Dibujo en blanco y negro

Descripción generada automáticamente con confianza media

**Estudio para el manejo de algoritmos sobre grafos haciendo un análisis de algunos algoritmos en el ámbito de problemas de optimización**

**Alfredo Valentierra Quiñonez**

**Andres Felipe Rodriguez**

**Luz Enith Guerrero Mendietai**

**Universidad Nacional de Colombia**

**Sede Manizales**

**Administración de sistemas informáticos**

**Análisis y diseño de algoritmos**

**2022-1**

Dibujo en blanco y negro

Descripción generada automáticamente con confianza media

**Estudio para el manejo de algoritmos sobre grafos haciendo un análisis de algunos algoritmos en el ámbito de problemas de optimización**

**Alfredo Valentierra Quiñonez**

**Andres Felipe Rodriguez**

**MANUAL TÉCNICO**

**Universidad Nacional de Colombia**

**Sede Manizales**

**Administración de sistemas informáticos**

**Análisis y diseño de algoritmos**

**2022-1**

INTRODUCCION:

El propósito de esta guía es proporcionar detalles del diseño del programa. Este fue creado para ayudar con el manejo de algoritmos para la manipulación de grafos. Este proyecto está desarrollado en Python y está conectado a Una base de datos Mongodb.

La primera parte de esta guía contiene una descripción de cómo hacer funcionar la aplicación y los requisitos para el mismo. En la segunda parte, se tendrá una pequeña descripción de la implementación del código fuente de la aplicación.

PRIMERA PARTE:

Antes de ejecutar la aplicación se debe instalar un editor de código (Ejm. Visual studio code) o simplemente tener instalado Python <https://www.python.org/downloads/release/python-380> para trabajarlo desde la consola de comandos del sistema, para este ejemplo usaremos simplemente Python y se estará trabajando desde la consola de comandos (cmd).

Después de haber instalado Python, se necesita instalar algunas librerías: Se debe entrar a la consola de comandos y empezar a instalar las siguientes librerías usando el comando “ pip install <nombre> --user” ejemplo “pip install fpdf –user”.

fpdf

matplotlib

openpyxl

reportlab

pymongo

bson

networkx

sklearn

Después de haber instalados esas librerías se procede a ejecutar la aplicación de la siguiente manera.

Interfaz de usuario gráfica, Aplicación, Word

Descripción generada automáticamente

Nota: Para acceder a una carpeta se usa el comando “cd “seguido del nombre de la carpeta, para retroceder se usa el comando “cd ..” con esos dos puntos seguidos.

Después de abrir el CMD, de debe acceder a la carpeta donde está el proyecto

En mi caso, puse la carpeta proyecto en el escritorio

>> cd onedrive\escritorio\proyecto

Lo siguiente es activar el entorno virtual

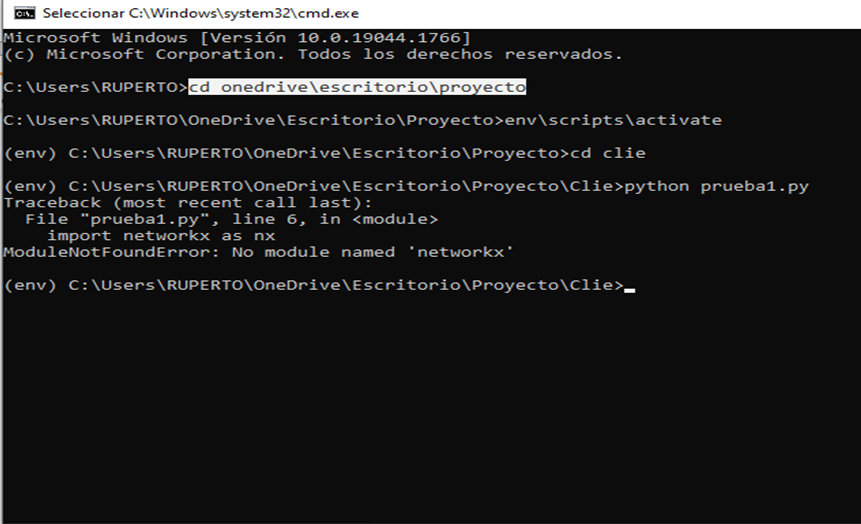
>>> env\scripts\activate

Debe de aparecer así (env) C:\Users\RUPERTO\OneDrive\Escritorio\Proyecto> con ese ( env), significa que ya está activado

El siguiente paso es acceder a la carpeta Clie

(env) C:\Users\RUPERTO\OneDrive\Escritorio\Proyecto>cd Clie

Después simplemente se debe ejecutar la aplicación con el comando “Python prueba1.py”



En caso de que no se ejecute y salga algo así, es porque hay librerías que no han sida instaladas como se muestra en el ejemplo, En otro Cmd se va instalando las librerías que el sistema le vaya pidiendo, usando el comando “pip install networkx” en caso de que salga algún error usar el siguiente comando “pip install networkx –user”

Interfaz de usuario gráfica, Aplicación, Word

Descripción generada automáticamente

SEGUNDA PARTE

1. En este apartado tenemos varias clases para el manejo de los datos del grafo.

La primera clase es “grafo” manejaremos el objeto de las estructuras de almacenamiento de los nodos y las aristas, la siguiente clase es “arista” donde manejará la estructura de datos que va tener las relaciones entre cada par de vértices (NodoOrigen, NodoDestino, CoordendasOrigen, coordenadasDestino) y por ultimo tenemos la clase “vertice” el cual maneja la estructura de cada nodo (nombreNodo, coordenada X, coordenada Y).

class grafo():

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

self.listav = []

self.listaA = []

#self.vertices = []

#self.matriz\_Adyac = [[None]\*len(self.listav)]\*len(self.listav)

def \_\_str\_\_(self):

return str(self.listav)

return str(self.matrizA)

def agregarVertice(self, v):

self.listav.append(v)

def agregarArista(self,a):

self.listaA.append(a)

class vertice():

def \_\_init\_\_(self, nombre, x, y):

self.la = []

self.ld = []

self.nombre = nombre

self.x = x

self.y = y

self.distancia = sys.maxsize

self.visitado = False

self.predecesor = None

def \_\_str\_\_(self):

return str(self.la)

def vecino(self, ver\_ad):

self.la.append(ver\_ad)

class arista():

def \_\_init\_\_(self, frm, to, distancia, x1, y1, x2, y2):

self.ld = []

self.frm = frm

self.to = to

self.distancia = distancia

#self.bloqueo = False

self.var = tkinter.IntVar()

self.x1 = x1

self.y1 = y1

self.x2 = x2

self.y2 = y2

1. **DESARROLLO DE LA INTERFAZ**

**Ventana, Canvas:**

Ventana = tk.Tk()

#Ventana.state("zoomed")

Ventana.geometry("750x650")

Ventana.title("Grafos")

#Ventana.iconbitmap('Clie\cp-logo.ico')

canvas3 = Canvas(Ventana, width=400, height=450, bg="#B3B6B7")#taget

canvas3.pack(fill=BOTH, expand=YES)

**Menú de opciones:**

barra\_menu=Menu(Ventana)

Ventana.config(menu= barra\_menu)

canvas3.bind("<Double-1>", dobleclick)

Archivo=tk.Menu(barra\_menu, tearoff=0)

Editar=tk.Menu(barra\_menu, tearoff=0)

Analizar=tk.Menu(barra\_menu, tearoff=0)

Herramienta=tk.Menu(barra\_menu, tearoff=0)

Aplicacion=tk.Menu(barra\_menu, tearoff=0)

Ventana=tk.Menu(barra\_menu, tearoff=0)

Ayuda=tk.Menu(barra\_menu, tearoff=0)

############ SubMenu Archivo

#############SubMenu Nuevo Grafo del SubMenu Archivo

sub\_Nuevo\_Grafo=tk.Menu(Archivo,tearoff=False)

sub\_Nuevo\_Grafo.add\_command(label='Personalizado')

sub\_Nuevo\_Grafo.add\_command(label='Aleatorio',command=Aleatorio)

Archivo.add\_cascade(label = 'Nuevo Grafo', menu=sub\_Nuevo\_Grafo)

Archivo.add\_command(label='Abrir',command=cargarmapa)

Archivo.add\_command(label='cerrar',command=quit)

Archivo.add\_command(label='Guardar',command=Guaradar\_JSON)

Sub\_Guardar\_datos=tk.Menu(Archivo, tearoff=False)

Sub\_Guardar\_datos.add\_command(label='XML', command=Guardar\_Como\_XML)

Sub\_Guardar\_datos.add\_command(label='JSON',command=Guaradar\_Como\_JSON)

Archivo.add\_cascade(label='Guardar Como',menu=Sub\_Guardar\_datos)

#Archivo.add\_command(label='Guardar Como',command=Guardar\_Como\_XML)

##############subMenu exportar datos del subMenu Archivo

Sub\_Exportar\_datos=tk.Menu(Archivo, tearoff=False)

Sub\_Exportar\_datos.add\_command(label='Excel',command=Guardar\_EXCEL)

Sub\_Exportar\_datos.add\_command(label='imagen', command=GuardarImagen)

Sub\_Exportar\_datos.add\_command(label='PDF', command=Guaradar\_PDF)

Archivo.add\_cascade(label = 'Exportar Grafo', menu=Sub\_Exportar\_datos)

Sub\_Importar\_datos=tk.Menu(Archivo, tearoff=False)

Sub\_Importar\_datos.add\_command(label='Desde la Base de datos',command=Importar\_Base)

Sub\_Importar\_datos.add\_command(label='Formato .JSON',command=Importar\_Json)

Sub\_Importar\_datos.add\_command(label='Formato .XML')

Archivo.add\_cascade(label = 'Importar Grafo', menu=Sub\_Importar\_datos)

#Archivo.add\_command(label='Importar datos',command=Importar\_Base)

Archivo.add\_command(label='Reiniciar/Limpiar',command=inicio)

Archivo.add\_command(label='Imprimir')

################ SubMenu Editar

Editar.add\_command(label='Deshacer')

############## SubMenu Nodo del SubMenu Editar

Sub\_Nodo=tk.Menu(Editar, tearoff=False)

Sub\_Nodo.add\_command(label='Agregar',command=dobleclick\_2 )

Sub\_Nodo.add\_command(label='Editar',command=clickeditarv)

Sub\_Nodo.add\_command(label='Eliminar', command=clickeliminarv)

Editar.add\_cascade(label = 'Nodo', menu=Sub\_Nodo)

################## SubMenu Archo del SubMenu Editar

Sub\_Arco=tk.Menu(Editar, tearoff=False)

Sub\_Arco.add\_command(label='Agregar',command=clickrelacion)

Sub\_Arco.add\_command(label='Eliminar', command=clickeliminara)

Sub\_Arco.add\_command(label='Editar',command=clickeditar)

Editar.add\_cascade(label = 'Arco', menu=Sub\_Arco)

################### SubMenu Algoritmos

################## SubMenu Algoritmo del SubMenu Algoritmo

Sub\_Algoritmo=tk.Menu(Analizar, tearoff=False)

Sub\_Algoritmo.add\_command(label='Queyranne',command=algoritmo1)

Sub\_Algoritmo.add\_command(label='Algorimo2')

Sub\_Algoritmo.add\_command(label='otros sin considerar')

Analizar.add\_cascade(label = 'Algoritmos', menu=Sub\_Algoritmo)

################## SubMenu Herramientas

Herramienta.add\_command(label='Ejecusion',command=Ejecusion)

##################### SubMenu Aplicacion

Aplicacion.add\_command(label='Aplicacion 1')

Aplicacion.add\_command(label='Aplicacion 2')

Aplicacion.add\_command(label='Agregar las demas')

##################### SubMenu Vnetana

Ventana.add\_command(label='Grafica',command=ventan\_grafica)

Ventana.add\_command(label='Tabla',command=ventan\_tabla)

##################### SubMenu ventana

Ayuda.add\_command(label='Ayuda')

Ayuda.add\_command(label='Acerca de Grafos')

barra\_menu.add\_cascade(label='Archivo',menu=Archivo)

barra\_menu.add\_cascade(label='Editar',menu=Editar)

barra\_menu.add\_cascade(label='Analizar',menu=Analizar)

barra\_menu.add\_cascade(label='Herrameienta',menu=Herramienta)

barra\_menu.add\_cascade(label='Aplicacion',menu=Aplicacion)

barra\_menu.add\_cascade(label='Ventana',menu=Ventana)

barra\_menu.add\_cascade(label='Ayuda',menu=Ayuda)

1. **FUNCIONES**

**Funciones para agregar Nodos:**

def dobleclick\_2():

# Ingresar Vertice

global nct

listaX=[188,419,361,260,311,408,227,546,116,249,588,341,197,554,97,515,430,467,577,324,145,190,97,332,215,358,256,291,328,59,114,473,445,447,168,106,302,511,396,413]

listaY=[145,190,97,332,215,358,256,291,328,59,114,473,445,447,168,106,302,511,396,413,188,419,361,260,311,408,227,546,116,249,588,341,197,554,97,515,430,467,577,324]

nct=nct+1

ingreso = Toplevel()

ingreso.title("Agregar Punto")

tnombre = Label(ingreso, text="Ingrese nombre del vertice:")

tnombre.grid(row=0, column=0)

nombrev = Entry(ingreso)

nombrev.grid(row=0, column=1)

agregar = Button(ingreso, text="Agregar",

command=lambda: agregarv(listaX[nct], listaY[nct], nombrev.get(), ingreso))

agregar.grid(row=3, columnspan=2)

def agregarv(x, y, nombre, ingreso):

try:

gradoa = float(100)

if (gradoa < 1 ):

ingreso.destroy()

#messagebox.showerror("ERROR", "El grado de accidentalidad no esta permitido")

else:

vtemp = vertice(nombre, x, y)

g.agregarVertice(vtemp)

ingreso.destroy()

actualizar()

except ValueError:

ingreso.destroy()

# messagebox.showerror("ERROR", "Usted no digito un grado de accidentalidad correcto")

Funciones para agregar aristas:

def agregarA(frm, to, distancia, x1, y1, x2, y2):

atemp = arista(frm, to, distancia, x1, y1, x2, y2)

g.agregarArista(atemp)

actualizar()

# Relacionar vertices

def clickrelacion():

vrelacion = Toplevel()

vrelacion.title("Agregar Ruta")

opciones1 = Listbox(vrelacion, exportselection=0)

opciones2 = Listbox(vrelacion, exportselection=0)

for v in g.listav:

opciones1.insert(END, v.nombre)

opciones2.insert(END, v.nombre)

opciones1.pack(side=LEFT)

opciones2.pack(side=LEFT)

#t1 = Label(vrelacion, text="Desde")

#t1.pack()

#nv1 = Entry(vrelacion)

#nv1.pack()

#t2 = Label(vrelacion, text="Hasta")

#t2.pack()

#nv2 = Entry(vrelacion)

#nv2.pack()

t3=Label(vrelacion, text="Ingrese el peso")

t3.pack()

nv3=Entry(vrelacion)

nv3.pack()

#relacionar = Button(vrelacion, text="Relacionar", command=lambda: relacion(nv1.get(), nv2.get(), vrelacion))

# relacionar.pack()

relacionar2 = Button(vrelacion, text="Dirigida",

command=lambda: relacion(opciones1.get(opciones1.curselection()),

opciones2.get(opciones2.curselection()),nv3.get()))

relacionar2.pack()

relacionar3 = Button(vrelacion, text="No Dirijida",

command=lambda: relacion\_No\_Dirijida(opciones1.get(opciones1.curselection()),

opciones2.get(opciones2.curselection()),nv3.get()))

relacionar3.pack()

def relacion(nv1, nv2,nv3):

if (nv1 == nv2):

#vrelacion.destroy()

messagebox.showinfo("Denegado", "No puede seleccionar el mismo vertice")

else:

for v in g.listav:

if (nv1 == v.nombre):

a = v

for v in g.listav:

if (nv2 == v.nombre):

b = v

a.vecino(b)

d = int(nv3)

agregarA(a, b, d, a.x, a.y, b.x, b.y)

# vrelacion.destroy()

actualizar()

def relacion\_No\_Dirijida(nv1, nv2,nv3):

if (nv1 == nv2):

#vrelacion.destroy()

messagebox.showinfo("Denegado", "No puede seleccionar el mismo punto de interes")

else:

for v in g.listav:

if (nv1 == v.nombre):

a = v

for v in g.listav:

if (nv2 == v.nombre):

b = v

a.vecino(b)

d = int(nv3)

agregarA(a, b, d, a.x, a.y, b.x, b.y)

