



# Instituto Politécnico Nacional

## Escuela Superior de Cómputo



Unidad de académica:

Análisis y diseño de algoritmos

Alumno:

Solares Velasco Arturo Misael      2023630538

Solis Lugo Mayra      2023630449

Grupo: 3CV1

Profesor : Floriano García Andres

El algoritmo de Huffman es un método de compresión de datos sin pérdida que asigna códigos de longitud variable a los caracteres en un texto, de manera que los caracteres más frecuentes tengan códigos más cortos y los menos frecuentes tengan códigos más largos. Esto resulta en una representación más eficiente del texto, lo que permite reducir el tamaño del archivo sin perder información.

## 2. Descripción del Código:

El código proporcionado implementa el algoritmo de Huffman en Python, utilizando programación orientada a objetos y las librerías estándar `heapq` y `os`. A continuación, se describen las principales clases y funciones:

### Clase NodoHuffman:

Esta clase representa un nodo en el árbol de Huffman. Cada nodo tiene un carácter, una frecuencia y referencias a sus nodos hijos izquierdo y derecho. Además, implementa el método `\_\_lt\_\_` para comparar nodos por frecuencia.

```
2 # Clase NodoHuffman: Representa un nodo en el árbol de Huffman.
3 class NodoHuffman:
4     # Atributos del nodo Huffman
5     def __init__(self, caracter, frecuencia):
6         self.caracter = caracter
7         self.frecuencia = frecuencia
8         self.izquierda = None
9         self.derecha = None
10
11     # Método __lt__: Compara dos nodos Huffman por frecuencia.
12     def __lt__(self, otro):
13         return self.frecuencia < otro.frecuencia
```

### Clase CodificadorHuffman:

Esta clase contiene métodos para codificar y decodificar texto utilizando el algoritmo de Huffman. Incluye funciones para generar frecuencias de caracteres, construir el árbol de Huffman, generar códigos de Huffman y realizar la codificación y decodificación del texto.

```
# Clase CodificadorHuffman. Codifica y decodifica texto utilizando el algoritmo de Huffman.
class CodificadorHuffman:
    # Genera un diccionario de frecuencias de caracteres en un texto.
    def generar_frecuencias(self, texto):
        frecuencias = {}
        for caracter in texto:
            if caracter in frecuencias:
                frecuencias[caracter] += 1
            else:
                frecuencias[caracter] = 1
        return frecuencias
```

### -Función generar\_frecuencias:

Este método de la clase `CodificadorHuffman` recorre el texto y genera un diccionario de frecuencias de caracteres.

```
# Genera un diccionario de frecuencias de caracteres en un texto.
def generar_frecuencias(self, texto):
    frecuencias = {}
    for caracter in texto:
        if caracter in frecuencias:
            frecuencias[caracter] += 1
        else:
            frecuencias[caracter] = 1
    return frecuencias
```

# Codifica un texto utilizando un diccionario de frecuencias.

### Función generar\_arbol\_huffman:

Este método de la clase `CodificadorHuffman` construye el árbol de Huffman a partir del diccionario de frecuencias utilizando una cola de prioridad implementada con `heapq`.

```
# Genera un árbol de Huffman a partir del diccionario de frecuencias.
def generar_arbol_huffman(self, frecuencias):
    cola_prioridad = [NodoHuffman(caracter, frecuencia) for caracter, frecuencia in frecuencias.items()]
    heapq.heapify(cola_prioridad)
    while len(cola_prioridad) > 1:
        nodo_izq = heapq.heappop(cola_prioridad)
        nodo_der = heapq.heappop(cola_prioridad)
        nodo_combinado = NodoHuffman(None, nodo_izq.frecuencia + nodo_der.frecuencia)
        nodo_combinado.izquierda = nodo_izq
        nodo_combinado.derecha = nodo_der
        heapq.heappush(cola_prioridad, nodo_combinado)
    return cola_prioridad[0]
```

### Función generar\_codigos\_huffman:

Este método de la clase `CodificadorHuffman` genera un diccionario de códigos de Huffman a partir del árbol de Huffman utilizando un enfoque de recorrido en profundidad.

```
# Genera un diccionario de códigos de Huffman a partir del árbol de Huffman.
def generar_codigos_huffman(self, nodo, prefijo="", codigos={}):
    if nodo is not None:
        if nodo.caracter is not None:
            codigos[nodo.caracter] = prefijo
        self.generar_codigos_huffman(nodo.izquierda, prefijo + "0", codigos)
        self.generar_codigos_huffman(nodo.derecha, prefijo + "1", codigos)
    return codigos
```

# Codifica un texto utilizando un diccionario de códigos de Huffman.

### Función codificar\_texto:

Este método de la clase `CodificadorHuffman` codifica el texto utilizando el diccionario de códigos de Huffman generado anteriormente.

```
# Codifica un texto utilizando un diccionario de códigos de Huffman.
def codificar_texto(self, texto, codigos):
    texto_codificado = ""
    for caracter in texto:
        texto_codificado += codigos[caracter]
    return texto_codificado
```

