INSTITUTO POLITÉCNICO NACIONAL

Unidad Profesional Interdisciplinaria de Ingeniería
Campus Zacatecas.

MATERIA:

- Analisis y diseño de algoritmos

MAESTRA:

- Erika Sánchez Femat

ALUMNA:

- Naylin Yusseth Martínez Mireles

MÉTODO DIJKSTRA

Índice

1. INTRODUCCIÓN

1.1. ¿DE QUÉ TRATA?

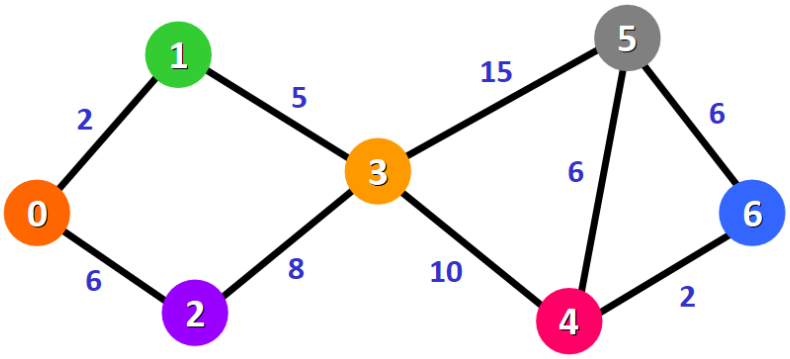
En esta práctica, exploraremos el algoritmo de Dijkstra, una herramienta fundamental en teoría de grafos, este método, lo conocí en el primer semestre de la carrera, en la materia de matemáticas discretas, como lo dije antes es una teoría de grafos para encontrar el camino más corto entre dos puntos en un grafo ponderado.
Analizaremos su funcionamiento paso a paso, además de implementar este método en un código de Python. Es importante decir que el algoritmo de Dijkstra es muy importante, destacando su eficiencia en la búsqueda de rutas óptimas.
Antes de introducir en el desarrollo de la práctica, me gustaría que conociéramos cuál es el origen de este método.
El método de Dijkstra lleva el nombre de su creador, el científico de la computación holandés Edsger Dijkstra. Dijkstra desarrolló este algoritmo en 1956 mientras trabajaba en la solución de problemas relacionados con la optimización de rutas en el contexto de la programación de ordenadores. Desde entonces, el algoritmo de Dijkstra ha demostrado ser una herramienta invaluable en teoría de grafos y ha encontrado aplicaciones en una variedad de campos, desde redes de comunicación hasta planificación de rutas en logística y transporte.
Ya conociendo su origen, podemos adentrarnos en el desarrollo de la práctica.

2. DESARROLLO

2.1. ¿CÓMO FUNCIONA?

En esta parte de la práctica, explicaré como es que implemente el código de Dijkstra en el lenguaje que estamos manejando en la clase de análisis y diseño de algoritmos, este lenguaje es el ya conocido Python.
Lo primero que hice fue, recordar como es que funciona el algoritmo de manera manual, fue un repaso de cómo es que desarrolla, ya que ya tenía un idea de cuál era su objetivo. El algoritmo de Dijkstra (o de caminos mínimos) es un algoritmo voraz para la determinación del camino más corto de un grafo.
Este algoritmo usa los valores de los arcos para encontrar el camino que minimiza el valor total entre el nodo de origen y los demás nodos del grafo.
Este valor depende de lo que representa el valor de los arcos en el grafo.
La idea subyacente en este algoritmo consiste en ir explorando todos los caminos más cortos que parten del vértice origen y que llevan a todos los demás vértices; cuando se obtiene el camino más corto desde el vértice origen, al resto de vértices que componen el grafo, el algoritmo se detiene.
Este algoritmo utiliza dos conjuntos de nodos, SyC.S contiene los nodos ya seleccionados y cuya distancia mínima al origen ya se conoce y C contiene los demás nodos, $C = N/S$, aquellos cuya distancia mínima al origen no se conoce todavía.
Al inicio del algoritmo S sólo contiene el nodo origen y cuando finaliza el algoritmo contiene todos los nodos del grafo y además se conocen las longitudes mínimas desde el origen a cada uno de ellos.
La función de selección elegirá en cada paso el nodo de C cuya distancia al origen sea mínima.

