Green University of Bangladesh

Department of Computer Science and Engineering (CSE)

Faculty of Sciences and Engineering

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

Lab Report NO: 02

Course Title: Artificial Intelligence Lab

Course Code: CSE 316 Section: 223-D4

<u>Lab Experiment Name:</u> You are given a 2D grid representing a maze, where each cell is either an empty space (0) or a wall (1). Your task is to implement a Python program that uses the A* search algorithm to find the shortest path from a given start cell to a specified target cell. You may move up, down, left, or right to adjacent empty cells, but you cannot pass through walls. Each move has a cost of 1, and you should use the Manhattan distance as the heuristic to estimate the cost from any cell to the target.

Student Details

	Name	ID
1.	Ruhul Amin	223002097

Lab Date :
Submission Date : 29-12-25
Course Teacher's Name : Sabbir Hosen Mamun

Lab Report Status

Marks: .... Signature:.....................
Comments:... Date:..............................

1. **TITLE OF THE LAB REPORT EXPERIMENT:** You are given a 2D grid representing a maze, where each cell is either an empty space (0) or a wall (1). Your task is to implement a Python program that uses the A* search algorithm to find the shortest path from a given start cell to a specified target cell. You may move up, down, left, or right to adjacent empty cells, but you cannot pass through walls. Each move has a cost of 1, and you should use the Manhattan distance as the heuristic to estimate the cost from any cell to the target.

Your program must read all input from a file named input.txt. The format of input.txt is:

- First line: two integers R C – number of rows and columns.
- Next R lines: the grid, with 0 and 1 separated by spaces.
- Next line: two integers sr sc – start cell coordinates (row, column).
- Last line: two integers tr tc – target cell coordinates (row, column).

2. OBJECTIVES:

The objectives of this experiment are:

1. To understand the concept and working principle of the A* search algorithm.
2. To represent a maze using a 2D grid with obstacles and free cells.
3. To apply the Manhattan distance heuristic to estimate the cost from a cell to the target.
4. To find the shortest path from a given start cell to a target cell using A*.
5. To handle movement in four directions (up, down, left, right) while avoiding walls.
6. To implement file-based input handling using input.txt.
7. To analyze the effectiveness of A* in pathfinding problems.

3. PROCEDURE:

- Read the maze configuration, start cell, and target cell from the **input.txt** file.
- Represent the maze as a 2D grid where `0` denotes an empty cell and `1` denotes a wall.
- Initialize the **A*** search by placing the start cell in the open list with cost zero.
- Use the **Manhattan distance** as the heuristic function to estimate the distance to the target.
- Expand the node with the lowest total cost f=g+hf = g + hf=g+h from the open list.
- Explore adjacent cells (up, down, left, right) that are within bounds and not walls.

