Introduction to Artificial Intelligence

Chapter 2: Solving Problems by Searching (6)

Adversarial Search

Nguyễn Hải Minh, Ph.D

nhminh@fit.hcmus.edu.vn

Outline

1. Games
2. Optimal Decisions in Games
3. α-β Pruning
4. Imperfect, Real-time Decisions

Games vs. Search Problems

- Unpredictable opponent
  - → specifying a move for every possible opponent reply
- Competitive environments:
  - → the agents' goals are in conflict
- Time limits
  - → unlikely to find goal, must approximate
- Example of complexity:
  - Chess: b=35 , d = 100 → Tree Size: $\sim 10^{154}$
  - Go: b=1000 (!)

Types of Games

	Deterministic	Chance
Perfect information	Chess, Checkers, Go, Othello	Backgammon Monopoly
Imperfect information		Bridge, poker, scrabble nuclear war

Types of Games

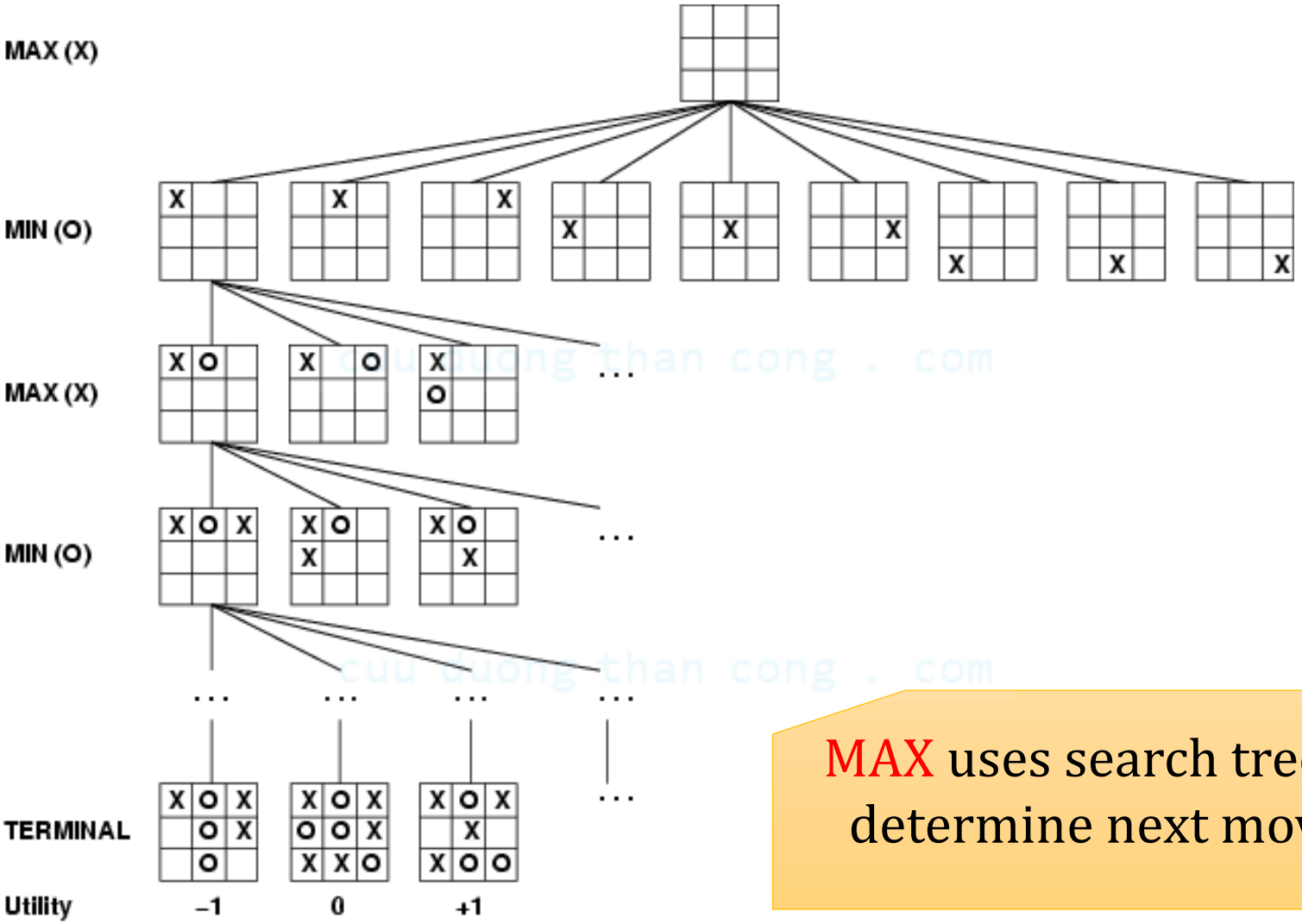
Primary Assumptions

- Assume only two players
- There is no element of chance
  - No dice thrown, no cards drawn, etc
- Both players have complete knowledge of the state of the game
  - Examples are chess, checkers and Go
  - Counter examples: poker
- Zero-sum games
  - Each player wins (+1), loses (0), or draws (1/2)
- Rational Players
  - Each player always tries to maximize his/her utility

Game Setup (Formulation)

- Two players: MAX and MIN
- MAX moves first and then they take turns until the game is over
  - Winner gets reward, loser gets penalty.
- Games as search:
  - S_0 – Initial state: how the game is set up at the start
    - e.g. board configuration of chess
  - PLAYER(s): MAX or MIN is playing
  - ACTIONS(s) – Successor function: list of (move, state) pairs specifying legal moves.
  - RESULT(s, a) – Transition model: result of a move a on state s
  - TERMINAL-TEST(s): Is the game finished?
  - UTILITY(s, p) – Utility function: Gives numerical value of terminal states s for a player p
    - e.g. win (**+1**), lose (**0**) and draw (**1/2**) in tic-tac-toe or chess

Tic-Tac-Toe Game Tree



MAX uses search tree to determine next move.

