Adversarial Search

Tomáš Svoboda

Vision for Robots and Autonomous Systems, Center for Machine Perception
Department of Cybernetics
Faculty of Electrical Engineering, Czech Technical University in Prague

March 6, 2019

Games, man vs. algorithm

- Deep Blue
- Alpha Go
- Deep Stack
- Why Games, actually?

Games are interesting for AI *because* they are hard (to solve).

Games, man vs. algorithm

- Deep Blue
- Alpha Go
- Deep Stack
- Why Games, actually?

Games are interesting for AI *because* they are hard (to solve).

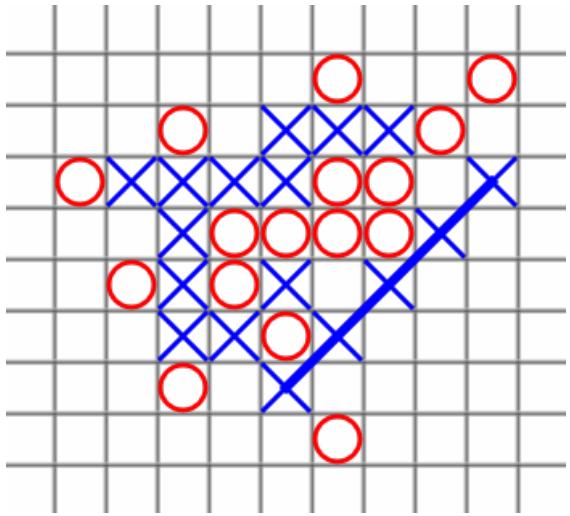
More: Adversarial Learning

Video: Adversing visual segmentation

Elements of the game

- s_0: The initial state
- PLAYER(s). Which player has to move in s.
- ACTIONS(s). What are the legal moves?
- RESULT(s, a). Transition, result of a move.
- TERMINAL-TEST(s). Game over?
- TERMINAL-UTILITY(s, p). What is prize?

Examples for some games ...

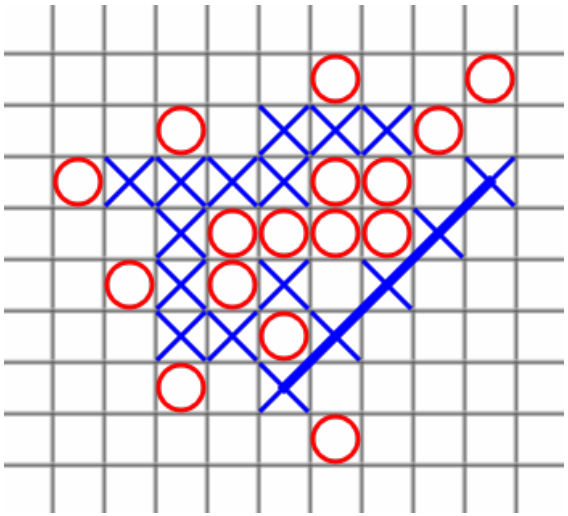


https://commons.wikimedia.org/wiki/File:Tic-tac-toe_5.png

Elements of the game

- s_0: The initial state
- PLAYER(s). Which player has to move in s.
- ACTIONS(s). What are the legal moves?
- RESULT(s, a). Transition, result of a move.
- TERMINAL-TEST(s). Game over?
- TERMINAL-UTILITY(s, p). What is prize?

Examples for some games ...

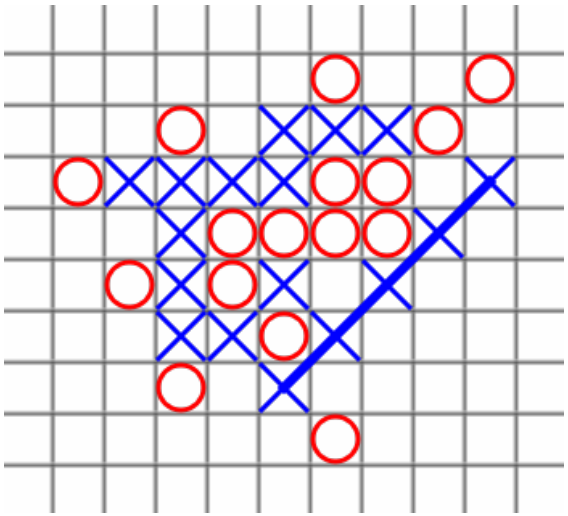


https://commons.wikimedia.org/wiki/File:Tic-tac-toe_5.png

Elements of the game

- s_0: The initial state
- PLAYER(s). Which player has to move in s.
- ACTIONS(s). What are the legal moves?
- RESULT(s, a). Transition, result of a move.
- TERMINAL-TEST(s). Game over?
- TERMINAL-UTILITY(s, p). What is prize?

Examples for some games ...

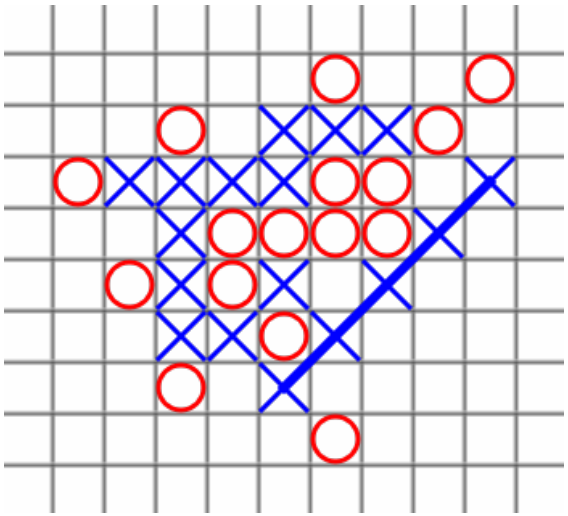


https://commons.wikimedia.org/wiki/File:Tic-tac-toe_5.png

Elements of the game

- s_0: The initial state
- PLAYER(s). Which player has to move in s.
- ACTIONS(s). What are the legal moves?
- RESULT(s, a). Transition, result of a move.
- TERMINAL-TEST(s). Game over?
- TERMINAL-UTILITY(s, p). What is prize?

Examples for some games ...

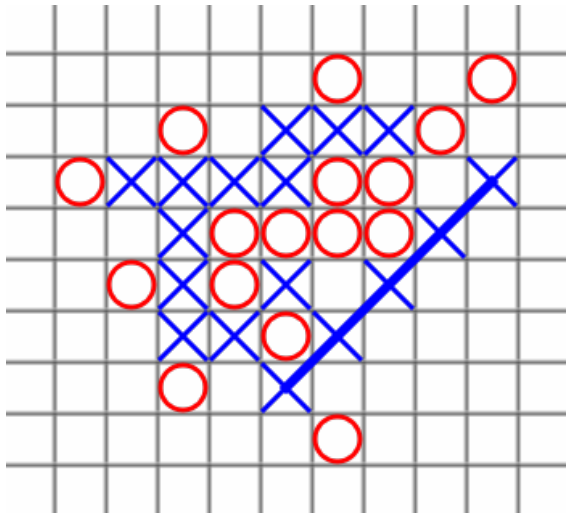


https://commons.wikimedia.org/wiki/File:Tic-tac-toe_5.png

Elements of the game

- s_0: The initial state
- PLAYER(s). Which player has to move in s.
- ACTIONS(s). What are the legal moves?
- RESULT(s, a). Transition, result of a move.
- TERMINAL-TEST(s). Game over?
- TERMINAL-UTILITY(s, p). What is prize?

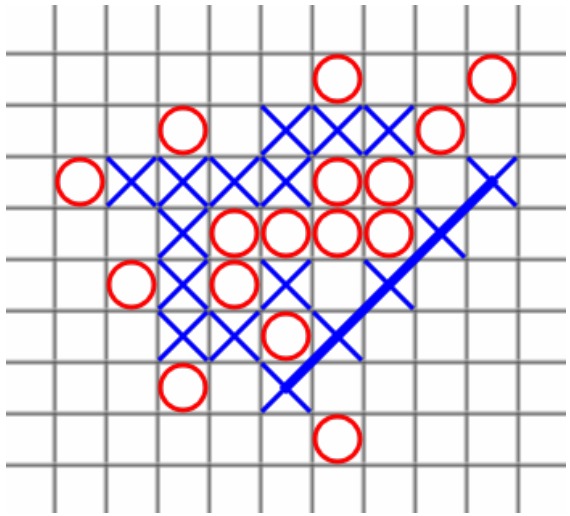
Examples for some games ...



https://commons.wikimedia.org/wiki/File:Tic-tac-toe_5.png

Elements of the game

- s_0: The initial state
- PLAYER(s). Which player has to move in s.
- ACTIONS(s). What are the legal moves?
- RESULT(s, a). Transition, result of a move.
- TERMINAL-TEST(s). Game over?
- TERMINAL-UTILITY(s, p). What is prize?
  Examples for some games ...



https://commons.wikimedia.org/wiki/File:Tic-tac-toe_5.png

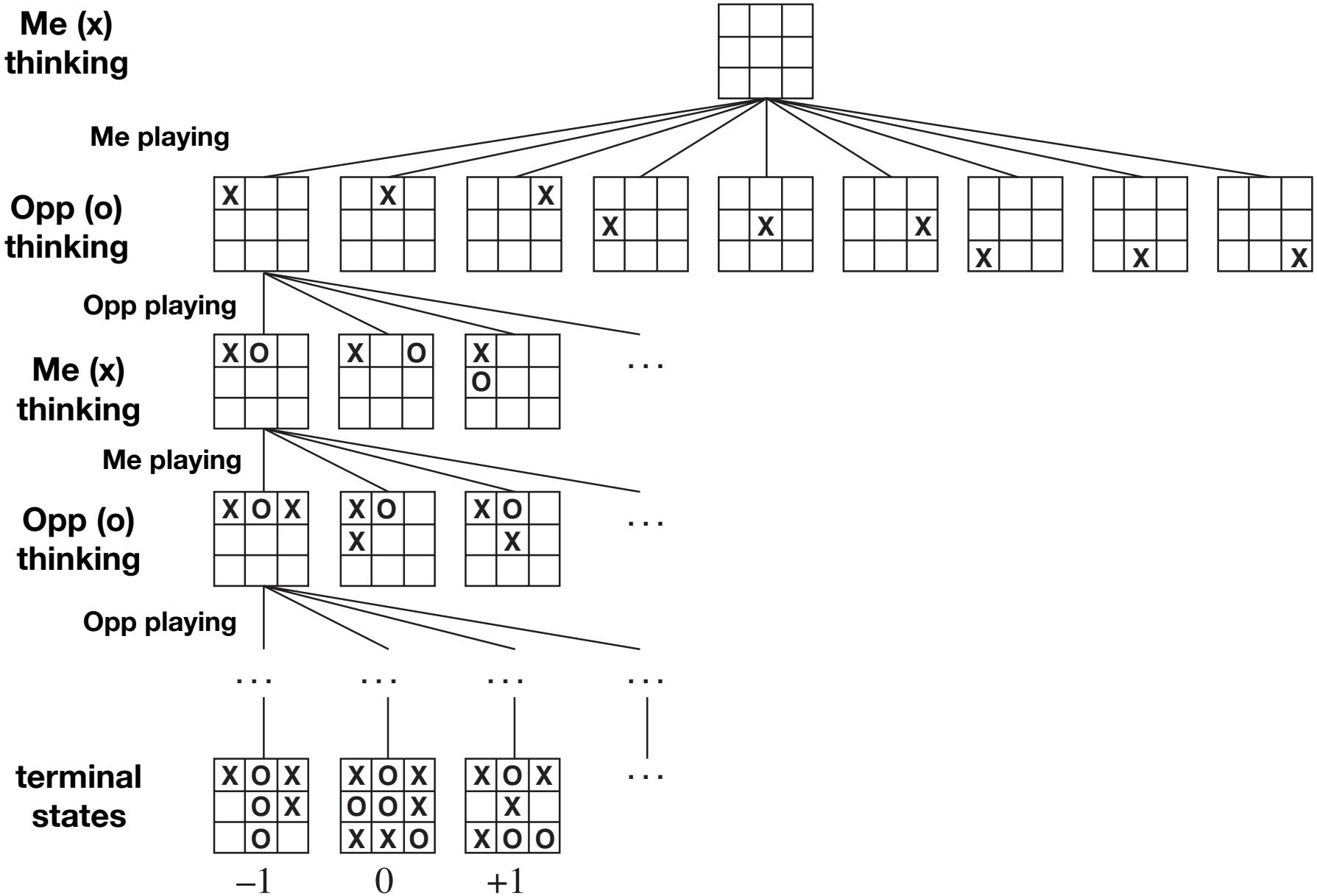
Terminal utilitity: Zero-Sum and General games

- Zero-sum: players have opposite utilities (values)
- Zero-sum: playing against
- General game: independent utilities
- General game: cooperations, competition, ...

Terminal utilitity: Zero-Sum and General games

- Zero-sum: players have opposite utilities (values)
- Zero-sum: playing against
- General game: independent utilities
- General game: cooperations, competition, ...

Game Tree(s)



TERMINAL-UTILITY$(s, \mathbf{x})$

State Value $V(s)$

$V(s)$ – value V of a state s : The best utility achievable from this state.

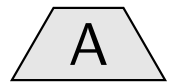
$$V(s) = \max_{s' \in \text{children}(s)} V(s')$$

State Value $V(s)$

$V(s)$ – value V of a state s : The best utility achievable from this state.

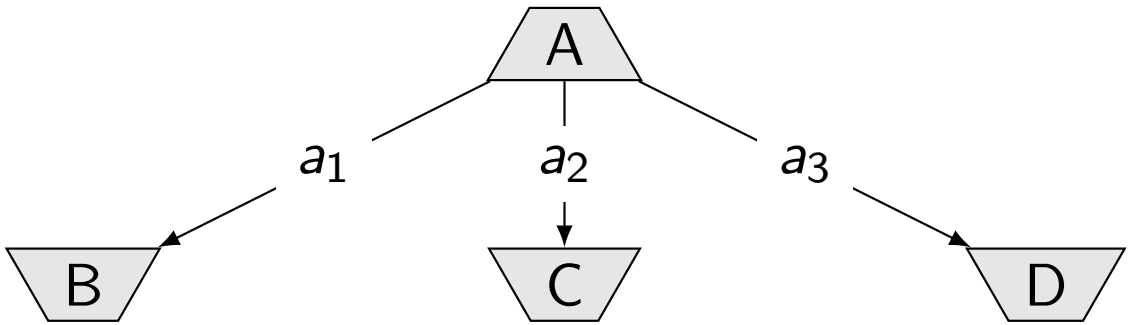
$$V(s) = \max_{s' \in \text{children}(s)} V(s')$$

Two-ply game: **max** for me, **min** for the opponent.



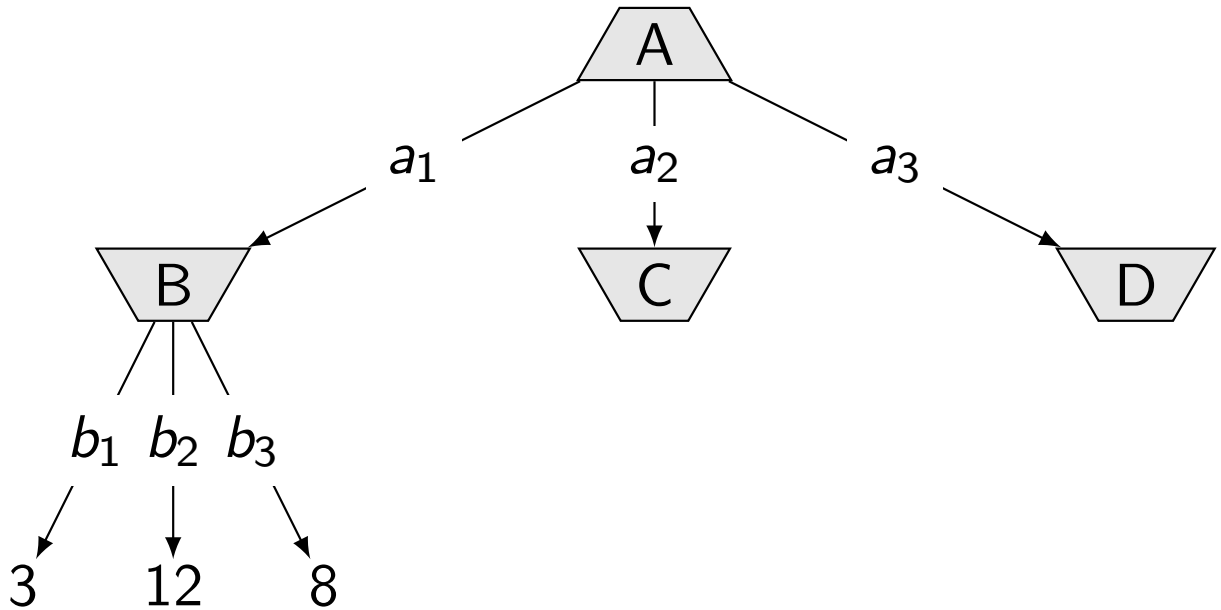
$$a_1 = \underset{a \in \textsc{Actions}(A)}{\arg\max} \; \textsc{Result}(A, a)$$

Two-ply game: **max** for me, **min** for the opponent.



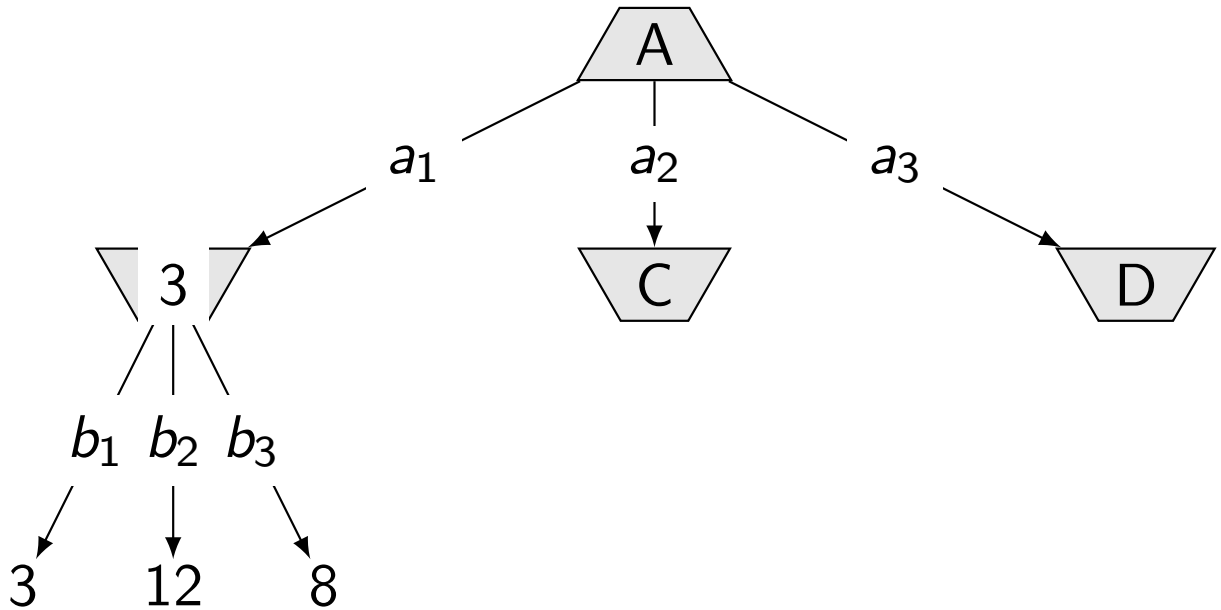
$$a_1 = \arg\max_{a \in \text{ACTIONS}(A)} \text{RESULT}(A, a)$$

Two-ply game: **max** for me, **min** for the opponent.



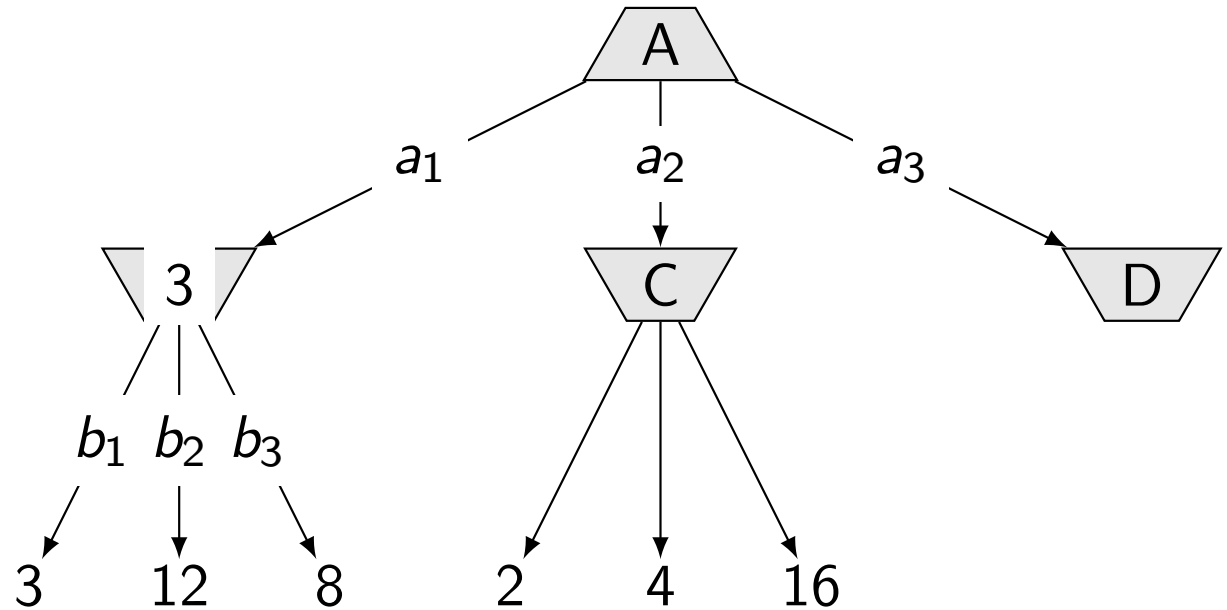
$$a_1 = \arg\max_{a \in \textsc{Actions}(A)} \textsc{Result}(A, a)$$

Two-ply game: **max** for me, **min** for the opponent.



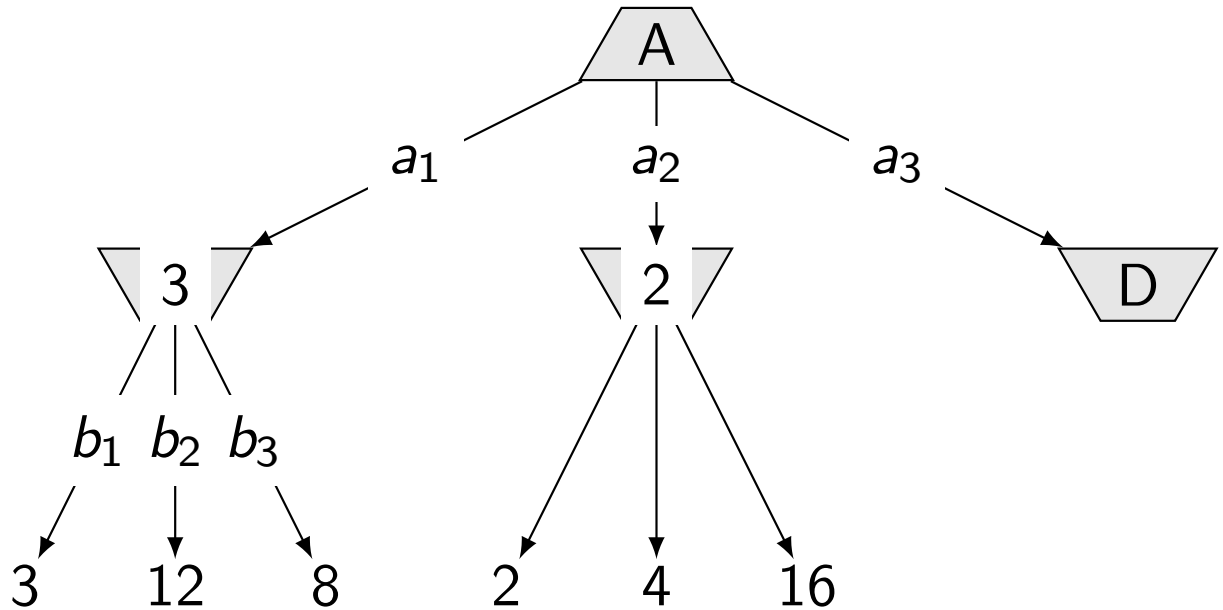
$$a_1 = \arg\max_{a \in \textsc{Actions}(A)} \textsc{Result}(A, a)$$

Two-ply game: **max** for me, **min** for the opponent.



