Alpha-Beta Pruning

Outline

I. Alpha-beta pruning

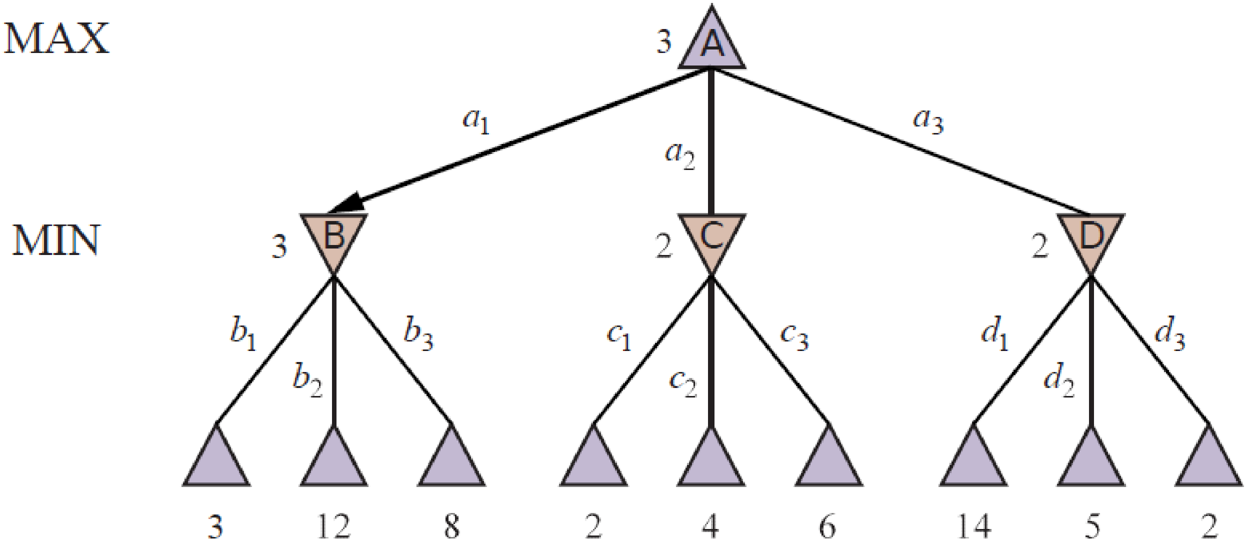
II. Heuristic alpha-beta tree approach

* Figures/images are from the textbook site (or by the instructor). Otherwise, the source is cited unless such citation would make little sense due to the triviality of generating the image.

I. Alpha-Beta Cutoff

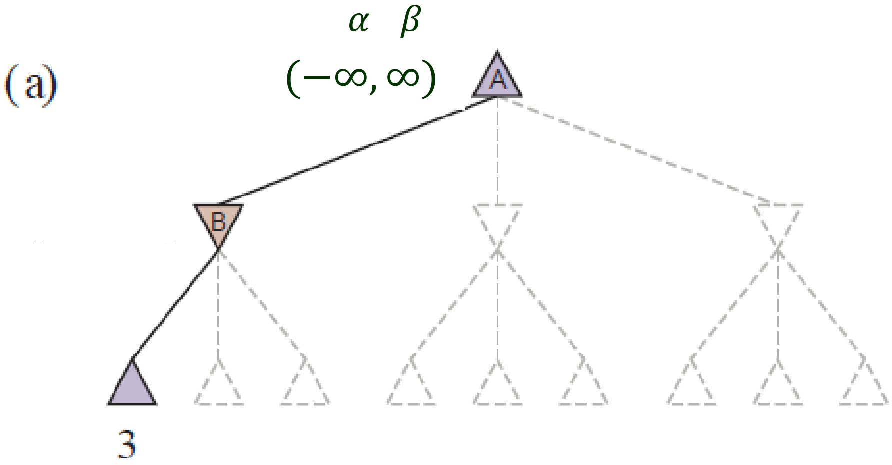
♣ #states is exponential in the depth of the game tree.

♦ But we can often compute the correct minimax decision by pruning large parts of the tree that do not affect the outcome.



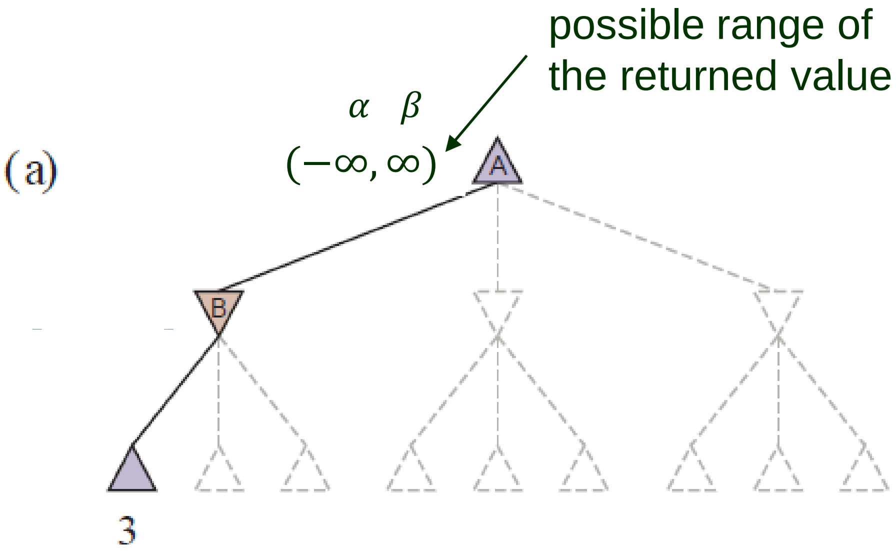
Re-examining the Game Tree

Fig. 5.5 in the textbook *incorrectly executes* the algorithm in Fig. 5.7.



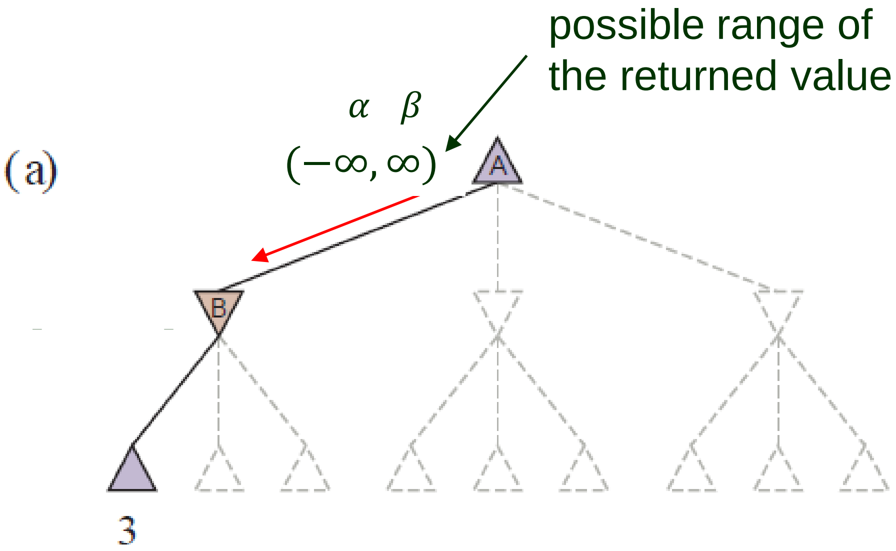
Re-examining the Game Tree

Fig. 5.5 in the textbook *incorrectly executes* the algorithm in Fig. 5.7.



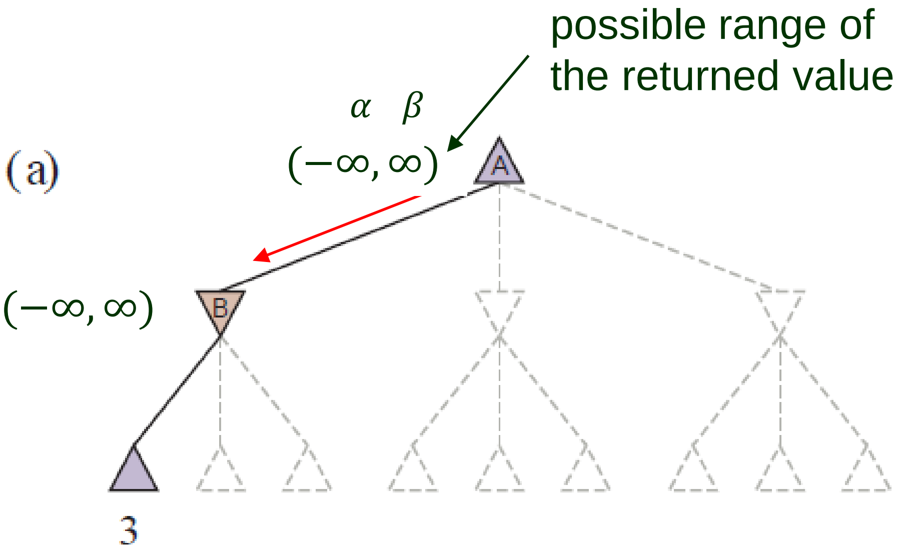
Re-examining the Game Tree

Fig. 5.5 in the textbook *incorrectly executes* the algorithm in Fig. 5.7.



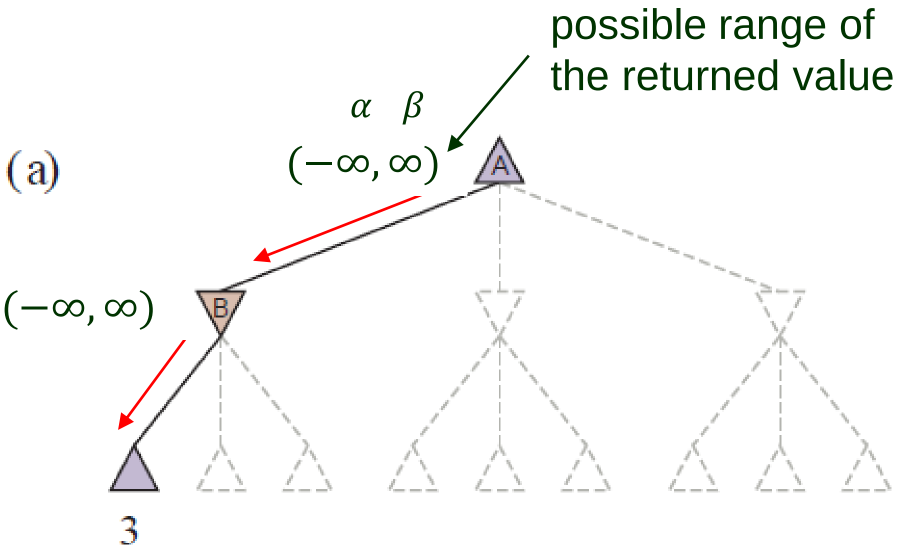
Re-examining the Game Tree

Fig. 5.5 in the textbook *incorrectly executes* the algorithm in Fig. 5.7.



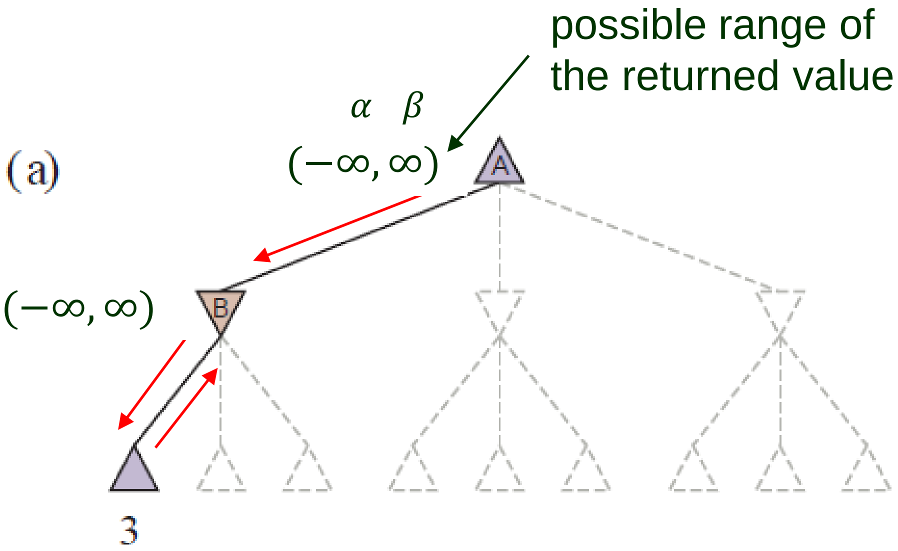
Re-examining the Game Tree

Fig. 5.5 in the textbook *incorrectly executes* the algorithm in Fig. 5.7.



Re-examining the Game Tree

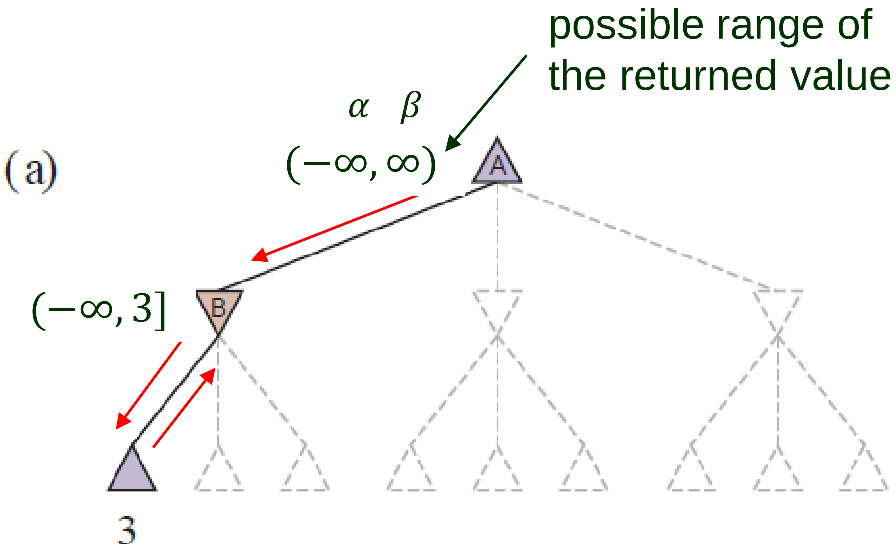
Fig. 5.5 in the textbook *incorrectly executes* the algorithm in Fig. 5.7.



possible range of
the returned value

Re-examining the Game Tree

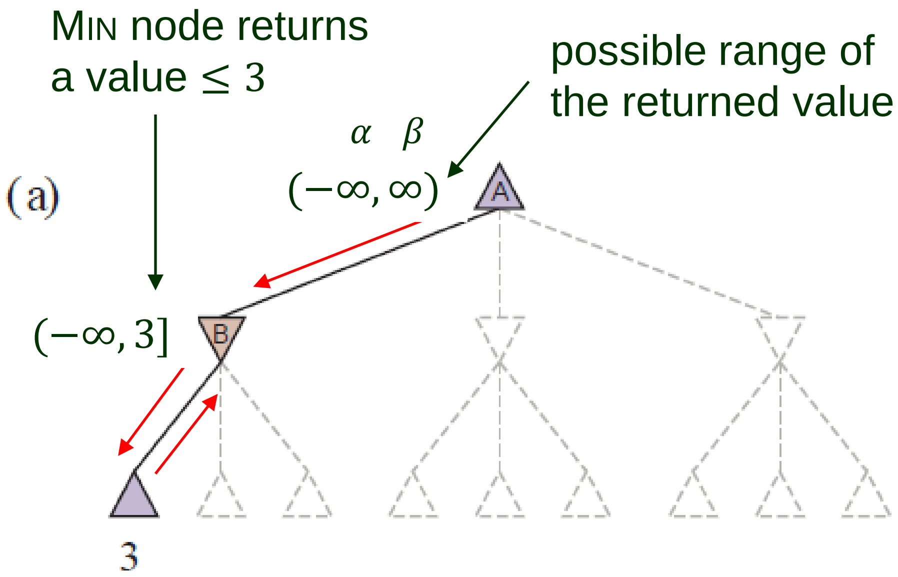
Fig. 5.5 in the textbook *incorrectly executes* the algorithm in Fig. 5.7.



possible range of the returned value

Re-examining the Game Tree

Fig. 5.5 in the textbook *incorrectly executes* the algorithm in Fig. 5.7.