Chess

- Complexity:
  - b ~ 35
  - d ~100
  - search tree is ~ 10^{154} nodes (!!)

  →*completely impractical to search this*

- **Deep Blue**: (May 11, 1997)
  - Kasparov lost a 6-game match against IBM's Deep Blue (1 win Kasp – 2 wins DB) and 3 ties.
- In the future, focus will be to allow computers to **LEARN** to play chess rather than being **TOLD** how it should play

Deep Blue

- Ran on a parallel computer with 30 IBM RS/6000 processors doing alpha–beta search.
- Searched up to 30 billion positions/move, average depth 14 (be able to reach to 40 plies).
- Evaluation function: 8000 features
  - highly specific patterns of pieces (~4000 positions)
  - 700,000 grandmaster games in database
- Working at 200 million positions/sec, even Deep Blue would require $\mathbf{10^{100}}$ years to evaluate all possible games. (The universe is only 10^{10} years old.)
- Now: algorithmic improvements have allowed programs running on standard PCs to win World Computer Chess Championships.
  - Pruning heuristics reduce the effective branching factor to less than 3

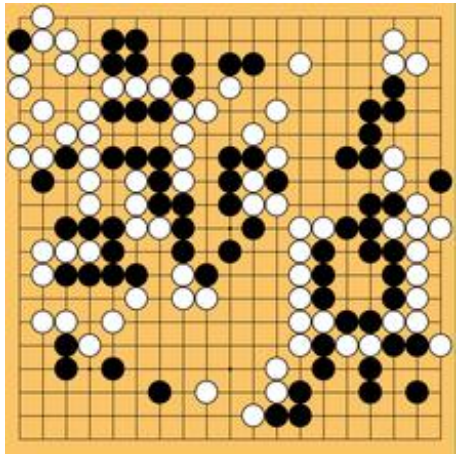
Checkers

- **Complexity:**
  - search tree is ~ 10^{18} nodes

  → requires 100k years if solving 106 positions/sec
- **Chinook** (1989-2007)
  - The first computer program to win the world champion title in a competition against humans.
  - 1990: won 2 games in competition with world champion Tinsley (final score: 2-4, 33 draws)
  - 1994: 6 draws
- Chinook's search:
  - Ran on regular PCs, used alpha-beta search.
  - Play perfectly using alpha-beta search combining with a database of 39 trillion endgame positions.

GO

1 million trillion trillion trillion trillion more configurations than chess!



- **Complexity:**
  - Board: 19x19 → Branching factor: 361, average depth ~ 200
  - ~ 10^{174} possible board configuration.
  - Control of territory is unpredictable until the endgame.
- **AlphaGo** (2016) by Google
  - Beat 9-dan professional Lee Sedol (4-1)
  - Machine learning + Monte Carlo search guided by a "value network" and a "policy network" (implemented using *deep neural network* technology)
  - Learn from human + Learn by itself (self-play games)

