ADVERSARIAL SEARCH

Abdus Salam Azad

Objectives

Learning how to act when the other agents are acting against us

Games

Games vs. Search

- Search – no adversary
  - Examples: path planning, scheduling activities
- Games – adversary
  - **Unpredictable opponent(s)**
  - Solution is strategy (strategy specifies move for every possible opponent reply).
  - **Time limits force an *approximate* solution**
  - **Inefficiency is intolerable**

Zero Sum Game

- Total pay off to all players is the same for every instance of games
- Chess/Tic-tac-toe:
  - Win – 1
  - Lose – 0
  - Draw – ½

When I Win – You Lose

Non-Zero Sum


My Interests
"I win."
Area of
Overlapping
Interests
"Win/Win."
Your Interests
"You lose."

Assumptions

- **Two agents acting alternately**
- **Utility values for each agent are the opposite of the other**
- Deterministic
- Fully observable
- Can generalize to stochastic games, multiple players, non zero-sum, etc
- In game theory terms:
  - "Deterministic, turn-taking, zero-sum games of perfect information"

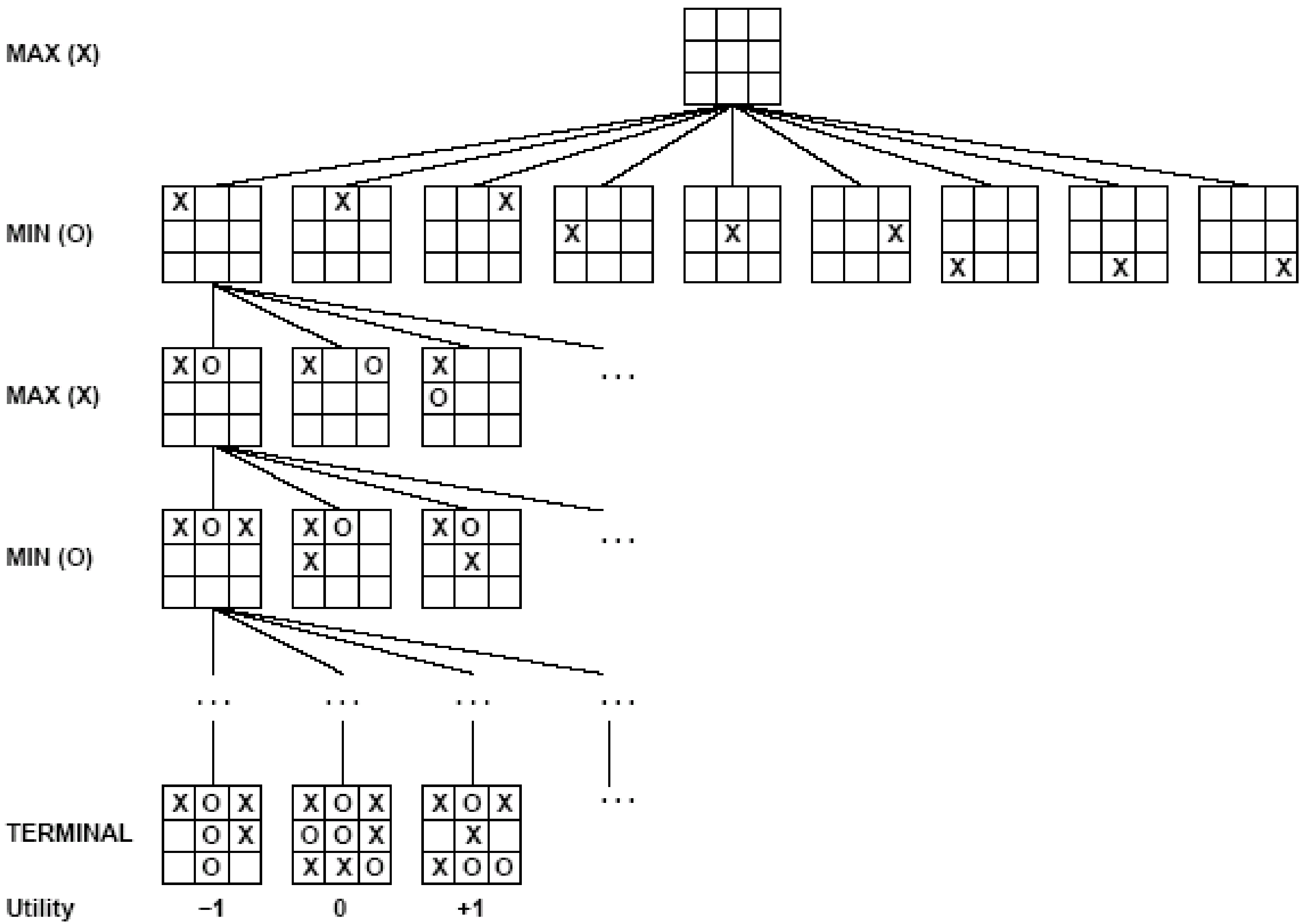
Games as search

- Initial state: e.g. board configuration of chess
- Player: which player to give the current move
- Successor function: list of (move,state) pairs specifying legal moves.
- Terminal test: Is the game finished?
- Utility function: Gives numerical value of terminal states. E.g. win (+1), lose (-1) and draw (0) in tic-tac-toe or chess

Game Setup

- **Two players: MAX and MIN**
- **MAX tries to maximize the utility**
- MAX moves first and they take turns until the game is over
- MAX uses search tree to determine next move.
- We plan as MAX

Partial Game Tree for Tic-Tac-Toe



Size of search trees

- b = branching factor
- d = number of moves by both players
- Search tree is $O(b^d)$

- Chess
  - b ~ 35
  - D ~100
    - search tree is ~ 10^{154} (!!)
    - completely impractical to search this

- Game-playing emphasizes being able to make optimal decisions in a finite amount of time

Optimal Strategy

An optimal strategy leads to outcomes at least as good as any other strategy when the opponent plays optimally

