# 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

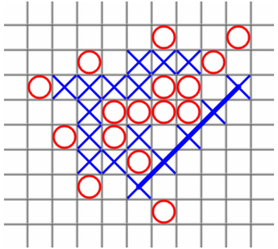
**Video: Adversing visual segmentation**

Vision for Robotics and Autonomous Systems, `http://cyber.felk.cvut.cz/vras`

# 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



Considering the notation, we are making slight transition from [1] to [2].

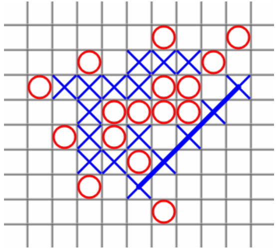
- Players: $P = \{1, 2, \ldots, N\}$ (often just $N = 2$)
- Transition functions: $S \times A \rightarrow S$.
- Terminal utilities: $S \times P \rightarrow R$. ($R$ - as a Reward)

What are we loking for? A strategy/policy $S \rightarrow A$

# 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



Considering the notation, we are making slight transition from [1] to [2].

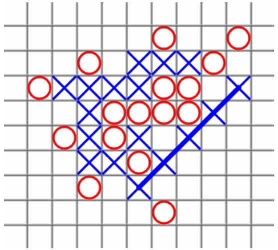
- Players: $P = \{1, 2, \ldots, N\}$ (often just $N = 2$)
- Transition functions: $S \times A \to S$.
- Terminal utilities: $S \times P \to R$. ($R$ - as a Reward)

What are we loking for? A strategy/policy $S \to A$

# 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



Considering the notation, we are making slight transition from [1] to [2].

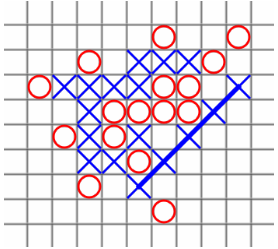
- Players: $P = \{1, 2, \ldots, N\}$ (often just $N = 2$)
- Transition functions: $S \times A \to S$.
- Terminal utilities: $S \times P \to R$. ($R$ - as a Reward)

What are we loking for? A strategy/policy $S \to A$

# 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



Considering the notation, we are making slight transition from [1] to [2].

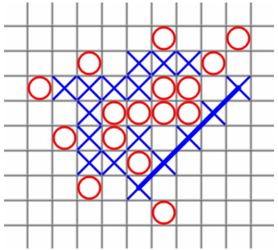
- Players: $P = \{1, 2, \ldots, N\}$ (often just $N = 2$)
- Transition functions: $S \times A \to S$.
- Terminal utilities: $S \times P \to R$. ($R$ - as a Reward)

What are we loking for? A strategy/policy $S \to A$

# 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



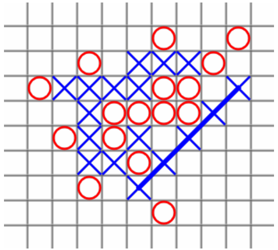
Considering the notation, we are making slight transition from [1] to [2].

- Players: $P = \{1, 2, \ldots, N\}$ (often just $N = 2$)
- Transition functions: $S \times A \to S$.
- Terminal utilities: $S \times P \to R$. ($R$ - as a Reward)

What are we loking for? A strategy/policy $S \to A$

# 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



Considering the notation, we are making slight transition from [1] to [2].

- Players: $P = \{1, 2, \ldots, N\}$ (often just $N = 2$)
- Transition functions: $S \times A \rightarrow S$.
- Terminal utilities: $S \times P \rightarrow R$. ($R$ - as a Reward)

What are we loking for? A strategy/policy $S \rightarrow A$

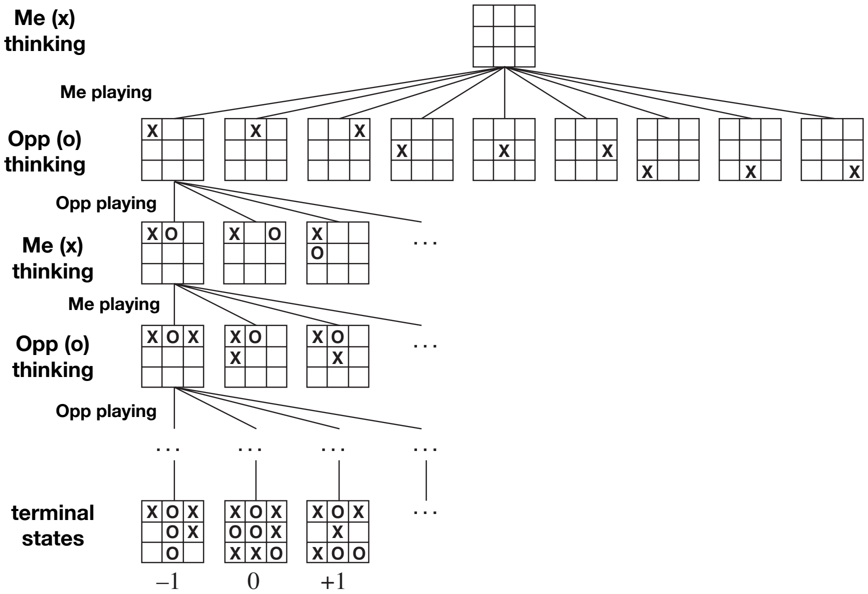
# Terminal utilitity: Zero-Sum and General games

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

Init state, ACTIONS function, and RESULT function defines game tree

# State Value $V(s)$

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

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



Think about the State Value. It is a theoretical construct, definition. Depending on the problem, there may be various computational algorithms. In a game, what State Values are known? Usually, only terminal states.

Think, for a moment, you are the only player. You can control every step.

How would you compute the $V(s)$ for a given state $s$?

# State Value $V(s)$

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

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



Think about the State Value. It is a theoretical construct, definition. Depending on the problem, there may be various computational algorithms. In a game, what State Values are known? Usually, only terminal states.

Think, for a moment, you are the only player. You can control every step.

How would you compute the $V(s)$ for a given state $s$?

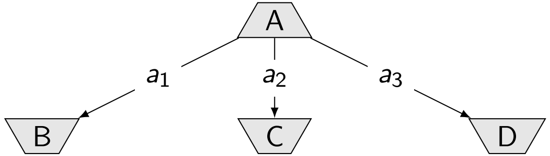
## Two-ply game: **max** for me, **min** for the opponent.


$$a_1 = \operatorname*{arg\,max}_{a \in \textsc{Actions}(A)} \textsc{Result}(A, a)$$



I'm player that starts (state A) and want to decide what to play, actions/plays/moves $a_1, a_2, a_3$ are the options. B, C, D are the possible outcomes of my moves. Now the opponent is about to play. The numbers in terminal states denote *my* profit/utility.

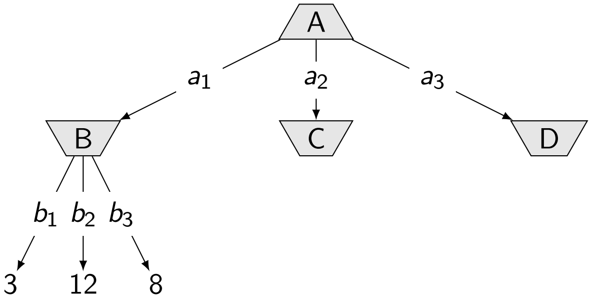
# Two-ply game: **max** for me, **min** for the opponent.




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

I'm player that starts (state A) and want to decide what to play, actions/plays/moves $a_1, a_2, a_3$ are the options. B, C, D are the possible outcomes of my moves. Now the opponent is about to play. The numbers in terminal states denote *my* profit/utility.

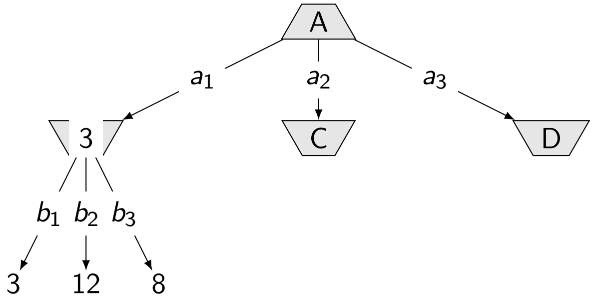
# Two-ply game: **max** for me, **min** for the opponent.




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

I'm player that starts (state A) and want to decide what to play, actions/plays/moves $a_1, a_2, a_3$ are the options. B, C, D are the possible outcomes of my moves. Now the opponent is about to play. The numbers in terminal states denote *my* profit/utility.

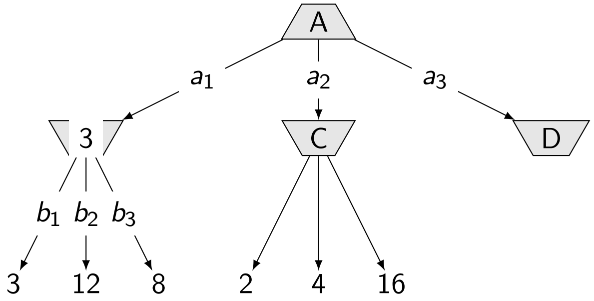
# Two-ply game: **max** for me, **min** for the opponent.




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

I'm player that starts (state A) and want to decide what to play, actions/plays/moves $a_1, a_2, a_3$ are the options. B, C, D are the possible outcomes of my moves. Now the opponent is about to play. The numbers in terminal states denote *my* profit/utility.

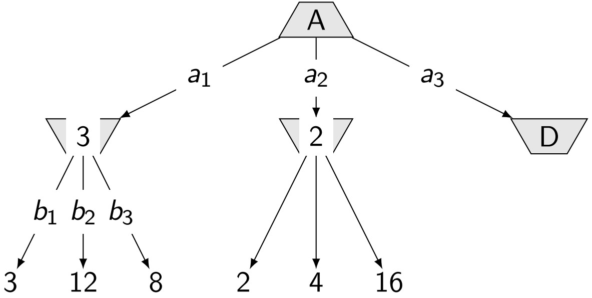
# Two-ply game: **max** for me, **min** for the opponent.




