CS249 – ARTIFICIAL INTELLIGENCE - 1

8th Week Slides by:

2201AI38 – Swathi

2201AI39 – Tanay

2201AI40 – Tanisha

2201AI41 – Vaani

2201AI42 – Vaishika

*Everyone had equal contributions.

α-β Pruning

α Bond of J:

- The maximum current payoff of all MAX ancestors of J.
- Exploration of a min node J is stopped when its payoff equals/falls below α.
- In a min node, we update β.

β Bond of J:

- The minimum current payoff of all MIN ancestors of J.
- Exploration of a max node J is stopped when its payoff expands/exceeds below β.
- In a max node, we update α.

α - β Pruning Procedure

- Initial call is with V (root, α, β)

1. If J is a terminal, return $V(J) = h(J)$

2. If J is a max node:

for each successor J_k of J in successor:

Set $\alpha = \max(\alpha, V(J_k, \alpha, \beta))$

if $\alpha \geq \beta$, return β

else continue

return α

3. If J is a min node:

for each successor J_k of J in successor:

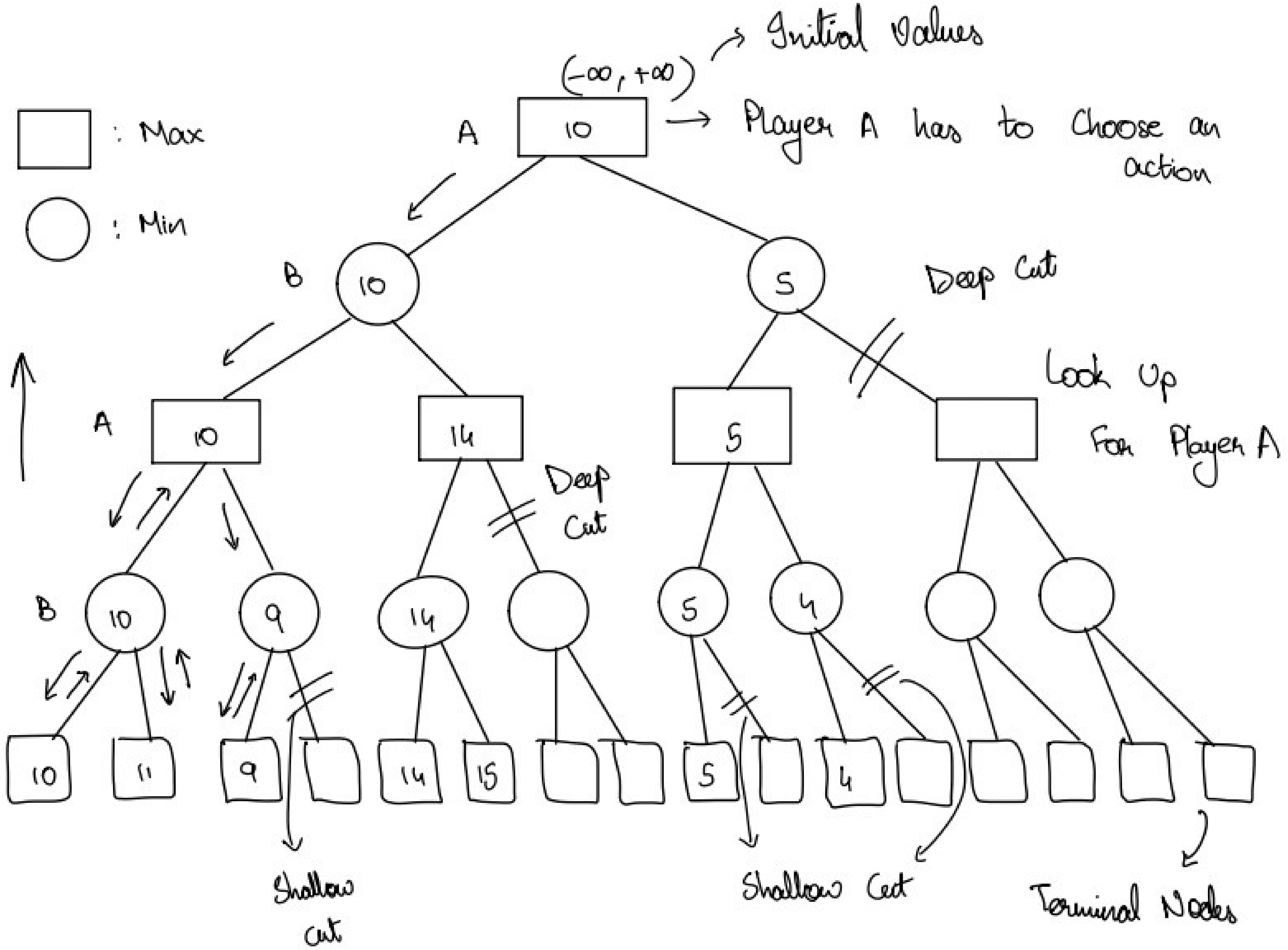
Set $\beta = \min(\beta, V(J_k, \alpha, \beta))$