Ejemplo 1.



2.2. CODIGO PYTHON

```
import heapq
from collections import defaultdict
import time

class GrafoPonderado:
        def __init__(self):
        self.vertices = set()
        self.aristas = defaultdict(list)

        def agregar_vertice(self, vertice):
    # Agrega un vertice al conjunto de vertices.
            self.vertices.add(vertice)

    def agregar_arista(self, desde_vertice,
       hacia_vertice, peso):
    # Agrega una arista bidireccional con su
       respectivo peso.
        self.vertices.add(desde_vertice)
        self.vertices.add(hacia_vertice)
```

```
        self.aristas[desde_vertice].append((
            hacia_vertice, peso))
        self.aristas[hacia_vertice].append((
            desde_vertice, peso))

    def dijkstra(self, vertice_inicial):
        # Implementacion del algoritmo de Dijkstra
            para encontrar caminos mas cortos.
        distancias = {vertice: float('infinity') for
             vertice in self.vertices}
        distancias[vertice_inicial] = 0
        padres = {vertice: None for vertice in self.
            vertices}
        cola_prioridad = [(0, vertice_inicial)]
        tiempo_inicio = time.time()

        while cola_prioridad:
                distancia_actual, vertice_actual =
                    heapq.heappop(cola_prioridad)

            if distancia_actual > distancias[
                vertice_actual]:
                continue
            for vecino, peso_arista in self.aristas[
                vertice_actual]:
                distancia = distancia_actual +
                    peso_arista
                if distancia < distancias[vecino]:
                    distancias[vecino] = distancia
                    padres[vecino] = vertice_actual
                    heapq.heappush(cola_prioridad, (
                        distancia, vecino))

        tiempo_fin = time.time()
```

```
        tiempo_transcurrido = tiempo_fin -
            tiempo_inicio
        caminos_minimos = {vertice: self.
            construir_camino(vertice_inicial, vertice,
             padres) for vertice in self.vertices}
        return distancias, caminos_minimos,
            tiempo_transcurrido

    def construir_camino(self, vertice_inicial,
        vertice_final, padres):
        # Construye el camino mas corto desde el
            vertice inicial hasta el vertice final.
        camino = []
        vertice_actual = vertice_final
        while vertice_actual is not None:
                camino.insert(0, vertice_actual)
                vertice_actual = padres[
                    vertice_actual]
        return camino

 # Crear instancias de GrafoPonderado para cada caso
grafo1 = GrafoPonderado()
grafo2 = GrafoPonderado()
grafo3 = GrafoPonderado()
grafo4 = GrafoPonderado()
grafo5 = GrafoPonderado()

# Caso 1
grafo1.agregar_vertice("A")
grafo1.agregar_vertice("B")
grafo1.agregar_vertice("C")
grafo1.agregar_arista("A", "B", 5)
grafo1.agregar_arista("A", "C", 3)
grafo1.agregar_arista("B", "C", 2)
```

