Universidad de Guadalajara

CENTRO UNIVERSITARIO DE CIENCIAS EXACTAS E INGENIERÍAS

análisis de algoritmos | ICOM

Guia de usuario:
ALGORITMO DE COMPRESIÓN DE HUFFMAN .

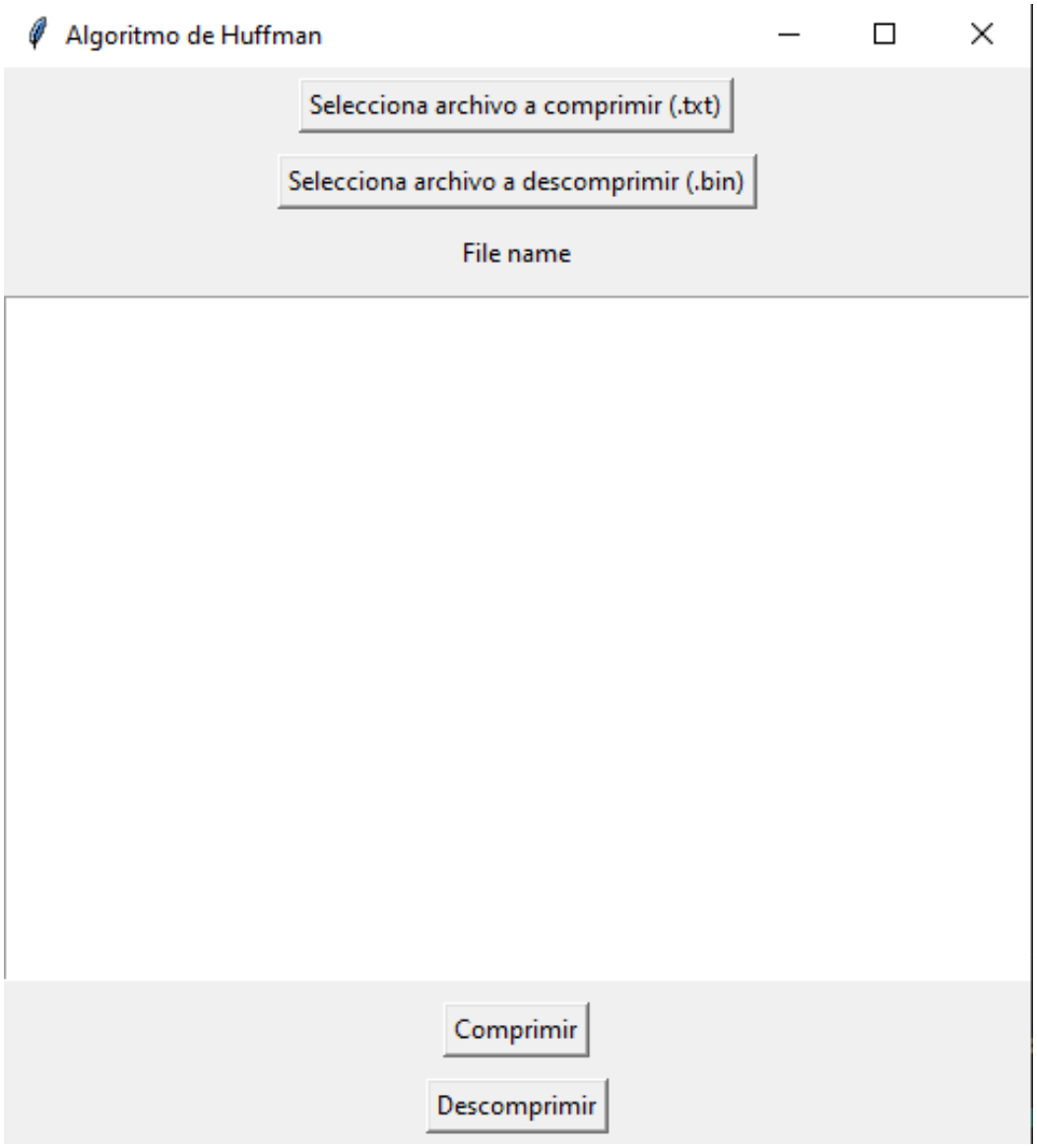
Integrantes del equipo: Flores Garcia Bryan Miguel-220932104
Martinez Rojo Arturo Gael-220589701
Morales Pedroza Arturo- 220574062
Ortiz Guizar Andres-220761695
Rodriguez Arellano Angel Ariel-220934484

