CMSC 474, Introduction to Game Theory

28. Game-tree Search and Pruning Algorithms

Mohammad T. Hajiaghayi
University of Maryland

Finite perfect-information zero-sum games

- **Finite**:
  - finitely many agents, actions, states, histories
- **Perfect information**:
  - Every agent knows
    - all of the players' utility functions
    - all of the players' actions and what they do
    - the history and current state
  - No simultaneous actions – agents move one-at-a-time
- **Constant sum** (or **zero-sum**):
  - Constant k such that regardless of how the game ends,
    - $\Sigma_{i=1,\ldots,n}\, u_i = k$
  - For every such game, there's an equivalent game in which $k = 0$

Examples

- **Deterministic**:
  - chess, checkers
  - go, gomoku
  - reversi (othello)
  - tic-tac-toe, qubic, connect-four
  - mancala (awari, kalah)
  - 9 men's morris (merelles, morels, mill)
- **Stochastic**:
  - backgammon, monopoly, yahtzee, parcheesi, roulette, craps
- For now, we'll consider just the deterministic games

Outline

- A brief history of work on this topic
- Restatement of the Minimax Theorem
- Game trees
- The minimax algorithm
- α-β pruning
- Resource limits, approximate evaluation

- Most of this isn't in the game-theory book
- For further information, look at the following
  - Russell & Norvig's *Artificial Intelligence: A Modern Approach*
    - There are 3 editions of this book
    - In the 2nd edition, it's Chapter 6

Brief History

1846 (Babbage)	designed machine to play tic-tac-toe
1928 (von Neumann)	minimax theorem
1944 (von Neumann & Morgenstern)	backward induction
1950 (Shannon)	minimax algorithm (finite-horizon search)
1951 (Turing)	program (on paper) for playing chess
1952–7 (Samuel)	checkers program capable of beating its creator
1956 (McCarthy)	pruning to allow deeper minimax search
1957 (Bernstein)	first complete chess program, on IBM 704 vacuum-tube computer could examine about 350 positions/minute
1967 (Greenblatt)	first program to compete in human chess tournaments 3 wins, 3 draws, 12 losses
1992 (Schaeffer)	Chinook won the 1992 US Open checkers tournament
1994 (Schaeffer)	Chinook became world checkers champion; Tinsley (human champion) withdrew for health reasons
1997 (Hsu et al)	Deep Blue won 6-game match vs world chess champion Kasparov
2007 (Schaeffer et al)	Checkers solved: with perfect play, it’s a draw 10^{14} calculations over 18 years

Restatement of the Minimax Theorem

- Suppose agents 1 and 2 use strategies s and t on a 2-person game G
  - Let $u(s,t) = u_1(s,t) = -u_2(s,t)$
  - Call the agents *Max* and *Min* (they want to maximize and minimize u)

Minimax Theorem: If G is a two-person finite zero-sum game, then there are strategies s^* and t^*, and a number v called G's *minimax value*, such that

- If Min uses t^*, Max's expected utility is $\le v$, i.e., $\max_s u(s,t^*) = v$
- If Max uses s^*, Min's expected utility is $\ge v$, i.e., $\min_t u(s^*,t) = v$

Corollary 1:

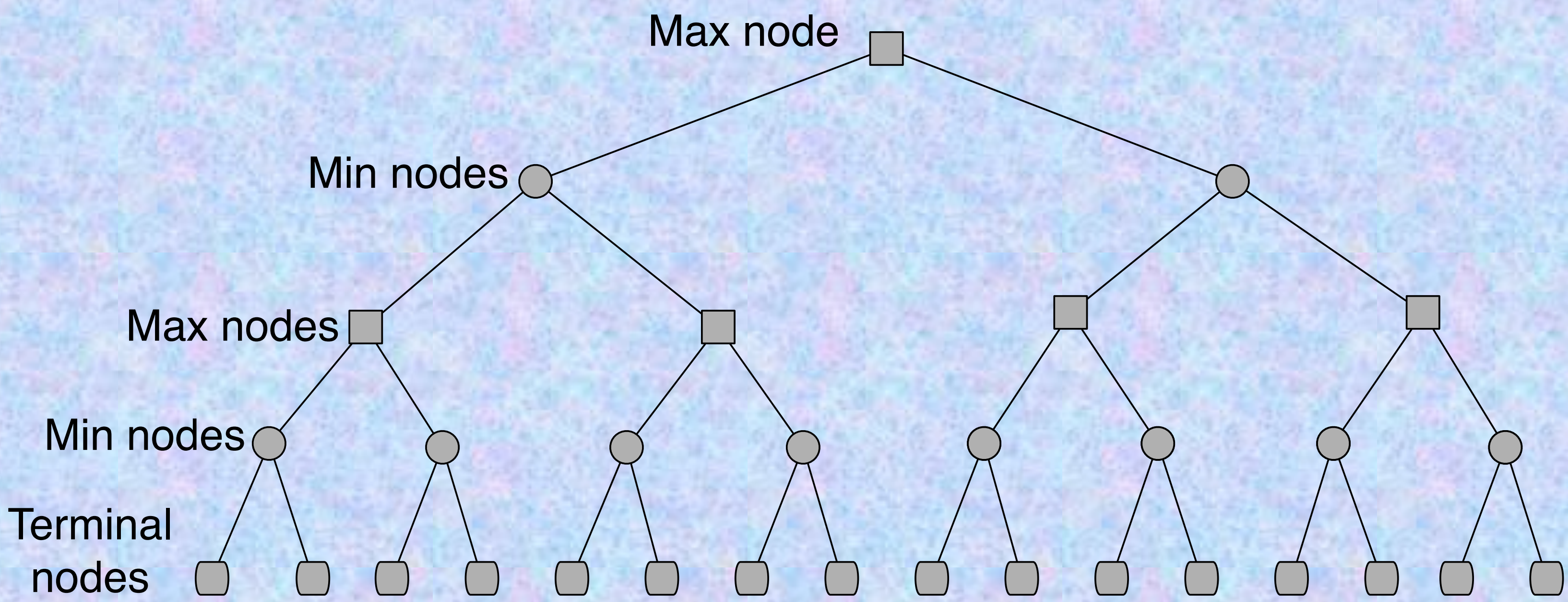
- $u(s^*,t^*) = v$
- (s^*,t^*) is a Nash equilibrium
- s^* (or t^*) is Max's (or Min's) minimax strategy and maximin strategy

s^* (or t^*) is also called "perfect play" for Max (or Min)

Corollary 2: If G is a perfect-information game, then there are subgame-perfect pure strategies s^* and t^* that satisfy the theorem.

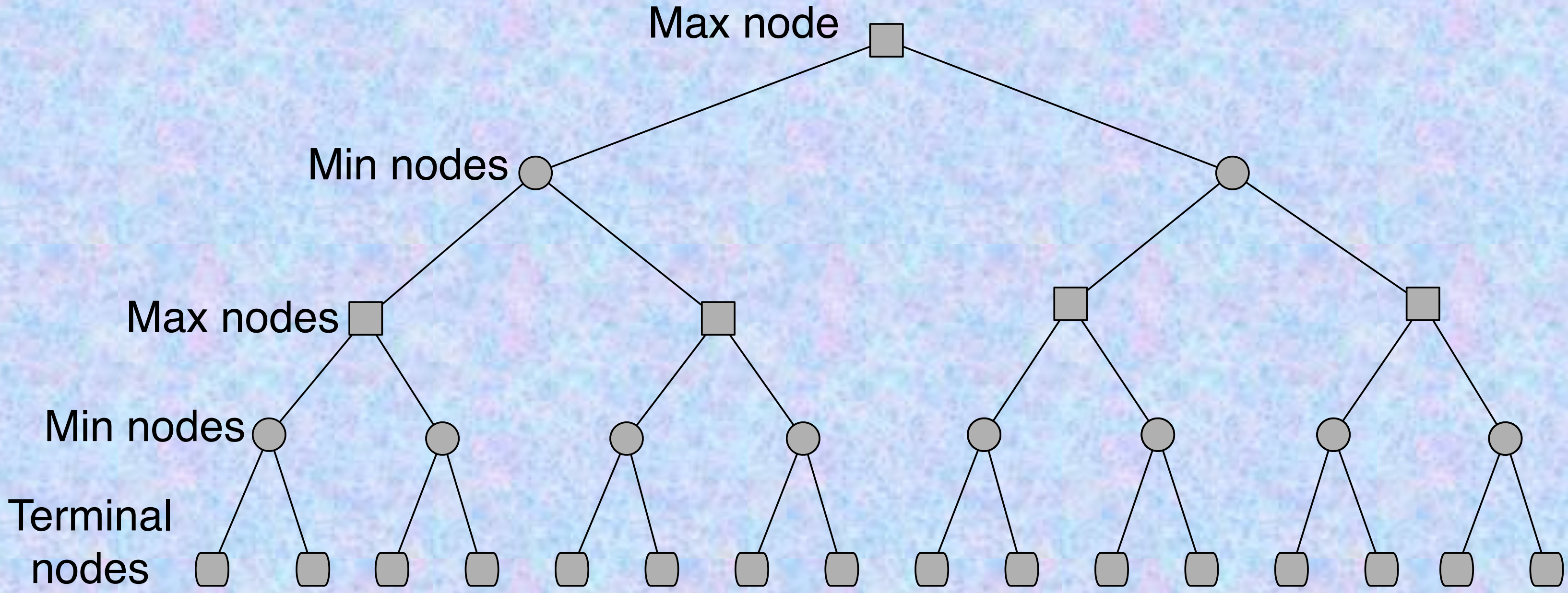
Game Tree Terminology

- **Root** node: where the game starts
- **Max** (or **Min**) node: a node where it's Max's (or Min's) move
  - Usually draw Max nodes as squares, Min nodes as circles
- A node's **children**: the possible "next nodes"
- **Terminal** node: a node where the game ends



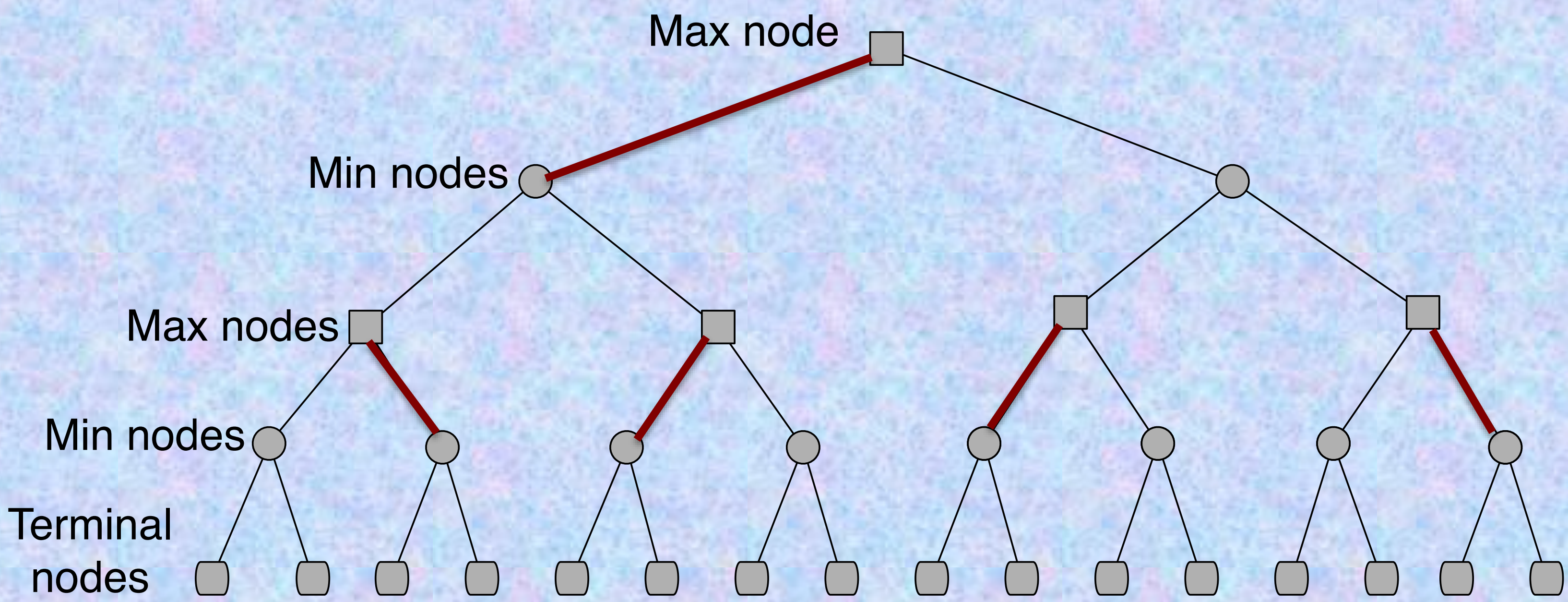
Number of Nodes

- Let b = maximum branching factor
- Let h = height of tree (maximum depth of any terminal node)
- If h is even and the root is a Max node, then
  - The number of Max nodes is $1 + b^2 + b^4 + \ldots + b^{h-2} = O(b^h)$
  - The number of Min nodes is $b + b^3 + b^5 + \ldots + b^{h-1} = O(b^h)$
- What if h is odd?



