# Artificial Intelligence (CS308)

## Assignment - 8

## U19CS012

Q1.) Implement **N Queens** Problem using below Algorithms in PROLOG.

- ✓ Breadth First Search
- ✓ Depth First Search

## Code

```
% Implement N-Queens using BFS and DFS.

% To make a empty board we first require to make rows and then columns

% Rows
row(0,[]).
row(N,[0|T]) :- N > 0,
N1 is N - 1, row(N1,T).

% Columns using the same logic as rows
col(0,_,[]).
col(N,H, [H|T1]) :- N > 0,
N1 is N - 1, col(N1,H,T1).

% Calling row and column to make an empty board.
empty(N,Board) :-
    row(N,Row),
    col(N,Row,Board).

% Utility function to print output in a matrix format
printBoard([]).
printBoard([X|Pt]) :- print(X),nl,printBoard(Pt).

% To get a cell value of coordinates (X,Y)
getXY(X,Y,[_|Mat], Z) :-
    Y > 0,
    Y1 is Y - 1,
    getXY(X,Y1,Mat,Z), ! .
getXY(X,0,[M|_],H) :-
    getX(X,M,H).

% To get a particular cell in a specific row
getX(X,[_|Mat],H) :-
    X > 0,
    X1 is X - 1,
```

```
    getX(X1,Mat,H), ! .
getX(0,[H|_],H).

% To change the a particular cell in a matrix
changeXY(X,Y,[M|Mat],Z) :-
    Y > 0,
    Y1 is Y - 1,
    changeXY(X,Y1,Mat,Z1),
    Z = [M|Z1].
changeXY(X,0,[M|Mat],Z) :-
    changeX(X,M,Z1),
    Z = [Z1|Mat].

% To change a particular cell {0,1} in a row
changeX(X,[H|T],Z):-
    X > 0,
    X1 is X - 1,
    changeX(X1,T,Z1),
    Z = [H|Z1].
changeX(0,[_|T],[1|T]).

% To check the queen in the UP direction.
checkUp(-1,_,_).
checkUp(X,Y,Board) :-
    X>=0,
    X1 is X-1,
    getXY(X,Y,Board,Val),
    Val is 0,
    checkUp(X1,Y,Board).

% To check the queen in the Left Diagonal direction.
checkLeftUpDiagonal(-1,_,_).
checkLeftUpDiagonal(_,-1,_).
checkLeftUpDiagonal(X,Y,Board) :-
    X>=0,
    Y>=0,
    X1 is X-1,
    Y1 is Y-1,
    getXY(X,Y,Board,Val),
    Val is 0,
    checkLeftUpDiagonal(X1,Y1,Board).

% To check the queen in the Right Diagonal direction.
checkRightUpDiagonal(_,N,N,_).
checkRightUpDiagonal(-1,_,_,_).
checkRightUpDiagonal(X,Y,N,Board) :-
    X>=0,
    X1 is X-1,
    Y<N,
    Y1 is Y+1,
```

```
    getXY(X,Y,Board,Val),
    Val is 0,
    checkRightUpDiagonal(X1,Y1,N,Board).

% Call check functions to check if the state is stable or not
validityCheck(I, N, J, Board, NewBoard) :-
    checkUp(I,J,Board),
    checkLeftUpDiagonal(I,J,Board),
    checkRightUpDiagonal(I,J,N,Board),
    changeXY(I,J,Board,NewBoard).
validityCheck(I,N,J,Board,Res) :-
        J>0,
        J1 is J-1,
        validityCheck(I,N,J1,Board,Res).

% Place the queen and check for validity
placeQueenAtNewPos(I, N, Board, NewBoard) :-
    validityCheck(I, N, N, Board, NewBoard).

% DFS implementation
dfs(CurrentBoard, N, N, FinalBoard):-
    FinalBoard = CurrentBoard.
dfs(CurrentBoard, I, N, FinalBoard):-
    I<N,
    I1 is I+1,
    placeQueenAtNewPos(I, N, CurrentBoard, NewBoard),
    dfs(NewBoard, I1, N, FinalBoard).

% BFS implementation
bfs([],_,[]).
bfs([[CurrentBoard, I]|Tail],N,FinalBoard) :-
    I is N,
    bfs(Tail,N,NewTail),
    FinalBoard=[CurrentBoard|NewTail].

bfs([[CurrentBoard, I]|Tail],N,FinalBoard) :-
    I < N,
    I1 is I+1,
    placeQueenAtNewPos(I, N, CurrentBoard, NewBoard),
    append(Tail,[[NewBoard, I1]],NBoard),
    bfs(NBoard,N,FinalBoard).

% Mian Driver function to call the above functions
nQueens(N) :-
    empty(N,Board),
    write('1. BFS'), nl,
    write('2. DFS'), nl,
    read(Choice),
    (
        Choice == 1 ->
```

```
        bfs([[Board,0]],N,[FinalBoard|_]),
        printBoard(FinalBoard)
    ;
    Choice == 2 ->
        dfs(Board,0,N,FinalBoard),
        printBoard(FinalBoard)
    ;
    % Else Invalid Input
    write('Invalid Choice Entered!')
).
```

