DS/CMPSC 442: Artificial Intelligence
Fall 2023
Project-1 Solve the game of 8-puzzle
(Due: Sep 25th 12:20 PM)

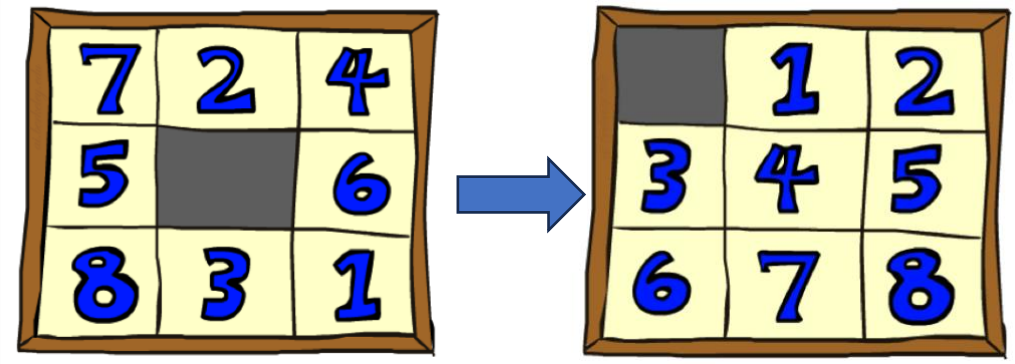

Figure.1 A simple example of 8 puzzle

Project Description

An 8 puzzle is a simple game consisting of a 3 x 3 grid (containing 9 squares). One of the squares is empty. The remaining 8 squares in the grid have a movable tile that is annotated with a digit ranging from 1-8 (hence, the name 8 puzzle). The objective in solving 8-puzzle is to keep moving the tiles around into different positions one by one (so you move one tile at a time, and then you think about your next move) until the numbers on the 3x3 satisfy a "goal test". In this project, we are going to experiment with more than one kind of goal test.

In this project, every student is required to implement search algorithms learned during the class using **Python** 3.9 (this is important, make sure that your code compiles with Python 3.9) to solve the 8-puzzle problem under different scenarios. The detail of each question is mentioned below.

Files to Submit: As part of the project submission, we expect you to submit a zip file which contains only three files. You will need to create and submit 2 files for the assignment by yourselves (See Table 1 below for file name and explanation). ***Make sure that you name these files exactly like this.***

File Name	Explanation
File-1: solution_q1.py	Include all functions that could generate answers for each sub-problem of Q1. Include solutions to all the problems in a single Python file. Make sure that your solution file could be directly run in the terminal (through the command line "python solution_q1.py")
File-2: solution_q2.py	Include all functions that could generate answers for each sub-problem of Q2. Include solutions to all the problems in a single Python file. Make sure

	that your solution file could be directly run in the terminal (through the command line "python solution_q2.py")
File-3: proj-1-submission.pdf	Answer the rest of the questions in this project

Table 1. Submission files for project-1

In solution.py, you should have the ability to read from a file named "input.txt" (includes initial state of 8 puzzle) and print out the corresponding solutions in the terminal. In "input.txt" (***make sure that you use the same file name***), you should have a single line that specifies the starting state of the 8-puzzle problem. As an example, take the initial state in Figure.1 as an example, the input.txt file should contain a single line that looks like this (The symbol "_" represents the empty square):

```
7,2,4,5,_,6,8,3,1
```

As another example, if the initial state that you wanted to specify was the right-hand side state shown in Figure 1, then input.txt should look like this.

```
_,1,2,3,4,5,6,7,8
```

NOTE: Make sure that there is no whitespace in your input.txt, so no space or tab characters in between the numbers. Only use a single comma without any whitespaces to delimit the different numbers in your file.

And the expected output inside the Terminal is (In the expected output window below, the phrase "#answer generated by your function." is a coding style comment. We don't expect that to be part of the output):

```
The solution of Q1.1 is:
#answer generated by your function.

The solution of Q1.2 is:
#answer generated by your function.

The solution of Q1.3 is:
#answer generated by your function.
……
...
...
```

Figure.2 The expected output in the terminal when running "python solution_q1.py"

Evaluation: Your code will be graded manually by TA for technical correctness. To foster the grading process, they will directly check the output printed in the terminal after running the command "python solution.py" with different testing version of "input.py" that the instructor has created. So make sure your code can run in the terminal smoothly before making the submission.

Academic Dishonesty: We will be checking your code against other submissions in the class for logical redundancy (using automated software). If you copy someone else's code and submit it with minor changes, we will know. These cheat detectors are quite hard to fool, so please don't try. We trust you all to submit your own work only; please don't let us down. If you do, we will pursue the strongest consequences available to us.

Now, let's get down to the fun part and let's code up some search algorithms ☺ ☺ ☺

Question-1 Find the solution of 8-puzzle

For question 1, our goal test is the following: "*all numbers should be arranged in increasing order on the 3x3 grid and the empty cell should be on the top left corner*" (HINT: this is exactly the goal test that we have talked about in class, and this goal state can also be seen on the right side of Figure 1 above)

Q1.1 The Coding Component

Q1.1.a

Implement a DFS that can work with an "input.txt" file that contains any possible starting state. Please notice that the output of this implementation should be the actual path returned. Submit your code into File-1(solution_q1.py).

