UNIVERSIDAD DE GUADALAJARA

CENTRO UNIVERSITARIO DE CIENCIAS EXACTAS E INGENIERÍAS

INGENIERO EN COMPUTACIÓN

ANÁLISIS DE ALGORITMOS

EQUIPO TR3S

MACIEL VARGAS OSWALDO DANIEL

GARCÍA SALDIVAR HUGO GABRIEL

D06 26/10/2025

Divide y Vencerás – Programación Dinámica

Algoritmo de compresión

de imágenes mediante Quadtrees

I. Introducción

Los algoritmos pueden ser abordados de muchas maneras, donde se tiene el mismo objetivo y se produce el mismo resultado, pero dependiendo de la forma en que lo implementemos, tal vez nos encontremos con menos o más uso de memoria, más o menos tiempo o un código más o menos corto. Es por eso por lo que, en este trabajo, en búsqueda de entender distintas estrategias para implementar algoritmos, vamos a implementar y mostrar resultados para observar la diferencia entre estas.

En este trabajo vamos a implementar el algoritmo de compresión de imágenes mediante quadtrees, pero con el añadido de que usaremos tanto divide y vencerás como programación dinámica. El algoritmo de comprensión de imágenes busca reducir el tamaño de estás unificando los colores de cuadrantes de pixeles para adoptar u solo color y reducir los detalles. Este algoritmo es útil en diferentes contextos, ya que muchas veces tal vez no necesitamos imágenes de distintas resoluciones, si no que tal vez solo necesitamos distinguir de un solo elemento en la imagen, algo en lo que el algoritmo es bastante bueno, ya que es capaz de reducir considerablemente los detalles que no son el foco principal de una imagen.

Para lo anterior haremos uso de la técnica divide y vencerás, el cual es una alternativa bastante a la fuerza bruta importante, ya que esta nos ayuda a ser más eficientes, donde cada problema se divide en subproblemas y estos se van resolviendo de manera recursiva, hasta juntarlos para crear una solución grande y significativa. A su vez vamos a implementar la técnica de programación dinámica donde consiste en resolver pequeños problemas y guardar su solución, si en un futuro nos encontramos con el mismo problema, ya no es necesario efectuar un cálculo porque ya tenemos su solución. A continuación, la definición formal de ambas técnicas.

Divide y vencerás

El método "divide y vencerás" es una estrategia algorítmica que consiste en descomponer un problema complejo en subproblemas más pequeños, resolver estos subproblemas de manera recursiva y luego combinar sus soluciones para obtener la solución del problema original. Sus tres pasos principales son: dividir, vencer (resolver recursivamente los subproblemas hasta que sean casos base) y combinar.

Programación dinámica

La programación dinámica es una técnica matemática y de informática para resolver problemas complejos de manera eficiente, dividiéndolos en subproblemas más pequeños y almacenando sus soluciones para evitar recálculos. Se utiliza principalmente en la optimización, buscando la mejor solución (máxima o mínima) entre muchas alternativas, almacenando los resultados en una tabla para su uso posterior.

II. Objetivos

General

Implementar las técnicas de divide y vencerás y programación dinámica tomando como base el algoritmo de compresión de imágenes mediante quadtree, donde se pondrán a prueba distintas imágenes para producir resultados efectivos y comparar ambas técnicas en base a su complejidad temporal, evaluando así la efectividad de cada una.

Específicos

- Investigar el uso y aplicación del algoritmo de compresión de imágenes mediante quadtrees.

- Desarrollar el algoritmo de compresión de imágenes para obtener resultados certeros utilizando la técnica divide y vencerás.

- Implementar la técnica de programación dinámica al programa base y comprobar su funcionamiento.

- Implementar una GUI al programa para facilitar el funcionamiento de ambos programas.

-Añadir una gráfica de complejidad temporal para comparar y evaluar ambas técnicas implementadas.

III. Desarrollo

Algoritmos implementados

Como se mencionó anteriormente, en este trabajo implementamos el algoritmo de compresión de imágenes mediante quadtrees, cosa la cual no hay que confundir, el quadtrees es la manera en la que se dividen las operaciones y el área sobre la cual va operando el algoritmo, básicamente es el lienzo sobre el que se pinta.

Este algoritmo consiste en representar una imagen como un árbol, donde cada nodo puede dividirse en 4 cuadrantes y a su vez esos nuevos nodos en más nodos. Su objetivo es reducir la cantidad de datos necesarios para representar una imagen, aprovechando las zonas donde los píxeles son similares.

Primero se toma una imagen como un cuadrado inicial, ese nuestro primer nodo, nuestra primera imagen, iniciando con el proceso, se verifican si los pixeles son homogéneos según un umbral de error tomando en cuenta el color, si resultan ser homogéneos, se guarda ese nodo con el valor medio del color, en caso de que no lo sean, se divide en 4 cuadrantes donde se repite el proceso de comprobación. El resultado final es un árbol, donde cada rama es un valor que describe la imagen con diferentes niveles de detalle.

