Recitation 3
Search 03

Adversarial Search

Today

Game Trees

- Introduction
- MiniMax Algorithm
- Alpha Beta Pruning
- Expectimax Search Trees

Constraint Satisfaction Problems

- Introduction
- CSP Dependency Graph
- Backtracking Search
- CSP Heuristics
- Forward Checking
- Arc Consistency Propagation

Game Playing

- So far we have assumed the situation is not going to change while we search
- Consider the case of uncertainty being caused by intermission of other agents actions, for example when playing games.
- The search tree is now a **Game Tree**
  - Each node is a state of the game (Leaves are "game-over")
  - Each agent evaluates the leaves using it's own utility function
  - Each (rational) agent maximize his utility at his nodes

Two-players zero-sum game

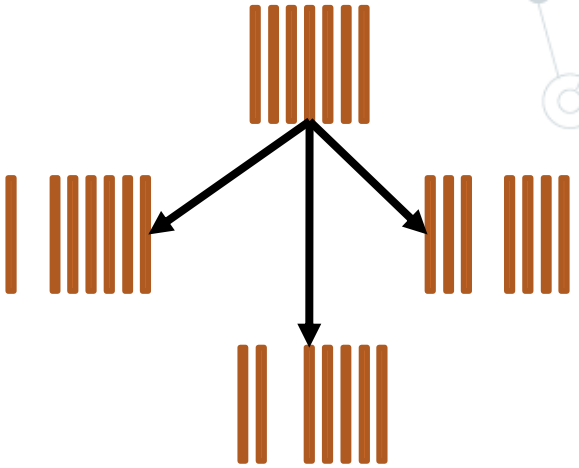
$$u_{1(v)} = -u_{2(v)}$$

- Game tree is layered with max and min nodes

Example – Nim

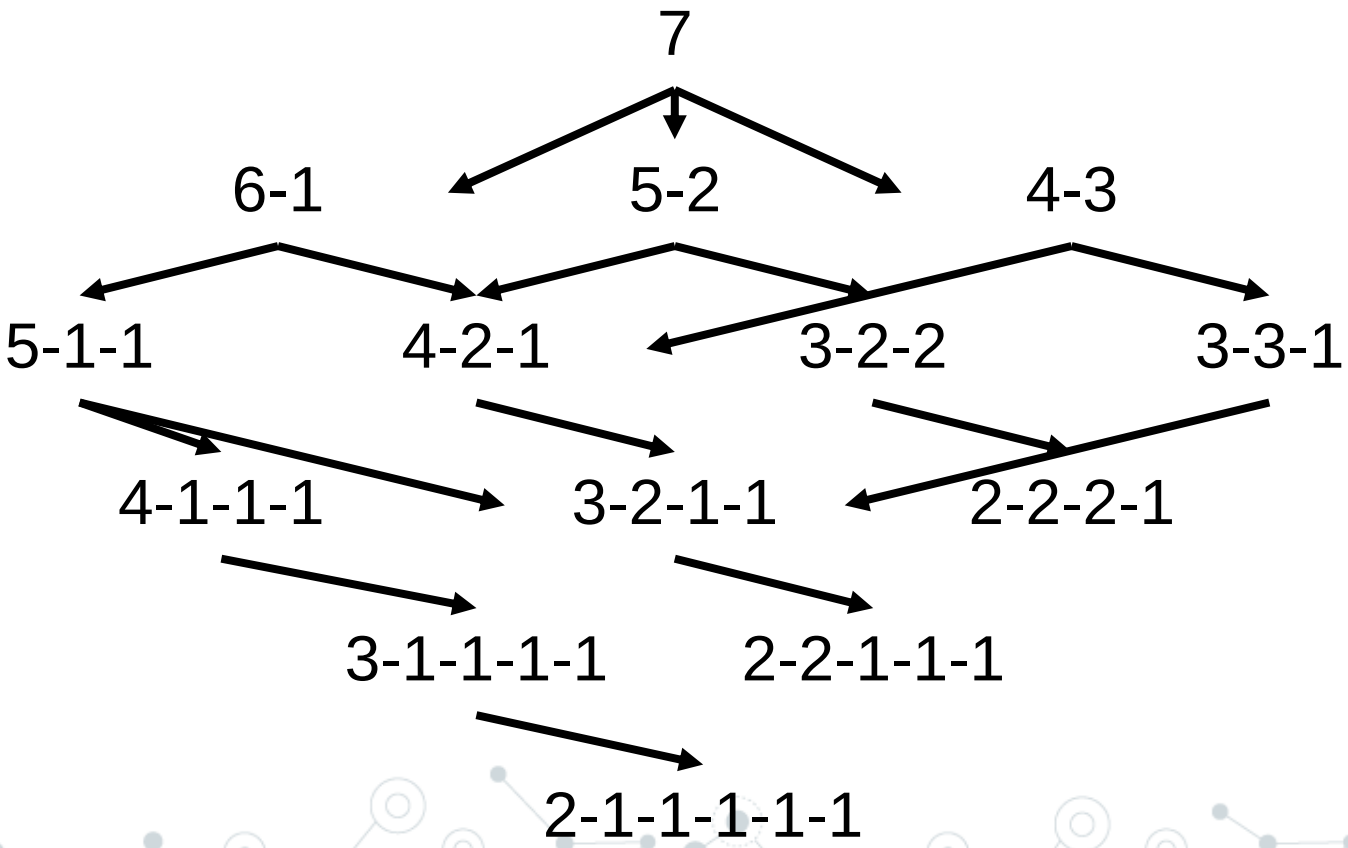
Consider a simplified version of Nim

- Start with a single pile of tokens (7 in this example)
- At each step, a player must select a pile and divide the tokens into two non-empty, non-equal piles
- Game ends when there are only piles of 2 or 1 tokens
- Winner is the player making the last division



Example – Nim Game Graph

How can we use the Game Tree in order to decide how to play?




I looked forward in time, I saw
14,000,605 futures.


How many did we win?

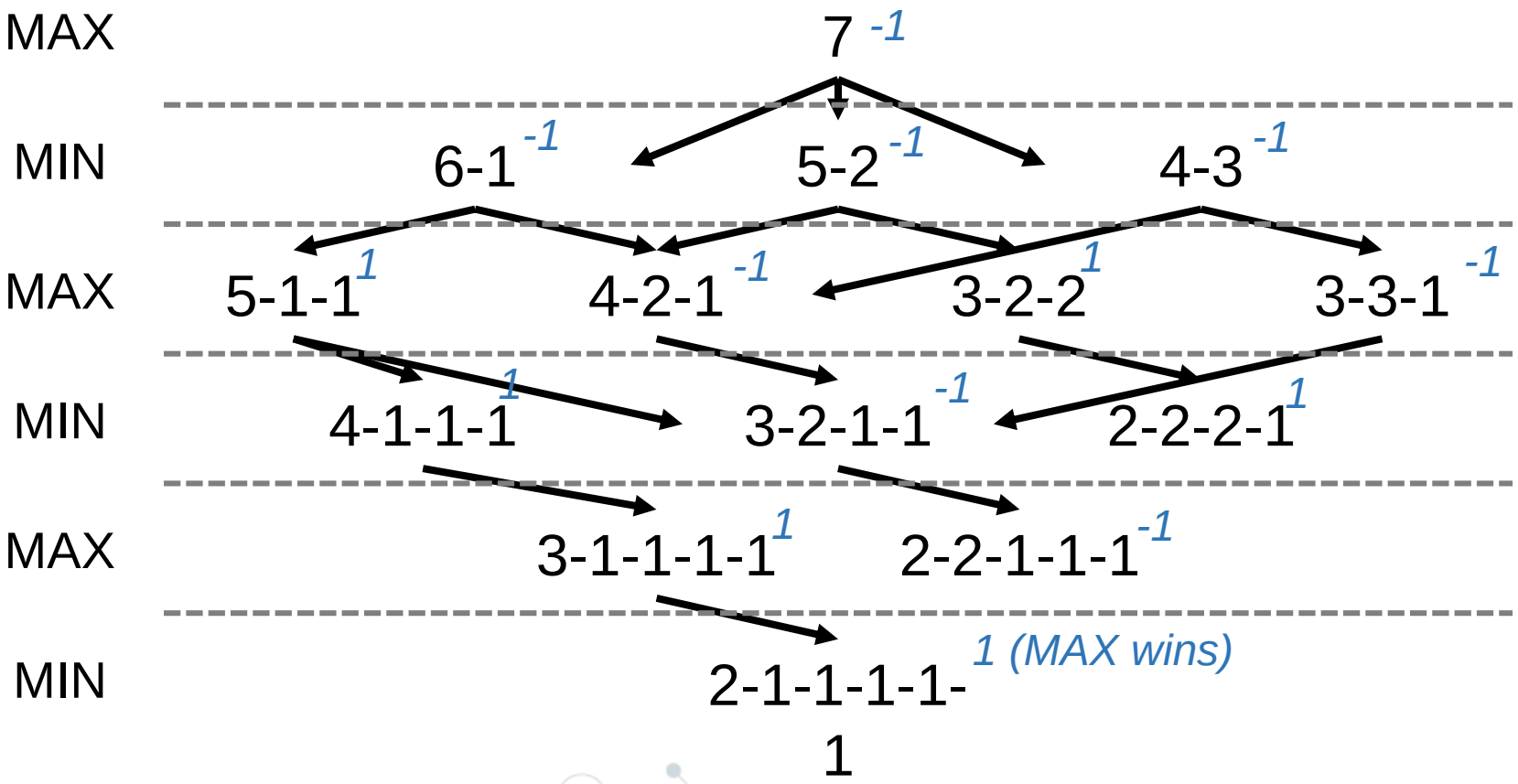

One.

MiniMax Algorithm

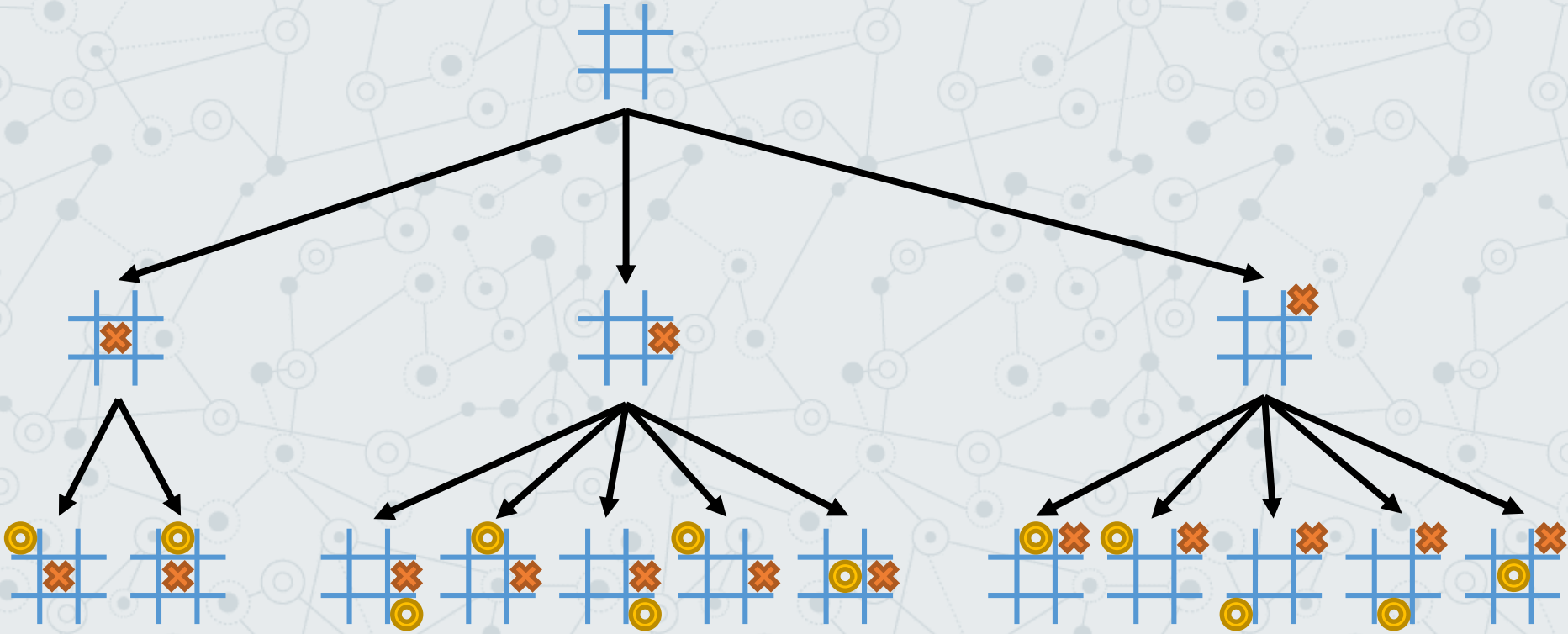
How can we use the Game Tree in order to decide how to play?

- For zero-sum games, the utility of any given state, $MiniMaxVal(v)$ is determined recursively:
  - If v is a terminal,
    - $MiniMaxVal(v) = u(v)$
  - If v is a MAX node,
    - $MiniMaxVal(v) = \max_{s \in Succ(v)} MiniMaxVal(s)$
  - If v is a MIN node,
    - $MiniMaxVal(v) = \min_{s \in Succ(v)} MiniMaxVal(s)$

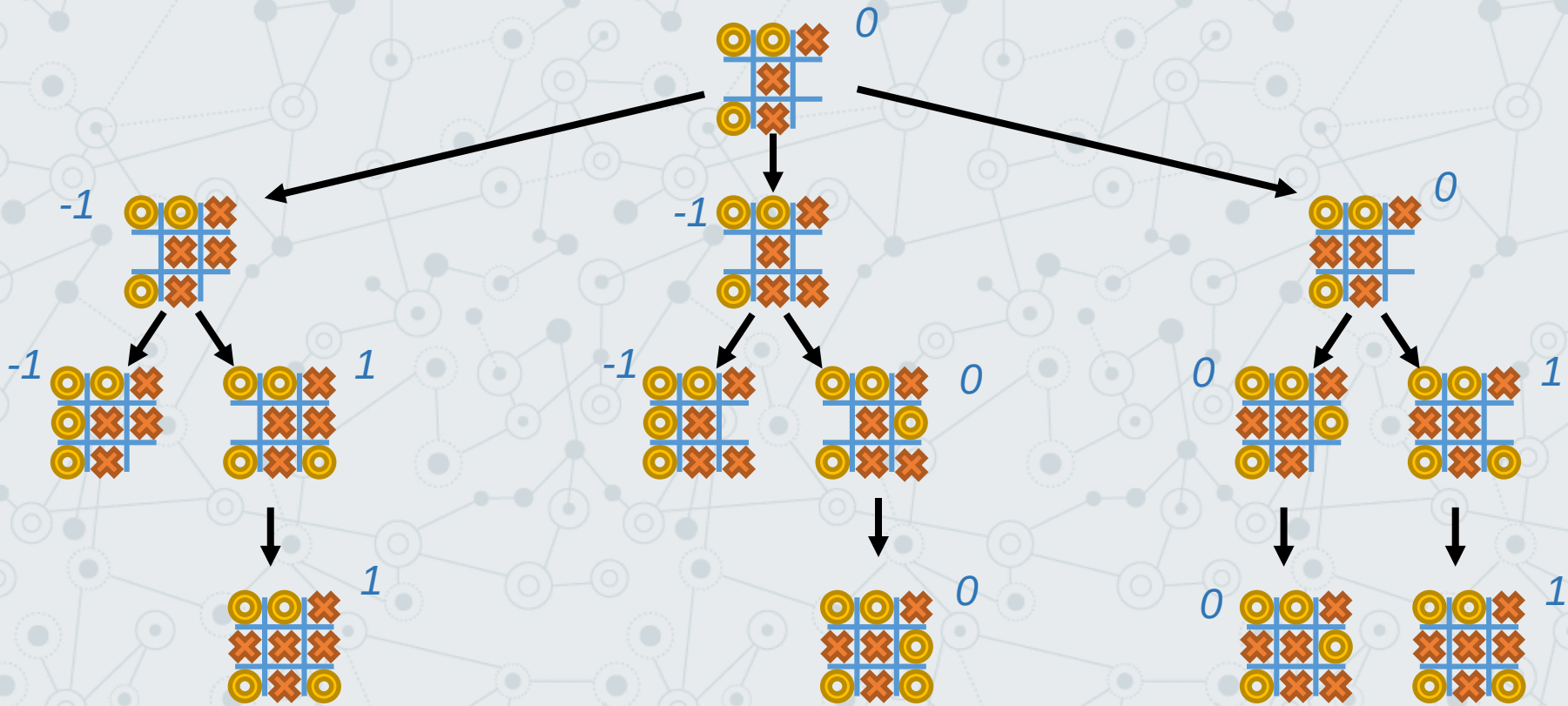
Example – Nim Game Tree



Example – Tic Tac Toe



Example – Tic Tac Toe



MiniMax Properties

- **Complete** (Only if tree is finite)
- **Optimal** (Only against an optimal opponent)
- **Time complexity** $O(b^m)$
- **Space complexity**
  - $O(b^m)$ for BFS ,
  - $O(b \cdot m)$ for DFS

The Need for Heuristics

Consider the game of **Chess**:

- Size of search space (10^{120} – average of 40 moves)
- $10^{120} >$ number of atoms in the universe

- 200 million positions/second = 10^{100} years to evaluate all possible games
- Age of universe = 10^{10}

Heuristics in Real Games

- In real games, search trees are bigger and deeper than Nim
- Cannot possibly evaluate the entire tree
- Have to put a bound on the depth of the search
- The terminal states are no longer a definite win/loss – we will approximate states quality using **heuristics**

Tic Tac Tow Heuristic

$h(v) =$ number of rows, columns, and diagonals open for MAX - number of rows, columns, and diagonals open for MIN

$h(v) = 8 - 8 = 0$

$h(v) = 6 - 4 = 2$

$h(v) = 3 - 3 = 0$

Chess Heuristic

For chess, typically linear weighted sum of features

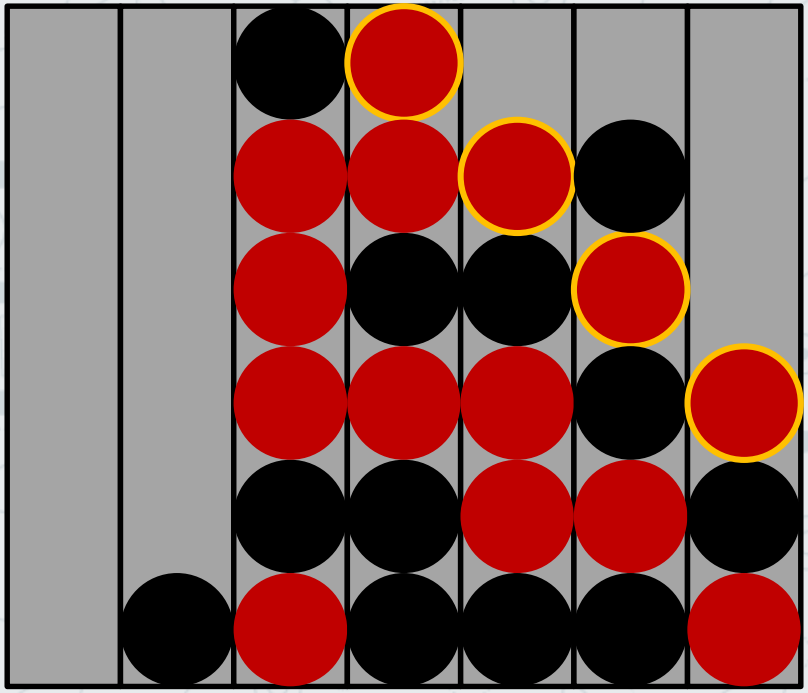
$$h(v) = w_1 \cdot f_1(v) + \cdots + w_n \cdot f_n(v)$$

e.g.

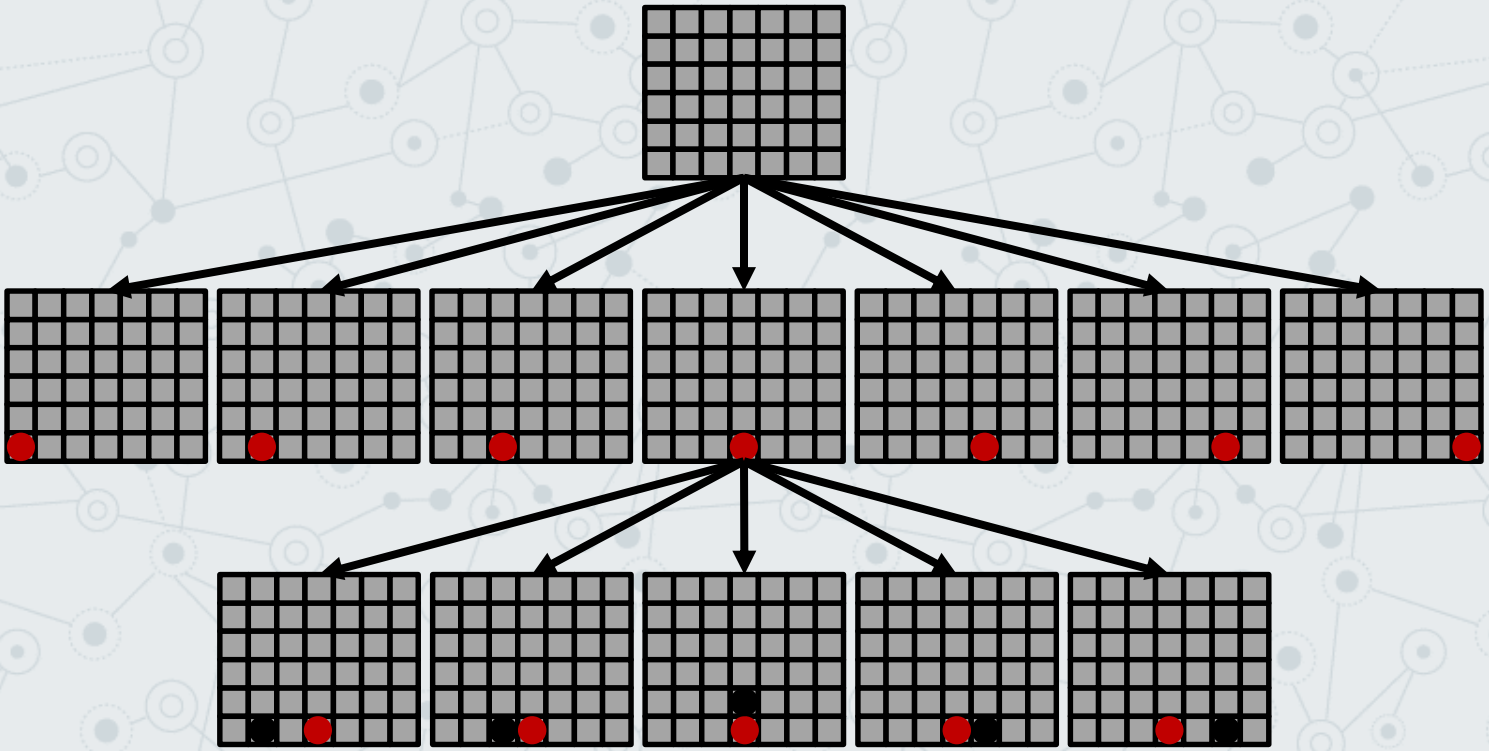
$w_1 = 8$ with

$f_1(v) =$ (number of white queens) - (number of black queens)

Connect 4



Connect 4 – Game Tree

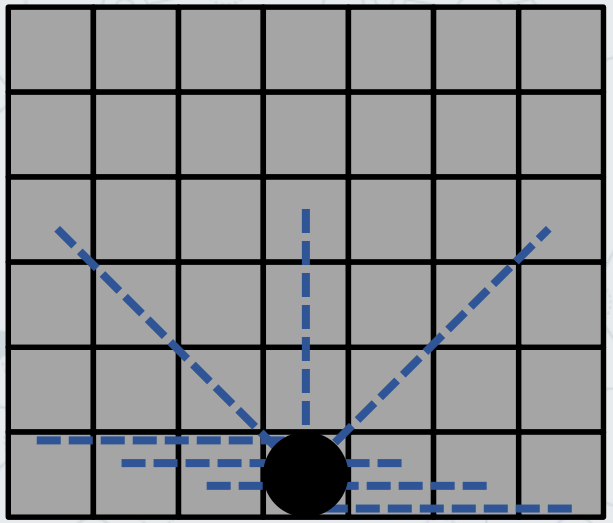


Connect 4 (Simple) Heuristic

Number of possible winning lines

e.g. horizontally: 4, vertically: 1, diagonally: 2

Total heuristic score = 7



Connect 4 (IBEF) Heuristic

Intuituve Board Evaluation Function (IBEF):

The Connect4 board is a 7 × 6 grid. There are 69 unique possible winning "lines" on the board (24 horizontal, 21 vertical, 24 diagonal)

Given a specific board configuration:

for each of these 69 lines

for each of the four squares in the line

if square occupied by the max, its valued 1

if square occupied by the min, its valued -1

otherwise, its valued 0

The value of each line is the sum of the values of its 4 individual squares

The value of the board is the sum of the values of the 69 potential winning lines

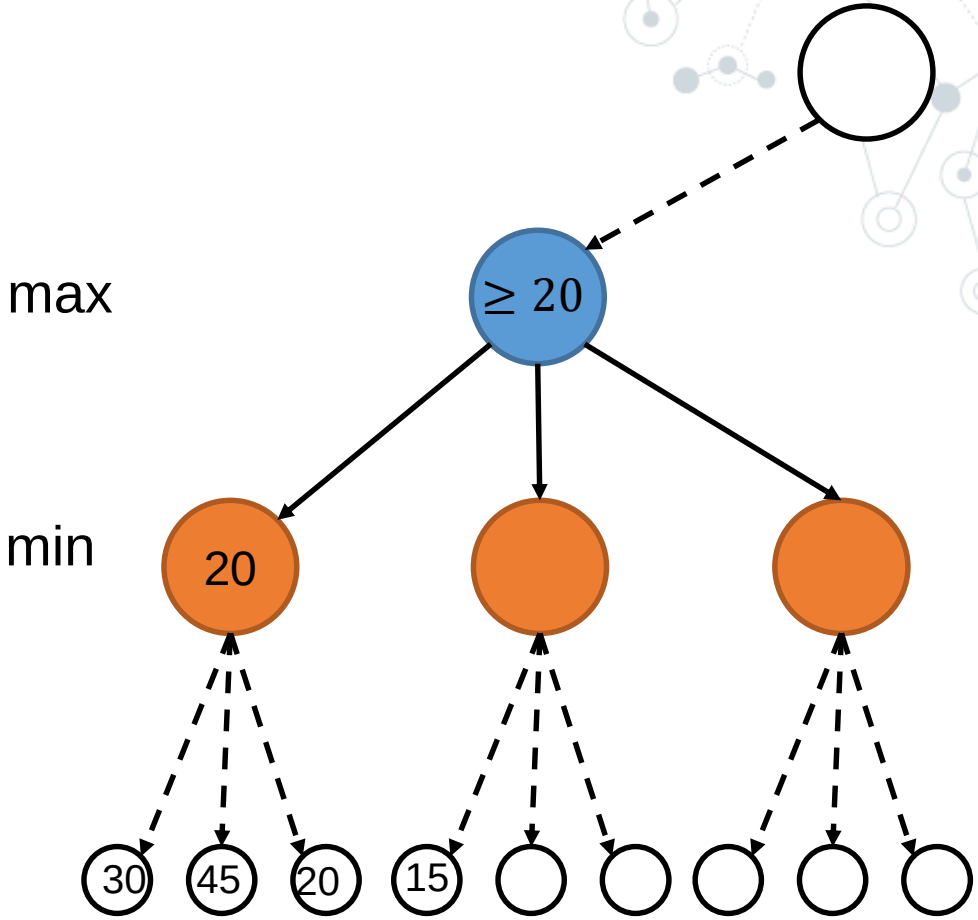
Connect 4 (IBEF) Heuristic

In other words, each square is weighted according to the number of unique winning lines it can be a member of

3	4	5	7	5	4	3
4	6	8	10	8	6	4
5	8	11	13	11	8	5
5	8	11	13	11	8	5
4	6	8	10	8	6	4
3	4	5	7	5	4	3

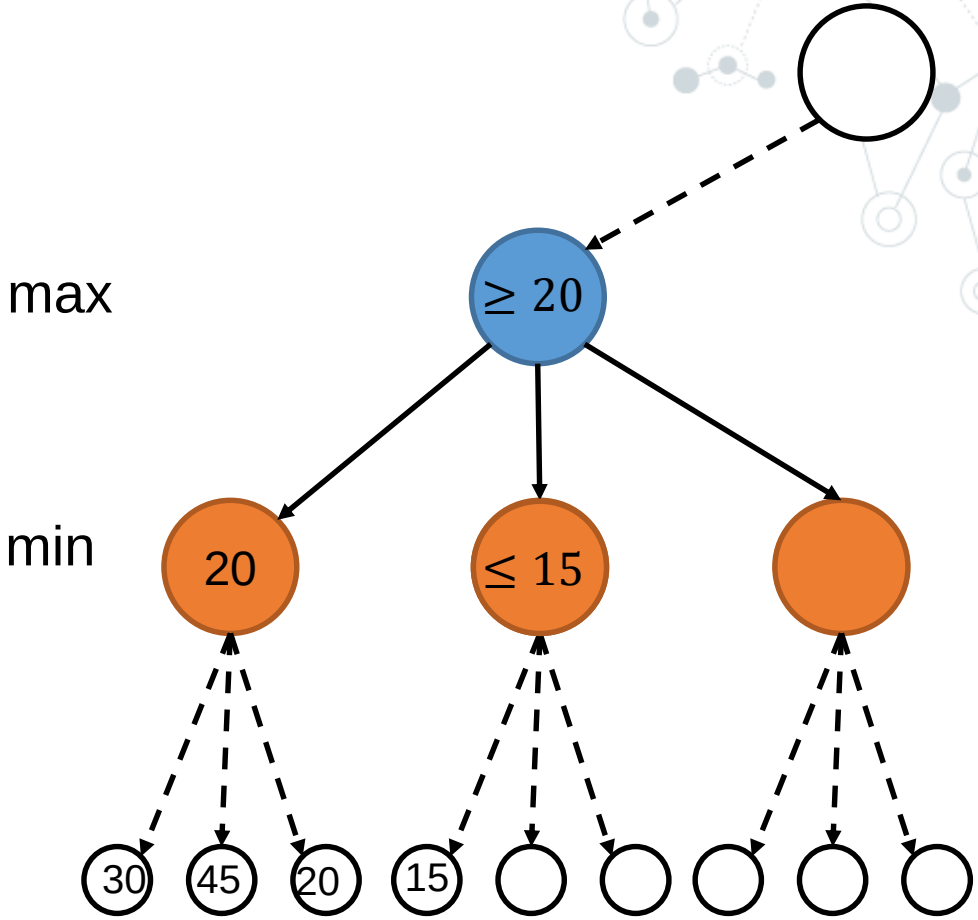
Alpha-Beta Pruning Idea

- Game Trees might be very big
- Reduce nodes expanded by pruning a branch if there cannot be any rational sequence of play that would go through it

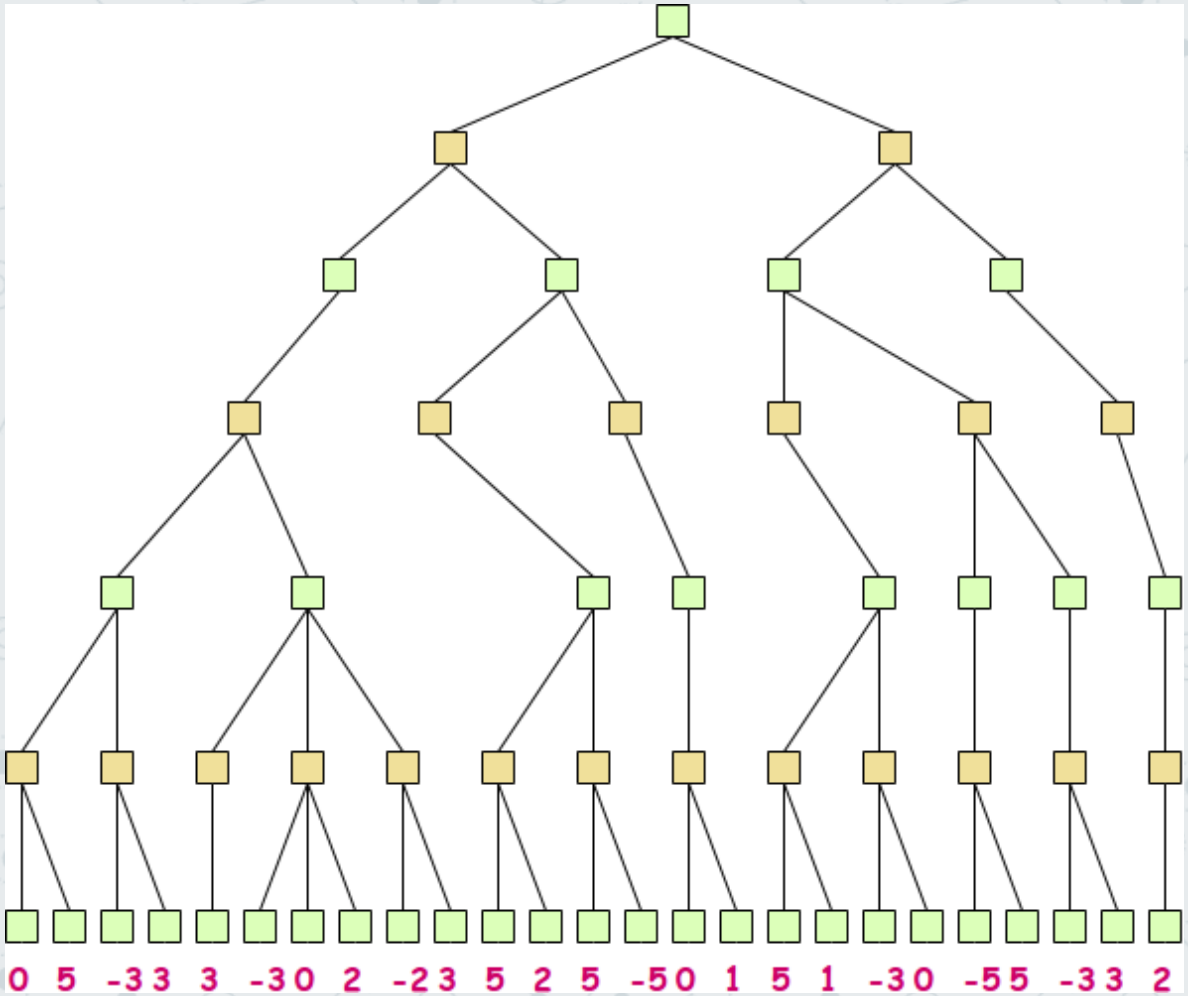


Alpha-Beta Pruning Idea

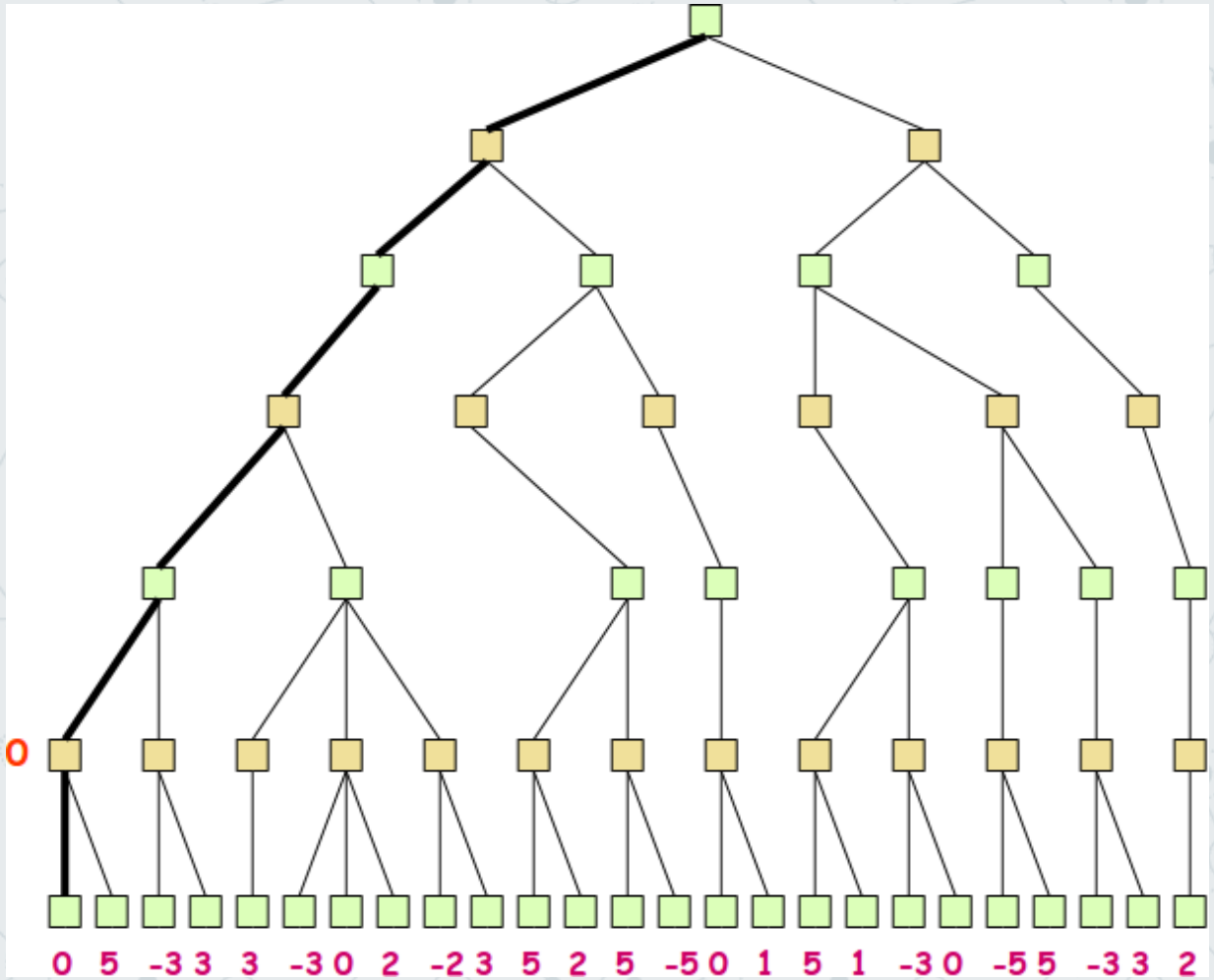
- Game Trees might be very big
- Reduce nodes expanded by pruning a branch if there cannot be any rational sequence of play that would go through it



