**UNIVERSIDAD DE PANANÁ**

**CENTRO REGIONAL UNIVERSITARIO DE COCLÉ**

**FACULTAD DE INFORMÁTICA, ELECTRÓNICA Y COMUNICACIÓN.**

**LICENCIATURA EN INGENIERIA INFORMÁTICA**

**ASIGNATURA: INFORMATICA TEÓRICA**

**PROYECTO FINAL**

**IMPLEMENTANDO UN SISTEMA EN PYTHON CON EL ALGORITMO DIJKDTRA**

**PROFESOR:**

**LUIS DOMINGUEZ**

**ESTUDIANTES:**

**JAHIR RODRÍGUEZ 2-749-1152**

**FERNANDA NARIÑO DOUBLAS 2-753-897**

**FECHA:**

7/**12/2023**

**¿Por qué se usaron estas librerías?**

import tkinter as tk: Es la librería estándar de Python para crear interfaces gráficas de usuario (GUI). Proporciona herramientas para crear ventanas, botones, etiquetas, etc. El alias tk es para referirse a ella.

from tkinter import ttk: Proviene del módulo de temas de tkinter. Ofrece widgets más avanzados y con un aspecto más moderno que los proporcionados por tkinter.

import networkx as nx: es una librería para la creación, manipulación y estudio de estructuras, dinámicas y funciones de redes complejas. Es útil para analizar y visualizar redes.

import matplotlib.pyplot as plt: es una librería para crear gráficos estáticos, como gráficos de líneas, barras, dispersión, histogramas, etc.

from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg: es una clase que permite integrar gráficos creados con matplotlib en aplicaciones de tkinter. Facilita mostrar gráficos matplotlib dentro de ventanas de tkinter.

**Código de ejecución**

#Declaracion de la librerias a utilizar

import tkinter as tk

from tkinter import ttk

import networkx as nx

import matplotlib.pyplot as plt

from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg

#se define la clase Graphapp

class GraphApp:

def \_\_init\_\_(G, master):

G.master = master

G.master.title("Dijkstra Shortest Path Visualization")

# sentencia que Crea un grafo de ejemplo

G.graph = nx.Graph()

node\_names = ["Anton", "Cabuya", "El chiru", "El retiro", "El valle", "Juan diaz", "Rio hato", "San juan de dios", "Santa rita", "Caballero",

"Penonome", "Cañaveral","Cocle", "Chiguiri arriba", "El coco", "Pajonal", "Rio grande", "Rio indio", "Toabre", "Tulu","Aguadulce",

"El cristo", "El roble", "pocri", "Barrios unidos", "Pueblos unidos", "Virgen del carmen", "El hato",

"La pintada", "El harino", "El Potrero", "Llano Grande", "Piedras Gordas", "Las Lomas", "Llano Norte",

"Ola", "El Cope", "El Palmar", "El Picacho", "La Pava","Nata de los caballeros", "Capellania", "El Caño", "Guzman", "Las Huacas","Toza","Villarreal"]

G.graph.add\_nodes\_from(node\_names)

