Adversarial Search

Tomáš Svoboda, Petr Pošík, Matěj Hoffmann

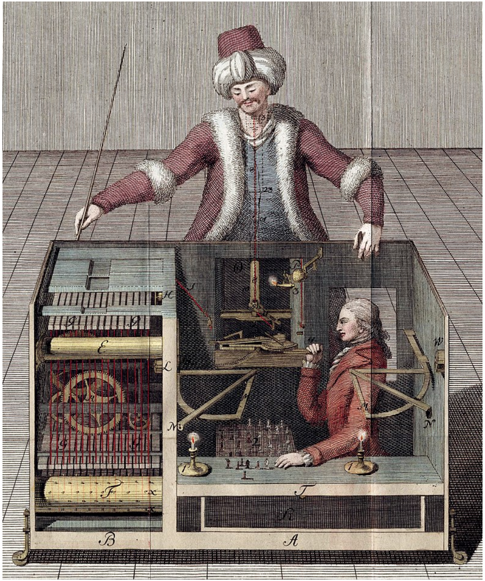
Vision for Robots and Autonomous Systems, Center for Machine Perception
Department of Cybernetics
Faculty of Electrical Engineering, Czech Technical University in Prague

March 4, 2024

Games, man vs. algorithm

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

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

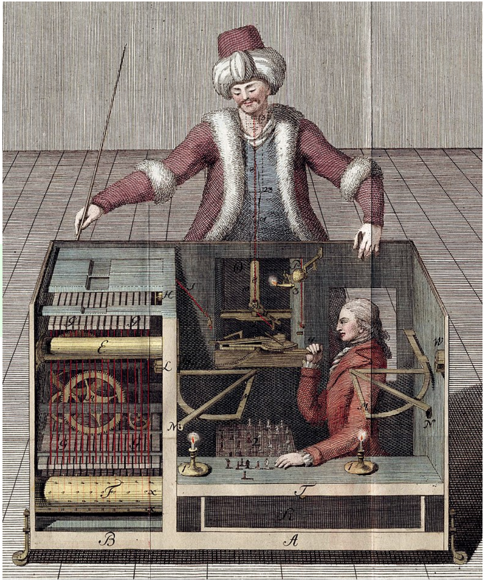


https://en.wikipedia.org/wiki/Mechanical_Turk

Games, man vs. algorithm

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

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



https://en.wikipedia.org/wiki/Mechanical_Turk

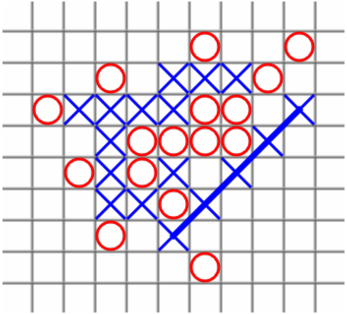
More: Adversarial Learning

Video: Adversing visual segmentation

Vision for Robotics and Autonomous Systems, http://cyber.felk.cvut.cz/vras, video at YT: https://youtu.be/KvdZmtVguOo

Elements of the game

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

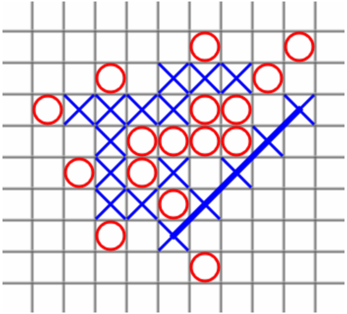


https://commons.wikimedia.org/wiki/File:Tic-tac-toe_5.png

Think about what do the functions return?

Elements of the game

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



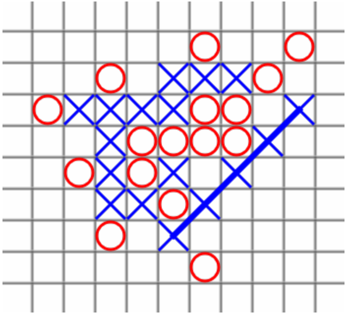
https://commons.wikimedia.org/wiki/File:Tic-tac-toe_5.png

Think about what do the functions return?

Elements of the game

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

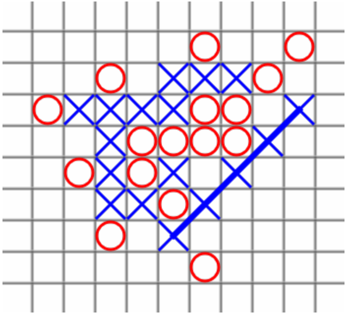
Think about what do the functions return?



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

Elements of the game

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

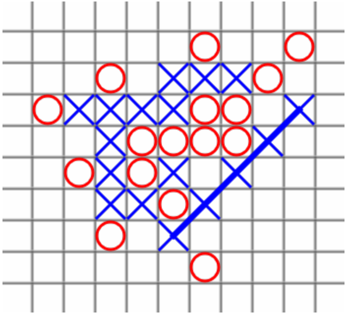


https://commons.wikimedia.org/wiki/File:Tic-tac-toe_5.png

Think about what do the functions return?

Elements of the game

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

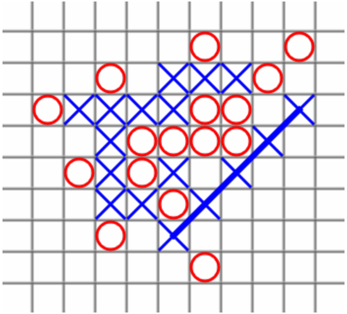


https://commons.wikimedia.org/wiki/File:Tic-tac-toe_5.png

Think about what do the functions return?

Elements of the game

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



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

Think about what do the functions return?

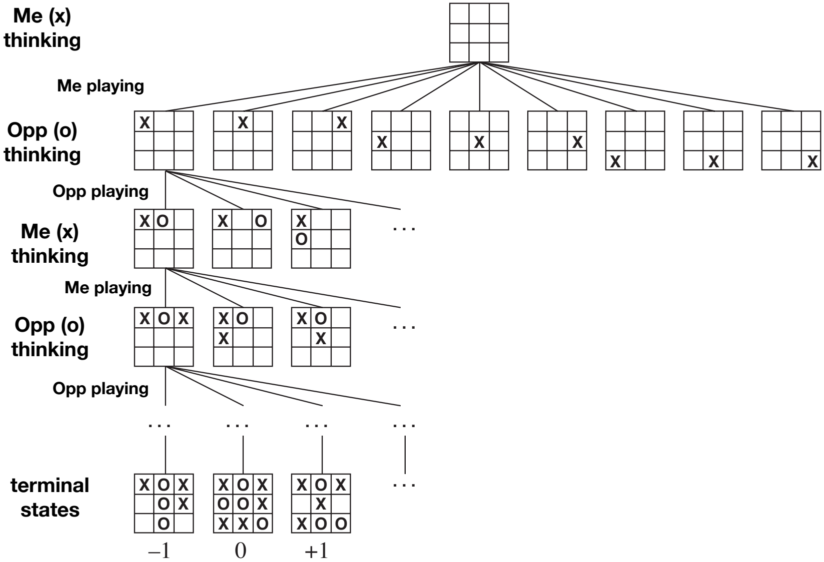
Terminal utilitity: Zero–Sum and General games

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

Game Tree(s)



$\textsc{Terminal-Utility}(s, \mathbf{x})$

State Value $V(s)$

$V(s)$ – value V of a state s : The best utility achievable from state s, assuming optimal actions from s':

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

For games, it (notion of the best) also depends on player p (assuming both players play optimally from s'):

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

State Value $V(s)$

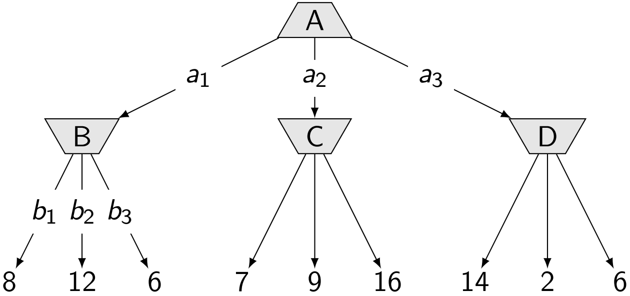
$V(s)$ – value V of a state s : The best utility achievable from state s, assuming optimal actions from s':

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

For games, it (notion of the best) also depends on player p (assuming both players play optimally from s'):

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

What is the Value of the root $V(A)$?



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

A, B, C, D - states of the game. I start, values represent values of terminal states, more is better for me - think about the (my) money prize. Assume (strictly) rational players.

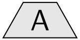
A: $V(A) = 6$

B: $V(A) = 2$

C: $V(A) = 7$

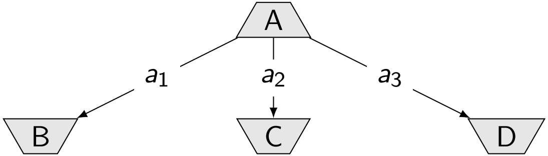
D: $V(A) = 16$

Two-ply game: **max** for me, **min** for the opponent. What is the best action a?



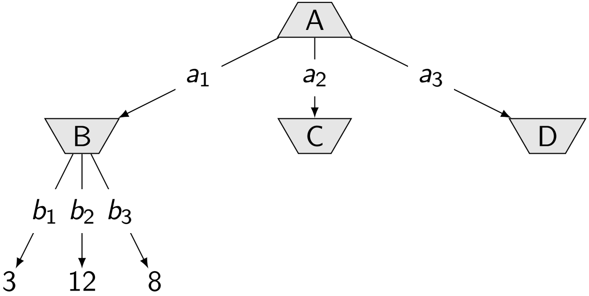
$$a^* = \underset{a \in \textsc{Actions}(\text{state}=A)}{\arg\max} V(\textsc{Result}(\text{state} = A, a))$$

Two-ply game: **max** for me, **min** for the opponent. What is the best action a?



