**Background pattern

Description automatically generated with medium confidenceManual técnico**

**APLICACIÓN INTERACTIVA PARA LA CREACION, ANALISIS Y ESTUDIO DE GRAFOS**

**Logo

Description automatically generated with medium confidence**

**Background pattern

Description automatically generated with medium confidence**

**Manual técnico**

**A screenshot of a computer

Description automatically generated with low confidence**

**Logo

Description automatically generated with medium confidence LUZ ENITH GUERRERO**

**UNIVERSIDAD NACIONAL DE COLOMBIA**

**MANIZALES-COLOMBIA**

**ANALISIS Y DISEÑOS DE ALGORITMOS**

**2022-01**

Tabla de Contenido

[Introducción 5](#_Toc107882034)

[Objetivo 6](#_Toc107882035)

[Caras del software 7](#_Toc107882036)

[Lenguaje de desarrollo 9](#_Toc107882037)

[Características del software 10](#_Toc107882038)

[Ejecución 11](#_Toc107882039)

[Código 12](#_Toc107882040)

[Main.py 12](#_Toc107882041)

[Window\_main.py 14](#_Toc107882042)

[Class VentanaPrincipal(tk.Tk): 14](#_Toc107882043)

[def \_\_init\_\_(self, \*args, \*\*kwargs): 14](#_Toc107882044)

[def init\_menubar(self): 16](#_Toc107882045)

[def init\_buttons(self): 18](#_Toc107882046)

[def func\_actualizar\_figure(self, \*args): 20](#_Toc107882047)

[def func\_eliminar\_nodo(self, \*args): 20](#_Toc107882048)

[def func\_salta\_o\_d(self, \*args): 20](#_Toc107882049)

[def func\_salta\_d\_p(self, \*args): 20](#_Toc107882050)

[def func\_prueba(self, \*args): 21](#_Toc107882051)

[def func\_agregar\_nodo(self, \*args): 21](#_Toc107882052)

[def func\_agregar\_arista(self, \*args): 21](#_Toc107882053)

[def archivo\_nuevo\_presionado(self, \*args): 22](#_Toc107882054)

[def exportar\_CSV(self, \*args): 22](#_Toc107882055)

[def importar\_CSV(self, \*args): 23](#_Toc107882056)

[def exportar\_PDF(self, \*args): 23](#_Toc107882057)

[def exportar\_imagen(self, \*args): 23](#_Toc107882058)

[def guardar\_archivo(self, \*args): 24](#_Toc107882059)

[def abrirarchivo(self, \*args): 24](#_Toc107882060)

[def menuarchivo(self, menu, bar\_menu): 24](#_Toc107882061)

[def menuanalizar(self, menu, bar\_menu): 25](#_Toc107882062)

[def menuherramienta(self, menu, bar\_menu): 26](#_Toc107882063)

[def menuaplicacion(self, menu, bar\_menu): 26](#_Toc107882064)

[def menuventana(self, menu, bar\_menu): 27](#_Toc107882065)

[def menuayuda(self, menu, bar\_menu): 27](#_Toc107882066)

[Class PartitionKL(tk.Toplevel): 27](#_Toc107882067)

[def \_\_init\_\_(self, Graph, \*args, \*\*kwargs): 28](#_Toc107882068)

[def destroy(self): 28](#_Toc107882069)

[def abrir\_ventana\_partition(self, Graph): 28](#_Toc107882070)

[Class PartitionGN(tk.Toplevel): 29](#_Toc107882071)

[def \_\_init\_\_(self, Graph, \*args, \*\*kwargs): 29](#_Toc107882072)

[def destroy(self): 30](#_Toc107882073)

[def abrir\_ventana\_partition\_Girvan\_Newman(self, Graph): 30](#_Toc107882074)

[Class PartitionLouvain(tk.Toplevel): 31](#_Toc107882075)

[def \_\_init\_\_(self, Graph, \*args, \*\*kwargs): 31](#_Toc107882076)

[def destroy(self): 32](#_Toc107882077)

[def abrir\_ventana\_partition\_Louvain(self, Graph): 32](#_Toc107882078)

[Window\_start.py 33](#_Toc107882079)

[Class TypeGraph(tk.simpledialog.Dialog): 33](#_Toc107882080)

[def \_\_init\_\_(self, parent, title): 33](#_Toc107882081)

[def body(self, frame): 33](#_Toc107882082)

[def ok\_pressed(self): 34](#_Toc107882083)

[def buttonbox(self): 34](#_Toc107882084)

[References 34](#_Toc107882085)

Introducción

El presente manual contiene la explicación del código para un correcto funcionamiento y enseña todas las funcionalidades que posee el software desarrollado para que pueda ser fácilmente implementado por otro usuario, siguiendo este manual no requiere conocimientos avanzados para replicar este proyecto.

Objetivo

El desarrollo que lleva el software busca crear un ambiente amigable al usuario el cual permita su fácil uso, el software se enfoca en el desarrollo de una aplicación interactiva para la creación, análisis y estudio de grafos. el cual cumple con ciertos criterios explicados más adelante y goza de las características de estar desarrollado en español, con las herramientas necesarias para un manejo fácil del grafo, los colores son adaptables al tema de su computadora y cuenta con un área donde el usuario puede ver en tiempo real el trabajo que realiza con el grafico.

