Constraint Satisfaction Problems

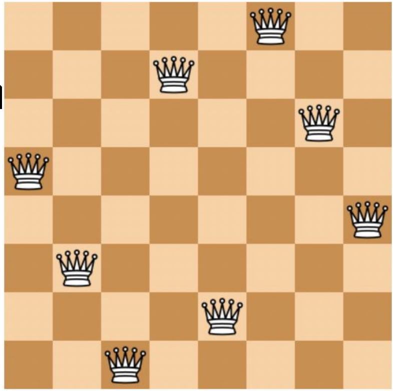
Kushal Shah @ Sitare

	1	2	3	4	5	6	7	8	9
A			3		2		6		
B	9			3		5			1
C			1	8		6	4		
D			8	1		2	9		
E	7								8
F			6	7		8	2		
G			2	6		9	5		
H	8			2		3			9
I			5		1		3		

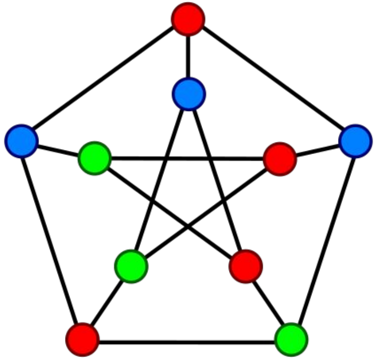
Constraint Satisfaction Problems

- Mathematical questions defined as a set of objects whose state must satisfy a certain number of constraints or inequalities

- A problem is solved when each variable h
that satisfies all the constraints on the variable



- Some famous CSPs:
  - Sudoku and many other logical puzzles
  - Graph Coloring
  - Boolean Satisfiability Problem (SAT)
    - (a AND NOT b) vs (a AND NOT a)
    - First problem proven to be NP-complete
  - 8-Queens Puzzle



6.1 DEFINING CONSTRAINT SATISFACTION PROBLEMS

A constraint satisfaction problem consists of three components, X, D, and C:

X is a set of variables, $\{X_1, \ldots, X_n\}$.

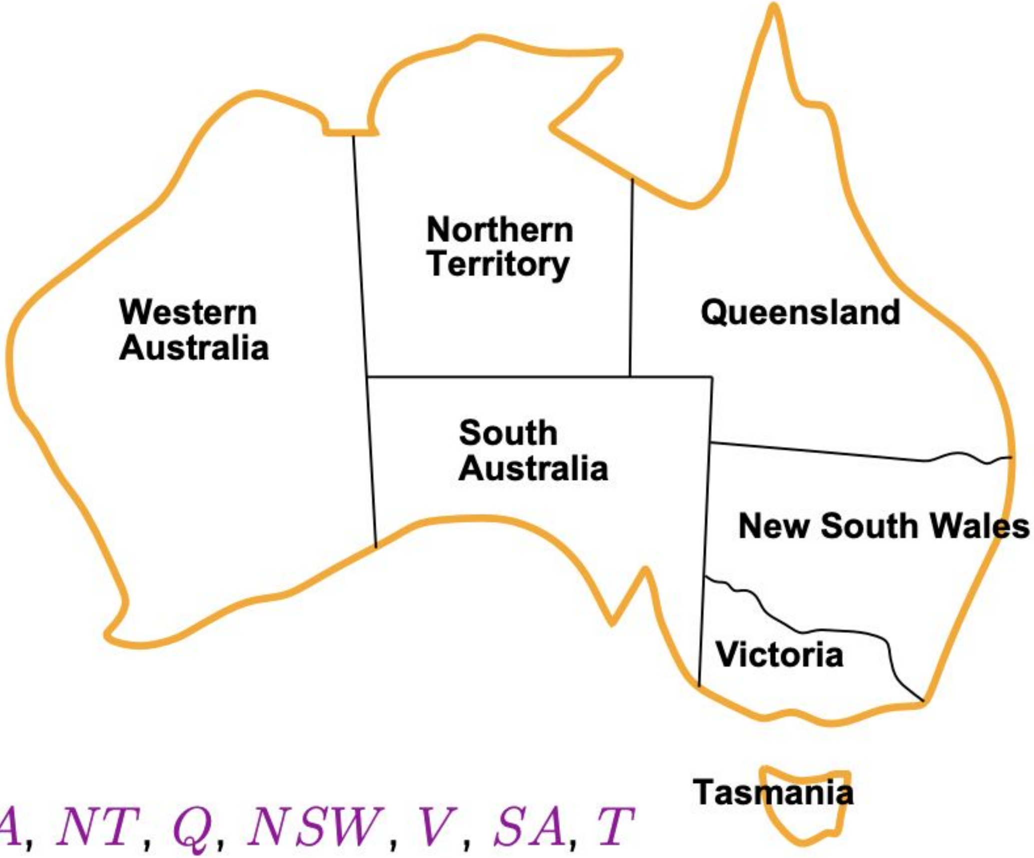
D is a set of domains, $\{D_1, \ldots, D_n\}$, one for each variable.

C is a set of constraints that specify allowable combinations of values.

Each domain D_i consists of a set of allowable values, $\{v_1, \ldots, v_k\}$ for variable X_i.

* Non-linear constraints are very hard to solve for!

	1	2	3	4	5	6	7	8	9
A			3		2		6		
B	9			3		5			1
C			1	8		6	4		
D			8	1		2	9		
E	7								8
F			6	7		8	2		
G			2	6		9	5		
H	8			2		3			9
I			5		1		3		

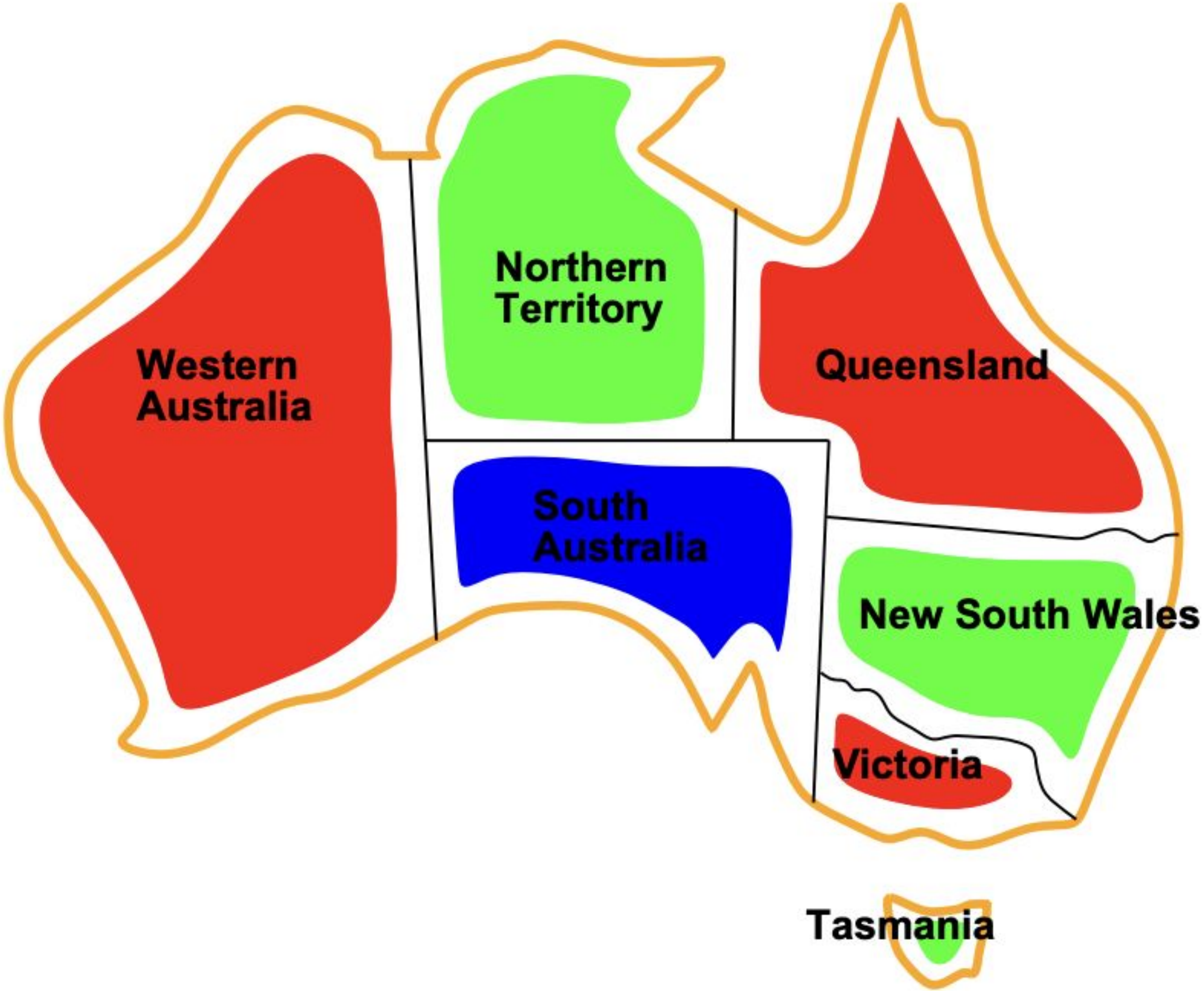


Variables WA, NT, Q, NSW, V, SA, T

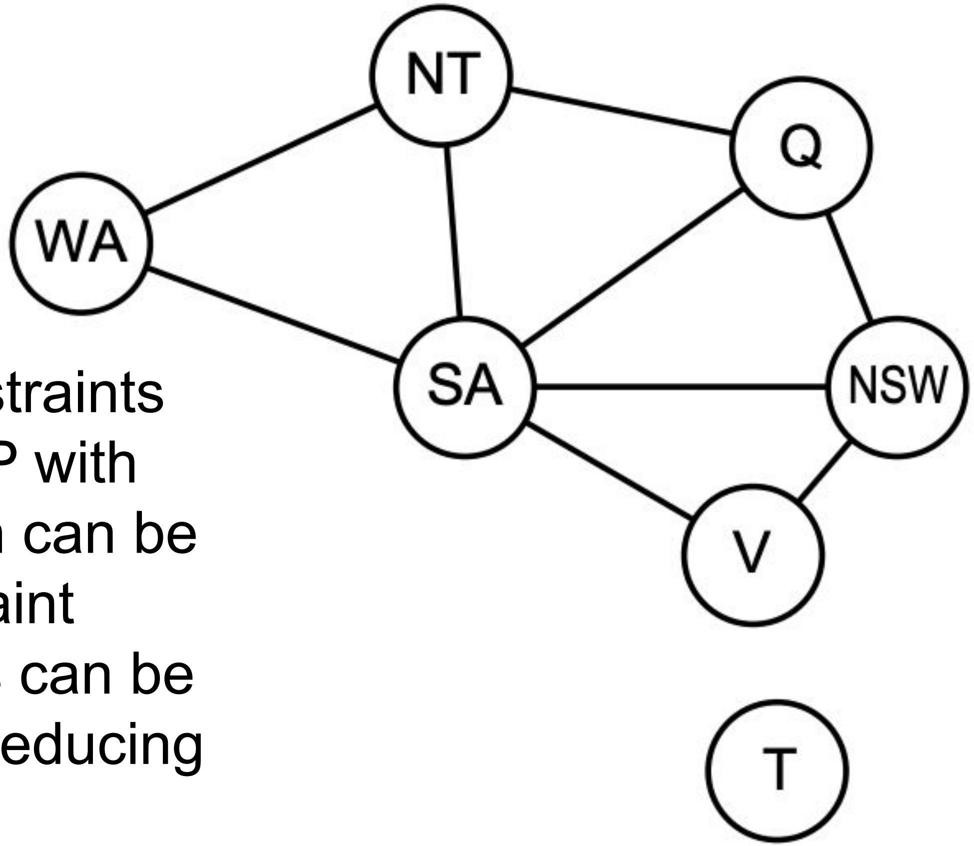
Domains $D_i = \{red, green, blue\}$

Constraints: adjacent regions must have different colors

e.g., $WA \neq NT$ (if the language allows this), or


Northern Territory
Western Australia
Queensland
South Australia
New South Wales
Victoria
Tasmania

Constraint graph: nodes are variables, arcs show constraints



All CSPs with n-ary constraints can be reduced to a CSP with binary constraints, which can be represented by a constraint graph. Unary constraints can be taken care of by simply reducing the domain.

