Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Department of Computer Science and Engineering (Data Science)

Subject: Intelligent Systems Laboratory (DJS23DLPC503)

AY: 2025-26

Experiment 1

(Problem Solving)

Name : Soham Walam **Roll No : D102** **SAP ID : 6009230094**

Aim: Implement domain specific functions for given problems required for problem solving.

Theory:

There are two domain specific functions required in all problem-solving methods.
1. GoalTest Function:

goalTest(State) Returns *true* if the input state is the goal state and *false* otherwise.

goalTest(State, Goal) Returns *true* if *State* matches *Goal,* and *false* otherwise.

2. MoveGen function:

```
Initialize set of successors C to empty set.
Add M to the complement of given state N to get new state S.
If given state has Left, then add Right to S, else add Left.
If legal(S) then add S to set of successors C.
For each other-entity E in N
    make a copy S' of S,
    add E to S',
    If legal (S'), then add S' to C.
Return (C).
```

Lab Assignment to do:
Create MoveGen and GoalTest Functions for the given problems
1. **Water Jug Problem**
There are two jugs available of different volumes such as a 3 litres and a 7 litres and you have to measure a different volume such as 6 litre.

MoveGen:

```
def water_jug_movegen(state, capacities=(3, 7)):
  x, y = state
```

```
    moves = []
    moves.append((capacities[0], y))
    moves.append((x, capacities[1]))

    moves.append((0, y))
    moves.append((x, 0))
    pour = min(x, capacities[1] - y)
    moves.append((x - pour, y + pour))
    pour = min(y, capacities[0] - x)
    moves.append((x + pour, y - pour))
    return set(moves)
```

GoalTest:

```
def water_jug_goaltest(state, goal=6):
    return goal in state
```

```
Water Jug Problem
Path to goal: [(0, 0), (3, 0), (0, 3), (3, 3), (0, 6)]
```

2. **Travelling Salesman Problem**

A salesman is travelling and selling his/her product to in different cities. The condition is that it has to travel each city just once.

MoveGen:

```
def tsp_movegen(cities):
    return list(itertools.permutations(cities))
```

GoalTest:

```
def tsp_goaltest(path, start_city):
    return path[0] == start_city and path[-1] == start_city and len(path) == len(set(path))
```

```
Travelling Salesman Problem
Best path: ['A', 'B', 'D', 'C', 'A'] with cost: 80
```

3**. 8 Puzzle Problem**

An initial state is given in an 8 puzzles where one place is blank out of 9 places. You can shift this blank space and get a different state to reach to a given goal state.

MoveGen:

```
def puzzle_movegen(state):
    moves = []
    size = 3
    zero_index = state.index(0)
    row, col = divmod(zero_index, size)
    directions = {
        "up": (row - 1, col),
        "down": (row + 1, col),
        "left": (row, col - 1),
        "right": (row, col + 1),
    }
    for (r, c) in directions.values():
        if 0 <= r < size and 0 <= c < size:
            new_index = r * size + c
            new_state = list(state)
            new_state[zero_index], new_state[new_index] = new_state[new_index], new_state[zero_index]
            moves.append(tuple(new_state))
    return moves
```

GoalTest:

```
def puzzle_goaltest(state, goal=(1,2,3,4,5,6,7,8,0)):
    return state == goal
```

```
8 Puzzle Problem
Steps to goal:
(1, 2, 3, 4, 0, 5, 6, 7, 8)
(1, 2, 3, 4, 5, 0, 6, 7, 8)
(1, 2, 3, 4, 5, 8, 6, 7, 0)
(1, 2, 3, 4, 5, 8, 6, 0, 7)
(1, 2, 3, 4, 5, 8, 0, 6, 7)
(1, 2, 3, 0, 5, 8, 4, 6, 7)
(1, 2, 3, 5, 0, 8, 4, 6, 7)
(1, 2, 3, 5, 6, 8, 4, 0, 7)
(1, 2, 3, 5, 6, 8, 4, 7, 0)
(1, 2, 3, 5, 6, 0, 4, 7, 8)
(1, 2, 3, 5, 0, 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)
```