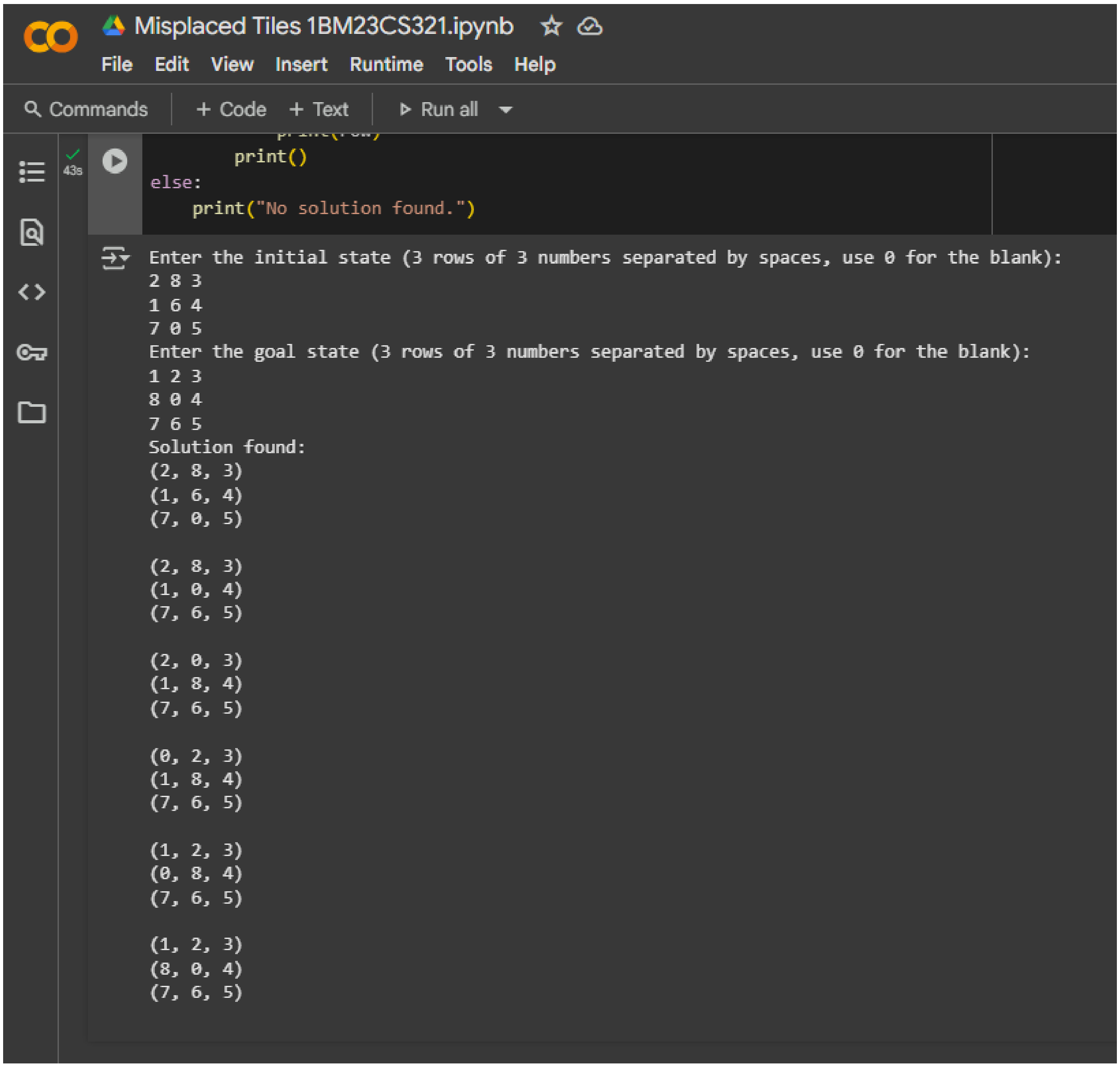
MISPLACED TILES

```
def misplaced_tiles(state, goal):
    count = 0
    for i in range(3):
        for j in range(3):
            if state[i][j] != goal[i][j] and state[i][j] != 0:
                count += 1
    return count

def get_neighbors(state):
    neighbors = []
    for i in range(3):
        for j in range(3):
            if state[i][j] == 0:
                x, y = i, j
                break
        else:
            continue
        break

    moves = [(0, 1), (0, -1), (1, 0), (-1, 0)]
    for dx, dy in moves:
        nx, ny = x + dx, y + dy
        if 0 <= nx < 3 and 0 <= ny < 3:
            new_state = [list(row) for row in state]
            new_state[x][y], new_state[nx][ny] = new_state[nx][ny], new_state[x][y]
            neighbors.append(tuple(tuple(row) for row in new_state))
    return neighbors

def astar_search(initial, goal):
    frontier = [(misplaced_tiles(initial, goal), 0, initial)]
    explored = set()
    parent = {}
    cost = {initial: 0}

    while frontier:
```

```
        f, g, current = heapq.heappop(frontier)

        if current == goal:
            path = []
            while current in parent:
                path.append(current)
                current = parent[current]
            path.append(initial)
            return path[::-1]

        explored.add(current)

        for neighbor in get_neighbors(current):
            new_cost = cost[current] + 1
            if neighbor not in cost or new_cost < cost[neighbor]:
                cost[neighbor] = new_cost
                priority = new_cost + misplaced_tiles(neighbor, goal)
                heapq.heappush(frontier, (priority, new_cost, neighbor))
                parent[neighbor] = current
    return None

def get_state_input(prompt):
    print(prompt)
    state = []
    for _ in range(3):
        row = list(map(int, input().split()))
        state.append(row)
    return tuple(tuple(row) for row in state)

initial_state = get_state_input("Enter the initial state (3 rows of 3 numbers separated by spaces, use 0 for the blank):")
goal_state = get_state_input("Enter the goal state (3 rows of 3 numbers separated by spaces, use 0 for the blank):")

path = astar_search(initial_state, goal_state)

if path:
    print("Solution found:")
```

```
    for step in path:
        for row in step:
            print(row)
        print()
else:
    print("No solution found.")
```



