

| **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. |



**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**](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**](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**](http://www.mathcs.emory.edu/~cheung/Courses/170.2010/Syllabus/Backtracking/8queens.html)
7. [**http://www.geeksforgeeks.org/backtracking-set-3-n-queen-problem/**](http://www.geeksforgeeks.org/backtracking-set-3-n-queen-problem/)
8. [**http://www.hbmeyer.de/backtrack/achtdamen/eight.htm**](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);

}

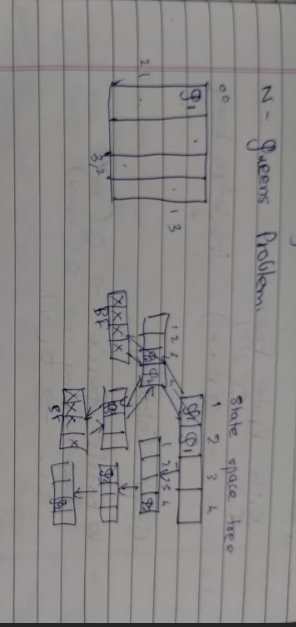
**Example 8-Queens Problem:**

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.

**State Space tree for N-Queens (Solution):**

****

**Implementation (Code):**

**#include <bits/stdc++.h>**

**using namespace std;**

**void func(int i, int n, vector<int> &col, vector<int> &d1, vector<int> &d2, vector<vector<char>> & board)**

**{**

**if(i == n)**

**{**

**cout << "\n\nSolution:\n\n";**

**for(int k = 0; k < n; k++)**

**{**

**for(int j = 0; j < n; j++)**

**{**

**cout << board[k][j] << " ";**

**}**

**cout << "\n";**

**}**

**return;**

**}**

**for(int j = 0; j < n; j++)**

**{**

**if(!col[j] && !d1[abs(i + j)] && !d2[n+i-j])**

**{**

**//cout << i << j << endl;**

**board[i][j] = 'Q';**

**col[j] =1;**

**d1[abs(i + j)] = 1;**

**d2[n+i-j] = 1;**

**func(i + 1, n, col, d1, d2, board);**

**board[i][j] = '.';**

**col[j] =0;**

**d1[abs(i + j)] = 0;**

**d2[n+i-j] = 0;**

**}**

**}**

**}**

**int main()**

**{**

**cout << "Enter the size of board\n";**

**int n; cin >> n;**

**vector<int> col(n), d1(n+2), d2(2\* n+2);**

**vector<vector<char>> board(n, vector<char>(n, '.'));**

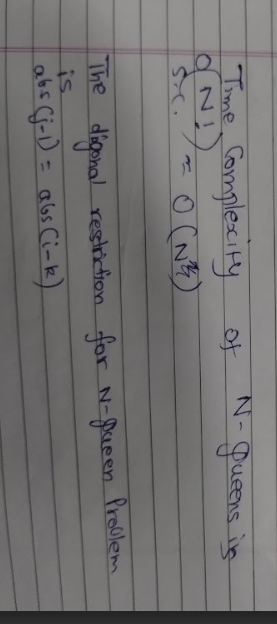
**func(0, n, col,d1,d2,board);**

**}**

**OUTPUT:**



**Analysis of Backtracking solution:**

****

**CONCLUSION:**

**We have found and analysed the solutions to the n-queens problem**