UNIDAD PROFESIONAL INTERDISCIPLINARIA DE INGENIERÍA CAMPUS ZACATECAS
INSTITUTO POLITÉCNICO NACIONAL

PRÁCTICA 02: IMPLEMENTACIÓN Y EVALUACIÓN DEL ALGORITMO DE DIJKSTRA

2 de diciembre de 2023

Análisis y diseño de algoritmos
Juan F. Alba Avalos 3CM1
Unidad II

0.1. 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.

0.2. Algoritmo

0.2.1. 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.

0.2.2. Descripción del Algoritmo de Dijkstra

El algoritmo de Dijkstra, desarrollado por Edsger Dijkstra en 1956, es un algoritmo voraz que resuelve el problema de caminos mínimos en grafos dirigidos con pesos no negativos. A continuación se detalla su funcionamiento:

0.2.2.1. Inicialización

Se asigna una distancia inicial de 0 al nodo de inicio y de infinito a todos los demás nodos. Se mantiene una cola de prioridad (min-heap) que contiene los nodos no visitados, ordenados por sus distancias actuales.

0.2.2.2. Iteración

Mientras la cola de prioridad no esté vacía, se extrae el nodo con la distancia mínima. Se actualizan las distancias de los nodos adyacentes si se encuentra una ruta más corta a través del nodo actual

0.2.2.3. Finalización

Una vez que todos los nodos han sido visitados o la cola de prioridad está vacía, el algoritmo termina.

0.2.2.4. Resultado

Al finalizar, las distancias mínimas desde el nodo de inicio a todos los demás nodos se han calculado y se pueden utilizar para determinar las rutas más cortas

0.3. Desarrollo

0.3.1. Implementación del algoritmo

El siguiente código implementa el algoritmo de Dijkstra para encontrar las distancias mínimas desde un nodo inicial hacia todos los demás nodos en un grafo ponderado, calculando su distancia minima desde un nodo de inicio y un nodo final, además, calcula el tiempo de ejecucion del ejemplo.

0.3.1.1. Clase grafo

La clase Grafo define la estructura del grafo como un conjunto de vértices con sus respectivas aristas y sus pesos. En este caso, la estructura de datos utilizada es un diccionario donde las claves son los nombres de los vértices y los valores son diccionarios que representan las conexiones salientes y sus respectivos pesos.

0.3.1.2. Dijkstra

La función calcula las distancias mínimas desde un nodo inicial hacia todos los demás nodos en el grafo, se inicializa un diccionario distancias con todas las distancias establecidas como infinito excepto para el nodo inicial, cuya distancia se establece como 0. Se utiliza una cola de prioridad (pq) implementada con heapq para explorar los nodos de manera eficiente.
Mientras haya nodos en la cola de prioridad (pq):
- Se extrae el nodo con la distancia mínima actual.
- Se actualizan las distancias a los vecinos del nodo actual si se encuentra una ruta más corta a través del nodo actual.
- Los vecinos se agregan a la cola de prioridad si se actualizan sus distancias.

0.3.1.3. Resultado del algoritmo

- Se crea una instancia de la clase Grafo y se agregan algunos vértices y aristas.
- Se ejecuta el algoritmo de Dijkstra desde el nodo .A".
- Se calcula el tiempo de ejecución del algoritmo.

Finalmente se imprime el resultado de la distancia minima y el tiempo de ejecucion.

0.3.1.4. Código para el ejemplo del algoritmo

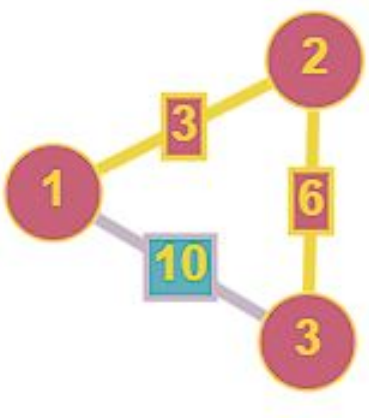
0.4. Resultados

Durante esta parte se modificó el código para que arrojara 5 ejemplos del algoritmo para que diera su tiempo de ejecucion y su distancia minima, a continuación se muestran los datos de los siguientes ejemplos y sus resultados correspondientes:

0.4.1. Ejemplos

0.4.1.1. Ejemplo 1

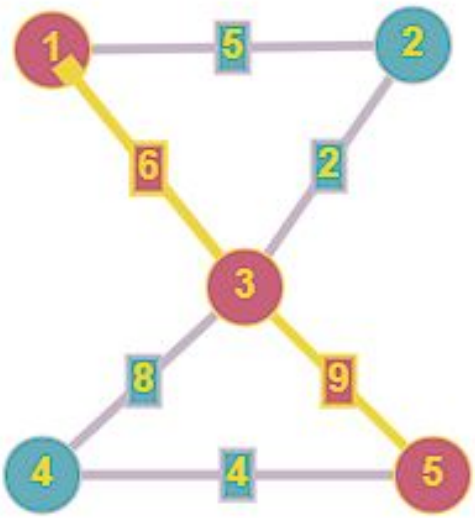
Ejemplo 1: el grafo inicia en el nodo A y termina en el nodo C, el nodo A esta conectado al nodo B con un peso de 3 y al nodo C con un peso de 10, el nodo B esta conectado al nodo C con un peso de 6




```
Ejemplo 1:
Distancia mínima desde el nodo 'A' hasta el nodo 'C': 9
Tiempo de ejecución: 1.0013580322265625e-05 segundos
```