MANHATTAN DISTANCE

```
import heapq

def manhattan_distance(state, goal):
    distance = 0
    for i in range(3):
        for j in range(3):
            if state[i][j] != 0:
                value = state[i][j]
                # Find the position of the value in the goal state
                for gi in range(3):
                    for gj in range(3):
                        if goal[gi][gj] == value:
                            goal_pos = (gi, gj)
                            break
                    else:
                        continue
                    break
                distance += abs(i - goal_pos[0]) + abs(j - goal_pos[1])
    return distance

def get_neighbors(state):
    neighbors = []
    for i in range(3):
        for j in range(3):
            if state[i][j] == 0:
                x, y = i, j
                break
        else:
            continue
        break

    moves = [(0, 1), (0, -1), (1, 0), (-1, 0)]
    for dx, dy in moves:
        nx, ny = x + dx, y + dy
        if 0 <= nx < 3 and 0 <= ny < 3:
```

```
            new_state = [list(row) for row in state]
            new_state[x][y], new_state[nx][ny] = new_state[nx][ny], new_state[x][y]
            neighbors.append(tuple(tuple(row) for row in new_state))
    return neighbors

def astar_search_manhattan(initial, goal):
    frontier = [(manhattan_distance(initial, goal), 0, initial)]
    explored = set()
    parent = {}
    cost = {initial: 0}

    while frontier:
        f, g, current = heapq.heappop(frontier)

        if current == goal:
            path = []
            while current in parent:
                path.append(current)
                current = parent[current]
            path.append(initial)
            return path[::-1]

        explored.add(current)

        for neighbor in get_neighbors(current):
            new_cost = cost[current] + 1
            if neighbor not in cost or new_cost < cost[neighbor]:
                cost[neighbor] = new_cost
                priority = new_cost + manhattan_distance(neighbor, goal)
                heapq.heappush(frontier, (priority, new_cost, neighbor))
                parent[neighbor] = current
    return None

def get_state_input(prompt):
    print(prompt)
    state = []
    for _ in range(3):
        row = list(map(int, input().split()))
```

```
        state.append(row)
    return tuple(tuple(row) for row in state)

initial_state_m = get_state_input("Enter the initial state for Manhattan distance (3 rows of 3
numbers separated by spaces, use 0 for the blank):")
goal_state_m = get_state_input("Enter the goal state for Manhattan distance (3 rows of 3
numbers separated by spaces, use 0 for the blank):")

path_m = astar_search_manhattan(initial_state_m, goal_state_m)

if path_m:
    print("Solution found using Manhattan distance:")
    for step in path_m:
        for row in step:
            print(row)
        print()
else:
    print("No solution found using Manhattan distance.")
```

Manhattan Distance 1BM23CS321.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

```python
if path_m:
    print("Solution found using Manhattan distance:")
    for step in path_m:
        for row in step:
            print(row)
        print()
else:
    print("No solution found using Manhattan distance.")
```

```
Enter the initial state for Manhattan distance (3 rows of 3 numbers separated by spaces, use 0 for the blank):
2 8 3
1 6 4
7 0 5
Enter the goal state for Manhattan distance (3 rows of 3 numbers separated by spaces, use 0 for the blank):
1 2 3
8 0 4
7 6 5
Solution found using Manhattan distance:
(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)
```