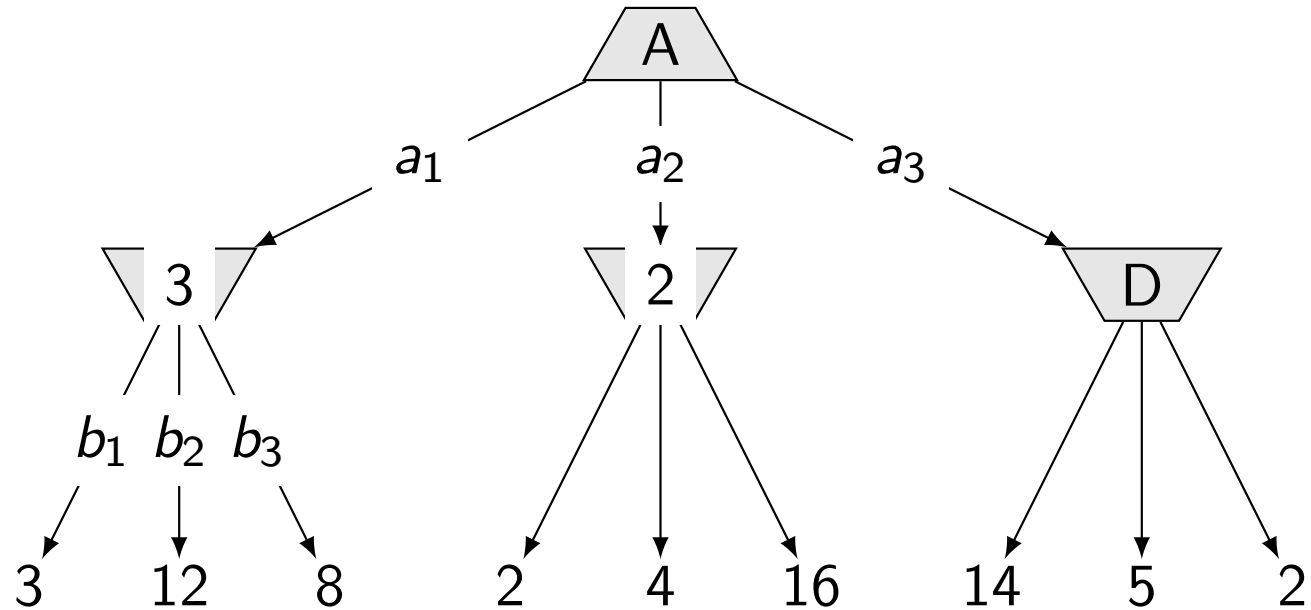
$$a_1 = \arg\max_{a \in \text{ACTIONS}(A)} \text{RESULT}(A, a)$$

Two-ply game: **max** for me, **min** for the opponent.



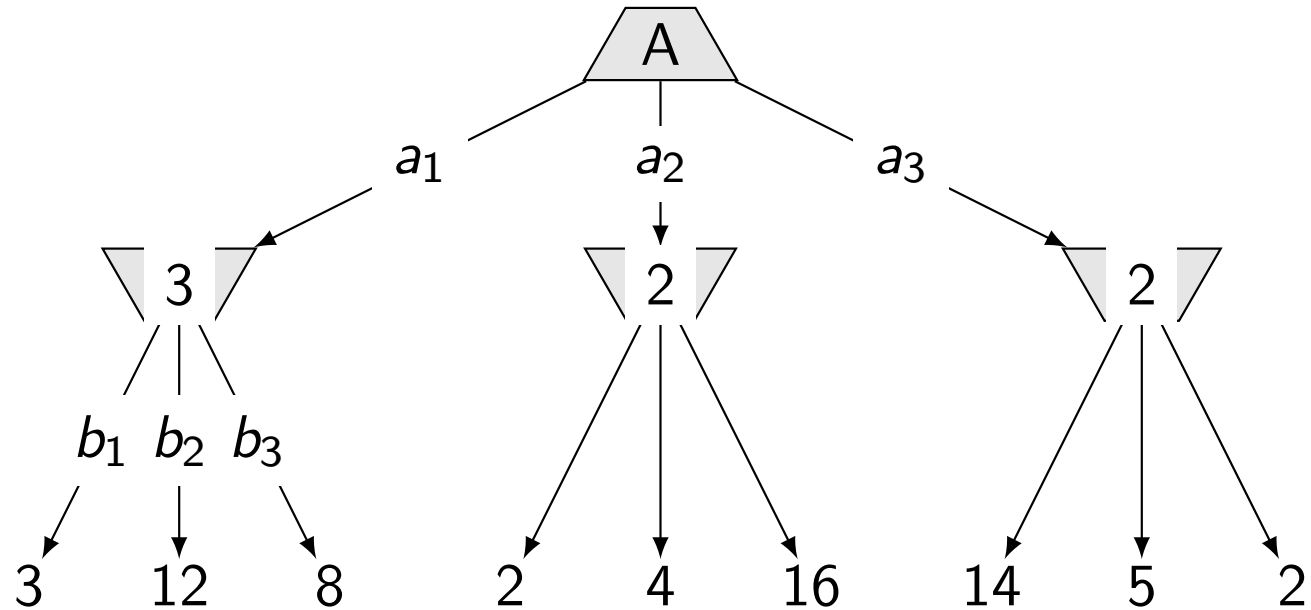
$$a_1 = \arg\max_{a \in \text{ACTIONS}(A)} \text{RESULT}(A, a)$$

Two-ply game: **max** for me, **min** for the opponent.



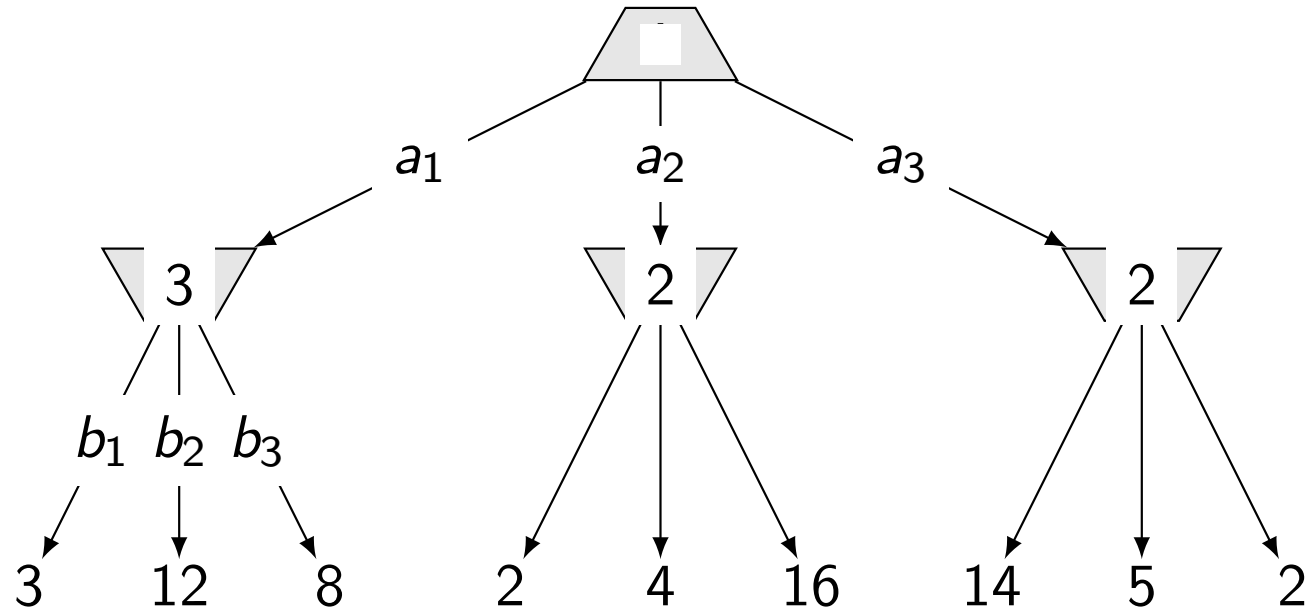
$$a_1 = \arg\max_{a \in \text{ACTIONS}(A)} \text{RESULT}(A, a)$$

Two-ply game: **max** for me, **min** for the opponent.



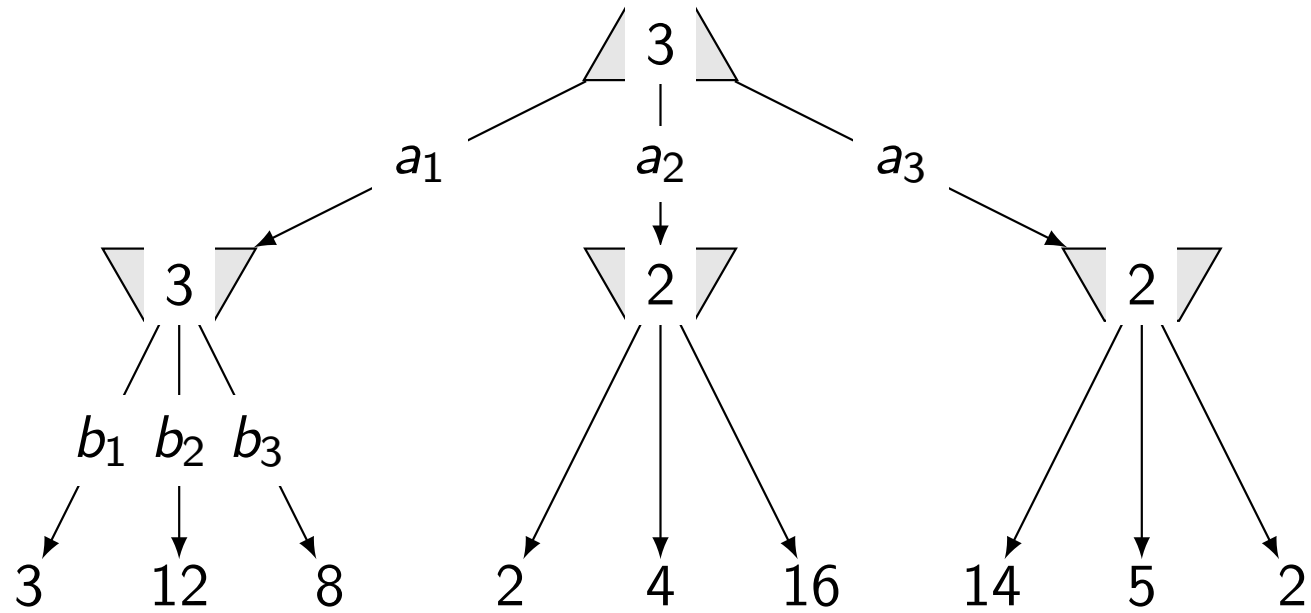
$$a_1 = \arg\max_{a \in \text{ACTIONS}(A)} \text{RESULT}(A, a)$$

Two-ply game: **max** for me, **min** for the opponent.



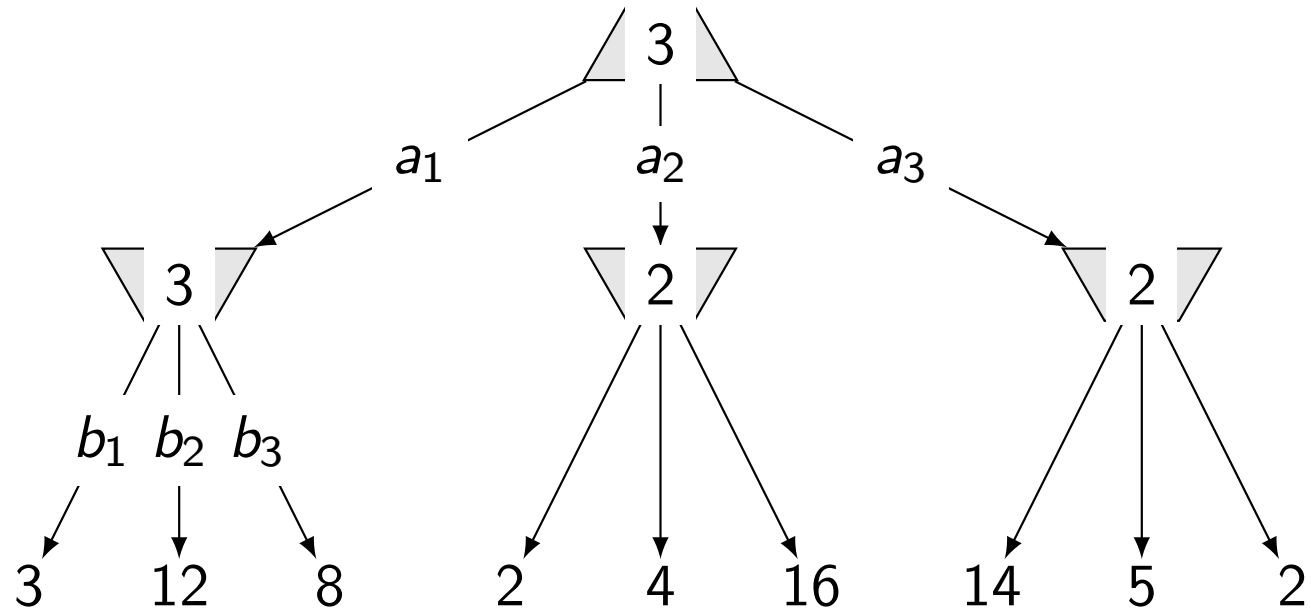
$$a_1 = \arg\max_{a \in \text{ACTIONS}(A)} \text{RESULT}(A, a)$$

Two-ply game: **max** for me, **min** for the opponent.



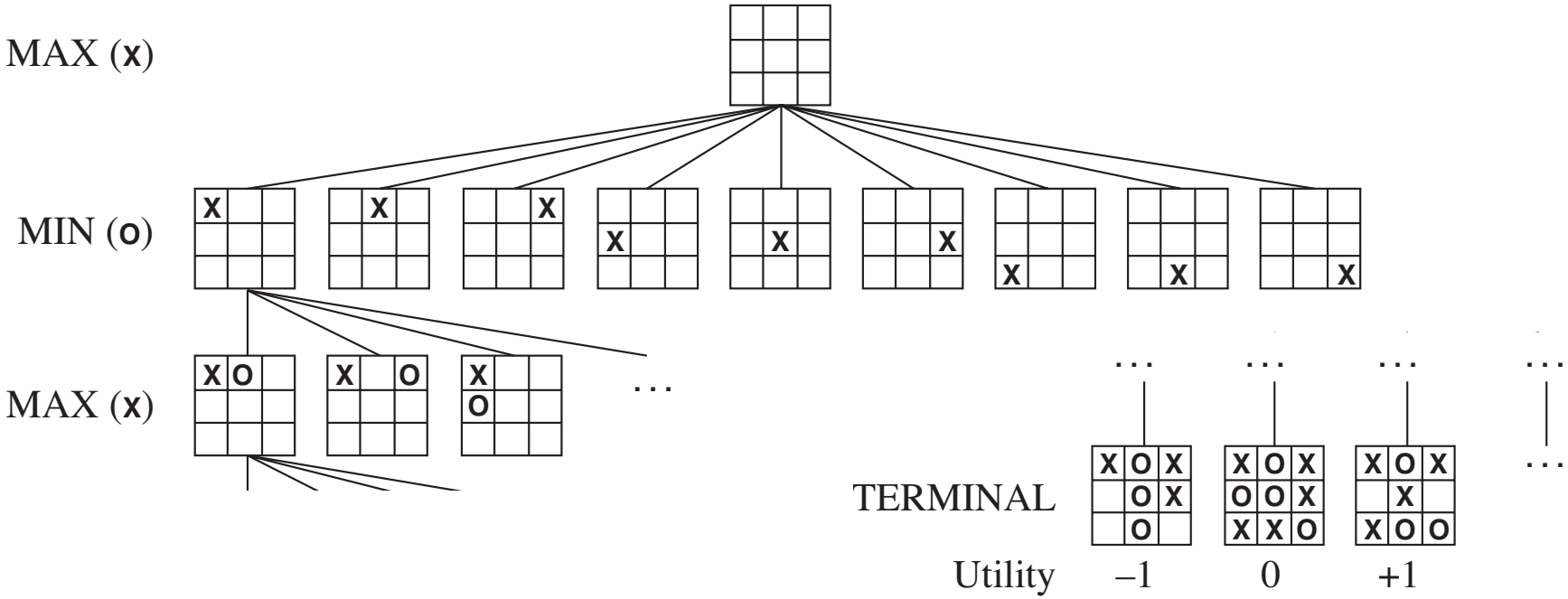
$$a_1 = \arg\max_{a \in \textsc{Actions}(A)} \textsc{Result}(A, a)$$

Two-ply game: **max** for me, **min** for the opponent.



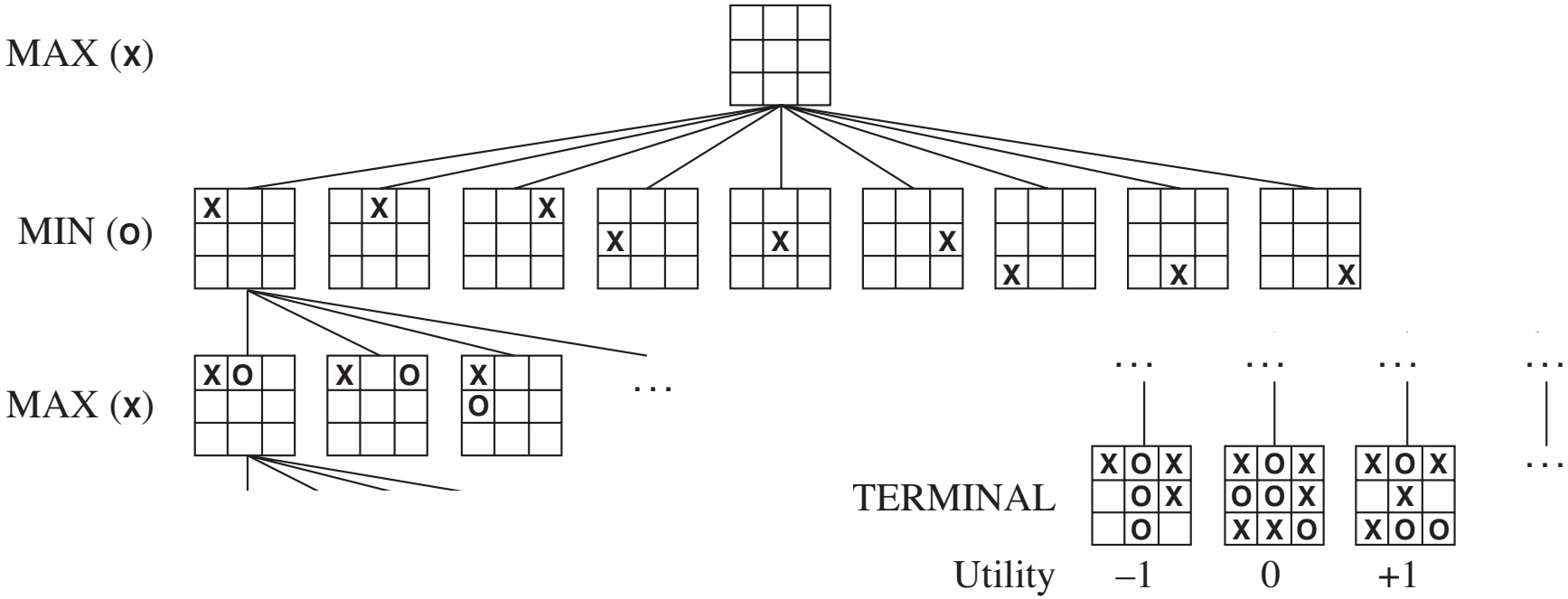
$$a_1 = \underset{a \in \textsc{Actions}(A)}{\arg\max} \; \textsc{Result}(A, a)$$

Zero-Sum game: **max** for me, **min** for the opponent.



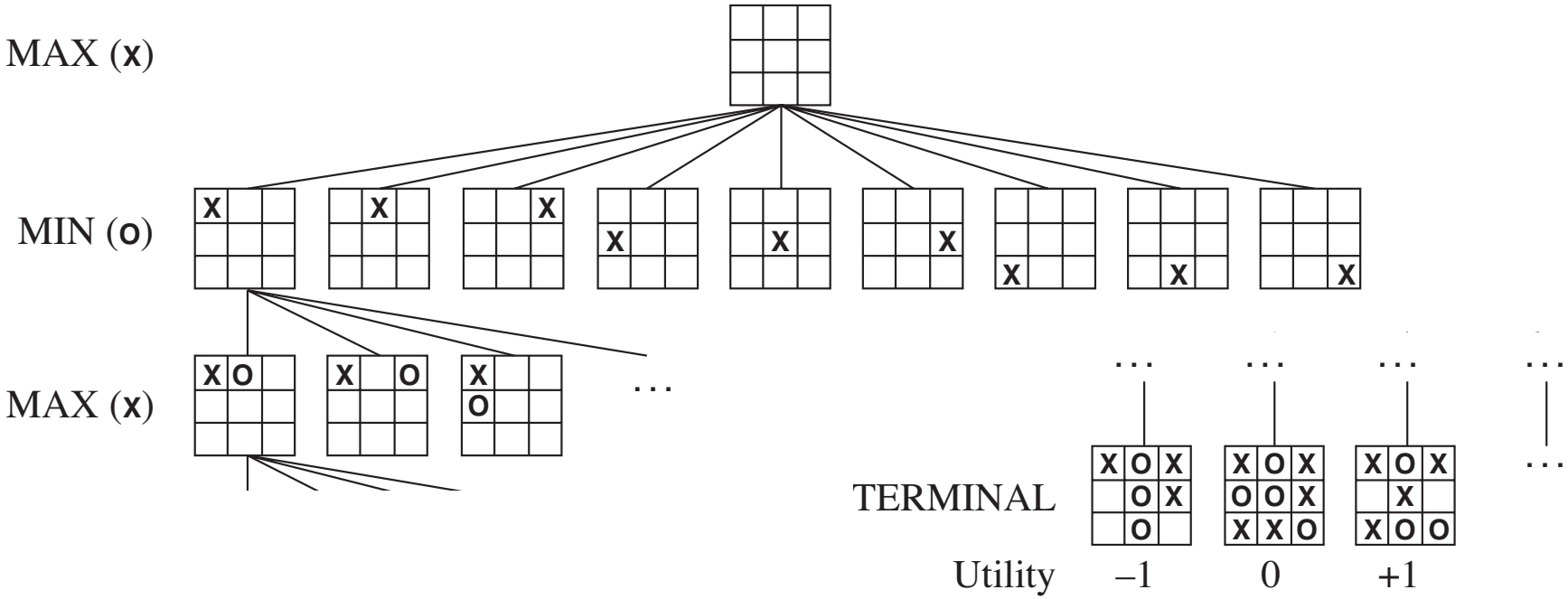
$$\text{MINIMAX}(s) = \begin{cases} \text{UTILITY}(s) & \text{if TERMINAL-TEST}(s) \\ \max_{a \in \text{ACTIONS}(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MAX} \\ \min_{a \in \text{ACTIONS}(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MIN} \end{cases}$$

Zero-Sum game: **max** for me, **min** for the opponent.



