Elements of Artificial Intelligence

Documentation of Assignment 0

Prahasan Gadugu.
UID: 2000435702
prgadugu@iu.edu

1) Steps of Abstraction:

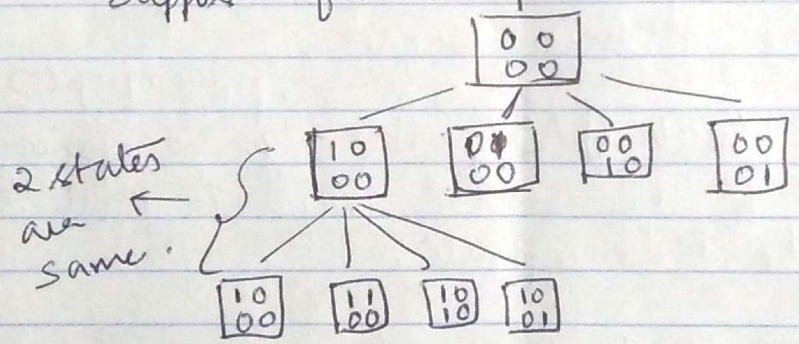
(i) Set of states in a N-rook is given by
$N^2 \times (N^2-1) \times (N^2-2)$ ..... N times.

$(N^2)! / (N^2-N)!$ are the no. of states.

A state space is the placement of N rooks on the board.

(ii) The set of valid states are those in which no rook clashes with each other.

(iii) The successor function given in the starter code, is not an optimised one since, Suppose for N=4



Here the successor function gives the next possible states, given a state by simply placing a rook in each position of the board.

(iv) Cost function is defined as the one that specifies how much does a move cost you. In this case we have not defined a cost function since it is not relevant.

(v) Goal state is the one that you need to reach starting from the initial state.
Here in this program our goal state is placing of 'N' rooks on the NxN board such that no rook clashes with each other.

(vi) The initial state is the board without any rooks, the one which you start with, (∵ initial board = [0*N]*N).

(2) Initially the starter~~d~~ code is written in DFS, when we execute the program, the code runs only for until, N = 2. for N=3, it goes on searching for the goal states.

DFS works in this manner,

fringe.pop()

Stack () ← → Has only one opening.

~~As~~ it, Even if the state graph is finite the search tree is infinite.

Now to implement the BFS we need to. make the fringe data structure a queue, By popping out the first ~~element~~ state that is added,

which can be done using,

fringe.pop(0). in the solve() function.

→ () queue. () →

After doing this,

we get upto N=5, goes slow for N=6.

By doing this, BFS is good if the goal state needs to be checked at the same level of tree

So for values of N (small) Bfs is fast until N=5

So to change from DFS to BFS we write the code in the ~~successor~~ solve function,
Successor(fringe.pop(0)) thus implementing a queue.

3) Explanation for successor_2 function:

In the successor function, my code checks ~~whether~~, N+1 rooks case,
by implementing,
if (count_pieces(board) <N)

and also eliminates the states that, in which a rook is placed when already there is a rook on it. by,
if(board[r][c]! =1).

Here the BFS is giving me upto N=5 and DFS is giving N=5.

As per the running times.

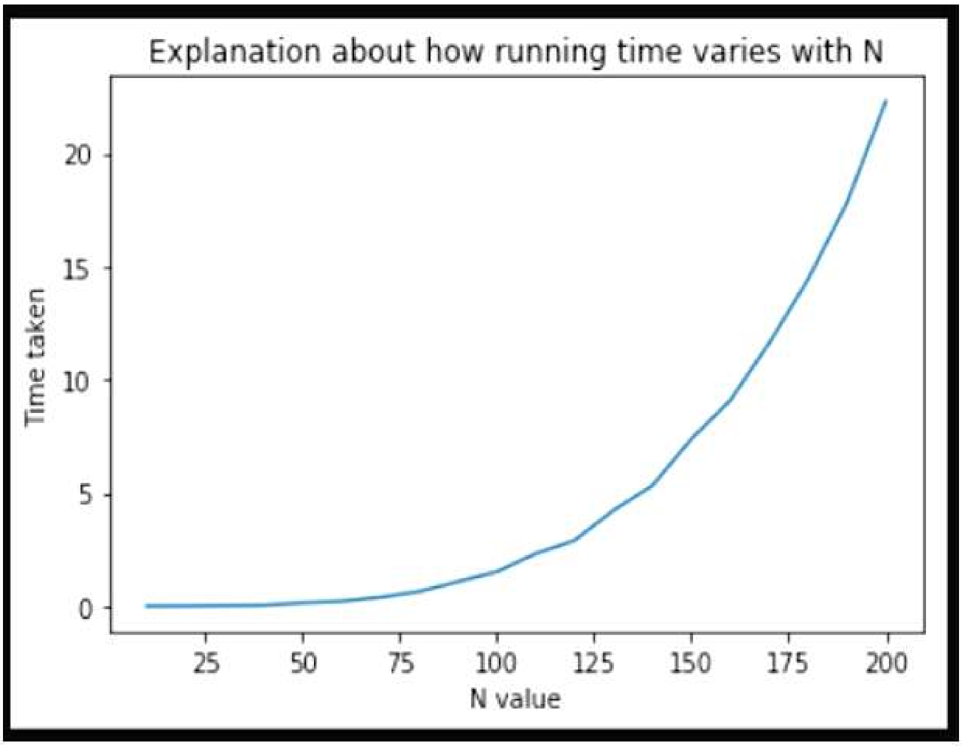
time N	DFS	BFS
N=1	0	0
N=2	0	0
N=3	0	0
N=4	0.004s	0.120s
N=5	0.2929s	1.93s

=> So DFS takes less running time here.

Hence in successor_3 function I used DFS for a specific reason, BFS would add one rook at a time and searches for the solutions in the same level, where as when we use DFS would go deeper into the branch and as per our filtering it checks the N rooks condition and gives out fast solution.

4) The maximum value that I got for the n-rooks' program is N = 255.

The graph for how run time varies with N (# of Rooks) value is given below:



As per the graph, as the N value increases, the time taken for the code to execute also increases exponentially.