

MANUAL TECNICO

librerías necesarias para el funcionamiento del proyecto.

1. **networkx (nx):** Utilizado para la creación y manipulación de estructuras de grafos complejas. Permite realizar operaciones comunes de grafos como añadir nodos, aristas, y realizar algoritmos de grafos.
2. **customtkinter (ctk):** Biblioteca que extiende las capacidades de Tkinter con un estilo visual mejorado y widgets adicionales.
3. **matplotlib:** Usada para visualizar datos en forma gráfica dentro de Python. En este contexto, se utiliza para dibujar los grafos y visualizar las estructuras de datos.

```
# Importa las bibliotecas necesarias para trabajar con grafos, interfaces de usuario personalizadas y gráficos.
import networkx as nx
import customtkinter as ctk
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from matplotlib.figure import Figure
import matplotlib.pyplot as plt
```

Creación del grafo

G: Es un objeto de grafo de NetworkX que se inicia vacío y puede ser modificado dinámicamente añadiendo nodos y aristas a través de la interfaz gráfica.

```
# Crea un grafo vacío usando NetworkX.
G = nx.Graph()
```

Configuración de la ventana principal

root: Es la ventana principal de la aplicación creada con `ctk.CTk()`, donde se configura el título y el comportamiento de ajuste automático de tamaño.

main_frame: Marco principal dentro de root que organiza visualmente el espacio de la aplicación y contiene otros widgets.

```
# Crea la ventana principal de la aplicación usando CustomTkinter.
root = ctk.CTk()
root.title("Algoritmo de búsqueda en Anchura") # Establece el título de la ventana.
```

Creación de widgets

Etiquetas y entradas para vértices y aristas: Permiten al usuario introducir y enviar datos al grafo.

Botones: Incluyen funcionalidades para agregar vértices y aristas, y botones para operaciones como dibujar el grafo y ejecutar algoritmos de búsqueda.

print_info_button: Botón que imprime información básica del grafo en la consola, como el número de nodos y aristas.

```

# Crea y posiciona los widgets para los nodos y aristas.
vertex_label = ctk.CTkLabel(main_frame, text="Vertice:")
vertex_label.grid(row=0, column=0, columnspan=2, pady=(0, 5), sticky="ew")

vertex_entry = ctk.CTkEntry(main_frame)
vertex_entry.grid(row=1, column=0, columnspan=2, pady=(0, 10), sticky="ew")

add_vertex_button = ctk.CTkButton(main_frame, text="Agregar vertice", command=lambda: G.add_node(vertex_entry.get()))
add_vertex_button.grid(row=2, column=0, columnspan=2, sticky="ew")

edge_label_1 = ctk.CTkLabel(main_frame, text="Arista inicio:")
edge_label_1.grid(row=3, column=0, pady=(0, 0), sticky="ew")

edge_entry_1 = ctk.CTkEntry(main_frame)
edge_entry_1.grid(row=4, column=0, pady=(0, 0), sticky="ew")

edge_label_2 = ctk.CTkLabel(main_frame, text="Arista fin:")
edge_label_2.grid(row=5, column=1, pady=(0, 0), sticky="ew")

edge_entry_2 = ctk.CTkEntry(main_frame)
edge_entry_2.grid(row=6, column=1, pady=(0, 0), sticky="ew")

add_edge_button = ctk.CTkButton(main_frame, text="Agregar arista", command=lambda: G.add_edge(edge_entry_1.get(), edge_entry_2.get()))
add_edge_button.grid(row=7, column=0, columnspan=2, sticky="ew")

print_info_button = ctk.CTkButton(main_frame, text="Info de datos agregados (consola)", command=lambda: print("Numero de nodos:", G.number_of_nodes(), "Numero de aristas:", G.number_of_edges()))
print_info_button.grid(row=8, column=0, columnspan=2, pady=(0, 5), sticky="ew")

```

Funciones para manipulación y visualización del grafo

draw_graph(G, ax, highlight=None): Esta función dibuja el grafo en un subplot de matplotlib. Utiliza la posición calculada por `nx.spring_layout` para el acomodo visual de los nodos. Los nodos o aristas pasados al parámetro `highlight` se resaltan en color rojo.