#vrelacion.destroy()

relacion(nv2,nv1,nv3)

actualizar()

1. **FUNCIONES**

## Editar vertice

def clickeditarv():

veliminar = Toplevel()

veliminar.title("Editar vertice")

t1 = Label(veliminar, text="Selecione vertice de interes")

t1.pack()

opciones = Listbox(veliminar, exportselection=0)

for v in g.listav:

opciones.insert(END, v.nombre)

opciones.pack()

t2 = Label(veliminar, text="Ingrese Nuevo vertice")

t2.pack()

Nuevo\_nombrev = Entry(veliminar)

Nuevo\_nombrev.pack()

eliminar2 = Button(veliminar, text="Editar desde listas",

command=lambda: editarv(opciones.get(opciones.curselection()),Nuevo\_nombrev.get(), veliminar))

eliminar2.pack()

def editarv(nombrev,Nuevo, veliminar):

try:

for v in g.listav:

if (nombrev == v.nombre):

v.nombre=Nuevo

for a in g.listaA:

if nombrev == a.frm.nombre:

a.frm.nombre=Nuevo

if nombrev==a.to.nombre:

a.to.nombre=Nuevo

veliminar.destroy()

actualizar()

except:

veliminar.destroy()

messagebox.showerror("ERROR", "No fue editado")

# Eliminar Vertices

def clickeliminarv():

veliminar = Toplevel()

veliminar.title("Eliminar Punto")

t1 = Label(veliminar, text="Nombre punto de interes")

t1.pack()

opciones = Listbox(veliminar, exportselection=0)

for v in g.listav:

opciones.insert(END, v.nombre)

opciones.pack()

nombrev = Entry(veliminar)

nombrev.pack()

eliminar = Button(veliminar, text="Eliminar", command=lambda: eliminarv(nombrev.get(), veliminar))

eliminar.pack()

eliminar2 = Button(veliminar, text="Elimina desde listas",

command=lambda: eliminarv(opciones.get(opciones.curselection()), veliminar))

eliminar2.pack()

def eliminarv(nombrev, veliminar):

try:

for v in g.listav:

if (nombrev == v.nombre):

a = v

for i in v.la:

if (nombrev == i.nombre):

b = i

v.la.remove(b)

g.listav.remove(a)

c = 0

indice = []

for a in g.listaA:

if nombrev == a.frm.nombre or nombrev == a.to.nombre:

c += 1

indice.append(g.listaA.index(a))

for i in reversed(indice): # sorted(indice, reverse=True)

del g.listaA[i]

veliminar.destroy()

actualizar()

except:

veliminar.destroy()

messagebox.showerror("ERROR", "El punto no se encuentra")

# Eliminar Aristas

def clickeliminara():

veliminara = Toplevel()

veliminara.title("Eliminar Ruta")

desde = Listbox(veliminara, exportselection=0)

hasta = Listbox(veliminara, exportselection=0)

for v in g.listav:

desde.insert(END, v.nombre)

hasta.insert(END, v.nombre)

desde.pack(side=LEFT)

hasta.pack(side=LEFT)

t1 = Label(veliminara, text="Desde")

t1.pack()

nv1 = Entry(veliminara)

nv1.pack()

t2 = Label(veliminara, text="Hasta")

t2.pack()

nv2 = Entry(veliminara)

nv2.pack()

eliminar = Button(veliminara, text="Eliminar", command=lambda: eliminara(nv1.get(), nv2.get(), veliminara))

eliminar.pack()

eliminar2 = Button(veliminara, text="Eliminar desde listas",

command=lambda: eliminara(desde.get(desde.curselection()), hasta.get(hasta.curselection()),

veliminara))

eliminar2.pack()

def eliminara(desde, hasta, veliminara):

try:

for v in g.listav:

if (desde == v.nombre):

a = v

for i in a.la:

if (hasta == i.nombre):

b = i

a.la.remove(b)

for ar in g.listaA:

if (desde == ar.frm.nombre and hasta == ar.to.nombre):

temp = ar

g.listaA.remove(temp)

veliminara.destroy()

actualizar()

except:

print("No elimina ruta")

def clickeditar():

veliminara = Toplevel()

veliminara.title("Editar Arista")

desde = Listbox(veliminara, exportselection=0)

hasta = Listbox(veliminara, exportselection=0)

for v in g.listav:

desde.insert(END, v.nombre)

hasta.insert(END, v.nombre)

desde.pack(side=LEFT)

hasta.pack(side=LEFT)

t1 = Label(veliminara, text="Ingrese Nuevo valor")

t1.pack()

Nuevo\_valor = Entry(veliminara)

Nuevo\_valor.pack()

eliminar2 = Button(veliminara, text="Editar desde listas",

command=lambda: editara(desde.get(desde.curselection()), hasta.get(hasta.curselection()),Nuevo\_valor.get(),

veliminara))

eliminar2.pack()

def editara(desde, hasta,Nuevo, veliminara):

try:

for v in g.listaA:

if (desde == v.to.nombre and hasta==v.frm.nombre):

v.distancia=Nuevo

if (hasta == v.to.nombre and desde==v.frm.nombre):

v.distancia=Nuevo

veliminara.destroy()

actualizar()

except:

print("No Editada")

# Calcular Distancias

def distancias(a, b):

distancia = math.sqrt(math.pow(b.x - a.x, 2) + math.pow(b.y - a.y, 2))

return round(distancia, 2)

def cargarmapa():

listaX=[188,419,361,260,311,408,227,546,116,249,588,341,197,554,97,515,430,467,577,324]

listaY=[145,190,97,332,215,358,256,291,328,59,114,473,445,447,168,106,302,511,396,413]

try:

del g.listav[:]

del g.listaA[:]

file = askopenfile(title="Abrir mapa", filetypes=[("Archivo de texto", "\*.txt")])

ftemp = []

for t in file.readlines():

ftemp.append(t)

NumeroVertice=int(len(ftemp))

for n in range(0, NumeroVertice):

# vtemp=vertice(str(n), -0.859\*100, -0.963\*10)

vtemp=vertice(str(n), listaX[n], listaY[n])

#vtemp=vertice(str(n), int(random.randint(220, 540)), int(random.randint(230, 570)))

g.agregarVertice(vtemp)

actualizar()

cargarrelaciones(ftemp)

file.close()

except:

messagebox.showerror("No se cargo el Grafo")

# usando matriz de adyacencia

def cargarrelaciones(ftemp):

c=int(len(ftemp))

matrizA = [[None]\*c]\*c

for i in range(len(ftemp)):

temp3 = ftemp[i].split("\n")

temp=temp3[0].split(",")

for j in range(len(ftemp)):

#print("i:",i,"j:",j)""

if int(temp[j])!=0:

matrizA[i][j]=int(temp[j])

relacion(str(i), str(j),int(temp[j]))

else:

matrizA[i][j]=int(temp[j])

def mostrarprofundidad():

l = []

l = g.profundidad(g.listav[0], l)

for vert in l:

print(vert.nombre)

def algoritmo1():

global bandera

bandera=1

def algoritmo2():

global bandera

bandera=2

def algoritmo3():

global bandera

bandera=3

def algoritmo4():

global bandera

bandera=4

def Ejecusion():

try:

if bandera==1:

