







**Code:**

# Shubham - 311118

class Node:

    def \_\_init\_\_(self, data, level, fval) -> None:

        self.data = data

        self.level = level

        self.fval = fval

    def generate\_child(self):

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

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

        childrens = []

        for i in directions:

            child = self.move\_tile(self.data, x, y, i[0], i[1])

            if child is not None:

                child\_node = Node(child, self.level+1, 0)

                childrens.append(child\_node)

        return childrens

    def move\_tile(self, puz, x1, y1, x2, y2):

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

            tmp\_puz = []

            tmp\_puz = self.copy(puz)

            tmp = tmp\_puz[x2][y2]

            tmp\_puz[x2][y2] = tmp\_puz[x1][y1]

            tmp\_puz[x1][y1] = tmp

            return tmp\_puz

        else:

            return None

    # create cope of same node

    def copy(self, root):

        tmp = []

        for i in root:

            t = []

            for j in i:

                t.append(j)

            tmp.append(t)

        return tmp

    # find position of blank space

    def find(self, puz, x):

        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) -> None:

        self.n = size

        self.open = []

        self.closed = []

    def get\_input(self):

        puz = []

        for i in range(0, self.n):

            tmp = input().split(" ")

            puz.append(tmp)

        return puz

    def f(self, start, goal):

        return self.h(start.data, goal)+start.level

    def h(self, start, goal):

        tmp = 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] != '\_':

                    tmp += 1

        return tmp

    def process(self):

        print("Enter Start State:\n")

        start = self.get\_input()

        print()

        print("Enter Goal State:\n")

        goal = self.get\_input()

        print()

        start = Node(start, 0, 0)

        start.fval = self.f(start, goal)

        self.open.append(start)

        print("Solving..\n")

        while True:

            cur = self.open[0]

            for i in cur.data:

                for j in i:

                    print(j, end=" ")

                print()

            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)

        print("\nFinished!")

puz = Puzzle(3)

puz.process()

**Output:**

