

**Unidad Profesional Interdisciplinaria de Ingeniería
Campus Zacatecas**

INGENIERÍA EN SISTEMAS 3CM2

**IMPLEMENTACIÓN Y EVALUACIÓN DEL
ALGORITMO DE DIJKSTRA.**

Proyecto Final del parcial

Autor:

Enid Aimee Perez Robles

Junio 2023

0.1 Introducción

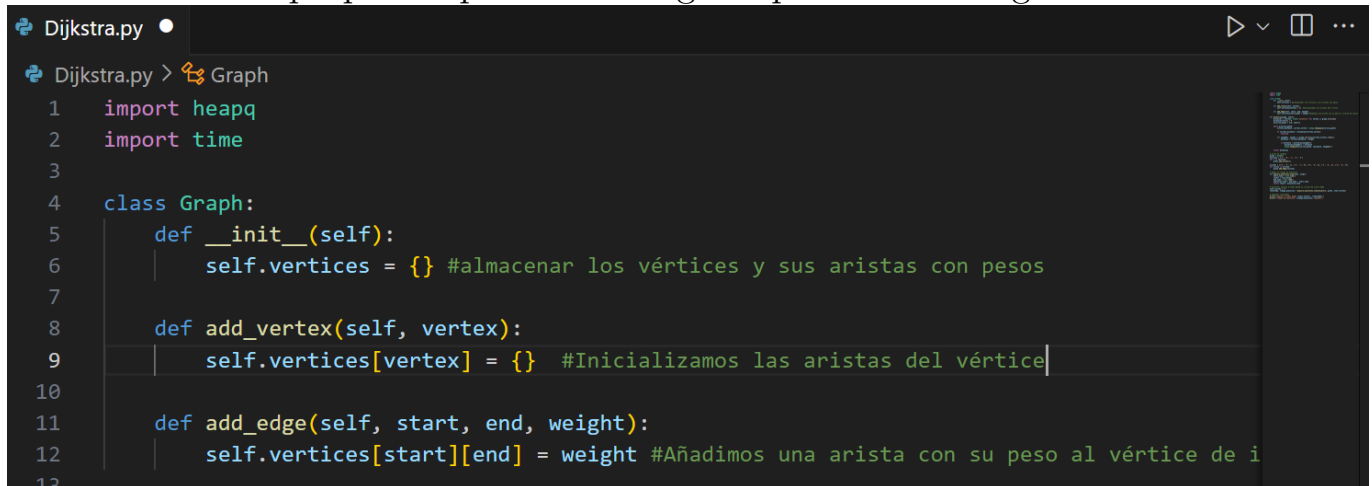
La resolución del problema de encontrar los caminos mínimos en un grafo ponderado es esencial en diversos contextos, desde redes de transporte hasta sistemas de información. El algoritmo de Dijkstra es una herramienta muy valiosa en este sentido, y en esta práctica, se explorará su funcionamiento y lo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

El algoritmo de Dijkstra, concebido por el científico Edsger Dijkstra en 1956, destaca como una herramienta fundamental en la teoría de grafos y la optimización de rutas. Su aplicación principal radica en encontrar el camino más corto entre dos nodos en un grafo ponderado, donde cada arista tiene asignado un peso que representa la distancia entre los nodos conectados.

El algoritmo de Dijkstra surge en un período en el que las telecomunicaciones y la planificación de rutas estaban en rápido desarrollo. La necesidad de encontrar caminos óptimos en redes de comunicación impulsó la creación de este algoritmo, que rápidamente se convirtió en un pilar en campos como la ingeniería de redes y la logística.

0.2 Desarrollo

Se ha implementado el algoritmo de Dijkstra en Python, utilizando una representación de grafo mediante listas de adyacencia. El código se estructura de manera clara y eficiente, asegurando la correcta identificación del camino más corto. Primero se Importa el módulo `heapq` para trabajar con colas de prioridad y el módulo `time` para medir el tiempo de ejecución y luego Definimos la clase `Graph` para representar un grafo ponderado dirigido.



```
Dijkstra.py
Dijkstra.py > Graph
1 import heapq
2 import time
3
4 class Graph:
5     def __init__(self):
6         self.vertices = {} #almacenar los vértices y sus aristas con pesos
7
8     def add_vertex(self, vertex):
9         self.vertices[vertex] = {} #Inicializamos las aristas del vértice
10
11     def add_edge(self, start, end, weight):
12         self.vertices[start][end] = weight #Añadimos una arista con su peso al vértice de i
13
```

Definimos la función `dijkstra` en la cual esta Función:

Inicializa las distancias desde el vértice de inicio como infinito y el vértice de inicio como 0. Utiliza una cola de prioridad (implementada como un heap) para almacenar las distancias y los vértices. e Itera sobre la cola de prioridad, seleccionando el vértice con la distancia mínima actual. Actualiza las distancias de los vecinos y los agrega a la cola de prioridad si se encuentra un camino más corto.

```

def dijkstra(graph, start):
    distances = {vertex: float('infinity') for vertex in graph.vertices}
    distances[start] = 0
    priority_queue = [(0, start)]

    while priority_queue:
        current_distance, current_vertex = heapq.heappop(priority_queue)

        if current_distance > distances[current_vertex]:
            continue

        for neighbor, weight in graph.vertices[current_vertex].items():
            distance = current_distance + weight

            if distance < distances[neighbor]:
                distances[neighbor] = distance
                heapq.heappush(priority_queue, (distance, neighbor))

    return distances

```

Creamos una instancia de la clase Graph y Agregamos vértices ('A', 'B', 'C', 'D', 'E') y aristas con pesos

```

33
34 # grafo de ejemplo
35 grafo = Graph()
36 vertices = ['A', 'B', 'C', 'D', 'E']
37 for v in vertices:
38     grafo.add_vertex(v)
39
40 aristas = [('A', 'B', 1), ('A', 'C', 4), ('B', 'D', 3), ('C', 'D', 1), ('D', 'E', 7)]
41 for arista in aristas:
42     grafo.add_edge(*arista)
43

```

Medimos el tiempo de ejecución de la función y devolvemos el resultado y el tiempo con la función `measureexecutiontime`.

Procedemos a especificar el vértice de inicio (A) e imprimimos los caminos mínimos y el tiempo de ejecución.

```

43
44 # medir el tiempo de ejecución
45 def measure_execution_time(func, *args):
46     start_time = time.time()
47     result = func(*args)
48     end_time = time.time()
49     execution_time = end_time - start_time
50     return result, execution_time
51
52 # Encontrar caminos mínimos desde un vértice de inicio dado
53 start_vertex = 'A'
54 resultado, tiempo_ejecucion = measure_execution_time(dijkstra, grafo, start_vertex)
55
56 # Imprimir resultados
57 print(f"Caminos mínimos desde {start_vertex}: {resultado}")
58 print(f"Tiempo de ejecución: {tiempo_ejecucion} segundos")
59

```

0.3 Resultados

El resultado obtenido fue lo siguiente:

```
PROBLEMS  OUTPUT  TERMINAL  PORTS  DEBUG CONSOLE

● PS C:\Users\Enida\OneDrive\Documentos\aimme\AnálisisDiseño\ExamenII> python .\Dijkstra.py
Camino mínimos desde A: {'A': 0, 'B': 1, 'C': 4, 'D': 4, 'E': 11}
Tiempo de ejecución: 0.0 segundos
○ PS C:\Users\Enida\OneDrive\Documentos\aimme\AnálisisDiseño\ExamenII> █
```

```
39
40 aristas = [('A', 'B', 5), ('A', 'C', 3), ('B', 'D', 3), ('C', 'D', 1), ('D', 'E', 7)]
41 for arista in aristas:
42     grafo.add_edge(*arista)
43
44 # medir el tiempo de ejecución
45 def measure execution time(func, *args):

PROBLEMS  OUTPUT  TERMINAL  PORTS  DEBUG CONSOLE

● PS C:\Users\Enida\OneDrive\Documentos\aimme\AnálisisDiseño\ExamenII> python .\Dijkstra.py
Camino mínimos desde A: {'A': 0, 'B': 5, 'C': 3, 'D': 4, 'E': 11}
Tiempo de ejecución: 0.0 segundos
○ PS C:\Users\Enida\OneDrive\Documentos\aimme\AnálisisDiseño\ExamenII> █
```

0.4 Conclusiones

La implementación del algoritmo de Dijkstra proporciona resultados precisos al encontrar los caminos más cortos desde un vértice de inicio a todos los demás vértices en un grafo ponderado dirigido.

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

La complejidad temporal del algoritmo de Dijkstra, en el peor de los casos, es $O((V + E) * \log(V))$, y la complejidad espacial es $O(V + E)$. Estas complejidades son aceptables para grafos de tamaño moderado y contribuyen a la eficiencia del algoritmo.

0.5 Referencias

- E. C. Navone, “Algoritmo de la ruta más corta de Dijkstra - Introducción gráfica y detallada,” freeCodeCamp.org, Aug. 02, 2023.
<https://www.freecodecamp.org/espanol/news/algoritmo-de-la-ruta-mas-corta-de-dijkstra-introduccion-grafica/>
- B. S. López, “Algoritmo de Dijkstra,” Ingenieria Industrial Online, Oct. 28, 2019.
<https://www.ingenieriaindustrialonline.com/investigacion-de-operaciones/algoritmo-de-dijkstra/>