Unary constraints involve a single variable,

e.g., $SA \neq green$

Binary constraints involve pairs of variables,

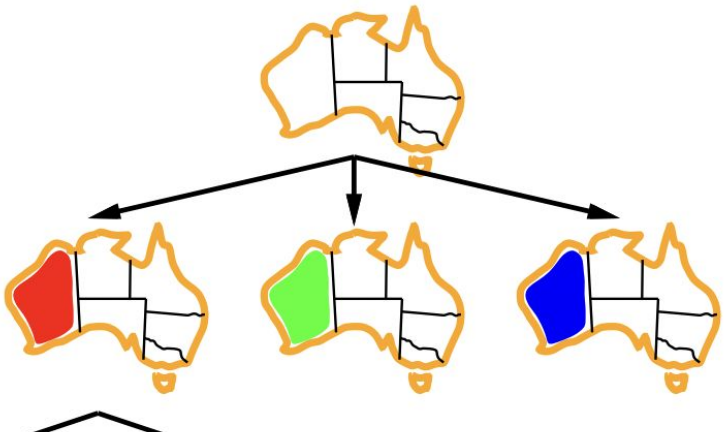
e.g., $SA \neq WA$

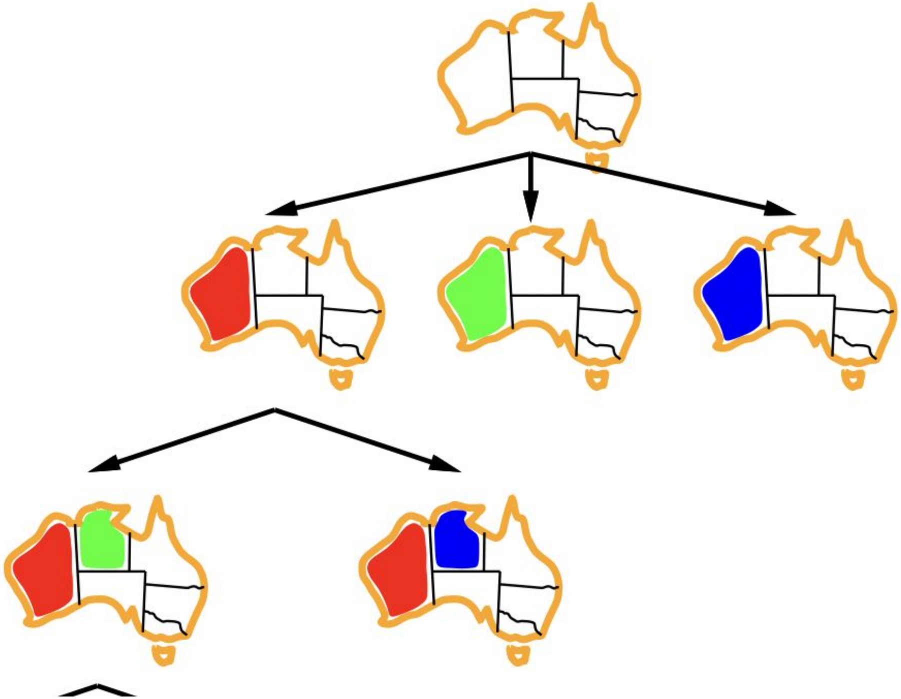
Higher-order constraints involve 3 or more variables,

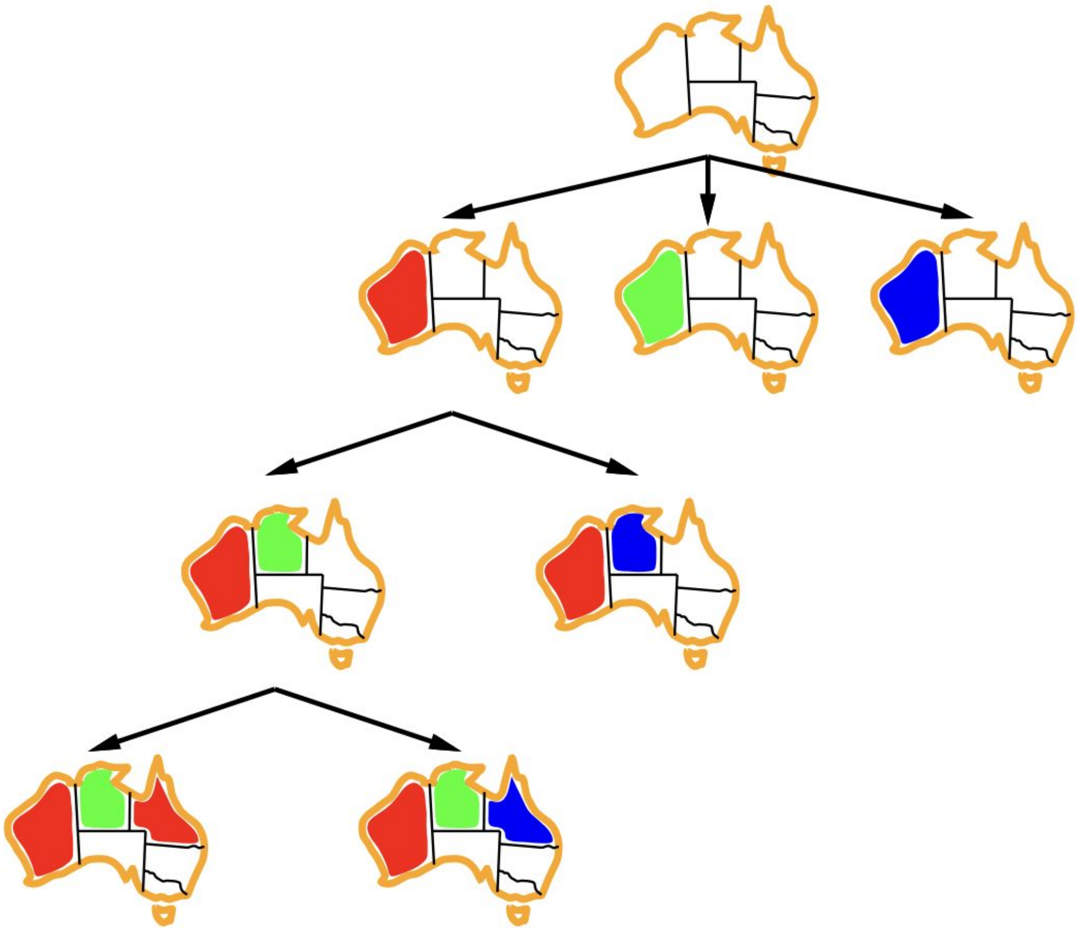
e.g., cryptarithmetic column constraints

Preferences (soft constraints), e.g., red is better than $green$
often representable by a cost for each variable assignment

$\rightarrow$ constrained optimization problems







Backtracking search

Variable assignments are commutative, i.e.,

$[WA = red$ then $NT = green]$ same as $[NT = green$ then $WA = red]$

Only need to consider assignments to a single variable at each node

$\Rightarrow$ $b = d$ and there are d^n leaves

Depth-first search for CSPs with single-variable assignments is called backtracking search

Backtracking search is the basic uninformed algorithm for CSPs

BACKTRACKING SEARCH

Why do we do DFS and not BFS?

What is the branching factor of the search tree?

FORWARD CHECKING (INFERENCE)

	WA	NT	Q	NSW	V	SA	T
Domains	RGB	RGB	RGB	RGB	RGB	RGB	RGB
After WA	R	GB	RGB	RGB	RGB	GB	RGB
After Q	R	B	G	RB	RGB	B	RGB
After V	R	B	G	R	B		RGB

ARC CONSISTENCY

- X_i is arc-consistent with respect to X_j if for every value in the domain D_i there is some value in the domain D_j that satisfies the binary constraint on the arc (X_i, X_j).

- A variable is arc consistent if there is no value in its domain that is ruled out by all of its neighboring constraints.

- A graph is arc-consistent if every variable is arc consistent with every other variable in the graph.

ARC CONSISTENCY

- X_i is arc-consistent with respect to X_j if for every value in the domain D_i there is some value in the domain D_j that satisfies the binary constraint on the arc (X_i, X_j).

- A variable is arc consistent if there is no value in its domain that is ruled out by all of its neighboring constraints.

- A graph is arc-consistent if every variable is arc consistent with every other variable in the graph.

- **Problem :** Consider a problem with two variables with the constraint

$$Y = X^2$$

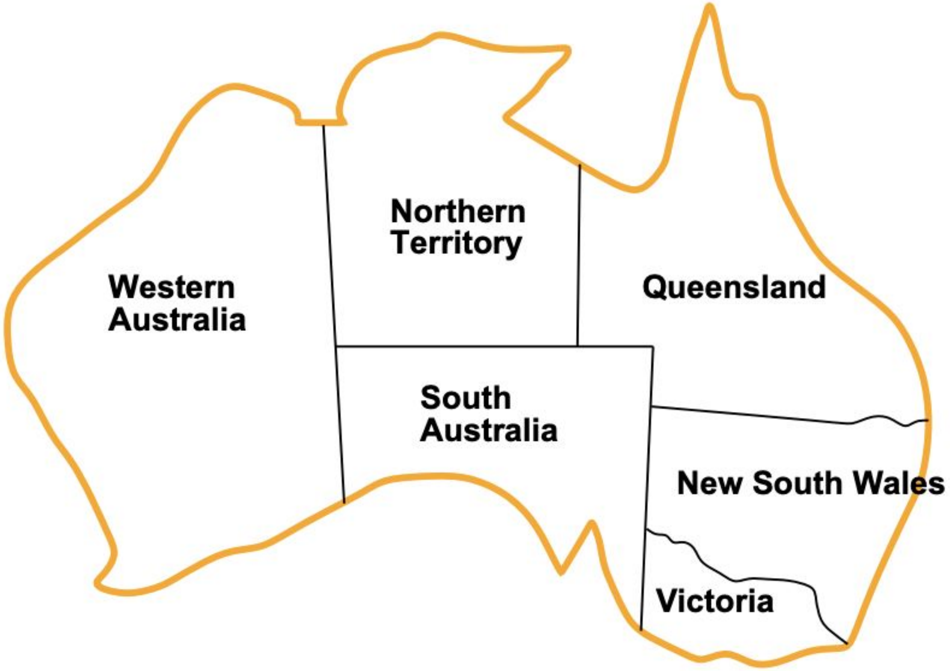
where the domain of X and Y is the set of digits.

Are X and Y arc consistent with respect to each other?

PATH CONSISTENCY

A two-variable set $\{X_i, X_j\}$ is path-consistent with respect to a third variable X_m if, for every assignment $\{X_i = a, X_j = b\}$ consistent with the constraints on $\{X_i, X_j\}$, there is an assignment to X_m that satisfies the constraints on $\{X_i, X_m\}$ and $\{X_m, X_j\}$. This is called path consistency because one can think of it as looking at a path from X_i to X_j with X_m in the middle.

How do we make this arc consistent?

