15/10/2024

Lab-03

Program title: For 8-puzzle A* implementation, to calculate, f(n), consider two cases,
1. g(n): Depth of the node, h(n): Number of Misplaced tiles

Code:

```
import heapq


# Goal state where blank (0) is the first tile
goal_state = [
    [1, 2, 3],
    [8, 0, 4],
    [7, 6, 5]
]


# Helper functions
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):
    # Priority queue (min-heap) and visited states
    frontier = []
    heapq.heappush(frontier, (misplaced_tiles(initial_state), 0, initial_state, []))
    visited = set()

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

        # Print the current state
        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  # No solution found

# Initial puzzle state
initial_state = [
    [2, 8, 3],
    [1, 6, 4],
    [7, 0, 5]
]

solution = a_star_misplaced_tiles(initial_state)
if solution:
    print("Solution found!")
else:
    print("No solution found.")
```

Algorithm:

program title: A* search algorithm with misplaced tiles

algorithm:

1. initialize:
   (i) start with the initial state of puzzle
   (ii) set the goal state (which is when the blank tile is in the top left corner and all tiles in order)

2. priority queue:
   (i) use a priority queue (or min-heap) to store states of puzzle prioritized by $f(n) = g(n) + h(n)$
   (i) $g(n)$ is number of moves (steps) taken from the start
   (ii) $h(n)$ is the misplaced tiles heuristic, which counts how many tiles are not in their goal positions

3. explore states:
   (i) remove the state with smallest $f(n)$ from the queue
   (ii) if this state is the goal state stop and return the solution
   (iii) otherwise generate all possible new states by moving the blank tile up, down, left (or) right.

4. evaluate new states:
   (i) for each new state
   (i) calculate $g(n)$ (number of steps taken so far)
   (ii) calculate $h(n)$ (number of misplaced tiles in new state)
   (iii) add this new state to the priority queue with its $f(n) = g(n) + h(n)$

5. repeat:
   (i) continue exploring states from the queue until the goal state is reached

6. goal reached:
   (i) once the goal state is reached the algorithm terminates and outputs the solution

15/10/2024

~~program title : 8 puzzle~~

program title : 8 puzzle problem with missing tiles

algorithm :-

node class :

initialization : create a node with

data : puzzle state
level : depth in the search tree
fval : total cost estimate

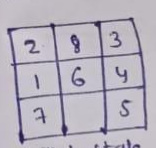
generate_child() :

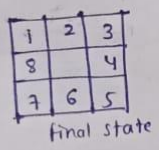
find blank space position
generate possible moves (up, down, left, right)
create child nodes for valid moves

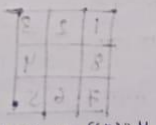
shuffle(puzzle, x1, y1, x2, y2) :

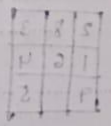
swap blank space with the specified position if within bounds

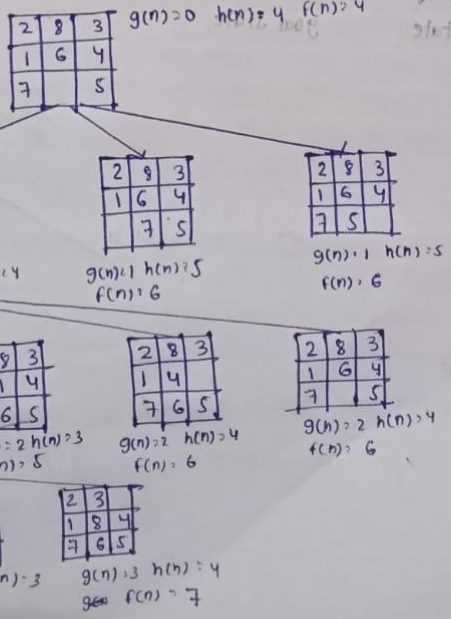
implement A* search algorithm using missing tiles