Strategy 1

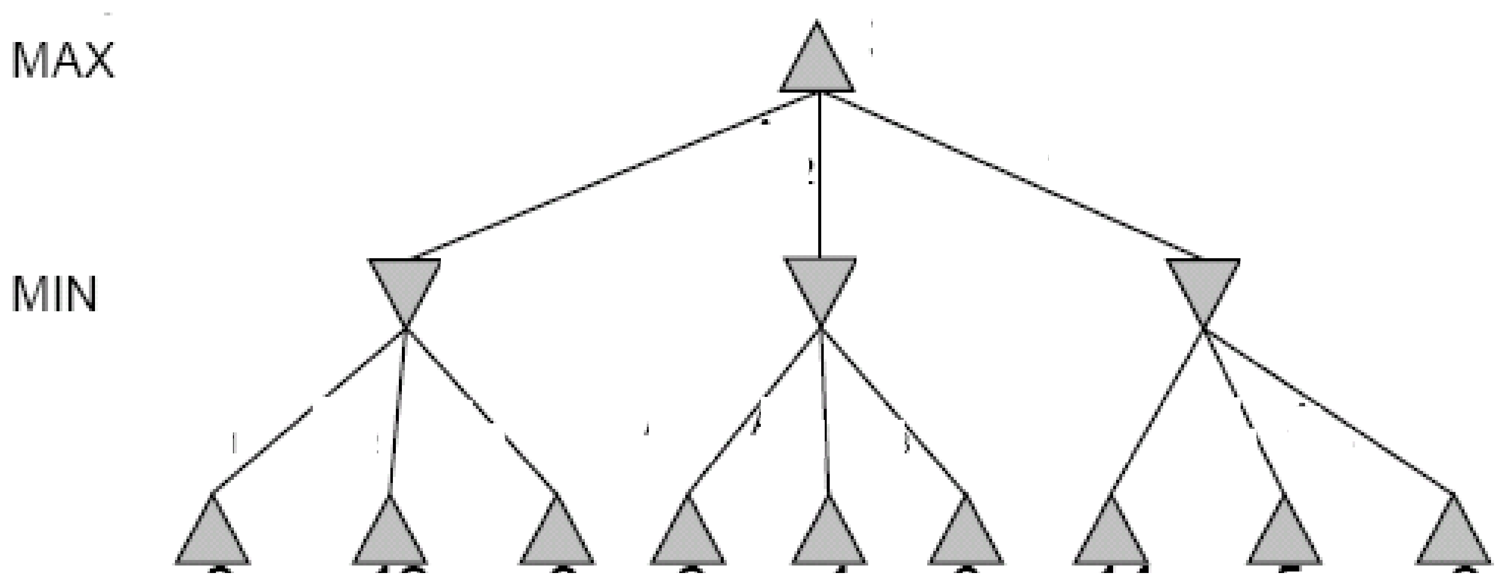
The minimax algorithm

The minimax algorithm

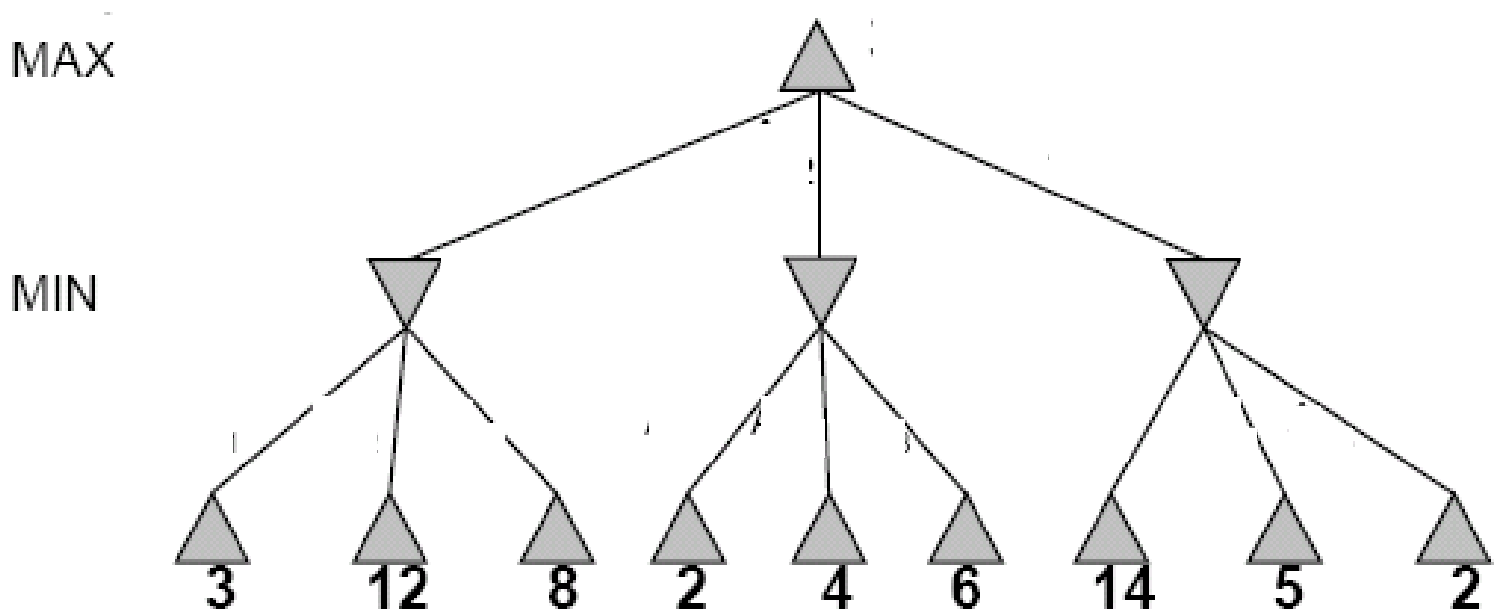
- Find the optimal *strategy* for MAX assuming an optimal MIN opponent

- Assumption: Both players play optimally!

