# Manish Singh

# 2017HT12447

# AI Assignment

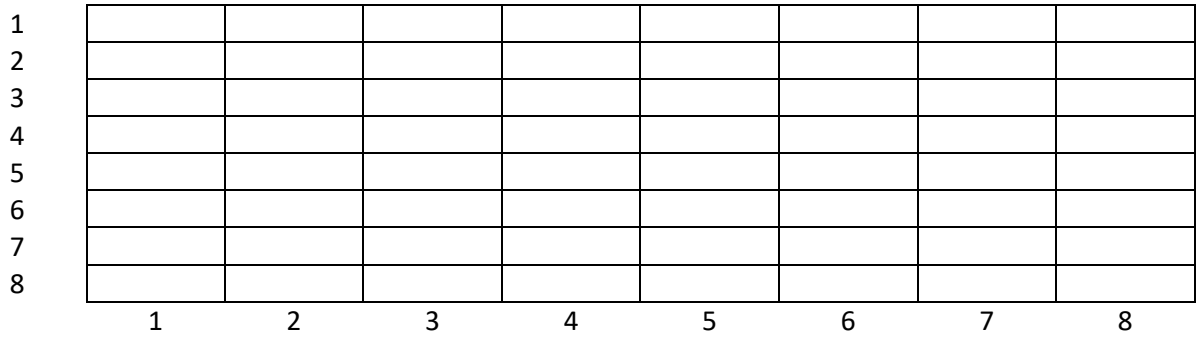
# N-Queen problem in prolog

# N-queen problem

## Approach to solve the problem

To solve the problem, we will follow the following approach.

We will first construct a board which has 1 to N rows and 1 to N columns




Now, we will try to perform permutation on Column and see on which Row, Column position we can place the queens so that it satisfies the problem.

As we are taking permutations, we don't need to consider the problem of same rows and columns as there will never be a conflict in this sense.

Now to check if no 2 queens are on the same diagonals, we will use the following check.

1. For right diagonal (/), if two queens are on the same diagonal, then sum of Row and Column for each queen would be same.

2. For the left diagonal (\), if two queens are on the same diagonal, then the difference of Row and Column for each queen would be same.

If we get a permutation, where diagonals rule check is true i.e for right diagonal, no two queens have the same sum of Row and Column & for left diagonal, the difference of row and column of two queens are not same, then we can consider that permutation of columns as a possible solution.

## Code

```
initBoard(N,Q):-N1 is N+1,init(N1,1,Q).

init(N,N,[]).

init(N,N2,[Q1|Q]) :-Q1 is N2,N2=<N,N1 is N2+1,init(N,N1,Q).
```

This section initializes the board. We make rows from 1 to N.

```
del(X,[X|LIST1],LIST1).

del(X,[Y|LIST1],[Y|LIST2]):-del(X,LIST1,LIST2).


l_perm([],[]).

l_perm(L,[X|P]):-del(X,L,L1),l_perm(L1,P).
```

This section of the code calculates the possible column permutations

```
l_mem(A,[A|_]).

l_mem(A,[_|T]):-l_mem(A,T).
```

This section checks if a given element is a member of the list. This will be used to check the sum & diff of all rows and columns to check if two or more queens are on the same diagonal. Is true if two or more queens are on the same diagonal.

```
calcDiag([],[],[],[]).
calcDiag([Row|Rows],[Col|Cols],[RD|Rds],[LD|Lds]) :-
    RD is Row + Col,
    LD is Row - Col,
    calcDiag(Rows,Cols,Rds,Lds).
```

This section calculates the left diagonal (LD) and right diagonal (RD)

```
checkDiag([_]).
checkDiag([D1|D]) :-  \+l_mem(D1,D), checkDiag(D).
```

Checks the diagonal to see if two or more queens are on the same diagonal. We want all the sums unique so we put the not provable sign.

```
nqueens(N,Col) :-
        initBoard(N,Row),
    l_perm(Row,Col),
    calcDiag(Row,Col,Rds,Lds),
    checkDiag(Rds),
    checkDiag(Lds).
```

This is the execution part where we want all conditions to be true.

## Line by line explanation of code

```
initBoard(N,Q):-N1 is N+1,init(N1,1,Q).                    /*Initialize rows with initial params*/
init(N,N,[]).                                              /*End condition for recursion*/
init(N,N2,[Q1|Q]) :-Q1 is N2,N2=<N,N1 is N2+1,init(N,N1,Q).   /*Write list with 1 to N*/

del(X,[X|LIST1],LIST1).                                    /*if the head matches the variable then only return the tail*/
del(X,[Y|LIST1],[Y|LIST2]):-del(X,LIST1,LIST2).            /*If X does not match head then pass the TAIL and
append Y on tail2. The purpose is to delete X from the tail1.*/

l_perm([],[]).                                             /*End condition for recursion*/
l_perm(L,[X|P]):-del(X,L,L1),l_perm(L1,P).                 /*Calculate the permutation*/

l_mem(A,[A|_]).                                            /*If variable A matches head */
l_mem(A,[_|T]):-l_mem(A,T).                                /*Pass the tail*/


calcDiag([],[],[],[]).                                     /*End condition for recursion*/
calcDiag([Row|Rows],[Col|Cols],[RD|Rds],[LD|Lds]) :-       /*Calculate the diagonals*/
    RD is Row + Col,                                       /*Right diagonal Row+Col*/
    LD is Row - Col,                                       /* Left diagonal Row-Col*/
    calcDiag(Rows,Cols,Rds,Lds).                           /*Call recursively*/


checkDiag([_]).                                            /*End condition for recursion*/
```

```
checkDiag([D1|D]) :- \+l_mem(D1,D), checkDiag(D).      /*Now 2 same sum/diff should not be part of the list*/


nqueens(N,Col) :-
        initBoard(N,Row),                    /*Initialize rows with initial params*/
    l_perm(Row,Col),                         /*Col is permutation*/
    calcDiag(Row,Col,Rds,Lds),               /*Calculate diagonals*/
    checkDiag(Rds),                          /*Check right diagonal*/
    checkDiag(Lds).                          /*Check left diagonal*/
```

Source code is in the zip file.

## Executing code

nqueens(N,Q). /*For NxN board. N is a positive integer and Q is a variable*/

Eg:

nqueens(4, Q). /*For 4x4 board*/

nqueens(8, Q). /*For 8x8 board*/

**Note**: This program was tested in GNU Prolog 1.4.5

# Output

Output comes in the form of :

Variable=[Col, Col, ….,Col]

Where 1st result is placing of queen in column number against Row1, 2nd result is placing of queen in column number against Row2,…,Nth result is placing of queen in column number against RowN