**Practical 7**

**Aim: - Backtracking**

7.1 Eight Queen Problem.

**Theory: -**

**What is Eight queens problem?**

Problem of placing 8 queens on 8\*8 chess board so that no queens attack each other. No two queens should share same row, column or diagonal.

**Concept:**

Each call places a queen in specific column.

State of the board from previous placement is known.

**Current Step backtracking:**

If placement in the column doesn’t give you the solution, then move to the next row in the same column.

**Previous Step backtracking:**

If all rows in a column have been checked, call terminates and backtracks to the previous column.

**Program: -**

**Code: -**

#include <iostream>

#include <cstdio>

#include <cstdlib>

#define N 8

using namespace std;

void printSolution(int board[N][N])

{

for (int i = 0; i < N; i++)

{

for (int j = 0; j < N; j++)

cout<<board[i][j]<<" ";

cout<<endl;

}

}

bool isSafe(int board[N][N], int row, int col)

{

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[N][N], int col)

{

if (col >= N)

return true;

for (int i = 0; i < N; i++)

{

if ( isSafe(board, i, col) )

{

board[i][col] = 1;

if (solveNQUtil(board, col + 1) == true)

return true;

board[i][col] = 0;

}

}

return false;

}

bool solveNQ()

{

int board[N][N] = {0};

if (solveNQUtil(board, 0) == false)

{

cout<<"Solution does not exist"<<endl;

return false;

}

printSolution(board);

return true;

}

int main()

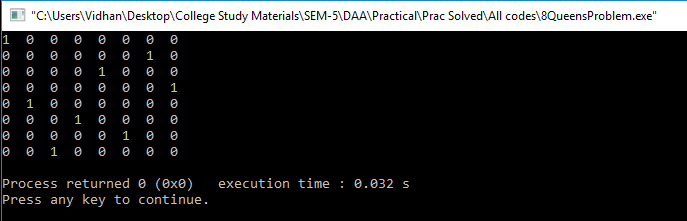
{

solveNQ();

return 0;

}

**Output: -**



**Conclusion: -**

The 8 Queens problem is a problem which uses the concept of backtracking and works with the time complexity of O (n^2).