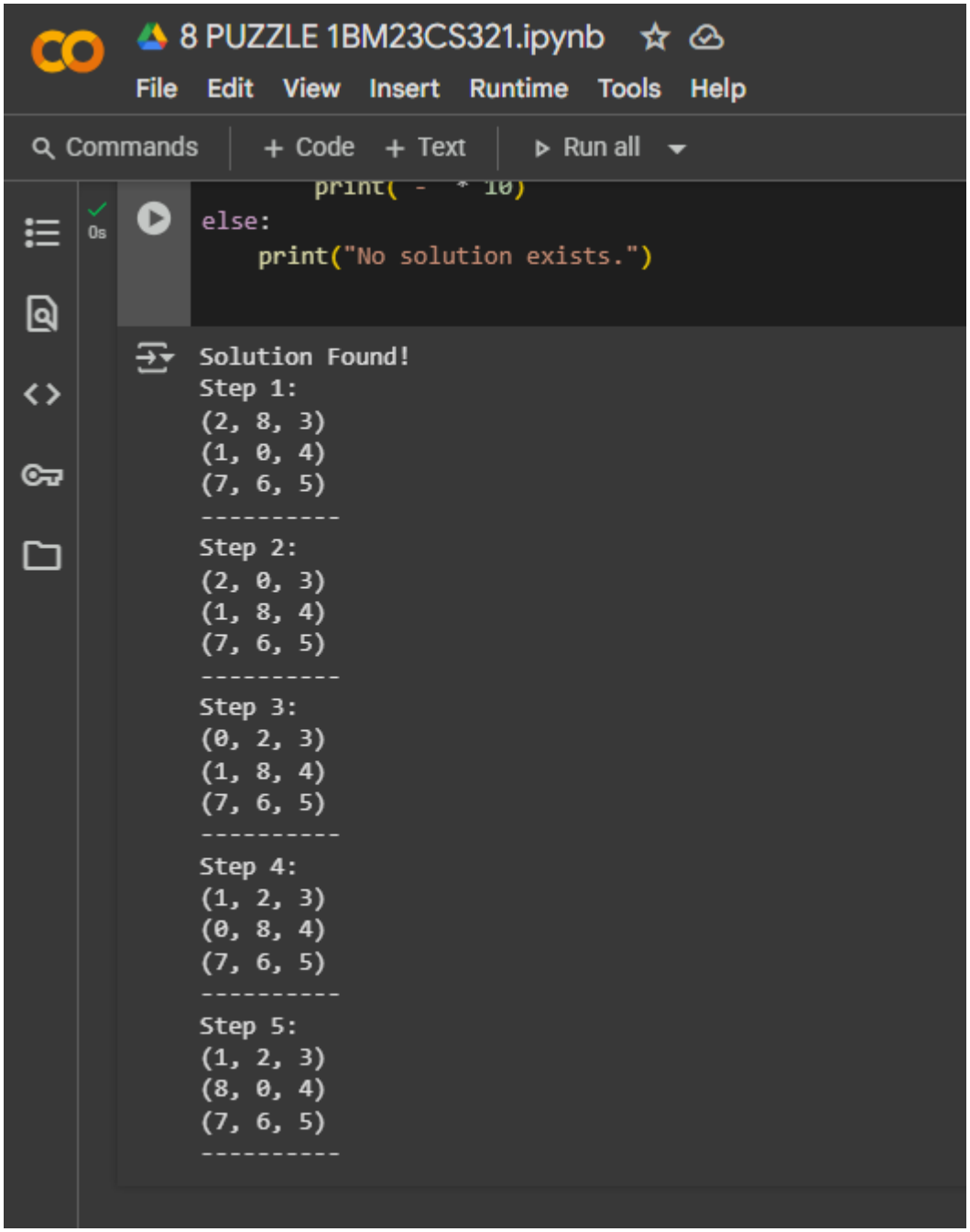
BFS TRAVERSAL

```
from collections import deque

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

def get_neighbors(state):
    neighbors = []
    blank_row, blank_col = find_blank(state)
    moves = [(0, 1), (0, -1), (1, 0), (-1, 0)]

    for move_row, move_col in moves:
        new_row, new_col = blank_row + move_row, blank_col + move_col

        if 0 <= new_row < 3 and 0 <= new_col < 3:
            new_state = [list(row) for row in state]
            new_state[blank_row][blank_col], new_state[new_row][new_col] = \
                new_state[new_row][new_col], new_state[blank_row][blank_col]
