CMSI 2130 – Classwork 6

Instructions:
This worksheet gives you some important practice with the fundamentals of CSPS!

- Provide answers to each of the following questions and write your responses in the blanks. If you are expected to show your work in arriving at a particular solution, space will be provided for you.
- Place the names of your group members below:

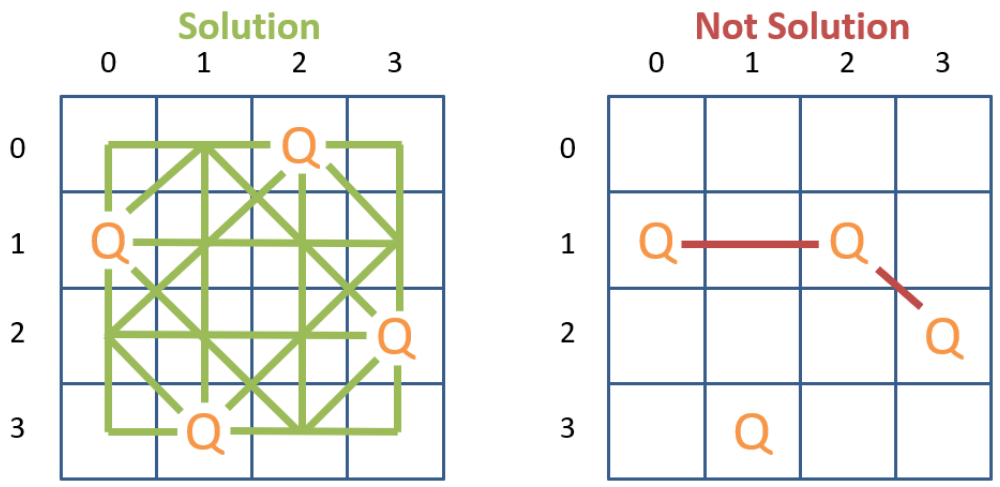
Group Members:

1. ________________________________
2. ________________________________
3. ________________________________

Problem 1 – CSP Formalization & Backtracking

Time for another rite of passage in this Tour de Algorithms: The N-Queens Problem. *The N-Queens Problem* is a CSP in which the goal is to place N chess queens on an $N \times N$ chess board such that no queen can capture any other. For those unfamiliar with chess, Queens are the most powerful piece that, on any given turn, can move as many tiles as desired in horizontal, vertical, and diagonal directions. A Queen Q_i is capturable if an attacker can move from its tile to that of Q_i in a single turn.

Below, observe (left) a solution to the 4-Queens problem where every move one of the queens can make (green lines) cannot lead to a capture one of the other queens, and (right) constraints violated on that same problem (the Queen at $(2,1)$ can capture both the one at $(0,1)$ and $(3,2)$).

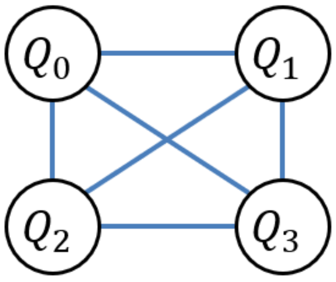


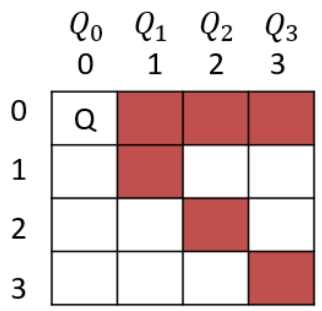
In these problems, we have N Queen variables $Q_0, \dots, Q_{N-1}$ to place with the implicit constraint that none may capture one another (too verbose to list explicitly). Suppose now we structure the *assignment state* of a backtracking solver of the N-queens to be a single array of N integers like $[Q_0, Q_1, Q_2, Q_3]$ to represent the row occupied by queen Q_i where i is the index of the Queen being placed, *as well as* the *column* in which they are placed, as well as their index in the array. For example, the solution state on the left could be represented as $[1,3,0,2]$ and the non-solution on the right as $[1,3,1,2]$.

1.1. Using this representation (i.e., assigning each Queen to a specific row), would constraint propagation *preprocessing* (i.e., before any backtracking and assignment is done) be effective at pruning any domains? Why or why not?

Using our assignment state indexing from the previous page, we can carry this forward into the backtracking algorithm itself by representing Queens that have *not* been assigned yet with a dash (-). For example: $[0,2,-,-]$ denotes a *partial assignment* in which $Q_0 = (c,r) = (0,0)$, $Q_1 = (1,2)$ and Q_2, Q_3 have yet to be assigned.

Now *during* backtracking, suppose we apply *forward checking* to remove any inconsistent values from the domains of adjacent (in the constraint graph), unassigned variables based on the assignments made to variables thus far. This means that an assignment of $Q_0 = 0$, denoted by the partial assignment $[0,-,-,-]$, would have the following domain restrictions by forward checking (with the constraint graph on the left and domain reductions depicted in red):





Assigning $Q_0 = 1 \Rightarrow$
Forward Checking Domain Changes:

$$D_{Q_1} -= \{0,1\}$$
$$D_{Q_2} -= \{0,2\}$$
$$D_{Q_3} -= \{0,3\}$$

1.2. Draw the backtracking recursion tree *with forward checking during backtracking* that would solve the 4-Queens problem using the above partial-assignment-notation for each state. The first three partial-assignment states are done for you, starting with the root being (for any backtracking problem) the *empty assignment* $[-,-,-,-]$. Assume queens are assigned-to in ascending index order and each possible row in their domain also.

$$\begin{tikzcd}
 & & {[-,-,-,-]} \arrow[dl] \\
 & {[0,-,-,-]} \arrow[dl] & \\
{[0,2,-,-]} & &
\end{tikzcd}$$

Problem 2 – Constraint Propagation

Since it's relevant to your upcoming assignment, let's talk about a nice numerical example:

- *Variables:* $V = \{W, X, Y, Z\}$
- *Domains:* $D_i = \{0, 1, 2\} \; \forall \, V_i \in V$
- *Constraints:* $C = \{W > Y, \quad Y > X, \quad Y == Z\}$

2.1. Draw the *constraint graph* associated with the CSP above.

2.2. Perform the AC-3 algorithm for constraint propagation to restrict the domains of the above. The Set of arcs and Domains have been *initialized* for you. Fill in the table, the set, and cross off values in each domain as you perform the steps of AC-3. *You may not need the entire table for your answer, but you will not need more than it!*

Arc Set:

$$(W \to Y), (Y \to W), (X \to Y), (Y \to X), (Z \to Y), (Y \to Z)$$

D_W	0	1	2

D_X	0	1	2

D_Y	0	1	2

D_Z	0	1	2

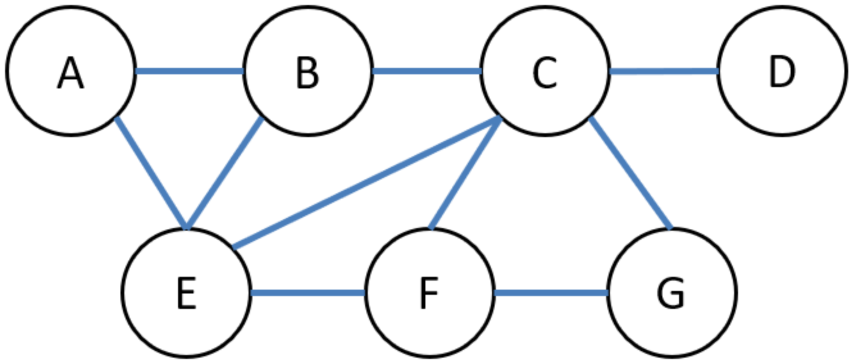
Arc	Constraint	Consistent?	Domain Change	Arcs Added

Problem 3 – Tree-Structured CSPs & Cutset Conditioning

Let's get back to basics with some good old Map Coloring. As a refresher, in the Map Coloring problem, the goal is to assign one of k colors to variables in some geographically connected setting such that no two adjacent variables share the same color.

Suppose we have the following constraint graph associated with a Map Coloring problem in which edges denote adjacency between variables, and we have $k = 3$ colors to work with:

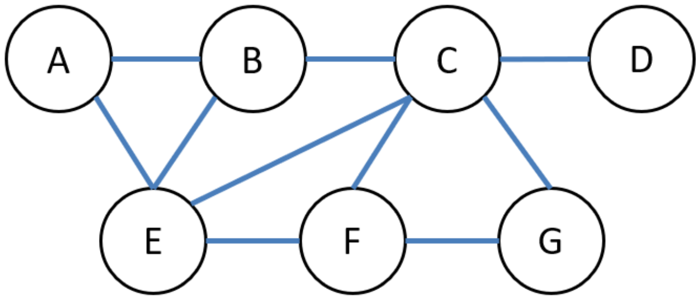
$$D_i = \{red, green, blue\}$$



3.1. Suppose, during backtracking, we begin by assigning $\{E = red, F = blue\}$ and are performing *forward checking*. Which variable would be wise to assign to next, and why?

3.2. Suppose, during backtracking with forward checking, we begin by assigning $\{A = green, F = blue\}$ and have decided to assign to B next. At this point, the domain of B will have been reduced to $D_B = \{blue, red\}$ from forward checking; which of these values would we be wise to try assigning to B next, and why?

Repeating the Constraint Graph from the previous page for convenience:



3.3. Find a *minimal* cutset such that the remaining variables are tree-structured.

3.4. *Condition* on your cutset and adjust the remaining tree's domains accordingly:

Cutset Assignment:
Other Variable Domain Changes:

3.5. Perform the *Directed Arc Consistency Tree CSP Solver* like done in class in the table here:

Index	**0**	**1**	**2**	**3**	**4**
Tree Struct.					
Domains					
Arc Cons.					
Assign					