UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
FACULTAD DE INGENIERÍA
ESCUELA DE CIENCIAS
DEPARTAMENTO DE MATEMÁTICA
AUX. BRAYAN HAMLLELO PRADO MARROQUIN

MC2-P

Proyecto. Unico

ASPECTOS	PUNTEO
PRESENTACION	/ 10
EJERCICIOS RESUELTOS	/ 70
EJERCICIOS CALIFICADOS	/ 20
TOTAL	/100

Nombre: Luis Eduardo Guzmán Monterroso,
Evelyn Priscilla Carrillo Gutierrez
Carné: 202406657, 201708918
Sección: P
Fecha: 25-04-2025

INTRODUCCION

Este proyecto tiene como objetivo crear una aplicación web que permita encriptar y desencriptar frases utilizando el algoritmo de Huffman, un método clásico de compresión de texto basado en árboles binarios. La idea principal del algoritmo es asignar códigos más cortos a los caracteres que más se repiten y códigos más largos a los menos frecuentes, logrando así una codificación eficiente del mensaje.

La aplicación fue desarrollada utilizando una arquitectura de dos capas: el frontend está hecho con React, que permite una interfaz amigable para el usuario, mientras que el backend fue construido en Python usando Flask, encargado de aplicar el algoritmo de Huffman, generar el árbol binario, y devolver los resultados al frontend. Además, se utilizó Graphviz para generar y exportar una imagen del árbol en formato PDF.

Este proyecto nos permitió poner en práctica conceptos clave como estructuras de datos (árboles binarios), algoritmos de compresión, comunicación entre frontend y backend mediante API, y la integración de herramientas externas para visualizar estructuras. También reforzamos nuestras habilidades de trabajo en equipo y resolución de problemas al desarrollar una solución funcional y útil.

OBJETIVOS

1. Implementar desde cero el algoritmo de Huffman en Python para codificar y decodificar mensajes.
2. Diseñar una interfaz de usuario en React que permita ingresar frases, mostrar resultados y visualizar el árbol de Huffman.

MARCO TEÓRICO

Un árbol binario es una estructura de datos en la cual cada nodo tiene como máximo dos hijos: uno a la izquierda y otro a la derecha. Esta estructura es ampliamente utilizada en informática para representar jerarquías, realizar búsquedas eficientes, y en este caso, para codificar información. Los árboles permiten organizar los datos de manera que ciertos algoritmos, como el de Huffman, puedan utilizarlos para generar códigos únicos y sin ambigüedad.

El algoritmo de Huffman es un método de compresión de datos sin pérdida, desarrollado por David A. Huffman en 1952. Este algoritmo utiliza un árbol binario para asignar a cada carácter de un mensaje una secuencia de bits única, más corta para los caracteres más frecuentes y más larga para los menos frecuentes. El árbol se construye uniendo los nodos

con menor frecuencia de aparición hasta formar una sola estructura que representa todo el mensaje. Esto permite reducir el tamaño total del mensaje al codificarlo.

En este proyecto, la encriptación se refiere al proceso de convertir una frase normal en una secuencia de ceros y unos utilizando el árbol de Huffman. Esta secuencia puede ser transmitida o almacenada de manera más compacta. La desencriptación, por otro lado, es el proceso inverso: se toma la secuencia codificada y, utilizando el árbol de Huffman, se reconstruye el mensaje original.

