Name : Pratik Kanhaiya Rathod
Roll : 321056
PRN : 22010885
Batch : A2

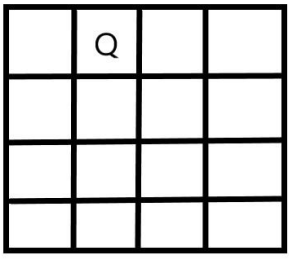
Assignment No. 3

Aim : Implement 8 queens problem using backtracking.

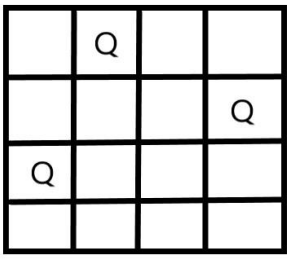
Theory:

- **What is backtracking?**

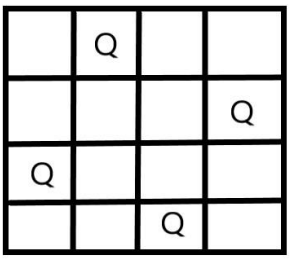
1) Backtracking is a general algorithmic technique that involves incrementally building a solution to a problem by trying different candidates, and then undoing the changes and backtracking to try other candidates if a candidate is found to be invalid or does not lead to a solution. It is typically used in solving combinatorial problems, such as finding all possible solutions to a puzzle or searching for a path through a maze.
2) The idea behind backtracking is to iteratively construct a potential solution to a problem, one step at a time, and then check whether the current partial solution is valid. If it is not valid, the algorithm "backtracks" to the previous step and tries a different candidate. This process continues until a complete solution is found or all possibilities have been exhausted.
3) Backtracking algorithms can be implemented using recursion or an iterative approach. They typically involve the use of a state space tree, which represents all possible solutions to a problem. The algorithm explores the tree by systematically trying all possible combinations of choices until a solution is found or it is determined that no solution exists.
4) Backtracking algorithms can be very effective for solving problems that involve a large search space, but they can also be quite time-consuming and resource-intensive. They are often used in combination with other techniques, such as pruning and heuristics, to optimize the search process and improve performance.



the next iteration of our algorithm will begin with the second column and start placing the queens again



queens 2 and 3 are easily placed onto the chessboard without creating the possibility of attacking one another.



the 4th queen has also been placed accordingly. Since the number of queens = 4, this solution will be printed.

Queens problem:

1. The 8 Queens Problem is a classic puzzle that involves placing eight queens on an 8x8 chessboard so that no two queens threaten each other. In other words, no two queens can be placed on the same row, column, or diagonal of the chessboard.
2. The problem was first proposed in the mid-1800s, and has since become a classic example of a constraint satisfaction problem, which is a type of problem in computer science and mathematics that involves finding a solution that satisfies a set of constraints. The 8 Queens Problem has many variations, such as using a larger or smaller chessboard or a different number of queens, but the basic challenge remains the same: to find a way to place the queens so that they are not attacking each other.

The 8 queens problem can be generalized and titled as 'n queens', since backtracking will always give each and every possible solution, the size of 'n' can vary and not just be 8 always.

Implementation :

```
#include <iostream>
#include <vector>
#include <chrono>

using namespace std;
using namespace std::chrono;

bool canPlaceQueen(int row, int col, vector<int>& queens) {
    for (int i = 0; i < row; i++) {
        if (queens[i] == col || abs(queens[i] - col) == abs(row - i)) {
            return false;
        }
    }
    return true;
}

void printBoard(int n, vector<int>& queens) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (queens[i] == j) {
                cout << "1 ";
            } else {
                cout << "0 ";
            }
        }
        cout << endl;
    }
```

```
    cout << endl;
}

bool solveNQueens(int n, int row, vector<int>& queens) {
    if (row == n) {
        printBoard(n, queens);
        return true;
    }
    for (int col = 0; col < n; col++) {
        if (canPlaceQueen(row, col, queens)) {
            queens[row] = col;
            if (solveNQueens(n, row + 1, queens)) {
                return true;
            }
        }
    }
    return false;
}

int main() {
    int n;
    cout<<"Enter the number of queen: ";
    cin>>n;
    vector<int> queens(n);
    auto start = high_resolution_clock::now();
    bool foundSolution = solveNQueens(n, 0, queens);
    auto stop = high_resolution_clock::now();
    auto duration = duration_cast<milliseconds>(stop - start);
    cout << "Time taken: " << duration.count() << " milliseconds" << endl;
    return 0;
}
```

Output :

```
PS C:\Users\PRATIK RATHOD\OneDrive\Desktop\TY\DAA\gitassg> cd "c:\Users\PRATIK RATHOD\OneDrive\Desktop\TY\DAA\gitassg\Assg3\" ; if ($?)
 { g++ NQueen.cpp -o NQueen } ; if ($?) { .\NQueen }
Enter the number of queen: 8
1 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 1
0 0 0 0 0 1 0 0
0 0 1 0 0 0 0 0
0 0 0 0 0 0 1 0
0 1 0 0 0 0 0 0
0 0 0 1 0 0 0 0

Time taken: 35 milliseconds
```

Analysis:

- Time complexity: O(n!). Since, algorithm tries to put queen at every cell.
- Space complexity: O(n^2) for storing the board. O(n) stack space for recursion.