En este algoritmo podemos encontrar como resultado diferentes imágenes con distintos niveles de detalle, donde podemos obtener prácticamente la imagen original, pero perdiendo un poco de calidad en las áreas no importantes como aquellas zonas con el mismo color, pero manteniendo una calidad casi original en las zonas más importantes o detalladas de la imagen.

Código

```
# [Act.Codigo]Divide y Venceras Enntrega 2
# Equipo Tr3s
# Garcia Saldivar Hugo Gabriel
# Maciel Vargas Oswaldo Daniel
from PIL import Image, ImageDraw
import time
import matplotlib.pyplot as plt
import os
import shutil
import tkinter as tk
from tkinter import ttk, filedialog, messagebox
import threading
from PIL import ImageTk
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg

PADDING = 0
OUTPUT_SCALE = 1
ERROR_THRESHOLD = 15
```

```
GIF_DISPLAY_SIZE = (320, 320)

# Bloque para Quadtree
def weighted_average(hist):
    """Calcula el color promedio ponderado y el error de un histograma."""
    total = sum(hist)
    value, error = 0, 0
    if total > 0:
        value = sum(i * x for i, x in enumerate(hist)) / total
        error = sum(x * (value - i) ** 2 for i, x in enumerate(hist)) / total
        error = error ** 0.5
    return value, error

def color_from_histogram(hist):
    """Obtiene el color RGB promedio y el error combinado de un histograma."""
    r, re = weighted_average(hist[:256])
    g, ge = weighted_average(hist[256:512])
    b, be = weighted_average(hist[512:768])
    e = re * 0.2989 + ge * 0.5870 + be * 0.1140
    return (int(r), int(g), int(b)), e


class QuadtreeNode(object):
    """Nodo individual del Quadtree, representa una region de la imagen."""

    def __init__(self, img, box, depth):
        """Inicializa el nodo, calculando su color promedio y error."""
        self.box = box
        self.depth = depth
        self.children = None
        self.leaf = False
        image = img.crop(box)
        self.width, self.height = image.size
        hist = image.histogram()
        self.color, self.error = color_from_histogram(hist)

    def is_leaf(self):
```

```
            """Verifica si el nodo es una hoja (no tiene hijos)."""
            return self.leaf

        def split(self, img):
            """Divide el nodo actual en cuatro hijos (cuadrantes)."""
            l, t, r, b = self.box
            lr = int(l + (r - l) / 2)
            tb = int(t + (b - t) / 2)
            tl = QuadtreeNode(img, (l, t, lr, tb), self.depth + 1)
            tr = QuadtreeNode(img, (lr, t, r, tb), self.depth + 1)
            bl = QuadtreeNode(img, (l, tb, lr, b), self.depth + 1)
            br = QuadtreeNode(img, (lr, tb, r, b), self.depth + 1)
            self.children = [tl, tr, bl, br]


    class QuadtreeBase(object):
        """Clase base con la logica compartida para construir y renderizar el
arbol."""

        def __init__(self, image):
            self.root = None
            self.width, self.height = image.size
            self.max_depth = 0

        def get_leaf_nodes(self, depth):
            """Devuelve una lista de todos los nodos hoja en una profundidad dada."""
            if depth > self.max_depth:
                depth = self.max_depth
            leaf_nodes = []
            def get_leaf_nodes_recursion(node, target_depth):
                if node.is_leaf() or node.depth == target_depth:
                    leaf_nodes.append(node)
                elif node.children is not None:
                    for child in node.children:
                        get_leaf_nodes_recursion(child, target_depth)
            get_leaf_nodes_recursion(self.root, depth)
            return leaf_nodes
```

```
        def _create_image_from_depth(self, depth):
            """Renderiza una imagen de Pillow usando los nodos hoja de una profundidad."""
            m = OUTPUT_SCALE
            dx, dy = (PADDING, PADDING)
            image = Image.new('RGB', (int(self.width * m + dx), int(self.height * m + dy)))
            draw = ImageDraw.Draw(image)
            draw.rectangle((0, 0, self.width * m, self.height * m), (0, 0, 0))
            leaf_nodes = self.get_leaf_nodes(depth)
            for node in leaf_nodes:
                l, t, r, b = node.box
                box = (l * m + dx, t * m + dy, r * m - 1, b * m - 1)
                draw.rectangle(box, node.color)
            return image

        def create_gif(self, base_name, gif_dir, frames_dir, duration=500, loop=0):
            """Crea un GIF animado de la compresion, nivel por nivel."""
            folder_name = f"{base_name}_frames"
            frames_folder_path = os.path.join(frames_dir, folder_name)
            os.makedirs(frames_folder_path, exist_ok=True)
            print(f"Guardando fotogramas individuales en: '{frames_folder_path}/'")
            gif_file_path = os.path.join(gif_dir, f"{base_name}.gif")