AlphaGo

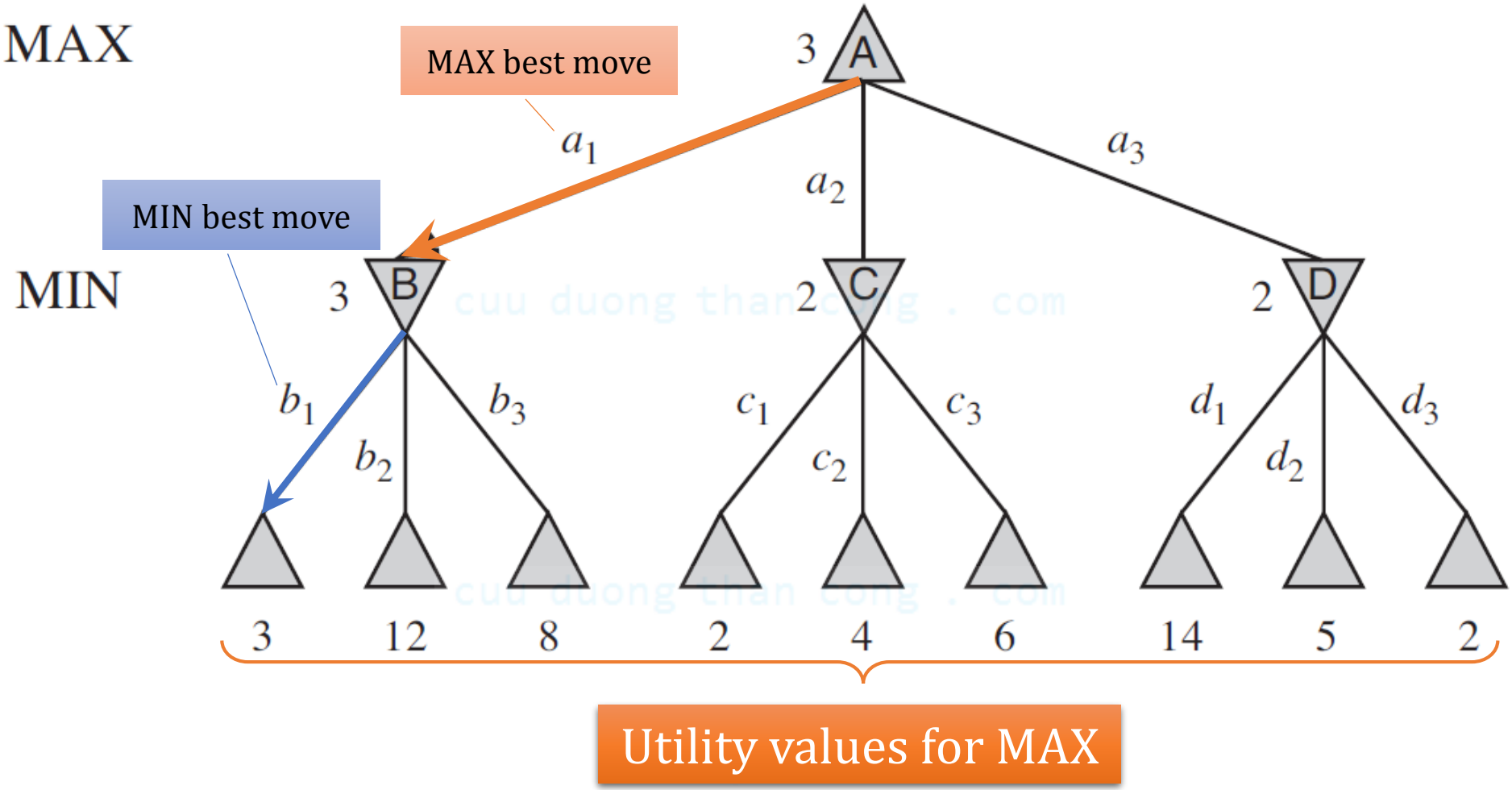
Optimal Decision in Games

- In normal search problem:
  - Optimal solution is a sequence of action leading to a goal state
- In games:
  - A search path that guarantee win for a player
  - The optimal strategy can be determined from the minimax value of each node

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

For MAX

A two-ply game tree



Minimax Algorithm

- John von Neumann devised a search technique, called Minimax
- You play against an opponent
  - Your objectives are in direct opposition
  - MAX tries to maximize his play while trying to minimize his opponent's (MIN's) play
- To implement Minimax, you need to know how good (or bad) your position is.
  - That is called the *Utility function*

Minimax Algorithm

- Definition of optimal play for MAX assumes MIN plays optimally:
  - maximizes worst-case outcome for MAX
- But if MIN does not play optimally, MAX will do even better
- **Minimax** uses depth first search to traverse the game tree
  - Complete depth-first exploration of the game tree