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

I'm player that starts (state A) and want to decide what to play, actions/plays/moves $a_1, a_2, a_3$ are the options. B, C, D are the possible outcomes of my moves. Now the opponent is about to play. The numbers in terminal states denote *my* profit/utility.

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

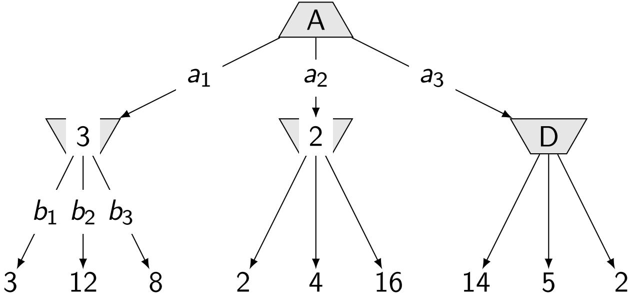



$$a_1 = \underset{a \in \textsc{Actions}(A)}{\arg\max} \; \textsc{Result}(A, a)$$



I'm player that starts (state A) and want to decide what to play, actions/plays/moves $a_1, a_2, a_3$ are the options. B, C, D are the possible outcomes of my moves. Now the opponent is about to play. The numbers in terminal states denote *my* profit/utility.

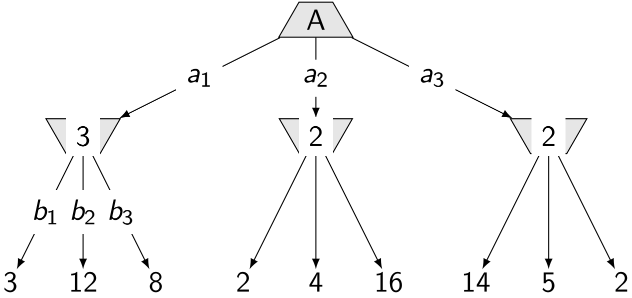
# Two-ply game: **max** for me, **min** for the opponent.




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

I'm player that starts (state A) and want to decide what to play, actions/plays/moves $a_1, a_2, a_3$ are the options. B, C, D are the possible outcomes of my moves. Now the opponent is about to play. The numbers in terminal states denote *my* profit/utility.

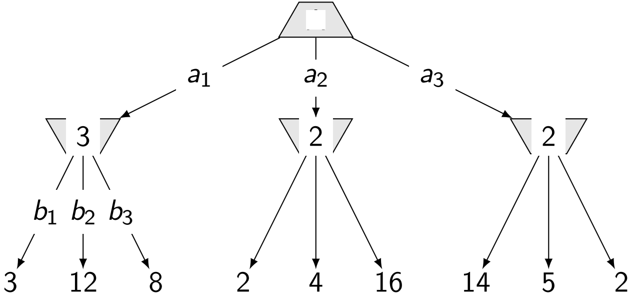
Two-ply game: **max** for me, **min** for the opponent.




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

I'm player that starts (state A) and want to decide what to play, actions/plays/moves $a_1, a_2, a_3$ are the options. B, C, D are the possible outcomes of my moves. Now the opponent is about to play. The numbers in terminal states denote *my* profit/utility.

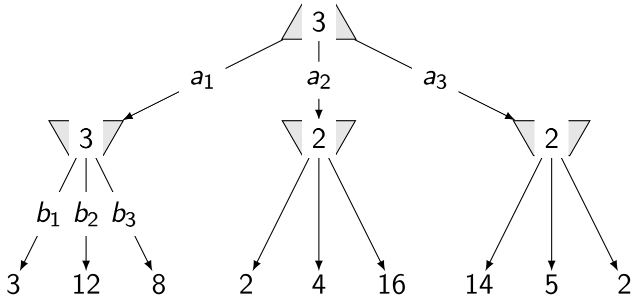
# Two-ply game: **max** for me, **min** for the opponent.




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

I'm player that starts (state A) and want to decide what to play, actions/plays/moves $a_1, a_2, a_3$ are the options. B, C, D are the possible outcomes of my moves. Now the opponent is about to play. The numbers in terminal states denote *my* profit/utility.

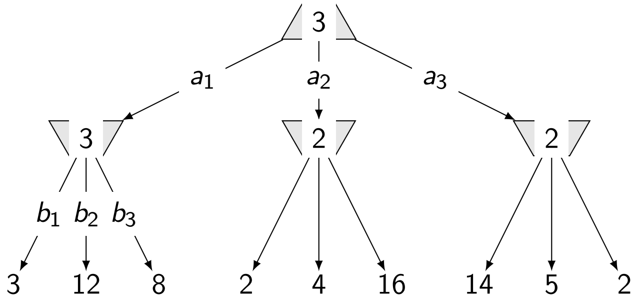
# Two-ply game: **max** for me, **min** for the opponent.




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

I'm player that starts (state A) and want to decide what to play, actions/plays/moves $a_1, a_2, a_3$ are the options. B, C, D are the possible outcomes of my moves. Now the opponent is about to play. The numbers in terminal states denote *my* profit/utility.

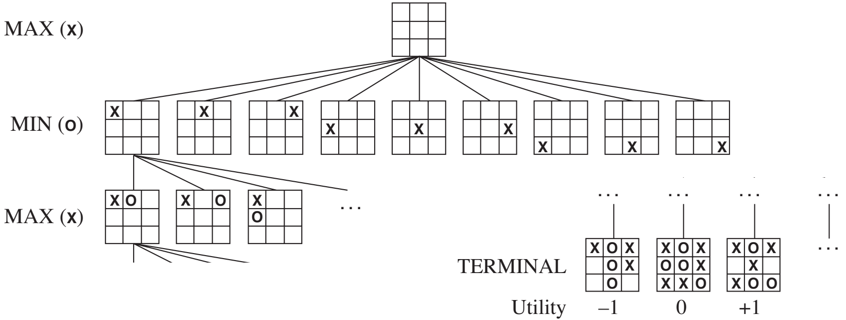
# Two-ply game: **max** for me, **min** for the opponent.




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

I'm player that starts (state A) and want to decide what to play, actions/plays/moves $a_1, a_2, a_3$ are the options. B, C, D are the possible outcomes of my moves. Now the opponent is about to play. The numbers in terminal states denote *my* profit/utility.

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



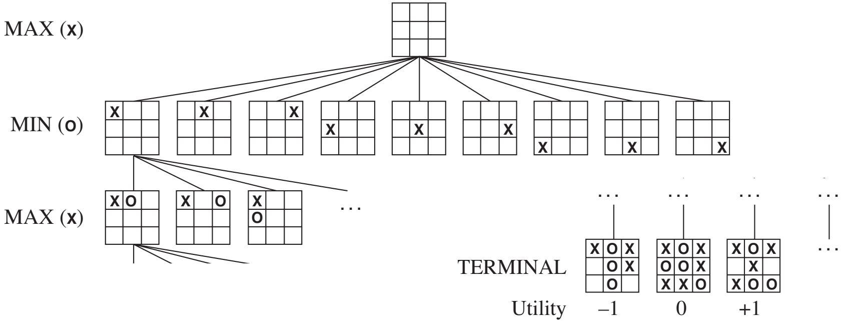

$$\text{MINIMAX}(s) = \begin{cases} \text{UTILITY}(s) & \text{if} \quad \text{TERMINAL-TEST}(s) \\ \max_{a \in \text{ACTIONS}(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if} \quad \text{PLAYER}(s) = \text{MAX} \\ \min_{a \in \text{ACTIONS}(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if} \quad \text{PLAYER}(s)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



Max step: I want to maximize my outcome.

Min step: I want to minimize the outcome of the opponent.

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



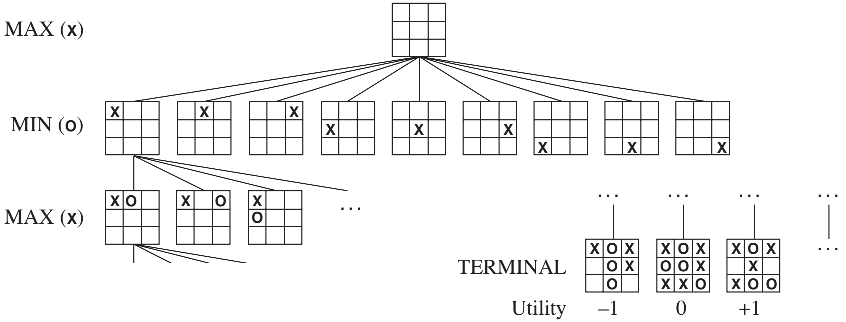

$$\text{MINIMAX}(s) = \begin{cases} \text{UTILITY}(s) & \text{if } \text{TERMINAL-TEST}(s) \\ \max_{a \in \text{ACTIONS}(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if } \text{PLAYER}(s) = \text{MAX} \\ \min_{a \in \text{ACTIONS}(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if } \text{PLAYER}(s)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



Max step: I want to maximize my outcome.

Min step: I want to minimize the outcome of the opponent.

