Assignment a0

Sanket Patole

Q.1 Abstraction

- Initial State(S0): An empty board of size NxN.
- Valid States: Set of boards with rooks placed at any position.
- Successor Function(S1..Sn): Children nodes(no. of boards) of the current board.
- Cost Function: Cost for each move(placement of rook) is constant fore every placement of rook.
- Goal State: Any state with N number of rooks placed on the board with only one rook in each row and column.

Q.2

All the successors of initial stored are being stored in the fringe and according to the initial program, fringe pops the board in a queue manner making it consume more time. To solve this, we pop the most recently added board using pop(0), hence switching to BFS and reducing the time to obtain the goal state.

Q.3

In this case we fixed the problem where the function was generating states which had N+1 rooks or states where there are no other rooks at all. Now the problem of choosing between BFS and DFS doesn't really matter as we are eliminating the unwanted states which was increasing the time of our run.

```
def successors2(board):
    tempList = [ add_piece(board, r, c) for r in range(0, N) for c in range(0,N)]
    mainList=[]
    #collects a list of boards which don't have N+1 rooks and boards which don't have any rooks at all
    for i in tempList:
        if i !=board and count_pieces(i)<(N+1) and \
           all( [ count_on_col(i, c) <= 1 for c in range(0, N) ] ) and\
           all( [ count_on_row(i, r) <= 1 for r in range(0, N) ] ) :
            mainList.append(i)
    return mainList
```

Q.4

```
#checks for object in the leftmost column
def find_left_col(board):
     for c in range(0,N):
         if(count_on_col(board,c) == 0):
             return c
```

The above function checks for the left most empty column and helps in placing the rook at the leftmost column rather than placing it anywhere.

This check is done in the successors3 function posted below.

Implementation of successor3():

```
def successors3(board):
    leftmost_col = find_left_col(board)
    #adds piece to leftmost column and not in "unavailable" positions
    tempList = [ add_piece(board, r, leftmost_col) for r in range(0, N) if[r,leftmost_col] not in cancel ]

    mainList=[]
    for i in tempList:
        #checks for no of pieces <= N and no duplicate boards are present in the list
        if i !=board and count_pieces(i)<(N+1) and \
           all( [ count_on_col(i, c) <= 1 for c in range(0, N) ] ) and \
           all( [ count_on_row(i, r) <= 1 for r in range(0, N) ] ) :
            mainList.append(i)
    #Check condition for queens
    if Q=="nqueen":
        diagonalList=[]
        for i in mainList:
            if diagonalCheck(i):
                diagonalList.append(i)
        return diagonalList
    else:
        return mainList
```

Graph of Time vs N: