October 20, 2024

Programming Assignment 1

Pathfinding Using A* Algorithm with Cycle Detection

Escamilla, Jeffrey
Computer Science
The University of Texas at El Paso

Programming Assignment 1

Assignment: Pathfinding Using A* Algorithm with Cycle Detection

Jeffrey Escamilla

Table of Contents

Programming Assignment 1

Assignment: Pathfinding Using A* Algorithm with Cycle Detection

Jeffrey Escamilla

Introduction

Course Details

Name: Jeffrey Escamilla; PhD CS Student

Student ID: 80268293

Course Details: CS5314 Decision Making

Instructor: Dr. Md M

Purpose

This homework assignment explores the use of the A-star algorithm and examines various heuristics and their usefulness. The Dijkstra Algorithm is also examined as a special case of the A-star algorithm.

Objective

In this assignment, you will implement the *A* (A-star) search algorithm* to solve pathfinding problems using **Python**. You will explore how A* balances exploration and exploitation by using both the **actual cost** from the start and an **estimated cost** to the goal, leveraging a heuristic function. Additionally, you will implement **cycle detection** to handle scenarios where the search space contains loops. The goal is to find the optimal path from a start node to a goal node in a grid.

Assignment Overview

You will:

1. Implement A* algorithm with **cycle detection**.
2. Apply the algorithm to find optimal paths in a grid-based environment.
3. Experiment with different heuristic functions and analyze their impact on performance.
4. Solve multiple pathfinding scenarios, comparing A* to **Dijkstra's algorithm**.

Deliverables

1. **Source Code:** The Python implementation of A*, including:
    a. Cycle detection mechanism.
    b. Heuristic functions (Manhattan and Euclidean).
    c. Dijkstra's algorithm for comparison
2. **Test Results:** The output showing the paths, costs, number of nodes expanded, and cycle detection results for each scenario.

3. **Comparison Report:** A report comparing A* with different heuristics and Dijkstra's algorithm, focusing on the performance and efficiency of each.

Submission

Submit a ZIP file containing:

1. The **source code** files (including A*, Dijkstra, heuristics).
2. Txt files containing the grids.
3. A **report** (PDF format) with your comparison and findings.
4. A **README** file with clear instructions on how to run the code from the command line.

Tasks

Task 1: Implement A* with Cycle Detection

- Implement the A* search algorithm in **Python**, ensuring that it runs from the **command line/terminal**.
- You do not need to start from scratch. You may use this base implementation to get started: https://medium.com/@nicholas.w.swift/easy-a-starpathfinding-7e6689c7f7b2

Your implementation should:

1. Accept a **grid** representation from a file (as a 2D array).
2. Define **g(n)** (the actual cost from the start to node **n**) and **h(n)** (the heuristic estimate of the cost from **n** to the goal).
3. Combine these to compute **f(n) = g(n) + h(n)**.
4. Implement **cycle detection** to prevent the algorithm from revisiting nodes unnecessarily. This can be achieved by maintaining a **visited set**.
5. Return the **optimal path** and its cost.

Cycle Detection Requirement

Ensure that your implementation handles loops and avoids infinite cycles by keeping track of previously visited nodes.

Grid Constraints

The input grid will contain **cell values between 0 and 5**:

- A value of **0** means the cell is **unpassable** (an obstacle).
- Values **1 to 5** represent the **cost of traversing the cell**, with **5** being more expensive and **1** being the least expensive (apart from 0, which is unpassable).

Command Line Usage

Your program should be runnable via the command line as follows:

```
python a_star.py input_grid.txt manhattan
```

Where:

- `input_grid.txt` is a file containing the grid.
- `manhattan` specifies the heuristic to be used (other option: `euclidean`).

Output

The program should print:

- The **path found**.
- The **total cost**.
- The number of **nodes expanded**.
- Whether **cycles were detected**.

Sample Test Case

```
# input_grid.txt
5 5 5 5 5
0 0 0 5 5
5 5 5 5 0
5 1 1 1 5
5 5 5 0 5

Start: (0,0)
Goal: (4,4)
Heuristic: Manhattan
```

Expected Output

```
Path: [(0, 0), (1, 3), (2, 3), (3, 3), (4, 4)]
Cost: 16
Nodes Expanded: [Insert number]
Cycles Detected: No
```

Solution & Analysis

All the sub-tasks listed were completed.

*Accept a **grid** representation from a file (as a 2D array).*

The program accepts a grid representation given as a simple text file. This is achieved via the following function:

```
def extract_grid(file_path)
```

It effectively takes the text file and processes each line to build the 2d-list that will be needed as input for the A-star algorithm. The start and end positions default to the top-left (start) and bottom-right (end). This is achieved by computing the rows and columns of the given grid.

*Define **g(n)** (the actual cost from the start to node **n**) and **h(n)** (the heuristic estimate of the cost from **n** to the goal).*

`g(n)` is computed by simply adding the cell value (from the array) of the child node being processed to the current node's g-value:

```
child.g = current_node.g + maze[child.position[0]][child.position[1]]
```

`h(n)` is computed via a new function:

```
def heuristic(pos, goal, heuristic_type=None):
```

It gets used when computing h in the A-star algorithm:

```
child.h = heuristic(child.position, end_node.position, heuristic_type)
```

By doing it this way, the program can take any heuristic provided by the user.

*Combine these to compute **f(n) = g(n) + h(n)**.*

This update happens when processing the child nodes. It is a simple computation:

```
child.f = child.g + child.h
```

*Implement **cycle detection** to prevent the algorithm from revisiting nodes unnecessarily. This can be achieved by maintaining a **visited set**.*

This is accomplished by keeping a list called `closed_list` of all the visited nodes. When new nodes are being processed, they are checked against the `closed_list`. If they appear in this list, they are effectively skipped, and the next node in the queue is selected for processing.

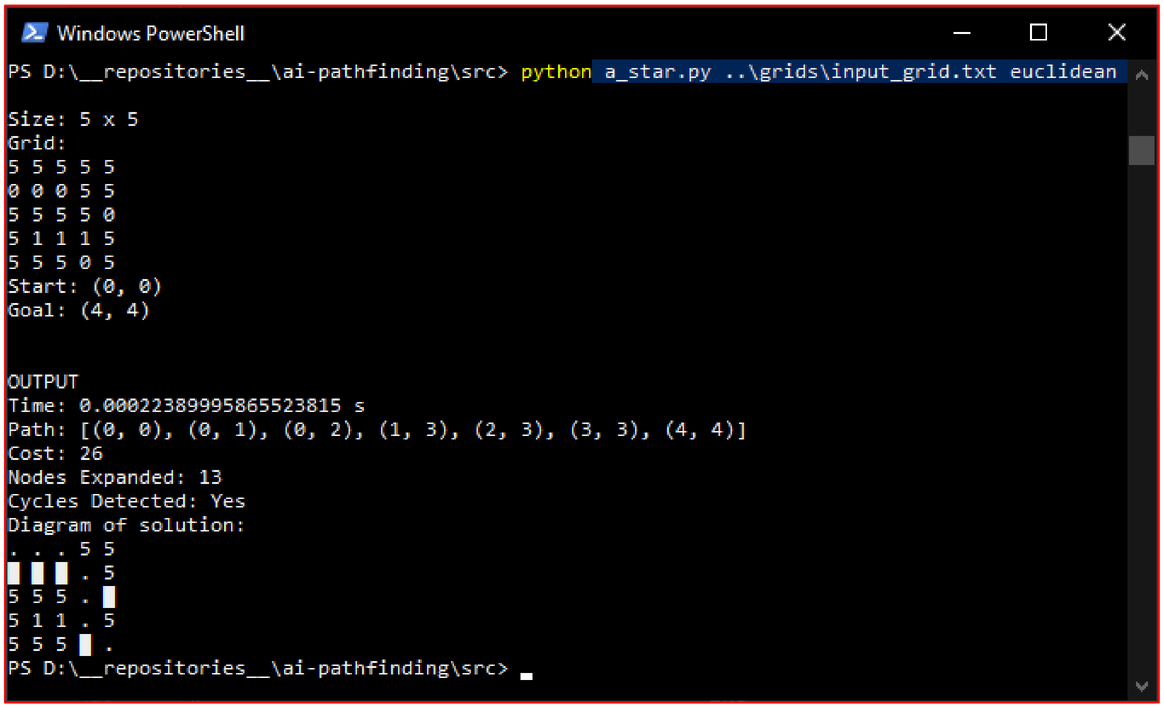
Return the **optimal path** and its cost.

Once the current node reaches the end, we can simply extract the cost (i.e., the g value) and return it. The optimal path is constructed by

```
def return_path(current_node):
```

It works by creating a list and appending the parent of the current of the node, updating the current node to that parent, and then getting the next parent, and so on and so forth. It returns the list in reverse order as it's constructed from end to start.

Below is the input for the sample test case as well as the output:



```
PS D:\__repositories__\ai-pathfinding\src> python a_star.py ..\grids\input_grid.txt euclidean

Size: 5 x 5
Grid:
5 5 5 5 5
0 0 0 5 5
5 5 5 5 0
5 1 1 1 5
5 5 5 0 5
Start: (0, 0)
Goal: (4, 4)


OUTPUT
Time: 0.00022389995865523815 s
Path: [(0, 0), (0, 1), (0, 2), (1, 3), (2, 3), (3, 3), (4, 4)]
Cost: 26
Nodes Expanded: 13
Cycles Detected: Yes
Diagram of solution:
. . . 5 5
█ █ █ . 5
5 5 5 . █
5 1 1 . 5
5 5 5 █ .
PS D:\__repositories__\ai-pathfinding\src> _
```

Fig 1: Test with input_grid.txt

From Figure 1, it can be seen that

- The implementation accounts for varying traversal costs as well as obstacles.
- The program is runnable via the command line according to the instructions.
- The output shows the (1) path found, (2) total cost, (3) number of nodes expanded, and (4) whether cycles were detected.
- Additionally, the total time is displayed, and a visual of the solution path is shown.

Task 2: Test A with Cycle Detection on Different Scenarios

Create and test your A* implementation with the following grid scenarios:

Task 2.1: Basic Grid

A 10×10 grid with no obstacles. The start is at the top-left, and the goal is at the bottom-right.

Sample Test Case

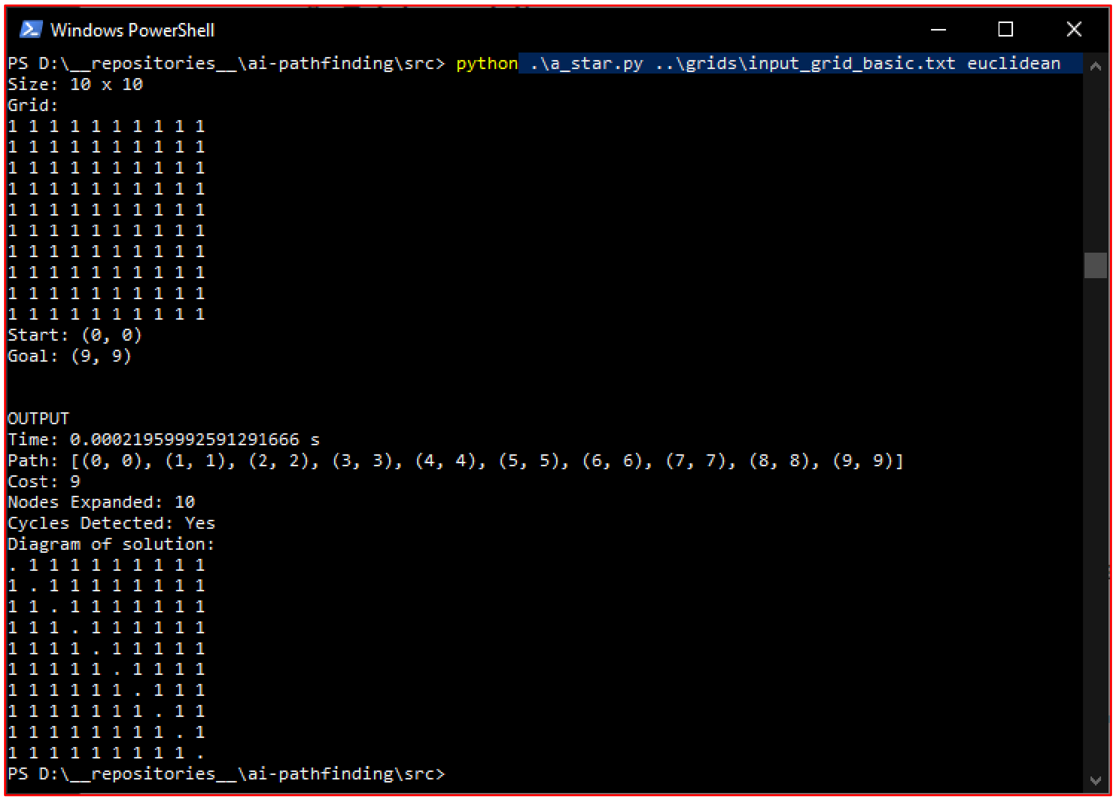
```
# input_grid_basic.txt
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1

Start: (0,0)
Goal: (9,9)
```

Expected Output

```
Path: [(0, 0), (1, 1), ..., (9, 9)]
Cost: 18
Nodes Expanded: [Insert number]
Cycles Detected: No
```

Results

Running the sample test case with the Euclidean heuristic and diagonal movement:



```
PS D:\__repositories__\ai-pathfinding\src> python .\a_star.py ..\grids\input_grid_basic.txt euclidean
Size: 10 x 10
Grid:
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
Start: (0, 0)
Goal: (9, 9)


OUTPUT
Time: 0.00021959992591291666 s
Path: [(0, 0), (1, 1), (2, 2), (3, 3), (4, 4), (5, 5), (6, 6), (7, 7), (8, 8), (9, 9)]
Cost: 9
Nodes Expanded: 10
Cycles Detected: Yes
Diagram of solution:
. 1 1 1 1 1 1 1 1 1
1 . 1 1 1 1 1 1 1 1
1 1 . 1 1 1 1 1 1 1
1 1 1 . 1 1 1 1 1 1
1 1 1 1 . 1 1 1 1 1
1 1 1 1 1 . 1 1 1 1
1 1 1 1 1 1 . 1 1 1
1 1 1 1 1 1 1 . 1 1
1 1 1 1 1 1 1 1 . 1
1 1 1 1 1 1 1 1 1 .
PS D:\__repositories__\ai-pathfinding\src>
```

Fig 2.1: Test with input_grid_basic.txt

The A-star algorithm runs relatively fast when the Euclidean heuristic is used.

