

**Faculty of Computer Science & Information**

**Technology**

**2023-2027**

**Programming for Artificial Intelligence**

**Lab**

**Task 4**

**The Superior University**

**Submitted To:**

**Mr. Rasikh Ali**

**Submitted By:**

**Masooma Zahra**

**Roll No:**

**SU92-BSAIM-F23-088**

**Section:**

**4B**

**Department:**

**SE**

**Overview:**

The **N-Queens problem** is a classic combinatorial challenge of placing **N queens on an N×N chessboard** such that no two queens attack each other. This program uses **backtracking** to explore possible placements, ensuring validity at each step. If a valid configuration is found, it prints the board; otherwise, it reports that no solution exists. The approach systematically places queens column by column, backtracking when necessary to find a feasible arrangement.

**Code Overview:**

This code solves the **N-Queens problem** using **backtracking**. It works by:

* Placing queens **one column at a time**.
* Ensuring each placement is **safe** before proceeding.
* **Backtracking** if a conflict occurs.
* Displaying the valid **board configuration** if a solution is found.

**Functions:**

**def issafe(board, row, column, n):**

* Checks if placing a queen at (row, column) is **safe** by ensuring:
  + No other queens are in the **same row**.
  + No queens are in the **upper-left diagonal**.
  + No queens are in the **lower-left diagonal**.

**Parameters:**

* board: A **2D list** representing the chessboard.
* row: The **row index** where the queen is being placed.
* column: The **column index** where the queen is being placed.
* n: The **size of the chessboard (N×N)**.

**Returns:**

* True if the queen can be placed **safely**, otherwise False.

**def solve(board, column, n):**

* Recursively places **N queens column by column** while ensuring no two queens attack each other.
* Calls issafe() before placing a queen.
* Uses **backtracking** to explore all possible configurations.

**Parameters:**

* board: A **2D list** representing the chessboard.
* column: The **current column** where a queen is being placed.
* n: The **size of the chessboard (N×N)**.

**Returns:**

* True if a **valid solution** is found, otherwise False.

**def solvequeen(n):**

* Initializes an **N×N chessboard** with all positions empty.
* Calls the solve() function to find a valid placement for **N queens**.
* Prints the solution if found, otherwise displays "Solution does not exist".

**Parameters:**

* n: The **size of the chessboard (N×N)**.

**Returns:**

* True if a **valid solution** is found, otherwise False.

**def printsol(board):**

* Prints the **chessboard solution**, where:
  + "Q" represents a **queen**.
  + "." represents an **empty space**.

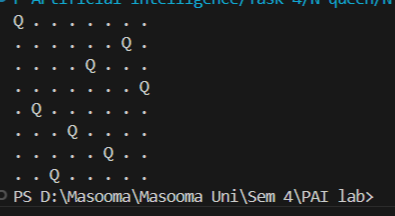
**Parameters:**

* board: A **2D list** representing the chessboard.

If we change the values of N in the code, then the size of the board changes and the solution is provided accordingly. Few examples are as following:

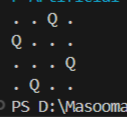
**Example 1:**

If the value of N is given “N=8”:



**Example 2:**

If the value of N is given “N=4”:



**Example 3:**

If the value of N is given “N=16”:

