Análisis y Representación Computacional de Grafos mediante Python

J. C. Barrera Guevara , D. A. Machado Tovar, J. G. Delgado
Facultad de Ciencias Básicas e Ingeniería, Universidad de los Llanos
Villavicencio, Colombia
jc.bguevara@unillanos.edu.co
damachado@unillanos.edu.co
jg.delgado@unillanos.edu.co

I. INTRODUCCIÓN

La teoría de grafos constituye uno de los pilares conceptuales en la modelación matemática de sistemas complejos. Su versatilidad permite representar y analizar relaciones de dependencia, flujo o conectividad entre entidades, lo cual resulta fundamental en campos como la optimización, la ingeniería de sistemas y las ciencias de la computación.

El objetivo de la presente práctica fue desarrollar una aplicación en Python que permita construir y analizar grafos mediante su matriz de adyacencia, realizar validaciones estructurales y aplicar el algoritmo de Dijkstra para la determinación de rutas de costo mínimo. El trabajo busca integrar los fundamentos teóricos con un enfoque práctico de ingeniería, en el que la visualización del grafo favorece la comprensión y verificación del comportamiento de los algoritmos.

II. REFERENTE TEÓRICO

A. Definición general

Un grafo se define como una estructura compuesta por un conjunto de vértices o nodos y un conjunto de aristas que representan las relaciones entre ellos. Cuando las aristas carecen de orientación se habla de un grafo no dirigido, mientras que en los grafos dirigidos cada arista posee una dirección que determina el sentido de la relación.

Si a cada arista se le asigna un valor numérico denominado peso o costo, el grafo se considera ponderado. Esta característica permite modelar fenómenos cuantitativos, como distancias, tiempos de transporte o capacidad de flujo, haciendo de los grafos una herramienta esencial en la optimización de sistemas reales.

B. Representaciones matemáticas

La matriz de adyacencia es una de las formas más comunes de representar un grafo. En ella, cada elemento a_{ij} indica la existencia o el peso de la conexión entre los nodos v_i y v_j. Esta representación permite implementar fácilmente operaciones de consulta y validación mediante estructuras matriciales.

Otra representación ampliamente utilizada es la lista de adyacencia, en la cual cada vértice almacena un conjunto de sus nodos vecinos. Esta forma resulta más eficiente en términos de memoria cuando el número de conexiones es bajo en relación con el número total de vértices.

C. Propiedades estructurales

Dentro de los conceptos fundamentales se destacan los siguientes:

- **Camino:** secuencia ordenada de vértices conectados por aristas consecutivas.
- **Ciclo:** camino cerrado donde el primer y último vértice coinciden.
- **Conectividad:** propiedad que garantiza la existencia de al menos un camino entre cualquier par de vértices.
- **Grafo completo:** aquel en el que cada par de vértices se encuentra directamente conectado por una arista.

Estas propiedades son determinantes para evaluar la estructura y robustez de un sistema modelado mediante grafos.

D. Algoritmo de Dijkstra

El algoritmo de Dijkstra es un procedimiento determinista que permite encontrar el camino de costo mínimo entre un nodo origen y los demás nodos de un grafo ponderado con pesos positivos. Se basa en el principio de relajación progresiva de las distancias, asignando en cada iteración el valor mínimo alcanzable para cada vértice en función de los pesos acumulados.
El proceso consiste en:

1. Inicializar las distancias, asignando valor cero al nodo origen y valor infinito a los demás.
2. Seleccionar el vértice no visitado con menor distancia provisional.
3. Actualizar las distancias de sus vértices adyacentes si el nuevo camino resulta más corto.
4. Repetir hasta que todos los nodos hayan sido visitados o no existan caminos adicionales.

El algoritmo garantiza la optimalidad del resultado bajo la condición de no negatividad de los pesos y constituye la base de diversos modelos de ruteo y planificación.