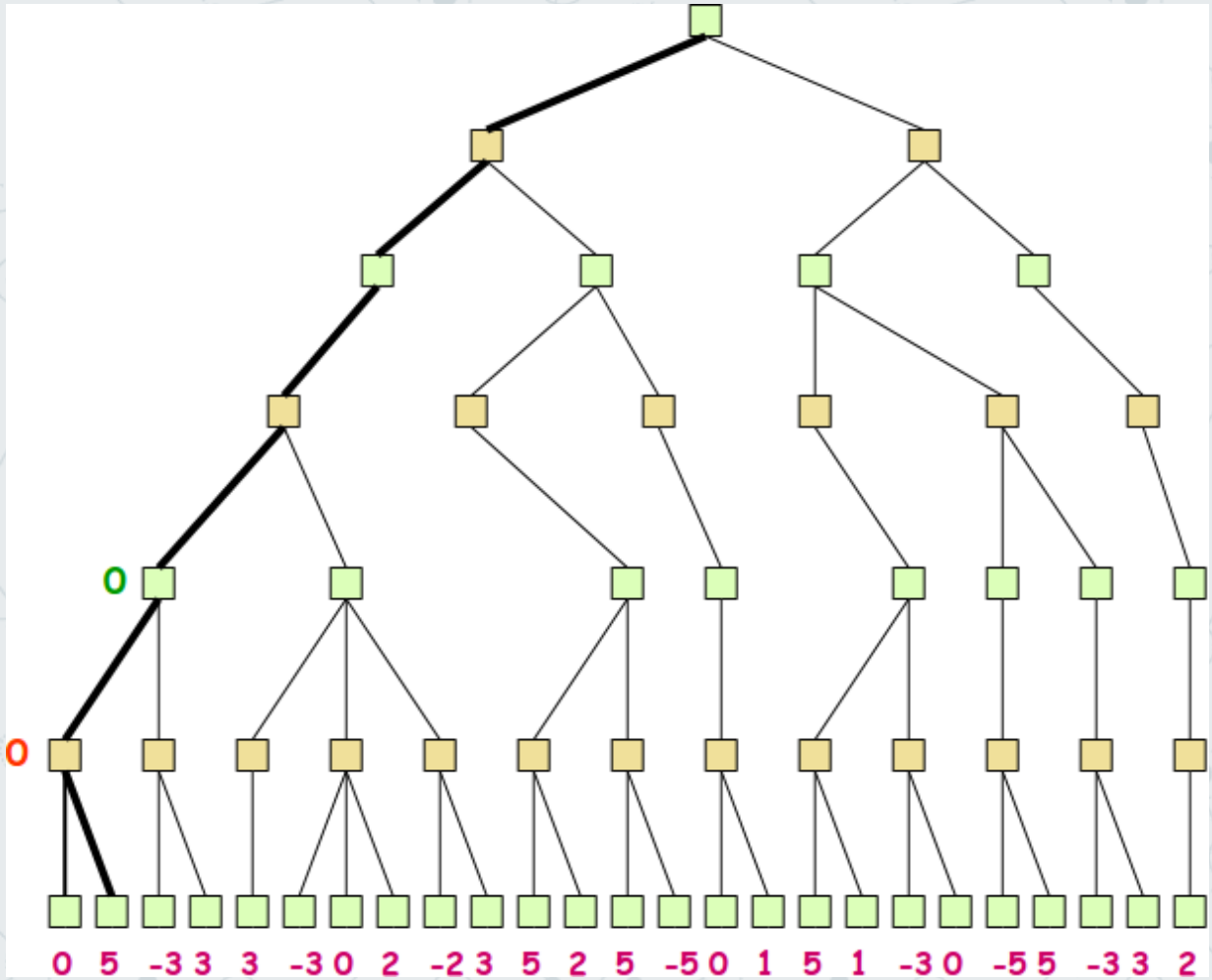
Alpha-Beta Pruning Example



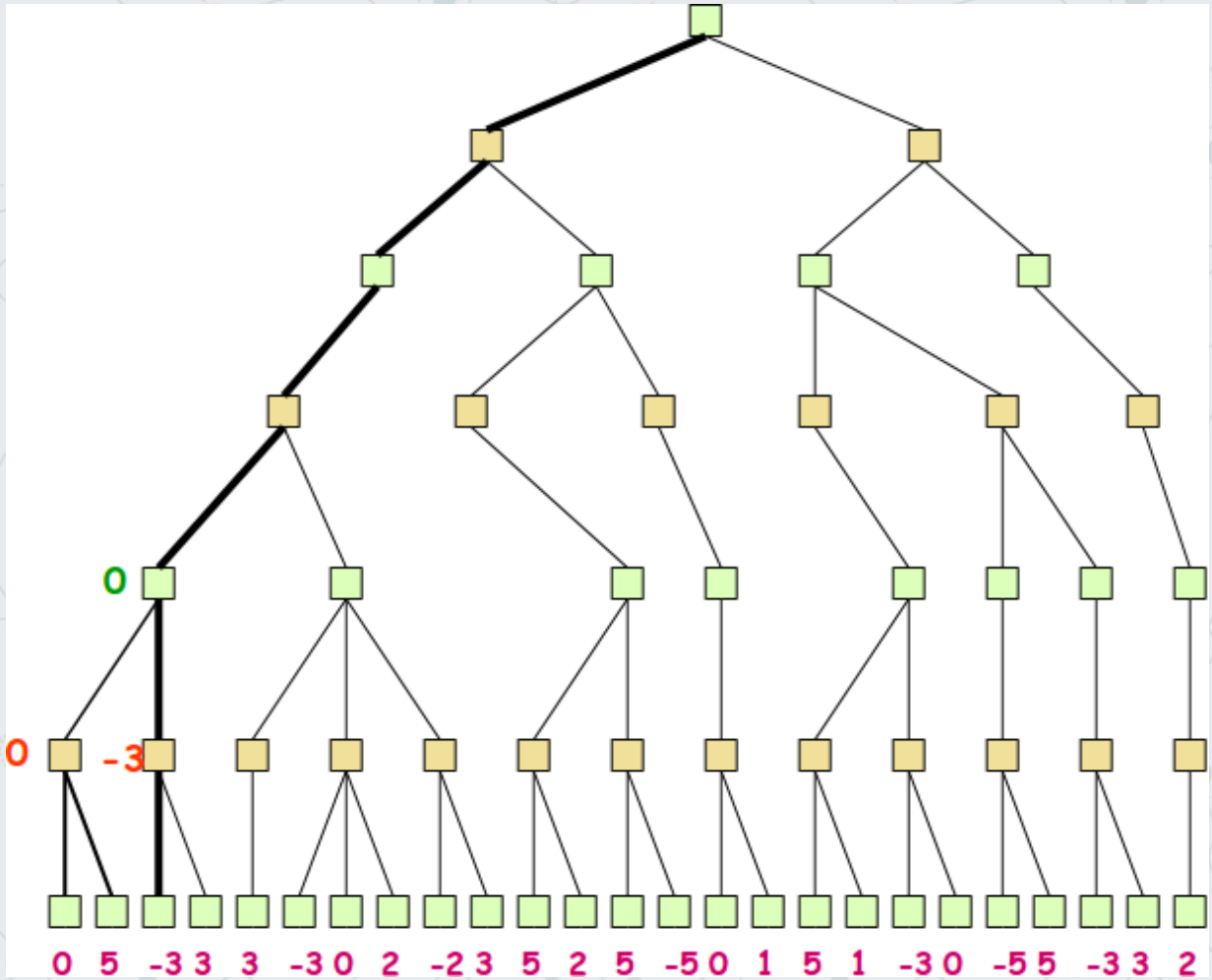
Alpha-Beta Pruning Example



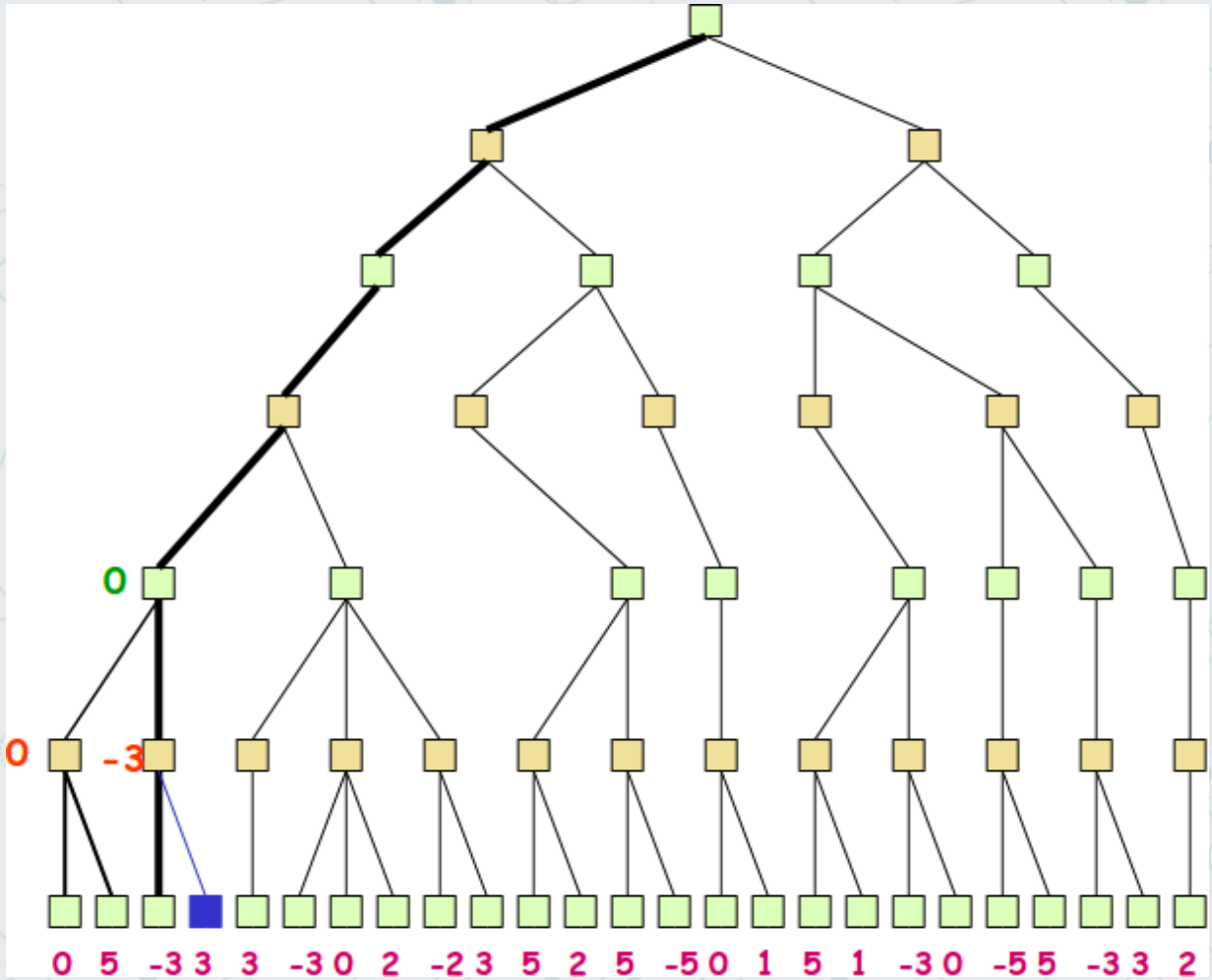
Alpha-Beta Pruning Example



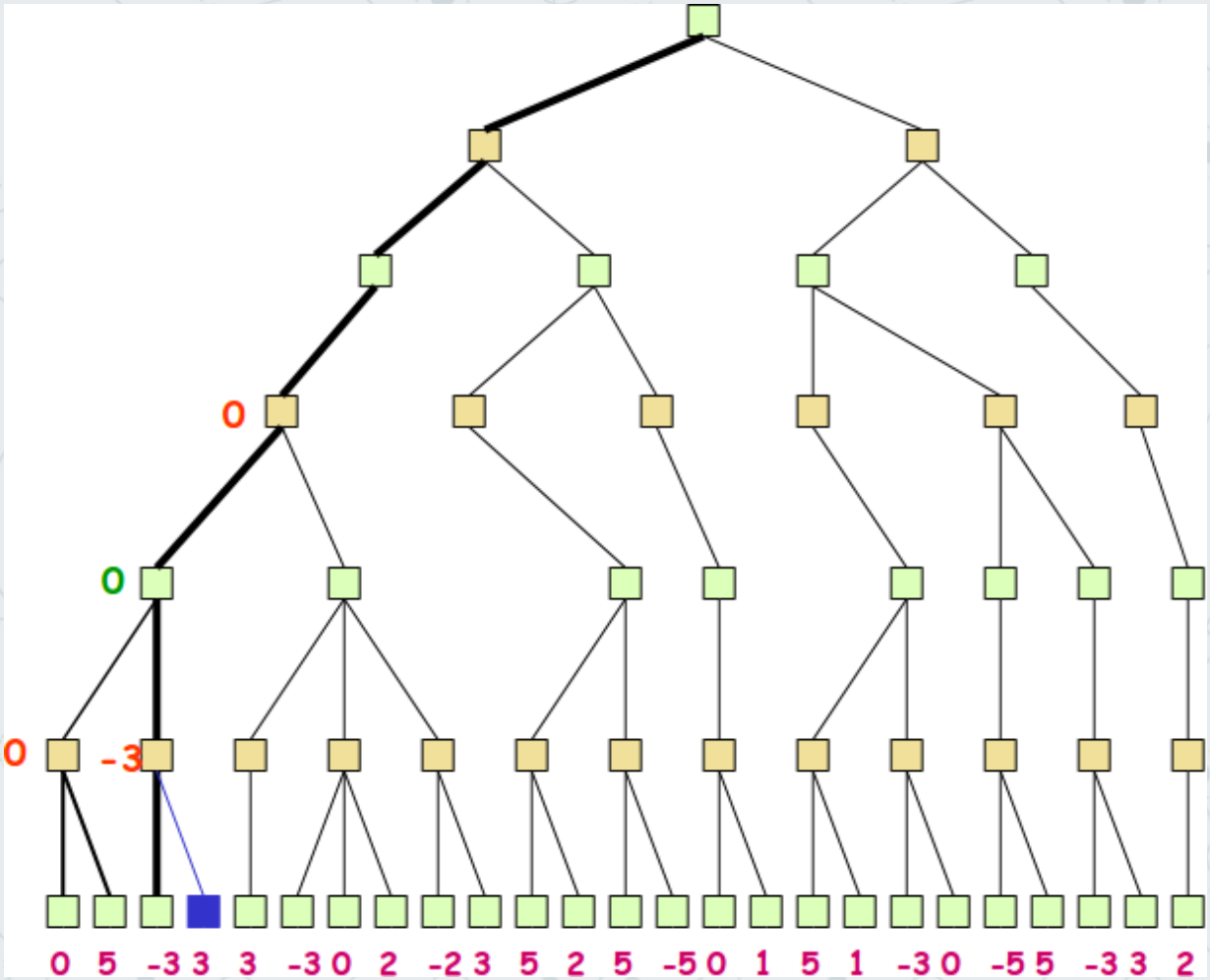
Alpha-Beta Pruning Example



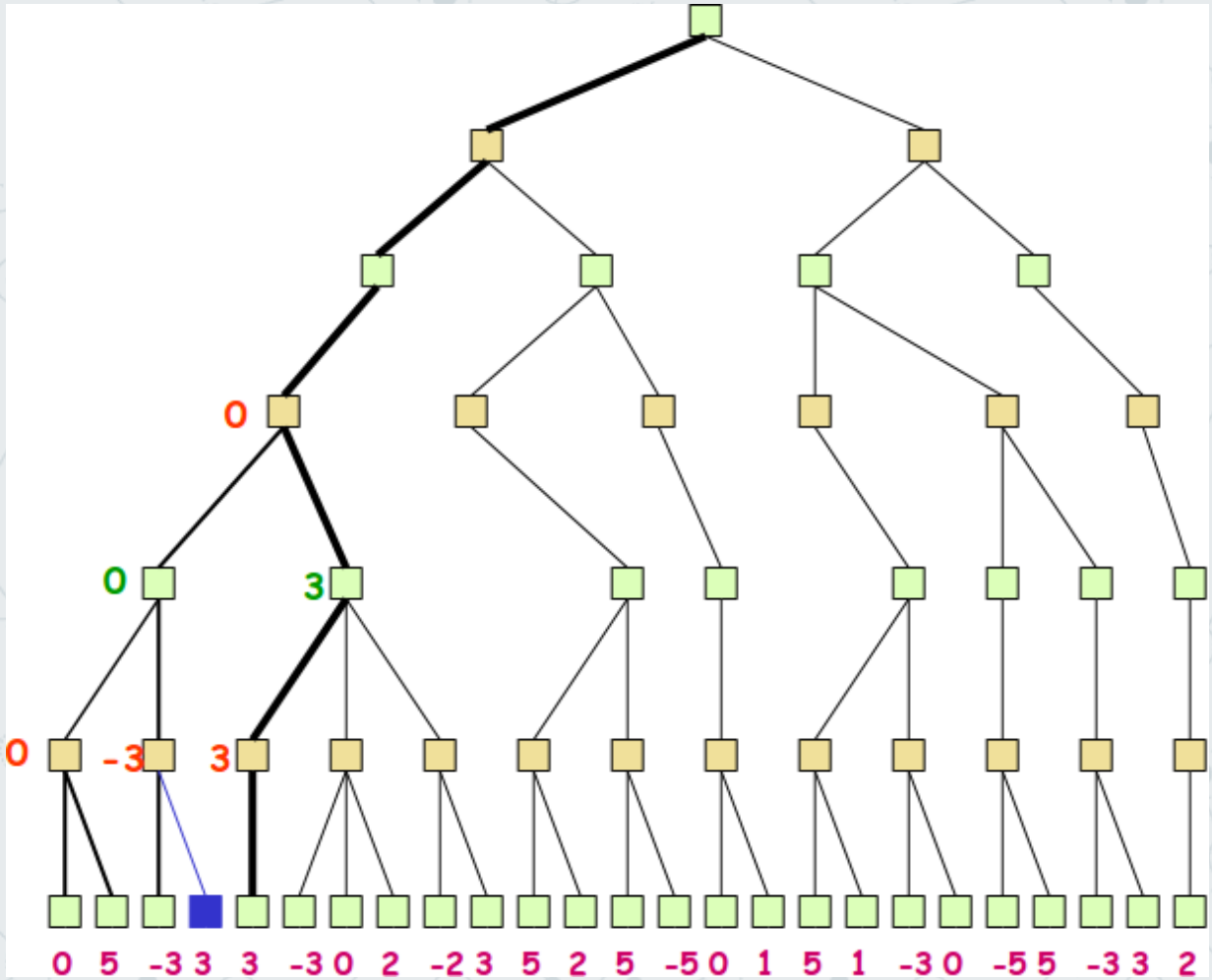
Alpha-Beta Pruning Example



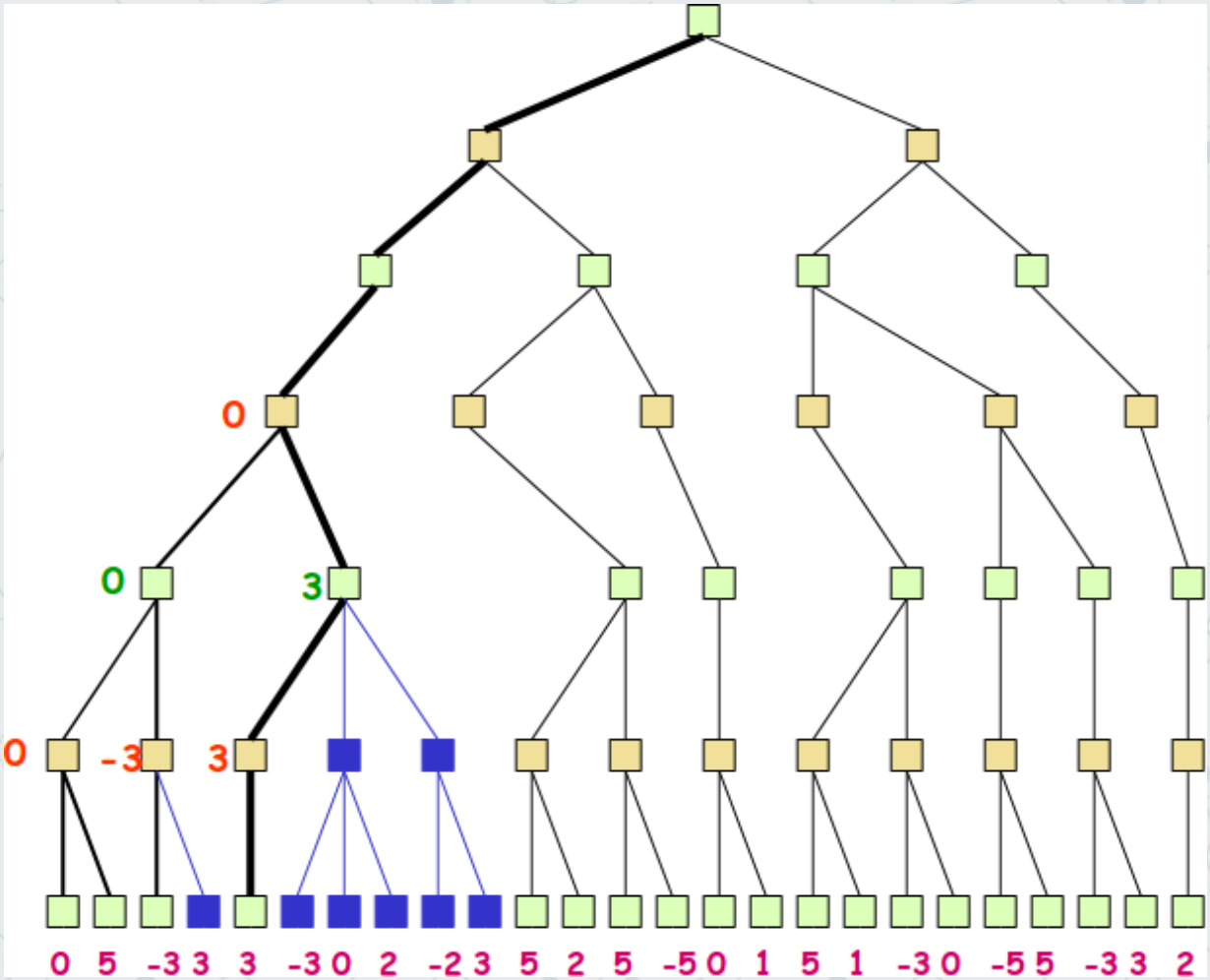
Alpha-Beta Pruning Example



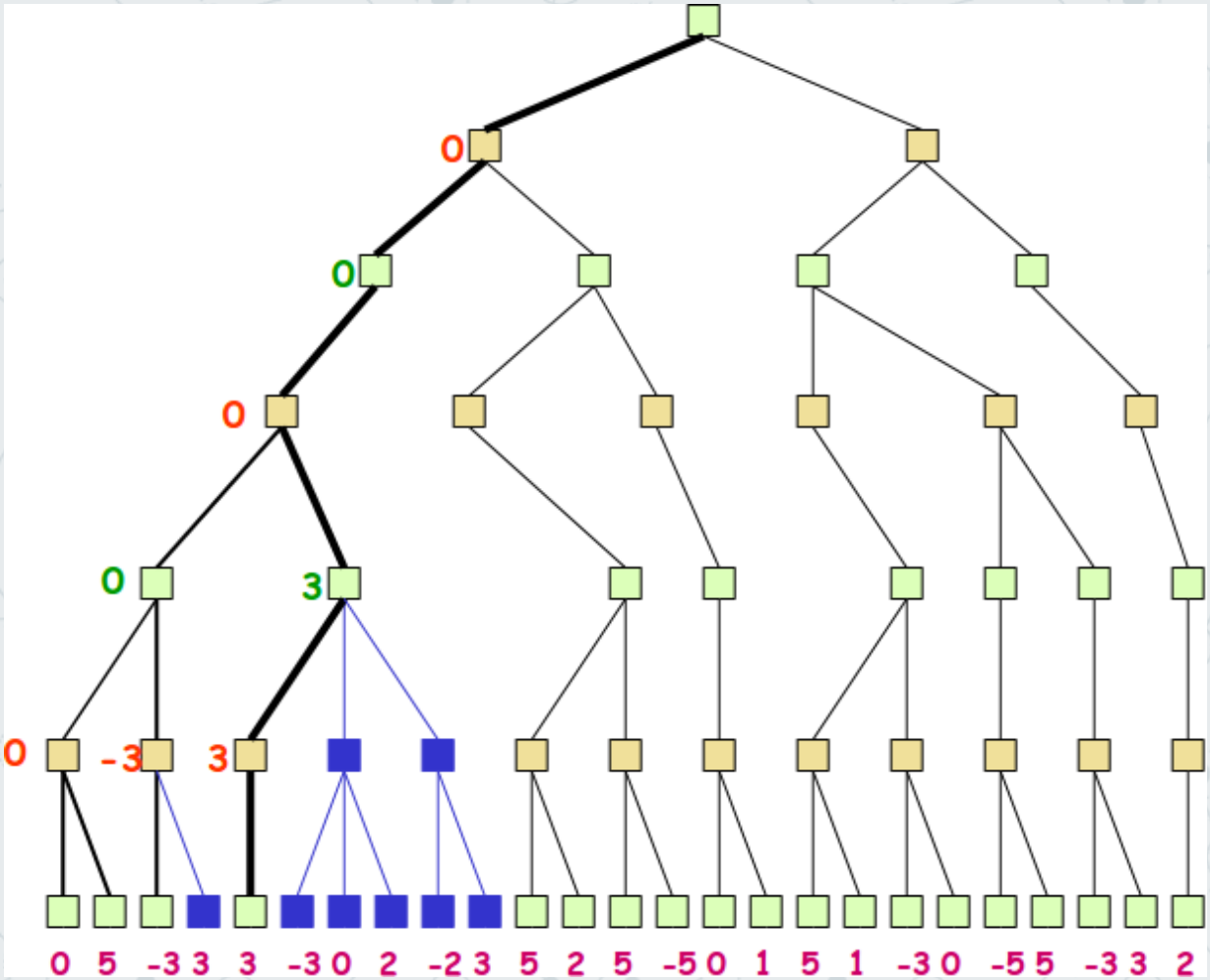
Alpha-Beta Pruning Example



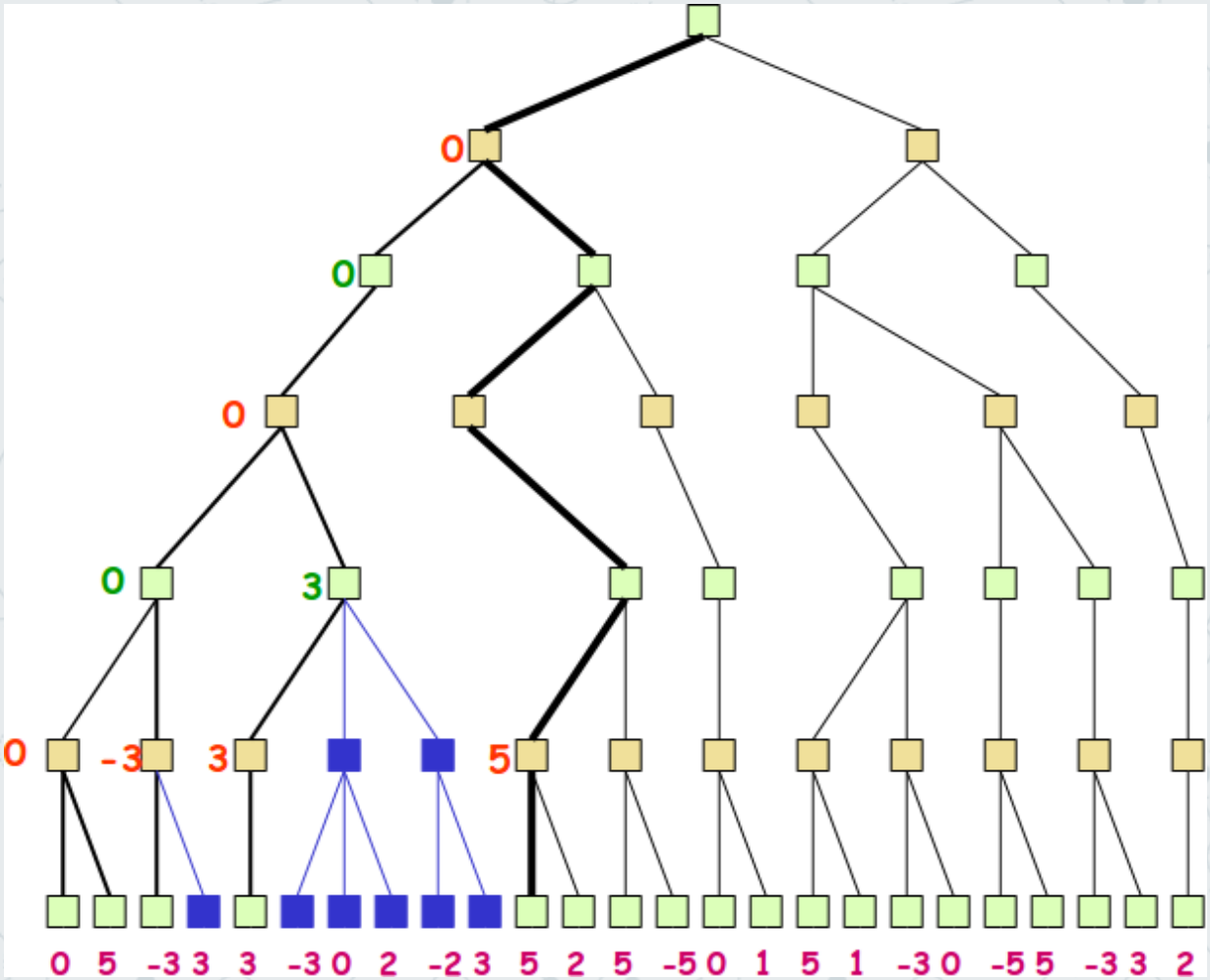
Alpha-Beta Pruning Example



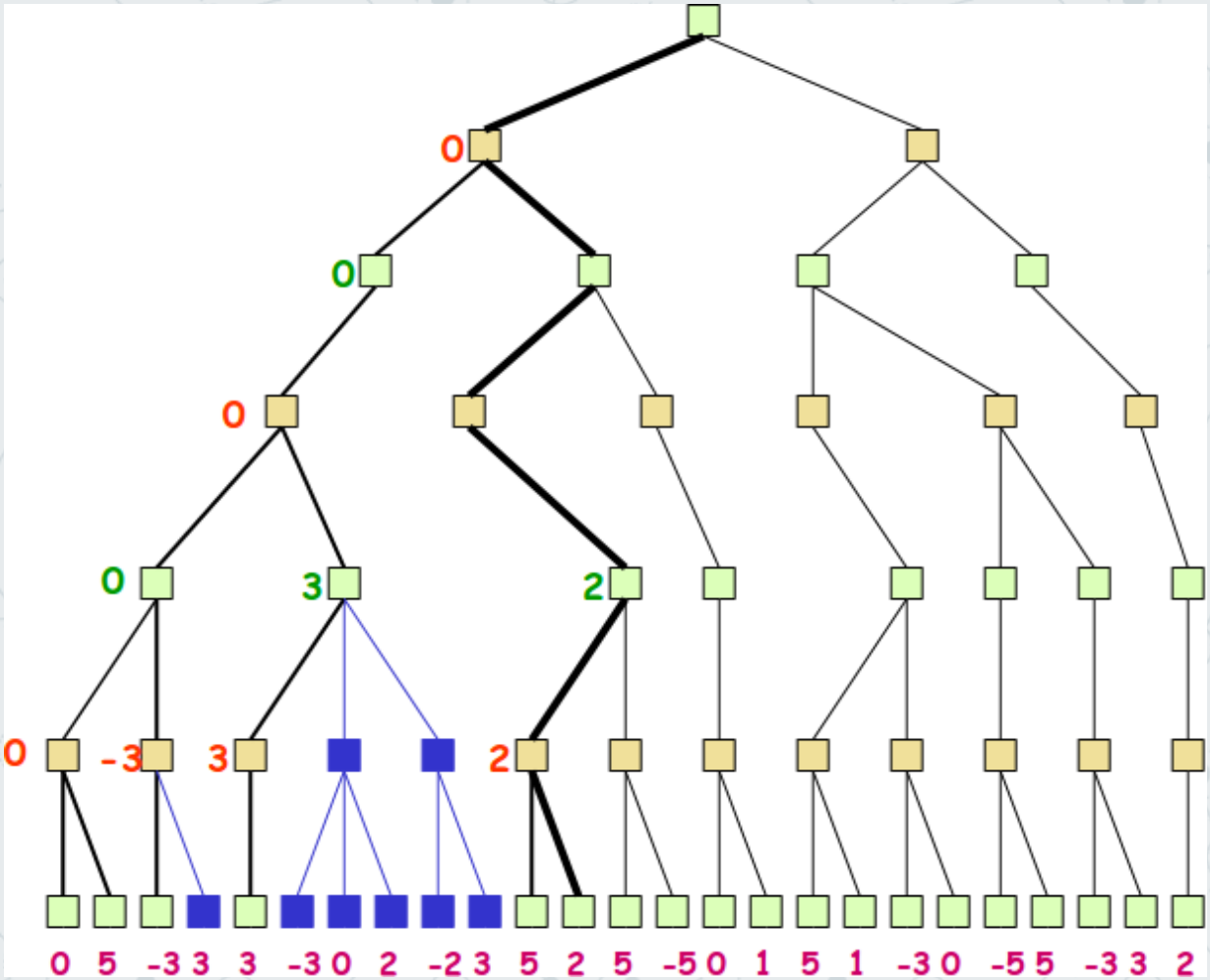
Alpha-Beta Pruning Example



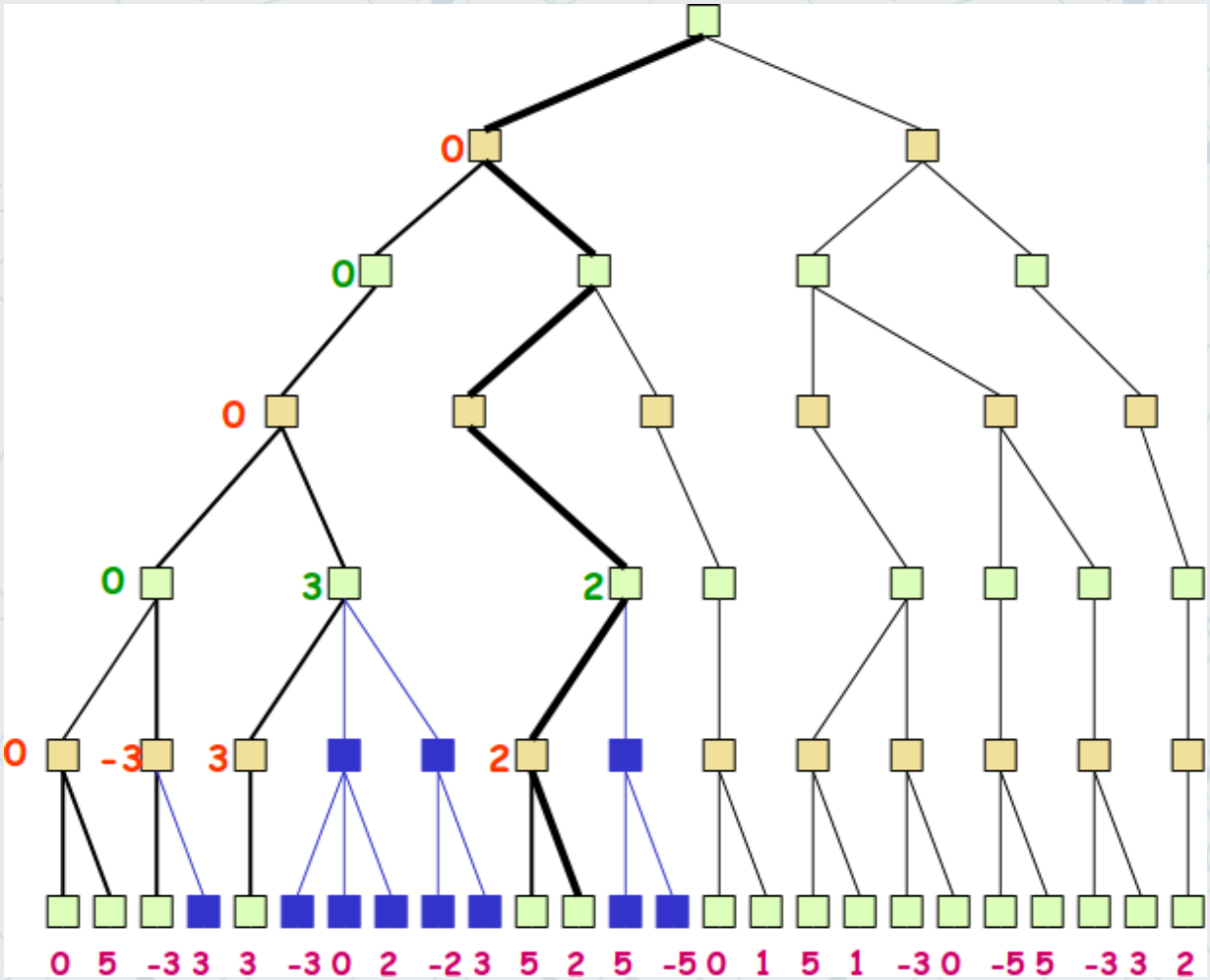
Alpha-Beta Pruning Example



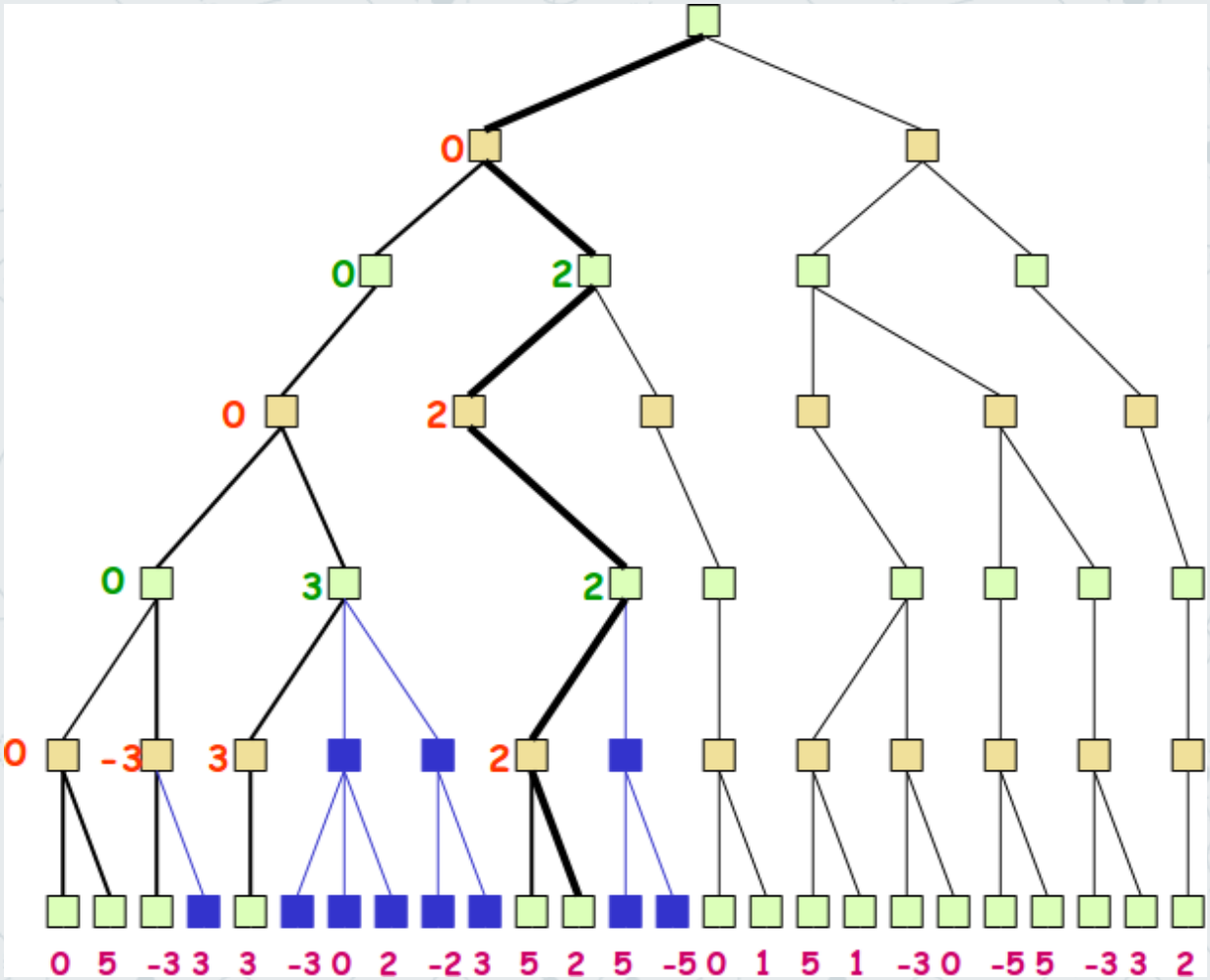
Alpha-Beta Pruning Example



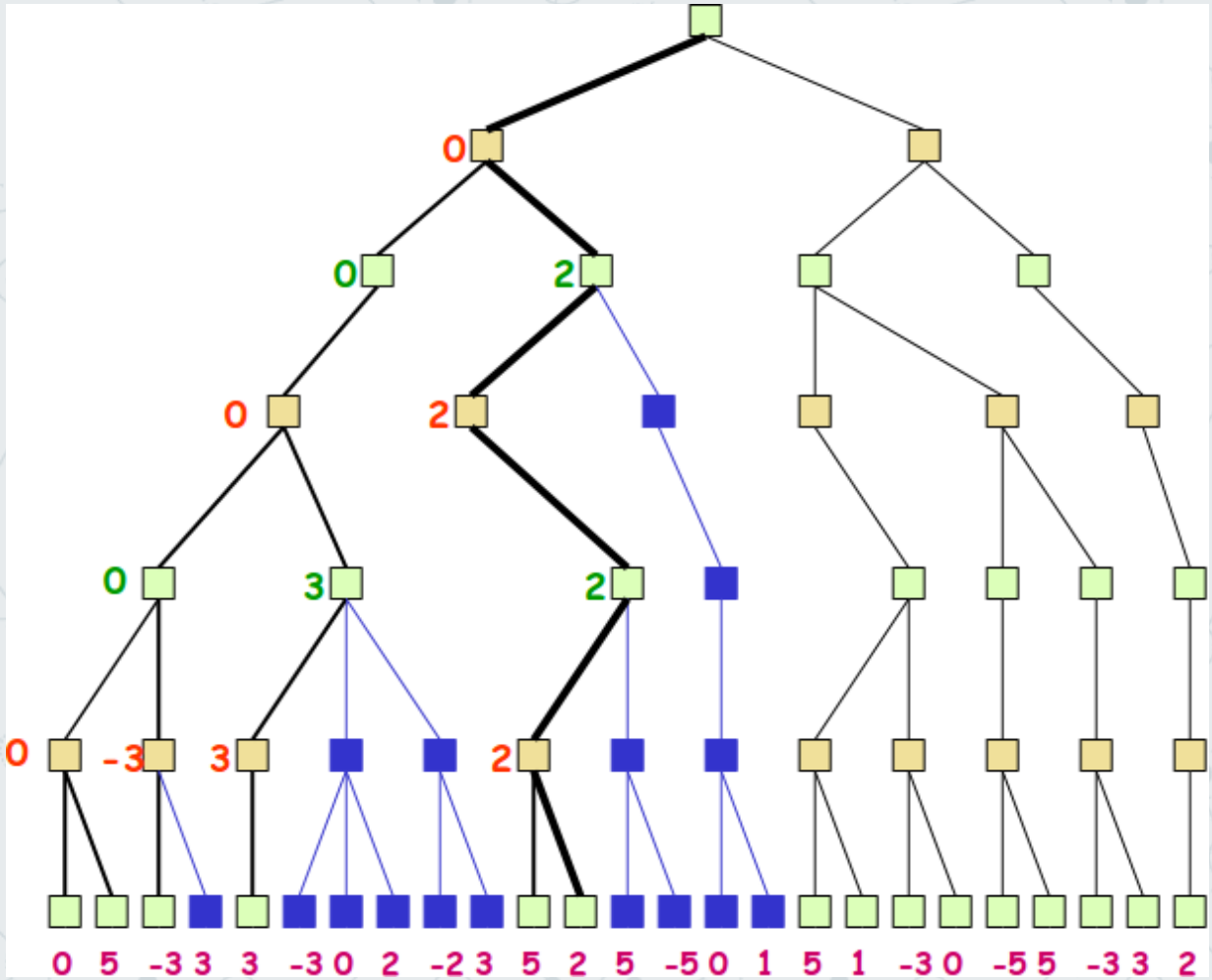
Alpha-Beta Pruning Example



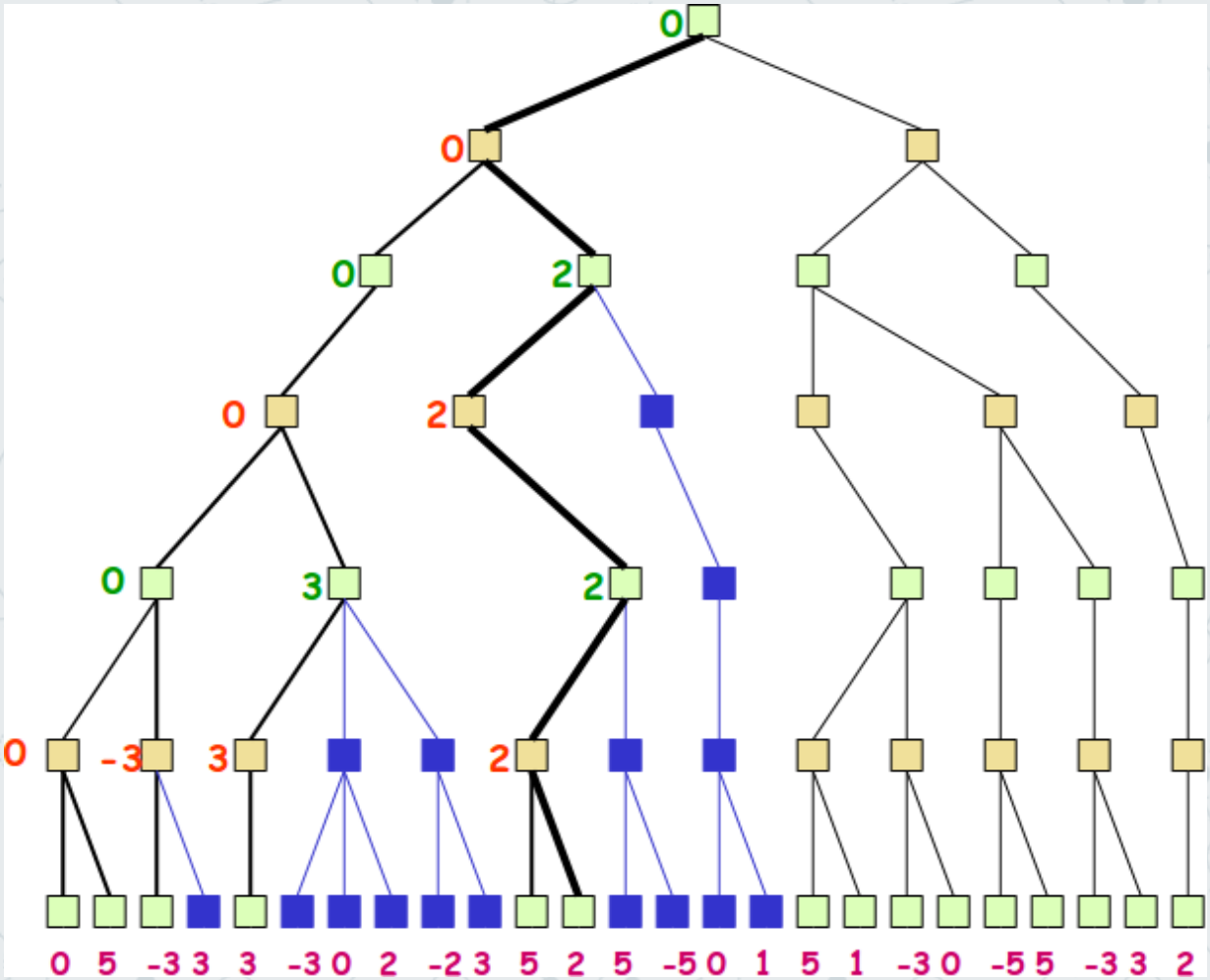
Alpha-Beta Pruning Example



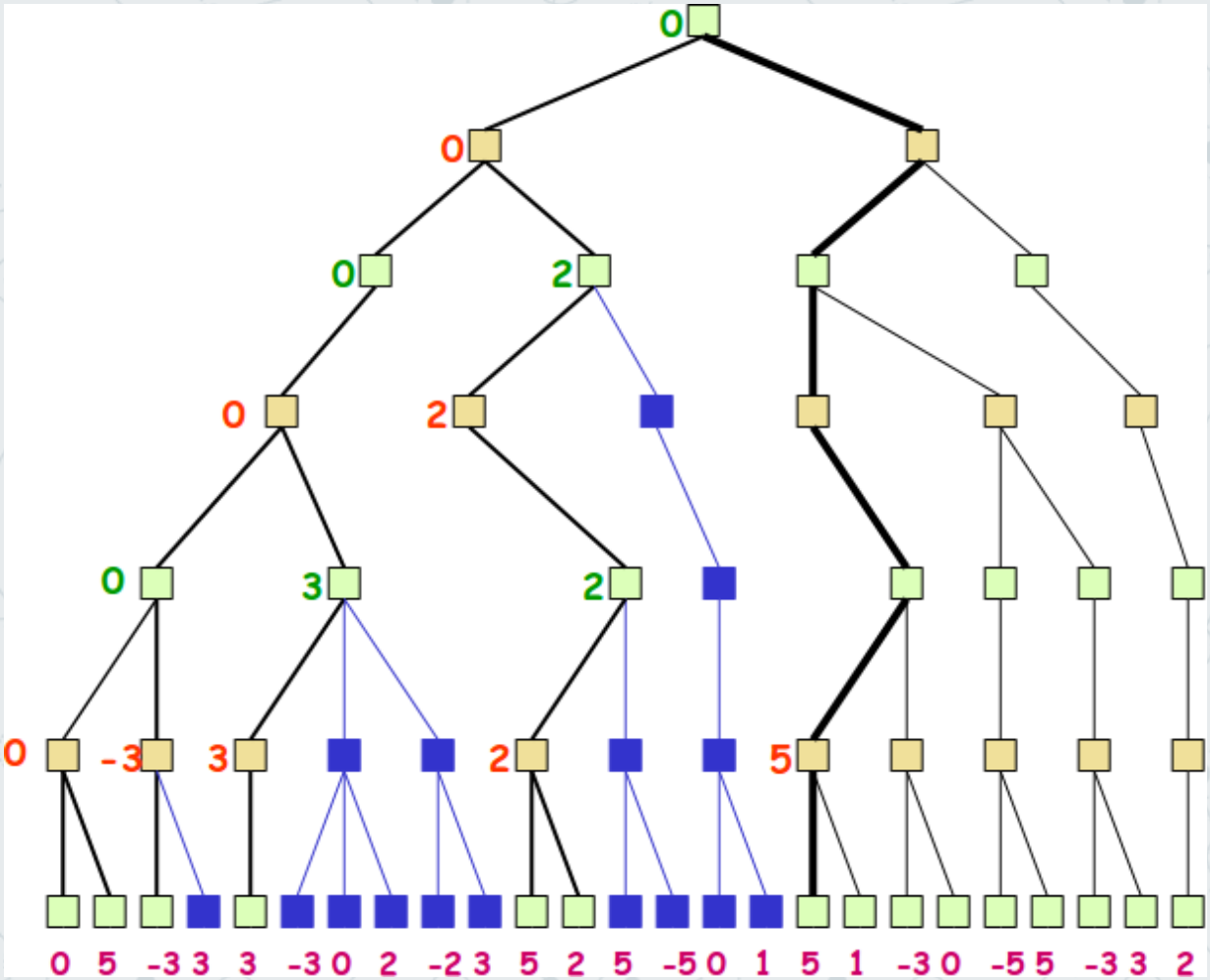
Alpha-Beta Pruning Example



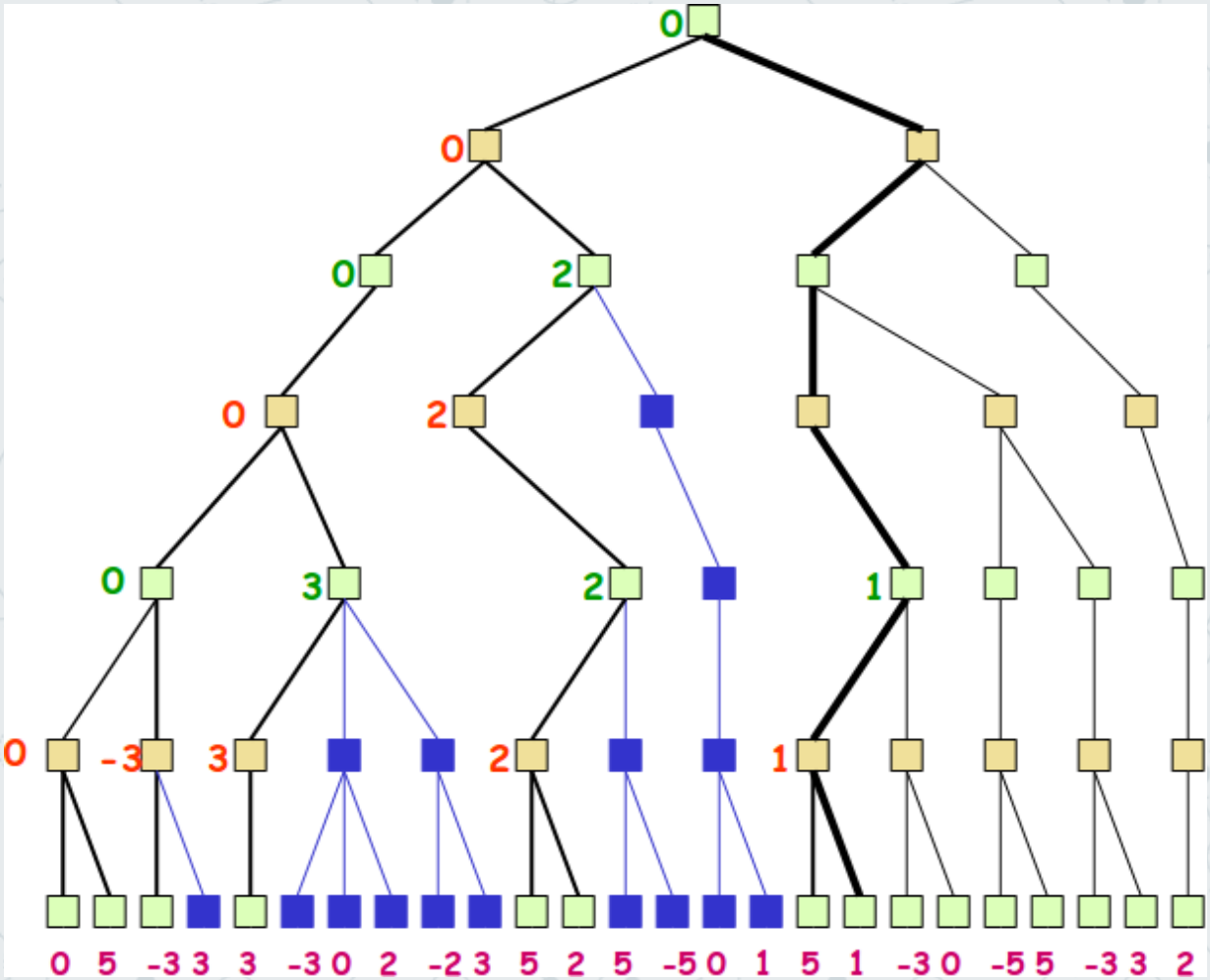
Alpha-Beta Pruning Example



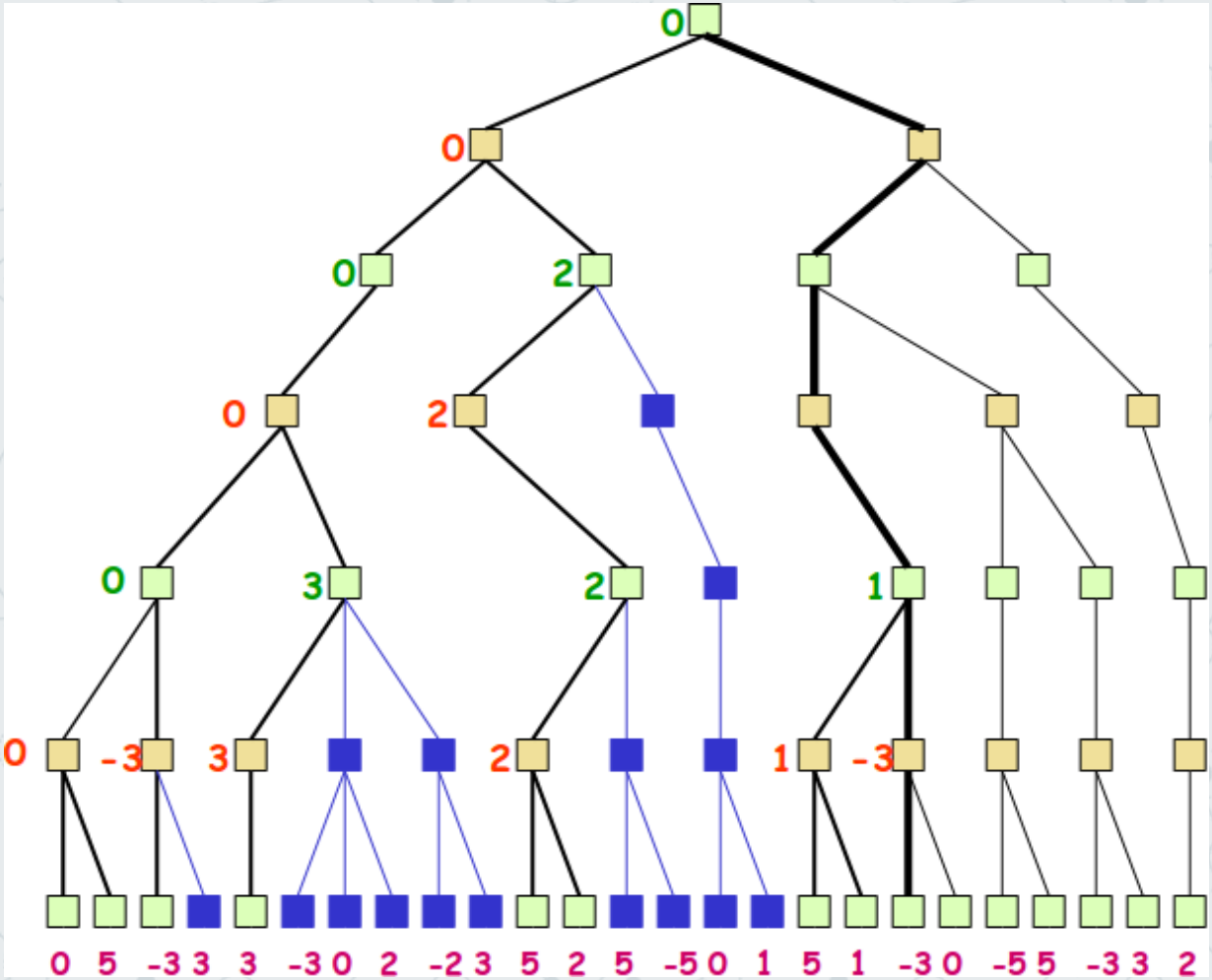
Alpha-Beta Pruning Example



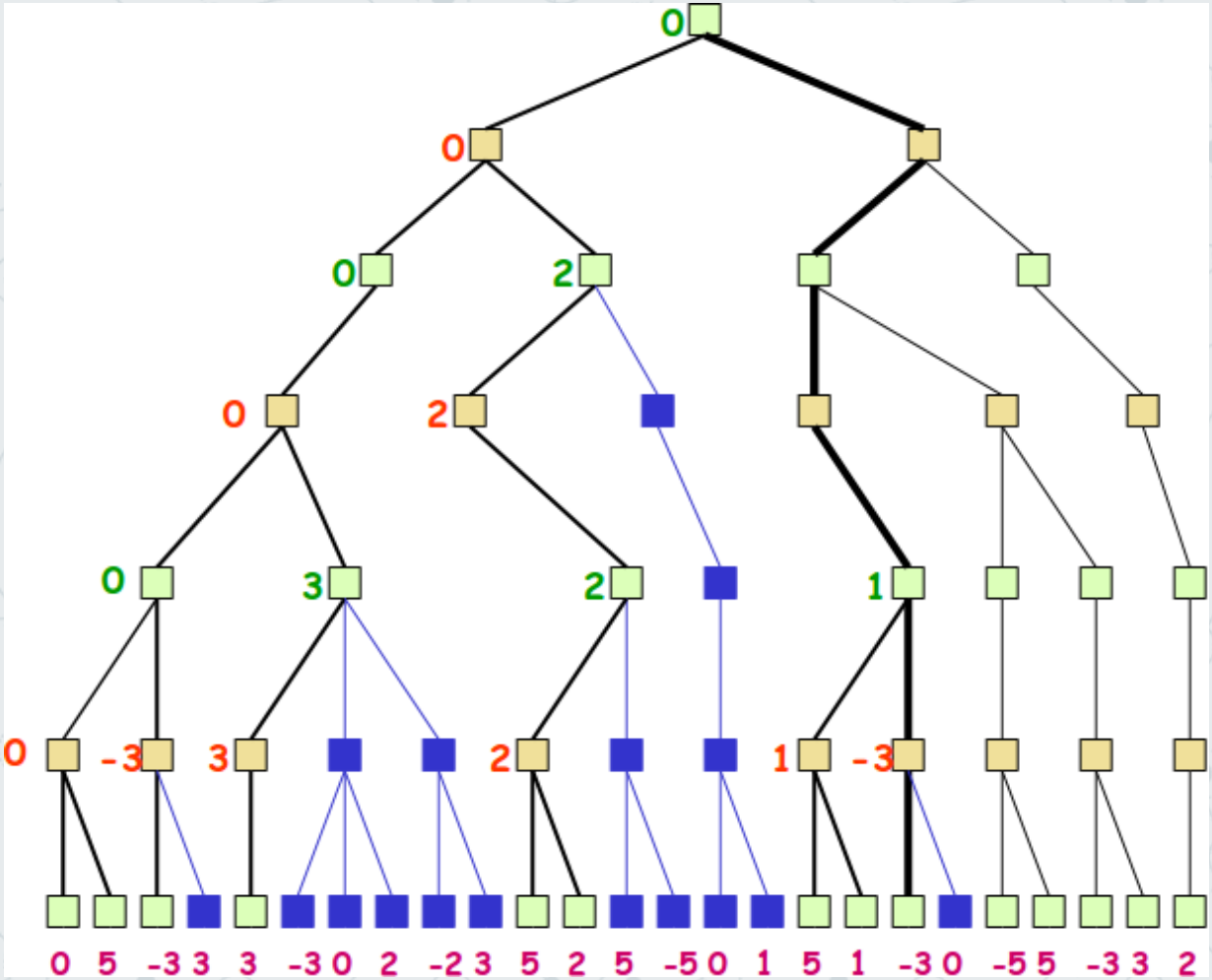
Alpha-Beta Pruning Example



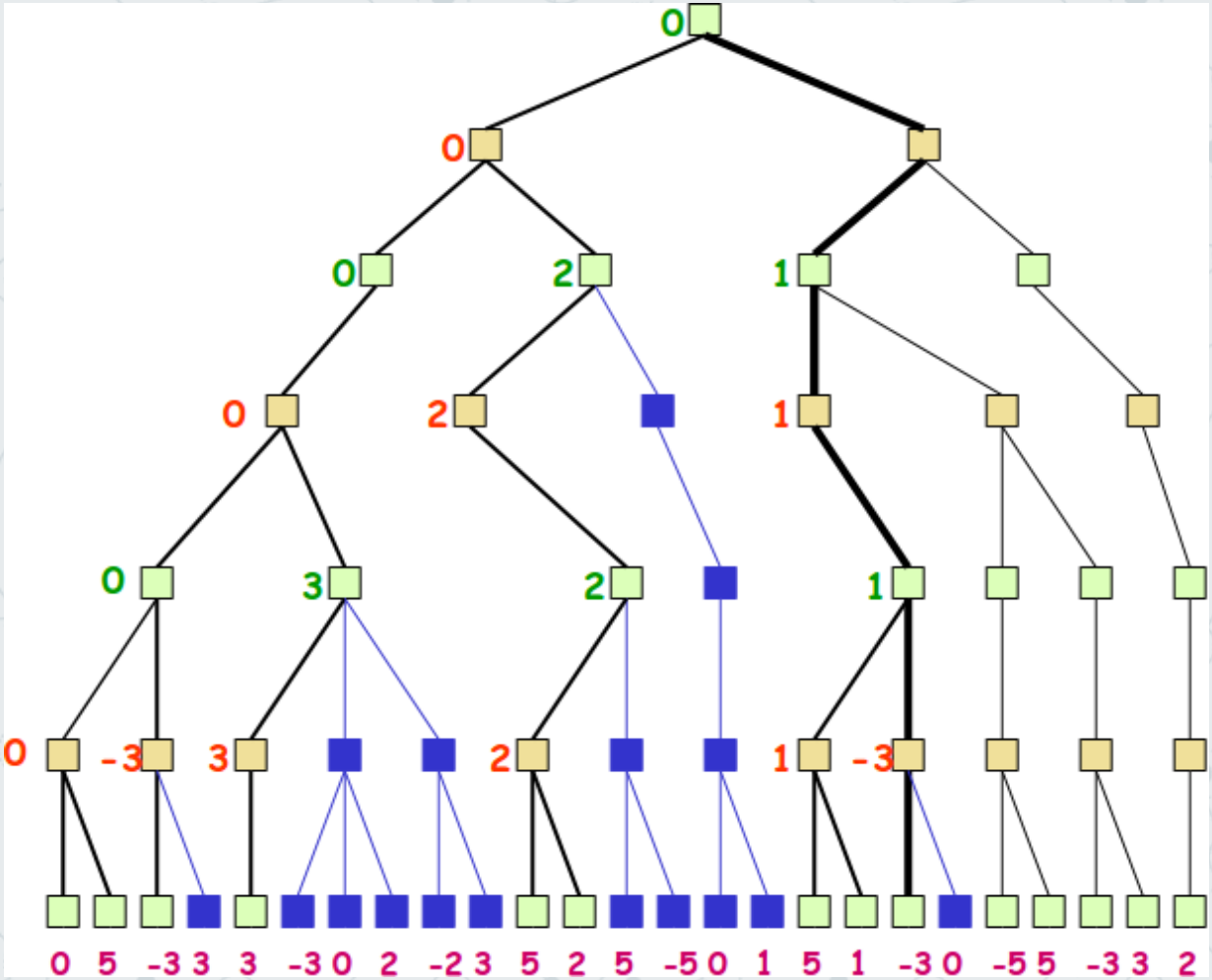
Alpha-Beta Pruning Example



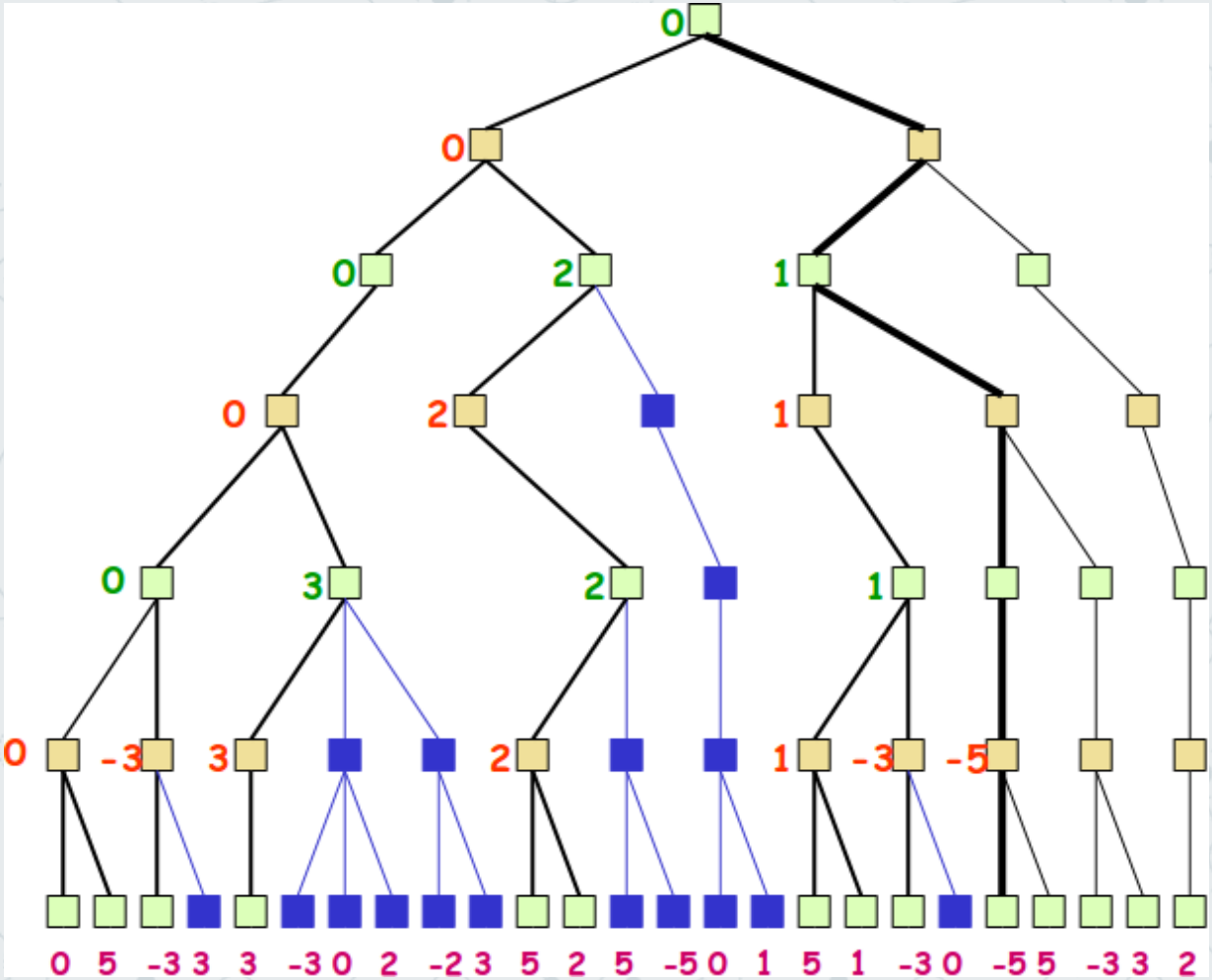
Alpha-Beta Pruning Example



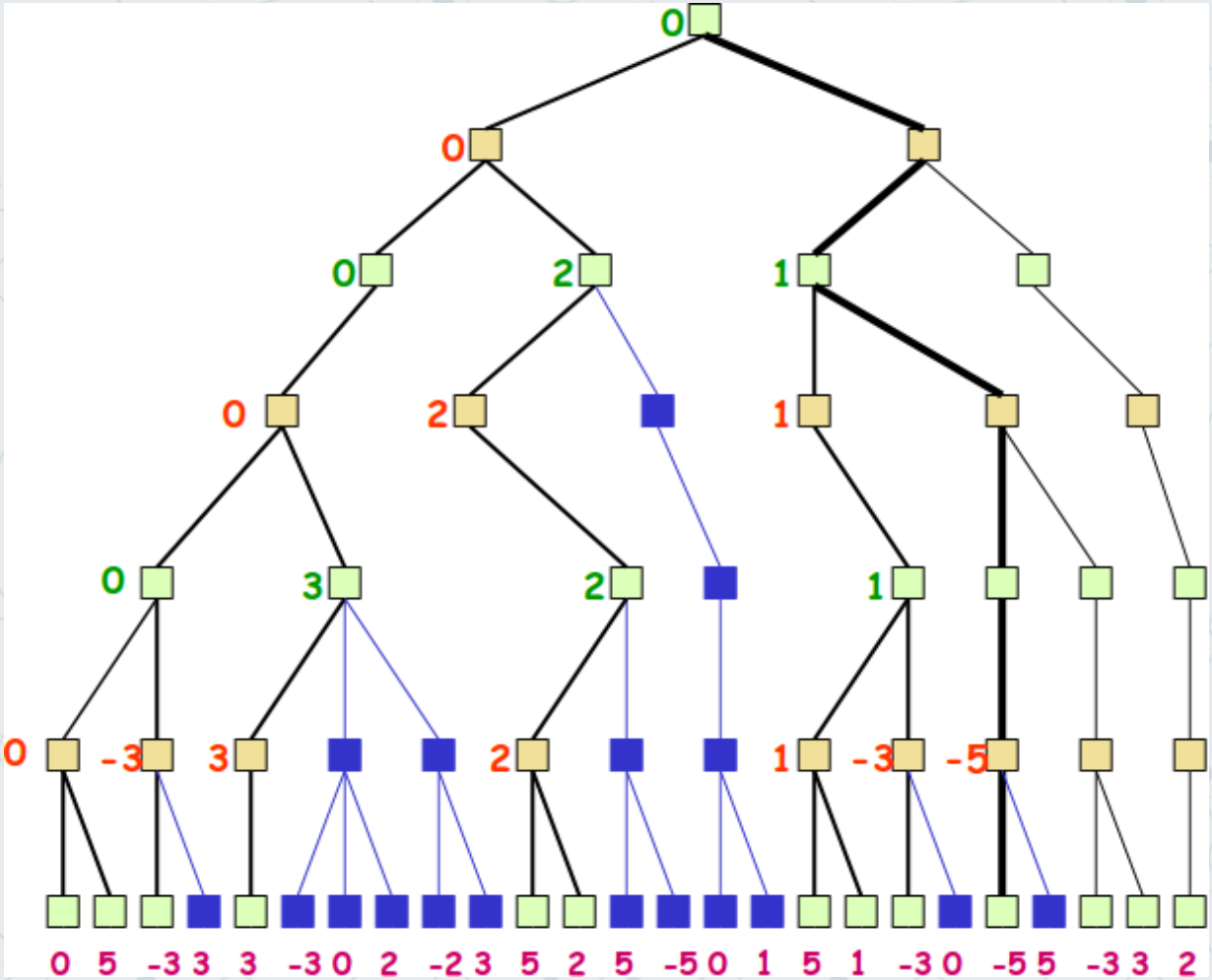
Alpha-Beta Pruning Example



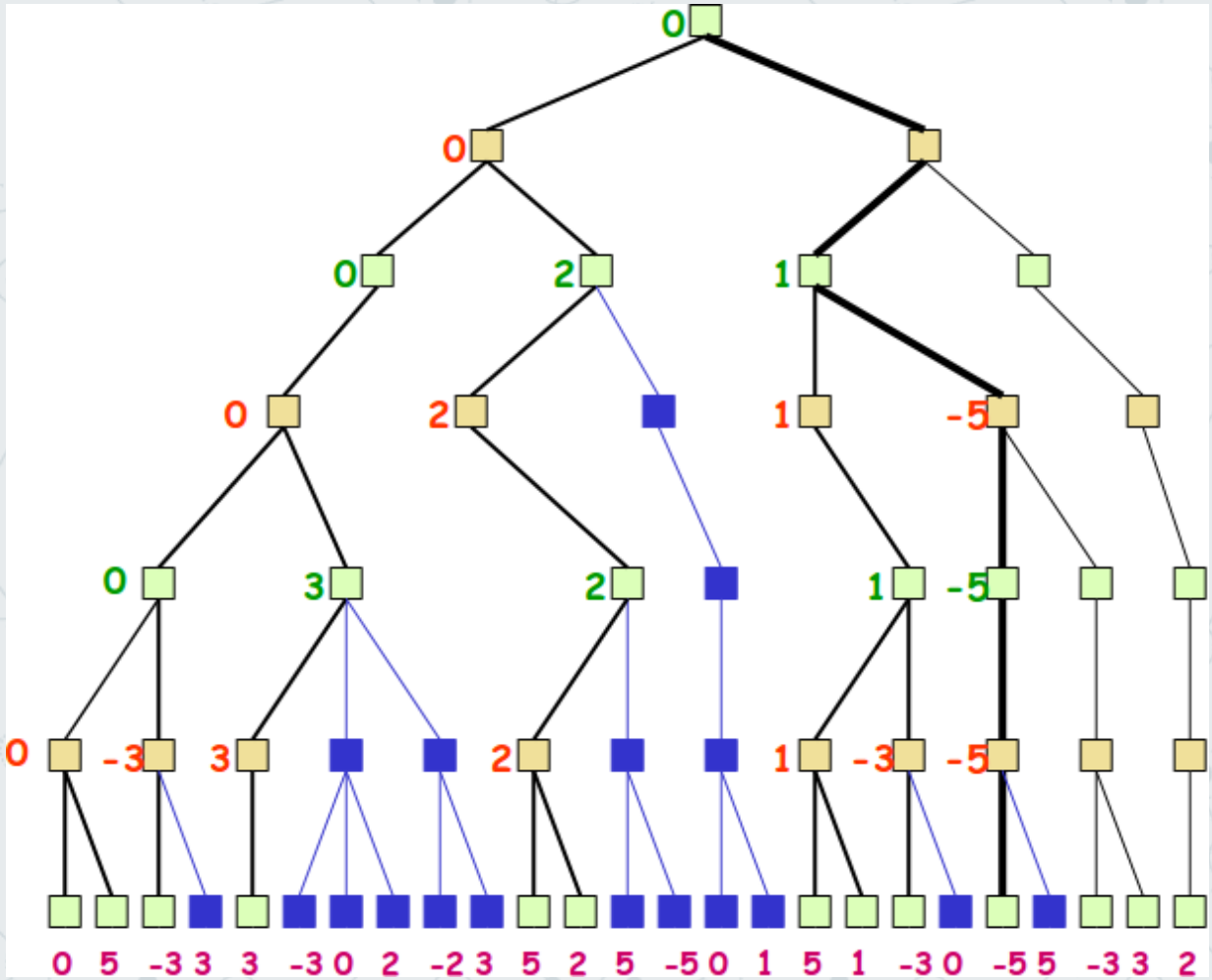
Alpha-Beta Pruning Example



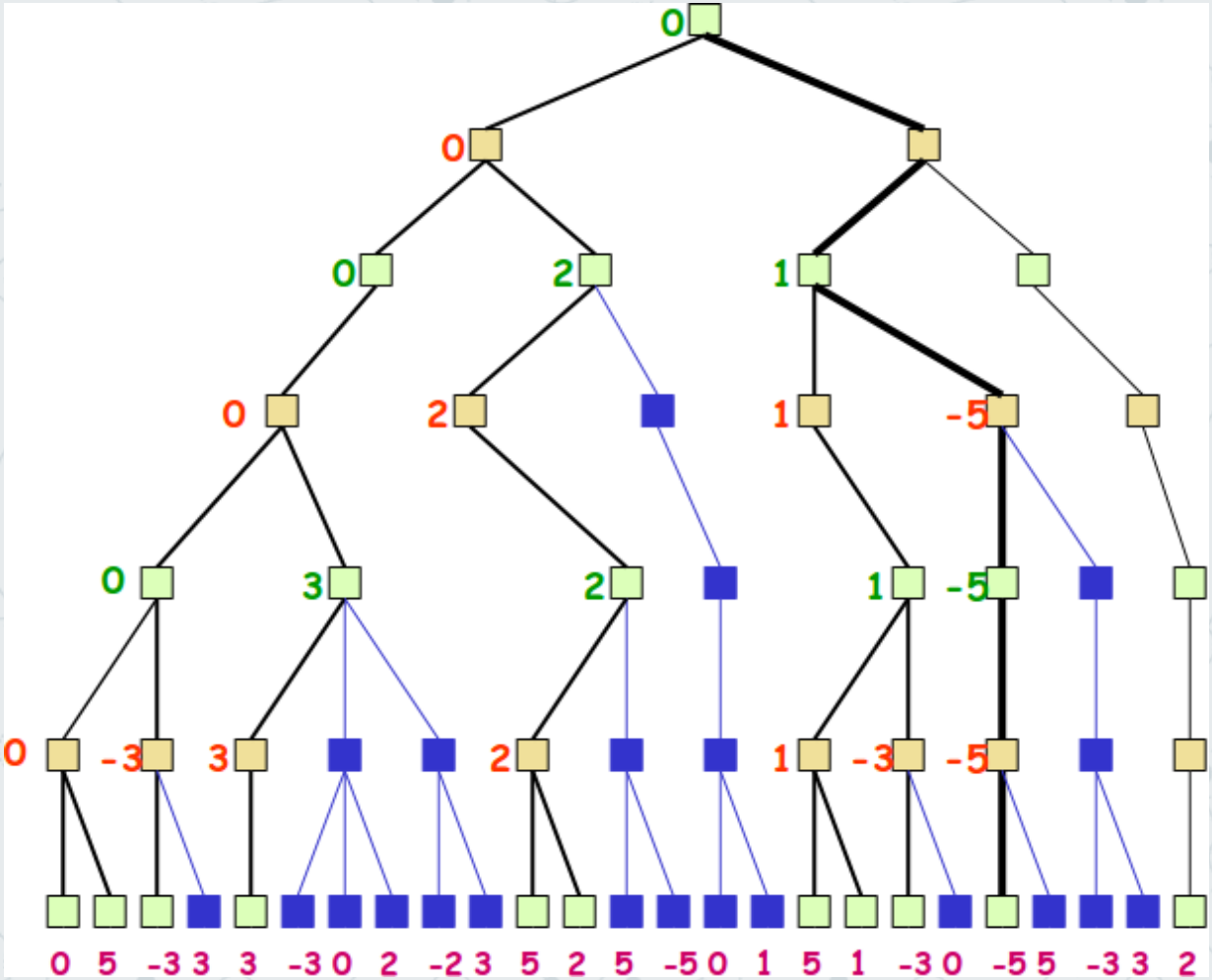
