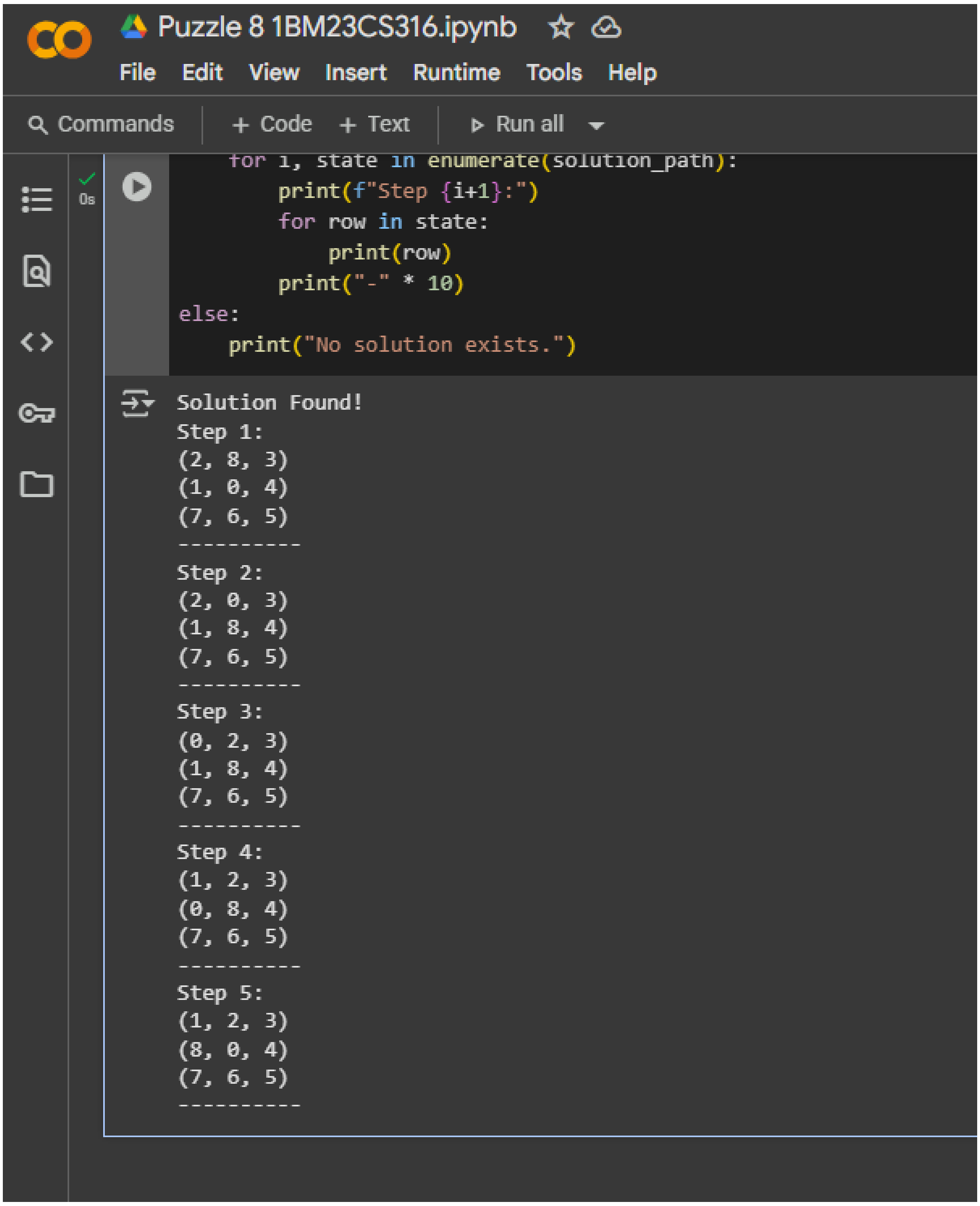
```python
from collections import deque
def find_blank(state):
    for i in range(3):
        for j in range(3):
            if state[i][j] == 0:
                return (i, j)
def get_neighbors(state):
    neighbors = []
    blank_row, blank_col = find_blank(state)
    moves = [(0, 1), (0, -1), (1, 0), (-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 is not None:
    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.")
```


Puzzle 8 1BM23CS316.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
0s

```
    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.")
```

```
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)
----------
```

Algorithm : BFS

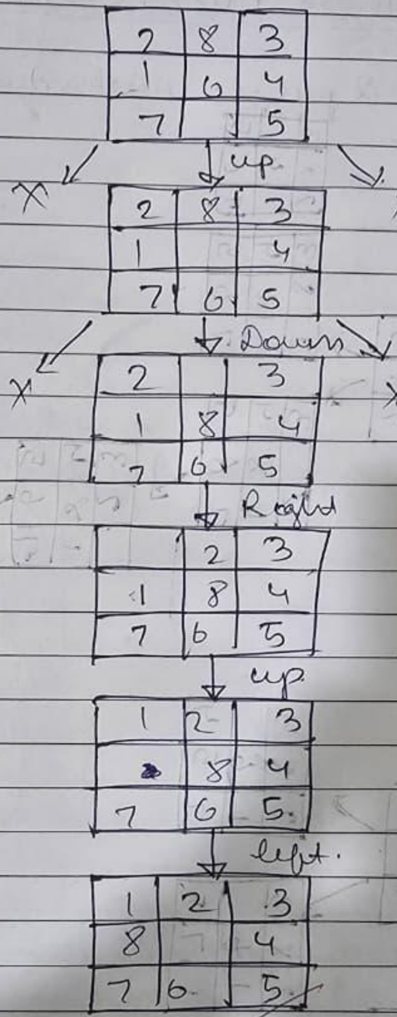
1.) Start - write goal state
2.) Take input in string form
3.) BFS (start state)
4.) Move according to valid moves and choose least misplaced tile for next BFS
5.) Push each h value, state, path chosen into queue.
6.) Choose the least valid visited branch and move ahead.
7.) If ~~state st~~ start state = goal state, calculate number of visits.