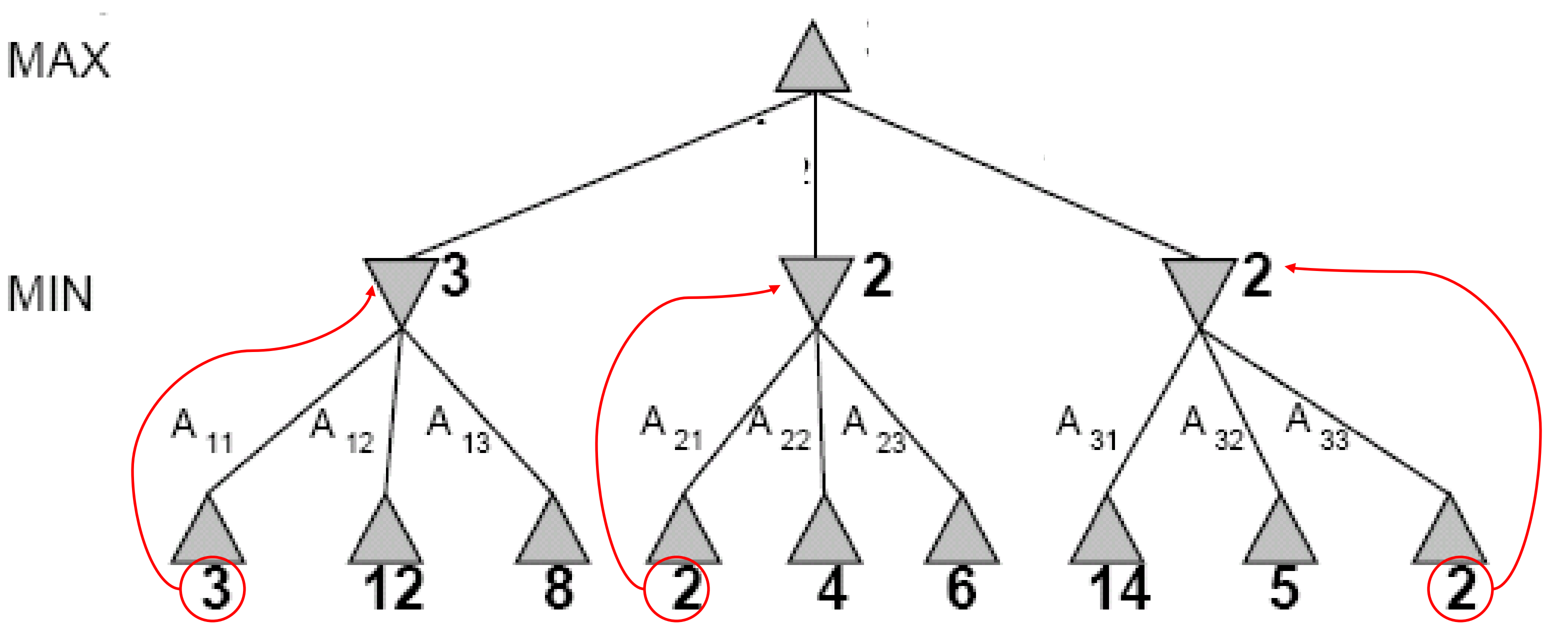
Two-Ply Game Tree



Two-Ply Game Tree

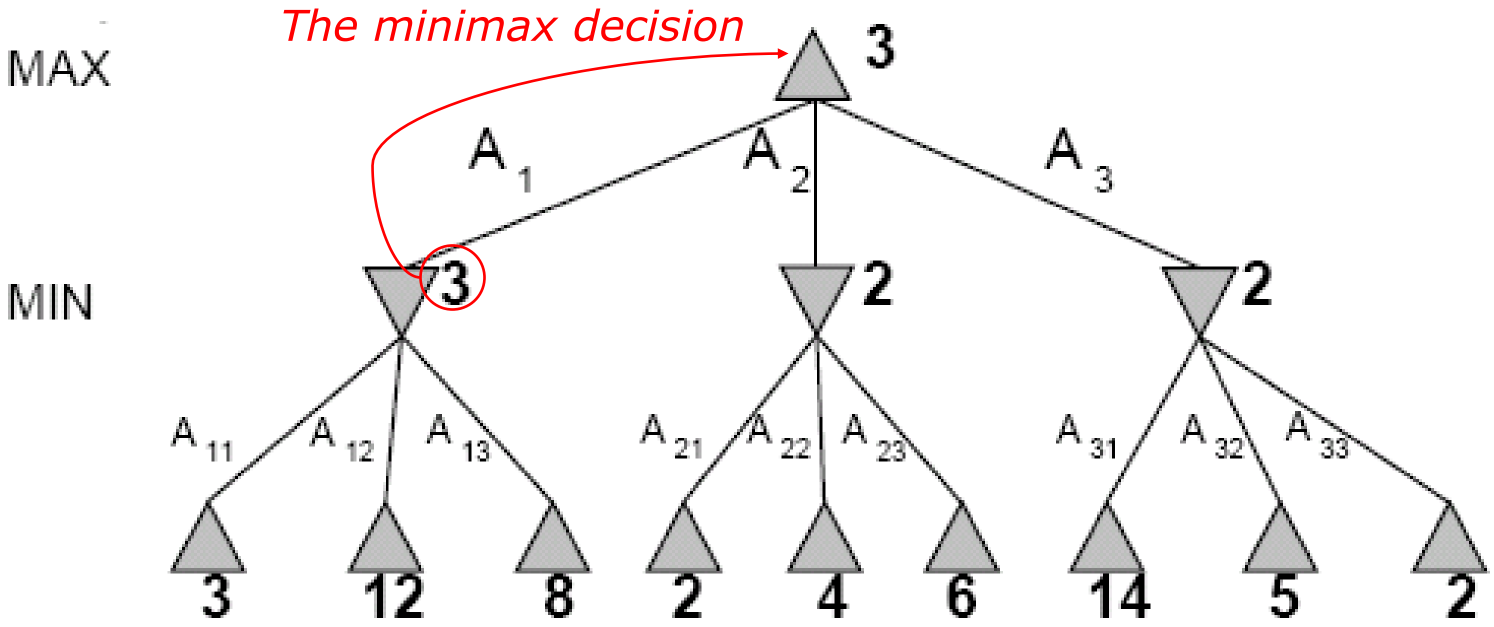


Two-Ply Game Tree



Two-Ply Game Tree

Minimax maximizes the utility for the worst-case outcome for max



The minimax algorithm

- Minimax value is the utility of MAX for being in the corresponding state

MINIMAX-VALUE(*n*)

