ESDI II

Universidad Panamericana Campus Aguascalientes

Cristian Aragón Salazar - 0250005

Compresor de Archivos (Github)

ESDI II 1
Aún más información en el README.md de github 1
IDEA 1
Pensamiento de la idea 1
Problemas 2
FUNCIONAMIENTO 2
config.py 2
main.py 2
Identifier.py 3
Find.py 3
Find_replace.py 4

Aún más información en el README.md de github

IDEA

Pensamiento de la idea

1. Al tratarse de un compresor de archivos, lo primero que pensé fue en WinRAR, ya que es el referente de archivos comprimidos, por lo cual lo había pensado hacer como él, pero iba a ser demasiado complejo, por lo cual descarté la idea y mejor decidí hacer algo más simple, con una interfaz gráfica simple pero funcional.
2. **Interfaz gráfica:** Mi primera idea fue hacer un botón específico para cada tipo de archivo, pero después me di cuenta que podía hacerla más sencilla y cómoda si fuese un solo botón para comprimir y otro para descomprimir, entonces los incorporé, para luego darme cuenta que hacía falta cómo se seleccionaría el archivo, por lo cual incorporé un

nuevo botón en el que se seleccionara el archivo y también una barra donde mostrara la ruta del archivo a comprimir o descomprimir.

3. **Distribución de proyecto:** Como en el otro proyecto que fue el RegEx, para tener un código más limpio y ordenado pensé en distribuirlo en diferentes archivos que hicieran diferentes cosas y llamarlas cuando fuese necesario, en este caso fueron 3 archivos, **comprimir.py**, **descomprimir.py** y **main.py**.

Problemas

1. Al principio no se me ocurría cómo encontrar la extensión del archivo de una manera sencilla y que funcionara bien, sin embargo, después me di cuenta que lo podía hacer usando el comando `os.path.splitext(os.path.basename(self.archivo))` ya que con esto obtengo el nombre del archivo con su extensión y luego a eso le hago un split y me separa el nombre del archivo y su extensión correspondiente.

FUNCIONAMIENTO

main.py

Complejidad: Depende de comprimir.py y descomprimir.py

En este archivo main se tiene a la clase *CompresorArchivoApp,* en esta clase se define a la interfaz para el proyecto, además tiene métodos que sirven para su funcionamiento, como lo son seleccionar_archivo, la cual se encarga de seleccionar el archivo que se va a comprimir o descomprimir. Otros como afirmative_message o error_message, lo que hacen es crear la ventana de compresión exitosa o, en caso de que haya un error, mostrarlo en otra ventana.

Por último, los métodos de compresión y descompresión, que son comprimir_archivo y descomprimir_archivo, los cuales se encargan primeramente de obtener el nombre del archivo y su extensión, para así poder ir a cada caso de extensión diferente. Además de que en el caso de compresión se crea un archivo aparte que se llama *nombre_archivo_huffman_tree.txt,* el cual se encargará de almacenar el árbol de huffman para que se pueda descomprimir el archivo. En el caso de descomprimir hace lo mismo, solo que al momento del árbol, en lugar de crearlo lo busca y lo abre.

comprimir.py

Complejidad:

- build_huffman_tree: **O(n log n) *Donde n es la altura del árbol***

- process_text, process_image, process_video, process_audio: **O(m) *Donde m es el tamaño del archivo en bytes***
- compress_file, compress_img_file, compress_video_file, compress_audio_file: **O(m) *Donde m es el tamaño del archivo en bytes***

Este archivo contiene dos clases que finalmente ya no utilicé, ya que quería usar un método de compresión diferente a la hora de comprimir video y que este fuese más eficiente, pero no resultaba ya que o hacía más grande el archivo comprimido o simplemente no logré hacer que funcionara, por lo cual terminé decidiendo por utilizar huffman al igual que los demás.

La clase HuffmanNode es la cual representa los nodos que se usan en el árbol de Huffman.

En el caso de HuffmanTree se trata de la clase que representa el árbol de Huffman, es la que se encarga de generar el código de Huffman con la frecuencia de los caracteres. Empieza con el constructor de la clase, en la cual se define la frecuencia de los caracteres y la codificación de Huffman, además yo utilicé una variable que fuera root para volver a la raíz del árbol.