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



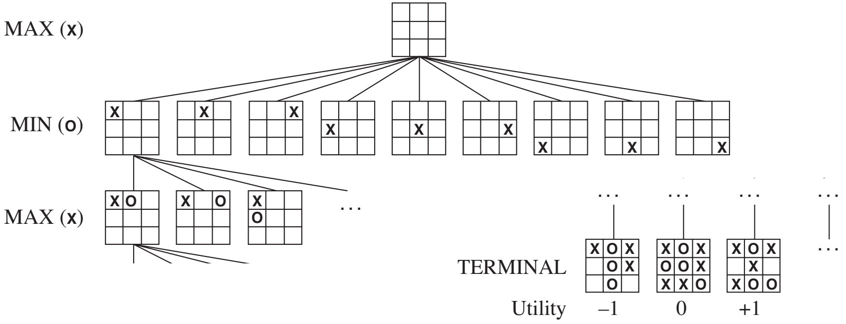

$$\textsc{minimax}(s) = \begin{cases} \textsc{utility}(s) & \text{if } \textsc{terminal-test}(s) \\ \max_{a \in \textsc{actions}(s)} \textsc{minimax}(\textsc{result}(s,a)) & \text{if } \textsc{player}(s) = \textsc{max} \\ \min_{a \in \textsc{actions}(s)} \textsc{minimax}(\textsc{result}(s,a)) & \text{if } \textsc{player}(s) = \textsc{min} \end{cases}$$



Max step: I want to maximize my outcome.

Min step: I want to minimize the outcome of the opponent.

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



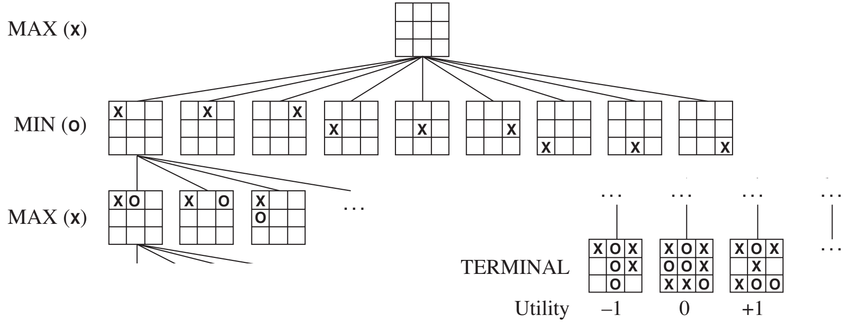

$$
\text{MINIMAX}(s) =
\begin{cases}
\text{UTILITY}(s) & \text{if } \text{TERMINAL-TEST}(s) \\
\max_{a \in \text{ACTIONS}(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if } \text{PLAYER}(s) = \text{MAX} \\
\min_{a \in \text{ACTIONS}(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if } \text{PLAYER}(s)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



Max step: I want to maximize my outcome.

Min step: I want to minimize the outcome of the opponent.

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




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



Max step: I want to maximize my outcome.

Min step: I want to minimize the outcome of the opponent.

# Minimax algorithm

**function** MINIMAX(state) **returns** an action
  **return** $\underset{a \in \text{Actions}(s)}{\textbf{argmax}}$ 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 argmax_{a∈Actions(s)} 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 ← ∞
    for all ACTIONS(state) do
        v ← 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 ← −∞
    for all ACTIONS(state) do
        v ← max(v, MIN-VALUE(RESULT(state,a)))
    end for
end function
```

# Minimax algorithm

