DESIGN AND ANALYSIS OF ALGORITHMS

EXPERIMENT 7

Name: Janhavi R Deshmukh

Branch: CSE(DS) D1

UID: 2021700017

Aim:- Backtracking (N Queens problem)

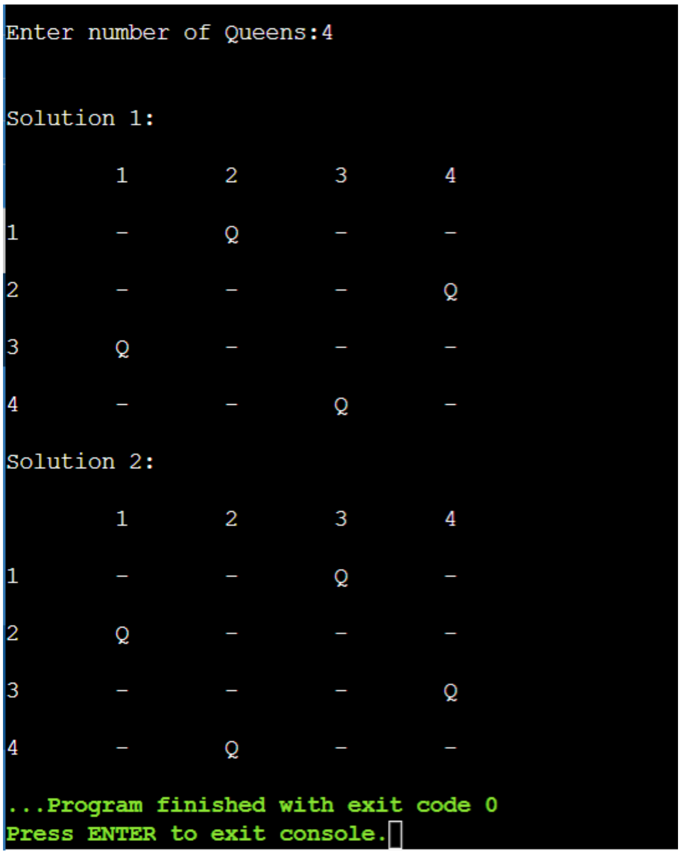
Algorithm:

a. Initialize an empty chessboard of size NxN.
b. Start with the leftmost column and place a queen in the first row of that column.
c. Move to the next column and place a queen in the first row of that column.
d. Repeat step 3 until either all N queens have been placed or it is impossible to place a queen in the current column without violating the rules of the problem.
e. If all N queens have been placed, print the solution.
f. If it is not possible to place a queen in the current column without violating the rules of the problem, backtrack to the previous column.
g. Remove the queen from the previous column and move it down one row.
h. Repeat steps 4-7 until all possible configurations have been tried.

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       -       -

...Program finished with exit code 0
Press ENTER to exit console.
```

Complexity Analysis :
Time complexity: O(N!):
The first queen has N placements, the second queen must not be in the same column as the first as well as at an oblique angle, so the second queen has N-1 possibilities, and so on, with a time complexity of O(N!).
Spatial Complexity: O(N): Need to use arrays to save information.

Conclusion:
In this experiment, we implemented N queens problem using backtracking