Artificial Intelligence Spring 2018
Homework 1: Search Algorithms
Due Date: Thursday, February 15 at 11:59pm

ACADEMIC HONESTY

As usual, the standard honor code and academic honesty policy applies. We run **automated checks for plagiarism** to ensure only original work is given credit. Submissions isomorphic to (1) those that exist anywhere online, (2) those submitted by classmates, or (3) those submitted by students in prior semesters, will be considered plagiarism.

SUBMISSION

You will submit one zip file, named **hw1_myUNI.zip**, which contains two files:

- written.pdf, and
- driver.py or driver_3.py

You can submit as many times as you like before the deadline. Only your most recent submission will be graded. The written submission can be handwritten and scanned (PDF scanning mobile apps typically work well), or typed, but it must be legible in order to receive credit.

WRITTEN

Question 1: Virtual Assistant as an Intelligent Agent

From the Google Home website: “Google Home is a powerful speaker and voice Assistant. Play your music. Call your friends. Ask it questions.”
The Google Home is a microphone and speaker duo with Google Assistant listening for user directives, which at a basic level span search queries, updating personal calendars, and making phone calls via wifi. Imagine you have a Google Home in your living room which is set up to recognize your voice. For simplicity, your Home has no chromecasts or other linked devices. When the home recognizes your voice, you can ask for information regarding your calendar schedule, make phone calls, and update personal data. Anyone can ask general questions, and directives can go more than one statement. For example, “OK Google, how do I cook an omelette?” prompts the Assistant to recite a list of ingredients and first step of a recipe, and then waits for you to say “Next step,” or give a different directive.

1. Do a *PEAS* analysis of the Google Home.
2. Describe the Google Home environment, using the classifications from lecture (classify based on observability, number of agents, deterministic/stochastic, and discrete/continuous).

Please justify your answers.

Question 2

What’s the size of the state space in the N-puzzle game? Please justify. Remember the N-puzzle game contains N = m^2 - 1 tiles and one empty space.

Question 3

Let h1 and h2 be two admissible heuristics. Which of the following heuristics are admissible?
Justify each answer.

(a) $h(n) = min\{5 * h1(n), h2(n)\}$

(b) $h(n) = max\{h1(n), h2(n)\}$

(c) $h(n) = w * h1(n) + (1 - w) * h2(n)$, with $0 \leq w \leq 1$

(d) $h(n) = \frac{max\{2 * h1(n), h2(n)\}}{2}$

PROGRAMMING

In this assignment you will create an agent to solve the n-puzzle game (i.e. 8-puzzle game generalized to an n × n array). Visit mypuzzle.org/sliding for a refresher of the game's rules. You will implement and compare several search algorithms, and collect some statistics related to their performances.
Please read all sections carefully:

I. Introduction

The N-puzzle game consists of a board holding N = m^2 − 1 distinct movable tiles, plus one empty space. There is one tile for each number in the set {1, …, m^2 − 1}. In this assignment, we will represent the blank space with the number 0 and focus on the m = 3 case (8-puzzle).

In this combinatorial search problem, the aim is to get from any initial board state to the configuration with all tiles arranged in ascending order ⟨0, 1,…, m^2 − 1⟩ -- this is your goal state. The search space is the set of all possible states reachable from the initial state. Each move consists of swapping the empty space with a component in one of the four directions {'Up', 'Down', 'Left', 'Right'}. Give each move a cost of one. Thus, the total cost of a path will be equal to the number of moves made.

II. Algorithm Review

Recall from lecture that search begins by visiting the root node of the search tree, given by the initial state. Three main events occur when visiting a node:

- First, we remove a node from the frontier set.
- Second, we check if this node matches the goal state.
- If not, we then expand the node. To expand a node, we generate all of its immediate successors and add them to the frontier, if they (i) are not yet already in the frontier, and (ii) have not been visited yet.

This describes the life cycle of a visit, and is the basic order of operations for search agents in this assignment—(1) remove, (2) check, and (3) expand. We will implement the assignment algorithms as described here. Please refer to lecture notes for further details, and review the lecture pseudo-code before you begin.

IMPORTANT: You may encounter implementations that attempt to short-circuit this order by performing the goal-check on successor nodes immediately upon expansion of a parent node. For example, Russell & Norvig's implementation of BFS does precisely this. Doing so may lead to edge-case gains in efficiency, but do not alter the general characteristics of complexity and optimality for each method. For simplicity and grading purposes in this assignment, do not make such modifications to algorithms learned in lecture.

