UNIVERSIDAD
Panamericana

Proyecto Final - Compresión de archivos

Estructura de datos y algoritmos

Atzin Eduardo Cruz Briones

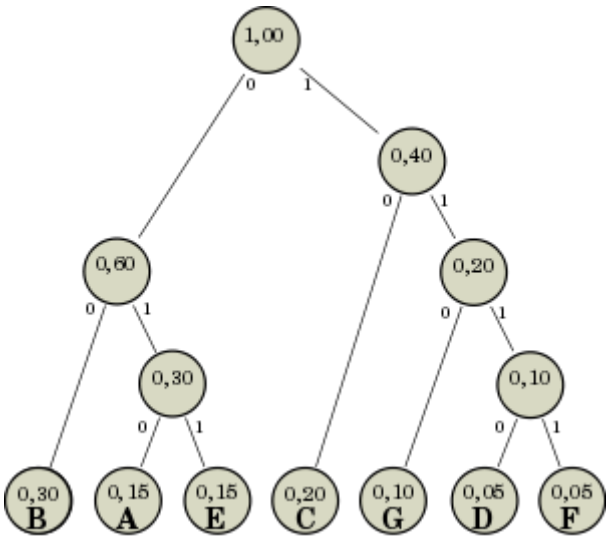
Jose Armando Cerna Villaseñor

Repositorio: https://github.com/Joe-IA/File_compression_sys

2 de Diciembre del 2023

Algoritmo principal - Huffman Adaptativo

Este algoritmo es una variante del Huffman tradicional, con la excepción de que en el huffman normal la tabla de de códigos es construida a priori basándose en la frecuencia de los símbolos, el Huffman adaptativo permite que la tabla evolucione y se adapte a medida que se procesa la secuencia de entrada.



Pasos:

1.- Inicio del árbol de Huffman
- Comienza con el árbol de huffman tradicional que representa una frecuencia uniforme a todos los símbolos

2.- Lectura de símbolo
- Se lee un símbolo de la secuencia de entrada

3.- Codificación y actualización del árbol
- Se utiliza el árbol de huffman para codificar el símbolo leído
- Luego, se actualiza el árbol para reflejar la nueva frecuencia del símbolo.

4.- Reconstruido el árbol
- Después de cada codificación el árbol se reorganiza para mantener las propiedades del Huffman

5.- Repetición del proceso:
- Se repiten los pasos del 2 al 4 hasta lograr terminar el texto

Archivos:

Tenemos el archivo **Main.py** donde se definen las clases con los métodos para lograr la codificación Huffman: la clase Nodo, para establecer los diversos nodos del árbol que se utilizaran para la codificación; la clase Huffman, para la construcción del árbol y la generación de la tabla para los caracteres en el archivo mediante el algoritmo de ***Huffman adaptativo***; la clase Encoding que contiene los métodos para codificar y decodificar los archivos; y la clase Compresión que es la que se encarga de implementar la codificación o deco
edificación de acuerdo al tipo de archivo que se quiere comprimir o descomprimir.

Se cuenta con tres módulos distintos (**video_optimizado, File Txt y audio**), en el que se implementan clases para la modificación de los archivos, para obtener su información en formato utilizable para el algoritmo o para guardar el contenido ya procesado en un nuevo archivo binario. Todos estos se importan y utilizan en la función Main.

También se tiene el archivo más importante, en el que se va a correr el programa y el cual no solo se utiliza para mostrar la interfaz que permite al usuario interactuar con el programa, sino el que implementa todas las funciones para que juntas realicen el trabajo de comprimir, que es el archivo de ***user_interface.*** Este es el archivo que hay que ejecutar.

Uso de librerías:

- pydub: Libreria que nos permite trabajar de manera más flexible con audio.
- base64: permite codificar la representación binaria de los archivos en caracteres utf-8
- tkinter: Contiene los widgets y las funciones para la construcción de interfaces gráficas.
- bitarray: Permite guardar 0’s y 1’s como bits y no como enteros.
- Pickle: permite guardar objetos en archivos binarios.
- Pillow: Permite la modificación de los distintos formatos de imágenes.
- Numpy: Permite trabajar con matrices y vectores.

Funciones:

Con el fin de ahorrar memoria y aumentar el rendimiento, los métodos de las distintas clases se declararon como estáticos, esto con la finalidad de no instanciar la clase de la que provienen y llamarlos utilizando solamente el nombre de la clase.

