Artificial intelligence in games
Nullsummenspiele mit vollständiger Information
oder wie spielt ein Computer Schach
Teil II

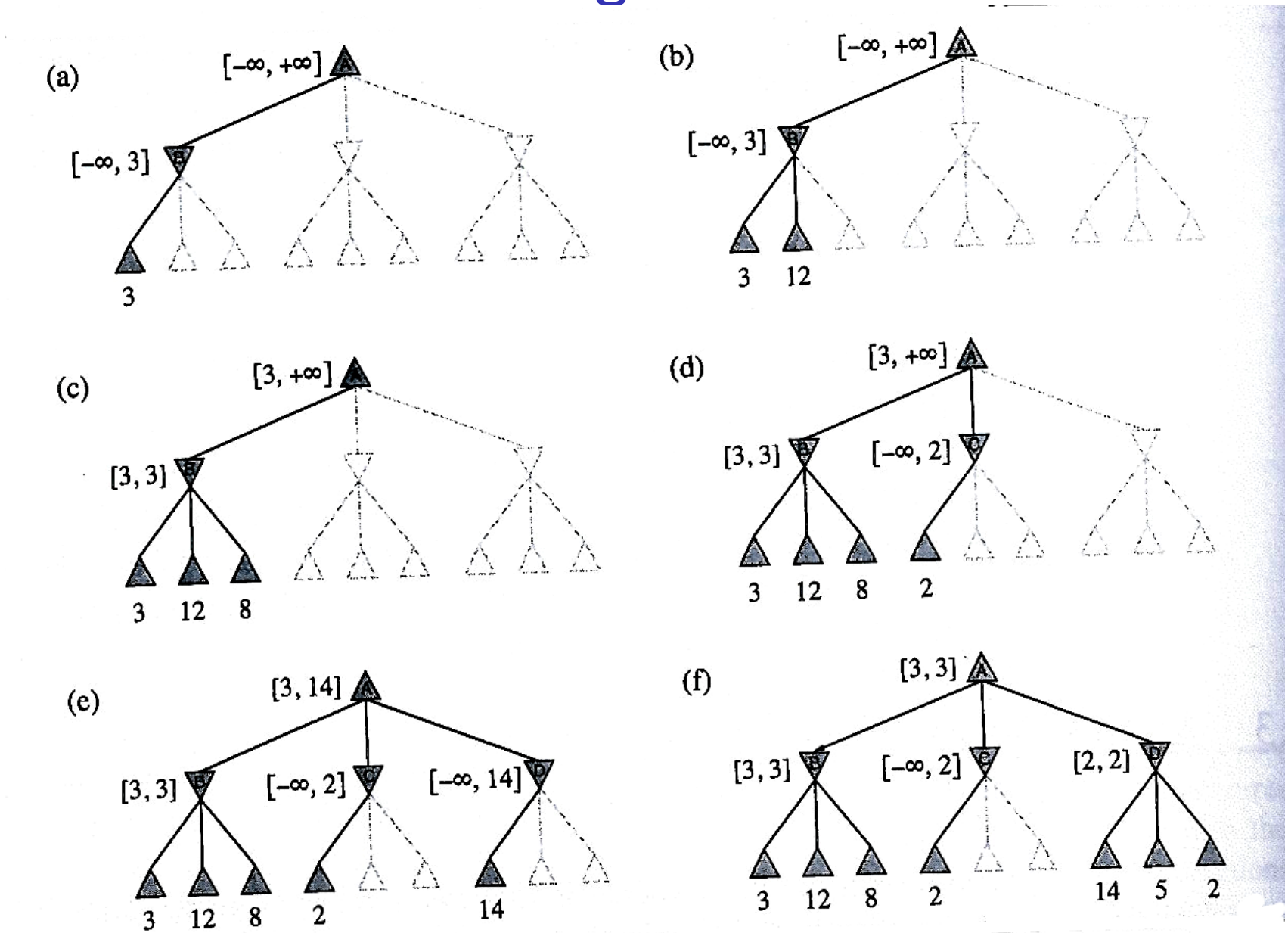
Gregor Lämmel

laemmel@htw-berlin.de

19.05.2011

Outline

ALPHA-BETA Pruning



Quelle: Russel, S. & Norvig, P. Artificial intelligence: A modern approach Prentice Hall, 1995

ALPHA-BETA Pruning in Pseudocode

AlphaBetaSearch(*state*)

$\quad$ *bestUtility* = **MaxValue(***state*, $-\infty$, $+\infty$**)**

return *a* $\in$ Actions(*state*) with *utility* == *bestUtility*

ALPHA-BETA Pruning in Pseudocode

MaxValue($state, \alpha, \beta$)
    if TerminalTest(state) **then**
        return Utility($state$)
    end
    $utility = -\infty$
    for each $a \in$ Actions($state$):
        $tmp =$ **MinValue**(Result($state, a$), α, β)
        $utility =$ **Max**($utility, tmp$)
        if $utility \geq \beta$ **then**
            return $utility$
        end
        $\alpha =$ **Max**($\alpha, utility$)
    end
return utility

ALPHA-BETA Pruning in Pseudocode

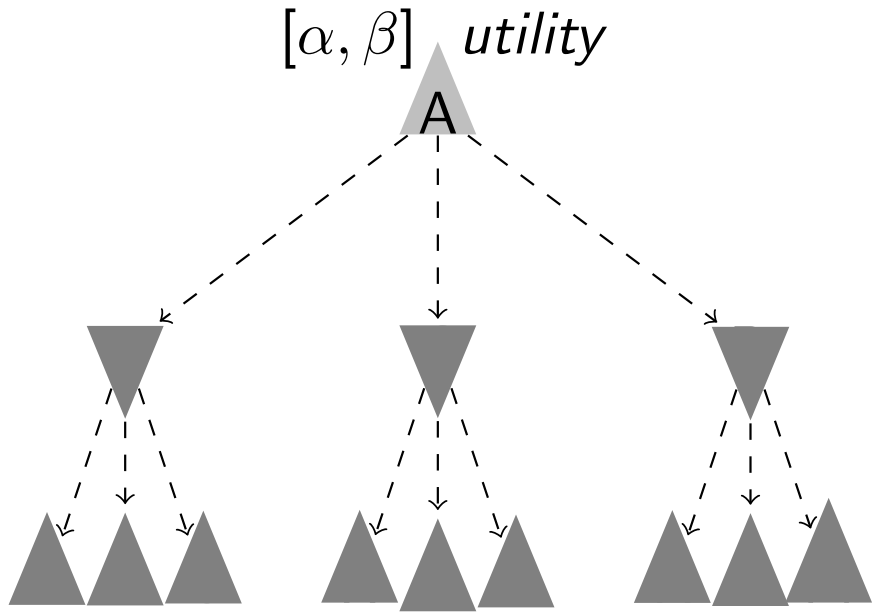
MinValue($state, \alpha, \beta$)
    if TerminalTest(state) **then**
        return Utility($state$)
    end
    $utility = +\infty$
    for each $a \in$ Actions($state$):
        $tmp =$ **MaxValue**(Result($state, a$), α, β)
        $utility =$ **Min**($utility, tmp$)
        if $utility \leq \alpha$ **then**
            return $utility$
        end
        $\beta =$ **Min**($\beta, utility$)
    end
return utility

ALPHA-BETA Pruning – Beispiel

AlphaBetaSearch(*state*)

$bestUtility$ = **MaxValue(***state*, $-\infty, +\infty$**)**

return $a \in \mathrm{Actions}(state)$ with
$utility == bestUtility$



ALPHA-BETA Pruning – Beispiel

MaxValue($state, \alpha, \beta$)

if TerminalTest(state) **then**

return Utility($state$)

end

$utility = -\infty$

for each $a \in$ Actions($state$):

$tmp =$
MinValue(Result($state, a$), α, β)

$utility =$ **Max**($utility, tmp$)

if $utility \geq \beta$ **then**

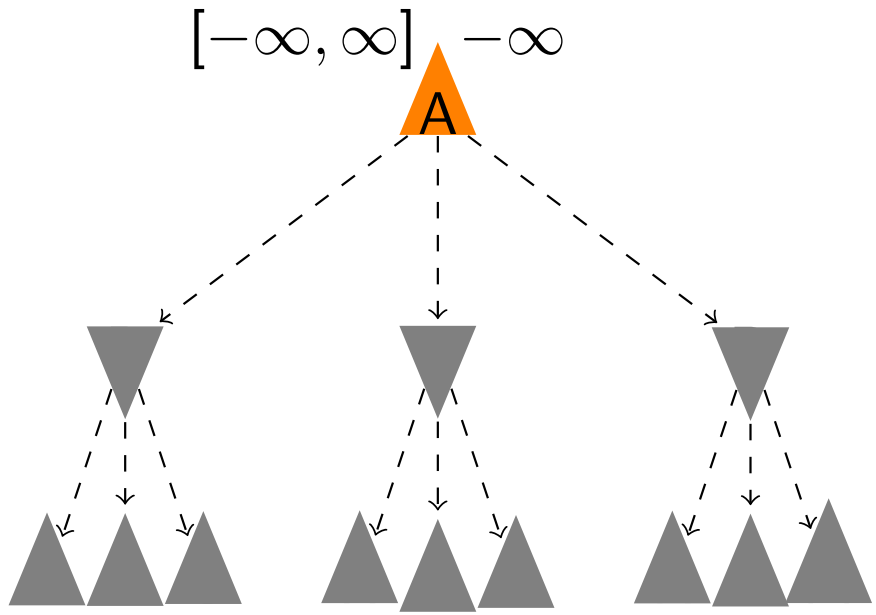
return $utility$

end

$\alpha =$ **Max**($\alpha, utility$)

end

return utility



ALPHA-BETA Pruning – Beispiel

MinValue(*state*, α, β)

  if TerminalTest(state) **then**

    return Utility(*state*)

  end

  utility $= +\infty$

  for each *a* $\in$ Actions(*state*):

    tmp $=$
    MaxValue(Result(*state*, *a*), α, β)

    utility $=$ **Min**(*utility*, *tmp*)

    if *utility* $\leq \alpha$ **then**

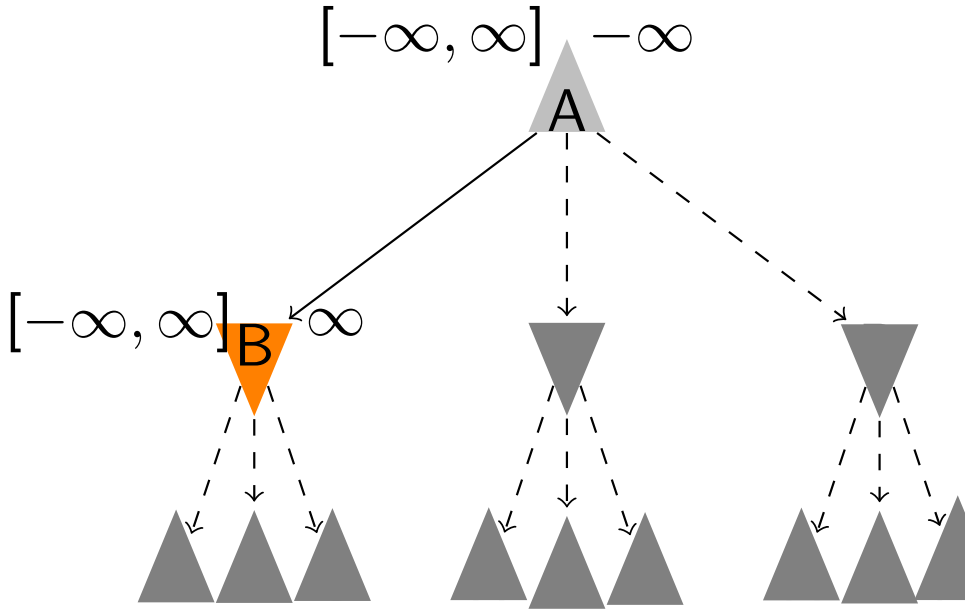
      return *utility*

    end

    $\beta =$ **Min**(β, *utility*)

  end

return utility



ALPHA-BETA Pruning – Beispiel

MaxValue($state, \alpha, \beta$)

if TerminalTest(state) **then**

return Utility($state$)

end

$utility = -\infty$

for each $a \in$ Actions($state$):