Caras del software

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with low confidence

A picture containing text, sky, different

Description automatically generated

Los anteriores son imágenes que presentan las 3 ventanas programables del software, se relacionan las 3 de manera que la primera genera la segunda y en esta se inicia la tercera. Se toman como la ventana de selección, funcionamiento y resultados de partición.

Lenguaje de desarrollo

El lenguaje en que se desarrolló la aplicación interactiva para la creación, análisis y estudio de grafos, está basada en Python 3, provisto por PyCharm.

El editor se encuentra en la versión 2022.1.3.

Características del software

El presente proyecto pretende estudiar el manejo de algoritmos sobre grafos haciendo un análisis de algunos algoritmos en el ámbito de problemas de optimización.

El funcionamiento de cada algoritmo estudiado se representará en la máquina mediante un simulador para poder observar los resultados de ejecución de forma más exacta.

Se implementará, para el estudio de los problemas de procesamiento de grafos, un simulador didáctico con el cual el usuario podrá analizar los posibles resultados de diferentes algoritmos para resolver el problema de partición de grafos. Esto será de gran beneficio para quien haga uso del aplicativo creado con este proyecto, al poder ejecutar y comparar varios algoritmos, midiendo las respuestas y comportamientos, así como el tiempo de respuesta real y teórico, entre otros. El sistema debe ser desarrollado en el lenguaje de programación Python y deberá tener una interfaz gráfica de usuario en entorno web. Dicha interfaz gráfica deberá contar con todas las opciones para que el usuario cargue grafos desde archivos si así lo desea (en formato JSON) o simplemente crearlo manualmente. La información que se cargue al sistema (desde archivo o manualmente) y los resultados generados a raíz de la ejecución de un algoritmo en particular deberán ser almacenados en base de datos con el fin de mantener un histórico de las ejecuciones y poder comparar los resultados entre los diferentes algoritmos por ejemplo para un mismo grafo. Se recomienda utilizar la base de datos mongodb (mongodb Atlas) debido a su dinamismo por ser NoSql. La aplicación consistirá en un simulador didáctico para la aplicación de algoritmos de partición sobre grafos como se explica a continuación: El simulador permitirá diseñar un grafo agregando o eliminando nodos, permitiendo manejar parámetros de entrada como el tipo de grafo a generar, los algoritmos a utilizar, las consultas que se desplegarán al ejecutar el algoritmo, etc. Con esto se podrá analizar los resultados del algoritmo que se probará para el grafo teniendo en cuenta los parámetros y experimentando con posibles configuraciones, pesos, etc. También se tendrá la opción de dejar que la aplicación genere el grafo automáticamente con determinados valores (cantidad de nodos, cantidad de arcos, etc). El simulador también permitirá guardar y modificar los grafos, si así lo desea el usuario. El simulador mostrará de forma gráfica tanto el grafo generado como el análisis del algoritmo que usará el grafo.

El simulador permitirá exportar el grafo a Excel o como imagen. El menú principal de la aplicación se describe en el [Manual de Usuario-Aplicación Grafos](https://unaledu-my.sharepoint.com/:w:/g/personal/nasuarezro_unal_edu_co/ETpwXjkm37JEkstCYAC1eOIBkQhb9WeRse8PPxz6bnUGCw).

Ejecución

En la ejecución del programa inicialmente se determina el tipo de grafo a trabajar, y posterior a ello se encuentra la interfaz donde el usuario puede dibujar su grafo, mediante cuadros de entrada de texto para que el usuario ingrese los nodos y aristas, aquí mismo presenta una ventana que se actualiza en tiempo real para llevar un mejor control del grafo realizado, a esto se le suma un campo de entrada donde el usuario puede eliminar nodos en caso de requerirlo.

Con lo anterior descrito, la aplicación presenta una serie de herramientas con las cuales se puede realizar el análisis y estudio de los grafos, incluyendo la posibilidad de importar y exportar los proyectos realizados.

En cuanto a los temas de la aplicación durante la ejecución, se tiene que estos son adaptables con respecto a la configuración de nuestra computadora, de preferencia recomendamos Windows 11.

Código

Graphical user interface, text, application, chat or text message

Description automatically generated

El código se encuentra distribuido en 3 archivos (“*\*.py”*) dentro de el paquete (“*GraphUlinteractive*”) y imagen (“*\*.ico*”) dentro del mismo paquete, los cuales están interrelacionados por la Clase window\_main.py, alojada en el archivo de mismo nombre, la clase Main es usada solo como el ejecutable del resto del código, donde crea el objeto de tipo ventana principal.

Empezaremos no por el orden alfabético de los archivos sino por el Main, siguiendo por window\_main y por ultimo window\_start, para asi ver que hace cada uno y cómo actúan juntos.

Main.py

EL archivo “main.py” se encuentra compuesto por:

1. import darkdetect
2. from window\_main import \*
3. import sv\_ttk

El anterior correspondiente a las librerías importadas y al paquete del que hace parte (window\_main).

