15/10/24 LAB 03- A STAR SEARCH ALGORITHM

ALGORITHM-

classmate
Date 15/10/24
Page 05

WEEK-3

Implementing A* (A star) search algorithm.

ALGORITHM :

Step 1: Initialization :
- start from initial state of puzzle
- use priority queue to explore nodes based on $f(n) = g(n) + h(n)$

Step 2: Expansion :
- expand node with lowest $f(n)$
- generate child nodes by sliding adjacent tiles into blank space

Step 3: Evaluation
- for each child node calculate $f(n)$
- continue until goal state is reached.

Step 4: Termination
- the search ends when current node corresponds to goal state
- backtrack to get solution.

Heuristics →

Number of misplaced Tiles:
counts of how many tiles are not in current position

Manhattan distance :
sums the distance of each tile from current position to goal state.

OUTPUT:

1. Number of tiles.

solution path using misplaced tiles:

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

2 8 3
1 6 4
0 7 5

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

2 8 3
1 6 4
7 0 5

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

2 8 3
1 0 4
7 6 5

step 3: h(n)=3 g(n)= 3 f(n)= 6

2 0 3
1 8 4
7 6 5

step 4: h(n)=2 g(n)=4 f(n)= 6

0 2 3
1 8 4
7 6 5

step 5 : $h(n) = 1$ $g(n) = 5$ $f(n) = 6$

1	2	3
0	8	4
7	6	5

step 6 : $h(n) = 0$ $g(n) = 6$ $f(n) = 6$

1	2	3
8	0	4
7	6	5

2. Manhattan distance

step 0 : $h(n) = 6$, $g(n) = 0$, $f(n) = 6$

2	8	3
1	6	4
0	7	5

step 1 : $h(n) = 5$ $g(n) = 1$ $f(n) = 6$

2	8	3
1	6	4
7	0	5

step 2 : $h(n) = 4$, $g(n) = 2$, $f(n) = 6$

2	8	3
1	0	4
7	6	5

step 3 : $h(n) = 3$ $g(n) = 3$ $f(n) = 6$

2	0	3
1	8	4
7	6	5

Page 12

step 4 : h(n) = 2, g(n) = 4, f(n) = 6

0	2	3
1	8	4
7	6	5

step 5 : h(n) = 1, g(n) = 5 f(n) = 6

1	2	3
0	8	4
7	6	5

step 6 : h(n) = 0, g(n) = 6, f(n) = 6

1	2	3
8	0	4
7	6	5

13/10/24

CODE / INPUT -

MISPLACED TILES-

```
goal_state = [
    [1, 2, 3],
    [8, 0, 4],  # 0 represents the blank space
    [7, 6, 5]
]
# Function to calculate the number of misplaced tiles
def misplaced_tiles(state, goal_state):
    count = 0
    for i in range(3):
```

```
        for j in range(3):
            if state[i][j] != 0 and state[i][j] != goal_state[i][j]:
                count += 1
    return count
import heapq
def find_blank_tile(state):
    for i in range(3):
        for j in range(3):
            if state[i][j] == 0:
                return i, j
def get_possible_moves(state):
    row, col = find_blank_tile(state)
    moves = []
    if row > 0:
        new_state = [r[:] for r in state]
        new_state[row][col], new_state[row - 1][col] = new_state[row - 1][col],
new_state[row][col]
        moves.append(new_state)
    if row < 2:
        new_state = [r[:] for r in state]
        new_state[row][col], new_state[row + 1][col] = new_state[row + 1][col],
new_state[row][col]
        moves.append(new_state)
    if col > 0:
        new_state = [r[:] for r in state]
        new_state[row][col], new_state[row][col - 1] = new_state[row][col - 1],
new_state[row][col]
        moves.append(new_state)
    if col < 2:
        new_state = [r[:] for r in state]
        new_state[row][col], new_state[row][col + 1] = new_state[row][col + 1],
new_state[row][col]
        moves.append(new_state)
    return moves
def a_star_search_misplaced(initial_state, goal_state, heuristic):
    open_list = [(heuristic(initial_state, goal_state), 0, initial_state, [])]  # (f(n), g(n), state, path)
    closed_set = set()
    while open_list:
        f_n, g_n, current_state, path = heapq.heappop(open_list)
```

```
        if current_state == goal_state:
            return path + [current_state]
       state_tuple = tuple(map(tuple, current_state))
        if state_tuple in closed_set:
            continue
        closed_set.add(state_tuple)
        for neighbor in get_possible_moves(current_state):
            neighbor_tuple = tuple(map(tuple, neighbor))
            if neighbor_tuple in closed_set:
                continue
            new_g_n = g_n + 1
            new_h_n = heuristic(neighbor, goal_state)
            new_f_n = new_g_n + new_h_n
            heapq.heappush(open_list, (new_f_n, new_g_n, neighbor, path + [current_state]))
    return None
def display_solution_misplaced(path, heuristic, goal_state):
    if path:
        print(f"Solution path using misplaced tiles heuristic:")
        for step, state in enumerate(path):
            h = heuristic(state, goal_state)
            g = step
            f = g + h
            print(f"Step {step}: h(n)={h}, g(n)={g}, f(n)={f}")
            for row in state:
                print(row)
            print()
    else:
        print(f"No solution found using misplaced tiles heuristic.")
initial_state = [
    [2, 8, 3],
    [1, 6, 4],
    [0, 7, 5]
]
solution_path_misplaced = a_star_search_misplaced(initial_state, goal_state, misplaced_tiles)
display_solution_misplaced(solution_path_misplaced, misplaced_tiles, goal_state)
```