Q1.1.b

Implement a BFS that can work with an "input.txt" file that contains any possible starting state. Please notice that the output of this implementation should be the actual path returned. Submit your code into File-1(solution_q1.py).

Q1.1.c

Implement a UCS that can work with an "input.txt" file that contains any possible starting state. Assume that each tile movement requires paying a cost of 1 unit. Please notice that the output of this implementation should be the actual path returned. Submit your code into File-1(solution_q1.py).

Q1.1.d

Implement A* search that can work with an "input.txt" file that contains any possible starting state. For the choice of your heuristic function, use the Manhattan distance heuristic (this can be found in Slide 64 of the A* slide deck that was covered in class). Submit your code into File-1(solution_q1.py). Please notice that the output of this implementation should be the actual path returned.

Q1.1.e

Implement A* search that can work with an "input.txt" file that contains any possible starting state. For the choice of your heuristic function, use the Straight Line Distance heuristic which calculates the straight line distance between the source tile and the destination tile, and adds this distance across each tile on the 3x3 grid (we covered this heuristic in class). Submit your code into File-1(solution_q1.py). Please notice that the output of this implementation should be the actual path returned.

Q1.2 The Writing Component

For all parts to this question, use the following starting state of the 8-puzzle search problem:
7,4,_,1,2,5,8,3,6

Q1.2.a

For your DFS implementation in Q1.1.a, run it with the start state given above, and tell us how many node expansions of the search tree did DFS do to find the solution? Write down the answer to this question into file-3 (proj-1-submission.pdf).

Q1.2.b

For your BFS implementation in Q1.1.b, run it with the start state given above, and tell us how many node expansions of the search tree did BFS do to find the solution? Write down the answer to this question into file-3 (proj-1-submission.pdf).

Q1.2.c

For your UCS implementation in Q1.1.c, run it with the start state given above, and tell us how many node expansions of the search tree did UCS do to find the solution? Write down the answer to this question into file-3 (proj-1-submission.pdf). Note: Assume that each tile movement requires paying a cost of 1 unit.

Q1.2.d

For your A* implementation in Q1.1.d, run it with the start state given above, and tell us how many node expansions of the search tree did A* search do to find the solution? Write down the answer to this question into file-3 (proj-1-submission.pdf).

Q1.2.e

For your A* implementation in Q1.1.e, run it with the start state given above, and tell us how many node expansions of the search tree did A* search do to find the solution? Write down the answer to this question into file-3 (proj-1-submission.pdf).

Questions-2 Find the solution of modified 8-puzzle

We now want to solve a different search problem. For question 2, our goal test is the following: "*the sum of numbers in the top row of the 8-puzzle should sum up to 11. If that is true, we have reached a goal state*".

Q2.1 The Coding Component

Q2.1.a

Modify your DFS implementation to handle the new goal test. Submit your code into File-2(solution_q2.py). Please notice that the output of this implementation should be the actual path returned.

Q2.1.b

Modify your BFS implementation to handle the new goal test. Submit your code into File-2(solution_q2.py). Please notice that the output of this implementation should be the actual path returned.

Q2.1.c

Modify your UCS implementation to handle the new goal test. Assume that each tile movement requires paying a cost of 1 unit. Submit your code into File-2(solution_q2.py). Please notice that the output of this implementation should be the actual path returned.

Q2.1.d

Modify your A* implementation to handle the new goal test. For the choice of your heuristic function, use the Manhattan distance heuristic (this can be found in Slide 64 of the A* slide deck that was covered in class). Submit your code into File-2(solution_q2.py). Please notice that the output of this implementation should be the actual path returned.

Q2.1.e

Modify your A* implementation to handle the new goal test. For the choice of your heuristic function, use the Straight Line Distance heuristic which calculates the straight line distance between the source tile and the destination tile, and adds this distance across each tile on the 3x3 grid (we covered this heuristic in class). Submit your code into File-2(solution_q2.py). Please notice that the output of this implementation should be the actual path returned.

Q2.2 The Writing Component

For all parts to this question, use the following starting state of the 8-puzzle search problem:
8,1,4,5,2,6,3,7,_

Q2.2.a

For your DFS implementation in Q2.1.a, run it with the start state given above, and tell us how many node expansions of the search tree did DFS do to find the solution? Write down the answer to this question into file-3 (proj-1-submission.pdf).

Q2.2.b

For your BFS implementation in Q2.1.b, run it with the start state given above, and tell us how many node expansions of the search tree did BFS do to find the solution? Write down the answer to this question into file-3 (proj-1-submission.pdf).

Q2.2.c

For your UCS implementation in Q2.1.c, run it with the start state given above, and tell us how many node expansions of the search tree did UCS do to find the solution? Write down the answer to this question into file-3 (proj-1-submission.pdf). Note: Assume that each tile movement requires paying a cost of 1 unit.

Q2.2.d

For your A* implementation in Q2.1.d, run it with the start state given above, and tell us how many node expansions of the search tree did A* search do to find the solution? Write down the answer to this question into file-3 (proj-1-submission.pdf).

Q2.2.e

For your A* implementation in Q2.1.e, run it with the start state given above, and tell us how many node expansions of the search tree did A* search do to find the solution? Write down the answer to this question into file-3 (proj-1-submission.pdf).

----------END OF PROJECT----------