Task 2.2: Grid with Obstacles and Cycles

A 10×10 grid with obstacles and edges forming cycles. The start and goal positions must not be blocked.

Sample Test Case

```
# input_grid_obstacles_cycles.txt
1 1 1 1 1 1 1 1 1 1
1 0 0 0 1 1 1 0 0 1
1 0 1 0 1 0 0 0 1 1
1 0 1 0 1 1 1 0 1 1
1 0 1 1 1 1 1 1 1 1
1 1 1 1 0 0 1 0 1 1
1 1 0 0 0 1 1 0 0 1
1 1 1 1 1 1 1 1 0 1
1 0 0 0 0 0 0 1 0 1
1 1 1 1 1 1 1 1 1 1
```

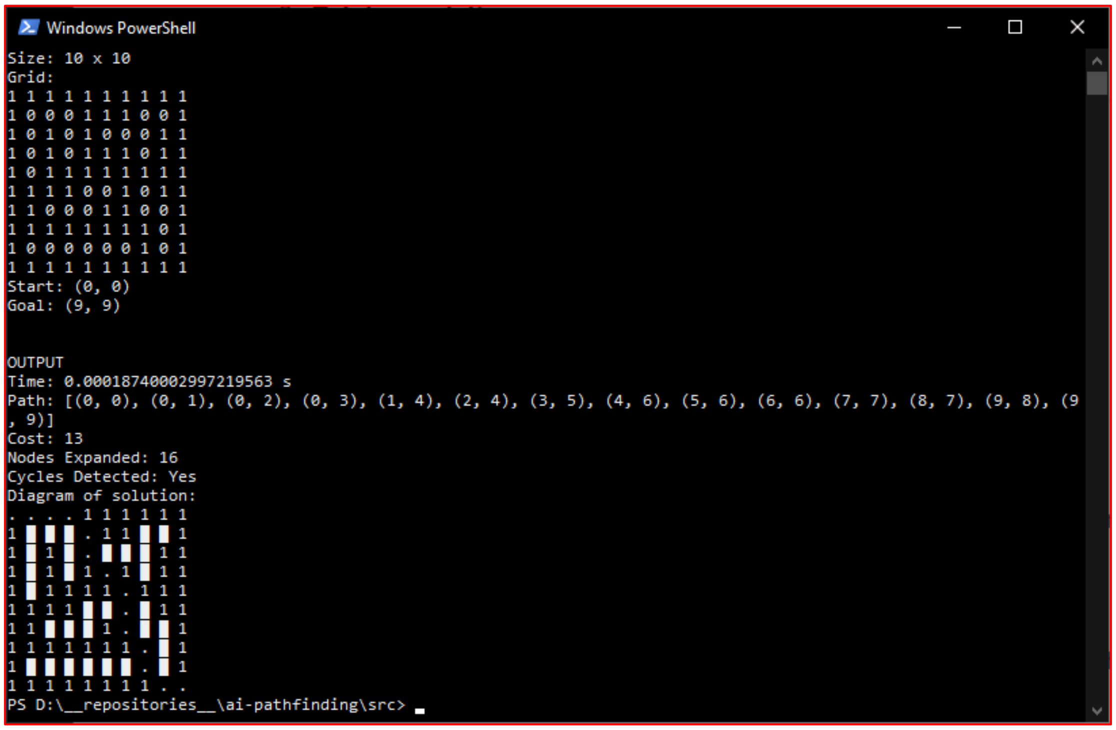
```
Start: (0,0)
Goal: (9,9)
```

Expected Output

```
Path: [(0, 0), ..., (9, 9)]
Cost: [Insert cost]
Nodes Expanded: [Insert number]
Cycles Detected: Yes
```

Results

Running the sample test case with the Euclidean heuristic and diagonal movement:



```
Size: 10 x 10
Grid:
1 1 1 1 1 1 1 1 1 1
1 0 0 0 1 1 1 0 0 1
1 0 1 0 1 0 0 0 1 1
1 0 1 0 1 1 1 0 1 1
1 0 1 1 1 1 1 1 1 1
1 1 1 1 0 0 1 0 1 1
1 1 0 0 0 1 1 0 0 1
1 1 1 1 1 1 1 1 0 1
1 0 0 0 0 0 0 1 0 1
1 1 1 1 1 1 1 1 1 1
Start: (0, 0)
Goal: (9, 9)


OUTPUT
Time: 0.00018740002997219563 s
Path: [(0, 0), (0, 1), (0, 2), (0, 3), (1, 4), (2, 4), (3, 5), (4, 6), (5, 6), (6, 6), (7, 7), (8, 7), (9, 8), (9
, 9)]
Cost: 13
Nodes Expanded: 16
Cycles Detected: Yes
Diagram of solution:
. . . . 1 1 1 1 1 1
1 █ █ █ . 1 1 █ █ 1
1 █ 1 █ . █ █ █ 1 1
1 █ 1 █ 1 . 1 █ 1 1
1 █ 1 1 1 1 . 1 1 1
1 1 1 1 █ █ . █ 1 1
1 1 █ █ █ 1 . █ █ 1
1 1 1 1 1 1 1 . █ 1
1 █ █ █ █ █ █ . █ 1
1 1 1 1 1 1 1 1 . .
PS D:\__repositories__\ai-pathfinding\src>
```

Fig 2.2: Test with input_grid_obstacles_cycles.txt

The runtime went down as obstacles are added.

Task 2.3: Larger Grid with Obstacles and Cycles

A 20×20 grid with multiple paths that form loops.

Sample Test Case

```
# input_grid_basic.txt
```

```
3 4 1 1 0 3 4 5 3 3 0 1 2 4 3 2 5 4 5 1
5 4 5 1 1 1 5 5 2 1 3 1 4 5 4 3 3 3 4 3
1 5 4 4 4 5 1 2 4 1 4 2 1 1 5 3 2 1 2 5
3 1 4 1 3 4 5 3 4 5 2 4 5 2 5 3 4 2 2 2
1 5 4 3 5 5 4 3 1 5 2 5 2 3 4 3 4 1 5 4
5 1 5 4 5 1 3 1 4 5 1 5 1 4 2 4 2 5 3 1
2 1 2 2 4 3 3 1 3 3 5 4 3 1 2 1 0 5 2 2
5 3 1 1 1 1 1 1 2 5 5 3 1 3 2 5 4 0 1 4
4 4 0 1 2 3 3 2 2 1 1 5 3 5 3 3 1 4 2 3
3 2 1 5 4 3 5 5 1 1 4 5 3 2 5 3 5 1 4 3
5 2 4 1 4 4 2 1 1 4 5 3 2 4 5 2 5 2 3 5
2 4 5 4 2 5 2 4 0 5 3 3 2 2 1 5 3 5 3 1
3 4 4 2 5 5 1 2 5 1 2 2 1 5 3 2 3 2 3 3
2 1 2 2 2 5 4 2 2 5 4 2 5 5 5 2 2 1 1 1
5 2 3 4 2 5 3 2 1 5 1 3 3 5 2 2 4 5 4 1
3 5 4 5 4 3 1 1 3 0 5 4 4 2 2 2 4 4 2 3
1 2 5 5 4 3 1 2 3 5 1 5 1 1 4 4 5 1 5 4
2 5 2 3 5 5 5 0 1 2 4 3 4 5 1 4 1 1 1 5
1 3 3 5 2 5 2 2 4 5 2 4 2 1 4 2 5 1 5 4
2 4 1 2 4 5 2 5 2 5 2 5 1 4 2 2 2 5 4 5

(You will provide a 20x20 grid here)
```

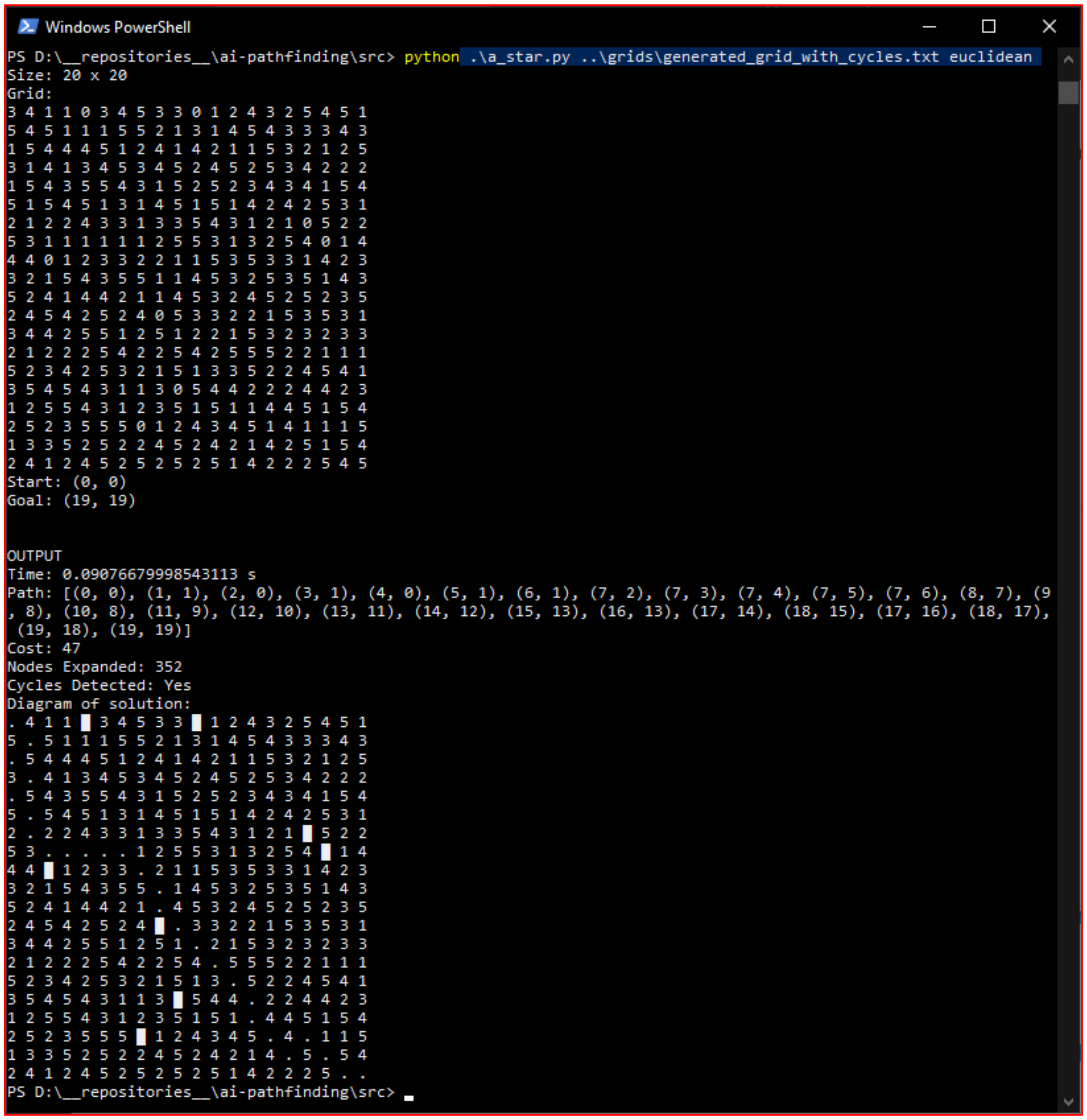
Expected Output

```
Path: [Insert path]
Cost: [Insert cost]
Nodes Expanded: [Insert number]
Cycles Detected: Yes
```

Results

The grid was generated using the provided grid-generator function. It has obstacles and cycles as well.

