8/10/24 DATE: PAGE:

Lab - 3.

Algorithm:

1) Input double ended queue from the collections.
2) Initialize a final goal state in form of a matrix.
3) Now, Initialize a moves like up, down, left, right.
4) We are taking 0 for the empty cells.
5) Define a function called manhattan distance with parameter state.

```
distance = 0
for i in range(3):
for j in range(3):
    if state[i][j] != 0
        goal_i, goal_j = divmod(state[i][j]-1, 3).
```

6) Define a goal_state. to return the state of the goal.
7) Define a get_neighbours :

```
neighbours = []
for i in range(3):
    for j in range(3):
        if state[i][j] == 0:
            for move in moves:
            new_i, new_j = i + move[0], j + move[1].
```

It will check the row in the state and it will append the neighbours. return neighbours.

```
neighbours.append(new_state)
return neighbours.
```

Galaxy M33 5G

DATE: PAGE:

7) define a function dfs with parameter state.
take a two queue one for dequeue and another visited.

while queue():

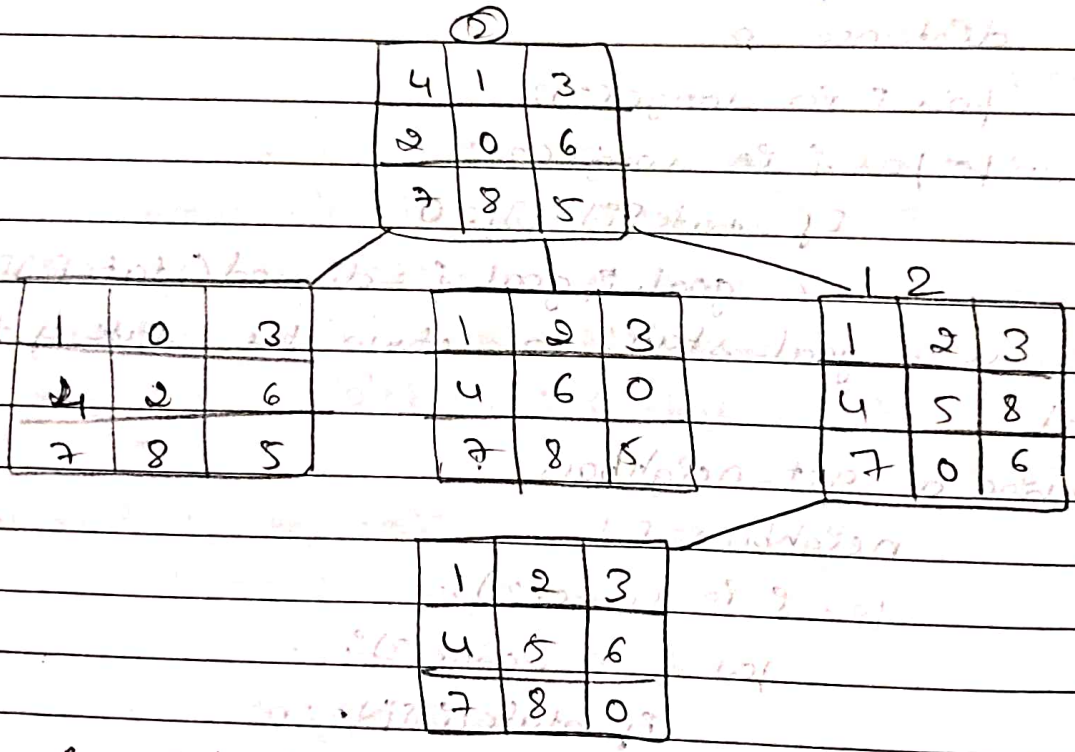
current_state, path = queue.popleft().

8) if goal state (current_state) return path.

9) ~~Take~~ using a map function we are converting tuple to state.

10) Take a initial_state, assign a path = dfs (initial_state) to

11) check the path if path is to a state it will print the solution ② else it won't print a solution.



i=2, j=1

new_i = i + move[0] new_j = j + move[1]

= 2 + -1 = 1 + 0

= 1 = 1

Proved

Galaxy M33 5G

DATE: PAGE:

```
from collections import deque
GOAL_STATE = [[1,2,3],
              [4,5,6],
              [7,8,0]]
MOVES = [(-1,0),(1,0),(0,-1),(0,1)]
def manhattan_distance(state):
    distance = 0
    for i in range(3):
        for j in range(3):
            if state[i][j] != 0:
                goal_i, goal_j = divmod(state[i][j]-1,
                                        3)
                distance += abs(i-goal_i)+abs(j-goal_j)
    return distance
def is_goal_state(state):
    return state == GOAL_STATE
def get_neighbours(state):
    neighbours = []
    for i in range(3):
        for j in range(3):
            if state[i][j]==0:
                for move in MOVES:
                    new_i, new_j = i+move[0],
                                   j+move[1]
                    if 0<=new_i<3 and
                       0<=new_j<3:
                        new_state = [row[:] for
                                     row in state]
                        new_state[i][j], new_state
                        [new_i][new_j]=new_state[new_i][new_j],
                        new_state[i][j]
```

laxy M33 5G

DATE: PAGE:

```
            neighbours.append(new_state)
    return neighbours


def dfs(state):
    queue = deque([(state, [state])])
    visited = set()
    while queue:
        current_state, path = queue.popleft()
        if is_goal_state(current_state):
            return path
        if tuple(map(tuple, current_state)) in
                visited:
            continue
        visited.add(tuple(map(tuple, current_state)))
        for neighbor in get_neighbours(current_state):
            queue.append((neighbor, path +
                    [neighbor]))

    return None
initial_state = [[4,1,3],
                 [7,2,6],
                 [5,8,0]]
path = dfs(initial_state)
if path:
    print(f"Solution found in {len(path)-1} moves:")
    for state in path:
        for row in state:
            print(row)
        print()
else:
    print("No solution found")
```

Galaxy M33 5G

DATE: PAGE:

output:

Solution found in 10.9 moves:

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

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

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

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

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

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

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

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

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

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

Galaxy M33 5G