Bipart()

if bandera==2:

print

if bandera==3:

print

if bandera==4:

print

except:

messagebox.showerror("No ha")

def Guaradar\_PDF():

try:

ingreso = Toplevel()

ingreso.title("Guardar PDF")

tnombre = Label(ingreso, text="Ingrese nombre del Pdf:")

tnombre.grid(row=0, column=0)

nombrev = Entry(ingreso)

nombrev.grid(row=0, column=1)

agregar = Button(ingreso, text="Guardar",

command=lambda: PDF(nombrev.get(), ingreso))

agregar.grid(row=3, columnspan=2)

except:

messagebox.showerror("No se Guardo el PDF")

def PDF(nombre\_pdf,ingreso):

try:

num = ancho / 2

global image2

image2 = Image.new("RGB", (width, height), white)

draw2 = ImageDraw.Draw(image2)

for i in range(len(g.listav)):

draw2.ellipse([g.listav[i].x-60, g.listav[i].y-60, g.listav[i].x-60 + ancho, g.listav[i].y-60 + ancho], fill="#3498DB", outline ="green")

for i in range(len(g.listaA)):

if g.listaA[i].x1 >= g.listaA[i].x2 and g.listaA[i].y1 > g.listaA[i].y2:

draw2.line([g.listaA[i].x1-60 + num, g.listaA[i].y1-60, g.listaA[i].x2-60 + ancho, g.listaA[i].y2-60 + num],fill="#A3E4D7", width=3, joint=None)

if g.listaA[i].x1 > g.listaA[i].x2 and g.listaA[i].y1 < g.listaA[i].y2:

draw2.line([g.listaA[i].x1-60 + num, g.listaA[i].y1-60 + ancho, g.listaA[i].x2-60 + ancho,g.listaA[i].y2-60 + num],fill="#A3E4D7", width=3, joint=None)

if g.listaA[i].x1 <= g.listaA[i].x2 and g.listaA[i].y1 > g.listaA[i].y2:

draw2.line([g.listaA[i].x1-60 + num, g.listaA[i].y1-60, g.listaA[i].x2-60, g.listaA[i].y2-60 + num],fill="#A3E4D7", width=3,

joint="curve")

if g.listaA[i].x1 < g.listaA[i].x2 and g.listaA[i].y1 < g.listaA[i].y2:

draw2.line([g.listaA[i].x1-60 + num, g.listaA[i].y1-60 + ancho, g.listaA[i].x2-60, g.listaA[i].y2-60 + num],

fill="#A3E4D7",width=3, joint=None)

for i in range(len(g.listaA)):

if g.listaA[i].x1 >= g.listaA[i].x2 and g.listaA[i].y1 > g.listaA[i].y2:

tx1 = ((g.listaA[i].x1-60 + num) + (g.listaA[i].x2-60 + ancho)) / 2

ty1 = (g.listaA[i].y1-60 + (g.listaA[i].y2-60 - num)) / 2

draw2.text([tx1, ty1+ num], text=str(g.listaA[i].distancia),fill=black )

if g.listaA[i].x1 > g.listaA[i].x2 and g.listaA[i].y1 < g.listaA[i].y2:

tx1 = ((g.listaA[i].x1-60 + num) + (g.listaA[i].x2-60 + ancho)) / 2

ty1 = ((g.listaA[i].y1-60 - ancho) + (g.listaA[i].y2-60 + num)) / 2

draw2.text([tx1, ty1 + ancho], text=str(g.listaA[i].distancia),fill=black)

if g.listaA[i].x1 <= g.listaA[i].x2 and g.listaA[i].y1 > g.listaA[i].y2:

tx1 = ((g.listaA[i].x1-60 + num) + g.listaA[i].x2-60) / 2

ty1 = (g.listaA[i].y1-60 + (g.listaA[i].y2-60 - num)) / 2

draw2.text([tx1, ty1 + num], text=str(g.listaA[i].distancia),fill=black )

if g.listaA[i].x1 < g.listaA[i].x2 and g.listaA[i].y1 < g.listaA[i].y2:

tx1 = (g.listaA[i].x1-60 + (num + g.listaA[i].x2-60)) / 2

ty1 = ((g.listaA[i].y1-60 - ancho) + (g.listaA[i].y2-60 + num)) / 2

draw2.text([tx1, ty1 + ancho], text=str(g.listaA[i].distancia),fill=black )

for i in range(len(g.listav)):

nombre = str(g.listav[i].nombre)

if len(nombre) > 5:

nombre = nombre[0:4] + ".."

draw2.text([g.listav[i].x-60 + num, g.listav[i].y-60 + num], text=nombre, fill="#A93226" )

path = Path("C:\Grafos")

path.mkdir(parents=True, exist\_ok=True)

messagebox.showinfo("Se guardo en la carpeta : Grafos en el disco")

nombrePdf="C:\carpeta-graf\Grafo\_"+str(nombre\_pdf)+".pdf"

c = canvas.Canvas(nombrePdf)

ingreso.destroy()

c.drawString(220,800, "Particion de Grafo")

c.drawString(80, 480, "Criterio: Menor perdida de información (distancia entre nodos)")

c.drawString(80, 460, "Perdida: " )

c.drawString(150, 460, str(minimo))

c.drawString(80, 440, "Particion: ")

c.drawString(150, 440, str(particion))

c.drawString(80, 420, "Tiempo: ")

c.drawString(150, 420, str(end-star))

nommbreJPG=str(nombre\_pdf)+ ".jpg"

image2.save(nommbreJPG)

c.drawImage(nommbreJPG, 25, 480, 480, 270) # Dibujamos una imagen (IMAGEN, X,Y, WIDTH, HEIGH)

c.save()

except:

messagebox.showinfo("No se Guardo el pdf")

def Guardar\_EXCEL():

lista=['A','B','C','D','E','F','G','H','I','J','K','L','M','N','Ñ','O','P','Q','R','S','T','U','V','W','X','Y','Z']

wb = openpyxl.Workbook()

hoja = wb.active

c=0

for i in range(len(g.listav)):

cad=lista[0]+str(i+1)

hoja[cad]=g.listav[i].nombre

for j in range(len(g.listaA)):

cadena=str(g.listaA[j].frm.nombre)

if g.listav[i].nombre==cadena:

c=c+1

cad=lista[c]+str(i+1)

cadena2=str(g.listaA[j].to.nombre) +','+ str(g.listaA[j].distancia)

hoja[cad]=cadena2

c=0

file = asksaveasfile(mode="wb", title="Guardar Grafo en formato EXCEL", defaultextension=".xlsx",filetypes=[("Archivo de texto", "\*.xlsx")])

wb.save(file)

def quit():

Ventana.destroy()

def GuardarMatriz():

try:

global filename

filename = "mp.txt"

archivo = asksaveasfile(mode="w", title="Guardar mapa", defaultextension=".txt",filetypes=[("Archivo de texto", "\*.txt")])

for i in range(len(g.listav)):

for j in range(len(g.listav)):

archivo.writelines(str(int(matriz\_Adyac[i][j])))

archivo.writelines(", ")

archivo.writelines("\n")

archivo.close()

except:

messagebox.showerror("No se guardo la Matriz")

def GuardarMatriz2():

try:

global filename

filename = "mp.txt"

archivo = asksaveasfile(mode="w", title="Guardar Matriz", defaultextension=".txt",

filetypes=[("Archivo de texto", "\*.txt")])

#archivo=open(fil, "w")

for i in range(len(g.listav)):

for j in range(len(g.listav)):

archivo.write(matriz\_Adyac[i][j])

archivo.write("\n")

archivo.close()

except:

messagebox.showerror("No se guardo la Matriz")

def Guaradar\_JSON():

### Guardamos en formato JSON el grafo

data3={}

data3['datos']=[]

data4={}

data4['grafico']=[]

data2=[]

try:

#data['Grafo'].append({"\_id":1})

for i in range(len(g.listav)):

# cadena= g.listav[i].nombre

#data['grafico'].append()

# data['Grafo'].append({"id":i+1,"etiqueta":cadena,'cordenadas:':{'x:':g.listav[i].x,'y:':g.listav[i].y} })

#data['Grafo'].append("vinculado a: ")

for j in range(len(g.listaA)):

if g.listaA[j].frm.nombre==g.listav[i].nombre:

data2.append( {'nodeId':g.listaA[j].to.nombre,'distancia':g.listaA[j].distancia })

data3['datos'].append({"id":g.listav[i].nombre, "etiqueta":"N"+str(i+1), "datos": {}, "tipo": "Objeto" , "vinculado a": data2, "radio": 0.4, "coordenadas": {"x": 25, "y": 34 }} )

data2=[]

data4['grafico'].append(data3)

# data.append({data4})

#cadena='N\_'

#cadena= cadena + str(g.listav[i].nombre)

#data['Grafo'].append({"identificacion":i+1,"etiqueta":cadena,'cordenadas:':{'x:':g.listav[i].x,'y:':g.listav[i].y} })

#data['Grafo'].append("vinculado a: ")

#for j in range(len(g.listaA)):

# if g.listaA[j].frm.nombre==g.listav[i].nombre:

# data['Grafo'].append({ 'Id. Nodo':g.listaA[j].to.nombre,'distancia':g.listaA[j].distancia})

with open('Formato3.json', 'w') as file:

json.dump(data4, file, indent=4)

except:

messagebox.showerror("No se Gurado el Grafo")

