Informe de Teoría de Grafos

José M. Benavides, Emmanuel M. Sierra

Ingeniería de Sistemas, Universidad de Los Llanos

Villavicencio, Colombia

jmbenavides@unillanos.edu.co

Emartinez.sierra@unillanos.edu.co

Resumen – En esta práctica se desarrolló una aplicación gráfica en Python para representar gráficos dirigidos mediante matrices de adyacencia. Utilizando bibliotecas como tkinter, networkxy, matplotlib, el programa permite al usuario ingresar la cantidad de vértices, definir los pesos entre ellos y visualizar tanto la representación matemática como gráfica del gráfico generado. Esta actividad tuvo como objetivo reforzar la comprensión de la teoría de gráficos, especialmente en lo que respecta a su estructura, representación y visualización computacional.

I. INTRODUCCIÓN

En esta práctica se desarrolló una aplicación gráfica en Python para representar gráficos dirigidos mediante matrices de adyacencia. Utilizando bibliotecas como tkinter, networkxy matplotlib, el programa permite al usuario ingresar la cantidad de vértices, definir los pesos entre ellos y visualizar tanto la representación matemática como gráfica del gráfico generado. Esta actividad tuvo como objetivo reforzar la comprensión de la teoría de gráficos, especialmente en lo que respecta a su estructura, representación y visualización computacional.

II. REFERENTE TEORICO

Un grafo dirigido es una estructura matemática formada por un conjunto de vértices y un conjunto de aristas dirigidas, donde cada arista tiene una dirección que indica un vínculo que va desde un vértice origen hacia un vértice destino. Formalmente, un grafo dirigido se representa como G = (V, E), donde V es el conjunto de vértices y E ⊆ V × V es el conjunto de pares ordenados que representan las aristas.

Una forma común de representar un grafo dirigido en la computación es mediante una matriz de adyacencia, que es una matriz cuadrada de orden "n" en la que la entrada "a_{ij}" representa el peso de la arista que va del vértice "i" al vértice "j". Si no existe dicha arista, el valor suele ser cero.

El estudio de grafos tiene aplicaciones en diversas áreas como ciencias de la computación, redes, análisis de algoritmos, inteligencia artificial y teoría de autómatas. En este contexto, la visualización de un grafo dirigido ayuda a entender mejor sus relaciones, ciclos, caminos y estructuras jerárquicas. Herramientas como networkx permiten modelar grafos de forma flexible y potente, mientras que matplotlib permite representarlos visualmente, facilitando su análisis. Además, tkinter proporciona un entorno gráfico interactivo útil para diseñar interfaces amigables que permitan la manipulación de estos elementos de manera intuitiva.

III. PROCEDIMIENTO

El procedimiento consistió en desarrollar una aplicación interactiva usando la biblioteca tkinter para construir una interfaz gráfica que permite al usuario ingresar la cantidad de vértices de un grafo y posteriormente, llenar una matriz de adyacencia con los pesos de las aristas como se observa en la figura 2

.

A partir de estos datos, el programa, implementado en Python, construye un grafo dirigido utilizando la librería networkx como se ve en la figura 1, extrayendo los valores numéricos ingresados, gestionando posibles errores y omisiones, y representando los conjuntos de vértices y aristas con su notación matemática.

Luego, mediante matplotlib, se genera una visualización gráfica del grafo, mostrando los nodos, las aristas, y los pesos correspondientes, integrando esta visualización en la interfaz mediante FigureCanvasTkAgg. Todo el desarrollo del código se organizó de forma modular dentro de una clase principal, facilitando su comprensión, mantenimiento y reutilización.

```python
def generar_grafo(self):
    matriz = []
    for i, row in enumerate(self.entries):
        fila = []
        for j, entry in enumerate(row):
            val = entry.get()
            try:
                peso = float(val) if val.strip() != "" else 0.0
            except ValueError:
                messagebox.showerror("Error", f"Peso inválido en V{i} a V{j}")
                return
            fila.append(peso)
        matriz.append(fila)
    vertices = [f"V{i}" for i in range(self.n)]
    aristas = []
    G = nx.DiGraph()
    for i in range(self.n):
        G.add_node(vertices[i])
        for j in range(self.n):
            peso = matriz[i][j]
            if peso != 0:
                G.add_edge(vertices[i], vertices[j], weight=peso)
                aristas.append((vertices[i], vertices[j], peso))
    self.mostrar_grafo_completo(G, vertices, aristas)
```

