**Universidad de Panamá.**

**Centro Regional Universitario de Coclé**

**Facultad de Informática, Electrónica y Comunicación.**

**Lic. En Ingeniería en informática.**

**Grupo: G02-2**

**Asignatura:**

Informática Teórica

**Tema:**

Proyecto Final- Semestral

**Estudiantes:**

Francisco Domínguez 2-752-1297

Santiago Martínez 2-751-571

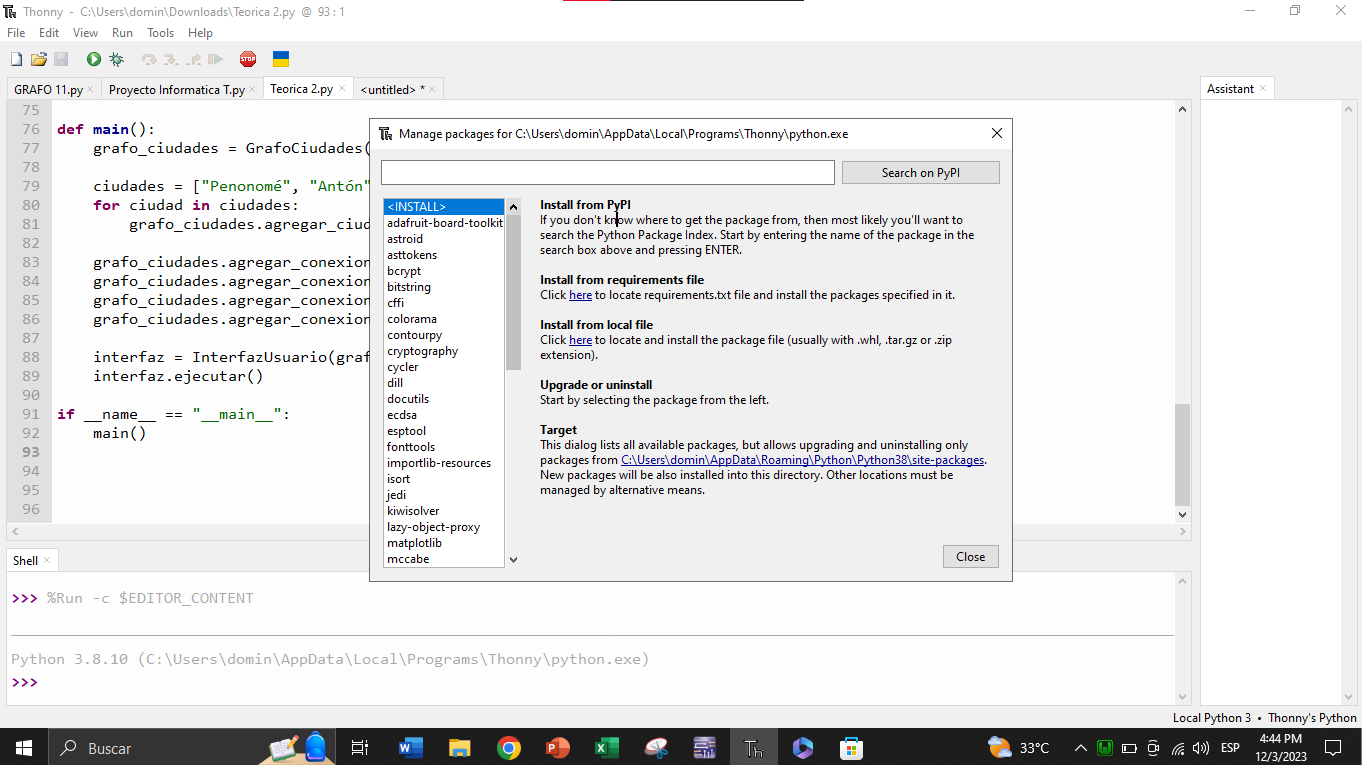
**Profesora:**

Luis Domínguez

**1. Instalación de Paquetes:**

En el menú principal, selecciona "Herramientas" y luego "Gestor de paquetes."

En el gestor de paquetes, ingresa networkx, matplotlib, y tk y haz clic en "Instalar."



**2. Ejecución del Código:**

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamenteAbre tu script en Thonny, Asegúrate de haber guardado el script. Haz clic en el botón "Run" (icono de un triángulo verde) en la parte superior de la ventana de Thonny.

**3. Instrucciones Adicionales:**

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamenteAsegúrate de tener una versión de Python 3.x seleccionada en Thonny. Puedes verificar y seleccionar la versión de Python en el menú desplegable en la esquina inferior derecha de la ventana de Thonny.

**Datos de Ejemplo:**

El código utiliza datos de ejemplo para ciudades ("Penonomé", "Antón", "Aguadulce", "Natá") y distancias ficticias entre ellas. Puedes modificar estos datos según tus necesidades.

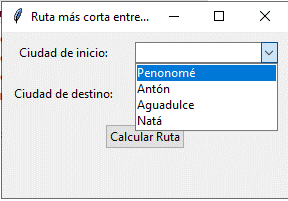
**Entendiendo el Código:**

Se proporciona una clase GrafoCiudades que utiliza NetworkX para manejar el grafo.

La clase InterfazUsuario crea una interfaz gráfica con Tkinter para interactuar con el grafo.

**Ejecución del código**

Al ejecutar lo primero que saldrá es una ventana donde el usuario seleccionara la ciudad de inicio y la ciudad de destino.

****

Luego de haber colocado las ciudades, presionar el botón calcular ruta y aparecerá una imagen con el recorrido mas corto entre las ciudades indicándolo con un texto en la parte superior.

**Imagen que contiene pesca, esquiando, barco, agua

Descripción generada automáticamente**

**CODIGO**

# Importación de bibliotecas

import networkx as nx # networkx se utiliza para trabajar con grafos

import matplotlib.pyplot as plt # matplotlib se usa para visualizar gráficos

import tkinter as tk # tkinter es una biblioteca para crear interfaces gráficas de usuario

from tkinter import ttk # ttk es un módulo adicional de tkinter que proporciona widgets temáticos mejorados

# Definición de la clase para el grafo de ciudades

class GrafoCiudades:

def \_\_init\_\_(self):

# Constructor: Inicializa un grafo vacío utilizando networkx

self.grafo = nx.Graph()

def agregar\_ciudad(self, ciudad):

# Añade un nodo (ciudad) al grafo

self.grafo.add\_node(ciudad)

def agregar\_conexion(self, ciudad\_origen, ciudad\_destino, distancia):

# Añade una conexión entre dos ciudades con una distancia específica al grafo

self.grafo.add\_edge(ciudad\_origen, ciudad\_destino, weight=distancia)

def visualizar\_grafo(self, nombre\_archivo="Graph.png", mensaje="", ruta\_resaltada=None):

# Visualiza el grafo utilizando matplotlib y guarda la representación en un archivo PNG

pos = nx.spring\_layout(self.grafo)

nx.draw(self.grafo, pos, with\_labels=True, node\_size=800, node\_color='skyblue', font\_size=10, font\_weight='bold', edge\_color='gray', width=2, font\_color='black')

if ruta\_resaltada:

# Si se proporciona una ruta resaltada, la resalta en rojo

edges = list(zip(ruta\_resaltada, ruta\_resaltada[1:]))

nx.draw(self.grafo, pos, edgelist=edges, edge\_color='red', width=3)

# Etiquetas de las conexiones con las distancias redondeadas a 2 decimales

labels = nx.get\_edge\_attributes(self.grafo, 'weight')

rounded\_labels = {k: round(v, 2) for k, v in labels.items()}

nx.draw\_networkx\_edge\_labels(self.grafo, pos, edge\_labels=rounded\_labels)

# Anotación del mensaje en el gráfico

plt.annotate(mensaje, xy=(0.5, 0.95), xycoords='axes fraction', ha='center', va='center', bbox=dict(boxstyle="round,pad=0.1", edgecolor="black", facecolor="white"))

# Guarda el gráfico como un archivo PNG y lo muestra

plt.savefig(nombre\_archivo, format="PNG")

plt.show()

def encontrar\_ruta\_mas\_corta(self, inicio, destino):

try:

# Utiliza el algoritmo de Dijkstra para encontrar la ruta más corta y su distancia

distancia\_corta = nx.single\_source\_dijkstra\_path\_length(self.grafo, inicio, weight='weight')

ruta = nx.shortest\_path(self.grafo, inicio, destino, weight='weight')

return distancia\_corta[destino], ruta

except nx.NetworkXNoPath:

# Maneja la excepción cuando no hay ruta entre las ciudades

return float('inf'), []

# Definición de la clase para la interfaz de usuario

class InterfazUsuario:

def \_\_init\_\_(self, grafo\_ciudades):

# Constructor: Inicializa la interfaz gráfica de usuario (GUI) con tkinter

self.grafo\_ciudades = grafo\_ciudades

self.ventana = tk.Tk()

self.ventana.title("Ruta más corta entre ciudades")

# Configuración y creación de widgets para la selección de ciudades de inicio

self.label\_inicio = ttk.Label(self.ventana, text="Ciudad de inicio:")

# Crea una etiqueta (label) con el texto "Ciudad de inicio"

self.label\_inicio.grid(row=0, column=0, padx=10, pady=10)

# Coloca la etiqueta en la fila 0, columna 0 de la interfaz, con márgenes de 10 píxeles en x y 10 píxeles en y

self.ciudades = grafo\_ciudades.grafo.nodes()

