UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

MATEMATICA PARA LA COMPUTACION 2

CATEDRÁTICO: ING. JOSE ALFREDO GONZALEZ DÍAZ

TUTOR ACADÉMICO: BRAYAN MEJÍA

MANUAL TÉCNICO
Proyecto

JAVIER ANDRÉS VELÁSQUEZ BONILLA

CARNÉ: 202307775

AXEL ABRAHAM ROBLES SOLIZ

CARNÉ: 202307805

RODRIGO ANDRES GUAY MINERA

CARNÉ: 202308208

SECCIÓN: A

GUATEMALA, 29 DE OCTUBRE DEL 2,024

ÍNDICE

INTRODUCCIÓN

Este documento técnico explica en detalle los requisitos del sistema, la organización estructural y el funcionamiento interno de un programa de visualización de grafos. Esta aplicación, desarrollada en Python, emplea bibliotecas como tkinter, networkx y matplotlib para integrar algoritmos y proporcionar una interfaz gráfica amigable.

El programa permite al usuario introducir grafos y aplicar algoritmos específicos, tales como el de búsqueda en anchura. Con una presentación clara y dinámica, se pueden observar tanto el grafo de origen como el grafo resultante del algoritmo aplicado, haciéndolo accesible y comprensible.

Este manual está diseñado para ser utilizado por desarrolladores y programadores que busquen entender el funcionamiento del programa o que deseen personalizar el código, ampliándolo o modificándolo según sus propias necesidades.

OBJETIVOS DEL SISTEMA

GENERAL

El propósito central de este manual es brindar una guía completa para que los desarrolladores comprendan a fondo el funcionamiento del programa, facilitando así las modificaciones o ampliaciones que consideren necesarias.

ESPECÍFICOS

- **Desglose de la estructura del código:** Ofrecer una descripción detallada de la organización del código fuente, abarcando la función de cada módulo y archivo, así como la forma en que interactúan entre sí.
- **Explicación del funcionamiento de los algoritmos implementados:** Proporcionar una descripción paso a paso de los algoritmos integrados en el programa, con un enfoque particular en el algoritmo de búsqueda en anchura.

ALCANCES DEL SISTEMA

1. Interfaz Gráfica de Usuario (GUI):

El programa ofrece una interfaz gráfica intuitiva que permite al usuario ingresar grafos, aplicar diversos algoritmos y observar los resultados de forma interactiva.

2. Visualización de Grafos:

La aplicación facilita la representación gráfica de grafos simples que los usuarios ingresan, así como los resultados generados al aplicar algoritmos, como la búsqueda en anchura.

3. Implementación del Algoritmo de Búsqueda en Anchura (BFS):

Este algoritmo, implementado específicamente para el programa, permite explorar los grafos introducidos por el usuario y visualizar el resultado del proceso de búsqueda directamente en la interfaz gráfica.

4. Ingreso de Grafos:

Los usuarios pueden crear grafos de forma manual mediante la adición de vértices y aristas desde la interfaz. No se han incluido opciones avanzadas para la generación automática de grafos o la carga de grafos desde archivos externos.

5. Funcionalidades Básicas:

El enfoque del programa es brindar funciones esenciales para la visualización de grafos y el uso de algoritmos básicos. No se incluye soporte para optimizaciones avanzadas, manipulación de grafos de gran tamaño, ni visualización de datos en tiempo real.

6. Plataforma de Desarrollo:

Desarrollado en Python, el programa hace uso de las bibliotecas tkinter, networkx y matplotlib, y está diseñado para ser ejecutado en plataformas compatibles, tales como Windows, macOS y Linux.

REQUISITOS DEL SISTEMA

- REQUISITOS DE HARDWARE
    1. **Procesador:**

       Para un rendimiento óptimo, se recomienda contar con un procesador de al menos 1 GHz o superior.
    2. **Memoria RAM:**

       Es aconsejable disponer de al menos 2 GB de memoria RAM para asegurar un funcionamiento eficiente del programa, especialmente al trabajar con grafos de tamaño medio.
    3. **Espacio de Almacenamiento:**

       El programa en sí requiere un espacio mínimo de almacenamiento; sin embargo, se sugiere disponer de espacio adicional si se planea guardar grafos en archivos externos.
    4. **Tarjeta Gráfica:**

       No es necesaria una tarjeta gráfica dedicada, ya que el programa no demanda procesamiento gráfico intensivo. Las tarjetas gráficas integradas suelen ser suficientes para ejecutarlo sin problemas.
- REQUISITOS DEL SOFTWARE

1. **Sistema Operativo:**

   El programa es compatible con sistemas operativos como Windows, macOS y Linux.
2. **Python:**

   Es necesario tener instalada una versión de Python 3.x para ejecutar el programa, siendo preferible contar con la última versión estable disponible.

Bibliotecas de Python:

- **tkinter:** La biblioteca tkinter, incluida en la instalación estándar de Python, es requerida para la interfaz gráfica.