UTILITY(*n*) **If *n* is a terminal**

$\mathbf{max}_{s \in successors(n)}$ **MINIMAX-VALUE(*s*)**

If *n* is a max node

$\mathbf{min}_{s \in successors(n)}$ **MINIMAX-VALUE(*s*)**

If *n* is a min node

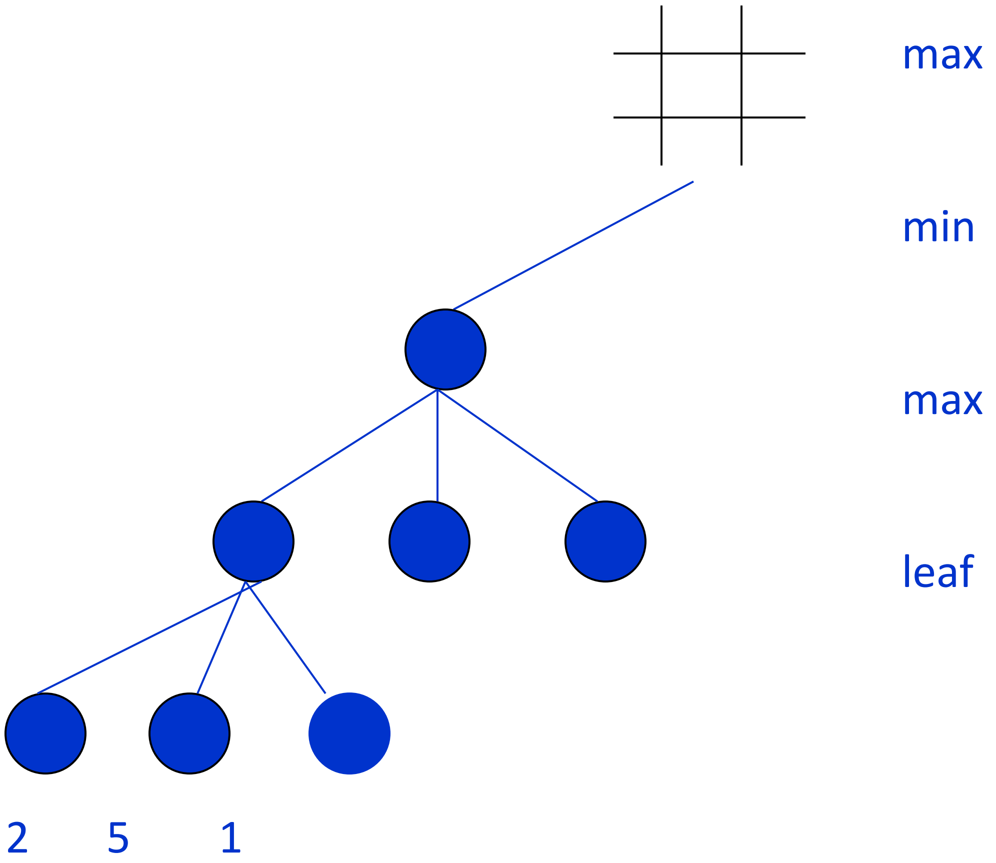
Minimax algorithm

```
function MINIMAX-DECISION(state) returns an action
    v ← MAX-VALUE(state)
    return the action in SUCCESSORS(state) with value v

function MAX-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← −∞
    for a, s in SUCCESSORS(state) do
        v ← MAX(v, MIN-VALUE(s))
    return v

function MIN-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← ∞
    for a, s in SUCCESSORS(state) do
        v ← MIN(v, MAX-VALUE(s))
    return v
```

Minimax is done depth-first



Properties of Minimax

- Complete? Yes (if tree is finite)
- Optimal? Yes (against an optimal opponent)
- Time complexity? $O(b^m)$
- Space complexity? $O(bm)$ (depth-first exploration)

- For chess, $b \approx 35$, $m \approx 100$ for "reasonable" games
  → exact solution completely infeasible

Need to speed it up.

Strategy 2

Alpha Beta Pruning

Alpha-Beta Procedure

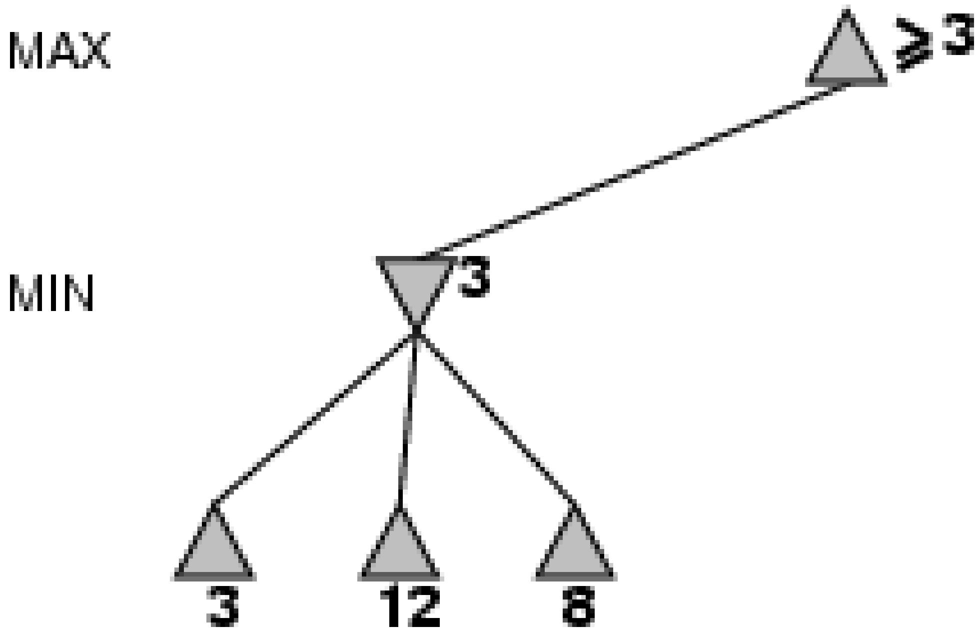
- The alpha-beta procedure can speed up a depth-first minimax search.
- Alpha: a lower bound on the value that a max node may ultimately be assigned