Number of Pure Strategies

- Pure strategy for Max: at every Max node, choose one branch
- $O(b^h)$ Max nodes, b choices at each of them => $O({}_{b}b^{h})$ pure strategies
  - ➢ In the following tree, how many pure strategies for Max?
- What about Min?



Finding the Minimax Strategy

- Brute-force way to find minimax strategy for Max
  - Construct the sets S and T of all distinct strategies for Max and Min, then choose

$$s^* = \arg\min_{s \in S} \max_{t \in T} u(s,t)$$

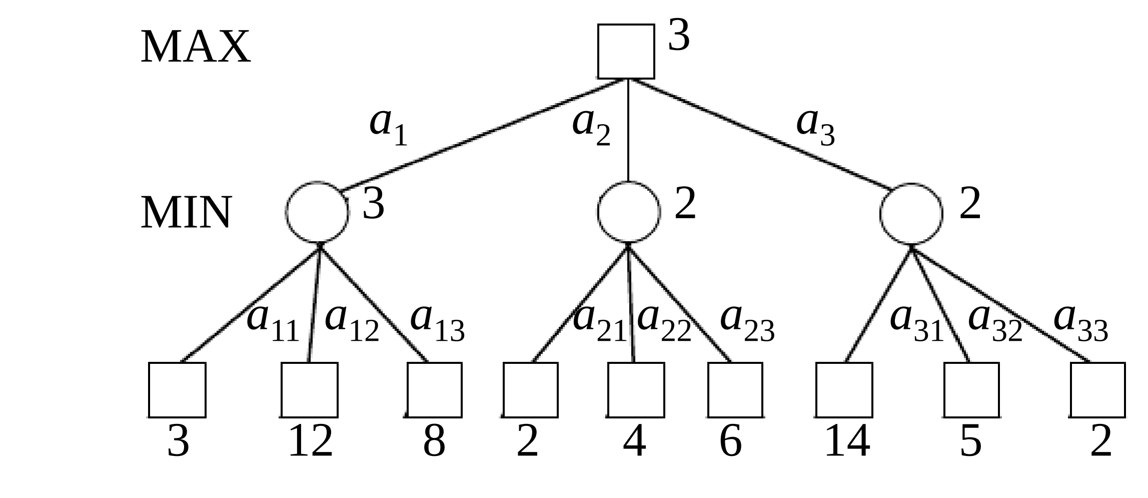
- Complexity analysis:
  - Need to construct and store $O(b^{b^h})$ distinct strategies
  - Each distinct strategy has size $O(b^h)$
  - Thus space complexity is $O(b(b^h+h)) = b^{O(b^h)}$
  - Time complexity is slightly worse
- But there's an easier way to find the minimax strategy

- Notation: $v(x)$ = minimax value of the tree rooted at x
  - If x is terminal then $v(x) = u_{\text{Max}}(x)$

Backward Induction

- Depth-first implementation of backward induction (Chapter 4)
  - Returns $v(x)$
  - Can easily modify it to return both $v(x)$ and strategy a

function Backward-Induction(x)
  if x is terminal then return $v(x)$
  else if it is Max's move at x then
    return max{Backward-Induction($\sigma(x,a)$) : $a \in \chi(x)$}
  else return min{Backward-Induction($\sigma(x,a)$) : $a \in \chi(x)$}



Complexity Analysis

- Space complexity
  = O(maximum path length)•(space needed to store the path)
  = $O(bh)$
- Time complexity = size of the game tree = $O(b^h)$
  - where b = branching factor, h = height of the game tree
- This is a lot better than $b^{O(b^h)}$
- But it still isn't good enough for games like chess
  - $b \approx 35$, $h \approx 100$ for "reasonable" chess games
  - $b^h \approx 35^{100} \approx 10^{135}$ nodes

- Number of particles in the universe $\approx 10^{87}$
  - 10^{135} nodes is $\approx 10^{55}$ times the number of particles in the universe
    $\Rightarrow$ no way to examine every node!

Minimax Algorithm (Shannon,1950)

function Minimax(x,d)

if x is terminal then return $v(x)$

else if $d = 0$ then return $e(x)$

else if it is Max's move at x then

return $\max\{\text{Minimax}(\sigma(x,a)), d-1) : a \in \chi(x)\}$

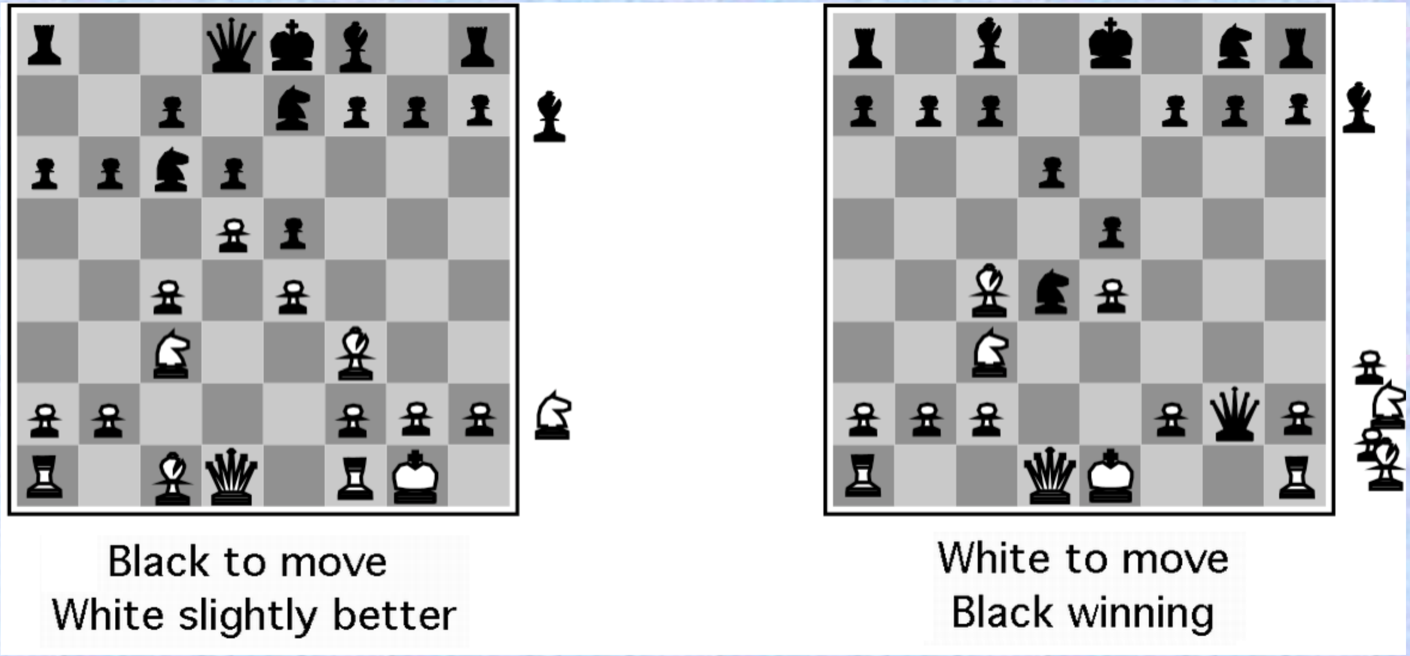
else return $\min\{\text{Minimax}(\sigma(x,a)), d-1) : a \in \chi(x)\}$

- Backward induction with an upper bound d on the search depth
  - Whenever we reach a nonterminal node of depth d, return $e(x)$
  - $e(x)$ is a **static evaluation function**: returns an estimate of $v(x)$
- If $d = \infty$, the algorithm is identical to Backward-Induction

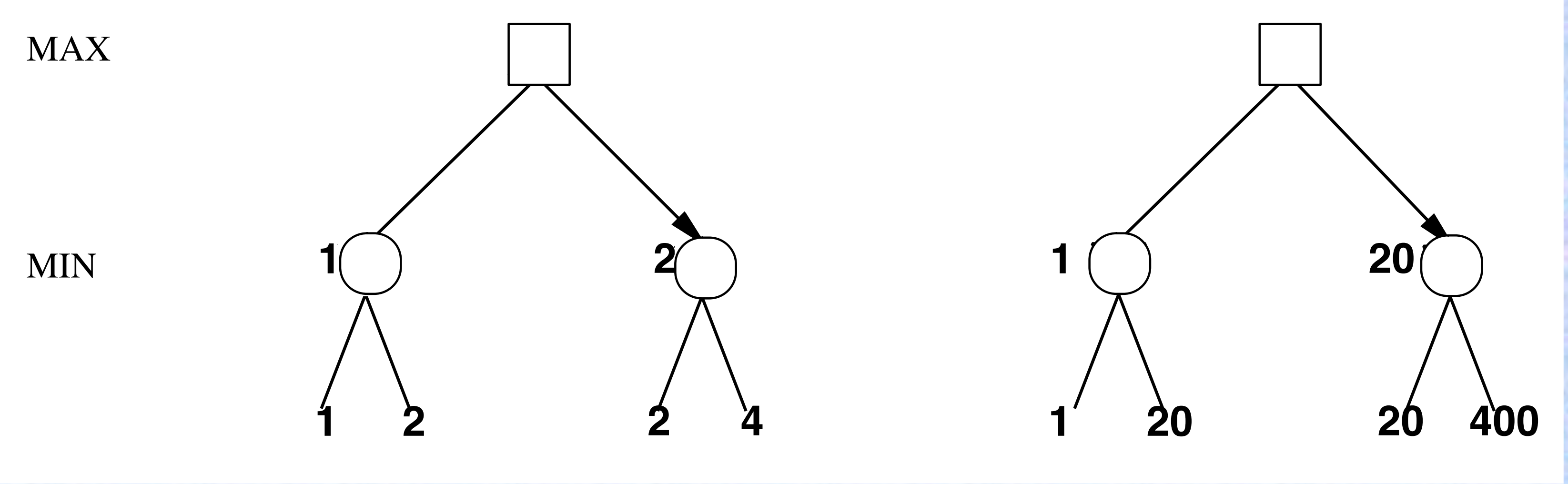
- Space complexity = $O(\min(bh,bd))$
- Time complexity = $O(\min(b^h, b^d))$

Evaluation Functions

- $e(x)$ is often a weighted sum of features
  - $e(x) = w_1 f_1(x) + w_2 f_2(x) + \ldots + w_n f_n(x)$
- E.g., in chess,
  - 1•(white pawns − black pawns) + 3•(white knights − black knights) + …



