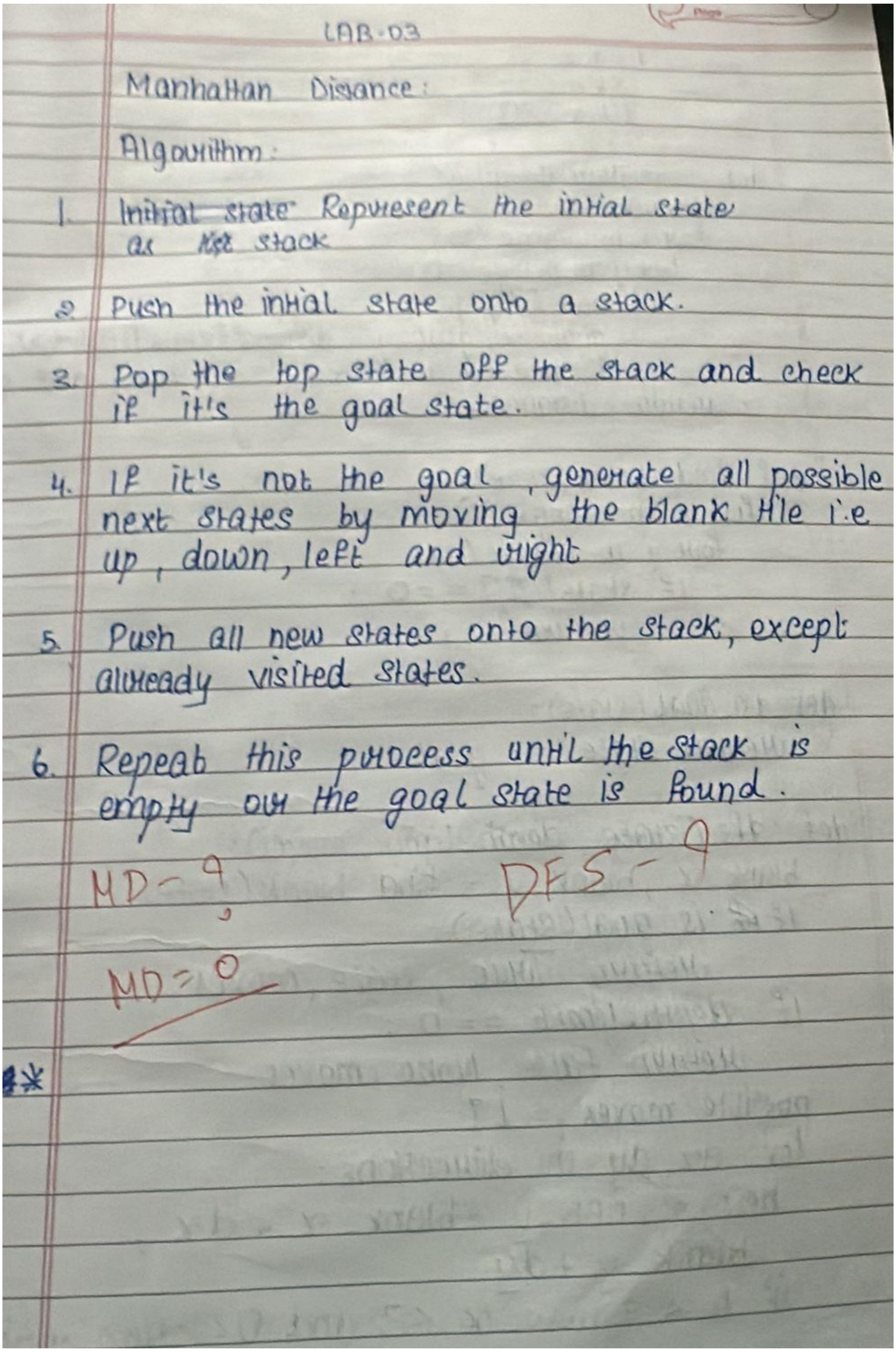
LAB 3 : 8 puzzle problems using DFS and Manhattan distance.