  $v \geq \alpha$

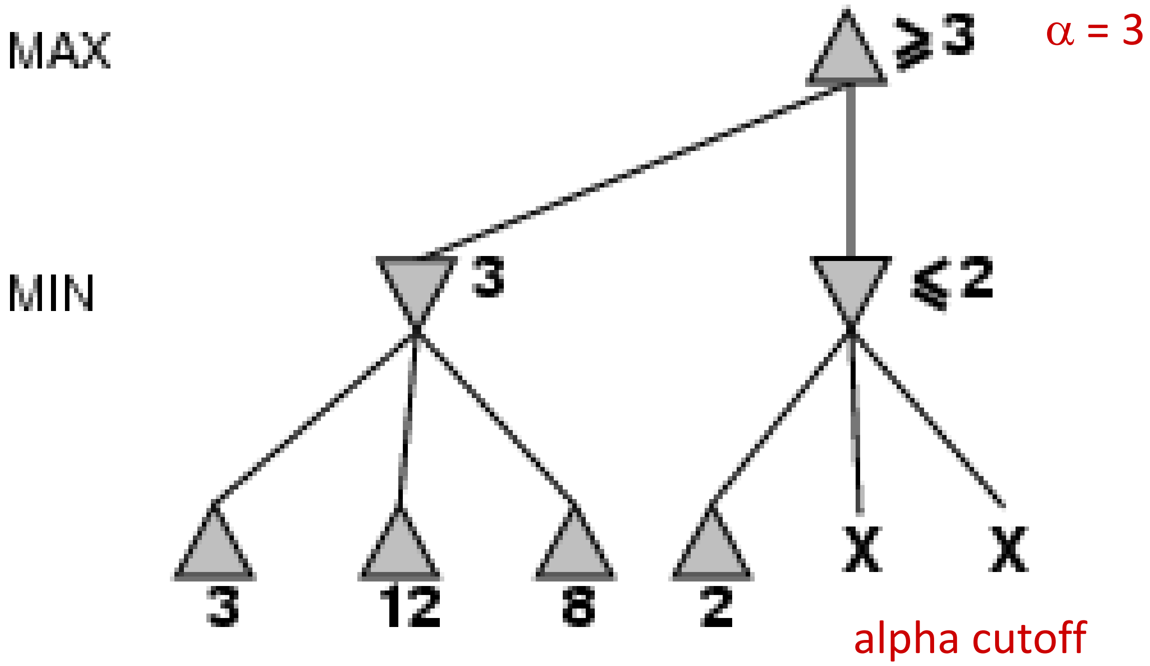
- Beta: an upper bound on the value that a minimizing node may ultimately be assigned