###Guardamos en la base de datos el grafo

try:

client = pymongo.MongoClient("mongodb://avalentierra:MHRurv73Alifer@cluster0-shard-00-00.rbl8j.mongodb.net:27017,cluster0-shard-00-01.rbl8j.mongodb.net:27017,cluster0-shard-00-02.rbl8j.mongodb.net:27017/Grafos?ssl=true&replicaSet=atlas-9cwijg-shard-0&authSource=admin&retryWrites=true&w=majority")

db = client['Grafos']

col=db['GRA']

col.insert\_one(data4)

messagebox.showinfo("!Se cargó exitosamente!")

except:

messagebox.showinfo("Error: Vaya a Ayuda")

def Guardar\_Como\_XML():

datas = ET.Element('Grafo')

try:

for i in range(len(g.listav)):

cadena='N\_'

cadena= cadena + str(g.listav[i].nombre)

cadena1="[ identificacion : "+ str(i) + " etiqueta :" + str(cadena) + ' [cordenadas :' + str('x:'+str(g.listav[i].x)+ 'y:'+str(g.listav[i].y)+']')+']'

ET.SubElement(datas,cadena1)

ET.SubElement(datas, 'Vinculado a:')

for j in range(len(g.listaA)):

if g.listaA[j].frm.nombre==g.listav[i].nombre:

cadena1='(Id. Nodo :'+str(g.listaA[j].to.nombre)+', distancia :'+str(g.listaA[j].distancia )+')'

ET.SubElement(datas,cadena1)

mydata = ET.tostring(datas)

myfile = asksaveasfile(mode="w", title="Guardar Grafo en formato XML", defaultextension=".xml",

filetypes=[("Archivo de texto", "\*.xml")])

myfile.write(str(mydata))

except:

messagebox.showerror("No se guardo el XML")

def Guaradar\_Como\_JSON():

### Guardamos en formato JSON el grafo

data3={}

data3['datos']=[]

data4={}

data4['grafico']=[]

data2=[]

try:

for i in range(len(g.listav)):

for j in range(len(g.listaA)):

if g.listaA[j].frm.nombre==g.listav[i].nombre:

data2.append( {'nodeId':g.listaA[j].to.nombre,'distancia':g.listaA[j].distancia })

data3['datos'].append({"id":g.listav[i].nombre, "etiqueta":"N"+str(i+1), "datos": {}, "tipo": "Objeto" , "vinculado a": data2, "radio": 0.4, "coordenadas": {"x": 25, "y": 34 }} )

data2=[]

data4['grafico'].append(data3)

file = asksaveasfile(mode="w", title="Guardar Grafo en formato .JSON", defaultextension=".json",

filetypes=[("Archivo de texto", "\*.json")])

json.dump(data4, file, indent=4)

except:

messagebox.showerror("No se Gurado el Grafo")

def buscar(a):

for i in range(len(g.listav)):

if a==g.listav[i].nombre:

return i

def ventan\_tabla():

tabla = Tk()

tabla.title("Matriz de Adyacencia")

tabla.geometry("620x590")

cantidad=len(g.listav)

global matriz\_Adyac

matriz\_Adyac = np.zeros((cantidad,cantidad))

text\_area = scrolledtext.ScrolledText(tabla, wrap = tk.WORD, width = 65, height = 25, font = ("Times New Roman",15))

text\_area.grid(row=0,column =1 )

for i in range(len(g.listav)):

for j in range(len(g.listaA)):

if g.listav[i].nombre==g.listaA[j].frm.nombre:

frm=buscar(g.listaA[j].frm.nombre)

to=buscar(g.listaA[j].to.nombre)

matriz\_Adyac[frm][to]=g.listaA[j].distancia

text\_area.insert(END, matriz\_Adyac)

Butt = Button(tabla, text="Guardar Matriz",

command=lambda: GuardarMatriz())

Butt.grid(row=1,column=1)

def ventan\_grafica():

tabla = Tk()

tabla.title("Grafo")

tabla.geometry("500x500")

canvas\_grafica = Canvas(tabla, width=400, height=400, bg="#B3B6B7")#taget

canvas\_grafica.pack(fill=BOTH, expand=YES)

Butt = Button(tabla, text="Guardar Grafico",

command=lambda: GuardarImagen())

Butt.pack()

num = ancho / 2

canvas\_grafica.delete("all")

for i in range(len(g.listav)):

canvas\_grafica.create\_oval(g.listav[i].x-80, g.listav[i].y-80, g.listav[i].x-80 + ancho, g.listav[i].y-80 + ancho, fill="#3498DB", width=0)

canvas\_grafica.create\_oval(g.listav[i].x+5-80, g.listav[i].y+5-80, g.listav[i].x-80 + ancho-5, g.listav[i].y-80 + ancho-5, fill="#3498DB", activefill="#E67E22", width=0)

for i in range(len(g.listaA)):

if g.listaA[i].x1 >= g.listaA[i].x2 and g.listaA[i].y1 > g.listaA[i].y2:

canvas\_grafica.create\_line(g.listaA[i].x1-80 + num, g.listaA[i].y1-80, g.listaA[i].x2-80 + ancho, g.listaA[i].y2-80 + num,

width=3, fill="#A3E4D7", arrow="last", smooth=True)

if g.listaA[i].x1 > g.listaA[i].x2 and g.listaA[i].y1 < g.listaA[i].y2:

canvas\_grafica.create\_line(g.listaA[i].x1-80 + num, g.listaA[i].y1-80 + ancho, g.listaA[i].x2-80 + ancho,

g.listaA[i].y2-80 + num, width=3, fill="#A3E4D7", arrow="last", smooth=True)

if g.listaA[i].x1 <= g.listaA[i].x2 and g.listaA[i].y1 > g.listaA[i].y2:

canvas\_grafica.create\_line(g.listaA[i].x1-80 + num, g.listaA[i].y1-80, g.listaA[i].x2-80, g.listaA[i].y2-80 + num, width=3,

fill="#A3E4D7", joint=None)

if g.listaA[i].x1 < g.listaA[i].x2 and g.listaA[i].y1 < g.listaA[i].y2:

canvas\_grafica.create\_line(g.listaA[i].x1-80 + num, g.listaA[i].y1-80 + ancho, g.listaA[i].x2-80, g.listaA[i].y2-80 + num,

width=3, fill="#A3E4D7", arrow="last", smooth=True)

for i in range(len(g.listaA)):

if g.listaA[i].x1 >= g.listaA[i].x2 and g.listaA[i].y1 > g.listaA[i].y2:

tx1 = ((g.listaA[i].x1-80 + num) + (g.listaA[i].x2-80 + ancho)) / 2

ty1 = (g.listaA[i].y1-80 + (g.listaA[i].y2-80 - num)) / 2

canvas\_grafica.create\_text([tx1, ty1 + num], text=str(g.listaA[i].distancia) )

if g.listaA[i].x1 > g.listaA[i].x2 and g.listaA[i].y1 < g.listaA[i].y2:

tx1 = ((g.listaA[i].x1-80 + num) + (g.listaA[i].x2-80 + ancho)) / 2

ty1 = ((g.listaA[i].y1-80 - ancho) + (g.listaA[i].y2-80 + num)) / 2

canvas\_grafica.create\_text(tx1, ty1 + ancho, text=str(g.listaA[i].distancia) )

if g.listaA[i].x1 <= g.listaA[i].x2 and g.listaA[i].y1 > g.listaA[i].y2:

tx1 = ((g.listaA[i].x1-80 + num) + g.listaA[i].x2-80) / 2

ty1 = (g.listaA[i].y1-80 + (g.listaA[i].y2-80 - num)) / 2

canvas\_grafica.create\_text(tx1, ty1 + num, text=str(g.listaA[i].distancia) )

if g.listaA[i].x1 < g.listaA[i].x2 and g.listaA[i].y1 < g.listaA[i].y2:

tx1 = (g.listaA[i].x1-80 + (num + g.listaA[i].x2-80)) / 2

ty1 = ((g.listaA[i].y1-80 - ancho) + (g.listaA[i].y2-80 + num)) / 2

canvas\_grafica.create\_text(tx1, ty1 + ancho, text=str(g.listaA[i].distancia) )

for i in range(len(g.listav)):

nombre = str(g.listav[i].nombre)

if len(nombre) > 5:

nombre = nombre[0:4] + ".."

