A Formal Basis for the Heuristic Determination of Minimum Cost Paths

PETER E. HART, MEMBER, IEEE, NILS J. NILSSON, MEMBER, IEEE, AND BERTRAM RAPHAEL

Search Algorithm A:*

1) Mark s "open" and calculate $\hat{f}(s)$.

2) Select the open node n whose value of $\hat{f}$ is smallest. Resolve ties arbitrarily, but always in favor of any node $n \in T$.

3) If $n \in T$, mark n "closed" and terminate the algorithm.

4) Otherwise, mark n closed and apply the successor operator Γ to n. Calculate $\hat{f}$ for each successor of n and mark as open each successor not already marked closed. Remark as open any closed node n_i which is a successor of n and for which $\hat{f}(n_i)$ is smaller now than it was when n_i was marked closed. Go to Step 2.

*C. Proof of the Optimality of A**

The next lemma makes the important observation about the operation of A^* that, under the consistency assumption, if node n is closed, then $\hat{g}(n) = g(n)$. This fact is important for two reasons. First, it is used in the proof of the theorem about the optimality of A^* to follow, and second, it states that A^* need never reopen a closed node. That is, if A^* expands a node, then the optimal path to that node has already been found. Thus, in Step 4 of the algorithm A^*, the provision for reopening a closed node is vacuous and may be eliminated.

Adversarial Search & Pruning

Virginia Tech CS 4804
Introduction to Artificial Intelligence

Plan

- Minimax
- Pruning minimax search

Game Representation

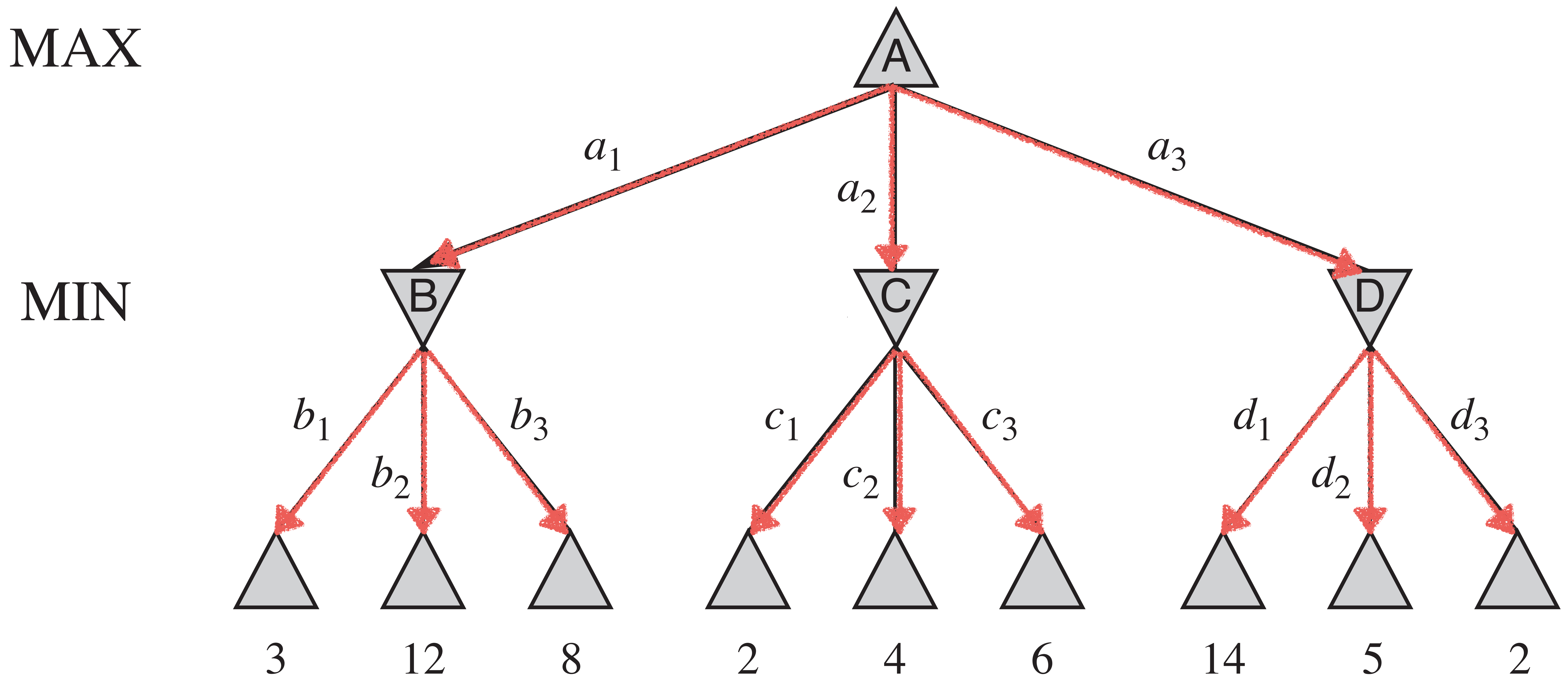
- **zero-sum** games of **perfect information**
- Two players: MAX vs MIN
- Players alternate actions: state space transitions

Representation Elements

- PLAYER(*s*): which player chooses the action in state *s*
- ACTIONS(*s*): what actions are available from state *s*
- RESULT(*s*,*a*): the state that results from action a in state *s*
- TERMINAL-TEST(*s*): whether state *s* is a terminal state
- UTILITY(*s*): the value of state *s*, usually only if terminal

