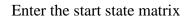
AI Practical: 2

Implement A star Algorithm for any game search problem.

```
class Node:
  def __init__(self, data, level, fval):
     self.data = data
     self.level = level
     self.fval = fval
  def generate_child(self):
     """ Generate child nodes from the given node by moving the blank space
       either in the four directions {up,down,left,right} """
     x, y = self.find(self.data, ' ')
     """ val list contains position values for moving the blank space in either of
       the 4 directions [up,down,left,right] respectively. """
     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 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 \ge 0 and x2 < len(self.data) and y2 \ge 0 and y2 < len(self.data):
       temp_puz = []
       temp_puz = self.copy(puz)
       temp = temp puz[x2][y2]
       temp_puz[x2][y2] = temp_puz[x1][y1]
       temp_puz[x1][y1] = 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 = \prod
       for i in i:
          t.append(j)
       temp.append(t)
     return temp
  def find(self, puz, x):
     """ Specifically used to find the position of the blank space """
     for i in range(0, len(self.data)):
       for j in range(0, len(self.data)):
          if puz[i][j] == x:
             return i, j
class Puzzle:
  def __init__(self, size):
     """ Initialize the puzzle size by the specified size, open and closed lists to empty """
     self.n = size
     self.open = []
```

```
self.closed = []
  def accept(self):
     """ Accepts the puzzle from the user """
     puz = []
     for i in range(0, self.n):
        temp = input().split(" ")
        puz.append(temp)
     return puz
  def f(self, start, goal):
      """ Heuristic Function to calculate hueristic value f(x) = h(x) + g(x) """
     return self.h(start.data, goal) + start.level
  def h(self, start, goal):
     """ Calculates the different between the given puzzles """
     temp = 0
     for i in range(0, self.n):
        for j in range(0, self.n):
           if start[i][j] != goal[i][j] and start[i][j] != '_':
              temp += 1
     return temp
  def process(self):
     """ Accept Start and Goal Puzzle state"""
     print("Enter the start state matrix \n")
     start = self.accept()
     print("Enter the goal state matrix \n")
     goal = self.accept()
     start = Node(start, 0, 0)
     start.fval = self.f(start, goal)
     """ Put the start node in the open list"""
     self.open.append(start)
     print("\n\n")
     while True:
        cur = self.open[0]
        print("")
        print(" | ")
        print(" | ")
        print(" \ \backslash\!\backslash\!\!\!\ \backslash \ \backslash\!\!\!\ \backslash \ \backslash\!\!\!\ )
        for i in cur.data:
           for i 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.closed.append(cur)
        del self.open[0]
        self.open.sort(key=lambda x: x.fval, reverse=False)
puz = Puzzle(3)
puz.process()
```

OUTPUT:



123

_46 7 5 8

Enter the goal state matrix

123

456

78_

\'/

123

 $\begin{smallmatrix} &4&6\\7&5&8\end{smallmatrix}$

\'/

123

4_6 758

\''/

123

4 5 6

7_8

\'/

123

4 5 6

78_

Process finished with exit code 0