            images = []
            end_product_image = self._create_image_from_depth(self.max_depth)
            for i in range(self.max_depth + 1):
                image = self._create_image_from_depth(i)
                images.append(image)
                try:
                    frame_path = os.path.join(frames_folder_path, f"frame_{i:02d}.png")
                    image.save(frame_path)
                except Exception as e:
                    print(f"Advertencia: No se pudo guardar el fotograma {frame_path}. Error: {e}")
            for _ in range(3):
```

```
                images.append(end_product_image)
            print(f"Creando GIF con {len(images)} fotogramas en '{gif_file_path}'...")
            images[0].save(
                gif_file_path,
                save_all=True,
                append_images=images[1:],
                duration=duration,
                loop=loop)
            return gif_file_path


    class QuadtreeDivideAndConquer(QuadtreeBase):
        """Implementa el Quadtree usando una estrategia Top-Down (Divide y Venceras)."""

        def __init__(self, image, max_depth=10):
            super().__init__(image)
            self.root = QuadtreeNode(image, image.getbbox(), 0)
            self.max_depth = 0
            self._build_tree_dc(image, self.root, max_depth)

        def _build_tree_dc(self, image, node, max_depth):
            """Construye el arbol recursivamente, dividiendo solo si el error es alto."""
            if (node.depth >= max_depth) or (node.error <= ERROR_THRESHOLD):
                if node.depth > self.max_depth:
                    self.max_depth = node.depth
                node.leaf = True
                return
            node.split(image)
            for child in node.children:
                self._build_tree_dc(image, child, max_depth)


    class QuadtreeDynamicProgramming(QuadtreeBase):
        """Implementa el Quadtree usando una estrategia Bottom-Up (Prog.
```

```
Dinamica)."""

            def __init__(self, image, max_depth=10):
                super().__init__(image)
                self.root = QuadtreeNode(image, image.getbbox(), 0)
                self._build_full_tree_dp(image, self.root, max_depth)
                self._prune_tree_dp(self.root)
                self.max_depth = 0
                self._update_max_depth(self.root)

            def _build_full_tree_dp(self, image, node, max_depth):
                """Paso 1 (DP): Construye el arbol completo hasta la profundidad
maxima."""
                if node.depth < max_depth:
                    node.split(image)
                    for child in node.children:
                        self._build_full_tree_dp(image, child, max_depth)
                else:
                    node.leaf = True

            def _prune_tree_dp(self, node):
                """Paso 2 (DP): Poda el arbol en post-orden (bottom-up) si el error es
bajo."""
                if not node.children:
                    return
                for child in node.children:
                    self._prune_tree_dp(child)
                if (node.error <= ERROR_THRESHOLD):
                    node.leaf = True
                    node.children = None
                else:
                    node.leaf = False

            def _update_max_depth(self, node):
                """Paso 3 (DP): Recalcula la profundidad real del arbol despues de
podar."""
                if node.is_leaf():
```

```
                if node.depth > self.max_depth:
                    self.max_depth = node.depth
            elif node.children:
                for child in node.children:
                    self._update_max_depth(child)

    # Bloque para GUI
    gif_frames_data = []
    gif_animation_job = None

    def get_unique_output_dir(base_dir_name):
        """Busca un nombre de carpeta de salida unico"""
        if not os.path.exists(base_dir_name):
            return base_dir_name

        counter = 1
        while True:
            new_dir_name = f"{base_dir_name}_{counter}"
            if not os.path.exists(new_dir_name):
                return new_dir_name
            counter += 1

    def run_analysis_in_thread(image_path, max_n, root, fig, ax, canvas,
start_button, clear_button, gif_label):
        """
        Funcion principal de analisis; se ejecuta en un hilo para no bloquear la
GUI.
        Realiza la Tarea 1 (Grafica) y Tarea 2 (GIFs) y actualiza la GUI.
        """
        try:
            print(f"Iniciando analisis con N={max_n} para '{image_path}'...")

            def pre_run_clear():
                ax.clear()
                ax.set_title("Calculando nueva grafica...", color="#E0E0E0")
                ax.set_xlabel("N", color="#E0E0E0")
                ax.set_ylabel("Tiempo (s)", color="#E0E0E0")
```

```
    ax.set_facecolor("#2D2D2D")
    ax.grid(False)