Alpha-Beta Pruning Example



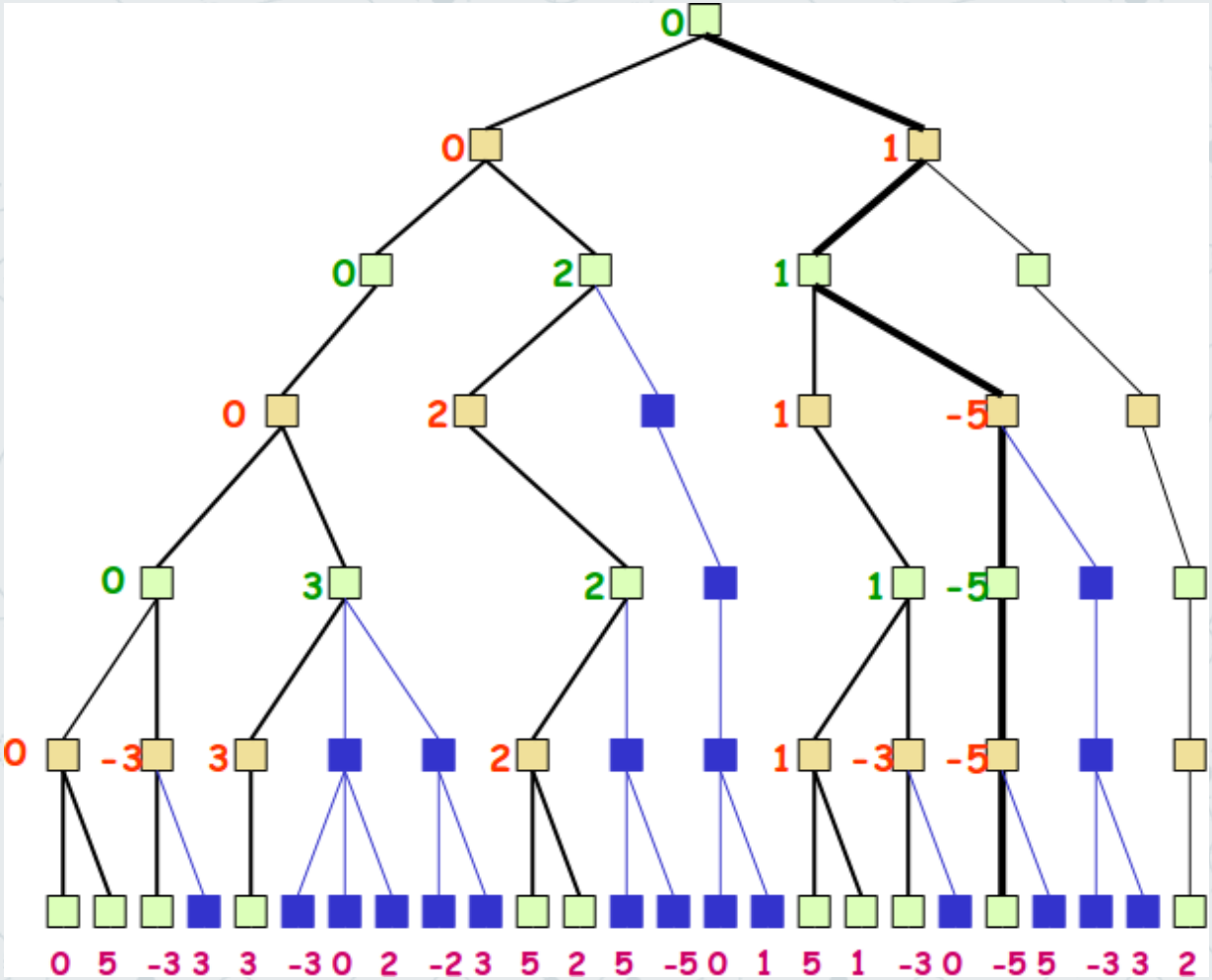
Alpha-Beta Pruning Example



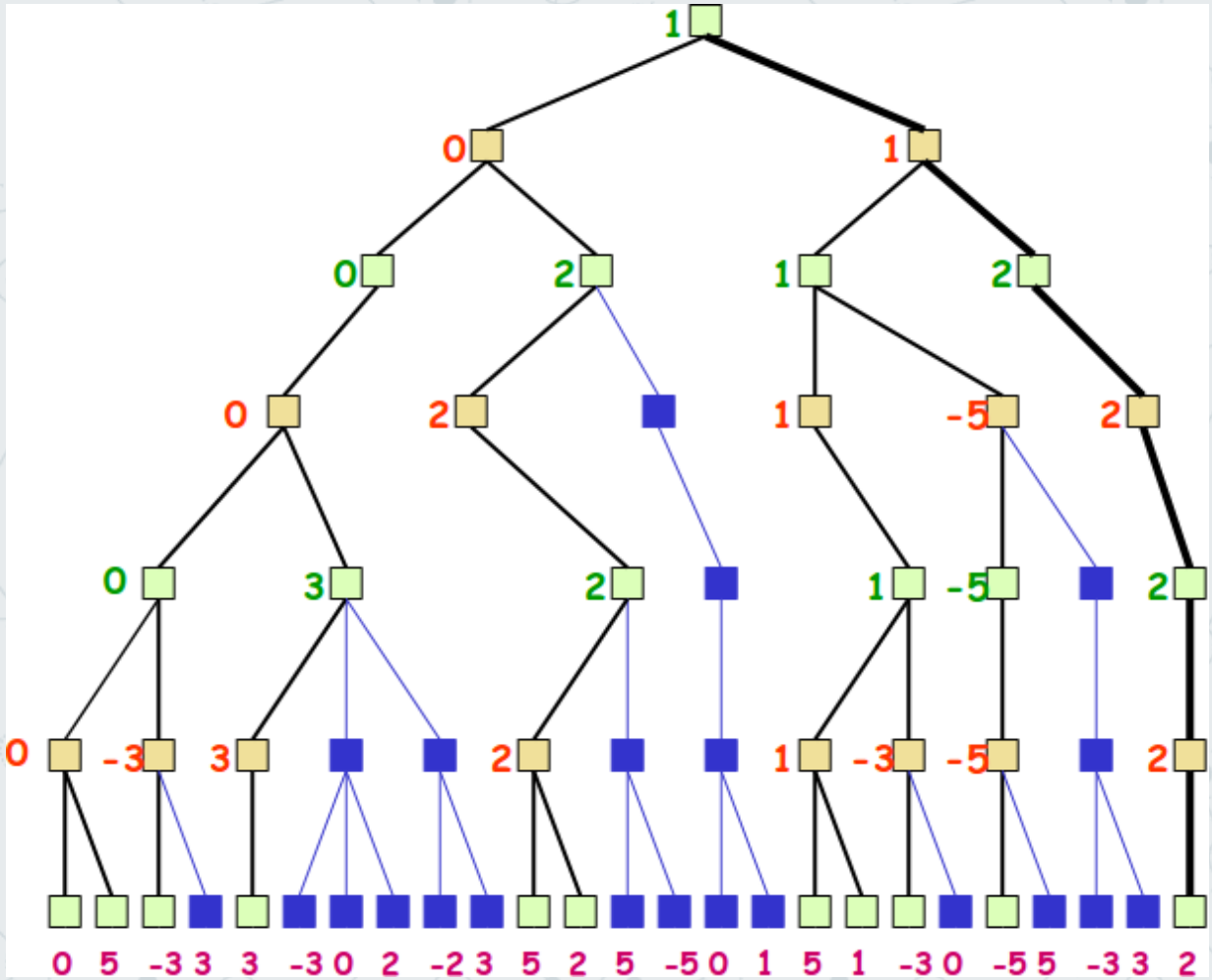
Alpha-Beta Pruning Example



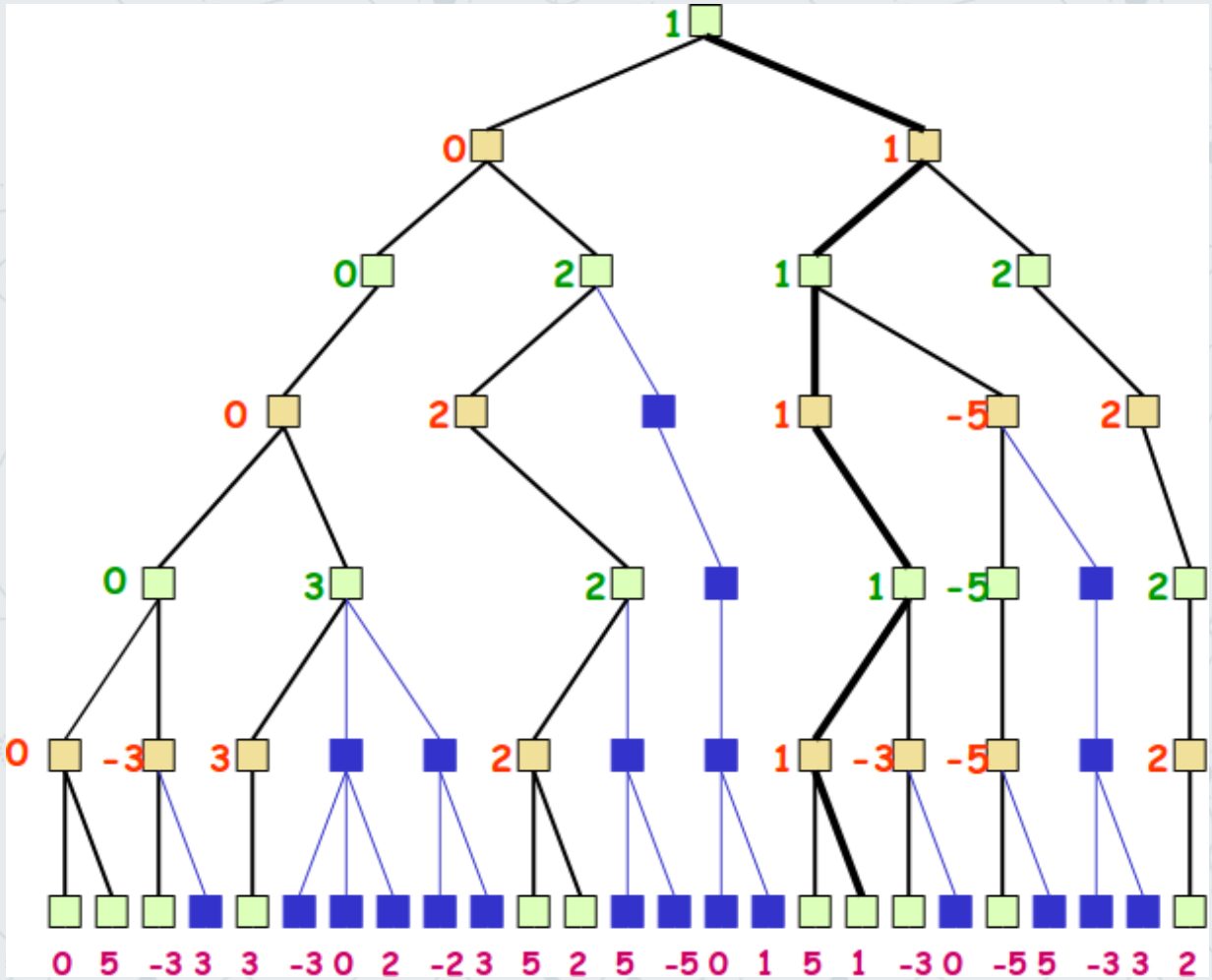
Alpha-Beta Pruning Example



Alpha-Beta Pruning Example



Alpha-Beta Pruning Example



Alpha-Beta Pruning Formally

Keep track of two bounds, the values of the best choices so far along the path:

- α – the best value for MAX
- β – the best value for MIN

Stop branch evaluation if limit is breached
Note that by "keep track", we mean that α and β are stored for **every** node

- Initially all $\alpha = -\infty$ and all $\beta = +\infty$

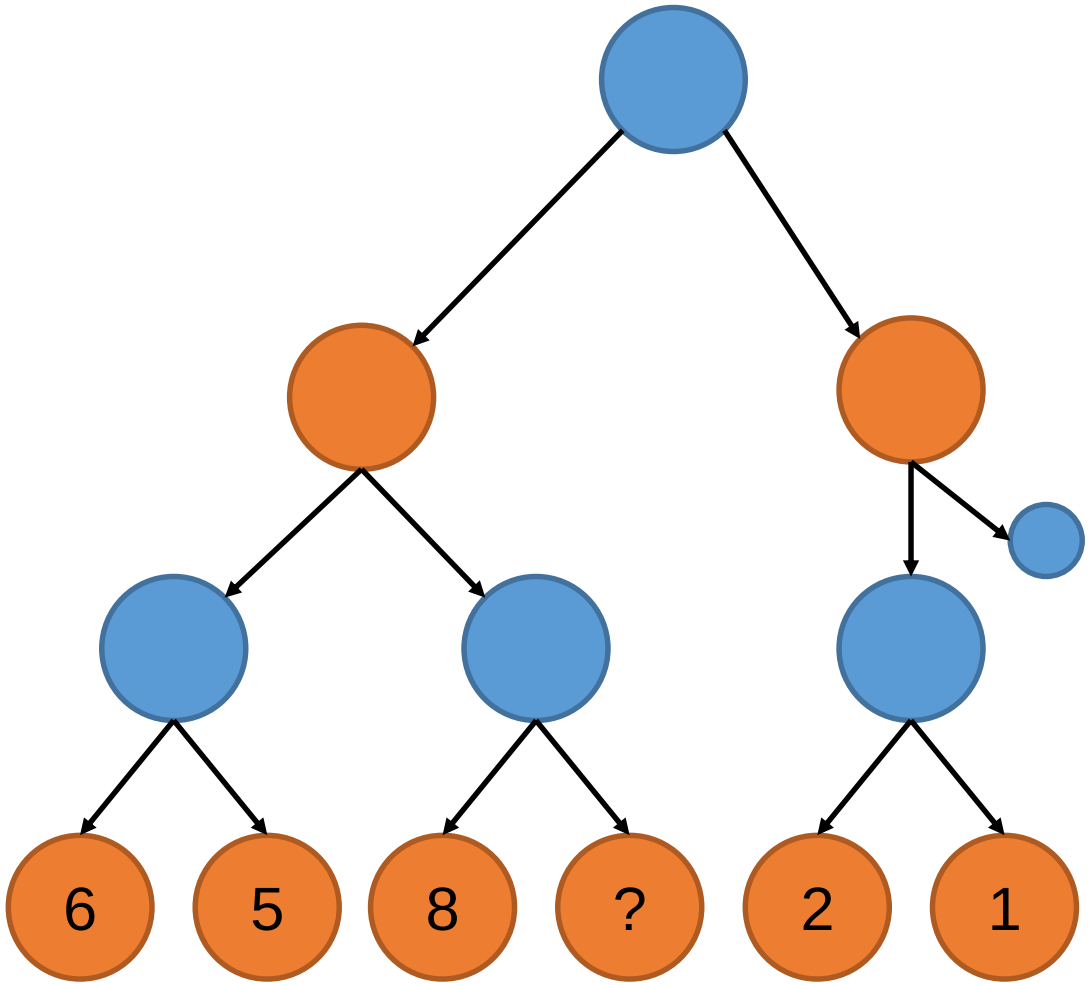
Pay attention that ordering of nodes is important in order to maximize pruning

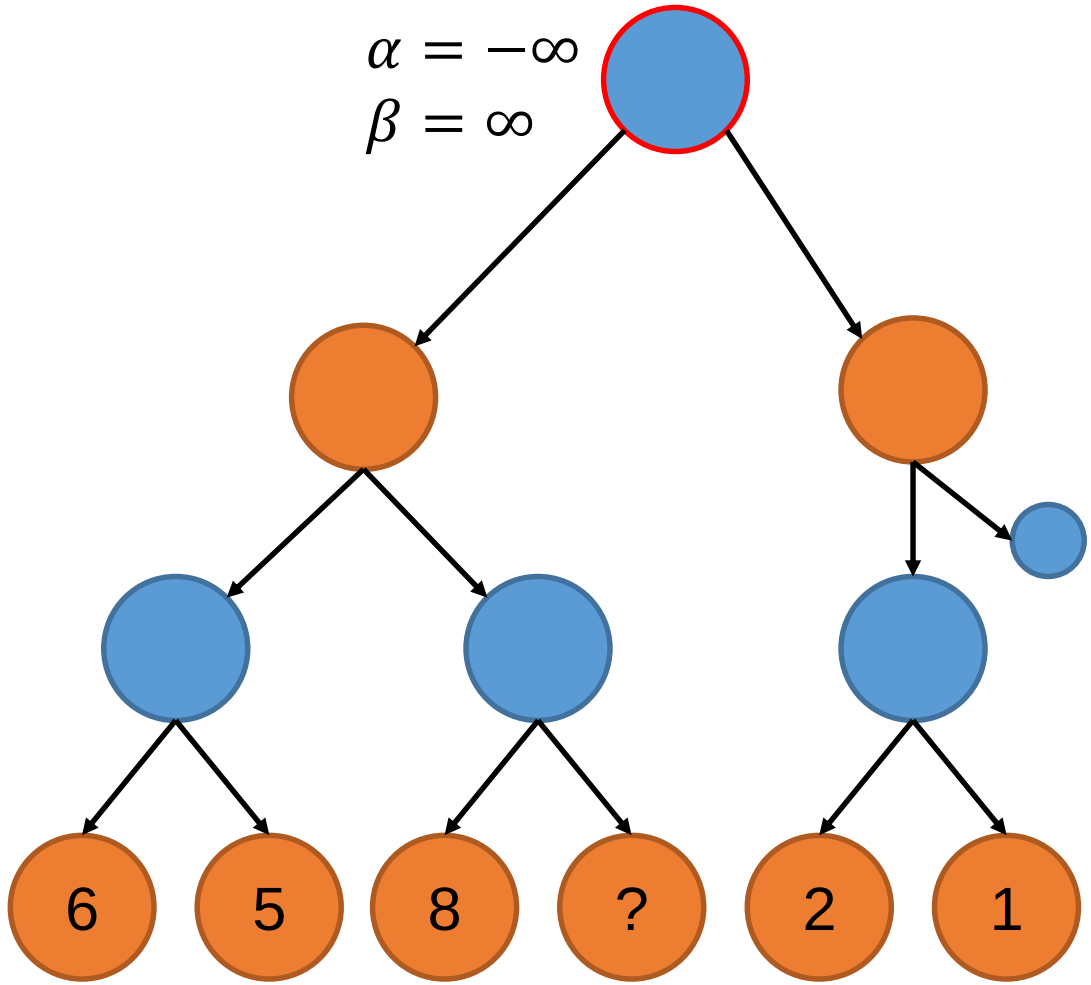
Alpha-Beta Pruning Pseudocode

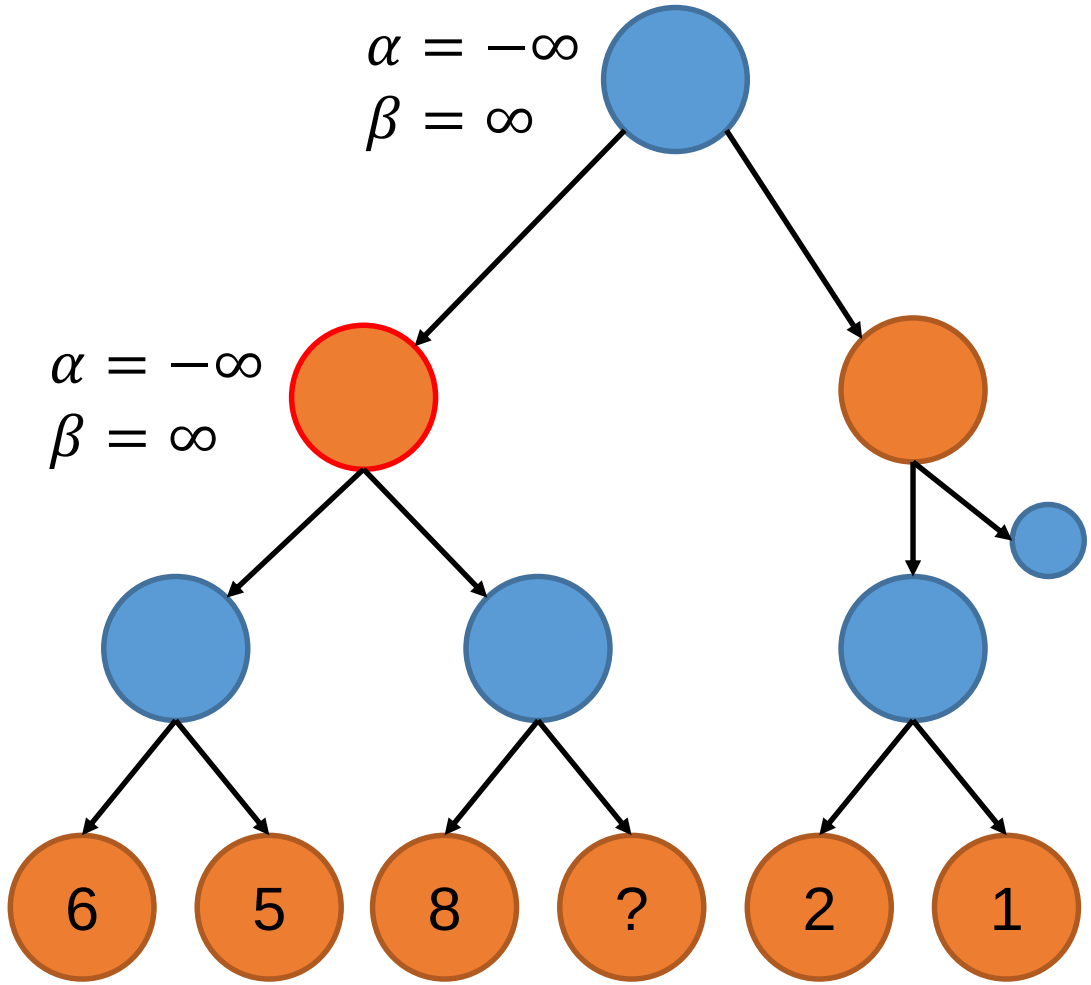
$\boldsymbol{alphabeta(node, depth, \alpha, \beta, Player)}$
if $depth = 0$ or $node$ is a terminal node
return the heuristic value of $node$

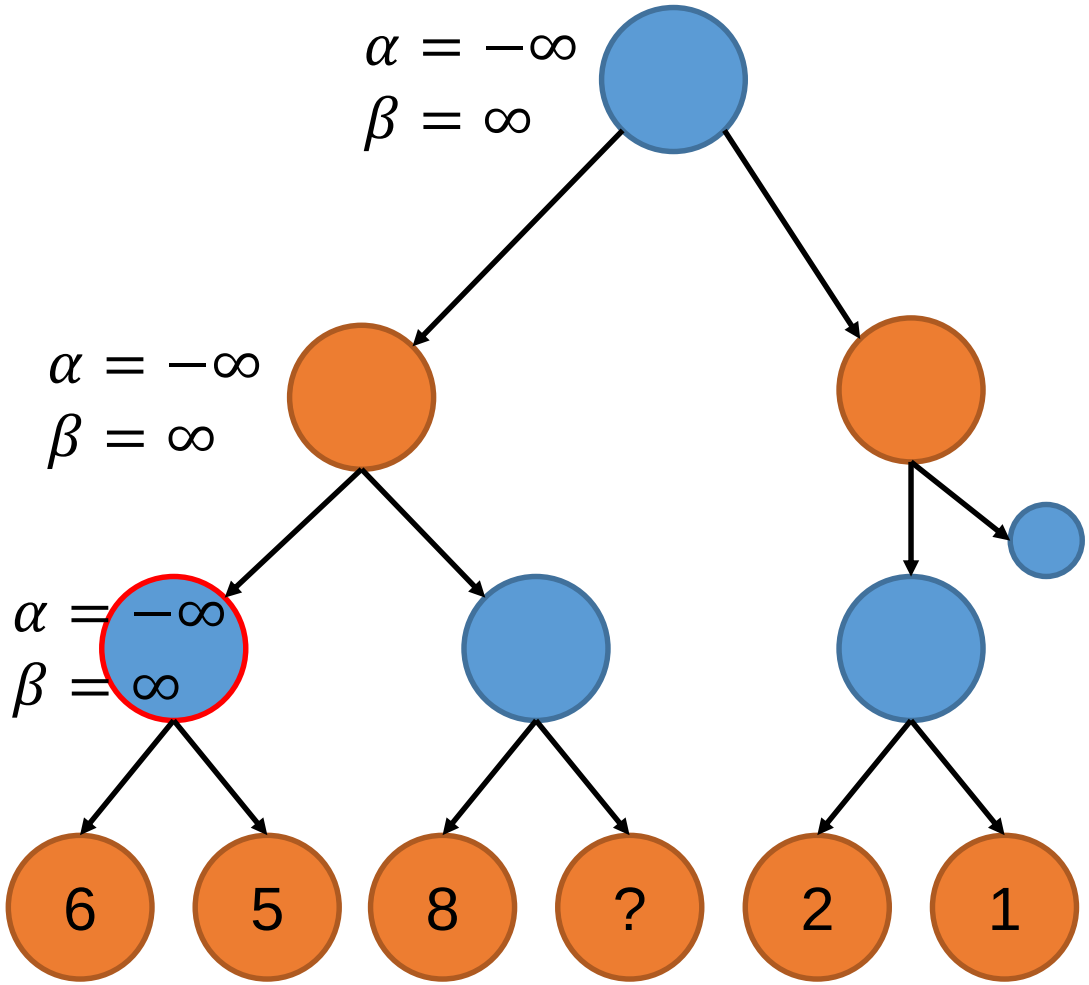
if $Player = MaxPlayer$
- for each $child$ of $node$
  - $\alpha = \max\{\alpha, alphabeta(child, depth - 1, \alpha, \beta, not(Player))\}$
  - if $\beta \leq \alpha$
    - break \\β cut-off
  - return α

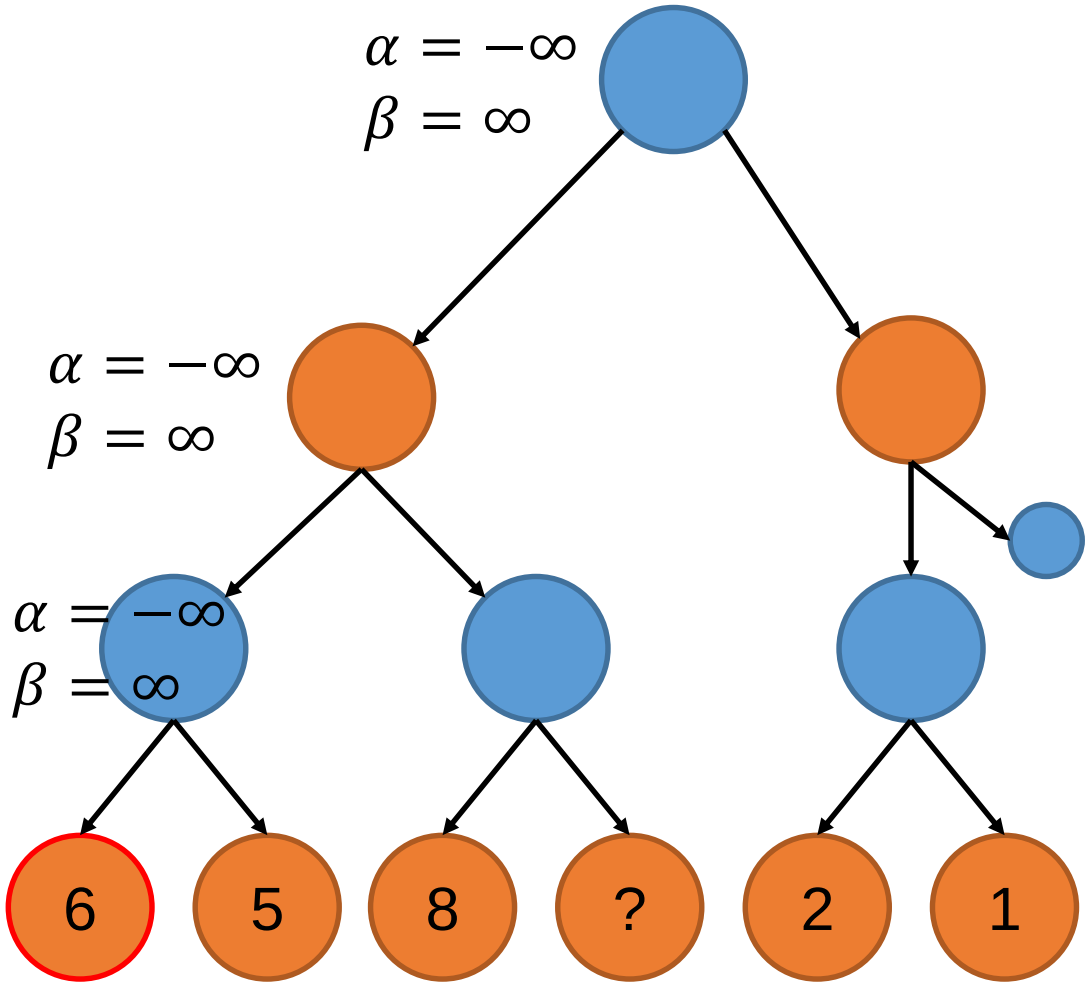
if $Player = MinPlayer$
- for each $child$ of $node$
  - $\beta = \min\{\alpha, alphabeta(child, depth - 1, \alpha, \beta, not(Player))\}$
  - if $\beta \leq \alpha$
    - break \\α cut-off
  - return β