2. Ventana = VentanaPrincipal()
3. ntkutils.placeappincenter(Ventana)

En estas dos líneas de código, se instancia el objeto llamado Ventana y se imprime en el centro de la pantalla con ayuda de la librería tkinter.

1. if darkdetect.theme() == "Dark":
2. sv\_ttk.set\_theme("dark")
3. ntkutils.dark\_title\_bar(Ventana)
4. ntkutils.blur\_window\_background(Ventana)
5. else:
6. sv\_ttk.set\_theme("light")
7. ntkutils.blur\_window\_background(Ventana)

Con este condicional, se coloca el tema a nuestra ventana de la aplicación, si el pc tiene un tema oscuro, la ventana también lo tendrá, si este es clarito, por supuesto también lo tomará.

1. Ventana.focus()
2. Ventana.init\_menubar()
3. Ventana.init\_buttons()
4. Ventana.mainloop()

La línea 15 hace referencia al objeto Ventana para trabajar sobre el mismo y genera la ventana sobre la cual se va a trabajar.

La línea 16 instancia al método menubar() de la clase ventanaprincipal contenida en el archivo window\_main() y se encarga de imprimir la barra de herramientas (menú).

La línea 17 instancia al método buttons() de la clase ventanaprincipal contenida en el archivo window\_main() y se encarga de imprimir los botones de agregar nodo, agregar arista, eliminar nodo y cada una con su respectivo formulario, adicionalmente imprime el espacio donde va a imprimir el grafo en tiempo real.

Window\_main.py

Este archivo se encuentra conformado por lo siguiente:

1. import json
2. from tkinter.filedialog import askopenfile, asksaveasfile
3. import networkx as nx
4. import ntkutils
5. from matplotlib import pyplot as plt
6. from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg
7. from networkx import node\_link\_data
8. from networkx.readwrite import json\_graph
9. from window\_start import \*

Librerías necesarias para el funcionamiento de la ventana: Libreria JSON, Libreria para el manejo de archivos, Libreria para el manejo de grafos, Libreria para mostrar pyplot, Libreria para mostrar figuras, Libreria para exportar el grafo, Libreria para importar el grafo y la ulrima línea hace referencia a la Clase para mostrar la ventana de la selección de tipo de grafo.

## Class VentanaPrincipal(tk.Tk):

Esta clase es la estructura de la aplicación, en esta clase se muestra toda la información relacionada con el grafo a trabajar, contiene el siguiente codigo:

22. class VentanaPrincipal(tk.Tk):

Esta clase contiene los siguientes métodos

### def \_\_init\_\_(self, \*args, \*\*kwargs):

def \_\_init\_\_(self, \*args, \*\*kwargs): #

super().\_\_init\_\_(\*args, \*\*kwargs) #

ntkutils.placeappincenter(self) #   
 self.iconbitmap(r'icon.ico') #   
 init = TypeGraph(self, "Selección") #

print(init.var.get()) #

La función Queda abierta a n argumentos con identificador y Se almacena por herencia el \*args \*\*kwargs, luego Se coloca en el centro de la pantalla, en ese momento Se carga el icono que esta almacenado en el archivo icono.ico

Siguiendo esto, se crea la ventana de selección de tipo de grafo y se imprime el valor de cada variable para que el usuario seleccione una.

Teniendo eso, empieza la Creación del grafo, de la siguiente manera

if init.var.get() == "Grafo":  
 self.G = nx.Graph()  
elif init.var.get() == "Grafo Dirigido":  
 self.G = nx.DiGraph(directed=True)  
elif init.var.get() == "Multi Grafo":  
 self.G = nx.MultiGraph()  
elif init.var.get() == "Multi Grafo Dirigido":  
 self.G = nx.MultiDiGraph(directed=True)  
self.focus\_force()

luego iniciaremos los objetos para agregar aristas

self.label\_Add\_edge\_title = None  
self.label\_Add\_edge\_vertice\_o = None  
self.label\_Add\_edge\_vertice\_d = None  
self.label\_Add\_edge\_peso = None  
self.entry\_Add\_edge\_vertice\_o = None  
self.entry\_Add\_edge\_vertice\_d = None  
self.entry\_Add\_edge\_peso = None  
self.button\_Add\_edge = None  
self.frameAdd\_edge = ttk.Frame(self, style="TFrame")

lo siguiente es inicializar los objetos para agregar nodos

self.label\_Add\_node\_title = None  
self.label\_Add\_node = None  
self.entry\_Add\_node = None  
self.button\_Add\_node = None  
self.frameAdd\_node = ttk.Frame(self, style="TFrame")

lo siguiente es inicializar los objetos para eliminar nodos

self.label\_error\_delete = None  
self.label\_Delete\_node\_title = None  
self.label\_Delete\_node = None  
self.entry\_Delete\_node = None  
self.button\_Delete\_node = None  
self.frameDelete\_node = ttk.Frame(self, style="TFrame")

lo siguiente es Inicializar los objetos para el "tablero" o donde se creara la figura "Grafo"

self.frameFigure = ttk.Frame(self)  
self.figure = None

lo siguiente es inicializar los objetos para Inicializar url o path del archivo de guardado

self.urlGraph = None  
self.urlPNG = None  
self.urlPDF = None  
self.urlCSV\_export = None  
self.urlCSV\_import = None

lo siguiente es inicializar los menus

self.menu\_archivo = None  
self.menu\_analizar = None  
self.menu\_herramienta = None  
self.menu\_aplicacion = None  
self.menu\_ventana = None  
self.menu\_ayuda = None  
self.bar\_menu = None

y por último, en esta función se define la configuración de la ventana que se va a mostrar, dándole un tamaño de la siguiente manera

self.geometry("1200x700")  
self.title("")

hasta aquí tendríamos el método finalizado.