if $\alpha \geq \beta$, return α

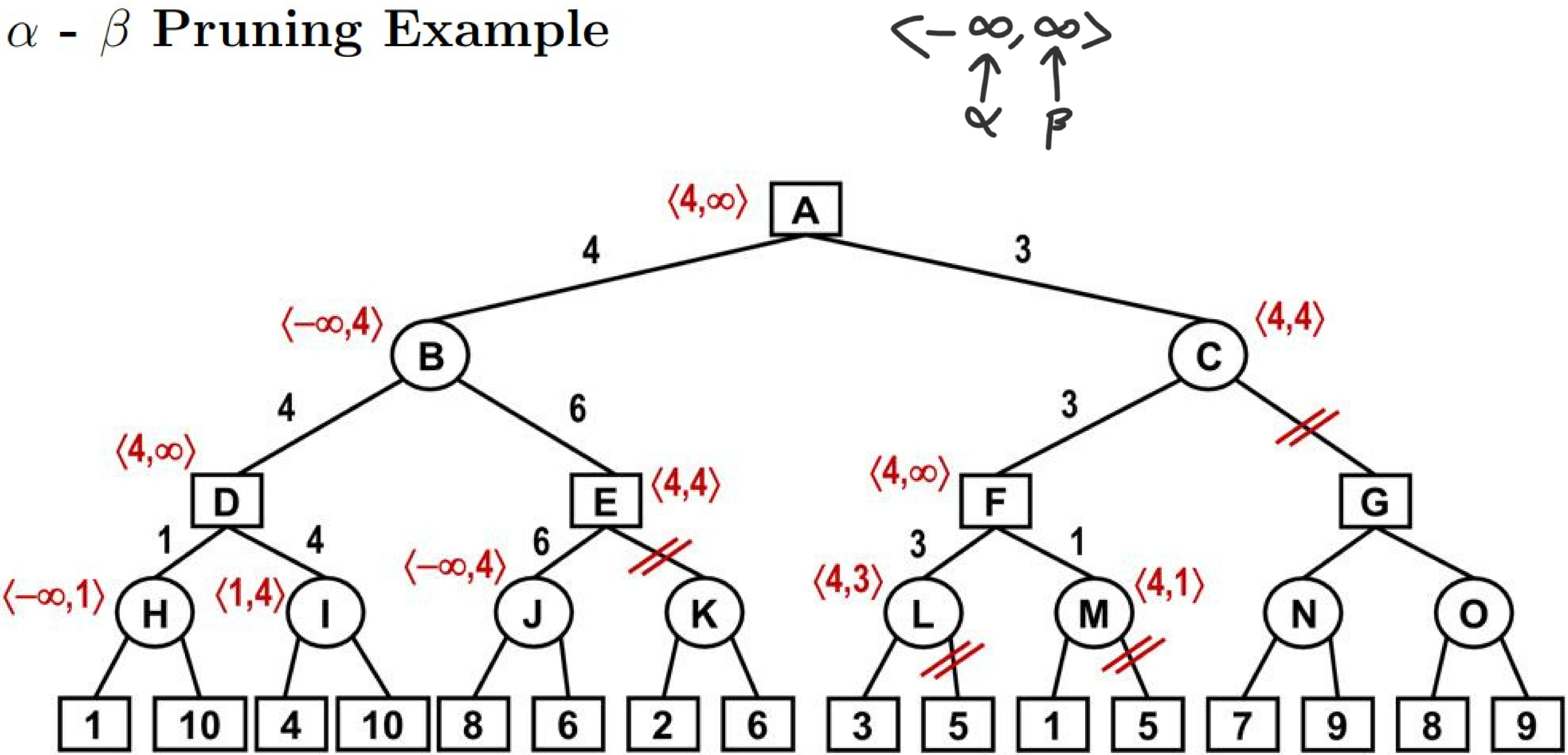
else continue

return β

α - β Pruning Example



α - β Pruning Example



Traversal follows DFS (System Stack)

The α–β algorithm

Basically MINIMAX + keep track of α, β + prune

```
function MAX-VALUE(state, game, α, β) returns the minimax value of state
    inputs: state, current state in game
            game, game description
            α, the best score for MAX along the path to state
            β, the best score for MIN along the path to state

    if CUTOFF-TEST(state) then return EVAL(state)
    for each s in SUCCESSORS(state) do
        α ← MAX(α, MIN-VALUE(s, game, α, β))
        if α ≥ β then return β
    end
    return α
```

