**Assignment 2: Implement Constraint Satisfaction Problem (CSP)**

**Problem Statement:**

The goal of this assignment is to solve a constraint satisfaction problem (CSP) like Sudoku using backtracking. You will learn to represent the problem, define its constraints, and implement an algorithm to find a valid solution.

**Objectives:**

* Understand how to represent and solve constraint satisfaction problems (CSPs).
* Implement an efficient backtracking algorithm to find solutions to CSPs, such as Sudoku.

**Theory**

**What is a CSP?**  
A CSP is composed of the following elements:

* **Variables:** These are the elements that need to be assigned values (e.g., cells in a Sudoku puzzle).
* **Domains:** The set of possible values that can be assigned to each variable (e.g., numbers 1-9 for Sudoku).
* **Constraints:** Rules governing the assignment of values to variables (e.g., in Sudoku, no two cells in the same row, column, or 3x3 sub-grid can have the same value).

**Methodology**

1. **Define Variables, Domains, and Constraints:**
   * In a Sudoku puzzle, each cell is a variable.
   * The domain for each variable consists of the numbers 1 through 9.
   * Constraints ensure that no two variables within the same row, column, or 3x3 sub-grid have the same value.
2. **Start with an Empty Assignment:**
   * Begin with an empty Sudoku grid where values will be assigned to variables.
3. **Apply Backtracking:**
   * Assign a value to a variable and verify whether the assignment satisfies all constraints.
   * If the assignment is valid, move on to the next variable.
   * If it violates any constraints, backtrack and try a different value for the previous variable.
   * This process continues until a valid solution is found or all possibilities are exhausted.
4. **Repeat Until a Valid Solution is Found:**
   * Once a valid configuration of the entire grid is reached, output the solution.
   * If no valid solution exists, return a message indicating failure.

**Working Principle / Algorithm**

The following outlines the backtracking algorithm for solving Sudoku:

1. **Find the Next Empty Cell:**
   * Search the grid to locate the next cell that requires a value.
2. **Try Each Possible Value:**
   * For each number between 1 and 9:
     + Assign the number to the empty cell.
     + Check whether the assignment is valid (i.e., it doesn’t violate Sudoku constraints).
     + If valid, recursively proceed to the next cell.
3. **Backtrack When Necessary:**
   * If the recursive function results in a solution, propagate that solution upwards.
   * If not, reset the current cell and try a different number.
4. **Terminate When Solved:**
   * When all cells are filled without conflicts, the puzzle is solved.

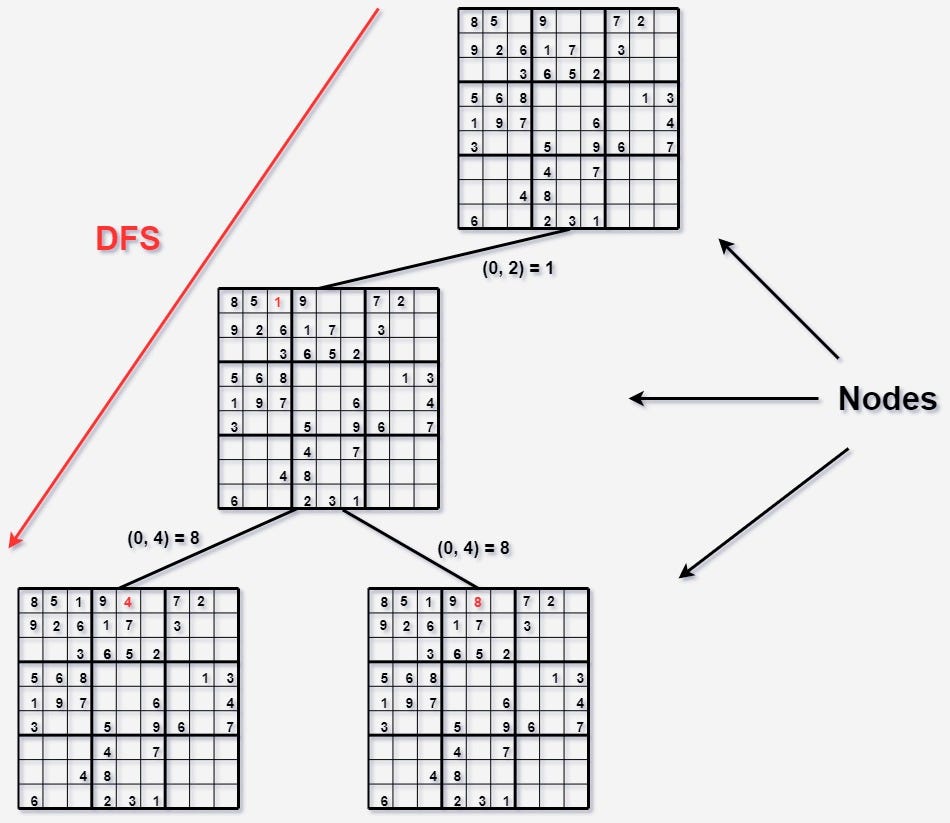
**Advantages**

* **Systematic Approach:** CSPs provide a structured method to frame problems using variables, domains, and constraints.
* **Effective for Complex Problems:** The backtracking algorithm works well for problems involving numerous variables and complex constraints.

**Disadvantages / Limitations**

* **Performance Issues:** Backtracking can be computationally expensive, especially for larger CSPs with many variables and constraints.
* **Exponential Time Complexity:** In the worst case, time complexity can increase exponentially with the problem size.

**Diagram:**



**Conclusion:**  
CSPs present an organized way to tackle complex problems by defining variables, domains, and constraints. The backtracking algorithm is a powerful technique for efficiently exploring potential solutions, making it especially useful for solving problems like Sudoku.