### def init\_menubar(self):

Crear barra de menú luego Define cuál será la barra de menús (Se deben añadir menús, else no se ve) luego va a la Creación de menú archivo, ahora se hace llamado a la Función que añade submenús, etc.  
Añado a la barra de menús y se hacen las Creaciones de menús.

def init\_menubar(self):  
 self.bar\_menu = tk.Menu()   
 self.config(menu=self.bar\_menu)  
  
 self.menu\_archivo = tk.Menu(self.bar\_menu, tearoff=False)   
 self.menu\_archivo, self.bar\_menu = self.menuarchivo(self.menu\_archivo,  
 self.bar\_menu)   
 self.bar\_menu.add\_cascade(menu=self.menu\_archivo, label="Archivo")   
  
 self.menu\_analizar = tk.Menu(self.bar\_menu, tearoff=False)   
 self.menu\_analizar, self.bar\_menu = self.menuanalizar(self.menu\_analizar,  
 self.bar\_menu)  
 self.bar\_menu.add\_cascade(menu=self.menu\_analizar, label="Analizar")  
  
 self.menu\_herramienta = tk.Menu(self.bar\_menu, tearoff=False)   
 self.menu\_herramienta, self.bar\_menu = self.menuherramienta(self.menu\_herramienta,  
 self.bar\_menu)   
 self.bar\_menu.add\_cascade(menu=self.menu\_herramienta, label="Herramienta")  
  
 self.menu\_aplicacion = tk.Menu(self.bar\_menu, tearoff=False)   
 self.menu\_aplicacion, self.bar\_menu = self.menuaplicacion(self.menu\_aplicacion,  
 self.bar\_menu)   
 self.bar\_menu.add\_cascade(menu=self.menu\_aplicacion, label="Aplicación")   
  
 self.menu\_ventana = tk.Menu(self.bar\_menu, tearoff=False)   
 self.menu\_ventana, self.bar\_menu = self.menuventana(self.menu\_ventana,  
 self.bar\_menu)   
 self.bar\_menu.add\_cascade(menu=self.menu\_ventana, label="Ventana")   
  
 self.menu\_ayuda = tk.Menu(self.bar\_menu, tearoff=False)   
 self.menu\_ayuda, self.bar\_menu = self.menuayuda(self.menu\_ayuda,  
 self.bar\_menu)   
 self.bar\_menu.add\_cascade(menu=self.menu\_ayuda, label="Ayuda")

### def init\_buttons(self):

**Text

Description automatically generated**

**Text

Description automatically generated**

**Text

Description automatically generated**

### def func\_actualizar\_figure(self, \*args):

**Text

Description automatically generated**

### def func\_eliminar\_nodo(self, \*args):

**Text

Description automatically generated**

### def func\_salta\_o\_d(self, \*args):

Función básica, atajo para dirigir el foco al siguiente campo, si se presiona enter, y el origen no es vacio.

def func\_salta\_o\_d(self, \*args):  
 if self.entry\_Add\_edge\_vertice\_o.get() != '':  
 self.entry\_Add\_edge\_vertice\_d.focus\_set()

### def func\_salta\_d\_p(self, \*args):

Función básica, atajo para dirigir el foco al siguiente campo, si se presiona enter, y el origen no es vacio.

def func\_salta\_d\_p(self, \*args):  
 if self.entry\_Add\_edge\_vertice\_d.get() != '': # Si el entry de destino no está vacío  
 self.entry\_Add\_edge\_peso.focus\_set()

### def func\_prueba(self, \*args):

La siguiente es una función para realizar pruebas, y verificar que si se esten desarrollando.

def func\_prueba(self, \*args): # Función de prueba  
 print("probado")

### def func\_agregar\_nodo(self, \*args):

def func\_agregar\_nodo(self, \*args): # Función para agregar nodos  
 if (self.entry\_Add\_node.get() != ''): # Si el entry de nombre nodo no está vacío  
 self.G.add\_node(self.entry\_Add\_node.get()) # Agrego el nodo  
  
 self.func\_actualizar\_figure() # Actualizo la gráfica  
  
 self.entry\_Add\_node.delete(0, tk.END) # Borro el entry de agregar nodos  
  
 print("Nodos:", list(self.G.nodes)) # Imprimo los nodos  
 self.entry\_Delete\_node.focus\_set() # Foco en el entry de eliminar nodos  
 self.entry\_Add\_node.focus\_set() # Foco en el entry de agregar nodos

### def func\_agregar\_arista(self, \*args):

**Text

Description automatically generated**

### def archivo\_nuevo\_presionado(self, \*args):

def archivo\_nuevo\_presionado(self, \*args):  
 print("¡Has presionado para crear un nuevo archivo!")

