



CUCEI

CENTRO UNIVERSITARIO DE
CIENCIAS EXACTAS E INGENIERÍAS

UNIVERSIDAD DE GUADALAJARA
CENTRO UNIVERSITARIO DE CIENCIAS EXACTAS E INGENIERÍAS

Análisis de algoritmos

Mtro. Jorge Ernesto Lopez Arce Delgado

Act. 5: Técnica Voraz Huffman

Integrantes:

Braulio Hurtado Escoto 220426225

Jorge Daniel Hernández Reyes 220027797

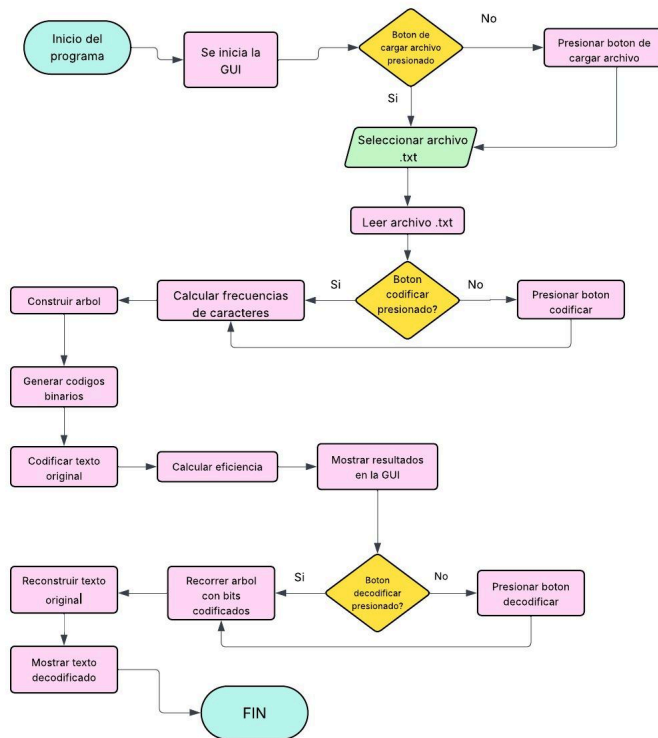
Juan Pablo Solis Regin 220468416

R&R

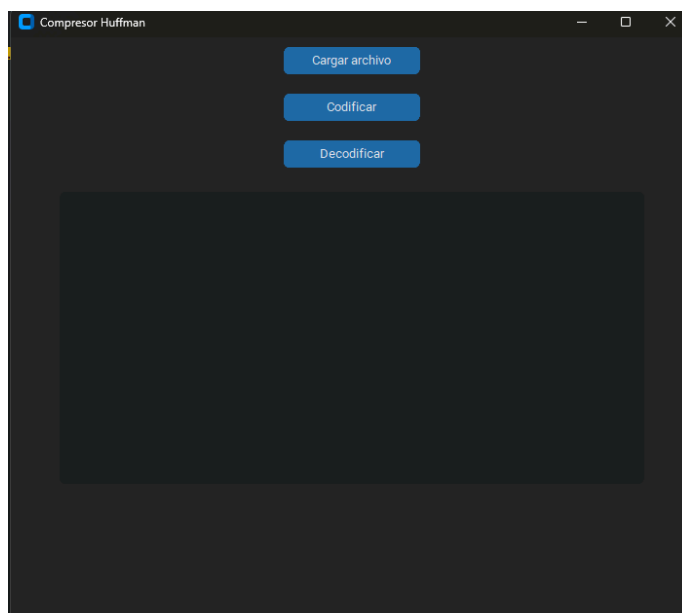
Actividades	Braulio Hurtado Escoto	Jorge Daniel Hernández Reyes	Juan Pablo Solis Regin
R&R (Roles y Responsabilidades)	R A	R C	R C
Diagrama de Flujo del Algoritmo	A	R	R C
Front end (GUI solo visual)	R A	C	C
Pseudocódigo ó Código Base	R A	C	R
Libro elegido a comprimir	R A	R C	R C
Archivo del equipo (<Nombre>.txt)	C A	C	R
Función <code>calcular_frecuencias</code>	R A	C	A
Función <code>decodificar_texto</code>	A C	R	R
Cálculo de Eficiencia	A C	R	C

R: Responsable (Quien ejecuta la tarea). **A: Aprobar** (Quien tiene la autoridad final y aprueba el entregable). **C: Consultado** (Debe ser consultado antes de la finalización). **I: Informado** (Debe ser informado una vez que la tarea ha sido completada).

Diagrama de Flujo del algoritmo.



Front end (GUI solo visual)



Pseudocódigo ó Código Base

```
import heapq

from collections import Counter

class Node:

    def __init__(self, char, freq):

        self.char = char

        self.freq = freq

        self.left = None

        self.right = None

    def __lt__(self, other):

        return self.freq < other.freq

def calcular_frecuencias(texto):

    return dict(Counter(texto))

def construir_arbol(frecuencias):

    heap = [Node(char, freq) for char, freq in frecuencias.items()]

    heapq.heapify(heap)

    while len(heap) > 1:

        nodo1 = heapq.heappop(heap)

        nodo2 = heapq.heappop(heap)
```

```

        nuevo = Node(None, nodo1.freq + nodo2.freq)

        nuevo.left = nodo1

        nuevo.right = nodo2

        heapq.heappush(heap, nuevo)

    return heap[0]

def generar_codigos(nodo, codigo_actual="", codigos={}):

    if nodo is None:

        return

    if nodo.char is not None:

        codigos[nodo.char] = codigo_actual

    generar_codigos(nodo.left, codigo_actual + "0", codigos)

    generar_codigos(nodo.right, codigo_actual + "1", codigos)

    return codigos

def codificar_texto(texto, codigos):

    return "".join(codigos[char] for char in texto)

def decodificar_texto(codificado, raiz):

    resultado = ""

    nodo = raiz

    for bit in codificado:

```

```
nodo = nodo.left if bit == "0" else nodo.right

if nodo.char is not None:

    resultado += nodo.char

    nodo = raiz

return resultado


def calcular_eficiencia(original, codificado):

    original_bits = len(original) * 8

    comprimido_bits = len(codificado)

    if original_bits == 0:

        return 0

    return (1 - (comprimido_bits / original_bits)) * 100
```

Libro elegido a comprimir

The great Gatsby