UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

MATEMATICAS PARA COMPUTACION 2

CATEDRÁTICO: ING. JOSÉ ALFREDO GONZÁLES

TUTOR ACADÉMICO: ROBERTO CARLOS GÓMEZ DONIS

MANUAL TECNICO
PROGRAMA DE GRAFOS Y ALGORITMO DE PROFUNDIDAD Y ANCHURA

Christian Fernando Boror Romero	202307765
German Estiven Barrera Tojin	202201588
Samuel Alejandro De León Maldonado	202111587
Ana Paola García de Paz	202300560
Brayan Oswaldo Rojas Morales	202108307

GUATEMALA, 25 DE OCTUBRE DEL 2,024

ÍNDICE

OBJETIVOS DEL SISTEMA

GENERAL

Desarrollar un sistema de interfaz gráfica en Python utilizando el módulo tkinter para la gestión y visualización de datos relacionados con grafos, permitiendo al usuario ingresar y almacenar vértices y aristas, así como realizar búsquedas en anchura y profundidad sobre el grafo previamente generado generado.

ESPECÍFICOS

- Objetivo 1: Generar ventanas de entrada de datos para agregar vértices y aristas, permitiendo al usuario introducir información como el nombre del vértice, su posición en la fila y columna, y los nombres de los vértices que componen una arista.
- Objetivo 2: Desarrollar funciones que reciban los datos introducidos por el usuario, los almacenen correctamente mediante clases como CrearVertice y CrearArista, y cierren las ventanas de entrada tras completar la operación de inserción de datos..

INTRODUCCIÓN

Este manual técnico ofrece una guía completa para desarrollar y utilizar un sistema de interfaz gráfica en Python con el módulo tkinter. Este sistema permite gestionar y visualizar datos de grafos, posibilitando que el usuario introduzca, almacene y manipule vértices y aristas, además de ejecutar búsquedas en anchura y profundidad en el grafo creado. Con una interfaz intuitiva y funciones claramente definidas, el manual tiene como objetivo proporcionar a los desarrolladores y usuarios una comprensión del sistema, además de instrucciones detalladas para su implementación y uso eficaz.

INFORMACIÓN DEL SISTEMA

Desarrollado en Python con el módulo tkinter, este sistema ofrece una interfaz gráfica intuitiva y funcional para la gestión de grafos. Permite al usuario ingresar y almacenar vértices y aristas, además de realizar búsquedas en anchura y profundidad. Su estructura modular facilita tanto el mantenimiento como la expansión, mientras que la implementación de clases y métodos claramente definidos asegura un código organizado y fácil de entender.

EXPLORACIÓN DETALLADA DE LOS CÓDIGOS FUENTE

1. AlmacenarDatos, CrearVertice y CrearArista. La clase AlmacenarDatos gestiona la información de vértices y aristas, con métodos para agregar, obtener y vaciar elementos de listas. Las clases CrearVertice y CrearArista son modelos simples para representar vértices y aristas, con métodos__init__() que inicializan atributos específicos para cada uno.

```
clase AlmacenarDatos:
    definición __en eso__(ser):
        ser.Vértices =[]
        ser.Aristas =[]

    definición LlenarLista(ser,vértice):
        ser.Vértices.adjuntar(vértice)

    definición RetornarÚltimoVértice(ser):
        si ser.Vértices:
            ultimo_vertice = ser.Vértices[-1]
            devolver ultimo_vertice.nombre,ultimo_vertice.fila,ultimo_vertice.columna
        demás:
            devolver Ninguno

    definición RetornarLista(ser):
        devolver ser.Vértices

    definición LlenarListaAristas(ser,Arista):
        ser.Aristas.adjuntar(Arista)

    definición RetornarUltimaArista(ser):
        si ser.Aristas:
            ultima_arista = ser.Aristas[-1]
            devolver ultima_arista.NombreVérticeInicial,ultima_arista.NombreVérticeFinal
        demás:
            devolver Ninguno
```

