UNIVERSIDAD DE GUADALAJARA
Red Universitaria e Institución Benemérita de Jalisco

CUCEI
CENTRO UNIVERSITARIO DE CIENCIAS EXACTAS E INGENIERÍAS

Inteligencia Artificial 1

Practica 2 Laberinto búsqueda en profundidad y profundidad iterativa

Veloz Alcaraz Axel Abraham

Espinoza Sucilla Samuel

Cornejo Chavez Edwin Joel | 220791268 | 07-11-2023

LUIS ANGEL BELTRAN CARRILLO

DIEGO ALBERTO OLIVA NAVARRO

Objetivo:

Considere el problema de encontrar el camino más corto entre dos puntos en un plano de dos dimensiones. Dentro del plano se encuentran diversos obstáculos.

Problema:

Definimos el laberinto usando una matriz de 10 x 10;

```
laberinto = [
    [0, 1, 1, 1, 1, 1, 1, 1, 1, 1],
    [0, 0, 1, 0, 0, 0, 0, 0, 0, 1],
    [1, 0, 0, 1, 1, 0, 1, 1, 0, 1],
    [1, 0, 1, 1, 0, 0, 1, 0, 0, 1],
    [1, 0, 0, 0, 1, 1, 1, 0, 1, 1],
    [1, 1, 1, 0, 1, 0, 1, 0, 1, 1],
    [1, 0, 1, 0, 0, 0, 1, 0, 0, 1],
    [1, 0, 1, 1, 1, 0, 1, 1, 0, 1],
    [1, 0, 0, 0, 1, 0, 0, 0, 0, 0],
    [1, 1, 1, 1, 1, 1, 1, 1, 1, 0]
]
```

Introducción:

El algoritmo de profundidad. comienza en un nodo de inicio y explora continuamente uno de sus hijos antes de retroceder. A medida que avanza, va profundizando en el árbol o grafo hasta encontrar la solución o llegar a un nodo sin hijos. Por otro lado, la profundidad iterativa es una variante del algoritmo de profundidad que se ajusta para evitar problemas de profundidad extrema al iterar sucesivamente con límites de profundidad crecientes, lo que garantiza que se explore todo el espacio de búsqueda en profundidad. Este enfoque combina la eficiencia del algoritmo de profundidad con la completitud del algoritmo de amplitud. Por ende estos algoritmos son muy útiles en la resolución de nuestro problema, es decir encontrar la solución del laberinto.

Desarrollo

```
# Función para moverse por el laberinto utilizando DFS
def moverse_en_laberinto(laberinto, inicio, fin):
    movimientos = [(1, 0), (-1, 0), (0, 1), (0, -1)]

    visitado = [[False for _ in range(len(laberinto[0]))] for _ in
range(len(laberinto))]
    pila = []
    pila.append(inicio)

    while pila:
        actual = pila.pop()
        if actual == fin:
```

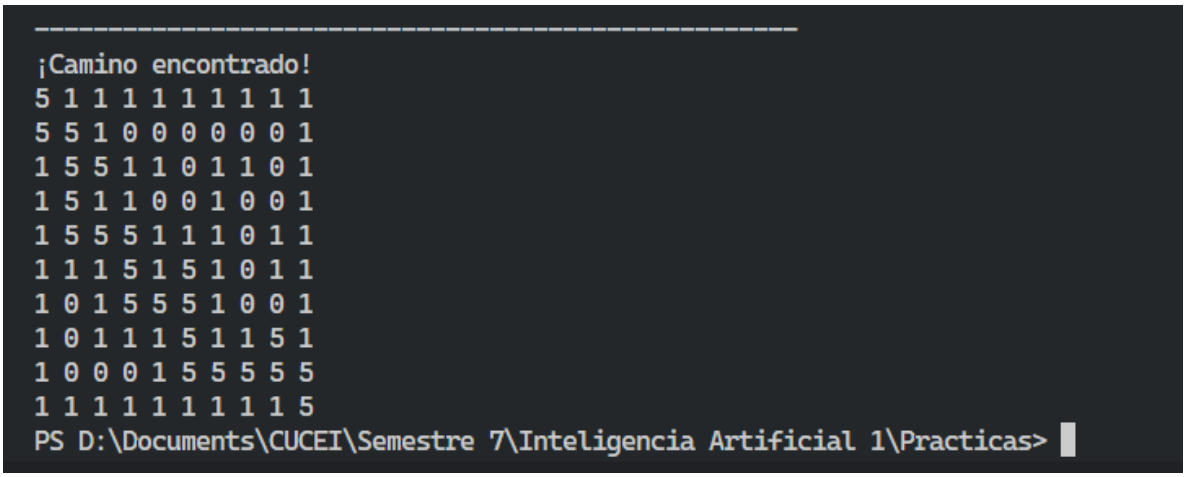
```
            return True

        x, y = actual
        for dx, dy in movimientos:
            nuevo_x, nuevo_y = x + dx, y + dy
            if 0 <= nuevo_x < len(laberinto) and 0 <= nuevo_y <
len(laberinto[0]) and not visitado[nuevo_x][nuevo_y] and
laberinto[nuevo_x][nuevo_y] == 0:
                pila.append((nuevo_x, nuevo_y))
                visitado[nuevo_x][nuevo_y] = True
                laberinto[nuevo_x][nuevo_y] = 5  # Movimiento del
número 5
                imprimir_laberinto(laberinto)
                print("-----------------------------------------------
----")
    return False
```

Solución

Inicio del programa, comienza a moverse en las casillas que están disponibles, es decir comienza a seguir el camino.

```
PS D:\Documents\CUCEI\Semestre 7\Inteligencia Artificial 1\Practicas> &
cia Artificial 1/Practicas/LaberintoProfundidad.py"
0 1 1 1 1 1 1 1 1 1
5 0 1 0 0 0 0 0 0 1
1 0 0 1 1 0 1 1 0 1
1 0 1 1 0 0 1 0 0 1
1 0 0 0 1 1 1 0 1 1
1 1 1 0 1 0 1 0 1 1
1 0 1 0 0 0 1 0 0 1
1 0 1 1 1 0 1 1 0 1
1 0 0 0 1 0 0 0 0 0
1 1 1 1 1 1 1 1 1 0
--------------------------------------------------
```

Al encontrar el camino nos da un mensaje y se muestra el recorrido que siguió para llegar a la solución.



Conclusión

En conclusión, se logró implementar el algoritmo de búsqueda en profundidad (DFS) de manera efectiva para navegar el laberinto en busca de un camino desde una posición de inicio hasta una posición de destino. El uso de una pila para rastrear las celdas a explorar y una matriz para llevar un registro de las visitas es un enfoque sólido. Además, la capacidad de marcar el camino recorrido con el número 5 facilita el seguimiento del progreso. En resumen, se obtuvo bases sólidas del algoritmo DFS para resolver un laberinto y es un buen punto de partida útil para aplicaciones de búsqueda en profundidad en problemas similares.