



Experiment No. : 7

Title: N Queen problem using Backtracking



Batch: A1

Roll No.: 16010422013

Experiment No.: 7

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

Algorithm of N Queen problem:

- Start in the leftmost column
- If all queens are placed return true
- Try all rows in the current column. Do the following for every row.
 - If the queen can be placed safely in this row
 - Then mark this **[row, column]** as part of the solution and recursively check if placing queen here leads to a solution.
 - If placing the queen in **[row, column]** leads to a solution then return **true**.
 - If placing queen doesn't lead to a solution then unmark this **[row, column]** then backtrack and try other rows.
 - If all rows have been tried and valid solution is not found return **false** to trigger backtracking.

Working of N Queen problem:

The N-Queens problem is a classic problem in computer science and combinatorial optimization. The objective is to place N chess queens on an N×N chessboard in such a way that no two queens threaten each other. In other words, no two queens can share the same row, column, or diagonal.

Backtracking is a systematic method used to solve the N-Queens problem efficiently by exploring all possible configurations of queens on the board without explicitly constructing every configuration. Here's how the backtracking algorithm works for the N-Queens problem:

1. **Start from an empty board:** Initially, the chessboard is empty, and no queens are placed.
2. **Placing queens row by row:** The algorithm starts by placing a queen in the first row of the chessboard. It then tries to place queens row by row, starting from the top row and moving downwards.
3. **Choosing a column:** For each row, the algorithm iterates through each column of the current row and tries to place a queen in that column.
4. **Checking if the placement is valid:** Before placing a queen in a particular column of the current row, the algorithm checks if placing the queen there violates any of the N-Queens constraints:
 - No two queens share the same row.
 - No two queens share the same column.
 - No two queens share the same diagonal.
5. **Backtracking:** If placing a queen in the current column violates any of the constraints, the algorithm backtracks and tries placing the queen in the next column of the same row. If there are no valid columns left to place the queen in the current row, the algorithm backtracks to the previous row and tries a different column for the queen in that row.

6. **Reaching a solution:** The algorithm continues this process recursively, trying different configurations of queens until either a solution is found (i.e., all queens are successfully placed on the board) or all possibilities have been exhausted.
7. **Termination:** The algorithm terminates when it finds a valid placement of queens on the board or when it exhausts all possibilities without finding a solution.

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>

Simulation of 4 QUEENS/ N QUEENS

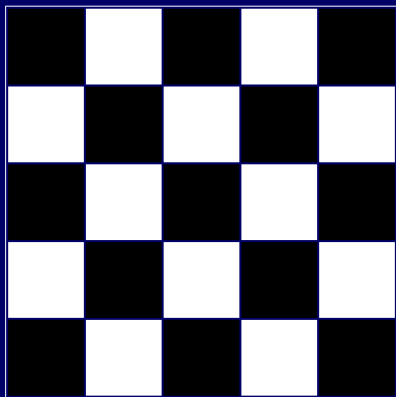
The problem here is to place queens on the empty chessboard in such a way that no queen attacks any other queen. Select Play mode as manual and place the queen on the board to get the solution. Or select Play mode as Simulation to get the solution directly.

Please select the play mode: ☒ Manual ☐ Simulation

Please select Simulation Speed: ☒ Slow ☐ Moderate ☐ Fast

Please select Boardsize N x N: 5 x 5 ▼

Number of solutions found: 0



← → ↻

vsit.edu.in/vlab/AI/main%20ckt/simulator/4QueenSolution.htm

Simulation of 4 QUEENS/ N QUEENS

The problem here is to place queens on the empty chessboard in such a way that no queen attacks any other queen. Select Play mode as manual and place the queen on the board to get the solution. Or select Play mode as Simulation to get the solution directly.

Please select the play mode: ☒ Manual ☐ Simulation

Please select Simulation Speed: ☒ Slow ☐ Moderate ☐ Fast

Please select Boardsize N x N:

Number of solutions found: 0

vsit.edu.in says

This queen position is not possible

OK

← → ↻

vsit.edu.in/vlab/AI/main%20ckt/simulator/4QueenSolution.htm

Simulation of 4 QUEENS/ N QUEENS

The problem here is to place queens on the empty chessboard in such a way that no queen attacks any other queen. Select Play mode as manual and place the queen on the board to get the solution. Or select Play mode as Simulation to get the solution directly.

Please select the play mode: ☒ Manual ☐ Simulation

Please select Simulation Speed: ☒ Slow ☐ Moderate ☐ Fast

Please select Boardsize N x N:

Number of solutions found: 0

Simulation of 4 QUEENS/ N QUEENS

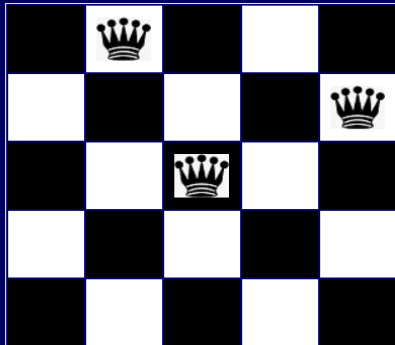
The problem here is to place queens on the empty chessboard in such a way that no queen attacks any other queen.
Select Play mode as manual and place the queen on the board to get the solution.
Or select Play mode as Simulation to get the solution directly.