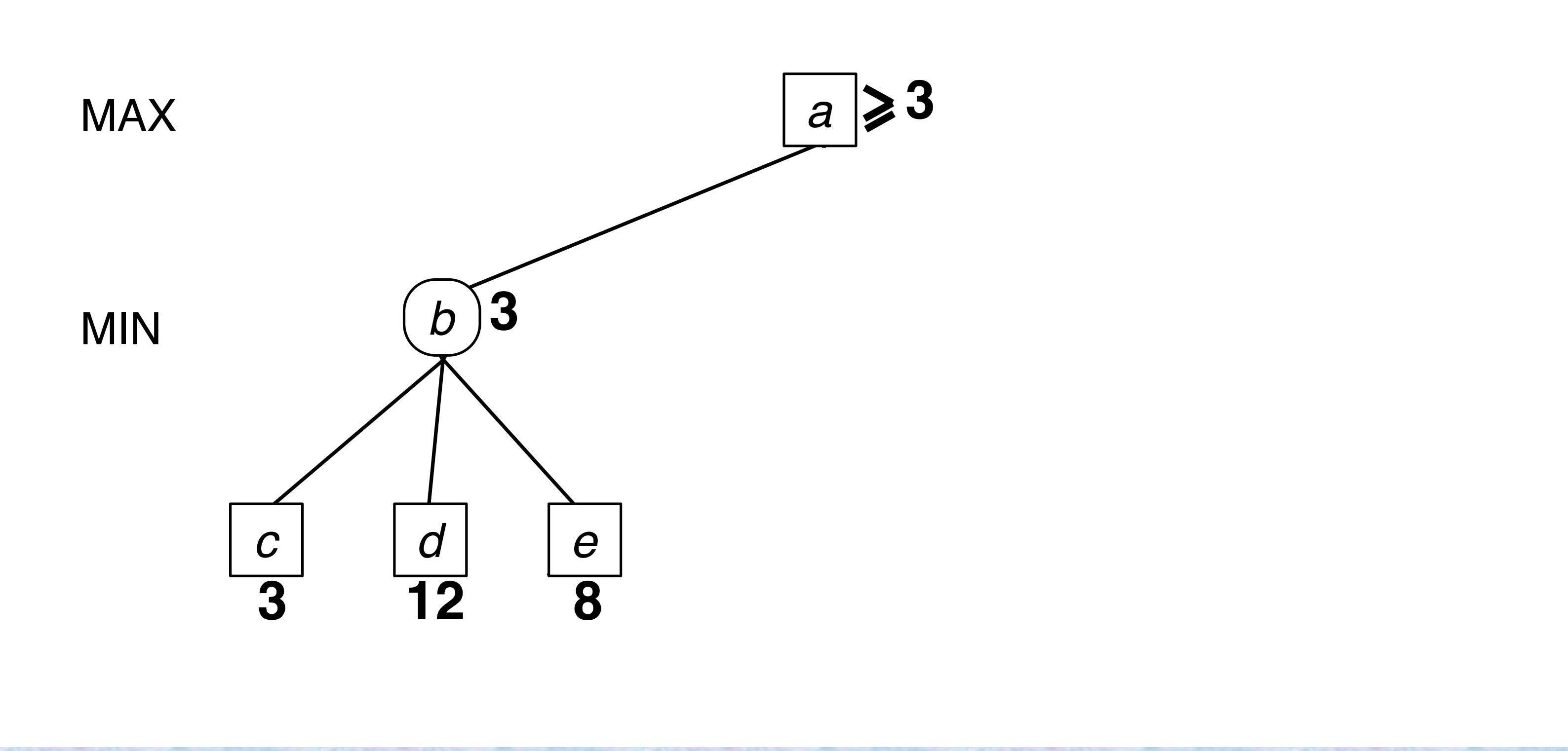
Exact Values for *e*(*x*) Don't Matter



- Behavior is preserved under any **monotonic** transformation of *e*
  - Only the order matters

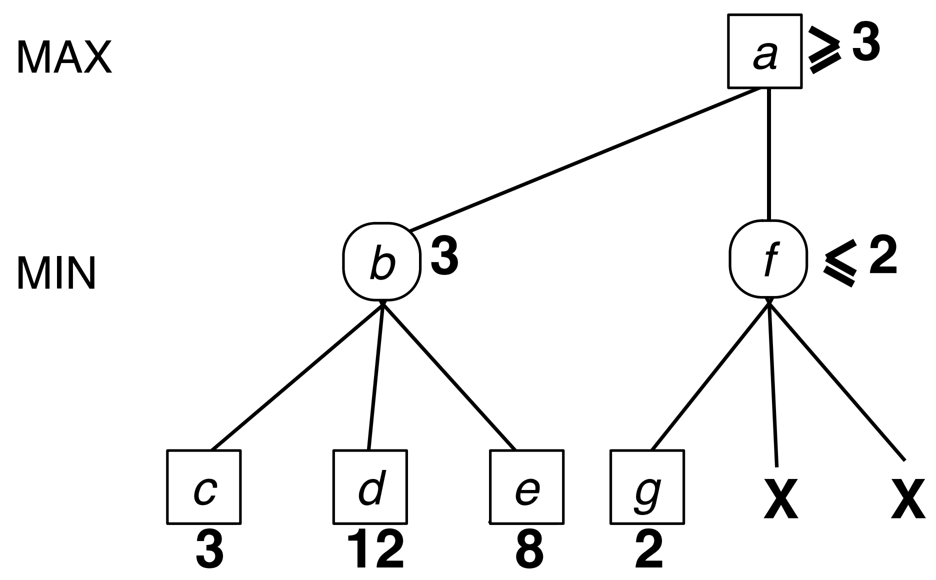
Pruning (in 2-player games)

- Let's go back to 2-player games …
  - Backward-Induction and Minimax both examine nodes that don't need to be examined



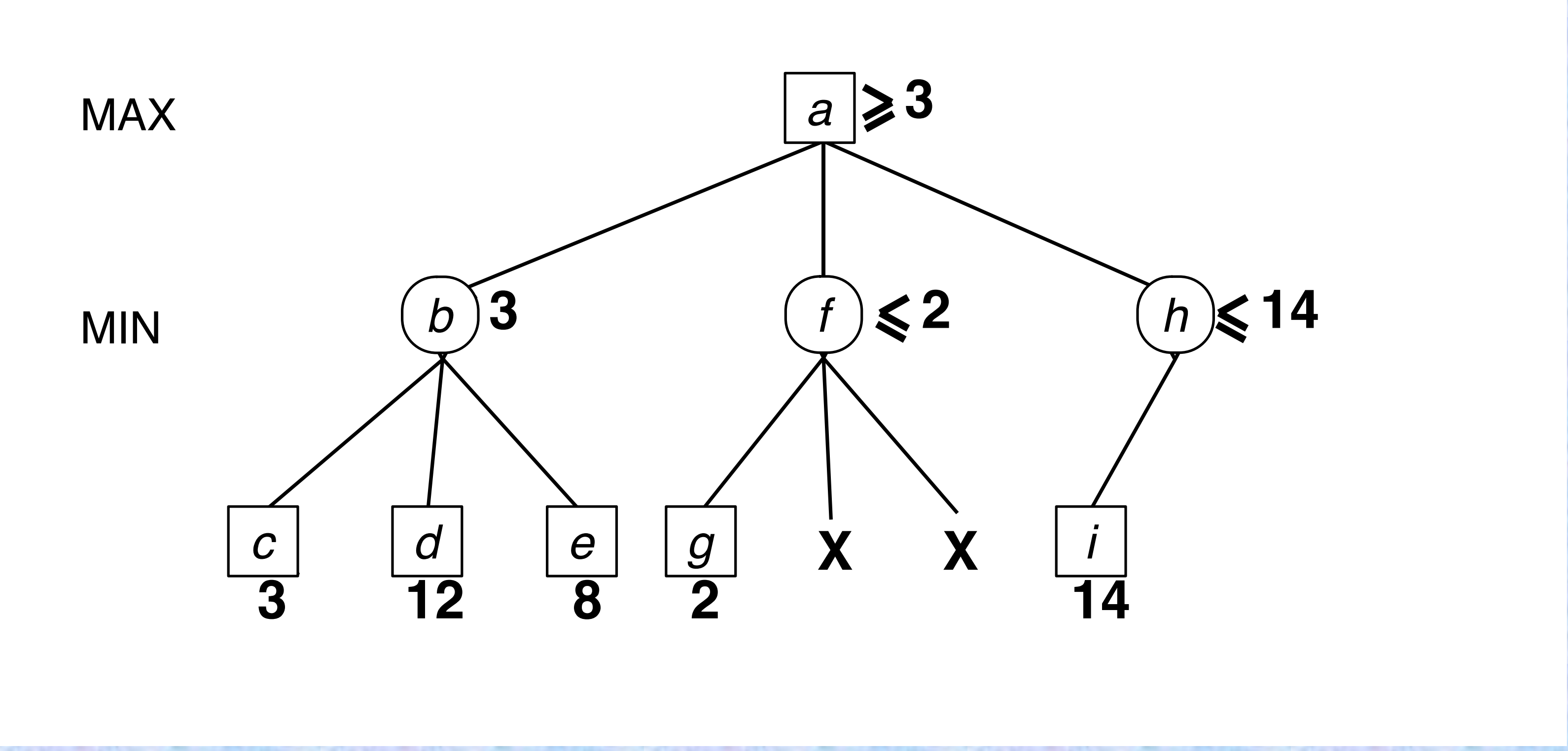
Pruning

- *b* is better for Max than *f* is
- If Max is rational then Max will never choose *f*
- So don't examine any more nodes below *f*
  - They can't affect *v*(*a*)



