DAA CASE STUDY REPORT

TEAM MEMBERS:

1) Alana Mariya P.C [CB.SC.U4CSE23205]
2) M. Chandana [CB.SC.U4CSE23228]
3) A Snigdha Sirivalli [CB.SC.U4CSE23264]
4) A Aadhithya Bharathi [CB.SC.U4CSE23269]

Design :

1. A* with Path Reconstruction

Goal:

This module allows robots to move around a grid with the A* algorithm, building the optimal path from the starting position to the goal with obstacle avoidance.

Modules:

- Node1 Struct: A grid cell with coordinates, cost, and heuristic.
- A Search Function (a_star_search1):
    - Utilizes a priority queue to handle nodes by cost + heuristic.
    - Maintains parent data for path construction.
    - Moves in four directions (U, D, L, R) with obstacle avoidance.
- Grid Generation (generate_random_grid1):
    - Generates a random grid of dimensions N x M.
    - Inserts obstacles at random locations.
    - Inserts robots at various locations.

2. A* with Fuel and Exploration Constraints

Goal:

This module updates A* search to take into account fuel limitation and keep an eye on discovered areas while in search of the target.

Modules:

- Node2 Struct:
    - Records position, cost, heuristic, fuel remaining, and number of explored areas.
- Modified A Search (a_star_search2):

    - Like regular A*, but now with fuel as a limitation.
    - Maximizes exploration while attempting to reach the target.
    - Ends if fuel runs out prior to reaching the target.

- Grid Generation (generate_grid2):

    - Like before with grid generation except it takes into account fuel values.

3. Multi-Robot Path Planning

Goal:

This module extends A* to plan paths for a group of robots dynamically avoiding collisions.

Components:

- Node3 Struct: Just like the previous nodes but in multi-robot environments.

- A Search with Obstacle Avoidance (a_star_search3):*

    - Creates a path for every robot avoiding other robots.
    - Prioritizes robots based on shorter paths.
    - Applies a step-by-step simulation for dynamic movement of robots.

- Grid Generation (generate_random_grid3):
    - Generates obstacles and designates initial locations for several robots.

4. Robot Assignments and Code Recovery

Goal:

- This module maps robots to grids with codes and tries to match codes based on Hamming distance and pre-defined valid codes.

Parts:

- Grid Struct & Robot Struct:

- Saves grid IDs, positions, and distinct 10-bit codes.

- Saves robot IDs, initial positions, range, and assignment limit.

Bipartite Matching (bpm function):

- Employing a DFS-based strategy to map robots to the best possible grid.

- Hamming Distance Calculation:
    - Compares 10-bit grid codes with a list of valid codes.
    - Supports error tolerance (k-bit differences).

- Grid and Robot Generation (generate_random_grid4):
    - Randomly positions grids with codes and robots with range restrictions.

Test cases :

Sample input:

Case -1 :

```
    Enter 1 for A* with Path Reconstruction, 2 for A* with Fuel and Exploration, 3 for
Multi-Robot Path Planning, or 4 for Robot Assignments and Code Recovery (or any
other number to exit): 1

Enter grid size (N M): 5 5
Enter number of robots: 3
Generated Grid:
. # S1 # .
. . . . .
. . # # #
. . S2 . .
. . . . S3

Enter target positions for each robot (x y):
1 3
3 4
0 0
```

Case 2:

```
Enter 1 for A* with Path Reconstruction, 2 for A* with Fuel and Exploration, 3 for
Multi-Robot Path Planning, or 4 for Robot Assignments and Code Recovery (or any
other number to exit): 2
Enter grid size (N M): 5 5
Enter number of robots: 3
Enter fuel amount: 5
Enter robot 1's starting position (x y): 1 1
Enter robot 1's target position (x y): 1 2
Enter robot 2's starting position (x y): 2 4
Enter robot 2's target position (x y): 2 3
Enter robot 3's starting position (x y): 3 3
Enter robot 3's target position (x y): 1 3
```