            neighbors.append(tuple(tuple(row) for row in new_state))

    return neighbors

def bfs(initial_state, goal_state):
    queue = deque([(initial_state, [])])
    visited = set([initial_state])

    while queue:
        current_state, path = queue.popleft()

        if current_state == goal_state:
            return path

        for neighbor in get_neighbors(current_state):
            if neighbor not in visited:
                visited.add(neighbor)
                queue.append((neighbor, path + [neighbor]))

    return None

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

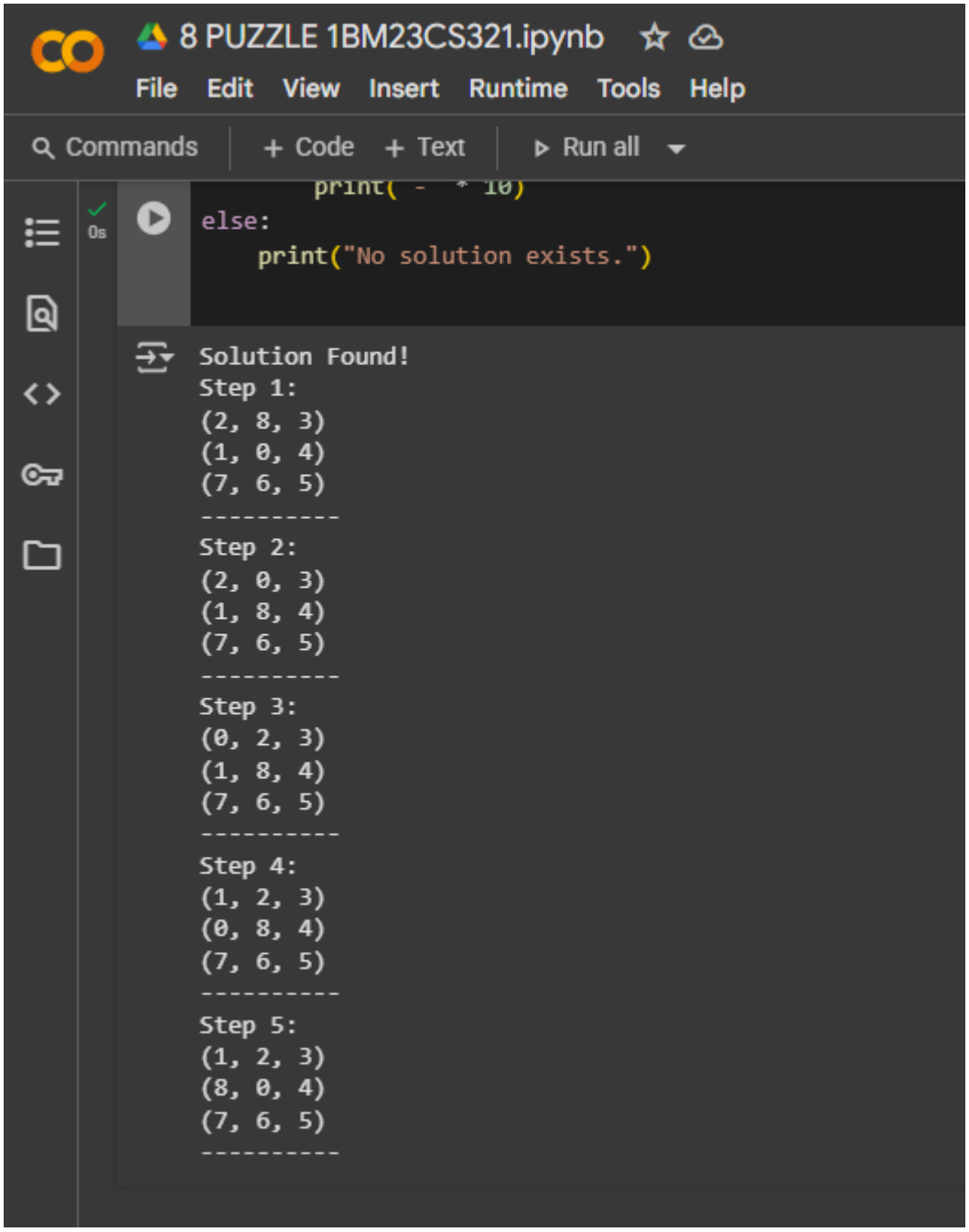
```
)

