Práctica 3: “Algoritmo de Dijkstra”

Angel Bernardo Márquez Valdivia

22110348

Inteligencia Artificial

6E2

¿Qué es?

El algoritmo de Dijkstra, también llamado algoritmo de caminos mínimos es un algoritmo para la determinación del camino más corto, dado un vértice origen, hacia el resto de los vértices en un grafo que tiene pesos en cada arista. Su nombre alude a Edsger Dijkstra, científico de la computación de los Países Bajos que lo concibió en 1956 y lo publicó por primera vez en 1959.12

La idea subyacente en este algoritmo consiste en ir explorando todos los caminos más cortos que parten del vértice origen y que llevan a todos los demás vértices; cuando se obtiene el camino más corto desde el vértice origen hasta el resto de los vértices que componen el grafo, el algoritmo se detiene. Se trata de una especialización de la búsqueda de costo uniforme y, como tal, no funciona en grafos con aristas de coste negativo (al elegir siempre el nodo con distancia menor, pueden quedar excluidos de la búsqueda nodos que en próximas iteraciones bajarían el costo general del camino al pasar por una arista con costo negativo).

Funcionamiento

El algoritmo se basa en iterar a través de los nodos de un grafo, actualizando las distancias tentativas desde un nodo inicial hasta otros, seleccionando siempre el nodo con la menor distancia conocida. Utiliza una estructura de datos, como una cola de prioridad, para mantener eficiencia al procesar nodos.

Los pasos principales son:

1. **Inicialización:**

- Asignar una distancia de infinito a todos los nodos, excepto al nodo de inicio, que se establece en 0.
- Crear un conjunto de nodos no visitados.

2. **Seleccionar nodo:** Elegir el nodo no visitado con la menor distancia calculada.

3. **Actualizar distancias:** Para cada nodo vecino del nodo seleccionado:

- Calcular la distancia potencial desde el nodo inicial.
- Si esta nueva distancia es menor que la registrada, actualizar la distancia y registrar el nodo predecesor.

4. **Marcar nodo como visitado:** Una vez procesado, no se vuelve a considerar ese nodo.

5. **Repetir:** Continuar hasta que todos los nodos hayan sido visitados o la distancia mínima a los nodos restantes sea infinita.

Características principales:

1. **Grafo dirigido o no dirigido:** Funciona con grafos donde las aristas tienen pesos no negativos.

2. **Entrada:**
- Un grafo con nodos y aristas, donde cada arista tiene un peso asociado.
- Un nodo de inicio desde el cual calcular las distancias más cortas a los demás nodos.

3. **Salida:** La distancia mínima desde el nodo inicial a todos los demás nodos, y el camino que lleva a cada uno de ellos.

¿Para qué sirve?

El algoritmo de Dijkstra sirve principalmente para resolver problemas de rutas más cortas en un grafo ponderado, donde los pesos representan costos, distancias o tiempos. Es una herramienta fundamental en diversas aplicaciones prácticas y teóricas, gracias a su capacidad para encontrar eficientemente el camino óptimo entre un nodo de inicio y otros nodos.

Usos Principales:

1. **Navegación y Mapas:**
- Se utiliza en sistemas GPS para calcular la ruta más corta entre dos ubicaciones.
- Encuentra rutas óptimas teniendo en cuenta distancias o tiempos de viaje.

2. **Redes de Comunicación:**
- Optimiza el enrutamiento de datos en redes de computadoras, como Internet.
- Ayuda a calcular rutas mínimas en redes de telefonía y sistemas distribuidos.

3. **Gestión de Infraestructuras:**
- Planeación de redes de transporte, distribución de energía, o tuberías de agua.
- Diseña rutas eficientes para reducir costos operativos.

4. **Juegos y Simulaciones:**
- Implementado en inteligencia artificial para mover personajes en un entorno virtual.
- Se utiliza en simulaciones de tráfico y planeación urbana.

5. **Análisis de Grafos:**
- Resuelve problemas de conectividad en teoría de grafos.
- Permite estudiar redes sociales, mapas conceptuales, y sistemas logísticos.

¿Cómo se implementa en el mundo?

