Review

- Uninformed Search
  - Breadth-first search
  - Uniform-cost search
  - Depth-first search
  - Depth-limited search
  - Iterative deepening search

- Informed Search
  - Greedy search
  - A* Search

HomeWorks

- Uninformed Search
  - Breadth-first search
  - Uniform-cost search
  - Depth-first search

HW 1 <Due: Jan 30>

- Informed Search
  - A* Search

HW 2 <Due: Feb 13>

MidTerm

- In-Class
- Wednesday, March 4

Adversarial Search

Adversarial Search

- In which we examine the problems that arise
  - when we try to plan ahead to get the best result
    - in a world that includes a <u>hostile</u> agent (other agent planning against us).

Games

- Adversarial search problems
  - Competitive environments in which goals of multiple agents are in conflict (often known as games)
- Game theory
  - Views any multi-agent environment as game
  - Provided the impact of each agent on the others is "significant"
- Classic AI games
  - Deterministic, turn-taking, two-player, perfect information
- Game playing is idealization of worlds in which hostile agents act so as to diminish one's well-being!
  - Games problems are like real world problems

Classic AI Games

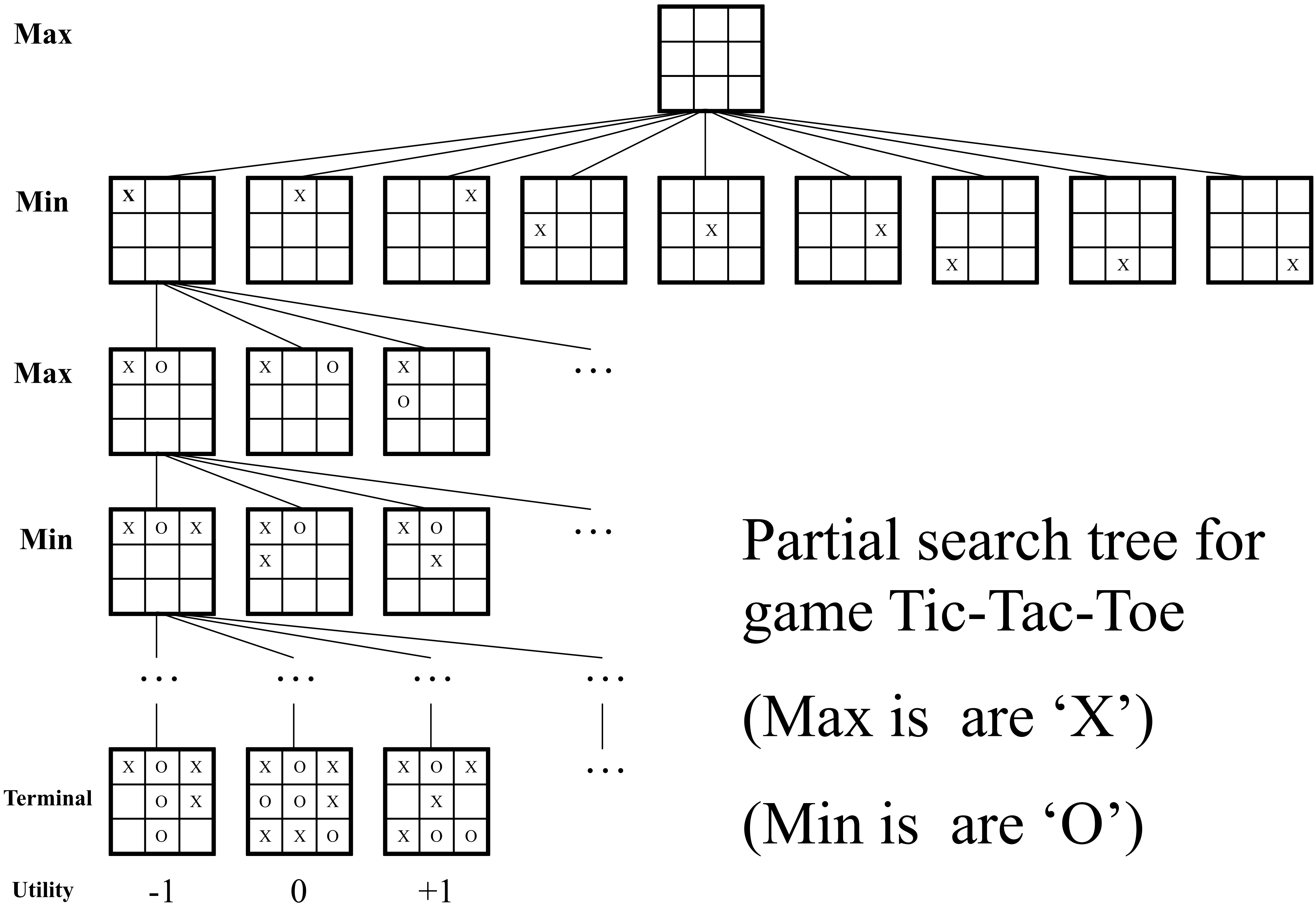
- State of game easy to represent
- Agents usually restricted to fairly small number of well-defined actions
- Opponent introduces uncertainty
- Games usually much too hard to solve
  - Chess
    - Branching factor 35
    - Often go to 50 moves by each player
    - About $35^{2*50} = 35^{100}$ nodes!
- Good domain to study

AI Game Play

- Define optimal move and algorithm for finding it
- Ignore portions of search tree that make no difference to final choice
    - evaluation functions to approximate the true utility of a state without complete tree search.
        - Pruning

A Game Defined as Search Problem

- Initial state
  - Board position
  - Whose move it is
- Operators (successor function)
  - Defines legal moves and resulting states
- Terminal (goal) test
  - Determines when game is over (terminal states)
- Utility (objective, payoff) function
  - Gives numeric value for the game outcome at terminal states
  - e.g., {win = +1, loss = -1, draw = 0}



Partial search tree for game Tic-Tac-Toe

(Max is are ‘X’)

(Min is are ‘O’)

Optimal Strategies: Perfect Decisions in Two-Person Games

- Two players
  - MAX
  - MIN
- (Assume) MAX moves first, then they take turns moving until game over
- At end, points awarded to winning player
  - Or penalties given to loser
- Can formulate this gaming structure into a search problem

An Opponent

- If were normal search problem, then MAX (you/agent) need only search for sequence of moves leading to winning state
- But, MIN (the opponent) has input
- MAX must use a “strategy” that will lead to a winning state regardless of what MIN does
  - Strategy picks best move for MAX for each possible move by MIN

Techniques

- “Minimax”
- Alpha-beta pruning

Minimax

- Determines the best moves for MAX, assuming that MAX and opponent (MIN) play <u>perfectly</u>
  - MAX attempts to maximize its score
  - MIN attempts to minimize MAX's score
- Decides best first move for MAX
- Serves as basis for analysis of games and algorithms

Minimax

- Perfect play for deterministic, perfect-information games
- Two players: MAX, MIN
  - MAX moves first, then take turns until game is over
  - Points are awarded to winner
    - Sometimes penalties may be given to loser
- Choose move to position with highest *minimax* value
  - Best achievable payoff against best play
  - Maximizes the worst-case outcome for MAX

Minimax Algorithm

- Generate whole game tree (or from current state downward – depth-first process online)
  - Initial state(s) to terminal states