2. La clase BusquedaEnAnchura se encarga de realizar búsquedas en anchura o profundidad en un grafo representado por una lista de aristas. En su método__init__() se inicializan variables como la instancia de AlmacenarDatos, la lista de aristas y el tipo de búsqueda. BuscarTodosLosVertices identifica todos los vértices presentes en el grafo y los almacena. DefinirConexiones crea una lista de conexiones entre los vértices. CrearListaParaGrafoAnchura() y CrearListaParaGrafoProfundidad() realizan la búsqueda en anchura o profundidad respectivamente. Además, hay métodos para buscar aristas y aristas iniciales durante la búsqueda. Las clases ConexionVertices y CrearNuevasAristas ayudan a representar conexiones entre vértices y a crear nuevas aristas durante la búsqueda, respectivamente.

```
clase BusquedaEnAnchura:
    definición __en eso__(ser,instancia,lista,tipo):
        ser.AlmacenarDatos = instancia
        ser.ListaAristas = lista
        ser.TipoBusqueda = tipo
        ser.Vértices =[]
        ser.ListaConexiones =[]
        ser.NuevasAristas =[]
        ser.VérticesTotales =[]
        ser.VérticeAnterior = ""
        ser.BuscarTodosLosVertices()

    definición BuscarTodosLosVertices(ser):
        para i en ser.ListaAristas:
            si len(ser.Vértices)> 0:
                si i.NombreVérticeInicial no en ser.Vértices:
                    ser.Vértices.adjuntar(i.NombreVérticeInicial)
                si i.NombreVérticeFinal no en ser.Vértices:
                    ser.Vértices.adjuntar(i.NombreVérticeFinal)
            demás:
                ser.Vértices.adjuntar(i.NombreVérticeInicial)
                ser.Vértices.adjuntar(i.NombreVérticeFinal)

        ser.Vértices.clasificar()
        para k en ser.Vértices:
            ser.VérticesTotales.adjuntar(cadena(k))
```

3. La clase InterfazGrafica define una interfaz gráfica utilizando el módulo customtkinter, donde se crea una ventana con diversos elementos como cuadros de texto, botones y lienzo para dibujar. Se proporcionan métodos para agregar texto a los cuadros de texto, dibujar vértices y aristas en el lienzo, limpiar la interfaz, obtener datos de vértices y aristas, y ejecutar algoritmos de búsqueda en anchura o profundidad. La ventana se

inicializa con una configuración específica y se ejecuta el bucle principal de la interfaz. Cada vez que se agrega un vértice o una arista, se actualiza la visualización en el lienzo correspondiente, y al ejecutar los algoritmos de búsqueda, se muestran los resultados en el lienzo de salida.

```
importar personalizado como ctk
importar tkinter
importar tkinter como tk
de SolicitarDatosVertice importar *
de SolicitarDatosAristas importar *
de AlmacenarDatos importar *
de Busqueda importar *

clase InterfazGráfica:
    definición __en eso__(ser):
        ser.ConfiguraciónInicial()
        ser.CrearCuadros()
        ser.CrearBotones()
        ser.dibujar_cuadricula()
        ser.AlmacenamientoDeDatos = AlmacenarDatos()
éis

    definición ConfiguraciónInicial(ser):
        ser.VentanaInterfaz = ctk.CTk()
        ctk.establecer_modo_apariencia("luz")
        ctk.set_default_color_theme("verde")
        ser.VentanaInterfaz.redimensionable(0,0)
        ser.VentanaInterfaz.título("Grafos")
        ser.ancho_ventana = 1200
        ser.alto_ventana = 675
        ser.ancho_pantalla = ser.VentanaInterfaz.winfo_screenwidth()
        ser.alto_pantalla = ser.VentanaInterfaz.winfo_screenheight()
```

4. Este bloque de código utiliza PyInstaller para empaquetar un programa Python en un archivo ejecutable independiente. Se inicia con la configuración de una instancia de la clase Analysis para analizar los módulos y sus dependencias. Luego, se definen las opciones de empaquetado, como la inclusión de archivos binarios y datos, la exclusión de ciertos módulos, la compresión con UPX, y la configuración del archivo ejecutable final, como el nombre, la depuración y la habilitación de la consola. Al final, PyInstaller crea el archivo ejecutable con todas las especificaciones dadas.

```
# -*- modo: python; codificación: utf-8 -*-


a = Análisis(
    ['InterfazGráfica.py'],
    patex=[],
    binarios=[],
    datos=[],
    importaciones ocultas=[],
    camino de gancho=[],
    configuración de ganchos={},
    runtime_hooks=[],
    excluye=[],
    sin archivo=FALSO,
    optimizar=0,
)
pyz = PIZ(a.puro)

exe = EXE(
    pyz,
    a.guiones,
    a.binarios,
    a.datos,
    [],
    nombre='InterfazGráfica',
    depurar=FALSO,
    gestor de arranque_ignorar_señales=FALSO,
```

