```
class Solver:
    def __init__(self, size):
        self.size = size
        self.open = []
        self.closed = []

    def execute(self):
        print('Initial config:\n _ for blank)')
        start_matrix = self.accept_matrix()
        print('Goal config:\n')
        goal_matrix = self.accept_matrix()

        #convert start matrix to node

        start_node = pnode(start_matrix, 0, 0)

        #find f of start,append to open list
        start_node.f_value = self.find_f_value(start_node, goal_matr
ix)
        self.open.append(start_node)
        while len(self.open) != 0:
            current_node = self.open[0]
            current_node.print_node_info()
            if self.find_h_value(current_node.matrix, goal_matrix) =
= 0:
                break
            #move blank, find f and append to open
            for i in current_node.generate_child_nodes():
                if self.is_not_duplicate(i.matrix):
                    i.f_value = self.find_f_value(i, goal_matrix)
                    self.open.append(i)
            #add current to closed, delete from open
            self.closed.append(current_node)
            del self.open[0]
            self.open.sort(key = lambda node:node.f_value)
            print('-------------------')
        print('\nGoal config achieved, checked {} state spaces'.form
at(len(self.closed)+1))

    def accept_matrix(self):
        '''accepts square of given size'''
        temp = []
        for i in range(self.size):
            temp.append(input().split(" "))
        return temp

    def find_f_value(self, start_node, goal_matrix):
        '''finds heuristic value of a node'''
        return start_node.level + self.find_h_value(start_node.matri
x, goal_matrix)
```

```
    def find_h_value(self, start_matrix, goal_matrix):
        '''finds the difference between given puzzles'''
        diff = 0
        for i in range(self.size):
            for j in range(self.size):
                if start_matrix[i][j] != goal_matrix[i][j] and start
_matrix[i][j] != '_':
                    diff = diff+1
        return diff

    def is_not_duplicate(self, matrix):
        '''check if matrix exists in open or closed list'''
        for i in self.open:
            if i.matrix == matrix:
                return False
        for i in self.closed:
            if i.matrix == matrix:
                return False
        return True
n = int(input('Enter the size of matrix (square root of Solver size
+ 1) : '))
puzzle = Solver(n)
puzzle.execute()

class pnode:
    def __init__(self, matrix, level, f_value):
        self.matrix = matrix
        self.level = level
        self.f_value = f_value

    def print_node_info(self):
        print('\nLevel={},h={},f={}\nMatrix : \n'.format(self.level,
self.f_value-self.level, self.f_value))
        for i in self.matrix:
            for j in i:
                print(j, '\t', end="")
            print('\n')
        print('\n')

    def generate_child_nodes(self):
        #move blank to L,R,U,D
        x_blank, y_blank = self.get_blank()
        #find possible possitions, then moves
        pospos = []
        if x_blank-1 != -1:
            pospos.append([x_blank-1, y_blank])
        if x_blank+1 != len(self.matrix):
            pospos.append([x_blank+1, y_blank])
        if y_blank-1 != -1:
            pospos.append([x_blank, y_blank-1])
        if y_blank+1 != len(self.matrix):
```

```
            pospos.append([x_blank, y_blank+1])
        #generate children, swap all possible with blank
        children_nodes = []
        for new_pos in pospos:
            temp_matrix = self.duplicate(self.matrix)
            temp_matrix[x_blank][y_blank], temp_matrix[new_pos[0]][n
ew_pos[1]] = temp_matrix[new_pos[0]][new_pos[1]], temp_matrix[x_blan
k][y_blank]
            children_nodes.append(pnode(temp_matrix, self.level+1, 0
))
        return children_nodes

    def get_blank(self):
        for i in range(len(self.matrix)):
            for j in range(len(self.matrix)):
                if self.matrix[i][j] == '_':
                    return i, j

    def duplicate(self, matrix):
        temp = []
        for i in matrix:
            temp_row = []
            for j in i:
                temp_row.append(j)
            temp.append(temp_row)
        return temp
```

```
Enter the size of matrix (square root of Solver size + 1) : 3
Initial config:
 _ for blank)
_ 1 3
4 2 5
7 8 6
Goal config:

1 2 3
4 5 6
7 8 _

Level=0,h=4,f=4
Matrix :

_       1       3

4       2       5

7       8       6



-------------------

Level=1,h=3,f=4
Matrix :
```

```
1       _       3

4       2       5

7       8       6


-------------------

Level=2,h=2,f=4
Matrix :

1       2       3

4       _       5

7       8       6


-------------------

Level=3,h=1,f=4
Matrix :

1       2       3

4       5       _

7       8       6


-------------------

Level=4,h=0,f=4
Matrix :

1       2       3

4       5       6

7       8       _



Goal config achieved, checked 5 state spaces
```