OUTPUT-

```
Solution path using misplaced tiles heuristic:
Step 0: h(n)=5, g(n)=0, f(n)=5
[2, 8, 3]
[1, 6, 4]
[0, 7, 5]

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

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

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

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

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

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

CODE / INPUT -

MANHATTAN DISTANCE-

```
goal_state = [
    [1, 2, 3],
    [8, 0, 4],  # 0 represents the blank space
    [7, 6, 5]
]

# Function to calculate Manhattan distance
def manhattan_distance(state, goal_state):
    distance = 0
    for i in range(3):
        for j in range(3):
            if state[i][j] != 0:
                goal_position = [(row, col) for row in range(3) for col in range(3) if
goal_state[row][col] == state[i][j]][0]
                distance += abs(i - goal_position[0]) + abs(j - goal_position[1])
    return distance
import heapq
def find_blank_tile(state):
    for i in range(3):
        for j in range(3):
            if state[i][j] == 0:
                return i, j
def get_possible_moves(state):
    row, col = find_blank_tile(state)
    moves = []
    if row > 0:
        new_state = [r[:] for r in state]
        new_state[row][col], new_state[row - 1][col] = new_state[row - 1][col],
new_state[row][col]
        moves.append(new_state)
    if row < 2:
        new_state = [r[:] for r in state]
        new_state[row][col], new_state[row + 1][col] = new_state[row + 1][col],
new_state[row][col]
        moves.append(new_state)
    if col > 0:
```

```
        new_state = [r[:] for r in state]
        new_state[row][col], new_state[row][col - 1] = new_state[row][col - 1],
new_state[row][col]
        moves.append(new_state)
    if col < 2:
        new_state = [r[:] for r in state]
        new_state[row][col], new_state[row][col + 1] = new_state[row][col + 1],
new_state[row][col]
        moves.append(new_state)
    return moves
def a_star_search_manhattan(initial_state, goal_state, heuristic):
    open_list = [(heuristic(initial_state, goal_state), 0, initial_state, [])]  # (f(n), g(n), state, path)
    closed_set = set()
    while open_list:
        f_n, g_n, current_state, path = heapq.heappop(open_list)
        if current_state == goal_state:
            return path + [current_state]
        state_tuple = tuple(map(tuple, current_state))
        if state_tuple in closed_set:
            continue
        closed_set.add(state_tuple)
        for neighbor in get_possible_moves(current_state):
            neighbor_tuple = tuple(map(tuple, neighbor))
            if neighbor_tuple in closed_set:
                continue
            new_g_n = g_n + 1
            new_h_n = heuristic(neighbor, goal_state)
            new_f_n = new_g_n + new_h_n
            heapq.heappush(open_list, (new_f_n, new_g_n, neighbor, path + [current_state]))

    return None
def display_solution_manhattan(path, heuristic, goal_state):
    if path:
        print(f"Solution path using Manhattan distance heuristic:")
        for step, state in enumerate(path):
            h = heuristic(state, goal_state)
            g = step
            f = g + h
            print(f"Step {step}: h(n)={h}, g(n)={g}, f(n)={f}")
            for row in state:
```

