Lecture 7

Instructor: Amit Kumar Das

Senior Lecturer,

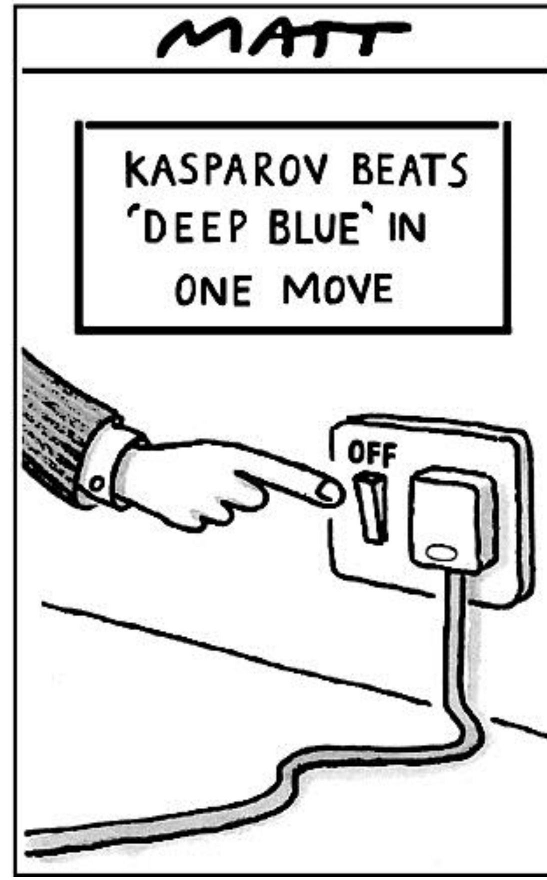
Department of Computer Science & Engineering,

East West University

Dhaka, Bangladesh.

Games and adversarial search (Chapter 5)

World Champion chess player Garry Kasparov is defeated by IBM's Deep Blue chess-playing computer in a six-game match in May, 1997 (link)



Why study games?

- Games are a traditional hallmark of intelligence
- Games are easy to formalize
- Games can be a good model of real-world competitive or cooperative activities
  - Military confrontations, negotiation, auctions, etc.

Types of game environments

	Deterministic	**Stochastic**
Perfect information (fully observable)	Chess, checkers, go	Backgammon, monopoly
Imperfect information (partially observable)	Battleships	Scrabble, poker, bridge

Alternating two-player zero-sum games

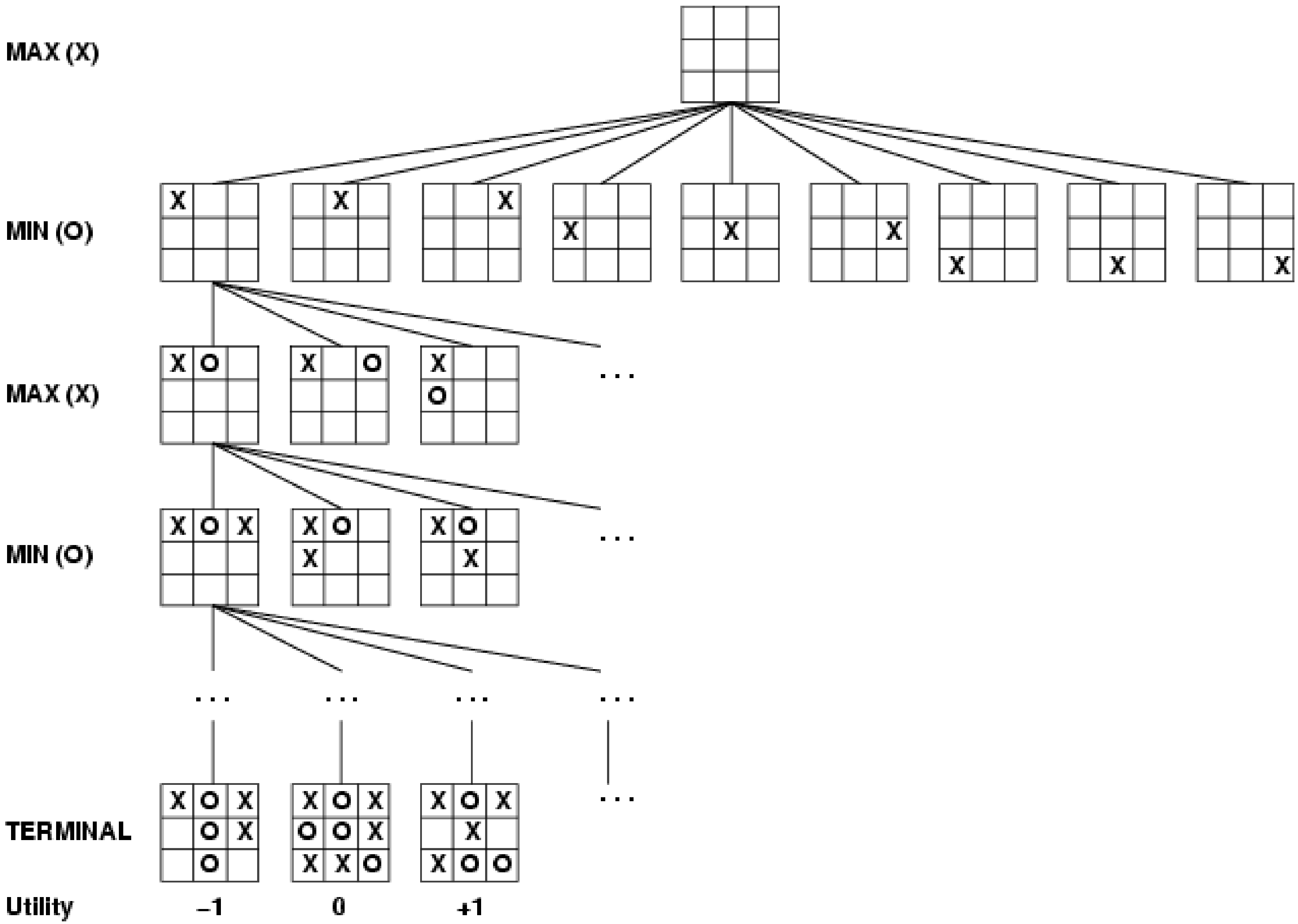
- Players take turns
- Each game outcome or **terminal state** has a **utility** for each player (e.g., 1 for win, 0 for loss)
- The sum of both players' utilities is a constant

Games vs. single-agent search

- We don’t know how the opponent will act
  - The solution is not a fixed sequence of actions from start state to goal state, but a ***strategy*** or ***policy*** (a mapping from state to best move in that state)
- Efficiency is critical to playing well
  - The time to make a move is limited
  - The branching factor, search depth, and number of terminal configurations are huge
    - In chess, branching factor ≈ 35 and depth ≈ 100, giving a search tree of 10^{154} nodes
      - Number of atoms in the observable universe ≈ 10^{80}
  - This rules out searching all the way to the end of the game

Game tree

- A game of tic-tac-toe between two players, “max” and “min”