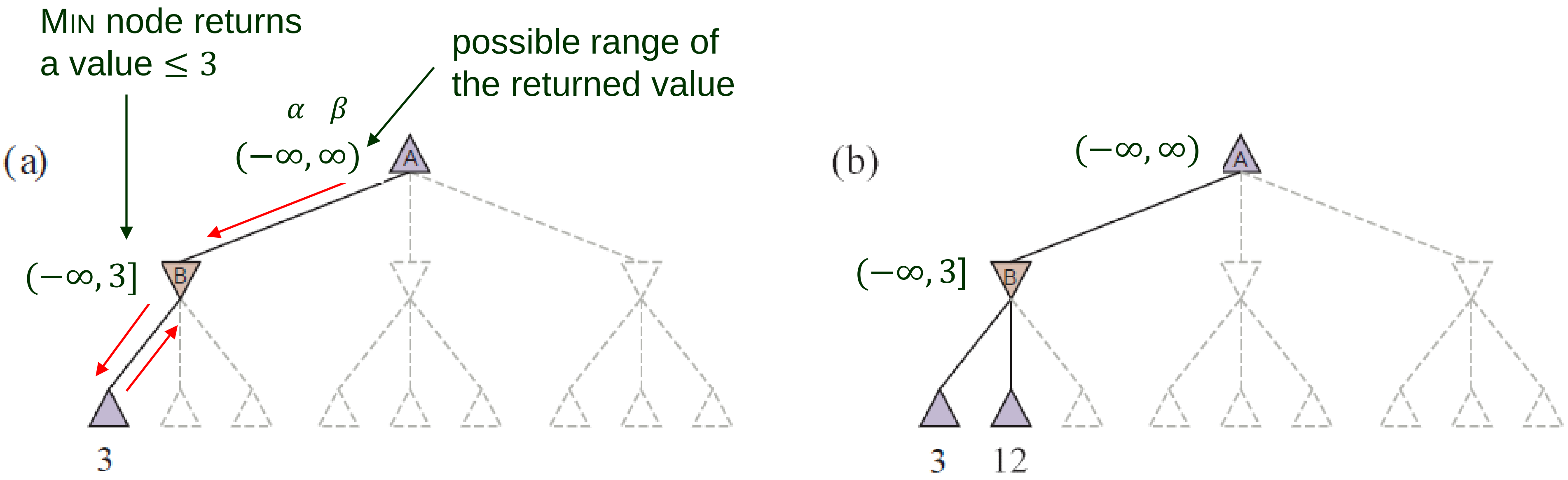


MIN node returns a value ≤ 3

possible range of the returned value

Re-examining the Game Tree

Fig. 5.5 in the textbook *incorrectly executes* the algorithm in Fig. 5.7.

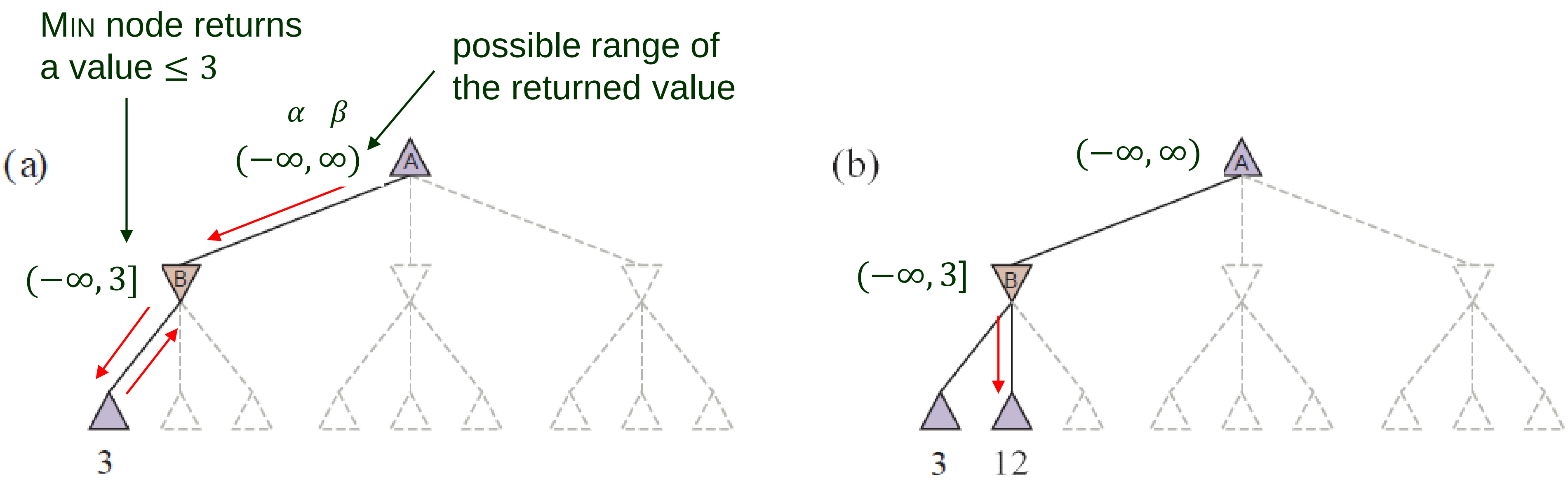


MIN node returns a value ≤ 3

possible range of the returned value

Re-examining the Game Tree

Fig. 5.5 in the textbook *incorrectly executes* the algorithm in Fig. 5.7.

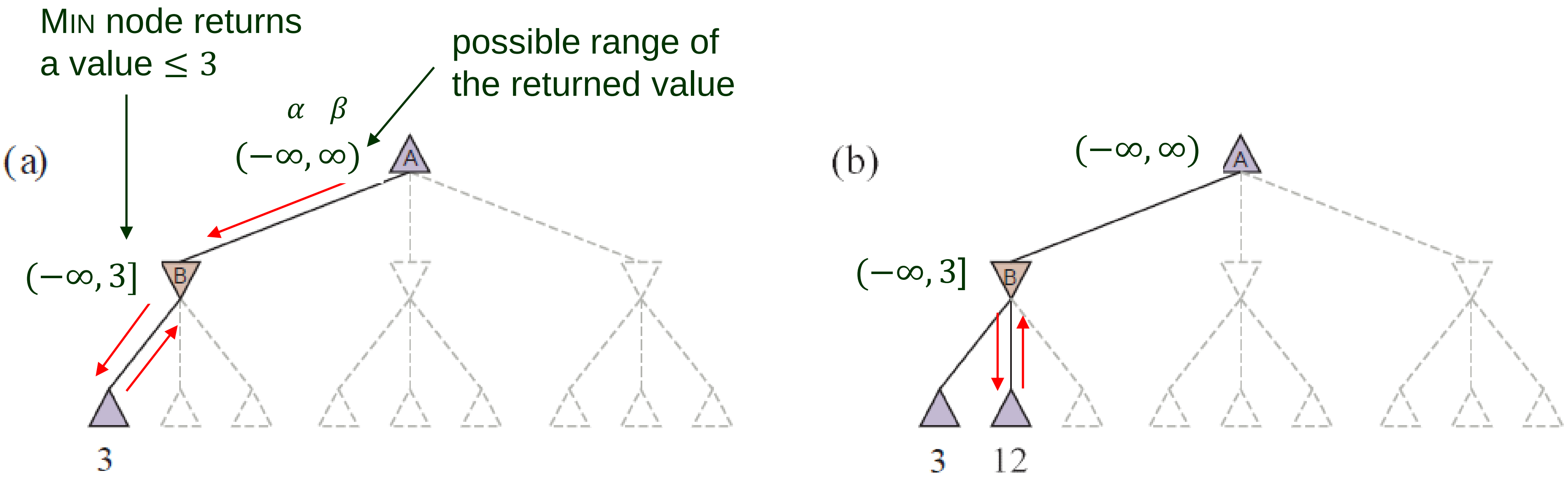


MIN node returns a value ≤ 3

possible range of the returned value

Re-examining the Game Tree

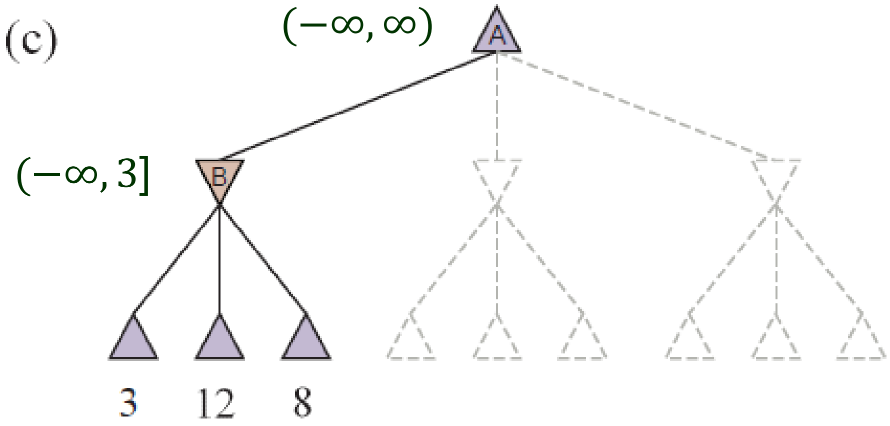
Fig. 5.5 in the textbook *incorrectly executes* the algorithm in Fig. 5.7.