```
            print(row)
        print()
    else:
        print(f"No solution found using Manhattan distance heuristic.")

initial_state = [
    [2, 8, 3],
    [1, 6, 4],
    [0, 7, 5]
]
solution_path_manhattan = a_star_search_manhattan(initial_state, goal_state,
manhattan_distance)
display_solution_manhattan(solution_path_manhattan, manhattan_distance, goal_state)
```

OUTPUT -

```
Solution path using Manhattan distance heuristic:
Step 0: h(n)=6, g(n)=0, f(n)=6
[2, 8, 3]
[1, 6, 4]
[0, 7, 5]

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

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

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

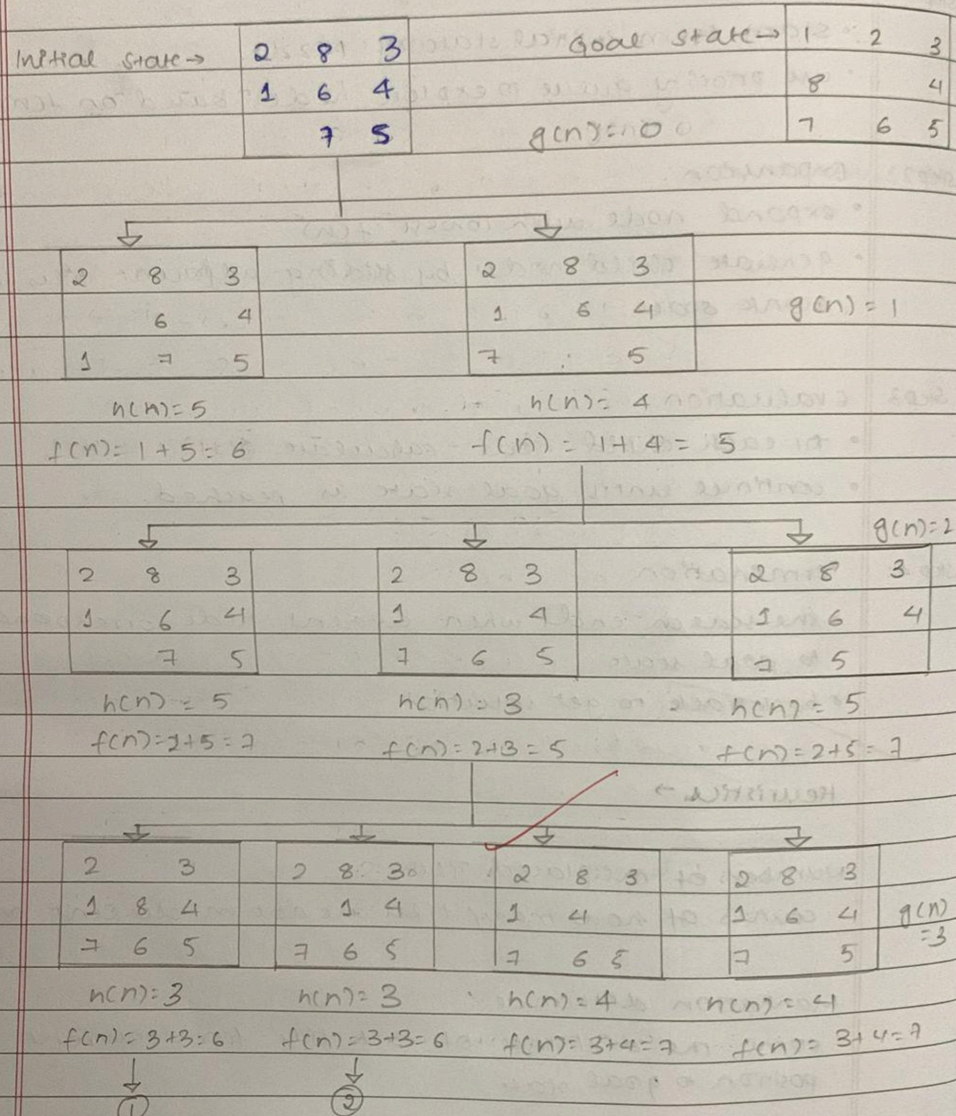
```
Step 4: h(n)=2, g(n)=4, f(n)=6
[0, 2, 3]
[1, 8, 4]
[7, 6, 5]

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

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

STATE SPACE TREE-

STATE SPACE TREES

1. Number of misplaced tiles.

$f(n) = g(n) + h(n)$



①

	2	3
1	8	4
7	6	5

h(n) = 2
f(n) = 4+2 = 6

2	3	
1	8	4
7	6	5

h(n) = 4
f(n) = 4+4 = 8

2	8	3
1		4
7	6	5

h(n) = 3
f(n) = 4+3 = 7

②

	8	3
2	1	4
7	6	5

h(n) = 3
f(n) = 4+3 = 7

g(n) = 4.

2	8	3
1.		4
7	6	5

h(n) = 3
f(n) = 4+3 = 7

2	8	3
7	1	4
	6	5

h(n) = 4
f(n) = 4+4 = 8

2		3
1	8	4
7	6	5

h(n) : 3
f(n) : 5+3 = 8

1.	2	3
	8	4
7	6	5

h(n) = 1
f(n) = 5+1 = 6

g(n) = 5

	2	3
1	8	4
7	6	5

h(n) = 2
f(n) = 6+2 = 8

1	2	3
8		4
7	6	5

h(n) = 0
f(n) = 6+0 = 6

1	2	3
7.	8	4
	6	5

h(n) = 2
f(n) = 6+2 = 8.

g(n) = 6

Goal state reached

2. Manhattan Method.

$f(n) = g(n) + h(n)$

Initial state →

2	8	3
1	6	4
	7	5

goal state →

1	2	3
8		4
7	6	5

$g(n) = 0$

2	8	3
1	6	4
7		5

$h(n) = 1+1+[illegible]+2 = 5$

$f(n) = 1+5 = 6$

2	8	3
	6	4
1	7	5

$h(n) = 1+2+1+1+2$

$f(n) = 1+7 = 8$

$g(n) = 1$

2	8	3
1	6	4
	7	5

$h(n) = 1+1+2+1+1$

$f(n) = 2+6 = 8$

2	8	3
1		4
7	6	5

$h(n) = 1+1+2$

$f(n) = 2+4 = 6$

2	8	3
1	6	4
7	5	

$h(n) = 1+1+2+1+1$

$f(n) = 2+6 = 8$

$g(n) = 2$

2		3
1	8	4
7	6	5

$h(n) = 1+1+1$

$f(n) = 3+3 = 6$

↓

1

2	8	3
	1	4
7	6	5

$h(n) = 1+2+2$

$f(n) = 3+5 = 8$

2	8	3
1	4	
7	6	5

$h(n) = 1+1+2+1$

$f(n) = 3+5 = 8$

2	8	3
1	6	4
7		5

$h(n) = 1+1+2+1$

$f(n) = 3+5 = 8$

$g(n) = 3$

	2	3
1	8	4
7	6	5

h(n) = 1+1

f(n) = 4+2 = 6

2	3	
1	8	4
7	6	5

h(n) = 1+1+1+1

f(n) = 4+4 = 8

2	8	3
1		4
7	6	5

g(n) = 4

h(n) = 1+2+1

f(n) = 4 + 4 = 8

1	2	3
	8	4
7	6	5

h(n) = 1

f(n) = 5+1 = 6

2		3
1	8	4
7	6	5

g(n) = 5

h(n) = 1+1+1

f(n) = 5 + 3 = 8

1	2	3
8		4
7	6	5

h(n) = 0

f(n) = 6

goal state reached

	2	3
1	8	4
7	6	5

h(n) = 1+1

f(n) = 6 + 2 = 8

1	2	3
7	8	4
	6	5

g(n) = 6

h(n) = 1+1

f(n) = 6+2 = 8