canvas\_grafica.create\_text(g.listav[i].x-80 + num, g.listav[i].y-80 + num, text=nombre, fill="#A93226", font="bold")

def GuardarImagen():

num = ancho / 2

fileI = asksaveasfile(mode="w", title="Guardar grafo como Imagen .PNG", defaultextension=".jpg",filetypes=[("Archivo de texto", "\*.jpg")])

filename2 = fileI

global image1

image1 = Image.new("RGB", (width, height), white)

draw = ImageDraw.Draw(image1)

#os.startfile(filename2)

for i in range(len(g.listav)):

draw.ellipse([g.listav[i].x-60, g.listav[i].y-60, g.listav[i].x-60 + ancho, g.listav[i].y-60 + ancho], fill="#3498DB", outline ="green")

for i in range(len(g.listaA)):

if g.listaA[i].x1 >= g.listaA[i].x2 and g.listaA[i].y1 > g.listaA[i].y2:

draw.line([g.listaA[i].x1-60 + num, g.listaA[i].y1-60, g.listaA[i].x2-60 + ancho, g.listaA[i].y2-60 + num],

fill="#A3E4D7", width=3, joint=None)

if g.listaA[i].x1 > g.listaA[i].x2 and g.listaA[i].y1 < g.listaA[i].y2:

draw.line([g.listaA[i].x1-60 + num, g.listaA[i].y1-60 + ancho, g.listaA[i].x2-60 + ancho,

g.listaA[i].y2-60 + num],fill="#A3E4D7", width=3, joint=None)

if g.listaA[i].x1 <= g.listaA[i].x2 and g.listaA[i].y1 > g.listaA[i].y2:

draw.line([g.listaA[i].x1-60 + num, g.listaA[i].y1-60, g.listaA[i].x2-60, g.listaA[i].y2-60 + num],fill="#A3E4D7", width=3,

joint="curve")

if g.listaA[i].x1 < g.listaA[i].x2 and g.listaA[i].y1 < g.listaA[i].y2:

draw.line([g.listaA[i].x1-60 + num, g.listaA[i].y1-60 + ancho, g.listaA[i].x2-60, g.listaA[i].y2-60 + num],

fill="#A3E4D7",width=3, joint=None)

for i in range(len(g.listaA)):

if g.listaA[i].x1 >= g.listaA[i].x2 and g.listaA[i].y1 > g.listaA[i].y2:

tx1 = ((g.listaA[i].x1-60 + num) + (g.listaA[i].x2-60 + ancho)) / 2

ty1 = (g.listaA[i].y1-60 + (g.listaA[i].y2-60 - num)) / 2

draw.text([tx1, ty1 + num], text=str(g.listaA[i].distancia),fill=black )

if g.listaA[i].x1 > g.listaA[i].x2 and g.listaA[i].y1 < g.listaA[i].y2:

tx1 = ((g.listaA[i].x1-60 + num) + (g.listaA[i].x2-60 + ancho)) / 2

ty1 = ((g.listaA[i].y1-60 - ancho) + (g.listaA[i].y2-60 + num)) / 2

draw.text([tx1, ty1 + ancho], text=str(g.listaA[i].distancia),fill=black)

if g.listaA[i].x1 <= g.listaA[i].x2 and g.listaA[i].y1 > g.listaA[i].y2:

tx1 = ((g.listaA[i].x1-60 + num) + g.listaA[i].x2-60) / 2

ty1 = (g.listaA[i].y1-60 + (g.listaA[i].y2-60 - num)) / 2

draw.text([tx1, ty1 + num], text=str(g.listaA[i].distancia),fill=black )

if g.listaA[i].x1 < g.listaA[i].x2 and g.listaA[i].y1 < g.listaA[i].y2:

tx1 = (g.listaA[i].x1-60 + (num + g.listaA[i].x2-60)) / 2

ty1 = ((g.listaA[i].y1-60 - ancho) + (g.listaA[i].y2-60 + num)) / 2

draw.text([tx1, ty1 + ancho], text=str(g.listaA[i].distancia),fill=black )

for i in range(len(g.listav)):

nombre = str(g.listav[i].nombre)

if len(nombre) > 5:

nombre = nombre[0:4] + ".."

draw.text([g.listav[i].x-60 + num, g.listav[i].y-60 + num], text=nombre, fill="#A93226" )

image1.save(filename2)

############## IMPORTAR INFORMACION

def cargarDocumento\_Formato\_JSON(data2):

c=0

for i in data2['grafico']:

listaX=[188,419,361,260,311,408,227,546,116,249,588,341,197,554,97,515,430,467,577,324,145,190,97,332,215,358,256,291,328,59,114,473,445,447,168,106,302,511,396,413]

listaY=[145,190,97,332,215,358,256,291,328,59,114,473,445,447,168,106,302,511,396,413,188,419,361,260,311,408,227,546,116,249,588,341,197,554,97,515,430,467,577,324]

for j in i['datos']:

id\_Nodo\_Origen=(j['id']) ### de esta forma acede al id del nodo

control=0

for h in g.listav:

if str(id\_Nodo\_Origen) == str(h.nombre): ##### Tal vez aqui haya un error

control=1

if control==0:

vtemp=vertice(str(id\_Nodo\_Origen), int(listaX[c]), int(listaY[c]))

c=c+1

g.agregarVertice(vtemp)

actualizar()

#### Nodos adyacentes

keysLista=list(j['vinculado a'])

for k in keysLista:

id\_Nodo\_Destino=k['nodeId']

dist=k['distancia']

control=0

for p in g.listav:

if str(id\_Nodo\_Destino) == str(p.nombre):

control=1

for q in g.listav:

if str(id\_Nodo\_Origen)== str(q.nombre):

relacion(str(id\_Nodo\_Origen), str(id\_Nodo\_Destino),int(dist))

#atemp = arista(str(id\_Nodo\_Origen), str(id\_Nodo\_Destino), int(dist), int(p.x), int(p.y), int(q.x), int(q.y))

#g.agregarArista(atemp)

#actualizar()

break

if control==0:

vtemp=vertice(str(id\_Nodo\_Destino), int(listaX[c]), int(listaY[c]))

c=c+1

g.agregarVertice(vtemp)

actualizar()

for p in g.listav:

if str(id\_Nodo\_Destino)==str(p.nombre):

for q in g.listav:

if str(id\_Nodo\_Origen)== str(q.nombre):

relacion(str(id\_Nodo\_Origen), str(id\_Nodo\_Destino),int(dist))

# atemp = arista(str(id\_Nodo\_Origen), str(id\_Nodo\_Destino), int(dist), int(q.x),int(q.y),int(p.x),int(p.y))

#g.agregarArista(atemp)

#actualizar()

break

def Importar\_Base():

Base\_datos = Toplevel()

Base\_datos.title("Selecione el documento")

desde = Listbox(Base\_datos, width=60, height=10, exportselection=0)

try:

client = pymongo.MongoClient("mongodb://avalentierra:MHRurv73Alifer@cluster0-shard-00-00.rbl8j.mongodb.net:27017,cluster0-shard-00-01.rbl8j.mongodb.net:27017,cluster0-shard-00-02.rbl8j.mongodb.net:27017/Grafos?ssl=true&replicaSet=atlas-9cwijg-shard-0&authSource=admin&retryWrites=true&w=majority")

db = client['Grafos']

col=db['GRA']

for doc in col.find():

desde.insert(END, doc['\_id'])

desde.grid(row=1, column=0)

except:

messagebox.showinfo("Error\_Base\_datos")

t1 = Label(Base\_datos, text="Documentos \_ Base de Datos")

t1.grid(row=0, column=0)

caminob = Button(Base\_datos, text="Cargar",

command=lambda: Base(desde.get(desde.curselection()),Base\_datos))

caminob.grid(row=1, column=2)

#IMPORTAR DE LA BASE DE DATOS

def Base(id, Base\_datos):

Base\_datos.destroy()

try:

client = pymongo.MongoClient("mongodb://avalentierra:MHRurv73Alifer@cluster0-shard-00-00.rbl8j.mongodb.net:27017,cluster0-shard-00-01.rbl8j.mongodb.net:27017,cluster0-shard-00-02.rbl8j.mongodb.net:27017/Grafos?ssl=true&replicaSet=atlas-9cwijg-shard-0&authSource=admin&retryWrites=true&w=majority")

db = client['Grafos']

col=db['GRA']

for doc in col.find({"\_id":ObjectId(str(id))}):

cargarDocumento\_Formato\_JSON(doc)

