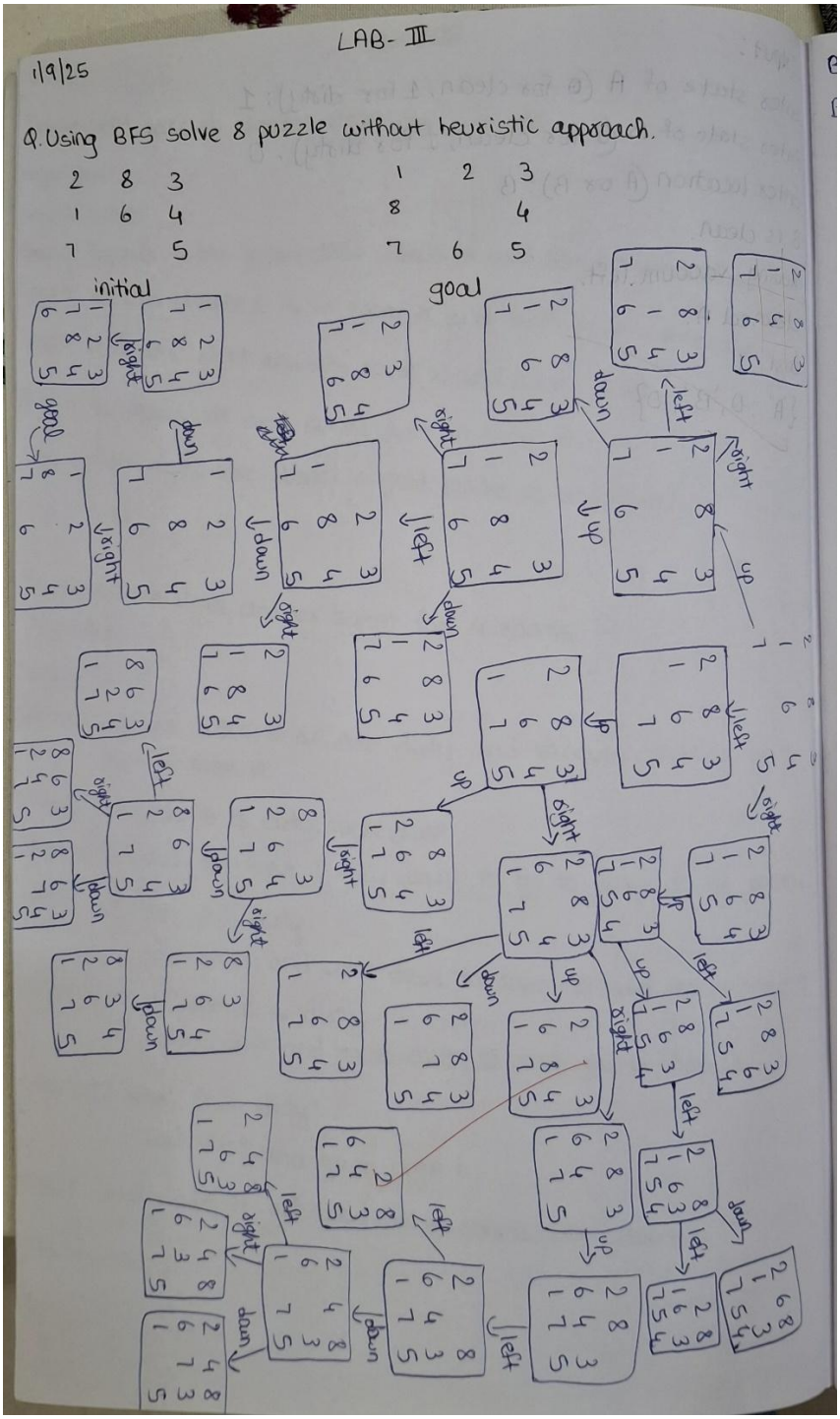
Lab 2

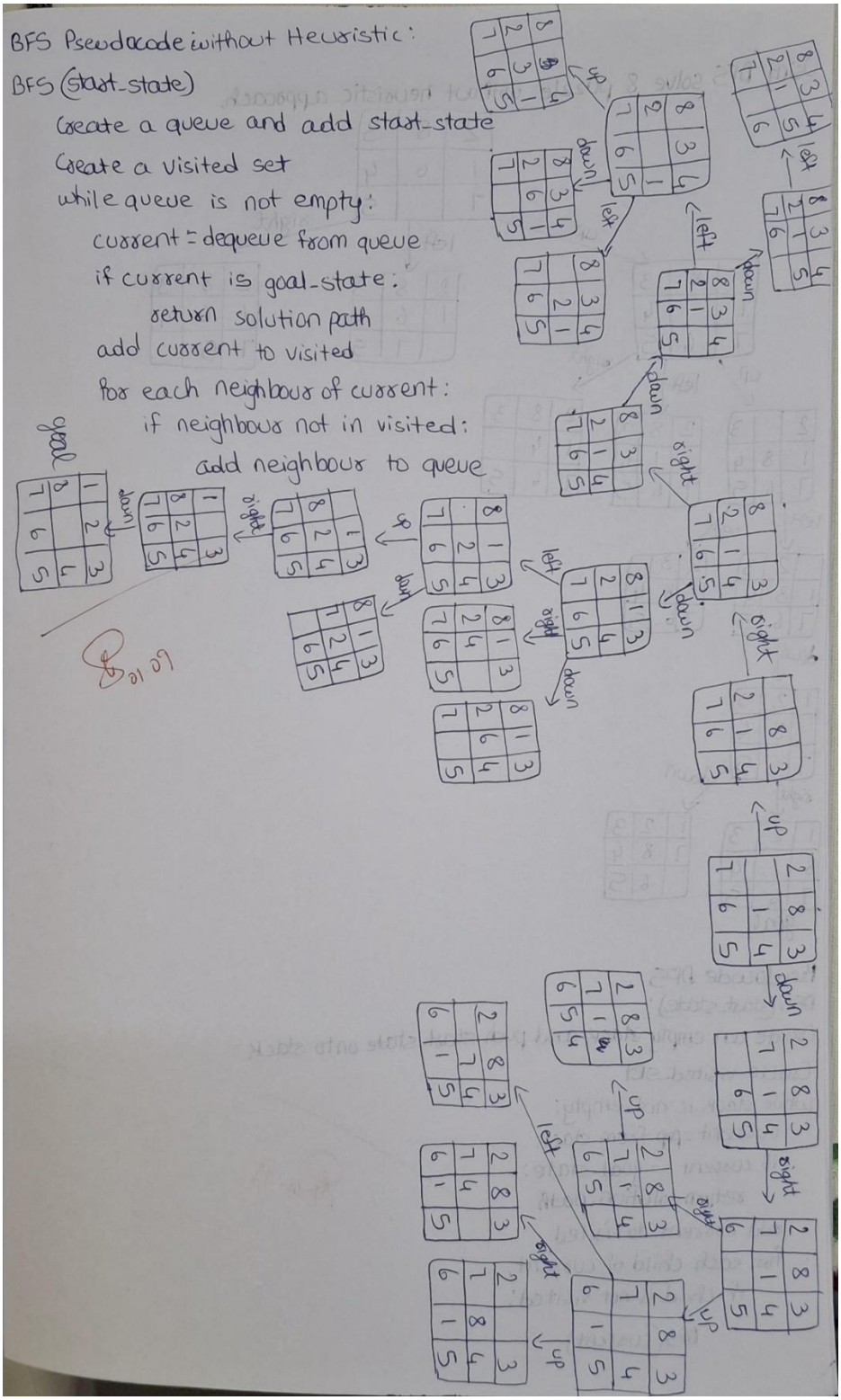
Using BFS solve 8 puzzle without heuristic approach.

