UNIVESIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERIA

ESCUELA DE CIENCAS Y SISTEMAS

MATEMATICA PARA COMPUTACION 2

SECCIÓN A

PRIMER SEMESTRE 2023

MANUAL TECNICO

BRANDON EDUARDO PABLO GARCIA 202112092

EVER STEVEN VELASQUEZ AKASAKI 202011875

Guatemala, abril del 2023

CONTENIDO

INTRODUCCION

Este manual describe los pasos necesarios para cualquier persona que tenga ciertas bases de sistemas pueda realizar el código implementado en Python donde se crea un código para un sistema de generador de grafos utilizando POO (Programación Orientada a Objetos) de la misma manera Tkinter y así poder implementarlo de la mejor manera. El siguiente código se explicó de la manera más detalla posible para la mejor compresión de la persona.

OBJETIVOS

- Brindar la información necesaria para poder representar la funcionalidad técnica de la estructura, diseño y definición del aplicativo.

- Describir las herramientas utilizadas para el diseño y desarrollo del prototipo

CONTENIDO TECNICO

1. Parte lógica

El código implementa un algoritmo de recorrido en anchura (BFS) en un grafo no dirigido representado como un diccionario de listas de tuplas. El algoritmo comienza con un nodo de origen ingresado por el usuario y explora todos los nodos adyacentes en orden de cercanía al nodo de origen, antes de avanzar a los nodos adyacentes de los nodos explorados.

Primero, se muestra el grafo completo antes de comenzar el recorrido. Luego, se inicializa una lista de nodos visitados y una cola para almacenar los nodos que aún no han sido visitados pero que son adyacentes a los nodos visitados. Se solicita al usuario que ingrese el nodo de origen.

En el paso 1, se agrega el nodo de origen a la cola. Luego, en el paso 2, el algoritmo comienza a explorar los nodos en la cola mientras haya elementos en ella.

En el paso 3, se extrae el primer elemento de la cola, que se convierte en el nodo actual. En el paso 4, se verifica si el nodo actual ya ha sido visitado. Si no ha sido visitado, se imprime en pantalla en el paso 5. El nodo actual se agrega a la lista de visitados en el paso 6.

En el paso 7, el algoritmo examina todos los nodos adyacentes al nodo actual que aún no han sido visitados, y los agrega a la cola para su posterior exploración.

El proceso se repite hasta que no haya nodos en la cola, lo que significa que todos los nodos alcanzables desde el nodo de origen han sido visitados.

prueba.py | interfaz.py

prueba.py > ...

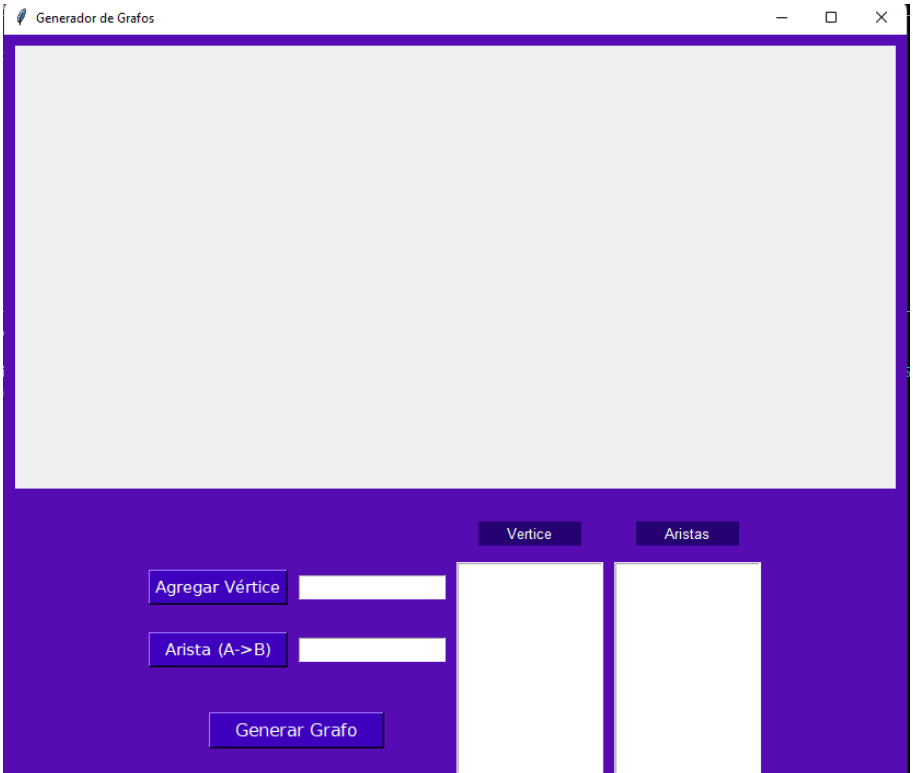
```python
import os

grafo = {'a': [('p',4), ('j',15), ('b',1)],
         'b': [('a',1), ('d',2), ('e',2), ('c',3)],
         'j': [('a',15)],
         'p': [('a', 4)],
         'd': [('b',2), ('g',3)],
         'e': [('b',2), ('g',4), ('f',5), ('c',2)],
         'c': [('b',3), ('e',2), ('f',5), ('i',20)],
         'g': [('d',3), ('e',4), ('f',10), ('h',1)],
         'f': [('g',10), ('e',5), ('c',5)],
         'i': [('c',20)],
         'h': [('g',1)]
        }

print("Muestra el grafo antes del recorrido: \n")
for key, lista in grafo.items():
    print(key)
    print(lista)

print()
os.system("pause")

visitados = []
cola = []

origen = input("Ingresa el nodo origen: ")
print("\nLista de recorrido en anchura\n")
#Paso 1: SE COLOCA EL VÉRTICE ORIGEN EN UNA COLA
cola.append(origen)
#Paso 2: MIENTRAS LA COLA NO ESTE VACÍA
while cola:
    #paso 3: DESENCOLAR UN VÉRTICE, ESTE SERÁ AHORA EL VÉRTICE ACTUAL
    actual = cola.pop(0)

    #paso 4: SI EL VÉRTICE ACTUAL NO HA SIDO VISITADO
    if actual not in visitados:
        #paso 5: PROCESAR (IMPRIMIR) EL VÉRTICE ACTUAL
        print("Vertice actual -> ", actual)
        #paso 6: COLOCAR VÉRTICE ACTUAL EN LA LISTA DE VISITADOS
        visitados.append(actual)
    #paso 7: PARA CADA VÉRTICE QUE EL VÉRTICE ACTUAL TIENE COMO DESTINO,
    #        Y QUE NO HA SIDO VISITADO:
    #        ENCOLAR EL VERTICE
    for key, lista in grafo[actual]:
        if key not in visitados:
            cola.append(key)

print()
os.system("pause")
```