Minimax Strategy

- Choose best move assuming opponent plays **optimally**
  - i.e., opponent also uses minimax
- MINIMAX(*s*) =
  if TERMINAL-TEST(*s*) then UTILITY(*s*)
  if PLAYER(*s*) = MAX then
  max of MINIMAX(RESULT(*s*,*a*)) for *a* in ACTIONS(s)
  if PLAYER(s) = MIN then
  min of MINIMAX(RESULT(*s*,*a*)) for *a* in ACTIONS(s)



- MINIMAX(*s*) =

  if TERMINAL-TEST(*s*) then UTILITY(*s*)

  if PLAYER(*s*) = MAX then

  max of MINIMAX(RESULT(*s*,*a*)) for *a* in ACTIONS(s)

  if PLAYER(s) = MIN then

  min of MINIMAX(RESULT(*s*,*a*)) for *a* in ACTIONS(s)

function MINIMAX-DECISION($state$) **returns** *an action*
  return $\arg\max_{a \in \text{ACTIONS}(s)}$ MIN-VALUE(RESULT($state, a$))

---

function MAX-VALUE($state$) **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)))
  return v

---

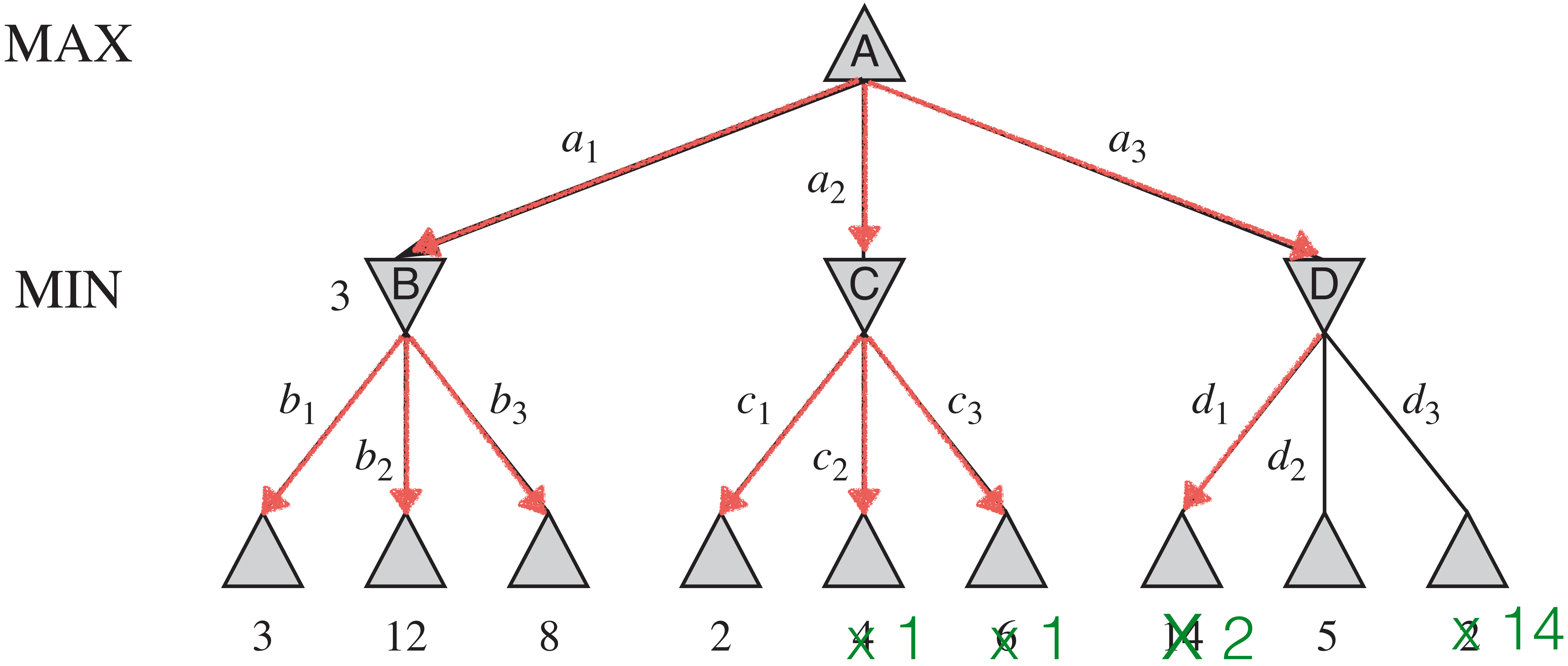
function MIN-VALUE($state$) **returns** *a utility value*
  if TERMINAL-TEST($state$) **then return** UTILITY($state$)
  $v \leftarrow \infty$
  for each a **in** ACTIONS($state$) **do**
    $v \leftarrow$ MIN(v, MAX-VALUE(RESULT(s, a)))
  return v

What about the MINIMAX algorithm does not work if the game is **not** zero-sum?

How can we adjust the algorithm to address this problem?