# Obtiene la lista de nodos (ciudades) del grafo

self.combo\_inicio = ttk.Combobox(self.ventana, values=list(self.ciudades))

# Crea un cuadro combinado (combobox) con las ciudades como opciones

self.combo\_inicio.grid(row=0, column=1, padx=10, pady=10)

# Coloca el cuadro combinado en la fila 0, columna 1 de la interfaz, con márgenes de 10 píxeles en x y 10 píxeles en y

# Configuración y creación de widgets para la selección de ciudades de destino

self.label\_destino = ttk.Label(self.ventana, text="Ciudad de destino:")

# Crea una etiqueta (label) con el texto "Ciudad de destino"

self.label\_destino.grid(row=1, column=0, padx=10, pady=10)

# Coloca la etiqueta en la fila 1, columna 0 de la interfaz, con márgenes de 10 píxeles en x y 10 píxeles en y

self.combo\_destino = ttk.Combobox(self.ventana, values=list(self.ciudades))

# Crea un cuadro combinado (combobox) con las ciudades como opciones

self.combo\_destino.grid(row=1, column=1, padx=10, pady=10)

# Coloca el cuadro combinado en la fila 1, columna 1 de la interfaz, con márgenes de 10 píxeles en x y 10 píxeles en y

# Botón para calcular la ruta más corta

self.boton\_calcular = ttk.Button(self.ventana, text="Calcular Ruta", command=self.calcular\_ruta)

self.boton\_calcular.grid(row=2, column=0, columnspan=2, pady=10)

# Coloca el botón en la fila 2, columnas 0 y 1 de la interfaz, abarcando ambas columnas, con un margen de 10 píxeles en y

# Etiqueta para mostrar el resultado

self.label\_resultado = ttk.Label(self.ventana, text="")

self.label\_resultado.grid(row=3, column=0, columnspan=2, pady=10)

# Coloca la etiqueta en la fila 3, columnas 0 y 1 de la interfaz, abarcando ambas columnas, con un margen de 10 píxeles en y

def calcular\_ruta(self):

# Obtiene las ciudades seleccionadas por el usuario

inicio = self.combo\_inicio.get()

destino = self.combo\_destino.get()

# Llama al método para encontrar la ruta más corta

distancia\_corta, ruta = self.grafo\_ciudades.encontrar\_ruta\_mas\_corta(inicio, destino)

if distancia\_corta != float('inf'):

# Mensaje con la distancia más corta y la ruta sugerida

mensaje = f"La distancia más corta entre {inicio} y {destino} es {round(distancia\_corta, 2)} km.\nRuta sugerida: {ruta}"

else:

# Mensaje cuando no hay ruta válida

mensaje = f"No hay ruta válida entre {inicio} y {destino}."

# Actualiza la etiqueta de resultado y visualiza el grafo con la ruta resaltada

self.label\_resultado.config(text=mensaje)

self.grafo\_ciudades.visualizar\_grafo(mensaje=mensaje, ruta\_resaltada=ruta)

def ejecutar(self):

# Inicia el bucle principal de la interfaz gráfica

self.ventana.mainloop()

# Función principal

def main():

# Creación de una instancia de GrafoCiudades

grafo\_ciudades = GrafoCiudades()

# Agregando ciudades y conexiones al grafo

ciudades = ["Penonomé", "Antón", "Aguadulce", "Natá", "La Pintada", "Olá", "Los Uveros", "Universidad", "El Ingenio", "Rio Grande", "La Soledad"]

for ciudad in ciudades:

grafo\_ciudades.agregar\_ciudad(ciudad)

grafo\_ciudades.agregar\_conexion("Penonomé", "Los Uveros", 7.5)

grafo\_ciudades.agregar\_conexion("Los Uveros", "La Pintada", 11.3)

grafo\_ciudades.agregar\_conexion("Penonomé", "Universidad", 4.5)

grafo\_ciudades.agregar\_conexion("Universidad", "Antón", 13)

grafo\_ciudades.agregar\_conexion("Penonomé", "Rio Grande", 27.3)

grafo\_ciudades.agregar\_conexion("Rio Grande", "Natá", 35.0)

grafo\_ciudades.agregar\_conexion("Olá", "La Soledad", 4.2)

grafo\_ciudades.agregar\_conexion("La Soledad", "Aguadulce", 32)

grafo\_ciudades.agregar\_conexion("Aguadulce", "El Ingenio", 9.3)

grafo\_ciudades.agregar\_conexion("El Ingenio", "Natá", 6.6)

# Creación de una instancia de InterfazUsuario y ejecución

interfaz = InterfazUsuario(grafo\_ciudades)

interfaz.ejecutar()

# Verifica si el script está siendo ejecutado directamente

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

main()