Clave del Curso: IL355
NRC: 204835
Calendario: 2025B
Sección: D01
Profesor: JORGE ERNESTO LOPEZ ARCE DELGADO
Calendario: 2025B

GUIA DE USO

A continuación se describe como hacer uso correcto de la aplicación desarrollada usando el algoritmo de Huffman.

Al iniciar la aplicación se nos desplegará una ventana en la cual podremos interactuar y poner en práctica el algoritmo de Huffman.

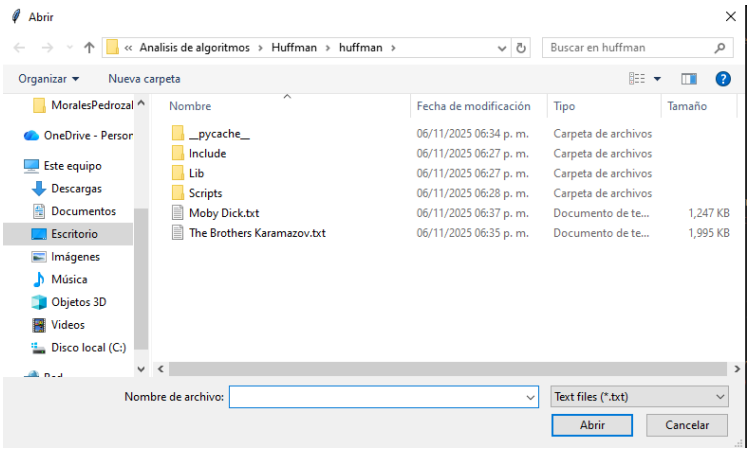


Dentro de este menú podremos encontrar tres secciones:

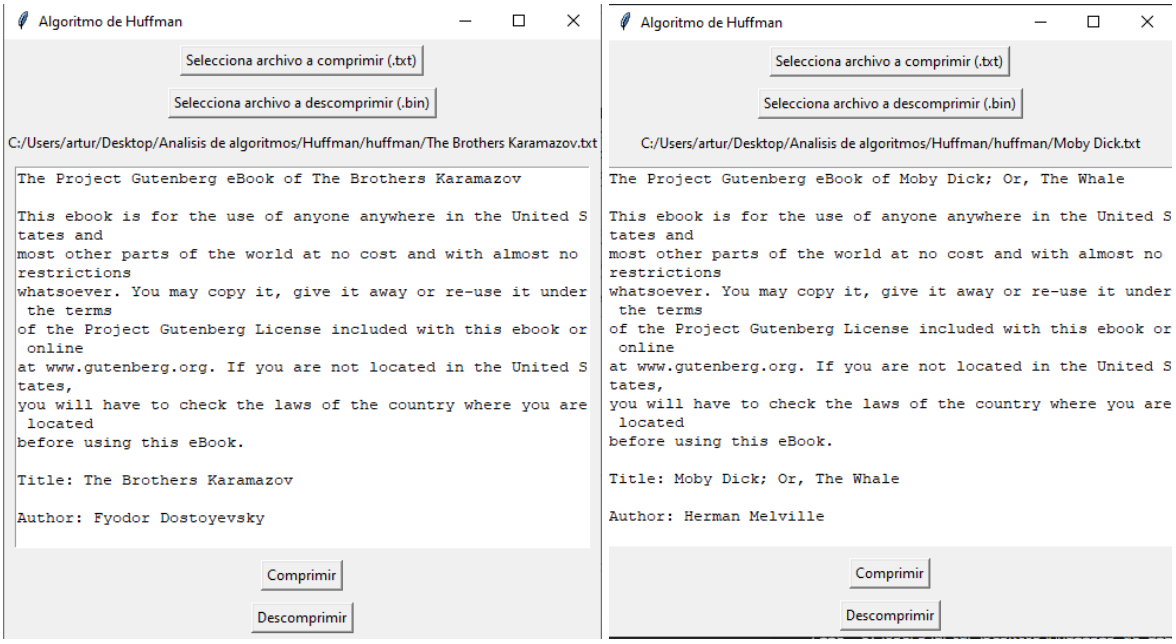
Azul: Botones para poder seleccionar los archivos a modificar
Morado: Previsualización del archivo seleccionado.
Rojo: Botones de acción (comprimir y descomprimir)

