Aaron New

8 Queens Problem

1a. Formalize the constraints satisfaction problem (CSP) by means of defining:

V: A set of variables

64 variables $Cell_{1,1}$…, $Cell_{8,8}$ this would be our board

D: A set of domains, one for each Variable in V

Q_1 … Q_8 can occupy any of the variables (our Queens)

v: Each domain in D consists of a set of allowable values,

Allowable values: occupied, or attack

C: A set of constraints that specify allowable combinations of values.

- A queen may move left, right, up, down, as well as diagonal.
- Queens may not intersect with each other on the board as they will attack.
- If a Queen is placed on the board then the proceeding tiles around it will be considered attacked and it will be an invalid move.

Goal: Define the goal for this given 8-Queens problem.

The goal of this 8-Queens problem is to be able to place 8 queens individually on a given cell on the board. The Queens cannot intersect as they can move up, down, left, and right, as well as diagonal. If they intersect the program will backtrack and try again. Until a total of 8 queens are placed on the board.

1b. In definition, what is the solution to the CSP? Define the CSP is satisfiable.

By definition: a constraint satisfaction problem (CSP) is solved when each variable has a value that satisfies the constraints on the variable. A constraint satisfaction problem (CSP) is unsatisfiable if no solution exists. Otherwise, we can call this a consistent complete assignment when a solution is found.

1c. A CSP could be viewed as a search problem. Use backtracking search to derive a solution for this 8-Queens problem. (To respond this question, you need to provide partial search tree which demonstrate a path that fails to reach a goal and a path that reach a goal.

This tree will get us to a solution:
Key:
A placed queen is a green number 1
Starting queen at 4,4 (base of zero) is highlighted
When we backtrack, we are RED
When we find the move AFTER backtracking we are Green and highlighted

Goal one of 3 possible solutions

Current Board || Attack

```
0 0 1 0 0 0 0 0     2 1 1 1 2 1 1 1
0 0 0 0 0 0 0 0     0 2 1 1 1 0 0 1
0 0 0 0 0 0 0 0     1 0 2 0 2 0 1 0
0 0 0 0 0 0 0 0     0 0 1 1 1 2 0 0
0 0 0 0 1 0 0 0     1 1 2 1 1 1 2 1
0 0 0 0 0 0 0 0     0 0 1 1 1 1 0 1
0 0 0 0 0 0 0 0     0 0 2 0 1 0 1 0
0 0 0 0 0 0 0 0     0 1 1 0 1 0 0 1
-----------------------
0 0 1 0 0 0 0 0     3 2 1 1 2 1 1 1
1 0 0 0 0 0 0 0     1 3 2 2 2 1 1 2
0 0 0 0 0 0 0 0     2 1 2 0 2 0 1 0
0 0 0 0 0 0 0 0     1 0 2 1 1 2 0 0
0 0 0 0 1 0 0 0     2 1 2 2 1 1 2 1
0 0 0 0 0 0 0 0     1 0 1 1 2 1 0 1
0 0 0 0 0 0 0 0     1 0 2 0 1 1 1 0
0 0 0 0 0 0 0 0     1 1 1 0 1 0 1 1
-----------------------
0 0 1 0 0 0 0 0     3 3 1 2 2 2 1 1
1 0 0 0 0 0 0 0     1 3 3 3 3 1 1 2
0 0 0 1 0 0 0 0     3 2 3 1 3 1 2 1
0 0 0 0 0 0 0 0     1 0 3 2 2 2 0 0
0 0 0 0 1 0 0 0     2 2 2 3 1 2 2 1
0 0 0 0 0 0 0 0     2 0 1 2 2 1 1 1
0 0 0 0 0 0 0 0     1 0 2 1 1 1 1 1
0 0 0 0 0 0 0 0     1 1 1 1 1 0 1 1
-----------------------
0 0 1 0 0 0 0 0     3 4 1 2 3 2 1 1
1 0 0 0 0 0 0 0     1 4 3 4 3 1 1 2
0 0 0 1 0 0 0 0     4 3 4 1 3 1 2 1
0 1 0 0 0 0 0 0     2 1 4 3 3 3 1 1
0 0 0 0 1 0 0 0     3 3 3 3 1 2 2 1
0 0 0 0 0 0 0 0     2 1 1 3 2 1 1 1
0 0 0 0 0 0 0 0     1 1 2 1 2 1 1 1
0 0 0 0 0 0 0 0     1 2 1 1 1 1 1 1
-----------------------
0 0 1 0 0 0 0 0     3 3 1 3 2 2 2 1
1 0 0 0 0 0 0 0     1 3 3 3 4 1 2 2
0 0 0 1 0 0 0 0     3 2 3 1 3 2 3 2
0 0 0 0 0 0 1 0     2 1 4 3 3 3 1 1
0 0 0 0 1 0 0 0     2 2 2 3 1 3 3 2
0 0 0 0 0 0 0 0     2 0 1 2 3 1 2 1
0 0 0 0 0 0 0 0     1 0 2 2 1 1 2 1
0 0 0 0 0 0 0 0     1 1 2 1 1 0 2 1
```

Solution 1							
0	0	1	0	0	0	0	0
0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	1
1	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0
0	0	0	1	0	0	0	0
Solution 2							
0	0	0	0	0	1	0	0
0	0	0	1	0	0	0	0
0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0
1	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0
Solution 3							
0	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0
1	0	0	0	0	0	0	0
0	0	0	0	0	0	1	0
0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	1
0	1	0	0	0	0	0	0
0	0	0	1	0	0	0	0