    if ax.get_legend():
        ax.get_legend().remove()
    canvas.draw()
    gif_label.config(text="Generando nuevo GIF...")

root.after(0, pre_run_clear)


OUTPUT_DIR = get_unique_output_dir('quadtree_resultados')
print(f"Se usara el directorio de salida: '{OUTPUT_DIR}'")

GIF_DIR = os.path.join(OUTPUT_DIR, 'gif')
FRAMES_DIR = os.path.join(OUTPUT_DIR, 'frames')

print("Creando estructura de carpetas...")
os.makedirs(GIF_DIR, exist_ok=True)
os.makedirs(FRAMES_DIR, exist_ok=True)

depths_n = []
times_dc = []
times_dp = []

print(f"Cargando imagen desde '{image_path}'...")
image = Image.open(image_path)
image = image.convert('RGB')

print("\n--- TAREA 1: Iniciando analisis de complejidad temporal ---")
print(f"Probando profundidades de 1 a {max_n}...")
print("-" * 30)

for n in range(1, max_n + 1):
    print(f"Calculando para n = {n}...")

    start_time_dc = time.perf_counter()
    _ = QuadtreeDivideAndConquer(image, max_depth=n)
```

```
                end_time_dc = time.perf_counter()
                time_taken_dc = end_time_dc - start_time_dc

                start_time_dp = time.perf_counter()
                _ = QuadtreeDynamicProgramming(image, max_depth=n)
                end_time_dp = time.perf_counter()
                time_taken_dp = end_time_dp - start_time_dp

                depths_n.append(n)
                times_dc.append(time_taken_dc)
                times_dp.append(time_taken_dp)

                print(f"  D&C: {time_taken_dc:.6f} segundos")
                print(f"  DP:  {time_taken_dp:.6f} segundos")

            print("-" * 30)
            print("Medicion completada. Generando grafica...")

            if depths_n:
                ax.plot(depths_n, times_dc, marker='o', linestyle='-', label='Divide
y Venceras')
                ax.plot(depths_n,     times_dp,     marker='x',     linestyle='--',
label='Programacion Dinamica')
                ax.set_xlabel('n (Nivel de Profundidad Maxima)')
                ax.set_ylabel('Tiempo (segundos)')
                ax.set_title('Comparacion de Complejidad Temporal')

                legend = ax.legend()
                legend.get_frame().set_facecolor('#555555')
                for text in legend.get_texts():
                    text.set_color('#E0E0E0')

                ax.grid(True, color='#555555')
                ax.set_xticks(range(1, max_n + 1))

                root.after(0, lambda: update_matplotlib_canvas(canvas))
```

```
            else:
                print("No se generaron datos para graficar.")
                ax.set_title("No se generaron datos para graficar")
                root.after(0, lambda: update_matplotlib_canvas(canvas))

            print("\n--- TAREA 2: Iniciando generacion de GIFs (usando MAX_DEPTH) ---")

            print(f"Construyendo Quadtree con Divide y Venceras (Top-Down)...")
            quadtree_dc = QuadtreeDivideAndConquer(image, max_depth=max_n)
            print(f"Profundidad maxima (D&C): {quadtree_dc.max_depth}")

            gif_path_dc = quadtree_dc.create_gif('quadtree_dc', GIF_DIR, FRAMES_DIR)
            print(f"Exito D&C! Archivos guardados en '{OUTPUT_DIR}'")

            root.after(0, lambda: load_and_play_gif(gif_label, gif_path_dc))

            print("-" * 30)

            print(f"Construyendo Quadtree con Programacion Dinamica (Bottom-Up)...")
            quadtree_dp = QuadtreeDynamicProgramming(image, max_depth=max_n)
            print(f"Profundidad maxima (DP): {quadtree_dp.max_depth}")
            quadtree_dp.create_gif('quadtree_dp', GIF_DIR, FRAMES_DIR)
            print(f"Exito DP! Archivos guardados en '{OUTPUT_DIR}'")

            print("-" * 30)
            print(f"Proceso completado.")

        except FileNotFoundError:
            print(f"Error: No se pudo encontrar la imagen '{image_path}'.")
            messagebox.showerror("Error", f"No se pudo encontrar la imagen:\n{image_path}")
        except Exception as e:
            print(f"Ocurrio un error inesperado: {e}")
            messagebox.showerror("Error", f"Ocurrió un error:\n{e}")
            ax.set_title(f"Error: {e}")
            root.after(0, lambda: update_matplotlib_canvas(canvas))
```

```
            root.after(0, lambda: gif_label.config(text=f"Error: {e}"))
        finally:
            print("Re-habilitando botones...")
            root.after(0, lambda: start_button.config(state="normal"))
            root.after(0, lambda: clear_button.config(state="normal"))

    def update_matplotlib_canvas(canvas):
        """Refresca el lienzo de Matplotlib (debe llamarse desde el hilo principal)."""
        canvas.draw()

    def load_and_play_gif(gif_label, gif_path):
        """Carga los fotogramas del GIF generado y comienza la animacion."""
        global gif_frames_data, gif_animation_job

        if gif_animation_job:
            gif_label.after_cancel(gif_animation_job)
