15/10/2024

Lab-03

Program title: For 8-puzzle A* implementation, to calculate, f(n), consider two cases,
2. g(n): Depth of the node, h(n): Manhattan Distance

Code:

```
import heapq


goal_state = [

    [1, 2, 3],

    [8, 0, 4],

    [7, 6, 5]

]


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 manhattan_distance(puzzle):

    distance = 0

    # Define target positions based on the goal state

    target_positions = {

        1: (0, 0), 2: (0, 1), 3: (0, 2),

        4: (1, 2), 5: (2, 2), 6: (2, 1),

        7: (2, 0), 8: (1, 0), 0: (1, 1)  # Blank space

    }
```

```
    for i in range(3):
        for j in range(3):
            value = puzzle[i][j]
            if value != 0:  # Skip the blank tile
                target_x, target_y = target_positions[value]
                distance += abs(i - target_x) + abs(j - target_y)
    return distance

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_manhattan_distance(initial_state):
    frontier = []
```

```python
    heapq.heappush(frontier, (manhattan_distance(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 = manhattan_distance(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 = manhattan_distance(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_manhattan_distance(initial_state)
if solution:
    print("Solution found!")
else:
    print("No solution found.")
```

Algorithm:

program title : A* search algorithm with manhattan value

algorithm:

1. initialize :
   (i) start with the initial state of the puzzle
   (ii) set the goal state

2. priority queue :
   (i) use a priority queue (or min-heap) to store states of the puzzle ~~priori~~ prioritized by $f(n) = g(n) + h(n)$
      (i) $g(n)$: the number of moves (steps) taken from the start
      (ii) $h(n)$: the manhattan distance ~~heu~~ heuristic which sums the distances of each tile from its correct position

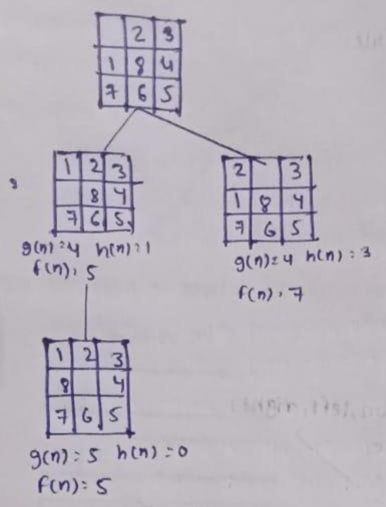
3. explore states :
   (i) remove the state with smallest $f(n)$ from the queue
   (ii) if this state is the goal state stop and return the solution
   (iii) otherwise generate all possible new states by moving the blank tile left, up, down (or) right

4. evaluate new states :
   (i) for each new state
      (i) calculate $g(n)$ (number of steps taken so far)
      (ii) calculate $h(n)$ (the manhattan distance)
      (iii) add this new state to the priority queue with it's $f(n) = g(n) + h(n)$

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

15/10/24



~~implement A~~
~~non~~

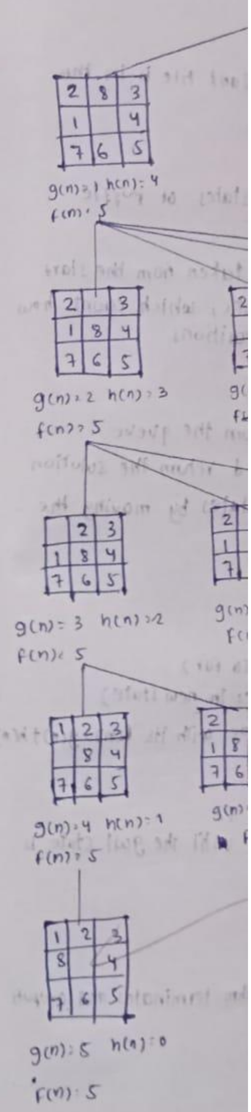
implement A* search algorithm with manhattan distance