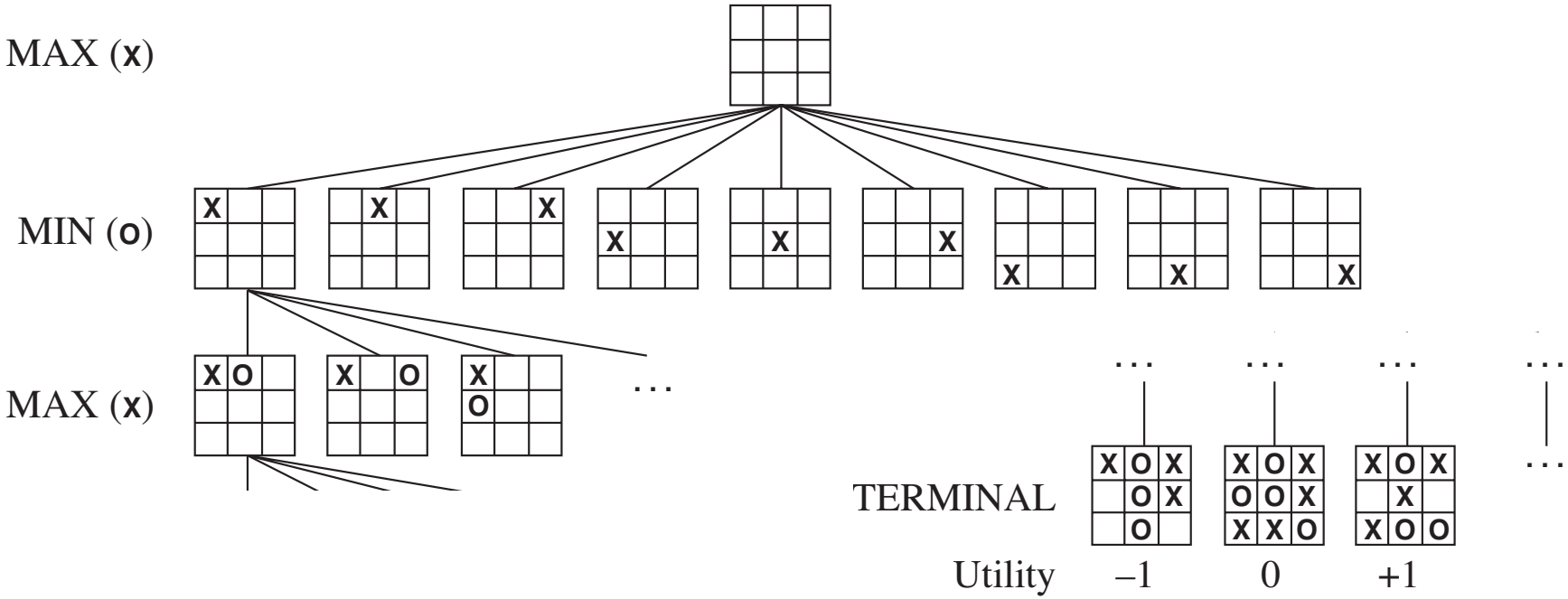
$$\text{MINIMAX}(s) = \begin{cases} \text{UTILITY}(s) & \text{if TERMINAL-TEST}(s) \\ \max_{a \in \text{ACTIONS}(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MAX} \\ \min_{a \in \text{ACTIONS}(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MIN} \end{cases}$$

Zero-Sum game: **max** for me, **min** for the opponent.



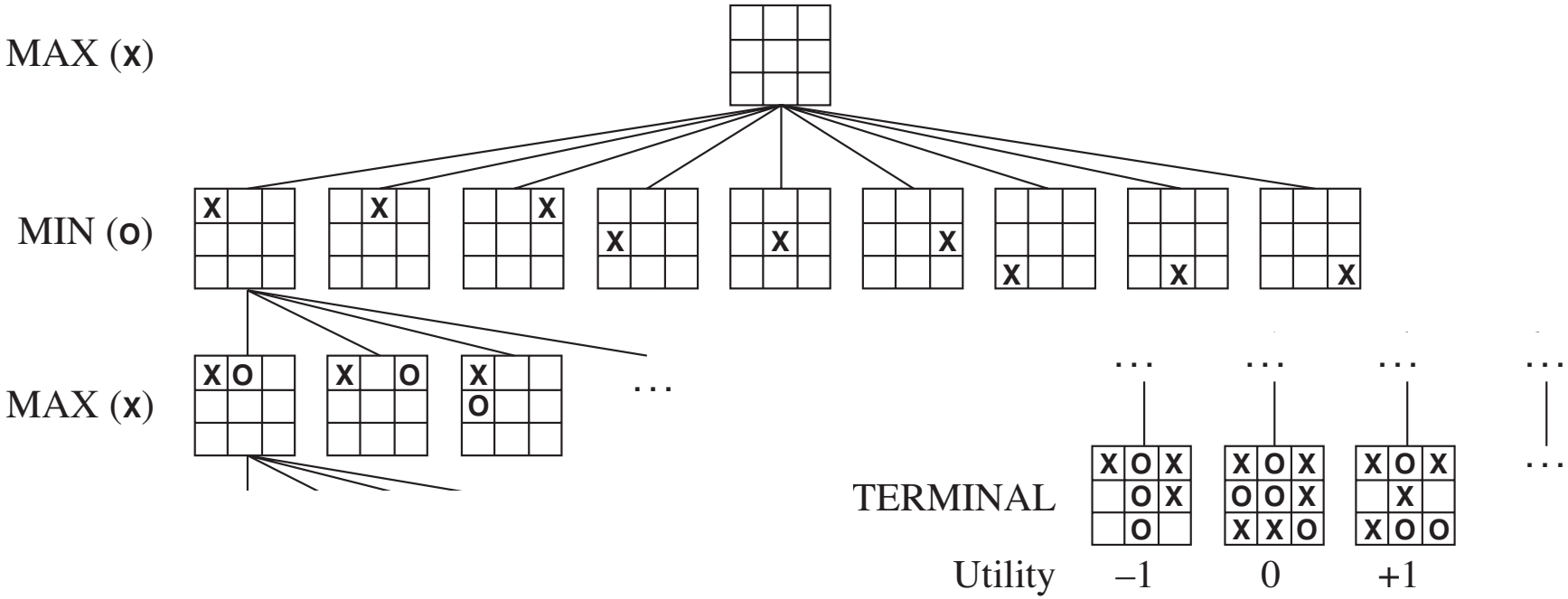
$$\text{MINIMAX}(s) = \begin{cases} \text{UTILITY}(s) & \text{if TERMINAL-TEST}(s) \\ \max_{a \in \text{ACTIONS}(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MAX} \\ \min_{a \in \text{ACTIONS}(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MIN} \end{cases}$$

Zero-Sum game: **max** for me, **min** for the opponent.



$$\text{MINIMAX}(s) = \begin{cases} \text{UTILITY}(s) & \text{if } \text{TERMINAL-TEST}(s) \\ \max_{a \in \text{ACTIONS}(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if } \text{PLAYER}(s) = \text{MAX} \\ \min_{a \in \text{ACTIONS}(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if } \text{PLAYER}(s) = \text{MIN} \end{cases}$$

Zero-Sum game: **max** for me, **min** for the opponent.



$$\text{MINIMAX}(s) = \begin{cases} \text{UTILITY}(s) & \text{if TERMINAL-TEST}(s) \\ \max_{a \in \text{ACTIONS}(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MAX} \\ \min_{a \in \text{ACTIONS}(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MIN} \end{cases}$$

Minimax algorithm

function MINIMAX(state) **returns** an action

    return $\underset{a \in \text{Actions}(s)}{\text{argmax}}$ MIN-VALUE(RESULT(state, a))

end function

function MIN-VALUE(state) **returns** a utility value v

    if TERMINAL-TEST(state) **then return** UTILITY(state)
    end if
    $v \leftarrow \infty$
    for all ACTIONS(state) **do**
        $v \leftarrow$ **min**(v, MAX-VALUE(RESULT(state,a)))
    end for

end function

function MAX-VALUE(state) **returns** a utility value v

    if TERMINAL-TEST(state) **then return** UTILITY(state)
    end if
    $v \leftarrow -\infty$
    for all ACTIONS(state) **do**
        $v \leftarrow$ **max**(v, MIN-VALUE(RESULT(state,a)))
    end for

end function

Minimax algorithm

function MINIMAX(state) **returns** an action
  return $\underset{a \in \text{Actions}(s)}{\text{arg}\mathbf{max}}$ MIN-VALUE(RESULT(state, a))
end function

---

function MIN-VALUE(state) **returns** a utility value v
  if TERMINAL-TEST(state) **then return** UTILITY(state)
  end if
  $v \leftarrow \infty$
  for all ACTIONS(state) **do**
    $v \leftarrow$ **min**(v, MAX-VALUE(RESULT(state,a)))
  end for
end function

function MAX-VALUE(state) **returns** a utility value v
  if TERMINAL-TEST(state) **then return** UTILITY(state)
  end if
  $v \leftarrow -\infty$
  for all ACTIONS(state) **do**
    $v \leftarrow$ **max**(v, MIN-VALUE(RESULT(state,a)))
  end for
end function

Minimax algorithm

function MINIMAX(state) **returns** an action
  return $\underset{a \in \text{Actions}(s)}{\text{arg}\mathbf{max}}$ MIN-VALUE(RESULT(state, a))
end function

---

function MIN-VALUE(state) **returns** a utility value v
  if TERMINAL-TEST(state) **then return** UTILITY(state)
  end if
  $v \leftarrow \infty$
  for all ACTIONS(state) **do**
    $v \leftarrow$ **min**(v, MAX-VALUE(RESULT(state,a)))
  end for
end function

---

function MAX-VALUE(state) **returns** a utility value v
  if TERMINAL-TEST(state) **then return** UTILITY(state)
  end if
  $v \leftarrow -\infty$
  for all ACTIONS(state) **do**
    $v \leftarrow$ **max**(v, MIN-VALUE(RESULT(state,a)))
  end for
end function

Minimax algorithm

function MINIMAX(state) **returns** an action
  return $\underset{a \in \text{Actions}(s)}{\text{arg}\mathbf{max}}$ MIN-VALUE(RESULT(state, a))
end function

---

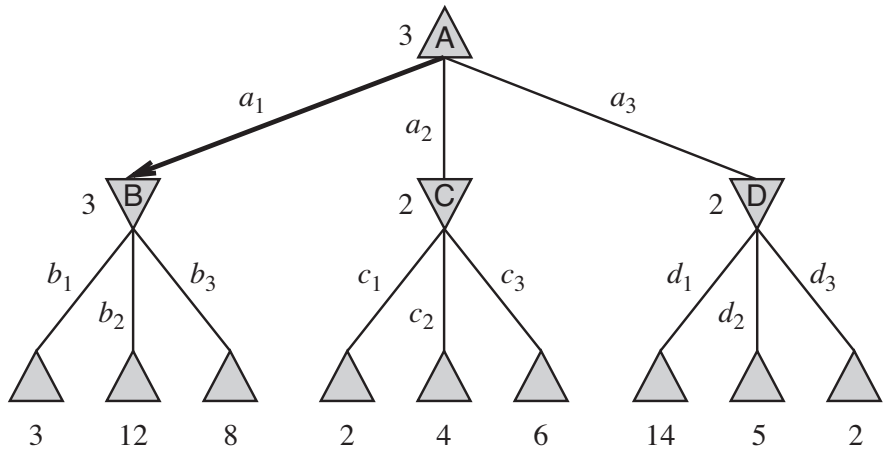
function MIN-VALUE(state) **returns** a utility value v
  if TERMINAL-TEST(state) **then return** UTILITY(state)
  end if
  $v \leftarrow \infty$
  for all ACTIONS(state) **do**
    $v \leftarrow$ **min**(v, MAX-VALUE(RESULT(state,a)))
  end for
end function

---

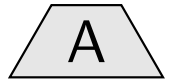
function MAX-VALUE(state) **returns** a utility value v
  if TERMINAL-TEST(state) **then return** UTILITY(state)
  end if
  $v \leftarrow -\infty$
  for all ACTIONS(state) **do**
    $v \leftarrow$ **max**(v, MIN-VALUE(RESULT(state,a)))
  end for
end function

A two ply game, down to terminal and back again . . .

```
function MINIMAX(s) returns a                                   MAX
    argmax_{a∈Actions(s)} MINVAL(RES(s, a))
end function
-------------------------------------------------------------   MIN
function MINVAL(s) returns v
    if TERMINAL(s) then UTIL(s)
    end if
    v ← ∞
    for all ACTIONS(s) do
        v ← min(v, MAXVAL(RES(s, a)))
    end for
end function
-------------------------------------------------------------
function MAXVAL(s) returns v
    if TERMINAL(s) then UTIL(s)
    end if
    v ← −∞
    for all ACTIONS(s) do
        v ← max(v, MINVAL(RES(s, a)))
    end for
end function
```



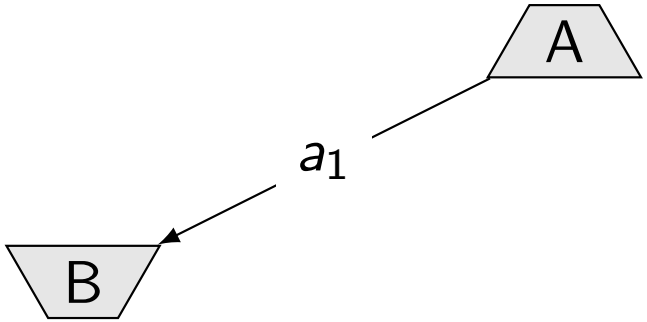
A two ply game, recursive run



What is the complexity? How many nodes to visit?

Can we do better? How?

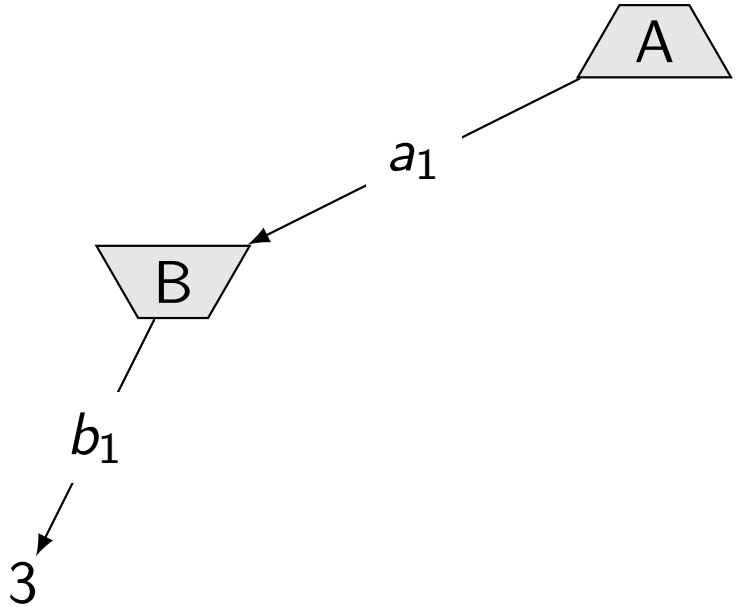
A two ply game, recursive run



What is the complexity? How many nodes to visit?

Can we do better? How?

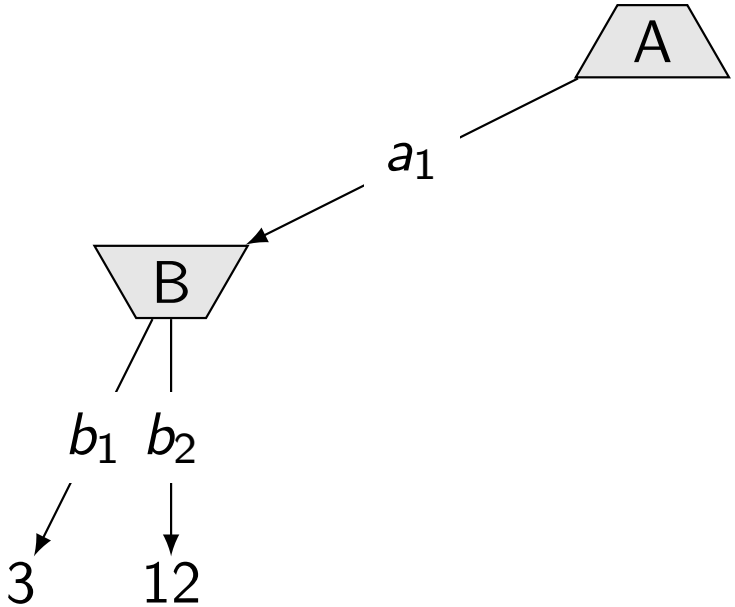
A two ply game, recursive run



What is the complexity? How many nodes to visit?

Can we do better? How?

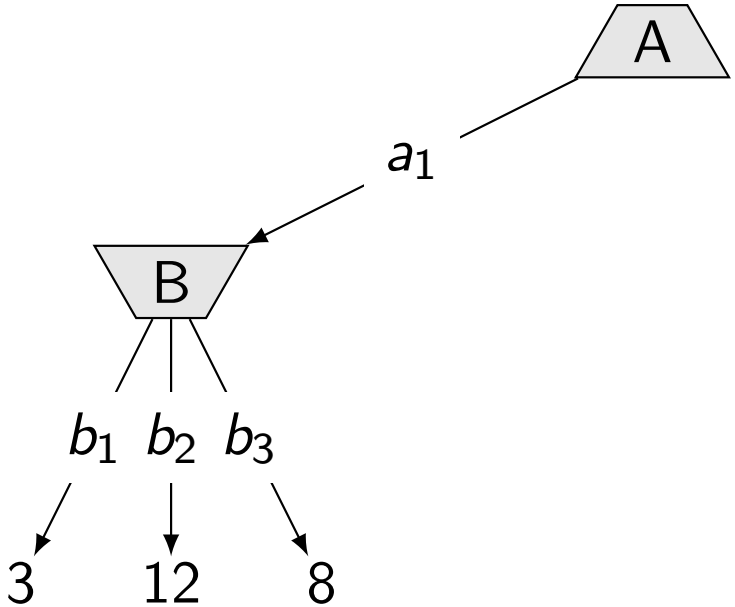
A two ply game, recursive run



What is the complexity? How many nodes to visit?

Can we do better? How?

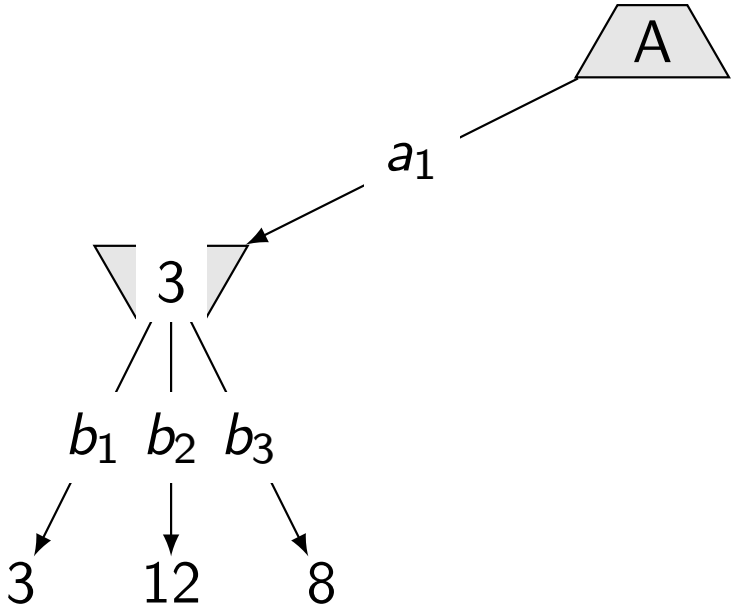
A two ply game, recursive run



What is the complexity? How many nodes to visit?

Can we do better? How?

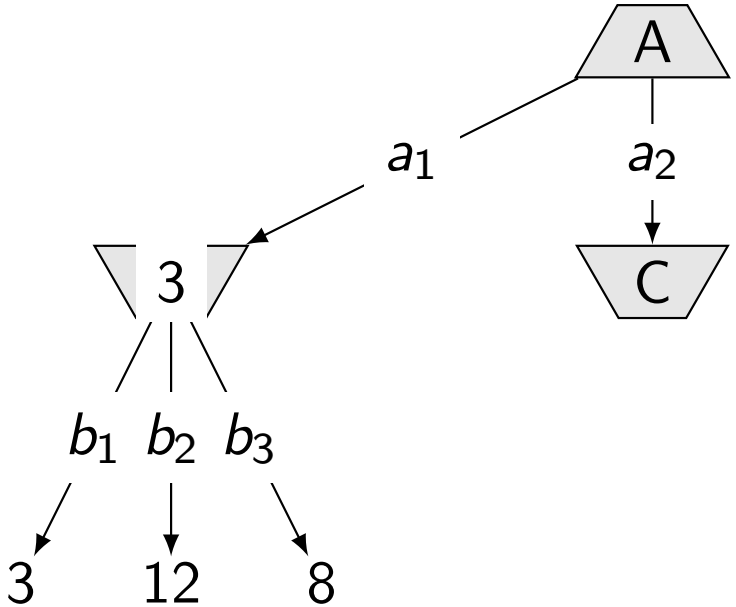
A two ply game, recursive run



What is the complexity? How many nodes to visit?

Can we do better? How?

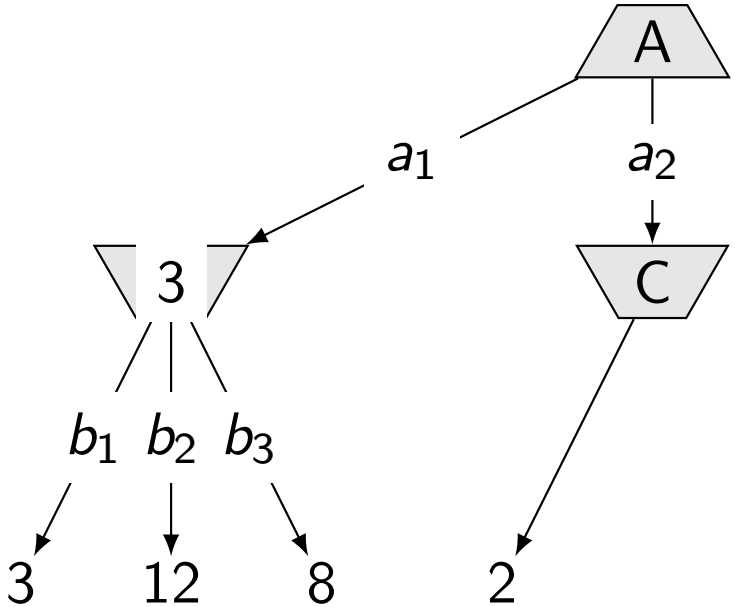
A two ply game, recursive run



What is the complexity? How many nodes to visit?

Can we do better? How?

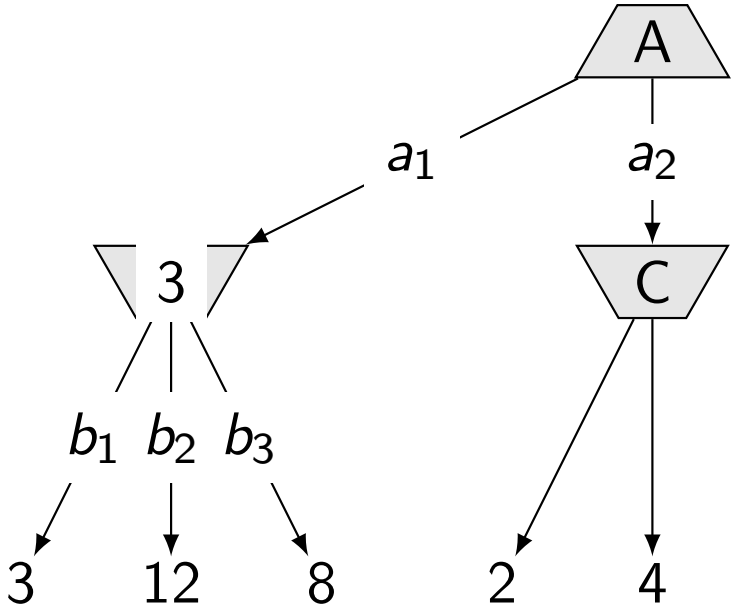
A two ply game, recursive run



What is the complexity? How many nodes to visit?

Can we do better? How?

