Artificial Intelligence

K. N Toosi University of Technology

Course Instructor:

Dr. Omid Azarkasb

Teaching Assistants:

Atena Najaf Abadi Farahani

Saeed Mahmoudian

Games

What is game?

Definition:
A Multi agent environment

Multi agent environments :
1- Cooperative
2- Competitive (Adversarial Search)

Search vs Games

Search — no adversary

- Solution is (heuristic) method for finding goal
- Heuristics and CSP techniques can find optimal solution
- Evaluation function: estimate of cost from start to goal through given node

Examples: path planning, scheduling activities

Games — adversary

- Solution is strategy (strategy specifies move for every possible opponent reply).
- Time limits force an approximate solution
- Evaluation function: evaluate “goodness” of game position

Examples: Chess – Checkers – Othello - Backgammon

Types of Games

	Deterministic	Stochastic
Observable	Chess, Checkers, Go, Othello	Backgammon, Monopoly
Partially Observable		Bridge, Poker, Scrable nuclear war

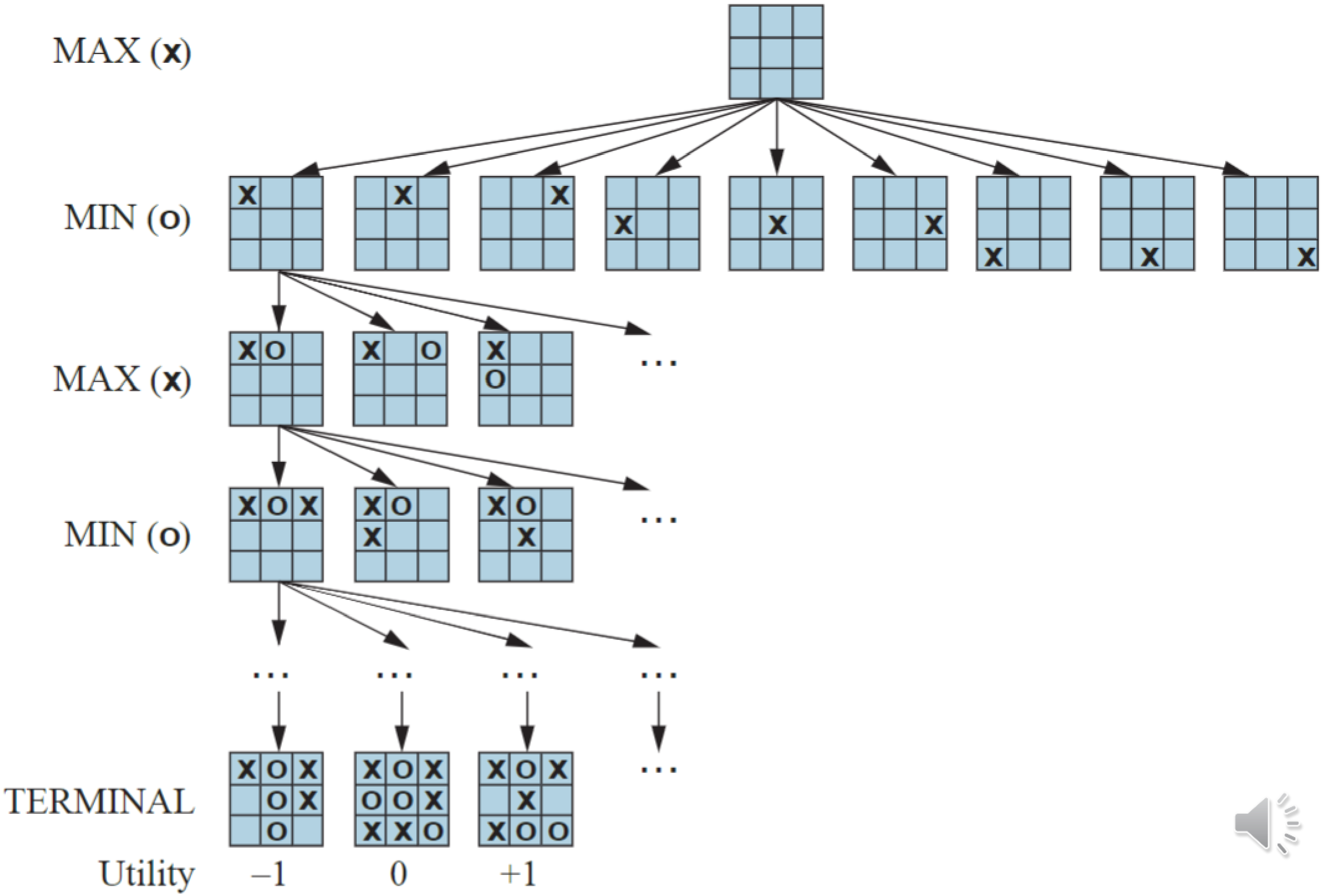
Game properties

Players: 2 Players named MIN and MAX
Game Progress: MAX starts the game. The Game Continues till the end.
Decision Making: MAX uses Tree Search to select the next move

Games as Search:

Initial State: The very first state of the game
Successor Function: list of (move,state) pairs specifying legal moves.
Terminal Test: is the game finished?
Utility Function: Gives numerical value to the terminal states. e.g win(+1), loose(-1) and draw(0)

