Universidad de Costa Rica

Herramientas de Ciencias de Datos II

Algoritmo de Dijkstra

Proyecto individual

José Eduardo López Corella - C24343

31 de mayo de 2024

1. Objetivo General

Desarrollar la búsqueda de rutas óptimas entre dos nodos de un grafo en escenarios complejos mediante la implementación del algoritmo de Dijkstra en Python.

2. Objetivos específicos

- Explicar el funcionamiento del algoritmo de Dijkstra.
- Implementar el algoritmo de Dijkstra en Python.
- Ejemplificar los usos que puede llegar a tener este algoritmo en la vida real.

3. Algoritmo de Dijkstra

El algoritmo de Dijkstra, llamado así por su creador Edsger Dijkstra, es un algoritmo que nos ayuda a encontrar el camino más "*corto*" entre dos vértices de un grafo en el que las aristas entre vértices tienen pesos asignados. Este algoritmo tiene muchas aplicaciones en la vida real, algunos de los usos más conocidos son las aplicaciones de mapas de navegación o en el tema de vuelos.

Estos pesos entre aristas son fundamentales pues en distintas aplicaciones pueden tener distintos significados, como en el caso de mapas en el que los pesos pueden significar el tiempo o distancia mientras que en el caso de aerolíneas con el tema de escalas podría significar qué tan costosa es dicha conexión a la aerolínea con el fin de maximizar ganancias.

3.1. Funcionamiento

1. Inicialización: Se crean dos diccionarios: uno para registrar las distancias provisionales desde el vértice de inicio a todos los demás vértices, y otro para registrar los predecesores de cada vértice en la ruta más corta. Todas las distancias se inicializan a infinito, excepto la distancia al vértice de inicio, que se establece en 0. Los predecesores se inicializan en `None`.
2. Selección del vértice actual: En cada iteración, se selecciona el vértice no explorado con la menor distancia provisional. Esta selección se realiza mediante la función `minimo`, que encuentra la clave (vértice) con el menor valor en el diccionario de distancias provisionales.
3. Actualización de distancias: Para cada vecino del vértice actual, se calcula una distancia tentativa sumando la distancia provisional del vértice actual y el peso de la arista que lo conecta con el vecino. Si la distancia tentativa es menor que la distancia provisional almacenada para ese vecino, se actualiza la distancia provisional y se registra el vértice actual como predecesor del vecino.
4. Eliminación del vértice actual: Después de actualizar las distancias a todos los vecinos, el vértice actual se elimina del conjunto de vértices no explorados.
5. Finalización: El algoritmo termina cuando se han explorado todos los vértices o cuando se ha alcanzado el vértice destino. Para reconstruir la ruta más corta, se sigue el rastro de los predecesores desde el vértice destino hasta el vértice de inicio.

Finalmente, el código que desarrollamos utilizando este algoritmo visualiza el grafo utilizando la librería `networkx` y `matplotlib.pyplot`, además que se importo la librería `random` para la creación de vértices y rutas aleatorias para ejecutar el algoritmo en ejemplos más complejos. Se dibuja el grafo con todos sus vértices y aristas, y se destaca la ruta más corta utilizando color verde para las aristas que forman parte de dicha ruta.

3.2. Otros Ejemplos

- **Redes de Computadoras:** Se utiliza para encontrar la ruta más corta para enviar datos a través de una red de computadoras.
- **Telecomunicaciones y Sistemas de energía:** Ayuda en el diseño eficiente de redes de telecomunicaciones o energía con el fin de minimizar el costo de instalación y operación.
- **Logística y Cadena de Suministro:** Se utiliza para optimizar rutas de entrega y recolección en sistemas de transporte y logística en empresas de reparto como UPS y FedEx, así como Amazon, entre otras.
- **Infraestructura Urbana:** Ayuda en la planificación y desarrollo de infraestructura urbana eficiente, así como la optimización en la creación de nuevas rutas.
- **Sistemas de Transporte Público:** Optimiza las rutas de autobuses, trenes y otros medios de transporte público para mejorar la eficiencia y reducir los tiempos de viaje.
- **Juegos y Simulaciones:** En el desarrollo de videojuegos y simulaciones, se utiliza para encontrar caminos óptimos y mejorar la inteligencia artificial de los personajes optimizando así los recursos utilizados por estos dentro del juego.
- **Agricultura de Precisión:** Se utiliza para planificar rutas eficientes para la maquinaria agrícola, optimizando el uso de recursos y reduciendo el tiempo de operación.
- **Planeación de Evacuaciones:** En casos de emergencia, ayuda a determinar las rutas de evacuación más rápidas y seguras.

