INSTITUTO POLITÉCNICO NACIONAL
ESCUELA SUPERIOR DE CÓMPUTO
INTELIGENCIA ARTIFICIAL
ING. EN SISTEMAS COMPUTACIONALES

Laboratorio 4: Búsqueda informada

Alumno: Hurtado Morales Emiliano - 2021630390

Maestro: Andrés García Floriano

Grupo: 6CV3

Fecha de entrega: 04/10/2024

Ciclo Escolar: 2025 – 1

Introducción

El problema de búsqueda de caminos es un tema fundamental en los campos de la inteligencia artificial, la teoría de grafos y la informática. En la vida cotidiana, este tipo de problema es aplicable a numerosas situaciones, como la planificación de rutas en sistemas de navegación, la resolución de laberintos en videojuegos, el diseño de circuitos, y el modelado de redes de tráfico. Uno de los algoritmos más efectivos y ampliamente utilizados para abordar este tipo de problemas es el algoritmo A* (A-estrella).

El algoritmo A* destaca por su capacidad de encontrar la ruta más corta entre un punto de inicio y un punto final en un entorno definido, a través del uso de una función heurística que guía su búsqueda hacia el objetivo de manera eficiente. A diferencia de otros métodos de búsqueda, como la búsqueda en anchura o en profundidad, A* combina elementos de búsqueda informada y no informada, logrando un equilibrio entre la rapidez y la precisión al encontrar la ruta óptima.

En este reporte, se presentará la aplicación del algoritmo A* para resolver dos laberintos de diferente complejidad: un laberinto sencillo y uno más complejo. El objetivo es demostrar cómo el algoritmo puede adaptarse a entornos con diferentes niveles de obstáculos y caminos, identificando de manera eficaz la ruta más corta. Para evaluar su desempeño, se medirá el tiempo de ejecución en cada escenario y se analizará el rendimiento del algoritmo en función de la complejidad del laberinto.

El algoritmo A* es esencial en el campo de la inteligencia artificial y en la resolución de problemas de caminos debido a su capacidad para encontrar soluciones óptimas mientras minimiza el costo computacional. En un entorno de laberinto, cada casilla representa un nodo y las conexiones entre ellas representan posibles caminos que el algoritmo debe explorar. Utilizando un enfoque basado en la función heurística, A* puede priorizar aquellos caminos que parecen más prometedores, evitando la exploración innecesaria de áreas que no conducen al objetivo.

Este reporte busca no solo ilustrar el funcionamiento y la eficiencia del algoritmo A*, sino también proporcionar un análisis detallado de sus resultados y comportamiento al enfrentarse a laberintos de diferentes complejidades.

Explicación de los Algoritmos

El algoritmo A* es un método de búsqueda que se basa en la idea de encontrar el camino más corto entre dos puntos utilizando una estrategia informada, que combina el costo acumulado de alcanzar un punto y una estimación heurística de la distancia que queda para alcanzar el destino. Esta combinación permite que el algoritmo se enfoque en la solución más prometedora de manera eficiente, en lugar de explorar todas las opciones posibles.

Descripción del Código:

El algoritmo A* utiliza tres elementos principales en su funcionamiento:

- **Conjunto abierto (Open Set):** Este conjunto almacena los nodos que están por explorar. Inicialmente, solo contiene el nodo de inicio. A medida que el algoritmo avanza, los nodos vecinos se agregan al conjunto abierto si se considera que podrían llevar a la solución.
- **Conjunto cerrado (Closed Set):** Este conjunto almacena los nodos que ya han sido explorados y no necesitan ser revisados nuevamente, optimizando el proceso de búsqueda.
- **Función de costo total (f(n) = g(n) + h(n)):** Esta función es la base de A*. Aquí, g(n) representa el costo desde el nodo inicial hasta el nodo actual, y h(n) es la estimación heurística del costo desde el nodo actual hasta el nodo objetivo.