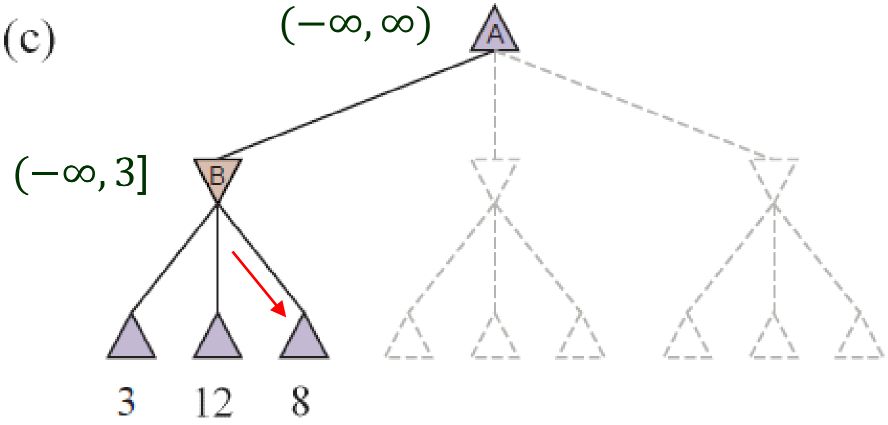
MIN node returns a value ≤ 3

possible range of the returned value

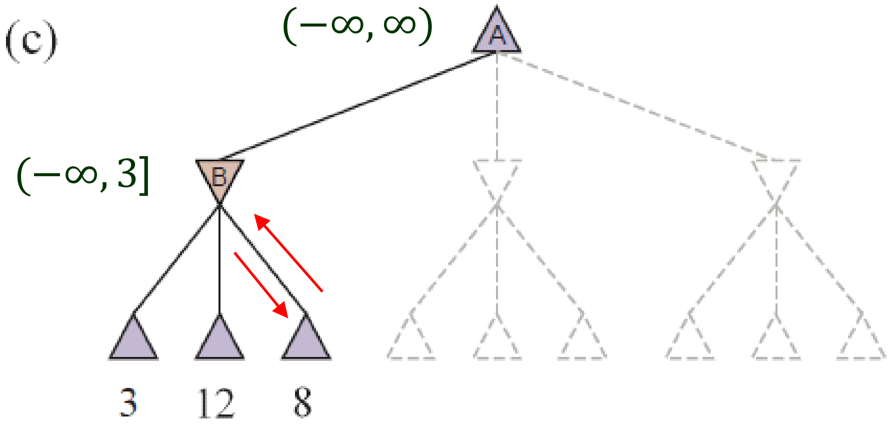
Cont’d



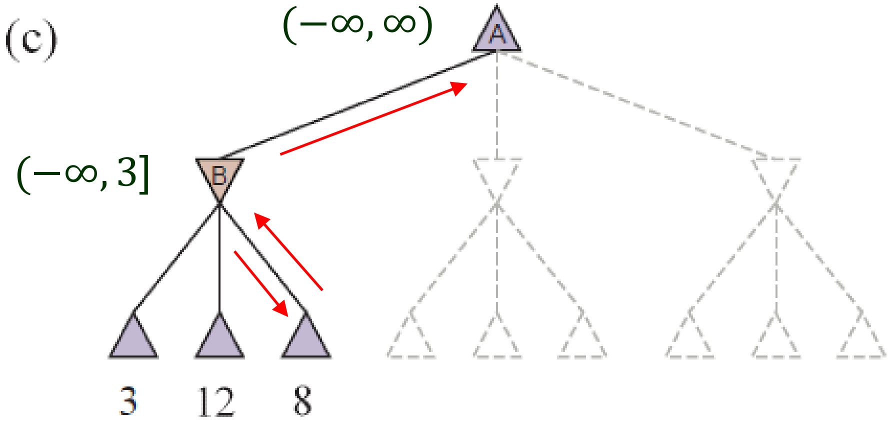
Cont'd



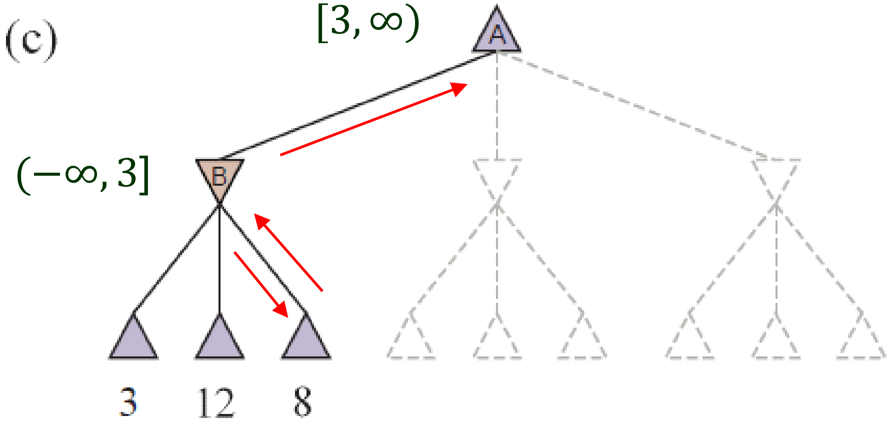
Cont’d



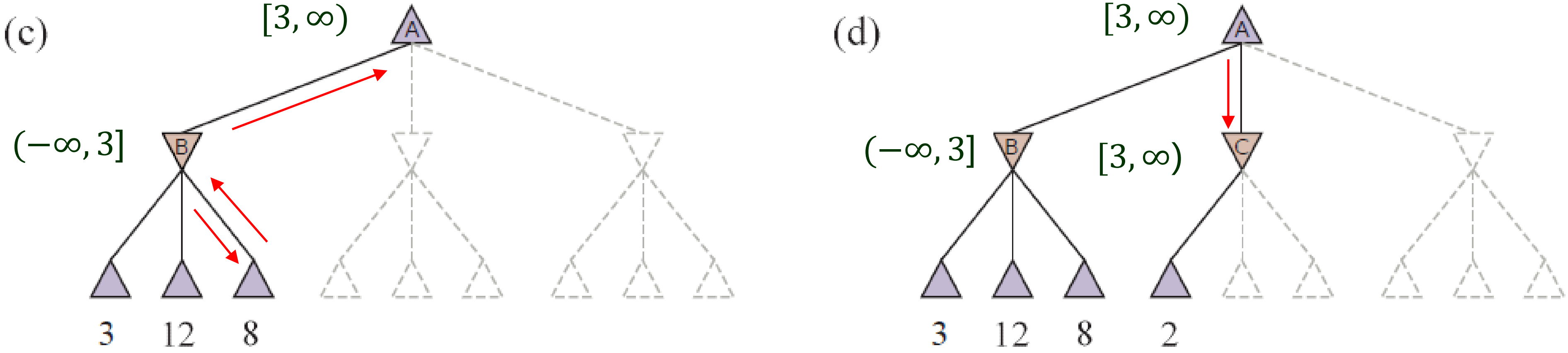
Cont’d



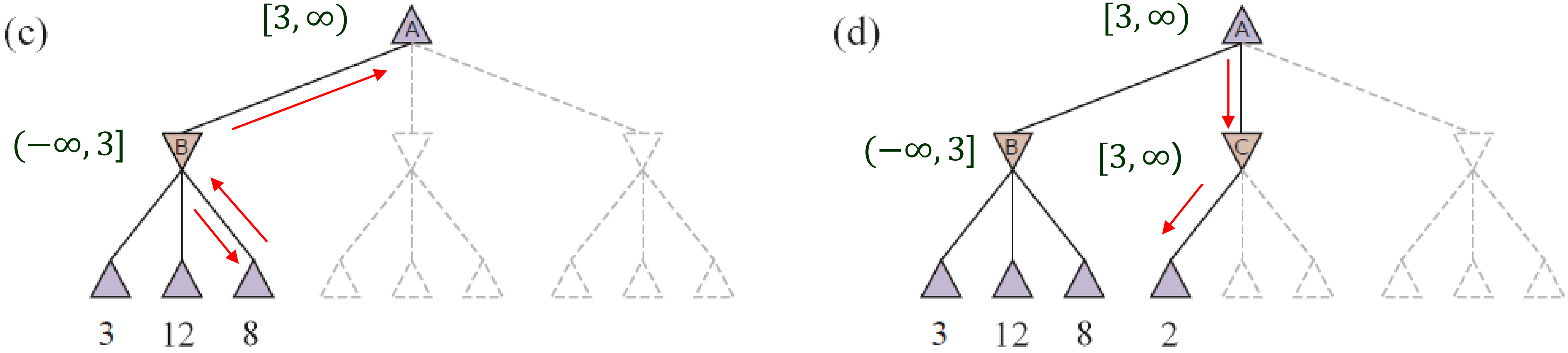
Cont’d



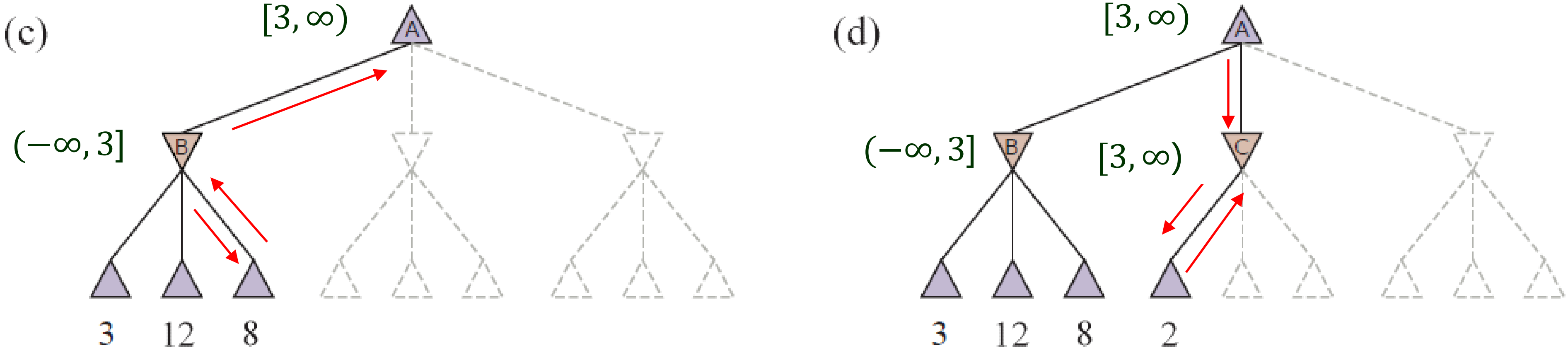
Cont’d



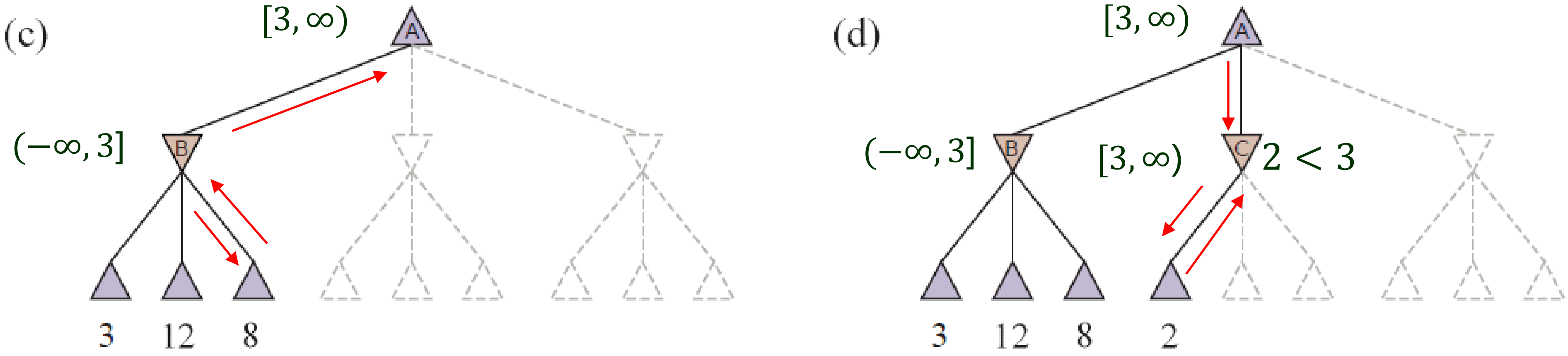
Cont’d



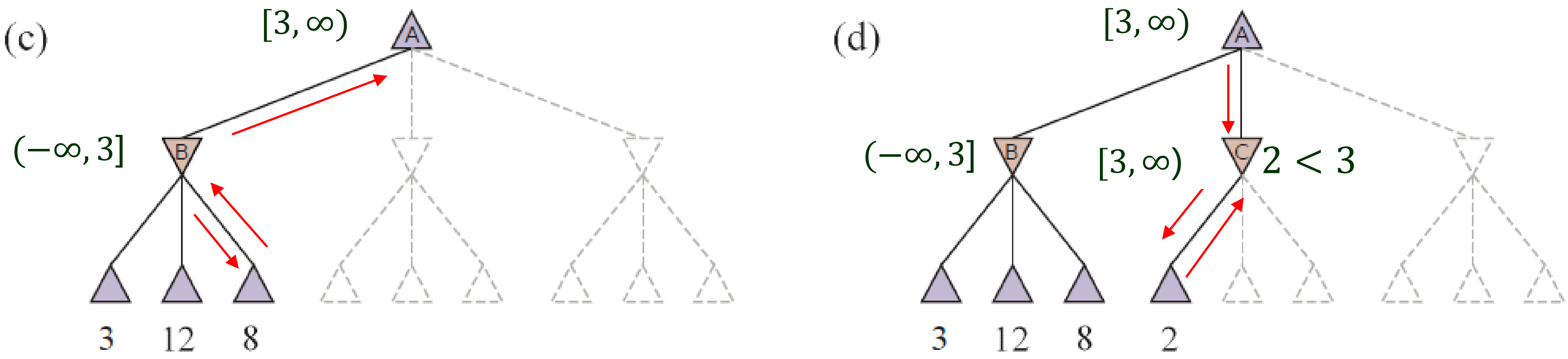
Cont’d



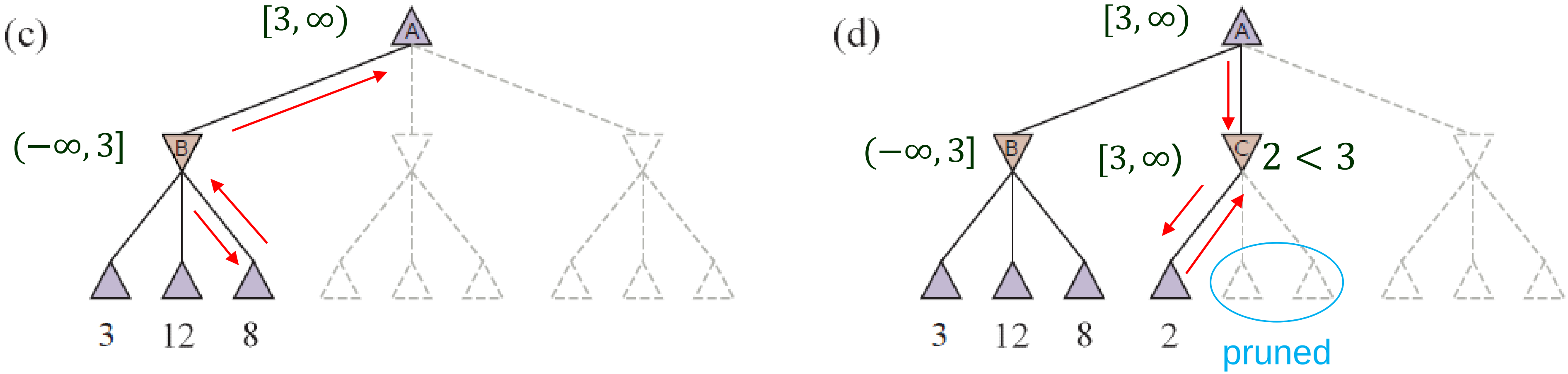