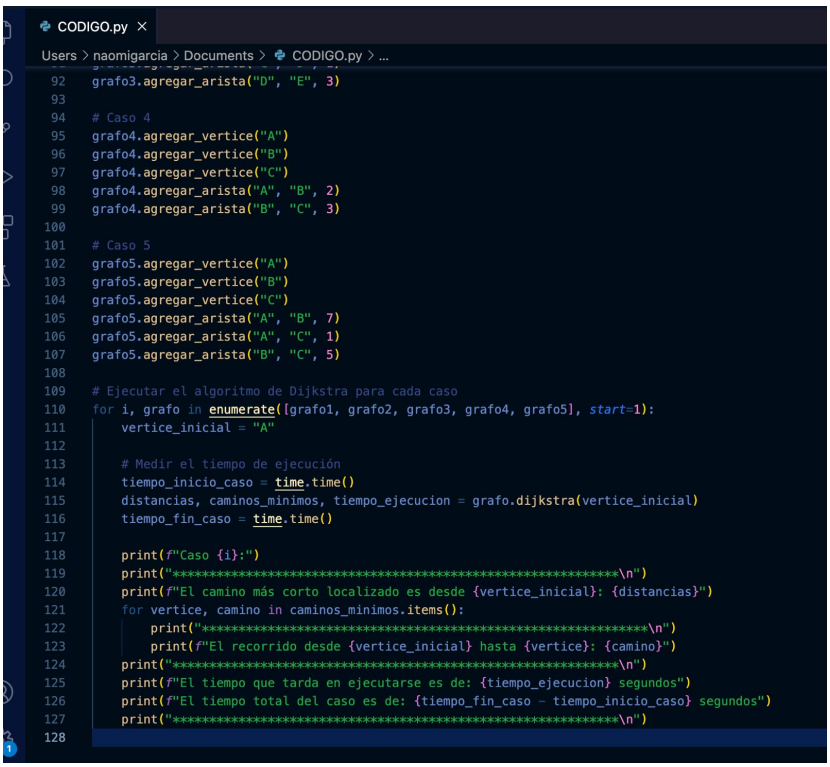
```
# Caso 2
grafo2.agregar_vertice("A")
grafo2.agregar_vertice("B")
grafo2.agregar_vertice("C")
grafo2.agregar_vertice("D")
grafo2.agregar_arista("A", "B", 2)
grafo2.agregar_arista("A", "C", 4)
grafo2.agregar_arista("B", "C", 1)
grafo2.agregar_arista("B", "D", 7)
grafo2.agregar_arista("C", "D", 3)

# Caso 3
grafo3.agregar_vertice("A")
grafo3.agregar_vertice("B")
grafo3.agregar_vertice("C")
grafo3.agregar_vertice("D")
grafo3.agregar_vertice("E")
grafo3.agregar_arista("A", "B", 4)
grafo3.agregar_arista("A", "C", 2)
grafo3.agregar_arista("B", "C", 5)
grafo3.agregar_arista("B", "D", 10)
grafo3.agregar_arista("C", "D", 1)
grafo3.agregar_arista("D", "E", 3)

# Caso 4
grafo4.agregar_vertice("A")
grafo4.agregar_vertice("B")
grafo4.agregar_vertice("C")
grafo4.agregar_arista("A", "B", 2)
grafo4.agregar_arista("B", "C", 3)

# Caso 5
grafo5.agregar_vertice("A")
```

```
grafo5.agregar_vertice("B")
grafo5.agregar_vertice("C")
grafo5.agregar_arista("A", "B", 7)
grafo5.agregar_arista("A", "C", 1)
grafo5.agregar_arista("B", "C", 5)

# Ejecutar el algoritmo de Dijkstra para cada caso
for i, grafo in enumerate([grafo1, grafo2, grafo3,
    grafo4, grafo5], start=1):
    vertice_inicial = "A"

    # Medir el tiempo de ejecucion
    tiempo_inicio_caso = time.time()
    distancias, caminos_minimos, tiempo_ejecucion =
        grafo.dijkstra(vertice_inicial)
    tiempo_fin_caso = time.time()

    print(f"Caso {i}:")
    print("********************\n")
    print(f"El camino mas corto localizado es desde
        {vertice_inicial}: {distancias}")
    for vertice, camino in caminos_minimos.items():
        print("**********************\n")
        print(f"El recorrido desde {vertice_inicial}
            hasta {vertice}: {camino}")
    print("********************\n")
    print(f"El tiempo que tarda en ejecutarse es de:
        {tiempo_ejecucion} segundos")
    print(f"El tiempo total del caso es de: {
        tiempo_fin_caso - tiempo_inicio_caso} segundos
        ")
    print("**********************\n")
```

2.3. IMPLEMENTACION DE LOS 5 CASOS

Para explicar mejor el codigo, tome capturas de como era que se veia ya ejecutado.
Acontinuacion presento la captura de los 5 casos