En la función build_huffman_tree, lo que hace es generar el árbol de huffman y después de haberlo generado se genera el código de huffman, en la función generate_huffman_codes. En la función de serialize_huffman_tree, lo que hace es que toma el archivo de árbol de huffman y lo convierte a secuencias de bytes para poder almacenarlo en un archivo.

Luego, en las funciones de process_(tipo de archivo) lo que hacen es abrir el archivo a manera de bytes y obtienen la frecuencia de aparición de los datos.

Finalmente en las funciones de compress_(tipo de archivo) se abre el archivo y se lee el contenido, para después crear un objeto de tipo bitarray, el cual lo usé para almacenar los bits de la compresión, luego, con el método encode de bitarray, lo uso para comprimir el contenido del archivo original, en donde tomo un diccionario que asigna caracteres de los códigos de Huffman que le corresponden. Por último abro el archivo de salida pero como escritura binaria, para después con el método tofile escribo los bits en el archivo de salida.

Cabe aclarar que para los archivos de texto utilicé la apertura como archivo normal, pero para los demás, imagen, video y audio utilicé la apertura ‘rb’ para trabajar como archivos binarios.

descomprimir.py

Complejidad:

- decode_huffman: **O(m) *Donde m es la longitud del contenido comprimido***
- decompress_file, decompress_img_file, decompress_vid_file, decompress_audio_file: **O(m) *Donde m es el tamaño del archivo en bytes***

En la parte de descomprimir lo que hice fue crear la misma clase HuffmanNode para representar los nodos del árbol de huffman. En la clase HuffmanDecoder lo primero es su constructor, el cual igual que en el caso de descomprimir toma la frecuencia de los caracteres.

La función deserialize_huffman_tree, lo que hace es abrir el archivo de árbol de forma binaria y después carga el contenido a la raíz, la cual representa la raíz del árbol de huffman.

En la parte de decompress_(tipo de archivo) lo que hace es que abre el archivo y crea un objeto de tipo bitarray para que en él se guarde el contenido del archivo comprimido, para después en una variable llamada decompressed se llama a la función decode_huffman_(tipo de archivo), la cual lo que hace es se va a la raíz del árbol y en una variable va guardando el contenido ya descomprimido yendo a las hojas del árbol y añadiendo el código original. Después de ese proceso vuelve a decompress_(tipo de archivo) abre el archivo de salida y lo guarda con la función write.

LIBRERÍAS

Utilicé varias librerías, las enumero y pongo el propósito para el cual las ocupé:

tkinter:

Propósito: Para la creación de la interfaz gráfica.

Importación en el código: import tkinter as tk

Uso: La uso para construir la interfaz del programa.

filedialog y messagebox de tkinter:

Propósito: Para manejar diálogos de selección de archivos y mostrar mensajes de información o error.

Importación en el código: from tkinter import filedialog, messagebox

os:

Propósito: Para operaciones relacionadas con el sistema operativo, como la manipulación de rutas de archivos.

Importación en el código: import os

sys:

Propósito: Para manipular la ruta del sistema y agregar la ruta del directorio de funciones al path.

Importación en el código: import sys

pickle:

Propósito: Para la serialización y deserialización de objetos.

Importación en el código: import pickle

Uso: La uso para almacenar y recuperar la estructura del árbol de Huffman en archivos.

heapq:

Propósito: Para realizar operaciones en colas de prioridad.

Importación en el código: import heapq

Uso: La utilizo para construir el árbol de Huffman.

bitarray:

Propósito: Para representar y manipular secuencias de bits.

Importación en el código: from bitarray import bitarray

Uso: La uso en las operaciones de compresión y descompresión.

Estas dos ya no fueron usadas en la implementación final, pero las menciono:

cv2 (OpenCV):

Propósito: Para operaciones de procesamiento de imágenes y videos.

Importación en el código: import cv2

Uso: La usaba en la cuantización de color para videos.

numpy:

Propósito: Para operaciones matriciales y manipulación de datos.

Importación en el código: import numpy as np

Uso: La usaba en conjunción con OpenCV para manipulación de datos de imágenes.