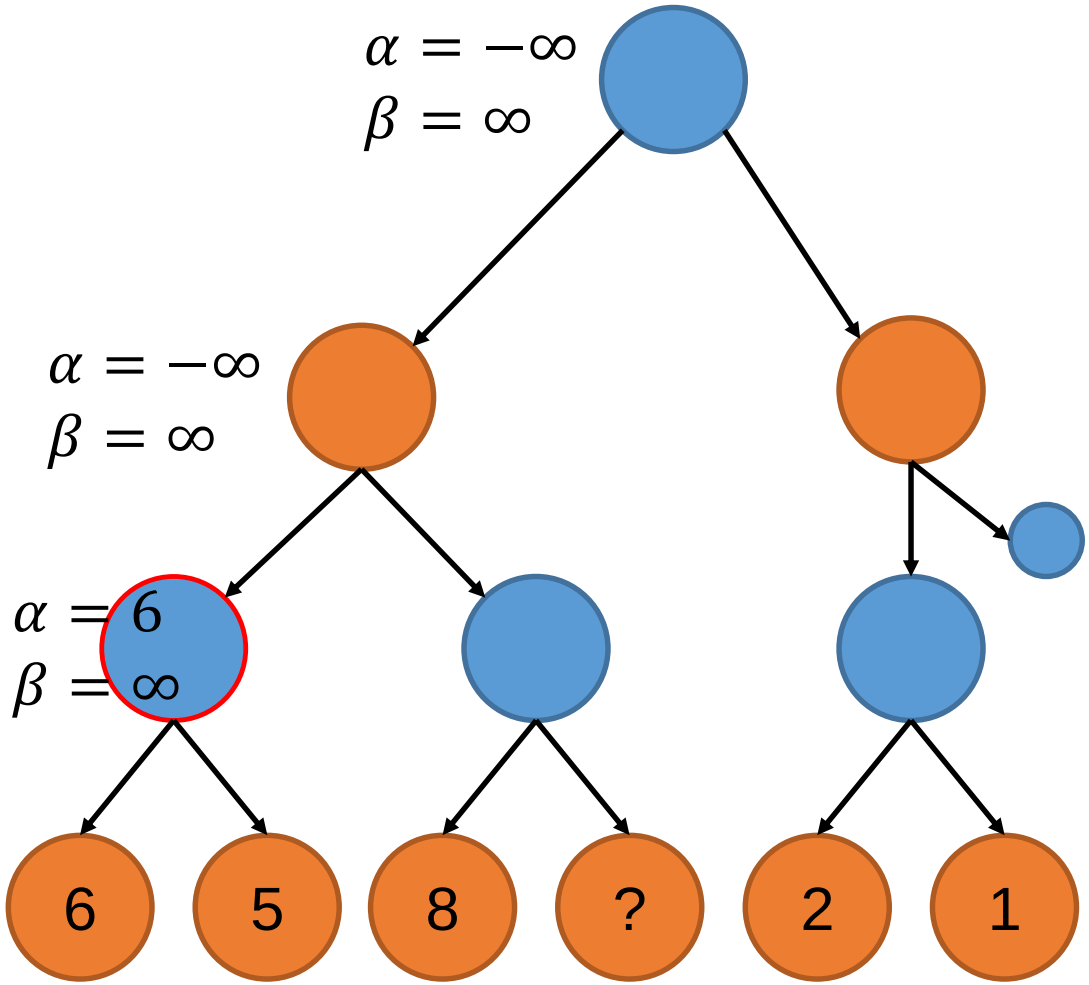

6
5
8
?
2
1

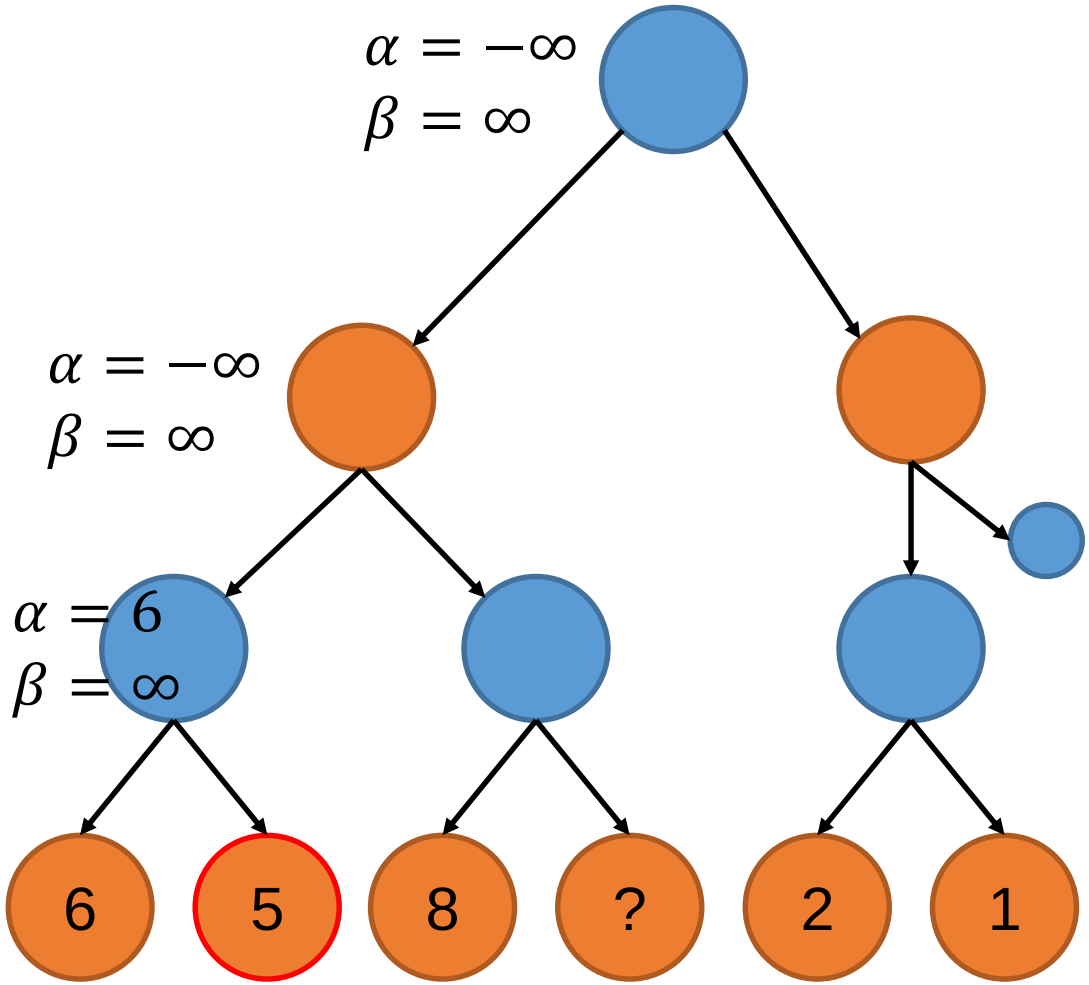

$\alpha = -\infty$
$\beta = \infty$
6
5
8
?
2
1

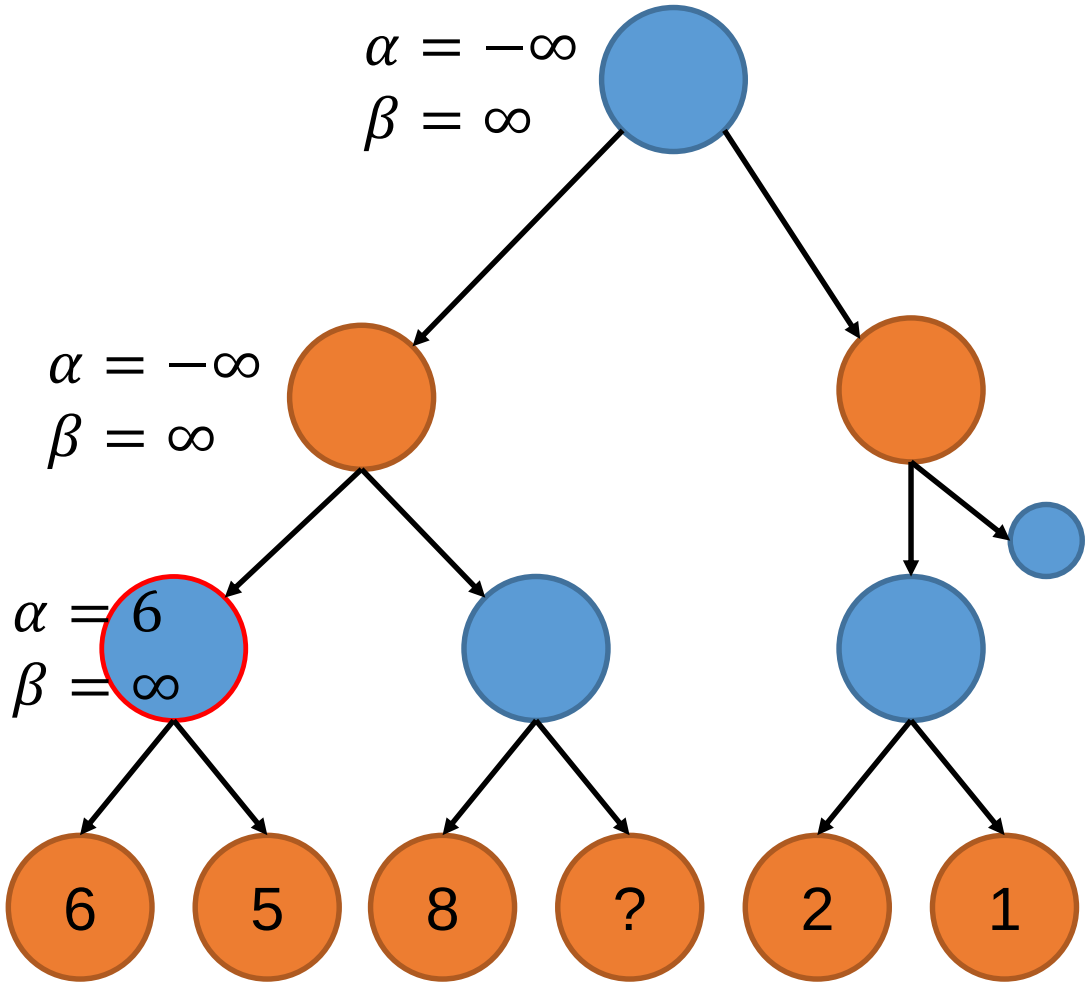

$\alpha = -\infty$
$\beta = \infty$
$\alpha = -\infty$
$\beta = \infty$
6
5
8
?
2
1

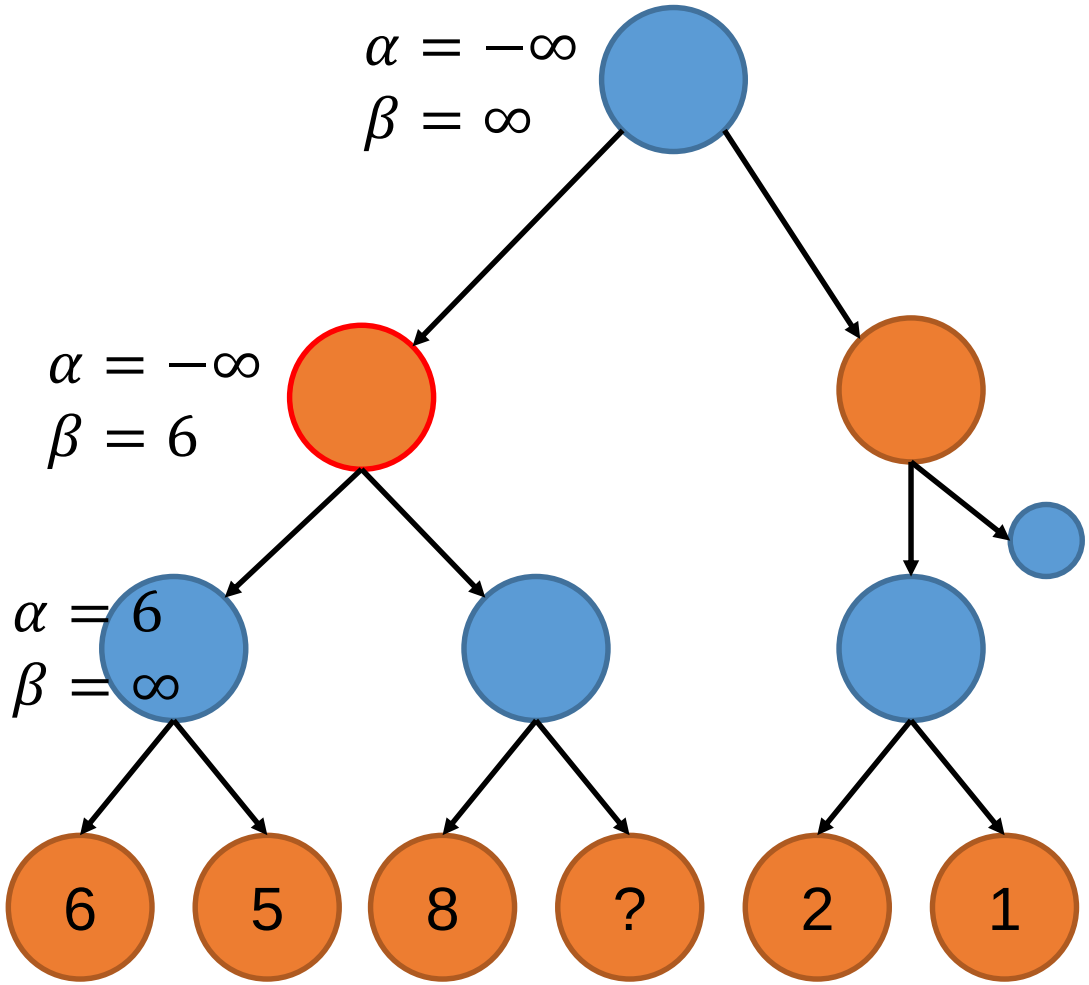

$\alpha = -\infty$
$\beta = \infty$
$\alpha = -\infty$
$\beta = \infty$
$\alpha = -\infty$
$\beta = \infty$
6
5
8
?
2
1

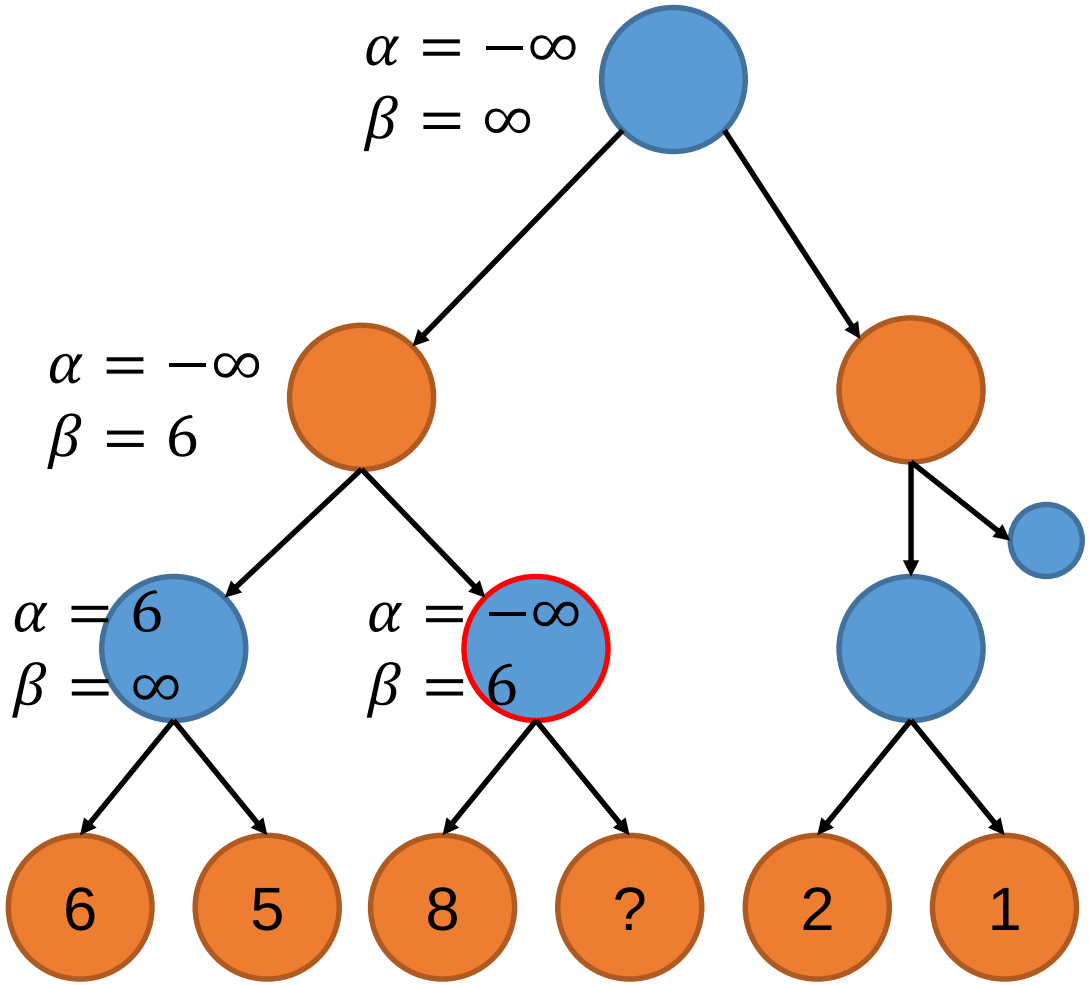

$\alpha = -\infty$
$\beta = \infty$
$\alpha = -\infty$
$\beta = \infty$
$\alpha = -\infty$
$\beta = \infty$
6
5
8
?
2
1

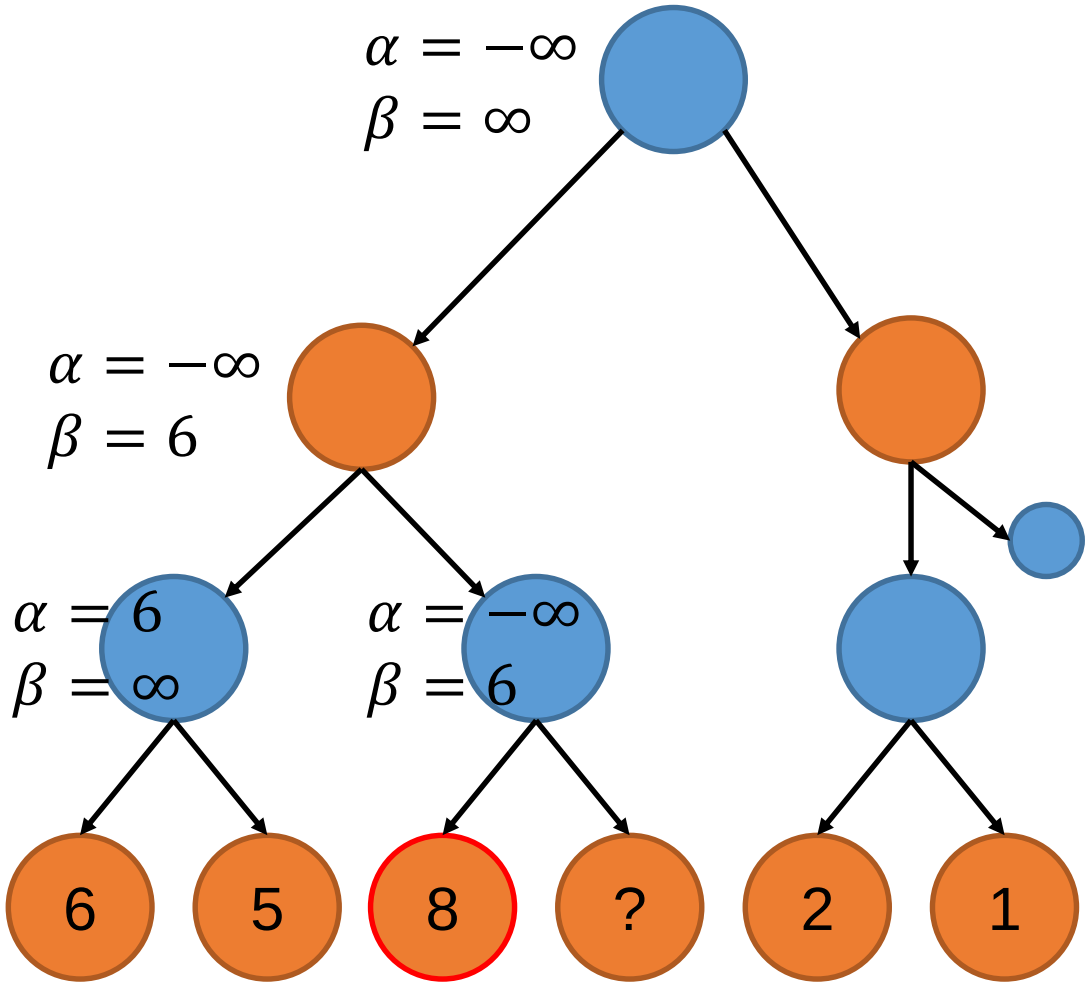

$\alpha = -\infty$
$\beta = \infty$
$\alpha = -\infty$
$\beta = \infty$
$\alpha = 6$
$\beta = \infty$
6
5
8
?
2
1

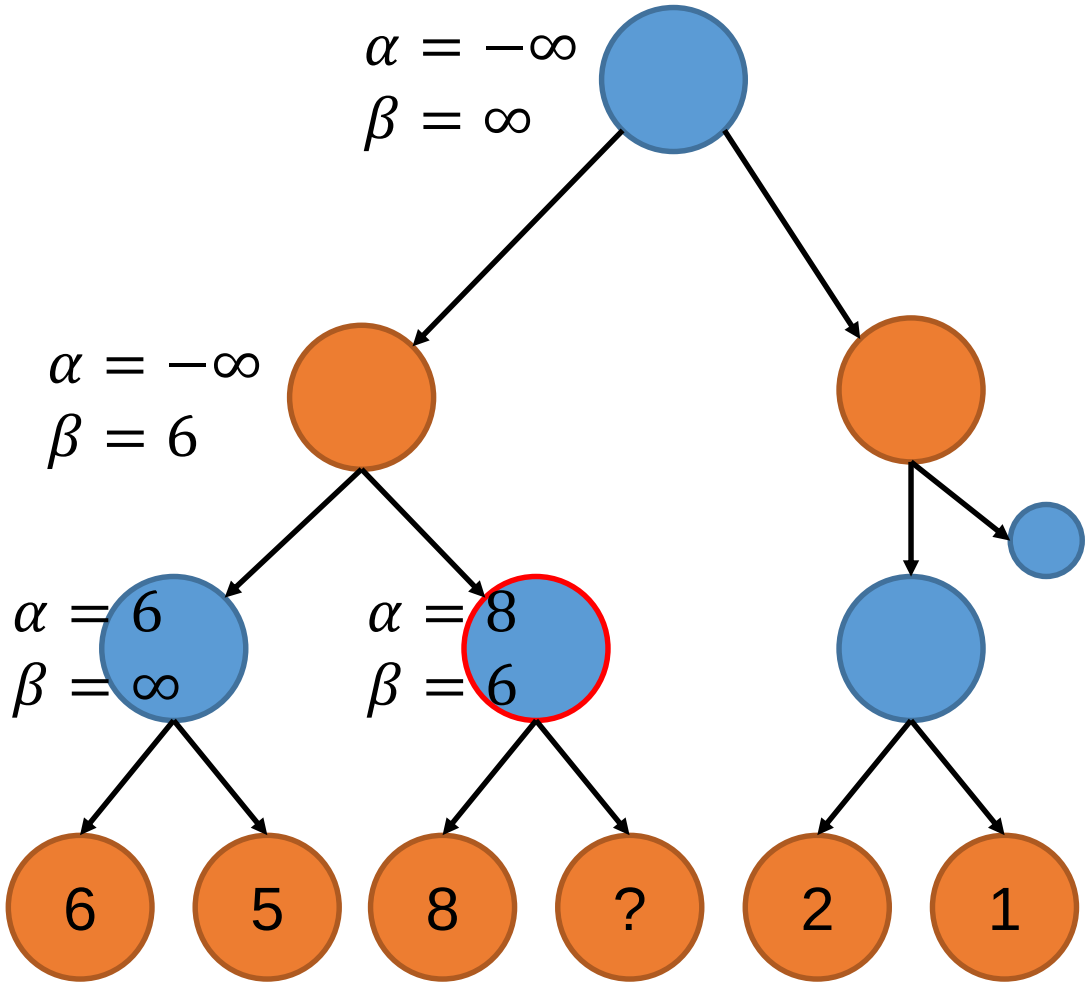

$\alpha = -\infty$
$\beta = \infty$
$\alpha = -\infty$
$\beta = \infty$
$\alpha = 6$
$\beta = \infty$
6
5
8
?
2
1

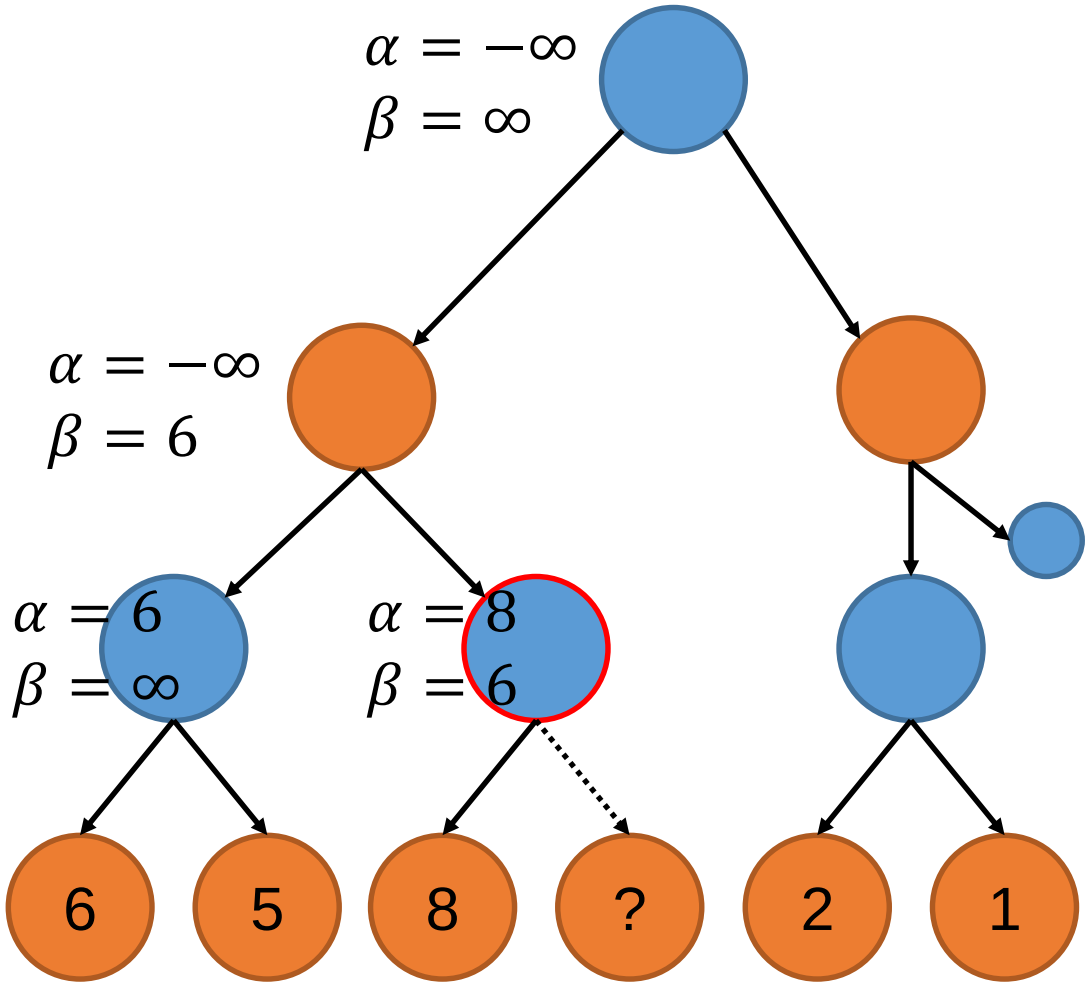

$\alpha = -\infty$
$\beta = \infty$
$\alpha = -\infty$
$\beta = \infty$
$\alpha = 6$
$\beta = \infty$
6
5
8
?
2
1

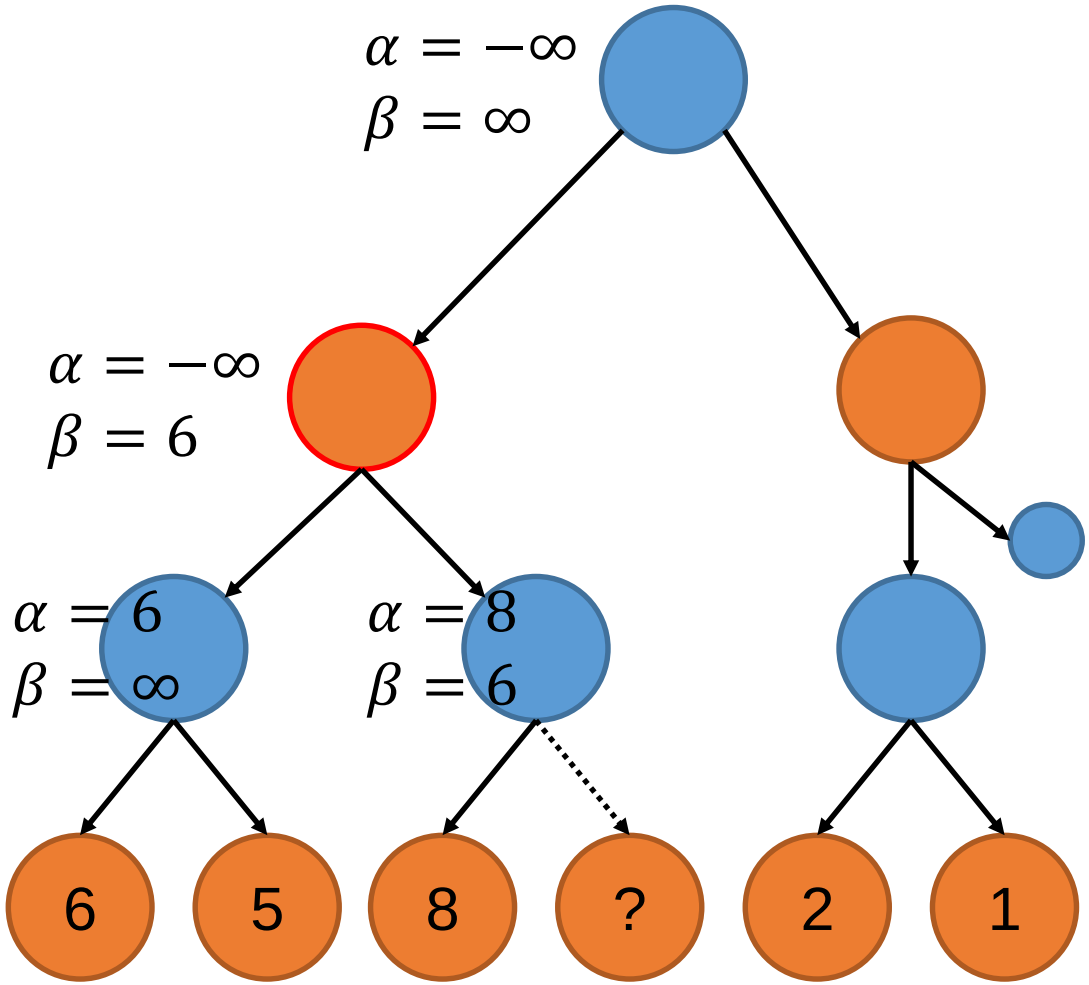

$\alpha = -\infty$
$\beta = \infty$
$\alpha = -\infty$
$\beta = 6$
$\alpha = 6$
$\beta = \infty$
6
5
8
?
2
1

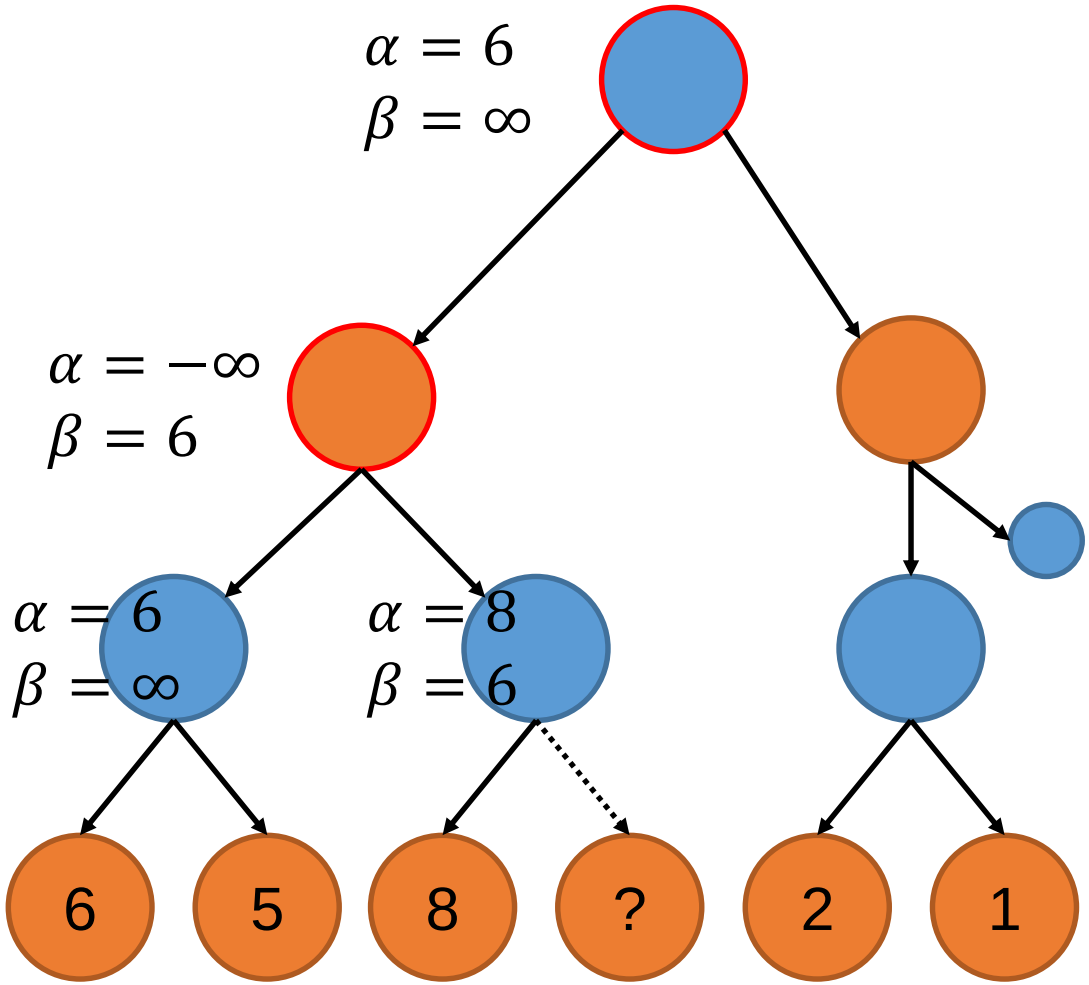

$\alpha = -\infty$
$\beta = \infty$
$\alpha = -\infty$
$\beta = 6$
$\alpha = 6$
$\beta = \infty$
$\alpha = -\infty$
$\beta = 6$
6
5
8
?
2
1

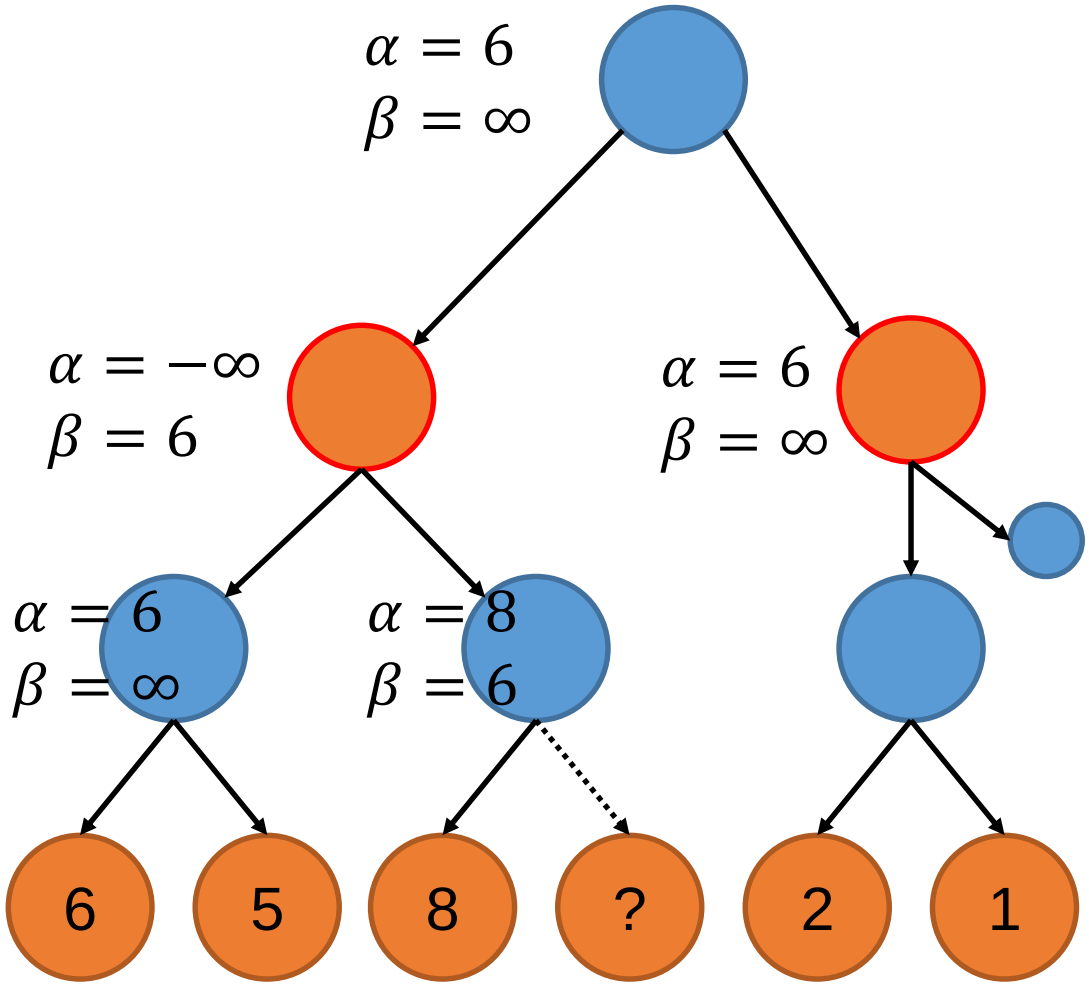

$\alpha = -\infty$
$\beta = \infty$
$\alpha = -\infty$
$\beta = 6$
$\alpha = 6$
$\beta = \infty$
$\alpha = -\infty$
$\beta = 6$
6
5
8
?
2
1

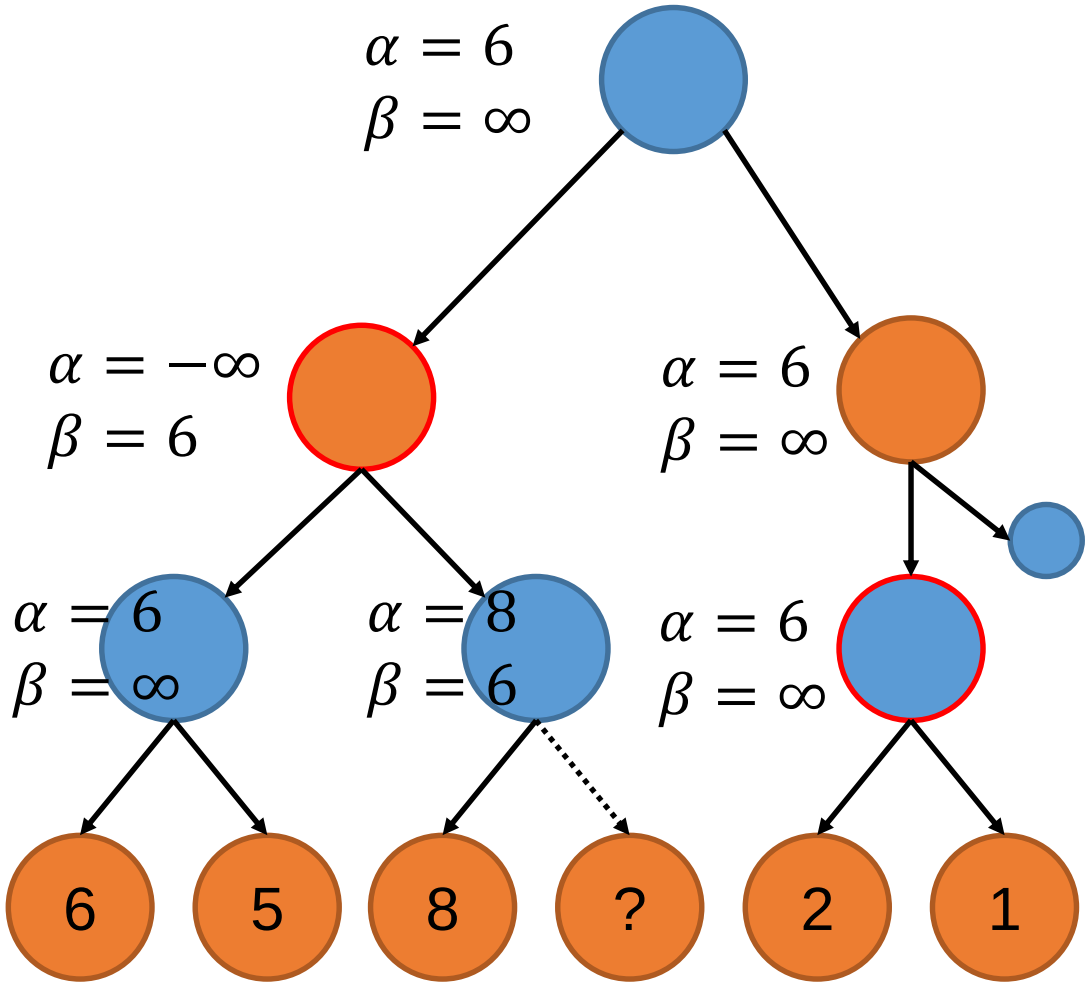

$\alpha = -\infty$
$\beta = \infty$
$\alpha = -\infty$
$\beta = 6$
$\alpha = 6$
$\beta = \infty$
$\alpha = 8$
$\beta = 6$
6
5
8
?
2
1

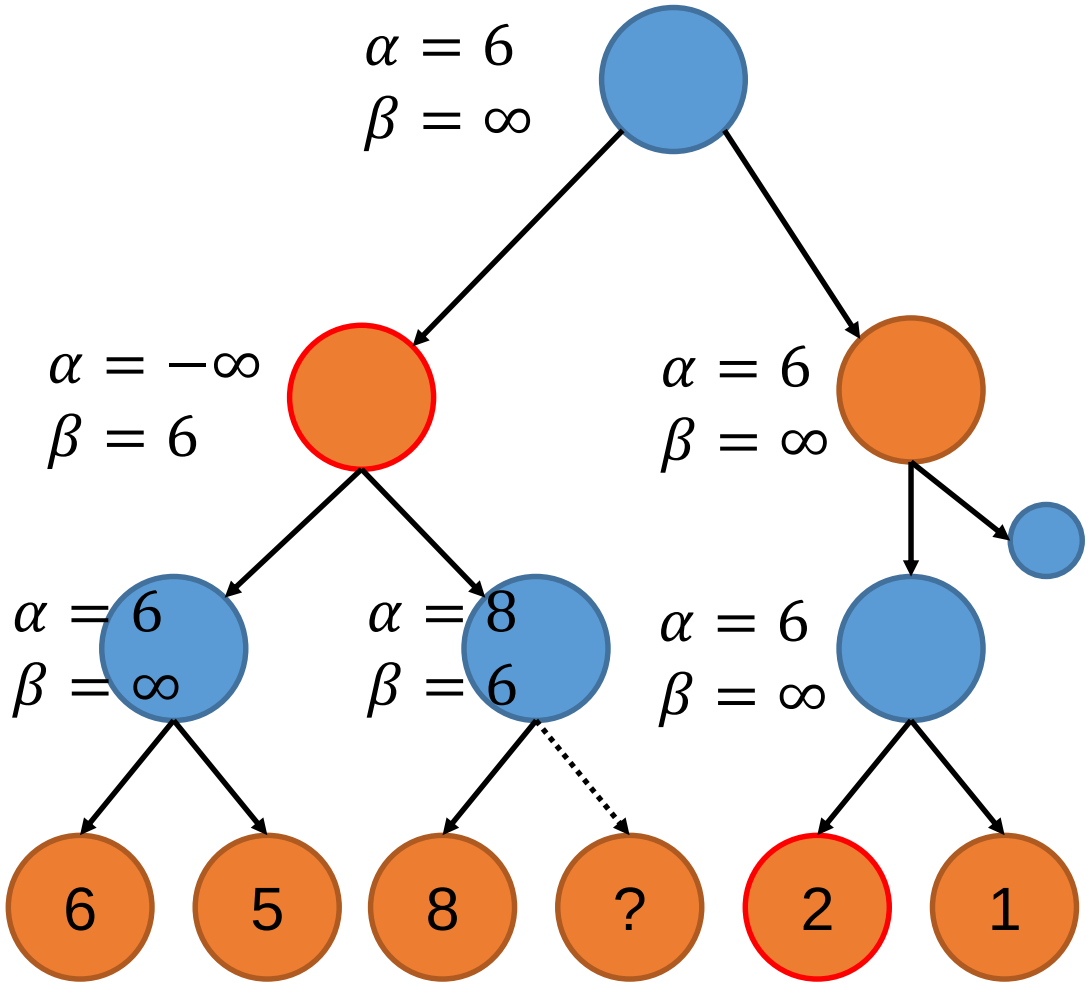

$\alpha = -\infty$
$\beta = \infty$
$\alpha = -\infty$
$\beta = 6$
$\alpha = 6$
$\beta = \infty$
$\alpha = 8$
$\beta = 6$
6
5
8
?
2
1

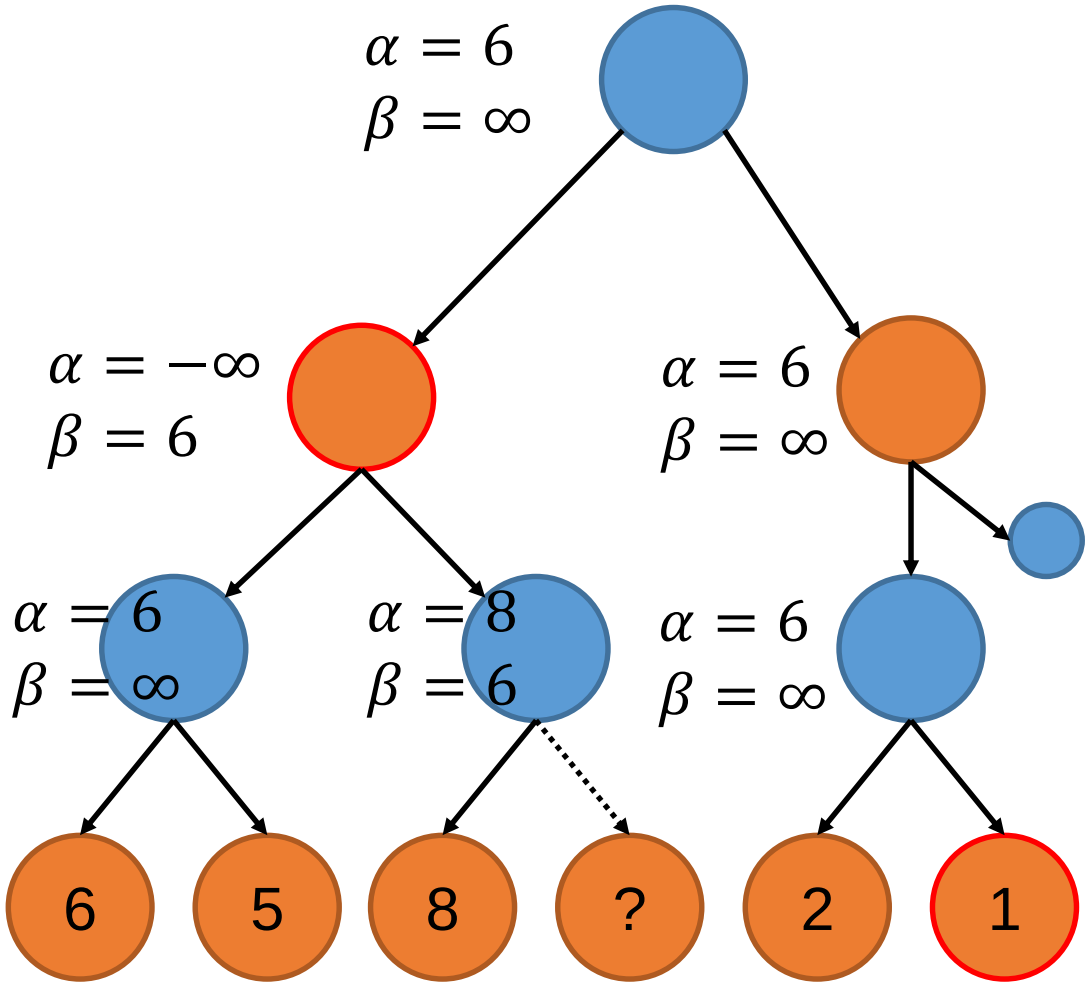

$\alpha = -\infty$
$\beta = \infty$
$\alpha = -\infty$
$\beta = 6$
$\alpha = 6$
$\beta = \infty$
$\alpha = 8$
$\beta = 6$
6
5
8
?
2
1

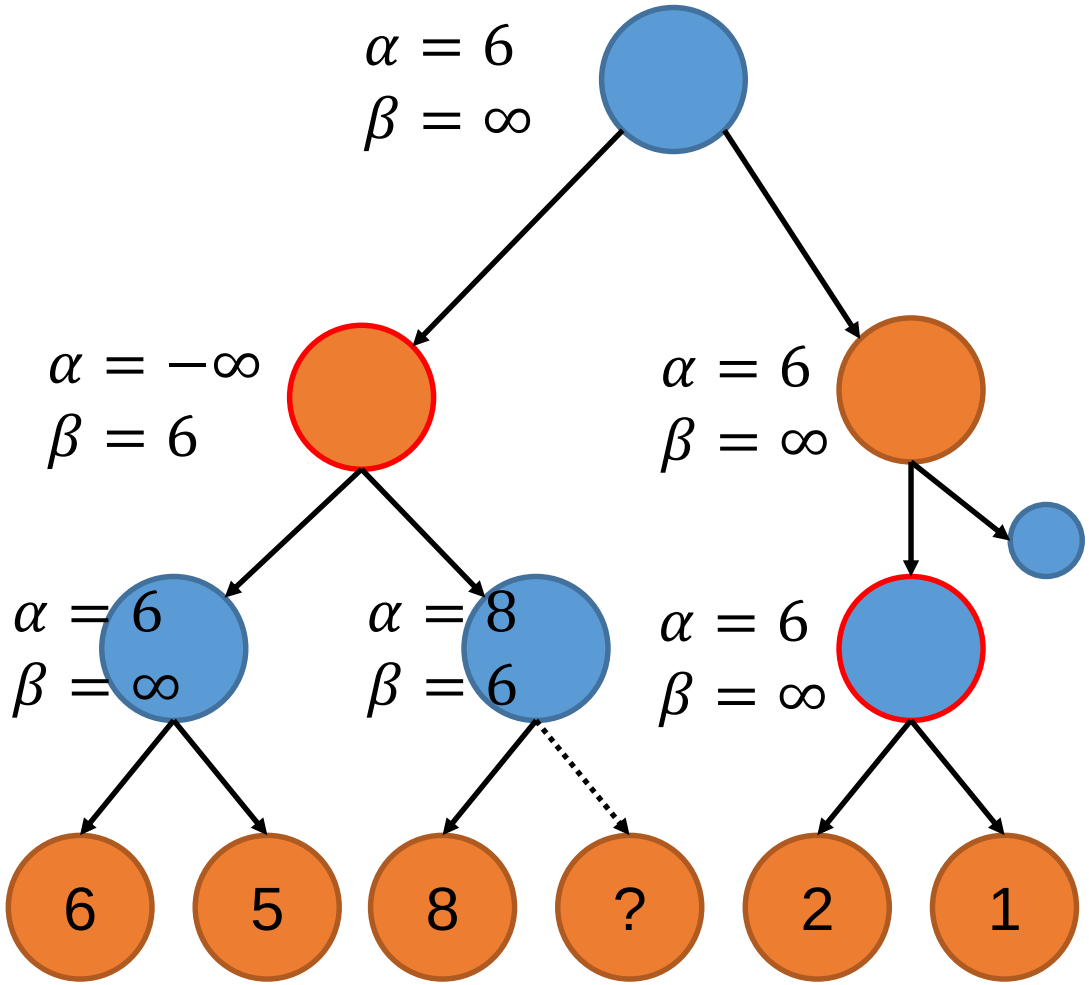

$\alpha = 6$
$\beta = \infty$
$\alpha = -\infty$
$\beta = 6$
$\alpha = 6$
$\beta = \infty$
$\alpha = 8$
$\beta = 6$
6
5
8
?
2
1