$tmp =$
MinValue(Result($state, a$), α, β)

$utility =$ **Max**($utility, tmp$)

if $utility \geq \beta$ **then**

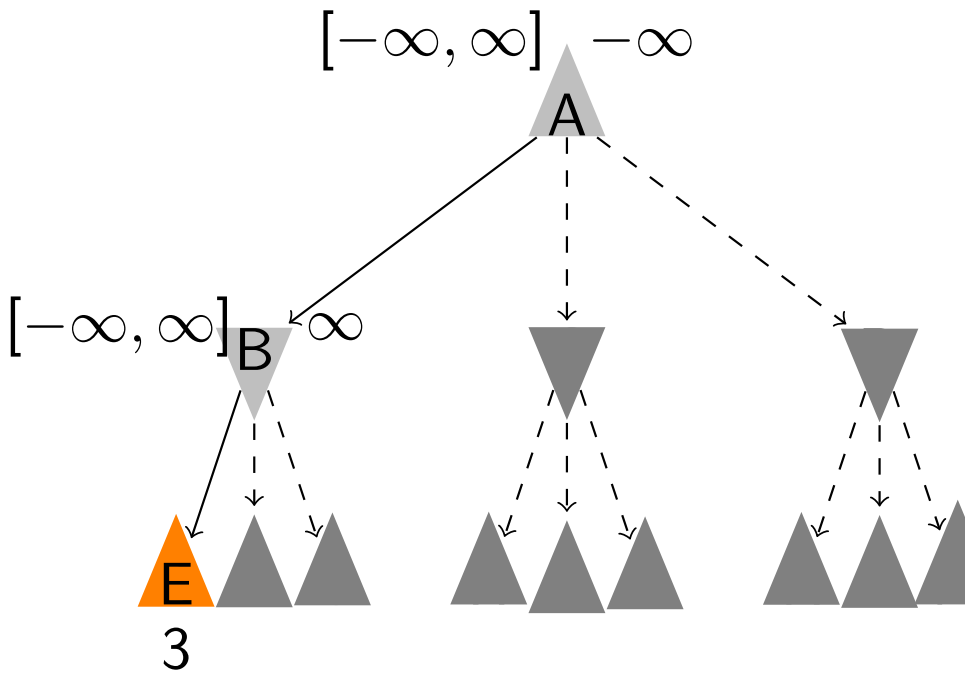
return $utility$

end

$\alpha =$ **Max**($\alpha, utility$)

end

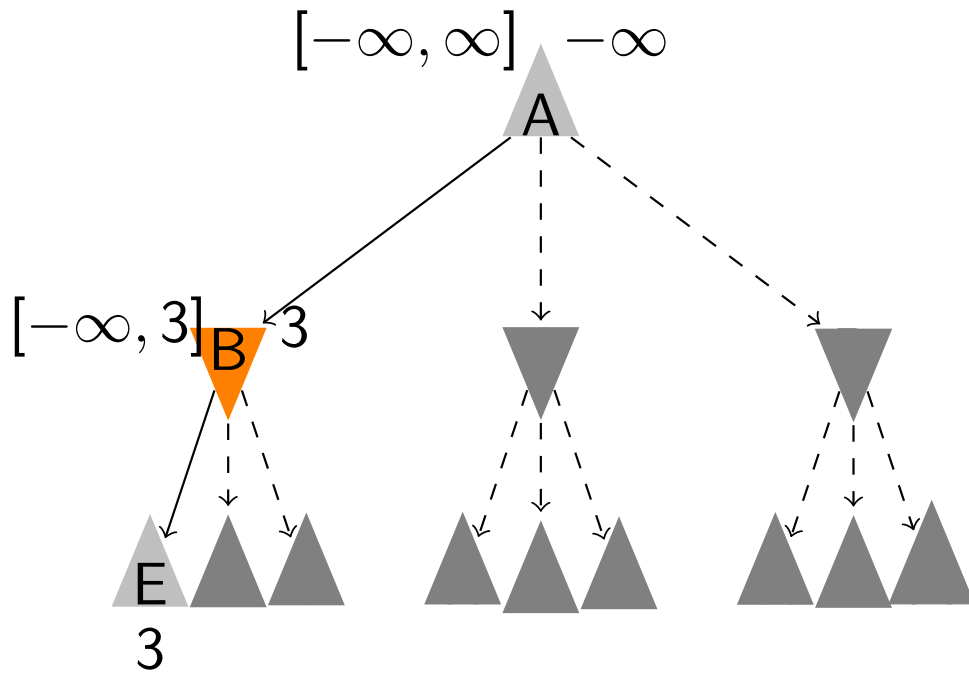
return utility



ALPHA-BETA Pruning – Beispiel

MinValue(*state*, α, β)

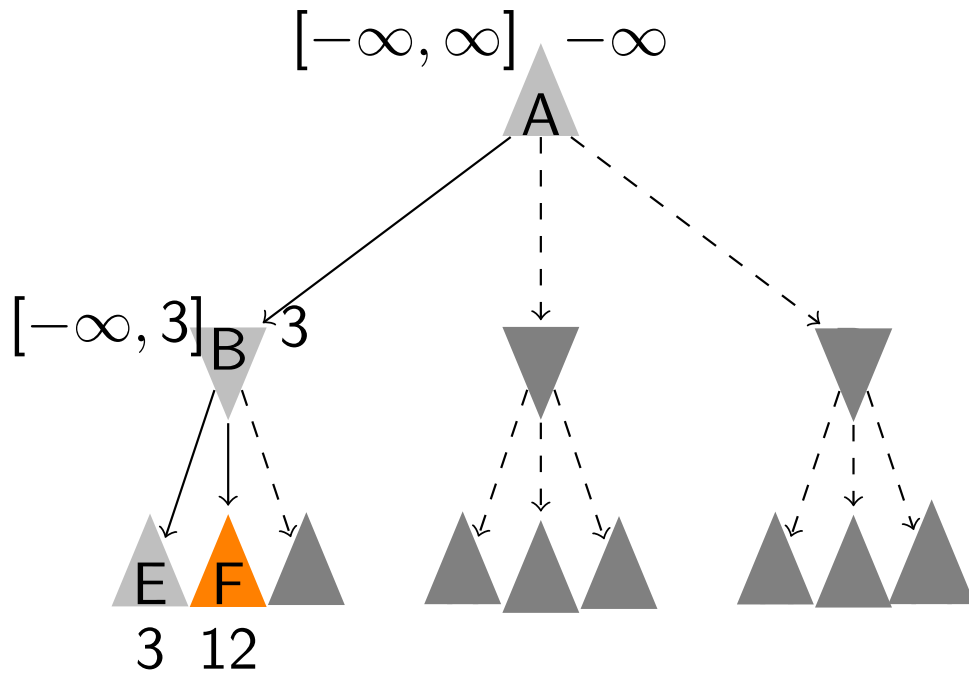
```
    if TerminalTest(state) then
        return Utility(state)
    end
    utility = +∞
    for each a ∈ Actions(state):
        tmp =
        MaxValue(Result(state, a), α, β)
        utility = Min(utility, tmp)
        if utility ≤ α then
            return utility
        end
        β = Min(β, utility)
    end
return utility
```



ALPHA-BETA Pruning – Beispiel

MaxValue($state, \alpha, \beta$)

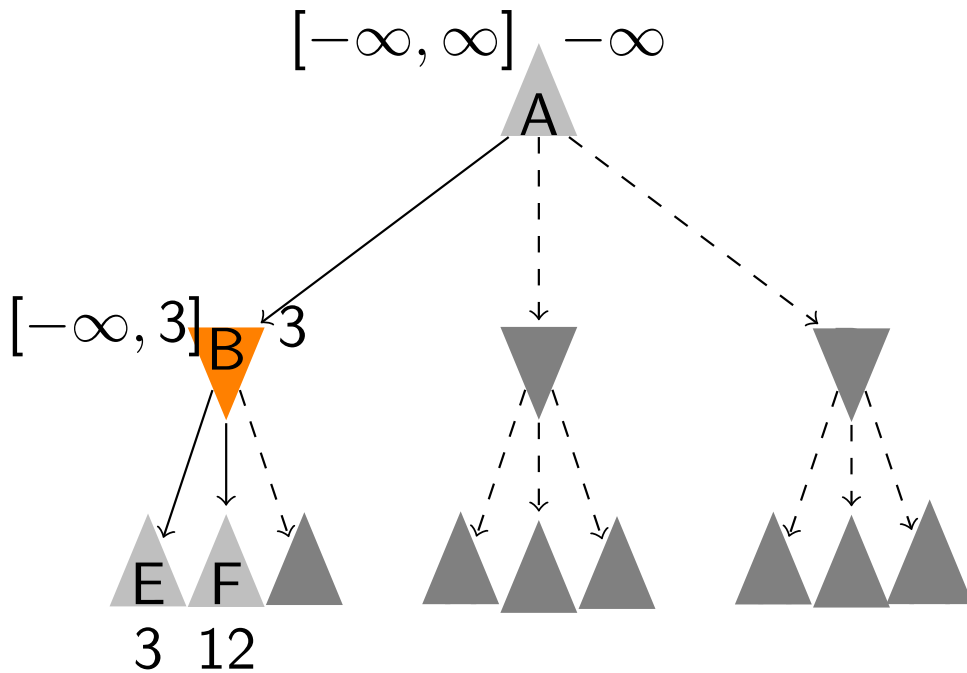
```
    if TerminalTest(state) then
        return Utility(state)
    end
    utility = −∞
    for each a ∈ Actions(state):
        tmp =
        MinValue(Result(state, a), α, β)
        utility = Max(utility, tmp)
        if utility ≥ β then
            return utility
        end
        α = Max(α, utility)
    end
return utility
```



ALPHA-BETA Pruning – Beispiel

MinValue(*state*, α, β)

```
    if TerminalTest(state) then
        return Utility(state)
    end
    utility = +∞
    for each a ∈ Actions(state):
        tmp =
        MaxValue(Result(state, a), α, β)
        utility = Min(utility, tmp)
        if utility ≤ α then
            return utility
        end
        β = Min(β, utility)
    end
return utility
```



ALPHA-BETA Pruning – Beispiel

MaxValue($state, \alpha, \beta$)

if TerminalTest(state) **then**

return Utility($state$)

end

$utility = -\infty$

for each $a \in$ Actions($state$):

$tmp =$
MinValue(Result($state, a$), α, β)