Please select the play mode: ☒ Manual ☐ Simulation

Please select Simulation Speed: ☒ Slow ☐ Moderate ☐ Fast

Please select Boardsize N x N:

Number of solutions found: 0



Simulation of 4 QUEENS/ N QUEENS

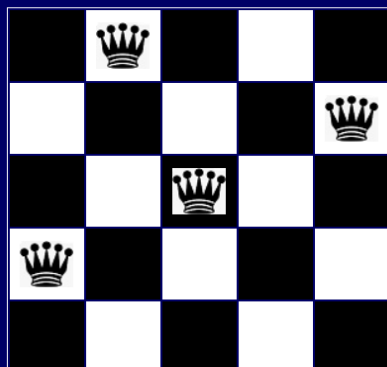
The problem here is to place queens on the empty chessboard in such a way that no queen attacks any other queen.
Select Play mode as manual and place the queen on the board to get the solution.
Or select Play mode as Simulation to get the solution directly.

Please select the play mode: ☒ Manual ☐ Simulation

Please select Simulation Speed: ☒ Slow ☐ Moderate ☐ Fast

Please select Boardsize N x N:

Number of solutions found: 0



Simulation of 4 QUEENS/ N QUEENS

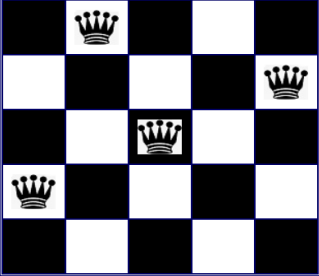
The problem here is to place queens on the empty chessboard in such a way that no queen attacks any other queen. Select Play mode as manual and place the queen on the board to get the solution. Or select Play mode as Simulation to get the solution directly.

Please select the play mode: ☒ Manual ☐ Simulation

Please select Simulation Speed: ☒ Slow ☐ Moderate ☐ Fast

Please select Boardsize N x N: 5 x 5

Number of solutions found: 0



vsit.edu.in says
Solution found (1 out of 10)
OK

Simulation of 4 QUEENS/ N QUEENS

The problem here is to place queens on the empty chessboard in such a way that no queen attacks any other queen. Select Play mode as manual and place the queen on the board to get the solution. Or select Play mode as Simulation to get the solution directly.

Please select the play mode: ☒ Manual ☐ Simulation

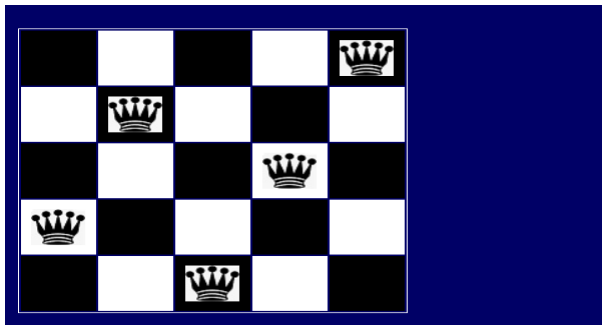
Please select Simulation Speed: ☒ Slow ☐ Moderate ☐ Fast

Please select Boardsize N x N: 5 x 5

Number of solutions found: 1
[1352]



Another Solution :



Time Complexity Derivation of N Queen problem:

The time complexity of $O(N!)$ for the N-Queens problem is derived from the fact that for each placement of the queens, the number of possibilities decreases exponentially.

Here's how the time complexity of $O(N!)$ is explained using your provided reasoning:

1. **First queen placement:** The first queen can be placed in any of the N columns on the first row. So, there are N possibilities for the placement of the first queen.
2. **Second queen placement:** For each placement of the first queen, the second queen must not be in the same column or diagonally aligned with the first queen. This leaves $N-1$ possibilities for placing the second queen on the second row.
3. **Third queen placement:** Similarly, for each placement of the first two queens, the third queen must not be in the same column or diagonally aligned with either of the first two queens. This leaves $N-2$ possibilities for placing the third queen on the third row.
4. **And so on...:** This process continues until all N queens are placed on the N rows of the chessboard, each time reducing the number of possibilities by one.

So, the total number of possibilities for placing N queens on an $N \times N$ chessboard is $N \times (N-1) \times (N-2) \times \dots \times 2 \times 1$, which is precisely $N!$.

Therefore, the time complexity is $O(N!)$, indicating that the number of possibilities grows factorially with the size of the problem N , making it very large and impractical for large values of N .