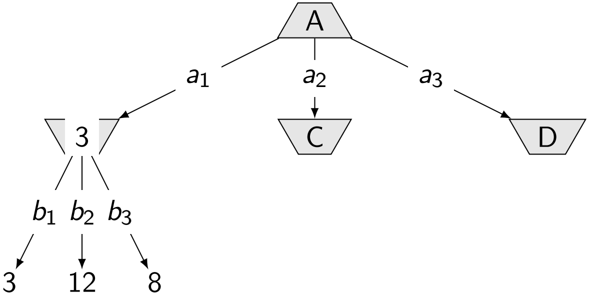
$$a^* = \arg\max_{a \in \textsc{Actions}(\text{state}=A)} V(\textsc{Result}(\text{state}=A, a))$$

Two-ply game: **max** for me, **min** for the opponent. What is the best action a?



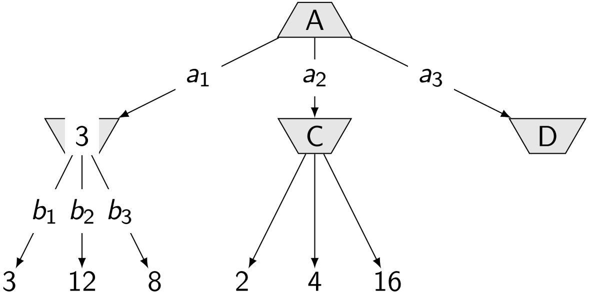
$$a^* = \arg\max_{a \in \textsc{Actions}(\text{state}=A)} V(\textsc{Result}(\text{state} = A, a))$$

Two-ply game: **max** for me, **min** for the opponent. What is the best action a?



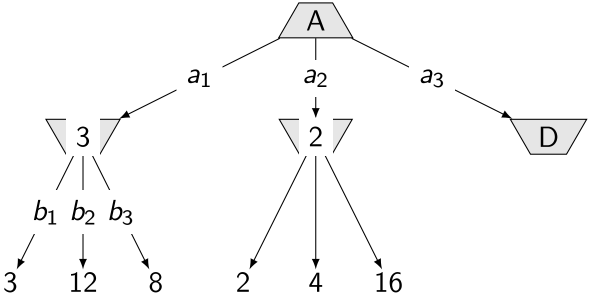
$$a^* = \underset{a \in \text{Actions}(\text{state}=A)}{\arg\max} V(\text{Result}(\text{state}=A, a))$$

Two-ply game: **max** for me, **min** for the opponent. What is the best action a?



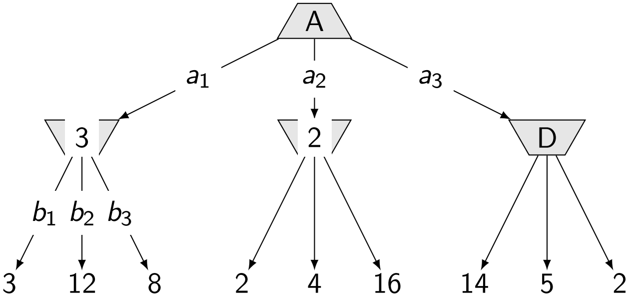
$$a^* = \arg\max_{a \in \text{ACTIONS}(\text{state}=A)} V(\text{RESULT}(\text{state} = A, a))$$

Two-ply game: **max** for me, **min** for the opponent. What is the best action *a*?



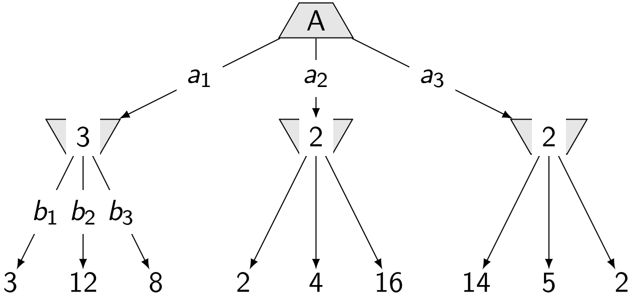
$$a^* = \underset{a \in \textsc{Actions}(\text{state}=A)}{\arg\max} V(\textsc{Result}(\text{state} = A, a))$$

Two-ply game: **max** for me, **min** for the opponent. What is the best action a?



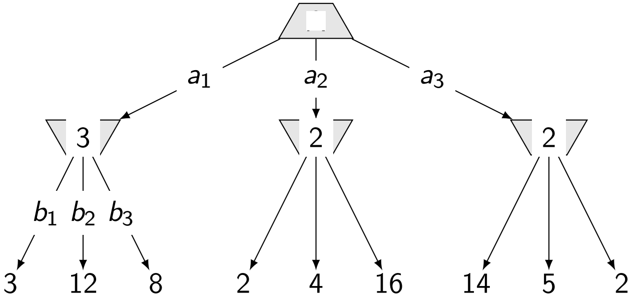
$$a^* = \underset{a \in \textsc{Actions}(\text{state}=A)}{\arg\max} V(\textsc{Result}(\text{state} = A, a))$$

Two-ply game: **max** for me, **min** for the opponent. What is the best action a?



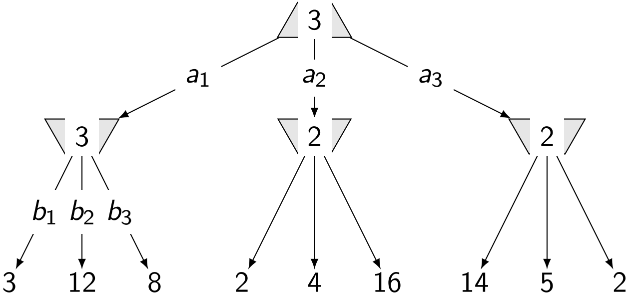
$$a^* = \underset{a \in \textsc{Actions}(\text{state}=A)}{\arg\max} V(\textsc{Result}(\text{state} = A, a))$$

Two-ply game: **max** for me, **min** for the opponent. What is the best action a?



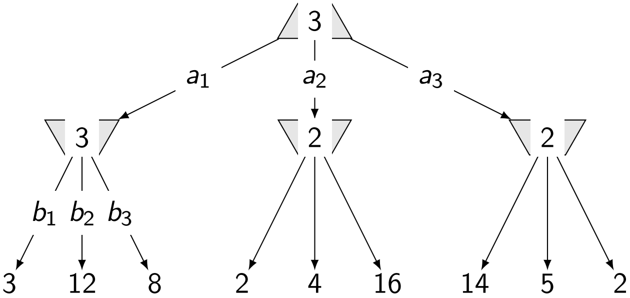
$$a^* = \arg\max_{a \in \text{ACTIONS}(\text{state}=A)} V(\text{RESULT}(\text{state} = A, a))$$

Two-ply game: **max** for me, **min** for the opponent. What is the best action a?



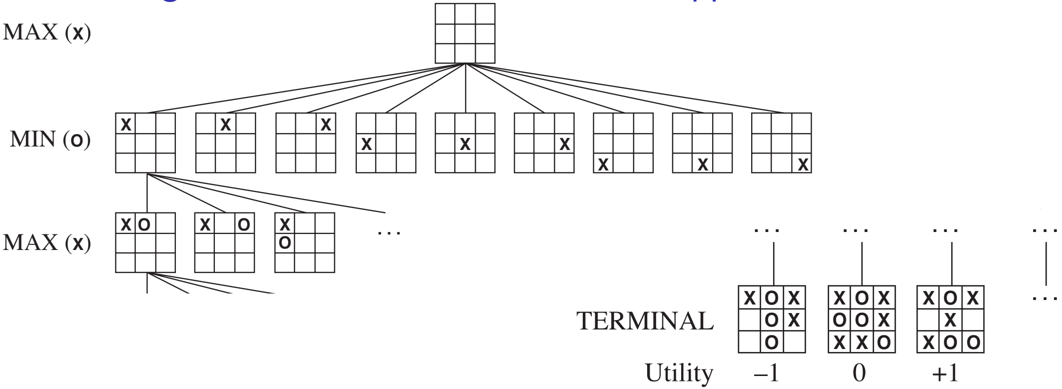
$$a^* = \underset{a \in \textsc{Actions}(\text{state}=A)}{\arg\max} V(\textsc{Result}(\text{state} = A, a))$$

Two-ply game: **max** for me, **min** for the opponent. What is the best action a?



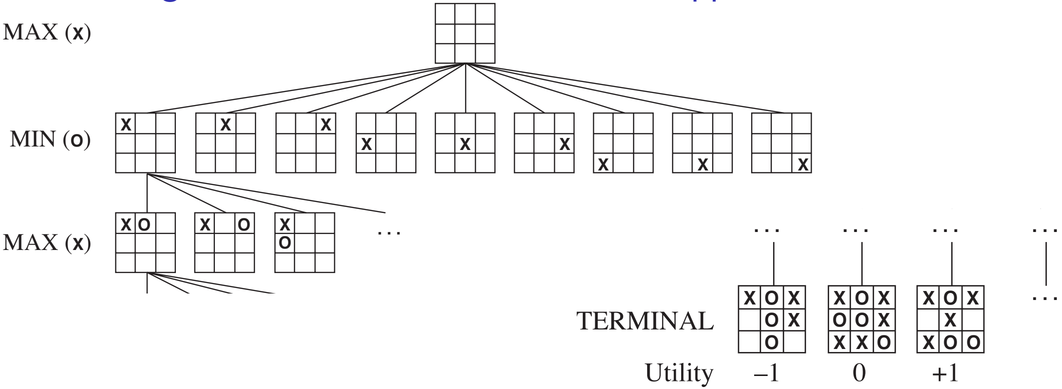
$$a^* = \arg\max_{a \in \textsc{Actions}(\text{state}=A)} V(\textsc{Result}(\text{state} = A, a))$$