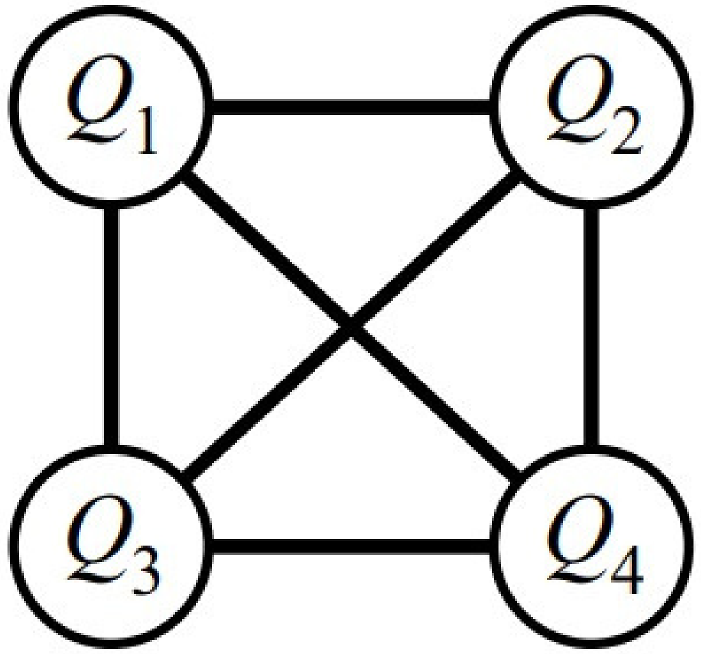
```
function MIN-VALUE(state, game, α, β) returns the minimax value of state

    if CUTOFF-TEST(state) then return EVAL(state)
    for each s in SUCCESSORS(state) do
        β ← MIN(β, MAX-VALUE(s, game, α, β))
        if β ≤ α then return α
    end
    return β
```

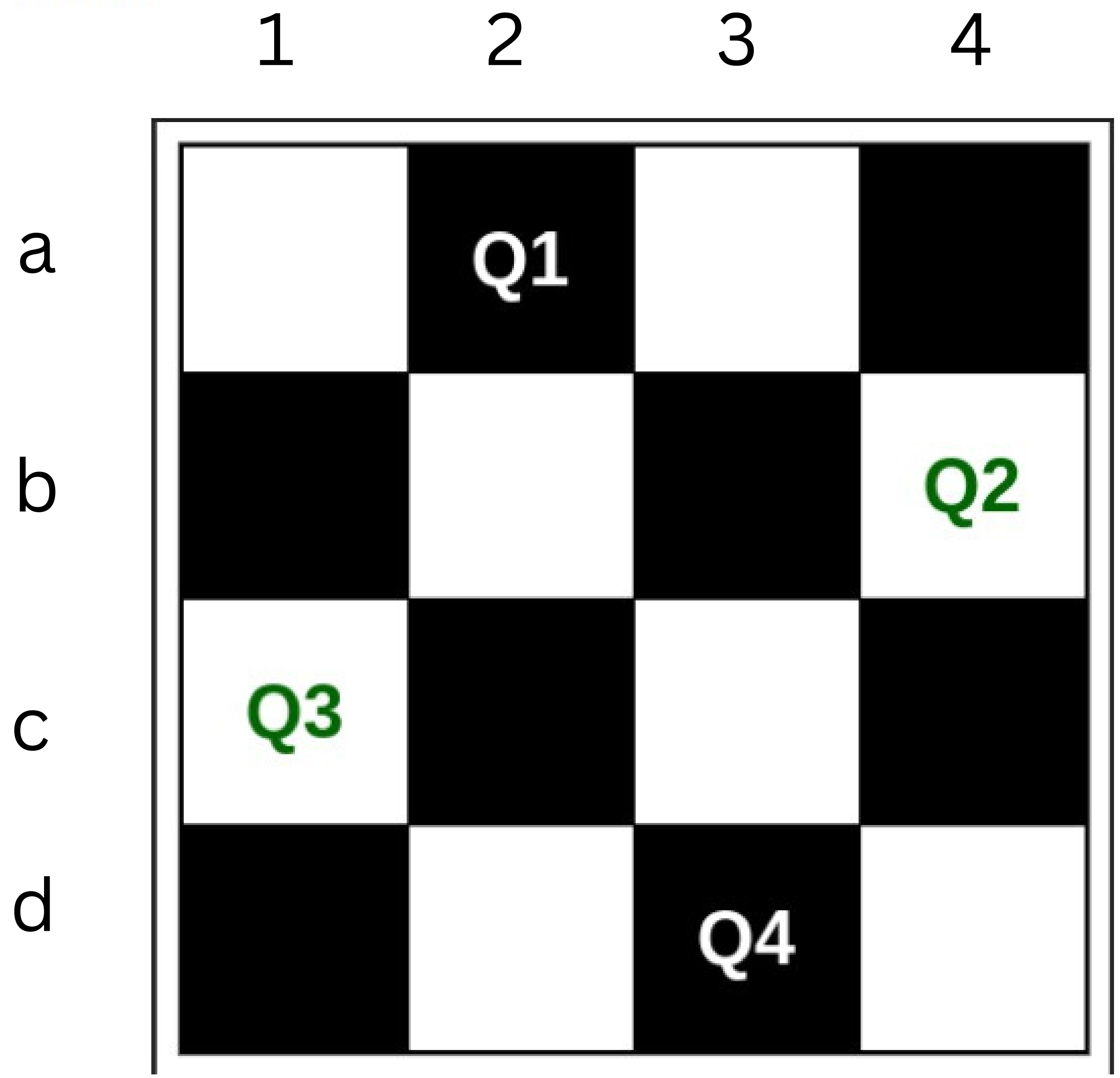
Basic Constraint Satisfaction Problem (CSP) Formulation

