Reporte de Proyecto Indivual U2
Equipo 6: Gráfos

Lilian Sayli García Puente* and,
José Andrik Martínez Rodríguez *
*Ingeniería en Tecnologías de la Información
Universidad Politécnica de Victoria

***Resumen*—En este informe, presentamos una herramienta para visualizar y analizar gráficos dirigidos utilizando Python[1] y la biblioteca NetworkX. La herramienta permite a los usuarios ingresar una matriz de adyacencia que representa un gráfico dirigido y visualizarla usando Matplotlib y NetworkX. Además, la herramienta puede identificar componentes fuertemente conectados del gráfico y codificarlos por colores para una fácil visualización. Demostramos la eficacia de la herramienta con dos gráficos de ejemplo, que muestran cómo se puede utilizar para obtener información sobre la estructura de un gráfico dirigido.**

I. Introducción

Un grafo dirigido es un tipo de grafo en el cual las aristas tienen una dirección asociada. Esto significa que cada arista conecta dos nodos, pero tiene una orientación, es decir, va desde un nodo de partida a un nodo de llegada. En un grafo dirigido, las aristas suelen representar relaciones unidireccionales entre objetos. Los grafos dirigidos son útiles en una amplia variedad de aplicaciones, como en la planificación de rutas, el modelado de redes sociales, la optimización de sistemas, la representación de algoritmos, entre otros. El programa que desarrollamos es un algoritmo que crea un grafo conectado a partir de una matriz de adyacencia dada y luego realiza la reducción del grafo utilizando el lenguaje de programación Python[2] y la biblioteca PyQT5[3] para la interfaz gráfica de usuario. El proceso de creación del grafo conectado se realiza mediante la creación de nodos y bordes entre ellos, según la información de la matriz de conveniencia proporcionada. Una vez que el grafo está creado, se lleva a cabo la reducción del grafo, que se realiza eliminando los nodos y bordes redundantes para simplificar el grafo y hacerlo más fácil de entender y procesar. PyQT5[4] se utiliza para proporcionar una interfaz gráfica de usuario para el programa, lo que permite a los usuarios interactuar con el programa de una manera más fácil e intuitiva. La interfaz gráfica de usuario proporciona opciones para cargar la matriz de conveniencia, visualizar el grafo conectado y reducido, y guardar el grafo reducido en un archivo[5].

II. Desarrollo Experimental

El programa que se desarrolló crea un grafo conectado a partir de una matriz de adyacencia y luego lo reduce. El programa está escrito en Python[6] y utiliza la biblioteca PyQt5 para la interfaz gráfica de usuario (GUI)[7]. La GUI consta de un label para recibir la matriz de adyacencia y dos botones: uno para calcular los componentes conectados y otro para mostrar el grafo dirigido a partir de la matriz de adyacencia. Para empezar, el programa leerá la matriz de adyacencia ingresada por el usuario a través del label en la GUI. Luego, el programa utilizará la matriz de adyacencia para crear un grafo no dirigido conectado. Esto se logra mediante el uso de algoritmos de búsqueda, después se realiza un seguimiento de los nodos que han sido visitados y se crean conexiones entre nodos que están conectados[8]. Una vez que se ha creado el grafo conectado, el programa procederá a reducirlo. La reducción del grafo implica eliminar nodos y conexiones redundantes para simplificar su estructura. Finalmente, el programa tiene un botón para mostrar el grafo dirigido resultante. La GUI mostrará el grafo en la ventana separada, utilizando la biblioteca de gráficos de PyQt5.[9]
Para poder correr este programa se debe de llevar a cabo la instalación de las librerías: numpy, networkx, matplotlib y pyqt5.
El constructor de la clase **'MainWindow'** se encarga de crear la ventana principal y todos los widgets que contiene. Se crea una etiqueta Qlabel[10] para indicar al usuario que debe ingresar una matriz de adyacencia en la caja de texto, y dos botones QPushButton para realizar las operaciones deseadas en la matriz. El primer botón 'self.compute_button' se utiliza para calcular los componentes fuertemente conectados de un grafo dirigido creado a partir de la matriz de adyacencia ingresada. El segundo botón 'self.graph_button' se utiliza para mostrar el grafo dirigido creado a partir de la matriz de adyacencia. La función **'compute_strongly_connected_components'** obtiene la matriz de adyacencia del cuadro de texto en el formato de una cadena de texto utilizando el método **'toPlainText()'** y la convierte en una matriz NumPy de enteros utilizando 'np.array()' y la función **'map()'** para aplicar la conversión a cada elemento de la fila. Luego crea un grafo dirigido utilizando NetworkX a partir de la matriz de adyacencia recién creada utilizando **'nx.DiGraph()'**. Para después calcular los componentes fuertemente conectados del grafo usando **'nx.strongly_connected_components()'**. Los componentes fuertemente conectados son subgrafos en los que existe un camino desde cualquier nodo a cualquier otro nodo en el mismo subgrafo. A continuación dibuja el grafo utilizando **'plt.figure()'** para crear una nueva figura, **'nx.circular_layout()'** para asignar una posición circular a los nodos, y **'nx.draw_networkx()'** para dibujar el grafo. Los nodos de cada componente fuertemente conectado se colorea-

