Analsis y diseño de algoritmos

Unidad II
Hurísticas Voraces
Practica 03:
Implementación y Evaluación del algoritmo de Djkstra
Fernando Ruiz Correa 3M2

Docente:
M. en C. Erika Sanchez-Fermat

Unidad Profesional Interdedisciplinaria de Ingenieria Campus Zacatecas
Instituto Politecnico Nacional
Noviembre 2023

1. Introducción

La resolución del problema de encontrar los caminos mínimos en un grafo ponderado es esencial en diversos contextos, desde redes de transporte hasta sistemas de información. El algoritmo de Dijkstra es una herramienta valiosa en este sentido, y en este práctica, se explorará su funcionamiento y lo se implementará en Python.

2. Desarrollo

2.1. Algoritmo

2.1.1. Definición del problema

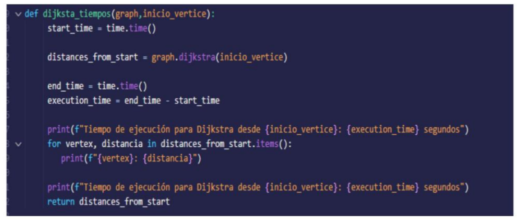
El problema de caminos mínimos en grafos ponderados busca encontrar la ruta más corta desde un nodo de origen a todos los demás nodos en el grafo, considerando los pesos asociados a las aristas. Este problema es central en la teoría de grafos y tiene aplicaciones prácticas en la planificación de rutas en redes de transporte, optimización de rutas de paquetes en redes de comunicación y en la determinación de la distancia más corta entre ubicaciones en mapas.

Figura 1: clase graph.

2.2. Análisis del Algoritmo

En la figura 1 se muestra el codigo generado para realizar el algoritmo de Djikstra, donde se utiliza un constuctor de la clase graph, que permite inicializar el diccionario vertices como un diccionario vacio -Posteriormente se aplica el metodo add Vertex que permite agregar un vertice al grafo y ademas verifica si el vertice existe o no dentro del grafo. Tambien se agrega el metodo add edge que permite agregar una arista con peso al grafo. Ademas se encarga de verificar si existen ambos vertices.

Finalmente existe se crea la función djkstra



```python
def dijksta_tiempos(graph,inicio_vertice):
    start_time = time.time()

    distances_from_start = graph.dijkstra(inicio_vertice)

    end_time = time.time()
    execution_time = end_time - start_time

    print(f"Tiempo de ejecución para Dijkstra desde {inicio_vertice}: {execution_time} segundos")
    for vertex, distancia in distances_from_start.items():
        print(f"{vertex}: {distancia}")

    print(f"Tiempo de ejecución para Dijkstra desde {inicio_vertice}: {execution_time} segundos")
    return distances_from_start
```

En la figura se muestra el cálculo del tiempo utilizando la libreria "time.time()" además, se manda a llamar al método dijkstra para calcular las distancias mínimas desde el vértice de inicio, el resultado es un diccionario donde las claves son los vértices, mientras que los valores las distancias mínimas

Figura 2: Medición del tiempo.

2.2.1. Medición del Tiempo

Se deberá calcular la complejidad del algoritmo con su respectivo análisis manual del mismo. En la figura anterior se muestra la función dedicada a la medición del tiempo, utilizando la libreria "time.time.ªdemás, se manda a llamar al método dijkstra para calcular las distancias mínimas desde el vértice de inicio, el resultado es un diccionario donde las claves son los vértices, mientras que los valores las distancias mínimas

3. Resultados

En la siguientes figuras se muestran los grafos y el camino mas corto obtenido a traves de el cogido realizado.

En la figura se muestra un grafo de 5 nodos con un total de 7 aristas, las cuales son caracterisiticas de un grafo dirigido, se aplica el codigo para obtener el camino mas corto de desde el nodo A hasta D

En la figura se muestra un grafo de 5 nodos con un total de 7 aristas, las cuales son caracterisiticas de un grafo dirigido, se aplica el codigo para obtener el camino mas corto de desde el nodo A hasta D En la figura se muestra un grafo de 5 nodos con un total de 7 aristas, las cuales son caracterisiticas de un grafo dirigido, se aplica el codigo para obtener el camino mas corto de desde el nodo A hasta D

En la figura se muestra un grafo de 5 nodos con un total de 7 aristas, las cuales son caracterisiticas de un grafo dirigido, se aplica el codigo para obtener el camino mas corto de desde el nodo A hasta D