1. **Variables:**
   - Finite set of variables $\{V_1, V_2, \ldots, V_n\}$.
2. **Domains:**
   - Each variable has a domain $D_1, D_2, \ldots, D_n$ from which it can take a value.
   - The domain may be discrete or continuous.
3. **Satisfaction Constraints (S.C.):**
   - Finite set of satisfaction constraints $C_1, C_2, \ldots, C_n$.
   - Constraint may be unary/binary among many variables of the domain.
   - All constraints have a Yes/No answer for satisfaction for given values of variables.
4. **Optimization Criteria (Optional):**
   - A set of optimization functions $O_1, O_2, \ldots, O_p$, generally of MAX/MIN type.
5. **Solution:**
   - To find a consistent assignment of domain values to each variable so that all constraints are satisfied and optimal criteria (if any) are met.

1. n-Queens Problem: (4-Queens example)

- **Variables:** $\{a, b, c, d\}$ (rows)
- **Domain:** $\{1, 2, 3, 4\}$ (columns)
- **Constraints:** No pair of queens should be attacking each other
- **Solution:** Positioning Queens in non-attacking positions





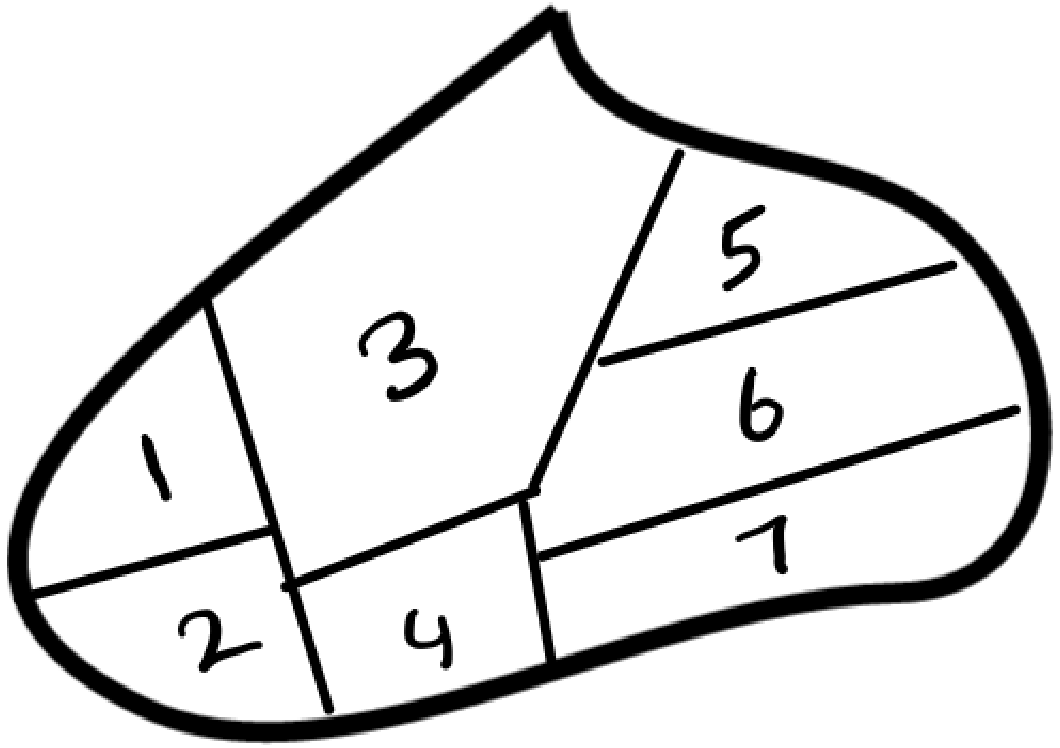