  $v \leq \beta$

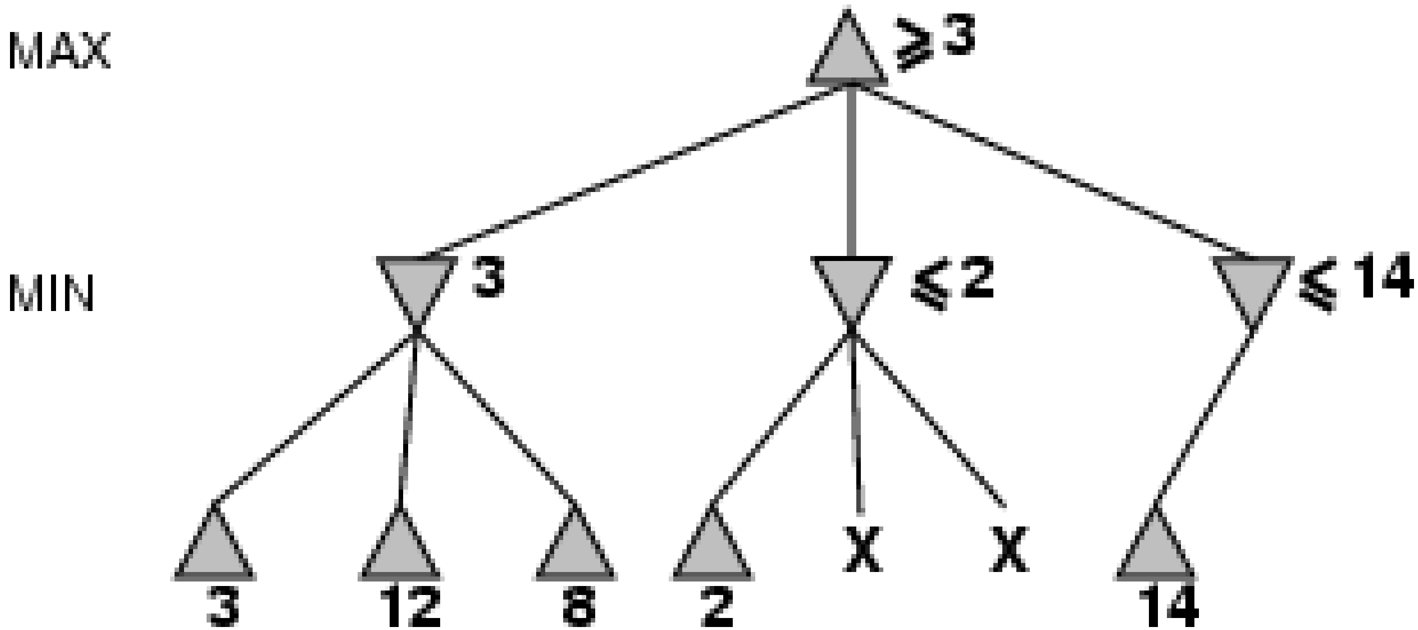
α-β pruning example



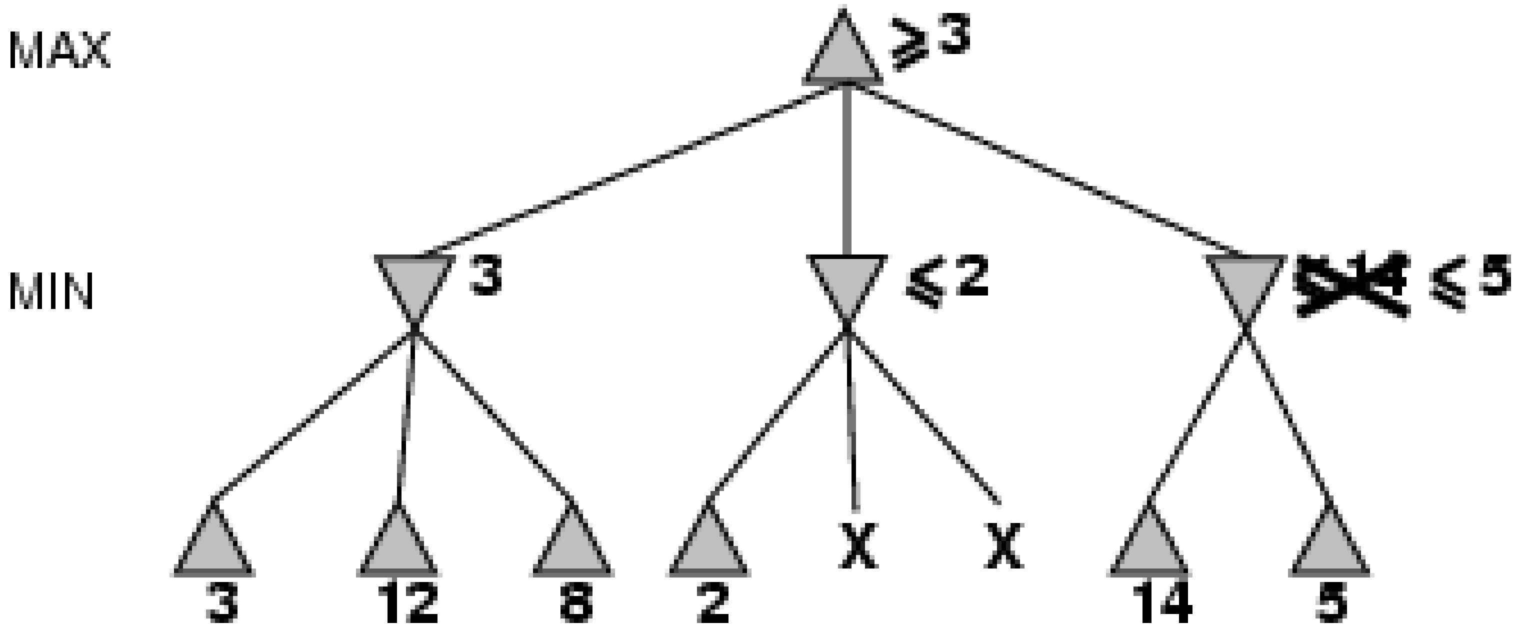
α-β pruning example



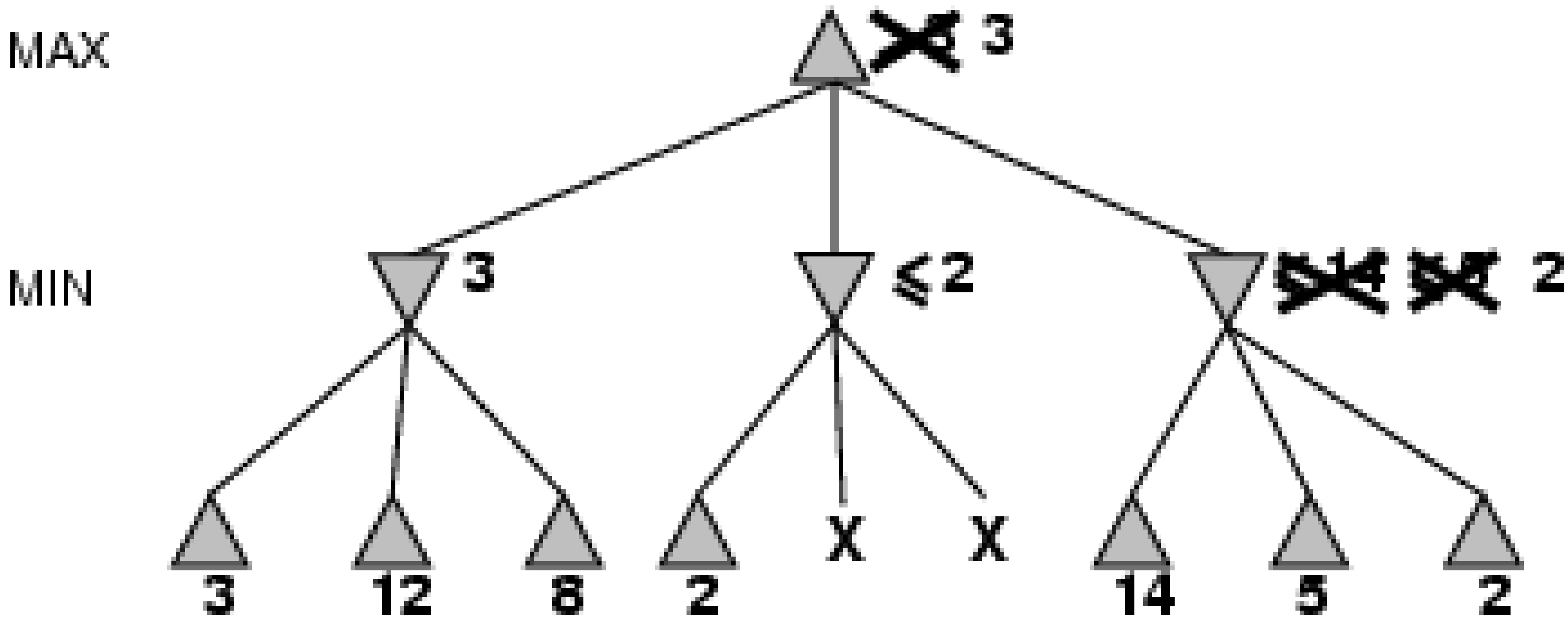
α-β pruning example



α-β pruning example



α-β pruning example



Properties of α-β

- Pruning does not affect final result. This means that it gets the exact same result as does full minimax.