### def exportar\_CSV(self, \*args):

**Text

Description automatically generated**

### def importar\_CSV(self, \*args):

**Text

Description automatically generated**

### def exportar\_PDF(self, \*args):

def exportar\_PDF(self, \*args): # Función para exportar a PDF  
 if self.urlPDF == None: # Si el urlPDF es None  
 self.urlPDF = asksaveasfile(filetypes=[('PDF', '\*.pdf')],  
 defaultextension=[('PDF', '\*.pdf')],  
 initialfile="grafo\_en\_pdf.pdf") # Pido al usuario que guarde el archivo  
 if self.urlPDF != None: # Si el urlPDF no es None  
 print("PDF guardado en:", str(self.urlPDF.name)) # Imprimo el nombre del archivo  
 plt.savefig(str(self.urlPDF.name), format="PDF") # Guardo la gráfica en el archivo  
 self.urlPDF = None # Reseteo el urlPDF

### def exportar\_imagen(self, \*args):

def exportar\_imagen(self, \*args): # Función para exportar a PNG  
 if self.urlPNG == None: # Si el urlPNG es None  
 self.urlPNG = asksaveasfile(filetypes=[('PNG', '\*.png')],  
 defaultextension=[('PNG', '\*.png')],  
 initialfile="grafo\_en\_png.png") # Pido al usuario que guarde el archivo  
 if self.urlPNG != None: # Si el urlPNG no es None  
 print("Imagen guardada en:", str(self.urlPNG.name)) # Imprimo el nombre del archivo  
 plt.savefig(str(self.urlPNG.name), format="PNG") # Guardo la gráfica en el archivo  
 self.urlPNG = None # Reseteo el urlPNG

### def guardar\_archivo(self, \*args):

def guardar\_archivo(self, \*args): # Función para guardar el archivo  
 if self.urlGraph == None: # Si el urlGraph es None  
 self.urlGraph = asksaveasfile(filetypes=[('JSON Document', '\*.json')],  
 defaultextension=[('JSON Document', '\*.json')],  
 initialfile="nuevo\_archivo.json") # Pido al usuario que guarde el archivo  
 if self.urlGraph != None: # Si el urlGraph no es None  
 with open(self.urlGraph.name, "w") as fw: # Abro el archivo  
 json.dump(node\_link\_data(self.G), fw) # Guardo el archivo  
 fw.close() # Cierro el archivo

### def abrirarchivo(self, \*args):

def abrirarchivo(self, \*args): # Función para abrir un archivo  
 ubication = (askopenfile(title='Por favor, seleccione un archivo JSON.',  
 mode='r', filetypes=[('JSON Files', '\*.json')])) # Pido al usuario que seleccione un archivo  
 if ubication != None: # Si el archivo no es None  
 with open(ubication.name) as f: # Abro el archivo  
 js\_graph = json.load(f) # Cargo el archivo  
 self.G = json\_graph.node\_link\_graph(js\_graph) # Convierto el diccionario en una gráfica  
 self.func\_actualizar\_figure() # Actualizo la gráfica

### def menuarchivo(self, menu, bar\_menu):

def menuarchivo(self, menu, bar\_menu): # Función para el menú de archivo  
 sub\_menu\_archivo\_nuevo = tk.Menu(menu, tearoff=False) # Creo un submenú  
  
 menu.add\_command(  
 label="Abrir",  
 # accelerator="Ctrl+N",  
 command=self.abrirarchivo # Agrego el comando para abrir un archivo  
 )  
 menu.add\_command(  
 label="Guardar",  
 # accelerator="Ctrl+N",  
 command=self.guardar\_archivo # Agrego el comando para guardar un archivo  
 )  
 menu.add\_command(  
 label="Guardar como",  
 # accelerator="Ctrl+N",  
 command=self.guardar\_archivo # Agrego el comando para guardar un archivo  
 )  
  
 sub\_menu\_archivo\_exportar = tk.Menu(menu, tearoff=False) # Creo un submenú  
  
 sub\_menu\_archivo\_exportar.add\_command(  
 label="Excel",  
 # accelerator="Ctrl+N",  
 command=self.exportar\_CSV # Agrego el comando para exportar a CSV  
 )  
 sub\_menu\_archivo\_exportar.add\_command(  
 label="Imagen",  
 # accelerator="Ctrl+N",  
 command=self.exportar\_imagen # Agrego el comando para exportar a PNG  
 )  
 sub\_menu\_archivo\_exportar.add\_command(  
 label="PDF",  
 # accelerator="Ctrl+N",  
 command=self.exportar\_PDF # Agrego el comando para exportar a PDF  
 )  
 menu.add\_cascade(menu=sub\_menu\_archivo\_exportar, label="Exportar") # Agrego el submenú de exportar  
  
 menu.add\_command(  
 label="Importar datos",  
 # accelerator="Ctrl+N",  
 command=self.importar\_CSV # Agrego el comando para importar datos  
 )  
 menu.add\_command(  
 label="Imprimir",  
 # accelerator="Ctrl+N",  
 command=self.archivo\_nuevo\_presionado # Agrego el comando para imprimir  
 )  
  