$utility =$ **Max**($utility, tmp$)

if $utility \geq \beta$ **then**

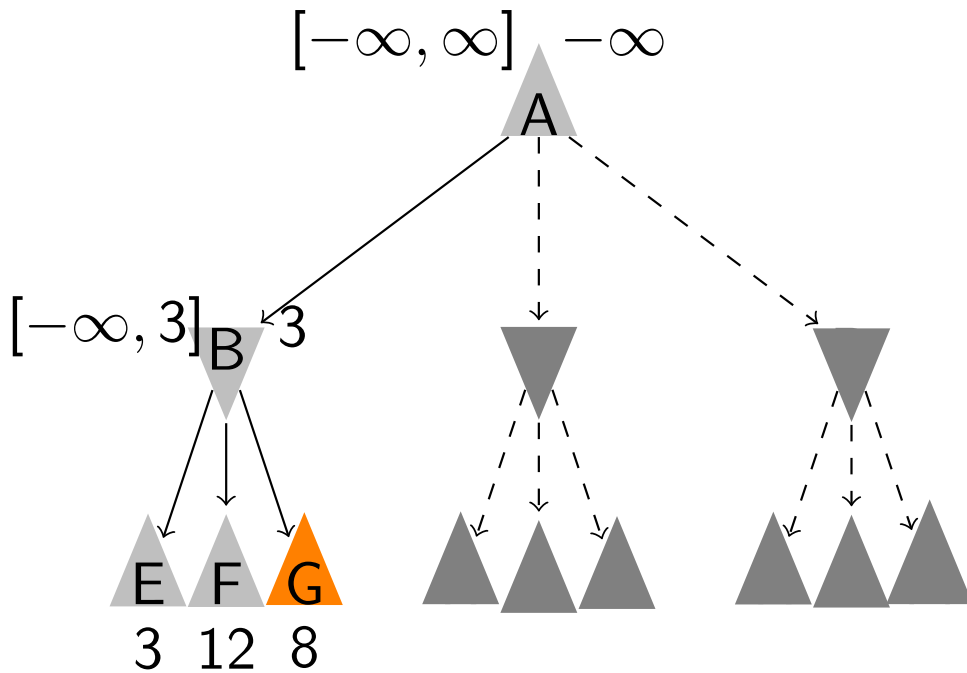
return $utility$

end

$\alpha =$ **Max**($\alpha, utility$)

end

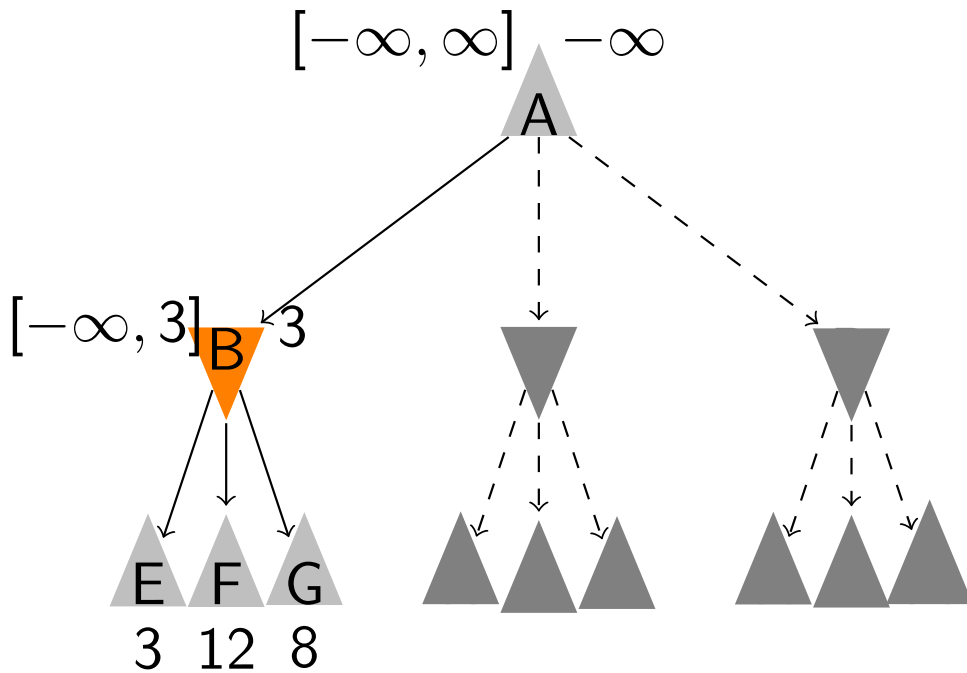
return utility



ALPHA-BETA Pruning – Beispiel

MinValue(*state*, α, β)

```
    if TerminalTest(state) then
        return Utility(state)
    end
    utility = +∞
    for each a ∈ Actions(state):
        tmp =
        MaxValue(Result(state, a), α, β)
        utility = Min(utility, tmp)
        if utility ≤ α then
            return utility
        end
        β = Min(β, utility)
    end
return utility
```



ALPHA-BETA Pruning – Beispiel

MaxValue($state, \alpha, \beta$)

if TerminalTest(state) **then**

return Utility($state$)

end

$utility = -\infty$

for each $a \in$ Actions($state$):

$tmp =$
MinValue(Result($state, a$), α, β)

$utility =$ **Max**($utility, tmp$)

if $utility \geq \beta$ **then**

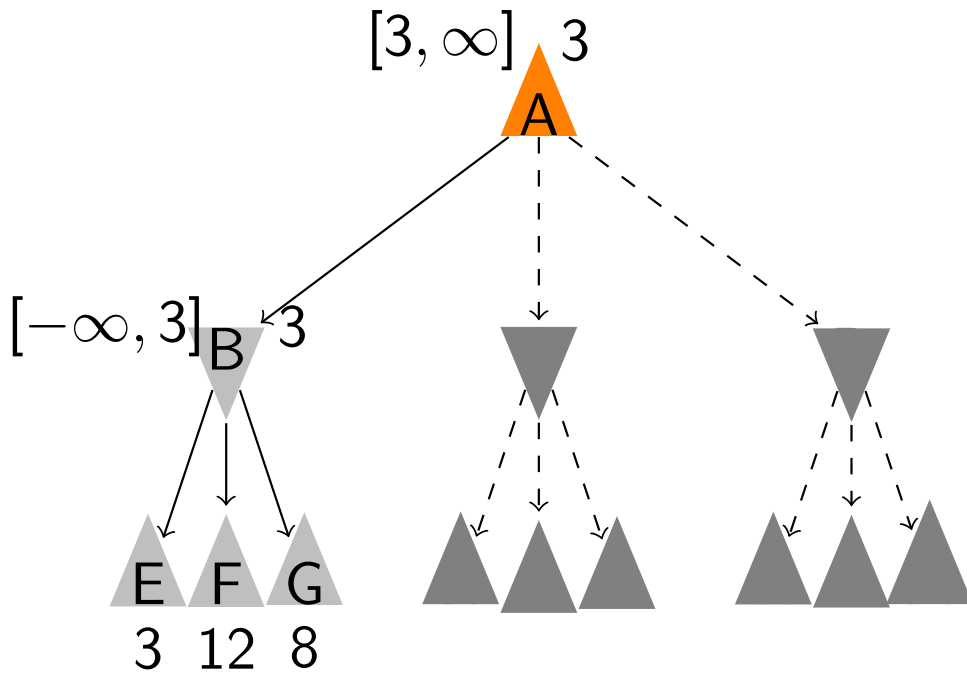
return $utility$

end

$\alpha =$ **Max**($\alpha, utility$)

end

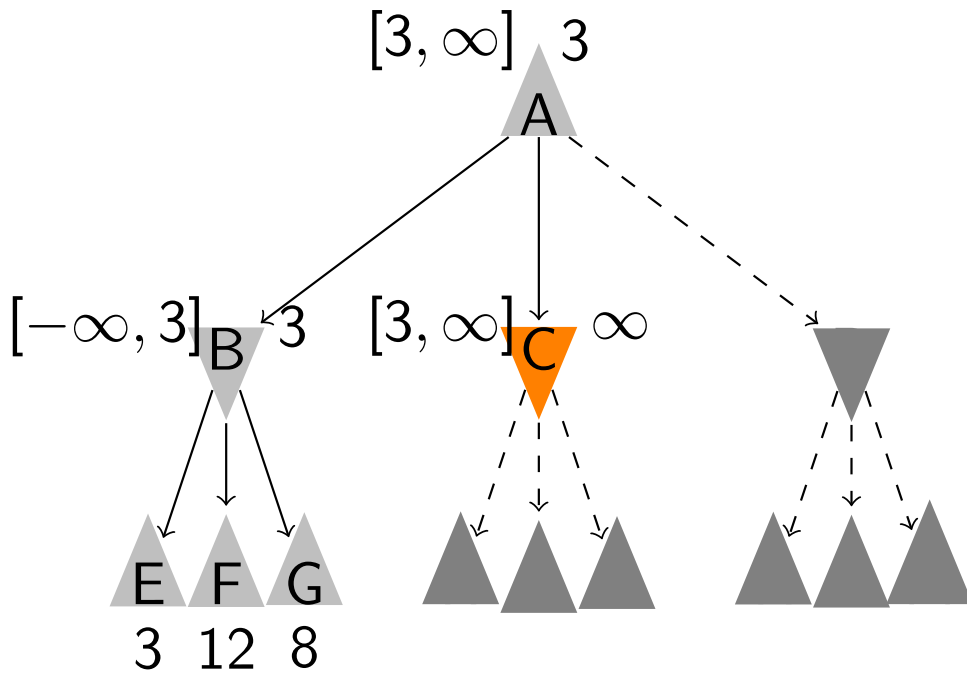
return utility



ALPHA-BETA Pruning – Beispiel

MinValue($state, \alpha, \beta$)

    if TerminalTest(state) **then**
        return Utility($state$)
    end
    $utility = +\infty$
    for each $a \in$ Actions($state$):
        $tmp =$
        MaxValue(Result($state, a$), α, β)
        $utility =$ **Min**($utility, tmp$)
        if $utility \leq \alpha$ **then**
            return $utility$
        end
        $\beta =$ **Min**($\beta, utility$)
    end
return utility



ALPHA-BETA Pruning – Beispiel

MaxValue($state, \alpha, \beta$)

if TerminalTest(state) **then**

return Utility($state$)

end

$utility = -\infty$

for each $a \in$ Actions($state$):

$tmp =$
MinValue(Result($state, a$), α, β)

$utility =$ **Max**($utility, tmp$)

if $utility \geq \beta$ **then**

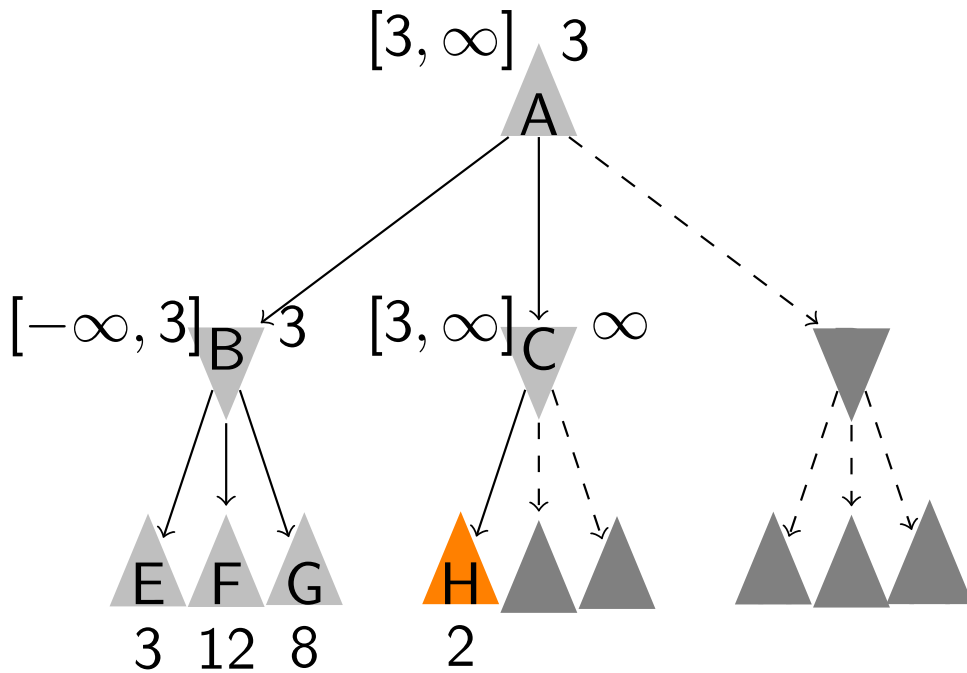
return $utility$

end

$\alpha =$ **Max**($\alpha, utility$)

end

return utility



ALPHA-BETA Pruning – Beispiel

MinValue($state, \alpha, \beta$)

if TerminalTest(state) **then**

return Utility($state$)

end

$utility = +\infty$

for each $a \in$ Actions($state$):

$tmp =$
MaxValue(Result($state, a$), α, β)

