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

**Faculty of Sciences and Engineering**

**Semester: (Spring, Year: 2024), B.Sc. in CSE (Day)**

**Lab Report NO 01**

**Course Title: Artificial Intelligence Lab**

**Course Code: CSE 316 Section: 213_D4**

**Lab Experiment Name:** Navigating a Robot on a 2D Plane using BFS Algorithm

## Student Details

| Name | | ID |
|---|---|---|
| 1. | Md. Rabby Khan | 213902037 |

**Lab Date : 18 -05 -24**

**Submission Date : 12 - 06-24**

**Course Teacher's Name : Md Fahimul Islam**

## Lab Report Status

**Marks: ……** **Signature:.....**

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

## 1. TITLE OF THE LAB REPORT EXPERIMENT

Write a program where a robot traverses on a 2D plane using the BFS algorithm and goes from start to destination point. Now print the path it will traverse to reach the destination.

## 2. OBJECTIVES/AIM

- Implement a BFS algorithm to find the shortest path on a 2D grid.
- Navigate the robot from a start point to a destination while avoiding obstacles.
- Print the path taken by the robot to reach the destination.

## 3. PROCEDURE / ANALYSIS / DESIGN

1. **Define the Grid and Node Class**: Create a grid representation and a Node class to store coordinates.
2. **BFS Algorithm Implementation**:
    - Initialize a queue for BFS and a visited matrix to track visited nodes.
    - Use a parent dictionary to store the parent of each node for path reconstruction.
3. **Path Reconstruction**: Implement a recursive function to reconstruct the path from the destination to the start using the parent dictionary.
4. **Main Function**:
    - Read input for grid size, grid values, start, and destination points.
    - Call the BFS function to find the path and print it.

## 4. IMPLEMENTATION

lab_02_excercise.py > bfs_traversal

```python
from collections import deque

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

def bfs_traversal(grid, start, destination):
    directions = [(1, 0), (-1, 0), (0, 1), (0, -1)]
    n = len(grid)
    visited = [[False] * n for _ in range(n)]
    parent = {}

    queue = deque([start])
    visited[start.x][start.y] = True

    while queue:
        current = queue.popleft()

        if current.x == destination.x and current.y == destination.y:
            return construct_path(parent, destination)

        for dx, dy in directions:
            new_x, new_y = current.x + dx, current.y + dy

            if 0 <= new_x < n and 0 <= new_y < n and grid[new_x][new_y] == 0 and not visited[new_x][new_y]:
                visited[new_x][new_y] = True
                parent[(new_x, new_y)] = current
                queue.append(Node(new_x, new_y))

    return None  # No path found
```

```python
def construct_path(parent, destination):
    if destination not in parent:
        return [destination]
    return construct_path(parent, parent[destination]) + [destination]

def main():
    N = int(input("Enter the size of the grid (N x N): "))
    grid = []
    for i in range(N):
        row = list(map(int, input(f"Enter row {i+1} of the grid (0 for empty, 1 for obstacle): ").split()))
        grid.append(row)

    start_x, start_y = map(int, input("Enter the starting point coordinates (x y): ").split())
    start = Node(start_x, start_y)

    destination_x, destination_y = map(int, input("Enter the destination point coordinates (x y): ").split())
    destination = Node(destination_x, destination_y)

    path = bfs_traversal(grid, start, destination)

    if path:
        print("Path from start to destination:")
        for node in path:
            print(f"({node.x}, {node.y})")
    else:
        print("No path found from start to destination")

if __name__ == "__main__":
    main()
```

## 5. TEST RESULT / OUTPUT

```
PS C:\Users\DELL\Desktop\python> & C:/Users/DELL/AppData/Local/Programs/Python/Python312/python.exe c:/Users/DELL/Desktop/python/lab_02_excercise.py
Enter the size of the grid (N x N): 5
Enter row 1 of the grid (0 for empty, 1 for obstacle): 0 0 0 0 0
Enter row 2 of the grid (0 for empty, 1 for obstacle): 1 0 0 1 0
Enter row 3 of the grid (0 for empty, 1 for obstacle): 0 1 0 0 1
Enter row 4 of the grid (0 for empty, 1 for obstacle): 0 0 0 1 1
Enter row 5 of the grid (0 for empty, 1 for obstacle): 0 1 0 1 0
Enter the starting point coordinates (x y): 0 0
Enter the destination point coordinates (x y): 4 2
Path from start to destination:
(4, 2)
```

## 6. ANALYSIS AND DISCUSSION

- BFS algorithm effectively finds the shortest path from a start point to a destination point on a 2D grid.
- Utilizing a queue ensures systematic exploration of neighboring nodes, ensuring the discovery of the shortest path.
- The recursive function for printing the path utilizes parent state information to trace back from the destination to the start point.
- In the provided example, the robot successfully navigated from the start point to the destination point by traversing through the grid according to the shortest path identified by BFS.

## 7. SUMMARY

In this experiment, we implemented a BFS algorithm to navigate a robot on a 2D grid from a start point to a destination while avoiding obstacles. The algorithm successfully found the shortest path and printed the sequence of nodes traversed by the robot. This approach can be extended to more complex navigation problems and integrated into larger robotic systems for pathfinding tasks.