IBEROAMERICANA
CORPORACIÓN UNIVERSITARIA

Inteligencia Artificial

Actividad 2 - Búsqueda y Sistemas basados en reglas

Presentado Por:

Rafael Ricardo Rojas Rúa

Código Banner: 100140224

Profesora:

Karla Sánchez

Corporación Universitaria Iberoamericana Atlántico

Julio 2024

Introducción

En este trabajo, se aborda la implementación de un sistema inteligente para encontrar la ruta óptima en un sistema de transporte masivo utilizando el algoritmo de Dijkstra. Este algoritmo es fundamental en el campo de la inteligencia artificial y las ciencias de la computación debido a su capacidad para resolver problemas de rutas más cortas en grafos ponderados.

Codigo .PY

dijkstra.py

C: > Users > RafaelRojas > Documents > IA > IA > dijkstra.py > ...

```python
import heapq

# Definición de la clase Grafo
class Grafo:
    def __init__(self):
        self.nodos = set()          # Conjunto de nodos del grafo
        self.aristas = {}           # Diccionario de aristas, donde cada nodo apunta a una lista de tuplas (vecino, costo)

    def agregar_nodo(self, valor):
        self.nodos.add(valor)       # Agrega un nodo al conjunto de nodos
        self.aristas[valor] = []    # Inicializa la lista de aristas para el nodo agregado

    def agregar_arista(self, desde, hasta, costo):
        self.aristas[desde].append((hasta, costo))  # Agrega una arista del nodo 'desde' al nodo 'hasta' con el costo asociado
        self.aristas[hasta].append((desde, costo))  # Agrega una arista del nodo 'hasta' al nodo 'desde' (grafo no dirigido)

# Implementación del algoritmo de Dijkstra
def dijkstra(grafo, inicio):
    queue = []  # Cola de prioridad para seleccionar el nodo con la menor distancia
    heapq.heappush(queue, (0, inicio))  # Inserta el nodo inicial en la cola con distancia 0
    distancias = {nodo: float('infinity') for nodo in grafo.nodos}  # Inicializa distancias a infinito
    distancias[inicio] = 0  # La distancia al nodo inicial es 0
    camino = {nodo: None for nodo in grafo.nodos}  # Diccionario para almacenar el camino más corto

    while queue:  # Mientras haya nodos en la cola
        (costo_actual, nodo_actual) = heapq.heappop(queue)  # Extrae el nodo con la menor distancia

        for vecino, peso in grafo.aristas[nodo_actual]:  # Recorre los vecinos del nodo actual
            costo = costo_actual + peso  # Calcula el costo de llegar al vecino
            if costo < distancias[vecino]:  # Si se encuentra una ruta más corta
                distancias[vecino] = costo  # Actualiza la distancia mínima al vecino
                camino[vecino] = nodo_actual  # Actualiza el nodo anterior en el camino más corto
                heapq.heappush(queue, (costo, vecino))  # Inserta el vecino en la cola con la nueva distancia

    return distancias, camino  # Retorna las distancias y el camino más corto

# Función para construir el camino más corto desde el nodo inicial hasta el nodo final
def construir_camino(camino, inicio, fin):
    ruta = []  # Lista para almacenar la ruta
    nodo_actual = fin  # Comienza desde el nodo final
    while nodo_actual != inicio:  # Mientras no se llegue al nodo inicial
        ruta.append(nodo_actual)  # Agrega el nodo actual a la ruta
        nodo_actual = camino[nodo_actual]  # Se mueve al nodo anterior en el camino más corto
    ruta.append(inicio)  # Agrega el nodo inicial a la ruta
    ruta.reverse()  # Invierte la ruta para obtenerla en el orden correcto
    return ruta  # Retorna la ruta

# Ejemplo de uso del algoritmo
grafo = Grafo()
nodos = ['A', 'B', 'C', 'D', 'E']  # Lista de nodos
for nodo in nodos:
    grafo.agregar_nodo(nodo)  # Agrega cada nodo al grafo

aristas = [
    ('A', 'B', 1),
    ('A', 'C', 4),
    ('B', 'C', 2),
```

```
    ('B', 'D', 5),
    ('C', 'D', 1),
    ('D', 'E', 3)
]

for arista in aristas:
    grafo.agregar_arista(*arista)  # Agrega cada arista al grafo

inicio = 'A'  # Nodo inicial
fin = 'E'  # Nodo final
distancias, camino = dijkstra(grafo, inicio)  # Ejecuta el algoritmo de Dijkstra
ruta = construir_camino(camino, inicio, fin)  # Construye el camino más corto

# Imprime la mejor ruta y su costo
print(f"La mejor ruta desde {inicio} hasta {fin} es: {ruta} con un costo de {distancias[fin]}")
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\RafaelRojas> cd Documents
PS C:\Users\RafaelRojas\Documents> cd ia
PS C:\Users\RafaelRojas\Documents\ia> cd ia
PS C:\Users\RafaelRojas\Documents\ia\ia> py dijkstra.py
La mejor ruta desde A hasta E es: ['A', 'B', 'C', 'D', 'E'] con un costo de 7
PS C:\Users\RafaelRojas\Documents\ia\ia>
```

Conclusión

La implementación del algoritmo de Dijkstra en este proyecto ha demostrado ser una herramienta poderosa y eficiente para determinar la ruta más corta , A través de la representación de las estaciones y las rutas como un grafo ponderado, y la aplicación de técnicas basadas en búsquedas heurísticas, se ha logrado optimizar la navegación y mejorar la experiencia del usuario. Este proyecto no solo ilustra la aplicabilidad práctica de los conceptos avanzados de inteligencia artificial, sino que también subraya la importancia de los sistemas basados en reglas para la toma de decisiones automatizada. En resumen, la combinación de la teoría y la práctica en este trabajo proporciona una base sólida para futuros desarrollos en el campo de los sistemas inteligentes .