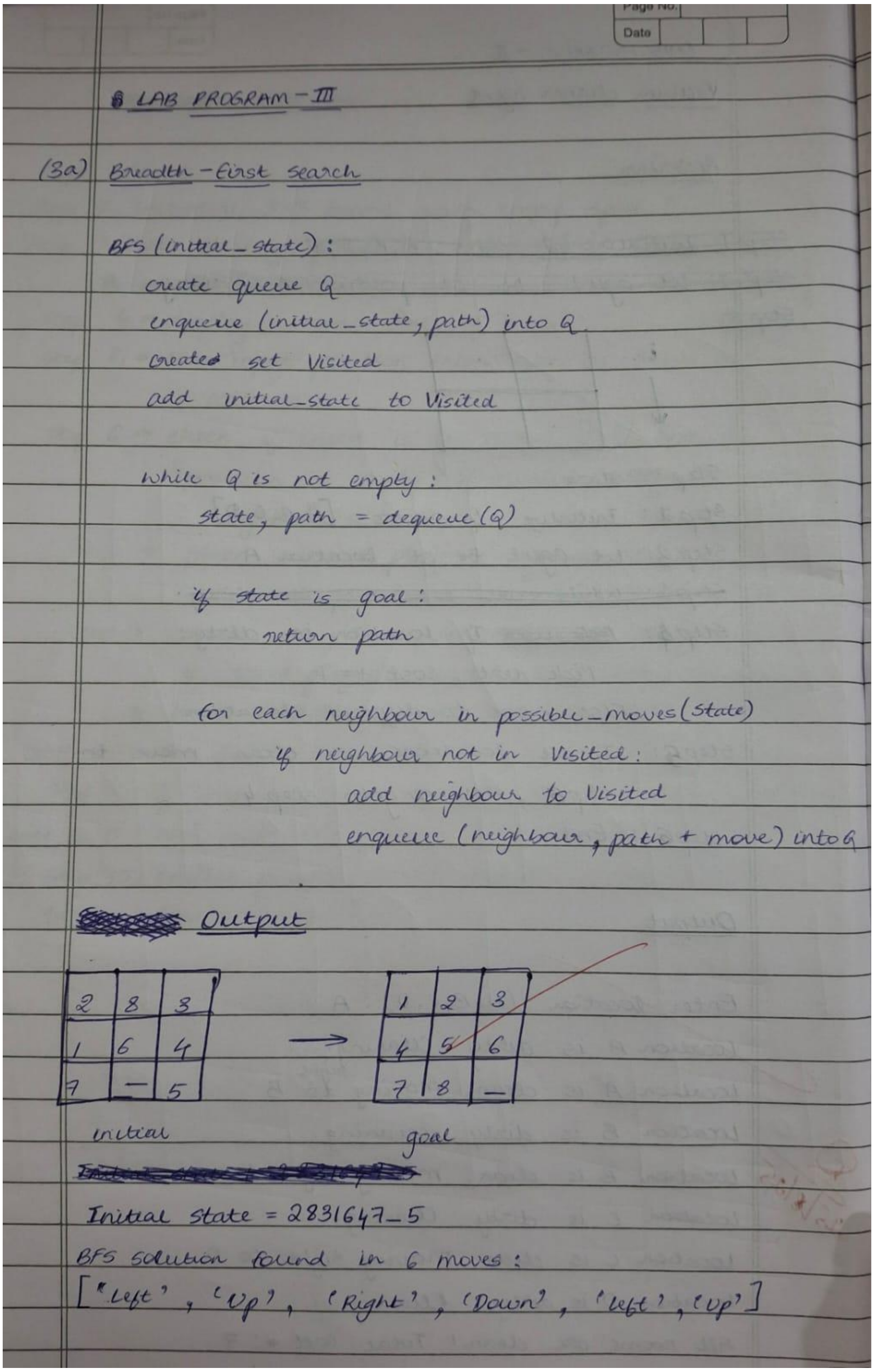
LAB PROGRAM 3

(a) Solve 8 puzzle by Breadth First Search method (non-heuristic approach)

PSEUDOCODE:



LAB PROGRAM - III

(3a) Breadth-First Search

```
BFS (initial_state):
    create queue Q
    enqueue (initial_state, path) into Q
    created set Visited
    add initial_state to Visited

    while Q is not empty:
        state, path = dequeue(Q)

        if state is goal:
            return path

        for each neighbour in possible_moves(state)
            if neighbour not in Visited:
                add neighbour to Visited
                enqueue (neighbour, path + move) into Q
```

Output

2	8	3
1	6	4
7	_	5

initial

→

1	2	3
4	5	6
7	8	_

goal

Initial state = 2831647_5

BFS solution found in 6 moves:

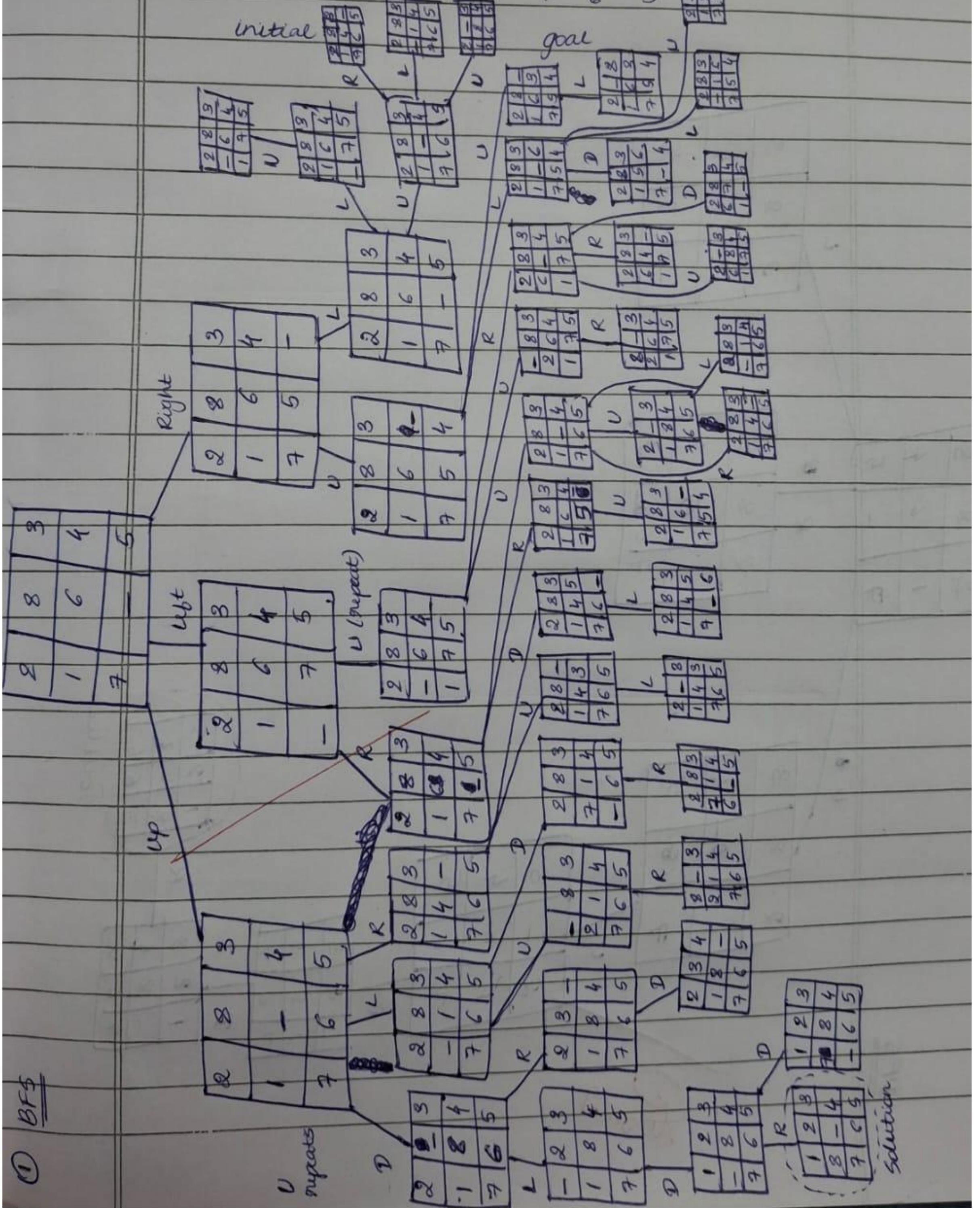
['Left', 'Up', 'Right', 'Down', 'Left', 'Up']

