



- **Universidad de Guadalajara**
- **CUCEI**
- **División de Tecnologías para la Integración Ciber-Humana**
- **Departamento de Ciencias Computacionales**
- **Tema:** Algoritmo de Huffman
- **Integrantes del equipo:**
- Nicolás Alejandro Garín Gutiérrez
- Gerardo Dagnino Zatarain
- **Materia:** Análisis de algoritmos
- **Sección:** D01
- **Calendario:** 2025-B
- **Profesor:** Jorge Ernesto López Arce Delgado
- **Fecha:** 06/11/2025

Índice:

Introducción.....	2
Requerimientos.....	2
Requerimientos del sistema.....	2
Requerimientos del usuario.....	2
Funcionamiento General.....	3
Comprimido.....	3
Descomprimido.....	4
Instrucciones de uso.....	5

Introducción

El objetivo de este programa es entender la manera en que el algoritmo de Huffman funciona, siguiendo esto, el programa comprime archivos de texto, terminación “.txt”, y para hacerlo más completo también los descomprime, demostrando el tamaño del archivo y su tamaño después de comprimirlo, junto con su porcentaje de compresión.

Requerimientos

Requerimientos del sistema

Estos son los requisitos técnicos para ejecutar el programa correctamente:

- **Sistema operativo:** Windows 10 o superior, macOS, o distribuciones Linux (Ubuntu 20.04+, Debian, etc.)
- **Python:** Versión 3.10 o superior instalada
- **Memoria RAM:** Mínimo 2 GB; recomendado 4 GB o más
- **Espacio en disco:** Al menos 50 MB libres para el programa y archivos de entrada/salida
- **Dependencias:** Revisar requirements.txt

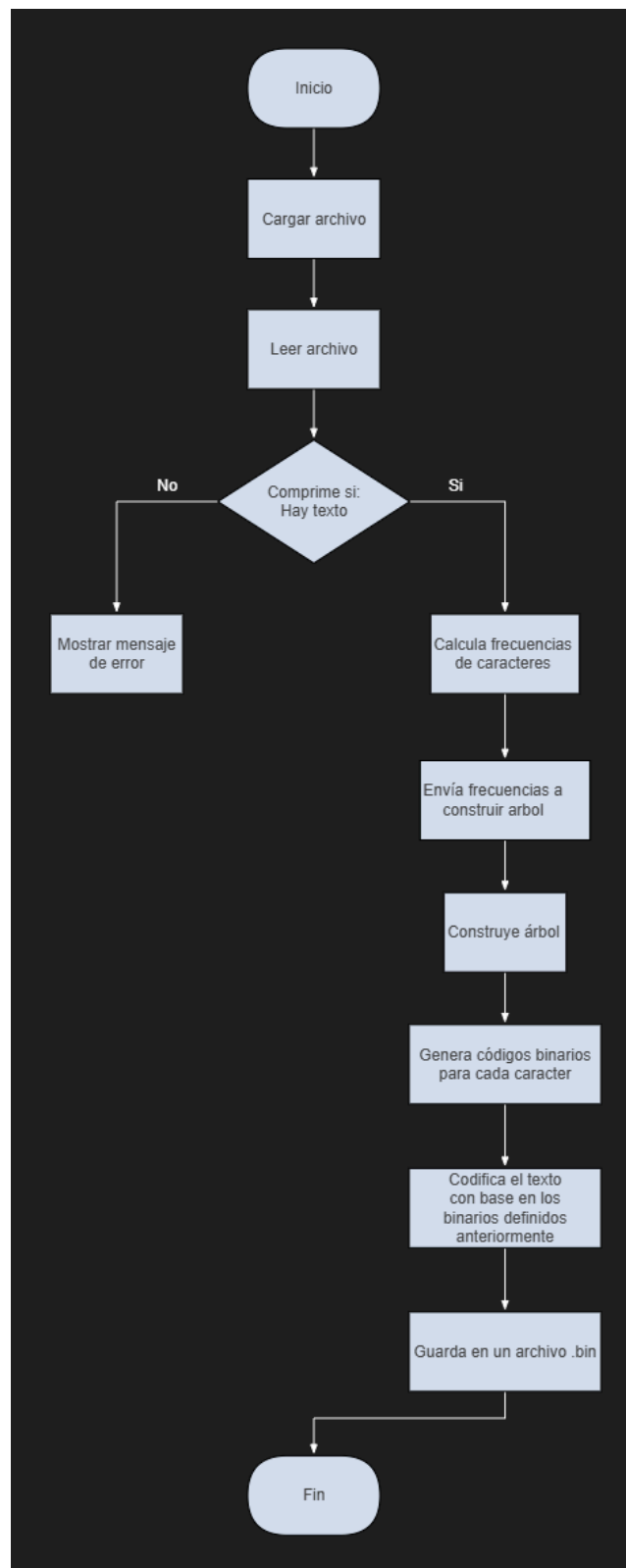
Requerimientos del usuario

Para que el usuario pueda operar el programa sin complicaciones:

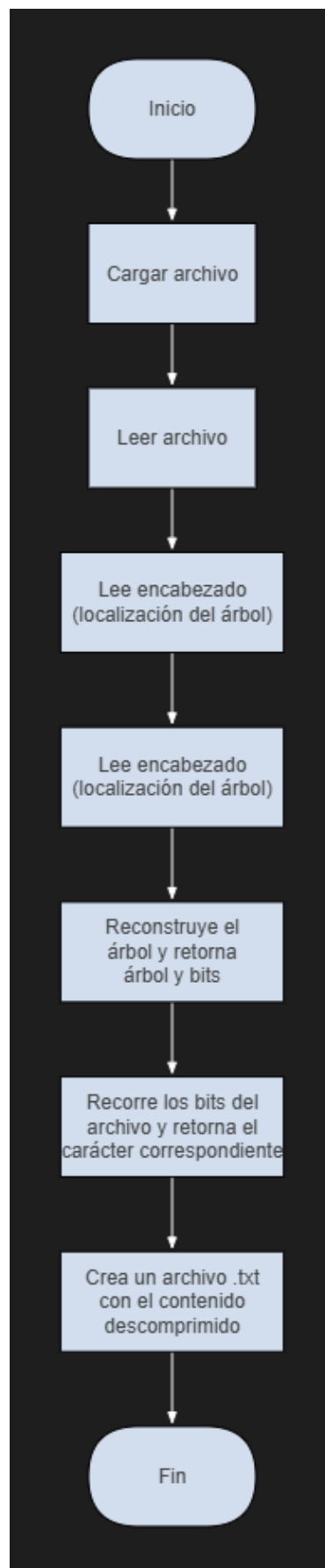
- **Conocimientos básicos de Python:** Ejecutar scripts desde terminal o IDE
- **Acceso a archivos de texto:** El programa trabaja con archivos .txt para compresión y descompresión
- **Permisos de lectura/escritura:** En las carpetas donde se ejecuta el programa

