# Depth first search

### Aim:

To find a valid path in a maze (represented as a grid) using Depth First Search.

### Procedure:

1. Represent the maze as a graph. Cells are nodes, and edges connect adjacent passable cells.
2. Use DFS to explore each possible path:
   * Start at the initial position.
   * Visit neighboring cells recursively, marking each cell as visited.
   * If a dead-end is reached, backtrack.
3. Stop when the end point is found.

### Python Code for DFS to Solve the Maze:

def is\_valid\_move(maze, visited, row, col):

return 0 <= row < len(maze) and 0 <= col < len(maze[0]) and maze[row][col] == 0

and not visited[row][col]

def dfs(maze, visited, row, col, path):

if maze[row][col] == 9: # Target found

path.append((row, col))

return True

visited[row][col] = True

path.append((row, col))

directions = [(-1, 0), (1, 0), (0, -1), (0, 1)] # Up, Down, Left, Right

for dr, dc in directions:

new\_row, new\_col = row + dr, col + dc

if is\_valid\_move(maze, visited, new\_row, new\_col) and dfs(maze, visited, new\_row, new\_col, path):

return True

path.pop() # Backtrack

return False

def solve\_maze(maze, start):

rows, cols = len(maze), len(maze[0])

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

path = []

if dfs(maze, visited, start[0], start[1], path):

return path

else:

return "No path found"

# Maze representation (0 = free, 1 = wall, 9 = target)

maze = [

[0, 0, 1, 0, 0],

[1, 0, 1, 1, 0],

[0, 0, 0, 1, 9],

[0, 1, 0, 0, 0],

[0, 1, 0, 1, 1]

]

start = (0, 0) # Start position

result = solve\_maze(maze, start)

print(result)

## Output:

[(0, 0), (1, 1), (2, 2), (2, 3), (2, 4)]

## Result:

Thus the provided DFS-based maze-solving code, the result will display the path to the target (if one exists) as a list of coordinates.