Name: Aditya Somani | TE (IT) – A | Roll No. T1851061 | PRN No. 71901204L

Assignment No. 5

AIM: Write a recursive program to find the solution of placing n queens on chessboard so that no two queens attack each other using Backtracking.

OBJECTIVE:

1. To understand the concept of recursive Backtracking.
2. To understand the concept of n queens problem.
3. To find the solution of placing n queens on chessboard.

THEORY:

Backtracking:

For many real-world problems, the solution process consists of working your way through a sequence of decision points in which each choice leads you further along some path. If you make the correct set of choices, you end up at the solution. On the other hand, if you reach a dead end or otherwise discover that you have made an incorrect choice somewhere along the way, you have to backtrack to a previous decision point and try a different path. Algorithms that use this approach are called backtracking algorithms.

If you think about a backtracking algorithm as the process of repeatedly exploring paths until you encounter the solution, the process appears to have an iterative character. As it happens, however, most problems of this form are easier to solve recursively. The fundamental recursive insight is simply this: a backtracking problem has a solution if and only if at least one of the smaller backtracking problems that results from making each possible initial choice has a solution.

Here is the algorithm (in pseudocode) for doing recursive backtracking from a given node n:

```
boolean solve(Node n) {
   if n is a leaf node {
      if the leaf is a goal node, return true
      else return false
   } else {
      for each child c of n {
         if solve(c) succeeds, return true
      }
      return false
   }
}
```

Notice that the algorithm is expressed as a boolean function. This is essential to understanding the algorithm. If solve(n) is true, that means node n is part of a solution--that is,

node n is one of the nodes on a path from the root to some goal node. We say that n is solvable. If solve(n) is false, then there is no path that includes n to any goal node.

Algorithm of n queens problem:

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.

1) Start in the leftmost column
2) If all queens are placed
   return true
3) Try all rows in the current column. Do following for every tried row.
   a) 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.
   b) If placing queen in [row, column] leads to a solution then return true.
   c) If placing queen doesn't lead to a solution then unmark this [row, column] (Backtrack) and go to step (a) to try other rows.
4) If all rows have been tried and nothing worked, return false to trigger backtracking.

Conclusion:

Thus we have implemented n queens problem using backtracking.

PROGRAM

```
#include<stdio.h>

#include<math.h>

int board[20],count;

int main()

{

 int n,i,j;

 void queen(int row,int n);
```

```
 printf(" - N Queens Problem Using Backtracking -");
 printf("\n\nEnter number of Queens:");
 scanf("%d",&n);
 queen(1,n);
 return 0;
}
 //function for printing the solution
void print(int n)
{
 int i,j;
 printf("\n\nSolution %d:\n\n",++count);
  for(i=1;i<=n;++i)
  printf("\t%d",i);
  for(i=1;i<=n;++i)
 {
  printf("\n\n%d",i);
  for(j=1;j<=n;++j) //for nxn board
  {
   if(board[i]==j)
    printf("\tQ"); //queen at i,j position
   else
    printf("\t-"); //empty slot
  }
 }
}
```

```
/*funtion to check conflicts If no conflict for desired postion returns 1 otherwise returns 0*/
int place(int row,int column)
{
 int i;
 for(i=1;i<=row-1;++i)
 {
  //checking column and digonal conflicts
  if(board[i]==column)
   return 0;
  else
   if(abs(board[i]-column)==abs(i-row))
    return 0;
 }
  return 1; //no conflicts
}
 //function to check for proper positioning of queen
void queen(int row,int n)
{
 int column;
 for(column=1;column<=n;++column)
 {
  if(place(row,column))
  {
   board[row]=column; //no conflicts so place queen
   if(row==n) //dead end
    print(n); //printing the board configuration
```

```
  else //try queen with next position
   queen(row+1,n);
 }
}
}
```

Output

Enter number of Queens:4

Solution 1:

	1	2	3	4
1	-	Q	-	-
2	-	-	-	Q
3	Q	-	-	-
4	-	-	Q	-

Solution 2:

	1	2	3	4
1	-	-	Q	-
2	Q	-	-	-
3	-	-	-	Q
4	-	Q	-	-