Los botones de la sección con el color azul nos abrirá nuestro administrador de archivos y podremos seleccionar un archivo con extensión .txt o .bin dependiendo de la acción que queramos realizar.

Ventana que se visualizará si se presiona cualquiera de esos botones.

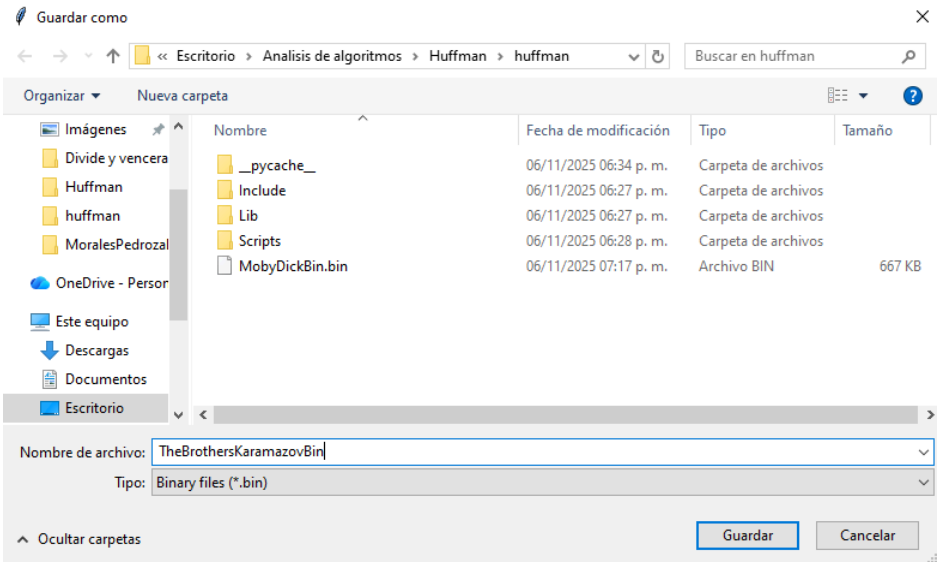


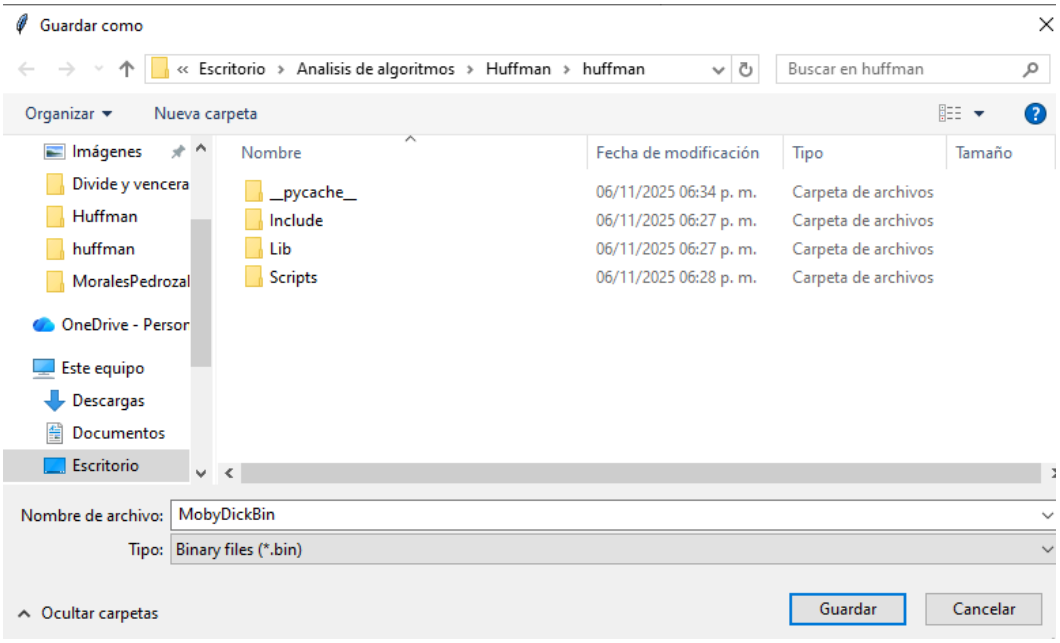
Para nuestros ejemplos usaremos los libros de The Brothers Karamazov y Moby Dick, primero tendremos que comprimirlos y después descomprimirlos, entonces primero los seleccionaremos en nuestra ventana del administrador de archivos que aparece cuando presionamos el botón para comprimir.