2. Cryptarithmetic Puzzle

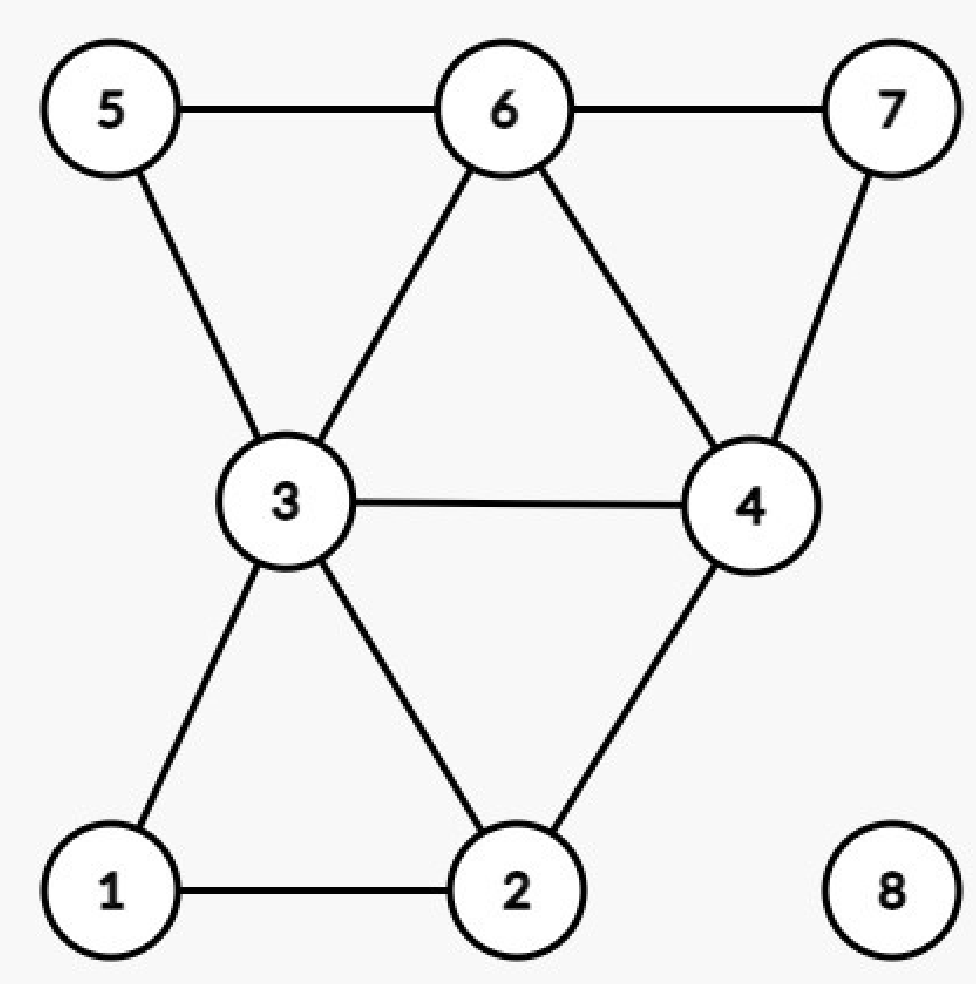
- **Variables:** $\{B, O, Y, E, M, L, I, A, R\}$
- **Domain:** $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
- **Constraints:**
  - Uniqueness: Each variable must be assigned a unique value from the domain.
  - Multiplication should be matched: The multiplication should be correct when the values of the variables are substituted into the puzzle.
- **Solution:** Assigning values to variables from the domain such constraints are satisfied and the puzzle is solved.

$$
\begin{array}{cccccc}
 & & & B & O & B \\
 & & \times & B & O & B \\
\hline
 & & & M & E & O \\
 & M & I & L & O & - \\
M & E & O & Y & - & - \\
\hline
M & A & R & L & E & Y
\end{array}
$$

3. Region Coloring (Graph Coloring)

- **Variables:** $\{1, 2, 3, 4, 5, 6, 7, 8\}$ (Nodes)
- **Domain:** $\{C_1, C_2, \ldots, C_k\}$ (Colors)
- **Constraints:**
  - Number of two adjacent nodal regions should be of the same color.
- **Solution:** To each node n_i, assign color C_j, where $i = 1 \to 8$ and $j = 1 \to k$. The optimization objective is to minimize k.