Funcionamiento General

Comprimido

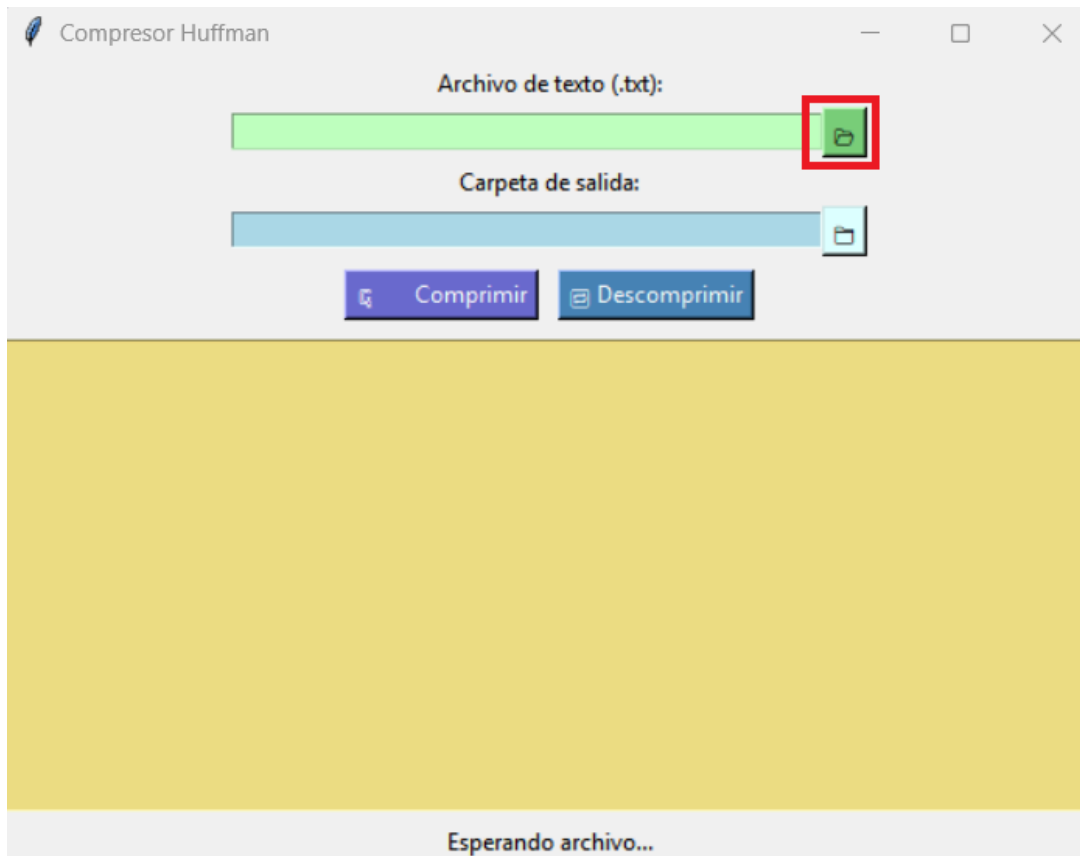


Descomprimido



Instrucciones de uso

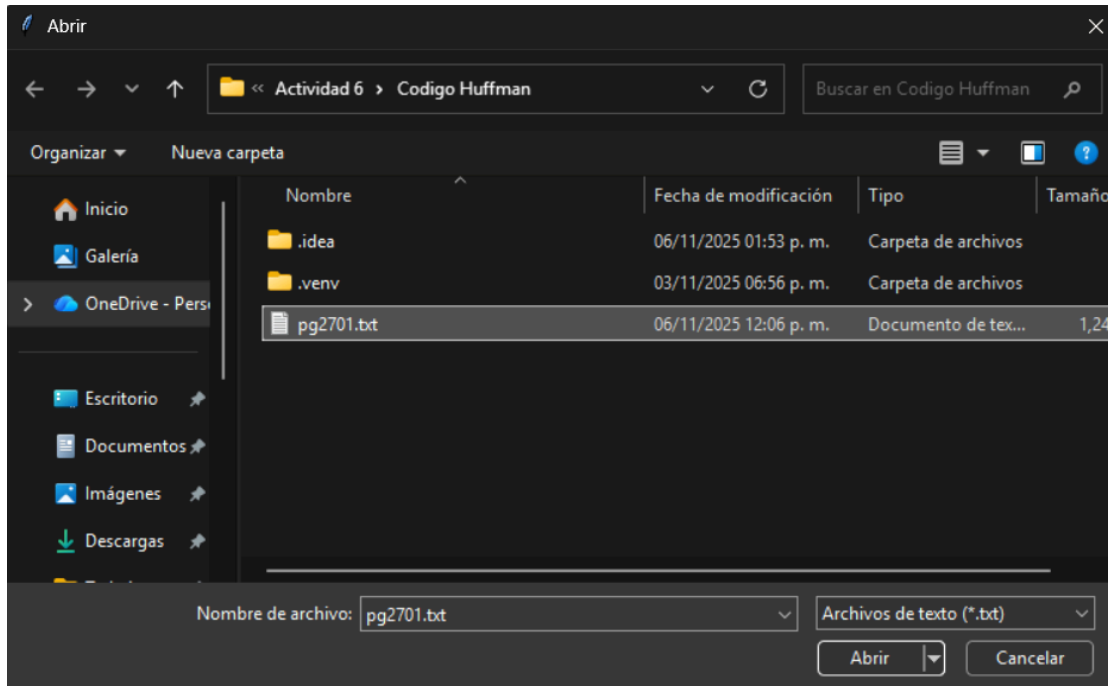
1. El programa iniciará mostrando una variedad de botones por presionar, para su correcto uso, se deberá presionar el siguiente botón.



