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 amplitud

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:

La búsqueda por amplitud, también conocida como búsqueda en anchura, es un algoritmo utilizado en informática y ciencias de la computación para explorar y buscar en estructuras de datos, como árboles o grafos, de manera sistemática. Su principal característica es que explora todos los nodos de un nivel antes de pasar al siguiente nivel en la estructura. Con esto establecido podemos comenzar a desarrollar el programa del laberinto usando dicho algoritmo para encontrar la solución a nuestro problema.

Desarrollo

```
# Función para moverse por el laberinto utilizando BFS
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))]
    cola = queue.Queue()
    cola.put(inicio)

    while not cola.empty():
        actual = cola.get()
        if actual == fin:
            return True
```

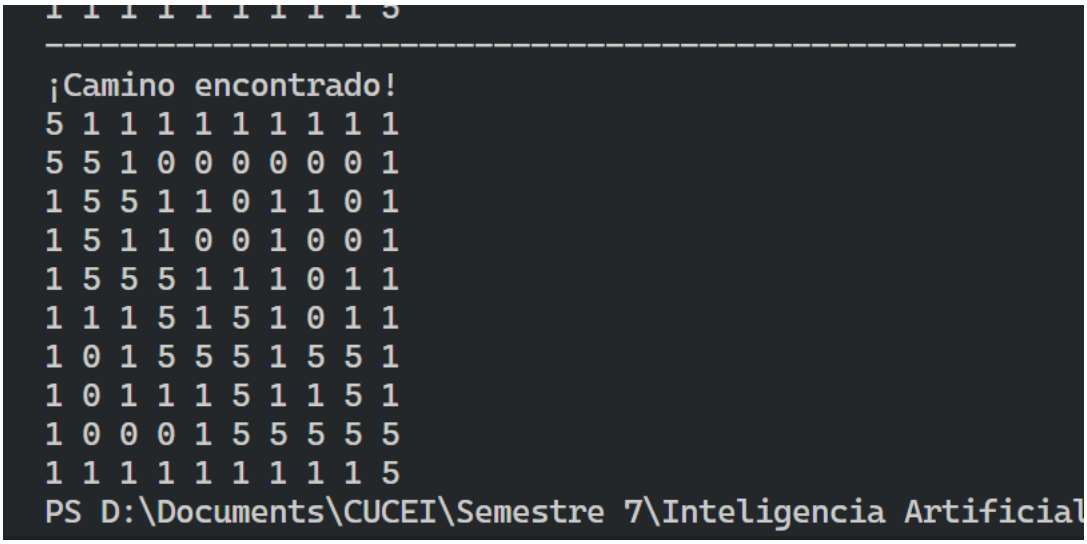
```
        x, y = actual
        for dx, dy in movimientos:
            nuevo_x, nuevo_y = x + dx, y + dy
            if 0 <= nuevo_x < len(laberinto) and 0 <= nuevo_y <
len(laberinto[0]) and not visitado[nuevo_x][nuevo_y] and
laberinto[nuevo_x][nuevo_y] == 0:
                cola.put((nuevo_x, nuevo_y))
                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> & C:/U
ments/CUCEI/Semestre 7/Inteligencia Artificial 1/Practicas/LaberintoAmplitud
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, el algoritmo de búsqueda por amplitud (BFS) para resolver el laberinto es una solución sólida y eficaz y a su vez garantiza que se encuentre la ruta más corta entre el punto de inicio y el punto de destino en el laberinto. Además, el marcado de nodos visitados y la representación gráfica del laberinto con valores de 5 para indicar el camino recorrido permiten un seguimiento visual del proceso de búsqueda. Entre las ventajas notables de este algoritmo se encuentra su capacidad para encontrar soluciones óptimas en términos de número de pasos, lo que lo hace especialmente útil en problemas de búsqueda de rutas y exploración de estructuras de datos.