Experiment No. : 7

Title: 8 Queen Problem using Backtracking

Batch: A2 **Roll No.: 16010421059** **Experiment No.: 7**

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.

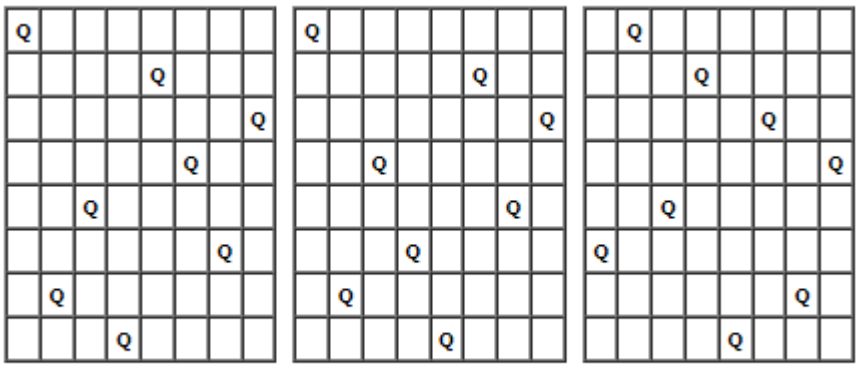
Step 5: Recursively use the above-listed steps to reach the last column. Print the solution matrix if a queen is successfully placed in the last column.

Step 6: Backtrack to find other solutions after printing one possible solution.

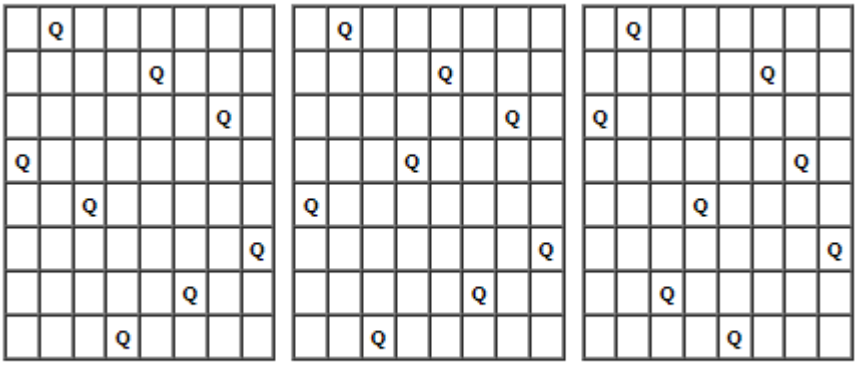
END

Output :

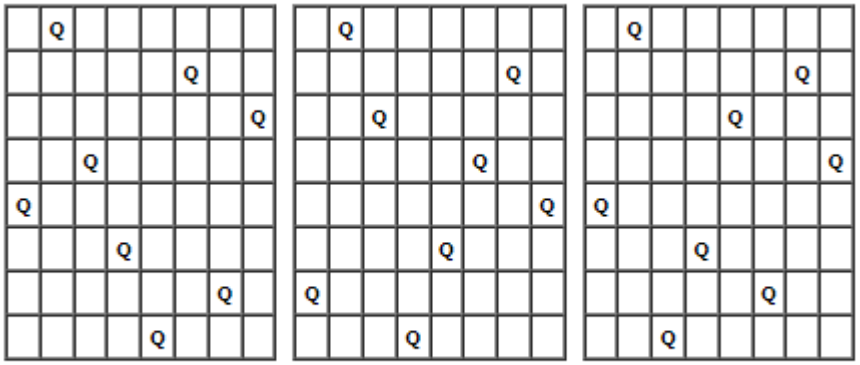
1. It asks in how many ways eight queens can be placed on a chess board so that no two attack each other.
2. Each of the twelve solutions shown on this page represents an equivalence class of solutions resulting from each other by rotating the chessboard and/or flipping it along one of its axes of symmetry.
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.



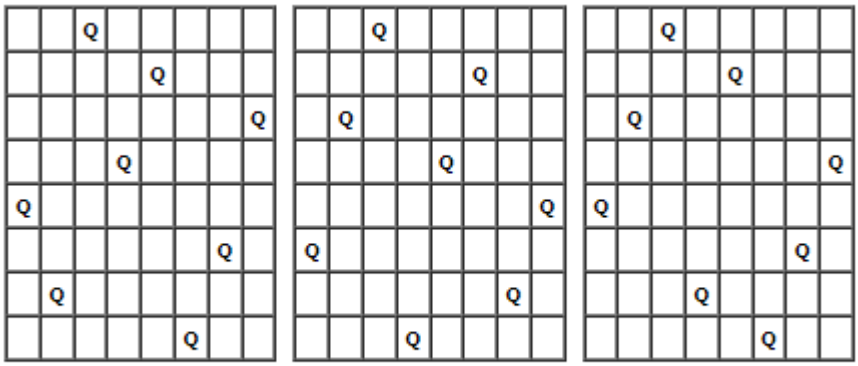
Solution 1 Solution 2 Solution 3



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.