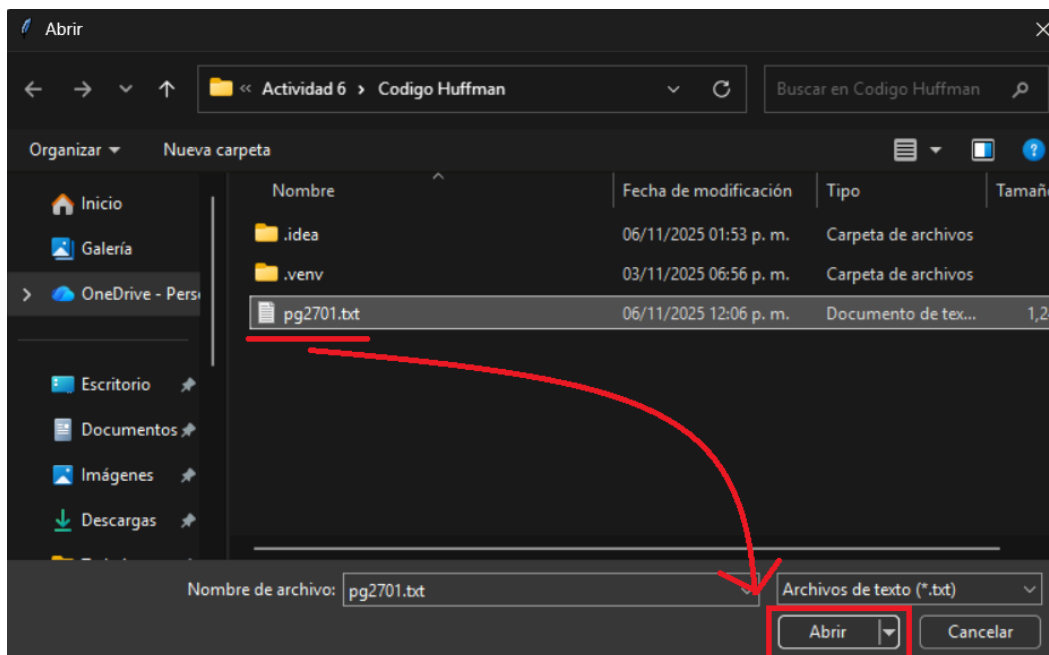
Este botón funciona bajo los lineamientos del siguiente código:

```
# --- Sección de selección de archivos ---
tk.Label(root, text="Archivo de texto (.txt):").pack()
frame_in = tk.Frame(root)
frame_in.pack(pady=2)
self.entry_in = tk.Entry(frame_in, width=50, bg="DarkSeaGreen1")
self.entry_in.pack(side=tk.LEFT)
tk.Button(frame_in, text="📁", command=self.elegir_archivo, bg="PaleGreen3",
          fg="black", activebackground="PaleGreen2", activeforeground="black").pack(side=tk.LEFT)
```

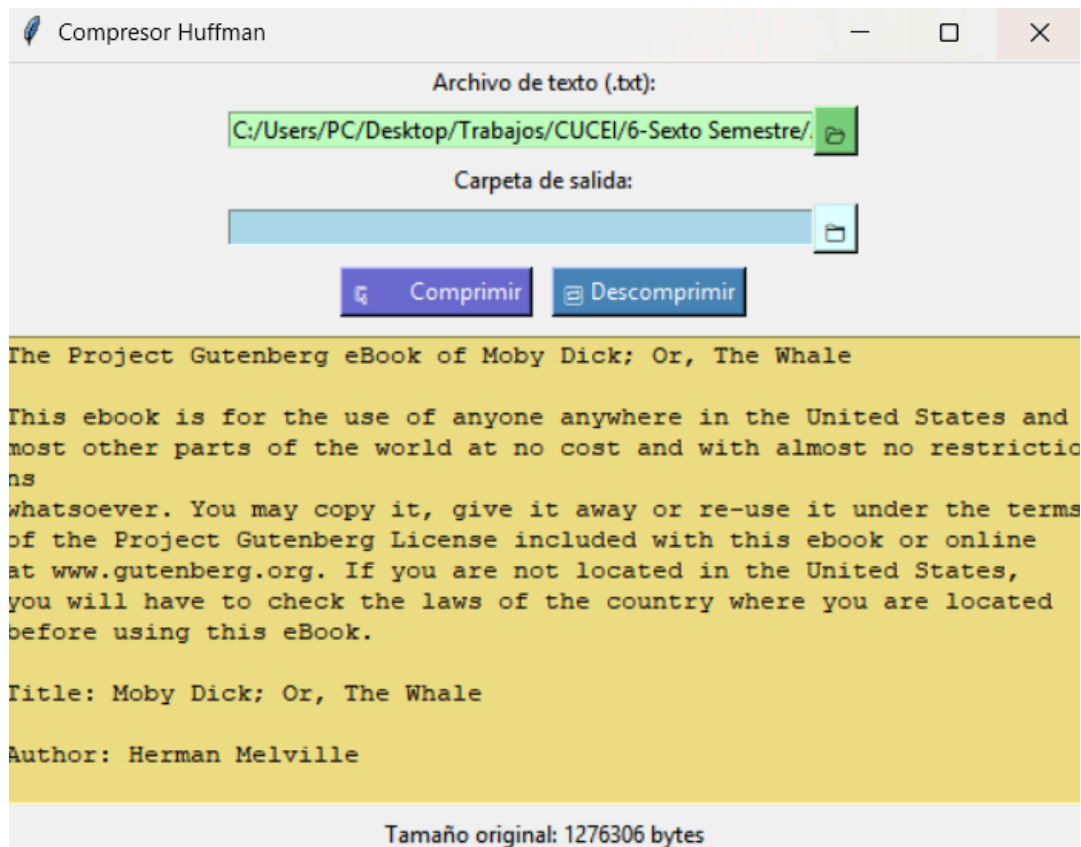
2. Se abrirá una ventana que permitirá seleccionar el archivo de extensión .txt a comprimir.



3. Se seleccionará el archivo extensión .txt y se presionará el botón “Abrir”.



4. Una vez seleccionado el programa mostrará una vista del contenido dentro del archivo extensión .txt junto con su tamaño en bytes.



El menú de selección de archivos es la consecuencia de presionar el botón y activar el siguiente comando:

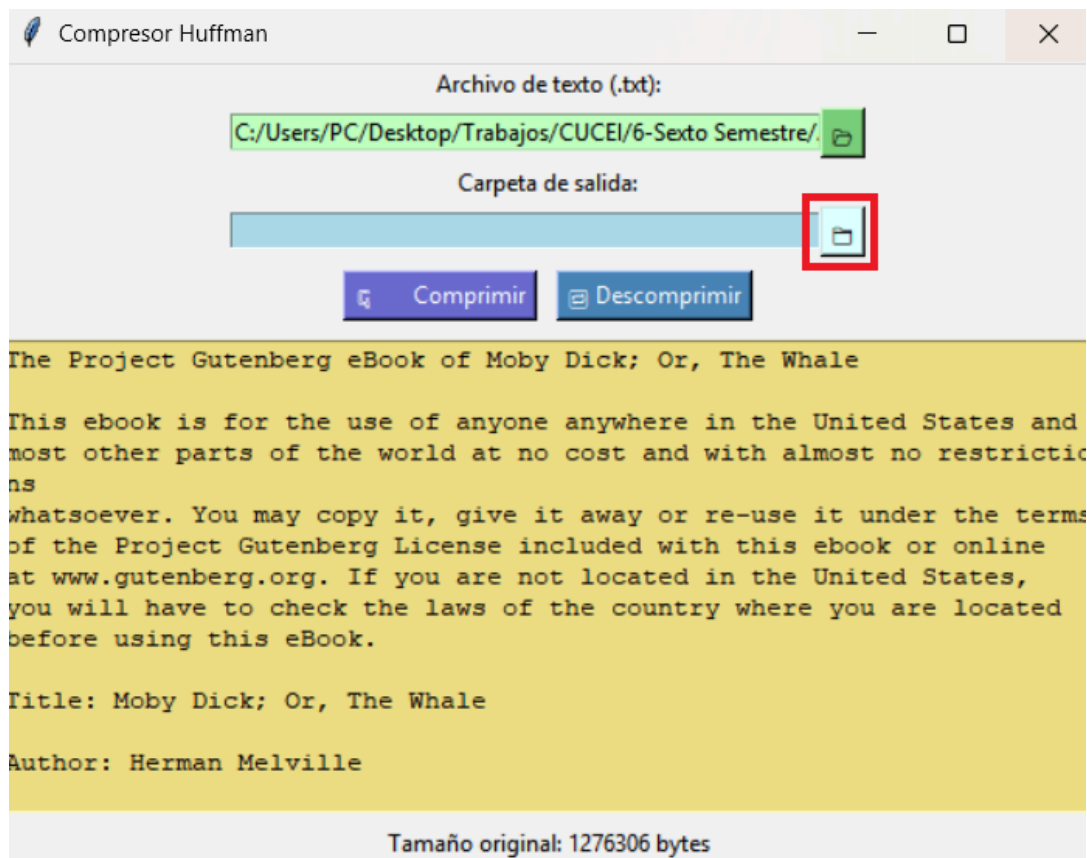
```
# --- Sección de selección de archivos ---
tk.Label(root, text="Archivo de texto (.txt):").pack()
frame_in = tk.Frame(root)
frame_in.pack(pady=2)
self.entry_in = tk.Entry(frame_in, width=50, bg="DarkSeaGreen1")
self.entry_in.pack(side=tk.LEFT)
tk.Button(frame_in, text="📁", command=self.elegir_archivo, bg="PaleGreen3",
          fg="black", activebackground="PaleGreen2", activeforeground="black").pack(side=tk.LEFT)
```

el cuál luego de seleccionar el archivo hace lo siguiente:

```
def elegir_archivo(self): 1 usage
    ruta = filedialog.askopenfilename(filetypes=[("Archivos de texto", "*.txt")])
    if ruta:
        self.archivo_entrada = ruta
        self.entry_in.delete(first=0, tk.END)
        self.entry_in.insert(index=0, ruta)
        with open(ruta, 'r', encoding='utf-8') as f:
            self.texto = f.read()
        self.vista.delete(index1=1.0, tk.END)
        self.vista.insert(tk.END, self.texto)
        self.tamano.config(text=f"Tamaño original: {os.path.getsize(ruta)} bytes")
```

- define variable ruta, que contiene la ruta del archivo .txt seleccionado.
- comprueba la existencia de una ruta y después:

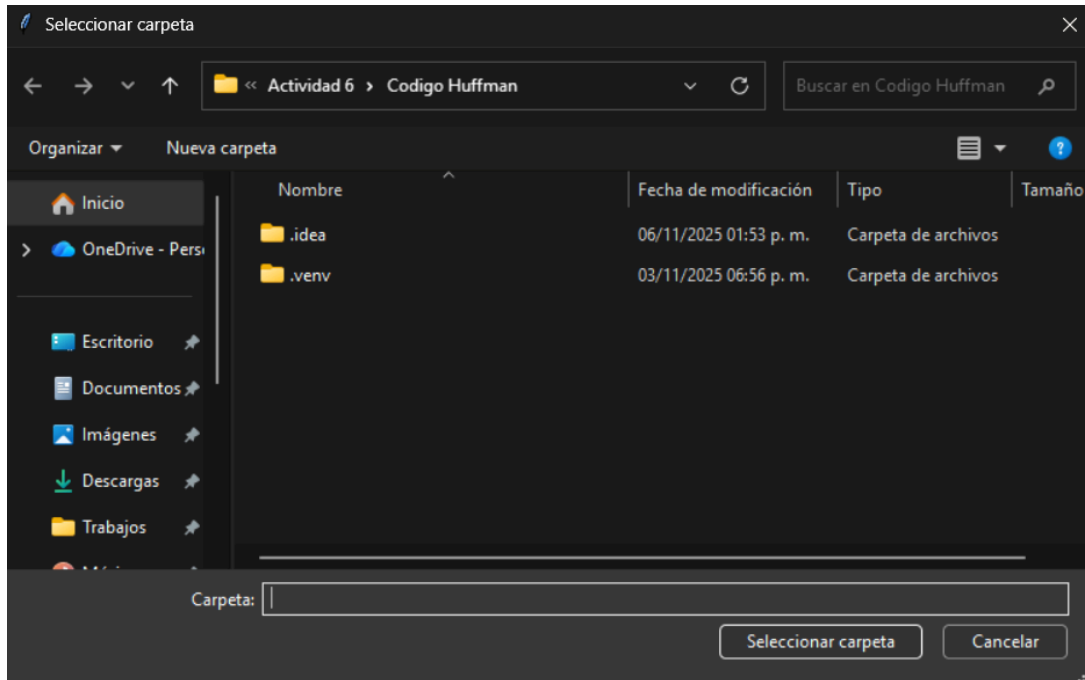
- - define el archivo_entrada como la ruta
 - - elimina lo que sea que haya estado inscrito antes en el campo de la ruta
 - - inserta la ruta en el campo correspondiente
 - - lee el archivo de la ruta
 - - borra lo que sea que haya en la preview del texto
 - - inserta el contenido del .txt en la preview del texto
 - - inserta el tamaño del archivo en la parte inferior de la ventana
5. Se deberá seleccionar una carpeta de salida que contendrá tanto el archivo comprimido (extensión .bin) como el archivo descomprimido presionando el siguiente botón.



