Constraint Satisfaction Problems are a subclass of problems where

- the path to the solution is not relevant
- the state space is sufficiently large

We solve these problems by breaking up the state space into smaller atomic elements.

A constraint satisfaction consists of a few parts:

1. variables $X_1, ...X_n$
2. domain $D_1, ..., D_n$ of values for each X to be assigned
3. constraints $X_i D_i$ | consistent assignments

Consider a state map of Australia:
We wish to color the map in 3 colors such that adjacent states differ in color.
{X} = {WA, NT, NSW, V, SA, T}
D = {R, G, B}
WA != ST...
We can visualize this problem using a constraint graph, since constraints are binary.
This is as opposed to unary constraints (single element rules).
The Rules:

1. each variable forms a node
2. each constraint forms an edge

Constraint Satisfaction Problems seem specific, but the algorithms end up very generalizable.
There are two categories of problems

1. **Hard CSP** ≡ no violation of constraints allowed
2. **Soft CSP** ≡ some constraints are preference constraints and not required.
   (These are also called constraint optimization problems)

There is one major form of problem where contests are held for who can solve it the best:

1 Satisfiability (SAT)

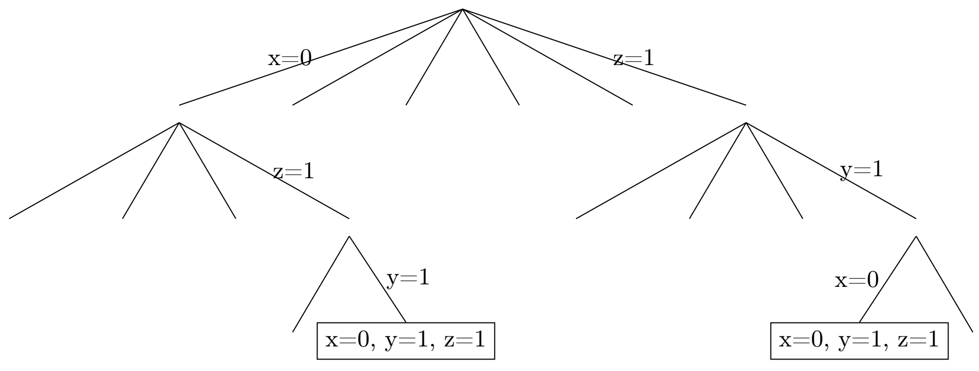
Variables: $X_1, X_2, ..., X_n$
Values: $1, 0$
Constraints: $\{X_1 \lor \neg X_2 \lor X_4, X_2 \lor \neg X_3 \lor X_5, ...\}$

This was the first and is now the prototypical NP-Complete problem!
Complete:

- the "hardest" problem in a given complexity class
- if you can provide a solution, you can solve all problems in the class

Formulating CSP as Search

Let's walk through an example tree w/ var = x, y, z & $D = 0, 1$:



This is terrible and would be unfeasible; luckily we can note there are only 8 complete states!
$\Longrightarrow$ If we let nodes be variables and edges values with DFS, we get time $O(d^n)$ & space $O(dn)$.

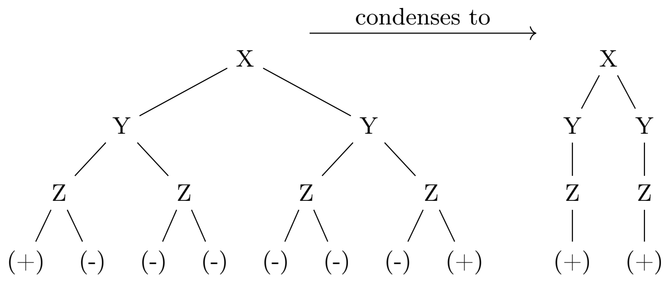
A straight BFS may not be optimal, however, as we may violate constraints early;
thus we backtrack, failure testing at intermediate nodes.

We can improve the efficiency of backtracking by answering a few questions:

1. Which variable do we assign next?
2. Which value should we try next?
3. Can we detect failure early?
4. Can we take advantage of the problem structure?

Variable & Value Ordering

Consider the following problem:
Variables: x, y, z
Values: $0, 1$
Constraints: $x = y, y = z$



Note that if we use the tree to the left, we must search the entire tree.
If we reorder the variables, however, we can prune the tree and use the right tree.

