LAB (3) :- 8 puzzle problem : (8|10|24)

1	2	3
0	5	6
4	7	8

start state (convert to arr)

1	2	3
4	5	6
0	8	7

Final state (convert to arr)

stack
visited set

→ push in the stack the current arr
→ pop the top & u get top_state = current arr
return ...

→ shuffle left, right, up, down adjacent to 0 and call it again until u get the result.

Algorithm (using DFS)

```
start_state = [ . . . ]
goal_state = [ . . . ]
stack.push(start_state)
visited_set = { }
moves = 0
~~while (stack.isempty())~~ f(i,j)
{
    visited_set.add(curr state)
    if (curr.state == goal_state) return moves
    moves++;
    if (not in visited set)    // recursion
    {
        left = f(i, j+1)
        right = f(i, j-1)
        up = f(i-1, j)
        down = f(i+1, j)
    }
}
print(moves)
```

1	2	3
0	5	6
4	7	8

→

1	2	3
4	5	6
0	7	8

→

1	2	3
4	5	6
7	0	8

→

1	2	3
4	5	6
7	8	0

1	2	3
4	5	6
0	8	7

Proceed

Algorithm (using manhattan dist)

DFS_Manhattan (start, goal):

{

neighbours = [ ]

for each move (up, down, left, right):

move (new)

if new is not in visited:

dist = manhattan (next, goal)

add (new, distance) to neighbours

sort neighbours by distance (lowest first)

for each (new, dist) in neighbours:

add to stack

return "no solution"

Initial:

8	7	3
4	0	5
2	6	1

Final:

1	2	3
4	5	6
7	8	0

Code:

```
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]
                        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)))
    return None

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

path = dfs(initial_state)
if path:
    print("solution found)
    for state in path:
        for row in state:
            print(row)
    print()
```

Output:

Solution found:

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

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

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

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

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

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

[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, 8, 0]

No. of moves: 8

8/10/24