A\* algorithm

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()

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

if inv%2==0:

return True

return False

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

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)

row = emptyat//3

col = emptyat%3

t1,t2,t3,t4 = start[:],start[:],start[:],start[:]

f1,f2,f3,f4 = 100,100,100,100

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)

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

g+=1

movetile(start,goal)

print\_board(start)

f = heuristic(start,goal)

if f == g:

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

return

solveEight(start,goal)

def main():

global g

start = list()

goal = list()

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()

theory

8-Puzzle Problem Overview:

The 8-puzzle consists of a 3x3 board with 8 numbered tiles and one empty space (represented by -1).

The goal is to move the tiles by sliding them into the empty space to reach a specified final configuration.

🔹 Key Concepts Used in the Code:

1. State Representation:

The board is represented as a list of 9 elements, where each index corresponds to a cell in the 3x3 grid.

-1 is used to represent the empty space.

2. Solvability Check:

A function solvable() checks if the initial state is solvable by counting inversions.

If the number of inversions is even, the puzzle is solvable.

3. Heuristic Function:

The heuristic() function calculates the Manhattan Distance:

It adds the row and column distance of each tile from its goal position.

This is a common and effective heuristic in solving sliding puzzles.

4. Tile Movement Functions:

moveleft, moveright, moveup, and movedown functions swap the empty space with the adjacent tile in the given direction.

Boundary checks ensure valid moves (e.g., no left move from column 0).

5. Heuristic-Based Move Selection:

movetile() generates all possible next states and selects the one with the lowest heuristic value.

It tries to make the best local move toward the goal (a greedy approach).

6. Recursive Solver:

solveEight() calls movetile() recursively, updating the board and increasing the move count g.

The recursion continues until the heuristic value equals the number of moves, indicating the goal is reached.

🔹 Limitations:

The algorithm uses greedy choice (selects move with the best heuristic at each step), so it may not guarantee the shortest path.

It does not use backtracking or a visited set, which can lead to loops in some cases.

Output:

The program prints each board configuration at every move.

Finally, it shows the number of moves taken to reach the goal, or says it's unsolvable.