Informe de Algoritmos de la ruta más corta en un Modelo de Redes

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 un programa en Python para calcular la ruta más corta entre dos nodos en un grafo dirigido y ponderado, utilizando los algoritmos de Dijkstra y Bellman-Ford. El sistema permite ingresar la matriz de adyacencia, seleccionar nodos origen y destino, visualizar el grafo con la ruta resaltada y mostrar el costo total del trayecto.

I. INTRODUCCIÓN

El cálculo de rutas más cortas en redes es una herramienta fundamental en optimización de sistemas, especialmente en áreas como telecomunicaciones, logística y análisis de datos. En esta práctica se desarrolló una aplicación interactiva en Python utilizando Tkinter para la interfaz gráfica y NetworkX para la representación visual de grafos. El programa permite al usuario ingresar una matriz de adyacencia, seleccionar nodos de origen y destino, y aplicar los algoritmos de Dijkstra y Bellman-Ford para determinar la ruta más eficiente entre ambos puntos. Esta implementación facilita la comparación entre ambos métodos y su comportamiento ante diferentes configuraciones de grafos, incluyendo aquellos con pesos negativos.

II. REFERENTE TEORICO

La teoría de grafos proporciona el marco matemático para modelar redes como conjuntos de nodos conectados por aristas ponderadas. El problema de la ruta más corta consiste en encontrar el camino de menor costo entre un nodo origen y un nodo destino dentro de un grafo dirigido y ponderado.

El algoritmo de Dijkstra es una técnica de búsqueda greedy que permite encontrar rutas óptimas desde un nodo fuente hacia todos los demás en grafos con pesos positivos. Funciona seleccionando iterativamente el nodo no visitado con la menor distancia acumulada, actualizando las distancias de sus vecinos.

Por otro lado, el algoritmo de Bellman-Ford también encuentra rutas más cortas desde un nodo origen, pero a diferencia de Dijkstra, puede manejar aristas con pesos negativos. Su funcionamiento se basa en relajar todas las aristas del grafo repetidamente, y verificar la existencia de ciclos negativos en una iteración adicional.

Ambos algoritmos son esenciales en optimización de redes y su implementación computacional permite simular y analizar soluciones en diversos escenarios prácticos.

III.PROCEDIMIENTO

En esta práctica se desarrolló una aplicación en Python que permite calcular la ruta más corta entre dos nodos de un grafo dirigido y ponderado, utilizando los algoritmos de Dijkstra y Bellman-Ford. El procedimiento consistió en diseñar una interfaz gráfica con Tkinter para ingresar la matriz de adyacencia para posteriormente cargarla como se ve en la figura 1 y los nodos origen/destino, implementar ambos algoritmos para el cálculo de rutas como se ve en la figura 2, y visualizar los resultados mediante NetworkX y Matplotlib. La aplicación muestra tanto el costo como la secuencia de nodos de la ruta óptima, destacándola gráficamente. Además, se incorporaron validaciones para asegurar el correcto ingreso de datos y el manejo de errores como ciclos negativos o matrices no cuadradas.

```
def load_matrix(self):
    raw = self.text_input.get("1.0", tk.END).strip().split("\n")
    try:
        self.adj_matrix = [
            list(map(lambda x: float('inf') if x.strip() == 'inf' else float(x), row.split(',')))
            for row in raw
        ]
        self.nodes = len(self.adj_matrix)
        for row in self.adj_matrix:
            if len(row) != self.nodes:
                raise ValueError("Matriz no es cuadrada")
        messagebox.showinfo("Éxito", "Matriz cargada correctamente")
    except Exception as e:
        messagebox.showerror("Error", f"Matriz inválida: {e}")
```

Figura 1. Código para cargar la matriz ingresada.

```python
def run_dijkstra(self):
    try:
        origin = int(self.origin_var.get())
        dest = int(self.dest_var.get())
        dist, path = self.dijkstra(origin, dest)
        self.show_result(dist, path, "Dijkstra")
    except Exception as e:
        messagebox.showerror("Error", str(e))

def run_bellman_ford(self):
    try:
        origin = int(self.origin_var.get())
        dest = int(self.dest_var.get())
        dist, path = self.bellman_ford(origin, dest)
        self.show_result(dist, path, "Bellman-Ford")
    except Exception as e:
        messagebox.showerror("Error", str(e))
```

Figura 2. Código para ejecutar Dijkstra y Bellman-Ford.

