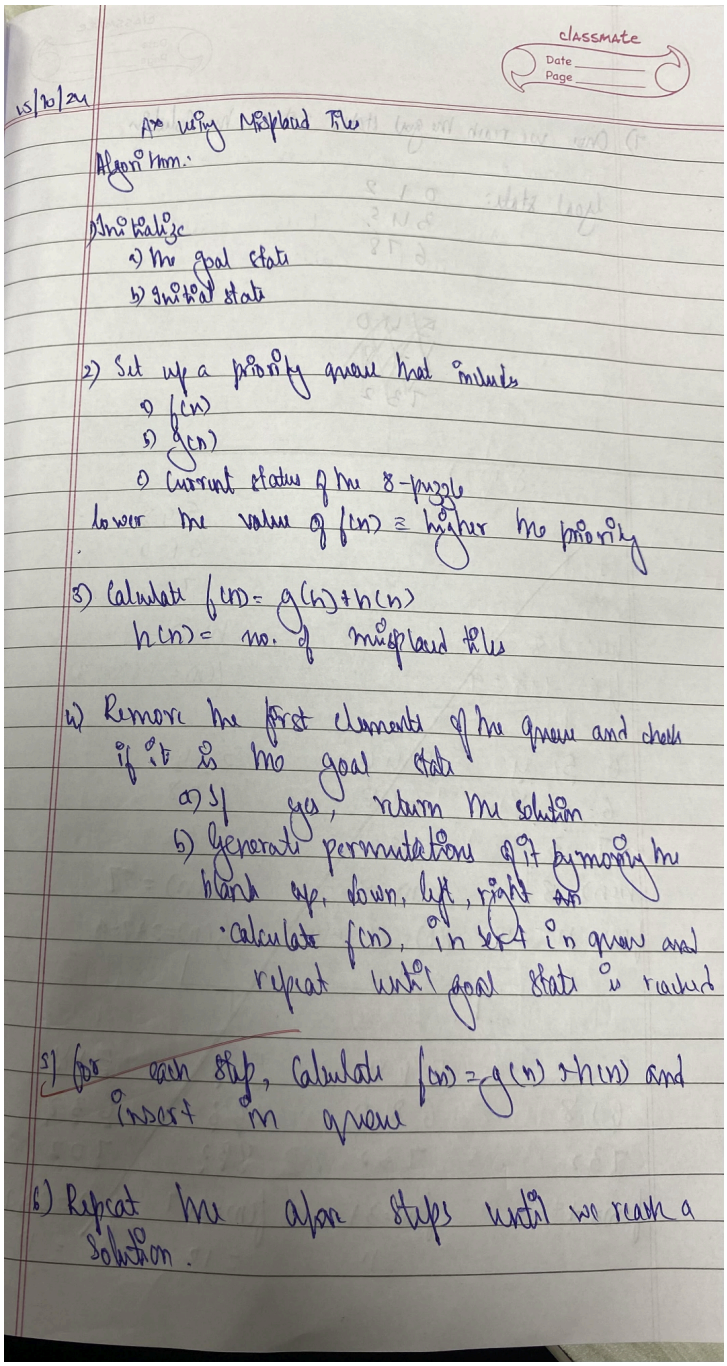
Date:15/10/2024

Program Title: A* Search using misplaced tiles method

Algorithm:



Code:

```python
import heapq
goal_state = [
    [0, 1, 2],
    [3, 4, 5],
    [6, 7, 8]
]
def flatten(puzzle):
    return [item for row in puzzle for item in row]
def find_blank(puzzle):
    for i in range(3):
        for j in range(3):
            if puzzle[i][j] == 0:
                return i, j
def misplaced_tiles(puzzle):
    flat_puzzle = flatten(puzzle)
    flat_goal = flatten(goal_state)
    return sum([1 for i in range(9) if flat_puzzle[i] != flat_goal[i] and flat_puzzle[i] != 0])
def generate_neighbors(puzzle):
    x, y = find_blank(puzzle)
    neighbors = []
    moves = [(-1, 0), (1, 0), (0, -1), (0, 1)]
    for dx, dy in moves:
        nx, ny = x + dx, y + dy
        if 0 <= nx < 3 and 0 <= ny < 3:
            new_puzzle = [row[:] for row in puzzle]
            new_puzzle[x][y], new_puzzle[nx][ny] = new_puzzle[nx][ny], new_puzzle[x][y]
            neighbors.append(new_puzzle)
    return neighbors
def is_goal(puzzle):
    return puzzle == goal_state
def print_puzzle(puzzle):
    for row in puzzle:
        print(row)
    print()
def a_star_misplaced_tiles(initial_state):
    frontier = []
    heapq.heappush(frontier, (misplaced_tiles(initial_state), 0, initial_state, []))
    visited = set()
    while frontier:
        f, g, current_state, path = heapq.heappop(frontier)
        print("Current State:")
        print_puzzle(current_state)
        h = misplaced_tiles(current_state)
        print(f"g(n) = {g}, h(n) = {h}, f(n) = {g + h}")
        print("-" * 20)
        if is_goal(current_state):
            print("Goal reached!")