El algoritmo sigue los siguientes pasos:

- **Inicialización:** Se comienza agregando el nodo de inicio al conjunto abierto, estableciendo su g(n) en 0 y calculando su f(n) utilizando la función heurística h(n).

```
# Función heurística para el algoritmo A* (distancia de Manhattan)
def heuristica(a, b):
    return abs(a[0] - b[0]) + abs(a[1] - b[1])
```

- **Selección del Nodo Actual:** En cada iteración, el nodo con el menor valor de f(n) en el conjunto abierto es seleccionado como el nodo actual. Este nodo representa la opción más prometedora para alcanzar el objetivo de manera eficiente.

```
while conjunto_abierto:
        _, costo_actual, actual = heapq.heappop(conjunto_abierto)
```

- **Verificación del Objetivo:** Si el nodo actual es el nodo objetivo (destino), el algoritmo termina y se reconstruye el camino desde el nodo inicial hasta el objetivo utilizando la información almacenada.

```
        if actual == final:
            # Reconstrucción del camino
            camino = []
            while actual in proveniente_de:
                camino.append(actual)
                actual = proveniente_de[actual]
            camino.append(inicio)
            return camino[::-1], costo_actual
```

- **Exploración de Vecinos:** Si el nodo actual no es el objetivo, se examinan sus vecinos inmediatos (nodos adyacentes). Para cada vecino:

```
# Exploración de vecinos (arriba, abajo, izquierda, derecha)
        vecinos = [(0, 1), (1, 0), (0, -1), (-1, 0)]
        for dr, dc in vecinos:
            vecino = (actual[0] + dr, actual[1] + dc)

            if 0 <= vecino[0] < filas and 0 <= vecino[1] < columnas and
laberinto[vecino[0]][vecino[1]] == 0:
                puntaje_g_tentativo = puntaje_g[actual] + 1

                if vecino not in puntaje_g or puntaje_g_tentativo <
puntaje_g[vecino]:
                    puntaje_g[vecino] = puntaje_g_tentativo
                    puntaje_f = puntaje_g_tentativo + heuristica(vecino, final)
                    heapq.heappush(conjunto_abierto, (puntaje_f,
puntaje_g_tentativo, vecino))
                    proveniente_de[vecino] = actual
```

  - Se calcula el costo acumulado g(n) desde el nodo de inicio hasta el vecino.
  - Si el vecino no está en el conjunto abierto o se encuentra un camino más corto hacia él, se actualizan sus valores de g(n) y f(n), y se añade al conjunto abierto si aún no está presente.

- **Repetición:** El proceso se repite hasta que se alcance el objetivo o se determine que no hay un camino posible.

Para medir la eficiencia del algoritmo A* en cada escenario, se utilizó la función **time.perf_counter()**, que ofrece una mayor precisión en la medición de tiempos de ejecución. Esta medición es crucial para comparar el rendimiento del algoritmo en entornos de diferentes niveles de complejidad y comprobar su eficacia al encontrar la ruta más corta en el menor tiempo posible.

Resultados Esperados

1. Laberinto Sencillo

```
# Definición del laberinto proporcionado
laberinto = [
    [1, 0, 1, 1, 1],
    [1, 0, 0, 0, 1],
    [1, 1, 1, 0, 1],
    [1, 0, 0, 0, 0],
    [1, 1, 1, 1, 1]
]
inicio = (0, 1)  # Coordenadas de inicio (fila, columna)
final = (3, 4)   # Coordenadas de salida (fila, columna)
```

```
Resultados para el laberinto proporcionado:
Camino encontrado: [(0, 1), (1, 1), (1, 2), (1, 3), (2, 3), (3, 3), (3, 4)]
Costo del camino: 6
Tiempo de ejecución: 0.0000355000 segundos
```