Question

Using BFS solve 8 puzzle without heuristic

2	8	3		1	2	3
1	6	4	→	8	–	4
7	–	5		7	6	5

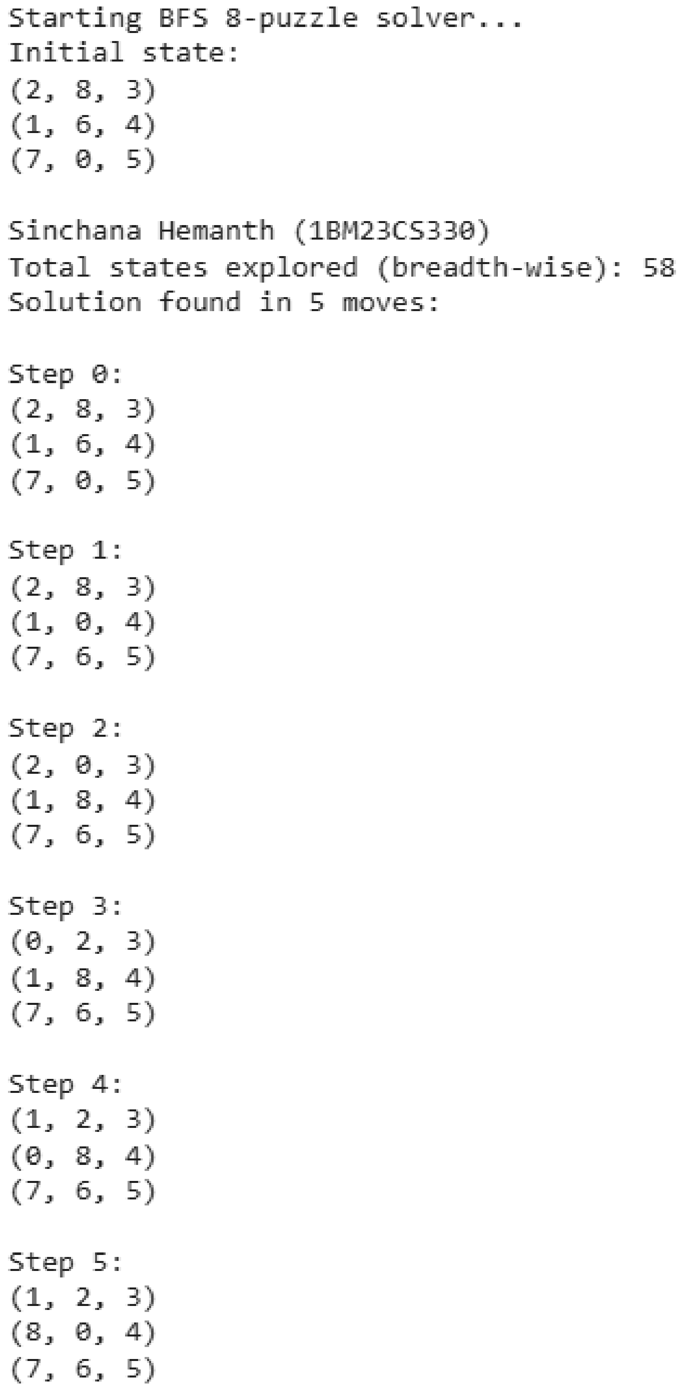
initial goal



CODE:

```
from collections import deque

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

MOVES = {
    'left': -1,
    'right': 1,
    'up': -3,
    'down': 3,
}

def is_valid_move(blank_idx, move):
    if move == 'left' and blank_idx % 3 == 0:
        return False
    if move == 'right' and blank_idx % 3 == 2:
        return False
    if move == 'up' and blank_idx < 3:
        return False
    if move == 'down' and blank_idx > 5:
        return False
    return True

def get_neighbors(state):
    neighbors = []
    blank_idx = state.index(0)
    for move, delta in MOVES.items():
        if is_valid_move(blank_idx, move):
            new_idx = blank_idx + delta
            new_state = list(state)
            new_state[blank_idx], new_state[new_idx] =
new_state[new_idx], new_state[blank_idx]
            neighbors.append(tuple(new_state))
    return neighbors

def bfs(start_state):
    queue = deque([start_state])
    visited = set([start_state])
    parent = {start_state: None}
    explored_count = 0

    while queue:
        current = queue.popleft()
        explored_count += 1

        if current == GOAL_STATE:
            path = []
            while current:
                path.append(current)
                current = parent[current]
            path.reverse()
            print(f"Total states explored (breadth-wise):
{explored_count}")
            return path

        for neighbor in get_neighbors(current):
            if neighbor not in visited:
                visited.add(neighbor)
                parent[neighbor] = current
                queue.append(neighbor)
    print(f"Total states explored (breadth-wise): {explored_count}")
    return None

def print_state(state):
    for i in range(0, 9, 3):
        print(state[i:i+3])
    print()

if __name__ == "__main__":
    start = (2, 8, 3,
             1, 6, 4,
             7, 0, 5)

    print("Starting BFS 8-puzzle solver...\nInitial state:")
    print_state(start)
    print("Sinchana Hemanth (1BM23CS330)")
    solution = bfs(start)

    if solution:
        print(f"Solution found in {len(solution)-1} moves:\n")
        for step_num, state in enumerate(solution):
            print(f"Step {step_num}:")
            print_state(state)
    else:
        print("No solution found.")
```