            gif_animation_job = None

        gif_frames_data = []

        try:
            gif = Image.open(gif_path)
            print(f"Cargando {gif.n_frames} fotogramas del GIF...")
            for i in range(gif.n_frames):
                gif.seek(i)

                frame = gif.copy()
                frame.thumbnail(GIF_DISPLAY_SIZE, Image.Resampling.LANCZOS)
                frame_image = ImageTk.PhotoImage(frame)

                gif_frames_data.append(frame_image)

            if gif_frames_data:
                gif_label.config(text="")
                animate_gif_frame(gif_label, 0)
            else:
```

```
                gif_label.config(text="No se pudo cargar el GIF.")

            except Exception as e:
                print(f"Error al cargar GIF: {e}")
                gif_label.config(text=f"Error al cargar GIF:\n{e}")

        def animate_gif_frame(gif_label, frame_index):
            """Funcion recursiva para mostrar cada fotograma del GIF en un bucle."""
            global gif_frames_data, gif_animation_job

            if not gif_frames_data:
                return

            frame = gif_frames_data[frame_index]
            gif_label.config(image=frame)

            next_index = (frame_index + 1) % len(gif_frames_data)

            gif_animation_job = gif_label.after(500, animate_gif_frame, gif_label,
next_index)


        def create_gui():
            """Crea, estiliza y lanza la ventana principal de la GUI con Tkinter."""

            root = tk.Tk()
            root.title("Comparador Quadtree (D&C vs DP)")
            root.geometry("1200x800")

            BG_COLOR = "#3E3E3E"
            FG_COLOR = "#E0E0E0"
            FRAME_BG = "#2D2D2D"
            BUTTON_BG = "#555555"
            BUTTON_ACTIVE = "#666666"
            DISABLED_BG = "#4A4A4A"
            DISABLED_FG = "#888888"
            ENTRY_BG = "#555555"
```

```
            root.config(bg=BG_COLOR)

            style = ttk.Style()
            style.theme_use('clam')
            style.configure('.',        background=BG_COLOR,        foreground=FG_COLOR,
bordercolor=FRAME_BG)
            style.configure('TFrame', background=BG_COLOR)
            style.configure('TLabel', background=BG_COLOR, foreground=FG_COLOR)
            style.configure('TLabelFrame',
                            background=BG_COLOR,
                            foreground=FG_COLOR,
                            bordercolor=BUTTON_BG)
            style.configure('TLabelFrame.Label',
                            background=BG_COLOR,
                            foreground=FG_COLOR)
            style.configure('TButton',
                            background=BUTTON_BG,
                            foreground=FG_COLOR,
                            bordercolor=BUTTON_BG)
            style.map('TButton',
                background=[('active', BUTTON_ACTIVE), ('disabled', DISABLED_BG)],
                foreground=[('disabled', DISABLED_FG)]
            )
            style.configure('TEntry',
                            fieldbackground=ENTRY_BG,
                            foreground=FG_COLOR,
                            insertcolor=FG_COLOR,
                            bordercolor=BUTTON_BG)
            style.map('TEntry',
                fieldbackground=[('disabled', DISABLED_BG)],
                foreground=[('disabled', DISABLED_FG)])
            style.configure('TSpinbox',
                            fieldbackground=ENTRY_BG,
                            foreground=FG_COLOR,
                            insertcolor=FG_COLOR,
                            bordercolor=BUTTON_BG,
```

```
                    arrowcolor=FG_COLOR)
        style.map('TSpinbox',
            fieldbackground=[('disabled', DISABLED_BG)],
            foreground=[('disabled', DISABLED_FG)])

        root.grid_rowconfigure(0, weight=1)
        root.grid_rowconfigure(1, weight=10)
        root.grid_columnconfigure(0, weight=2)
        root.grid_columnconfigure(1, weight=2)
        root.grid_columnconfigure(2, weight=1)

        image_path_var = tk.StringVar()
        n_var = tk.IntVar(value=8)

        frame_controls = ttk.LabelFrame(root, text="Configurar y Ejecutar")
        frame_controls.grid(row=0, column=0, columnspan=3, sticky="nsew", padx=10,
pady=5)
        frame_controls.grid_columnconfigure(0, weight=3)
        frame_controls.grid_columnconfigure(1, weight=2)

        frame_path = ttk.Frame(frame_controls, style='TFrame')
        frame_path.grid(row=0, column=0, sticky="nsew", padx=5, pady=5)

        ttk.Label(frame_path, text="Ruta:").pack(side=tk.LEFT, padx=(0, 5))
        path_entry     =     ttk.Entry(frame_path,     textvariable=image_path_var,
state="readonly", width=70)
        path_entry.pack(side=tk.LEFT, fill=tk.X, expand=True, padx=5)

        def select_image():
