UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

MATEMATICA PARA COMPUTACION 2

CATEDRÁTICO: ING. JOSÉ ALFREDO GONZÁLEZ DÍAS

AUXILIAR: BRAYAN MEJIA

Manual Técnicopara el Proyecto: Aplicación de Algoritmos de Teoría de Grafos en Python

NOMBRES	CARNÉ
ADRIANA LIZETH CASTRO GONZALEZ	202202912
ADELAYDA MARIA ISABEL MORALES SAQUIZ	202200330
EVELYN YAINLETH CHACON ARANA	202107155
DAVID NORBERTO FABRO GUZMÁN	202307499
LUIS FERNANDO GONZALEZ	202307727

OBJETIVOS

1.1 General

Desarrollar una aplicación interactiva que permita visualizar la implementación de algoritmos de teoría de grafos (búsqueda en anchura y profundidad) en un entorno gráfico, facilitando la comprensión de su funcionamiento.

1.2 Específicos

- **Comprender la utilidad de la teoría de grafos** en el modelado y solución de problemas relacionados con Ciencias de la Computación.
- **Aplicar los conceptos de la teoría de grafos** en la programación mediante la implementación de algoritmos de búsqueda en anchura y profundidad.
- **Desarrollar una interfaz gráfica interactiva** para ingresar grafos, visualizar su estructura y evidenciar los resultados de los algoritmos.
- **Facilitar la visualización gráfica de los grafos** utilizando herramientas como Graphviz y matplotlib integradas en una interfaz gráfica con tkinter.

INTRODUCCION

La teoría de grafos es una rama importante de la matemática discreta que permite modelar relaciones y estructuras complejas mediante vértices (nodos) y aristas (conexiones). En la computación, se utiliza para resolver una variedad de problemas, desde la representación de redes hasta la optimización de rutas.

Este proyecto tiene como propósito principal implementar una aplicación interactiva que permita a los usuarios experimentar con grafos, ingresando datos manualmente y visualizando el efecto de dos algoritmos de búsqueda clásicos: búsqueda en anchura y búsqueda en profundidad. El enfoque es proporcionar una herramienta educativa que permita visualizar cómo estos algoritmos exploran un grafo y generan árboles de búsqueda.

La aplicación se ha desarrollado utilizando Python, con las bibliotecas tkinter para la interfaz gráfica, networkx para la manipulación de grafos y matplotlib para la visualización gráfica.

Manual Técnico para el Proyecto: Aplicación de Algoritmos de Teoría de Grafos en Python

1. Importaciones de Módulos

Para implementar la aplicación gráfica que interactúa con grafos, importamos los siguientes módulos:

- tkinter: Para crear la interfaz gráfica de usuario.
- networkx: Para la creación y manipulación de grafos.
- matplotlib: Para la visualización gráfica de los grafos.
- FigureCanvasTkAgg: Para integrar gráficos de matplotlib dentro de tkinter.

```
1  import tkinter as tk
2  from tkinter import messagebox
3  import networkx as nx
4  import matplotlib.pyplot as plt
5  from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
6
```

2. Definición de la Clase GraphApp

Se crea la clase GraphApp que representa la estructura principal de la aplicación. Esta clase gestiona la interfaz gráfica y las operaciones relacionadas con los grafos.

3. Inicialización de la Aplicación Gráfica

El método __init__ inicializa la ventana principal de la aplicación con el parámetro master, que corresponde a la ventana de tkinter. También se define el título de la ventana.

```
def __init__(self, master):
    # Inicialización de la aplicación gráfica
    self.master = master
    self.master.title("Grafos")
```

4. Creación de Lienzos para los Grafos

Se crearon dos lienzos utilizando FigureCanvasTkAgg, uno para mostrar el grafo original y otro para mostrar el resultado del algoritmo de búsqueda (árbol). Estos lienzos permiten visualizar los grafos en áreas separadas.

```
# Creación de los lienzos para mostrar los grafos
self.canvas_left = FigureCanvasTkAgg(plt.figure(figsize=(5, 5)), master=self.master)
self.canvas_left.get_tk_widget().pack(side=tk.LEFT, fill=tk.BOTH, expand=True)

self.canvas_right = FigureCanvasTkAgg(plt.figure(figsize=(5, 5)), master=self.master)
self.canvas_right.get_tk_widget().pack(side=tk.RIGHT, fill=tk.BOTH, expand=True)
```

5. Creación del Frame de la Interfaz Gráfica

El Frame contiene los elementos interactivos de la interfaz, como los campos de entrada y los botones. El Frame se posiciona en la parte superior de la ventana.

```
self.frame = tk.Frame(master)
self.frame.pack(side=tk.TOP, fill=tk.BOTH)
```

6. Elementos de la Interfaz Gráfica

Se crearon varios componentes de la interfaz:

- **Etiquetas**: Para indicar qué tipo de dato se espera (vértices y aristas).
- **Campos de entrada**: Para que el usuario ingrese los vértices y aristas.
- **Botones**: Para agregar aristas, generar el grafo original y aplicar los algoritmos de búsqueda.
- **Menú desplegable**: Permite seleccionar el algoritmo de búsqueda (Anchura o Profundidad).

```
self.vertices_label = tk.Label(self.frame, text="Vértices:")
self.vertices_label.grid(row=0, column=0)

self.vertices_entry = tk.Entry(self.frame)
self.vertices_entry.grid(row=0, column=1)

self.edge_label = tk.Label(self.frame, text="Arista (A--B):")
self.edge_label.grid(row=1, column=0)

self.edge_entry = tk.Entry(self.frame)
self.edge_entry.grid(row=1, column=1)

# Botón para agregar aristas al grafo
self.add_button = tk.Button(self.frame, text="Agregar", command=self.add_edge)
self.add_button.grid(row=1, column=2)

# Resumen de las aristas ingresadas
self.summary_label = tk.Label(self.frame, text="Resumen:")
self.summary_label.grid(row=2, column=0)
```

```
self.summary_text = tk.Text(self.frame, height=10, width=30)
self.summary_text.grid(row=2, column=1, columnspan=2)

# Botón para generar el grafo original
self.generate_button = tk.Button(self.frame, text="Generar Grafo", command=self.generate_graph)
self.generate_button.grid(row=3, column=1, columnspan=2)

# Botón para agregar el árbol de búsqueda
self.add_tree_button = tk.Button(self.frame, text="Agregar Árbol", command=self.add_tree)
self.add_tree_button.grid(row=4, column=1, columnspan=2)

# Selección del algoritmo de búsqueda (Anchura o Profundidad)
self.search_label = tk.Label(self.frame, text="Algoritmo:")
self.search_label.grid(row=5, column=0)

self.search_var = tk.StringVar()
self.search_var.set("Anchura")  # Valor predeterminado
self.search_dropdown = tk.OptionMenu(self.frame, self.search_var, "Anchura", "Profundidad")
self.search_dropdown.grid(row=5, column=1, columnspan=2)
```

7. Creación del Grafo

Se crea un grafo vacío utilizando networkx.Graph(), que servirá como estructura base donde se agregarán los vértices y aristas ingresados por el usuario.

```
# Creación del grafo
self.graph = nx.Graph()
```

8. Funciones de la Aplicación

- **add_edge**: Agrega una arista al grafo cuando se hace clic en el botón "Agregar". La arista debe ingresarse en el formato "A--B", y se valida que el formato sea correcto antes de agregarla al grafo.

```
def add_edge(self):
    # Función para agregar una arista al grafo
    edge = self.edge_entry.get()
    try:
        a, b = edge.split("--")
        self.graph.add_edge(a.strip(), b.strip())
        self.summary_text.insert(tk.END, f"Arista: {a.strip()}--{b.strip()}\n")
    except ValueError:
        messagebox.showerror("Error", "Formato de arista incorrecto (A--B)")
```

- **generate_graph**: Dibuja el grafo original cuando se hace clic en el botón "Generar Grafo", utilizando matplotlib para la

```
def generate_graph(self):
    # Función para generar el grafo original
    self.draw_graph(self.graph, self.canvas_left)
```

- **add_tree**: Aplica el algoritmo de búsqueda seleccionado (Anchura o Profundidad) y dibuja el árbol resultante.

```
def add_tree(self):
    # Función para agregar el árbol de búsqueda
    search_algorithm = self.search_var.get()
    source_nodes = [node.strip() for node in self.vertices_entry.get().split(",")]
    if all(node in self.graph for node in source_nodes):
        if search_algorithm == "Anchura":
            tree = nx.bfs_tree(self.graph, source=source_nodes[0])
        else:
            tree = nx.dfs_tree(self.graph, source=source_nodes[0])
        self.draw_graph(tree, self.canvas_right)
    else:
        messagebox.showerror("Error", "Uno o más nodos no están presentes en el grafo.")
```

- **draw_graph**: Función para dibujar cualquier grafo en uno de los lienzos gráficos. Se utiliza nx.draw para dibujar los nodos y aristas en el grafo.

```
def draw_graph(self, graph, canvas):
    # Función para dibujar un grafo en un lienzo
    canvas.figure.clear()  # Limpiar la figura antes de dibujar

    pos = nx.spring_layout(graph)
    nx.draw(graph, pos, with_labels=True, ax=canvas.figure.add_subplot(111))

    canvas.draw()
```

9. Función Principal main

Se crea una instancia de la clase GraphApp y se ejecuta el bucle principal de la aplicación mediante mainloop de tkinter.

```
def main():
    root = tk.Tk()
    app = GraphApp(root)
    root.mainloop()
```

10. Condición de Ejecución Directa

Esta condición verifica si el script está siendo ejecutado directamente o importado como un módulo. Si se ejecuta directamente, llama a la función main.

```
if __name__ == "__main__":
    main()
```

VIDEO:

https://drive.google.com/drive/folders/1GP9FpIzr3lAn8L293gx-8sY3IfrI7jFV?usp=sharing