Course: **Artificial Intelligence** **(Fall 2022)**

Resource Person: **Hafiz Ahmad Farooq** **ASSIGNMENT–2**

Total Points: 40
Submission Due: Tuesday, April 12, 2022

Instructions: Please Read Carefully!

- **This is an individual assignment. Everyone is expected to complete the given assignment on their own, without seeking any help from any website or any other individual.** There will be strict penalties for any work found copied from any source and the university policy on plagiarism will be strictly enforced.
- You are expected to submit this assignment as:
    a. Create a single code file for the assignment solution.
    b. Make sure to run the test program using all algorithms and test data.
- Assignment is to be submitted in reply to the assignment posted on the **Google Classroom**.
- Late submissions will have a **10% negative** penalty on each day.

INSTRUCTIONS

In this assignment you will create an agent to solve the **8-puzzle** game. You may visit **mypuzzle.org/sliding** for a refresher of the rules of the game. You will implement and compare several search algorithms and collect some statistics related to their performances. Please read all sections of the instructions carefully:

CHOICE OF PROGRAMMING LANGUAGE

You may use only **Python** programming language to write the solution of this programming assignment. No other programming language should be used.

ASSIGNMENT CONTENTS

NOTE: This assignment incorporates material learned from both Uninformed search and Informed search (Chapter-3).

I. Introduction

An instance of the N-puzzle game consists of a **board** holding N = m^2 − 1 (m = 3, 4, 5, ...) distinct movable tiles, plus an empty space. The tiles are numbers from the set {1, …, m^2 − 1}. For any such board, the empty space may be legally swapped with any tile horizontally or vertically adjacent to it. In this assignment, we will represent the **blank space** with the number **0** and focus **only** on the **m = 3** case (**8-puzzle**).

Given an initial **state** of the board, the combinatorial search problem is to find a sequence of moves that transitions this state to the goal state; that is, the configuration with all tiles arranged in ascending order ⟨0, 1, …, m^2 − 1⟩. The search space is the set of all possible states reachable from the initial state.

The blank space may be swapped with a component in one of the four directions **{'Up', 'Down', 'Left', 'Right'}**, one move at a time. The cost of moving from one configuration of the board to another is the **same** and equal to **one**. Thus, the **total cost of path** is equal to the **number of moves made from the initial state to the goal state**.

II. Algorithm Review

Recall from the lectures that search begin by visiting the root node of the search tree, given by the initial state. Among other book-keeping details, three major things happen in sequence to visit a node:

- First, we **remove** a node from the frontier (fringe) set.
- Second, we **check** the state against the goal state to determine if a solution has been found.
- Finally, if the result of the check is negative, we then expand the node. To expand a given node, we generate successor nodes adjacent to the current node, and add them to the frontier (fringe) set. Note that

if these successor nodes are already in the frontier, or have already been visited, then they should not be added to the frontier again.

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**. In this assignment, we will implement algorithms as described here.

III. What You Need to Submit

Your job in this assignment is to write:

- **`solve8Puzzle.py`**

The method argument will be one of the following. You need to implement all **three** of them:

- **`bfs` (Breadth-First Search)**
- **`dfs` (Depth-First 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 solve8Puzzle.py bfs 0,8,7,6,5,4,3,2,1`**

Remember: the number **0** represents the space in the 8-puzzle.

IV. What Your Program Outputs

When executed, your program will create / write to a file called **`searchResults.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

Example #1: Breadth-First Search

Suppose the program is executed for breadth-first search as follows or it can be asked for initial board configuration:

- **`python solve8Puzzle.py bfs 1,2,5,3,4,0,6,7,8`**
- Which should lead to the following solution to the input board:

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

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

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

The output file contains **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
```

Example #2: Depth-First Search

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

- **`python solve8Puzzle.py dfs 1,2,5,3,4,0,6,7,8`**

Which should lead to the following solution to the input board:

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

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

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

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

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

Note on Correctness

Of course, the specific values for running_time variable will vary greatly depending on the machine used and the specific implementation details; there is no "correct" value to look for. This is intended to enable you to check the time complexity characteristics of your code, and you should spend time to do so. All the other variables, however, will have **one and only one** correct answer for each algorithm. A good way to check the correctness of your program is to walk through small examples by hand, like the ones above.

In general, **for any initial board whatsoever**, for BFS and DFS there is one and only one correct answer. You will be fine as long as your algorithm conforms to all **specifications** listed in these instructions.

V. Important Information

Please read the following information carefully. Since this is the first programming assignment, you are being provided with many hints and explicit instructions. Before you post a clarifying question on the discussion board, make sure that your question is not already answered in the following sections.

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.

    - **NOTE**: You may use the built-in priority queue data structure or **`simpleai`** library.

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; dequeuing results in UDLR order.
- **Depth-First Search**. Push onto the stack in reverse-UDLR order; popping off results in UDLR order.

3. 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. Your submission will receive full credit if your driver program outputs the correct information.

VI. Before You Finish

- **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 if your program fails to terminate. Running times of more than a minute or two may indicate a problem with your implementation.**
- **Make sure** your program output follows the specified format exactly. 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 if your format differs from the provided examples above.

END OF ASSIGNMENT