**Unidad Profesional Interdisciplinaria de Ingeniería**
**Campus Zacatecas**
**Instituto Politécnico Nacional**

Unidad 2 Estrategias de diseño de algoritmos

# Práctica 02: Implementación y Evaluación del Algoritmo de Dijkstra.

Alumno:
Kevin Marin Ramirez Reyna

Octubre 2023

# 1. Introduccion

Los grafos son una composición interesante de conjuntos de objetos que denominamos nodos. En ellos se almacena diferentes tipos de elementos o datos que podemos utilizar para procesar o conocer con fines específicos. Adicionalmente estos nodos, suelen estar unidos o conectados a otros nodos a través de elementos que denominamos aristas.

Un grafo, por si no conoces el término, se refiere a un tipo de representación simbólica o gráfica de distintos tipos de elementos que conforman un conjunto o sistema.
Entonces, supongamos que de un vértice de origen tienes muchos caminos, sin saber cuál es el más corto. Este algoritmo te va a permitir recorrer todos los demás vértices y, cuando obtiene el más corto, lo ejecuta y se detiene.

Algoritmo de Dijkstra. También llamado algoritmo de caminos mínimos, es un algoritmo para la determinación del camino más corto dado un vértice origen al resto de vértices en un grafo con pesos en cada arista. Su nombre se refiere a Edsger Dijkstra, quien lo describió por primera vez en 1959.

Este algoritmo es utilizado para determinar el camino más corto para ejecutar desde un vértice origen hasta el resto de los vértices ubicados en un grafo con pesos en cada arista.

# 2. Desarrollo

**Se incluyeron librerias para tiempo y memoria**
import time
import heapq

**Creamos la clase**
class Graph:

**Constructor self Vertex = Vertice / Edge = Arista**
def init (self):
self.vertex = set ()
self.edge =

**Funcion para agregar vertices**
def addVertex(self, value):
self.vertex.add(value)
self.edge[value] = []

**Funcion para agregar aristas from = desde / until = hasta / weight = peso**
def addEdge(self, From, until, weight):
self.edge[From].append((until, weight))

**Si el grafo es no dirigido**
self.edge[until].append((From, weight))
def dijkstra(self, begin):

**Inicializando el diccionario**
distance = v: float('infinity') for v in self. vertex
distance[begin] = 0

**Se usa un heap para tomar el vertice con la distancia minima mejor eficiente**
reir priority = [(0, begin)]

**Inciamos el tiempo**
begin time = time.time()

while reir priority:
distance present, vertex present = heapq.heappop(reir priority

**Si la distancia que esta actual es mayor a la que esta en el diccionario, se cambia al siguiente vertice**
if distance present > distance[vertex present]: continue

**Recorrer los vertices vecinos del vertice actual**
for neighbor, weight in self.edge[vertex present]:
distance new = distance present + weight

**Se cambian las distancisa minimas si hay un camino mas corto**
if distance new < distance[neighbor]:
distance[neighbor] = distance new
heapq.heappush(reir priority, (distance new, neighbor))
**Detenemos y calculamos el tiempo**
end time = time.time()
time carry = end time - begin time
**Devuelve el tiempo y el diccionario**
return distance, time carry

if name == ' main ':

**Creamos el grafo que queremos usar**
Graph example = Graph()

**Agregamos los vertices**
Graph example.addVertex('A')
Graph example.addVertex('B')
Graph example.addVertex('C')
Graph example.addVertex('D')
Graph example.addVertex('E')

**Agregamos las aristas**
Graph example.addEdge('A', 'B', 1)
Graph example.addEdge('A', 'C', 2)
Graph example.addEdge('B', 'C', 5)
Graph example.addEdge('B', 'D', 6)
Graph example.addEdge('D', 'E', 3)
Graph example.addEdge('A', 'E', 7)

