Artificial Intelligence
Machine Problem 1 – A* for Sliding Puzzle

Introduction

For this assignment, you will implement the A* algorithm to solve the sliding tile puzzle game. Your goal is to return the instructions for solving the puzzle and show the configuration after each move.

Requirements

You are to create a program in Python 3 that performs the following:

1. Loads the `mp1input.txt` file from the current directory. This represents the starting state of the sliding puzzle. The format of this file is composed of 3 rows of 3 values, each value separated by a single space. The values are the integers 0 through 8 that represent the puzzle. The 0 integer represents an empty space (no tile). Here is an example of the input file contents:
```
3 1 2
4 7 5
0 6 8
```

2. Displays heading information to the screen:
```
Artificial Intelligence
MP1: A* for Sliding Puzzle
SEMESTER: [put semester and year here]
NAME: [your name here]
```

3. Executes the A* algorithm with the Manhattan distance heuristic (as discussed in the textbook). The goal state is this configuration:
```
0 1 2
3 4 5
6 7 8
```

4. Shows the solution in form of the puzzle configurations after each move, the move number, and the action taken. This format should match the sample output shown on the last page.

5. Displays the number of states that A* had to visit in order to get to the solution.

Grading Rubric

Category	Unsatisfactory (0-1 points)	Satisfactory (2-3 point)	Distinguished (4-5 points)
Program Correctness	• Program does not execute due to errors • Incorrect results for most or all input	• Program works and completes most tasks appropriately • Program fails to work for special cases	• Program runs and completes all required tasks • Handles any required special cases • Executes without errors
Programming Style	• No name, date, or assignment title included • Poor use of white space • Disorganized and messy • No or few comments in the source code • Poor use of variables (improper scope/visibility, ambiguous naming).	• Includes name, date, and assignment title. • White space makes program fairly easy to read. • Well organized code. • Some comments missing in the source code or too many comments • Good use of variables (few issues with scope/visibility or unambiguous naming).	• Includes name, date, and assignment title. • Excellent use of white space. • Perfectly organized code. • Source code is commented throughout when needed • Excellent use of variables (no issues with scope/visibility or unambiguous naming).
Following Specifications	• Incorrect filenames • Incorrect specified identifier names	• Correct filenames and class names • Few issues with other	• Correct filenames and specified identifier names • Source code organization

	• Source code organization different from requirements • Additional requirements not satisfied	specified identifier names • Source code organization close to requirements • Some additional requirements not satisfied	satisfies all requirements • All additional requirements satisfied

Additional Requirements

1. The name of your source code file should be `mp1.py`. All your code should be within a single file.
2. You can only import numpy and queue packages.
3. Your code should follow good coding practices, including good use of whitespace and use of both inline and block comments.
4. You need to use meaningful identifier names that conform to standard naming conventions.
5. At the top of each file, you need to put in a block comment with the following information: your name, date, course name, semester, and assignment name.
6. The output should **exactly** match the sample output shown on the last page. Note that for a different input state, the output may be different. I will be testing on a different input than shown in the sample.

What to Turn In

You will turn in the single mp1.py file using BlackBoard.

HINTS

- It's easiest to implement A* if you first define the class of PuzzleState objects. These should contain at least the following: the puzzle configuration (a numpy 2d array), the g/h/f costs, predecessor state reference, and the action that was taken to get to this state from the predecessor.
- Use Python's built-in PriorityQueue from the queue package to implement the frontier (open) set. Make sure to implement the __lt__ method in your PuzzleState class.
- Use Python's built-in set object to implement the closed set. You will need to make sure you implement the __hash__ method in your PuzzleState class to do this.

Sample Program Output 1

```
Artificial Intelligence
MP1: A* for Sliding Puzzle
SEMESTER: [put semester and year here]
NAME: [your name here]

START
[[3 1 2]
 [4 7 5]
 [0 6 8]]
Move 1 ACTION: right
[[3 1 2]
 [4 7 5]
 [6 0 8]]
Move 2 ACTION: up
[[3 1 2]
 [4 0 5]
 [6 7 8]]
Move 3 ACTION: left
[[3 1 2]
 [0 4 5]
 [6 7 8]]
Move 4 ACTION: up
[[0 1 2]
 [3 4 5]
 [6 7 8]]

Number of states visited = 5
```

Sample Program Output 2

```
Artificial Intelligence
MP1: A* for Sliding Puzzle
SEMESTER: [put semester and year here]
NAME: [your name here]

START
[[1 2 5]
 [6 3 8]
 [0 4 7]]
Move 1 ACTION: up
[[1 2 5]
 [0 3 8]
 [6 4 7]]
Move 2 ACTION: right
[[1 2 5]
 [3 0 8]
 [6 4 7]]
Move 3 ACTION: down
[[1 2 5]
 [3 4 8]
 [6 0 7]]
Move 4 ACTION: right
[[1 2 5]
 [3 4 8]
 [6 7 0]]
Move 5 ACTION: up
[[1 2 5]
 [3 4 0]
 [6 7 8]]
Move 6 ACTION: up
[[1 2 0]
 [3 4 5]
 [6 7 8]]
Move 7 ACTION: left
[[1 0 2]
 [3 4 5]
 [6 7 8]]
Move 8 ACTION: left
[[0 1 2]
 [3 4 5]
 [6 7 8]]

Number of states visited = 9
```