import copy

final = [[1,2,3],[4,5,6],[7,8,-1]]

initial = [[1,2,3],[-1,4,6],[7,5,8]]

#function to find heuristic cost

def gn(state, finalstate):

count = 0

for i in range(3):

for j in range(3):

if(state[i][j]!=-1):

if(state[i][j] != finalstate[i][j]):

count+=1

return count

def findposofblank(state):

for i in range(3):

for j in range(3):

if(state[i][j] == -1):

return [i,j]

def move\_left(state, pos):

if(pos[1]==0):

return None

retarr = copy.deepcopy(state)

retarr[pos[0]][pos[1]],retarr[pos[0]][pos[1]-1] = retarr[pos[0]][pos[1]-1],retarr[pos[0]][pos[1]]

return retarr

def move\_up(state, pos):

if(pos[0]==0):

return None

retarr = copy.deepcopy(state)

#for i in state:

#retarr.append(i)

retarr[pos[0]][pos[1]],retarr[pos[0]-1][pos[1]] = retarr[pos[0]-1][pos[1]],retarr[pos[0]][pos[1]]

return retarr

def move\_right(state, pos):

if(pos[1]==2):

return None

retarr = copy.deepcopy(state)

#for i in state:

#retarr.append(i)

retarr[pos[0]][pos[1]],retarr[pos[0]][pos[1]+1] = retarr[pos[0]][pos[1]+1],retarr[pos[0]][pos[1]]

return retarr

def move\_down(state, pos):

if(pos[0]==2):

return None

retarr = copy.deepcopy(state)

retarr[pos[0]][pos[1]],retarr[pos[0]+1][pos[1]] = retarr[pos[0]+1][pos[1]],retarr[pos[0]][pos[1]]

return retarr

def printMatrix(matricesArray):

print("")

counter = 1

for matrix in matricesArray:

print("Step {}".format(counter))

for row in matrix:

print(row)

counter+=1

print("")

def eightPuzzle(initialstate, finalstate):

hn=0

explored = []

while(True):

explored.append(initialstate)

if(initialstate == finalstate):

break

hn+=1

left = move\_left(initialstate, findposofblank(initialstate))

right = move\_right(initialstate, findposofblank(initialstate))

up = move\_up(initialstate, findposofblank(initialstate))

down = move\_down(initialstate, findposofblank(initialstate))

fnl=1000

fnr=1000

fnu=1000

fnd=1000

if(left!=None):

fnl = hn + gn(left,finalstate)

if(right!=None):

fnr = hn + gn(right,finalstate)

if(up!=None):

fnu = hn + gn(up,finalstate)

if(down!=None):

fnd = hn + gn(down,finalstate)

minfn = min(fnl, fnr, fnu, fnd)

if((fnl == minfn) and (left not in explored)):

initialstate = left

elif((fnr == minfn) and (right not in explored)):

initialstate = right

elif((fnu == minfn) and (up not in explored)):

initialstate = up

elif((fnd == minfn) and (down not in explored)):

initialstate = down

printMatrix(explored)

#eightPuzzle(initial, final)

def main():

while(True):

ch = int(input("PRESS 1 to continue and 0 to Exit : "))

if(not ch):

break

start = []

print("START STATE\n")

for i in range(3):

arr=[]

for j in range(3):

a = int(input("Enter element at {},{}: ".format(i,j)))

arr.append(a)

start.append(arr)

final = []

print("FINAL STATE\n")

for i in range(3):

arr=[]

for j in range(3):

a = int(input("Enter element at {},{}: ".format(i,j)))

arr.append(a)

final.append(arr)

eightPuzzle(start, final)

main()

**Output:**

PRESS 1 to continue and 0 to Exit : 1

START STATE

Enter element at 0,0: 1

Enter element at 0,1: 2

Enter element at 0,2: 3

Enter element at 1,0: 4

Enter element at 1,1: 5

Enter element at 1,2: 6

Enter element at 2,0: 7

Enter element at 2,1: 8

Enter element at 2,2: -1

FINAL STATE

Enter element at 0,0: 1

Enter element at 0,1: 2

Enter element at 0,2: 3

Enter element at 1,0: -1

Enter element at 1,1: 4

Enter element at 1,2: 6

Enter element at 2,0: 7

Enter element at 2,1: 5

Enter element at 2,2: 8

Step 1

[1, 2, 3]

[4, 5, 6]

[7, 8, -1]

Step 2

[1, 2, 3]

[4, 5, 6]

[7, -1, 8]

Step 3

[1, 2, 3]

[4, -1, 6]

[7, 5, 8]

Step 4

[1, 2, 3]

[-1, 4, 6]

[7, 5, 8]

PRESS 1 to continue and 0 to Exit :