Week 2

Solve 8 puzzle problem using DFS and BFS

Algorithm

1. Input the Puzzle
   Prompt the user to input the initial state and the goal state of the puzzle.
   Each state is a 1D list of 9 elements where 0 represents the blank tile.

2. Choose Algorithm:
   Ask the user to choose which search Algorithm to use:
   BFS: Guarantees the shortest solution path but can consume more memory.
   DFS: ~~Guarantees the shortest~~ solution Can be more efficient but might not guarantee the shortest path.

3. Initialization:
   BFS: Use a Queue (FIFO) to store the current state and the path taken to reach it.
   ~~Initialize the queue with start state and an empty path.~~
   Maintain a visited list to avoid revisiting the same state.
   DFS: Use recursion to explore deeper branches first.
   Use a repeated list to avoid cycles & revisiting states.
   Initialize with the start state and an empty path.

4. BFS Procedure:

- While the queue is not empty.
- Dequeue the first state from the queue.
- if the dequeued state is goal state, return the sequence of moves (solution path)
- ~~If the~~ Get the position of the blank state and generate all possible the moves for the current state and the blank state.
- ~~(Append the move and the path in the Queue.)~~
- For each valid move:
  Create a new state by swapping the blank tile with the adjacent tile.
- If the new state has not been visited, add it to the queue with the updated path and mark it as visited.

5. DFS Procedure:

- Use a recursive function to explore the current state
- if the current state is the goal state, return the solution path.
- Mark the current state as visited.
- Get the position of the blank tile and generate all possible moves (up, down, left, right).
- For each move:
  Recursively explore the new state.
  If a valid solution is found, return the path.
  If no solution is found, backtrack to explore other states.

6. Move Generation
- Find the position of the blank tile in 3x3 grid.
- For each direction (up, down, left, right), calculate the new position of the blank tile.
- If the new position is within the grid bounds, generate a new state by swapping the blank tile with the adjacent tile.

7 Check for Goal State:
After every move, compare the current state with the goal state! if they match, the puzzle is solved.

8. Output:
if a solution is found, output the sequence of moves (states) that lead to the goal state.
if no solution is found, report that no solution exists.

State Space Tree

Initial state

1	2	3
8	6	0
7	6	5

Final State

1	2	3
8	0	4
7	6	5

Initial state

1	2	3
8	6	0
7	5	4

Right 6

1	2	3
8	0	6
7	5	4

up 4

1	2	3
8	6	4
7	5	0

down 3

1	2	0
8	6	3
7	5	4

Right 5

1	2	3
8	6	4
7	0	5

down 4

1	2	3
8	6	0
7	5	4

down 6

1	2	3
8	0	4
7	6	5

~~right~~ down 4

1	2	3
8	6	0
7	5	4

Goal state. (Final state)

Time Complexity:

$$\boxed{TC_{(8puzzle)} = O(b^d)}$$

b → branching factor.
d → depth of the solution.

08.10