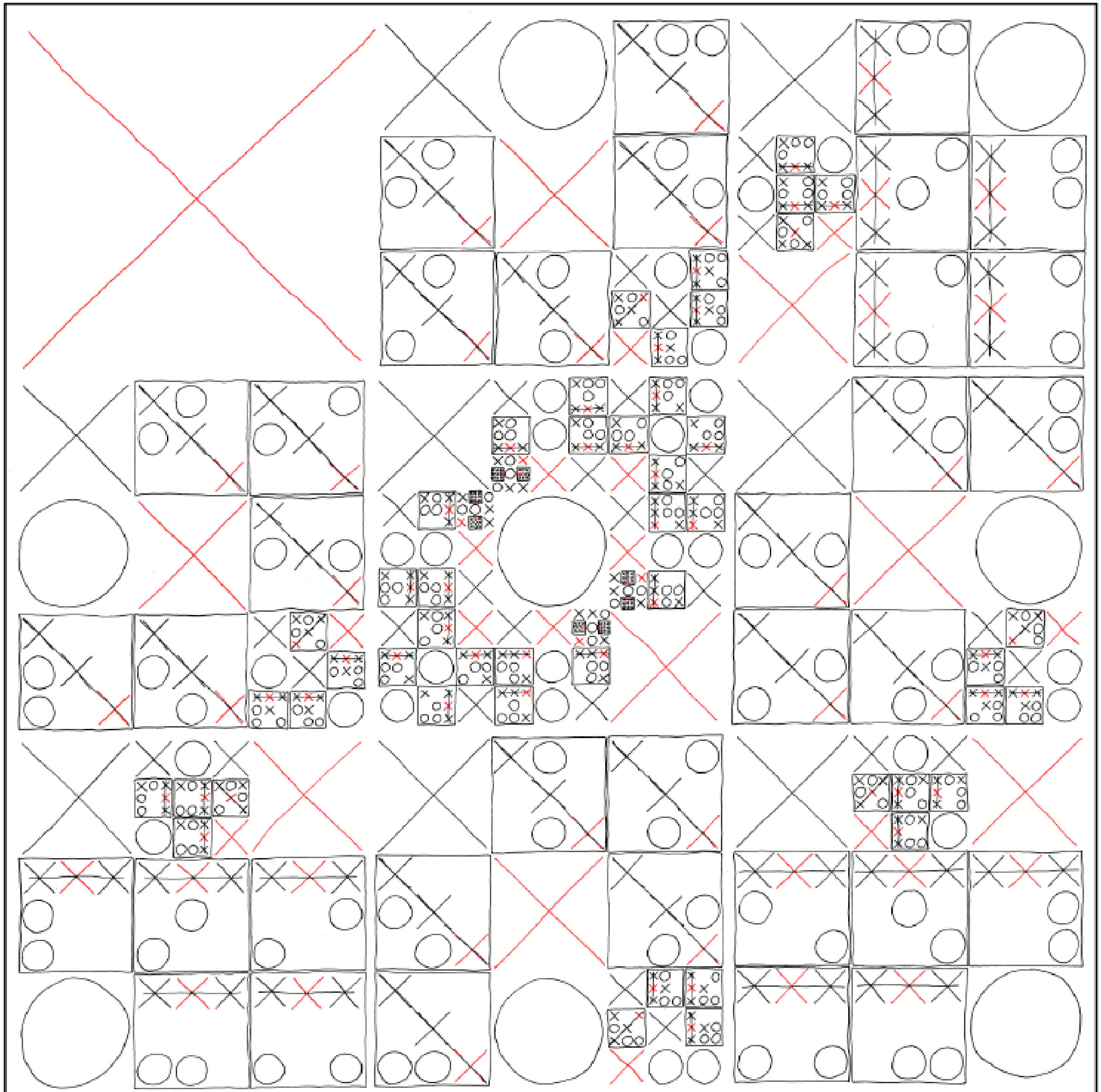


COMPLETE MAP OF OPTIMAL TIC-TAC-TOE MOVES

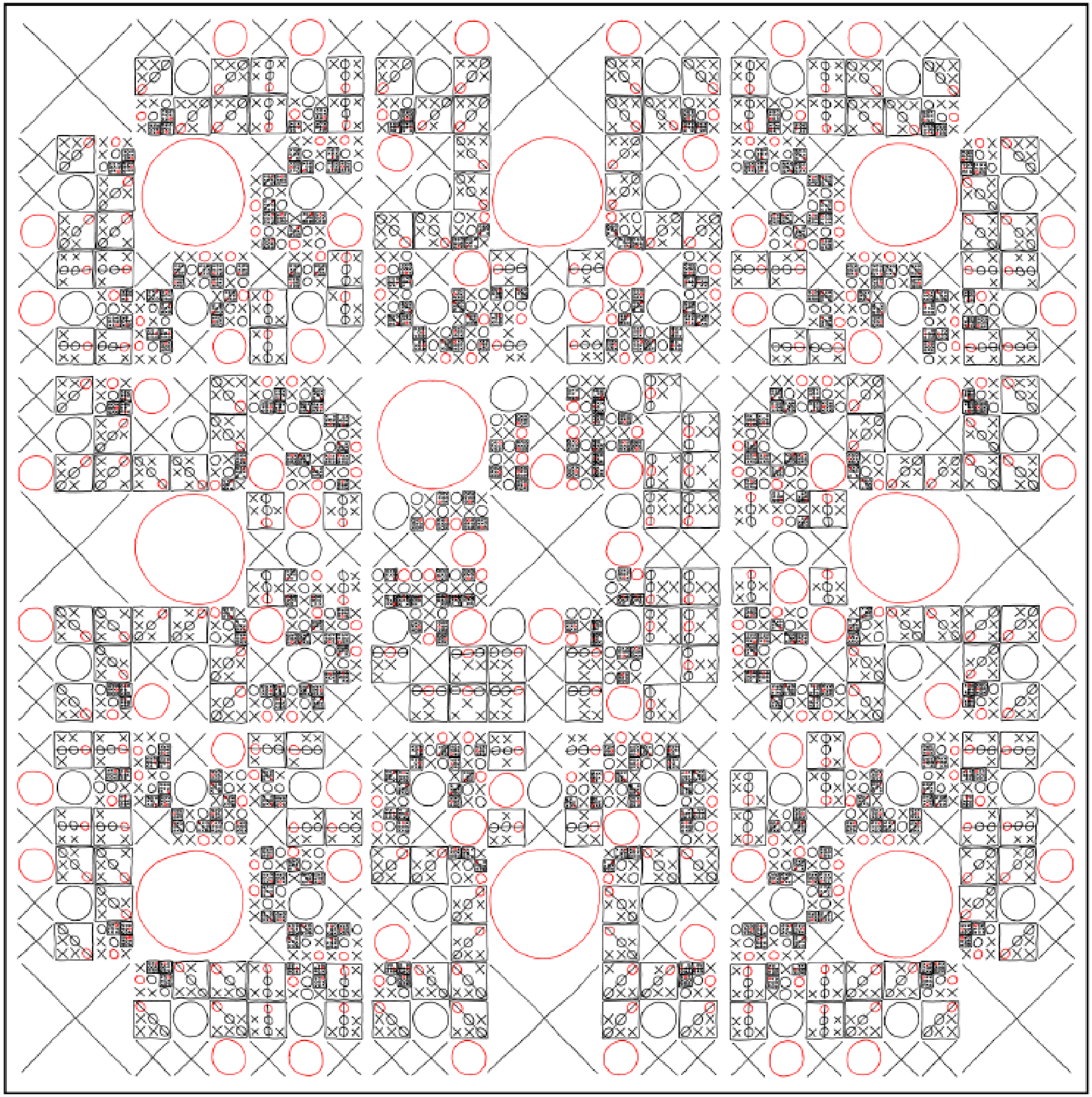
YOUR MOVE IS GIVEN BY THE POSITION OF THE LARGEST RED SYMBOL ON THE GRID. WHEN YOUR OPPONENT PICKS A MOVE, ZOOM IN ON THE REGION OF THE GRID WHERE THEY WENT. REPEAT.

