Assignment 01

→ Problem Statement :

Position N Queens on a N*N chessboard such that no 2 queens are in the same row, column or diagnols.

→ Theory

A] Backtracking

a. Backtracking is a technique based on algorithm to solve problem. It uses recursive calling to find the solution by building the solution step by step increasing values with time. It removes the solutions that doesn't give rise to the solution of the problem based on the constraints given to solve the problem.

b. Backtracking solution for NQueens.

1. Start at First Row.
2. If all queens are placed
   then return True
3. Try all column in the current row.
   Do the following for every tried column.
   i. If the queen can be placed safely in this row then mark [row, col] as part of the solution & recursively check if

placing queen here leads to a solution.

ii. If placing the queen in [row, col] leads to a solution then return True.

iii. otherwise backtrack i.e unmark [row, col] and go to step (i).

4. If all queens have been placed return the solution.

5. If all cols have been tried and nothing worked, return false to trigger backtracking.

B] The NQueens Problem.

- In implementing the n-queens problem we imagine the chessboard as a 2-dimensional array
- The condition to test whether two queens at position (i, j) & (k, l) are on the same row; column is simply to check $l = k$ or $j = l$
- The conditions to check whether 2 queens are on the same diagnol is:

$$|j - i| = |i - k|$$

C] Brute Force Approach to Solve NQueens

- Generate all possible configurations of queens on board and print a configuration that satisfy the given constraints.
- while there are untried configurations {
  generate the next config.
  if queens don't attack then {
  print the config.
  }
  }

D] Pseudocode :

```
1.  Algorithm NQueens (k, n) {
2.  for i := 1 to n do {
3.      if (Place (k, i) then {
4.          x[k] := i
6.          if (k == n) then write (x[2 : n])
7.          else NQueens (k+1, n)
8.      }
9.  }
10. }
```

```
1.  Algorithm Place (k, i) {
2.      for j := 1 to k-1 {
3.          if ( abs (x[j] - i) == abs (j - k) or
                    x[j] == i)
4.              return false
```

```
5.    }
6.    return true.
7.  }
```

Program:

```
#include <iostream>
#include <cstdlib>
using namespace std;

int N;

void Solution(int arr[N]) {

  for(int i = 0; i < N; i++) {

    for(int j = 0; j < N; j++) {

      if(arr[i] == j)
        cout<<"Q"<<i+1<<" ";
      else
        cout<<" * ";
    }
    cout<<endl;
  }
}

bool Place(int arr[N], int Qi, int Qj) {

  for(int i = 0; i < Qi ; i++) {
    if(arr[i] == Qj || abs(arr[i] - Qj) == abs(i - Qi)) {
      return false;
    }
  }
  return true;
}

int NQueens(int arr[N], int i) {

    static int count = 0;
    for(int j = 0; j < N; j++) {

      if(Place(arr, i, j)) {

        arr[i] = j;
        if(i == N-1) {
          cout<<endl<<"******************"<<endl;
          cout<<"\nFinal Solution:\n\n";
          Solution(arr);
          cout<<endl<<"******************"<<endl<<endl;
          count++;
```

```
        }
        else {
          NQueens(arr, i + 1);
          for(int m = 0; m < N; m++) {
            cout<<m+1<<", "<<arr[m]+1<<"; ";
          }
          cout<<endl;
        }
      }
    }
  return count;
}

int main() {

  cout<<"\nHow many Queens troubling the King? ";
  cin>>N;

  int arr[N];

  for(int i = 0; i < N; i++) {
    arr[i] = -1;
  }

  if(N >= 3) {
    cout<<endl;
    int count = NQueens(arr, 0);
    cout<<endl<<"******************"<<endl;
    if(!count) {
      cout<<"\nSorry King, No Solution\n";
    }
    else if(count > 0) {
      cout<<"\nPossible Solutions: "<<count<<endl;
    }
    cout<<endl<<"******************"<<endl<<endl;
  }
  cout<<endl;
}
```

Output:

```
How many Queens troubling the King? 3

1, 1; 2, 3; 3, 0;
1, 1; 2, 3; 3, 0;
1, 2; 2, 3; 3, 0;
```

```
1, 3; 2, 1; 3, 0;
1, 3; 2, 1; 3, 0;

******************

Sorry King, No Solution

******************


Someshwars-MacBook-Pro:practice someshwargaikwad$ ./a.out

How many Queens troubling the King? 4

1, 1; 2, 3; 3, 0; 4, 0;
1, 1; 2, 4; 3, 2; 4, 0;
1, 1; 2, 4; 3, 2; 4, 0;
1, 1; 2, 4; 3, 2; 4, 0;

******************

Final Solution:

 * Q1  *  *
 *  *  * Q2
Q3  *  *  *
 *  * Q4  *

******************

1, 2; 2, 4; 3, 1; 4, 3;
1, 2; 2, 4; 3, 1; 4, 3;
1, 2; 2, 4; 3, 1; 4, 3;

******************

Final Solution:

 *  * Q1  *
Q2  *  *  *
 *  *  * Q3
 * Q4  *  *

******************

1, 3; 2, 1; 3, 4; 4, 2;
1, 3; 2, 1; 3, 4; 4, 2;
1, 3; 2, 1; 3, 4; 4, 2;
```

```
1, 4; 2, 1; 3, 3; 4, 2;
1, 4; 2, 1; 3, 3; 4, 2;
1, 4; 2, 2; 3, 3; 4, 2;
1, 4; 2, 2; 3, 3; 4, 2;

******************

Possible Solutions: 2

******************


Someshwars-MacBook-Pro:practice someshwargaikwad$
```

→ Conclusion :

Topics Covered :

1. Backtracking Concept.
2. NQueens Problem.
3. Solution for NQueens.
   i) Brute force
   ii) Backtracking.