$utility =$ **Min**($utility, tmp$)

if $utility \leq \alpha$ **then**

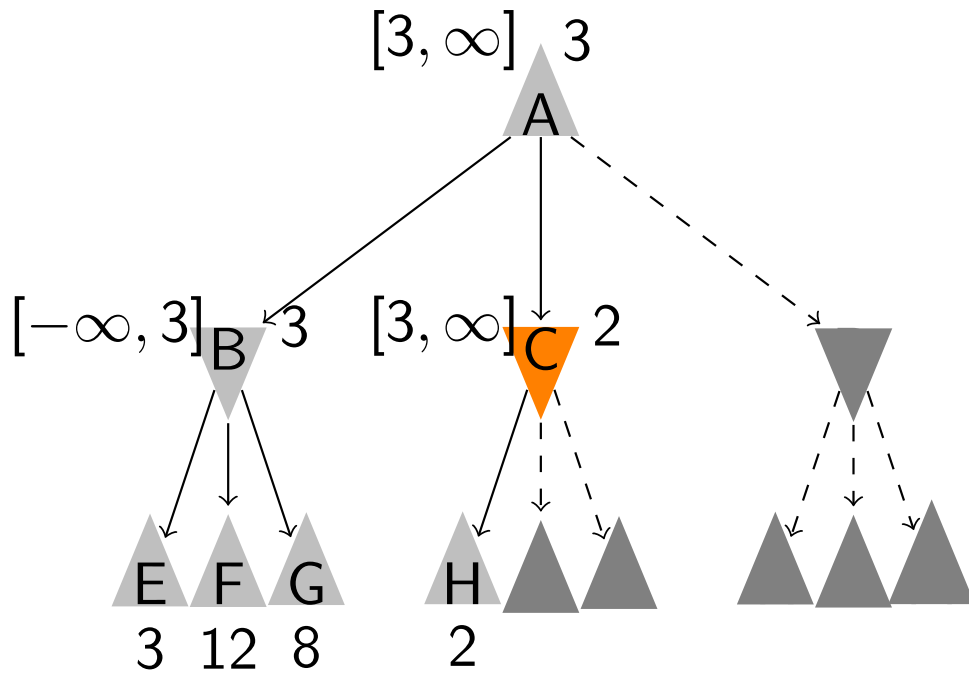
return $utility$

end

$\beta =$ **Min**($\beta, utility$)

end

return utility



ALPHA-BETA Pruning – Beispiel

MinValue($state, \alpha, \beta$)

  if TerminalTest(state) **then**

    return Utility($state$)

  end

  $utility = +\infty$

  for each $a \in$ Actions($state$):

    $tmp =$
    MaxValue(Result($state, a$), α, β)

    $utility =$ **Min**($utility, tmp$)

    if $utility \leq \alpha$ **then**

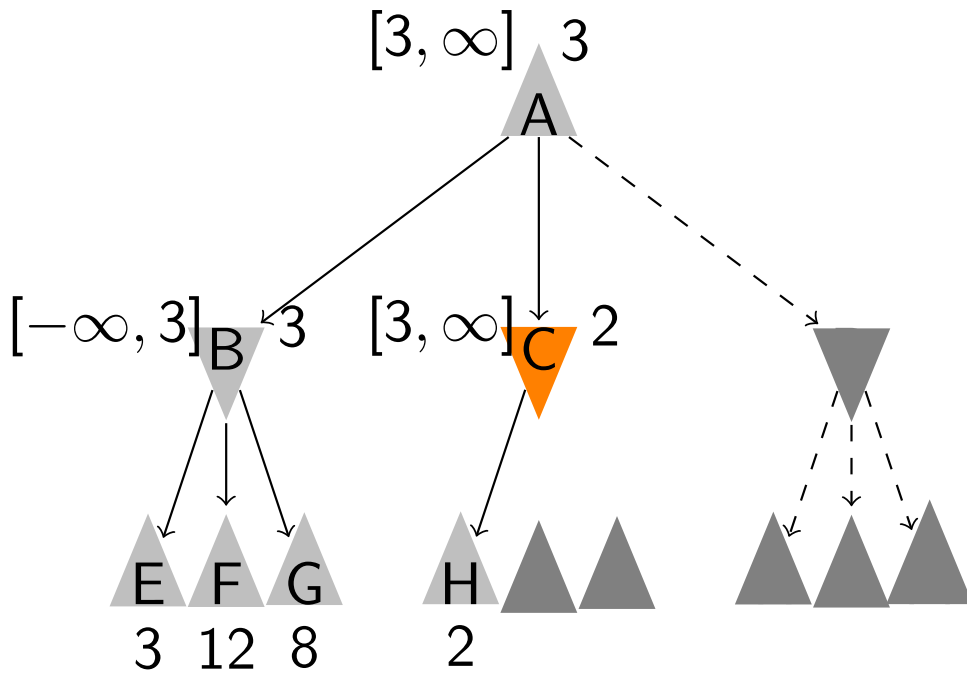
      return $utility$

    end

    $\beta =$ **Min**($\beta, utility$)

  end

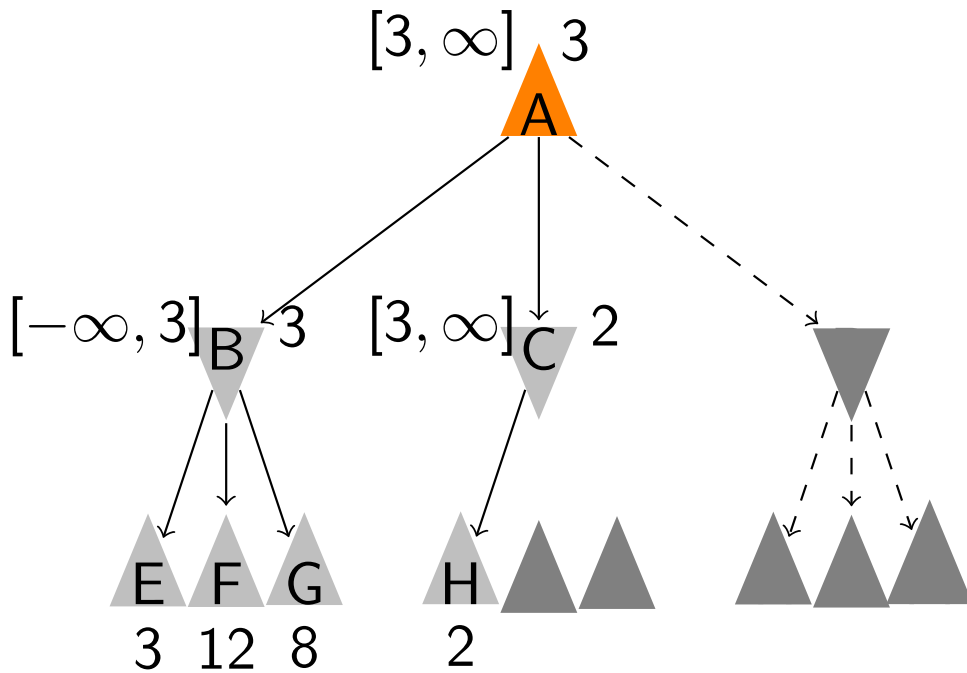
return utility



ALPHA-BETA Pruning – Beispiel

MaxValue($state, \alpha, \beta$)

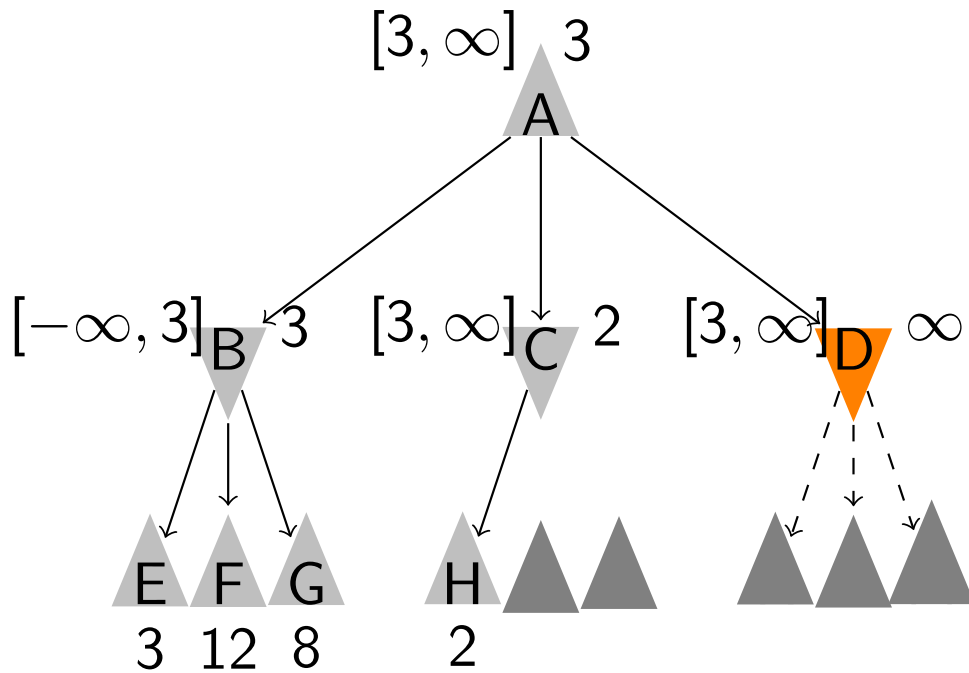
    if TerminalTest(state) **then**
        return Utility($state$)
    end
    $utility = -\infty$
    for each $a \in$ Actions($state$):
        $tmp =$
        MinValue(Result($state, a$), α, β)
        $utility =$ **Max**($utility, tmp$)
        if $utility \geq \beta$ **then**
            return $utility$
        end
        $\alpha =$ **Max**($\alpha, utility$)
    end
return utility



ALPHA-BETA Pruning – Beispiel

MinValue(*state*, α, β)

```
    if TerminalTest(state) then
        return Utility(state)
    end
    utility = +∞
    for each a ∈ Actions(state):
        tmp =
        MaxValue(Result(state, a), α, β)
        utility = Min(utility, tmp)
        if utility ≤ α then
            return utility
        end
        β = Min(β, utility)
    end
return utility
```



ALPHA-BETA Pruning – Beispiel

MaxValue($state, \alpha, \beta$)

if TerminalTest(state) **then**

return Utility($state$)

end

$utility = -\infty$

for each $a \in$ Actions($state$):

$tmp =$
MinValue(Result($state, a$), α, β)

$utility =$ **Max**($utility, tmp$)

if $utility \geq \beta$ **then**

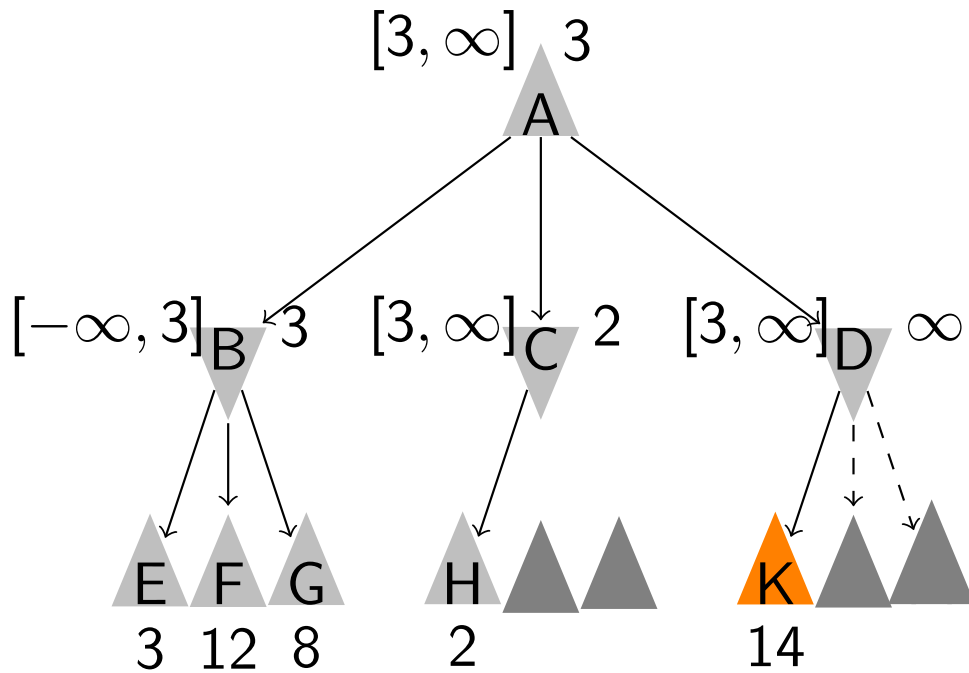
return $utility$

end

$\alpha =$ **Max**($\alpha, utility$)

end

return utility



ALPHA-BETA Pruning – Beispiel

MinValue($state, \alpha, \beta$)

  if TerminalTest(state) **then**

    return Utility($state$)

  end

  $utility = +\infty$

  for each $a \in$ Actions($state$):