            file_path = filedialog.askopenfilename(
                title="Seleccionar imagen",
                filetypes=[("Imagenes", "*.png *.jpg *.jpeg *.bmp"), ("Todos los
archivos", "*.*")]
            )
            if file_path:
                image_path_var.set(file_path)
```

```
        browse_button = ttk.Button(frame_path, text="Examinar...",
command=select_image)
        browse_button.pack(side=tk.LEFT, padx=(5, 0))

        frame_config = ttk.Frame(frame_controls, style='TFrame')
        frame_config.grid(row=0, column=1, sticky="nsew", padx=5, pady=5)
        ttk.Label(frame_config, text="N:").pack(side=tk.LEFT, padx=(10, 0))
        n_spinbox = ttk.Spinbox(frame_config, from_=1, to=12, textvariable=n_var,
width=5)
        n_spinbox.pack(side=tk.LEFT, padx=5)
        ttk.Label(frame_config, text="(Recomendado: 8)",
foreground="#AAAAAA").pack(side=tk.LEFT, padx=5)

        clear_button = ttk.Button(frame_config, text="Limpiar")
        clear_button.pack(side=tk.RIGHT, padx=(0, 5))

        start_button = ttk.Button(frame_config, text="Iniciar")
        start_button.pack(side=tk.RIGHT, padx=(5, 20))

        frame_graph = ttk.LabelFrame(root, text="Resultados: Gráfica")
        frame_graph.grid(row=1, column=0, columnspan=2, sticky="nsew", padx=10,
pady=10)

        fig = plt.Figure(figsize=(5, 4), dpi=100, facecolor=BG_COLOR)
        ax = fig.add_subplot(111, facecolor=FRAME_BG)
        ax.set_title("La grafica aparecera aqui", color=FG_COLOR)
        ax.set_xlabel("N", color=FG_COLOR)
        ax.set_ylabel("Tiempo (s)", color=FG_COLOR)
        ax.tick_params(axis='x', colors=FG_COLOR)
        ax.tick_params(axis='y', colors=FG_COLOR)
        ax.spines['top'].set_color(BUTTON_BG)
        ax.spines['bottom'].set_color(BUTTON_BG)
        ax.spines['left'].set_color(BUTTON_BG)
        ax.spines['right'].set_color(BUTTON_BG)

        canvas = FigureCanvasTkAgg(fig, master=frame_graph)
        canvas.get_tk_widget().config(bg=BG_COLOR)
```

```
        canvas.get_tk_widget().pack(side=tk.TOP, fill=tk.BOTH, expand=True)

        frame_gif = ttk.LabelFrame(root, text="Resultados: GIF Generado")
        frame_gif.grid(row=1, column=2, sticky="nsew", padx=10, pady=10)

        gif_label = ttk.Label(frame_gif, text="El GIF de D&C aparecera aqui...",
anchor="center", style="TLabel")
        gif_label.pack(fill=tk.BOTH, expand=True)

        def clear_results_callback():
                """Callback del boton 'Limpiar': resetea la grafica y el visor de
GIF."""
                print("Limpiando resultados ...")

                ax.clear()
                ax.set_title("La grafica aparecera aqui", color=FG_COLOR)
                ax.set_xlabel("N", color=FG_COLOR)
                ax.set_ylabel("Tiempo (s)", color=FG_COLOR)
                ax.set_facecolor(FRAME_BG)
                ax.grid(False)
                if ax.get_legend():
                    ax.get_legend().remove()
                canvas.draw()

                global gif_frames_data, gif_animation_job
                if gif_animation_job:
                    gif_label.after_cancel(gif_animation_job)
                    gif_animation_job = None
                gif_frames_data = []

                try:
                    gif_label.config(image=None, text="El GIF de D&C aparecera
aqui...")
                except tk.TclError:
                    gif_label.config(image='', text="El GIF de D&C aparecera
aqui...")
```

```
                print("Forzando actualizacion de la GUI...")
                root.update_idletasks()
                try:
                    gif_label.config(image=None, text="El GIF de D&C aparecera
aqui...")
                except tk.TclError:
                    gif_label.config(image='', text="El GIF de D&C aparecera
aqui...")

                print("Limpieza completada.")

        clear_button.config(command=clear_results_callback)

        def start_analysis_callback():
            """Callback del boton 'Iniciar': valida la entrada y lanza el hilo de
analisis."""
            img_path = image_path_var.get()
            n_val = n_var.get()

            if not img_path:
                messagebox.showwarning("Falta Imagen", "Por favor, selecciona una
imagen primero.")
                return

            start_button.config(state="disabled")
            clear_button.config(state="disabled")

            analysis_thread = threading.Thread(
                target=run_analysis_in_thread,
                args=(img_path, n_val, root, fig, ax, canvas, start_button,
clear_button, gif_label),
                daemon=True
            )
            analysis_thread.start()

        start_button.config(command=start_analysis_callback)
```

```
    root.mainloop()


# Bloque Main
if __name__ == '__main__':
    """Punto de entrada principal: inicia la interfaz grafica."""
    create_gui()
```

