Universidad del Caribe

2000 CANCUN, QUINTANA ROO, MÉXICO

CONOCIMIENTO Y CULTURA PARA EL DESARROLLO HUMANO

# 1. Introducción

El **algoritmo de Dijkstra** (1959) es un método para encontrar el camino más corto desde un nodo origen a todos los demás nodos en un grafo **con pesos no negativos**. Fue creado por el científico de computación neerlandés **Edsger W. Dijkstra** y publicado en 1959.

# 2. Historia

Dijkstra desarrolló este algoritmo mientras trabajaba en el **Centro Matemático de Ámsterdam**. Originalmente, lo diseñó para demostrar las capacidades de las computadoras ARMAC, pero rápidamente se convirtió en un pilar de la teoría de grafos y las redes de routing (como Internet).

# 3. Descripción del Algoritmo

Dado un grafo $G = (V, E)$ con:

- $V$: conjunto de vértices (nodos).
- $E$: conjunto de aristas (con pesos $w(e) \geq 0$).

El algoritmo sigue estos pasos:

1. Inicializar distancias: $d(s) = 0$ (origen) y $d(v) = \infty$ para los demás nodos.
2. Seleccionar el nodo con la distancia mínima no procesado.
3. Relajar las aristas: actualizar las distancias de los nodos adyacentes.
4. Repetir hasta que todos los nodos sean procesados.

# 4. Implementación en Python

## 4.1. Estructura del Algoritmo

El siguiente código implementa Dijkstra usando:

- `heapq` para gestión eficiente de colas de prioridad
- Diccionarios para almacenar distancias y predecesores

```python

import heapq

def dijkstra(graph, start):
    # Inicializar distancias como infinito
    distances = {node: float('infinity') for node in graph}
    distances[start] = 0
    predecessors = {node: None for node in graph}
    priority_queue = [(0, start)]

    while priority_queue:
        current_distance, current_node = heapq.heappop(
            priority_queue)
        if current_distance > distances[current_node]:
            continue

        for neighbor, weight in graph[current_node].items():
            distance = current_distance + weight
            if distance < distances[neighbor]:
                distances[neighbor] = distance
                predecessors[neighbor] = current_node
                heapq.heappush(priority_queue, (distance,
                    neighbor))
```

```
    return distances, predecessors
```

Listing 1: Implementación del Algoritmo de Dijkstra

### 4.2. Función Auxiliar

```
def get_shortest_path(predecessors, target):
    path = []
    current = target

    while current is not None:
        path.append(current)
        current = predecessors[current]

    return path[::-1]  # Invertir el camino
```

Listing 2: Reconstrucción del Camino Más Corto

## 5. Ejemplo Práctico

### 5.1. Grafo de Prueba

```
graph = {
    'A': {'B': 5, 'C': 1},
    'B': {'A': 5, 'C': 2, 'D': 1},
    'C': {'A': 1, 'B': 2, 'D': 4, 'E': 8},
    'D': {'B': 1, 'C': 4, 'E': 3, 'F': 6},
    'E': {'C': 8, 'D': 3},
    'F': {'D': 6}
}
```

Listing 3: Ejemplo de Grafo

### 5.2. Resultados

La ejecución desde el nodo 'A' produciría:

```
Distancias más cortas desde A:
A -> A: 0
A -> B: 3
A -> C: 1
A -> D: 4
A -> E: 7
A -> F: 10

Camino más corto de A a F: A -> C -> B -> D -> F
```

## 6. Análisis de Complejidad

- **Tiempo**: $O((V + E) \log V)$ con cola de prioridad
- **Espacio**: $O(V)$ para almacenar distancias y predecesores

## 7. Aplicaciones

- Sistemas de navegación (GPS).
- Routing en redes de computadoras.
- Optimización en redes de transporte.

## 8. Conclusión

El algoritmo de Dijkstra representa un hito fundamental en la teoría de grafos y la computación moderna. A través de este trabajo, hemos demostrado que:

- Su **elegante diseño greedy** lo hace óptimo para resolver problemas de caminos mínimos en grafos con pesos no negativos, alcanzando una complejidad computacional eficiente de $O((V + E) \log V)$ mediante el uso de colas de prioridad.
- La **versatilidad de sus aplicaciones** abarca desde sistemas de navegación GPS hasta routing en redes de telecomunicaciones, demostrando su relevancia seis décadas después de su publicación.
- Nuestra implementación en Python evidenció cómo la combinación de diccionarios y la estructura `heapq` permite una traducción directa del fundamento matemático a código ejecutable, manteniendo claridad conceptual.

**Reflexión final:** Pese a la aparición de alternativas como el algoritmo A* para casos específicos, el legado de Dijkstra persiste como piedra angular en la enseñanza de algoritmos y como herramienta práctica en ingeniería. Su simplicidad conceptual y eficiencia lo mantienen vigente en la era de los macrodatos y las redes complejas, confirmando que los clásicos algorítmicos, cuando están bien diseñados, trascienden las barreras temporales de la tecnología.