UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
FACULTAD DE INGENIERÍA
MATEMÁTICA PARA COMPUTACIÓN 2
CATEDRÁTICO: ING. JOSÉ ALFREDO GONZÁLES
TUTOR AUXILIAR: ROBERTO GÓMEZ

MANUAL TÉCNICO

Marcos Daniel Bonifasi de León - 202202410
Kevin Daniel Catún Landaverde - 202200378
José Sebastian Pirir Romero - 202300335
SECCIÓN: A

GUATEMALA, 29 DE ABRIL DEL 2,024

ÍNDICE

INTRODUCCIÓN

En el ámbito de la informática y las ciencias de la computación, los grafos son una estructura de datos fundamental que se utiliza para representar relaciones entre entidades. Consisten en nodos (vértices) conectados por arcos (bordes), y su versatilidad los convierte en una herramienta poderosa para modelar una amplia gama de problemas y sistemas. En este manual, exploraremos una implementación en Python de un visualizador de grafos, desarrollado utilizando las bibliotecas NetworkX, Tkinter y Matplotlib. Los grafos desempeñan un papel crucial en la resolución de problemas en informática y disciplinas relacionadas. Permiten modelar y analizar diversas situaciones del mundo real, desde redes sociales hasta sistemas de transporte y redes de computadoras. Los algoritmos de búsqueda en anchura y profundidad son dos técnicas básicas para recorrer y analizar grafos. La búsqueda en anchura se centra en explorar todos los nodos vecinos antes de pasar a los nodos de niveles más profundos, lo que la hace ideal para encontrar el camino más corto entre dos nodos en un grafo no ponderado. Por otro lado, la búsqueda en profundidad se basa en explorar lo más profundamente posible a lo largo de un camino antes de retroceder, lo que resulta útil para recorrer todos los nodos de un grafo y encontrar componentes conectados. Comprender y aplicar estos algoritmos es esencial para resolver una variedad de problemas en informática, desde la navegación en redes hasta la planificación de rutas y la detección de ciclos en estructuras de datos.

OBJETIVOS

1. GENERAL

2. El objetivo general del código es proporcionar una herramienta interactiva y visual para la manipulación, representación y análisis de grafos en Python. Especialmente en el caso de los algoritmos de búsqueda por profundidad y anchura.

3. ESPECÍFICOS

3.1. Objetivo 1: Implementar una interfaz de usuario intuitiva utilizando Tkinter y Matplotlib que permita a los usuarios interactuar con grafos, agregar nodos y aristas, y ejecutar algoritmos de b√∫squeda en anchura y profundidad.

3.2. Objetivo 2: Utilizar la biblioteca NetworkX para la manipulaci√≥n y representaci√≥n de grafos, garantizando la eficiencia y la precisi√≥n en la visualizaci√≥n y an√°lisis de estructuras de datos complejas.

ALCANCES DEL SISTEMA

El objetivo del manual es proporcionar una guía detallada y práctica para utilizar el visualizador de grafos implementado en Python. Este manual busca instruir a los usuarios sobre cómo crear, modificar y analizar grafos utilizando la interfaz proporcionada, así como también profundizar en el entendimiento de los algoritmos de búsqueda en anchura y profundidad aplicados en el contexto de grafos. Además, el manual pretende servir como una referencia útil para aquellos interesados en comprender y resolver problemas complejos mediante el uso de grafos y algoritmos asociados.

ESPECIFICACIÓN TÉCNICA

- **REQUISITOS DE HARDWARE**
  - Procesador: Se recomienda un procesador de al menos 2 GHz para un rendimiento óptimo.
  - Memoria RAM: Se recomienda al menos 4 GB de RAM para ejecutar Python y las bibliotecas asociadas sin problemas.
  - El visualizador de grafos en Python es compatible con varios sistemas operativos, incluyendo Windows, macOS y Linux. Por lo tanto, el programador puede elegir el sistema operativo que mejor se adapte a sus necesidades y preferencias.
- **REQUISITOS DE SOFTWARE**
  - El programador necesitará tener instalado Python en su sistema. Se recomienda una versión reciente de Python 3.x para garantizar la compatibilidad con las bibliotecas y herramientas utilizadas en el proyecto.
  - Es una buena práctica crear un entorno virtual para el proyecto para mantener las dependencias del proyecto separadas del sistema global de Python y evitar conflictos entre versiones. El programador puede utilizar herramientas como virtualenv o conda para crear y gestionar entornos virtuales.
  - El programador necesitará un editor de código o un entorno de desarrollo integrado (IDE) para escribir, depurar y ejecutar el código Python. Algunas opciones populares incluyen Visual Studio Code, PyCharm, Sublime Text y Atom.