#messagebox.showinfo("¡!Exit!¡")

except:

messagebox.showinfo("Error\_Base\_datos")

def Importar\_Json():

# try:

filename = askopenfile(title="Abrir JSON", filetypes=[("Formato Json", "\*.json")])

documento=json.load(filename)

cargarDocumento\_Formato\_JSON(documento)

# except:

# messagebox.showinfo("Error\_ .Json")

def inicio():

canvas3.delete("all")

g.listav=[]

g.listaA=[]

def k\_subset(s, k):

if k == len(s):

return (tuple([(x,) for x in s]),)

k\_subs = []

for i in range(len(s)):

partials = k\_subset(s[0:i] + s[i + 1:len(s)], k)

for partial in partials:

for p in range(len(partial)):

k\_subs.append(partial[:p] + (partial[p] + (s[i],),) + partial[p + 1:])

return k\_subs

def uniq\_subsets(s):

u = set()

for x in s:

t = []

for y in x:

y = list(y)

y.sort()

t.append(tuple(y))

t.sort()

u.add(tuple(t))

return u

def find\_partitions(V,k):

k\_subs = k\_subset(V,k)

k\_subs = uniq\_subsets(k\_subs)

return k\_subs

def desintegrar(vector2):

temp=[]

lista=[]

for i in range(1,len(vector2)):

if vector2[i]==")" and vector2[i-1]==")":

temp.append(vector2[i])

cade="".join(temp)

lista.append(cade)

temp=[]

else:

temp.append(vector2[i])

return lista

def Listas\_Bipartes(cadena):

lista1=[]

lista2=[]

c=0

for i in range(0,len(cadena)):

if cadena[i]!="(":

if cadena[i]!=",":

if cadena[i]!=" ":

if cadena[i]!="'":

if cadena[i]!=")":

lista1.append(cadena[i])

else:

c=c+1

break

c=c+1

for i in range(c,len(cadena)):

if cadena[i]!="(":

if cadena[i]!=",":

if cadena[i]!=" ":

if cadena[i]!="'":

if cadena[i]!=")":

lista2.append(cadena[i])

else:

break

return lista1,lista2

def buscar\_relacion(a,b,y):

distancia=0

for i in g.listaA:

if i.frm.nombre==a and i.to.nombre==b:

i.color=y

distancia=i.distancia

break

return distancia

def MinimaPerdida(lista):

minimo=99999

biparticion=" "

for i in range(len(lista)):

temp1,temp2=Listas\_Bipartes(lista[i])

# Hcaer la comparacion de las conexiones entre ambas listas

acumular=0

for t1 in range(len(temp1)):

a=temp1[t1]

for t2 in range(len(temp2)):

b=temp2[t2]

peso=buscar\_relacion(a,b,-1)

acumular=acumular+peso

if acumular<minimo:

minimo=acumular

biparticion=lista[i]

#print(minimo,biparticion)

return minimo,biparticion

def color(tupla):

lista1=[]

lista2=[]

c=0

for i in range(len(tupla)):

if tupla[i]!="'":

if tupla[i]!=" ":

if tupla[i]!=",":

if tupla[i]!="(":

if tupla[i]!=")":

lista1.append(tupla[i])

else:

c=c+1

break

c=c+1

for i in range(c,len(tupla)):

if tupla[i]!="'":

if tupla[i]!=" ":

if tupla[i]!=",":

if tupla[i]!="(":

if tupla[i]!=")":

lista2.append(tupla[i])

else:

break

for i in range(len(lista1)):

a=lista1[i]

for j in range(len(lista2)):

b=lista2[j]

buscar\_relacion(a,b,1)

buscar\_relacion(b,a,1)

def Actualiza\_color():

for i in g.listaA:

i.color=-1

def Bipart():

global minimo

global particion

global end

global star

Actualiza\_color()

lista=[]

for i in g.listav:

lista.append(i.nombre)

star=time.time()

vector=str(find\_partitions(lista,2))

temp=desintegrar(vector)

minimo,particion=MinimaPerdida(temp)

color(str(particion))

actualizar\_color\_cortes()

end=time.time()

#cadena="Perdida\_info: "+str(minimo)+ " Particion: "+ str(particion)+ " Tiempo: "+str(end-star)

#canvas3.create\_text(400,30, text=cadena, fill="#A93226", font="bold")

canvas3.create\_text(400,10, text=str("Criterio de particion: menor perdida de informacion( distancia) entre los nodos que se corrtan"), fill="#000000", font="bold")

canvas3.create\_text(400,30, text=str("Perdida: ")+str(minimo), fill="#000000", font="bold")

canvas3.create\_text(400,50, text=str("Particion: ")+str(particion), fill="#000000", font="bold")

canvas3.create\_text(400,70, text=str("Tiempo: ")+str(end-star), fill="#000000", font="bold")

def agregar\_arista(G, u, v, Peso,di):

G.add\_edge(u, v, weight=Peso)

# Si el grafo no es dirigido

if not di:

# Agrego otra arista en sentido contrario

G.add\_edge(v, u, weight=Peso)

def Aleatorio():

G=nx.DiGraph()

inicio()

listaX=[188,419,361,260,311,408,227,546,116,249,588,341,197,554,97,515,430,467,577,324]

listaY=[145,190,97,332,215,358,256,291,328,59,114,473,445,447,168,106,302,511,396,413]

lista\_N=['A','B','C','D','E','F','G','H','J','K','L','W','R','T','Y','U','O','P','Z','X','V','N']

lista\_nodos=[]

listaMA=[]

listaMa0=[]

listaMa0.append("0")

listaMa0.append("0")

listaMA.append(listaMa0)

aleat=random.randint(2,8)# cantidad de nodos que tendrá el grafo aleatorio

for i in range(aleat):

lista\_nodos.append(lista\_N[i])

# agregamos los nodos a la listaV

for i in range(aleat):

vtemp=vertice(str(lista\_N[i]), listaX[i], listaY[i])

g.agregarVertice(vtemp)

actualizar()

# agregamos aristas a la listaA

for i in range(len(lista\_nodos)):

to1=lista\_nodos[i]

aristas\_por\_nodo=random.randint(1,3)

c=0

bandt=0

while c<=aristas\_por\_nodo:

#print("WWWW")

frm1=random.choice(lista\_nodos)

#print(to1,frm1)

if frm1!=to1:

c=c+1

peso=random.randint(1, 50)

for f in listaMA:

if frm1==f[0] and to1==f[1] or to1==f[0] and frm1==f[1]:

bandt=1

if bandt!=1:

listaMa1=[]

listaMa1.append(frm1)

listaMa1.append(to1)

listaMA.append(listaMa1)

#print(listaMA)

agregar\_arista(G, frm1, to1, peso, False)

#agregar\_arista(G, to1, frm1, peso, False)

relacion\_No\_Dirijida(frm1,to1,peso)

pos = nx.layout.planar\_layout(G)

nx.draw\_networkx(G, pos)

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

# print(labels)

nx.draw\_networkx\_edge\_labels(G, pos, edge\_labels=labels)

plt.title("Grafo con NetworkX")

plt.show()

actualizar()

def actualizar():

num = ancho / 2

canvas3.delete("all")

for i in range(len(g.listav)):

canvas3.create\_oval(g.listav[i].x, g.listav[i].y, g.listav[i].x + ancho, g.listav[i].y + ancho, fill="#3498DB", width=0)

canvas3.create\_oval(g.listav[i].x+5, g.listav[i].y+5, g.listav[i].x + ancho-5, g.listav[i].y + ancho-5, fill="#3498DB", activefill="#E67E22", width=0)

for i in range(len(g.listaA)):

if g.listaA[i].x1 >= g.listaA[i].x2 and g.listaA[i].y1 > g.listaA[i].y2:

canvas3.create\_line(g.listaA[i].x1 + num, g.listaA[i].y1, g.listaA[i].x2 + ancho, g.listaA[i].y2 + num,

width=3, fill="#A3E4D7", arrow="last", smooth=True)

if g.listaA[i].x1 > g.listaA[i].x2 and g.listaA[i].y1 < g.listaA[i].y2:

canvas3.create\_line(g.listaA[i].x1 + num, g.listaA[i].y1 + ancho, g.listaA[i].x2 + ancho,