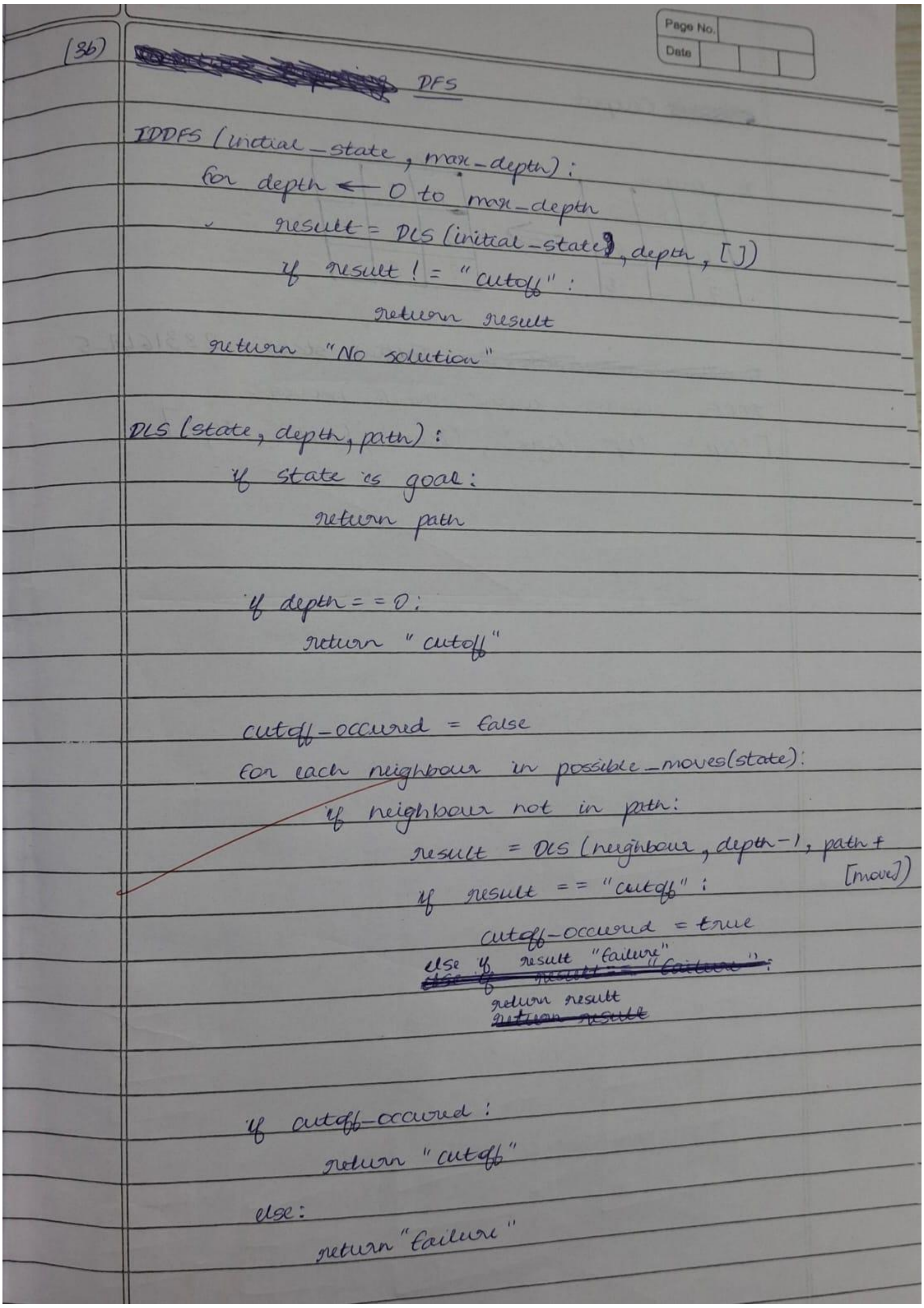
OUTPUT:



```
Starting BFS 8-puzzle solver...
Initial state:
(2, 8, 3)
(1, 6, 4)
(7, 0, 5)

Sinchana Hemanth (1BM23CS330)
Total states explored (breadth-wise): 58
Solution found in 5 moves:

Step 0:
(2, 8, 3)
(1, 6, 4)
(7, 0, 5)

Step 1:
(2, 8, 3)
(1, 0, 4)
(7, 6, 5)

Step 2:
(2, 0, 3)
(1, 8, 4)
(7, 6, 5)

Step 3:
(0, 2, 3)
(1, 8, 4)
(7, 6, 5)

Step 4:
(1, 2, 3)
(0, 8, 4)
(7, 6, 5)

Step 5:
(1, 2, 3)
(8, 0, 4)
(7, 6, 5)
```