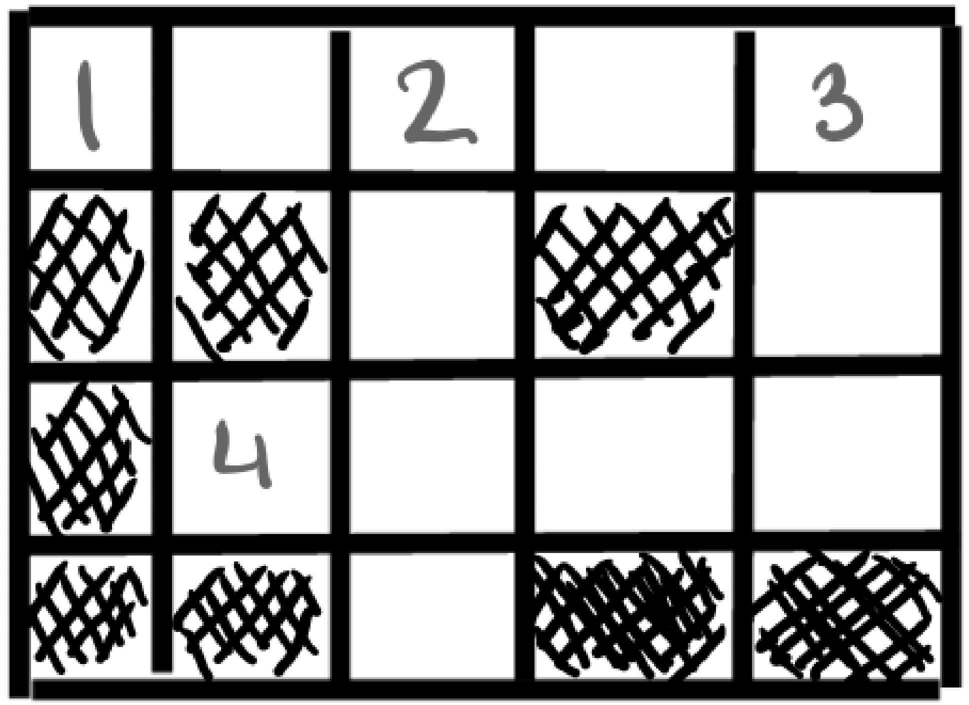
4. Knapsack Problem

- **Set of items S:** $\{s_1, s_2, s_3, \ldots, s_n\}$
- **Each item has weights W:** $\{w_1, w_2, \ldots, w_n\}$
- **Each item has value V:** $\{v_1, v_2, v_3, \ldots, v_n\}$
- **Capacity of bag:** C such that $\sum s_i w_i \leq C$
- **Profit:** $\max\left(\sum s_i v_i\right)$

- **Variable:** $\{s_1, s_2, \ldots, s_n\}$
- **Domain:** $\{0, 1\}$
- **Constraint:** $\sum_i s_i w_i \leq C$
- **Optimization:** Maximize $\sum_i s_i v_i$
- **Solution:** Set of chosen items which maximize the profit

5. Crossword Puzzle

- **Variable:** $\{1, 2, 3, 4\}$
- **Domain:** Word-List
- **Constraint:** $(1, 2), (1, 3), (2, 4), (4, 3)$ have interrelations
- **Solution:** All places correctly filled