ran con un color diferente mediante la lista **‘node_colors’**. Finalmente, se muestra el gráfico utilizando **‘plt.show()’**. La función **‘show_input_matrix_graph()’** se encarga de mostrar el grafo dirigido creado a partir de la matriz de adyacencia ingresada. Al igual que en la función anterior, obtiene la matriz de adyacencia de la caja de texto y la convierte en un objeto ‘numpy.ndarray’. Luego, utiliza la biblioteca NetworkX para crear un objeto ‘DiGraph’ a partir de la matriz de adyacencia. Finalmente, utiliza la biblioteca Matplotlib para dibujar el grafo dirigido. La función **‘show_input_matrix_graph’** es una función que toma la matriz de adyacencia que se encuentra en un cuadro de texto en la interfaz gráfica de usuario, convierte la matriz de adyacencia en una matriz NumPy, donde cada fila representa un nodo y cada columna representa una conexión a otro nodo. Crea un grafo dirigido a partir de la matriz de adyacencia utilizando la clase ‘DiGraph’ de NetworkX. Dibuja el grafo dirigido utilizando la función **‘draw_networkx’** de NetworkX. Utiliza **‘circular_layout’** para definir una distribución circular de los nodos en el gráfico dirigido y después llama a la función **‘axis(.ºff”)’** para ocultar los ejes del gráfico, para finalmente mostrar el gráfico utilizando la función show de Matplotlib.

III. Resultados

En la figura1 se muestra la ventana de la aplicación con un label y dos botones para cargar una matriz o para mostrar los componentes fuertemente conectados del grafo, en la figura 2 se muestra la matriz de adyacencia ingresada en el cuadro de texto de la aplicación. La figura 3 muestra el grafo dirigido generado a partir de la matriz de adyacencia y la figura 4 muestra el mismo grafo, pero con los elementos fuertemente conectados coloreados de manera diferente. La figura 5 muestra la donde se ingresa una segunda matriz de adyacencia, la figura 6 muestra el grafo dirigido generado a partir de la segunda matriz y por último la figura 7 muestra el grafo con los componentes fuertemente conectados coloreados.

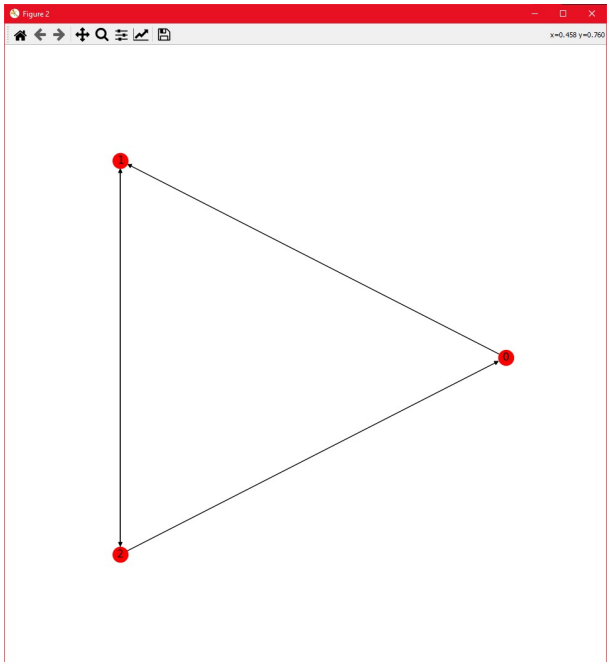

Figura 1: Vista de la ventana con el label y los dos botones.

