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 404-CSE(181) **Section:** 223_D4

Lab Experiment Name: Implementation of A Star Search Algorithm

Student Details

Name	ID
Mujahidul Islam	193002052

Lab Date: 29/11/2025

Submission Date: 12/12/2025

Course Teacher's Name: Md. Sabbir Hosen Mamun

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

Lab Report Status

Marks: …… Signature:.....

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

1. Introduction

The purpose of this lab is to implement the A* search algorithm to find the shortest path in a grid-based maze. The algorithm uses cost-based exploration combined with a Manhattan distance heuristic to efficiently reach the target.

2. Objectives:

- Implement A* search using Python
- Read maze input from a file (input.txt)
- Determine whether a path exists
- Output the minimum cost and full path

3. Implementation:

README.md U | lab02.py | input.txt U

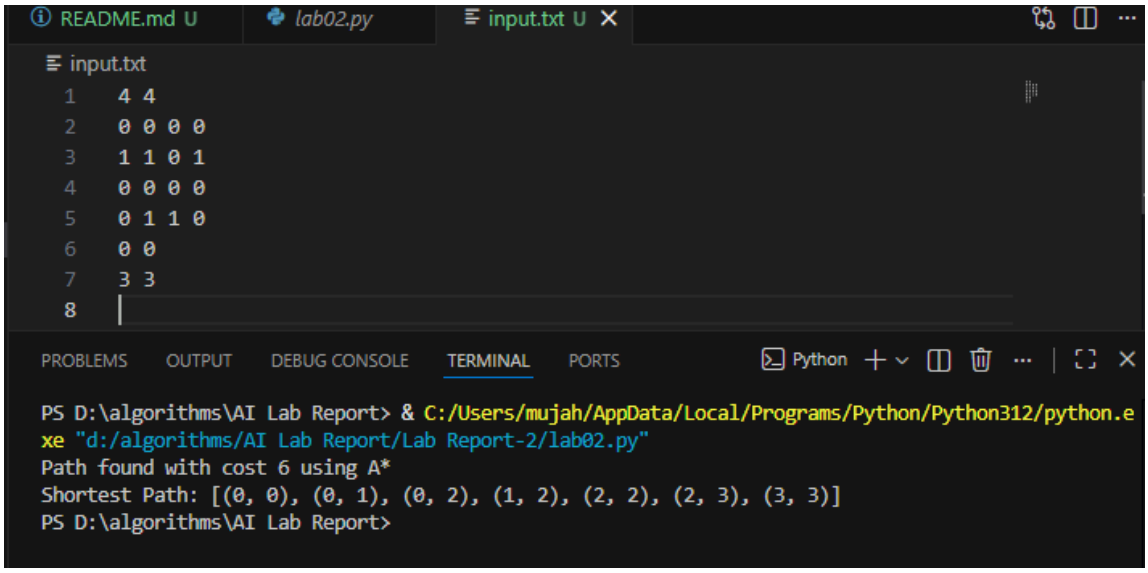
Lab Report-2 > lab02.py > a_star

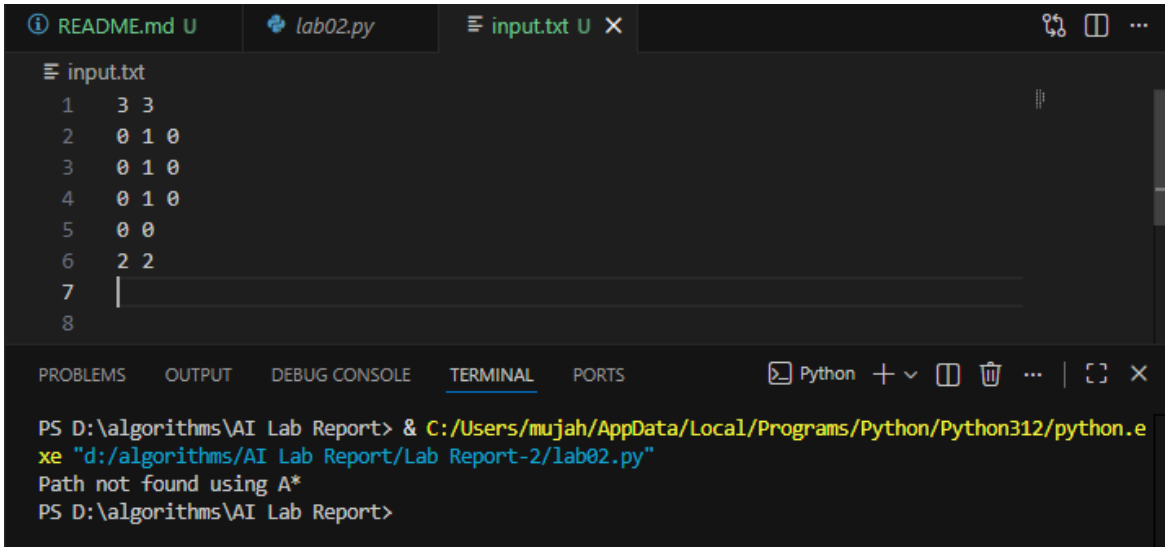
```python
import heapq

# CHANGE THIS TO YOUR OWN FILE PATH IF NEEDED
INPUT_FILE = r"D:\algorithms\AI Lab Report\input.txt"

def read_input():
    with open(INPUT_FILE, "r") as f:
        R, C = map(int, f.readline().split())
        grid = []
        for _ in range(R):
            grid.append(list(map(int, f.readline().split())))
        sr, sc = map(int, f.readline().split())
        tr, tc = map(int, f.readline().split())
    return R, C, grid, (sr, sc), (tr, tc)

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

def a_star(R, C, grid, start, target):
    (sr, sc) = start
    (tr, tc) = target

    # If the start or target is blocked
    if grid[sr][sc] == 1 or grid[tr][tc] == 1:
        return None

    open_set = []
    heapq.heappush(open set, (0, sr, sc))
```

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

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

    while open_set:
        _, r, c = heapq.heappop(open_set)

        if (r, c) == target:
            path = []
            cur = target
            while cur is not None:
                path.append(cur)
                cur = parent[cur]
            return path[::-1]

        for dr, dc in directions:
            nr, nc = r + dr, c + dc

            if 0 <= nr < R and 0 <= nc < C and grid[nr][nc] == 0:
                new_cost = g_cost[(r, c)] + 1

                if (nr, nc) not in g_cost or new_cost < g_cost[(nr, nc)]:
                    g_cost[(nr, nc)] = new_cost
                    f_cost = new_cost + manhattan((nr, nc), target)
                    heapq.heappush(open_set, (f_cost, nr, nc))
                    parent[(nr, nc)] = (r, c)

    return None

def main():
```

```
def main():
    R, C, grid, start, target = read_input()
    path = a_star(R, C, grid, start, target)

    if path is None:
        print("Path not found using A*")
    else:
        print(f"Path found with cost {len(path)-1} using A*")
        print("Shortest Path:", path)

if __name__ == "__main__":
    main()

```

4. Output:





5. Discussion:

In this lab, the A* algorithm worked well for finding a short path in the grid. It uses both the actual cost and the heuristic, so it does not waste time exploring bad directions. The Manhattan distance heuristic fits well because movement is only in four directions. When a path exists, A* finds it quickly, and when the walls block all routes, it correctly reports no path. Overall, A* is more efficient than basic searches and gives good results for this type of problem