Una vez seleccionados los archivos se verá la previsualización en el cuadro de texto.

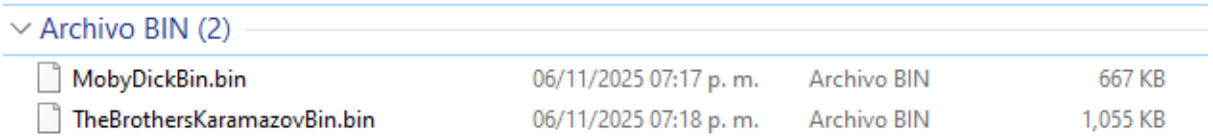
Después podremos comprimir nuestros archivos presionando el botón comprimir en la parte inferior.
Ya que el programa realice el algoritmo de forma exitosa se abrirá nuevamente el administrador de archivos para poder guardar el nuevo archivo generado, esta vez con extensión .bin.





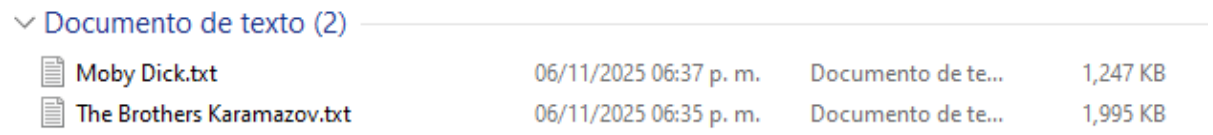
Y podremos ver los archivos nuevos dentro del espacio asignado por el usuario.

(Archivo comprimido)

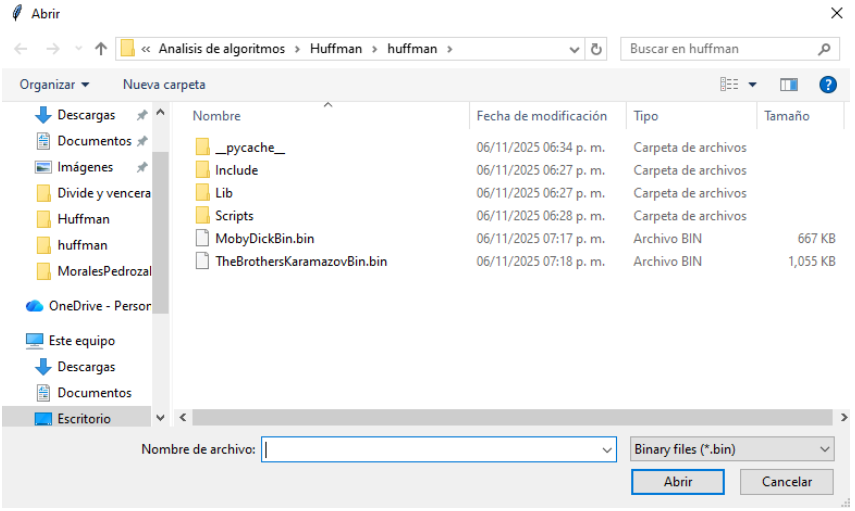


Como podemos observar el algoritmo de Huffman está siendo implementado de manera correcta, reduciendo casi a la mitad la cantidad de bits necesarios para almacenar el libro.