#sentencia que permite agregar el peso de las aristas a los nodos

G.graph.add\_edges\_from([("Anton", "Penonome", {"weight": 15.6}),

("Cabuya", "Caballero", {"weight": 3.3}),

("El chiru", "Anton", {"weight": 6.4}),

("El retiro", "Santa rita", {"weight": 3.9}),

("El valle", "Caballero", {"weight": 10.1}),

("Juan diaz", "Penonome", {"weight": 2.0}),

("Rio hato", "El chiru", {"weight": 12.9}),

("Rio hato", "El valle", {"weight": 12.9}),

("San juan de dios", "Juan diaz", {"weight": 3.9}),

("Santa rita", "Juan diaz", {"weight": 11.6}),

("Caballero", "Juan diaz", {"weight": 13.4}),

####Corregimientos de Penonome#########

("Penonome", "Rio grande", {"weight":15.0 }),

("Cañaveral", "Penonome", {"weight": 8.7}),

("Cocle", "Rio grande", {"weight": 5}),

("Cocle", "Penonome", {"weight": 13.4}),

("Chiguiri arriba", "Pajonal", {"weight":11.4 }),

("El coco", "Penonome", {"weight":14.3 }),

("Pajonal", "Penonome", {"weight": 13.9}),

("Rio grande", "Aguadulce", {"weight":22.3 }),

("Rio indio", "Chiguiri arriba", {"weight": 13.0}),

("Toabre", "Pajonal", {"weight":9.4 }),

("Tulu", "Toabre", {"weight":14.9}),

#####Corregimientos de Aguadulce#########

("Aguadulce", "pocri", {"weight": 1.9}),

("El cristo", "pocri", {"weight": 10.5}),

("Barrios unidos", "Aguadulce", {"weight": 6.4}),

("Pueblos unidos", "El roble", {"weight": 5.0}),

("Virgen del carmen", "pocri", {"weight": 1.2}),

("El hato", "El cristo", {"weight": 2.3}),

("Aguadulce", "El roble", {"weight": 14.6}),

#####Corregimientos de La Pintada#########

("La pintada", "Cañaveral", {"weight":8.4 }),

("El harino", "Piedras Gordas", {"weight": 4.4}),

("El Potrero", "La pintada", {"weight": 11.5}),

("Llano Grande", "La pintada", {"weight":4.4 }),

("Piedras Gordas", "La pintada", {"weight":8.9}),

("Piedras Gordas", "Las Lomas", {"weight":11.8}),

("Llano Norte", "Llano Grande", {"weight": 16.4}),

#####Corregimientos de Olá#########

("Ola", "Rio grande", {"weight":19.0 }),

("El Cope", "Ola", {"weight": 3.6 }),

("El Palmar", "El Cope", {"weight":15.4 }),

("El Picacho", "Ola", {"weight":2.6}),

("La Pava", "Ola", {"weight":4.7}),

#####Corregimientos de Nata#########

("Nata de los caballeros", "Rio grande", {"weight":12.8}),

("Nata de los caballeros", "Aguadulce", {"weight":9.3}),

("Capellania", "Nata de los caballeros", {"weight":5.7}),

("El Caño", "Rio grande", {"weight": 6.7}),

("Guzman", "El Caño", {"weight": 9.0}),

("Las Huacas", "Ola", {"weight": 12.5}),

("Las Huacas", "Toza", {"weight": 17.9}),

("Toza","Capellania", {"weight": 10.7}),

("Villarreal", "Capellania", {"weight": 10.3}),

("Villarreal", "Toza", {"weight": 2.9})]),

G.positions = nx.spring\_layout(G.graph)

G.path\_edges = []

# permite crear Variables de control para los menús desplegables

G.start\_node\_var = tk.StringVar()

G.end\_node\_var = tk.StringVar()

# permite inicializar la interfaz gráfica

G.init\_gui()

def init\_gui(G):

# Crear ventana para el grafo

graph\_window = tk.Toplevel(G.master)

graph\_window.title("Graph Window")

# Crear un lienzo para el grafo

G.fig, G.ax = plt.subplots()

G.ax.set\_facecolor('Wheat') # permite dar color al lienzo

G.canvas = FigureCanvasTkAgg(G.fig, master=graph\_window)

G.canvas\_widget = G.canvas.get\_tk\_widget()

G.canvas\_widget.pack(side=tk.TOP, fill=tk.BOTH, expand=1)

# Permite crear la ventana principal para la interfaz gráfica

main\_window = G.master

# Permite cambiar el color de fondo a menta

G.master.configure(bg="Darkgray")

# Crear menús desplegables para seleccionar nodos de inicio y fin

start\_label = ttk.Label(main\_window, text="¿Donde te encuentras?")

start\_label.pack(pady=5)

start\_menu = ttk.Combobox(main\_window, textvariable=G.start\_node\_var,

values=list(G.graph.nodes))

start\_menu.pack(pady=5)

end\_label = ttk.Label(main\_window, text="¿A Donde Quieres ir?")

end\_label.pack(pady=5)

end\_menu = ttk.Combobox(main\_window, textvariable=G.end\_node\_var,

values=list(G.graph.nodes))

end\_menu.pack(pady=5)

# Botón para encontrar la ruta más corta

find\_button = ttk.Button(main\_window, text="Encontrar Ruta",

command=G.find\_shortest\_path)

find\_button.pack(pady=10)

def find\_shortest\_path(G):

start\_node = G.start\_node\_var.get()

end\_node = G.end\_node\_var.get()

# Ejecucion del algoritmo de Dijkstra

shortest\_path = nx.shortest\_path(G.graph, source=start\_node, target=end\_node, weight="weight")

G.path\_edges = list(zip(shortest\_path, shortest\_path[1:]))

# Dibujar el grafo

G.draw\_graph()

def draw\_graph(G):

# Limpiar la visualización anterior

G.ax.clear()

# Dibuja los nodos y aristas

nx.draw\_networkx\_nodes(G.graph, G.positions, node\_size=150, node\_color="cyan", ax=G.ax)

nx.draw\_networkx\_edges(G.graph, G.positions, ax=G.ax)

nx.draw\_networkx\_labels(G.graph, G.positions, font\_size=9, ax=G.ax)

# permite dibujar nodos de inicio y destino en colores específicos

start\_node = G.start\_node\_var.get()

end\_node = G.end\_node\_var.get()

nx.draw\_networkx\_nodes(G.graph, G.positions, nodelist=[start\_node], node\_color="red", ax=G.ax)

nx.draw\_networkx\_nodes(G.graph, G.positions, nodelist=[end\_node], node\_color="green", ax=G.ax)

# Resalta la ruta más corta en rojo

nx.draw\_networkx\_edges(G.graph, G.positions, edgelist=G.path\_edges, edge\_color="red", width=2, ax=G.ax)

# Mostrar los pesos de las aristas

edge\_labels = nx.get\_edge\_attributes(G.graph, 'weight')

nx.draw\_networkx\_edge\_labels(G.graph, G.positions, edge\_labels=edge\_labels, ax=G.ax)

# Calcular y muestra la distancia total recorrida

total\_distance = sum(G.graph.edges[edge]["weight"] for edge in G.path\_edges)

distance\_text = f"Distancia total: {total\_distance} km"

G.ax.text(0.5, -0.1, distance\_text, ha="center", va="center", transform=G.ax.transAxes)

# Muestra los nodos visitados debajo de la distancia total

shortest\_path\_nodes = nx.shortest\_path(G.graph, source=start\_node, target=end\_node)

visited\_nodes\_text = f"Nodos visitados: {' -> '.join(shortest\_path\_nodes)}"

G.ax.text(0.5, -0.07, visited\_nodes\_text, ha="center", va="center", transform=G.ax.transAxes)

# Actualiza la interfaz gráfica

G.canvas.draw()

if \_\_name\_\_ == "\_\_main\_\_":

root = tk.Tk()

app = GraphApp(root)

root.mainloop()