Minimax Algorithm

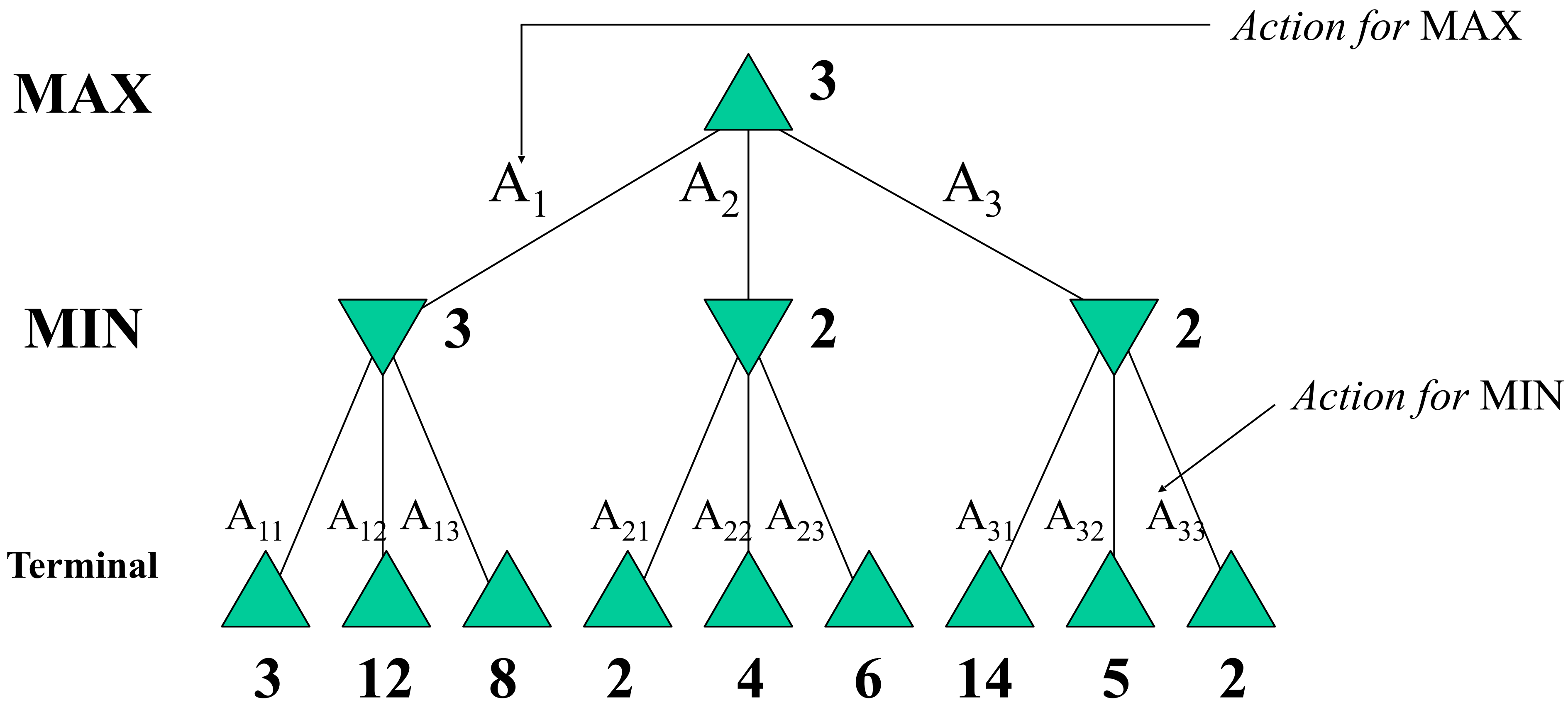
- Generate whole game tree (or from current state downward – depth-first process online)
  - Initial state(s) to terminal states
- Apply utility function to terminal states
- Use utilities at terminal states to determine utility of nodes one level higher in tree
  - Find MIN's best attempt to minimize high payoff for MAX at terminal level

Minimax Algorithm

- Generate whole game tree (or from current state downward – depth-first process online)
  - Initial state(s) to terminal states
- Apply utility function to terminal states
- Use utilities at terminal states to determine utility of nodes one level higher in tree
  - Find MIN's best attempt to minimize high payoff for MAX at terminal level
- Continue backing up the values to the root
  - One layer at a time
- Value at root is determines the best payoff and opening move for MAX (minimax decision)

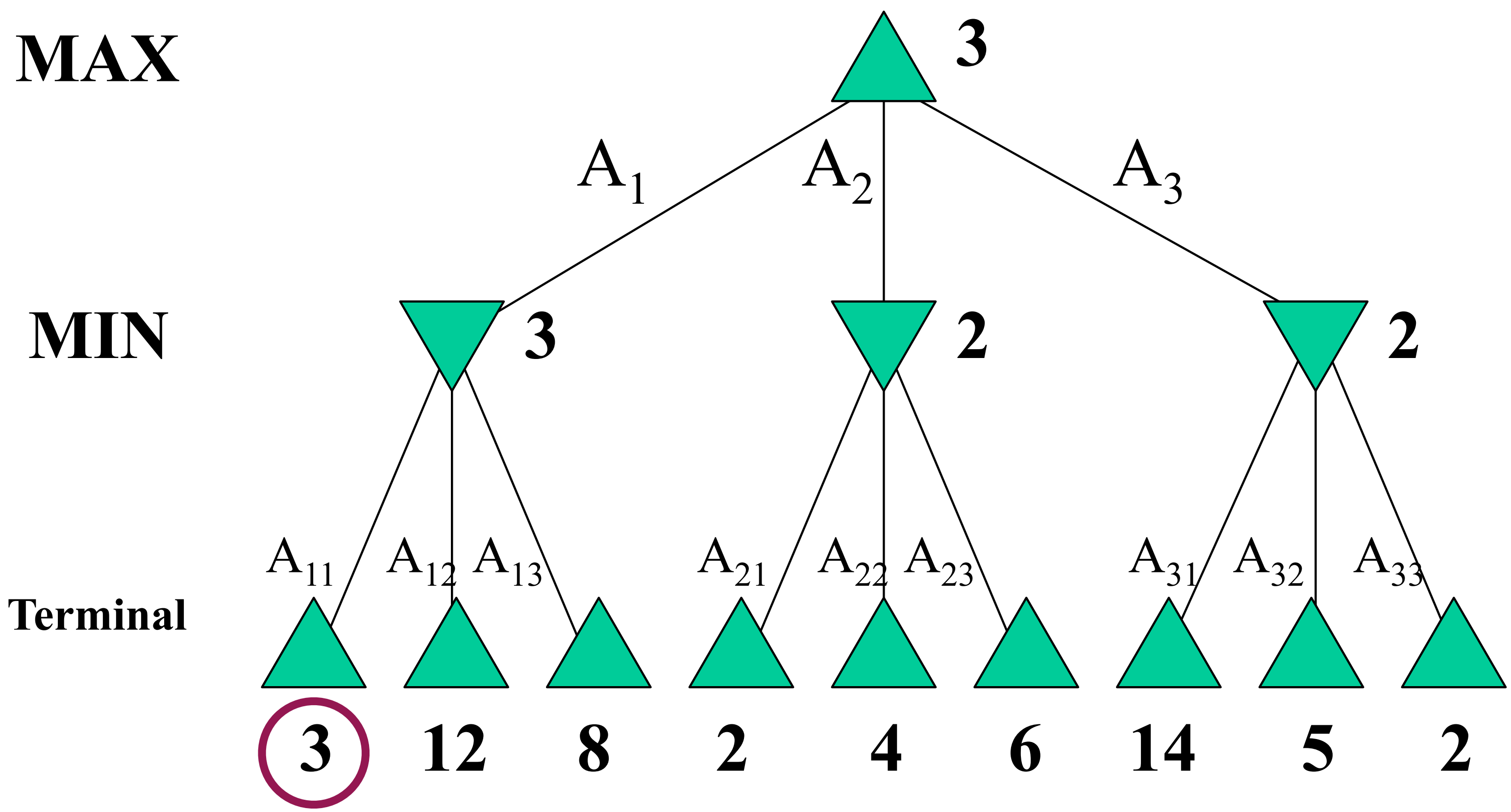
2-Ply Minimax Game

(one move for each player)