(Archivo original)



Ahora para convertir de .bin a .txt volvemos al inicio pero ahora seleccionamos el botón que dice “descomprimir” y seleccionamos nuestro archivo .bin generado hace unos momentos.

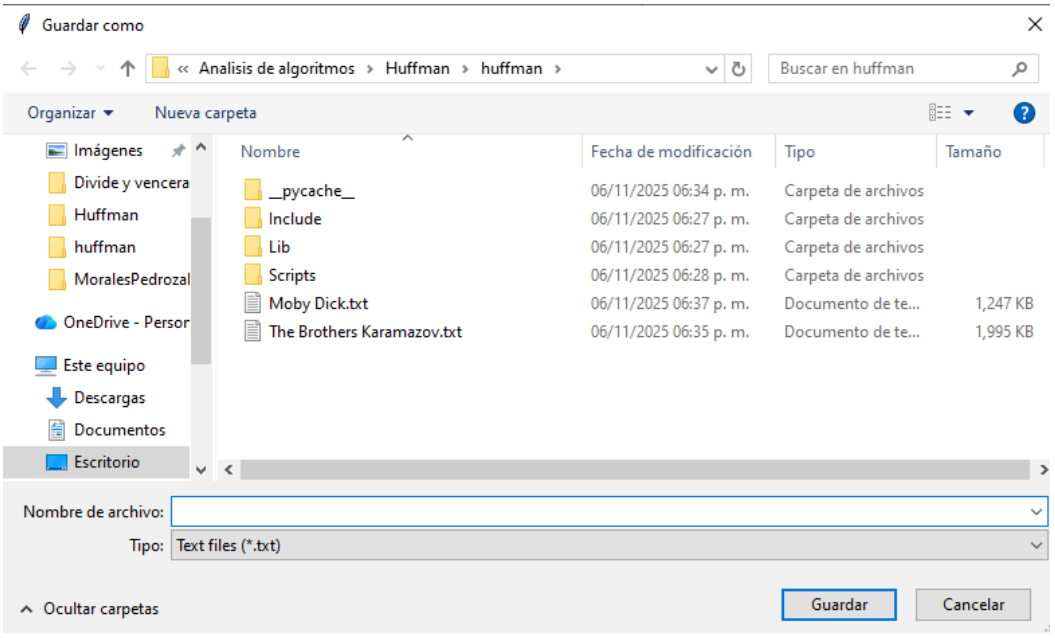


Una vez seleccionado podremos ver el archivo en la etiqueta entre los botones de la parte superior y el cuadro de texto con la previsualización.

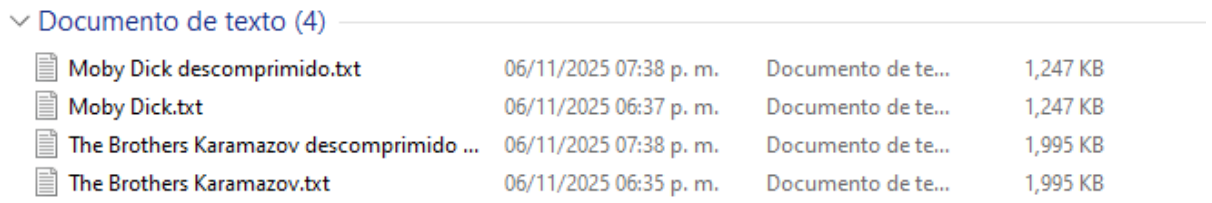
```
C:/Users/artur/Desktop/Analisis de algoritmos/Huffman/huffman/MobyDickBin.bin
```

Y clickeamos en descomprimir en la parte inferior.