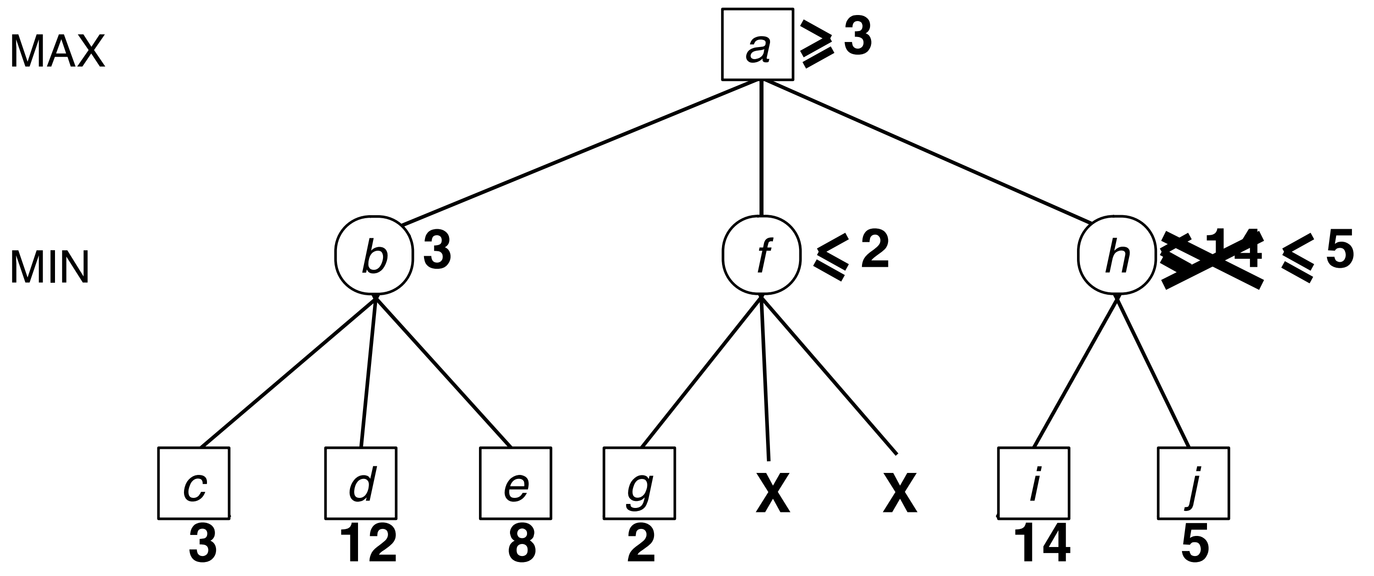
Pruning

- Don't know whether *h* is better or worse than *b*



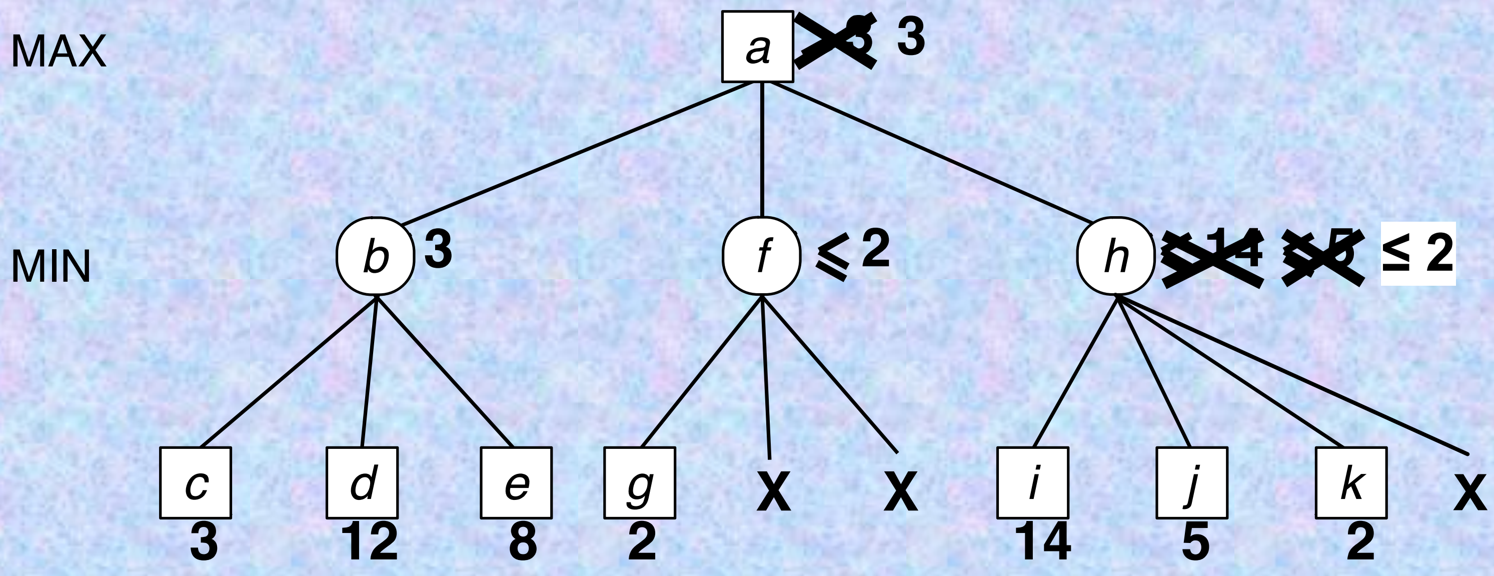
Pruning

- Still don't know whether *h* is better or worse than *b*



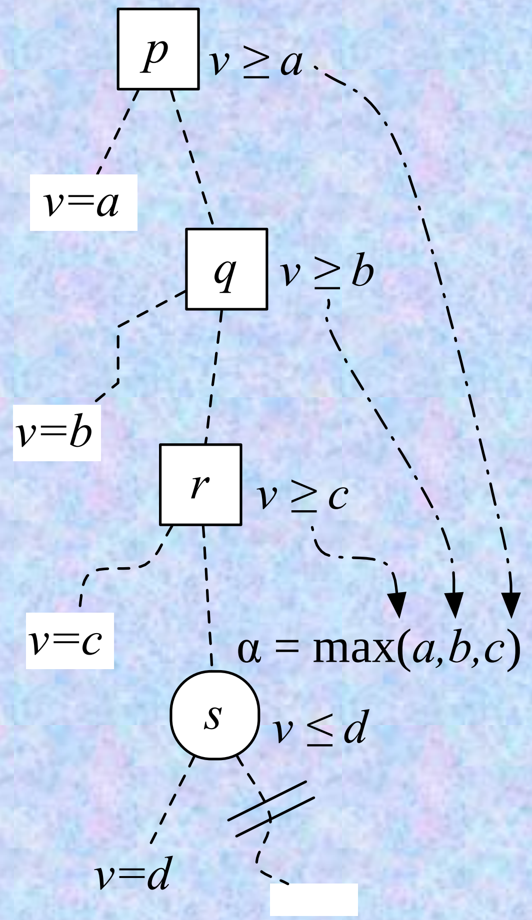
Pruning

- *h* is worse than *b*
  - Don't need to examine any more nodes below *h*
- $v(a) = 3$



Alpha Cutoff

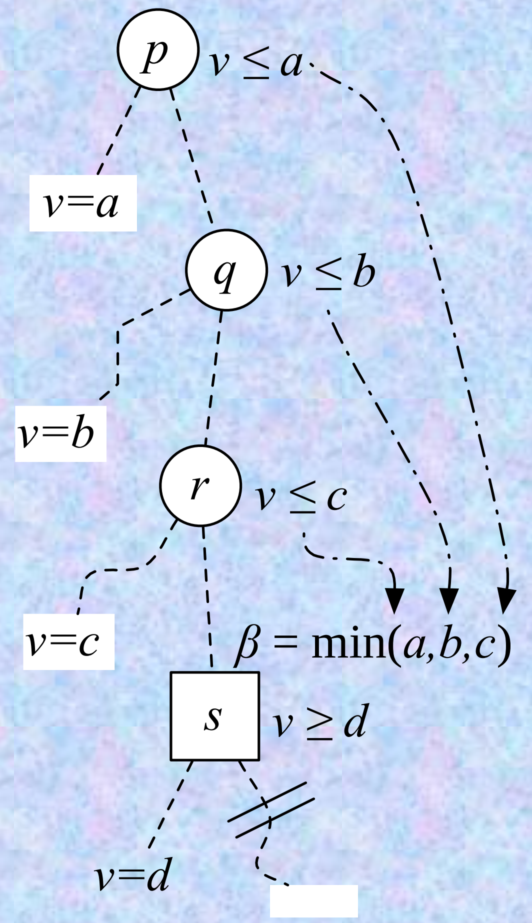
- Squares are Max nodes, circles are Min nodes
- Let $\alpha = \max(a,b,c)$, and suppose $d < \alpha$
- To reach s, the game must go through p, q, r
- By moving elsewhere at one of those nodes, Max can get $v \geq \alpha$
- If the game ever reaches node s, then Min can achieve $v \leq d <$ what Max can get elsewhere
  - Max will never let that happen
  - We don't need to know anything more about s
- What if $d = \alpha$?



Beta Cutoff

- Squares are Max nodes, circles are Min nodes
- Let $\beta = \min(a,b,c)$, and suppose $d > \beta$
- To reach *s*, the game must go through *p*, *q*, *r*
- By moving elsewhere at one of those nodes, Min can achieve $v \leq \beta$
- If the game ever reaches node *s*, then Max can achieve $v \geq d >$ what Min can get elsewhere
  - Min will never let that happen
  - We don't need to know anything more about *s*

- What if $d = \beta$?



Alpha-Beta Pruning

function Alpha-Beta(x, d, α, β)

  if x is terminal then return $u_{\text{Max}}(x)$

  else if $d = 0$ then return $e(x)$

  else do everything in the 2nd column

    return v

if it is Max's move at x then

  $v \leftarrow -\infty$

  for every child y of x do

    $v \leftarrow \max(v,$ Alpha-Beta $(y, d-1, \alpha, \beta))$

    if $v \geq \beta$ then return v

    else $\alpha \leftarrow \max(\alpha, v)$

else

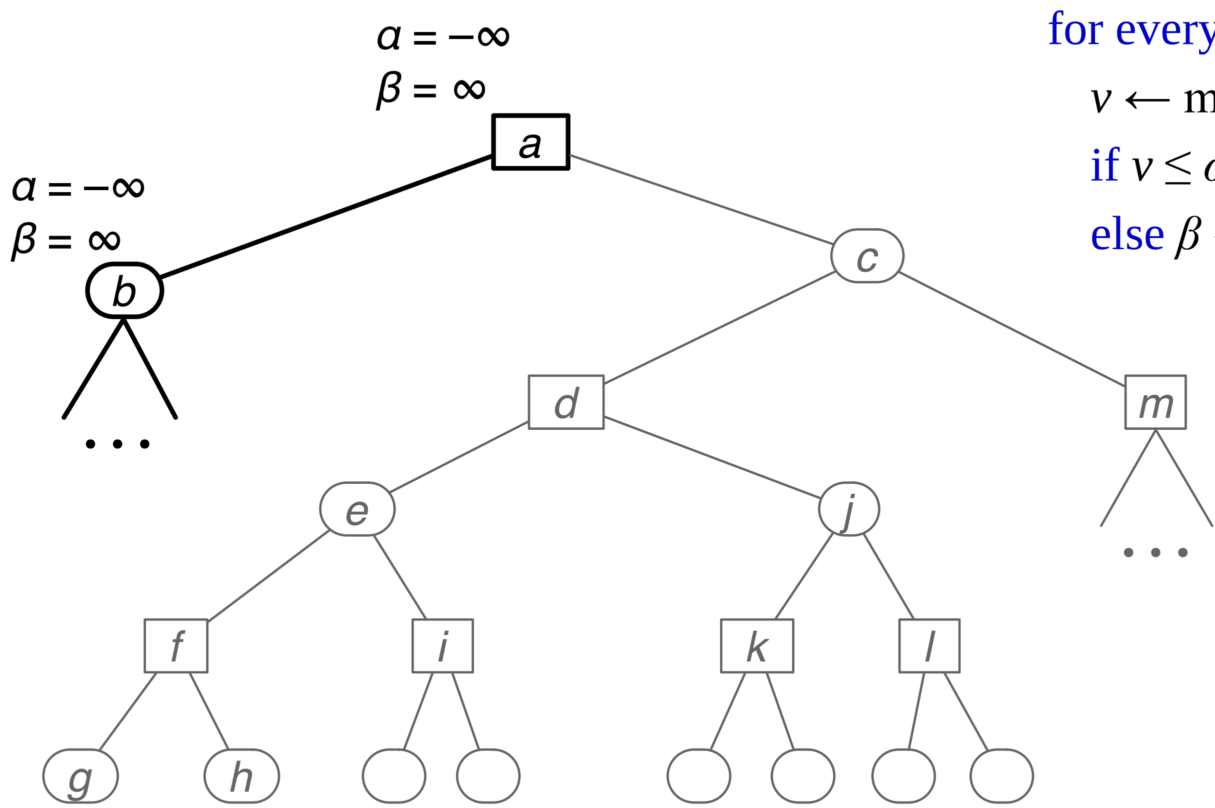
  $v \leftarrow \infty$

  for every child y of x do

    $v \leftarrow \min(v,$ Alpha-Beta $(y, d-1, \alpha, \beta))$

    if $v \leq \alpha$ then return v

    else $\beta \leftarrow \min(\beta, v)$



Alpha-Beta Pruning

function Alpha-Beta(x, d, α, β)