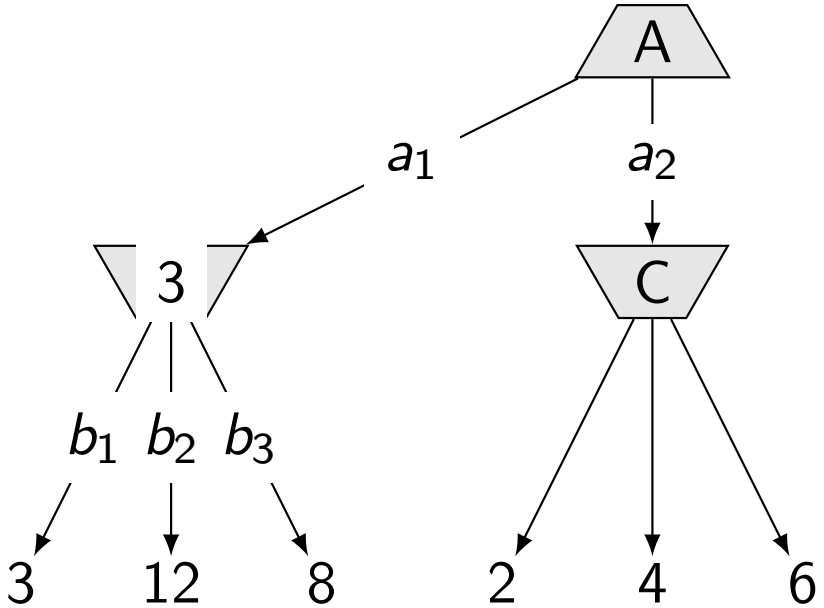
A two ply game, recursive run



What is the complexity? How many nodes to visit?

Can we do better? How?

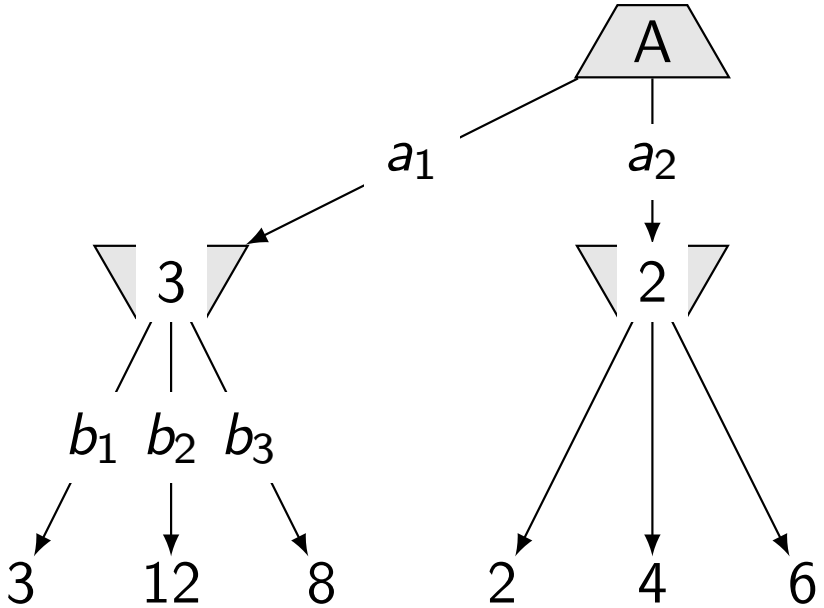
A two ply game, recursive run



What is the complexity? How many nodes to visit?

Can we do better? How?

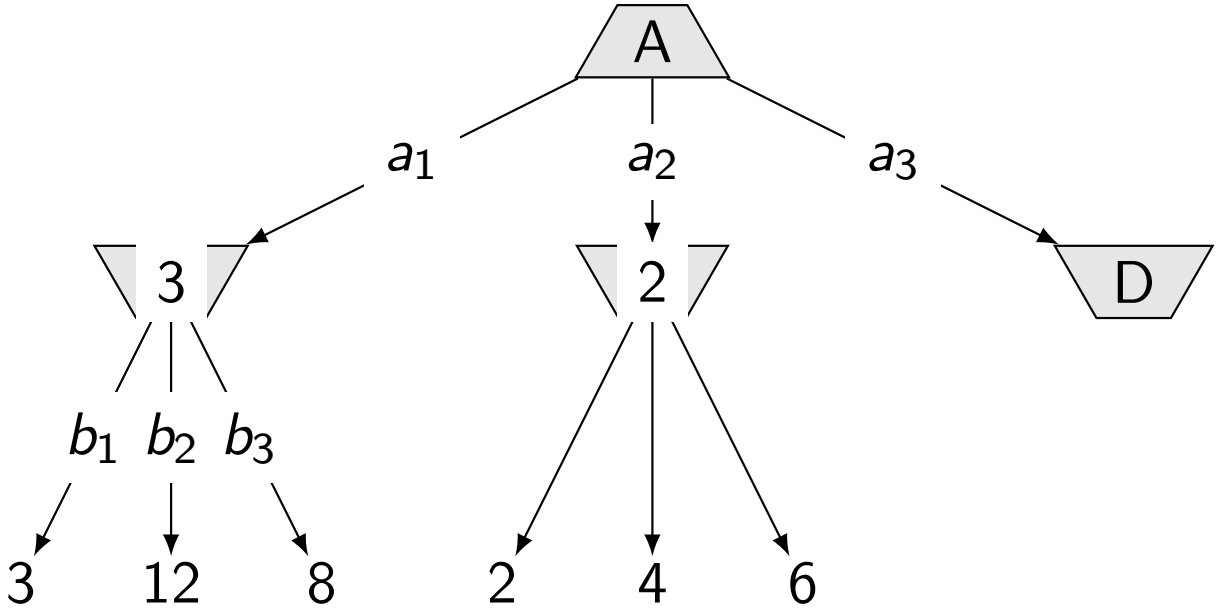
A two ply game, recursive run



What is the complexity? How many nodes to visit?

Can we do better? How?

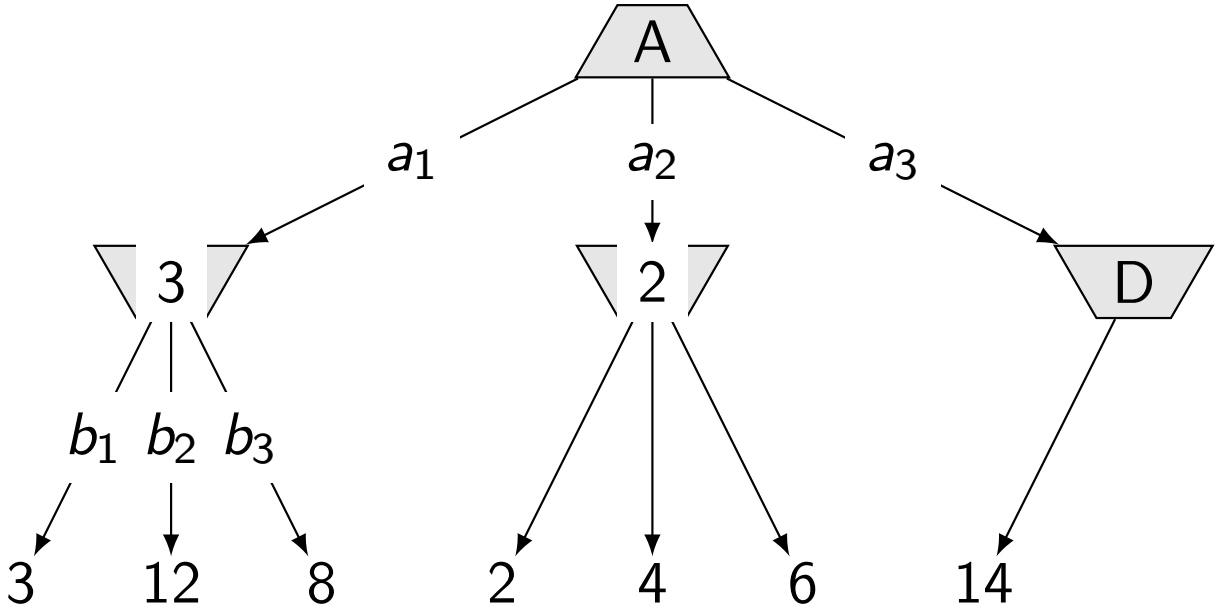
A two ply game, recursive run



What is the complexity? How many nodes to visit?

Can we do better? How?

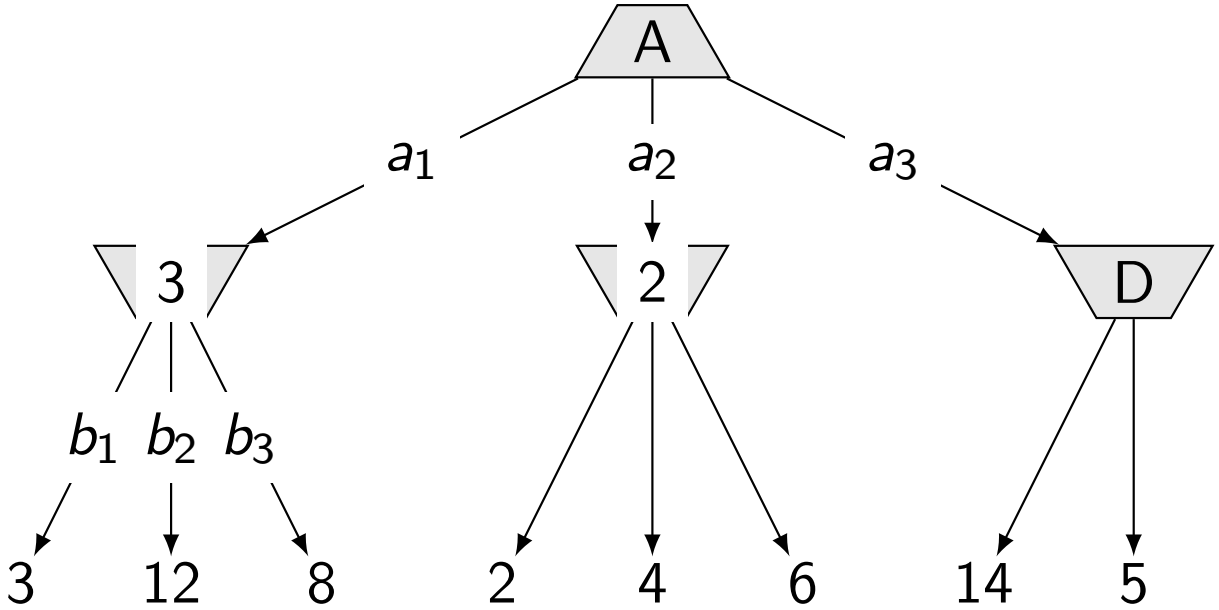
A two ply game, recursive run



What is the complexity? How many nodes to visit?

Can we do better? How?

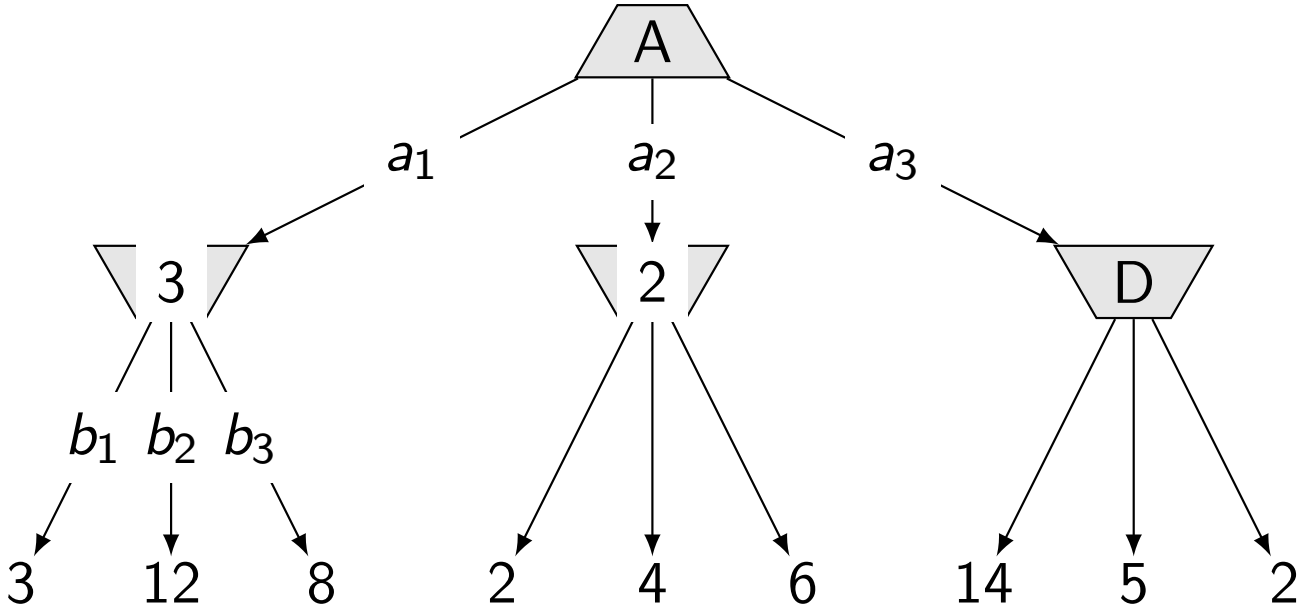
A two ply game, recursive run



What is the complexity? How many nodes to visit?

Can we do better? How?

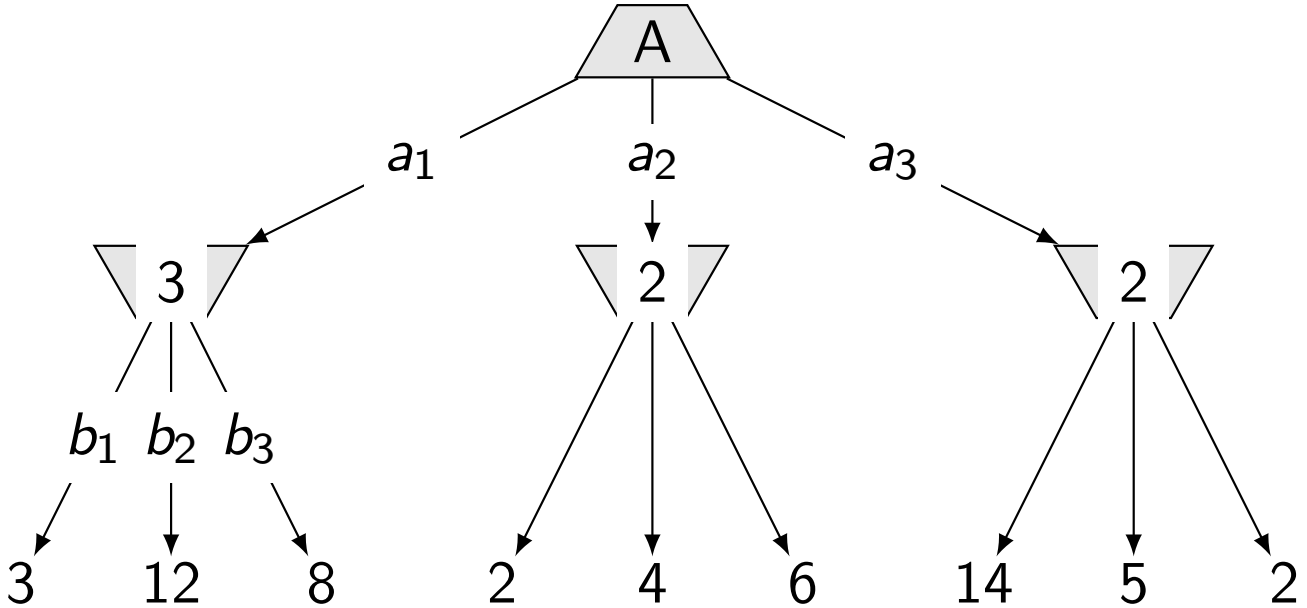
A two ply game, recursive run



What is the complexity? How many nodes to visit?

Can we do better? How?

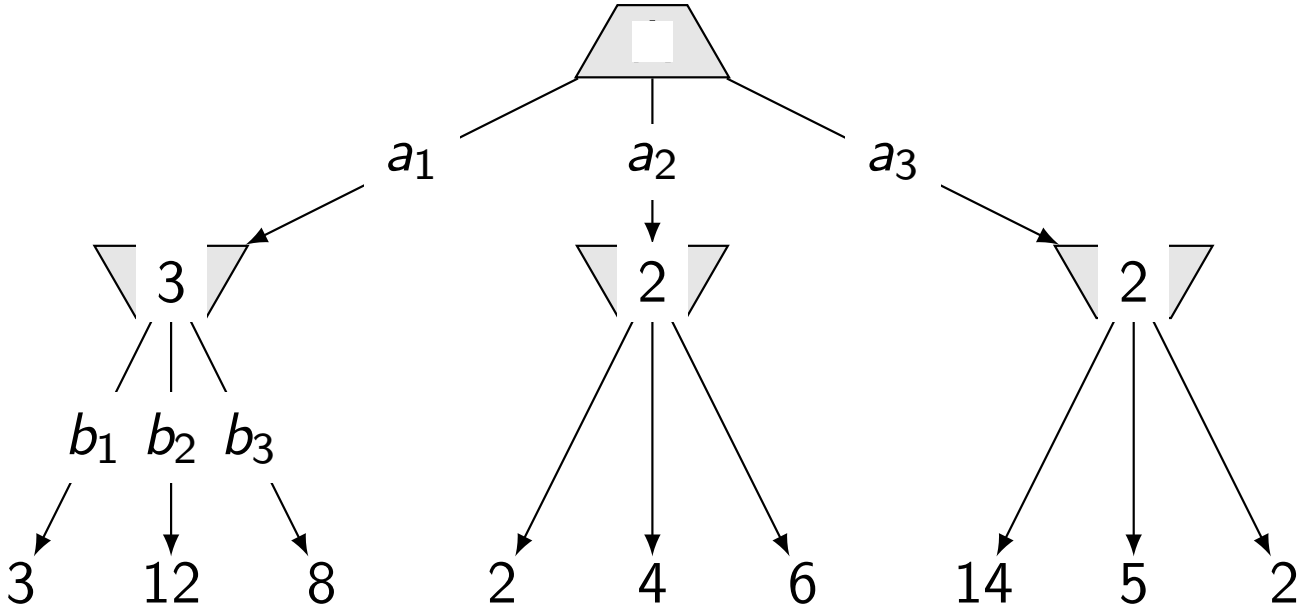
A two ply game, recursive run



What is the complexity? How many nodes to visit?

Can we do better? How?

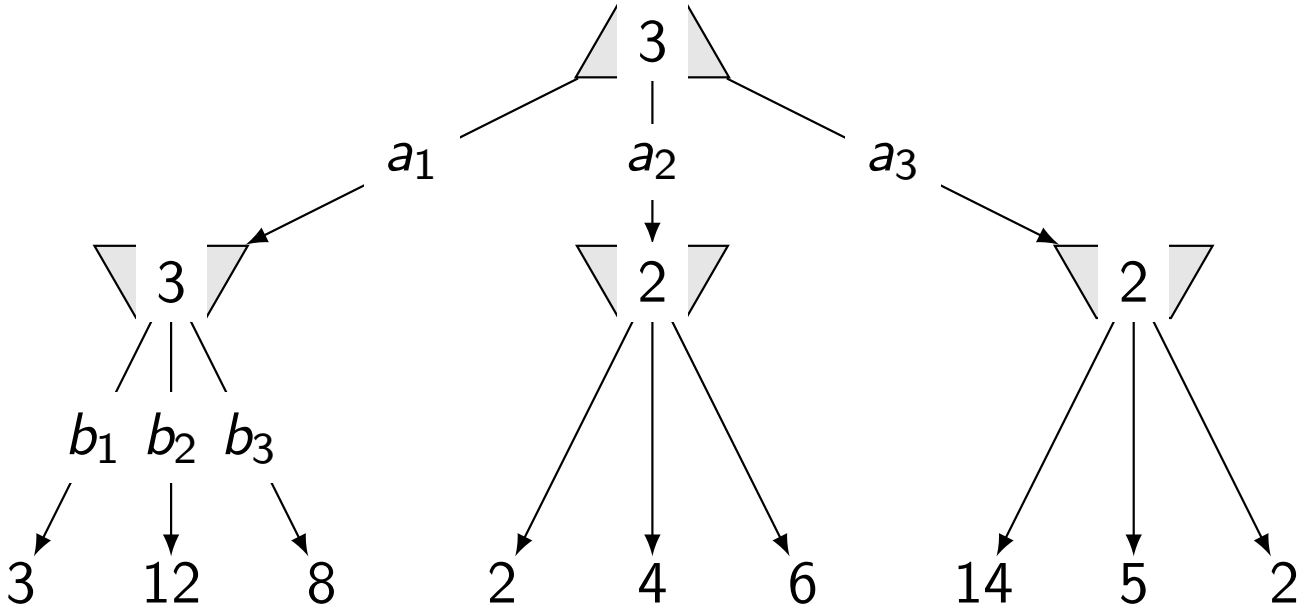
A two ply game, recursive run



What is the complexity? How many nodes to visit?

Can we do better? How?

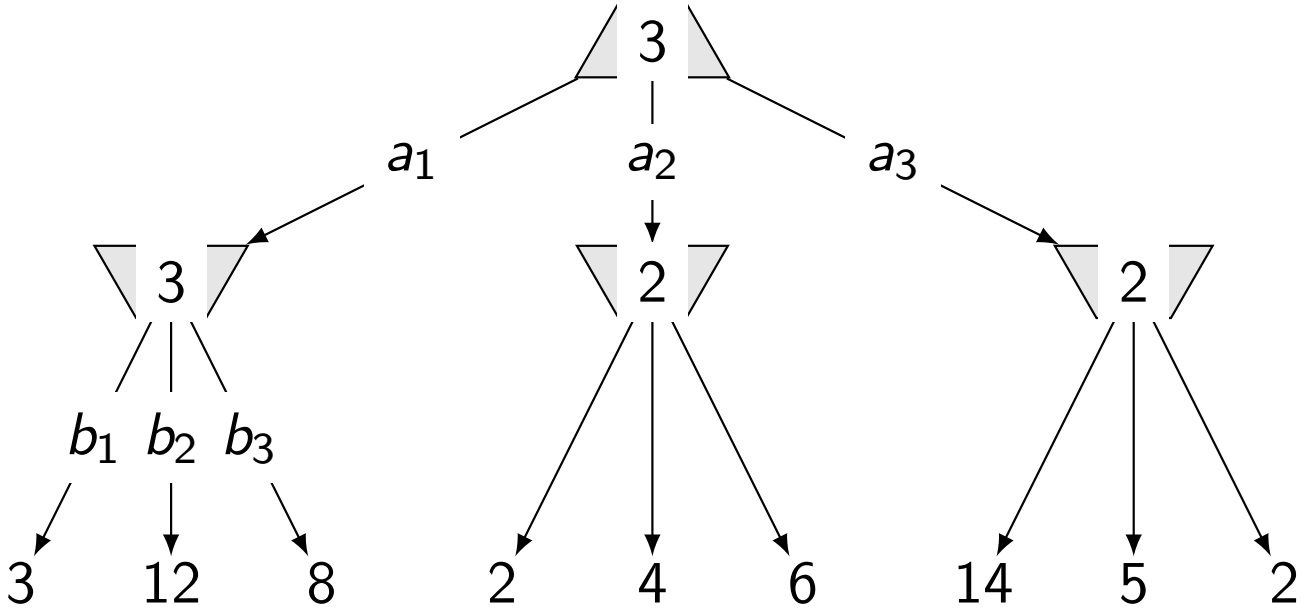
A two ply game, recursive run



What is the complexity? How many nodes to visit?

Can we do better? How?

