Problem 1 Constraint Satisfaction Problem

In this problem we consider the formulation of the *confused n-queen* problem as a CSP and algorithms for solving it. The confused n-queen problem is to place n queens, one on each row of a $n \times n$ chessboard, in such a way that every pair of queens attacks each other. The solutions of the confused n-queen problems are rather simple. For $n > 3$, there are $n + 2$ solutions: one for each of the n columns and the two principal diagonals. For example, the confused 4-queen problem has 6 solutions. Two of the 6 solutions are shown below:

		Q	
		Q	
		Q	
		Q	

			Q
		Q	
	Q		
Q			

Figure 1: Two solutions to the confused 4-queen problem.

To cast the problem into a CSP, we choose as variables $\{Q_i \mid i = 1 \text{ to } n\}$, and values domain$(Q_i) = \{1, 2, \ldots, n\}$ for each i. The variable Q_i represents the position of the queen on row i. For example, $Q_1 = 3$ represents a queen in the (3,1) position of the chessboard, i.e., row 1 and column 3. We represent the condition that any pair of queens attacks each other by a binary constraint `consistent?`. The `consistent?` constraint takes two queen positions (i,j) and (m,n) and returns `true` if they attack each other, and `false` otherwise.

We can conveniently represent the confused $n-$queen problem graphically. For example, the graph of the confused 4-queen looks like:

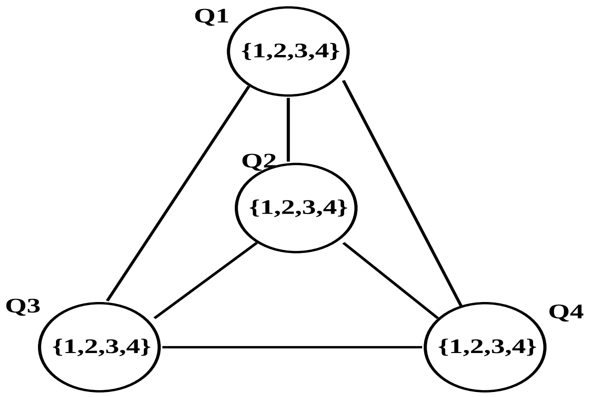

Figure 2: CSP formulation of the confused 4-queen problem.

Each edge connecting Q_i and Q_j in the graph represents two constraints: one directed from Q_i to Q_j and the other from Q_j to Q_i.

Part A

Define the constraint `consistent?` by a mathematical expression involving i, j, m, and n:

Part B

Let us consider solving a confused 4-queen problem with the following initial configuration:

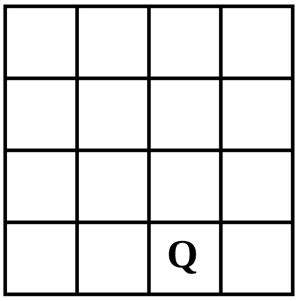

Figure 3: Initial configuration: A queen on row 1 and column 3.

Suppose we run the `forward-check` procedure (as in problem set 5) on the variable Q_1. In the following graph cross out the inconsistent values detected by the procedure after the procedure terminates.

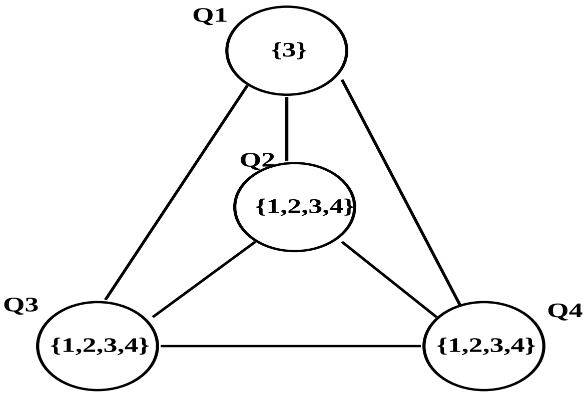

Figure 4: Initial configuration: A queen on row 1 and column 3.