 return menu, bar\_menu # Retorno el menu y la barra de menú

### def menuanalizar(self, menu, bar\_menu):

def menuanalizar(self, menu, bar\_menu):  
 sub\_menu\_analizar\_algoritmo = tk.Menu(menu, tearoff=False)  
 sub\_menu\_analizar\_algoritmo.add\_command(  
 label="Kernighan Lin",  
 # accelerator="Ctrl+N",  
 command=lambda: abrir\_ventana\_partition(self, self.G)  
 )  
 sub\_menu\_analizar\_algoritmo.add\_command(  
 label="Kernighan Lin",  
 # accelerator="Ctrl+N",  
 command=lambda: abrir\_ventana\_partition(self, self.G)  
 )  
 sub\_menu\_analizar\_algoritmo.add\_command(  
 label="Algoritmo 3",  
 # accelerator="Ctrl+N",  
 command=self.archivo\_nuevo\_presionado  
 )  
 sub\_menu\_analizar\_algoritmo.add\_command(  
 label="Algoritmo k",  
 # accelerator="Ctrl+N",  
 command=self.archivo\_nuevo\_presionado  
 )  
 menu.add\_cascade(menu=sub\_menu\_analizar\_algoritmo, label="Partitioner")  
  
 return menu, bar\_menu

### def menuherramienta(self, menu, bar\_menu):

def menuherramienta(self, menu, bar\_menu):  
 menu.add\_command(  
 label="Ejecuciones",  
 # accelerator="Ctrl+N",  
 command=self.archivo\_nuevo\_presionado  
 )  
 return menu, bar\_menu

### def menuaplicacion(self, menu, bar\_menu):

def menuaplicacion(self, menu, bar\_menu):  
 sub\_menu\_aplicacion = tk.Menu(menu, tearoff=False)  
 sub\_menu\_aplicacion.add\_command(  
 label="Aplicación 1",  
 # accelerator="Ctrl+N",  
 command=self.archivo\_nuevo\_presionado  
 )  
 sub\_menu\_aplicacion.add\_command(  
 label="Aplicación 2",  
 # accelerator="Ctrl+N",  
 command=self.archivo\_nuevo\_presionado  
 )  
 sub\_menu\_aplicacion.add\_command(  
 label="Aplicación 3",  
 # accelerator="Ctrl+N",  
 command=self.archivo\_nuevo\_presionado  
 )  
 sub\_menu\_aplicacion.add\_command(  
 label="Aplicación m",  
 # accelerator="Ctrl+N",  
 command=self.archivo\_nuevo\_presionado  
 )  
 menu.add\_cascade(menu=sub\_menu\_aplicacion, label="Aplicación")  
  
 return menu, bar\_menu

### def menuventana(self, menu, bar\_menu):

def menuventana(self, menu, bar\_menu):  
 menu.add\_command(  
 label="Gráfica",  
 # accelerator="Ctrl+N",  
 command=self.archivo\_nuevo\_presionado  
 )  
 menu.add\_command(  
 label="Tabla",  
 # accelerator="Ctrl+N",  
 command=self.archivo\_nuevo\_presionado  
 )  
 return menu, bar\_menu

### def menuayuda(self, menu, bar\_menu):

def menuayuda(self, menu, bar\_menu):  
 menu.add\_command(  
 label="Ayuda",  
 # accelerator="Ctrl+N",  
 command=self.archivo\_nuevo\_presionado  
 )  
 menu.add\_command(  
 label="Acerca de Grafos",  
 # accelerator="Ctrl+N",  
 command=self.archivo\_nuevo\_presionado  
 )  
 return menu, bar\_menu

## Class PartitionKL(tk.Toplevel):

Esta clase es para generar una ventada secundaria, la cual solo va a aparecer cuando sea requerida, en este caso aparecerá cuando el usuario desee mostrar una partición de un grafo, seleccionando el algoritmo de partición preferido para realizar esta separación.

En esta clase se va a tener una variable de control, esta es de tipo booleana y inicialmente estará en false, mientras no contenga un grafo para mostrar, esta no cambiara de estado.

Adicionalmente, esta clase contiene tres métodos, el constructor, el destructor y el método que abre la ventana de partición.

### def \_\_init\_\_(self, Graph, \*args, \*\*kwargs):

def \_\_init\_\_(self, Graph, \*args, \*\*kwargs):   
 super().\_\_init\_\_(\*args, \*\*kwargs) # Se almacena por herencia el \*args \*\*kwargs  
 self.\_\_class\_\_.en\_uso = True  
 sv\_ttk.set\_theme("dark")  
 ntkutils.dark\_title\_bar(self)  
 ntkutils.blur\_window\_background(self)  
 self.G = Graph  
 self.frameFigure = ttk.Frame(self)  
 self.frameFigure.grid(row=0, column=1, rowspan=4, pady=20)  
  
 self.figure = plt.figure(frameon=True, figsize=(7, 5), dpi=100)  
 canvas = FigureCanvasTkAgg(self.figure, master=self.frameFigure)  
  
 self.figure.set\_facecolor('#eafff5')  
 plt.axis('off')  
 communities = nx.algorithms.community.girvan\_newman(self.G, most\_valuable\_edge=None)  
  
 node\_groups = []  
 for com in next(communities):  
 node\_groups.append(list(com))  
  
 color\_map = []  
 for node in self.G:  
 if node in node\_groups[0]:  
 color\_map.append('blue')  
 else:  
 color\_map.append('green')  
 pos = nx.spring\_layout(self.G, 25)  
 nx.draw\_networkx(self.G, pos=pos, node\_color=color\_map, alpha=0.9, edge\_color='#bb85f0')  
 nx.draw\_networkx\_edge\_labels(self.G, pos, nx.get\_edge\_attributes(self.G, "weight"))  
  
 canvas.draw()  
 canvas.get\_tk\_widget().pack()