ALGORITMO DE LA SOLUCIÓN E IMPLEMENTACIÓN

```
from flask import Flask, request, jsonify
import heapq
import os
from graphviz import Digraph
from flask_cors import CORS
from flask import send_from_directory

app = Flask(__name__)
CORS(app, resources={r"/encriptar": {"origins": "http://localhost:3000"}}) #habilita CORS para todas las rutas

class NodoHuffman:
    def __init__(self, caracter, frecuencia):
        self.caracter = caracter #caracter
        self.frecuencia = frecuencia #frecuencia de caracteres
        self.izquierda = None #rama izquierda
        self.derecha = None
```

```
    def __lt__(self, otro):
        return self.frecuencia < otro.frecuencia

def generar_arbol(texto):
    frecuencias = {}
    for caracter in texto:
        frecuencias[caracter] = frecuencias.get(caracter, 0) + 1

    # Ordenamos manualmente los caracteres en orden ascendente por frecuencia
    caracteres_ordenados = sorted(frecuencias.items(), key=lambda x: (x[1], x[0]))

    heap = [NodoHuffman(caracter, freq) for caracter, freq in caracteres_ordenados]
    heapq.heapify(heap)

    while len(heap) > 1:
        izquierda = heapq.heappop(heap)  # Extraemos el menor
        derecha = heapq.heappop(heap)  # Extraemos el siguiente menor

        # Aseguramos que los nodos con mayor frecuencia se coloquen a la derecha
        if izquierda.frecuencia > derecha.frecuencia:
            izquierda, derecha = derecha, izquierda

        nodo = NodoHuffman(None, izquierda.frecuencia + derecha.frecuencia)
        nodo.izquierda = izquierda
        nodo.derecha = derecha
        heapq.heappush(heap, nodo)
```

```
    return heap[0]

def generar_codigos(raiz, codigo_actual="", codigos={}):
    if raiz is None:
        return
    if raiz.caracter is not None:
        codigos[raiz.caracter] = codigo_actual
    generar_codigos(raiz.izquierda, codigo_actual + "0", codigos)
    generar_codigos(raiz.derecha, codigo_actual + "1", codigos)
    return codigos

def exportar_arbol_pdf(raiz, nombre_archivo="arbol_huffman"):
    dot = Digraph(comment="Árbol de Huffman")
    def agregar_nodos(nodo, parent_id=None, label=""):
        if nodo is None:
            return
        nodo_id = f"{id(nodo)}"
        dot.node(nodo_id, label=f"{nodo.frecuencia}\n {nodo.caracter or "}")
        if parent_id:
            dot.edge(parent_id, nodo_id, label=label)
        agregar_nodos(nodo.izquierda, nodo_id, "0")
        agregar_nodos(nodo.derecha, nodo_id, "1")
    agregar_nodos(raiz)
    dot.render(nombre_archivo, format="pdf", cleanup=True)
    return f"{nombre_archivo}.pdf"

@app.route('/pdf/<filename>')
```

