



ALGORITMO DE HUFFMAN

ANÁLISIS DE ALGORITMOS



Herrera Coss y León Andrea Sofia

Universidad de Guadalajara
C.U. de Ciencias Exactas e Ingenierías
Ingeniería en Computación

Abril 2024

COMPRESOR DE ARCHIVOS CON ALGORITMO DE HUFFMAN

Introducción

En el ámbito digital actual, la compresión de archivos es esencial para optimizar el almacenamiento y la transmisión de datos. Uno de los algoritmos más utilizados para este propósito es el algoritmo de Huffman, desarrollado por David A. Huffman en 1952. Este algoritmo asigna códigos de longitud variable a los caracteres del archivo original, de manera que los caracteres más frecuentes tengan códigos más cortos, lo que resulta en una compresión eficiente.

Este documento presenta una implementación de un compresor de archivos utilizando el algoritmo de Huffman. Se analiza tanto el frontend como el backend del código, describiendo sus funcionalidades y características.

Objetivo

El objetivo de este proyecto es desarrollar un compresor de archivos que utilice el algoritmo de Huffman para comprimir y descomprimir archivos de manera eficiente. Se busca proporcionar una interfaz de usuario intuitiva para que los usuarios puedan comprimir y descomprimir archivos de forma sencilla, así como implementar el algoritmo de Huffman en el backend para garantizar una compresión y descompresión precisas.

Desarrollo

El desarrollo del compresor de archivos con el algoritmo de Huffman consta de dos partes principales: el frontend y el backend.

- **Frontend:**
El frontend del compresor de archivos es responsable de la interfaz de usuario y la interacción con el usuario. Está desarrollado utilizando la biblioteca Tkinter de Python, que proporciona herramientas para crear interfaces gráficas de usuario (GUI). En el frontend, los usuarios pueden seleccionar un archivo para comprimir o descomprimir utilizando un explorador de archivos. Una vez seleccionado el archivo, se muestra la frecuencia de cada carácter en el archivo, así como una representación gráfica del árbol de Huffman correspondiente. Además, los usuarios pueden iniciar el proceso de compresión o descompresión con los botones correspondientes.
- **Backend**
El backend del compresor de archivos implementa la lógica del algoritmo de Huffman para comprimir y descomprimir archivos. Está desarrollado utilizando funciones y clases de Python. Las operaciones principales del backend incluyen la construcción del árbol de Huffman a partir de las frecuencias de los caracteres, la generación de la tabla de códigos Huffman, la compresión del archivo original utilizando los códigos Huffman

generados y la descompresión del archivo comprimido utilizando la tabla de códigos Huffman y el árbol de Huffman.