Minimax algorithm

MAX best move

```
function MINIMAX-DECISION(state) returns an action
    return arg max_{a ∈ ACTIONS(s)} MIN-VALUE(RESULT(state, a))

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

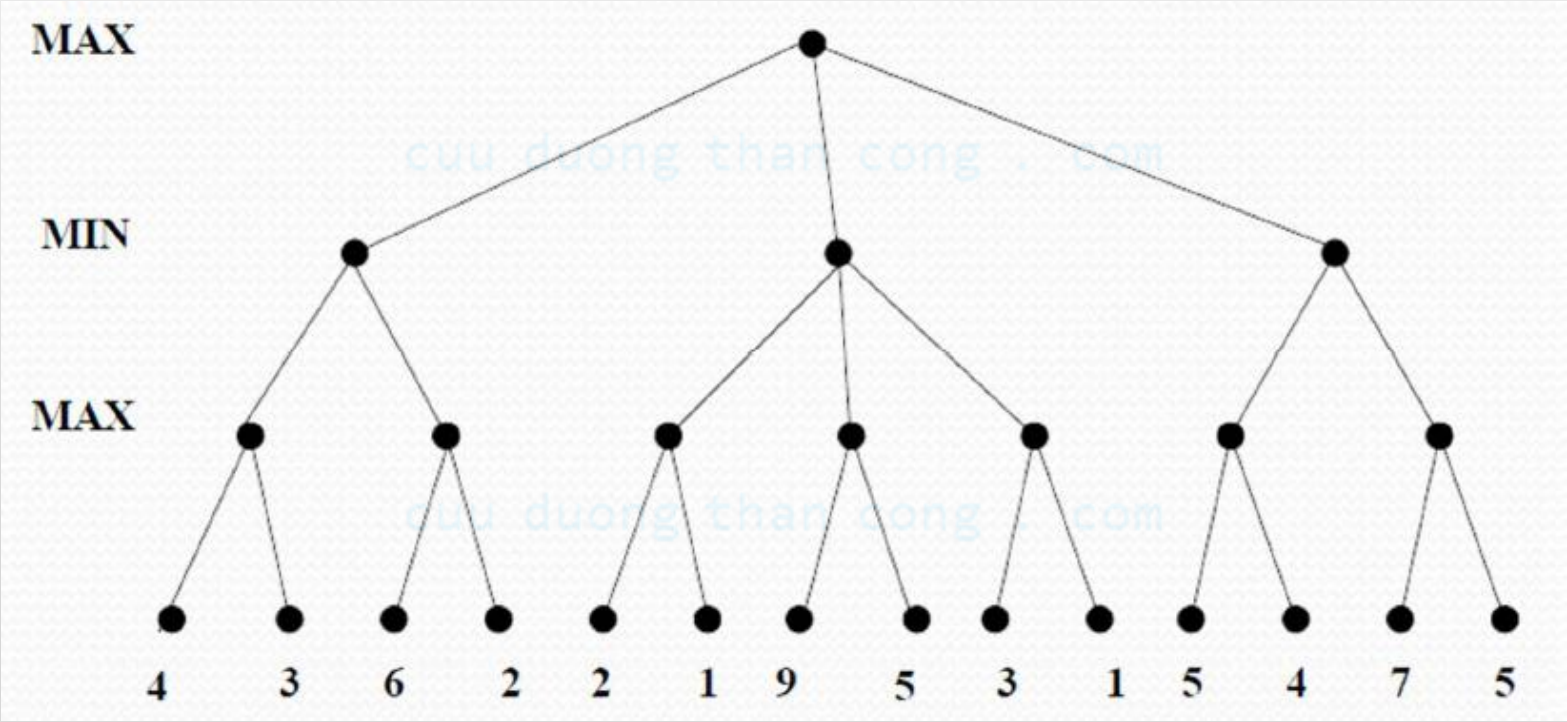
Properties of minimax

- 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 ≈ 35, m ≈100 for "reasonable" games
→ exact solution completely infeasible

QUIZ

Calculate the utility value for the remaining nodes. Which node should MAX and MIN choose?



Problem with Minimax Search

- Number of game states is *exponential* in the number of moves.
  - Solution: Do not examine every node

  →*pruning: Remove branches that do not influence final decision*

- Bounded lookahead
  - Limit depth for each search