III. METODOLOGÍA

El desarrollo de la práctica se llevó a cabo mediante un enfoque computacional modular, empleando el lenguaje Python 3.11 y bibliotecas estándar de manejo gráfico y estructuración de datos.
El proyecto se estructuró en los siguientes componentes principales:

graph_model.py: define las funciones encargadas de validar la matriz de adyacencia, construir las listas de conexiones y ejecutar el algoritmo de Dijkstra utilizando estructuras de prioridad tipo heap para optimizar la búsqueda del nodo con menor distancia.

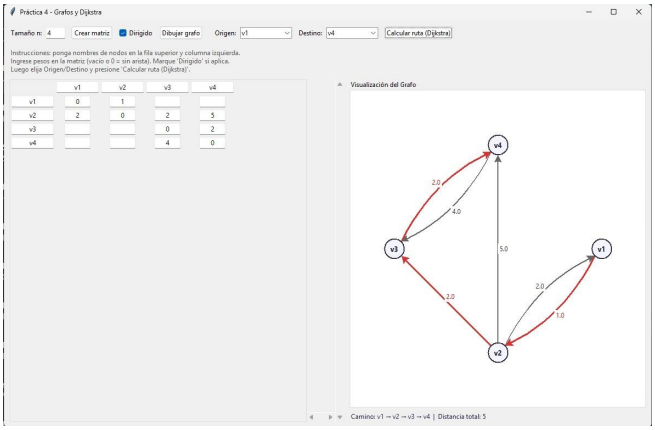
graph_drawer.py: contiene las rutinas matemáticas para el cálculo geométrico de trayectorias y curvas Bézier utilizadas en la representación de aristas y bucles dentro del grafo.

graphgui.py: implementa la interfaz gráfica de usuario mediante la librería Tkinter, permitiendo ingresar los datos, construir la matriz, visualizar el grafo y ejecutar las operaciones de análisis.

main.py: constituye el punto de entrada del sistema y gestiona la interacción general entre los módulos.

A. Etapas experimentales

- **Definición de parámetros del grafo:** ingreso del número de vértices, etiquetas y tipo de grafo (dirigido o no dirigido).
- **Construcción de la matriz de adyacencia:** asignación de pesos que representan la magnitud de las conexiones.
- **Validación estructural:** verificación de coherencia de los datos y control de duplicidades en grafos dirigidos.
- **Visualización:** generación de la representación gráfica del grafo con distribución radial de vértices.
- **Ejecución del algoritmo de Dijkstra:** cálculo del camino mínimo entre los nodos seleccionados y resaltado visual del recorrido óptimo.



· **Figura 1.** Implementación del algoritmo de Dijkstra.

IV. RESULTADOS Y ANÁLISIS

Los resultados obtenidos demostraron la correcta integración entre el modelo matemático y su implementación computacional. El sistema representó de manera precisa tanto grafos dirigidos como no dirigidos, reflejando fielmente las relaciones establecidas en la matriz de adyacencia.

El algoritmo de Dijkstra mostró un comportamiento consistente con la teoría, generando resultados óptimos para distintas configuraciones de grafos ponderados. Los caminos mínimos identificados fueron visualizados en la interfaz, acompañados de la distancia total acumulada, lo que facilitó la interpretación de los resultados.

Durante la fase de validación se observaron las siguientes características destacables:

- Control adecuado de la consistencia en la matriz de entrada.
- Representación geométrica precisa de aristas y bucles.
- Respuesta inmediata ante errores de definición o pesos negativos.
- Capacidad de distinguir casos de no conectividad, informando la ausencia de rutas válidas.

Estas observaciones confirman la fiabilidad del sistema y su correspondencia con los fundamentos teóricos de la teoría de grafos.

