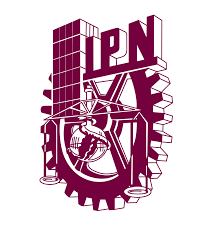
**Escuela Superior de Computo (ESCOM)**

**Instituto Politécnico Nacional**

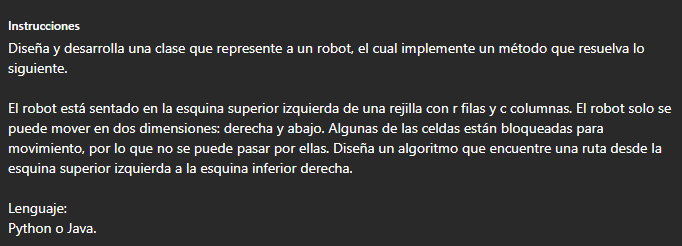
**Paradigmas de Programación**

Olivares Cruz Victor Manuel

**Práctica 9**

Fecha de entrega: 22/05/2024

**Instrucciones**



**Main.py**

from robot import Robot

def main():

    grid = [

        [0, 0, 1],

        [0, 1, 0],

        [0, 0, 0]

    ]

    robot = Robot(grid)

    path = robot.find\_path()

    if path:

        print("Ruta encontrada:", path)

    else:

        print("No hay ruta disponible.")

if \_\_name\_\_ == "\_\_main\_\_":

    main()

**Robot.py**

class Robot:

    def \_\_init\_\_(self, grid):

        self.grid = grid

        self.rows = len(grid)

        self.cols = len(grid[0])

        self.path = []

        self.visited = [[False] \* self.cols for \_ in range(self.rows)]

    def is\_valid(self, x, y):

        return 0 <= x < self.rows and 0 <= y < self.cols and not self.grid[x][y] and not self.visited[x][y]

    def find\_path(self):

        if self.dfs(0, 0):

            return self.path

        else:

            return None

    def dfs(self, x, y):

        if x == self.rows - 1 and y == self.cols - 1:

            self.path.append((x, y))

            return True

        if self.is\_valid(x, y):

            self.visited[x][y] = True

            self.path.append((x, y))

            # Moverse a la derecha

            if self.dfs(x, y + 1):

                return True

            # Moverse hacia abajo

            if self.dfs(x + 1, y):

                return True

            # Si no se encuentra una ruta, hacer backtrack

            self.path.pop()

            self.visited[x][y] = False

        return False

Métodos:

* **\_\_init\_\_(self, grid):**

Inicializa la rejilla, las dimensiones, la lista path y la matriz visited.

* **is\_valid(self, x, y):**

Verifica si una celda (x, y) es válida: dentro de los límites, no bloqueada y no visitada.

* **find\_path(self):**

Inicia la búsqueda de la ruta desde (0, 0).

Retorna self.path si se encuentra una ruta, None si no.

* dfs(self, x, y):

Implementa la búsqueda en profundidad recursiva.

Marca la celda como visitada y trata de moverse a la derecha y hacia abajo.

Hace backtracking si no encuentra una ruta y desmarca la celda.

**Prueba de ejecución:**

****