Captura del codigo en visual studio code.



```
CODIGO.py
Users > naomigarcia > Documents > CODIGO.py > ...
grafo3.agregar_arista("D", "E", 3)

# Caso 4
grafo4.agregar_vertice("A")
grafo4.agregar_vertice("B")
grafo4.agregar_vertice("C")
grafo4.agregar_arista("A", "B", 2)
grafo4.agregar_arista("B", "C", 3)

# Caso 5
grafo5.agregar_vertice("A")
grafo5.agregar_vertice("B")
grafo5.agregar_vertice("C")
grafo5.agregar_arista("A", "B", 7)
grafo5.agregar_arista("A", "C", 1)
grafo5.agregar_arista("B", "C", 5)

# Ejecutar el algoritmo de Dijkstra para cada caso
for i, grafo in enumerate([grafo1, grafo2, grafo3, grafo4, grafo5], start=1):
    vertice_inicial = "A"

    # Medir el tiempo de ejecución
    tiempo_inicio_caso = time.time()
    distancias, caminos_minimos, tiempo_ejecucion = grafo.dijkstra(vertice_inicial)
    tiempo_fin_caso = time.time()

    print(f"Caso {i}:")
    print("**********************************************************\n")
    print(f"El camino más corto localizado es desde {vertice_inicial}: {distancias}")
    for vertice, camino in caminos_minimos.items():
        print("**********************************************************\n")
        print(f"El recorrido desde {vertice_inicial} hasta {vertice}: {camino}")
    print("**********************************************************\n")
    print(f"El tiempo que tarda en ejecutarse es de: {tiempo_ejecucion} segundos")
    print(f"El tiempo total del caso es de: {tiempo_fin_caso - tiempo_inicio_caso} segundos")
    print("**********************************************************\n")
```

Caso 1

```
Caso 1:
**********************************************************

El camino más corto localizado es desde A: {'B': 5, 'C': 3, 'A': 0}
**********************************************************

El recorrido desde A hasta B: ['A', 'B']
**********************************************************

El recorrido desde A hasta C: ['A', 'C']
**********************************************************

El recorrido desde A hasta A: ['A']
**********************************************************

El tiempo que tarda en ejecutarse es de: 4.0531158447265625e-06 segundos
El tiempo total del caso es de: 1.4066696166992188e-05 segundos
**********************************************************
```

Caso 2

```
Caso 2:
************************************************************

El camino más corto localizado es desde A: {'D': 6, 'B': 2, 'C': 3, 'A': 0}
************************************************************

El recorrido desde A hasta D: ['A', 'B', 'C', 'D']
************************************************************

El recorrido desde A hasta B: ['A', 'B']
************************************************************

El recorrido desde A hasta C: ['A', 'B', 'C']
************************************************************

El recorrido desde A hasta A: ['A']
************************************************************

El tiempo que tarda en ejecutarse es de: 5.245208740234375e-06 segundos
El tiempo total del caso es de: 1.0967254638671875e-05 segundos
************************************************************
```

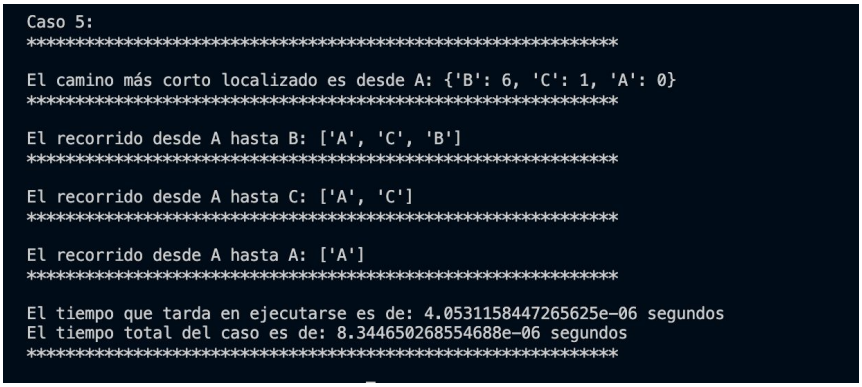
Caso 3

```
Caso 3:
************************************************************

El camino más corto localizado es desde A: {'E': 6, 'D': 3, 'B': 4, 'C': 2, 'A': 0}
************************************************************

El recorrido desde A hasta E: ['A', 'C', 'D', 'E']
************************************************************

El recorrido desde A hasta D: ['A', 'C', 'D']
************************************************************

El recorrido desde A hasta B: ['A', 'B']
************************************************************

El recorrido desde A hasta C: ['A', 'C']
************************************************************

El recorrido desde A hasta A: ['A']
************************************************************

El tiempo que tarda en ejecutarse es de: 3.814697265625e-06 segundos
El tiempo total del caso es de: 1.0013580322265625e-05 segundos
************************************************************
```