Observation book:



LAB-03

Manhattan Disance:

Algouithm:

1. ~~Initial state~~ Repuresent the initial state as ~~the~~ stack

2. Push the initial state onto a stack.

3. Pop the top state off the stack and check if it's the goal state.

4. If it's not the goal, generate all possible next states by moving the blank tile i.e up, down, left and right

5. Push all new states onto the stack, except aluready visited states.

6. Repeat this puuocess until the stack is empty our the goal state is found.

MD - 9 DFS - 9

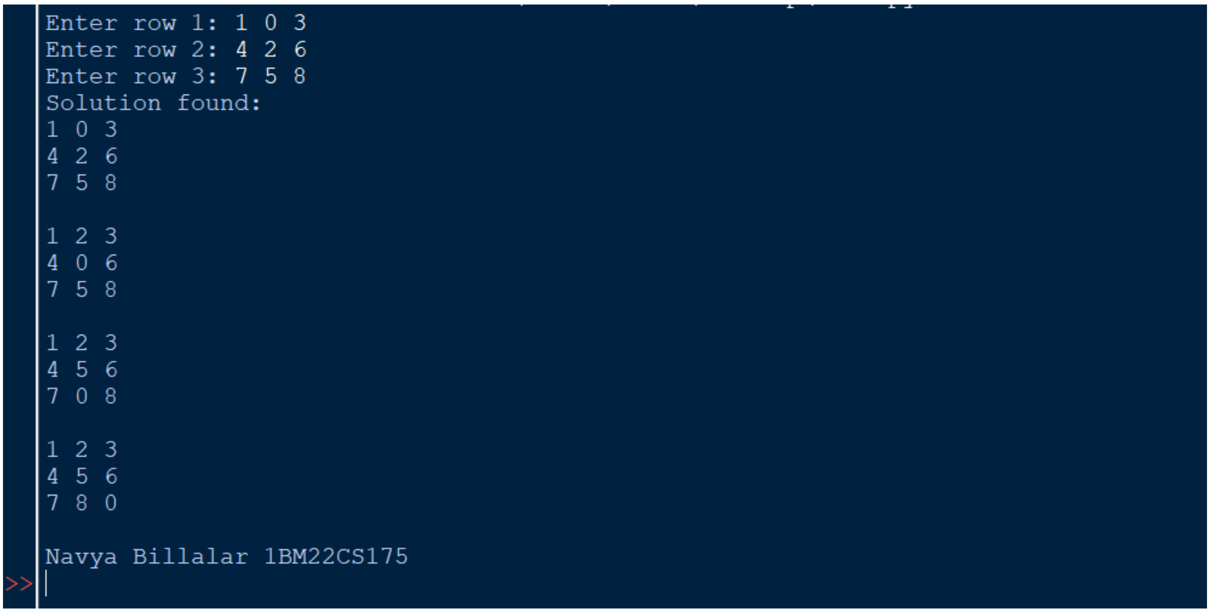
MD = 0

*

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

def manhattan_distance(state):
    distance = 0
    for i in range(3):
        for j in range(3):
            if state[i][j] != 0:
                goal_x, goal_y = divmod(state[i][j] -1, 3)
                distance + = abs (i -goal_x) + abs(j - goal_y)
    return distance

def find_blank(state):
    for i in range (3):
        for j in range(3):
            if state[i][j] = = 0 :
                return i, j

def is_goal(state):
    return state == goal

def dfs(state, depth_limit, moves):
    blank_x , blank_y = find_blank(state)
    if is_goal(state):
        return True , state, moves
    if depth_limit == 0 :
        return False, None , moves
    possible_moves = []
    for dx, dy in directions:
        new_x, new_y = blank_x + dx ,
            blank_y + dy
        if 0 < = new_x < 3 and 0< = new_y <3:
```

```
        new_state = copy.deepcopy(state)
        new_state[blank_x][blank_y], new_state[
            new_x][new_y] = new_state[new_x]
            [new_y], new_state[blank_x][blank_y]
        md = manhattan_distance(new_state)
        possible_moves.append((md, new_state))
        possible_moves.sort(key = lambdax :x[0])
       for _, next_state in possible_moves:
            moves.append(next_state)
       print("move made:")
       print_board(next_state)
      found, result, moves = dfs(next_state, depth_
      limit-1, moves)
       if found:
           return True, result, moves
       moves.pop()
      return False, None, moves

 def solve_puzzle(initial_state, depth_limit = 30):
 moves = [initial_state]
 print("initial state :")
 print_board(initial_state)
 found, final_state, moves = dfs(initial_state,
 depth_limit, moves)
 if found:
      print("Solution found!")
      ~~print( "Final~~
      print_board(final_state)
else:
      print("no solution")
Initial_state = [[1,2,3], [4,0,6], [7,5,8]]

solve_puzzle(initial_state)
```

Output:



```
Enter row 1: 1 0 3
Enter row 2: 4 2 6
Enter row 3: 7 5 8
Solution found:
1 0 3
4 2 6
7 5 8

1 2 3
4 0 6
7 5 8

1 2 3
4 5 6
7 0 8

1 2 3
4 5 6
7 8 0

Navya Billalar 1BM22CS175
>>
```