Cont’d



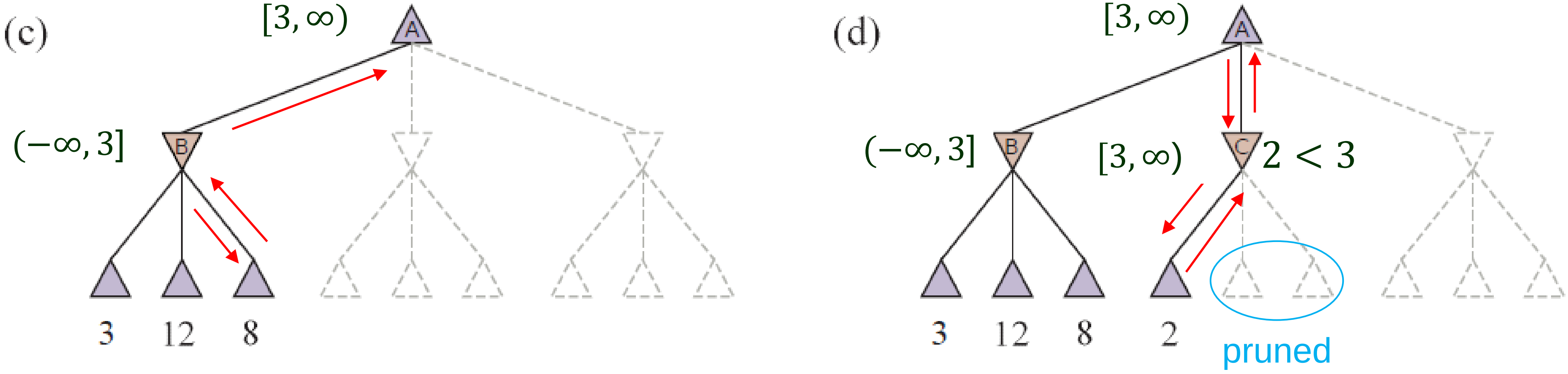
Cont'd



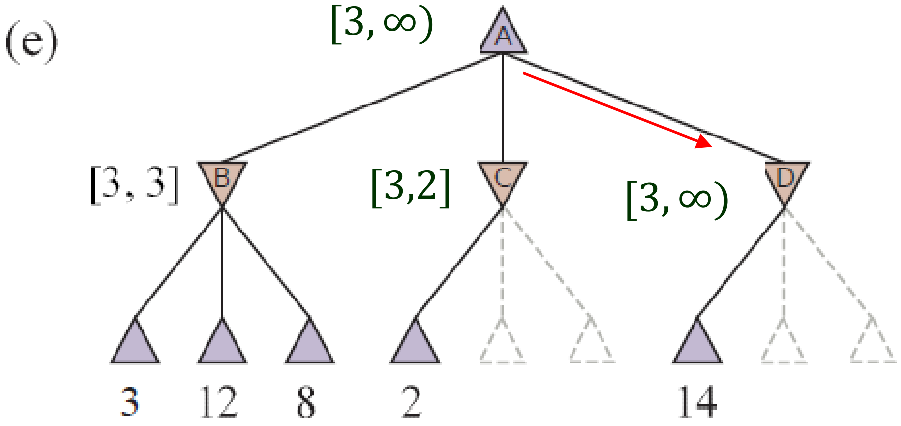
Cont’d



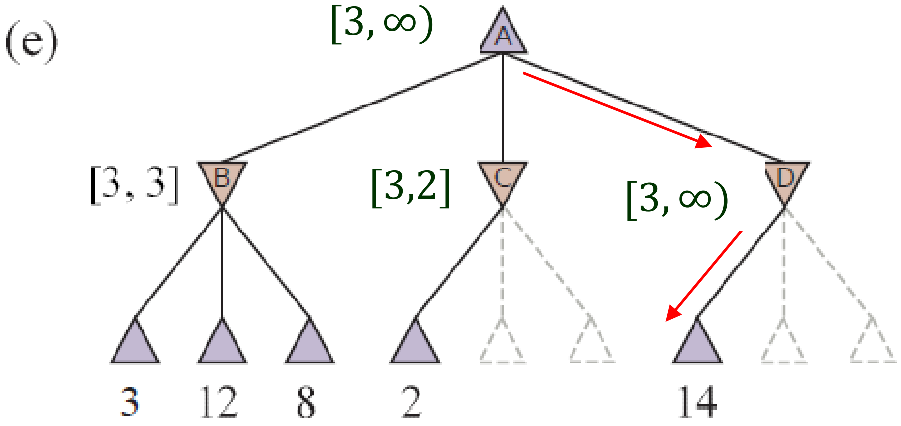
Cont’d



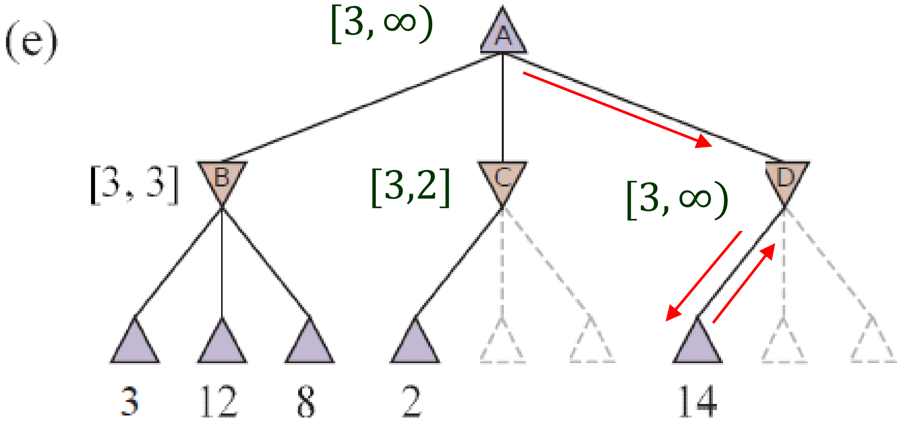
Cont'd



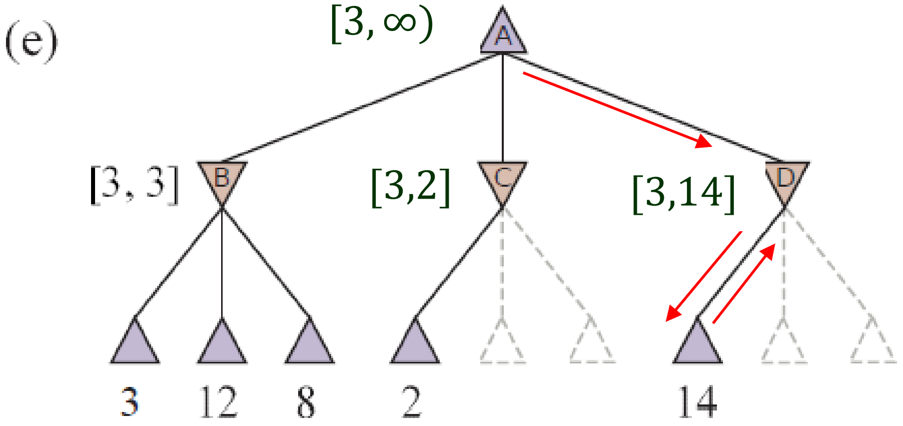
Cont’d



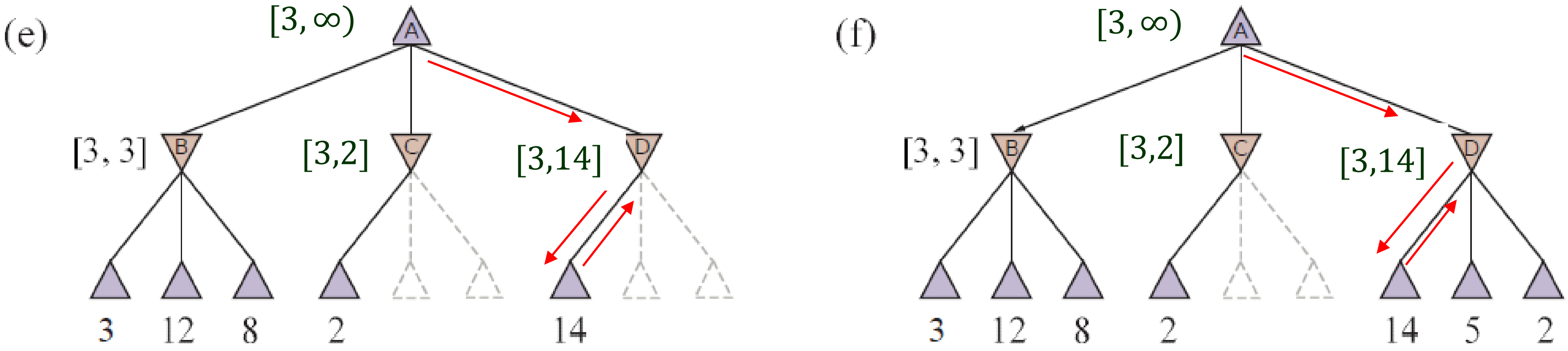
Cont’d



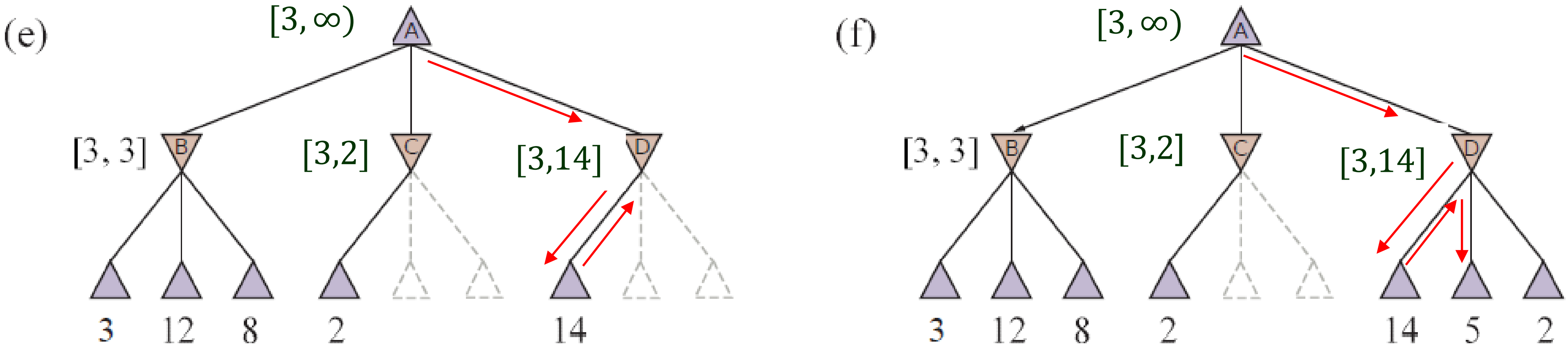
Cont’d



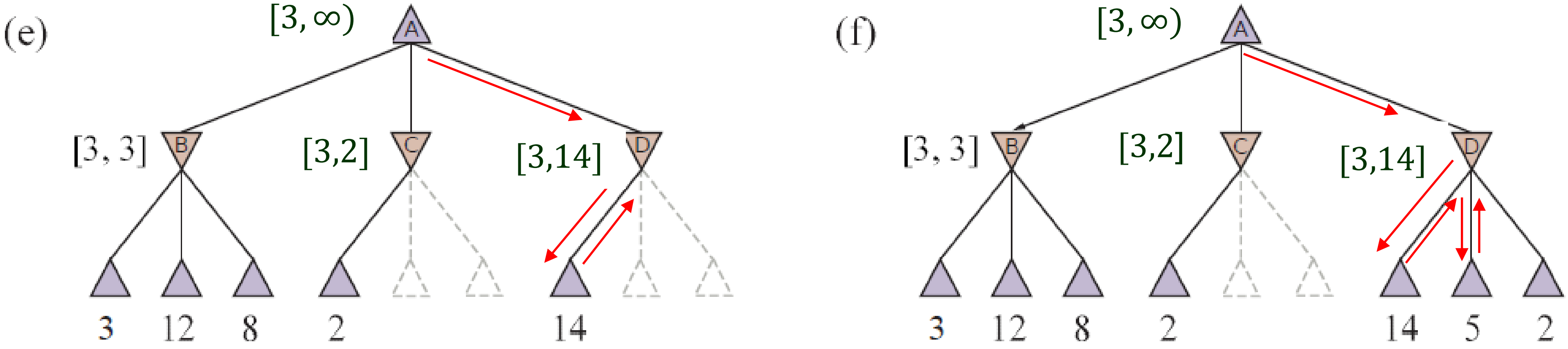
Cont’d



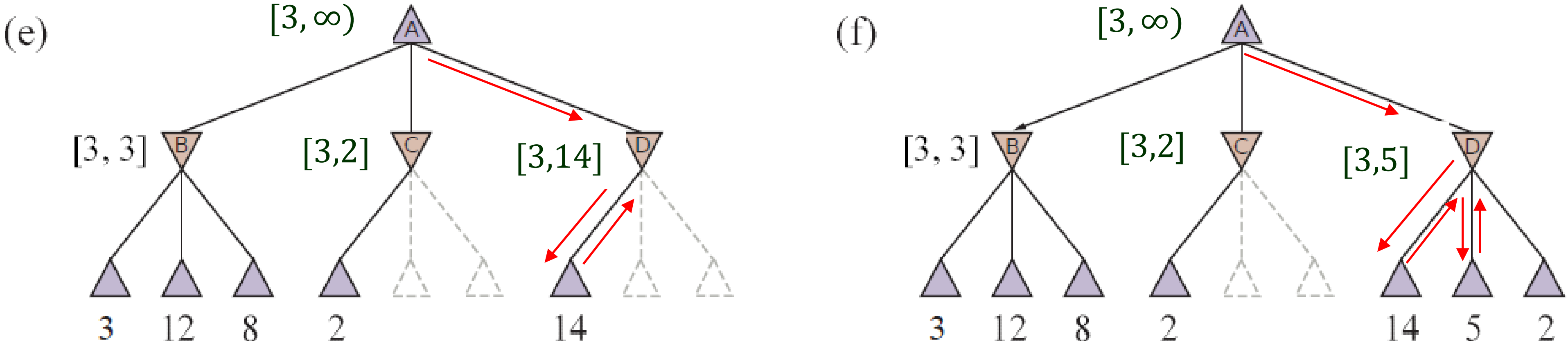
Cont’d



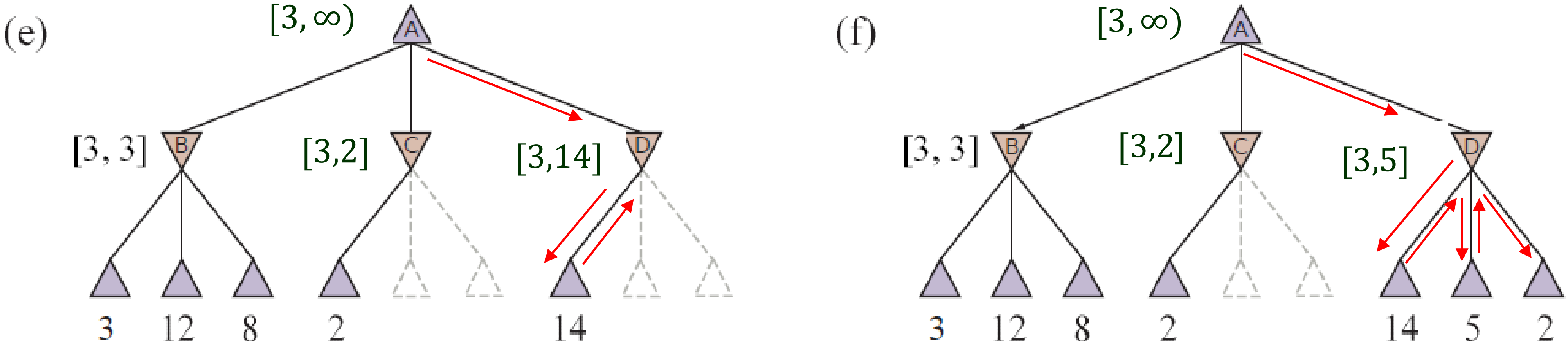
Cont’d



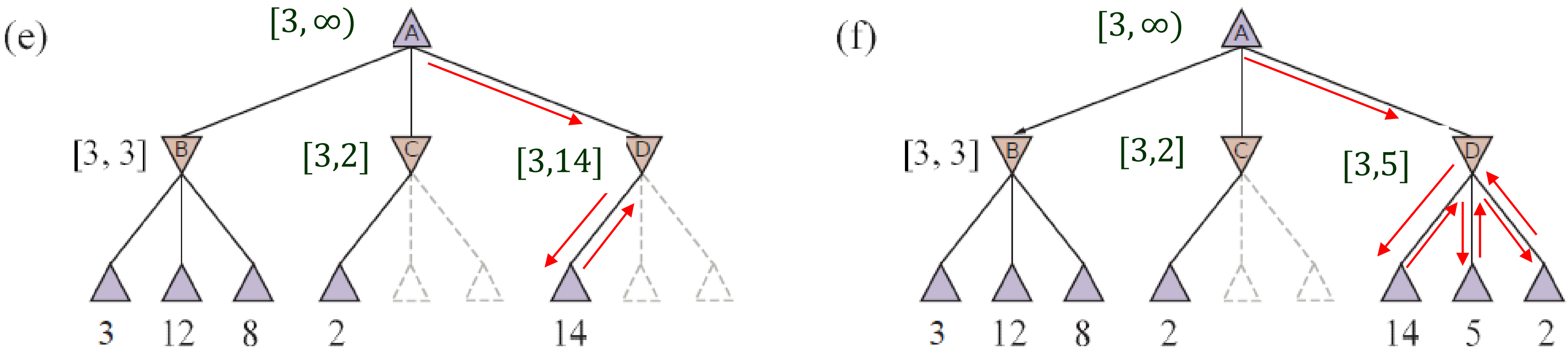