4. Código

El codigo completo, desarrollado y documentado se encuentra en:
https://github.com/Corella3435/Proyecto-individual-Herramientas-II

5. Resultados

En el desarrollo de este proyecto se desarrollaron 3 clases, todas utilizan la misma idea del algoritmo de Dijkstra aunque hay ligeras modificaciones.
La clase *DijkstraAlgorithm* es la primera, esta clase al llamarse debe recibir como parámetros los vértices, rutas, inicio y destino como se muestra en el siguiente código.

```python
vertices = ['A', 'B', 'C', 'D', 'E']
rutas = {
    ('A', 'B'): 7,
    ('A', 'C'): 3,
    ('B', 'C'): 1,
    ('B', 'D'): 3,
    ('C', 'D'): 5,
    ('C', 'E'): 6,
    ('E', 'D'): 1,
}
inicio = 'A'
destino = 'D'
# Crear una instancia de la clase DijkstraAlgorithm
dijkstra_basico = DijkstraAlgorithm(vertices, rutas, inicio, destino)
```

El grafico de la Figura 1 es lo obtenido al ejecutar el codigo anterior cuando se crea una instancia de la clase *DijkstraAlgorithm*, con este ejemplo de uso vemos que hallar el camino más corto de forma primitiva no es tan dificil.

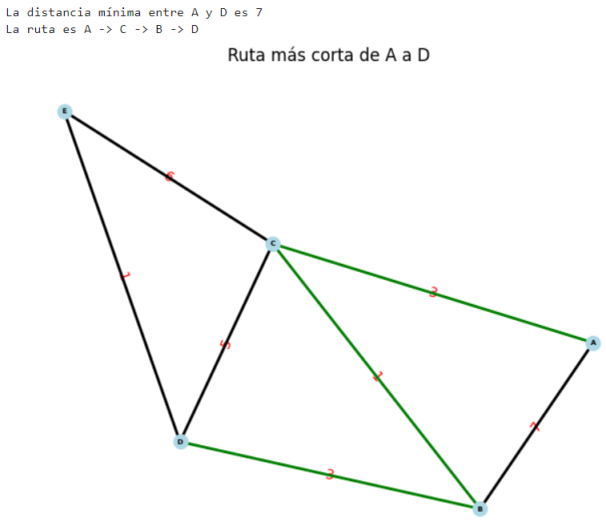

Figura 1: Gráfico del output obtenido del Ejemplo de uso

Sin embargo, en el gráfico de la figura 2 representa lo obtenido en una ejecución random del código para un caso de 40 vértices, con 100 rutas aleatorias cuyas distancias de aristas tienen valores entre 1 y 20, acá ya se evidencia que encontrar esta ruta de manera primitiva sin el uso de algoritmos se empieza a complicar aunque computacionalmente no representa una dificultad considerable.

```
La distancia mínima entre A14 y A7 es 24
La ruta es A14 -> A19 -> A24 -> A11 -> A1 -> A7
```