show_bfs() y **show_dfs():** Estas funciones ejecutan los algoritmos de Búsqueda en Anchura y Profundidad desde un nodo fuente. Después de ejecutar el algoritmo correspondiente, se crea y se muestra un grafo que representa el árbol de búsqueda con las aristas del árbol resaltadas. Los grafos resultantes se visualizan en paneles adyacentes para comparar con el grafo original.

```

# Define las funciones para dibujar el grafo y realizar la búsqueda en anchura.
def draw_graph(G, ax, highlight=None):
    ax.clear() # Limpia el subplot antes de dibujar
    pos = nx.spring_layout(G) # calcula la posición de los nodos
    nx.draw(G, pos, ax=ax, with_labels=True, node_color='skyblue')
    if highlight: # Si se proporciona una lista de nodos/aristas para resaltar
        # Asegurate de que highlight contenga aristas y no nodos
        highlight_nodes = [edge[0] for edge in highlight] + [edge[1] for edge in highlight]
        highlight_edges = highlight
        nx.draw_networkx_nodes(G, pos, nodelist=highlight_nodes, node_color='red', ax=ax)
        nx.draw_networkx_edges(G, pos, edgelist=highlight_edges, edge_color='red', ax=ax)

def show_bfs():
    fig, ax = plt.subplots(1, 2, figsize=(10, 5))
    source_node = vertex_entry.get().strip() # Asegurate de eliminar espacios en blanco
    print(f"Intentando buscar desde el nodo fuente: '{source_node}'") # Diagnóstico
    if source_node not in G:
        print("El nodo fuente no existe en el grafo.")
        return

    bfs_edges = list(nx.bfs_edges(G, source=source_node))
    bfs_tree = nx.Graph()
    bfs_tree.add_edges_from(bfs_edges) # Crea un nuevo grafo solo con las aristas de BFS

    draw_graph(G, ax[0]) # Dibuja el grafo original en el primer subplot
    draw_graph(bfs_tree, ax[1], highlight=bfs_edges) # Dibuja el grafo de BFS en el segundo subplot, resaltando las aristas de BFS

# Muestra la figura en el canvas de Tkinter
canvas = FigureCanvasTkAgg(fig, master=main_frame) # Asume que main_frame es tu marco principal
canvas_widget = canvas.get_tk_widget()
canvas_widget.grid(row=1, column=2, rowspan=6, padx=(10, 0), sticky="nsew")
canvas.draw()

```

Configuración de Matplotlib

Se prepara un `FigureCanvasTkAgg` para mostrar gráficos de matplotlib dentro de la interfaz de Tkinter.

Ejecución de la aplicación

Se configura el mainloop de Tkinter, que es esencial para que la interfaz de usuario permanezca activa y responda a eventos del usuario.

```

# Ahora que show bfs está definida, crea los botones que la utilizan.
draw_button = ctk.CTkButton(main_frame, text="Dibujar grafo", command=show_bfs)
draw_button.grid(row=7, column=0, padx=(0,5), sticky="ew")

bfs_button = ctk.CTkButton(main_frame, text="Búsqueda en anchura", command=lambda: show_bfs())
bfs_button.grid(row=7, column=1, padx=(5,0), sticky="ew")

dfs_button = ctk.CTkButton(main_frame, text="Búsqueda en profundidad", command=show_dfs)
dfs_button.grid(row=8, column=0, colspan=2, sticky="ew", pady=(5,0))

# Configura Matplotlib para la visualización del grafo.
figure = Figure(figsize=(5, 5))
ax = figure.add_subplot(111)
canvas = FigureCanvasTkAgg(figure, main_frame)
canvas_widget = canvas.get_tk_widget()
canvas_widget.grid(row=1, column=2, rowspan=6, padx=(10, 0), sticky="nsew")

# Configura la ventana principal para ajustarse automáticamente al tamaño de sus contenidos.
root.grid_rowconfigure(1, weight=1)
root.grid_columnconfigure(2, weight=1)

# Inicia el bucle principal de la aplicación Tkinter.
root.mainloop()

```

DIAGRAMA DE FLUJO

