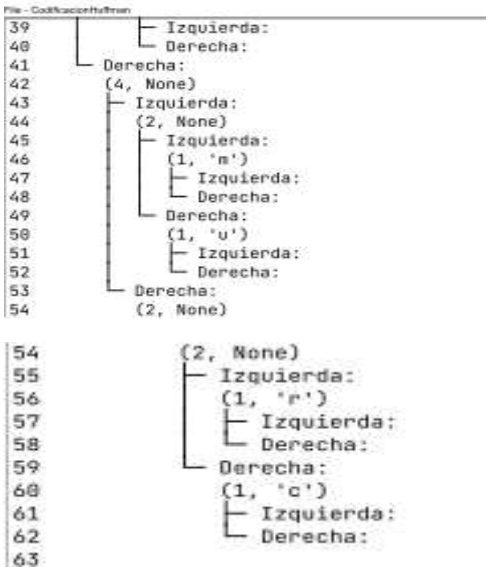
Ejemplos y conclusiones del ejercicio:

Ejemplo1:

```
2 Texto original: murcielago
3 Texto codificado: 1100110111101111000010100111001010
```



```
 5 Árbol de Huffman:
 6 (10, None)
 7 ├─ Izquierda:
 8 │  (4, None)
 9 │  ├─ Izquierda:
10 │  │  (2, None)
11 │  │  ├─ Izquierda:
12 │  │  │  (1, 'i')
13 │  │  │  ├─ Izquierda:
14 │  │  │  └─ Derecha:
15 │  │  └─ Derecha:
16 │  │     (1, 'e')
17 │  │     ├─ Izquierda:
18 │  │     └─ Derecha:
19 │  └─ Derecha:
20 │     (2, None)
21 │     ├─ Izquierda:
22 │     │  (1, 'l')
23 │     │  ├─ Izquierda:
24 │     │  └─ Derecha:
25 │     └─ Derecha:
26 │        (1, 'a')
27 │        ├─ Izquierda:
28 │        └─ Derecha:
29 └─ Derecha:
30    (6, None)
31    ├─ Izquierda:
32    │  (2, None)
33    │  ├─ Izquierda:
34    │  │  (1, 'g')
35    │  │  ├─ Izquierda:
36    │  │  └─ Derecha:
37    │  └─ Derecha:
38    │     (1, 'o')
```



```
39    │        ├─ Izquierda:
40    │        └─ Derecha:
41    └─ Derecha:
42       (4, None)
43       ├─ Izquierda:
44       │  (2, None)
45       │  ├─ Izquierda:
46       │  │  (1, 'm')
47       │  │  ├─ Izquierda:
48       │  │  └─ Derecha:
49       │  └─ Derecha:
50       │     (1, 'u')
51       │     ├─ Izquierda:
52       │     └─ Derecha:
53       └─ Derecha:
54          (2, None)
```

```
54             (2, None)
55             ├─ Izquierda:
56             │  (1, 'r')
57             │  ├─ Izquierda:
58             │  └─ Derecha:
59             └─ Derecha:
60                (1, 'c')
61                ├─ Izquierda:
62                └─ Derecha:
63
```

```
Tabla de Codificación:
i : 000
e : 001
l : 010
a : 011
g : 100
o : 101
m : 1100
u : 1101
r : 1110
c : 1111

Texto decodificado: murcielago

Tabla de resumen:
Total de caracteres: 10
Total de caracteres binarios: 34
Porcentaje de compresión: 98.6719%
Total de nodos del árbol de Huffman: 19
Profundidad del árbol de Huffman: 5
```

Ejemplo 2:

```
Texto original: Hola como estas, soy Miguel
Texto codificado:
110100111010101110011011011111000111001100010111011 0
110101111010001001111111100000000010010001111001010
```

```
Árbol de Huffman:
(27, None)
├─ Izquierda:
│  (11, None)
│  ├─ Izquierda:
│  │  (4, None)
│  │  ├─ Izquierda:
│  │  │  (2, None)
│  │  │  ├─ Izquierda:
│  │  │  │  (1, 'M')
│  │  │  │  ├─ Izquierda:
│  │  │  │  └─ Derecha:
│  │  │  └─ Derecha:
│  │  │     (1, 'i')
│  │  │     ├─ Izquierda:
│  │  │     └─ Derecha:
│  │  └─ Derecha:
│  │     (2, None)
│  │     ├─ Izquierda:
│  │     │  (1, 'g')
│  │     │  ├─ Izquierda:
│  │     │  └─ Derecha:
│  │     └─ Derecha:
│  │        (1, 'u')
│  │        ├─ Izquierda:
│  │        └─ Derecha:
│  └─ Derecha:
│     (7, None)
│     ├─ Izquierda:
│     │  (3, 's')
│     │  ├─ Izquierda:
```

```
│     │  └─ Derecha:
│     └─ Derecha:
│        (4, 'o')
│        ├─ Izquierda:
│        └─ Derecha:
└─ Derecha:
   (16, None)
   ├─ Izquierda:
   │  (8, None)
   │  ├─ Izquierda:
   │  │  (4, ' ')
   │  │  ├─ Izquierda:
   │  │  └─ Derecha:
   │  └─ Derecha:
   │     (4, None)
   │     ├─ Izquierda:
   │     │  (2, 'l')
   │     │  ├─ Izquierda:
   │     │  └─ Derecha:
   │     └─ Derecha:
```

```
      │        (2, 'a')
      │        ├─ Izquierda:
      │        └─ Derecha:
      └─ Derecha:
         (8, None)
         ├─ Izquierda:
         │  (4, None)
         │  ├─ Izquierda:
         │  │  (2, 'e')
         │  │  ├─ Izquierda:
         │  │  └─ Derecha:
         │  └─ Derecha:
         │     (2, None)
         │     ├─ Izquierda:
         │     │  (1, 'H')
         │     │  ├─ Izquierda:
         │     │  └─ Derecha:
         │     └─ Derecha:
         │        (1, 'o')
         │        ├─ Izquierda:
         │        └─ Derecha:
```

```
         └─ Derecha:
            (4, None)
            ├─ Izquierda:
            │  (2, None)
            │  ├─ Izquierda:
            │  │  (1, 'a')
            │  │  ├─ Izquierda:
            │  │  └─ Derecha:
            │  └─ Derecha:
            │     (1, 't')
            │     ├─ Izquierda:
            │     └─ Derecha:
            └─ Derecha:
               (2, None)
               ├─ Izquierda:
               │  (1, ',')
               │  ├─ Izquierda:
               │  └─ Derecha:
               └─ Derecha:
                  (1, 'y')
                  ├─ Izquierda:
                  └─ Derecha:
```

```
183 Tabla de Codificación:
184 M : 0000
185 i : 0001
186 g : 0010
187 u : 0011
188 s : 010
189 o : 011
190   : 100
191 l : 1010
192 a : 1011
193 e : 1100
194 H : 11010
195 c : 11011
196 m : 11100
197 t : 11101
198 , : 11110
199 y : 11111
```

```
201 Texto decodificado: Hola como estas, soy Miguel
202
203 Tabla de resumen:
204 Total de caracteres: 27
205 Total de caracteres binarios: 103
206 Porcentaje de compresión: 98.5098%
207 Total de nodos del árbol de Huffman: 31
208 Profundidad del árbol de Huffman: 6
```

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.