    $tmp =$
    MaxValue(Result($state, a$), α, β)

    $utility =$ **Min**($utility, tmp$)

    if $utility \leq \alpha$ **then**

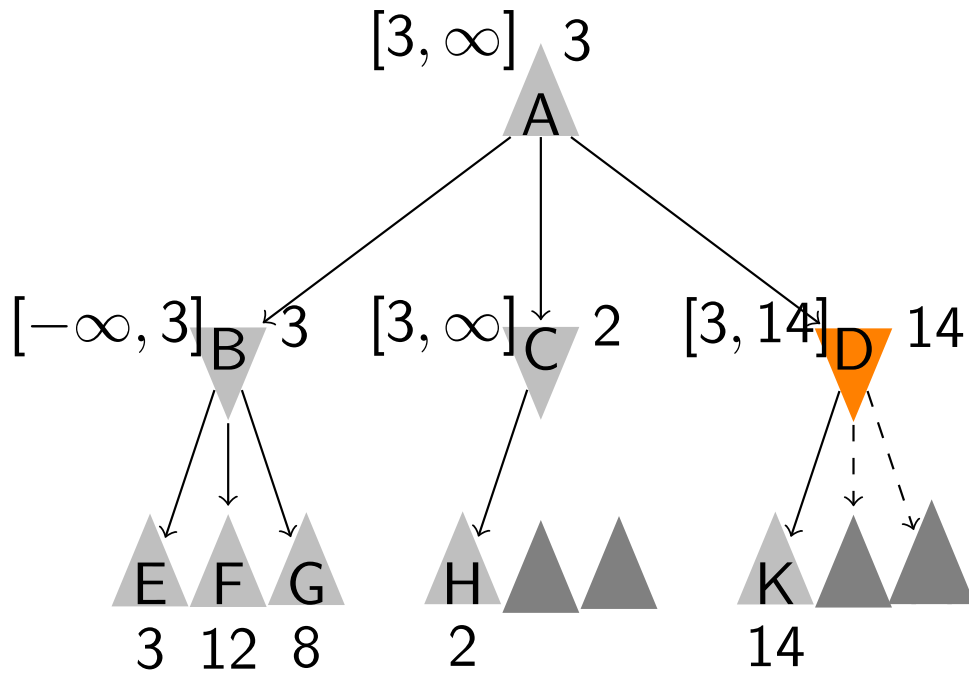
      return $utility$

    end

    $\beta =$ **Min**($\beta, utility$)

  end

return utility



ALPHA-BETA Pruning – Beispiel

MaxValue($state, \alpha, \beta$)

  if TerminalTest(state) **then**

    return Utility($state$)

  end

  $utility = -\infty$

  for each $a \in$ Actions($state$):

    $tmp =$
    MinValue(Result($state, a$), α, β)

    $utility =$ **Max**($utility, tmp$)

    if $utility \geq \beta$ **then**

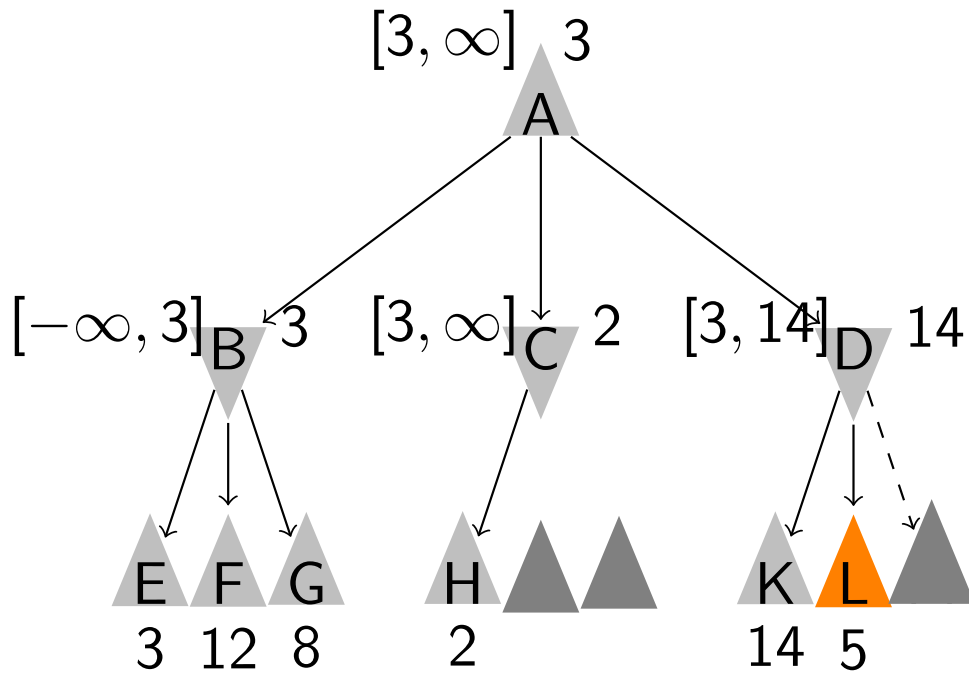
      return $utility$

    end

    $\alpha =$ **Max**($\alpha, utility$)

  end

return utility



ALPHA-BETA Pruning – Beispiel

MinValue($state, \alpha, \beta$)

if TerminalTest(state) **then**

return Utility($state$)

end

$utility = +\infty$

for each $a \in$ Actions($state$):

$tmp =$
MaxValue(Result($state, a$), α, β)

$utility =$ **Min**($utility, tmp$)

if $utility \leq \alpha$ **then**

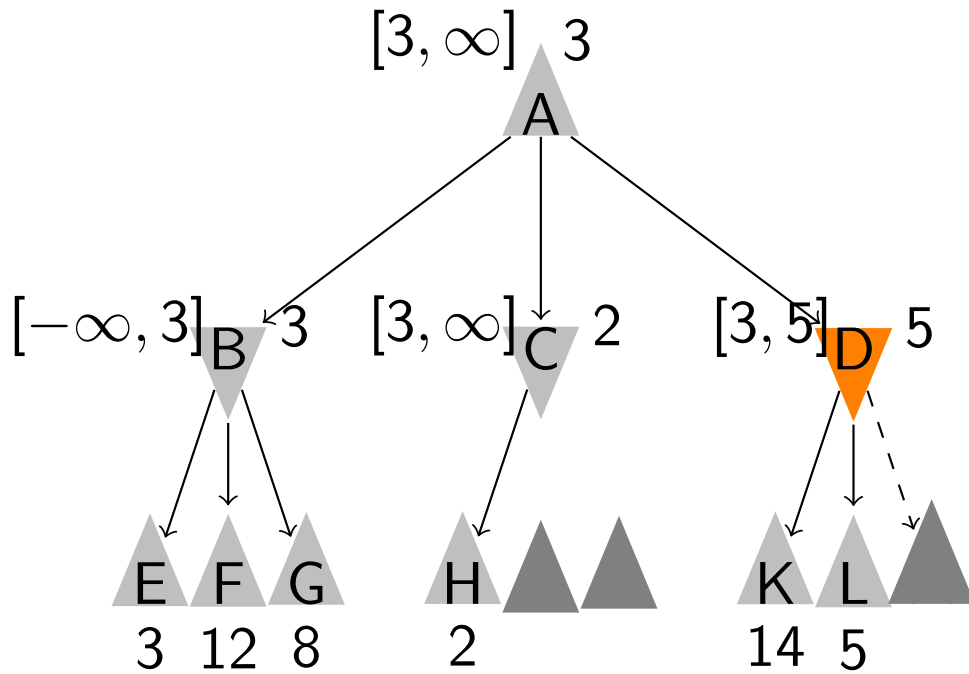
return $utility$

end

$\beta =$ **Min**($\beta, utility$)

end

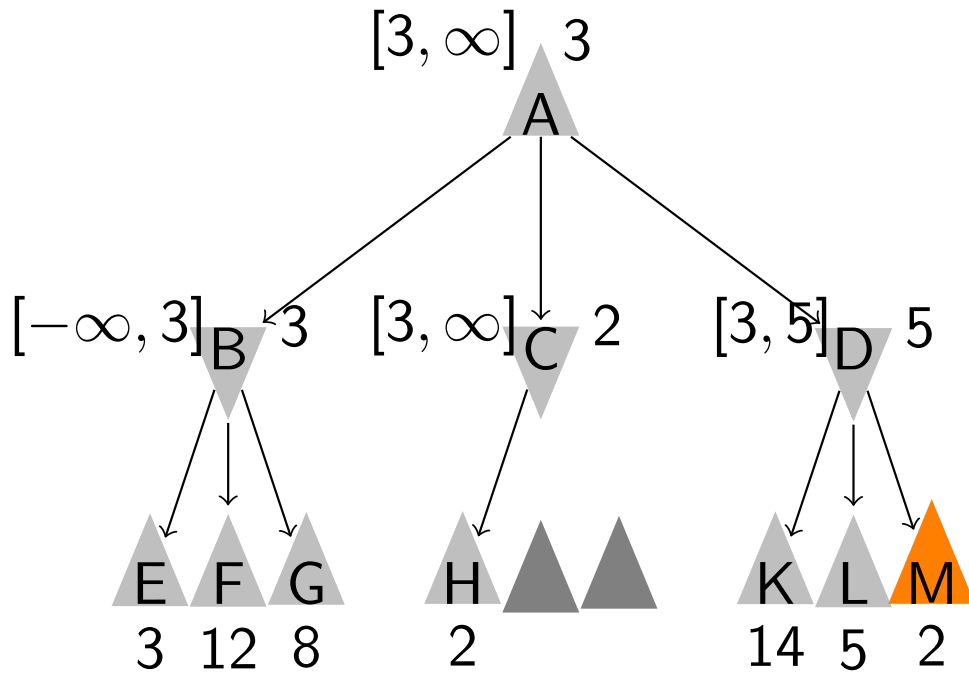
return utility



ALPHA-BETA Pruning – Beispiel

MaxValue($state, \alpha, \beta$)

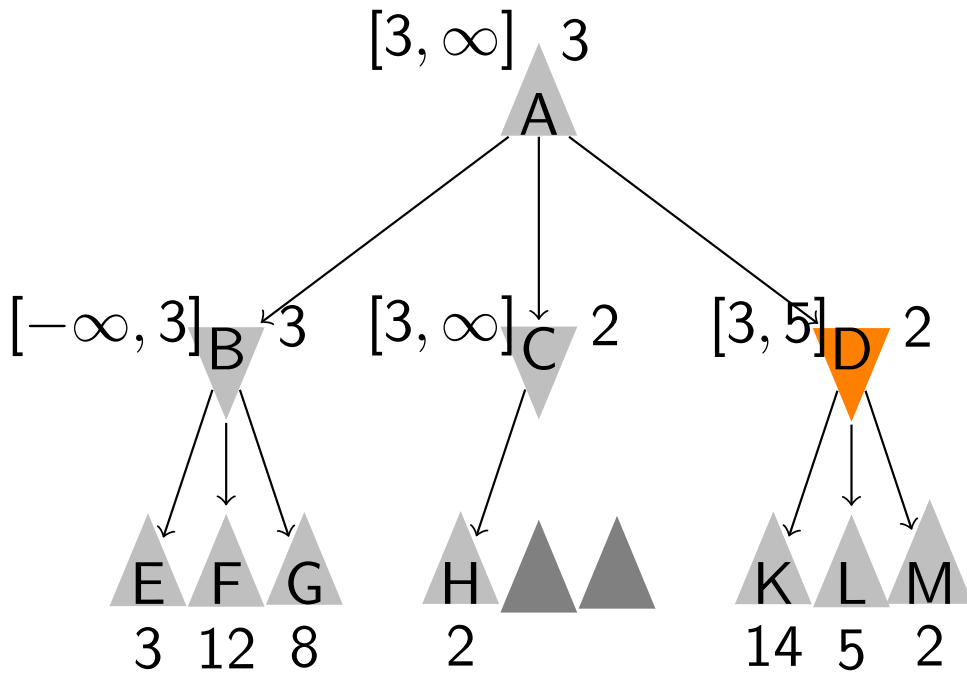
    if TerminalTest(state) **then**
        return Utility($state$)
    end
    $utility = -\infty$
    for each $a \in$ Actions($state$):
        $tmp =$
        MinValue(Result($state, a$), α, β)
        $utility =$ **Max**($utility, tmp$)
        if $utility \geq \beta$ **then**
            return $utility$
        end
        $\alpha =$ **Max**($\alpha, utility$)
    end
return utility