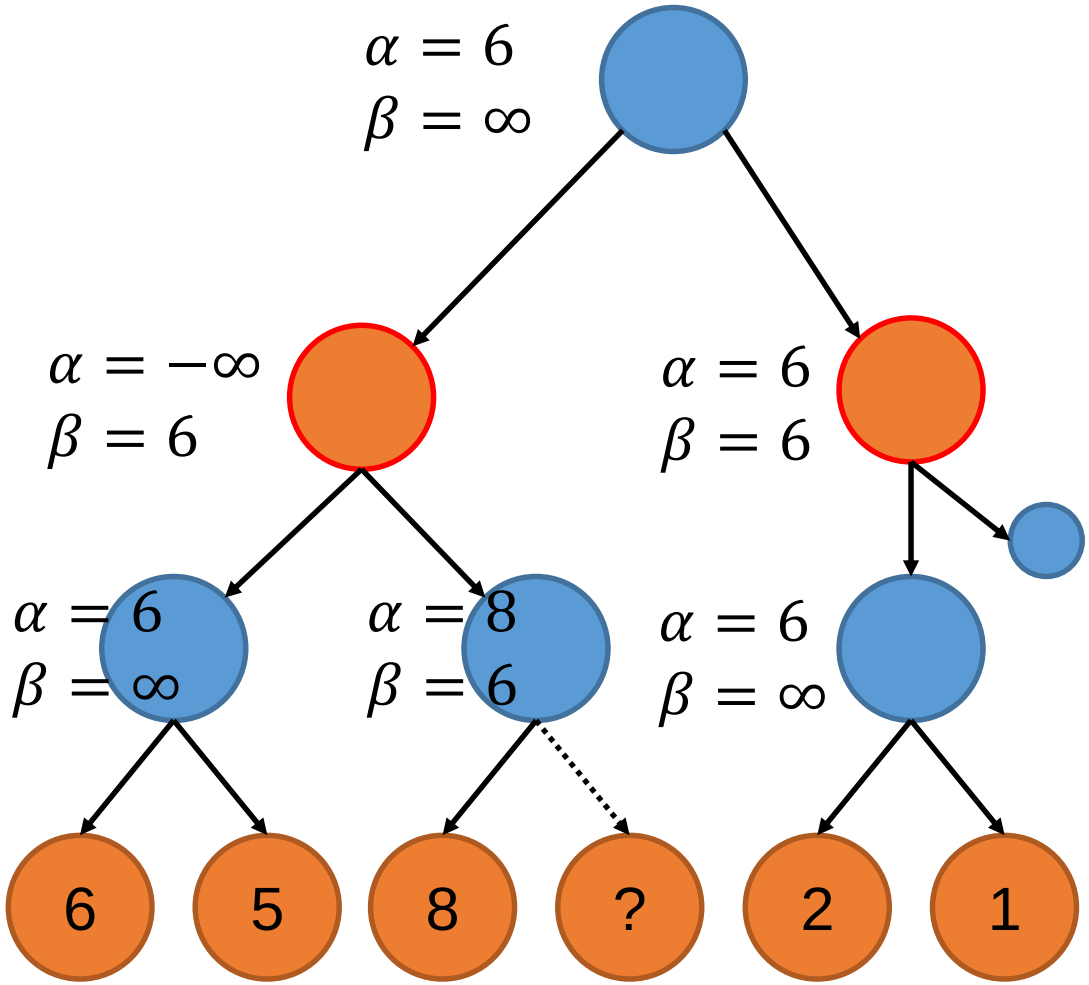

$\alpha = 6$
$\beta = \infty$
$\alpha = -\infty$
$\beta = 6$
$\alpha = 6$
$\beta = \infty$
$\alpha = 6$
$\beta = \infty$
$\alpha = 8$
$\beta = 6$
6
5
8
?
2
1

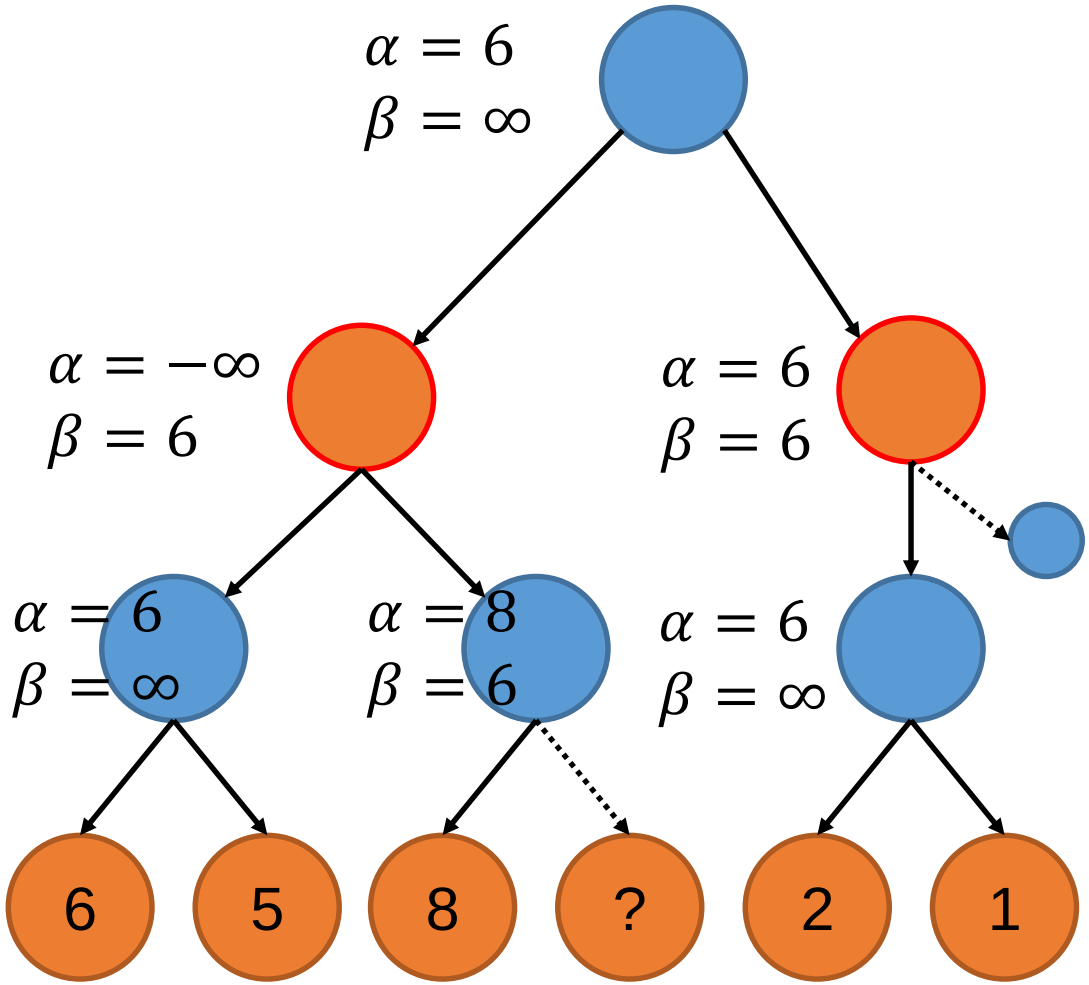

$\alpha = 6$
$\beta = \infty$
$\alpha = -\infty$
$\beta = 6$
$\alpha = 6$
$\beta = \infty$
$\alpha = 6$
$\beta = \infty$
$\alpha = 8$
$\beta = 6$
$\alpha = 6$
$\beta = \infty$
6
5
8
?
2
1

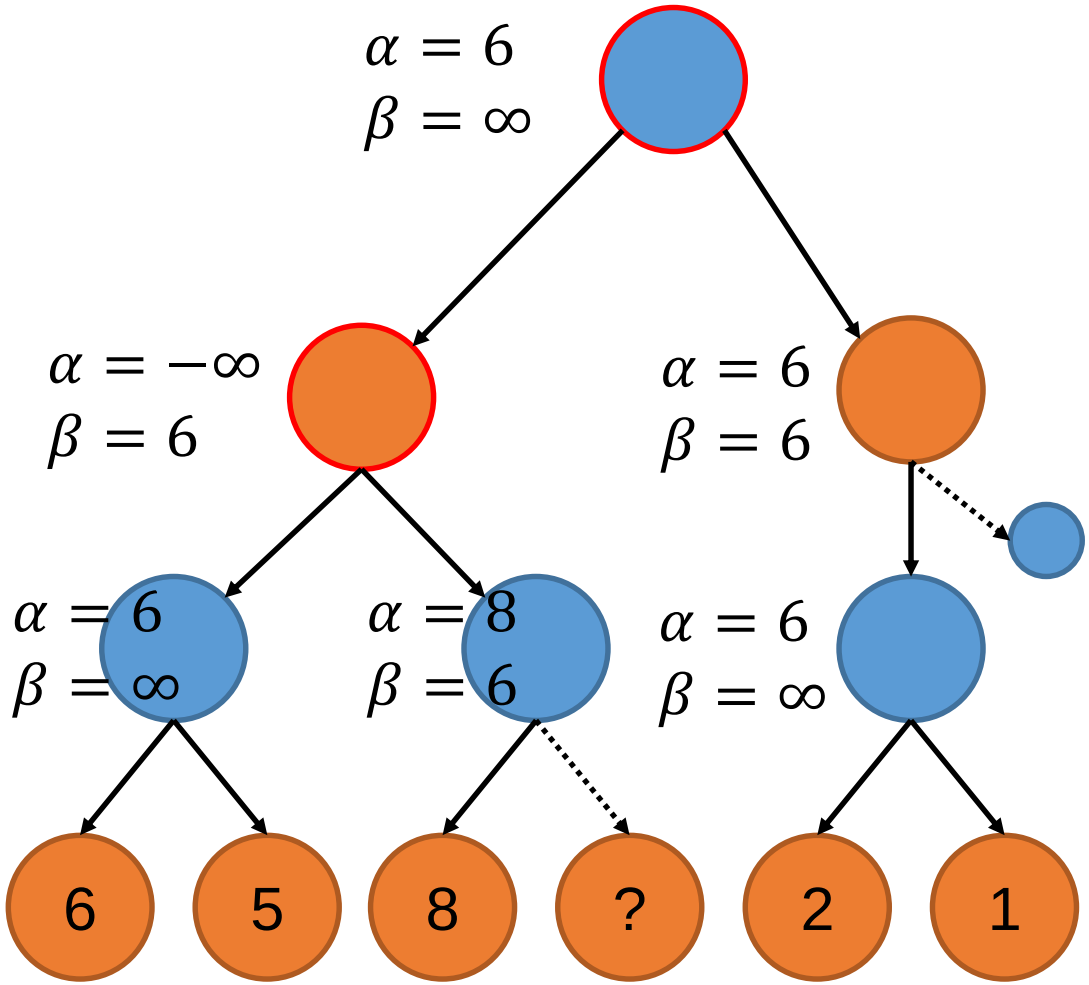

$\alpha = 6$
$\beta = \infty$
$\alpha = -\infty$
$\beta = 6$
$\alpha = 6$
$\beta = \infty$
$\alpha = 6$
$\beta = \infty$
$\alpha = 8$
$\beta = 6$
$\alpha = 6$
$\beta = \infty$
6
5
8
?
2
1


$\alpha = 6$
$\beta = \infty$
$\alpha = -\infty$
$\beta = 6$
$\alpha = 6$
$\beta = \infty$
$\alpha = 6$
$\beta = \infty$
$\alpha = 8$
$\beta = 6$
$\alpha = 6$
$\beta = \infty$
6
5
8
?
2
1


$\alpha = 6$
$\beta = \infty$
$\alpha = -\infty$
$\beta = 6$
$\alpha = 6$
$\beta = \infty$
$\alpha = 6$
$\beta = \infty$
$\alpha = 8$
$\beta = 6$
$\alpha = 6$
$\beta = \infty$
6
5
8
?
2
1


$\alpha = 6$
$\beta = \infty$
$\alpha = -\infty$
$\beta = 6$
$\alpha = 6$
$\beta = 6$
$\alpha = 6$
$\beta = \infty$
$\alpha = 8$
$\beta = 6$
$\alpha = 6$
$\beta = \infty$
6
5
8
?
2
1


$\alpha = 6$
$\beta = \infty$
$\alpha = -\infty$
$\beta = 6$
$\alpha = 6$
$\beta = 6$
$\alpha = 6$
$\beta = \infty$
$\alpha = 8$
$\beta = 6$
$\alpha = 6$
$\beta = \infty$
6
5
8
?
2
1

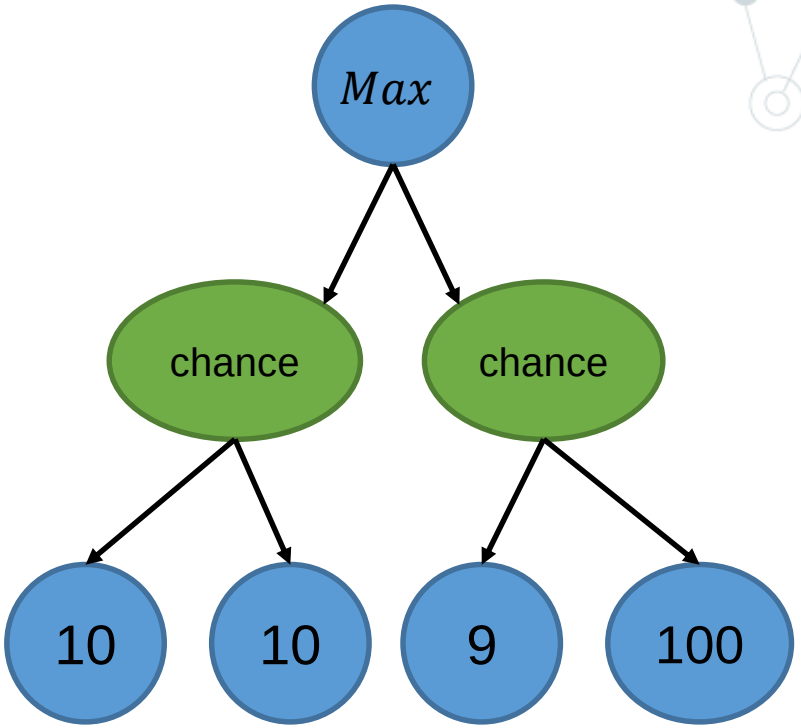

$\alpha = 6$
$\beta = \infty$
$\alpha = -\infty$
$\beta = 6$
$\alpha = 6$
$\beta = 6$
$\alpha = 6$
$\beta = \infty$
$\alpha = 8$
$\beta = 6$
$\alpha = 6$
$\beta = \infty$
6
5
8
?
2
1

Alpha-Beta Pruning Properties

- Pruning does not affect final result
- Good move ordering improves effectiveness of pruning
- With "perfect ordering" time complexity = $O(b \cdot 1 \cdot b \cdot 1 \cdot \ldots) = O\left(b^{\frac{m}{2}}\right)$
- (In worst case, there is no improvement)

Expectimax Search Trees

- What if we don't know what the result of an action will be?
- We can deal with it by considering min nodes as chance nodes – calculate **expected utility** a node by averaging its children
- We can do **expectimax** search to maximize **average score**



Expectimax Search Trees

$value(s)$:

- if s is a max node return $maxValue(s)$
- if s is a chance node return $expValue(s)$
- if s is a terminal node return $utility(s)$

$maxValue(s)$:

- $values = \{ value(s') : s' \in successors(s)\}$
- return $\max(values)$

$expValue(s)$:

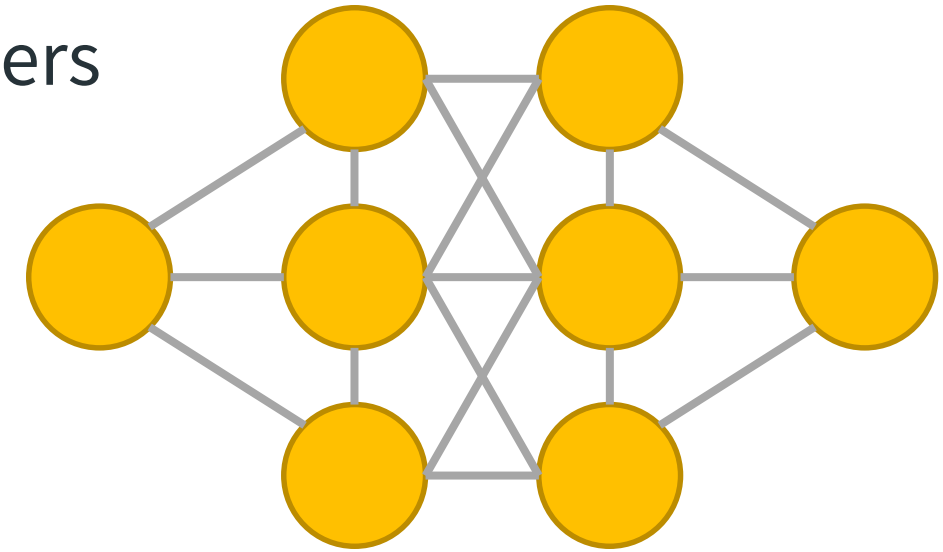
- $values = \{ value(s') : s' \in successors(s)\}$
- Return $\sum_{s'} value(s') \cdot \text{Prob}(s,s')$

Constraint Satisfaction Problems

Motivating Example

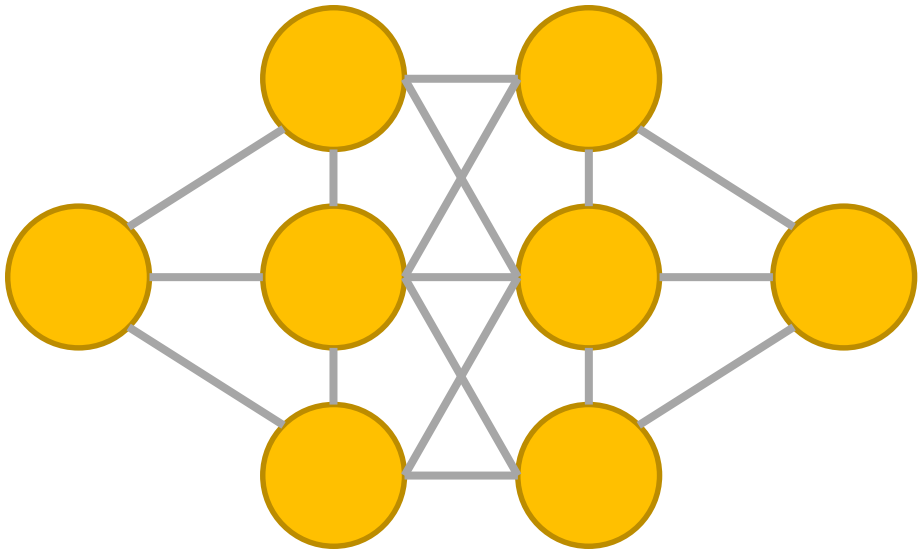
Place numbers 1-8 on nodes

- each number appears exactly once
- no connected nodes have consecutive numbers



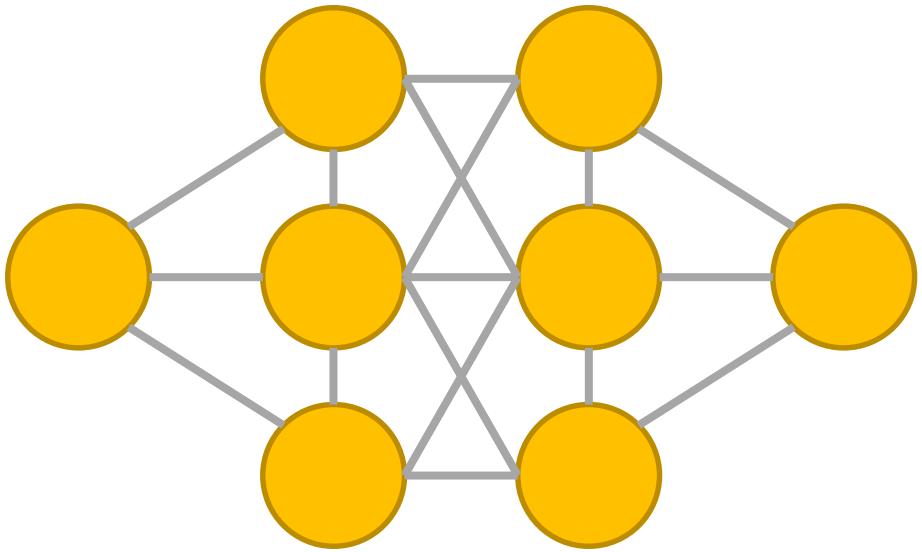
Motivating Example

Guess a value (might need to backtrack)



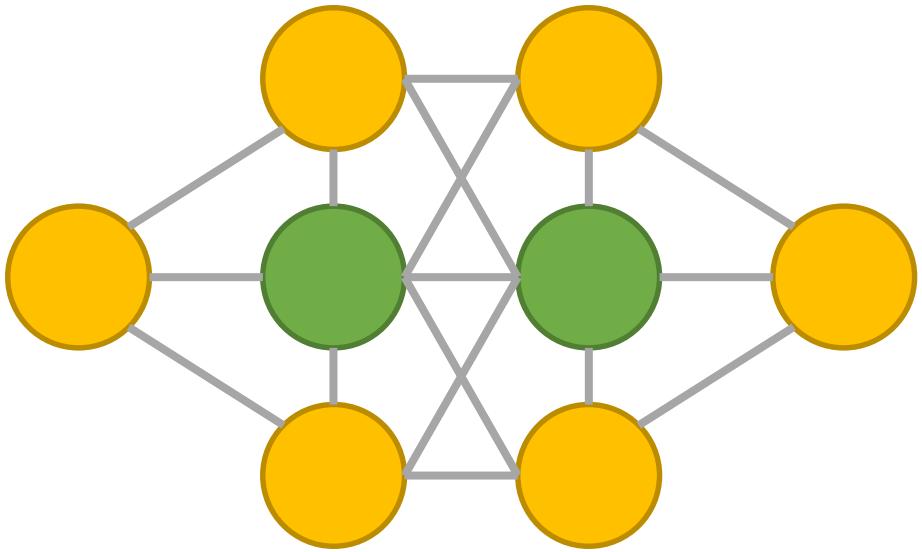
Motivating Example

Which nodes are hardest to number?



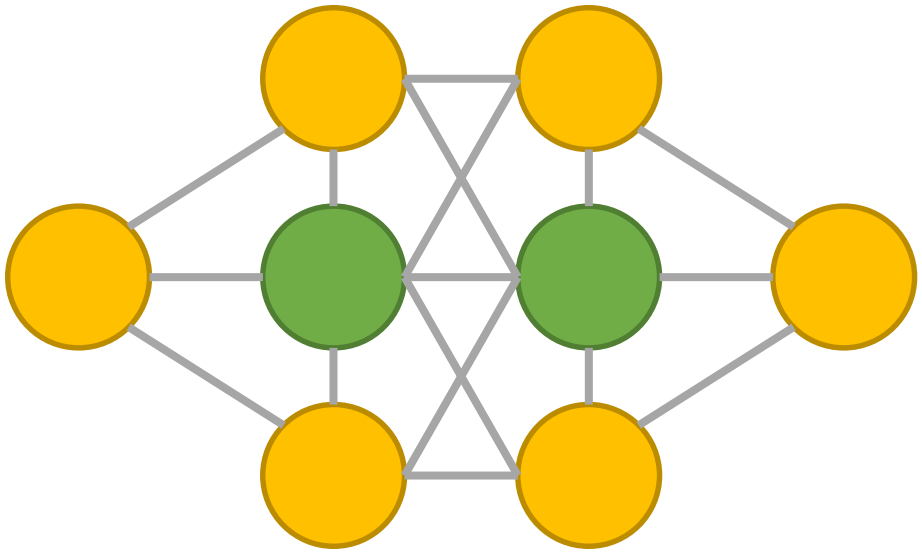
Motivating Example

Which nodes are hardest to number?



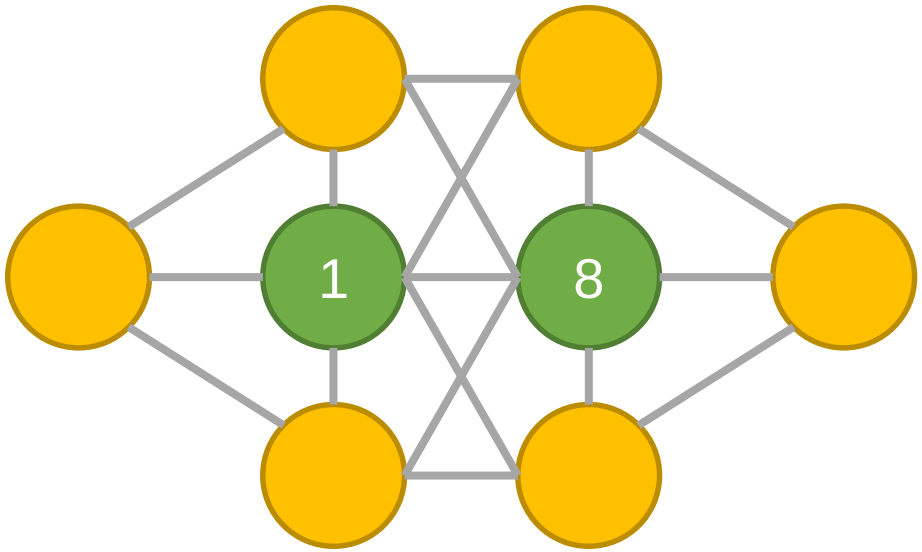
Motivating Example

Which are the least constraining values to use?



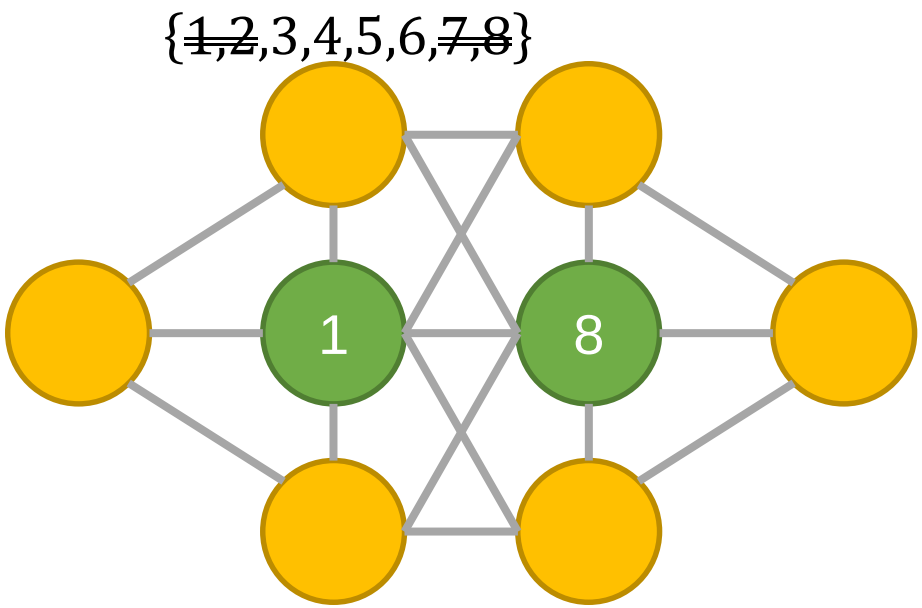
Motivating Example

Least constraining values are 1-8 (symmetric to 8-1)



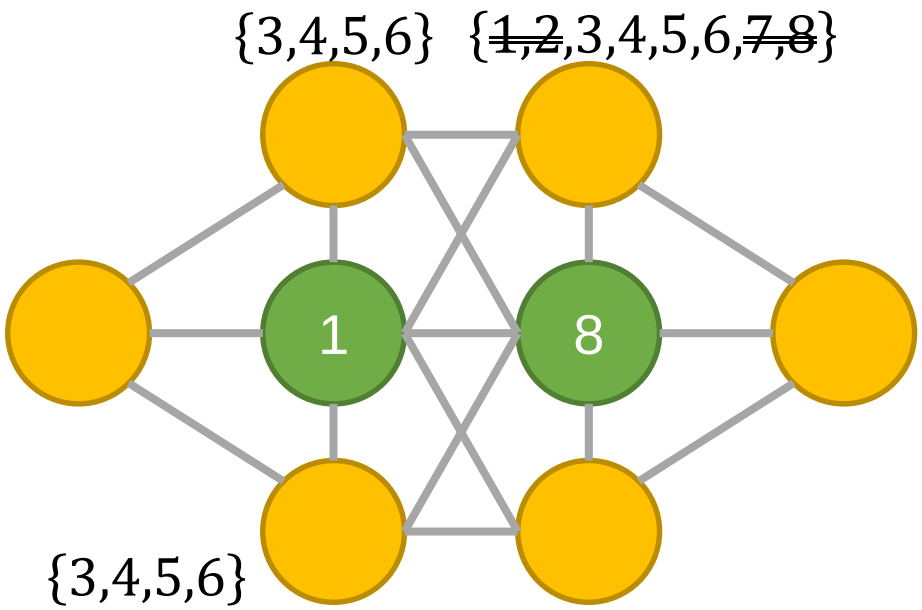
Motivating Example

We can now eliminate many values for other nodes

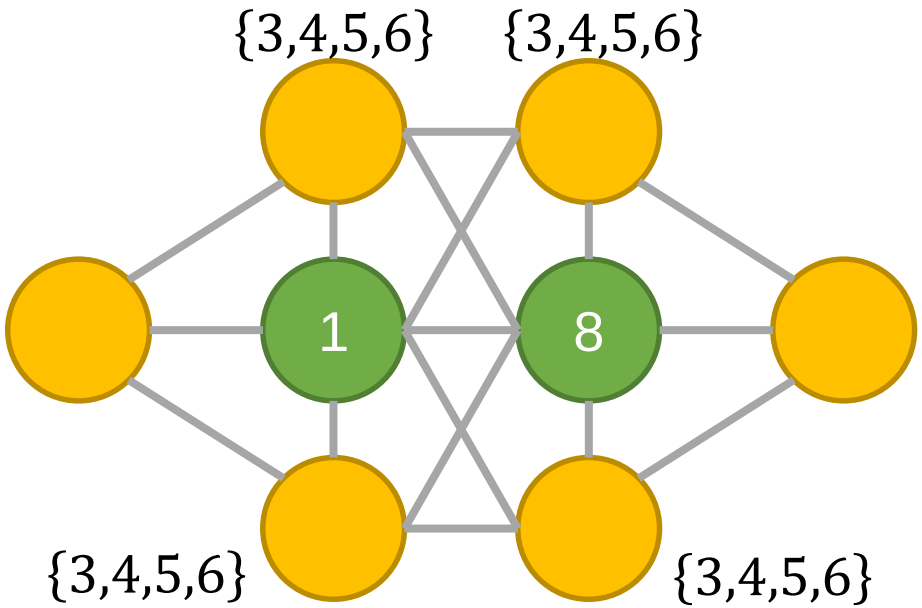


Motivating Example

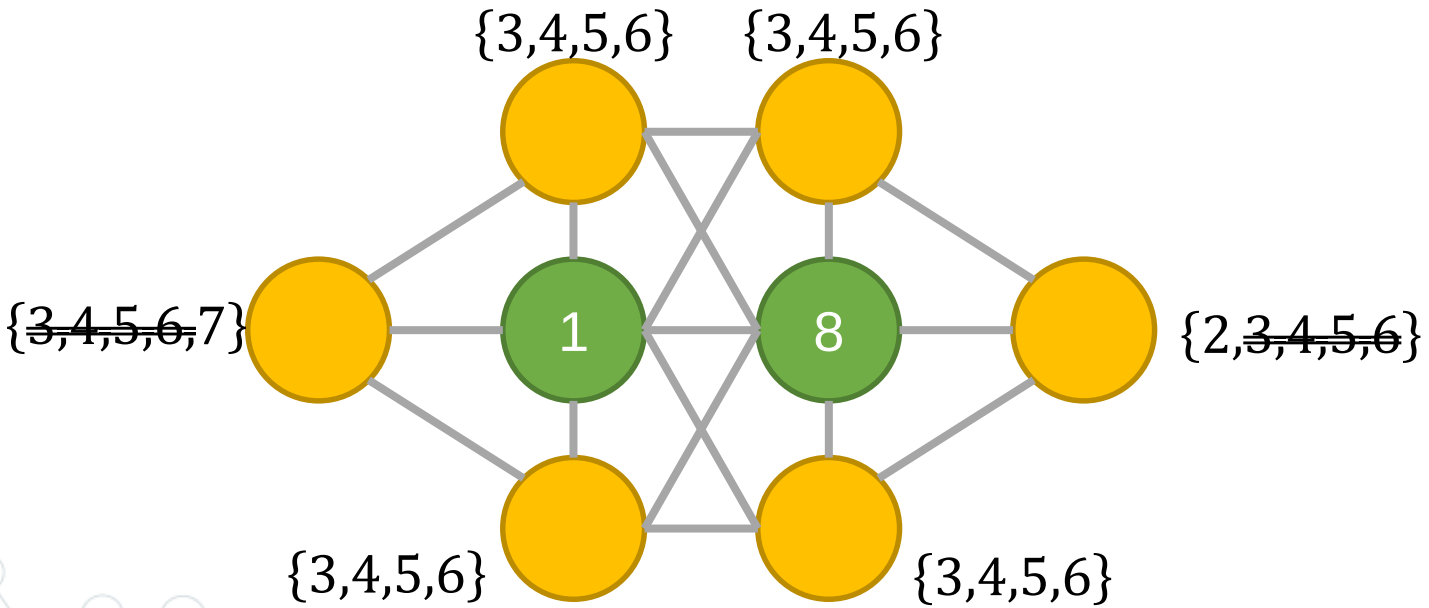
Again, by symmetry



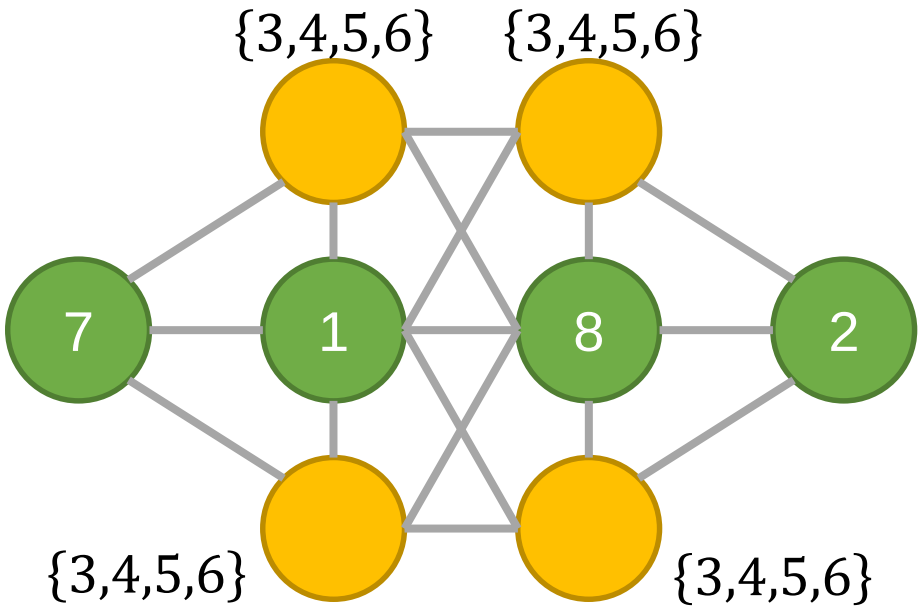
Motivating Example



Motivating Example

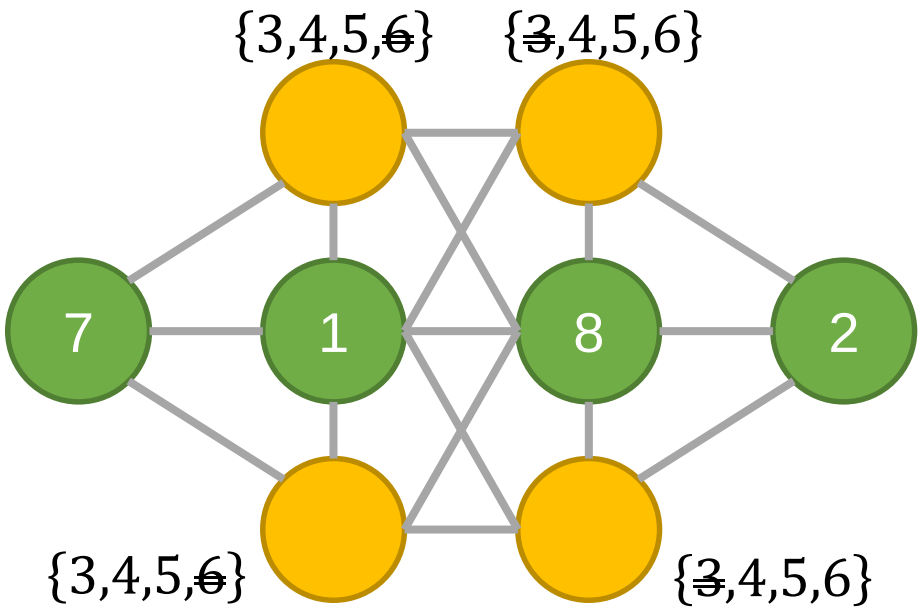


Motivating Example



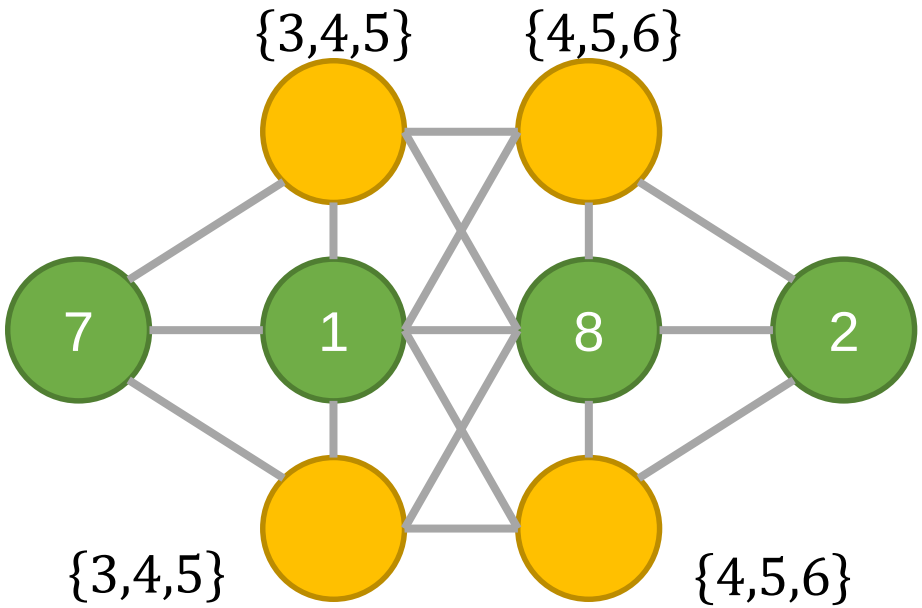
Motivating Example

Propagating the new assignments



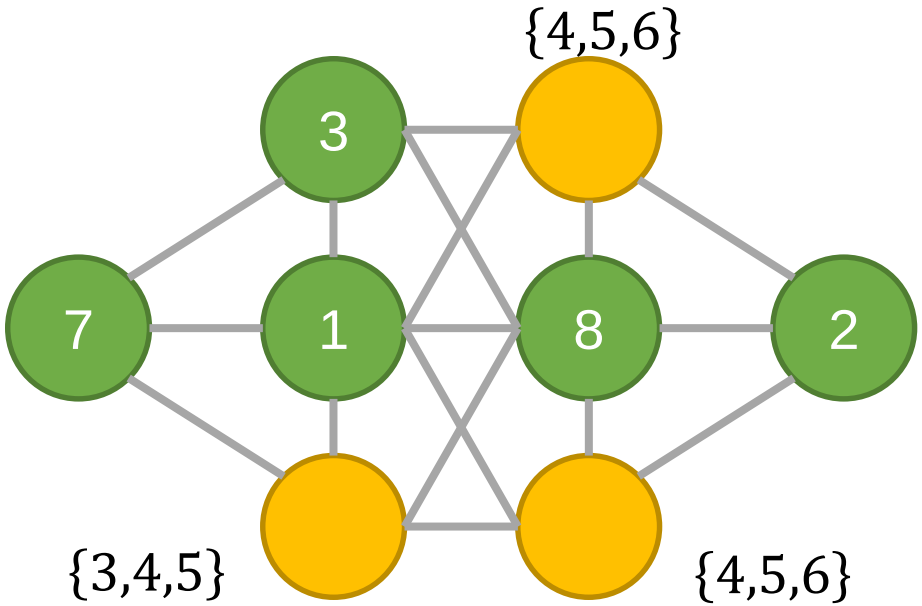
Motivating Example

Assign and prepare to backtrack



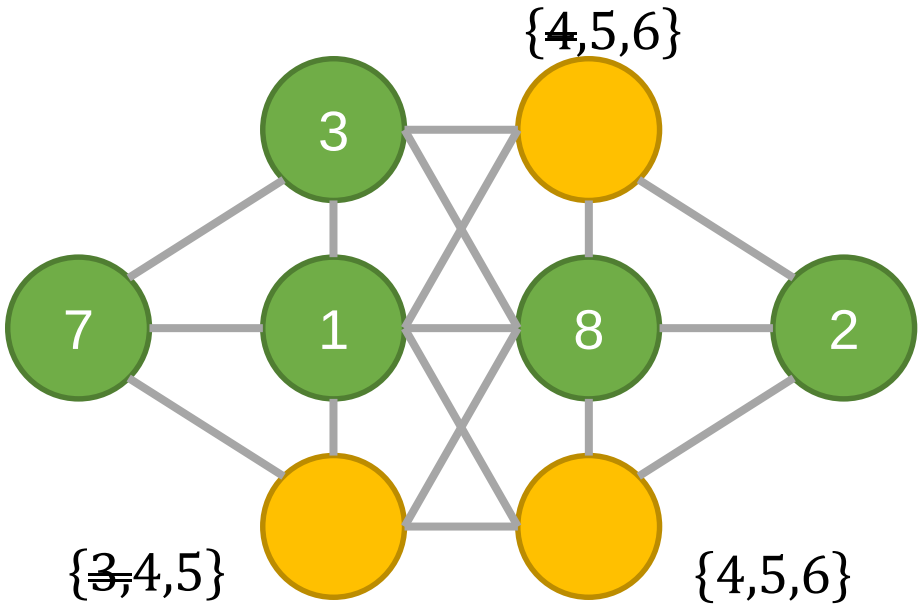
Motivating Example

Assign and prepare to backtrack



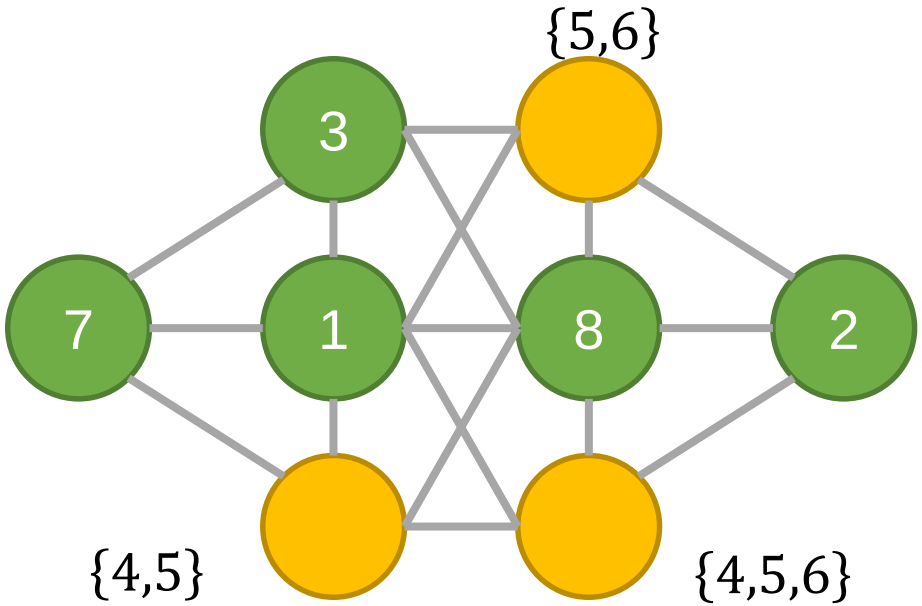
Motivating Example

Assign and prepare to backtrack



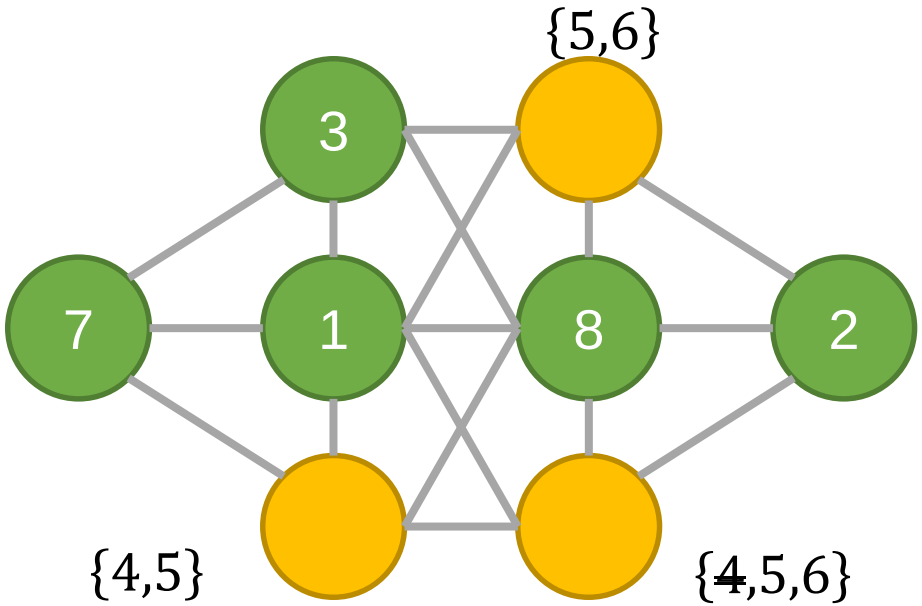
Motivating Example

Can we propagate more?



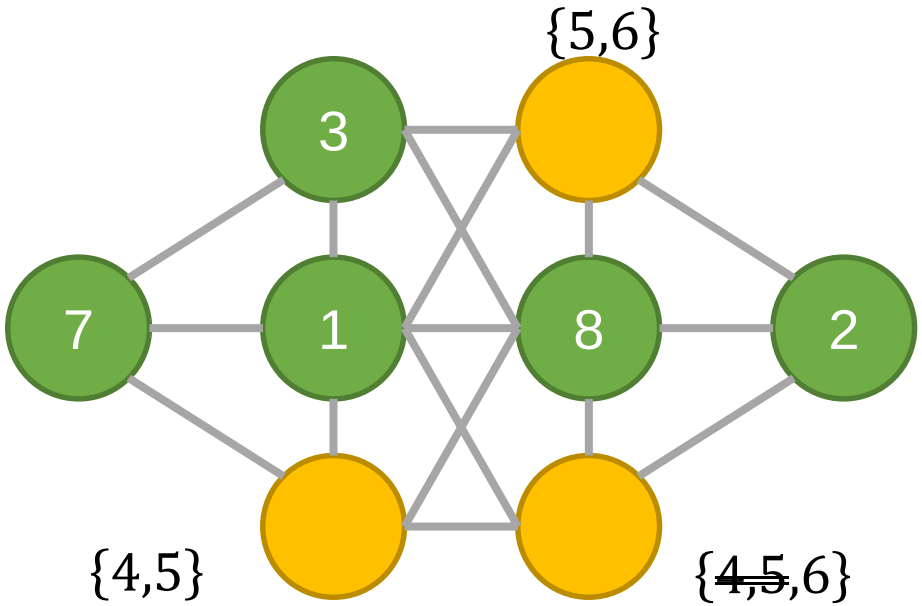
Motivating Example

Can we propagate more?



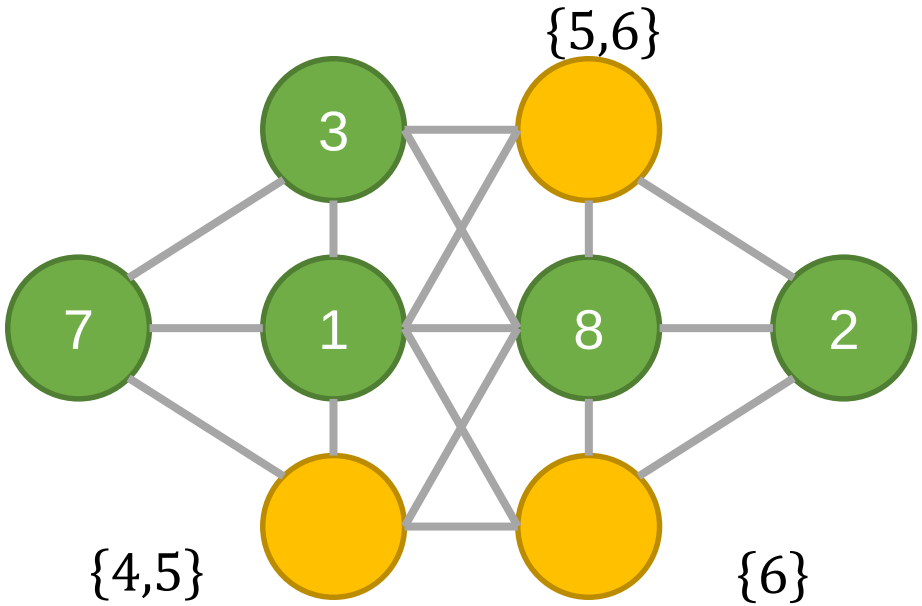
Motivating Example

Can we propagate more?



