Assignment No:

**Title of the Assignment:** N-Queens Problem Using CSP

**Problem statement:** Implement a solution for a Constraint Satisfaction Problem using Branch and Bound and Backtracking for n-queen’s problem or a graph coloring problem.

# Objective:

* To understand the n-Queen’s Problem and Constraint Satisfaction Problem
* To implement a n-queen problem or graph coloring problem using backtracking or branch and bound

# Theory:

**Constraint Satisfaction Problems (CSP) in Artificial Intelligence**

Finding a solution that meets a set of constraints is the goal of constraint satisfaction problems (CSPs), a type of AI issue. Finding values for a group of variables that fulfill a set of restrictions or rules is the aim of constraint satisfaction problems. For tasks including resource allocation, planning, scheduling, and decision-making, CSPs are frequently employed in AI.

**There are mainly three basic components in the constraint satisfaction problem:**

**Variables:**The things that need to be determined are variables. Variables in a CSP are the objects that must have values assigned to them in order to satisfy a particular set of constraints. Boolean, integer, and categorical variables are just a few examples of the various types of variables Variables, for instance, could stand in for the many puzzle cells that need to be filled with numbers in a sudoku puzzle.

**Domains:**The range of potential values that a variable can have is represented by domains. Depending on the issue, a domain may be finite or limitless. For instance, in Sudoku, the set of numbers from 1 to 9 can serve as the domain of a variable representing a problem cell.

**Constraints:** The guidelines that control how variables relate to one another are known as constraints. Constraints in a CSP define the ranges of possible values for variables. Unary constraints, binary constraints, and higher-order constraints are only a few examples of the various sorts of constraints. For instance, in a sudoku problem, the restrictions might be that each row, column, and 3×3 box can only have one instance of each number from 1 to 9.

**Constraint Satisfaction Problems (CSP) representation:**

The finite set of variables V1, V2, V3 ……………..Vn.

Non-empty domain for every single variable D1, D2, D3 …………..Dn.

The finite set of constraints C1, C2 …….…, Cm.

where each constraint Ci restricts the possible values for variables,

e.g., V1 ≠ V2

 Each constraint Ci is a pair <scope, relation>

Example: <(V1, V2), V1 not equal to V2>

Scope = set of variables that participate in constraint.

Relation = list of valid variable value combinations.

There might be a clear list of permitted combinations. Perhaps a relation that is abstract and that allows for membership testing and listing.

**Constraint Satisfaction Problems (CSP) algorithms:**

**The** **backtracking algorithm** is a depth-first search algorithm that methodically investigates the search space of potential solutions up until a solution is discovered that satisfies all the restrictions. The method begins by choosing a variable and giving it a value before repeatedly attempting to give values to the other variables. The method returns to the prior variable and tries a different value if at any time a variable cannot be given a value that fulfills the requirements. Once all assignments have been tried or a solution that satisfies all constraints has been discovered, the algorithm ends.

**The** **forward-checking algorithm** is a variation of the backtracking algorithm that condenses the search space using a type of local consistency. For each unassigned variable, the method keeps a list of remaining values and applies local constraints to eliminate inconsistent values from these sets. The algorithm examines a variable’s neighbors after it is given a value to see whether any of its remaining values become inconsistent and removes them from the sets if they do. The algorithm goes backward if, after forward checking, a variable has no more values.

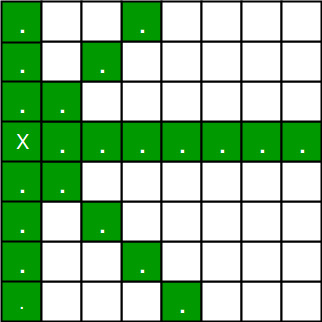
**Algorithms for propagating constraints** are a class that uses local consistency and inference to condense the search space. These algorithms operate by propagating restrictions between variables and removing inconsistent values from the variable domains using the information obtained.

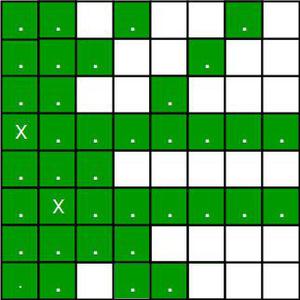
**N Queen Problem using Branch And Bound**

The **N queens puzzle** is the problem of placing N [chess](https://en.wikipedia.org/wiki/Chess) [queens](https://en.wikipedia.org/wiki/Queen_%28chess%29) 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 a backtracking solution, we backtrack when we hit a dead end. ***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 the 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.”





*Placing 1st queen on (3, 0) and 2nd queen on (5, 1)*

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.

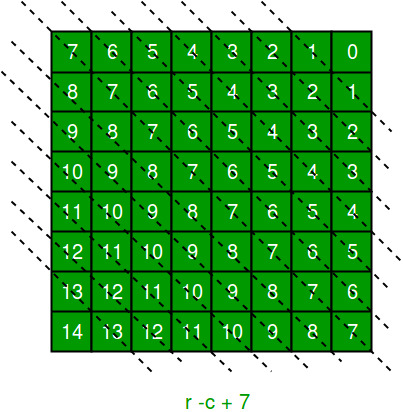
Basically, we have to ensure 4 things:

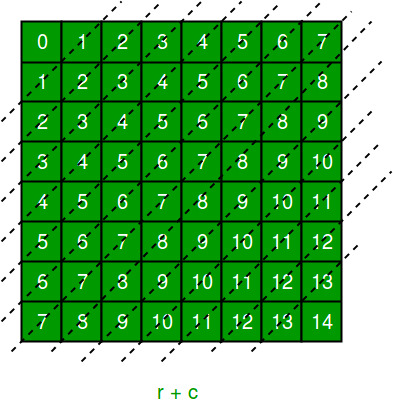
1. No two queens share a column.
2. No two queens share a row.
3. No two queens share a top-right to left-bottom diagonal.
4. No two queens share a top-left to bottom-right diagonal.

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**.

Lets do some pre-processing first. Let’s create two N x N matrix one for / diagonal and other one for \ diagonal. Let’s call them slashCode and backslashCode respectively. The trick is to fill them in such a way that two queens sharing a same /­diagonal will have the same value in matrix slashCode, and if they share same \­diagonal, they will have the same value in backslashCode matrix.  
For an N x N matrix, fill slashCode and backslashCode matrix using below formula –  
slashCode[row][col] = row + col   
backslashCode[row][col] = row – col + (N-1)

Using above formula will result in below matrices





The ‘N – 1’ in the backslash code is there to ensure that the codes are never negative because we will be using the codes as indices in an array.   
Now before we place queen i on row j, we first check whether row j is used (use an array to store row info). Then we check whether slash code ( j + i ) or backslash code ( j – i + 7 ) are used (keep two arrays that will tell us which diagonals are occupied). If yes, then we have to try a different location for queen i. If not, then we mark the row and the two diagonals as used and recurse on queen i + 1. After the recursive call returns and before we try another position for queen i, we need to reset the row, slash code and backslash code as unused again

# Conclusion:

We have understood concept of CSP and implemented an n queen’s problem.

**Oral questions:**

1. What is an example of a constraint satisfaction problem?
2. What are the three main elements of any constraint satisfaction problem?
3. Which algorithm is used in constraint satisfaction problem?
4. What are the application of constraint satisfaction problems?
5. What is the structure of a CSP?
6. What are the components of CSP in AI?
7. What is the N-Queen problem asked in the interview?
8. What is the 8 queen problem question?
9. How do you solve the n-queens problem?
10. What is the problem of the N Queens?
11. What is 4 queen problem in artificial intelligence?
12. Who formalized the N Queen problem?