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



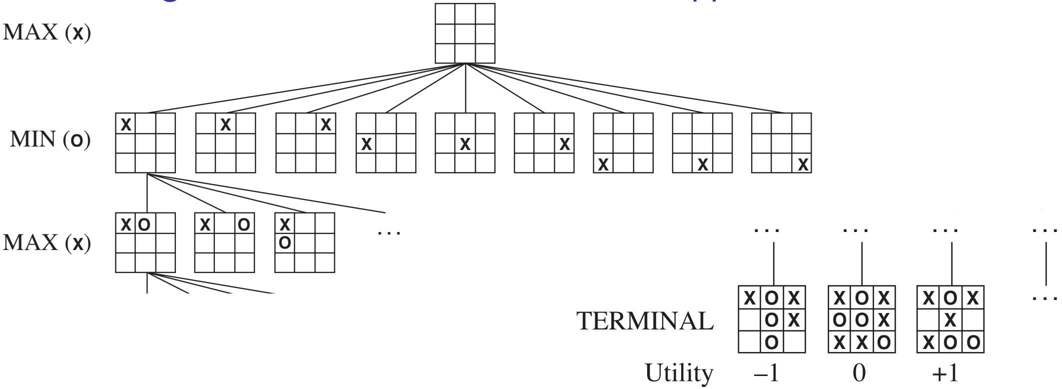
$$\text{MINIMAX}(s) = \begin{cases} \text{UTILITY}(s, \text{MAX}) & \text{if } \text{IS-TERMINAL}(s) \\ \max_{a \in \text{ACTIONS}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if } \text{TO-PLAY}(s) = \text{MAX} \\ \min_{a \in \text{ACTIONS}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if } \text{TO-PLAY}(s)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

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



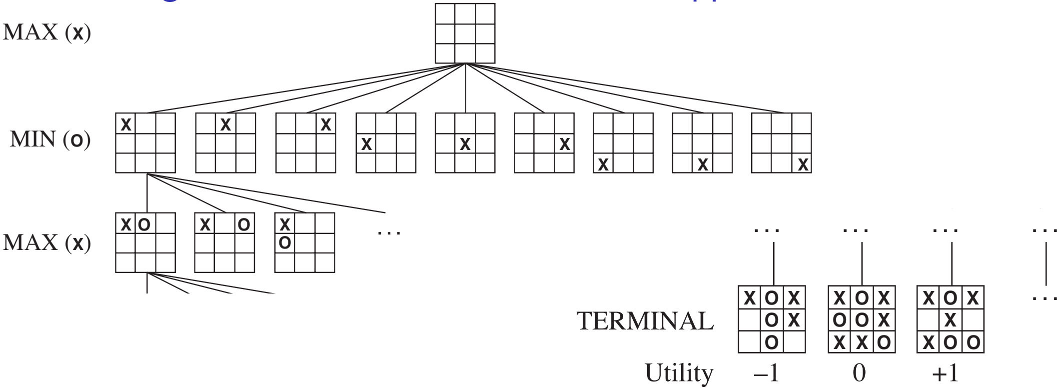
$$\text{MINIMAX}(s) = \begin{cases} \text{UTILITY}(s, \text{MAX}) & \text{if} \quad \text{IS-TERMINAL}(s) \\ \max_{a \in \text{ACTIONS}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if} \quad \text{TO-PLAY}(s) = \text{MAX} \\ \min_{a \in \text{ACTIONS}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if}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

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



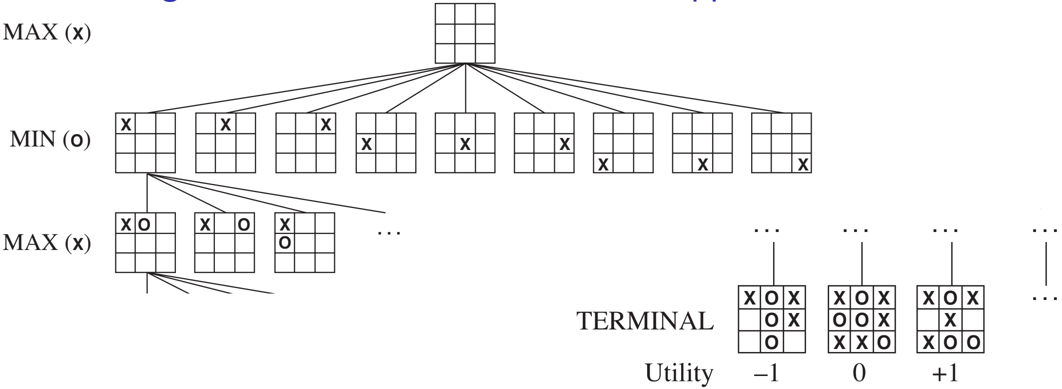
$$\text{MINIMAX}(s) = \begin{cases} \text{UTILITY}(s, \text{MAX}) & \text{if } \text{IS-TERMINAL}(s) \\ \max_{a \in \text{ACTIONS}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if } \text{TO-PLAY}(s) = \text{MAX} \\ \min_{a \in \text{ACTIONS}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if } \text{TO-PLAY}(s)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

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



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

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



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

Minimax algorithm

function MINIMAX(state) **returns** an action
    return $\operatorname{argmax}_{a \in \text{Actions}(s)}$ MIN-VALUE(RESULT(state, a))

---

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

---

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

Minimax algorithm

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

---

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

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

Minimax algorithm

function MINIMAX(state) **returns** an action
  return $\mathrm{argmax}_{a \in \mathrm{Actions}(s)}$ MIN-VALUE(RESULT(state, a))

---

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

---

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

Minimax algorithm

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

---

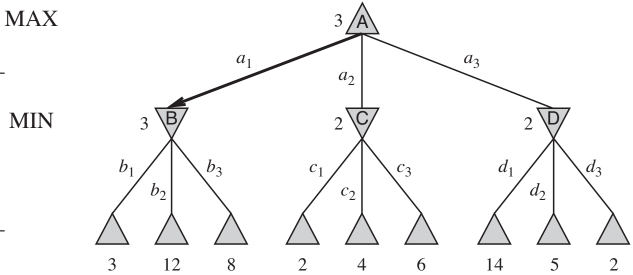
function MIN-VALUE(state) **returns** a utility value v
    if TERMINAL-TEST(state) **then return** UTILITY(state)
    $v \leftarrow \infty$
    for all $a \in$ ACTIONS(state) **do**
        $v \leftarrow$ **min**(v, MAX-VALUE(RESULT(state,a)))

---

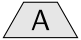
function MAX-VALUE(state) **returns** a utility value v
    if TERMINAL-TEST(state) **then return** UTILITY(state)
    $v \leftarrow -\infty$
    for all $a \in$ ACTIONS(state) **do**
        $v \leftarrow$ **max**(v, MIN-VALUE(RESULT(state,a)))

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

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



A two ply game, recursive run

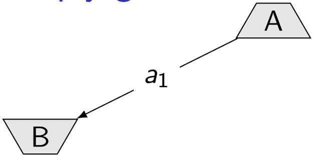


Is it like DFS or BFS?

What is the complexity? How many nodes to visit?

Can we do better? How?

A two ply game, recursive run

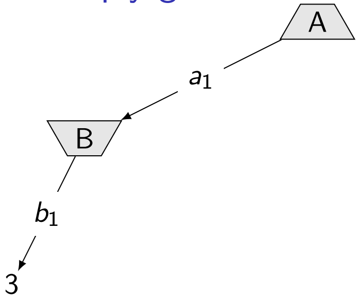


Is it like DFS or BFS?

What is the complexity? How many nodes to visit?

Can we do better? How?

A two ply game, recursive run

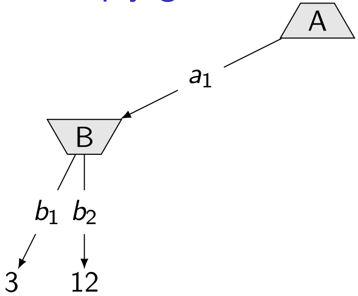


Is it like DFS or BFS?

What is the complexity? How many nodes to visit?

Can we do better? How?

A two ply game, recursive run

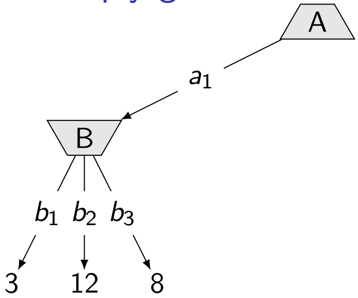


Is it like DFS or BFS?

What is the complexity? How many nodes to visit?

Can we do better? How?

A two ply game, recursive run

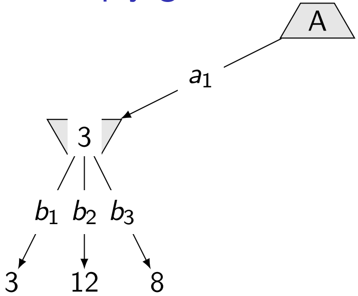


Is it like DFS or BFS?

What is the complexity? How many nodes to visit?

Can we do better? How?

A two ply game, recursive run

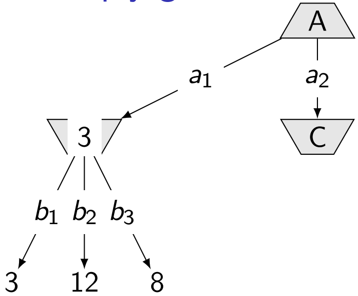


Is it like DFS or BFS?

What is the complexity? How many nodes to visit?

Can we do better? How?

A two ply game, recursive run

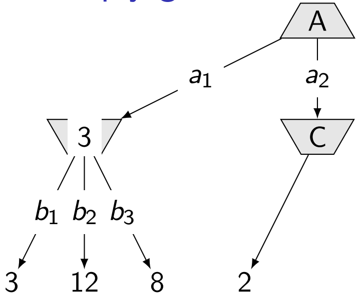


Is it like DFS or BFS?

What is the complexity? How many nodes to visit?

Can we do better? How?

A two ply game, recursive run

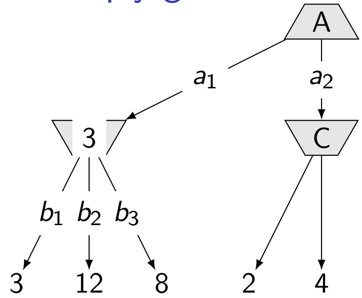


Is it like DFS or BFS?

What is the complexity? How many nodes to visit?

Can we do better? How?

