Lab program - 4

4) 8 Puzzle A* search algorithm

Algorithm

Step 1: Define Puzzle Node class

- Each node represents a state of the puzzle
- The state is a 3X3 grid.

Step 2: Define PuzzleNode methods

calculate_heuristic(): calculate the heuristic value for the current state

__lt__(self, other)

Step 3: Define helper functions

get_blank_position(state)
Find the position of the blank(0)

get_neighbors(node)
Get valid neighbours for a given tile.

apply_action(state, action)
Apply the given action.

Step 4: A search algorithm

Initialize the initial state as a puzzle node.

Initialize a set to store explored states

Step 5: Print solution

Trace back from goal state to initial state

Step 6: Main function
Define the initial state of puzzle
Call the A* algorithm with the initial state.

Code

```
class
Node:
    def __init__(self, data, level, fval):
        """ Initialize the node with the data,
        level of the node with the data,
        level of the node and the calculated
            f value """
        self.data = data
        self.level = level
        self.fval = fval

    def generate_child(self):
        """ Generate child nodes from the give node
            by moving the blank space either
            in the four directions { up, down,
            left, right } """
        x, y = self.find(self.data, x, y,
                         i[0], i[1])

        if child is not None:
            childnode = Node(child, self.level+1)
```

```
        self.children.append(child_node)
        return self.children

    def shuffle(self, puz, x, y, s, d):
        """Move the blank space in the given
           direction."""
        if x + dx >= 0 and x + dx < len(self.data) and
           y + dy >= 0 and y + dy < len(self.data):

            temp_puz = []
            temp_puz = self.copy(puz)
            temp = temp_puz[x][y]
            temp_puz[x][y] = temp_puz[x+dx][y+dy]
            temp_puz[x+dx][y+dy] = temp
            return temp_puz
        else:
            return None

    def copy(self, root):
        """Copy function to create a similar
           matrix of the given node"""
        temp = []
        for i in root:
            t = []
            for j in i:
                t.append(j)
            temp.append(t)
        return temp
```

```
def process(self):
    "Accept start and goal puzzle state"
    print("Enter the start state matrix")
    start = self.f(start, goal)
    while True:
        cur = self.open[0]
        print("")
        print("|")
        print("|")
        for i in cur.data:
            for j in i:
                print(j, end="")
            print("")

puz = Puzzle(3)
puz.process()
```

output

Enter the start state matrix

1 2 3
0 4 6
7 5 8

Enter the goal state matrix

1 2 3
4 5 6
7 8 0

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