TIC-TAC-TOE Example

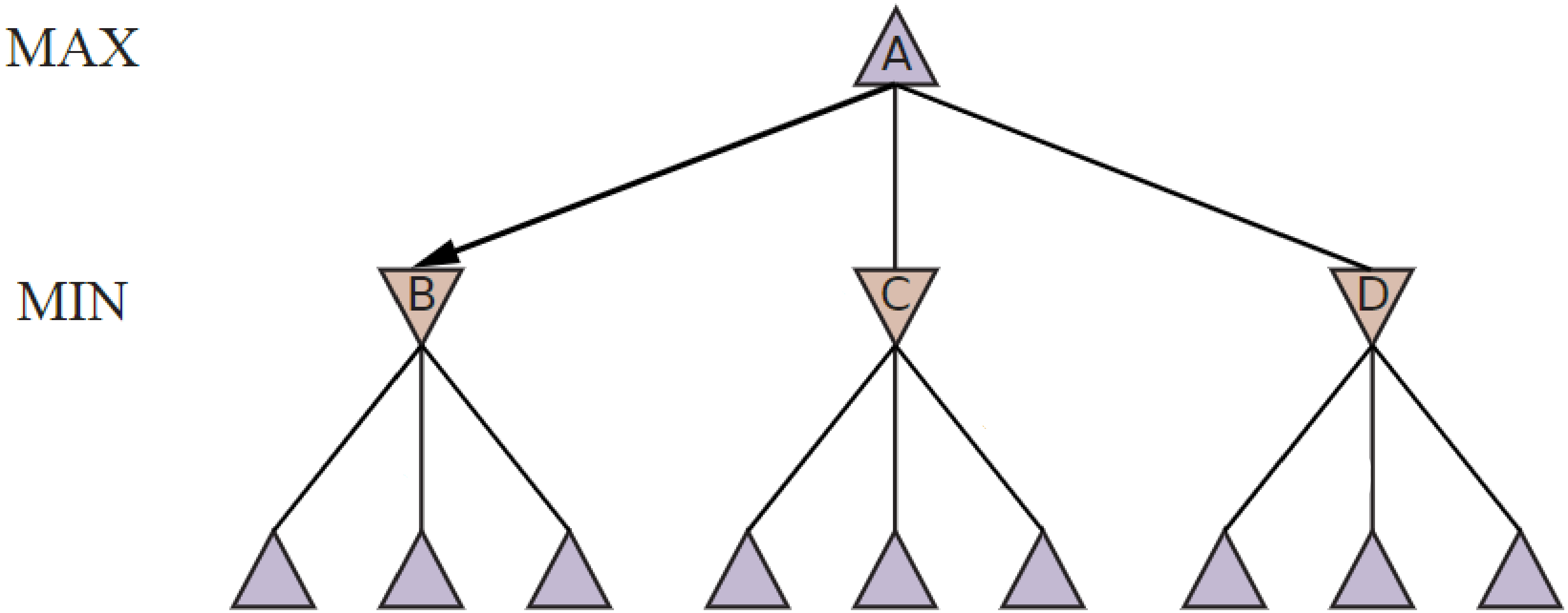


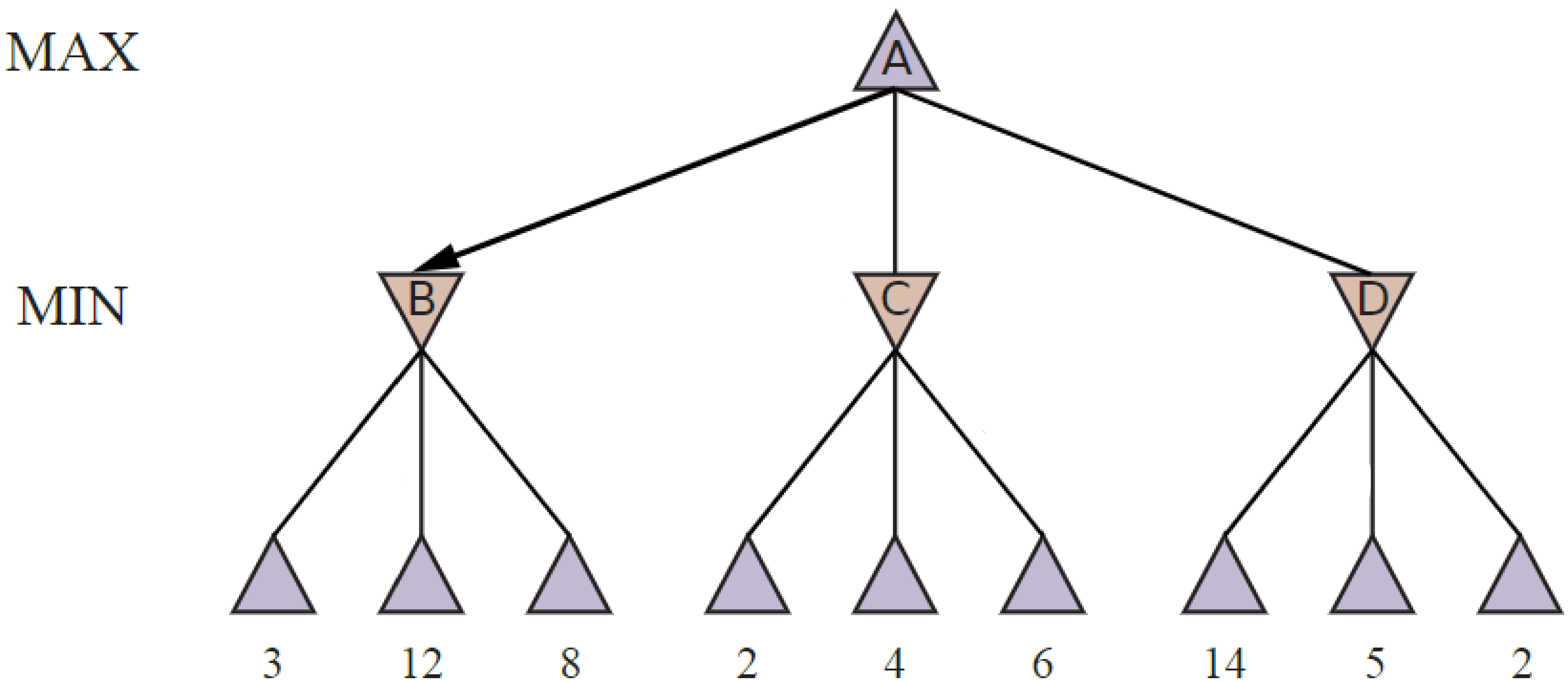
MINIMAX Algorithm

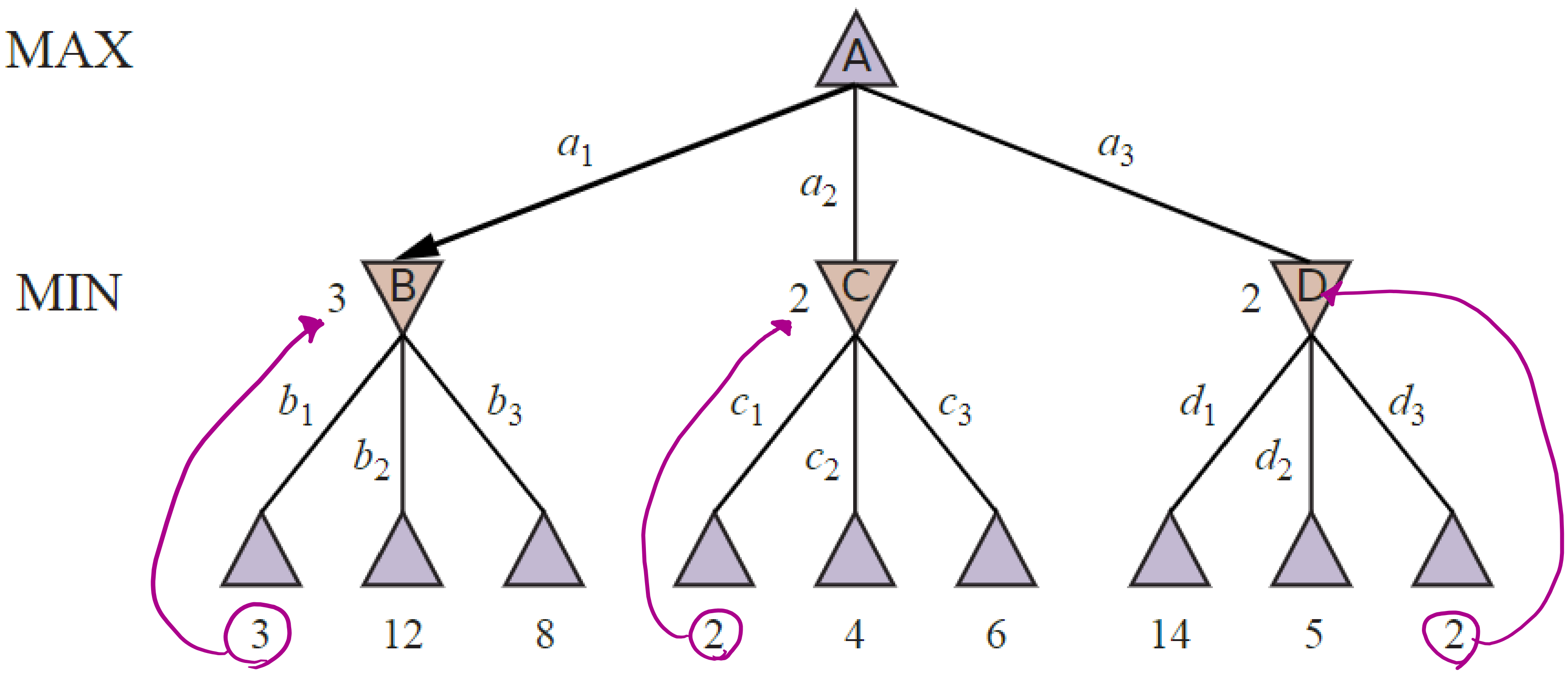
Purpose: Finding the best strategy for MAX (player)
Default: Both players play as (well/smart/efficient) as possible.

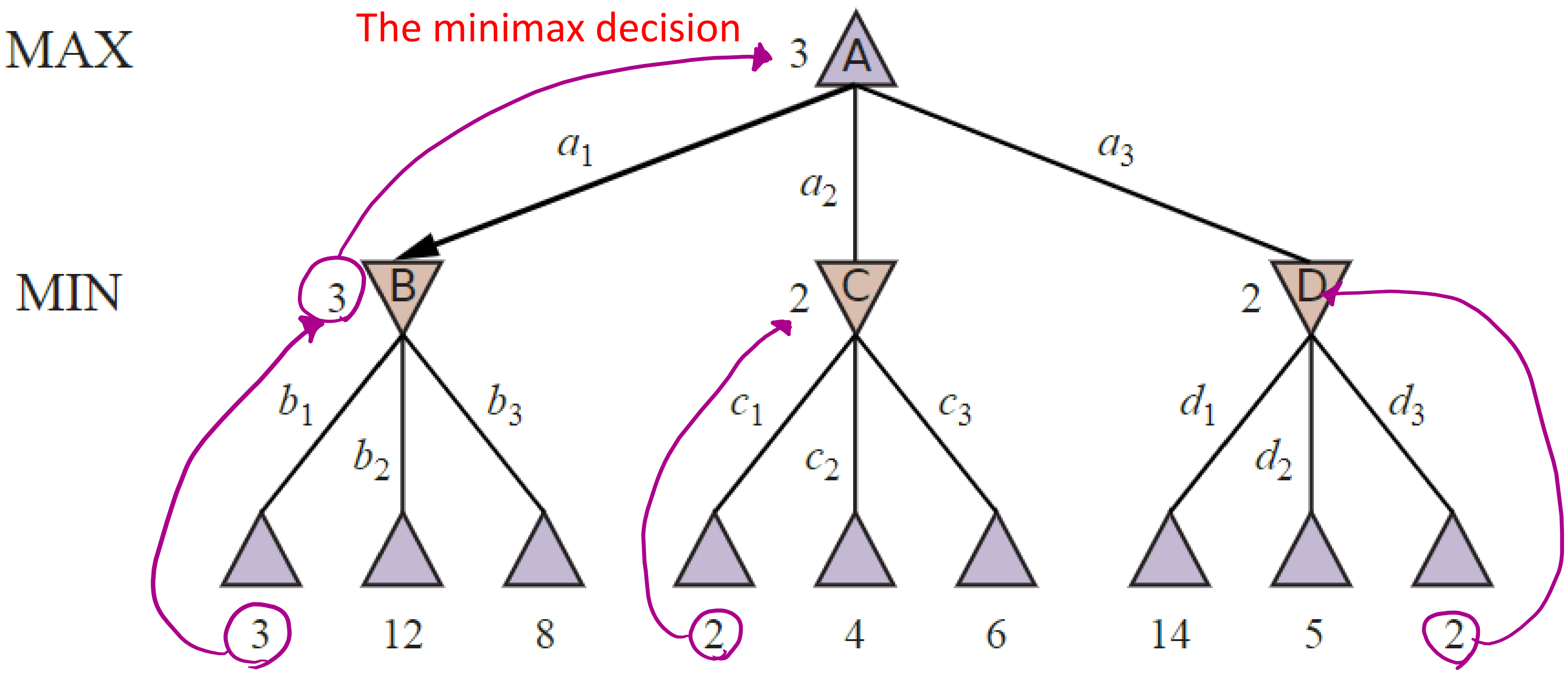
$$\text{MINIMAX}(s) = \begin{cases} \text{UTILITY}(s, \text{MAX}) & \text{if IS-TERMINAL}(s) \\ \max_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if TO-MOVE}(s) = \text{MAX} \\ \min_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if TO-MOVE}(s) = \text{MIN} \end{cases}$$

```
function MINIMAX-SEARCH(game, state) returns an action
  player ← game.TO-MOVE(state)
  value, move ← MAX-VALUE(game, state)
  return move

function MAX-VALUE(game, state) returns a (utility, move) pair
  if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
  v ← −∞
  for each a in game.ACTIONS(state) do
    v2, a2 ← MIN-VALUE(game, game.RESULT(state, a))
    if v2 > v then
      v, move ← v2, a
  return v, move

function MIN-VALUE(game, state) returns a (utility, move) pair
  if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
  v ← +∞
  for each a in game.ACTIONS(state) do
    v2, a2 ← MAX-VALUE(game, game.RESULT(state, a))
    if v2 < v then
      v, move ← v2, a
  return v, move
```


