Assignment 0: Implementing 8-queens

Submitted By: Tejas Deoras

Question 1:

Spend some time familiarizing yourself with the code. 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?

Answer 1:

Set of States : Arrangement of all possible 0 to N rooks on a NxN board

= $\sum_{i=0}^{i=N} P_i^{N*N}$

Initial State : NxN board with no rooks

Successor Function: Adding Rook to any one square

= x € S with r rooks then succ(x) = P_{r+1}^{N*N}

Set of Goal States : N rooks on a NxN board such that no rooks attack each other

Cost function : Let us consider 1 unit for adding one rook and removing one rook i.e each branch from one node to another will have cost 1.

Question 2:

You've probably noticed that the code given is implemented by depth first search (DFS). Modify the code to switch to BFS instead. What happens when you run the code for different values of N now, and why?

Answer 2:

The DFS can only performs upto N = 2 after which it takes lot of time. When the algorithm is converted to BFS simply by adding the new successor nodes to start of fringe rather that at the end the N value extends all the way upto N = 5.
DFS uses a Stack - fringe.append(s)
BFS uses a Queue - fringe.insert(0,s)

The reason for this is a DFS may pick up a node with wrong positioning of pieces and may go all the way exploring that wrong choice until a certain depth(unless specified) and never arrive at the answer.
A BFS on the other hand explores all the nodes at the particular level and would always be successful in expanding the node which would eventually make it achieve the goal state.

Question 3:

Recall from class that there are usually many ways to define a state space and successor function, and some are more practical than others (e.g. some have much larger search spaces). The successor function in the code is defined in a very simplistic way, including generating states that have N+1 rooks on them, and allowing “moves” that involve not adding a rook at all. Create a new successors() function called successors2() that fixes these two problems. Now does the choice of BFS or DFS matter? Why or why not?

Answer 3:

Adding a limitation to the N value i.e going only after a certain depth and avoiding the possibility on not adding a piece in the successor sates leads to DFS performing faster that BFS as N value increases.
This is because adding the conditions causes DPS to not spend time on a wrong choice and revert back after a certain depth. As N increases the no of nodes needed for BFS to expand increases and this causes DFS to outperform BFS as N increases.

Question 4:

Even with the modifications so far, N=8 is still very slow. Instead of allowing the successor function to place a piece anywhere on the board, let’s define a successor that is much more orderly: it’s only allowed to add a piece to the

leftmost column of the board that is currently empty. (For example, if the board is empty, the rook has to be placed in the first column; if there are 4 rooks on the board, the next one has to go in the fifth column, etc.) Modify the code to implement this alternative abstraction with a successors function called successors3(). Feel free to make other code improvements as well. What is the largest value of N your new version can run on within about 1 minute?

Answer 4:
The maximum value for which N runs with the provided modification is 115.

Question 5:

N-rooks and N-queens problems

Answer 5:

In code