http://xkcd.com/832/

MAP FOR X:

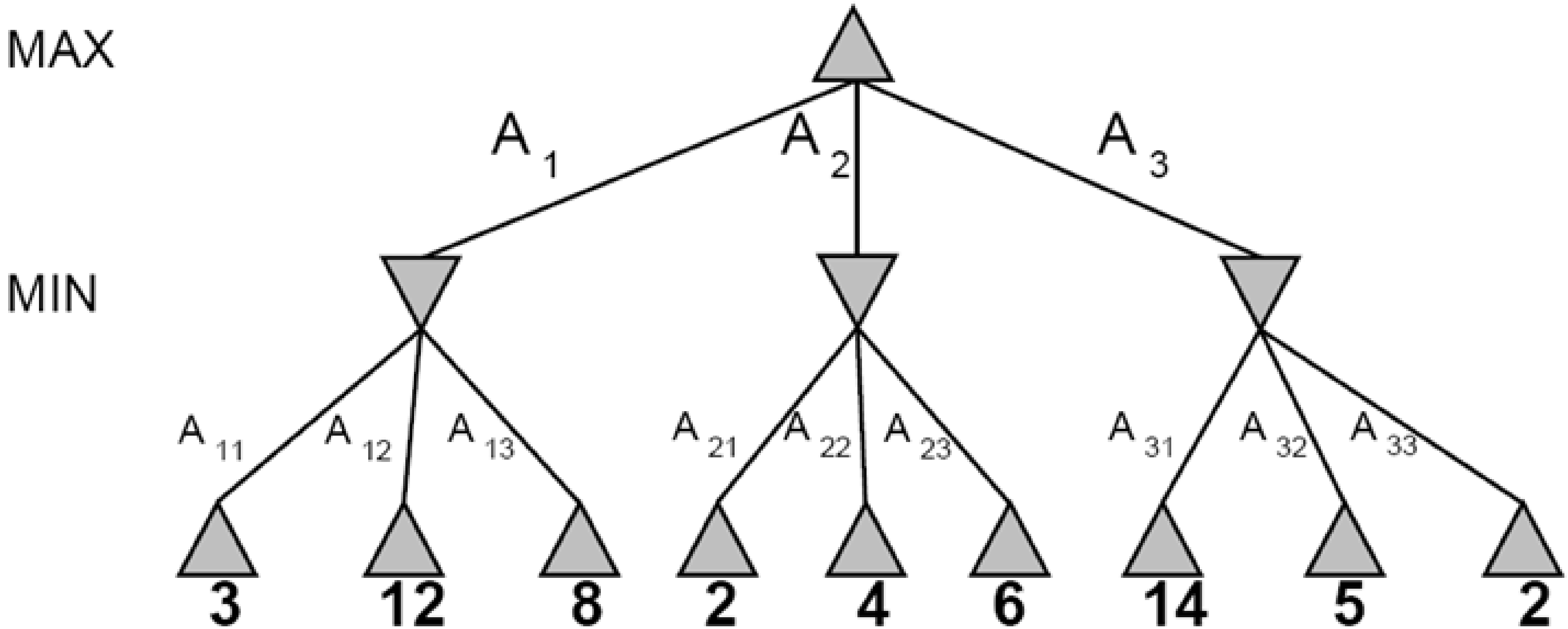


MAP FOR O:



http://xkcd.com/832/

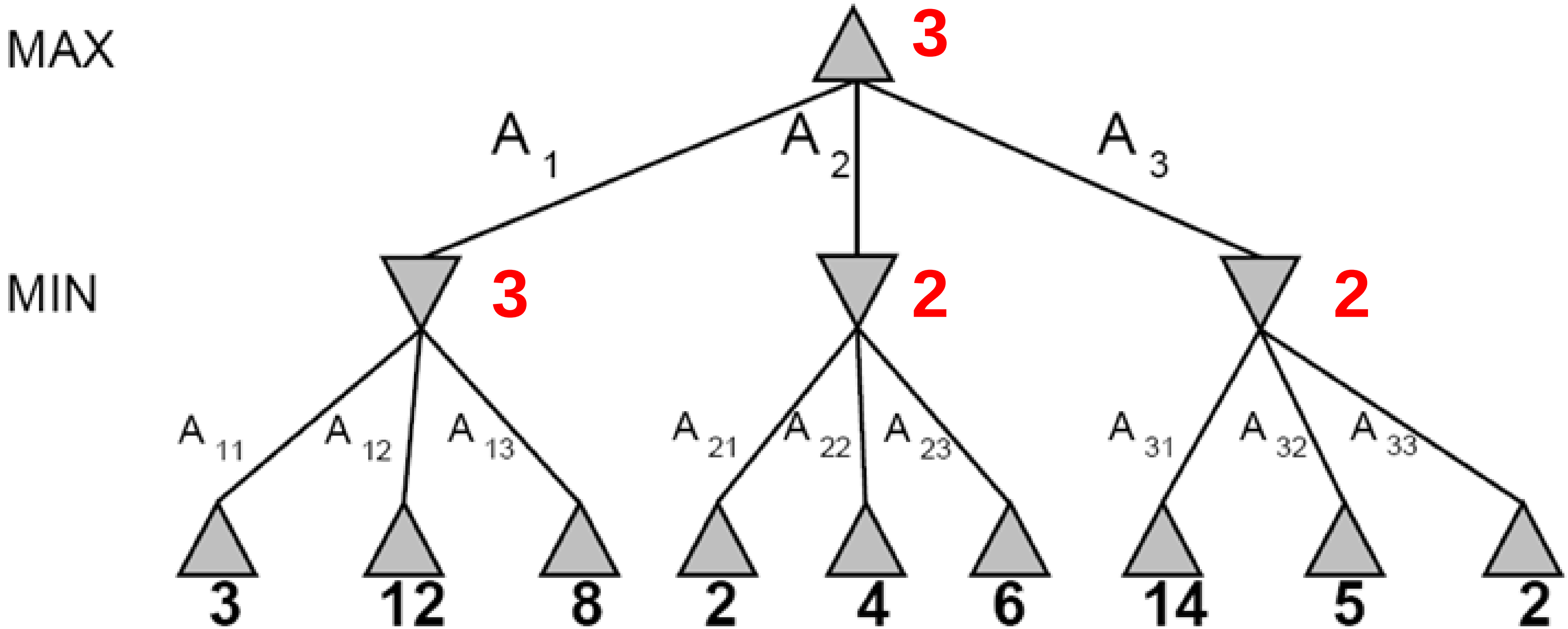
A more abstract game tree



Terminal utilities (for MAX)