            return path
```

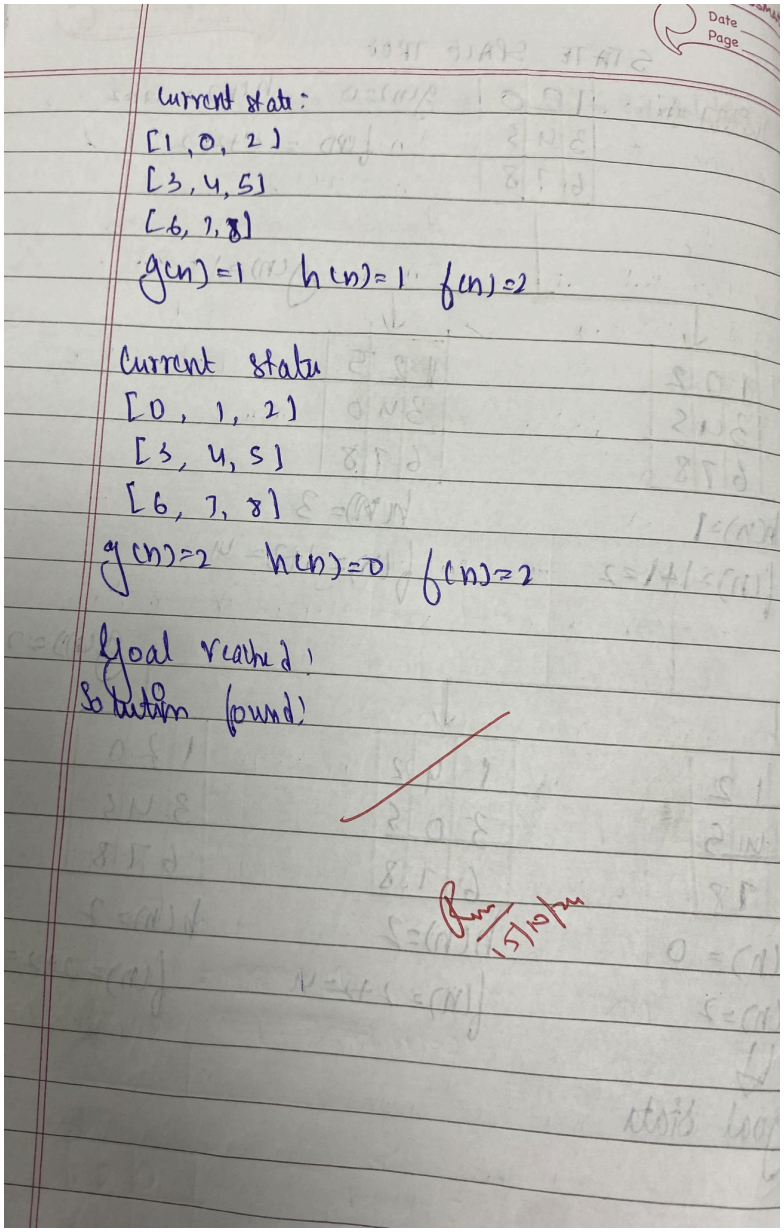
```
            return path
        visited.add(tuple(flatten(current_state)))
        for neighbor in generate_neighbors(current_state):
            if tuple(flatten(neighbor)) not in visited:
                h = misplaced_tiles(neighbor)
                heapq.heappush(frontier, (g + 1 + h, g + 1, neighbor, path + [neighbor]))
    return None
initial_state = [
    [1, 2, 0],
    [3, 4, 5],
    [6, 7, 8]
]
solution = a_star_misplaced_tiles(initial_state)
if solution:
    print("Solution found!")
else:
    print("No solution found.")
```

```
import heapq
goal_state = [
    [0, 1, 2],
    [3, 4, 5],
    [6, 7, 8]
]
def flatten(puzzle):
    return [item for row in puzzle for item in row]
def find_blank(puzzle):
    for i in range(3):
        for j in range(3):
            if puzzle[i][j] == 0:
                return i, j
def misplaced_tiles(puzzle):
    flat_puzzle = flatten(puzzle)
    flat_goal = flatten(goal_state)
    return sum([1 for i in range(9) if flat_puzzle[i] != flat_goal[i] and flat_puzzle[i] != 0])
def generate_neighbors(puzzle):
    x, y = find_blank(puzzle)
    neighbors = []
    moves = [(-1, 0), (1, 0), (0, -1), (0, 1)]
    for dx, dy in moves:
        nx, ny = x + dx, y + dy
        if 0 <= nx < 3 and 0 <= ny < 3:
            new_puzzle = [row[:] for row in puzzle]
            new_puzzle[x][y], new_puzzle[nx][ny] = new_puzzle[nx][ny], new_puzzle[x][y]
            neighbors.append(new_puzzle)
    return neighbors
def is_goal(puzzle):
    return puzzle == goal_state
```

