Assignment 1

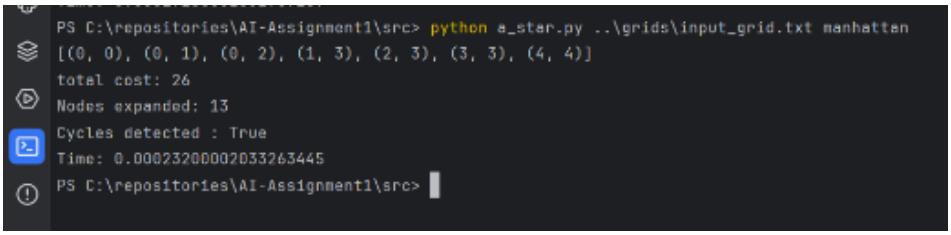
Cristina Chavira

10/20/2024

**Task 1: Implement A* with Cycle Detection**

My program is runnable via the command line below, it prints path found, total cost, nodes expanded, and cycles detected:

python a_star.py input_grid.txt manhattan

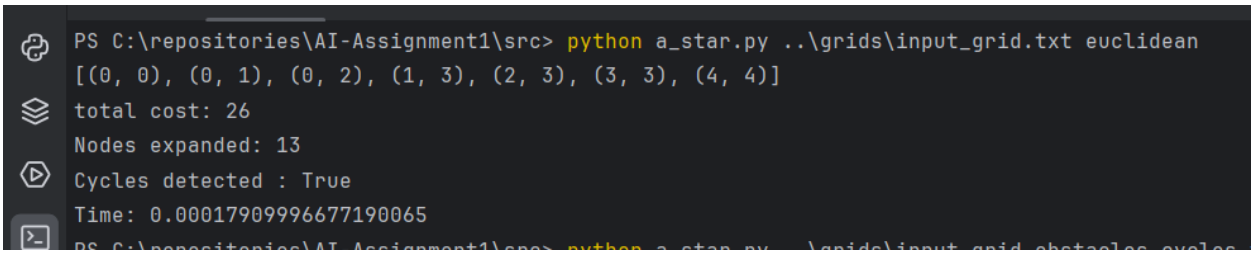



python a_star.py input_grid.txt euclidean

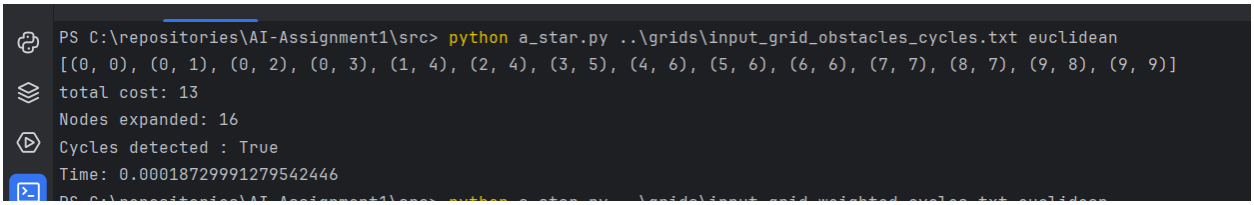

**Task 2: Test A* with Cycle Detection on Different Scenarios**

Create and test A* implementation with the grid scenarios:

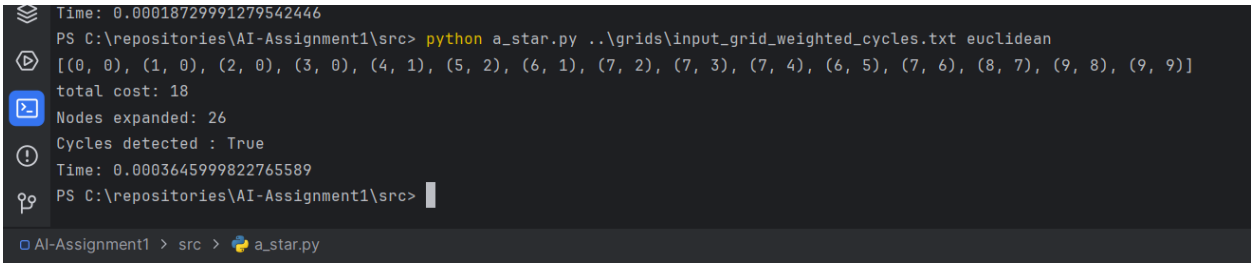
input_grid_basic.txt



input_grid_obstacles_cycles.txt



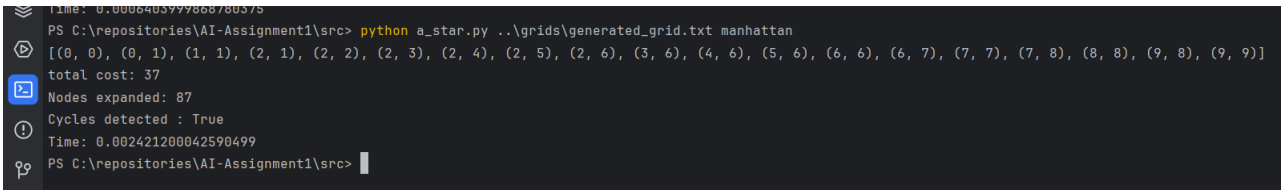

input_grid_weighted_cycles.txt



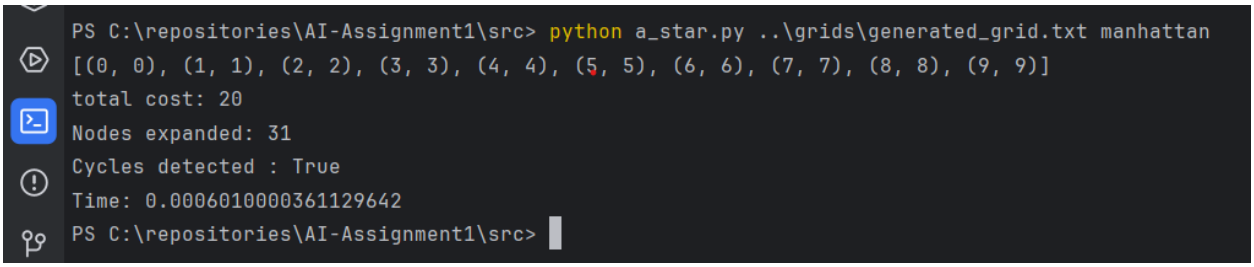

**Task 3: Experiment with Heuristic Functions**

1. Manhattan Distance

Vertical and Horizontal:

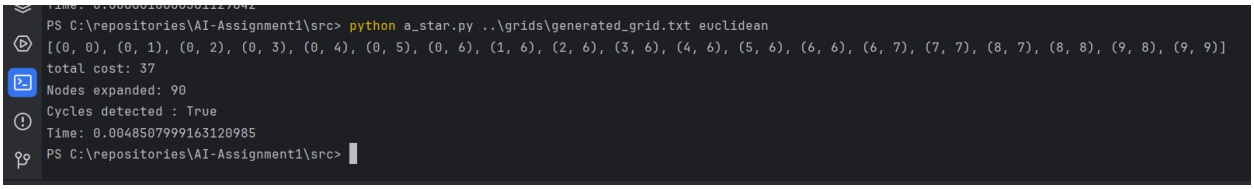



Diagonal:

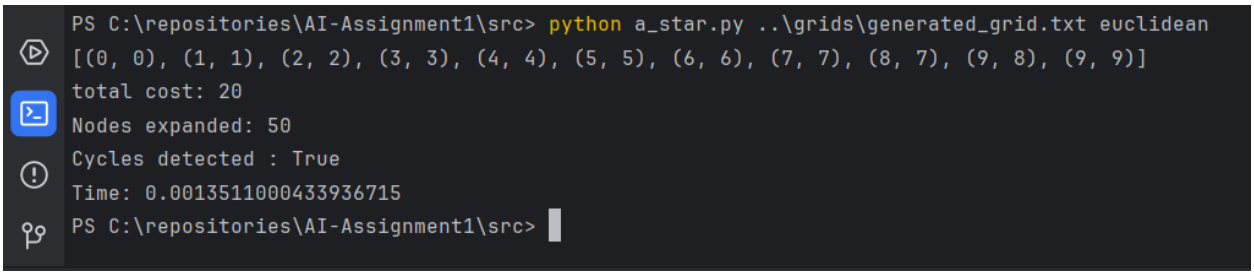



2. Euclidean Distance

Vertical and Horizontal:




Diagonal:




**Comparison:**

1. Manhattan Heuristic:

- Number of Nodes Expanded: Fewer nodes expanded in most scenarios.
- Search Time: Faster overall, particularly in grids that restrict diagonal movement.
- Path Optimality: Optimal for grids where only horizontal and vertical movement is allowed but leads to less optimal paths (higher cost) when diagonal movement is possible.

2.Euclidean Heuristic:

- Number of Nodes Expanded: Expands more nodes in some cases, especially when diagonal movement is allowed.
- Search Time: Slower due to considering diagonal paths and additional complexity.
- Path Optimality: Provides optimal paths in grids where diagonal movement is allowed, resulting in lower path costs when compared to Manhattan in those cases.

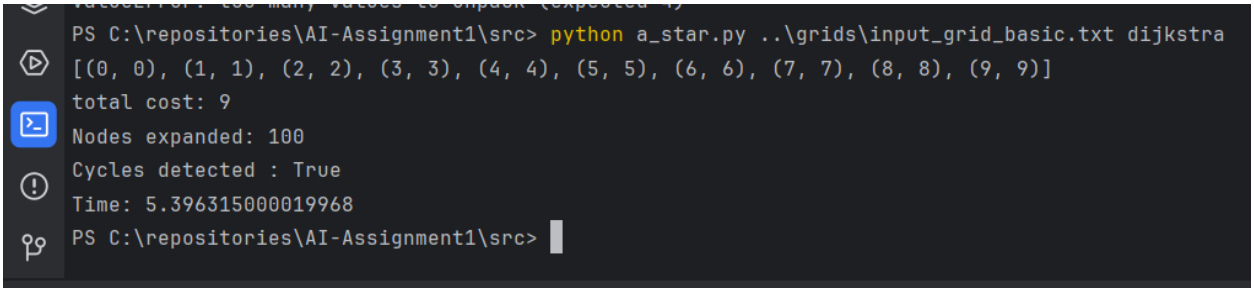
**Summary:**

The Manhattan heuristic is more efficient in terms that both nodes expanding and search time when diagonal movement is not required or allowed. The Euclidean heuristic provides better path optimality when diagonal movement is allowed but at the cost of expanding more nodes and taking more time to complete the search. While the Manhattan heuristic is faster and more efficient for grid-based movement, the Euclidean heuristic is more accurate but slower,

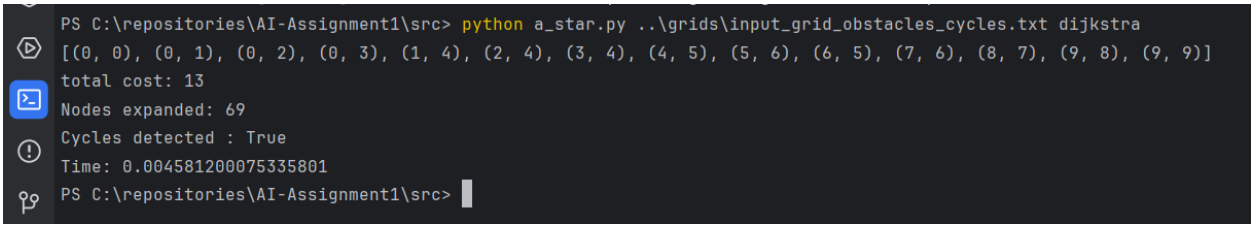
making it better suited for environments where diagonal movement is necessary for optimal paths.

