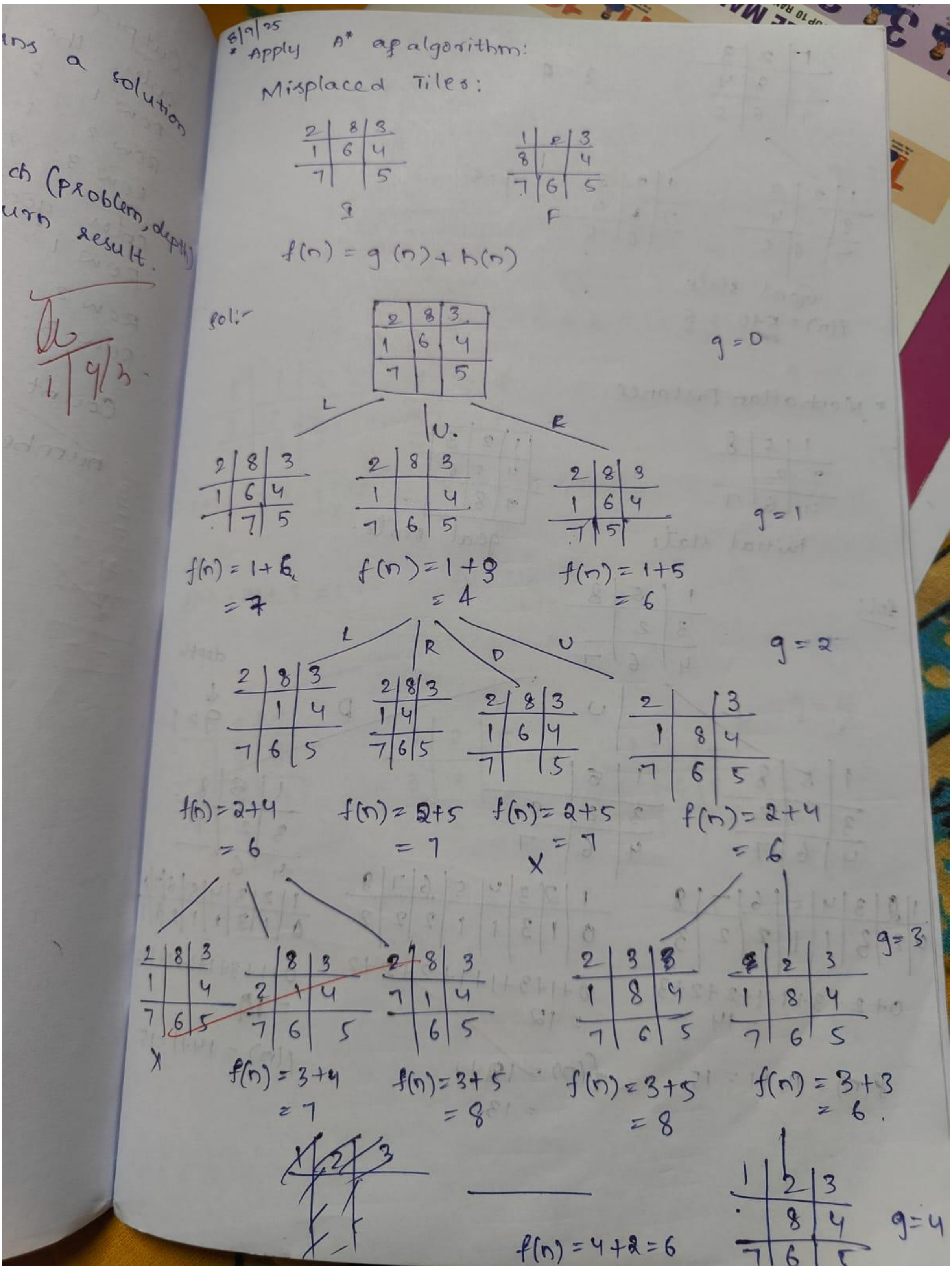
Implement a* Search using misplaced tiles



* A* search algorithm:

→ A* search evaluates nodes by combining g(n), the cost to reach the node and h(n), the cost to get from the node to the goal.

→ f(n) = g(n) + h(n)

- f(n) - evaluation funct^n which gives cheapest solution cost.
- g(n) - exact cost to reach node n from initial state.
- h(n) - estimation of the assumed cost from current state to reach the goal.

A* search using misplaced tiles:

Output: Initial state:

2 8 3 1 6 4 7 0 5

Goal state:

1 2 3 8 0 4 7 6 5

cost: 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]

A* search

Output:

Initial sta

2 8 3

Goal st

1 2

Total co

steps:

[2, 8, 3]
[1, 6, 4]
[7, 0, 5]

[2, 8, 3
[1, 0,
[7, 6,

[2, 0,
[1, 8,
[7, 6

[0,
[1
[

Output:

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

Solution found in 5 moves.
Total cost: 5

Steps:
(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)

Sareddy Poojya Sree
1BM23CS303
```

Code:

```
import heapq

# Heuristic: Misplaced tiles
def misplaced_tiles(state, goal):
    return sum(1 for i in range(9) if state[i] != 0 and state[i] != goal[i])

# Generate possible next states
def get_neighbors(state):
    neighbors = []
```

```
    blank = state.index(0)
    x, y = divmod(blank, 3)

    moves = [(-1,0), (1,0), (0,-1), (0,1)]  # up, down, left, right
    for dx, dy in moves:
        nx, ny = x + dx, y + dy
        if 0 <= nx < 3 and 0 <= ny < 3:
            new_blank = nx*3 + ny
            new_state = list(state)
            new_state[blank], new_state[new_blank] = new_state[new_blank],
new_state[blank]
            neighbors.append(tuple(new_state))
    return neighbors

# A* Search
def a_star(initial, goal):
    frontier = []
    heapq.heappush(frontier, (misplaced_tiles(initial, goal), 0, initial, [initial]))
    visited = set()

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

        if state == goal:
            return path, g  # return path and cost

        if state in visited:
            continue
        visited.add(state)

        for neighbor in get_neighbors(state):
            if neighbor not in visited:
                new_g = g + 1
                new_f = new_g + misplaced_tiles(neighbor, goal)
                heapq.heappush(frontier, (new_f, new_g, neighbor, path + [neighbor]))
    return None, None

# Print puzzle in 3x3 grid
def print_state(state):
    for i in range(0, 9, 3):
        print(state[i:i+3])
    print()

# Main
if __name__ == "__main__":
    print("Enter initial state (9 numbers, use 0 for blank):")
    initial = tuple(map(int, input().split()))
    print("Enter goal state (9 numbers, use 0 for blank):")
    goal = tuple(map(int, input().split()))
```

```
    path, cost = a_star(initial, goal)

    if path:
        print("\nSolution found in", len(path)-1, "moves.")
        print("Total cost:", cost)
        print("\nSteps:")
        for step in path:
            print_state(step)
    else:
        print("No solution found!")
print("Sareddy Poojya Sree\n1BM23CS303")
```

Implement a* using Manhattan Distance:

1	2	3
	8	4
7	6	5

g = 4

1	2	3
8		4
7	6	5

1	2	3
7	8	4
	6	5

g = 5

Goal state

$f(n) = \underline{5} + 0 = \underline{\underline{5}}$.

* Manhattan Distance.

1	5	8
3	2	
4	6	7

Initial state

1	2	3
4	5	6
7	8	

goal state.

Sol:

1	5	8
3	2	
4	6	7

depth

↓

g = 1

L

1	5	8
3		2
4	6	7

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

$0+2+3+1+1+2+2+3$
$= 14$

$f(n) = 14 + 1 = 15$

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

$0+1+3+1+1+2+2+2$
$= 12$

$f(n) = 12 + 1$
$= 13$

D

1	5	8
3	2	7
4	6	

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

$0+1+3+1+1+2+3+2$
$= 14$

$f(n) = 14 + 1 = 15$

1	5	
3	2	8
4	6	7

L

1		5
3	2	8
4	6	7

$g = 2$

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

$0+1+3+1+2+2+2+2$

$= 13$

$f(n) = 13 + 2 = 15$

$g = 3$

	1	5
3	2	8
4	6	7

$1+1+3+1+2+2+2+2$

$= 14$

$f(n) = 14 + 3 = 17$

1	2	5
3		8
4	6	7

$0+0+3+1+2+2+2+2$

$= 12$

$f(n) = 12 + 3$

$= 15$

$g = 4$

1	2	5
	3	8
4	6	7

1	2	5
3	8	
4	6	7

1	2	5
3	6	8
4		7

depth

↓

$g = 1$

8

7

5	6	7	8
1	2	3	2

$1+2+3+3$

$+1 = 15$

by combining
node and h(n).
ode to the goal.

gives cheapest

node n from

med cost from
goal.

A* search using Manhattan Distance:

Output:

Initial state:
2 8 3 1 6 4 7 0 5

Goal state:
1 2 3 8 0 4 7 6 5

Total cost: 5

steps:

[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]

Output:

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

Solution found in 5 moves.
Total cost: 5

Steps:
(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)

Sareddy Poojya Sree
1BM23CS303
```

Code:

```python
import heapq

# Heuristic: Manhattan distance
def manhattan(state, goal):
    distance = 0
    for i in range(1, 9):  # ignore blank (0)
        x1, y1 = divmod(state.index(i), 3)
        x2, y2 = divmod(goal.index(i), 3)
        distance += abs(x1 - x2) + abs(y1 - y2)
```

```
    return distance

# Generate possible next states
def get_neighbors(state):
    neighbors = []
    blank = state.index(0)
    x, y = divmod(blank, 3)

    moves = [(-1,0), (1,0), (0,-1), (0,1)]  # up, down, left, right
    for dx, dy in moves:
        nx, ny = x + dx, y + dy
        if 0 <= nx < 3 and 0 <= ny < 3:
            new_blank = nx*3 + ny
            new_state = list(state)
            new_state[blank], new_state[new_blank] = new_state[new_blank],
new_state[blank]
            neighbors.append(tuple(new_state))
    return neighbors

# A* Search
def a_star(initial, goal):
    frontier = []
    heapq.heappush(frontier, (manhattan(initial, goal), 0, initial, [initial]))
    visited = set()

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

        if state == goal:
            return path, g  # return path and cost

        if state in visited:
            continue
        visited.add(state)

        for neighbor in get_neighbors(state):
            if neighbor not in visited:
                new_g = g + 1
                new_f = new_g + manhattan(neighbor, goal)
                heapq.heappush(frontier, (new_f, new_g, neighbor, path + [neighbor]))
    return None, None

# Print puzzle in 3x3 grid
def print_state(state):
    for i in range(0, 9, 3):
        print(state[i:i+3])
```

```python
    print()

# Main
if __name__ == "__main__":
    print("Enter initial state (9 numbers, use 0 for blank):")
    initial = tuple(map(int, input().split()))
    print("Enter goal state (9 numbers, use 0 for blank):")
    goal = tuple(map(int, input().split()))

    path, cost = a_star(initial, goal)

    if path:
        print("\nSolution found in", len(path)-1, "moves.")
        print("Total cost:", cost)
        print("\nSteps:")
        for step in path:
            print_state(step)
    else:
        print("No solution found!")
print("Sareddy Poojya Sree\n1BM23CS303")
```