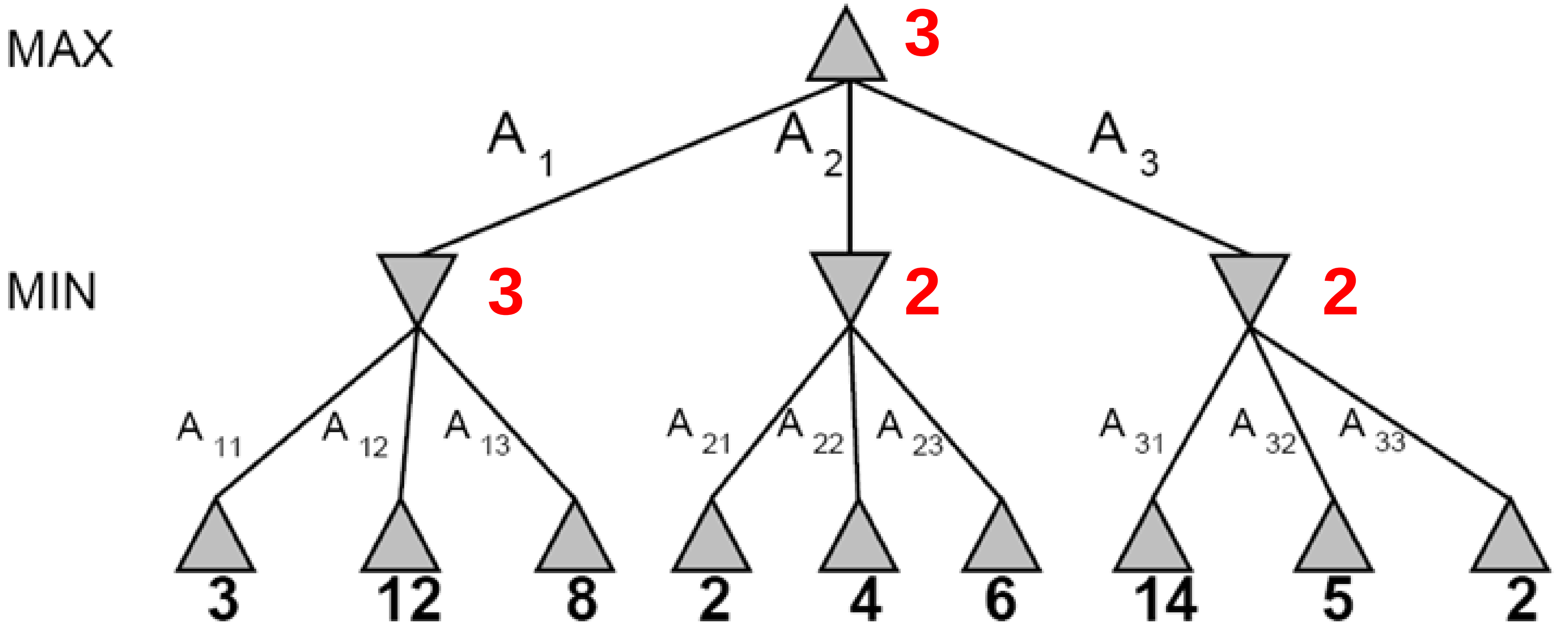
A *two-ply* game

Game tree search



- **Minimax value of a node**: the utility (for MAX) of being in the corresponding state, assuming perfect play on both sides
- **Minimax strategy:** Choose the move that gives the best worst-case payoff

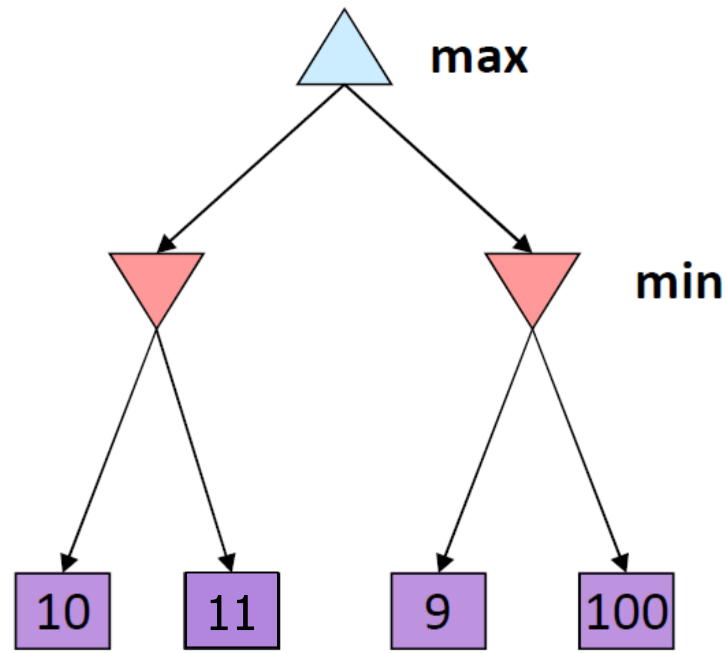
Computing the minimax value of a node



- **Minimax**(*node*) =
  - Utility(*node*) if *node* is terminal
  - $\max_{action}$ **Minimax**(Succ(*node, action*)) if *player* = MAX
  - $\min_{action}$ **Minimax**(Succ(*node, action*)) if *player* = MIN

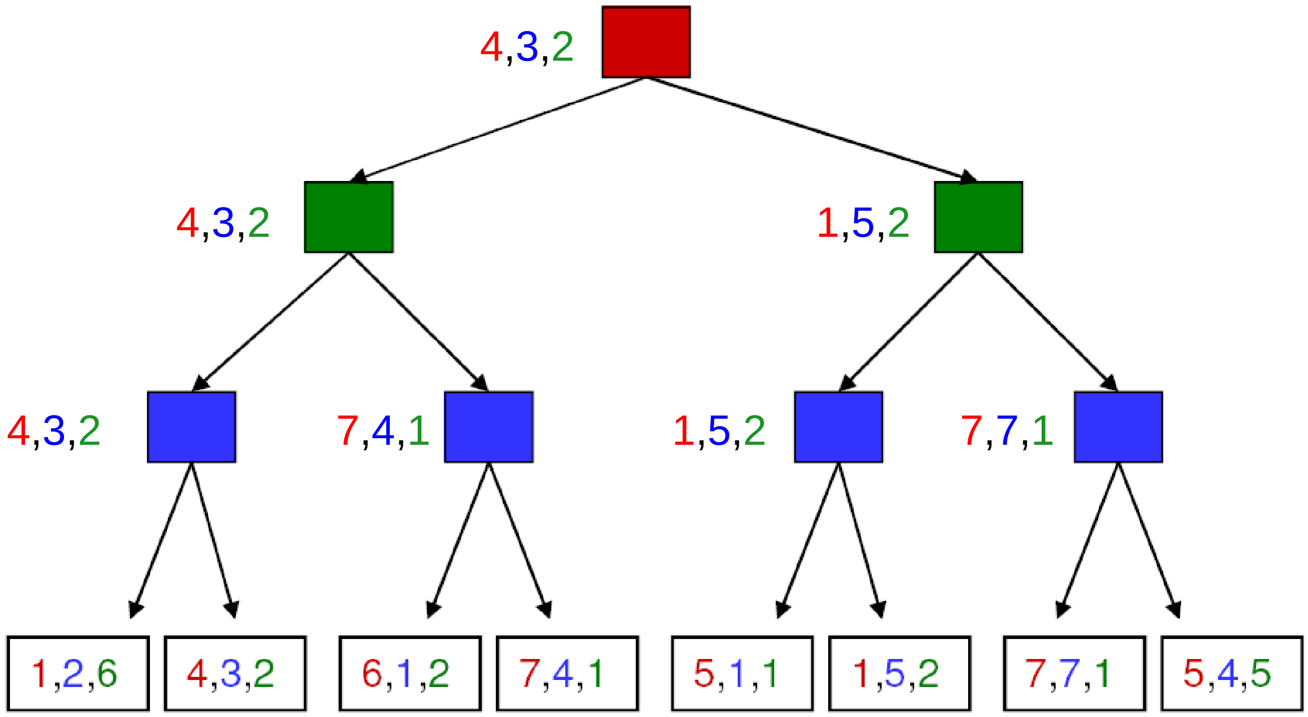
Optimality of minimax

- The minimax strategy is optimal against an optimal opponent
- What if your opponent is suboptimal?
  - Your utility can only be higher than if you were playing an optimal opponent!
  - A different strategy may work better for a sub-optimal opponent, but it will necessarily be worse against an optimal opponent