Running the sample test case with the Euclidean heuristic and diagonal movement:



```
PS D:\__repositories__\ai-pathfinding\src> python .\a_star.py ..\grids\generated_grid_with_cycles.txt euclidean
Size: 20 x 20
Grid:
3 4 1 1 0 3 4 5 3 3 0 1 2 4 3 2 5 4 5 1
5 4 5 1 1 1 5 5 2 1 3 1 4 5 4 3 3 3 4 3
1 5 4 4 4 5 1 2 4 1 4 2 1 1 5 3 2 1 2 5
3 1 4 1 3 4 5 3 4 5 2 4 5 2 5 3 4 2 2 2
1 5 4 3 5 5 4 3 1 5 2 5 2 3 4 3 4 1 5 4
5 1 5 4 5 1 3 1 4 5 1 5 1 4 2 4 2 5 3 1
2 1 2 2 4 3 3 1 3 3 5 4 3 1 2 1 0 5 2 2
5 3 1 1 1 1 1 1 2 5 5 3 1 3 2 5 4 0 1 4
4 4 0 1 2 3 3 2 2 1 1 5 3 5 3 3 1 4 2 3
3 2 1 5 4 3 5 5 1 1 4 5 3 2 5 3 5 1 4 3
5 2 4 1 4 4 2 1 1 4 5 3 2 4 5 2 5 2 3 5
2 4 5 4 2 5 2 4 0 5 3 3 2 2 1 5 3 5 3 1
3 4 4 2 5 5 1 2 5 1 2 2 1 5 3 2 3 2 3 3
2 1 2 2 2 5 4 2 2 5 4 2 5 5 5 2 2 1 1 1
5 2 3 4 2 5 3 2 1 5 1 3 3 5 2 2 4 5 4 1
3 5 4 5 4 3 1 1 3 0 5 4 4 2 2 2 4 4 2 3
1 2 5 5 4 3 1 2 3 5 1 5 1 1 4 4 5 1 5 4
2 5 2 3 5 5 5 0 1 2 4 3 4 5 1 4 1 1 1 5
1 3 3 5 2 5 2 2 4 5 2 4 2 1 4 2 5 1 5 4
2 4 1 2 4 5 2 5 2 5 2 5 1 4 2 2 2 5 4 5
Start: (0, 0)
Goal: (19, 19)


OUTPUT
Time: 0.09076679998543113 s
Path: [(0, 0), (1, 1), (2, 0), (3, 1), (4, 0), (5, 1), (6, 1), (7, 2), (7, 3), (7, 4), (7, 5), (7, 6), (8, 7), (9
, 8), (10, 8), (11, 9), (12, 10), (13, 11), (14, 12), (15, 13), (16, 13), (17, 14), (18, 15), (17, 16), (18, 17),
 (19, 18), (19, 19)]
Cost: 47
Nodes Expanded: 352
Cycles Detected: Yes
Diagram of solution:
. 4 1 1 █ 3 4 5 3 3 █ 1 2 4 3 2 5 4 5 1
5 . 5 1 1 1 5 5 2 1 3 1 4 5 4 3 3 3 4 3
. 5 4 4 4 5 1 2 4 1 4 2 1 1 5 3 2 1 2 5
3 . 4 1 3 4 5 3 4 5 2 4 5 2 5 3 4 2 2 2
. 5 4 3 5 5 4 3 1 5 2 5 2 3 4 3 4 1 5 4
5 . 5 4 5 1 3 1 4 5 1 5 1 4 2 4 2 5 3 1
2 . 2 2 4 3 3 1 3 3 5 4 3 1 2 1 █ 5 2 2
5 3 . . . . . 1 2 5 5 3 1 3 2 5 4 █ 1 4
4 4 █ 1 2 3 3 . 2 1 1 5 3 5 3 3 1 4 2 3
3 2 1 5 4 3 5 5 . 1 4 5 3 2 5 3 5 1 4 3
5 2 4 1 4 4 2 1 . 4 5 3 2 4 5 2 5 2 3 5
2 4 5 4 2 5 2 4 █ . 3 3 2 2 1 5 3 5 3 1
3 4 4 2 5 5 1 2 5 1 . 2 1 5 3 2 3 2 3 3
2 1 2 2 2 5 4 2 2 5 4 . 5 5 5 2 2 1 1 1
5 2 3 4 2 5 3 2 1 5 1 3 . 5 2 2 4 5 4 1
3 5 4 5 4 3 1 1 3 █ 5 4 4 . 2 2 4 4 2 3
1 2 5 5 4 3 1 2 3 5 1 5 1 . 4 4 5 1 5 4
2 5 2 3 5 5 5 █ 1 2 4 3 4 5 . 4 . 1 1 5
1 3 3 5 2 5 2 2 4 5 2 4 2 1 4 . 5 . 5 4
2 4 1 2 4 5 2 5 2 5 2 5 1 4 2 2 2 5 . .
PS D:\__repositories__\ai-pathfinding\src> _
```

Fig 2.3: Test with generated_grid_with_cycles.txt

The runtime increased by a significant amount relative to the smaller grid. This makes sense since the grid is 20x20 and many more nodes are expanded.

Task 2.4: Weighted Grid with Cycles

A 10×10 grid where each cell has a **cost between 1 and 5** for entering that cell, along with the possible cycles. Cells with **0** are impassable.

Sample Test Case

```
# input_grid_weighted_cycles.txt
1 2 3 4 5 1 1 1 1 5
```

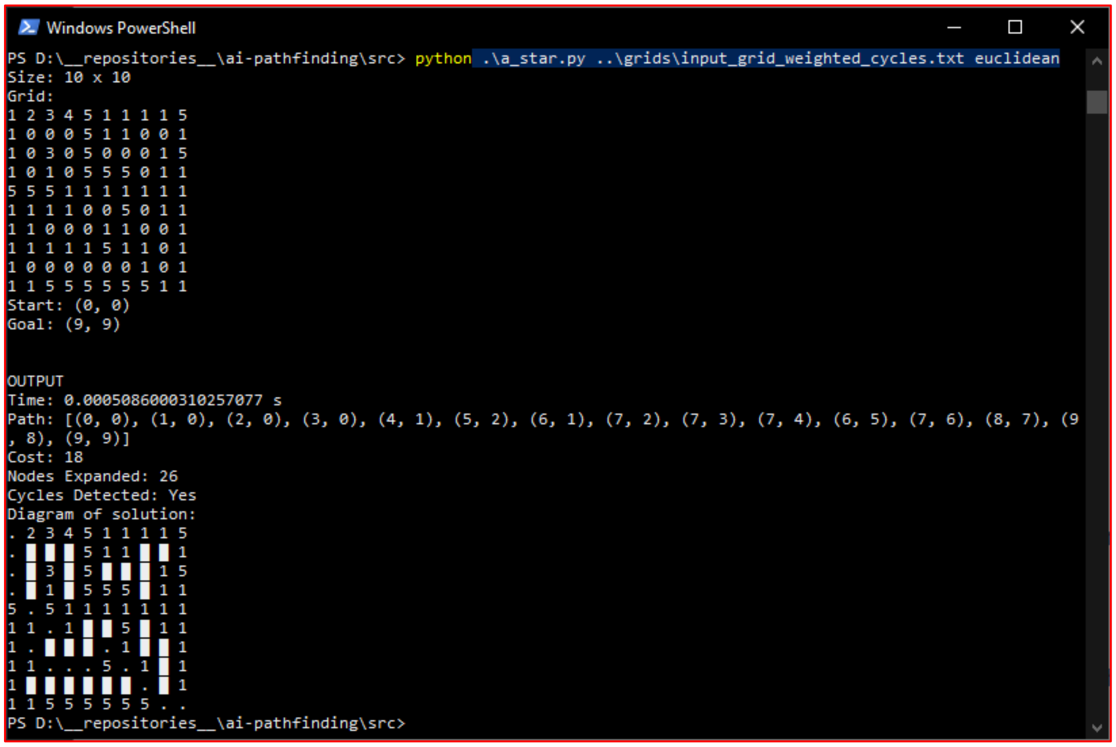
```
1 0 0 0 5 1 1 0 0 1
1 0 3 0 5 0 0 0 1 5
1 0 1 0 5 5 5 0 1 1
5 5 5 1 1 1 1 1 1 1
1 1 1 1 0 0 5 0 1 1
1 1 0 0 0 1 1 0 0 1
1 1 1 1 1 5 1 1 0 1
1 0 0 0 0 0 0 1 0 1
1 1 5 5 5 5 5 5 1 1

Start: (0,0)
Goal: (9,9)
```

Expected Output

```
Path: [(0, 0), ..., (9, 9)]
Cost: [Insert cost]
Nodes Expanded: [Insert number]
Cycles Detected: Yes
```

Results

Running the sample test case with the Euclidean heuristic and diagonal movement:



```
PS D:\__repositories__\ai-pathfinding\src> python .\a_star.py ..\grids\input_grid_weighted_cycles.txt euclidean
Size: 10 x 10
Grid:
1 2 3 4 5 1 1 1 1 5
1 0 0 0 5 1 1 0 0 1
1 0 3 0 5 0 0 0 1 5
1 0 1 0 5 5 5 0 1 1
5 5 5 1 1 1 1 1 1 1
1 1 1 1 0 0 5 0 1 1
1 1 0 0 0 1 1 0 0 1
1 1 1 1 1 5 1 1 0 1
1 0 0 0 0 0 0 1 0 1
1 1 5 5 5 5 5 5 1 1
Start: (0, 0)
Goal: (9, 9)


OUTPUT
Time: 0.0005086000310257077 s
Path: [(0, 0), (1, 0), (2, 0), (3, 0), (4, 1), (5, 2), (6, 1), (7, 2), (7, 3), (7, 4), (6, 5), (7, 6), (8, 7), (9
, 8), (9, 9)]
Cost: 18
Nodes Expanded: 26
Cycles Detected: Yes
Diagram of solution:
. 2 3 4 5 1 1 1 1 5
. █ █ █ 5 1 1 █ █ 1
. █ 3 █ 5 █ █ █ 1 5
. █ 1 █ 5 5 5 █ 1 1
5 . 5 1 1 1 1 1 1 1
1 1 . 1 █ █ 5 █ 1 1
1 . █ █ █ . 1 █ █ 1
1 1 . . . 5 . 1 █ 1
1 █ █ █ █ █ █ . █ 1
1 1 5 5 5 5 5 5 . .
PS D:\__repositories__\ai-pathfinding\src>
```

Fig 2.4: Test with input_grid_weighted_cycles.txt

The runtime is relatively higher than the cases without weighted values, but it's still low overall when compared to the larger grid for instance.

Task 3: Experiment with Heuristic Functions

Implement and experiment with two different heuristic functions for the A* algorithm:

1. **Manhattan Distance** (for grids where only horizontal and vertical movement is allowed).
2. **Euclidean Distance** (for grids where diagonal movement is allowed).

Compare the performance of A* using these two heuristics, including:

- The **number of nodes expanded** using each heuristic.
- How each heuristic affects the **path optimality** and **search efficiency**.