Program(s) of N Queen problem:

```
#include <stdbool.h>
```

```
#include <stdio.h>
```

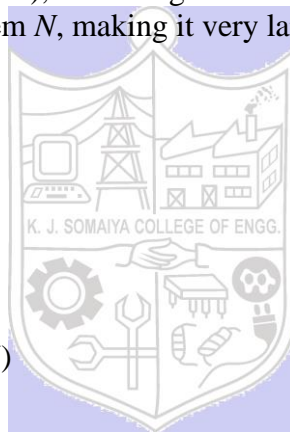
```
#include <stdlib.h>
```

```
void printSolution(int **board, int N)
```

```
{
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
        {
            if (board[i][j])
                printf("Q ");
            else
                printf(". ");
        }
        printf("\n");
    }
}
```

```
bool isSafe(int **board, int row, int col, int N)
```

```
{
    int i, j;
    for (i = 0; i < col; i++)
        if (board[row][i])
            return false;
}
```



```

for (i = row, j = col; i >= 0 && j >= 0; i--, j--)
    if (board[i][j])
        return false;
for (i = row, j = col; j >= 0 && i < N; i++, j--)
    if (board[i][j])
        return false;
return true;
}

```

```

bool solveNQUtil(int **board, int col, int N)
{

```

```

    if (col >= N)
        return true;
    for (int i = 0; i < N; i++)
    {
        if (isSafe(board, i, col, N))
        {
            board[i][col] = 1;
            if (solveNQUtil(board, col + 1, N))
                return true;
            board[i][col] = 0; // BACKTRACK
        }
    }
    return false;
}

```

```

bool solveNQ(int N)
{

```

```

    int **board = (int **)malloc(N * sizeof(int *));
    for (int i = 0; i < N; i++)
    {
        board[i] = (int *)malloc(N * sizeof(int));
        for (int j = 0; j < N; j++)
        {
            board[i][j] = 0;
        }
    }
}

```

```

if (solveNQUtil(board, 0, N) == false)
{
    printf("Solution does not exist\n");
    return false;
}
printSolution(board, N);
for (int i = 0; i < N; i++)
{
    free(board[i]);
}
free(board);
return true;

```




```

}

int main()
{
    int N;
    printf("Enter the size of the chessboard (N): ");
    scanf("%d", &N);
    if (N <= 0)
    {
        printf("Invalid board size\n");
        return 1;
    }
    solveNQ(N);
    return 0;
}

```

Output(o) of N Queen problem:

```

Enter the size of the chessboard (N): 4
. . Q .
Q . . .
. . . Q
. Q . .

=== Code Execution Successful ===

```

```

Enter the size of the chessboard (N): 5
Q . . . .
. . . Q .
. Q . . .
. . . . Q
. . Q . .

=== Code Execution Successful ===

```

```

Enter the size of the chessboard (N): 8
Q . . . . . . .
. . . . . Q .
. . . . Q . . .
. . . . . . Q
. Q . . . . . .
. . . Q . . . .
. . . . . Q . .
. . Q . . . . .

=== Code Execution Successful ===

```

Post Lab Questions:- Explain the process of backtracking ? What are the advantages of backtracking as against brute force algorithms? What do you mean by P and NP Problem?

Ans)

Backtracking is a systematic method to find all possible solutions to a problem by incrementally building candidates and abandoning a candidate as soon as it's determined that it cannot possibly lead to a valid solution. It involves trying out different choices and exploring each choice until a valid solution is found or all choices have been exhausted.

Here's a general process for backtracking:

1. **Choose:** Make a choice for the current decision point.
2. **Explore:** Recur to explore the remaining options starting from the choice made.
3. **Backtrack:** If the exploration doesn't lead to a solution, undo the choice made and go back to the previous decision point to explore other options.

Advantages of backtracking over brute force algorithms:

1. **Efficiency:** Backtracking can often significantly reduce the search space compared to a brute force approach. It prunes the search tree by abandoning paths that can't lead to a solution.
2. **Memory:** Backtracking usually requires less memory compared to brute force as it only needs to store the current path being explored rather than storing all possible combinations.
3. **Optimization:** Backtracking allows for the incorporation of heuristics or intelligent decision-making at each step to further optimize the search process.

P and NP Problems:

P and NP are classes of decision problems in computational complexity theory.

- **P Problems:** These are decision problems that can be solved by a deterministic Turing machine (an abstract computing machine) in polynomial time. In simpler terms, P problems are those for which a solution can be found efficiently. Example: Sorting a list of numbers in ascending order using algorithms like quicksort or mergesort.
- **NP Problems:** These are decision problems for which a given solution can be verified quickly (in polynomial time) but there's no known efficient way to find a solution. NP stands for "nondeterministic polynomial time." Example: The Traveling Salesman Problem, where the task is to find the shortest possible route that visits each city exactly once and returns to the original city.

The central question in computational complexity theory is whether P equals NP, meaning whether every problem whose solution can be quickly verified can also be solved quickly. This is one of the most famous open problems in computer science and mathematics. If P equals NP, it would imply that every NP problem has a polynomial-time algorithm, making a wide range of currently difficult problems efficiently solvable.

Outcome:

CO3 Implement Backtracking and Branch-and-bound algorithms

Conclusion: (Based on the observations):

Through this experiment, we learnt the concept of N-Queens Problem and use of Backtracking approach to solve the N-Queens Problem. We also implemented a C program for solving the N-Queens Problem using the backtracking approach and hence understood its time complexity. We also understood the advantages of backtracking as against brute force algorithms and the concept of P and NP Problem.

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. Cormen ,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.