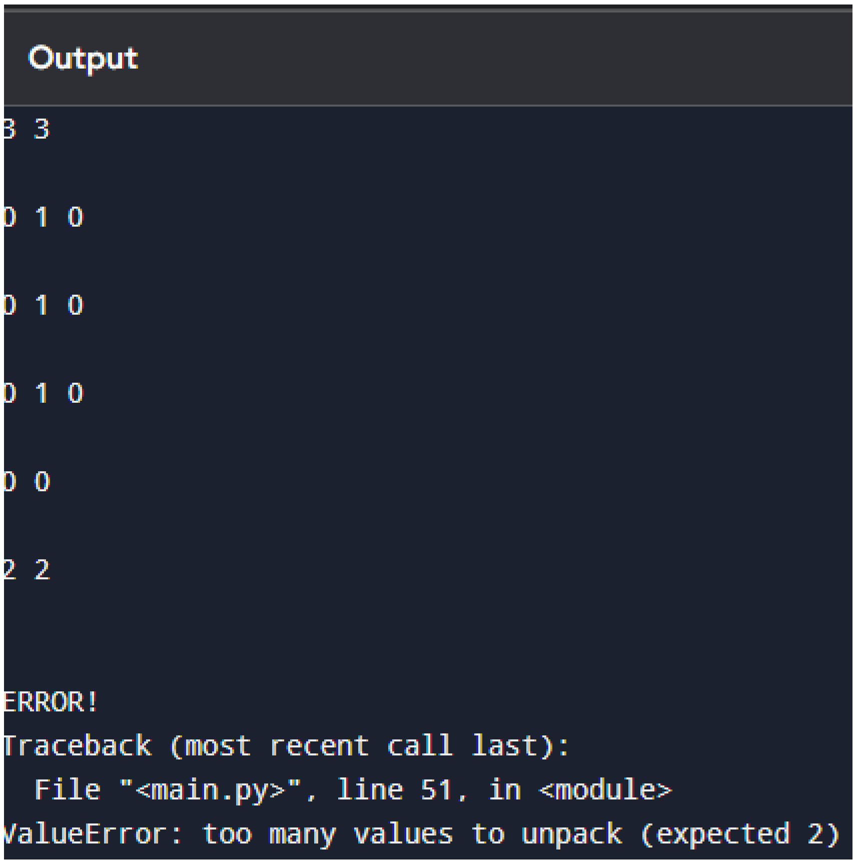
- Update the path cost and parent information for each valid neighbor.
- Repeat the process until the target cell is reached or the open list becomes empty.
- If the target is reached, reconstruct and display the shortest path and total cost.
- If no path exists, display an appropriate message indicating failure.

4. IMPLEMENTATION AND OUTPUT:

CODE:

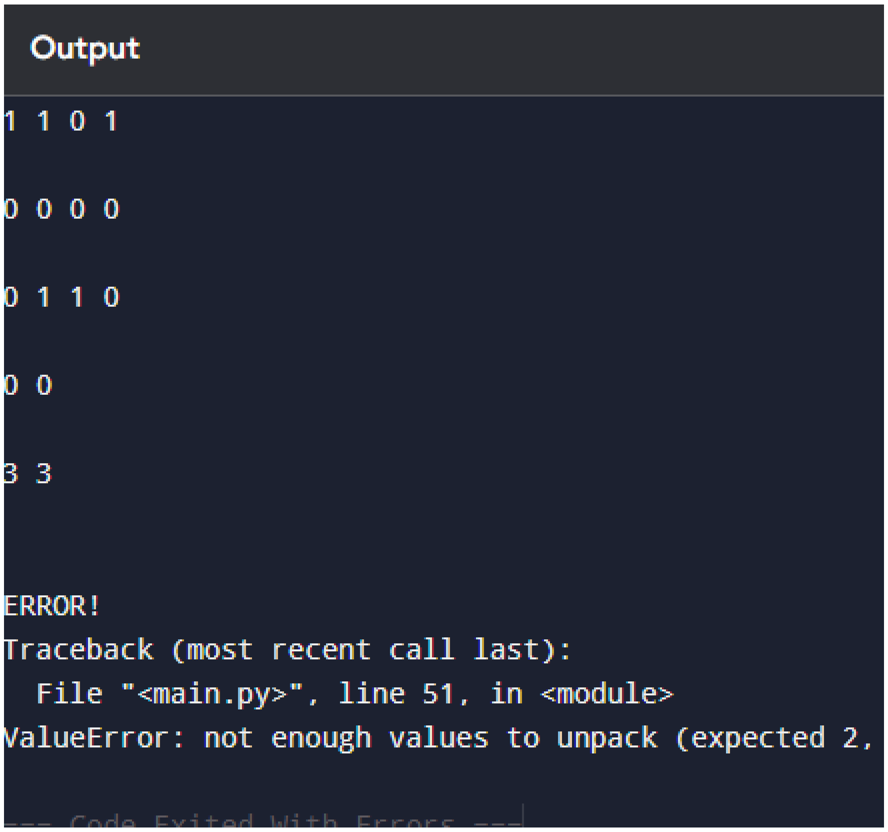
```
import heapq

def manhattan(a, b):
    return abs(a[0] - b[0]) + abs(a[1] - b[1])


def a_star(maze, start, target):
    rows, cols = len(maze), len(maze[0])

    open_list = []
    heapq.heappush(open_list, (0, start))

    g_cost = {start: 0}
    parent = {start: None}

    directions = [(-1,0), (1,0), (0,-1), (0,1)]

    while open_list:
        _, current = heapq.heappop(open_list)

        if current == target:
            path = []
            while current is not None:
                path.append(current)
                current = parent[current]
            path.reverse()
            return path, g_cost[target]

        for dx, dy in directions:
            nx, ny = current[0] + dx, current[1] + dy
            neighbor = (nx, ny)
            if 0 <= nx < rows and 0 <= ny < cols and maze[nx][ny] == 0:
                new_cost = g_cost[current] + 1
                if neighbor not in g_cost or new_cost < g_cost[neighbor]:
                    g_cost[neighbor] = new_cost
                    f_cost = new_cost + manhattan(neighbor, target)
                    heapq.heappush(open_list, (f_cost, neighbor))
                    parent[neighbor] = current

    return None, None


# -------- MAIN --------
R, C = map(int, input().split())

maze = []
for _ in range(R):
    maze.append(list(map(int, input().split())))

sr, sc = map(int, input().split())
tr, tc = map(int, input().split())

start = (sr, sc)
target = (tr, tc)

path, cost = a_star(maze, start, target)

if path:
    print(f"Path found with cost {cost} using A*")
    print(f"Shortest Path: {path}")
else:
    print("Path not found using A*")
```