0.4.1.2. Ejemplo 2

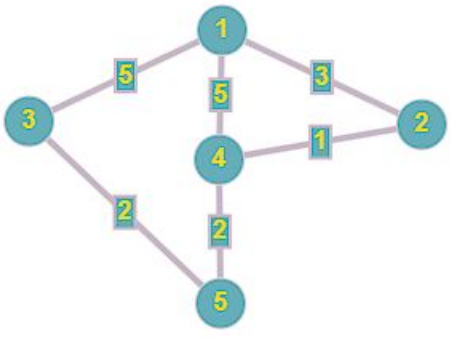
Ejemplo 2: el grafo inicia en el nodo A y termina en el nodo E, el nodo A esta conectado al nodo B con un peso de 5 y al nodo C con un peso de 6, el nodo B está conectado al nodo C con un peso de 2, el nodo C esta conectado al nodo D con un peso de 8 y al nodo E con un peso de 9, el nodo D esta conectado al nodo E con un peso de 4




```
Ejemplo 2:
Distancia mínima desde el nodo 'A' hasta el nodo 'E': 15
Tiempo de ejecución: 8.106231689453125e-06 segundos
```

0.4.1.3. Ejemplo 3

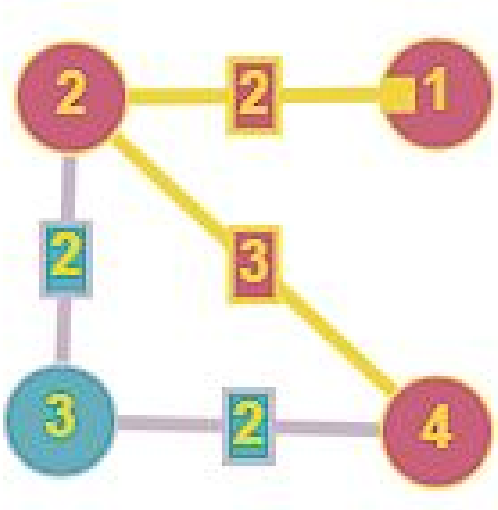
Ejemplo 3: el grafo inicia en el nodo A y termina en el nodo E, el nodo A esta conectado al nodo B con un peso de 3 y al nodo C con un peso de 5, el nodo B está conectado al nodo D con un peso de 1, el nodo C está conectado al nodo E con un peso de 2, el nodo D está conectado al nodo E con un peso de 2




```
Ejemplo 3:
Distancia mínima desde el nodo 'A' hasta el nodo 'E': 6
Tiempo de ejecución: 7.152557373046875e-06 segundos
```

0.4.1.4. Ejemplo 4

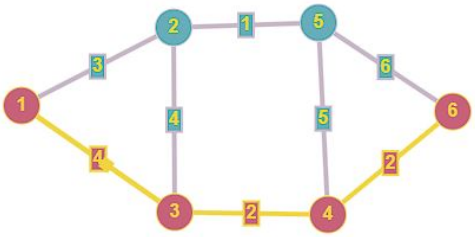
Ejemplo 4: el grafo inicia en el nodo A y termina en el nodo D, el nodo A esta conectado al nodo B con un peso de 2, el nodo B está conectado al nodo D con un peso de 3 y al nodo C con un peso de 2, el nodo C está conectado al nodo D con un peso de 2




```
Ejemplo 4:
Distancia mínima desde el nodo 'A' hasta el nodo 'D': 5
Tiempo de ejecución: 5.4836273193359375e-06 segundos
```

0.4.1.5. Ejemplo 5

Ejemplo 5: el grafo inicia en el nodo A y termina en el nodo F, el nodo A esta conectado al nodo B con un peso de 3 y al nodo C con un peso de 4, el nodo B está conectado al nodo C con un peso de 4 y al nodo E con un peso de 1, el nodo C está conectado al nodo D con un peso de 2, el nodo D está conectado al nodo E con un peso de 5 y al nodo F con un peso de 2, el nodo E esta conectado al nodo F con un peso de 6



```
Ejemplo 5:
Distancia mínima desde el nodo 'A' hasta el nodo 'F': 8
Tiempo de ejecución: 8.344650268554688e-06 segundos
```

0.5. Conclusiones

El algoritmo de Dijkstra es una herramienta bastante buena para encontrar las rutas más cortas en un grafo ponderado, a partir de lo realizado en esta practica se puede destacar sus siguientes virtudes:

- Encontrar la ruta más corta: Dijkstra encuentra la ruta más corta desde un nodo inicial hacia todos los demás nodos en un grafo, considerando pesos en las aristas
- Eficiencia en grafos conexos: Es eficiente en grafos conexos y con pesos no negativos, utilizando una cola de prioridad para explorar nodos de manera óptima
- Implementación versátil: Los códigos proporcionados presentan una implementación clara y estructurada del algoritmo de Dijkstra, con métodos para agregar vértices y aristas en un grafo, así como para calcular las distancias mínimas

En pocas palabras, el algoritmo de Dijkstra es útil para problemas de rutas más cortas y optimización en redes, y los códigos ofrecen una base sólida para comprender y aplicar este algoritmo en la resolución de problemas prácticos.

0.6. Referencias

- https://github.com/Factral/Algoritmo-de-Dijkstra
- https://es.stackoverflow.com/questions/473973/imprimir-el-camino-mas-corto-con-algoritmo-de-dijkstra-repetidas-veces
- https://www.ingenieriaindustrialonline.com/investigacion-de-operaciones/algoritmo-de-dijkstra/