if x is terminal then return $u_{\text{Max}}(x)$

else if $d = 0$ then return $e(x)$

else do everything in the 2nd column

return v

if it is Max's move at x then

$v \leftarrow -\infty$

for every child y of x do

$v \leftarrow \max(v,$ Alpha-Beta $(y, d-1, \alpha, \beta))$

if $v \geq \beta$ then return v

else $\alpha \leftarrow \max(\alpha, v)$

else

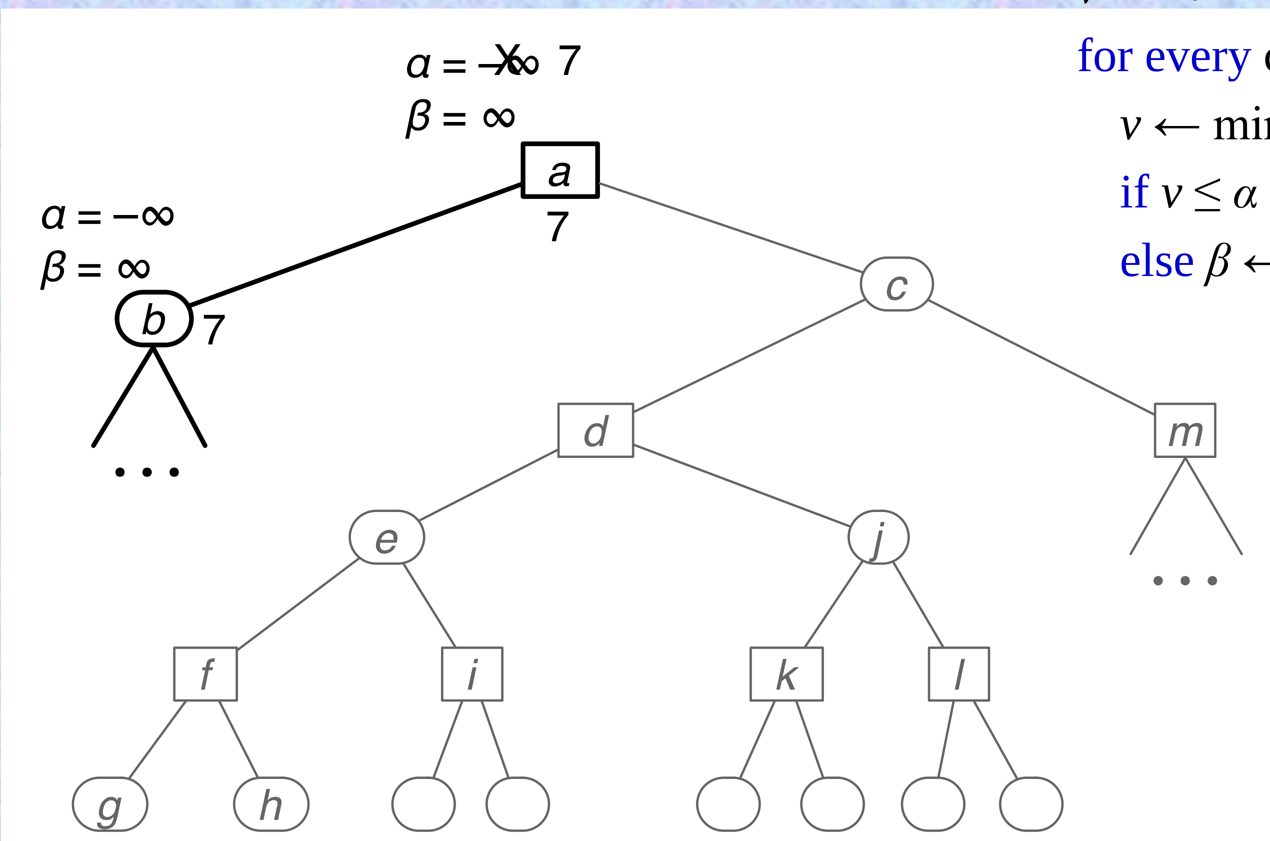
$v \leftarrow \infty$

for every child y of x do

$v \leftarrow \min(v,$ Alpha-Beta $(y, d-1, \alpha, \beta))$

if $v \leq \alpha$ then return v

else $\beta \leftarrow \min(\beta, v)$



Alpha-Beta Pruning

function Alpha-Beta(x, d, α, β)

  if x is terminal **then return** $u_{\text{Max}}(x)$

  else if $d = 0$ **then return** $e(x)$

  else do everything in the 2nd column

    return v

if it is Max's move at x **then**

  $v \leftarrow -\infty$

  for every child y of x **do**

    $v \leftarrow \max(v,$ Alpha-Beta $(y, d-1, \alpha, \beta))$

    if $v \geq \beta$ **then return** v

    else $\alpha \leftarrow \max(\alpha, v)$

else

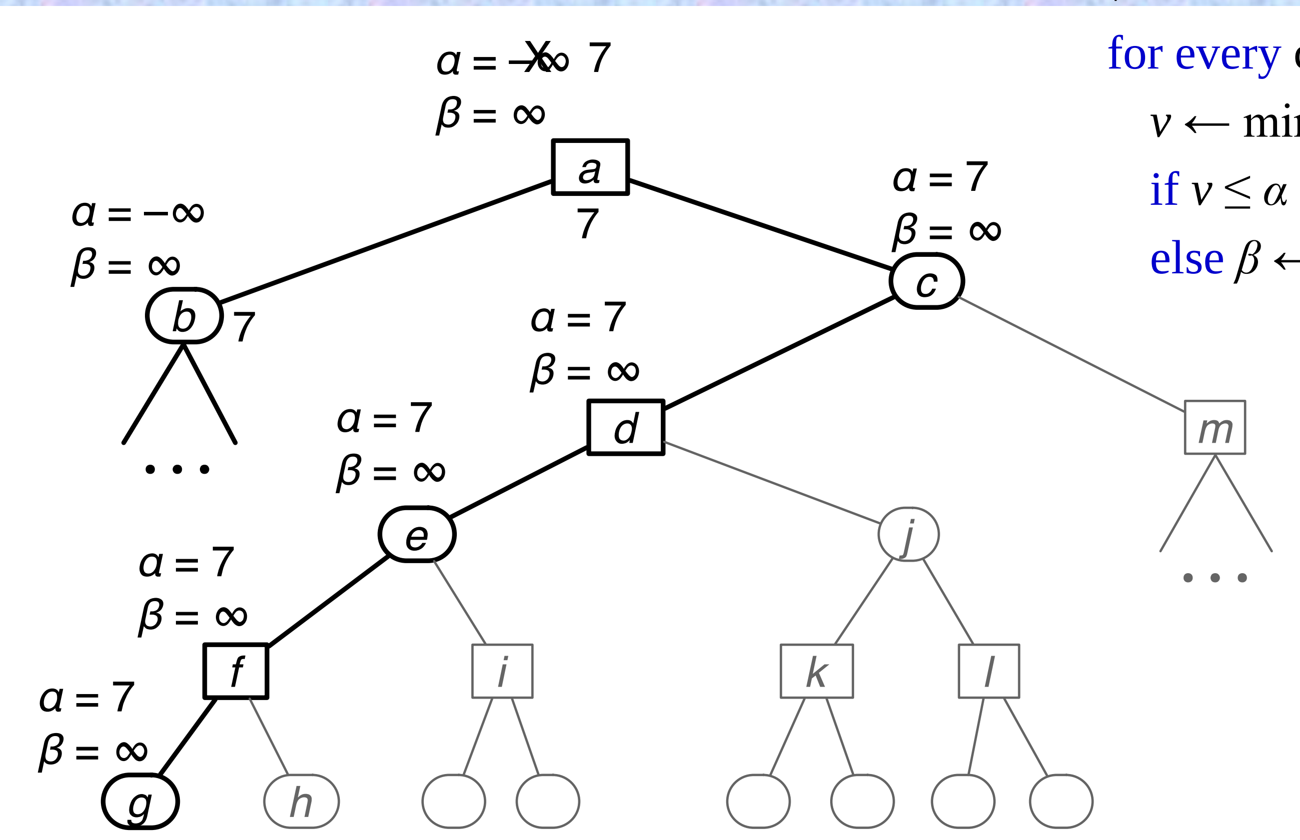
  $v \leftarrow \infty$

  for every child y of x **do**

    $v \leftarrow \min(v,$ Alpha-Beta $(y, d-1, \alpha, \beta))$

    if $v \leq \alpha$ **then return** v

    else $\beta \leftarrow \min(\beta, v)$



Alpha-Beta Pruning

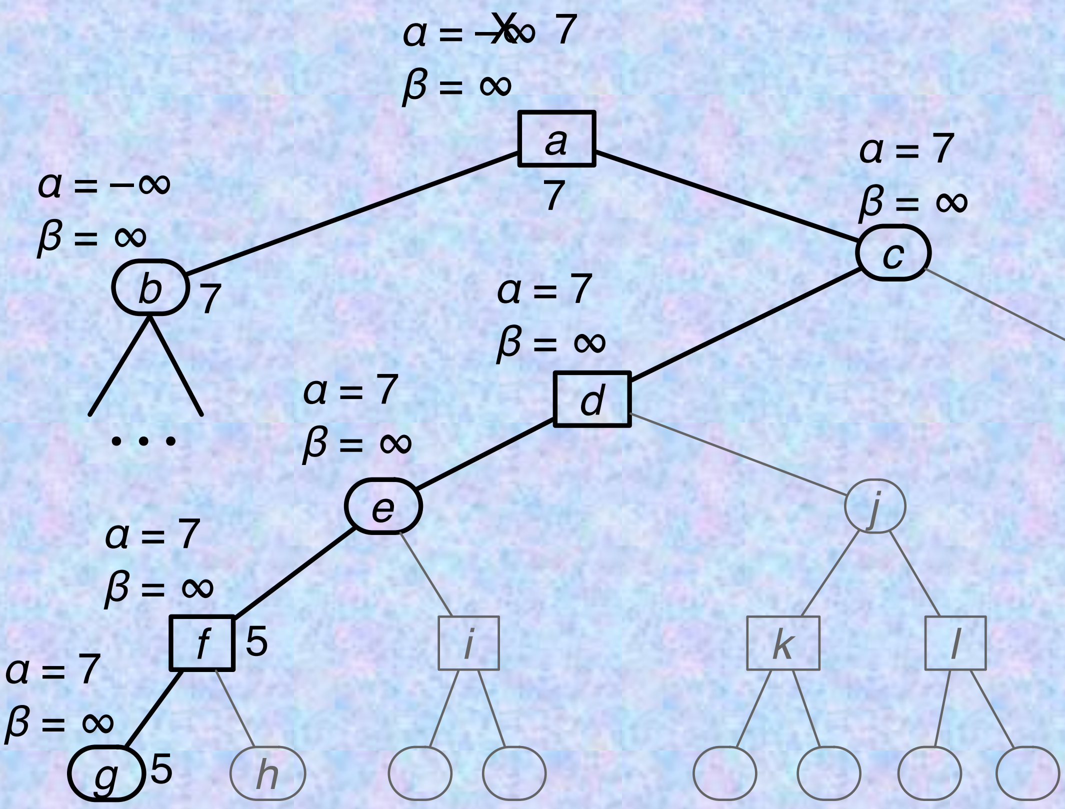
function Alpha-Beta(x, d, α, β)

  if x is terminal then return $u_{\text{Max}}(x)$

  else if $d = 0$ then return $e(x)$

  else do everything in the 2nd column

    return v



if it is Max's move at x then

  $v \leftarrow -\infty$

  for every child y of x do

    $v \leftarrow \max(v,$ Alpha-Beta $(y, d-1, \alpha, \beta))$

    if $v \geq \beta$ then return v

    else $\alpha \leftarrow \max(\alpha, v)$

else

  $v \leftarrow \infty$

  for every child y of x do

    $v \leftarrow \min(v,$ Alpha-Beta $(y, d-1, \alpha, \beta))$

    if $v \leq \alpha$ then return v

    else $\beta \leftarrow \min(\beta, v)$

Alpha-Beta Pruning

function Alpha-Beta(x, d, α, β)

