Analysis of Algorithm

Practical no 10 :

N-queen Algorithm

**Code :**

public class NQueens {

    private static final int N = 8; // Number of queens

    // Function to check whether the queens are threaten or not

    private static boolean isSafe(int board[][], int row, int col) {

        // Check this row

        for (int i = 0; i < col; i++) {

            if (board[row][i] == 1)

                return false;

        }

        // Check upper diagonal

        for (int i = row, j = col; i >= 0 && j >= 0; i--, j--) {

            if (board[i][j] == 1)

                return false;

        }

        // Check lower diagonal

        for (int i = row, j = col; i < N && j >= 0; i++, j--) {

            if (board[i][j] == 1)

                return false;

        }

        return true;

    }

    private static boolean solveNQueens(int board[][], int col) {

        if (col >= N)

            return true; //returns true if all queens are placed

        // Try to place this queen in all columns one by one

        for (int i = 0; i < N; i++) {

            if (isSafe(board, i, col)) {

                board[i][col] = 1;

                if (solveNQueens(board, col + 1))

                    return true;

                board[i][col] = 0;

            }

        }

        return false;

    }

    private static void printBoard(int board[][]) {

        for (int i = 0; i < N; i++) {

            for (int j = 0; j < N; j++)

                System.out.print((board[i][j] == 1 ? "Q " : "- "));

            System.out.println();

        }

    }

    public static void main(String[] args) {

        int board[][] = new int[N][N];

        if (!solveNQueens(board, 0))

            System.out.println("Solution does not exist");

        else

            printBoard(board);

    }

}

**Output :**

**A screenshot of a computer screen

AI-generated content may be incorrect.**

**Analysis :**

1. Constants and Board Initialization:
   * N is set to 8, indicating an 8×8 board.
   * The board is represented by a 2D integer array initialized to 0 (no queens placed).
2. Safety Check (isSafe):
   * This function checks if a queen can be safely placed at board[row][col].
   * It checks:
     + The current row (left side).
     + The upper diagonal (to the left).
     + The lower diagonal (to the left).
3. Backtracking (solveNQueens):
   * This function attempts to place queens column by column.
   * For each column, it tries each row, calling isSafe to check if a queen can be placed.
   * If placing a queen leads to a solution, it returns true; otherwise, it backtracks (removes the queen).
4. Printing the Board (printBoard):
   * It prints the board with 'Q' representing a queen and '-' representing an empty cell.
5. Main Function:
   * Initializes the board and calls the solveNQueens method.
   * If a solution is found, it prints the board; otherwise, it indicates no solution exists.

Time Complexity

The time complexity of the N-Queens problem using backtracking can be analyzed as follows:

* Recursive Calls: The algorithm makes recursive calls for each column (N columns).
* Placement Attempts: For each column, it tries placing a queen in each of the N rows. In the worst case, it tries to place a queen in all rows for each column, leading to a total of NNN^NNN possibilities in the worst-case scenario.
* Safe Check: The isSafe function checks three directions for each placement, which takes O(N)O(N)O(N) time in the worst case.

Thus, the overall time complexity can be approximated as:

O(N!⋅N)O(N! \cdot N)O(N!⋅N)

This is because, in the worst case, the solution may require evaluating every possible arrangement of queens, leading to a factorial growth with NNN.

Space Complexity

The space complexity can be analyzed based on:

1. Board Storage:
   * The board requires O(N2)O(N^2)O(N2) space as it is a 2D array of size N×NN \times NN×N.
2. Recursion Stack:
   * The maximum depth of the recursion stack is NNN (one for each column).

Therefore, the overall space complexity is:

O(N2)O(N^2)O(N2)

This accounts for the space needed to store the board.