ASSIGNMENT #1 – COMP 3106 INTRODUCTION TO ARTIFICIAL INTELLIGENCE

The assignment is an opportunity to demonstrate your knowledge on informed graph search and practice applying it to a problem.

The assignment may be completed individually, or it may be completed in small groups of two or three students. The expectations will not depend on group size (i.e. same expectations for all group sizes).

Assignment due date: Friday, October 6, 2023

Assignments are to be submitted electronically through Brightspace. It is your responsibility to ensure that your assignment is submitted properly. Copying of assignments is NOT allowed. Discussion of assignment work with others is acceptable but each individual or small group are expected to do the work themselves.

Components

The assignment should contain two components: an implementation and a technical document.

Implementation

Programming language: Python 3

You may use the Python Standard Library (https://docs.python.org/3/library/). You may also use the NumPy, Pandas, and SciPy packages. Use of any additional packages requires approval of the instructor.

You must implement your code yourself. Do not copy-and-paste code from other sources, but you may use any pseudo-code we wrote in class as a basis for your implementation. Your implementation must follow the outlined specifications. Implementations which do not follow the specifications may receive a grade of zero. Please make sure your code is readable, as it will also be assessed for correctness. You do not need to prove correctness of your implementation.

You may be provided with a set of examples to test your implementation. Note that the provided examples do not necessarily represent a complete set of test cases. Your implementation may be evaluated on a different set of test cases.

The implementation will be graded both on content and use of good programming practices.

Submit the implementation as a single PY file.

Technical Document

Your technical document should answer all questions posed below. Ensure you answers are clear and concise.

If the assignment was completed in a small group of students, the technical document must include a statement of contributions. This statement should identify: (1) whether each group member made significant contribution, (2) whether each group member made an approximately equal contribution, and (3) exactly which aspects of the assignment each group member contributed to.

Submit the technical document as a single PDF file.

Implementation

Consider the problem of finding a path from a start position to an end position in a grid-like environment. Implement an algorithm using A* search to find the optimal path to goal square.

In this problem, assume we have an agent that starts at a start state on an m x n rectangular grid. The agent tries to find a path from the start state to the goal state by moving to adjacent horizontal or vertical squares on the grid (cannot move diagonally). The cost of moving to an adjacent square is 1. There may also be obstacles on the grid; the agent can traverse at most one obstacle along its path from the start state to the goal state. The cost of moving to a square with an obstacle is the same as the cost of moving to a regular square. Assume the environment is fully observable; assume there always exists a path from the start state to the goal state.

Suppose we have an m x n grid where each square in the grid can be one of the following:
The start square (indicated by the letter "S")
The goal square (indicated by the letter "G")
An obstacle (indicated by the letter "X")
A regular square (indicated by the letter "O")

Your implementation must contain a single file named "assignment1.py" with a function named "pathfinding", which takes one input argument (you may have other variables/functions/classes in your file). The input argument should be the full file path to a comma separated value (CSV) file that contains the input grid.

The CSV file will contain a grid describing the environment. The CSV will contain a rectangular grid with one S (indicating the start square), one G (indicating the goal square), zero or more X (indicating obstacles), and zero or more O (indicating regular squares).

Your function should return three values.

The first returned value should be a list of tuples indicating the coordinates of the squares occupied along the optimal path, in order. Assume the first index indicates the row and the second index indicates the column. Both row and column indices are integers and start at zero.

The second returned value should be the cost of the optimal path.

The third returned value should be the number of states explored during A* search.

Attached are example CSV files and corresponding example text files containing the optimal path, cost of the optimal path, and number of explored nodes. Note that your function should not write anything to file. These examples are provided in separate files for convenience. Also attached is skeleton code indicating the format your implementation should take.

Note that for some examples there may be multiple correct solutions. The number of explored nodes may also be different depending on the heuristic you choose. All correct solutions will be accepted.

Example CSV file:
S,O,X
X,X,O
X,X,O
G,O,O

Example first output (i.e. optimal path):
[(0, 0), (0, 1), (0, 2), (1, 2), (2, 2), (3, 2), (3, 1), (3, 0)]

Example second output (i.e. cost of optimal path):
7

Example third output (i.e. number of states explored):
12

Grading

The implementation will be worth 60 marks.

40 marks will be allocated to correctness on a series of test cases, with consideration to each of the three outputs (i.e. optimal path, optimal path cost, and number of states explored). These test cases will be run automatically by calling your implementation from another Python script. To facilitate this, your implementation must adhere exactly to the specifications.

20 marks will be allocated to human-based review of code. This human-based review will consider both correctness and use of good programming practices.

Technical Document

Please answer the following questions in the technical document. For all questions, explain why your answers are correct.

1. Briefly describe how your implementation works. Include information on any important algorithms, design decisions, data structures, etc. used in your implementation. [10 marks]

The implementation uses A* search with a heuristic (Manhattan distance) that estimates the minimum number of steps needed. The Manhattan distance heuristic, also known as the L1 norm or taxicab distance, measures the distance between two points in a grid based on the sum of the absolute differences of their coordinates. In a 2D grid like the one being used here; it is calculated as:

→ h = |goal_row - current_row| + |goal_column - current_column|

where **goal_row** and **goal_column** are the row and column coordinates of the goal square, and **current_row** and **current_column** are the row and column coordinates of the current square.