(b) Solve 8 puzzle by Depth First Search method (non-heuristic approach)

PSEUDOCODE:

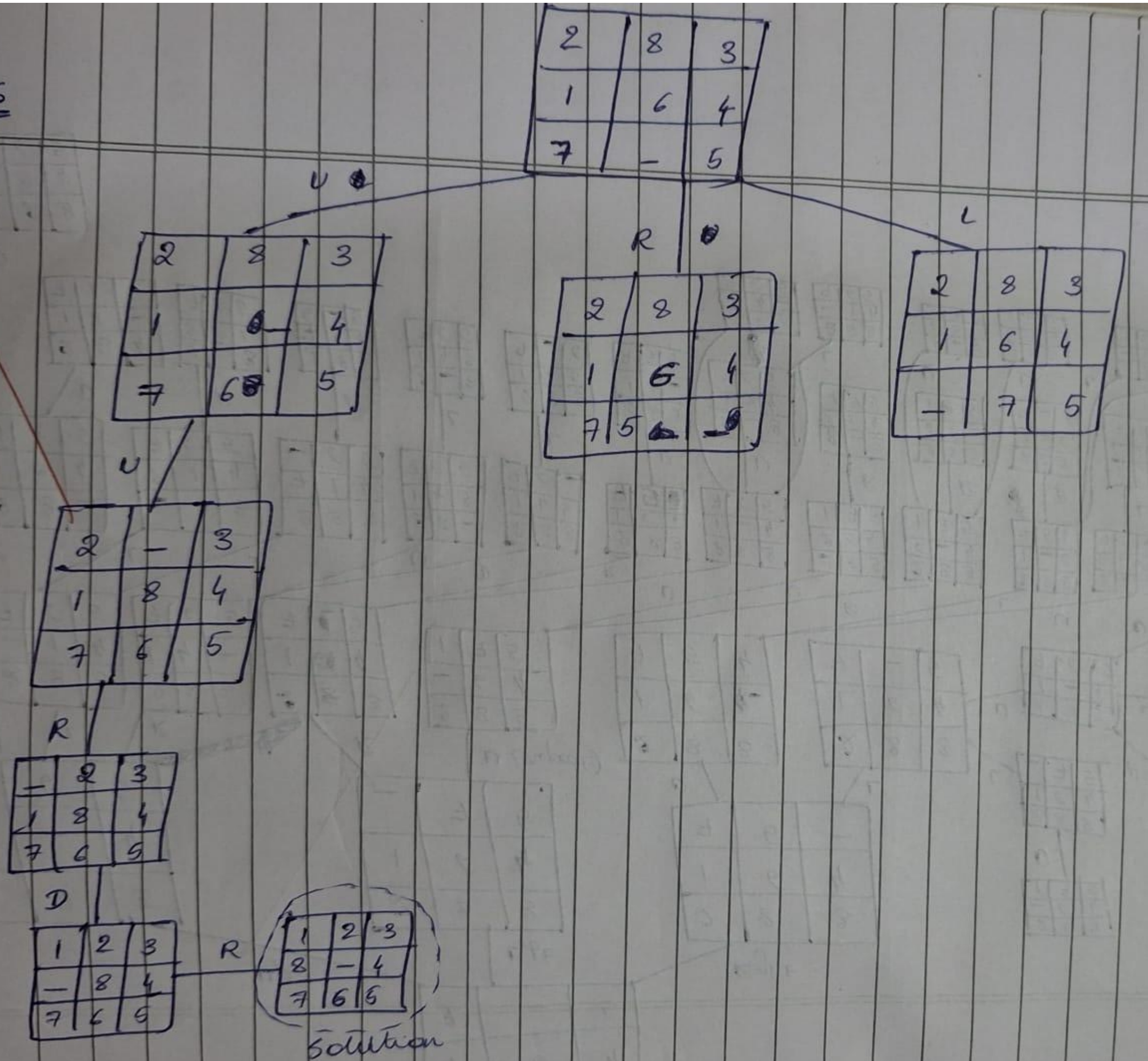


(36) [redacted] DFS

Page No.
Date

```
IDDFS (initial_state, max-depth):
    for depth ← 0 to max-depth
        result = DLS (initial-state, depth, [])
        if result != "cutoff":
            return result
    return "No solution"

DLS (state, depth, path):
    if state is goal:
        return path

    if depth == 0:
        return "cutoff"

    cutoff-occured = false
    for each neighbour in possible_moves(state):
        if neighbour not in path:
            result = DLS (neighbour, depth-1, path + [move])
            if result == "cutoff":
                cutoff-occured = true
            else if result "failure"
            ~~else if result == "failure":~~
                return result
                ~~return result~~

    if cutoff-occured:
        return "cutoff"
    else:
        return "failure"
```