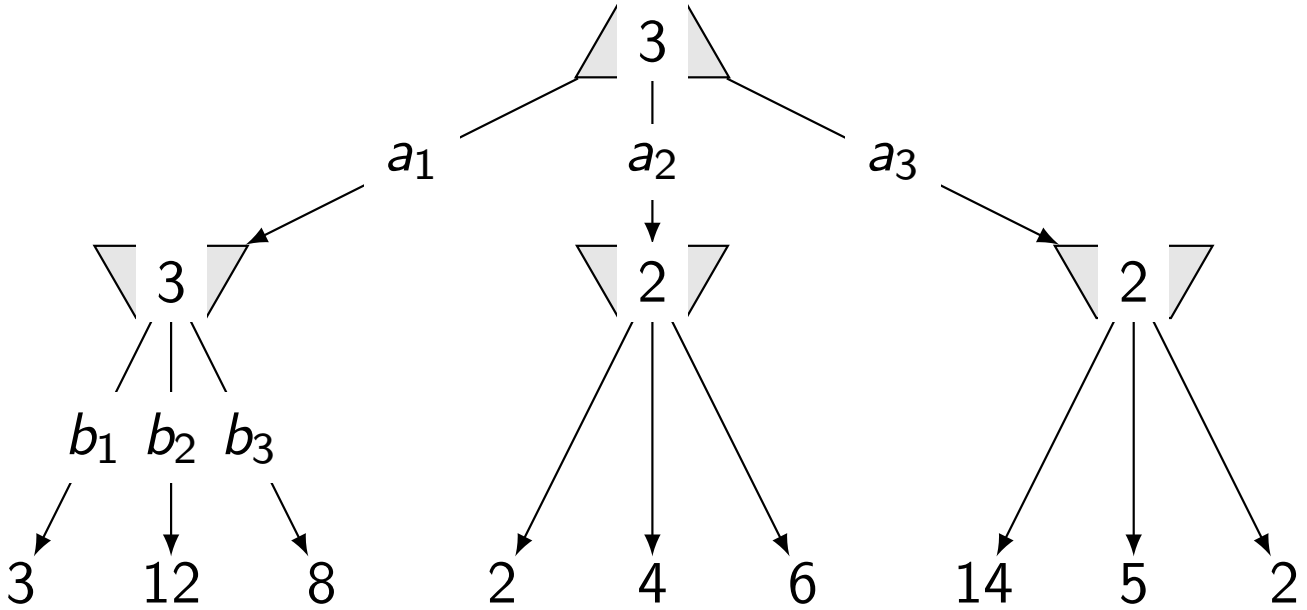
A two ply game, recursive run



What is the complexity? How many nodes to visit?

Can we do better? How?

A two ply game, recursive run

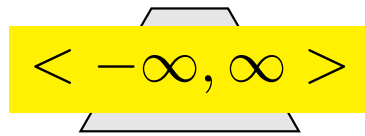


What is the complexity? How many nodes to visit?

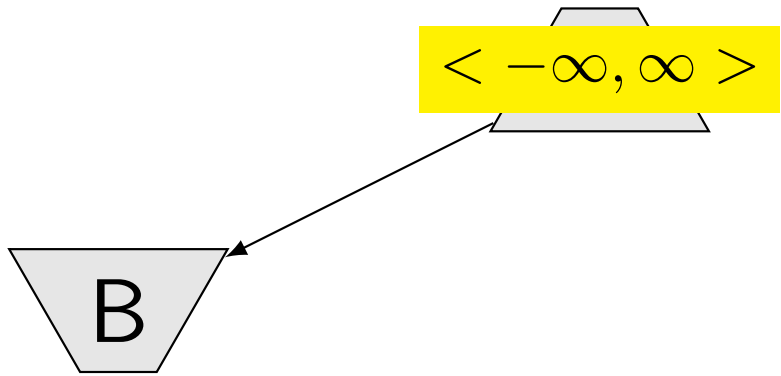
Can we do better? How?