<u>Case 3:</u>

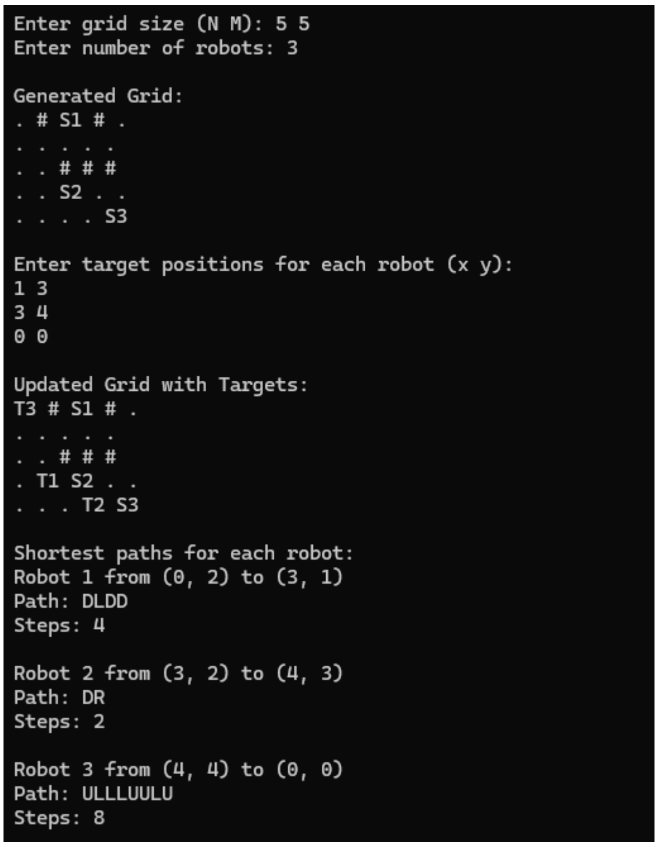
```
Enter 1 for A* with Path Reconstruction, 2 for A* with Fuel and Exploration, 3 for
Multi-Robot Path Planning, or 4 for Robot Assignments and Code Recovery (or any
other number to exit): 3
Enter grid size (N M): 5 5
Enter number of robots: 3

Generated Grid:
. S2 S3 . #
. . S1 . #
. . . # .
# # . . .
. . . . .

Enter target positions for each robot (x y):
1 4
3 3
4 4
```

<u>Case 4:</u>

```
Enter 1 for A* with Path Reconstruction, 2 for A* with Fuel and Exploration, 3 for
Multi-Robot Path Planning, or 4 for Robot Assignments and Code Recovery (or any
other number to exit): 4
Enter grid size (N M): 5 5
Enter number of robots: 3
Enter number of grids: 3

Generated Grid:
. G1 . . R1
G2 . . . .
. R3 . . .
R2 . . . .
. . . . G3
Enter max error tolerance (k): 2
```

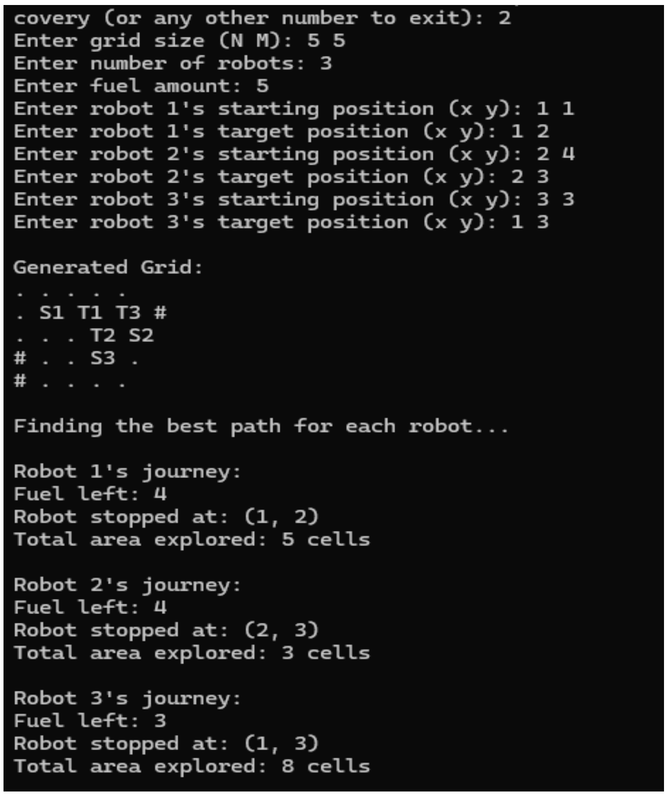
OUTPUT :