Compresión de audio - O(n log n)
Archivos que acepta - .wav

Para lograr la compresión de audio lo que se hace es que pasa el audio a archivo binario, donde el archivo binario lo pasa a texto mediante una librería llamada base64, este archivo de texto se pasa al huffman. Cosas que pasan de convertir el audio a base64 es que aumenta el peso en un 30%, y al pasar del texto al huffman se reduce en 35% es decir, el audio nomas se comprime un 35%.
Pasos que sigue para ser comprimido:
1.- Audio a binario
2.- Binario a base64
3.- base64 a HuffMan

```
class AudioToText:
    def audio_to_binary(self, audio_file):
        audio = AudioSegment.from_file(audio_file, format="wav")
        audio = audio.set_frame_rate(44100)
        audio = audio.set_channels(1)
        return audio.raw_data

    def binary_to_text(self, binary_data):
        return base64.b64encode(binary_data).decode()

    def process_audio_to_text(self, input_audio_file, output_text_file):
        # Convertir audio WAV a binario
        binary_data = self.audio_to_binary(input_audio_file)

        # Convertir binario a texto ASCII y guardar en un archivo
        text_data = self.binary_to_text(binary_data)
        with open(output_text_file, 'w') as file:
            file.write(text_data)
```

Pasos que sigue para descomprimir:
1.- Huffman a base64
2.- base64 a binario
3.- binario a Audio

Compresión de texto (.txt) - O(n log n)

La compresión de los archivos de texto plano son los que se realizan con mayor facilidad, debido a que todo su contenido está en texto plano que no necesita ser transformado a ningún otro formato y que no presenta otro tipo de metadata que necesite ser procesada. Para reducir su tamaño, se lee su contenido y se pasa su texto al algoritmo de Huffman con el objetivo de representar cada uno de sus caracteres con un menor número de bits. Después de que el texto es procesado, se crea un diccionario con las claves de decodificación que se almacenará junto al contenido codificado dentro de un archivo binario.

Esta función se declara dentro del módulo principal y se importa al módulo de la interfaz gráfica.

```
@staticmethod
def compress_text(directory, name, encoding):
    with open(f"{directory}/{name}.txt", "r") as f:
        text = f.read()
    encoded_text, tree_code_table = encoding.adaptive_huffman_encoding(text)
    bits = bitarray()
    bits = bitarray([int(i) & 1 for i in encoded_text])
    with open(f"{directory}/{name}.bin", "wb") as bf:
        pickle.dump((tree_code_table, bits), bf)
```

Compresión de imagen (.bmp) - O(n log n)

Para aplicar la codificación Huffman a una imagen es necesario transformarla en un matriz de bits, para eso hacemos uso de las librerías Pillow y Numpy; la primera permite abrir y leer el contenido de una imagen y la segunda manipular objetos vectoriales. Además de la información de los píxeles dentro de una imagen, necesitamos obtener su metadata cómo sería su tamaño y sus canales de colores, los cuales se obtienen en un inicio y se almacenan para reconstruir la imagen durante la descompresión. Mediante la función ***tobytes()*** de un objeto de la librería Numpy se obtienen la metadata y la información de los píxeles de una imagen.

Los bytes de la matriz se transforman a un texto plano y se toman de 7 en 7 para transformarlos a su equivalente en ascii, en caso de que la cadena no termine con múltiplos de 7, se le agrega un margen el cual será eliminado después. Una vez se obtenga la cadena de caracteres que corresponde a la matriz de la imagen, se aplica el algoritmo de Huffman para reducir el tamaño que ocupan. Se almacena en el archivo binario, la metadata, el diccionario de decodificación y la serie de bits que construyen el mensaje de la imagen en texto plano.

```python
@staticmethod
def compress_image(directory, name, encoding):
    width, height, channels, binarios = ImagesToTxt.image_to_text(f"{directory}/{name}.bmp")
    cadena_bits = ''.join(format(byte, '08b') for byte in binarios)
    paddinf = 7 - len(cadena_bits) % 7 if len(cadena_bits) % 7 != 0 else 0
    cadena_bits = cadena_bits + "0" * paddinf
    binarios = ''.join(chr(int(cadena_bits[i:i+7], 2)) for i in range(0, len(binarios), 7))
    encoded_text, tree_code_table = encoding.adaptive_huffman_encoding(binarios)
    bits = bitarray()
    bits = bitarray([int(i) & 1 for i in encoded_text])
    with open(f"{directory}/{name}.bin", "wb") as bf:
        pickle.dump((tree_code_table, bits, width, height, channels, paddinf), bf)
```

