

**The Superior University**

***Session 2023-2025***

***Department of Software Engineering***

***Faculty of Computer Science & Information Technology***

***The Superior University, Lahore***

***Course: Programming For Artificial Intelligence***

***Course Instructor:Sir Rasikh ali***

***Semester 4 BSAI***

***Fall 2025***

***Name:Sara hanif***

***Roll no:082***

***LAB TASK 4***

***Task: N-Queens Problem (Dynamic):***

***Step 1:is\_valid function:***

*def is\_valid(diagonal, row, col):*

*for i in range(row):*

*if diagonal[i] == col or \*

*diagonal[i] - i == col - row or \*

*diagonal[i] + i == col + row:*

*return False*

*return True*

***Purpose***

This function checks if placing a queen at position (row, col) is **valid.**

### **How it Works**

* diagonal[i] == col: Checks if another queen is already placed in the same **column.**
* diagonal[i] - i == col - row: Checks the **main diagonal** (↘ direction).
* diagonal[i] + i == col + row: Checks the **anti-diagonal** (↙ direction).

If any of these conditions are met, the function returns False (not a valid position); otherwise, it returns True.

**Step 2: solve\_n\_queens(n) Function:**

def solve\_n\_queens(n):

def solve(row, diagonal):

if row == n:

result.append(diagonal[:])

return

for col in range(n):

if is\_valid(diagonal, row, col):

diagonal[row] = col

solve(row + 1, diagonal)

diagonal[row] = -1

result = []

diagonal = [-1] \* n

solve(0, diagonal)

return result

### **Purpose**

This function solves the **N-Queens problem** using **backtracking.**

### ***How it Works***

1. **Base Case:**
   * If row == n, all queens are placed correctly **→ store the solution** in result and return.
2. **Recursive Case:**
   * Try placing a queen in every column of the current row.
   * Use is\_valid() to check if it is a safe position.
   * If valid, place the queen (diagonal[row] = col).
   * Recur for the next row (solve(row + 1, diagonal)).
   * **Backtrack:** Remove the queen (diagonal[row] = -1) to explore other options.

## ***Step 3: print\_solution(solutions) Function***

python

CopyEdit

def print\_solution(solutions):

for solution in solutions:

for i in range(len(solution)):

row = ['.'] \* len(solution)

row[solution[i]] = 'Q'

print(" ".join(row))

print("\n")

### ***Purpose***

This function **displays the solutions** in a chessboard format.

### ***How it Works***

* It iterates through each solution.
* For each row, it initializes ['.', '.', '.', ..., '.'].
* It places a queen ('Q') at the correct column position.
* It prints the board.

## ***Step 4: Running the Program***

python

CopyEdit

N = 6

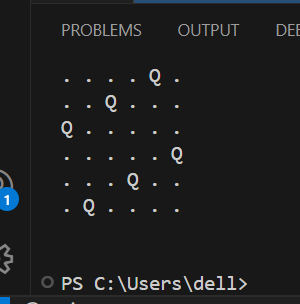
solutions = solve\_n\_queens(N)

print(f"Number of solutions: {len(solutions)}")

print\_solution(solutions)

### ***How it Works***

* Solves the 6 × 6 N-Queens problem.
* Prints the number of valid solutions.
* Displays the solutions.
* \_valid(diagonal, row, col):
* for i in range(row):
* if diagonal[i] == col or \
* diagonal[i] - i == col - row or \
* diagonal[i] + i == col + row:
* return False
* return True
* def solve\_n\_queens(n):
* def solve(row, diagonal):
* if row == n:
* result.append(diagonal[:])
* return
* for col in range(n):
* if is\_valid(diagonal, row, col):
* diagonal[row] = col
* solve(row + 1, diagonal)
* diagonal[row] = -1
* result = []
* diagonal = [-1] \* n
* solve(0, diagonal)
* return result
* def print\_solution(solutions):
* for solution in solutions:
* for i in range(len(solution)):
* row = ['.'] \* len(solution)
* row[solution[i]] = 'Q'
* print(" ".join(row))
* print("\n")
* N = 6
* solutions = solve\_n\_queens(N)
* print(f"Number of solutions: {len(solutions)}")
* print\_solution(solutions)

******