You are given a 2D grid representing a maze, where each cell is either an empty space (0) or a wall (1). Your task is to implement a Python program that uses Iterative Deepening Depth-First Search (IDDFS) to determine whether a valid path exists from a given start cell to a specified target cell. You may move up, down, left, or right to adjacent empty cells, but you cannot pass through walls, and each cell may be visited only once during a single path exploration.

Case#1Input:

4 4

0 0 1 0

1 0 1 0

0 0 0 0

1 1 0 1

Start: 0 0

Target: 2 3

Case#1Output:

Path found at depth 5 using IDDFS

Traversal Order: [(0,0), (1,0), (1,1), (0,1), (2,1), (2,2), (2,3)]

Case#2Input:

3 3

0 1 0

0 1 0

0 1 0

Start: 0 0

Target: 2 2

Case#2Output:

Path not found at max depth 6 using IDDFS

Code:  
def main():

# Read input

rows, cols = map(int, input().split())

maze = []

for \_ in range(rows):

maze.append(list(map(int, input().split())))

start = tuple(map(int, input().split()))

target = tuple(map(int, input().split()))

# Find path using IDDFS

found, path, traversal\_order = iddfs(maze, start, target)

# Output results

if found:

depth = len(path) - 1

print(f"Path found at depth {depth} using IDDFS")

print(f"Traversal Order: {traversal\_order}")

else:

max\_depth = rows \* cols # Worst case is all cells

print(f"Path not found at max depth {max\_depth} using IDDFS")

def iddfs(maze, start, target):

max\_depth = len(maze) \* len(maze[0]) # Total cells in maze

for depth in range(1, max\_depth + 1):

visited = set()

path = []

traversal\_order = []

found = dls(maze, start, target, depth, visited, path, traversal\_order)

if found:

return True, path, traversal\_order

return False, [], []

def dls(maze, current, target, depth, visited, path, traversal\_order):

if depth == 0 and current == target:

path.append(current)

return True

if depth > 0:

x, y = current

rows = len(maze)

cols = len(maze[0])

# Mark current as visited and add to path

visited.add(current)

traversal\_order.append(current)

path.append(current)

# Check all 4 directions

directions = [(-1, 0), (1, 0), (0, -1), (0, 1)]

for dx, dy in directions:

nx, ny = x + dx, y + dy

if (0 <= nx < rows and 0 <= ny < cols and

maze[nx][ny] == 0 and (nx, ny) not in visited):

if dls(maze, (nx, ny), target, depth - 1, visited, path, traversal\_order):

return True

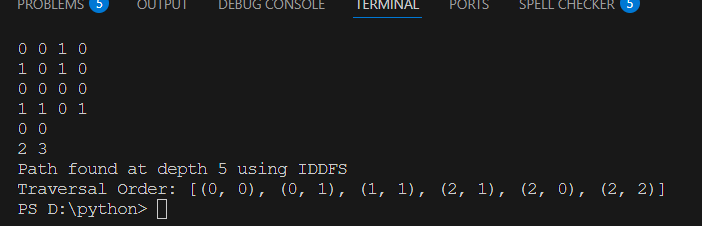
# Backtrack

path.pop()

return False

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

main()

Output 01:  


Output 02:  