Compresión de video - O(n log n)

Para lograr aplicar la compresión huffman al video lo que se hace es leer el archivo mediante el lector de python para luego pasarlo en su forma raw, de esta manera logramos pasar el video a bites, despues de eso por medio de base64 pasamos todos esos bites a su forma de letra que posteriormente es analizada y comprimida por el archivo Huffman.

```python
def compress_video(directory, name, encoding):
    print("Comprimiendo video...")
    video = VideoProcessor.read_video(f"{directory}/{name}.mp4")
    video = VideoProcessor.binary_to_text(video)
    encoded_text, tree_code_table = encoding.adaptive_huffman_encoding(video)
    bits = bitarray()
    bits = bitarray([int(i) & 1 for i in encoded_text])
    with open(f"{directory}/{name}.bin", "wb") as bf:
        pickle.dump((tree_code_table, bits), bf)
```

Para la descompresión hace lo opuesto, del huffman se descomprime, pasa a bites y se escribe en un nuevo archivo con terminación .mp4.

```python
@staticmethod
def decompress_video(directory, name, encoding):
    print("Descomprimiendo video...")
    with open(f"{directory}/{name}.bin", "rb") as bf:
        tree_code_table, bits = pickle.load(bf)
    decoded_text = encoding.decode(bits.to01(), tree_code_table)
    video = VideoProcessor.text_to_binary(decoded_text)
    VideoProcessor.write_video(f"{directory}/{name}.mp4", video)
```

Descompresión de audio O(n)

Se crea una instancia de Audioprocessor que nos ayuda a cambiar un audio a binario, a texto o devuelta a audio. Obtenemos el texto después de decodificarlo y con un método de Audioprocessor convertimos el texto en audio. Finalmente con la función AudioSegment exportamos el audio en el formato que le corresponde; en este caso, wav.

```python
@staticmethod
def decompress_audio(directory, name):
    audio_processor = AudioProcessor()
    input_audio_file = f"{name}.bin"
    audio_processor.process_text_to_audio(directory, input_audio_file, f"{name}
    _decompressed.wav")
```

Descompresión de texto O(n)

Se obtiene el árbol y el texto codificado del árbol binario; se pasan a la función de decodificación y el texto obtenido de la reversión del proceso de codificación se envía a un archivo de texto plano.

```python
@staticmethod
def decompress_text(directory, name, encoding):
    with open(f"{directory}/{name}.bin", "rb") as bf:
        tree_code_table, bits = pickle.load(bf)
    decoded_text = encoding.decode(bits.to01(), tree_code_table)
    with open(f"{directory}/{name}.txt", "w") as f:
        f.write(decoded_text)
```