```
function MINIMAX(state) returns an action
    return argmax_{a∈Actions(s)} 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 ← ∞
    for all ACTIONS(state) do
        v ← min(v, MAX-VALUE(RESULT(state,a)))
    end for
end function
```

---

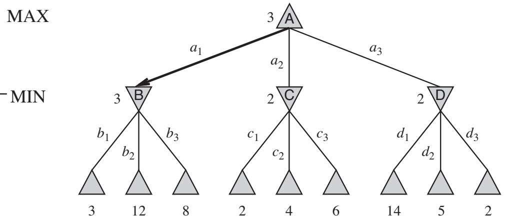
```
function MAX-VALUE(state) returns a utility value v
    if TERMINAL-TEST(state) then return UTILITY(state)
    end if
    v ← −∞
    for all ACTIONS(state) do
        v ← max(v, MIN-VALUE(RESULT(state,a)))
    end for
end function
```

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

**function** MINIMAX($s$) **returns** $a$
  $\underset{a \in \text{Actions}(s)}{\text{arg}\mathbf{max}}$ MINVAL(RES($s$, $a$))
**end function**

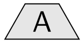
**function** MINVAL($s$) **returns** $v$
  **if** TERMINAL($s$) **then** UTIL($s$)
  **end if**
  $v \leftarrow \infty$
  **for all** ACTIONS($s$) **do**
    $v \leftarrow$ **min**($v$, MAXVAL(RES($s$, $a$)))
  **end for**
**end function**

**function** MAXVAL($s$) **returns** $v$
  **if** TERMINAL($s$) **then** UTIL($s$)
  **end if**
  $v \leftarrow -\infty$
  **for all** ACTIONS($s$) **do**
    $v \leftarrow$ **max**($v$, MINVAL(RES($s$, $a$)))
  **end for**
**end function**




Before going to the animation on the next slide, try to follow the algorithm by a pencil and paper.

# A two ply game, recursive run




What is the complexity? How many nodes to visit?

Can we do better? How?



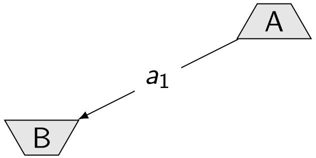
Efficiency/complexity:

- Exhaustive DFS
- Time $O(b^m)$
- Space $O(bm)$

Chess $b \approx 35, m \approx 100$ ...

- We cannot go(dive) to the end
- Can we save something?

# A two ply game, recursive run




What is the complexity? How many nodes to visit?

Can we do better? How?



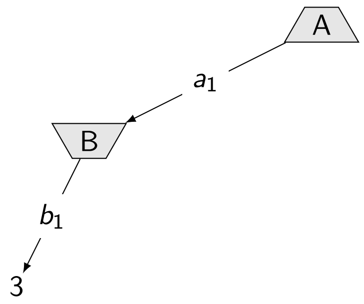
Efficiency/complexity:

- Exhaustive DFS
- Time $O(b^m)$
- Space $O(bm)$

Chess $b \approx 35, m \approx 100 \ldots$

- We cannot go(dive) to the end
- Can we save something?

# A two ply game, recursive run




What is the complexity? How many nodes to visit?

Can we do better? How?



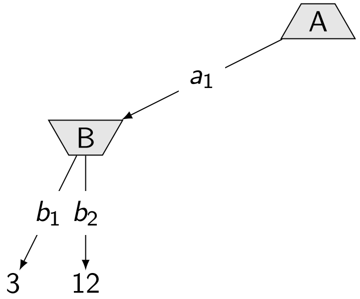
Efficiency/complexity:

- Exhaustive DFS
- Time $O(b^m)$
- Space $O(bm)$

Chess $b \approx 35, m \approx 100 \ldots$

- We cannot go(dive) to the end
- Can we save something?

# A two ply game, recursive run




What is the complexity? How many nodes to visit?

Can we do better? How?



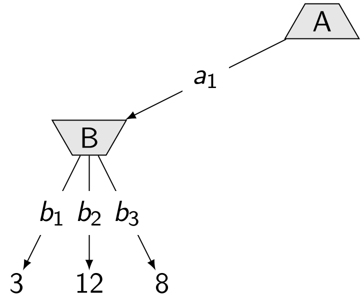
Efficiency/complexity:

- Exhaustive DFS
- Time $O(b^m)$
- Space $O(bm)$

Chess $b \approx 35, m \approx 100 \ldots$

- We cannot go(dive) to the end
- Can we save something?

# A two ply game, recursive run




What is the complexity? How many nodes to visit?

Can we do better? How?

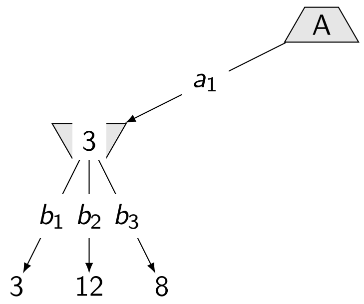
Efficiency/complexity:

- Exhaustive DFS
- Time $O(b^m)$
- Space $O(bm)$

Chess $b \approx 35, m \approx 100 \ldots$

- We cannot go(dive) to the end
- Can we save something?

# A two ply game, recursive run




What is the complexity? How many nodes to visit?

Can we do better? How?

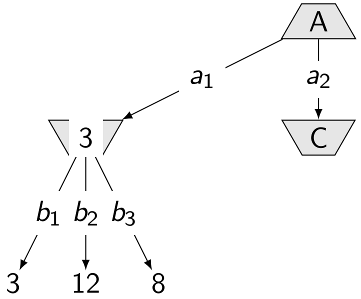
Efficiency/complexity:

- Exhaustive DFS
- Time $O(b^m)$
- Space $O(bm)$

Chess $b \approx 35, m \approx 100 \ldots$

- We cannot go(dive) to the end
- Can we save something?

# A two ply game, recursive run




What is the complexity? How many nodes to visit?

Can we do better? How?



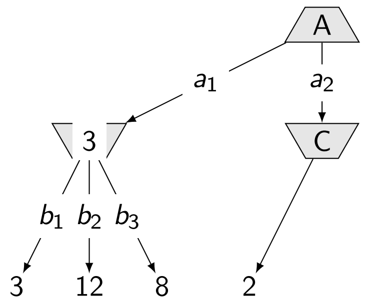
Efficiency/complexity:

- Exhaustive DFS
- Time $O(b^m)$
- Space $O(bm)$

Chess $b \approx 35, m \approx 100 \ldots$

- We cannot go(dive) to the end
- Can we save something?

# A two ply game, recursive run




What is the complexity? How many nodes to visit?

Can we do better? How?

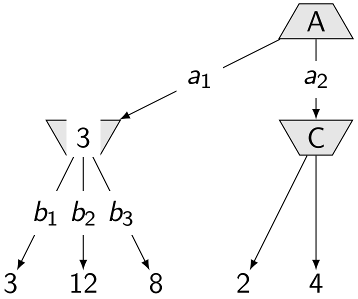
Efficiency/complexity:

- Exhaustive DFS
- Time $O(b^m)$
- Space $O(bm)$

Chess $b \approx 35, m \approx 100$ ...

- We cannot go(dive) to the end
- Can we save something?

# A two ply game, recursive run




What is the complexity? How many nodes to visit?

Can we do better? How?

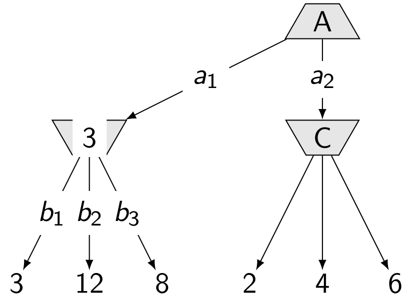
Efficiency/complexity:

- Exhaustive DFS
- Time $O(b^m)$
- Space $O(bm)$

Chess $b \approx 35, m \approx 100 \ldots$

- We cannot go(dive) to the end
- Can we save something?

# A two ply game, recursive run




What is the complexity? How many nodes to visit?

Can we do better? How?

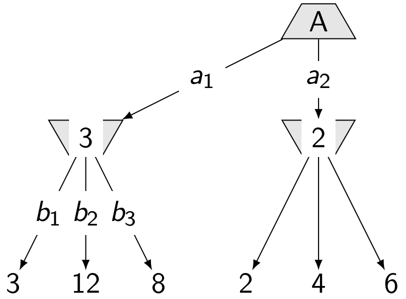
Efficiency/complexity:

- Exhaustive DFS
- Time $O(b^m)$
- Space $O(bm)$

Chess $b \approx 35, m \approx 100 \ldots$

- We cannot go(dive) to the end
- Can we save something?

# A two ply game, recursive run




What is the complexity? How many nodes to visit?

Can we do better? How?

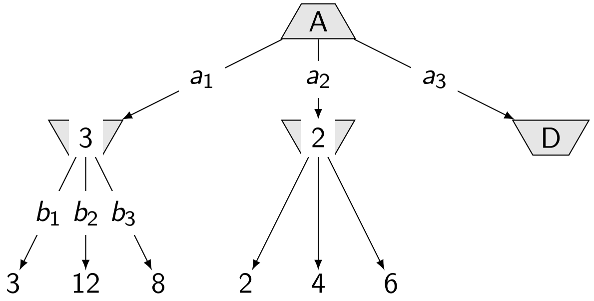
Efficiency/complexity:

- Exhaustive DFS
- Time $O(b^m)$
- Space $O(bm)$

Chess $b \approx 35, m \approx 100 \ldots$

- We cannot go(dive) to the end
- Can we save something?

# A two ply game, recursive run




What is the complexity? How many nodes to visit?

Can we do better? How?



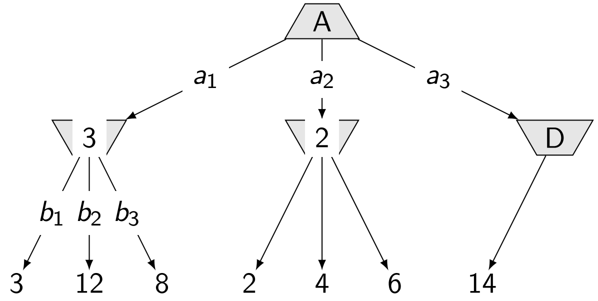
Efficiency/complexity:

- Exhaustive DFS
- Time $O(b^m)$
- Space $O(bm)$

Chess $b \approx 35, m \approx 100 \ldots$

- We cannot go(dive) to the end
- Can we save something?

# A two ply game, recursive run




What is the complexity? How many nodes to visit?

Can we do better? How?



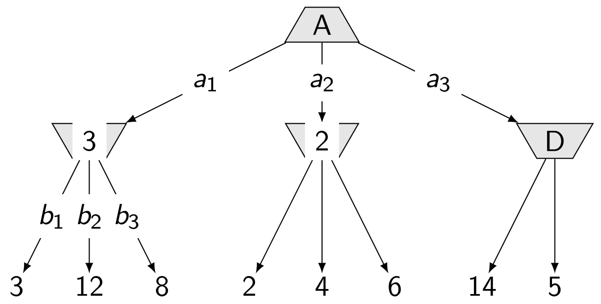
Efficiency/complexity:

- Exhaustive DFS
- Time $O(b^m)$
- Space $O(bm)$

Chess $b \approx 35, m \approx 100$ ...

- We cannot go(dive) to the end
- Can we save something?

# A two ply game, recursive run




What is the complexity? How many nodes to visit?

Can we do better? How?



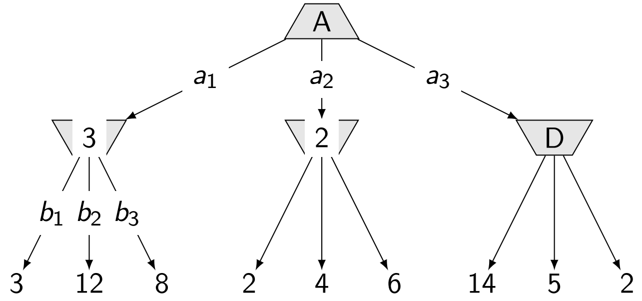
Efficiency/complexity:

- Exhaustive DFS
- Time $O(b^m)$
- Space $O(bm)$