El algoritmo de Dijkstra tiene aplicaciones prácticas en diversas áreas del mundo real debido a su capacidad para encontrar rutas óptimas y gestionar recursos eficientemente en grafos ponderados. A continuación, se describen ejemplos de su implementación en el mundo:

1. **Sistemas de Navegación GPS**

Cómo se usa: Calcula la ruta más corta entre dos puntos en mapas, optimizando distancias o tiempos de viaje.

Ejemplo: Servicios como Google Maps, Waze o Apple Maps utilizan variaciones del algoritmo para guiar a los usuarios. Toman en cuenta factores como tráfico en tiempo real o restricciones de caminos.

2. **Redes de Telecomunicaciones**

Cómo se usa: En el diseño y operación de redes de datos (como Internet), Dijkstra encuentra las rutas más eficientes para transmitir paquetes de información.

Ejemplo: El protocolo OSPF (Open Shortest Path First), ampliamente utilizado en redes de routers, implementa una versión del algoritmo para determinar las rutas más cortas entre dispositivos.

3. **Gestión de Infraestructura y Logística**

Cómo se usa: Optimiza rutas en sistemas de transporte público, distribución de bienes, o redes de suministro eléctrico y agua.

Ejemplo:

- Empresas de logística como FedEx y UPS usan este algoritmo para planificar rutas de entrega.
- Sistemas de planificación urbana lo implementan para diseñar carreteras y redes de transporte público.

4. **Inteligencia Artificial y Videojuegos**

Cómo se usa: Planifica movimientos en mundos virtuales, como videojuegos, donde los personajes necesitan encontrar el camino más corto hacia un objetivo.

Ejemplo: Juegos de estrategia en tiempo real (RTS) o simuladores de tráfico emplean este algoritmo para calcular trayectorias.

5. **Optimización de Redes Sociales**

Cómo se usa: Ayuda a analizar y optimizar la conectividad en redes sociales al encontrar caminos más cortos entre usuarios o nodos.

Ejemplo: Plataformas como Facebook podrían utilizar este tipo de algoritmos para mejorar la conectividad entre personas con intereses similares.

6. **Robótica y Automatización**

Cómo se usa: En robots autónomos, se utiliza para la navegación en espacios físicos, evitando obstáculos y optimizando trayectorias.

Ejemplo: Vehículos autónomos como los desarrollados por Tesla o Waymo lo emplean como parte de sus sistemas de navegación.

7. **Simulaciones y Análisis de Tráfico**

Cómo se usa: Modela flujos de tráfico en sistemas de transporte urbano para identificar rutas congestionadas y sugerir alternativas.

Ejemplo: Ciudades inteligentes utilizan el algoritmo para gestionar el tráfico en tiempo real, reduciendo embotellamientos.

8. **Análisis de Redes Biológicas**

Cómo se usa: En bioinformática, encuentra rutas óptimas en redes moleculares, como rutas metabólicas.

Ejemplo: Ayuda en el análisis de interacciones proteína-proteína y redes de genes.

¿Cómo lo implementarías en tu vida?

En mi día a día, especialmente lo implementaria cuando estoy conduciendo, encuentro aplicaciones prácticas del algoritmo de Dijkstra en sistemas de navegación como Google Maps. Estos sistemas utilizan este algoritmo para calcular la ruta más corta entre mi ubicación actual y mi destino, considerando factores como la distancia y el tiempo estimado. Mientras conduzco, el sistema analiza todas las posibles rutas y selecciona aquella que tenga el menor tiempo de recorrido, incluso ajustándose en tiempo real si detecta tráfico u obstáculos inesperados. Esto me permite optimizar mi trayecto y llegar de forma más rápida y eficiente.

Otra área donde podría implementar este concepto es en simulaciones de tráfico. Antes de salir, pudiera simular diferentes escenarios de tráfico basados en datos en tiempo real, como accidentes o construcciones en la carretera. Usando un modelo de tráfico con el algoritmo de Dijkstra, podría identificar qué trayectos se encuentran más despejados y ajustar mi ruta antes de empezar.

También, por último, lo podría implementar en la realización de un proyecto que involucra dependencias entre tareas, este algoritmo me podría ayudar a encontrar la secuencia más eficiente para completar todas las actividades sin retrasos innecesarios. Prioriza las tareas mas importantes que otras para ser más eficiente.