Nodes (sub-trees) worth visiting

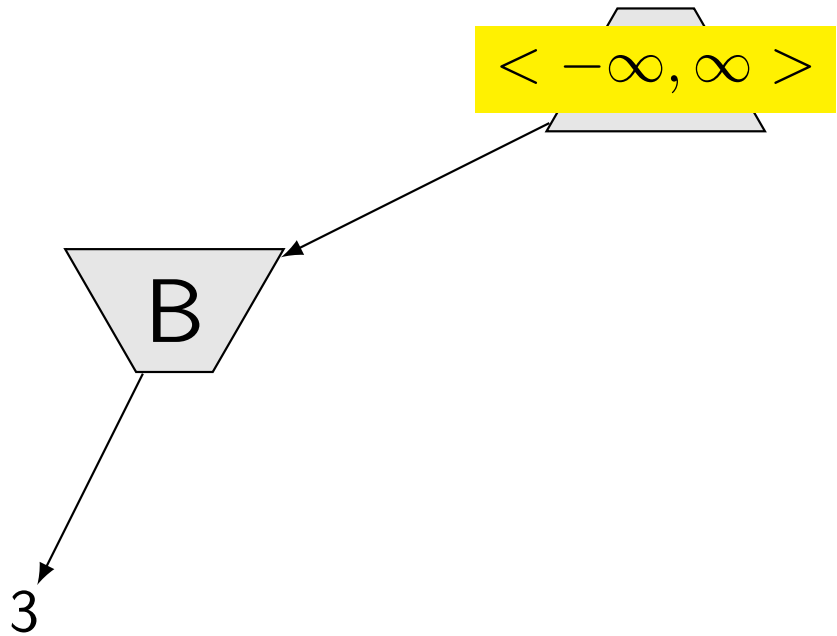
Nodes (sub-trees) worth visiting



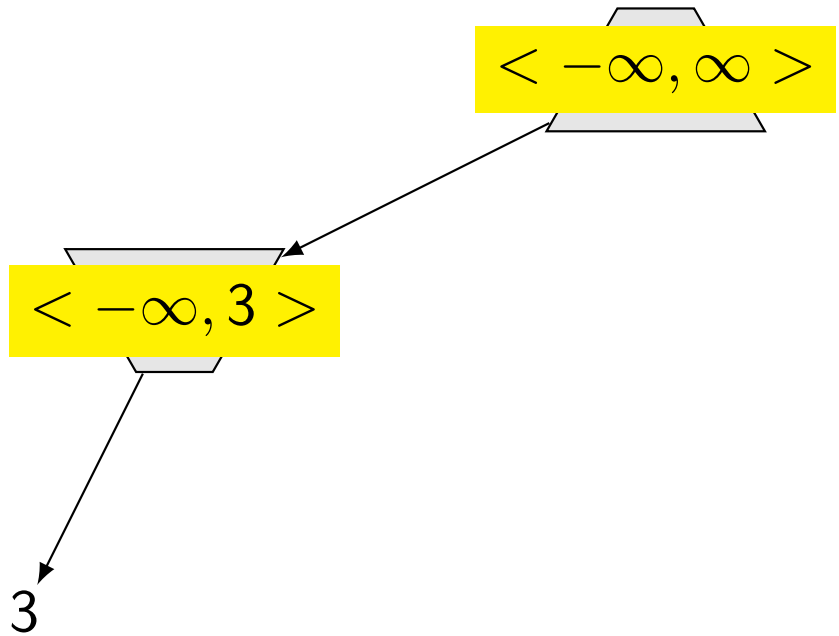
Nodes (sub-trees) worth visiting



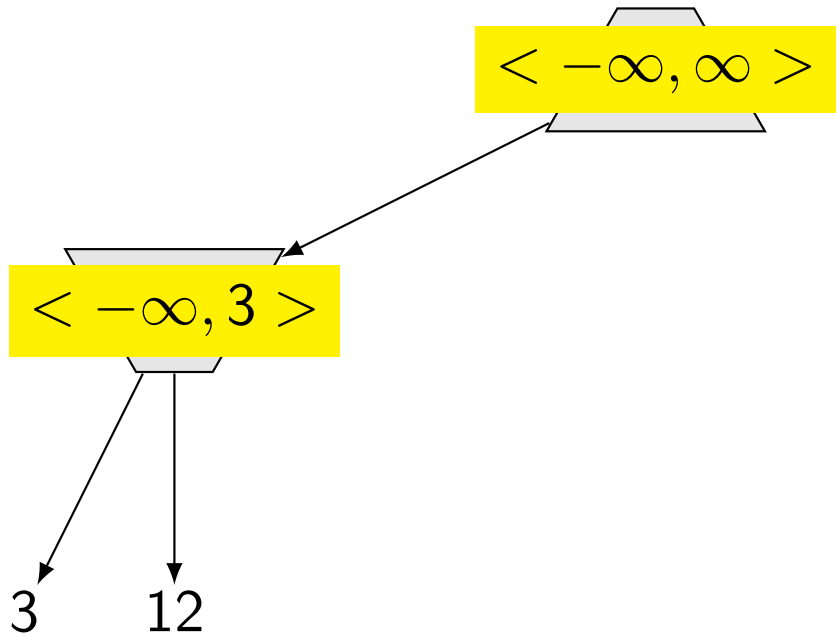
Nodes (sub-trees) worth visiting



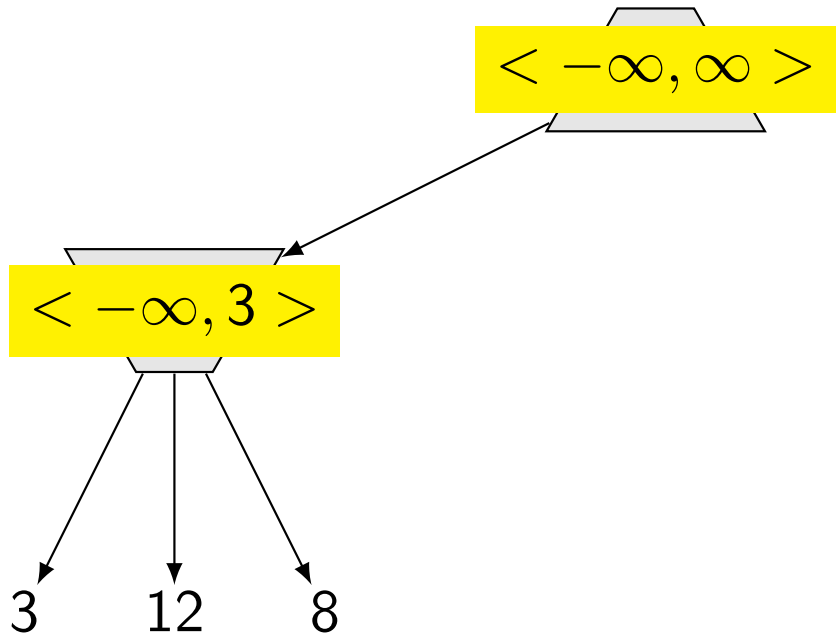
Nodes (sub-trees) worth visiting



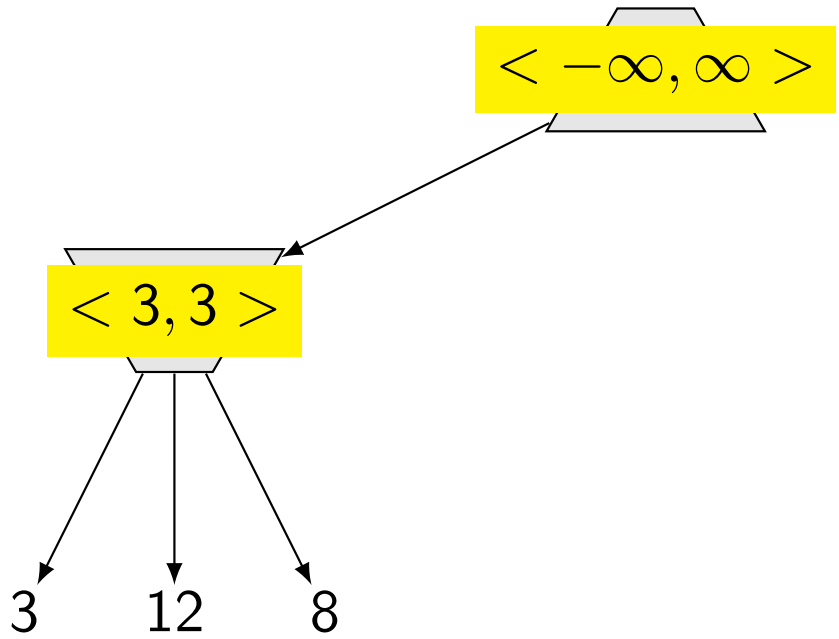
Nodes (sub-trees) worth visiting



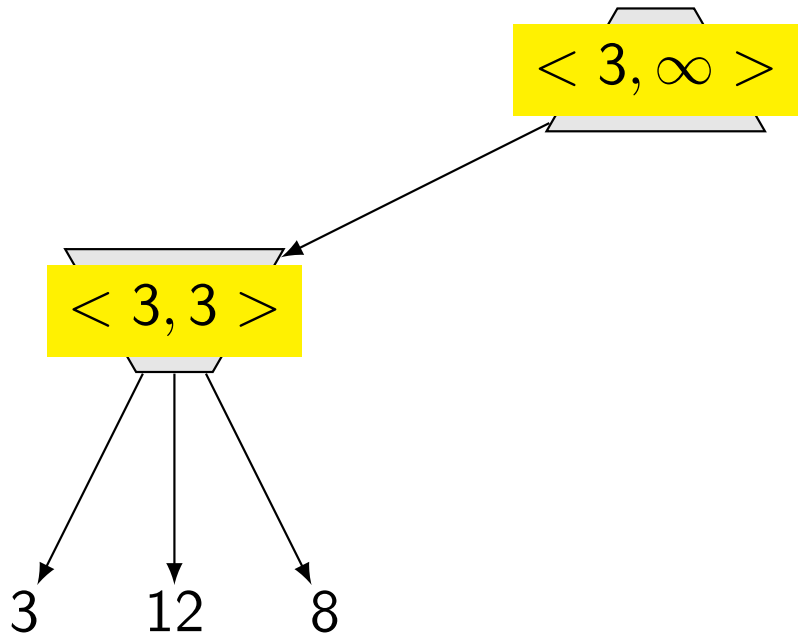
Nodes (sub-trees) worth visiting



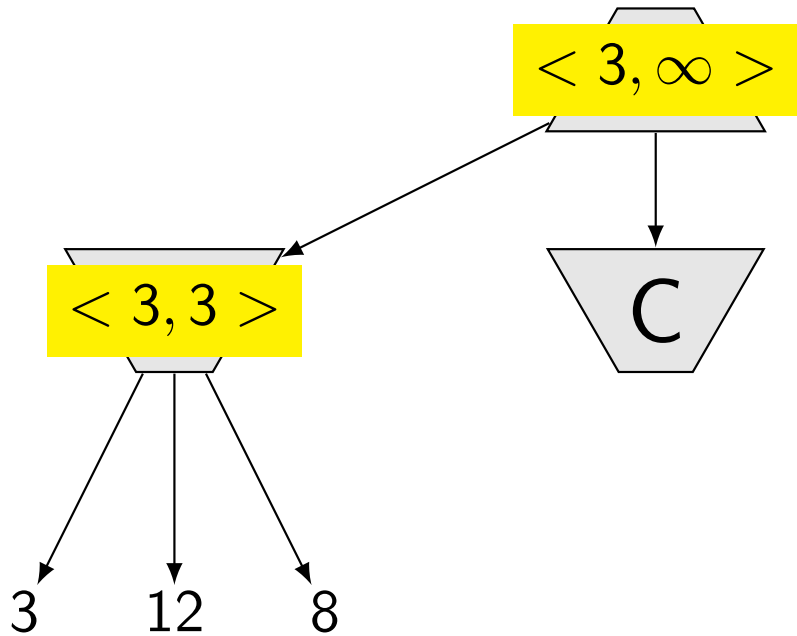
Nodes (sub-trees) worth visiting



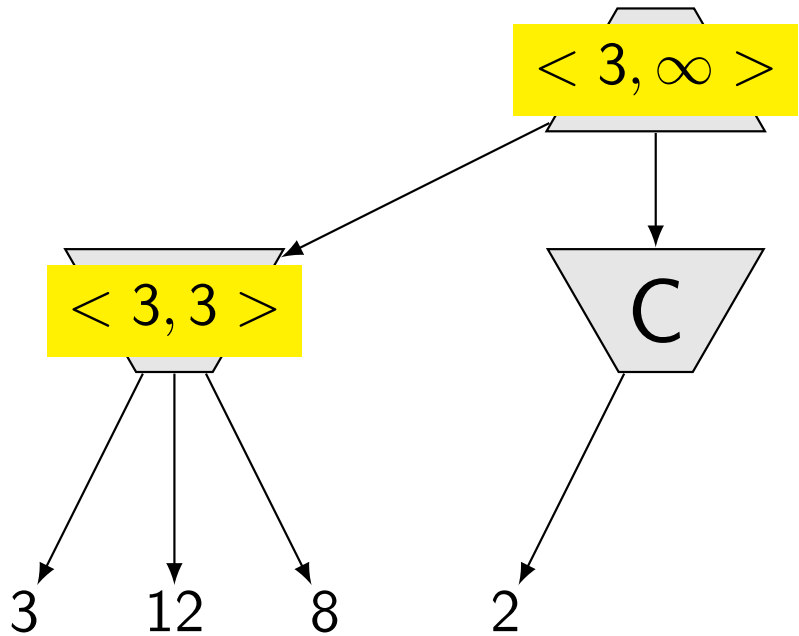
Nodes (sub-trees) worth visiting



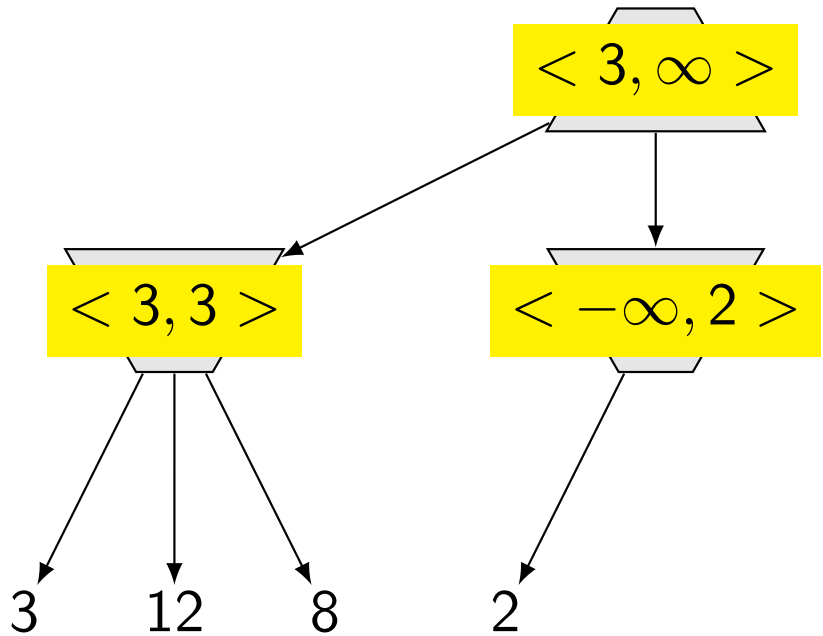
Nodes (sub-trees) worth visiting



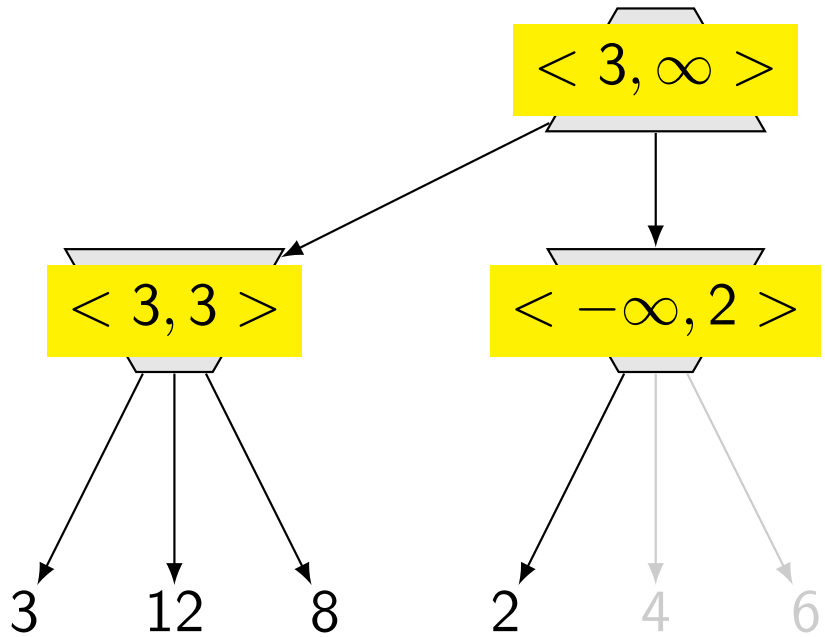
Nodes (sub-trees) worth visiting



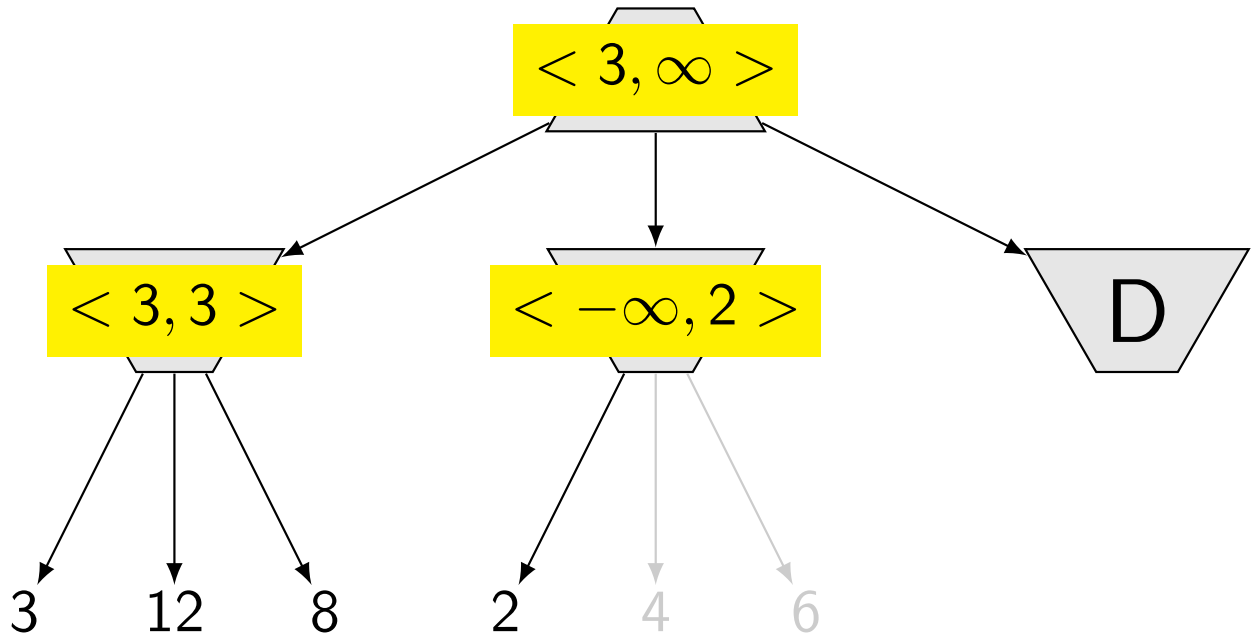
Nodes (sub-trees) worth visiting



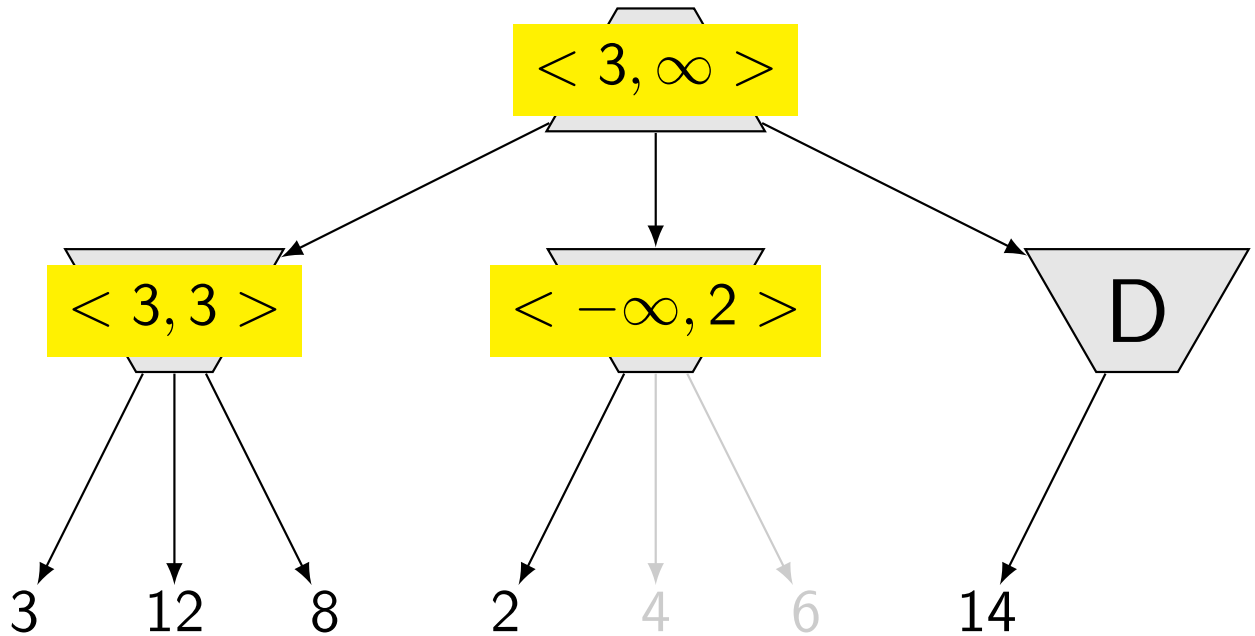
Nodes (sub-trees) worth visiting



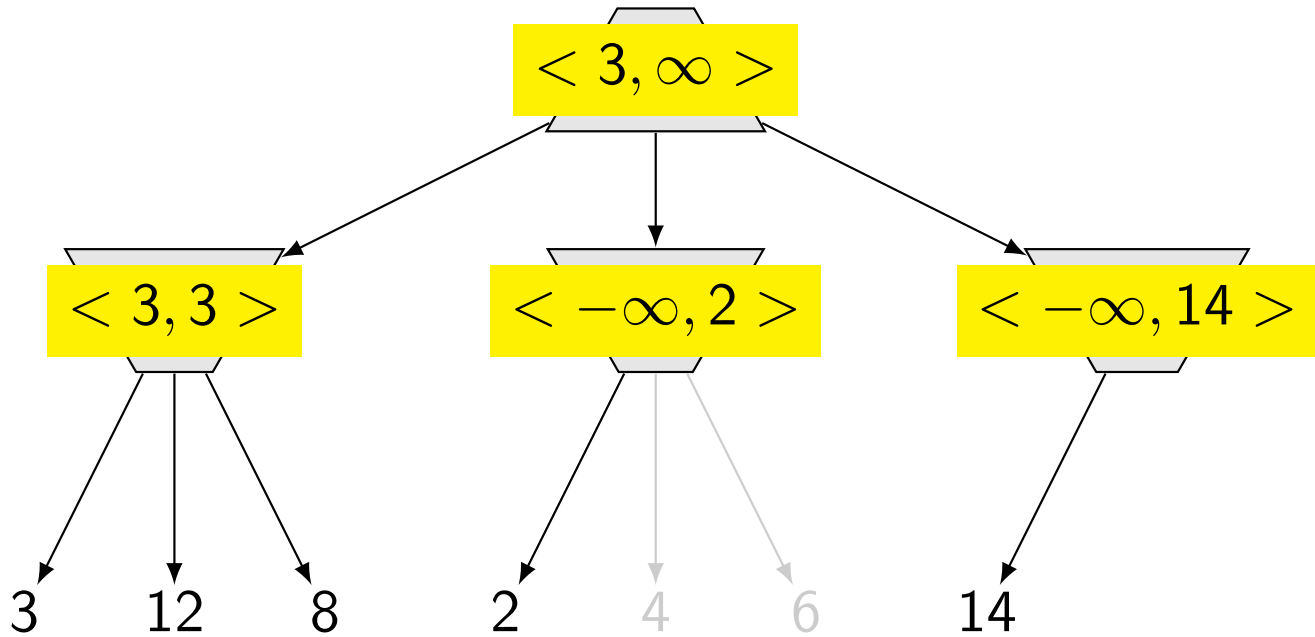
Nodes (sub-trees) worth visiting



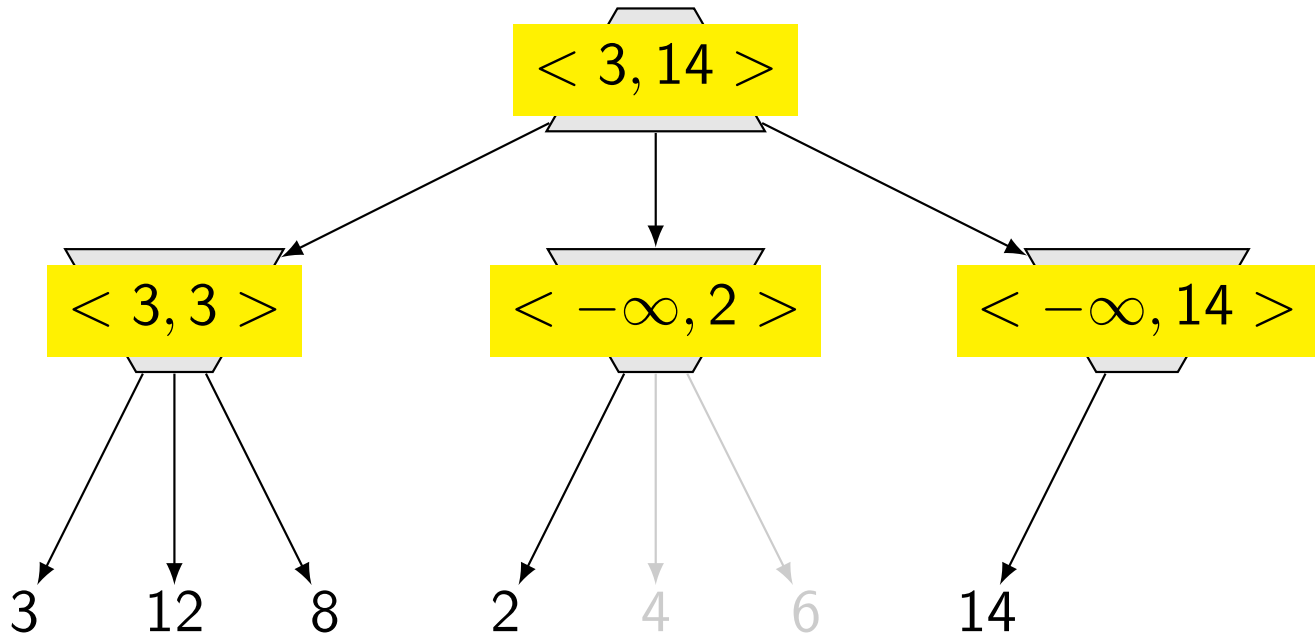
Nodes (sub-trees) worth visiting



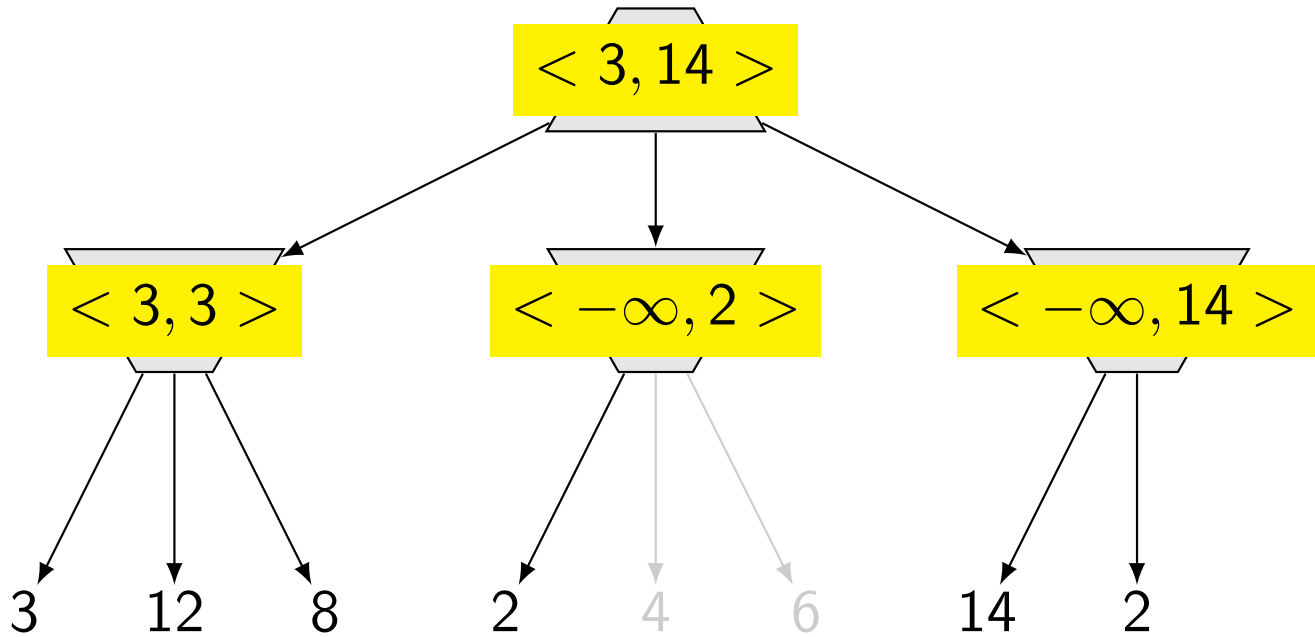
Nodes (sub-trees) worth visiting



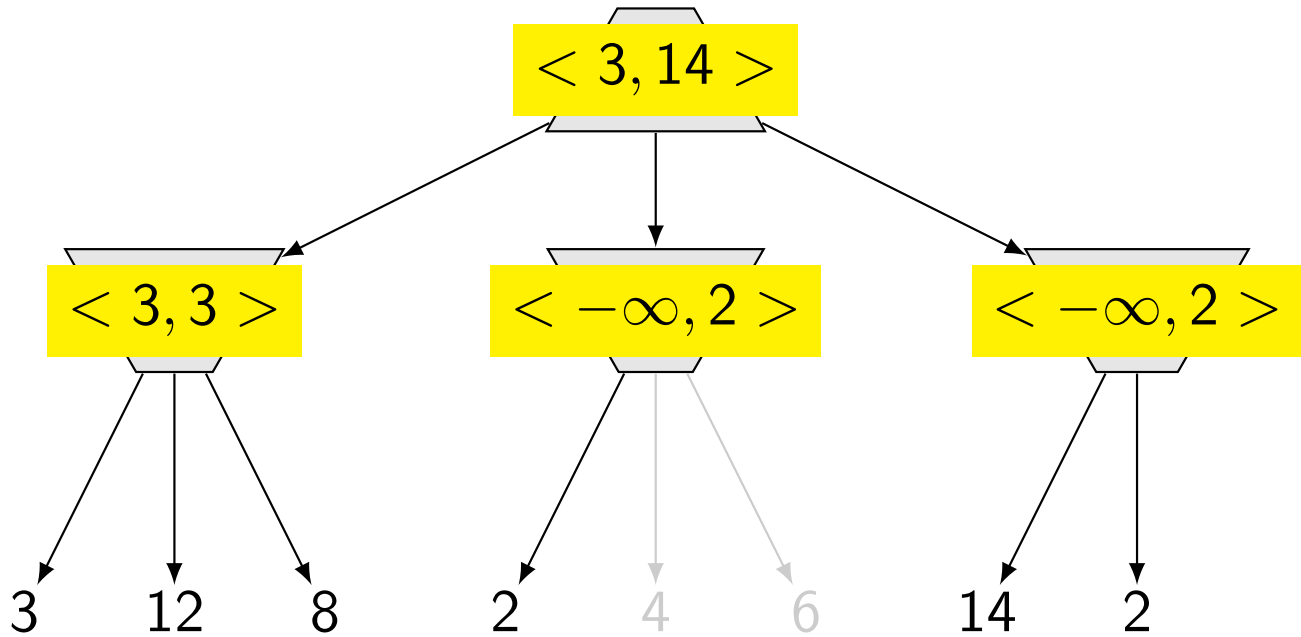
Nodes (sub-trees) worth visiting



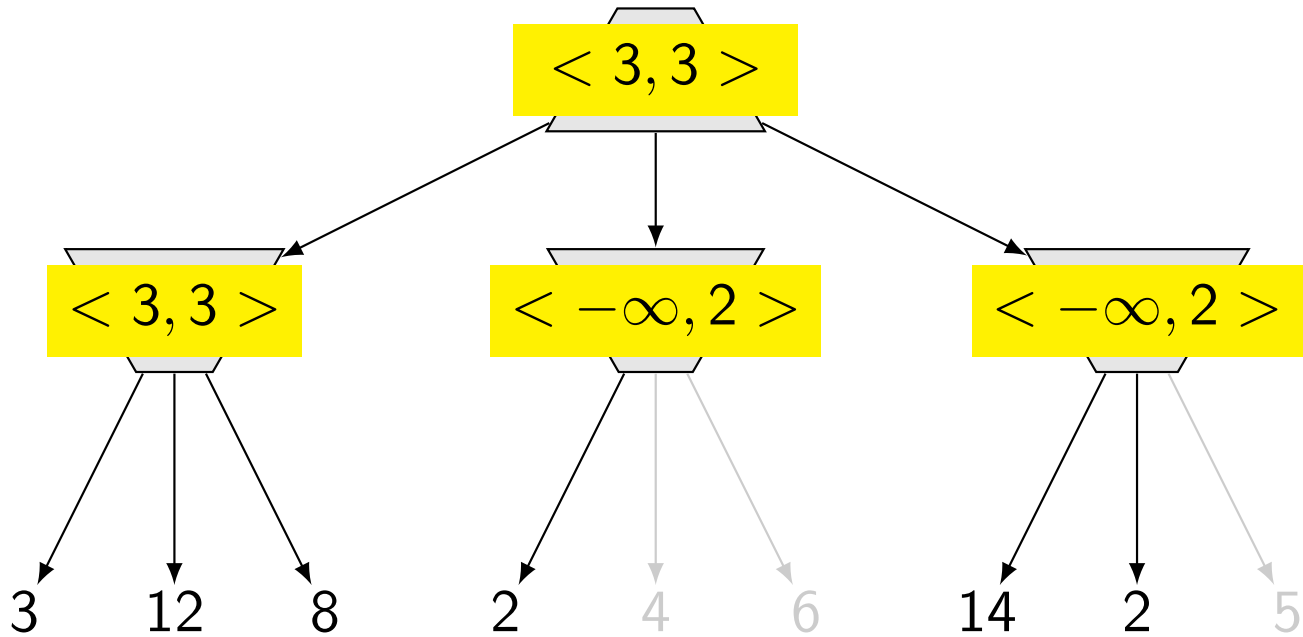
Nodes (sub-trees) worth visiting



Nodes (sub-trees) worth visiting



Nodes (sub-trees) worth visiting



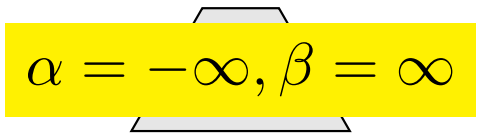
α-β prunning

α highest (best) value choice found so far for any choice along MAX
β lowest (best) value choice found so far for any choice along MIN

In MIN-VAL: $v \leftarrow 2$
$v \leq \alpha$ then: return v!

α-β prunning

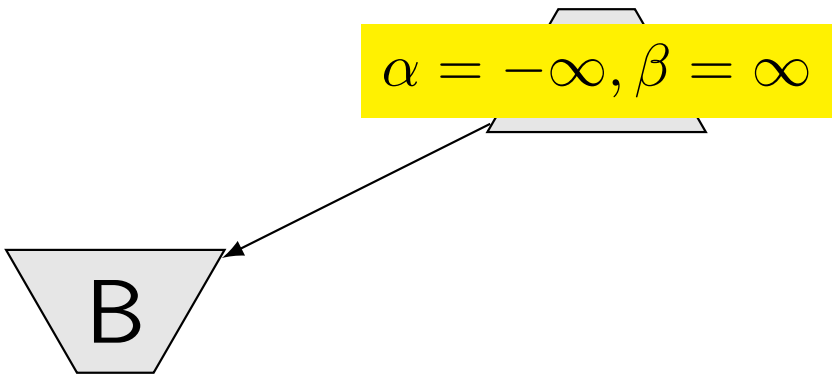
α highest (best) value choice found so far for any choice along MAX
β lowest (best) value choice found so far for any choice along MIN



In MIN-VAL: $v \leftarrow 2$
$v \leq \alpha$ then: return v!

α-β prunning

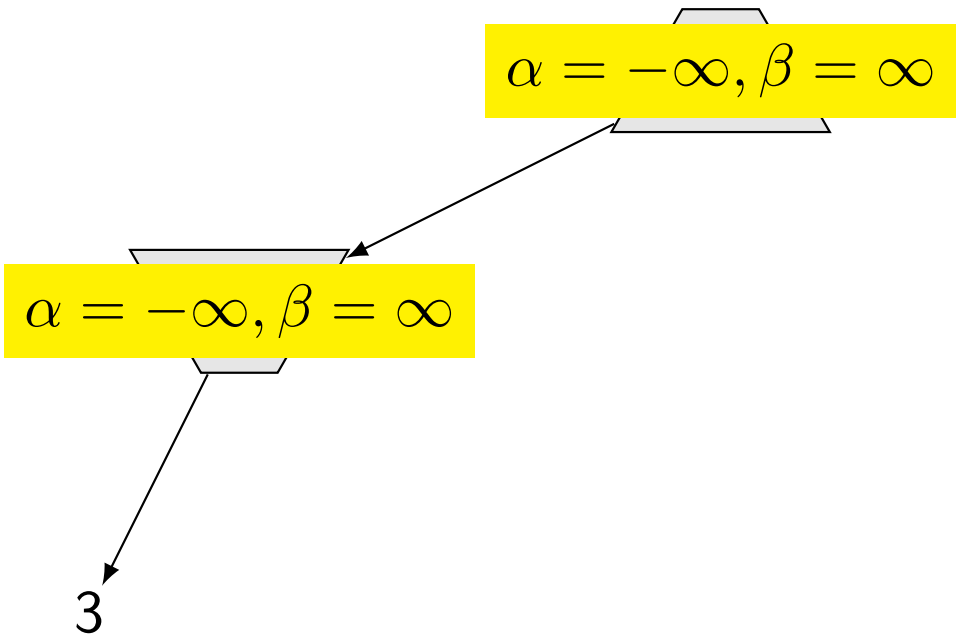
α highest (best) value choice found so far for any choice along MAX
β lowest (best) value choice found so far for any choice along MIN



In MIN-VAL: $v \leftarrow 2$
$v \leq \alpha$ then: return v!

α-β prunning

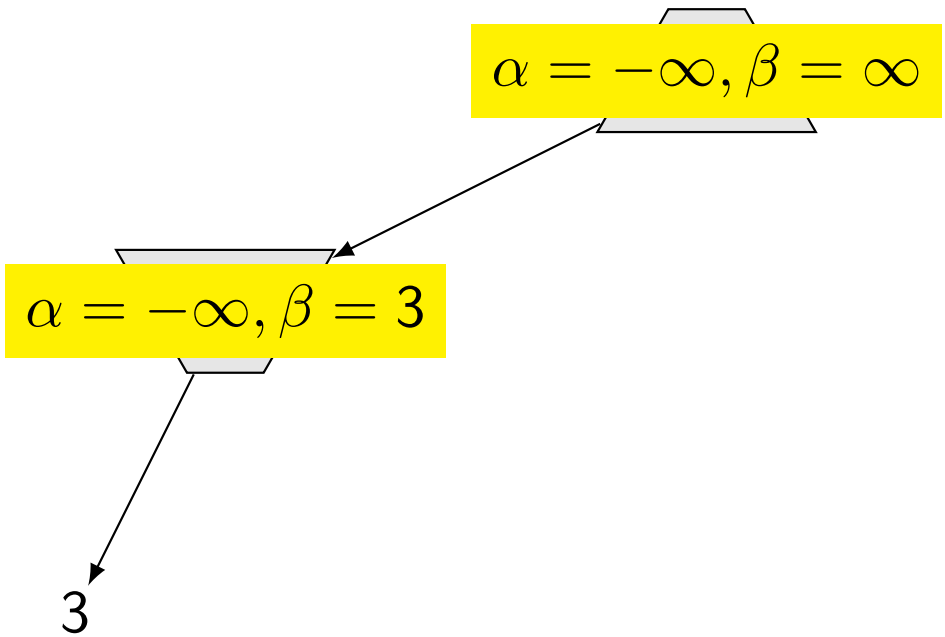
α highest (best) value choice found so far for any choice along MAX
β lowest (best) value choice found so far for any choice along MIN



In MIN-VAL: $v \leftarrow 2$
$v \leq \alpha$ then: return v!

α-β prunning

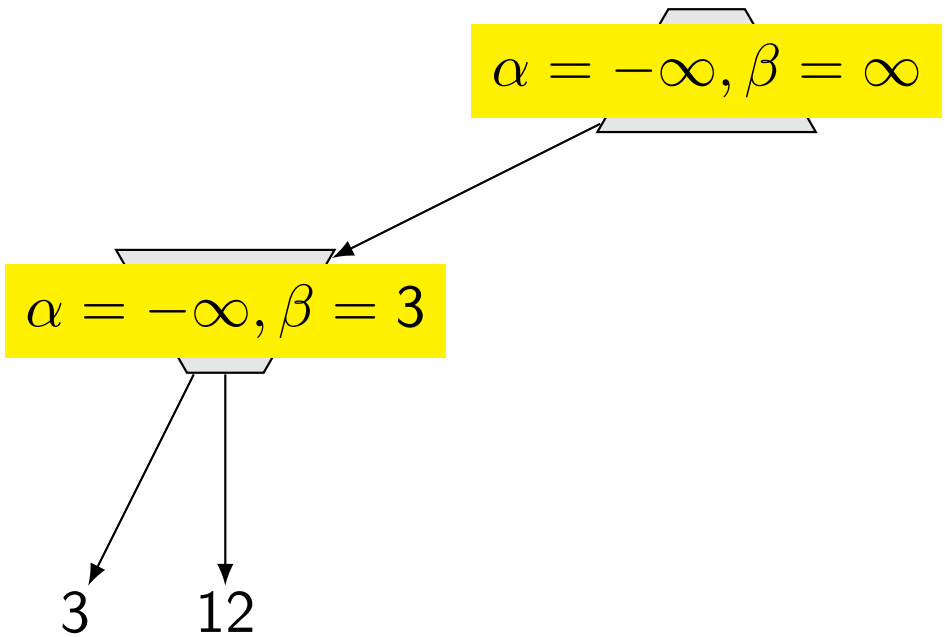
α highest (best) value choice found so far for any choice along MAX
β lowest (best) value choice found so far for any choice along MIN



In MIN-VAL: $v \leftarrow 2$
$v \leq \alpha$ then: return v!

α-β prunning

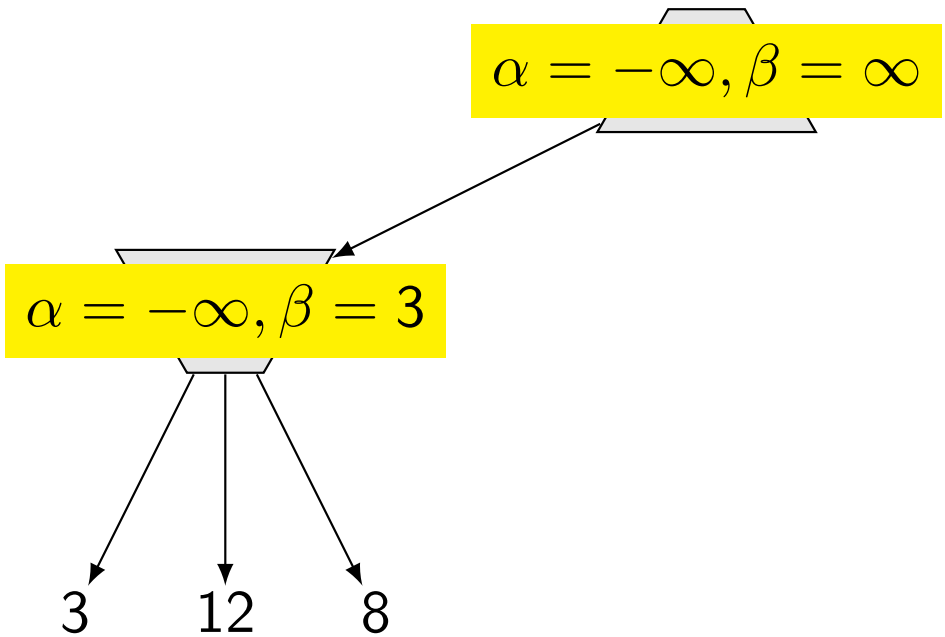
α highest (best) value choice found so far for any choice along MAX
β lowest (best) value choice found so far for any choice along MIN



In MIN-VAL: $v \leftarrow 2$
$v \leq \alpha$ then: return v!