- Good move ordering improves effectiveness of pruning

- With "perfect ordering," time complexity = $O(b^{m/2})$
  → doubles depth of search

The α-β algorithm

function ALPHA-BETA-SEARCH(*state*) **returns** *an action*
  inputs: *state*, current state in game

  $v \leftarrow$ MAX-VALUE($state$, $-\infty$, $+\infty$)
  return the *action* in SUCCESSORS(*state*) with value v

---

function MAX-VALUE($state, \alpha, \beta$) **returns** *a utility value*
  inputs: *state*, current state in game
    α, the value of the best alternative for MAX along the path to *state*
    β, the value of the best alternative for MIN along the path to *state*

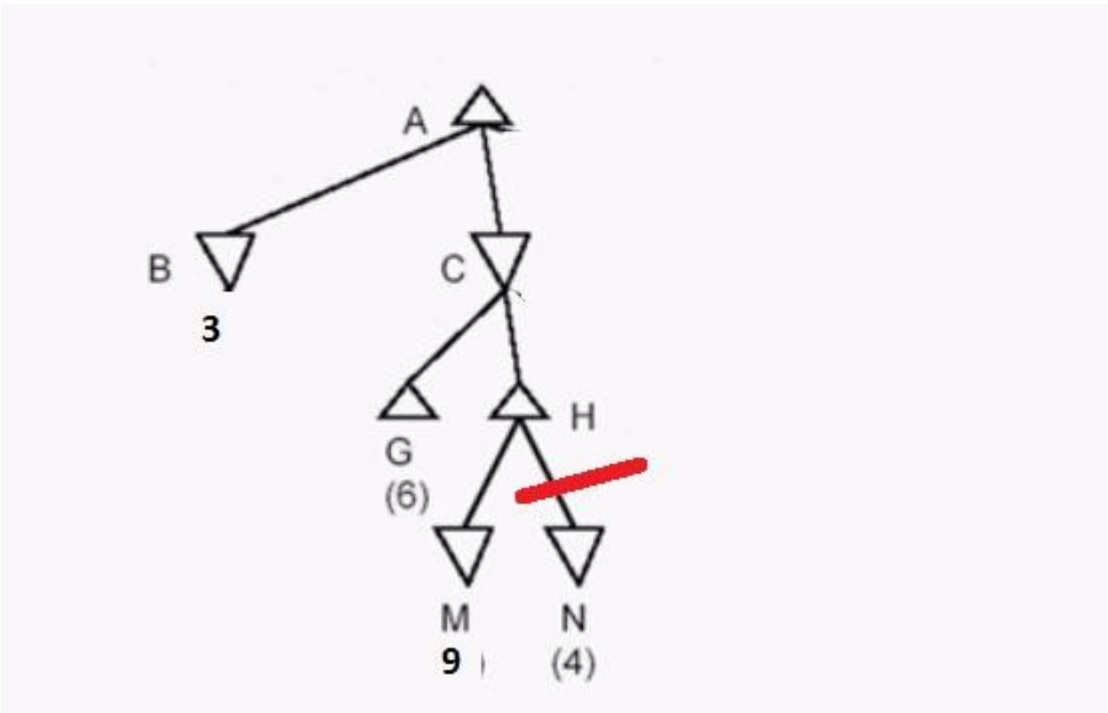
  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow -\infty$
  for a, s in SUCCESSORS(*state*) **do**
    $v \leftarrow$ MAX(v, MIN-VALUE(s, α, β))
    if $v \geq \beta$ **then return** v
    $\alpha \leftarrow$ MAX(α, v)
  return v

The α-β algorithm

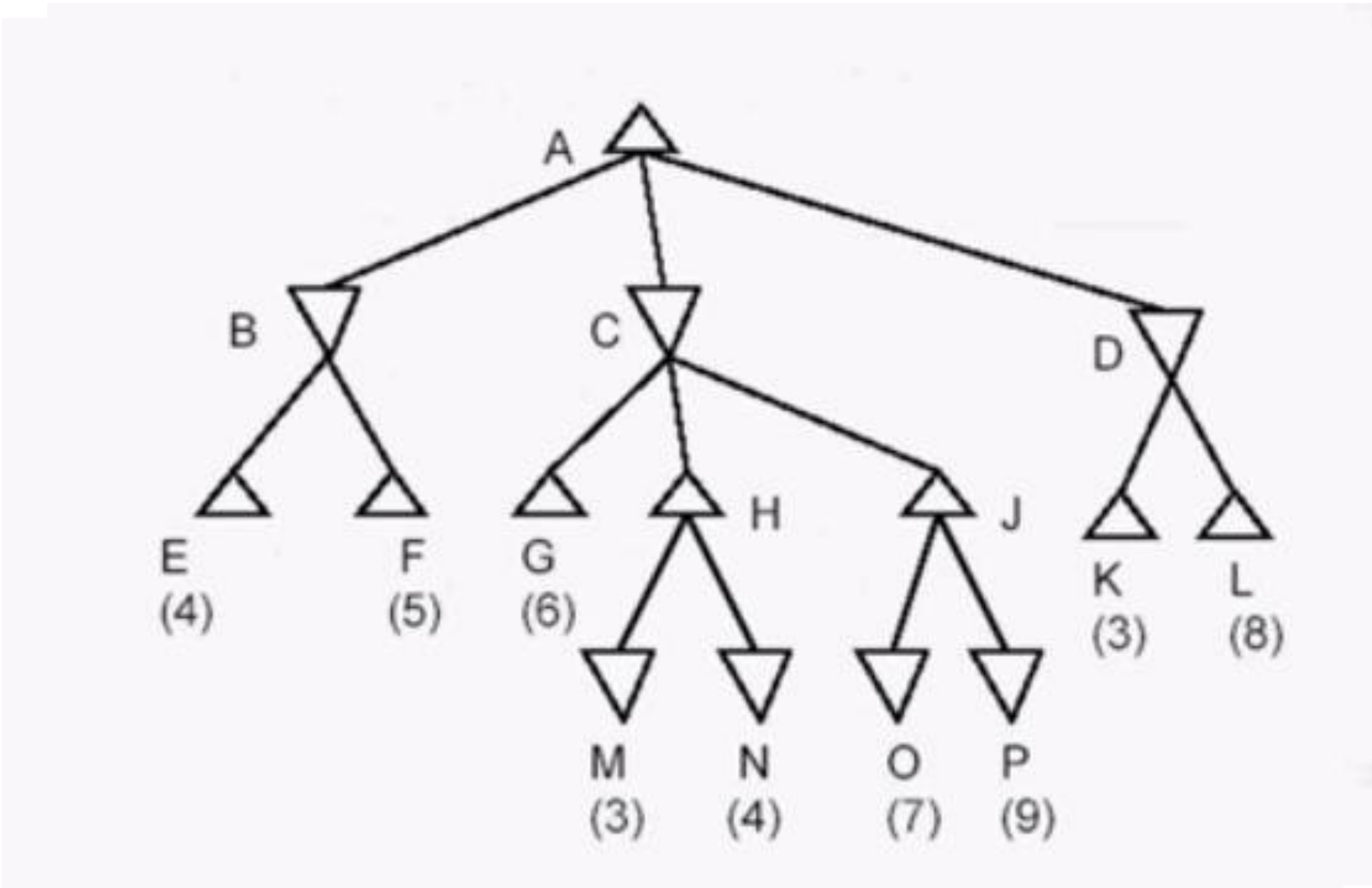
function MIN-VALUE($state, \alpha, \beta$) **returns** *a utility value*
  inputs: *state*, current state in game
    α, the value of the best alternative for MAX along the path to *state*
    β, the value of the best alternative for MIN along the path to *state*