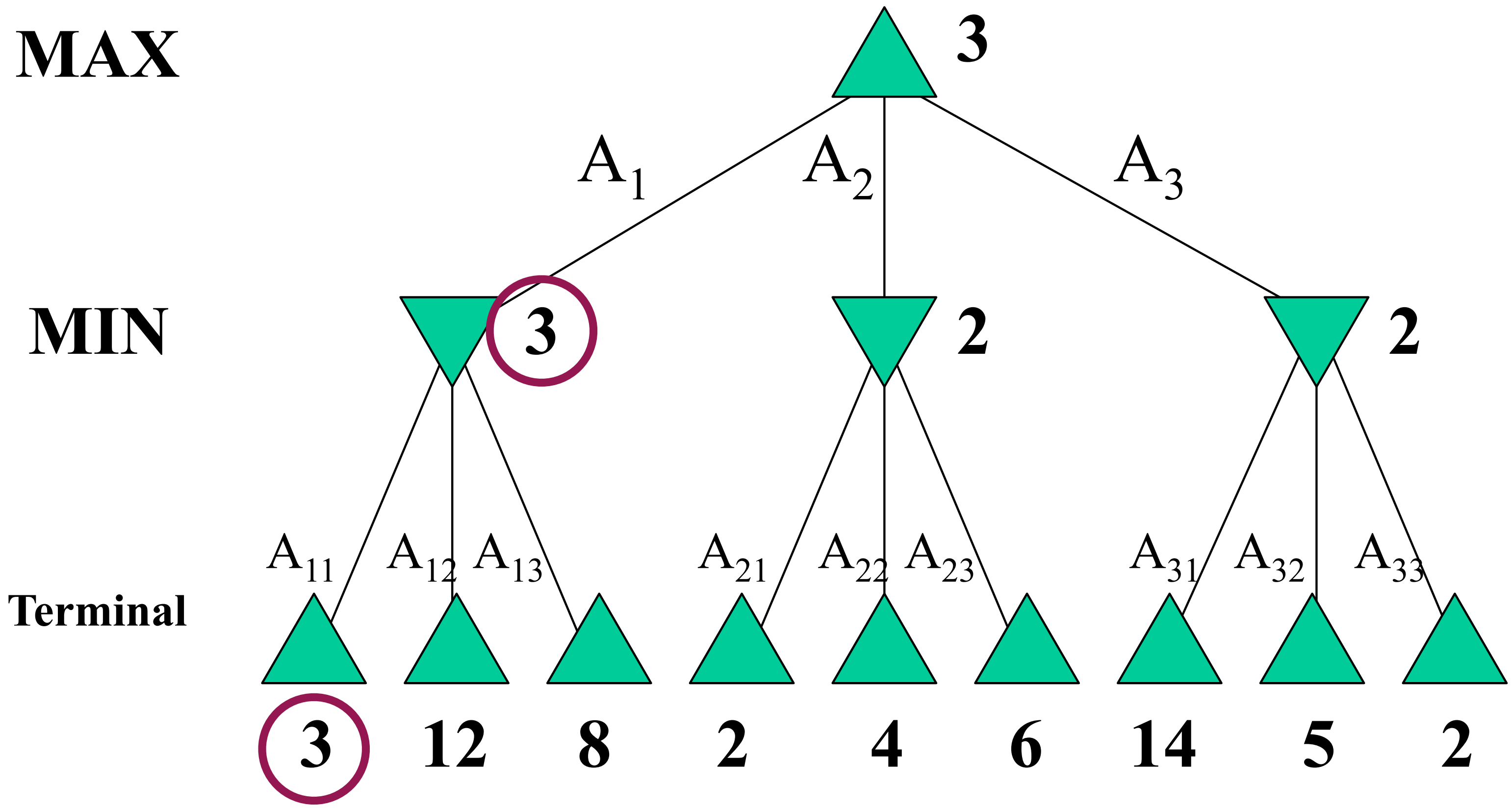
2-Ply Minimax Game

(one move for each player)



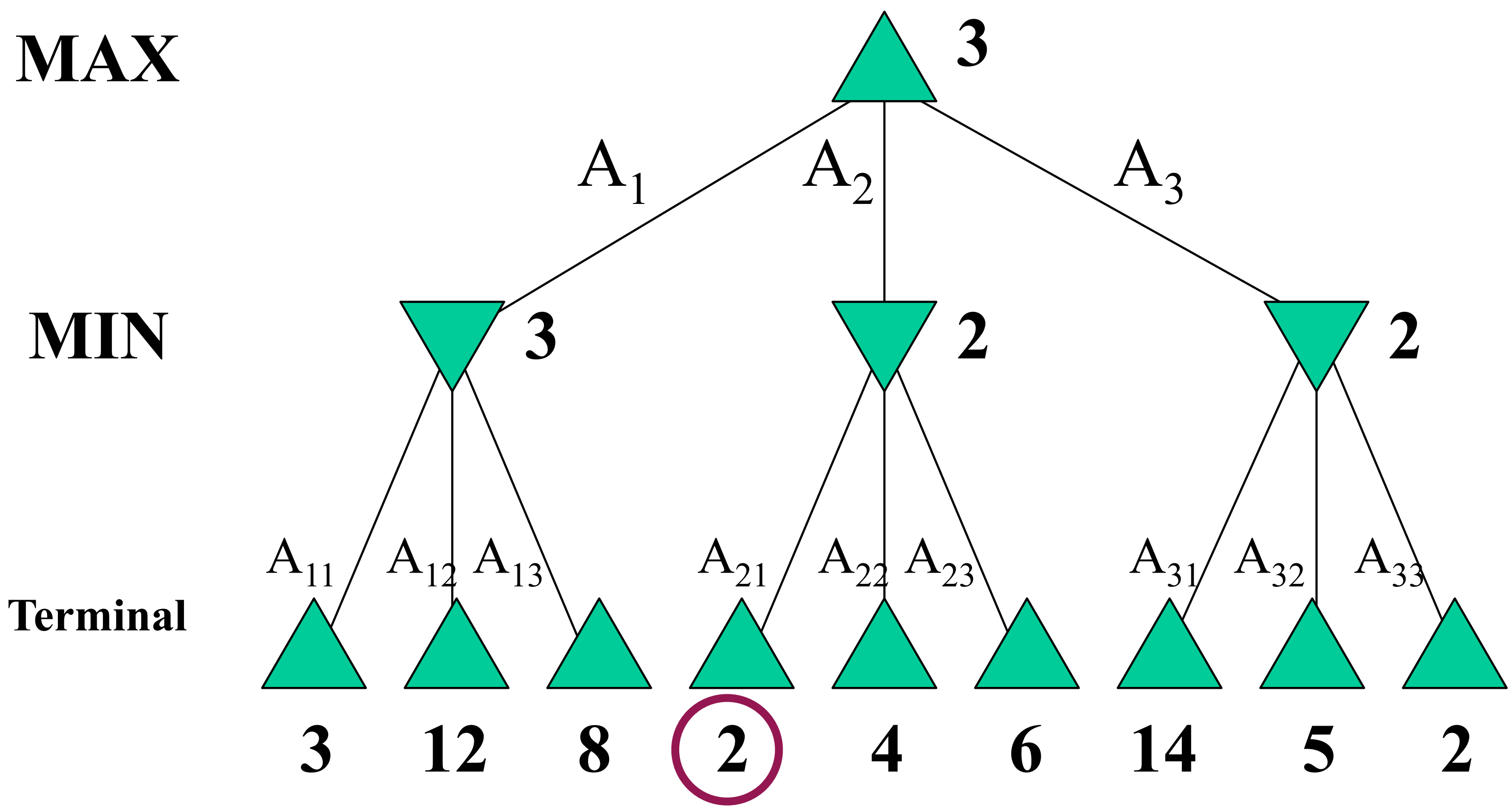
2-Ply Minimax Game

(one move for each player)



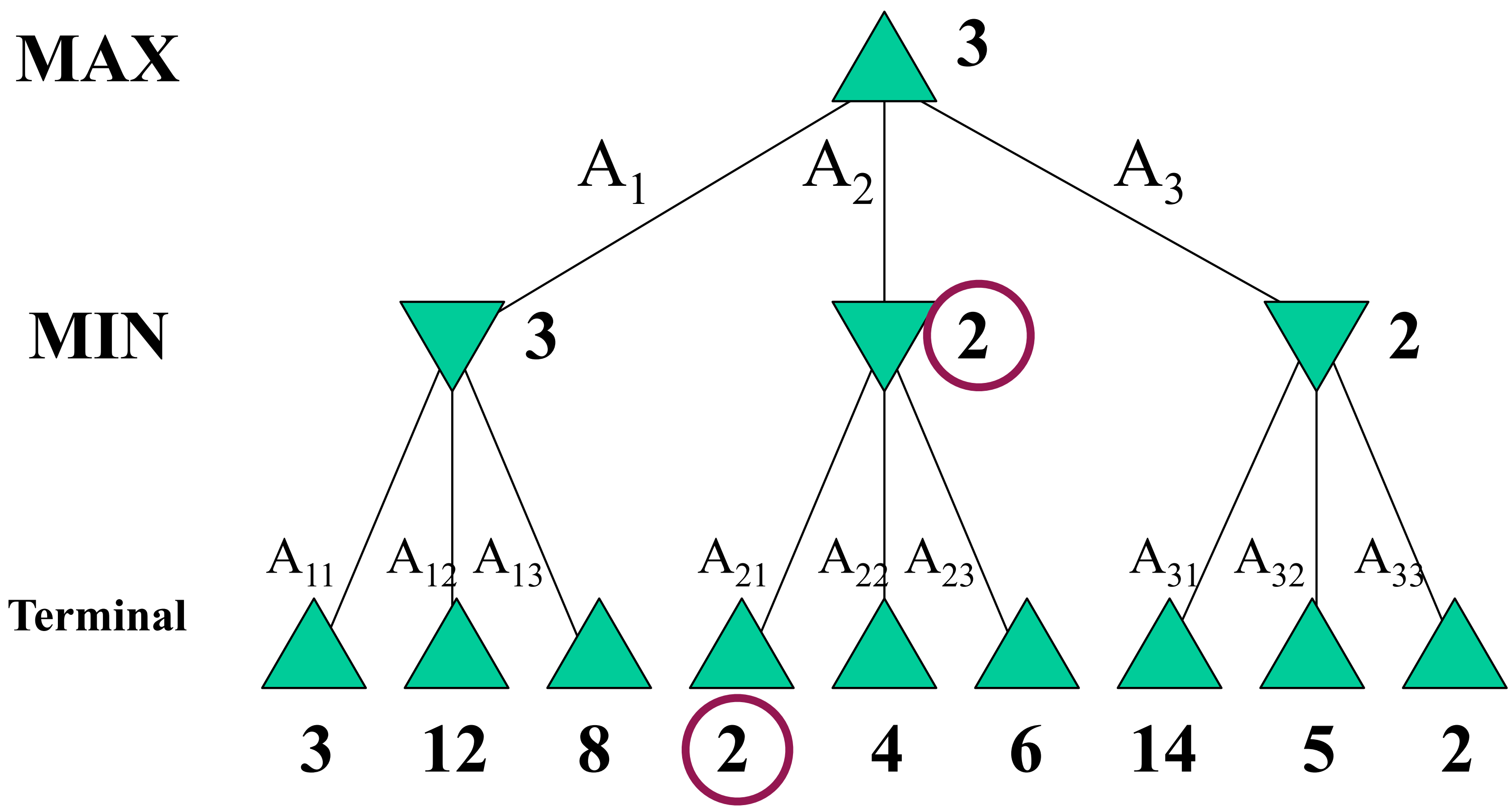
2-Ply Minimax Game

(one move for each player)



