**CSE-AI TY A div**

**Student Name**: Prachee Prasad

**Roll No.**: 381060

# Assignment 6

## Implement Basic Search Strategies – 8-Queens Problem

### Problem Statement:

To implement the **8-Queens Problem** using basic search strategies such as **Backtracking**, ensuring that eight queens are placed on a chessboard in such a way that no two queens attack each other.

### Objective:

To understand how **search and constraint satisfaction** techniques are used in Artificial Intelligence to explore solution spaces and eliminate invalid configurations through systematic searching and backtracking.

### Requirements:

- Input: Chessboard of size **N × N** (default N = 8).
- Output: All possible valid configurations of queens where no two attack each other.
- Approach: Use **backtracking** to explore possible queen placements column by column.

### Operating System:

Windows / Linux / macOS

### Libraries and Packages Used:

- **C++ iostream**, **vector**, **cmath** for recursion and safe position checking.
- No external libraries required.

## Theory:

**Definition:**

The **8-Queens Problem** is a classic **constraint satisfaction problem (CSP)** in which the goal is to place eight queens on an 8×8 chessboard such that no two queens threaten each other — meaning no two queens share the same row, column, or diagonal.

**Structure:**

- **State Space:** Each configuration of queens placed on the board.
- **Constraints:**
  - Only one queen per column.
  - No two queens in the same row or diagonal.
- **Goal State:** A configuration where all eight queens are safely placed.

## Methodology:

1. Represent the chessboard as a **2D array** or **1D list** where index = column and value = row position of the queen.
2. Place queens **column by column**.
3. For each column, try all possible row positions.
4. Before placing a queen, check if it's **safe** (no conflict with previously placed queens).
5. If safe, place the queen and move to the next column.
6. If no valid position exists, **backtrack** to the previous column and try a different position.
7. Repeat until all queens are placed successfully.

## Advantages:

- Systematic and guarantees finding all valid solutions.
- Demonstrates the concept of **depth-first search** with pruning.

- Works for any N (generalized N-Queens problem).

## Limitations:

- Computationally expensive for larger N (exponential complexity).
- Doesn’t scale well without optimization techniques (like heuristic pruning).

## Working / Algorithm:

### Algorithm Steps:

1. Start from the **first column**.
2. Try placing a queen in each **row** of the current column.
3. Check if the position is **safe**:
    - No other queen in the same row.
    - No other queen in upper or lower diagonals.
4. If safe, place the queen and move to the next column recursively.
5. If not safe or no row is valid, **backtrack** to the previous column.
6. Continue until all queens are placed or all possibilities are exhausted.
7. Display all valid solutions.

## Example Output:

```
Solution 1:
. Q . . . . . .
. . . Q . . . .
. . . . . Q . .
. . Q . . . . .
Q . . . . . . .
. . . . Q . . .
. . . . . . Q .
. . Q . . . . .
```

Each `Q` represents a queen, and `.` represents an empty space.

## Conclusion:

The **8-Queens Problem** demonstrates how **AI search strategies** like **backtracking** efficiently explore the state space to satisfy multiple constraints.
It is a foundational problem in **constraint satisfaction**, **recursive problem-solving**, and **AI search methods**, forming the basis for more advanced algorithms like **Hill Climbing**, **Genetic Algorithms**, and **Simulated Annealing**.