CS461 Advanced Artificial Intelligence

Assignment 1: The Eight Puzzle Problem – BFS and IDS

In this assignment, you are going to write a program that solves the eight puzzle problem. You will solve it using

1) [20 points] Iterative deepening depth-first search (100 cases in the given file)
2) [20 points] Breadth First Search (If it takes too long in your BFS, you need to give the results based on at least 10 cases. But you will lose 5 points.)

To test your program and analyze the efficiency, you need to test your program on the 100 different input cases. The input data is *Input8PuzzleCases.txt*. This file contains a hundred different states of 8-puzzle. All the given cases are ***solvable***.

An 8-puzzle case is given in the format as the following line,

3, 2, 4, 5, 8, 6, 0, 1, 7

means state

3	2	4
5	8	6
0	1	7

You need to find solutions for all 100 different states to the goal state.

The Goal state is

0	1	2
3	4	5
6	7	8

Some Hints:

1. **Handling Puzzle Moves**
   - There are four possible moves in the 8-puzzle: up, down, left, right.
   - Ensure that moves stay within the grid boundaries.
   - The empty space (0) swaps places with the adjacent tile when a move is made.
   - Use tuple or list representations for states.

2. **Breadth-First Search (BFS):**
   - You need to maintain a queue (FIFO structure) to store the frontier (nodes to be explored).
   - Use a set or dictionary to track visited states to avoid redundant searches.

Implementation Steps:

1. Initialize a queue with the start state.
2. Use a loop to explore states until the goal is found:
    - Dequeue a state.
    - If it's the goal, return the solution path.
    - Otherwise, generate all possible next states (valid moves).
    - Add new states to the queue if they haven't been visited.
3. Stop when the goal is reached or all possibilities are exhausted.

Potential Challenges:

- BFS can consume a lot of memory if many states are stored.

2. Iterative Deepening Depth-First Search (IDS)

- IDS is a combination of Depth-First Search (DFS) and Breadth-First Search.
- It uses a depth limit, increasing it iteratively until a solution is found.
- This prevents DFS from getting stuck in deep branches.

Implementation Steps:

1. Start with depth 0 and perform DFS with a depth limit.
2. If the goal state isn't found, increase the depth limit and restart DFS.
3. Continue until a solution is found.

Potential Challenges:

- **Efficiency**: Since IDS repeatedly runs DFS at increasing depths, it can be slower than BFS in some cases.
- **Stack limit**: DFS can run into recursion depth limits in Python.

Requirements:

1. The search algorithms must be programmed by you. Plagiarism will be filed.

2. You are ***required*** to use the given IPython Notebook file to do this assignment.
3. You need to install Anaconda Environment to use the notebook file. Refer to the following links.
    a. https://www.datacamp.com/community/tutorials/tutorial-jupyter-notebook
    b. https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/
4. You are required to follow the notebook file's requirement to print out results.
5. The search algorithms must be implemented by you! That means you are not allowed to import Iterative deepening depth-first search or BFS from other libraries. Otherwise, the assignment will receive 0.
6. You need to follow our class to design your program. For example, you need to implement a frontier queue.