WORD LIST :- ASTAR, HAPPY, HELLO, HOSES, LIVE, LOAD, LOOM, PEAL, PEEL, SAVE, TALK, ANT, OAK, OLD

SOLUTION :- 1: Hello, 2: Load, 3: OAK, 4: TALK

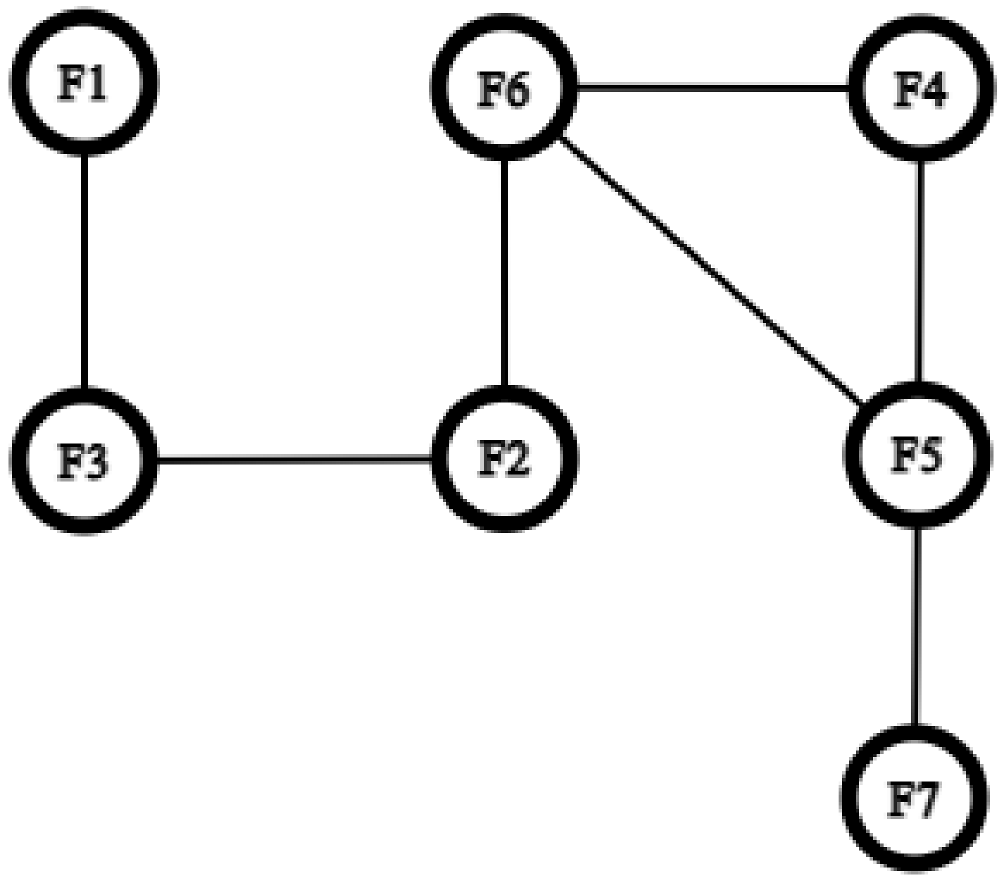
6. Flight Scheduling

Flight No.	Departure Time	Gate Opening Time	Gate Closing Time
F1	7:00	6:15	7:15
F2	8:30	7:45	8:45
F3	7:45	7:00	8:00
F4	9:45	9:00	10:00
F5	10:00	9:15	10:15
F6	9:00	8:15	9:15
F7	11:00	10:15	11:15

Flight Gate Assignment

- **Variables:** $\{F1, F2, F3, F4, F5, F6, F7\}$
- **Domain (No. of gates):** $\{G1, G2, G3, G4\}$
- **Optimization:** Minimize number of gates
- **Solution:** Assigning flights to gates
- **Constraint:** Flights with overlapping times should not be assigned the same gates

The flights are represented by nodes where , the flights which have overlapping times have been connected via edges. So, the gates can be represented using colours. So, the adjacent nodes will be assigned different gates.



CSP Solution Overview-

The following basic steps are followed::

1) CSP Graph Creation: Create a NODE for every VAR. All possible DOM values are assigned VAR initially.

2) Draw edges between nodes if there is a Binary constraint. Otherwise, draw HYPER-EDGE between nodes (for constraints more than 2).

3)Const. Propagation: Reduce the valid DOM of each VAR by applying NODE consistency, ARC/EDGE consistency, and K- consistency/Path consistency. No further reduction is possible.

If a solution is found or the problem does not have any consistent solution then **terminate**.

4)Search for a solution: Apply a search algorithm to find the solution. There are interesting properties of CSP graphs that lead to efficient algo in some cases (Trees, perfect graphs, interval graph..etc)

5) **Issues for searching** :Backtracking scheme , formal checking .

Implementation

CSP state keeps track of which variables have values so far
Each variable has a domain and a current value

datatype CSP-STATE
  components: UNASSIGNED, a list of variables not yet assigned
  ASSIGNED, a list of variables that have values
datatype CSP-VAR
  components: NAME, for i/o purposes
  DOMAIN, a list of possible values
  VALUE, current value (if any)

Constraints can be represented
- explicitly as sets of allowable values, or
- implicitly by a function that tests for satisfaction of the constraint

Constraint Propagation

In regular state-space search, an algorithm can do only one thing: search. In CSPs there is a choice: an algorithm can search (choose a new variable assignment from several possibilities) or do a specific type of inference called constraint propagation

Unary constraints - Node consistency
Binary constraints - Edgeb/w CSP node
Hyper Edge - for higher order

Node Consistency

Node consistency ensures that each variable's domain in a Constraint Satisfaction Problem (CSP) adheres to its unary constraints, simplifying the problem space by focusing on binary constraints, thus enhancing problem-solving efficiency.
For every VAR Vi remove all elements of Di that do not satisfy the unary constraints.

ARC/Edge Consistency

A variable in a CSP is **arc-consistent** if every value in its domain satisfies the variable's binary constraints. More formally, X_i is arc-consistent with respect to another variable X_j if for every value in the current domain D_i there is some value in the domain D_j that satisfies the binary constraint on the arc (X_i, X_j).

function AC-3(csp) **returns** false if an inconsistency is found and true otherwise
  inputs: csp, a binary CSP with components $(X,\ D,\ C)$
  local variables: $queue$, a queue of arcs, initially all the arcs in csp

  while $queue$ is not empty **do**
    $(X_i,\ X_j) \leftarrow$ REMOVE-FIRST($queue$)
    if REVISE(csp, $X_i,\ X_j$) **then**
      if size of $D_i\ =\ 0$ **then return** $false$
      for each X_k **in** X_i.NEIGHBORS - $\{X_j\}$ **do**
        add $(X_k,\ X_i)$ to $queue$
  return $true$

---

function REVISE(csp, $X_i,\ X_j$) **returns** true iff we revise the domain of X_i
  $revised \leftarrow false$
  for each x **in** D_i **do**
    if no value y in D_j allows (x,y) to satisfy the constraint between X_i and X_j **then**
      delete x from D_i
      $revised \leftarrow true$
  return $revised$

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.

K- Consistency

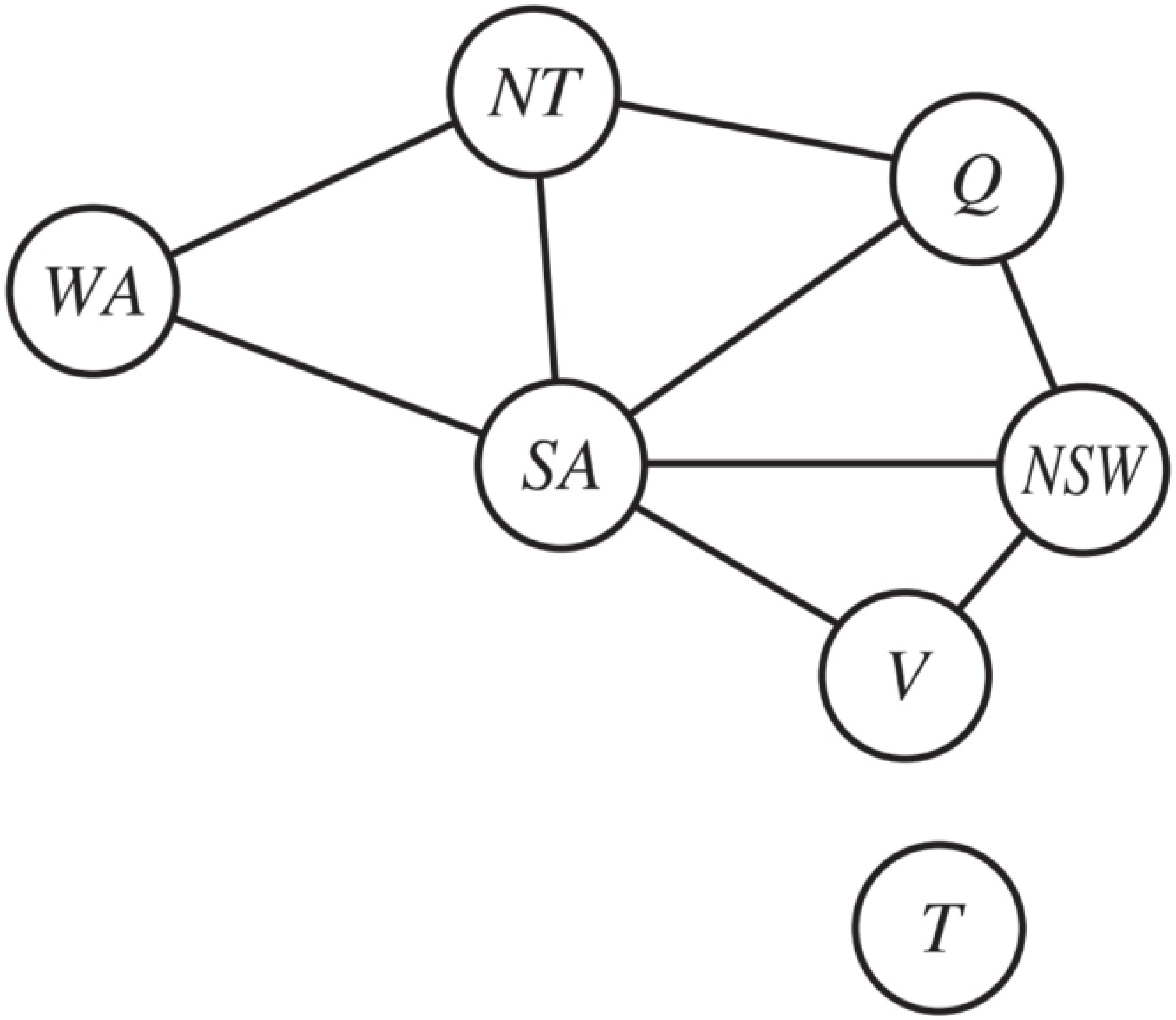
1-consistency says that, given the empty set, we can make any set of one variable consistent: this is what we called node consistency. 2-consistency is the same as arc consistency. For binary constraint networks, 3-consistency is the same as path consistency.

Back Tracking for CSP

```
BT(a){
X ->   Select Unassigned Variable.
D ->   Select an ordering for the DOM of X.
For each value V in D :
        -Add (X=V)
        -Result  ->  BT(a)
        - if Result != failure
        RETURN Result
RETURN failure }
```

Forward Checking

Propagates info from assigned to unassigned VAR.



	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

Figure 6.7 The progress of a map-coloring search with forward checking. $WA = red$ is assigned first; then forward checking deletes *red* from the domains of the neighboring variables NT and SA. After $Q = green$ is assigned, *green* is deleted from the domains of NT, SA, and NSW. After $V = blue$ is assigned, *blue* is deleted from the domains of NSW and SA, leaving SA with no legal values.