α-β prunning

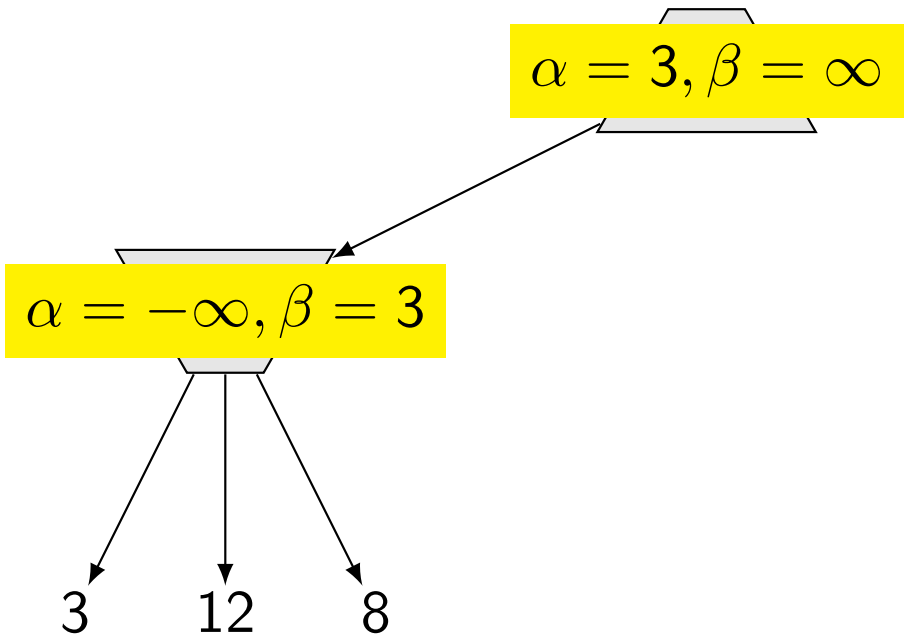
α highest (best) value choice found so far for any choice along MAX
β lowest (best) value choice found so far for any choice along MIN



In MIN-VAL: $v \leftarrow 2$
$v \leq \alpha$ then: return v!

α-β prunning

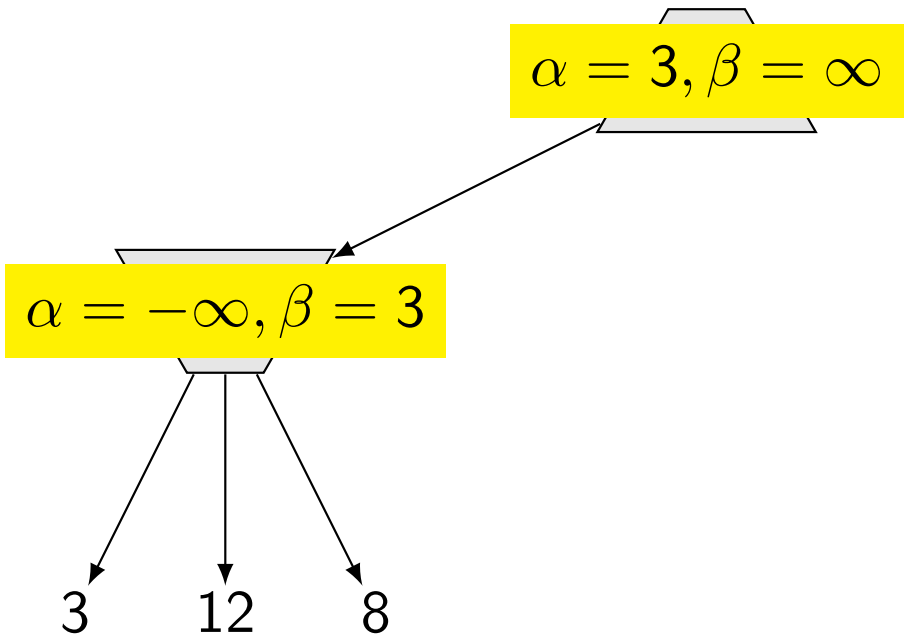
α highest (best) value choice found so far for any choice along MAX
β lowest (best) value choice found so far for any choice along MIN



In MIN-VAL: $v \leftarrow 2$
$v \leq \alpha$ then: return v!

α-β prunning

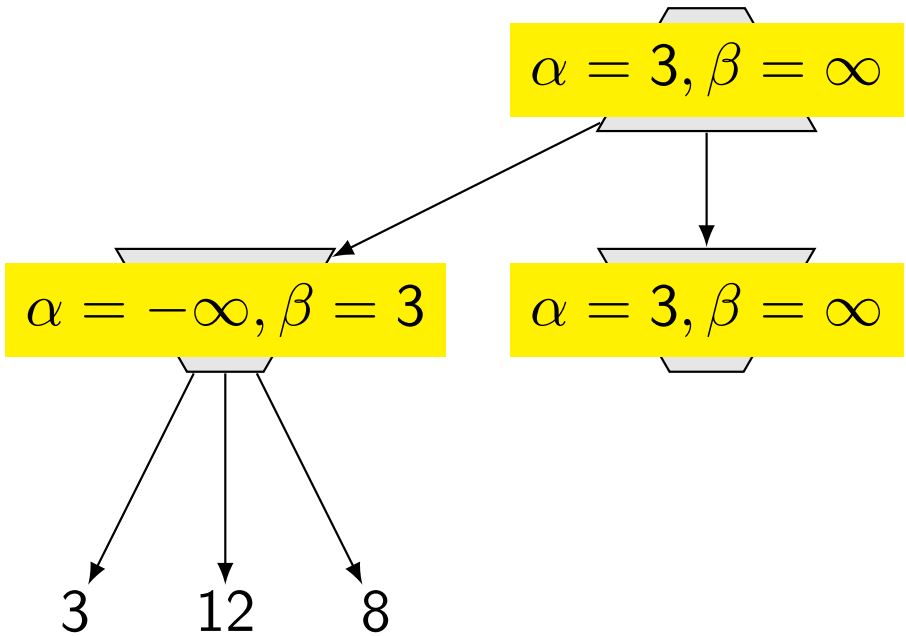
α highest (best) value choice found so far for any choice along MAX
β lowest (best) value choice found so far for any choice along MIN



In MIN-VAL: $v \leftarrow 2$
$v \leq \alpha$ then: return v!

α-β prunning

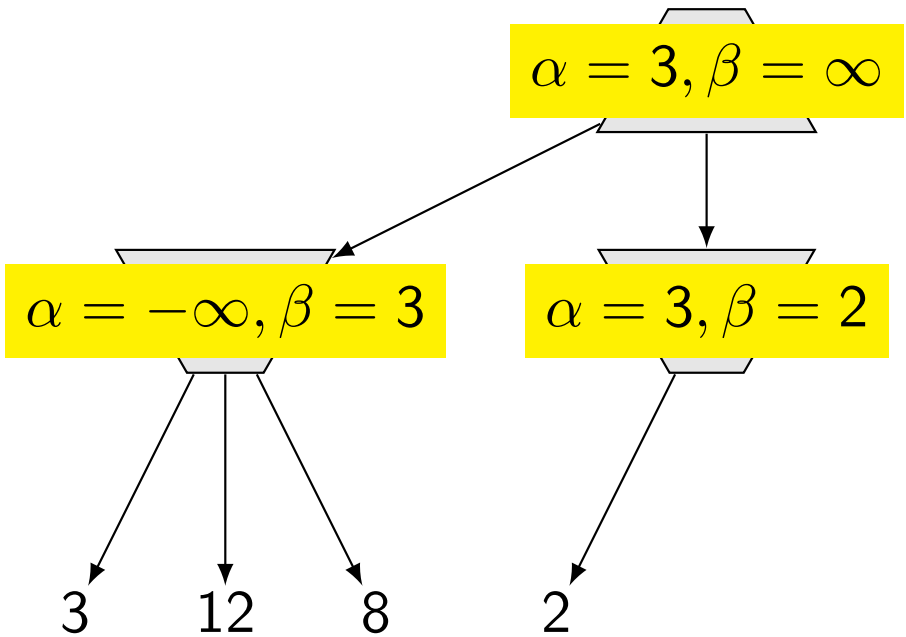
α highest (best) value choice found so far for any choice along MAX
β lowest (best) value choice found so far for any choice along MIN



In MIN-VAL: $v \leftarrow 2$
$v \leq \alpha$ then: return v!

α-β prunning

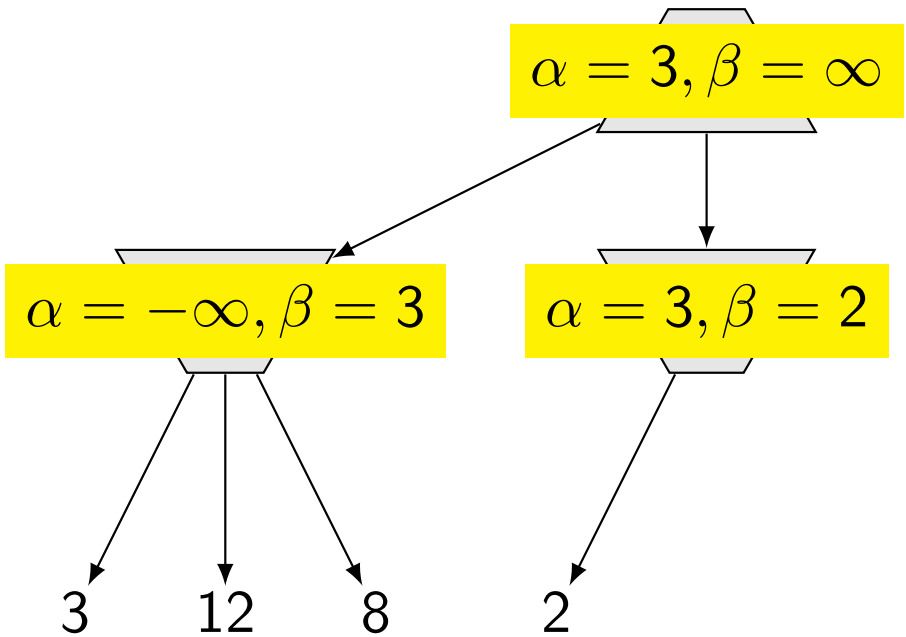
α highest (best) value choice found so far for any choice along MAX
β lowest (best) value choice found so far for any choice along MIN



In MIN-VAL: $v \leftarrow 2$
$v \leq \alpha$ then: return v!

α-β prunning

α highest (best) value choice found so far for any choice along MAX
β lowest (best) value choice found so far for any choice along MIN

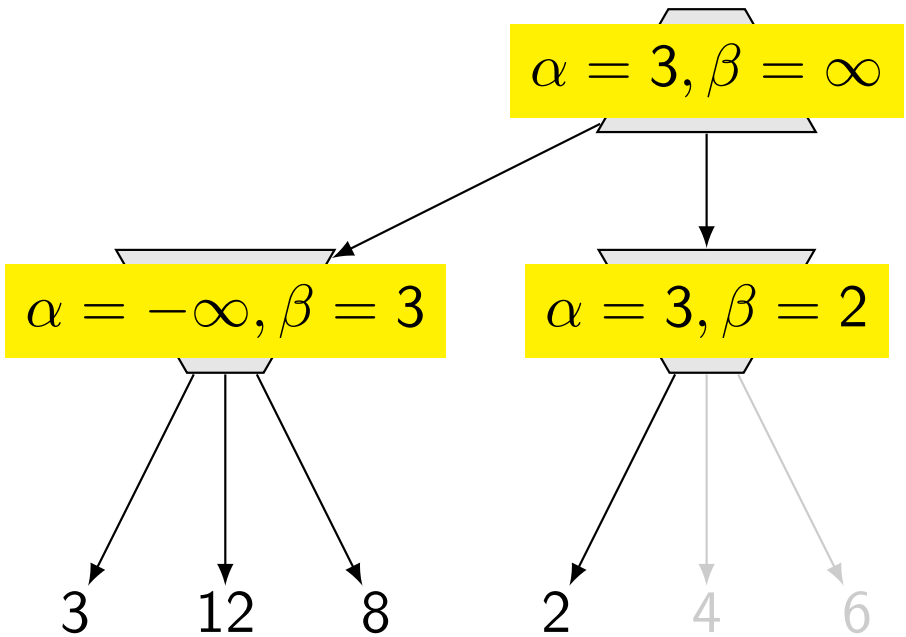


In MIN-VAL: $v \leftarrow 2$

$v \leq \alpha$ then: return v!

α-β prunning

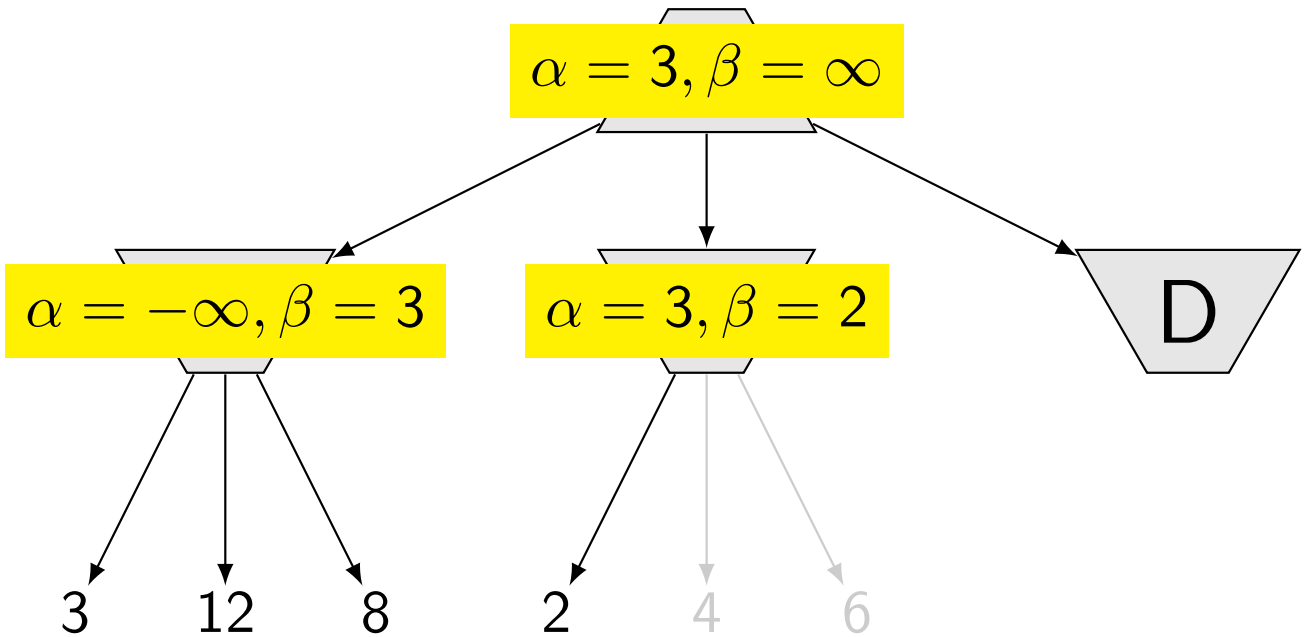
α highest (best) value choice found so far for any choice along MAX
β lowest (best) value choice found so far for any choice along MIN



In MIN-VAL: $v \leftarrow 2$
$v \leq \alpha$ then: return v!

α-β prunning

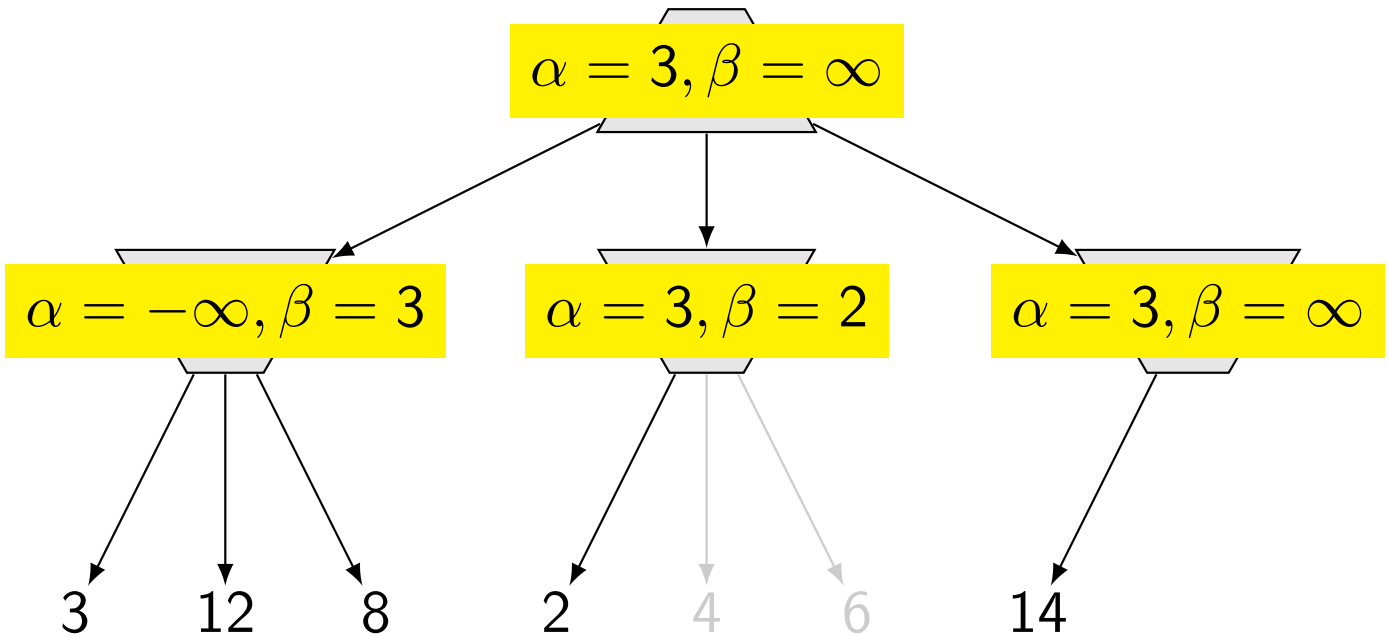
α highest (best) value choice found so far for any choice along MAX
β lowest (best) value choice found so far for any choice along MIN



In MIN-VAL: $v \leftarrow 2$
$v \leq \alpha$ then: return v!

α-β prunning

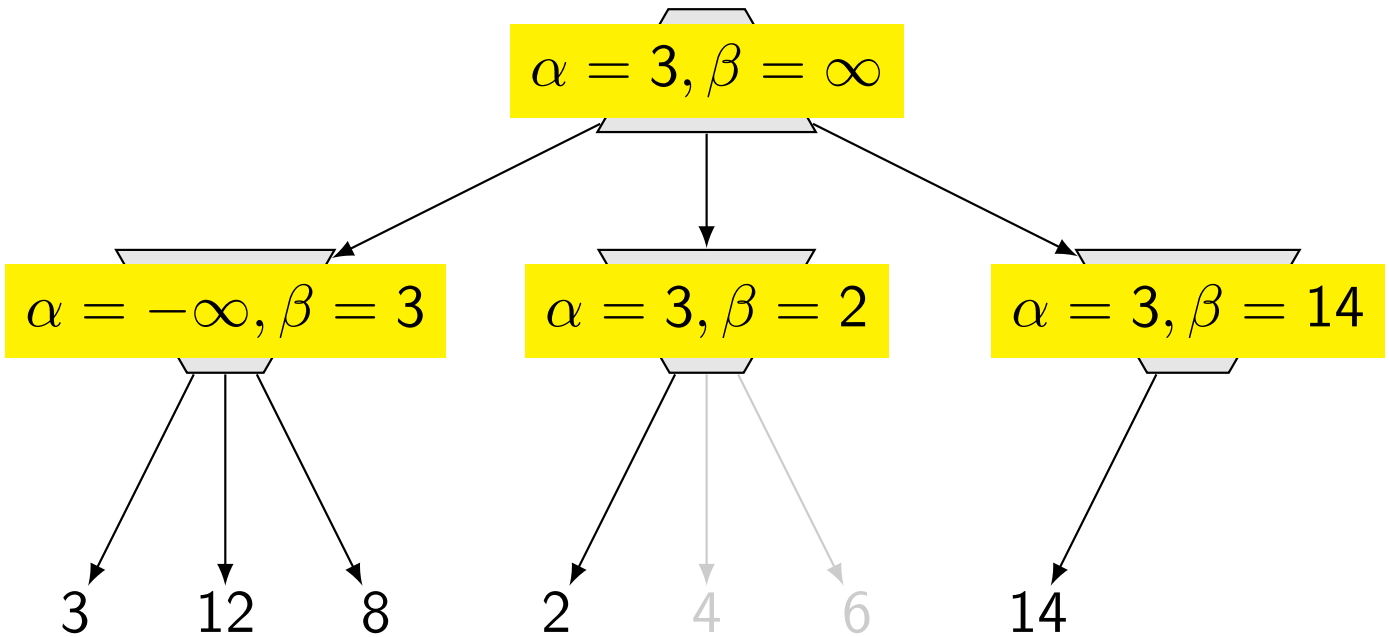
α highest (best) value choice found so far for any choice along MAX
β lowest (best) value choice found so far for any choice along MIN



In MIN-VAL: $v \leftarrow 2$
$v \leq \alpha$ then: return v!

α-β prunning

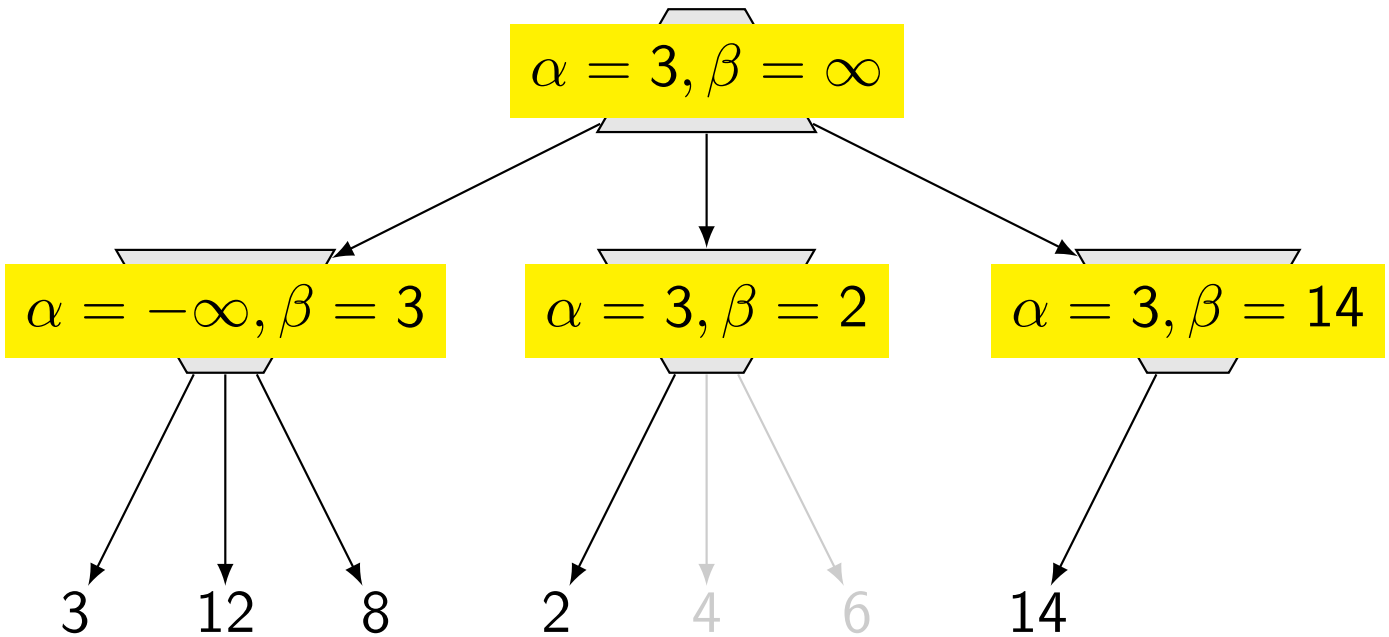
α highest (best) value choice found so far for any choice along MAX
β lowest (best) value choice found so far for any choice along MIN



In MIN-VAL: $v \leftarrow 2$
$v \leq \alpha$ then: return v!