Cont’d



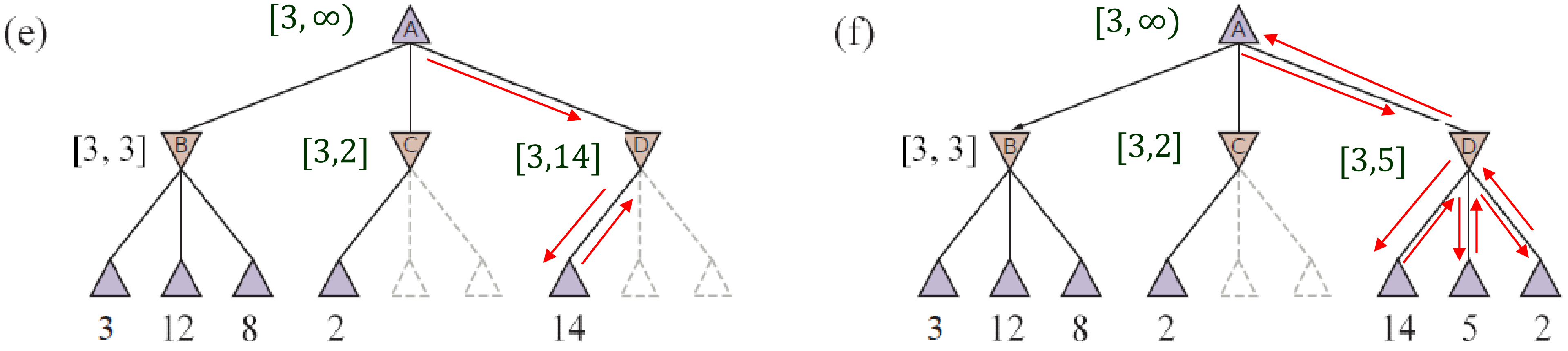
Cont’d



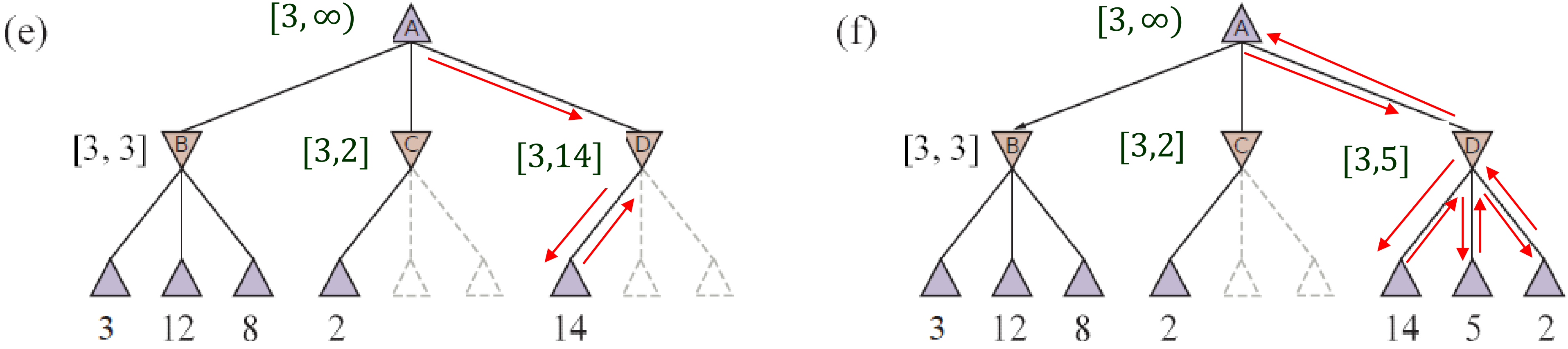
Cont’d



Cont’d

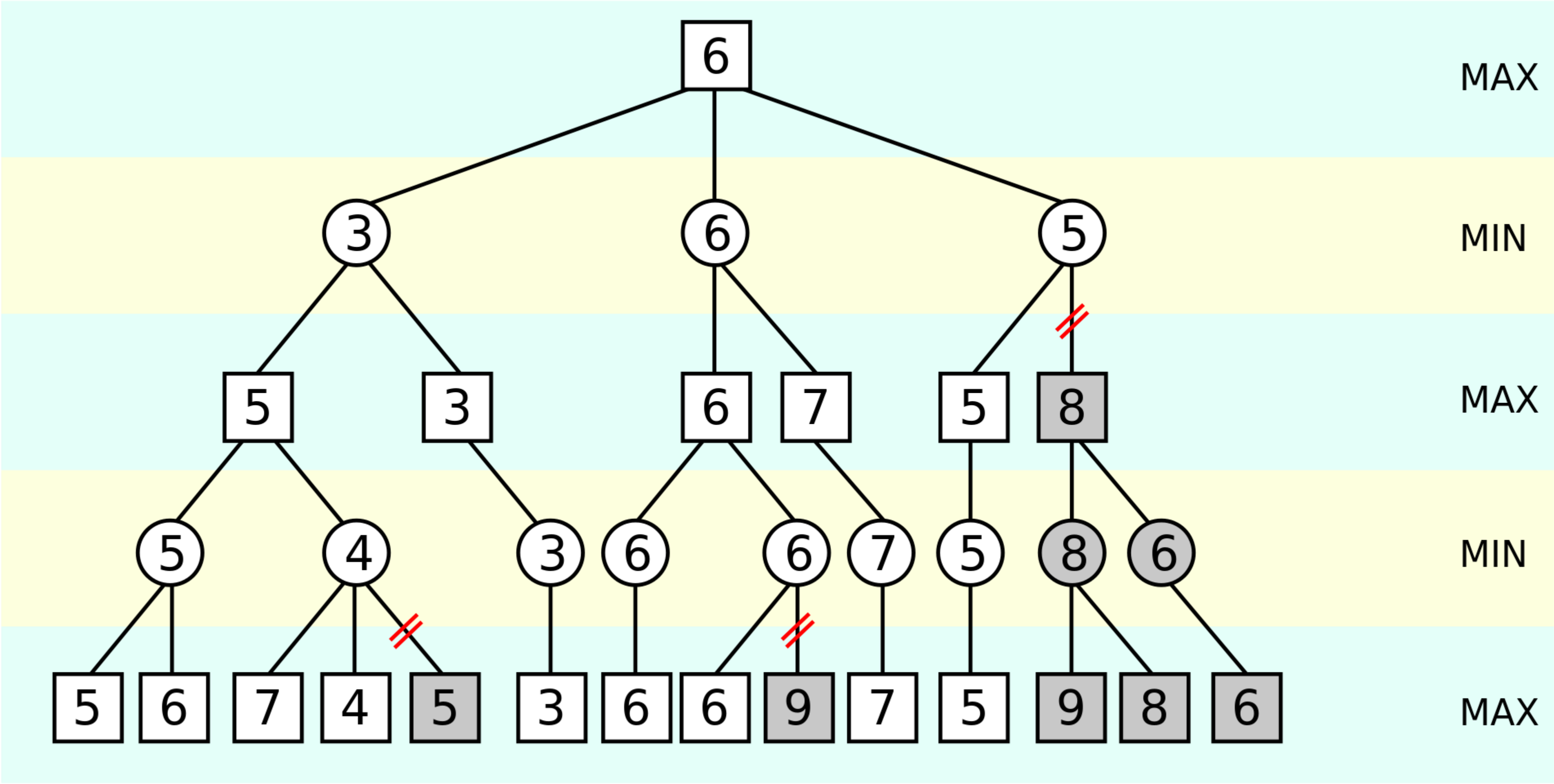


Cont'd

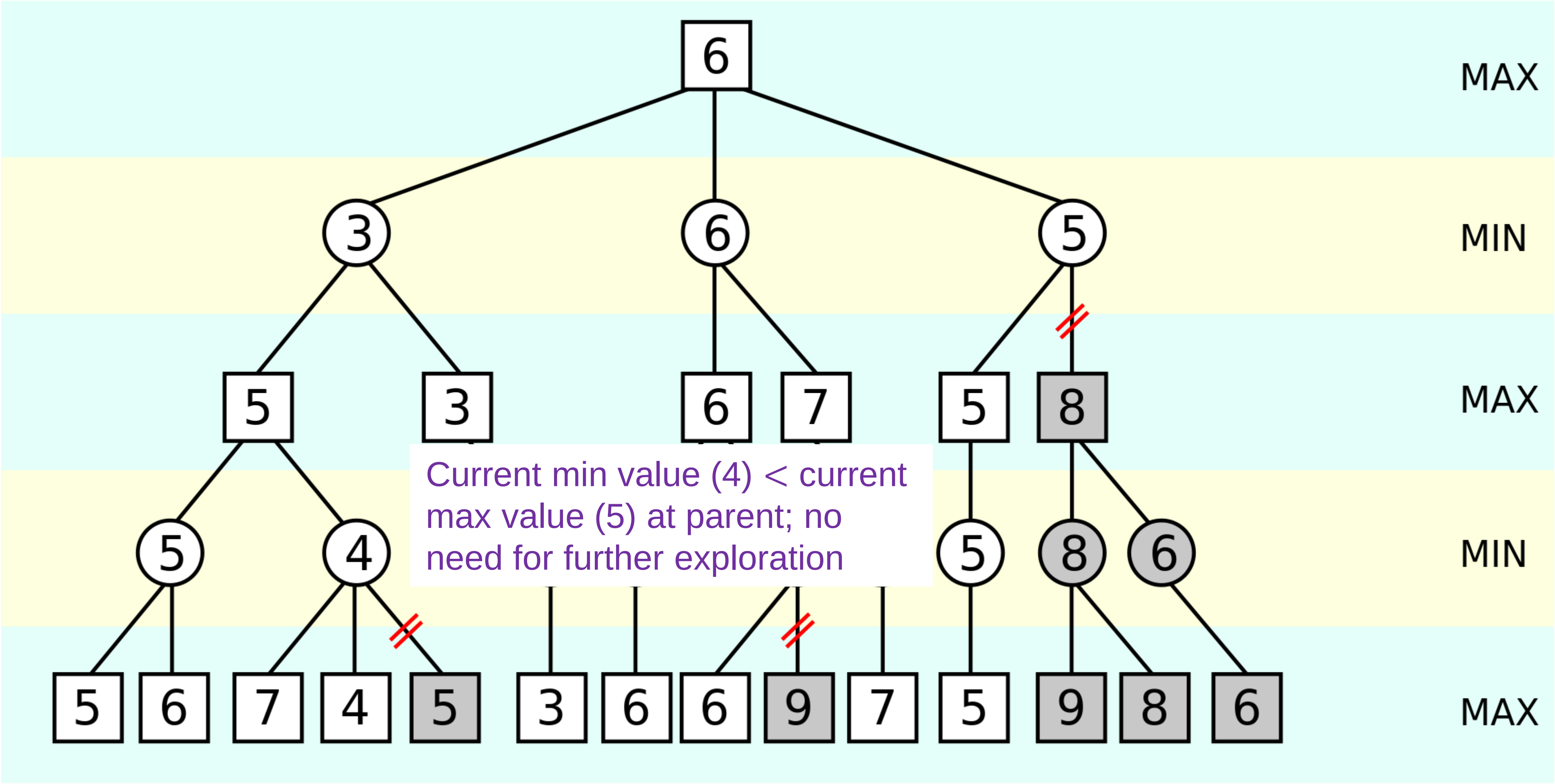


$$
\begin{aligned}
\text{MINIMAX}(\text{root}) &= \max(\min(3,12,8), \min(2,x,y), \min(14,5,2)) \\
&= \max(3, \min(2,x,y), 2) \\
&= \max(3, z, 2) \qquad \text{where } z = \min(2,x,y) \le 2 \\
&= 3.
\end{aligned}
$$

A Larger Example (Wikipedia)

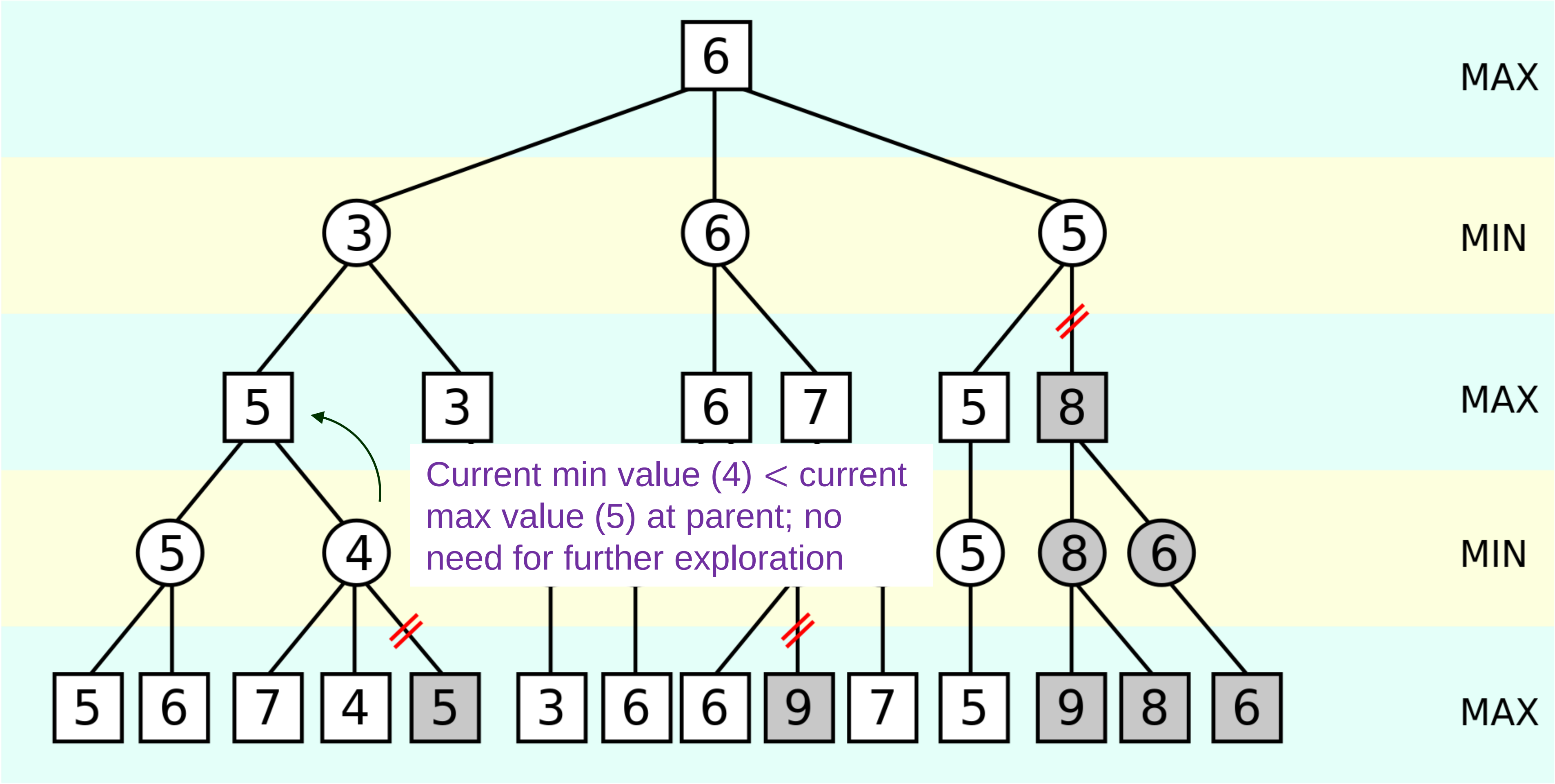


A Larger Example (Wikipedia)



