COMP 472	**Assignment 2**	**Winter 2021**

Due dates:	**March 28, 2021**
Late submission:	20% per day
Teams:	You can do the assignment individually or in teams of 4 students.
Purpose:	The purpose of this assignment is to implement and analyze a search for a game.

1. The S-Puzzle

In this assignment you will implement and analyze a variety of search algorithms to solve S-Puzzle.

1.1 Rules of the Puzzle

The S-Puzzle for this assignment is a type of sliding-Tiles puzzle played on a n-by-n grid, with numbers occupying the cells. In the Tile game, S-Puzzle There is no space; all n^2 tiles are occupied. The game is played by swapping any two adjacent tiles, where adjacency is defined grid-wise (diagonal tiles are not adjacent) and does not wrap around. Like in the original Sliding Tiles puzzle, the goal of the S-Puzzle is to arrange the numbers in ascending order.

The goal of the 3-by-3 S-Puzzle is to reach: $\begin{bmatrix}1&2&3\\4&5&6\\7&8&9\end{bmatrix}$

1.2 Example

Here is an example of how to arrive at this goal, from a random start state:

$$\begin{bmatrix}6&1&2\\7&8&3\\5&4&9\end{bmatrix}\quad\begin{bmatrix}1&6&2\\7&8&3\\5&4&9\end{bmatrix}\quad\begin{bmatrix}1&2&6\\7&8&3\\5&4&9\end{bmatrix}\quad\begin{bmatrix}1&2&3\\7&8&6\\5&4&9\end{bmatrix}\quad\begin{bmatrix}1&2&3\\7&8&6\\4&5&9\end{bmatrix}\quad\begin{bmatrix}1&2&3\\4&8&6\\7&5&9\end{bmatrix}\quad\begin{bmatrix}1&2&3\\4&5&6\\7&8&9\end{bmatrix}$$

2. Your Tasks

Implement a solution for the S-Puzzle board using the following 3 search algorithms:

1. depth-first.
2. iterative-deepening.
3. Algorithm A*.

For the informed A* algorithm you need to rely on heuristics, which you will develop by encoding domain-specific knowledge. By their very nature, the informed algorithms are more sophisticated than the brute algorithms. Indeed, if your heuristics are well designed, you will find that you can solve larger S-Puzzle game instances using informed search algorithms.
For the A*, you must develop and experiment with two different heuristics h_1, and h_2. The 2 heuristics that you developed (h_1 and h_2) cannot be equal.
A* is optimal when it is endowed with an admissible (i.e., optimistic) heuristic. When its heuristic is inadmissible, optimality is no longer guaranteed. Still, you will experiment with inadmissible heuristics in this assignment. Since inadmissible heuristics are less constrained than admissible ones, they are often faster, so when they are not very inadmissible (i.e., when they are only ever slightly pessimistic), they may find optimal or only slightly sub-optimal solutions faster than admissible heuristics.

In Modern AI, where the goal is usually to design machines that make rational decisions relative to some objective, the aim of the S-Puzzle game is to develop heuristics that trade-off between optimality and complexity (in terms of time and/or space). That is the focus of attention of this assignment.

2.1 Programming Environment

To program the assignment, you must use Python 3.8. In addition, you must use GitHub (make sure your project is private while developing).

2.2 Data Structures: Input

Your code should represent a state in the S-Puzzle game as an n-tuple of n-tuples. The outer n-tuple represents the rows in the game, while the inner n-tuples represent the entries in the corresponding row's columns. For example, the input state of the example in section 1.2 will be represented with ((6; 1; 2); (7; 8; 3); (5; 4; 9)) and the goal state is represented as: ((1; 2; 3); (4; 5; 6); (7; 8; 9)).

Your code should be able to receive the path of an input puzzle. There is no need to check the validity of the input puzzle; you can assume that it will be correct.

2.3 Algorithms

Recall from class that search can be conceptualized as expanding a tree. Note, however, that no search algorithm builds this search tree in its totality. On the contrary, these algorithms simply move through the search space, by taking actions that lead them from one state to another. Moreover, while it is convenient to think about searching a tree, a search space is not necessarily a tree.
In general, it is a graph, because it is possible to revisit states.

