Green University of Bangladesh

Department of Computer Science and Engineering (CSE)
Faculty of Sciences and Engineering
Spring 2024, B.Sc. in CSE (DAY)

Lab Report NO # 02

Course Title: Artificial Intelligence Lab
Course Code: CSE 316 Section: CSE 213 – D1

Problems / Tasks / Domains:

- 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.
  (***Hint: Save parent state information and use that information to write a recursive function for printing the path.***)

Student Details

Name	ID
Md. Shahidul Islam Prodhan	**213902017**

Lab Assigned Date: 15th March 2024

Submission Date: 17th March 2024

Course Teacher's Name: Fairuz Shaiara, Lecturer

[For Teacher's use only: Don't write anything inside this box]

Lab Report Status

Marks:	**Signature:**
Comments:	**Date:**

1. TITLE OF THE LAB REPORT EXPERIMENT

Implement Breadth-First Search Traversal.

2. OBJECTIVES/AIM

The primary objectives of this lab is to understand how to represent states and nodes in a graph and understand how Bread-First Search (BFS) works on a two-dimensional (2D) plane.

3. PROCEDURE / ANALYSIS / DESIGN

- Problem Description:

The problem requires implementing a robot's traversal on a 2D plane using the ***BFS algorithm*** to reach the destination point while printing the path traversed.

- Approach Overview:

The approach involves defining a grid representing the 2D plane where the robot can move. A BFS algorithm is used to explore possible paths from the start to the destination point. Parent state information is saved during traversal to backtrack and determine the path taken.

- Implementation Key Points:

The program defines ***directions*** as ***tuples*** representing *movement (Up, Down, Left, Right)* and their corresponding names. A ***bfs function*** performs BFS traversal from start to destination, saving the path traversed. ***A print path*** function utilizes parent state information to print the path taken by the robot.

- Design Considerations:

The design ensures readability and modularity by encapsulating ***BFS traversal*** and path printing into separate functions. Use of ***deque*** for the ***queue*** in BFS ensures efficient exploration of paths. The code follows a clear and concise structure, enhancing its comprehensibility.

- Analysis:

The BFS algorithm guarantees finding the shortest path to the destination point in a 2D grid. By saving parent state information, the program efficiently reconstructs the path traversed by the robot. The design allows for easy extension or modification to accommodate variations in the problem requirements or grid structures.

4. IMPLEMENTATION

1.Imports:

- **from collections import deque**: This import is used to utilize the **deque** data structure, which is an efficient double-ended queue implementation used in BFS traversal.

2.Global Variables:

- **directions**: This list contains tuples representing the possible directions the robot can move in the 2D plane. Each tuple corresponds to a movement: Up, Down, Left, Right.
- **direction_names**: This list contains the names of the directions corresponding to the indices of the **directions** list.

3.Helper Functions:

- **is_valid_move(grid, row, col)**: This function checks whether a move to the given row and column is valid, i.e., within the boundaries of the grid and not obstructed by an obstacle.
- **bfs(grid, start, dest)**: This function performs a breadth-first search traversal from the start point to the destination point on the grid. It returns the path taken by the robot from start to destination.
- **print_path(path)**: This function prints the path taken by the robot, given the list of coordinates representing the path.

4.BFS Function (**bfs**):

- It initializes a queue with the starting position and an empty path list.
- While the queue is not empty:
  - Dequeue a position **(row, col)** and its corresponding path from the queue.
  - If the position is the destination, return the path.
  - Otherwise, mark the position as visited and enqueue neighboring positions that are valid moves.
- If the destination is not reached, return **None**.

5.Print Path Function (**print_path**):

- It prints the path taken by the robot by iterating over the list of coordinates in the path.
- For each coordinate, it prints whether it's the start, end, or a move from one position to another.

6.Grid Definition and Example Parameters:

- **grid**: This variable represents the 2D grid where the robot can move. It contains 0s for empty spaces and 1s for obstacles.
- **start** and **destination**: These variables define the starting and destination points for the robot's traversal.

7.Main Execution:

- It calls the **bfs** function to find the path from the start to the destination on the given grid.
- It then calls the **print_path** function to print the path taken by the robot.

