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

Batch: A3 **Roll No.: 16010121045**

Experiment No. 8

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of the Staff In-charge with date

Title: Implementation of N-Queen Problem using Backtracking Algorithm

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.
CO 3	Analyze and solve problems for different string matching algorithms.

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 kth row and ith 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);
 }
```

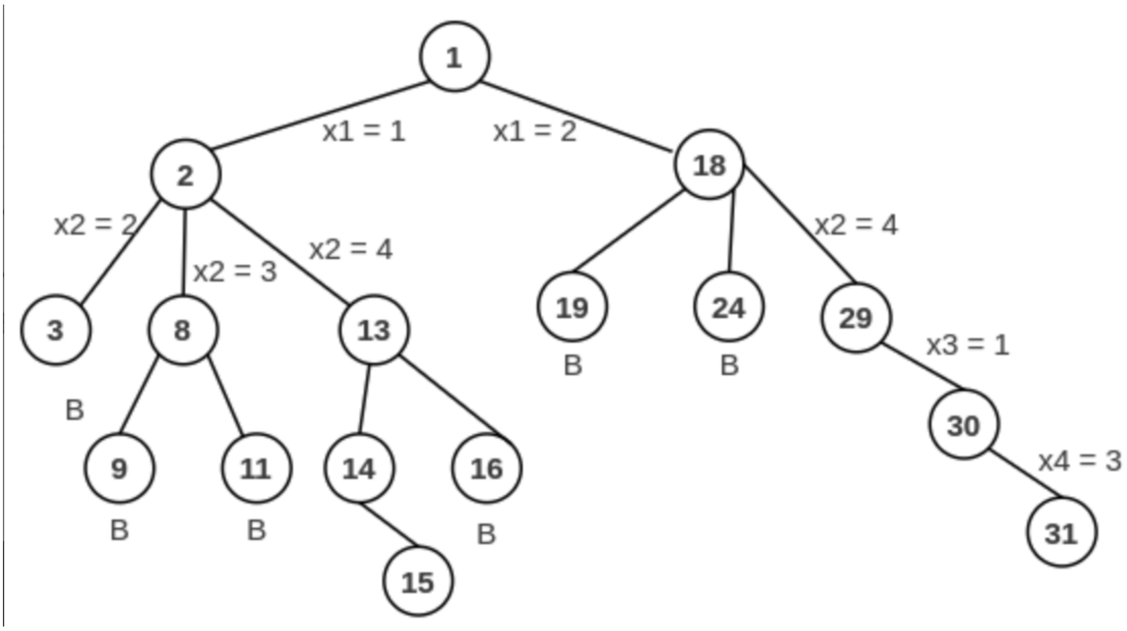
Example 8-Queens Problem:

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.

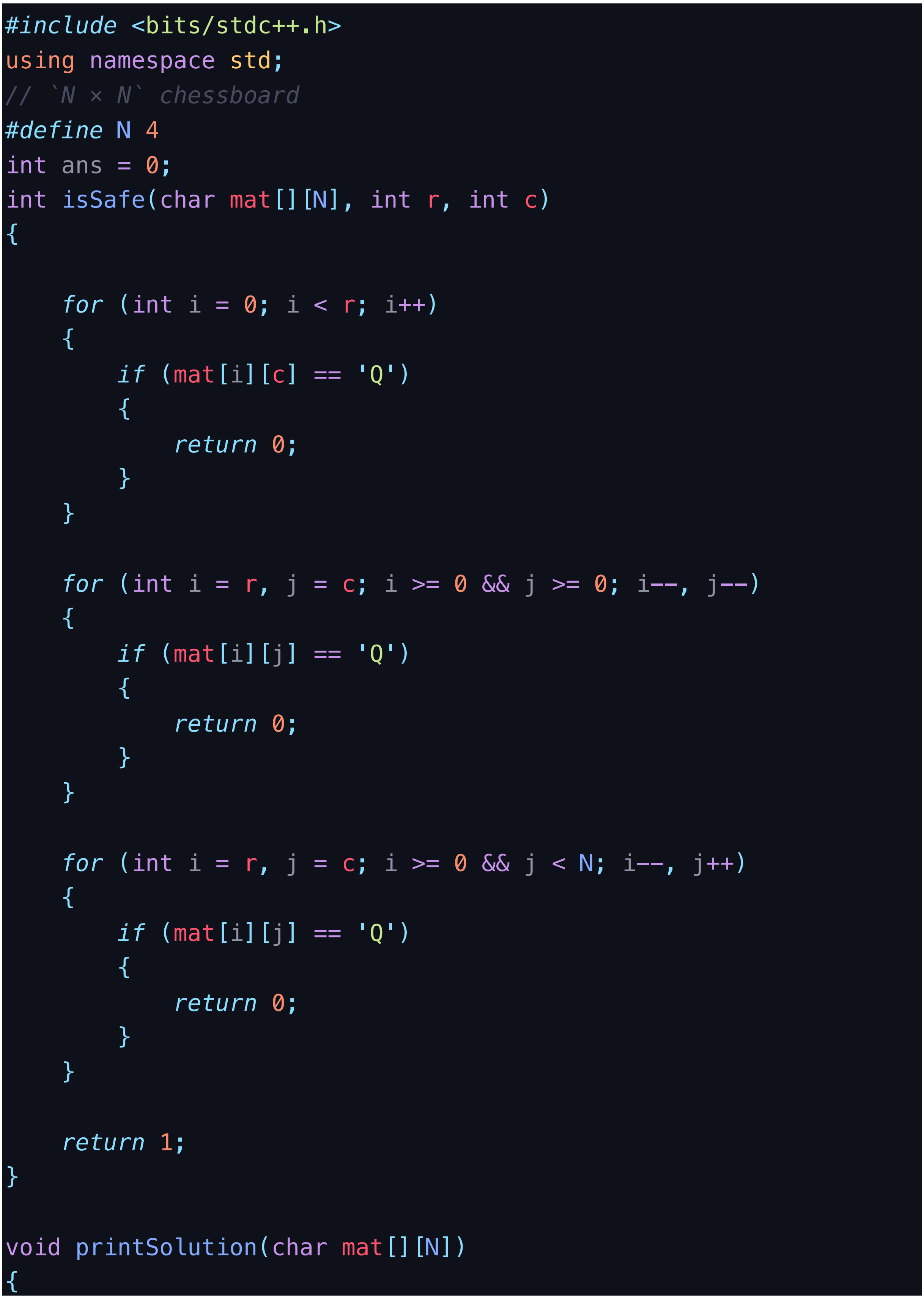
Solution Using Backtracking Approach:

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.

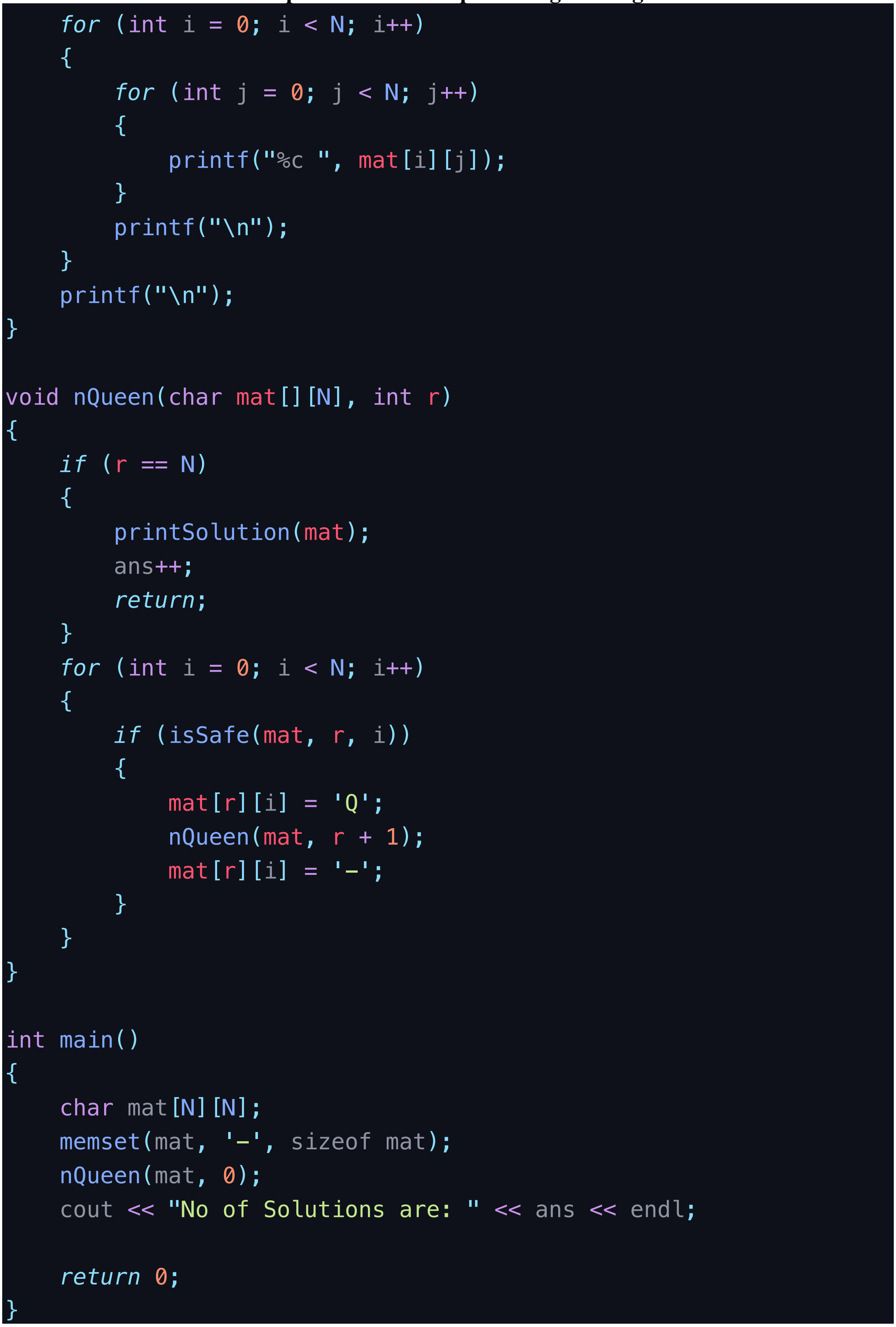
<u>State Space tree for N-Queens (Solution):</u>



Implementation (Code):

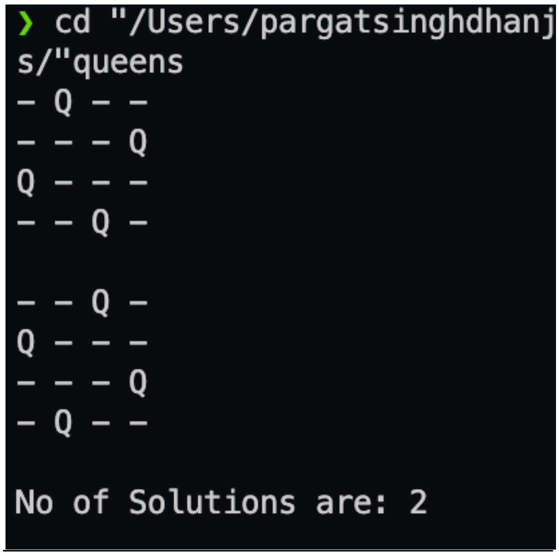