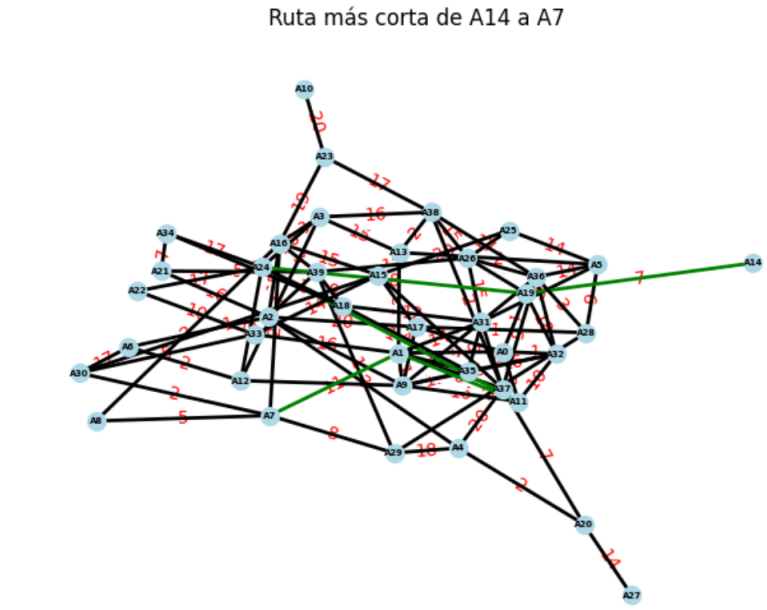

Figura 2: Gráfico del output obtenido con 40 vertices

Es importante mencionar que esta primera clase como se puede ver en el código anterior ya cuando se incrementa el número de rutas no se visualiza de buena manera, por esto es que se programó una versión 2 de esta clase llamada *DijkstraAlgorithmjueguito* con unas ligeras modificaciones que nos genera una imagen del grafo con mejor calidad además de plotear dos figuras, una con la ruta óptima trazada y otra en donde no, con el fin de desarrollar una actividad durante la presentación. A continuación, se muestran dos gráficos en donde se muestra un output de la segunda clase que desarrollamos.

