UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
FACULTAD DE INGENIERÍA
MATEMATICA PARA COMPUTACION 2
CATEDRÁTICO: ING. JOSE ALFREDO GONZALEZ DÍAZ
TUTOR ACADÉMICO: ROBERTO GOMEZ

MANUAL TÉCNICO

ANGEL JOSE CARDENAS POCON
CARNÉ: 202304063
SECCIÓN: A

GUATEMALA, 29 DE ABRIL DEL 2,024

ÍNDICE

INTRODUCCIÓN

El presente manual técnico proporciona una guía detallada sobre la estructura, el funcionamiento interno y los requisitos del sistema del programa de visualización de grafos con algoritmos implementados, desarrollado utilizando el lenguaje de programación Python y las bibliotecas tkinter, networkx y matplotlib.

El programa ofrece una interfaz gráfica intuitiva que permite a los usuarios ingresar grafos, aplicar algoritmos como la búsqueda en anchura y visualizar tanto el grafo original como el resultado de la aplicación del algoritmo de forma dinámica y comprensible.

Este manual está dirigido a desarrolladores y programadores interesados en comprender cómo funciona el programa, así como en realizar modificaciones o extensiones al código para adaptarlo a sus necesidades específicas.

OBJETIVOS

1. GENERAL

1.1. Facilitar la comprensión y la modificación del programa: El objetivo general de este manual es proporcionar una guía completa que permita a los desarrolladores comprender en profundidad el funcionamiento del programa y realizar modificaciones o extensiones según sea necesario.

2. ESPECÍFICOS

2.1. Describir la estructura del código: Proporcionar una explicación detallada de la organización del código fuente del programa, incluyendo la función de cada archivo y módulo, así como la interacción entre ellos.

2.2. Detallar el funcionamiento de los algoritmos implementados: Explicar paso a paso el funcionamiento de los algoritmos implementados en el programa, centrándose especialmente en el algoritmo de búsqueda en anchura (BFS).

ALCANCES DEL SISTEMA

1. Interfaz Gráfica de Usuario (GUI): El programa se centra en proporcionar una interfaz gráfica intuitiva para que los usuarios puedan ingresar grafos, aplicar algoritmos y visualizar los resultados de manera interactiva.
2. Visualización de Grafos: El sistema permite la visualización de grafos simples ingresados por el usuario, así como la representación gráfica de los resultados de la aplicación de algoritmos como la búsqueda en anchura.
3. Algoritmo de Búsqueda en Anchura (BFS): Se implementa específicamente el algoritmo de búsqueda en anchura para explorar los grafos ingresados por el usuario y mostrar visualmente el resultado de la búsqueda en la interfaz gráfica.
4. Ingreso de Grafos: Los usuarios pueden ingresar vértices y aristas manualmente a través de la interfaz gráfica. Sin embargo, el programa no proporciona funcionalidades avanzadas para la generación automática de grafos o la importación de grafos desde archivos externos.
5. Funcionalidades Básicas: El programa se enfoca en proporcionar funcionalidades básicas para la visualización de grafos y la aplicación de algoritmos simples. No se incluyen funcionalidades avanzadas como la optimización de algoritmos, la manipulación de grafos grandes o la visualización de datos en tiempo real.
6. Plataforma de Desarrollo: El programa se desarrolla utilizando Python y las bibliotecas tkinter, networkx y matplotlib. Se asume que el sistema será ejecutado en plataformas compatibles con Python y estas bibliotecas, como sistemas Windows, macOS o Linux

ESPECIFICACIÓN TÉCNICA

- **REQUISITOS DE HARDWARE**
    - Procesador: Se recomienda un procesador de al menos 1 GHz o superior para un rendimiento óptimo del programa.
    - Memoria RAM: Se recomienda al menos 2 GB de RAM para ejecutar el programa de manera eficiente, especialmente al trabajar con grafos de tamaño moderado.
    - Espacio de Almacenamiento: El espacio de almacenamiento requerido para el programa en sí es mínimo. Sin embargo, se debe considerar el espacio adicional necesario para almacenar grafos si se desea guardarlos en archivos externos.
    - Tarjeta Gráfica: No se requiere una tarjeta gráfica dedicada, ya que el programa no realiza tareas gráficamente intensivas. La mayoría de las tarjetas gráficas integradas son adecuadas para ejecutar el programa.
    -
- **REQUISITOS DE SOFTWARE**
    - Sistema Operativo: El programa es compatible con los principales sistemas operativos, incluidos Windows, macOS y Linux.
    - Python: Se requiere una instalación de Python 3.x para ejecutar el programa. Se recomienda utilizar la última versión estable de Python disponible.
    - Bibliotecas Python:
        - tkinter: La biblioteca tkinter debe estar instalada como parte de la instalación estándar de Python.
        - networkx: Se requiere la instalación de la biblioteca networkx para trabajar con grafos en Python. Puede instalarse utilizando el administrador de paquetes pip.
        - matplotlib: La biblioteca matplotlib debe estar instalada para la visualización de grafos en la interfaz gráfica. Puede instalarse utilizando pip.