Caso 4

```
Caso 4:
************************************************************

El camino más corto localizado es desde A: {'B': 2, 'C': 5, 'A': 0}
************************************************************

El recorrido desde A hasta B: ['A', 'B']
************************************************************

El recorrido desde A hasta C: ['A', 'B', 'C']
************************************************************

El recorrido desde A hasta A: ['A']
************************************************************

El tiempo que tarda en ejecutarse es de: 5.0067901611328125e-06 segundos
El tiempo total del caso es de: 1.4781951904296875e-05 segundos
************************************************************
```

Caso 5



2.4. COMPLEJIDAD

Cuando el ordenamiento rápido siempre tiene las particiones más desbalanceadas posibles, entonces la llamada original tarda un tiempo çn"para alguna constante c, la llamada recursiva sobre n-1 elementos tarda un tiempo c(n-1), la llamada recursiva sobre n-2 elementos tarda un tiempo c(n-2), y así sucesivamente.

Mostrar que el tiempo de ejecución del caso promedio también es $=O(n \log_2 n)$ requiere unas matemáticas bastante complicadas, así que no lo vamos a hacer. Pero podemos obtener un poco de intuición al ver un par de otros casos para entender por qué podría ser $O(n \log_2 n)$.
Una vez que tengamos $O(n \log_2 n)$, la cota de $O(n \log_2 n)$ se sigue porque el tiempo de ejecución del caso promedio no puede ser mejor que el tiempo de ejecución del mejor caso.
Primero, imaginemos que no siempre obtenemos particiones igualmente balanceadas, pero que siempre obtenemos una razón de 3 a 1
Es decir, imagina que cada vez que hacemos una partición, un lado obtiene $\frac{3n}{4}$ elementos y el otro lado obtiene $\frac{n}{4}$ para mantener limpias las matemáticas, no vamos a preocuparnos por el pivote.
Entonces, el árbol de los tamaños de los sub-problemas y los tiempos para hacer las particiones se vería así:

Para analizar el caso promedio, sea $T(n)$ que denota el número de pasos necesarios para llevar a cabo el quick sort en el caso promedio para n elementos. Se supondrá que después de la operación de división la lista se ha dividido en dos sublistas. La primera de ellas contiene s elementos y la segunda contiene $(n-s)$ elementos. El valor de s varía desde 1 hasta n y es necesario tomar en consideración todos los casos posibles a fin de obtener el desempeño del caso promedio.
Para obtener $T(n)$ es posible aplicar la siguiente fórmula:

$T(n) = Promedio(T(s) + T(n-s)) + cnconl <= s <= n$

donde cn denota el número de operaciones necesario para efectuar la primera operación de división. Cada elemento es analizado antes de dividir en dos sublistas la lista original. Al resolver matemáticamente, se obtiene una eficiencia $O(nlogn)$.

3. CONCLUSIÓN

En conclusión para esta práctica, solo me queda resaltar que el algoritmo visto aquí, es un algoritmo de suma importancia en la teoría de grafos.
A través de la implementación y análisis del código del algoritmo de Dijkstra, eh logrado una comprensión más grande de su funcionamiento. Este algoritmo demuestra su eficacia al encontrar de manera eficaz los caminos más cortos en grafos ponderados.
La experiencia práctica con el código me ayudó a fortalecer mi entendimiento lógico computacional además de reforzar el teórico del que ya tenía una idea ligera de cómo se implementaba.
Esto también me ayudó a resaltar la importancia y versatilidad del algoritmo de Dijkstra en la resolución eficiente de problemas del mundo real.

4. REFERENCIAS

http://atlas.uned.es/algoritmos/voraces/dijkstra.html
https://www.freecodecamp.org/espanol/news/algoritmo-de-la-ruta-mas-cort
#:~:text=El%20algoritmo%20de%20Dijkstra%20encuentra,los%20dems%
20nodos%20del%20grafo.
https://www.ingenieriaindustrialonline.com/investigacion-de-operaciones/
algoritmo-de-dijkstra/amp/
https://www.codingame.com/playgrounds/7656/los-caminos-mas-cortos-con-e
el-algoritmo-de-dijkstra
https://www.udb.edu.sv/udb_files/recursos_guias/informatica-ingenieria/
programacion-iv/2019/ii/guia-10.pdf