**Task 4: Comparison with Dijkstra's Algorithm**

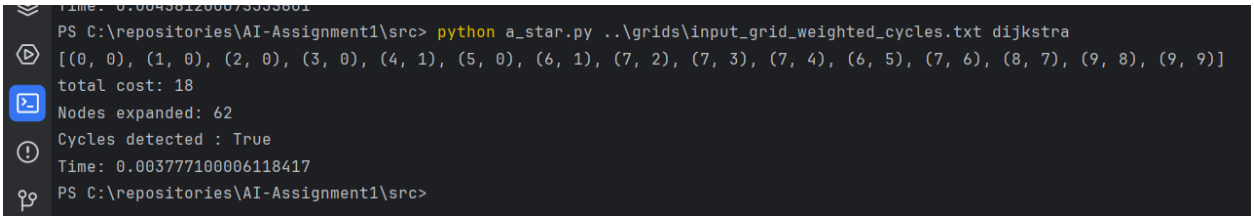
input_grid_basic.txt




input_grid_obstacles_cycles.txt




input_grid_weighted_cycles.txt




**Comparison (reference Task 2 for screenshots for A* algorithm):**

1.Number of Nodes Expanded:

- Dijkstra:
    a. input_grid_basic.txt: 100 nodes
    b. input_grid_obstacles_cycles.txt: 69 nodes
    c. input_grid_weighted_cycles.txt: 62 nodes

- A* (Euclidean):
    a. input_grid_basic.txt: 10 nodes
    b. input_grid_obstacles_cycles.txt: 16 nodes
    c. input_grid_weighted_cycles.txt: 26 nodes

**Observation**: A* with the Euclidean heuristic expands significantly fewer nodes than Dijkstra in all cases. This is expected because A* uses the heuristic to focus the search on the goal, whereas Dijkstra explores nodes more exhaustively.

2.Total Search Time:

3. Dijkstra:
    a. input_grid_basic.txt: 4.7579 seconds
    b. input_grid_obstacles_cycles.txt: 0.00458 seconds
    c. input_grid_weighted_cycles.txt: 0.00378 seconds
4. A* (Euclidean):
    a. input_grid_basic.txt: 0.00053 seconds
    b. input_grid_obstacles_cycles.txt: 0.00025 seconds
    c. input_grid_weighted_cycles.txt: 0.00071 seconds

**Observation**: A* is much faster than Dijkstra in all test cases, particularly in the input_grid_basic.txt case, where Dijkstra's time is greater. The heuristic helps A* reduce the search time by exploring fewer nodes.

3. Path Optimality (Total Cost):

- Dijkstra:
    - input_grid_basic.txt: Cost = 9
    - input_grid_obstacles_cycles.txt: Cost = 13
    - input_grid_weighted_cycles.txt: Cost = 18
- A* (Euclidean):
    - input_grid_basic.txt: Cost = 9
    - input_grid_obstacles_cycles.txt: Cost = 13
    - input_grid_weighted_cycles.txt: Cost = 18

**Observation**: Both A* and Dijkstra find the same optimal paths with the same costs. This confirms that A* with the Euclidean heuristic is admissible (it doesn't overestimate the cost) and produces optimal paths like Dijkstra.

## Conclusion

In this project, I implemented and tested the A* pathfinding algorithm with cycle detection, incorporating both Manhattan and Euclidean heuristics, while also comparing its performance to Dijkstra's algorithm. I evaluated the algorithms based on several metrics, including total path cost, the number of nodes expanded, the time taken to find the path, and whether cycles were detected. Through this process, I was able to observe how the choice of heuristic impacts the algorithm's efficiency. For instance, the Manhattan heuristic performed better in grids with straight paths, while the Euclidean heuristic was more suitable for diagonally connected nodes. In comparing A* and Dijkstra's algorithms, I found that A* generally expanded fewer nodes, making it more efficient in certain cases. This highlights the importance of selecting the right heuristic based on specific characteristics of the grid to optimize performance.