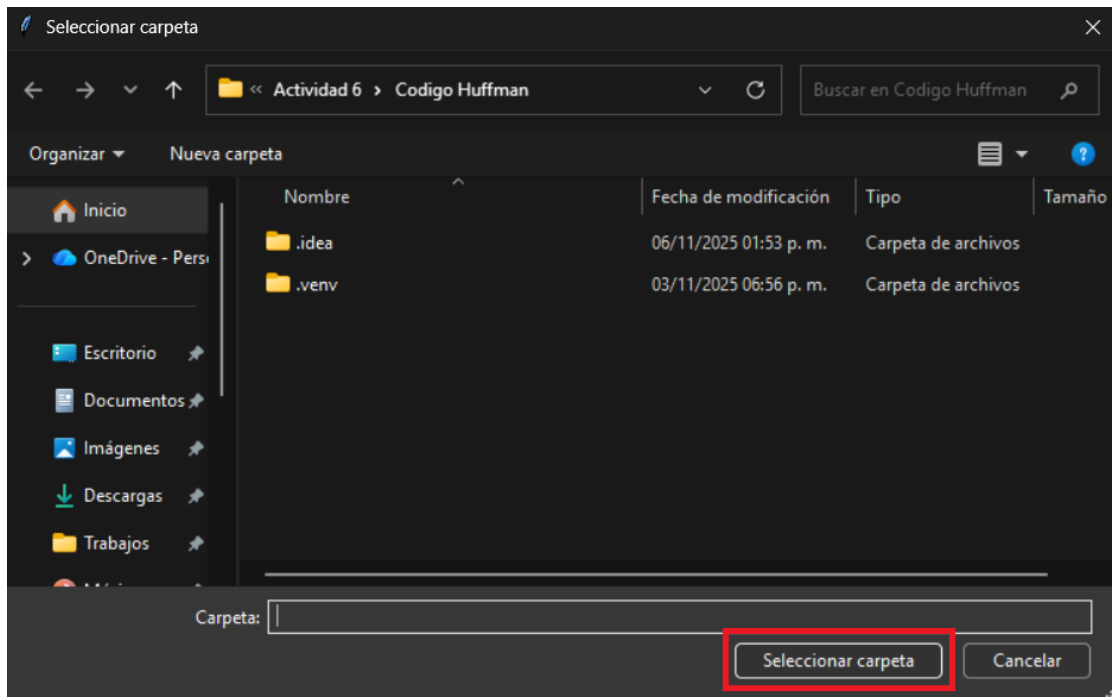
Este botón funciona bajo los lineamientos del siguiente bloque de código:

```
tk.Label(root, text="Carpeta de salida:").pack()
frame_out = tk.Frame(root)
frame_out.pack(pady=2)
self.entry_out = tk.Entry(frame_out, width=50, bg="lightblue")
self.entry_out.pack(side=tk.LEFT)
tk.Button(frame_out, text="📁", command=self.elegir_carpeta, bg="lightcyan",
          fg="black", activebackground="lightblue", activeforeground="black").pack(side=tk.LEFT)
```

6. Se abrirá una ventana que permitirá seleccionar la ruta destinada a los fines especificados en el paso 5.



7. Una vez seleccionado el directorio, se deberá presionar el botón “Seleccionar carpeta”.



Una vez seleccionada la ruta, se terminará de ejecutar también el siguiente comando:

```

tk.Label(root, text="Carpeta de salida:").pack()
frame_out = tk.Frame(root)
frame_out.pack(pady=2)
self.entry_out = tk.Entry(frame_out, width=50, bg="lightblue")
self.entry_out.pack(side=tk.LEFT)
tk.Button(frame_out, text="📁", command=self.elegir_carpeta, bg="lightcyan",
          fg="black", activebackground="lightblue", activeforeground="black").pack(side=tk.LEFT)

```

que hace lo siguiente:

```

def elegir_carpeta(self): 1 usage
    carpeta = filedialog.askdirectory()
    if carpeta:
        self.carpeta_salida = carpeta
        self.entry_out.delete(first=0, tk.END)
        self.entry_out.insert(index=0, carpeta)

```

- define la dirección de la carpeta abriendo la ventana de selección de carpeta
- si hay una carpeta entonces:
 - define carpeta_salida con la ruta definida en carpeta
 - borra lo que sea que haya en el campo de texto azul
 - inserta la ruta de la carpeta dentro del campo de texto azul

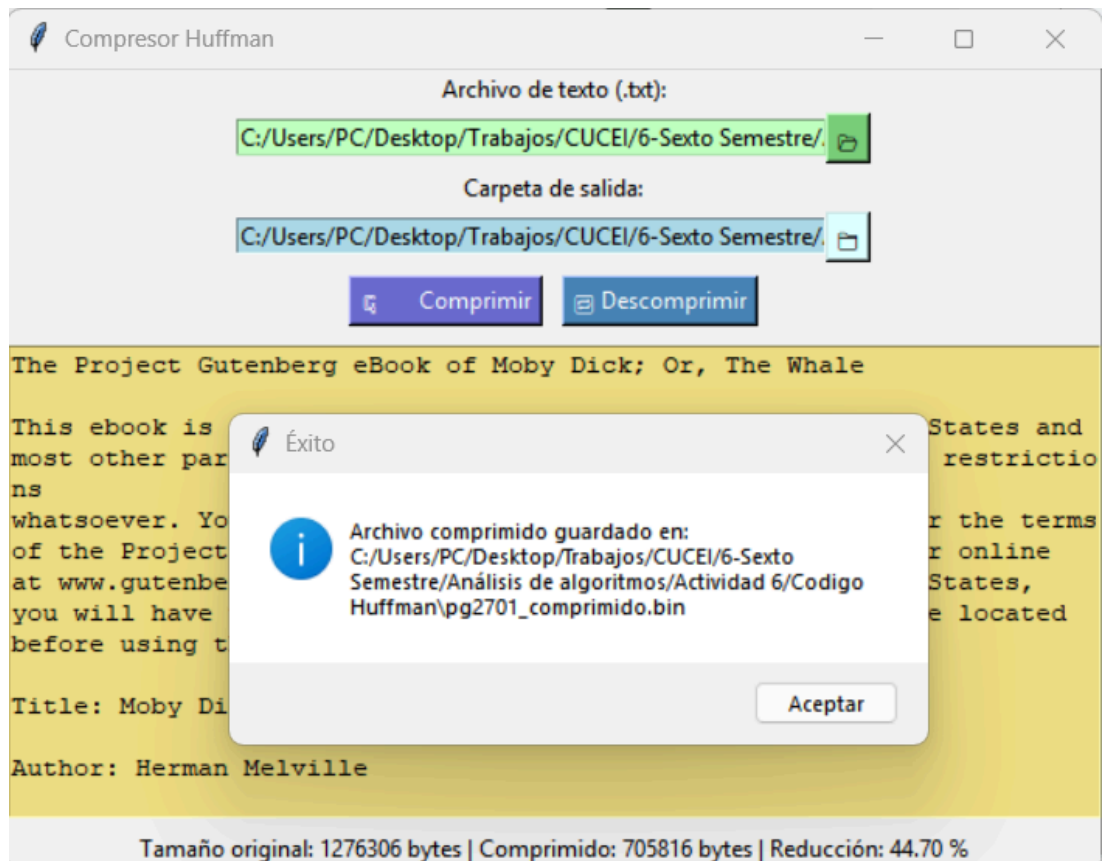
8. Una vez seleccionado el archivo a comprimir, se deberá presionar el botón "Comprimir".