DFS

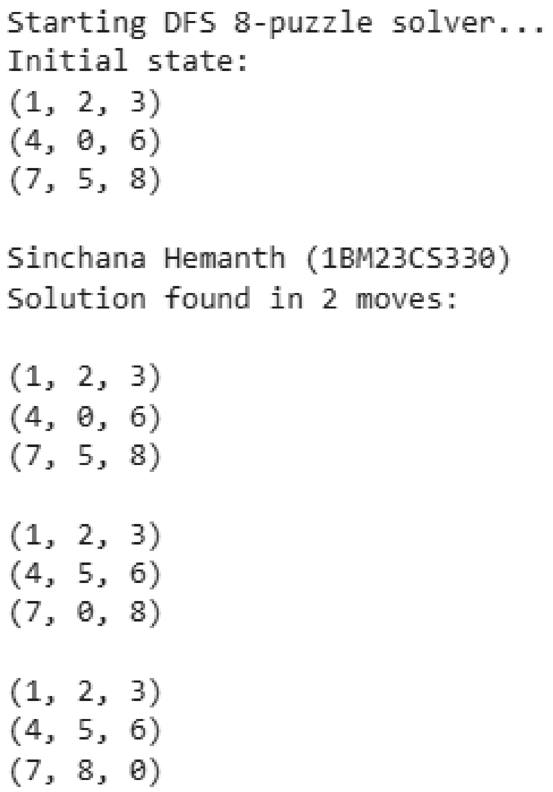

2 8 3
1 6 4
7 — 5
U
R
L
2 8 3
1 — 4
7 6 5
2 8 3
1 6 4
7 5 —
2 8 3
1 6 4
— 7 5
U
2 — 3
1 8 4
7 6 5
R
— 2 3
1 8 4
7 6 5
D
1 2 3
— 8 4
7 6 5
R
1 2 3
8 — 4
7 6 5
Solution

01.09

CODE:

```
GOAL_STATE = (1, 2, 3, 4, 5, 6, 7, 8, 0)
MOVES = {
    'left': -1,
    'right': 1,
    'up': -3,
    'down': 3,
}
def is_valid_move(blank_idx, move):
    if move == 'left' and blank_idx % 3 == 0:
        return False
    if move == 'right' and blank_idx % 3 == 2:
        return False
    if move == 'up' and blank_idx < 3:
        return False
    if move == 'down' and blank_idx > 5:
        return False
    return True
def get_neighbors(state):
    neighbors = []
    blank_idx = state.index(0)
    for move, delta in MOVES.items():
        if is_valid_move(blank_idx, move):
            new_idx = blank_idx + delta
            new_state = list(state)
            new_state[blank_idx], new_state[new_idx] =
new_state[new_idx], new_state[blank_idx]
            neighbors.append(tuple(new_state))
    return neighbors
def dfs(start_state, max_depth=50):
    stack = [(start_state, 0)]
    visited = set([start_state])
    parent = {start_state: None}
    while stack:
        current, depth = stack.pop()
        if current == GOAL_STATE:
            path = []
            while current:
                path.append(current)
                current = parent[current]
            path.reverse()
            return path

        if depth < max_depth:
            for neighbor in get_neighbors(current):
                if neighbor not in visited:
                    visited.add(neighbor)
                    parent[neighbor] = current
                    stack.append((neighbor, depth + 1))
    return None
def print_state(state):
    for i in range(0, 9, 3):
        print(state[i:i+3])
    print()
if __name__ == "__main__":
    start = (1, 2, 3,
             4, 0, 6,
             7, 5, 8)
    print("Starting DFS 8-puzzle solver...\nInitial state:")
    print_state(start)
    print("Sinchana Hemanth (1BM23CS330)")
    solution = dfs(start, max_depth=20)

    if solution:
        print(f"Solution found in {len(solution)-1} moves:\n")
        for step in solution:
            print_state(step)
    else:
        print("No solution found or max depth exceeded.")
```

