

Worksheet 2.3

Student Name: Deepak Saini

Branch: 20BCC-1

Semester: 5th

Subject Name: Advance Programming Lab

UID: 20BCS4066

Section/Group: A

Date of Performance: 10-11-2022

Subject Code: 20CSP-334

1. Aim/Overview of the practical:

Implement N Queen's problem using Back Tracking.

2. Task to be done:

Implement N Queen's problem using Back Tracking.

3. Algorithm/Flowchart (For programming-based labs):

The N Queen is the problem of placing N chess queens on an $N \times N$ chessboard so that no two queens attack each other. 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.

1) Start in the leftmost column

2) If all queens are placed

return true

3) Try all rows in the current column.

Do following for every tried row.

a) 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.

b) If placing the queen in [row, column] leads to a solution then return true.

c) If placing queen doesn't lead to a solution then unmark this [row, column] (Backtrack) and go to step (a) to try other rows.

4) If all rows have been tried and nothing worked return false to trigger backtracking.

4. CODE:

```
/*N Queen Problem using backtracking */
#include <bits/stdc++.h>
#define N 4
using namespace std;

/* A utility function to print solution */
void printSolution(int board[N][N])
{
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++)
            cout << " " << board[i][j] << " ";
        printf("\n");
    }
}

/* A utility function to check if a queen can
be placed on board[row][col]. Note that this
function is called when "col" queens are
already placed in columns from 0 to col -1.
So we need to check only left side for
attacking queens */
bool isSafe(int board[N][N], int row, int col)
{
    int i, j;

    /* Check this row on left side */
    for (i = 0; i < col; i++)
        if (board[row][i])
            return false;

    /* Check upper diagonal on left side */
    for (i = row, j = col; i >= 0 && j >= 0; i--, j--)
        if (board[i][j])
            return false;

    /* Check lower diagonal on left side */
    for (i = row, j = col; j >= 0 && i < N; i++, j--)
        if (board[i][j])
            return false;

    return true;
}
```

/* A recursive utility function to solve N

Queen problem */

bool solveNQUtil(int board[N][N], int col)

{

 /* base case: If all queens are placed

 then return true */

 if (col >= N)

 return true;

 /* Consider this column and try placing

 this queen in all rows one by one */

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

 /* Check if the queen can be placed on

 board[i][col] */

 if (isSafe(board, i, col)) {

 /* Place this queen in board[i][col] */

 board[i][col] = 1;

 /* recur to place rest of the queens */

 if (solveNQUtil(board, col + 1))

 return true;

 /* If placing queen in board[i][col]

 doesn't lead to a solution, then

 remove queen from board[i][col] */

 board[i][col] = 0; // BACKTRACK

 }

 }

 /* If the queen cannot be placed in any row in

 this column col then return false */

 return false;

}

/* This function solves the N Queen problem using

Backtracking. It mainly uses solveNQUtil() to solve the problem.

It returns false if queens cannot be placed, otherwise, return true and prints placement of queens in the form of 1s.

Please note that there may be more than one

solutions, this function prints one of the feasible solutions.*/

bool solveNQ()

