Assignment 4

Name: Krunal Rank
Roll No: U18CO081

Implement N queens problem using below algorithms in prolog.
Compare the complexity of both algorithms.
Which algorithm is best suited for implementing N queens problem and why ?
1. Breadth First Search
2. Depth First Search

```
n_queens(N, Qs) :-
  length(Qs, N),
  Qs ins 1..N,
  safe_queens(Qs).

safe_queens([]).
safe_queens([Q|Qs]) :-
  safe_queens(Qs, Q, 1),
  safe_queens(Qs).

safe_queens([], _, _).
safe_queens([Q|Qs], Q0, D0) :-
  Q0 #\= Q,
  abs(Q0 - Q) #\= D0,
  D1 #= D0 + 1,
  safe_queens(Qs, Q0, D1).

n_queens_dfs(N,Qs):-
  statistics(walltime, [TimeSinceStart | [TimeSinceLastCall]]),
  length(Qs,N),
  maplist(between(1,N),Qs),
  n_queens(N,Qs),
  statistics(walltime, [NewTimeSinceStart | [ExecutionTime]]),
  write('Execution took '), write(ExecutionTime), write(' ms.'), nl.

n_queens_bfs(N,Qs):-
  statistics(walltime, [TimeSinceStart | [TimeSinceLastCall]]),
  n_queens(N,Qs),
  maplist(between(1,N),Qs),
  statistics(walltime, [NewTimeSinceStart | [ExecutionTime]]),
  write('Execution took '), write(ExecutionTime), write(' ms.'), nl.
```

Output:

DFS:-

```
?- set_prolog_flag(answer_write_options,[max_depth(0)]).
true.

?- use_module(library(clpfd)).
true.

?- [n_queens].
Warning: /mnt/0FB812900FB81290/BTech/Assignments/3rd_Year/AIML/Assignment_4/n_queens.pl:20:
        Singleton variables: [TimeSinceStart,TimeSinceLastCall,NewTimeSinceStart]
Warning: /mnt/0FB812900FB81290/BTech/Assignments/3rd_Year/AIML/Assignment_4/n_queens.pl:29:
        Singleton variables: [TimeSinceStart,TimeSinceLastCall,NewTimeSinceStart]
true.

?- n_queens_dfs(1,Qs).
Execution took 0 ms.
Qs = [1].

?- n_queens_dfs(2,Qs).
false.

?- n_queens_dfs(3,Qs).
false.

?- n_queens_dfs(4,Qs).
Execution took 2 ms.
Qs = [2,4,1,3] .

?- n_queens_dfs(5,Qs).
Execution took 14 ms.
Qs = [1,3,5,2,4] .
```

```
?- n_queens_dfs(6,Qs).
Execution took 410 ms.
Qs = [2,4,6,1,3,5]
```

BFS:-

```
?- n_queens_bfs(1,Qs).
Execution took 0 ms.
Qs = [1].

?- n_queens_bfs(2,Qs).
false.

?- n_queens_bfs(3,Qs).
false.

?- n_queens_bfs(4,Qs).
Execution took 0 ms.
Qs = [2,4,1,3] .

?- n_queens_bfs(5,Qs).
Execution took 0 ms.
Qs = [1,3,5,2,4] .

?- n_queens_bfs(6,Qs).
Execution took 1 ms.
Qs = [2,4,6,1,3,5] .

?- n_queens_bfs(7,Qs).
Execution took 1 ms.
Qs = [1,3,5,7,2,4,6] .

?- n_queens_bfs(8,Qs).
Execution took 4 ms.
Qs = [1,5,8,6,3,7,2,4] .
```

Analysis:

I have used CLFPD constraints library functions in the prolog program along with statistics in built function to record time.

As seen from the above screenshots, we find that BFS is a lot faster than DFS. Here, DFS has been implemented using the Generate and Test Approach. BFS has been implemented using the Early Pruning Approach.

As seen from the execution times, we find that BFS is very fast as it can process the goal state for N=8 in 4ms whereas for DFS, it requires 410 ms for N=6 goal state.