Algorithm :- DFS.

1.) Start and get the goal state.
2.) Take the input in string form.
3.) DFS (start state)
4.) Move accordingly to valid moves and choose least positive tile
5.) Add each state to recursion
6.) If start state = goal state, return path.

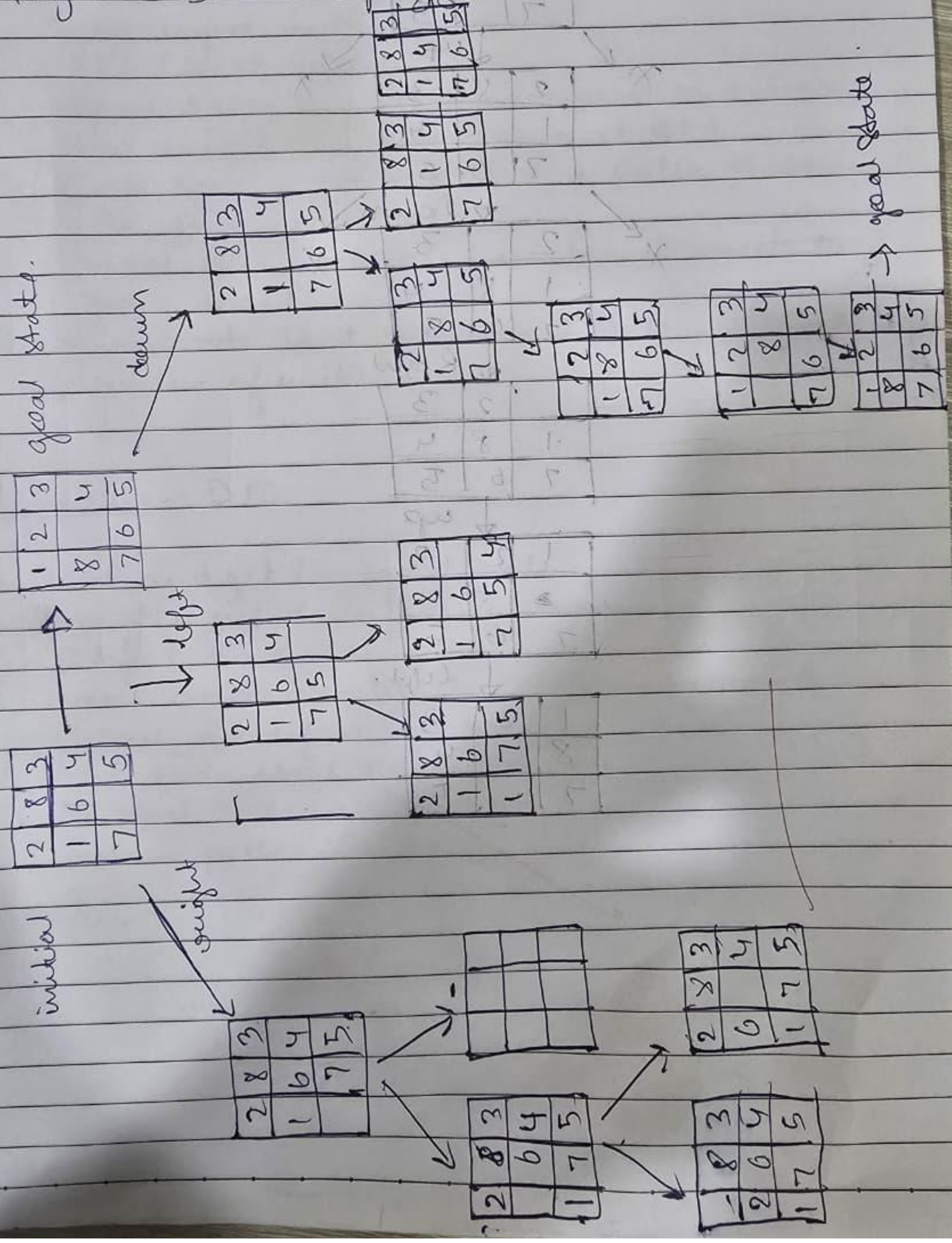
01.09

DFS :


2 8 3 / 1 6 4 / 7 _ 5
X
up
X
2 8 3 / 1 _ 4 / 7 6 5
X
Down
X
2 _ 3 / 1 8 4 / 7 6 5
Right
_ 2 3 / 1 8 4 / 7 6 5
up
1 2 3 / _ 8 4 / 7 6 5
left
1 2 3 / 8 _ 4 / 7 6 5

LAB - III 8 PUZZLE PROBLEM

Using BFS solve 8 puzzle without heuristic.


initial
goal state
down
left
right
goal state