Figura 1. Código para generar el grafo.

```python
def crear_matriz(self):
    for widget in self.root.winfo_children()[3:]:
        widget.destroy()
    self.entries.clear()
    try:
        n = int(self.vertex_entry.get())
        self.n = n
    except ValueError:
        messagebox.showerror("Error", "Ingrese un número entero válido.")
        return
    encabezado = tk.Label(self.root, text="Matriz de adyacencia (pesos de las aristas)", font=("Arial", 12, "bold"))
    encabezado.grid(row=1, column=0, columnspan=n+2, pady=10)
    tk.Label(self.root, text="").grid(row=2, column=0, padx=10)
    for j in range(n):
        tk.Label(self.root, text=f"V{j}", width=5, anchor="center").grid(row=2, column=1+j, padx=5, pady=2)
    for i in range(n):
        tk.Label(self.root, text=f"V{i}", width=5, anchor="w").grid(row=3+i, column=0, padx=10, pady=2)
        row_entries = []
        for j in range(n):
            e = tk.Entry(self.root, width=6, justify="center")
            e.grid(row=3+i, column=1+j, padx=5, pady=5)
            row_entries.append(e)
        self.entries.append(row_entries)
    tk.Button(self.root, text="Generar grafo", command=self.generar_grafo).grid(row=3+n, column=0, columnspan=n+2, pady=10)
```

Figura 2. Código para crear la matriz.

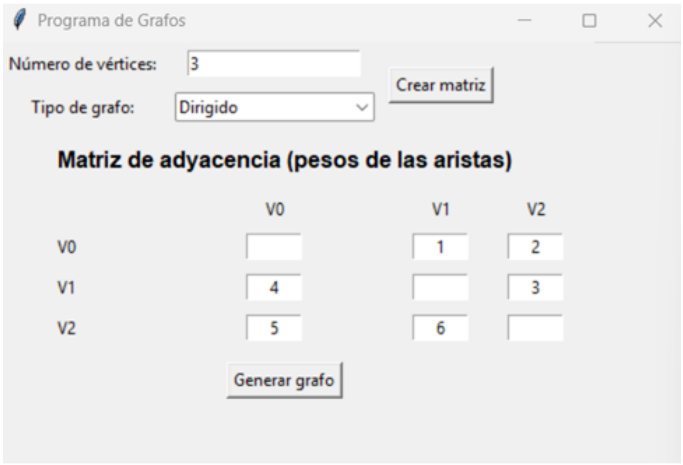

Figura 3. Datos ingresados.

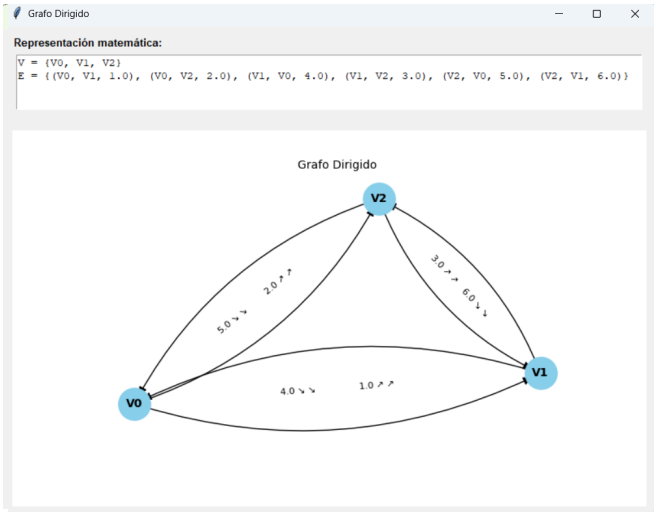

Figura 4. Grafo resultante.

IV.CONCLUSIÓN

En base a los resultados mostrados en la imagen de la figura 4, se puede concluir que el grafo dirigido ha sido correctamente representado tanto de manera matemática como visual, evidenciando claramente los vértices y las aristas con sus respectivos pesos. La representación matemática del grafo define el conjunto de vértices V = {V0, V1, V2} y el conjunto de aristas “E”, donde cada arista está acompañada por su peso correspondiente. Visualmente, el grafo ilustra todas las conexiones posibles entre los nodos, incluidos los ciclos y múltiples conexiones hacia un mismo nodo, lo que indica que es un grafo dirigido denso. Esta representación facilita la comprensión de rutas posibles, análisis de caminos mínimos y algoritmos relacionados como Dijkstra o Bellman-Ford. Además, al mostrar pesos diferenciados en cada arista, se resalta la importancia del costo en la elección de rutas dentro del grafo.

V.REFERENCIAS

[1] Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (3rd ed.). MIT Press.

[2] Wilson, R. J. (2010). *Introduction to Graph Theory* (5th ed.). Pearson.

[3] Gross, J. L., & Yellen, J. (2005). *Graph Theory and Its Applications* (2nd ed.). CRC Press.