----

0 0 1 0 0 0 0 0   3 3 1 2 3 2 1 2
1 0 0 0 0 0 0 0   1 3 3 3 3 2 1 3
0 0 0 1 0 0 0 0   3 2 3 1 3 1 3 2
0 0 0 0 0 0 0 1   2 1 4 3 3 3 1 1
0 0 0 0 1 0 0 0   2 2 2 3 1 2 3 2
0 0 0 0 0 0 0 0   2 0 1 2 2 2 1 2
0 0 0 0 0 0 0 0   1 0 2 1 2 1 1 2
0 0 0 0 0 0 0 0   1 1 1 2 1 0 1 2

----

0 0 1 0 0 0 0 0   3 2 1 2 2 2 1 2
1 0 0 0 0 0 0 0   1 3 2 2 3 2 2 2
0 0 0 0 0 1 0 0   3 2 3 1 3 1 2 1
0 0 0 0 0 0 0 0   1 0 2 1 2 3 1 0
0 0 0 0 1 0 0 0   2 1 2 3 1 2 2 2
0 0 0 0 0 0 0 0   1 0 2 1 2 2 0 1
0 0 0 0 0 0 0 0   1 1 2 0 1 2 1 0
0 0 0 0 0 0 0 0   2 1 1 0 1 1 1 1

----

0 0 1 0 0 0 0 0   3 3 1 2 3 2 1 2
1 0 0 0 0 0 0 0   1 4 2 3 3 2 2 2
0 0 0 0 0 1 0 0   4 3 4 1 3 1 2 1
0 1 0 0 0 0 0 0   2 1 3 2 3 4 2 1
0 0 0 0 1 0 0 0   3 2 3 3 1 2 2 2
0 0 0 0 0 0 0 0   1 1 2 2 2 2 0 1
0 0 0 0 0 0 0 0   1 2 2 0 2 2 1 0
0 0 0 0 0 0 0 0   2 2 1 0 1 2 1 1

----

0 0 1 0 0 0 0 0   3 4 1 2 3 2 2 2
1 0 0 0 0 0 0 0   1 4 3 3 3 2 3 2
0 0 0 0 0 1 0 0   4 3 4 2 3 1 3 1
0 1 0 0 0 0 0 0   2 1 3 2 4 4 3 1
0 0 0 0 1 0 0 0   3 2 3 3 1 3 3 3
0 0 0 0 0 0 1 0   2 2 3 3 3 3 1 2
0 0 0 0 0 0 0 0   1 2 2 0 2 3 2 1
0 0 0 0 0 0 0 0   2 2 1 0 2 2 2 1

----

0 0 1 0 0 0 0 0   3 4 1 3 3 2 2 2
1 0 0 0 0 0 0 0   1 4 3 4 3 2 3 2
0 0 0 0 0 1 0 0   4 3 4 3 3 1 3 2
0 1 0 0 0 0 0 0   3 1 3 3 4 4 4 1
0 0 0 0 1 0 0 0   3 3 3 4 1 4 3 3
0 0 0 0 0 0 1 0   2 2 4 4 4 3 1 2
0 0 0 1 0 0 0 0   2 3 3 1 3 4 3 2
0 0 0 0 0 0 0 0   2 2 2 1 3 2 2 1

----

0 0 1 0 0 0 0 0   3 2 1 2 3 2 1 3
1 0 0 0 0 0 0 0   1 3 2 2 3 3 2 3
0 0 0 0 0 1 0 0   3 2 3 1 3 1 3 2
0 0 0 0 0 0 0 1   2 1 3 2 3 4 2 1
0 0 0 0 1 0 0 0   2 1 2 3 1 2 3 3
0 0 0 0 0 0 0 0   1 0 2 1 2 3 0 2
0 0 0 0 0 0 0 0   1 1 2 0 2 2 1 1
0 0 0 0 0 0 0 0   2 1 1 1 1 1 1 2

----

0 0 1 0 0 0 0 0   3 3 1 2 3 2 2 3
1 0 0 0 0 0 0 0   1 3 3 2 3 3 3 3
0 0 0 0 0 1 0 0   3 2 3 2 3 1 4 2
0 0 0 0 0 0 0 1   2 1 3 2 4 4 3 1
0 0 0 0 1 0 0 0   2 1 2 3 1 3 4 4
0 0 0 0 0 0 1 0   2 1 3 2 3 4 1 3
0 0 0 0 0 0 0 0   1 1 2 0 2 3 2 2
0 0 0 0 0 0 0 0   2 1 1 1 2 1 2 2

----

0 0 1 0 0 0 0 0   3 3 1 3 3 2 2 3
1 0 0 0 0 0 0 0   1 3 3 3 3 3 3 3
0 0 0 0 0 1 0 0   3 2 3 3 3 1 4 3
0 0 0 0 0 0 0 1   3 1 3 3 4 4 4 1
0 0 0 0 1 0 0 0   2 2 2 4 1 4 4 4
0 0 0 0 0 0 1 0   2 1 4 3 4 4 1 3
0 0 0 1 0 0 0 0   2 2 3 1 3 4 3 3
0 0 0 0 0 0 0 0   2 1 2 2 3 1 2 2