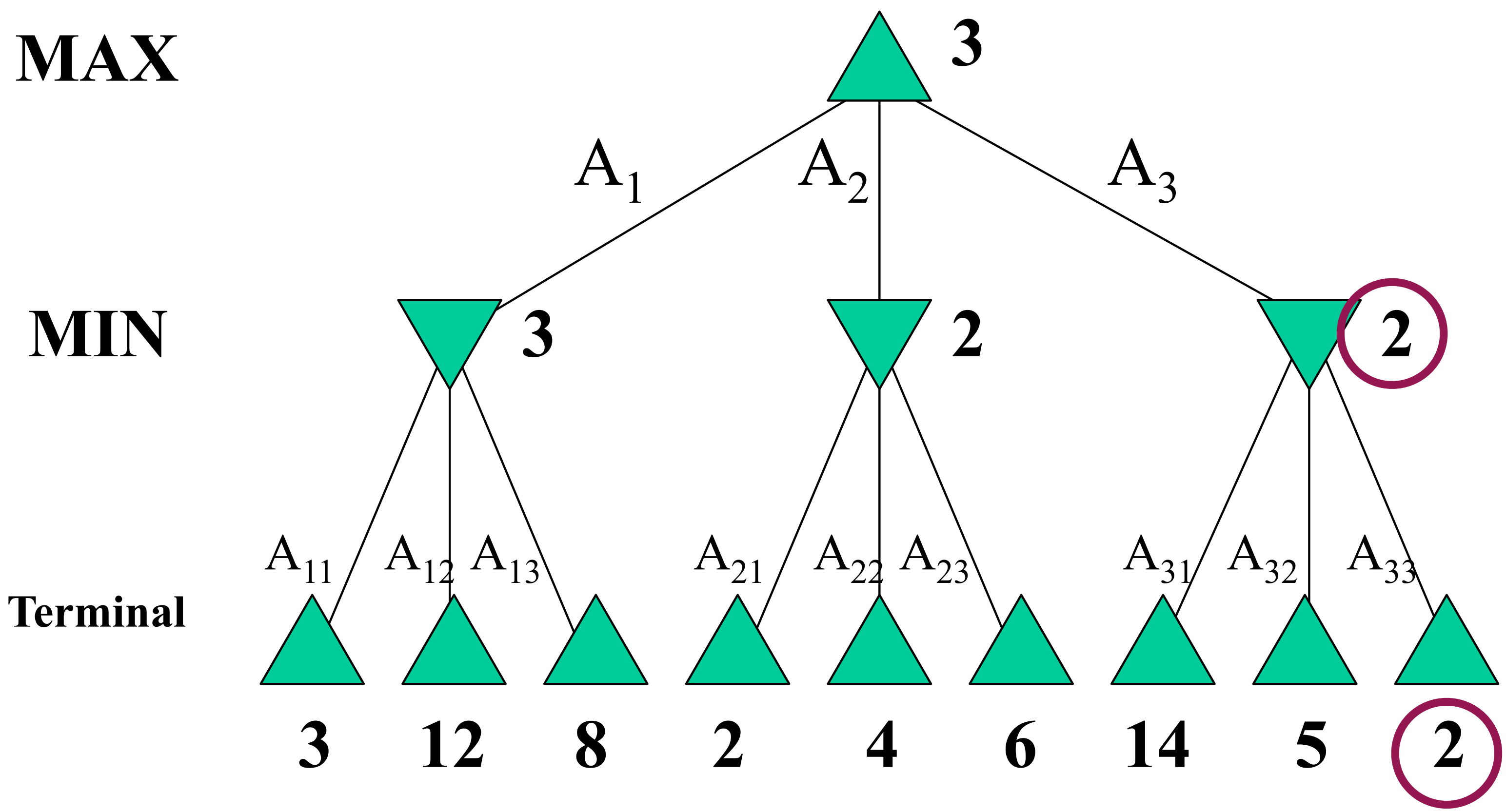
2-Ply Minimax Game

(one move for each player)



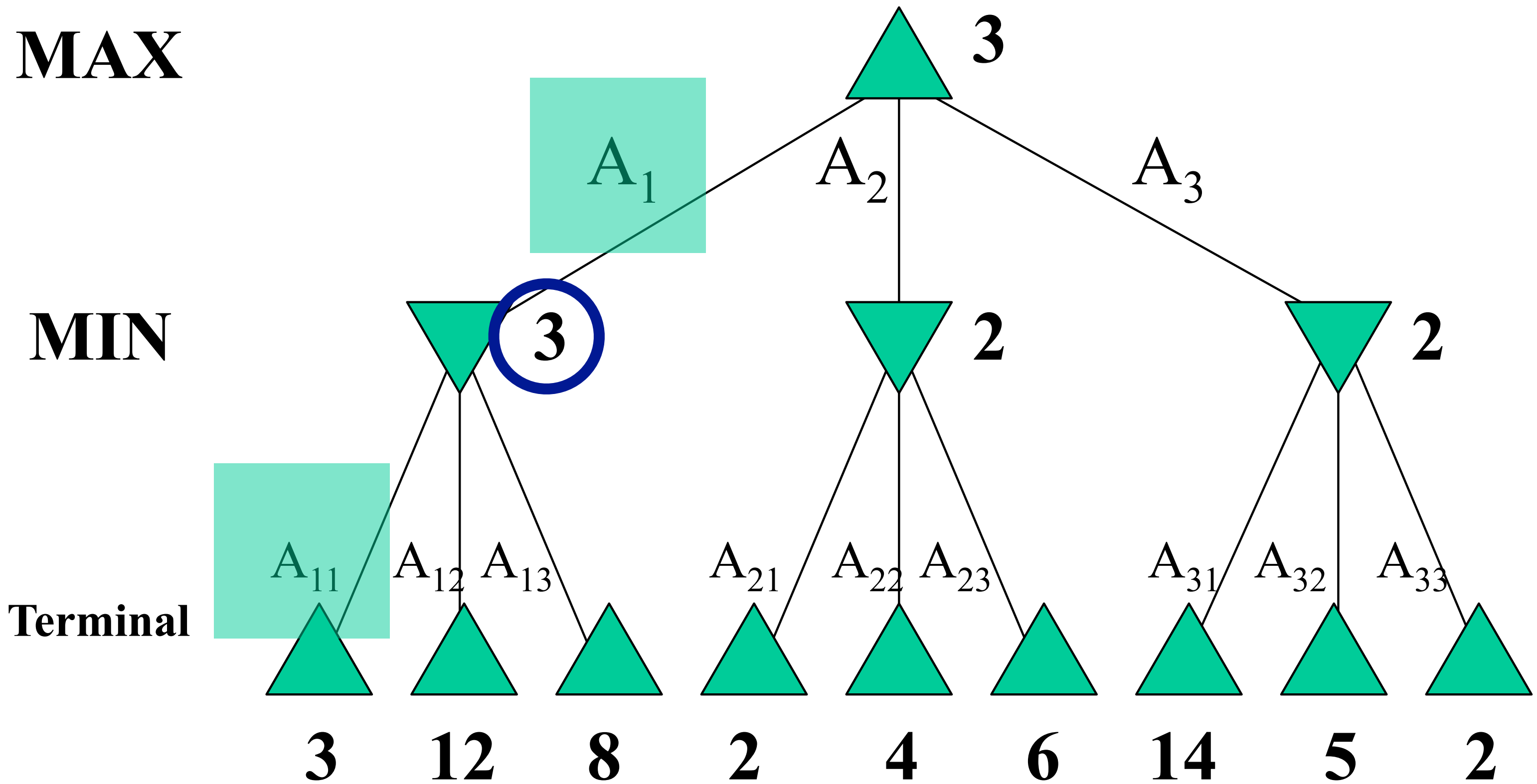
2-Ply Minimax Game

(one move for each player)