### def destroy(self):

def destroy(self): # Se cierra la ventana  
 # Restablecer el atributo al cerrarse.  
 self.\_\_class\_\_.en\_uso = False # Restablecer el atributo  
 return super().destroy() # Se llama al método destroy de la clase padre.

### def abrir\_ventana\_partition(self, Graph):

def abrir\_ventana\_partition(self, Graph): # Se abre la ventana secundaria  
 if not PartitionKL.en\_uso: # Si no está en uso  
 self.ventana\_secundaria = PartitionKL(Graph=Graph, master=self) # Se crea la ventana secundaria

## Class PartitionGN(tk.Toplevel):

Esta clase es para generar una ventada secundaria, la cual solo va a aparecer cuando sea requerida, en este caso aparecerá cuando el usuario desee mostrar una partición de un grafo, seleccionando el algoritmo de partición preferido para realizar esta separación.

En esta clase se va a tener una variable de control, esta es de tipo booleana y inicialmente estará en false, mientras no contenga un grafo para mostrar, esta no cambiará de estado.

Adicionalmente, esta clase contiene tres métodos, el constructor, el destructor y el método que abre la ventana de partición.

### def \_\_init\_\_(self, Graph, \*args, \*\*kwargs):

**Text

Description automatically generated**

**Text

Description automatically generated**

### def destroy(self):

def destroy(self): # Se cierra la ventana  
 # Restablecer el atributo al cerrarse.  
 self.\_\_class\_\_.en\_uso = False # Restablecer el atributo  
 return super().destroy() # Se llama al método destroy de la clase padre.

### def abrir\_ventana\_partition\_Girvan\_Newman(self, Graph):

def abrir\_ventana\_partition\_Girvan\_Newman(self, Graph): # Se abre la ventana secundaria  
 if not PartitionGN.en\_uso and len(Graph.nodes) > 0: # Si no está en uso  
 self.ventana\_secundaria = PartitionGN(Graph=Graph, master=self) # Se crea la ventana secundaria  
 ntkutils.placeappincenter(self.ventana\_secundaria) # Se coloca en el centro  
 if darkdetect.theme() == "Dark":  
 sv\_ttk.set\_theme("dark")  
 ntkutils.dark\_title\_bar(self.ventana\_secundaria)  
 ntkutils.blur\_window\_background(self.ventana\_secundaria)  
 else:  
 sv\_ttk.set\_theme("light")  
 ntkutils.blur\_window\_background(self.ventana\_secundaria)  
 self.ventana\_secundaria.focus() # Se pone el foco en la ventana

## Class PartitionLouvain(tk.Toplevel):

Esta clase es para generar una ventada secundaria, la cual solo va a aparecer cuando sea requerida, en este caso aparecerá cuando el usuario desee mostrar una partición de un grafo, seleccionando el algoritmo de partición preferido para realizar esta separación.

En esta clase se va a tener una variable de control, esta es de tipo booleana y inicialmente estará en false, mientras no contenga un grafo para mostrar, esta no cambiará de estado.

Adicionalmente, esta clase contiene tres métodos, el constructor, el destructor y el método que abre la ventana de partición.

### def \_\_init\_\_(self, Graph, \*args, \*\*kwargs):

def \_\_init\_\_(self, Graph, \*args, \*\*kwargs): # Queda abierto a n argumentos o n argumentos con identificador  
 super().\_\_init\_\_(\*args, \*\*kwargs) # Se almacena por herencia el \*args \*\*kwargs  
 self.\_\_class\_\_.en\_uso = True # Se indica que la ventana está en uso  
 self.G = Graph  
 self.frameFigure = ttk.Frame(self)  
 self.frameFigure.grid(row=1, column=0, rowspan=4, pady=20)  
  
 self.figure = plt.figure(frameon=True, figsize=(7, 5), dpi=100)  
 canvas = FigureCanvasTkAgg(self.figure, master=self.frameFigure)  
  
 self.figure.set\_facecolor('#eafff5')  
 plt.axis('off')  
  
