Experiment No. : 7

Title: 8 Queen Problem using Backtracking

Batch:A2 **Roll No.: 16010421063**
Experiment No.:

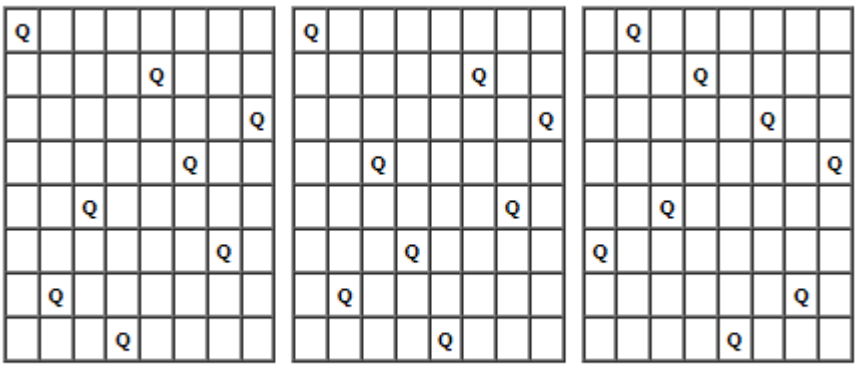
Aim: To Implement 8 Queen problem using Backtracking.

Algorithm of 8 Queen problem:
Let's go through the steps below to understand how this algorithm of solving the 8 queens problem using backtracking works:
START

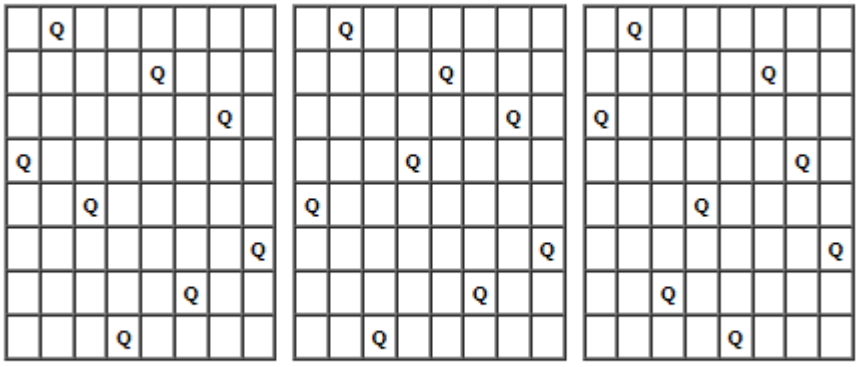
Step 1: Traverse all the rows in one column at a time and try to place the queen in that position.
Step 2: After coming to a new square in the left column, traverse to its left horizontal direction to see if any queen is already placed in that row or not. If a queen is found, then move to other rows to search for a possible position for the queen.
Step 3: Like step 2, check the upper and lower left diagonals. We do not check the right side because it's impossible to find a queen on that side of the board yet.
Step 4: If the process succeeds, i.e. a queen is not found, mark the position as '1' and move ahead.
Step 5: Recursively use the above-listed steps to reach the last column. Print the solution matrix if a queen is successfully placed in the last column.
Step 6: Backtrack to find other solutions after printing one possible solution.

END
Output :

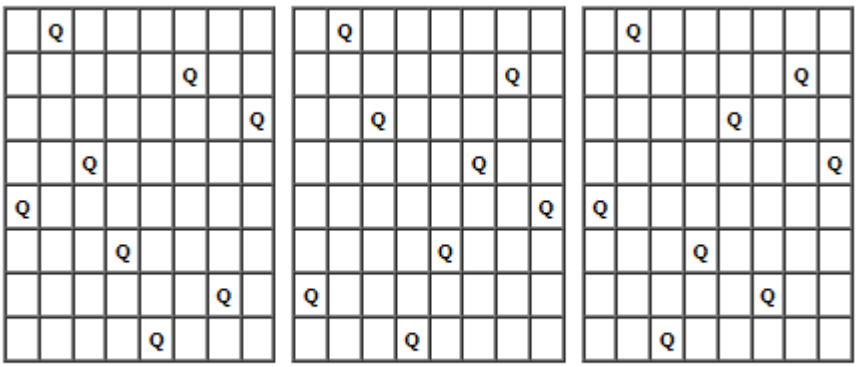
1. It asks in how many ways eight queens can be placed on a chess board so that no two attack each other.
2. Each of the twelve solutions shown on this page represents an equivalence class of solutions resulting from each other by rotating the chessboard and/or flipping it along one of its axes of symmetry.
3. Each of the equivalence classes represented by Solutions 1,2,...,11 consists of eight solutions. Since Solution 12 is invariant under rotating the chessboard by 180 degrees, its equivalence class consists of only four solutions.
4. Altogether, this page represents 92 solutions to the problem of eight queens; brute force shows that no other solutions exist.