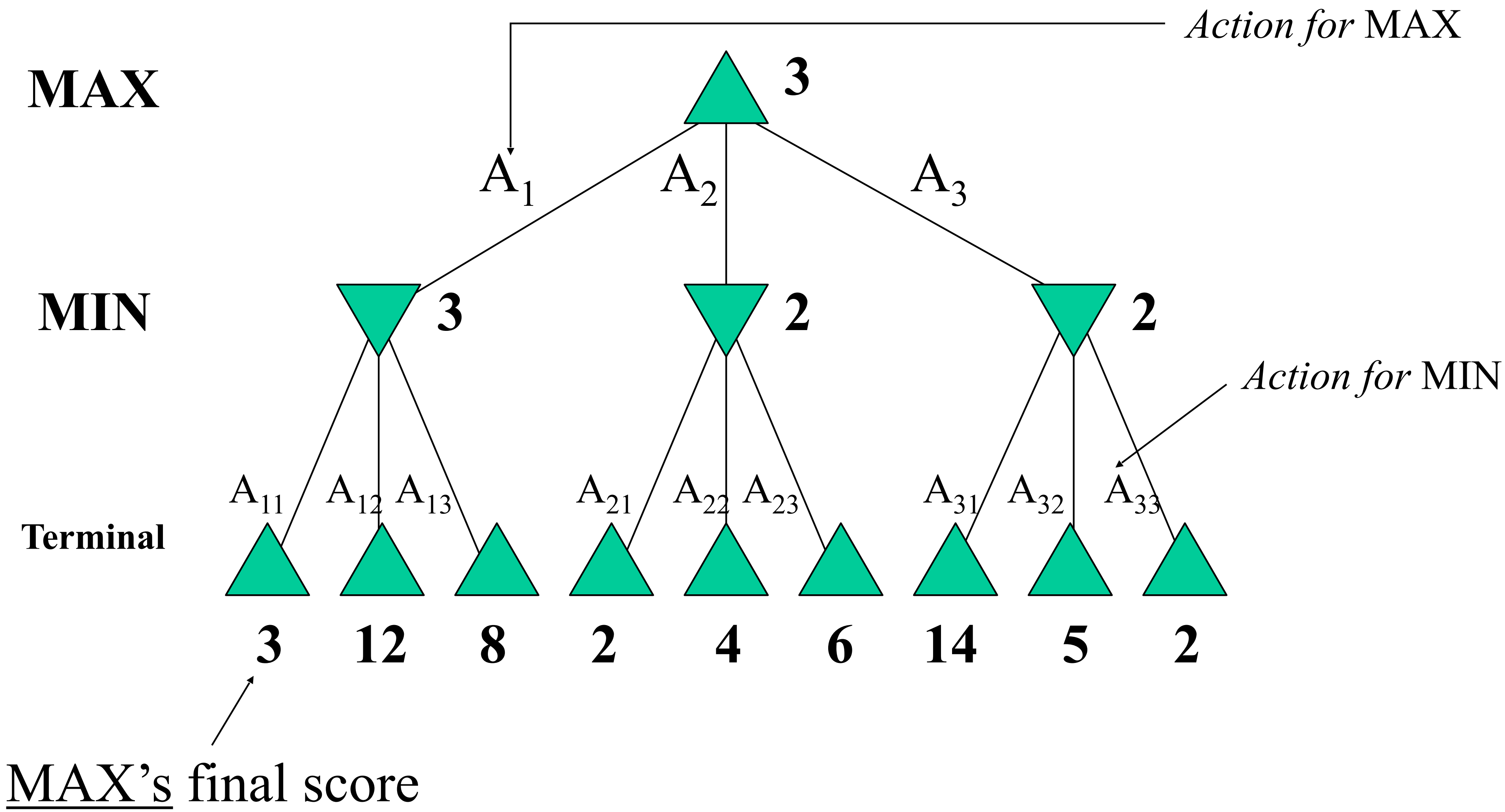
2-Ply Minimax Game

(one move for each player)



2-Ply Minimax Game

(one move for each player)



Properties of Minimax

- Complete
  - If tree is finite
- Time
  - Depth-first exploration
  - $O(b^m)$, max depth of m with b legal moves at each point (impractical for real games)
- Space
  - Depth-first exploration
  - $O(bm)$
- Optimality
  - Yes against an optimal opponent
  - Does even better when MIN not play optimally

Pruning

- Minimax search has to search large number of states
- But possible to compute correct minimax decision without looking at every node in search tree
- Eliminating a branch of search tree from consideration (without looking at it) is called <u>pruning</u>

Alpha-beta pruning

- Ignore portions of search tree that make no difference to final choice
- Prunes away branches that <u>cannot possibly influence</u> final minimax decision
- Returns <u>same</u> move as general minimax

Alpha-Beta Pruning

- Can be applied to trees of any depth
- Often possible to prune entire subtrees rather than just leaves

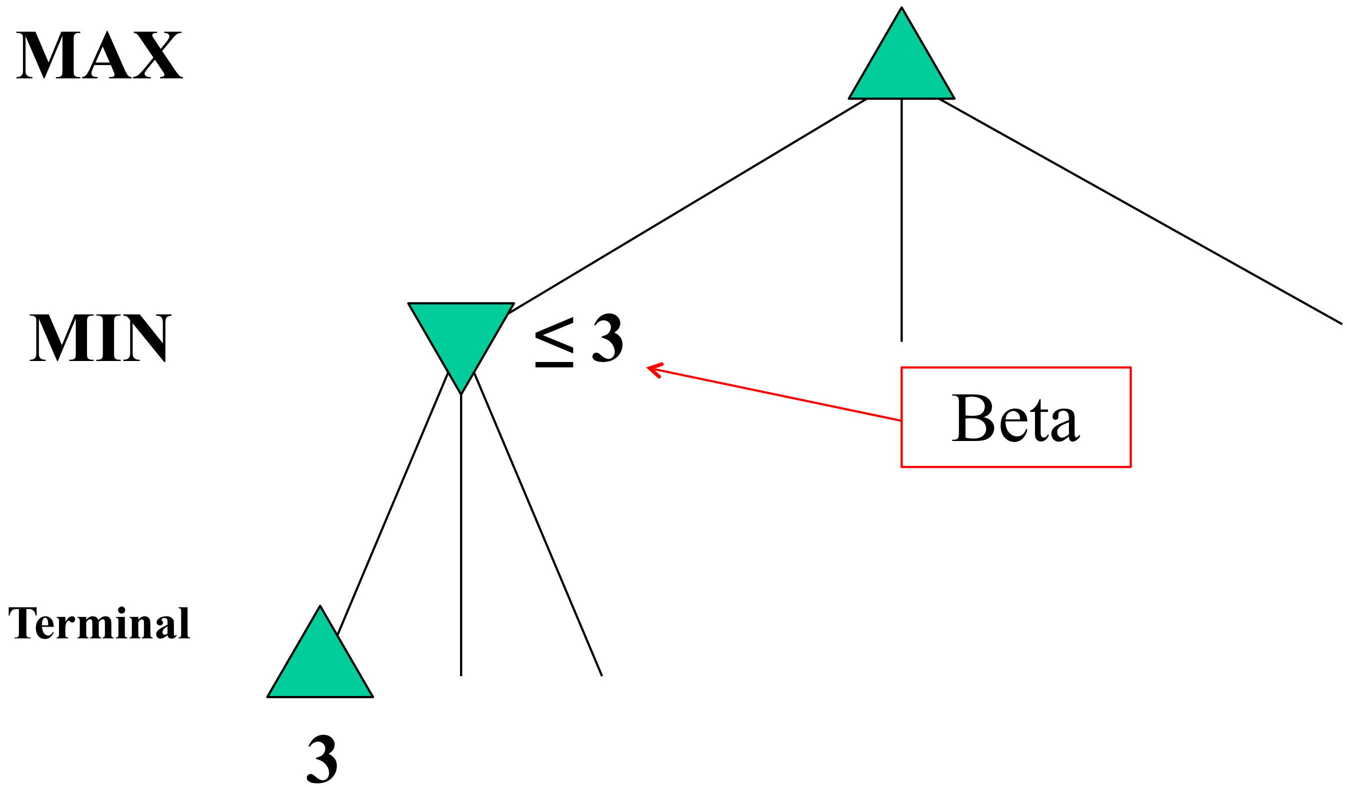
Alpha-Beta Pruning

- Can be applied to trees of any depth
- Often possible to prune entire subtrees rather than just leaves
- Alpha-beta
  - Alpha = value of best (highest-value) choice found so far at any choice point along path for MAX
    - In other words, the worst score (lowest) MAX could possibly get
    - Update alpha <u>only</u> during MAX’s turn/ply