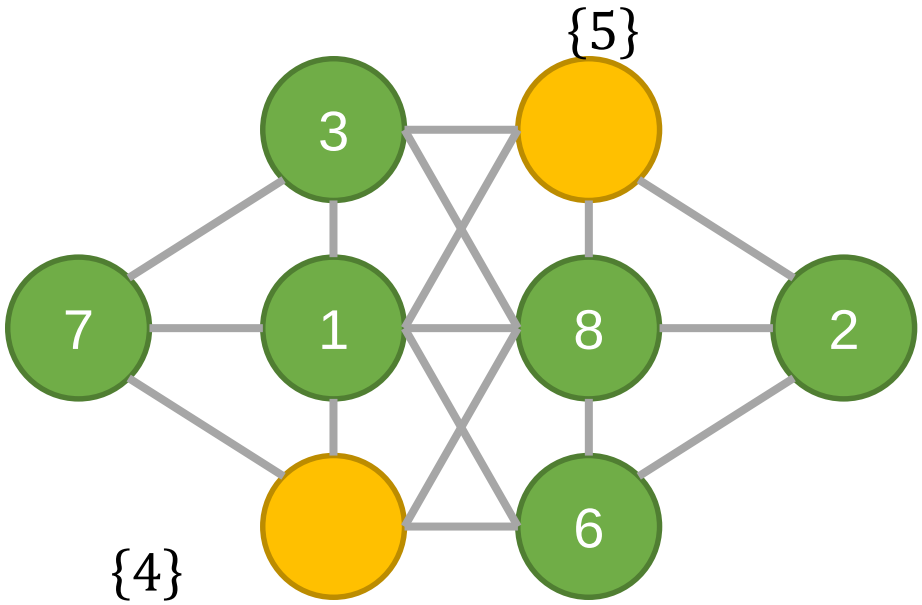
Motivating Example

Can we propagate more?



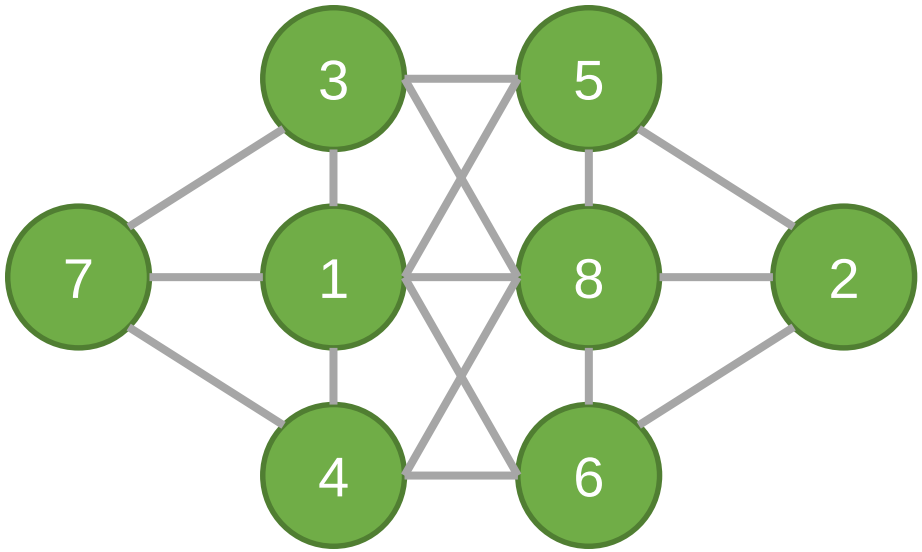
Motivating Example

Can we propagate more?



Motivating Example

Can we propagate more?



Formal Constraint Satisfaction Problem

A set of **variables** $\mathcal{X} = \{x_1, \dots, x_n\}$

A set of **domains** $\mathcal{D} = \{D_1, \dots, D_n\}$

A set **constraints** $\mathcal{C} = \{C_1, \dots, C_m\}$

where $C_j \subset D_1 \times \cdots \times D_n$

Find a **substitution** $x_i \leftarrow d_i$ s.t. :

- $\forall 1 \leq i \leq n, d_i \in D_i$
- $\forall\, 1 \leq i \leq m, (d_1, \dots, d_n) \in C_i$

Note that constraints can be given in a functional form:

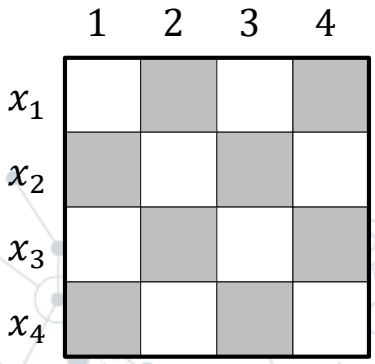
$$C_j : D_1 \times \cdots \times D_n \rightarrow \{0,1\}$$

- $\forall 1 \leq i \leq m, C_j(d_1, \dots, d_n) = 1$

Example – 4-Queens

The 4-Queens problem is $\langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$ where

- Variables: $\mathcal{X} = \{x_1, x_2, x_3, x_4\}$
- Domains: $\mathcal{D} = \{D_1, D_2, D_3, D_4\}$, and $\forall i \ , D_i = \{1,2,3,4\}$
- Constraints: $\mathcal{C} = \{R_{1,2}, R_{1,3}, R_{1,4}, R_{2,3}, R_{2,4}, R_{3,4}\}$



- $R_{1,2} = \{(1,3), (1,4), (2,4), (3,1), (4,1), (4,2)\}$
- $R_{1,3} = \{(1,2), (1,4), (2,1), (2,3), (3,4), (4,1), (4,3)\}$
- $R_{1,4} = \{(1,2), (1,4), (2,1), (2,3), (2,4), (3,1), (3,2), (3,4), (4,2), (4,3)\}$
- $R_{2,3} = \{(1,3), (1,4), (2,4), (3,1), (4,1), (4,2)\}$
- $R_{2,4} = \{(1,2), (1,4), (2,1), (2,3), (3,2), (3,4), (4,1), (4,3)\}$
- $R_{3,4} = \{(1,3), (1,4), (2,4), (3,1), (4,1), (4,2)\}$

*Note the abuse of notation!

More Definitions

- **Unary Constraint** – A constrain containing only one variable
- **Binary Constraint** – A constrain containing two variables
- **Binary CSP** – A CSP with only unary or binary constrains

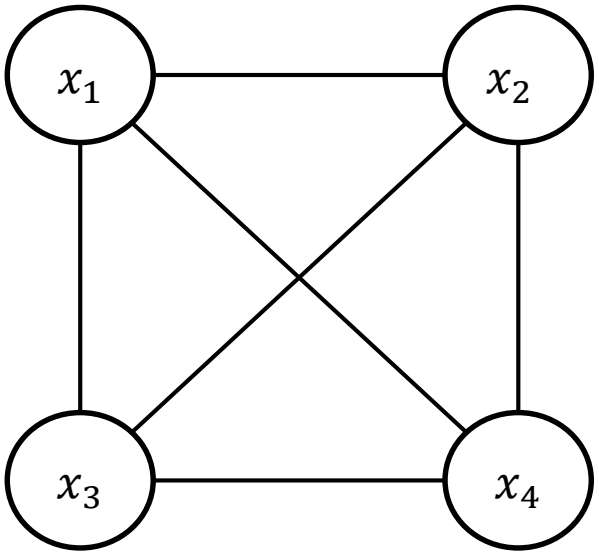
CSP Dependency Graph

Given a Binary CSP we define the **Constraint Graph** as follows:

- $V = \{x_1, \ldots, x_n\}$
- $(x_i, x_j) = e_k \in E \Leftrightarrow \exists (d_1, \ldots, d_n) \in C_k \wedge$
  $\exists\, d_i', d_j'$:
  $(d_1, \ldots, d_i', \ldots, d_n)$,
  $(d_1, \ldots, d_j', \ldots, d_n) \notin C_k$

Example – 4-Queens Dependency Graph

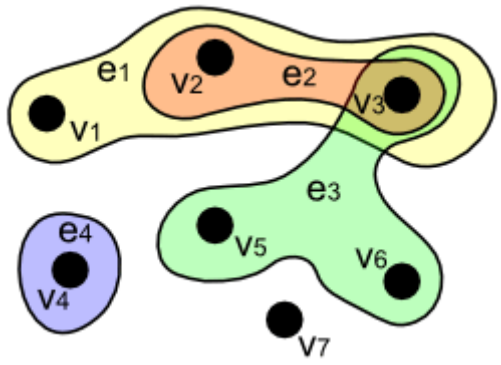
- The 4-Queens constraint graph



CSP Dependency Graph

Given a Binary CSP we define the **Constraint Graph** as follows:

- $V = \{x_1, \ldots, x_n\}$
- $(x_i, x_j) = e_k \in E \Leftrightarrow \exists (d_1, \ldots, d_n) \in C_k \wedge$
  $\exists\, d_i', d_j'$:
  $(d_1, \ldots, d_i', \ldots, d_n)$,
  $(d_1, \ldots, d_j', \ldots, d_n) \notin C_k$



- In the same way for a general CSP we may define the **Constraint Hyper-Graph**

Backtracking Search

Function: Backtracking Search

Input: A CSP problem $\langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$

Output: Either a solution or a failure

$i \leftarrow 1$ initialize variable counter

$D_i' \leftarrow D_i$ copy domain

While $1 \leq i \leq n$

- $x_i \leftarrow SELECT - VALUE\big((x_1, \dots, x_{i-1}), D_i'\big)$
- If x_i is $null$ no value was returned
  - $i \leftarrow i - 1$ backtrack
- Else
  - $i \leftarrow i + 1$ step forward
  - $D_i' \leftarrow D_i$.

If $i = 0$

- Return "failure“

Else

- Return $x_1, \dots, x_n$

Backtracking Search

Function: SELECT-VALUE

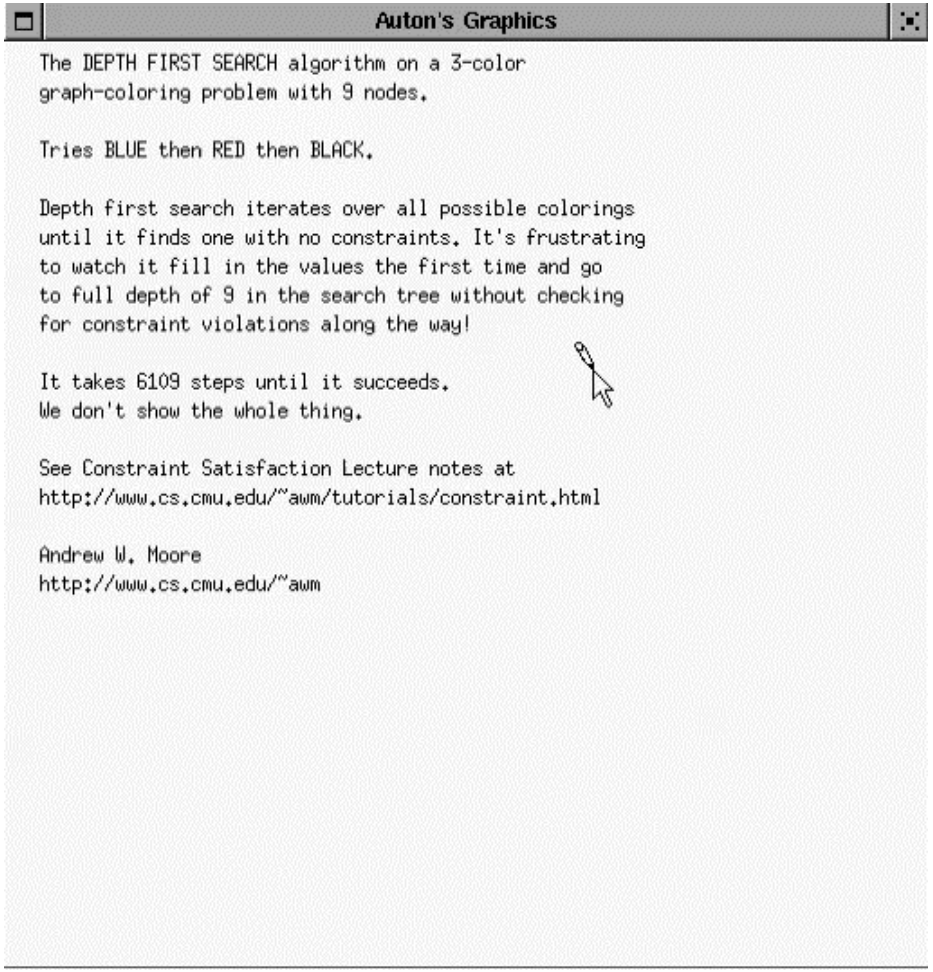
Output: A value in D_i' consistent with $x_1, \ldots, x_{i-1}$

While D_i' is not empty

- select an arbitrary element $x_i \in D_i'$ and remove it from D_i'
- If x_i is consistent with $x_1, \ldots, x_{i-1}$
- Return x

Return $null$

Naive Backtracking Search Demonstration



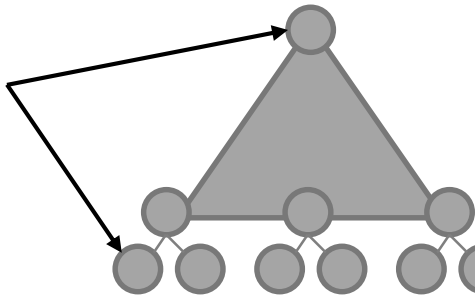
Thrashing

- Note that if the backtracking is performed naively, the same failure can be rediscovered an exponential number of times
- An efficient cure for thrashing in all cases is unlikely, since the problem is NP-complete

But we can:

- Employ preprocessing algorithms that reduce the size of the underlying search space
- Dynamically improve the search control strategy
  - look-ahead schemes
  - look-back schemes

The first choice is incomputable with any last choice



Look-ahead & Look-back

Lookahead – invoked whenever the algorithm is preparing to assign a value to the next variable

- I.e. how current decisions about variable and value selection will restrict future search

Lookback – invoked when the algorithm prepares to backtrack after encountering a dead-end

- I.e. decide intelligently how deep to backtrack

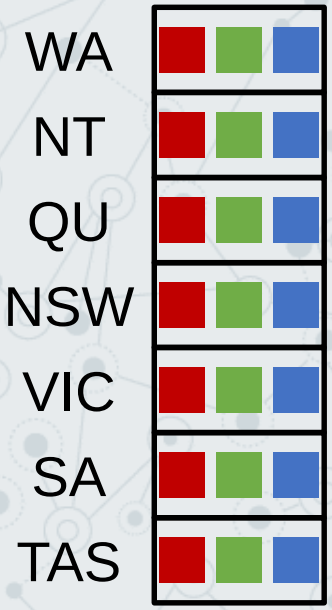
We will discuss these schemes shortly, but first some heuristics.

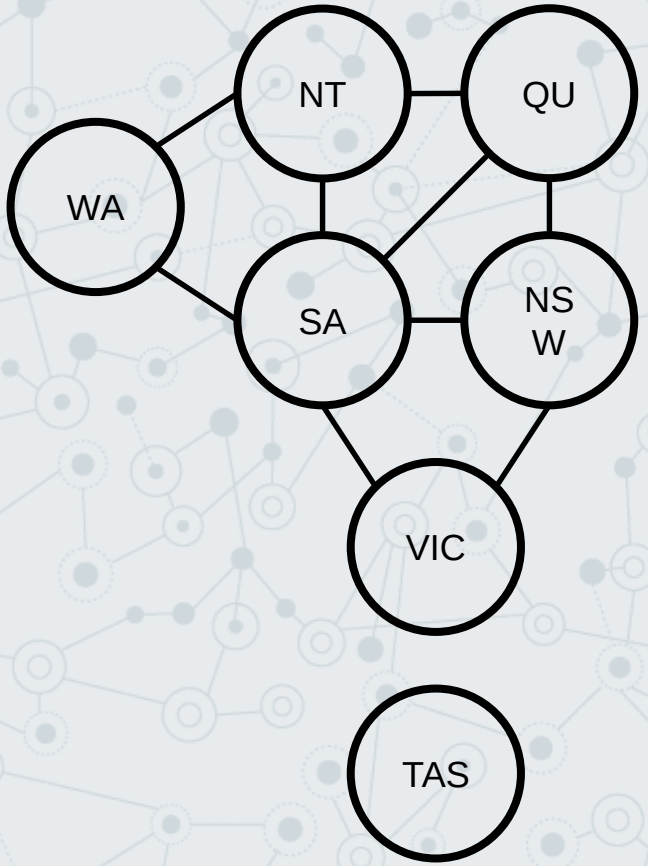
CSP Heuristics (Variable/Value Ordering)

- Complete Formulation Heuristics
  - **Minimum-Conflict**: Select value that minimizes number of constraint violations
- Incremental Formulation Heuristics
  - Heuristics for choosing variable
    - **Minimum Remaining Value**: Select most constrained variable (fewest legal values) first
    - **Degree Heuristic**: Choose the variable with most constraints on remaining values (MRV tie-breaker)
  - Heuristics for choosing value
    - **Least Constrained Value**: Select value least restricting the neighboring variables

Minimum Remaining Values Example – Map Coloring

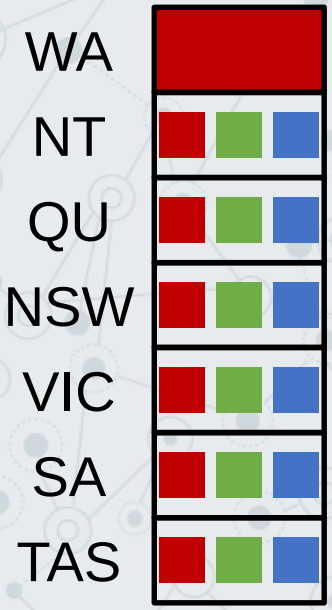
Minimum remaining values (MRV): choose the variable with the fewest legal values

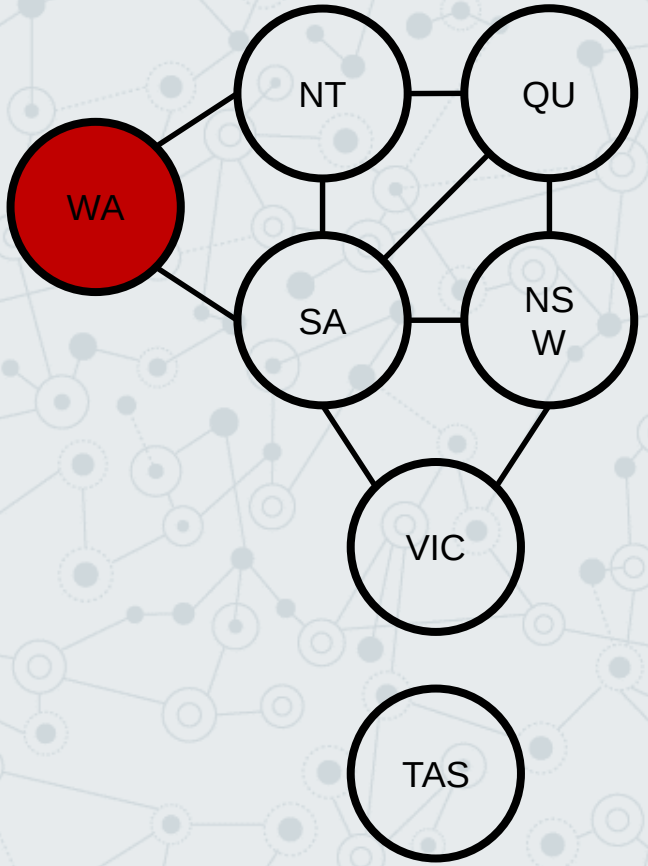




Minimum Remaining Values Example – Map Coloring

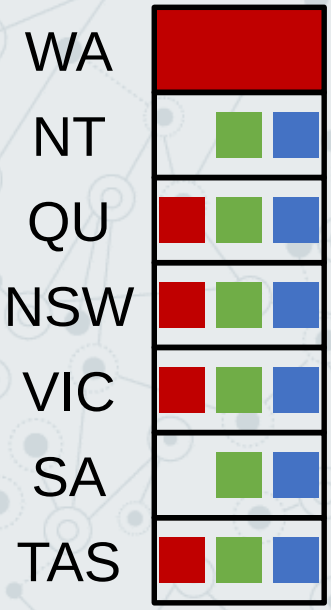
Minimum remaining values (MRV): choose the variable with the fewest legal values

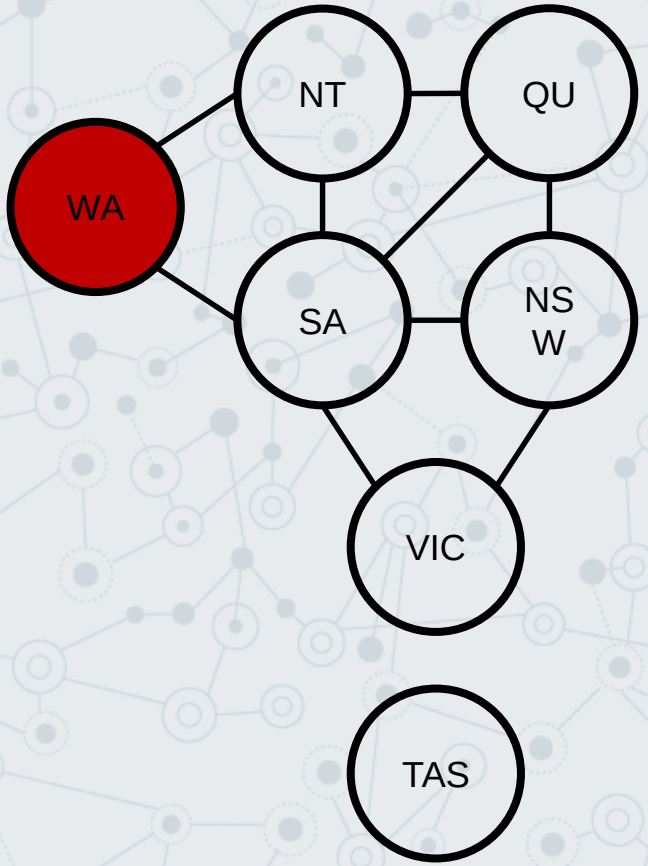




Minimum Remaining Values Example – Map Coloring

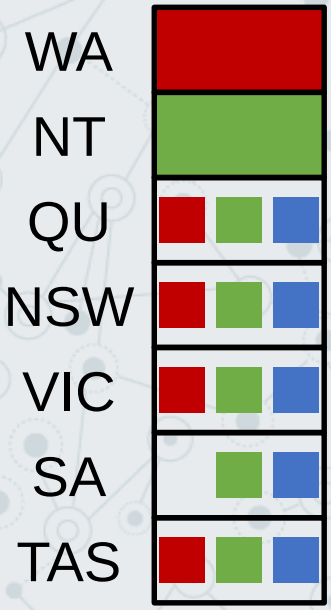
Minimum remaining values (MRV): choose the variable with the fewest legal values

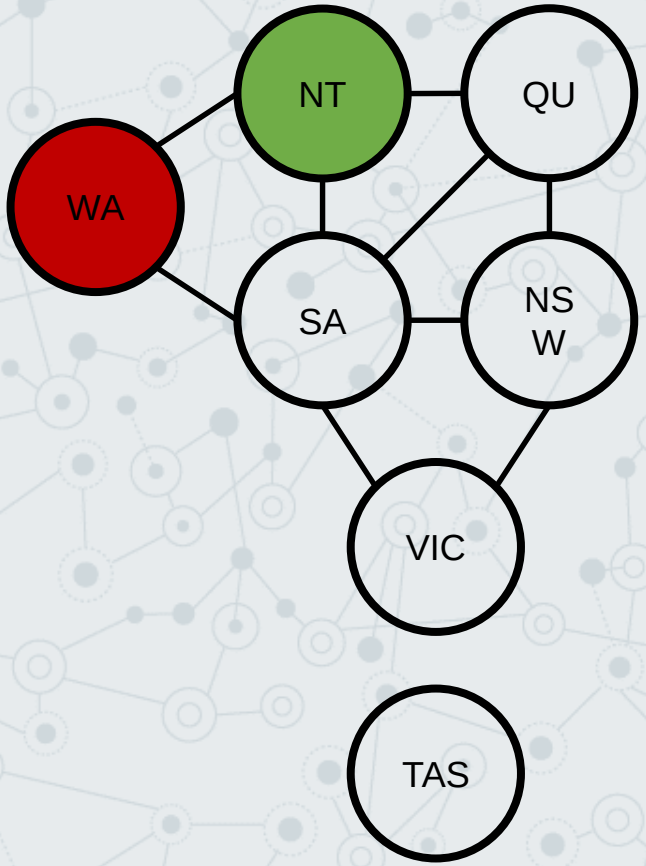




Minimum Remaining Values Example – Map Coloring

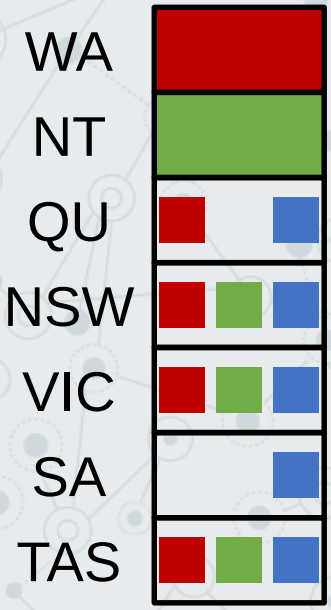
Minimum remaining values (MRV): choose the variable with the fewest legal values

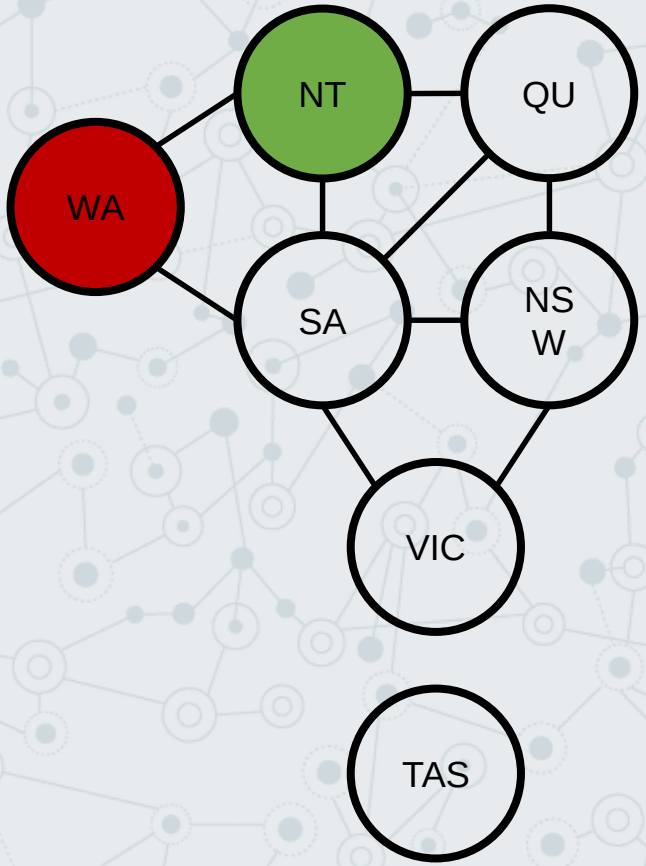




Minimum Remaining Values Example – Map Coloring

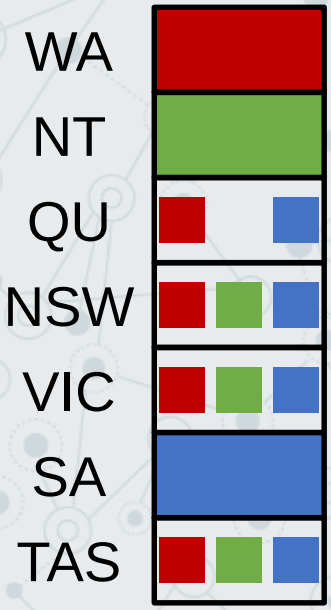
Minimum remaining values (MRV): choose the variable with the fewest legal values

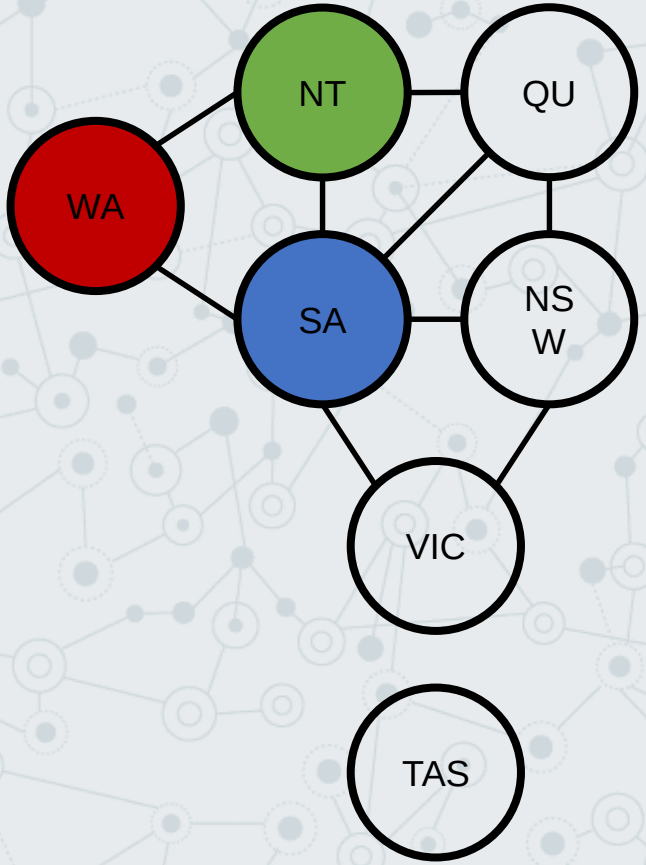




Minimum Remaining Values Example – Map Coloring

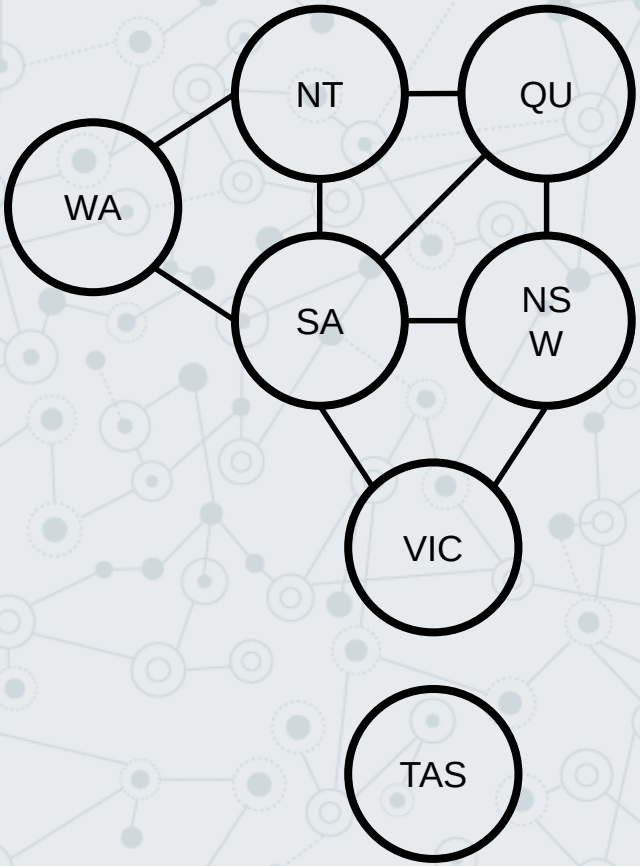
Minimum remaining values (MRV): choose the variable with the fewest legal values

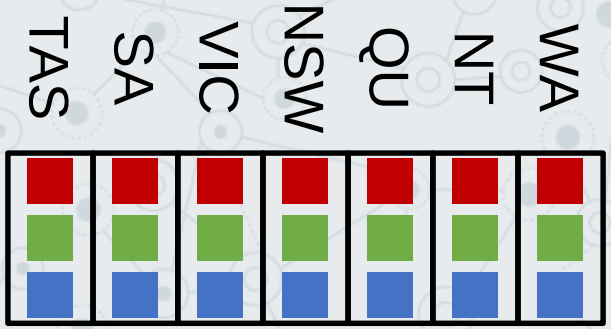




Degree Heuristic Example – Map Coloring

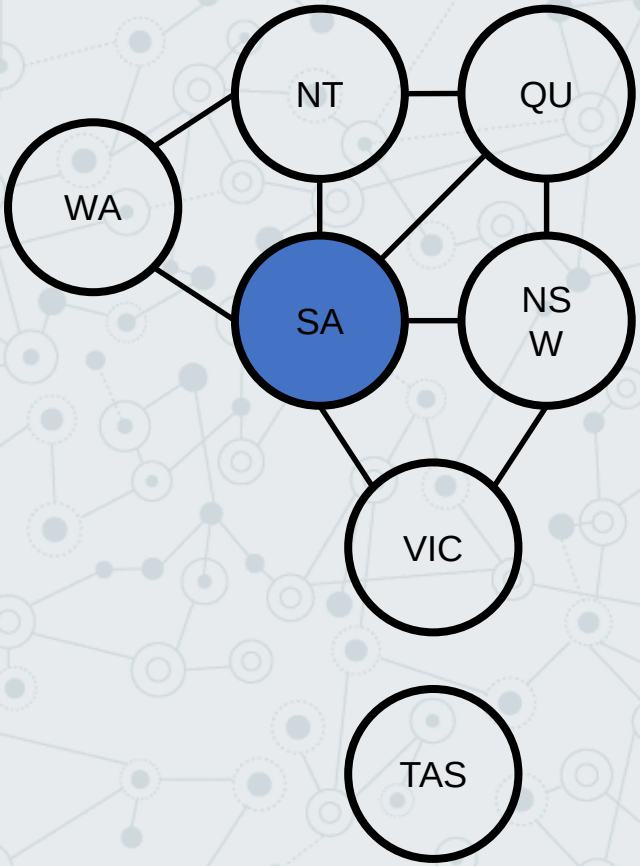
Degree heuristic: choose the variable with the most constraints on remaining variables (used as tie-breaker among MRV variables)

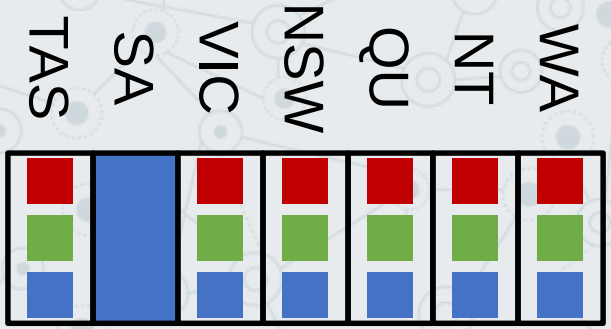




Degree Heuristic Example – Map Coloring

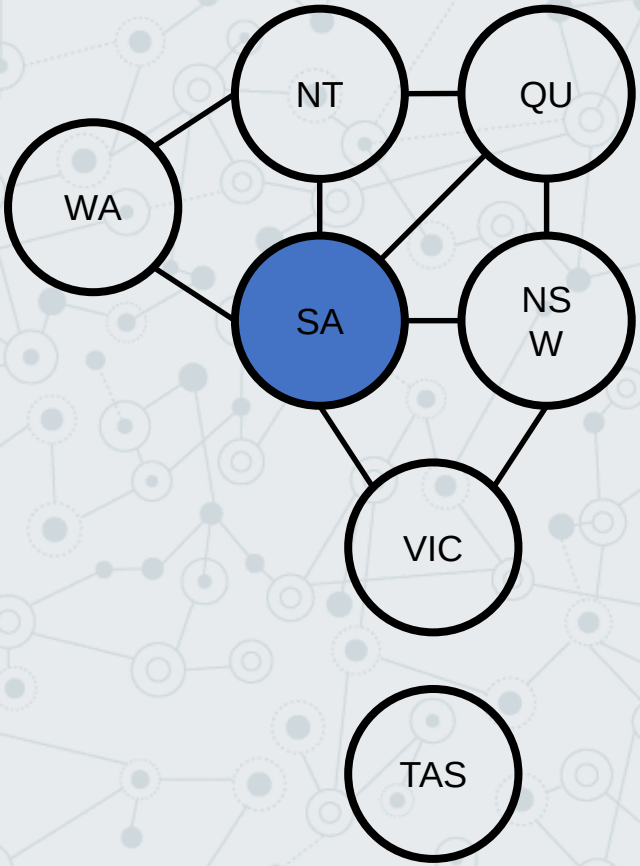
Degree heuristic: choose the variable with the most constraints on remaining variables (used as tie-breaker among MRV variables)

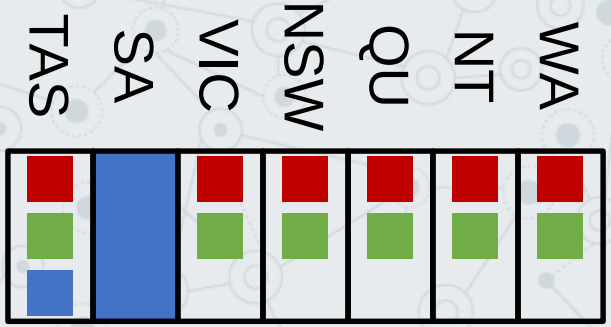




Degree Heuristic Example – Map Coloring

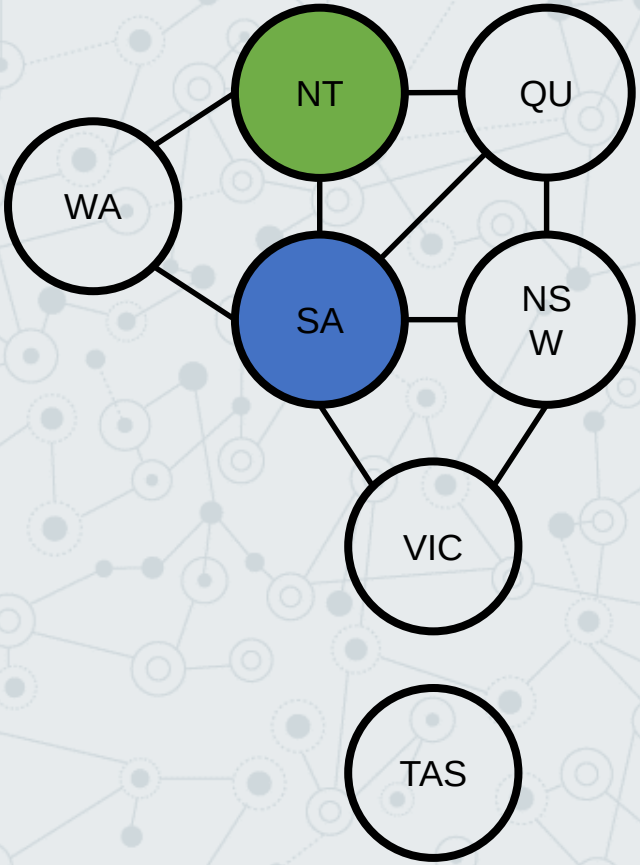
Degree heuristic: choose the variable with the most constraints on remaining variables (used as tie-breaker among MRV variables)

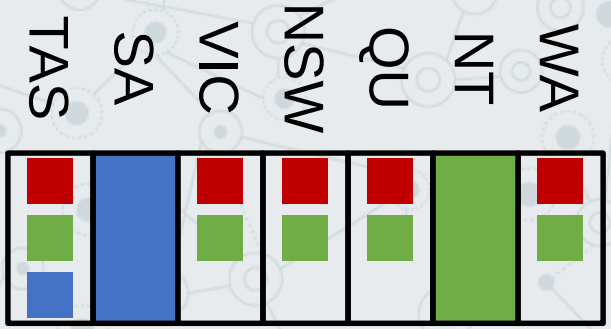




Degree Heuristic Example – Map Coloring

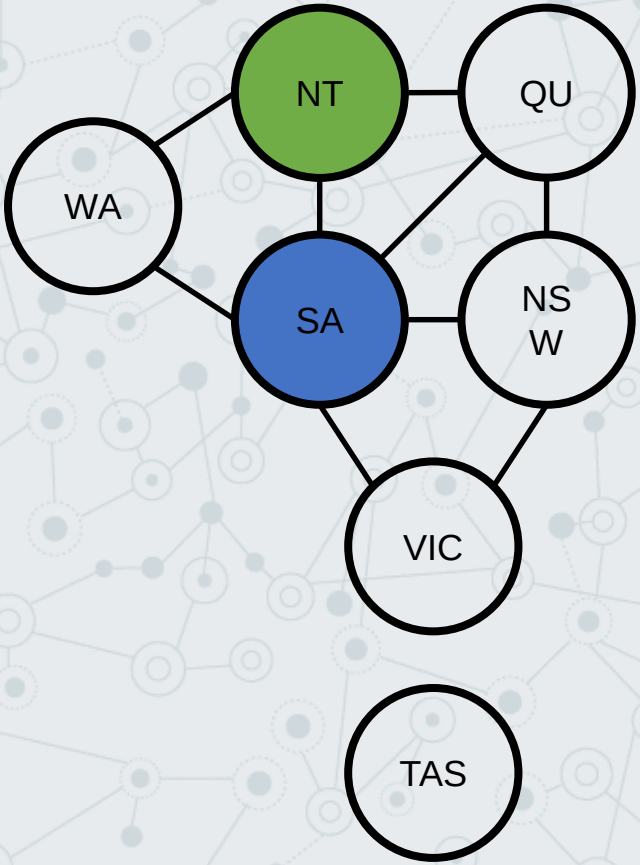
Degree heuristic: choose the variable with the most constraints on remaining variables (used as tie-breaker among MRV variables)

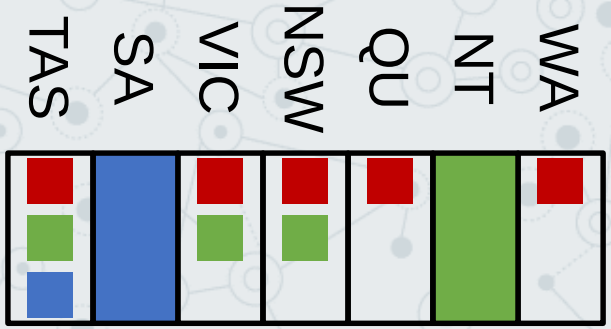




Degree Heuristic Example – Map Coloring

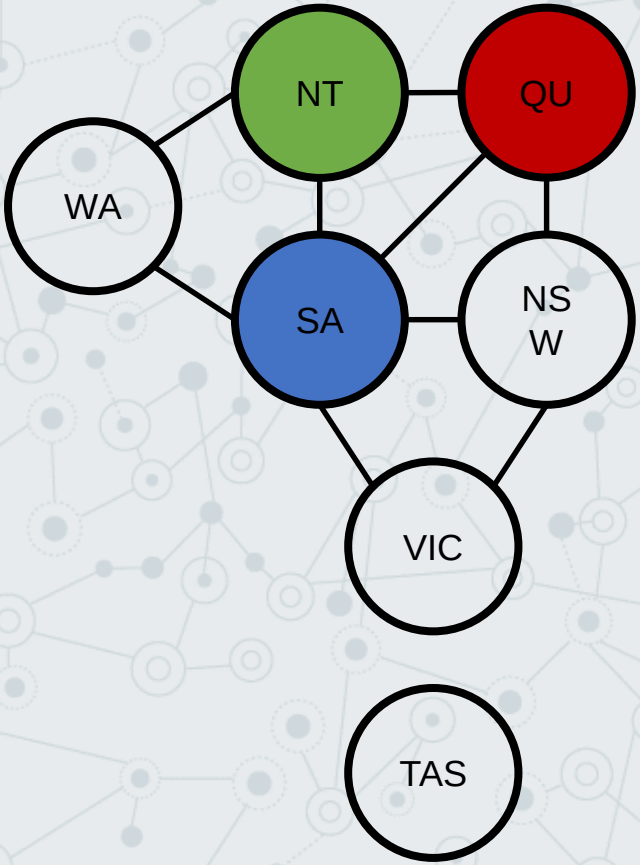
Degree heuristic: choose the variable with the most constraints on remaining variables (used as tie-breaker among MRV variables)

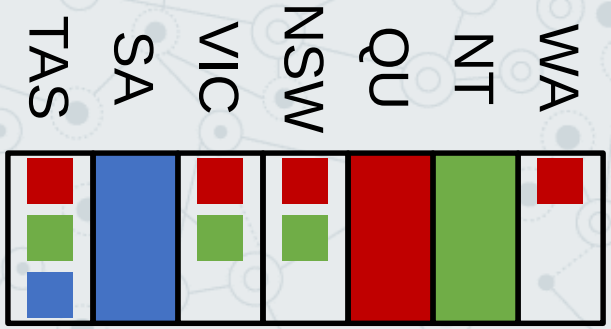




Degree Heuristic Example – Map Coloring

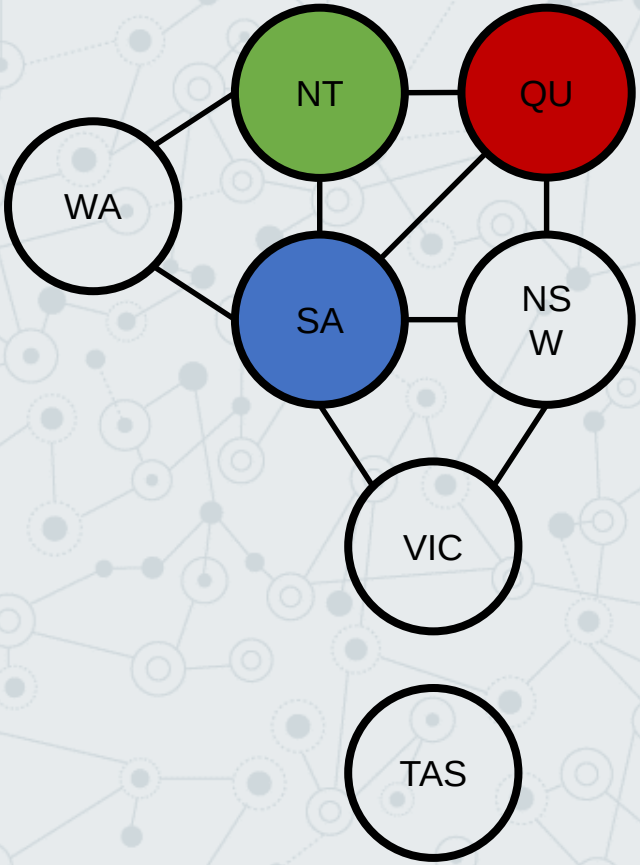
Degree heuristic: choose the variable with the most constraints on remaining variables (used as tie-breaker among MRV variables)

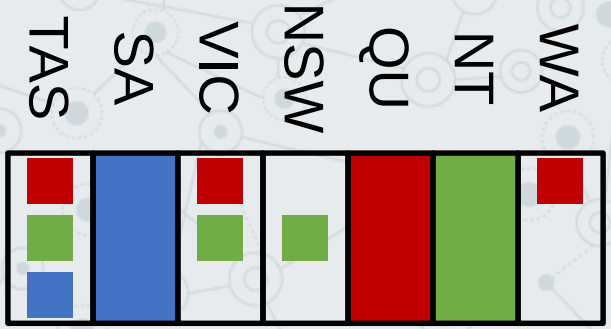




Degree Heuristic Example – Map Coloring

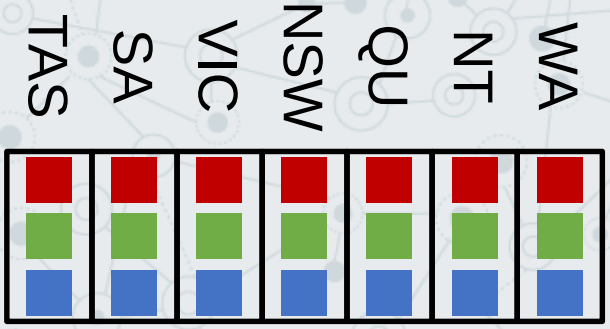
Degree heuristic: choose the variable with the most constraints on remaining variables (used as tie-breaker among MRV variables)

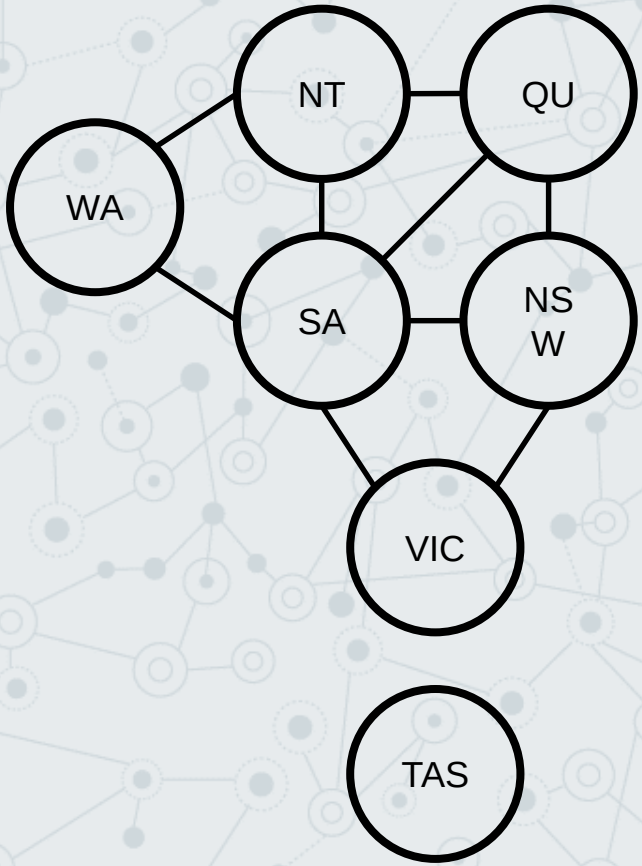




Least Constraining Value Example – Map Coloring

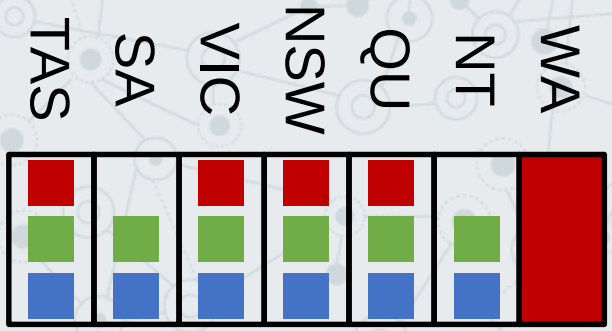
Given a variable, choose the **least constraining value**: the one that rules out the fewest values in the remaining variables

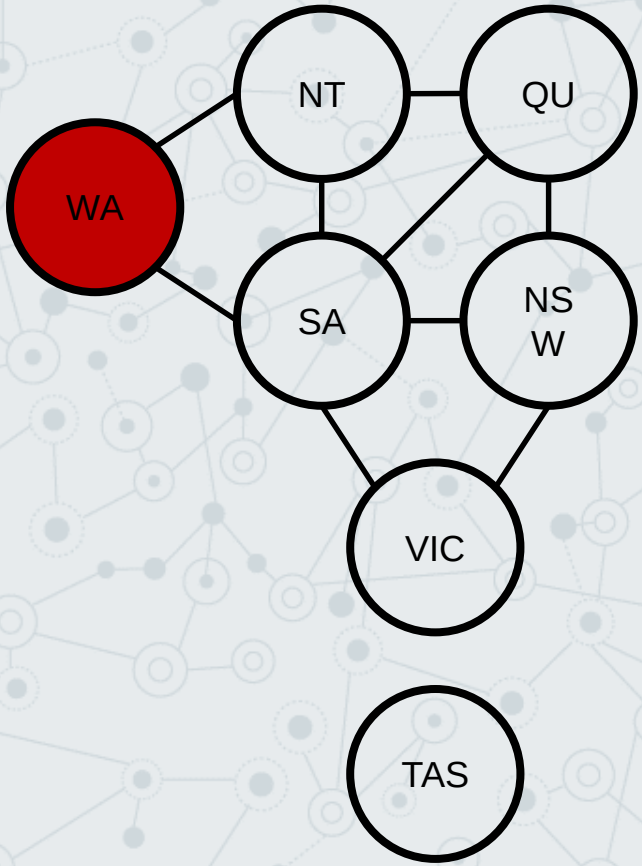




Least Constraining Value Example – Map Coloring

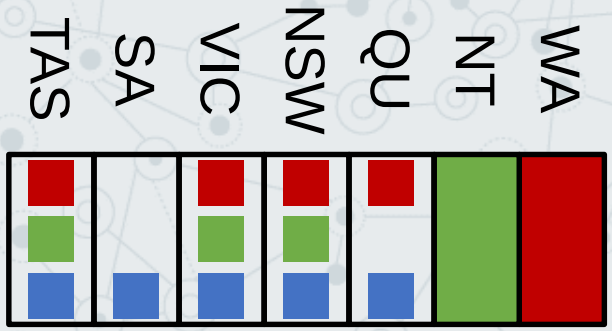
Given a variable, choose the **least constraining value**: the one that rules out the fewest values in the remaining variables