ALPHA-BETA Pruning – Beispiel

MinValue($state, \alpha, \beta$)

    if TerminalTest(state) **then**
        return Utility($state$)
    end
    $utility = +\infty$
    for each $a \in$ Actions($state$):
        $tmp =$
        MaxValue(Result($state, a$), α, β)
        $utility =$ **Min**($utility, tmp$)
        if $utility \leq \alpha$ **then**
            return $utility$
        end
        $\beta =$ **Min**($\beta, utility$)
    end
return utility



ALPHA-BETA Pruning – Beispiel

MaxValue($state, \alpha, \beta$)

if TerminalTest(state) **then**

return Utility($state$)

end

$utility = -\infty$

for each $a \in$ Actions($state$):

$tmp =$
MinValue(Result($state, a$), α, β)

$utility =$ **Max**($utility, tmp$)

if $utility \geq \beta$ **then**

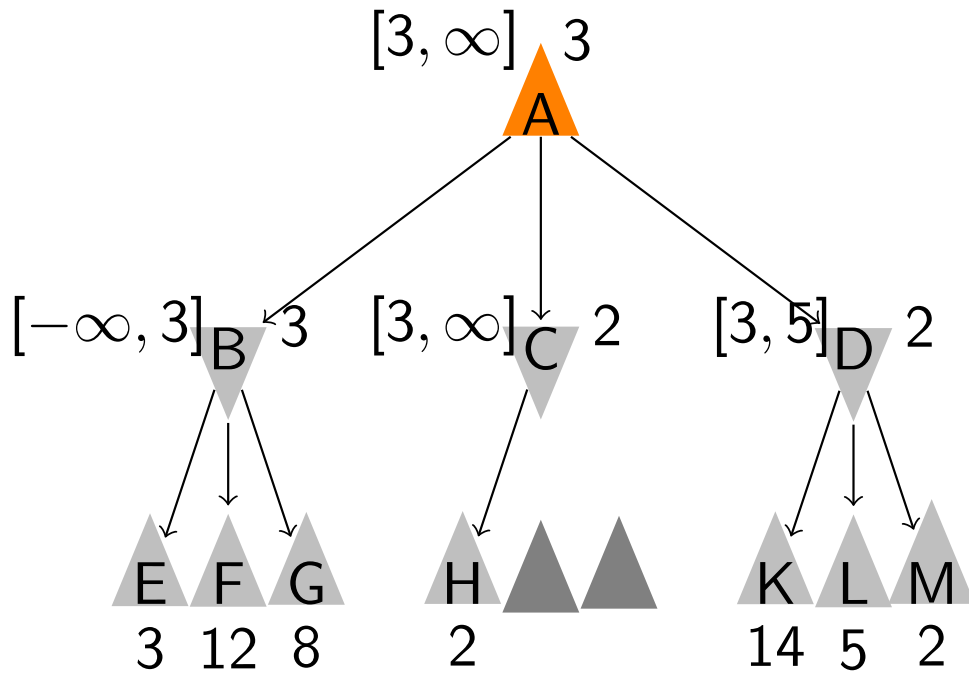
return $utility$

end

$\alpha =$ **Max**($\alpha, utility$)

end

return utility

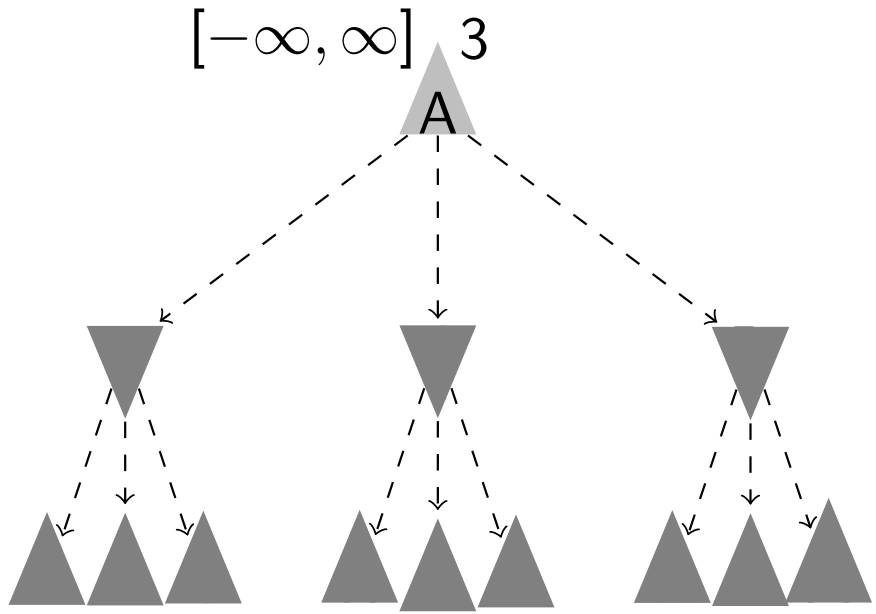


ALPHA-BETA Pruning – Beispiel

AlphaBetaSearch(*state*)

bestUtility = **MaxValue(***state*, $-\infty$, $+\infty$**)**

return *a* $\in$ Actions(*state*) with
utility == *bestUtility*

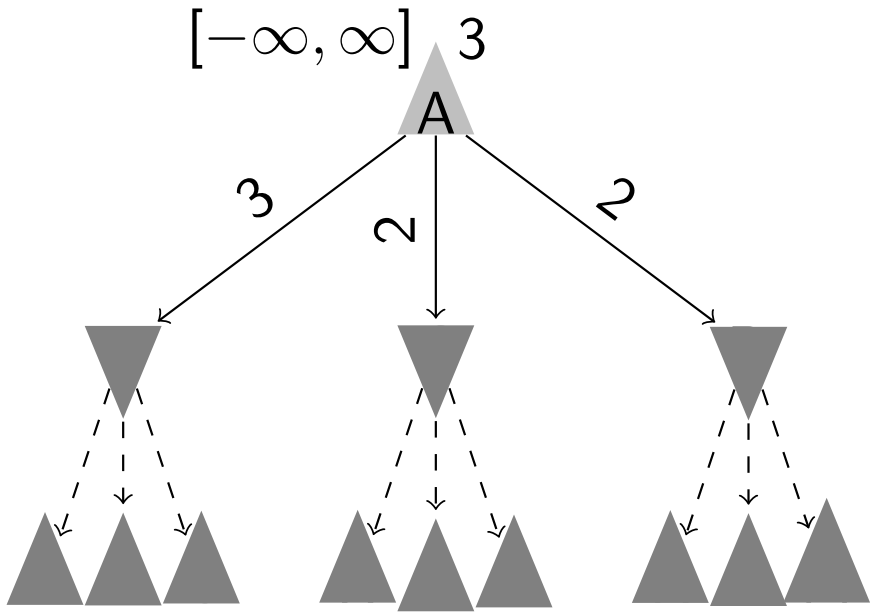


ALPHA-BETA Pruning – Beispiel

AlphaBetaSearch(*state*)

bestUtility = **MaxValue(***state*, $-\infty, +\infty$**)**

return *a* $\in$ Actions(*state*) with
utility == *bestUtility*

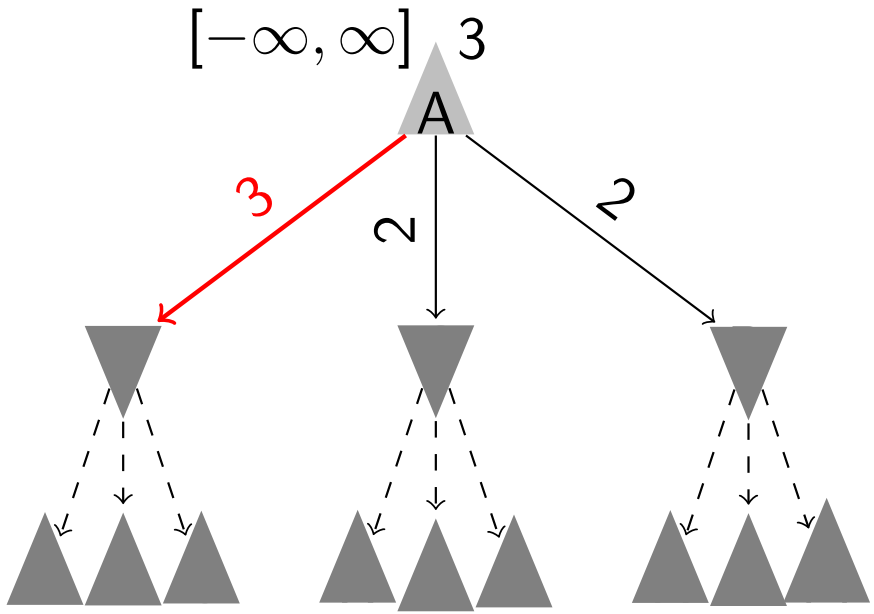


ALPHA-BETA Pruning – Beispiel

AlphaBetaSearch(*state*)

bestUtility = **MaxValue(***state*, $-\infty, +\infty$**)**

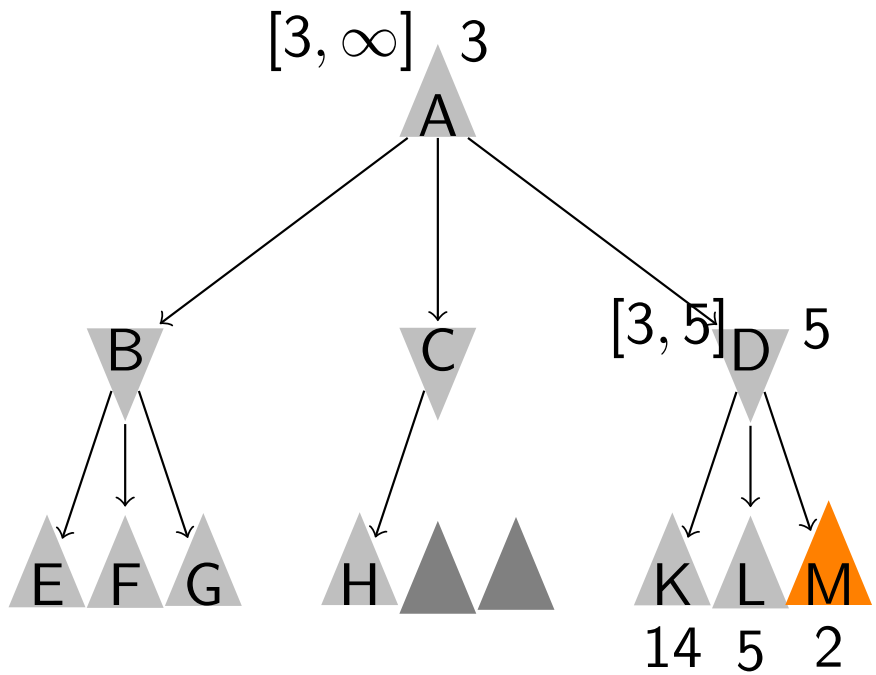
return *a* $\in$ Actions(*state*) with
utility == *bestUtility*



Outline

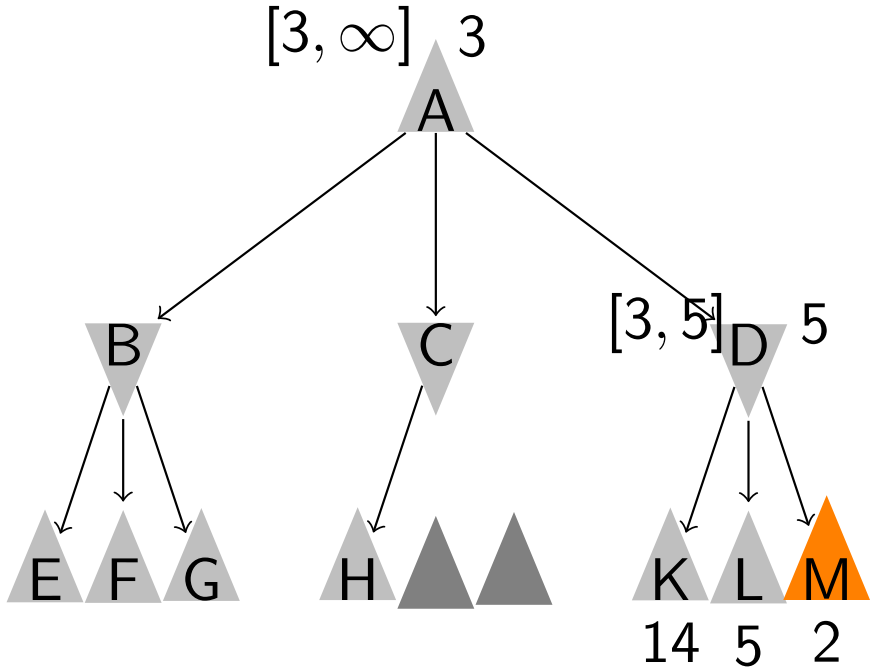
Zugsortierung

- Wenn wir Zustand **M** zuerst untersucht hätten, hätten wir uns **K** und **L** sparen können.