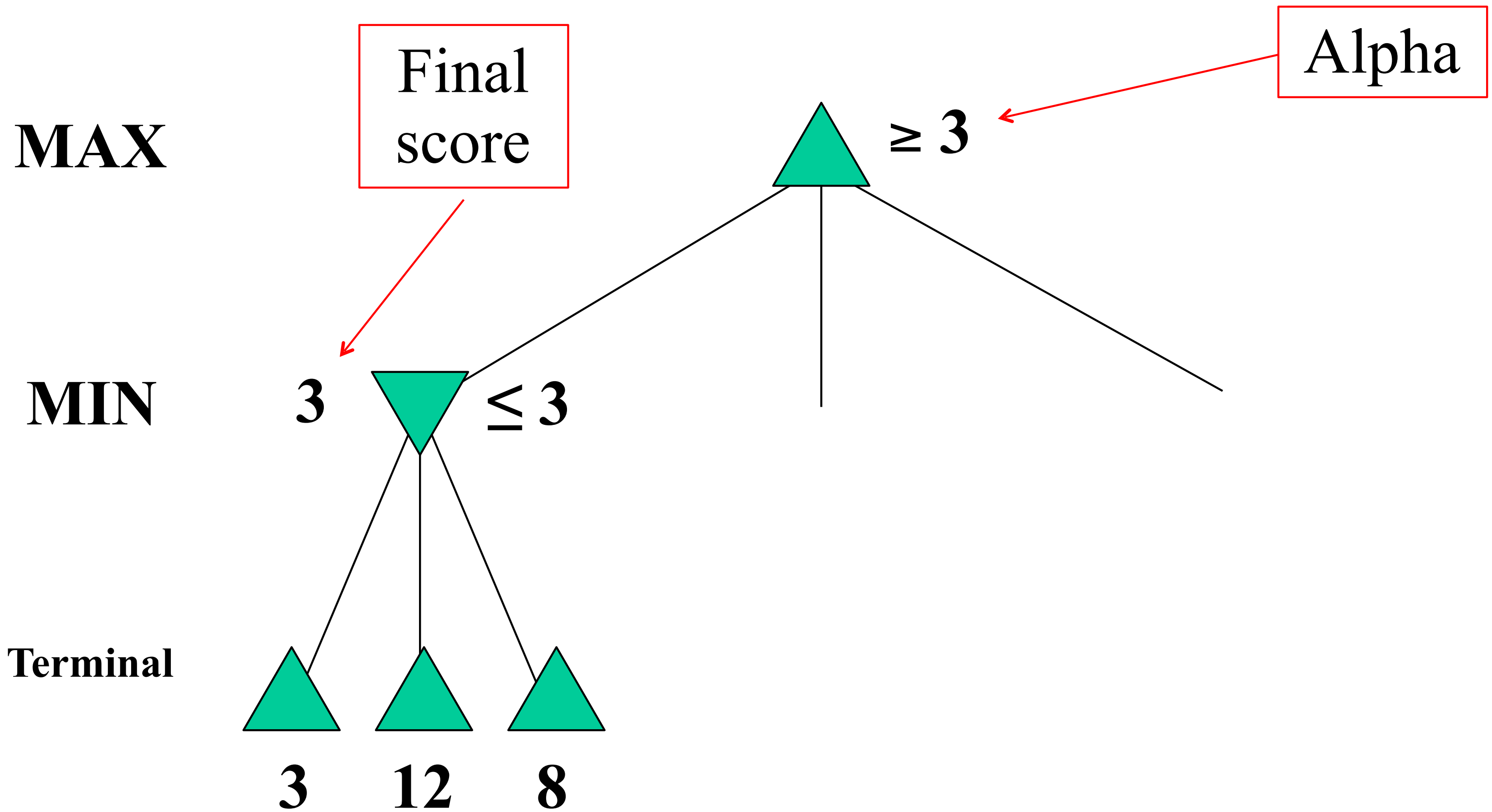
Alpha-Beta Pruning

- Can be applied to trees of any depth
- Often possible to prune entire subtrees rather than just leaves
- Alpha-beta
  - Alpha = value of best (highest-value) choice found so far at any choice point along path for MAX
    - In other words, the worst score (lowest) MAX could possibly get
    - Update alpha <u>only</u> during MAX's turn/ply
  - Beta = value of best (lowest-value) choice found so far at any choice point along path for MIN
    - In other words, the worst score (highest) MIN could possibly get
    - Update beta <u>only</u> during MIN's turn/ply