 partition\_1, partition\_2 = nx.algorithms.community.louvain\_communities(self.G, resolution=0.9)  
 node\_groups = [list(partition\_1), list(partition\_2)]  
 edges\_subgraph1 = []  
 edges\_subgraph2 = []  
 edges\_subgraph\_over = []  
 color\_list = []  
 for edge in self.G.edges():  
 if edge[0] in node\_groups[0] and edge[1] in node\_groups[0]:  
 edges\_subgraph1.append(edge)  
 color\_list.append('#44e011')  
 elif edge[0] in node\_groups[1] and edge[1] in node\_groups[1]:  
 edges\_subgraph2.append(edge)  
 color\_list.append('#bb85f0')  
 else:  
 edges\_subgraph\_over.append(edge)  
 color\_list.append('red')  
 color\_map = []  
 for node in self.G:  
 if node in node\_groups[0]:  
 color\_map.append('#44e011')  
 else:  
 color\_map.append('#bb85f0')  
 pos = nx.spring\_layout(self.G, 25)  
 nx.draw\_networkx(self.G, pos=pos, node\_color=color\_map, alpha=0.9, edge\_color=color\_list)  
 nx.draw\_networkx\_edge\_labels(self.G, pos, nx.get\_edge\_attributes(self.G, "weight"))  
  
 canvas.draw()  
 canvas.get\_tk\_widget().pack()

### def destroy(self):

def destroy(self): # Se cierra la ventana  
 # Restablecer el atributo al cerrarse.  
 self.\_\_class\_\_.en\_uso = False # Restablecer el atributo  
 return super().destroy() # Se llama al método destroy de la clase padre.

### def abrir\_ventana\_partition\_Louvain(self, Graph):

def abrir\_ventana\_partition\_Louvain(self, Graph): # Se abre la ventana secundaria  
 if not PartitionLouvain.en\_uso and len(Graph.nodes) > 0: # Si no está en uso  
  
 self.ventana\_secundaria = PartitionLouvain(Graph=Graph, master=self) # Se crea la ventana secundaria  
 ntkutils.placeappincenter(self.ventana\_secundaria) # Se coloca en el centro  
 if darkdetect.theme() == "Dark":  
 sv\_ttk.set\_theme("dark")  
 ntkutils.dark\_title\_bar(self.ventana\_secundaria)  
 ntkutils.blur\_window\_background(self.ventana\_secundaria)  
 else:  
 sv\_ttk.set\_theme("light")  
 ntkutils.blur\_window\_background(self.ventana\_secundaria)  
 self.ventana\_secundaria.focus() # Se pone el foco en la ventana

Window\_start.py

import tkinter as tk  
from tkinter import ttk  
  
import sv\_ttk

Librerías necesarias para el funcionamiento de la ventana: Libreria tkinter y sv\_ttk.

## Class TypeGraph(tk.simpledialog.Dialog):

### def \_\_init\_\_(self, parent, title):

def \_\_init\_\_(self, parent, title):  
 self.ok\_button = None  
 self.var = None  
 self.choseOption = None  
 self.my\_option = None  
 self.Title\_label = None  
 self.GraphNormal = None  
 self.GraphDirigido = None  
 self.MultiGraph = None  
 self.MultiGraphDir = None  
 sv\_ttk.use\_dark\_theme() # Set dark theme  
 super().\_\_init\_\_(parent, title)

### def body(self, frame):

def body(self, frame):  
 self.var = tk.StringVar()  
 self.choseOption = ["Grafo", "Grafo Dirigido", "Multi Grafo", "Multi Grafo Dirigido"]  
 self.var.set(self.choseOption[0])  
 self.Title\_label = ttk.Label(frame, width=25, text="Seleccione el tipo de grafo")  
 self.Title\_label.grid(row=0, rowspan=2, column=0, pady=25)  
 self.GraphNormal = ttk.Radiobutton(frame, text="Grafo", variable=self.var, value="Grafo")  
 self.GraphNormal.grid(row=1, rowspan=2, column=0, pady=20)  
 self.GraphDirigido = ttk.Radiobutton(frame, text="Grafo Dirigido", variable=self.var, value="Grafo Dirigido")  
 self.GraphDirigido.grid(row=2, rowspan=2, column=0, pady=20)  
 self.MultiGraph = ttk.Radiobutton(frame, text="Multi Grafo", variable=self.var, value="Multi Grafo")  
 self.MultiGraph.grid(row=3, rowspan=2, column=0, pady=20)  
 self.MultiGraphDir = ttk.Radiobutton(frame, text="Multi Grafo Dirigido", variable=self.var,  
 value="Multi Grafo Dirigido")  
 self.MultiGraphDir.grid(row=4, rowspan=2, column=0, pady=20)  
 return frame

### def ok\_pressed(self):

def ok\_pressed(self):  
 self.destroy()

### def buttonbox(self):

def buttonbox(self):  
 self.ok\_button = ttk.Button(self, text='OK', width=40, command=self.ok\_pressed)  
 self.ok\_button.pack(fill="both")  
 self.bind("<Return>", lambda event: self.ok\_pressed())

# Referencias

Felipe Pulgar, Nicolas Suarez. (1 de Julio de 2022). *Manual de usuario*. Obtenido de https://unaledu-my.sharepoint.com/:w:/g/personal/nasuarezro\_unal\_edu\_co/ETpwXjkm37JEkstCYAC1eOIBkQhb9WeRse8PPxz6bnUGCw?rtime=hZbDiFte2kg

Ver [Manual de Usuario](https://unaledu-my.sharepoint.com/:w:/g/personal/nasuarezro_unal_edu_co/ETpwXjkm37JEkstCYAC1eOIBkQhb9WeRse8PPxz6bnUGCw)