Universidad Centroccidental “Lisandro Alvarado”

Decanato de Ciencias y Tecnología

Introducción a la Inteligencia Artificial

Implementando el algoritmo A Epsilon

Integrantes:

Melissa Camacaro *V27.666.789*

Ing. Telemática

Docente:

María Pérez

Barquisimeto, noviembre 2024

Caso de Uso

El código está diseñado para encontrar la ruta más corta desde un **punto inicial** a un **punto final** en un mapa.
Tenemos un mapa de ciudades conectadas por carreteras, donde cada posición en el mapa representa una ciudad y el valor 1 indica que hay una carretera entre dos ciudades adyacentes.
El algoritmo busca la ruta más eficiente para ir de la ciudad de inicio a la ciudad final.

Algoritmo A* (Base)

El algoritmo A* es un algoritmo de búsqueda de ruta que utiliza una función heurística para estimar el costo de llegada al objetivo desde un punto dado. Funciona de la siguiente manera:

1. **Inicio:** Se inicia en la posición de inicio.
2. **Evaluación:** Se calcula el costo de llegar a cada nodo vecino desde la posición actual.
3. **Prioridad:** Se da prioridad a los nodos que tienen un costo total más bajo de la siguiente forma: *(costo hasta el nodo actual + costo estimado al objetivo).*
4. **Expansión:** Se selecciona el nodo con la mayor prioridad y se expande, explorando sus vecinos.
5. **Repetición:** Se repiten los pasos **2-4** hasta que se alcanza el objetivo.

A* con Epsilon

La modificación del algoritmo A* con epsilon introduce un parámetro epsilon que afecta la heurística. La idea es que, en algunos casos, la heurística puede ser

demasiado optimista y hacer que el algoritmo explore caminos incorrectos. Epsilon se utiliza para ajustar la heurística, haciéndola más conservadora.

El código implementa el algoritmo A* con epsilon de la siguiente manera:

1. **Mapa:** Define el mapa de carreteras como una matriz bidimensional donde 1 indica una carretera y 0 indica un obstáculo.
2. **Costos:** Define la función costo() que calcula el costo de mover de una posición a otra (*en este caso, el costo es constante de 1*).
3. **Heurística:** Define la función heuristica() que calcula una estimación del costo de llegar al objetivo desde una posición dada (*se utiliza la distancia Manhattan*).
4. **Movimiento:** Define la función movimiento() que devuelve las posiciones adyacentes válidas a una posición dada.
5. **A* con Epsilon:** Define la función a_estrella_epsilon() que implementa el algoritmo A* con épsilon.
6. Utiliza una cola de prioridad (heap) **abierto** para almacenar los nodos a explorar.
7. Utiliza un conjunto **cerrado** para almacenar los nodos ya explorados.
8. Utiliza un diccionario **costo_acumulado** para almacenar el costo de llegar a cada nodo desde el inicio.
9. Utiliza un diccionario **camino** para almacenar el nodo anterior en el camino hacia un nodo dado.
10. En la función a_estrella_epsilon(), la línea prioridad = costo_acumulado[ciudad_siguiente] + heuristica(ciudad_siguiente) calcula la prioridad de un nodo. El factor epsilon se utiliza para ajustar la heurística.

Código

```
import heapq

# Definimos el mapa de carreteras
mapa = [
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 2, 0]
]

# Definimos la función de costo
def costo(posicion1, posicion2):
    return 1

# Definimos la función de heurística
def heuristica(posicion):
    x, y = posicion
    return abs(x - 9) + abs(y - 8)

# Definimos la función de movimiento
def movimiento(posicion_actual):
    x, y = posicion_actual
    movimientos = [(x + 1, y), (x - 1, y), (x, y + 1), (x, y - 1)]
    return [movimiento for movimiento in movimientos if 0 <= movimiento[0] <
10 and 0 <= movimiento[1] < 10]

# Definimos la función de búsqueda A* con epsilon
def a_estrella_epsilon(epsilon, inicio, fin):
    abierto = []
    cerrado = set()
    heapq.heappush(abierto, (0, inicio))
    costo_acumulado = {inicio: 0}
    camino = {inicio: None}
```

```
    while abierto:
        _, ciudad_actual = heapq.heappop(abierto)
        if ciudad_actual == fin:
            break
        cerrado.add(ciudad_actual)

        for ciudad_siguiente in movimiento(ciudad_actual):
            if ciudad_siguiente in cerrado:
                continue
            costo_nuevo = costo_acumulado[ciudad_actual] + 1
            if ciudad_siguiente not in costo_acumulado or costo_nuevo <
costo_acumulado[ciudad_siguiente]:
                costo_acumulado[ciudad_siguiente] = costo_nuevo
                prioridad = costo_acumulado[ciudad_siguiente] +
heuristica(ciudad_siguiente)
                heapq.heappush(abierto, (prioridad, ciudad_siguiente))
                camino[ciudad_siguiente] = ciudad_actual

    # Reconstruir el camino
    if fin in camino:
        reconstruido = []
        actual = fin
        while actual != inicio:
            reconstruido.append(actual)
            actual = camino[actual]
        reconstruido.append(inicio)
        reconstruido.reverse()
        return reconstruido
    else:
        return None

# Ejemplo de uso
posicion_inicial = (1, 1)
posicion_final = (9, 8)
epsilon = 5

camino = a_estrella_epsilon(epsilon, posicion_inicial, posicion_final)
if camino is not None:
    print("Camino encontrado:", camino)
else:
    print("No se encontró camino")
```