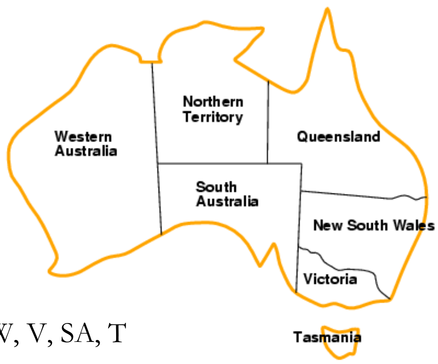
Constrained Satisfaction Problems (CSP)

- A **constraint satisfaction problem** (CSP) is defined by a set of **variables**, *X1, X2, …, Xn*, and a set of **constraints**, *C1, C2, …, Cm*.
- Each variable *Xi* has a nonempty **domain** *Di* of possible **values**.
- A **solution** to a CSP is a complete assignment of values to all variables that satisfies all the constraints.

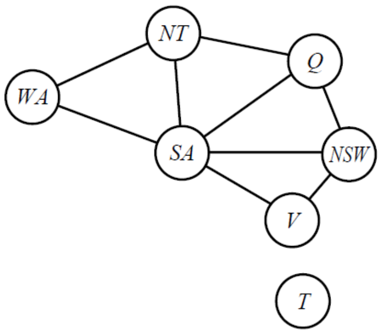
Example 1: Map-coloring



- **Variables:** WA, NT, Q, NSW, V, SA, T
- **Domains:** {red, green, blue}
- **Constraints:** adjacent regions must have different colors e.g., WA ≠ NT.

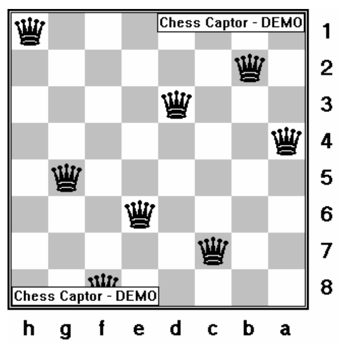
Solutions are *complete* and *consistent* assignments, e.g., WA = red, NT = green, Q = red, NSW = green, V = red, SA = blue, T = green.

CSP can be visualized as **constraint graph,** where nodes are variables and arcs are constraints.



- If domain size is *d*, and there are *n* variables, then there are d^n complete solutions.

Example 2: n-queen



- **Variables**: Q1, …., Qn.
- **Values**: the set {1, ...,n}.
- **Constraints**: no queen Qi threaten the others.

There are n^n possible assignments in the search space.

CSP can be solved by *a standard search* method as follows:
Initial state: the empty assignment {}, in which all variables are unassigned.
Successor function: a value can be assigned to any unassigned variable, provided that it does not conflict with previously assigned variables.
Goal test: the current assignment is complete.
Path cost: a constant cost (e.g., 1) for every step.

Suppose we apply breadth-first search to the generic CSP problem, then:

- The deep of any solution is *n*. (good news).
- The number of leaves is $n!.d^n$ (bad news), even though there are only d^n complete solutions.

BACKTRACKING SEARCH FOR CSPS

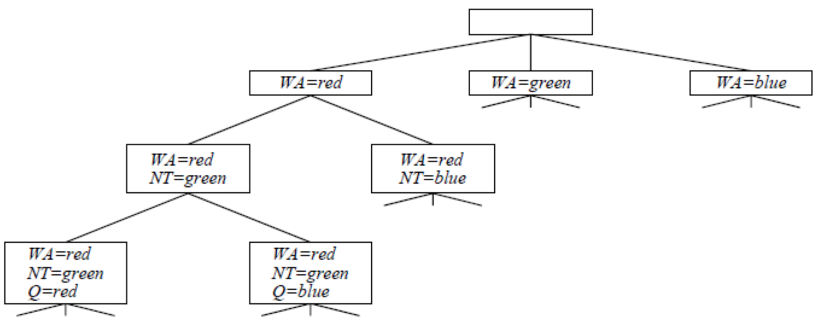
In CSP's, variable assignments are **commutative.**

For example, *[WA = red then NT = green]* is the same as *[NT = green then WA = red]*

The term **backtracking search** is used for a depth-first search that chooses values for one variable at a time and backtracks when a variable has no legal values left to assign.

Backtracking search algorithm:

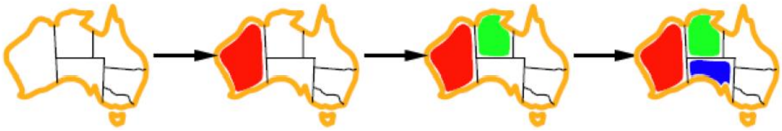
```
function BACKTRACKING-SEARCH(csp) returns solution/failure
   return RECURSIVE-BACKTRACKING({ }, csp)

function RECURSIVE-BACKTRACKING(assignment, csp) returns soln/failure
   if assignment is complete then return assignment
   var ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp], assignment, csp)
   for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
       if value is consistent with assignment given CONSTRAINTS[csp] then
           add {var = value} to assignment
           result ← RECURSIVE-BACKTRACKING(assignment, csp)
           if result ≠ failure then return result
           remove {var = value} from assignment
   return failure
```