  - This is what chess players do: look ahead for a few moves and see what looks best

α-β pruning

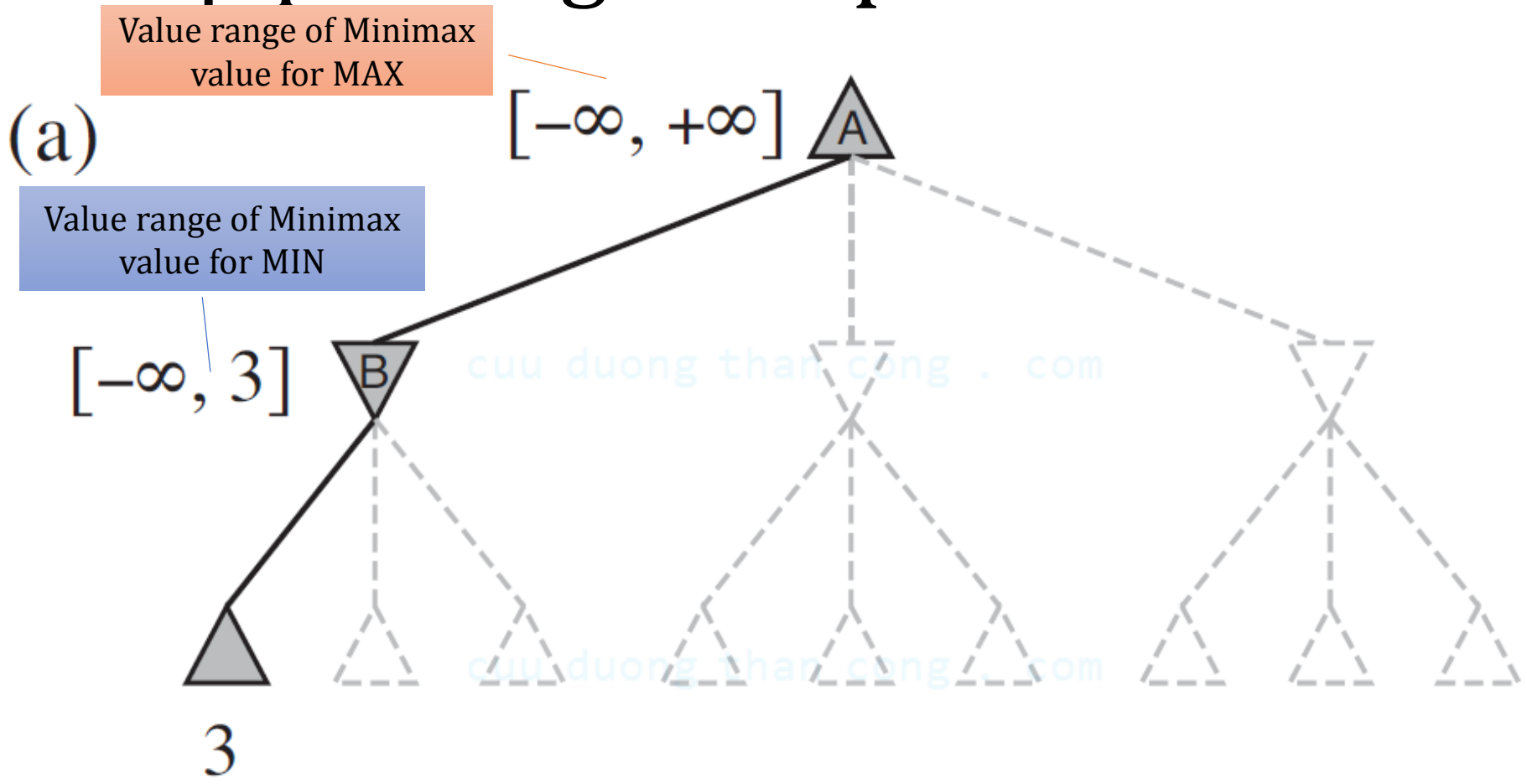
- Idea:
  - If a move A is determined to be worse than move B that has already been examined and discarded, then examining move A once again is ***pointless***.
    - α: best already explored option (utility value) along path to the root for MAX
    - β: best already explored option (utility value) along path to the root for MIN

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

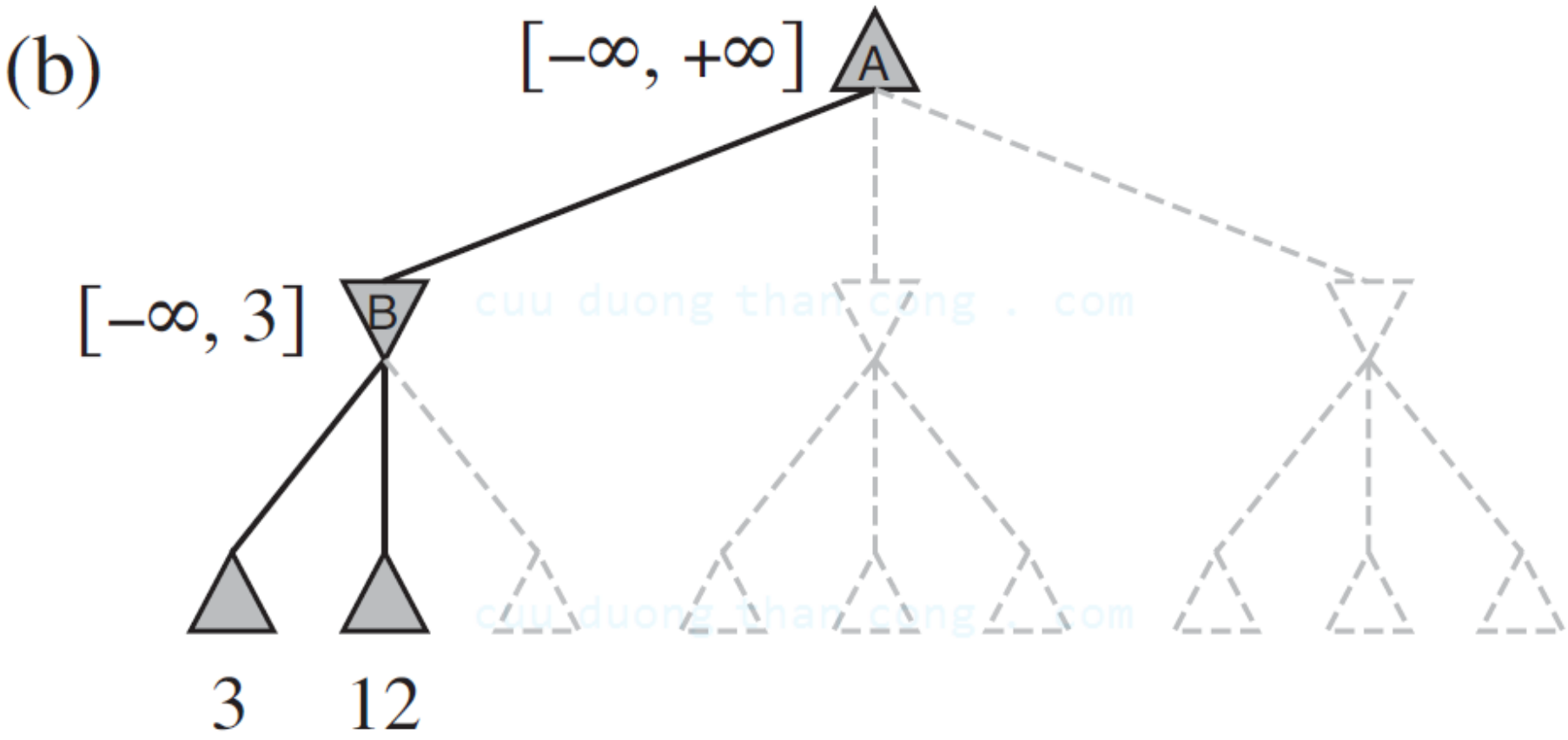
```
function MAX-VALUE(state, α, β) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← −∞
    for each a in ACTIONS(state) do
        v ← MAX(v, MIN-VALUE(RESULT(s,a), α, β))
        if v ≥ β then return v
        α ← MAX(α, v)
    return v
```