El botón “Comprimir” se rige bajo el siguiente bloque de código:

```
tk.Button(frame_btn, text="Comprimir", command=self.comprimir, bg="#6a6acd",  
         fg="white", activebackground="#836fff", activeforeground="white").pack(side=tk.LEFT, padx=5)
```

9. Una vez comprimido el archivo, emergerá una ventana que confirma el éxito de la compresión, adicionalmente mostrando la ruta donde se guardó el archivo el cuál se guardará junto con el sufijo “_comprimido”, y en la ventana principal se mostrará en la parte inferior, tanto el peso original, como el comprimido en bytes, junto con el porcentaje de reducción conseguido.



Al momento de presionar el botón “Comprimir” se ejecuta el siguiente comando:

```
tk.Button(frame_btn, text="Comprimir", command=self.comprimir, bg="#6a6acd",  
         fg="white", activebackground="#836fff", activeforeground="white").pack(side=tk.LEFT, padx=5)
```

el cuál hace lo siguiente:

```
def comprimir(self): 1 usage
    if not self.archivo_entrada or not self.carpeta_salida:
        messagebox.showerror( title: "Error", message: "Selecciona archivo y carpeta de salida.")
        return
    frecuencias = Counter(self.texto)
    arbol = construir_arbol(frecuencias)
    codigos = generar_codigos(arbol)
    bits = codificar(self.texto, codigos)

    # Nombramiento del archivo
    nombre_base = os.path.splitext(os.path.basename(self.archivo_entrada))[0]
    ruta_salida = os.path.join(self.carpeta_salida, f"{nombre_base}_comprimido.bin")

    guardar_comprimido(ruta_salida, arbol, bits)

    # Cálculo de tamaño y porcentaje
    tam_original = os.path.getsize(self.archivo_entrada)
    tam_comp = os.path.getsize(ruta_salida)
    reduccion = 100 * (1 - tam_comp / tam_original) if tam_original > 0 else 0

    self.tamano.config(text=f"Tamaño original: {tam_original} bytes | Comprimido: {tam_comp} bytes | Reducción: {reduccion:.2f} %")

    messagebox.showinfo( title: "Éxito", message: f"Archivo comprimido guardado en:\n{ruta_salida}")
```

- comprueba que haya un archivo_entrada o carpeta_salida, en caso de no haber, muestra un error
- comienza a ejecutar el código Huffman
- define la frecuencia colocándole un contador al texto
- define el árbol con construir_arbol (parte del código Huffman) recibiendo las frecuencias;

```
def construir_arbol(frecuencias): 1 usage
    heap = [NodoHuffman(c, f) for c, f in frecuencias.items()]
    heapq.heapify(heap)
    while len(heap) > 1:
        n1 = heapq.heappop(heap)
        n2 = heapq.heappop(heap)
        combinado = NodoHuffman( character: None, n1.frecuencia + n2.frecuencia)
        combinado.izquierda = n1
        combinado.derecha = n2
        heapq.heappush(heap, combinado)
    return heap[0]
```

- genera los códigos en la variable “codigos” con generar_codigos recibiendo el árbol creado;

```
def generar_codigos(nodo, codigo=None, codigos=None): 3 usages
    if codigos is None:
        codigos = {}
    if codigo is None:
        codigo = bytearray()
    if nodo.caracter is not None:
        codigos[nodo.caracter] = codigo.copy()
    else:
        generar_codigos(nodo.izquierda, codigo + bytearray('0'), codigos)
        generar_codigos(nodo.derecha, codigo + bytearray('1'), codigos)
    return codigos
```

- define los bits con “codificar” recibiendo tanto el texto como los códigos;

```
def codificar(texto, codigos): 1 usage
    resultado = bytearray()
    for c in texto:
        resultado.extend(codigos[c])
    return resultado
```

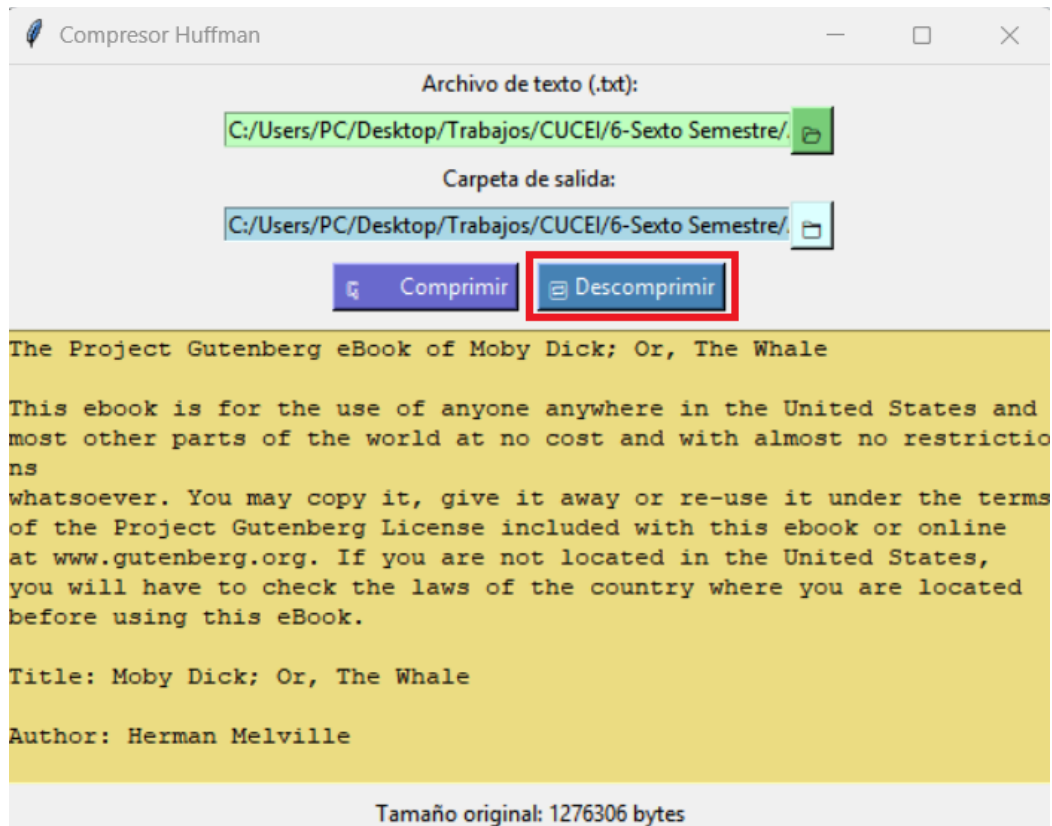
- termina la ejecución del código Huffman
- se define el nombre_base del archivo con la ruta definida anteriormente
- se define la ruta_salida con el nombre sacado de nombre_base con el sufijo “_comprimido.bin”
- se utiliza guardar_comprimido con la ruta_salida, el árbol, y los bits;

```
def guardar_comprimido(ruta, arbol, bits): 1 usage
    arbol_serializado = pickle.dumps(arbol)
    with open(ruta, 'wb') as f:
        f.write(struct.pack( fmt: 'I', *v: len(arbol_serializado)))
        f.write(arbol_serializado)
    bits.tofile(f)
```

