TITLE: IMPLEMENTATION OF TOY PROBLEMS

Name : Ashish Prakash Singh
Register No : RA2011033010060
Subject : Artificial Intelligence

AIM:

To place 8 queens such that no queens attack each other by being in the same row or in the same column or on the same diagonal.

ALGORITHM :

Step 1 - Place the queen row-wise, starting from the left-most cell.

Step 2 - If all queens are placed then return true and print the solution matrix.

Step 3 - Else try all columns in the current row.

- ***Condition 1 - Check if the queen can be placed safely in this column then mark the current cell [Row, Column] in the solution matrix as 1 and try to check the rest of the problem recursively by placing the queen here leads to a solution or not.***
- ***Condition 2 - If placing the queen [Row, Column] can lead to the solution return true and print the solution for each queen's position.***
- ***Condition 3 - If placing the queen cannot lead to the solution then unmark***
- ***this [row, column] in the solution matrix as 0, BACKTRACK, and go back to condition 1 to try other rows.***

Step 4 - If all the rows have been tried and nothing worked, return false to trigger backtracking.

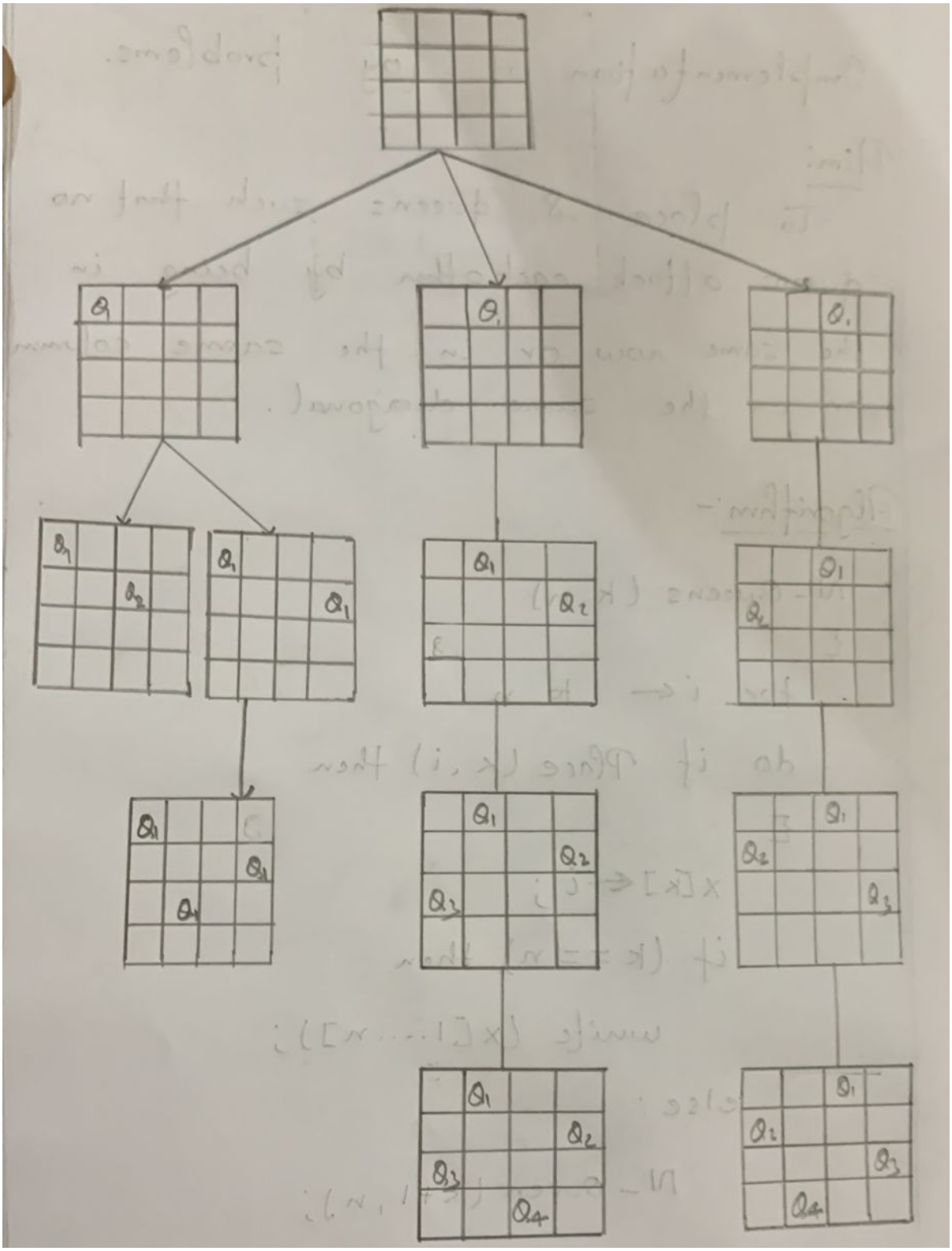
PROGRAM:

Get Started exp1.cpp

Exp1 > exp1.cpp > ...

```cpp
// RA2011033010060 - Ashish Prakash Singh
#include<iostream>
using namespace std;
#define N 8
void printBoard(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 isValid(int board[N][N], int row, int col) {
   for (int i = 0; i < col; i++) //check whether there is queen in the left or not
      if (board[row][i])
         return false;
   for (int i=row, j=col; i>=0 && j>=0; i--, j--)
      if (board[i][j]) //check whether there is queen in the left upper diagonal or not
         return false;
   for (int i=row, j=col; j>=0 && i<N; i++, j--)
      if (board[i][j]) //check whether there is queen in the left lower diagonal or not
         return false;
   return true;
}
bool solveNQueen(int board[N][N], int col) {
   if (col >= N) //when N queens are placed successfully
      return true;
   for (int i = 0; i < N; i++) { //for each row, check placing of queen is possible or not
      if (isValid(board, i, col) ) {
         board[i][col] = 1; //if validate, place the queen at place (i, col)
         if ( solveNQueen(board, col + 1)) //Go for the other columns recursively
            return true;
         board[i][col] = 0; //When no place is vacant remove that queen
      }
   }
   return false; //when no possible order is found
}
bool checkSolution() {
   int board[N][N];
   for(int i = 0; i<N; i++)
   for(int j = 0; j<N; j++)
   board[i][j] = 0; //set all elements to 0
   if ( solveNQueen(board, 0) == false ) { //starting from 0th column
      cout << "Solution does not exist";
      return false;
   }
   printBoard(board);
   return true;
}
int main() {
   checkSolution();
}
```

MANUAL CALCULATION:



OUTPUT:

RESULT:

The N-Queens problem was implemented and executed.