When it comes to ordering,
Variable ordering changes branch count.
Value ordering changes branch order.
We can choose variables and values based on a few models:

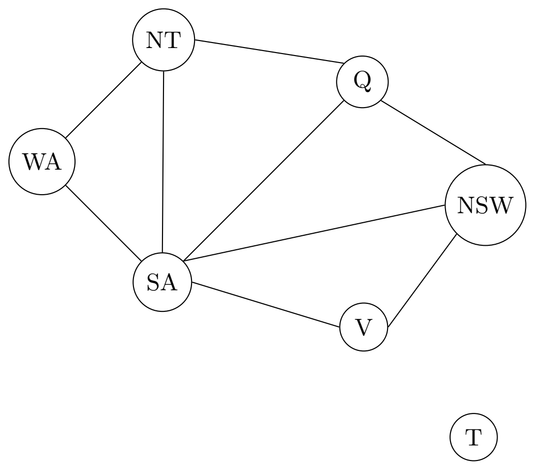
1. **Most Constrained Variable** $\equiv$ choose the variable with the minimum remaining values
2. **Most Constraining Variable** $\equiv$ choose the variable that places the most constraints
3. **Least Constraining Value** $\equiv$ choose the value that places the least restrictions

We choose this model heuristically via direct time comparison.

This leads us toward our methods of early failure detection
Forward Checking – less effective but more efficient
Arc Consistency – more effective but less efficient

Forward Checking

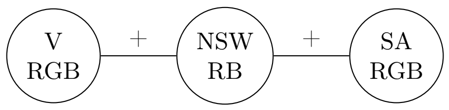
$\equiv$ keep track of the consistent values unassigned variable; if a variable has no values left, then failure!



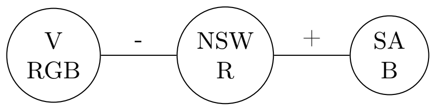
WA	NT	Q	NSW	V	SA	T
(R)GB	~~RG~~B	R(G)B	R~~GB~~	RG(B)	~~RGB~~	RGB

Arc Consistency

≡ make all arcs consistent; if we can, then assign arbitrarily left to right, else fail.
An arc is **consistent** if for all values of x, there exists a corresponding value of y;
 we can therefore propagate constraints across the arc from the right.



This arc is **consistent**



This arc is **inconsistent**

ripple effect ≡ check an arc when a value is removed.
Single ripple effect: $O(d)$
 A binary CSP arc count: $O(n^2)$
 Checking consistency of an arc: $O(n^2)$
$\implies T = O(n^2 d^3)$ for a binary CSP; that is not bad!

Symmetry

Disconnection

Consider the AU coloring problem;
 we know Tanzania is disconnected, so we can assign it a value arbitrarily.
As a general rule, we can solve disconnected components of a search problem separately.

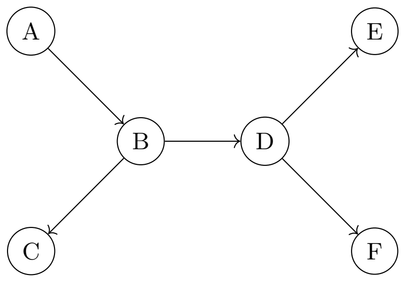
Consider a search problem with n=80 & d=2 @ 10 million nodes per second;
 the time complexity without division is $2^8$0 or 4 billion years!
 breaking the problem into 4 disconnected reduces the time cost to $4 * 2^2 0 = 0.4$ seconds!

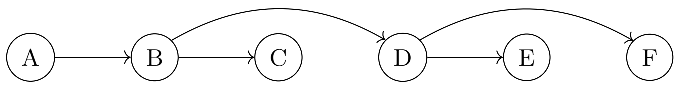
Trees

Tree structured CSPs are especially simple to solve.
Algorithm:

1. Orient the tree
2. Make arcs consistent
3. Assign values

We call the resulting structure a **backtrack tree**.
Since the arcs are consistent, we can assign any permissible value!





As an addendum, we can introduce symmetry via a constraint (such as $x < y < z$);
this greatly constricts the problem size, BUT deleting all symmetry is NP-Hard!