Classes and Data Structures:

Square class represents a position in the grid by its row and column indices.

State class represents a state in the search space. It includes the current square, the number of obstacles traversed, and information for A* search (e.g., g, h values).

Initialization:

It starts by reading a CSV file containing the grid. It identifies the start state (S) and goal state (G) and creates initial **State** objects for them.

A Search*:

The algorithm employs a priority queue (**frontier**) to keep track of states to explore. States with lower **f** values (sum of **g** and **h**) are given higher priority. The algorithm proceeds until either a solution is found or the **frontier** is empty.

Exploration:

The algorithm explores adjacent squares (up, down, left, right) from the current state. It generates potential next states and evaluates their cost and heuristic values. It updates the **frontier** with new states if they have not been explored or have a lower cost.

Obstacle Handling:

It keeps track of the number of obstacles traversed by the agent and ensures it doesn't exceed a specified limit (**MAX_OBSTACLES_TRAVERSABLE**).

Algorithm to handle the encountered obstacles:

1. The loop iterates through all adjacent squares to the current state.
2. For each adjacent square, it checks if it's an obstacle (**curr_obstacle** is **True** if it is).
3. It creates a new **State** object (**curr_node**) with updated obstacle count based on whether an obstacle was encountered.
4. If the obstacle count exceeds the maximum allowable (**MAX_OBSTACLES_TRAVERSABLE**), it skips this option.
5. It calculates the path cost to the new state.
6. It then checks if the new state has already been explored or is in the frontier.

- If it's in the explored set, it continues to the next option.
- If it's in the frontier, it compares the path cost and updates the node if the new path cost is lower.

***Termination and Path Reconstruction**:*

If a goal state is reached, the algorithm reconstructs the optimal path by following the parent pointers from the goal state back to the start state.

Output:
The function returns three values: the optimal path as a list of tuples, the cost of the optimal path, and the number of states explored.

Other Considerations:
The implementation uses heapq for efficient priority queue operations.
It applies Manhattan distance as the heuristic function (**h**) for A* search.

2. What type of agent have you implemented (simple reflex agent, model-based reflex agent, goal-based agent, or utility-based agent)? [3 marks]

This is a utility-based agent. It needs to find the lowest total cost path to reach a goal square. The utility function it optimizes is the cost of the path to a goal square.

3. Is the task environment: [7 marks]
    a. Fully or partially observable?
    b. Single or multiple agent?
    c. Deterministic or stochastic?
    d. Episodic or sequential?
    e. Static or dynamic?
    f. Discrete or continuous?
    g. Known or unknown?

a. The environment is fully observable because the agent can see every square in the grid (even though it can only move to adjacent squares).
b. This a single agent environment because there no other entity that change their behavior based on the agent's actions.
c. The environment is deterministic. The outcome of actions is completely determined by the state.
d. This is a sequential environment; prior actions affect subsequent states.
e. The environment is static. There is no mention of any elements in the environment (e.g., obstacles) changing over time.
f. The environment is discrete. The agent moves from one grid cell to another, which are discrete states.
g. The environment is known. The agent is provided with the complete grid layout, and it knows the location of the start state, goal state, obstacles, and regular squares.

4. What heuristic did you use for A* search for this environment? Show that your heuristic is consistent. [8 marks]

The heuristic used for A* search in this environment is the Manhattan distance.
The Manhattan distance is calculated as the sum of the absolute differences between the current state's row and column indices and the goal state's row and column indices:
→ h = |goal_row - current_row| + |goal_column - current_column|

Clearly, h(goal) = 0

Consider a state s, a successor s', and a goal state g with positions A, B, and C, respectively.
According to the Triangle Inequality, for any three points A, B, and C, the sum of the distances AB <= cost(s, s') and BC = h(s') is greater than or equal to the distance AC = h(s). That is, cost(s, s') + h(s') >= AB + BC >= AC = h(s).

Thus, the heuristic is consistent, A* search will always return the optimal path to a goal state when it is used.

5. Suggest a particular instance of this problem (i.e. grid) where A* search using your heuristic would find the optimal solution faster than uniform cost search. [4 marks]

To find a scenario where A* search with the Manhattan distance heuristic would find the optimal solution faster than uniform cost search, we should look for a grid that has a clear and direct path to the goal but has some obstacles that can be bypassed using the heuristic.

Consider the following grid:
S,O,O,O
X,X,O,X
O,O,O,G

In this example, the optimal path is quite direct and doesn't require moving around many obstacles. However, if the agent were to use uniform cost search, it would still explore all available squares, including the obstacles, to ensure it finds the optimal path.