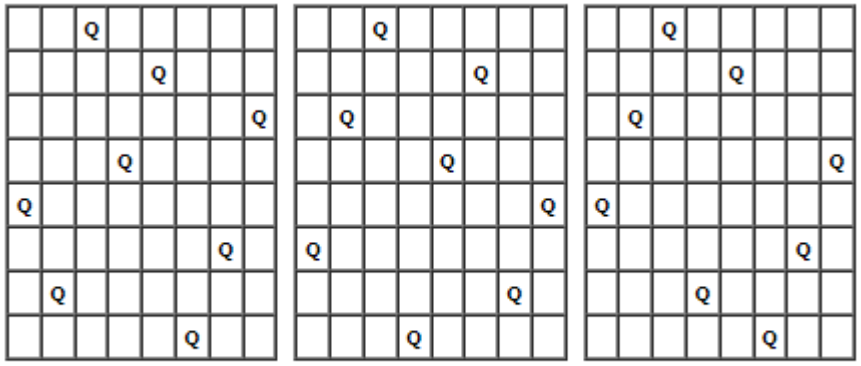
Solution 1 Solution 2 Solution 3



Solution 4 Solution 5 Solution 6



Solution 7 Solution 8 Solution 9



Solution 10 Solution 11 Solution 12

Working of 8 Queens Problem:

Problem Statement

Given an 8x8 chess board, you must place 8 queens on the board so that no two queens attack each other. Print all possible matrices satisfying the conditions with positions with queens marked with '1' and empty spaces with '0'. You must solve the 8 queens problem using backtracking.

Note 1: A queen can move vertically, horizontally and diagonally in any number of steps.

Note 2: You can also go through the N-Queen Problem for the general approach to solving this problem.

Solution

Derivation of 8 Queen problem:

Time complexity Analysis:
For the first column we will have N choices, then for the next column, we will have N-1 choices, and so on. Therefore the total time taken will be N*(N-1)*(N-2)...., which makes the time complexity to be O(N!).
the isPossible method takes O(n) time
for each invocation of loop in nQueenHelper, it runs for O(n) time
the isPossible condition is present in the loop and also calls nQueenHelper which is recursive
adding this up, the recurrence relation is:
T(n) = O(n^2) + n * T(n-1)
solving the above recurrence by iteration or recursion tree,
the time complexity of the nQueen problem is = O(N!)

Program(s) of 8 Queen problem:

```
#include <bits/stdc++.h>
#define int long long
using namespace std;
// vector<int> x(n,0);

int place(int Q, vector<int> &x)
{
   for (int i = 1; i < Q; i++)
   {
       if (x[i] == x[Q] || abs(x[Q] - x[i]) == Q - i)
           return 0;
   }
   return 1;
}

void queen(int n)
{
   int q = 1, i;
   vector<int> x(n, 0);
   while (q > 0)
   {
       x[q] = x[q] + 1;
       while (!place(q, x) && x[q] <= n)
       {
           x[q] = x[q] + 1;
```

```
        }
        if (x[q] > n)
        {
            q = q - 1;
        }
        else if (q == n)
        {
            for (i = 1; i <= n; i++)
            {
                cout << x[i] << " ";
            }
            cout<<"\n";
            for(int i=1;i<=n;i++){
                for(int j=1;j<=n;j++){
                    if(j==x[i])cout<<"Q ";
                    else cout<<"* ";
                }
                cout<<"\n";
            }

            cout << "\n";

        }
        else
        {
            q = q + 1;
            x[q] = 0;
        }
    }
}

int32_t main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
#ifndef ONLINE_JUDGE
    freopen("input.txt", "r", stdin);
    freopen("output.txt", "w", stdout);
#endif
    int n;
    cin >> n;

    queen(n);
```

```
}
```

Output(o) of 8 Queen problem:

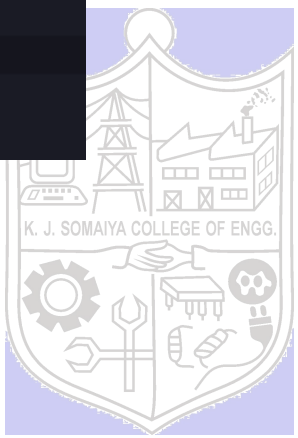
input.txt

```
4
```

output.txt

```
2 4 1 3
* Q * *
* * * Q
Q * * *
* * Q *

3 1 4 2
* * Q *
Q * * *
* * * Q
* Q * *
```



input.txt

```
5
```

output.txt

```
1 3 5 2 4
Q * * * *
* * Q * *
* * * * Q
* Q * * *
* * * Q *

1 4 2 5 3
Q * * * *
* * * Q *
* Q * * *
* * * * Q
* * Q * *

2 4 1 3 5
* Q * * *
* * * Q *
Q * * * *
* * Q * *
* * * * Q

2 5 3 1 4
* Q * * *
* * * * Q
* * Q * *
Q * * * *
* * * Q *

3 1 4 2 5
* * Q * *
Q * * * *
* * * Q *
* Q * * *
```

input.txt

```
1 8
```

output.txt

```
1 5 8 6 3 7 2 4
Q * * * * * * *
* * * * Q * * *
* * * * * * * Q
* * * * * Q * *
* * Q * * * * *
* * * * * * Q *
* Q * * * * * *
* * * Q * * * *

1 6 8 3 7 4 2 5
Q * * * * * * *
* * * * * Q * *
* * * * * * * Q
* * Q * * * * *
* * * * * * Q *
* * * Q * * * *
* Q * * * * * *
* * * * Q * * *

1 7 4 6 8 2 5 3
Q * * * * * * *
* * * * * * Q *
* * * Q * * * *
* * * * * Q * *
* * * * * * * Q
* Q * * * * * *
* * * * Q * * *
* * Q * * * * *

1 7 5 8 2 4 6 3
Q * * * * * * *
* * * * * Q * *
```

Post Lab Questions:- Describe the process of backtracking ? State the advantages of backtracking as against brute force algorithms?

Backtracking is a systematic algorithmic approach that is used to solve problems by exploring all possible solutions, and eliminating those solutions that do not satisfy the problem's constraints. It works by starting with an initial partial solution and then incrementally adding more components or variables until a complete solution is found, or all possible combinations have been explored.

The process of backtracking typically involves the following steps:

1. Define the problem and its constraints: The problem to be solved is defined, along with any constraints that must be satisfied in the solution.
2. Define the solution space: The solution space is the set of all possible solutions to the problem.
3. Choose a path and explore: Starting with an initial partial solution, we choose a path to explore and incrementally add more components or variables until we arrive at a complete solution or until we reach a dead end.
4. Backtrack and try another path: If we reach a dead end, we backtrack to the previous step and try another path.
5. Repeat until a solution is found or all paths have been explored: We repeat steps 3 and 4 until we either find a complete solution or exhaust all possible paths.

The advantage of backtracking over brute force algorithms is that it can often greatly reduce the search space by using constraints to eliminate large portions of the solution space that do not satisfy the problem's constraints. This can make the algorithm much more efficient and faster than brute force algorithms, which may need to explore all possible solutions in the solution space.

Backtracking is particularly useful in solving problems with multiple, interdependent decisions or variables, such as the n-queens problem, Sudoku, or the traveling salesman problem. By exploring the solution space incrementally and intelligently, backtracking can find a solution faster and more efficiently than other methods.

Conclusion: (Based on the observations):
Successfully understood the algorithm of N queens problem and implemented the same in CPP and printed out the entire chess board

---

Outcome: CO 3. Implement Backtracking and Branch-and-bound algorithms

---

References:

1. Richard E. Neapolitan, " Foundation of Algorithms ", 5th Edition 2016, Jones & Bartlett Students Edition
2. Harsh Bhasin , " Algorithms : Design & Analysis", 1st Edition 2013, Oxford Higher education, India
3. T.H. Coreman ,C.E. Leiserson,R.L. Rivest, and C. Stein, " Introduction to algorithms", 3rd Edition 2009, Prentice Hall India Publication
4. Jon Kleinberg, Eva Tardos, " Algorithm Design", 10th Edition 2013, Pearson India Education Services Pvt. Ltd.