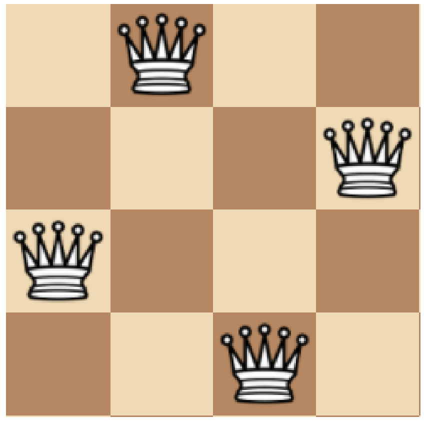
## Output

Trivial Test Cases - for **n = 1, 2, 3**

```
9 ?- nQueens(1).
1. BFS
2. DFS
|: 2.
[1]
true .

10 ?- nQueens(2).
1. BFS
2. DFS
|: 2.

false.

11 ?- nQueens(3).
1. BFS
2. DFS
|: 2.

false.
```

## n = 4

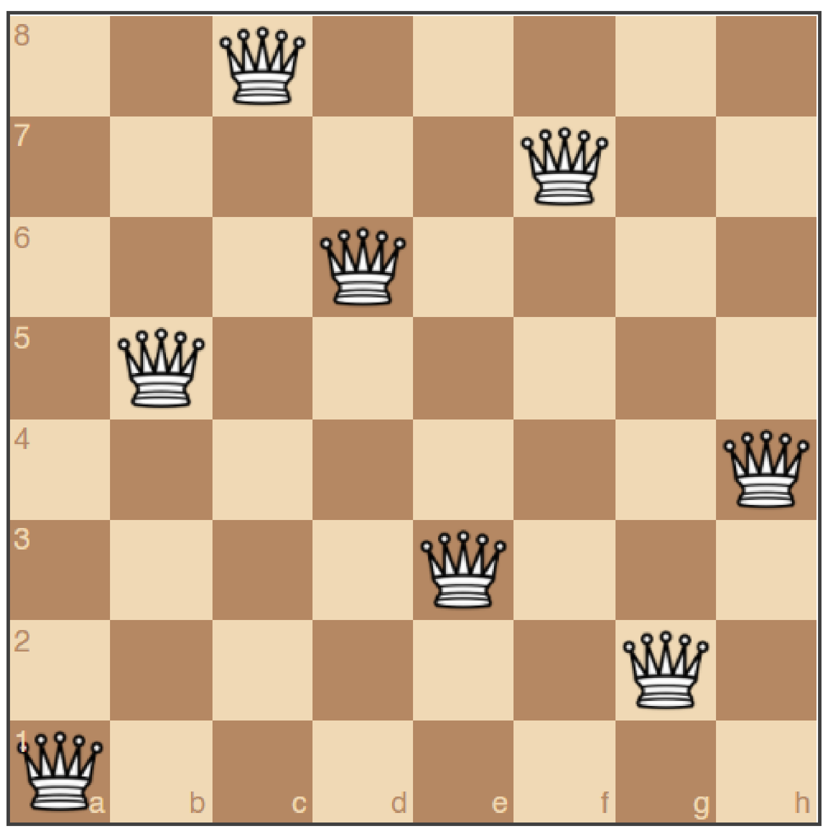
```
5 ?- nQueens(4).
1. BFS
2. DFS
|: 1.
[0,1,0,0]
[0,0,0,1]
[1,0,0,0]
[0,0,1,0]
true .
```



```
6 ?- nQueens(4).
1. BFS
2. DFS
|: 2.
[0,1,0,0]
[0,0,0,1]
[1,0,0,0]
[0,0,1,0]
true .
```