### Función decodificar\_texto:

Este método de la clase `CodificadorHuffman` decodifica el texto codificado utilizando el árbol de Huffman.

```
def decodificar_texto(self, texto_codificado, arbol_huffman):
    texto_decodificado = ""
    nodo_actual = arbol_huffman
    for bit in texto_codificado:
        if bit == '0':
            nodo_actual = nodo_actual.izquierda
        else:
            nodo_actual = nodo_actual.derecha
        if nodo_actual.caracter is not None:
            texto_decodificado += nodo_actual.caracter
            nodo_actual = arbol_huffman
    return texto_decodificado
```

### Función main:

La función principal del programa solicita al usuario introducir un texto, codifica el texto utilizando el algoritmo de Huffman, escribe el texto original y el texto codificado en archivos separados, lee el texto codificado desde el archivo, decodifica el texto y muestra el resultado en la consola.

```
2 def main():
3     codificador = CodificadorHuffman()
4
5     # Parte 1: Codificación de Huffman
6     texto = input("Introduce el texto a codificar: ")
7     frecuencias = codificador.generar_frecuencias(texto)
8     arbol_huffman = codificador.generar_arbol_huffman(frecuencias)
9     codigos = codificador.generar_codigos_huffman(arbol_huffman)
10    texto_codificado = codificador.codificar_texto(texto, codigos)
11
12    with open("./practica_lab_7/texto_original.txt", "w") as archivo_original:
13        archivo_original.write(texto)
14
```

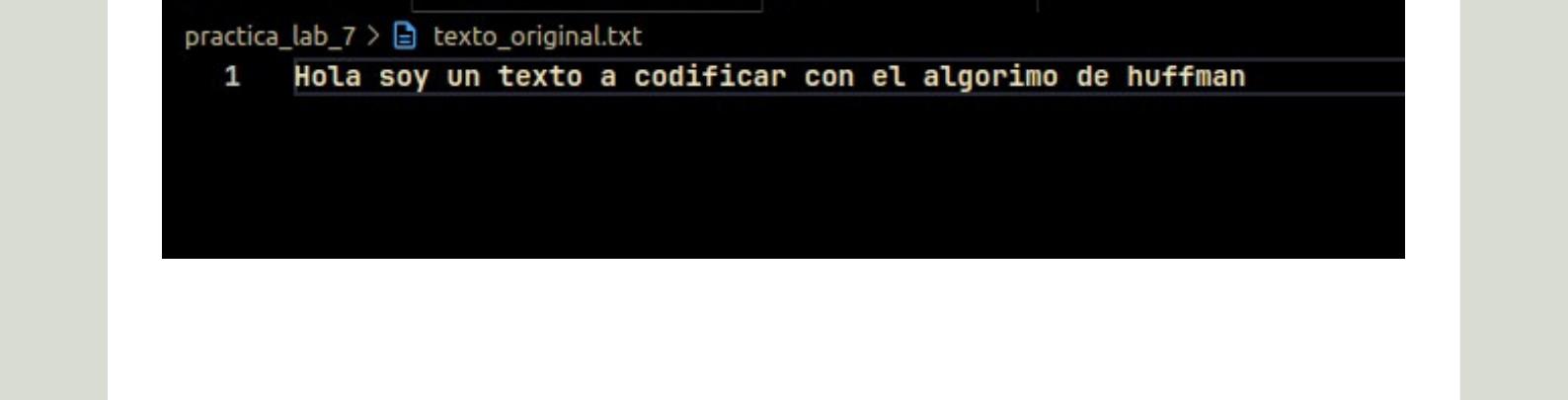
### . Ejecución y Resultados:

Al ejecutar el programa, se solicita al usuario introducir un texto. Luego, el texto se codifica utilizando el algoritmo de Huffman y se escriben el texto original y el texto codificado en archivos separados. A continuación, se lee el texto codificado desde el archivo, se decodifica y se muestra el resultado en la consola

```
scom_2024/practica_lab_7/huffman.py
Introduce el texto a codificar: Hola soy un texto a codificar con el algoritmo de huffman
Texto decodificado: Hola soy un texto a codificar con el algoritmo de huffman
```



```
huffman.py texto_original.txt README.md
practica_lab_7 > texto_original.txt
1 Hola soy un texto a codificar con el algoritmo de huffman
```



```
huffman.py texto_codificado.txt README.md
practica_lab_7 > texto_codificado.txt
1 101001100011100011101000100110010111101001101110110010111001110100100110000010001100100000
10101111010010001101101010111110000111101111001010010101111001111100010111110110110011001110000110
```

El resultado final es la demostración de que el texto original y el texto decodificado son idénticos, lo que confirma que el algoritmo de Huffman ha funcionado correctamente para comprimir y descomprimir el texto.

Conclusiones:

La implementación del algoritmo de Huffman en Python proporciona una manera eficiente de comprimir y descomprimir texto utilizando códigos de longitud variable. Este enfoque demuestra cómo la utilización de estructuras de datos como los árboles y las colas de prioridad puede llevar a soluciones elegantes para problemas de compresión de datos.