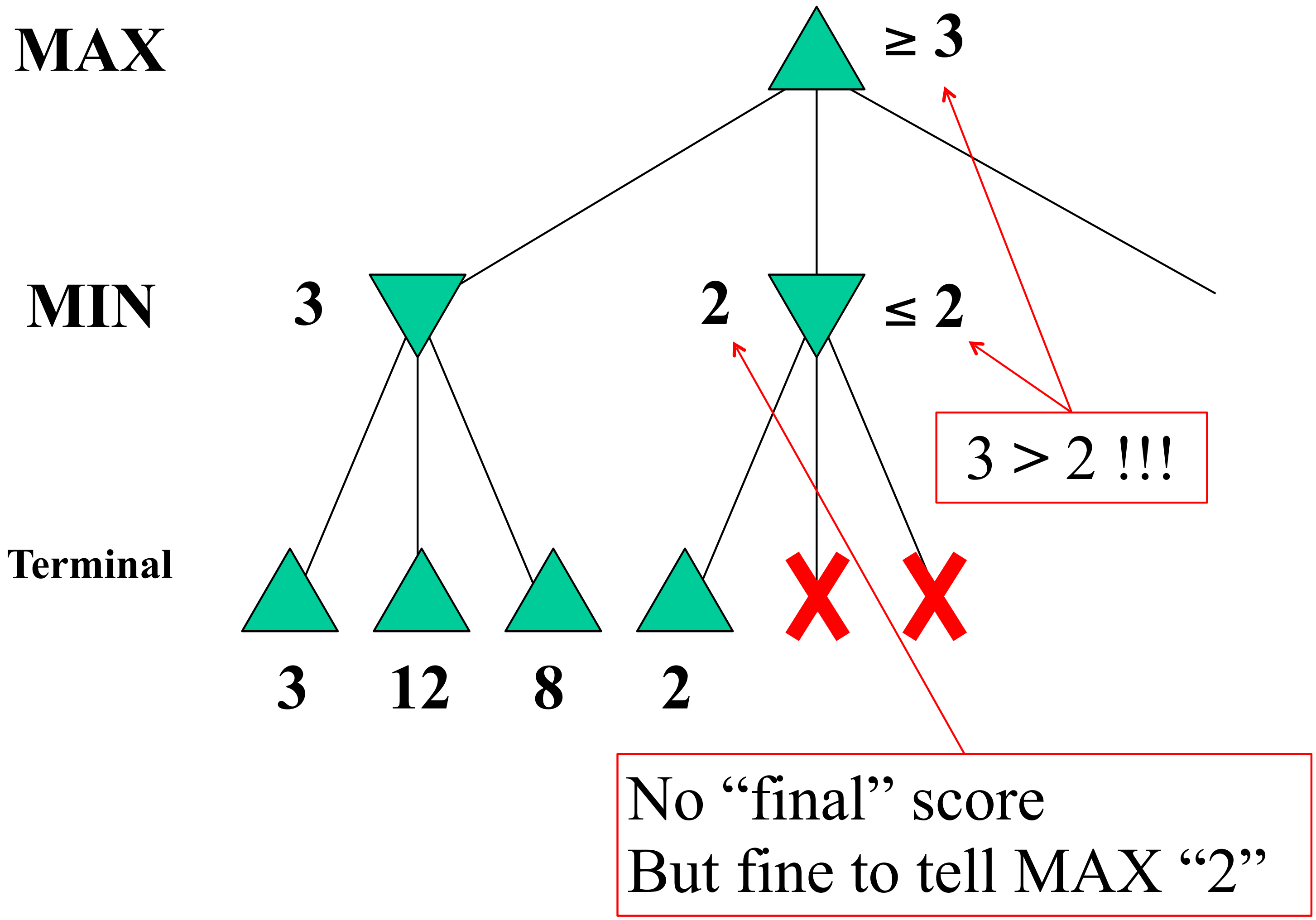
Alpha-Beta Pruning



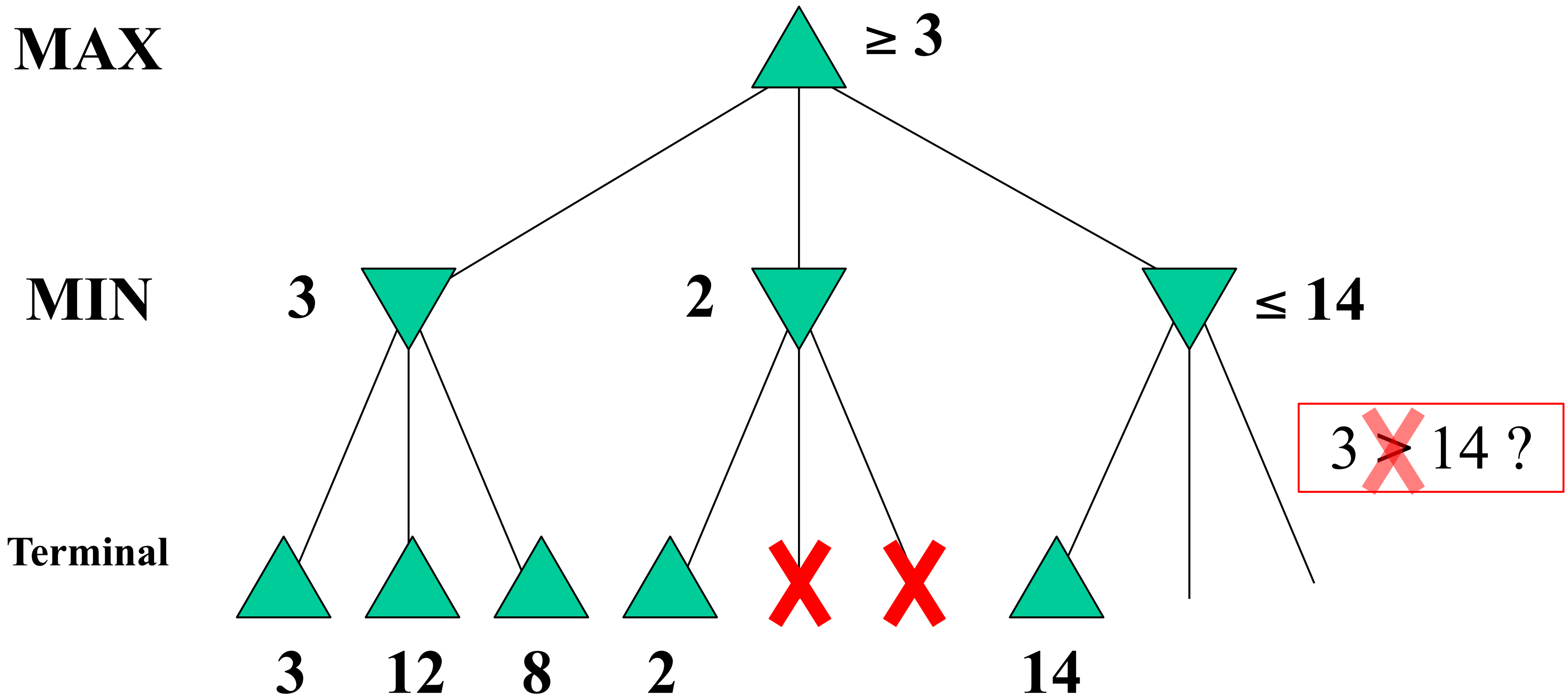
Alpha-Beta Pruning



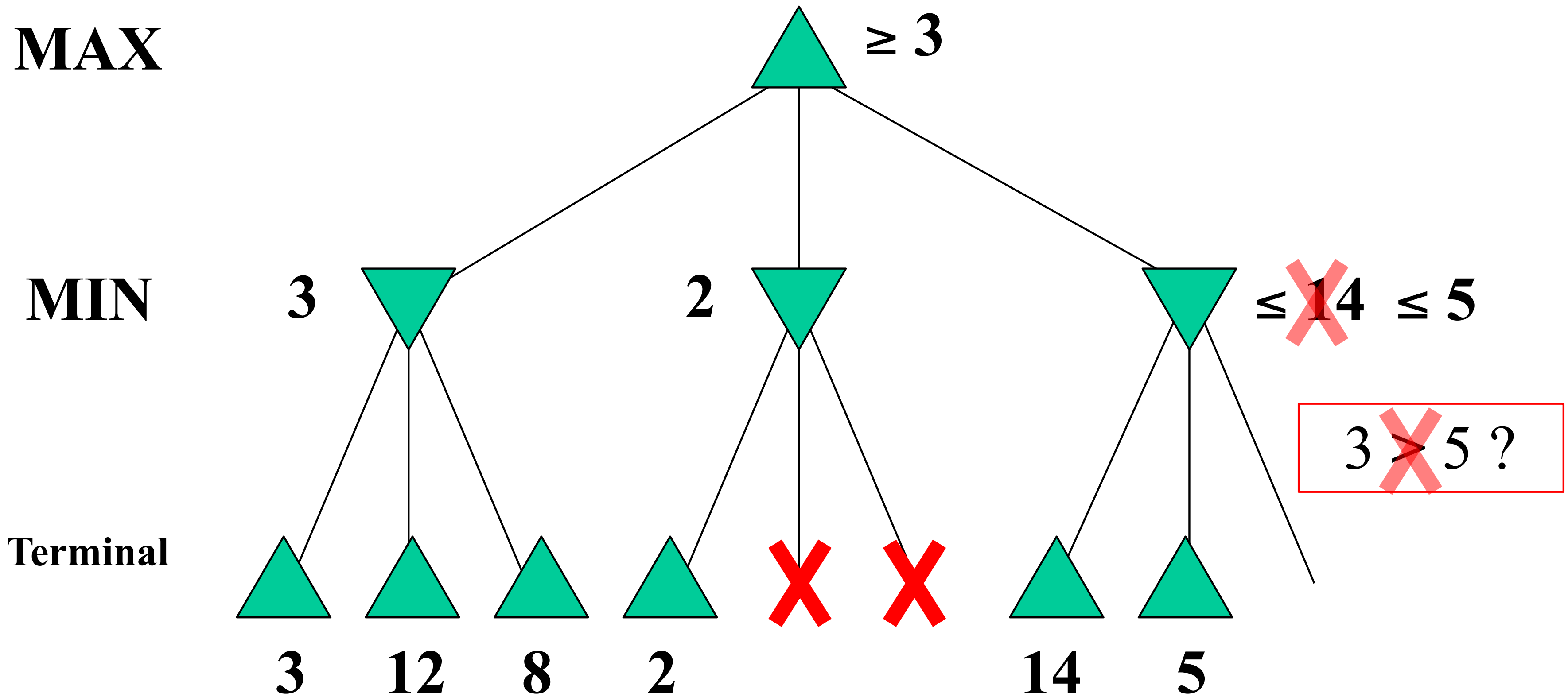
Alpha-Beta Pruning



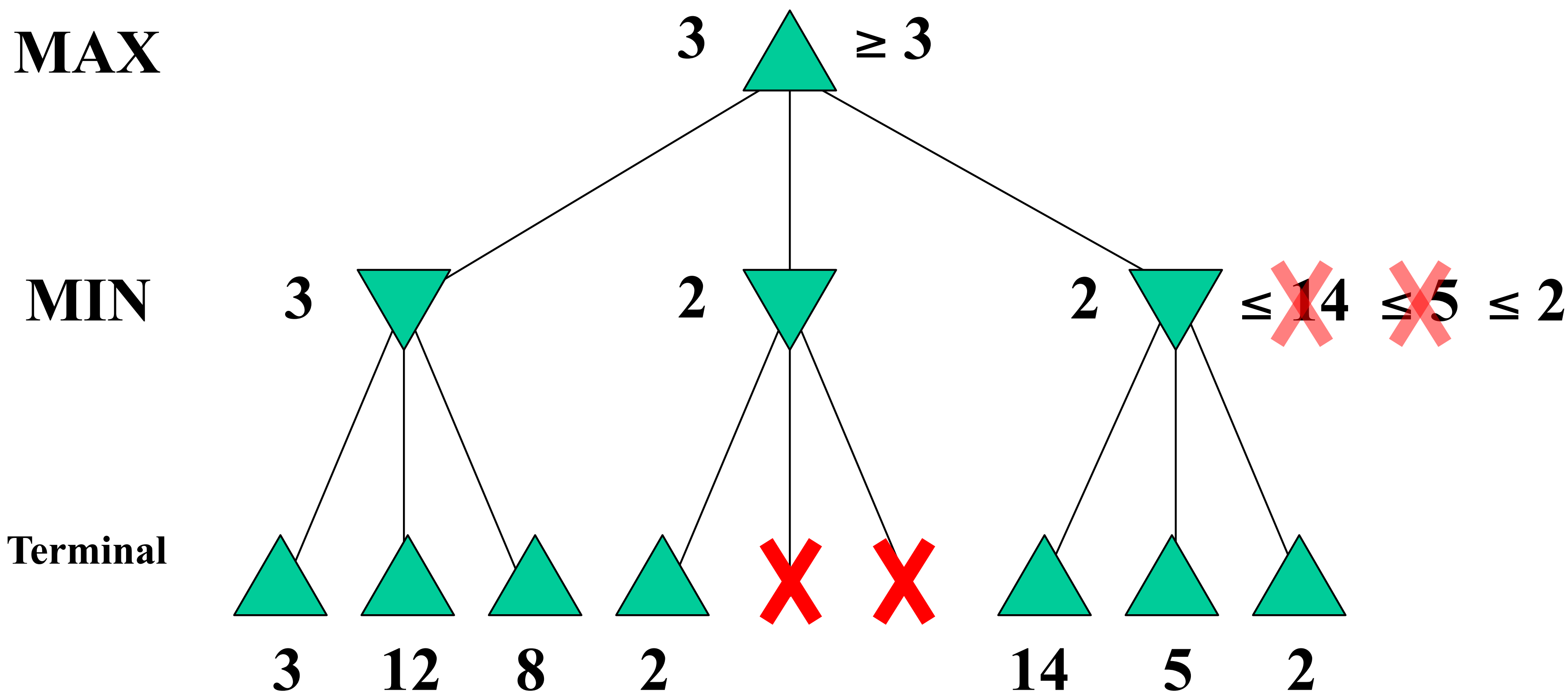
Alpha-Beta Pruning



Alpha-Beta Pruning



Alpha-Beta Pruning



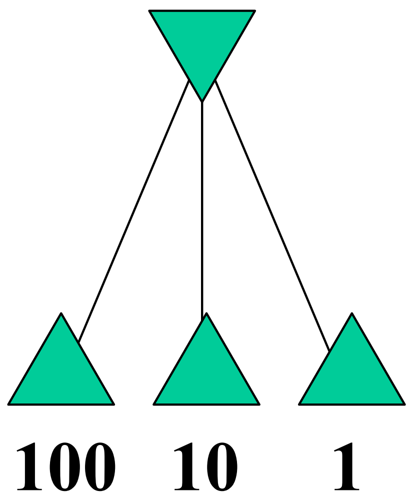
Note: Only showed MIN pruning here
In general, both MIN <u>and</u> MAX check Alpha > Beta, prune