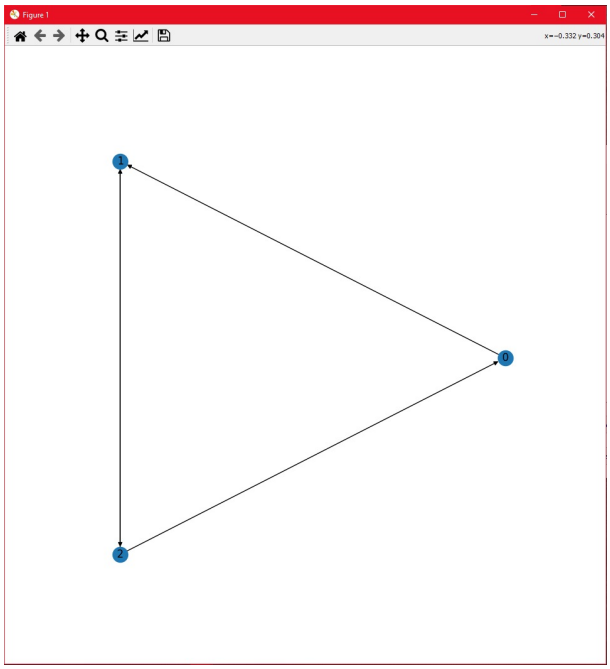

Figura 2: Se ingresa una matriz de adyacencia.

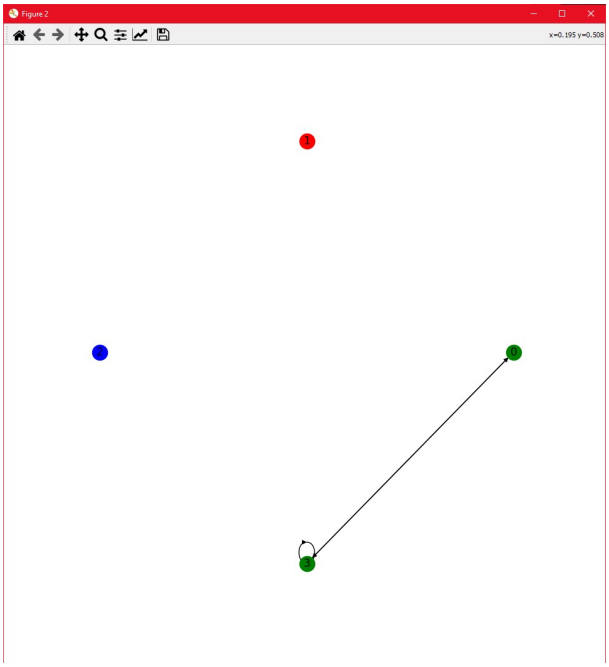

Figura 3: Se muestra el grafo dirigido.

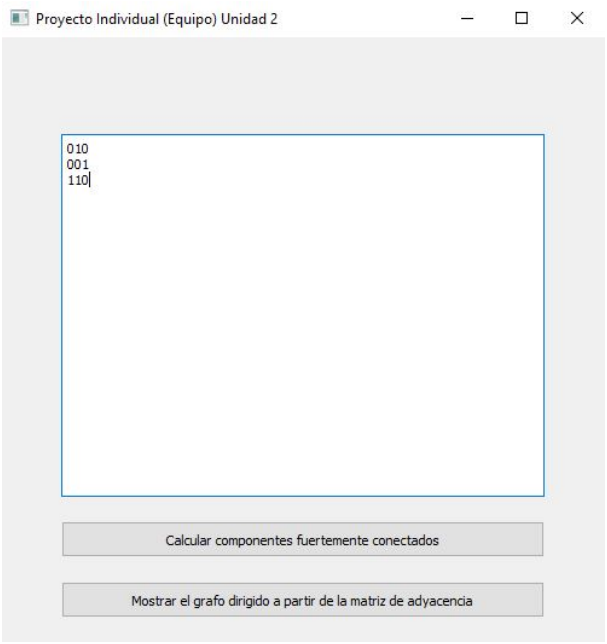

Figura 5: Se ingresa otra matriz

Figura 4: Se muestra el grafo dirigido con los elementos fuertemente conectados.