Chess $b \approx 35, m \approx 100 \ldots$

- We cannot go(dive) to the end
- Can we save something?

# A two ply game, recursive run




What is the complexity? How many nodes to visit?

Can we do better? How?

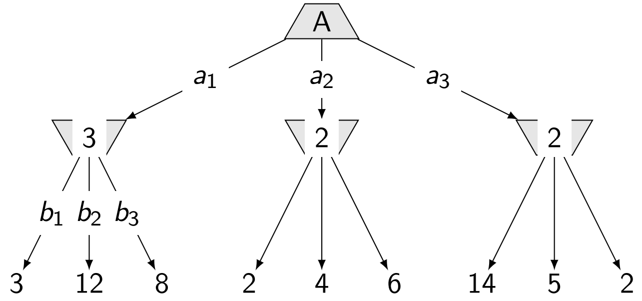
Efficiency/complexity:

- Exhaustive DFS
- Time $O(b^m)$
- Space $O(bm)$

Chess $b \approx 35, m \approx 100 \ldots$

- We cannot go(dive) to the end
- Can we save something?

# A two ply game, recursive run




What is the complexity? How many nodes to visit?

Can we do better? How?



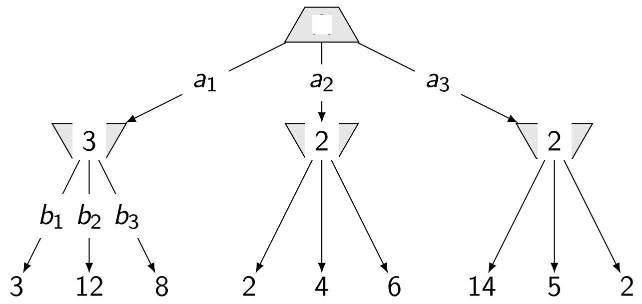
Efficiency/complexity:

- Exhaustive DFS
- Time $O(b^m)$
- Space $O(bm)$

Chess $b \approx 35, m \approx 100 \ldots$

- We cannot go(dive) to the end
- Can we save something?

# A two ply game, recursive run




What is the complexity? How many nodes to visit?

Can we do better? How?



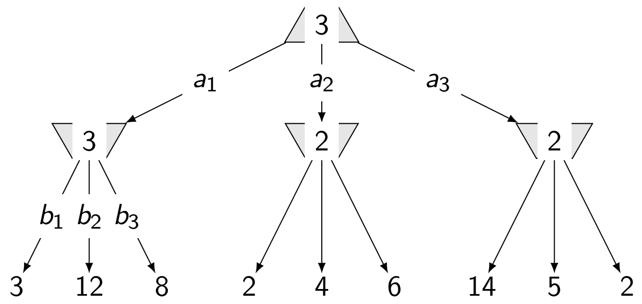
Efficiency/complexity:

- Exhaustive DFS
- Time $O(b^m)$
- Space $O(bm)$

Chess $b \approx 35, m \approx 100 \ldots$

- We cannot go(dive) to the end
- Can we save something?

# A two ply game, recursive run




What is the complexity? How many nodes to visit?

Can we do better? How?



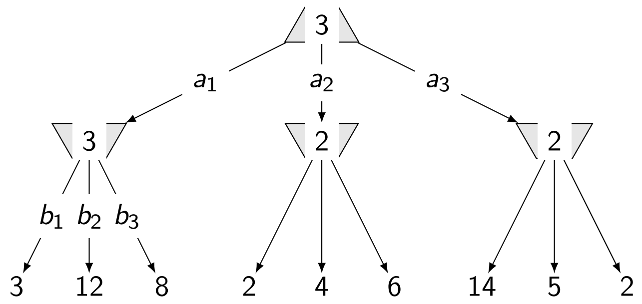
Efficiency/complexity:

- Exhaustive DFS
- Time $O(b^m)$
- Space $O(bm)$

Chess $b \approx 35, m \approx 100 \ldots$

- We cannot go(dive) to the end
- Can we save something?

# A two ply game, recursive run




What is the complexity? How many nodes to visit?

Can we do better? How?



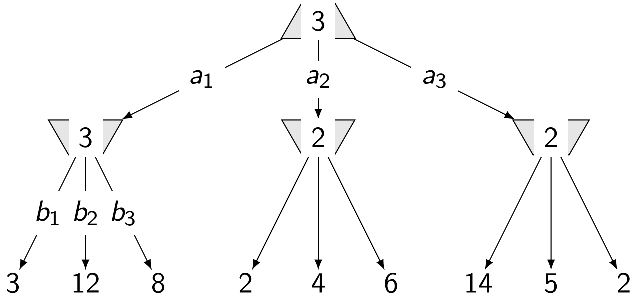
Efficiency/complexity:

- Exhaustive DFS
- Time $O(b^m)$
- Space $O(bm)$

Chess $b \approx 35, m \approx 100 \ldots$

- We cannot go(dive) to the end
- Can we save something?

# A two ply game, recursive run




What is the complexity? How many nodes to visit?

Can we do better? How?



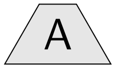
Efficiency/complexity:

- Exhaustive DFS
- Time $O(b^m)$
- Space $O(bm)$

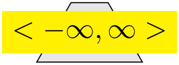
Chess $b \approx 35, m \approx 100 \ldots$

- We cannot go(dive) to the end
- Can we save something?

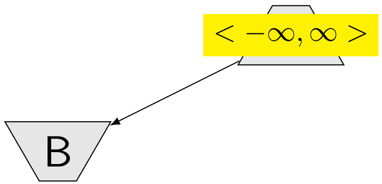
# Nodes (sub-trees) worth visiting



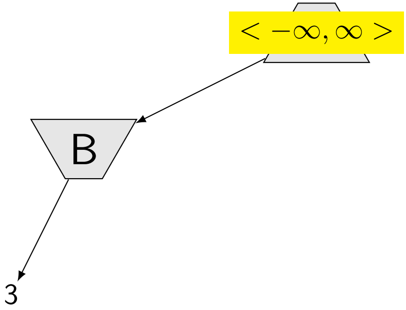
# Nodes (sub-trees) worth visiting



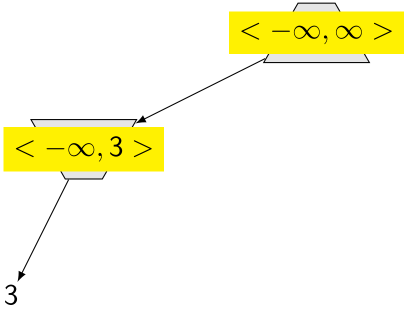
# Nodes (sub-trees) worth visiting



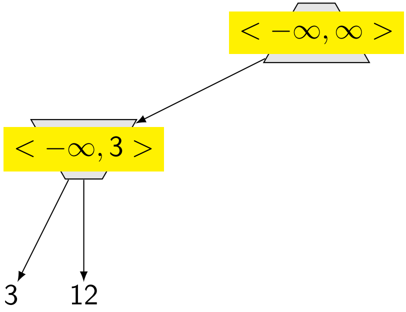
# Nodes (sub-trees) worth visiting



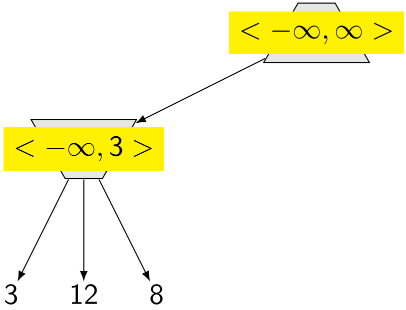
# Nodes (sub-trees) worth visiting



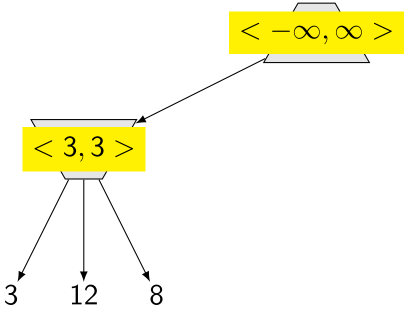
# Nodes (sub-trees) worth visiting



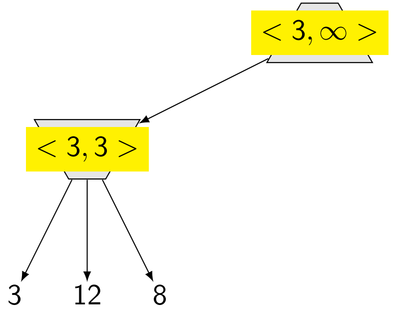
# Nodes (sub-trees) worth visiting



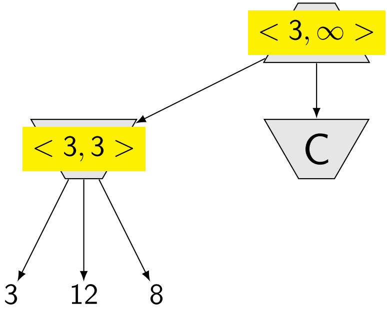
# Nodes (sub-trees) worth visiting



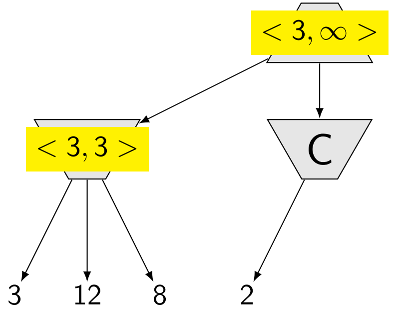
# Nodes (sub-trees) worth visiting



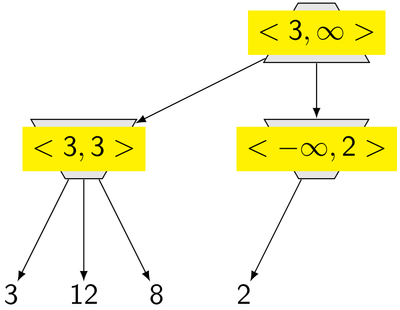
# Nodes (sub-trees) worth visiting



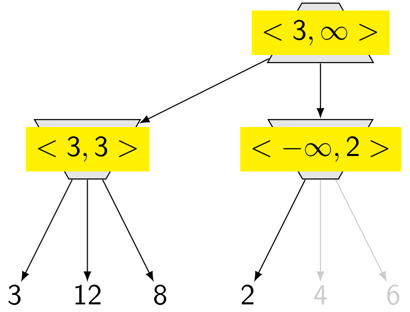
# Nodes (sub-trees) worth visiting



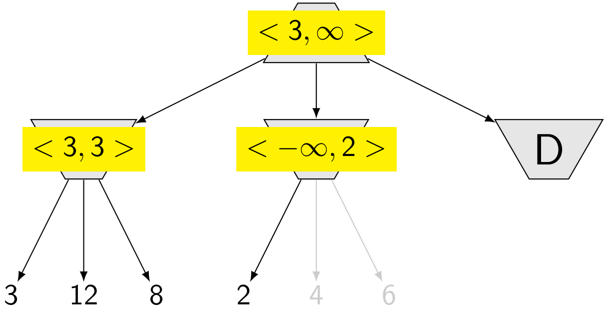
# Nodes (sub-trees) worth visiting



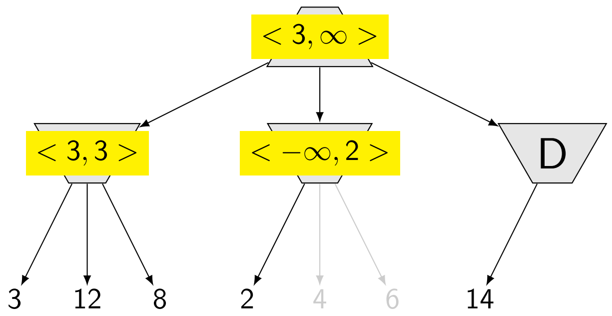
# Nodes (sub-trees) worth visiting



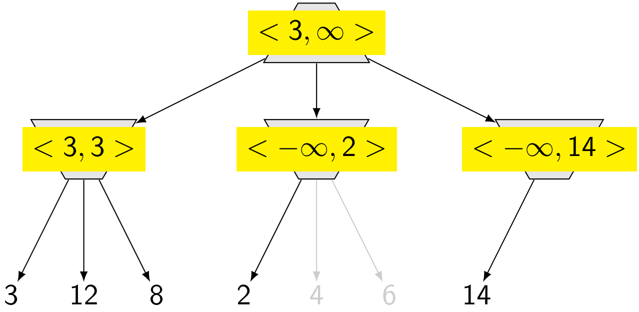
# Nodes (sub-trees) worth visiting



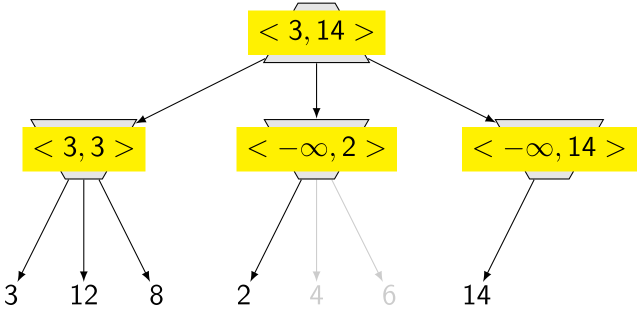
# Nodes (sub-trees) worth visiting



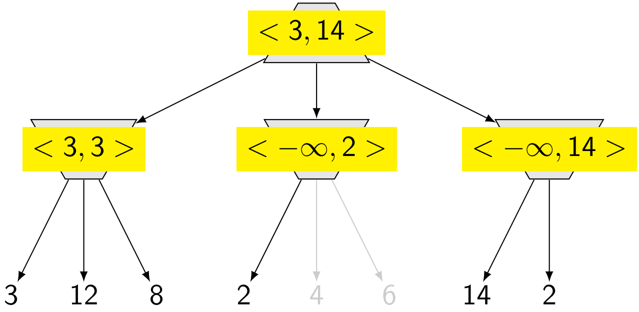
# Nodes (sub-trees) worth visiting



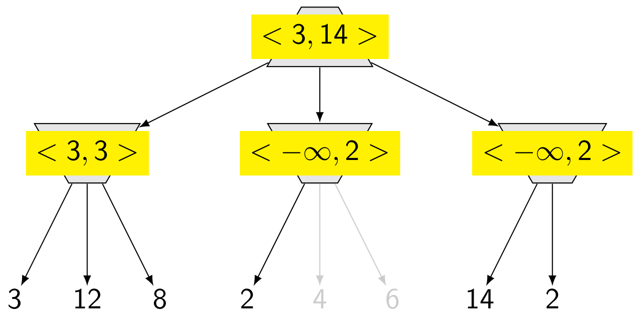
# Nodes (sub-trees) worth visiting



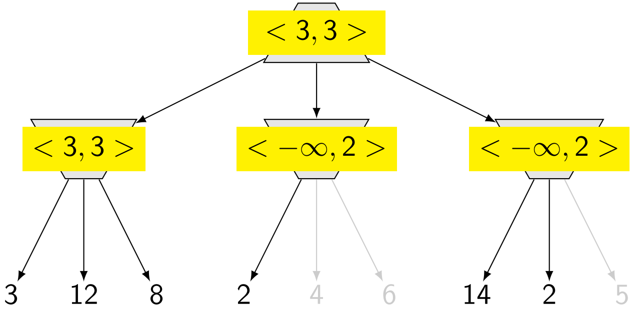
# Nodes (sub-trees) worth visiting



# Nodes (sub-trees) worth visiting



# Nodes (sub-trees) worth visiting



# $\alpha$-$\beta$ prunning

$\alpha$ highest (best) value choice found so far for any choice along MAX
$\beta$ lowest (best) value choice found so far for any choice along MIN


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



In MAX nodes $\alpha$ is changing and $\beta$ is stopping, in MIN nodes $\beta$ is changing and $\alpha$ is stopping.

It is clear that ordering of child nodes matters. Draw tree of $\alpha$-$\beta$ search in case of perferct ordering. Effective branching factor becomes $\sqrt{b}$ instead of $b$ which effectively doubles the depth can be searched.

# $\alpha$-$\beta$ prunning

$\alpha$ highest (best) value choice found so far for any choice along MAX
$\beta$ lowest (best) value choice found so far for any choice along MIN

$\alpha = -\infty, \beta = \infty$

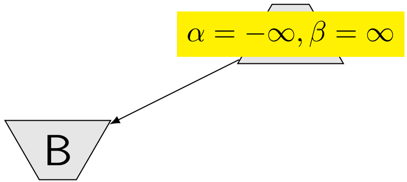
In MIN-VAL: $v \leftarrow 2$
$v \leq \alpha$ then: return $v$!



In MAX nodes $\alpha$ is changing and $\beta$ is stopping, in MIN nodes $\beta$ is changing and $\alpha$ is stopping.
It is clear that ordering of child nodes matters. Draw tree of $\alpha$-$\beta$ search in case of perferct ordering. Effective branching factor becomes $\sqrt{b}$ instead of $b$ which effectively doubles the depth can be searched.

# $\alpha$-$\beta$ prunning

$\alpha$ highest (best) value choice found so far for any choice along MAX
$\beta$ lowest (best) value choice found so far for any choice along MIN

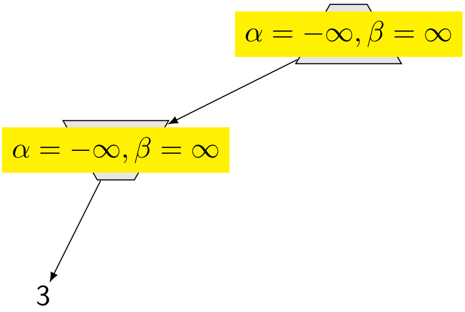



In MIN-VAL: $v \leftarrow 2$
$v \leq \alpha$ then: return $v$!

In MAX nodes $\alpha$ is changing and $\beta$ is stopping, in MIN nodes $\beta$ is changing and $\alpha$ is stopping.
It is clear that ordering of child nodes matters. Draw tree of $\alpha$-$\beta$ search in case of perferct ordering. Effective branching factor becomes $\sqrt{b}$ instead of $b$ which effectively doubles the depth can be searched.

# $\alpha$-$\beta$ prunning

$\alpha$ highest (best) value choice found so far for any choice along MAX
$\beta$ lowest (best) value choice found so far for any choice along MIN



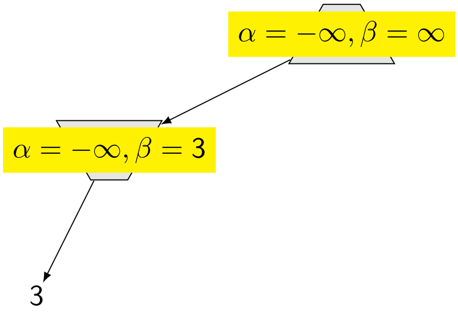

In MIN-VAL: $v \leftarrow 2$
$v \leq \alpha$ then: return $v$!



