Solving 8-puzzle using DFS :-

→ Considering 3×3 [2D] grid
importing Random states for each individual grid expect one of them.
Totally 8-puzzle consists of 8 numbered tiles
Empty space → 0
The goal is to move tiles in a order that they fit to form a Ideal or goal 8-puzzle form.

Initial state

1	0	2
3	4	6
7	5	8

Our Goal

1	2	3
4	5	6
7	8	

DFS uses stack data structure, often implicitly via recursion, so always keep a track of visited nodes

initial state

initial_state = (1,0,2,3,4,6,7,5,8)

Determine possible moves (up, down, left, right) based on the position of 0.

Set a depth limit to avoid running indefinitely in case no solution found.

Date __/__/__
Page

✱ Functions:—

- Base Case: If the current state is the goal state, return the path.
- Mark the current state as visited.
- Neighbor (possible move):
  - If neighbor is not visited:
  - Recursively perform DFS on the neighbor
  - If a solution is found return the path.
- Just back track if no solution is found in the current path.

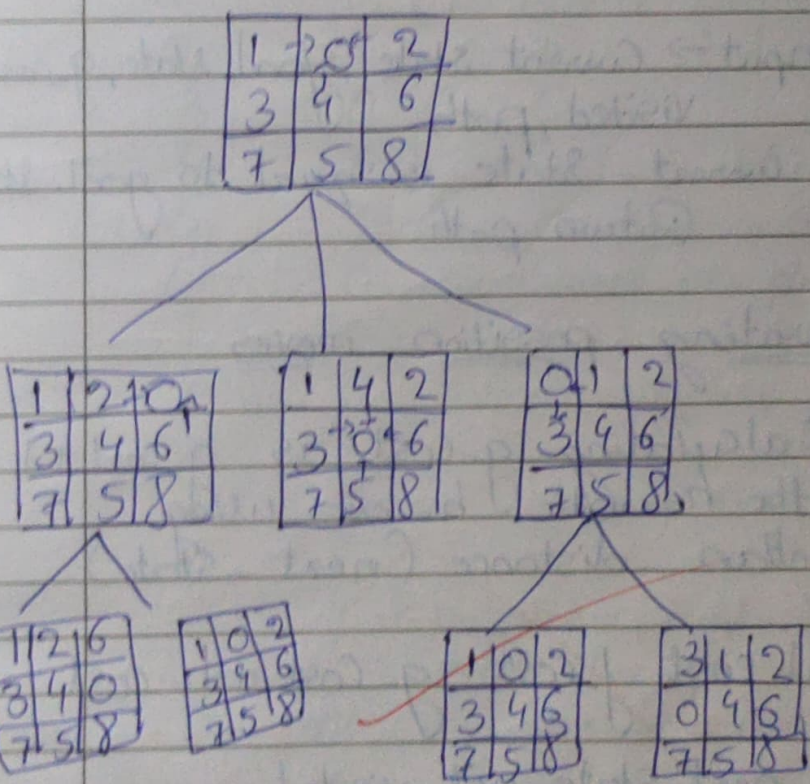
Algo

```
def get_possible_moves(index: int) -> list[i
    moves = []
    row, col = divmod(index, 3)
    directions = {
        'up': (-1, 0),
        'down': (1, 0),
        'left': (0, -1),
        'right': (0, 1),
    }

def swap_positions(state: Tuple[int],
    pos1: int, pos2: int) -> Tuple[int]:

    state = list(state)
    state[pos1], state[pos2] = state[pos2], state[
    return tuple(state)
```

```
def dfs_8_puzzle(start_state: Tuple[int],
    goal_state: Tuple[int]) -> Optional
    [List[Tuple[int]]]:

while stack:
    current_state, path = stack.pop()

    if current_state == goal_state:
        return path

    if current_state in visited:
        continue
```

Goal

1	2	3
4	5	6
7	8	0



Date __/__/__
Page

Solving 8-puzzle using Manhattan Distance

Initializing the grid

- define the initial state of puzzle
- Also define goal state
- Arrange number tiles from (1 to 8)
- Empty tile -> 0

Defining Heuristic Functions -

Create a function manhattan(state)
initializes variable distance to 0
Calc target position based
on tile value

» Input -> Current state, goal state, g_cost
visited, path
if current - state is Equal to goal state
Return path.

Generating possition moves

Calculating g_cost as g_cost + 1
cal the heuristic h_cost using
manhattan_distance (next_state)

total cost f_cost = g_Cost + h_cost

if next state not visited.

Add (f_cost, next_state, updated path
to priority queue)

Date / /
Page 17

Manhattan Distance = $|i - goal_i| + |j - goal_j|$

Compared to goal state calculate the distance from the given state.

initial State

1	0	3
2	4	6
7	5	8

Goal

1	2	3
4	5	6
7	8	0

	V	H
1	0	0
2	1	1
3	0	0
4	0	1
5	1	0
6	0	0
7	0	0
8	0	1

Assign this to priority queue

↓

Calc $f(n)$

lowest $f(n)$ is the optimal

8/10/24