Lab1: The 8-puzzle

(100 pts)

Problem Description

In this programming assignment you will write a code to solve 8-puzzle problems. The objective of the puzzle is to rearrange a given initial configuration (starting state) of 8 numbers on a 3 x 3 board into a final configuration (goal state) with a minimum number of actions.

For this assignment our goal state is:

1 2 3
4 5 6
7 8 _

In the above representation, underscore ('_') represents an empty space. An orthogonally adjacent tile can be slid into the empty space. That's the only way to rearrange a given configuration. Thus given a state, at max 4 actions (**L**eft, **R**ight, **U**p and **D**own) are possible. Some of these actions may not be allowed if the empty tile is at the side or corner.

For example for the initial state below, the goal state can be reached in 4 steps as:

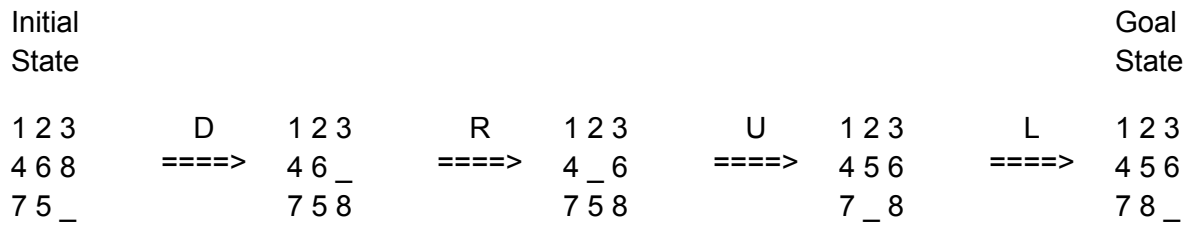

Fig 1: An instance of solution for 8-puzzle.

Task

Your goal is to solve 8-puzzle problems using different search techniques learned in the class and compare them on their run time and number of nodes generated.
Specifically code these 5 algorithms

1. BFS
2. IDS (Iterative deepening DFS)
3. A* with misplaced title heuristic. (h1)
4. A* with Manhattan distance heuristic (h2).
5. A* with one more heuristic (invent or check the literature for this) (h3)

Note 1: Please feel free to use the code for the AIMA book at https://github.com/aimacode.

Note 2: For a new 8-puzzle heuristic, either invent one yourself or find one from the literature (ref. Exercises 3.39 and 3.40 at https://aimacode.github.io/aima-exercises/search-exercises/).

Part 1: (40 pts)

Write a code which will scan the input file from a given path and solve it using one of the five algorithms (you need to code all five algorithms).

Input: Two command line arguments:

1. File path and
2. Algorithm to be used (BFS/IDS/h1/h2/h3).

Output:

1. Total nodes generated (for A* this includes nodes in closed list and fringe).
2. Total time taken.
3. A valid sequence of actions which will take the given state to the goal state.
    a. Please note that the meaning of action is important (action is associated with movement of a tile, not with movement of blank space). For example in Fig1 above, the action sequence is: DRUL.
4. Also note that not all puzzles are solvable (See Appendix A1). Your code should check whether the puzzle is solvable or not before attempting to solve it.

A note on efficiency:

- Some of the algorithms may take a prohibitive amount of time. In such cases terminate your code after a maximum of 15 minutes.
- A* algorithm should not run into memory or time issues, if implemented properly.

Below are sample outputs for A* for two heuristics h2 (Manhattan distance) and h1 (misplaced tiles). Your code should work in similar fashion for other algorithms. Please note:

- Exact format of the output is not important, it should capture all the information though.
- Your solution can take different paths and can generate a different number of nodes. Also different heuristics could give different optimal paths. However the optimal path length should be the same.