## n = 8

```
2 ?- consult('nqueens.pl').
true.

2 ?- nQueens(8).
1. BFS
2. DFS
|: 1.
[0,0,1,0,0,0,0,0]
[0,0,0,0,0,1,0,0]
[0,0,0,1,0,0,0,0]
[0,1,0,0,0,0,0,0]
[0,0,0,0,0,0,0,1]
[0,0,0,0,1,0,0,0]
[0,0,0,0,0,0,1,0]
[1,0,0,0,0,0,0,0]
true .

3 ?- nQueens(8).
1. BFS
2. DFS
|: 2.
[0,0,1,0,0,0,0,0]
[0,0,0,0,0,1,0,0]
[0,0,0,1,0,0,0,0]
[0,1,0,0,0,0,0,0]
[0,0,0,0,0,0,0,1]
[0,0,0,0,1,0,0,0]
[0,0,0,0,0,0,1,0]
[1,0,0,0,0,0,0,0]
true .
```



## n = 10, 12 & 16

```
2 ?- nQueens(10).
1. BFS
2. DFS
|: 2.
[0,0,0,0,1,0,0,0,0,0]
[0,0,0,0,0,0,1,0,0,0]
[0,0,0,1,0,0,0,0,0,0]
[0,0,0,0,0,0,0,0,0,1]
[0,0,1,0,0,0,0,0,0,0]
[0,0,0,0,0,1,0,0,0,0]
[0,0,0,0,0,0,0,0,1,0]
[0,1,0,0,0,0,0,0,0,0]
[0,0,0,0,0,0,0,1,0,0]
[1,0,0,0,0,0,0,0,0,0]
true .

3 ?- nQueens(12).
1. BFS
2. DFS
|: 2.
[0,0,0,0,0,1,0,0,0,0,0,0]
[0,0,0,0,0,0,0,1,0,0,0,0]
[0,0,0,0,1,0,0,0,0,0,0,0]
[0,0,0,0,0,0,0,0,0,0,1,0]
[0,0,0,1,0,0,0,0,0,0,0,0]
[0,0,0,0,0,0,0,0,0,1,0,0]
[0,0,0,0,0,0,1,0,0,0,0,0]
[0,0,1,0,0,0,0,0,0,0,0,0]
[0,0,0,0,0,0,0,0,0,0,0,1]
[0,1,0,0,0,0,0,0,0,0,0,0]
[0,0,0,0,0,0,0,0,1,0,0,0]
[1,0,0,0,0,0,0,0,0,0,0,0]
true .
```

```
5 ?- nQueens(16).
1. BFS
2. DFS
|: 2.
[0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0]
[0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0]
[0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0]
[0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0]
[0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0]
[0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0]
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1]
[0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0]
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0]
[0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0]
[0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0]
[0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0]
[0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0]
[0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
[0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0]
[1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
true .
```

The n-queens problem is solvable for **n=1** and **n≥4**.