2.4 The output

For each input puzzle and each search algorithm, your program should generate 2 output: one for the solution path, and one for the search path. This means that for each line of the input puzzle, your program should generate 8 text outputs (2 dfs +2 id+ 4 A*: 2 h_1+2h_2)

2.4.1 Solution output

Each solution output will contain the final solution found by the algorithm, or the string `no solution`.
If your program cannot find a solution after 60 seconds, then it should output: `no solution` in both the solution and the search outputs

2.4.2 Search output

Each search output will contain the search path (the list of nodes that have been searched) by the algorithm.

2.5 Analysis

Once your code is running, you will need to analyze and compare the performance of the algorithms and the heuristics. Generate a file with 20 random puzzles, then use it as input to your algorithms and compile the following data for each algorithm:

1. average & total length of the solution and search paths.
2. average & total number of no solution.
3. average & total cost and execution time.

4. optimality of the solution path.

Prepare a few slides explaining your heuristics and presenting and analyzing the above data. You will present these slides at the demo (see Section 4.1.2). Your analysis should compare the data above across search algorithms and across heuristics.

2.6 Scaling Up

Once your analysis is done (see Section 2.5), take your best-performing algorithm and run it on a scaled-up version of the S-Puzzle. This means that instead of using a 3x3 puzzle, increase the dimensions of the puzzle gradually and perform tests with random puzzles to see how the size of the puzzle influences the performance of your search. You can remove the 60-second time-out for this experiment.
Prepare a couple of slides presenting your analysis of the scaled-up puzzle, to present at the demo (see Section 4.1.1).

3 Competition (Just for fun)

Just for the fun of it, we will organize a competition to compare the solutions found by different teams with the same set of input puzzles. This competition will not count for points, we are just doing this for fun (and for the thrill of having your team publicly acknowledged on the Moodle page as the winner of the competition!).
For the competition, we will give all teams some random puzzles, and teams will be ranked based on the total cost of the solutions and execution time. More details on the competition will be posted on Moodle.

4 Deliverables

The submission of the project will consist of 2 deliverables: the code and a demo.

4.1 The Demos

You will have to demo your code to the TAs. Regardless of the demo time, you will demo the program that was uploaded as the official submission on or before the due date. The schedule of the demos will be posted on Moodle. The demos will consist in 2 parts: a small presentation of your analysis and a Q/A.

4.1.1 Presentation of your analysis

Prepare a few slides for a 5 minutes presentation explaining your heuristics, your analysis and your scaled-up experiment (see Sections 2.4 and 2.5).

4.1.2 Q/A

No special preparation is necessary for the Q/A part of the demo. Your TA will give you a small input puzzles and you will run your three algorithms. You will generate the corresponding outputs
In addition, your TA will ask each student questions on the code/assignment, and the student will be required to answer the TA satisfactorily. Hence every member of team is expected to attend the demo. Your individual grade will be a function of your individual Q/A (see Section 5).

5. Evaluation Scheme

Students in teams can be assigned different grades based on their individual contribution to project. Individual grades will be based on:

a) a peer-evaluation done after the submission.
b) the contribution of each student as indicated on GitHub.
c) the Q/A of each student during the demo.

The team grade will be based on:

Code	functionality, correctness and format of input and output, design and programming style,…	**60%**
Heuristics-h_1 and h_2	quality, originality, usability for A*…	**10%**
Demo- QA	clear answers to questions, knowledge of the program, . . .	**10%**
Demo- Presentation	clarity and conciseness, depth of the analysis, presentation skills, Time management . . .	**20%**
Total		**100%**

6 Submission

If you work in a team, identify one member as the team leader. The only additional responsibility of the team leader is to upload all required files from her/his account and book the demo on the Moodle scheduler. If you work individually, by definition, you are the team leader of your one-person team.

Submission Checklist

on GitHub:

- In your GitHub project, include a README.md file that contains on its first line the URL of your GitHub repository, as well as specific and complete instructions on how to run your program.

on Moodle:

- Create a PDF of your slides, and name your slides 472 Slides2 ID1 ID2 ID3 ID4.pdf where ID1 is the ID of the team leader.
- Create one zip file containing all your code, your file of 20 random puzzles, the corresponding output files and the README.md file. Name this zip file 472 Code2 ID1 ID2 ID3 ID4.zip where ID1 is the ID of the team leader.
- Zip 472 Slides2 ID1 ID2 ID3 ID4.pdf and 472 Code2 ID1 ID2 ID3 ID4.pdf and name the resulting zip file: 472 Assignment2 ID1 ID2 ID3 ID$.zip where ID1 is the ID of the team leader.
- Have the team leader upload the zip file on Moodle Assignment 2 submission.

Have fun!