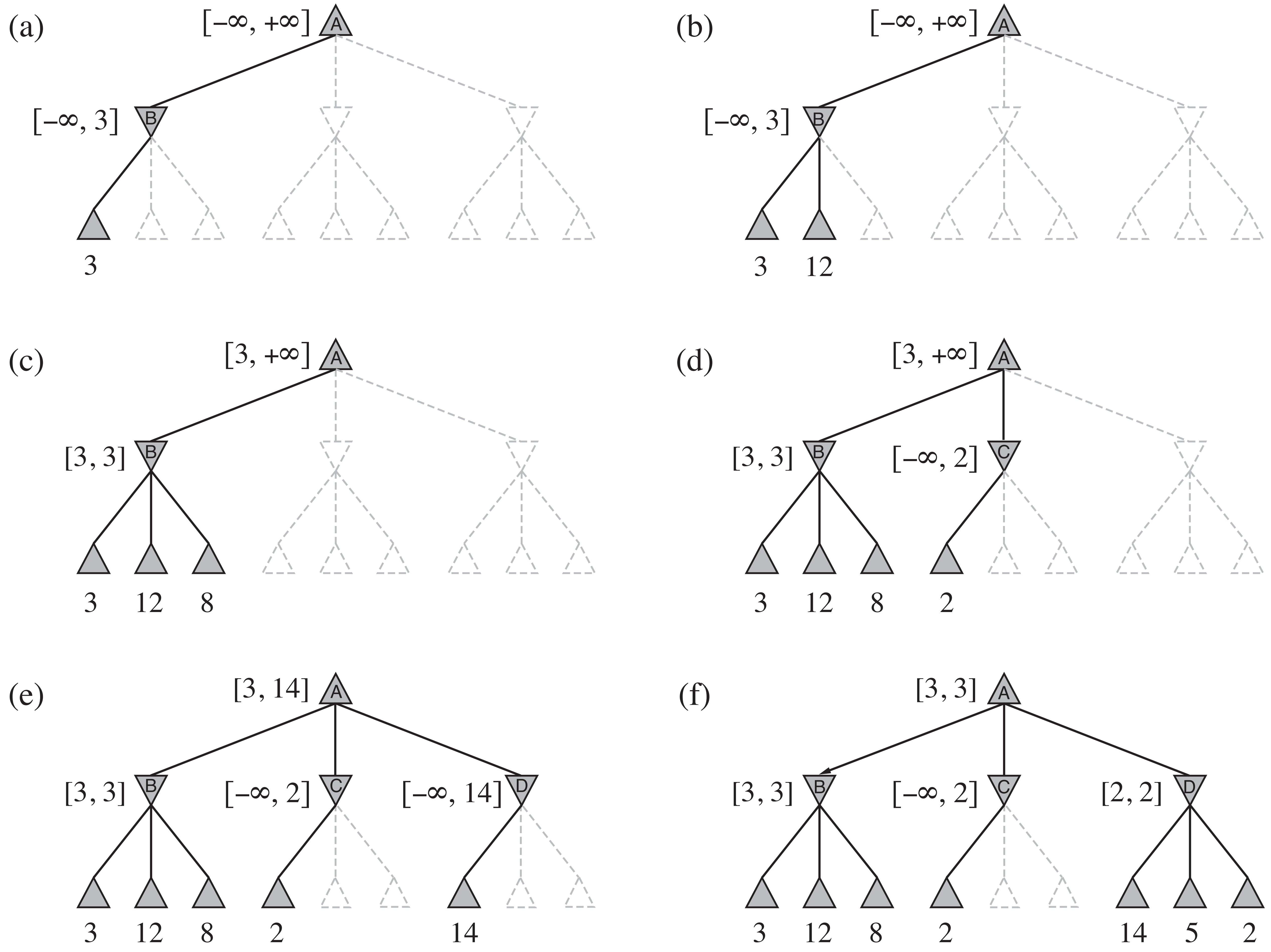
For every game tree, the utility obtained by MAX using minimax decisions against a **suboptimal** MIN will never be lower than the utility obtained against an optimal MIN