Example from D. Klein and P. Abbeel

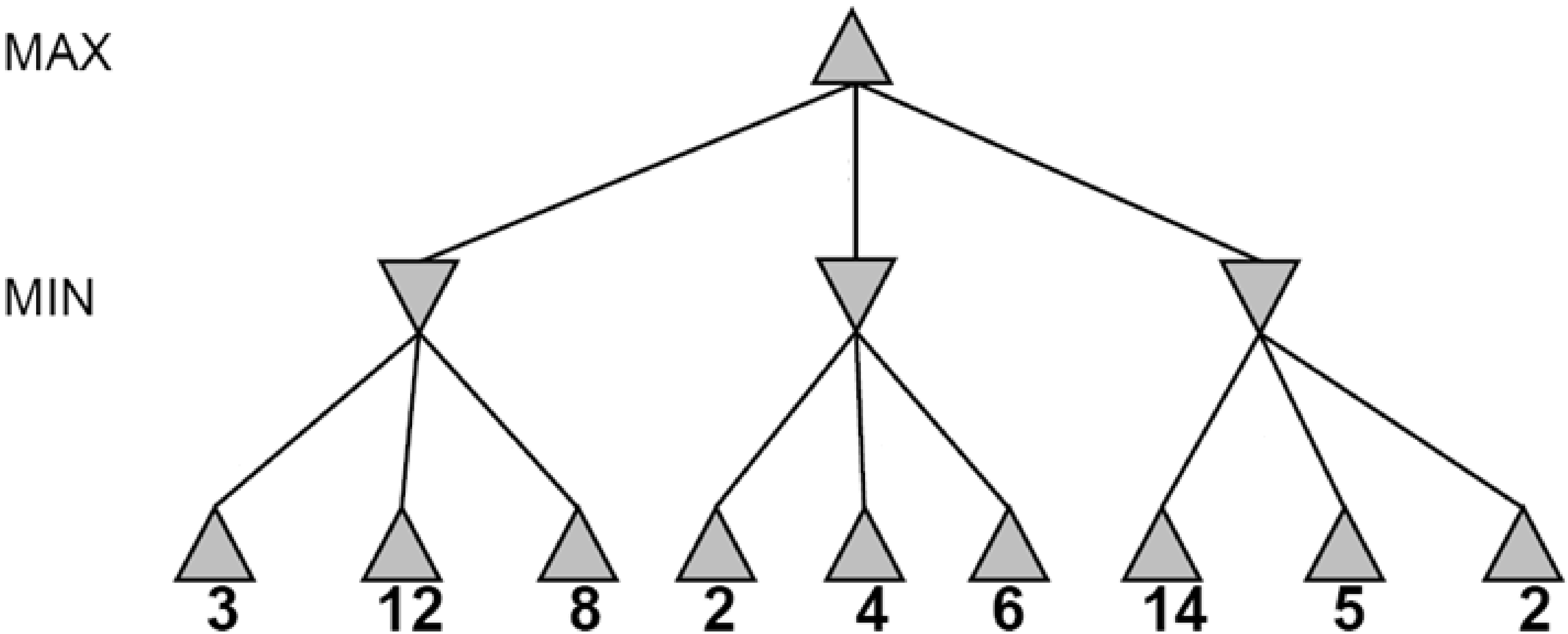
More general games



- More than two players, non-zero-sum
- Utilities are now tuples
- Each player maximizes their own utility at their node
- Utilities get propagated (*backed up*) from children to parents

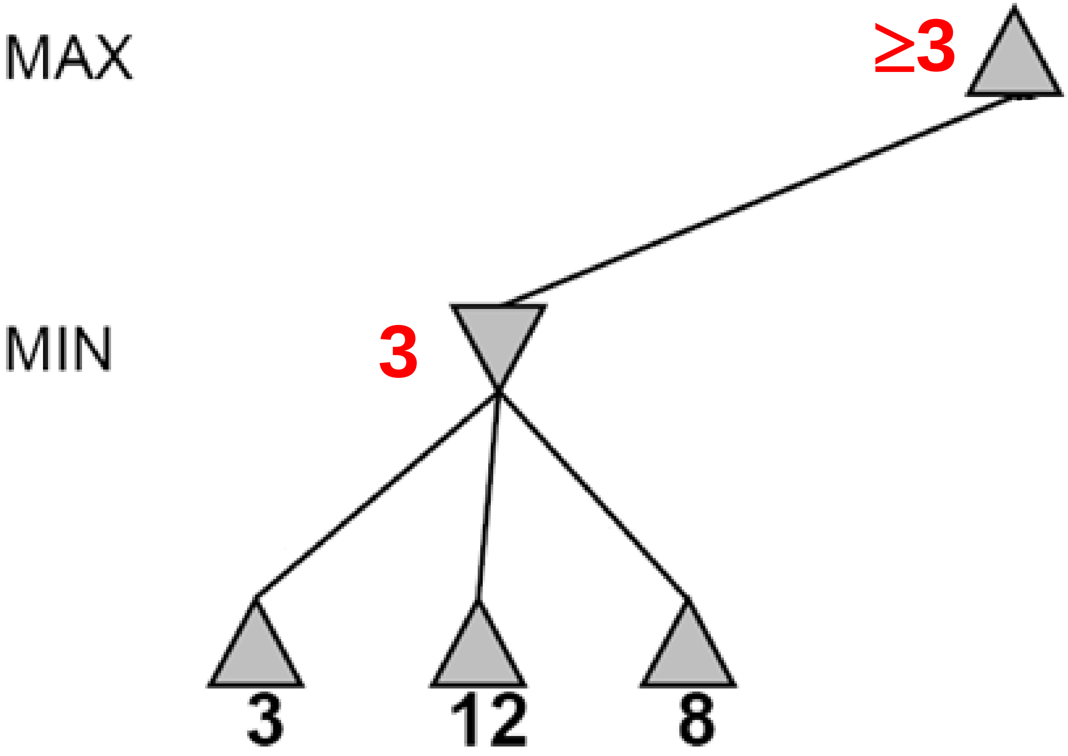
Alpha-beta pruning

- It is possible to compute the exact minimax decision without expanding every node in the game tree



