


**Experiment No. : 7**

**Title: N Queen problem using Backtracking**



Batch:SY-IT(B3)

Roll No.: 16010423076

Experiment No.: 7

**Aim:** To Implement N Queen problem using Backtracking. & Virtual Lab on N Queen algorithm using Backtracking

---

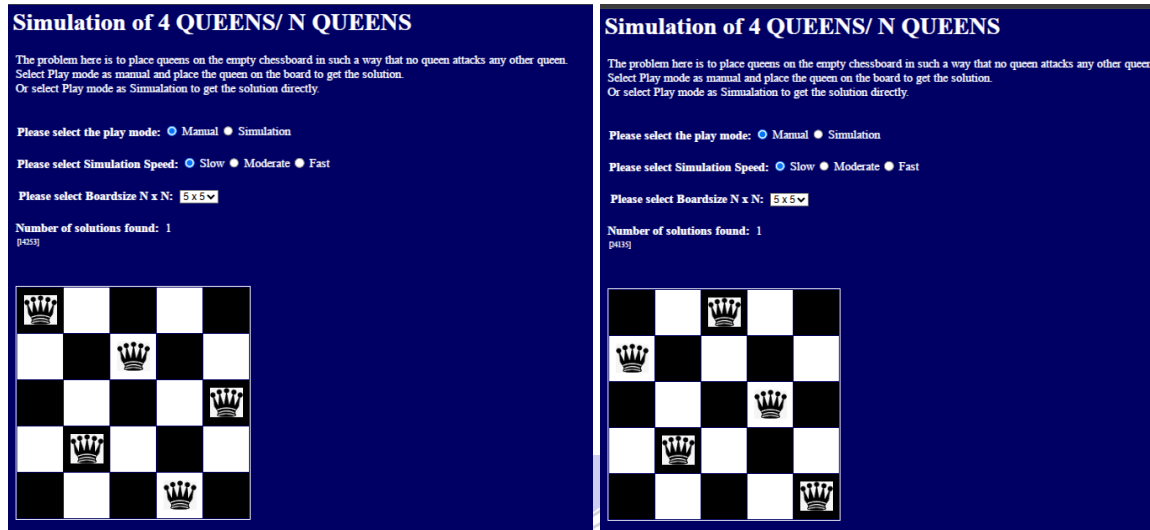
**Algorithm of N Queen problem:**

```
n_queens(n) {  
    rn_queens(1,n)  
}  
  
rn_queens(k,n) {  
    for row[k] = 1 to n  
        if (position_ok(k,n) )  
            if ( k == n ) {  
                for i = 1 to n  
                    print(row[k] + " ")  
                println  
            }  
            else  
                rn_queens(k+1,n)  
        }  
}  
  
position_ok(k,n) {  
    for i = 1 to k-1  
        if (row[k] == row[i] || abs(row[k]-row[i]) == k-i)  
            return false  
    return true  
}
```

**Working of N Queen problem:****Virtual Lab of N Queen problem:**

**Simulate Manual steps of N Queen Problem using Backtracking for 5x5 Matrix using following Link and display the Output.**

<https://vsit.edu.in/vlab/AI/main%20ckt/simulator/4QueenSolution.htm>

**Time Complexity Derivation of N Queen problem:**

The N-Queens problem has an exponential time complexity due to the recursive backtracking approach. Here's how we derive it:

1. At row 1, we have N choices for placing a queen.
2. At row 2, we have at most N-1 choices (some columns get eliminated due to attacks).
3. At row 3, we have at most N-2 choices, and so on.

This results in an upper bound of  $O(N!)$  (factorial time complexity). The actual number of valid solutions is much smaller, but in the worst case, we explore most of the possibilities.