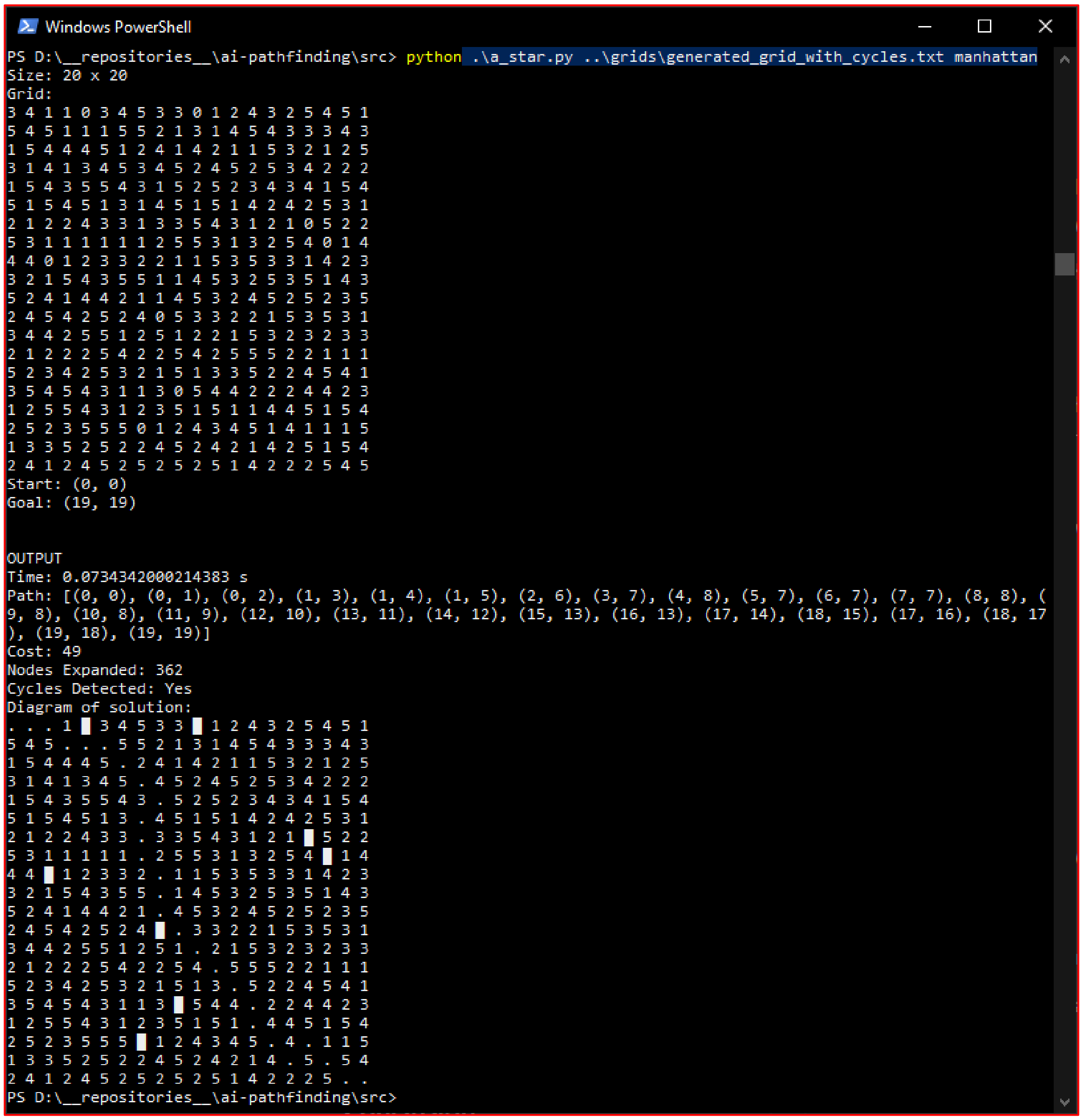
Path optimality will be affected depending on whether diagonal movement is allowed or not. If diagonal movement is allowed, then the expectation is that the Manhattan heuristic will be suboptimal since diagonal paths are ignored, but it will be optimal if only vertical and horizontal movement is allowed. In terms of search efficiency, it really depends on whether diagonal movement is allowed once again.

The Euclidean heuristic tends to be a bigger computation which could result in reduced search efficiency, but the accuracy is better when diagonal movement is allowed when compared to Manhattan. It should also lead to less nodes being expanded.

A few tests were run using both heuristics to compare the results.

In this first test, diagonal movement is allowed.

Below is the **Manhattan** heuristic:

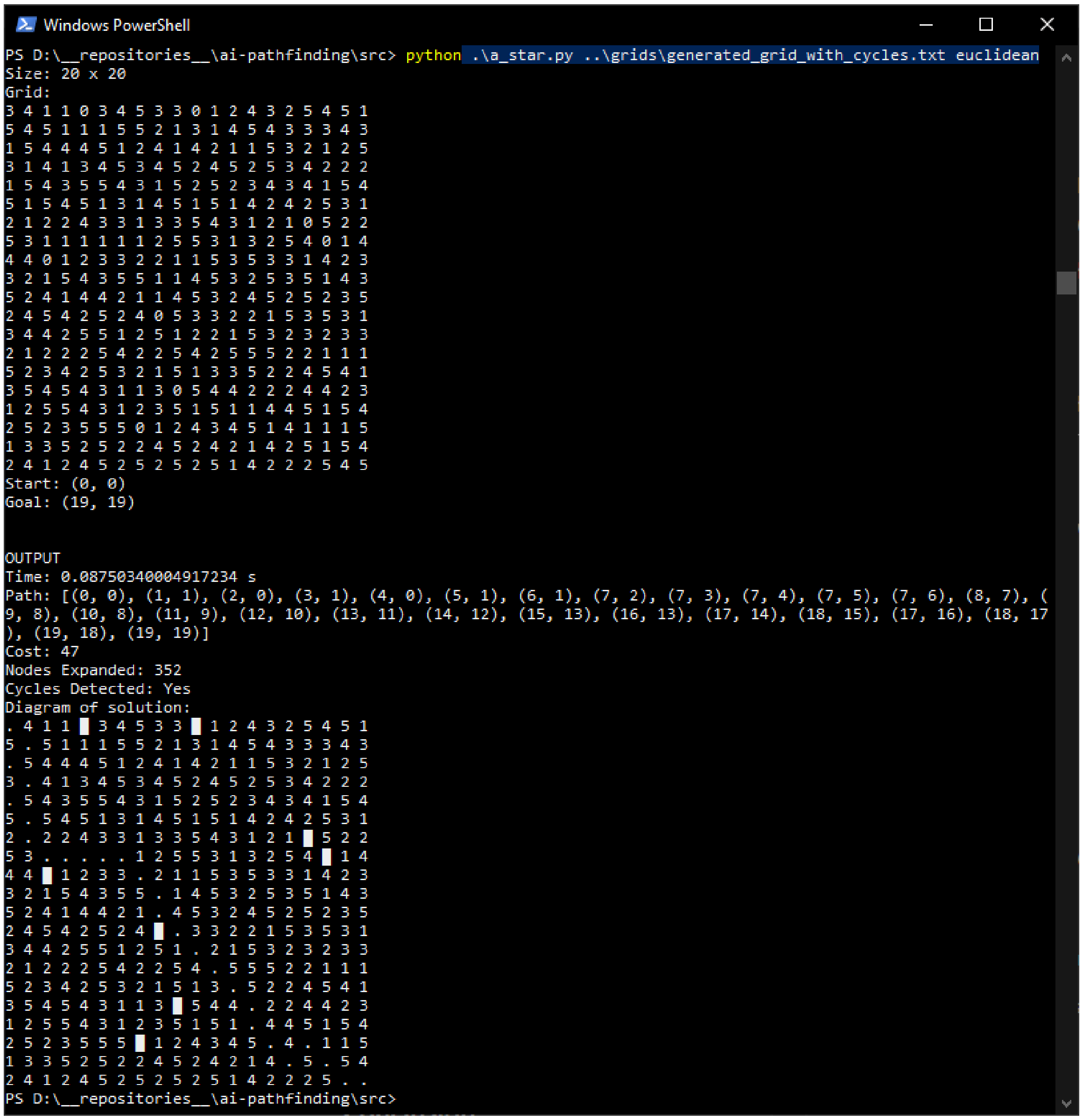


Windows PowerShell

```
PS D:\__repositories__\ai-pathfinding\src> python .\a_star.py ..\grids\generated_grid_with_cycles.txt manhattan
Size: 20 x 20
Grid:
3 4 1 1 0 3 4 5 3 3 0 1 2 4 3 2 5 4 5 1
5 4 5 1 1 1 5 5 2 1 3 1 4 5 4 3 3 3 4 3
1 5 4 4 4 5 1 2 4 1 4 2 1 1 5 3 2 1 2 5
3 1 4 1 3 4 5 3 4 5 2 4 5 2 5 3 4 2 2 2
1 5 4 3 5 5 4 3 1 5 2 5 2 3 4 3 4 1 5 4
5 1 5 4 5 1 3 1 4 5 1 5 1 4 2 4 2 5 3 1
2 1 2 2 4 3 3 1 3 3 5 4 3 1 2 1 0 5 2 2
5 3 1 1 1 1 1 1 2 5 5 3 1 3 2 5 4 0 1 4
4 4 0 1 2 3 3 2 2 1 1 5 3 5 3 3 1 4 2 3
3 2 1 5 4 3 5 5 1 1 4 5 3 2 5 3 5 1 4 3
5 2 4 1 4 4 2 1 1 4 5 3 2 4 5 2 5 2 3 5
2 4 5 4 2 5 2 4 0 5 3 3 2 2 1 5 3 5 3 1
3 4 4 2 5 5 1 2 5 1 2 2 1 5 3 2 3 2 3 3
2 1 2 2 2 5 4 2 2 5 4 2 5 5 5 2 2 1 1 1
5 2 3 4 2 5 3 2 1 5 1 3 3 5 2 2 4 5 4 1
3 5 4 5 4 3 1 1 3 0 5 4 4 2 2 2 4 4 2 3
1 2 5 5 4 3 1 2 3 5 1 5 1 1 4 4 5 1 5 4
2 5 2 3 5 5 5 0 1 2 4 3 4 5 1 4 1 1 1 5
1 3 3 5 2 5 2 2 4 5 2 4 2 1 4 2 5 1 5 4
2 4 1 2 4 5 2 5 2 5 2 5 1 4 2 2 2 5 4 5
Start: (0, 0)
Goal: (19, 19)


OUTPUT
Time: 0.0734342000214383 s
Path: [(0, 0), (0, 1), (0, 2), (1, 3), (1, 4), (1, 5), (2, 6), (3, 7), (4, 8), (5, 7), (6, 7), (7, 7), (8, 8), (
9, 8), (10, 8), (11, 9), (12, 10), (13, 11), (14, 12), (15, 13), (16, 13), (17, 14), (18, 15), (17, 16), (18, 17
), (19, 18), (19, 19)]
Cost: 49
Nodes Expanded: 362
Cycles Detected: Yes
Diagram of solution:
. . . 1 █ 3 4 5 3 3 █ 1 2 4 3 2 5 4 5 1
5 4 5 . . . 5 5 2 1 3 1 4 5 4 3 3 3 4 3
1 5 4 4 4 5 . 2 4 1 4 2 1 1 5 3 2 1 2 5
3 1 4 1 3 4 5 . 4 5 2 4 5 2 5 3 4 2 2 2
1 5 4 3 5 5 4 3 . 5 2 5 2 3 4 3 4 1 5 4
5 1 5 4 5 1 3 . 4 5 1 5 1 4 2 4 2 5 3 1
2 1 2 2 4 3 3 . 3 3 5 4 3 1 2 1 █ 5 2 2
5 3 1 1 1 1 1 . 2 5 5 3 1 3 2 5 4 █ 1 4
4 4 █ 1 2 3 3 2 . 1 1 5 3 5 3 3 1 4 2 3
3 2 1 5 4 3 5 5 . 1 4 5 3 2 5 3 5 1 4 3
5 2 4 1 4 4 2 1 . 4 5 3 2 4 5 2 5 2 3 5
2 4 5 4 2 5 2 4 █ . 3 3 2 2 1 5 3 5 3 1
3 4 4 2 5 5 1 2 5 1 . 2 1 5 3 2 3 2 3 3
2 1 2 2 2 5 4 2 2 5 4 . 5 5 5 2 2 1 1 1
5 2 3 4 2 5 3 2 1 5 1 3 . 5 2 2 4 5 4 1
3 5 4 5 4 3 1 1 3 █ 5 4 4 . 2 2 4 4 2 3
1 2 5 5 4 3 1 2 3 5 1 5 1 . 4 4 5 1 5 4
2 5 2 3 5 5 5 █ 1 2 4 3 4 5 . 4 . 1 1 5
1 3 3 5 2 5 2 2 4 5 2 4 2 1 4 . 5 . 5 4
2 4 1 2 4 5 2 5 2 5 2 5 1 4 2 2 2 5 . .
PS D:\__repositories__\ai-pathfinding\src>
```

Fig 3a: Test with generated_grid_with_cycles.txt and diagonal movement allowed

And then the **Euclidean** heuristic:



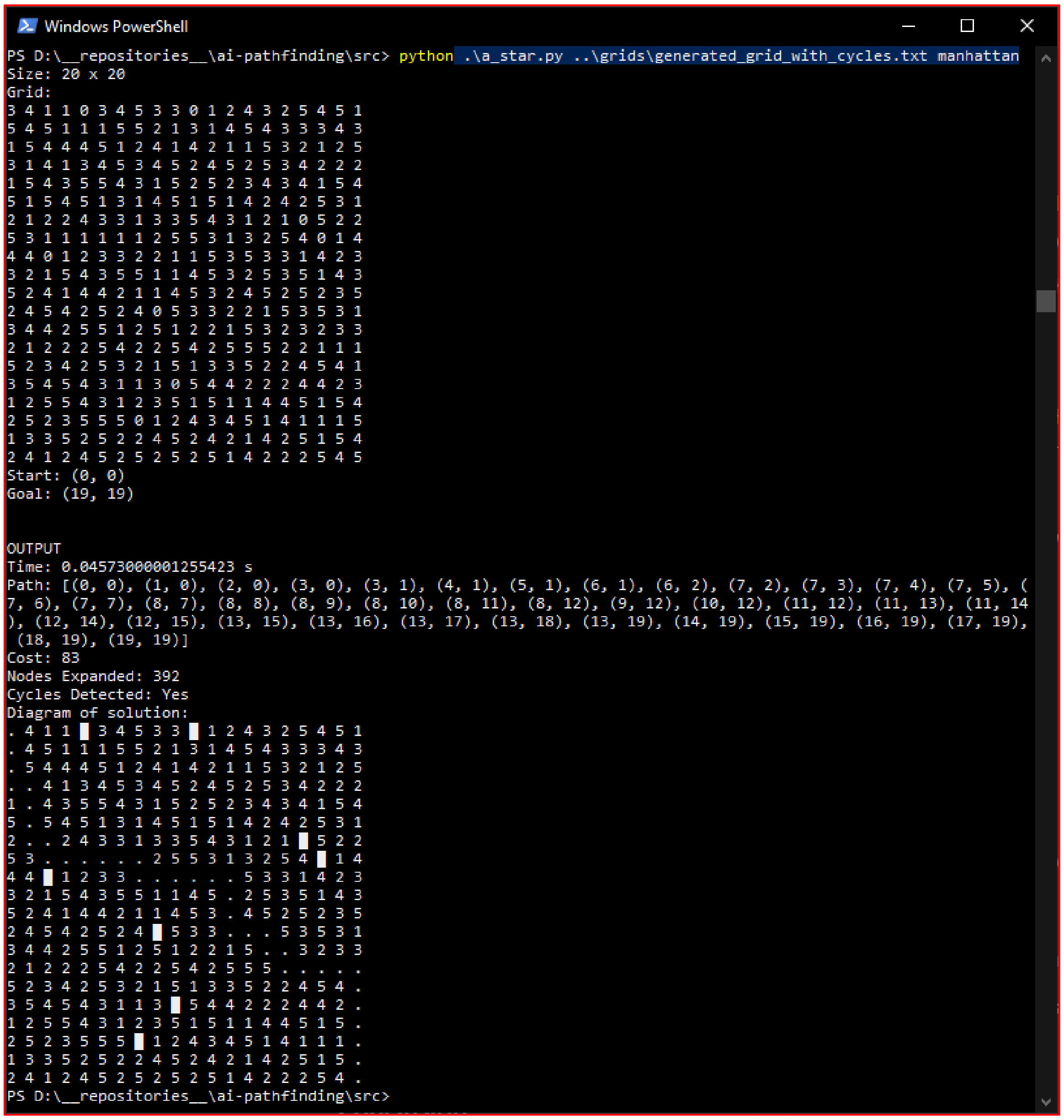
```
PS D:\__repositories__\ai-pathfinding\src> python .\a_star.py ..\grids\generated_grid_with_cycles.txt euclidean
Size: 20 x 20
Grid:
3 4 1 1 0 3 4 5 3 3 0 1 2 4 3 2 5 4 5 1
5 4 5 1 1 1 5 5 2 1 3 1 4 5 4 3 3 3 4 3
1 5 4 4 4 5 1 2 4 1 4 2 1 1 5 3 2 1 2 5
3 1 4 1 3 4 5 3 4 5 2 4 5 2 5 3 4 2 2 2
1 5 4 3 5 5 4 3 1 5 2 5 2 3 4 3 4 1 5 4
5 1 5 4 5 1 3 1 4 5 1 5 1 4 2 4 2 5 3 1
2 1 2 2 4 3 3 1 3 3 5 4 3 1 2 1 0 5 2 2
5 3 1 1 1 1 1 1 2 5 5 3 1 3 2 5 4 0 1 4
4 4 0 1 2 3 3 2 2 1 1 5 3 5 3 3 1 4 2 3
3 2 1 5 4 3 5 5 1 1 4 5 3 2 5 3 5 1 4 3
5 2 4 1 4 4 2 1 1 4 5 3 2 4 5 2 5 2 3 5
2 4 5 4 2 5 2 4 0 5 3 3 2 2 1 5 3 5 3 1
3 4 4 2 5 5 1 2 5 1 2 2 1 5 3 2 3 2 3 3
2 1 2 2 2 5 4 2 2 5 4 2 5 5 5 2 2 1 1 1
5 2 3 4 2 5 3 2 1 5 1 3 3 5 2 2 4 5 4 1
3 5 4 5 4 3 1 1 3 0 5 4 4 2 2 2 4 4 2 3
1 2 5 5 4 3 1 2 3 5 1 5 1 1 4 4 5 1 5 4
2 5 2 3 5 5 5 0 1 2 4 3 4 5 1 4 1 1 1 5
1 3 3 5 2 5 2 2 4 5 2 4 2 1 4 2 5 1 5 4
2 4 1 2 4 5 2 5 2 5 2 5 1 4 2 2 2 5 4 5
Start: (0, 0)
Goal: (19, 19)


OUTPUT
Time: 0.08750340004917234 s
Path: [(0, 0), (1, 1), (2, 0), (3, 1), (4, 0), (5, 1), (6, 1), (7, 2), (7, 3), (7, 4), (7, 5), (7, 6), (8, 7), (
9, 8), (10, 8), (11, 9), (12, 10), (13, 11), (14, 12), (15, 13), (16, 13), (17, 14), (18, 15), (17, 16), (18, 17
), (19, 18), (19, 19)]
Cost: 47
Nodes Expanded: 352
Cycles Detected: Yes
Diagram of solution:
. 4 1 1 █ 3 4 5 3 3 █ 1 2 4 3 2 5 4 5 1
5 . 5 1 1 1 5 5 2 1 3 1 4 5 4 3 3 3 4 3
. 5 4 4 4 5 1 2 4 1 4 2 1 1 5 3 2 1 2 5
3 . 4 1 3 4 5 3 4 5 2 4 5 2 5 3 4 2 2 2
. 5 4 3 5 5 4 3 1 5 2 5 2 3 4 3 4 1 5 4
5 . 5 4 5 1 3 1 4 5 1 5 1 4 2 4 2 5 3 1
2 . 2 2 4 3 3 1 3 3 5 4 3 1 2 1 █ 5 2 2
5 3 . . . . . 1 2 5 5 3 1 3 2 5 4 █ 1 4
4 4 █ 1 2 3 3 . 2 1 1 5 3 5 3 3 1 4 2 3
3 2 1 5 4 3 5 5 . 1 4 5 3 2 5 3 5 1 4 3
5 2 4 1 4 4 2 1 . 4 5 3 2 4 5 2 5 2 3 5
2 4 5 4 2 5 2 4 █ . 3 3 2 2 1 5 3 5 3 1
3 4 4 2 5 5 1 2 5 1 . 2 1 5 3 2 3 2 3 3
2 1 2 2 2 5 4 2 2 5 4 . 5 5 5 2 2 1 1 1
5 2 3 4 2 5 3 2 1 5 1 3 . 5 2 2 4 5 4 1
3 5 4 5 4 3 1 1 3 █ 5 4 4 . 2 2 4 4 2 3
1 2 5 5 4 3 1 2 3 5 1 5 1 . 4 4 5 1 5 4
2 5 2 3 5 5 5 █ 1 2 4 3 4 5 . 4 . 1 1 5
1 3 3 5 2 5 2 2 4 5 2 4 2 1 4 . 5 . 5 4
2 4 1 2 4 5 2 5 2 5 2 5 1 4 2 2 2 5 . .
PS D:\__repositories__\ai-pathfinding\src>
```

Fig 3b: Test with generated_grid_with_cycles.txt and diagonal movement allowed

They are very close in time, but the Manhattan heuristic is only slightly better even though more nodes were expanded. The expectation was that the Euclidean heuristic would outperform, but this is not the case as can be seen. Also consider that the Manhattan heuristic is admissible in grids where only horizontal and vertical movement are allowed.

Next, we can remove diagonal movements and only allow vertical and horizontal movements.

First up is the **Manhattan** heuristic:

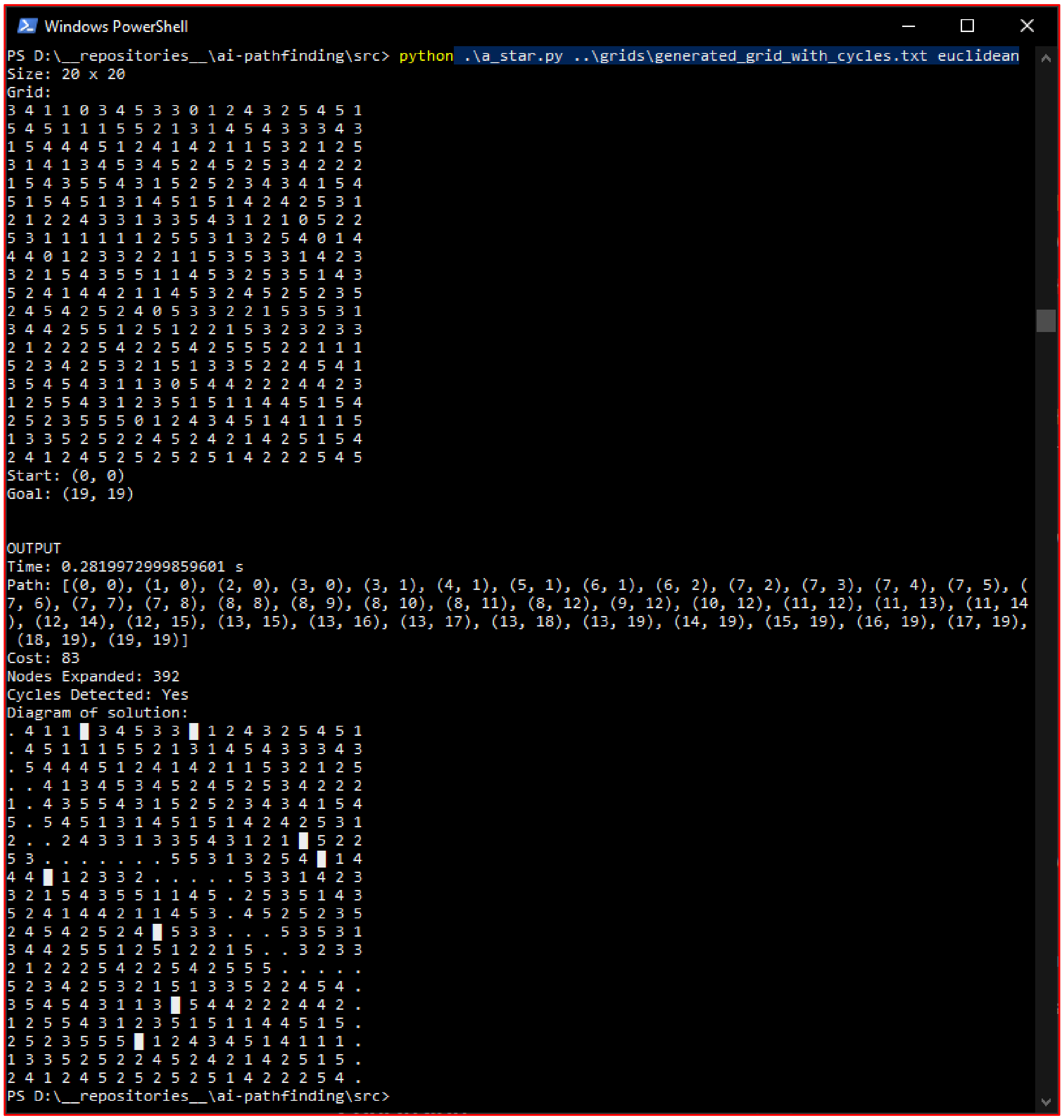


```
Windows PowerShell
PS D:\__repositories__\ai-pathfinding\src> python .\a_star.py ..\grids\generated_grid_with_cycles.txt manhattan
Size: 20 x 20
Grid:
3 4 1 1 0 3 4 5 3 3 0 1 2 4 3 2 5 4 5 1
5 4 5 1 1 1 5 5 2 1 3 1 4 5 4 3 3 3 4 3
1 5 4 4 4 5 1 2 4 1 4 2 1 1 5 3 2 1 2 5
3 1 4 1 3 4 5 3 4 5 2 4 5 2 5 3 4 2 2 2
1 5 4 3 5 5 4 3 1 5 2 5 2 3 4 3 4 1 5 4
5 1 5 4 5 1 3 1 4 5 1 5 1 4 2 4 2 5 3 1
2 1 2 2 4 3 3 1 3 3 5 4 3 1 2 1 0 5 2 2
5 3 1 1 1 1 1 1 2 5 5 3 1 3 2 5 4 0 1 4
4 4 0 1 2 3 3 2 2 1 1 5 3 5 3 3 1 4 2 3
3 2 1 5 4 3 5 5 1 1 4 5 3 2 5 3 5 1 4 3
5 2 4 1 4 4 2 1 1 4 5 3 2 4 5 2 5 2 3 5
2 4 5 4 2 5 2 4 0 5 3 3 2 2 1 5 3 5 3 1
3 4 4 2 5 5 1 2 5 1 2 2 1 5 3 2 3 2 3 3
2 1 2 2 2 5 4 2 2 5 4 2 5 5 5 2 2 1 1 1
5 2 3 4 2 5 3 2 1 5 1 3 3 5 2 2 4 5 4 1
3 5 4 5 4 3 1 1 3 0 5 4 4 2 2 2 4 4 2 3
1 2 5 5 4 3 1 2 3 5 1 5 1 1 4 4 5 1 5 4
2 5 2 3 5 5 5 0 1 2 4 3 4 5 1 4 1 1 1 5
1 3 3 5 2 5 2 2 4 5 2 4 2 1 4 2 5 1 5 4
2 4 1 2 4 5 2 5 2 5 2 5 1 4 2 2 2 5 4 5
Start: (0, 0)
Goal: (19, 19)


OUTPUT
Time: 0.04573000001255423 s
Path: [(0, 0), (1, 0), (2, 0), (3, 0), (3, 1), (4, 1), (5, 1), (6, 1), (6, 2), (7, 2), (7, 3), (7, 4), (7, 5), (
7, 6), (7, 7), (8, 7), (8, 8), (8, 9), (8, 10), (8, 11), (8, 12), (9, 12), (10, 12), (11, 12), (11, 13), (11, 14
), (12, 14), (12, 15), (13, 15), (13, 16), (13, 17), (13, 18), (13, 19), (14, 19), (15, 19), (16, 19), (17, 19),
 (18, 19), (19, 19)]
Cost: 83
Nodes Expanded: 392
Cycles Detected: Yes
Diagram of solution:
. 4 1 1 █ 3 4 5 3 3 █ 1 2 4 3 2 5 4 5 1
. 4 5 1 1 1 5 5 2 1 3 1 4 5 4 3 3 3 4 3
. 5 4 4 4 5 1 2 4 1 4 2 1 1 5 3 2 1 2 5
. . 4 1 3 4 5 3 4 5 2 4 5 2 5 3 4 2 2 2
1 . 4 3 5 5 4 3 1 5 2 5 2 3 4 3 4 1 5 4
5 . 5 4 5 1 3 1 4 5 1 5 1 4 2 4 2 5 3 1
2 . . 2 4 3 3 1 3 3 5 4 3 1 2 1 █ 5 2 2
5 3 . . . . . . 2 5 5 3 1 3 2 5 4 █ 1 4
4 4 █ 1 2 3 3 . . . . . . 5 3 3 1 4 2 3
3 2 1 5 4 3 5 5 1 1 4 5 . 2 5 3 5 1 4 3
5 2 4 1 4 4 2 1 1 4 5 3 . 4 5 2 5 2 3 5
2 4 5 4 2 5 2 4 █ 5 3 3 . . . 5 3 5 3 1
3 4 4 2 5 5 1 2 5 1 2 2 1 5 . . 3 2 3 3
2 1 2 2 2 5 4 2 2 5 4 2 5 5 5 . . . . .
5 2 3 4 2 5 3 2 1 5 1 3 3 5 2 2 4 5 4 .
3 5 4 5 4 3 1 1 3 █ 5 4 4 2 2 2 4 4 2 .
1 2 5 5 4 3 1 2 3 5 1 5 1 1 4 4 5 1 5 .
2 5 2 3 5 5 5 █ 1 2 4 3 4 5 1 4 1 1 1 .
1 3 3 5 2 5 2 2 4 5 2 4 2 1 4 2 5 1 5 .
2 4 1 2 4 5 2 5 2 5 2 5 1 4 2 2 2 5 4 .
PS D:\__repositories__\ai-pathfinding\src>
```