A two ply game, recursive run

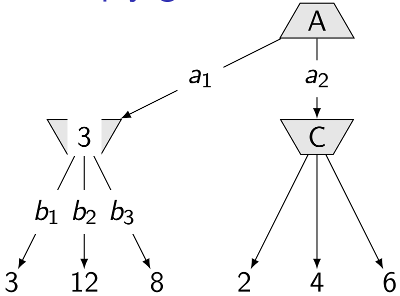


Is it like DFS or BFS?

What is the complexity? How many nodes to visit?

Can we do better? How?

A two ply game, recursive run

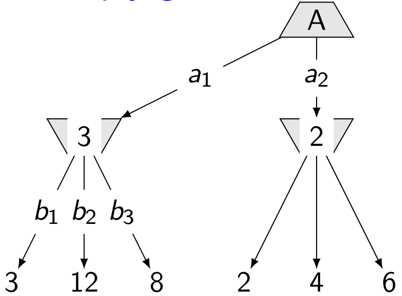


Is it like DFS or BFS?

What is the complexity? How many nodes to visit?

Can we do better? How?

A two ply game, recursive run

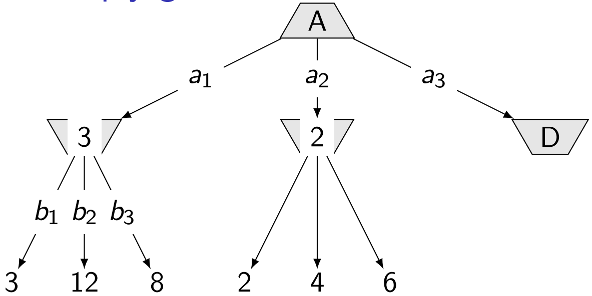


Is it like DFS or BFS?

What is the complexity? How many nodes to visit?

Can we do better? How?

A two ply game, recursive run

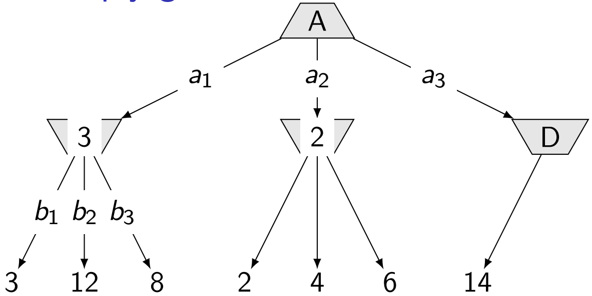


Is it like DFS or BFS?

What is the complexity? How many nodes to visit?

Can we do better? How?

A two ply game, recursive run

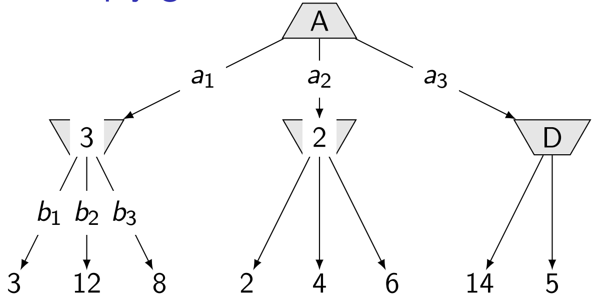


Is it like DFS or BFS?

What is the complexity? How many nodes to visit?

Can we do better? How?

A two ply game, recursive run

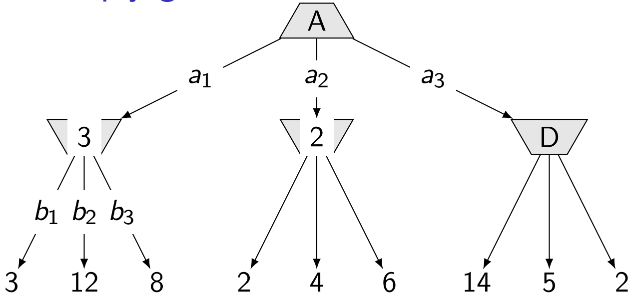


Is it like DFS or BFS?

What is the complexity? How many nodes to visit?

Can we do better? How?

A two ply game, recursive run

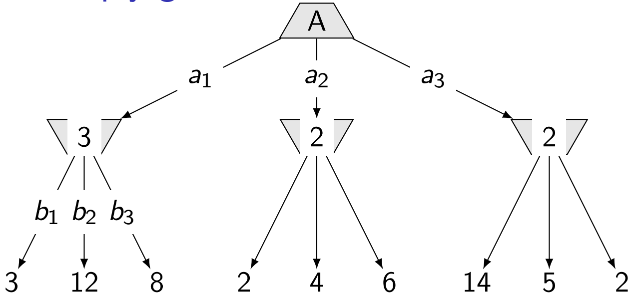


Is it like DFS or BFS?

What is the complexity? How many nodes to visit?

Can we do better? How?

A two ply game, recursive run

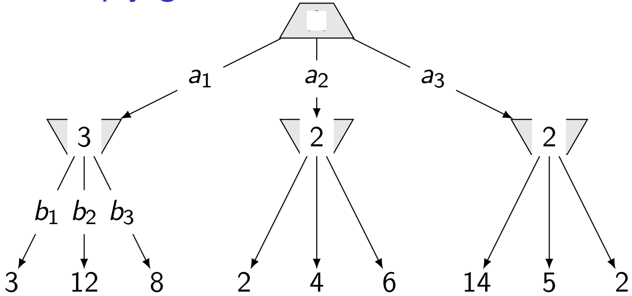


Is it like DFS or BFS?

What is the complexity? How many nodes to visit?

Can we do better? How?

A two ply game, recursive run

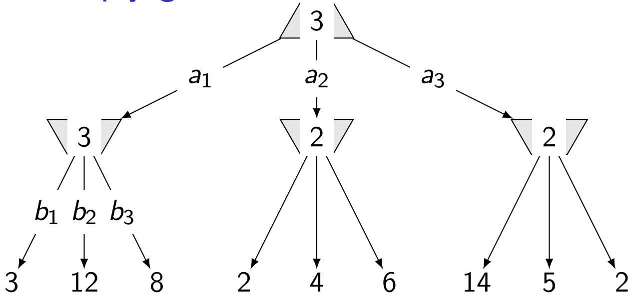


Is it like DFS or BFS?

What is the complexity? How many nodes to visit?

Can we do better? How?

A two ply game, recursive run

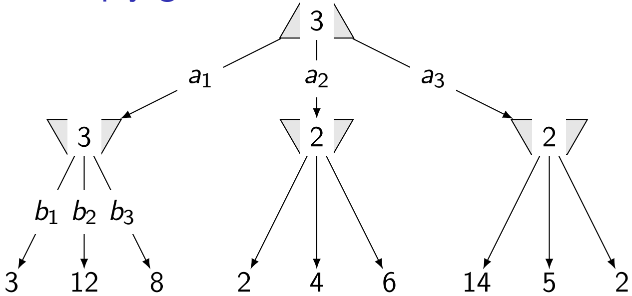


Is it like DFS or BFS?

What is the complexity? How many nodes to visit?

Can we do better? How?

A two ply game, recursive run

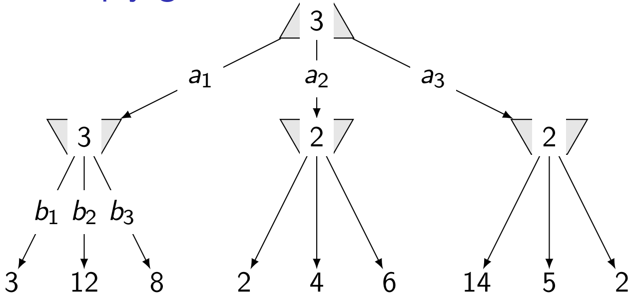


Is it like DFS or BFS?

What is the complexity? How many nodes to visit?

Can we do better? How?

A two ply game, recursive run

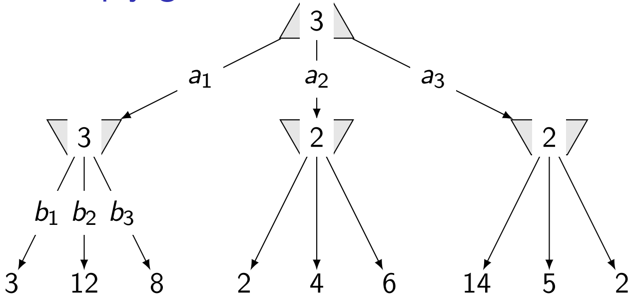


Is it like DFS or BFS?

What is the complexity? How many nodes to visit?

Can we do better? How?

A two ply game, recursive run



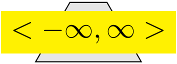
Is it like DFS or BFS?

What is the complexity? How many nodes to visit?

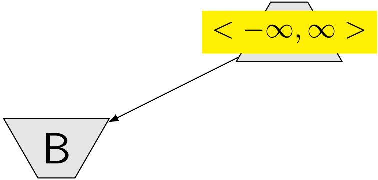
Can we do better? How?