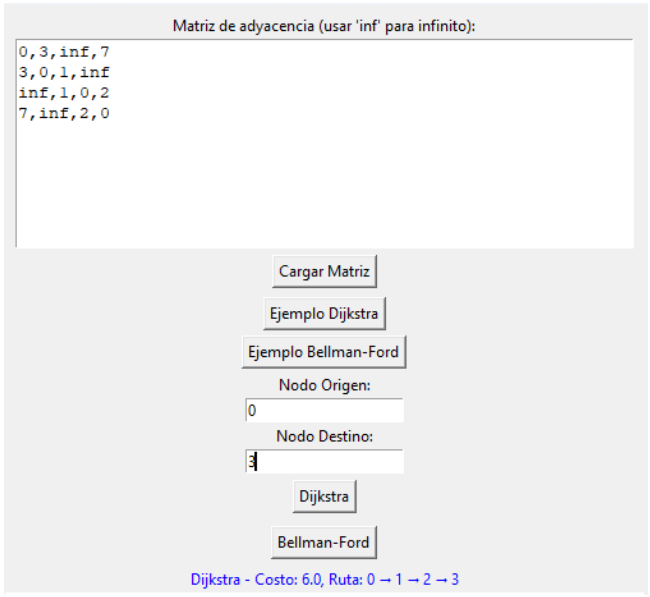

Figura 3. Datos ingresados.

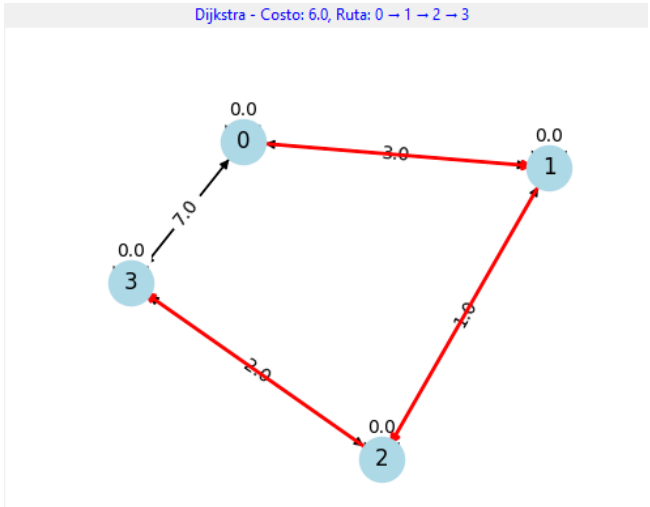

Figura 4. Grafica generada a partir de Dijkstra.

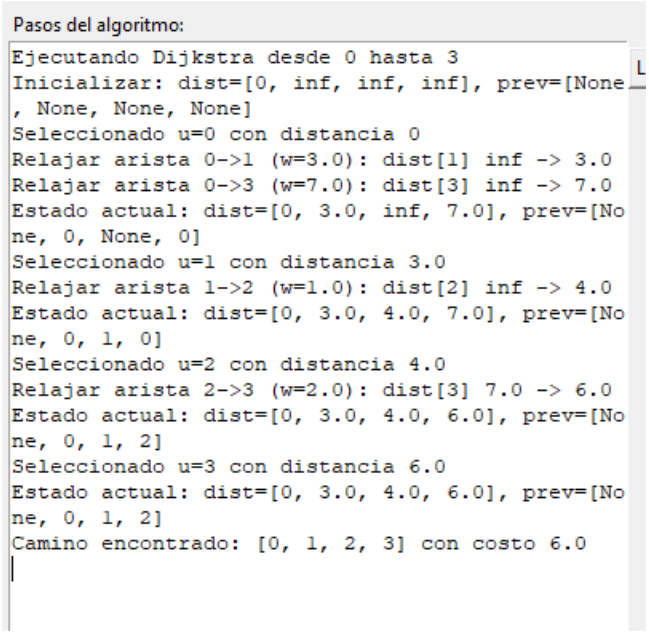

Figura 5. Paso a paso explicado de Dijkstra.

