UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
FACULTAD DE INGENIERÍA
MATEMÁTICA PARA COMPUTACIÓN
CATEDRÁTICO: ING. JOSÉ ALFREDO GONZÁLEZ DÍAZ
TUTOR ACADÉMICO: BRAYAN MEJÍA

MANUAL TÉCNICO

KEVIN EMMANUEL SALAZAR MONTERROSO
CARNÉ: 202300669
SECCIÓN: A

GUATEMALA, 25 DE OCTUBRE DEL 2,024

ÍNDICE

INTRODUCCIÓN

Este manual técnico está diseñado para brindar una guía completa y detallada sobre el funcionamiento interno del programa, incluyendo su estructura de código, los procedimientos técnicos para su ejecución y los requisitos de sistema necesarios. Su propósito es proporcionar a los desarrolladores y usuarios técnicos la información necesaria para comprender, modificar y optimizar el programa de generación y análisis de grafos, así como las búsquedas por BFS y DFS.

OBJETIVOS

1. GENERAL

1.1. Proveer una referencia técnica clara y detallada que facilite la comprensión y uso eficiente del código, orientando al usuario en la operación, mantenimiento y actualización del programa.

2. ESPECÍFICOS

2.1. Describir de manera precisa y paso a paso la estructura y funcionalidad del código, facilitando el entendimiento de sus módulos y procedimientos clave.

2.2. Explicar los métodos para modificar y ajustar el código, incluyendo ejemplos y lineamientos de buenas prácticas para optimizar el programa según las necesidades del usuario.

ALCANCES DEL SISTEMA

El objetivo de este manual es proporcionar una guía exhaustiva que permita al usuario comprender y manipular el código de la aplicación de generación y análisis de grafos, así como los algoritmos de búsqueda BFS y DFS. Este

documento cubre el funcionamiento detallado de cada módulo, los pasos para ejecutar correctamente el programa, y las instrucciones para realizar modificaciones en el código, ajustarlo a nuevas especificaciones o solucionar posibles errores. Además, se pretende que el usuario pueda optimizar el rendimiento del programa y extender su funcionalidad de forma autónoma, brindándole un soporte técnico integral y accesible.

ESPECIFICACIÓN TÉCNICA

- **REQUISITOS DE HARDWARE**

Para el desarrollo, mantenimiento y posible mejora de esta aplicación, el programador necesita un equipo con las siguientes especificaciones mínimas de hardware:

- ✓ **Procesador**: Intel Core i5 de 8ª generación o AMD Ryzen 5, o superior.
- ✓ **Memoria RAM**: Al menos 8 GB (16 GB recomendados para un rendimiento óptimo).
- ✓ **Almacenamiento**: Mínimo 500 MB de espacio libre para la aplicación y sus archivos generados, más espacio adicional para el sistema operativo y demás aplicaciones.
- ✓ **Monitor**: Resolución mínima de 1920x1080 para mejor visualización de la interfaz y gráficos.

- **REQUISITOS DE SOFTWARE**

Para trabajar en el desarrollo y futuras mejoras de esta aplicación, se deben cumplir los siguientes requisitos de software:

- **Sistema Operativo**: Windows 10 o superior, o distribuciones de Linux compatibles.
- **Entorno de Desarrollo Integrado (IDE)**: Un IDE o editor de texto avanzado, como Visual Studio Code, PyCharm o cualquier otro que permita la integración de Python y Fortran.
- **Lenguajes**:
    - **Python**: Versión 3.8 o superior, para la interfaz gráfica con Tkinter y el control del flujo del programa.
    - **Fortran**: Compatible con el estándar Fortran 90 o superior, requerido para el backend de análisis de datos y algoritmos de búsqueda.
- **Bibliotecas de Python**:
    - **Tkinter**: Incluida con la instalación estándar de Python para el desarrollo de la interfaz gráfica.
    - **Graphviz**: Instalado y configurado en el PATH del sistema para la visualización de grafos.
    - **subprocess**: Para la comunicación entre Python y Fortran.
- **Compilador de Fortran**: Necesario para compilar y ejecutar el código Fortran. Se recomienda GNU Fortran (gfortran).
- **Graphviz**: Para la generación y visualización de grafos, con las rutas configuradas en el sistema.
- **Documentación**: Un software para edición de documentos (ej. Microsoft Word, Google Docs o LaTeX) para realizar y actualizar documentación técnica.