Nodes (sub-trees) worth visiting

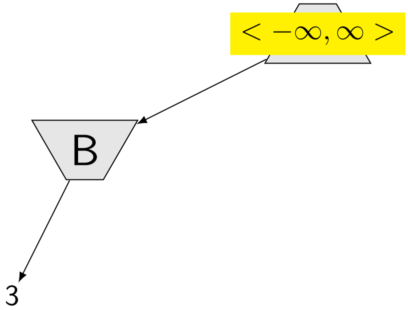
Nodes (sub-trees) worth visiting



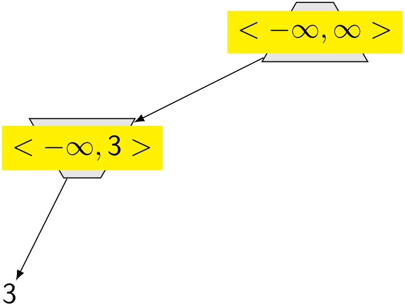
Nodes (sub-trees) worth visiting



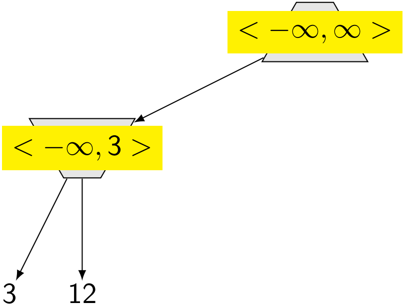
Nodes (sub-trees) worth visiting



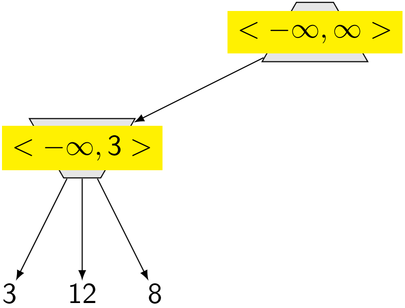
Nodes (sub-trees) worth visiting



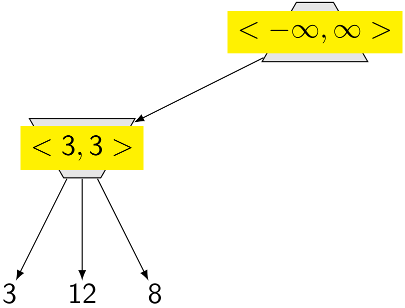
Nodes (sub-trees) worth visiting



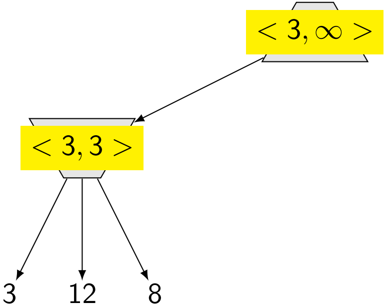
Nodes (sub-trees) worth visiting



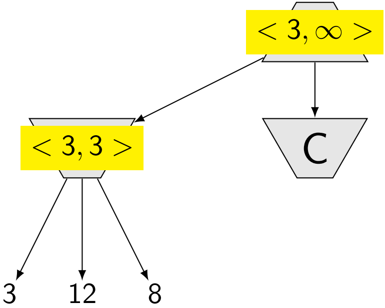
Nodes (sub-trees) worth visiting



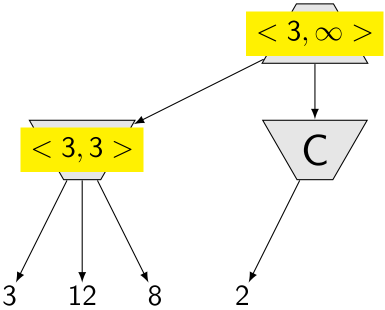
Nodes (sub-trees) worth visiting



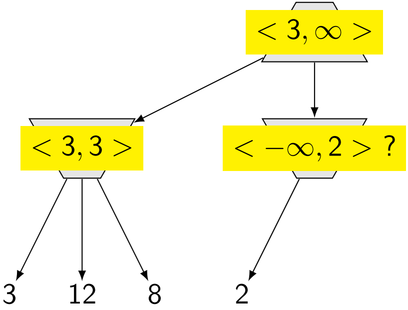
Nodes (sub-trees) worth visiting



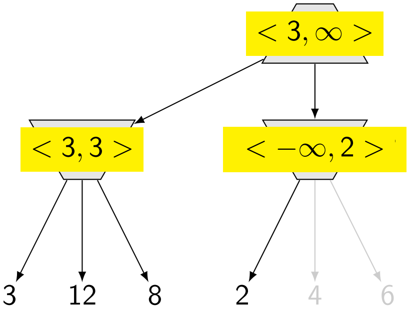
Nodes (sub-trees) worth visiting



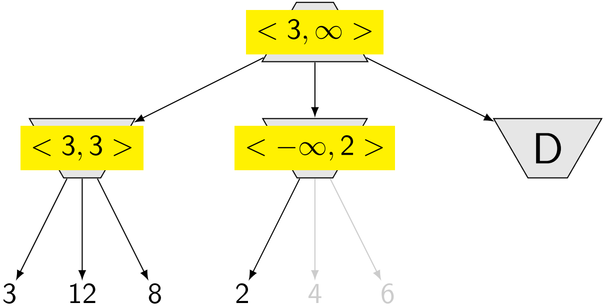
Nodes (sub-trees) worth visiting



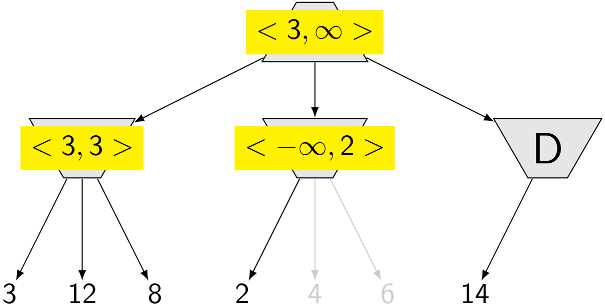
Nodes (sub-trees) worth visiting



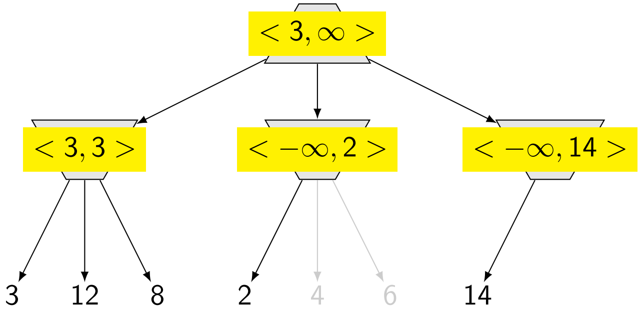
Nodes (sub-trees) worth visiting



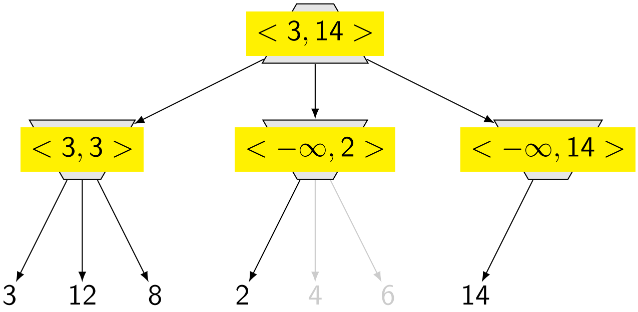
Nodes (sub-trees) worth visiting



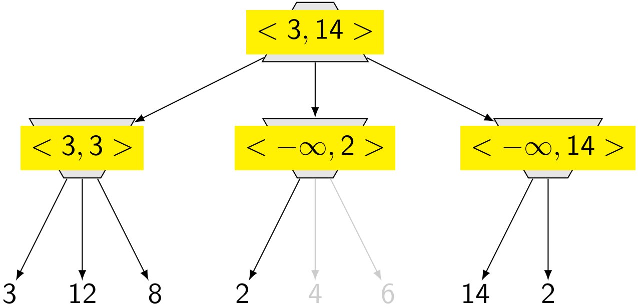
Nodes (sub-trees) worth visiting



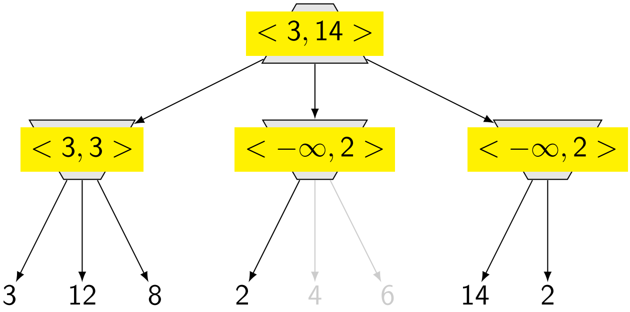
Nodes (sub-trees) worth visiting



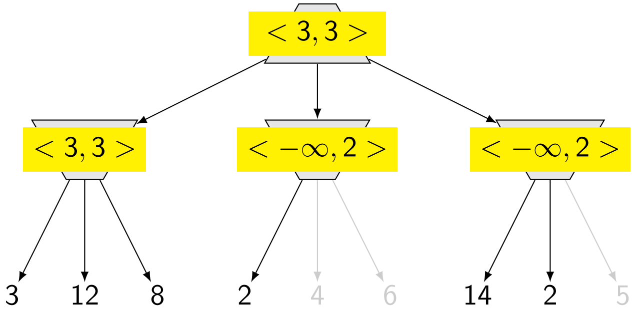
Nodes (sub-trees) worth visiting



Nodes (sub-trees) worth visiting

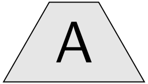


Nodes (sub-trees) worth visiting



α-β pruning

α: highest (best) value choice found so far for any choice along MAX (think "at least")
β: lowest (best) value choice found so far for any choice along MIN (think "at most")

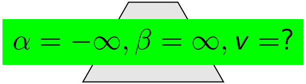


v value of the state

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

α-β pruning

α: highest (best) value choice found so far for any choice along MAX (think "at least")
β: lowest (best) value choice found so far for any choice along MIN (think "at most")

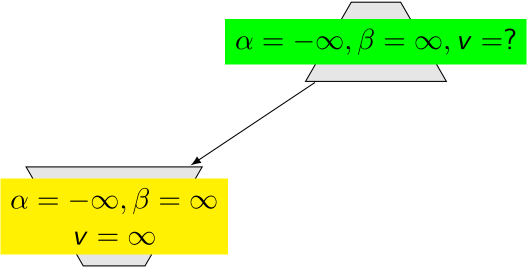


v value of the state

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

α-β pruning

α: highest (best) value choice found so far for any choice along MAX (think "at least")
β: lowest (best) value choice found so far for any choice along MIN (think "at most")

