UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

MATEMATICA PARA COMPUTACION 2

MANUAL TÉCNICO

SECCIÓN: A

GUATEMALA, 28 DE ABRIL DEL 2,024

OBJETIVOS

1. GENERAL

1.1. Desarrollar un manual técnico que facilite la comprensión y aplicación de la teoría de grafos en programación,

2. ESPECÍFICOS

2.1. Objetivo 1: Explicar la utilidad y relevancia de la teoría de grafos en el contexto de las Ciencias de la Computación, destacando su papel en la modelación de fenómenos discretos y la comprensión de las estructuras de datos.

2.2. Objetivo 2: Guiar al lector en la aplicación práctica de la teoría de grafos mediante la implementación de algoritmos utilizando diferentes lenguajes de programación, con énfasis en la resolución de problemas específicos.

ALCANCES DEL SISTEMA

El objetivo de este manual es proporcionar a los usuarios la orientación necesaria para comprender, aplicar y visualizar la teoría de grafos en el contexto de la programación. Esto implica ofrecer instrucciones detalladas para implementar algoritmos específicos, como la búsqueda en anchura y la búsqueda en profundidad, utilizando distintos lenguajes de programación. Además, el manual tiene como objetivo guiar a los usuarios en la creación de un programa con una interfaz gráfica que les permita ingresar grafos, observar su estructura y aplicar los algoritmos mencionados, con el fin de demostrar de manera práctica y visual su funcionamiento.

ESPECIFICACIÓN TÉCNICA

- **REQUISITOS DE HARDWARE**
  - Procesador: Un procesador de al menos 1 GHz debería ser suficiente para ejecutar el programa sin problemas.
  - Memoria RAM: Se recomienda al menos 1 GB de RAM para ejecutar el programa de manera fluida.

- **REQUISITOS DE SOFTWARE**
  - Framework de Ejecución de Python: Aunque el archivo .exe contiene todo lo necesario para ejecutar el programa en un sistema Windows, es posible que requiera ciertas dependencias del entorno de ejecución de Python. Si estas dependencias no están incluidas en el archivo .exe, el usuario puede necesitar tener instalado Python en su sistema.
  - Librerías y Dependencias: El programa utiliza librerías específicas de Python, como Tkinter para la interfaz gráfica y matplotlib para visualización, es posible que estas dependencias deban estar instaladas en el sistema del usuario para que el programa se ejecute correctamente.
  - Sistema operativo: El archivo .exe generado desde Python debería ser compatible con las versiones modernas de Windows, como Windows 7, Windows 8, Windows 10, y posteriores

DESCRIPCIÓN DE LA SOLUCIÓN

La solución propuesta para el proyecto implica el desarrollo de un programa informático en Python que permita a los usuarios ingresar grafos, aplicar algoritmos como la búsqueda en anchura y la búsqueda en profundidad, y visualizar los resultados de manera gráfica a través de una interfaz gráfica de usuario (GUI).

El programa constará de los siguientes componentes principales:

1. Interfaz Gráfica de Usuario (GUI):
   - La GUI permitirá a los usuarios ingresar vértices y aristas para construir un grafo.
   - Proporcionará opciones para seleccionar el algoritmo deseado (búsqueda en anchura, búsqueda en profundidad).
   - Mostrará visualmente el grafo original ingresado por el usuario, así como el grafo resultante después de aplicar el algoritmo seleccionado.
2. Funcionalidad de Ingreso de Datos:
   - Permitirá al usuario ingresar vértices y aristas de manera intuitiva a través de la GUI.
   - Validará la entrada del usuario para garantizar la correcta construcción del grafo.
3. Implementación de Algoritmos:
   - Incluirá la implementación de los algoritmos de búsqueda en anchura y búsqueda en profundidad para operar en el grafo ingresado por el usuario.
   - Los algoritmos se ejecutarán en el grafo para encontrar la solución deseada según el algoritmo seleccionado.
4. Visualización Gráfica:
   - Utilizará herramientas gráficas para representar visualmente el grafo original y el grafo resultante después de aplicar el algoritmo.
   - La visualización ayudará a los usuarios a comprender mejor el funcionamiento de los algoritmos y los efectos de su aplicación en el grafo.

5. Generación del Ejecutable:
    - El programa será convertido en un archivo ejecutable (.exe) para facilitar su distribución y ejecución en sistemas Windows.
    - Se asegurará de incluir todas las dependencias necesarias para que el programa se ejecute correctamente en sistemas Windows sin requerir instalaciones adicionales.

LÓGICA DEL PROGRAMA

Captura de las librerías usadas:

```
1   import networkx as nx
2   import tkinter as tk
3   from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
4   from matplotlib.figure import Figure
5
```

- NetworkX: NetworkX es una biblioteca de Python para el estudio de grafos y análisis de redes. NetworkX es un software libre publicado bajo la licencia BSD-new
- Matplotlib: Matplotlib es una biblioteca para la generación de gráficos en dos dimensiones, a partir de datos contenidos en listas o arrays en el lenguaje de programación Python

➢ **Creación de etiquetas y entradas de texto:**

```
origin_label = tk.Label(root, text="Vertice inicial para la búsqueda:") # Crear una etiqueta para el campo de entrada de vértices
origin_label.pack() # Empaquetar la etiqueta en la ventana

vertex_entry = tk.Entry(root) # Crear un campo de entrada para el vértice
vertex_entry.pack() # Empaquetar el campo de entrada en la ventana

edge1_label = tk.Label(root, text="vertice origen:") # Crear una etiqueta para el campo de entrada de aristas
edge1_label.pack() # Empaquetar la etiqueta en la ventana

edge_entry_1 = tk.Entry(root) # Crear un campo de entrada para las aristas
edge_entry_1.pack() # Empaquetar el campo de entrada en la ventana

edge2_label = tk.Label(root, text="Vertice destino:") # Crear una etiqueta para el campo de entrada de aristas
edge2_label.pack() # Empaquetar la etiqueta en la ventana

edge_entry_2 = tk.Entry(root) # Crear un campo de entrada para las aristas
edge_entry_2.pack() # Empaquetar el campo de entrada en la ventana
```