2. Laberinto Complejo

```
# Definición de un laberinto más complejo y de mayor tamaño
laberinto_complejo = [
    [1, 0, 1, 1, 1, 1, 1, 1, 1, 1],
    [1, 0, 0, 0, 1, 0, 0, 0, 0, 1],
    [1, 1, 1, 0, 1, 1, 1, 1, 0, 1],
    [1, 0, 0, 0, 0, 0, 0, 1, 0, 1],
    [1, 0, 1, 1, 1, 1, 0, 1, 0, 1],
    [1, 0, 1, 0, 0, 0, 0, 1, 0, 1],
    [1, 0, 1, 0, 1, 1, 1, 1, 0, 1],
    [1, 0, 0, 0, 0, 0, 0, 0, 0, 1],
    [1, 1, 1, 1, 1, 1, 1, 1, 0, 1],
    [1, 1, 1, 1, 1, 1, 1, 1, 0, 1]
]
inicio_complejo = (0, 1)  # Coordenadas de inicio (fila, columna)
final_complejo = (9, 8)   # Coordenadas de salida (fila, columna)
```

```
Resultados para el laberinto más complejo:
Camino encontrado: [(0, 1), (1, 1), (1, 2), (1, 3), (2, 3), (3, 3), (3, 2), (3, 1), (4, 1), (5, 1),
Costo del camino: 20
Tiempo de ejecución: 0.0001051000 segundos
```

```
(6, 1), (7, 1), (7, 2), (7, 3), (7, 4), (7, 5), (7, 6), (7, 7), (7, 8), (8, 8), (9, 8)]
```

Conclusión

El algoritmo A* ha demostrado ser una herramienta altamente eficiente y eficaz para la resolución de problemas de búsqueda de caminos, como se observó en la experimentación con los dos laberintos de diferente complejidad. A través de la implementación y aplicación del algoritmo en un laberinto sencillo y uno más complejo, se confirmó que A* es capaz de encontrar la ruta más corta de manera rápida y precisa, incluso cuando el entorno presenta múltiples obstáculos y rutas alternativas.

Uno de los aspectos más destacados del algoritmo es su capacidad para integrar la función heurística h(n) y el costo acumulado g(n), lo que permite tomar decisiones informadas durante el proceso de búsqueda. Esta combinación garantiza que A* no solo explore los caminos más prometedores, sino que también evite la exploración innecesaria de rutas que no llevan al objetivo, optimizando así la búsqueda. En el laberinto sencillo, el algoritmo fue capaz de encontrar el camino más corto casi instantáneamente, mientras que, en el laberinto más complejo, mostró un rendimiento igualmente notable al identificar la ruta óptima en un tiempo de ejecución muy reducido.

La medición de los tiempos de ejecución utilizando time.perf_counter() confirmó que A* es extremadamente rápido, incluso en escenarios de mayor complejidad. Esto respalda la aplicación del algoritmo en situaciones del mundo real donde la búsqueda de rutas rápidas y eficientes es crucial, como en la navegación por GPS, la planificación de rutas en videojuegos, la gestión de tráfico, y la robótica.

Sin embargo, es importante destacar que la eficiencia del algoritmo A* depende en gran medida de la función heurística seleccionada. En este caso, la distancia de Manhattan fue adecuada para un entorno de laberinto, pero en otros contextos o espacios de búsqueda, podría ser necesario utilizar o adaptar otras heurísticas para obtener resultados óptimos.

En conclusión, la experimentación demostró que el algoritmo A* es una solución robusta y versátil para problemas de búsqueda de caminos. Su capacidad para encontrar la ruta más corta de manera eficiente y su adaptabilidad a diferentes entornos lo convierten en una herramienta valiosa en el ámbito de la inteligencia artificial y la informática. A medida que se enfrentan problemas más complejos y se exploran entornos de búsqueda más grandes, el uso del algoritmo A* continuará siendo relevante y efectivo, reafirmando su posición como uno de los algoritmos de búsqueda de caminos más destacados y fiables.