K. J. Somaiya College of Engineering, Mumbai-77
(A Constituent College of Somaiya Vidyavihar University)
Department of Computer Engineering

Batch: E-2 Roll No.: 16010123325
Experiment No. 9
Grade: AA / AB / BB / BC / CC / CD /DD
Signature of the Staff In-charge with date

Title: Study, Implementation and Analysis of 8-Queens problem.

Objective: To learn the Backtracking strategy of problem solving for 8-Queens problem

CO to be achieved:

Sr. No	Objective
CO 1	Compare and demonstrate the efficiency of algorithms using asymptotic complexity notations.
CO 2	Analyze and solve problems for divide and conquer strategy, greedy method, dynamic programming approach and backtracking and branch & bound policies.

Books/ Journals/ Websites referred:

1. **Ellis horowitz, Sarataj Sahni, S.Rajsekaran," Fundamentals of computer algorithm", University Press**
2. **T.H.Cormen ,C.E.Leiserson,R.L.Rivest and C.Stein," Introduction to algortihtms",2nd Edition ,MIT press/McGraw Hill,2001**
3. **http://www.math.utah.edu/~alfeld/queens/queens.html**
4. **http://www-isl.ece.arizona.edu/ece175/assignments275/assignment4a/Solving%208%20queen%20problem.pdf**
5. **http://www.slideshare.net/Tech_MX/8-queens-problem-using-back-tracking**
6. **http://www.mathcs.emory.edu/~cheung/Courses/170.2010/Syllabus/Backtracking/8queens.html**
7. **http://www.geeksforgeeks.org/backtracking-set-3-n-queen-problem/**
8. **http://www.hbmeyer.de/backtrack/achtdamen/eight.htm**

Pre Lab/ Prior Concepts:
Data structures, Concepts of algorithm analysis

Historical Profile:
The **N-Queens puzzle** is the problem of placing N queens on an N×N chessboard so that no two queens attack each other. Thus, a solution requires that no two queens share the same row, column, or diagonal.

New Concepts to be learned:
Application of algorithmic design strategy to any problem, Backtracking method of problem-solving Vs other methods of problem solving, 8- Queens problem and its applications.

Algorithm N Queens Problem: -

```
void NQueens(int k, int n)
// Using backtracking, this procedure prints all possible placements of n queens on an n X n
chessboard so that they are nonattacking.
{       for (int i=1; i<=n; i++)
      {
            if (Place(k, i))
             {
               x[k] = i;
               if (k==n)
                        for (int j=1;j<=n;j++)          Print  x[j] ;
               else NQueens(k+1, n);
             }
       }
}

Boolean Place(int k, int i)
// Returns true if a queen can be placed in k^th row and i^th column.  Otherwise it returns false.
// x[] is a global array whose first (k-1) values have been set. abs(r) returns absolute value of
r.
{
for (int j=1; j < k; j++)
        if ((x[j] == i)  // Two in the same column
      || (abs(x[j]-i) == abs(j-k)))                  // or in the same diagonal
          return(false);
return(true);
 }
```

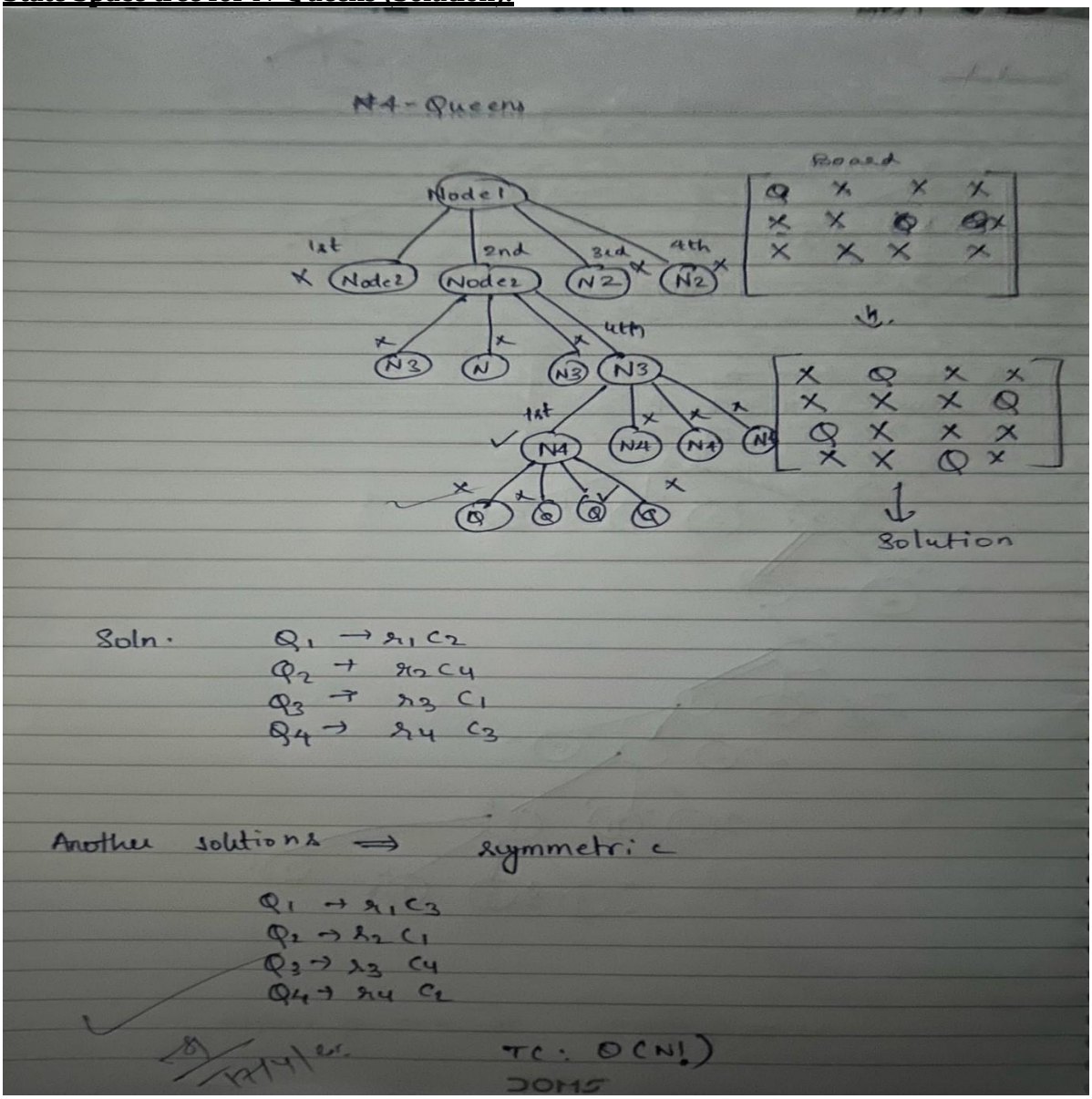
Example 8-Queens Problem:

The eight queens puzzle is the problem of placing eight chess queens on an 8×8 chessboard so that no two queens threaten each other i.e. no two queens share the same row, column, or diagonal.

Solution Using Backtracking Approach:

The idea is to place queens one by one in different columns, starting from the leftmost column. When we place a queen in a column, we check for clashes with already placed queens. In the current column, if we find a row for which there is no clash, we mark this row and column as part of the solution. If we do not find such a row due to clashes then we backtrack and return false.

State Space tree for N-Queens (Solution):



Implementation (Code):

```
public class EightQueens {

    static final int N = 8;
    static int[] board = new int[N];

    public static void main(String[] args) {
        if (solve(0)) {
            printSolution();
        } else {
            System.out.println("Solution does not exist");
        }
    }

    static boolean solve(int row) {
        if (row == N) {
            return true;
        }

        for (int col = 0; col < N; col++) {
            if (isSafe(row, col)) {
                board[row] = col; // Place the queen
                if (solve(row + 1)) {
                    return true;
                }
            }
        }
        return false;
    }

    // Check if it's safe to place a queen at board[row][col]
    static boolean isSafe(int row, int col) {
        for (int i = 0; i < row; i++) {
            // Check if any queen is in the same column or diagonal
            if (board[i] == col || Math.abs(board[i] - col) == Math.abs(i - row)) {
                return false;
            }
        }
        return true;
    }

    static void printSolution() {
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < N; j++) {
                if (board[i] == j) {
                    System.out.print("Q ");
                } else {
                    System.out.print(". ");
```

```
                    }
                }
                System.out.println();
            }
        }
}
```

OUTPUT:

```
cd "/Users/shreya/Desktop/CE/SY/Lab/AOA/" && javac EightQueens.java && java EightQueens
shreya@Shreyas-MacBook-Air-2 AOA % cd "/Users/shreya/Desktop/CE/SY/Lab/AOA/" && javac EightQueens.java && java EightQueens
Q . . . . . .
. . . . Q . . .
. . . . . . . Q
. . . . . Q . .
. . Q . . . . .
. . . . . . Q .
. Q . . . . . .
. . . Q . . . .
shreya@Shreyas-MacBook-Air-2 AOA %
```

Algorithm:

1. **Initialize**:
   Create an array board[N] where each index represents a row and the value at each index represents the column position of a queen in that row.

2. **Backtracking function solve(row)**:

   - If row == N: Return true (all queens have been placed successfully).
   - For each col from 0 to N-1 (representing columns):
     - Check if placing a queen at (row, col) is safe (no queen in the same column or diagonal).
     - If safe, place the queen at board[row] = col.
     - Recursively call solve(row + 1) to place queens in the next row.
     - If placing the queen leads to a solution, return true.
     - If not, backtrack: remove the queen and try the next column.

3. **Safety check function isSafe(row, col)**:
   a. For each row i before the current row, check if a queen is already placed in the same column col or in any diagonal.
   b. If no conflicts, return true; otherwise, return false.

4. **Print the board**:
   a. Once a solution is found, print the board where Q represents a queen and . represents an empty space.

Analysis of Backtracking solution:

- **Time Complexity**: The worst-case time complexity is O(N^N), but backtracking reduces the number of recursive calls significantly.

- **Space Complexity**: O(N), where N is the number of queens (due to the storage of the board and recursion stack).

CONCLUSION:

The above experiment highlights implementation of N-Queens problem and implementation of 4-Queens problem.