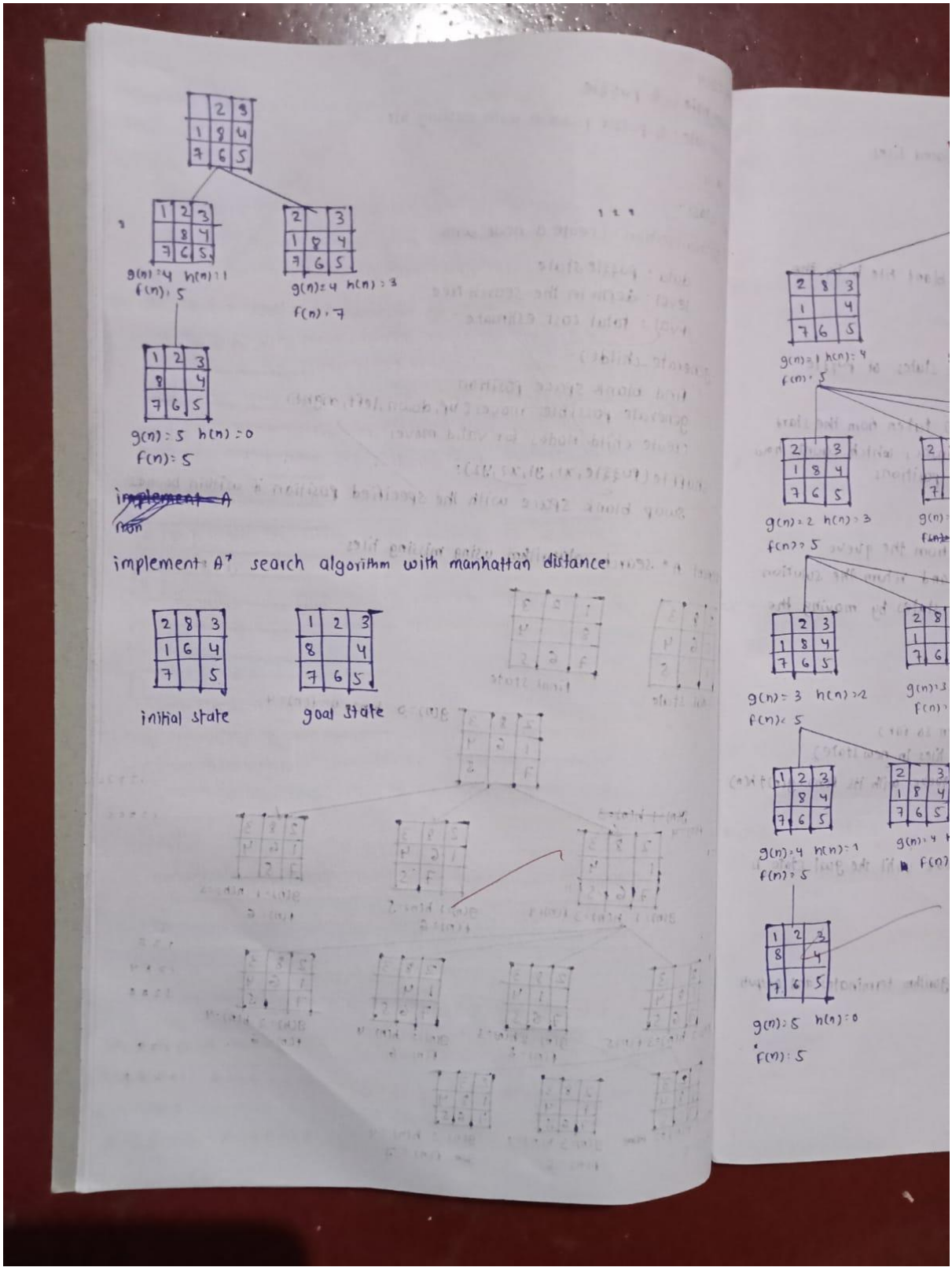

initial state


final state









Output:

Current State:

[2, 8, 3]

[1, 6, 4]

[7, 0, 5]

g(n) = 0, h(n) = 4, f(n) = 4

Current State:

[2, 8, 3]

[1, 0, 4]

[7, 6, 5]

g(n) = 1, h(n) = 3, f(n) = 4

Current State:

[2, 0, 3]

[1, 8, 4]

[7, 6, 5]

g(n) = 2, h(n) = 3, f(n) = 5

Current State:

[2, 8, 3]

[0, 1, 4]

[7, 6, 5]

g(n) = 2, h(n) = 3, f(n) = 5

Current State:

[0, 2, 3]

[1, 8, 4]

[7, 6, 5]

g(n) = 3, h(n) = 2, f(n) = 5

Current State:

[1, 2, 3]

[0, 8, 4]

[7, 6, 5]

g(n) = 4, h(n) = 1, f(n) = 5

Current State:

[1, 2, 3]

[8, 0, 4]

[7, 6, 5]

g(n) = 5, h(n) = 0, f(n) = 5

Goal reached!

Solution found!