Después volverá a aparecer el administrador de archivos pero ahora para guardar el nuevo archivo .txt que se generó de la descompresión.

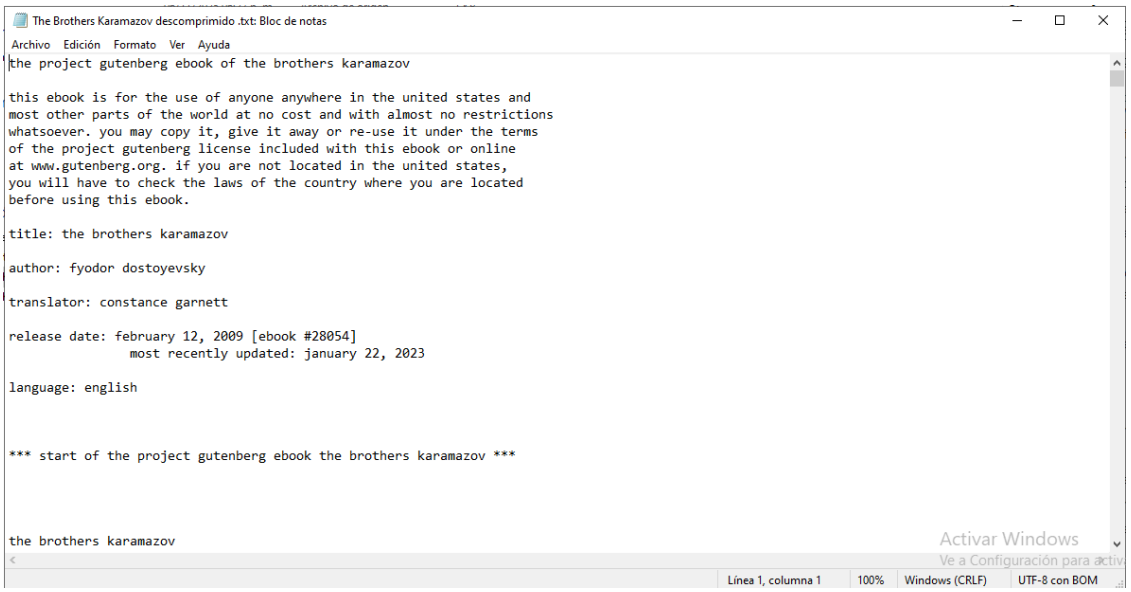


Podremos observar que el tamaño coincide con el archivo original.