III. What You Need To Submit

Your job in this assignment is to write driver.py, which solves any 8-puzzle board when given an arbitrary starting configuration. The program will be executed as follows:

$ python driver.py <method> <board>

The method argument will be one of the following. You must implement all three of them:
bfs (Breadth-First Search)
dfs (Depth-First Search)
ast (A-Star Search)

The board argument will be a comma-separated list of integers containing no spaces. For example, to use the bread-first search strategy to solve the input board given by the starting configuration {0,8,7,6,5,4,3,2,1}, the program will be executed like so (with no spaces between commas):

$ python driver.py bfs 0,8,7,6,5,4,3,2,1

IMPORTANT: If you are using Python 3, please name your file driver_3.py, so we use the correct version while grading. If you name your file driver.py, the default version for our box is Python 2.

IV. What Your Program Outputs

Your program will create and/or write to a file called output.txt, containing the following statistics:

path_to_goal: the sequence of moves taken to reach the goal
cost_of_path: the number of moves taken to reach the goal
nodes_expanded: the number of nodes that have been expanded
search_depth: the depth within the search tree when the goal node is found
max_search_depth: the maximum depth of the search tree in the lifetime of the algorithm
running_time: the total running time of the search instance, reported in seconds
max_ram_usage: the maximum RAM usage in the lifetime of the process as measured by the **ru_maxrss** attribute in the **resource** module, reported in megabytes

Example #1: Breadth-First Search
Suppose the program is executed for breadth-first search as follows:

$ python driver.py bfs 1,2,5,3,4,0,6,7,8

This should result in the solution path:

$$parent = \begin{array}{|c|c|c|}\hline 1 & 2 & 5 \\ \hline 3 & 4 & \\ \hline 6 & 7 & 8 \\ \hline \end{array} \implies child = \begin{array}{|c|c|c|}\hline 1 & 2 & \\ \hline 3 & 4 & 5 \\ \hline 6 & 7 & 8 \\ \hline \end{array}$$

$$parent = \begin{array}{|c|c|c|}\hline 1 & 2 & \\ \hline 3 & 4 & 5 \\ \hline 6 & 7 & 8 \\ \hline \end{array} \implies child = \begin{array}{|c|c|c|}\hline 1 & & 2 \\ \hline 3 & 4 & 5 \\ \hline 6 & 7 & 8 \\ \hline \end{array}$$

$$parent = \begin{array}{|c|c|c|}\hline 1 & & 2 \\ \hline 3 & 4 & 5 \\ \hline 6 & 7 & 8 \\ \hline \end{array} \implies child = \begin{array}{|c|c|c|}\hline & 1 & 2 \\ \hline 3 & 4 & 5 \\ \hline 6 & 7 & 8 \\ \hline \end{array}$$

The output file will contain **exactly** the following lines:

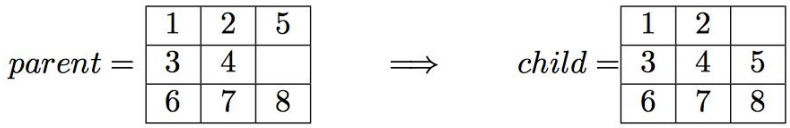
path_to_goal: ['Up', 'Left', 'Left']
cost_of_path: 3
nodes_expanded: 10
search_depth: 3
max_search_depth: 4
running_time: 0.00188088
max_ram_usage: 0.07812500

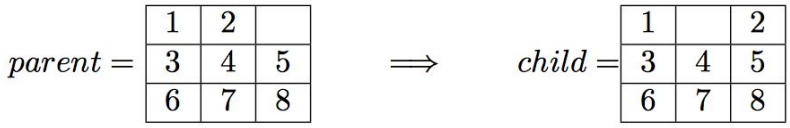
Example #2: Depth-First Search

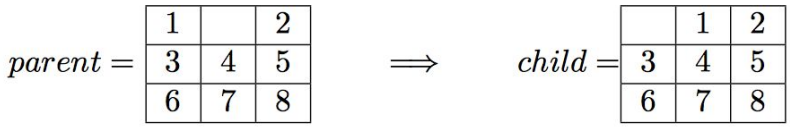
Suppose the program is executed for depth-first search as follows:

$ python driver.py dfs 1,2,5,3,4,0,6,7,8

This should result in the solution path:







The output file will contain **exactly** the following lines:

path_to_goal: ['Up', 'Left', 'Left']
cost_of_path: 3
nodes_expanded: 181437
search_depth: 3
max_search_depth: 66125
running_time: 5.01608433
max_ram_usage: 4.23940217

More test cases are provided in **Implementation FAQs**.