  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow +\infty$
  for a, s in SUCCESSORS(*state*) **do**
    $v \leftarrow$ MIN(v, MAX-VALUE(s, α, β))
    if $v \leq \alpha$ **then return** v
    $\beta \leftarrow$ MIN(β, v)
  return v

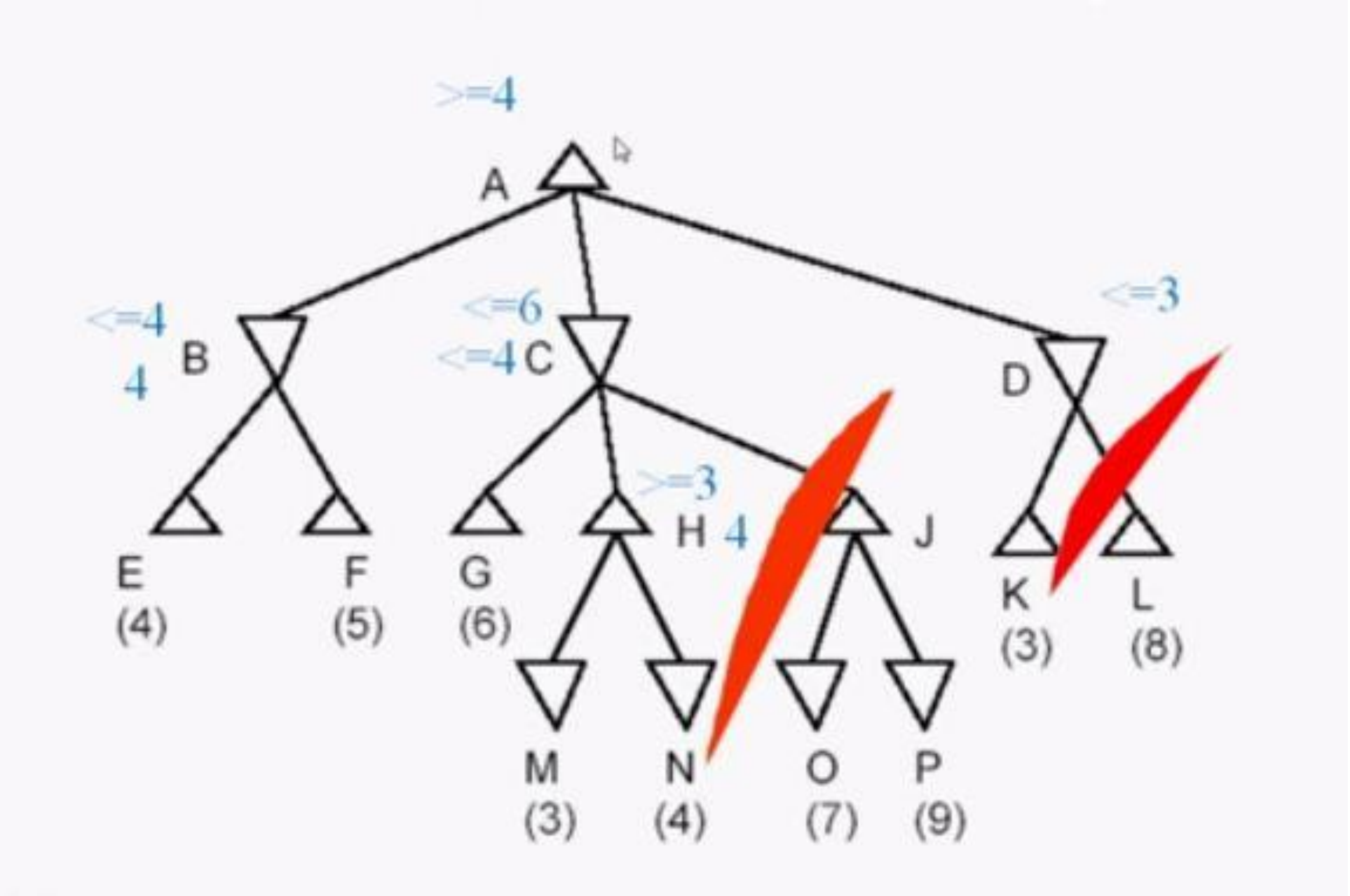
Example



Example



Example



Resource

- Chapter 5
  - 5.1, 5.2, 5.3