Green University of Bangladesh
Department of Computer Science and Engineering (CSE)

Faculty of Sciences and Engineering

Semester: (Spring, Year:2025), B.Sc. in CSE (Day)

LabPerformance 03: BFS

Course Title: Artificial Intelligence Lab

Course Code: CSE-316 Section:221-14

Student Details

Name		ID
1.	Md.Mohibullah	221902083

Lab Date : 11-02-2025

Submission Date : 12-02-2025

Course Teacher's Name : Md. Sabbir Hosen Mamun

Lab Report Status

Marks: …… Signature:.....

Comments:..... Date:.....

Code:

```
import random
from collections import deque

class Node:
    def __init__(self,x,y, level):
        self.x = x
        self.y = y
        self.level = level

class BFS:
    def __init__(self):
        self.directions = [(1,0),(-1,0),(0,1),(0,-1)]
        self.found=False
        self.maze = None
        self.start = None
        self.end = None

def init(self, graph, source_x, source_y, goal_x, goal_y):
        self.N = len(graph)
        self.source = Node(source_x, source_y, 0)
        self.goal = Node(goal_x, goal_y, float('inf'))
        self.st_bfs(graph)
        if self.found:
            print("Goal found")
            print("Number of moves =", self.goal_level)
        else:
            print("Goal cannot be reached.")
def st_bfs(self, graph):
        queue = deque()
        queue.append(self.source)
        while queue:
            u = queue.popleft()
            for dx, dy in self.directions:
                v_x, v_y = u.x + dx, u.y + dy
                if 0 <= v_x < self.N and 0 <= v_y < self.N and
graph[v_x][v_y] == 0:
                    v_level = u.level + 1
                    if v_x == self.goal.x and v_y == self.goal.y:
                        self.found = True
```

```
                        self.goal_level = v_level
                        self.goal.level = v_level
                        return
                    graph[v_x][v_y] = 1
                    child = Node(v_x, v_y, v_level)
                    queue.append(child)

def generate_random_grid(N, obstacle_probability=0.3):
    grid = [[0 for _ in range(N)] for _ in range(N)]
    for i in range(N):
        for j in range(N):
            if random.random() < obstacle_probability:
                grid[i][j] = 1
    return grid

def is_valid_position(x, y, N, graph):
    return 0 <= x < N and 0 <= y < N and graph[x][y] == 0

if __name__ == "__main__":
    N = int(input("Enter grid size : "))
    graph = generate_random_grid(N)
    print("Generated Grid ")
    for row in graph:
        print(" ".join(str(cell) for cell in row))
    while True:
        source_x, source_y = map(int, input(f"Enter start position between
0 and {N-1}: ").split())
        if is_valid_position(source_x, source_y, N, graph):
            break
        else:
            print("Invalid start position!")
    while True:
        goal_x, goal_y = map(int, input(f"Enter goal position between 0
and {N-1}: ").split())
        if is_valid_position(goal_x, goal_y, N, graph):
            break
        else:
            print("Invalid goal position!")
    bfs = BFS()
    bfs.init(graph, source_x, source_y, goal_x, goal_y)
```