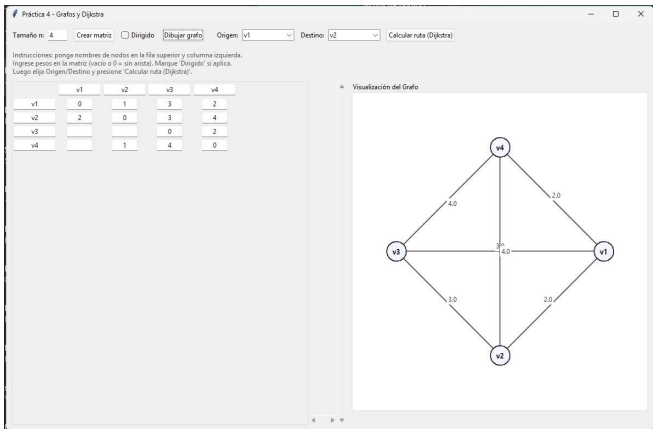


Figura 2. Implementación de un grafo no dirigido.

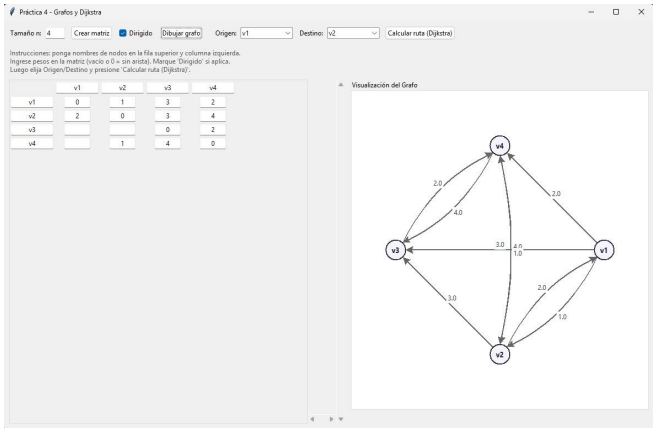


Figura 3. Implementación de un digrafo o grafo dirigido.

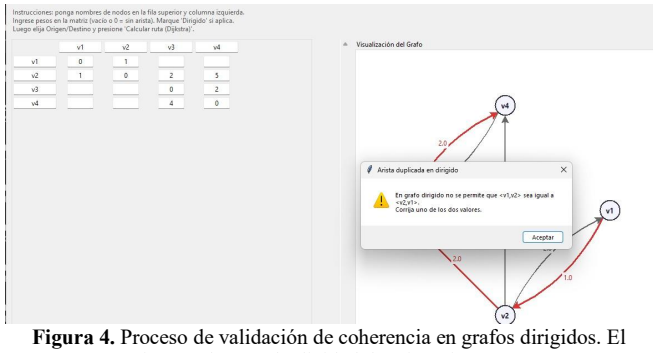


Figura 4. Proceso de validación de coherencia en grafos dirigidos. El sistema detecta duplicidad de aristas inversas.

V.CONCLUSIONES

El desarrollo de la práctica permitió consolidar la comprensión del formalismo teórico de los grafos y su aplicabilidad en la resolución de problemas de optimización. La implementación computacional evidenció cómo los conceptos matemáticos pueden ser traducidos en estructuras de datos y procesos algorítmicos eficientes.

El sistema diseñado cumple con las propiedades de corrección, modularidad y legibilidad del código, y demuestra la pertinencia del uso de Python como herramienta didáctica y de prototipado en investigación de operaciones.

Se concluye que el uso de grafos como modelo abstracto

constituye un instrumento poderoso para analizar sistemas complejos, facilitando la representación de relaciones, la planificación de rutas y la optimización de recursos en contextos reales.

Asimismo, la interfaz desarrollada ofrece un recurso visual valioso para el aprendizaje y la experimentación, permitiendo vincular la teoría con la práctica de forma tangible.

VI. BIBLIOGRAFÍA

[1] R. Diestel, Graph Theory, 5th ed., Springer, 2017.

[2] J. L. Gersting, Mathematical Structures for Computer Science, 7th ed., W.H. Freeman, 2014.

[3] A. A. Hagberg, D. A. Schult, and P. J. Swart, “Exploring network structure, dynamics, and function using NetworkX,” in Proceedings of the 7th Python in Science Conference, 2008, pp. 11–15.

[4] R. Sedgewick and K. Wayne, Algorithms, 4th ed., Addison-Wesley Professional, 2011.