{

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

 { 0, 0, 0, 0 },

 { 0, 0, 0, 0 },

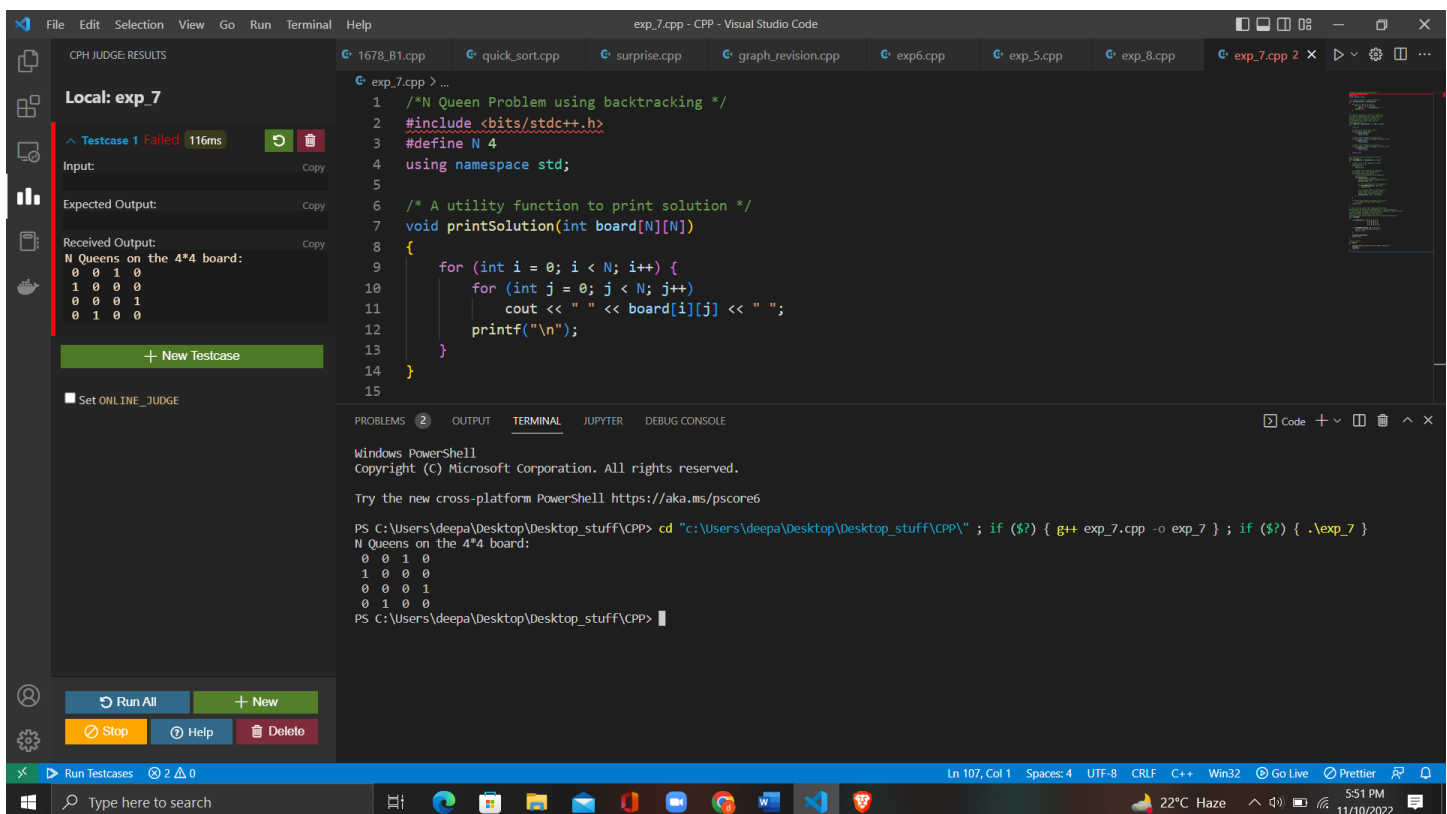
```
{ 0, 0, 0, 0 } };
```

```
if (solveNQUtil(board, 0) == false) {
    cout << "Solution does not exist";
    return false;
}
```

```
printSolution(board);
return true;
}
```

```
//main function
int main()
{
    cout<<"N Queens on the "<< N<<"* "<<N<<" board:\n";
    solveNQ();
    return 0;
}
```

5. Result/Output:



The screenshot shows the Visual Studio Code editor with a C++ file named `exp_7.cpp`. The code implements a backtracking algorithm to solve the N-Queens problem. The left sidebar shows the 'LOCAL: exp_7' section with a 'Testcase 1 Failed 116ms' status. The 'RECEIVED OUTPUT' section displays the output of the program for N=4:

```
N Queens on the 4*4 board:
0 0 1 0
1 0 0 0
0 0 0 1
0 1 0 0
```

The bottom terminal window shows the command prompt output, which matches the received output. The command used to run the program is:

```
PS C:\Users\deepa\Desktop\Desktop_stuff\CPP> cd "c:\Users\deepa\Desktop\Desktop_stuff\CPP\" ; if ($?) { g++ exp_7.cpp -o exp_7 } ; if ($?) { .\exp_7 }
```

```

PROBLEMS 2 OUTPUT TERMINAL JUPYTER DEBUG CONSOLE
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\deepa\Desktop\Desktop_stuff\CPP> cd "c:\Users\deepa\Desktop\Desktop_stuff\CPP\" ; if ($?) { g++ exp_7.cpp -o exp_7 } ; if ($?) { .\exp_7 }
N Queens on the 4*4 board:
0 0 1 0
1 0 0 0
0 0 0 1
0 1 0 0
PS C:\Users\deepa\Desktop\Desktop_stuff\CPP>

```

Learning Outcomes:

- To learn the implementation of Backtracking
- Examine the complexity of searching and sorting algorithms, and optimization through arrays, linked structures, stacks, queues, trees, and graphs.
- Implement N Queen's problem using Back Tracking.

Evaluation Grid (To be created as per the SOP and Assessment guidelines by the faculty):

Sr. No.	Parameters	Marks Obtained	Maximum Marks
1.			
2.			
3.			