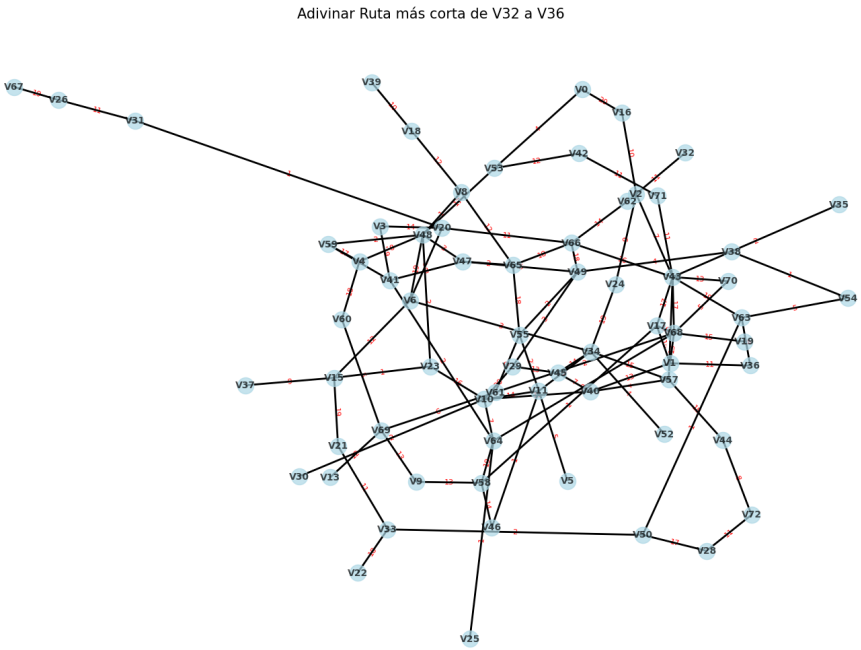

Figura 3: Gráfico del output sin trazar ruta

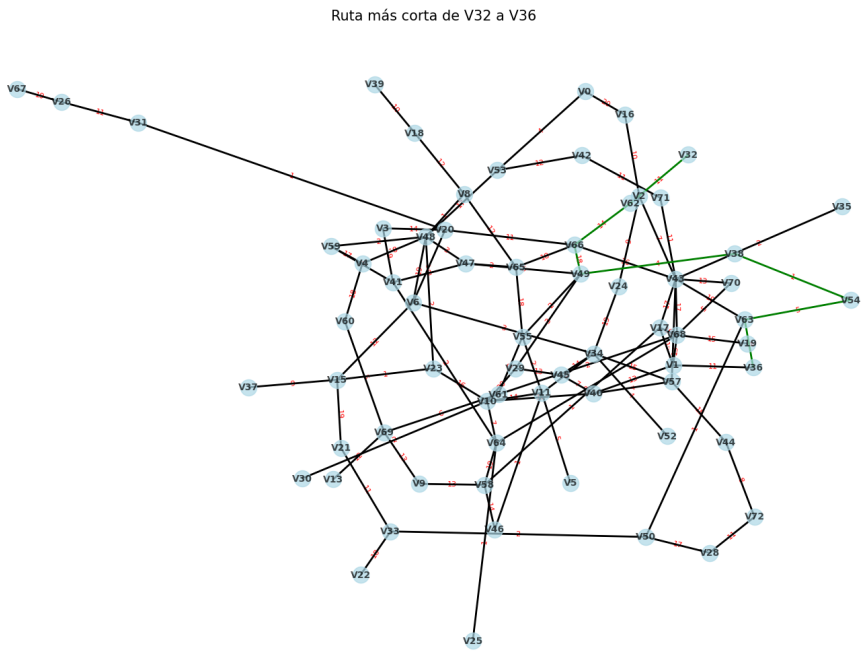

Figura 4: Gráfico del output trazando ruta óptima

Por último, con estas dos clases anteriores como se realiza el plot de la imagen genera que el tiempo de ejecución sea considerablemente mayor y para el funcionamiento de la función realmente no se necesita esto y solo es un complemento a la clase. Por este incremento considerable en los tiempos de ejecución es que se programó una tercera clase que ejecuta el mismo algoritmo sin crear una imagen, es decir, solo nos devuelve la distancia de la ruta más corta y dicha ruta. Al ejecutar este algoritmo sin gráficar se tiene que en un grafo con 2500 vértices y 5000 rutas aleatorias el tiempo promedio de ejecución ronda el segundo.

Por último, se agrego una cuarta clase en la que se realiza una alternativa al método tradicional, la cuál es un método bidireccional que a pesar de que pierde precisión pues puede ser que no encuentre exactamente la ruta más corta aunque si una buena ruta si mejoran los tiempos considerablemente para cantidades de vértices y aristas grandes.

6. ¿Cómo se puede implementar en un caso de la vida real?

El algoritmo de Dijkstra es de suma importancia para sistemas de navegación. La implementación comienza con el modelado del grafo, las intersecciones de calles se modelan como vértices y las calles como aristas con ponderaciones que reflejan la distancia o el tiempo de viaje. Es necesario recopilar datos precisos sobre distancias y tiempos de viaje, que pueden ajustarse según las condiciones de tráfico que se presente en dicho momento. Una vez que tenemos esto, se puede aplicar este algoritmo para encontrar la ruta óptima entre dos puntos, también es importante recalcar que este no es el único algoritmo que se implementa en temas de rutas, sino que también existen otros como lo son el Algoritmo de Bellman-Ford y el Algoritmo A* (A star) y las grandes empresas usan combinaciones de distintos algoritmos junto con modificaciones con el fin de optimizar aún más este proceso, una forma en la que se podría optimizar este proceso en la vida real es ejecutando este algoritmo pero delimitando la región de búsqueda, puesto que si se quiere ir por ejemplo de Alajuela-SJ, no sería óptimo que se incluyerán en los cálculos internos rutas de provincias como Limón o Guanacaste, sino que se podría

trabajar como con un subgrafo que no contenga rutas y vértices irrelevantes para la búsqueda.

7. Conclusiones

En resumen, el algoritmo de Dijkstra optimiza desplazamientos y mejora la eficiencia en diversas aplicaciones del mundo real.

Referencias

Azkardm. (2023). *Dijkstra's algorithm in python: Finding the shortest path.* Descargado de https://medium.com/@azkardm/dijkstras-algorithm-in-python-finding-the-shortest-path-bcb3bcd4a4ea#:~:text=Dijkstra's%20algorithm%20is%20an%20algorithm,Djikstra%20in%201956.

Computerphile. (2017). *Shortest path algorithm - computerphile.* Descargado de https://youtu.be/GazC3A4OQTE?si=q7g4CSGwKPSgLZTE

freeCodeCamp. (2020). *Dijkstra's shortest path algorithm - a visual introduction.* Descargado de https://www.freecodecamp.org/news/dijkstras-shortest-path-algorithm-visual-introduction/

GeeksforGeeks. (2022). *Visualize graphs in python.* Descargado de https://www.geeksforgeeks.org/visualize-graphs-in-python/

Rezaei, A. (2023). *Flights dataset.* Descargado de https://www.kaggle.com/datasets/arezaei81/flights

W3Schools. (2024). *Dijkstra's algorithm.* Descargado de https://www.w3schools.com/dsa/dsa_algo_graphs_dijkstra.php