# Inteligencia Artificial


## Ingenieria en mecatrónica 6E1

## Practica #3

**Alumno(a):** Alma Paola Garcia Landeros
**Maestro(a):** Mauricio Alejandro Cabrera Arellano

# Algoritmo de Dijkstra

## ¿Qué es?

El algoritmo de Dijkstra es un método para encontrar el camino más corto en un grafo dirigido o no dirigido con pesos no negativos desde un nodo origen a todos los demás nodos del grafo.

## ¿Para qué sirve?

Sirve para resolver problemas de optimización de rutas, como encontrar la ruta más corta en redes de transporte, planificación de rutas en logística, en sistemas de navegación, en análisis de redes sociales, entre otros. Es utilizado en cualquier contexto donde se necesite encontrar la ruta más eficiente entre puntos que están interconectados y tienen costos asociados.

## ¿Cómo se implementa en el mundo?

El algoritmo de Dijkstra se implementa en software de navegación GPS para calcular las rutas más rápidas entre dos ubicaciones, en sistemas de logística para optimizar la distribución de recursos y en redes de telecomunicaciones para determinar caminos de datos eficientes.

## ¿Cómo lo implementarías en tu vida?

En la vida diaria, podrías usar una versión simplificada del algoritmo de Dijkstra mentalmente para planificar tu ruta más rápida al trabajo, a la escuela o a cualquier destino, considerando diferentes caminos y eligiendo aquellos que minimicen el tiempo o los costos involucrados.

## ¿Cómo lo implementarías en tu trabajo o tu trabajo de ensueño?

En mi trabajo soñado de diseño asistido por computadora para proyectos de automatización, usaría el algoritmo de Dijkstra para optimizar rutas de robots y vehículos autónomos, diseñar redes de comunicación eficientes, planificar trayectorias para brazos robóticos, y mejorar la gestión de recursos en procesos de fabricación y logística. Esto me ayudaría a minimizar costos, maximizar la eficiencia operativa y asegurar un diseño y control precisos en entornos industriales avanzados.

# Algoritmo:

```python
import sys

def dijkstra(graph, start):
    """
    Función para encontrar los caminos más cortos desde el nodo de inicio a todos los demás nodos
    utilizando el algoritmo de Dijkstra.

    Args:
    - graph: Grafo representado como una matriz de adyacencia.
    - start: Nodo de inicio desde el cual se calculan los caminos más cortos.

    Returns:
    - distances: Lista donde el índice representa el nodo y el valor representa la distancia más corta desde el nodo de inicio.
    - parent: Lista que contiene el nodo padre en el camino más corto desde el nodo de inicio.
    """
    n = len(graph)
    distances = [float('inf')] * n  # Distancias iniciales a todos los nodos como infinito
    distances[start] = 0  # Distancia al nodo de inicio es 0
    visited = [False] * n  # Lista para rastrear nodos visitados
    parent = [-1] * n  # Lista para rastrear el nodo padre en el camino más corto

    for _ in range(n):
        # Encontrar el nodo no visitado con la distancia mínima actual
        u = min_distance_node(distances, visited)
        visited[u] = True

        # Mostrar paso a paso la distancia mínima desde el nodo de inicio a cada nodo
        print_step_by_step(distances, parent, u)

        # Actualizar las distancias de los nodos adyacentes no visitados
        for v in range(n):
            if not visited[v] and graph[u][v] != 0 and distances[u] + graph[u][v] < distances[v]:
                distances[v] = distances[u] + graph[u][v]
                parent[v] = u

    return distances, parent
```

```python
def min_distance_node(distances, visited):
    """
    Función para encontrar el nodo no visitado con la distancia mínima actual.

    Args:
    - distances: Lista de distancias a cada nodo.
    - visited: Lista que indica si cada nodo ha sido visitado.

    Returns:
    - El nodo no visitado con la distancia mínima.
    """
    min_dist = float('inf')
    min_node = -1

    for i in range(len(distances)):
        if not visited[i] and distances[i] < min_dist:
            min_dist = distances[i]
            min_node = i

    return min_node

def print_step_by_step(distances, parent, u):
    """
    Función para mostrar paso a paso la distancia mínima desde el nodo de inicio a cada nodo.

    Args:
    - distances: Lista de distancias a cada nodo.
    - parent: Lista que indica el nodo padre en el camino más corto.
    - u: Nodo actual en el paso actual.
    """
    print(f"Explorando nodo {u}. Distancia mínima desde el nodo de inicio:")
    for i in range(len(distances)):
        path = get_path(parent, i)
        if distances[i] == float('inf'):
```

```
        if distances[i] == float('inf'):
            print(f"Nodo {i}: No alcanzable")
        else:
            print(f"Nodo {i}: Distancia = {distances[i]}, Camino = {' -> '.join(path)}")

def get_path(parent, node):
    """
    Función para obtener el camino desde el nodo de inicio hasta el nodo dado utilizando el padre de cada nodo.

    Args:
    - parent: Lista que indica el nodo padre en el camino más corto.
    - node: Nodo para el cual se desea obtener el camino.

    Returns:
    - Lista que representa el camino desde el nodo de inicio hasta el nodo dado.
    """
    path = []
    current = node
    while current != -1:
        path.append(str(current))
        current = parent[current]
    return path[::-1]

# Ejemplo de uso:
graph = [
    [0, 4, 0, 0, 0, 0, 0, 8, 0],
    [4, 0, 8, 0, 0, 0, 0, 11, 0],
    [0, 8, 0, 7, 0, 4, 0, 0, 2],
    [0, 0, 7, 0, 9, 14, 0, 0, 0],
    [0, 0, 0, 9, 0, 10, 0, 0, 0],
    [0, 0, 4, 14, 10, 0, 2, 0, 0],
    [0, 0, 0, 0, 0, 2, 0, 1, 6],
    [8, 11, 0, 0, 0, 0, 1, 0, 7],
    [0, 0, 2, 0, 0, 0, 6, 7, 0]
]
```

```

start_node = 0
print(f"Calculando caminos más cortos desde el nodo de inicio: {start_node}")
distances, parent = dijkstra(graph, start_node)

print("\nResultados finales:")
for i in range(len(distances)):
    path = get_path(parent, i)
    if distances[i] == float('inf'):
        print(f"Nodo {i}: No alcanzable")
    else:
        print(f"Nodo {i}: Distancia = {distances[i]}, Camino = {' -> '.join(path)}")


```

El código implementa el algoritmo de Dijkstra para encontrar los caminos más cortos desde un nodo de inicio dado hacia todos los demás nodos en un grafo representado como una matriz de adyacencia.