Código

- Frontend

```
• import tkinter as tk
• from tkinter import filedialog
• from collections import Counter
• from backend import comprimir, descomprimir, construir_arbol_huffman
•
• class HuffmanFrontend:
•     def __init__(self, master):
•         self.master = master
•         self.entradaCadena = entradaCadena
•         self.entradaFrecuencia = entradaFrecuencia
•         self.entradaArbol = entradaArbol
•         master.title("Compresor de Archivos")
•
•         self.examinar_button = tk.Button(master, text="Examinar",
• command=self.examinar_archivo)
•         self.examinar_button.grid(row=4, column=1, padx=(10, 2), pady=20,
• sticky="ew")
•
•         self.comprimir_button = tk.Button(master, text="Comprimir",
• command=self.comprimir_archivo, state=tk.DISABLED)
•         self.comprimir_button.grid(row=4, column=2, padx=2, pady=20,
• sticky="ew")
•
•         self.descomprimir_button = tk.Button(master, text="Descomprimir",
• command=self.descomprimir_archivo, state=tk.DISABLED)
•         self.descomprimir_button.grid(row=4, column=3, padx=2, pady=20,
• sticky="ew")
•
•     def examinar_archivo(self):
•         file_path = filedialog.askopenfilename()
•         if file_path:
•             self.archivo = file_path
•             self.comprimir_button.config(state=tk.NORMAL)
•             self.descomprimir_button.config(state=tk.NORMAL)
•             with open(self.archivo, 'r') as file:
•                 contenido = file.read()
•                 self.entradaCadena.delete('1.0', tk.END) # Limpiar el campo de
entrada
•                 self.entradaCadena.insert(tk.END, contenido) # Insertar
contenido del archivo
```

```

•         self.calcular_frecuencia()
•
•     def calcular_frecuencia(self):
•         with open(self.archivo, 'r') as file:
•             contenido = file.read()
•             self.frecuencias = Counter(contenido)
•             self.entradaFrecuencia.delete('1.0', tk.END) # Limpiar el campo de
entrada
•             for caracter, frecuencia in self.frecuencias.items():
•                 self.entradaFrecuencia.insert(tk.END, f"{caracter}:
{frecuencia}\n") # Insertar frecuencia
•             self.mostrar_arbol_huffman()
•
•     def mostrar_arbol_huffman(self):
•         arbol_huffman = construir_arbol_huffman(self.frecuencias)
•         self.entradaArbol.delete('1.0', tk.END) # Limpiar el campo de entrada
•         self.dibujar_arbol(arbol_huffman)
•
•     def dibujar_arbol(self, nodo, nivel=0):
•         if nodo:
•             self.entradaArbol.insert(tk.END, "    " * nivel)
•             self.entradaArbol.insert(tk.END, f"({nodo.frecuencia})\n")
•             self.dibujar_arbol(nodo.izquierda, nivel + 1)
•             self.dibujar_arbol(nodo.derecha, nivel + 1)
•
•     def comprimir_archivo(self):
•         output_file = self.archivo + '.huf'
•         tabla_codigos = comprimir(self.archivo, output_file)
•         print("Archivo comprimido correctamente.")
•
•     def descomprimir_archivo(self):
•         if self.archivo.endswith('.huf'):
•             output_file = self.archivo[:-4] # Eliminar la extensión .huf
•             descomprimir(self.archivo, output_file)
•             print("Archivo descomprimido correctamente.")
•         else:
•             print("El archivo seleccionado no es un archivo comprimido.")
•
• root = tk.Tk()
• root.title("Algoritmo de Huffman")
• root.geometry("800x550")
• root.resizable(0, 0)
•
• Titulo = tk.Label(root, text="ALGORITMO DE HUFFMAN")
• Titulo.grid(row=0, column=0, columnspan=6, padx=30, pady=20, sticky="nsew")
•
• cadena = tk.Label(root, text="    Cadena: ")

```

- `cadena.grid(row=1, column=0, padx=10, pady=10, sticky="w")`
- `entradaCadena = tk.Text(root, width=80, height=3, wrap=tk.WORD) # Ajustar el ancho y alto del widget entradaCadena`
- `entradaCadena.grid(row=1, column=1, columnspan=5, padx=5, pady=10, sticky="w")`
-
- `frecuencia = tk.Label(root, text=" Frecuencia: ")`
- `frecuencia.grid(row=2, column=0, padx=10, pady=10, sticky="w")`
- `entradaFrecuencia = tk.Text(root, width=80, height=8, wrap=tk.WORD) # Ajustar el ancho y alto del widget entradaCadena`
- `entradaFrecuencia.grid(row=2, column=1, columnspan=5, padx=5, pady=10, sticky="w")`
-
- `arbol = tk.Label(root, text=" Arbol: ")`
- `arbol.grid(row=3, column=0, padx=10, pady=10, sticky="w")`
- `entradaArbol = tk.Text(root, width=80, height=11, wrap=tk.WORD) # Ajustar el ancho y alto del widget entradaCadena`
- `entradaArbol.grid(row=3, column=1, columnspan=5, padx=5, pady=10, sticky="w")`
-
- `app = HuffmanFrontend(root)`
- `root.mainloop()`
-