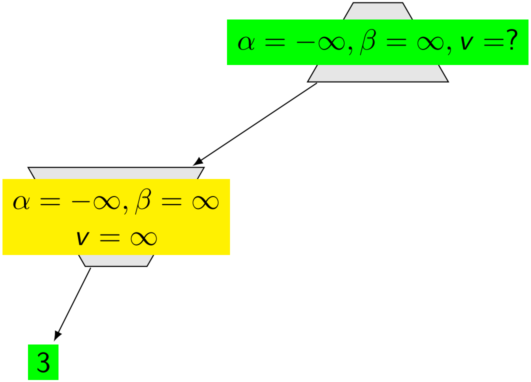


v value of the state

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

α-β pruning

α: highest (best) value choice found so far for any choice along MAX (think "at least")
β: lowest (best) value choice found so far for any choice along MIN (think "at most")



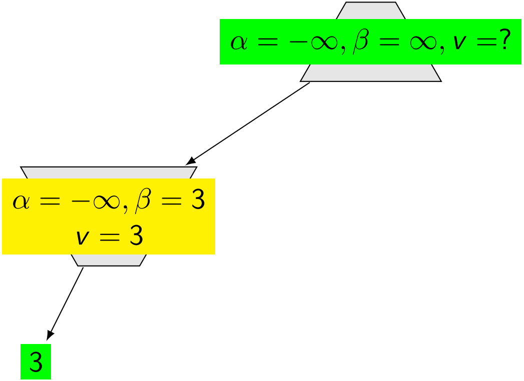
v value of the state

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

α-β pruning

α: highest (best) value choice found so far for any choice along MAX (think "at least")

β: lowest (best) value choice found so far for any choice along MIN (think "at most")



v value of the state

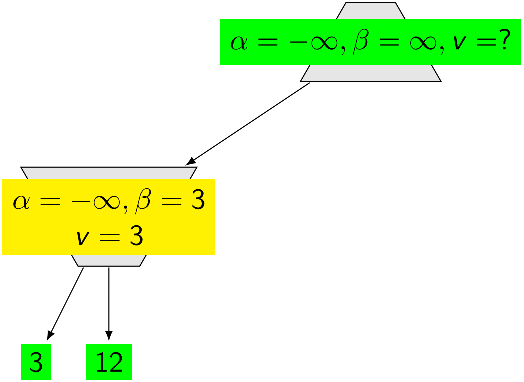
In MIN-VAL: $v \leftarrow 2$

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

α-β pruning

α: highest (best) value choice found so far for any choice along MAX (think "at least")

β: lowest (best) value choice found so far for any choice along MIN (think "at most")



v value of the state

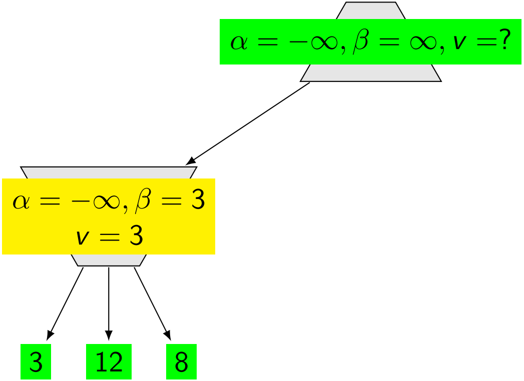
In MIN-VAL: $v \leftarrow 2$

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

α-β pruning

α: highest (best) value choice found so far for any choice along MAX (think "at least")

β: lowest (best) value choice found so far for any choice along MIN (think "at most")



v value of the state

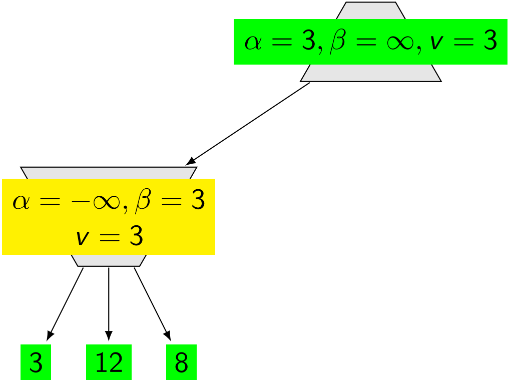
In MIN-VAL: $v \leftarrow 2$

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

α-β pruning

α: highest (best) value choice found so far for any choice along MAX (think "at least")

β: lowest (best) value choice found so far for any choice along MIN (think "at most")



v value of the state

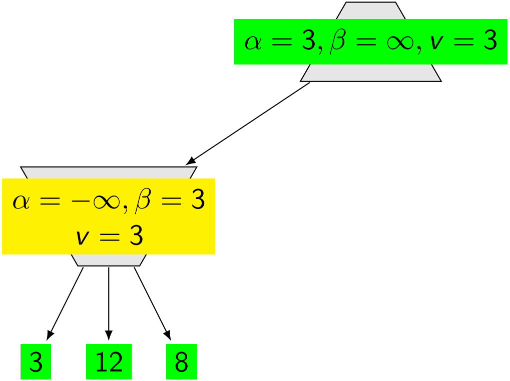
In MIN-VAL: $v \leftarrow 2$

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

α-β pruning

α: highest (best) value choice found so far for any choice along MAX (think "at least")

β: lowest (best) value choice found so far for any choice along MIN (think "at most")



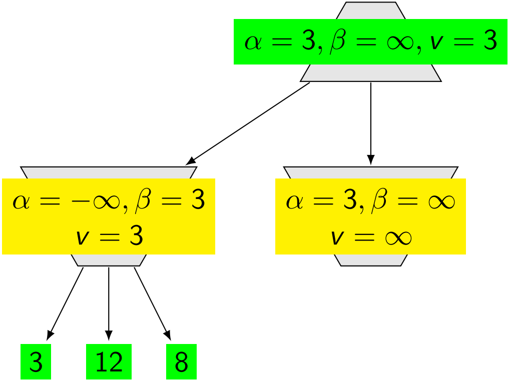
v value of the state

In MIN-VAL: $v \leftarrow 2$

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

α-β pruning

α: highest (best) value choice found so far for any choice along MAX (think "at least")
β: lowest (best) value choice found so far for any choice along MIN (think "at most")

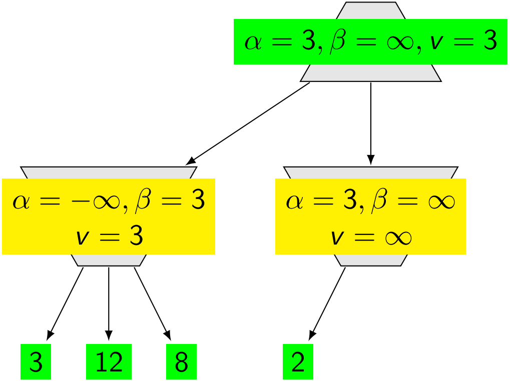


v value of the state

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

α-β pruning

α: highest (best) value choice found so far for any choice along MAX (think "at least")
β: lowest (best) value choice found so far for any choice along MIN (think "at most")



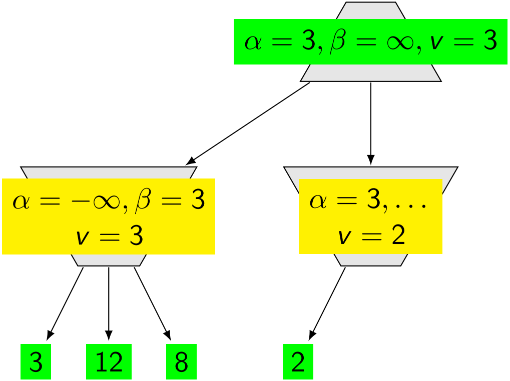
v value of the state

In MIN-VAL: $v \leftarrow 2$

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

α-β pruning

α: highest (best) value choice found so far for any choice along MAX (think "at least")
β: lowest (best) value choice found so far for any choice along MIN (think "at most")



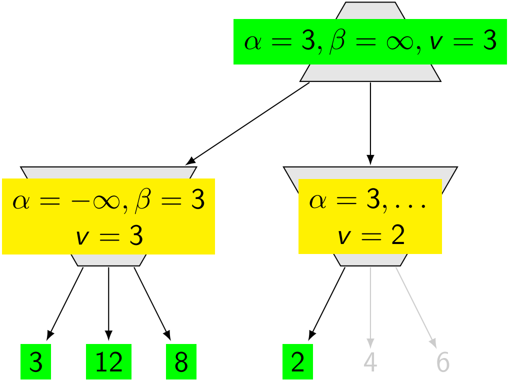
v value of the state

In MIN-VAL: $v \leftarrow 2$

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

α-β pruning

α: highest (best) value choice found so far for any choice along MAX (think "at least")
β: lowest (best) value choice found so far for any choice along MIN (think "at most")

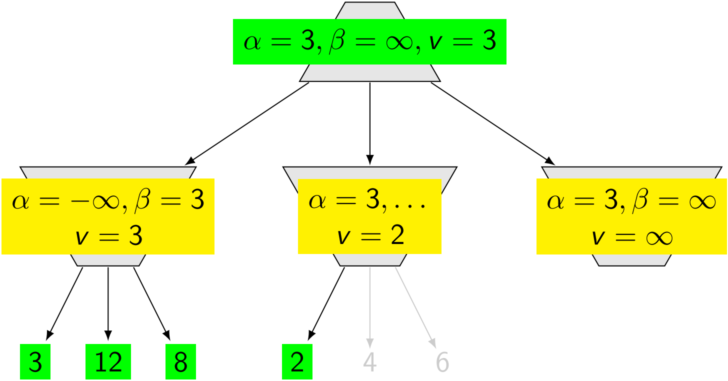


v value of the state

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

α-β pruning

α: highest (best) value choice found so far for any choice along MAX (think "at least")
β: lowest (best) value choice found so far for any choice along MIN (think "at most")