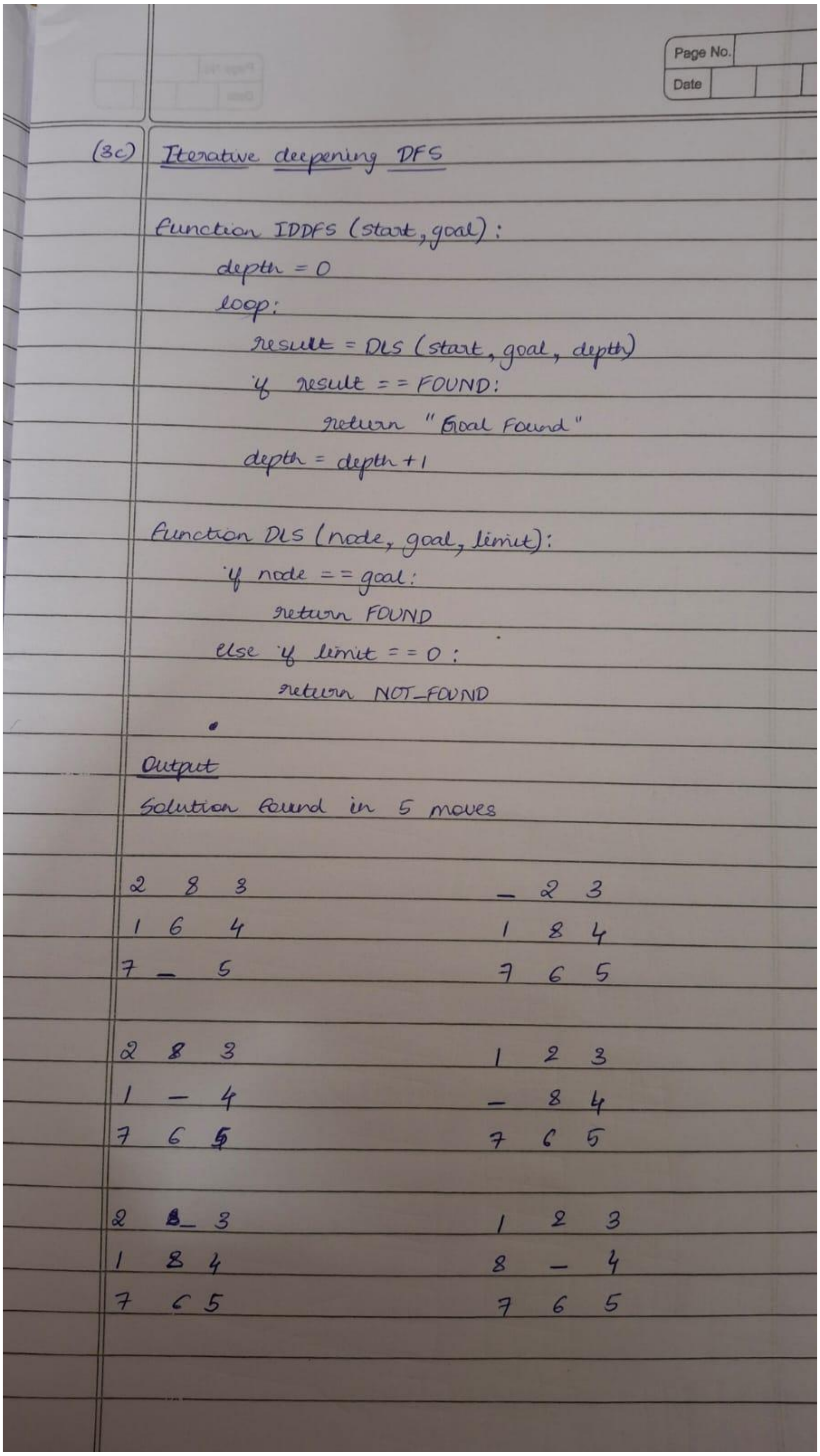
OUTPUT:



```
Starting DFS 8-puzzle solver...
Initial state:
(1, 2, 3)
(4, 0, 6)
(7, 5, 8)

Sinchana Hemanth (1BM23CS330)
Solution found in 2 moves:

(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)
```

(c) Solve 8 puzzles by Iterative Deepening Depth First Search (IDDFS)

PSEUDOCODE:



Page No.
Date

(3c) Iterative deepening DFS

```
function IDDFS (start, goal):
    depth = 0
    loop:
        result = DLS (start, goal, depth)
        if result == FOUND:
            return "Goal Found"
        depth = depth + 1

function DLS (node, goal, limit):
    if node == goal:
        return FOUND
    else if limit == 0:
        return NOT_FOUND
```