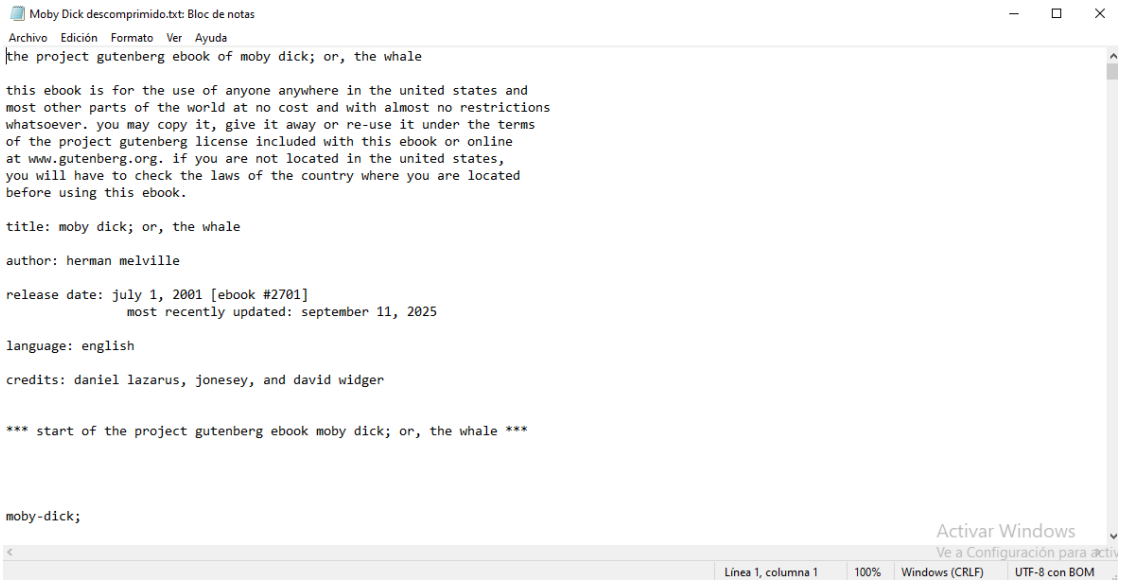


Y si abrimos el archivo para observar el contenido tendremos una visualización como la siguiente:

The brothers Karamazov



Moby Dick



EXPLICACIÓN DEL CÓDIGO.

Se implementó el algoritmo de huffman para comprimir y descomprimir texto de un libro recuperado de la página de project gutenberg (Free eBooks | Project Gutenberg), el libro que escogimos se llama 'Los hermanos Karamazov', el cual es un archivo de texto y nosotros nos encargamos de usar el algoritmo y mostrar en una interfaz el resultado de la compresión y la comparación de los tamaños entre el archivo original y el archivo comprimido, también podemos escoger el archivo a descomprimir.

ARCHIVO main.py

Comprimir.

```
36    def compress_file():
```

primero implementa 2 algoritmos que tenemos en el archivo equipoHuffman.py

```
    diccionario_h = huffman.huffman_encoding(contenido) # Se genera el diccionario Huffman para la codificacion
    comprimido = huffman.compresion(contenido, diccionario_h) # Se comprime el archivo con el diccionario previamente hecho
```

esto para obtener el diccionario de huffman y también para comprimir el diccionario y el contenido a comprimir en la función 'compresion'.

```
48      savefile = asksaveasfilename(defaultextension=".bin", filetypes=[("Binary files", "*.bin")]) # Configuracion del archivo binario
49      if savefile:
50          with open(savefile, "wb") as file:
51              pickle.dump(diccionario_h, file) # Guardar el diccionario en el archivo binario
52              comprimido.tofile(file) # Guardar la cadena binaria en el archivo binario
53
```

Después guardamos esta información en un archivo binario usando la librería pickle, creando un archivo para poder contener esta información.

Descomprimir.

Después tenemos la función de descomprimir llamada 'descomprimir'.

```
54    def descomprimir_archivo():
```

Que necesita la siguiente información:

- contenido
- diccionario
- y el nombre del archivo comprimido.

```
57          comprimido = BitArray(bin=contenido_bin.to01()) # Comvertir el bita
58          contenido = huffman.descompresion(comprimido, diccionario_h) # Desc
```

Después transformamos la información en un bitArray, y usando la función de descompresión del archivo huffman obtenemos el contenido descomprimido.

```
60      savefile = asksaveasfilename(defaultextension=".txt", filetypes=[("Text files", "*.txt")])
61      if savefile:
62          with open(savefile, "w", encoding="utf-8") as file:
63              file.write(f"{contenido}") # Escribir en el archivo de texto creado
64
```

Por último escribimos en archivo de texto la información obtenida para ser impresa y utilizada.

ARCHIVO equipoHuffman.py

COMPRIMIR

La primera función que encontraremos será la función de 'huffman_encoding' que recibe información como la variable data.

```
3   def huffman_encoding(data):
```

Lo primero que hacemos es usar la librería bitstring para manejar la información en bits.

```
equipoHuffman.py > huffman_encoding
1   from bitstring import BitArray
2
```

Esta función genera un diccionario de huffman, primero comprueba que se haya pasado como parámetro información, si no es así regresará nulo o vacío.

```
    if not data:
        return "", None
```

Después declaramos el diccionario de frecuencia, lo definimos vacío, y entramos a un for donde vamos a sacar caracter por caracter y el carácter que no se encuentre en data lo usamos como una índice el diccionario, y en ese índice lo igualamos a 0 de lo contrario si ese carácter ya estaba en el diccionario le sumamos 1 en el índice del diccionario con ese carácter.

```
    frequency = {}
    for char in data:
    # Calculamos la frecuencia de cada caracter
        if char not in frequency:
            frequency[char] = 0
        frequency[char] += 1
```