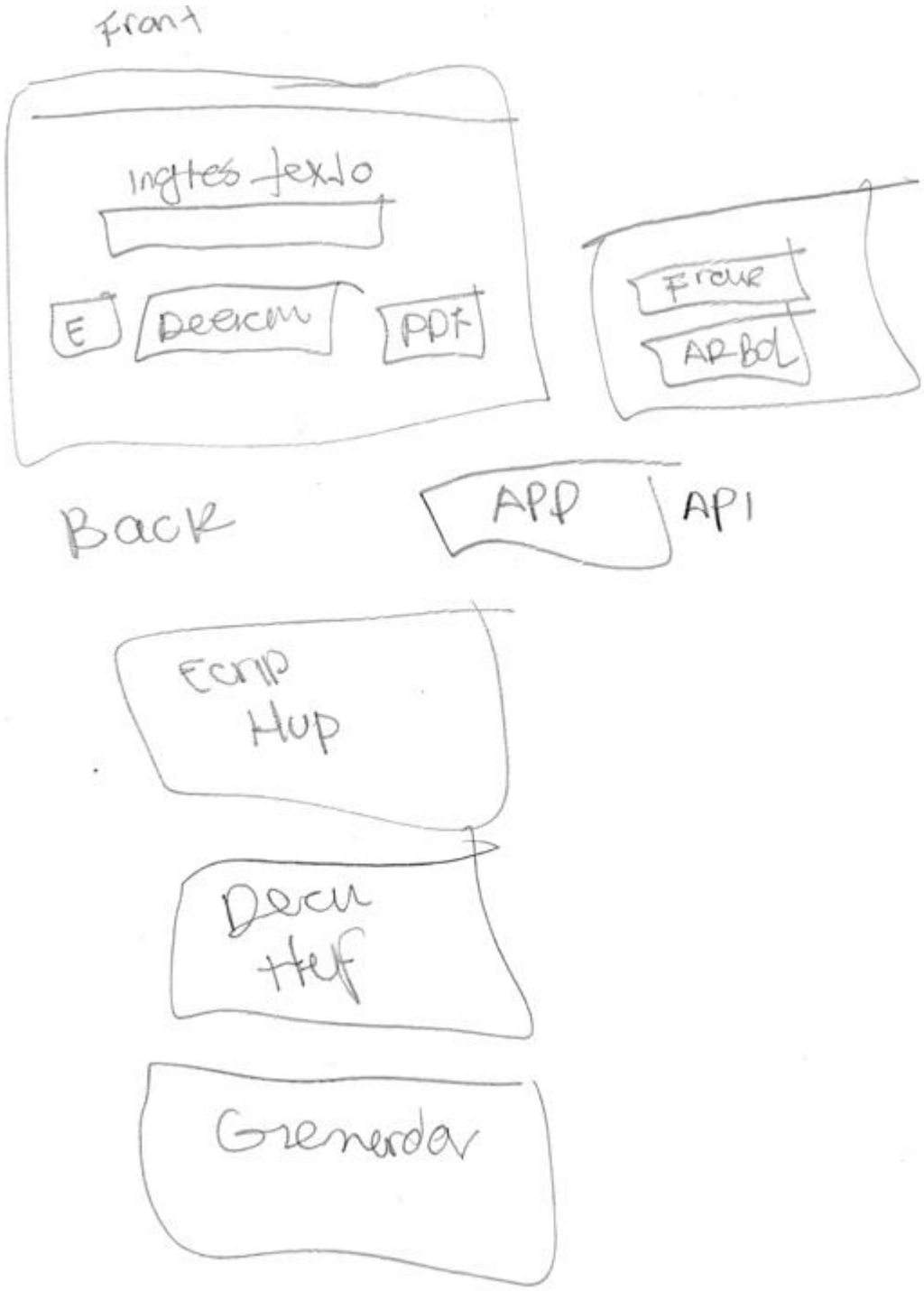
```
def serve_pdf(filename):
    return send_from_directory('.', filename)  # Sirve archivos desde la carpeta actual

@app.route("/descargar_pdf", methods = ["GET"])
def descargaar_pdf():
    archivo_pdf ="arbol_huffman.pdf"
    if os.path.exists(archivo_pdf):
        return send_from_directory('.', archivo_pdf, as_attachment=True)
    return jsonify({"Error": "Archivo no encontrado"}),

@app.route("/encriptar", methods=["POST"])#solo POST
def encriptar():
    data = request.get_json()
    texto = data.get("texto", "")
    raiz = generar_arbol(texto)
    codigos = generar_codigos(raiz)
    texto_encriptado = " ".join([codigos[caracter] for caracter in texto])
    pdf_path = exportar_arbol_pdf(raiz)
    return jsonify({
        "encriptado": texto_encriptado,
        "codigos": codigos, #agregamos  los codigo a la respuesta
        "pdf_path": f"/pdf/{pdf_path}"  # Cambia la ruta para usar el endpoint nuevo
    })

if __name__ == "__main__":
    app.run(debug=True, port=5000)
```

MOCUSKPS DEL SOFTWARE



CONCLUSIONES Y REFERENCIAS

El proyecto permitió aplicar el algoritmo de Huffman para la encriptación y desencriptación de mensajes, logrando comprimir la información de manera eficiente mediante árboles binarios.

La implementación en Python para el backend y React para el frontend facilitó la división del trabajo y la creación de una aplicación web funcional y amigable para el usuario.

La generación y visualización del árbol de Huffman con Graphviz permitió entender mejor la estructura interna del algoritmo y facilitó la comprobación de los resultados.

Este proyecto reforzó conceptos clave de estructuras de datos, algoritmos y desarrollo web, además de promover el trabajo en equipo y la integración de diferentes tecnologías.

Se concluye que el uso de algoritmos clásicos como Huffman sigue siendo relevante y aplicable en el contexto actual para problemas de compresión y encriptación simples.