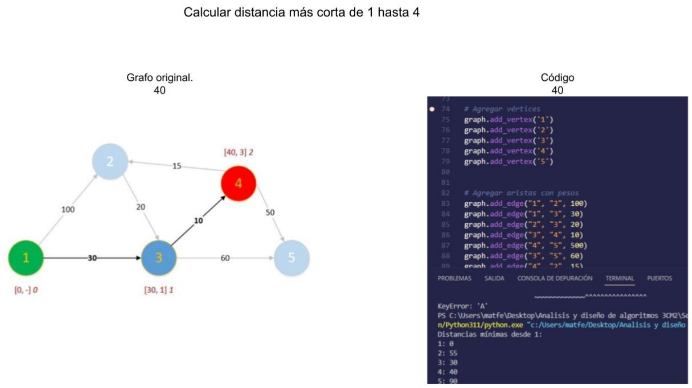


Figura 3: Distancia mas corta del nodo 1 al nodo 4.

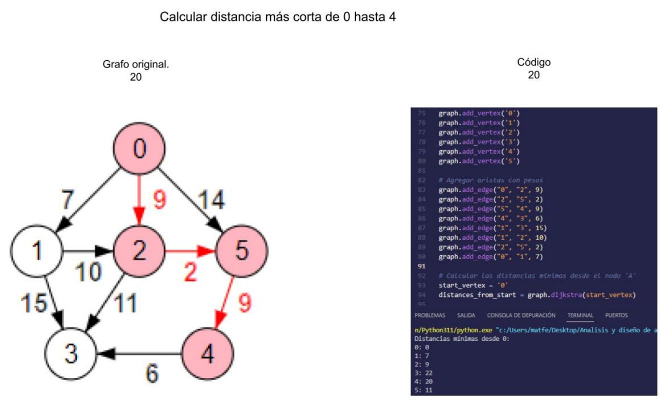


Figura 4: Distancia mas corta del nodo 0 al nodo 4.

4. Conclusiones

- Los algoritmos voraces son algoritmos que toman decisiones basadas en la información disponible en el momento, sin tomar en cuenta lo que suceda mas adelante. En el caso del algoritmo de Dijkstra es considerado voraz, debido a su toma de decisiones locales .ºptimas.en cada paso.
- El enfoque voraz, en el algoritmo de Dijikstra es evidente en su elección de nodos mas cercanos en cada paso, sin cosiderar el panorma del grafo completo, es decir solamente toma en cuenta los nodos vecinos
- Los algoritmos voraces son algoritmos que no siempre toman "la decisión"mas efectiva,

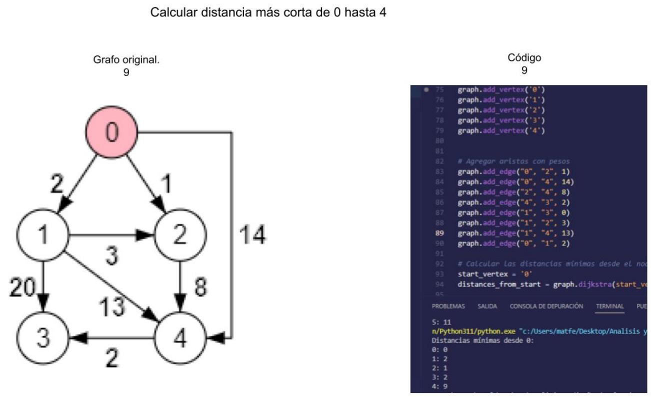


Figura 5: Distancia mas corta del nodo 0 al nodo 4.

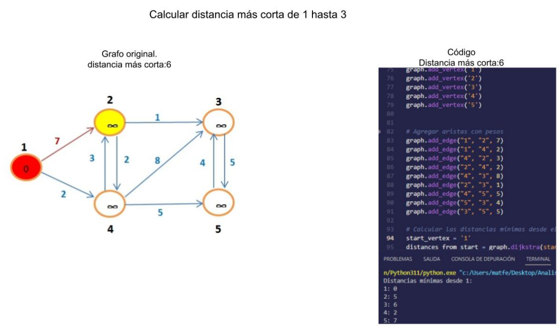


Figura 6: Distancia mas corta del nodo 1 al nodo 3.

regularmente toman la decisión mas corta, por lo que en ocasiones no son lo suficientemente utiles en terminos de complejidad y de tiempo.

5. Referencias

- QuickSort Algorithm Overview — Quick Sort (Article) — Khan Academy. (s. f.). Khan Academy. https://www.khanacademy.org/computing/computer-science/algorithms/quick-sort/a/overview-of-quicksort

- QuickSort Algorithm Overview — Quick Sort (Article) — Khan Academy. (s. f.). Khan Academy. https://www.khanacademy.org/computing/computer-science/algorithms/quick-sort/a/overview-of-quicksort