if x is terminal then return $u_{\text{Max}}(x)$

else if $d = 0$ then return $e(x)$

else do everything in the 2nd column

return v

if it is Max's move at x then

$v \leftarrow -\infty$

for every child y of x do

$v \leftarrow \max(v,$ Alpha-Beta $(y, d-1, \alpha, \beta))$

if $v \geq \beta$ then return v

else $\alpha \leftarrow \max(\alpha, v)$

else

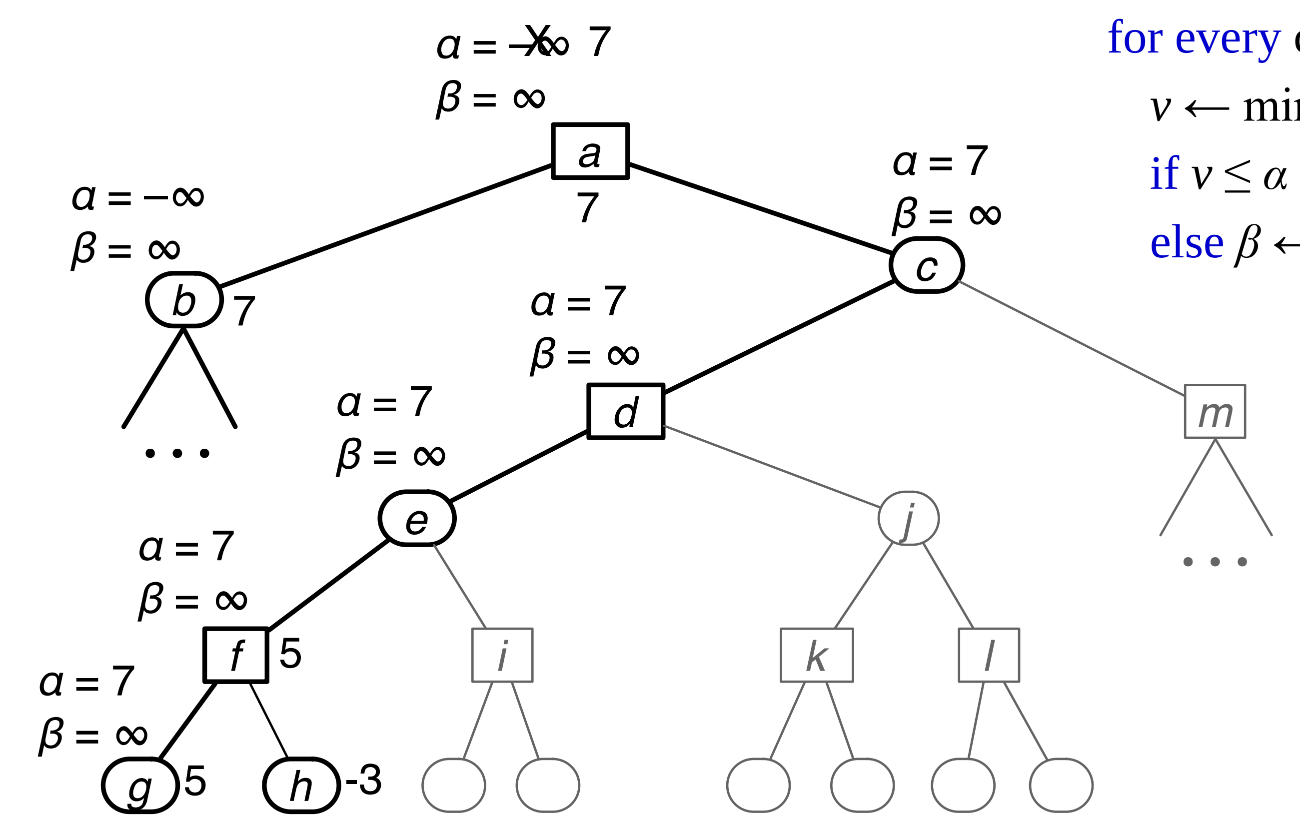
$v \leftarrow \infty$

for every child y of x do

$v \leftarrow \min(v,$ Alpha-Beta $(y, d-1, \alpha, \beta))$

if $v \leq \alpha$ then return v

else $\beta \leftarrow \min(\beta, v)$



Alpha-Beta Pruning

function Alpha-Beta(x, d, α, β)

  if x is terminal then return $u_{\text{Max}}(x)$

  else if $d = 0$ then return $e(x)$

  else do everything in the 2nd column

    return v

if it is Max's move at x then

  $v \leftarrow -\infty$

  for every child y of x do

    $v \leftarrow \max(v,$ Alpha-Beta $(y, d-1, \alpha, \beta))$

    if $v \geq \beta$ then return v

    else $\alpha \leftarrow \max(\alpha, v)$

else

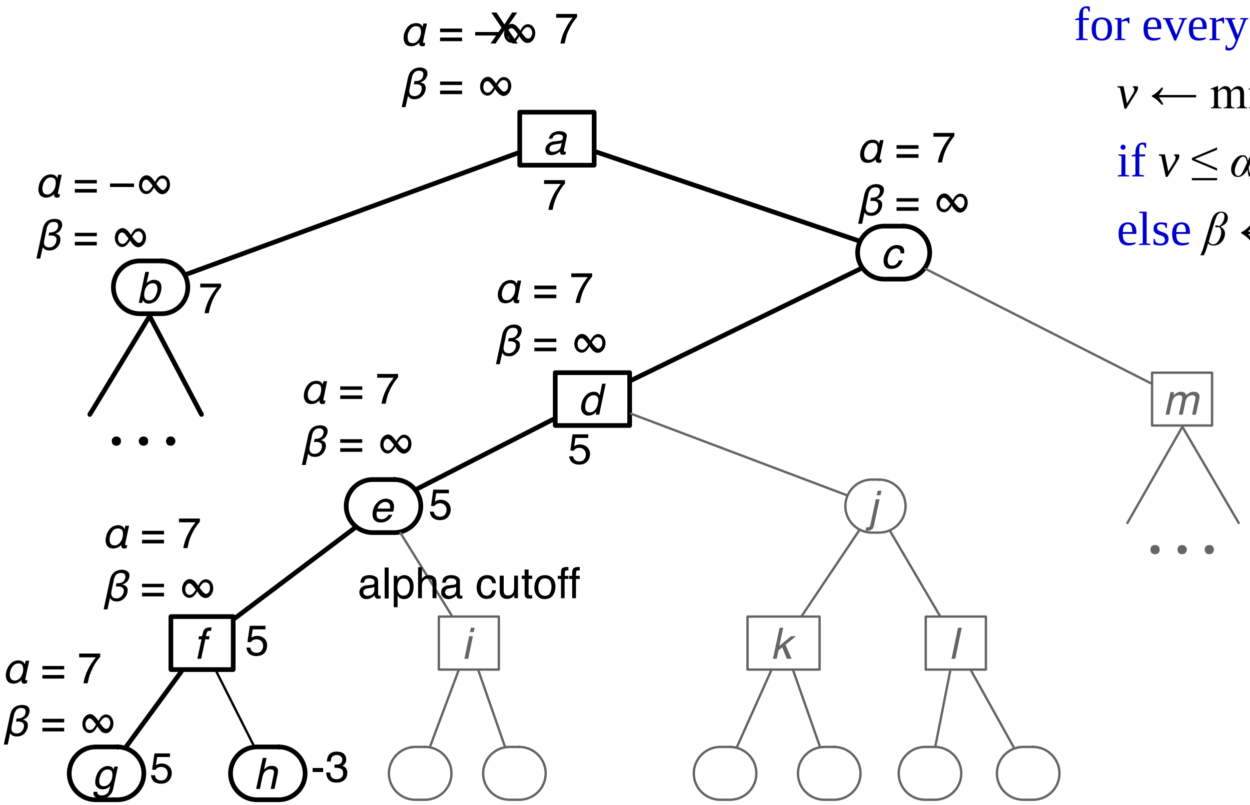
  $v \leftarrow \infty$

  for every child y of x do

    $v \leftarrow \min(v,$ Alpha-Beta $(y, d-1, \alpha, \beta))$

    if $v \leq \alpha$ then return v

    else $\beta \leftarrow \min(\beta, v)$



Alpha-Beta Pruning

function Alpha-Beta(x, d, α, β)

  if x is terminal then return $u_{\text{Max}}(x)$

  else if $d = 0$ then return $e(x)$

  else do everything in the 2nd column

    return v

if it is Max's move at x then

  $v \leftarrow -\infty$

  for every child y of x do

    $v \leftarrow \max(v,$ Alpha-Beta $(y, d-1, \alpha, \beta))$

    if $v \geq \beta$ then return v

    else $\alpha \leftarrow \max(\alpha, v)$

else

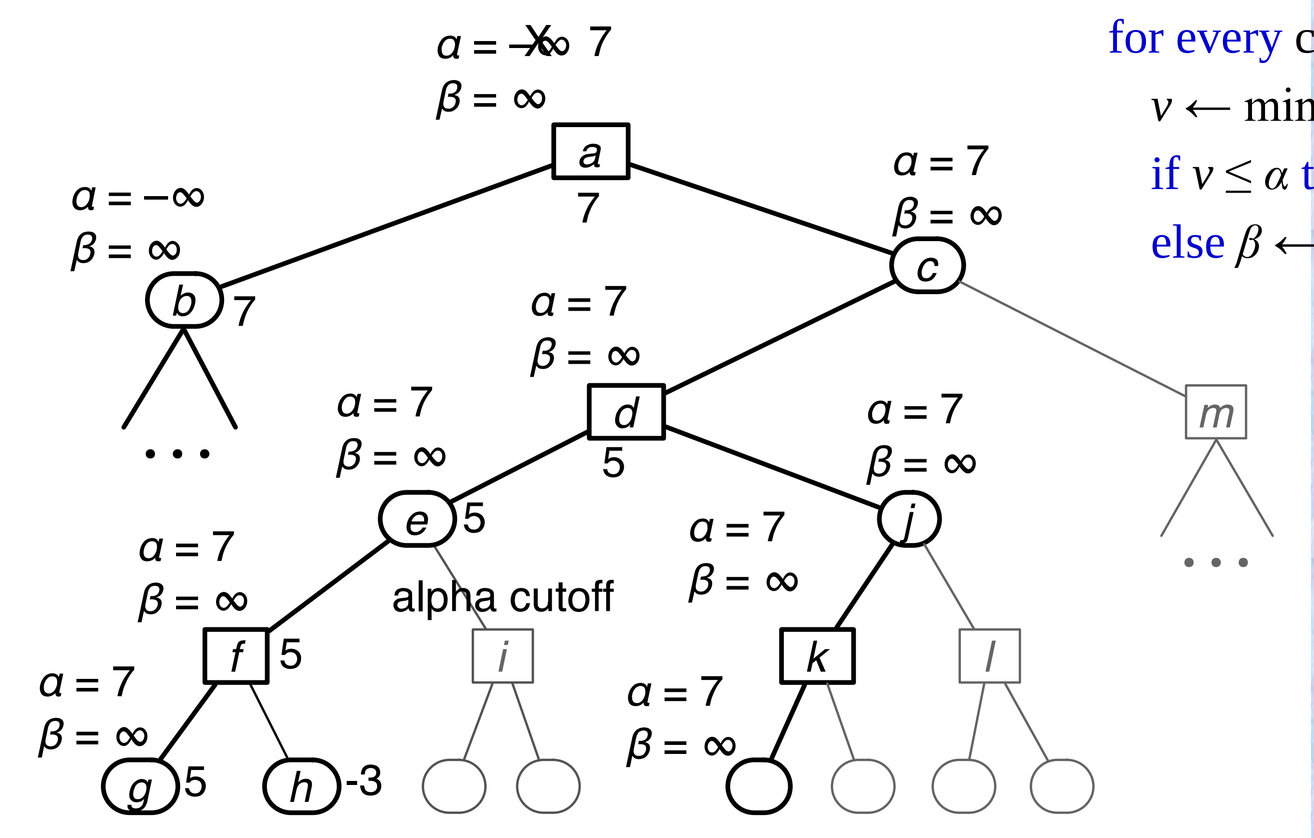
  $v \leftarrow \infty$

  for every child y of x do

    $v \leftarrow \min(v,$ Alpha-Beta $(y, d-1, \alpha, \beta))$

    if $v \leq \alpha$ then return v

    else $\beta \leftarrow \min(\beta, v)$



Alpha-Beta Pruning

function Alpha-Beta(x, d, α, β)