Case 1



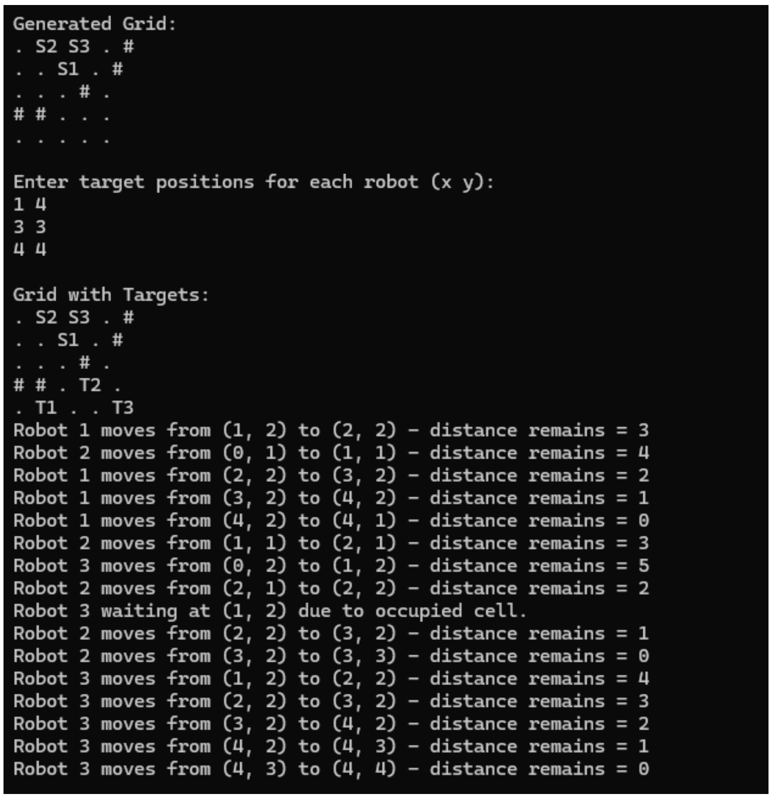
```
Enter grid size (N M): 5 5
Enter number of robots: 3

Generated Grid:
. # S1 # .
. . . . .
. . # # #
. . S2 . .
. . . . S3

Enter target positions for each robot (x y):
1 3
3 4
0 0

Updated Grid with Targets:
T3 # S1 # .
. . . . .
. . # # #
. T1 S2 . .
. . . T2 S3

Shortest paths for each robot:
Robot 1 from (0, 2) to (3, 1)
Path: DLDD
Steps: 4

Robot 2 from (3, 2) to (4, 3)
Path: DR
Steps: 2

Robot 3 from (4, 4) to (0, 0)
Path: ULLLUULU
Steps: 8
```

Case 2



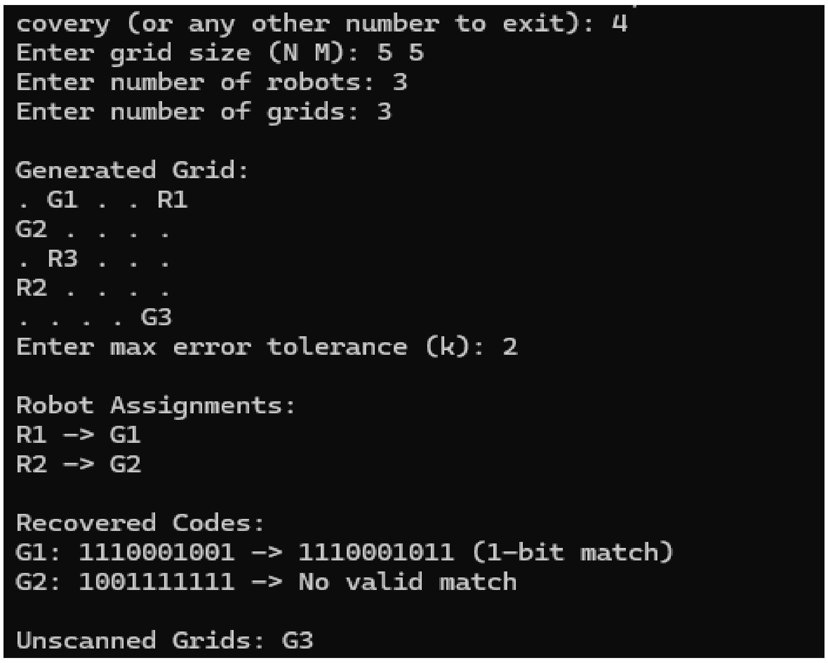
```
covery (or any other number to exit): 2
Enter grid size (N M): 5 5
Enter number of robots: 3
Enter fuel amount: 5
Enter robot 1's starting position (x y): 1 1
Enter robot 1's target position (x y): 1 2
Enter robot 2's starting position (x y): 2 4
Enter robot 2's target position (x y): 2 3
Enter robot 3's starting position (x y): 3 3
Enter robot 3's target position (x y): 1 3

Generated Grid:
. . . . .
. S1 T1 T3 #
. . . T2 S2
# . . S3 .
# . . . .

Finding the best path for each robot...

Robot 1's journey:
Fuel left: 4
Robot stopped at: (1, 2)
Total area explored: 5 cells

Robot 2's journey:
Fuel left: 4
Robot stopped at: (2, 3)
Total area explored: 3 cells

Robot 3's journey:
Fuel left: 3
Robot stopped at: (1, 3)
Total area explored: 8 cells
```

