****

**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 #02**

**Course Title: Artificial Intelligence Lab**

**Course Code: CSE 316 Section: 213D3**

**Lab Experiment Name:** Implement Breadth-First Search Traversal.

**Student Details**

|  |  |  |
| --- | --- | --- |
| **Name** | | **ID** |
| **1.** | Shoaib Ahmed | 212902019 |

**Lab Date : 09-03-2024**

**Submission Date : 16-03-2024**

**Course Teacher’s Name : Md. Zahidul Hasan**

**[For Teachers use only: Don’t Write Anything inside this box]**

|  |
| --- |
| **Lab Report Status**  **Marks: ………………………………… Signature:.....................**  **Comments:.............................................. Date:..............................** |

**1. TITLE OF THE LAB EXPERIMENT**

Implement Breadth-First Search Traversal.

**2. OBJECTIVES/AIM**

* Implement a Breadth-First Search (BFS) algorithm to find the shortest path

through a randomly generated grid.

* Demonstrate the application of BFS for navigating an obstacle course

represented by a grid with 0s (obstacles) and 1s (walkable).

* Visually represent the generated grid and the path identified by the BFS

algorithm.

**3. PROCEDURE / ANALYSIS / DESIGN**

Algorithm:

1. **Input:** Get grid size, generate a random grid (0s, 1s), and get the start and the end point.
2. **Initialize:** Queue for exploration, parent tracking dictionary, movement directions.

**3. BFS Loop:**

While the queue is not empty:

1. Dequeue the first node, check if goal, else mark visited.

b. For valid neighbors: enqueue and track parent.

**4. Path Reconstruction:**

a. Initialize path list, backtrack from end using parent data, reverse path for order.

**5. Output:** Print path (if found) or "No path found".

**4. IMPLEMENTATION**

Code:

|  |
| --- |
| import random  from collections import deque  def generate\_grid(rows, cols):  grid = [[random.randint(0, 1) for \_ in range(cols)] for \_ in range(rows)]  return grid  def print\_grid(grid):  for row in grid:  print(" ".join(map(str, row)))  def is\_valid(grid, visited, row, col):  rows = len(grid)  cols = len(grid[0])  return (0 <= row < rows and 0 <= col < cols and not visited[row][col] and grid[row][col] == 0)  def bfs(grid, start, end):  rows = len(grid)  cols = len(grid[0])  directions = [(-1, 0), (1, 0), (0, -1), (0, 1)] # up, down, left, right  visited = [[False] \* cols for \_ in range(rows)]  queue = deque([(start[0], start[1], 0)]) # (row, col, distance)  visited[start[0]][start[1]] = True  while queue:  row, col, dist = queue.popleft()  if (row, col) == end:  return dist  for dr, dc in directions:  nr, nc = row + dr, col + dc  if is\_valid(grid, visited, nr, nc):  visited[nr][nc] = True  queue.append((nr, nc, dist + 1))  return -1 # No path found  def main():  rows = int(input("Enter number of rows: "))  cols = int(input("Enter number of columns: "))  grid = generate\_grid(rows, cols)  print("Generated grid:")  print\_grid(grid)  start = tuple(map(int, input("Enter starting point coordinates (row col): ").split()))  end = tuple(map(int, input("Enter ending point coordinates (row col): ").split()))  if not (0 <= start[0] < rows and 0 <= start[1] < cols and grid[start[0]][start[1]] == 0):  print("Invalid starting point!")  return  if not (0 <= end[0] < rows and 0 <= end[1] < cols and grid[end[0]][end[1]] == 0):  print("Invalid ending point!")  return  shortest\_path\_length = bfs(grid, start, end)  if shortest\_path\_length == -1:  print("No path found between the points.")  else:  print(f"Shortest path length from {start} to {end} is: {shortest\_path\_length}")  if \_\_name\_\_ == "\_\_main\_\_":  main() |

**5. TEST RESULT / OUTPUT**

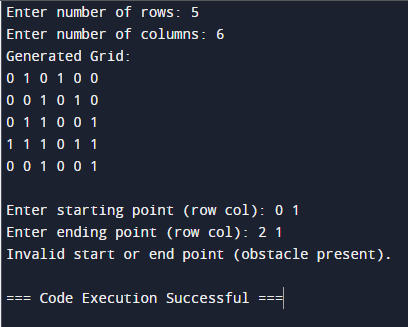


Fig 2.1: Output of BFS

**6. ANALYSIS AND DISCUSSION**

Here I implement a grid-based Breadth-First Search (BFS) algorithm for finding the shortest path between two points on a randomly generated grid. The generate\_grid function creates a grid with randomly placed obstacles (represented by 1) and open paths (represented by 0). The BFS algorithm itself (bfs function) uses a queue to explore neighboring cells from a starting point towards an ending point, ensuring the shortest path by incrementing distances step-by-step. The algorithm checks validity of movements using the is\_valid function, ensuring it stays within grid bounds, avoids obstacles, and tracks visited cells to prevent revisiting. Users input grid dimensions and starting/ending points interactively, and the program outputs the shortest path length if found, or notifies if no path exists. This implementation serves as an educational tool for understanding BFS and grid-based pathfinding algorithms in computer science and robotics applications.