Experiment:- 6

Objective:- Solve Constraint Satisfaction Problem.

Theory:-

A **Constraint Satisfaction Problem (CSP)** is a mathematical framework used to solve problems where a set of variables must be assigned values that satisfy specific constraints. CSPs are widely applied in Artificial Intelligence (AI) for tasks like scheduling, planning, resource allocation, and puzzle solving.

Key Components of CSP

1. **Variables:** These represent the unknowns that need to be assigned values. For example, in Sudoku, each cell is a variable.
2. **Domains:** Each variable has a domain, which is the set of possible values it can take. For instance, in Sudoku, the domain for each cell is {1, 2, ..., 9}.
3. **Constraints**: These are rules that restrict the values variables can take.

   Constraints can be unary (affecting one variable), binary (involving two variables), or higher-order (involving multiple variables). For example, in Sudoku, no two cells in the same row can have the same value.

Solving CSPs

CSPs are solved using algorithms that explore the search space while ensuring constraints are met. Common techniques include:

1. **Backtracking**: A depth-first search approach that assigns values to variables and backtracks when a constraint is violated.
2. **Forward Checking**: Enhances backtracking by eliminating inconsistent values from the domains of unassigned variables after each assignment.
3. **Constraint Propagation**: Enforces local consistency by propagating constraints across variables, reducing the search space.

Types of Constraint Satisfaction Problems

CSPs can be classified into different types based on their constraints and problem characteristics:

1. **Binary CSPs**:In these problems each constraint involves only two variables. Like in scheduling problem constraints could specify that task A must be completed before task B. 2. **Non-Binary CSPs:** These problems have constraints that involve more than two variables. For instance in a seating arrangement problem a constraint could state that three people cannot sit next to each other.

3. **Hard and Soft Constraints**: Hard constraints must be strictly satisfied while soft constraints can be violated but at a certain cost. This is often used in real-world applications where not all constraints are equally important

Program:-

Solving Sudoku with Constraint Satisfaction Problem (CSP)

```
class SudokuSolver:

  def ____init__(self, sudoku):
     self.sudoku = sudoku
     self.n = len(sudoku)
     self.sqrt_n=int(self.n ** 0.5)

  def solve_sudoku(self):
     if self.solve():
        return self.sudoku
     else:
        return None

  def solve(self):
     row, col = self.find_unassigned_space()
     if row
```

```
                    -1 and col
                                    -1:
            return True # Puzzle solved

        for num in range(1, self.n + 1):
            if self.is safe(row, col, num):
    self.sudoku[row][col] = num if
                self.solve():
                    return True
                self.sudoku[row][col] = 0 # Backtrack

        return False # Trigger backtracking

    def find_unassigned_space(self):
        for row in range(self.n):
            for col in range(self.n):
                if self.sudoku[row][col] == 0:
                    return row, col
        return -1, -1

    def is_safe(self, row, col, num):
        #Row & Column check
        for i in range(self.n):
            if self.sudoku[row][i] == num or self.sudoku[i][col]
                return False
                                                        == num:
```

```
        # Subgrid check
subgrid_row_start = (row // self.sqrt_n) * self.sqrt_n
        subgrid_col_start = (col // self.sqrt_n) * self.sqrt_n


        for i in range(subgrid_row_start, subgrid_row_start + self.sqrt_n):
            for j in range(subgrid_col_start, subgrid_col_start + self.sqrt_n):
                if self.sudoku[i][j]
                                                   == num:
                    return False


        return True


# Initial Sudoku Puzzle (0 means unfilled)
sudoku = [
                                    6],
    [0, 0, 0, 0, 0, 0, 0, 7, [7, 0,
    2, 9, 6, 0, 4, 5, 1],
]
    [5, 3, 0, 0, 7, 1, 2, 8, 0],
    [0, 7, 0, 0, 0, 6, 9, 2, 0],
    [0, 0, 3, 8, 9, 0, 0, 4, 7],
    [6, 0, 0, 0, 0, 0, 0, 1, 0],
    [0, 0, 0, 6, 8, 2, 0, 0, 0],
    [3, 6, 0, 5, 1, 4, 0, 9, 0],
    [4, 2, 8, 7, 0, 9, 1, 0, 0]


# Solve it
solver = SudokuSolver(sudoku)
solution = solver.solve_sudoku()

# Display result
if solution:
    print("Sudoku solution:")
    for row in solution:


else:
        print(row)


    print("No solution exists.")
```

Output:-

```
Sudoku solution:
```

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

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

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

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

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

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

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

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

VIKAS KATARE EN23CS3011131 5TH SEM