Number of states explored during A* search: 14

[<Obstacles traversed: 0, Row: 0, Column: 0, g: 0, h: 5, f: 5>, <Obstacles traversed: 1, Row: 1, Column: 0, g: 1, h: 4, f: 5>, <Obstacles traversed: 0, Row: 0, Column: 1, g: 1, h: 4, f: 5>, <Obstacles traversed: 1, Row: 2, Column: 0, g: 2, h: 3, f: 5>, <Obstacles traversed: 1, Row: 1, Column: 1, g: 2, h: 3, f: 5>, <Obstacles traversed: 0, Row: 0, Column: 2, g: 2, h: 3, f: 5>, <Obstacles traversed: 1, Row: 1, Column: 2, g: 3, h: 2, f: 5>, <Obstacles traversed: 0, Row: 1, Column: 2, g: 3, h: 2, f: 5>, <Obstacles traversed: 0, Row: 0, Column: 3, g: 3, h: 2, f: 5>, <Obstacles traversed: 1, Row: 2, Column: 1, g: 3, h: 2, f: 5>, <Obstacles traversed: 1, Row: 1, Column: 3, g: 4, h: 1, f: 5>, <Obstacles traversed: 0, Row: 2, Column: 2, g: 4, h: 1, f: 5>, <Obstacles traversed: 1, Row: 2, Column: 2, g: 4, h: 1, f: 5>, <Obstacles traversed: 0, Row: 2, Column: 3, g: 5, h: 0, f: 5>]

Number of states explored during Uniform Cost Search: 17

[<Obstacles traversed: 0, Row: 0, Column: 0, g: 0, h: 0, f: 0>, <Obstacles traversed: 1, Row: 1, Column: 0, g: 1, h: 0, f: 1>, <Obstacles traversed: 0, Row: 0, Column: 1, g: 1, h: 0, f: 1>, <Obstacles traversed: 1, Row: 0, Column: 0, g: 2, h: 0, f: 2>, <Obstacles traversed: 1, Row: 1, Column: 1, g: 2, h: 0, f: 2>, <Obstacles traversed: 0, Row: 0, Column: 2, g: 2, h: 0, f: 2>, <Obstacles traversed: 1, Row: 2, Column: 0, g: 2, h: 0, f: 2>, <Obstacles traversed: 0, Row: 1, Column: 2, g: 3, h: 0, f: 3>, <Obstacles traversed: 0, Row: 0, Column: 3, g: 3, h: 0, f: 3>, <Obstacles traversed: 1, Row: 2, Column: 1, g: 3, h: 0, f: 3>, <Obstacles traversed: 1, Row: 1, Column: 2, g: 3, h: 0, f: 3>, <Obstacles traversed: 1, Row: 0, Column: 1, g: 3, h: 0, f: 3>, <Obstacles traversed: 0, Row: 2, Column: 2, g: 4, h: 0, f: 4>, <Obstacles traversed: 1, Row: 2, Column: 2, g: 4, h: 0, f: 4>, <Obstacles traversed: 1, Row: 0, Column: 2, g: 4, h: 0, f: 4>, <Obstacles traversed: 1, Row: 1, Column: 3, g: 4, h: 0, f: 4>, <Obstacles traversed: 0, Row: 2, Column: 3, g: 5, h: 0, f: 5>]

For these states, “Obstacles traversed” indicates how many obstacles have already been traversed in each state. “Row” and “Column” indicate the position of the agent, “g” indicates path cost, “h” indicates heuristic value, “f” indicates estimated total path cost.

6. Suggest a particular instance of this problem (i.e. grid) where a greedy heuristic search using your heuristic would not find the optimal solution. [4 marks]

Consider the grid provided in Example2.

O,O,S,O,O
O,X,X,X,O
O,O,O,O,O
O,O,O,O,O
O,X,X,X,O

O,X,G,X,O
O,X,O,X,O
O,O,O,O,O

A* search finds the path: [(0, 2), (0, 3), (0, 4), (1, 4), (2, 4), (2, 3), (2, 2), (3, 2), (4, 2), (5, 2)]

This has path cost: 9

Greedy heuristic search finds the path: [(0, 2), (1, 2), (2, 2), (3, 2), (3, 1), (3, 0), (4, 0), (5, 0), (6, 0), (7, 0), (7, 1), (7, 2), (6, 2), (5, 2)]

The path has cost: 13

7. Consider a modification of this problem where the agent must traverse exactly one obstacle (for this question, assume all grids have at least one obstacle). Determine whether your heuristic is still consistent. [4 marks]

The same reasoning as before applies in this case – the heuristic is still consistent.

The heuristic used for A* search in this environment is the Manhattan distance.
The Manhattan distance is calculated as the sum of the absolute differences between the current state's row and column indices and the goal state's row and column indices:
→ h = |goal_row - current_row| + |goal_column - current_column|

Clearly, h(goal) = 0

Consider a state s, a successor s', and a goal state g with positions A, B, and C, respectively.
According to the Triangle Inequality, for any three points A, B, and C, the sum of the distances AB <= cost(s, s') and BC = h(s') is greater than or equal to the distance AC = h(s). That is, cost(s, s') + h(s') >= AB + BC >= AC = h(s).

Thus, the heuristic is consistent, A* search will always return the optimal path to a goal state when it is used.

Note that we could perhaps find a more informed heuristic that would find the solution faster.

Grading

The technical document will be worth 40 marks, allocated as described above.