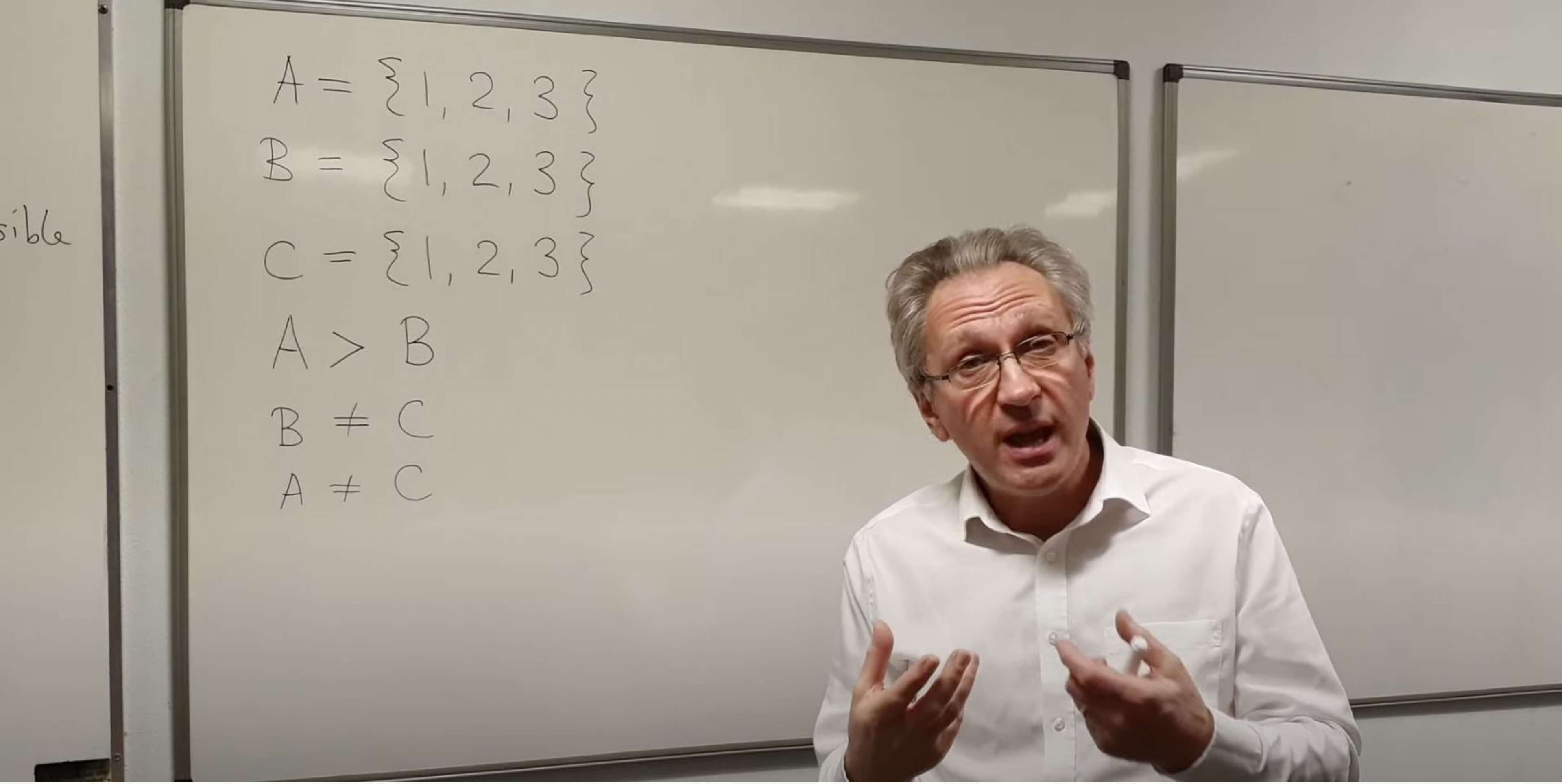


Variables WA, NT, Q, NSW, V, SA, T

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

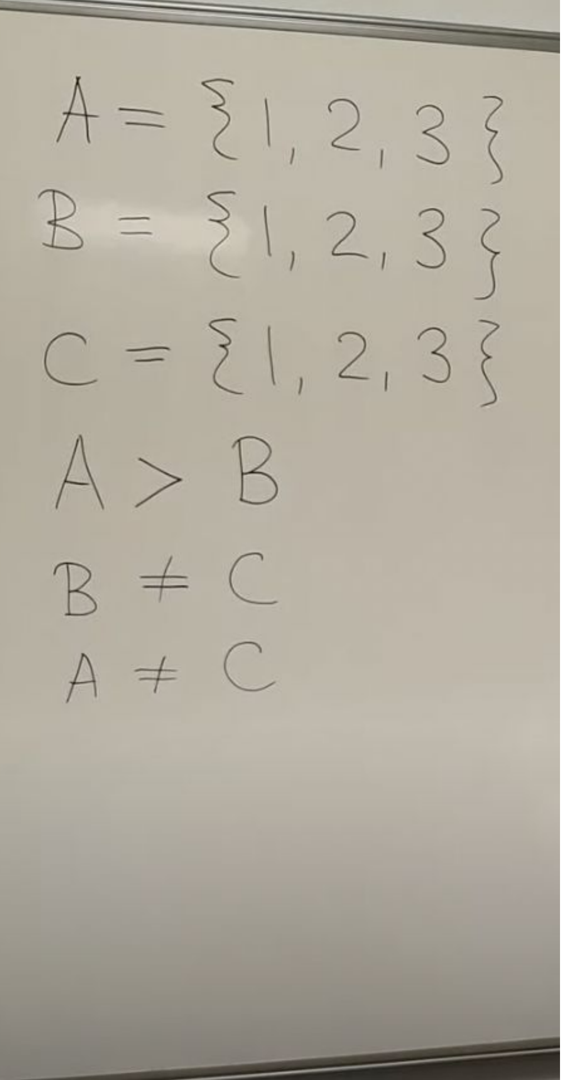
Constraints: adjacent regions must have different colors

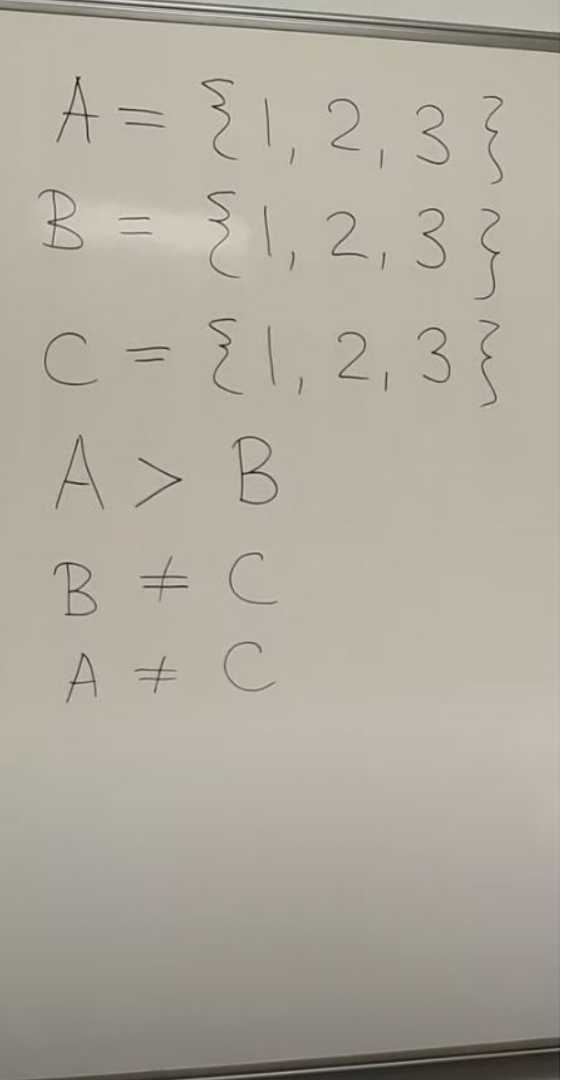
e.g., $WA \neq NT$ (if the language allows this), or



John Levine

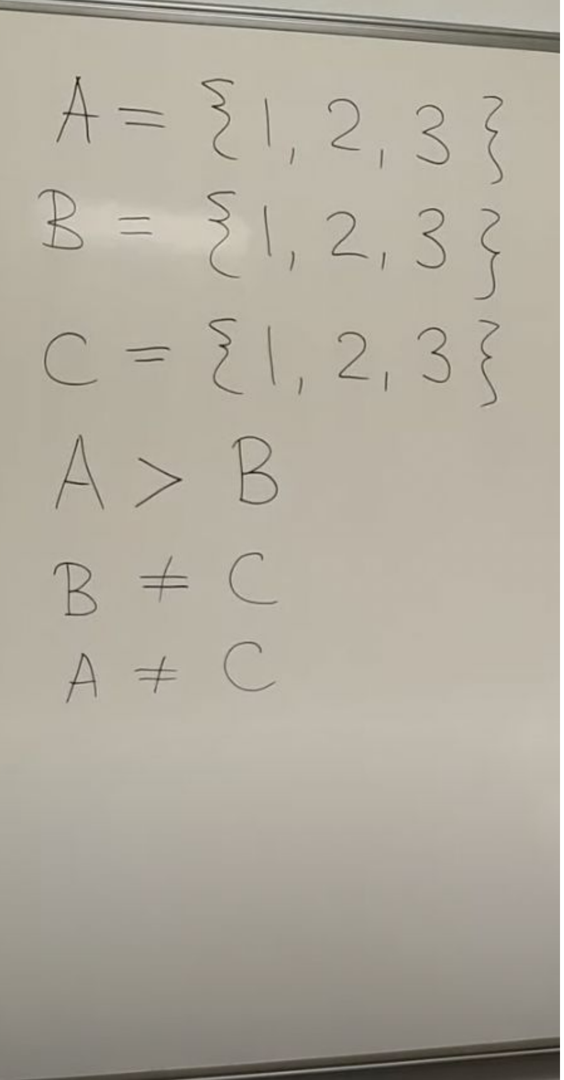
Backtracking Search:





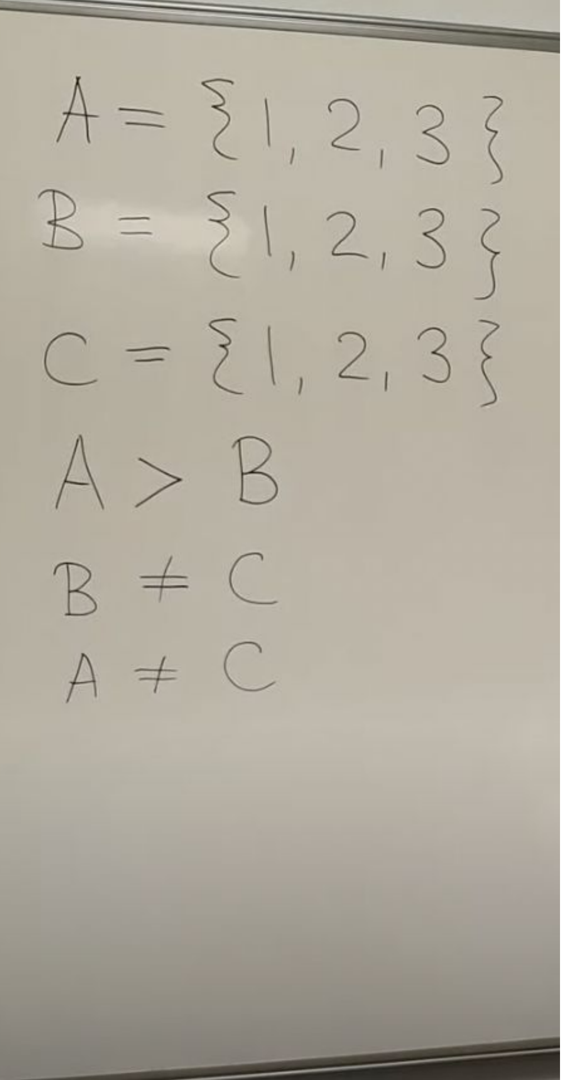
Backtracking Search:

- DFS with single-variable assignments for a CSP



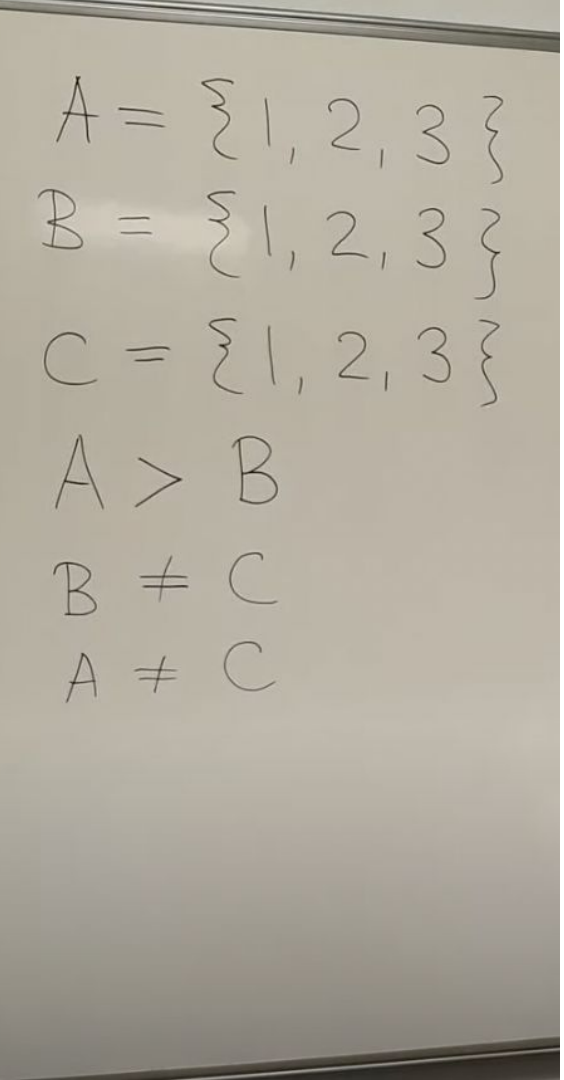
Backtracking Search:

- DFS with single-variable assignments for a CSP
- Basic uninformed search for solving CSPs



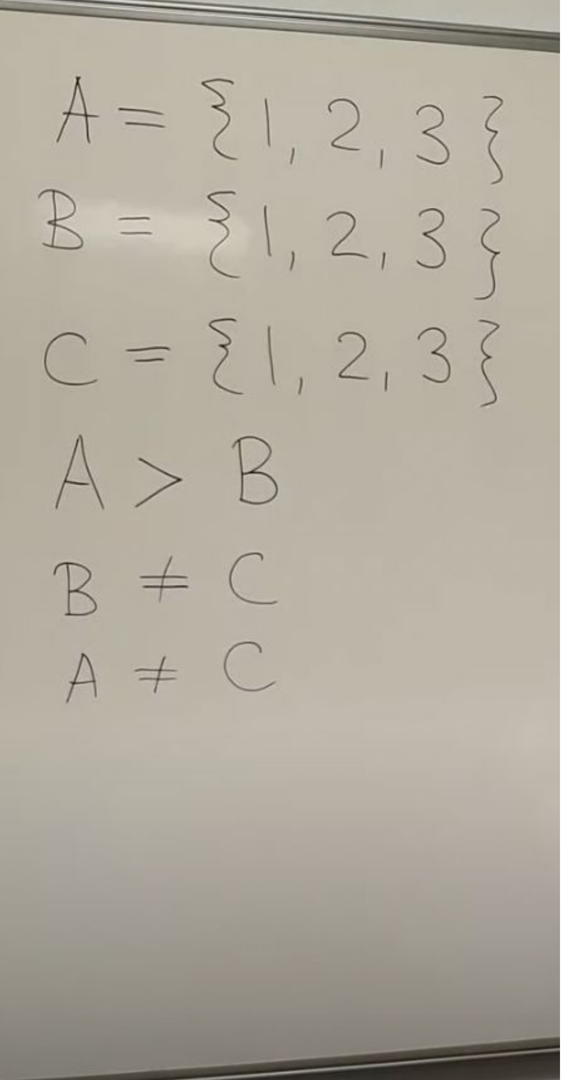
Backtracking Search:

- DFS with single-variable assignments for a CSP
- Basic uninformed search for solving CSPs
- Searches a tree of partial assignments



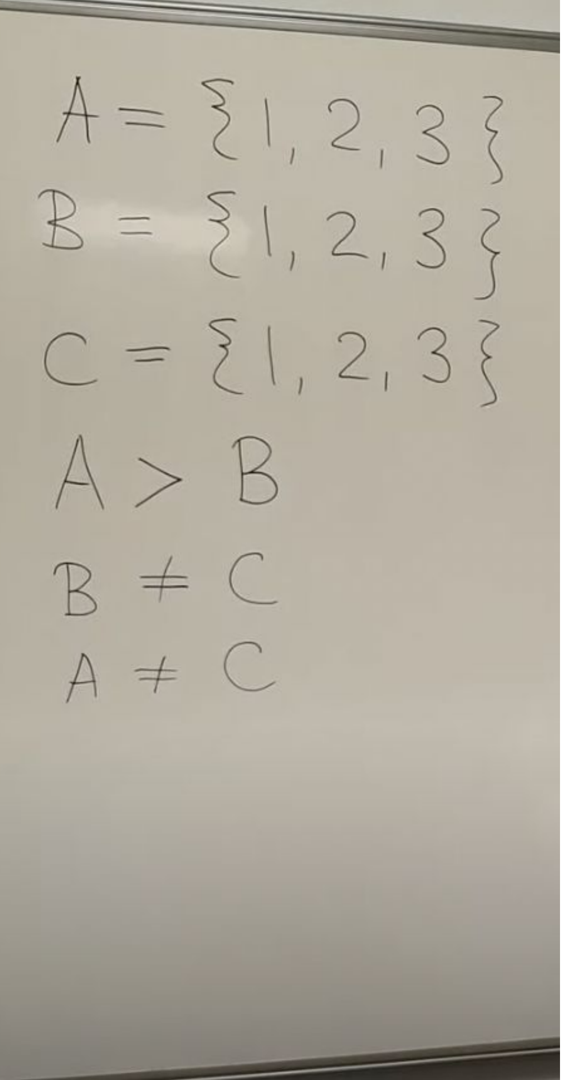
Backtracking Search:

- DFS with single-variable assignments for a CSP
- Basic uninformed search for solving CSPs
- Searches a tree of partial assignments
- Gets rid of unnecessary permutations in search tree



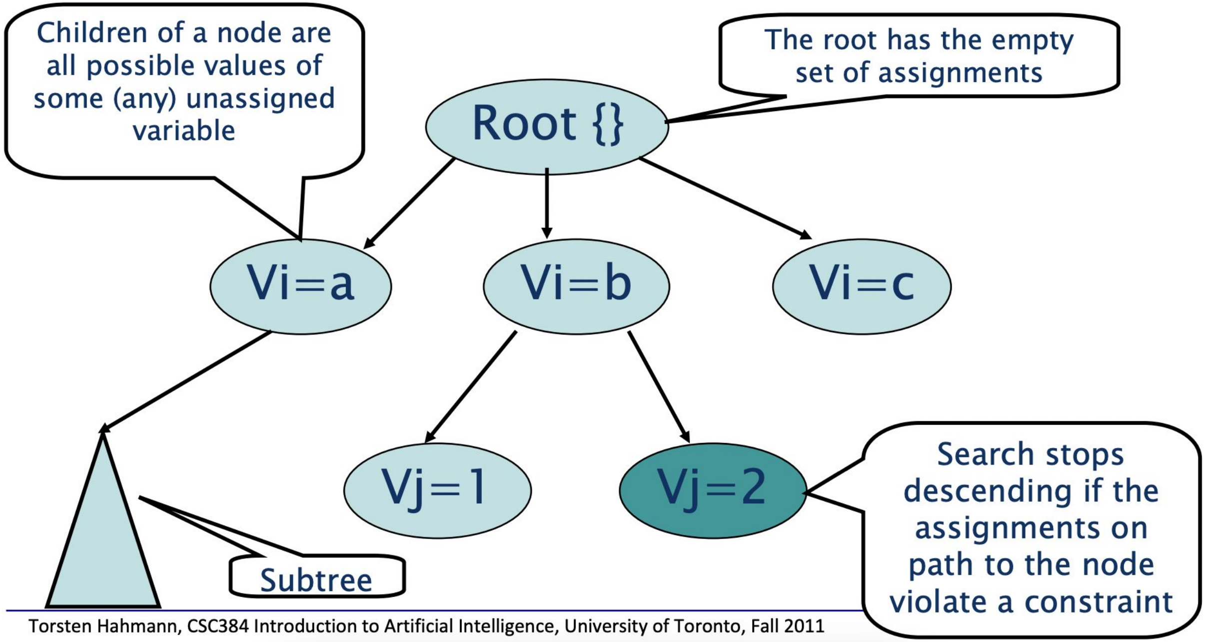
Backtracking Search:

- DFS with single-variable assignments for a CSP
- Basic uninformed search for solving CSPs
- Searches a tree of partial assignments
- Gets rid of unnecessary permutations in search tree
- Significantly reduces search space



Backtracking Search:

- DFS with single-variable assignments for a CSP
- Basic uninformed search for solving CSPs
- Searches a tree of partial assignments
- Gets rid of unnecessary permutations in search tree
- Significantly reduces search space



