ALGORITMO DE DIJKSTRA

CON ANIMACIÓN

Módulo 1: Implementación y Visualización

Grafo de 15 Nodos con Distancias Reales

CONTENIDO

1. INTRODUCCIÓN

Este proyecto implementa el algoritmo de Dijkstra para encontrar la ruta más corta entre nodos en un grafo ponderado. El grafo representa 15 ubicaciones conectadas por caminos con distancias reales en kilómetros.

El algoritmo de Dijkstra es uno de los algoritmos más importantes en teoría de grafos y tiene aplicaciones prácticas en sistemas de navegación GPS, enrutamiento de redes, y planificación logística.

2. DESCRIPCIÓN DEL PROYECTO

El proyecto incluye las siguientes características:

- ✓ Implementación completa del algoritmo de Dijkstra
- ✓ Animación paso a paso del proceso de búsqueda
- ✓ Visualización del camino óptimo encontrado
- ✓ Múltiples versiones del programa (animada, simple, comparador)
- ✓ Suite de pruebas automatizadas
- ✓ Documentación completa

3. INSTALACIÓN Y REQUISITOS

Requisitos del Sistema:

• Python 3.7 o superior

• matplotlib

• networkx

Instalación:

```
pip install -r requirements.txt
```

O manualmente:

```
pip install matplotlib networkx
```

4. ARCHIVOS DEL PROYECTO

Archivo	Descripción
`dijkstra_animado.py`	Versión con animación completa paso a paso
`dijkstra_simple.py`	Versión simple que muestra solo el resultado final
`comparador_rutas.py`	Permite comparar múltiples rutas simultáneamente
`test_dijkstra.py`	Suite de pruebas automatizadas
`requirements.txt`	Lista de dependencias del proyecto
`README.md`	Documentación en formato Markdown
`Guia_Dijkstra.pdf`	Este documento

5. USO DE LOS PROGRAMAS

5.1 Versión Animada (dijkstra_animado.py)

Esta versión muestra una animación paso a paso del algoritmo de Dijkstra:

```
python dijkstra_animado.py
```

El programa solicitará el nodo de inicio y el nodo de destino, luego mostrará una animación donde se puede observar cómo el algoritmo explora los nodos, actualiza las distancias y finalmente encuentra el camino óptimo.

5.2 Versión Simple (dijkstra_simple.py)

Esta versión muestra directamente el resultado sin animación:

```
python dijkstra_simple.py
```

Ideal para obtener resultados rápidos o cuando no se necesita ver el proceso paso a paso.

5.3 Comparador de Rutas (comparador_rutas.py)

Permite comparar visualmente múltiples rutas simultáneamente:

```
python comparador_rutas.py
```

Útil para analizar diferentes opciones de ruta y compararlas lado a lado.

5.4 Suite de Pruebas (test_dijkstra.py)

Ejecuta pruebas automatizadas para verificar el correcto funcionamiento:

```
python test_dijkstra.py
```

6. ESTRUCTURA DEL GRAFO

El grafo contiene 15 nodos representados por letras (A-N, Ñ) y 17 aristas con pesos que representan distancias en kilómetros.

Conexiones del Grafo:

Nodo	Conexiones
A	B (0.9 km), D (1.1 km)
B	A (0.9 km), C (1.5 km)
C	B (1.5 km), L (2.2 km)
D	A (1.1 km), F (1.2 km)
E	G (1.1 km)
F	D (1.2 km), G (1.1 km), H (1.3 km)
G	E (1.1 km), F (1.1 km), I (0.8 km), K (3.5 km)
H	F (1.3 km), I (1.5 km)
I	G (0.8 km), H (1.5 km), J (1.3 km)
J	I (1.3 km), Ñ (2.1 km)
K	G (3.5 km), M (1.1 km), Ñ (1.4 km)
L	N (3.1 km), C (2.2 km)
M	K (1.1 km), N (1.1 km)
N	M (1.1 km), L (3.1 km)
Ñ	J (2.1 km), K (1.4 km)

7. ALGORITMO DE DIJKSTRA

El algoritmo de Dijkstra encuentra el camino más corto desde un nodo origen a todos los demás nodos en un grafo ponderado con pesos no negativos.

Funcionamiento:

1. **Inicialización:** Se establece la distancia del nodo origen como 0 y todas las demás distancias como infinito.

2. **Selección:** Se selecciona el nodo no visitado con la menor distancia.

3. **Actualización:** Para cada vecino del nodo actual, se calcula la distancia a través del nodo actual. Si esta distancia es menor que la distancia registrada, se actualiza.

4. **Marcado:** Se marca el nodo actual como visitado.

5. **Repetición:** Se repiten los pasos 2-4 hasta que todos los nodos hayan sido visitados o se alcance el nodo destino.

Complejidad Temporal:

Con una cola de prioridad implementada con heap: **O((V + E) log V)**
donde V = número de nodos y E = número de aristas.

8. EJEMPLOS DE USO

Rutas de Ejemplo:

Origen	Destino	Distancia	Camino
A	**N**	7.70 km	A → B → C → L → N
A	**Ñ**	7.60 km	A → D → F → G → I → J → Ñ
E	**M**	6.90 km	E → G → K → M
G	**I**	0.80 km	G → I
H	**N**	7.60 km	H → I → J → Ñ → K → M → N

Leyenda de Colores en la Visualización:

Color	Significado
Verde claro	Nodo de inicio
Rojo claro	Nodo de destino
Amarillo	Nodo actual en exploración
Azul claro	Nodo ya visitado
Amarillo claro	Nodo en cola de prioridad
Gris claro	Nodo no visitado
Verde (arista)	Camino óptimo final

9. PERSONALIZACIÓN

9.1 Modificar el Grafo

Para cambiar las conexiones o agregar nuevos nodos, edite el diccionario 'grafo' en cualquiera de los archivos Python:

```
grafo = {
'A': [('B', 0.9), ('D', 1.1)],
# Agrega o modifica conexiones aquí
}
```

9.2 Ajustar Posiciones de Nodos

Para cambiar la disposición visual de los nodos:

```
posiciones = {
'A': (0, 5), # Coordenadas (x, y)
# Modifica las posiciones aquí
}
```

9.3 Cambiar Velocidad de Animación

En dijkstra_animado.py, modifique el parámetro 'interval' (en milisegundos):

```
anim = animation.FuncAnimation(fig, actualizar,
frames=frames_totales,
interval=1000, # 1000 ms = 1 segundo
repeat=True)
```

10. CONCLUSIONES

Este proyecto proporciona una implementación completa y educativa del algoritmo de Dijkstra con múltiples herramientas de visualización. Es ideal para:

- Aprender el funcionamiento del algoritmo de Dijkstra
- Visualizar el proceso paso a paso
- Comparar diferentes rutas
- Experimentar con diferentes configuraciones de grafos
- Comprender conceptos de teoría de grafos

El código está bien documentado y estructurado para facilitar su comprensión y modificación. Se incluyen pruebas automatizadas para verificar el correcto funcionamiento del algoritmo.

Recursos Adicionales:

- Wikipedia - Algoritmo de Dijkstra: `https://es.wikipedia.org/wiki/Algoritmo_de_Dijkstra`
- NetworkX Documentation: `https://networkx.org/documentation/stable/`
- Matplotlib Animation: `https://matplotlib.org/stable/api/animation_api.html`