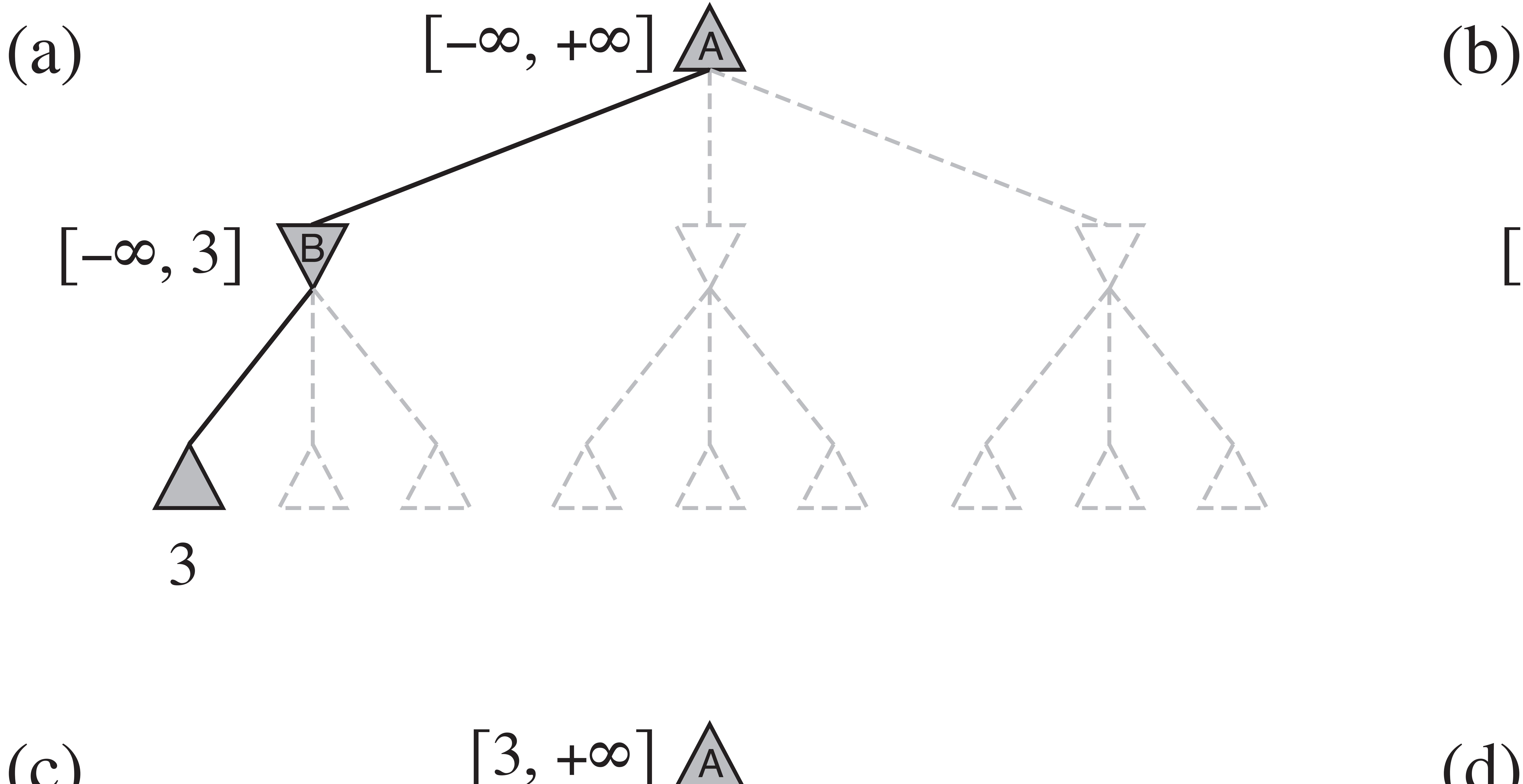
Pruning


MAX
MIN
A
a_1
a_2
a_3
3
B
C
D
b_1
b_2
b_3
c_1
c_2
c_3
d_1
d_2
d_3
3
12
8
2
~~4~~ 1
~~6~~ 1
~~14~~ 2
5
~~2~~ 14

Alpha-Beta Pruning

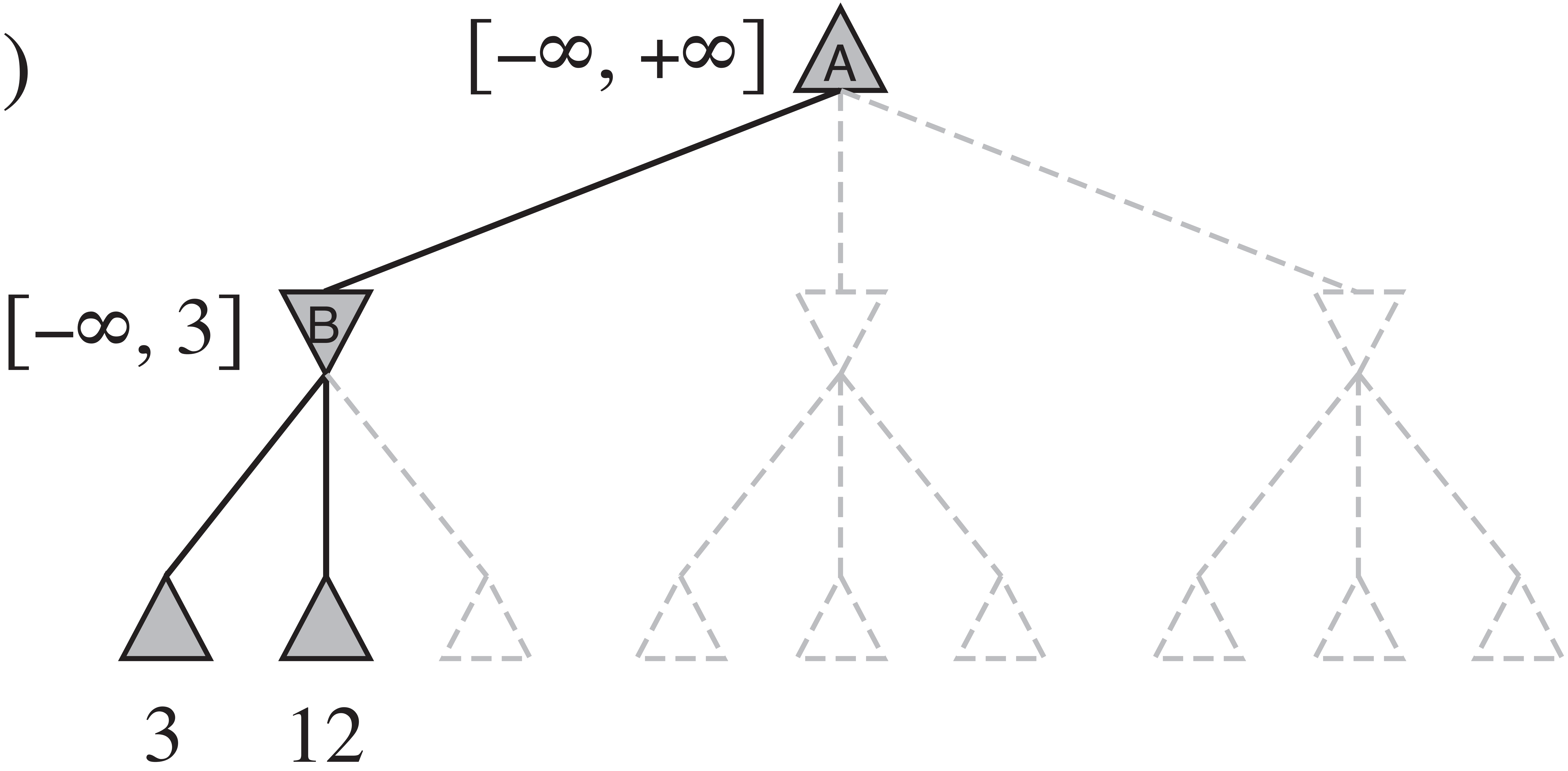
- [alpha, beta]
  alpha = upper-bound on minimax value
  beta = lower-bound on minimax value