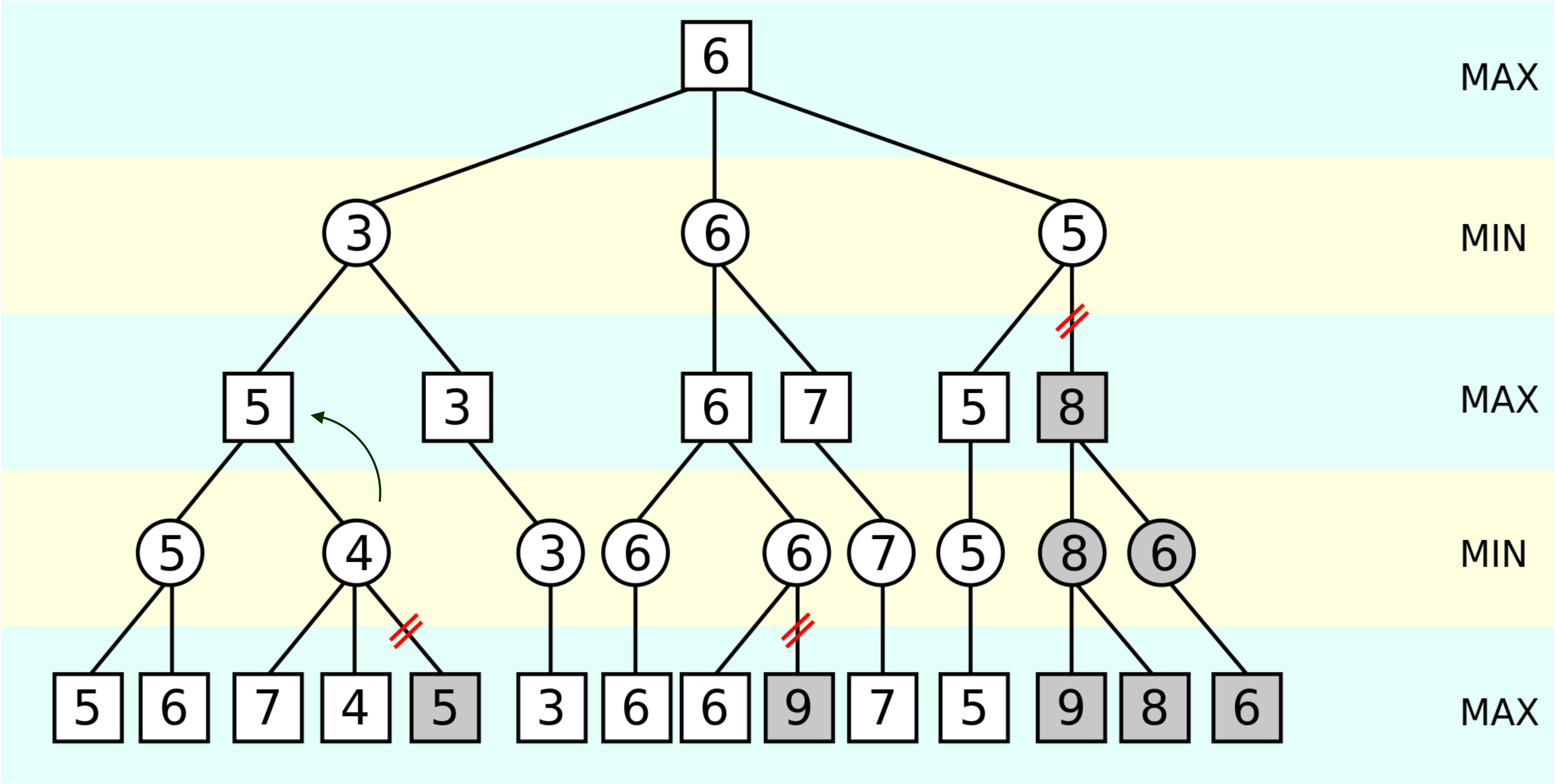
Current min value (4) < current max value (5) at parent; no need for further exploration

A Larger Example (Wikipedia)

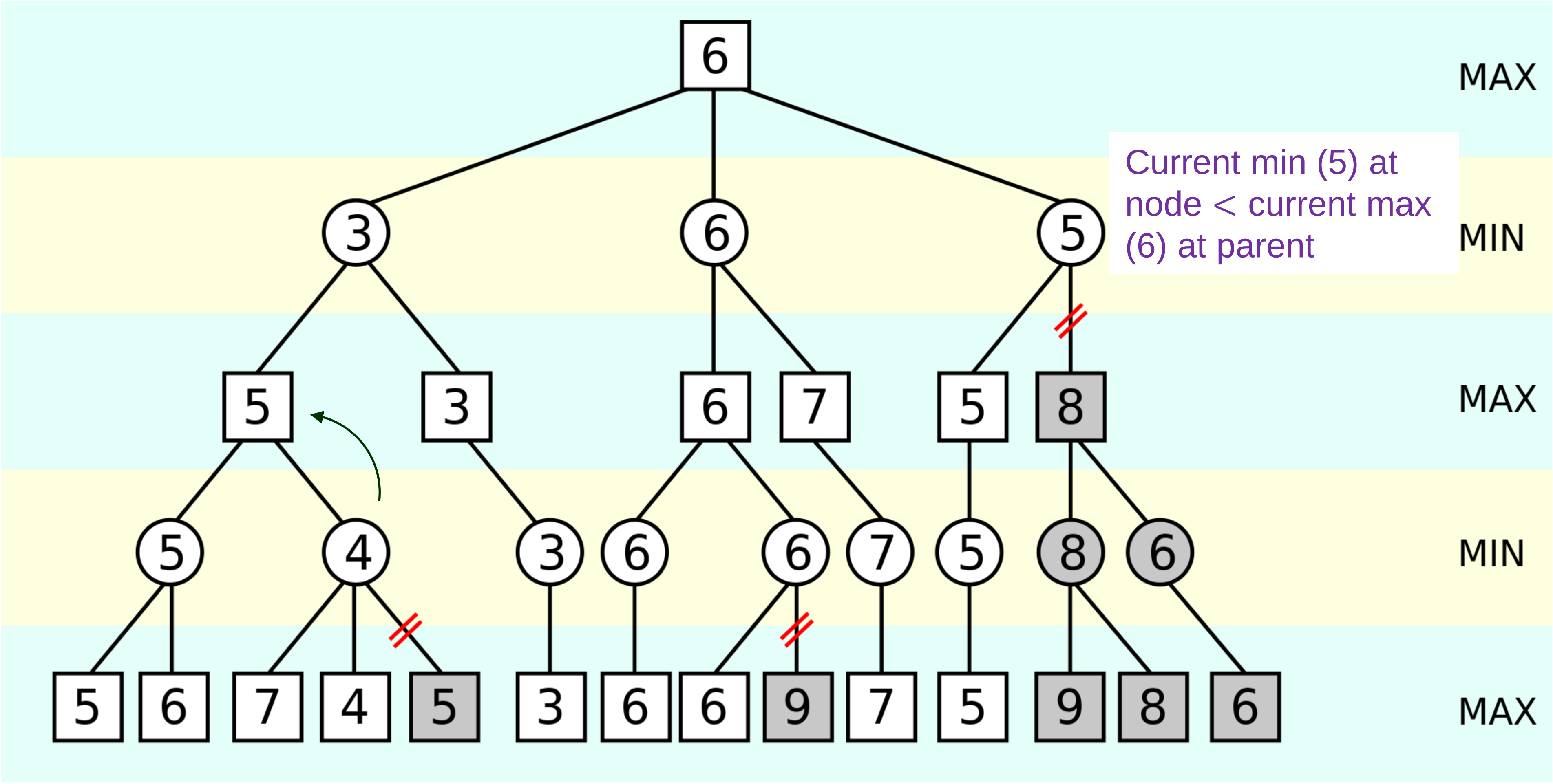


Current min value (4) < current max value (5) at parent; no need for further exploration

A Larger Example (Wikipedia)

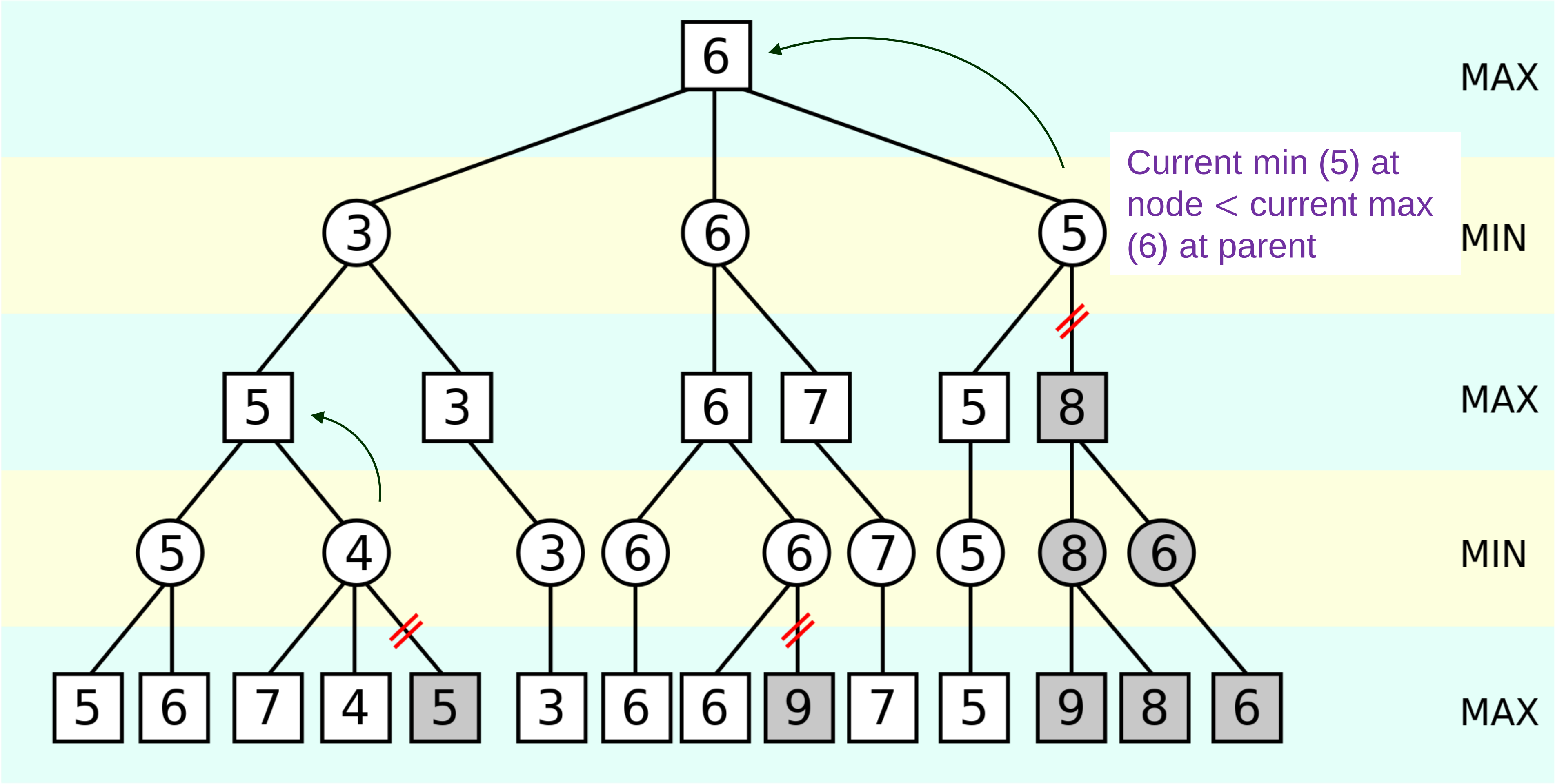


A Larger Example (Wikipedia)



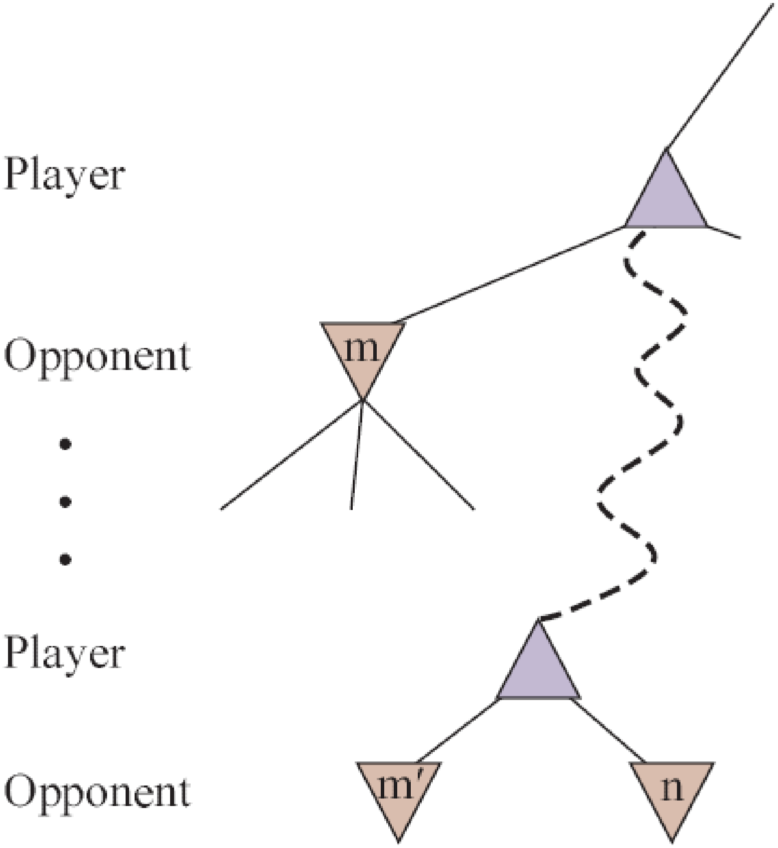
Current min (5) at node < current max (6) at parent

A Larger Example (Wikipedia)



Current min (5) at node < current max (6) at parent

General Case



The player will not move to node n if it has a better choice

- either at the same level (e.g., m')
- or at any node (e.g., m) higher up in the tree.