Algorithm:



BFS Pseudocode without Heuristic:

```
BFS (start_state)
    Create a queue and add start_state
    Create a visited set
    while queue is not empty:
        current = dequeue from queue
        if current is goal_state:
            return solution path
        add current to visited
        for each neighbour of current:
            if neighbour not in visited:
                add neighbour to queue
```



Code:

```
from collections import deque


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


def bfs(start, goal):
    queue = deque([(start, [])])
    visited = set([start])
    while queue:
        state, path = queue.popleft()
        if state == goal:
            return path + [state]
        zero = state.index(0)
        moves = []
        if zero % 3 > 0:
            moves.append(zero - 1)
        if zero % 3 < 2:
            moves.append(zero + 1)
        if zero // 3 > 0:
            moves.append(zero - 3)
        if zero // 3 < 2:
            moves.append(zero + 3)
        for move in moves:
            new_state = list(state)
```

```
            new_state[zero], new_state[move] = new_state[move], new_state[zero]
            new_state = tuple(new_state)
            if new_state not in visited:
                visited.add(new_state)
                queue.append((new_state, path + [state]))
    return None

def input_state(prompt):
    s = input(prompt).strip().split()
    return tuple(map(int, s))

start = input_state("Enter initial state (9 numbers with 0 for blank): ")
goal = input_state("Enter goal state (9 numbers with 0 for blank): ")

result = bfs(start, goal)
if result:
    for step in result:
        print_state(step)
else:
    print("No solution found")
```

Output:

1BM23CS317_AI_Lab2a.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

27s

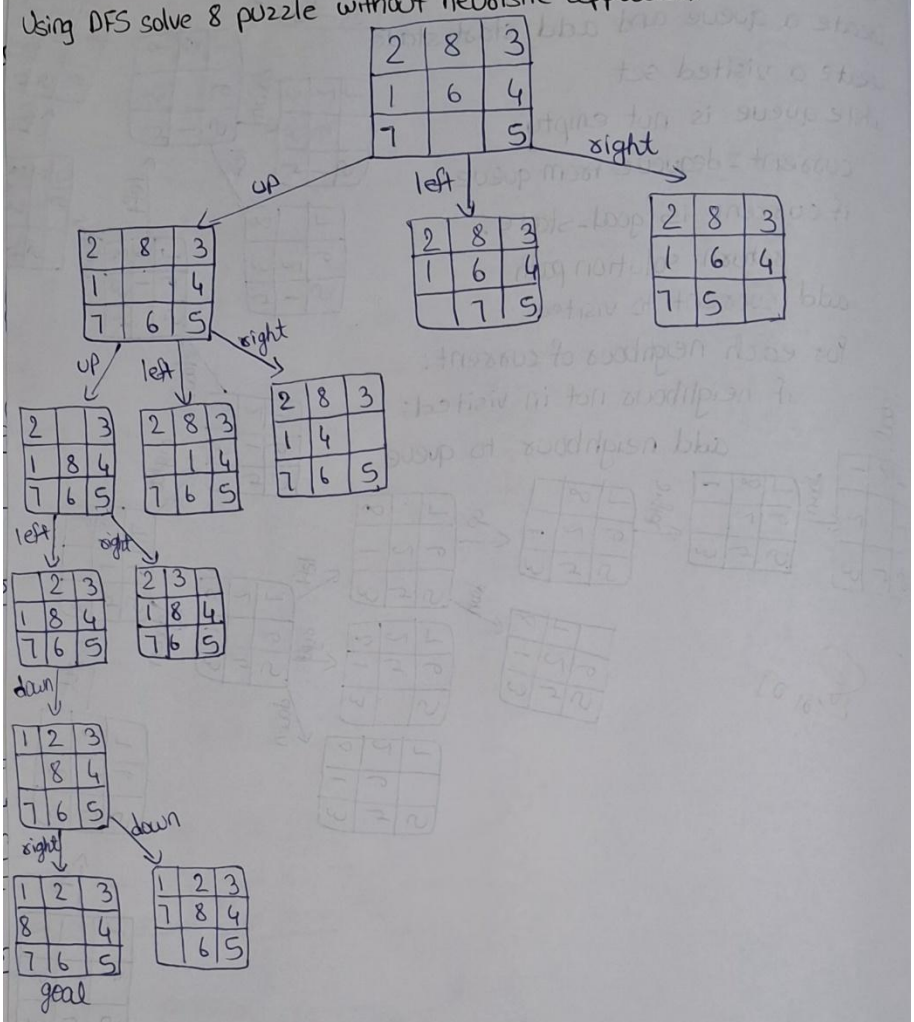
```
                queue.append((new_state, path + [state]))
    return None

def input_state(prompt):
    s = input(prompt).strip().split()
    return tuple(map(int, s))

start = input_state("Enter initial state (9 numbers with 0 for blank): ")
goal = input_state("Enter goal state (9 numbers with 0 for blank): ")

result = bfs(start, goal)
if result:
    for step in result:
        print_state(step)
else:
    print("No solution found")
```

```
Enter initial state (9 numbers with 0 for blank): 2 8 3 1 6 4 7 0 5
Enter goal state (9 numbers with 0 for blank): 1 2 3 8 0 4 7 6 5
(2, 8, 3)
(1, 6, 4)
(7, 0, 5)

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

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

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

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

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

Using DFS solve 8 puzzle without heuristic approach.

Algorithm:

Using DFS solve 8 puzzle without heuristic approach.



Pseudocode DFS