- Entorno de Desarrollo Integrado (IDE): No se requiere un IDE específico, pero se recomienda el uso de un entorno de desarrollo integrado como Visual Studio Code, PyCharm o Jupyter Notebook para facilitar el desarrollo y la depuración del código.

DESCRIPCIÓN DE LA SOLUCIÓN

- La solución propuesta es un programa informático que permite la visualización interactiva de grafos y la aplicación de algoritmos sobre ellos. La aplicación está desarrollada en Python y utiliza las bibliotecas tkinter, networkx y matplotlib para crear una interfaz gráfica intuitiva y proporcionar funcionalidades de manipulación de grafos.

LÓGICA DEL PROGRAMA

❖ NOMBRE DE LA CLASE

```
import tkinter as tk
from tkinter import ttk
import networkx as nx
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
```

➢ Librerías

- ❖ tkinter: Biblioteca de Python para crear interfaces gráficas de usuario (GUI). Se utiliza para crear la interfaz gráfica del programa donde los usuarios pueden interactuar.
- ❖ networkx: Biblioteca de Python para la creación, manipulación y estudio de grafos. Se utiliza para crear y manipular grafos, así como para aplicar algoritmos como la búsqueda en anchura.
- ❖ matplotlib: Biblioteca de Python para la visualización de datos en 2D y 3D. Se utiliza para generar representaciones visuales de los grafos en la interfaz gráfica del programa.

➢ Variables Globales de la clase _(El nombre de su clase actual)

```
class GraphVisualization(tk.Frame):
    def __init__(self, master=None):
        super().__init__(master)
        self.master = master
        self.master.title("Búsqueda en Anchura")
        self.master.attributes('-fullscreen', True)  # Hacer la ventana de pantalla completa
        self.master.configure(bg="#FFEEEE") # Cambiar el color de fondo de la ventana

        self.original_graph = nx.Graph()
        self.graph = nx.Graph()
        self.vertices = []
        self.edges = []
        self.canvas = None
        self.bfs_canvas = None
        self.create_widgets()
```

1. master: Es el elemento padre de la interfaz gráfica, sobre el cual se construyen todos los demás elementos.
2. original_graph: Es un objeto de tipo Graph de la biblioteca networkx que representa el grafo original ingresado por el usuario.
3. graph: Es otro objeto de tipo Graph de networkx que representa el grafo actual sobre el cual se están aplicando los algoritmos.
4. vertices: Es una lista que almacena los vértices del grafo ingresado por el usuario.
5. edges: Es una lista que almacena las aristas del grafo ingresado por el usuario.
6. canvas: Es el área de dibujo donde se muestra el grafo original.
7. bfs_canvas: Es otra área de dibujo donde se muestra el grafo con el algoritmo de búsqueda en anchura aplicado.
8. create_widgets(): Es un método que inicializa y coloca los elementos de la interfaz gráfica en la ventana principal.

➢ Función Main

```
if __name__ == "__main__":
    root = tk.Tk()
    app = GraphVisualization(master=root)
    app.mainloop()
```

1. if __name__ == "__main__": Este es un bloque condicional que verifica si el script está siendo ejecutado directamente como un programa independiente, en lugar de ser importado como un módulo por otro script. Si es así, se ejecuta el código dentro del bloque.
2. root = tk.Tk(): Esta línea crea una instancia de la clase Tk() de la biblioteca tkinter, que representa la ventana principal de la aplicación.
3. app = GraphVisualization(master=root): Aquí se crea una instancia de la clase GraphVisualization, que representa la aplicación de visualización de grafos. Se pasa la ventana principal (root) como el parámetro master para indicar que la ventana principal es el elemento padre de la aplicación.

4. app.mainloop(): Este método inicia el bucle principal de eventos de la interfaz gráfica, lo que significa que la aplicación comienza a responder a las interacciones del usuario. El programa permanecerá en este bucle hasta que la ventana principal sea cerrada por el usuario.