MAX
MIN
A
B
C
D


MAX
MIN
A
B
C
D
3
12
8
2
4
6
14
5
2


MAX
MIN
A
B
C
D
a1
a2
a3
3
2
2
b1
b2
b3
c1
c2
c3
d1
d2
d3
3
12
8
2
4
6
14
5
2



Minimax Algorithm maximizes the worst-case outcome for MAX.

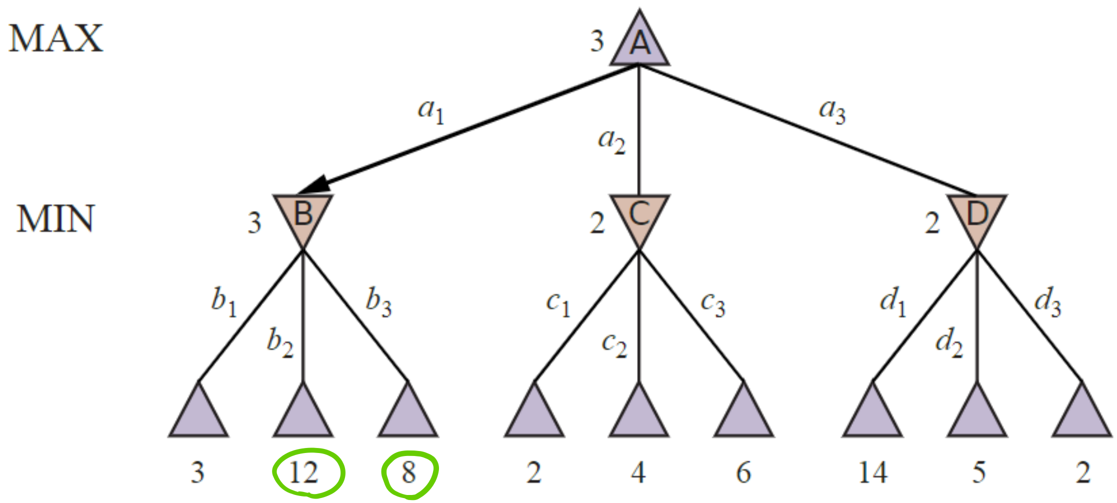
Question

Q:

What if the MIN player doesn't play as (well/smart/efficient) as possible?

A:

The result would be better then!



Games with more than two players

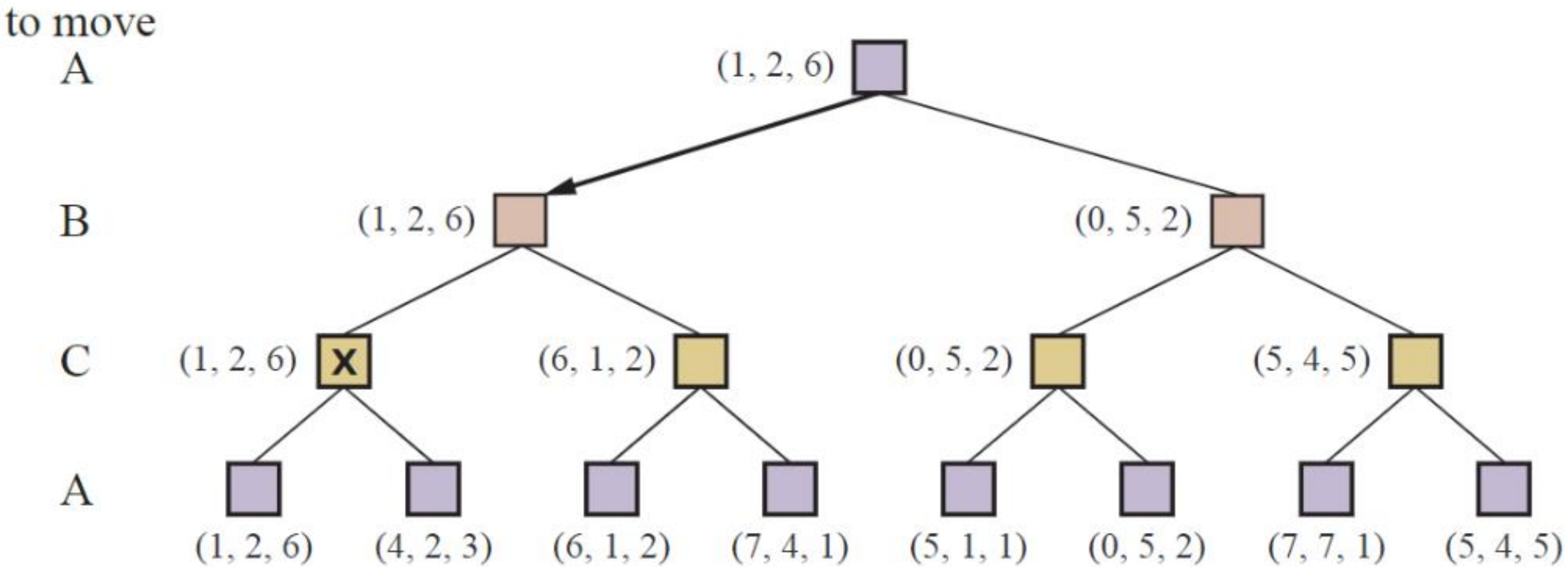


Figure 5.4 The first three ply of a game tree with three players (*A*, *B*, *C*). Each node is labeled with values from the viewpoint of each player. The best move is marked at the root.