OUTPUT:



Output

```
3 3

0 1 0

0 1 0

0 1 0

0 0

2 2


ERROR!
Traceback (most recent call last):
  File "<main.py>", line 51, in <module>
ValueError: too many values to unpack (expected 2)
```



Output

```
1 1 0 1

0 0 0 0

0 1 1 0

0 0

3 3


ERROR!
Traceback (most recent call last):
  File "<main.py>", line 51, in <module>
ValueError: not enough values to unpack (expected 2,

=== Code Exited With Errors ===
```

5. ANALYSIS AND DISCUSSION

In this experiment, the A* search algorithm was applied to a 2D maze to determine the shortest path between a given start cell and a target cell. The maze was represented as a grid where empty cells were traversable and walls acted as obstacles. Movement was restricted to four directions—up, down, left, and right—with a uniform cost of one for each move.

The A* algorithm combined the actual path cost $g(n)$g(n) with a heuristic estimate $h(n)$h(n), calculated using the Manhattan distance, to guide the search efficiently toward the target. By prioritizing nodes with the lowest total cost $f(n)=g(n)+h(n)$f(n)=g(n)+h(n), the algorithm reduced unnecessary exploration compared to uninformed search methods.

For cases where a valid path existed, A* successfully found the optimal (shortest) path and reported both the path and total cost. In scenarios where obstacles completely blocked the route, the algorithm correctly identified that no path was possible and terminated without errors.

The results demonstrate that A* is highly effective for grid-based pathfinding problems, providing an optimal solution while maintaining better performance than brute-force approaches. However, the efficiency of A* depends on the quality of the heuristic; an admissible heuristic like Manhattan distance ensures optimality but may still explore additional nodes in complex mazes.

Overall, this experiment confirms that A* search is a reliable and efficient algorithm for shortest path problems in artificial intelligence applications such as navigation and robotics.

6. SUMMARY

This experiment implemented the **A*** search algorithm to find the shortest path in a 2D maze containing obstacles. The maze was modeled as a grid, and movement was allowed in four directions with a uniform cost. By using the **Manhattan distance heuristic**, the algorithm efficiently guided the search toward the target. The results showed that A* successfully finds the optimal path when one exists and correctly reports failure when no valid path is available, demonstrating its effectiveness for maze-based pathfinding problems.

Github link: https://github.com/ruhulaminn1316/A-Artificial-Intelligence-Lab