```
function MIN-VALUE(state, α, β) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← +∞
    for each a in ACTIONS(state) do
        v ← MIN(v, MAX-VALUE(RESULT(s,a) , α, β))
        if v ≤ α then return v
        β ← MIN(β, v)
    return v
```

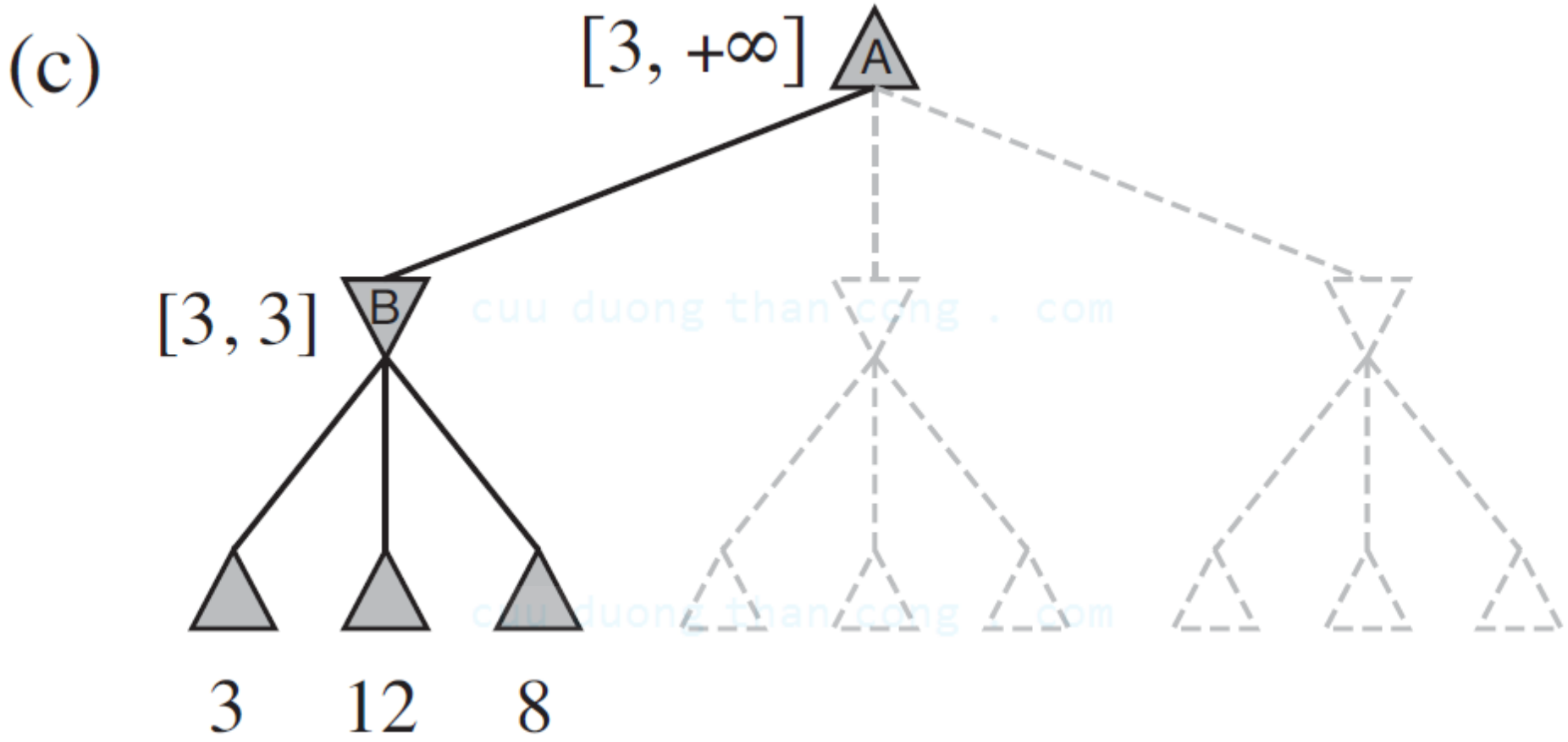
α-β pruning example



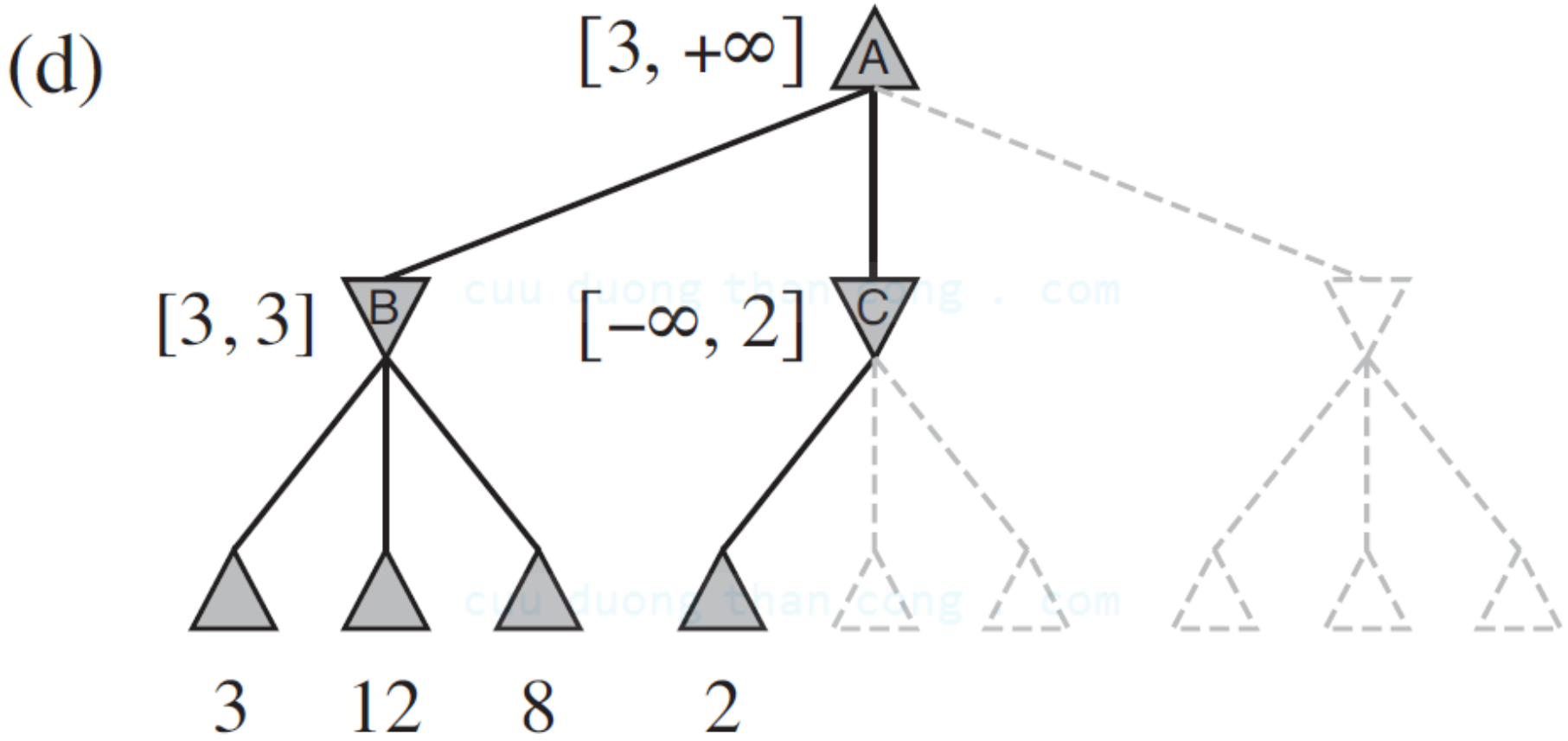
α-β pruning example



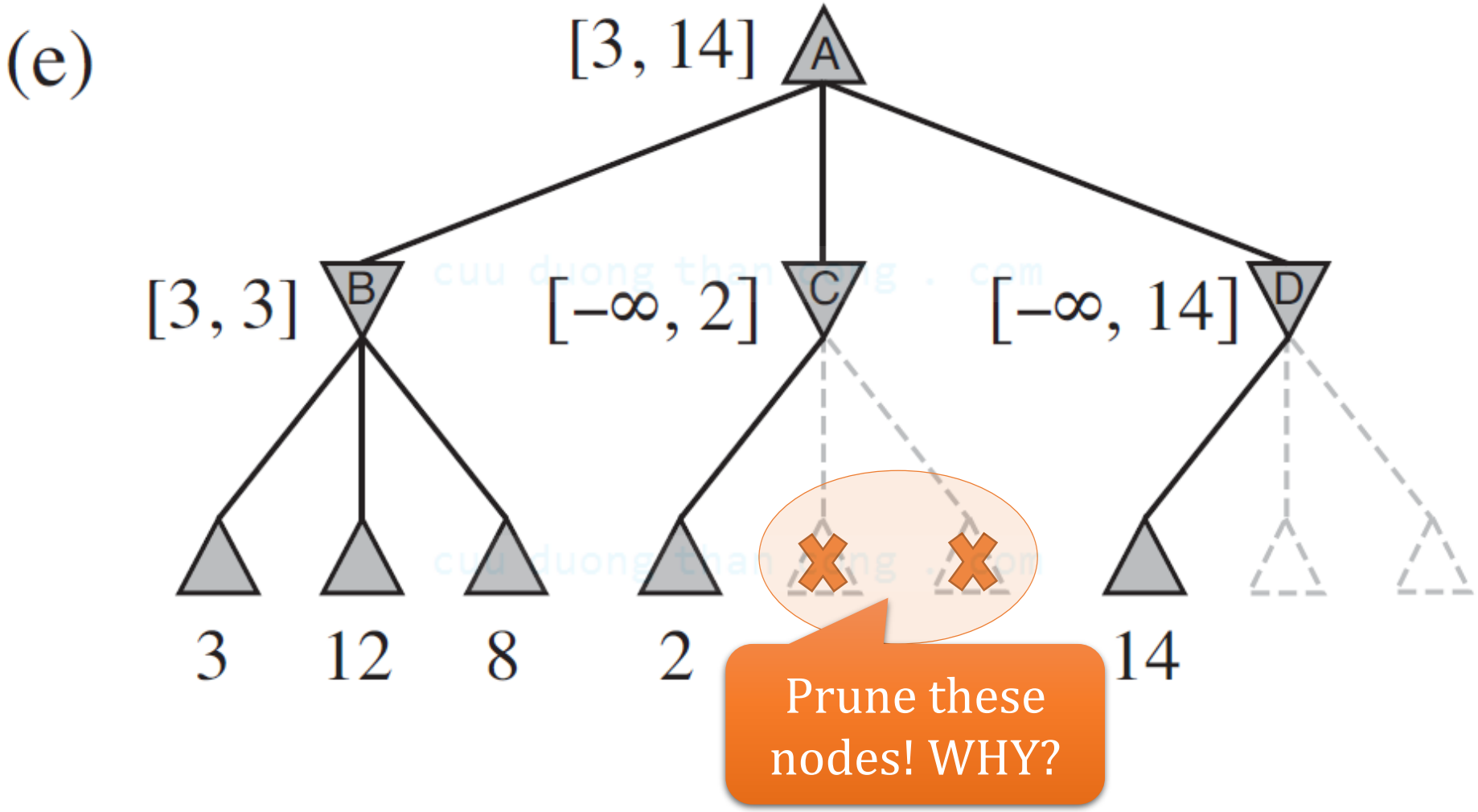
α-β pruning example



α-β pruning example



α-β pruning example

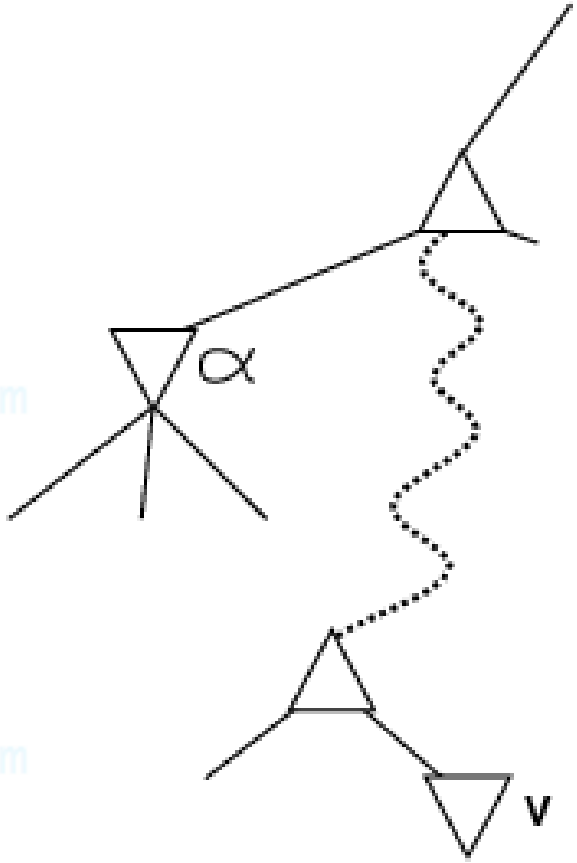


Properties of α-β pruning

- Pruning does not affect final result
  - Best case: Pruning can reduce tree size
  - Worst case: as good as Minimax algorithm
- Good move ordering improves effectiveness of pruning
- With "perfect ordering," time complexity = $O(b^{m/2})$
  - → doubles depth of search
- In chess, Deep Blue achieved reduced the depth from 38 to 6

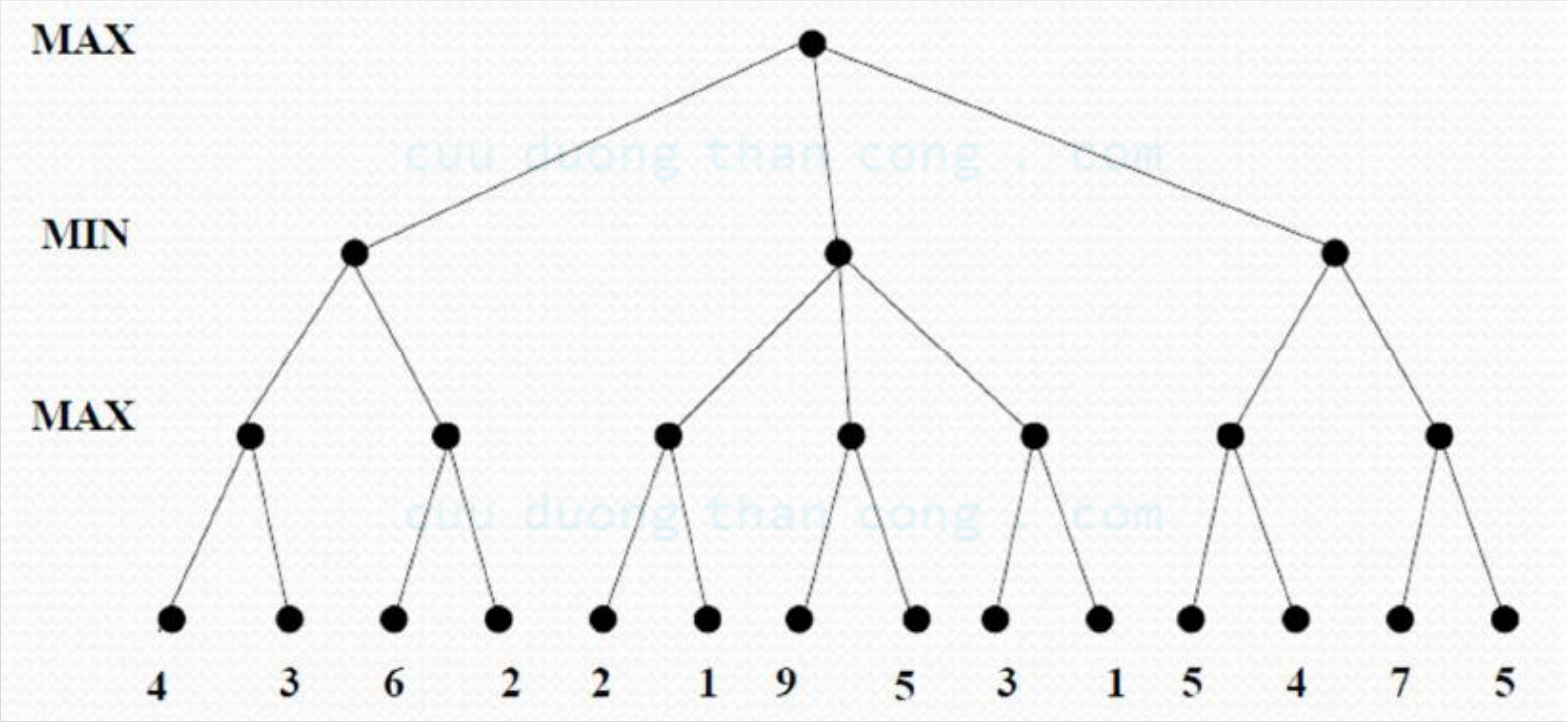
