6-11-2025

Técnica Voraz Huffman

Análisis de Algoritmos

Equipo: **Códigos:**

Eduardo Ramos Ochoa **304489918**

Andres Santiago Aguirre Macias **220293594**

Cesar Emmanuel Gómez Martínez **214843698**

Profesor:

JORGE ERNESTO LOPEZ ARCE DELGADO

Modulo:

D01

Centro Universitario de Ciencias Exactas e Ingenierías

1. **Del Objetivo General del Programa.**

El objeto principal del programa objeto de esta es, el Implementar y analizar el **algoritmo de Huffman**, el cual aplica la **técnica voraz (Greedy)** para la **compresión de datos** mediante la construcción de un **árbol binario óptimo** que asigna códigos binarios más cortos a los caracteres más frecuentes del texto.

2. **Descripción del programa de modo general.**

Desde una óptica general, el programa objeto de esta guía, realiza las funciones siguientes:

I. Leer un archivo de texto **(.txt)** — para el caso que ahora nos ocupa, el libro seleccionado (e incluido en los entregables) es "***The Antichrist***" (El Anticristo) de Nietzsche.

II. Una vez cargado el libro, el programa procede a calcular la **frecuencia de cada carácter**.

III. Una vez calculada la frecuencia, se procede a construir el **árbol de Huffman** a partir de las frecuencias encontradas.

IV. El programa, procede entonces a generar los **códigos binarios** correspondientes para cada carácter.

V. El programa procede con la **codificación** del texto completo con dichos códigos binarios obtenidos.

VI. **Guardar el resultado comprimido** en formato binario y generar estadísticas de compresión.

VII. (Opcional) **Decodificar** para recuperar el texto original.

El programa está listo para usarse tanto en **interfaz gráfica (GUI con Tkinter)** como desde **línea de comandos**.

3. De las Instrucciones de Uso del programa.

Paso 1. Preparación del entorno o IDE

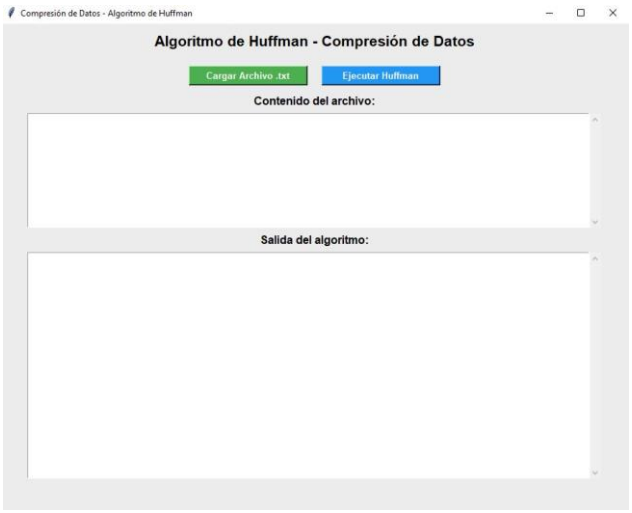
I. Abrir el IDE "**Visual Studio Code"**.

II. Asegurarse de tener instalado **Python** en la versión adecuada (versión 3.8 o superior).

III. Una vez dentro del IDE, proceder a ejecutar el programa .py.

IV. Una vez compilado el programa, proceder a seleccionar el libro "***The Antichrist***" (El Anticristo) de Nietzsche desde el directorio donde se tenga descargado / guardado.