El usuario puede agregar vértices y aristas al gráfico y luego generar el gráfico. El gráfico generado se muestra en el lienzo utilizando la PILbiblioteca. Donde se utilizo graphiz para realizar dicha interfaz.

interfaz.py

interfaz.py > GraphGenerator

```python
import tkinter as tk
from tkinter import messagebox
from grafo import generargrafo

from PIL import ImageTk, Image

#Se debe instalar el modulo pillow con el siguiente comando en consola ----> pip install Pillow

from prueba import recorridoancho


class GraphGenerator:

    FONDO_VENTANA = '#570CB3'
    FONDO_MENU = '#570CB3'

    def __init__(self):
        self.vertices = []
        self.aristas = []
        self.root = tk.Tk()
        self.root.title("Generador de Grafos")
        self.canvas = tk.Canvas(self.root, width=800, height=400)
        self.canvas.pack(side=tk.TOP, padx=10, pady=10)
        self.frame = tk.Frame(self.root)
        self.root['bg'] = self.FONDO_VENTANA
        self.frame['bg'] = self.FONDO_MENU
        self.frame.pack(side=tk.BOTTOM, padx=10, pady=10)

        self.button3 = tk.Button(self.frame, text="Agregar Vértice", command=self.agregar_vertice, font=("DejaVu Sans", 11),fg="white
        self.button3.grid(row=1, column=0, padx=5, pady=5)

        self.entry1 = tk.Entry(self.frame, font=("DejaVu Sans", 11), width=13, )
        self.entry1.grid(row=1, column=1, padx=5, pady=5)

        self.button = tk.Button(self.frame, text="Arista (A->B)", command=self.agregar_arista, font=("DejaVu Sans", 11),fg="white", b
        self.button.grid(row=2, column=0, padx=5, pady=5)
```



2. Interfaz

El código es parte de una aplicación que permite crear y visualizar grafos. Se definen tres métodos que se encargan de agregar vértices y aristas al grafo, y de generar la visualización del grafo.

El método “agregar_vertice” recibe como entrada un vértice a agregar al grafo. Si el campo de entrada está vacío, se muestra un mensaje de error. Si el vértice ya existe en el grafo, también se muestra un mensaje de error. Si el vértice es válido, se agrega a la lista de vértices y se inserta en un cuadro de lista en la interfaz gráfica. Finalmente, se limpia el campo de entrada.

El método “agregar_arista” recibe como entrada una arista a agregar al grafo. Si el campo de entrada está vacío, se muestra un mensaje de error. Si la arista no tiene el formato correcto (X--Y), se muestra un mensaje de error. Si alguno de los vértices de la arista no existe en el grafo, se muestra un mensaje de error. Si los vértices de la arista son iguales, también se muestra un mensaje de error. Si la arista ya existe en el grafo, se muestra un mensaje de error. Si la arista es válida, se agrega a la lista de aristas y se inserta en un cuadro de lista en la interfaz gráfica. Finalmente, se limpia el campo de entrada.