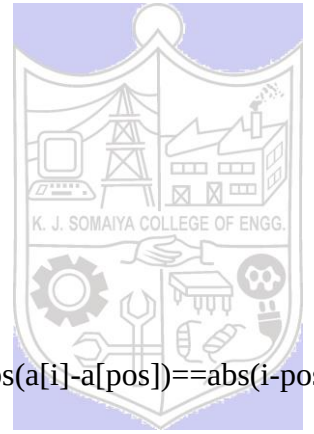
Note 1: A queen can move vertically, horizontally and diagonally in any number of steps.

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

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 is Possible method takes O(n) time
for each invocation of loop in n-Queen Helper, it runs for O(n) time
the is Possible condition is present in the loop and also calls n-Queen Helper 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 n-Queen problem is = O(N!)

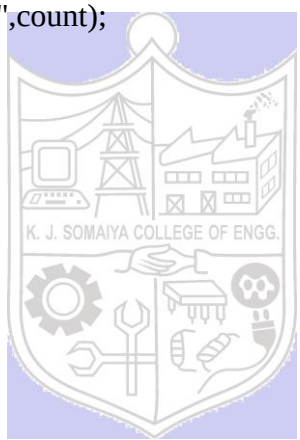
Program(s) of 8 Queen problem:



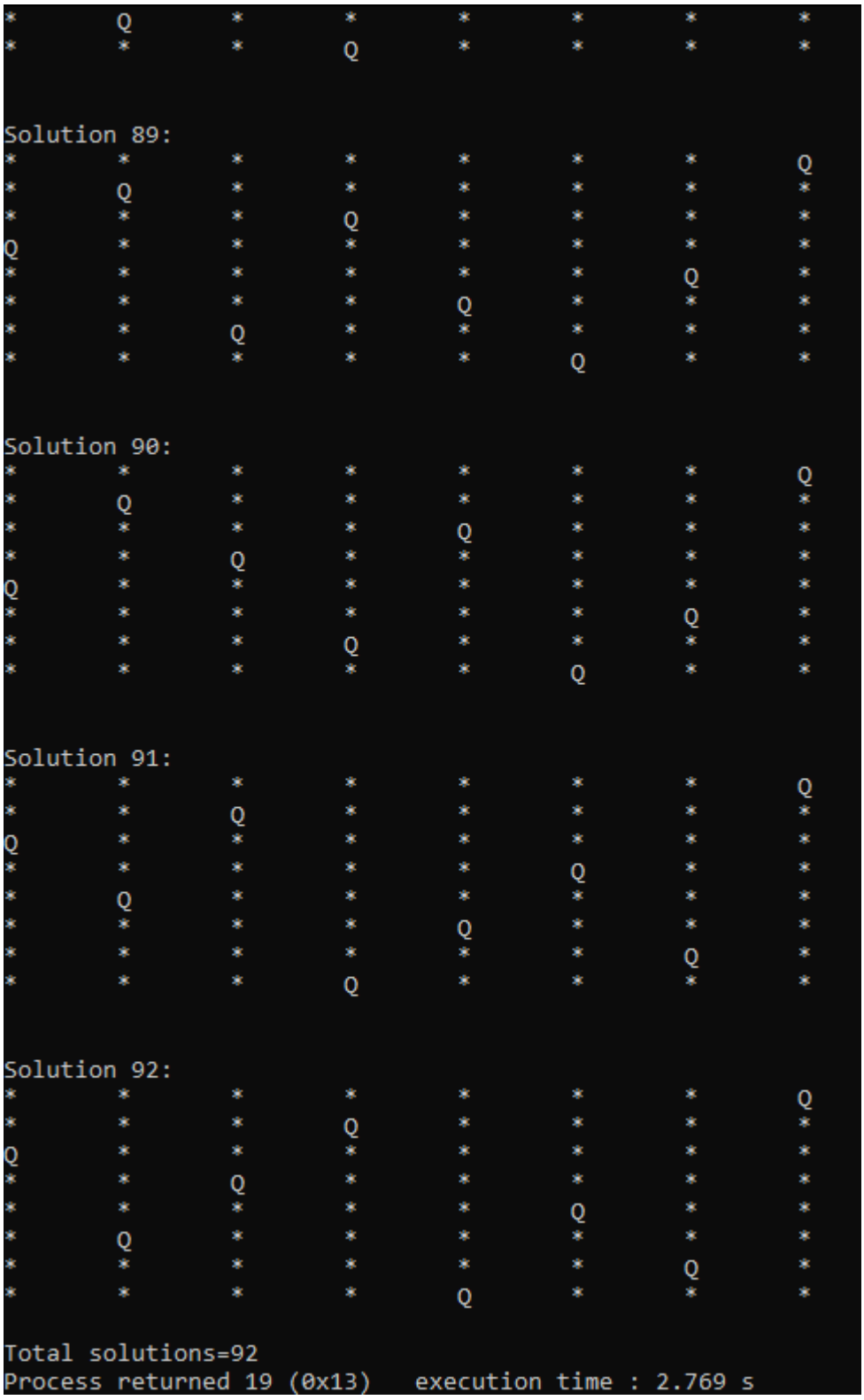

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
int a[30],count=0;
int place(int pos) {
        int i;
        for (i=1;i<pos;i++) {
                if((a[i]==a[pos])||((abs(a[i]-a[pos])==abs(i-pos))))
                  return 0;
        }
        return 1;
}
void print_sol(int n) {
        int i,j;
        count++;
        printf("\n\nSolution %d:\n",count);
        for (i=1;i<=n;i++) {
                for (j=1;j<=n;j++) {
                        if(a[i]==j)
                           printf("Q\t"); else
                           printf("*\t");
                }
                printf("\n");
        }
}
void queen(int n) {
        int k=1;
        a[k]=0;
```

```
        while(k!=0) {
                a[k]=a[k]+1;
                while((a[k]<=n)&&!place(k))
                  a[k]++;
                if(a[k]<=n) {
                        if(k==n)
                           print_sol(n); else {
                                k++;
                                a[k]=0;
                        }
                } else
                  k--;
        }
}
void main() {
        int i,n;
        printf("Enter the number of Queens\n");
        scanf("%d",&n);
        queen(n);
        printf("\nTotal solutions=%d",count);
        return 0;
}
```

Output(o) of 8 Queen problem:



```
Enter the number of Queens
8


Solution 1:
Q       *       *       *       *       *       *       *
*       *       *       *       Q       *       *       *
*       *       *       *       *       *       *       Q
*       *       *       *       *       Q       *       *
*       *       Q       *       *       *       *       *
*       *       *       *       *       *       Q       *
*       Q       *       *       *       *       *       *
*       *       *       Q       *       *       *       *


Solution 2:
Q       *       *       *       *       *       *       *
*       *       *       *       *       Q       *       *
*       *       *       *       *       *       *       Q
*       *       Q       *       *       *       *       *
*       *       *       *       *       *       Q       *
*       *       *       Q       *       *       *       *
*       Q       *       *       *       *       *       *
*       *       *       *       Q       *       *       *


Solution 3:
Q       *       *       *       *       *       *       *
*       *       *       *       *       *       Q       *
*       *       *       Q       *       *       *       *
*       *       *       *       *       Q       *       *