5. Este código define una clase VentanaCrearArista que crea una ventana de interfaz gráfica usando el módulo tkinter para solicitar datos de una arista. La ventana solicita al usuario que ingrese el nombre del vértice inicial y final, luego, al presionar un botón, los datos ingresados se envían a una instancia de AlmacenarDatos para ser almacenados como una nueva arista. Una vez enviado, la ventana se cierra.

```
importar tkinter como tk
de AlmacenarDatos importar CrearArista

clase VentanaCrearArista:
    definición __en eso__(ser):
        ser.ConfiguraciónInicial()
        ser.CreandoEspacios()

    definición Recibir Instancia(ser,instancia):
        ser.AlmacenarDatos = instancia

    definición ConfiguraciónInicial(ser):
        ser.VentanaInterfaz = tk.tk()
        ser.VentanaInterfaz.título("Datos para vértice")
        ser.ancho_ventana = 300
        ser.alto_ventana = 230
        ser.ancho_pantalla = ser.VentanaInterfaz.winfo_screenwidth()
        ser.alto_pantalla = ser.VentanaInterfaz.winfo_screenheight()
        ser.posición_x =(ser.ancho_pantalla - ser.ancho_ventana)// 2
        ser.posicion_y =(ser.alto_pantalla - ser.alto_ventana)// 2
        ser.VentanaInterfaz.geometría(F"{ser.ancho_ventana}X{ser.alto_ventana}+{ser.posición_x}+{ser.posicion_y}")

    definición CreandoEspacios(ser):
        ser.etiqueta_vertice_inicial = tk.Etiqueta(maestro=ser.VentanaInterfaz,texto="Ingrese el nombre del vértice IN
        ser.etiqueta_vertice_inicial.embalar(padx=20,arroz=(20,0))
        ser.campo_vertice_inicial = tk.Entrada(maestro=ser.VentanaInterfaz,ancho=30)
        ser.campo_vertice_inicial.embalar(padx=20,arroz=(0,10))
```

6. Este código define una clase VentanaCrearVertice que crea una ventana de interfaz gráfica utilizando tkinter para solicitar datos de un vértice, incluyendo su nombre, fila y columna. Una vez que el usuario completa los campos y presiona el botón "Agregar vértice", los datos ingresados se envían a una instancia de AlmacenarDatos para ser almacenados como un nuevo vértice. Luego de enviar los datos, la ventana se cierra.

```
importar tkinter como tk
de AlmacenarDatos importar CrearVértice

clase VentanaCrearVértice:
    definición __en eso__(ser):
        ser.ConfiguraciónInicial()
        ser.CreandoEspacios()

    definición Recibir Instancia(ser,instancia):
        ser.AlmacenarDatos = instancia

    definición ConfiguraciónInicial(ser):
        ser.VentanaInterfaz = tk.tk()
        ser.VentanaInterfaz.título("Datos para vértice")
        ser.ancho_ventana = 300
        ser.alto_ventana = 350
        ser.ancho_pantalla = ser.VentanaInterfaz.winfo_screenwidth()
        ser.alto_pantalla = ser.VentanaInterfaz.winfo_screenheight()
        ser.posición_x =(ser.ancho_pantalla - ser.ancho_ventana)// 2
        ser.posicion_y =(ser.alto_pantalla - ser.alto_ventana)// 2
        ser.VentanaInterfaz.geometría(F"{ser.ancho_ventana}X{ser.alto_ventana}+{ser.posición_x}+{ser.posicion_y}")

    definición CreandoEspacios(ser):
        ser.etiqueta_nombre = tk.Etiqueta(maestro=ser.VentanaInterfaz,texto="Ingrese el nombre del vértice:")
        ser.etiqueta_nombre.embalar(padx=20,arroz=(20,0))
        ser.campo_nombre = tk.Entrada(maestro=ser.VentanaInterfaz,ancho=30)
        ser.campo_nombre.embalar(padx=20,arroz=(0,10))
```