In MAX nodes $\alpha$ is changing and $\beta$ is stopping, in MIN nodes $\beta$ is changing and $\alpha$ is stopping.
It is clear that ordering of child nodes matters. Draw tree of $\alpha$-$\beta$ search in case of perferct ordering. Effective branching factor becomes $\sqrt{b}$ instead of $b$ which effectively doubles the depth can be searched.

# $\alpha$-$\beta$ prunning

$\alpha$ highest (best) value choice found so far for any choice along MAX
$\beta$ lowest (best) value choice found so far for any choice along MIN



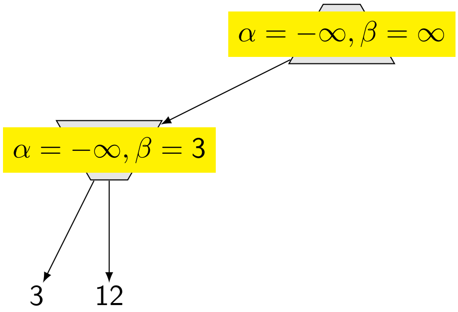

In MIN-VAL: $v \leftarrow 2$
$v \leq \alpha$ then: return $v$!



In MAX nodes $\alpha$ is changing and $\beta$ is stopping, in MIN nodes $\beta$ is changing and $\alpha$ is stopping.
It is clear that ordering of child nodes matters. Draw tree of $\alpha$-$\beta$ search in case of perferct ordering. Effective branching factor becomes $\sqrt{b}$ instead of $b$ which effectively doubles the depth can be searched.

# $\alpha$-$\beta$ prunning

$\alpha$ highest (best) value choice found so far for any choice along MAX
$\beta$ lowest (best) value choice found so far for any choice along MIN



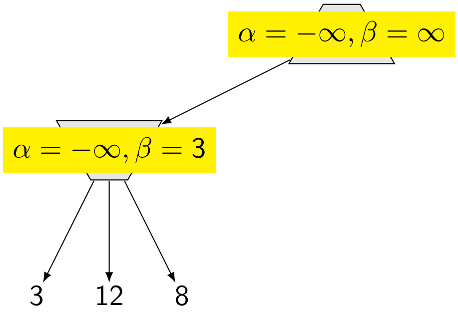

In MIN-VAL: $v \leftarrow 2$
$v \leq \alpha$ then: return $v$!



In MAX nodes $\alpha$ is changing and $\beta$ is stopping, in MIN nodes $\beta$ is changing and $\alpha$ is stopping.
It is clear that ordering of child nodes matters. Draw tree of $\alpha$-$\beta$ search in case of perferct ordering. Effective branching factor becomes $\sqrt{b}$ instead of $b$ which effectively doubles the depth can be searched.

# $\alpha$-$\beta$ prunning

$\alpha$ highest (best) value choice found so far for any choice along MAX
$\beta$ lowest (best) value choice found so far for any choice along MIN




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

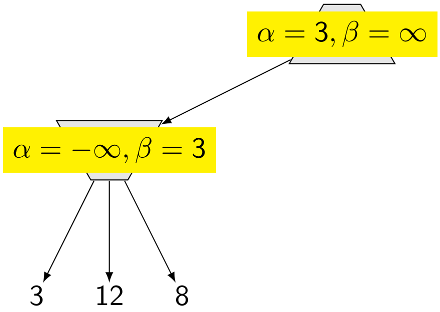


In MAX nodes $\alpha$ is changing and $\beta$ is stopping, in MIN nodes $\beta$ is changing and $\alpha$ is stopping.

It is clear that ordering of child nodes matters. Draw tree of $\alpha$-$\beta$ search in case of perferct ordering. Effective branching factor becomes $\sqrt{b}$ instead of $b$ which effectively doubles the depth can be searched.

# $\alpha$-$\beta$ prunning

$\alpha$ highest (best) value choice found so far for any choice along MAX
$\beta$ lowest (best) value choice found so far for any choice along MIN



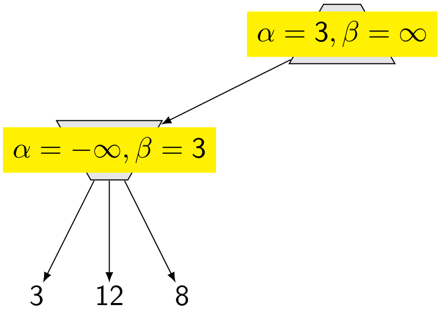

In MIN-VAL: $v \leftarrow 2$
$v \leq \alpha$ then: return $v$!



In MAX nodes $\alpha$ is changing and $\beta$ is stopping, in MIN nodes $\beta$ is changing and $\alpha$ is stopping.
It is clear that ordering of child nodes matters. Draw tree of $\alpha$-$\beta$ search in case of perferct ordering. Effective branching factor becomes $\sqrt{b}$ instead of $b$ which effectively doubles the depth can be searched.

# $\alpha$-$\beta$ prunning

$\alpha$ highest (best) value choice found so far for any choice along MAX
$\beta$ lowest (best) value choice found so far for any choice along MIN




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

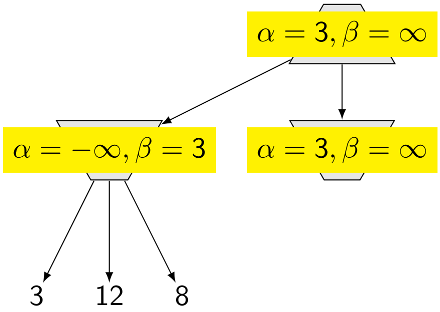


In MAX nodes $\alpha$ is changing and $\beta$ is stopping, in MIN nodes $\beta$ is changing and $\alpha$ is stopping.

It is clear that ordering of child nodes matters. Draw tree of $\alpha$-$\beta$ search in case of perferct ordering. Effective branching factor becomes $\sqrt{b}$ instead of $b$ which effectively doubles the depth can be searched.

# $\alpha$-$\beta$ prunning

$\alpha$ highest (best) value choice found so far for any choice along MAX
$\beta$ lowest (best) value choice found so far for any choice along MIN



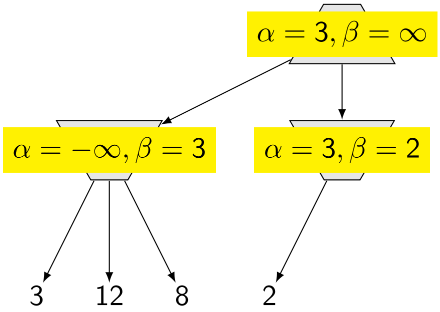

In MIN-VAL: $v \leftarrow 2$
$v \leq \alpha$ then: return $v$!



In MAX nodes $\alpha$ is changing and $\beta$ is stopping, in MIN nodes $\beta$ is changing and $\alpha$ is stopping.
It is clear that ordering of child nodes matters. Draw tree of $\alpha$-$\beta$ search in case of perferct ordering. Effective branching factor becomes $\sqrt{b}$ instead of $b$ which effectively doubles the depth can be searched.

# $\alpha$-$\beta$ prunning

$\alpha$ highest (best) value choice found so far for any choice along MAX
$\beta$ lowest (best) value choice found so far for any choice along MIN



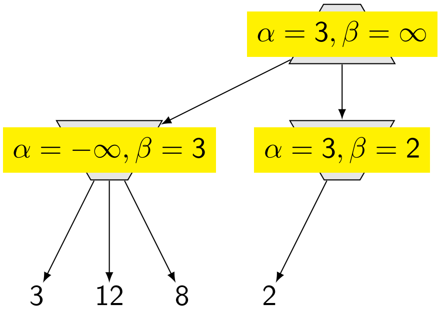

In MIN-VAL: $v \leftarrow 2$
$v \leq \alpha$ then: return $v$!



In MAX nodes $\alpha$ is changing and $\beta$ is stopping, in MIN nodes $\beta$ is changing and $\alpha$ is stopping.
It is clear that ordering of child nodes matters. Draw tree of $\alpha$-$\beta$ search in case of perferct ordering. Effective branching factor becomes $\sqrt{b}$ instead of $b$ which effectively doubles the depth can be searched.

# $\alpha$-$\beta$ prunning

$\alpha$ highest (best) value choice found so far for any choice along MAX
$\beta$ lowest (best) value choice found so far for any choice along MIN




In MIN-VAL: $v \leftarrow 2$

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

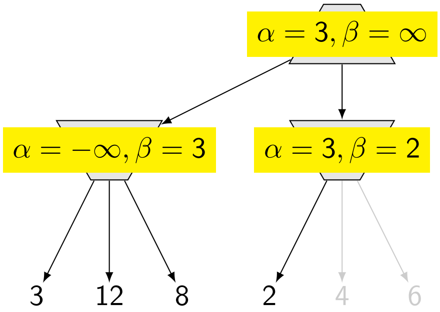


In MAX nodes $\alpha$ is changing and $\beta$ is stopping, in MIN nodes $\beta$ is changing and $\alpha$ is stopping.

It is clear that ordering of child nodes matters. Draw tree of $\alpha$-$\beta$ search in case of perfect ordering. Effective branching factor becomes $\sqrt{b}$ instead of $b$ which effectively doubles the depth can be searched.

# $\alpha$-$\beta$ prunning

$\alpha$ highest (best) value choice found so far for any choice along MAX
$\beta$ lowest (best) value choice found so far for any choice along MIN




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

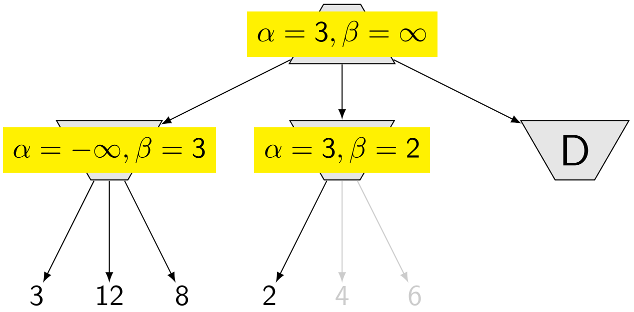


In MAX nodes $\alpha$ is changing and $\beta$ is stopping, in MIN nodes $\beta$ is changing and $\alpha$ is stopping.