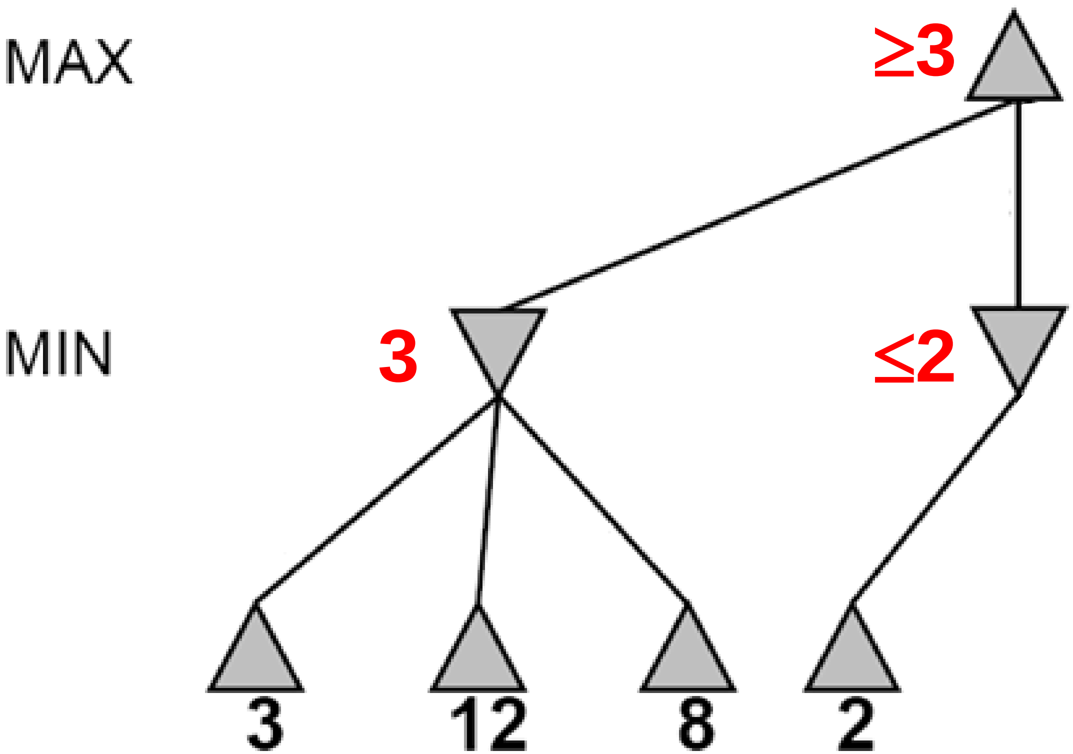
Alpha-beta pruning

- It is possible to compute the exact minimax decision without expanding every node in the game tree



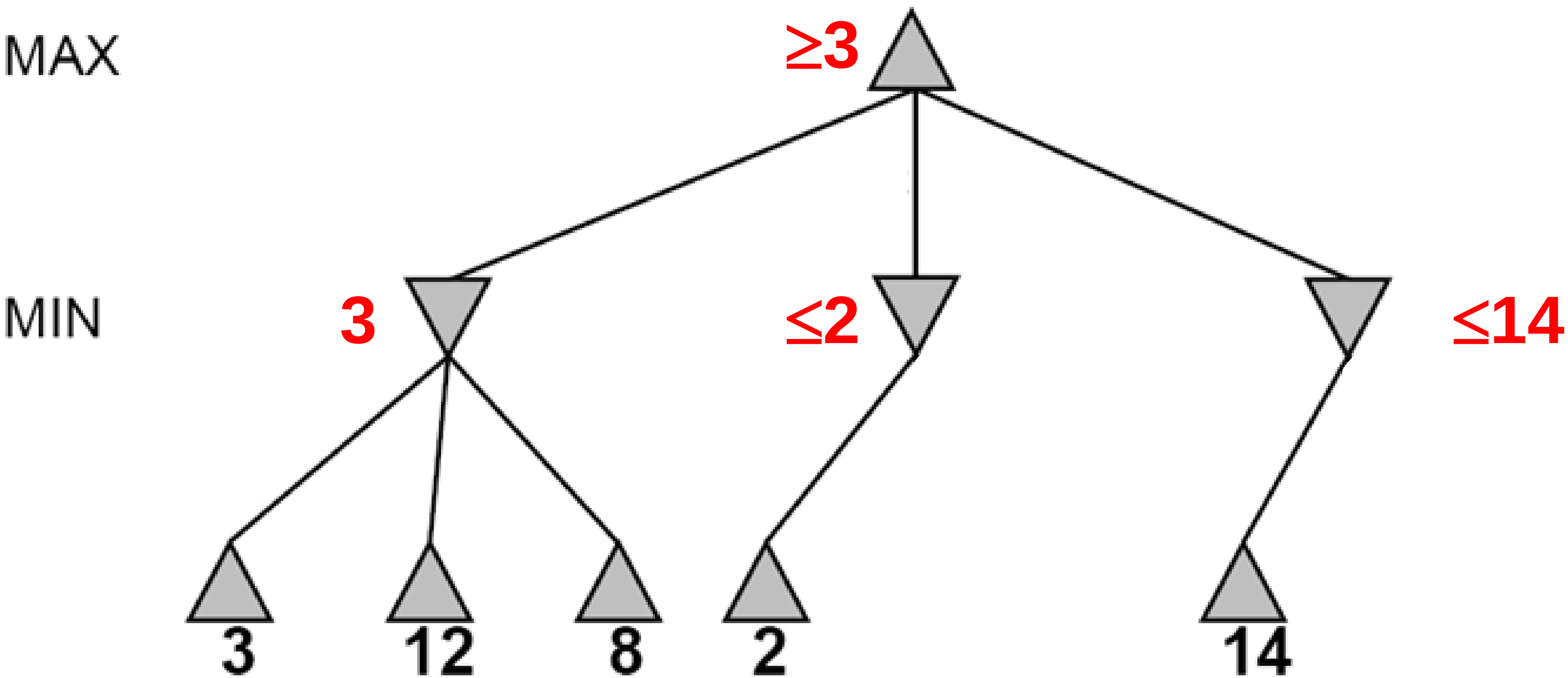
Alpha-beta pruning

- It is possible to compute the exact minimax decision without expanding every node in the game tree



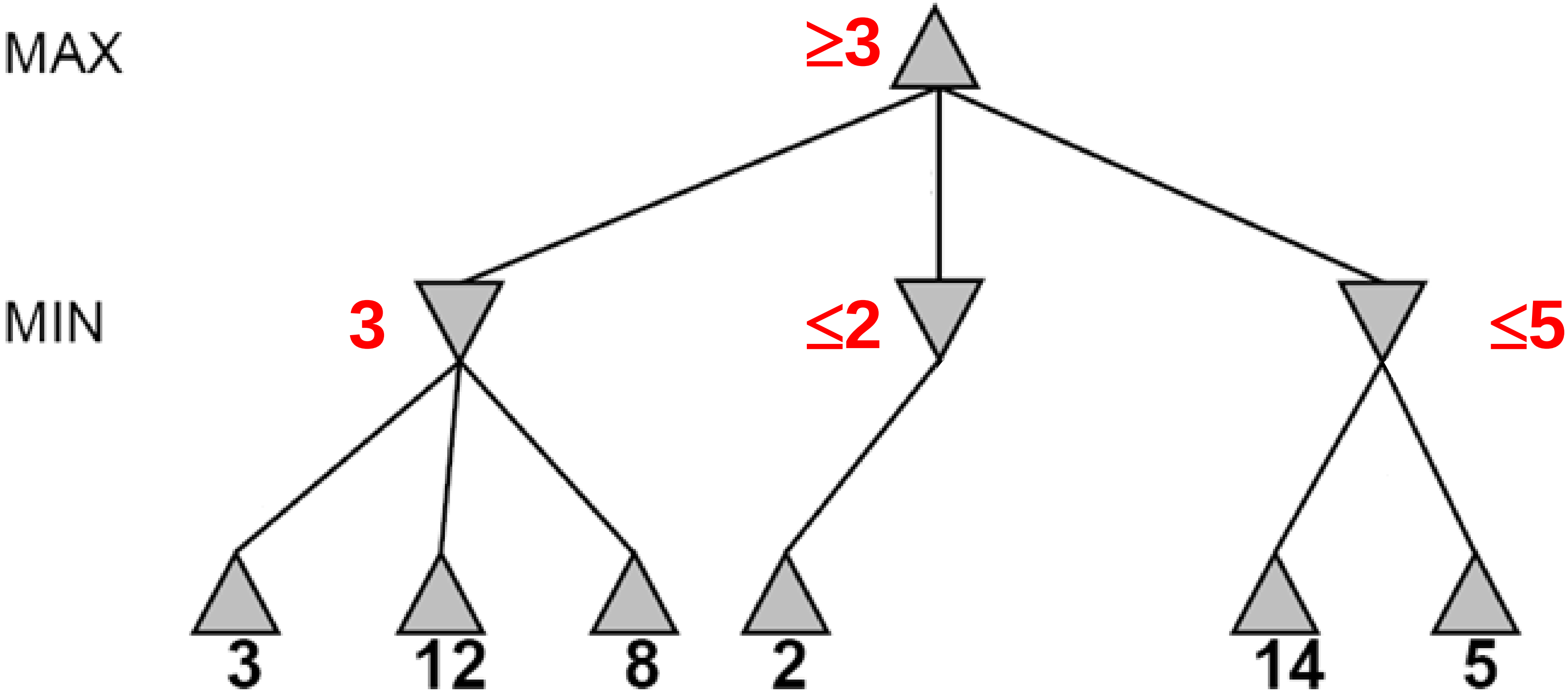
Alpha-beta pruning

- It is possible to compute the exact minimax decision without expanding every node in the game tree