**Tomamos nuestro vertice de inicio**
begin vertex = 'A'

**Aplicamos dijkstra al grafo**
distance minimum, time carry= Graph example.dijkstra(begin vertex)

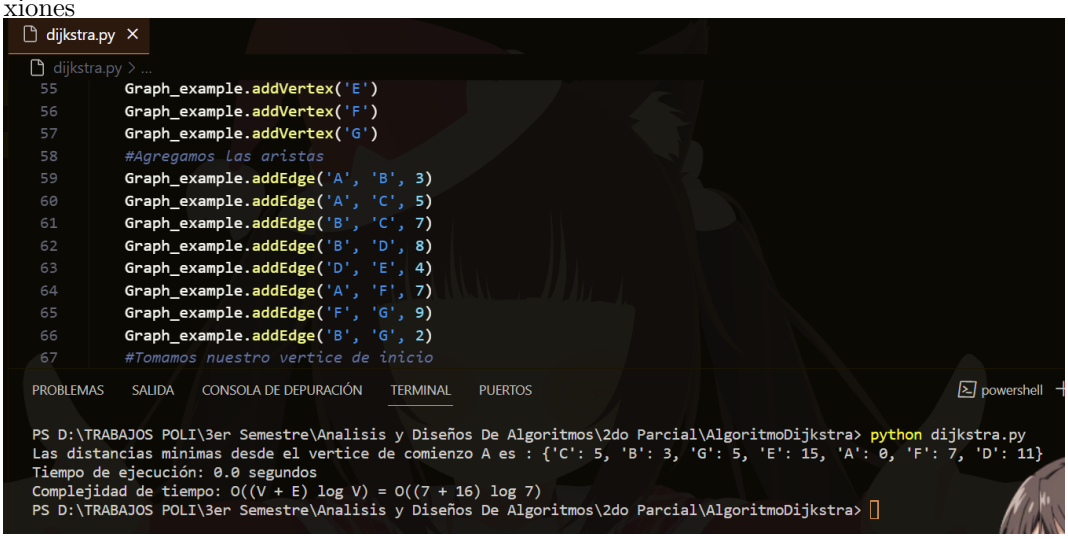
**Lo imprimimos**
print(f'Las distancias minimas desde el vertice de comienzo begin vertex es : distance minimum')
print(f'Tiempo de ejecución: time carry segundos')

**Calculamos la complejidad**
v = len(Graph example.vertex)
e = sum(len(edges) for edges in Graph example.edge.values())

**El algoritmo tiene una complejidad en el tiempo donde V es el número de vértices y E es el número de aristas**
print(f'Complejidad de tiempo: O((V + E) log V) = O((v + e) log v)')

# 3. Resultados


En esta imagen podemos ver el primer ejemplo y los resultados que este arroja en la imagen se puede apreciar que los nodos del grafo que ocupan son A,B,C,D,E,F,asimismo comosus valores en las conexiones




En esta imagen podemos ver el segundo ejemplo y los resultados que este arroja en los variados caminos que tienes,y con ello ver que tiene nodos A,B,C,D,E,F,G, y valores de cada una de las aristas

```
dijkstra.py > ...
48      #Creamos el grafo que queremos usar
49      Graph_example = Graph()
50      #Agregamos los vertices
51      Graph_example.addVertex('A')
52      Graph_example.addVertex('B')
53      Graph_example.addVertex('C')
54      Graph_example.addVertex('D')
55      #Agregamos las aristas
56      Graph_example.addEdge('A', 'B', 9)
57      Graph_example.addEdge('A', 'C', 3)
58      Graph_example.addEdge('B', 'D', 7)
59      Graph_example.addEdge('C', 'D', 4)
60      #Tomamos nuestro vertice de inicio
61      begin_vertex = 'A'
62      #Aplicamos dijkstra al grafo

PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL  PUERTOS   powershell

PS D:\TRABAJOS POLI\3er Semestre\Analisis y Diseños De Algoritmos\2do Parcial\AlgoritmoDijkstra> python dijkstra.py
Las distancias minimas desde el vertice de comienzo A es : {'C': 3, 'B': 9, 'D': 7, 'A': 0}
Tiempo de ejecución: 0.0 segundos
Complejidad de tiempo: O((V + E) log V) = O((4 + 8) log 4)
PS D:\TRABAJOS POLI\3er Semestre\Analisis y Diseños De Algoritmos\2do Parcial\AlgoritmoDijkstra>
```