Fig 3c: Test with generated_grid_with_cycles.txt and diagonal movement not allowed

When diagonal movement is removed, the Manhattan heuristic has improved performance, but more nodes are expanded it appears when compared to the case where diagonal movement is allowed.

Finally, the **Euclidean** heuristic once again:



```
PS D:\__repositories__\ai-pathfinding\src> python .\a_star.py ..\grids\generated_grid_with_cycles.txt euclidean
Size: 20 x 20
Grid:
3 4 1 1 0 3 4 5 3 3 0 1 2 4 3 2 5 4 5 1
5 4 5 1 1 1 5 5 2 1 3 1 4 5 4 3 3 3 4 3
1 5 4 4 4 5 1 2 4 1 4 2 1 1 5 3 2 1 2 5
3 1 4 1 3 4 5 3 4 5 2 4 5 2 5 3 4 2 2 2
1 5 4 3 5 5 4 3 1 5 2 5 2 3 4 3 4 1 5 4
5 1 5 4 5 1 3 1 4 5 1 5 1 4 2 4 2 5 3 1
2 1 2 2 4 3 3 1 3 3 5 4 3 1 2 1 0 5 2 2
5 3 1 1 1 1 1 1 2 5 5 3 1 3 2 5 4 0 1 4
4 4 0 1 2 3 3 2 2 1 1 5 3 5 3 3 1 4 2 3
3 2 1 5 4 3 5 5 1 1 4 5 3 2 5 3 5 1 4 3
5 2 4 1 4 4 2 1 1 4 5 3 2 4 5 2 5 2 3 5
2 4 5 4 2 5 2 4 0 5 3 3 2 2 1 5 3 5 3 1
3 4 4 2 5 5 1 2 5 1 2 2 1 5 3 2 3 2 3 3
2 1 2 2 2 5 4 2 2 5 4 2 5 5 5 2 2 1 1 1
5 2 3 4 2 5 3 2 1 5 1 3 3 5 2 2 4 5 4 1
3 5 4 5 4 3 1 1 3 0 5 4 4 2 2 2 4 4 2 3
1 2 5 5 4 3 1 2 3 5 1 5 1 1 4 4 5 1 5 4
2 5 2 3 5 5 5 0 1 2 4 3 4 5 1 4 1 1 1 5
1 3 3 5 2 5 2 2 4 5 2 4 2 1 4 2 5 1 5 4
2 4 1 2 4 5 2 5 2 5 2 5 1 4 2 2 2 5 4 5
Start: (0, 0)
Goal: (19, 19)


OUTPUT
Time: 0.2819972999859601 s
Path: [(0, 0), (1, 0), (2, 0), (3, 0), (3, 1), (4, 1), (5, 1), (6, 1), (6, 2), (7, 2), (7, 3), (7, 4), (7, 5), (
7, 6), (7, 7), (7, 8), (8, 8), (8, 9), (8, 10), (8, 11), (8, 12), (9, 12), (10, 12), (11, 12), (11, 13), (11, 14
), (12, 14), (12, 15), (13, 15), (13, 16), (13, 17), (13, 18), (13, 19), (14, 19), (15, 19), (16, 19), (17, 19),
 (18, 19), (19, 19)]
Cost: 83
Nodes Expanded: 392
Cycles Detected: Yes
Diagram of solution:
. 4 1 1 █ 3 4 5 3 3 █ 1 2 4 3 2 5 4 5 1
. 4 5 1 1 1 5 5 2 1 3 1 4 5 4 3 3 3 4 3
. 5 4 4 4 5 1 2 4 1 4 2 1 1 5 3 2 1 2 5
. . 4 1 3 4 5 3 4 5 2 4 5 2 5 3 4 2 2 2
1 . 4 3 5 5 4 3 1 5 2 5 2 3 4 3 4 1 5 4
5 . 5 4 5 1 3 1 4 5 1 5 1 4 2 4 2 5 3 1
2 . . 2 4 3 3 1 3 3 5 4 3 1 2 1 █ 5 2 2
5 3 . . . . . . . 5 5 3 1 3 2 5 4 █ 1 4
4 4 █ 1 2 3 3 2 . . . . . 5 3 3 1 4 2 3
3 2 1 5 4 3 5 5 1 1 4 5 . 2 5 3 5 1 4 3
5 2 4 1 4 4 2 1 1 4 5 3 . 4 5 2 5 2 3 5
2 4 5 4 2 5 2 4 █ 5 3 3 . . . 5 3 5 3 1
3 4 4 2 5 5 1 2 5 1 2 2 1 5 . . 3 2 3 3
2 1 2 2 2 5 4 2 2 5 4 2 5 5 5 . . . . .
5 2 3 4 2 5 3 2 1 5 1 3 3 5 2 2 4 5 4 .
3 5 4 5 4 3 1 1 3 █ 5 4 4 2 2 2 4 4 2 .
1 2 5 5 4 3 1 2 3 5 1 5 1 1 4 4 5 1 5 .
2 5 2 3 5 5 5 █ 1 2 4 3 4 5 1 4 1 1 1 .
1 3 3 5 2 5 2 2 4 5 2 4 2 1 4 2 5 1 5 .
2 4 1 2 4 5 2 5 2 5 2 5 1 4 2 2 2 5 4 .
PS D:\__repositories__\ai-pathfinding\src>
```

Fig 3d: Test with generated_grid_with_cycles.txt and diagonal movement not allowed

The performance of the A-star algorithm with the Euclidean heuristic when diagonal movement is not allowed is greatly reduced as can be seen from the time elapsed. This has to do with Euclidean distance being computed but only horizontal and vertical movements are allowed. More nodes are also expanded when compared to the prior case where diagonal movement was allowed. Consider that the Euclidean heuristic is admissible but efficiency is lowered.

Task 4: Comparison with Dijkstra's Algorithm

Implement **Dijkstra's algorithm** and run it on the same scenarios as in Task 2.

Comparison

Compare the performance of A* and Dijkstra's algorithms in terms of:

1. **Number of nodes expanded**.
2. **Total search time**.
3. **Path optimality**.

Task 4.1: Basic Grid

A 10×10 grid with no obstacles. The start is at the top-left, and the goal is at the bottom-right.

Sample Test Case

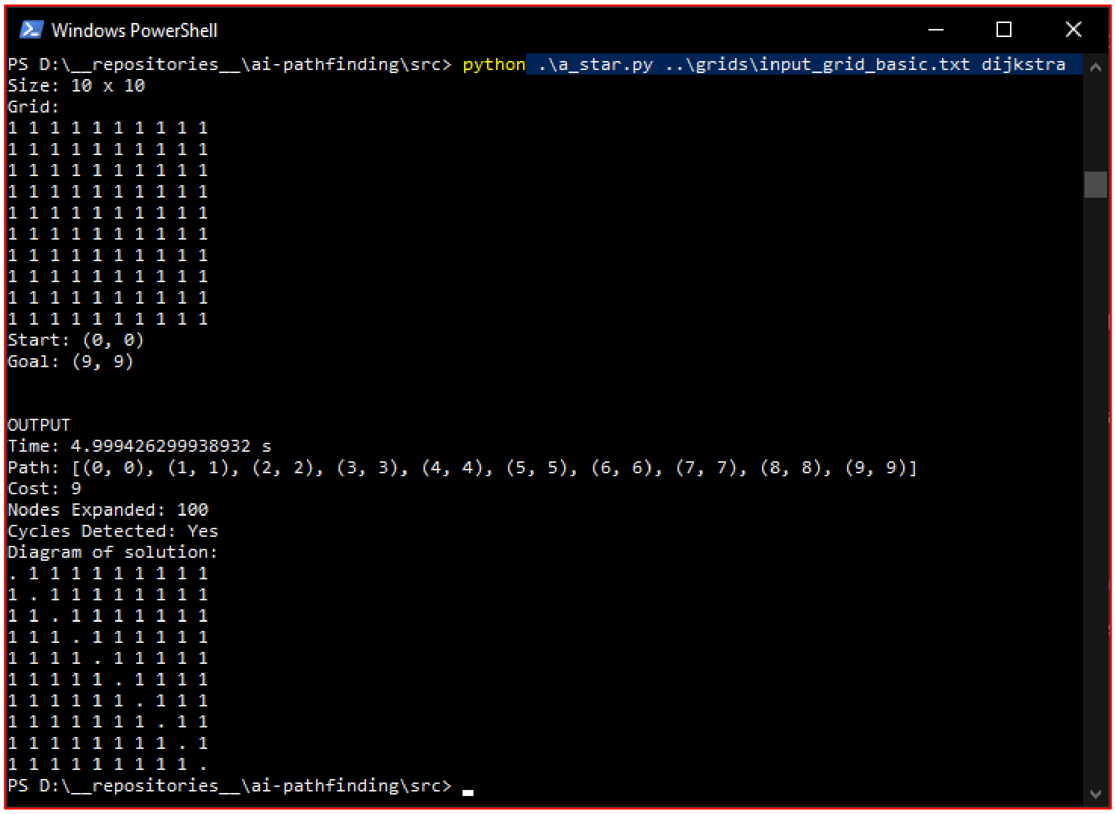
```
# input_grid_basic.txt
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1

Start: (0,0)
Goal: (9,9)
```