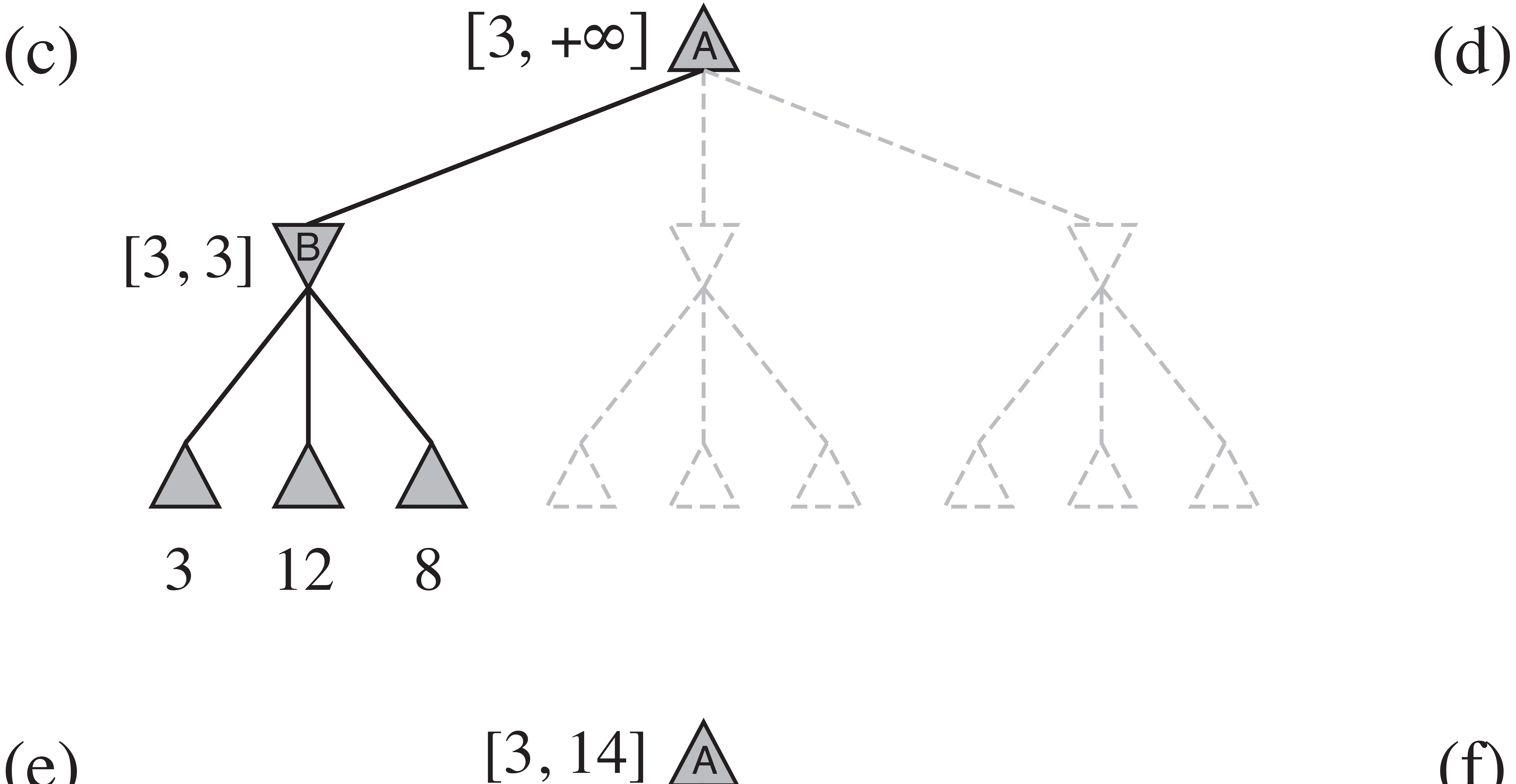


(a)
[−∞, +∞]
A
(b)
[−∞, 3]
B
3
(c)
[3, +∞]
A
(d)