DESCRIPCIÓN DE LA SOLUCIÓN

Al abordar los requisitos del enunciado del código de Python, nuestro enfoque se centró en entender completamente las necesidades del proyecto y seleccionar las herramientas más adecuadas para su implementación. Para ello, primero analizamos detenidamente los requisitos específicos, que incluían la creación de un visualizador de grafos con funcionalidades para agregar nodos y aristas, así como la capacidad de ejecutar algoritmos de búsqueda en anchura y profundidad.

Una vez que tuvimos una comprensión clara de los requisitos, comenzamos a evaluar las diferentes opciones disponibles para implementar el visualizador de grafos. Optamos por utilizar la biblioteca NetworkX de Python debido a su robustez y versatilidad en la manipulación y representación de grafos. NetworkX proporciona una amplia gama de funciones que nos permiten crear, modificar y analizar grafos de manera eficiente.

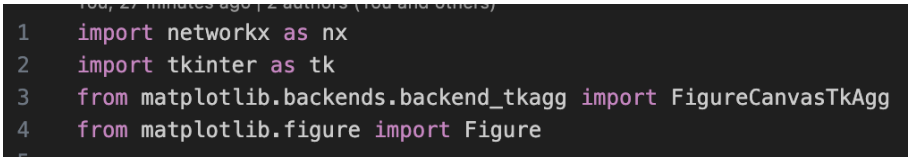
Para la interfaz gráfica de usuario, elegimos Tkinter junto con Matplotlib. Tkinter es una biblioteca estándar de Python que nos permite crear interfaces gráficas de usuario de manera sencilla y eficiente. Combinado con Matplotlib, que ofrece capacidades avanzadas de visualización, pudimos diseñar una interfaz gráfica intuitiva y atractiva para nuestro visualizador de grafos.

Además, consideramos la modularidad y la extensibilidad del código durante todo el proceso de desarrollo. Diseñamos una arquitectura flexible que permitiría incorporar nuevas funcionalidades y realizar modificaciones en el futuro sin dificultades significativas.

LÓGICA DEL PROGRAMA

- **Clase: GraphVisualizer**

- **Librerías**



  - NetworkX: Biblioteca de Python para la creación, manipulación y estudio de grafos. La utilizamos para crear y manipular grafos, agregar nodos y aristas, y ejecutar algoritmos de búsqueda en anchura y profundidad.
  - Tkinter: Biblioteca estándar de Python para crear interfaces gráficas de usuario. La usamos para construir la interfaz gráfica de usuario del visualizador de grafos, utilizando widgets como Entry, Button y Canvas.

- **Función Run**

  La función run se utiliza para iniciar el bucle principal de eventos de la interfaz gráfica de usuario (GUI). En nuestro contexto, esta función se encuentra en el objeto principal del visualizador de grafos y se llama para iniciar la ejecución del programa. Dentro de la función run, se inicia el bucle de eventos que espera y maneja las interacciones del usuario con la interfaz gráfica, como hacer clic en botones o ingresar texto. Esto asegura que la aplicación esté en constante escucha de las acciones del usuario y pueda responder adecuadamente.

```
117        def run(self):
118            self.root.mainloop()
```

- **Métodos y Funciones utilizadas**

  A continuación se dará una explicación general de lo que hace cada método:

  - *__init__:* Este método es el constructor de la clase y se llama automáticamente cuando se crea una nueva instancia del objeto.

```
7       def __init__(self):
8           self.graph = nx.Graph()
9           self.root = tk.Tk()
10          self.root.title("Algoritmo de búsqueda en Anchura y Profundidad")
11          self.root.geometry("800x600")
12          self.source_vertex = None
```

  - setup_ui(): Este método se encarga de configurar la interfaz de usuario del visualizador de grafos.

```
16          def setup_ui(self):
17              self.setup_vertex_entry()
18              self.setup_add_vertex_button()
19              self.setup_edge_entries()
20              self.setup_add_edge_button()
21              self.setup_info_button()
22              self.setup_graph_canvas()
23              self.setup_buttons()
```

