**Practical: 2**

**Aim: Write a program to solve the 8-puzzle. Starting with any input state it should display the sequence of**

**movies to reach the ordered output state.**

**Rule to move: Slide a tile from a non-empty square to adjoining empty square**

**Example:**

|  |  |  |
| --- | --- | --- |
| **1** | **5** | **7** |
| **0** | **6** | **8** |
| **3** | **2** | **4** |

**Input state**

|  |  |  |
| --- | --- | --- |
| **1** | **2** | **3** |
| **4** | **5** | **6** |
| **7** | **8** | **0** |

**Output state**

**Program:**

from collections import deque

def print\_matrix(matrix):

for row in matrix:

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

print()

def find\_empty\_tile(puzzle):

for i in range(len(puzzle)):

for j in range(len(puzzle[i])):

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

return i, j

def get\_neighbors(x, y, n):

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

neighbors = []

for dx, dy in moves:

nx, ny = x + dx, y + dy

if 0 <= nx < n and 0 <= ny < n:

neighbors.append((nx, ny))

return neighbors

def solve\_puzzle(initial, goal):

n = len(initial)

start = tuple(tuple(row) for row in initial)

goal = tuple(tuple(row) for row in goal)

queue = deque([(start, find\_empty\_tile(initial), [])])

visited = set()

visited.add(start)

while queue:

current, (x, y), path = queue.popleft()

if current == goal:

for step in path:

print\_matrix(step)

print("Solved!")

return

for nx, ny in get\_neighbors(x, y, n):

new\_puzzle = [list(row) for row in current]

new\_puzzle[x][y], new\_puzzle[nx][ny] = new\_puzzle[nx][ny], new\_puzzle[x][y]

new\_puzzle\_tuple = tuple(tuple(row) for row in new\_puzzle)

if new\_puzzle\_tuple not in visited:

visited.add(new\_puzzle\_tuple)

queue.append((new\_puzzle\_tuple, (nx, ny), path + [new\_puzzle]))

print("No solution exists.")

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

initial = [

[1, 5, 7],

[0, 6, 8],

[3, 2, 4],

]

goal = [

[1, 2, 3],

[4, 5, 6],

[7, 8, 0],

]

solve\_puzzle(initial, goal)

**Output:**

0 5 7

1 6 8

3 2 4

5 0 7

1 6 8

3 2 4

5 6 7

1 0 8

3 2 4

5 6 7

1 2 8

3 0 4

5 6 7

1 2 8

0 3 4

5 6 7

0 2 8

1 3 4

5 6 7

2 0 8

1 3 4

5 6 7

2 3 8

1 0 4

5 6 7

2 3 8

1 4 0

5 6 7

2 3 0

1 4 8

5 6 0

2 3 7

1 4 8

5 0 6

2 3 7

1 4 8

5 3 6

2 0 7

1 4 8

5 3 6

2 7 0

1 4 8

5 3 0

2 7 6

1 4 8

5 0 3

2 7 6

1 4 8

0 5 3

2 7 6

1 4 8

2 5 3

0 7 6

1 4 8

2 5 3

1 7 6

0 4 8

2 5 3

1 7 6

4 0 8

2 5 3

1 0 6

4 7 8

2 0 3

1 5 6

4 7 8

0 2 3

1 5 6

4 7 8

1 2 3

0 5 6

4 7 8

1 2 3

4 5 6

0 7 8

1 2 3

4 5 6

7 0 8

1 2 3

4 5 6

7 8 0

Solved!