MINIMAX

Algorithm	MINIMAX
Complete	Yes
Time	$O(b^m)$
Space	$O(bm)$
Optimal	Yes

Alpha-Beta Pruning

Problem: The number of states is exponential in compared to the number of moves.
Solution: Don't check all nodes.

Alpha = Maximum value in the node MAX
Beta = Minimum Value in the node MIN

Rules:

- **First Rule:**

In the MIN node P, prune if Alpha(Parent(P)) > Beta(P)

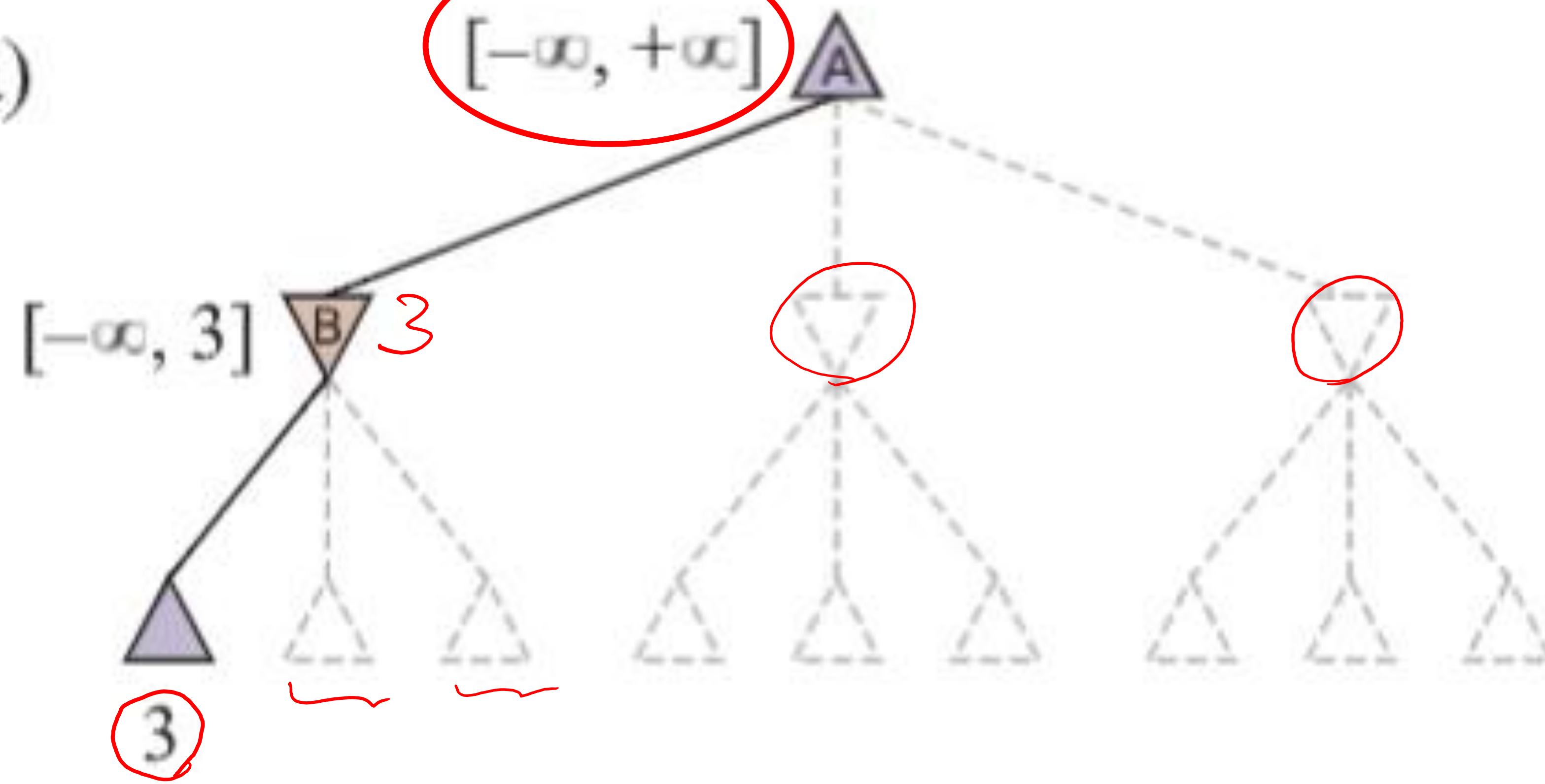
- **Second Rule:**

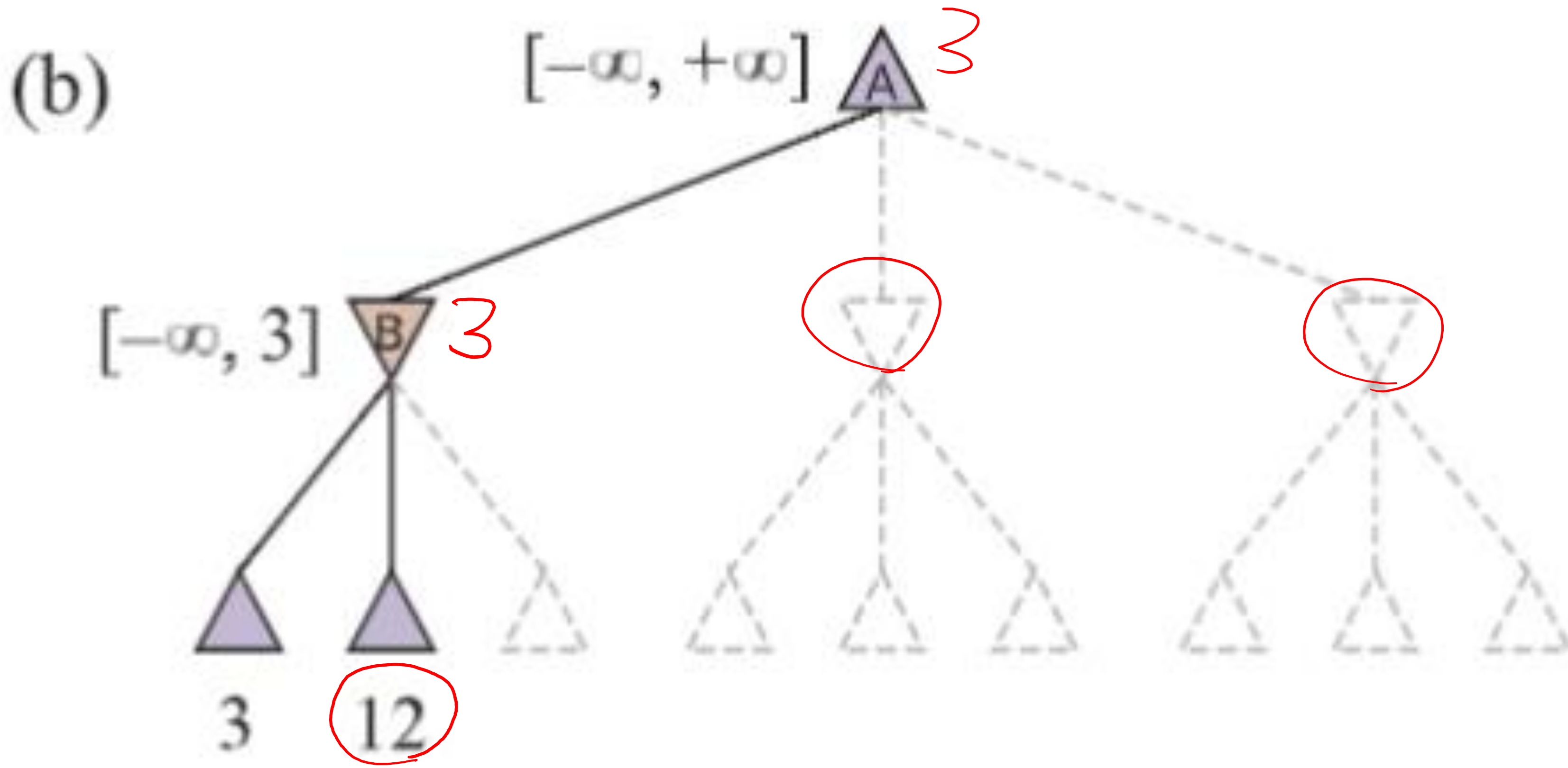
In the MAX node Q, prune if Alpha(Q) > Beta(Parent(Q))

```
function ALPHA-BETA-SEARCH(game, state) returns an action
  player ← game.TO-MOVE(state)
  value, move ← MAX-VALUE(game, state, −∞, +∞)
  return move

function MAX-VALUE(game, state, α, β) returns a (utility, move) pair
