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

MANUAL TÉCNICO

GRUPO 6
Samuel Gerardo Maldonado López 202307470
Álvaro Gabriel Ceballos Gil 202300673
Daniel Alejandro Portillo Garcia 202307534
José Emanuel Monzón Lémus 202300539

GUATEMALA, 29 DE ABRIL DEL 2,024

ÍNDICE

INTRODUCCIÓN

El fin de este manual es, exponer la lógica general de todos los procesos del programa de grafos, que hace en aspectos generales el Código para que funcione, describir Código funciones, variables y entre más aspectos de los sistemas que se utilizaron dentro del programa Generador de grafos.

OBJETIVOS

1. GENERAL

1.1. Exponer las funciones del Código del sistema generador de grafos de forma general.

2. ESPECÍFICOS

2.1. Objetivo 1: Explicar el funcionamiento de funciones utilizadas dentro de código

2.2. Objetivo 2: Indicar qué es lo que se hará en este manual relacionado a su código

2.3. Objetivo 3: Indicar el uso de librerías externas y el porqué de su uso.

2.4. Objetivo 5: Explicar el orden de la lógica que sigue el programa según las necesidades del usuario.

2.5. Objetivo 6: mostrar los requisitos del programa.

ALCANCES DEL SISTEMA

Este manual explica el funcionamiento detallado en sus aspectos generales, sobre cada método, función, condicional y demás procesos sobre el funcionamiento del sistema de generador de grafos y algoritmos de búsqueda de ancho a lo largo en el cual se darán características de la lógica utilizando el entorno de programación python. Para exponer cómo es posible poner en práctica conocimientos de programación en la lógica en el uso de problemas de manejo de grafos y algoritmos de búsqueda para ejemplificar y demostrar la teoría de grafos en una interfaz gráfica.

ESPECIFICACIÓN TÉCNICA

- **REQUISITOS DE HARDWARE**
    - Para continuar un futuro desarrollo del programa los requisitos utilizados para el desarrollo del programa son los siguientes:

    - Computadora Windows, procesador, mínimo corei5, 7th generación, con memoria RAM de 4 gb como mínimo.

- **REQUISITOS DE SOFTWARE**
    - Requisitos de software necesario y obligatorios son los siguientes:

- Sistema operativo Windows 10/11.
- Microsoft Visual Studio Code 64 bits.
- Python 3.12.3/3.11.3 64 bits
- Librerias graficas tkinter, matplotlib y networkx

DESCRIPCIÓN DE LA SOLUCIÓN

- El enunciado del problema solicita la creación de un sistema de entorno grafico que se encargue de generar vértices y aristas de un grafo, para posteriormente trazar el grafo y formar su algoritmo de búsqueda a lo ancho y a lo largo. Para ello se opto por el entorno de programación Python en visual studio code para facilitar la lógica del programa, para posteriormente utilizar funciones dentro del programa de filtrado por orden para generar la búsqueda de cada grafo. Esto por medio de librerías de interfaz grafica que facilitaran la vista del proyecto al momento de ejecutarse.

LÓGICA DEL PROGRAMA

➢ Librerías

Librerías encargadas de importar las funciones graficas dentro de Python, se usaron para generar grafos e interfaz gráfica.

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

➢ Funciones

- ❖ Funciones para crear el grafo, parámetros del grafo, agregar vértices y aristas.

```
#Se crea grafo
grafo = nx.Graph()

#Se crea funcion que dibujará el grafo
def nuevoGrafo():
    ax.set_visible(True)
    ax.clear()
    pos = nx.spring_layout(grafo)
#Se asignan los parámetros que tendrá el grafo
    nx.draw(grafo, pos, with_labels=True, node_size=650, node_color="#5DD8CC", font_size=16, font_color="black", ax
    ax.set_xticks([])
    ax.set_yticks([])
    espacio.draw()

#Se crea función para agregar vertice
def nuevoVertice(nombre_vertice):
    grafo.add_node(nombre_vertice)
    nuevoGrafo()


#Se crea función para agregar aristas
def nuevaArista(arista):
   #Se especifica la forma en la que se debe de ingresar la arista
    vertice = arista.split("-")
    if len(vertice) != 2:
        messagebox.showerror("Error", "Separar los vertice con - para poder generar aristas")
        return
    v1, v2 = vertice
    if v1 not in grafo.nodes or v2 not in grafo.nodes:
        messagebox.showerror("Error", "Verificar que existan los vértices")
        return
    grafo.add_edge(v1, v2)
    nuevoGrafo()
```

