PAGE NO :
DATE :

08/10/24

Breadth

8 Puzzle using BFS and DFS

BFS (Breadth first Search)

→ Explores all possible moves level by level before going deeper
→ Guarantees the shortest path by using a queue to explore states in order

Algorithm

① Initialize
→ Create a queue queue and enqueue the initial state along with an empty path
→ Create a set visited to keep track of visited States

② Loop
→ While the queue is not empty
① Dequeue the front element from the queue let this be the current_State and its associated path
② If the current State is the goal_state, return the path
③ Add the String representation of the current_State to the visited set
④ For each possible move (up, down, left, right)
→ Generate the new_state after performing the move on the current_state

→ If the new_State is valid and hasn't been visited
• Enqueue the new_State along with the updated path.

③ End
→ If the queue becomes empty and no solution is found, return "No Solution found"

② DFS [Depth first Search].

→ Follows one branch as deep as possible before backtracking.
→ May not find the shortest path but is faster in some cases using a stack to explore the most recent state first.

Algorithm

① Initialize
→ Create a stack Stack and push the initial state along with an empty path
→ Create a Set visited to keep track of visited states

② Loop
→ While the stack is not empty
1. pop the top element from the stack let this be the current_State and its associated path.
→ If the current_State is the goal State return the path

PAGE NO :
DATE :

→ Add the string representation of the current_state to the visited set.

→ For each possible move (up, down, left, right):

→ Create the new_state after performing the move on the current_state

→ If the new_state is valid and hasn't been visited

→ push the new_state onto the stack along with the updated path

End

→ If the stack becomes empty and no solution is found, return "No solution found"

OUTPUT

Initial State :

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 0 & 6 \\ 7 & 5 & 8 \end{bmatrix}$$

Solve using BFS

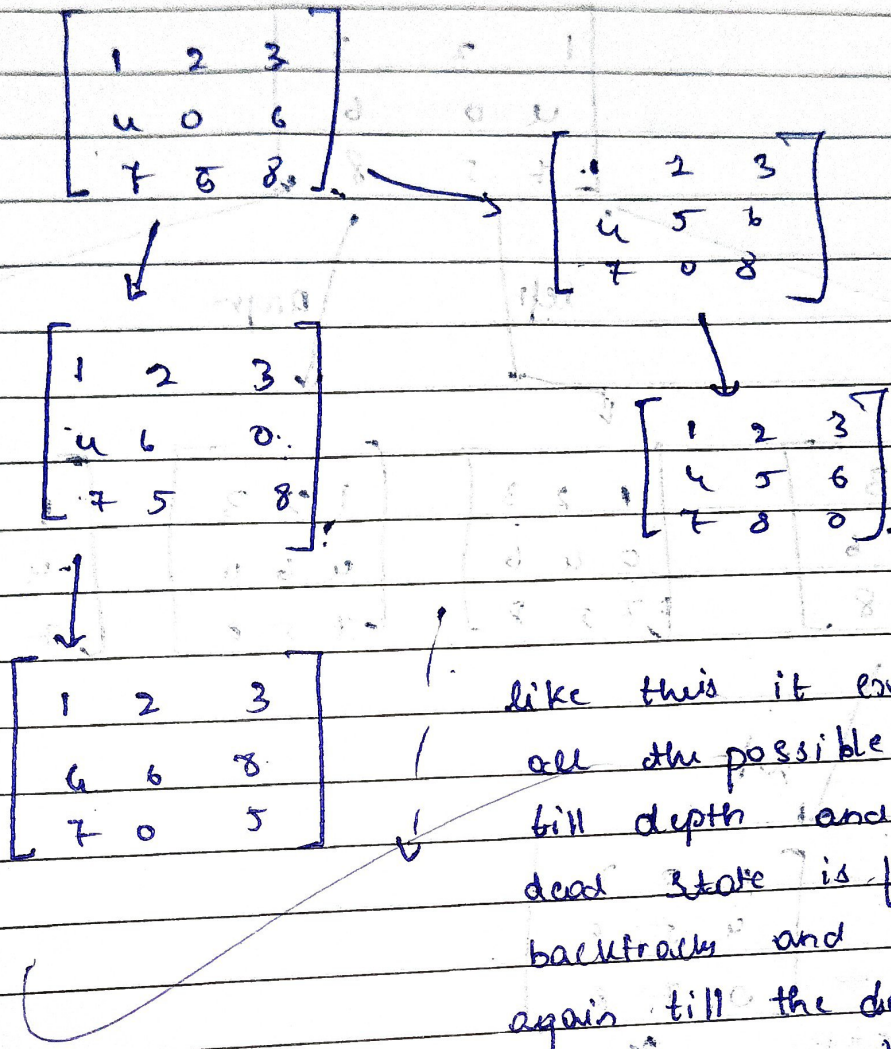
$$\begin{bmatrix} 1 & 0 & 3 \\ 4 & 2 & 6 \\ 7 & 5 & 8 \end{bmatrix} \quad \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 0 & 8 \end{bmatrix} \quad \begin{bmatrix} 1 & 2 & 3 \\ 4 & 6 & 0 \\ 7 & 5 & 8 \end{bmatrix} \quad \begin{bmatrix} 1 & 2 & 3 \\ 0 & 4 & 6 \\ 7 & 5 & 8 \end{bmatrix}$$

$$\downarrow$$

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{bmatrix}$$

PAGE NO :
DATE :