- el cuál utilizando los datos del árbol y pickle, serializa el árbol y escribe la codificación obtenida en el .bin
- obtiene el tam_original del archivo con .os.path.getsize del archivo_entrada
- obtiene el tam_comp del archivo nuevo con .os.path.getsize de la ruta_salida
- calcula la reducción obtenida con una regla de 3 donde:
 - reducción = $100 * (1 - \text{tam_comp} / \text{tam_original})$, esto solo si el tam_original es mayor a 0, sino, será 0 la reducción.
- muestra el tamaño original, el tamaño al comprimir, y la reducción obtenida debajo de la pantalla con tkinter

- muestra un mensaje confirmando el éxito de la operación

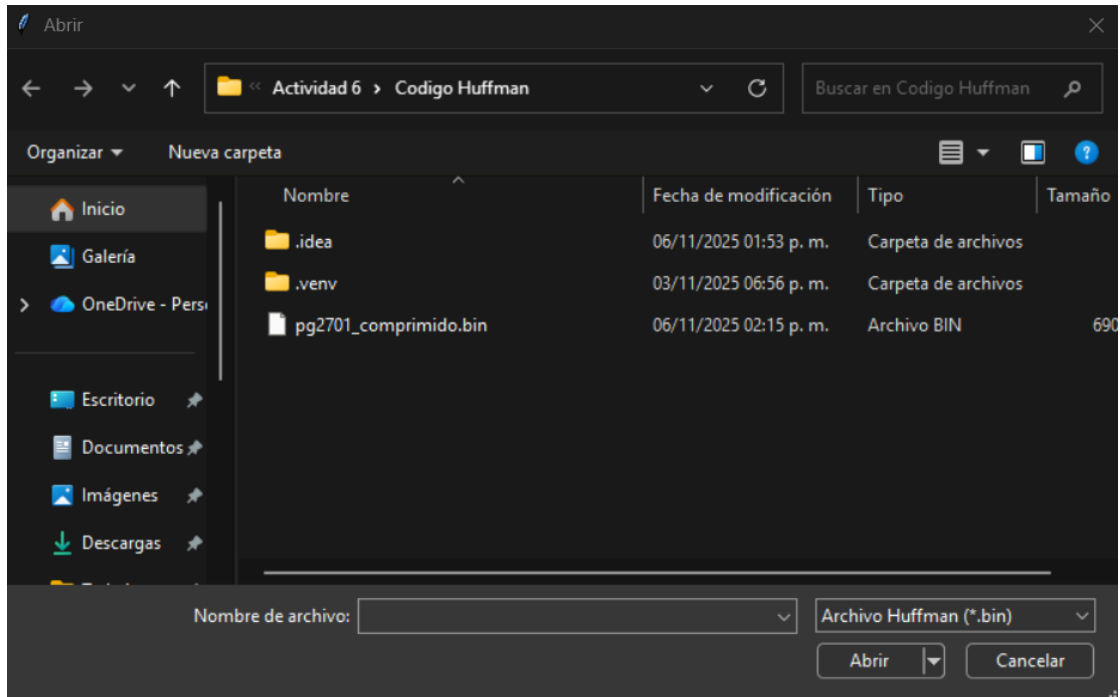
10. Con un archivo comprimido, ahora podremos darle utilidad el botón “Descomprimir”, se deberá presionar en caso de que se busque tal cosa.



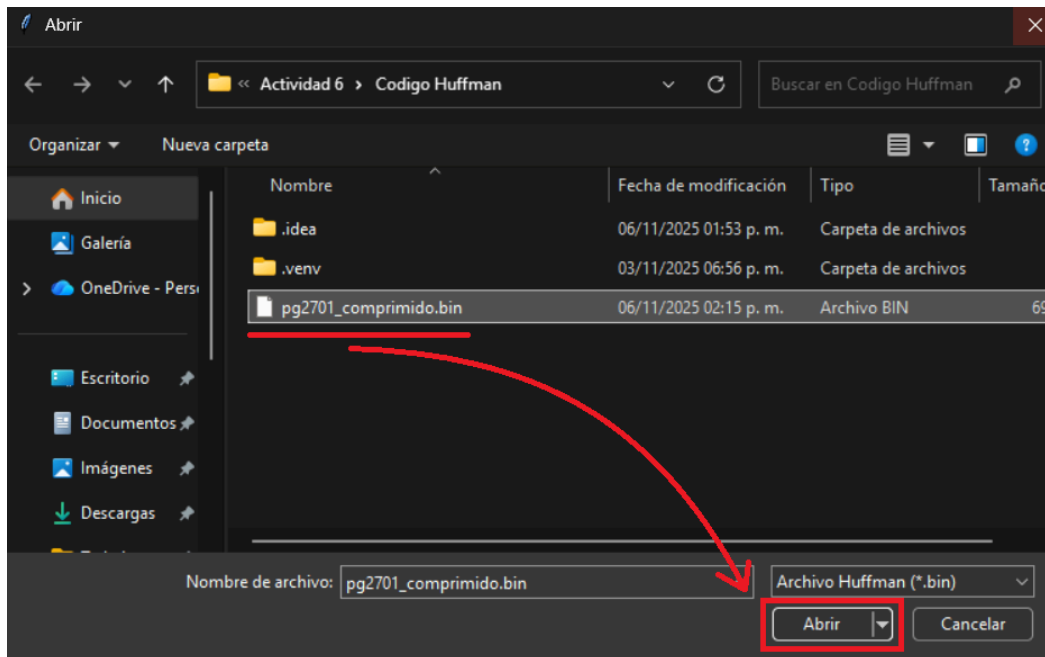
El botón “Descomprimir” se rige bajo los lineamientos del siguiente bloque de código:

```
tk.Button(frame_btn, text="🗑 Descomprimir", command=self.descomprimir, bg="#4682b4",  
         fg="white", activebackground="#5a9bd6", activeforeground="white").pack(side=tk.LEFT, padx=5)
```

