Instituto Politécnico Nacional

Escuela Superior de Cómputo

Unidad de académica: Paradigmas de programación

Actividad: “Practica 9: Clase Robot y la ruta más corta”

Equipo:

• Elizalde Hernández Alan

• Reyes Ruíz Yoselyn Estefany

• Solares Velasco Arturo Misael

• Solís Lugo Mayra

• Toral Hernández Leonardo Javier

Grupo: 3CV1

Profesor: García Floriano Andrés

Fecha:

22 de mayo de 2024

El objetivo de esta practica es diseñar y desarrollar la clase Robot que implementa el método find_path() o buscar_camino() que:

- Buscará el camino mas corto desde la casilla [0,0] hasta la casilla [r, c] siempre y cuando exista ese camino.
- Algunas casillas (marcadas por un '1') no podrán ser accesibles por el robot y no podrá pasar por estas
- Implementarlo en lenguaje Python o Java

Sobre la solución

Buscar el camino

Solo valida la DFS si encuentra la ruta la devuelve, si no, manda el mensaje de ruta no encontrada

DFS

La búsqueda en profundidad (DFS, por sus siglas en inglés) es un algoritmo fundamental utilizado para recorrer grafos y árboles. La idea principal es explorar cada rama lo más profundamente posible antes de retroceder, lo que lo hace útil para resolver acertijos como laberintos, encontrar caminos y realizar búsquedas donde se debe explorar todo el espacio de soluciones.

Procedimiento

- Verificar si estamos fuera de los límites o en una celda bloqueada

```
    if not (0 <= row < self.rows and 0 <= col < self.cols) or
self.grid[row][col] == 1:
        return False
```

- Verificar si hemos llegado a la esquina inferior derecha

```
    if row == self.rows - 1 and col == self.cols - 1:
        self.path.append((row, col))
        return True
```

- Marcar la celda como parte de la ruta

```
    self.path.append((row, col))
```

```
    self.grid[row][col] = 1  # Marcar como visitado
```

- Intentar moverse a la derecha

```
    if self._dfs(row, col + 1):
        return True
```

- Intentar moverse hacia abajo

```
    if self._dfs(row + 1, col):
        return True
```

- Si ninguna dirección es válida, retroceder

```
    self.path.pop()
    self.grid[row][col] = 0  # Desmarcar la celda
    return False
```

Generar la rejilla

Genera una rejilla aleatoria de dimensiones r x c.

Requiere los parámetros filas y columnas y devuelve una lista de listas o una matriz cuadrada para representar la rejilla

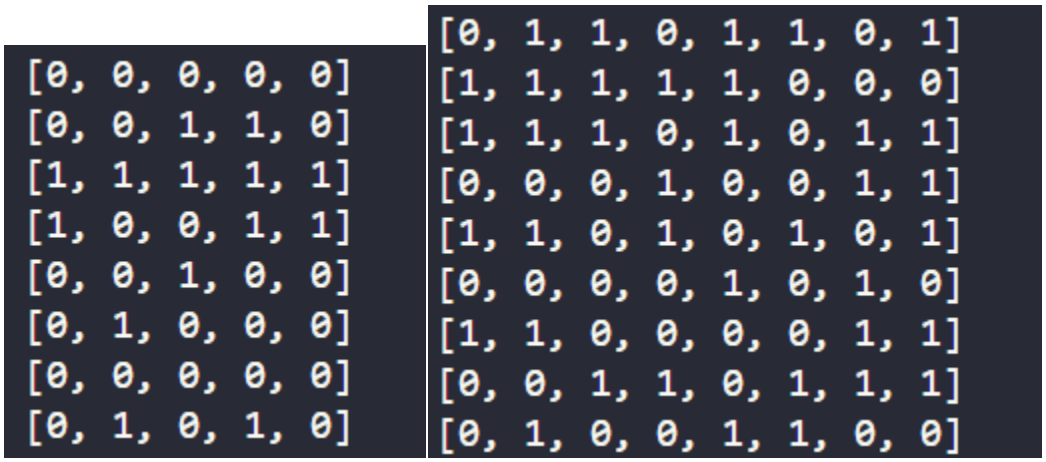
Siempre dejara disponibles los espacios de inicio y final para el robot

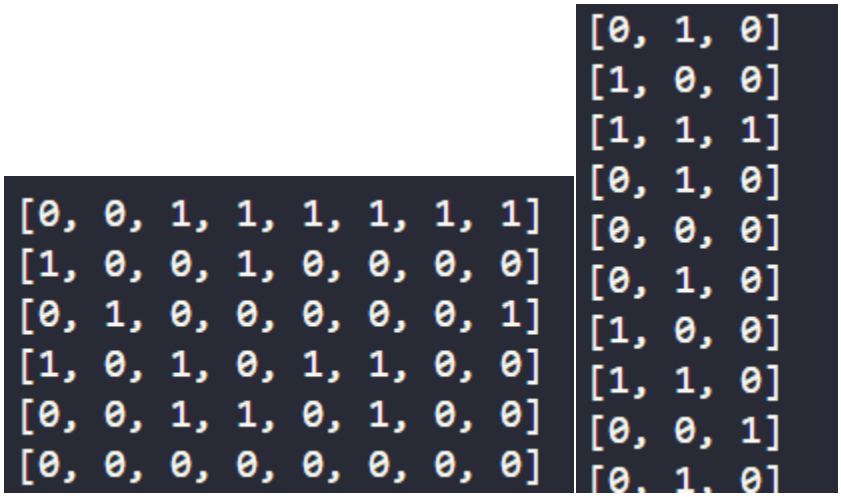
La funcion random permite colocar el valor 0 o 1 de manera aleatoria y sin distinción en la rejilla desde [0,0] hasta [r, c] y luego intercambia estas ultimas dos casillas por 0 representando que el robot puede estar en esos lugares

```
def generate_random_grid(r, c):
    grid = [[random.choice([0, 1]) for _ in range(c)] for _ in range(r)]
    grid[0][0] = 0  # Asegurar que el punto de inicio esté libre
    grid[r-1][c-1] = 0  # Asegurar que el punto de destino esté libre
    return grid
```

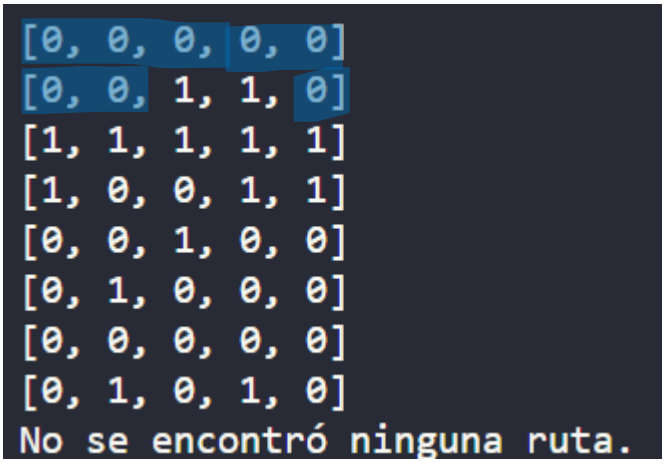
Pruebas de código

Generar la rejilla

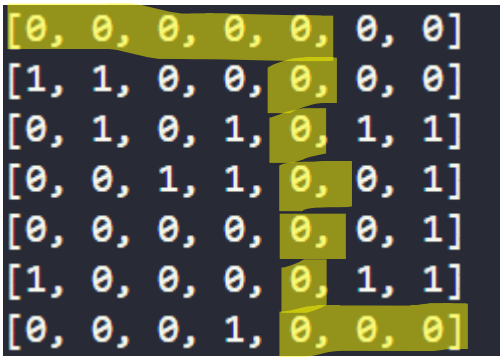


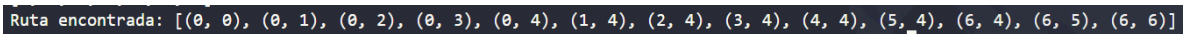


Buscar camino



En este primer caso no se encuentra la ruta porque no existe una forma posible de llegar a la esquina inferior derecha





En este segundo caso si es posible acceder a la esquina entonces el programa devuelve el camino más corto a través de coordenadas

Conclusión

Esta práctica permitió crear una clase especializada en la búsqueda de caminos a partir del algoritmo DFS, así como el poder utilizar los conocimientos de clase y juntarlos con otras materias, facilitando así el análisis y abstracción de conceptos para su aplicación. Este ejercicio fue útil para comprender la aplicación de conceptos de programación orientada a objetos, estructuras de datos y algoritmos.