🔍 What is a Constraint Satisfaction Problem (CSP)?

A **Constraint Satisfaction Problem (CSP)** is a class of mathematical problems defined by:

- **A set of variables,**
- **A domain of values for each variable**, and
- **A set of constraints** that specify allowable combinations of values.

The **goal** is to assign a value to each variable such that **all constraints are satisfied.**

CSPs are central to **Artificial Intelligence (AI)** and **operations research,** especially in domains involving **combinatorial search** and **decision-making** under rules or guidelines.

🧠 Why Are CSPs Important in AI?

CSPs are widely used in AI because they:

- Allow for clear problem formulation.
- Are solvable using **systematic** and **heuristic** techniques.
- Apply to a vast array of real-world applications.

Examples of AI tasks solved with CSPs:

- Scheduling meetings without conflicts.
- Solving puzzles (e.g., Sudoku, N-Queens).
- Planning robot paths that avoid obstacles.
- Allocating resources efficiently in distributed systems.

🧩 Components of a CSP

A CSP is defined by **three key components**:

1. Variables

These represent the unknowns we need to find values for.

- Example: In a Sudoku puzzle, each cell is a variable.
- Can be binary (e.g., True/False), categorical (e.g., red/green/blue), or numeric.

2. Domains

Each variable has a **domain** — the set of possible values it can take.

- Example: A cell in Sudoku has the domain {1, 2, ..., 9}.

3. Constraints

These define relationships or rules between variables.
Types of constraints:

- **Unary**: Involve a single variable (e.g., "X ≠ 5").
- **Binary**: Involve pairs of variables (e.g., "X ≠ Y").
- **Higher-order**: Involve more than two variables (e.g., "all variables in a row must be unique").

🔤 Types of CSPs

CSPs are categorized based on structure and constraint nature:

✅ Binary CSPs

- All constraints involve exactly **two variables**.
- Easier to represent as **graphs** (nodes: variables, edges: constraints).

✅ Non-Binary CSPs

- Constraints involve more than two variables.
- Requires **hypergraphs** or additional modeling techniques.

✅ Hard vs Soft Constraints

- **Hard Constraints**: Must be strictly satisfied (e.g., "A ≠ B").
- **Soft Constraints**: Can be violated with a penalty or cost (e.g., preference-based rules).

The **backtracking algorithm** is a depth-first search method used to systematically explore possible solutions in CSPs. It operates by assigning values to variables and backtracks if any assignment violates a constraint.

How it works:

- The algorithm selects a variable and assigns it a value.
- It recursively assigns values to subsequent variables.
- If a conflict arises i.e a variable cannot be assigned a valid value then algorithm backtracks to the previous variable and tries a different value.
- The process continues until either a valid solution is found or all possibilities have been exhausted.

This method is widely used due to its simplicity but can be inefficient for large problems with many variables.

Solving Sudoku with Constraint Satisfaction Problem (CSP) Algorithms

Step 1: Define the Problem (Sudoku Puzzle Setup)

The first step is to define the Sudoku puzzle as a 9x9 grid where 0 represents an empty cell. We also define a function **print_sudoku** to display the puzzle in a human readable format.

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

def print_sudoku(puzzle):
    for i in range(9):
        if i % 3 == 0 and i != 0:
            print("- - - - - - - - - - - ")
        for j in range(9):
            if j % 3 == 0 and j != 0:
                print(" | ", end="")
            print(puzzle[i][j], end=" ")
        print()

print("Initial Sudoku Puzzle:\n")
print_sudoku(puzzle)
```