07/10/24

03> Solve 8 puzzle problem using DFS and BFS.

Algorithm:

BFS Algorithm:

step1: start at a chosen node (source node)

step2: Mark the node as visited.

~~step3:~~

=>

1. Input the ~~user~~ puzzle.

prompt the user to input the initial state and the goal state of the puzzle.

(eg $\begin{bmatrix} 1, 2, 3, \\ 4\ 0, 5 \\ 6\ 7\ .8 \end{bmatrix}$) where '0' represent the black tile.

2. Choose Algorithm:

BFS: Generate the shortest solution path but can consume more memory

DFS: Can be more memory-effecent but might not generate the shortest path.

3. Intialization:

BFS: use a queue (FIFO) to store the current state and the path taken to reach it.

DFS: use a recursion to Explore deeper branches first.

use a visited list to avoid cycles and ~~[illegible]~~. revisiting state.

4. BFS procedure:

- when the queue is not empty
- Dequeue the first state from the queue.
- If the dequeued state is the goal state, return the sequence of moves. (up, down, left, right).
- for each valid move; create a new state ~~has~~ ~~Previously~~ ~~not~~ ~~been visited~~ by swapping the blank tile with the adjacent tile.

5. DFS procedure:

- use the recursivefunction to Explore the current state.
- If the current state is the goal state, return the solution path.
- Mark the current state as the visited,
- Get the position of the black tile and generate all possible moves (up, dow, left, right).

6. Move Generation:

- find the position of the blank tile the 3x3[or any square matrix] grid.
- for each valid direction (up, down, left, right), calculate the new position of the black tile.

7. check for goal state;

- After every move, compare the sequence of moves. with current stat with the goal state.
- If they math, the puzzle is solved.

8. output.

- If a solution is found, output the sequence of moves the lead to the goal state.
- If no. solution is found, report that no solution exists.

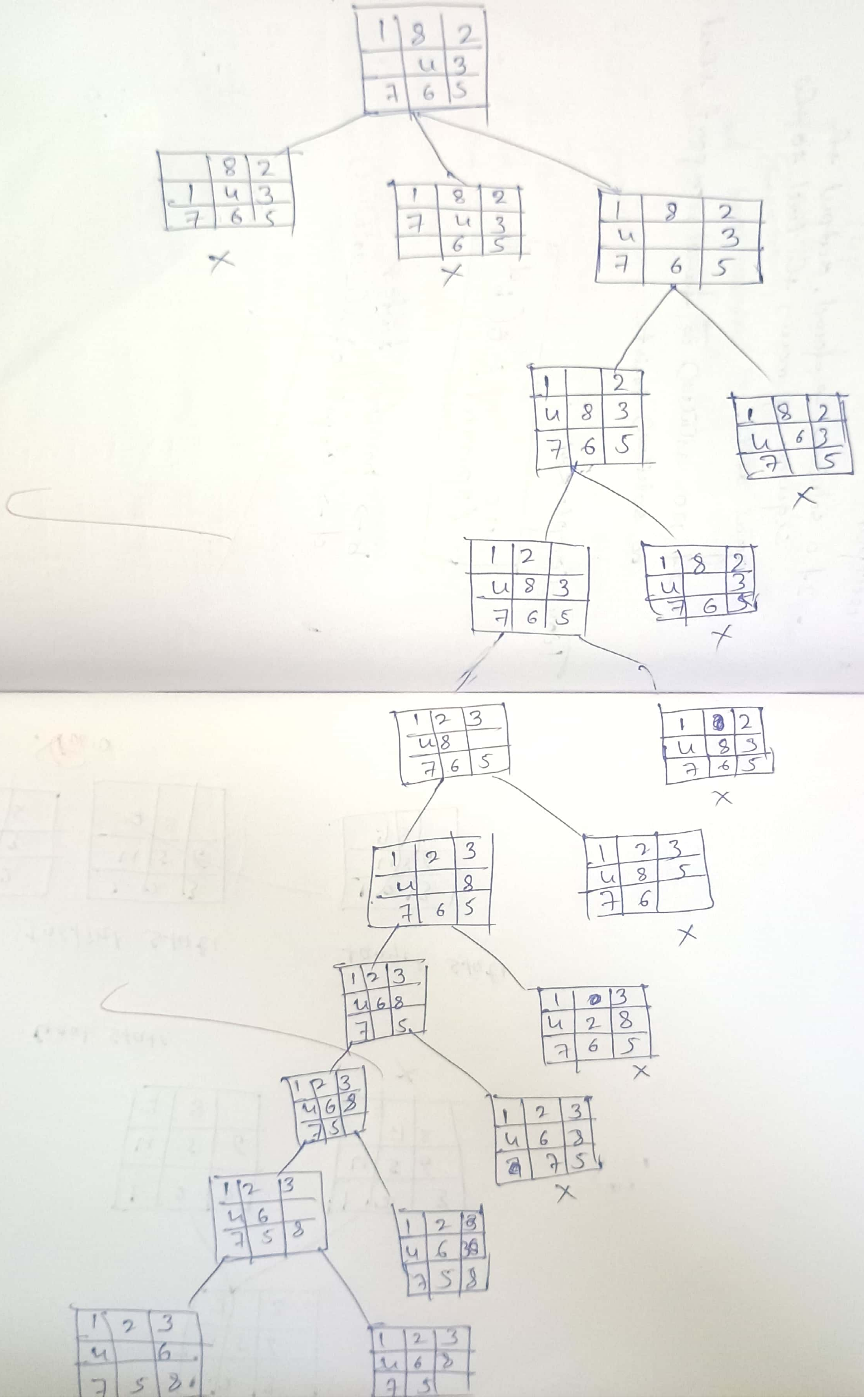
Time complexity:

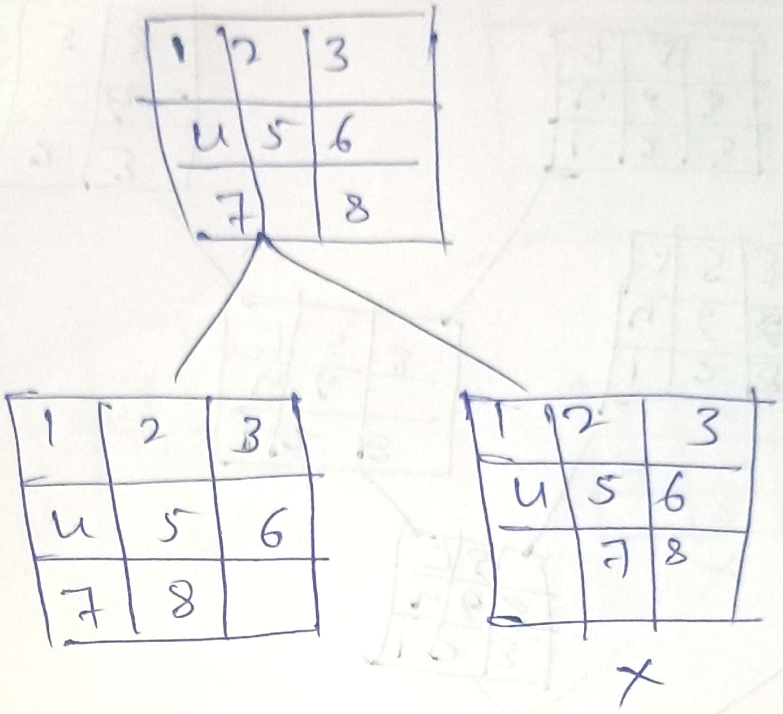
$$\boxed{TC_{(8\,puzzle)} := O(b^{d})}$$

b -> branching factor

d -> depth of the solution

~~Stack Space~~ State space Tree





1	2	3
4	5	6
7		8

1	2	3
4	5	6
7	8	

1	2	3
4	5	6
	7	8

X

Goal state.

Initial state

target state

1	8	2
	4	3
7	6	5

1	2	3
4	5	6
7	8	

1	2	3
4	5	6
7	8	

08.10