  if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
  v ← −∞
  for each a in game.ACTIONS(state) do
    v2, a2 ← MIN-VALUE(game, game.RESULT(state, a), α, β)
    if v2 > v then
      v, move ← v2, a
      α ← MAX(α, v)
    if v ≥ β then return v, move
  return v, move

function MIN-VALUE(game, state, α, β) returns a (utility, move) pair
  if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
  v ← +∞
  for each a in game.ACTIONS(state) do
    v2, a2 ← MAX-VALUE(game, game.RESULT(state, a), α, β)
    if v2 < v then
      v, move ← v2, a
      β ← MIN(β, v)
    if v ≤ α then return v, move
  return v, move
```

Range of possible values

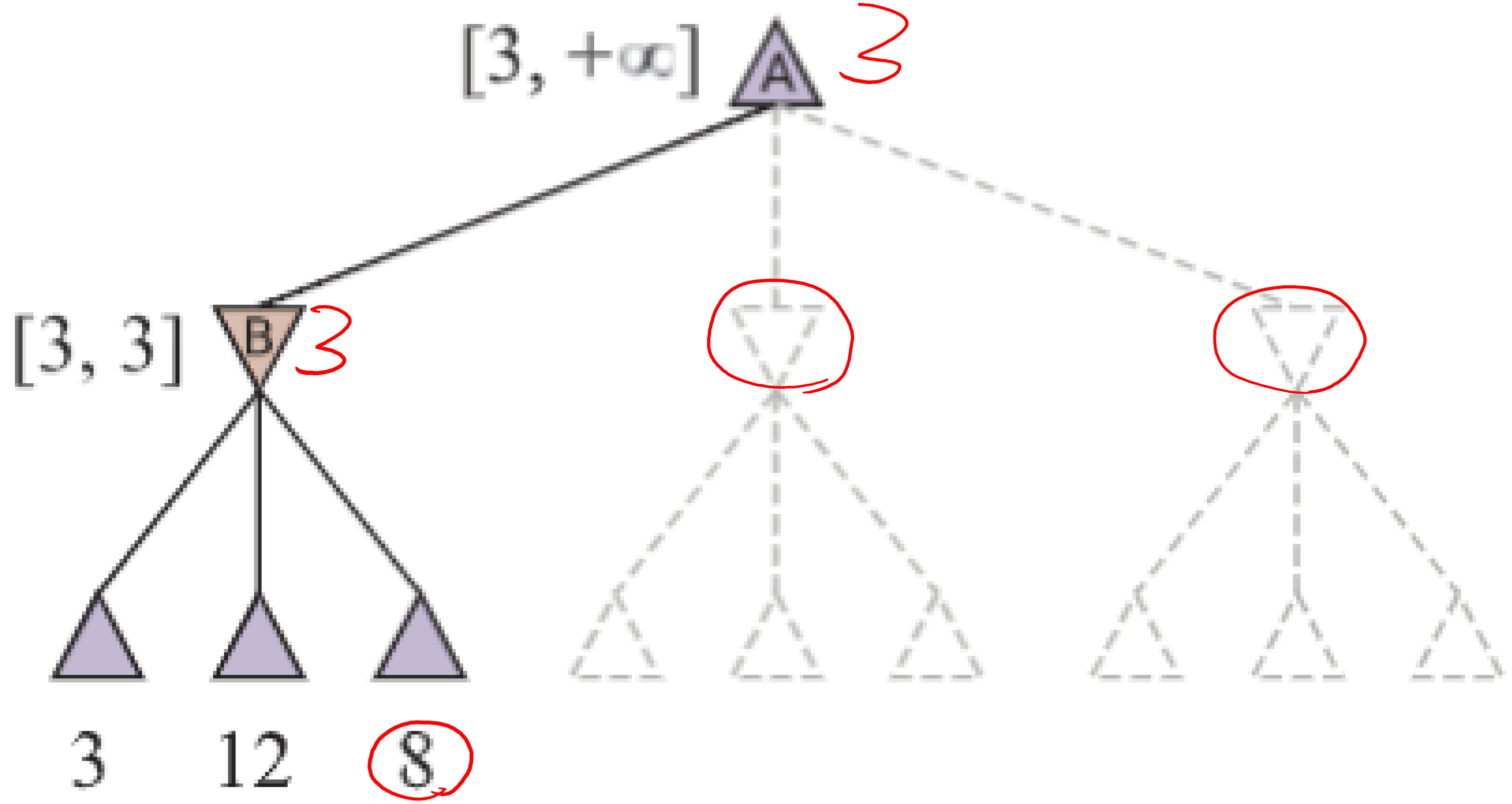
(a)