Expected Output

```
Path: [(0, 0), (1, 1), ..., (9, 9)]
Cost: 18
Nodes Expanded: [Insert number]
Cycles Detected: No
```

Results

Running the sample test case using Dijkstra's algorithm yields:


Windows PowerShell

```
PS D:\__repositories__\ai-pathfinding\src> python .\a_star.py ..\grids\input_grid_basic.txt dijkstra
Size: 10 x 10
Grid:
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
Start: (0, 0)
Goal: (9, 9)


OUTPUT
Time: 4.999426299938932 s
Path: [(0, 0), (1, 1), (2, 2), (3, 3), (4, 4), (5, 5), (6, 6), (7, 7), (8, 8), (9, 9)]
Cost: 9
Nodes Expanded: 100
Cycles Detected: Yes
Diagram of solution:
. 1 1 1 1 1 1 1 1 1
1 . 1 1 1 1 1 1 1 1
1 1 . 1 1 1 1 1 1 1
1 1 1 . 1 1 1 1 1 1
1 1 1 1 . 1 1 1 1 1
1 1 1 1 1 . 1 1 1 1
1 1 1 1 1 1 . 1 1 1
1 1 1 1 1 1 1 . 1 1
1 1 1 1 1 1 1 1 . 1
1 1 1 1 1 1 1 1 1 .
PS D:\__repositories__\ai-pathfinding\src> _
```

Fig 4.1: Test with input_grid_basic.txt

It takes quite a bit of time (approximately 5 s) to complete. It's clear why as 100 nodes were expanded.

Task 4.2: Grid with Obstacles and Cycles

A 10×10 grid with obstacles and edges forming cycles. The start and goal positions must not be blocked.

Sample Test Case

```
# input_grid_obstacles_cycles.txt
1 1 1 1 1 1 1 1 1 1
1 0 0 0 1 1 1 0 0 1
1 0 1 0 1 0 0 0 1 1
1 0 1 0 1 1 1 0 1 1
1 0 1 1 1 1 1 1 1 1
1 1 1 1 0 0 1 0 1 1
1 1 0 0 0 1 1 0 0 1
1 1 1 1 1 1 1 1 0 1
```

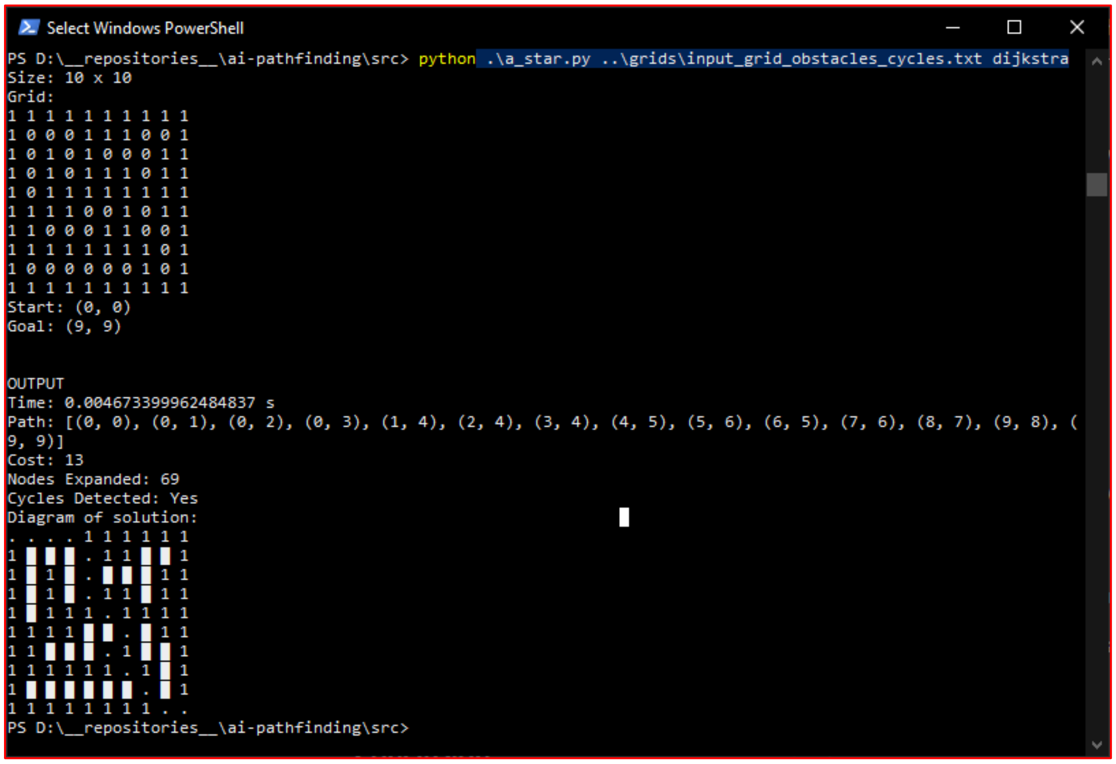
```
1 0 0 0 0 0 0 1 0 1
1 1 1 1 1 1 1 1 1 1

Start: (0,0)
Goal: (9,9)
```

Expected Output

```
Path: [(0, 0), ..., (9, 9)]
Cost: [Insert cost]
Nodes Expanded: [Insert number]
Cycles Detected: Yes
```

Results

Running the sample test case with Dijkstra's Algorithm yields:



```
PS D:\__repositories__\ai-pathfinding\src> python .\a_star.py ..\grids\input_grid_obstacles_cycles.txt dijkstra
Size: 10 x 10
Grid:
1 1 1 1 1 1 1 1 1 1
1 0 0 0 1 1 1 0 0 1
1 0 1 0 1 0 0 0 1 1
1 0 1 0 1 1 1 0 1 1
1 0 1 1 1 1 1 1 1 1
1 1 1 1 0 0 1 0 1 1
1 1 0 0 0 1 1 0 0 1
1 1 1 1 1 1 1 1 0 1
1 0 0 0 0 0 0 1 0 1
1 1 1 1 1 1 1 1 1 1
Start: (0, 0)
Goal: (9, 9)


OUTPUT
Time: 0.004673399962484837 s
Path: [(0, 0), (0, 1), (0, 2), (0, 3), (1, 4), (2, 4), (3, 4), (4, 5), (5, 6), (6, 5), (7, 6), (8, 7), (9, 8), (
9, 9)]
Cost: 13
Nodes Expanded: 69
Cycles Detected: Yes
Diagram of solution:
. . . . 1 1 1 1 1 1
1 █ █ █ . 1 1 █ █ 1
1 █ 1 █ . █ █ █ 1 1
1 █ 1 █ . 1 1 █ 1 1
1 █ 1 1 1 . 1 1 1 1
1 1 1 1 █ █ . █ 1 1
1 1 █ █ █ . 1 █ █ 1
1 1 1 1 1 1 . 1 █ 1
1 █ █ █ █ █ █ . █ 1
1 1 1 1 1 1 1 1 . .
PS D:\__repositories__\ai-pathfinding\src>
```

Fig 4.2: Test with input_grid_obstacles_cycles.txt

The runtime went down as obstacles are added.

Task 4.3: Larger Grid with Obstacles and Cycles

A 20×20 grid with multiple paths that form loops.

Sample Test Case

```
# input_grid_basic.txt
3 4 1 1 0 3 4 5 3 3 0 1 2 4 3 2 5 4 5 1
5 4 5 1 1 1 5 5 2 1 3 1 4 5 4 3 3 3 4 3
1 5 4 4 4 5 1 2 4 1 4 2 1 1 5 3 2 1 2 5
3 1 4 1 3 4 5 3 4 5 2 4 5 2 5 3 4 2 2 2
1 5 4 3 5 5 4 3 1 5 2 5 2 3 4 3 4 1 5 4
5 1 5 4 5 1 3 1 4 5 1 5 1 4 2 4 2 5 3 1
2 1 2 2 4 3 3 1 3 3 5 4 3 1 2 1 0 5 2 2
5 3 1 1 1 1 1 1 2 5 5 3 1 3 2 5 4 0 1 4
4 4 0 1 2 3 3 2 2 1 1 5 3 5 3 3 1 4 2 3
3 2 1 5 4 3 5 5 1 1 4 5 3 2 5 3 5 1 4 3
5 2 4 1 4 4 2 1 1 4 5 3 2 4 5 2 5 2 3 5
2 4 5 4 2 5 2 4 0 5 3 3 2 2 1 5 3 5 3 1
3 4 4 2 5 5 1 2 5 1 2 2 1 5 3 2 3 2 3 3
2 1 2 2 2 5 4 2 2 5 4 2 5 5 5 2 2 1 1 1
5 2 3 4 2 5 3 2 1 5 1 3 3 5 2 2 4 5 4 1
3 5 4 5 4 3 1 1 3 0 5 4 4 2 2 2 4 4 2 3
1 2 5 5 4 3 1 2 3 5 1 5 1 1 4 4 5 1 5 4
2 5 2 3 5 5 5 0 1 2 4 3 4 5 1 4 1 1 1 5
1 3 3 5 2 5 2 2 4 5 2 4 2 1 4 2 5 1 5 4
2 4 1 2 4 5 2 5 2 5 2 5 1 4 2 2 2 5 4 5

(You will provide a 20x20 grid here)
```

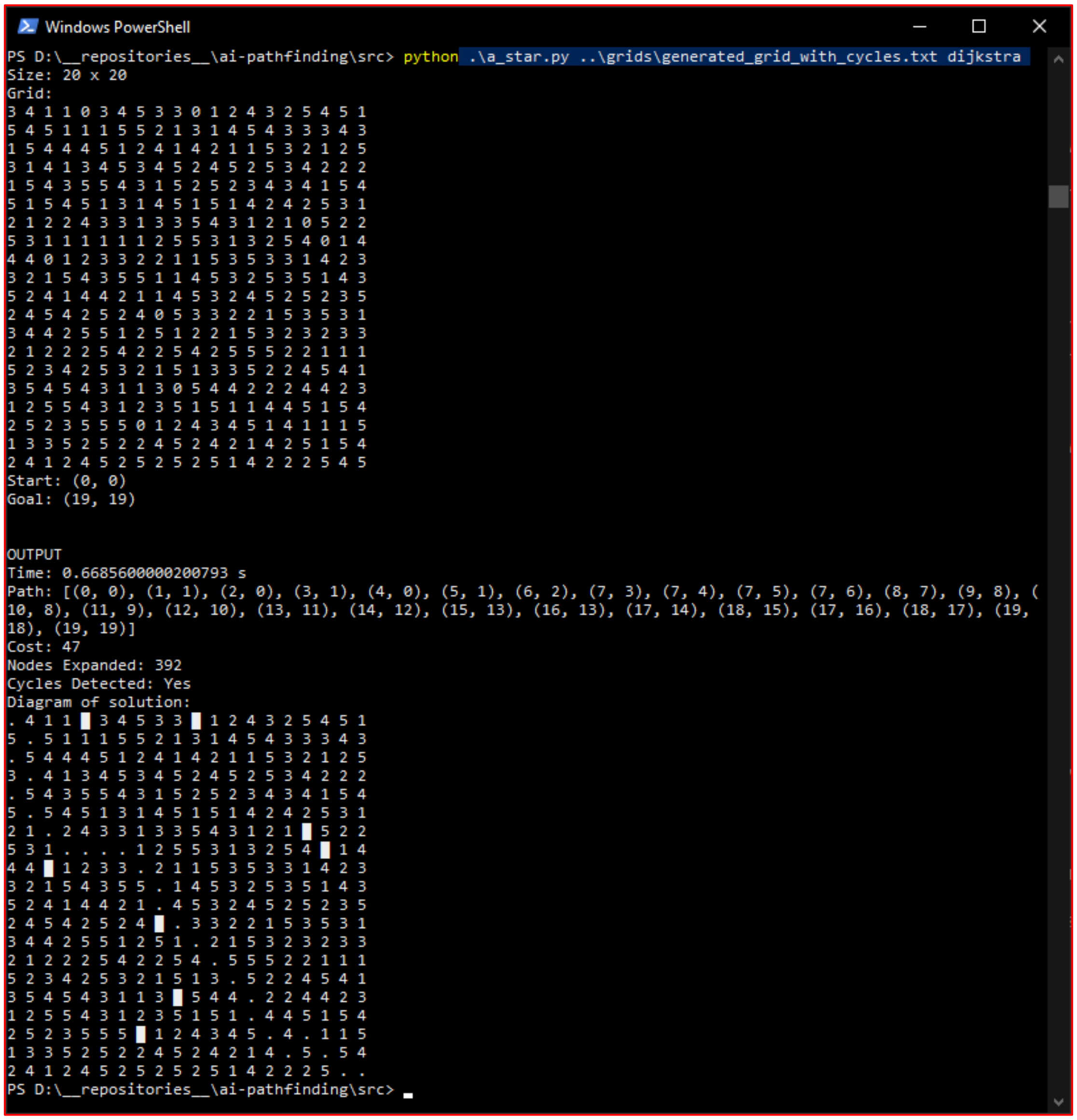
Expected Output

```
Path: [Insert path]
Cost: [Insert cost]
Nodes Expanded: [Insert number]
Cycles Detected: Yes
```

Results

The grid was generated using the provided grid-generator function. It has obstacles and cycles as well.