Why is it called α-β?

- α is the value of the best (i.e., highest-value) choice found so far at any choice point along the path for *max*
- If *v* is worse than α, *max* will avoid it
  - → prune that branch
- Define β similarly for *min*



QUIZ

Calculate the utility value for the remaining nodes. Which node(s) should be pruned?



Imperfect, Real-time Decisions

- Both Minimax and α-β pruning search all the way to terminal states
  - This depth is usually not practical because moves must be made in a reasonable amount of time (~ minutes)
- Standard approach:
  - cutoff test:

    e.g., depth limit
  - evaluation function

    = estimated desirability of position (win, lose, tie?)

Evaluation functions

- For chess, typically linear weighted sum of features

$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

Where w_i: the value of the i^{th} chess piece

- e.g., $w_1 = 9$ with $f_1(s)$ = (#white queen) – (#black queen), etc.
- e.g. q = #queens, r = #rooks, n = #knights, b = #bishops, p=#pawns

  → $Eval(s) = 9q + 5r + 3b + 3n + p$

Cutting off search

- *Minimax Cutoff* is identical to *MinimaxValue* except
  1. *Terminal?* is replaced by *Cutoff?*
  2. *Utility* is replaced by *Eval*
- Does it work in practice?
  - $b^m = 10^6$, $b=35$ → $m=4$
  - 4-ply lookahead is a hopeless chess player!
  - 4-ply ≈ human novice
  - 8-ply ≈ typical PC, human master
  - 12-ply ≈ Deep Blue, Kasparov

Summary

- Games are fun to work on!
- They illustrate several important points about AI
  - perfection is unattainable → must approximate
  - good idea to think about what to think about

More reading (textbook, chapter 5.5—5.7)

- Search vs lookup
- Stochastic games
- Partially observable games
- State-of-the-art game programs

Next week

- Wednesday (Jun 13):
  - Midterm Examination
  - Close-book
  - 45 mins
- Lecture:
  - Constraint Satisfaction Problems