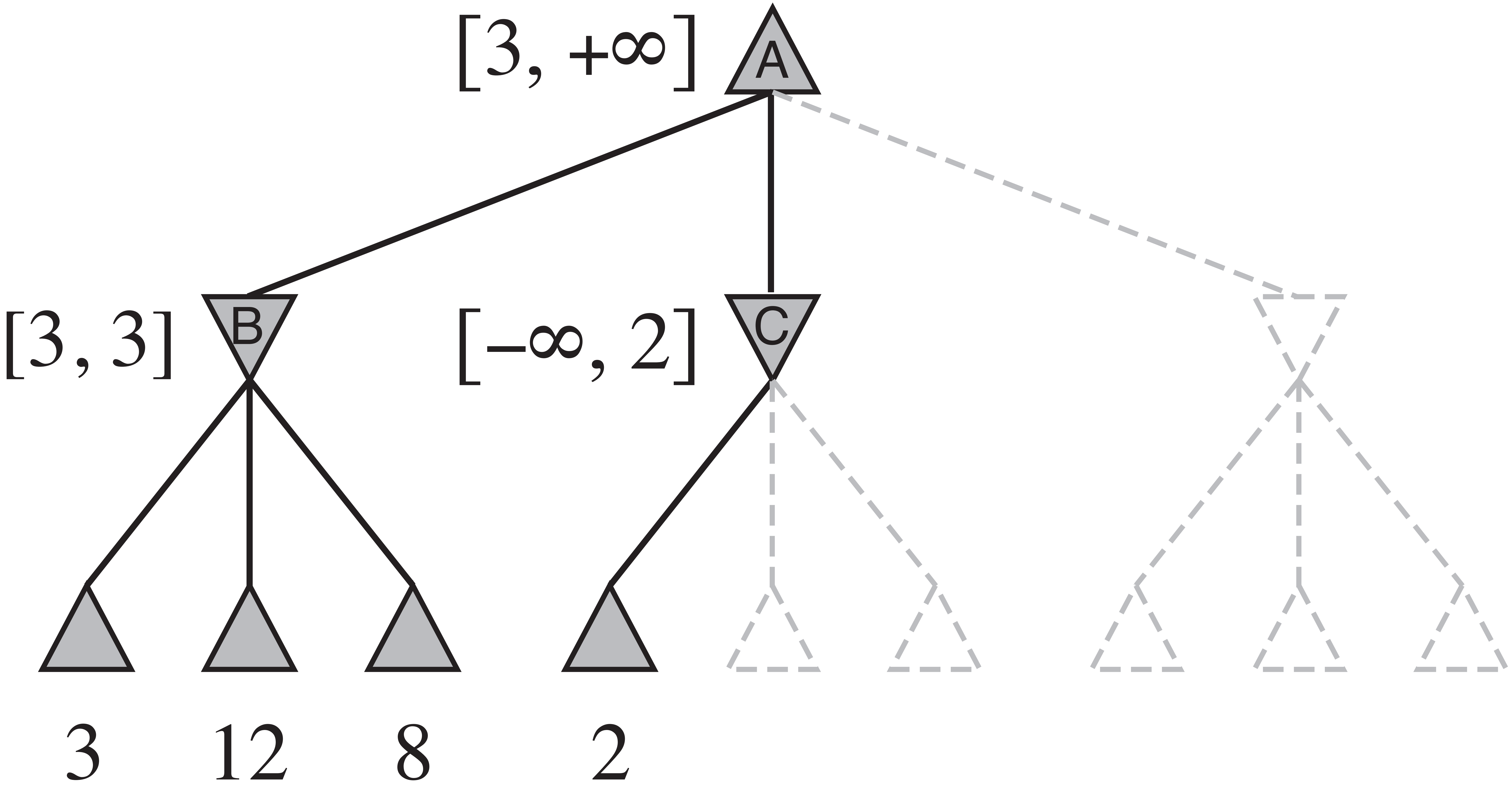
(b)



(d)


(c)
[3, +∞]
A
(d)
[3, 3]
B
3
12
8
(e)
[3, 14]
A
(f)

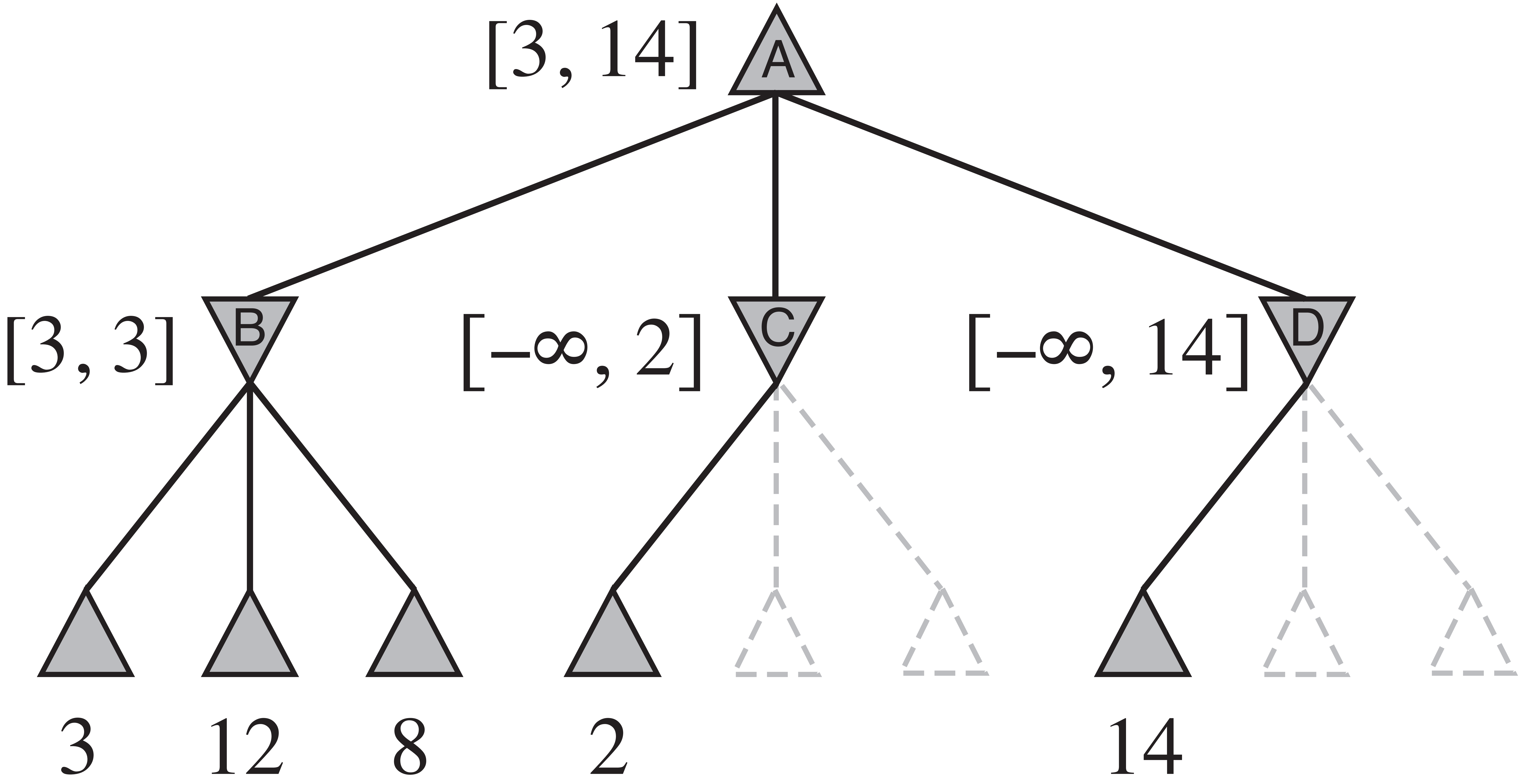
(d)

