B551 Assignment 0: N-queens and Python Fall 2018

Name: Virendra Wali`
IU ID: 2000423399

1) Write down the precise abstraction that the program is using and include it in your report. In other words, what is the set of valid states, the successor function, the cost function, the goal state definition, and the initial state?

Initial State: It is the starting state of board from which we start searching the solution(goal state), in case of N-rooks, initial state is the empty board with no rook placed on N*N board.

Goal State: Goal state of N-rook problem will be a board of N rooks placed in such way that no two rook should attack one another. There will be multiple goal states for N-rook problem. In given solution, we are checking if all rooks are present on board such that each row and column should have one rook and board contain N rooks in it.

Successor function: The function which creates all possible state boards for given state board.

State Space: It is the pool of all possible state boards for N-rook.

Cost Function: Here, we are not calculating any cost of state. We are just showing first goal state from multiple possible goal state found by BFS or DFS.

2) How to modify the code to switch to BFS ?

In nrook-2.py, we are using DFS by appending in the list and then popping state board from same end. Instead of adding state board from end of list, insert state board from start and pop it from end in this way we will be switching data structure from stack to queue, means we are using BFS by implementing queue data structure. After code modification i.e. after switching to BFS from DFS, I am able to reach solution state much quicker than DFS. As in BFS we search the state space level by level so eventually reach to goal state. On the other hand DFS is going in the deadlock as it is popping same state to find next available states.

States with DFS for 3-Rooks problem:

```
Value of s [[1, 0, 0], [0, 0, 0], [0, 0, 0]]

Value of s [[0, 1, 0], [0, 0, 0], [0, 0, 0]]

Value of s [[0, 0, 1], [0, 0, 0], [0, 0, 0]]

Value of s [[0, 0, 0], [1, 0, 0], [0, 0, 0]]

Value of s [[0, 0, 0], [0, 1, 0], [0, 0, 0]]

Value of s [[0, 0, 0], [0, 0, 1], [0, 0, 0]]

Value of s [[0, 0, 0], [0, 0, 0], [1, 0, 0]]

Value of s [[0, 0, 0], [0, 0, 0], [0, 1, 0]]

Value of s [[0, 0, 0], [0, 0, 0], [0, 0, 1]]
Value of s [[1, 0, 0], [0, 0, 0], [0, 0, 1]]

Value of s [[0, 1, 0], [0, 0, 0], [0, 0, 1]]

Value of s [[0, 0, 1], [0, 0, 0], [0, 0, 1]]

Value of s [[0, 0, 0], [1, 0, 0], [0, 0, 1]]

Value of s [[0, 0, 0], [0, 1, 0], [0, 0, 1]]

Value of s [[0, 0, 0], [0, 0, 1], [0, 0, 1]]

Value of s [[0, 0, 0], [0, 0, 0], [1, 0, 1]]

Value of s [[0, 0, 0], [0, 0, 0], [0, 1, 1]]

Value of s [[0, 0, 0], [0, 0, 0], [0, 0, 1]]
```

So each time successor function getting [[0, 0, 0], [0, 0, 0], [0, 0, 1]] board to explore next possible boards in 3-rook problem.

3) Explain successor2 function and explain if` the choice of BFS or DFS matter? Why or why not?

In new modified successor function(successors2), I am checking for given row and column combination. If rook is already present, I am avoiding to create same state for that row and column and simultaneously I am not placing rook in column where a rook is already present. This will make sure that we will not be creating board for N+1 rooks and avoiding deadlock by not placing rook on same position. After applying those changes, DFS is comparatively more faster than BFS. Observation table as follows:

DFS Observations	**BFS Observations**
N = 3 total time 0.00200009346008	N = 3 total time 0.00600004196167
N = 4 total time 0.0260000228882	N = 4 total time 0.206000089645
N = 5 total time 0.828000068665	N = 5 total time 16.381000042

4) Explain successor3.

In successors3, I am placing a rook in first left most position in such a way that no 2 rooks can attack each other and no two rooks will present in same row or same column. By this code change, I am able to find solution up to N = 1520 in 1 minute. As we are deciding single place for each rook on board . So for successors3, it is not creating multiple states and placing rook in next possible place in same board. Therefore, even if the value on N is bigger it is taking linear time to place N rooks in board.

For N = 1520 total time 59.6339998245