BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI WORK INTEGRATED LEARNING PROGRAM DIVISION

DSECL ZG557, Artificial and Computational Intelligence
LAB REFERENCE SHEET MODULE 1

Exercise 1: Implement Uninformed Search

1.A) Breadth First Search (BFS) and Depth First Search (DFS)

Tool: Python

Libraries Used: queue

Sample Problem:

A binary maze is an n*m matrix where,

maze[i][j] = 1 represent traversable blocks

maze[i][j] = 0 represent obstacles

Given such a binary maze with obstacles and traversable blocks, find the shortest path between a source cell and destination cell. Permissible moves are up, down, left and right.

Input Type: Binary matrix, source indices, destination indices

Input Binary Matrix:

```
inputMaze = [[1, 0, 0, 0],
             [1, 1, 0, 1],
             [0, 1, 0, 0],
             [1, 1, 1, 1]]
src = [0, 0]
dest = [3, 3]
```

Logic / Search Technique: Breadth First Search

Output Type: Integer representing shortest path

Implementation

```
import queue

#These two lists help in getting coordinates of the adjacent 4 cells
rowNums = [0, -1, 0, 1]
colNums = [-1, 0, 1, 0]
```

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI WORK INTEGRATED LEARNING PROGRAM DIVISION

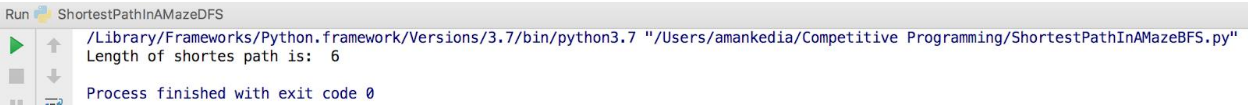
DSECL ZG557, Artificial and Computational Intelligence

LAB REFERENCE SHEET MODULE 1

```
class Point():

    # Objects of this class would be used to define coordinates in the matrix

    def __init__(self, x, y):
        self.x = x
        self.y = y

class Node():

    #Objects of this class would be used to define every entry in the matrix
    #including its coordinates and distance from source

    def __init__(self, coordinates, dist):
        self.coordinates = coordinates
        self.dist = dist

def isSafe(x, y, nrow, ncol):

    #This method would help to evaluate if a pair of coordinates is valid or not

    if x >= 0 and x < nrow and y >= 0 and y < ncol:
        return True
    return False


def shortestPathInAMaze(maze, src, dest):

    nrow = len(maze)
    ncol = len(maze[0])

    val = BFS(maze, src, dest, nrow, ncol)

    return val


def BFS(maze, src, dest, nrow, ncol):

    visited = []

    # if src or destination is an obstacle, we cannot have a path
    if maze[src.x][src.y] is not 1 or maze[dest.x][dest.y] is not 1:
        return -1

    for i in range(len(maze)):
        visited.append([False] * len(maze[i]))

    #Mark the source as visited
    visited[src.x][src.y] = True

    q = queue.Queue(maxsize = (nrow*ncol))

    #Add source to queue
    q.put(Node(src, 0))

    while not q.empty():
        current = q.get()
        point = current.coordinates

        #If coordinates of cuurent node are same as destination, the goal has been 
reached
        if point.x == dest.x and point.y == dest.y:
            return current.dist

        for i in range(0, 4):
            row = point.x + rowNums[i]
            col = point.y + colNums[i]

            #add the adjacent node to queue if it is a valid coordinate, it is not an 
obstacle and has not been visited yet
            if isSafe(row, col, nrow, ncol) and maze[row][col] is not 0 and 
visited[row][col] is False:
                visited[row][col] = True
                newNode = Node(Point(row, col), current.dist + 1)
                q.put(newNode)

    #If a path has not been found then return -1
    return -1


if __name__ == "__main__":

    inputMaze = [[1, 0, 0, 0]
                 [1, 1, 0, 1],
                 [0, 1, 0, 0],
                 [1, 1, 1, 1]]

    src  = [0, 0]
    dest = [3, 3]

    srcObject = Point(src[0], src[1])
    destObject = Point(dest[0], dest[1])

    val = shortestPathInAMaze(inputMaze, srcObject, destObject)

    if val == -1:
        print("Path does not exist")
    else:
        print("Length of shortes path is: ", val)
```

