

NAME: Aditya Somani

PRN: 71901204L

ROLL: T1851061

=

SL-5(Group'B')

Assignment No. (B)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;
}
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)
        {
            if(board[i]==j)
                printf("\tQ");
            else
                printf("\t-");
        }
    }
}
int place(int row,int column)
{
    int i;
    for(i=1;i<=row-1;++i)
    {
        if(board[i]==column)
            return 0;
        else
            if(abs(board[i]-column)==abs(i-row))
                return 0;
    }
    return 1;
}
void queen(int row,int n)
{
    int column;
    for(column=1;column<=n;++column)
    {
        if(place(row,column))
        {
            board[row]=column;
            if(row==n)
                print(n);
            else
                queen(row+1,n);
        }
    }
}

```

OUTPUT:-

a

The screenshot shows a C++ IDE with a file named `main.cpp` and an `Output` window. The code implements a backtracking algorithm for the N-Queens problem. The `Output` window displays the program's execution for `n=5`, showing the input, the solution found, and a 5x5 board representation.

```
main.cpp
41     return 0;
42 }
43     return 1;
44 }
45 void queen(int row,int n)
46 {
47     int column;
48     for(column=1;column<=n;++column)
49     {
50         if(place(row,column))
51         {
52             board[row]=column;
53             if(row==n)
54                 print(n);
55             else
56                 queen(row+1,n);
57         }
58     }
59 }
60
```

Output

```
/tmp/NkCStA1PmC.o
- N Queens Problem Using Backtracking -

Enter number of Queens:5

Solution 1:

   1  2  3  4  5
1   Q  -  -  -  -
2   -  -  Q  -  -
3   -  -  -  -  Q
4   -  Q  -  -  -
5   -  -  -  Q  -
```

Activate Windows
Go to Settings to activate Windows.

The screenshot shows the same C++ IDE as above, but with the `Output` window displaying the results for `n=9`. The board representation is a 9x9 grid.

```
main.cpp
41     return 0;
42 }
43     return 1;
44 }
45 void queen(int row,int n)
46 {
47     int column;
48     for(column=1;column<=n;++column)
49     {
50         if(place(row,column))
51         {
52             board[row]=column;
53             if(row==n)
54                 print(n);
55             else
56                 queen(row+1,n);
57         }
58     }
59 }
60
```

Output

```
/tmp/NkCStA1PmC.o
- N Queens Problem Using Backtracking -

Enter number of Queens:9

Solution 1:

   1  2  3  4  5  6  7  8  9
1   Q  -  -  -  -  -  -  -  -
2   -  -  Q  -  -  -  -  -  -
3   -  -  -  -  -  Q  -  -  -
4   -  -  -  -  -  -  -  Q  -
5   -  Q  -  -  -  -  -  -  -
```

Activate Windows
Go to Settings to activate Windows.