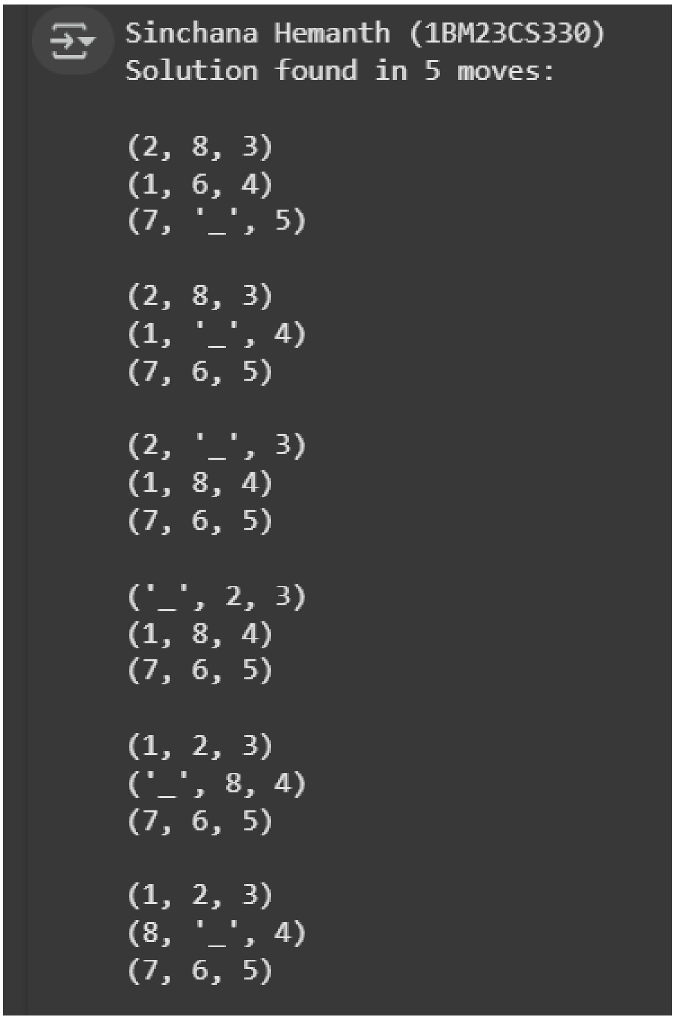
Output

```
Solution found in 5 moves

2 8 3                    _ 2 3
1 6 4                    1 8 4
7 _ 5                    7 6 5

2 8 3                    1 2 3
1 _ 4                    _ 8 4
7 6 5                    7 6 5

2 _ 3                    1 2 3
1 8 4                    8 _ 4
7 6 5                    7 6 5
```

CODE:

```
from collections import deque

N = 3

moves = [(-1,0),(1,0),(0,-1),(0,1)]

def find_blank(state):
    idx = state.index("_")
    return divmod(idx, N)

def swap(state, i1, j1, i2, j2):
    s = list(state)
    idx1, idx2 = i1*N+j1, i2*N+j2
    s[idx1], s[idx2] = s[idx2], s[idx1]
    return tuple(s)

def expand(state):
    x, y = find_blank(state)
    children = []
    for dx, dy in moves:
        nx, ny = x+dx, y+dy
        if 0 <= nx < N and 0 <= ny < N:
            children.append(swap(state, x, y, nx, ny))
    return children

def dls(state, goal, limit, path, visited):
    if state == goal:
        return path

    if limit == 0:
        return None

    visited.add(state)
    for child in expand(state):
        if child not in visited:
            result = dls(child, goal, limit-1, path+[child], visited)
            if result is not None:
                return result
    visited.remove(state)
    return None

def iddfs(start, goal, max_depth=20):
    for depth in range(max_depth):
        visited = set()
        result = dls(start, goal, depth, [start], visited)
        if result is not None:
            return result
    return None

print("Sinchana Hemanth (1BM23CS330)")
initial = (2,8,3,1,6,4,7,"_",5)
goal    = (1,2,3,8,"_",4,7,6,5)

solution = iddfs(initial, goal, max_depth=30)

if solution:
    print("Solution found in", len(solution)-1, "moves:\n")
    for step in solution:
        for i in range(0, 9, 3):
            print(step[i:i+3])
        print()
else:
    print("No solution found within depth limit")
```

OUTPUT:



```
Sinchana Hemanth (1BM23CS330)
Solution found in 5 moves:

(2, 8, 3)
(1, 6, 4)
(7, '_', 5)

(2, 8, 3)
(1, '_', 4)
(7, 6, 5)

(2, '_', 3)
(1, 8, 4)
(7, 6, 5)

('_', 2, 3)
(1, 8, 4)
(7, 6, 5)

(1, 2, 3)
('_', 8, 4)
(7, 6, 5)

(1, 2, 3)
(8, '_', 4)
(7, 6, 5)
```