PROGRAM-1

8 PUZZLE PROBLEM

AIM:-

To write and execute the python program for the 8 puzzle program.

PROCEDURE:-

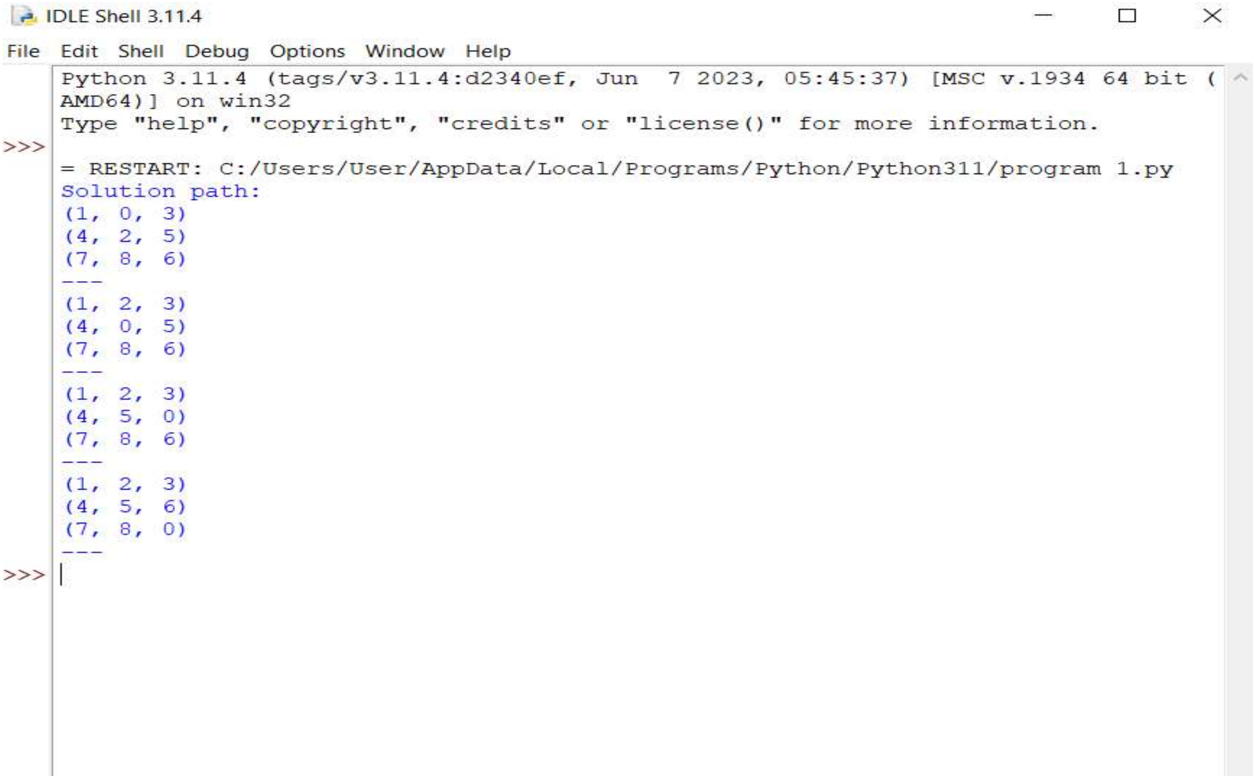
- **Define Goal State and Moves:**
  - The goal variable represents the goal state of the puzzle.
  - The moves list contains possible moves: 1 (right), -1 (left), 3 (down), and -3 (up).
- **Initialize Data Structures**
- **Main Loop:**
  - While open_list is not empty:
    - Pop the state with the lowest cost from the open_list.
    - If the current state is the goal state, return the path.
    - Add the current state to the closed_set to mark it as visited.
- **Generate Successors:**
  - Iterate over possible moves from the empty space and generate successor states.
  - Calculate the cost of each successor and add it to the open_list if it's not in the closed_set.
- **Return Solution Path:**
  - If a solution is found, print the solution path by iterating over the steps and printing each state.
- **Print State Function:**
  - The print_state function prints the state of the puzzle in a readable format.
- **Example Usage:**
  - Define the initial state of the puzzle.
  - Call the solve_puzzle function with the initial state to find the solution path.
  - Print the solution path if a solution is found; otherwise, print "No solution found.

CODING:-

```
import heapq

def solve_puzzle(initial):
    goal = (1, 2, 3, 4, 5, 6, 7, 8, 0)
    moves = [1, -1, 3, -3]
    open_list, closed_set = [(0, initial, [])], set()

    while open_list:
        _, current, path = heapq.heappop(open_list)
        if current == goal:
            return path + [current]
        closed_set.add(current)

        empty = current.index(0)
        for m in moves:
            if 0 <= empty // 3 + m // 3 < 3 and m + empty in range(9):  # Check valid moves considering edges
                neighbor = list(current)
                neighbor[empty], neighbor[empty + m] = neighbor[empty + m], neighbor[empty]
                neighbor_tuple = tuple(neighbor)
                if neighbor_tuple not in closed_set:
                    heapq.heappush(open_list, (len(path) + 1, neighbor_tuple, path + [current]))

    return []
```

```
def print_state(state):
    for i in range(0, 9, 3):
        print(state[i:i+3])
    print('---')

initial_state = (1, 0, 3, 4, 2, 5, 7, 8, 6)
solution_path = solve_puzzle(initial_state)
if solution_path:
    print("Solution path:")
    for step in solution_path:
        print_state(step)
else:
    print("No solution found.")
```

OUTPUT:-



```
Python 3.11.4 (tags/v3.11.4:d2340ef, Jun  7 2023, 05:45:37) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/User/AppData/Local/Programs/Python/Python311/program 1.py
Solution path:
(1, 0, 3)
(4, 2, 5)
(7, 8, 6)
---
(1, 2, 3)
(4, 0, 5)
(7, 8, 6)
---
(1, 2, 3)
(4, 5, 0)
(7, 8, 6)
---
(1, 2, 3)
(4, 5, 6)
(7, 8, 0)
---
>>>
```

RESULT:-

Hence the program has been successfully executed and verified.