  - setup_vertex_entry(), setup_add_vertex_button(), setup_edge_entries(), setup_add_edge_button(), setup_info_button(), setup_graph_canvas(), setup_buttons(): Estos métodos son responsables de configurar widgets específicos de la interfaz de usuario, como entradas de texto, botones y lienzo de gráficos.

```
    def setup_vertex_entry(self):
        self.vertex_entry = tk.Entry(self.root)
        self.vertex_entry.pack()

    def setup_add_vertex_button(self):
        self.add_vertex_button = tk.Button(self.root, text="Agregar vértice", command=self.add_vertex)
        self.add_vertex_button.pack()

    def setup_edge_entries(self):
        self.edge_entry_1 = tk.Entry(self.root)
        self.edge_entry_1.pack()
        self.edge_entry_2 = tk.Entry(self.root)
        self.edge_entry_2.pack()

    def setup_add_edge_button(self):
        self.add_edge_button = tk.Button(self.root, text="Agregar arista", command=self.add_edge)
        self.add_edge_button.pack()

    def setup_info_button(self):
        self.print_info_button = tk.Button(self.root, text="Info de datos agregados (consola)", command=self.print_info)
        self.print_info_button.pack()

    def setup_graph_canvas(self):
        self.figure = Figure(figsize=(7, 7))
        self.ax = self.figure.add_subplot(111)
        self.canvas = FigureCanvasTkAgg(self.figure, self.root)
        self.canvas.get_tk_widget().pack()

    def setup_buttons(self):
        self.setup_draw_button()
        self.setup_bfs_button()
        self.setup_dfs_button()
```

- add_vertex(), add_edge(), print_info(), draw_graph(), show_bfs(), show_dfs(), draw_graph_with_search(): Estos métodos realizan acciones específicas en respuesta a eventos del usuario, como agregar vértices y aristas, imprimir información sobre el grafo, dibujar el grafo en el lienzo, y ejecutar algoritmos de búsqueda en anchura y profundidad.

```
    def add_vertex(self):
        vertex = self.vertex_entry.get()
        if vertex:
            self.graph.add_node(vertex)
            self.vertex_entry.delete(0, tk.END)
            self.source_vertex = vertex
        else:
            print("Introduce un nombre de vértice válido.")

    def add_edge(self):
        source = self.edge_entry_1.get()
        target = self.edge_entry_2.get()
        if source and target:
            self.graph.add_edge(source, target)
            self.edge_entry_1.delete(0, tk.END)
            self.edge_entry_2.delete(0, tk.END)
            self.source_vertex = source
        else:
            print("Introduce nodos válidos.")

    def print_info(self):
        print("Número de nodos:", self.graph.number_of_nodes(), "\nNúmero de bordes:", self.graph.number_of_edges())

    def draw_graph(self):
        self.ax.clear()
        pos = nx.spring_layout(self.graph)
        nx.draw(self.graph, pos=pos, ax=self.ax, with_labels=True, node_color='skyblue', edge_color='gray')
        self.canvas.draw()

    def show_bfs(self):
        bfs_edges = list(nx.bfs_edges(self.graph, source=self.source_vertex))
        self.draw_graph_with_search(bfs_edges, 'bfs')

    def show_dfs(self):
        dfs_edges = list(nx.dfs_edges(self.graph, source=self.source_vertex))
        self.draw_graph_with_search(dfs_edges, 'dfs')

    def draw_graph_with_search(self, search_edges, search_type):
        self.ax.clear()
        pos = nx.spring_layout(self.graph)
        nx.draw(self.graph, pos=pos, ax=self.ax, with_labels=True, node_color='skyblue', edge_color='gray')
        if search_edges:
            edge_color = 'red' if search_type == 'bfs' else 'green'
            nx.draw_networkx_edges(self.graph, pos=pos, edgelist=search_edges, edge_color=edge_color, ax=self.ax)
            nx.draw_networkx_nodes(self.graph, pos=pos, nodelist=[self.source_vertex] + [v for u, v in search_edges], node_color=edge_color, ax=self.ax)
        self.canvas.draw()

```