1. Write the python program to solve 8-Puzzle problem

**AIM :** program to solve 8-Puzzle problem

**ALGORITHM :**

1.Define the initial state and goal state.

2. Create an open list to store states yet to be explored, initially containing the initial state.

3. Create a closed set to store states that have been explored.

4. Initialize the **g\_score** and **f\_score** dictionaries for each state.

5. While the open list is not empty:

Pop the state with the lowest **f\_score** value from the open list.

If the current state is the goal state, reconstruct and return the solution path.

Add the current state to the closed set.

Generate possible moves from the current state (swap the empty tile with neighboring tiles).

For each possible move:

Calculate the tentative **g\_score** for the new state.

If the new state is already in the closed set and the tentative **g\_score** is greater or equal, skip it.

If the new state is not in the open list or the tentative **g\_score** is lower:

Update the **came\_from** dictionary to store the parent state.

Update the **g\_score** and **f\_score** for the new state.

Add the new state to the open list.

6. If the open list becomes empty and the goal state hasn't been reached, there is no solution.

**PROGRAM :**

import heapq

goal\_state = (1, 2, 3, 4, 5, 6, 7, 8, 0)

moves = [(0, 1), (0, -1), (1, 0), (-1, 0)]

def get\_neighbors(state):

neighbors = []

empty\_index = state.index(0)

empty\_row, empty\_col = empty\_index // 3, empty\_index % 3

for dr, dc in moves:

new\_row, new\_col = empty\_row + dr, empty\_col + dc

if 0 <= new\_row < 3 and 0 <= new\_col < 3:

neighbor\_state = list(state)

neighbor\_index = new\_row \* 3 + new\_col

neighbor\_state[empty\_index], neighbor\_state[neighbor\_index] = neighbor\_state[neighbor\_index], neighbor\_state[empty\_index]

neighbors.append(tuple(neighbor\_state))

return neighbors

def manhattan\_distance(state):

distance = 0

for i in range(9):

if state[i] != 0:

current\_row, current\_col = i // 3, i % 3

target\_row, target\_col = (state[i] - 1) // 3, (state[i] - 1) % 3

distance += abs(current\_row - target\_row) + abs(current\_col - target\_col)

return distance

def solve\_puzzle(initial\_state):

open\_list = [(manhattan\_distance(initial\_state), initial\_state)]

closed\_set = set()

while open\_list:

current\_state = heapq.heappop(open\_list)[1]

if current\_state == goal\_state:

return current\_state

closed\_set.add(current\_state)

for neighbor in get\_neighbors(current\_state):

if neighbor not in closed\_set:

heapq.heappush(open\_list, (manhattan\_distance(neighbor), neighbor))

return None

initial\_state = (1, 0, 3, 4, 2, 5, 7, 8, 6)

solution = solve\_puzzle(initial\_state)

if solution:

print("Solution found:")

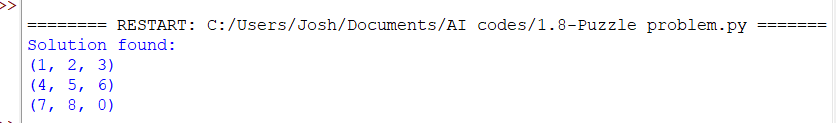
for i in range(0, 9, 3):

print(solution[i:i+3])

else:

print("No solution found.")

**OUT PUT :**

****