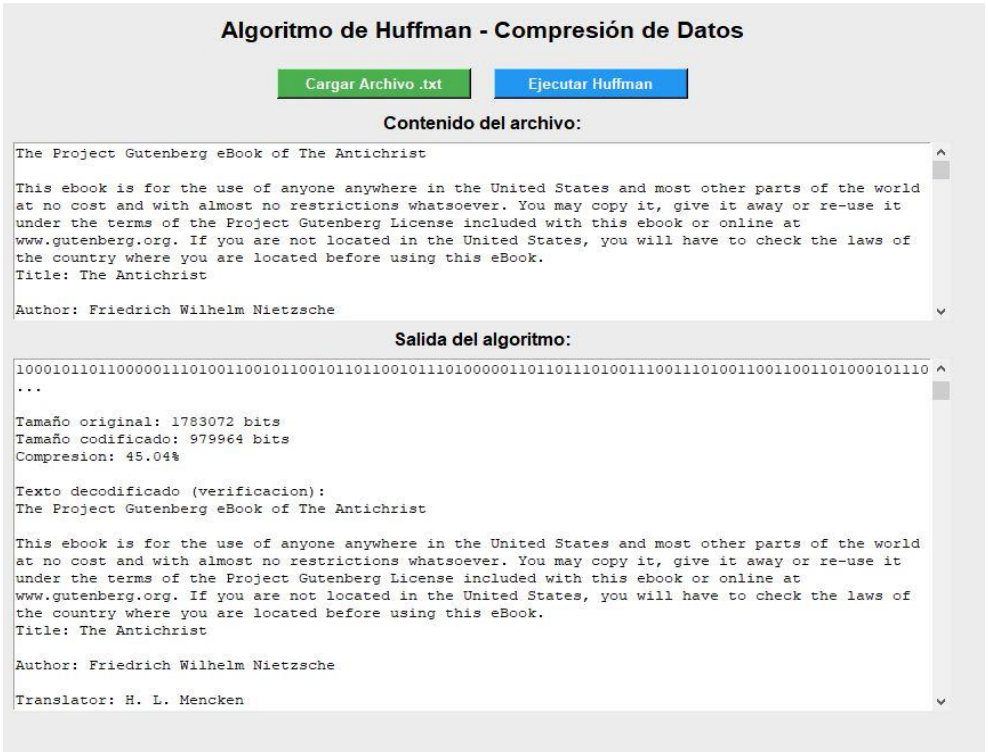
V. Una vez desplegada la interfaz gráfica de usuario, dar clic la opción "**Cargar Archivo"** como se muestra en la imagen siguiente



VI. Se abrirá una ventana para proceder a seleccionar el libro "***The Antichrist***" (El Anticristo) de Nietzsche desde el directorio donde se tenga descargado / guardado.

VII. Una vez que el libro ha sido correctamente cargado, proceder a dar clic en la opción" **Ejecutar Huffman"**

VIII. Si se han seguido correctamente las instrucciones de uso, el programa, procederá a realizar la compresión del libro, mostrando los códigos binarios asignados a cada carácter encontrado y, mostrará en la propia GUI, el porcentaje de compresión obtenido, tal cual se aprecia en la imagen siguiente.



4. Explicación por Bloques del Código objeto de esta guía.

a) Librerías que se han empleado.

heapq → implementa una cola de prioridad (min-heap), ideal para seleccionar los nodos con menor frecuencia.
os → se usa para manejar nombres y rutas de archivos.
tkinter → permite crear la interfaz gráfica del programa.
filedialog, messagebox y scrolledtext → facilitan la interacción con el usuario, mostrando ventanas de diálogo, alertas y cuadros de texto con desplazamiento.

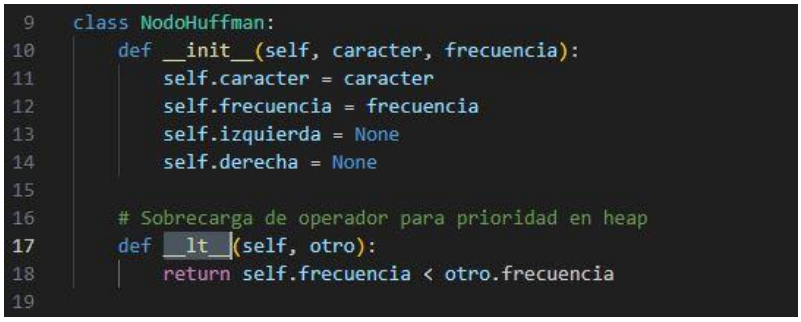
```
1  import heapq
2  import os
3  from tkinter import *
4  from tkinter import filedialog, messagebox, scrolledtext
5
```

b) Clase “NodoHuffman”

Esta clase define la estructura de los nodos del árbol de Huffman. Cada nodo puede ser:

I. Una hoja, que contiene un carácter y su frecuencia.
II. Un nodo interno, que combina las frecuencias de dos subnodos.

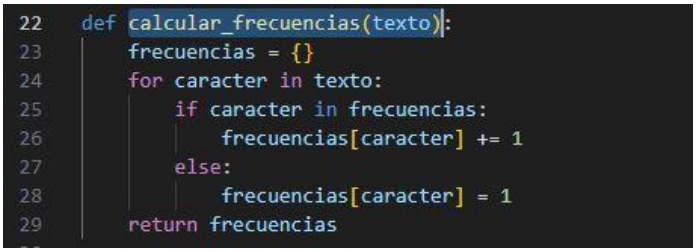
El método **__lt__** redefine el operador “menor que” (<), para que los nodos puedan ser ordenados automáticamente dentro del heap de prioridades según su frecuencia.



```
9   class NodoHuffman:
10      def __init__(self, caracter, frecuencia):
11          self.caracter = caracter
12          self.frecuencia = frecuencia
13          self.izquierda = None
14          self.derecha = None
15
16      # Sobrecarga de operador para prioridad en heap
17      def __lt__(self, otro):
18          return self.frecuencia < otro.frecuencia
19
```

c) Función calcular_frecuencias(texto)

El objetivo de esta función es, recorre el texto del archivo de texto que se cargue en el programa y **cuenta cuántas veces aparece cada carácter**.



```
22  def calcular_frecuencias(texto):
23      frecuencias = {}
24      for caracter in texto:
25          if caracter in frecuencias:
26              frecuencias[caracter] += 1
27          else:
28              frecuencias[caracter] = 1
29      return frecuencias
```

d) Función construir_arbol(frecuencias):

Construye el árbol binario de Huffman de la siguiente manera:

I. Convierte cada carácter en un nodo (NodoHuffman).
II. Inserta todos los nodos en un heap ordenado por frecuencia.
III. Repite:
   - Extrae los dos nodos con menor frecuencia.
   - Los fusiona en un nuevo nodo con la suma de ambas frecuencias.
   - Vuelve a insertar el nodo combinado al heap.
IV. Cuando queda un solo nodo, se obtiene la raíz del árbol de Huffman.

```python
def construir_arbol(frecuencias):
    heap = [NodoHuffman(c, f) for c, f in frecuencias.items()]
    heapq.heapify(heap)

    while len(heap) > 1:
        nodo1 = heapq.heappop(heap)
        nodo2 = heapq.heappop(heap)

        nodo_fusion = NodoHuffman(None, nodo1.frecuencia + nodo2.frecuencia)
        nodo_fusion.izquierda = nodo1
        nodo_fusion.derecha = nodo2

        heapq.heappush(heap, nodo_fusion)

    return heap[0]  # raíz del árbol
```

e) Función generar_codigos:

```python
def generar_codigos(nodo, codigo_actual="", codigos={}):
    if nodo is None:
        return

    if nodo.caracter is not None:
        codigos[nodo.caracter] = codigo_actual
        return

    generar_codigos(nodo.izquierda, codigo_actual + "0", codigos)
    generar_codigos(nodo.derecha, codigo_actual + "1", codigos)

    return codigos
```

Genera los códigos binarios para cada carácter:

- “0” si el recorrido va hacia la izquierda.
- “1” si va hacia la derecha.

Cada carácter termina con un código único (sin prefijos repetidos).

f) Función codificar_texto

```
63  def codificar_texto(texto, codigos):
64      return "".join(codigos[c] for c in texto)
65
```

Reemplaza cada carácter del texto por su código binario correspondiente, generando la secuencia comprimida.

g) Función decodificar_texto:

Recorre el árbol de Huffman según los bits del texto codificado y reconstruye el texto original.

Cada vez que se alcanza una hoja (carácter válido), se añade al texto decodificado y se reinicia desde la raíz.

```
def decodificar_texto(codigo_binario, nodo_raiz):
    texto_decodificado = ""
    nodo_actual = nodo_raiz

    for bit in codigo_binario:
        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 = nodo_raiz

    return texto_decodificado
```

h) Función cargar_archivo:

Su funcionamiento es el de abrir un cuadro de dialogo (en la GUI) para proceder a seleccionar el archivo “.txt”.