It is clear that ordering of child nodes matters. Draw tree of $\alpha$-$\beta$ search in case of perferct ordering. Effective branching factor becomes $\sqrt{b}$ instead of $b$ which effectively doubles the depth can be searched.

# $\alpha$-$\beta$ prunning

$\alpha$ highest (best) value choice found so far for any choice along MAX
$\beta$ lowest (best) value choice found so far for any choice along MIN

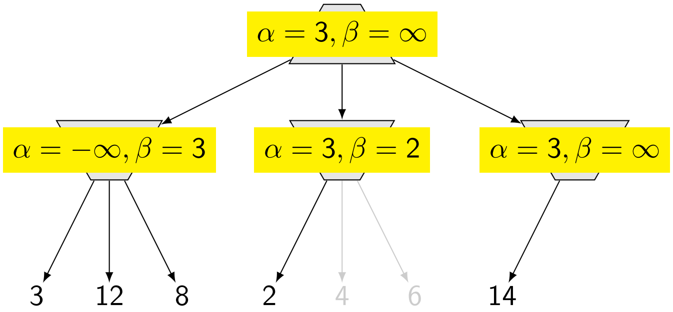



In MIN-VAL: $v \leftarrow 2$
$v \leq \alpha$ then: return $v$!

In MAX nodes $\alpha$ is changing and $\beta$ is stopping, in MIN nodes $\beta$ is changing and $\alpha$ is stopping.
It is clear that ordering of child nodes matters. Draw tree of $\alpha$-$\beta$ search in case of perferct ordering. Effective branching factor becomes $\sqrt{b}$ instead of $b$ which effectively doubles the depth can be searched.

# $\alpha$-$\beta$ prunning

$\alpha$ highest (best) value choice found so far for any choice along MAX
$\beta$ lowest (best) value choice found so far for any choice along MIN



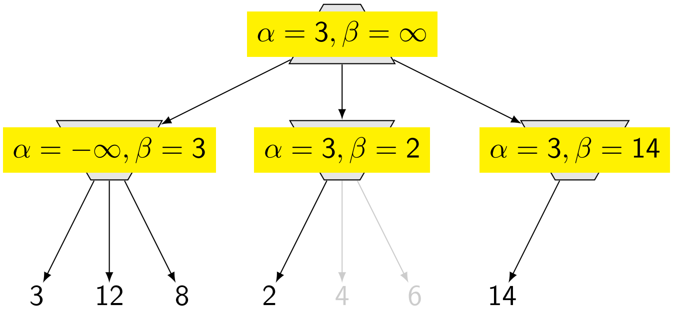

In MIN-VAL: $v \leftarrow 2$
$v \leq \alpha$ then: return $v$!



In MAX nodes $\alpha$ is changing and $\beta$ is stopping, in MIN nodes $\beta$ is changing and $\alpha$ is stopping.
It is clear that ordering of child nodes matters. Draw tree of $\alpha$-$\beta$ search in case of perferct ordering. Effective branching factor becomes $\sqrt{b}$ instead of $b$ which effectively doubles the depth can be searched.

# $\alpha$-$\beta$ prunning

$\alpha$ highest (best) value choice found so far for any choice along MAX
$\beta$ lowest (best) value choice found so far for any choice along MIN




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

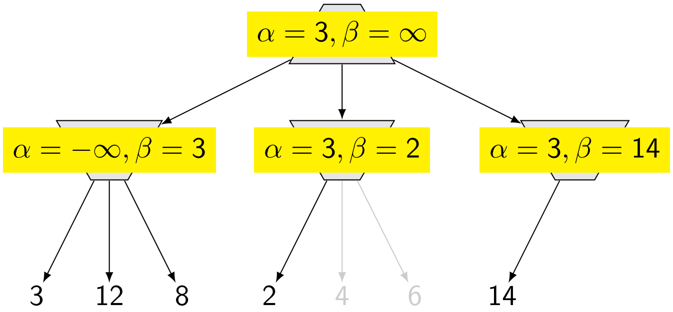


In MAX nodes $\alpha$ is changing and $\beta$ is stopping, in MIN nodes $\beta$ is changing and $\alpha$ is stopping.

It is clear that ordering of child nodes matters. Draw tree of $\alpha$-$\beta$ search in case of perferct ordering. Effective branching factor becomes $\sqrt{b}$ instead of $b$ which effectively doubles the depth can be searched.

# $\alpha$-$\beta$ prunning

$\alpha$ highest (best) value choice found so far for any choice along MAX
$\beta$ lowest (best) value choice found so far for any choice along MIN




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

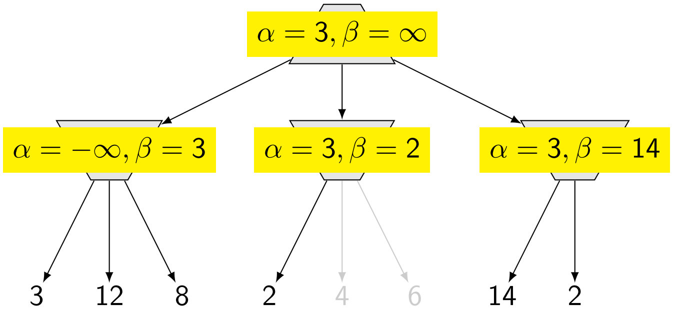


In MAX nodes $\alpha$ is changing and $\beta$ is stopping, in MIN nodes $\beta$ is changing and $\alpha$ is stopping.

It is clear that ordering of child nodes matters. Draw tree of $\alpha$-$\beta$ search in case of perferct ordering. Effective branching factor becomes $\sqrt{b}$ instead of $b$ which effectively doubles the depth can be searched.

# $\alpha$-$\beta$ prunning

$\alpha$ highest (best) value choice found so far for any choice along MAX
$\beta$ lowest (best) value choice found so far for any choice along MIN

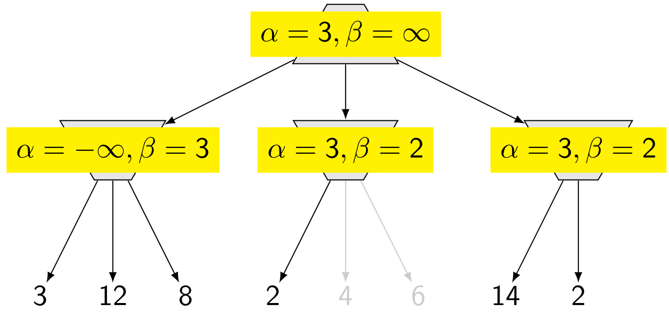



In MIN-VAL: $v \leftarrow 2$
$v \leq \alpha$ then: return $v$!

In MAX nodes $\alpha$ is changing and $\beta$ is stopping, in MIN nodes $\beta$ is changing and $\alpha$ is stopping.
It is clear that ordering of child nodes matters. Draw tree of $\alpha$-$\beta$ search in case of perferct ordering. Effective branching factor becomes $\sqrt{b}$ instead of $b$ which effectively doubles the depth can be searched.

# $\alpha$-$\beta$ prunning

$\alpha$ highest (best) value choice found so far for any choice along MAX
$\beta$ lowest (best) value choice found so far for any choice along MIN



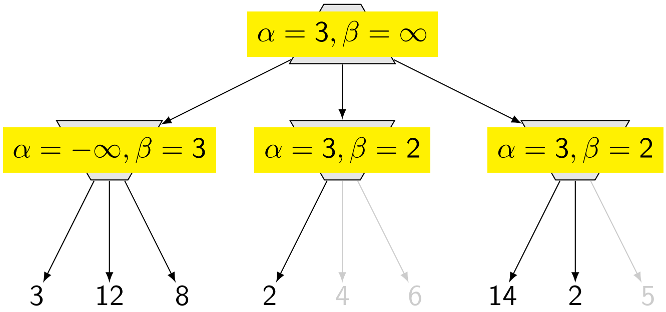

In MIN-VAL: $v \leftarrow 2$
$v \leq \alpha$ then: return $v$!



In MAX nodes $\alpha$ is changing and $\beta$ is stopping, in MIN nodes $\beta$ is changing and $\alpha$ is stopping.
It is clear that ordering of child nodes matters. Draw tree of $\alpha$-$\beta$ search in case of perferct ordering. Effective branching factor becomes $\sqrt{b}$ instead of $b$ which effectively doubles the depth can be searched.

# $\alpha$-$\beta$ prunning

$\alpha$ highest (best) value choice found so far for any choice along MAX
$\beta$ lowest (best) value choice found so far for any choice along MIN




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

In MAX nodes $\alpha$ is changing and $\beta$ is stopping, in MIN nodes $\beta$ is changing and $\alpha$ is stopping.
It is clear that ordering of child nodes matters. Draw tree of $\alpha$-$\beta$ search in case of perferct ordering. Effective branching factor becomes $\sqrt{b}$ instead of $b$ which effectively doubles the depth can be searched.

**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,$\alpha, \beta$) **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,$a$),$\alpha, \beta$))
    **if** $v \geq \beta$ **return** $v$
    $\alpha \leftarrow \max(\alpha, v)$
  **end for**
**end function**

---

**function** MIN-VALUE(state, $\alpha, \beta$) **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,$a$),$\alpha, \beta$))
    **if** $v \leq \alpha$ **return** $v$
    $\beta \leftarrow \min(\beta, v)$
  **end for**
**end function**



Take the tree from the previous slide and try to go step-by-step, watch $\alpha$, $\beta$ and $v$

**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,$\alpha, \beta$) **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,$a$),$\alpha, \beta$))
        **if** $v \geq \beta$ **return** $v$
        $\alpha \leftarrow \max(\alpha, v)$
    **end for**
**end function**

---

**function** MIN-VALUE(state, $\alpha, \beta$) **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,$a$),$\alpha, \beta$))
        **if** $v \leq \alpha$ **return** $v$
        $\beta \leftarrow \min(\beta, v)$
    **end for**
**end function**



Take the tree from the previous slide and try to go step-by-step, watch $\alpha$, $\beta$ and $v$