**Program(s) of N Queen problem:**

```

#include <stdio.h>
#include <stdlib.h>

#define N 6 // Change this for different board sizes

int board[N];

void printSolution(int n) {
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j++) {
            if (board[i] == j)
                printf("Q ");
            else
                printf("- ");
        }
        printf("\n");
    }
    printf("\n");
}

int isSafe(int row, int col) {
    for (int i = 1; i < row; i++) {
        if (board[i] == col || abs(board[i] - col) == abs(i -
row)) {
            return 0;
        }
    }
    return 1;
}

void solveNQueens(int row, int n) {
    if (row > n) {
        printSolution(n);
        return;
    }
    for (int col = 1; col <= n; col++) {
        if (isSafe(row, col)) {
            board[row] = col;
            solveNQueens(row + 1, n);
        }
    }
}

int main() {
    solveNQueens(1, N);
    return 0;
}

```

**Output(o) of N Queen problem:**

```
-Q----
---Q--
-----Q
Q-----
--Q---
-----Q-
```

```
--Q---
-----Q
-Q----
-----Q-
Q-----
---Q--
```

```
---Q--
Q-----
-----Q-
-Q-----
-----Q
--Q---
```

```
---Q-
--Q---
Q-----
-----Q
---Q--
-Q----
```

```
=== Code Execution Successful ===
```

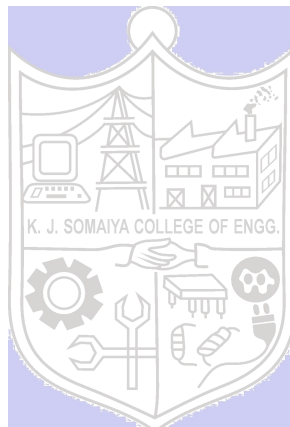
```
Output
-Q - - -
- - Q - -
- - - Q
Q - - - -
- Q - - -
- - - Q -

- Q - - -
- - - Q
-Q - - -
- - Q - -
Q - - - -
- - Q - -

- Q - - -
Q - - - -
- - - Q
-Q - - -
- - - Q
-Q - - -
- - Q - -

- - - Q -
- Q - - -
Q - - - -
- - - Q
- - Q - -
-Q - - -

=== Code Execution Successful ===
```



---

**Post Lab Questions:-****Explain the process of backtracking ?**

Backtracking is a trial-and-error algorithmic approach that incrementally builds a solution by exploring possible choices. If a choice leads to a valid solution, it continues; otherwise, it backtracks (undoes the last choice) and tries a different path.

In N-Queens, we place queens row by row, ensuring no conflicts. If a conflict arises, we backtrack to the previous row and try a different column.

**What are the advantages of backtracking as against brute force algorithms?**

Efficiency: Backtracking avoids exploring invalid solutions, whereas brute force checks all possible placements.

Pruning: It eliminates impossible paths early, reducing computations.

Optimization: It works well for constraint-based problems like N-Queens, Sudoku, and pathfinding.

**What do you mean by P and NP Problem?**

P (Polynomial Time): Problems that can be solved in polynomial time (e.g., sorting).

NP (Nondeterministic Polynomial Time): Problems whose solutions can be verified in polynomial time but may not be solvable in polynomial time (e.g., N-Queens, Traveling Salesman).

NP-Complete: The hardest problems in NP, where if one NP-complete problem is solved in polynomial time, all NP problems can be solved in polynomial time.

---

**Conclusion: (Based on the observations):**

From this experiment, I learned that the N-Queens problem can be efficiently solved using the backtracking approach. By placing queens row by row and backtracking whenever conflicts arise, we optimize the search process. The algorithm significantly reduces redundant computations compared to brute force and showcases the power of recursive problem-solving.

---

**Outcome:** CO 3. Implement Advance Programming algorithms with its application.

---

**References:**

1. Richard E. Neapolitan, " Foundation of Algorithms ", 5th Edition 2016, Jones & Bartlett Students Edition
2. Harsh Bhasin , " Algorithms : Design & Analysis", 1st Edition 2013, Oxford Higher education, India
3. T.H. Coreman ,C.E. Leiserson,R.L. Rivest, and C. Stein, " Introduction to algorithms", 3rd Edition 2009, Prentice Hall India Publication

4. Jon Kleinberg, Eva Tardos, " Algorithm Design", 10th Edition 2013, Pearson India Education Services Pvt. Ltd.