Zugsortierung

- Wenn wir Zustand **M** zuerst untersucht hätten, hätten wir uns **K** und **L** sparen können.
- Es ist aber klar, dass wenn wir das vorher gewusst hätten, hätten wir das Entscheidungsproblem bereits gelöst.



Outline

Evaluierungsfunktionen

- Eine Evaluierungsfunktion evaluiert einen Zustand oder eine Aktion gemäß der zu erwartenden Utility.

Evaluierungsfunktionen

- Eine Evaluierungsfunktion evaluiert einen Zustand oder eine Aktion gemäß der zu erwartenden Utility.

Evaluierungsfunktionen

- Eine Evaluierungsfunktion evaluiert einen Zustand oder eine Aktion gemäß der zu erwartenden Utility.
- Evaluierungsfunktionen sind Heuristiken. Absolut sicher kann man sich erst sein, wenn man einen Terminalen Zustand erreicht hat.

Evaluierungsfunktionen

- Eine Evaluierungsfunktion evaluiert einen Zustand oder eine Aktion gemäß der zu erwartenden Utility.
- Evaluierungsfunktionen sind Heuristiken. Absolut sicher kann man sich erst sein, wenn man einen Terminalen Zustand erreicht hat.

Evaluierungsfunktionen

- Eine Evaluierungsfunktion evaluiert einen Zustand oder eine Aktion gemäß der zu erwartenden Utility.
- Evaluierungsfunktionen sind Heuristiken. Absolut sicher kann man sich erst sein, wenn man einen Terminalen Zustand erreicht hat.
- Bei Tic-Tac-Toe könnte man z.B. Züge bevorzugen (d.H. zuerst nachverfolgen) in denen dem Gegner Chancen verbaut werden oder man selbst einen Schritt weiter kommt, um 3 Felder in einer Reihe zu haben.

Evaluierungsfunktionen

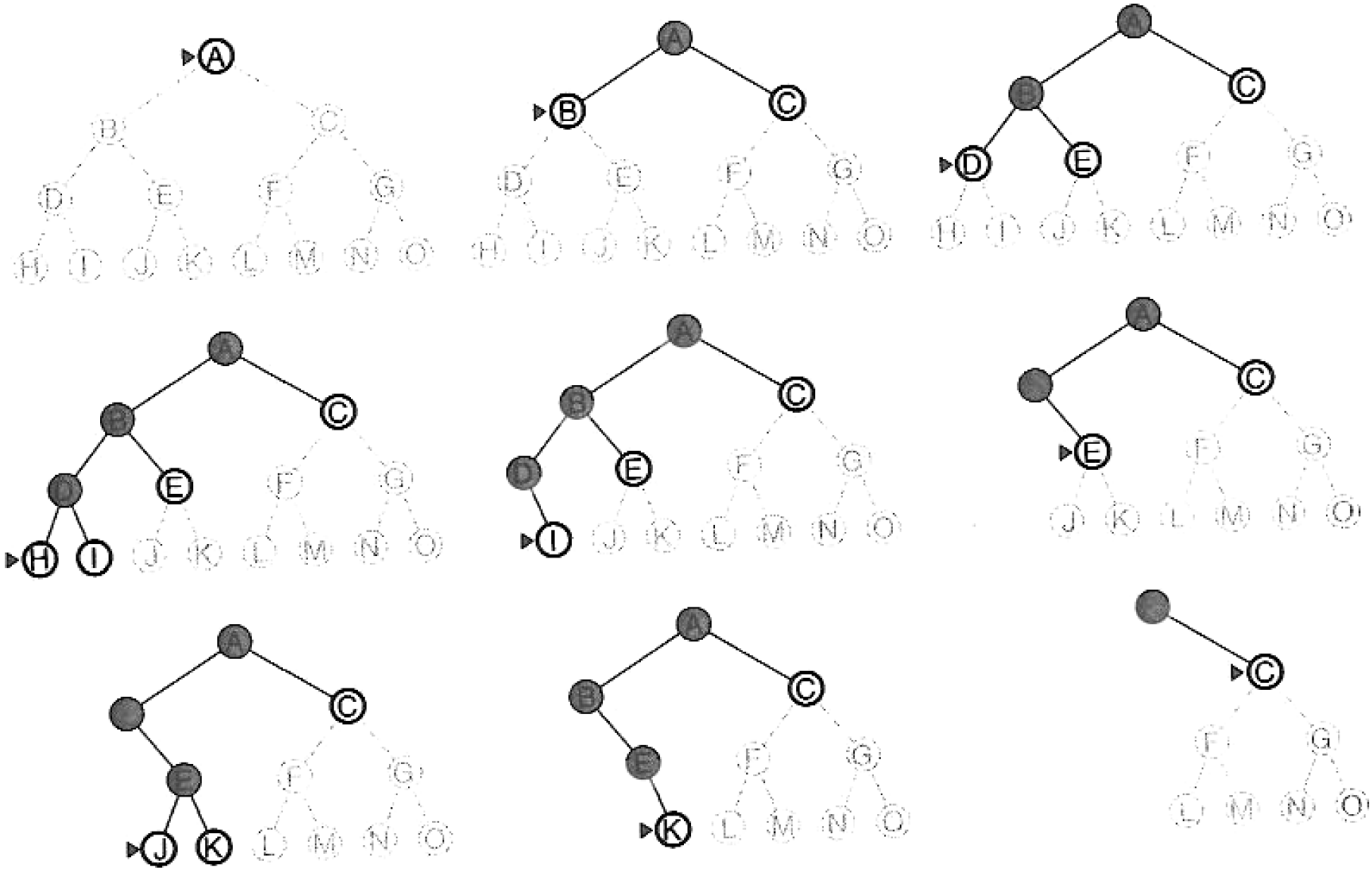
- Eine Evaluierungsfunktion evaluiert einen Zustand oder eine Aktion gemäß der zu erwartenden Utility.
- Evaluierungsfunktionen sind Heuristiken. Absolut sicher kann man sich erst sein, wenn man einen Terminalen Zustand erreicht hat.
- Bei Tic-Tac-Toe könnte man z.B. Züge bevorzugen (d.H. zuerst nachverfolgen) in denen dem Gegner Chancen verbaut werden oder man selbst einen Schritt weiter kommt, um 3 Felder in einer Reihe zu haben.

Evaluierungsfunktionen

- Eine Evaluierungsfunktion evaluiert einen Zustand oder eine Aktion gemäß der zu erwartenden Utility.
- Evaluierungsfunktionen sind Heuristiken. Absolut sicher kann man sich erst sein, wenn man einen Terminalen Zustand erreicht hat.
- Bei Tic-Tac-Toe könnte man z.B. Züge bevorzugen (d.H. zuerst nachverfolgen) in denen dem Gegner Chancen verbaut werden oder man selbst einen Schritt weiter kommt, um 3 Felder in einer Reihe zu haben.
- Achtung! **Min** und **Max** haben umgekehrte Ziele.

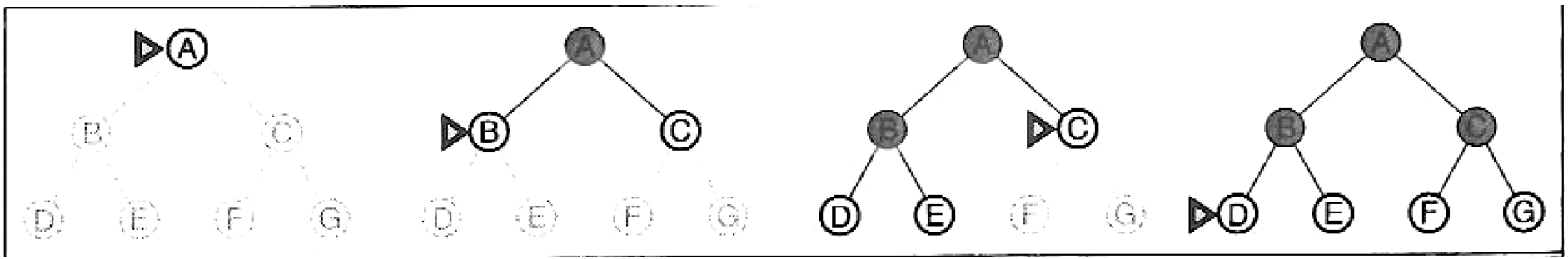
Outline

Tiefensuche



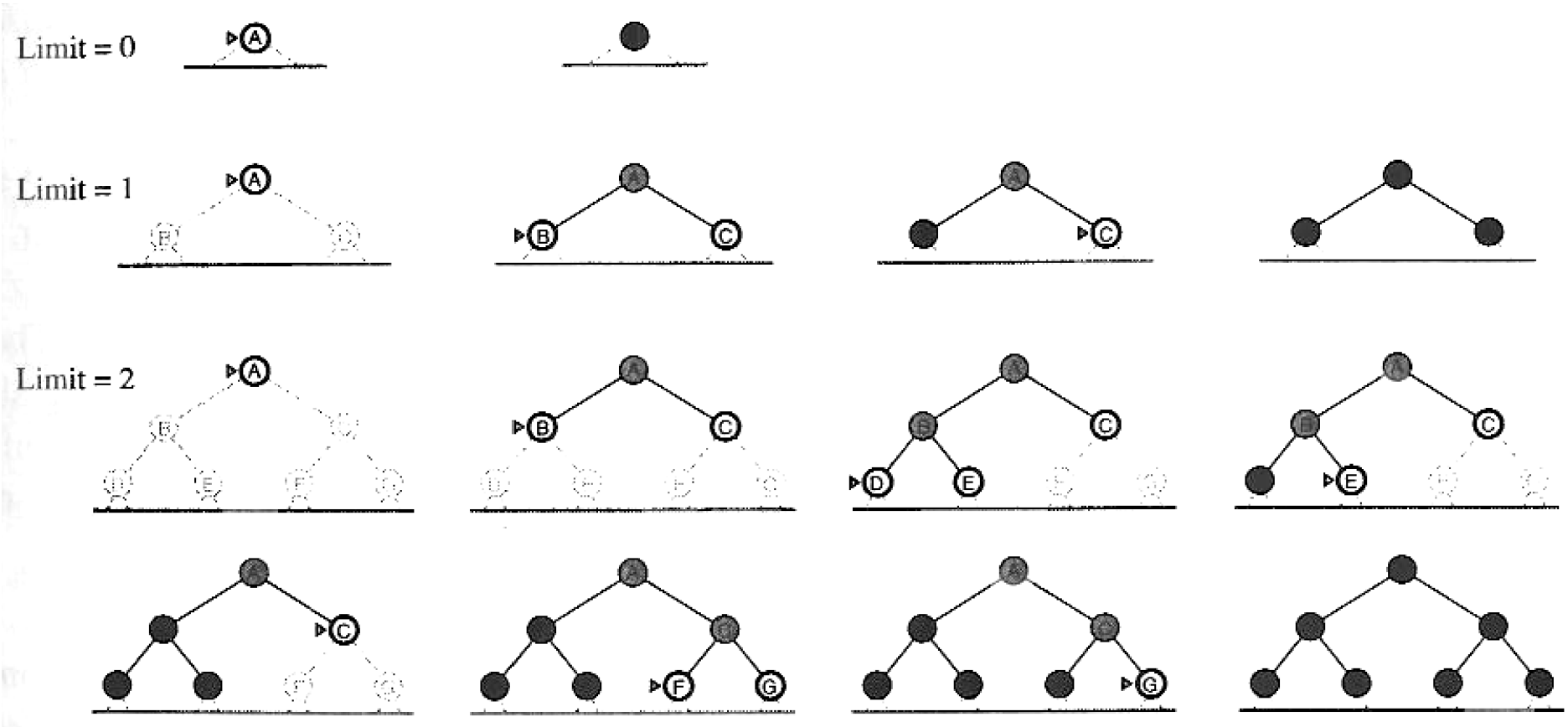
Quelle: Russel, S. & Norvig, P. Artificial intelligence: A modern approach Prentice Hall, 1995