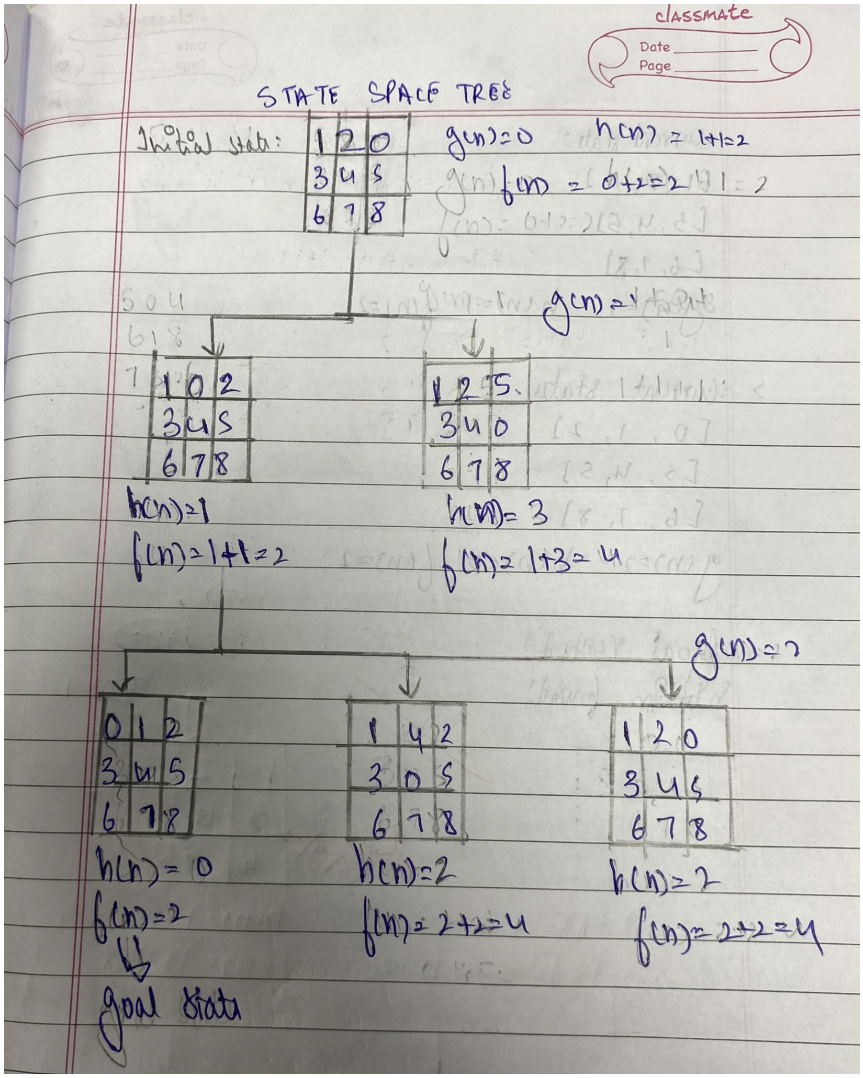
```
def print_puzzle(puzzle):
    for row in puzzle:
        print(row)
    print()
def a_star_misplaced_tiles(initial_state):
    frontier = []
    heapq.heappush(frontier, (misplaced_tiles(initial_state), 0, initial_state, []))
    visited = set()
    while frontier:
        f, g, current_state, path = heapq.heappop(frontier)
        print("Current State:")
        print_puzzle(current_state)
        h = misplaced_tiles(current_state)
        print(f"g(n) = {g}, h(n) = {h}, f(n) = {g + h}")
        print("-" * 20)
        if is_goal(current_state):
            print("Goal reached!")
            return path
        visited.add(tuple(flatten(current_state)))
        for neighbor in generate_neighbors(current_state):
            if tuple(flatten(neighbor)) not in visited:
                h = misplaced_tiles(neighbor)
                heapq.heappush(frontier, (g + 1 + h, g + 1, neighbor, path + [neighbor]))
    return None
initial_state = [
    [1, 2, 0],
    [3, 4, 5],
    [6, 7, 8]
]
solution = a_star_misplaced_tiles(initial_state)
if solution:
    print("Solution found!")
else:
    print("No solution found.")
```

Output:

```
Current State:
[1, 2, 0]
[3, 4, 5]
[6, 7, 8]

g(n) = 0, h(n) = 2, f(n) = 2
--------------------
Current State:
[1, 0, 2]
[3, 4, 5]
[6, 7, 8]

g(n) = 1, h(n) = 1, f(n) = 2
--------------------
Current State:
[0, 1, 2]
[3, 4, 5]
[6, 7, 8]

g(n) = 2, h(n) = 0, f(n) = 2
--------------------
Goal reached!
Solution found!
```



State Space tree:


classmate
Date
Page
STATE SPACE TREE
Initial state:
1 2 0
3 4 5
6 7 8
g(n)=0
h(n) = 1+1=2
f(n) = 0+2=2
g(n)=1
1 0 2
3 4 5
6 7 8
h(n)=1
f(n)=1+1=2
1 2 5
3 4 0
6 7 8
h(n)= 3
f(n)= 1+3= 4
g(n)=2
0 1 2
3 4 5
6 7 8
h(n) = 0
f(n)=2
goal state
1 4 2
3 0 5
6 7 8
h(n)=2
f(n)= 2+2=4
1 2 0
3 4 5
6 7 8
h(n)=2
f(n)= 2+2=4