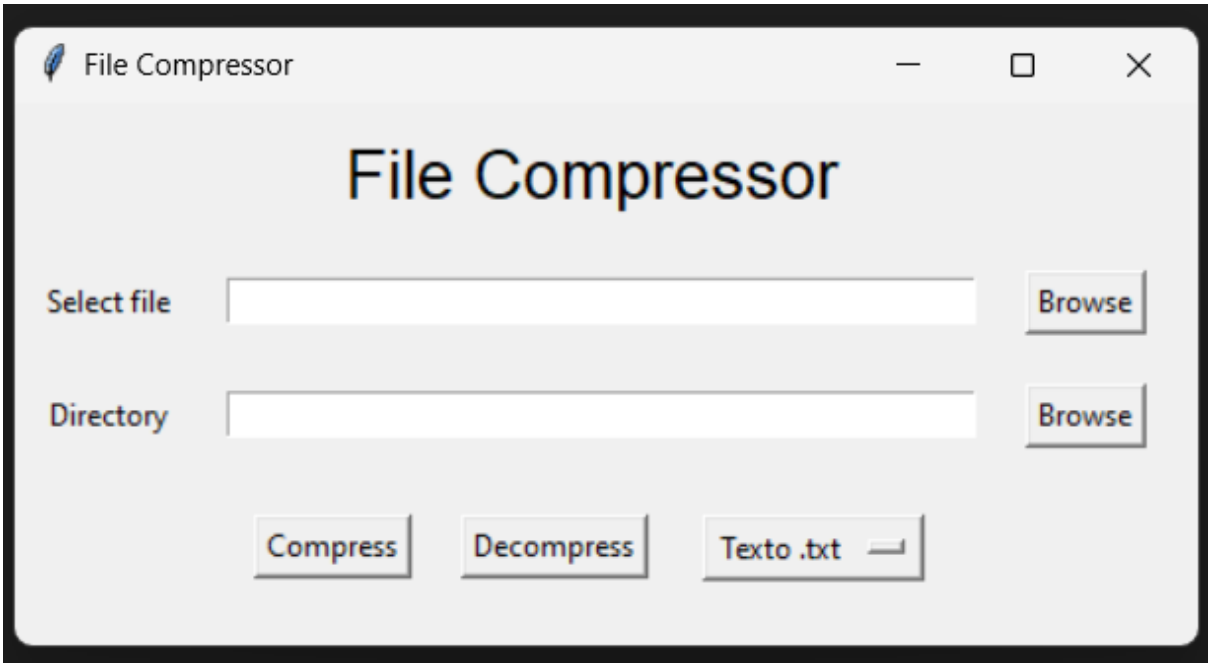
Descompresión de video O(n)

Se obtiene el árbol y el texto codificado del archivo binario, mediante la separación con la función de pickle, al tener las dos variables, la del árbol y el mensaje comprimido, este se pasa a una función llamada

decode, donde convierte el mensaje comprimido en un mensaje de texto, para luego pasar de texto a video mediante dos funciones.

```python
@staticmethod
def decompress_video(directory, name, encoding):
    with open(f"{directory}/{name}.bin", "rb") as bf:
        tree_code_table, bits = pickle.load(bf)
    decoded_text = encoding.decode(bits.to01(), tree_code_table)
    video = VideoProcessor.text_to_binary(decoded_text)
    VideoProcessor.write_video(f"{directory}/{name}.mp4", video)
```