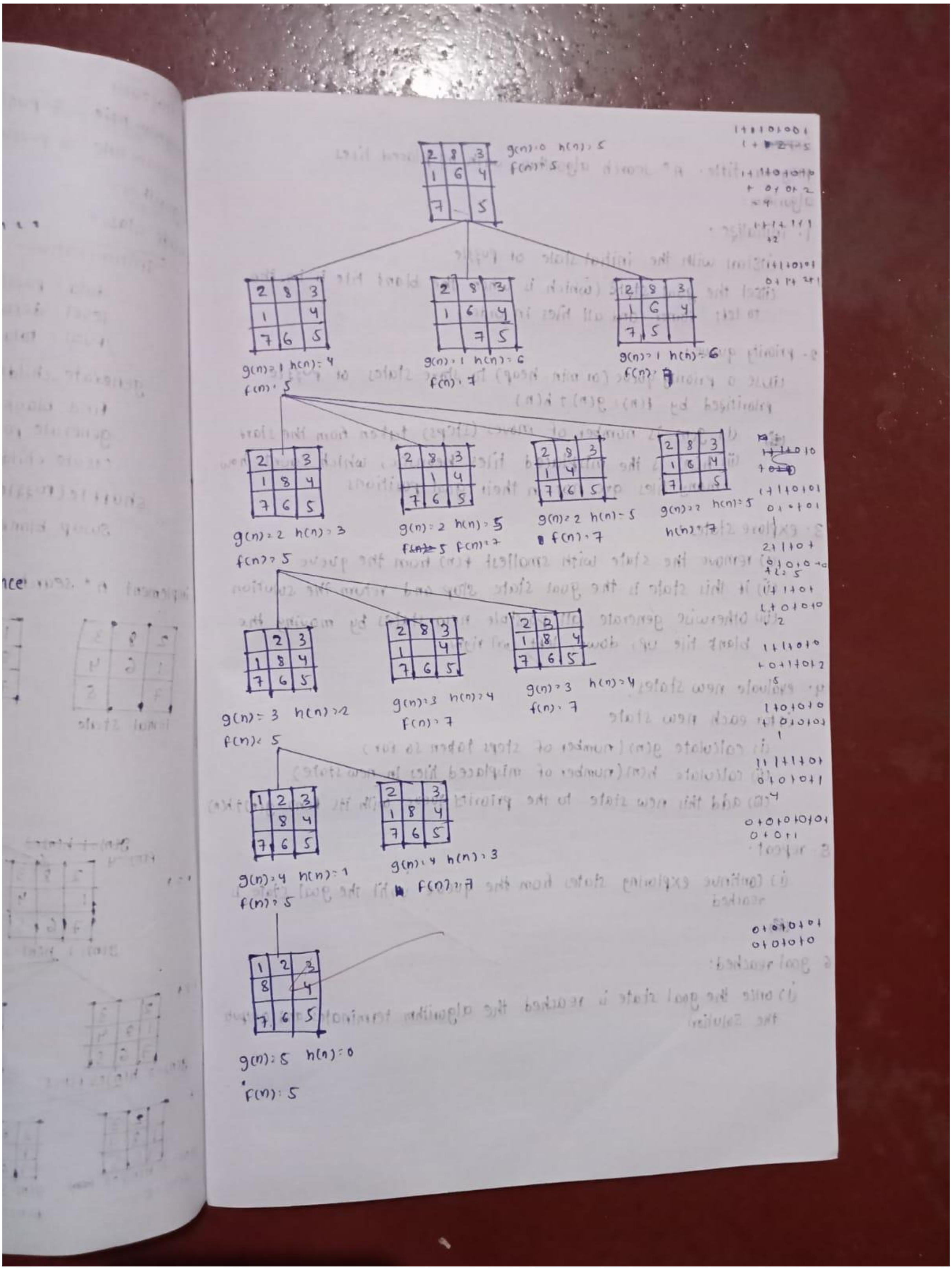
2	8	3
1	6	4
7		5

initial state

1	2	3
8		4
7	6	5

goal state





Output:

Current State:

[2, 8, 3]

[1, 6, 4]

[7, 0, 5]

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

Current State:

[2, 8, 3]

[1, 0, 4]

[7, 6, 5]

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

Current State:

[2, 0, 3]

[1, 8, 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!