def accept(n):

    puz = []

    for i in range(n):

        puz.append([val for val in input().split()])

    return puz

def print\_board(board,n):

    for i in range(n):

        print()

        for j in range(n):

            print(board[i][j],end=' ')

#Find the position of blank space

def find\_space(Current,n):

        for blank\_row\_pos in range(n):

            for blank\_col\_pos in range(n):

                if Current[blank\_row\_pos][blank\_col\_pos]=='\_':

                    return blank\_row\_pos,blank\_col\_pos

#Copy the current node to new node for shuffling the blank space and create a new configuration

def copy\_current(Current):

    temp=[]

    for i in range(len(Current)):

        row=[]

        for val in Current[i]:

            row.append(val)

        temp.append(row)

    return(temp)

#Move the blank space in given direction, if out of range return None

def shuffle(Current,brow\_pos,bcol\_pos,move\_x,move\_y):

    if move\_x >= 0 and move\_x < len(Current) and move\_y >= 0 and move\_y < len(Current):

        temp=[]

        temp=copy\_current(Current)

        change=temp[move\_x][move\_y]

        temp[move\_x][move\_y]=temp[brow\_pos][bcol\_pos]

        temp[brow\_pos][bcol\_pos]=change

        return temp

    else:

        return None

#Function to calculate g\_score: the number of nodes traversed from a start node to get to the current node

def g\_score(Node):

    return Node[1] #Node=[Board,level,fscore]

#Function to calculate h\_score: the number of misplaced tiles by comparing the current state and the goal state

def h\_score(Current,Goal,n):

    hscore=0

    for i in range(n):

        for j in range(n):

            if (Current[i][j] != Goal[i][j]) and (Current[i][j]!='\_'):

                hscore +=1

    return hscore

#Function to calculate f\_Score= g\_score + h\_Score

def f\_score(Node,Goal,n):

    Current=Node[0]

    return g\_score(Node) + h\_score(Current,Goal,n)

#Generate the child nodes by moving the blank in any four direction (up,down,left,right)

def move\_gen(Node,Goal,n):

    Current=Node[0]

    level=Node[1]

    fscore=0

    row,col=find\_space(Current,n)

    move\_positions=[[row,col-1],[row,col+1],[row-1,col],[row+1,col]] #left,right,up,down

    children=[] #List of child nodes of current node

    for move in move\_positions:

        child=shuffle(Current,row,col,move[0],move[1])

        if child is not None:

            cNode=[child,0,0] #Dummy node for calculating f\_Score

            fscore=f\_score(cNode,Goal,n)

            Node=[child,level+1,fscore]

            children.append(Node)

    print("\n\n The Children ::",children)

    return children

#Function goal\_test to see the goal configuration is reached

def goal\_test(Current,Goal,n):

    if h\_score(Current,Goal,n) == 0:

        return True

    else:

        return False

#Function to Sort OPEN based on f\_score

def sort(L):

    L.sort(key = lambda x: x[2],reverse=False)

    return L

#Function for starting the Game

def play\_game(Start, Goal, n):

    #when game starts

    fscore=0 #fscore initialized to zero

    gscore=0 #gscore initialized to zero

    level=0 #the start configuration is root node s at level-0 of the state space tree

    Node=[Start,level,fscore]

    fscore=f\_score(Node,Goal,n)

    #Every Node is [board configuration ,level,gscore]

    Node = [Start,level,fscore] # current node is Start node

    print("\nThe Node is=\n",Node)

    OPEN = [] #OPEN list as frontier

    CLOSED = [] #CLOSED as explored

    OPEN.append(Node)

    levelcount=0

    #Explored the current node to reach to the Goal configuration

    while True:

        N=OPEN[0] #first node of open

        del OPEN[0] # delete first node of open

        Current=N[0] #Extract board configuration

        print("\n\n The current configuration is ::",Current)

        CLOSED.append(N)

        #if goal configuration is reached terminate

        if goal\_test(Current,Goal,n) == True:

            print("\nGoal reached!!")

            print("CLOSED=",CLOSED)

            break

        CHILD=move\_gen(N,Goal,n)

        #print("\n\n The CHILD is ::",CHILD)

        OPEN=[]

        for child in CHILD:

            OPEN.append(child)

        #sort the OPEN list based on fscore value of each node

        sort(OPEN)

#Drive Code

n=int(input("Enter the board size:"))

print("\nEnter Start Configuration of board")

Start=accept(n)

print("\nEnter Goal Configuration of board")

Goal=accept(n)

play\_game(Start, Goal, n)