Creamos una lista de nodos, cada nodo contiene su frecuencia que es su peso, y además una súbita con el carácter y su código binario.

```
nodo = [[weight, [char, ""]] for char, weight in frequency.items()]
```

Mientras haya más de 1 nodo se ordenara los nodos por frecuencia, así que usamos un while pasando cada nodo, toma los dos nodos con menor frecuencia.

```
while len(nodo) > 1:
    # Ordenamos los nodos por frecuencia
    nodo = sorted(nodo)
    # Tomamos los dos nodos con menor frecuencia
    izquierda = nodo[0]
    derecha = nodo[1]
```

Después de eso entramos en un bucle for donde se añade 0 a los códigos del subárbol izquierdo y después de ese bucle for hay otro bucle for donde se le añade 1 al árbol derecho, esto construye el árbol binario de huffman.

```
22          for par in izquierda[1:]:
23              # agregamos '0' al frente de cada código
24              par[1] = '0' + par[1]
25          for par in derecha[1:]:
26              # agregamos '1' al frente de cada código
27              par[1] = '1' + par[1]
```

Después para que pueda parar el while eliminamos los dos nodos que usamos y que fueron agregados a cada rama del árbol pero agregamos uno nuevo que los combina, sumando su frecuencia.

```
28          nodo = nodo[2:]
29          # combinamos los dos nodo
30          nodo.append([izquierda[0] + derecha[0]] + izquierda[1:] + derecha[1:])
```

cuando llegue a un solo nodo, se detiene el while.

```
32      huffman_code = sorted(nodo[0][1:], key=lambda p: (len(p[-1]), p))
33      # Lo convertimos en un diccionario
34      huffman_diccionario = {char: code for char, code in huffman_code}
35      return huffman_diccionario
```

Extrae los códigos finales, los ordena por longitud y los convierte en un diccionario, el diccionario de huffman, retornamos el diccionario.

COMPRESIÓN.

tenemos ahora la función compresión.

```
37    def compresion(data, huffman_diccionario):
```

Que recibe como parámetros la información y el diccionario.
Entonces reemplaza cada carácter por su código huffman del diccionario, haciéndolo una cadena binaria y guardandolo en data_codificada.

```
# Codificamos la data usando el diccionario de Huffman
data_codificada = ''.join(huffman_diccionario[char] for char in data)
```

Después convertimos la cadena binaria en un objeto BitArray, que permite manipular bits eficientemente.

```
42        bit_array = BitArray(bin=data_codificada)
43        return bit_array
```

DESCOMPRESIÓN:

Recibe el bit_array y el diccionario de huffman para descomprimir.

```
44
45    def descompresion(bit_array, huffman_diccionario):
```

Esta función decodifica el Bit_array que recibe usando el diccionario inverso, invierte el diccionario para que se pueda buscar por código y regresar el valor del carácter.

```
47        diccionario_inverso = {codigo: char for char, codigo in huffman_diccionario.items()}
48
```

Convierte el Bit_array que recibe en una cadena binaria, esto para poder realizar ahora la búsqueda bit por bit, acumulando en una variable llamada codigo_actual cada bit hasta que dentro de un if se pregunta que si código actual se encuentra en el diccionario inverso, cuando encuentra un codigo que esta en el diccionario, se agrega a la variable data_decodificada el valor que tiene el diccionario inverso en el índice ‘codigo_actual’, despues de eso a código actual se le declara vacío para que esté limpio en la siguiente búsqueda.

```
    data_codificada = bit_array.bin

    codigo_actual = ""
    data_decodificada = ""

    # Decodificamos la cadena binaria a cadena de caracteres con el diccionario inverso
    for bit in data_codificada:
        codigo_actual += bit
        if codigo_actual in diccionario_inverso:
            data_decodificada += diccionario_inverso[codigo_actual]
            codigo_actual = ""
    return data_decodificada

```