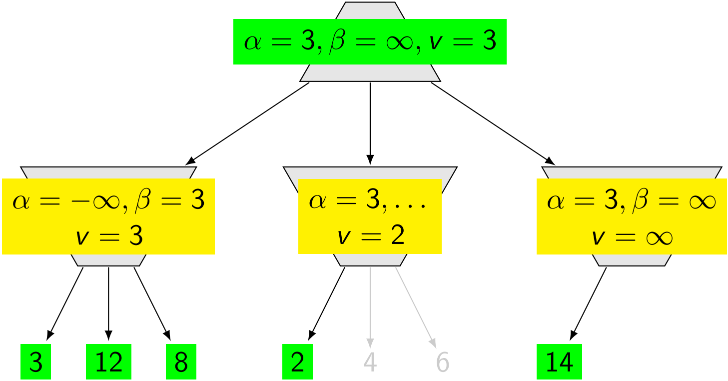
v value of the state

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

α-β pruning

α: highest (best) value choice found so far for any choice along MAX (think "at least")

β: lowest (best) value choice found so far for any choice along MIN (think "at most")



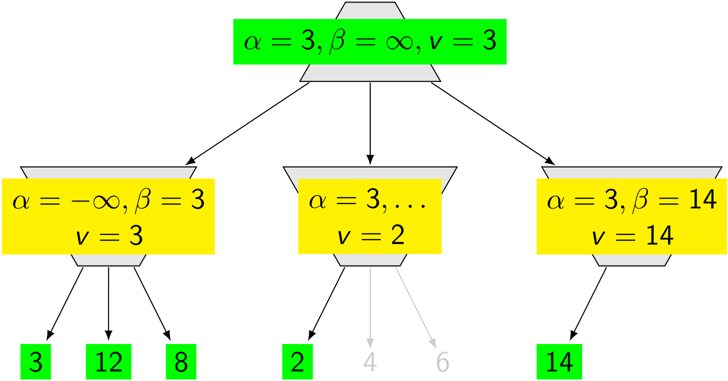
v value of the state

In MIN-VAL: $v \leftarrow 2$

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

α-β pruning

α: highest (best) value choice found so far for any choice along MAX (think "at least")
β: lowest (best) value choice found so far for any choice along MIN (think "at most")



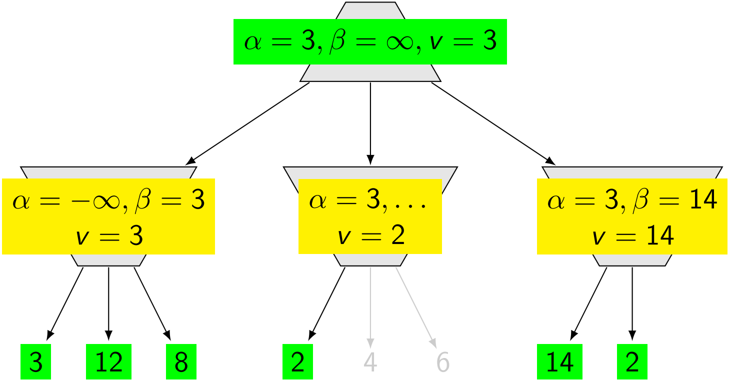
v value of the state

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

α-β pruning

α: highest (best) value choice found so far for any choice along MAX (think "at least")

β: lowest (best) value choice found so far for any choice along MIN (think "at most")



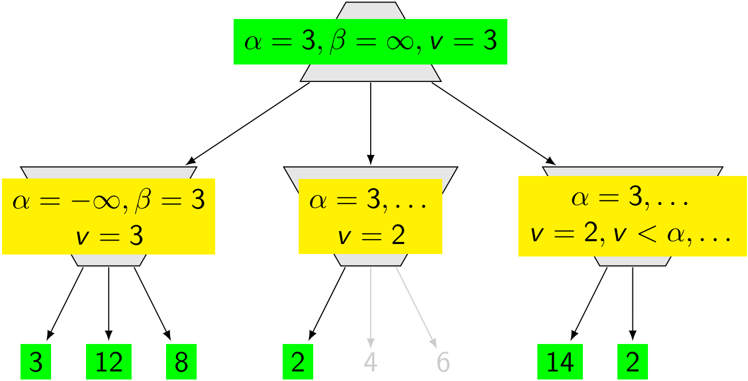
v value of the state

In MIN-VAL: $v \leftarrow 2$

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

α-β pruning

α: highest (best) value choice found so far for any choice along MAX (think "at least")
β: lowest (best) value choice found so far for any choice along MIN (think "at most")



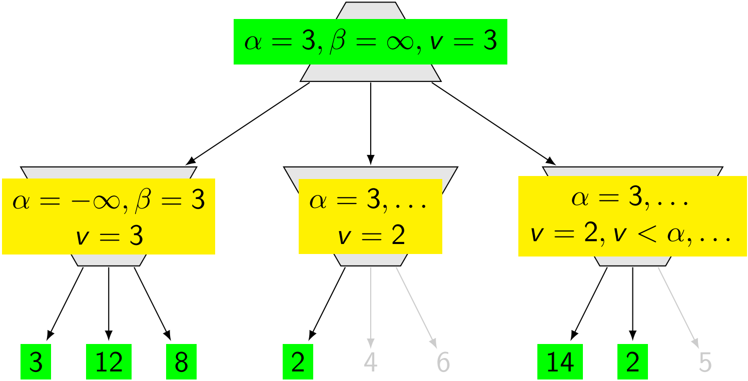
v value of the state

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

α-β pruning

α: highest (best) value choice found so far for any choice along MAX (think "at least")

β: lowest (best) value choice found so far for any choice along MIN (think "at most")



v value of the state

In MIN-VAL: $v \leftarrow 2$

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

α-β pruning – How much can we save?

original: Time: $O(b^m)$

- how to consider next actions/moves (in what order)?
- perfect ordering?

function ALPHA-BETA-SEARCH(state) **returns** an action
  $v \leftarrow$ MAX-VALUE(state, $\alpha = -\infty$, $\beta = \infty$)
  return action corresponding to v

---

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

---

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

function ALPHA-BETA-SEARCH(state) **returns** an action
 $v \leftarrow$ MAX-VALUE(state, $\alpha = -\infty$, $\beta = \infty$)
 return action corresponding to v

---

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

---

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

function ALPHA-BETA-SEARCH(state) **returns** an action
    $v \leftarrow$ MAX-VALUE(state, $\alpha = -\infty$, $\beta = \infty$)
    return action corresponding to v

---

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

---

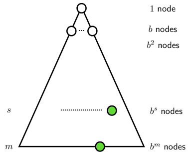
function MIN-VALUE(state, α, β) **returns** a utility value v
    if TERMINAL-TEST(state) **return** UTILITY(state)
    $v \leftarrow \infty$
    for all $a \in$ ACTIONS(state) **do**
        $v \leftarrow$ **min**(v, MAX-VALUE(RESULT(state,a),α, β))
        if $v \leq \alpha$ **return** v
        $\beta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

Recall: Iterative deepening DFS (ID-DFS)

- Start with `maxdepth = 1`
- Perform DFS with limited depth. Report success or failure.
- If failure, forget everything, increase `maxdepth` and repeat DFS.

The “wasting” of resources is not too bad. Recall:

- Most nodes are at the deepest levels.
- Asymptotic complexity unchanged.



Bonus for α-β pruning: previous “shallower” iterations can be reused for node ordering.

Imperfect but real-time decisions: iterative deepening

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

What do we want from the $\text{EVAL}(s,p)$?:

- For terminal states: $\text{EVAL}(s,p) = \text{UTILITY}(s,p)$
- For non-terminal states: $\text{UTILITY}(\textit{loss},p) \leq \text{EVAL}(s,p) \leq \text{UTILITY}(\textit{win},p)$
- Fast enough

Imperfect but real-time decisions: iterative deepening

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

What do we want from the $\text{EVAL}(s,p)$?:

- For terminal states: $\text{EVAL}(s,p) = \text{UTILITY}(s,p)$
- For non-terminal states: $\text{UTILITY}(\textit{loss},p) \leq \text{EVAL}(s,p) \leq \text{UTILITY}(\textit{win},p)$
- Fast enough

Imperfect but real-time decisions: iterative deepening

$$
\textsc{h-minimax}(s, d) = \begin{cases} \textsc{eval}(s,\ \textsc{max}) & \text{if} \quad \textsc{Is-Cutoff}(s, d) \\ \max\limits_{a \in \textsc{actions}(s)} \textsc{h-minimax}(\textsc{result}(s, a), d + 1) & \text{if} \quad \textsc{to-play}(s) = \textsc{max} \\ \min\limits_{a \in \textsc{actions}(s)} \textsc{h-minimax}(\textsc{result}(s, a), d + 1) & \text{if} \quad \textsc{to-play}(s) = \textsc{min} \end{cases}
$$

What do we want from the $\textsc{eval}(s, p)$?:

- For terminal states: $\textsc{eval}(s, p) = \textsc{utility}(s, p)$
- For non-terminal states: $\textsc{utility}(loss, p) \leq \textsc{eval}(s, p) \leq \textsc{utility}(win, p)$
- Fast enough

Imperfect but real-time decisions: iterative deepening

$$
\textsc{h-minimax}(s, d) = \begin{cases} \textsc{eval}(s,\ \textsc{max}) & \text{if} \quad \textsc{Is-Cutoff}(s, d) \\ \max\limits_{a \in \textsc{actions}(s)} \textsc{h-minimax}(\textsc{result}(s, a), d + 1) & \text{if} \quad \textsc{to-play}(s) = \textsc{max} \\ \min\limits_{a \in \textsc{actions}(s)} \textsc{h-minimax}(\textsc{result}(s, a), d + 1) & \text{if} \quad \textsc{to-play}(s) = \textsc{min} \end{cases}
$$

What do we want from the $\textsc{eval}(s, p)$?:

- For terminal states: $\textsc{eval}(s, p) = \textsc{utility}(s, p)$
- For non-terminal states: $\textsc{utility}(loss, p) \leq \textsc{eval}(s, p) \leq \textsc{utility}(win, p)$
- Fast enough

Imperfect but real-time decisions: iterative deepening

$$\textsc{h-minimax}(s,d) = \begin{cases} \textsc{eval}(\mathsf{s},\ \textsc{max}) & \text{if} \quad \textsc{Is-Cutoff}(s,d) \\ \max\limits_{a \in \textsc{actions}(s)} \textsc{h-minimax}(\textsc{result}(s,a), d+1) & \text{if} \quad \textsc{to-play}(s) = \textsc{max} \\ \min\limits_{a \in \textsc{actions}(s)} \textsc{h-minimax}(\textsc{result}(s,a), d+1) & \text{if} \quad \textsc{to-play}(s) = \textsc{min} \end{cases}$$

