Experiment 9

Aim:Program to implement N-Queen Problem using Backtracking Method.

9.1CO Attained: CO3 and CO5

9.2Objective:

N - Queens problem is to place n - queens in such a manner on an n x n chessboard that no queens attack each other by being in the same row, column or diagonal.we have to place on the chessboard, such that no two queens attack each other.

9.3Resources: Turbo c/Dev C++

9.4Program Logic:

Backtracking is an algorithmic technique for solving problems recursively by trying to build a solution incrementally, one piece at a time, removing those solutions that fail to satisfy the constraints of the problem at any point of time (by time, here, is referred to the time elapsed till reaching any level of the search tree).

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 rowthen mark this [row, column] as part of the solution and recursively check if placingqueen here leads to a solution.
b) If placing the queen in [row, column] leads toa solution then return true.
c) If placing queen doesn't lead to a solution thenunmark this [row, column] (Backtrack) and go
step (a) to try other rows.
4) If all rows have been tried and nothing worked return false to trigger backtracking.

9.5Procedure:

1. Create: Open Dev C++/C and write a program after that save the program with the .c extension.
2. Compile: Alt + F9
3. Execute: Ctrl + F10

9.6Program Code:

```
#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);
  }}}
```

9.7Conclusion:

```
- N Queens Problem Using Backtracking -

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    -    -
```

9.8Analysis:

Time complexity for the N-Queen problem solved using Backtracking is O(N!) where N denotes number of Queens and dimensions of the chess board.

9.9Lab Viva Questions:

1. What is the most efficient approach to solve this problem?
2. Define Backtracking approach.
3. Define N – Queen problem with types.