In the following, the state in 140173.txt is:
5 3 1
_ 8 7
2 6 4

python3 8puz.py --fPath 140173.txt --alg h2
Total nodes generated: 3827
Total time taken: 2 sec 576648 microSec.
Path length: 25

Path: ULLDRRULLDRDLURRDLLURRULL

python3 8puz.py --fPath 140173.txt --alg h1
Total nodes generated: 42123
Total time taken: 57 sec 99037 microSec.
Path length: 25
Path: ULLDRRULLDRDLURRDLLURRULL

If your code fails to find a solution in 15 minutes, report that it timed out
python3 8puz.py --fPath 140173.txt --alg IDS
Total nodes generated: <<??>>
Total time taken: >15 min
Path length: Timed out.
Path: Timed out.

If the puzzle is not solvable, your code should detect that and report like:
python3 8puz.py --fPath nonSolvable.txt --alg h1
The inputted puzzle is not solvable:
2 1 3
8 _ 4
7 5 6

Part 2: (20 pts)

Now, run your code on all the 5 states given in Part2.zip for all the 5 algorithms (i.e., in total there'll be 25 runs). Compile all your results into a single file, indicating which file/state leads to which output for each algorithm.

Part 3: (40 pts)

In this part, you will compare the performance of the algorithms you have coded.

1. In file part3.zip you'll find 60 8-puzzles. 20 from each of 8, 15, and 24 levels, where level indicates the optimal path length of the state from the goal.
2. For states in each level solve the puzzle and calculate the average run time and average nodes generated for all the five algorithms.
3. Tabulate your results in the form of a table as shown below. Also discuss conclusions drawn from the performance of different algorithms.

Depth	**BFS**		**IDS**		**A*: h1**		**A*: h2**		**A*: h3**	
	Avg run time	**Avg #nodes Explr'**	**Avg run time**	**Avg #nodes Explr'**	**Avg run time**	**Avg #nodes Explr'**	**Avg run time**	**Avg #nodes Explr'**	**Avg run time**	**Avg #nodes Explr'**
8										
15										

24										

You don't need to generate the table programmatically, however your submission should have information on how to reproduce your result if needed.

Choice of programming language:

You may choose to code in any of the following programming languages:

- python (recommended)
- Java
- C/C++
- Or any popular programming language with instructions on how to run your code.

What to Turn In

1. All of your source code.
2. A README file explaining how to compile and run your program.
3. A short lab report that includes:
    a. Results for Part 2 of the task.
    b. Table generated in Part 3 of the task along with information on how to reproduce your results if needed, and conclusions drawn.

Appendix

A1: Solvability of 8-puzzle

Not all the states in the 8 puzzle are reachable from each other. In fact the states are divided into two separate sets. There is a simple way to check if a puzzle is solvable or not, by counting total numbers of "inversions" pairs. If the number is even, then the puzzle is solvable else it is not.

What is an inversion? We consider our goal state as the one where all tiles have correct order (not including '_')
1 2 3
4 5 6
7 8 _

From top to bottom and left to right (i.e, if we arrange the tiles row by row), the order of the tiles is: 1 2 3 4 5 6 7 8 _. Here, every smaller number tile appears before a bigger number tile. In this case we say that there is no inversion.

Now consider another state say:
1 2 3
8 _ 4
7 6 5

For this state if we list the tiles in a similar fashion, we get: 1 2 3 8 _ 4 7 6 5.

Here note that, the following pairs of tiles are not in correct order (i.e., a bigger number tile comes before a smaller number tile). Each such pair is violating the order and is an “inversion”.

- (8, 4)
- (8, 7)
- (8, 6)
- (8, 5)
- (7, 6)
- (7, 5)
- (6, 5)

Thus there are a total of 7 inversions. And as 7 is odd, the above state is not solvable.

Consider another state:
5 3 1
_ 8 7
2 6 4

Here the order is: 5 3 1 _ 8 7 2 6 4. All the pairs in inversion are:

- (5, 3)
- (5, 1)
- (5, 2)
- (5, 4)
- (3, 1)
- (3, 2)
- (8, 7)
- (8, 2)
- (8, 6)
- (8, 4)
- (7, 2)
- (7, 6)
- (7, 4)
- (6, 4)

Here total inversions are: 14. Since 14 is even, this is a solvabel state.

Logic for this rule is: From any state if we would do a legal state change then it will change the inversions by an even amount.