Name	Parth Gandhi
UID No.	2021300033
Class & Division	S.E. COMPS A (BATCH B)
Experiment No.	7

Aim: Experiment on solving N-Queen problem using backtracking.

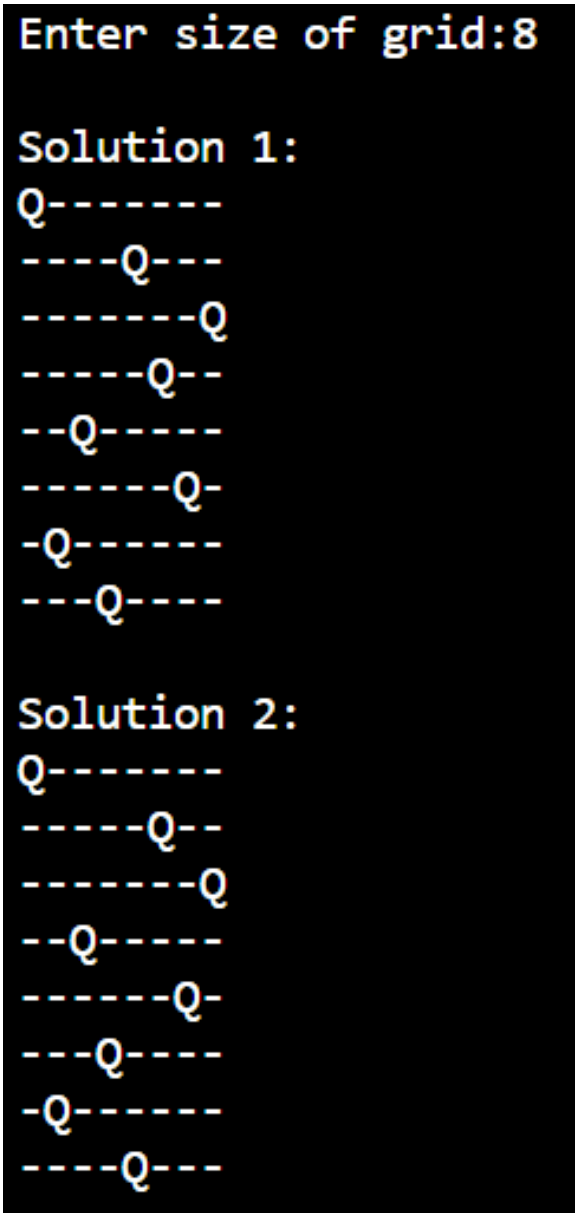
Theory:
The N-Queens Problem is a classic puzzle that involves placing N chess queens on an N x N chessboard so that no two queens threaten each other. That is, no two queens share the same row, column, or diagonal. In the N-Queens Problem, the constraint is that no two queens can threaten each other, which means that they cannot share the same row, column, or diagonal. The goal is to find a placement of N queens on the chessboard that satisfies this constraint. Solving the N-Queens Problem requires finding all possible placements of N queens that satisfy the constraint. This can be done using backtracking. The basic idea behind backtracking is to build a solution incrementally, one piece at a time, by trying all possible choices for the next piece, and then recursively trying all possible choices for the next piece until a solution is found or no more choices are available. If a candidate solution fails to satisfy the constraints of the problem at any point, the algorithm backtracks to the last decision point and tries a different choice.

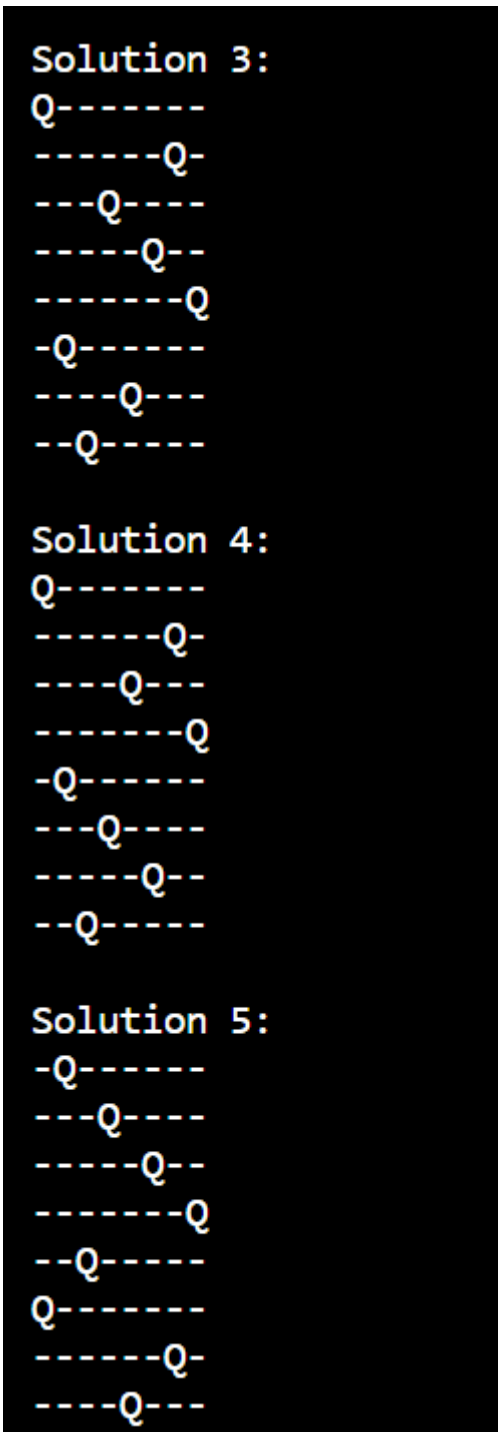
Algorithm:
1. Initialize an empty board of size n x n.
2. Initialize an empty list to store solutions.
3. Call the recursive solve() function with the initial parameters (0, board, solutions).
4. In the solve() function: a. If the current row is equal to n, add the current board to the list of solutions and return. b. For each column in the current row: i. If the current position is not under attack, mark it as a queen and call solve() for the next row. ii. If the current position is under attack, continue to the next column. iii. If no column can be found that does not result in an attack, backtrack to the previous row.
5. Return the list of solutions.
6. If the current row is equal to n, add the current board to the list of solutions and return.
7. For each column in the current row: a. If the current position is not under attack, mark it as a queen and call solve() for the next row. b. If the current position is under attack, continue to the next column. c. If no column can be found that does not result in an attack, backtrack to the previous row.
8. Return.

Code:
```
#include<bits/stdc++.h>
using namespace std;

int x[100],f=0;
bool place(int k,int i)
{
   for(int j=0;j<k;j++)
```

```
    {
        if(x[j]==i || abs(x[j]-i)==abs(j-k))
        return false;
    }
    return true;
}

void printsol(int n)
{
    cout<<"\nSolution "<<++f<<":"<<endl;
    char a[n][n];
    for(int i=0;i<n;i++)
    {
        for(int j=0;j<n;j++)
        a[i][j]='-';
        a[i][x[i]]='Q';
    }
    for(int i=0;i<n;i++)
    {
        for(int j=0;j<n;j++)
        cout<<a[i][j];
        cout<<endl;
    }
}

void nqueen(int k,int n)
{
    for(int i=0;i<n && f<5;i++)
    {
        if(place(k,i))
        {
            x[k]=i;
            if(k==n-1)
            printsol(n);
            else
            nqueen(k+1,n);
        }
    }
}

int main()
{
    int n;
    cout<<"Enter size of grid:";
    cin>>n;
    nqueen(0,n);
    return 0;
}
```

Output:





Conclusion: Successfully wrote a program to solve N-Queen problem using backtracking method.