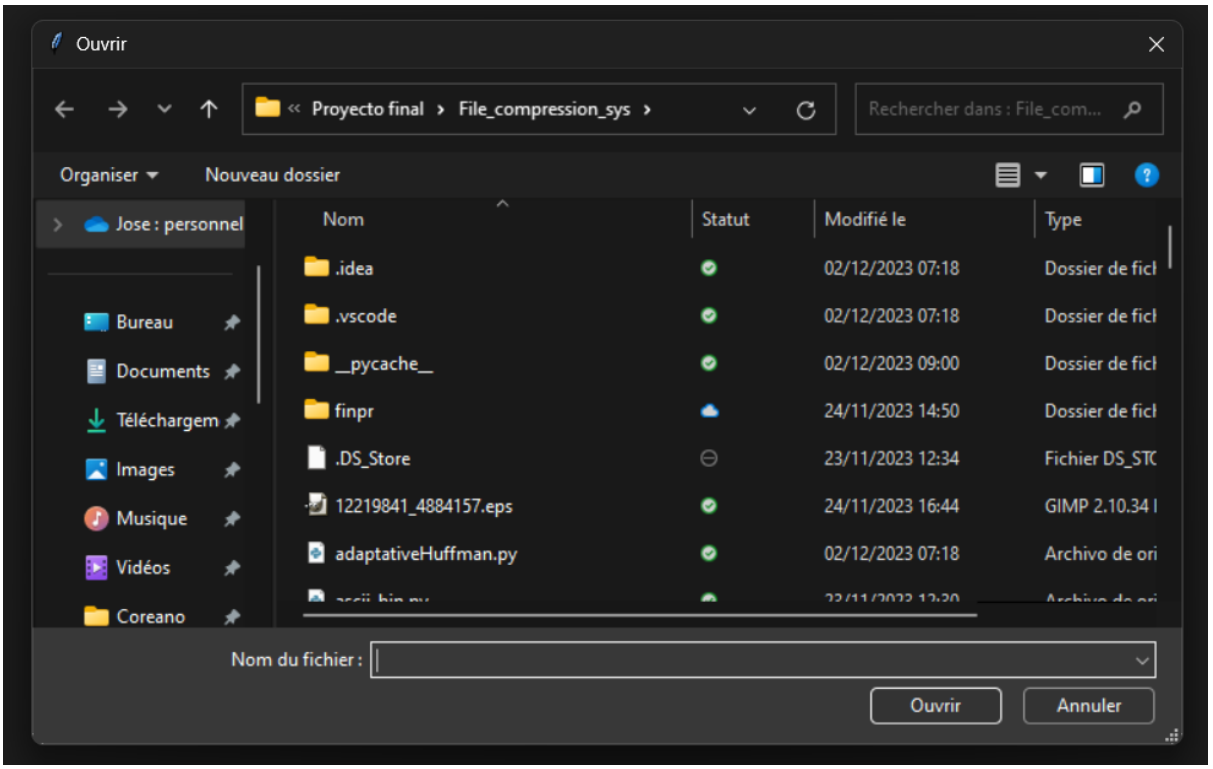
Interfaz gráfica



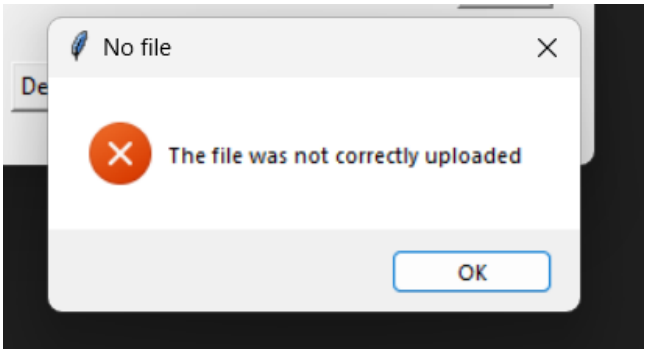
La interfaz cuenta con espacios para la colocación del archivo que se quiere comprimir o descomprimir y del espacio en almacenamiento en el que se guardará el archivo procesado. El archivo que se cree tendrá el mismo nombre que el archivo original. Si el archivo original se comprimió, el nuevo archivo tendrá la extensión .bin y si el archivo se descomprimió, el archivo tendrá la misma extensión que tenía antes de ser comprimido, siempre y cuando esté dentro de los tipos de archivos que admita el programa.

Los cuadros de entrada solo muestra de manera ilustrativa, el archivo seleccionado y la carpeta empezando desde 3 niveles arriba, pero el archivo, ni la dirección deben ser ingresados vía teclado, puesto que se necesita la ruta absoluta para obtener la información de ambos. Es por eso que se debe hacer uso de los dos botones que se encuentran a

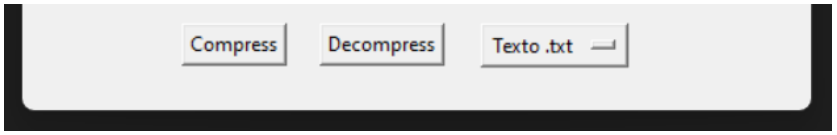
lado de las barras de entrada. Estos les permitirá acceder al sistema de archivos y escoger el archivo o carpeta vista desde la carpeta raíz.

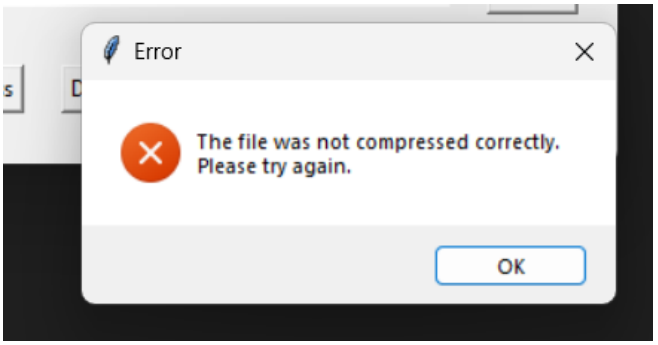


En caso de que no se seleccione un archivo o directorio, el programa avisará que hay un error.



Los botones de abajo indican que acción se quiere realizar con el archivo que se envió, si se desea comprimir, descomprimir y el tipo de archivo que se está comprimiendo y descomprimiendo. Si el archivo no puede ser comprimido o descomprimido a un formato específico, quiere decir que no son compatibles y el programa indicará que el procesamiento no será posible.


Compress
Decompress
Texto .txt


Error
The file was not compressed correctly.
Please try again.
OK