Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Name	Tejas Jadhav
UID No.	2022301006
Class	COMPS A (B batch)
Experiment No.	7

Aim: To use Backtracking to solve the N Queen problem

Theory:

- **Backtracking**

Backtracking is an algorithmic technique for solving problems recursively by trying to build a solution incrementally, one piece at a time, removing those solutions that fail to satisfy the constraints of the problem at any point in time.

Backtracking can also be said as an improvement to the brute force approach. So basically, the idea behind the backtracking technique is that it searches for a solution to a problem among all the available options.

Initially, we start the backtracking from one possible option and if the problem is solved with that selected option then we return the solution else we backtrack and select another option from the remaining available options. There also might be a case where none of the options will give you the solution and hence we understand that backtracking won't give any solution to that particular problem

Algorithm:

is_safe(chessboard, row, col, n)

```
1. for i = 0 to row - 1
2.     if chessboard[i][col] == 1
3.         return false
4. for i = row, j = col; i >= 0 and j >= 0; i--, j--
5.     if chessboard[i][j] == 1
6.         return false
7. for i = row, j = col; i >= 0 and j < n; i--, j++
8.     if chessboard[i][j] == 1
```

9. return false
10. return true

solve_nqueen(chessboard, row, n)

1. if row == n
2. print "Solution " ++ sol_count ++ ":"
3. print_chessboard(chessboard, n)
4. return
5. for i = 0 to n - 1
6. if is_safe(chessboard, row, i, n)
7. chessboard[row][i] = 1
8. solve_nqueen(chessboard, row + 1, n)
9. chessboard[row][i] = 0

Code:

```
#include <iostream>
using namespace std;

int sol_count = 0;

int** create_chessboard(int n) {
    int** chessboard = new int*[n];
    for (int i = 0; i < n; i++) {
        chessboard[i] = new int[n];
    }
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            chessboard[i][j] = 0;
        }
    }
    return chessboard;
}

void print_chessboard(int** chessboard, int n) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
```

```
            if (chessboard[i][j] == 1) {
                cout << "Q ";
            } else {
                cout << "- ";
            }
        }
        cout << endl;
    }
}

bool is_safe(int** chessboard, int row, int col, int n) {
    // check if there is a queen in the same column
    for (int i = 0; i < row; i++) {
        if (chessboard[i][col] == 1) {
            return false;
        }
    }
    // check if there is a queen in the upper left diagonal
    for (int i = row, j = col; i >= 0 && j >= 0; i--, j--) {
        if (chessboard[i][j] == 1) {
            return false;
        }
    }
    // check if there is a queen in the upper right diagonal
    for (int i = row, j = col; i >= 0 && j < n; i--, j++) {
        if (chessboard[i][j] == 1) {
            return false;
        }
    }

    return true;
}

void solve_nqueen(int** chessboard, int row, int n) {
    if (row == n) {
        cout << "Solution " << ++sol_count << ":" << endl;
        print_chessboard(chessboard, n);
        cout << endl;
        return;
    }
```

```
    for (int i = 0; i < n; i++) {
        if (is_safe(chessboard, row, i, n)) {
            chessboard[row][i] = 1;
            solve_nqueen(chessboard, row + 1, n);
            chessboard[row][i] = 0;
        }
    }
}
int main() {
    int n;

    cout << "Enter the number of queens: ";
    cin >> n;
    int** chessboard = create_chessboard(n);
    solve_nqueen(chessboard, 0, n);

    if (sol_count == 0) {
        cout << "No solution found." << endl;
    }
    return 0;
}
```

Output:

1) All Solution for 4 queens

```
PS D:\Tejas\clg\daa\Experiment 07\code> g++ .\nqueen.cpp
PS D:\Tejas\clg\daa\Experiment 07\code> ./a
Enter the number of queens: 4
Solution 1:
- Q - -
- - - Q
Q - - -
- - Q -

Solution 2:
- - Q -
Q - - -
- - - Q
- Q - -
```

2) Solution for 8 Queen

Starting 5 solutions

```
PS D:\Tejas\clg\daa\Experiment 07\code> ./a
Enter the number of queens: 8
Solution 1:
Q - - - - - - -
- - - - Q - - -
- - - - - - - Q
- - - - - Q - -
- - Q - - - - -
- - - - - - Q -
- Q - - - - - -
- - - Q - - - -

Solution 2:
Q - - - - - - -
- - - - - Q - -
- - - - - - - Q
- - Q - - - - -
- - - - - - Q -
- - - Q - - - -
- Q - - - - - -
- - - - Q - - -

Solution 3:
Q - - - - - - -
- - - - - - Q -
- - - Q - - - -
- - - - - Q - -
- - - - - - - Q
- Q - - - - - -
- - - - Q - - -
- - Q - - - - -
```

```
Solution 4:
Q - - - - - - -
- - - - - - Q -
- - - - Q - - -
- - - - - - - Q
- Q - - - - - -
- - - Q - - - -
- - - - - Q - -
- - Q - - - - -

Solution 5:
- Q - - - - - -
- - - Q - - - -
- - - - - Q - -
- - - - - - - Q
- - Q - - - - -
Q - - - - - - -
- - - - - - Q -
- - - - Q - - -
```

Ending 5 solutions

```
Solution 88:
- - - - - - Q -
- - - - Q - - -
- - Q - - - - -
Q - - - - - - -
- - - - - Q - -
- - - - - - - Q
- Q - - - - - -
- - - Q - - - -

Solution 89:
- - - - - - - Q
- Q - - - - - -
- - - Q - - - -
Q - - - - - - -
- - - - - - Q -
- - - - Q - - -
- - Q - - - - -
- - - - - Q - -
```

```
Solution 90:
- - - - - - - - Q
- Q - - - - - - -
- - - - Q - - - -
- - Q - - - - - -
Q - - - - - - - -
- - - - - - - Q -
- - - Q - - - - -
- - - - - Q - - -

Solution 91:
- - - - - - - - Q
- - Q - - - - - -
Q - - - - - - - -
- - - - - Q - - -
- Q - - - - - - -
- - - - Q - - - -
- - - - - - - Q -
- - - Q - - - - -

Solution 92:
- - - - - - - - Q
- - - Q - - - - -
Q - - - - - - - -
- - Q - - - - - -
- - - - - Q - - -
- Q - - - - - - -
- - - - - - - Q -
- - - - Q - - - -
```

2 Intermediate solution:

Total solution count: 92

Observation:

- The n-queen problem has a significant computational complexity for large board sizes.
- The number of solutions increases rapidly with increasing board size.
- Backtracking algorithms like the one used in this implementation can efficiently find all solutions.

The time complexity is O(n!) as n queen is an combinatorics problem, we are just arranging n items.

Conclusion:

In conclusion, the n-queen problem is a challenging and computationally complex problem. However, with backtracking algorithms, it is possible to efficiently find all solutions for small to moderate board sizes.