```
DFS (start_state):
Create an empty stack and push start_state onto stack
Create visited_set
while stack is not empty:
    current = pop from stack
    if current == goal_state:
        return solution_path
    add current to visited
    for each child of current:
        if child is not visited:
            DFS(current)
```

01.09

Code:

```
def dfs_limited(state, goal, depth, path, visited):
    if state == goal:
        return path + [state]
    if depth == 0:
        return None

    visited.add(state)
    zero = state.index(0)
    moves = []
    # Determine valid moves
    if zero % 3 > 0:
        moves.append(zero - 1)
    if zero % 3 < 2:
        moves.append(zero + 1)
    if zero // 3 > 0:
        moves.append(zero - 3)
    if zero // 3 < 2:
        moves.append(zero + 3)

    for move in moves:
        new_state = list(state)
        new_state[zero], new_state[move] = new_state[move], new_state[zero]
        new_state = tuple(new_state)

        if new_state not in visited:
            result = dfs_limited(new_state, goal, depth - 1, path + [state], visited)
```

```
            if result:
                return result

    visited.remove(state)
    return None

def input_state(prompt):
    s = input(prompt).strip().split()
    return tuple(map(int, s))

def print_state(state):
    for i in range(0, 9, 3):
        print(state[i], state[i + 1], state[i + 2])
    print()
# Main Execution
start = input_state("Enter initial state (9 numbers with 0 for blank): ")
goal = input_state("Enter goal state (9 numbers with 0 for blank): ")

max_depth = 10
result = dfs_limited(start, goal, max_depth, [], set())

if result:
    print("\nSolution path:")
    for step in result:
        print_state(step)
else:
    print("No solution found within depth limit")
```

Output:

1BM23CS317_AI_Lab2b.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

```
Enter initial state (9 numbers with 0 for blank): 2 8 3 1 6 4 7 0 5
Enter goal state (9 numbers with 0 for blank): 1 2 3 8 0 4 7 6 5

Solution path:
2 8 3
1 6 4
7 0 5

2 8 3
1 0 4
7 6 5

2 8 3
0 1 4
7 6 5

0 8 3
2 1 4
7 6 5

8 0 3
2 1 4
7 6 5

8 1 3
2 0 4
7 6 5

8 1 3
0 2 4
7 6 5

0 1 3
8 2 4
7 6 5

1 0 3
8 2 4
7 6 5

1 2 3
8 0 4
7 6 5
```