Prune n once we have found enough about it to reach the above conclusion.

Alpha and Beta Values

Alpha-beta pruning gets its name from two extra parameters α, β

$\alpha =$ the highest-value (i.e., the best choice) so far along a path for MAX.

Alpha and Beta Values

Alpha-beta pruning gets its name from two extra parameters α, β

$\alpha =$ the highest-value (i.e., the best choice) so far along a path for MAX.

eventual value $\geq \alpha$ “at least”

Alpha and Beta Values

Alpha-beta pruning gets its name from two extra parameters α, β

$\alpha =$ the highest-value (i.e., the best choice) so far along a path for MAX.

eventual value $\geq \alpha$ "at least"

$\beta =$ the lowest-value (i.e., the best choice) so far along a path for MIN.

eventual value $\leq \beta$ "at most"

Alpha and Beta Values

Alpha-beta pruning gets its name from two extra parameters α, β

$\alpha =$ the highest-value (i.e., the best choice) so far along a path for MAX.

eventual value $\geq \alpha$ “at least”

$\beta =$ the lowest-value (i.e., the best choice) so far along a path for MIN.

eventual value $\leq \beta$ “at most”

- ♦ Update the values of α and β as the search goes along.
- ♦ Prune the remaining branches at a MIN node with current value $\leq \alpha$ or a MAX node with current value $\geq \beta$.

Alpha and Beta Values

Alpha-beta pruning gets its name from two extra parameters α, β

$\alpha =$ the highest-value (i.e., the best choice) so far along a path for MAX.

eventual value $\geq \alpha$ "at least"

$\beta =$ the lowest-value (i.e., the best choice) so far along a path for MIN.

eventual value $\leq \beta$ "at most"

- Update the values of α and β as the search goes along.
- Prune the remaining branches at a MIN node with current value $\leq \alpha$ or a MAX node with current value $\geq \beta$.

Alpha-Beta Search Algorithm (3rd Edition of Textbook)

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


function MAX-VALUE (state,$\alpha, \beta$) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    $v \leftarrow -\infty$
    for each $a$ in ACTIONS(state) do
        $v \leftarrow$ MAX($v$, MIN-VALUE(RESULT($s, a$),$\alpha, \beta$))
        if $v \geq \beta$ then return $v$
        $\alpha \leftarrow$ MAX($\alpha, v$)
    return $v$


function MIN-VALUE (state,$\alpha, \beta$) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    $v \leftarrow +\infty$
    for each $a$ in ACTIONS(state) do
        $v \leftarrow$ MIN($v$, MAX-VALUE(RESULT($s, a$),$\alpha, \beta$))
        if $v \leq \alpha$ then return $v$
        $\beta \leftarrow$ MIN($\beta, v$)
    return $v$
```

Alpha-Beta Search Algorithm (3rd Edition of Textbook)

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


function MAX-VALUE (state,$\alpha, \beta$) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    $v \leftarrow -\infty$
    for each $a$ in ACTIONS(state) do
        $v$ ←MAX($v$, MIN-VALUE(RESULT($s, a$),$\alpha, \beta$))
        if $v \geq \beta$ then return $v$    // pruning
        $\alpha$ ←MAX($\alpha, v$)
    return $v$


function MIN-VALUE (state,$\alpha, \beta$) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    $v \leftarrow +\infty$
    for each $a$ in ACTIONS(state) do
        $v$ ←MIN($v$, MAX-VALUE(RESULT($s, a$),$\alpha, \beta$))
        if $v \leq \alpha$ then return $v$
        $\beta$ ←MIN($\beta, v$)
    return $v$
```

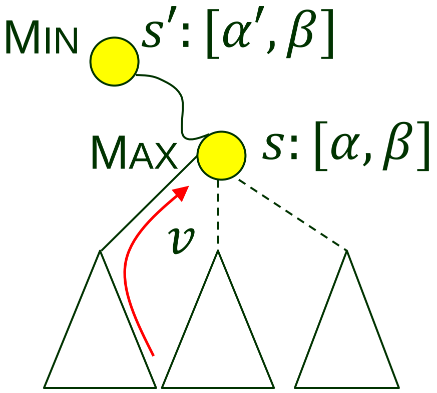
Alpha-Beta Search Algorithm (3rd Edition of Textbook)

```
function Alpha-Beta-Search(state) returns an action
    $v$ ←Max-Value(state, $-\infty, +\infty$)
    return the action in Actions(state) with value $v$


function Max-Value (state,$\alpha, \beta$) returns a utility value
    if Terminal-Test(state) then return Utility(state)
    $v \leftarrow -\infty$
    for each $a$ in Actions(state) do
        $v$ ←Max($v$, Min-Value(Result($s, a$),$\alpha, \beta$))
        if $v \geq \beta$ then return $v$    // pruning
        $\alpha$ ←Max($\alpha, v$)  // new $\alpha$ to be passed on to the rest Min-Value
    return $v$                   // calls within the for loop.


function Min-Value (state,$\alpha, \beta$) returns a utility value
    if Terminal-Test(state) then return Utility(state)
    $v \leftarrow +\infty$
    for each $a$ in Actions(state) do
        $v$ ←Min($v$, Max-Value(Result($s, a$),$\alpha, \beta$))
        if $v \leq \alpha$ then return $v$
        $\beta$ ←Min($\beta, v$)
    return $v$
```

Alpha-Beta Search Algorithm (3rd Edition of Textbook)

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

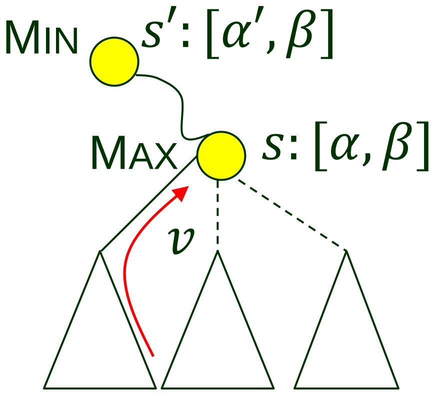
```
function MAX-VALUE (state,$\alpha, \beta$) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    $v \leftarrow -\infty$
    for each $a$ in ACTIONS(state) do
        $v$ ←MAX($v$, MIN-VALUE(RESULT($s, a$),$\alpha, \beta$))
        if $v \geq \beta$ then return $v$    // pruning
        $\alpha$ ←MAX($\alpha, v$)  // new $\alpha$ to be passed on to the rest MIN-VALUE
    return $v$                      // calls within the for loop.
```



```
function MIN-VALUE (state,$\alpha, \beta$) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    $v \leftarrow +\infty$
    for each $a$ in ACTIONS(state) do
        $v$ ←MIN($v$, MAX-VALUE(RESULT($s, a$),$\alpha, \beta$))
        if $v \leq \alpha$ then return $v$
        $\beta$ ←MIN($\beta, v$)
    return $v$
```

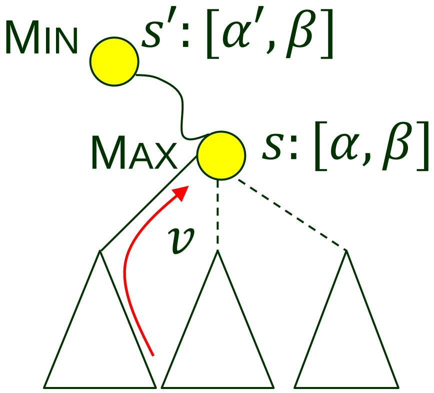
Alpha-Beta Search Algorithm (3rd Edition of Textbook)

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

// no change of $\beta$ value within MAX-VALUE()
function MAX-VALUE (state,$\alpha, \beta$) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    $v \leftarrow -\infty$
    for each $a$ in ACTIONS(state) do
        $v$ ←MAX($v$, MIN-VALUE(RESULT($s, a$),$\alpha, \beta$))
        if $v \geq \beta$ then return $v$      // pruning
        $\alpha$ ←MAX($\alpha, v$)  // new $\alpha$ to be passed on to the rest MIN-VALUE
    return $v$                  // calls within the for loop.

function MIN-VALUE (state,$\alpha, \beta$) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    $v \leftarrow +\infty$
    for each $a$ in ACTIONS(state) do
        $v$ ←MIN($v$, MAX-VALUE(RESULT($s, a$),$\alpha, \beta$))
        if $v \leq \alpha$ then return $v$
        $\beta$ ←MIN($\beta, v$)
    return $v$
```



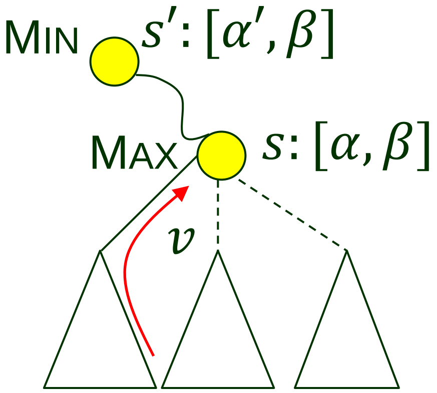
Alpha-Beta Search Algorithm (3rd Edition of Textbook)

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

// no change of $\beta$ value within MAX-VALUE()
function MAX-VALUE (state,$\alpha, \beta$) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    $v \leftarrow -\infty$
    for each $a$ in ACTIONS(state) do
        $v$ ←MAX($v$, MIN-VALUE(RESULT($s, a$),$\alpha, \beta$))
        if $v \geq \beta$ then return $v$    // pruning
        $\alpha$ ←MAX($\alpha, v$)  // new $\alpha$ to be passed on to the rest MIN-VALUE
    return $v$                      // calls within the for loop.

function MIN-VALUE (state,$\alpha, \beta$) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    $v \leftarrow +\infty$
    for each $a$ in ACTIONS(state) do
        $v$ ←MIN($v$, MAX-VALUE(RESULT($s, a$),$\alpha, \beta$))
        if $v \leq \alpha$ then return $v$    // pruning
        $\beta$ ←MIN($\beta, v$)
    return $v$
```



Alpha-Beta Search Algorithm (3rd Edition of Textbook)

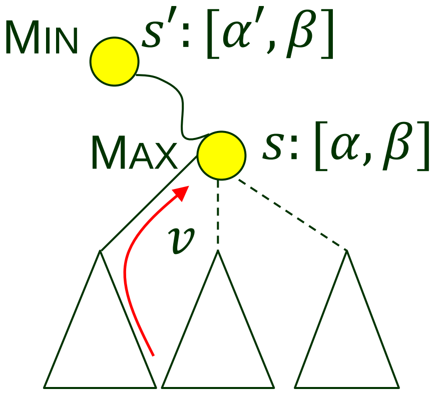
```
function ALPHA-BETA-SEARCH(state) returns an action
    $v$ ←MAX-VALUE(state, $-\infty, +\infty$)
    return the action in ACTIONS(state) with value $v$

// no change of $\beta$ value within MAX-VALUE()
function MAX-VALUE (state,$\alpha, \beta$) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    $v \leftarrow -\infty$
    for each $a$ in ACTIONS(state) do
        $v$ ←MAX($v$, MIN-VALUE(RESULT($s, a$),$\alpha, \beta$))
        if $v \geq \beta$ then return $v$    // pruning
        $\alpha$ ←MAX($\alpha, v$)  // new $\alpha$ to be passed on to the rest MIN-VALUE
    return $v$                      // calls within the for loop.
```



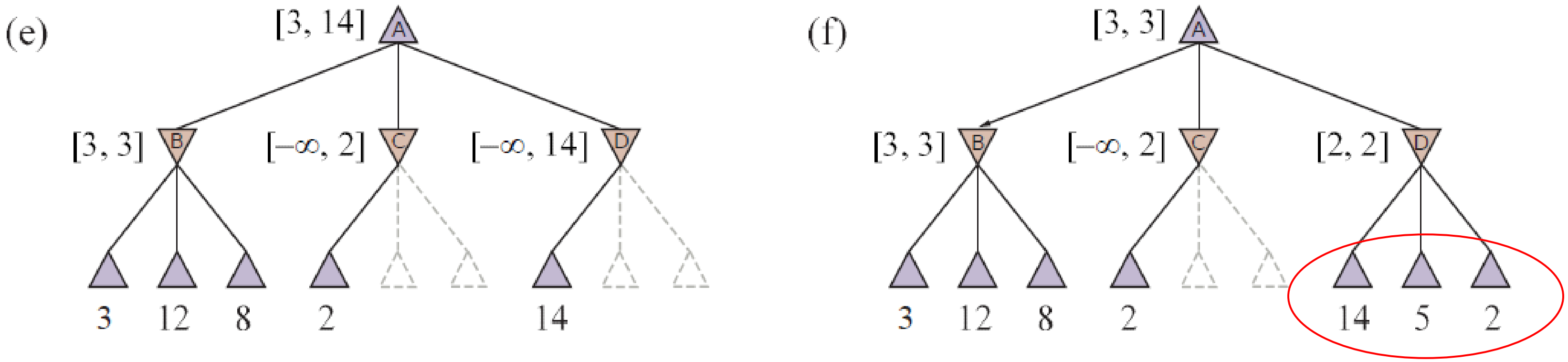
```
function MIN-VALUE (state,$\alpha, \beta$) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    $v \leftarrow +\infty$
    for each $a$ in ACTIONS(state) do
        $v$ ←MIN($v$, MAX-VALUE(RESULT($s, a$),$\alpha, \beta$))
        if $v \leq \alpha$ then return $v$    // pruning
        $\beta$ ←MIN($\beta, v$)  // new $\beta$ to be passed on to the rest MIN-VALUE calls within the for loop.
    return $v$
```

Alpha-Beta Search Algorithm (3rd Edition of Textbook)

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

// no change of $\beta$ value within MAX-VALUE()
function MAX-VALUE (state,$\alpha, \beta$) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    $v \leftarrow -\infty$
    for each $a$ in ACTIONS(state) do
        $v$ ←MAX($v$, MIN-VALUE(RESULT($s, a$),$\alpha, \beta$))
        if $v \geq \beta$ then return $v$    // pruning
        $\alpha$ ←MAX($\alpha, v$)  // new $\alpha$ to be passed on to the rest MIN-VALUE
    return $v$                    // calls within the for loop.

