EXPERIMENT 11

Experiment No: 11 **Date:** 05/05/2021

Aim: Implementation of N-Queens problem (Using Backtracking)

Theory:

N- Queen Problem

- N-Queens problem is one of the most common examples of backtracking.
- Our goal is to arrange N queens on an NxN chessboard such that no queen can strike down any other queen.
- A queen can attack horizontally, vertically, or diagonally.
- So, we start by placing the first queen anywhere arbitrarily and then place the next queen in any of the safe places.
- We continue this process until the number of unplaced queens becomes zero (a solution is found) or no safe place is left.
- If no safe place is left, then we change the position of the previously placed queen.

EXPERIMENT 11

Backtracking Algorithm

- The idea is to place queens one by one in different columns, starting from the leftmost column.
- When we place a queen in a column, we check for clashes with already placed queens.
- In the current column, if we find a row for which there is no clash, we mark this row and column as part of the solution.
- If we do not find such a row due to clashes then we backtrack and return false.

Algorithm Writing

- Start in the leftmost column
- If all queens are placed

  return true

- Try all rows in the current column.
- Do following for every tried row.
  - If the queen can be placed safely in this row then mark this [row, column] as part of the solution and recursively check if placing queen here leads to a solution.
  - If placing the queen in [row, column] leads to a solution then return true.

EXPERIMENT 11

- If placing queen doesn't lead to a solution then unmark this [row, column] (Backtrack) and go to step (a) to try other rows.
- If all rows have been tried and nothing worked, return false to trigger backtracking.

Algorithm

```
Algorithm NQueens(k,n)
// Using backtracking, this procedre prints all
// possible placements of n queens on an n x n
//chessboard so that they are nonattacking
{
    for i := 1 to n do
    {
        if Place(k,i)then
        {
            x[k] := i;
            if (k=n) then write (x[1:n]);
            else Nqueens(k + 1,n);
        }
    }
}
```

EXPERIMENT 11

Time Complexity

- The isSafe method takes O(N) time as it iterates through our array every time.
- For each invocation of the placeQueen method, there is a loop which runs for O(N) time.
- In each iteration of this loop, there is isSafe invocation which is O(N) and a recursive call with a smaller argument.
- If we add all this up and define the run time as T(N).
- Then ***T(N) = O(N2) + N*T(N-1).***
- If you draw a recursion tree using this recurrence, the final term will be something like n3+ n!O(1).
- By the definition of Big O, this can be reduced to O(n!) running time.

EXPERIMENT 11

Program

```
#include<iostream>

using namespace std;


int stepcount=0;

bool isSafe(int** arr, int x,int y,int n)

{

   for(int row =0; row<x;row++)

   {

      stepcount++;


      if(arr[row][y]==1)

      {

         stepcount++;

         return false;

      }

   }

         stepcount++;

         int row =x;

         stepcount++;
```

EXPERIMENT 11

```
    int col =y;
    stepcount++;
while(row>=0 && col>=0)
{
   stepcount++;
   if(arr[row][col]==1)
   {
      stepcount++;
      return false;
   }
            row--;
            stepcount++;
            col--;
            stepcount++;
    }
    stepcount++;
    row =x;
    stepcount++;
    col =y;
    stepcount++;
```

EXPERIMENT 11

```
    while(row>=0 && col<n)
    {
        stepcount++;
        if(arr[row][col]==1)
        {
            stepcount++;
            return false;
        }
        row--;
                    stepcount++;
        col++;
                    stepcount++;
    }
          return true;
}


bool nQueen(int** arr, int x, int n)
{
    stepcount++;
    if(x>=n)
```

EXPERIMENT 11

```
{
   stepcount++;
   return true;
}
for(int col =0;col<n;col++)
{
   stepcount++;
   if(isSafe(arr,x,col,n))
   {
      arr[x][col] =1;
                  stepcount++;
      stepcount++;
      if(nQueen(arr,x+1,n))
      {
         stepcount++;
         return true;
      }
      arr[x][col]=0;
                  stepcount++;
   }
```

EXPERIMENT 11

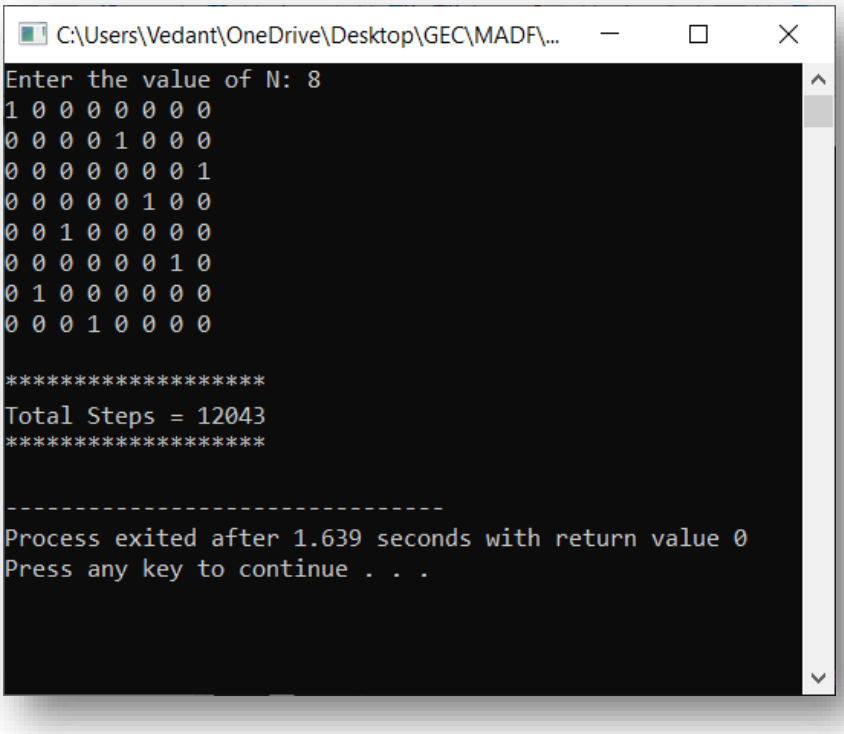
```
    }
    stepcount++;
    return false;
}

int main()
{
        int n;
        cout<<"Enter the value of N: ";
        cin>>n;
        int** arr = new int*[n];
        stepcount++;
        for(int i=0;i<n;i++)
    {
        stepcount++;
        arr[i]= new int [n]; for(int j=0;j<n;j++)
        {
                arr[i][j]=0;
                        stepcount++;
        }
```

EXPERIMENT 11

```
    }
        stepcount++;
    if(nQueen(arr,0,n))
    {
       for(int i=0;i<n;i++)
       {
          stepcount++;
          for(int j=0;j<n;j++)
          {
             stepcount++;
             cout<<arr[i][j]<<" ";
          }
                          cout<<endl;stepcount++;
       }
    }
    cout<<"\n*******************"<<endl;
          cout<<"Total Steps = "<<stepcount<<endl;
          cout<<"*******************"<<endl;
}
```

EXPERIMENT 11

Output



Conclusion

- Detailed concept of N-Queens problem (Using Backtracking)was studied successfully.
- Program using N-Queens Algorithm was executed successfully.
- The step count for the N-Queens Algorithm was obtained.