Este fragmento de código está utilizando la biblioteca Tkinter de Python para crear una interfaz gráfica de usuario (GUI) con campos de entrada para que el usuario ingrese datos relacionados con grafos

➢ Funcionalidades de Interacción y Visualización

```
def add_edge(): # Crear una función para agregar aristas
    G.add_edge(edge_entry_1.get(), edge_entry_2.get()) # Agregar una arista al grafo
    draw_graph() # Dibujar el grafo

add_edge_button = tk.Button(root, text="Agregar arista", command=add_edge) # Crear un botón para agregar aristas
add_edge_button.pack() # Empaquetar el botón en la ventana

print_info_button = tk.Button(root, text="Info. de datos agregados", command=lambda:print("Numero de vertices:",G.number_of_nodes(),"\nN
print_info_button.pack() # Empaquetar el botón en la ventana

figure = Figure(figsize=(5,5)) # Crear una figura
ax = figure.add_subplot(111) # Agregar un subplot a la figura
canvas = FigureCanvasTkAgg(figure, root) # Crear un lienzo para mostrar la figura en la ventana
canvas.get_tk_widget().pack() # Empaquetar el lienzo en la ventana
```

Este fragmento de código agrega funcionalidades de interacción al programa al permitir al usuario agregar aristas al grafo y ver información sobre los datos agregados. Además, proporciona una interfaz gráfica para visualizar el grafo utilizando una figura y un lienzo.

➢ Funciones de Dibujo y Recorrido

```
def draw_graph(traversal_nodes=None, traversal_edges=None): # Crear una función para dibujar el grafo
    ax.clear() # Limpiar el subplot
    if traversal_edges: # Si hay aristas de recorrido
        pos = nx.spring_layout(G) # Asignar posiciones a los nodos
        nx.draw(G, pos=pos, ax=ax, with_labels=True) # Dibujar el grafo
        nx.draw_networkx_edges(G, pos=pos, edgelist=traversal_edges, edge_color='r', ax=ax) # Dibujar las aristas de recorrido
        nx.draw_networkx_nodes(G, pos=pos, nodelist=traversal_nodes, node_color='r', ax=ax) # Dibujar los nodos de recorrido
    else:
        nx.draw(G, ax=ax, with_labels=True) # Dibujar el grafo
    canvas.draw() # Dibujar el lienzo que es la figura en la ventana

def show_traversal(traversal_func): # Crear una función para mostrar el recorrido
    source_node = vertex_entry.get() # Obtener el vértice inicial
    traversal_nodes = [] # Crear una lista para almacenar los nodos visitados
    traversal_edges = [] # Crear una lista para almacenar las aristas visitadas
    if traversal_func== nx.bfs_edges: # Si la función de recorrido es la de búsqueda en anchura
        traversal_edges = list(traversal_func(G, source=source_node)) # Realizar el recorrido en anchura
        traversal_nodes = [source_node] + [v for u, v in traversal_edges] # Almacenar los nodos visitados
    elif traversal_func == nx.dfs_edges: # Si la función de recorrido es la de búsqueda en profundidad
        traversal_edges = list(traversal_func(G, source=source_node)) # Realizar el recorrido en profundidad
        traversal_nodes = [source_node] + [v for u, v in traversal_edges] # Almacenar los nodos visitados
    print("Orden de visita durante la búsqueda:", traversal_nodes) # Imprimir el orden de visita
    draw_graph(traversal_nodes, traversal_edges) # Dibujar el grafo con los nodos y aristas visitados
    canvas.draw() # Dibujar el lienzo que es la figura en la ventana
```

Este fragmento de código permite al usuario interactuar con un grafo mediante la adición de aristas, la visualización de información sobre los datos agregados y la representación gráfica del grafo con la capacidad de resaltar aristas y nodos visitados durante un recorrido específico. Esto proporciona una herramienta interactiva y visualmente intuitiva para explorar y comprender la teoría de grafos.

➢ Funciones de Dibujo y Recorrido

```
bfs_button = tk.Button(root, text="Busqueda en Anchura", command=lambda: show_traversal(nx.bfs_edges)) # Crear un botón para la búsqueda
bfs_button.pack() # Empaquetar el botón en la ventana

dfs_button = tk.Button(root, text="Busqueda en Profundidad", command=lambda: show_traversal(nx.dfs_edges)) # Crear un botón para la búsqu
dfs_button.pack() # Empaquetar el botón en la ventana
def Limpiar_todo():     # Crear una función para limpiar todo
    G.clear()   # Limpiar el grafo
    vertex_entry.delete(0)   # Limpiar el campo de entrada de vértices
    edge_entry_1.delete(0)  # Limpiar el campo de entrada de aristas
    edge_entry_2.delete(0)  # Limpiar el campo de entrada de aristas
    draw_graph() # Dibujar el grafo

borrar_button = tk.Button(root, text= "Limpiar todo", command= Limpiar_todo) # Crear un botón para limpiar todo
borrar_button.pack() # Empaquetar el botón en la ventana

root.mainloop() # Mostrar la ventana
```

Este fragmento de código amplía las funcionalidades de la interfaz gráfica al permitir al usuario realizar búsquedas en anchura y en profundidad en el grafo, así como limpiar todos los elementos relacionados con el grafo y la interfaz gráfica para comenzar de nuevo.