Artificial Intelligence (18CSC305J)

Faculty : Helen Victoria A

Ex- 3 : Team Tesla 2.0

Name	Regno.	Email Id
Abhighyan B	RA1911033010002	ab2134@srmist.edu.in
Sanjana N B	RA1911033010016	sn8740@srmist.edu.in
Roehit Ranganathan	RA1911033010017	rr9344@srmist.edu.in
Venkata Naga Sai Ram Nomula	RA1911033010021	vn4903@srmist.edu.in
K. Dushyant Reddy	RA1911033010029	kr7119@srmist.edu.in

Experiment 3 - Implementation of Constraint Satisfaction Problem

Problem Statement:

The purpose is to implement the Constraint Satisfaction Problem which has been developed using Solving **Sudoku** algorithm in order to reduce the computing time and utilize lower memory space.

Algorithm:

1. Sudoku puzzle involves a 9x9 matrix of squares sub-divided into a 3x3 grid.
2. Every submatrix has to contain a single number from [1-9] to be used.
3. Using backtracking algorithm we have two termination conditions:
    a. Board is already filled (No white space).
    b. No empty area left to check and the current user doesn't reach our goal.
4. Simultaneously, use the numbers to fill in to reveal more squares.
5. Repeat until the Sudoku is solved.

Code:

```
N = 9

def printmatrix(arr):
    for i in range(N):
        for j in range(N):
            print(arr[i][j], end=" ")
        print()


def isValid(matrix, row, col, num):

    for x in range(9):
        if matrix[row][x] == num:
            return False

    for x in range(9):
        if matrix[x][col] == num:
            return False

    startRow = row - row % 3
    startCol = col - col % 3
    for i in range(3):
        for j in range(3):
            if matrix[i + startRow][j + startCol] == num:
                return False
    return True


def solve(matrix, row, col):

    if (row == N - 1 and col == N):
        return True

    if col == N:
        row += 1
        col = 0

    if matrix[row][col] > 0:
        return solve(matrix, row, col + 1)
    for num in range(1, N + 1, 1):

        if isValid(matrix, row, col, num):
```

```
            matrix[row][col] = num

            if solve(matrix, row, col + 1):
                return True

        matrix[row][col] = 0
    return False


matrix = [[0, 6, 0, 4, 0, 0, 0, 7, 0],
          [0, 8, 0, 0, 0, 0, 0, 2, 9],
          [0, 7, 0, 0, 2, 0, 5, 0, 0],
          [0, 0, 5, 6, 0, 0, 0, 0, 4],
          [9, 0, 0, 0, 0, 0, 0, 0, 0],
          [0, 0, 0, 5, 0, 0, 0, 0, 3],
          [0, 0, 4, 1, 0, 0, 0, 0, 0],
          [8, 0, 0, 0, 9, 0, 0, 0, 0],
          [0, 0, 0, 0, 8, 0, 1, 0, 6]]

if (solve(matrix, 0, 0)):
    printmatrix(matrix)
else:
    print("no solution  exists ")
```

Output:

```
PS D:\SRM\SEM 6\AI lab\EXP-3> python -u "d:\SRM\SEM 6\AI lab\EXP-3\exp3.py"
2 6 9 4 1 5 3 7 8
5 8 1 7 6 3 4 2 9
4 7 3 9 2 8 5 6 1
1 3 5 6 7 2 9 8 4
9 4 6 8 3 1 2 5 7
7 2 8 5 4 9 6 1 3
6 9 4 1 5 7 8 3 2
8 1 2 3 9 6 7 4 5
3 5 7 2 8 4 1 9 6
PS D:\SRM\SEM 6\AI lab\EXP-3>
```

Real World Solution:

- Location: building facilities and supplying every customer with the minimal total cost
- Job shop scheduling: processing all jobs following the capacity and route constraints with the minimal makespan
- Car sequencing: sequencing the cars on the assembly line so that no workstation capacity is exceeded
- Cutting stock: using different cutting patterns to meet the demand of different-shaped pieces with the minimal cost

Result: Constraint Satisfaction Problem was successfully implemented.