1 Basics

To start, we'll recall some basic ideas about CSPs.

1. Consider the following: you're a student with one hour blocks to get homework done throughout a day. You have multiple pieces of homework that each take a different amount of uninterruptible time. How can we model this as a CSP? Cover the **a) variables**, **b) their domains**, and **c) the constraints** that exist between the variables.

2. In this sub-problem consider CSPs in general, that is, not related to any specific problem. To how many variables do each of the following constraints apply?

   (a) Unary

   (b) Binary

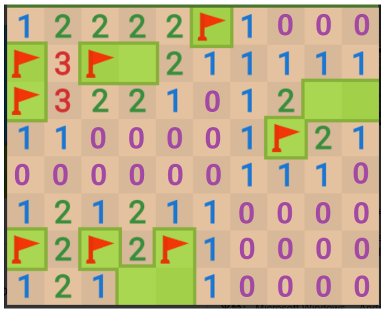
   (c) Ternary

   (d) n-ary

2 Complex

In this problem, we will try to solve a classical puzzle game, *Minesweeper*.
We denote grid squares using (x, y) notation, where **x is the row number** and **y is the column number**. $(1, 1)$ would be the top-left grid square.
We denote grid square (a, b) as **adjacent** to grid square (x, y) if $(a, b) \neq (x, y)$, $|a - x| \leq 1$, and $|b - y| \leq 1$. So $(1, 1)$ has 3 adjacent grid squares, and grid squares that are not on the corners or edges have 8 adjacent grid squares.
You are presented an N by M table where each grid square can be contain a mine or it is empty. The grid squares with a digit are explored, and other grid squares are unexplored. Explored grid squares cannot be a mine (or you would have exploded), and the digit shows the total number of mines in its adjacent grid squares.

The player can **mark** a mine by placing a flag on an unexplored grid square. The final goal is to correctly mark all the mines. Below is an example game with some mines already marked.



1	2	2	2	2	🚩	1	0	0	0
🚩	3	🚩	🟩	2	1	1	1	1	1
🚩	3	2	2	1	0	1	2	🟩	🟩
1	1	0	0	0	0	1	🚩	2	1
0	0	0	0	0	0	1	1	1	0
1	2	1	2	1	1	0	0	0	0
🚩	2	🚩	2	🚩	1	0	0	0	0
1	2	1	🟩	🟩	1	0	0	0	0

1. We can formulate the game as a CSP as follows:

   - Each *unexplored grid square* is a variable of domain 2 — either a mine or empty.
   - Each *digit* is a constraint indicating the total number of mines among its adjacent grid squares.

   Consider the CSPs corresponding to the following two games, where grid squares without a digit are unexplored. Among all the constraints (digits), how many are unary, binary and ternary?

(a)

1	2	2
1		
1	2	2

Unary:	
Binary:	
Ternary:	

(b)

	1	1
1		
1	1	1

Unary:	
Binary:	
Ternary:	

2. We run the following **filtering process** to eliminate invalid solutions for a game.

   (a) For each constraint, if there exists a variable that has exactly one value satisfying the constraint, we assign the variable that value.

   (b) If any constraint is violated, return "Constraint violated" and terminate.

   (c) Repeats step 1 and 2 until either all the variables are assigned a value, in which case we have found a solution, or we can no longer uniquely determine the value of any variable, in which case "Game unsolved" is returned.

   Select (A) if the filtering process finds a solution (i.e., assign a value to each variable) , and select (B) otherwise. To help you understand the process, the answer to game (1) as well as the explanation are given.

1	2	2
1		
1	2	2

[✓ (A)]
○ (B)

Explanation: Consider $\boxed{1}$ at grid square $(2, 1)$. For variable $(2, 2)$, there is only 1 value ("mine") that satisfies the constraint, so we assign "mine" to variable $(2, 2)$. No constraints are violated, so we repeat step 1, considering $\boxed{2}$ at grid square $(1, 2)$. For variable $(2, 3)$, there is only 1 value ("mine") that satisfies the constraint, so we assign "mine" to variable $(2, 3)$. No constraints are violated. We've assigned a value to each variable, hence (A) is selected.

(a)

	2	1
1		
	1	1

○ (A)
○ (B)

(b)

	1	1
1		
1	1	1

○ (A)
○ (B)

(c) Because we cannot always use filtering to find a solution, we must use backtracking search. Below is an example of a game that cannot be solved by only running filtering.