Properties of Alpha-Beta

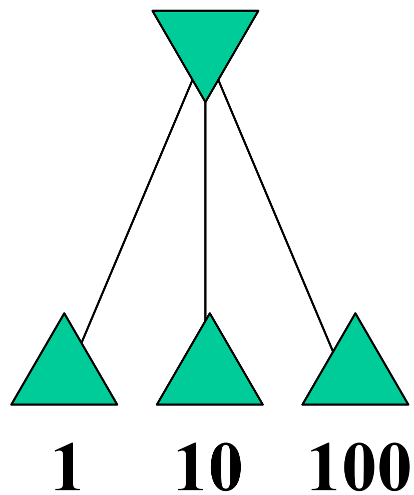
- Pruning does not affect final result
- With "perfect ordering":
  - Time complexity $O(b^{m/2})$
- A simple example of the value of
  - "reasoning about which computations are relevant"

Node Ordering

- Good move ordering would improve effectiveness of pruning
  - Try to first examine successors that are likely to be best
  - Prunes faster

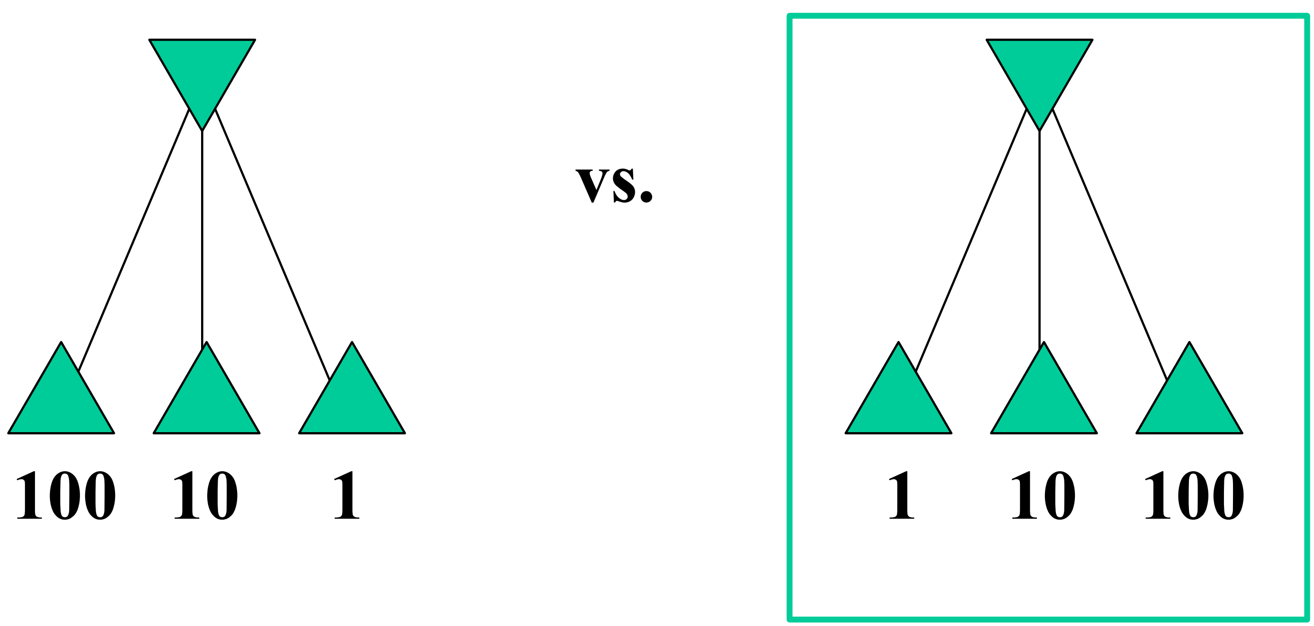


vs.



Node Ordering

- Good move ordering would improve effectiveness of pruning
  - Try to first examine successors that are likely to be best
  - Prunes faster



Tie Breaking

- What if MAX ends up with multiple choices with the same (maximum) score?
  - According to basic MiniMax, doesn’t matter which
  - But may have outside preferences to inform choice

Tie Breaking

- What if MAX ends up with multiple choices with the same (maximum) score?
  - According to basic MiniMax, doesn’t matter which
  - But may have outside preferences to inform choice
- Tie Breaking Strategies
  - Earliest Move
  - Latest Move
  - Random – Make algorithm less predictable

Tie Breaking

- What if MAX ends up with multiple choices with the same (maximum) score?
  - According to basic MiniMax, doesn’t matter which
  - But may have outside preferences to inform choice
- Tie Breaking Strategies
  - Earliest Move
  - Latest Move
  - Random – Make algorithm less predictable
- Also, consider adjusting utility function so less ties are possible

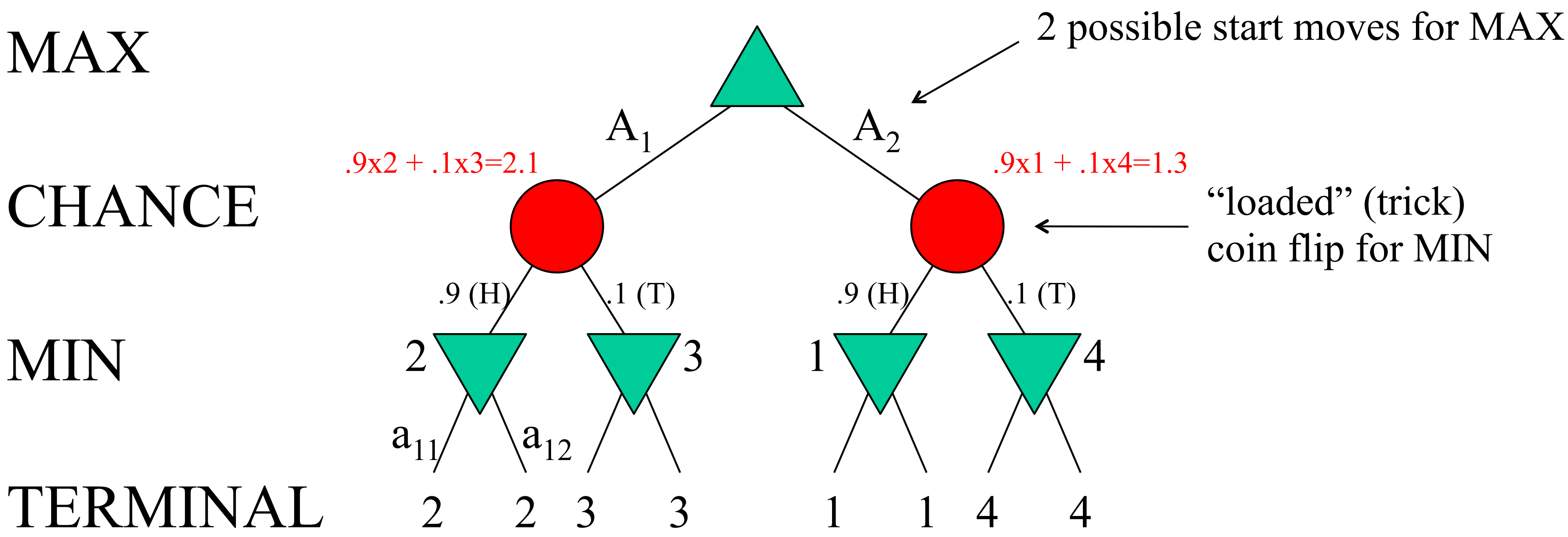
Games with Chance

- Many games have a random element
  - e.g., throwing dice to determine next move
- Cannot construct standard game tree as before
  - As in Tic-Tac-Toe
- Need to include "**CHANCE nodes**"
- Branches leading from chance node represent the possible chance-outcomes and probability
  - e.g., die rolls: each branch has the roll value (1-6) and its chance of occurring (1/6th)

Expecti-MiniMax

- TERMINAL, MAX, MIN nodes work same way as before
- CHANCE nodes are evaluated by taking weighted average of values (expected value) resulting from all possible chance outcomes (e.g., die rolls)
- Process is backed-up recursively all the way to root (as before)

Simple Example



Move A_1 is “expected” to be best for MAX

Alpha-Beta with Chance?

- Analysis for MAX and MIN nodes are same
- But can also prune CHANCE nodes

Summary

- Games can be defined as search problems
  - With complexity of real world problems
- Minimax algorithm determines the best move for a player
  - Assuming the opponent plays perfectly
  - Enumerates entire game tree
- Alpha-beta algorithm similar to minimax, but prunes away branches that are irrelevant to the final outcome
  - May need to cut off search at some point if too deep
- Can incorporate "chance"