B551 Assignment 0: N-queens and Python

By Harika Putti

Q1. N-Rooks problem.

Aim: To place 'N' no of rooks on an N*N board such that no 2 rooks are in the same row or column. The code uses DFS to check and place rooks at particular positions.

Abstraction: Using the abstraction model of prof. Crandall, we find the following –

1. *Set of states S*

   The state space for my nrooks implementation code is:

   This state space includes all the different ways in which 'n' rooks can be placed on the board, excluding all the possibilities where the total number of rooks on the board is greater than 'n'.

   We're placing the rooks only where it's intended and not searching for the goal state within every possible state, this way the number of states in the state space is lesser.

2. *Initial state s0*

   The initial state is [0*N] N.

   That is, for N = 3

   $$s0 = \begin{matrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{matrix}$$

3. *A function SUCC: S → 2^S that encodes possible transitions of the system*

   So, the successor state is one of the possible ways of placing the rook in its correct position. Only the successor states that have less than N rooks and the goal state are included.

4. *Set of goal states*

   All possible ways of placing 'n' rooks such that no 2 rooks are on the same row or column.

   Example for N = 3 ⇒

   $$s0 = \begin{matrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{matrix}$$

5. *A cost function that calculates how "expensive" a given set of moves is*

   The cost function is irrelevant in this model.

Q2. The following are the changes I've made to the nrooks-2.py

Defining a new function and modifying another:

```
def successors2(board):
   successor_list = []
   if count_pieces(board) < N:
       for r in range (0,N):
              if count_on_row(board,r) == 0:
                     for c in range (0,N):
                            if count_on_row(board,r) == 0 and \
                          count_on_col(board,r) == 0 :
                                           successor_list.append(add_piece(board,r,c))
   return successor_list

def is_goal(board):
   return count_pieces(board) == N
```

What this function does is, it checks if a rook can be placed at a position before placing it, to reduce the number of times the solve function calls the is_goal function or he successor function. What this does, is include only the successors states that do not have more than or equal to N rooks. We should make sure each row or column on the board does not have a rook already. So, we get the list of successors that only contain cases where the number of rooks on the current board are less than that total rooks on the goal state i.e, N.

I have also modified the is_goal function to exclude the steps to check the row and column of the board which determines if the rooks are at a position.

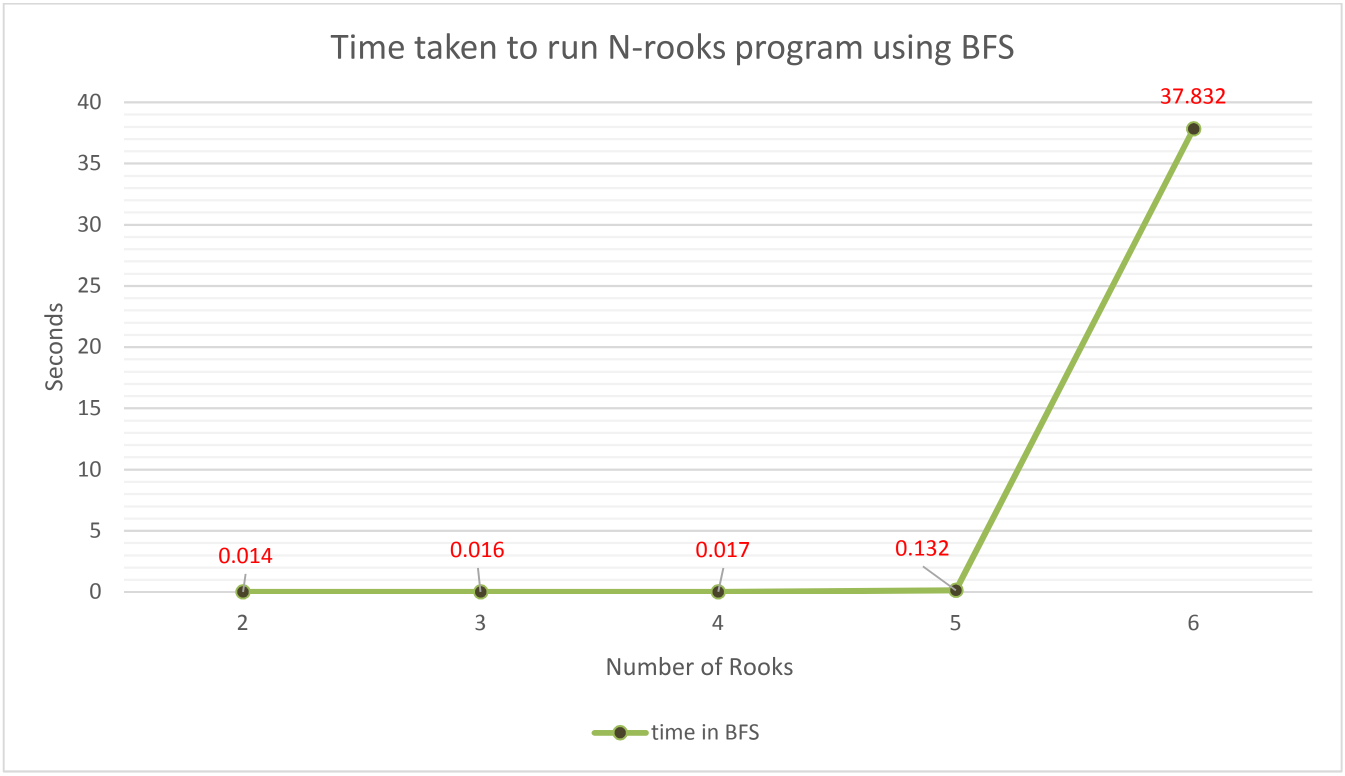
Q3. Changing code to do BFS instead of DFS

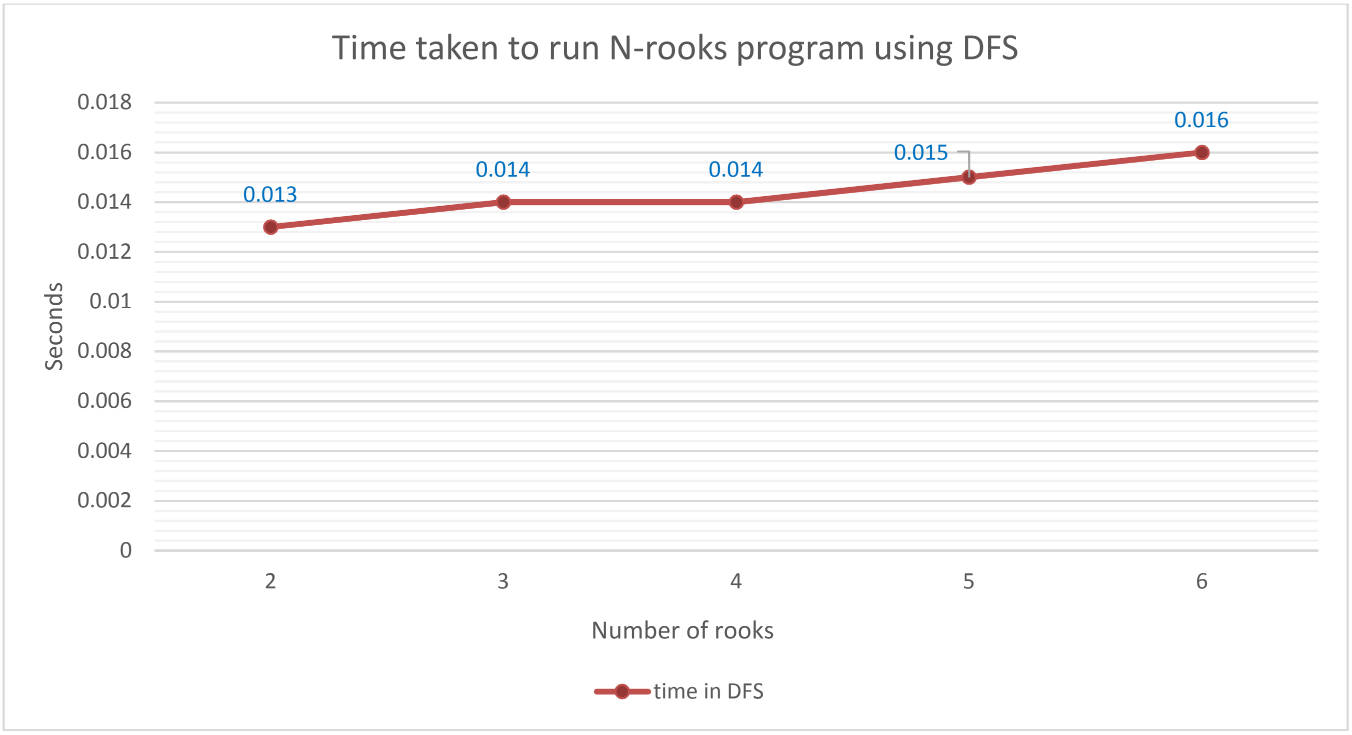
To make my code do a BFS instead of DFS all I did was change the way the fringe is accessed.

Modifications I made to my code –

```
def solve(initial_board):
   fringe = [initial_board]
   while len(fringe) > 0:
      for s in successors2( fringe.pop(0) ):
# I've modeled it to always access the first element of the fringe variable.
         if is_goal(s):
            return(s)
         fringe.append(s)
   return False
```

Q4. The following are the time graphs of my codes performing BFS and DFS.





As depicted by the graphs, DFS is much faster than BFS for the code.

This is because in DFS we're following the path of one valid successor state to another valid successor state and reaching the goal state. While, in BFS we're checking all the valid successors of board1 and then checking all the valid successors of board 2 and so on. This makes the code traverse through all successor states which we already know are not the goal.

So as the value of N increases the depth and the breadth increase. In DFS we keep adding the rooks in each row and move on to the next one and so on. But in BFS after adding the rook, it checks the positions of other successor states of the same row.

Therefore, as the N increase the time taken to execute also increases.