Note on Correctness

All variables, except running_time and max_ram_usage, have **one and only one** correct answer when running BFS and DFS. A* nodes_expanded might vary slightly depending on implementation details. You'll be fine as long as your algorithm follows all specifications listed in these instructions.
As running_time and max_ram_usage values vary greatly depending on your machine and implementation details, there is no "correct" value to look for. They are for you to monitor time and space complexity of your code, which we highly recommend. A good way to check the correctness of your program is to walk through small examples by hand, like the ones above.

V. Implementation and Testing

Since this is the first programming project, we are providing hints and explicit instructions. Before posting a question on the discussion board, make sure your question is not already answered here.

1. Implementation

You will implement the following three algorithms as demonstrated in lecture. In particular:

- **Breadth-First Search**. Use an explicit queue, as shown in lecture.
- **Depth-First Search**. Use an explicit stack, as shown in lecture.
- **A-Star Search**. Use a priority queue, as shown in lecture. For the choice of heuristic, use the Manhattan priority function; that is, the sum of the distances of the tiles from their goal positions. Note that the blanks space is not considered an actual tile here.

2. Order of Visits

In this assignment, where an arbitrary choice must be made, we always **visit** child nodes in the "**UDLR**" order; that is, ['Up', 'Down', 'Left', 'Right'] in that exact order. Specifically:

- **Breadth-First Search**. Enqueue in UDLR order; de-queuing results in UDLR order.
- **Depth-First Search**. Push onto the stack in reverse-UDLR order; popping off results in UDLR order.
- **A-Star Search**. Since you are using a priority queue, what happens with duplicate keys? How do you ensure nodes are retrieved from the priority queue in the desired order?

3. Submission Test Cases

Run all three of your algorithms on the following test cases:

Test Case #1

```
python driver.py bfs 3,1,2,0,4,5,6,7,8
python driver.py dfs 3,1,2,0,4,5,6,7,8
python driver.py ast 3,1,2,0,4,5,6,7,8
```

Test Case #2

```
python driver.py bfs 1,2,5,3,4,0,6,7,8
python driver.py dfs 1,2,5,3,4,0,6,7,8
python driver.py ast 1,2,5,3,4,0,6,7,8
```

Make sure your code passes at least these test cases and follows our formatting exactly. The results of each test are assessed by 8 items: 7 are listed in **Section IV. What Your Program Outputs**. The last point is for code that executes and produces any output at all. Each item is worth 0.75 point.

4. Grading and Stress Tests

We will grade your project by running **additional test cases** on your code. In particular, there will be five test cases in total, each tested on all three of your algorithms, for a total of **15** distinct tests. Similar to the submission test cases, each test will be graded by 8 items, for a total of 90 points. **Plus, we give 10 points for code completing all 15 test cases within 10 minutes**. If you implement your code with reasonable designs of data structures, your code will solve all 15 test cases within a minute in total. We will be using a wide variety of inputs to stress-test your algorithms to check for correctness of implementation. So, we recommend that you test your own code extensively.

Don't worry about checking for malformed input boards, including boards of non-square dimensions, or unsolvable boards.

You will not be graded on the absolute values of your running time or RAM usage statistics. The values of these statistics can vary widely depending on the machine. **However, we recommend that you take advantage of them in testing your code.** Try batch-running your algorithms on various inputs, and plotting your results on a graph to learn more about the space and time complexity characteristics of your code. Just because an algorithm provides the correct path to goal does not mean it has been implemented correctly.

5. Tips on Getting Started

Begin by writing a class to represent the **state** of the game at a given turn, including parent and child nodes. We suggest writing a separate **solver** class to work with the state class. Feel free to experiment with your design, for example including a **board** class to represent the low-level physical configuration of the tiles, delegating the high-level functionality to the state class. When comparing your code with pseudocode, you might come up with another class for organizing specific aspects of your search algorithm elegantly.

You will not be graded on your design, so you are at a liberty to choose among your favorite programming paradigms. Students have successfully completed this project using an entirely object-oriented approach, and others have done so with a purely functional approach. Your submission will receive full credit as long as your driver program outputs the correct information.

VI. Before You Finish

- **Make sure** your code passes at least the submission test cases.
- **Make sure** your algorithms generate the correct solution for an arbitrary solvable problem instance of 8-puzzle.
- **Make sure** your program always terminates without error, and in a reasonable amount of time. **You will receive zero points from the grader if your program fails to terminate. Running times of more than a minute or two may indicate a problem with your implementation.** If your implementation exceeds the time limit allocated (20 minutes for all test cases), your grade may be incomplete.
- **Make sure** your program output follows the specified format exactly. In particular, for the path to goal, use square brackets to surround the list of items, use single quotes around each item, and capitalize the first letter of each item. Round floating-point numbers to 8 places after the decimal. You will not receive proper credit from the grader if your format differs from the provided examples above.