```
#include <bits/stdc++.h>
using namespace std;
// `N × N` chessboard
#define N 4
int ans = 0;
int isSafe(char mat[][N], int r, int c)
{

    for (int i = 0; i < r; i++)
    {
        if (mat[i][c] == 'Q')
        {
            return 0;
        }
    }

    for (int i = r, j = c; i >= 0 && j >= 0; i--, j--)
    {
        if (mat[i][j] == 'Q')
        {
            return 0;
        }
    }

    for (int i = r, j = c; i >= 0 && j < N; i--, j++)
    {
        if (mat[i][j] == 'Q')
        {
            return 0;
        }
    }

    return 1;
}

void printSolution(char mat[][N])
{
```



```
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
        {
            printf("%c ", mat[i][j]);
        }
        printf("\n");
    }
    printf("\n");
}

void nQueen(char mat[][N], int r)
{
    if (r == N)
    {
        printSolution(mat);
        ans++;
        return;
    }
    for (int i = 0; i < N; i++)
    {
        if (isSafe(mat, r, i))
        {
            mat[r][i] = 'Q';
            nQueen(mat, r + 1);
            mat[r][i] = '-';
        }
    }
}

int main()
{
    char mat[N][N];
    memset(mat, '-', sizeof mat);
    nQueen(mat, 0);
    cout << "No of Solutions are: " << ans << endl;

    return 0;
}
```

OUTPUT:



```
❯ cd "/Users/pargatsinghdhanj
s/"queens
- Q - -
- - - Q
Q - - -
- - Q -

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

No of Solutions are: 2
```

Algorithm:

0) Make a board, make a space to collect all solution states.
1) Start in the topmost row.
2) Make a recursive function which takes state of board and the current row number as its parameter.
3) Fill a queen in a safe place and use this state of board to advance to next recursive call, add 1 to the current row. Revert the state of board after making the call.
  a) Safe function checks the current column, left top diagonal and right top diagonal.
  b) If no queen is present then fill else return false and stop exploring that state and track back to the next possible solution state
4) Keep calling the function till the current row is out of bound.
5) If current row reaches the number of rows in the board then the board is filled.
6) Store the state and return.

Analysis of Backtracking solution:

Time Complexity: O(N!)

Auxiliary Space: O(N2)

CONCLUSION:

Successfully implemented the given problem using backtracking algorithm.