Comparativa Teórica y práctica

En este trabajo como lo mencionamos anteriormente, vamos a realizar la comparativa entre la estrategia divide y vencerás y la programación dinámica. En la teoría, la programación dinámica debería ser más rápida que usando exclusivamente la estrategia divide y vencerás, porque en la programación dinámica vamos haciendo cada paso de igual manera que divide y vencerás, con la ventaja de que las operaciones realizadas previamente se guardan en caso de que se vuelvan a reutilizar, así que, si lo vemos como un árbol, esto cortaría distintas ramas de subproblemas en divide y vencerás, como consecuencia tendríamos mayor consumo de memoria.

Primero comenzamos con la estrategia divide y vencerás, y es muy sencillo, cada cuadrante de la imagen se divide en mas cuadrantes, lo que representan los subproblemas, donde estos se resuelven recursivamente haciendo las comparaciones, después se combinan esos valores de imágenes para construir la imagen comprimida.

En cambio, la técnica de programación dinámica es peculiar, ya que mientras va formando el árbol, cada operación es diferente, cada comparación de imágenes es diferente, entonces no se puede almacenar un valor, entonces tendría que formarse todo el árbol primero y después ir decidiendo que nodos se conservan.

En la práctica, el resultado fue muy sorprendente, ya que la estrategia de divide y vencerás, resultó ser mucho más rápida, que la programación dinámica, ya que el hecho de construir todo el árbol primero, hizo que la complejidad de tiempo fuera demasiada, ya que primero divide cada cuadrante n veces y después ya se pone a comparar, en cambio divide y vencerás, mientras va formando el árbol va haciendo las comparaciones hasta un máximo de división de cuadrantes de n profundidad, pro lo tanto si no se requiere no se hacen tantas operaciones.

Resultados

Obtuvimos los resultados esperados, la compresión de imágenes tal como se esperaba, a continuación, algunos ejemplos.

Primero encontramos la imagen original:

Esta imagen por si sola, es demasiado pixeleada, y hay algunos detalles que podemos omitir, como las zonas azules, donde estas podrían omitir detalles y no se perdería información, a continuación, la imagen comprimida:

Como se puede observar, se perdío un poco de información, pero se siguen distinguiendo la información clave.

A continuación, otro ejemplo, pero con un mayo nivel de profundidad:

Imagen original:

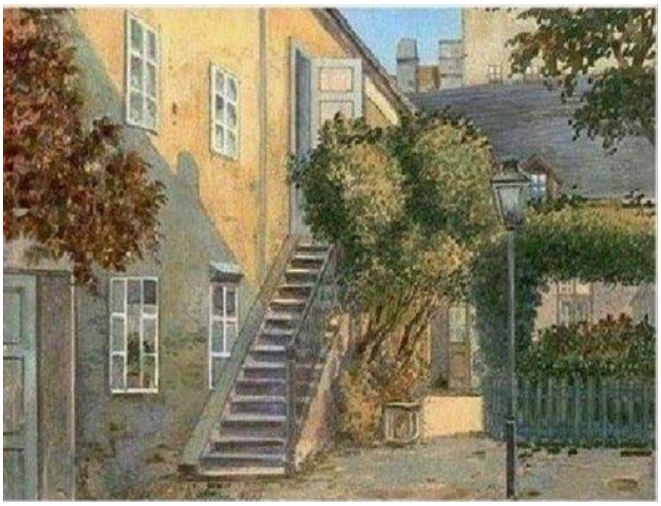
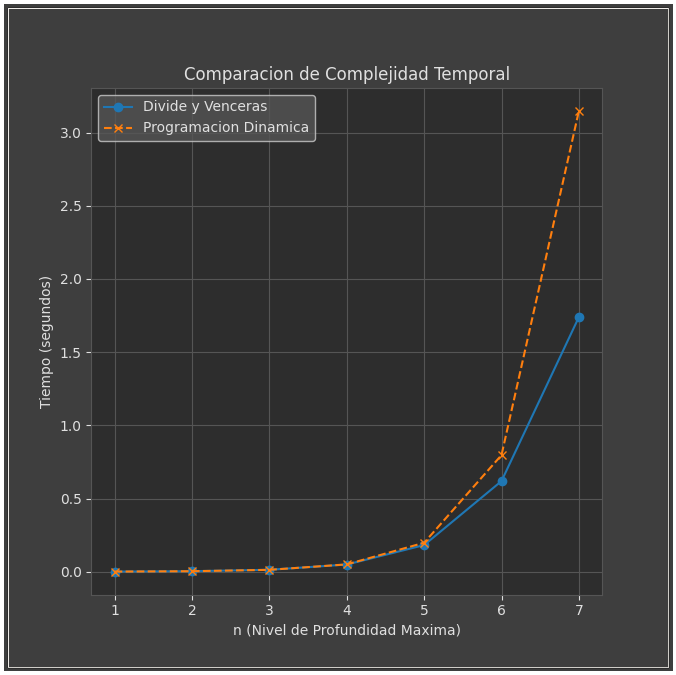