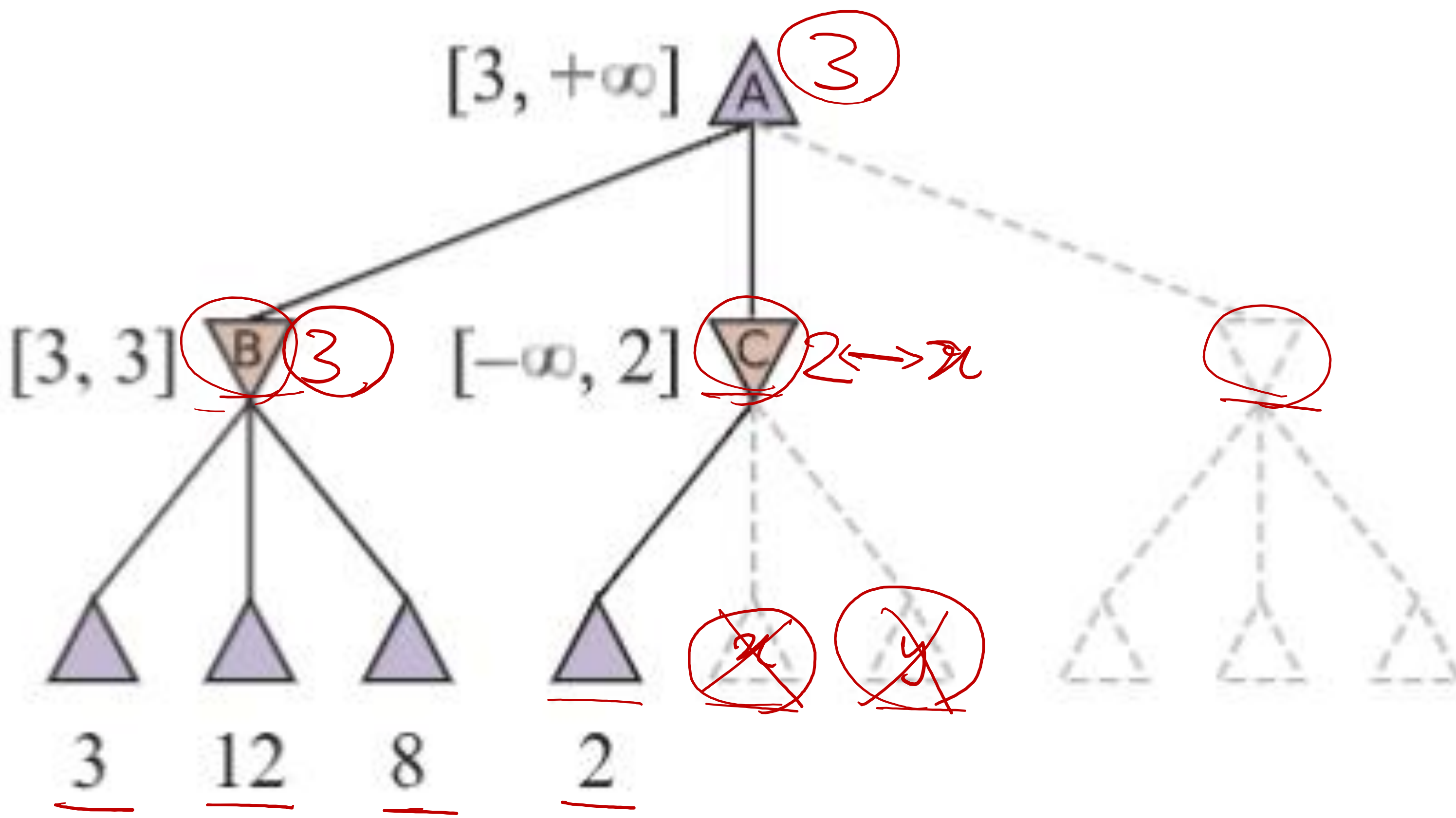


(b)
[−∞, +∞]
A
3
[−∞, 3]
B
3
3
12

(c)