➢ Métodos y Funciones utilizadas

```
def create_widgets(self):
    # Etiqueta para entrada de vértices
    self.vertex_label = tk.Label(self.master, text="Entrada de Vértices:")
    self.vertex_label.grid(row=0, column=0, padx=5, pady=5)
    # Entrada de vértices
    self.vertex_entry = tk.Entry(self.master)
    self.vertex_entry.grid(row=1, column=0, padx=5, pady=5)
    # Etiqueta para entrada de aristas
    self.edge_label = tk.Label(self.master, text="Entrada de Aristas:")
    self.edge_label.grid(row=0, column=1, padx=5, pady=5)
    self.edge_labelll = tk.Label(self.master, text="Grafo:")
    self.edge_label.grid(row=0, column=1, padx=5, pady=5)
    # Entrada de aristas
    self.edge_entry = tk.Entry(self.master)
    self.edge_entry.grid(row=1, column=1, padx=5, pady=5)
    # Botón para agregar aristas
    self.add_button = tk.Button(self.master, text="Agregar", command=self.add_graph)
    self.add_button.grid(row=1, column=2, padx=5, pady=5)
    # Tabla para mostrar vértices y aristas
    self.table_label = tk.Label(self.master, text="Vértices y Aristas:")
    self.table_label.grid(row=2, column=0, columnspan=2, padx=5, pady=5)
    self.treeview = ttk.Treeview(self.master, columns=("Vértices", "Aristas"), show="headings")
    self.treeview.grid(row=3, column=0, columnspan=2, padx=5, pady=5)
    self.treeview.heading("Vértices", text="Vértices")
    self.treeview.heading("Aristas", text="Aristas")
    # Área para mostrar el grafo original
    self.figure = plt.Figure(figsize=(5, 4), dpi=100)
    self.canvas = FigureCanvasTkAgg(self.figure, master=self.master)
    self.canvas.get_tk_widget().grid(row=5, column=0, padx=5, pady=5)

    # Área para mostrar el grafo de búsqueda en anchura
    self.bfs_figure = plt.Figure(figsize=(5, 4), dpi=100)
    self.bfs_canvas = FigureCanvasTkAgg(self.bfs_figure, master=self.master)
    self.bfs_canvas.get_tk_widget().grid(row=5, column=1, padx=5, pady=5)
```

Entonces:

1. Se crean etiquetas (Label) para indicar dónde ingresar los vértices y las aristas del grafo.
2. Se crean campos de entrada (Entry) para que el usuario ingrese los vértices y las aristas.

3. Se crea un botón (Button) que el usuario puede hacer clic para agregar las aristas ingresadas al grafo.
4. Se crea una tabla (Treeview) para mostrar los vértices y aristas ingresados.
5. Se crea un área de dibujo (FigureCanvasTkAgg) para mostrar el grafo original.
6. Se crea otra área de dibujo para mostrar el grafo resultante después de aplicar el algoritmo de búsqueda en anchura.

```
def add_graph(self):
    vertices = self.vertex_entry.get().split(",")
    edges = self.edge_entry.get().split(",")
    for edge in edges:
        v1, v2 = edge.split("--")
        self.original_graph.add_edge(v1.strip(), v2.strip())
        self.edges.append(edge.strip())
    for vertex in vertices:
        self.original_graph.add_node(vertex.strip())
        self.vertices.append(vertex.strip())
    self.update_table()
    self.update_graph()
    self.update_bfs_graph()
```

1. Se obtienen los vértices y aristas ingresados por el usuario desde los campos de entrada y se dividen en listas separadas.
2. Se recorren las aristas ingresadas y se dividen en dos vértices. Luego, se agregan al grafo original utilizando el método add_edge de la biblioteca networkx, y se añaden a la lista de aristas.
3. Se recorren los vértices ingresados y se agregan al grafo original utilizando el método add_node de networkx, y se añaden a la lista de vértices.
4. Se llama a los métodos update_table, update_graph y update_bfs_graph para actualizar la tabla de vértices y aristas, la visualización del grafo original y la visualización del grafo con el algoritmo de búsqueda en anchura aplicado, respectivamente.

y por ultimo:

```
def update_bfs_graph(self):
    self.bfs_figure.clear()
    ax = self.bfs_figure.add_subplot(111)
    nx.draw(self.original_graph, with_labels=True, ax=ax)
    self.bfs_canvas.draw()

def update_graph(self):
    self.graph = self.original_graph.copy()  # Copiar el grafo original al grafo actual
    self.figure.clear()
    ax = self.figure.add_subplot(111)
    nx.draw(self.graph, with_labels=True, ax=ax)
    self.canvas.draw()

def update_table(self):
    # Limpiar tabla
    for item in self.treeview.get_children():
        self.treeview.delete(item)
    # Insertar vértices y aristas en la tabla
    for vertex in self.vertices:
        self.treeview.insert("", "end", values=(vertex, ""))
    for edge in self.edges:
        self.treeview.insert("", "end", values=("", edge))
```

1. update_bfs_graph(self): Este método actualiza la visualización del grafo con el algoritmo de búsqueda en anchura aplicado en el área de dibujo correspondiente. Primero, borra cualquier representación visual previa del grafo con el algoritmo de búsqueda en anchura aplicado. Luego, dibuja el grafo original utilizando la biblioteca matplotlib y lo muestra en el área de dibujo.
2. update_graph(self): Este método actualiza la visualización del grafo original en el área de dibujo correspondiente. Primero, realiza una copia del grafo original para evitar modificar el grafo original. Luego, borra cualquier representación visual previa del grafo original y lo dibuja utilizando la biblioteca matplotlib. Finalmente, muestra el grafo actualizado en el área de dibujo.
3. update_table(self): Este método actualiza la tabla que muestra los vértices y aristas ingresados por el usuario. Primero, limpia la tabla eliminando todos los elementos existentes. Luego, inserta los vértices y aristas almacenados en las listas vertices y edges en la tabla, respectivamente. Cada vértice se inserta en una fila con su respectivo nombre, y cada arista se inserta en una fila con un espacio en blanco para el vértice y el nombre de la arista.