  if x is terminal then return $u_{Max}(x)$

  else if $d = 0$ then return $e(x)$

  else do everything in the 2nd column

    return v

if it is Max's move at x then

  $v \leftarrow -\infty$

  for every child y of x do

    $v \leftarrow \max(v,$ Alpha-Beta $(y, d-1, \alpha, \beta))$

    if $v \geq \beta$ then return v

    else $\alpha \leftarrow \max(\alpha, v)$

else

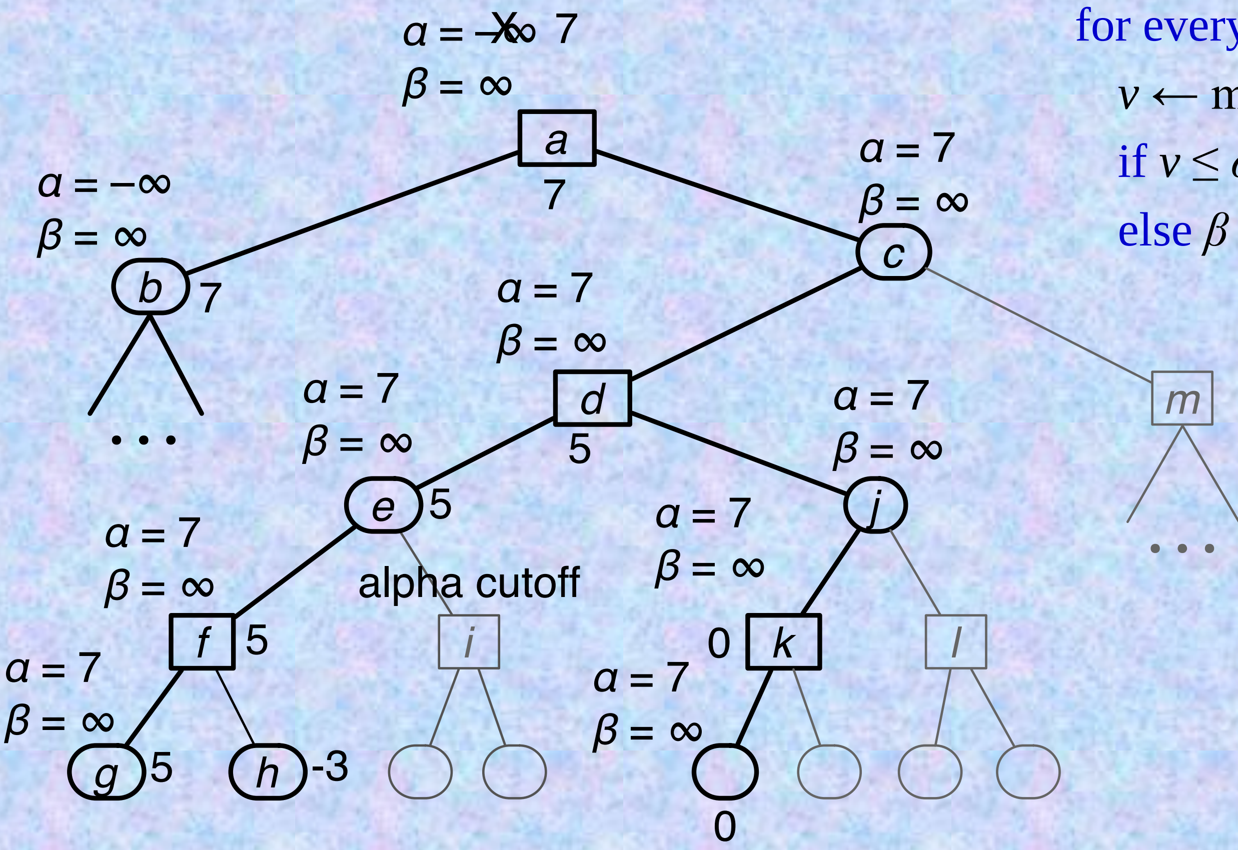
  $v \leftarrow \infty$

  for every child y of x do

    $v \leftarrow \min(v,$ Alpha-Beta $(y, d-1, \alpha, \beta))$

    if $v \leq \alpha$ then return v

    else $\beta \leftarrow \min(\beta, v)$



Alpha-Beta Pruning

function Alpha-Beta(x, d, α, β)

  if x is terminal **then return** $u_{\text{Max}}(x)$

  else if $d = 0$ **then return** $e(x)$

  else do everything in the 2nd column

    return v

if it is Max's move at x **then**

  $v \leftarrow -\infty$

  for every child y of x **do**

    $v \leftarrow \max(v,$ Alpha-Beta $(y, d-1, \alpha, \beta))$

    if $v \geq \beta$ **then return** v

    else $\alpha \leftarrow \max(\alpha, v)$

else

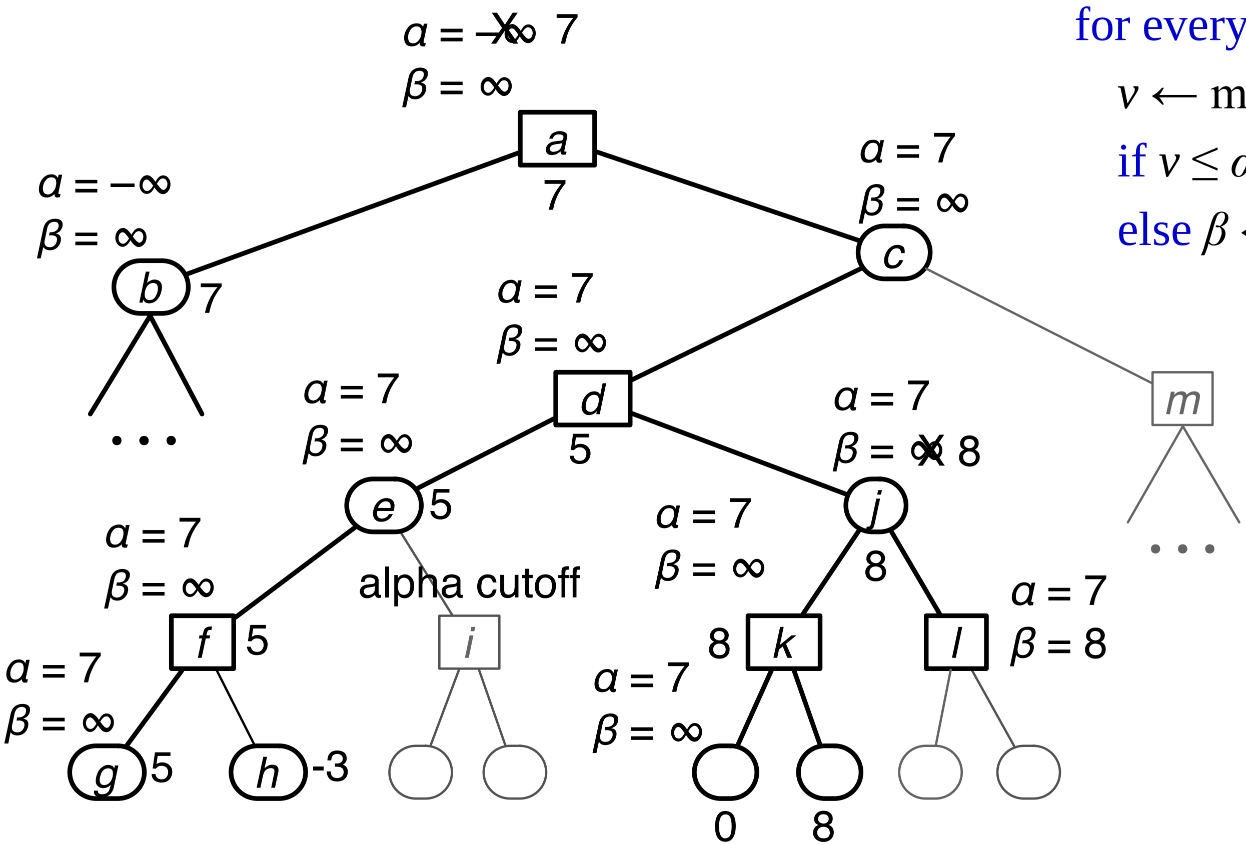
  $v \leftarrow \infty$

  for every child y of x **do**

    $v \leftarrow \min(v,$ Alpha-Beta $(y, d-1, \alpha, \beta))$

    if $v \leq \alpha$ **then return** v

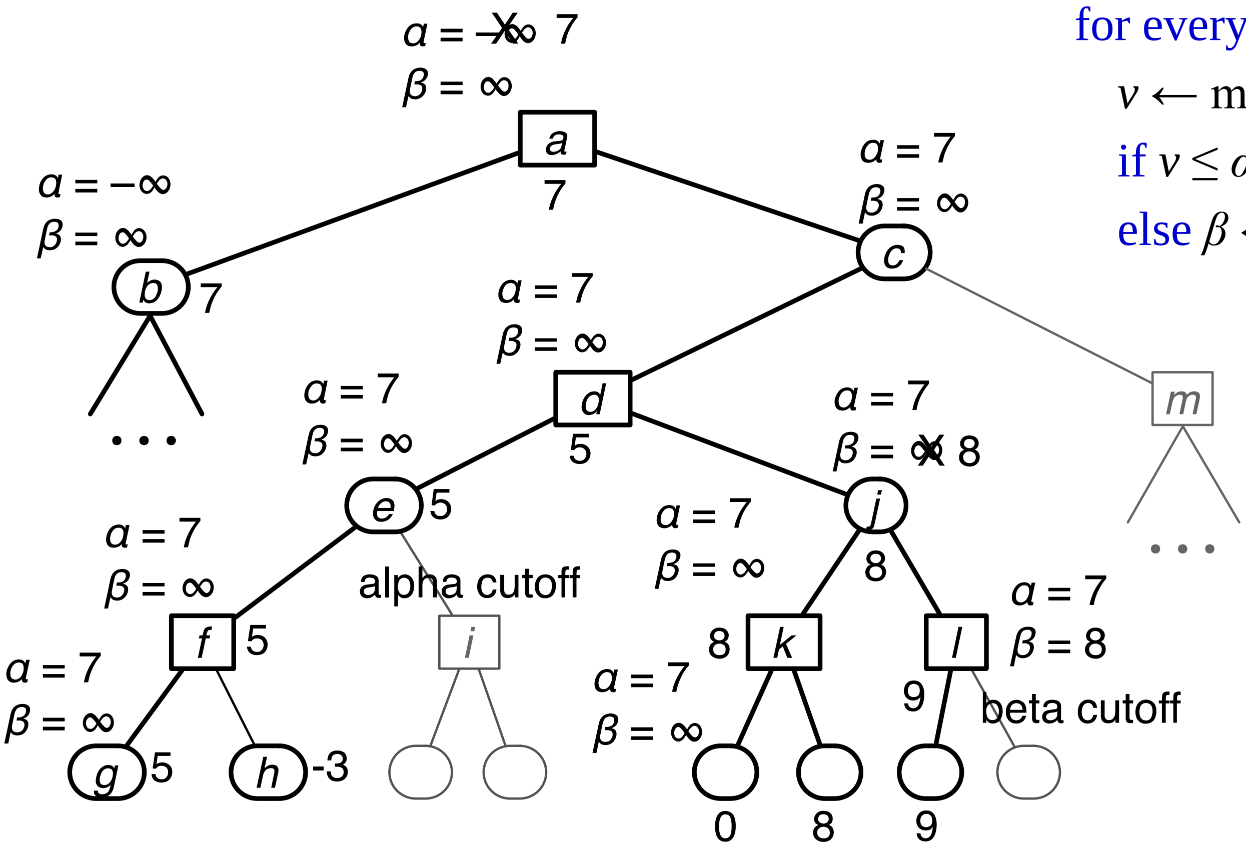
    else $\beta \leftarrow \min(\beta, v)$



Alpha-Beta Pruning

function Alpha-Beta(x, d, α, β)
  if x is terminal **then return** $u_{\mathrm{Max}}(x)$
  else if $d = 0$ **then return** $e(x)$
  else do everything in the 2nd column
    return v

if it is Max's move at x **then**
  $v \leftarrow -\infty$
  for every child y of x **do**
    $v \leftarrow \max(v,$ Alpha-Beta $(y, d-1, \alpha, \beta))$
    if $v \geq \beta$ **then return** v
    else $\alpha \leftarrow \max(\alpha, v)$
else
  $v \leftarrow \infty$
  for every child y of x **do**
    $v \leftarrow \min(v,$ Alpha-Beta $(y, d-1, \alpha, \beta))$
    if $v \leq \alpha$ **then return** v
    else $\beta \leftarrow \min(\beta, v)$