----

0 0 1 0 0 0 0 0   3 2 1 1 2 2 1 2
1 0 0 0 0 0 0 0   1 3 2 2 2 1 2 3
0 0 0 0 0 0 0 1   3 2 3 1 3 1 2 1
0 0 0 0 0 0 0 0   1 0 2 1 1 2 1 1
0 0 0 0 1 0 0 0   2 1 2 2 1 2 2 2
0 0 0 0 0 0 0 0   1 0 1 1 3 1 0 2
0 0 0 0 0 0 0 0   1 0 2 1 1 1 1 1
0 0 0 0 0 0 0 0   1 1 2 0 1 0 1 2

----

0 0 1 0 0 0 0 0   3 3 1 1 3 2 1 2
1 0 0 0 0 0 0 0   1 4 2 3 2 1 2 3
0 0 0 0 0 0 0 1   4 3 4 1 3 1 2 1
0 1 0 0 0 0 0 0   2 1 3 2 2 3 2 2
0 0 0 0 1 0 0 0   3 2 3 2 1 2 2 2
0 0 0 0 0 0 0 0   1 1 1 2 3 1 0 2
0 0 0 0 0 0 0 0   1 1 2 1 2 1 1 1
0 0 0 0 0 0 0 0   1 2 2 0 1 1 1 2

```
-----------------------
0 0 1 0 0 0 0 0      3 4 1 1 3 2 2 2
1 0 0 0 0 0 0 0      1 4 3 3 2 1 3 3
0 0 0 0 0 0 0 1      4 3 4 2 3 1 3 1
0 1 0 0 0 0 0 0      2 1 3 2 3 3 3 2
0 0 0 0 1 0 0 0      3 2 3 2 1 3 3 3
0 0 0 0 0 0 1 0      2 2 2 3 4 2 1 3
0 0 0 0 0 0 0 0      1 1 2 1 2 2 2 2
0 0 0 0 0 0 0 0      1 2 2 0 2 1 2 2
-----------------------
0 0 1 0 0 0 0 0      2 1 1 1 3 2 2 1
0 0 0 0 0 1 0 0      1 3 2 2 2 1 1 2
0 0 0 0 0 0 0 0      1 0 2 0 3 1 2 0
0 0 0 0 0 0 0 0      0 0 1 2 1 3 0 1
0 0 0 0 1 0 0 0      1 1 3 1 1 2 2 1
0 0 0 0 0 0 0 0      0 1 1 1 1 2 0 1
0 0 0 0 0 0 0 0      1 0 2 0 1 1 1 0
0 0 0 0 0 0 0 0      0 1 1 0 1 1 0 1
-----------------------
0 0 1 0 0 0 0 0      2 2 1 2 3 2 2 1
0 0 0 0 0 1 0 0      2 4 3 2 2 1 1 2
0 1 0 0 0 0 0 0      2 1 3 1 4 2 3 1
0 0 0 0 0 0 0 0      1 1 2 2 1 3 0 1
0 0 0 0 1 0 0 0      1 2 3 2 1 2 2 1
0 0 0 0 0 0 0 0      0 2 1 1 2 2 0 1
0 0 0 0 0 0 0 0      1 1 2 0 1 2 1 0
0 0 0 0 0 0 0 0      0 2 1 0 1 1 1 1
-----------------------
0 0 1 0 0 0 0 0      2 2 1 3 3 2 3 1
0 0 0 0 0 1 0 0      2 4 3 2 3 1 2 2
0 1 0 0 0 0 0 0      2 1 3 1 4 3 4 2
0 0 0 0 0 0 1 0      2 2 3 3 2 4 1 2
0 0 0 0 1 0 0 0      1 2 3 2 1 3 3 2
0 0 0 0 0 0 0 0      0 2 1 1 3 2 1 1
0 0 0 0 0 0 0 0      1 1 2 1 1 2 2 0
0 0 0 0 0 0 0 0      0 2 2 0 1 1 2 1
-----------------------
0 0 1 0 0 0 0 0      2 2 1 2 3 3 2 1
0 0 0 0 0 1 0 0      1 3 3 3 3 1 1 2
0 0 0 1 0 0 0 0      2 1 3 1 4 2 3 1
0 0 0 0 0 0 0 0      0 0 2 3 2 3 0 1
0 0 0 0 1 0 0 0      1 2 3 2 1 3 2 1
0 0 0 0 0 0 0 0      1 1 1 2 1 2 1 1
0 0 0 0 0 0 0 0      1 0 2 1 1 1 1 1
0 0 0 0 0 0 0 0      0 1 1 1 1 1 0 1
-----------------------
```

NO SOLUTION HERE! WE NEED TO BACKTRACK!!!

0 0 1 0 0 0 0 0	3 2 1 3 3 3 2 1
0 0 0 0 0 1 0 0	2 3 4 3 3 1 1 2
0 0 0 1 0 0 0 0	3 2 3 1 4 2 3 1
1 0 0 0 0 0 0 0	1 1 3 4 3 4 1 2
0 0 0 0 1 0 0 0	2 3 3 2 1 3 2 1
0 0 0 0 0 0 0 0	2 1 2 2 1 2 1 1
0 0 0 0 0 0 0 0	2 0 2 2 1 1 1 1
0 0 0 0 0 0 0 0	1 1 1 1 2 1 0 1

----

0 0 1 0 0 0 0 0	2 3 1 2 4 3 2 1
0 0 0 0 0 1 0 0	1 4 3 4 3 1 1 2
0 0 0 1 0 0 0 0	3 2 4 1 4 2 3 1
0 1 0 0 0 0 0 0	1 1 3 4 3 4 1 2
0 0 0 0 1 0 0 0	2 3 4 2 1 3 2 1
0 0 0 0 0 0 0 0	1 2 1 3 1 2 1 1
0 0 0 0 0 0 0 0	1 1 2 1 2 1 1 1
0 0 0 0 0 0 0 0	0 2 1 1 1 2 0 1