Running the sample test case with Dijkstra's algorithm yields:



```
PS D:\__repositories__\ai-pathfinding\src> python .\a_star.py ..\grids\generated_grid_with_cycles.txt dijkstra
Size: 20 x 20
Grid:
3 4 1 1 0 3 4 5 3 3 0 1 2 4 3 2 5 4 5 1
5 4 5 1 1 1 5 5 2 1 3 1 4 5 4 3 3 3 4 3
1 5 4 4 4 5 1 2 4 1 4 2 1 1 5 3 2 1 2 5
3 1 4 1 3 4 5 3 4 5 2 4 5 2 5 3 4 2 2 2
1 5 4 3 5 5 4 3 1 5 2 5 2 3 4 3 4 1 5 4
5 1 5 4 5 1 3 1 4 5 1 5 1 4 2 4 2 5 3 1
2 1 2 2 4 3 3 1 3 3 5 4 3 1 2 1 0 5 2 2
5 3 1 1 1 1 1 1 2 5 5 3 1 3 2 5 4 0 1 4
4 4 0 1 2 3 3 2 2 1 1 5 3 5 3 3 1 4 2 3
3 2 1 5 4 3 5 5 1 1 4 5 3 2 5 3 5 1 4 3
5 2 4 1 4 4 2 1 1 4 5 3 2 4 5 2 5 2 3 5
2 4 5 4 2 5 2 4 0 5 3 3 2 2 1 5 3 5 3 1
3 4 4 2 5 5 1 2 5 1 2 2 1 5 3 2 3 2 3 3
2 1 2 2 2 5 4 2 2 5 4 2 5 5 5 2 2 1 1 1
5 2 3 4 2 5 3 2 1 5 1 3 3 5 2 2 4 5 4 1
3 5 4 5 4 3 1 1 3 0 5 4 4 2 2 2 4 4 2 3
1 2 5 5 4 3 1 2 3 5 1 5 1 1 4 4 5 1 5 4
2 5 2 3 5 5 5 0 1 2 4 3 4 5 1 4 1 1 1 5
1 3 3 5 2 5 2 2 4 5 2 4 2 1 4 2 5 1 5 4
2 4 1 2 4 5 2 5 2 5 2 5 1 4 2 2 2 5 4 5
Start: (0, 0)
Goal: (19, 19)


OUTPUT
Time: 0.6685600000200793 s
Path: [(0, 0), (1, 1), (2, 0), (3, 1), (4, 0), (5, 1), (6, 2), (7, 3), (7, 4), (7, 5), (7, 6), (8, 7), (9, 8), (
10, 8), (11, 9), (12, 10), (13, 11), (14, 12), (15, 13), (16, 13), (17, 14), (18, 15), (17, 16), (18, 17), (19,
18), (19, 19)]
Cost: 47
Nodes Expanded: 392
Cycles Detected: Yes
Diagram of solution:
. 4 1 1 █ 3 4 5 3 3 █ 1 2 4 3 2 5 4 5 1
5 . 5 1 1 1 5 5 2 1 3 1 4 5 4 3 3 3 4 3
. 5 4 4 4 5 1 2 4 1 4 2 1 1 5 3 2 1 2 5
3 . 4 1 3 4 5 3 4 5 2 4 5 2 5 3 4 2 2 2
. 5 4 3 5 5 4 3 1 5 2 5 2 3 4 3 4 1 5 4
5 . 5 4 5 1 3 1 4 5 1 5 1 4 2 4 2 5 3 1
2 1 . 2 4 3 3 1 3 3 5 4 3 1 2 1 █ 5 2 2
5 3 1 . . . . 1 2 5 5 3 1 3 2 5 4 █ 1 4
4 4 █ 1 2 3 3 . 2 1 1 5 3 5 3 3 1 4 2 3
3 2 1 5 4 3 5 5 . 1 4 5 3 2 5 3 5 1 4 3
5 2 4 1 4 4 2 1 . 4 5 3 2 4 5 2 5 2 3 5
2 4 5 4 2 5 2 4 █ . 3 3 2 2 1 5 3 5 3 1
3 4 4 2 5 5 1 2 5 1 . 2 1 5 3 2 3 2 3 3
2 1 2 2 2 5 4 2 2 5 4 . 5 5 5 2 2 1 1 1
5 2 3 4 2 5 3 2 1 5 1 3 . 5 2 2 4 5 4 1
3 5 4 5 4 3 1 1 3 █ 5 4 4 . 2 2 4 4 2 3
1 2 5 5 4 3 1 2 3 5 1 5 1 . 4 4 5 1 5 4
2 5 2 3 5 5 5 █ 1 2 4 3 4 5 . 4 . 1 1 5
1 3 3 5 2 5 2 2 4 5 2 4 2 1 4 . 5 . 5 4
2 4 1 2 4 5 2 5 2 5 2 5 1 4 2 2 2 5 . .
PS D:\__repositories__\ai-pathfinding\src>
```

Fig 2.3: Test with generated_grid_with_cycles.txt

The runtime increased by a significant amount relative to the smaller grid. This makes sense since the grid is 20x20 and many more nodes are expanded as the heuristic is zero.

Task 4.4: Weighted Grid with Cycles

A 10×10 grid where each cell has a **cost between 1 and 5** for entering that cell, along with the possible cycles. Cells with **0** are impassable.

Sample Test Case

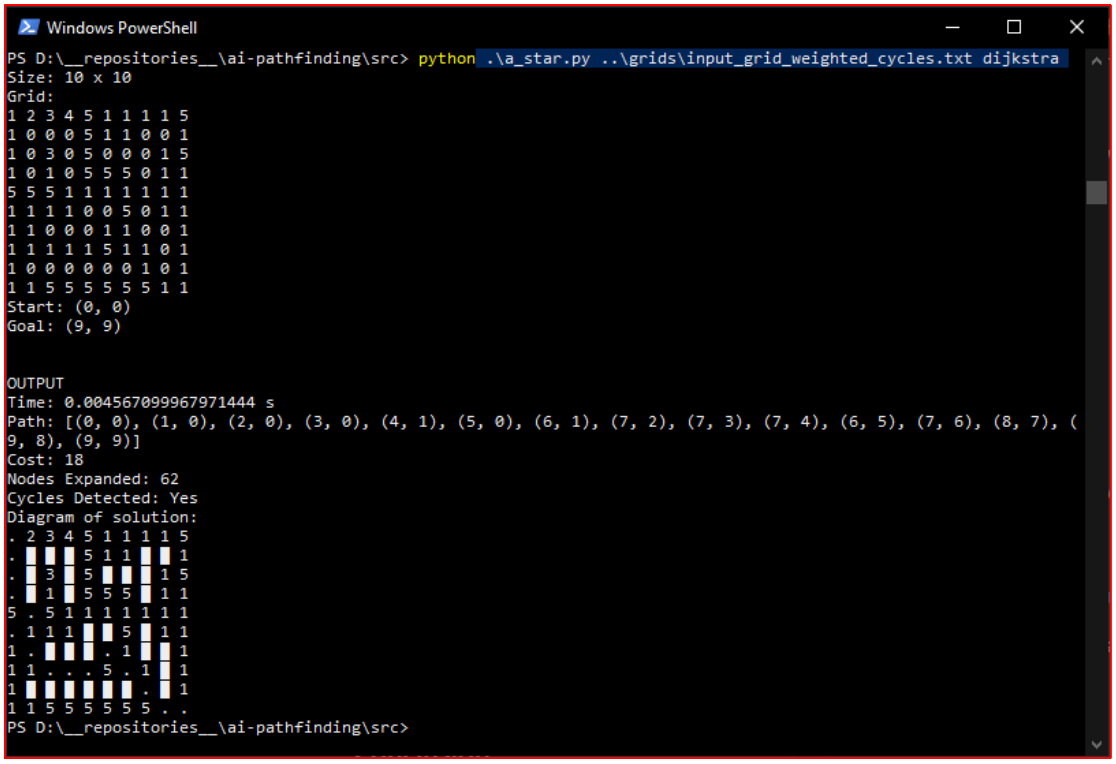
```
# input_grid_weighted_cycles.txt
1 2 3 4 5 1 1 1 1 5
1 0 0 0 5 1 1 0 0 1
1 0 3 0 5 0 0 0 1 5
1 0 1 0 5 5 5 0 1 1
5 5 5 1 1 1 1 1 1 1
1 1 1 1 0 0 5 0 1 1
1 1 0 0 0 1 1 0 0 1
1 1 1 1 1 5 1 1 0 1
1 0 0 0 0 0 0 1 0 1
1 1 5 5 5 5 5 5 1 1

Start: (0,0)
Goal: (9,9)
```

Expected Output

```
Path: [(0, 0), ..., (9, 9)]
Cost: [Insert cost]
Nodes Expanded: [Insert number]
Cycles Detected: Yes
```

Results

Running the sample test case with Dijkstra's Algorithm yields:



```
Windows PowerShell
PS D:\__repositories__\ai-pathfinding\src> python .\a_star.py ..\grids\input_grid_weighted_cycles.txt dijkstra
Size: 10 x 10
Grid:
1 2 3 4 5 1 1 1 1 5
1 0 0 0 5 1 1 0 0 1
1 0 3 0 5 0 0 0 1 5
1 0 1 0 5 5 5 0 1 1
5 5 5 1 1 1 1 1 1 1
1 1 1 1 0 0 5 0 1 1
1 1 0 0 0 1 1 0 0 1
1 1 1 1 1 5 1 1 0 1
1 0 0 0 0 0 0 1 0 1
1 1 5 5 5 5 5 5 1 1
Start: (0, 0)
Goal: (9, 9)


OUTPUT
Time: 0.004567099967971444 s
Path: [(0, 0), (1, 0), (2, 0), (3, 0), (4, 1), (5, 0), (6, 1), (7, 2), (7, 3), (7, 4), (6, 5), (7, 6), (8, 7), (
9, 8), (9, 9)]
Cost: 18
Nodes Expanded: 62
Cycles Detected: Yes
Diagram of solution:
. 2 3 4 5 1 1 1 1 5
. █ █ █ 5 1 1 █ █ 1
. █ 3 █ 5 █ █ █ 1 5
. █ 1 █ 5 5 5 █ 1 1
5 . 5 1 1 1 1 1 1 1
. 1 1 1 █ █ 5 █ 1 1
1 . █ █ █ . 1 █ █ 1
1 1 . . . 5 . 1 █ 1
1 █ █ █ █ █ █ . █ 1
1 1 5 5 5 5 5 5 . .
PS D:\__repositories__\ai-pathfinding\src>
```

Fig 4.4: Test with input_grid_weighted_cycles.txt

The runtime is relatively low even with the weights added.

Conclusion

This section just outlines some of the obstacles encountered as well as a summary of the results.

Summary

Making the program run from the command line required an import, and timing the performance also required an import. Overall, the A* algorithm had better performance when compared to the Dijkstra algorithm which makes sense since Dijkstra's algorithm effectively uses zero for the heuristic. All the nodes get expanded resulting in lower performance.

The result of the heuristic comparison was interesting as they performed about the same when only diagonal movements were allowed, but the Manhattan heuristic outperformed the Euclidean heuristic by an appreciable amount when only horizontal and vertical movements were allowed.

All the code used is included in the submission of the deliverables.

Obstacles

The base code provided had issues and was poorly commented resulting in a substantial amount of time being needed just to understand what most of the code was doing. Several modifications needed to be made for the code to run properly and more efficiently. This took several tries to get correct. The description of some of the tasks is ambiguous in some cases. Overall, the amount of time and effort required for this assignment was substantial.