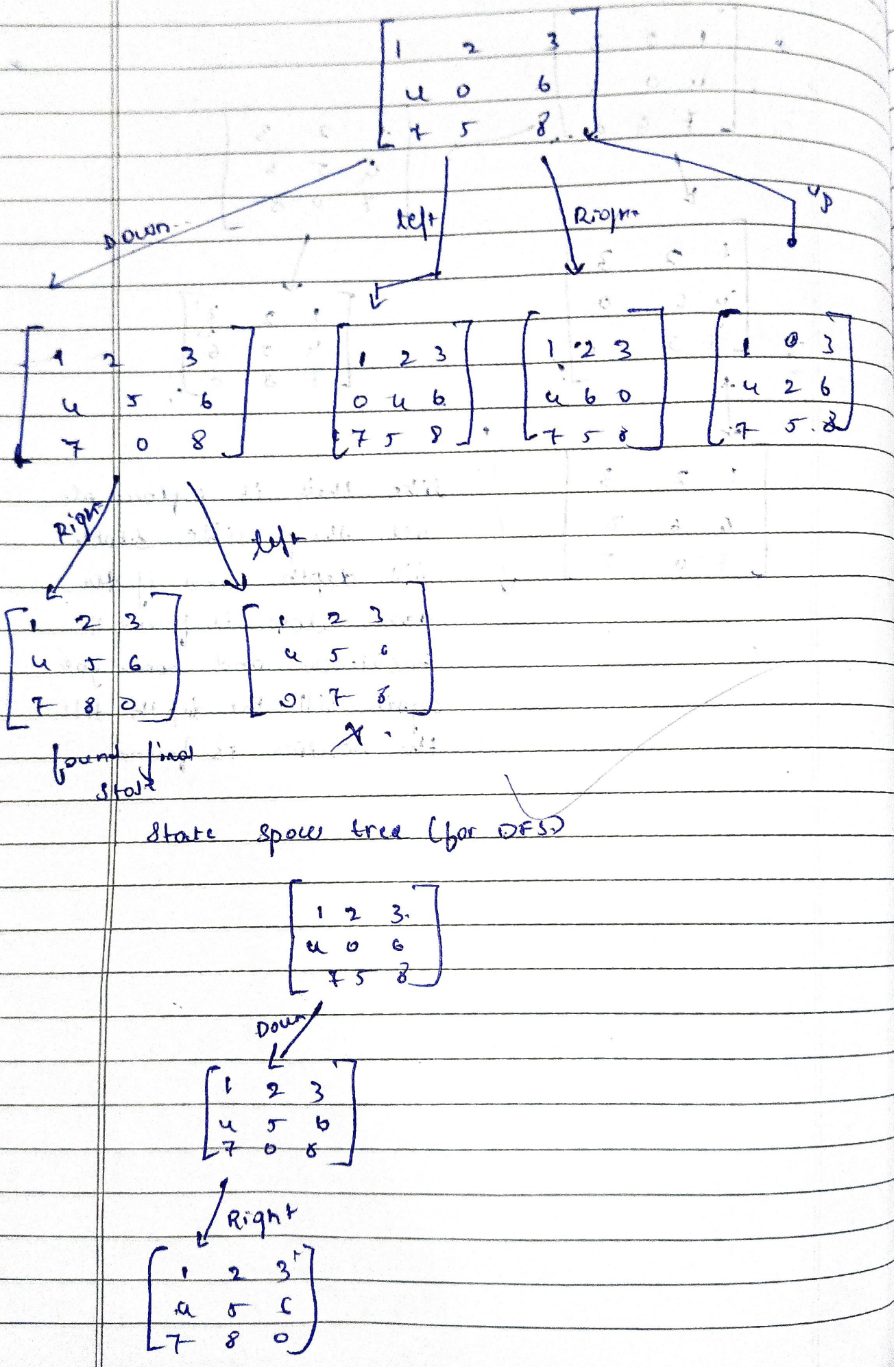
using DFS



like this it explores all the possible states till depth and if the dead state is found it backtracks and ~~come~~ goe again till the depths till the solution is found

PAGE NO :
DATE :

State space tree :



State space tree (for DFS)

PAGE NO :
DATE :

Evaluation steps

① BFS

① Completeness : BFS guarantees find a solution if one exists

② Time Complexity :- $O(b^d)$ ie

b -> branching factor

d -> depth

③ Space Complexity :-

$O(b^d)$.

b -> branching factor

d -> depth

④ optimal : BFS is optimal

② DFS

① Completeness :- DFS may not find solution if one exists due to its tendency to explore deep branches

② Time Complexity :-

$O(b^m)$

b -> branches

m -> maximum depth

③ Space Complexity

$O(b.m)$.

b -> branches

m -> maximum depth

S. Pradeep
8/10/24

④ optimal :- DFS does not guarantee optimal solution