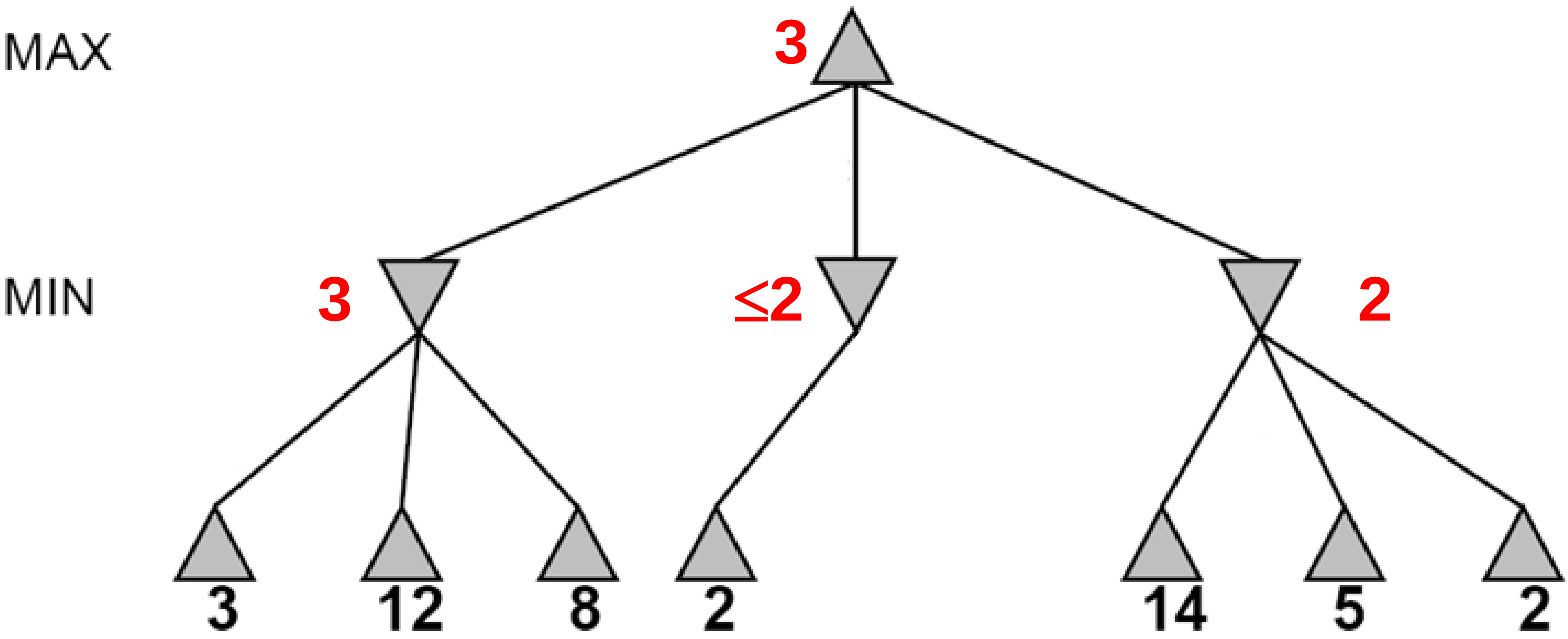
Alpha-beta pruning

- It is possible to compute the exact minimax decision without expanding every node in the game tree



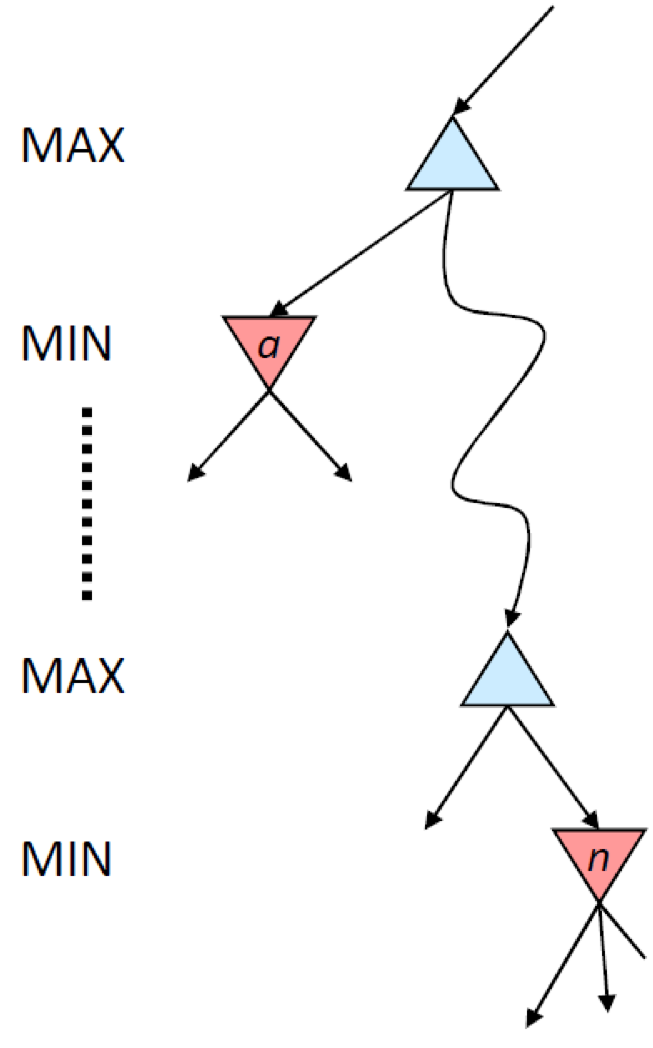
Alpha-beta pruning

- It is possible to compute the exact minimax decision without expanding every node in the game tree



Alpha-beta pruning

- **α** is the value of the best choice for the MAX player found so far at any choice point above node *n*
- We want to compute the MIN-value at *n*
- As we loop over *n*'s children, the MIN-value decreases
- If it drops below **α**, MAX will never choose *n*, so we can ignore *n*'s remaining children
- Analogously, **β** is the value of the lowest-utility choice found so far for the MIN player



Alpha-beta pruning

Function *action* = **Alpha-Beta-Search**(*node*)

$\quad v$ = **Min-Value**(*node*, $-\infty$, ∞)

$\quad$ return the *action* from *node* with value v

α: best alternative available to the Max player

β: best alternative available to the Min player

Function v = **Min-Value**(*node*, α, β)

$\quad$ if Terminal(*node*) return Utility(*node*)

