Constraint Satisfaction Problem:

Constraint satisfaction is a technique where a problem is solved when its values satisfy certain constraints or rules of the problem. Such type of technique leads to a deeper understanding of the problem structure as well as its complexity. Constraint satisfaction depends on three components, namely:

- X: It is a set of variables.
- D: It is a set of domains where the variables reside. There is a specific domain for each variable.
- C: It is a set of constraints which are followed by the set of variables.

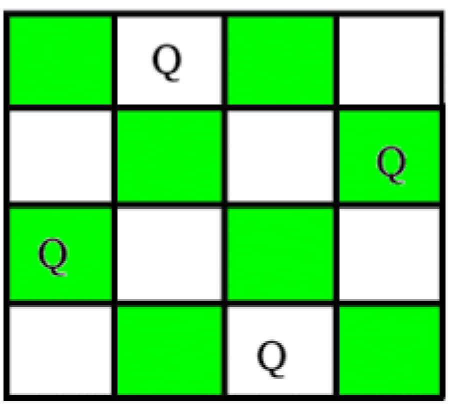
Types of Domains in CSP

There are following two types of domains which are used by the variables :

- Discrete Domain: It is an infinite domain which can have one state for multiple variables. For example, a start state can be allocated infinite times for each variable.
- Finite Domain: It is a finite domain which can have continuous states describing one domain for one specific variable. It is also called a continuous domain.

N—Queens Problem:

The N Queen is the problem of placing N chess queens on an N×N chessboard so that no two queens attack each other. For example, following is a solution for 4 Queen problem.



The expected output is a binary matrix which has 1s for the blocks where queens are placed. For example, following is the output matrix for above 4 queen solution.

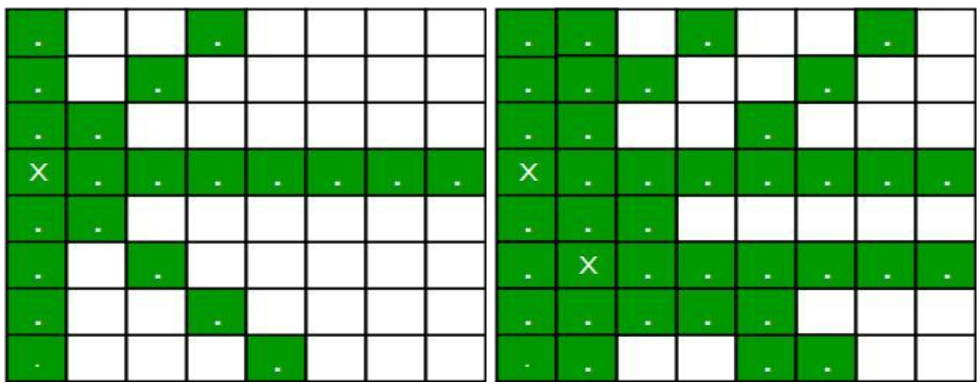
{ 0, 1, 0, 0}
{ 0, 0, 0, 1}
{ 1, 0, 0, 0}
{ 0, 0, 1, 0}

Backtracking Algorithm: The idea is to place queens one by one in different columns, starting from the leftmost column. When we place a queen in a column, we check for clashes with already placed queens. In the current column, if we find a row for which there is no clash, we mark this row and column as part of the solution. If we do not find such a row due to clashes then we backtrack and return false.

Branch and bound is an algorithm:

Branch and bound is an algorithm design paradigm which is generally used for solving combinatorial optimization problems. These problems are typically exponential in terms of time complexity and may require exploring all possible permutations in worst case. The N queens puzzle is the problem of placing N chess queens on an N×N chessboard so that no two queens threaten each other. Thus, a solution requires that no two queens share the same row, column, or diagonal. In Branch and Bound solution, after building a partial solution, we figure out that there is no point going any deeper as we are going to hit a dead end.

Let's begin by describing backtracking solution. "The idea is to place queens one by one in different columns, starting from the leftmost column. When we place a queen in a column, we check for clashes with already placed queens. In the current column, if we find a row for which there is no clash, we mark this row and column as part of the solution. If we do not find such a row due to clashes, then we backtrack and return false."



For the 1st Queen, there are total 8 possibilities as we can place 1st Queen in any row of first column. Let's place Queen 1 on row 3. After placing 1st Queen, there are 7 possibilities left for the 2nd Queen. But wait, we don't really have 7 possibilities. We cannot place Queen 2 on rows 2, 3 or 4 as those cells are under attack from Queen 1. So, Queen 2 has only 8 – 3 = 5 valid positions left. After picking a position for Queen 2, Queen 3 has even fewer options as most of the cells in its column are under attack from the first 2 Queens. We need to figure out an efficient way of keeping track of which cells are under attack. In previous solution we kept an 8-by-8 Boolean matrix and update it each time we placed a queen, but that required linear time to update as we need to check for safe cells.

Basically, we have to ensure 4 things:

1. No two queens share a column.
2. No two queens share a row.
3. No two queens share a top-right to left-bottom diagonal.
4. No two queens share a top-left to bottom-right diagonal.

Number 1 is automatic because of the way we store the solution. For number 2, 3 and 4, we can perform updates in O(1) time. The idea is to keep three Boolean arrays that tell us which rows and which diagonals are occupied.

Algorithms:

//Placing an Queen in k row and i column