$A = \{1, 2, 3\}$

$B = \{1, 2, 3\}$

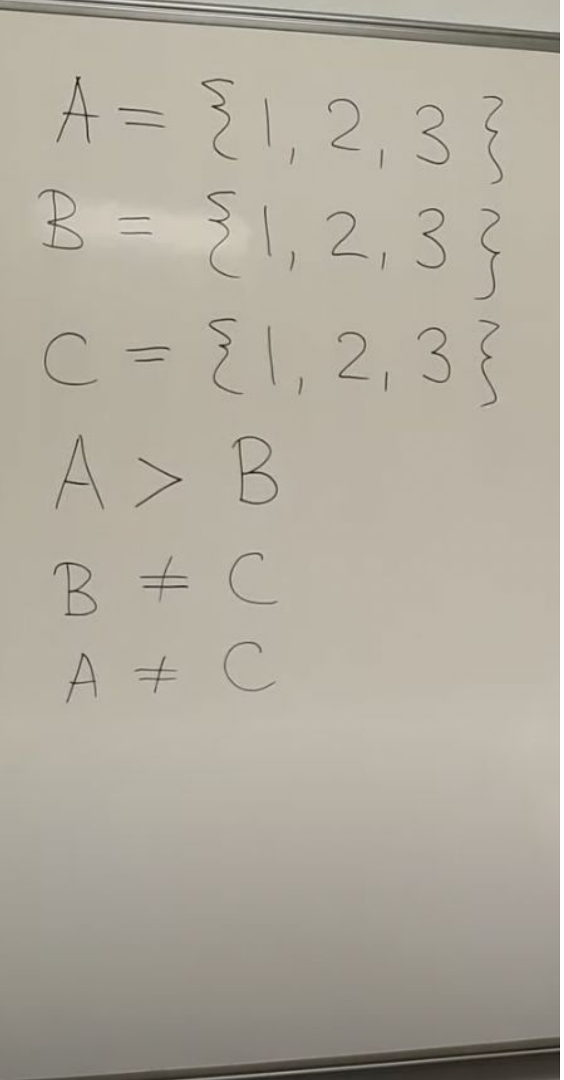
$C = \{1, 2, 3\}$

$A > B$

$B \neq C$

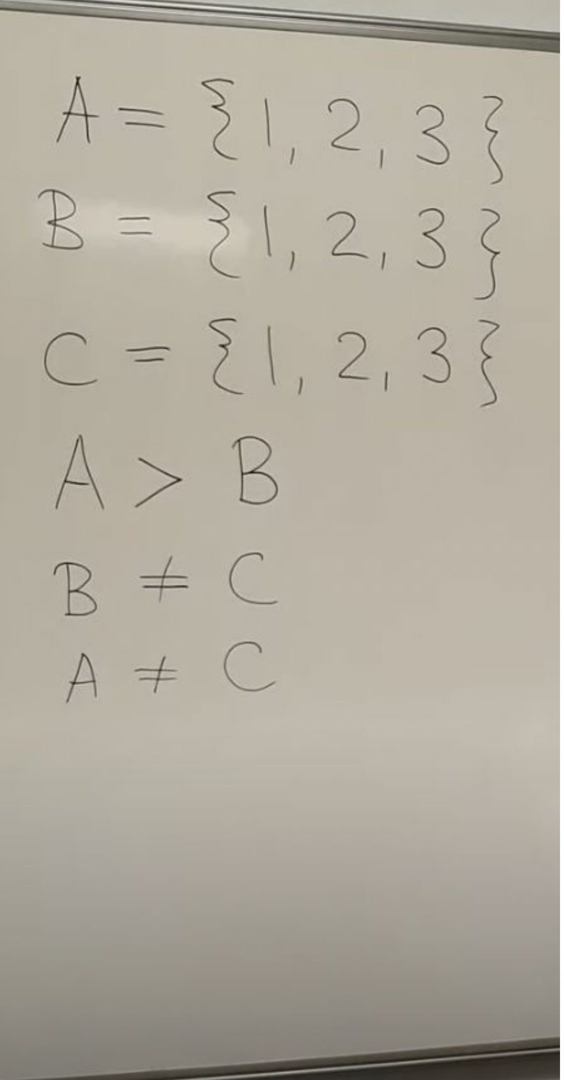
$A \neq C$

Backtracking Search



Backtracking Search

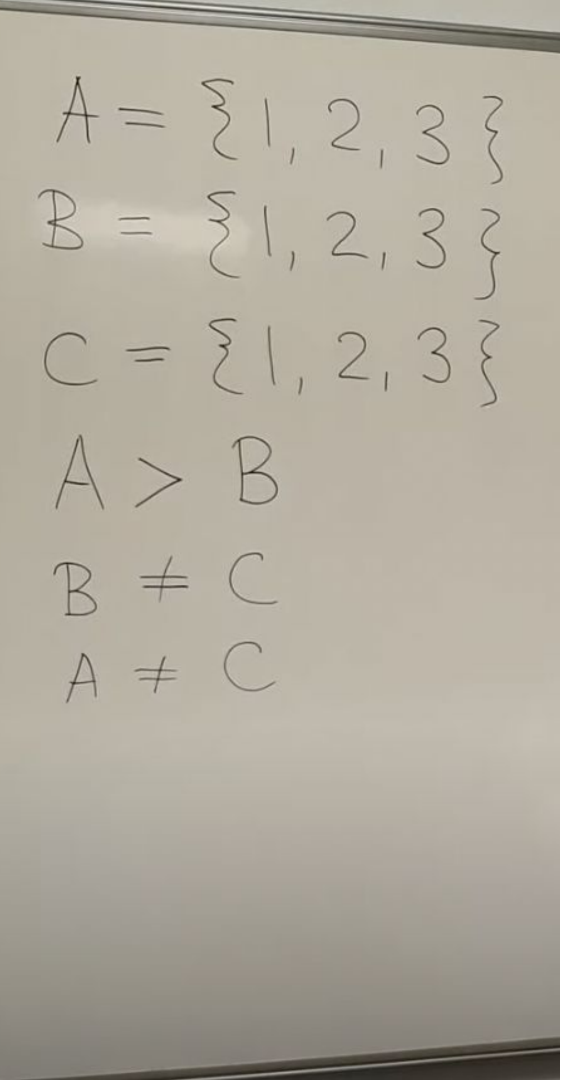
A = 1	FINE, so propagate to next variable



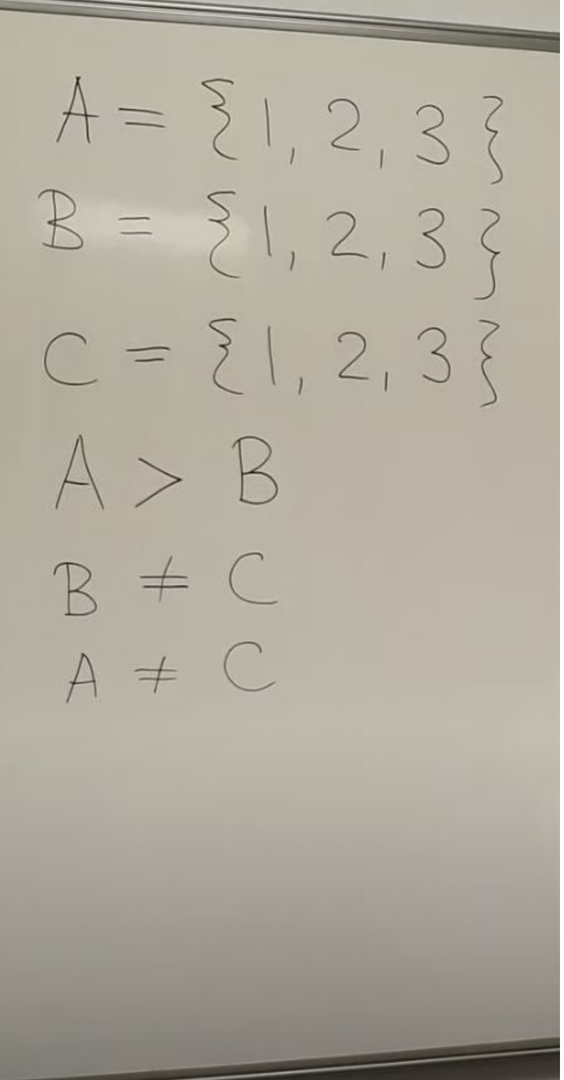
Backtracking Search

A = 1	FINE, so propagate to next variable
A = 1, B = 1	NOT FINE, so change value of B

Backtracking Search

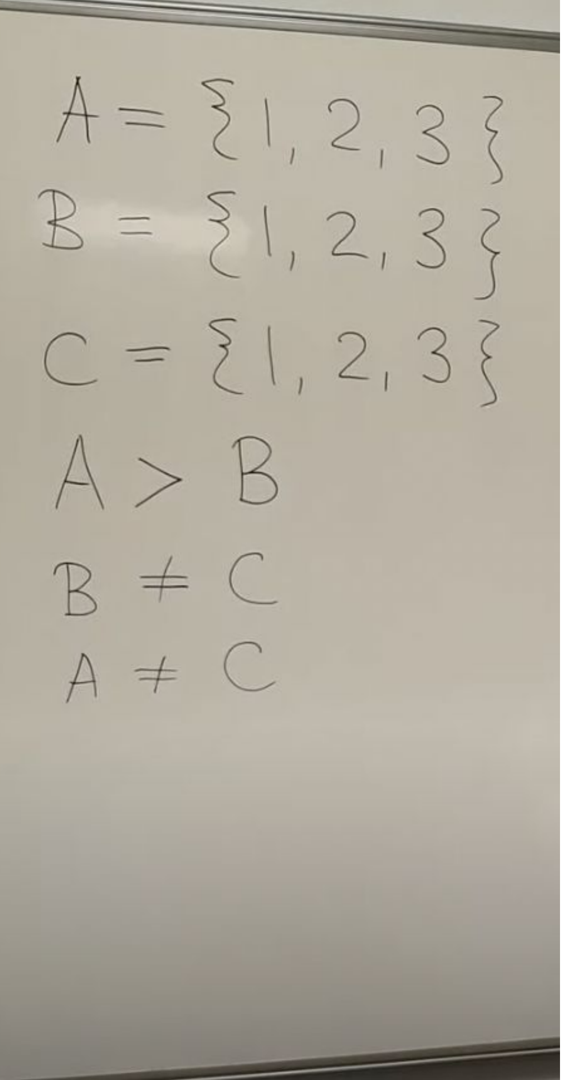


A = 1	FINE, so propagate to next variable
A = 1, B = 1	NOT FINE, so change value of B
A = 1, B = 2	NOT FINE, so change value of B



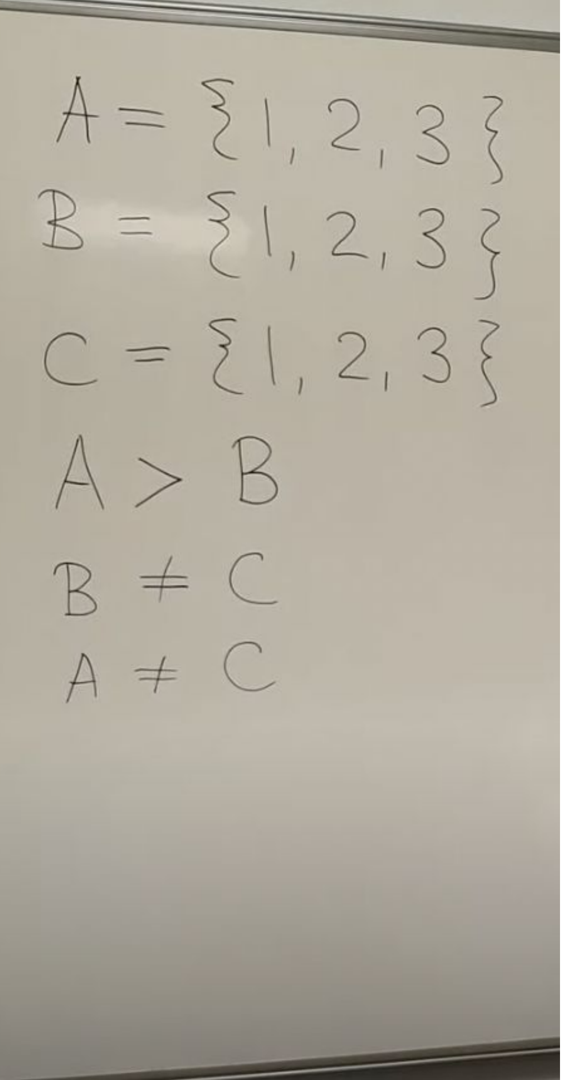
Backtracking Search

A = 1	FINE, so propagate to next variable
A = 1, B = 1	NOT FINE, so change value of B
A = 1, B = 2	NOT FINE, so change value of B
A = 1, B = 3	NOT FINE, so “backtrack” since set B exhausted



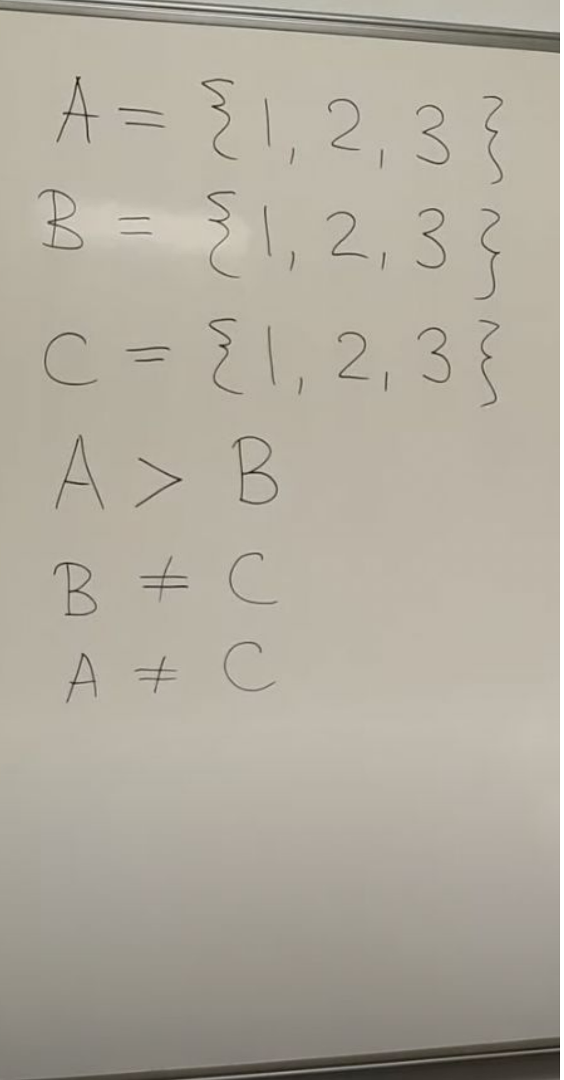
Backtracking Search

A = 1	FINE, so propagate to next variable
A = 1, B = 1	NOT FINE, so change value of B
A = 1, B = 2	NOT FINE, so change value of B
A = 1, B = 3	NOT FINE, so “backtrack” since set B exhausted
A = 2	FINE, so propagate to next variable