// no change of $\alpha$ value within MIN-VALUE()
function MIN-VALUE (state,$\alpha, \beta$) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    $v \leftarrow +\infty$
    for each $a$ in ACTIONS(state) do
        $v$ ←MIN($v$, MAX-VALUE(RESULT($s, a$),$\alpha, \beta$))
        if $v \leq \alpha$ then return $v$    // pruning
        $\beta$ ←MIN($\beta, v$)  // new $\beta$ to be passed on to the rest MIN-VALUE calls within the for loop.
    return $v$
```

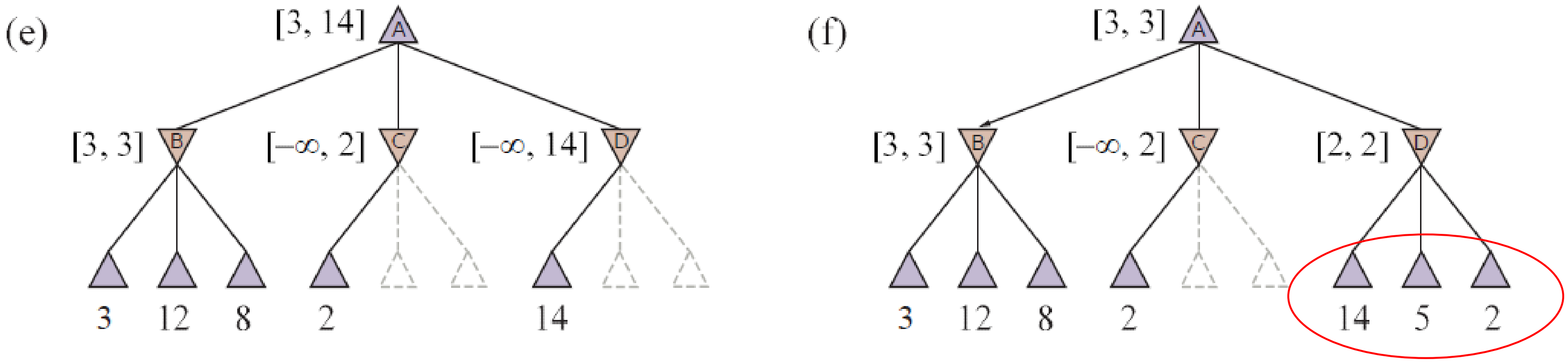


II. Move Ordering



Successors 14 and 5 would’ve been pruned had 2 been generated first.

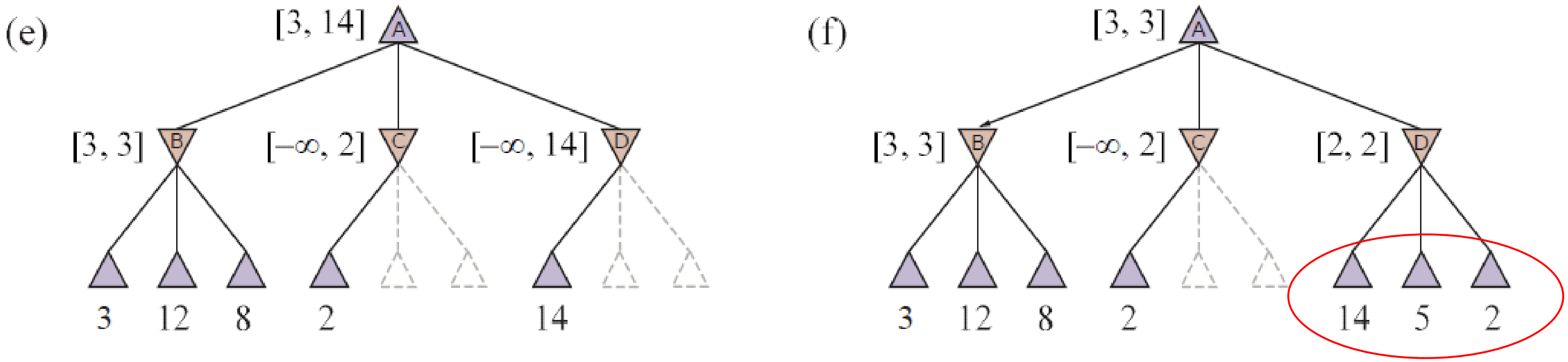
II. Move Ordering



Successors 14 and 5 would’ve been pruned had 2 been generated first.

- Effectiveness of pruning is highly dependent on the order in which successors are generated.

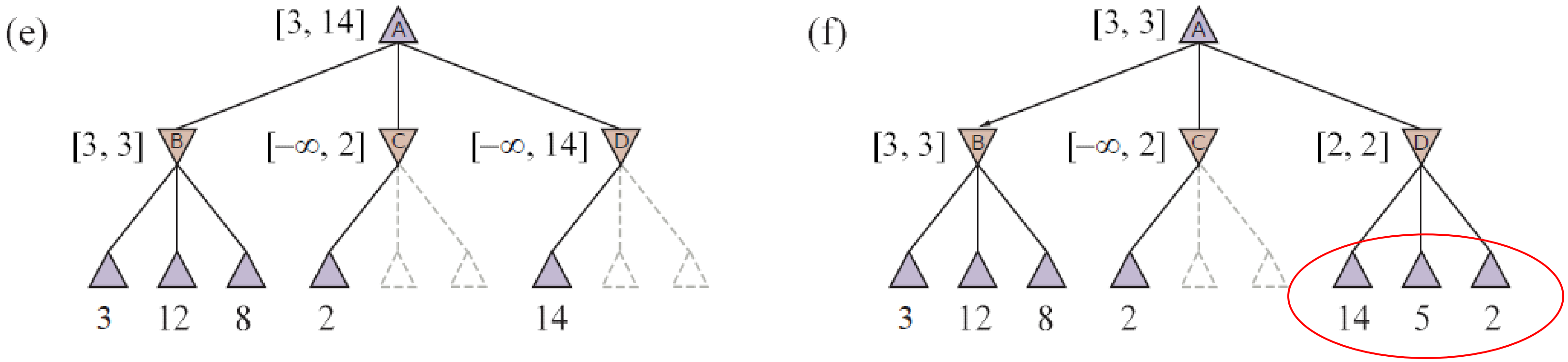
II. Move Ordering



Successors 14 and 5 would've been pruned had 2 been generated first.

- Effectiveness of pruning is highly dependent on the order in which successors are generated.
- "Perfect ordering" has effective branching factor $\sqrt{b}$, which limits examination to only $O(b^{m/2})$ nodes compared to $O(b^m)$ for minimax.

II. Move Ordering



Successors 14 and 5 would’ve been pruned had 2 been generated first.

- Effectiveness of pruning is highly dependent on the order in which successors are generated.
- “Perfect ordering” has effective branching factor $\sqrt{b}$, which limits examination to only $O(b^{m/2})$ nodes compared to $O(b^m)$ for minimax.
- $O(b^{3m/4})$ nodes for random move ordering.

Heuristic Alpha-Beta Tree Search

- Cut off the search early by applying a heuristic evaluation function.
- Replace UTILITY with EVAL, which estimates a state’s utility.

Heuristic Alpha-Beta Tree Search

- ♦ Cut off the search early by applying a heuristic evaluation function.
- ♦ Replace UTILITY with EVAL, which estimates a state's utility.

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

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

Evaluation Functions

EVAL(s, p) returns an estimate of the expected utility s to player p.

- EVAL$(s, p) =$ UTILITY(s, p) if s is terminal;
- UTILITY$(loss, p) \leq$ EVAL$(s, p) \leq$ UTILITY(win, p) if s is nonterminal.

Evaluation Functions

EVAL(s, p) returns an estimate of the expected utility s to player p.

- EVAL(s, p) = UTILITY(s, p) if s is terminal;
- UTILITY$(loss, p) \leq$ EVAL$(s, p) \leq$ UTILITY(win, p) if s is nonterminal.

Criteria:

- ♦ No excessive computation time.
- ♦ Strong correlation with actual chances of winning.

Eval Function: State Categorization

- Calculate various features of the state (e.g., #pawns, #queens in chess).

Eval Function: State Categorization

- Calculate various features of the state (e.g., #pawns, #queens in chess).
- Define categories (equivalent classes) of states (e.g., all two-pawn vs one-pawn endgames).
  - Each category may contain states leading to wins, draws, and losses.
  - Nevertheless, all such states have the same feature values.

Eval Function: State Categorization

- Calculate various features of the state (e.g., #pawns, #queens in chess).
- Define categories (equivalent classes) of states (e.g., all two-pawn vs one-pawn endgames).
  - Each category may contain states leading to wins, draws, and losses.
  - Nevertheless, all such states have the same feature values.
- Determine an expected value for each category.

Eval Function: State Categorization

- Calculate various features of the state (e.g., #pawns, #queens in chess).
- Define categories (equivalent classes) of states (e.g., all two-pawn vs one-pawn endgames).
  - Each category may contain states leading to wins, draws, and losses.
  - Nevertheless, all such states have the same feature values.
- Determine an expected value for each category.

e.g., two-pawns vs. one-pawn

1 ↤	wins	82%
0 ↤	losses	2%
0.5 ↤	draws	16%

$$(0.82 \times 1) + (0.02 \times 0) + (0.16 \times 0.5) = 0.9$$

Eval Function: State Categorization

- Calculate various features of the state (e.g., #pawns, #queens in chess).
- Define categories (equivalent classes) of states (e.g., all two-pawn vs one-pawn endgames).
  - Each category may contain states leading to wins, draws, and losses.
  - Nevertheless, all such states have the same feature values.
- Determine an expected value for each category.

e.g., two-pawns vs. one-pawn

$1 \mapsfrom$	wins	82%
$0 \mapsfrom$	losses	2%
$0.5 \mapsfrom$	draws	16%

$$(0.82 \times 1) + (0.02 \times 0) + (0.16 \times 0.5) = 0.9$$

- Too many categories and too much dependence on experience.

Eval Function: Feature Combination

Compute separate numerical contributions from each feature and combine them.

$$\boxed{\textsc{Eval}(s) = \underset{\substack{\uparrow \\ \text{weight}}}{w_1} \underset{\substack{\uparrow \\ \text{e.g., \#pawns in chess}}}{f_1}(s) + w_2 f_2(s) + \cdots + w_n f_n(s)}$$

Eval Function: Feature Combination

Compute separate numerical contributions from each feature and combine them.

$$\boxed{\textsc{Eval}(s) = \underset{\substack{\uparrow \\ \text{weight}}}{w_1}\,\underset{\substack{\uparrow \\ \text{e.g., \#pawns in chess}}}{f_1}(s) + w_2 f_2(s) + \cdots + w_n f_n(s)}$$

♠ Assumes independent feature contributions.

Eval Function: Feature Combination

Compute separate numerical contributions from each feature and combine them.

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

weight

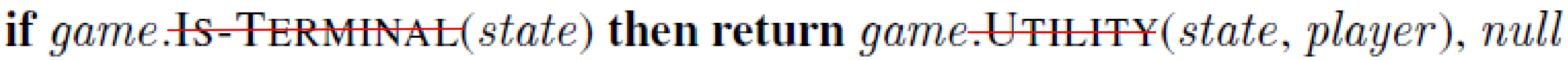
e.g., #pawns in chess

♠ Assumes independent feature contributions.

♦ Use a nonlinear feature combination.

e.g., two bishops might be worth more than twice the value of a single bishop.

Cutting off Search

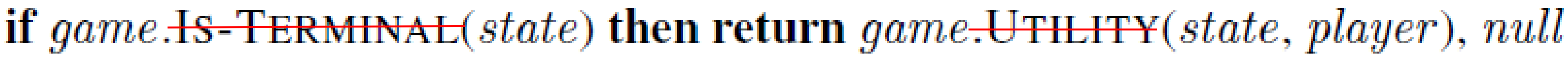


```
if $game$.~~IS-TERMINAL~~($state$) then return $game$.~~UTILITY~~($state$, $player$), $null$
```

IS-CUTOFF

EVAL

Cutting off Search



if *game*.~~IS-TERMINAL~~(*state*) **then return** *game*.~~UTILITY~~(*state*, *player*), *null*

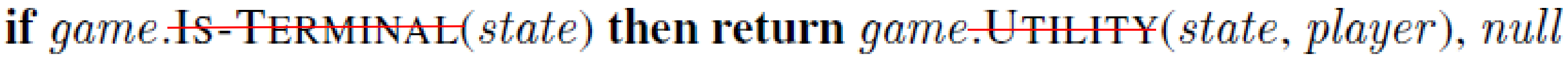
IS-CUTOFF EVAL

Some strategies:

- Set a fixed depth limit d to control the amount of search.

IS-CUTOFF returns true if depth $> \ d$.

Cutting off Search



if *game*.~~Is-Terminal~~(*state*) **then return** *game*.~~Utility~~(*state*, *player*), *null*

Is-Cutoff   Eval

Some strategies:

- Set a fixed depth limit d to control the amount of search.

  Is-Cutoff returns true if depth $> d$.

- Apply iterative deepening:

  When time runs out, returns the move selected by the deepest completed search.

Real-Time Decisions

♦ Minimax with alpha-beta pruning.

♦ Extensively tuned evaluation function.

♦ Pruning heuristics.

♦ A transposition table of repeated states and evaluations.

- To avoid re-searching the game tree below that state.

♦ A large database of optimal opening and endgame moves.

- Table lookup instead of search.
- Chess endgames with up to 7 pieces solved.

♣ Minimax unsuccessful in Go.

Real-Time Decisions

♦ Minimax with alpha-beta pruning.

♦ Extensively tuned evaluation function.

♦ Pruning heuristics.

♦ A transposition table of repeated states and evaluations.

- To avoid re-searching the game tree below that state.

♦ A large database of optimal opening and endgame moves.

- Table lookup instead of search.
- Chess endgames with up to 7 pieces solved.

♣ Minimax unsuccessful in Go.