Lab 1 Code

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

    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

    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 shuffle(self,puz,x1,y1,x2,y2):
        """ Move the blank space in the given direction and if the position value 
are out
            of limits the return None """
        if x2 >= 0 and x2 < len(self.data) and y2 >= 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 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,0)
        """ 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

class Puzzle:

    def __init__(self):
        # Initialize the open and closed lists to empty
        self.open = []
        self.closed = []

    def accept(self):
        # Accepts the puzzle from the user
        print('Enter the values of a row and press enter')
        puz = []
        for i in range(0,3):
            temp = list(map(int, input().split(" ")))
            puz.append(temp)
        return puz

    # def solvable(self, tiles):
    #     # Checks if the puzzle is solvable in this state
    #     count = 0
    #     for i in range(8):
    #         for j in range(i+1, 9):
    #             if tiles[j] and tiles[i] and tiles[i] > tiles[j]:
    #                 count += 1

        # return True if count%2==0 else False

    def h(self,start,goal):
        # Calculates the different between the given puzzles
        temp = 0
        for i in range(0,3):
            for j in range(0,3):
                if start[i][j] != goal[i][j] and start[i][j] != '0':
                    temp += 1
        return temp

    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 solve(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_combined = [item for sublist in start for item in sublist]
        # isSolvable = self.solvable(start_combined)
        # if (isSolvable == False):
        #     print("This puzzle is unsolvable")
        #     quit()

        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")
        iteration = 0
        while True:
            cur = self.open[0]
            print("\n")
            print("Step: {}".format(iteration))
            for i in cur.data:
                for j in i:
                    print(j,end=" ")
                print("")
            # If the difference between current and goal node is 0 we have reached
the goal node
            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]

            # sort the open list based on f value
            self.open.sort(key = lambda x:x.fval,reverse=False)
            iteration += 1


puzzle = Puzzle()
puzzle.solve()


'''
OUTPUT-

C:\Users\Divyang\Desktop\Final Year B.Tech\T9\AI\Lab 1>python a-star.py
Enter the start state matrix

Enter the values of a row and press enter
1 2 3
4 0 6
```

```
7 5 8
Enter the goal state matrix

Enter the values of a row and press enter
1 2 3
4 5 6
7 8 0





Step: 0
1 2 3
4 0 6
7 5 8


Step: 1
1 2 3
4 5 6
7 0 8


Step: 2
1 2 3
4 5 6
7 8 0
```

'''