- ❖ Funciones para lo algoritmos de búsqueda a lo largo y a lo ancho.

```python
#Funcion para el algoritmo de busqueda a anchura
def ancho():
    if not grafo:
        messagebox.showerror("Error", "Debe de ingresar el grafo para aplicar el algoritmo de búsqueda")
        return
    # Creaa una un segundo grafo para mostrar el resultado a lo ancho
    GrafoAncho = grafo.copy()
    # Aplicar el algoritmo de búsqueda en anchura
    AlgAncho = nx.bfs_tree(GrafoAncho, source=list(GrafoAncho.nodes)[0])
    #Ordena de forma ascendente los nodos
    ordenar = sorted(GrafoA
    #Se dibuja el resultad
    pos = nx.spring_layout(AlgAncho)
    AlgAx.clear()
    #Se asignan los parámetros del grafo a anchura
    nx.draw(AlgAncho, pos,  nodelist=ordenar, with_labels=True, node_size=300, node_color="#87CF79", font_size=10, font_colo
    AlgAx.set_visible(True)
    AlgAx.set_xticks([])
    AlgAx.set_yticks([])
    espacioR.draw()
```

(variable) AlgAncho: Any

```python
#Se crea funcion para aplicar el algortimo de busqueda a lo largo
def largo():
    if not grafo:
        messagebox.showerror("Error", "Debe de ingresar el grafo para aplicar el algoritmo de búsqueda")
        return

  # Creaa una un segundo grafo para mostrar el resultado a lo largo
    GrafoLargo = grafo.copy()
    #Se aplica el algorithm de busqueda a lo largo
    AlgLargo = nx.dfs_tree(GrafoLargo, source=list(GrafoLargo.nodes)[0])
    #Se ordenan los nodos
    ordenar = sorted(GrafoLargo.nodes())
    #Se dibuja el resultado
    pos = nx.spring_layout(AlgLargo)
    AlgAx.clear()
    nx.draw(AlgLargo, pos, nodelist=ordenar, with_labels=True, node_size=300, node_color="#87CF79", font_size=10, font_color
    AlgAx.set_visible(True)
    AlgAx.set_xticks([])
    AlgAx.set_yticks([])
    espacioR.draw()
```

❖ Ventana inicial donde se llaman las funciones y se crea la interfaz Grafica.

```python
#Se crea la ventana inicial
ventana1 = tk.Tk()
ventana1.title("Proyecto Final Mate Para Computacion 1 - Grupo 6")
ventana1.geometry("1000x600")
ventana1.configure(bg="#79DDC6")

#Se crea frame
JFrame = tk.Frame(ventana1, bg="#79DDC6")
JFrame.pack(fill=tk.BOTH, expand=True)

#Se crea espacio donde se dibujará el grafo inicial
espacio = FigureCanvasTkAgg(plt.figure(figsize=(4, 4), facecolor="#79DDC6"), master=JFrame)
espacio.get_tk_widget().pack(side=tk.LEFT, fill=tk.BOTH, expand=True)
espacioResultado = tk.Frame(ventana1, bg="#79DDC6")
espacioResultado.pack(side=tk.RIGHT, fill=tk.BOTH, expand=True)
Figura = plt.figure(figsize=(4, 4), facecolor="#79DDC6")
AlgAx = Figura.subplots()
AlgAx.set_visible(False)
espacioR = FigureCanvasTkAgg(Figura, master=espacioResultado)
espacioR.get_tk_widget().pack(side=tk.TOP, fill=tk.BOTH, expand=True)


#Se crea frame para las funcionalidades
JFrameBtn = tk.Frame(ventana1, bg="#967D3A")
JFrameBtn.pack(fill="both", expand=1)
JFrameBtn.config(width=480,height=420)

xd = tk.Label(JFrameBtn, text="Para generar los vertice, debe de ingresas la letra que este llevará, \n y para las aristas d
xd.pack()
entry = tk.Entry(JFrameBtn, width=40, bg="white", fg="#6B1717")
entry.place(x=5,y=58)
```

LINK DEL VIDEO

https://www.youtube.com/watch?v=IuW6qhSKj5E