	1	1
1		
	1	1

We run backtracking search by considering one variable at a time and applying filtering after each assignment. Suppose we are now considering (1, 1). Note (1,1) is the top left square in the grid.

   i. If we assign value "mine" to grid square (1, 1) and then run filtering, what can we conclude?
   ○ (A) We will find a solution ○ (B) Constraint violated ○ (C) Game unsolved

   ii. If we assign value "empty" to grid square (1, 1) and then run filtering, what can we conclude?
   ○ (A) We will find a solution ○ (B) Constraint violated ○ (C) Game unsolved

(d) Is there a unique solution if we run backtracking search on the third game (2.c) above? If so, select A and fill in the grid squares of mines while leaving others empty. Otherwise select "no solution" or "multiple solutions."

○ (A) Unique solution
○ (B) No solutions
○ (C) Multiple solutions

D ☐	1	1
1	E ☐	F ☐
G ☐	1	1

3. For the rest of this question, we are running the arc consistency algorithm on a different generic CSP. You finish backtracking on a huge 18 by 8 grid and found that 20 grid squares are empty! You come to understand that each empty grid square can have an integer number of coins and that each square is related to some of the other squares. As a result, you set up a generic CSP with 20 variables, where X_i is the number of coins in empty square i. In particular, you find that X_1 is involved in constraints with 6 other variables. X_2 is involved in constraints with 9 other variables.
While running the arc consistency algorithm we reach a point when all variables have 4 values left in their domains, and we have one last arc in the queue: $X_1 \longrightarrow X_2$.

   (a) Now we are processing the arc $X_1 \longrightarrow X_2$. We are able to remove a value from the domain of a variable, and add the necessary arcs into the queue.
   How many arcs are in the queue now?
   ○ (A) 3 ○ (B) 4 ○ (C) 5 ○ (D) 6 ○ (E) 9 ○ (F) 18
   ○ (G) None of these ○ (H) We cannot determine the exact number of arcs

   (b) Following the previous part, we processed any arcs that may have been added to the queue. No more values were removed from any variable. As a result, we plan to assign a value to one of the variables to continue with backtracking search. Pick the statement below that is most valid.
   ○ (A) We should assign a value to X_1 because it has the Least Constraining Value
   ○ (B) We should assign a value to X_1 because it has the Minimum Remaining Values
   ○ (C) We should assign a value to X_2 because it has the Least Constraining Value
   ○ (D) We should assign a value to X_2 because it has the Minimum Remaining Values
   ○ (E) We should assign a value to some X_i ($i \geq 3$) that has the Least Constraining Value
   ○ (F) We should assign a value to some X_i ($i \geq 3$) that has the Minimum Remaining Values

3 CSPs: Domains and Arc Consistency

1. Consider a simple two-variable CSP, with variables A and B, both having domains $\{1, 2, 3, 4\}$. There is only one constraint, which is that A and B sum to 6 ($A + B = 6$). In the table below, circle the the values that remain in the domains after enforcing arc consistency.

Variable	Values remaining in domain
A	{ 1 , 2 , 3 , 4 }
B	{ 1 , 2 , 3 , 4 }

Now consider a general CSP C, with variables X_i having domains D_i. Assume that all constraints are between pairs of variables. C is not necessarily arc consistent, so we enforce arc consistency, by running AC-3 (the arc consistency algorithm from class). For each variable X_i, let D_i^{AC} be its resulting **arc consistent** domain.

2. Circle all of the following statements that are *guaranteed* to be true about the domains D_i^{AC} compared to D_i.

   (i) For all i, $D_i^{AC} \subseteq D_i$.

   (ii) If C is initially arc consistent, then for all i, $D_i^{AC} = D_i$.

   (iii) If C is not initially arc consistent, then for at least one i, $D_i^{AC} \neq D_i$.

3. Let n be the number of solutions to C and let n^{AC} be the number of solutions existing after arc consistency has been enforced. Circle all of the following statements that *could* be true, if any.

   (i) $n^{AC} = 0$ but no D_i^{AC} are empty.

   (ii) $n^{AC} \geq 1$ but some D_i^{AC} is empty.

   (iii) $n^{AC} > 1$

   (iv) $n^{AC} < n$

   (v) $n^{AC} = n$

   (vi) $n^{AC} > n$