El método “generar_grafo” se encarga de generar la visualización del grafo. Si no hay vértices en el grafo, se muestra un mensaje de error. De lo contrario, se llama a la función recorridoancho que recorre el grafo en anchura para obtener los nodos y las aristas en el orden en que se visitan. Luego se llama a la función generargrafo que crea un archivo de imagen PNG con la visualización del grafo utilizando la librería Graphviz. Este archivo de imagen se carga en la interfaz gráfica en dos widgets que se muestran en pantalla, uno en la parte superior y otro en la parte inferior de la ventana. Se utilizan las funciones ImageTk.PhotoImage y Image.open de la librería Pillow para abrir y redimensionar la imagen PNG y poder visualizarla en la interfaz gráfica.

```
68      def agregar_vertice(self):
69          vertice = self.entry1.get()
70          if not vertice:
71              messagebox.showerror("Error", "Por favor ingrese todos los campos")
72              return
73          if vertice in self.vertices:
74              messagebox.showerror("Error", "No se pueden agregar vertices repetidos")
75              return
76          self.vertices.append(vertice)
77          self.listbox.insert(tk.END, f"{vertice}")
78          self.entry1.delete(0, 'end')
79
80      def agregar_arista(self):
81          arista = self.entry2.get()
82          if not arista:
83              messagebox.showerror("Error", "Por favor ingrese todos los campos")
84              return
85          if '--' not in arista:
86              messagebox.showerror("Error", "La entrada tiene que tener el formato (X--Y)")
87              return
88          lis = arista.split('--')
89          if lis[0] not in self.vertices:
90              messagebox.showerror("Error", f'El vertice "{lis[0]}" no existe')
91              return
92          elif lis[1] not in self.vertices:
93              messagebox.showerror("Error", f'El vertice "{lis[1]}" no existe')
94              return
95          elif lis[0] == lis[1]:
96              messagebox.showerror("Error", 'Los vertices deben ser distintos')
97              return
98          invert = f'{lis[1]}--{lis[0]}'
99          if invert in self.aristas or arista in self.aristas:
100             messagebox.showerror("Error", "No se pueden agregar aristas repetidas")
101             return
102         self.aristas.append(arista)
103         self.listbox1.insert(tk.END, f"{arista}")
104         self.entry2.delete(0, 'end')
```

El siguiente código es una función llamada generargrafo que toma como entrada una lista de vértices, una lista de aristas y una lista de recorrido. Esta función tiene la tarea de generar dos gráficos, uno para mostrar el grafo original y otro para mostrar el grafo con las aristas coloreadas de rojo si están en la lista de recorrido.

Primero, se construye un string llamado texto que representa la definición del grafo en formato dot. Luego se agrega cada vértice al string texto en un formato específico y se define un string texto2 igual a texto en su estado actual.

Después, si hay aristas en la lista de aristas, se agrega cada arista al string texto en un formato específico y se agrega a texto2 de la misma manera. Además, se genera un número aleatorio entre 1 y 5 que se utiliza para definir la longitud de la arista en el gráfico.

grafo.py

grafo.py > generarrecorrido

```python
import subprocess
import random
import os

def generargrafo(vertices,aristas,recorrido):
    if os.path.exists('Grafo.dot'):
        os.remove('Grafo.dot')
    if os.path.exists('Grafo2.dot'):
        os.remove('Grafo2.dot')

    texto = 'graph {\n'
    texto += '\tlayout = neato'
    for vertice in vertices:
        texto += f'\t{vertice}[shape="point", width=0.2, fontsize=25 xlabel="{vertice}"] \n'
    texto2 = texto
    if aristas != []:
        for arista in aristas:
            r = random.uniform(1,5)
            texto += f'\t{arista} [penwidth=3, len={r}]\n'
            aux = arista.split('--')
            rever = f'{aux[1]}--{aux[0]}'
            if arista in recorrido:
                texto2 += f'\t{arista} [penwidth=3, len={r}, color="red"]\n'
            elif rever in recorrido:
                texto2 += f'\t{arista} [penwidth=3, len={r}, color="red"]\n'
            else:
                texto2 += f'\t{arista} [penwidth=3, len={r}]\n'

    texto += '}'
    texto2 += '}'
    Destino = open('Grafo.dot', 'w', encoding='utf-8')
    Destino.write(texto)
    Destino.close()
    cmd_str = "dot -Tpng Grafo.dot -o  Grafo.png"
    subprocess.run(cmd_str, shell=True)
    Destino2 = open('Grafo2.dot', 'w', encoding='utf-8')
    Destino2.write(texto2)
    Destino2.close()
    cmd_str = "dot -Tpng Grafo2.dot -o  Grafo2.png"
```