EXPERIMENT 1:

DATE: 01.03.2024

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

**Aim:**

The aim is to perform and solve the 8puzzle problem using python program.

**PROGRAM:**

import heapq

class PuzzleState:

def \_\_init\_\_(self, board, parent, move, depth, cost):

self.board = board

self.parent = parent

self.move = move

self.depth = depth

self.cost = cost

def \_\_lt\_\_(self, other):

return self.cost < other.cost

def \_\_eq\_\_(self, other):

return self.board == other.board

def print\_solution(node):

path = []

while node is not None:

path.append(node.board)

node = node.parent

path.reverse()

for state in path:

for row in state:

print(" ".join(map(str, row)))

print()

def is\_valid(x, y):

return 0 <= x < 3 and 0 <= y < 3

def get\_neighbors(state):

neighbors = []

x, y = None, None

for i in range(3):

for j in range(3):

if state[i][j] == 0:

x, y = i, j

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

for dx, dy in directions:

new\_x, new\_y = x + dx, y + dy

if is\_valid(new\_x, new\_y):

neighbor = [row[:] for row in state]

neighbor[x][y], neighbor[new\_x][new\_y] = neighbor[new\_x][new\_y], neighbor[x][y]

neighbors.append(neighbor)

return neighbors

def heuristic(state, goal):

h = 0

for i in range(3):

for j in range(3):

if state[i][j] != goal[i][j]:

h += 1

return h

def solve\_puzzle(initial\_state, goal\_state):

open\_set = []

closed\_set = set()

start\_node = PuzzleState(initial\_state, None, None, 0, 0)

heapq.heappush(open\_set, start\_node)

while open\_set:

current\_node = heapq.heappop(open\_set)

if current\_node.board == goal\_state:

print("Solution found!")

print\_solution(current\_node)

return

closed\_set.add(tuple(map(tuple, current\_node.board)))

neighbors = get\_neighbors(current\_node.board)

for neighbor in neighbors:

if tuple(map(tuple, neighbor)) not in closed\_set:

move = current\_node.move

if move is not None:

move += 1

new\_cost = current\_node.cost + 1

new\_depth = current\_node.depth + 1

new\_node = PuzzleState(neighbor, current\_node, move, new\_depth, new\_cost + heuristic(neighbor, goal\_state))

heapq.heappush(open\_set, new\_node)

print("No solution found.")

if \_\_name\_\_ == "\_\_main\_\_":

initial\_state = [

[1, 2, 3],

[8, 0, 4],

[7, 6, 5]

]

goal\_state = [

[1, 2, 3],

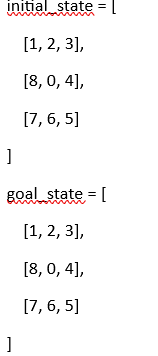
[8, 0, 4],

[7, 6, 5]

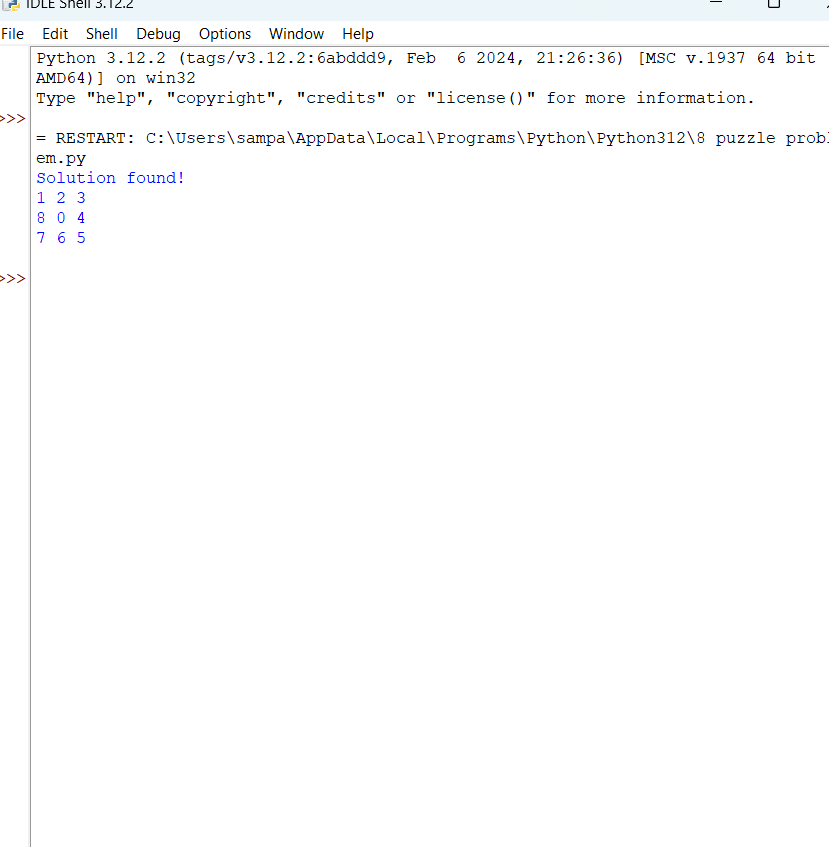
]

solve\_puzzle(initial\_state, goal\_state)

**INPUT:**

****

**OUTPUT:**

****

**RESULT:**

8puzzle using python program executed successfully.