Breitensuche



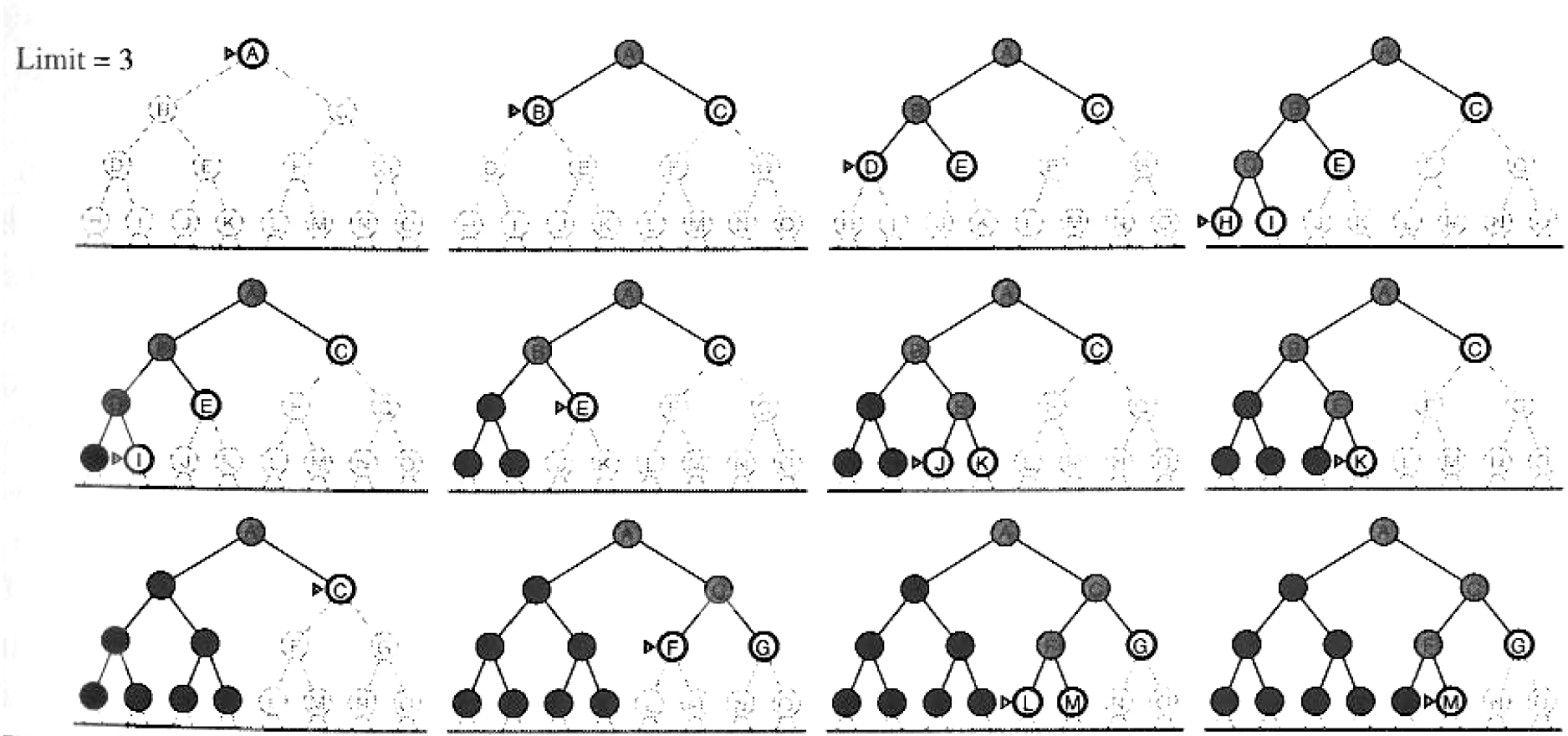
Quelle: Russel, S. & Norvig, P. Artificial intelligence: A modern approach Prentice Hall, 1995

Iterative Tiefensuche



Quelle: Russel, S. & Norvig, P. Artificial intelligence: A modern approach Prentice Hall, 1995

Iterative Tiefensuche



Quelle: Russel, S. & Norvig, P. Artificial intelligence: A modern approach Prentice Hall, 1995

Iterative Tiefensuche für Zugsortierung – eine Idee

- Erweiterung der Methode **TerminalTest**(*state*, *depth*) -> liefert auch **true** wenn maximale Tiefe erreicht wurde.

Iterative Tiefensuche für Zugsortierung – eine Idee

- Erweiterung der Methode **TerminalTest**(*state*, *depth*) -> liefert auch **true** wenn maximale Tiefe erreicht wurde.
- Erweiterung der Methode **Utility**(*state*), so dass nicht Terminale Zustände gemäß einer Evaluierungsfunktion bewertet werden.

Iterative Tiefensuche für Zugsortierung – eine Idee

- Erweiterung der Methode **TerminalTest**(*state*, *depth*) -> liefert auch **true** wenn maximale Tiefe erreicht wurde.
- Erweiterung der Methode **Utility**(*state*), so dass nicht Terminale Zustände gemäß einer Evaluierungsfunktion bewertet werden.
- Bevor die Methoden **MaxValue** bzw **MinValue** verlassen werden, werden die Aktionen gemäß ihrer (zu erwarteten) Utility sortiert (in **MaxValue** absteigend und in **MinValue** aufsteigend).

Iterative Tiefensuche für Zugsortierung – eine Idee

- Erweiterung der Methode **TerminalTest**(*state*, *depth*) -> liefert auch **true** wenn maximale Tiefe erreicht wurde.
- Erweiterung der Methode **Utility**(*state*), so dass nicht Terminale Zustände gemäß einer Evaluierungsfunktion bewertet werden.
- Bevor die Methoden **MaxValue** bzw **MinValue** verlassen werden, werden die Aktionen gemäß ihrer (zu erwarteten) Utility sortiert (in **MaxValue** absteigend und in **MinValue** aufsteigend).
- Abbruchkriterium ist: Entweder kann ein Terminaler Zustand mit maximaler Utility (Sieg) vom Anfangsknoten aus erreicht werden oder das Zeitlimit wurde erreicht. Bei Abbruch wegen erreichen des Zeitlimits wird die bis dahin (voraussichtlich) beste Aktion ausgewählt.

Iterative Tiefensuche für Zugsortierung – eine Idee

- Erweiterung der Methode **TerminalTest**(*state*, *depth*) -> liefert auch **true** wenn maximale Tiefe erreicht wurde.
- Erweiterung der Methode **Utility**(*state*), so dass nicht Terminale Zustände gemäß einer Evaluierungsfunktion bewertet werden.
- Bevor die Methoden **MaxValue** bzw **MinValue** verlassen werden, werden die Aktionen gemäß ihrer (zu erwarteten) Utility sortiert (in **MaxValue** absteigend und in **MinValue** aufsteigend).
- Abbruchkriterium ist: Entweder kann ein Terminaler Zustand mit maximaler Utility (Sieg) vom Anfangsknoten aus erreicht werden oder das Zeitlimit wurde erreicht. Bei Abbruch wegen erreichen des Zeitlimits wird die bis dahin (voraussichtlich) beste Aktion ausgewählt.

Iterative Tiefensuche für Zugsortierung – eine Idee

- Erweiterung der Methode **TerminalTest**(*state*, *depth*) -> liefert auch **true** wenn maximale Tiefe erreicht wurde.
- Erweiterung der Methode **Utility**(*state*), so dass nicht Terminale Zustände gemäß einer Evaluierungsfunktion bewertet werden.
- Bevor die Methoden **MaxValue** bzw **MinValue** verlassen werden, werden die Aktionen gemäß ihrer (zu erwarteten) Utility sortiert (in **MaxValue** absteigend und in **MinValue** aufsteigend).
- Abbruchkriterium ist: Entweder kann ein Terminaler Zustand mit maximaler Utility (Sieg) vom Anfangsknoten aus erreicht werden oder das Zeitlimit wurde erreicht. Bei Abbruch wegen erreichen des Zeitlimits wird die bis dahin (voraussichtlich) beste Aktion ausgewählt.
- Rückgabe von **AlphaBetaSearch** ist dann die (voraussichtlich) beste Aktion.

Iterative Tiefensuche für Zugsortierung – eine Idee

- Erweiterung der Methode **TerminalTest**(*state*, *depth*) -> liefert auch **true** wenn maximale Tiefe erreicht wurde.
- Erweiterung der Methode **Utility**(*state*), so dass nicht Terminale Zustände gemäß einer Evaluierungsfunktion bewertet werden.
- Bevor die Methoden **MaxValue** bzw **MinValue** verlassen werden, werden die Aktionen gemäß ihrer (zu erwarteten) Utility sortiert (in **MaxValue** absteigend und in **MinValue** aufsteigend).
- Abbruchkriterium ist: Entweder kann ein Terminaler Zustand mit maximaler Utility (Sieg) vom Anfangsknoten aus erreicht werden oder das Zeitlimit wurde erreicht. Bei Abbruch wegen erreichen des Zeitlimits wird die bis dahin (voraussichtlich) beste Aktion ausgewählt.
- Rückgabe von **AlphaBetaSearch** ist dann die (voraussichtlich) beste Aktion.

Iterative Tiefensuche für Zugsortierung – eine Idee

- Erweiterung der Methode **TerminalTest**(*state*, *depth*) -> liefert auch **true** wenn maximale Tiefe erreicht wurde.
- Erweiterung der Methode **Utility**(*state*), so dass nicht Terminale Zustände gemäß einer Evaluierungsfunktion bewertet werden.
- Bevor die Methoden **MaxValue** bzw **MinValue** verlassen werden, werden die Aktionen gemäß ihrer (zu erwarteten) Utility sortiert (in **MaxValue** absteigend und in **MinValue** aufsteigend).
- Abbruchkriterium ist: Entweder kann ein Terminaler Zustand mit maximaler Utility (Sieg) vom Anfangsknoten aus erreicht werden oder das Zeitlimit wurde erreicht. Bei Abbruch wegen erreichen des Zeitlimits wird die bis dahin (voraussichtlich) beste Aktion ausgewählt.
- Rückgabe von **AlphaBetaSearch** ist dann die (voraussichtlich) beste Aktion.
- Achtung! Vor Beginn einer neuen Iteration α und β zurücksetzen.

Outline

Hausaufgabe

1. Implementieren Sie das Spiel "4 Gewinnt"(siehe http://de.wikipedia.org/wiki/Vier_gewinnt) als Computersimulation, wo man als Mensch gegen den Computer spielen kann.
2. Die KI des Spiels soll auf dem Minimax Algorithmus mit Alpha-Beta Pruning und einer heuristischen Evaluierungsfunktion, um aussichtsreiche Züge zuerst zu untersuchen, basieren. Außerdem soll es möglich sein, die für einen Spielzug zur Verfügung stehende Zeit variabel zu gestalten. Verwenden Sie dafür z.B. die iterative Tiefensuche.
3. Untersuchen Sie die Leistungsfähigkeit des Algorithmus in Abhängigkeit von zur Verfügung stehenden Zeit.
4. Bereiten Sie bis zur nächsten Sitzung eine Präsentation inklusive Demonstration Ihres Programms vor. Die Präsentation soll ca. 10 Minuten dauern.
5. Diesmal bitte keinen Sourcecode an mich senden!