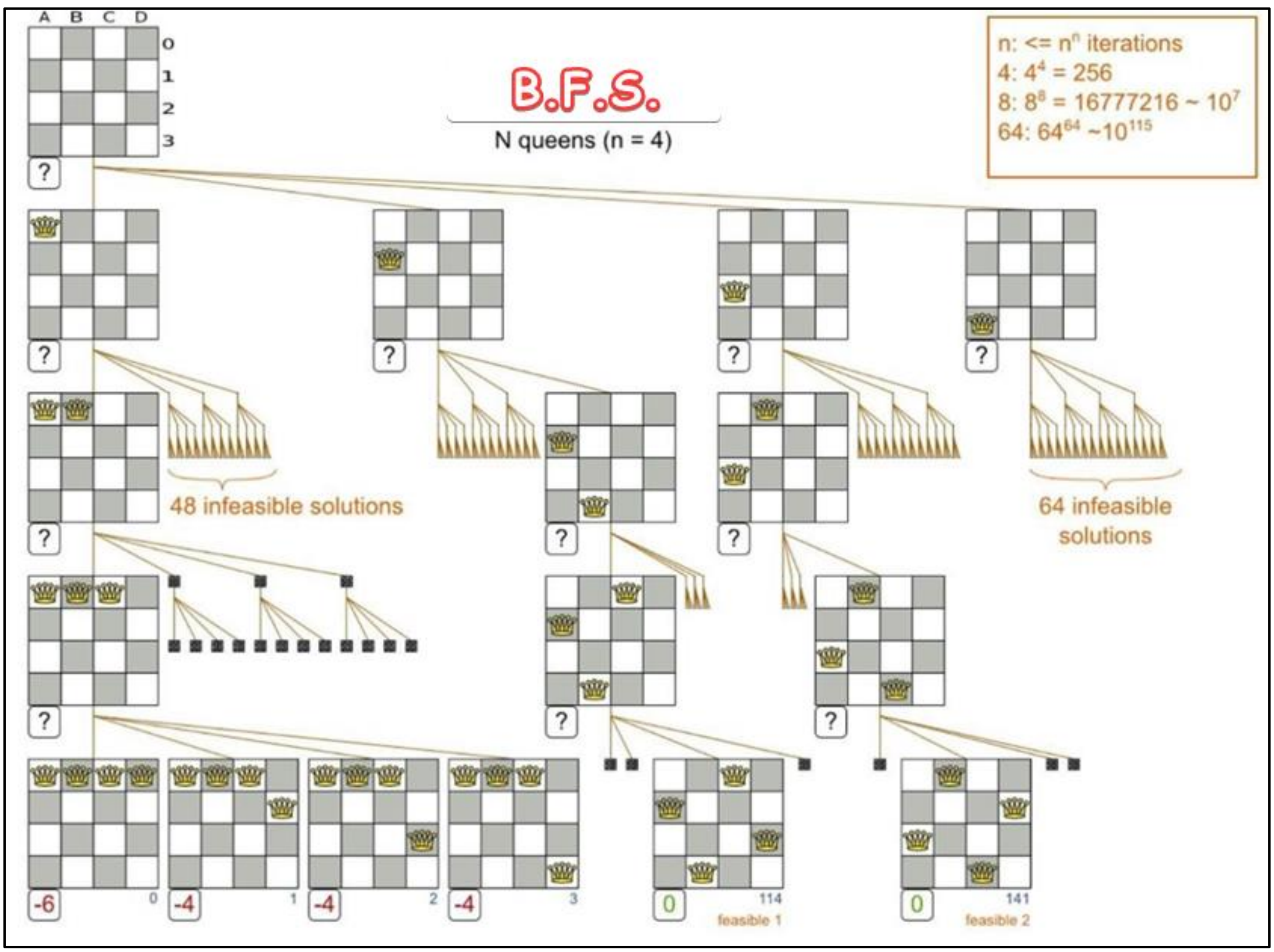
Q2.) Compare the **Complexity** of Both Algorithms.

Which algorithm is **best** suited for implementing N Queen's problem and why?