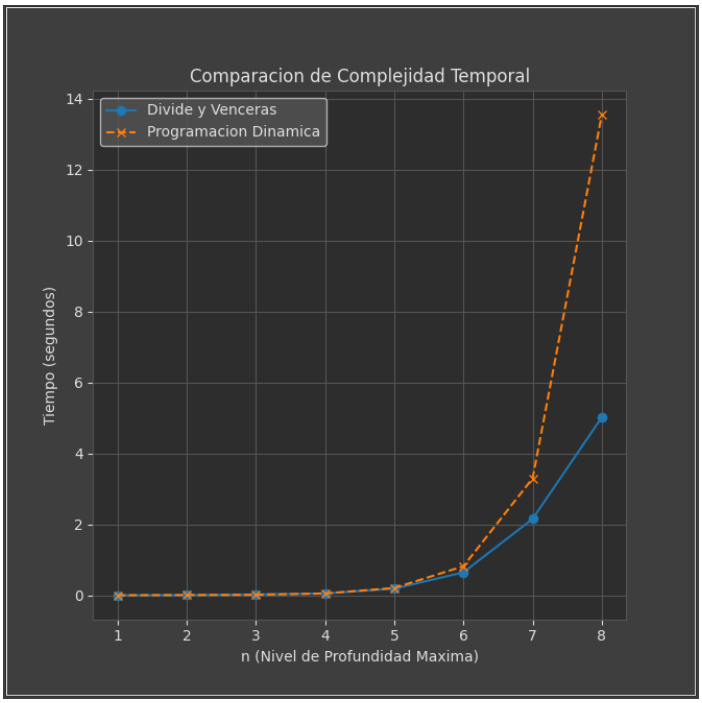
Imagen comprimida:

Gráficas

Como se puede observar, efectivamente es más tardada la programación dinámica por lo que se comentó anteriormente:



A continuación, otro ejemplo de gráfica, pero con un mayor nivel de profundidad:



IV. Conclusión

Oswaldo Maciel:

En lo personal considero que mi desempeño en esta actividad el adecuado y que cumplimos con todos los puntos esperados.

En esta actividad aprendí que realmente hay muchas formas de programar, pero, sobre todo, de abordar el mismo problema, de distintas soluciones. Al principio se intentó continuar nuestro algoritmo anterior (generación de cuadrados latinos con fuerza bruta), pero nos dimos cuenta de que era imposible abordarlo con divide y vencerás, entonces hicimos una búsqueda para encontrar un algoritmo interesante. Desde la actividad de fmnist, adquirí mucho interés por los algoritmos que usan imágenes, por lo que decidimos buscar un algoritmo relacionado con ello. Así fue donde encontramos el algoritmo de compresión de imágenes. Fue muy interesante observar la diferencia entre divide y vencerás y la programación dinámica ya en la práctica, pero también nos encontramos con algunas sorpresas como que en la misma programación dinámica se puede abordar de distintas maneras el uso de esta técnica y que no necesariamente es mejor que la estrategia divide y vencerás base.

Este trabajo me motiva a seguir aprendiendo distintas estrategias con distintos algoritmos, donde aparte de lo anterior, aprendamos muchas cosas interesantes sobre nuevos temas.

Hugo Garcia:

Fue una actividad muy interesante y gratificante de realizar, honestamente con este algoritmo de comprensión de imágenes por quad trees entendí de una mejor manera los algoritmos con un enfoque de divide y vencerás. Sin embargo, debo mencionar que mi desempeño en este programa al inicio no fue el mejor pues me costaba comprender el funcionamiento del algoritmo. Pero con el paso de los dias y el entendimiento del código se me facilito comprenderlo y sobre todo detectar las aplicaciones que tiene el mismo programa. Por lo que en términos de aprendizajes aprendí tanto mayor lógica para algoritmos de divide y vencerás, un mejor entendimiento para la programación dinámica, asi como seguir afinando mis habilidades para comprender graficas comparativas.

V. Referencias

- York, T. (2020, 15 julio). Quadtrees for Image Compression. Medium. https://medium.com/@tannerwyork/quadtrees-for-image-processing302536c95c00
- MrHeyheyhey27. (2014, 3 diciembre). Quadtree explanation [Vídeo]. YouTube. https://www.youtube.com/watch?v=jxbDYxm-pXg
- Wikipedia contributors. (2025, 13 octubre). Quadtree. Wikipedia. https://en.wikipedia.org/wiki/Quadtree