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

solution_path = bfs(initial_state, goal_state)

if solution_path:
    print("Solution Found!")
    for i, state in enumerate(solution_path):
        print(f"Step {i+1}:")
        for row in state:
            print(row)
        print("-" * 10)
else:
    print("No solution exists.")
```



DFS TRAVERSAL

```
from collections import deque

def find_blank(state):
    """Finds the position of the blank tile (0)."""
    for i in range(3):
        for j in range(3):
            if state[i][j] == 0:
                return (i, j)
```

```
def get_neighbors(state):
    """Generates all possible next states from the current state."""
    neighbors = []
    blank_row, blank_col = find_blank(state)
    moves = [(0, 1), (0, -1), (1, 0), (-1, 0)]  # Right, Left, Down, Up

    for move_row, move_col in moves:
        new_row, new_col = blank_row + move_row, blank_col + move_col

        if 0 <= new_row < 3 and 0_state = (
    (1, 2, 3),
    (4, 0, 5),
    (6, 7, 8)
)

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

solution_path = dfs(initial_state, goal_state)

if solution_path:
    print("Solution Found!")
    for i, state in enumerate(solution_path):
        print(f"Step {i+1}:")
        for row in state:
            print(row)
        print("-" * 10)
else:
    print("No solution exists.")
```


8 PUZZLE 1BM23CS321.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
print(- * 10)
else:
print("No solution exists.")
0s
Solution Found!
Step 1:
(2, 8, 3)
(1, 0, 4)
(7, 6, 5)

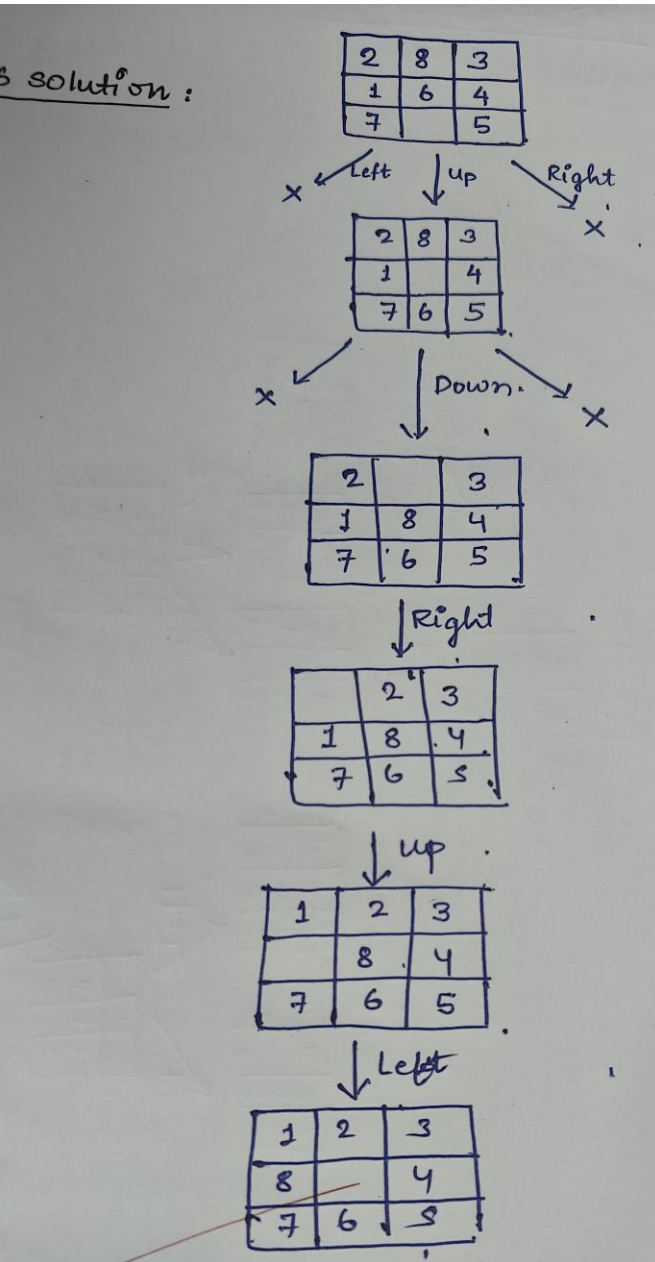
Step 2:
(2, 0, 3)
(1, 8, 4)
(7, 6, 5)

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)

DFS solution :

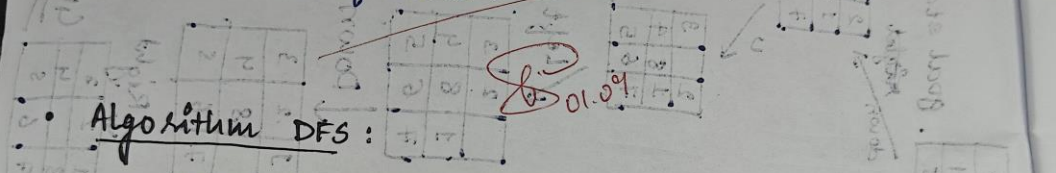


01.09

LAB-III : 8 PUZZLE PROBLEM

using BFS solve 8 puzzle without heuristic :

- **Algorithm BFS:**
  - * Write the correctly defined goal state
  - * Take input in string form.
  - * BFS (start state)
  - * Move accordingly to valid moves and choose least misplaced tile for next BFS.
  - * Push each h value, state, path chosen into queue.
  - * Choose the least visited branch and move ahead.
  - * If start state = goal state, calculate the number of visits.



- **Algorithm DFS:**
  - * start and get the goal state.
  - * Take input in string form.
  - * DFS (start state)
  - * Move accordingly to valid moves, and choose best possible tile.
  - * Add each state to recursion
  - * If start state = final goal state, return path.