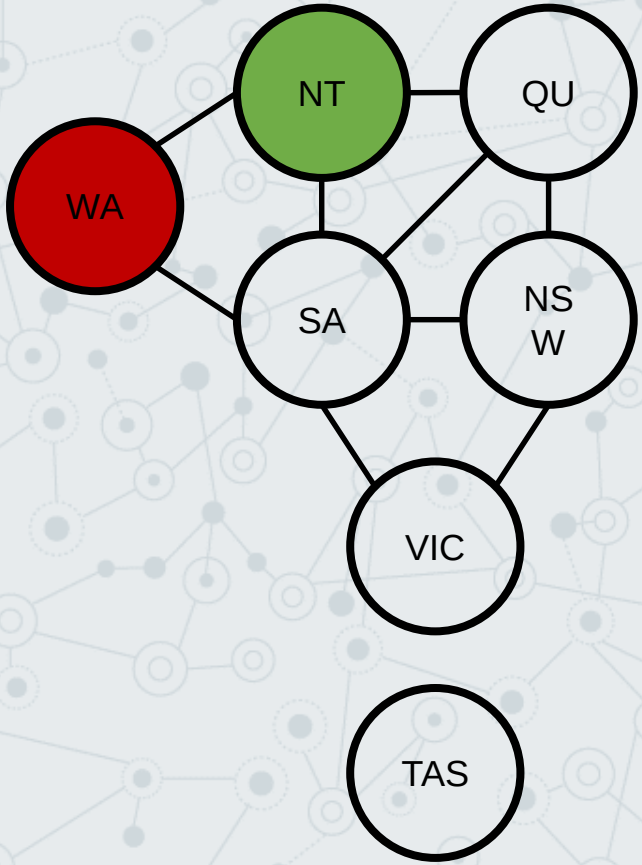




Least Constraining Value Example – Map Coloring

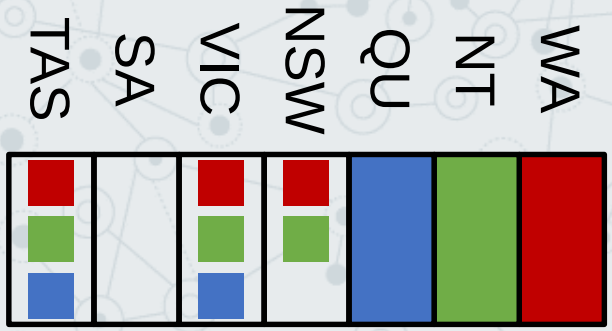
Given a variable, choose the **least constraining value**: the one that rules out the fewest values in the remaining variables

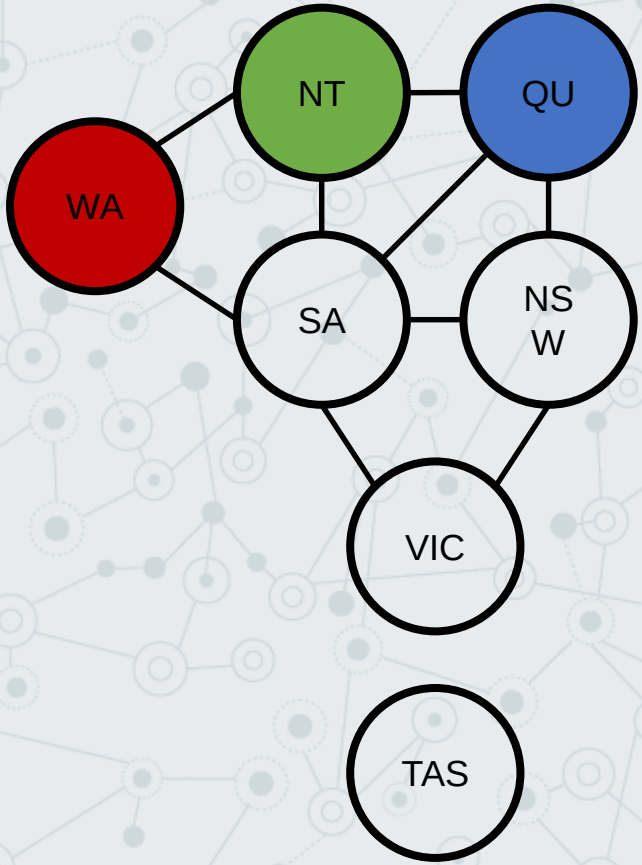




Least Constraining Value Example – Map Coloring

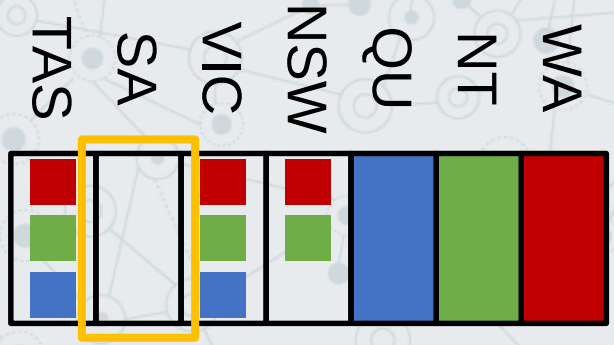
Given a variable, choose the **least constraining value**: the one that rules out the fewest values in the remaining variables

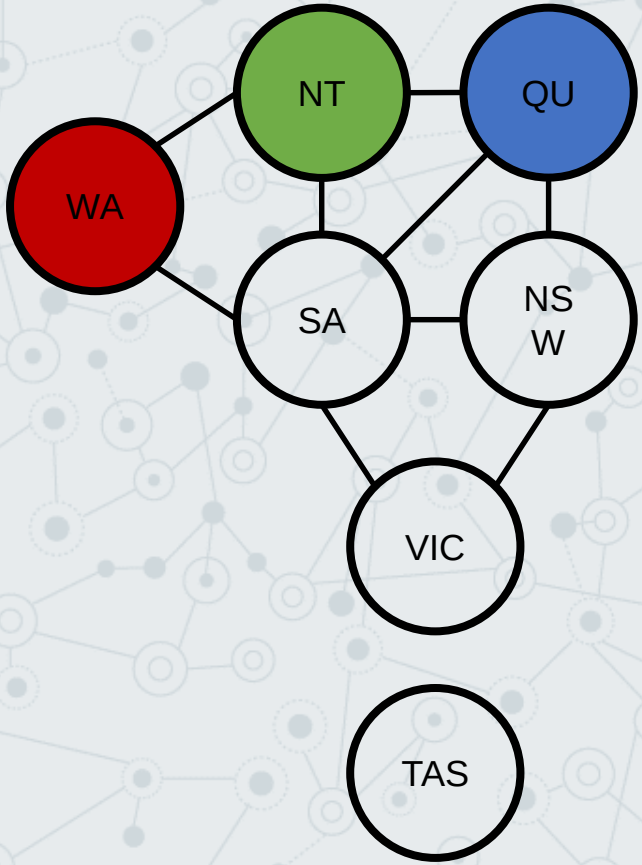




Least Constraining Value Example – Map Coloring

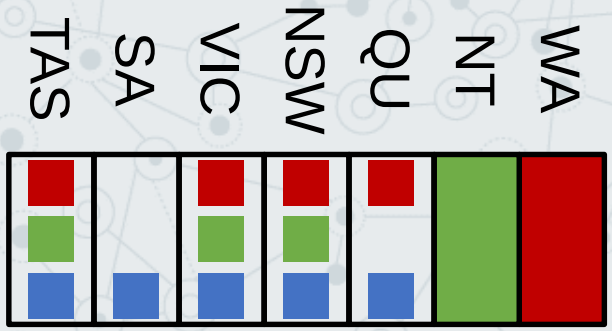
Given a variable, choose the **least constraining value**: the one that rules out the fewest values in the remaining variables

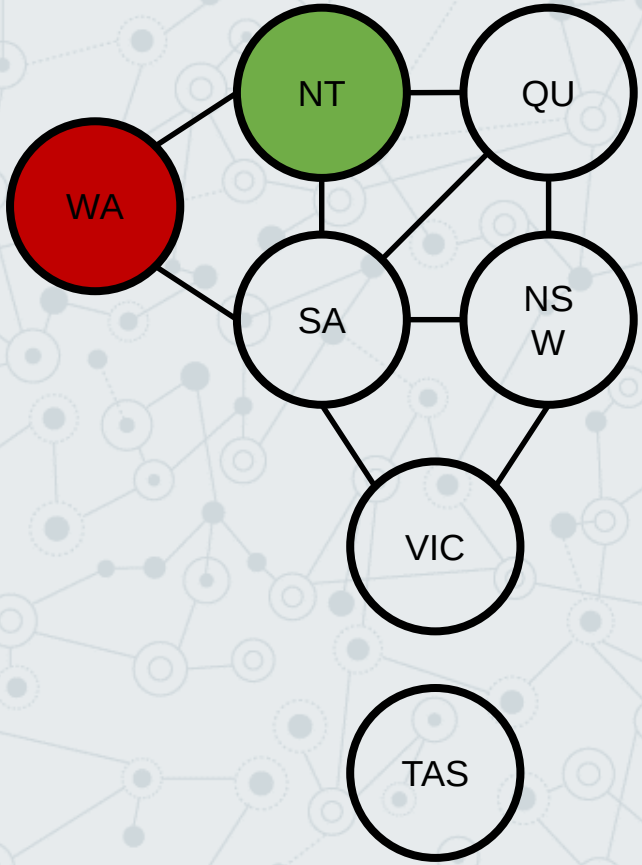




Least Constraining Value Example – Map Coloring

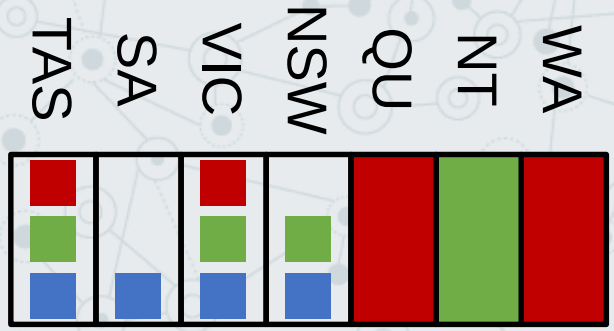
Given a variable, choose the **least constraining value**: the one that rules out the fewest values in the remaining variables

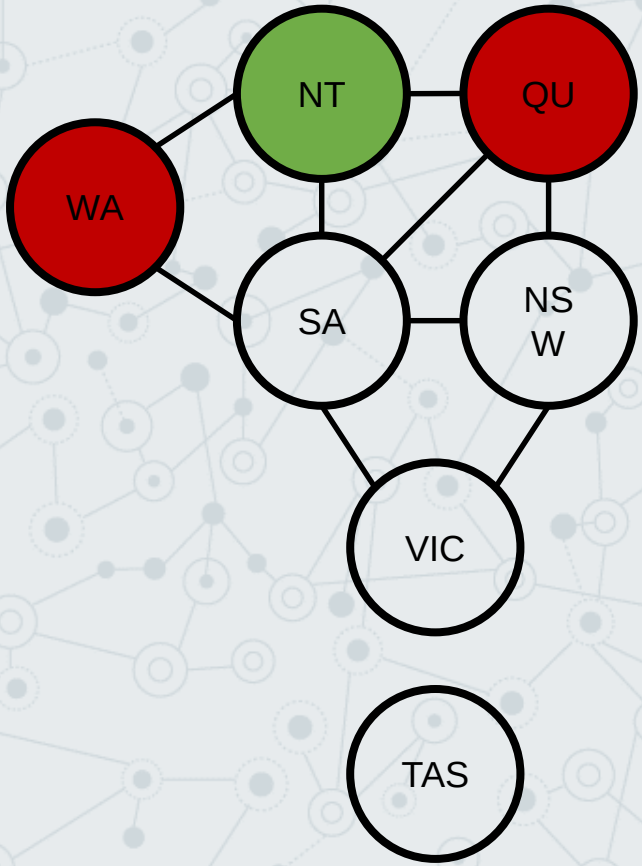




Least Constraining Value Example – Map Coloring

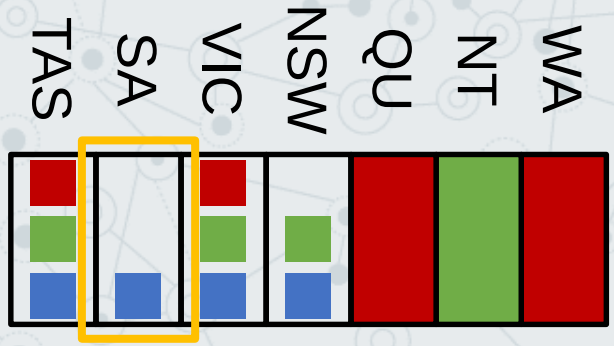
Given a variable, choose the **least constraining value**: the one that rules out the fewest values in the remaining variables

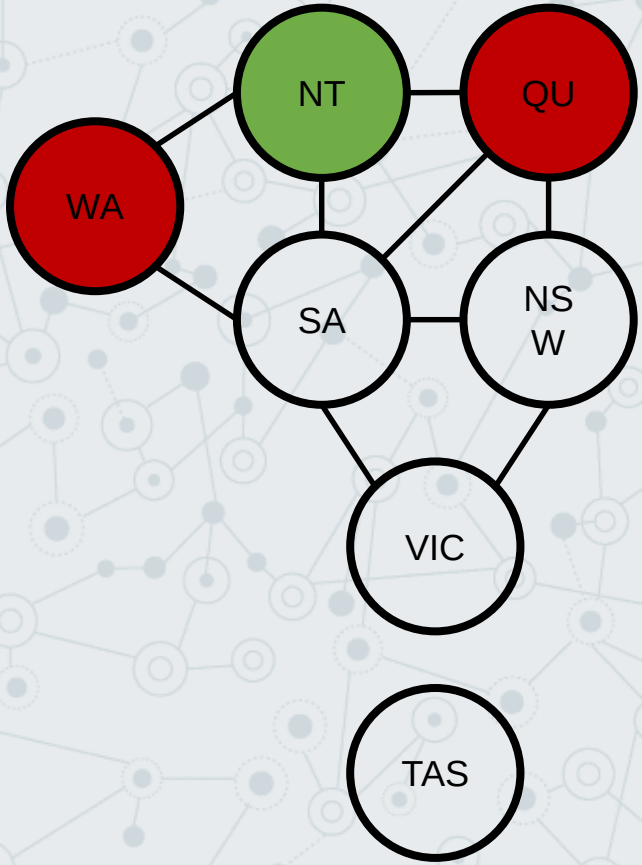




Least Constraining Value Example – Map Coloring

Given a variable, choose the **least constraining value**: the one that rules out the fewest values in the remaining variables





Incremental Heuristics Summary

Which variable should be assigned next?

- Minimum Remaining Values (MRV)
- Degree Heuristic

In what order should its values be tried?

- Least Constraining Value

Solving CSP via Stochastic Greedy Local Search

- Start from a randomly chosen complete instantiation
- Move from one complete instantiation to the next
- The search is guided by some cost function (e.g. the number of violated constraints)
- In its most greedy variant, at each step the value of the variable that leads to the greatest reduction of the cost function is changed (min-conflict heuristic)

Solving CSP via Stochastic Greedy Local Search

procedure SLS

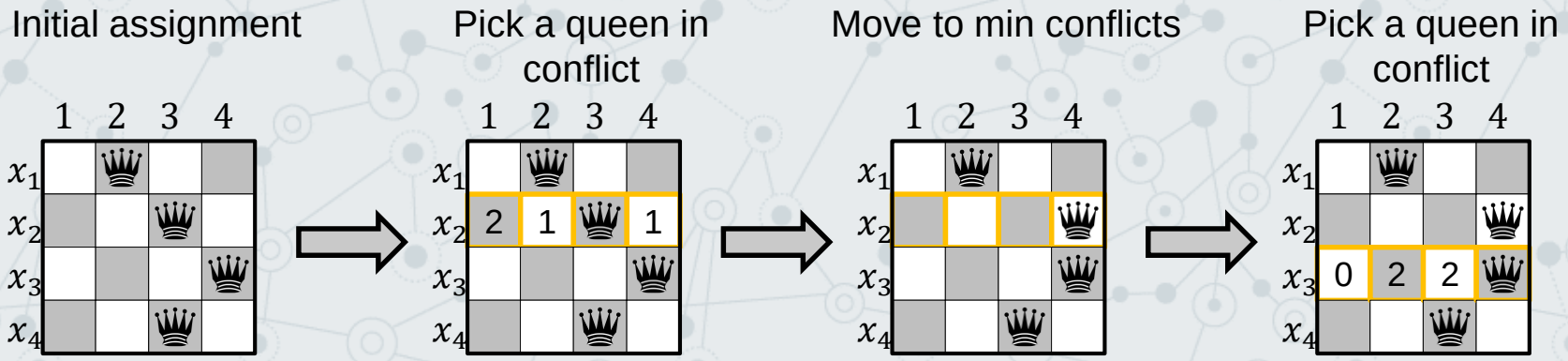
Input: A constraint network $\mathcal{R} = (X, D, C)$, number of tries MAX_TRIES. A cost function defined on full assignments.

Output: A solution iff the problem is consistent, "false" otherwise.

1. **for** i = 1 to MAX_TRIES
   - **initialization: let** $\bar{a} = (a_1, \ldots, a_n)$ be a random initial assignment to all variables.
   - **repeat**
     (a) **if** $\bar{a}$ is consistent, return $\bar{a}$ as a solution.
     (b) **else** let $Y = \{\langle x_i, a'_i \rangle\}$ be the set of variable-value pairs that when x_i is assigned a'_i, give a maximum improvement in the cost of the assignment; pick a pair $\langle x_i, a'_i \rangle \in Y$,
     $\bar{a} \leftarrow (a_1, \ldots, a_{i-1}, a'_i, a_{i+1}, \ldots, a_n)$ (just flip a_i to a'_i).
   - **until** the current assignment cannot be improved.
2. **endfor**
3. **return** false

Min-Conflict Example – 4-Queens SLS

Complete formulation **minimum-conflict** example



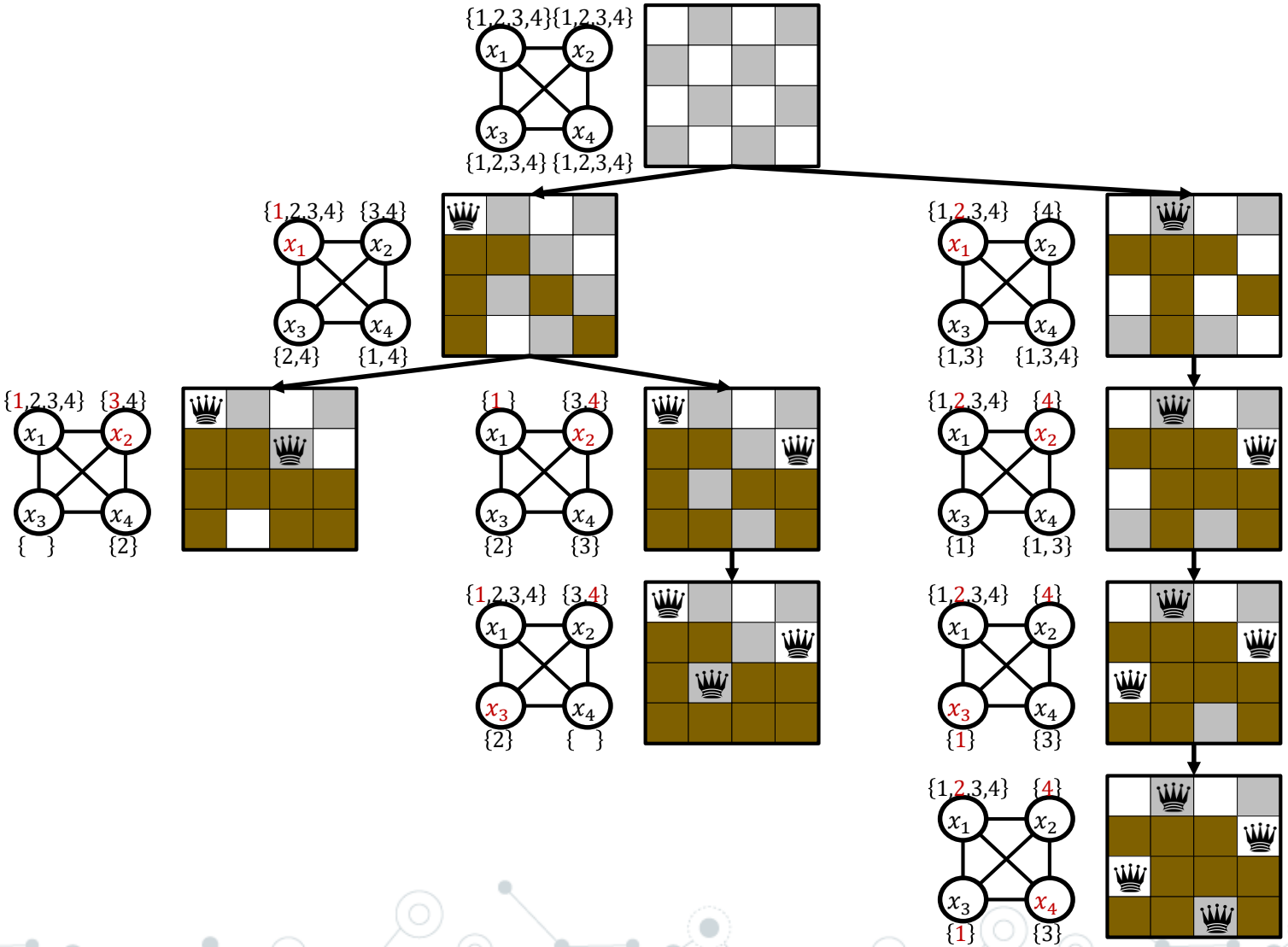
Lookahead

Detection of future (inevitable) search failure:

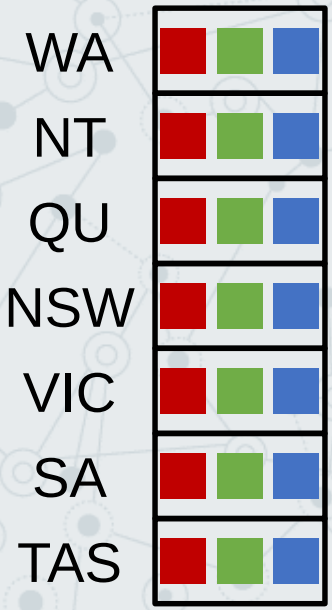
Forward Checking:

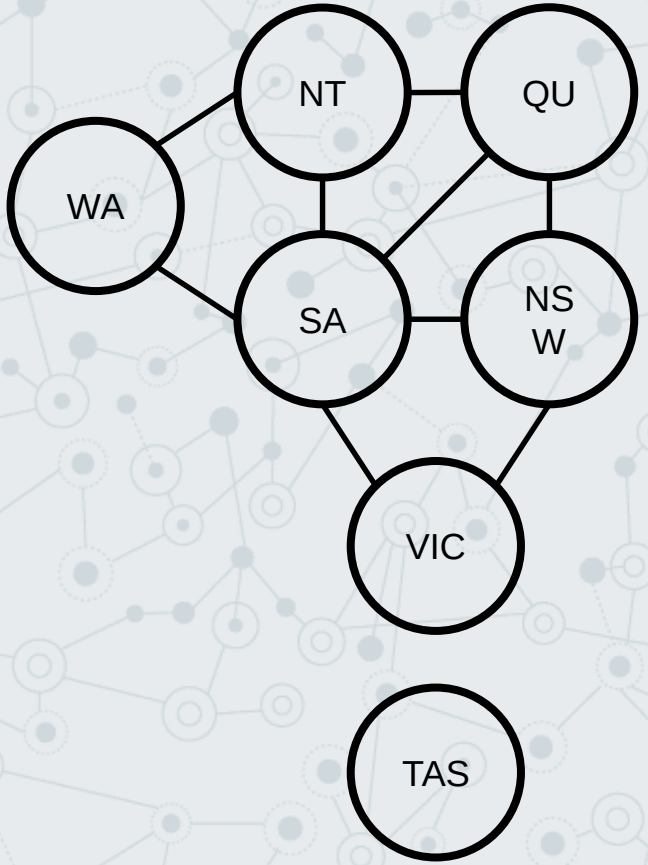
- Whenever a variable x_i is assigned, for every x_j s.t (x_i, x_j) in the CSP Graph, delete from x_j's domain any value that inconsist with the value chosen for x_i
- If x_j's domain becomes empty, abort (illegal assignment)

4 Queens Forward Checking

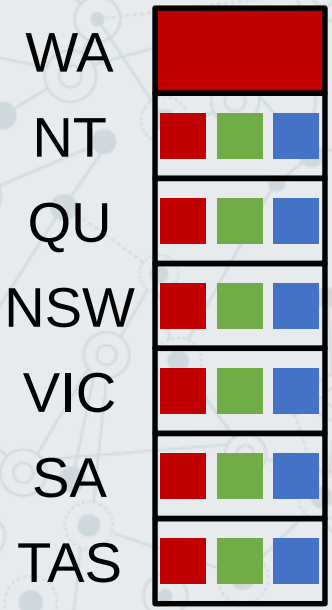


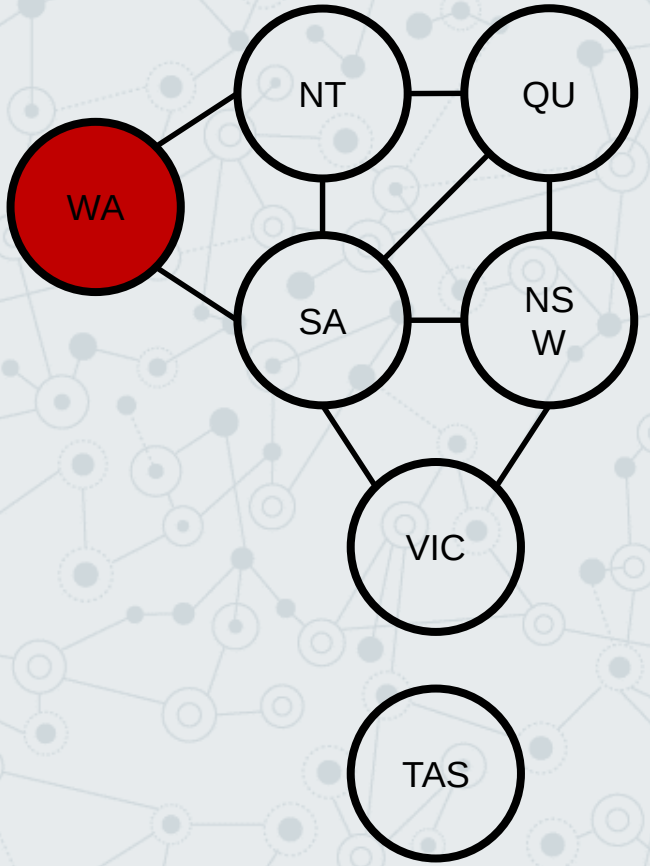
Forward Check – Map Coloring Example



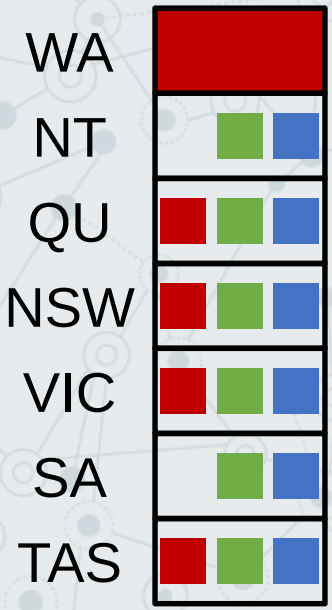


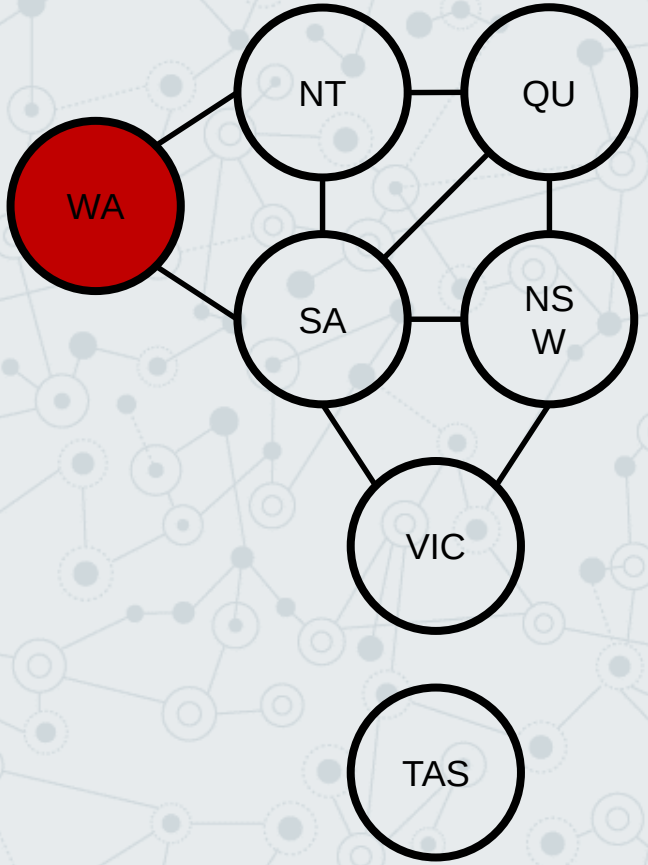
Forward Check – Map Coloring Example



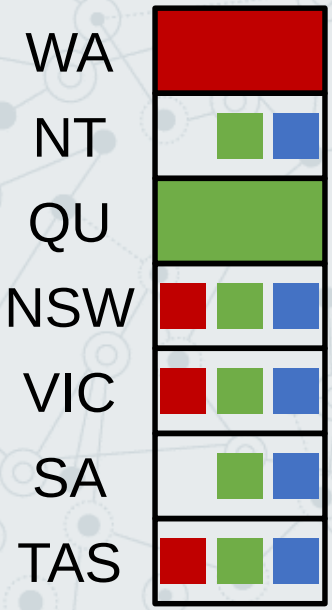


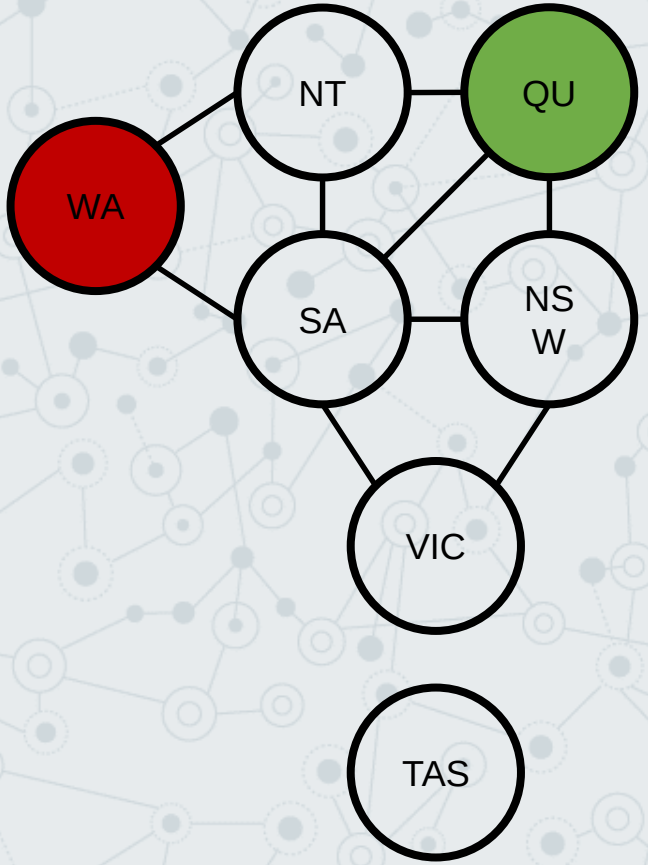
Forward Check – Map Coloring Example



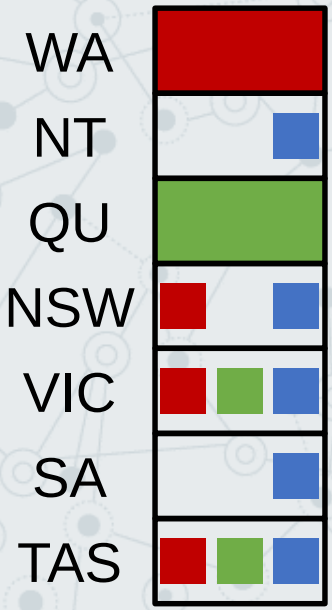


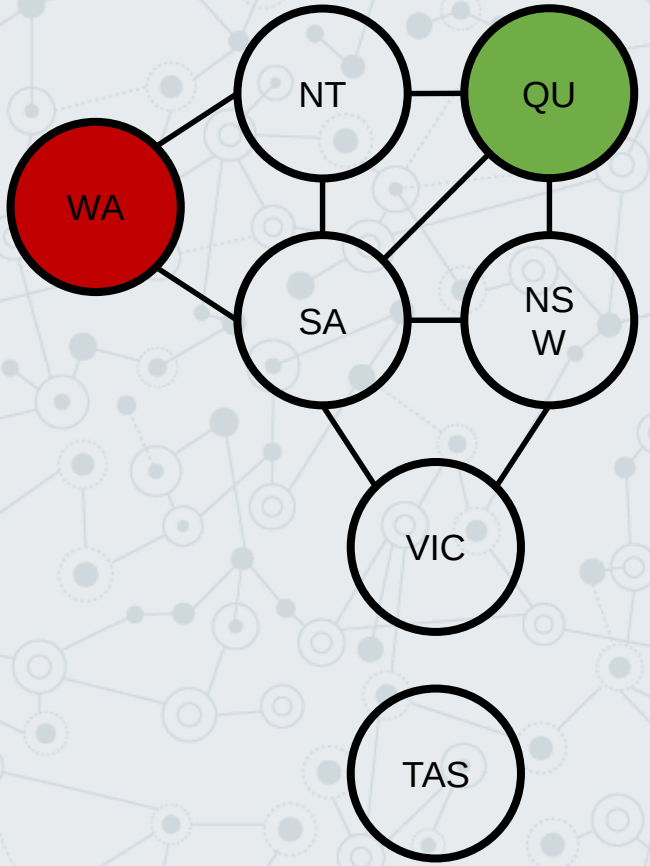
Forward Check – Map Coloring Example



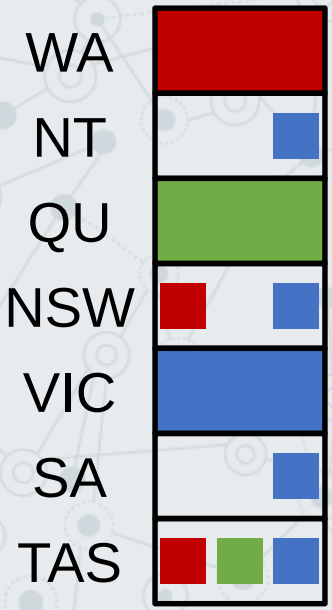


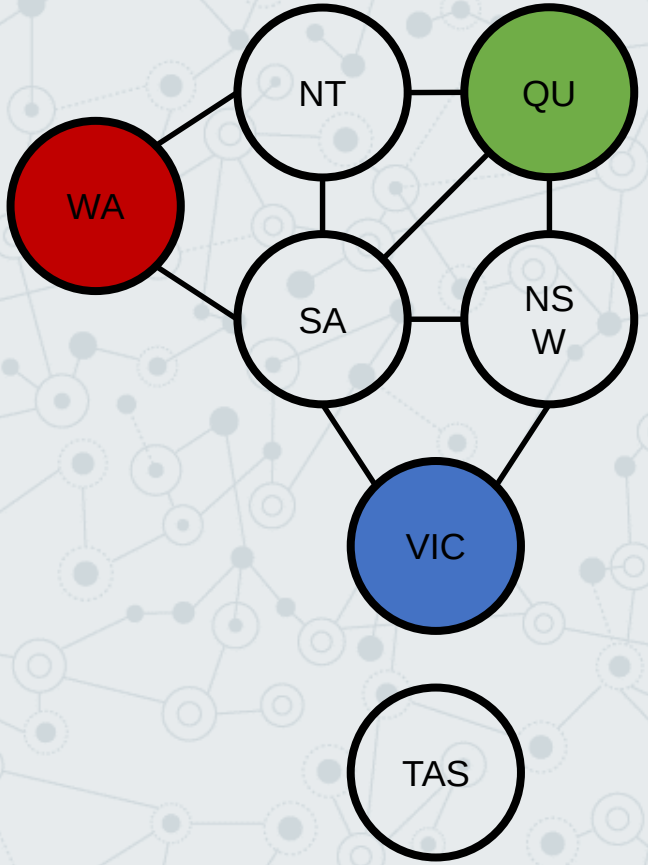
Forward Check – Map Coloring Example



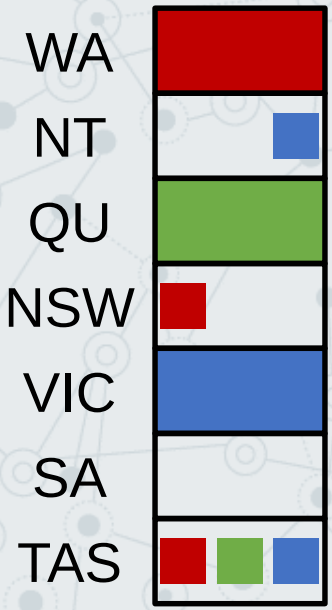


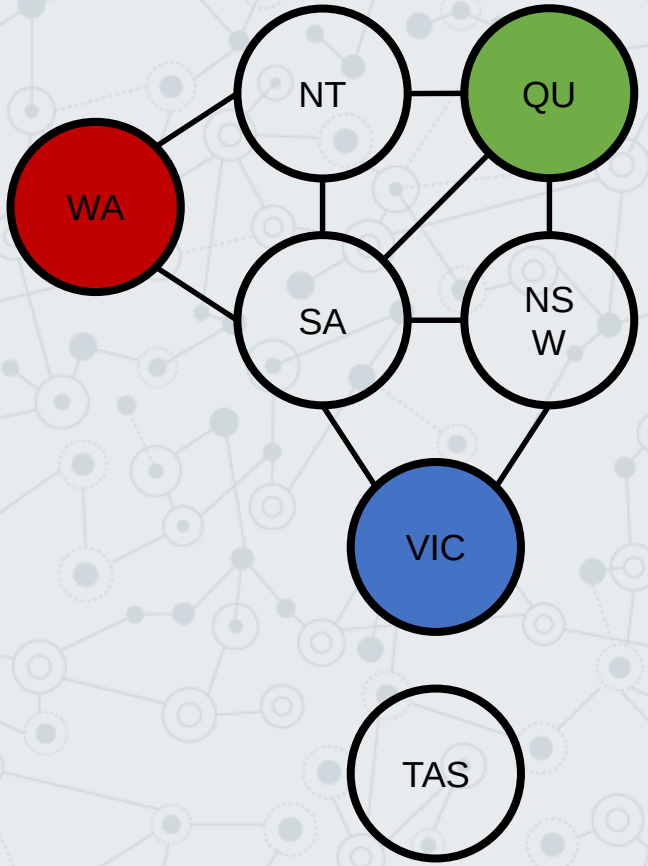
Forward Check – Map Coloring Example



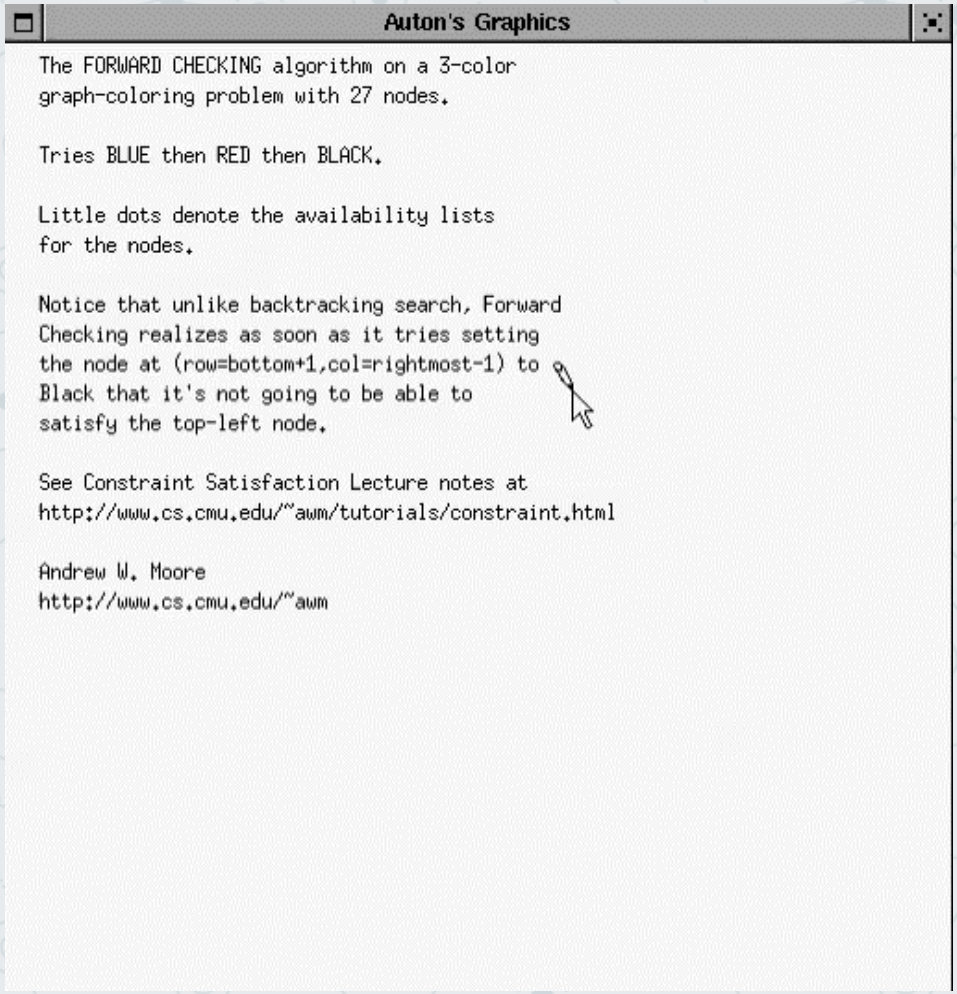


Forward Check – Map Coloring Example

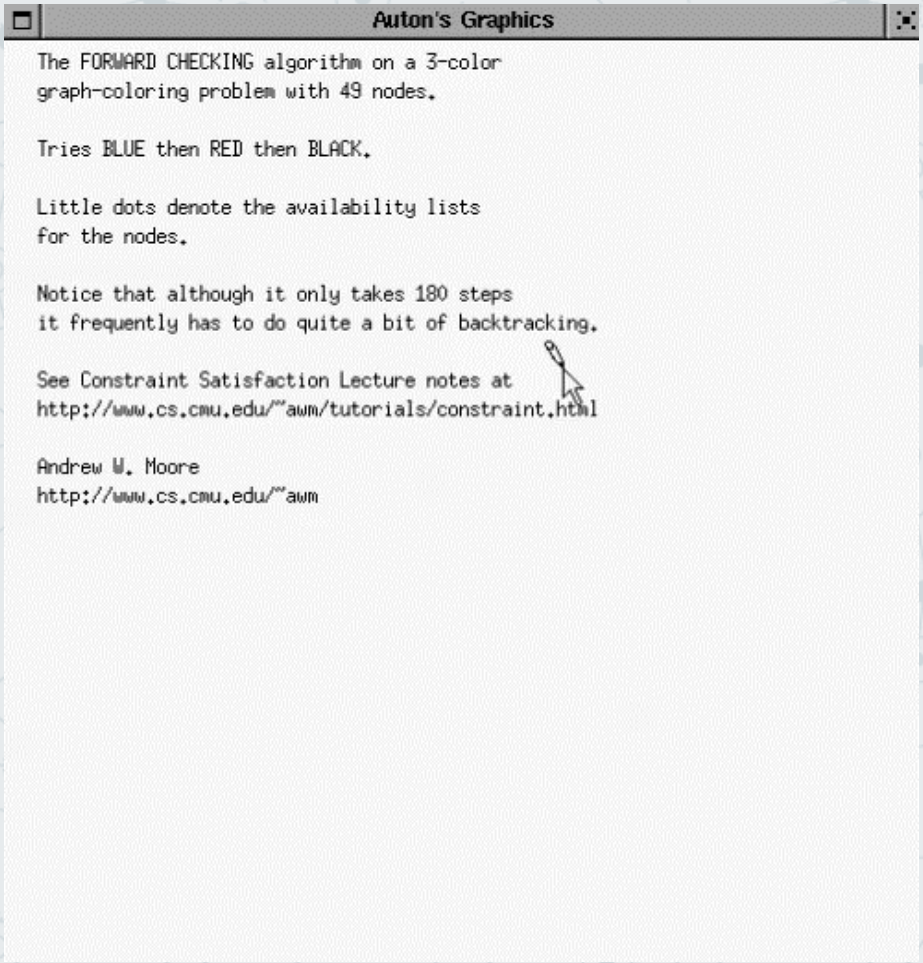




Forward Checking on 27-node graph coloring



Forward Checking on 49-node graph coloring



Look-ahead

Detection of future (inevitable) search failure:

Arc Consistency Propagation:

- For every arc $e_k = x_i, x_j$ of the CSP Graph, check that $\forall d_i \in D_i, \exists d_j \in D_j$ that satisfies this edge C_k
- If check fails, remove d_i from x_i's domain

Arc Consistency Propagation

Arc Consistency works only for Binary CSPs.

However, every general CSP can be reduced to a Binary CSP

Homework – show that for every CSP $\langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$ there is a binary CSP $\langle \widetilde{\mathcal{X}}, \widetilde{\mathcal{D}}, \tilde{\mathcal{C}} \rangle$ such that there is a solution to the original problem if and only if there is a solution to the binary problem.