En esta imagen podemos ver el tercer ejemplo y los resultados que nos dan cada una los posibles buenos caminos que hay ,y con ello ver que tiene nodos A,B,C,D y valores de cada una de las aristas

```
dijkstra.py
dijkstra.py > ...
62      Graph_example.addEdge('A', 'B', 1)
63      Graph_example.addEdge('A', 'C', 3)
64      Graph_example.addEdge('B', 'C', 5)
65      Graph_example.addEdge('B', 'D', 2)
66      Graph_example.addEdge('D', 'E', 7)
67      Graph_example.addEdge('A', 'F', 4)
68      Graph_example.addEdge('F', 'G', 8)
69      Graph_example.addEdge('B', 'G', 9)
70      Graph_example.addEdge('C', 'I', 4)
71      Graph_example.addEdge('E', 'H', 2)
72      Graph_example.addEdge('D', 'I', 2)
73      #Tomamos nuestro vertice de inicio

PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL  PUERTOS   powershell

PS D:\TRABAJOS POLI\3er Semestre\Analisis y Diseños De Algoritmos\2do Parcial\AlgoritmoDijkstra> python dijkstra.py
Las distancias minimas desde el vertice de comienzo A es : {'C': 3, 'J': inf, 'F': 4, 'I': 5, 'B': 1, 'D': 3, 'G': 10, 'E': 10, 'A': 0, 'H'
: 12}
Tiempo de ejecución: 0.0 segundos
Complejidad de tiempo: O((V + E) log V) = O((10 + 22) log 10)
PS D:\TRABAJOS POLI\3er Semestre\Analisis y Diseños De Algoritmos\2do Parcial\AlgoritmoDijkstra>
```

En esta imagen podemos ver el cuarto ejemplo y los resultados que nos realizara los mejores caminos que hay ,y con ello ver que tiene nodos A,B,C,D,E,F,G,H,I y valores de cada una de las aristas

```
dijkstra.py
dijkstra.py
51      Graph_example.addVertex('A')
52      Graph_example.addVertex('B')
53      Graph_example.addVertex('C')
54      Graph_example.addVertex('D')
55      #Agregamos las aristas
56      Graph_example.addEdge('A', 'B', 2)
57      Graph_example.addEdge('A', 'C', 3)
58      Graph_example.addEdge('A', 'D', 9)
59      Graph_example.addEdge('B', 'C', 8)
60      Graph_example.addEdge('B', 'D', 3)
61      Graph_example.addEdge('C', 'D', 6)
62      #Tomamos nuestro vertice de inicio
63      begin_vertex = 'A'

PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL  PUERTOS   powershell

PS D:\TRABAJOS POLI\3er Semestre\Analisis y Diseños De Algoritmos\2do Parcial\AlgoritmoDijkstra> python dijkstra.py
Las distancias minimas desde el vertice de comienzo A es : {'C': 3, 'B': 2, 'A': 0, 'D': 5}
Tiempo de ejecución: 0.0 segundos
Complejidad de tiempo: O((V + E) log V) = O((4 + 12) log 4)
```

En esta imagen podemos ver el quinto ejemplo y los resultados que nos dan cada una los eficientes que hay ,y con ello ver que tiene nodos A,B,C,D y valores de cada una de las aristas

# 4. Conclusión

Podemos llegar a que el método de Dijkstra y la teoría de grafos se entrelazan para ofrecer una solución eficaz a problemas fundamentales de optimización de rutas. Los grafos, al proporcionar una representación visual de conexiones entre nodos, permiten modelar sistemas complejos de manera intuitiva. El método de Dijkstra, por otro lado, se destaca como un algoritmo versátil y eficiente para encontrar el camino más corto en grafos ponderados, lo que lo convierte en una herramienta esencial en campos tan diversos como la logística, las redes de comunicación y la planificación de rutas.

La capacidad del método de Dijkstra para adaptarse a diferentes tipos de grafos, ya sean dirigidos o no dirigidos, lo hace ampliamente aplicable en una variedad de contextos. Su enfoque de búsqueda voraz, que selecciona y actualiza de manera inteligente las distancias mínimas, garantiza la eficiencia en la identificación de las rutas óptimas. Esta combinación de teoría de grafos y el método de Dijkstra se traduce en una herramienta valiosa para abordar desafíos del mundo real, contribuyendo a la optimización de procesos y la toma de decisiones informada en una amplia gama de disciplinas.

# 5. Referencias

Scribbr. (2022, 28 noviembre). Formato APA con el generador APA de Scribbr
https://www.scribbr.es/citar/generador/apa/
Ivaldi, T. (2023, 27 noviembre). Lo que necesitas saber sobre el algoritmo de Dijkstra. TFG Online.
https://tfgonline.es/algoritmo-de-dijkstra/