**THE SUPERIOR UNIVERSITY LAHORE** 

**LAB#4**

**Semester: 4th Se~~ctio~~n: AI (B)**

**Faculty of Computer Science and Information Technology Deadline:**

**Subject: PAI LAB Total Marks:**

**Name: M.Hassan Shahid**

**Roll No: 059**

***Instructions:***

* Copying of the assignment willresult in failure.
* Assignment should be submitted in word or pdf.

# ****Solving the N-Queens Problem with Backtracking****

## ****Introduction****

The **N-Queens problem** is a famous puzzle where we need to place N queens on an N × N chessboard so that none of them can attack each other. Since queens can move in any direction (horizontally, vertically, and diagonally), the challenge is to position them safely.

This report breaks down how the given Python code solves the problem using **backtracking**, a smart way of testing different queen placements while avoiding bad moves early on.

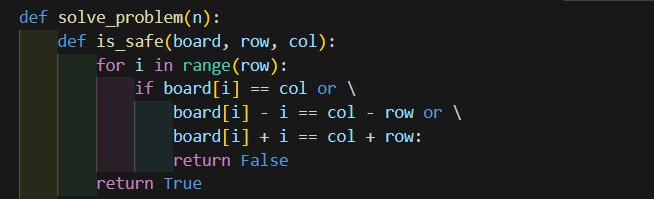
## ****Understanding the Code Step by Step****

### ****1. Defining the Main Function****

def solve\_problem(n):

This function is the starting point. It takes n (the size of the board and number of queens) as input and finds all possible ways to place the queens correctly.

### ****2. Checking If a Queen Can Be Placed (****is\_safe ****function)****

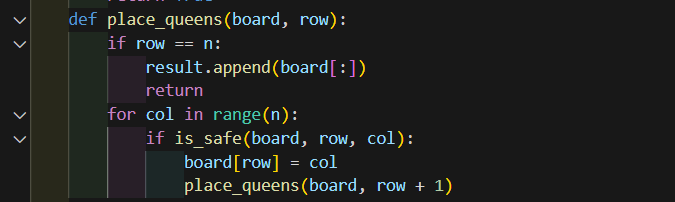


This function ensures that a queen at (row, col) is placed safely.

How does it check?

* It loops over previous rows (for i in range(row)) to see if any queens already placed **conflict** with the new queen:
  + **Same column** → If another queen is already in col, it's not safe.
  + **Same diagonal** → If another queen is on the same diagonal (difference between row and column positions is the same), it's also unsafe.
* If no conflicts are found, the function returns True, meaning the placement is valid.

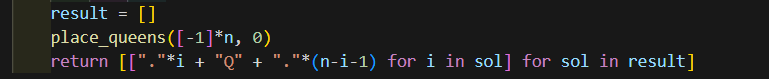
### ****3. Placing Queens on the Board (****place\_queens ****function)****

This is where the actual **backtracking** happens.

What’s going on here?

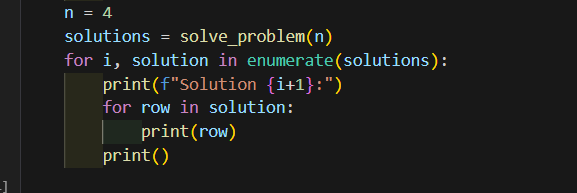
* If row == n, it means we have successfully placed all queens, so we **save the board** as a solution.
* Otherwise, we **try every column** in the current row (for col in range(n)).
* If a column is **safe** (is\_safe(board, row, col)), we place a queen there and **move to the next row**.
* If we reach a dead-end (no valid placements left), we **backtrack** by removing the last placed queen and trying a different column.

### ****4. Storing and Formatting the Solutions****

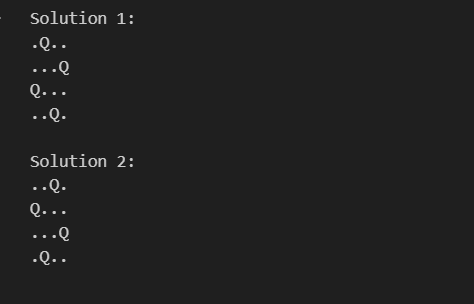
Once we find solutions, we **format them into a visual chessboard**.

* result = [] keeps track of all solutions.
* place\_queens([-1]\*n, 0) starts solving with an empty board (-1 means no queens placed).
* The return statement converts solutions into a **human-readable chessboard**, where:
  + "." represents an empty square.
  + "Q" represents a queen.

### ****5. Running the Code for**** N=4

This code runs the function for a **4×4 board** and prints all valid solutions.

**Example Output:**



Each solution shows one way to place 4 queens safely.

**Conclusion**

* The **backtracking approach** is an elegant way to solve the **N-Queens problem** efficiently.
* It finds **all possible valid solutions**, not just one.
* By **avoiding bad placements early**, it reduces unnecessary calculations.
* The code provides a **visual representation** of the chessboard, making it easy to understand.