Arc Consistency Propagation

We can check for arc consistency after every assignment during the search

Note that whenever a value is deleted from some variable's domain to remove an arc inconsistency, a new arc inconsistency could arise in arcs connected to that variable

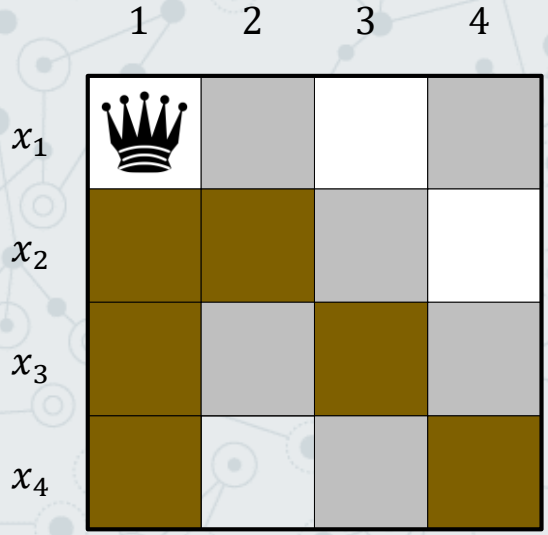
Thus, the process must be applied repeatedly until no more inconsistencies remain

One option for tracking the arcs that need to be checked for inconsistency is using a queue

- Each arc (v_i, v_j) is removed from the queue and checked
- Assume during check inconsistency found, and some value should be deleted from the domain X_i in order to resolve it
- So every arc (v_k, v_i) should be en-queue back for checking

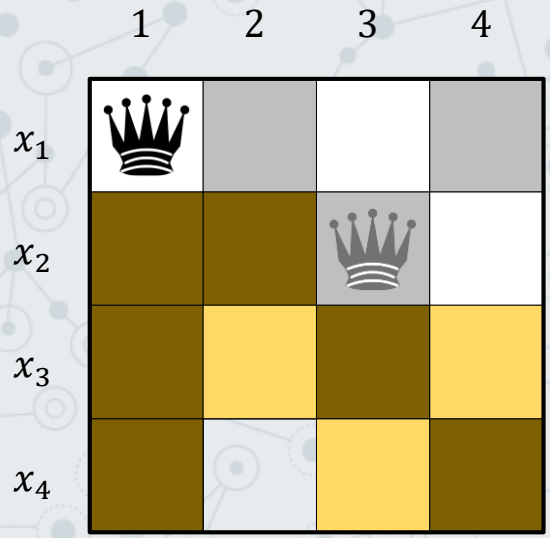
Example – 4-Queens Arc Consistency Propagation

Starting right after forward checking



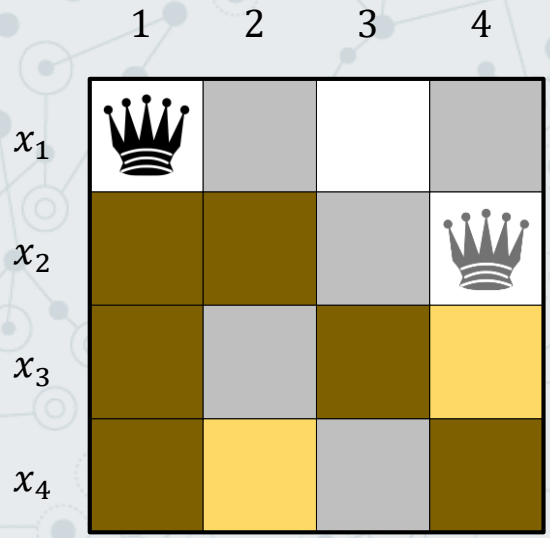
Example – 4-Queens Arc Consistency Propagation

Check x_3, x_4 domains for $x_2 = 3$



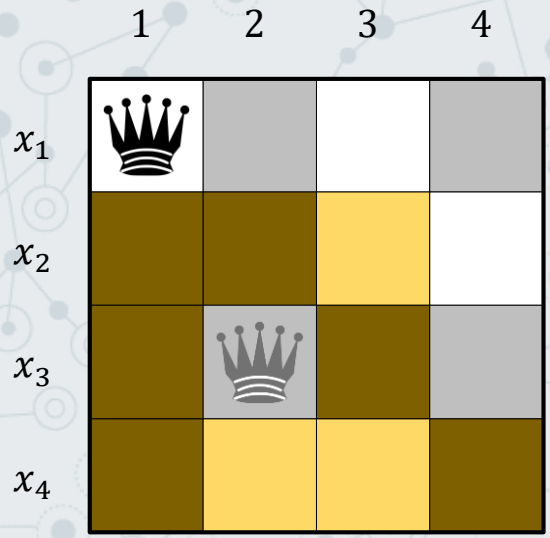
Example – 4-Queens Arc Consistency Propagation

Check x_3, x_4 domains for $x_2 = 4$



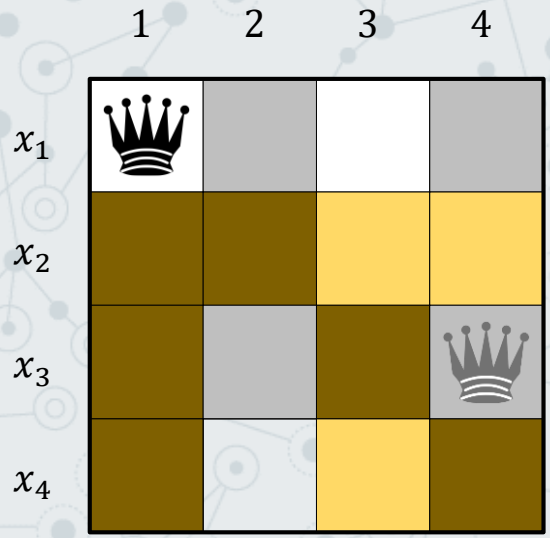
Example – 4-Queens Arc Consistency Propagation

Check x_2,x_4 domains for $x_3 = 2$



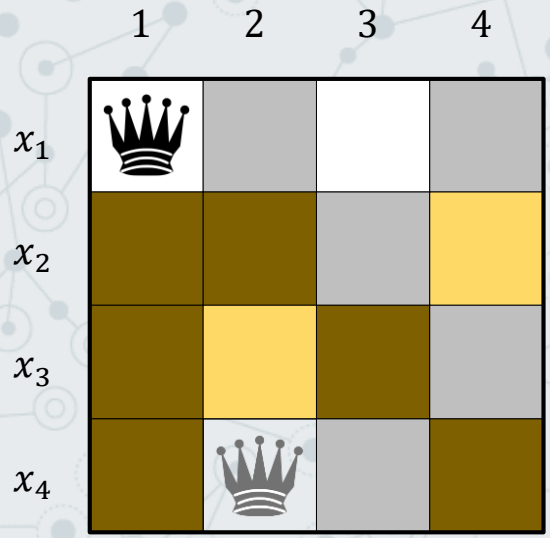
Example – 4-Queens Arc Consistency Propagation

Check x_2,x_4 domains for $x_3 = 4$



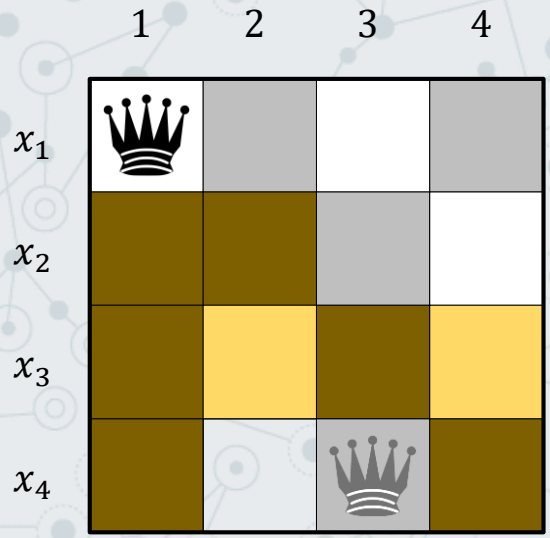
Example – 4-Queens Arc Consistency Propagation

Check x_2,x_3 domains for $x_4 = 2$



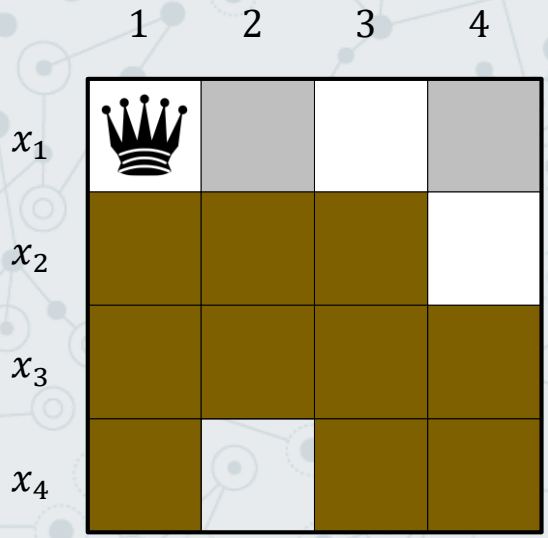
Example – 4-Queens Arc Consistency Propagation

Check x_2,x_3 domains for $x_4 = 3$



Example – 4-Queens Arc Consistency Propagation

So we conclude



Arc Consistency - Revise

REVISE simply tests each value of x_i and eliminates those values having no match in x_j

Function: REVISE

Input: A CSP problem $\langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$ two variables $x_i, x_j \in \mathcal{X}$

Output: D_i such that x_i is arc consistent relative to x_j

For all $d_i \in D_i$:

If there is no $d_j \in D_j$ such that (d_i, d_j) satisfies the constrain between x_i and x_j

delete d_i from D_i

Since each value in D_i is compared, in the worst case, with each value in D_j, the complexity of REVISE is $O(d^2)$ where d bounds the domain size

Arc Consistency

Function: AC-3

Input: A CSP problem $\langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$

Output: D' such that D' is the largest arc consistent domain equivalent to D

$queue \leftarrow \emptyset$

For every pair (x_i, x_j) that participates in a constrain

$queue \leftarrow queue \cup \{(x_i, x_j), (x_j, x_i)\}$

While $queue \neq \emptyset$

$(x_i, x_j) \leftarrow remove - first(queue)$

REVISE((x_i, x_j))

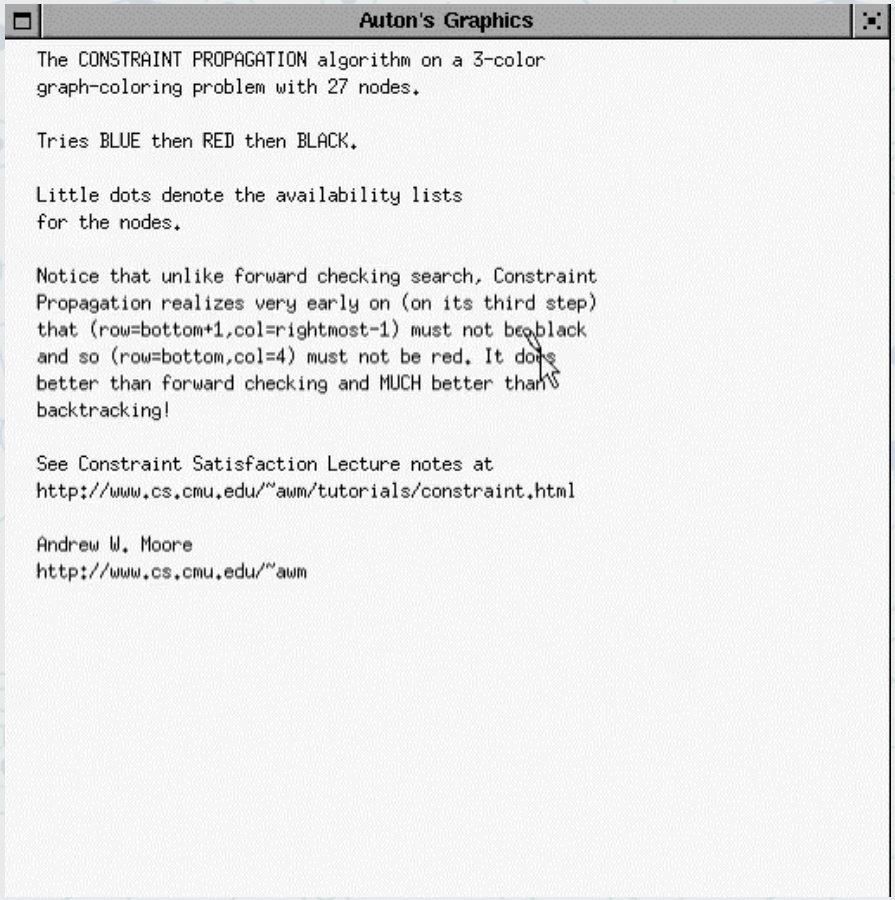
If REVISE((x_i, x_j) caused a change in D_i

$queue \leftarrow queue \cup \{(x_k, x_i): k \neq i, k \neq j\}$

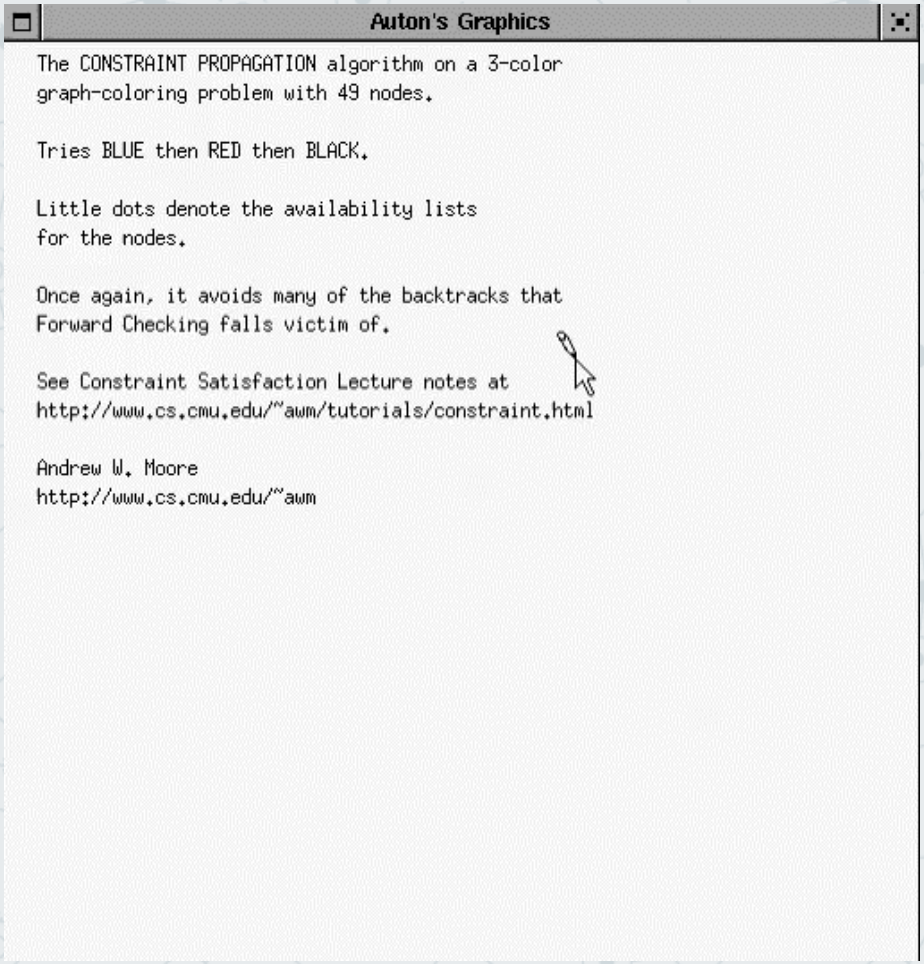
Arc Consistency Complexity (binary constraints)

- Each time a constraint reintroduced into the queue, the domain of a variable in its scope has just been reduced by at least one value
- There are at most $2d$ such values, where d bounds the domain size
- Thus, AC-3 processes each constraint at most $2d$ times
- Assume there are m binary constraints
- Processing each constraint (REVISE) takes $O(d^2)$,
- Conclude the time complexity of AC-3 is $O(m \cdot d^3)$

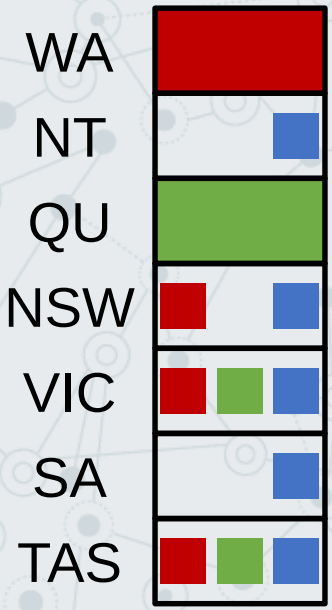
Consistency Propagation on 27-node graph coloring

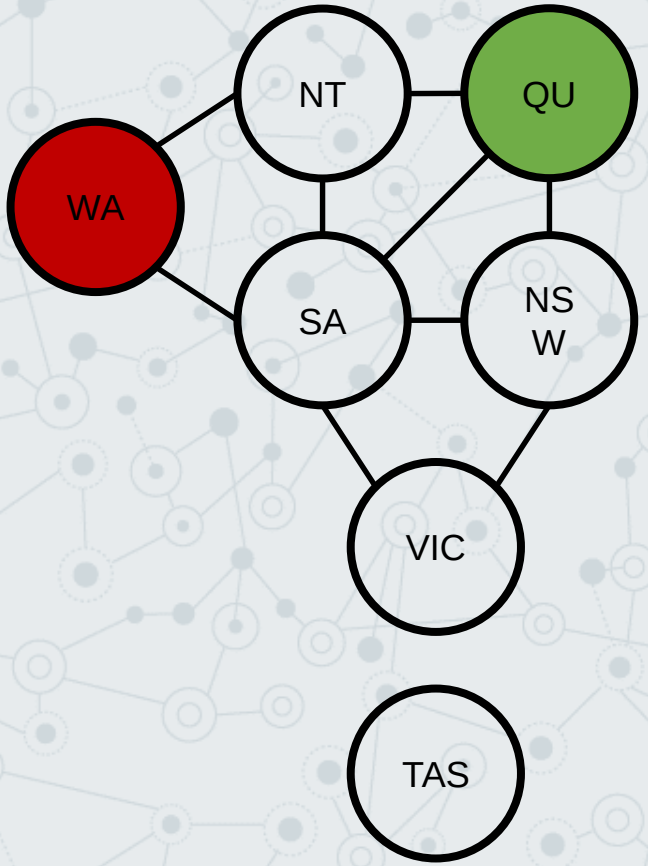


Consistency Propagation on 49-node graph coloring

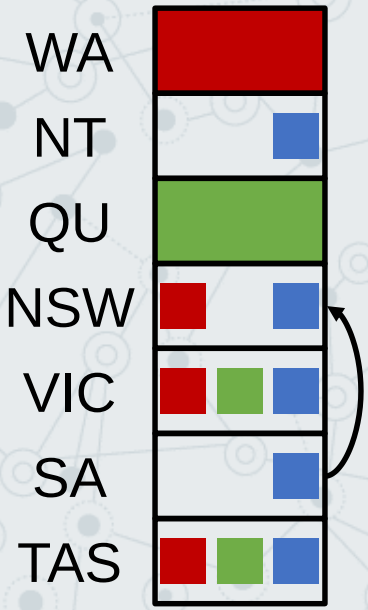


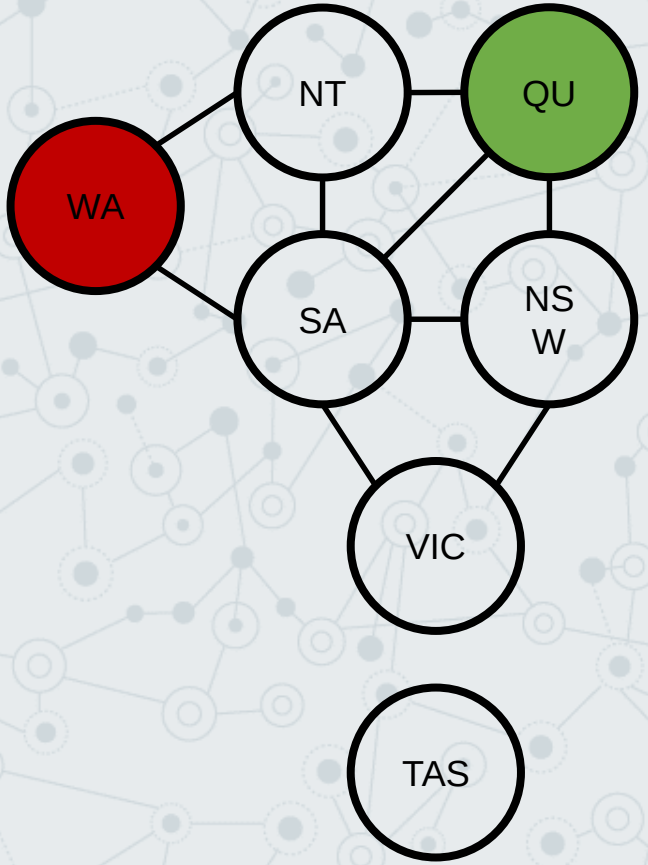
Example – Australia Arc Consistency Propagation



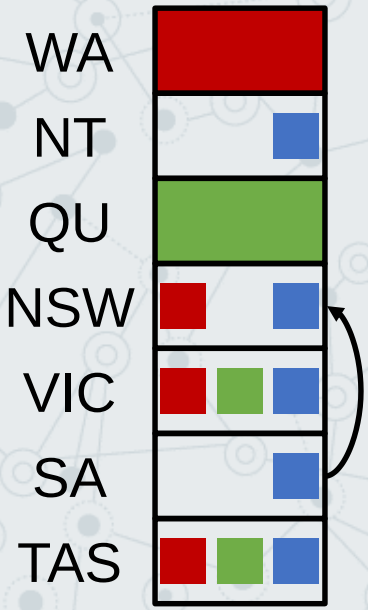


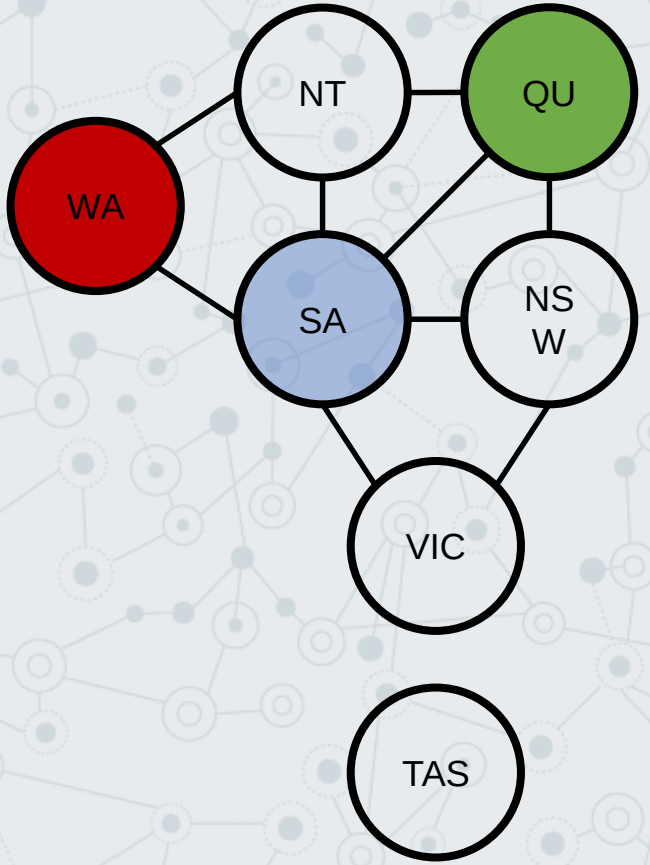
Example – Australia Arc Consistency Propagation



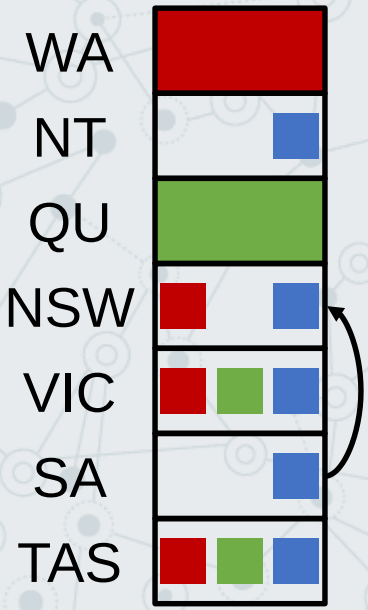


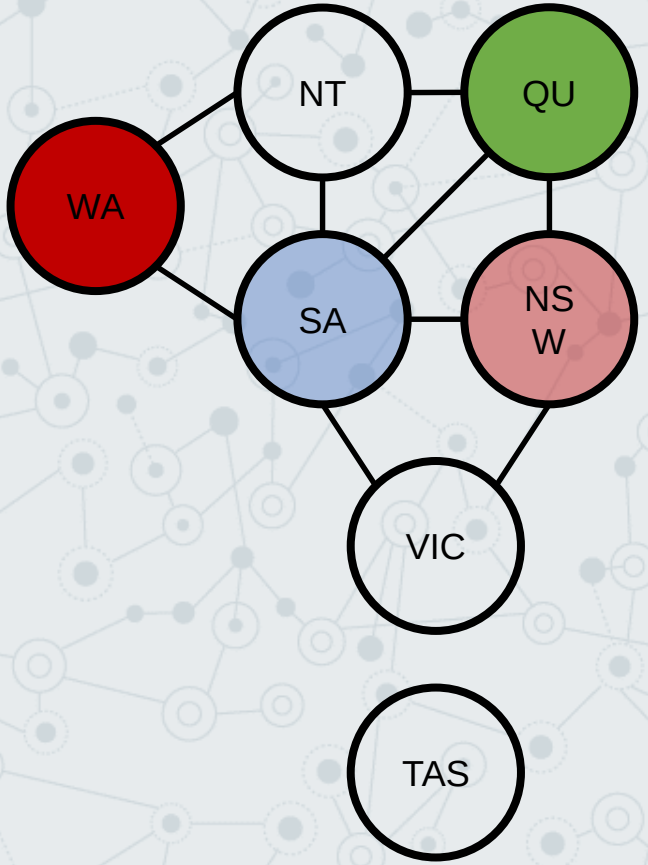
Example – Australia Arc Consistency Propagation



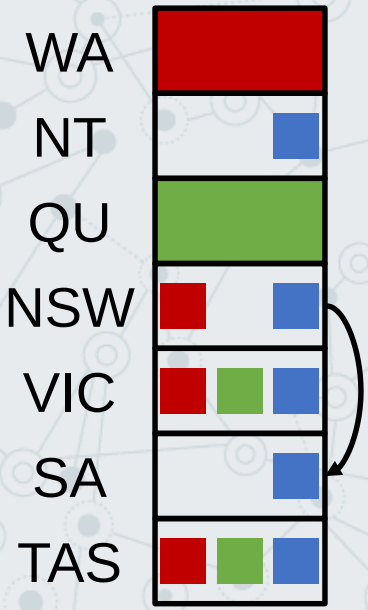


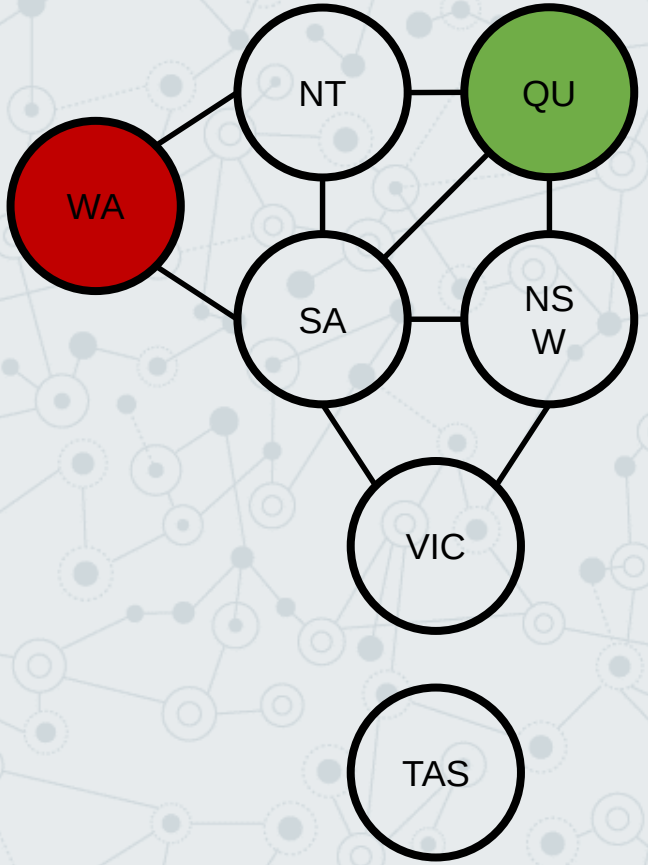
Example – Australia Arc Consistency Propagation



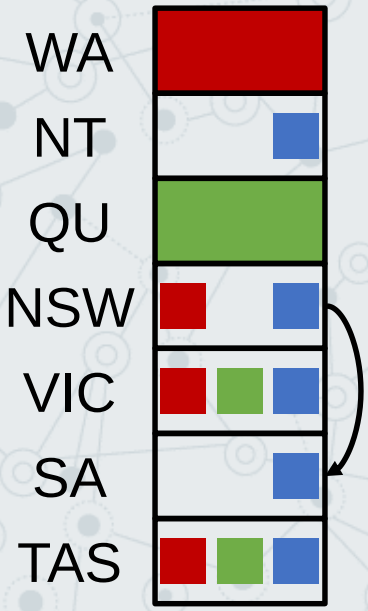


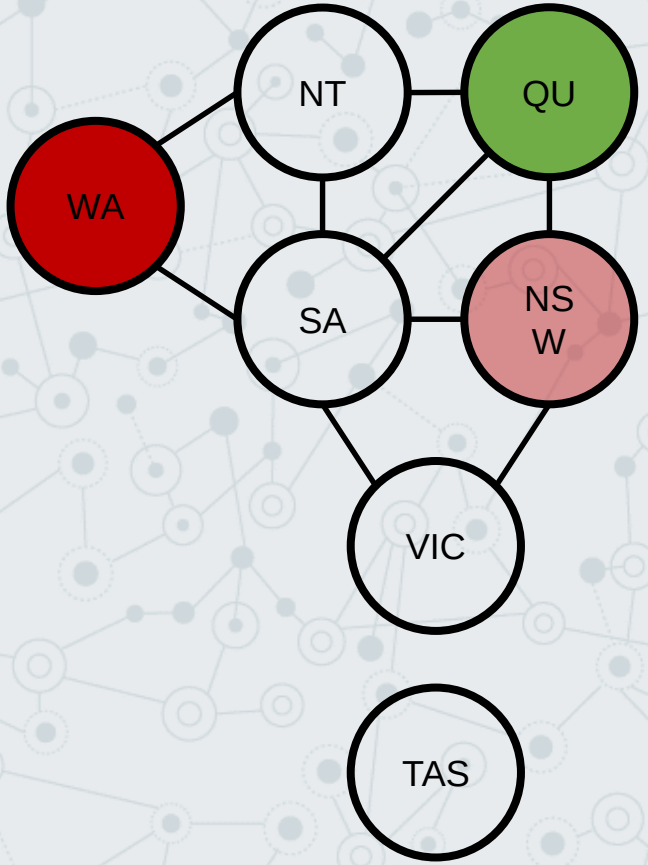
Example – Australia Arc Consistency Propagation



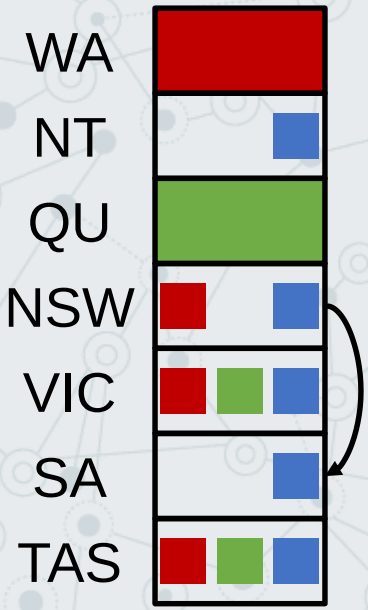


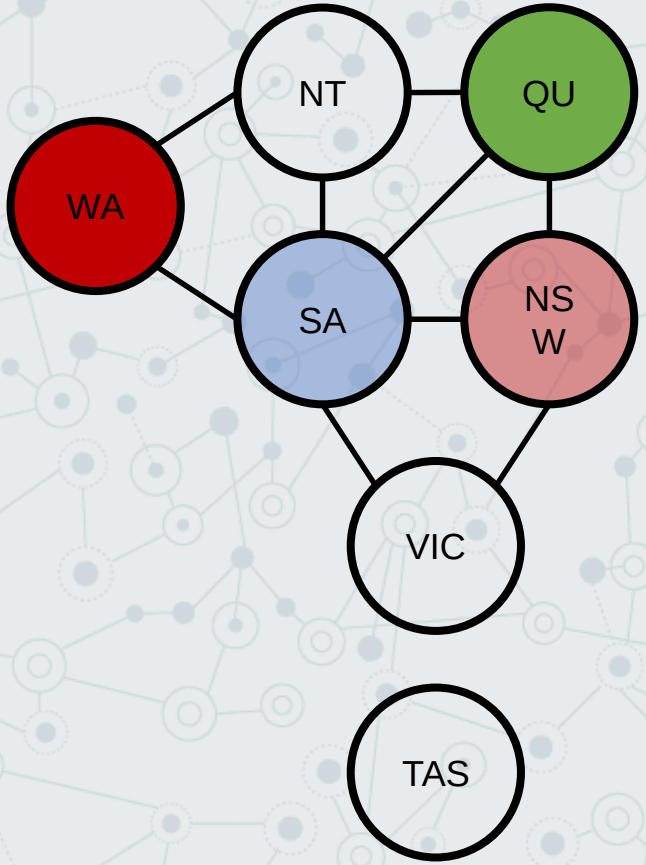
Example – Australia Arc Consistency Propagation



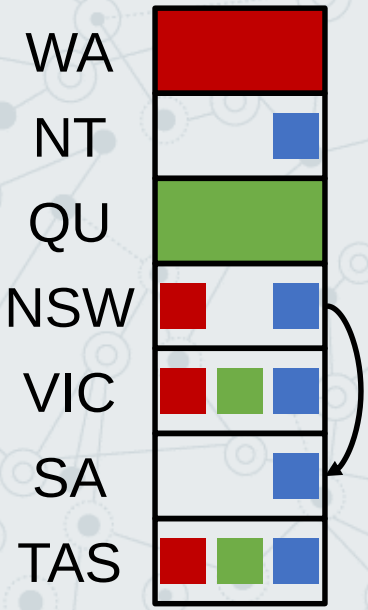


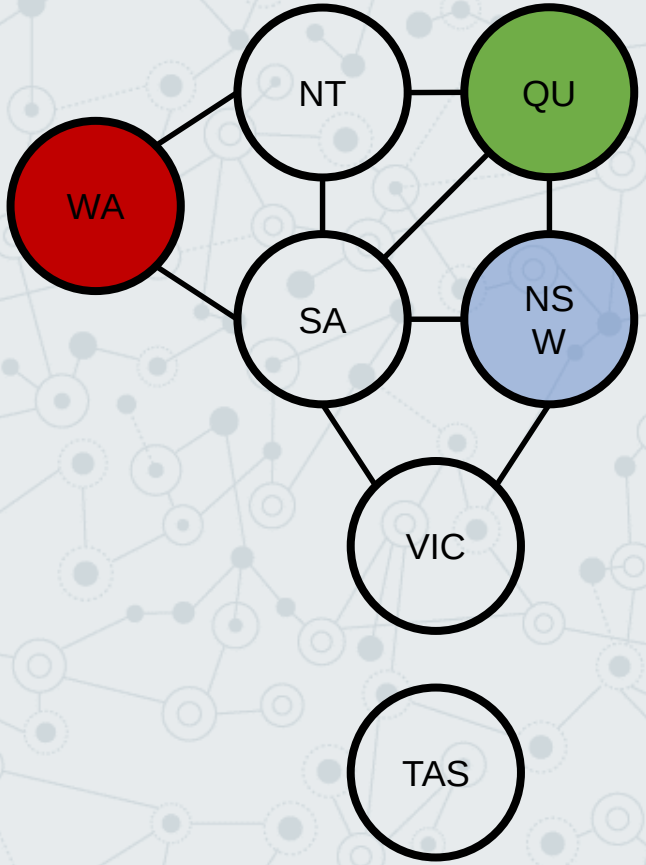
Example – Australia Arc Consistency Propagation



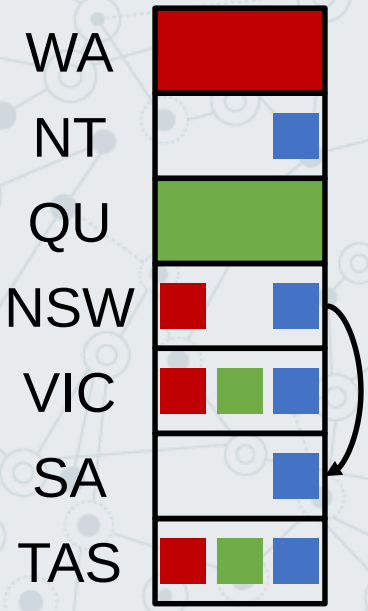


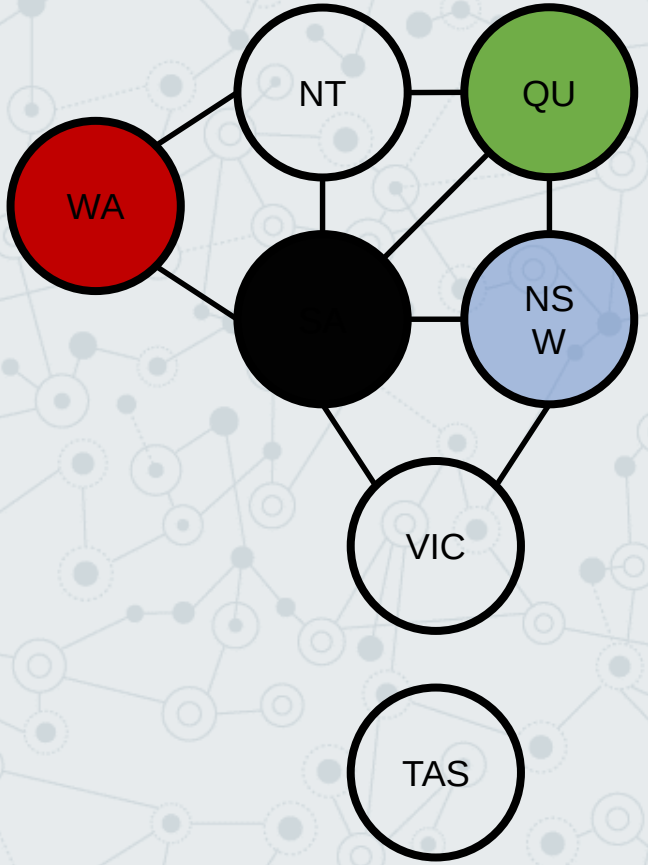
Example – Australia Arc Consistency Propagation



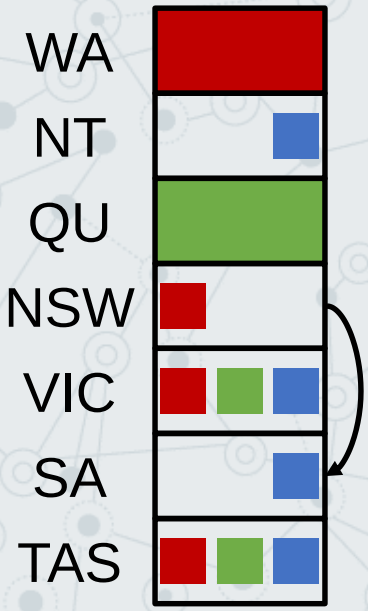


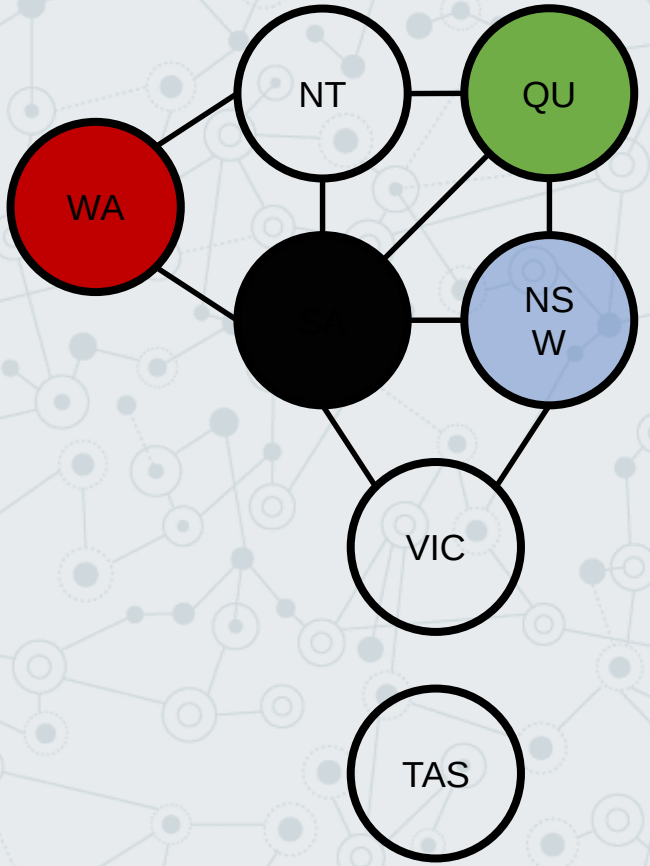
Example – Australia Arc Consistency Propagation



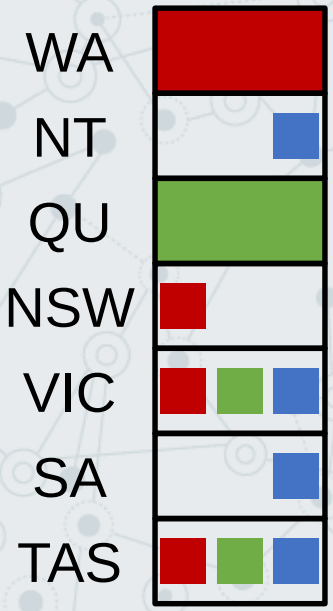


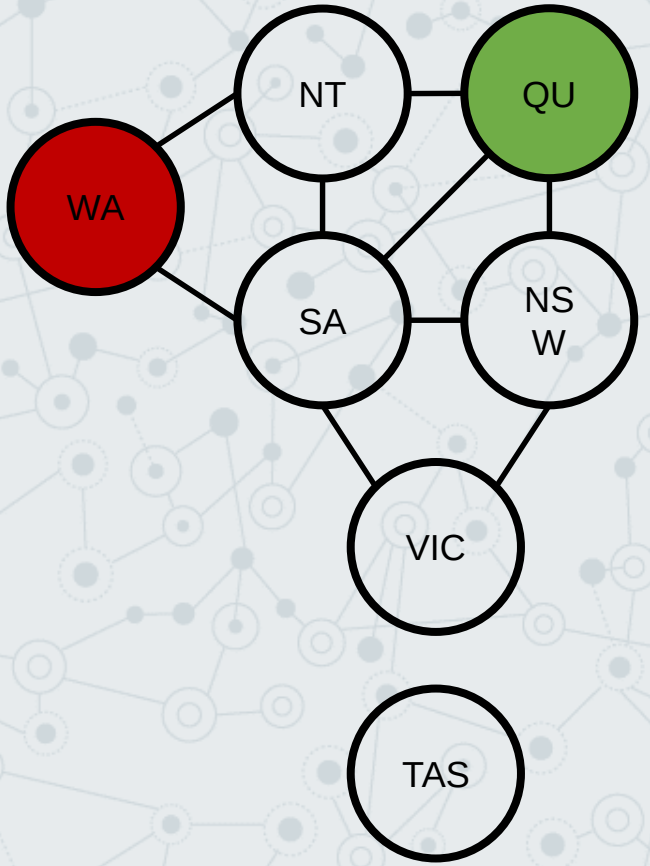
Example – Australia Arc Consistency Propagation



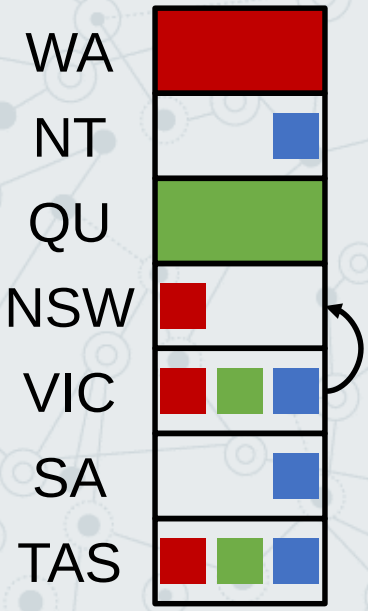


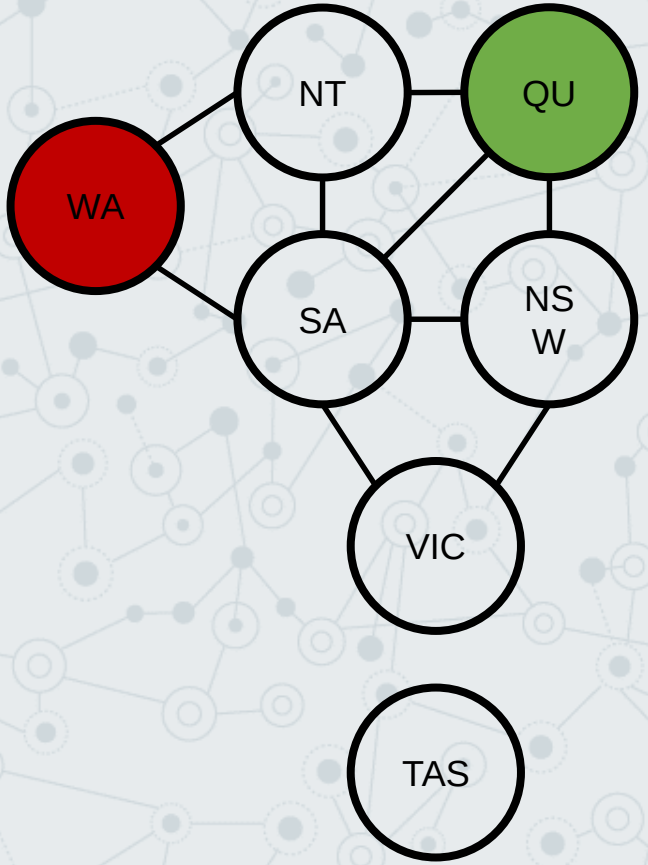
Example – Australia Arc Consistency Propagation



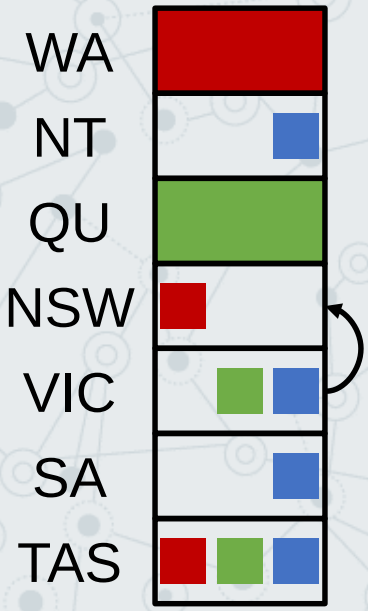


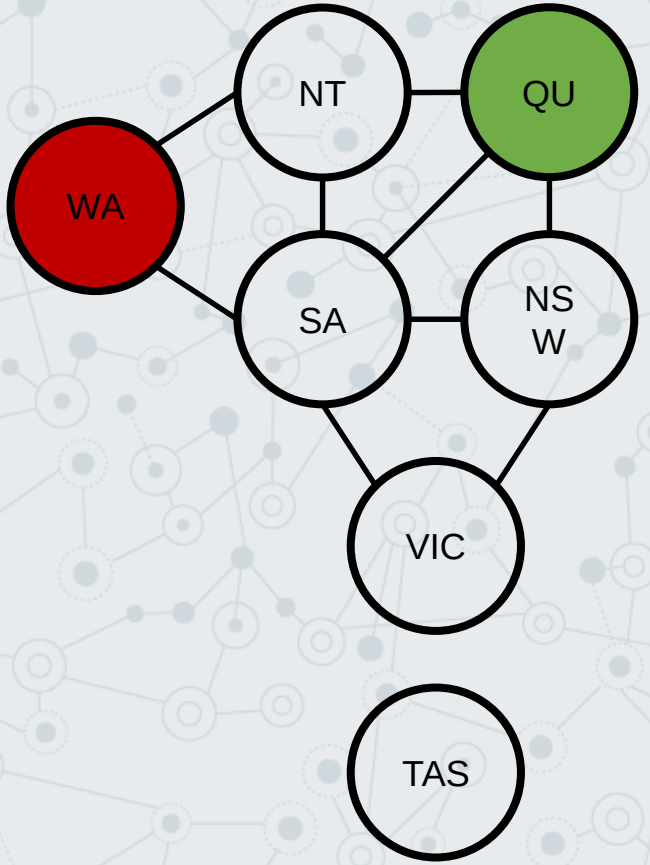
Example – Australia Arc Consistency Propagation



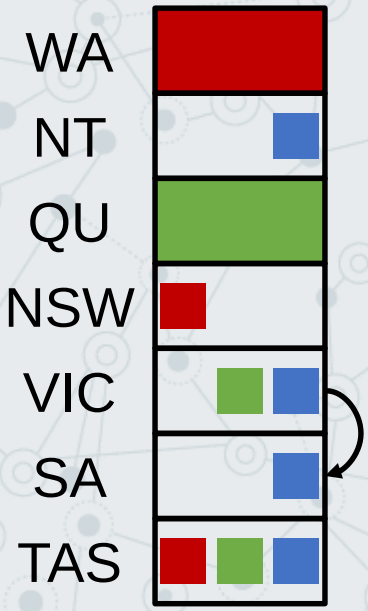


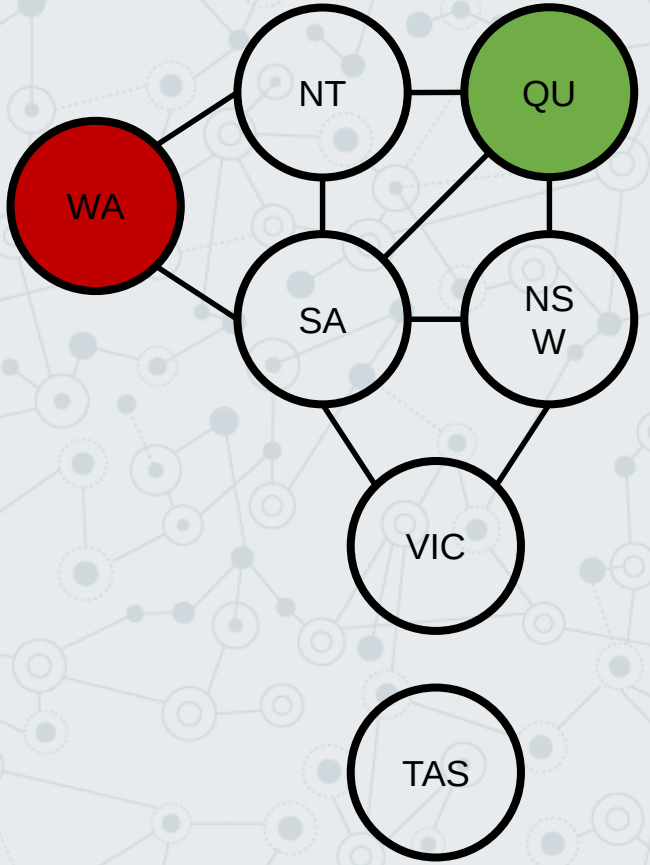
Example – Australia Arc Consistency Propagation



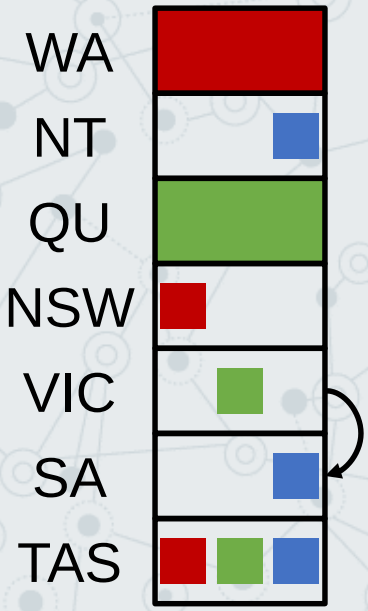


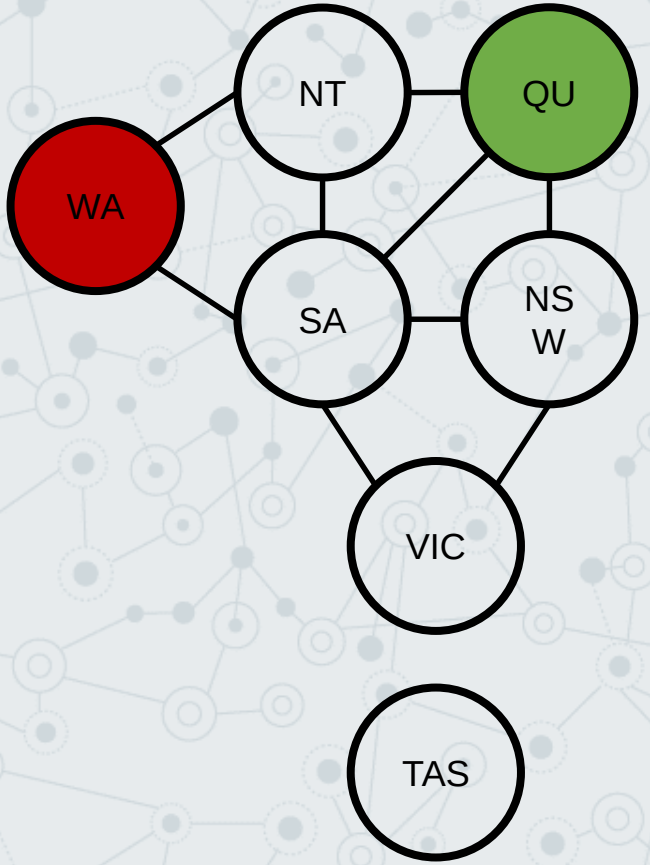
Example – Australia Arc Consistency Propagation





Example – Australia Arc Consistency Propagation





Example – Australia Arc Consistency Propagation