Improving backtracking efficiency:

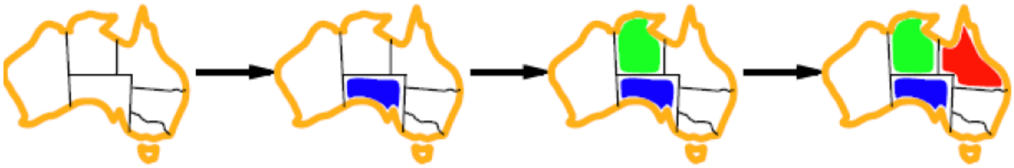
- **Q1**\ Which variable should be assigned next?
    - Choose the variable with the fewest legal values **minimum remaining values** (MRV) heuristic.

For example, after the assignments for WA=red and NT =green, there is only one possible value for SA, so it makes sense to assign SA=blue next rather than assigning Q.



- To select the first variable, we use the **degree heuristic**. It attempts to reduce the branching factor on future choices by selecting the variable that is involved in the largest number of constraints on other unassigned variables.

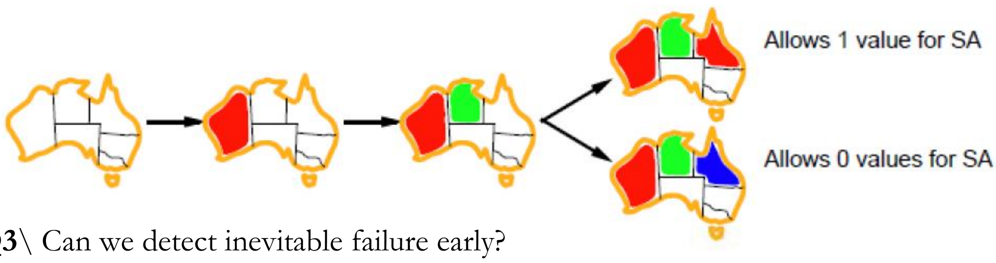
Example: SA is the variable with highest degree, 5; the other variables have degree 2 or 3, except for T, which has 0.



Q2\ In what order should its values be tried?

- Use **least-constraining-value** heuristic. It prefers the value that rules out the fewest choices for the neighboring variables in the constraint graph.

Example: if we assign WA=red and NT =green, so we assign q=red instead of blue to leave the maximum flexibility for subsequent variable assignments.



- **Q3**\ Can we detect inevitable failure early?
    - Use **Forward checking** to Keep track of remaining legal values for unassigned variables. Terminate search when any variable has no legal values

Example: after v=blue, SA has no legal value!

	WA	*NT*	*Q*	*NSW*	*V*	*SA*	*T*
Initial domains	R G B	R G B	R G B	R G B	R G B	R G B	R G B
After *WA=red*	R	G B	R G B	R G B	R G B	G B	R G B
After *Q=green*	R	B	G	R B	R G B	B	R G B
After *V=blue*	R	B	G	R	B		R G B

However, forward checking doesn't provide early detection for all failures. Example: NT and SA cannot both be blue!

We use **arc consistency to** provides a fast method of **constraint propagation** that is substantially stronger than forward checking.

Arc consistency:

$X \rightarrow Y$ is consistent iff for *every* value of X there is *some* allowed value of Y. When checking $X \rightarrow Y$, throw out any values of X for which there isn't an allowed value of Y.

	WA	*NT*	*Q*	*NSW*	*V*	*SA*	*T*
Initial domains	R G B	R G B	R G B	R G B	R G B	R G B	R G B
After *WA=red*	R	G B	R G B	R G B	R G B	G B	R G B
After *Q=green*	R	B	G	R B	R G B	B	R G B
After *V=blue*	R	B	G	R	B		R G B

	WA	*NT*	*Q*	*NSW*	*V*	*SA*	*T*
Initial domains	R G B	R G B	R G B	R G B	R G B	R G B	R G B
After *WA=red*	R	G B	R G B	R G B	R G B	G B	R G B
After *Q=green*	R	B	G	R ~~B~~	R G B	B	R G B
After *V=blue*	R	B	G	R	B		R G B

- If X loses a value, all pairs $Z \rightarrow X$ need to be rechecked

	WA	*NT*	*Q*	*NSW*	*V*	*SA*	*T*
Initial domains	R G B	R G B	R G B	R G B	R G B	R G B	R G B
After *WA=red*	R	G B	R G B	R G B	R G B	G B	R G B
After *Q=green*	R	B	G	R B	~~R~~ G B	B	R G B
After *V=blue*	R	B	G	R	B		R G B

	WA	*NT*	*Q*	*NSW*	*V*	*SA*	*T*
Initial domains	R G B	R G B	R G B	R G B	R G B	R G B	R G B
After *WA=red*	R	G B	R G B	R G B	R G B	G B	R G B
After *Q=green*	R	B	G	R B	R G B	~~B~~	R G B
After *V=blue*	R	B	G	R	B		R G B

- Arc consistency detects failure earlier than forward checking

- Can be run as a preprocessor or after each assignment.