4. IMPLEMENTATION

```python
from collections import deque

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


def is_valid_move(grid, row, col):
    return 0 <= row < len(grid) and 0 <= col < len(
        grid[0]) and grid[row][col] == 0


def bfs(grid, start, dest):
    queue = deque([(start, [])])
    visited = set()

    while queue:
        (row, col), path = queue.popleft()

        if (row, col) == dest:
            return path + [(row, col)]

        if (row, col) in visited:
            continue

        visited.add((row, col))

        for i, (dr, dc) in enumerate(directions):
            new_row, new_col = row + dr, col + dc

            if is_valid_move(grid, new_row, new_col):
                queue.append(((new_row, new_col), path + [(row, col)]))
```

```python
    return None


def print_path(path):
    if not path:
        print("No path found.")
        return

    print("Path from start to destination:")        #shahidul_213902017
    for i, (row, col) in enumerate(path):
        if i == 0:
            print(f"Start at ({row}, {col})")
        elif i == len(path) - 1:
            print(f"End at ({row}, {col})")
        else:
            print(
                f"Move {direction_names[path[i][0] - path[i-1][0] + 1]} from ({path[i-1][0]}, {path[i-1][1]}) to ({row}, {col})"
            )


# 0 is empty space, 1 is obstacles

grid = [[0, 0, 0, 0, 0], [1, 1, 1, 1, 0], [0, 0, 0, 0, 0], [0, 1, 1, 1, 1],
        [0, 0, 0, 0, 0]]

start = (0, 0)
destination = (4, 4)

path = bfs(grid, start, destination)
print_path(path)
```

5. TEST RESULT / OUTPUT

```
Run

Console   Shell

Packager

--> poetry lock --no-update
Resolving dependencies...

Run

Path from start to destination:
Start at (0, 0)
Move Down from (0, 0) to (0, 1)
Move Down from (0, 1) to (0, 2)
Move Down from (0, 2) to (0, 3)
Move Down from (0, 3) to (0, 4)
Move Left from (0, 4) to (1, 4)
Move Left from (1, 4) to (2, 4)
Move Down from (2, 4) to (2, 3)
Move Down from (2, 3) to (2, 2)
Move Down from (2, 2) to (2, 1)
Move Down from (2, 1) to (2, 0)
Move Left from (2, 0) to (3, 0)
Move Left from (3, 0) to (4, 0)
Move Down from (4, 0) to (4, 1)
Move Down from (4, 1) to (4, 2)
Move Down from (4, 2) to (4, 3)
End at (4, 4)
```

6. ANALYSIS AND DISCUSSION

- Efficiency of BFS:
  The BFS algorithm ensures optimal pathfinding in a grid-based environment, guaranteeing the shortest path to the destination point. It explores neighboring cells level by level, which is particularly efficient for finding paths in unweighted graphs like a grid.

- Path Reconstruction:
  Saving parent state information during BFS traversal enables efficient reconstruction of the path taken by the robot. This approach minimizes memory consumption compared to storing the entire path during traversal, making it suitable for large grids.

- Scalability and Adaptability:
  The code's modular design and use of Python's deque data structure for the queue in BFS ensure scalability and adaptability. It can handle grids of various sizes and configurations, accommodating changes in the problem requirements without significant modifications.

- Print Path Functionality:
  The print_path function effectively visualizes the path taken by the robot, providing insights into its traversal strategy.By displaying each movement from one cell to another, it enhances understanding and facilitates debugging or analysis of the algorithm.

- Potential Enhancements:
  The code lays the foundation for potential enhancements, such as incorporating obstacles or implementing more advanced pathfinding algorithms like A*. Optimization techniques could be explored to improve the efficiency of BFS traversal, particularly in larger grids or grids with complex structures.

- Practical Applications:
  The code's solution to the robot traversal problem has practical applications in robotics, gaming, and logistics, where pathfinding in grid-based environments is common. Understanding and implementing BFS algorithms are valuable skills for solving various traversal and search problems in computer science and engineering domains.

7. SUMMARY

In summary, this lab report presented a Python program implementing a robot's traversal on a 2D plane using the BFS (Breadth-First Search) algorithm to reach a destination point while printing the path taken. The code effectively demonstrated the application of BFS in solving traversal problems on grid-based environments. By utilizing parent state information and efficient data structures, such as deque, the program efficiently reconstructed the path traversed by the robot. This implementation showcased the principles of graph traversal and pathfinding algorithms, emphasizing clarity, efficiency, and adaptability in addressing similar problems.