#A\* 8 puzzle

g = 0

def print\_board(elements):

for i in range(9):

if i % 3 == 0:

print()

if elements[i] == -1:

print("\_", end=" ")

else:

print(elements[i], end=" ")

print()

##This function counts how many times a larger tile is before a smaller one.

##If the total count is even, the puzzle can be solved.

##If it’s odd, it cannot be solved.

def solvable(start):

inv = 0

for i in range(9):

if start[i] <= 1:

continue

for j in range(i + 1, 9):

if start[j] == -1:

continue

if start[i] > start[j]:

inv += 1

return inv % 2 == 0

##It tries to measure:“How many moves would each tile need to reach its goal spot?”

##This function calculates how far the puzzle is from the goal (based on tile positions).

##It uses Manhattan Distance to count how many steps each tile needs to reach its goal.

##Then it adds the number of moves already made (g) to give the total cost estimate f(n) for A\*.

def heuristic(start, goal):

global g

h = 0

for i in range(9):

for j in range(9):

if start[i] == goal[j] and start[i] != -1:

h += (abs(j - i)) // 3 + (abs(j - i)) % 3

return h + g

## moves the blank tile

def moveleft(start, position):

start[position], start[position - 1] = start[position - 1], start[position]

def moveright(start, position):

start[position], start[position + 1] = start[position + 1], start[position]

def moveup(start, position):

start[position], start[position - 3] = start[position - 3], start[position]

def movedown(start, position):

start[position], start[position + 3] = start[position + 3], start[position]

def movetile(start, goal):

emptyat = start.index(-1) ## find the position of the blank tile

row = emptyat // 3

col = emptyat % 3

t1, t2, t3, t4 = start[:], start[:], start[:], start[:] ##create 4 copies to try 4 directions

f1, f2, f3, f4 = 100, 100, 100, 100 ## initialize it with high heuristic value

if col - 1 >= 0:

moveleft(t1, emptyat)

f1 = heuristic(t1, goal)

if col + 1 < 3:

moveright(t2, emptyat)

f2 = heuristic(t2, goal)

if row + 1 < 3:

movedown(t3, emptyat)

f3 = heuristic(t3, goal)

if row - 1 >= 0:

moveup(t4, emptyat)

f4 = heuristic(t4, goal)

min\_heuristic = min(f1, f2, f3, f4) ## calculate the min heristic to reach the goal

if f1 == min\_heuristic:

moveleft(start, emptyat)

elif f2 == min\_heuristic:

moveright(start, emptyat)

elif f3 == min\_heuristic:

movedown(start, emptyat)

elif f4 == min\_heuristic:

moveup(start, emptyat)

def solveEight(start, goal):

global g ## represents g(n)..calculates how many moves have been made till now

g += 1

movetile(start, goal) ## makes the best move based on the previous logic

print\_board(start)

f = heuristic(start, goal) ##it calculates the current total cost

if f == g:

return

solveEight(start, goal)

def main():

print("Name: Prachi Karande")

print("Roll no.: TACO22134")

global g

start = []

goal = []

print("Enter the start state:(Enter -1 for empty):")

for i in range(9):

start.append(int(input()))

print("Enter the goal state:(Enter -1 for empty):")

for i in range(9):

goal.append(int(input()))

print\_board(start)

# To check if solvable

if solvable(start):

solveEight(start, goal)

print("Solved in {} moves".format(g))

else:

print("Not possible to solve")

if \_\_name\_\_ == '\_\_main\_\_':

main()