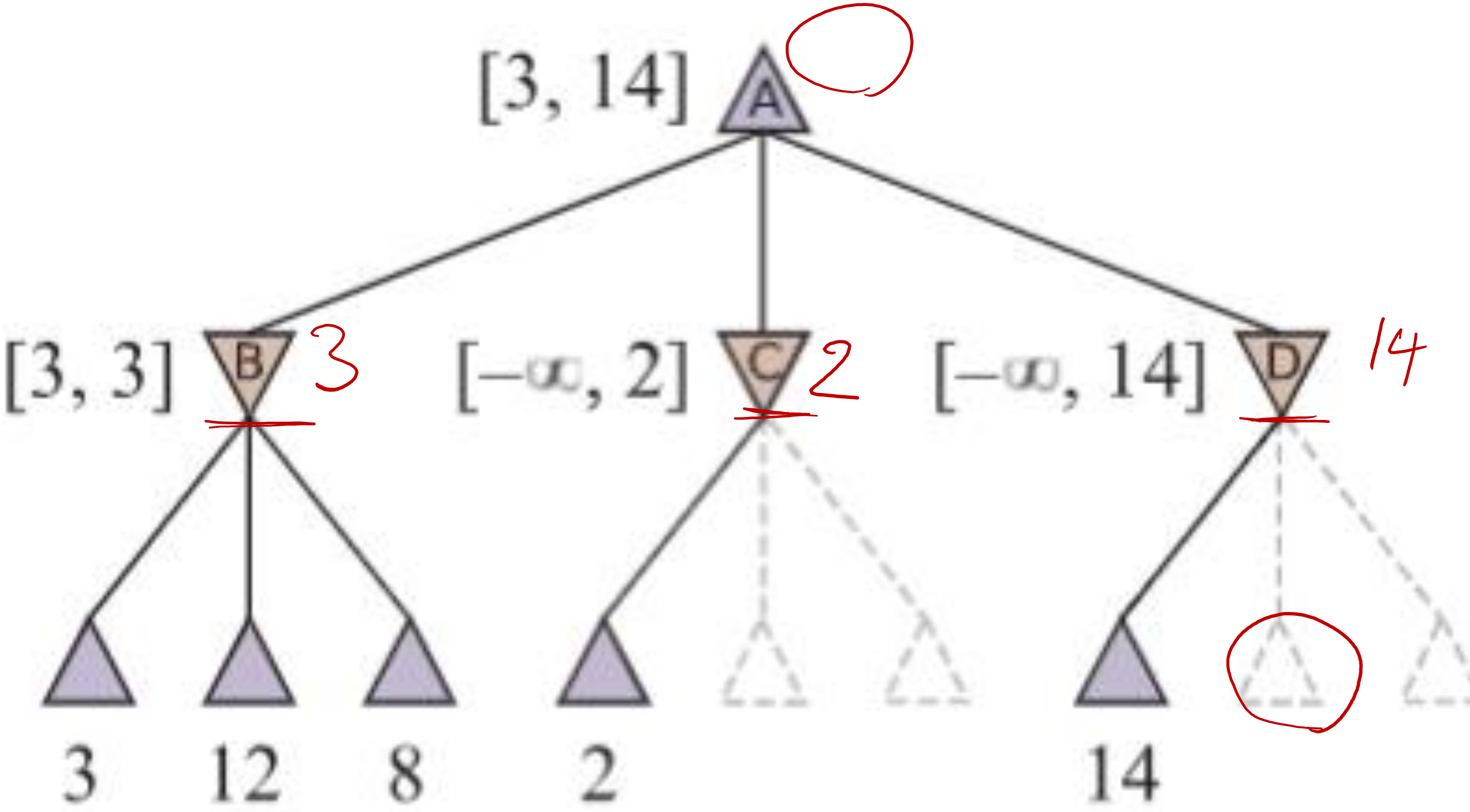
(d)

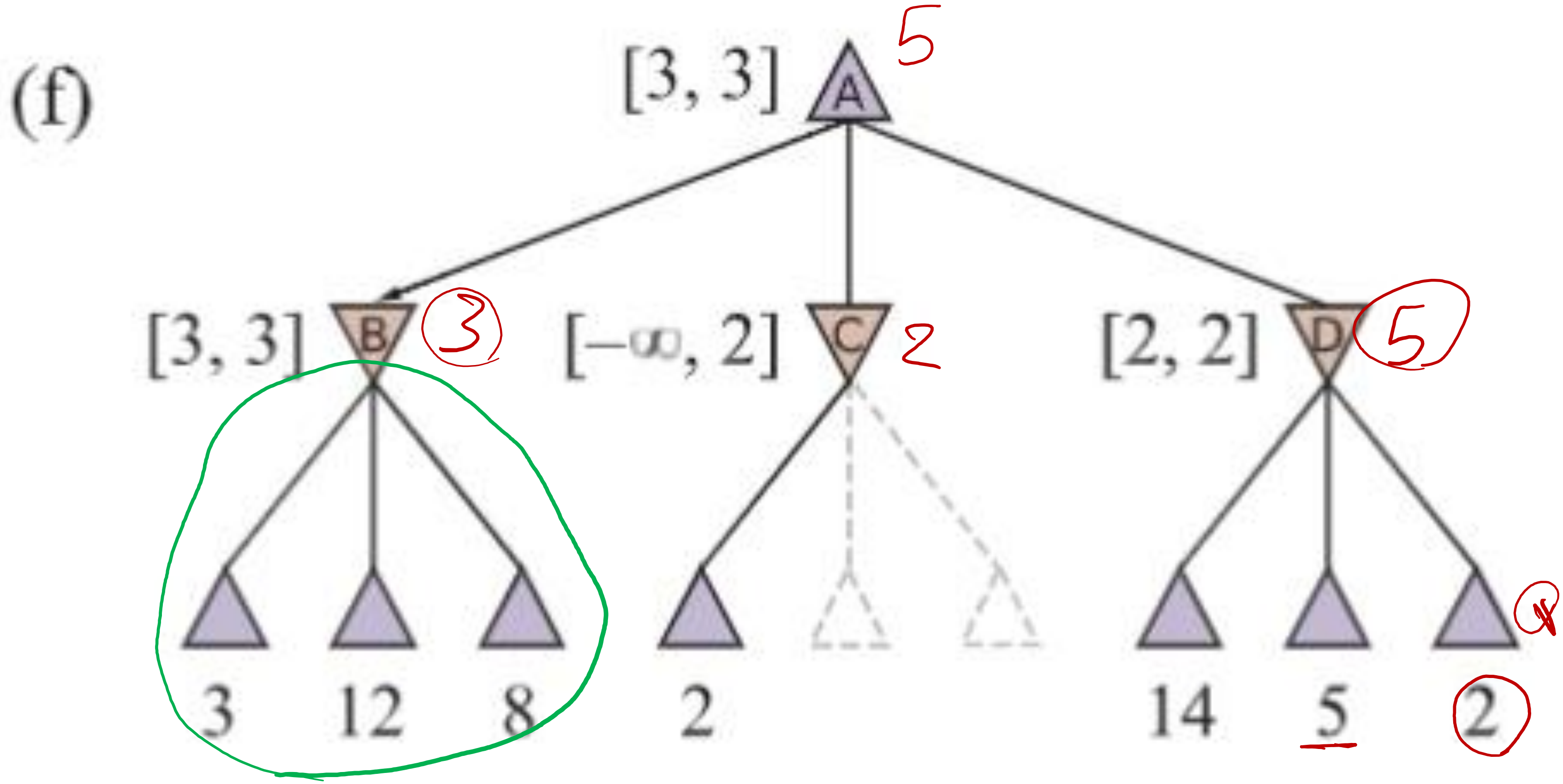


$x, y \geq 2$ (1)

$x, y < 2$ (2)

(e)





4, 3, 2, ...