*       *       *       *       *       *       *       Q
*       Q       *       *       *       *       *       *
*       *       *       *       Q       *       *       *
*       *       Q       *       *       *       *       *


Solution 4:
Q       *       *       *       *       *       *       *
*       *       *       *       *       *       Q       *
*       *       *       *       Q       *       *       *
*       *       *       *       *       *       *       Q
*       Q       *       *       *       *       *       *
*       *       *       Q       *       *       *       *
*       *       *       *       *       Q       *       *
*       *       Q       *       *       *       *       *


Solution 5:
*       Q       *       *       *       *       *       *
*       *       *       Q       *       *       *       *
*       *       *       *       *       Q       *       *
```



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

Ans.

Backtracking is a general algorithmic technique that involves searching through all possible solutions to a problem by incrementally building a candidate solution and undoing the steps taken if a dead end is reached. It is used when we need to find all possible solutions to a problem or when we need to find the best solution among all possible solutions. Backtracking is commonly used in problems where we need to make a sequence of decisions, each of which affects the subsequent decisions.

The process of backtracking can be described as follows:

- Start with an empty solution.
- Choose the first variable to be part of the solution.
- Generate all possible values for the chosen variable.
- For each value, check if it satisfies the constraints of the problem.
- If the value satisfies the constraints, add it to the solution and move to the next variable.
- If the value does not satisfy the constraints, remove it from the solution and try the next value.
- If there are no more values to try for the current variable, backtrack to the previous variable and try the next value for that variable.
- Repeat steps 4 to 7 until a solution is found or all possibilities have been exhausted.

The advantages of backtracking over brute force algorithms include:

- Backtracking allows us to efficiently search through a large space of possible solutions by pruning branches of the search tree that cannot lead to a valid solution.
- Backtracking can be used to find all possible solutions to a problem, whereas brute force algorithms may only find one solution or a small subset of solutions.
- Backtracking can be faster than brute force algorithms in some cases because it avoids exploring all possible solutions by quickly discarding infeasible ones.
- Backtracking can be more memory-efficient than brute force algorithms because it does not store all possible solutions in memory. Instead, it only stores the current partial solution and the choices made so far.

Conclusion: (Based on the observations):

Implemented 8 Queen problem using Backtracking Successfully.

Outcome:

CO3: 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.