α-β prunning

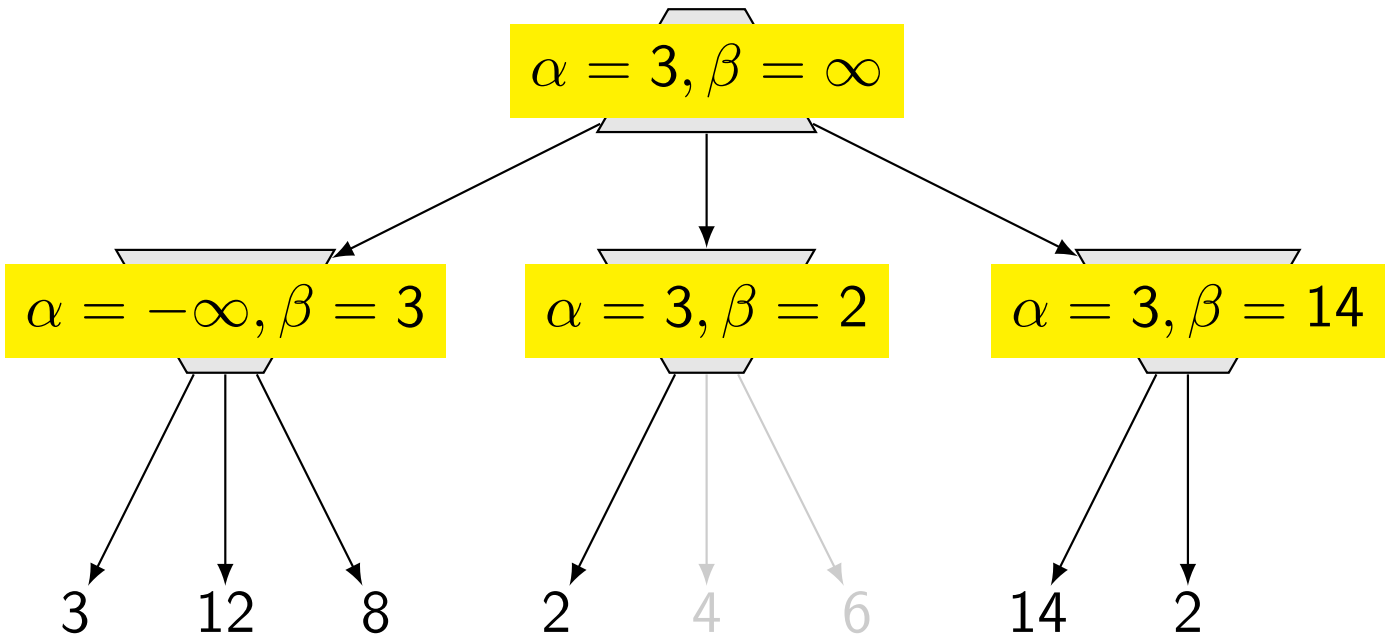
α highest (best) value choice found so far for any choice along MAX
β lowest (best) value choice found so far for any choice along MIN



In MIN-VAL: $v \leftarrow 2$
$v \leq \alpha$ then: return v!

α-β prunning

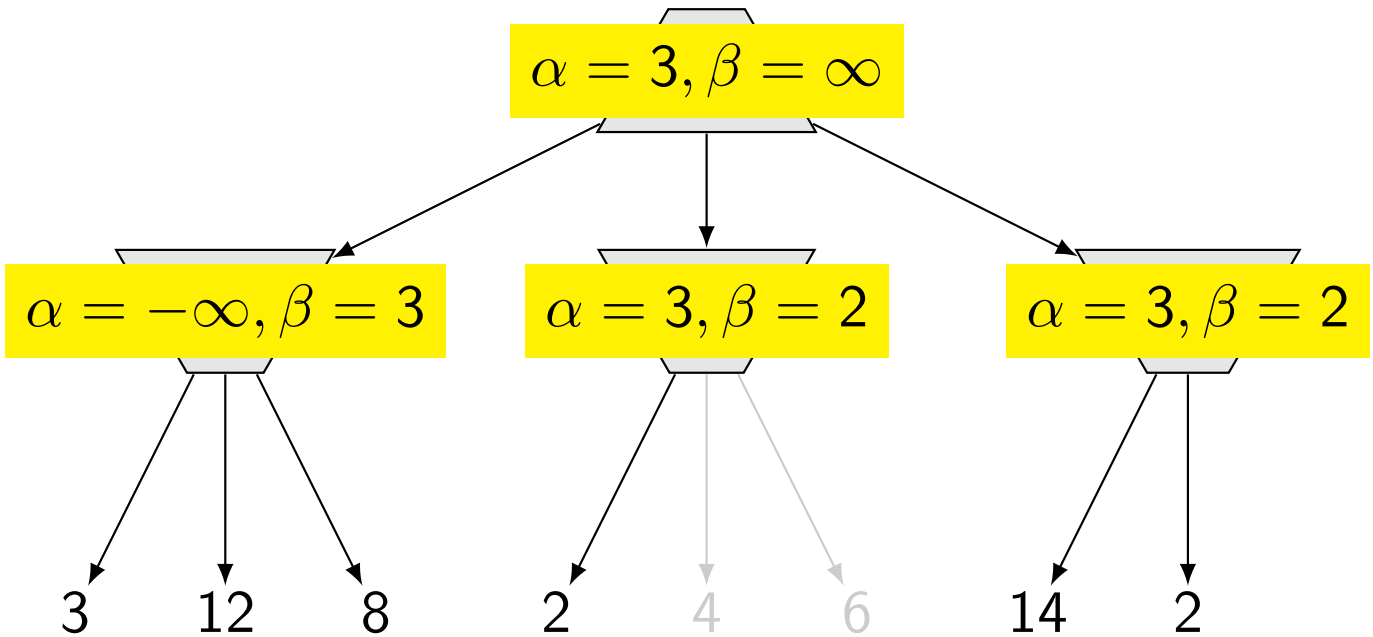
α highest (best) value choice found so far for any choice along MAX
β lowest (best) value choice found so far for any choice along MIN



In MIN-VAL: $v \leftarrow 2$
$v \leq \alpha$ then: return v!

α-β prunning

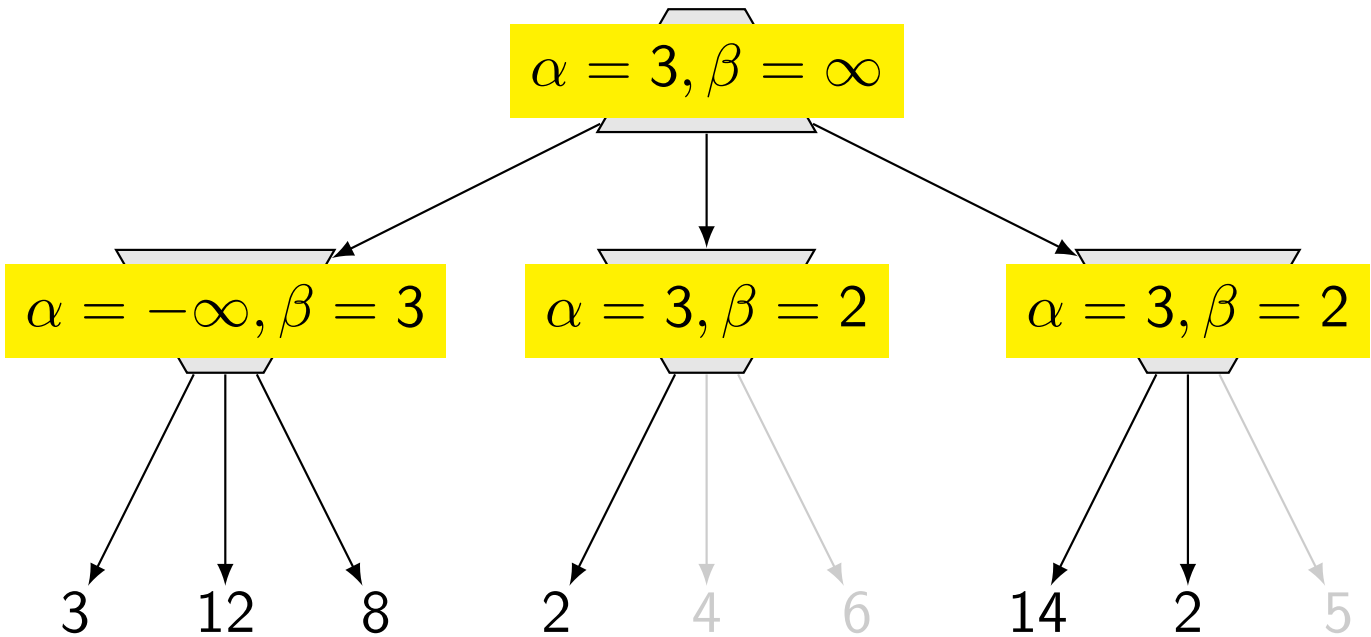
α highest (best) value choice found so far for any choice along MAX
β lowest (best) value choice found so far for any choice along MIN



In MIN-VAL: $v \leftarrow 2$
$v \leq \alpha$ then: return v!

α-β prunning

α highest (best) value choice found so far for any choice along MAX
β lowest (best) value choice found so far for any choice along MIN



In MIN-VAL: $v \leftarrow 2$
$v \leq \alpha$ then: return v!

function ALPHA-BETA-SEARCH(state) **returns** an action
  $v \leftarrow$ MAX-VALUE(state, $-\infty, \infty$)
  return the action in ACTIONS(state) with value v
end function

---

function MAX-VALUE(state, α, β) **returns** a utility value v
  if TERMINAL-TEST(state) **return** UTILITY(state)
  $v \leftarrow -\infty$
  for all ACTIONS(state) **do**
    $v \leftarrow$ **max**(v, MIN-VALUE(RESULT(state,a),α, β))
    if $v \geq \beta$ **return** v
    $\alpha \leftarrow \max(\alpha, v)$
  end for
end function

function MIN-VALUE(state, α, β) **returns** a utility value v
  if TERMINAL-TEST(state) **return** UTILITY(state)
  $v \leftarrow \infty$
  for all ACTIONS(state) **do**
    $v \leftarrow$ **min**(v, MAX-VALUE(RESULT(state,a),α, β))
    if $v \leq \alpha$ **return** v
    $\beta \leftarrow \min(\beta, v)$
  end for
end function

function ALPHA-BETA-SEARCH(state) **returns** an action
  $v \leftarrow$ MAX-VALUE(state, $-\infty, \infty$)
  return the action in ACTIONS(state) with value v
end function

---

function MAX-VALUE(state,α, β) **returns** a utility value v
  if TERMINAL-TEST(state) **return** UTILITY(state)
  $v \leftarrow -\infty$
  for all ACTIONS(state) **do**
    $v \leftarrow$ **max**(v, MIN-VALUE(RESULT(state,a),α, β))
    if $v \geq \beta$ **return** v
    $\alpha \leftarrow \max(\alpha, v)$
  end for
end function

---

function MIN-VALUE(state, α, β) **returns** a utility value v
  if TERMINAL-TEST(state) **return** UTILITY(state)
  $v \leftarrow \infty$
  for all ACTIONS(state) **do**
    $v \leftarrow$ **min**(v, MAX-VALUE(RESULT(state,a),α, β))
    if $v \leq \alpha$ **return** v
    $\beta \leftarrow \min(\beta, v)$
  end for
end function

function ALPHA-BETA-SEARCH(state) **returns** an action
  $v \leftarrow$ MAX-VALUE(state, $-\infty, \infty$)
  return the action in ACTIONS(state) with value v
end function

---

function MAX-VALUE(state,α, β) **returns** a utility value v
  if TERMINAL-TEST(state) **return** UTILITY(state)
  $v \leftarrow -\infty$
  for all ACTIONS(state) **do**
    $v \leftarrow$ **max**(v, MIN-VALUE(RESULT(state,a),α, β))
    if $v \geq \beta$ **return** v
    $\alpha \leftarrow \max(\alpha, v)$
  end for
end function

---

function MIN-VALUE(state, α, β) **returns** a utility value v
  if TERMINAL-TEST(state) **return** UTILITY(state)
  $v \leftarrow \infty$
  for all ACTIONS(state) **do**
    $v \leftarrow$ **min**(v, MAX-VALUE(RESULT(state,a),α, β))
    if $v \leq \alpha$ **return** v
    $\beta \leftarrow \min(\beta, v)$
  end for
end function

Imperfect but real-time decisions - iterative deepening

$$\text{H-MINIMAX}(s, d) =$$

$$\begin{cases} \text{EVAL}(s) & \text{if } \text{CUTOFF-TEST}(s, d) \\ \max_{a \in \text{ACTIONS}(s)} \text{H-MINIMAX}(\text{RESULT}(s, a), d+1) & \text{if } \text{PLAYER}(s) = \text{MAX} \\ \min_{a \in \text{ACTIONS}(s)} \text{H-MINIMAX}(\text{RESULT}(s, a, d+1)) & \text{if } \text{PLAYER}(s) = \text{MIN} \end{cases}$$

Imperfect but real-time decisions - iterative deepening

$$\text{H-MINIMAX}(s, d) =$$

$$\begin{cases} \text{EVAL}(s) & \text{if } \text{CUTOFF-TEST}(s, d) \\ \max_{a \in \text{ACTIONS}(s)} \text{H-MINIMAX}(\text{RESULT}(s, a), d+1) & \text{if } \text{PLAYER}(s) = \text{MAX} \\ \min_{a \in \text{ACTIONS}(s)} \text{H-MINIMAX}(\text{RESULT}(s, a, d+1)) & \text{if } \text{PLAYER}(s) = \text{MIN} \end{cases}$$

Imperfect but real-time decisions - iterative deepening

$$\text{H-MINIMAX}(s, d) = \begin{cases} \text{EVAL}(s) & \text{if } \text{CUTOFF-TEST}(s, d) \\ \max_{a \in \text{ACTIONS}(s)} \text{H-MINIMAX}(\text{RESULT}(s, a), d+1) & \text{if } \text{PLAYER}(s) = \text{MAX} \\ \min_{a \in \text{ACTIONS}(s)} \text{H-MINIMAX}(\text{RESULT}(s, a, d+1)) & \text{if } \text{PLAYER}(s) = \text{MIN} \end{cases}$$

Imperfect but real-time decisions - iterative deepening

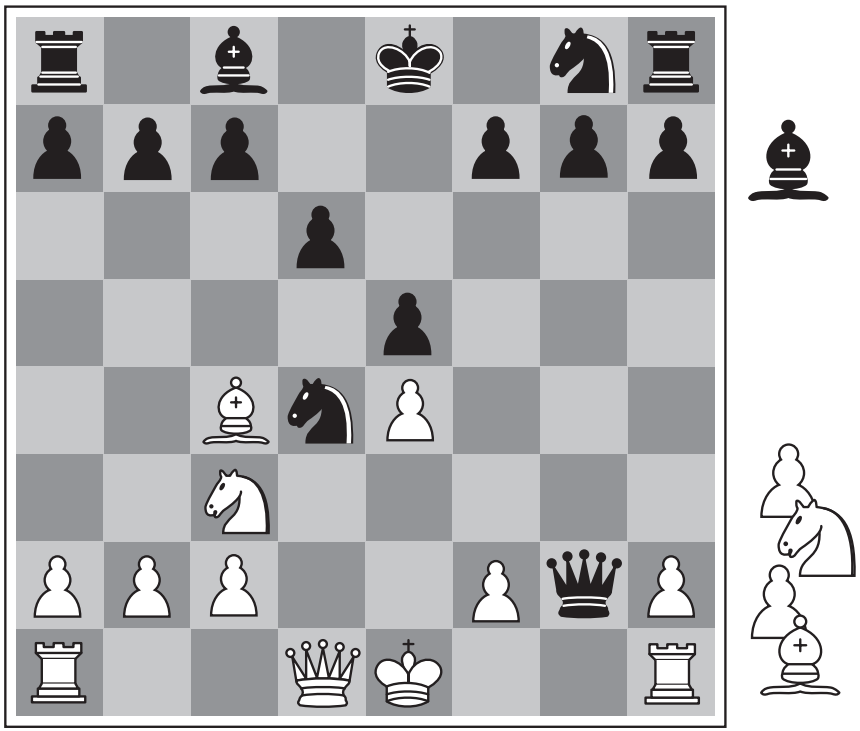
$$
\begin{aligned}
&\textsc{h-minimax}(s, d) = \\
&\qquad \textsc{eval}(s) \quad \text{if} \quad \textsc{cutoff-test}(s, d) \\
&\max_{a \in \textsc{actions}(s)} \textsc{h-minimax}(\textsc{result}(s, a), d + 1) \quad \text{if} \quad \textsc{player}(s) = \textsc{max} \\
&\min_{a \in \textsc{actions}(s)} \textsc{h-minimax}(\textsc{result}(s, a, d + 1)) \quad \text{if} \quad \textsc{player}(s) = \textsc{min}
\end{aligned}
$$

Cutting off search

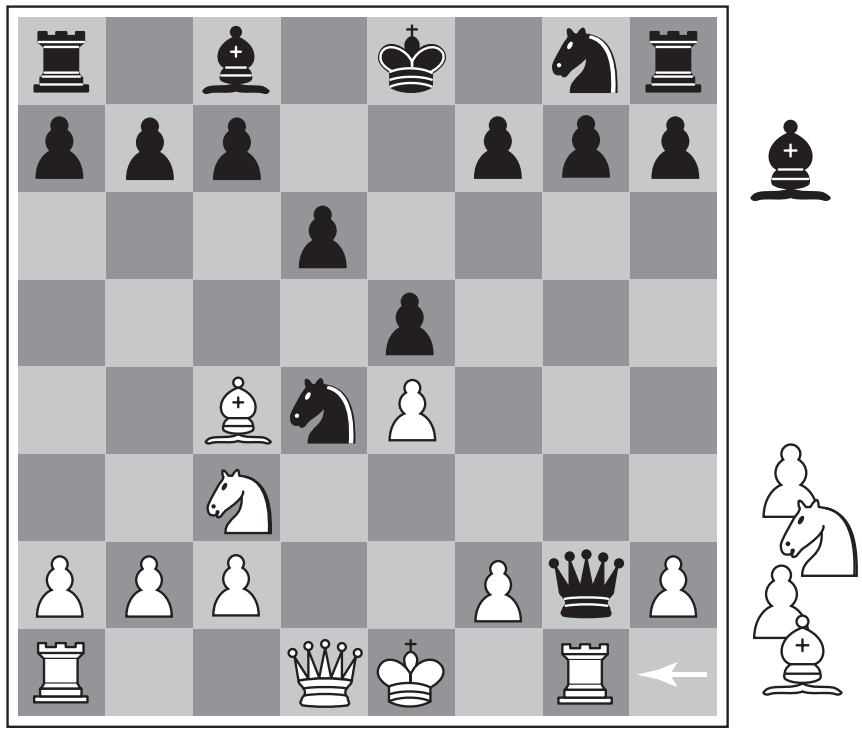
Replace
if TERMINAL-TEST(s) **then return** UTILITY(s)
with:
if CUTOFF-TEST(s,d) **then return** EVAL(s)

EVAL(s) – Evaluation functions

(estimate of) State value for non-terminal states



(a) White to move

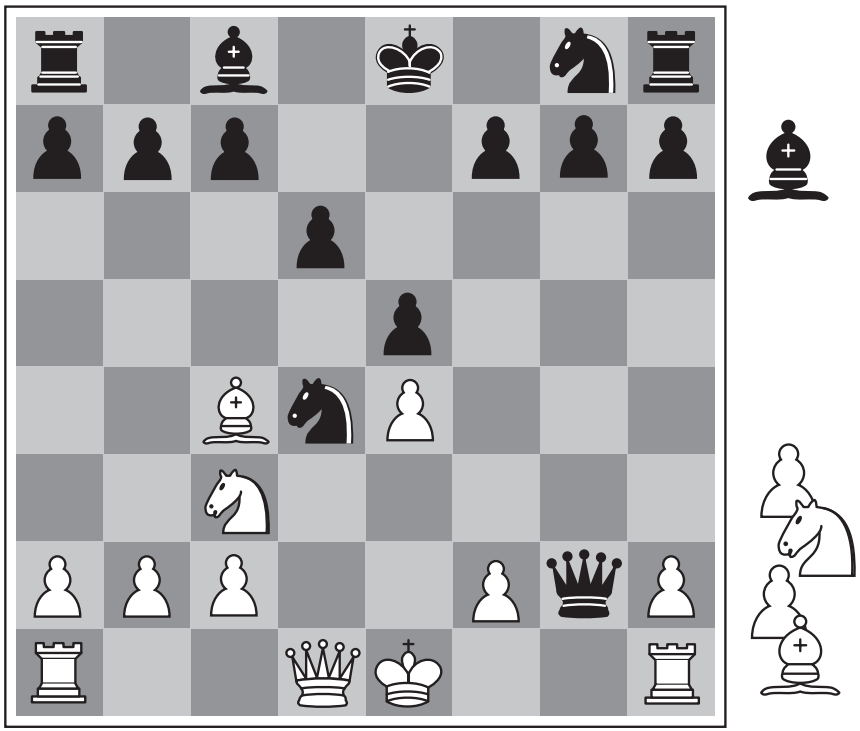


(b) White to move

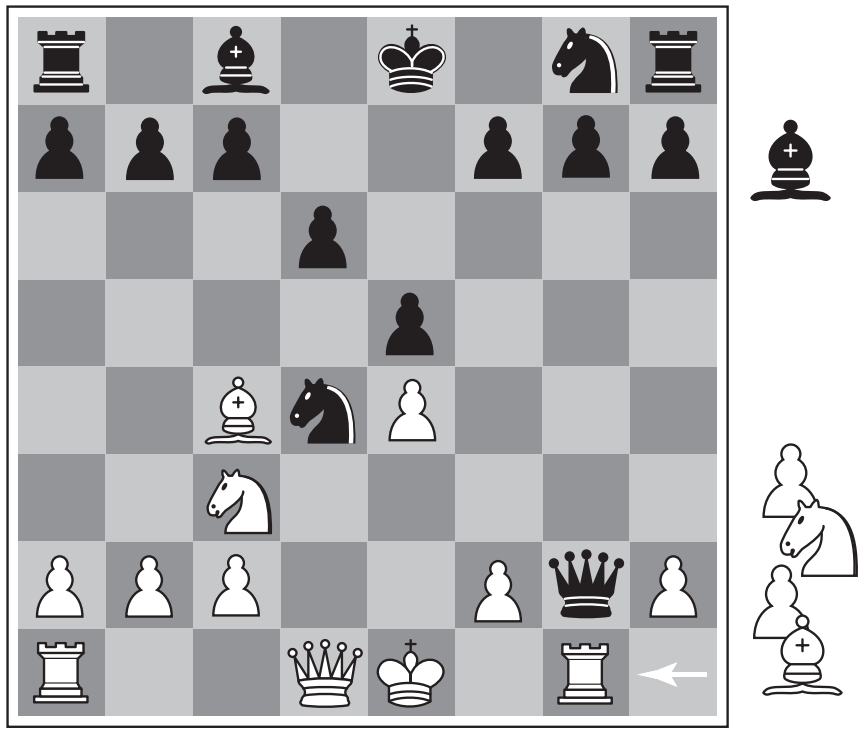
$$\text{EVAL}(s) = w_1 f_1(s) + w_2 f_2(s) + \cdots w_n f_n(s)$$

EVAL(s) – Evaluation functions

(estimate of) State value for non-terminal states



(a) White to move



(b) White to move

$$\textsc{Eval}(s) = w_1 f_1(s) + w_2 f_2(s) + \cdots w_n f_n(s)$$

References

[1] Stuart Russell and Peter Norvig.
Artificial Intelligence: A Modern Approach.
Prentice Hall, 3rd edition, 2010.
http://aima.cs.berkeley.edu/.

[2] Richard S. Sutton and Andrew G. Barto.
Reinforcement Learning; an Introduction.
MIT Press, 2nd edition, 2018.
http://www.incompleteideas.net/book/the-book-2nd.html.