


Assignment 1. Search


Gary Geunbae Lee
CSED342-01 Artificial Intelligence

Contact: TA Seonjeong Hwang (seonjeongh@postech.ac.kr)

General Instructions

This assignment has a written part and a programming part.

 This icon means a written answer is expected. Some of these problems are multiple choice questions that impose negative scores if the answers are incorrect. So, don't write answers unless you are confident. You should submit your answers to the written part in a pdf titled `name_studentID.pdf`.

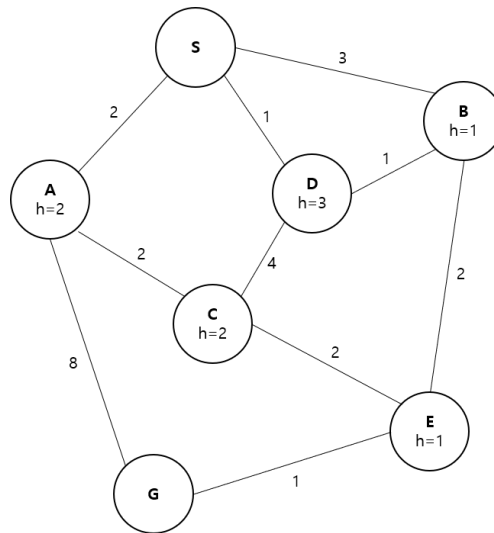
 This icon means you should write code. you can add other helper functions outside the answer block if you want. You should write both types of answers in `puzzle.py` between

```
# BEGIN_YOUR_ANSWER
```

```
and
```

```
# END_YOUR_ANSWER
```

1. Search Algorithms [25pt]

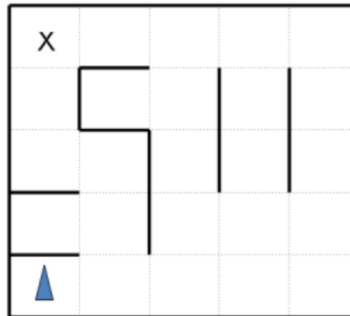


For each of the following graph search strategies, work out the **(1) order** in which states are expanded, as well as the **(2) path** returned by graph search. In all cases, assume ties resolve in such a way that states with earlier alphabetical order are expanded first. The start and goal state are S and G, respectively. Remember that in graph search, a state is expanded only once.

- (a) Depth-first search. [5pt]
- (b) Breadth-first search. [5pt]
- (c) Uniform cost search. [5pt]
- (d) Greedy search with the heuristic h . [5pt]
- (e) A* search with the same heuristic. [5pt]

2. Search and Heuristics [35pt]

Imagine a car-like agent wishes to exit a maze like the one shown below:



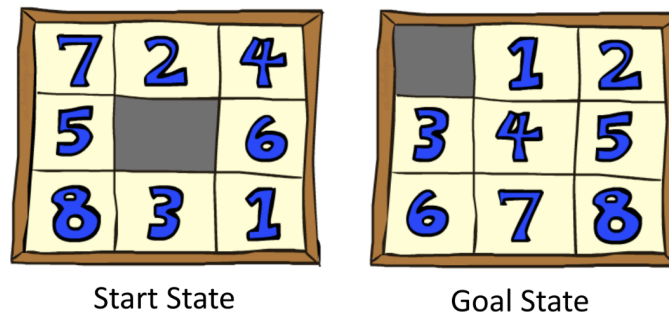
The agent is directional and at all times faces some direction $d \in (N, S, E, W)$. With a single action, the agent can *either* move forward at an adjustable velocity v or turn. The turning actions are *left* and *right*, which change the agent's direction by 90 degrees. Turning is only permitted when the velocity is zero (and leaves it at zero). The moving actions are *fast* and *slow*. *Fast* increments the velocity by 1 and *slow* decrements the velocity by 1; in both cases the agent then moves a number of squares equal to its NEW adjusted velocity. Any action that would result in a collision with a wall crashes the agent and is illegal. Any action that would reduce v below 0 or above a maximum speed V_{max} is also illegal. The agent's goal is to find a plan which parks it (stationary) on the exit square using as few actions (time steps) as possible.

As an example: if the agent in the figure above was initially stationary, it might first turn to the east using (*right*), then move one square east using *fast* (its current velocity is 1), then two more squares east using *fast* again (the current velocity is 2). To change direction, the agent must select the *slow* action twice to reduce its velocity to zero.

- If the grid is M by N , what is the size of the state space? We assume that all configurations are reachable from the start state. [5pt]
- What is the maximum and minimum branching factor of this problem? We assume that illegal actions are not returned by the successor function. Describe the states corresponding to the two cases. [10pt]
- Is the Manhattan distance from the agent's location to the exit's location admissible? Why or why not? [10pt]
- State and justify a non-trivial admissible heuristic for this problem which is not the Manhattan distance to the exit. [10pt]

3. 8-Puzzle [40pt]

The 8-puzzle is a popular sliding puzzle that consists of a 3x3 grid (9 squares) with 8 numbered tiles and one empty space. The goal of the puzzle is to rearrange the tiles from a given initial configuration to a goal configuration by sliding them one at a time into the empty space. In this assignment, you will write code to solve this 8-puzzle using the A* algorithm.



The given `puzzle.py` consists of the following classes:

- **Puzzle** represents a $N \times N$ puzzle containing $N^2 - 1$ tiles and one empty space.
- **Node** represents a node in a search tree.
- **Searcher** is used to find a path from a given puzzle state to a goal state using the A* algorithm.

and each class consists of complete or incomplete methods.

To run the code with your own sample, modify the `initial_state` variable in the `main` function in `puzzle.py` and type

```
python puzzle.py
```

Your code will be evaluated on five test cases, which you can see in `grader.py`. To run all the tests, type

```
python grader.py all
```

You can also run a single test (1 - 5) by typing

```
python grader.py 5
```

You cannot use additional Python modules (e.g. `collections`) to implement your code, but you are free to use Python built-in functions (e.g. `abs()`, `sorted()`).

Refer to <https://docs.python.org/3/library/functions.html>.

- (a) Complete `Puzzle.heuristic_cost`. We will use the *Manhattan distance* as a heuristic function. [10pt]
- (b) Complete `Puzzle.actions`. [10pt]
- (c) Complete `Searcher.search`. When the search is finished, print the search path through `show_path` of the goal node. [20pt]