- **networkx:** Para el trabajo con grafos, es necesario instalar la biblioteca networkx, disponible mediante el gestor de paquetes pip.
- **matplotlib:** Para la visualización de grafos en la interfaz gráfica, es imprescindible instalar matplotlib, también accesible a través de pip.

Entorno de Desarrollo Integrado (IDE):

Aunque no es obligatorio usar un IDE específico, se recomienda emplear uno como Visual Studio Code, PyCharm o Jupyter Notebook para facilitar tanto el desarrollo como la depuración del código.

1.

DESCRIPCIÓN DE LA SOLUCIÓN

La propuesta consiste en un programa informático diseñado para permitir la visualización interactiva de grafos y la aplicación de algoritmos sobre estos. Esta aplicación, desarrollada en Python, hace uso de las bibliotecas tkinter, networkx y matplotlib para ofrecer una interfaz gráfica intuitiva y brindar funciones de manipulación de grafos.

LÓGICA DEL PROGRAMA

Librerías Utilizadas

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

- **tkinter:** Biblioteca de Python usada para crear la interfaz gráfica, permitiendo la interacción de los usuarios con el programa.
- **networkx:** Herramienta en Python para la creación, manipulación y análisis de grafos. Es utilizada para modelar grafos y aplicar algoritmos, como el de búsqueda en anchura.
- **matplotlib:** Biblioteca empleada para representar visualmente los grafos en la interfaz gráfica de la aplicación.

Variables Globales de la Clase

```
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="#003366")  # Cambiar el color de fondo a azul oscuro

        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:** Elemento principal de la interfaz gráfica sobre el que se construyen todos los demás elementos.
2. **original_graph:** Objeto Graph de networkx que representa el grafo ingresado originalmente por el usuario.

3. **graph:** Otro objeto Graph de networkx, usado para representar el grafo sobre el cual se aplican los algoritmos.
4. **vertices:** Lista que almacena los vértices del grafo proporcionado por el usuario.
5. **edges:** Lista que contiene las aristas del grafo introducido.
6. **canvas:** Área de dibujo donde se visualiza el grafo original.
7. **bfs_canvas:** Área específica de dibujo para mostrar el grafo después de aplicar el algoritmo de búsqueda en anchura.
8. **create_widgets():** Método que inicializa y organiza los elementos de la interfaz gráfica dentro de la ventana principal.

Función Principal

1. **if name == "main":** Condición que verifica si el script se ejecuta directamente como un programa, en lugar de ser importado como módulo. Si es así, ejecuta el código dentro del bloque.
2. **root = tk.Tk():** Crea una instancia de Tk() de tkinter, que actúa como la ventana principal de la aplicación.
3. **app = GraphVisualization(master=root):** Instancia de la clase GraphVisualization, representando la aplicación de visualización de grafos y usando root como ventana principal.
4. **app.mainloop():** Inicia el bucle de eventos principal de la interfaz gráfica para que el programa responda a las interacciones de los usuarios hasta que se cierre la ventana principal.

Métodos y Funciones Utilizadas

1. Se crean **etiquetas (Label)** para indicar los campos donde se ingresarán vértices y aristas del grafo.

2. Se generan **campos de entrada (Entry)** donde el usuario puede ingresar los vértices y aristas.
3. Se añade un **botón (Button)** para que el usuario añada las aristas ingresadas al grafo.
4. Se implementa una **tabla (Treeview)** para mostrar los vértices y aristas agregados.
5. Se habilita un **área de dibujo (FigureCanvasTkAgg)** para la visualización del grafo original.
6. Se dispone de otra área de dibujo para representar el grafo con el algoritmo de búsqueda en anchura aplicado.

Operaciones de Manipulación de Grafos

1. Los vértices y aristas ingresados por el usuario se obtienen de los campos de entrada y se separan en listas.
2. Las aristas se procesan dividiéndolas en vértices y agregándolas al grafo original usando add_edge de networkx. Luego, se añaden a la lista de aristas.
3. Los vértices ingresados se agregan al grafo original mediante add_node de networkx y también se almacenan en la lista de vértices.
4. Se ejecutan los métodos update_table, update_graph, y update_bfs_graph para actualizar la tabla, la visualización del grafo original y la del grafo con el algoritmo de búsqueda en anchura, respectivamente.

Actualización de Visualizaciones y Tablas

1. **update_bfs_graph(self):** Actualiza la representación gráfica del grafo después de aplicar el algoritmo de búsqueda en anchura. Primero limpia cualquier visualización previa, luego usa matplotlib para redibujar el grafo en el área asignada.

2. **update_graph(self):** Refresca la visualización del grafo original. Realiza una copia del grafo para evitar cambios accidentales en el original, borra la visualización previa y muestra el grafo actualizado en el área de dibujo.
3. **update_table(self):** Actualiza la tabla de vértices y aristas ingresados por el usuario, limpiando elementos previos y luego insertando los vértices y aristas de las listas vértices y edges, respectivamente.