What do we want from the $\textsc{eval}(s,p)$?:

- For terminal states: $\textsc{eval}(s,p) = \textsc{utility}(s,p)$
- For non-terminal states: $\textsc{utility}(loss, p) \leq \textsc{eval}(s,p) \leq \textsc{utility}(win, p)$
- Fast enough

Imperfect but real-time decisions: iterative deepening

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

What do we want from the $\text{EVAL}(s,p)$?:

- For terminal states: $\text{EVAL}(s,p) = \text{UTILITY}(s,p)$
- For non-terminal states: $\text{UTILITY}(loss, p) \leq \text{EVAL}(s,p) \leq \text{UTILITY}(win, p)$
- Fast enough

Imperfect but real-time decisions: iterative deepening

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

What do we want from the $\text{EVAL}(s, p)$?:

- For terminal states: $\text{EVAL}(s,p) = \text{UTILITY}(s,p)$
- For non-terminal states: $\text{UTILITY}(\mathit{loss}, p) \leq \text{EVAL}(s,p) \leq \text{UTILITY}(\mathit{win}, p)$
- Fast enough

Imperfect but real-time decisions: iterative deepening

$$\text{H-MINIMAX}(s,d) = \begin{cases} \text{EVAL}(s,\ \text{MAX}) & \text{if} \quad \text{Is-Cutoff}(s,d) \\ \max_{a \in \text{ACTIONS}(s)} \text{H-MINIMAX}(\text{RESULT}(s,a), d+1) & \text{if} \quad \text{TO-PLAY}(s) = \text{MAX} \\ \min_{a \in \text{ACTIONS}(s)} \text{H-MINIMAX}(\text{RESULT}(s,a), d+1) & \text{if} \quad \text{TO-PLAY}(s) = \text{MIN} \end{cases}$$

What do we want from the $\text{EVAL}(s,p)$?:

- For terminal states: $\text{EVAL}(s,p) = \text{UTILITY}(s,p)$
- For non-terminal states: $\text{UTILITY}(loss,p) \leq \text{EVAL}(s,p) \leq \text{UTILITY}(win,p)$
- Fast enough

Cutting off search into minimax and α, β search

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

Historical note: cutting search off earlier and use of heuristic evaluation functions proposed by Claude Shannon in *Programming a Computer for Playing Chess* (1950).

EVAL(s) – Evaluation functions

(Estimate of) State value for non-terminal states.
We need an easy-to-compute function correlated with "chance of winning". For chess:

- $f_1(s)$ Material value for pieces—1 for pawn, 3 for knight/bishop, 5 for rook, 10 for queen. (minus opponent's pieces)
- $f_2(s)$ Finetuning: 2 bishops are worth 6.5; knights are worth more in closed positions...
- Other features worth evaluating: controlling the center of the board, good pawn structure (no double pawns), king safety...
- $f_i(s) = \cdots$ We can create many. How to combine them?

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

How to find/compute proper weights?
How to find/create $f_i(s)$?

EVAL(s) – Evaluation functions

(Estimate of) State value for non-terminal states.

We need an easy-to-compute function correlated with “chance of winning”. For chess:

- $f_1(s)$ Material value for pieces—1 for pawn, 3 for knight/bishop, 5 for rook, 10 for queen. (minus opponent’s pieces)
- $f_2(s)$ Finetuning: 2 bishops are worth 6.5; knights are worth more in closed positions...
- Other features worth evaluating: controlling the center of the board, good pawn structure (no double pawns), king safety...
- $f_i(s) = \cdots$ We can create many. How to combine them?

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

How to find/compute proper weights?

How to find/create $f_i(s)$?

EVAL(s) – Evaluation functions

(Estimate of) State value for non-terminal states.

We need an easy-to-compute function correlated with "chance of winning". For chess:

- $f_1(s)$ Material value for pieces—1 for pawn, 3 for knight/bishop, 5 for rook, 10 for queen. (minus opponent's pieces)
- $f_2(s)$ Finetuning: 2 bishops are worth 6.5; knights are worth more in closed positions...
- Other features worth evaluating: controlling the center of the board, good pawn structure (no double pawns), king safety...
- $f_i(s) = \cdots$ We can create many. How to combine them?

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

How to find/compute proper weights?

How to find/create $f_i(s)$?

EVAL(s) – Evaluation functions

(Estimate of) State value for non-terminal states.
We need an easy-to-compute function correlated with "chance of winning". For chess:

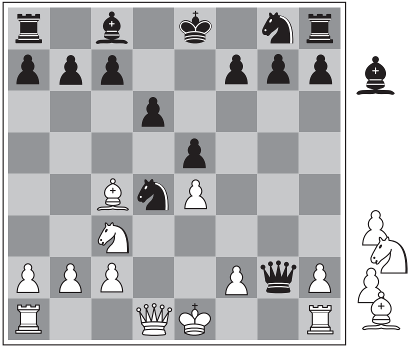
- $f_1(s)$ Material value for pieces—1 for pawn, 3 for knight/bishop, 5 for rook, 10 for queen. (minus opponent's pieces)
- $f_2(s)$ Finetuning: 2 bishops are worth 6.5; knights are worth more in closed positions...
- Other features worth evaluating: controlling the center of the board, good pawn structure (no double pawns), king safety...
- $f_i(s) = \cdots$ We can create many. How to combine them?

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

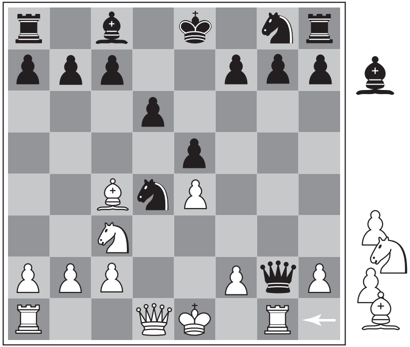
How to find/compute proper weights?
How to find/create $f_i(s)$?

EVAL(s) – Problems

What if something important happens just after the cut – in the next ply?



(a) White to move



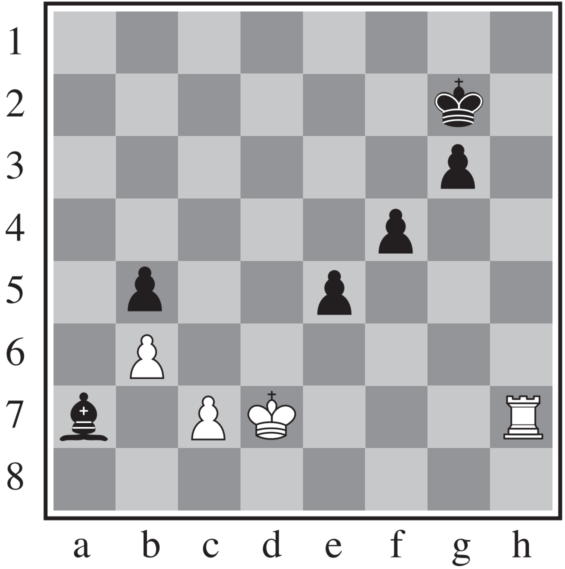
(b) White to move

Additional improvements:

- "Killer moves"—moves that prevent oponent to play a very good move.
- *Quiescence search* – EVAL function should be applied only once things calm down. During capturing of pieces, depth should be locally increased.

Horizon effect

Pushing unavoidable loss deeper in tree by a delaying tactics. We know it is useless but does the machine?
See the situation on right. Black is on move, her bishop is surely doomed. However, the inevitable loss can be postponed by moving her pawns and checking the white king. Depending on the searchable depth this may put the loss over the horizon and moving pawns may look promising.



Computer play vs. grandmaster play

- Computers are better since 1997 (Deep Blue defeating Garry Kasparov).
- The way they play is still very different: "dumb", relying on "brute force".
  - Deep Blue examined 200M positions per second.
  - In some cases, depth of search was 40 ply.
- Grandmasters do not excel in being able to compute very deep—many moves ahead.
  - They play based on experience: super-effective pruning and evaluation functions.
  - They consider only 2 to 3 moves in most positions (branching factor).

Monte Carlo Tree Search (MCTS)

- Simulate from state s.
- $V(s)$ average utility from the simulations
- Pure randomness may be not enough.
- Selection policy.
- Exploration vs. Exploitation (see RL in few weeks)
- Combine MCTS with evaluation heurstics.
- Learn from available game recordings.

Monte Carlo Tree Search (MCTS)

- Simulate from state s.
- $V(s)$ average utility from the simulations
- Pure randomness may be not enough.
- Selection policy.
- Exploration vs. Exploitation (see RL in few weeks)
- Combine MCTS with evaluation heurstics.
- Learn from available game recordings.

Monte Carlo Tree Search (MCTS)

- Simulate from state s.
- $V(s)$ average utility from the simulations
- Pure randomness may be not enough.
- Selection policy.
- Exploration vs. Exploitation (see RL in few weeks)
- Combine MCTS with evaluation heurstics.
- Learn from available game recordings.

Adversarial search - Summary

- Recursive algorithm – repeating What–if
- Search tree too huge – cutting, sorting candidate branches
- Value of a state $V(s, p) = \max_{s' \in \text{children}(s)} V(s', p)$
- $V(s, p)$ estimate for non-terminal states
- $\textsc{utility}(loss, p) \leq \textsc{eval}(s, p) \leq \textsc{utility}(win, p)$

References and further reading

Many images, including the chess plates are from Chapter 5, "Adversarial search" in [1]. Notation has been modified according to the new edition [2]; Chapter 6, "Adversarial search and games". Connection to Reinforcement Learning that comes in few weeks can be easily seen in section 1.5 in [3].

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

[2] Stuart Russell and Peter Norvig.
Artificial Intelligence: A Modern Approach.
Prentice Hall, 4th edition, 2021.

[3] 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.