Alpha-Beta Pruning

function Alpha-Beta(x, d, α, β)

if x is terminal then return $u_{\mathrm{Max}}(x)$

else if $d = 0$ then return $e(x)$

else do everything in the 2nd column

return v

if it is Max's move at x then

$v \leftarrow -\infty$

for every child y of x do

$v \leftarrow \max(v,$ Alpha-Beta $(y, d-1, \alpha, \beta))$

if $v \geq \beta$ then return v

else $\alpha \leftarrow \max(\alpha, v)$

else

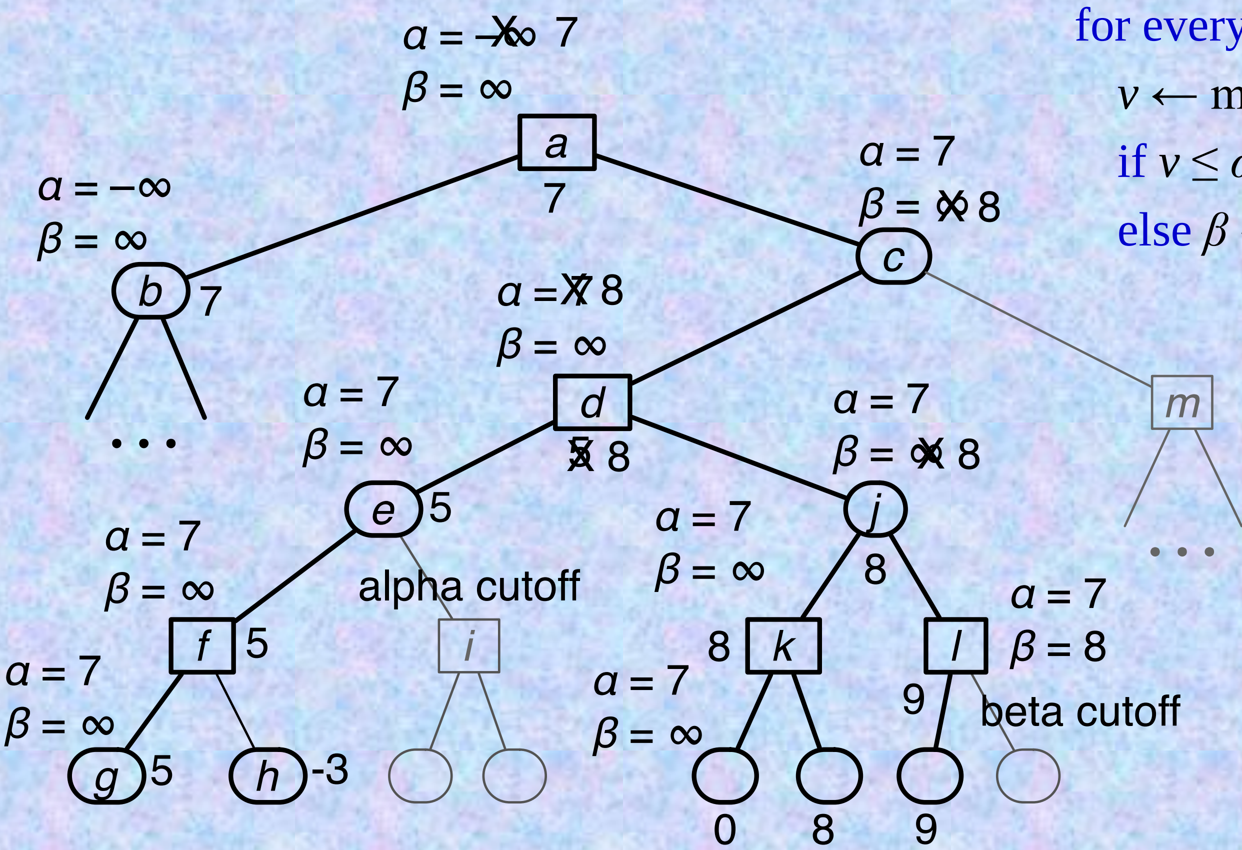
$v \leftarrow \infty$

for every child y of x do

$v \leftarrow \min(v,$ Alpha-Beta $(y, d-1, \alpha, \beta))$

if $v \leq \alpha$ then return v

else $\beta \leftarrow \min(\beta, v)$



Alpha-Beta Pruning

function Alpha-Beta(x, d, α, β)

  if x is terminal then return $u_{\text{Max}}(x)$

  else if $d = 0$ then return $e(x)$

  else do everything in the 2nd column

    return v

if it is Max's move at x then

  $v \leftarrow -\infty$

  for every child y of x do

    $v \leftarrow \max(v,$ Alpha-Beta $(y, d-1, \alpha, \beta))$

    if $v \geq \beta$ then return v

    else $\alpha \leftarrow \max(\alpha, v)$

else

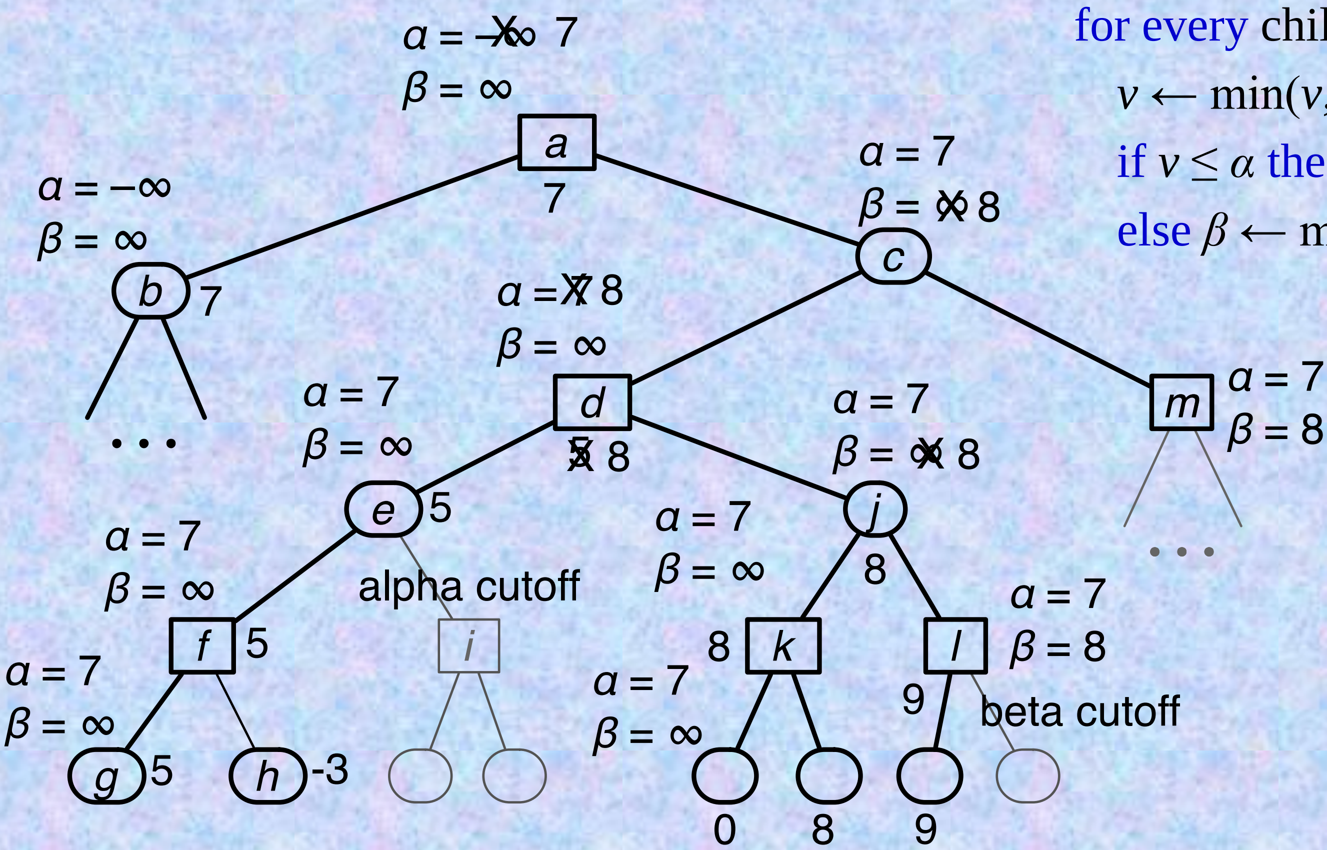
  $v \leftarrow \infty$

  for every child y of x do

    $v \leftarrow \min(v,$ Alpha-Beta $(y, d-1, \alpha, \beta))$

    if $v \leq \alpha$ then return v

    else $\beta \leftarrow \min(\beta, v)$



Properties of Alpha-Beta

- Alpha-beta pruning reasons about which computations are relevant
  - A form of **metareasoning**

Theorem:

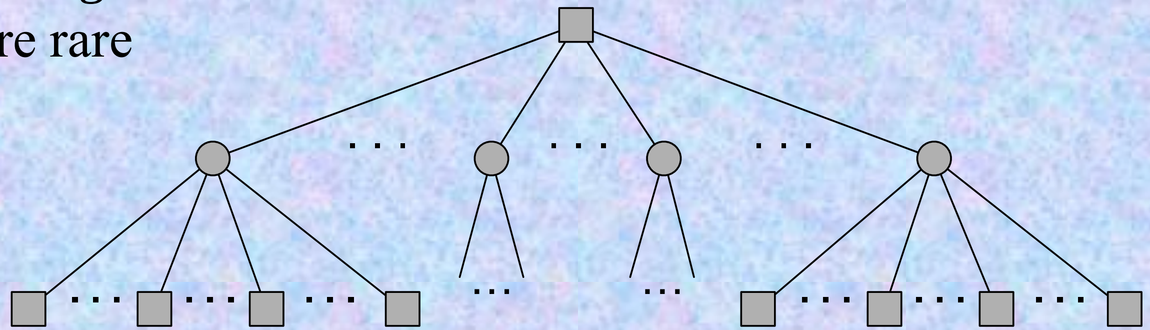
- If the value returned by Minimax(x, d) is in $[\alpha,\beta]$
  - then Alpha-Beta(x, d, α, β) returns the same value
- If the value returned by Minimax(x, d) is $\leq \alpha$
  - then Alpha-Beta(x, d, α, β) returns a value $\leq \alpha$
- If the value returned by Minimax(x, d) is $\geq \beta$
  - then Alpha-Beta(x, d, α, β) returns a value $\geq \beta$

Corollary:

- Alpha-Beta($x, d, -\infty, \infty$) returns the same value as Minimax(x, d)
- Alpha-Beta($x, \infty, -\infty, \infty$) returns $v(x)$

Node Ordering

- Deeper lookahead (larger d) usually gives better decisions
  - There are "pathological" games where it doesn't, but those are rare
- Compared to Minimax, how much farther ahead can Alpha-Beta look?



- Best case:
  - children of Max nodes are searched in greatest-value-first order, children of Min nodes are searched in least-value-first order
  - Alpha-Beta's time complexity is $O(b^{d/2})$ $\Rightarrow$ doubles the solvable depth
- Worst case:
  - children of Max nodes are searched in least-value first order, children of Min nodes are searched in greatest-value first order
  - Like Minimax, Alpha-Beta visits all nodes of depth $\leq d$: time complexity $O(b^d)$

Node Ordering

- How to get closer to the best case:
  - Every time you expand a state s, apply *e* to its children
  - When it's Max's move, sort the children in order of largest *e* first
  - When it's Min's move, sort the children in order of smallest *e* first

- Suppose we have 100 seconds, explore 10^4 nodes/second
  - 10^6 nodes per move
  - Put this into the form $b^{d/2} \approx 35^{8/2}$
  - Best case Alpha-Beta reaches depth 8 ⇒ pretty good chess program

Other Modifications

- Several other modifications that can improve the accuracy or computation time (*but not covered in this class*):
  - quiescence search and biasing
  - transposition tables
  - thinking on the opponent's time
  - table lookup of "book moves"
  - iterative deepening

Game-Tree Search in Practice

- **Checkers**: In 1994, Chinook ended 40-year-reign of human world champion Marion Tinsley
  - Tinsley withdrew for health reasons, died a few months later
- In 2007, Checkers was solved: with perfect play, it's a draw
  This took 10^{14} calculations over 18 years. Search space size 5×10^{20}
- **Chess**: In 1997, Deep Blue defeated Gary Kasparov in a six-game match
  - Deep Blue searches 200 million positions per second
  - Uses very sophisticated evaluation, and undisclosed methods for extending some lines of search up to 40 ply
- **Othello**: human champions don't compete against computers
  - The computers are too good
- **Go**: in 2006, good amateurs could beat the best go programs
    - Even with a 9-stone handicap
  - Go programs have improved a *lot* during the past 5 years

Summary

- Two-player zero-sum perfect-information games
  - the maximin and minimax strategies are the same
  - only need to look at pure strategies
  - can do a game-tree search
    - minimax values, alpha-beta pruning
- In sufficiently complicated games, perfection is unattainable
  - limited search depth, static evaluation function
  - Monte Carlo roll-outs
- Game-tree search can be modified for games in which there are stochastic outcomes