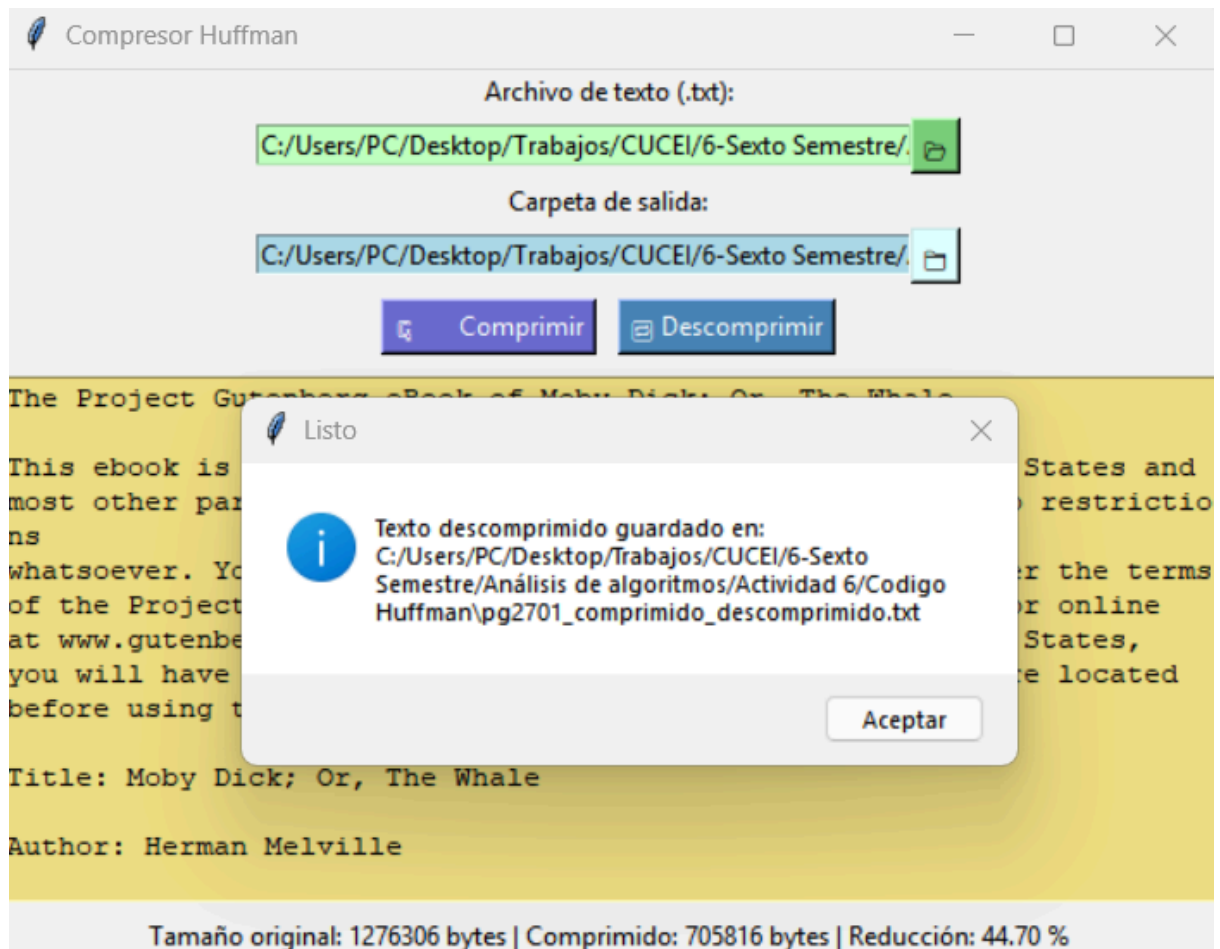
11. Se abrirá una ventana la cual permitirá seleccionar el archivo comprimido (en este caso con extensión .bin).



12. Se deberá seleccionar el archivo comprimido (con extensión .bin en este caso) y presionar el botón “Abrir”.



13. Esto provocará que el archivo se descomprima generando un nuevo archivo .txt con el sufijo “_descomprimido”, mostrando una ventana emergente que indicará el éxito de la acción y mostrando la localización del archivo descomprimido.



Al presionar el botón “Descomprimir” se ejecuta el siguiente comando:

```
tk.Button(frame_btn, text="Descomprimir", command=self.descomprimir, bg="#4682b4",
fg="white", activebackground="#5a9bd6", activeforeground="white").pack(side=tk.LEFT, padx=5)
```

el cuál hace lo siguiente:

```
def descomprimir(self): 1 usage
    ruta = filedialog.askopenfilename(filetypes=[("Archivo Huffman", "*.bin")])
    if ruta:
        arbol, bits = cargar_comprimido(ruta)
        texto = decodificar(bits, arbol)
        self.vista.delete(index1: 1.0, tk.END)
        self.vista.insert(tk.END, texto)

        nombre_base = os.path.splitext(os.path.basename(ruta))[0]

        if self.carpeta_salida:
            ruta_txt = os.path.join(self.carpeta_salida or os.path.dirname(ruta), f"{nombre_base}_descomprimido.txt")
            with open(ruta_txt, 'w', encoding='utf-8') as f:
                f.write(texto)
            messagebox.showinfo(title: "Listo", message: f"Texto descomprimido guardado en:\n{ruta_txt}")
        else:
            messagebox.showinfo(title: "Listo", message: "Texto descomprimido mostrado en pantalla.")
```

- obtiene la ruta del archivo huffman.bin
- si existe ruta, entonces:

- - define el árbol y los bits introduciendo la ruta a cargar_comprimido;

```
def cargar_comprimido(ruta): 1 usage
    with open(ruta, 'rb') as f:
        tam_arbol = struct.unpack('I', f.read(4))[0]
        arbol = pickle.loads(f.read(tam_arbol))
        bits = bytearray()
        bits.fromfile(f)
    return arbol, bits
```

- - - el cuál al abrir la ruta

- - - obtiene el tam_arbol desempaquetando con un struct.unpack con formato "I" (entero sin signo), junto a la ruta

- - - obtiene el árbol con pickle.loads leyendo el contenido del archivo de la ruta con parámetro tam_arbol

- - - define los bits como un arreglo de bits

- - - obtiene los bits del archivo

- - - y retorna el árbol junto con los bits

- - después elimina la vista previa anterior

- - introduce la vista previa con el texto descomprimido

- - define el nombre_base como el nombre del archivo seleccionado

- - si existe una carpeta de salida entonces:

- - - define la ruta_txt como la ruta anterior junto con el nombre_base y el sufijo "_descomprimido.txt"

- - - con la ruta_txt definida

- - - - escribe el texto sin comprimir

- - - muestra un mensaje de confirmación de éxito

- - si no existe una carpeta_salida entonces:

- - muestra mensaje de error.

14. Al terminar de usar, presionar la "X" situada arriba a la derecha.