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

En el trabajo de mis sueños me gustaría trabajar en una empresa enfocada en la Robótico y Automatización esencialmente en un área que involucren a los autos basándonos en esto este algoritmo lo podría implementar en múltiples cosas como:

1. **Sistemas de Navegación Autónoma**

Uno de los principales desafíos en los vehículos autónomos es garantizar que puedan navegar de manera segura y eficiente en entornos complejos. El algoritmo de Dijkstra sería fundamental para calcular rutas óptimas que eviten obstáculos, reduzcan el tiempo de viaje y minimicen el consumo de energía.

2. **Optimización de Movimientos en Robots de Ensamblaje**

En la línea de producción automatizada, los robots que ensamblan vehículos necesitan moverse entre diferentes estaciones. Utilizaría el algoritmo para optimizar las trayectorias de estos robots, reduciendo el tiempo total de ensamblaje y evitando colisiones.

3. **Simulaciones de Tráfico y Diseño Urbano Inteligente**

Trabajando en una empresa de robótica vehicular, también podría participar en proyectos que integren simulaciones de tráfico urbano. Aquí, el algoritmo sería crucial para modelar y prever flujos de tráfico, identificando áreas críticas y sugiriendo soluciones.

Ejemplo de Algoritmo de Dijkstra

El código es una aplicación interactiva con una interfaz gráfica que utiliza el algoritmo de Dijkstra para encontrar la ruta más corta entre dos puntos seleccionados por el usuario en un conjunto de ciudades interconectadas.

La interfaz permite al usuario elegir una ciudad de inicio y una ciudad de destino desde un menú desplegable. Luego, ejecuta el algoritmo de Dijkstra para calcular la ruta más corta entre esos puntos. Finalmente, la ruta calculada se resalta visualmente en un lienzo que representa las ciudades y sus conexiones.

Funcionamiento:

1. **Interfaz Gráfica al Iniciar:**

Se cargan las ciudades, las rutas entre ellas y se muestra una interfaz para seleccionar inicio y destino.

2. **El Usuario Selecciona un Punto de Inicio y Destino:**

Desde los menús desplegables, elige las ciudades deseadas.

3. **Presiona el Botón "Buscar Ruta":**

Se ejecuta el cálculo con el algoritmo de Dijkstra.

4. **La Ruta Más Corta se Resalta en el Lienzo:**

Se dibuja visualmente en rojo para mostrar el camino más corto.

5. **El Usuario Recibe un Mensaje:**

Se le informa mediante un cuadro de diálogo (messagebox) sobre el camino encontrado.

Estructura del Código

1. **Estructura del grafo:**

El grafo representa las ciudades como nodos y las rutas entre ellas como aristas con un peso numérico. Esto se almacena en un diccionario llamado graph:

```
graph = {
    'Ciudad A': {'Ciudad B': 4, 'Ciudad C': 2},
    'Ciudad B': {'Ciudad A': 4, 'Ciudad C': 5, 'Ciudad D': 10},
    'Ciudad C': {'Ciudad A': 2, 'Ciudad B': 5, 'Ciudad D': 3},
    'Ciudad D': {'Ciudad B': 10, 'Ciudad C': 3, 'Ciudad E': 7},
    'Ciudad E': {'Ciudad D': 7}
}
```

Cada clave es una ciudad y el valor asociado es un diccionario con sus ciudades vecinas y el peso (distancia) de la conexión. Por ejemplo:

'Ciudad A' tiene rutas hacia 'Ciudad B' y 'Ciudad C' con pesos 4 y 2 respectivamente.

2. **Algoritmo de Dijkstra**

La función dijkstra(graph, start, end) implementa el algoritmo de Dijkstra para encontrar la ruta más corta desde un nodo de inicio (start) hasta un nodo de destino (end). Aquí se hace uso de una cola de prioridad para explorar las rutas más cortas primero:

Explicación:

- **queue:** Se utiliza para realizar la búsqueda, manteniendo los nodos en orden de distancia.
- **distances:** Almacena la distancia más corta desde el nodo de inicio hasta cada nodo.
- **previous_nodes:** Almacena información sobre el camino para reconstruir la ruta más tarde.
- **heapq:** Mantiene la cola de prioridad ordenada automáticamente.