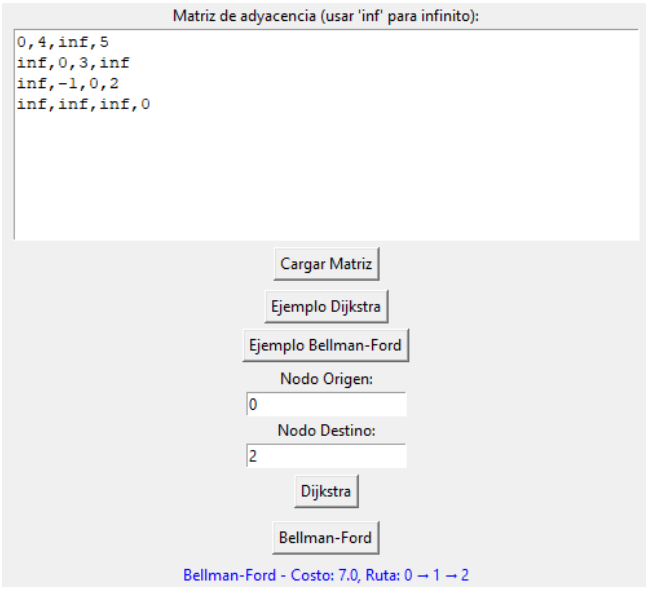

Figura 6. Datos ingresados para Bellman-Ford

.

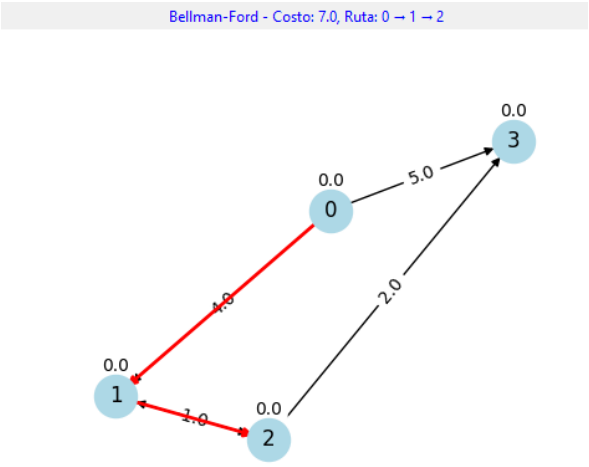

Figura 7. Grafica generada a partir de Bellman-Ford.

Pasos del algoritmo:

```
Ejecutando Bellman-Ford desde 0 hasta 2
Inicializar: dist=[0, inf, inf, inf], prev=[None
, None, None, None]
Iteración 1/3
Relajar arista 0->1 (w=4.0): dist[1] inf -> 4.0
Relajar arista 0->3 (w=5.0): dist[3] inf -> 5.0
Relajar arista 1->2 (w=3.0): dist[2] inf -> 7.0
Estado tras iteración 1: dist=[0, 4.0, 7.0, 5.0]
, prev=[None, 0, 1, 0]
Iteración 2/3
Estado tras iteración 2: dist=[0, 4.0, 7.0, 5.0]
, prev=[None, 0, 1, 0]
Iteración 3/3
Estado tras iteración 3: dist=[0, 4.0, 7.0, 5.0]
, prev=[None, 0, 1, 0]
Camino encontrado: [0, 1, 2] con costo 7.0
```

Figura 8. Paso a paso explicado de Bellman-Ford.

IV.CONCLUSIÓN

Basado en los resultados de la figura 4 y la figura 5 se puede observar que tanto el algoritmo de Dijkstra como el de Bellman-Ford calcularon correctamente la ruta más corta desde el nodo 0 al nodo 4, determinando un costo total de 11 y una ruta óptima que recorre los nodos $0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4$. Esto indica que, para este caso específico, ambos algoritmos produjeron resultados consistentes y eficientes. Además, se evidencia gráficamente que la ruta marcada en rojo corresponde al camino de menor costo dentro del grafo, destacando la utilidad de la visualización para la interpretación de los datos. Si bien en esta instancia no se presentaron pesos negativos, el uso de Bellman-Ford sigue siendo válido y muestra su versatilidad para detectar posibles ciclos negativos en otros escenarios. En resumen, ambos métodos demostraron su funcionalidad y exactitud en la resolución del problema de la ruta más corta en grafos ponderados.

V.REFERENCIAS

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

[2] Dijkstra, E. W. (1959). *A note on two problems in connexion with graphs*.

[3] *Implementación de los algoritmos de Dijkstra y Bellman-Ford para el cálculo de caminos mínimos en grafos*. Universidad Politécnica de Madrid.