Backtracking Search

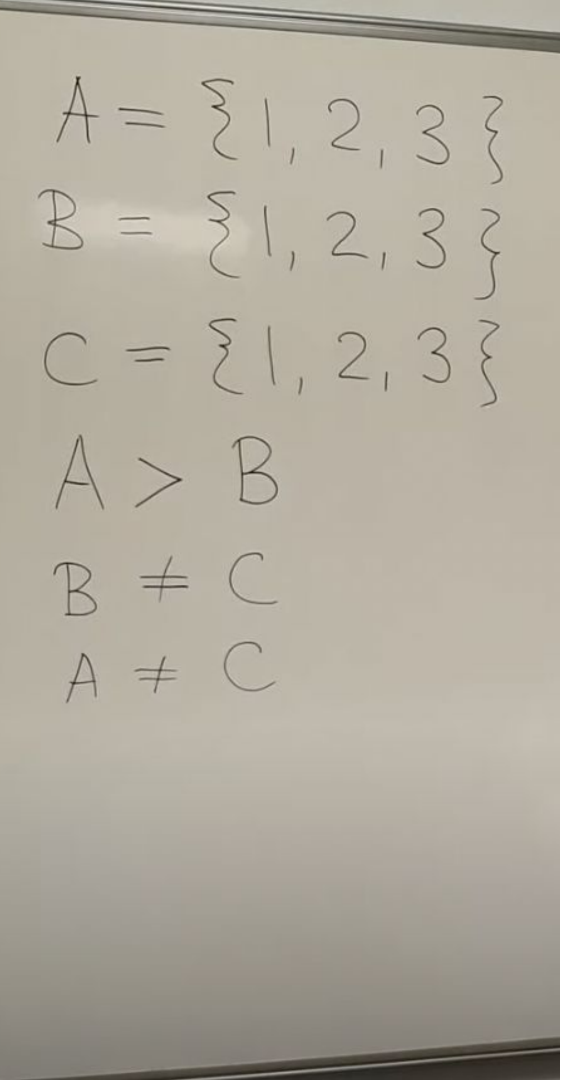
A = 1	FINE, so propagate to next variable
A = 1, B = 1	NOT FINE, so change value of B
A = 1, B = 2	NOT FINE, so change value of B
A = 1, B = 3	NOT FINE, so “backtrack” since set B exhausted
A = 2	FINE, so propagate to next variable
A = 2, B = 1	FINE, so propagate to next variable



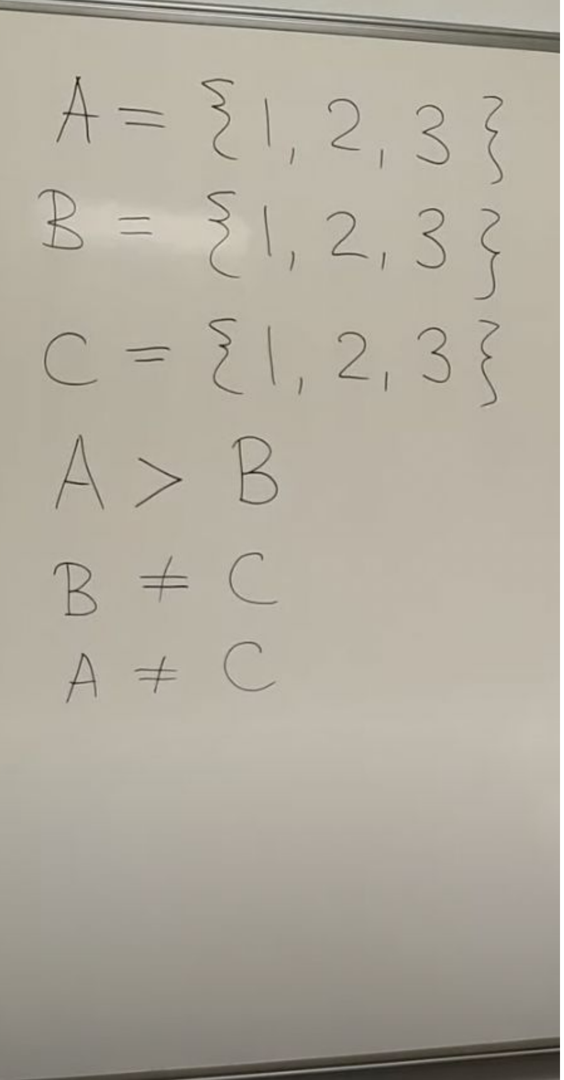
Backtracking Search

A = 1	FINE, so propagate to next variable
A = 1, B = 1	NOT FINE, so change value of B
A = 1, B = 2	NOT FINE, so change value of B
A = 1, B = 3	NOT FINE, so “backtrack” since set B exhausted
A = 2	FINE, so propagate to next variable
A = 2, B = 1	FINE, so propagate to next variable
A = 2, B = 1, C = 1	NOT FINE, so change value of C

Backtracking Search



A = 1	FINE, so propagate to next variable
A = 1, B = 1	NOT FINE, so change value of B
A = 1, B = 2	NOT FINE, so change value of B
A = 1, B = 3	NOT FINE, so “backtrack” since set B exhausted
A = 2	FINE, so propagate to next variable
A = 2, B = 1	FINE, so propagate to next variable
A = 2, B = 1, C = 1	NOT FINE, so change value of C
A = 2, B = 1, C = 2	NOT FINE, so change value of C



Backtracking Search

A = 1	FINE, so propagate to next variable
A = 1, B = 1	NOT FINE, so change value of B
A = 1, B = 2	NOT FINE, so change value of B
A = 1, B = 3	NOT FINE, so “backtrack” since set B exhausted
A = 2	FINE, so propagate to next variable
A = 2, B = 1	FINE, so propagate to next variable
A = 2, B = 1, C = 1	NOT FINE, so change value of C
A = 2, B = 1, C = 2	NOT FINE, so change value of C
A = 2, B = 1, C = 3	FINE, and no more variables to propagate

Arc Consistency Algorithm #3:

$A = \{1, 2, 3\}$

$B = \{1, 2, 3\}$

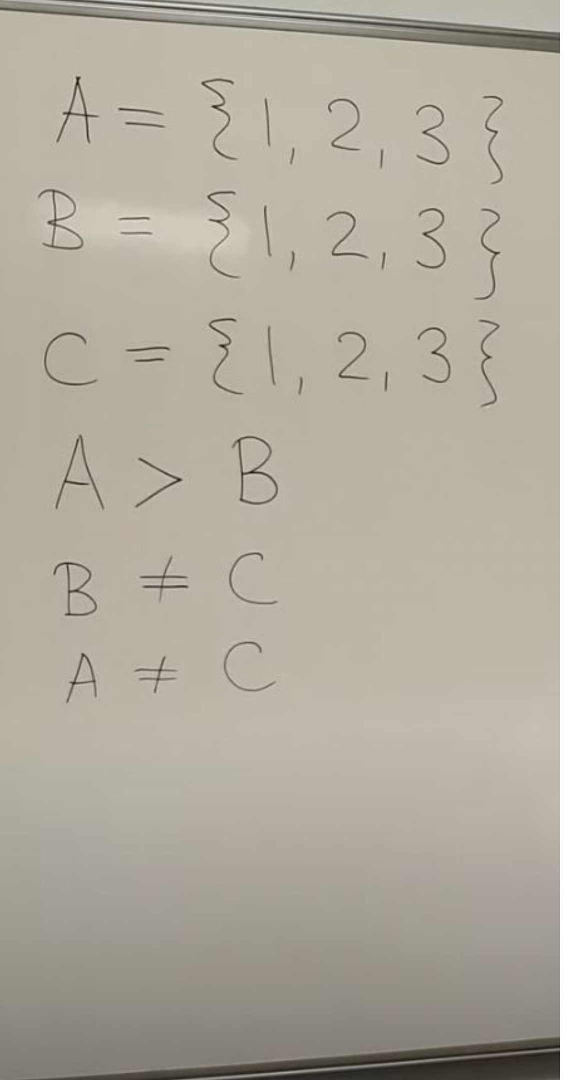
$C = \{1, 2, 3\}$

$A > B$

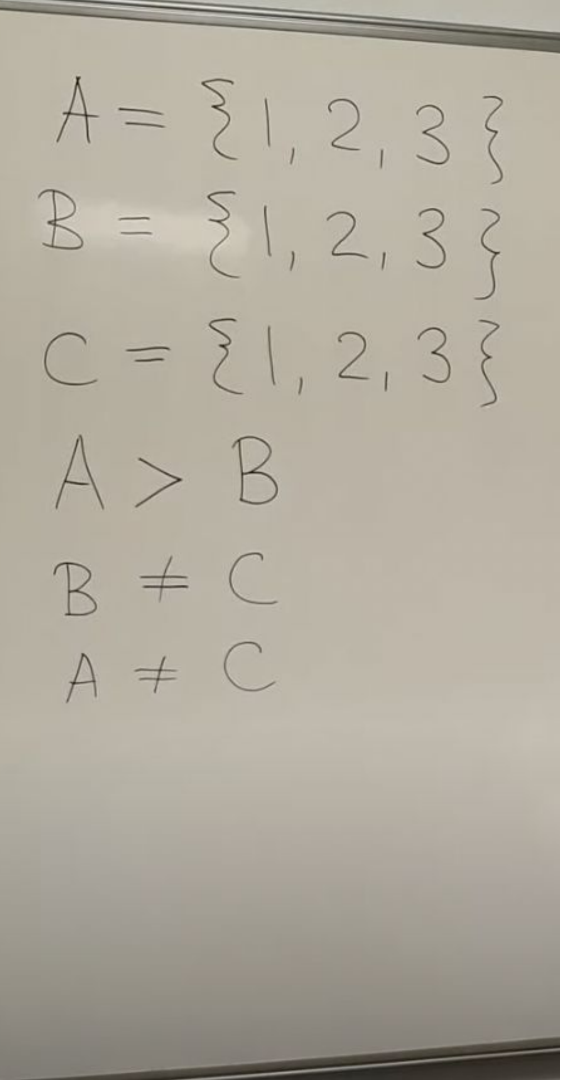
$B \neq C$

$A \neq C$

Arc Consistency Algorithm #3:

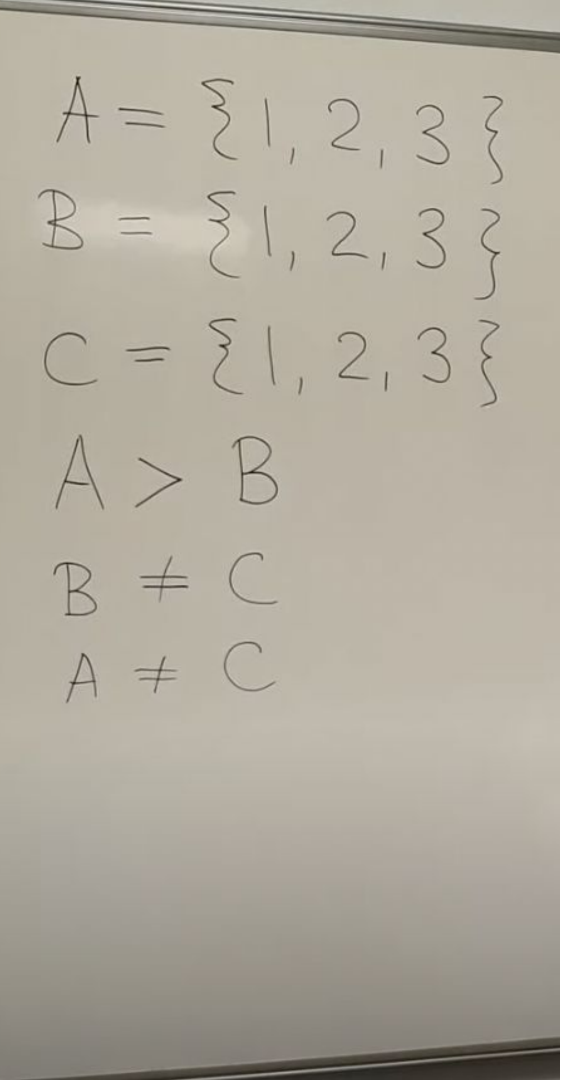


- AC-3 algorithm is one of a series of algorithms used for solving CSP



Arc Consistency Algorithm #3:

- AC-3 algorithm is one of a series of algorithms used for solving CSP
- Developed by Alan Mackworth in 1977

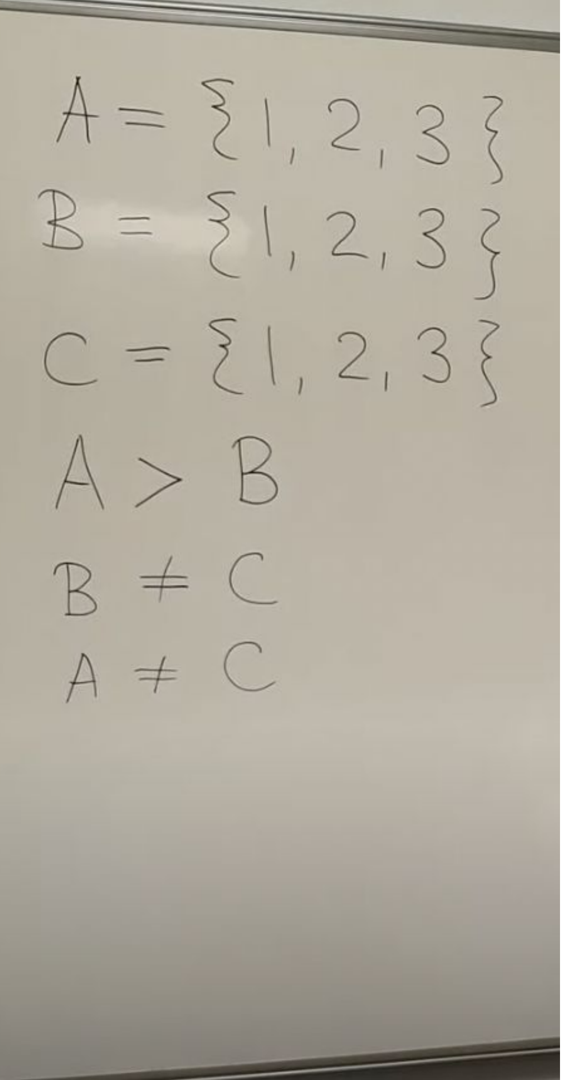


Arc Consistency Algorithm #3:

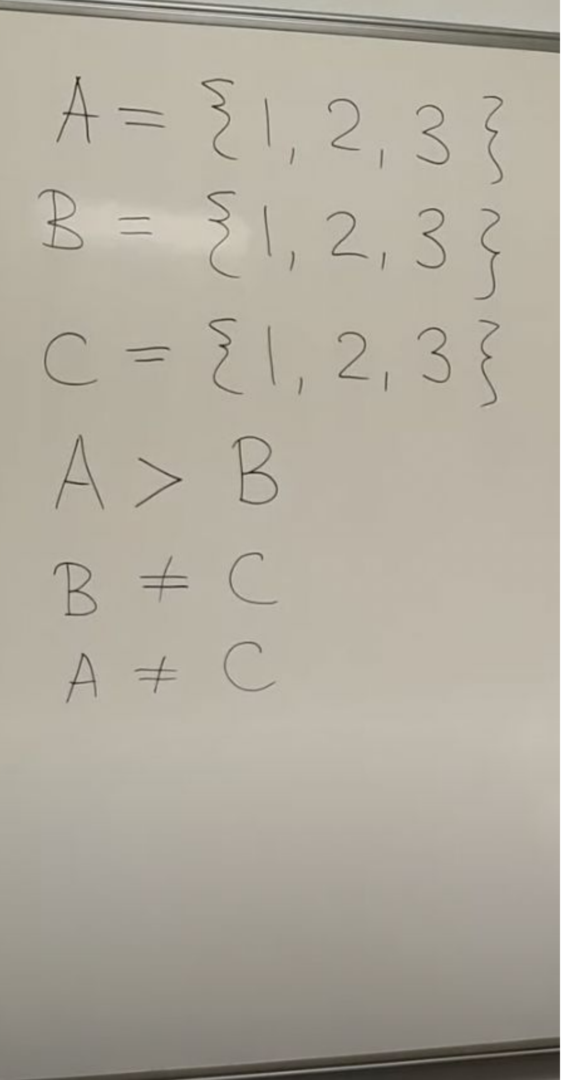
- AC-3 algorithm is one of a series of algorithms used for solving CSP
- Developed by Alan Mackworth in 1977
- The earlier AC algorithms are often considered too inefficient

Arc Consistency Algorithm #3:

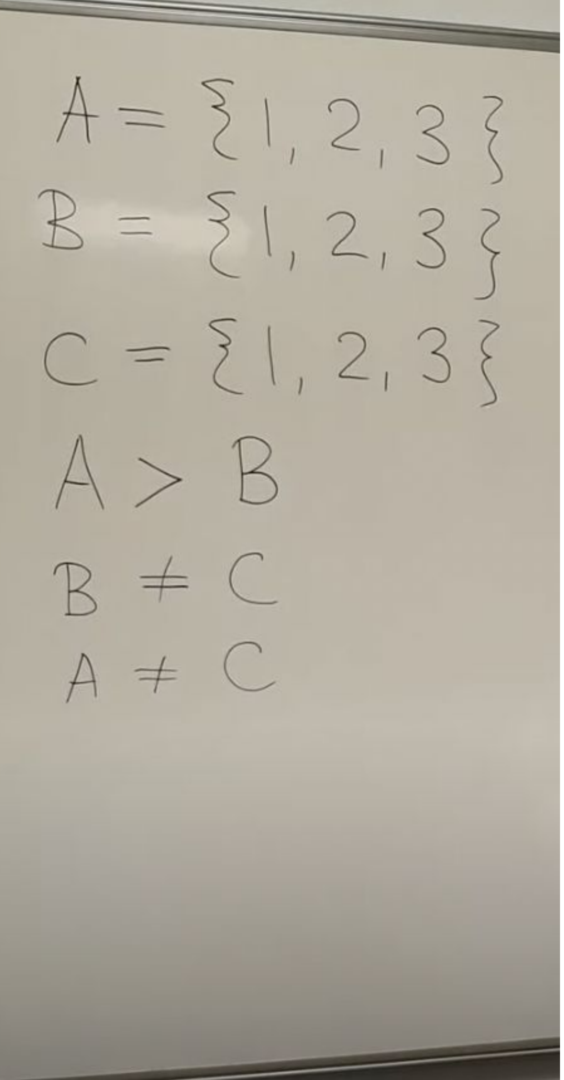
- AC-3 algorithm is one of a series of algorithms used for solving CSP
- Developed by Alan Mackworth in 1977
- The earlier AC algorithms are often considered too inefficient
- Many of the later ones are difficult to implement



Arc Consistency Algorithm #3:



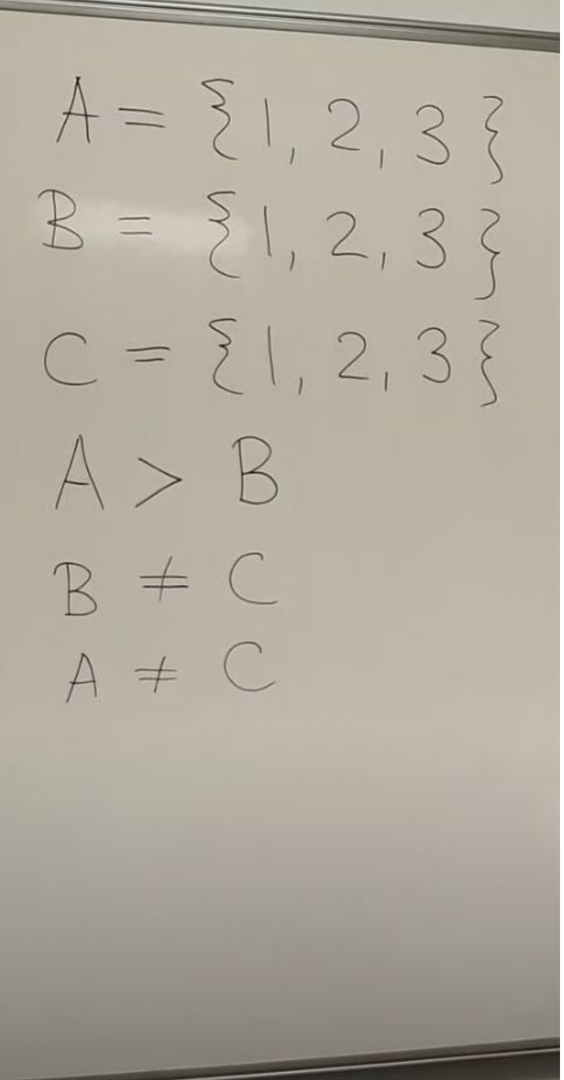
- AC-3 algorithm is one of a series of algorithms used for solving CSP
- Developed by Alan Mackworth in 1977
- The earlier AC algorithms are often considered too inefficient
- Many of the later ones are difficult to implement
- So AC-3 is the one most often taught and used for simple problems



Arc Consistency Algorithm #3:

- AC-3 algorithm is one of a series of algorithms used for solving CSP
- Developed by Alan Mackworth in 1977
- The earlier AC algorithms are often considered too inefficient
- Many of the later ones are difficult to implement
- So AC-3 is the one most often taught and used for simple problems

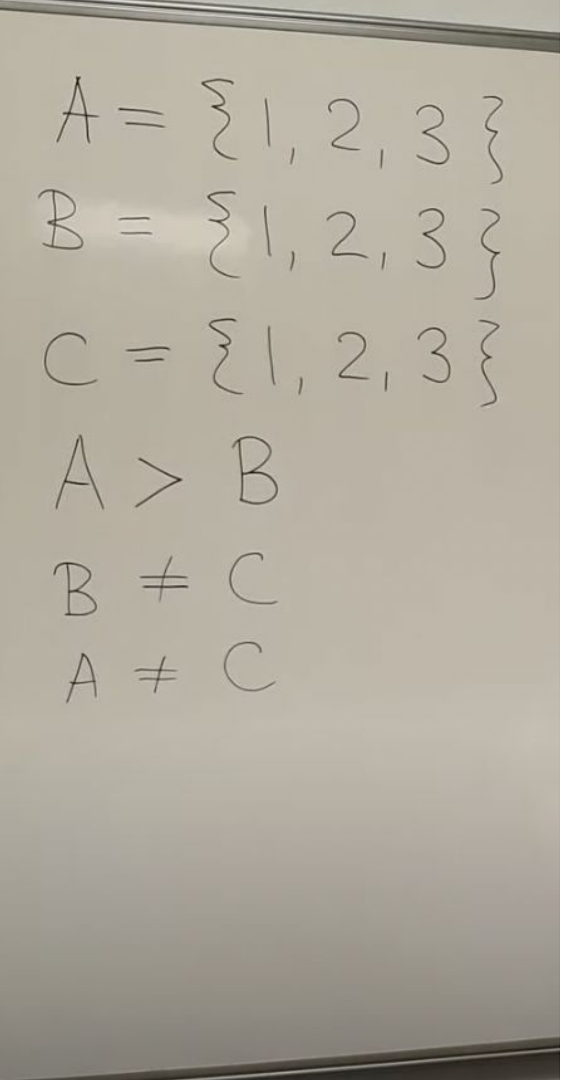
- AC-3 proceeds by examining the arcs between pairs of variables (x,y)