Lee su contenido y lo muestra en el cuadro de texto de entrada de la interfaz gráfica.
También almacena la ruta del archivo en la variable global ruta_archivo.

```
85  def cargar_archivo():
86      ruta = filedialog.askopenfilename(filetypes=[("Archivos de texto", "*.txt")])
87      if not ruta:
88          return
89
90      with open(ruta, "r", encoding="utf-8") as archivo:
91          texto = archivo.read()
92
93      entrada_texto.delete(1.0, END)
94      entrada_texto.insert(END, texto)
95
96      global ruta_archivo
97      ruta_archivo = ruta
98      messagebox.showinfo("Archivo cargado", f"Archivo '{os.path.basename(ruta)}' cargado correctamente.")
```

i) Función ejecutar_huffman

```python
def ejecutar_huffman():
    texto = entrada_texto.get(1.0, END).rstrip("\n")
    if not texto:
        messagebox.showwarning("Advertencia", "Debe cargar o escribir un texto para analizar.")
        return

    # Calcular frecuencias
    frecuencias = calcular_frecuencias(texto)

    # Construir árbol
    raiz = construir_arbol(frecuencias)

    # Generar códigos
    codigos = generar_codigos(raiz)

    # Codificar
    texto_codificado = codificar_texto(texto, codigos)

    # Decodificar para verificación
    texto_decodificado = decodificar_texto(texto_codificado, raiz)

    # Calcular tamaños
    tam_original = len(texto.encode('utf-8')) * 8  # bits
    tam_codificado = len(texto_codificado)

    compresion = (1 - tam_codificado / tam_original) * 100

    # Mostrar resultados
    salida_texto.delete(1.0, END)
    salida_texto.insert(END, "=== Resultados del Algoritmo de Huffman ===\n\n")
    salida_texto.insert(END, f"Frecuencias de caracteres:\n{frecuencias}\n\n")
    salida_texto.insert(END, f"Codigos generados:\n{codigos}\n\n")
    salida_texto.insert(END, f"Texto codificado:\n{texto_codificado[:300]}...\n\n")
    salida_texto.insert(END, f"Tamaño original: {tam_original} bits\n")
    salida_texto.insert(END, f"Tamaño codificado: {tam_codificado} bits\n")
    salida_texto.insert(END, f"Compresion: {compresion:.2f}%\n\n")
    salida_texto.insert(END, "Texto decodificado (verificacion):\n")
    salida_texto.insert(END, texto_decodificado)

```

Es la función principal que se ejecuta al presionar el botón **“Ejecutar Huffman”**.

Realiza todas las etapas del algoritmo:

- **I.** Obtiene el texto desde la interfaz.
- **II.** Calcula las frecuencias.
- **III.** Construye el árbol.
- **IV.** Genera los códigos.
- **V.** Codifica y decodifica el texto.
- **VI.** Calcula la **tasa de compresión**.
- **VII.** Muestra todos los resultados en el área de salida (frecuencias, códigos, texto codificado, verificación, etc.).

j) Interfaz gráfica (Tkinter)

```
# Interfaz grafica (Tkinter)

ventana = Tk()
ventana.title("Compresión de Datos - Algoritmo de Huffman")
ventana.geometry("900x700")
ventana.configure(bg="#ECECEC")

titulo = Label(ventana, text="Algoritmo de Huffman - Compresión de Datos", font=("Arial", 16, "bold"), bg="#ECECEC")
titulo.pack(pady=10)

frame_botones = Frame(ventana, bg="#ECECEC")
frame_botones.pack(pady=10)

btn_cargar = Button(frame_botones, text="Cargar Archivo .txt", command=cargar_archivo, width=20, bg="#4CAF50", fg="white", font=("Arial", 10, "bold"))
btn_cargar.grid(row=0, column=0, padx=10)

btn_ejecutar = Button(frame_botones, text="Ejecutar Huffman", command=ejecutar_huffman, width=20, bg="#2196F3", fg="white", font=("Arial", 10, "bold"))
btn_ejecutar.grid(row=0, column=1, padx=10)

Label(ventana, text="Contenido del archivo:", bg="#ECECEC", font=("Arial", 12, "bold")).pack()
entrada_texto = scrolledtext.ScrolledText(ventana, width=100, height=10, wrap=WORD)
entrada_texto.pack(pady=5)

Label(ventana, text="Salida del algoritmo:", bg="#ECECEC", font=("Arial", 12, "bold")).pack()
salida_texto = scrolledtext.ScrolledText(ventana, width=100, height=20, wrap=WORD)
salida_texto.pack(pady=5)

ventana.mainloop()

```

Configura y lanza la interfaz gráfica de usuario o GUI.