----

0 0 1 0 0 0 0 0	2 2 1 3 3 3 3 1
0 0 0 0 0 1 0 0	1 3 3 3 4 1 2 2
0 0 0 1 0 0 0 0	2 1 3 1 4 3 4 2
0 0 0 0 0 0 1 0	1 1 3 4 3 4 1 2
0 0 0 0 1 0 0 0	1 2 3 2 1 4 3 2
0 0 0 0 0 0 0 0	1 1 1 2 2 2 2 1
0 0 0 0 0 0 0 0	1 0 2 2 1 1 2 1
0 0 0 0 0 0 0 0	0 1 2 1 1 1 1 1

----

0 0 1 0 0 0 0 0	2 1 1 1 3 3 2 2
0 0 0 0 0 1 0 0	1 3 2 2 2 1 2 3
0 0 0 0 0 0 0 1	2 1 3 1 4 2 3 1
0 0 0 0 0 0 0 0	0 0 1 2 1 3 1 2
0 0 0 0 1 0 0 0	1 1 3 1 1 3 2 2
0 0 0 0 0 0 0 0	0 1 1 1 2 2 0 2
0 0 0 0 0 0 0 0	1 0 2 1 1 1 1 1
0 0 0 0 0 0 0 0	0 1 2 0 1 1 0 2

----

0 0 1 0 0 0 0 0	3 1 1 2 3 3 2 2
0 0 0 0 0 1 0 0	2 3 3 2 2 1 2 3
0 0 0 0 0 0 0 1	3 2 3 1 4 2 3 1
1 0 0 0 0 0 0 0	1 1 2 3 2 4 2 3
0 0 0 0 1 0 0 0	2 2 3 1 1 3 2 2
0 0 0 0 0 0 0 0	1 1 2 1 2 2 0 2
0 0 0 0 0 0 0 0	2 0 2 2 1 1 1 1
0 0 0 0 0 0 0 0	1 1 2 0 2 1 0 2

```
0 0 1 0 0 0 0 0     3 2 1 2 3 3 3 2
0 0 0 0 0 1 0 0     2 3 4 2 2 1 3 3
0 0 0 0 0 0 0 1     3 2 3 2 4 2 4 1
1 0 0 0 0 0 0 0     1 1 2 3 3 4 3 3
0 0 0 0 1 0 0 0     2 2 3 1 1 4 3 3
0 0 0 0 0 0 1 0     2 2 3 2 3 3 1 3
0 0 0 0 0 0 0 0     2 0 2 2 1 2 2 2
0 0 0 0 0 0 0 0     1 1 2 0 3 1 1 2
```

```
0 0 1 0 0 0 0 0     3 3 1 2 3 3 3 3
0 0 0 0 0 1 0 0     2 4 4 2 2 1 4 3
0 0 0 0 0 0 0 1     3 3 3 2 4 3 4 1
1 0 0 0 0 0 0 0     1 2 2 3 4 4 3 3
0 0 0 0 1 0 0 0     2 3 3 2 1 4 3 3
0 0 0 0 0 0 1 0     3 3 4 2 3 3 1 3
0 1 0 0 0 0 0 0     3 1 3 3 2 3 3 3
0 0 0 0 0 0 0 0     2 2 3 0 3 1 1 2
```

THIS IS A SOLUTION!!

```
0 0 1 0 0 0 0 0     3 3 1 3 3 3 3 3
0 0 0 0 0 1 0 0     2 4 4 3 2 1 4 3
0 0 0 0 0 0 0 1     3 3 3 3 4 3 4 1
1 0 0 0 0 0 0 0     1 2 2 4 4 4 3 4
0 0 0 0 1 0 0 0     3 3 3 3 1 4 4 3
0 0 0 0 0 0 1 0     3 4 4 3 3 4 1 3
0 1 0 0 0 0 0 0     3 1 4 4 3 3 3 3
0 0 0 1 0 0 0 0     3 3 4 1 4 2 2 3
```

1d. What is constraint propagation? Using this given 8-Queens problem, show an example of resulting after applying the constraint propagation. (one application to a given queen is good enough).

Defining Constraint Propagation: it refers to a technique of "looking ahead" at the yet unassigned variables in the search performed. Propagation is applied during the search, potentially at every node of the search tree. There are two main types of propagation: forward checking and generalized arc consistency

----

```
0 0 1 0 0 0 0 0     2 2 1 2 3 2 2 1
0 0 0 0 0 1 0 0     2 4 3 2 2 1 1 2
0 1 0 0 0 0 0 0     2 1 3 1 4 2 3 1
0 0 0 0 0 0 0 0     1 1 2 2 1 3 0 1
0 0 0 0 1 0 0 0     1 2 3 2 1 2 2 1
0 0 0 0 0 0 0 0     0 2 1 1 2 2 0 1
0 0 0 0 0 0 0 0     1 1 2 0 1 2 1 0
0 0 0 0 0 0 0 0     0 2 1 0 1 1 1 1
```

```
0 0 1 0 0 0 0 0     2 2 1 3 3 2 3 1
0 0 0 0 0 1 0 0     2 4 3 2 3 1 2 2
0 1 0 0 0 0 0 0     2 1 3 1 4 3 4 2
0 0 0 0 0 0 1 0     2 2 3 3 2 4 1 2
0 0 0 0 1 0 0 0     1 2 3 2 1 3 3 2      ← Example
0 0 0 0 0 0 0 0     0 2 1 1 3 2 1 1
0 0 0 0 0 0 0 0     1 1 2 1 1 2 2 0
0 0 0 0 0 0 0 0     0 2 2 0 1 1 2 1
```

In this bottom example we can see the program looking ahead. It notices that there are two 0's in the same column, this forces the program to backtrack and to try to find the next best path. This is due to the program looking ahead based on the number of attacks a cell has. When the cell is 0 we can place a queen, but not if it intersects with a new or an existing queen.

