Program 3

8 puzzle using A* search

~~Class Node~~

Algorithm:-

1. We move the empty in all possible directions in the start state and calculate f-score for each state.

   This is called expanding current state

2. After expanding the current state, it is pushed into the closed list & the ~~new~~ newly generated states are pushed into the open list

3. A state with the least f-score is selected and expanded again

   f score = h score + g-score

   h → heuristic score

   g → no of nodes traversed from start node to current node

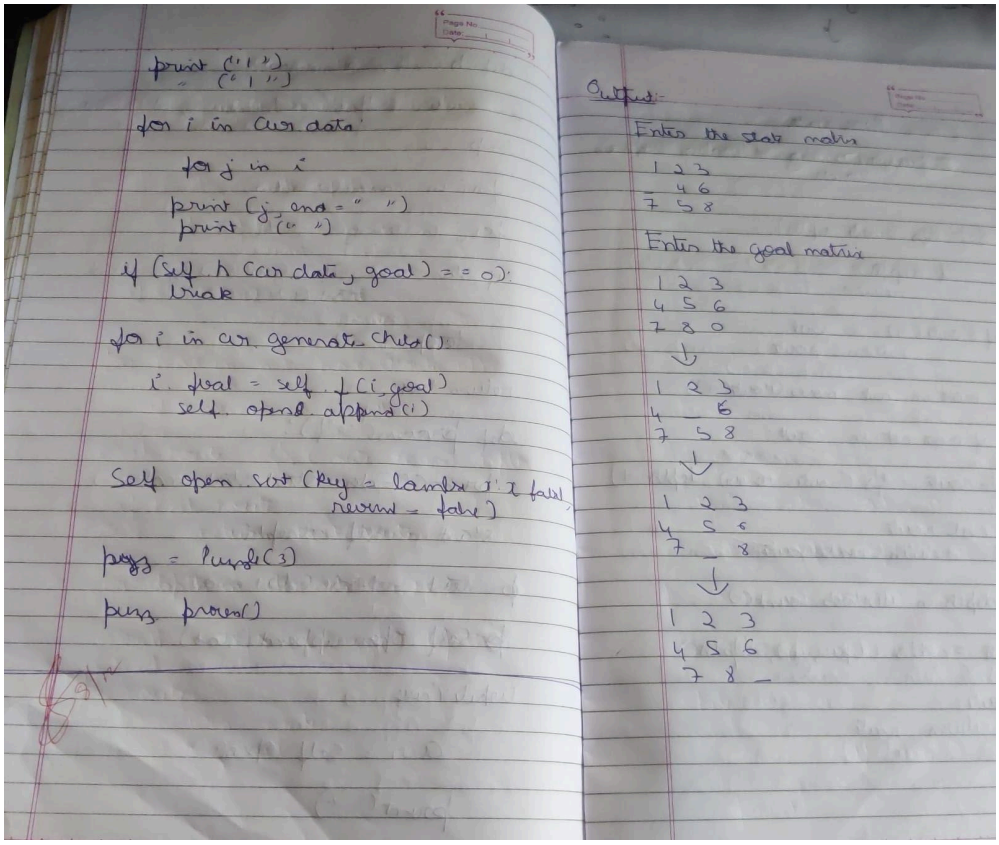
4. ~~Puzzle~~ There are 2 class - Node & Puzzle.

Node class - defines the structure of the state & also provides function to move empty space & generate child state from current state.

Puzzle class accepts initial & goal states of the N-Puzzle problem & provides function to calculate f score of any given node

```
class Node:

    def __init__(self, data, level, fval):

        self.data = data
        self.level = level
        self.fval = fval

    def generate_child(self):

        x, y = self.find(self.data, '_')

        val_list = [[x, y-1], [x, y+1], [x-1, y], [x+1, y]]

        children = []

        for i in val_list:

            child = self.shuffle(self.data, x, y, i[0], i[1])

            if (child
```

```
            if child is not None:

                child_node = Node(child, self.level+1, 0)
                children.append(child_node)

        return children

    def shuffle(self, puz, x1, y1, x2, y2):

        if x2 >= 0 & x2 < len(self.data) & y2 >= 0
            & y2 < len(self.data):

            temp_puz = []
            temp_puz = self.copy(puz)


            return temp_puz

        else:
            return None

    def copy(self, root):

        temp = []
        for i in root:
            t = []
            for j in i:
                t.append(j)
            temp.append(t)

        return temp
```

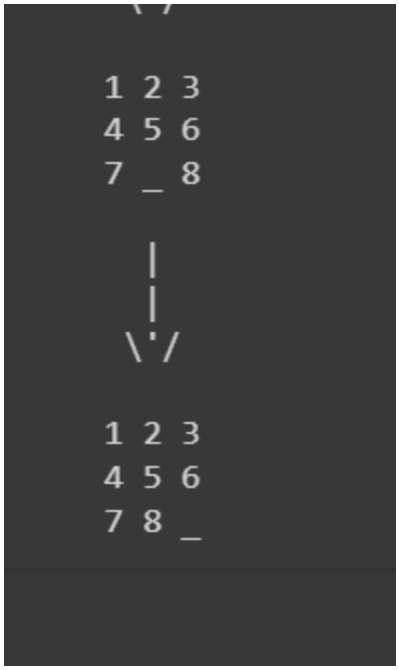


```
print ('|')
      ('|')
for i in cur_data:
    for j in i
    print (j, end = " ")
    print ('|')
if (self.h(cur_data, goal) == 0):
    break
for i in cur.generate_child():
    i.fval = self.f(i, goal)
    self.open.append(i)
self.open.sort(key = lambda x: x.fval,
               reverse = False)
puzz = Puzzle(3)
puzz.process()
```

Output:

```
Enter the start matrix
1 2 3
_ 4 6
7 5 8
Enter the goal matrix
1 2 3
4 5 6
7 8 0
|
1 2 3
4 _ 6
7 5 8
|
1 2 3
4 5 6
7 _ 8
|
1 2 3
4 5 6
7 8 _
```

```
Enter the start state matrix

1 2 3
4 5 6
_ 7 8
Enter the goal state matrix

1 2 3
4 5 6
7 8 _




  |
  |
 \'/

1 2 3
4 5 6
_ 7 8

  |
  |
 \'/

1 2 3
4 5 6
7 _ 8

  |
  |
 \'/
```


\'/
1 2 3
4 5 6
7 _ 8
|
|
\'/
1 2 3
4 5 6
7 8 _