Part C

Suppose we embed `forward-check` in the `backtrack` procedure (as in problem set 5) so that `forward-check` is called after each instantiation of a variable. Assume the following conditions hold:

- `backtrack` stops when it either finds a solution or exhausts all domain values.
- `backtrack` does not instantiate variables that have only one value left in their domains.
- Variables are chosen to instantiate in the order $\{Q_1, Q_2, Q_3, Q_4\}$.
- Values in a domain are instantiated in the order $\{1, 2, 3, 4\}$.

Run the combined `backtrack` and `forward-check` procedure on the graph you obtained in part A (i.e., after an initial `forward-check` has applied to Q_1). Show the state of the graph when the combined procedure terminates.

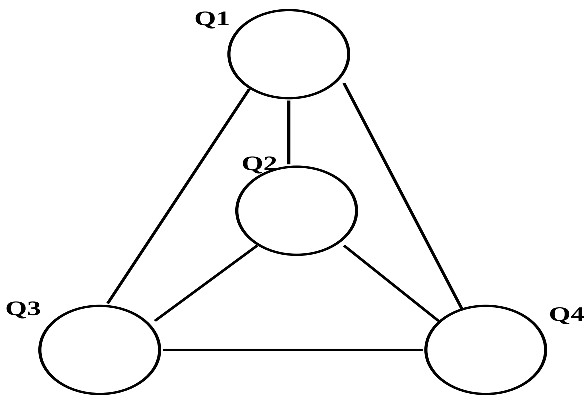

Figure 5: Graph of Part B

How many times is `backtrack` called to instantiate a variable? $\boxed{}$

Part D

Let us repeat Part A using `AC3` instead of `forward-check`. Go back to the initial configuration with $Q_1 = 3$. Suppose we run the `AC3` procedure (as in problem set 5) on it. In the following graph cross out the inconsistent values detected by the procedure after the procedure terminates.

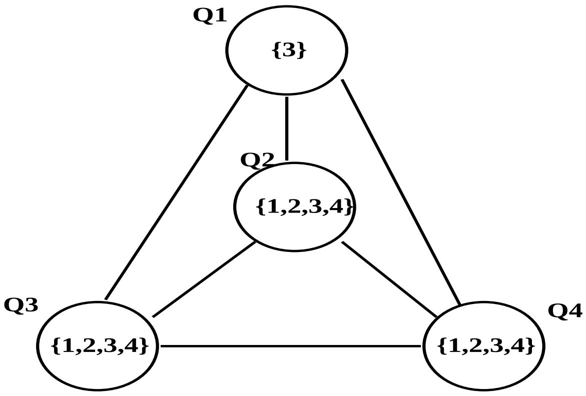

Figure 6: Initial configuration: A queen on row 1 and column 3.

Part E

Let us repeat Part B embedding `AC3` instead of `forward-check` in the `backtrack` procedure.

Run the combined `backtrack` and `AC3` procedure on the graph you obtained in part C (i.e., after an initial `AC3` has applied to Q_1). Show the state of the graph when the combined procedure terminates.

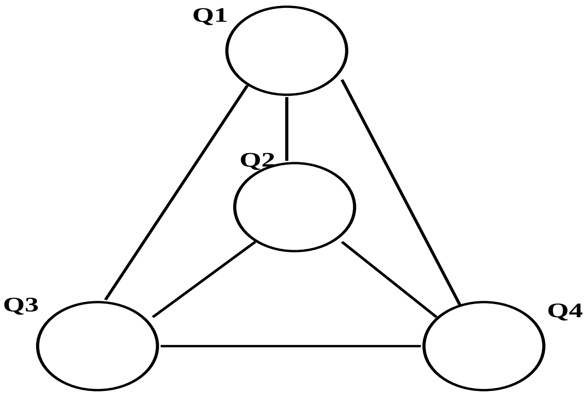

Figure 7: Graph of Part D

How many times is `backtrack` called to instantiate a variable? $\boxed{}$