```
function ALPHA-BETA-SEARCH(state) returns an action
    v ← MAX-VALUE(state, −∞, ∞)
    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 ← −∞
    for all ACTIONS(state) do
        v ← max(v, MIN-VALUE(RESULT(state,a),α, β))
        if v ≥ β return v
        α ← max(α, v)
    end for
end function
```

```
function MIN-VALUE(state, α, β) returns a utility value v
    if TERMINAL-TEST(state) return UTILITY(state)
    v ← ∞
    for all ACTIONS(state) do
        v ← min(v, MAX-VALUE(RESULT(state,a),α, β))
        if v ≤ α return v
        β ← min(β, v)
    end for
end function
```



Take the tree from the previous slide and try to go step-by-step, watch $\alpha$, $\beta$ and $v$

# Imperfect but real-time decisions - iterative deepening

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



Even with perfect ordering, $\alpha$-$\beta$ prunning does not save us.

# Imperfect but real-time decisions - iterative deepening

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



Even with perfect ordering, $\alpha$-$\beta$ prunning does not save us.

# Imperfect but real-time decisions - iterative deepening

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



Even with perfect ordering, $\alpha$-$\beta$ prunning does not save us.

# Imperfect but real-time decisions - iterative deepening

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



Even with perfect ordering, $\alpha$-$\beta$ prunning does not save us.

# Cutting off search

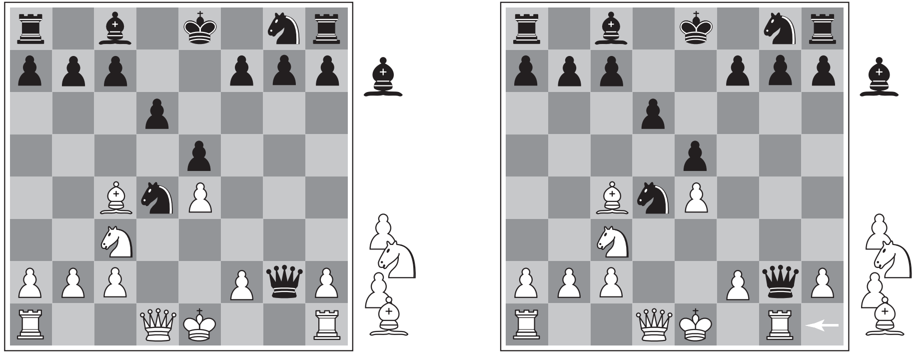
Replace
**if** TERMINAL-TEST(s) **then return** UTILITY(s)
with:
**if** CUTOFF-TEST(s,d) **then return** EVAL(s)



Cutting depends on *d* only, why we need *s* as the input parameter?

# EVAL(s) – Evaluation functions

(estimate of) State value for non-terminal states



(a) White to move (b) White to move

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



For many problems it is not so easy to find/construct proper function. We may try more functions and combine them conveniently.

$$f_1(s) = \text{number of white pawns} - \text{number of black pawns}$$

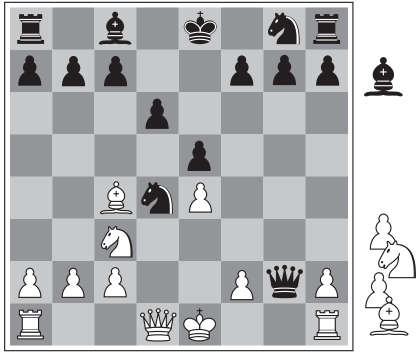
How to tune weights $w_i$?

or Deep Nets! Yeah!

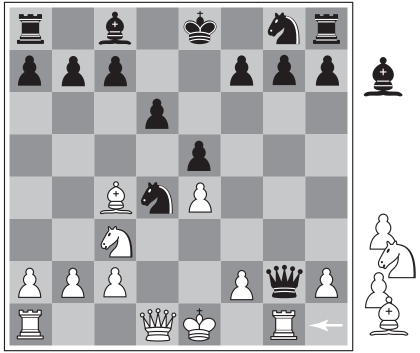
How the get training data for supervised learning? More later.

# EVAL(s) – Evaluation functions

(estimate of) State value for non-terminal states



(a) White to move



(b) White to move

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



For many problems it is not so easy to find/construct proper function. We may try more functions and combine them conveniently.

$$f_1(s) = \text{number of white pawns} - \text{number of black pawns}$$

How to tune weights $w_i$?
or Deep Nets! Yeah!

How the get training data for supervised learning? More later.

## 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.