$\quad v = +\infty$

$\quad$ for each *action* from *node*

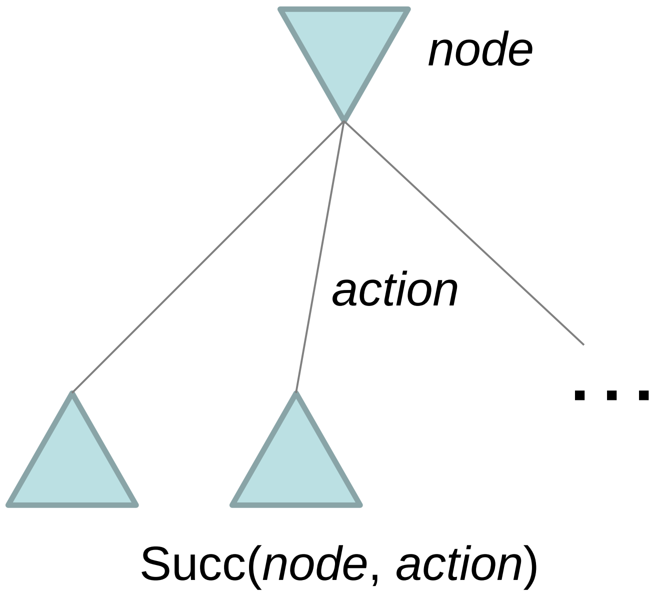
$\quad\quad v$ = Min(v, **Max-Value**(Succ(*node*, *action*), α, β))

$\quad\quad$ if $v \leq \alpha$ return v

$\quad\quad \beta$ = Min(β, v)

$\quad$ end for

$\quad$ return v



Alpha-beta pruning

Function *action* = **Alpha-Beta-Search**(*node*)

    v = **Max-Value**(*node*, $-\infty$, ∞)

    return the *action* from *node* with value v

α: best alternative available to the Max player

β: best alternative available to the Min player

Function v = **Max-Value**(*node*, α, β)

    if Terminal(*node*) return Utility(*node*)

    $v = -\infty$

    for each *action* from *node*

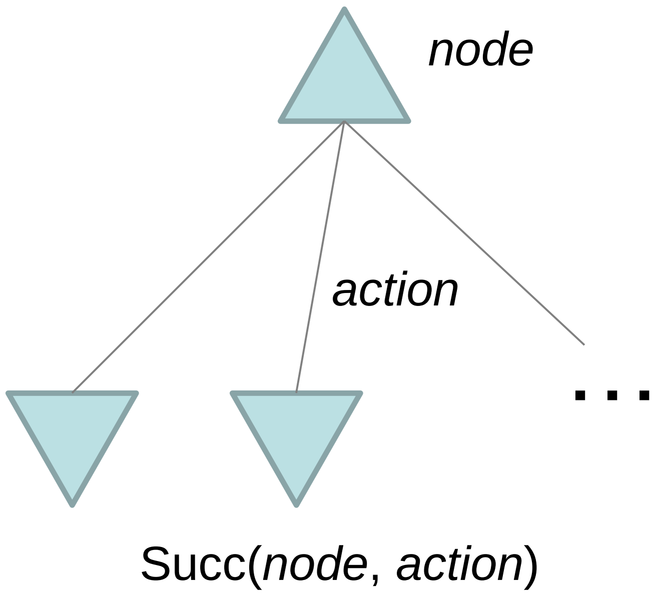
        v = Max(v, **Min-Value**(Succ(*node*, *action*), α, β))

        if $v \geq \beta$ return v

        α = Max(α, v)

    end for

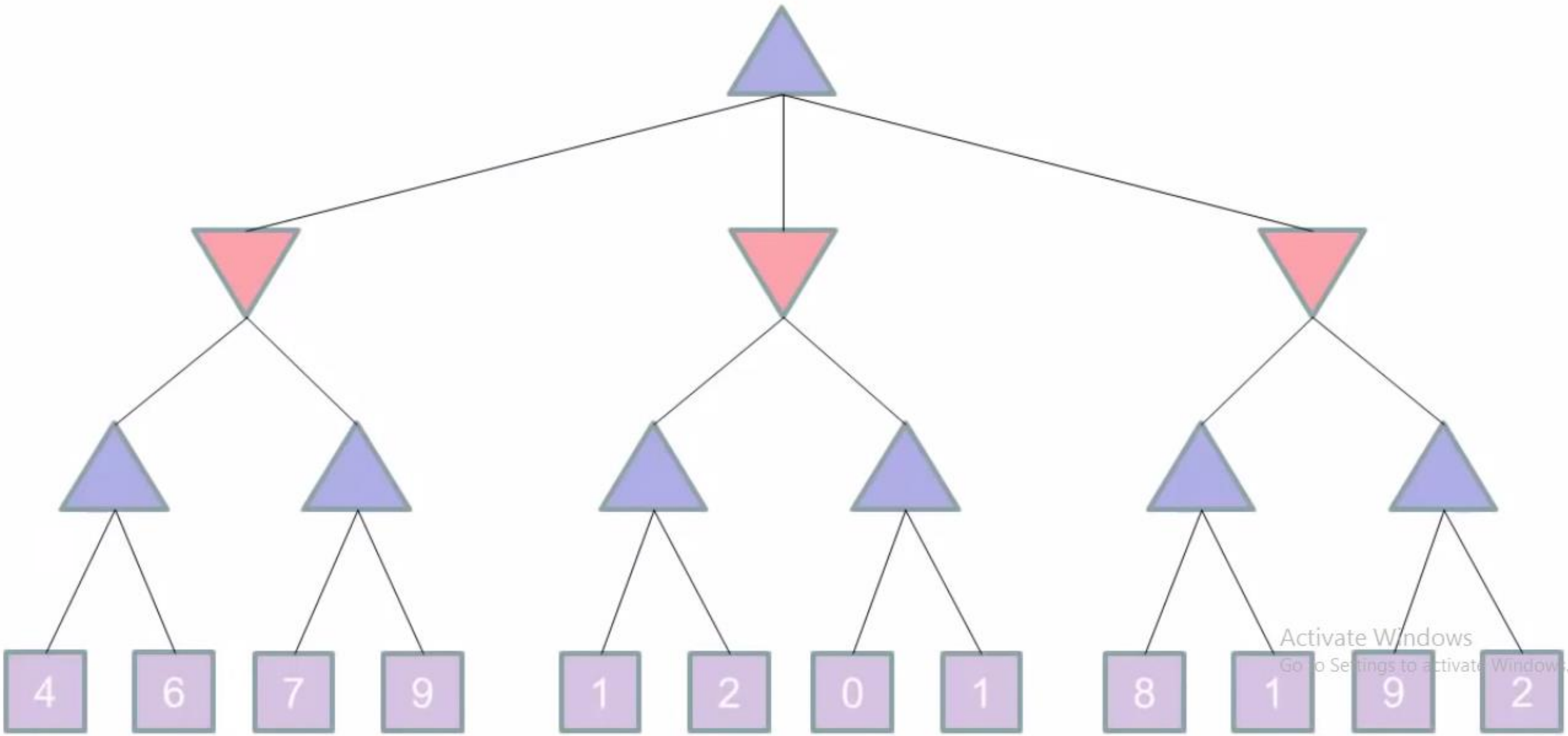
    return v



Alpha-beta pruning

- Pruning does not affect final result
- Amount of pruning depends on move ordering
  - Should start with the “best” moves (highest-value for MAX or lowest-value for MIN)
  - For chess, can try captures first, then threats, then forward moves, then backward moves
  - Can also try to remember “killer moves” from other branches of the tree
- With perfect ordering, the time to find the best move is reduced to $O(b^{m/2})$ from $O(b^m)$
  - Depth of search is effectively doubled

Alpha-Beta Example



Thank You