**Assignment 2**

**Title:**

Solving a Constraint Satisfaction Problem (CSP) Using Backtracking and Forward Checking

**Aim:**

To solve a constraint satisfaction problem (CSP) using Backtracking and Forward Checking algorithms and evaluate the performance of these techniques for consistency enforcement and optimization.

**Objectives:**

1. To understand the concept of Constraint Satisfaction Problems (CSPs).
2. To implement CSP solving techniques, including Backtracking and Forward Checking.
3. To explore how constraints can be applied to variables to find valid solutions.
4. To compare the performance of Backtracking with and without Forward Checking.
5. To apply CSP solving techniques to practical problems like graph colouring or Sudoku solving.

**Theory:**

**Constraint Satisfaction Problem (CSP):**

A **CSP** is a mathematical problem defined by:

* Variables: A set of variables, each of which must be assigned a value.
* Domains: Each variable has a domain, which is the set of possible values it can take.
* Constraints: A set of constraints that restrict the values that the variables can simultaneously take.

The goal is to assign values to all the variables such that all the constraints are satisfied. CSPs are widely applicable to problems like Sudoku, graph colouring, job scheduling, etc.

Example of a CSP (Graph Colouring Problem):

* Variables: The nodes of a graph.
* Domains: The possible colors for each node.
* Constraints: Adjacent nodes must have different colors.

The objective is to assign a color to each node so that no two adjacent nodes have the same color.

**Techniques to Solve CSPs:**

1. **Backtracking Algorithm:**

* This is a brute-force search algorithm where each variable is assigned a value, and the algorithm backtracks whenever a constraint is violated.
* It is a depth-first search approach that explores every possible variable assignment until a solution is found or all possibilities are exhausted.

1. **Forward Checking:**

* Forward checking is an improvement to backtracking where after assigning a value to a variable, the algorithm immediately checks the remaining unassigned variables to see if any domain becomes empty. This helps prune the search space early, making the algorithm more efficient.

**Procedure:**

1. **Define the problem:**

* Select a specific CSP to solve (e.g., graph coloring, Sudoku, n-queens).
* Define the variables, domains, and constraints for the chosen problem.
* For graph colouring:
* Variables: The nodes of the graph.
* Domains: A list of colors.
* Constraints: Adjacent nodes cannot share the same color.

1. **Implement Backtracking Algorithm:**

* Create a backtracking function that iteratively assigns a value to each variable.
* If all variables are assigned and no constraints are violated, return the solution.
* If a constraint is violated, backtrack and try a different assignment.
* Continue until either a solution is found or all possibilities are exhausted.

1. **Implement Forward Checking Algorithm:**

* After assigning a value to a variable, immediately check whether the assignment violates any constraints for the remaining unassigned variables.
* If any variable’s domain becomes empty, backtrack and try a different assignment.
* This early pruning helps reduce the number of invalid variable assignments explored during the search.

1. **Performance Comparison:**

* Run the CSP solver using backtracking alone and record the time taken and the number of recursive calls made.
* Run the solver using forward checking with backtracking and compare the results in terms of:
* Time taken to find the solution.
* Number of recursive calls (representing the search effort).
* Analyse the impact of forward checking in reducing the search space.

Procedure for Example Problem (Graph Coloring Problem):

1. Define the graph structure with nodes and edges.
2. Define the number of colors available to color the graph.
3. Set constraints: adjacent nodes should not have the same color.
4. Apply backtracking to assign colors to the nodes, respecting the constraints.
5. Use forward checking after each assignment to prune invalid future assignments.
6. Display the colored graph once the solution is found.

**Conclusion:**

In this lab, the CSP problem was solved using backtracking and forward checking techniques. The backtracking approach explored every possible assignment, while forward checking helped improve efficiency by pruning invalid assignments early. The results showed that forward checking significantly reduced the number of recursive calls and time taken, making the algorithm more efficient when solving CSPs.