Case 3



```
Generated Grid:
. S2 S3 . #
. . S1 . #
. . . # .
# # . . .
. . . . .

Enter target positions for each robot (x y):
1 4
3 3
4 4

Grid with Targets:
. S2 S3 . #
. . S1 . #
. . . # .
# # . T2 .
. T1 . . T3
Robot 1 moves from (1, 2) to (2, 2) - distance remains = 3
Robot 2 moves from (0, 1) to (1, 1) - distance remains = 4
Robot 1 moves from (2, 2) to (3, 2) - distance remains = 2
Robot 1 moves from (3, 2) to (4, 2) - distance remains = 1
Robot 1 moves from (4, 2) to (4, 1) - distance remains = 0
Robot 2 moves from (1, 1) to (2, 1) - distance remains = 3
Robot 3 moves from (0, 2) to (1, 2) - distance remains = 5
Robot 2 moves from (2, 1) to (2, 2) - distance remains = 2
Robot 3 waiting at (1, 2) due to occupied cell.
Robot 2 moves from (2, 2) to (3, 2) - distance remains = 1
Robot 2 moves from (3, 2) to (3, 3) - distance remains = 0
Robot 3 moves from (1, 2) to (2, 2) - distance remains = 4
Robot 3 moves from (2, 2) to (3, 2) - distance remains = 3
Robot 3 moves from (3, 2) to (4, 2) - distance remains = 2
Robot 3 moves from (4, 2) to (4, 3) - distance remains = 1
Robot 3 moves from (4, 3) to (4, 4) - distance remains = 0
```

Case 4



```
covery (or any other number to exit): 4
Enter grid size (N M): 5 5
Enter number of robots: 3
Enter number of grids: 3

Generated Grid:
. G1 . . R1
G2 . . . .
. R3 . . .
R2 . . . .
. . . . G3
Enter max error tolerance (k): 2

Robot Assignments:
R1 -> G1
R2 -> G2

Recovered Codes:
G1: 1110001001 -> 1110001011 (1-bit match)
G2: 1001111111 -> No valid match

Unscanned Grids: G3
```

Time complexity Analysis:

1. A with Path Reconstruction (Code 1)

A* search algorithm operates with a priority queue and explores the best possible paths first using heuristics. The complexity is based on the following factors:

- **Priority Queue Operations:** Each node is pushed into and popped from the priority queue, which takes O(logV) time per operation.
- **Neighbor Exploration:** Each node has at most 4 neighbors (in a grid-based movement system).
- **Number of Nodes Processed:** In the worst case, all nodes are visited before reaching the goal.

For an N×M grid, the worst-case scenario occurs when every cell is visited, leading to a complexity of:

O(VlogV)=O(NM log(NM))

Where V=N×M is the number of nodes (cells in the grid).

---

2. A with Fuel and Exploration (Code 2)

This version of A* introduces a fuel constraint and additional state tracking (fuel and exploration progress).

- **State Expansion:** Each node state includes fuel tracking, meaning each node could have multiple states.
- **Priority Queue Operations:** Each state is pushed and popped from the priority queue, taking O(logV) time.
- **Neighbor Exploration:** Similar to standard A*, each node has at most 4 neighbors, but the fuel constraint may limit further exploration.
- **Worst-case Nodes Processed:** In the worst case, each node is visited multiple times due to different fuel levels being tracked. If the fuel is at most FFF, each node could have up to FFF states.

Thus, the worst-case time complexity is:

O(NMF log(NMF))

Where FFF is the fuel limit.

---

3. Multi-Robot Path Planning (Code 3)

This version includes collision avoidance, meaning that robots may have to wait if a higher-priority robot is using a path.

- *A Complexity per Robot:** Each robot runs A* individually, leading to

  O(NM log(NM)) per robot.

- **Collision Handling:** In the worst case, robots may need to backtrack or wait, increasing the number of expanded nodes.

- **Number of Robots:** If there are RRR robots, each with its own A* search, the worst-case complexity is:

$O(R \cdot NM \log(NM))$

This assumes independent robots with minimal waiting. If collision resolution requires re-planning frequently, the complexity could be higher, but it remains within:

$O(R \cdot NM \log(NM))$

In extreme cases where robots continuously block each other, complexity could increase, but it typically remains polynomial.

4. Robot Assignments and Code Recovery

1.Bipartite Matching:
- Assigns robots to grids using DFS-based matching.
- Complexity: (O(R.G)), where R = number of robots, G = number of grids.

2. Hamming Distance Calculation:
- Compare grid codes to valid codes.
- Complexity: (O(G.V.L)), where V= number of valid codes, L = code length (10 bits).

3. Grid Generation:
- Randomly generates grids and robot positions.
- Complexity: (O(N. M)), where (N times M) = grid size.

Overall Time Complexity

O(G. (R + V. L))

Simplified Complexity
If V and Lare small constants:
O(G. R)

Dominated by bipartite matching and Hamming distance calculation.Scales linearly with the number of grids (G) and robots (R)