Moving forward this is how the result shows after the program looks forward

----

```
0 0 1 0 0 0 0 0     2 2 1 2 3 3 2 1
0 0 0 0 0 1 0 0     1 3 3 3 3 1 1 2
0 0 0 1 0 0 0 0     2 1 3 1 4 2 3 1
0 0 0 0 0 0 0 0     0 0 2 3 2 3 0 1
0 0 0 0 1 0 0 0     1 2 3 2 1 3 2 1
0 0 0 0 0 0 0 0     1 1 1 2 1 2 1 1
0 0 0 0 0 0 0 0     1 0 2 1 1 1 1 1
0 0 0 0 0 0 0 0     0 1 1 1 1 1 0 1
```

```
0 0 1 0 0 0 0 0     3 2 1 3 3 3 2 1
0 0 0 0 0 1 0 0     2 3 4 3 3 1 1 2
0 0 0 1 0 0 0 0     3 2 3 1 4 2 3 1
1 0 0 0 0 0 0 0     1 1 3 4 3 4 1 2
0 0 0 0 1 0 0 0     2 3 3 2 1 3 2 1
0 0 0 0 0 0 0 0     2 1 2 2 1 2 1 1
0 0 0 0 0 0 0 0     2 0 2 2 1 1 1 1
0 0 0 0 0 0 0 0     1 1 1 1 2 1 0 1
```

```
0 0 1 0 0 0 0 0     2 3 1 2 4 3 2 1
0 0 0 0 0 1 0 0     1 4 3 4 3 1 1 2
0 0 0 1 0 0 0 0     3 2 4 1 4 2 3 1
0 1 0 0 0 0 0 0     1 1 3 4 3 4 1 2
0 0 0 0 1 0 0 0     2 3 4 2 1 3 2 1
0 0 0 0 0 0 0 0     1 2 1 3 1 2 1 1
0 0 0 0 0 0 0 0     1 1 2 1 2 1 1 1
0 0 0 0 0 0 0 0     0 2 1 1 1 2 0 1
```

```
0 0 1 0 0 0 0 0     2 2 1 3 3 3 3 1
0 0 0 0 0 1 0 0     1 3 3 3 4 1 2 2
0 0 0 1 0 0 0 0     2 1 3 1 4 3 4 2
0 0 0 0 0 0 1 0     1 1 3 4 3 4 1 2
0 0 0 0 1 0 0 0     1 2 3 2 1 4 3 2
0 0 0 0 0 0 0 0     1 1 1 2 2 2 2 1
0 0 0 0 0 0 0 0     1 0 2 2 1 1 2 1
0 0 0 0 0 0 0 0     0 1 2 1 1 1 1 1
```

----

0 0 1 0 0 0 0 0	2 1 1 1 3 3 2 2	
0 0 0 0 0 1 0 0	1 3 2 2 2 1 2 3	
0 0 0 0 0 0 0 1	2 1 3 1 4 2 3 1	
0 0 0 0 0 0 0 0	0 0 1 2 1 3 1 2	
0 0 0 0 1 0 0 0	1 1 3 1 1 3 2 2	← This will go to find a solution
0 0 0 0 0 0 0 0	0 1 1 1 2 2 0 2	
0 0 0 0 0 0 0 0	1 0 2 1 1 1 1 1	
0 0 0 0 0 0 0 0	0 1 2 0 1 1 0 2	

1e. Use the CSP with forward checking algorithm for solving this given 8-Queens problem.
Using this method, we could solve the forward checking algorithm

```
boolean attacked(int x, int y)
{
   return (attacked[x][y] > 0);
}
```

This method will check inside of the attacked array to see if the cell has a value that is greater than 0. If this is not true we will utilize the recursive method solve(int row) where when we place a queen in a valid space we will move to the next row, if not we will remove the queen and increment our space

```
void solve(int row)
{
   int col;
   //prints out our current state showing the moves we make
   printState();

   // ******************************************
   // Check if all queens are placed
   // ******************************************
   if (row == 8)
   { // ----------- This case stops the recursion

      //to show the attacking count on each cell uncomment this
      //printState();
      printSol();
      solutionCount++;
      return;
   }

   // --------------------------------------------------
   // Try every column to place queen in row "row":
   // --------------------------------------------------
   for (col = 0; col < 8; col++)
   {
      //to start put a queen on 5 , 5 to fulfill the CSP
      if(queenCount == 0){

         putQueen(4, 4);
         printState();
         queenCount++;
         row = 0;
         col = 0;
      }
      //when we get to this row we need to skip in order for the program to not backtrack and cause 0 solutions
      if(row == 4 && col == 4){
         row = row + 1;
         col = col + 1;
      }
      //when the row and column of the attack array are 0 we will try and place a queen
```

```
      if ( ! attacked(row, col) )
      {
         // Make Move
         putQueen(row, col);
         // solve smaller problem
         solve(row+1);
         // Undo Move
         removeQueen(row, col);

      }
   }
}
```