Arc Consistency Algorithm #3:

- AC-3 algorithm is one of a series of algorithms used for solving CSP
- Developed by Alan Mackworth in 1977
- The earlier AC algorithms are often considered too inefficient
- Many of the later ones are difficult to implement
- So AC-3 is the one most often taught and used for simple problems

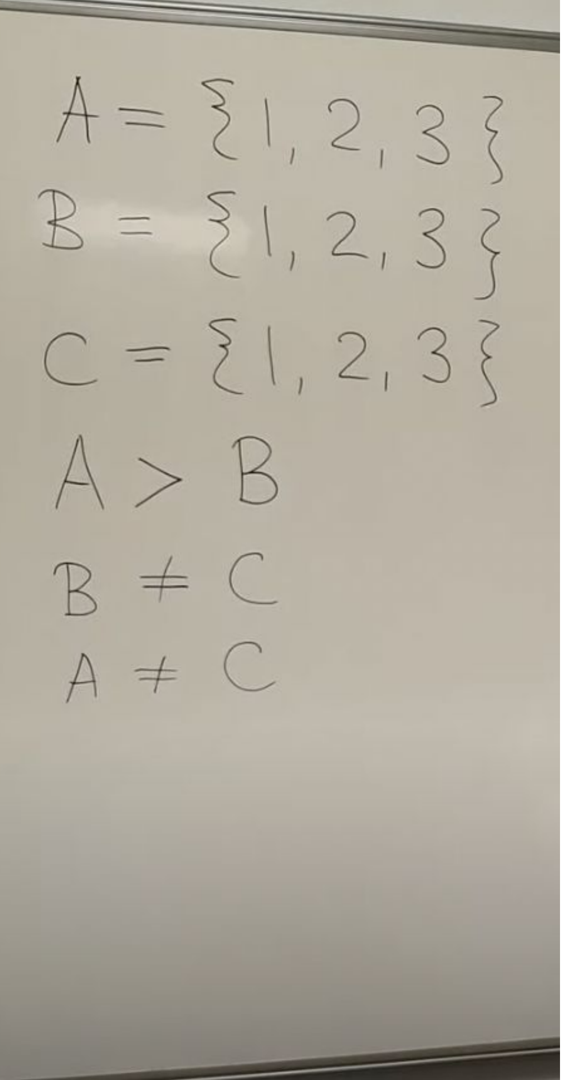
- AC-3 proceeds by examining the arcs between pairs of variables (x,y)
- Removes those values from the domain of x which aren't consistent with the constraints between x and y.



Arc Consistency Algorithm #3:

- AC-3 algorithm is one of a series of algorithms used for solving CSP
- Developed by Alan Mackworth in 1977
- The earlier AC algorithms are often considered too inefficient
- Many of the later ones are difficult to implement
- So AC-3 is the one most often taught and used for simple problems

- AC-3 proceeds by examining the arcs between pairs of variables (x,y)
- Removes those values from the domain of x which aren't consistent with the constraints between x and y.
- The algorithm keeps a collection of arcs that are yet to be checked

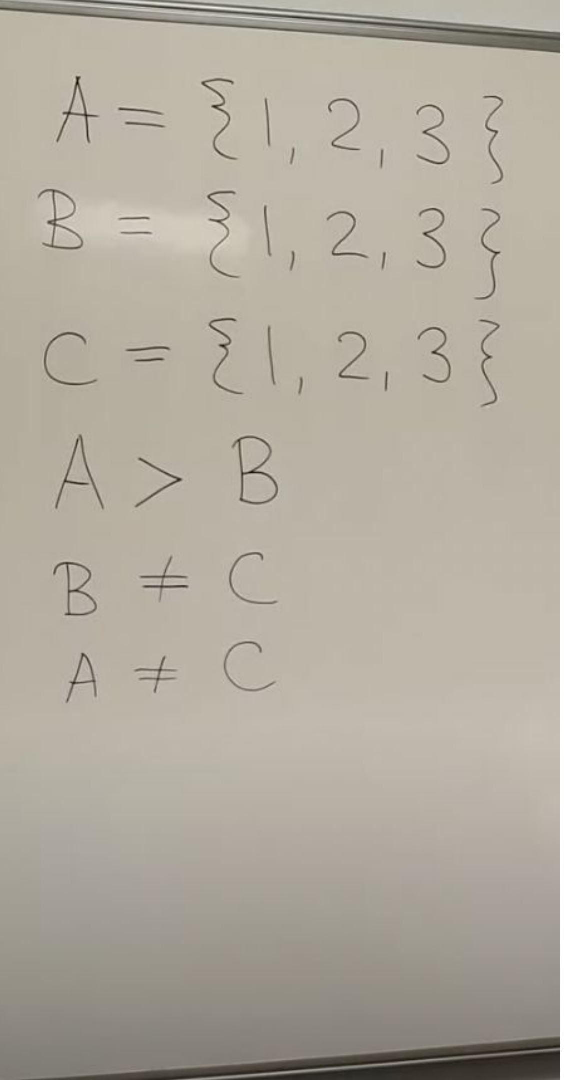


Arc Consistency Algorithm #3:

- AC-3 algorithm is one of a series of algorithms used for solving CSP
- Developed by Alan Mackworth in 1977
- The earlier AC algorithms are often considered too inefficient
- Many of the later ones are difficult to implement
- So AC-3 is the one most often taught and used for simple problems

- AC-3 proceeds by examining the arcs between pairs of variables (x,y)
- Removes those values from the domain of x which aren't consistent with the constraints between x and y.
- The algorithm keeps a collection of arcs that are yet to be checked

- When the domain of a variable has any values removed, all the arcs of constraints pointing to that pruned variable (except the arc of the current constraint) are added to the collection.



Arc Consistency Algorithm #3:

- AC-3 algorithm is one of a series of algorithms used for solving CSP
- Developed by Alan Mackworth in 1977
- The earlier AC algorithms are often considered too inefficient
- Many of the later ones are difficult to implement
- So AC-3 is the one most often taught and used for simple problems

- AC-3 proceeds by examining the arcs between pairs of variables (x,y)
- Removes those values from the domain of x which aren't consistent with the constraints between x and y.
- The algorithm keeps a collection of arcs that are yet to be checked

- When the domain of a variable has any values removed, all the arcs of constraints pointing to that pruned variable (except the arc of the current constraint) are added to the collection.

- Since the domains of the variables are finite and either one arc or at least one value are removed at each step, this algorithm is guaranteed to terminate.

Variables and Domain	Queue	Constraints and Arcs
A = {1, 2, 3}		A > B
B = {1, 2, 3}		
C = {1, 2, 3}		B = C

Variables and Domain	Queue	Constraints and Arcs
A = {1, 2, 3}		A > B
B = {1, 2, 3}		B < A
C = {1, 2, 3}		B = C
		C = B

Variables and Domain	Queue	Constraints and Arcs
A = {1, 2, 3}	A > B	A > B
B = {1, 2, 3}	B < A	B < A
C = {1, 2, 3}	B = C	B = C
	C = B	C = B

Variables and Domain	**Queue**	**Constraints and Arcs**
A = {~~1~~, 2, 3}	~~A > B~~	A > B
B = {1, 2, 3}	B < A	B < A
C = {1, 2, 3}	B = C	B = C
	C = B	C = B

Variables and Domain	**Queue**	**Constraints and Arcs**
A = {~~1~~, 2, 3}	~~A > B~~	A > B
B = {1, 2, ~~3~~}	~~B < A~~	B < A
C = {1, 2, 3}	B = C	B = C
	C = B	C = B

Variables and Domain	**Queue**	**Constraints and Arcs**
$A = \{\cancel{1}, 2, 3\}$	$\cancel{A > B}$	$A > B$
$B = \{1, 2, \cancel{3}\}$	$\cancel{B < A}$	$B < A$
$C = \{1, 2, 3\}$	$B = C$	$B = C$
	$C = B$	$C = B$
	$A > B$	

Variables and Domain	**Queue**	**Constraints and Arcs**
A = {~~1~~, 2, 3}	~~A > B~~	A > B
B = {1, 2, ~~3~~}	~~B < A~~	B < A
C = {1, 2, 3}	~~B = C~~	B = C
	C = B	C = B
	A > B	

Variables and Domain	Queue	Constraints and Arcs
A = {~~1~~, 2, 3}	~~A > B~~	A > B
B = {1, 2, ~~3~~}	~~B < A~~	B < A
C = {1, 2, ~~3~~}	~~B = C~~	B = C
	~~C = B~~	C = B
	A > B	

Variables and Domain	**Queue**	**Constraints and Arcs**
A = {~~1~~, 2, 3}	~~A > B~~	A > B
B = {1, 2, ~~3~~}	~~B < A~~	B < A
C = {1, 2, ~~3~~}	~~B = C~~	B = C
	~~C = B~~	C = B
	A > B	
	B = C	

Variables and Domain	Queue	Constraints and Arcs
A = {~~1~~, 2, 3} B = {1, 2, ~~3~~} C = {1, 2, ~~3~~}	~~A > B~~	A > B B < A B = C C = B
	~~B < A~~	
	~~B = C~~	
	~~C = B~~	
	~~A > B~~	
	B = C	

Variables and Domain	Queue	Constraints and Arcs
A = {~~1~~, 2, 3}	~~A > B~~	A > B
B = {1, 2, ~~3~~}	~~B < A~~	B < A
C = {1, 2, ~~3~~}	~~B = C~~	B = C
	~~C = B~~	C = B
	~~A > B~~	
	~~B = C~~	

Consider a graph with 8 nodes A_1, A_2, A_3, A_4, H, T, F_1, F_2.

- A_i is connected to A_{i+1} for all i
- each A_i is connected to H
- H is connected to T
- T is connected to each F_i

Find a 3-coloring of this graph by hand.

Facts:
In five houses, each with a different color, live five persons of different nationalities, each of whom prefers a different brand of candy, a different drink, and a different pet.

The Englishman lives in the red house. The Spaniard owns the dog.
The Norwegian lives in the first house on the left.
The green house is immediately to the right of the ivory house.
The man who eats Hershey bars lives in the house next to the man with the fox.
Kit Kats are eaten in the yellow house. The Norwegian lives next to the blue house.
The Smarties eater owns snails. The Snickers eater drinks orange juice.
The Ukrainian drinks tea. The Japanese eats Milky Ways.
Kit Kats are eaten in a house next to the house where the horse is kept.
Coffee is drunk in the green house. Milk is drunk in the middle house.

Question:
Where does the zebra live, and in which house do they drink water?

	1	2	3	4	5	6	7	8	9
A			3		2		6		
B	9			3		5			1
C			1	8		6	4		
D			8	1		2	9		
E	7								8
F			6	7		8	2		
G			2	6		9	5		
H	8			2		3			9
I			5		1		3		

Coding problem for your own practice (not for grading):

Write a code to solve the Sudoku puzzle with given starting state.

How many unique configurations are possible that satisfy all the constraints? Find this for a given starting state, and also when the starting state is blank.

If you can also make a frontend, I will upload it to the Sitare GitHub repo.

k-Queens Problem:

Place k queens on an n x n chessboard such that no two queens are attacking each other, where $k \leq n^2$ is given.

What is the maximum value of k for a given n?

For practice and not for grading.