Output: 6

Screenshot



```
/Library/Frameworks/Python.framework/Versions/3.7/bin/python3.7 "/Users/amankedia/Competitive Programming/ShortestPathInAMazeBFS.py"
Length of shortes path is:  6

Process finished with exit code 0
```

Logic / Search Technique: Depth First Search

Output Type: Integer representing shortest path

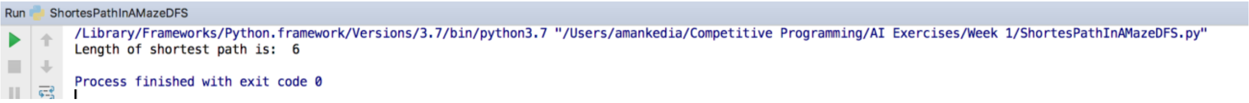
Implementation (Depth First Search):

```
#These two lists help in getting coordinates of the adjacent 4 cells
rowNums = [0, -1, 0, 1]
colNums = [-1, 0, 1, 0]

class Point():

    # Objects of this class would be used to define coordinates in the matrix

    def __init__(self, x, y):
        self.x = x
        self.y = y

class Node():

    #Objects of this class would be used to define every entry in the matrix
    #including it's coordinates and distance from source

    def __init__(self, coordinates, dist):
        self.coordinates = coordinates
        self.dist = dist

def isSafe(x, y, nrow, ncol):

    #This method would help to evaluate if a pair of coordinates is valid or not

    if x >= 0 and x < nrow and y >= 0 and y < ncol:
        return True
    return False


def shortestPathInAMaze(maze, src, dest):

    nrow = len(maze)
    ncol = len(maze[0])

    val = DFS(maze, src, dest, nrow, ncol)

    return val


def DFS(maze, src, dest, nrow, ncol):
    visited = []

    # if src or destination is an obstacle, we cannot have a path
    if maze[src.x][src.y] is not 1 or maze[dest.x][dest.y] is not 1:
        return -1

    for i in range(len(maze)):
        visited.append([False] * len(maze[i]))

    #Mark the source as visited
    visited[src.x][src.y] = True

    stack = []*(nrow*ncol)

    #Add source to stack
    stack.append(Node(src, 0))

    while not len(stack) == 0:
        current = stack.pop()

        point = current.coordinates

        #If coordinates of curent node are same as destination, the goal has been 
reached
        if point.x == dest.x and point.y == dest.y:
            return current.dist

        for i in range(0, 4):
            row = point.x + rowNums[i]
            col = point.y + colNums[i]

            #add the adjacent node to queue if it is a valid coordinate, it is not an 
obstacle and has not been visited yet
            if isSafe(row, col, nrow, ncol) and maze[row][col] is not 0 and 
visited[row][col] is False:
                visited[row][col] = True
                newNode = Node(Point(row, col), current.dist + 1)
                stack.append(newNode)

    #If a path has not been found then return -1
    return -1


if __name__ == "__main__":

    inputMaze = [[1, 1, 0, 0],
                 [1, 1, 0, 1],
                 [0, 1, 0, 0],
                 [1, 1, 1, 1]]

    src  = [0, 0]
    dest = [3, 3]

    srcObject = Point(src[0], src[1])
    destObject = Point(dest[0], dest[1])

    val = shortestPathInAMaze(inputMaze, srcObject, destObject)
```

```
if val == -1:
    print("Path does not exist")
else:
    print("Length of shortest path is: ", val)
```

Output: 6

Screenshot



```
/Library/Frameworks/Python.framework/Versions/3.7/bin/python3.7 "/Users/amankedia/Competitive Programming/AI Exercises/Week 1/ShortesPathInAMazeDFS.py"
Length of shortest path is:  6

Process finished with exit code 0
```

Lab Exercise (TODO)

1. **Modify the code and find the path which leads from source to destination for both BFS and DFS.**
2. **Modify the code and find all the nodes visited for both BFS and DFS.**
3. **Modify the code and find all the nodes added to queue or stack for both BFS and DFS respectively.**