1f. Using the integer value of row_number + row_number and the integer value of row_number - row_number as specification of the constraints for 8-Queens problem, construct a search tree for finding the goal.

```
0 0 1 0 0 0 0 0       2 1 1 1 3 3 2 2
0 0 0 0 0 1 0 0       1 3 2 2 2 1 2 3
0 0 0 0 0 0 0 1       2 1 3 1 4 2 3 Q
0 0 0 0 0 0 0 0       0 0 1 2 1 3 1 2
0 0 0 0 1 0 0 0       1 1 3 1 1 3 2 2
0 0 0 0 0 0 0 0       0 1 1 1 2 2 0 2
0 0 0 0 0 0 0 0       1 0 2 1 1 1 1 1
0 0 0 0 0 0 0 0       0 1 2 0 1 1 0 2
-----------------------
0 0 1 0 0 0 0 0       3 1 1 2 3 3 2 2
0 0 0 0 0 1 0 0       2 3 3 2 2 1 2 3
0 0 0 0 0 0 0 1       3 2 3 1 4 2 3 1
1 0 0 0 0 0 0 0       Q 1 2 3 2 4 2 3
0 0 0 0 1 0 0 0       2 2 3 1 1 3 2 2
0 0 0 0 0 0 0 0       1 1 2 1 2 2 0 2
0 0 0 0 0 0 0 0       2 0 2 2 1 1 1 1
0 0 0 0 0 0 0 0       1 1 2 0 2 1 0 2
-----------------------
0 0 1 0 0 0 0 0       3 2 1 2 3 3 3 2
0 0 0 0 0 1 0 0       2 3 4 2 2 1 3 3
0 0 0 0 0 0 0 1       3 2 3 2 4 2 4 1
1 0 0 0 0 0 0 0       1 1 2 3 3 4 3 3
0 0 0 0 1 0 0 0       2 2 3 1 1 4 3 3
0 0 0 0 0 0 1 0       2 2 3 2 3 3 Q 3
0 0 0 0 0 0 0 0       2 0 2 2 1 2 2 2
0 0 0 0 0 0 0 0       1 1 2 0 3 1 1 2
-----------------------
0 0 1 0 0 0 0 0       3 3 1 2 3 3 3 3
0 0 0 0 0 1 0 0       2 4 4 2 2 1 4 3
0 0 0 0 0 0 0 1       3 3 3 2 4 3 4 1
1 0 0 0 0 0 0 0       1 2 2 3 4 4 3 3
0 0 0 0 1 0 0 0       2 3 3 2 1 4 3 3
0 0 0 0 0 0 1 0       3 3 4 2 3 3 1 3
0 1 0 0 0 0 0 0       3 Q 3 3 2 3 3 3
0 0 0 0 0 0 0 0       2 2 3 0 3 1 1 2
-----------------------
```

THIS IS A SOLUTION!!

0 0 1 0 0 0 0 0	3 3 1 3 3 3 3 3
0 0 0 0 0 1 0 0	2 4 4 3 2 1 4 3
0 0 0 0 0 0 0 1	3 3 3 3 4 3 4 1
1 0 0 0 0 0 0 0	1 2 2 4 4 4 3 4
0 0 0 0 1 0 0 0	3 3 3 3 1 4 4 3
0 0 0 0 0 0 1 0	3 4 4 3 3 4 1 3
0 1 0 0 0 0 0 0	3 1 4 4 3 3 3 3
0 0 0 1 0 0 0 0	3 3 4 Q 4 2 2 3

1g. For the given 8-Queens problem, construct a constraint graph for the problem. Then, use arc consistency technique to solve this given problem.

For this example, we will try to place queens on a board.

> Highlighted is the original placement for the queen as it is arbitrarily placed on the value (5, 5) or (4,4) with a base of zero.
> We will show that we place them (the green colored “1”).
> We will also show that when we can’t make a move we turn RED.
> We will also show that when we can’t make a move we back track and show the next move with a highlighted 1 with a green text color

0 0 1 0 0 0 0 0	2 1 1 1 2 1 1 1
0 0 0 0 0 0 0 0	0 2 1 1 1 0 0 1
0 0 0 0 0 0 0 0	1 0 2 0 2 0 1 0
0 0 0 0 0 0 0 0	0 0 1 1 1 2 0 0
0 0 0 0 1 0 0 0	1 1 2 1 1 1 2 1
0 0 0 0 0 0 0 0	0 0 1 1 1 1 0 1
0 0 0 0 0 0 0 0	0 0 2 0 1 0 1 0
0 0 0 0 0 0 0 0	0 1 1 0 1 0 0 1

----

0 0 1 0 0 0 0 0	3 2 1 1 2 1 1 1
1 0 0 0 0 0 0 0	1 3 2 2 2 1 1 2
0 0 0 0 0 0 0 0	2 1 2 0 2 0 1 0
0 0 0 0 0 0 0 0	1 0 2 1 1 2 0 0
0 0 0 0 1 0 0 0	2 1 2 2 1 1 2 1
0 0 0 0 0 0 0 0	1 0 1 1 2 1 0 1
0 0 0 0 0 0 0 0	1 0 2 0 1 1 1 0
0 0 0 0 0 0 0 0	1 1 1 0 1 0 1 1

----

0 0 1 0 0 0 0 0	3 3 1 2 2 2 1 1
1 0 0 0 0 0 0 0	1 3 3 3 3 1 1 2
0 0 0 1 0 0 0 0	3 2 3 1 3 1 2 1
0 0 0 0 0 0 0 0	1 0 3 2 2 2 0 0
0 0 0 0 1 0 0 0	2 2 2 3 1 2 2 1
0 0 0 0 0 0 0 0	2 0 1 2 2 1 1 1
0 0 0 0 0 0 0 0	1 0 2 1 1 1 1 1
0 0 0 0 0 0 0 0	1 1 1 1 1 0 1 1

----

0 0 1 0 0 0 0 0	3 4 1 2 3 2 1 1
1 0 0 0 0 0 0 0	1 4 3 4 3 1 1 2
0 0 0 1 0 0 0 0	4 3 4 1 3 1 2 1
0 1 0 0 0 0 0 0	2 1 4 3 3 3 1 1
0 0 0 0 1 0 0 0	3 3 3 3 1 2 2 1
0 0 0 0 0 0 0 0	2 1 1 3 2 1 1 1
0 0 0 0 0 0 0 0	1 1 2 1 2 1 1 1
0 0 0 0 0 0 0 0	1 2 1 1 1 1 1 1

----

0 0 1 0 0 0 0 0	3 3 1 3 2 2 2 1
1 0 0 0 0 0 0 0	1 3 3 3 4 1 2 2
0 0 0 1 0 0 0 0	3 2 3 1 3 2 3 2
0 0 0 0 0 0 1 0	2 1 4 3 3 3 1 1
0 0 0 0 1 0 0 0	2 2 2 3 1 3 3 2
0 0 0 0 0 0 0 0	2 0 1 2 3 1 2 1
0 0 0 0 0 0 0 0	1 0 2 2 1 1 2 1
0 0 0 0 0 0 0 0	1 1 2 1 1 0 2 1

Define Arc Consistency: Arc X→Y is consistent iff for every value of x of X there is some allowed y for Y.
Using this graph above, we can show that when there isn't a 0 available inside of the attacked[i][j] array the next step that doesn't intersect with another 0 we will have to back track as there isn't another allowed value in that cell.

1h. Define states and state space representation with their production rules (actions), goal test and evaluation for this given CSP 8-Queens problem.

We are looking for a valid position to place a queen on the board, one that does not violate the non-attack constraint.

Initial state:

Where we start our search, we place a queen at 5,5 on the board and we will then move our position to 0, 0 or the top left of the board

The states are:

N-Queens placed on board
No attacks

Our goal test will be:

Checking for the ability to place a queen where there are 0 attacks occurring.

Our goal state will be:

When we have placed all 8 queens on the board and they do not intersect / attack each other

Operators (Actions) that are used to modify these states:

Place a queen
Remove a queen
Move a queen

Provide the task environments for the 8-Queens problem solver. (Use the correct words and do not use yes or no.)

Task Environment	Observable	Agents	Deterministic	Episodic	Static	Discrete
Problem-Solving Agent	Fully Observable	Single Agent	Deterministic	Episodic	Static	Discrete

Give its PEAS description for the 8-Queens problem.

2a. Use propositional logic to describe the constraints satisfaction problem.
What are in the Knowledge Base system.

Define a Knowledge Base: Informally a KB or knowledge base is a set of sentences. Not in the literal sense. The sentences are instructions or assertions that a program will use. When a program needs to make a decision a lot of times it will be quicker and more efficient if the system has access to the knowledge base of the environment that it is acting in.

Constraint languages are indeed logics and constraint solving is a form of logical reasoning

If a sentence α is true in a model 'm', then we say that m satisfies α , or that m is a model of α.

Satisfiability can be checked by enumerating the possible models until one is found that satisfies the sentence

2b. "The Knowledge base KB entails sentence α if and only if α is true in all worlds where KB is true". Based on this, give two entailments for the 8-Queens problem.

Define: Before we give an example of KB entailment, we will need to define it. A Knowledge base entailment means that one thing follows from another. Entailment is a relationship between sentences (i.e. syntax) that is based on semantics.

Using this type of logic, we can define the entailment of the problem "8 - Queens."

One of these entailments is that when a Queen is placed on the board we add it's attack to the attacking[i][j] array.
The second entailment is if the array "attacking[i][j]" is > 0 then we will move on to the next space as it currently conflicts (we don't want to place a Queen where it will be attacked as it doesn't follow the constraints placed)
Our Queen is entailed by Knowledge base, iff attacking[i][j] is zero. If it is greater we will not be able to move here!

Code for algorithm:

```
//Author: Aaron New
//Class: CS380 Artificial Intelligence

class Queens
{
   // Count # solutions found
   int solutionCount = 0;
   //counts our queen and is used as a gate to place the first queen on cell 4 , 4 as we are using 0 as a base
   int queenCount = 0;
   int[][] board = {
         {0, 0, 0, 0, 0, 0, 0, 0},
         {0, 0, 0, 0, 0, 0, 0, 0},
         {0, 0, 0, 0, 0, 0, 0, 0},
         {0, 0, 0, 0, 0, 0, 0, 0},
         {0, 0, 0, 0, 0, 0, 0, 0},
         {0, 0, 0, 0, 0, 0, 0, 0},
         {0, 0, 0, 0, 0, 0, 0, 0},
         {0, 0, 0, 0, 0, 0, 0, 0} };

   int[][] attacked = {
         {0, 0, 0, 0, 0, 0, 0, 0},
         {0, 0, 0, 0, 0, 0, 0, 0},
         {0, 0, 0, 0, 0, 0, 0, 0},
         {0, 0, 0, 0, 0, 0, 0, 0},
         {0, 0, 0, 0, 0, 0, 0, 0},
         {0, 0, 0, 0, 0, 0, 0, 0},
         {0, 0, 0, 0, 0, 0, 0, 0},
         {0, 0, 0, 0, 0, 0, 0, 0} };

   // ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
   // solve(row): try to place a queen at row "row"
   // ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
   void solve(int row)
{
   int col;
   //prints out our current state showing the moves we make
   printState();

   // ******************************************
   // Check if all queens are placed
   // ******************************************
   if (row == 8)
   { // ----------- This case stops the recursion

      //to show the attacking count on each cell uncomment this
      //printState();
      printSol();
      solutionCount++;
      return;
```

```
    }
    // --------------------------------------------------
    // Try every column to place queen in row "row":
    // --------------------------------------------------
    for (col = 0; col < 8; col++)
    {
       //to start put a queen on 5 , 5 to fulfill the CSP
       if(queenCount == 0){

          putQueen(4, 4);
          printState();
          queenCount++;
          row = 0;
          col = 0;
       }
       //when we get to this row we need to skip in order for the program to not backtrack and cause 0
solutions
       if(row == 4 && col == 4){
          row = row + 1;
          col = col + 1;
       }
       //when the row and column of the attack array are 0 we will try and place a queen
       if ( ! attacked(row, col) )
       {
          // Make Move
          putQueen(row, col);
          // solve smaller problem
          solve(row+1);
          // Undo Move
          removeQueen(row, col);

       }
    }
}

    // ***********************************************************
    // attacked(x,y): Check if square (x,y) is attacked...
    // ***********************************************************
    boolean attacked(int x, int y)
    {
       return (attacked[x][y] > 0);
    }

    // ***********************************************************
    // putQueen(x,y): Put a queen on square (x,y)
    // ***********************************************************
    void putQueen(int x, int y)
    {
       int i, j;
       // Mark square occupied by queen
       board[x][y] = 1;
```

```
    // Mark all squares attacked by queen
    // queen will attack (x,y)
    attacked[x][y]++;
    // queen will attack the row x
    for (j = 0; j <= 7; j++)
        if (j != y){
            attacked[x][j]++;
        }
    // queen will attack the column y
    for (i = 0; i <= 7; i++)
        if (i != x){
            attacked[i][y]++;
        }
    // queen will attack the diagonals through (x,y)
    i = x-1;
    j = y-1;
    while ( (i >= 0) && (j >= 0) )
    { attacked[i][j]++;
        i--;
        j--;
    }

    i = x+1; j = y+1;
    while ( (i < 8) && (j < 8) )
    { attacked[i][j]++;
        i++;
        j++;
    }

    i = x-1; j = y+1;
    while ( (i >= 0) && (j < 8) )
    { attacked[i][j]++;
        i--;
        j++;
    }

    i = x+1; j = y-1;
    while ( (i < 8) && (j >= 0) )
    { attacked[i][j]++;
        i++;
        j--;
    }

}
// ***********************************************************
// removeQueen(x,y): Remove a queen from square (x,y)
// ***********************************************************
void removeQueen(int x, int y)
{
    int i, j;
```

```
    // Unmark position occupied by queen
    board[x][y] = 0;

    // Unmark squares that were attacked by queen
    attacked[x][y]--;           // Queen was attacking (x,y)

    for (j = 0; j <= 7; j++)  // Queen was attacking the row x
       if (j != y)
          attacked[x][j]--;

    for (i = 0; i <= 7; i++)  // Queen was attacking the column y
       if (i != x)
          attacked[i][y]--;

    // Queen was attacking the diagonals through (x,y)
    i = x-1; j = y-1;
    while ( (i >= 0) && (j >= 0) )
    { attacked[i][j]--;
       i--;
       j--;
    }

    i = x+1; j = y+1;
    while ( (i < 8) && (j < 8) )
    { attacked[i][j]--;
       i++;
       j++;
    }

    i = x-1; j = y+1;
    while ( (i >= 0) && (j < 8) )
    { attacked[i][j]--;
       i--;
       j++;
    }

    i = x+1; j = y-1;
    while ( (i < 8) && (j >= 0) )
    { attacked[i][j]--;
       i++;
       j--;
    }
}
```

```java
// Print solution found....
    void printSol()
    {
        int i, j;

        for (i = 0; i < 8; i++)
        { for (j = 0; j < 8; j++)
            System.out.print(board[i][j]+" ");
            System.out.println();
        }
        System.out.println("THIS IS A SOLUTION!!!!!");
        System.out.println("----------------------");
    }

    // Print current state....
    void printState()
    {
        int i, j;

        for (i = 0; i < 8; i++)
        { for (j = 0; j < 8; j++)
            System.out.print(board[i][j]+" ");
            System.out.print("          ");
            for (j = 0; j < 8; j++)
                System.out.print(attacked[i][j]+" ");
            System.out.println();
        }
        System.out.println("----------------------");
    }


}

public class EightQueens
{
    public static void main(String[] s)
    {
        Queens x = new Queens();
        x.solve(0);    // solve starts by trying to place a queen in row 0
        System.out.println("There are " + x.solutionCount + " solutions");
    }
}
```