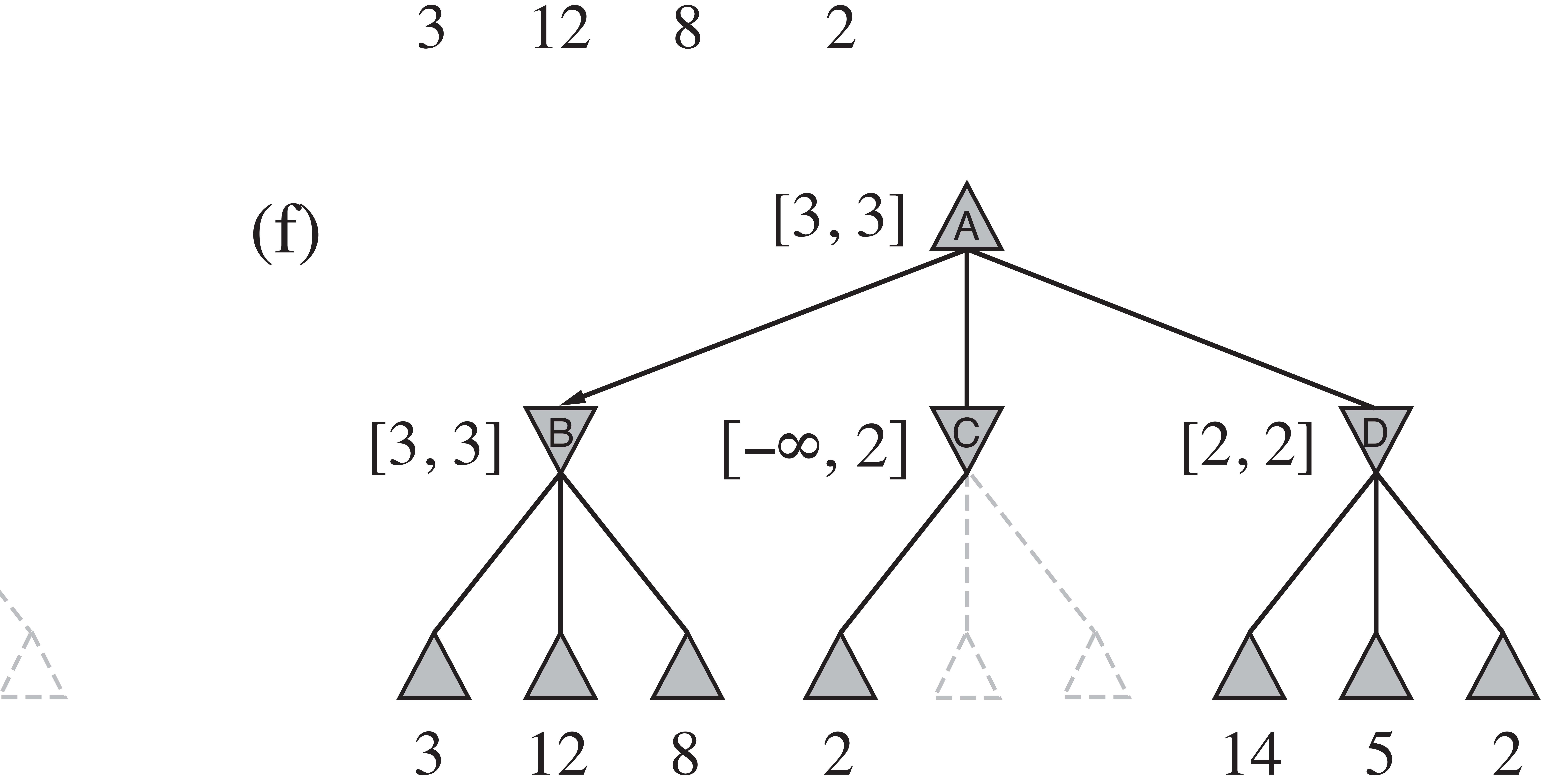


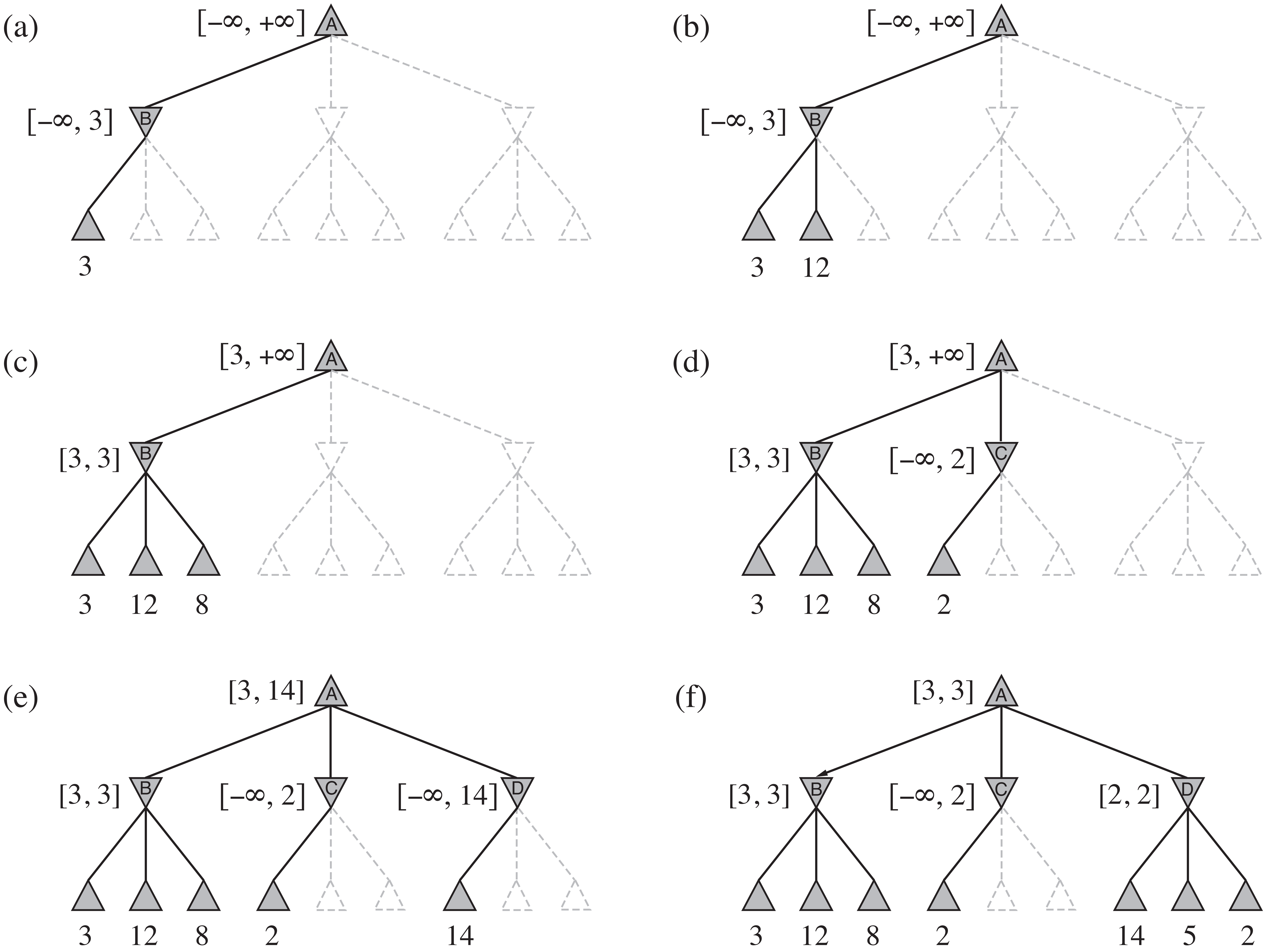
(f)

3 12 8

(e)




3 12 8 2
(f)
[3, 3]
A
[3, 3]
B
[−∞, 2]
C
[2, 2]
D
3 12 8 2 14 5 2


(a)
[−∞, +∞]
A
[−∞, 3]
B
3
(b)
[−∞, +∞]
A
[−∞, 3]
B
3
12
(c)
[3, +∞]
A
[3, 3]
B
3
12
8
(d)
[3, +∞]
A
[3, 3]
B
[−∞, 2]
C
3
12
8
2
(e)
[3, 14]
A
[3, 3]
B
[−∞, 2]
C
[−∞, 14]
D
3
12
8
2
14
(f)
[3, 3]
A
[3, 3]
B
[−∞, 2]
C
[2, 2]
D
3
12
8
2
14
5
2

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), α, β))
    if $v \geq \beta$ **then return** v
    $\alpha \leftarrow$ MAX(α, 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) , α, β))
    if $v \leq \alpha$ **then return** v
    $\beta \leftarrow$ MIN(β, v)
  return v

Notes

- Transposition table: cache previously-seen states
- Maximum-depth heuristics

Reading

- Chapter 5 up to 5.3