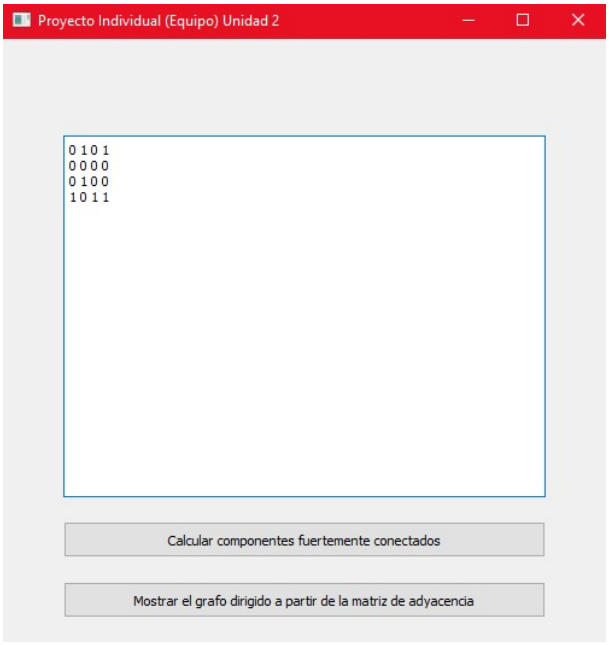

Figura 6: Se muestra el grafo dirigido de la segunda matriz.

IV. Conclusión

En conclusión, la herramienta presentada en este informe brinda una manera simple y efectiva de visualizar y analizar gráficos dirigidos. Con la capacidad de ingresar una matriz de adyacencia y ver el gráfico resultante, los usuarios pueden comprender rápidamente la estructura de un gráfico dirigido. Además, la capacidad de identificar componentes fuertemente conectados permite un análisis más detallado del gráfico. Esta herramienta será útil para una amplia gama de aplicaciones, incluido el análisis de redes, la visualización de datos y el aprendizaje automático.

Uso de referencias [Esta sección no es parte del informe]

Referencias

[1] *Python*. https://www.python.org/. Consultado el 27-03-2023.

[2] Mark Summerfield. "Building Applications with PyQt". En: *The Python Book*. Future Publishing Limited, 2008, págs. 122-131.

[3] Xiaohu Chen et al. "Constructing a Software System of Remote Education Based on Python and QT". En: *Journal of Physics: Conference Series* (2019), pág. 012009.

[4] Seungbum Lee, Jonghwa Park y Youngsik Kim. "Development of a Python QT Application for Image Processing". En: *2020 IEEE International Conference on Big Data and Smart Computing (BigComp)*. 2020, págs. 1-4.

[5] Bess Cukic. "Python QT as an Alternative to the Traditional GUI Toolkits". En: *Journal of Computing Sciences in Colleges* 35.4 (2019), págs. 127-131.

[6] Riverbank Computing Limited. *PyQt5 Reference Guide*. PyQt. 2021. URL: https://doc.qt.io/qtforpython/contents.html.

[7] Mark Summerfield. *Rapid GUI Programming with Python and QT: The Definitive Guide to PyQT Programming*. 1st. Prentice Hall, 2007.

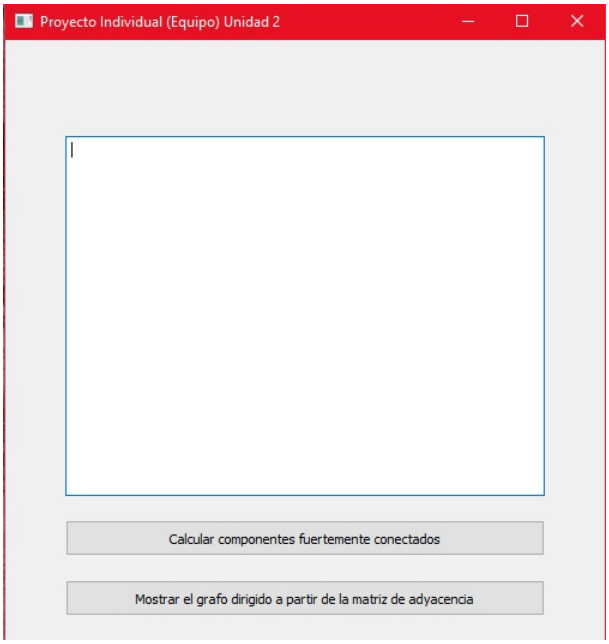

Figura 7: Se muestra el grafo de componentes conectados de la segunda matriz.

[8] Noah Schwartz y Yehuda Koren. "Training Restricted Boltzmann Machines using Approximations to the Partition Function". En: *Neural Computation* 28.8 (2016), págs. 1575-1597.

[9] Anthony Scopatz y Kathryn D. Huff. "Effective Computation in Physics: Field Guide to Research with Python". En: *Computing in Science Engineering* 17.3 (2015), págs. 75-81.

[10] The Qt Company. *Qt Documentation*. The Qt Company Ltd. 2021. URL: https://doc.qt.io/.