• Backend

- `from collections import Counter`
- `import heapq`
- `import os`
-
- `class NodoHuffman:`
- `def __init__(self, caracter, frecuencia):`
- `self.caracter = caracter`
- `self.frecuencia = frecuencia`
- `self.izquierda = None`
- `self.derecha = None`
-
- `def __lt__(self, otro):`
- `return self.frecuencia < otro.frecuencia`
-
- `def construir_arbol_huffman(frecuencias):`
- `cola_prioridad = [NodoHuffman(caracter, frecuencia) for caracter, frecuencia in frecuencias.items()]`
- `heapq.heapify(cola_prioridad)`
- `while len(cola_prioridad) > 1:`
- `izquierda = heapq.heappop(cola_prioridad)`
- `derecha = heapq.heappop(cola_prioridad)`
- `nodo_padre = NodoHuffman(None, izquierda.frecuencia + derecha.frecuencia)`

```

•     nodo_padre.izquierda = izquierda
•     nodo_padre.derecha = derecha
•     heapq.heappush cola_prioridad, nodo_padre)
•     return cola_prioridad[0]
•
•
• def generar_tabla_codigos(raiz, codigo_actual='', tabla_codigos={}):
•     if raiz is not None:
•         if raiz.caracter is not None:
•             tabla_codigos[raiz.caracter] = codigo_actual
•             generar_tabla_codigos(raiz.izquierda, codigo_actual + '0',
tabla_codigos)
•             generar_tabla_codigos(raiz.derecha, codigo_actual + '1', tabla_codigos)
•         return tabla_codigos
•
•
• def comprimir_archivo(input_file, output_file, tabla_codigos):
•     with open(input_file, 'r') as file:
•         contenido = file.read()
•
•
•     bits = ''
•     for caracter in contenido:
•         bits += tabla_codigos[caracter]
•
•
•     padding = 8 - len(bits) % 8
•     bits += padding * '0'
•
•
•     bytes_comprimidos = bytearray()
•     for i in range(0, len(bits), 8):
•         byte = bits[i:i+8]
•         bytes_comprimidos.append(int(byte, 2))
•
•
•     with open(output_file, 'wb') as file:
•         file.write(bytes_comprimidos)
•
•
• def descomprimir_archivo(input_file, output_file, raiz):
•     bits = ''
•     with open(input_file, 'rb') as file:
•         byte = file.read(1)
•         while byte:
•             bits += bin(ord(byte))[2:].rjust(8, '0')
•             byte = file.read(1)
•
•
•     nodo_actual = raiz
•     contenido_descomprimido = ''
•     for bit in bits:
•         if bit == '0':
•             nodo_actual = nodo_actual.izquierda
•         else:

```

```

•         nodo_actual = nodo_actual.derecha
•         if nodo_actual.caracter is not None:
•             contenido_descomprimido += nodo_actual.caracter
•             nodo_actual = raiz
•
•         with open(output_file, 'w') as file:
•             file.write(contenido_descomprimido)
•
• def comprimir(input_file, output_file):
•     with open(input_file, 'r') as file:
•         contenido = file.read()
•         frecuencias = Counter(contenido)
•         arbol_huffman = construir_arbol_huffman(frecuencias)
•         tabla_codigos = generar_tabla_codigos(arbol_huffman)
•         comprimir_archivo(input_file, output_file, tabla_codigos)
•     return tabla_codigos
•
• def descomprimir(input_file, output_file, tabla_codigos):
•     raiz = construir_arbol_huffman({caracter: frecuencia for frecuencia,
•                                     caracter in tabla_codigos.items()})
•     descomprimir_archivo(input_file, output_file, raiz)
•
• # Ejemplo de uso:
• input_file = r'c:\Users\sofia\Desktop\algoritmos\comprimir\archivo.txt'
• output_file_comprimido = 'archivo_comprimido.bin'
• output_file_descomprimido = 'archivo_descomprimido.txt'
•
• tabla_codigos = comprimir(input_file, output_file_comprimido)
• descomprimir(output_file_comprimido, output_file_descomprimido, tabla_codigos)
•

```

Conclusiones

El algoritmo de Huffman y su implementación en un compresor de archivos proporcionan una solución efectiva para la optimización del almacenamiento y la transmisión de datos en aplicaciones informáticas y de comunicación. El compresor de archivos desarrollado en este proyecto ofrece una interfaz gráfica intuitiva para los usuarios, facilitando la compresión y descompresión de archivos de manera rápida y sencilla. Además, la implementación del algoritmo de Huffman en el backend garantiza una compresión eficiente y una descompresión precisa de los archivos.

Referencias

- Anonimo. (Sin fecha). Algoritmo de Huffman:
<https://joselu.webs.uvigo.es/material/Algoritmo%20de%20Huffman.pdf>
- Sznajdleder, Pablo A. (Sin fecha). Algoritmos a fondo:
<https://libroweb.alfaomega.com.mx/book/393/free/data/Capitulos/cap16.pdf>