DESCRIPCIÓN DE LA SOLUCIÓN

La solución fue desarrollada a partir de un análisis detallado de los requerimientos del enunciado, enfocados en crear una aplicación capaz de gestionar y visualizar grafos con algoritmos de búsqueda BFS y DFS. Primero, identificamos la necesidad de una interfaz intuitiva para usuarios que permita el ingreso de vértices y aristas, así como una representación visual de los grafos generados. Se eligió Python con Tkinter para la interfaz gráfica debido a su facilidad de uso y personalización, y se integró el uso de Graphviz para el manejo visual de grafos, aprovechando su capacidad para generar representaciones gráficas de alta calidad y su compatibilidad con Python.

Por otro lado, se decidió usar Fortran para el backend debido a su eficiencia en el manejo de estructuras de datos y operaciones de análisis, que complementan las necesidades de procesamiento de las funciones de búsqueda BFS y DFS. La comunicación entre Python y Fortran se implementó a través de subprocess, permitiendo una interacción fluida entre el frontend y el backend. Este enfoque modular nos permitió desarrollar una aplicación con buena separación de responsabilidades y alta eficiencia, ajustándose a los objetivos establecidos en los requerimientos iniciales.

LÓGICA DEL PROGRAMA

❖ NOMBRE DE LA CLASE

graphviz > graphiz.py

```
1   import tkinter as tk
2   from tkinter import messagebox
3   from PIL import Image, ImageTk
4   import subprocess
5   import os
6   from collections import deque
7
8   class GraphApp:
9       def __init__(self, master):
10          self.master = master
11          master.title("Generador de Gráficos No Dirigidos con Pesos")
12
```

➢ Librerías

- **Tkinter**: Esta librería se utiliza para crear la interfaz gráfica de usuario (GUI) del programa. Proporciona herramientas para diseñar ventanas, botones, y otros elementos visuales que permiten al usuario interactuar con la aplicación de manera intuitiva.
- **Graphviz**: Se emplea para la visualización de grafos. Permite generar gráficos de alta calidad a partir de la representación de los datos de los grafos, facilitando una visualización clara y efectiva de la estructura y relaciones entre los vértices y aristas.
- **subprocess**: Esta librería se usa para ejecutar comandos de sistema y scripts externos. En este caso, se utiliza para compilar y ejecutar el código Fortran desde el entorno de Python, permitiendo la integración del backend con el frontend.

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

self.vertices = [] # Almacena los vértices del grafo

self.aristas = []   # Almacena las aristas del grafo

self.peso_vertices = {} # Diccionario que almacena los pesos de los vértices

Las variables globales son utilizadas para mantener el estado actual del grafo. self.vertices y self.aristas son listas que almacenan los elementos ingresados por el usuario, mientras que self.peso_vertices se encarga de almacenar los pesos asociados a cada vértice, facilitando su uso en las funciones de búsqueda.

➢ **Función Main**

La función main es el punto de entrada de la aplicación. Su función es inicializar la interfaz gráfica y configurar el bucle principal de eventos, permitiendo que el programa responda a las interacciones del usuario.

python
def main():
app = GrafoApp() # Crear una instancia de la clase GrafoApp
app.run() # Ejecutar el método que inicia la interfaz gráfica

➢ **Métodos y Funciones utilizadas**

❖ **`agregar_vertice()`**

➢ **Descripción**: Este método permite al usuario agregar un nuevo vértice al grafo. Recibe como parámetro el nombre del vértice y lo almacena en la lista `self.vertices`.

```
python
def agregar_vertice(self, nombre):
```

```
        self.vertices.append(nombre)  # Agrega el
    nombre del vértice a la lista
```

- **`crear_arista()`**
  - ➢ **Descripción**: Esta función se encarga de crear una arista entre dos vértices. Verifica si ambos vértices existen en la lista antes de añadir la arista a `self.aristas`.

```
python
def crear_arista(self, vertice1, vertice2):
    if vertice1 in self.vertices and vertice2
in self.vertices:
        self.aristas.append((vertice1,
vertice2))  # Agrega la arista a la lista
```

- **`dibujar_grafo()`**
  - ➢ **Descripción**: Este método utiliza la librería Graphviz para visualizar el grafo en una ventana nueva. Genera un archivo de imagen que representa la estructura del grafo basado en los vértices y aristas.

```
python
Copiar código
def dibujar_grafo(self):
    # Código para generar y mostrar la
visualización del grafo
```

Estas funciones son fundamentales para la interacción del usuario con la aplicación y permiten gestionar la creación y visualización de grafos de manera eficiente.