**BFS & DFS**

**Sayan Pramanick**

**RA1811030010023**

**Path Finding In A Maze With BFS**

Description:

**One of the common applications of breadth first search is to perform path finding. Typically this is done in a 2D maze. The code below implements the breadth first search algorithm to traverse and find the shortest path out of a maze.**

AIM:

**Perform path finding in a 2D maze.**

Code:

import queue

def createMaze():

    maze = []

    maze.append(["#", "#", "#", "#", "#", "O", "#"])

    maze.append(["#", " ", " ", " ", "#", " ", "#"])

    maze.append(["#", " ", "#", " ", "#", " ", "#"])

    maze.append(["#", " ", "#", " ", " ", " ", "#"])

    maze.append(["#", " ", "#", "#", "#", " ", "#"])

    maze.append(["#", " ", " ", " ", "#", " ", "#"])

    maze.append(["#", "#", "#", "#", "#", "X", "#"])

    return maze

def createMaze2():

    maze = []

    maze.append(["#", "#", "#", "#", "#", "O", "#", "#", "#"])

    maze.append(["#", " ", " ", " ", " ", " ", " ", " ", "#"])

    maze.append(["#", " ", "#", "#", " ", "#", "#", " ", "#"])

    maze.append(["#", " ", "#", " ", " ", " ", "#", " ", "#"])

    maze.append(["#", " ", "#", " ", "#", " ", "#", " ", "#"])

    maze.append(["#", " ", "#", " ", "#", " ", "#", " ", "#"])

    maze.append(["#", " ", "#", " ", "#", " ", "#", "#", "#"])

    maze.append(["#", " ", " ", " ", " ", " ", " ", " ", "#"])

    maze.append(["#", "#", "#", "#", "#", "#", "#", "X", "#"])

    return maze

def printMaze(maze, path=""):

    for x, pos in enumerate(maze[0]):

        if pos == "O":

            start = x

    i = start

    j = 0

    pos = set()

    for move in path:

        if move == "L":

            i -= 1

        elif move == "R":

            i += 1

        elif move == "U":

            j -= 1

        elif move == "D":

            j += 1

        pos.add((j, i))

    for j, row in enumerate(maze):

        for i, col in enumerate(row):

            if (j, i) in pos:

                print("+ ", end="")

            else:

                print(col + " ", end="")

        print()

def valid(maze, moves):

    for x, pos in enumerate(maze[0]):

        if pos == "O":

            start = x

    i = start

    j = 0

    for move in moves:

        if move == "L":

            i -= 1

        elif move == "R":

            i += 1

        elif move == "U":

            j -= 1

        elif move == "D":

            j += 1

        if not (0 <= i < len(maze[0]) and 0 <= j < len(maze)):

            return False

        elif (maze[j][i] == "#"):

            return False

    return True

def findEnd(maze, moves):

    for x, pos in enumerate(maze[0]):

        if pos == "O":

            start = x

    i = start

    j = 0

    for move in moves:

        if move == "L":

            i -= 1

        elif move == "R":

            i += 1

        elif move == "U":

            j -= 1

        elif move == "D":

            j += 1

    if maze[j][i] == "X":

        print("Found: " + moves)

        printMaze(maze, moves)

        return True

    return False

nums = queue.Queue()

nums.put("")

add = ""

maze = createMaze2()

while not findEnd(maze, add):

    add = nums.get()

    # print(add)

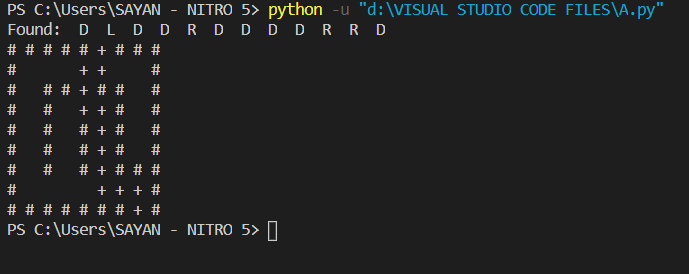
    for j in [" L ", " R ", " U ", " D "]:

        put = add + j

        if valid(maze, put):

            nums.put(put)

Output:



Result:

**The path is found and the directions are shown accordingly in the output with an additional graphical representation.**