8-9-25

LAB IV - A* ALGORITHM

1

Misplaced Tiles

Manhattan Distance

2	8	3
1	6	4
7		5

I

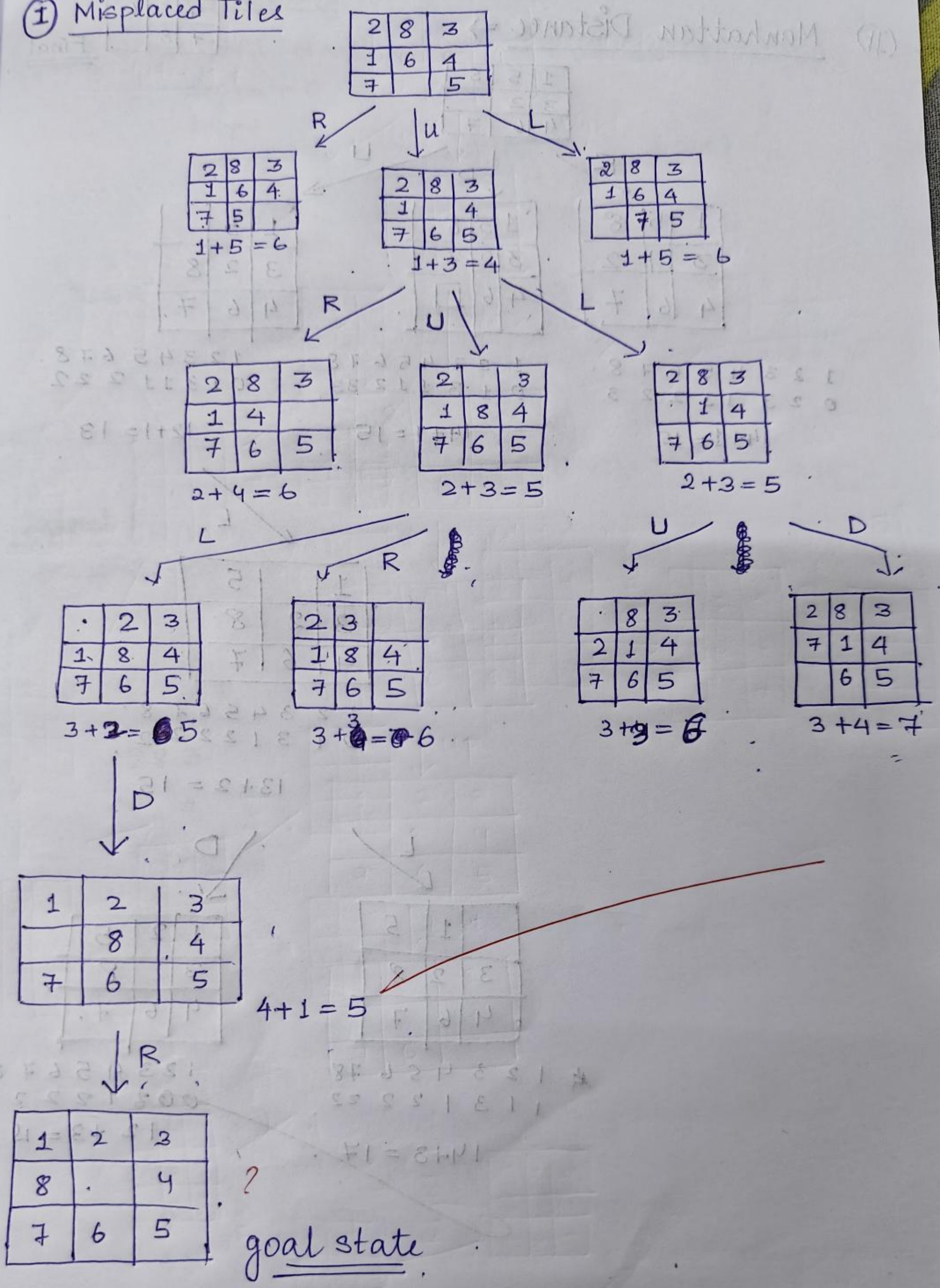
1	2	3
8		4
7	6	5

F

$$\boxed{f(n) = g(n) + h(n)}$$

(I) Misplac

(I) Misplaced Tiles



(II) Manhattan Distance ⇒

Final

1	2	3
4	5	6
7	8	

1	5	8
3	2	
4	6	7

R

1	5	8
3		2
4	6	7

1 2 3 4 5 6 7 8

0 2 3 1 1 2 2 3

14+1 = 15

D

1	5	8
3	2	7
4	6	

1 2 3 4 5 6 7 8

0 1 3 1 1 2 3 3

14+1 = 15

U

1	5	
3	2	8
4	6	7

1 2 3 4 5 6 7 8

0 1 3 1 1 2 2 2

12+1 = 13

L

1		5
3	2	8
4	6	7

1 2 3 4 5 6 7 8

0 1 3 1 2 2 2 2

13+2 = 15

L

	1	5
3	2	8
4	6	7

1 2 3 4 5 6 7 8

1 1 3 1 2 2 2 2

14+3 = 17

D

1	2	5
3		8
4	6	7

1 2 3 4 5 6 7 8

0 0 3 1 2 2 2 2

12 +3 = 15