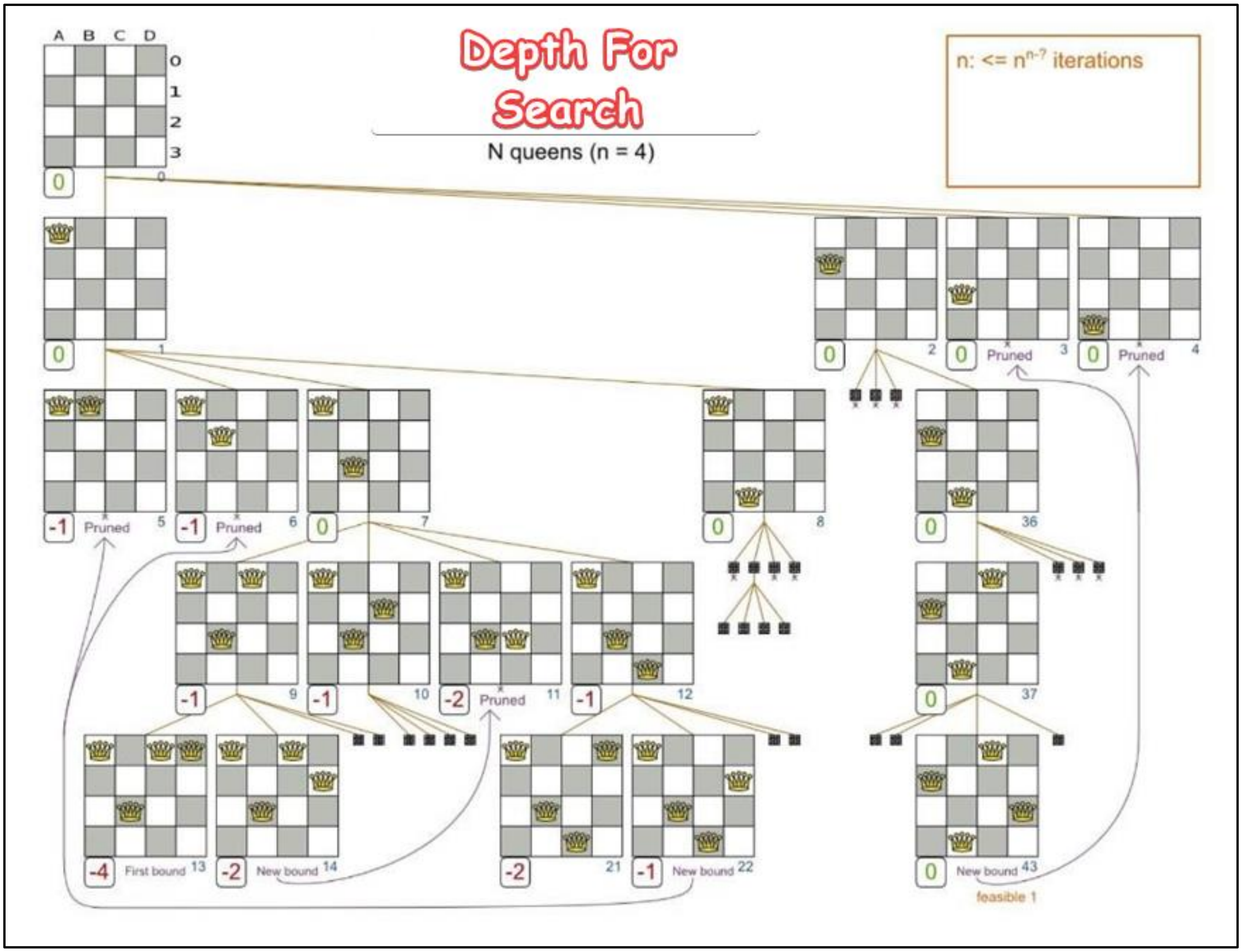
1. Breadth First Search
2. Depth First Search

| Algorithm | Time Complexity | Remarks |
|---|---|---|
| **BFS** | O(n^n) | It tries **every possible** solution |
| **DFS** | O(n!) | It **discards** the **Invalid solutions** and *their following Recursive calls* as and when they are found. |

BFS **O(n^n)**



DFS **O(n!)**




Therefore, **D.F.S.** is <u>best Suited</u> for **N-Queens** problem due to Lesser Time Complexity **(O(N!))** & <u>Reduces the Sample Space</u> at Every Step in Algorithm.

**SUBMITTED BY**: U19CS012

BHAGYA VINOD RANA