3. **Reconstruir la Ruta Más Corta**

La función reconstruct_path(previous_nodes, start, end) recupera la ruta más corta utilizando el diccionario previous_nodes generado por el algoritmo de Dijkstra:

```
def reconstruct_path(previous_nodes, start, end):
    path = []
    current_node = end
    while current_node:
        path.append(current_node)
        current_node = previous_nodes[current_node]
    path.reverse()
    return path
```

Esto permite reconstruir el camino desde el nodo destino hasta el nodo de inicio, utilizando el historial de nodos visitados.

4. **Interfaz Gráfica con Tkinter**

El código utiliza Tkinter, una biblioteca estándar para crear interfaces gráficas en Python.

Componentes en la Interfaz:

- Canvas (self.canvas):

Representa las ciudades como nodos visuales y las rutas entre ellas como líneas.

Se utiliza para dibujar la representación gráfica.

- **Botón de búsqueda (Buscar Ruta):**

Ejecuta el algoritmo cuando el usuario selecciona un punto de inicio y un destino.

- **Desplegables (ttk.Combobox):**

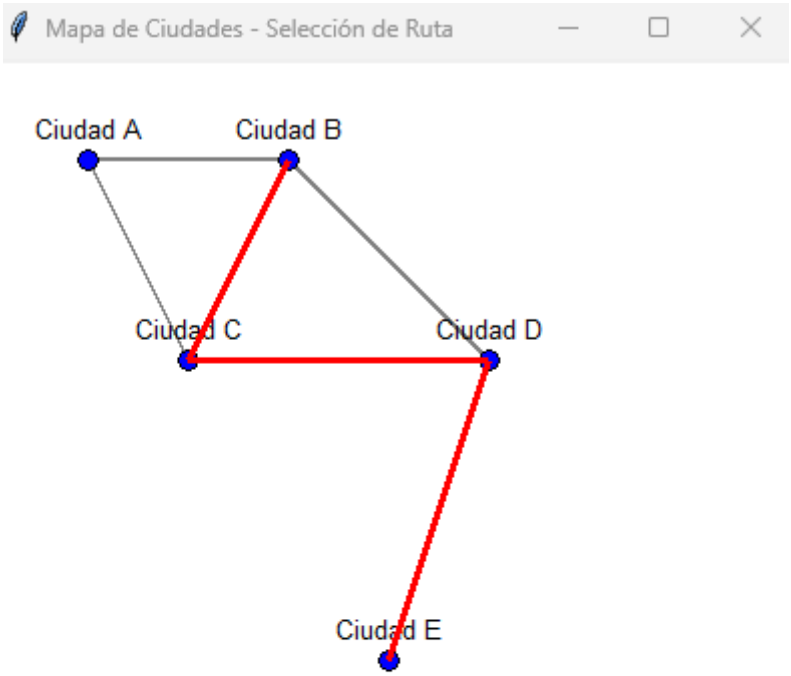
Permiten al usuario seleccionar el punto de inicio y el punto de destino de una lista de ciudades disponibles.

5. Visualización de la Ruta Más Corta

Una vez encontrada la ruta más corta, la función highlight_path(path) sobrescribe el lienzo para dibujar la ruta entre las ciudades seleccionadas en rojo:

```
def highlight_path(self, path):
    self.canvas.delete("highlight")
    for i in range(len(path) - 1):
        x1, y1 = self.city_positions[path[i]]
        x2, y2 = self.city_positions[path[i+1]]
        self.canvas.create_line(x1, y1, x2, y2, fill="red", width=3, tags="highlight")
```

- Borra la ruta previa para evitar duplicaciones.
- Dibuja las líneas en rojo entre los nodos consecutivos en la ruta seleccionada.


Mapa de Ciudades - Selección de Ruta
Ciudad A
Ciudad B
Ciudad C
Ciudad D
Ciudad E
Buscar Ruta
Selecciona punto de inicio:
Ciudad B
Selecciona punto de destino:
Ciudad E