g.listaA[i].y2 + num, width=3, fill="#A3E4D7", arrow="last", smooth=True)

if g.listaA[i].x1 <= g.listaA[i].x2 and g.listaA[i].y1 > g.listaA[i].y2:

canvas3.create\_line(g.listaA[i].x1 + num, g.listaA[i].y1, g.listaA[i].x2, g.listaA[i].y2 + num, width=3,

fill="#A3E4D7", joint=None)

if g.listaA[i].x1 < g.listaA[i].x2 and g.listaA[i].y1 < g.listaA[i].y2:

canvas3.create\_line(g.listaA[i].x1 + num, g.listaA[i].y1 + ancho, g.listaA[i].x2, g.listaA[i].y2 + num,

width=3, fill="#A3E4D7", arrow="last", smooth=True)

for i in range(len(g.listaA)):

if g.listaA[i].x1 >= g.listaA[i].x2 and g.listaA[i].y1 > g.listaA[i].y2:

tx1 = ((g.listaA[i].x1 + num) + (g.listaA[i].x2 + ancho)) / 2

ty1 = (g.listaA[i].y1 + (g.listaA[i].y2 - num)) / 2

canvas3.create\_text([tx1, ty1 + num], text=str(g.listaA[i].distancia) )

if g.listaA[i].x1 > g.listaA[i].x2 and g.listaA[i].y1 < g.listaA[i].y2:

tx1 = ((g.listaA[i].x1 + num) + (g.listaA[i].x2 + ancho)) / 2

ty1 = ((g.listaA[i].y1 - ancho) + (g.listaA[i].y2 + num)) / 2

canvas3.create\_text(tx1, ty1 + ancho, text=str(g.listaA[i].distancia) )

if g.listaA[i].x1 <= g.listaA[i].x2 and g.listaA[i].y1 > g.listaA[i].y2:

tx1 = ((g.listaA[i].x1 + num) + g.listaA[i].x2) / 2

ty1 = (g.listaA[i].y1 + (g.listaA[i].y2 - num)) / 2

canvas3.create\_text(tx1, ty1 + num, text=str(g.listaA[i].distancia) )

if g.listaA[i].x1 < g.listaA[i].x2 and g.listaA[i].y1 < g.listaA[i].y2:

tx1 = (g.listaA[i].x1 + (num + g.listaA[i].x2)) / 2

ty1 = ((g.listaA[i].y1 - ancho) + (g.listaA[i].y2 + num)) / 2

canvas3.create\_text(tx1, ty1 + ancho, text=str(g.listaA[i].distancia) )

for i in range(len(g.listav)):

nombre = str(g.listav[i].nombre)

if len(nombre) > 5:

nombre = nombre[0:4] + ".."

canvas3.create\_text(g.listav[i].x + num, g.listav[i].y + num, text=nombre, fill="#A93226", font="bold")

def actualizar\_color\_cortes():

num = ancho / 2

canvas3.delete("all")

for i in range(len(g.listav)):

canvas3.create\_oval(g.listav[i].x, g.listav[i].y, g.listav[i].x + ancho, g.listav[i].y + ancho, fill="#3498DB", width=0)

canvas3.create\_oval(g.listav[i].x+5, g.listav[i].y+5, g.listav[i].x + ancho-5, g.listav[i].y + ancho-5, fill="#3498DB", activefill="#E67E22", width=0)

for i in range(len(g.listaA)):

if g.listaA[i].color==-1:

if g.listaA[i].x1 >= g.listaA[i].x2 and g.listaA[i].y1 > g.listaA[i].y2:

canvas3.create\_line(g.listaA[i].x1 + num, g.listaA[i].y1, g.listaA[i].x2 + ancho, g.listaA[i].y2 + num,

width=3, fill="#A3E4D7", arrow="last", smooth=True)

else:

if g.listaA[i].x1 >= g.listaA[i].x2 and g.listaA[i].y1 > g.listaA[i].y2:

canvas3.create\_line(g.listaA[i].x1 + num, g.listaA[i].y1, g.listaA[i].x2 + ancho, g.listaA[i].y2 + num,

width=3, fill="#FF003C", arrow="last", smooth=True)

if g.listaA[i].color==-1:

if g.listaA[i].x1 > g.listaA[i].x2 and g.listaA[i].y1 < g.listaA[i].y2:

canvas3.create\_line(g.listaA[i].x1 + num, g.listaA[i].y1 + ancho, g.listaA[i].x2 + ancho,

g.listaA[i].y2 + num, width=3, fill="#A3E4D7", arrow="last", smooth=True)

else:

if g.listaA[i].x1 > g.listaA[i].x2 and g.listaA[i].y1 < g.listaA[i].y2:

canvas3.create\_line(g.listaA[i].x1 + num, g.listaA[i].y1 + ancho, g.listaA[i].x2 + ancho,

g.listaA[i].y2 + num, width=3, fill="#FF003C", arrow="last", smooth=True)

if g.listaA[i].color==-1:

if g.listaA[i].x1 <= g.listaA[i].x2 and g.listaA[i].y1 > g.listaA[i].y2:

canvas3.create\_line(g.listaA[i].x1 + num, g.listaA[i].y1, g.listaA[i].x2, g.listaA[i].y2 + num, width=3,

fill="#A3E4D7", joint=None)

else:

if g.listaA[i].x1 <= g.listaA[i].x2 and g.listaA[i].y1 > g.listaA[i].y2:

canvas3.create\_line(g.listaA[i].x1 + num, g.listaA[i].y1, g.listaA[i].x2, g.listaA[i].y2 + num, width=3,

fill="#FF003C", joint=None)

if g.listaA[i].color==-1:

if g.listaA[i].x1 < g.listaA[i].x2 and g.listaA[i].y1 < g.listaA[i].y2:

canvas3.create\_line(g.listaA[i].x1 + num, g.listaA[i].y1 + ancho, g.listaA[i].x2, g.listaA[i].y2 + num,

width=3, fill="#A3E4D7", arrow="last", smooth=True)

else:

if g.listaA[i].x1 < g.listaA[i].x2 and g.listaA[i].y1 < g.listaA[i].y2:

canvas3.create\_line(g.listaA[i].x1 + num, g.listaA[i].y1 + ancho, g.listaA[i].x2, g.listaA[i].y2 + num,

width=3, fill="#FF003C", arrow="last", smooth=True)

for i in range(len(g.listaA)):

if g.listaA[i].x1 >= g.listaA[i].x2 and g.listaA[i].y1 > g.listaA[i].y2:

tx1 = ((g.listaA[i].x1 + num) + (g.listaA[i].x2 + ancho)) / 2

ty1 = (g.listaA[i].y1 + (g.listaA[i].y2 - num)) / 2

canvas3.create\_text([tx1, ty1 + num], text=str(g.listaA[i].distancia) )

if g.listaA[i].x1 > g.listaA[i].x2 and g.listaA[i].y1 < g.listaA[i].y2:

tx1 = ((g.listaA[i].x1 + num) + (g.listaA[i].x2 + ancho)) / 2

ty1 = ((g.listaA[i].y1 - ancho) + (g.listaA[i].y2 + num)) / 2

canvas3.create\_text(tx1, ty1 + ancho, text=str(g.listaA[i].distancia) )

if g.listaA[i].x1 <= g.listaA[i].x2 and g.listaA[i].y1 > g.listaA[i].y2:

tx1 = ((g.listaA[i].x1 + num) + g.listaA[i].x2) / 2

ty1 = (g.listaA[i].y1 + (g.listaA[i].y2 - num)) / 2

canvas3.create\_text(tx1, ty1 + num, text=str(g.listaA[i].distancia) )

if g.listaA[i].x1 < g.listaA[i].x2 and g.listaA[i].y1 < g.listaA[i].y2:

tx1 = (g.listaA[i].x1 + (num + g.listaA[i].x2)) / 2

ty1 = ((g.listaA[i].y1 - ancho) + (g.listaA[i].y2 + num)) / 2

canvas3.create\_text(tx1, ty1 + ancho, text=str(g.listaA[i].distancia) )

for i in range(len(g.listav)):

nombre = str(g.listav[i].nombre)

if len(nombre) > 5:

nombre = nombre[0:4] + ".."

canvas3.create\_text(g.listav[i].x + num, g.listav[i].y + num, text=nombre, fill="#A93226", font="bold")